Integrating Machine Learning Into Process-Based Modeling to Predict Ammonia Losses From Stored Liquid Dairy Manure

Rana A. Genedy

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Biological Systems Engineering

Jactone A. Ogejo, Chair
Matthias Chung
Ryan S. Senger
Julie E. Shortridge

April 25, 2023
Blacksburg, Virginia

Keywords: Physics-informed neural networks, Machine and deep learning, Dairy manure, Ammonia emissions, Manure management.

Copyright 2023, Rana A. Genedy
Integrating Machine Learning Into Process-Based Modeling to Predict Ammonia Losses From Stored Liquid Dairy Manure

Rana A. Genedy

(ABSTRACT)

Storing manure on dairy farms is essential for maximizing its fertilizer value, reducing management costs, and minimizing potential environmental pollution challenges. However, ammonia loss through volatilization during storage remains a challenge. Quantifying these losses is necessary to inform decision-making processes, improve manure management, and design ammonia mitigation strategies. In 2003, the National Research Council recommended using process-based models to estimate emissions of pollutants, such as ammonia, from animal feeding operations. While much progress has been made to meet this call, still, their accuracy is limited because of the inadequate values of manure properties such as heat and mass transfer coefficients. Additionally, the process-based models lack realistic estimations for manure temperatures; they use ambient air temperature surrogates, which was found to underestimate the atmospheric emissions during storage. This study uses machine learning algorithms’ unique abilities to address some of the challenges of process-based modeling. Firstly, ammonia concentrations, manure temperature, and local meteorological factors were measured from three dairy farms with different manure management practices and storage types. This data was used to estimate the influence of manure characteristics and meteorological factors on the trend of ammonia emissions. Secondly, the data was subjected to four data-driven machine learning algorithms and a physics-informed neural network (PINN)
to predict manure temperature. Finally, a deep-learning approach that combines process-based modeling and recurrent neural networks (LSTM) was introduced to estimate ammonia loss from dairy manure during storage. This method involves inverse problem-solving to estimate the heat and mass transfer coefficients for ammonia transport and emission from stored manure using the hyperparameters optimization tool, Optuna. Results show that ammonia flux patterns mirrored manure temperature closely compared to ambient air temperature, with wind speed and crust thickness significantly influencing ammonia emissions. The data-driven machine learning models used to estimate the ammonia emissions had a high predictive ability; however, their generalization accuracy was poor. However, the PINN model had superior generalization accuracy with $R^2$ during the testing phase exceeded 0.70, in contrast to -0.03 and 0.66 for finite-elements heat transfer and data-driven neural network, respectively. In addition, optimizing the process-based model parameters has significantly improved performance. Finally, Physics-informed LSTM has the potential to replace conventional process-based models due to its computational efficiency and does not require extensive data collection. The outcomes of this study contribute to precision agriculture, specifically designing suitable on-farm strategies to minimize nutrient loss and greenhouse gas emissions during the manure storage periods.
Integrating Machine Learning Into Process-Based Modeling to Predict Ammonia Losses From Stored Liquid Dairy Manure

Rana A. Genedy

(GENERAL AUDIENCE ABSTRACT)

Dairy farming is critical for meeting the global demand for animal protein products; however, it generates a lot of manure that must be appropriately managed. Manure can only be applied to crop or pasture lands during growing seasons. Typically, manure is stored on farms until time permits for land application. During storage, microbial processes occur in the manure, releasing gases such as ammonia. Ammonia emitted contributes to the degradation of ambient air quality, human and animal health problems, biodiversity loss, and soil health deterioration. Furthermore, releasing ammonia from stored manure reduces the nitrogen fertilizer value of stored manure. Implementing control measures to mitigate ammonia emission is necessary to reduce nitrogen loss from stored manure. Deciding and applying appropriate control measures require knowledge of the rate of ammonia emission and when it occurs. Process-based models are a less expensive and more reliable method for estimating ammonia emissions from stored liquid dairy manure. Process-based model is a mathematical model that simulates processes related to ammonia production and emission from stored manure. However, process-based models have limitations because they require estimates of manure properties, which vary depending on the manure management. Additionally, these models use air temperature instead of manure temperature, underestimating the ammonia lost during storage. Therefore, this study used machine learning algorithms to develop more accurate models for predicting manure temperature and estimating ammonia emissions. First,
we collected manure temperature, ammonia emissions, and weather data from three dairy farms with different manure management practices and storage structures. We used it to estimate the factors that affect ammonia emissions. The data was then used to develop four machine-learning models and one integrated machine-learning-based to assess their ability to predict manure temperature. Finally, a different machine learning approach that combines process-based modeling and neural networks was used to directly estimate ammonia loss from dairy manure during storage. The results show that manure temperature is closely related to the amount of ammonia lost, and factors like wind speed and crust thickness also influence the amount of ammonia lost. Machine learning algorithms offer a more accurate way to predict manure temperature than traditional methods. Finally, combining machine learning and process-based modeling improved the ammonia emission estimates. This study contributes to precision agriculture by designing suitable on-farm strategies to minimize nutrient loss during manure storage periods. It provides valuable information for dairy farmers and policymakers on managing manure storage more effectively and sustainably.
Acknowledgments

I am immensely grateful to my advisor, Dr. Jactone Arogo Ogejo, for his exceptional guidance and support throughout my Ph.D. journey. Dr. Arogo has always challenged me to expand my knowledge, explore new ideas, and always think about the “plus more.” His mentoring style has empowered me to become a more independent and self-directed researcher. His willingness to listen, chat and provide feedback has been invaluable to both my academic and personal growth. I am profoundly grateful to my committee members, Dr. Matthias Chung, Dr. Julie Shortridge, and Dr. Ryan Senger, for their invaluable help and guidance during my research. They have played an important role in shaping my academic and professional trajectory. I also would like to sincerely thank the faculty, staff, and students of the Department of Biological Systems Engineering at Virginia Tech for providing a nurturing and collaborative environment. I would like to express my heartfelt appreciation to my family, who have been my constant support throughout my Ph.D. journey. To my parents, Ahmed and Abeer, my siblings, Reem and Abdelrahman, my grandma, and my aunt, I am eternally grateful for their encouragement and understanding. I would also like to thank my partner, Swailem, for his unwavering guidance and support throughout my Ph.D. My journey would not have been possible without my lifetime friends, Salma, Hoda, Zakria, Reem, Nada, Salma, Mariam, Yara, Merna, Ragab, and Eisa, and my Blacksburg family, who have been my pillars of support. I am thankful for your constant help and support. Lastly, I dedicate this dissertation to my grandfather and Omar el Daghar, whose memory will always be a source of inspiration and motivation for me.


# Contents

List of Figures xi

List of Tables xvi

1 Introduction and Literature Review 1

1.1 Sustainable dairy production 1

1.2 Fate of manure nitrogen during storage 2

1.3 Quantifying ammonia lost from stored manure 4

1.4 Machine learning 7

1.5 Objective 10

1.6 Structure and Content 11

References 12

2 Quantifying ammonia lost to the atmosphere during manure storage on a dairy farm as influenced by management and meteorological parameters. 21

2.1 Introduction 23

2.2 Materials and methods 27

2.2.1 Study site description 27

2.2.2 Ammonia concentrations, weather parameters, and manure temperature 28

2.2.3 Calculating ammonia flux and ammonia lost 33

2.3 Results 34

2.3.1 Weather conditions 34

2.3.2 Ammonia flux 35
2.3.3 Influence of meteorological factors and manure characteristics on ammonia loss ......................................................... 37
2.3.4 Surface manure crusting and ammonia emissions .................. 38

2.4 Discussion ........................................................................ 42
2.4.1 Study limitations .............................................................. 45

2.5 Conclusion ......................................................................... 45

References ............................................................................. 47

3 Using machine learning techniques to predict liquid dairy manure temperature during storage. 55

3.1 Introduction ....................................................................... 58
3.2 Materials and methods .......................................................... 62
3.2.1 On-farm manure management and storage ......................... 62
3.2.2 Manure temperature measurement and weather data collection .... 63
3.2.3 Data processing and analysis ............................................ 64
3.2.4 Data visualization and statistical analysis .......................... 64
3.2.5 Machine learning for manure temperature predictions ............ 65
3.2.6 Assessing the model performance ..................................... 66
3.2.7 Scenario Analysis ............................................................ 67

3.3 Results and Discussion .......................................................... 68
3.3.1 Manure temperature and weather parameters during the storage period 68
3.3.2 Comparing ambient air temperature to manure temperatures ...... 69
3.3.3 Influence of depth on manure temperature .......................... 74
3.3.4 Predicting manure temperature with machine learning algorithms .. 75
3.3.5 Scenario analysis and availability of models for use ................. 82

3.4 Conclusion ......................................................................... 85

4.1 Introduction ......................................................... 95
4.2 Data ................................................................. 100
  4.2.1 Manure Storage Description and Data Collection and Preparation . 100
  4.2.2 Data Processing and Features Selection .................................. 101
  4.2.3 Description of Models ............................................... 102
4.3 Results ............................................................... 111
  4.3.1 Neural networks’ input features and hyperparameters ................. 111
  4.3.2 Models performance during validation .................................. 115
  4.3.3 Models performance during testing .................................... 119
  4.3.4 Cumulative performance ............................................. 123
4.4 Conclusion .......................................................... 126

5 Combining process-based modeling and recurrent neural networks to estimate ammonia emissions from stored dairy manure.

5.1 Introduction .......................................................... 137
5.2 Materials and methods .................................................. 141
  5.2.1 Models’ development ................................................. 141
  5.2.2 Data collection and processing ...................................... 149
  5.2.3 Models’ evaluation .................................................. 154
5.3 Results ................................................................. 155
  5.3.1 Hyperparameters search and sensitivity analysis ...................... 155
  5.3.2 Comparison of Physics-Informed LSTM and Process-Based Model . 162
5.4 Conclusion ................................................................. 166
References ................................................................. 168

6 Summary, Conclusions, and Future Research 174

Appendices 179

Appendix A  First Appendix: codes for Chapter 3 180
A.1 Models: ................................................................. 181
   A.1.1 Boosting Trees ................................................. 181
   A.1.2 Bagged trees .................................................. 185
   A.1.3 Random forest ................................................. 188

Appendix B  Second Appendix: codes for chapter 4 192

Appendix C  Third Appendix: codes for Chapter 5 284
   C.1 Compartmental process-based model code ................. 284
   C.2 The physics-informed LSTM model code .................. 305
List of Figures

2.1 Instrumentation setup for the manure storage for data collection showing the Ogawa samplers mounted around the perimeter (downwind (DS1, DS2) and upwind (US1, US2) poles), local weather station (W.S.), and the manure inlet (In) and pump-out (out) locations. ........................................... 30

2.2 Wind roses (percent frequency) for each sampling period during the study. ........................................... 36

2.3 Ammonia flux from dairy manure during the study period (Jun 24, 2021 - Apr 08, 2022) and the associated ambient air temperature, manure temperature, wind speed, relative humidity, rainfall, and crust thickness. ................. 39

2.4 The surface crust on stored manure: (a) three weeks after manure agitation and pump-out with a 50 percent crust cover 5 cm thick; (b) when full before the manure pump-out with 100 percent crust cover 30 cm thick; and (c) mixing during the manure pump-out event. ........................................... 40

3.1 The ambient air temperature (AAT) and manure temperature (MT) at various depths in the clay pit (CP) during the study period. (045, 090, 122, 198, and 244 refer to placement depth (cm) from the bottom of the storage structure of the MT sensor; P1 to P5 were the periods when manure was pumped out of the storage structure). ........................................... 71

3.2 The ambient air temperature (AAT) and manure temperature (MT) at various depths in the concrete tank (CT) during the study period. (045, 090, 137, 275, and 290 refer to placement depth (cm) from bottom of the storage structure of MT sensor; P1 to P5 were the periods when manure was pumped out of the storage structure). ........................................... 72
3.3 Manure temperature in the clay pit (a) March 2020 (DOY: 53 to 64); (b) June 2020 (DOY: 169 to 182); and (c) September 2020 (DOY: 245 to 259). AAT - ambient air temperature; MT - manure temperature; 045, 090, 168, 244 – sensor placement depth (cm) from the bottom of the pit.

3.4 Manure temperatures in the concrete tank (a) March 2020 (DOY: 80 to 91) (b) July 2020 (DOY: 199 to 213); (c) September 2020 (DOY: 245 to 273). AAT - ambient air temperature; MT - manure temperature; 045, 090, 137, 275 – sensor placement depth (cm) from the bottom of the pit.

3.5 The predicted and actual manure temperature (a) clay pit and (b) concrete tank storage structures. GBT- Gradient Boosted trees; BTE - Bagged trees ensembles; RFE- Random Forests ensembles; NN - neural networks.

3.6 The average monthly (a) historical and measured ambient air temperature and predicted manure temperature at the 0.45 m depth, predicted by the machine learning algorithms using historical weather data and measured manure temperature during the study period at the clay pit (b) and concrete tank (c). AAT – ambient air temperature; GBT- gradient boosted trees; BTE - bagged trees ensembles; RFE- random Forests ensembles; NN - neural networks; CP - clay pit; CT - concrete tank.

4.1 Flow diagram for the physics-informed neural network model used to predict the stored manure temperature.
4.2 The performance of the DDNN, PINN, and SSNN models with different numbers of hidden layers, each with five neurons (a) during the validation phase, (b) when tested on the data collected from the clay pit on the scraped farm, and (c) when tested on the data collected from the concrete tank on the flushed farm. (DDNN – data-driven neural networks; PINN – physics-informed neural networks; SSNN - Simulation study trained neural networks).

4.3 The performance of two hidden layers DDNN, PINN, and SSNN models with different numbers of neurons (a) during the validation phase, (b) when tested on the data collected from the clay pit on the scraped farm, and (c) when tested on the data collected from the concrete tank on the flushed farm. (DDNN – data-driven neural networks; PINN – physics-informed neural networks; SSNN - Simulation study trained neural networks). . . . . . . . . . . 114

4.4 The performance metrics for the trained models during the models’ validation and testing on the data from the clay pit manure storage (SCP) and the concrete manure storage tank in a flushing farm (FCT). (MAPE is the mean absolute percentage error; RMSE is the root mean square error. $R^2$ is the coefficient of correlation; FEHT– finite-elements heat transfer; DDNN is the data-driven neural network; SSNN is a Simulation study trained neural network; PINN is the physics-informed neural network.) . . . . . . . . . . . 116

4.5 Models validation: the average daily observed manure temperature, ambient air temperature, and the predicted manure temperature in the concrete storage tank located in a scrap farm (FEHT– finite-elements heat transfer model; DDNN – data-driven neural networks; PINN – physics-informed neural networks; SSNN- Simulation study trained neural networks; SCT – concrete storage tank located in a scrap farm; DOY – day of the year). . . . . . . . . . 118
4.6 Models testing on the SCP dataset: the average daily ambient air temperature, observed manure temperature, and predicted manure temperatures in the clay storage pit (FEHT – finite-elements heat transfer model; DDNN – data-driven neural networks; PINN – physics-informed neural networks; SSNN - Simulation study trained neural networks; SCP - clay storage pit located in a scrap farm; DOY – day of the year).

4.7 Models testing on the FCT dataset: the average daily ambient air temperature, observed manure, and the predicted manure temperatures in the concrete storage tank receiving manure from a flushed farm (FEHT – finite-elements heat transfer model; DDNN – data-driven neural networks; PINN – physics-informed neural networks; SSNN - Simulation study trained neural networks; FCT - concrete storage tank located in a flushing farm; DOY – day of the year).

4.8 Residual plots of the differences between the observed and predicted manure temperatures (a) during model validation, (b) when the models were tested on the data from scraped manure pit, and (c) when the models were tested on the data from a flushed concrete tank, along with the average. (FEHT– finite-elements heat transfer model; DDNN – data-driven neural networks; PINN – physics-informed neural networks; SSNN- Simulation study trained neural networks).

5.1 The information flow for (a) the compartmental process-based models, and (b) the physics-informed long-short term memory (LSTM) model.

5.2 The compartmental process-based model flow diagram. Source: Karunarathne et al. (2020) Physics-informed LSTM model.

5.3 The architecture of the long-short-term memory (LSTM) cell unit.
5.4 A summary of the physics-informed LSTM model training process. 151
5.5 Hyperparameter importance for the optimized physics informed LSMT model. 157
5.6 Slice plots showing the relationship between hyperparameters and the resulting model physics-informed validation error. 158
5.7 The performance of the process-based model using (a) the default physics-based hyperparameters, and (b) the optimized physics-based hyperparameters to simulate the ammonia emissions from a lagoon on a flushed dairy farm. 160
5.8 The performance of the process-based model using (a) the default physics-based hyperparameters, and (b) the optimized physics-based hyperparameters to simulate the ammonia emissions from a steel tank on the scraped dairy farm. 161
5.9 The performance evaluation metrics for the trained models during the testing on the data from the lagoon in the flushed farm (FL) and the steel manure storage tank in a scraped farm (SST). (MAPE is the mean absolute error, and RMSE is the root mean squared error). 163
5.10 The performance of the physics-informed LSTM at predicting the ammonia flux using the scaped steel tank testing dataset. 163
5.11 The performance of the physics-informed LSTM at predicting the ammonia emissions from flushed lagoon. 165
## List of Tables

2.1 The average monthly wind direction (percent frequency) from two-year local weather data .......................... 31

2.2 The average monthly wind direction (percent frequency) from two-year local weather data .......................... 32

2.3 A summary of ammonia emissions for liquid dairy manure storage reported in the literature. .......................... 44

3.1 The descriptive statistics of the weather and manure parameters at the clay pit (CP) manure storage. ................. 69

3.2 The descriptive statistics of the weather and manure parameters at the concrete tank (CT) storage structure ............. 70

3.3 The performance parameters for machine and neural network models trained and tested with 67% and 33% of data, respectively. ................. 79

3.4 Relative Importance for the input parameters for the models generated for the clay pit and concrete tank manure storage structure. ................. 82

4.1 The thermophysical properties of dairy manure. ................................................................. 105

4.2 The hyperparameters tuned for each model and the search domain ................................................. 109

5.1 Hyperparameters and the optimization ranges ................................................................. 149

5.2 Location-dependent compartmental process-based model input parameters at the flushed lagoon and scraped steel tank. ................................. 152

5.3 Compartmental process-based model inputs that depend on the manure phys- biochemical properties. ................................................................. 153
5.4 The optimum hyperparameters for flushed lagoon dataset and the scraped steel tank dataset. ................................. 156
Chapter 1

Introduction and Literature Review

1.1 Sustainable dairy production

Assessing and improving the sustainability of food and agricultural systems is critical in meeting the nutritional needs of a growing global population and ensuring food security. Dairy products are a valuable and affordable source of proteins and essential dietary nutrients such as calcium, vitamin D, and potassium, which are crucial for public health (Veltman et al., 2018). Traditionally, dairy production systems have focused on enhancing production efficiency to increase profitability. However, with the increasing global demand for dairy products, improving production efficiencies in dairy farms while protecting our natural resources like soil, water, and air is essential (Miller & Auestad, 2013; von Keyserlingk et al., 2013). Achieving a more sustainable dairy production requires practices considering environmental, social, and economic aspects. The Innovation Center for US Dairy supported the development of decision-support tools and life cycle assessments (LCA) to identify areas for improvement and minimize negative environmental impacts, profitability, and related social issues to promote sustainable dairy production practices. Studies show that manure management on dairy farms is one of the main areas that require improved sustainable practices (von Keyserlingk et al., 2013). Managing manure suitably is essential to minimizing the environmental impact of dairy farming and ensuring the industry’s long-term viability.
The United States Department of Agriculture (USDA, 2020) reports that an average lactating dairy cow produces 36 kgs of manure per 450 kgs of body weight daily. However, the excretion rate varies depending on the cow’s age, breed, and health. Dairy manure components include urine, feces, wasted feed, operational effluent, animal bedding, and water. Manure is a primary fertilizer source for crop and pasture production in livestock production, making effective manure management crucial. Manure management typically involves collecting, storing, and treating manure to allow use as fertilizer at the right time. Storing manure is essential to best management practices on farms, as it provides opportunities to maximize the fertilizer value, reduce handling costs, and minimize its potential to pollute the environment (Rotz, 2004; Veltman et al., 2018).

1.2 Fate of manure nitrogen during storage

During storage, the microbial activities and the biogeochemical processes that occur in manure alter its quantity and quality, resulting in the formation and release of gaseous constituents such as nitrogen, carbon, and sulfur (Amon et al., 2006; Smith et al., 2007). Thus, there is a need to pay attention to nitrogen in the manure, given that most of the nitrogen consumed in livestock production systems ends up in the stored manure, with an estimated nitrogen use efficiency of only 8% in the food supply chain (Kanter et al., 2020). Up to 60% of the stored manure nitrogen can be lost as ammonia during storage, depending on management and environmental conditions (Arogo et al., 2006; Baldé et al., 2016; Misselbrook et al., 2016). Losing nitrogen means reducing the fertilizer (nitrogen content) and economic value of manure, negatively impacting farm efficiency (Sommer et al., 2019). Moreover, ammonia losses significantly contribute to environmental pollution (Galloway et al., 2003; Pitesky et al., 2009). For instance, ammonia emissions adversely affect air quality, human health, biodiversity, soil health, and climate change (Hill et al., 2019; Y. Li et al., 2016; Sanchis et al.,...)
When released into the atmosphere, ammonia has the potential to degrade the local air quality by forming fine particulate matter aerosols (PM2.5) that have repercussions on human health. Nitrogen deposition from emitted ammonia can cause aquatic eutrophication, soil acidification, and vegetation toxicity (Arogo et al., 2006; Sanchis et al., 2019; Shah et al., 2006; Veltman et al., 2018). Further, nitrogen accumulation in nitrogen-sensitive ecosystems may impact plant biodiversity. Balde et al. (2018) estimated that in 2006, the health costs associated with ammonia emissions in the United States were 36 billion dollars. Therefore, effective manure storage practices are crucial to reducing ammonia loss and maximizing manure’s fertilizer value.

Several management practices have been used to mitigate ammonia emissions from stored manure. These include physical methods like covers and location of the inlet of manure into storage, as well as chemical methods like chemical manure additives, and biological methods such as microbial manure additives (Borowski et al., 2017; Vanderzaag et al., 2008; C. J. Clanton et al., 1999, 2001; McCrory & Hobbs, 2001; Muck & Steenhuis, 1982). However, despite these efforts, the mitigation strategies have not been able to fully address ammonia emissions due to the large land area involved in dairy production facilities. Therefore, the Environmental Protection Agency (EPA) has set a goal to decrease ammonia emissions from dairy and beef facilities by 10%. Hence, it is necessary to develop farm-level mitigation strategies based on the current emission inventories to attain this goal (EPA, 2011). Further, the nation’s emission inventories should consider all sources contributing to nitrogen losses, including manure storage.
1.3 Quantifying ammonia lost from stored manure

As the National Research Council (2003) highlighted in their book, the need to accurately quantify ammonia emissions from livestock production remains important. Knowledge of the quantity of ammonia losses is necessary to compile nutrient inventories at local, state, regional, and national levels and inform policymaking and decision processes to improve ammonia emission mitigation strategies (NRC, 2003). Moreover, precise estimates of ammonia emitted from stored manure are essential for meeting the goals of safe and healthy working conditions for farm workers and livestock. For example, the Occupational Safety and Health Administration (OSHA) recommends limiting ammonia exposure to 50 ppm during any 8-hour workday within a 40-hour workweek (OSHA, 2017). Therefore, developing safety procedures that comply with these standards necessitates scientifically sound estimates of ammonia losses from manure.

Methods for quantifying ammonia losses from livestock production include direct measurements, emission factors, and mathematical models. Among these methods, direct measurement is considered the most reliable. Direct measurement of ammonia concentration in the atmosphere can be done using several techniques, such as passive samplers (acid-based filters and scrubbers, detector tubes), electrochemical cells, optical absorption, photoacoustic, and gas chromatography (Arogo et al., 2006; Flesch et al., 2007; Grant & Boehm, 2015; Kupper et al., 2021; Shah et al., 2006; Thöni et al., 2003; Todd et al., 2008). The choice of measurement technique depends on the desired outcome and cost. For example, real-time direct measurements that include gas chromatography and infrared spectroscopy can be challenging and cost prohibitive (Arogo et al., 2006; Grant & Boehm, 2015, 2020; Karunarathne et al., 2020). Examples of real-time aerial pollutants measurements in livestock operations include the National Air Emissions Monitoring Study in the United States (Grant & Boehm, 2015, 2020) and Kupper et al. (2021) in Switzerland. On the other hand, passive samplers
offer a more straightforward and cost-effective way to measure integrated concentrations of atmospheric ammonia. (Puchalski et al., 2011; Roadman et al., 2003; Sather et al., 2008; Sommer et al., 2005; Todd et al., 2008). However, integrated measurements are sensitive to turbulent backflow or poor alignment with wind directions and are limited to integrated concentration measurement (Herrero et al., 2021; Puchalski et al., 2011; Roadman et al., 2003). In general, direct measurement methods for estimating emissions can be challenging and expensive depending on the site because of the equipment and skill level required to set up and conduct the measurements (Arogo et al., 2006; Shah et al., 2006).

An emission factor is another method regulatory agencies use to estimate pollutant emissions from on-farm sources, including ammonia. The emission factors for estimating emitted ammonia express the quantities as a function of the stored manure’s total ammoniacal nitrogen (TAN), the number of animals raised, or the mass of live animals raised (Sommer et al., 2019). However, emission factors vary significantly based on local management practices, environmental conditions, and manure storage structures. The standard deviation for the estimated emission factors for the ammonia emissions from stored manure is high due to the varying designs of storage structure (e.g., the depth-to-area ratio), management practices, carbon-to-nitrogen ratios, surrounding environmental conditions, and manure surface features (Faulkner & Shaw, 2008; Sommer et al., 2019; NRC, 2003). Additionally, emission factors used for animal production systems should reflect dietary, housing, and management practices in the US (Faulkner & Shaw, 2008).

Lastly, mathematical models, namely empirical and process-based models, are used to quantify the transport and emitted ammonia from manure storage structures. Emission and transport models often combine empirical, statistical, and mechanistic (process) components. The National Research Council (NRC) recommended using process-based models as an alternative and more meaningful approach to estimating aerial emissions from animal feeding.
operations (NRC, 2003). Several process-based models are commonly used to estimate ammonia emissions from manure storage, such as the Manure Denitrification-Decomposition (Manure-DNDC) model (C. Li et al., 2012), Integrated Farm System Model (IFSM) (Rotz et al., 2014), Compartmental Process-based Model (Karunarathne et al., 2020), Process-Based Ammonia Emission Model (Michigan Air et al., 2005), and Farm Emissions Model (Pinder et al., 2004).

These models can be used to plan new facilities and evaluate the effects of changing climatic conditions and management on ammonia emissions (Arogo et al., 2006). In addition, the inputs to these models are readily available information such as weather, farm characteristics, and manure management, making them adaptable and amenable for use under various climatic conditions and management scenarios. For example, they have been used to estimate ammonia emissions in different geographical locations, assess emissions from various management scenarios, and estimate ammonia losses for farm-based life cycle assessments. However, the current process-based models have several shortcomings that limit accurate model predictions. Thus, further development and refinement are necessary to improve or increase their accuracy. These limitations include inadequate consideration of the following six factors:

I. Spatial and temporal variability of temperature and concentration of manure within the storage; and the potential temperature stratification in manure layers.

II. The pertinent biogeochemical processes and their implementation in model simulations.

III. The continuously changing volume of manure during storage.

IV. The crusting on the surface of the stored manure.

V. The impact of different manure management practices (agitation and frequency of re-
moval), type of storage structure (concrete tanks, steel tanks, clay pits), and the shape and dimensions of the storage structure.

VI. Heat and mass transfer coefficients for the ammonia transportation and emission from stored manure.

1.4 Machine learning

Machine learning (ML) is a branch of artificial intelligence that builds applications that learn from data without relying on a predetermined equation as a model. ML techniques can be implemented to address the limitations of traditional process-based models to improve the accuracy of ammonia emission predictions. The advent of ML algorithms has uncovered new ways to work with complex systems, such as manure storage structures, and potentially overcome modeling challenges. Machine learning can handle complex systems with no existing formula, equations, or written rules, where tasks constantly change, and the nature of data also keeps changing (Nevala, 2017; Paluszek & Thomas, 2016). This study explores various machine learning techniques to address some of the challenges faced by process-based models.

One of the main challenges is that there is no standard method to estimate manure temperature as an input parameter in the process-based models (Rennie et al., 2017), although it is known that manure temperature drives the biogeochemical reactions in manure. For example, higher reaction rates can double or triple for every ten-degree centigrade increase in temperature, leading to higher loss rates of gaseous and volatile compounds such as ammonia from stored manure (Campbell & Norman, 1998; Leytem et al., 2017; Massé et al., 2003). The current approaches for estimating ammonia emissions use variants of ambient air temperature averages as surrogates for the manure temperature. For example, the IFSM model
assumes manure temperature as the average ambient air temperature of the previous ten days (Rotz et al., 2014). The manure-DNDC (C. Li et al., 2012) assumes that the average daily manure temperature equals the daily ambient air temperature. Lastly, the compartmental process-based model by Karunaratne et al. (2020) uses a liquid temperature approximation based on ambient air temperature. However, a few studies have documented that manure temperature runs higher than the average ambient temperature for most of the year and, in some instances, by up to 10 °C (Baldé et al., 2016; Genedy & Ogejo, 2021; Rennie et al., 2017, 2018). Thus, based on the results of these studies, using ambient air temperature as a surrogate does not reflect the actual manure temperature, and more accurate prediction methods are needed.

This study uses data-driven machine learning approaches to predict manure temperature based on the continuously measured temperature and ambient weather parameters during storage on three dairy farms. The four data-driven machine algorithms used in this study were the gradient boosting tree, bagging tree, random forest ensembles, and neural network. However, exclusively data-driven tools are prone to overfit training data and sometimes fail to discern the underlying relationships between the features and the observed training data. Furthermore, these models lack robustness and accuracy when tested on unobserved datasets, i.e., poor generalization (Pang et al., 2020; Yang & Perdikaris, 2018, 2019). Also, as the complexity of machine learning models increases, the computation time, the memory footprint, and the power consumption increase. Lastly, data-driven algorithms require a large amount of data for adequate training, which can be cost-prohibitive for manure storage applications. Therefore, the study also explores using physics-informed neural networks (PINNs) as an alternative method to predict the stored manure temperature.

Briefly, PINNs are universal function approximators that embed known physical, chemical, and biological relationships governing the dynamics of a system in the learning process.
(Raissi et al., 2017). Using the known laws governing the system dynamics during the training of the neural networks regularizes and limits the permissible solutions space, thus increasing the accuracy of the approximation. Thus, encoding structured information into a learning algorithm amplifies the content of the information in the data seen by the algorithm, enabling it to steer itself toward accurate solutions and generalize quickly. Finally, the PINN approach reduces overfitting data and improves the robustness of the model by adding a physics-informed regularization term to the loss function. PINNs have been successfully applied to various applications, including solving nonlinear differential equations (Patel et al., 2020; Raissi et al., 2017, 2019) and computational fluid dynamics (Kadeethum et al., 2020; Wang et al., 2022). PINNs have also been shown to be successful in fault detection (Shen et al., 2021), and traffic state estimation (Shi et al., 2021).

Further, to address the other process-based modeling challenges, we explored using machine learning approaches to predict ammonia emissions to provide a new perspective of learning the time dynamics and the underlying complex biogeochemical processes in the manure without fixed parameters or structures. This study introduces a deep-learning approach that combines the physics of manure storage systems and recurrent neural networks as an alternative method to estimate ammonia loss from liquid dairy manure during storage. We use a long short-term memory (LSTM) architecture, a type of recurrent neural network, to develop the ammonia emission predictive model. This model uses a physics-informed regularization term that embeds known physical, chemical, and biological relationships governing the dynamics of ammonia emissions. The proposed approach has several advantages; for instance, it predicts ammonia losses using easily measured parameters such as ambient air temperature and wind data, and it can be generalized to different manure management practices.
Furthermore, this study proposes an innovative approach to address the limitations of traditional process-based models by introducing an inverse problem-solving method to estimate the heat and mass transfer coefficients for ammonia transport and emission from stored manure. This model treats multiple process-based model coefficients as hyperparameters, and using a physics-informed hyperparameter search strategy, the model estimates the value of the coefficients that best describe the manure storage system. Hence, improve the accuracy of the process-based models in general. Thus, this study investigates how the integration of physics knowledge into the hyperparameters selection process to achieve optimum physical and neural network hyperparameters. The outcomes of this study contribute to the field of precision agriculture, specifically designing suitable on-farm strategies to minimize nutrient loss and greenhouse gas emissions during the manure storage periods and improve the accuracy of metrics used to assess sustainable manure management practices.

1.5 Objective

The study’s overall objective was to use machine and deep-learning techniques to improve the performance of process-based models that estimate the production and emission of ammonia from stored dairy manure. The specific objectives were to:

i. Quantify losses and the influence of manure characteristics and meteorological factors on ammonia emissions from dairy manure during storage.

ii. Investigate the potential of data-driven machine learning techniques as an improved method for estimating manure temperature during storage.

iii. Develop a combined physics and machine learning unified and generalized model for predicting manure temperature during storage.
iv. Develop a deep-learning method that integrates process-based modeling and recurrent neural networks as an alternative method to estimate ammonia loss from stored dairy manure.

1.6 Structure and Content

This is a manuscript-style dissertation comprising six chapters that are either published, under revision, or ready to be submitted to peer-reviewed journals in the pertinent area of agricultural technologies and neural computing in engineering applications. All the manuscripts associated with chapters 2, 3, 4, and 5, are reformatted to conform to the conventional dissertation format.

Chapter 1: Provides an overview of the literature, describes the study’s motivation, and outlines the structure of the dissertation.

Chapter 2: "Quantifying ammonia lost to the atmosphere during manure storage on a dairy farm as influenced by management and meteorological parameters," published in the Agriculture, Ecosystems, and Environment (AGEE) Journal.


Chapter 5: "Physics-Informed LSTM hyperparameters selection for ammonia emission predictions," manuscript under preparation to be submitted to a peer-review Journal.
Chapter 6: Presents the conclusions of the study and identifies some future work to advance the concepts conceived.

Lastly, appendices A, B, and C include the codes developed for chapters three, four, and five.

References


nia Model for Agricultural Sources Prepared for.


Chapter 2

Quantifying ammonia lost to the atmosphere during manure storage on a dairy farm as influenced by management and meteorological parameters.

Attribution

This chapter is based on the following manuscript:

Abstract

Storing manure provides opportunities to maximize the manure value as a fertilizer, reduce handling costs, and minimize its potential to pollute the environment; However, it presents the potential for losing the nitrogen in manure to the atmosphere. Thus, it is critical to know the quantities of nitrogen losses to inform the decision-making process related to mitigation strategies. This study undertook the task of quantifying ammonia lost at scale from manure stored in a clay pit at a dairy farm during two storage periods. The ammonia concentrations, manure temperature, and local meteorological factors were measured and used to calculate the ammonia flux. Further, the crusting characteristics of the stored manure surface were assessed and related it to ammonia loss. The flux of ammonia from the storage surface ranged from 0.26 ± 0.01 $gm^{-2}d^{-1}$ to 1.30±0.05 $gm^{-2}d^{-1}$, averaging 0.57 ± 0.02 $gm^{-2}d^{-1}$. Wind speed had the most influence on ammonia loss of the local meteorological factors. The ammonia flux patterns related more closely to the manure temperature than the ambient air temperature. The crusting of the manure suppressed the ammonia loss, and manure agitation before land application accounted for about 25 percent of the nitrogen loss. The ammonia lost during storage accounted for about 5 percent of the total nitrogen in the manure input. The outcome of this work contributes to the knowledge base farmers, practitioners, and policymakers use to design and improve the guidelines to formulate effective farm manure management and mitigation practices and regulatory programs to minimize ammonia loss from manure during storage on dairy farms.
Graphical abstract

The graphical abstract for this study is displayed in this section

2.1 Introduction

Ammonia emitted into the atmosphere contributes to the degradation of ambient air quality, human health, biodiversity loss, soil health, and climate change effects (Hill et al., 2019; Y. Li et al., 2016; Sanchis et al., 2019). Agricultural activities, including volatilization from manure during storage and use as fertilizer on croplands, are significant sources of ammonia in the atmosphere (Paulot et al., 2014; Wang et al., 2021). Dairy and cattle production alone accounts for about 40 percent of the national $NH_3$ emission inventory (Grant & Boehm, 2020). Livestock production commonly uses manure as a primary source
of fertilizer for crop and pasture production. Thus, it is common to store manure as part of best management practices on farms to maximize its fertilizer value, reduce handling costs, and minimize its potential to pollute the environment (Rotz, 2004; Veltman et al., 2018). However, during storage, the ammonia lost due to volatilization depends on the management and prevailing environmental conditions (Grant & Boehm, 2020; EPA, 2011). Reportedly, there is the potential of losing up to 60 percent of manure nitrogen as ammonia during storage (Arogo et al., 2006; Baldé et al., 2018; Misselbrook et al., 2016). Thus, it is important to design strategies to mitigate the impacts of ammonia lost to the atmosphere from livestock operations, such as dairy farms, described in this study. Designing these strategies requires knowledge of the quantities and the contribution of the various sources, such as manure storage. Also, the quantities provide information useful in compiling local, state, regional, and national nutrient inventories to inform policymaking and decision processes.

Estimating quantities of ammonia emitted into the atmosphere requires the knowledge of concentration. The methods for measuring atmospheric ammonia concentrations on farms employ passive sampling (acid-based filters and scrubbers, detector tubes), electrochemical cells, optical absorption, photoacoustic, and gas chromatography techniques (Arogo et al., 2006; Flesch et al., 2007; Grant & Boehm, 2015; Kupper et al., 2021; Shah et al., 2006; Thöni et al., 2003; Todd et al., 2008). Usually, the choice of which method to use depends on the desired outcome and cost. Real-time direct measurements can be challenging and cost prohibitive as the procedure requires sophisticated equipment and high skill levels to implement (Arogo et al., 2006; Grant & Boehm, 2015; Karunarathne et al., 2020; Kupper et al., 2021). Examples of real-time aerial pollutants measurements in livestock operations include the National Air Emissions Monitoring Study in the United States (Grant & Boehm, 2015, 2020; Heber et al., 2008) and Kupper et al. (2021) in Switzerland. On the other hand, if adequate, the integrated concentration measurement approach provides the benefits
of reduced costs and simplicity (Puchalski et al., 2011; Roadman et al., 2003; Sather et al., 2008; Sommer et al., 2005; Todd et al., 2008). Typically, measuring integrated concentrations is implemented via passive sampling. Briefly, in passive samplers, the compound of interest in the atmosphere diffuses through a reactive surface, where it is chemically trapped. The samplers can be left unattended for a long time, are cost-effective, easy to handle, and do not always require electricity. However, they are sensitive to turbulent backflow or bad alignment with wind directions and are limited to integrated concentration measurement (Herrero et al., 2021; Puchalski et al., 2011; Roadman et al., 2003).

Measuring ammonia emitted to the atmosphere from animal feeding operations is highly uncertain and complex. The different farm management practices, topography, and meteorological conditions over space and time exacerbate the challenges for estimating ammonia emissions (National Research Council, 2003). Specific to storage pits, it is important to note that manure is biologically active and interacts with its surrounding environment. Some typical challenges associated with estimating or modeling manure characteristics during storage include the continuously changing volume of manure during storage, crusting that occurs on the surface of the stored manure, the potential for temperature stratification in manure layers, manure management practices (agitation and frequency of removal), type of storage structure (concrete tanks, steel tanks, clay pits), and the shape and dimensions of the storage structure (Genedy & Ogejo, 2021; Genedy & Ogejo, 2022; Rennie et al., 2017). Ogawa passive samplers were used in this study to provide integrated atmospheric ammonia concentration to estimate nitrogen losses during storage. The adequacy, suitability, and success of using Ogawa samplers are well documented in the literature (e.g., Puchalski et al., 2011; Roadman et al., 2003; Sather et al., 2008).

To estimate ammonia emission rates from measured concentrations, micrometeorological techniques, such as the integrated horizontal flux, theoretical profile shape, or Backward...
Langrangian Stochastic (bLS) dispersion modelling, are used. The integrated horizontal flux method considered a reference method, has been extensively used and validated with passive samplers (Herrero et al., 2021; Misselbrook et al., 2016; Sanz et al., 2010; Sommer et al., 2005). However, its suitability for large, irregularly shaped sources is limited. On the other hand, the bLS dispersion modeling with integrated concentration measurements presents a cost-efficient and simple on-site ammonia monitoring and quantification alternative (Herrero et al., 2021; Sommer et al., 2005). But, when using the bLS dispersion model, there is a need to pay attention to sampling periods to meet the homogeneous atmospheric stability requirement (Herrero et al., 2021; Sommer et al., 2005). However, assuming neutral atmospheric stability conditions in bLS dispersion model calculations with integrated concentrations provides reasonable results (Herrero et al., 2021; Sanz et al., 2010; Sommer et al., 2005).

The aim of this study is to quantify ammonia losses from a dairy manure storage structure in Virginia, U.S., which handles scraped manure. Despite establishing the National Air Emissions Monitoring Study in 2005 by the U.S. EPA to address uncertainties in national ammonia emissions inventories for livestock waste, there are still large uncertainties. The study only included three dairy lagoons, providing slight variation in manure handling systems (Leytem et al., 2018). Moreover, the National Research Council noted the need for quantifying ammonia emissions using well-documented measurements from different management systems and environmental conditions. There are long-term studies on ammonia emissions from manure storage systems of dairies in the western U.S. but few in the eastern U.S. (Grant & Boehm, 2020). The western U.S. has a larger dairy industry, more concentrated animal feeding operations (CAFOs), and more intensive manure management practices than the eastern US, which is dominated by smaller farms (Pinder et al., 2004). Additionally, differences in environmental conditions, such as temperature and humidity, contribute to the
differences in ammonia emissions between the two regions. For instance, Rotz et al. (2021) noted that the western U.S. has higher ammonia emissions than the east due to its warm and dry climate.

This study fills the knowledge gap on ammonia emissions from manure storage systems in the eastern US, particularly for small dairies in Virginia. Although not easy, the study quantified ammonia loss at the farm scale level in a way that captures interactions between meteorological factors and manure characteristics. The outcomes reflect unique interactions that are difficult to recreate for laboratory experiments that mimic real-world conditions. Additionally, the farm-scale measurements nature of the study over extended periods spanning the storage cycles contributes to the data gap of lacking or inadequate information that captures emissions under real operating conditions needed to improve source emissions estimates and design mitigation strategies. This study estimated the influence of manure characteristics and meteorological factors on ammonia emissions during manure storage. The outcome of this work contributes to the knowledge and information to support developing and improving guidelines to formulate effective farm manure management and mitigation practices and regulatory programs to minimize ammonia loss from manure during storage on dairy farms. Finally, the information is necessary for the farmer to assess the lost economic opportunities associated with ammonia volatilization.

2.2 Materials and methods

2.2.1 Study site description

This study was conducted at an 85-milking dairy farm in Franklin County, Virginia (36.95° N, 79.82° W) over two manure storage cycles (from mid-June 2021 to mid-April 2022). The manure management at the farm entails scraping from the barn floors (twice a day) and
channeling it into a clay-lined earthen pit for storage (Figure 2-1). The manure storage pit is oval, with top surface dimensions of 60 and 27 m on the long and wide sides. The manure inlet and pump-out (outlet) locations are on opposite sides of the longer dimension of the storage pit. The manure input into the storage is through a 0.45 m diameter pipe about 0.15 m above the pit’s bottom. The average depth of the pit is 3.8 m with a bottom slope of 0.5 percent downslope from the inlet to the outlet end. The pit storage capacity is four months, but sometimes the manure stays in storage for an extended period depending on the contractor’s availability or suitability of the field conditions for manure application. During these extended periods, manure is partially pump-out to prevent potential spills from the storage structure. Manure is homogenized by agitation using a tractor-powered mixing pump before partial or full pump-out events and then loaded into slurry trucks to transport to land application sites as a fertilizer. The average characteristics of the stored manure were 2.04 gN L-1 total nitrogen, 0.62 gN L-1 total ammonium nitrogen, 0.15 g P L-1 total phosphorus, 1.36 g K L-1 potassium, 4 to 6 percent dry matter content, and a pH of 7.3.

2.2.2 Ammonia concentrations, weather parameters, and manure temperature

The integrated (time-averaged) atmospheric ammonia concentrations were measured using Ogawa passive samplers (Ogawa USA, Inc., Pompano Beach, Florida). By design, Ogawa samplers consist of glass-fiber filters infused with citric acid that reacts with the entrained ammonia in the ambient air passing through it. The filter enclosure is an open-ended solid Teflon cylinder assembly with two stainless steel screens that minimize the effects of dust and other contaminants.

The Ogawa samplers were prepared and deployed in the field following the manufacturer’s instructions. The activities included in deploying the Ogawa samplers entailed disassembling
the components and cleaning them thoroughly before use to avoid contamination. At the end of each sampling period, the filters retrieved from the Ogawa samplers were placed in tightly capped glass vials preloaded with 16 ml of de-ionized water and transported to the lab for analysis. The vials with the field and blank filters were vortexed at the lab for 10 minutes to enhance the extraction of the captured ammonia. The extracted ammonia solution was then analyzed following the APHA 4500- F phenate method (Walter, 1961) and Ogawa’s guidelines. Three aliquots representing three replicates of each field sample were analyzed for quality assurance, and five (0.5, 1.0, 1.5, 2.0, and 2.5 mg N L-1) standard ammonia solutions were used as control to create the calibration curve. The ammonia concentration of the blank sample was used as a correction factor for the measured values by the field samplers.

Figure 2-1 shows the relative locations of the weather station, manure temperature sensors, and the Ogawa samplers in the field. The weather station and the manure temperature sensor were at fixed locations. Two pairs of the Ogawa samplers, mounted on support poles, were located on the manure pit’s perimeter. The paired Ogawa sampler placement was in up and downwind locations based on the wind direction (percent frequency) for each sampling period. The first pair’s location captured the direction with the highest percent frequency, and the second pair captured the direction with the second highest frequency. The wind direction statistics to inform the experimental design was estimated from the farm’s meteorological data from 2019 to 2021.

Lastly, the wind directions were designated as north to northeast, northeast to east, east to southeast, southeast to south, south to southwest, southwest to west, west to northwest, and northwest to the north (Table 2-1).
Figure 2.1: Instrumentation setup for the manure storage for data collection showing the Ogawa samplers mounted around the perimeter (downwind (DS1, DS2) and upwind (US1, US2) poles), local weather station (W.S.), and the manure inlet (In) and pump-out (out) locations.

The installation heights of the downwind Ogawa samplers were 0.6 m, 0.9 m, 1.2 m, 1.5 m, and 1.8 m above the ground, and the upwind sampler at 0.9 m. Since these concentration measurement heights are in the low range end, it was assumed that atmospheric stability did not affect the concentrations measured (Sommer et al., 2005). The sampling seasons are the summer (Jun, Jul, Aug, Sep), fall (Sep, Oct, Nov, Dec), winter (Dec, Jan, Feb, Mar), and spring (Mar, Apr) (Table 2-2). The maximum ammonia capture capacity for the Ogawa samplers was 2,510 µg m$^{-3}$. Using information from Li et al. (2008), the capacity of the samplers was assumed to be enough to capture ammonia for periods over 30 days.

The DYACON® weather station (model MS-130, DYACON®, Logan, UT, USA) recorded the meteorological parameters (i.e., ambient air temperature, rainfall, wind speed, wind direction, relative humidity, and solar radiation). The 10-minute average wind speed and
Table 2.1: The average monthly wind direction (percent frequency) from two-year local weather data

<table>
<thead>
<tr>
<th>Wind direction</th>
<th>Jan</th>
<th>Feb</th>
<th>Mar</th>
<th>Apr</th>
<th>May</th>
<th>Jun</th>
<th>Jul</th>
<th>Aug</th>
<th>Sep</th>
<th>Oct</th>
<th>Nov</th>
<th>Dec</th>
</tr>
</thead>
<tbody>
<tr>
<td>N to NE</td>
<td>13.4</td>
<td>13.2</td>
<td>12.5</td>
<td>9.1</td>
<td>9.5</td>
<td>6.8</td>
<td>8.5</td>
<td>10.6</td>
<td>15.5</td>
<td>12.1</td>
<td>14.7</td>
<td>14.2</td>
</tr>
<tr>
<td>NE to E</td>
<td>8.2</td>
<td>6.4</td>
<td>8.7</td>
<td>6.5</td>
<td>4.3</td>
<td>8.6</td>
<td>7.3</td>
<td>10.3</td>
<td>9.7</td>
<td>9.1</td>
<td>10.0</td>
<td>6.6</td>
</tr>
<tr>
<td>E to SE</td>
<td>5.9</td>
<td>4.9</td>
<td>10.2</td>
<td>11.5</td>
<td>9.1</td>
<td>12.4</td>
<td>11.3</td>
<td>10.5</td>
<td>8.6</td>
<td>12.1</td>
<td>9.4</td>
<td>7.7</td>
</tr>
<tr>
<td>SE to S</td>
<td>8.2</td>
<td>8.9</td>
<td>8.2</td>
<td>10.0</td>
<td>8.8</td>
<td>18.8</td>
<td>16.8</td>
<td>14.1</td>
<td>14.8</td>
<td>12.4</td>
<td>8.0</td>
<td>7.8</td>
</tr>
<tr>
<td>S to SW</td>
<td>19.5</td>
<td>19.2</td>
<td>20.3</td>
<td>25.3</td>
<td>21.5</td>
<td>16.6</td>
<td>20.5</td>
<td>18.4</td>
<td>16.7</td>
<td>17.4</td>
<td>19.1</td>
<td>18.2</td>
</tr>
<tr>
<td>SW to W</td>
<td>12.7</td>
<td>16.1</td>
<td>12.0</td>
<td>13.9</td>
<td>16.0</td>
<td>11.7</td>
<td>16.6</td>
<td>12.7</td>
<td>9.6</td>
<td>14.4</td>
<td>13.5</td>
<td>15.7</td>
</tr>
<tr>
<td>W to NW</td>
<td>16.5</td>
<td>13.9</td>
<td>14.0</td>
<td>13.1</td>
<td>12.0</td>
<td>10.5</td>
<td>8.7</td>
<td>9.8</td>
<td>9.0</td>
<td>10.4</td>
<td>12.4</td>
<td>16.1</td>
</tr>
<tr>
<td>NW to N</td>
<td>15.7</td>
<td>17.5</td>
<td>14.1</td>
<td>10.6</td>
<td>18.9</td>
<td>14.4</td>
<td>10.1</td>
<td>13.6</td>
<td>16.2</td>
<td>12.1</td>
<td>12.9</td>
<td>13.6</td>
</tr>
</tbody>
</table>

N to N.E. is North to the northeast; N.E. to E is northeast to the east; E to S.E. is east to the southeast; S.E. to S is southeast to the south; S to S.W. is south to the southwest; S.W. to W is southwest to the west; W to N.W. is west to the northwest, and N.W. to N is northwest to the north.

wind direction were used in this study. The manure temperature was measured using HOBO-TMCx-HD sensors mounted at 0.45 m, 0.9 m, 1.7 m, and 2.4 m from the bottom of the storage pit on poles marked with stage or depth indicators. The temperature sensor poles were placed near the inlet, mid-section about 3 m from the edge, and near the pump-out location of the storage pit. The manure temperature reported was the average of the temperatures measured at the three locations. The weather parameters and the manure temperature were recorded every 30 min during the experimental period. The manure depth in the storage pit was calculated based on the filling rate.
Table 2.2: The average monthly wind direction (percent frequency) from two-year local weather data

<table>
<thead>
<tr>
<th>Period</th>
<th>Dates</th>
<th>Season</th>
<th>Duration</th>
<th>dm</th>
<th>Crust Thickness (cm)</th>
<th>General observation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Jun 24 to Jul 08</td>
<td>Summer</td>
<td>14</td>
<td>1.49</td>
<td>20</td>
<td>07/08 – crust cover 50% of the manure surface</td>
</tr>
<tr>
<td>2</td>
<td>Jul 08 to Jul 21</td>
<td>Summer</td>
<td>13</td>
<td>1.82</td>
<td>20</td>
<td>07/21 – crust cover 60% of the manure surface</td>
</tr>
<tr>
<td>3</td>
<td>Jul 21 to Aug 10</td>
<td>Summer</td>
<td>20</td>
<td>2.08</td>
<td>30</td>
<td>08/10 – crust cover 80% of the manure surface</td>
</tr>
<tr>
<td>4</td>
<td>Aug 10 to Sep 03</td>
<td>Summer</td>
<td>24</td>
<td>2.36</td>
<td>30</td>
<td>09/03 – crust cover 100% of the manure surface</td>
</tr>
<tr>
<td>5</td>
<td>Sep 03 to Sep 23</td>
<td>Summer</td>
<td>20</td>
<td>2.59</td>
<td>25</td>
<td>09/23 – crust cover 100% of the manure surface</td>
</tr>
<tr>
<td>6</td>
<td>Sep 23 to Oct 18</td>
<td>Fall</td>
<td>25</td>
<td>2.83</td>
<td>30</td>
<td>10/18 – crust cover 100% of the manure surface</td>
</tr>
<tr>
<td>7a</td>
<td>Oct 18 to Nov 11</td>
<td>Fall</td>
<td>14</td>
<td>1.76</td>
<td>20</td>
<td>11/01 – crust covering 50% of the manure surface</td>
</tr>
<tr>
<td>8b</td>
<td>Nov 11 to Dec 03</td>
<td>Fall</td>
<td>32</td>
<td>1.19</td>
<td>&lt;2</td>
<td>12/03 - crust layer initiation on 10% of the surface</td>
</tr>
<tr>
<td>9</td>
<td>Dec 03 to Dec 17</td>
<td>Fall</td>
<td>14</td>
<td>0.44</td>
<td>&lt;2</td>
<td>12/17 – crust covers 10% of the manure surface</td>
</tr>
<tr>
<td>10</td>
<td>Dec 17 to Jan 11</td>
<td>Winter</td>
<td>25</td>
<td>0.82</td>
<td>&lt;2</td>
<td>01/11 - crust cover 20% of the manure surface</td>
</tr>
<tr>
<td>11</td>
<td>Jan 11 to Feb 02</td>
<td>Winter</td>
<td>22</td>
<td>1.30</td>
<td>2</td>
<td>02/02 - crust covers 40% of the manure surface</td>
</tr>
<tr>
<td>12</td>
<td>Feb 02 to Mar 11</td>
<td>Winter</td>
<td>37</td>
<td>1.95</td>
<td>3</td>
<td>03/11 - crust covers 50% of the manure surface</td>
</tr>
<tr>
<td>13</td>
<td>Mar 11 to Apr 08</td>
<td>Winter/Spring</td>
<td>28</td>
<td>2.54</td>
<td>4</td>
<td>04/08 - crust covers 70% of the manure surface</td>
</tr>
</tbody>
</table>

* manure agitation and partial pump-out on 10/18/2021;  † manure pumped out on 11/19/2021
To facilitate the description and estimation of the surface crusting, the storage structure during every sampling period was visually assessed to determine the stage on the temperature poles, surface conditions, and crusting characteristics. Also, pictures and videos were taken of the manure in the storage structure and used them to assess the surface crust coverage. Lastly, during each field visit, the stage marking on the manure temperature sensor poles helped in assessing the crust thickness.

### 2.2.3 Calculating ammonia flux and ammonia lost

The $NH_3$ flux from the stored manure was calculated using the WindTrax dispersion model (version 2.0.9.7, Thunder Beach Scientific). WindTrax uses the backward Lagrangian stochastic inverse-dispersion modeling technique to quantify emissions from a source. The details and underlying theory of the model are in Flesch et al. (1995, 2004, 2007), Flesch & Wilson (2005), and Sommer et al. (2005). The input parameters for running WindTrax included: the measured ammonia concentrations, concentration measurement heights, the distance between the upwind and downwind concentration measuring locations, the surface area of the manure storage, surface roughness, atmospheric stability, and the meteorological parameters (wind speed, wind direction, ambient air temperature, and pressure).

WindTrax (simulations of 50,000 trajectories) simulated the ammonia flux for each sampling period using the Ogawa samplers’ integrated concentrations as input. The input parameters included the ammonia concentrations measured and their relative placement (downwind, upwind, and height above the ground) to account for the variability of the samplers’ response. This approach enabled us to account for the ammonia concentrations simultaneously downwind in four different wind directions. Finally, the standard deviation for each measurement was calculated using WindTrax to account for the differences among the samplers. The meteorological factors for each sampling period were represented as the average of the half-hourly
data collected. The flux estimation assumes (1) constant flux over each sampling period, (2) surface roughness of 0.09 m following (Todd et al., 2008), and (3) neutral atmospheric stability (based on findings of Sommer et al., 2005). The nitrogen lost as ammonia from the manure calculations are as follows:

a. Total manure nitrogen during each sampling period is the product of manure’s total nitrogen concentration and the volume of manure in storage during each sampling period.

b. Nitrogen lost as ammonia during each sampling period is the product of flux, surface area, and period length.

c. The percent nitrogen lost during each sampling period is the nitrogen lost as ammonia divided by the total nitrogen in the stored manure.

d. Overall total nitrogen lost is the sum of all the nitrogen lost during each period.

e. Nitrogen lost per cow per day: Total nitrogen lost during all periods divided by the period length in days, divided by the average number of cows milked.

2.3 Results

2.3.1 Weather conditions

The ambient air temperature during the study period ranged from -13.7 °C to 32.7 °C, with an average of 14.4 °C. The summary of the wind speeds and associated directions for each sampling period are in Figure 2-2. The wind speed generally averaged 1.31 \( ms^{-1} \), ranging from 0 to 8.30 \( ms^{-1} \). The dominant wind directions (based on percent frequency) were the southeast to the south, the south to the southwest, and the northwest to the north. Thus, during the study period, the wind direction was mostly from the southern end of the manure
storage pit, agreeing with the two-year wind data.

### 2.3.2 Ammonia flux

The ammonia flux ranged from 0.32 (± 0.01) \( \text{g m}^{-2} \text{d}^{-1} \) to 1.41 (± 0.05) \( \text{g m}^{-2} \text{d}^{-1} \), with an average of 0.61 (±0.02) \( \text{g m}^{-2} \text{d}^{-1} \). The highest flux occurred during the period associated with mixing activities to homogenize manure to facilitate partial and complete manure pump-out for land application, i.e., between Oct 18 and Nov 01 for this study. In comparison, lower fluxes were associated with periods when there was a thick crust on the surface of the stored manure, e.g., between Sep 23 and Oct 18. The magnitudes of ammonia flux in this study are in the lower ranges of the literature values that range between < 0.05 \( \text{g m}^{-2} \text{d}^{-1} \) and 40.08 \( \text{g m}^{-2} \text{d}^{-1} \) (Arogo et al., 2006; Kupper et al., 2021).

The total ammonia lost during the study period (289 days) was 134 kg of nitrogen, which equates to approximately 0.46 kg per day for the entire herd or 5.45 g per milking cow per day. However, a deeper look at the nitrogen loss reveals that 32.4 kg occurred before, 32.9 kg occurred during, and 68.3 kg occurred after the manure removal events. In addition, the ammonia loss by season was 24.4, 39.7, and 24.6 percent for the summer, fall, and winter, respectively. Lastly, compared to the total nitrogen content of the stored manure, ammonia volatilization accounted for about 5 percent during study storage periods. Using the fertilizer prices published by the United States Department of Agriculture, Natural Resource Conservation Service (USDA NRCS), the cost of nitrogen-based fertilizer ranges from $1.9 to $6.6 per kg, depending on the type of fertilizer formulation. Thus, the ammonia volatilized in this study reflects a $255 to $885 (€210 - €820) revenue loss for the farmer.
Figure 2.2: Wind roses (percent frequency) for each sampling period during the study.
2.3.3 Influence of meteorological factors and manure characteristics on ammonia loss

Figure 2-3 shows the trends and magnitudes of ammonia flux and the averages of the ambient air temperature, manure temperature, wind speed, rainfall, and crust thickness. The ammonia flux increased with the wind speed and the manure temperature, consistent with Kupper et al. (2020, 2021) and Baldé et al. (2018). However, the magnitudes of ammonia fluxes during the periods with lower wind speeds were smaller than in the periods with high wind speeds, even if the manure and the air temperatures were high. This result suggests that the wind speed strongly influenced the ammonia flux more than the ambient and manure temperatures and is consistent with observations from previous studies reporting the factors that influence ammonia loss from manure during storage (Kupper et al., 2021; Leytem et al., 2018; Misselbrook et al., 2016; Smith et al., 2007; Todd et al., 2005). Further, based on the known ammonia chemistry in aqueous solutions, increasing wind speeds above the surface enhances the manure’s evaporative losses and disrupt the boundary layer. As ammonia loss to the ambient environment increases, the transport of ammonium ions in the bulk slurry to the manure surface increases to enhance the ammonia and ammonium equilibrium (Arogo et al., 1999; Kupper et al., 2020, 2021; C. Li et al., 2012).

The changes and variability of manure temperature were much lower than the observed ambient air temperatures. The manure temperature was stable and almost constant (24°C) until Oct 18, when the pit was agitated to prepare the manure for removal and land application. On the other hand, the air temperature dropped from 24°C to 14°C between Aug 10 and Oct 18. Concurrently, between Aug 10 and Oct 18, the ammonia flux remained almost constant (0.29 $gm^{-2}d^{-1}$) and followed the manure temperature patterns more closely than the ambient air temperature. This result suggests that the manure temperature influenced ammonia emissions more than the ambient air temperature. During the manure agitation
and pump-out events, the flux did not follow the trends in manure temperature or meteorological factors (i.e., Oct 18 and Dec 03). Still, the ammonia flux, manure temperature, and ambient air temperature patterns were similar after the manure removal. Finally, there were no clear patterns between rainfall, manure depth, and ammonia flux. This finding is inconsistent with what some studies report about ammonia flux declining with rainfall (Kupper et al., 2020; Petersen et al., 2013; Todd et al., 2005). The reasons presented as the influence of rainfall is suppressing the transport of the dissolved ammonia in the bulk manure and increasing the moisture content of the crust, reducing its ability to slow ammonia loss (Smith et al., 2007).

2.3.4 Surface manure crusting and ammonia emissions

The crusting phenomenon during manure storage occurs when gases released by the manure bubble to the surface carry fiber particles, creating the surface crust (Smith et al., 2007). The crusting characteristics observed in this study are in Table 2-2. At the beginning of the study, a 7 cm thick crust covering about 50 percent of the surface near the pump-out location was present. The crusting starts developing near the manure inlet for this manure storage, then propagates toward the pump-out location (Figure 2-4 a). As the manure filled the storage, the crust thickness grew to about 13 cm and covered the entire surface of the storage manure (Figure 2-4 b). The crust closer to the outlet was drier than the crust near the inlet, perhaps because the crust near the outlet is older and dried over time. During the manure pump-out event, the agitation and mixing destroyed the surface crust (Figure 2-4 c). After pumping out the manure, there is no surface crust, but over time, a thin layer develops near the inlet, becoming measurable approximately one month after the manure removal.

Interestingly but not surprisingly, the ammonia emissions during periods with a thick crust were lower than in periods with thin or no crusting (Figure 2-3). This result suggests
Figure 2.3: Ammonia flux from dairy manure during the study period (Jun 24, 2021 - Apr 08, 2022) and the associated ambient air temperature, manure temperature, wind speed, relative humidity, rainfall, and crust thickness.
that crusting creates a physical barrier that prevents ammonia losses, which is beneficial for retaining manure nitrogen. Notably, a thick crust layer increases the diffusion path, resulting in a lower ammonia mass transfer rate across the surface and retaining the dissolved ammonia in the bulk manure (Baldé et al., 2018; Kupper et al., 2021; Misselbrook et al., 2016; Smith et al., 2007).

Figure 2.4: The surface crust on stored manure: (a) three weeks after manure agitation and pump-out with a 50 percent crust cover 5 cm thick; (b) when full before the manure pump-out with 100 percent crust cover 30 cm thick; and (c) mixing during the manure pump-out event.

Typically, manure removal is associated with agitating and pumping out the manure over
The vigorous agitation and absence of crust during the manure pump-out events explains the substantial increase in the ammonia flux between Oct 18 and Dec 03. The ammonia emissions in the periods after agitation and manure pumping out (i.e., when the manure surface has minimal crust) accounted for 25 percent of the total measured emissions.

The nature (dry or wet) and the crust’s integrity (cracked or intact) affect ammonia flux. Crusts with low moisture content form a natural cover that is a barrier to the gas molecules between the liquid and the air. Also, when the crust is wet, its ability to suppress ammonia volatilization is reduced (Misselbrook et al., 2005). In this study, the crust on the manure surface during the summer and early fall (i.e., warm period) were drier than the crust in the winter and early spring (i.e., cold period). A reason for this could be a combination of higher air temperatures and reduced relative humidity influences on manure drying, as Smith et al. (2007) reported. The dry crust could also explain the low ammonia flux recorded during the warm periods. In general, the ammonia flux when the crust was thick and dry was 60 percent less than when the crust was thin with high moisture content.

Increasing manure temperature is expected to enhance ammonia volatilization since it directly affects the diffusion and enhances the convective transfer rate of ammonia across the liquid-air phase boundary (Arogo et al., 1999; Kupper et al., 2021; Sanchis et al., 2019). However, the influence of crust thickness on ammonia flux reduced the direct effect of temperature. Figure 2-4 shows that thick crusting forms during warm periods. Smith et al. (2007) reported that higher temperatures enhance the release of gases (e.g., CO2 and CH4) that coalesce around the fiber particles and form the crust. This phenomenon may explain the presence of thicker crusts during periods with high ambient air and manure temperatures. Thus, observing low ammonia emissions during warm periods suggests that crusting may override the temperature effects on ammonia loss. Finally, when a stable crust thickness is present at the manure surface, and wind speeds are constant (e.g., between Jun 24 and
Oct 18), the ammonia loss does not follow the manure or ambient air temperatures (Figure 2-3). This result implies the potential of combined crusting and wind speed overriding the temperature effects on ammonia loss.

2.4 Discussion

Table 2-3 compares the data collected in this study to the data available in the literature. The average ammonia flux in this study is at the lower end of the values reported in the literature. One of the reasons is that during warm periods when most of the studies reported high emissions, the clay pit had thick crusting, which acted as a barrier against ammonia emissions. For instance, during the summer, the average ammonia flux in this study was lower than the values reported by Kupper et al. (2021) and McGinn et al. (2008) by 77 and 92 percent, respectively. This is attributed to the fact that both manure storage structures experienced frequent agitation during the summer, disrupting the crusting and increasing the emissions. In contrast, a consistently thick crust covering the surface of the manure storage structure throughout the summer and most of the fall was observed in our study. This indicates that retaining thick crusting during warm periods can be a viable strategy for reducing ammonia emissions. According to Kupper et al. (2021), the ammonia emissions during the day when the agitation occurs are 65 percent higher than the emissions measured one to 14 days after agitation and 75 percent higher than the emissions measured 14 or more days after agitation.

The highest level of ammonia emissions occurred during the spring season, consistent with Leytem et al. (2014), due to the influence of high wind speeds and manure agitation. Although summer and fall seasons typically have warmer temperatures and are expected to result in higher emissions, the effect of wind speed and manure agitation prevailed over tem-
perature. However, the peak ammonia flux observed in this study was almost half than that reported by Leytem et al. (2014). Also, Grant & Boehm (2020) reported ammonia fluxes in Indiana and Wisconsin that were 80 percent and 450 percent higher, respectively, than the measurements in this study. These differences may be due to varying wind speeds, as this study recorded wind speeds that were 35-65 percent lower than those reported in Indiana and Wisconsin. Alternatively, the differences could be attributed to differences in manure management practices, specifically, the use of scraped manure in this study compared to the use of flushed dairy in Grant & Boehm (2020) and Leytem et al. (2014). The higher moisture content in flushed manure typically results in thinner or no crusting, which leads to higher ammonia volatilization rates. Lastly, it is worth noting that Grant & Boehm (2020) and Leytem et al. (2014) conducted their studies in the western region of the U.S., where the climate is typically warmer and drier than Virginia, which may stimulate ammonia emissions from manure storage structures (Rotz et al., 2021).

Our measurements show good agreement with the literature when manure management and weather conditions are comparable. For example, during the winter months (December, January, and February), our observed ammonia flux of $0.65 \text{ gm}^{-2} \text{d}^{-1}$ is in close agreement with the value reported by Kupper et al. (2021) of $0.6 \text{ gm}^{-2} \text{d}^{-1}$. The ambient air temperature ($3.5^\circ\text{C}$) and wind speed ($1.4 \text{ m/s}$) recorded in the study by Kupper et al. (2021) during winter were also similar to the values observed in our study during the same period. Furthermore, both studies involved similar manure storage structures with comparable crust characteristics and manure management practices during the winter season. This validates the ammonia quantification technique used in this study.
Table 2.3: A summary of ammonia emissions for liquid dairy manure storage reported in the literature.

<table>
<thead>
<tr>
<th>Manure handling and storage</th>
<th>Location</th>
<th>Manure characteristics</th>
<th>AAT, °C</th>
<th>W.S. m⁻¹</th>
<th>Experiment period</th>
<th>NH₃ flux, gm⁻²d⁻¹</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scraped, clay-lined pit</td>
<td>Virginia, U.S.</td>
<td>19.5 2.0 0.60 7.3</td>
<td>14.4</td>
<td>1.3</td>
<td>06/2021 - 04/2022</td>
<td>0.57</td>
<td>This study</td>
</tr>
<tr>
<td>Scraped, steel tank</td>
<td>Switzerland</td>
<td>56 1.39 11.6</td>
<td>1.3</td>
<td></td>
<td>01/2015 - 04/2017</td>
<td>1.54</td>
<td>Kupper et al. (2021)</td>
</tr>
<tr>
<td>Flushed, lagoon</td>
<td>Indiana, U.S.</td>
<td>7.8 8.9 2.6</td>
<td></td>
<td></td>
<td>09/2008 - 08/2009</td>
<td>0.09</td>
<td>Grant &amp; Boehm (2020)</td>
</tr>
<tr>
<td>Flushed, clay-lined basin</td>
<td>Wisconsin, U.S.</td>
<td>6.9 7.9 2.7</td>
<td></td>
<td></td>
<td>09/2007 - 10/2009</td>
<td>1.54</td>
<td>Grant &amp; Boehm (2020)</td>
</tr>
<tr>
<td></td>
<td>Hunan Province, China</td>
<td>0.4 8.7 33</td>
<td></td>
<td></td>
<td>07/25/2015 - 08/26/2015</td>
<td>1.09</td>
<td>Zhuang et al. (2020)</td>
</tr>
<tr>
<td>Scraped, earthen basin</td>
<td>Ottawa, Ontario, CA</td>
<td>39 2.5 1.08 7.3</td>
<td></td>
<td></td>
<td>06/2013 - 12/2014</td>
<td>2.66</td>
<td>Baldé et al. (2018)</td>
</tr>
<tr>
<td>Flushed, lagoon</td>
<td>Southern Idaho, U.S.</td>
<td>0.11 7.7 7.9 4.3</td>
<td></td>
<td></td>
<td>09/2010 - 06/2011</td>
<td>1.6</td>
<td>Leytem et al. (2018)</td>
</tr>
<tr>
<td>Scraped, lagoon</td>
<td>Alberta, CA</td>
<td>0.83 22.3</td>
<td></td>
<td></td>
<td>06/2007 - 07/2007</td>
<td>5.1</td>
<td>McGinn et al. (2008)</td>
</tr>
</tbody>
</table>

NH₃ - Ammonia; DM – dry matter; TN – Total nitrogen; TAN – Total ammoniacal nitrogen; AAT – ambient air temperature; W.S. – wind speed
2.4.1 Study limitations

Some study limitations include determining the precise locations of the upwind and downwind samplers. By nature, the prevailing wind characteristics vary and may not follow the patterns predicted by historical data. Thus, sampler locations may be inadequate. This challenge was addressed by using two pairs of samplers to capture a wider reach for the wind direction frequency. Another limitation is assuming constant ammonia flux over each experimental period, a standard limitation for integrated concentration measurement methods such as passive samplers. Using constant flux provides only the general trend of the ammonia flux under the meteorological factors and manure management practices.

2.5 Conclusion

The need for research at the farm scale to generate information to improve estimates of emissions from animal feeding operations remains, as reiterated by the National Research Council in 2003 (National Research Council, 2003). Making credible estimates at this scale are necessary to improve the understanding of animal production systems and guide developing technologies to manage, control, and regulate emissions from animal feeding operations. This study employed the use of Ogawa samplers to quantify ammonia lost from stored dairy manure at a farm with a scrape manure removal system. Ogawa samplers provide a simple and cost-effective approach to quantify ammonia losses from stored dairy manure. Ogawa samplers do not require extensive technical expertise or specialized equipment, making it accessible to a wide range of researchers and farmers. In addition, this study can still make a valuable contribution by providing quantified data on ammonia emissions from manure storage at small dairy farms in Virginia which can be useful for other small dairy farms facing similar resource limitations.
The study captured the influences and interactions of seasonal weather cycles, manure management, and manure characteristics on ammonia emissions. The average ammonia flux from the surface of the stored manure was 0.57 $gm^{-2}d^{-1}$ with a maximum of 1.30 $gm^{-2}d^{-1}$. The total ammonia lost during the study period was 134 kg of nitrogen, equating to a significant revenue loss for the farmer. This study has highlighted the importance of crust formation on the manure surface as a physical barrier that reduces ammonia emissions, thereby retaining the manure’s nitrogen value. However, the benefits of crusting are lost when manure is agitated and mixed in preparation for land application. These activities destroyed the manure’s natural crust and contributed to 25 percent of the ammonia lost from the manure during storage. Based on this study, retaining the manure crusting during warm periods and high wind speed events can highly reduce the ammonia emissions from manure storage structure. The ambient air temperature, wind speed, and manure temperature affect the ammonia emissions from stored manure. Small changes in the wind speed considerably affected the ammonia flux and the influence of wind speed can override the influence of the other weather conditions. In addition, ammonia flux trends followed the manure temperature trends more closely compared to the ambient air temperature.

This work potentially provides good information to guide the design and formulation of effective farm manure management practices that minimize nitrogen lost as ammonia. However, this study alone cannot make up for the deficiencies in accounting for ammonia emissions from manure storage systems in the eastern region. More research is needed to improve the on-farm regulations and laws. Finally, the data generated can be helpful to those developing and validating mathematical models and software applications to obtain scientifically sound estimates of air emissions from animal feeding operations.
Acknowledgments

This study is supported by funds from the USDA National Institute of Food and Agriculture (NIFA) AFRI Foundational Program grant no. 2017-67019-26399 and the Virginia Agricultural Experiment Station.

References


52


Chapter 3

Using machine learning techniques to predict liquid dairy manure temperature during storage.

Attribution

This chapter is based on the following manuscript:

Abstract

There is no standard method to predict manure temperature during storage. So, decision support tools, on-farm nutrient cycling models, and life cycle assessment tools to assess the sustainability of agricultural production systems that include manure typically use ambient air temperature as a surrogate for manure temperature. This study explores the application of machine learning algorithms’ unique abilities to predict manure temperature based on measured data. The data was collected from two on-farm manure storages (clay pit and concrete tank) instrumented with sensors to acquire manure temperature at various depths during the storage period. The local weather data (ambient air temperature, wind speed, wind direction, solar radiation, relative humidity, and rainfall) were recorded by stations installed at each farm. The data were subjected to four machine learning algorithms gradient boosted trees, bagged tree ensembles, random forest ensembles, and neural networks using the supervised learning approach. The weather data and two additional parameters, time (month) and the manure depth above a sensor were derived and used as inputs for the machine learning algorithms. Further, the developed machine learning algorithms were challenged with parameters from historical weather data (1990 to 2020) to assess their suitability to predict manure temperature where local weather is not available.

The results showed that, in general, the stored manure temperature lagged but followed a similar trend as the ambient air temperature and solar radiation. The average manure temperature was higher than the ambient air temperature for most of the year. Depth influenced the manure temperature; manure in the top layers had a higher temperature during warm periods than the bottom layers, and vice versa during cold seasons. The ensemble models performed better than the neural networks by predicting manure temperatures closer to the measured values and predictions during the scenario analysis. The random forests and bagged tree ensembles were the best performers. Models tended to make better predictions
as the depth of manure above a sensor increased. This work will provide added value for developing better decision-support tools and models for assessing nutrient cycling on farms. It also informs our knowledge to develop emission mitigation strategies during manure storage, leading to more sustainable manure management practices.

**Graphical abstract**

The graphical abstract for this study is displayed in this section
3.1 Introduction

Manure management, including collection, storage, treatment, and use, has important implications for dairy farms’ sustainable production. Typically, storage provides opportunities to maximize the value of manure as a fertilizer for crop production, reduce handling costs, and minimize the potential to pollute the environment (Kellogg et al., 2000). However, during storage, a series of microbial and physical processes occur to the manure that affects its quality and composition, leading to the loss of volatile nitrogen, carbon, and sulfur compounds (Pattey et al., 2005; Elsgaard et al., 2016). The loss of these volatile compounds presents the potential to pollute the environment. Factors that affect the physical, biogeochemical, and microbial activities associated with manure degradation include manure properties (organic matter, pH, temperature, and moisture content) and type of storage (Elsgaard et al., 2016). Manure temperature plays a major role in the physical, biogeochemical, and microbial activities associated with the degradation process. Higher manure temperatures favor increased microbial activities and rates of chemical reactions for any given operational temperature range. The reaction rates can double or triple for every ten-degree centigrade increase in temperature, leading to higher loss rates of gaseous and volatile compounds from stored manure (Campbell and Norman, 1998; Elsgaard et al., 2016).

Estimating manure temperature is common in using decision support tools or models in precision agriculture to assess food and agricultural production systems’ sustainability. However, there is no standard method to estimate manure temperature as an input parameter for the decision support tools, models for conducting on-farm nutrient balance, and life cycle assessments (Rennie et al., 2017). The current approaches for estimating manure temperature use variants of ambient air temperature averages as surrogates and experiences from lake water temperatures. For example, in the Intergovernmental Panel on Climate Change (IPCC) model, methane emissions from anaerobic lagoons are estimated using average monthly am-
ambient air temperature (Mangino et al., 2001). The integrated farm systems model (IFSM) manure storage module for assessing gaseous emissions assumes manure temperature as the average ambient air temperature of the previous 10 d (Chianese et al., 2009). The manure-DNDC (Li et al., 2012) assumes that the average daily manure temperature equals the daily ambient air temperature. The process-based model by Karunarathne et al. (2020) uses a liquid temperature approximation based on ambient air temperature.

A few studies have reported on-farm or field measurements of dairy manure temperature during storage in the past two decades (Johannesson et al., 2018; Rennie et al., 2017; Baldé et al., 2016; Massé et al., 2008). These studies report that the average manure temperature is usually higher than the ambient air temperature, in some instances, by up to 10 °C, suggesting that using ambient air temperature as a surrogate for manure temperature may not be appropriate. The most comprehensive effort to date that we found in the literature was Rennie et al. (2017), a 3-D numerical heat transfer model for estimating year-round manure temperature. In this model, the average manure temperature simulated and measured were almost identical. The model performed best in summer and autumn periods when manure temperature and manure volume were high. The model was judged adequate for these periods considered most critical for gaseous emissions from stored manure.

Some typical challenges associated with estimating or modeling manure temperature during storage include the continuously changing volume of manure during storage, crusting that occurs on top of the manure storage surface, the potential for temperature stratification in manure layers, manure management practices (agitation and frequency of removal), type of storage structure (concrete tanks, steel tanks, clay pits), and the shape and dimensions of the storage structure (Rennie et al., 2017; Massé et al., 2008). Additionally, there are no appropriate coefficients for manure to use in the classical heat and mass transfer modeling of manure temperature. Fortunately, the advances in machine learning (ML) algorithms
have uncovered new ways and opportunities to work with complex systems, such as the manure storage environment, to address and potentially overcome these challenges. Machine learning, briefly, is a branch of artificial intelligence focused on building applications that learn from data without relying on a predetermined equation as a model; typically, the algorithms improve their performance adaptively as the number of samples for learning increases (Witten et al., 2011). The strength of ML lies in its capacity to handle complex systems with no existing formula, equations, or written rules, where tasks are constantly changing, and the nature of data also keeps changing, with characteristics that require the program to adapt.

This study explores using the supervised machine learning approach to predict manure temperature based on continuously measured manure temperature and the associated ambient weather parameters during storage on a farm. The supervised machine learning approach entails training, testing, and validation steps to generate predicted manure temperatures. To our knowledge, no studies have reported on the application of ML tools to predict the temperature of dairy manure during storage. The use of artificial intelligence is emerging as key to solving many complex agricultural production systems, with reported applications in analyzing and providing solutions for agricultural sustainability and food security challenges (Cravero and Sepúlveda, 2021; Priya and Ramesh, 2020; Sharma et al., 2020; Liakos et al., 2018). Other examples of ML algorithms application in agriculture include predicting the soil temperature (Sanikhani et al., 2018; Samadianfard et al., 2018), soil moisture content (Prasad et al., 2018), irrigation scheduling and management (Romero et al., 2018), weather forecasting for crop production (Saggi and Jain, 2018; Crane-Droesch, 2018), and predicting crop yield (Zhang et al., 2017). In the livestock industry, ML has been used to model or track animal behavior, animal feeding, welfare, and productivity to generate data-driven information to inform decisions to improve operational efficiency and economic well-being of
the farm (Liakos et al., 2018; Morota et al., 2018; Sharma et al., 2020.

The four ML algorithms used in this study were the gradient boosting tree, bagging tree, random forest ensembles, and neural network. The ensemble methods combine several base models to make one optimal predictive model. The ensemble methods use bagging or boosting techniques. The bagging technique (bagging trees and random forest ensembles) allows the decision trees to learn independently (i.e., in parallel) and then averages the resulting predictions to create the final model. The boosting technique (gradient boosting tree), in general, is a family of algorithms that convert weak learners (e.g., one-level decision trees) to strong learners. The method works by sequentially adding predictors to an ensemble, each one correcting its predecessor to improve the outcome (Breiman, 1996; Breiman, 2001). The gradient boosting tree generally uses gradient computations to optimize the mean square error (MSE). It is also known for its high prediction accuracy compared to other computation methods, such as linear regression and bagging tree ensembles (Saggi and Jain, 2018). The bagging tree ensemble is one of the oldest and simplest ensemble methods that use multiple subsets of the training data independently to generate individual models and aggregate all the generated models’ predictions (Breiman, 1996). The random forest ensemble uses a similar approach to the bagging tree ensemble to build and assemble a group of independently trained decision trees using a random subset of the predictors (Breiman, 2001).

Neural networks are adaptive systems that learn by using interconnected nodes or neurons in a layered structure resembling the human brain. Neural networks can be implemented as shallow or deep-layer networks. Generally, in these networks, each layer of nodes trains on a distinct set of features based on the previous layer’s output. Several types of neural networks exist, convolutional, recurrent, feedforward, and autoencoder. In this study, the feedforward neural network was used. In a feedforward neural network, information is fed forward from one layer to the next, i.e., input to output, with no feedback loops (Samadianfard et al., 2018;
Şener et al., 2018). The two-layer feedforward neural network, for example, has been used as a primary tool in weather forecasting and predicting soil temperatures (Samadianfard et al., 2018).

The outcome of our work will inform and improve the understanding of the physical and biogeochemical temperature-driven process that impacts the quality of manure during storage. Specifically, this study provides an improved method for estimating manure temperature during storage that will benefit existing decision support tools and models for assessing nutrient cycling on farms and developing new models. The results can also inform the identification and design of suitable strategies to minimize the release of gaseous or volatile compounds to the environment during manure storage, thereby contributing to attaining more sustainable manure management practices.

3.2 Materials and methods

3.2.1 On-farm manure management and storage

Two on-farm liquid dairy manure storage structures, a clay pit (CP) and concrete tank (CT), were used in this study. The farms are located 20 km apart in Franklin County, Virginia, USA (36.9459° N, 79.8297° W). Both farms use a scrape system to move manure from the barn floors to the storage structure, where it is kept for periods between three to four months before land application on field crops as a fertilizer. The top surface of the CP storage structure is oval-shaped, with dimensions of 60 and 27 m on the long and short aspects. The manure inflow and outflow locations are located on the long dimension of the structure, and the average working depth is 3.8 m. The CT storage is a partially above-ground structure with a diameter of 18.3 m and 4.6 m deep. The study period was from February 2019 to October 2020. Manure was pumped out from the: CP in April, August, and December 2019
and in March and October 2020. The CT manure was pumped out in March, October, and December 2019 and June and October 2020.

3.2.2 Manure temperature measurement and weather data collection

The manure temperature was measured at various depths and locations in the storage structure. A combination of the HOBO® TMCx-HD temperature sensors and the HOBO® UX120-006M data logger (Onset Computer Corporation, Bourne, MA, USA) mounted on poles were used for temperature acquisition. In the CP storage structure, the manure temperature sensor poles were placed near the inflow, at the mid-section, and near the outflow. The manure temperature sensors were mounted at 0.45 (MT045), 1.22 (MT122), 2.44 (MT244), and 3.7 (MT370) m from the bottom of the storage structure. Additional manure temperature sensors were added in November 2019 at depths 0.70 (MT070), 0.9 (MT090), 1.68 (MT168), 1.98 (MT198), and 2.75 (MT275) m on the pole near the inlet to capture temperature at more layers or depths of manure. In the CT, manure temperature sensors were mounted at 0.45 (MT045), 1.68 (MT168), 2.9 (MT290), and 4.11 (MT410) m from the bottom of the tank. However, in November 2019, the manure temperature sensors were reconfigured and mounted at 0.45 m (MT045), 0.90 m (MT090), 1.37 m (MT137), 1.83 m (MT183), 2.30 m (MT230), 2.75 m (MT275), 3.20 m (MT320), and 3.66 m (MT366) from the bottom of the tank. The reconfiguration was necessary to secure and prevent the mounting poles from damage and submersion resulting from mixing the manure storage content during the pump-out events. The manure temperature data were downloaded monthly using the HOBOware Pro Software. The weather parameters at each location were obtained using the DYACON® weather station (model MS-130, DYACON®, Logan, UT, USA). The weather parameters recorded were the ambient air temperature, rainfall, wind speed, wind direction, relative humidity, and solar radiation. The manure temperature and the weather parameters

63
were recorded at 30-minute intervals.

### 3.2.3 Data processing and analysis

The manure temperature and weather data collected were cleaned before analysis. The cleaning process entailed checking the data for integrity and completeness. Each row of data represented a time step containing manure temperature and the associated weather parameters. Rows with missing or invalid data were deleted. The cleaning process resulted in 28,767 and 25,516 rows of data for the CP and CT storage structures. The data analysis included visualization, statistical analysis, and fitting ML techniques to discern temporal relationships between manure temperature and weather conditions.

### 3.2.4 Data visualization and statistical analysis

The data were plotted to provide visual context and discern trends, patterns, and outliers. The JMP® Pro 14.2.0 statistical software (SAS Institute Inc., Cary, NC, 2019) and Microsoft Excel version 16.45 (Microsoft Corporation); MATLAB R2020a (MathWorks Inc., Matick, MA, USA); and Python 3 Matplotlib python library software was used for visualization. Scatter plots were used to discern trends, if any, between the ambient air temperature and the manure temperature at the various depths recorded during the storage period and time of year.

Descriptive statistics, including the maximum, minimum, mean, standard deviation (Stdev), and the coefficient of variation were calculated were computed using JMP® Pro 14.2.0 statistical software (SAS Institute Inc., Cary, NC, 2019). Additionally, a one-way analysis of variance (ANOVA) was conducted to test the effect of depth on manure temperature, and Tukey’s multiple comparisons were performed to determine differences in means. The depth effect on manure temperature was performed using data from periods when all the manure
temperature sensors were submerged or covered by manure. The level of significance was set at 0.05.

### 3.2.5 Machine learning for manure temperature predictions

The supervised learning approach was used in this study. The algorithms included three regressive ML ensemble models (gradient boosting tree, bagging tree ensemble, random forest ensemble) and two-layer feedforward neural networks with ten hidden neurons. The ML ensemble models were deployed in Python 3 scikit-learn library, and the neural network used the Deep Learning Toolbox™ in MATLAB R2020a (MathWorks Inc., Matick, MA, USA). The observations used in developing ML models were 28,767 and 25,516 for the CP and CT, respectively. The ML algorithms were trained with 67% of the associated dataset, and the remaining 33% split equally between validation and testing.

The input variables included weather parameters (ambient air temperature, wind speed, wind direction, solar radiation, relative humidity, and rainfall); time (Tm) expressed as month; depth of manure layer above a sensor (dms); and the measured manure temperature. These inputs were selected because of their known influence on manure’s heat and energy balance (Rennie et al., 2017). The derived parameter, dms, was calculated using equation 1. All the input variables were standardized (feature scaling) as part of the data preprocessing to minimize the potential bias caused by the different units and scales and improve the performance of the ML algorithms.

\[
dms = dm - ds
\]

Where dms is the depth of manure above a sensor (m), dm is the depth of manure from the bottom of the storage structure (m), and ds is the depth at which a sensor is mounted from
the bottom of the storage structure (m).

### 3.2.6 Assessing the model performance

The performance of the models was assessed using the coefficient of correlation ($R^2$), mean absolute error (MAE), root mean square error (RMSE), and the relative importance factor (Siwek and Osowski, 2016; Şener et al., 2018). The $R^2$ (equation 2) is one of the most critical metrics for evaluating the accuracy of prediction models with a magnitude ranging from 0 to 1. Values of $R^2$ close to 1, implies highly correlated parameters and vice versa. MAE (equation 3) and RMSE (equation 4) represent the error between the actual and the predicted data, and smaller values reflect better predictions. Models were judged as having a better performance based on high $R^2$ and low MAE and RMSE. The relative importance of input parameters (equation 5) describes how sensitive the model is to the input variables. It is calculated by excluding an input parameter from the dataset, then training a model with the remaining parameters. The RMSE of the resulting model was then compared to the original RMSE (when the model trained with all the input parameters). A large difference (i.e., high relative importance) between the RMSEs indicates declining model performance (Şener et al., 2018). Hence, the input parameter’s importance is ranked based on the magnitude of relative importance, with the largest being the most important.

$$R^2 = \frac{\sum_{i=1}^{N} (y_i - \bar{y})(\hat{y} - \bar{y})}{\sqrt{\sum_{i=1}^{N} (y_i - \bar{y})^2 \sum_{i=1}^{N} (\hat{y} - \bar{y})^2}}$$  (3.2)

Where $y_i$ is the measured manure temperature at the time I, $\bar{y}$ is the mean of the measured manure temperature, $\hat{y}$ is the predicted manure temperature by the model, $\bar{\hat{y}}$ is the mean of the predicted manure temperatures, and $N$ is the number of data points in the dataset.
\[ MAE = \frac{1}{N} \sum_{i=1}^{N} |y_i - \hat{y}| \]  
\[ (3.3) \]

Where \( y_i \) is the measured manure temperature at the time \( i \), \( \hat{y} \) is the predicted manure temperature by the model, and \( N \) is the number of data points in the dataset.

\[ RMSE = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y})^2} \]  
\[ (3.4) \]

Where \( y_i \) is the measured manure temperature at the time \( i \), \( \hat{y} \) is the predicted manure temperature by the model, and \( N \) is the number of data points in the dataset.

\[ RI \ of \ variable \ i \ (\%) = \frac{\Delta RMSE_i \times 100}{\sum \Delta RMSE} \]  
\[ (3.5) \]

Where \( \Delta RMSE_i \) is the difference between the RMSE of the model trained with all the input variables and the RMSE of the model without the input variable \( i \).

### 3.2.7 Scenario Analysis

A scenario analysis was conducted using the models developed from the ML algorithms in this study. The analysis was done to assess how the models predicted manure temperature at 0.45 m from the bottom of the CP and CT using historical weather data compared to the measured manure temperatures. The average daily historical weather data for the 30 years (1990 to 2020) from Roanoke Regional Airport was used (https://www.ncdc.noaa.gov/). The Roanoke Regional Airport is the National Oceanic and Atmospheric Administration’s weather station because it is the closest to the manure storage sites used in this study. The variable \( dm \) was calculated using the average daily filling rate of each storage structure calculated during the experimental period and assuming that the manure storages were
pumped out or emptied once the storage’s maximum depth was attained. The results were summarized and presented as monthly averages for a calendar year, and comparisons were made with the measured manure temperatures.

3.3 Results and Discussion

3.3.1 Manure temperature and weather parameters during the storage period

A summary of the descriptive statistics, maximum, minimum, mean, standard deviation, and coefficient of variation of the weather and manure parameters used as input variables for the CP and CT storage structures are presented in Tables 3-1 and 3-2, respectively. The average values of most of the measured weather parameters were comparable at the two farms during the study period. The average ambient air temperature was about 17 ºC and ranged from -10 to 36 ºC, at both farms. The direction of the wind was mostly from the southwest at both locations. The average wind speed at the CT ran 60% of what was recorded at the CP site. Because of the generally higher wind speeds at the CP site, the site may be more prone to material loss via volatilization than the CT site. The solar radiation was within the same range and had a coefficient of variation >100% due to its high variability at both locations. However, the rainfall quantities were very different at both storage structures; the average rainfall at the CT was about 211% of the CP. The average dm of manure was about 65% of the working depth (3.7 m for the CP and 4.0 m for the CT). The average manure temperature was about 20 (± 6) ºC. The manure temperature was generally greater than the ambient air temperature. This result indicates that the manure temperature approximations are not captured accurately in the decision support tools for nutrient accounting and the models for predicting gaseous emissions from manure storages (e.g., Chianese et al., 2009;
3.3.2 Comparing ambient air temperature to manure temperatures

The manure and ambient air temperatures are presented in Figures 3-1 and 3-2 for the CP and CT storage structures. Breaks in the data were the periods when the sensors were removed for reconfiguration. The vertical dashed lines (i.e., P1, P2, P3, P4, and P5) indicate when manure was pumped out of the storage structure. In general, the manure temperature at all depths followed a similar annual sinusoidal trend as the ambient air temperature and solar radiation. The ambient air temperature and manure temperature trends lagged the solar radiation (not shown), a normal phenomenon for systems that store matter with resistance to flow (Campbell and Norman, 1998). The peak solar radiation occurred between mid-May and mid-June, while the manure temperature peak occurred in early July. The fluctuations in the daily ambient air temperatures were greater than the manure temperature.

Table 3.1: The descriptive statistics of the weather and manure parameters at the clay pit (CP) manure storage.

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Parameter</th>
<th>Weather</th>
<th>Manure</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AAT (°C)</td>
<td>WS (m s⁻¹)</td>
<td>WD (degree)</td>
</tr>
<tr>
<td>Maximum</td>
<td>35.90</td>
<td>8</td>
<td>359.90</td>
</tr>
<tr>
<td>Minimum</td>
<td>-10.30</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Mean</td>
<td>17.42</td>
<td>1.26</td>
<td>194.17</td>
</tr>
<tr>
<td>stdev</td>
<td>±8.58</td>
<td>±1.09</td>
<td>±100.17</td>
</tr>
<tr>
<td>CV (%)</td>
<td>49</td>
<td>86</td>
<td>52</td>
</tr>
</tbody>
</table>

AAT - ambient air temperature; WS - wind speed; WD - wind direction; SR - solar radiation; RH - relative humidity; RF - Rainfall; dm - manure depth from the bottom of the pit; MT - manure temperature; stdev - standard deviation; CV - the coefficient of variation.
Table 3.2: The descriptive statistics of the weather and manure parameters at the concrete tank (CT) storage structure

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Parameter</th>
<th>Weather</th>
<th>Manure</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AAT (°C)</td>
<td>WS (m s⁻¹)</td>
<td>WD (degree)</td>
</tr>
<tr>
<td>Max</td>
<td>36.50</td>
<td>9.80</td>
<td>359.9</td>
</tr>
<tr>
<td>Min</td>
<td>-10.1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Mean</td>
<td>16.26</td>
<td>0.75</td>
<td>231.5</td>
</tr>
<tr>
<td>stdev</td>
<td>±8.97</td>
<td>±1.04</td>
<td>±110.4</td>
</tr>
<tr>
<td>CV (%)</td>
<td>55</td>
<td>139</td>
<td>48</td>
</tr>
</tbody>
</table>

AAT - ambient air temperature; WS - wind speed; WD - wind direction; SR - solar radiation; RH - relative humidity; RF - Rainfall; dm - manure depth from the bottom of the pit; MT - manure temperature; stdev - standard deviation; CV - the coefficient of variation.
Figure 3.1: The ambient air temperature (AAT) and manure temperature (MT) at various depths in the clay pit (CP) during the study period. (045, 090, 122, 198, and 244 refer to placement depth (cm) from the bottom of the storage structure of the MT sensor; P1 to P5 were the periods when manure was pumped out of the storage structure).
Figure 3.2: The ambient air temperature (AAT) and manure temperature (MT) at various depths in the concrete tank (CT) during the study period. (045, 090, 137, 275, and 290 refer to placement depth (cm) from bottom of the storage structure of MT sensor; P1 to P5 were the periods when manure was pumped out of the storage structure).
The average daily manure temperature in both storages was higher than the average daily ambient air temperature for most of the year, with the highest difference between August and December (i.e., 2.4 and 4.4 °C, in the CP and CT, respectively). These results are similar to Rennie et al. (2017) and Baldé et al. (2016), who reported stored manure temperatures being 3 to 8 °C higher than the ambient air temperature, depending on the time of year. The similarity of manure temperature trends at the two locations in this study is not surprising and could be attributed to their comparable weather parameters due to their locational proximity. The different durations between pump-out times in Figures 3-1 and 3-2 reflected the manure management at the two farms during storage when partial pump-outs were conducted because weather conditions were unsuitable for the manure’s land application. Just like was observed by Rennie et al. (2017), Baldé et al. (2016), and Pattey et al. (2005), these results imply that using ambient air temperature as a surrogate for manure temperature in models may result in inaccurate results.

Before being submerged, the manure temperature sensors recorded the headspace air temperature above the surface of stored manure. Because the headspaces are open to the atmosphere, one would expect the air temperature to be similar to the ambient air. However, this was not the case. The daily air temperature above the manure surface was higher than the ambient air temperature for most of the year. The difference was especially pronounced during the year’s warmer periods, with magnitudes reaching 15.82 and 29.57 °C in the CP and CT, respectively. This difference could be due to the heat released by the manure’s microbial activities in storage and or reflecting the heat from the surface (Campbell and Norman, 1998). As the manure surface approached a manure temperature sensor, the temperature patterns recorded by the sensor drifted from following the ambient air temperature patterns to following the temperatures recorded by the submerged manure temperature sensors. This behavior pattern implies that few days before a manure temperature sensor is
submerged, the air temperature closer to the manure surface increased gradually until the
sensor got submerged and then tended to the manure temperature value. This phenomenon
was observed during the warm and cold periods of the year. A possible explanation is that
since the manure is warmer than the ambient air temperature, the air directly above the ma-
nure surface gains heat through convection and becomes warmer. This temperature increase
phenomenon was subtle during warm seasons but very steep during cold seasons when the
temperature difference between the ambient air and the manure was much higher.

3.3.3 Influence of depth on manure temperature

Depth (from the bottom of the storage structure) at which the manure temperature sensor
was placed had a significant effect on manure temperature (p<0.05) in both the CP and CT.
Figure 3-3 shows the temperature variability at four depths (MT045, MT090, MT168, and
MT244) and the ambient air temperature at the CP location for March 2020, June 2020, and
September 2020. During these periods, all the sensors were covered by manure. The average
ambient air temperature was significantly different from the manure temperature at all the
depths. The average ambient air temperature was lower than the manure temperature in
March and September but higher in June. In general, the manure temperature at the bottom
layer (0.45 m) was significantly different from the top layers (1.68 m in March and 2.44 m in
June and September). During the periods in March and September, the layers of manure in
the bottom region of the storage structure were higher than the top layers, and the reverse
was true for June (Figure 3-3). Mixed results were observed at the CT location (Figure
3-4). The average ambient air temperature was similar to the average manure temperatures
(MT045, MT090, MT137, MT275) in March 2020 but significantly different in July 2020 and
September 2020. The manure temperatures were very close in magnitudes. The temperatures
of manure at 0.45 and 0.90 m were higher than 137 and 275 in September and vice versa
during July.

Based on the periods between the pump-out events, i.e., P1 (January to April), P2 (May to August), and P3 (September to December), the largest difference between the average manure temperature and the ambient air temperature was recorded during P3 and the least, P2. The average manure temperature of the manure in the top layers was higher than at the bottom layers during P2 and vice versa during P1. These results imply manure temperature inversion occurs during storage, i.e., the top layers of the manure have higher temperatures in the warm periods compared to the bottom layers; and vice versa during the cooler weather, similar to what was reported by Karunarathne et al. (2020) and Rennie et al. (2017). Another general observation was that the variability of manure temperatures in the bottom layers was much less than those near the top surface. The temperature of manure close to the surface closely followed the ambient air temperature, exhibiting larger variability than the temperature of manure in layers further away (Johannesson et al., 2018; Campbell and Norman, 1998). The significant differences in manure temperature by depth suggest the need to consider temperature by depth as input parameters to improve manure nutrient accounting models or tools for estimating emissions from stored manure.

3.3.4 Predicting manure temperature with machine learning algorithms

The performance of the machine learning algorithms in predicting manure temperatures is summarized in Table 3-3. The Ensemble models (gradient boosting tree, bagging tree ensemble, and random forest ensemble) had faster computation speeds than the neural network. Based on the established ranking criteria ($R^2$, MAE, MSE, and RMSE) for this study, the random forest and bagging tree ensembles were the best performers. The magnitudes of the models’ performance parameters were very close for the CP and CT storage structures. The bagging tree ensemble and random forest ensemble models performed slightly better with
the CT than the CP.

The linear relationships between the actual and predicted manure temperature by the four models are shown in Figure 3-5. The results show good agreements between the actual and predicted manure temperature, indicating the models fit the datasets well. The results also indicate that machine learning algorithms can predict stored manure’s temperature by depth. The similarity in bagging trees and random forest ensemble performances was not surprising. The two methods use the same approach in their regression analysis. The predicted manure temperatures were greater than the ambient air temperature, similar to the results obtained with the measured manure temperatures. The largest difference between the predicted manure and ambient air temperatures was about 10 °C occurring before the P3 period. The results also suggest that the bagged trees and random forest ensemble models predicted manure temperature better than gradient boosted trees and neural networks soon after manure pump-out events. During these times, the magnitudes of dms were small, and only the sensors closer to the bottom (MT045 and MT090) were covered by manure. The bagging tree ensemble and random forest ensemble models had smaller differences between measured and predicted manure temperatures than gradient boosting and neural network models. The manure temperature predictions were better as the dms increased. The models’ performances were almost perfect when dms were greater than 3 m in the CP and 1.5 m in the CT.
Figure 3.3: Manure temperature in the clay pit (a) March 2020 (DOY: 53 to 64); (b) June 2020 (DOY: 169 to 182); and (c) September 2020 (DOY: 245 to 259). AAT - ambient air temperature; MT - manure temperature; 045, 090, 168, 244 - sensor placement depth (cm) from the bottom of the pit.
Figure 3.4: Manure temperatures in the concrete tank (a) March 2020 (DOY: 80 to 91) (b) July 2020 (DOY: 199 to 213); (c) September 2020 (DOY: 245 to 273). AAT - ambient air temperature; MT – manure temperature; 045, 090, 137, 275 – sensor placement depth (cm) from the bottom of the pit.
Table 3.3: The performance parameters for machine and neural network models trained and tested with 67% and 33% of data, respectively.

<table>
<thead>
<tr>
<th>Model</th>
<th>Storage</th>
<th>MAE</th>
<th>MSE</th>
<th>$R^2$</th>
<th>RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>GBT</td>
<td>CP</td>
<td>0.551</td>
<td>0.657</td>
<td>0.978</td>
<td>0.811</td>
</tr>
<tr>
<td>BTE</td>
<td>CP</td>
<td>0.214</td>
<td>0.300</td>
<td>0.990</td>
<td>0.547</td>
</tr>
<tr>
<td>RFE</td>
<td>CP</td>
<td>0.210</td>
<td>0.290</td>
<td>0.990</td>
<td>0.538</td>
</tr>
<tr>
<td>NN</td>
<td>CP</td>
<td>-</td>
<td>0.805</td>
<td>0.974</td>
<td>0.897</td>
</tr>
<tr>
<td>GBT</td>
<td>CT</td>
<td>0.681</td>
<td>0.987</td>
<td>0.977</td>
<td>0.994</td>
</tr>
<tr>
<td>BTE</td>
<td>CT</td>
<td>0.098</td>
<td>0.096</td>
<td>0.998</td>
<td>0.310</td>
</tr>
<tr>
<td>RFE</td>
<td>CT</td>
<td>0.092</td>
<td>0.082</td>
<td>0.998</td>
<td>0.286</td>
</tr>
<tr>
<td>NN</td>
<td>CT</td>
<td>-</td>
<td>1.329</td>
<td>0.970</td>
<td>1.153</td>
</tr>
</tbody>
</table>

CP – clay pit; CT - concrete tank; MAE - mean absolute error; MSE - mean square error; RMSE - root mean square error; $R^2$ - coefficient of correlation, GBT – gradient boosting tree, BTE – bagging tree ensemble, RFE – random forest ensemble, NN – neural network.

The effect of depth effect could be attributed to the fact that as the manure volume in storage increases, the manure temperature variability at the bottom layers decreases and is not affected much by the surrounding weather events (Campbell and Norman, 1998). The slow change in manure temperature in the bottom layers reflects the good insulation properties of manure. Thus, the manure temperature at lower depths or layers is more stable and perhaps easier to predict than the layers close to the manure surface. For example, in the CP, the $R^2$ for the predicted MT045 using the random forest algorithm is 0.993 compared to 0.894 for the predicted MT370. This result is similar to Rennie et al. (2017), who observed improved model predictions when the manure depth was greater than 1.0 m.
Figure 3.5: The predicted and actual manure temperature (a) clay pit and (b) concrete tank storage structures. GBT-Gradient Boosted trees; BTE - Bagged trees ensembles; RFE- Random Forests ensembles; NN - neural networks.
The relative importance of the input parameters for the models is presented in Table 3-4. The Tm had the largest relative importance, an expected result. One function of Tm is to capture the periodicity of manure temperature during the year by associating the measured manure temperature to the time (month) of the year. The Tm enabled the models to differentiate between the warm and cold periods, for example, when dealing with unusual weather events such as a hot day in winter or a cold day in summer, to minimize predicting unrealistic temperatures of the stored manure. Although these unusual weather events happen in real life, their impact on changing the manure temperature is slow. Typically, because of the stored heat energy in manure mass, the rate of change of manure temperature is slow and does not instantly react to unusual weather events.

The dms had the second-largest relative importance. While the effect of dms on the manure temperature has been presented in section 3.3 above, the high relative importance is not surprising. The dms indirectly indicate the mass or volume of manure above a sensor. The larger the dms, the larger the mass or volume of manure and the potential to store more energy. Thus, it is expected that dms are an indicator of activities associated with the system’s energy changes and contribute to manure temperature changes. This result is consistent with Rennie et al. (2017) finding that manure depth was one of the most influential factors in their model for predicting manure temperature. The ambient air temperature had the highest relative importance of the weather parameters, perhaps because the ambient air is one of the main mediums through which heat or energy is exchanged with the stored manure. The difference between the relative importance of the input parameters in the CP and the CT could be attributed to the weight assigned by each ML algorithm to the input parameter and how each parameter affects the model’s prediction ability for each structure. It is also possible that the difference was caused by the way manure was managed. The CT experienced more disruptions, i.e., removing small volumes of manure, especially when
the storage capacity was running out to create storage space to accommodate more manure before the main pump-out events. There were minimal disturbances at the CP.

Table 3.4: Relative Importance for the input parameters for the models generated for the clay pit and concrete tank manure storage structure.

<table>
<thead>
<tr>
<th>Variable (i)</th>
<th>Clay pit</th>
<th>Concrete tank</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RMSE</td>
<td>(RMSE_i)</td>
</tr>
<tr>
<td>Base</td>
<td>0.538</td>
<td>-</td>
</tr>
<tr>
<td>Tm</td>
<td>1.111</td>
<td>0.573</td>
</tr>
<tr>
<td>dms</td>
<td>0.788</td>
<td>0.250</td>
</tr>
<tr>
<td>AAT</td>
<td>0.564</td>
<td>0.026</td>
</tr>
<tr>
<td>WS</td>
<td>0.539</td>
<td>0.001</td>
</tr>
<tr>
<td>WD</td>
<td>0.539</td>
<td>0.000</td>
</tr>
<tr>
<td>SR</td>
<td>0.544</td>
<td>0.006</td>
</tr>
<tr>
<td>RH</td>
<td>0.546</td>
<td>0.007</td>
</tr>
<tr>
<td>RF</td>
<td>0.541</td>
<td>0.003</td>
</tr>
<tr>
<td>Summation</td>
<td>-</td>
<td>0.866</td>
</tr>
</tbody>
</table>

RI – relative importance; Base – all variables; Tm - time as month; RMSE - root mean square error; \(RMSE_i\) - the difference between RMSE of the model trained with all the variables and the model trained with all the variables except variable (i); dms- the depth of the manure covering the sensor; AAT - ambient air temperature; WS - wind speed; WD - wind direction; SR - solar radiation; RH - relative humidity; RF - Rainfall; MT - manure temperature.

### 3.3.5 Scenario analysis and availability of models for use

The magnitudes and patterns of the ambient air temperature at both sites were quite similar to the ambient air temperature from the regional weather station (Figure 3-6 a). The scenario analysis showed that historical weather data could suitably predict daily manure temperature using ML algorithms, especially the ensembles. The neural networks model underperformed and had large variabilities in its MT045 predictions. The neural network predictions suggest that the manure temperature at 0.45 cm, on average, does not change much throughout the year, a result that is not consistent with field measurements or observations. This result may suggest the unsuitability of using neural networks for predicting manure temperatures.
during storage. The ensemble ML model predictions (at both manure storage sites – Figure 3-6 b and 3-6 c) were quite similar in magnitude and trend to the average monthly measured MT045 during the study period. The predictions by the bagged trees and random forest, the models that returned the best performance during development, produced almost identical results and predicted MT045 in the CT almost perfectly from March through September (Figure 3-6 b).

On the other hand, the gradient boosted trees model at the CP made predictions closer to the measured MT045 than the bagging tree ensemble and random forest ensemble (Figure 3-6 c). Thus, this study’s results provide a better and more practical method to estimate the manure temperature to be used as an input parameter instead of the temperature of the ambient air surrogates in decision support tools and nutrient cycling models and (e.g., Mangino et al., 2001; Chianese et al., 2009; Li et al., 2012). Finally, the figures confirm earlier observations of (i) lower variations in manure temperature compared to ambient air temperature and (ii) the temperature of the manure and ambient air following the same trend over the year.

A key challenge to the ML models developed in this study that needs further investigation is their universal use. Although the models are not based on classical equations that need specific weather parameters, coefficients, manure properties, and manure management types, the integrity of their use may be limited. We are confident of the models’ use and performance in areas with similar weather patterns (i.e., winter, spring, summer, and fall) as the study area and storage structures. For these areas, the models associate the month of January (Tm = 1) with low manure temperatures; however, this might not be the case in other regions, for example, countries in the southern hemisphere or between the tropics. This study showed that the machine learning approach could predict manure temperature and validation with data from other regions to ascertain the universality of use.
Figure 3.6: The average monthly (a) historical and measured ambient air temperature and predicted manure temperature at the 0.45 m depth, predicted by the machine learning algorithms using historical weather data and measured manure temperature during the study period at the clay pit (b) and concrete tank (c). AAT – ambient air temperature; GBT- gradient boosted trees; BTE - bagged trees ensembles; RFE- random Forests ensembles; NN - neural networks; CP - clay pit; CT - concrete tank.
Another Challenge of using ML is the relatively large datasets needed for development. This amount of data may be hard to obtain and costly to generate. We have ongoing work that we hope to report in the future to supplement the contribution of this study. We will create and publish a web-based user interface to implement machine learning models generated in this study for farmers, engineers, scientists, and other practitioners to estimate manure temperature if it fits their needs.

3.4 Conclusion

Temperature affects biological, physical, and biogeochemical reactions in manure during storage. Thus, a better way to estimate manure temperatures used as inputs for the decision-support tools, models, and life cycle assessments related to on-farm manure nutrients is necessary. In this study, machine learning algorithms were developed as a potential substitute for ambient air temperature typically used as a surrogate for manure temperature in existing on-farm nutrient accounting tools. This study found that:

- The manure temperature follows a diurnal or sinusoidal pattern during storage, with the peak occurring during the warm seasons. However, the manure temperature always lags the ambient air temperature and solar radiation, the external heat energy sources.

- On average, the average manure temperature is higher than the average ambient temperature. The manure temperature was three degrees Celsius higher than the average ambient air temperature.

- The depth of storage matters. The manure temperature at the bottom layers was significantly different from the top layers. The changes in temperature of the manure in the top layers were impacted more by the ambient air temperature. Additionally, the increasing
depth of stored manure increases the dampening of temperature fluctuations at the bottom layers.

- The air temperature immediately above the stored manure surface is higher than the general ambient air temperature. Perhaps, because of gaining heat from the manure through convection and reflection of heat from solar radiation. This phenomenon occurred for most of the year, with the highest temperature difference during warm periods.

- Machine learning algorithms provided a realistic prediction of stored manure’s temperature over different seasons of the year. The performance $R^2$ values ranged from 0.970 to 0.998. The ensembles, random forest, bagging trees, and boosted trees fit the data better than the neural network algorithms used in this study.

- Historical weather data could be used to predict manure temperature if location-based weather data is unavailable. The ensemble machine learning algorithms predicted the manure temperature at 0.45 m from the bottom of the storage more accurately than the neural networks with historical weather data.

References


Chapter 4


Attribution

This chapter is based on the following manuscript:

Abstract

This study presents a physics-informed neural networks (PINN) approach as a practical application of neural computing to predict manure temperature at a dairy farm during storage. Manure temperature is an important factor impacting the microbial and chemical processes associated with releasing aerial pollutants. Also, manure temperature is a pertinent input parameter for on-farm decision support tools and nutrient accounting models; however, there is no standard method to estimate it. Currently, decision support tools use surrogates derived from various ambient air temperature averages instead of the manure temperature, which underestimates the contaminants lost to the atmosphere during the manure storage period. In this study, we compare the performance of the PINN model in predicting the stored manure temperature to three other models (finite-elements heat transfer, classical data-driven neural network, and simulation-based neural network). The models were trained and validated using data collected from a concrete storage structure in a scraped farm, then tested on data collected from a clay-lined pit on a scraped farm and a concrete tank on a flushed farm. The PINN model results were less biased and more data-efficient than the other models. Even though the performance of the PINN and the classical data-driven neural network model were comparable during the testing and validation phases ($R^2 > 0.9$), the PINN model had superior generalization accuracy. The $R^2$ for the PINN model during the testing phase exceeded 0.7, while it ranged between -0.03 and 0.66 for the finite-elements heat transfer, classical data-driven neural network, and simulation-based neural network models. Finally, the manure temperature predicted by the PINN model was the closest in magnitude to the measured temperatures at the three manure storages monitored. These results suggest that using PINN-based manure temperature predictions in decision support tools and nutrient cycling models would provide more realistic outcomes for assessing sustainable manure management practices. The outcomes of this study contribute to the
field of precision agriculture, specifically designing suitable on-farm strategies to minimize nutrient loss and greenhouse gas emissions during the manure storage periods and improve the accuracy of metrics used to assess sustainable manure management practices.
Graphical abstract

The graphical abstract for this study is displayed in this section.

**Finite-element heat transfer model:** The observed ambient air temperature and manure depth

\[ \frac{\partial}{\partial z} \left( k \frac{\partial T_m}{\partial z} \right) + q = \rho C_p \frac{\partial T_m}{\partial t} \rightarrow T_m \]

1. Input features (X)
   - The ambient weather parameters, time (month), and depth of manure layer above a sensor.

2. The observed manure temperature (y)

3. The predicted manure temperature by the finite-element heat transfer model (T_m)

**Loss functions**

- Data-driven neural networks
  \[ L_d(\theta) = \frac{1}{N} \sum_{x \in \mathcal{D}} (\hat{y}(x; \theta) - y(x))^2 \]

- Neural network trained on the simulation data
  \[ L_s(\theta) = \frac{1}{N} \sum_{x \in \mathcal{S}} (\hat{y}(x; \theta) - T_m(x; p))^2 \]

- Physics-informed neural networks
  \[ L_p(\theta) = \lambda L_d(\theta) + (1 - \lambda) L_s(\theta) \]

**Training and validation**

- Test 1
- Test 2

-Manure temperature predictions
4.1 Introduction

Storing manure is an essential activity on dairy farms and is usually practiced as a best management practice for handling manure nutrients. Typically, storing manure presents opportunities for farms to (i) use manure nutrients as fertilizer for crop and pasture production at the right time, (ii) decrease handling costs, and (iii) minimize the potential of the manure to pollute the environment (Rotz, 2004). However, manure, biological material, is subjected to a series of microbial, physical, and biogeochemical processes that affect its quality and composition during storage (Manyi-Loh et al., 2016; Nag et al., 2019). Temperature is central to these physical, biogeochemical, and microbial activities related to the degradation processes that lead to the loss of quality and value of manure (Grady Jr. et al., 2011). Higher manure temperatures (within the range that microorganisms thrive) favor increased microbial activities and rates of chemical reactions. For example, the reaction rates can double for every ten-degree centigrade increase in temperature, leading to higher loss rates of gaseous and volatile compounds from stored manure that are known to pollute the environment (Campbell & Norman, 1998; Leytem et al., 2017; Massé et al., 2003). Hence, knowledge of the temperature and how it impacts manure during storage on a dairy farm is essential to inform the understanding and decision-making on selecting technologies and making policies to mitigate atmospheric pollutants associated with dairy manure.

Estimating manure temperature is common in decision support tools or models in precision agriculture to assess the sustainability of food and agricultural production systems. However, limited information exists in the literature for predicting manure temperature during storage, especially using field-measured data. Thus, the current approaches for estimating manure temperature in decision-support tools and nutrient-accounting models use various forms of averaged ambient air temperature as a surrogate for manure temperature. For instance, [8] the Intergovernmental Panel on Climate Change model uses monthly average
air temperatures to estimate the methane emissions from anaerobic lagoons (Mangino et al., n.d.), and the Integrated Farm Systems model assumes manure temperature equals the average air temperature of the previous ten days (Chianese et al., 2009). However, studies that have measured ambient and manure temperature concurrently show that manure temperature runs higher than the average ambient temperature for most of the year and, in some instances, by up to 10 °C (Baldé et al., 2016; R. A. Genedy & Ogejo, 2021; R. Genedy & Ogejo, 2020; Masse et al., 2008; Rennie et al., 2017). Thus, using ambient air temperature as a surrogate for manure temperature is inadequate, suggesting the need for better methods to predict manure temperature accurately.

Our literature search revealed limited research on predicting manure temperature during storage, the most comprehensive being physics-based and machine-learning models (R. A. Genedy & Ogejo, 2021; Rennie et al., 2017). The most straightforward approaches to estimating the manure temperature are represented by empirical equations based on the correlation between the ambient air and manure temperatures. For example, Li et al. (Li et al., 2012) use a simplified heat transfer formula to estimate the manure temperature using the air temperature. Physics-based modeling is a more complicated and realistic approach to estimating manure temperature. A physics-based model is a representation of the governing laws of nature, usually expressed through empirical expressions or sophisticated numerical or visualization tools (Willcox et al., 2021). Rennie et al. (Rennie et al., 2017) combine field data and a 3-D finite-element heat transfer model to estimate manure temperature. Their model affirmed the power of coupling heat transfer principles and finite element modeling to estimate manure temperatures. The average manure temperature predicted by the model was similar to the observed temperature. Also, Genedy et al. [17] developed a one-dimensional finite element heat transfer model using COMSOL to predict the temperature of the stored manure. Similar to Rennie et al. [13], they found that the finite element model
captured the manure temperature pattern over time.

However, the finite element-based models did not adequately predict the magnitude of the manure temperature. For example, Rennie et al. (Rennie et al., 2017) found that the largest temperature discrepancy occurred during the manure agitation and removal period. This period is critically important since most of the nutrient losses arise during that period (R. Genedy & Ogejo, 2022). Further, Genedy et al. (R. A. Genedy et al., 2021) found that their model tends to predict lower manure temperatures than observed during warm periods, hence, is prone to underestimate the nutrient losses and greenhouse gas emissions from the stored manure (Baldé et al., 2016; R. A. Genedy et al., 2021). Some challenges in using classical heat transfer equations to model manure storage systems are accounting for crusting or snow cover on the manure surface and temperature stratification. Other shortcomings of the physics-based models include accounting for the constantly changing volume of manure due to manure addition or pumping out, different management practices on farms, and the availability of meaningful manure heat and mass transfer coefficients (Masse et al., 2008; Rennie et al., 2017).

Machine learning algorithms have proven their ability to solve similar challenges in complex systems, including predicting soil temperature (Samadianfard et al., 2018; Sanikhani et al., 2018), predicting air temperature and relative humidity in duck housing (Lee et al., 2022), predicting biochar yield from manure (Cao et al., 2016), and predicting fruits’ chemical attributes (Al-Saif et al., 2022). Further, our preceding work demonstrated the potential of traditional machine learning approaches to accurately predict manure temperature [14, 16]. Because conventional machine learning methods are solely data-dependent, we will refer to such methods as data-driven machine learning methods. Machine learning algorithms present and enable new ways to work with complex systems which do not have existing formulas, equations, or written rules, tasks that are constantly changing, requiring the model to adapt.
Our previous work shows the successful use of decision trees, bagged ensembles, boosted ensembles, and neural network models to predict the stored manure temperature using weather parameters and manure depth (R. A. Genedy et al., 2021; R. A. Genedy & Ogejo, 2021; R. Genedy & Ogejo, 2020). The results of the work showed that compared to finite element heat transfer modeling, the data-driven machine learning models improved the manure temperature predictions by approximately 20% (R. A. Genedy et al., 2021).

Although successful, exclusively data-driven tools are prone to overfit training data and sometimes fail to discern the underlying relationships between the features and the observed training data, hindering the model’s knowledge extraction (Isaac Abiodun et al., 2018). These models lack robustness and accuracy when tested on unobserved datasets, i.e., poor generalization (Isaac Abiodun et al., 2018; Pang et al., 2020; Sawant et al., 2021; Yang & Perdikaris, 2019). For example, we found that the predictive ability of the data-driven models deteriorated when tested using data collected from different manure management systems. Additionally, as the complexity of machine learning models increase, the computation time, the memory footprint, and the power consumption increase. Lastly, data-driven algorithms require a large amount of data for adequate training, and obtaining the needed data can be cost-prohibitive for manure storage applications (O. I. Abiodun et al., 2019).

This study applies a practical neural computing technique, physics-informed neural networks (PINN), to mitigate the limitations of the physics-based and machine learning-based approaches in predicting the stored manure temperature. Briefly, PINN uses universal function approximators that embed known physical, chemical, and biological relationships governing the dynamics of a system in the learning process (Raissi et al., 2017). Unlike the classical neural networks, which are purely data-driven and do not include physical models in their predictions. Including the known laws governing the system dynamics to train the
neural networks regularizes and restricts the permissible solutions space, thus increasing the approximation’s accuracy (Lawal et al., 2022; Willcox et al., 2021). Another benefit of encoding structured information into a learning algorithm is the amplification of the content of the information in the data seen by the algorithm, enabling it to steer itself toward accurate solutions and generalize quickly (He et al., 2020; Lawal et al., 2022; Shen et al., 2021). Finally, the PINN approach reduces overfitting data and improves the robustness of the model by adding a physics-informed regularization term to the loss function. Successful applications of PINNs include solving nonlinear differential equations (Patel et al., 2020; Raissi et al., 2017, 2019) and computational fluid dynamics (Kadeethum et al., 2020; Wang et al., 2021). Other successful applications of PINNs include fault detection in internal combustion engines (Shen et al., 2021), estimating the traffic state on roads (Shi et al., 2021), estimating subsurface transport parameters [33, 40], and cardiac activation mapping (Sahli Costabal et al., 2020).

Although the physics-based and machine learning-driven models reported success in predicting the manure’s temperature, especially for the same manure storage structure used for training, the models lack generalization. As a result, the models tend to underperform when used on manure storage structures with different handling types or management practices (i.e., lack generalization accuracy). Thus, there is a need for a universal model to predict manure temperature regardless of manure management practices. The work we describe here responds to the need, with a novelty rooted in exploiting the power of physics-based modeling with machine learning to develop a unified and generalized model as an alternative for predicting the temperatures under different manure management conditions.

The outputs of the PINN model are compared to the observed data, and the outputs of three other models the finite-elements heat transfer (FEHT), data-driven neural networks (DDNN), and a neural network trained by the data simulated by the FEHT model (SSNN). The FEHT
and DDNN models were included in the study because they have been successfully used in the literature to predict manure temperature during storage (R. A. Genedy et al., 2021; R. A. Genedy & Ogejo, 2021; Rennie et al., 2017). The SSNN model was included in the study because it is conceptually similar to the PINN model. The PINN and SSNN models use the physics of the underlying mechanisms of a process to train the neural network but differ in their implementation. The study’s outcome will improve the accuracy of predicting the stored manure temperature used in on-farm decision support tools. Further, the results of our work will contribute to designing suitable on-farm strategies to minimize nutrient loss and greenhouse gas emissions during the manure storage periods and improve the accuracy of metrics used to assess sustainable manure management practices.

4.2 Data

4.2.1 Manure Storage Description and Data Collection and Preparation

Data from three on-farm liquid dairy manure storage structures in Virginia, U.S., were used in this study. The manure management practices included scraping on two farms and flushing on one farm to move manure from the barns to the storage pit. The storage structure types included a clay pit receiving scraped manure (SCP), a concrete tank receiving scraped manure (SCT), and a concrete tank receiving flushed manure (FCT). The dry matter content of scraped manure was between 4 to 9%, while flushed manure was less than 2%. The manure temperature and the weather data were recorded at 30 min intervals. The experimental data from the SCP and SCT cover the period from February 2019 to August 2021 and the FCT from November 2019 to August 2021.

The manure temperature was measured using HOBO® TMCx-HD temperature sensors connected to the HOBO® UX120-006M data logger (Onset Computer Corporation, Bourne,
MA, USA). The temperature sensors were mounted on poles at predetermined depths and locations in each storage structure. The weather parameters were obtained at each farm using the DYACON® weather station (model MS-130, DYACON®, Logan, UT, USA). The weather parameters recorded included the ambient air temperature, rainfall, wind speed, wind direction, relative humidity, and solar radiation.

4.2.2 Data Processing and Features Selection

The collected data underwent a quality check for completeness and integrity before use. Each row contained the manure temperature and all the associated ambient weather parameters and represented a data point taken at each time step. Cleaning the data entailed inspection and deleting rows with missing or invalid data. The final dataset had approximately 70,000, 78,000, and 10,000 data points for the SCT, SCP, and FCT, respectively. The data was split for training, validation, and testing of the neural network models. The neural network models were trained using 67% of the data collected from the SCT, and the remaining 33% was used for validation. The network was tested twice, Test 1 using the data collected from the SCP and Test 2 using the data collected from the FCT.

Feature selection is a data preparation technique to characterize the most relevant, pertinent, and significant feature space (E. O. Abiodun et al., 2021). It entails selecting a subset of the relevant features for predictive modeling. The primary purpose of feature selection is dimensionality reduction. Reducing the dimensions of the input features increases the model’s accuracy and helps the model detect the main hidden intricacies that can improve the model’s performance (E. O. Abiodun et al., 2021). In this work, we used a filter-based feature selection method to select the input features for the neural network models (DDNN, SSNN, and PINN). We started by calculating the correlation coefficient between the features and the target value and selected the features with a correlation coefficient > 25%. Then we tested
for interdependency among the selected features (Guyon & De, 2003). We performed feature selections on the following: the ambient air temperature, wind speed, wind direction, solar radiation, relative humidity, rainfall, time (represented by month), and depth of manure layer above a sensor (equation 1). These features were selected because of their known influence on the manure heat and energy balance (Rennie et al., 2017). Finally, input features scaling was completed as part of the data processing to prevent potential bias due to the different units and scales of the parameters.

\[ d_{ms} = d_m - d_s \]  

(4.1)

Where \( d_{ms} \) is the depth of the manure layer above the sensor, \( d_m \) is the depth of the manure, and \( d_s \) is the depth at which the sensor was mounted.

### 4.2.3 Description of Models

We developed four models (FEHT, DDNN, SSNN, and PINN) to estimate the temperature of the stored manure in this study. The FEHT is a physics-based model developed using energy balance and heat transfer principles. The other three (DDNN, SSNN, and PINN) models were developed using neural network algorithms. The model details are presented below. All the models were deployed in Python using TensorFlow.

**Finite-elements heat transfer model (FEHT)**

We used a one-dimensional finite element heat transfer model with flux moving vertically, i.e., along the depth of stored manure. The findings of (Karunarathne et al., 2020; Rennie et al., 2017) were the basis of selecting a one-dimension model. They reported manure temperature during storage occurring more in the vertical (z-direction) than in the horizontal (x and y
directions). The time step in the model’s calculations was similar to the manure temperature and weather parameters recording intervals (i.e., 30 mins). Executing the model at each time step entailed implicitly calculating the manure temperature using the values from the previous time step. The boundary conditions for the manure temperature at the surface and bottom layers are assumed to equal the adjoining ambient air and soil temperature, assuming the soil temperature is constant and equal to the average annual air temperature (Rennie et al., 2017). Finally, the length of the finite element grid changed over time to reflect the manure addition and removal practices that affect the manure depth in the storage structure. Equation (2) presents the underlying heat transfer equation in the FEHT model.

\[
\frac{\partial}{\partial z} \left( k \frac{\partial T_m}{\partial z} \right) + q = \rho C_p \frac{\partial T_m}{\partial t} \tag{4.2}
\]

Where \( T_m \) is the temperature of the manure in the storage structure (K), \( z \) is the depth of the manure (m), \( \rho \) is the density of the dairy manure \( \left( \frac{Kg}{m^3} \right) \), and \( C_p \) is the specific heat capacity of manure \( \left( \frac{KJ}{Kg*K} \right) \), and \( q \) is the overall heat flux \( \left( \frac{W}{m^2} \right) \). The heat flux, \( q \) in equation (2), is the sum of radiative, convective, and conductive heat fluxes described in equation (3)

\[
q = q_r + q_{cv} + q_{sv} - q_{cd} \tag{4.3}
\]

Where \( q_r \) denotes the radiative heat flux \( \left( \frac{W}{m^2} \right) \), \( q_{cv} \) is the convective heat flux between the manure surface and the ambient air temperature \( \left( \frac{W}{m^2} \right) \), \( q_{sv} \) is the conductive heat flux between the soil and manure in the bottom layer of the storage structure \( \left( \frac{W}{m^2} \right) \), and \( q_{cd} \), the conductive heat transfer between the manure layers \( \left( \frac{W}{m^2} \right) \), calculated using the following equations:
\[ q_r = \varepsilon \sigma A_m (T_a^4 - T_m^4) \]
\[ q_{cv} = h_m (T_a - T_m) \cdot A_m \]
\[ q_{sv} = h_s (T_s - T_m) \cdot A_m \]
\[ q_{cd} = -k A_m \frac{\partial T_m}{\partial t} \]  
\hspace{1cm} (4.4)

Where \( A_m \) is the surface area of stored manure \((m^2)\), \( T_s \) is the temperature of the soil below the bottom layer of manure (K), \( T_a \) is the ambient air temperature (K), \( h_m \) is the conductive heat transfer coefficient between the manure surface and the ambient air \( \left( \frac{W}{K \cdot m^2} \right) \), \( h_s \) is the heat transfer coefficient between manure and soil \( \left( \frac{W}{K \cdot m^2} \right) \), \( \varepsilon \) is the surface emissivity of liquid dairy manure, \( \sigma \) is the Stefan-Boltzmann constant \( 5.67 \times 10^{-8} \left( \frac{W}{K^4 \cdot m^2} \right) \), and \( k \) is the thermal conductivity of liquid dairy manure \( \left( \frac{W}{K \cdot m} \right) \).

The heat from the microbial activities in the manure, the incoming manure temperature, and the evaporative heat transfer were not considered. The thermal conductivity \( k \) and the specific heat capacity \( C_p \), of the stored manure are calculated using equations by (Nayyeri et al., 2009), while the convective heat transfer coefficient at the manure surface \( h_m \) is calculated following (Greiner, 1980) equations. Finally, the manure density \( \rho \) and the soil thermal conductivity \( h_s \) are adapted from (Rennie et al., 2017). A summary of the models’ thermophysical properties is in Table 4-1.
Table 4.1: The thermophysical properties of dairy manure.

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k$</td>
<td>0.5958</td>
<td>(Nayyeri et al., 2009)</td>
</tr>
<tr>
<td>$C_p$</td>
<td>2.42386</td>
<td>(Nayyeri et al., 2009)</td>
</tr>
<tr>
<td>$\rho$</td>
<td>1014.2</td>
<td>(Rennie et al., 2017)</td>
</tr>
<tr>
<td>$\epsilon$</td>
<td>0.95</td>
<td>(Rennie et al., 2017)</td>
</tr>
<tr>
<td>$h_s$</td>
<td>2.0</td>
<td>(Rennie et al., 2017)</td>
</tr>
<tr>
<td>$h_m$</td>
<td>589</td>
<td>(Greiner, 1980)</td>
</tr>
</tbody>
</table>

$h_m$ is the conductive heat transfer coefficient between the manure surface and the ambient air, $\epsilon$ is the dairy manure surface emissivity, $C_p$ is the specific heat of manure, $h_s$ is the heat transfer coefficient between manure and soil, $\rho$ is the density of the dairy manure, $k$ is the liquid dairy manure thermal conductivity.

**Neural network-based models (DDNN, PINN, SSNN)**

**General neural network models formulation**

The neural network (DDNN, PINN, and SSNN) models used in this study had similar architecture. We used a fully connected feed-forward network architecture (known as multilayer perceptrons), where the basic computing units (neurons) are stacked in layers. The neural networks were implemented using the TensorFlow library in Python. The neural networks are composed of an input layer, $m_L$ hidden layers with $m_h$ neurons in each hidden layer, and an output layer with one neuron. The number of neurons in the input layer equals the dimensionality of the input features $x$.

Functionally, each neuron is connected to all the neurons in the previous layer with adjustable parameters (weights and biases). For each layer, the initial biases are set to zero, and the weights are initialized randomly using a normal distribution with mean $= 0$ and standard deviation $= 0.01$. The layers in the networks are connected by the rectified linear unit (ReLU), a nonlinear activation function (Agarap, 2018). Finally, to avoid variability caused by random initialization and sampling, attain reproducibility, and make fair comparisons, identical random seeds for the TensorFlow and NumPy libraries are used for all models. The
function form of neural networks with n hidden layers and parameters ($\theta$) is given as follows:

$$y(x) \approx \hat{y}(x; \theta) = y_{n+1}(y_n(...) (y_1(x)))$$ (4.5)

where $\hat{y}(x; \theta)$ denotes the neural network approximation for the measured data $y(x)$. In the case of three hidden layers, the network functions are

$$y_1(x) = \sigma(W_1x + b_1)$$
$$y_2(y_1) = \sigma(W_2 y_1 + b_2)$$
$$y_3(y_2) = \sigma(W_3 y_2 + b_3)$$
$$\hat{y}(x; \theta) = y_4(y_3) = (W_4 y_3 + b_4)$$ (4.6)

Where, $\sigma$ is the activation function, $x \in R^d$ is the d-dimensional coordinate vector representing the input features to the neural networks. $W_i$ and $b_i$ are the matrix of weights and vector of biases of the ith layer, respectively. Together, these form the neural network parameter vector $\theta$:

$$\theta = W_1, W_2, W_3, b_1, b_2, b_3$$ (4.7)

The neural networks permit information transmission in two directions, that is, from the input to the output (forward) and back to the input (backpropagation) (E. O. Abiodun et al., 2021). When the information flows in the forward direction, the network uses its parameters along with the activation functions to calculate the network’s output (equation 5). Then, the error between the output of the network and the target output is calculated using the loss function. The value of this error propagates backward in the network to compute the gradient required for updating the network parameters. Then, these parameters are adjusted using a stochastic optimization method to minimize the error generated from the loss function. This
process is called backpropagation (E. O. Abiodun et al., 2021).

Since the loss function is nonlinear and non-convex with respect to the network parameters, we used a gradient descent minimization algorithm called the ADAM optimizer (Kingma & Ba, 2014). The ADAM optimizer uses a stochastic gradient method with momentum, set at the default parameters and a learning rate of $\alpha = 0.001$. The ADAM method stops the neural network training once the predetermined number (3,000 in this study) of iterations (epochs) is completed. The loss function for determining the network's parameters ($\theta$) is as follows:

$$(\theta) = \arg \min_{\theta} \mathcal{L}(\theta)$$  \hspace{1cm} (4.8)

Equations (9), (10), and (11) describe the loss function $\mathcal{L}(\theta)$ for the DDNN, SSNN, and PINN models, respectively. Here, $\mathcal{L}_d(\theta)$ is the loss due to a mismatch with the data (i.e., the measurements of the manure temperatures, $y(x)$) and $\mathcal{L}_s(\theta)$ is the loss due to the mismatch with simulated manure temperatures by the FEHT model, $\tilde{y}(x;p)$. Finally, the loss function in the PINN model is the weighted average of $\mathcal{L}_d(\theta)$ and $\mathcal{L}_s(\theta)$.

$$\mathcal{L}_d(\theta) = \frac{1}{N} \sum_{x \in N} (\hat{y}(x;\theta) - y(x))^2 \quad (4.9)$$

$$\mathcal{L}_s(\theta) = \frac{1}{N} \sum_{x \in N} (\hat{y}(x;\theta) - T_m(x;p))^2 \quad (4.10)$$

$$\mathcal{L}_{PT}(\theta) = \lambda J_d(\theta) + (1 - \lambda) J_s(\theta) \quad (4.11)$$

The parameters $\lambda$ and $(1 - \lambda)$ reflect the weights of the $\mathcal{L}_d(\theta)$ and $\mathcal{L}_s(\theta)$ in the PINN model loss function, respectively. $y(x)$ is the target output (the measured manure temperature), $\hat{y}(x;\theta)$ is the predicted output by the neural network, and $T_m(x;p)$ is the output of the
FEHT model (the simulated manure temperature). $\mathcal{L}_d(\theta)$ is the observed data discrepancy represented as the MSE between $\hat{y}(x; \theta)$ and $y(x)$; while $\mathcal{L}_s(\theta)$ is the physics-based discrepancy obtained as the MSE between $\hat{y}(x; \theta)$ and the corresponding $T_m(x; p)$. Hence, the loss $\mathcal{L}_s(\theta)$ term in the minimization problem is the physics-informed regularization terms. The value of $\lambda$ lies between 0 and 1; for instance, if $\lambda$ equals 1, the model is entirely data-driven and corresponds to DDNN. Figure 4-1 describes the implementation of the neural network models.

![Figure 4.1](image)

Figure 4.1: Flow diagram for the physics-informed neural network model used to predict the stored manure temperature.
Tuning hyperparameters for neural network models

While an acknowledged robust modeling approach, one limitation of PINN presented in the literature is sensitivity to the initial hyperparameters of the neural network (Lawal et al., 2022). He et al. (He et al., 2020) also reiterated the need to study the hyperparameters’ influence on PINN model performance. Thus, we performed a parametric study on three hyperparameters: the number of hidden layers $m_L$, the number of neurons in each hidden layer $m_h$, and the weight of the data discrepancy in the PINN model loss function $\lambda$. In conducting the parametric study, we kept two hyperparameters constant while changing the third one. Then, we evaluate the performance of the models based on the $R^2$ value during the model validation and testing. Table 4-2 shows the hyperparameters tested for each model and the search domain for each hyperparameter.

Table 4.2: The hyperparameters tuned for each model and the search domain

<table>
<thead>
<tr>
<th>Model</th>
<th>DDNN</th>
<th>SSNN</th>
<th>PINN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hyperparameters</td>
<td>$m_h \in [5, 128]$</td>
<td>$m_h \in [5, 128]$</td>
<td>$m_h \in [5, 128]$</td>
</tr>
<tr>
<td></td>
<td>$m_L \in [1, 7]$</td>
<td>$m_L \in [1, 7]$</td>
<td>$m_L \in [1, 7]$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$\lambda \in [0.05, 0.09]$</td>
</tr>
</tbody>
</table>

Evaluating the models

The models were evaluated for their predictive power and generalization accuracy (robustness). The predictive power is reflected by the models’ ability to fit the input features to the observed training data and predict the unobserved data. The predictive power was assessed by the models’ performance during the validation phase, while model generalization accuracy refers to the model’s ability to meaningfully interpolate sparse and disparate data and make inferences for unseen combinations of the parameters (Cai et al., 2021; Willcox et al., 2021). In this study, the generalization accuracy was assessed when the models got
tested on the SCP (Test 1) and FCP (Test 2) datasets. Three performance metrics, the coefficient of determination ($R^2$), mean absolute percentage error (MAPE), and root means square error (RMSE) were used to evaluate the models’ performance (equations 12 – 14). The models with $R^2$ value > 0.7 and small (in magnitude) MAPE and RMSE values during the validation and testing phases are considered good performers.

$$R^2 = \frac{(\sum_{i=1}^{N} (y_i - \bar{y}) (\hat{y}_i - \bar{\hat{y}}))^2}{\sum_{i=1}^{N} (y_i - \bar{y})^2 \cdot \sum_{i=1}^{N} (\hat{y}_i - \bar{\hat{y}})^2}$$  \hspace{1cm} (4.12)

$$MAPE = \frac{1}{N} \sum_{i=1}^{N} \left| \frac{\hat{y}_i - y_i}{y_i} \right| \times 100$$  \hspace{1cm} (4.13)

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N} (\hat{y}_i - y_i)^2}{N}}$$  \hspace{1cm} (4.14)

Where $y_i$ is the target value, $\hat{y}_i$ is the predicted value, $\bar{y}$ is the averaged target value, $\bar{\hat{y}}$ is the averaged predicted value, and $N$ is the number of observations.

Further, we used boxplots to visually display the difference between the predicted and observed manure temperatures (i.e., the residuals) to confirm the models’ validity. They are used to assess the models’ biases. The models were classified as systematically biased (over-prediction or under-prediction) if the median of the boxplot was not close to zero. Further, skewed box plots suggest the model fails to explain some relations within the data and is qualitatively inconsistent. Finally, we evaluated the neural network models’ data efficiency by assessing each model’s required data to give valid and reliable results.
4.3 Results

4.3.1 Neural networks’ input features and hyperparameters

The features selection process yielded the ambient air temperature, wind speed, solar radiation, relative humidity, time (represented by month), and depth of manure layer above a sensor as the most relevant, pertinent, and significant input features. All these features had a correlation coefficient with the manure temperature of 25% or more and were all statistically independent.

Hidden layers

The performance of the neural network models is highly dependent on the network hyperparameters. Figures 4-2 and 4-3 show the performance evaluation metric, $R^2$, for the different models during the models’ validation and testing given a different number of hidden layers and hidden neurons, respectively. The models’ performance of neural network models improves during the validation phase (i.e., higher $R^2$ value) as the number of hidden layers increases. This result is not surprising since neural networks with more layers develop more complex functions to map the input features to the training data; however, these deep neural networks are more prone to overfitting the training data. Hence, increasing the number of layers is expected to increase the chance of data overfitting (i.e., poor generalization accuracy). However, Figure 4-2 shows that this is not always the case. Increasing the number of layers had a low impact on the DDNN and PINN models’ performance during the model testing on SCP and FCT datasets. The SSNN model exhibited a similar pattern to the DDNN and PINN models except for six hidden layers. We could not explain the reason behind the drop in the SSNN model when the number of hidden layers was six.
Figure 4.2: The performance of the DDNN, PINN, and SSNN models with different numbers of hidden layers, each with five neurons (a) during the validation phase, (b) when tested on the data collected from the clay pit on the scraped farm, and (c) when tested on the data collected from the concrete tank on the flushed farm. (DDNN – data-driven neural networks; PINN – physics-informed neural networks; SSNN- Simulation study trained neural networks).
Hidden neurons

The performance of the DDNN, PINN, and SSNN models was not affected by the variation in the number of neurons during the models’ validation. However, the generalization accuracy of the DDNN and PINN models deteriorates with the increasing number of neurons in each hidden layer, possibly due to data overfitting. Similar observations were reported by Dwivedi and Srinivasan (Dwivedi & Srinivasan, 2020). They found that increasing the number of neurons leads to an initial drop in accuracy followed by saturation to an even more inaccurate solution. The SSNN model performance was highly variable as the numbers of neurons changed; no clear pattern was observed.

In summary, we generally observe a superior performance of the PINN model compared to DDNN and SSNN models using different hidden layers and neurons. The computational time increased as the number of hidden layers and neurons increased. For instance, the computational time increased by 22% when the networks’ architecture changed from two hidden layers to three hidden layers. Also, when the architecture changed from two hidden layers with five neurons to two with ten, the computational time increased by almost 40%. This result is similar to Shen et al. [34], who found that time increased from 110 s to 140 s when hidden layers increased from one to five in their work using the PINN model in bearing fault detection. These results support the notion that selecting the appropriate number of hidden layers and neurons is critical; choosing an inadequate network architecture can decrease the model’s predictive ability and generalization accuracy. Thus, based on the results in Figures 4-2 and 4-3, the selected architecture for the neural network (DDNN, SSNN, and PINN) models is two hidden layers with five neurons each.
Figure 4.3: The performance of two hidden layers DDNN, PINN, and SSNN models with different numbers of neurons (a) during the validation phase, (b) when tested on the data collected from the clay pit on the scraped farm, and (c) when tested on the data collected from the concrete tank on the flushed farm. (DDNN – data-driven neural networks; PINN – physics-informed neural networks; SSNN- Simulation study trained neural networks).
Lambda

Tuning the value of $\lambda$ in the PINN model is essential since small changes can affect the manure temperature predictions. In this study, as $\lambda$ increases, the PINN model predictions approach the DDNN model; for example, during the models’ validation, the performance of the PINN model improved as the value of $\lambda$ increased until it coincided with the DDNN predictions. However, the generalization accuracy was very sensitive to the $\lambda$ changes. For example, changing $\lambda$ from 0.65 to 0.7 increased the PINN model performance by 33% and 18% when tested on SCP and FCT datasets, respectively. The optimum value of $\lambda$ for the selected physics-informed neural network’s architecture is 0.7.

4.3.2 Models performance during validation

The performance evaluation metrics of the models predicting the manure temperatures are in Figure 4-4. Based on the performance evaluation criteria (i.e., $R^2$, MAPE, and RMSE), the DDNN and PINN models performed best when predicting the unobserved data from the SCT storage structure (validation dataset). The $R^2$ values for the DDNN and PINN models are close to 1, and their MAPE and RMSE magnitudes are smaller than the FEHT and SSNN models. Additionally, the $R^2$ values for the FEHT and SSNN models are lower than 0.7, with the magnitudes of MAPE and RMSE almost double the DDNN and PINN models. Hence the FEHT and SSNN models have poorer performance and lower predictive power than the DDNN and PINN models.
Figure 4.4: The performance metrics for the trained models during the models’ validation and testing on the data from the clay pit manure storage (SCP) and the concrete manure storage tank in a flushing farm (FCT). (MAPE is the mean absolute percentage error; RMSE is the root mean square error. $R^2$ is the coefficient of correlation; FEHT—finite-elements heat transfer; DDNN is the data-driven neural network; SSNN is a Simulation study trained neural network; PINN is the physics-informed neural network.)
The predictive ability of the DDNN and PINN was higher than the FEHT and SSNN models, with an $R^2$ value of 0.93 and 0.92, respectively, for the SCT. The notable difference between the DDNN and PINN models’ predictions occurred in January (DOY 1 to 31) when the DDNN and the PINN models under-predicted manure temperatures by 3.5% and 16.8%, respectively. Potential reasons for the DDNN model being slightly more accurate than the PINN during the validation phase may be attributed to the large number of data points (42,000) being sufficient to train the neural networks accurately. Also, adding physics constraints made it more difficult for the loss function to optimize the network parameters, similar to He et al. (He et al., 2020) observations in the case of large measurements.

The results also show that the FEHT and SSNN models predict the overall trend of the manure temperature but fail to capture the magnitude. Another observation was that the FEHT and SSNN models predicted temperatures tend to be lower than the observed manure temperature. These results align with the results found in the literature (R. A. Genedy et al., 2021; Rennie et al., 2017). The differences between the predicted manure temperature by the FEHT and SSNN models and the observed manure temperature were highest in August and September (DOY 213 to 273). During that period, the SSNN and FEHT predicted manure temperatures were about 25% lower than the observed. This difference is noteworthy since most of the aerial emissions from the stored manure occur during the warm months of July, August, and September when the manure temperature is peaking. Thus, this result implies that relying on the governing heat transfer equations to predict manure temperature in decision support tools will most likely underpredict the aerial emissions from the manure storage structures.
Figure 4.5: Models validation: the average daily observed manure temperature, ambient air temperature, and the predicted manure temperature in the concrete storage tank located in a scrap farm (FEHT – finite-elements heat transfer model; DDNN – data-driven neural networks; PINN – physics-informed neural networks; SSNN- Simulation study trained neural networks; SCT – concrete storage tank located in a scrap farm; DOY – day of the year).
4.3.3 Models performance during testing

Figures 4-6 and 4-7 show the generalization accuracy of the models achieved by tests on the data collected from the clay pit on the scraped farm (Test 1) and the concrete tank on the flushed farm (Test 2), respectively. The PINN model generally has superior generalization accuracy compared to the other models. The PINN model predicted manure temperature closer in magnitude to the observed temperatures ($R^2$ values of 0.85 and 0.70 in Tests 1 and 2, respectively). On the other hand, the DDNN model performance declined in Tests 1 and 2 compared to its performance during the model validation ($R^2 < 0.7$). The decline in DDNN model performance is not surprising because purely data-driven deep learning does not fully capture the complex nature of activities in the stored manure and does not include information about the governing physics of heat and material transfer. Adding the physics constraints to the neural network loss function increased the generalization accuracy of the DDNN by up to 40%, similar to Tartakovsky et al. (Tartakovsky et al., 2020) who found that PINN improved the DDNN performance by 50%.

Further, DDNN model predictions tend to follow the training data instinctively. For instance, the DDNN model predicted manure temperatures for the SCP and FCT during the cold period were 3 ºC higher than the observed manure temperature (similar to observations for the SCT). Similar observations were made by Shen et al. (Shen et al., 2021) in their study, reporting that the data-driven model could not efficiently learn the physics of faults in bearings captured in sensor signals. The PINN model had better generalization accuracy and predictions for the SCP and FCT, which suggests that the PINN identifies the unknown patterns outside the training dataset and is less prone to overfitting data like the DDNN. Our observations conform to experiences in other applications that have used PINN models, such as solving Burgers’ and Navier-Stokes equations (Nabian & Meidani, 2018), cardiac activation mapping (Sahli Costabal et al., 2020), and bearing condition monitoring and fault
detection in agricultural machines (Shen et al., 2021).

The FEHT model was the second-best performer at predicting the manure temperature at the SCP ($R^2=0.75$), but it underperformed at the FCT ($R^2=-0.03$). Lastly, the SSNN model captures the overall manure temperature trend in SCP and FCT storage structures with relatively poor performance. This result suggests that FEHT and SSNN models are unsuitable for predicting manure temperature universally. All the manure temperature predictions were generally higher than the measured temperatures during the cold periods and were lower during the warm periods. During warm periods, the DDNN and PINN models predict manure temperatures better than FEHT and SSNN (Figures 4-6 and 4-7). For example, from July to October, when manure temperature peaks at the SCP, the PINN and DDNN models predicted manure temperatures are different from the observed manure temperature by 4.7% and 10%, respectively; on the other hand, the FEHT and SSNN predictions were about 17.5% lower than the observed manure temperatures. The MAPE and RMSE magnitudes are more prominent for the models tested on the FCT dataset than when tested on SCP data. These results suggest that models perform better at predicting the manure temperature at the SCP than at the FCT, perhaps because the SCP and SCT have similar manure management practices and are in the same geographical zone.
Figure 4.6: Models testing on the SCP dataset: the average daily ambient air temperature, observed manure temperature, and predicted manure temperatures in the clay storage pit (FEHT—finite-elements heat transfer model; DDNN—data-driven neural networks; PINN—physics-informed neural networks; SSNN—Simulation study trained neural networks; SCP—clay storage pit located in a scrap farm; DOY—day of the year).
Figure 4.7: Models testing on the FCT dataset: the average daily ambient air temperature, observed manure, and the predicted manure temperatures in the concrete storage tank receiving manure from a flushed farm (FEHT– finite-elements heat transfer model; DDNN – data-driven neural networks; PINN – physics-informed neural networks; SSNN–Simulation study trained neural networks; FCT - concrete storage tank located in a flushing farm; DOY – day of the year).
4.3.4 Cumulative performance

The overall manure temperature prediction accuracy of the PINN approach is higher than that of the FEHT, DDNN, and SSNN, indicating that the proposed PINN approach can generate a unified and generalized model for predicting the manure temperature in different manure management systems. The overall accuracy for the FEHT, DDNN, SSNN, and PINN models are relatively 0.44, 0.72, 0.61, and 0.82. Based on the proposed ranking criteria ($R^2 > 0.7$), both DDNN and PINN models performed well. The differences between the observed and the predicted manure temperatures (residuals) are shown in Figure 4-8. The FEHT, DDNN, and PINN models residuals tend toward a normal distribution, suggesting that the errors from these models have a random distribution (i.e., the models fit the data well for these three storage structures). On the other hand, the SSNN residual box plots for tests on SCP and FCT are skewed, denoting the potential bias of the SSNN, with a tendency to underpredict the manure temperature at the SCP and FCT. Overall, the PINN model has the most accurate predictions based on the residual analysis presented in Figure 4-8. However, the DDNN outperformed the PINN model during model validation because the validation data was collected from the same system as the training data. In contrast, the PINN model performed better during model testing than the other models. Thus, the PINN model provides a more practical and generalized way to estimate the stored manure temperature to be used as an input parameter in decision support tools and nutrient accounting models (e.g., (Chianese et al., 2009; Li et al., 2012; Mangino et al., n.d.)).

Data availability plays a vital role in models’ selection and predictive ability. The FEHT and SSNN models had the highest data efficiency; they do not require data collection and depend only on the simulated data. However, using such models in decision support tools and nutrient accounting models will likely underestimate the nutrient loss to the environment. On the other hand, the DDNN had the lowest data efficiency among the models; It requires
a large amount of data to give accurate predictions. However, DDNN is more amenable to change and adapt over time since data changes with evolving systems (Nevala, 2017; Paluszek & Thomas, 2016). Finally, physics-informed modeling provided more reliable results with less data than the DDNN by flexibly adjusting the $\lambda$ parameter in the loss function during the training process. For example, when working with a small amount of data, more priority could be given to the physics-driven component of the loss function (i.e., using a low value of $\lambda$). Therefore, the PINN model is more suitable for solving problems with known governing equations and limited data (Tartakovsky et al., 2020).

Despite the numerous benefits of using the PINN approach, they have several drawbacks and limitations. First, PINN training is relatively slow due to the gradient descent optimization. Although the PINN computation time was faster than the DDNN, they cannot be used in real-time applications. Also, for forward problems, such as ours, the PINN approach is not competitive compared to FEHT methods with respect to computational time (Henkes et al., 2022). Another limitation is that PINNs are vulnerable to vanishing gradient problems when used with deep networks (Dwivedi & Srinivasan, 2020; Lawal et al., 2022). This was shown when we increased the number of layers and neurons, indicating that the solution was possibly stuck at a local minimum point. Finally, in our study, the PINN learning process was fine-tuned by hand, limiting our ability to determine how much data or which architecture is good enough for the PINN training.
Figure 4.8: Residual plots of the differences between the observed and predicted manure temperatures (a) during model validation, (b) when the models were tested on the data from scraped manure pit, and (c) when the models were tested on the data from a flushed concrete tank, along with the average. (FEHT—finite-elements heat transfer model; DDNN—data-driven neural networks; PINN—physics-informed neural networks; SSNN—Simulation study trained neural networks).
4.4 Conclusion

Classical neural networks are considered "black box" models and often do not provide meaningful and physically explainable predictions. This work presents a new approach to predicting manure temperature during storage using PINN. PINN embeds the knowledge of heat transfer equations into neural network models to improve the models’ generalization and transportability. The performance and generalization accuracy of the models were evaluated using observed manure temperature data from three on-farm storage structures with different manure management practices. We conducted a parametric study that demonstrated the importance of the architecture on the performance of the neural network’s models. The PINN performed better, including a better generalization accuracy than DDNN, FEHT, and SSNN models. The number of hidden neurons, the weights of the data discrepancy, and the physical discrepancy in the loss function affected the PINN generalization accuracy.

The PINN model showed high predictive ability and generalization accuracy compared with DDNN, FEHT, and SSNN models. On average, the FEHT and SSNN models underpredict the manure temperature during the warm periods when most manure emissions occur; therefore, using them in decision support tools will underestimate the contaminants lost to the atmosphere during the manure storage period. The DDNN model lacks the robustness to enable its usage for other manure storage structures. The PINN model provides physically interpretable predictions while maintaining good performance. The PINN model presents a more accurate, less biased, and generalized approach to predicting stored manure temperatures. Thus, if used in decision support tools and nutrient cycling models, the results from this study would improve the accuracy of assessing sustainable manure management practices.

The overall accuracy for the PINN model was 82%; however, the model had several limi-
tions, including (1) the physics-based model used to constrain the loss function was simplified and lacked some important influencing factors (such as crusting) because they are not easy to describe by known physical equations. (2) The network’s hyperparameters were tuned by hand, possibly affecting the network’s overall accuracy. To address these two challenges, we propose using the PINN approach to solve inverse problems to estimate the manure’s heat and mass transfer coefficients in the governing heat and mass transfer equations, which can potentially improve the accuracy of physics-based modeling. Also, we will study the influence of the networks’ hyperparameters (alpha, network architecture, batch size, etc.) using automated tuning techniques such as random and grid searches. Our future work will also investigate the applicability of the proposed physics-informed deep learning approach to extended applications such as predicting the nutrient losses and aerial emissions from the stored manure. Finally, we hope our study will inspire research using physics-informed deep learning in related applications such as emission estimation and soil temperature predictive models.

References


view of Artificial Neural Network Applications to Pattern Recognition. In IEEE Access (Vol. 7, pp. 158820–158846). Institute of Electrical and Electronics Engineers Inc. https://doi.org/10.1109/ACCESS.2019.2945545


Naturally Ventilated Duck Houses: Application of Recurrent Neural Network. Agriculture (Switzerland), 12(3). https://doi.org/10.3390/agriculture12030318


Chapter 5

Combining process-based modeling and recurrent neural networks to estimate ammonia emissions from stored dairy manure.
Abstract

Effective manure storage management on dairy farms is crucial for maximizing its fertilizer value, reducing costs, and minimizing environmental pollution. However, manure nitrogen loss through ammonia volatilization during storage remains a challenge. Quantifying the ammonia losses is necessary to inform policymaking and decision processes to improve emission mitigation strategies. Although process-based modeling is a commonly used tool for estimating ammonia emissions, it can be challenging for complex systems such as manure storage structures. This study introduces a deep-learning approach that combines process-based modeling and recurrent neural networks as an alternative method to estimate ammonia loss from dairy manure during storage. In addition, it explores inverse problem-solving method to estimate the heat and mass transfer coefficients for ammonia transport and emission from stored manure using the hyperparameters optimization tool, Optuna. The study evaluated the models using open-source datasets from two on-farm liquid dairy manure storage structures. Results indicate that the process-based model parameters and the value of lambda have the greatest impact on the performance of the LSTM model. Optimizing the process-based model parameters has led to an improvement in its performance. The overall RMSE for the physics-informed LSTM model was 1.51 $gm^{-2}h^{-1}$ compared to 3.01 and 2.17 for the process-based model before and after hyperparameters optimization, respectively. The study also demonstrates that incorporating physical knowledge into machine learning models can enhance their generalization accuracy. Lastly, physics-informed LSTM has the potential to replace conventional process-based models as its computational efficiency and does not require extensive data collection. The outcomes of this study can aid in designing suitable on-farm strategies to minimize nutrient loss during manure storage periods. Keywords: Recurrent neural networks, physics-informed LSTM, process-based models, ammonia emissions.
5.1 Introduction

Manure storage is an essential component of best management practices for nutrients on livestock farms. It provides the farm opportunities to derive benefits and maximize the fertilizer value, reduce handling costs, and minimize the potential of manure to pollute the environment (Rotz, 2004; Veltman et al., 2018). However, during storage, the microbial degradation and biogeochemical processes that occur in manure alter the quality of manure, resulting in the formation and release of gaseous constituents such as ammonia and greenhouse gases. Up to 60 percent of the stored manure nitrogen can be lost as ammonia during storage, depending on management and environmental conditions (Arogo et al., 2006; Baldé et al., 2018; Misselbrook et al., 2016). Losing ammonia means reducing the fertilizer (nitrogen content) and economic value of manure. Moreover, ammonia losses significantly contribute to environmental pollution (Galloway et al., 2003; Sommer et al., 2019). Some reported adverse effects of ammonia emissions to the atmosphere include degraded air quality, human health, biodiversity, soil health, and climate change (Hill et al., 2019; Sanchis et al., 2019).

The National Research Council (NRC) stressed the need to accurately quantify ammonia emissions from livestock production operations is needed to design appropriate mitigation strategies (National Research Council, 2003). While much work has been done on quantifying ammonia emissions since 2003, more needs to be done to improve the accuracy of nutrient inventories at local, state, regional, and national levels, policies and regulations, and design appropriate mitigation strategies (Arogo et al., 2006; Grant & Boehm, 2020; Kupper et al., 2021).

Quantifying ammonia can be a challenge due to farm management practices, equipment, and skills needed (Arogo et al., 2006; National Research Council, 2003). Methods for quantifying ammonia emissions from livestock production include direct measurements, emission factors, and mathematical models. Direct measurements techniques using passive samplers...
(acid-based filters and scrubbers, detector tubes), electrochemical cells, optical absorption, photoacoustic, and gas chromatography, are considered the most reliable (Genedy et al., 2023; Grant & Boehm, 2020; Shah et al., 2006; Thöni et al., 2003; Todd et al., 2005). However, direct measurement can be challenging and expensive depending on the site because of the equipment and skill level required to set up and conduct the measurements (Arogo et al., 2006; Genedy & Ogejo, 2022). The emission factor method expresses the amount of emitted ammonia as a function of the stored manure’s total ammoniacal nitrogen (TAN), the number of animals raised, or the mass of live animals raised (Sommer et al., 2019). Despite their simplicity, emission factors can vary significantly based on local management practices, environmental conditions, and manure storage structures and cannot be generalized. Lastly, mathematical models, including empirical and process-based models, can quantify the transport and volatilization of ammonia from manure storage structures using the governing heat and mass transfer equations.

The NRC (2003) recommended using process-based models as an alternative and more meaningful approach to estimating aerial emissions from animal feeding operations (National Research Council, 2003). Several process-based models have been developed since the NRC (2003) recommendation and are used to estimate ammonia emissions from manure storage, such as the Manure Denitrification-Decomposition (Manure-DNDC) model (Li et al., 2012), Integrated Farm System Model (IFSM) (Rotz et al., 2014), and Compartmental Process-based Model (Karunarathne et al., 2020).

While process-based modeling provides more accurate estimates for ammonia emissions, it has been noted that the current process-based models have several shortcomings that limit accurate model predictions. They depend highly on the different manure management practices (agitation and frequency of removal), the type of storage structure (concrete tanks, steel tanks, clay pits), and the shape and dimensions of the storage structure, which can limit
the process-based models’ accuracy. Also, modeling the systems under conditions difficult to describe by physical equations, such as crusting or snow cover on the manure surface and temperature stratification, is challenging. Finally, Heat and mass transfer coefficients for the ammonia transportation and emission from stored manure are unavailable for different manure storage systems.

The advent of machine learning has brought a new perspective in the conduct of research when dealing with complex systems, such as manure storage structures, and potentially overcome modeling challenges with adequate data. Machine learning techniques can be used to overcome the limitations of traditional process-based models and improve the accuracy of predicting ammonia emissions from manure storage (Genedy et al., 2021). Machine learning algorithms, and more particularly recurrent neural network (RNN), offer a new perspective on complex systems, such as manure storage structures, and can learn the time dynamics and underlying complexity in a data-driven manner without fixed parameters or structures. RNNs have been used successfully with time series datasets, with various architectures such as long short-term memory (LSTM) and gated recurrent unit. The LSTM provides several advantages. First, it is rooted in their superior performance in a wide range of time-series predictive tasks, including agricultural applications, such as yield predictions (Shen et al., 2022; J. Wang et al., 2022), forecasting product prices (Kurumatani, 2020), and particulate matter (PM2.5) monitoring (Ong et al., 2016). Second, the LSTM architecture manages well the issues with the vanishing gradient problem that often arises in RNNs, allowing it to capture long-term dependencies in sequential data (Hochreiter, 1998; Hochreiter & Schmidhuber, 1997; Sun et al., 2022). However, data-driven LSTMs are prone to overfitting and local minimum problems. Also, the computation time, memory footprint, and power consumption can increase as the LSTM cell becomes more complex (Asrav & Aydin, 2023). Finally, data-driven machine learning often require significant training that may be infeasible
This study aims to develop a physics-informed LSTM model to estimate ammonia emissions from stored manure. The physics-informed LSTM model embeds the compartmental process-based model into the LSTM architecture for manure storage environment. This model is designed to overcome data overfitting and improve the robustness of the LSTM cell by adding a physics regularization term to the LSTM loss function. One advantage of regularization is that it restricts the permissible solutions space and increases the LSTM approximation and generalization accuracy. Also, the physics regularization enables the LSTM to steer itself toward accurate solutions quickly, making it more data and time efficient. Finally, the physics-informed LSTM model was used to perform inverse problem-solving to calculate the heat and mass transfer coefficients and parameters for the ammonia transportation and emission from stored manure. This was achieved through a physics-informed hyperparameter tuning strategy.

In summary, the physics-informed LSTM model developed by this study has three key contributions. First, it develops a generalized model for estimating ammonia emissions from various manure storage systems. Second, it optimizes the heat and mass transfer parameters and coefficients of process-based models through hyperparameter tuning, thereby improving their overall accuracy. Third, it investigates how to integrate physics knowledge into the hyperparameter selection process. The outcomes of this study contribute to enhancing the accuracy of ammonia emission quantification methods which will allow the designing of suitable on-farm strategies to minimize nutrient loss and greenhouse gas emissions.
5.2 Materials and methods

5.2.1 Models’ development

The two models that form the basis for estimating ammonia emissions from manure storages are the compartmental process-based and a physics-informed LSTM model. A summary of the information flow of the two models is shown in Figure 5-1.

Figure 5.1: The information flow for (a) the compartmental process-based models, and (b) the physics-informed long-short term memory (LSTM) model.
Compartmental process-based model

The compartmental process-based model used was developed by Karunarathne et al. (2020). Briefly, the compartmental process-based model comprises six sub-models describing the processes associated with ammonia production and emission. These sub-models include (1) determining the surface area of the manure storage, (2) calculating the material balance in the manure storage to assess changes in manure depth and volume over time, (3) analyzing heat transfer and temperature distribution in stored manure, (4) modeling organic N mineralization, (5) simulating diffusion of total ammonia nitrogen in stored manure, and (6) predicting ammonia volatilization (Karunarathne et al., 2020). The model is one-dimensional and compartmentalized by sectioning the manure stored in the vertical domain (z). It estimates the spatial distribution of ammonia concentration by using established heat and mass transfer equations. The pertinent equations used in the model are:

\[
A = \left( \frac{M_{\text{manure}}^{\text{NAU}}}{\rho_{\text{manure}}} + \frac{f_{VR} M_{\text{bedding}}^{\text{NAU}}}{\rho_{\text{bedding}}} \right) \frac{t_{\text{days}}}{D_T - R_a - E_a - H_{FB}} + (R_a + S_{25y}) \text{Area}_{\text{runoff}} \tag{5.1}
\]

\[
\frac{dM_{\text{manure}}}{dt} = \dot{M}_{\text{manure in}} + \dot{M}_{\text{wastewater in}} + \dot{M}_{\text{rain}} - \dot{M}_{\text{evaporation}} - \dot{M}_{\text{land application}} \tag{5.2}
\]

\[
\frac{\partial T}{\partial t} = \frac{k}{\rho_{\text{manure}} c} \left( \frac{\partial^2 T}{\partial z^2} \right) + \frac{Q(z, t)}{\rho_{\text{manure}} c} \tag{5.3}
\]

\[
C_{TAN,i}^n = C_{TANold,i}^{n-1} + C_{TAN~gen,i}^n \text{ dt} \tag{5.4}
\]

\[
\frac{\partial C}{\partial t} = D \left( \frac{\partial^2 C}{\partial z^2} \right) \tag{5.5}
\]

\[
M_{NH_3} = K_L \left( FC_{\text{liquid}} - C_{\text{air}} \right) \tag{5.6}
\]

Where \( A \) is the surface area of the storage. \( M_{\text{manure}} \) and \( M_{\text{bedding}} \) are the mass of the manure and bedding per animal per day, respectively. \( \text{NAU} \) is the number of animals. \( f_{VR} \) is the
volume reduction factor. $\rho_{\text{manure}}$ and $\rho_{\text{bedding}}$ are the density of the manure and bedding, respectively. $t_{\text{days}}$ is the number of storage days. $R_a$ is the annual rainfall, $S_{25y}$ is the 25-year 24-hour storm and $E_a$ is the annual evaporation of the selected location. $\text{Area}_{\text{runoff}}$ is the land area exposed to runoff, $D_T$ and $H_{FB}$ are the total depth and height of the freeboard of the manure storage structure. $\dot{M}_{\text{manurein}}$, $\dot{M}_{\text{wastewater in}}$, $\dot{M}_{\text{rain}}$, $\dot{M}_{\text{evaporation}}$ and $\dot{M}_{\text{land application}}$ are the mass flow rates of the manure flowing into the storage, wash water flowing into the storage, rate of precipitation, evaporation rate, and rate of manure removal for land application, respectively. $k$ and $c$ are the thermal conductivity and specific heat of the manure. $T$ is the temperature, $t$ is the time step, and $dt$ is the length of a time step. $Q(z,t)$ is the internal heat generation rate per unit length. $C_{TAN,i}^n$ is the concentration of TAN in the $i^{th}$ layer at the end of $n^{th}$ time step, $C_{TANold,i}^{n-1}$ is the concentration of TAN in the $i^{th}$ layer at the beginning of $n^{th}$ time step, and $C_{TAN gen,i}^n$ is the generation of TAN in the $i^{th}$ layer at $n^{th}$ time step as a result of the mineralization of manure’s organic nitrogen. $D$ is the diffusion coefficient and $K_L$ is the gas-liquid interface coefficient of the manure. $C$, $C_{\text{liquid}}$, and $C_{\text{air}}$ are the concentration of ammonia in the manure, in the slurry, and in the surrounding air, respectively. $M_{NH_3}$ is the ammonia Flux predicted by the process-based model. The general flow diagram of the compartmental process-based model is presented in Figure 5-2.

**Physics-informed LSTM model**

The recurrent neural network framework, LSTM is used to predict the ammonia lost from the manure storage structures (Asrav & Aydin, 2023; Borkowski et al., 2023). LSTMs can be regarded as neural networks transmitted on the time axis, whose depth is the length of time to capture temporal dependencies of the data (Hochreiter & Schmidhuber, 1997; Sun et al., 2022; J. Wang et al., 2022; Y. Wang et al., 2022). The LSTM network contains specialized units called memory cells ($c_t$) that can store information over time. These cells have gates
Figure 5.2: The compartmental process-based model flow diagram. Source: Karunarathne et al. (2020) Physics-informed LSTM model.
that regulate the flow of data into and out of the cells. Three gates determine how a cell should be updated at each time step, namely, the forget \((f_t)\), input \((i_t)\), and output \((o_t)\) gates, as shown in Figure 5-3. The gates comprise sigmoid activation functions that map each input value to a range between 0 and 1, representing its degree of importance. The input gate controls the new information stored in the memory cell, and the output gate determines the information passed from the memory cell to the output. The forget gate controls the old information retained; if its value is close to 0, the past data is ignored, while a value close to 1 means it is retained. The LSTM architecture consists of a sequence of repeating modules, each containing a memory cell and three gates, as shown in Figure 5-3. The equations for the LSTM network are as follows:

\[
\begin{align*}
    i_t &= \sigma \left( W_{ix} x_t + W_{ih} h_{t-1} + b_i \right) \\
    f_t &= \sigma \left( W_{fx} x_t + W_{fh} h_{t-1} + b_f \right) \\
    o_t &= \sigma \left( W_{ox} x_t + W_{oh} h_{t-1} + b_o \right) \\
    \tilde{c}_t &= \text{tanh} \left( W_h h_{t-1} + W_x x_t + b_h \right) \\
    c_t &= i_t \odot \tilde{c}_t + f_t \odot c_{t-1} \\
    h_t &= o_t \odot \text{tanh} \left( c_t \right)
\end{align*}
\]

Where \(\sigma\) is the sigmoid activation function, \(\text{tanh}\) is the hyperbolic tangent activation function, \(\odot\) denotes element-wise multiplication, \(x_t\) is the input at time \(t\), \(h_{t-1}\) is the output of the previous time step, and \(\tilde{c}_t\) is the candidate memory cell at time \(t\). Weight matrices \(W_c\), \(W_u\), \(W_f\) and \(W_o\), and biases \(b_c\), \(b_u\), \(b_f\) and \(b_o\) are the weights and biases that govern the behavior of the \(\tilde{c}_t\), \(i_t\), \(f_t\), and \(o_t\) gates, respectively.
The loss function is a critical component for training neural networks. It computes the misfit between the predicted and actual target outputs and guides the optimization algorithm to update the weights and biases of the network to minimize the loss. This study uses a customized physics-informed loss function that incorporates the compartmental process-based model into the LSTM cell. The proposed physics-informed loss function is designed to minimize the residual error between the predicted output and the actual target data. It also minimizes the residual error between the predicted output and the compartmental process-based model output. The loss function of the model is represented as follows:

$$\mathcal{L}_{PI}(\theta) = \lambda \mathcal{L}_d(\theta) + (1 - \lambda) \mathcal{L}_{PBM}(\theta)$$  \hspace{1cm} (5.13)$$

Where $\mathcal{L}_d(\theta) = \frac{1}{N} \sum_{x \in N} (\hat{y}(x; \theta; \delta) - y(x))^2$ is the mean square error between the predicted output and actual target data, while $\mathcal{L}_{PBM}(\theta) = \frac{1}{N} \sum_{x \in N} (\hat{y}(x; \theta) - M_{NH_3}(x; p; \delta))^2$ signifies the physics-based discrepancy obtained as the mean square error between the predicted output and the compartmental process-based model output. $\theta$ is the neural network parameter vector, $\delta$ is the set of hyperparameters, $p$ is the compartmental process-based model input parameters, $y(x)$ is the target output (the measured ammonia Flux), $\hat{y}(x; \theta)$ is the...
predicted output by the neural network, and $M_{NH_3}(x;p)$ is the estimated ammonia by the compartmental process-based model. The parameters $\lambda$ and $(1 - \lambda)$ represent the weights of the $\mathcal{L}_d(\theta)$ and $\mathcal{L}_{PBM}(\theta)$ in the PINN model loss function, respectively. The magnitude of $\lambda$ ranges from 0 to 1. Notably, if $\lambda = 1$, the model is entirely data-driven, and if $\lambda = 0$, the model becomes purely physics-based.

The calculated loss propagates backward (backpropagation) in the neural network to compute the gradient needed to update the weights and biases of the LSTM cell using a stochastic optimization method to minimize the error from the loss function. Since the network’s loss function is nonlinear and non-convex, we used a gradient descent minimization algorithm called the ADAM optimizer (Kingma & Ba, 2014). Once the predetermined number of iterations (epochs) is completed, the optimizer stops the training.

$$\theta = \arg\min_{\theta} \mathcal{L}(\theta)$$  \hspace{1cm} (5.14)

**Proposed physics-informed hyperparameters tuning strategy**

The performance of machine learning models mainly depends on the choice of parameters. Machine learning models have two types of parameters: model parameters and hyperparameters. Model parameters refer to the weights and biases that are optimized during the training process using backpropagation. In contrast, the model cannot learn hyperparameters and must be specified prior to training. The model hyperparameters include, but not limited to, the learning rate of ADAM optimizer, the batch size, and hidden dimension of the LSTM layer.

In this study, we investigate physics-informed hyperparameters tuning for two sets of hyperparameters on the performance on the physics-informed LSTM model. The first set of
hyperparameters is related to the LSTM cell architecture and includes the batch size, the hidden dimensions of the LSTM cell, ADAM’s learning rate, and lambda (l). The second set of hyperparameters are the physics-based hyperparameters employed in the compartmental process-based model as a part of the loss function. This set of hyperparameters includes the initial organic nitrogen concentration, diffusion coefficient of ammonia, manure pH, and the mineralization rate constant. These hyperparameters were chosen due to the high uncertainty of their values in the literature.

Hyperparameters can be tuned either by trial and error or by using optimization search algorithms. In the past decade, many studies have been devoted to improving optimization algorithms such as grid search, random search, and Bayesian optimization (Bergstra et al., 2012; Chen et al., 2022; Gelbart et al., 2014). In this study, we employed Optuna, an open-source hyperparameter optimization framework that uses a Tree-structured Parzen Estimator (TPE) algorithm (Akiba et al., 2019). The TPE algorithm iteratively samples hyperparameters from the search space and evaluates their performance using the objective function (typically validation error). Then, as it proceeds, it keeps tuning the search space to focus on the best-performing values of the hyperparameter. We defined the search space for each hyperparameter, including its distribution and range of possible values, as shown in Table 5-1. This process repeats for a predefined number of trials. In this study, 100 trials were implemented. Each trial produced physics-informed validation error, and the lowest of the overall trials produced the best hyperparameters for the final training of the physics-informed LSTM and compartmental process-based models.

The physics-informed validation error used in this study is computed as the average of the residual errors between the actual target and both the LSTM and the process-based model outputs using the validation dataset. Mathematically, the proposed physics-informed
Table 5.1: Hyperparameters and the optimization ranges

<table>
<thead>
<tr>
<th>Type</th>
<th>Hyperparameter</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hyperparameters for the LSTM cell architecture</td>
<td>Batch size</td>
<td>[32, 128]</td>
</tr>
<tr>
<td></td>
<td>Hidden dimension</td>
<td>[1 * 10^{-4}, 1 * 10^{-1}]</td>
</tr>
<tr>
<td></td>
<td>Learning rate</td>
<td>[8, 64]</td>
</tr>
<tr>
<td></td>
<td>Lambda (l)</td>
<td>[0.3, 0.9]</td>
</tr>
<tr>
<td>Physics-based hyperparameters</td>
<td>Initial organic nitrogen concentration</td>
<td>[1, 3.5]</td>
</tr>
<tr>
<td></td>
<td>Diffusion coefficient of ammonia</td>
<td>[1.5 * 10^{-9}, 3 * 10^{-9}]</td>
</tr>
<tr>
<td></td>
<td>pH</td>
<td>[6, 8]</td>
</tr>
<tr>
<td></td>
<td>Mineralization rate constant</td>
<td>[0.006, 0.06]</td>
</tr>
</tbody>
</table>

validation error is:

\[
\delta^* = \text{augmin} \{ \mathcal{E} (\delta) \} = \text{augmin} \left\{ \frac{1}{2} \mathcal{E}_d (\delta) + \frac{1}{2} \mathcal{E}_{PB} (\delta) \right\} \tag{5.15}
\]

Where \( \mathcal{E} (\theta) \) is the physics-informed objective function, \( \mathcal{E}_d (\delta) = \frac{1}{N_{val}} \sum_{x \in N_{val}} (y(x) - \hat{y}(x; \theta; \delta))^2 \) is the mean squared error between the actual target value and the predicted value, and \( \mathcal{E}_{PB} (\delta) = \frac{1}{N_{val}} \sum_{x \in N_{val}} (y(x) - M_{NH_3}(x; p; \delta))^2 \) is the mean squared error between the actual target value and the estimated value by the process-based model for the validation dataset. Finally, \( \delta^* \) is the optimum value for the hyperparameters. The physics-informed LSTM model training can be summarized as described in Figure 5-4.

5.2.2 Data collection and processing

Open-source datasets from two on-farm liquid dairy manure storages (1) a lagoon receiving flushed manure in Indiana, US (FL) (Grant & Boehm, 2020) (2) a steel tank receiving scraped manure (SST) in Switzerland (Kupper et al., 2021), were used in this study. The parameters measured at each farm are mainly local weather parameters, such as ambient air temperature
and wind data, and manure characteristics, such as manure temperature and pH. The data were collected from September 2007 to October 2009 with 30-minute measurement intervals.

The compartmental process-based model was tested on the two datasets. Obtaining meaningful results from process-based models relies heavily on the quality of input parameters (Grassini et al., 2015). In this study, we simulated the process-based model four times. Firstly, we simulated the ammonia emissions from the FL(PB01) and SST (PB02) using the parameters and heat transfer coefficients stated in the literature (Karunarathne et al., 2020; Kupper et al., 2021). Then, we simulated the model again for the FL(UPB01) and the SST (UPB02) using the optimized physics-based hyperparameters generated by Optuna. The input parameters of the process-based model are summarized in Tables 5-2 and 5-3. Table 5-2 lists input parameters based on the manure storage type, management practices, and farm location. Therefore, these parameters differ from one storage structure to another. On the other hand, Table 5-3 lists input parameters that depend only on the manure characteristics and physicochemical properties.
Figure 5.4: A summary of the physics-informed LSTM model training process.
Table 5.2: Location-dependent compartmental process-based model input parameters at the flushed lagoon and scraped steel tank.

<table>
<thead>
<tr>
<th>Data Category</th>
<th>Input</th>
<th>Unit</th>
<th>Flushed Lagoon</th>
<th>Scraped Steel Tank</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Weather (t)</strong></td>
<td>Average air temperature</td>
<td>°C</td>
<td>Measured</td>
<td>Measured</td>
</tr>
<tr>
<td></td>
<td>Total precipitation</td>
<td>cm</td>
<td>Measured</td>
<td>Measured</td>
</tr>
<tr>
<td></td>
<td>Average wind speed</td>
<td>m s⁻¹</td>
<td>Measured</td>
<td>Measured</td>
</tr>
<tr>
<td></td>
<td>Average relative humidity</td>
<td>percent</td>
<td>Measured</td>
<td>Measured</td>
</tr>
<tr>
<td></td>
<td>Standard height at which wind speed is measured</td>
<td>m</td>
<td>1.5</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Wind speed correction height</td>
<td>m</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td><strong>Herd and manure management</strong></td>
<td>Manure storage period</td>
<td>days</td>
<td>81</td>
<td>51</td>
</tr>
<tr>
<td></td>
<td>Number of animals</td>
<td>AU</td>
<td>3450</td>
<td>100</td>
</tr>
<tr>
<td><strong>Dimension of the storage</strong></td>
<td>Total depth/height</td>
<td>m</td>
<td>4.8</td>
<td>4.5</td>
</tr>
<tr>
<td></td>
<td>Surface area open to the air</td>
<td>m⁻²</td>
<td>9744</td>
<td>346</td>
</tr>
<tr>
<td></td>
<td>Depth of residual manure</td>
<td>m</td>
<td>0.3</td>
<td>0.8</td>
</tr>
<tr>
<td><strong>Manure characteristics</strong></td>
<td>Initial organic nitrogen concentration</td>
<td>kg m⁻³</td>
<td>1.39a</td>
<td>2.66</td>
</tr>
<tr>
<td></td>
<td>Initial TAN concentration</td>
<td>kg m⁻³</td>
<td>1.09a</td>
<td>1.39</td>
</tr>
<tr>
<td></td>
<td>pH</td>
<td></td>
<td>7.2</td>
<td>7.5</td>
</tr>
<tr>
<td><strong>Soil characteristics</strong></td>
<td>Average soil temperature</td>
<td>°C</td>
<td>12.9</td>
<td>7.9</td>
</tr>
<tr>
<td></td>
<td>The annual amplitude of surface soil temperature</td>
<td>°C</td>
<td>27.66</td>
<td></td>
</tr>
</tbody>
</table>
Table 5.3: Compartmental process-based model inputs that depend on the manure physiochemical properties.

<table>
<thead>
<tr>
<th>Process</th>
<th>Parameter</th>
<th>Value</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>$NH_3$ volatilization</td>
<td>Roughness height</td>
<td>$8 \times 10^{-5}$</td>
<td>m</td>
</tr>
<tr>
<td></td>
<td>Atmospheric pressure</td>
<td>1</td>
<td>atm</td>
</tr>
<tr>
<td>Diffusion of $NH_3$ in manure</td>
<td>Diffusion coefficient of ammonia</td>
<td>$1.24 \times 10^{-9}$</td>
<td>m$^2$ s$^{-1}$</td>
</tr>
<tr>
<td>Organic nitrogen mineralization</td>
<td>Temperature coefficient</td>
<td>1.036</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Mineralization rate constant</td>
<td>0.01</td>
<td>day$^{-1}$</td>
</tr>
<tr>
<td>Temperature variation in manure</td>
<td>Thermal conductivity of manure</td>
<td>0.09</td>
<td>W m$^{-1}$ °C$^{-1}$</td>
</tr>
<tr>
<td></td>
<td>Heat capacity of manure</td>
<td>1992</td>
<td>J kg$^{-1}$ °C$^{-1}$</td>
</tr>
<tr>
<td></td>
<td>Internal heat generation of stored manure</td>
<td>1.20</td>
<td>W m$^{-3}$</td>
</tr>
<tr>
<td></td>
<td>Thermal diffusivity of soil</td>
<td>0.03</td>
<td>m$^2$ day$^{-1}$</td>
</tr>
<tr>
<td>Evaporation</td>
<td>A parameter depends on the surrounding terrain</td>
<td>0.14</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Bulk aerodynamic transfer coefficient</td>
<td>$2.81 \times 10^{-3}$</td>
<td>-</td>
</tr>
<tr>
<td>Manure and bedding characteristics</td>
<td>Mass of manure produced per animal per day</td>
<td>95</td>
<td>kg</td>
</tr>
<tr>
<td></td>
<td>Mass of bedding used per animal per day – assumed shavings bedding</td>
<td>0.70</td>
<td>kg</td>
</tr>
<tr>
<td></td>
<td>Density of manure</td>
<td>993</td>
<td>kg m$^{-3}$</td>
</tr>
<tr>
<td></td>
<td>Density of bedding</td>
<td>128</td>
<td>kg m$^{-3}$</td>
</tr>
</tbody>
</table>
The physics-informed LSTM uses the ambient air temperature, wind speed, wind direction, relative humidity, rainfall, manure temperature predicted by a physics-informed neural network model (Genedy et al., 2023) and agitation event as input features. The input features used are known to influence ammonia diffusion and volatilization during manure storage (Genedy & Ogejo, 2022; Kupper et al., 2021). The physics-informed LSTM model was trained and validated using the SST dataset. For physics-informed LSTM model partitioned into 45 percent of the SST dataset was used for training, 25 percent for validation, and the remaining 20 percent for testing. In addition, the physics-informed LSTM model was tested on the FL dataset to evaluate the models’ generalization accuracy. All the data were normalized to eliminate bias. Finally, this study was coded in Python programming language with version 3.9. Data visualization, manipulation, and LSTM are implemented with PyTorch, Pandas, NumPy, Pyplot, matplotlib, and Scikit-Learn libraries.

5.2.3 Models’ evaluation

The models were evaluated for predictive power and generalization accuracy (robustness). Predictive power refers to the model’s ability to fit the input features to the observed training data and predict the unobserved data from the same dataset. Thus, the models’ performance during the validation phase determined their predictive power. In contrast, generalization accuracy refers to the model’s ability to meaningfully interpolate sparse and disparate data and make inferences for unseen parameter combinations (Cai et al., 2021; Willcox et al., 2021). Two metrics used to evaluate the model performance included the root mean square error (RMSE) and the mean absolute error (MAE) between the predicted outputs and the actual target.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N} (\hat{y}_i - y_i)^2}{N}}$$  \hspace{1cm} (5.16)
\[ MAE = \frac{\sum_{i=1}^{N} |\hat{y}_i - y_i|}{N} \]  

(5.17)

Where \( y_i \) is the target value, \( \hat{y}_i \) is the predicted value, \( \bar{y} \) is the averaged target value, and \( N \) is the number of observations. The RMSE and MAE measures the error between the target and the predicted output in \( g \, m^{-2}d^{-1} \). Small (in magnitude) RMSE and MAE values during the validation and testing phases are considered good performers.

5.3 Results

5.3.1 Hyperparameters search and sensitivity analysis

The hyperparameter optimization process was conducted for the FL and SST storages. Each optimization consisted of 100 trials and a budget of 50 epochs. The best hyperparameters configuration for the FL and the SST found by Optuna is summarized in Table 5-4. There was a notable difference between the optimized physics-based hyperparameters, and the values found in the literature for the FL (\( pH = 7.2 \), initial organic nitrogen concentration = 1.39 \( kgm^{-3} \), diffusion coefficient of ammonia = \( 1.24 \times 10^{-9} \), and mineralization rate constant = 0.01). The hyperparameter optimization estimated the FL’s manure to be more basic than what is commonly found in the literature with slightly lower initial organic nitrogen concentrations. It also estimated the mineralization rate constant to be lower and the diffusion coefficient higher than the literature. This indicates that relying on the literature to approximate the process-based model parameters and coefficients can be inadequate for the manure storage structure. On the other hand, the optimized physics-based hyperparameters for the SST are similar to the parameters measured on farm as reported by Kupper et al. (2021). Kupper et al. (2021) reported initial organic nitrogen of 2.66 and 7.5 pH, and the optimum initial organic nitrogen and pH were 2.5 and 7.5, respectively. This another indication that
physics-informed hyperparameters optimization framework used can be a simple and feasible alternative for on-farm measurements. Another valuable feature of Optuna is sensitivity analysis. It enabled the visualization of the significance of each hyperparameter in the optimization process. Figure 5-5 presents the relative importance of the hyperparameters used in the physics-informed LSTM model. The x-axis represents the importance score, which quantifies the contribution of each hyperparameter to the overall optimization process. The pH value has the highest importance score at 51 percent, followed by the initial organic nitrogen, lambda, and mineralization constant. These findings suggest that the physics-based hyperparameters have a strong influence on the model’s performance. Hence, adjusting these parameters could enhance the process-based model’s performance.

Table 5.4: The optimum hyperparameters for flushed lagoon dataset and the scraped steel tank dataset.

<table>
<thead>
<tr>
<th>Hyperparameters</th>
<th>Flushed lagoon</th>
<th>Scraped steel tank</th>
</tr>
</thead>
<tbody>
<tr>
<td>Learning rate</td>
<td>0.03</td>
<td>0.01</td>
</tr>
<tr>
<td>Hidden layer dimensions</td>
<td>64</td>
<td>56</td>
</tr>
<tr>
<td>Batch size</td>
<td>128</td>
<td>80</td>
</tr>
<tr>
<td>lambda</td>
<td>0.8</td>
<td>0.8</td>
</tr>
<tr>
<td>pH</td>
<td>8.00</td>
<td>7.5</td>
</tr>
<tr>
<td>Initial organic nitrogen (kg m⁻³)</td>
<td>2.5</td>
<td>2.5</td>
</tr>
<tr>
<td>Diffusion coefficient of ammonia</td>
<td>3 * 10⁻⁹</td>
<td>3 * 10⁻⁹</td>
</tr>
<tr>
<td>Mineralization rate constant</td>
<td>0.06</td>
<td>0.04</td>
</tr>
</tbody>
</table>

Figure 5-6 shows the relationship between the hyperparameters and the physics-informed validation error for FL storage. There appears to be a clear negative relationship between the pH and the validation error. Hence, high pH within the specified range leads to better ammonia emission predictions. This could be attributed to the fact that high pH enhances the transformation of ammonium salt to ammonia. Another notable relationship was between the mineralization rate constant and the model’s performance. High mineralization rate constants increase the validation error, hence, decrease the model’s performance. This
Figure 5.5: Hyperparameter importance for the optimized physics informed LSMT model.

confirms our previous notion that the pH and the mineralization rate constant are important hyperparameters for optimizing the model’s performance. There does not appear to be a clear relationship between the other parameters and the resulting model performance. However, specific hyperparameters’ values can be superior compared to the others. For example, the model performs better when the lambda value > 0.7. Also, the best-performing initial organic nitrogen concentrations range between 1.5 and 2.5 $kgm^{-3}$. Additionally, the models performed poorly at relatively small batch sizes and hidden dimensions, but increasing these parameters has little effect on the model’s performance beyond a certain limit.
Figure 5.6: Slice plots showing the relationship between hyperparameters and the resulting model physics-informed validation error.
Figures 5-7 and 5-8 show the performance of the compartmental process-based model simulating the emissions from the FL and SST before and after the hyperparameters tuning, respectively. It is apparent that the base compartmental process-based model performed better in capturing the trend and the magnitude of the ammonia Flux from the SST compared to the FL. This is due to the comprehensive information provided by Kupper et al. (2021) regarding the system’s parameters and initial conditions. The optimum physics-based hyperparameters for the FL and the SST were then deployed in the compartmental process-based model. For the FL, the compartmental process-based model with the optimum parameters achieved a MAE of 1.5, representing a 27.7 percent improvement over the model’s performance with recommended parameters by Karunarathne et al. (2020). Figure 5-7 shows the compartmental process-based model’s performance before and after optimizing the FL physics-based hyperparameters. Optimizing the physics-based hyperparameters has significantly improved the compartmental process-based model performance. The model was able to capture the emission peaks and overall trend. This confirms that the hyperparameters selected by Optuna are indeed optimal for the compartmental process-based model. The performance of the compartmental process-based model for the SST emissions was the same before and after hyperparameters optimization as shown in Figure 5-8. This is due to the high similarity between the physics parameters measured by Kupper et al. (2021) and that optimized using Optuna. Figure 5-8 shows great agreement between the simulated ammonia emissions with and without the optimized hyperparameters. Overall, our results demonstrate that Optuna can effectively optimize the hyperparameters of physics-informed deep learning models and can significantly improve model performance.
Figure 5.7: The performance of the process-based model using (a) the default physics-based hyperparameters, and (b) the optimized physics-based hyperparameters to simulate the ammonia emissions from a lagoon on a flushed dairy farm.
Figure 5.8: The performance of the process-based model using (a) the default physics-based hyperparameters, and (b) the optimized physics-based hyperparameters to simulate the ammonia emissions from a steel tank on the scraped dairy farm.
5.3.2 Comparison of Physics-Informed LSTM and Process-Based Model

The physics-informed LSTM model was trained using the optimal set of hyperparameters. The evaluation metrics (i.e., MAE and RMSE) for the model are presented in Figure 5-9. As previously stated, optimizing the hyperparameters improved the performance of the compartmental process-based model. The physics-informed LSTM achieved the highest performance when tested data from the same storage used for training, it achieved a RMSE of $1.64 \text{ gm}^{-2}\text{h}^{-1}$ during the SST testing phase. In addition, the physics-informed LSTM model improved the ammonia emission predictions from the SST by 25 percent compared to the compartmental process-based model. In addition, the physics-informed LSTM had notably better performance than the compartmental process-based model during the high emission periods. Figure 5-10 shows that the physics-informed LSTM captured the all the ammonia emission peaks including the peaks that occurred at the SST between February and April of 2015 and 2017. These emission peaks were not captured by the compartmental process-based model although they accounted for 14 percent of the total reported emissions. Overall, the compartmental process-based model underestimated the ammonia emissions and only estimated 77 percent of the actual ammonia lost from the SST. This is a significant drawback for process-based models since it can lead to inadequate design for emission mitigation strategies. By contrast, the physics-informed LSTM provides more accurate and reliable predictions of ammonia emissions, which can help with reducing its environmental impact.
Figure 5.9: The performance evaluation metrics for the trained models during the testing on the data from the lagoon in the flushed farm (FL) and the steel manure storage tank in a scraped farm (SST). (MAPE is the mean absolute error, and RMSE is the root mean squared error).

Figure 5.10: The performance of the physics-informed LSTM at predicting the ammonia flux using the scaped steel tank testing dataset.
Figure 5-11 shows the performance of the physics-informed LSTM model when tested data from storage structures not used for training to assess its generalization accuracy. The physics-informed LSTM model demonstrated a high ability to identify both the trend and magnitude of ammonia emissions from the FL, despite being solely trained on the SST dataset. Notably, the physics-informed LSTM performance exceeded the that of the compartmental process-based model by 13 percent for the FL. Although the physics-informed LSTM model did not entail the physics parameters of the flush manure system. It was able to capture the trend and the pattern of the ammonia emissions associated with a lagoon on a flushed farm although it was trained using data from a tank on a scraped farm. This indicates that physics-informed LSTM model was generalized enough to capture the ammonia emissions from different manure storages at different farm management systems. These findings challenge the conventional assumption that process-based should exhibit higher generalization accuracy across various systems compared to machine learning models. In contrast, our results demonstrate that the physics-informed LSTM model can higher generalization accuracy when tested on different systems that are governed by the same heat and mass transfer equations.
Figure 5.11: The performance of the physics-informed LSTM at predicting the ammonia emissions from flushed lagoon.
The benefit of using a physics-informed LSTM model over process-based modeling is that it eliminates the need for extensive data collection to capture model parameters and heat transfer coefficients. When using a compartmental process-based model to estimate ammonia emissions from the flush manure with limited data on the system parameters, the model performed poorly and did not accurately capture the trend or magnitude of ammonia emission. However, when the same model was used to simulate SST emissions, it had superior performance and accounted for almost all emission peaks just due to the use of appropriate physics-based hyperparameters. By contrast, the physics-informed LSTM models had high performance when tested on the systems that were not used for training (Figure 5-11). Hence, physics-informed LSTM modeling could eliminate the need for measuring system parameters, making it a more convenient option. Another benefit of the LSTM model is that it is computationally efficient. Although the physics-informed LSTM has a slow training process, once trained, it is more computationally efficient than the process-based model. The physics-informed LSTM takes 45 seconds for training one epoch, and less than 2 seconds for predicting one ammonia observation. In comparison, the process-based model takes 5 seconds to simulate one ammonia observation. Therefore, decision support makers and end-users are more likely to find the physics-informed deep learning model appealing and feasible.

5.4 Conclusion

The importance of improving estimates of emissions from animal feeding operations has been emphasized by the National Research Council (NRC 2003), and process-based modeling is a promising tool to achieve this. However, process-based modeling can be challenging for complex systems such as manure storage structures, and existing models suffer from certain limitations that restrict their accuracy. For example, they necessitate an estimate of and
use of manure properties, such as heat and mass transfer coefficients, which can constrain their accuracy. This study presents the physics-informed LSTM framework that embeds that integrates process-based models and machine learning to improve the models’ generalization and accuracy to estimate ammonia losses. In addition, this model was used to optimize some of the process-based model parameters using hyperparameters optimization.

The process-based model parameters and the value of lambda have the greatest impact on the performance of the LSTM model. Optimizing the process-based model parameters has significantly improved performance by reducing the mean absolute error by 35 percent with just four parameter changes. This highlights the importance of parameter engineering in process-based modeling. The performance and generalization accuracy of the physics-informed LSTM model was better than process-based models. The overall RMSE for the physics-informed LSTM model was 1.51 $gm^{-2}h^{-1}$ compared to 2.42 and 2.39 $gm^{-2}h^{-1}$ for the compartmental process-based model before and after hyperparameters optimization, respectively. The physics-informed LSTM model also demonstrated high accuracy when tested on a different system such as the SST with other manure management and storage types. The physics-informed LSTM had higher performance than compartmental process-based model without the need for measuring and accounting for the system’s parameters.

The findings of this study indicate that process-based models tend to underpredict ammonia emissions from manure storage structures, resulting in underestimated nutrient losses during these periods. Using the outputs of these models in decision support tools can lead to inaccurate metrics for assessing sustainable manure management practices. Finally, the outcomes of this study contribute to the field of precision agriculture, specifically designing suitable on-farm strategies to minimize nutrient loss and greenhouse gas emissions during the manure storage periods and improve the accuracy of metrics used to assess sustainable manure management practices. Thus, we propose for future work:
• Use the physics-informed machine learning approach to solve inverse problems to estimate the manure’s heat and mass transfer coefficients in the governing heat and mass transfer equations for different agricultural systems.

• Investigate the applicability of the proposed physics-informed deep learning approach to extended applications such as predicting the green-house gas emissions from the stored manure.

References


Chapter 6

Summary, Conclusions, and Future Research

Ensuring the sustainability of food and agricultural systems is essential to meet the nutritional needs of a growing global population and guarantee food security. Dairy products provide vital proteins and essential dietary nutrients like calcium, vitamin D, and potassium, making them a valuable and affordable food source. However, with the increasing global demand for dairy products, achieving sustainable dairy production practices while protecting natural resources like soil, water, and air can be challenging. One of the main areas that require improved sustainable practices on dairy farms is manure management. Manure is a primary fertilizer source for crop and pasture production in livestock production. Manure management typically involves collecting, storing, and treating manure to allow use as fertilizer at the right time. Storing manure is essential to best management practices on farms, as it provides opportunities to maximize the fertilizer value, reduce handling costs, and minimize its potential to pollute the environment.

During storage, manure undergoes microbial activities and biogeochemical processes that alter its quantity and quality, leading to the formation and release of gaseous constituents such as nitrogen, carbon, and sulfur. Up to 60% of stored manure nitrogen can be lost as ammonia, negatively impacting farm efficiency and contributing to environmental pollution,
including air quality degradation, human health effects, soil acidification, and vegetation toxicity. Losing nitrogen also means reducing the fertilizer and economic value of manure, negatively impacting farm efficiency. There is a need for research at the farm scale to generate information to improve estimates of ammonia emissions from stored manure, as reiterated by the National Research Council in 2003. Making credible estimates at this scale are necessary to improve the understanding of animal production systems and guide developing technologies to manage, control, and regulate emissions from animal feeding operations. This study employed Ogawa samplers to quantify ammonia lost from stored dairy manure at a farm with a scrape manure removal system (Chapter 2). In addition, the local weather parameters, manure temperature and farm management data were collected. The aim was to examine the effects of seasonal weather patterns, manure management, and manure characteristics on ammonia emissions.

During the study period, a total of 134 kg of nitrogen in the form of ammonia was lost, resulting in a substantial revenue loss for the farmer. The study showed that effective manure management practices can significantly reduce ammonia losses and preserve the nitrogen content of manure. The formation of a crust on the surface of the manure was found to be an efficient physical barrier that minimizes ammonia emissions, but this natural crust is disrupted by mixing and pumping, leading to a significant loss of ammonia. About 25% of the ammonia losses occurred during these events. The study identified ambient air temperature, wind speed, and manure temperature as the primary factors affecting ammonia emissions. Ammonia flux trends closely followed manure temperature trends compared to ambient air temperature, and small changes in wind speed had a considerable impact on ammonia flux, with wind speed overriding the effects of other weather conditions.

The study provides valuable insights into the design and development of effective farm manure management practices to minimize nitrogen loss as ammonia. However, this study
alone cannot make up for the deficiencies in accounting for ammonia emissions from different manure storage systems. Therefore, there is a need for a generalized tool to estimate the ammonia losses for different manure storage systems given direct measurements are expensive and require expertise. Process-based modeling is a promising tool to achieve address this need. However, process-based modeling can be challenging for complex systems such as manure storage structures, and the existing models suffer from certain limitations that restrict their accuracy. For example, they necessitate an estimate of and use of manure properties, such as heat and mass transfer coefficients, which can constrain their accuracy. Also, process-based models use surrogates derived from various ambient air temperature averages instead of the manure temperature, which underestimates the contaminants lost to the atmosphere during the manure storage period.

The study found that manure temperature affects biological, physical, and biogeochemical reactions during storage, and there is a high correlation between manure temperature and ammonia emissions. Therefore, there is a need for a better method of estimating manure temperatures to use as inputs for process-based models. Chapter 3 of the study describes the development of machine learning algorithms as a potential substitute for ambient air temperature used as a surrogate for manure temperature in existing on-farm nutrient accounting tools. The models’ performance was evaluated using manure temperature data collected from a concrete storage structure in a scraped farm, a clay-lined pit on a scraped farm, and a concrete tank on a flushed farm. The study found that the machine learning models provided a realistic prediction of stored manure temperature over different seasons of the year, with performance $R^2$ values ranging from 0.970 to 0.998. However, data-driven machine learning approaches are considered "black box" models and may not provide meaningful and physically explainable predictions. Additionally, the models showed poor generalization accuracy when tested on the manure storage structures that were not used for training.
Therefore, this study introduced a new approach to predicting manure temperature during storage using named physics-informed neural networks (PINN) in Chapter 4. PINN embeds the knowledge of heat transfer equations into neural network models to improve the models’ generalization and transportability. The PINN model was trained and validated using data collected from a concrete storage structure in a scraped farm, then tested on data collected from a clay-lined pit on a scraped farm and a concrete tank on a flushed farm. The PINN model showed high predictive ability and generalization accuracy compared with data-driven machine learning and physics-based finite element heat transfer model. It also presented a more accurate, less biased, physically interpretable, and generalized approach to predicting stored manure temperatures. Thus, if used process-based models, the results from this study would improve the accuracy of assessing sustainable manure management practices. The overall accuracy for the PINN model was 82%; however, the model had several limitations, including (1) the physics-based model used to constrain the loss function was simplified and lacked some important influencing factors (such as crusting) because they are not easy to describe by known physical equations. (2) The network’s hyperparameters were tuned by hand, possibly affecting the network’s overall accuracy.

To address these two challenges, we proposed using the physics-informed machine learning approach to solve inverse problems to estimate the manure’s heat and mass transfer coefficients in the process-based models used for estimating the ammonia losses from stored manure.

Chapter 5 presented a physics-informed LSTM framework that embeds that integrates process-based models and machine learning to improve the models’ generalization and accuracy to estimate ammonia losses. In addition, this model was used to optimize some of the process-based model parameters using hyperparameters optimization. The process-based model parameters and the value of lambda have the greatest impact on the performance
of the LSTM model. Optimizing the process-based model parameters has significantly improved performance by reducing the mean absolute error by 35% with just four parameter changes. This highlights the importance of parameter engineering in process-based modeling. The performance and generalization accuracy of the physics-informed LSTM model was better than process-based models. The overall RMSE for the physics-informed LSTM model was $1.51 \, gm^{-2}h^{-1}$ compared to 3.01 and 2.17 for the process-based model before and after hyperparameters optimization, respectively. The physics-informed LSTM model also demonstrated high accuracy when tested on a different system such as the SST with other manure management and storage types. The physics-informed LSTM had almost the same performance as the process-based model without the need for measuring and accounting for the system’s parameters. This work potentially provides good information to guide the design and formulation of effective farm manure management practices that minimize nitrogen lost as ammonia. However, this study alone cannot make up for the deficiencies in accounting for ammonia emissions from manure storage systems in the eastern region. More research is needed to improve the on-farm regulations and laws. Moreover, the data generated can be helpful to those developing and validating mathematical models and software applications to obtain scientifically sound estimates of air emissions from animal feeding operations. The outcomes of this study contribute to the field of precision agriculture and improve the accuracy of metrics used to assess sustainable manure management practices. Finally, future work could investigate the applicability of the proposed physics-informed deep learning approach to: 1. Solve inverse problems to estimate the manure’s heat and mass transfer coefficients in the governing heat and mass transfer equations for different agricultural systems. 2. Extended applications such as predicting the green-house gas emissions from the stored manure.
Appendices
First Appendix: codes for Chapter 3

```python
import pandas as pd
import numpy as np
from matplotlib import pyplot

all_dataset = pd.read_excel('file location')

print(all_dataset.head())

dataset= all_dataset[['predictors', 'actual MT_depth']].copy()

dataset.dropna(inplace=True)

X = dataset[['predictors']]  # predictors

y = dataset[['actual MT_depth ']]  # actual MT_depth

# for data splitting (Training 67% and testing 33%)

from sklearn.model_selection import train_test_split

x=int(round(np.mean(np.array(y))))
```

180
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=x-2, min_samples_leaf=1)

# Feature Scaling

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X_train = sc.fit_transform(X_train)

X_test = sc.transform(X_test)

A.1 Models:

A.1.1 Boosting Trees

import pandas as pd

import numpy as np

import pandas as pd

from matplotlib import pyplot

all_dataset = pd.read_excel('file location')

print(all_dataset.head())

dataset = all_dataset[['predictors', 'actual MT']].copy()

dataset.dropna(inplace=True)
X = dataset["predictors"]

y = dataset["actual MT"]

# for data splitting (Training 67% and testing 33%)

from sklearn.model_selection import train_test_split

x = int(round(np.mean(np.array(y))))

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=(x-2), min_samples_leaf=1)

# Feature Scaling

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X_train = sc.fit_transform(X_train)

X_test = sc.transform(X_test)

# Training the Algorithm

from sklearn.ensemble import GradientBoostingRegressor

regressor = GradientBoostingRegressor(n_estimators=no. of predictors, random_state=x)

regressor.fit(X_train, y_train)

y_pred = regressor.predict(X_test)

# Evaluating the Algorithm
from sklearn import metrics

from sklearn.metrics import r2_score

print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))

print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))

print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))

print('R2:', metrics.r2_score(y_test, y_pred))

diff = (y_test-y_pred)

mb = diff.mean()

print('Mean bias error = ', mbe)

pyplot.scatter(y_test, y_pred)

pyplot.xlabel('True Values')

pyplot.ylabel('Predictions')

fig = pyplot.figure(figsize=(5, 5))

ax = fig.add_subplot(111)

thick = ax.boxplot([y_test, y_pred])

ax.set_xticklabels(['Actual MT', 'Predicted MT'])

pyplot.show()
Other_dataset = pd.read_excel('file_location')

print(Other_dataset.head())

Other_dataset = CT_dataset[['predictors', 'actual MT_depth']].copy()

Other_dataset.dropna(inplace=True)

Other_Test = Other_dataset[['predictors']]  
Other_Target = Other_dataset[['actual MT_depth']]

Other_Test = sc.transform(Other_Test)

Other_pred = regressor.predict(Other_Test)

print('Mean Absolute Error:', metrics.mean_absolute_error(Other_Target, Other_pred))

print('Mean Squared Error:', metrics.mean_squared_error(Other_Target, Other_pred))

print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(Other_Target, Other_pred)))

print('R2:', metrics.r2_score(Other_Target, Other_pred))

pyplot.scatter(Other_Target, Other_pred)

pyplot.xlabel('True Values in other')

pyplot.ylabel('Predictions in other')

fig = pyplot.figure(figsize=(5, 5))

ax = fig.add_subplot(111)
thick = ax.boxplot([Other_Target - Other_pred])

ax.set_xticklabels(['Actual MT in other', 'Predicted MT in other'])

pyplot.show()

A.1.2 Bagged trees

#ensemble Bagging_trees_MT045_CP

import pandas as pd

import numpy as np

from matplotlib import pyplot

all_dataset = pd.read_excel('file location')

print(all_dataset.head())

dataset = all_dataset[['predictors', 'actual MT_depth']].copy()

dataset.dropna(inplace=True)

X = dataset[['parameters']]  

y = dataset[['actual MT_depth']]

# for data splitting (Training 67% and testing 33%)

from sklearn.model_selection import train_test_split

x=int(round(np.mean(np.array(y)))))

185
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=(x-2), min_samples_leaf=1)

# Feature Scaling

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X_train = sc.fit_transform(X_train)

X_test = sc.transform(X_test)

# Training the Algorithm

from sklearn.ensemble import BaggingRegressor

regressor = BaggingRegressor(n_estimators= no. of predictors, random_state=x)

regressor.fit(X_train, y_train)

y_pred = regressor.predict(X_test)

# Evaluating the Algorithm

from sklearn import metrics

from sklearn.metrics import r2_score

print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))

print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))

print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
print('R2:', metrics.r2_score(y_test, y_pred))

diff = (y_test - y_pred)

mb = diff.mean()

print('Mean bias error = ', mb)

pyplot.scatter(y_test, y_pred)

pyplot.xlabel('True Values')

pyplot.ylabel('Predictions')

fig = pyplot.figure(figsize=(5, 5))

ax = fig.add_subplot(111)

thick = ax.boxplot([y_test, y_pred])

ax.set_xticklabels(['Actual MT', 'Predicted MT'])

pyplot.show()

Other_dataset = pd.read_excel('file_location')

print(Other_dataset.head())

Other dataset = CT_dataset["predictors", "actual MT_depth"].copy()

Otherdataset.dropna(inplace=True)

OtherTest = Otherdataset[['predictors']]

187
Other_Target = Otherdataset["actual MT_depth"]

Other_Test = sc.transform(Other_Test)

Other_pred = regressor.predict(Other_Test)

print('Mean Absolute Error:', metrics.mean_absolute_error(Other_Target, Other_pred))

print('Mean Squared Error:', metrics.mean_squared_error(Other_Target, Other_pred))

print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(Other_Target, Other_pred)))

print('R2:', metrics.r2_score(Other_Target, Other_pred))

pyplot.scatter(Other_Target, Other_pred)

pyplot.xlabel('True Values in other')

pyplot.ylabel('Predictions in other')

fig = pyplot.figure(figsize=(5, 5))

ax = fig.add_subplot(111)

thick = ax.boxplot([Other_Target - Other_pred])

ax.set_xticklabels(['Actual MT in other', 'Predicted MT in other'])

pyplot.show()

A.1.3 Random forest

import pandas as pd
import numpy as np

from matplotlib import pyplot

all_dataset = pd.read_excel('file location')

print(all_dataset.head())

dataset = all_dataset["parameters", "actual MT_depth "].copy()

dataset.dropna(inplace=True)

X = dataset["parameters"]

y = dataset["actual MT_depth "]

from sklearn.model_selection import train_test_split

x = int(round(np.mean(np.array(y))))

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=(x-2), min_samples_leaf=1)

# Feature Scaling

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X_train = sc.fit_transform(X_train)

X_test = sc.transform(X_test)

# Training the Algorithm
from sklearn.ensemble import RandomForestRegressor

regressor = RandomForestRegressor(n_estimators=no.of parameters, random_state=x)

regressor.fit(X_train, y_train)

y_pred = regressor.predict(X_test)

# Evaluating the Algorithm

from sklearn import metrics
from sklearn.metrics import r2_score

print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))

print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))

print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))

print('R2:', metrics.r2_score(y_test, y_pred))

pyplot.plot([5,30],[5,30],'-',color='black')

pyplot.scatter(y_test, y_pred, color='grey')

pyplot.xlabel('True Values')

pyplot.ylabel('Predictions')

fig = pyplot.figure(figsize=(5, 5))

ax = fig.add_subplot(111)
thick = ax.boxplot([y_test, y_pred])

ax.set_xticklabels(['Actual MT in CP', 'Predicted MT in CP'])

pyplot.show()
Appendix B

Second Appendix: codes for chapter 4

This code includes the data driven neural networks, the physics-informed neural network and the finite elements heat transfer model.

#Import all the libraries required
#pandas for retrieving the data from the excel sheet
import pandas as pd
#tensorflow is for defining all the neural networks operations
import tensorflow as tf
#numpy is for performing the calculations and the data management
import numpy as np
from numpy import any
#matplot is for data plotting and results visualization
import matplotlib as mpl
import matplotlib.pyplot as plt
from matplotlib.ticker import MultipleLocator, FormatStrFormatter, MaxNLocator
%matplotlib inline
#time for computing the run time
import time
#sklearn is for calculating the evaluation metrics (R2, RMSE, MAE) and data
from sklearn import metrics
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import seaborn as sns
from sklearn.feature_selection import mutual_info_regression

# Math library for calculations
import math
from tabulate import tabulate

The finite-element heat transfer model

Data importing and pre-analysis processing

# Generate a pseudo-random numbers by numpy to have the generate the same random numbers for every operation
np.random.seed(111)
tf.random.set_seed(111)
tf.executing_eagerly()

# Initiate the time variable at the beginning of the computations to calculate the total computation time of the code
start_time = time.time()

# Retrieve the data from the excel sheet
all_dataset = pd.read_excel('used_data/CT_IPNN.xlsx')
dataset = all_dataset[['DOY', 'Mcoverage_m', 'AAT_deg.C', 'Time_month', 'WS_m/s', 'WD_deg', 'SR_W/m2', 'RH_%', 'RF_in', 'MT_deg.C', 'MT_FEM']].copy()
dataset.dropna(inplace=True)

all_dataset_CP = pd.read_excel('used_data/CP_IPNN.xlsx')
dataset_CP=all_dataset_CP[['Time_month','DOY','Mcoverage_m', 'AAT_deg.C',
                           'WS_m/s', 'WD_deg', 'SR_W/m2', 'RH_%', 'RF_in',
                           'MT_deg.C','MT_FEM']].copy()
dataset_CP.dropna(inplace=True)

all_dataset_VTA = pd.read_excel('used_data/VTA_IPNN.xlsx')
dataset_VTA=all_dataset_VTA[['Time_month','DOY','Mcoverage_m', 'AAT_deg.C',
                           'WS_m/s', 'WD_deg', 'SR_W/m2', 'RH_%', 'RF_in',
                           'MT_deg.C','MT_FEM']].copy()
dataset_VTA.dropna(inplace=True)

#features selection
x = dataset[['Time_month','Mcoverage_m', 'AAT_deg.C', 'WS_m/s', 'WD_deg',
               'SR_W/m2', 'RH_%', 'RF_in']]

y = dataset[['MT_deg.C']]
full_data= x.copy()
full_data['MT'] = y
print(full_data.head(2))

#identifying input features having high correlation with the target variable
importances = full_data.drop('MT', axis=1).apply(lambda x: x.corr(full_data.MT))
indices = np.argsort(importances)
print(importances[indices])

#Visualizing the correlations
names=['Month', 'Manure coverage', 'Ambient air temperature',

194
'Wind speed', 'Wind direction', 'Solar radiation',
'Relative humidity', 'Rainfall']

plt.title('Manure temperature')
plt.barh(range(len(indices)), importances[indices], color='k', align='center')
plt.yticks(range(len(indices)), [names[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()

# selecting features that have correlation higher than 20%
for i in range(0, len(indices)):
    if np.abs(importances[i])>0.2:
        print(names[i])

X= full_data[ "Time_month", "Mcoverage_m", "AAT_deg.C", "WS_m/s", "SR_W/m2", "RH_%"]
for i in range(0,len(X.columns)):
    for j in range(0,len(X.columns)):
        if i!=j:
            corr_1=np.abs(X[X.columns[i]].corr(X[X.columns[j]]))
            if corr_1 <0.3:
                print( X.columns[i] , " is not correlated with ", X.columns[j])
            elif corr_1>0.75:
                print( X.columns[i] , " is highly correlated with ", X.columns[j])

# Find the information gain or mutual information of the independent variable
with respect to a target variable
mi = mutual_info_regression(X, y)
mi = pd.Series(mi)
mi.index = X.columns

mi.sort_values(ascending=False).

mi.sort_values(ascending=False).plot.bar(figsize=(10, 4))

<table>
<thead>
<tr>
<th>Time_month</th>
<th>Mcoverage_m</th>
<th>AAT_deg.C</th>
<th>WS_m/s</th>
<th>WD_deg</th>
<th>SR_W/m2</th>
<th>RH_%</th>
<th>RF_in</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>1.230000</td>
<td>8.6</td>
<td>3.5</td>
<td>329.6</td>
<td>250.0</td>
<td>50.0</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1.230417</td>
<td>8.7</td>
<td>3.1</td>
<td>314.3</td>
<td>223.0</td>
<td>47.2</td>
</tr>
</tbody>
</table>

MT

| 0  | 10.654630 |
| 1  | 10.637963 |

| WS_m/s   | -0.229690 |
| SR_W/m2  | -0.227663 |
| RF_in    | -0.177165 |
| WD_deg   | 0.019512  |
| RH_%     | 0.265693  |
| Mcoverage_m | 0.419480 |
| Time_month | 0.675380  |
| AAT_deg.C | 0.719095  |

dtype: float64
Month
Manure coverage
Ambient air temperature
Wind speed
Solar radiation
Relative humidity

Time_month is not correlated with Mcoverage_m
Time_month is not correlated with WS_m/s
Time_month is not correlated with SR_W/m2
Time_month is not correlated with RH_%
Mcoverage_m is not correlated with Time_month
Mcoverage_m is not correlated with AAT_deg.C
Mcoverage_m is not correlated with WS_m/s
Mcoverage_m is not correlated with SR_W/m2
Mcoverage_m is not correlated with RH_%
AAT_deg.C is not correlated with Mcoverage_m
AAT_deg.C is not correlated with WS_m/s
AAT_deg.C is not correlated with SR_W/m2
AAT_deg.C is not correlated with RH_%
WS_m/s is not correlated with Time_month
WS_m/s is not correlated with Mcoverage_m
WS_m/s is not correlated with AAT_deg.C
SR_W/m2 is not correlated with Time_month
SR_W/m2 is not correlated with Mcoverage_m
SR_W/m2 is not correlated with AAT_deg.C
RH_% is not correlated with Time_month
RH_% is not correlated with Mcoverage_m
RH_% is not correlated with AAT_deg.C
#define the input parameters (x), the target values (y) and the physical model's outputs (y_fem)
x = dataset[[ "Time_month", "Mcoverage_m", "AAT_deg.C", "WS_m/s", "SR_W/m2", "RH_%", ]]
y = dataset[[ "MT_deg.C" ]]
y_fem = dataset[[ "MT_FEM" ]]
DOY = dataset[[ "DOY" ]]
month = dataset[[ "Time_month" ]]
AAT = dataset[[ "AAT_deg.C" ]]

x_CP = dataset_CP[[ "Time_month", "Mcoverage_m", "AAT_deg.C", "WS_m/s", "SR_W/m2", "RH_%", ]]
y_CP = dataset_CP[[ "MT_deg.C" ]]
y_FEM_CP = dataset_CP[[ "MT_FEM" ]]
DOY_CP = dataset_CP[[ "DOY" ]]
month_CP = dataset_CP[[ "Time_month" ]]
AAT_CP = dataset_CP[[ "AAT_deg.C" ]]

x_VTA = dataset_VTA[[ "Time_month", "Mcoverage_m", "AAT_deg.C", "WS_m/s", "SR_W/m2", "RH_%", ]]
y_VTA = dataset_VTA[[ "MT_deg.C" ]]
y_FEM_VTA = dataset_VTA[[ "MT_FEM" ]]
DOY_VTA = dataset_VTA[[ "DOY" ]]
month_VTA = dataset_VTA[[ "Time_month" ]]
AAT_VTA = dataset_VTA[[ "AAT_deg.C" ]]

# Split SCT to training and testing datasets
X_train, X_test, y_train, y_test, yFEM_train, yFEM_test, DOY_train, DOY_test, 
m_train, m_test, AAT_train, AAT_test = train_test_split(x, y, y_fem, DOY, 
    month, AAT, test_size=0.33, random_state=0)

# Feature Scaling
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.fit_transform(X_test)
X_CP = sc.fit_transform(x_CP)
X_VTA = sc.fit_transform(x_VTA)
print(len(X_train))
42419

# Define NN hyperparameters layers
num_epochs=3000
layers = np.array([6, 5, 5, 5, 5, 5, 5, 1])

Data-driven neural networks

# generate a pseudo-random numbers by numpy to have the generate the same random
# numbers for every operation
np.random.seed(111)
tf.random.set_seed(111)
tf.executing_eagerly()

# initiate the time variable at the beginning of the computations to calculate
# the total computation time of the code
start_time = time.time()

# initiate the neural network as a class
class Sequentialmodel_DD(tf.Module):
    #Initialize the network parameters (weights and biases)
    def __init__(self, layers, name=None):
        self.W = []  # Weights and biases
        self.parameters = 0  # total number of parameters
        # Assign the initial value of the parameters randomly using HE
        # initialization equation
        for i in range(len(layers)-1):
            input_dim = layers[i]
            output_dim = layers[i+1]

            # HE weights initialization (weight initialization is a procedure
            # to set the starting weights of the neural network to small random
            # values. The optimum initialization method for Relu activation
            # function, the used activation function, is HE weight initialization.

            n = (input_dim)
            std_dv = np.sqrt(2/n)
            w = tf.random.normal([input_dim, output_dim],
                                  dtype = 'float64') * std_dv

            w = tf.Variable(w, trainable=True, name = 'w' + str(i+1))

            b = tf.Variable(tf.cast(tf.zeros([output_dim]), dtype = 'float64'),
                            trainable = True, name = 'b' + str(i+1))
self.W.append(w)
self.W.append(b)
self.parameters += input_dim * output_dim + output_dim

# The definition of the evaluation function that is used to calculate the output
# of each layer by multiplying the layer's input data by the input parameter,
# then adding the biases.

def evaluate(self, X_train):

    a = X_train

    for i in range(len(layers)-2):

        z = tf.add(tf.matmul(a, self.W[2*i]), self.W[2*i+1])  # matmul is tensor multiplication
        a = tf.nn.relu(z)  # activation function

        a = tf.add(tf.matmul(a, self.W[-2]), self.W[-1])
    # For regression, no activation to last layer
    return a

def get_weights(self):

    parameters_1d = []

    for i in range(len(layers)-1):

        parameters_1d.append(self.parameters_1d)

    return parameters_1d
w_1d = tf.reshape(self.W[2*i],[-1])  # flatten weights
b_1d = tf.reshape(self.W[2*i+1],[-1])  # flatten biases

parameters_1d = tf.concat([parameters_1d, w_1d], 0)
# concat weights (merge the parameters array with the weights array)
parameters_1d = tf.concat([parameters_1d, b_1d], 0)
# concat biases (merge the parameters array with the biases array)

return parameters_1d

# Define the function that updates the parameters according to the optimization function

def set_weights(self, parameters):
    for i in range(len(layers)-1):
        shape_w = tf.shape(self.W[2*i]).numpy()
        # shape of the weight tensor
        size_w = tf.size(self.W[2*i]).numpy()
        # size of the weight tensor

        shape_b = tf.shape(self.W[2*i+1]).numpy()
        # shape of the bias tensor
        size_b = tf.size(self.W[2*i+1]).numpy()
        # size of the bias tensor

        pick_w = parameters[0:size_w]
        # pick the weights from the optimization function
        self.W[2*i].assign(tf.reshape(pick_w, shape_w))
        # assign the weights
        parameters = np.delete(parameters, np.arange(size_w), 0)
#delete the old weights

pick_b = parameters[0:size_b]

#pick the biases from the optimization function
self.W[2*i+1].assign(tf.reshape(pick_b,shape_b))

# assign the biases
parameters = np.delete(parameters,np.arange(size_b),0)

#delete the old biases

#The data driven loss function
def loss(self,X_train,y_train):

    loss = y_train-self.evaluate(X_train)
    loss= tf.sqrt(tf.reduce_mean(tf.square(loss)))
    return loss

#Define a function that watch and updates the weights and biases according
#to the optimization function
def adaptive_gradients(self):
    with tf.GradientTape() as tape:
        tape.watch(self.W)
        loss_val = self.loss(X_train,y_train)
        grads = tape.gradient(loss_val,self.W)
        del tape
        return loss_val, grads

#NN with 2 hidden layers with 20 hidden neurons each,
#input layer with 8 neurons and output layer with one output
DDNN = Sequentialmodel_DD(layers)

#define the optimization function
optimizer = tf.keras.optimizers.Adam()

# The number of times to go over the training set using the model to optimize
#the loss function and update the weights accordingly.
loss_graph_DD=[]

#an empty array to save the loss associated with the number of epoch

tf.executing_eagerly()

#loop over the number of epochs- withing each loop, the model calculates the
#loss, optimize the loss function through calculating weights and biases,
#then finally update the weights and biases

for epoch in tf.range(num_epochs):
    loss_value, grads = DDNN.adaptive_gradients()
    loss_graph_DD.append(loss_value.numpy())

    #To display the loss value after every 500 epochs
    if epoch % 500 == 0:
        tf.print('Loss value %d ' % loss_value)
        optimizer.apply_gradients(zip(grads, DDNN.W))

elapsed = time.time() - start_time
print('Training time: %.2f' % (elapsed))

Loss value 19
Loss value 2
Loss value 1
Loss value 1
Loss value 1
Physics-informed neural networks

#generate a pseudo-random numbers by numpy to have the generate the same random
#numbers for every operation
np.random.seed(111)
tf.random.set_seed(111)
tf.executing_eagerly()

#initiate the neural network as a class
class Sequentialmodel_PI(tf.Module):
    #Initialize the network parameters (weights and biases)
    def __init__(self, layers, name=None):

        self.W = []  #Weights and biases
        self.parameters = 0  #total number of parameters
        #Assign the initial value of the parameters randomly using HE
        #initialization equation
        for i in range(len(layers)-1):
            input_dim = layers[i]
            output_dim = layers[i+1]

            #HE weights initialization (weight initialization is a procedure
            #to set the starting weights of the neural network to small random
            #values. The optimum initialization method for Relu activation
            #function, the used activation function, is HE weight initialization.)
n = (input_dim)
std_dv = np.sqrt(2/n)
w = tf.random.normal([input_dim, output_dim],
    dtype = 'float64') * std_dv

w = tf.Variable(w, trainable=True, name = 'w' + str(i+1))

b = tf.Variable(tf.cast(tf.zeros([output_dim]), dtype = 'float64'),
    trainable = True, name = 'b' + str(i+1))

self.W.append(w)
self.W.append(b)
self.parameters += input_dim * output_dim + output_dim

#The definition of the evaluation function that is used to calculate the output
#of each layer by multiplying the layer's input data by the input parameter,
#then adding the biases.

def evaluate(self,X_train):

    a = X_train

    for i in range(len(layers)-2):

        z = tf.add(tf.matmul(a, self.W[2*i]), self.W[2*i+1])
        #matmul is tensor multiplication
        a = tf.nn.relu(z) #activation function
a = tf.add(tf.matmul(a, self.W[-2]), self.W[-1])

# For regression, no activation to last layer
return a

def get_weights(self):

    parameters_1d = []

    for i in range (len(layers)-1):

        w_1d = tf.reshape(self.W[2*i],[-1])  #flatten weights
        b_1d = tf.reshape(self.W[2*i+1],[-1])  #flatten biases

        parameters_1d = tf.concat([parameters_1d, w_1d], 0)  #concat weights (merge the parameters array with the weights array)
        parameters_1d = tf.concat([parameters_1d, b_1d], 0)  #concat biases (merge the parameters array with the biases array)

    return parameters_1d

#Define the function that updates the parameters according to the optimization function

def set_weights(self,parameters):

    for i in range (len(layers)-1):

        shape_w = tf.shape(self.W[2*i]).numpy()  # shape of the weight tensor
        size_w = tf.size(self.W[2*i]).numpy()  #size of the weight tensor
shape_b = tf.shape(self.W[2*i+1]).numpy()
# shape of the bias tensor
size_b = tf.size(self.W[2*i+1]).numpy()
#size of the bias tensor
pick_w = parameters[0:size_w]
#pick the weights from the optimization function
self.W[2*i].assign(tf.reshape(pick_w,shape_w))
# assign the weights
parameters = np.delete(parameters,np.arange(size_w),0)
#delete the old weights

pick_b = parameters[0:size_b]
#pick the biases from the optimization function
self.W[2*i+1].assign(tf.reshape(pick_b,shape_b))
# assign the biases
parameters = np.delete(parameters,np.arange(size_b),0)
#delete the old biases

#The data-driven loss function
def loss_DD(self,X_train,y_train):
    loss_u = y_train-self.evaluate(X_train)
    loss_u = tf.sqrt(tf.reduce_mean(tf.square(loss_u)))
    #no need to take the square root
    return loss_u

#The physical model loss function
def loss_FEM(self, X_train, yFEM_train):
    loss_f = yFEM_train - self.evaluate(X_train)
    loss_f = tf.sqrt(tf.reduce_mean(tf.square(loss_f)))
    # no need to take the square root
    return loss_f

# The total loss function
def loss(self, X_train, y_train, yFEM_train):
    loss_u = self.loss_DD(X_train, y_train)
    loss_f = self.loss_FEM(X_train, yFEM_train)
    loss = (loss_u * W_L + loss_f * (1 - W_L))
    # add a weighting parameter to the physical loss (try and error)
    return loss

# Define a function that watch and updates the weights and biases according
# to the optimization function
def adaptive_gradients(self):
    with tf.GradientTape() as tape:
        tape.watch(self.W)
        loss_val = self.loss(X_train, y_train, yFEM_train)
        grads = tape.gradient(loss_val, self.W)
        del tape
    return loss_val, grads

# The number of times to go over the training set using the model
# to optimize the loss function and update the weights accordingly.
# run it a little bit longer

210
#loss_graph_PI=[]
#an empty array to save the loss associated with the number of epoch
#tf.executing_eagerly()
#W_L=0.6
#print(W_L)

#neurons and output layer with one output
#PINN = Sequentialmodel_PI(layers)
#define the optimization function
#optimizer = tf.keras.optimizers.Adam()
#loop over the number of epochs - within each loop, the model calculates
#the loss, optimize the loss function through calculating weights and biases,
#then finally update the weights and biases
#for epoch in tf.range(num_epochs):
    #loss_value, grads = PINN.adaptive_gradients()
    #loss_graph_PI.append(loss_value.numpy())
    #To display the loss value after every 100 epochs
    #if epoch % 500 == 0:
        #tf.print('Loss value %d ' % loss_value)
        #optimizer.apply_gradients(zip(grads, PINN.W))
#elapsed = time.time() - start_time
#print('Training time: %.2f' % (elapsed))

Simulation Study

#generate a pseudo-random numbers by numpy to have the generate the same random numbers for every operation
np.random.seed(1234)
tf.random.set_seed(1234)
start_time=time.time()

#initiate the neural network as a class
class Sequentialmodel_SS(tf.Module):
    #Initialize the network parameters (weights and biases)
    def __init__(self, layers, name=None):

        self.W = [] #Weights and biases
        self.parameters = 0 #total number of parameters

        #Assign the initial value of the parameters randomly using HE initialization equation
        for i in range(len(layers)-1):
            input_dim = layers[i]
            output_dim = layers[i+1]

            #HE weights initialization (weight initialization is a procedure to set the starting
            weights of the neural network to small random values.
            The optimum initialization method for Relu activation function,
            the used activation function, is HE weight initialization.)
            n = (input_dim)
            std_dv = np.sqrt(2/n)
            w = tf.random.normal([input_dim, output_dim], dtype = 'float64') * std_dv
            w = tf.Variable(w, trainable=True, name = 'w' + str(i+1))

            b = tf.Variable(tf.cast(tf.zeros([output_dim]), dtype = 'float64'),
                            trainable = True, name = 'b' + str(i+1))
self.W.append(w)
self.W.append(b)

self.parameters += input_dim * output_dim + output_dim

#The definition of the evaluation function that is used to calculate the output of each layer by multiplying the layer’s input data by the input parameter, then adding the biases.

def evaluate(self, X_train):

    a = X_train

    for i in range(len(layers)-2):

        z = tf.add(tf.matmul(a, self.W[2*i]), self.W[2*i+1])
        #matmul is tensor multiplication
        a = tf.nn.relu(z)
        #activation function

    a = tf.add(tf.matmul(a, self.W[-2]), self.W[-1])
    # For regression, no activation to last layer
    return a

def get_weights(self):

    parameters_1d = []
for i in range(len(layers)-1):

    w_1d = tf.reshape(self.W[2*i],[-1])  # flatten weights
    b_1d = tf.reshape(self.W[2*i+1],[-1])  # flatten biases

    parameters_1d = tf.concat([parameters_1d, w_1d], 0)  # concat weights (merge the parameters array with the weights array)
    parameters_1d = tf.concat([parameters_1d, b_1d], 0)  # concat biases (merge the parameters array with the biases array)

return parameters_1d

# Define the function that updates the parameters according to the optimization function

def set_weights(self, parameters):
    for i in range(len(layers)-1):
        shape_w = tf.shape(self.W[2*i]).numpy()  # shape of the weight tensor
        size_w = tf.size(self.W[2*i]).numpy()  # size of the weight tensor

        shape_b = tf.shape(self.W[2*i+1]).numpy()  # shape of the bias tensor
        size_b = tf.size(self.W[2*i+1]).numpy()  # size of the bias tensor

        pick_w = parameters[0:size_w]  # pick the weights from the optimization function
        self.W[2*i].assign(tf.reshape(pick_w, shape_w))  # assign the weights
parameters = np.delete(parameters, np.arange(size_w), 0)
# delete the old weights

pick_b = parameters[0:size_b]
# pick the biases from the optimization function
self.W[2*i+1].assign(tf.reshape(pick_b, shape_b))  # assign the biases
parameters = np.delete(parameters, np.arange(size_b), 0)
# delete the old biases

# The data driven loss function

def loss(self, X_train, yFEM_train):
    loss = yFEM_train - self.evaluate(X_train)
    loss = tf.sqrt(tf.reduce_mean(tf.square(loss)))
    return loss

# Define a function that watch and updates the weights and biases according to the optimization function

def adaptive_gradients(self):
    with tf.GradientTape() as tape:
        tape.watch(self.W)
        loss_val = self.loss(X_train, yFEM_train)
        grads = tape.gradient(loss_val, self.W)
    del tape
    return loss_val, grads

SSNN = Sequentialmodel_SS(layers)
# define the optimization function
optimizer = tf.keras.optimizers.Adam()

# The number of times to go over the training set using the model to optimize the
# loss function and update the weights accordingly.
num_epochs=3000

loss_graph_SS=[]  # an empty array to save the loss associated with the number of epoch

# execute eagerly

# loop over the number of epochs - within each loop, the model calculates the loss,
# optimize the loss function through calculating weights and biases,
# then finally update the weights and biases
for epoch in tf.range(num_epochs):
    loss_value_SS, grads = SSNN.adaptive_gradients()
    loss_graph_SS.append(loss_value.numpy())

    # To display the loss value after every 100 epochs
    if epoch % 500 == 0:
        tf.print('Loss value %d ' % loss_value)
        optimizer.apply_gradients(zip(grads, SSNN.W))

elapsed = time.time() - start_time
print(elapsed)
Loss value 1
Loss value 1
Loss value 1
Loss value 1
Loss value 1
Loss value 1
Loss value 1
Loss value 1

64.62540984153748
PINN model training

np.random.seed(111)
tf.random.set_seed(111)
for i in [0.7]:
    start_time=time.time()
    W_L = i
    print(W_L)
    #neurons and output layer with one output
    PINN = Sequential model_PI(layers)

#define the optimization function
    optimizer = tf.keras.optimizers.Adam()
    loss_graph_PI=[]

#loop over the number of epochs- withing each loop, the model calculates
#the loss, optimize the loss function through calculating weights and biases,
#then finally update the weights and biases
    for epoch in tf.range(num_epochs):
        loss_value, grads = PINN.adaptive_gradients()
        loss_graph_PI.append(loss_value.numpy())
        parameters = PINN.get_weights()

    #To display the loss value after every 100 epochs
        if epoch % 500 == 0:
            tf.print('Loss value %d ' % loss_value)
            optimizer.apply_gradients(zip(grads, PINN.W))
        computational_time=time.time()-start_time
        print(computational_time)
#predicting the test dataset in SCT

y_test_pred_DDNN=DDNN.evaluate(X_test)
y_test_pred_PINN=PINN.evaluate(X_test)
y_test_pred_SSNN=SSNN.evaluate(X_test)

0.7

Loss value 18
Loss value 3
Loss value 2
Loss value 2
Loss value 2
Loss value 2

134.46906089782715

#evaluation parameters from the DDNN when tested on the unobserved data of the SCT

print('Mean absolute error from testing the DDNN on SCT:',
      metrics.mean_absolute_error(y_test,y_test_pred_DDNN))

print('Root mean square error from testing the DDNN on SCT:',
      np.sqrt(metrics.mean_squared_error(y_test,y_test_pred_DDNN)))

print('R2 from testing the DDNN on SCT:',
      metrics.r2_score(y_test,y_test_pred_DDNN))

#evaluation parameters from the PINN when tested on the unobserved data of the SCT

print('Mean absolute error from testing the PINN on SCT:',
      metrics.mean_absolute_error(y_test,y_test_pred_PINN))

print('Root mean square error from testing the PINN on SCT:',
      np.sqrt(metrics.mean_squared_error(y_test,y_test_pred_PINN)))

print('R2 from testing the PINN on SCT:',
      metrics.r2_score(y_test,y_test_pred_PINN))
#evaluation parameters from the SSNN when tested on the unobserved data of the SCT
print('Mean absolute error from testing the SSNN on SCT:',
metrics.mean_absolute_error(y_test,y_test_pred_SSNN))
print('Root mean square error from testing the SSNN on SCT:',
np.sqrt(metrics.mean_squared_error(y_test,y_test_pred_SSNN)))
print('R2 from testing the SSNN on SCT:', metrics.r2_score(y_test,y_test_pred_SSNN))

#Predicted MT by the models in SCP
y_CP_pred_DDNN=DDNN.evaluate(X_CP)
y_CP_pred_PINN=PINN.evaluate(X_CP)
y_CP_pred_SSNN=SSNN.evaluate(X_CP)

#evaluation parameters from the DDNN when tested on the data of the SCP
print('Mean absolute error from testing the DDNN on SCP:',
metrics.mean_absolute_error(y_CP,y_CP_pred_DDNN))
print('Root mean square error from testing the DDNN on SCP:',
np.sqrt(metrics.mean_squared_error(y_CP,y_CP_pred_DDNN)))
print('R2 from testing the DDNN on SCP:', metrics.r2_score(y_CP,y_CP_pred_DDNN))

#evaluation parameters from the PINN when tested on the data of the SCP
print('Mean absolute error from testing the PINN on SCP:',
metrics.mean_absolute_error(y_CP,y_CP_pred_PINN))
print('Root mean square error from testing the PINN on SCP:',
np.sqrt(metrics.mean_squared_error(y_CP,y_CP_pred_PINN)))
print('R2 from testing the PINN on SCP:', metrics.r2_score(y_CP,y_CP_pred_PINN))

#evaluation parameters from the PINN when tested on the data of the SCP
print('Mean absolute error from testing the SSNN on SCP:',
      metrics.mean_absolute_error(y_CP,y_CP_pred_SSNN))
print('Root mean square error from testing the SSNN on SCP:',
      np.sqrt(metrics.mean_squared_error(y_CP,y_CP_pred_SSNN)))
print('R2 from testing the SSNN on SCP:', metrics.r2_score(y_CP,y_CP_pred_SSNN))

#Predicted MT in the FCT by the models
y_VTA_pred_DDNN=DDNN.evaluate(X_VTA)
y_VTA_pred_PINN=PINN.evaluate(X_VTA)
y_VTA_pred_SSNN=SSNN.evaluate(X_VTA)

#evaluation parameters from the DDNN when tested on the data of the FCT
print('Mean absolute error from testing the DDNN on FCT:',
      metrics.mean_absolute_error(y_VTA,y_VTA_pred_DDNN))
print('Root mean square error from testing the DDNN on FCT:',
      np.sqrt(metrics.mean_squared_error(y_VTA,y_VTA_pred_DDNN)))
print('R2 from testing the DDNN on FCT:', metrics.r2_score(y_VTA,y_VTA_pred_DDNN))

#evaluation parameters from the PINN when tested on the data of the FCT
print('Mean absolute error from testing the PINN on FCT:',
      metrics.mean_absolute_error(y_VTA,y_VTA_pred_PINN))
print('Root mean square error from testing the PINN on FCT:',
      np.sqrt(metrics.mean_squared_error(y_VTA,y_VTA_pred_PINN)))
R2 from testing the PINN on SCP: {:.4f}, metrics.r2_score(y_VTA, y_VTA_pred_PINN)

# evaluation parameters from the SSNN when tested on the data of the FCT
print('Mean absolute error from testing the SSNN on FCT: {:.4f}',
      metrics.mean_absolute_error(y_VTA, y_VTA_pred_SSNN))
print('Root mean square error from testing the SSNN on SCP: {:.4f}',
      np.sqrt(metrics.mean_squared_error(y_VTA, y_VTA_pred_SSNN)))
print('R2 from testing the SSNN on SCP: {:.4f}, metrics.r2_score(y_VTA, y_VTA_pred_SSNN))

# elapsed = time.time() - start_time
# print('Training time: %.2f' % (elapsed))
Mean absolute error from testing the DDNN on SCT: 0.8803432692850907
Root mean square error from testing the DDNN on SCT: 1.2368838915945244
R2 from testing the DDNN on SCT: 0.9658759618982528
Mean absolute error from testing the PINN on SCT: 1.0099818705974468
Root mean square error from testing the PINN on SCT: 1.3350152766368242
R2 from testing the PINN on SCT: 0.9602465319704442
Mean absolute error from testing the SSNN on SCT: 15.370302393285451
Root mean square error from testing the SSNN on SCT: 16.765415569535527
R2 from testing the SSNN on SCT: -5.269475316014777
Mean absolute error from testing the DDNN on SCP: 3.683233048908973
Root mean square error from testing the DDNN on SCP: 4.614817386830794
R2 from testing the DDNN on SCP: 0.5981173667944146
Mean absolute error from testing the PINN on SCP: 3.4765169923860317
Root mean square error from testing the PINN on SCP: 4.278613189590336
R2 from testing the PINN on SCP: 0.6545412080445729
Mean absolute error from testing the SSNN on SCP: 14.446904240047745

221
Root mean square error from testing the SSNN on SCP: 16.177047968466898
R2 from testing the SSNN on SCP: -3.9384282155052457
Mean absolute error from testing the DDNN on FCT: 3.6932359733322953
Root mean square error from testing the DDNN on FCT: 4.680586186366113
R2 from testing the DDNN on FCT: 0.537025158798065
Mean absolute error from testing the PINN on FCT: 3.7707691899483695
Root mean square error from testing the PINN on SCP: 4.524565339573882
R2 from testing the PINN on SCP: 0.5673759817105758
Mean absolute error from testing the SSNN on FCT: 14.103192852412992
Root mean square error from testing the SSNN on SCP: 15.691394793693293
R2 from testing the SSNN on SCP: -4.2033192852412992

import pickle
pickle.dump(parameters, open('parameters','wb'))

Plotting the loss function optimization

#Plotting the DDNN loss optimization with the epochs
fig,axes = plt.subplots(nrows=1,ncols=2,figsize=(10,4),
                         gridspec_kw={'wspace':0.5,'hspace':0.5})
plt.xlabel('number of epochs')
axes[0].plot(loss_graph_DD, 'k')
axes[1].plot(np.log(loss_graph_DD),'tab:grey')
for ax in axes.flat:
    ax.set(xlabel='number of epochs')
axes[0].set(ylabel="Loss value in the DDNN")
axes[1].set(ylabel="Log loss value in the DDNN")
axes[0].yaxis.set_ticks(np.arange(0, 22, 2))
#Plotting the PINN loss optimization with the epochs
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(10, 4),
                        gridspec_kw={'wspace':0.5,'hspace':0.5})
plt.xlabel('epoch')
axes[0].plot(loss_graph_PI, 'k')
axes[1].plot(np.log(loss_graph_PI), 'tab:grey')
for ax in axes.flat:
    ax.set(xlabel='number of epochs')
axes[0].set(ylabel="Loss value in the PINN")
axes[1].set(ylabel="Log loss value in the PINN")
axes[0].yaxis.set_ticks(np.arange(0, 22, 2))
axes[1].yaxis.set_ticks(np.arange(0, 3.25, 0.25))
axes[0].xaxis.set_ticks(np.arange(0,3500,500))
axes[1].xaxis.set_ticks(np.arange(0,3500,500))
plt.savefig("Loss_optimization_PINN.jpg",bbox_inches='tight',dpi=250)
plt.show()

#Plotting the SSNN loss optimization with the epochs
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(8, 3),
                        gridspec_kw={'wspace':0.5,'hspace':0.5})
223
plt.xlabel('epoch')
axes[0].plot(loss_graph_SS, 'k')
axes[1].plot(np.log(loss_graph_SS),'tab:grey')
for ax in axes.flat:
    ax.set(xlabel='number of epochs')
axes[0].set(ylabel="Loss value in the SSNN")
axes[1].set(ylabel="Log loss value in the SSNN")
# plt.savefig("Loss_optimization_SSNN.jpg",bbox_inches='tight',dpi=250)
plt.show()

# plotting DDNN and PINN in one plot
p1=plt.plot(loss_graph_DD,'k',label='Loss value in the DDNN model')
p2=plt.plot(loss_graph_PI,'k',linestyle='--',label='Loss value in the PINN model')
plt.xlabel('number of epochs', fontsize=15)
plt.ylabel('Loss value', fontsize=15)
plt.legend()
plt.yticks(np.arange(0,22,2),fontsize=12)
plt.xticks(np.arange(0,3500,500),fontsize=12)
# plt.savefig("Loss_optimization_DDNN+PINN.jpg",bbox_inches='tight',dpi=250)
plt.show()
Testing the models on the unobserved data from the SCT

np.random.seed(111)
tf.random.set_seed(111)
y_test_pred_DDNN=DDNN.evaluate(X_test)
y_test_pred_PINN=PINN.evaluate(X_test)

#evaluation parameters from the DDNN when tested on the unobserved data of the SCT
print('Mean absolute error from testing the FEM on SCT:',
      metrics.mean_absolute_error(y_test,yFEM_test))
print('Root mean square error from testing the FEM on SCT:',
      np.sqrt(metrics.mean_squared_error(y_test,yFEM_test)))
print('R2 from testing the FEM on SCT:', metrics.r2_score(y_test,yFEM_test))

#evaluation parameters from the DDNN when tested on the unobserved data of the SCT
print('Mean absolute error from testing the DDNN on SCT:',
    metrics.mean_absolute_error(y_test,y_test_pred_DDNN))
print('Root mean square error from testing the DDNN on SCT:',
    np.sqrt(metrics.mean_squared_error(y_test,y_test_pred_DDNN)))
print('R2 from testing the DDNN on SCT:', metrics.r2_score(y_test,y_test_pred_DDNN))

#evaluation parameters from the PINN when tested on the unobserved data of the SCT
print('Mean absolute error from testing the PINN on SCT:',
    metrics.mean_absolute_error(y_test,y_test_pred_PINN))
print('Root mean square error from testing the PINN on SCT:',
    np.sqrt(metrics.mean_squared_error(y_test,y_test_pred_PINN)))
print('R2 from testing the PINN on SCT:', metrics.r2_score(y_test,y_test_pred_PINN))

#evaluation parameters from the SSNN when tested on the unobserved data of the SCT
print('Mean absolute error from testing the SSNN on SCT:',
    metrics.mean_absolute_error(y_test,y_test_pred_SSNN))
print('Root mean square error from testing the SSNN on SCT:',
    np.sqrt(metrics.mean_squared_error(y_test,y_test_pred_SSNN)))
print('R2 from testing the SSNN on SCT:', metrics.r2_score(y_test,y_test_pred_SSNN))

Mean absolute error from testing the FEM on SCT: 3.3439232429985806
Root mean square error from testing the FEM on SCT: 4.195252335410594
R2 from testing the FEM on SCT: 0.6074285367759455
Mean absolute error from testing the DDNN on SCT: 0.8803432692850903
Root mean square error from testing the DDNN on SCT: 1.2368838915945244
R2 from testing the DDNN on SCT: 0.9658759618982528
Mean absolute error from testing the PINN on SCT: 1.0244972175505438
Root mean square error from testing the PINN on SCT: 1.3553073429464095
R2 from testing the PINN on SCT: 0.9590288519311729
Mean absolute error from testing the SSNN on SCT: 15.370302393285451
Root mean square error from testing the SSNN on SCT: 16.765415569535527
R2 from testing the SSNN on SCT: -5.269475316014777

Plotting the actual vs. predicted manure temperature in SCT by each model

```python
np.random.seed(111)
tf.random.set_seed(111)
fig = plt.figure()
fig.set_size_inches(12, 8)
axes = fig.subplots(nrows=2, ncols=2)
axes[0, 0].scatter(y_test,yFEM_test, marker='o', c='k')
axes[0, 0].plot([0,30],[0,30], linestyle='--', c='grey')
axes[0, 1].scatter(y_test,y_test_pred_DDNN, marker='o', c='k')
axes[0, 1].plot([0,30],[0,30], linestyle='--', c='grey')
axes[1, 0].scatter(y_test,y_test_pred_PINN, marker='o', c='k')
axes[1, 0].plot([0,30],[0,30], linestyle='--', c='grey')
axes[1, 1].scatter(y_test,y_test_pred_SSNN, marker='o', c='k')
axes[1, 1].plot([0,30],[0,30], linestyle='--', c='grey')

axes[0, 0].set_title('Model: FEHT')
axes[0, 1].set_title('Model: DDNN')
axes[1, 0].set_title('Model: PINN')
axes[1, 1].set_title('Model: SSNN')
for ax in fig.axes:
    ax.set(xlabel='Actual Manure temperature in SCT ($^\circ$C)')
```
ylabel='Predicted manure temperature in SCT ($\circ$C)'

plt.subplots_adjust(left=0.1,
               bottom=0.1,
               right=0.9,
               top=0.9,
               wspace=0.5,
               hspace=0.5)

plt.savefig("AcvPr_SCT_each.jpg",bbox_inches='tight',dpi=250)
plt.show()

np.random.seed(111)
tf.random.set_seed(111)

my_array = np.concatenate((DOY_test,y_test,yFEM_test,y_test_pred_DDNN,
               y_test_pred_PINN,y_test_pred_SSNN),axis=1)

DOY_plot= pd.DataFrame(my_array, columns = ['DOY','y_test','y_FEM','y_test_pred_DD',  
'y_test_pred_PI','y_test_pred_SS'])

doy=(DOY_plot['DOY'])
y_test_doy=(DOY_plot['y_test'])
y_test_FEM_doy=(DOY_plot['y_FEM'])
y_test_pred_doy_DD=(DOY_plot['y_test_pred_DD'])
y_test_pred_doy_PI=(DOY_plot['y_test_pred_PI'])
y_test_pred_doy_SS=(DOY_plot['y_test_pred_SS'])

plt.figure(figsize=(18,10))
p1=plt.plot(y_test_doy, linestyle='-',c='black', linewidth=4)
p2=plt.plot(y_test_FEM_doy, linestyle=':',c='black', linewidth=3)
Plotting the actual and the predicted manure temperature vs. DOY

```
np.random.seed(111)
tf.random.set_seed(111)
my_array = np.concatenate((DOY_test,y_test,yFEM_test,y_test_pred_DDNN,
                           plt.figure(figsize=(18,10))
p1=plt.errorbar(y_test_doy[:,0],y_test_doy[:,1],xerr=0, markerfacecolor='black',
                              yerr=y_test_doy_std[:,1], fmt='.', ecolor='black', capsize=3,
                              markeredgecolor='black')
```
p2=plt.errorbar(y_test_FEM_doy[:,0], y_test_FEM_doy[:,1], xerr=0,
            yerr=y_test_FEM_doy_std[:,1], fmt='.', ecolor='grey',
            capsize=3, markerfacecolor='grey', markeredgecolor='grey')

p3=plt.errorbar(y_test_pred_doy_DD[:,0], y_test_pred_doy_DD[:,1], xerr=0,
            yerr=y_test_pred_doy_std_DD[:,1], fmt='.', ecolor='blue',
            capsize=3, markerfacecolor='blue', markeredgecolor='blue')

p4=plt.errorbar(y_test_pred_doy_PI[:,0], y_test_pred_doy_PI[:,1], xerr=0,
            yerr=y_test_pred_doy_std_PI[:,1], fmt='.', ecolor='green',
            capsize=3, markerfacecolor='green', markeredgecolor='green')

p5=plt.errorbar(y_test_pred_doy_SS[:,0], y_test_pred_doy_SS[:,1], xerr=0,
            yerr=y_test_pred_doy_std_SS[:,1], fmt='.', ecolor='red',
            capsize=3, markerfacecolor='green', markeredgecolor='red')

plt.xlabel('DOY', fontsize='15')
plt.ylabel('Manure temperature($^\circ$C)', fontsize='15')
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.legend(handles=[p1,p2,p3,p4,p5], labels=['Actual MT in SCT',
            'Predicted MT in SCT by FEM',
            'Predicted MT in SCT by DDNN',
            'Predicted MT in SCT by PINN',
            'Predicted MT in SCT by SSNN'],
            231
Plotting the actual and the predicted manure temperature vs. Month

```python
my_array = np.concatenate((m_test, y_test, yFEM_test, y_test_pred_DDNN, y_test_pred_PINN, y_test_pred_SSNN), axis=1)
DOY_plot = pd.DataFrame(my_array, columns=['month', 'y_test', 'y_FEM', 'y_test_pred_DD', 'y_test_pred_PI', 'y_test_pred_SS'])
```

```python
plt.figure(figsize=(12,10))
p1=plt.errorbar(y_test_doy[:,0], y_test_doy[:,1], xerr=0, markerfacecolor='black',
                 yerr=y_test_doy_std[:,1], fmt='.', ecolor='black', capsize=3,
                 markeredgecolor='black')
plt.plot(y_test_doy[:,0], y_test_doy[:,1], color='black', linewidth=3)
p2=plt.errorbar(y_test_FEM_doy[:,0], y_test_FEM_doy[:,1], xerr=0,
                 yerr=y_test_FEM_doy_std[:,1], fmt='o', ecolor='grey',
                 capsize=3, markerfacecolor='grey', markeredgecolor='grey')
plt.plot(y_test_FEM_doy[:,0], y_test_FEM_doy[:,1], color='grey', linewidth=3)
p3=plt.errorbar(y_test_pred_doy_DD[:,0], y_test_pred_doy_DD[:,1], xerr=0,
                 yerr=y_test_pred_doy_std_DD[:,1], fmt='x', ecolor='blue',
                 capsize=3, markerfacecolor='blue', markeredgecolor='blue')
plt.plot(y_test_pred_doy_DD[:,0], y_test_pred_doy_DD[:,1], color='blue', linewidth=3)
p4=plt.errorbar(y_test_pred_doy_PI[:,0], y_test_pred_doy_PI[:,1], xerr=0,
                 yerr=y_test_pred_doy_std_PI[:,1], fmt='s', ecolor='green',
                 capsize=3, markerfacecolor='green', markeredgecolor='green')
```
```python
plt.plot(y_test_pred_doy_PI[:,0], y_test_pred_doy_PI[:,1], color='green', linewidth=3)
p5 = plt.errorbar(y_test_pred_doy_SS[:,0], y_test_pred_doy_SS[:,1], xerr=0, yerr=y_test_pred_doy_std_SS[:,1], fmt='.', ecolor='red', capsize=3, markerfacecolor='red', markeredgecolor='red')
plt.plot(y_test_pred_doy_SS[:,0], y_test_pred_doy_SS[:,1], color='red', linewidth=3)
plt.xlabel('Month', fontsize='15')
plt.ylabel('Manure temperature(°C)', fontsize='15')
plt.xticks([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12], fontsize=15)
plt.yticks(fontsize=15)
plt.legend(handles=[p1, p2, p3, p4, p5], labels=['Actual MT in SCT', 'Predicted MT in SCT by FEHT', 'Predicted MT in SCT by DDNN', 'Predicted MT in SCT by PINN', 'Predicted MT in SCT by SSNN'], loc='best', bbox_to_anchor=(1,1), frameon=True, fontsize='15')
plt.savefig('CT_month_ALL.jpg', bbox_inches='tight', dpi=250)
plt.savefig('SCT_month_ALL.jpg', bbox_inches='tight', dpi=250)
plt.show()

# print monthly averages

table={'Month':y_test_doy[:,0], 'Actual MT':y_test_doy[:,1], 'FEM':y_test_FEM_doy[:,1], 'DDNN':y_test_pred_doy_DD[:,1], 'PINN':y_test_pred_doy_PI[:,1], 'SSNN':y_test_pred_doy_SS[:,1]}
print(tabulate(table, headers='keys', tablefmt='fancy_grid'))
```
table1={'Month':y_test_doy[:,0],
   'Actual MT':(y_test_doy[:,1]/y_test_doy[:,1]),
   'FEM':(y_test_FEM_doy[:,1]/y_test_doy[:,1]),
   'DDNN':(y_test_pred_doy_DD[:,1]/y_test_doy[:,1]),
   'PINN':(y_test_pred_doy_PI[:,1]/y_test_doy[:,1]),
   'SSNN':(y_test_pred_doy_SS[:,1]/y_test_doy[:,1])}
print(tabulate(table1, headers='keys', tablefmt='fancy_grid'))

print(1-(y_test_pred_doy_PI[:,1]/y_test_pred_doy_DD[:,1]))
np.random.seed(111)
tf.random.set_seed(111)
my_array = np.concatenate((DOY_test,y_test,yFEM_test,
   y_test_pred_DDNN,y_test_pred_PINN,y_test_pred_SSNN,AAT_test),axis=1)
DOY_plot= pd.DataFrame(my_array, columns = ['DOY','y_test','y_FEM',
   'y_test_pred_DD','y_test_pred_PI','y_test_pred_SS','AAT'])

y_test_doy=(DOY_plot.groupby('DOY', as_index=False)['y_test'] .mean())
y_test_FEM_doy=(DOY_plot.groupby('DOY', as_index=False)['y_FEM'].mean())
```python
y_test_pred_doy_DD = (DOY_plot.groupby('DOY', as_index=False)['y_test_pred_DD'].mean())
y_test_pred_doy_PI = (DOY_plot.groupby('DOY', as_index=False)['y_test_pred_PI'].mean())
y_test_pred_doy_SS = (DOY_plot.groupby('DOY', as_index=False)['y_test_pred_SS'].mean())
AAT_doy = (DOY_plot.groupby('DOY', as_index=False)['AAT'].mean())

fig = plt.figure()
fig.set_size_inches(15,10)
axs = fig.subplots(nrows=2, ncols=2)

axs[0, 0].plot(y_test_doy[:,0], y_test_doy[:,1], color='k')
axs[0, 0].plot(y_test_doy[:,0], y_test_FEM_doy[:,1], color='b')
axs[0, 0].plot(y_test_doy[:,0], AAT_doy[:,1], color='k', linestyle=':')
axs[0, 0].set_title('Model: FEHT')

axs[0, 1].plot(y_test_doy[:,0], y_test_doy[:,1], color='k')
axs[0, 1].plot(y_test_doy[:,0], y_test_pred_doy_DD[:,1], color='b')
axs[0, 1].plot(y_test_doy[:,0], AAT_doy[:,1], color='k', linestyle=':')
axs[0, 1].set_title('Model: DDNN')

axs[1, 0].plot(y_test_doy[:,0], y_test_doy[:,1], color='k')
axs[1, 0].plot(y_test_doy[:,0], y_test_pred_doy_PI[:,1], color='b')
axs[1, 0].plot(y_test_doy[:,0], AAT_doy[:,1], color='k', linestyle=':')
axs[1, 0].set_title('Model: PINN')

axs[1, 1].plot(y_test_doy[:,0], y_test_doy[:,1], color='k', label='Actual manure temperature in SCT')
axs[1, 1].plot(y_test_doy[:,0], y_test_pred_doy_SS[:,1], color='b',
```

235
predicted manure temperature in SCT')
axs[1, 1].plot(y_test_doy[:, 0], AAT_doy[:, 1], color='k', linestyle=':',
label='Ambient air temperature in SCT')
axs[1, 1].set_title('Model: SSNN')

lines = []
labels = []
for ax in fig.axes:
    axLine, axLabel = ax.get_legend_handles_labels()
    lines.extend(axLine)
    labels.extend(axLabel)
fig.legend(lines, labels, loc='lower center', fontsize=16)
for ax in fig.axes:
    ax.set(xlabel='DOY', ylabel='Manure temperature(\degree C)')
    for item in ([ax.title, ax.xaxis.label, ax.yaxis.label] +
                 ax.get_xticklabels() + ax.get_yticklabels()):
        item.set_fontsize(14)
plt.subplots_adjust(left=0.1,
                    bottom=0.18,
                    right=0.9,
                    top=0.9,
                    wspace=0.4,
                    hspace=0.4)
plt.savefig("CT_DOY_each.jpg", bbox_inches='tight', dpi=250)
plt.show()
# box_plots
np.random.seed(111)
tf.random.set_seed(111)
my_array = np.concatenate((DOY_test,y_test,yFEM_test,
y_test_pred_DDNN,y_test_pred_PINN,y_test_pred_SSNN,AAT_test),axis=1)
DOY_plot= pd.DataFrame(my_array, columns = ['DOY','y_test','y_FEM','y_test_pred_DD',
'y_test_pred_PI','y_test_pred_SS','AAT'])

x=DOY_plot['DOY']
fig,ax = plt.subplots(figsize=(9,7))
medianprops = {'color': 'black', 'linewidth': 2}
boxprops = {'color': 'black', 'linestyle': '-'}
whiskerprops = {'color': 'black', 'linestyle': '-'}
capprops = {'color': 'black', 'linestyle': '-'}
flierprops = {'color': 'black', 'marker': 'x'}
bp = ax.boxplot(x=[y_test_doy[:,1],y_test_FEM_doy[:,1],
y_test_pred_doy_DD[:,1],y_test_pred_doy_PI[:,1],
y_test_pred_doy_SS[:,1]],
                positions=[1,2,3,4,5])
ax.set(xlabel='Actual and predicted manure temperature in SCT',
ylabel='Manure temperature($^\circ$C)')
plt.savefig("CT_DOY_bp_each.jpg",bbox_inches='tight',dpi=250)
plt.show()
np.random.seed(111)
tf.random.set_seed(111)
my_array = np.concatenate((DOY_test,y_test,yFEM_test,
y_test_pred_DDNN,y_test_pred_PINN,y_test_pred_SSNN,AAT_test),axis=1)
DOY_plot = pd.DataFrame(my_array, columns = ['DOY', 'y_test', 'y_FEM', 'y_test_pred_DD', 'y_test_pred_PI', 'y_test_pred_SS', 'AAT'])

y_test_doy = (DOY_plot.groupby('DOY', as_index=False)['y_test'].mean())
y_test_FEM_doy = (DOY_plot.groupby('DOY', as_index=False)['y_FEM'].mean())
y_test_pred_doy_DD = (DOY_plot.groupby('DOY', as_index=False)['y_test_pred_DD'].mean())
y_test_pred_doy_PI = (DOY_plot.groupby('DOY', as_index=False)['y_test_pred_PI'].mean())
y_test_pred_doy_SS = (DOY_plot.groupby('DOY', as_index=False)['y_test_pred_SS'].mean())
AAT_doy = (DOY_plot.groupby('DOY', as_index=False)['AAT'].mean())

y_test_diff = y_test_doy - y_test_doy
y_test_FEM_doy_diff = y_test_doy - y_test_FEM_doy
y_test_pred_doy_DD_diff = y_test_doy - y_test_pred_doy_DD
y_test_pred_doy_PI_diff = y_test_doy - y_test_pred_doy_PI
y_test_pred_doy_SS_diff = y_test_doy - y_test_pred_doy_SS

fig = plt.figure()
fig.set_size_inches(12, 6)
axs = fig.subplots(nrows=2, ncols=2)
axs[0, 0].plot(y_test_doy[:, 0], y_test_FEM_doy_diff[:, 1], color='b')
axs[0, 0].set_title('Model: FEHT')

axs[0, 1].plot(y_test_doy[:, 0], y_test_pred_doy_DD_diff[:, 1], color='b')
axs[0, 1].set_title('Model: DDNN')

axs[1, 0].plot(y_test_doy[:, 0], y_test_pred_doy_PI_diff[:, 1], color='b')

238
axs[1, 0].set_title('Model: PINN')

axs[1, 1].plot(y_test_doy[:,0], y_test_pred_doy_SS_diff[:,1], color='b', label='difference between predicted and actual manure temperature in SCT')
axs[1, 1].set_title('Model: SSNN')

lines = []
labels = []
for ax in fig.axes:
    axLine, axLabel = ax.get_legend_handles_labels()
    lines.extend(axLine)
    labels.extend(axLabel)
fig.legend(lines, labels, loc='upper right')
for ax in fig.axes:
    ax.set(xlabel='DOY', ylabel='Manure temperature($^\circ$C)')
plt.subplots_adjust(left=0.1, bottom=0.1, right=0.73, top=0.9, wspace=0.5, hspace=0.5)
plt.savefig("diff_DOY_each_SCT.jpg", bbox_inches='tight', dpi=250)
plt.show()
plt.plot(y_test_doy[:,0],y_test_FEM_doy_diff[:,1], color='k',label='FEHT')
plt.plot(y_test_doy[:,0],y_test_pred_doy_DD_diff[:,1], color='k',linestyle='--',label='DDNN')
plt.plot(y_test_doy[:,0],y_test_pred_doy_PI_diff[:,1], color='b',label='PINN')
plt.plot(y_test_doy[:,0],y_test_pred_doy_SS_diff[:,1], color='k',linestyle=':',label='SSNN')
plt.plot(y_test_doy[:,0],y_test_diff[:,1], color='r')
plt.xlabel('DOY')
plt.ylabel('Actual-predicted MT in SCP')
plt.legend()
plt.savefig("diff_DOY_SCT.jpg",bbox_inches='tight',dpi=250)
plt.show()

#stats difference
print('Mean difference between actual MT and FEM predictions in SCT:', np.average(y_test_FEM_doy_diff[:,1]))
print('minimum difference between actual MT and FEM predictions in SCT:', np.min(y_test_FEM_doy_diff[:,1]))
print('maximum difference between actual MT and FEM predictions in SCT:', np.max(y_test_FEM_doy_diff[:,1]))
print('Mean difference between actual MT and DDNN predictions in SCT:', np.average(y_test_pred_doy_DD_diff[:,1]))
print('minimum difference between actual MT and DDNN predictions in SCT:', np.min(y_test_pred_doy_DD_diff[:,1]))
print('maximum difference between actual MT and DDNN predictions in SCT:', np.max(y_test_pred_doy_DD_diff[:,1]))
np.max(y_test_pred_doy_DD_diff[:,1])
print('Mean difference between actual MT and PINN predictions in SCT:',
np.average(y_test_pred_doy_PI_diff[:,1]))
print('minimum difference between actual MT and PINN predictions in SCT:',
np.min(y_test_pred_doy_PI_diff[:,1]))
print('maximum difference between actual MT and PINN predictions in SCT:',
np.max(y_test_pred_doy_PI_diff[:,1]))
print('Mean difference between actual MT and SSNN predictions in SCT:',
np.average(y_test_pred_doy_SS_diff[:,1]))
print('minimum difference between actual MT and SSNN predictions in SCT:',
np.min(y_test_pred_doy_SS_diff[:,1]))
print('maximum difference between actual MT and SSNN predictions in SCT:',
np.max(y_test_pred_doy_SS_diff[:,1]))

my_array = np.concatenate((m_test,y_test,yFEM_test,
y_test_pred_DDNN,y_test_pred_PINN,y_test_pred_SSNN),axis=1)

DOY_plot= pd.DataFrame(my_array, columns = ['month','y_test','y_FEM',
'y_test_pred_DD','y_test_pred_PI','y_test_pred_SS'])

y_test_doy=(DOY_plot.groupby('month', as_index=False)['y_test'].mean())
y_test_FEM_doy=(DOY_plot.groupby('month', as_index=False)['y_FEM'].mean())
y_test_pred_doy_DD=(DOY_plot.groupby('month', as_index=False)['y_test_pred_DD'].mean())
y_test_pred_doy_PI=(DOY_plot.groupby('month', as_index=False)['y_test_pred_PI'].mean())
y_test_pred_doy_SS=(DOY_plot.groupby('month', as_index=False)['y_test_pred_SS'].mean())

y_test_diff=y_test_doy-y_test_doy
```python
y_test_FEM_doy_diff = y_test_doy - y_test_FEM_doy
y_test_pred_doy_DD_diff = y_test_doy - y_test_pred_doy_DD
y_test_pred_doy_PI_diff = y_test_doy - y_test_pred_doy_PI
y_test_pred_doy_SS_diff = y_test_doy - y_test_pred_doy_SS

plt.figure(figsize=(12,6))
plt.plot(y_test_doy[:,0], y_test_FEM_doy_diff[:,1], color='k', label='FEHT')
plt.plot(y_test_doy[:,0], y_test_pred_doy_DD_diff[:,1], color='k', linestyle='--', label='DDNN')
plt.plot(y_test_doy[:,0], y_test_pred_doy_PI_diff[:,1], color='b', label='PINN')
plt.plot(y_test_doy[:,0], y_test_pred_doy_SS_diff[:,1], color='k', linestyle=':', label='SSNN')
plt.plot(y_test_doy[:,0], y_test_diff[:,1], color='r')
plt.xlabel('Month')
plt.ylabel('Actual-predicted MT in SCT')
plt.yticks([-10,-5,0,5,10])
plt.legend()
plt.savefig("diff_Month_SCT.jpg", bbox_inches='tight', dpi=250)
plt.show()

# stats difference
print('Mean difference between actual MT and FEM predictions in SCT:',
      np.average(y_test_FEM_doy_diff[:,1]))
print('minimum difference between actual MT and FEM predictions in SCT:',
      np.min(y_test_FEM_doy_diff[:,1]))
print('maximum difference between actual MT and FEM predictions in SCT:',
```
np.max(y_test_FEM_doy_diff[:,1])
print('Mean difference between actual MT and DDNN predictions in SCT:',
np.average(y_test_pred_doy_DD_diff[:,1]))
print('minimum difference between actual MT and DDNN predictions in SCT:',
np.min(y_test_pred_doy_DD_diff[:,1]))
print('maximum difference between actual MT and DDNN predictions in SCT:',
np.max(y_test_pred_doy_DD_diff[:,1]))
print('Mean difference between actual MT and PINN predictions in SCT:',
np.average(y_test_pred_doy_PI_diff[:,1]))
print('minimum difference between actual MT and PINN predictions in SCT:',
np.min(y_test_pred_doy_PI_diff[:,1]))
print('maximum difference between actual MT and PINN predictions in SCT:',
np.max(y_test_pred_doy_PI_diff[:,1]))
print('Mean difference between actual MT and SSNN predictions in SCT:',
np.average(y_test_pred_doy_SS_diff[:,1]))
print('minimum difference between actual MT and SSNN predictions in SCT:',
np.min(y_test_pred_doy_SS_diff[:,1]))
print('maximum difference between actual MT and SSNN predictions in SCT:',
np.max(y_test_pred_doy_SS_diff[:,1]))

Testing the models on the SCP data

np.random.seed(111)
tf.random.set_seed(111)
#Predicted MT by the models in SCP
y_CP_pred_DDNN=DDNN.evaluate(X_CP)
y_CP_pred_PINN=PINN.evaluate(X_CP)
y_CP_pred_SSNN = SSNN.evaluate(X_CP)

# evaluation parameters from the FEM when tested on the data of the SCP
print('Mean absolute error from testing the FEM on SCP:',
      metrics.mean_absolute_error(y_CP, y_FEM_CP))
print('Root mean square error from testing the FEM on SCP:',
      np.sqrt(metrics.mean_squared_error(y_CP, y_FEM_CP)))
print('R2 from testing the FEM on SCP:', metrics.r2_score(y_CP, y_FEM_CP))

# evaluation parameters from the DDNN when tested on the data of the SCP
print('Mean absolute error from testing the DDNN on SCP:',
      metrics.mean_absolute_error(y_CP, y_CP_pred_DDNN))
print('Root mean square error from testing the DDNN on SCP:',
      np.sqrt(metrics.mean_squared_error(y_CP, y_CP_pred_DDNN)))
print('R2 from testing the DDNN on SCP:', metrics.r2_score(y_CP, y_CP_pred_DDNN))

# evaluation parameters from the PINN when tested on the data of the SCP
print('Mean absolute error from testing the PINN on SCP:',
      metrics.mean_absolute_error(y_CP, y_CP_pred_PINN))
print('Root mean square error from testing the PINN on SCP:',
      np.sqrt(metrics.mean_squared_error(y_CP, y_CP_pred_PINN)))
print('R2 from testing the PINN on SCP:', metrics.r2_score(y_CP, y_CP_pred_PINN))

# evaluation parameters from the SSNN when tested on the data of the SCP
print('Mean absolute error from testing the SSNN on SCP:',
      metrics.mean_absolute_error(y_CP, y_CP_pred_SSNN))

print('Root mean square error from testing the SSNN on SCP:',
    np.sqrt(metrics.mean_squared_error(y_CP,y_CP_pred_SSNN)))
print('R2 from testing the SSNN on SCP:', metrics.r2_score(y_CP,y_CP_pred_SSNN))

Plotting the actual vs. predicted manure temperature in SCP by each model

np.random.seed(111)
tf.random.set_seed(111)
fig = plt.figure()
fig.set_size_inches(12, 8)
axes = fig.subplots(nrows=2, ncols=2)
axes[0, 0].scatter(y_CP,y_FEM_CP, marker='o', c='k')
axes[0, 0].plot([0,30],[0,30], linestyle='--', c='grey')
axes[0, 1].scatter(y_CP,y_CP_pred_DDNN, marker='o', c='k')
axes[0, 1].plot([0,30],[0,30], linestyle='--', c='grey')
axes[1, 0].scatter(y_CP,y_CP_pred_PINN, marker='o', c='k')
axes[1, 0].plot([0,30],[0,30], linestyle='--', c='grey')
axes[1, 1].scatter(y_CP,y_CP_pred_SSNN, marker='o', c='k')
axes[1, 1].plot([0,30],[0,30], linestyle='--', c='grey')
axes[0, 0].set_title('Model: FEHT')
axes[0, 1].set_title('Model: DDNN')
axes[1, 0].set_title('Model: PINN')
axes[1, 1].set_title('Model: SSNN')
for ax in fig.axes:
    ax.set(xlabel='Actual Manure temperature in SCP ($^\circ$C)',
           ylabel='Predicted manure temperature in SCP ($^\circ$C)')
plt.subplots_adjust(left=0.1,
bottom=0.1,
right=0.9,
top=0.9,
wspace=0.5,
hspace=0.5)
plt.savefig("AcvPr_CP_each.jpg",bbox_inches='tight',dpi=250)
plt.show()
np.random.seed(111)
tf.random.set_seed(111)

my_array = np.concatenate((DOY_CP,y_CP,y_FEM_CP,y_CP_pred_DDNN,
y_CP_pred_PINN,y_CP_pred_SSNN),axis=1)
DOY_plot= pd.DataFrame(my_array, columns = ['DOY','y_test','y_FEM','y_test_pred_DD',
'y_test_pred_PI','y_test_pred_SS'])

doy=(DOY_plot['DOY'])
y_test_doy=(DOY_plot['y_test'])
y_test_FEM_doy=(DOY_plot['y_FEM'])
y_test_pred_doy_DD=(DOY_plot['y_test_pred_DD'])
y_test_pred_doy_PI=(DOY_plot['y_test_pred_PI'])
y_test_pred_doy_SS=(DOY_plot['y_test_pred_SS'])

plt.figure(figsize=(18,10))
p1=plt.plot(y_test_doy, linestyle='-',c='black', linewidth=4)
p2=plt.plot(y_test_FEM_doy, linestyle=':',c='black', linewidth=3)
p3=plt.plot(y_test_pred_doy_DD, linestyle='-',c='grey', linewidth=3)
p4=plt.plot(y_test_pred_doy_PI, linestyle='-',c='blue', linewidth=3)
Plotting the actual and the predicted manure temperature vs. DOY for the FCP

np.random.seed(111)
tf.random.set_seed(111)
my_array = np.concatenate((DOY_CP,y_CP,y_FEM_CP,y_CP_pred_DDNN,
y_CP_pred_PINN,y_CP_pred_SSNN),axis=1)
DOY_plot= pd.DataFrame(my_array, columns = ['DOY','y_test','y_FEM','y_test_pred_DD',
'y_test_pred_PI','y_test_pred_SS'])

y_test_doy=(DOY_plot.groupby('DOY', as_index=False)[('y_test')].mean())
y_test_FEM_doy=(DOY_plot.groupby('DOY', as_index=False)[('y_FEM')].mean())
y_test_pred_doy_DD=(DOY_plot.groupby('DOY', as_index=False)[('y_test_pred_DD')].mean())
y_test_pred_doy_PI=(DOY_plot.groupby('DOY', as_index=False)[('y_test_pred_PI')].mean())
y_test_pred_doy_SS=(DOY_plot.groupby('DOY', as_index=False)[('y_test_pred_SS')].mean())
```python
y_test_pred_doy_PI = (DOY_plot.groupby('DOY', as_index=False)['y_test_pred_PI'].mean())
y_test_pred_doy_SS = (DOY_plot.groupby('DOY', as_index=False)['y_test_pred_SS'].mean())

y_test_doy_std = (DOY_plot.groupby('DOY', as_index=False)['y_test'].std())
y_test_FEM_doy_std = (DOY_plot.groupby('DOY', as_index=False)['y_FEM'].std())
y_test_pred_doy_std_DD = (DOY_plot.groupby('DOY', as_index=False)['y_test_pred_DD'].std())
y_test_pred_doy_std_PI = (DOY_plot.groupby('DOY', as_index=False)['y_test_pred_PI'].std())
y_test_pred_doy_std_SS = (DOY_plot.groupby('DOY', as_index=False)['y_test_pred_SS'].std())

plt.figure(figsize=(18,10))
p1 = plt.errorbar(y_test_doy[:,0], y_test_doy[:,1], xerr=0, markerfacecolor='black',
                  yerr=y_test_doy_std[:,1], fmt='.', ecolor='black', capsize=3,
                  markeredgecolor='black')

p2 = plt.errorbar(y_test_FEM_doy[:,0], y_test_FEM_doy[:,1], xerr=0,
                   yerr=y_test_FEM_doy_std[:,1], fmt='.', ecolor='grey',
                   capsize=3, markerfacecolor='grey', markeredgecolor='grey')

p3 = plt.errorbar(y_test_pred_doy_DD[:,0], y_test_pred_doy_DD[:,1], xerr=0,
                   yerr=y_test_pred_doy_std_DD[:,1], fmt='.', ecolor='blue',
                   capsize=3, markerfacecolor='blue', markeredgecolor='blue')

p4 = plt.errorbar(y_test_pred_doy_PI[:,0], y_test_pred_doy_PI[:,1], xerr=0,
                   yerr=y_test_pred_doy_std_PI[:,1], fmt='.', ecolor='green',
                   capsize=3, markerfacecolor='green', markeredgecolor='green')
```
plt.errorbar(y_test_pred_doy_SS[:,0], y_test_pred_doy_SS[:,1], xerr=0,
    yerr=y_test_pred_doy_std_SS[:,1], fmt='.', ecolor='red',
    capsize=3, markerfacecolor='green', markeredgecolor='red')

plt.xlabel('DOY', fontsize='15')
plt.ylabel('Manure temperature($^\circ$C)', fontsize='15')
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.legend(handles=[p1, p2, p3, p4, p5], labels=[
    'Actual MT in SCP',
    'Predicted MT in SCP by FEHT',
    'Predicted MT in SCP by DDNN',
    'Predicted MT in SCP by PINN',
    'Predicted MT in SCP by SSNN'],
    loc='best', bbox_to_anchor=(1, 1), frameon=True, fontsize='15')

plt.savefig("CP_DOY_ALL.jpg", bbox_inches='tight', dpi=250)
plt.show()

np.random.seed(111)
tf.random.set_seed(111)

my_array = np.concatenate((DOY_CP, y_CP, y_FEM_CP, y_CP_pred_DDNN,
    y_CP_pred_PINN, y_CP_pred_SSNN), axis=1)

DOY_plot = pd.DataFrame(my_array, columns=['DOY', 'y_test', 'y_FEM', 'y_test_pred_DD',
    'y_test_pred_PI', 'y_test_pred_SS'])

y_test_doy = (DOY_plot.groupby('DOY', as_index=False)['y_test'].mean())
```python
y_test_FEM_doy=(DOY_plot.groupby('DOY', as_index=False)['y_FEM'].mean())
y_test_pred_doy_DD=(DOY_plot.groupby('DOY', as_index=False)['y_test_pred_DD'].mean())
y_test_pred_doy_PI=(DOY_plot.groupby('DOY', as_index=False)['y_test_pred_PI'].mean())
y_test_pred_doy_SS=(DOY_plot.groupby('DOY', as_index=False)['y_test_pred_SS'].mean())
y_test_doy_std=(DOY_plot.groupby('DOY', as_index=False)['y_test'].std())
y_test_FEM_doy_std=(DOY_plot.groupby('DOY', as_index=False)['y_FEM'].std())
y_test_pred_doy_std_DD=(DOY_plot.groupby('DOY', as_index=False)['y_test_pred_DD'].std())
y_test_pred_doy_std_PI=(DOY_plot.groupby('DOY', as_index=False)['y_test_pred_PI'].std())
y_test_pred_doy_std_SS=(DOY_plot.groupby('DOY', as_index=False)['y_test_pred_SS'].std())

plt.figure(figsize=(16,10))
p1=plt.plot(y_test_doy[:,0],y_test_doy[:,1], linestyle='-',c='black', linewidth=3)
p2=plt.plot(y_test_FEM_doy[:,0],y_test_FEM_doy[:,1], linestyle=':',c='black', linewidth=3)
p3=plt.plot(y_test_pred_doy_DD[:,0],y_test_pred_doy_DD[:,1], linestyle='-',c='grey', linewidth=3)
p4=plt.plot(y_test_pred_doy_PI[:,0],y_test_pred_doy_PI[:,1], linestyle='-',c='blue', linewidth=3)
p5=plt.plot(y_test_pred_doy_SS[:,0],y_test_pred_doy_SS[:,1], linestyle='--',c='black', linewidth=3)

plt.xlabel('DOY', fontsize='15')
plt.ylabel('Manure temperature($^\circ$C)', fontsize='15')
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
```

250
plt.legend(handles=[p1[0],p2[0],p3[0],p4[0],p5[0]],labels=['Actual MT in SCP',
    'Predicted MT in SCP by FEHT',
    'Predicted MT in SCP by DDNN',
    'Predicted MT in SCP by PINN',
    'Predicted MT in SCP by SSNN'],
    loc='best',bbox_to_anchor=(1,1),frameon=True, fontsize='15')
plt.savefig("CP_DOY_ALL.jpg",bbox_inches='tight',dpi=250)
plt.show()

# print daily averages
table={'DOY':y_test_doy[:,0],'Actual MT':y_test_doy[:,1],'FEM':y_test_FEM_doy[:,1],
    'DDNN':y_test_pred_doy_DD[:,1],'PINN':y_test_pred_doy_PI[:,1],
    'SSNN':y_test_pred_doy_SS[:,1]}
print(tabulate(table, headers='keys', tablefmt='fancy_grid'))

table1={'DOY':y_test_doy[:,0],
    'Actual MT':(y_test_doy[:,1]/y_test_doy[:,1]),
    'FEM':(y_test_FEM_doy[:,1]/y_test_doy[:,1]),
    'DDNN':(y_test_pred_doy_DD[:,1]/y_test_doy[:,1]),
    'PINN':(y_test_pred_doy_PI[:,1]/y_test_doy[:,1]),
    'SSNN':(y_test_pred_doy_SS[:,1]/y_test_doy[:,1])}
print(tabulate(table1, headers='keys', tablefmt='fancy_grid'))

table2={'DOY':y_test_doy[:,0],
    'Actual MT':1-(y_test_doy[:,1]/y_test_doy[:,1]),
    'FEM':1-(y_test_FEM_doy[:,1]/y_test_doy[:,1]),
'DDNN': 1 - (y_test_pred_doy_DD[:,1]/y_test_doy[:,1]),
'PINN': 1 - (y_test_pred_doy_PI[:,1]/y_test_doy[:,1]),
'SSNN': 1 - (y_test_pred_doy_SS[:,1]/y_test_doy[:,1])

print(tabulate(table2, headers='keys', tablefmt='fancy_grid'))

print(1 - (y_test_pred_doy_PI[:,1]/y_test_pred_doy_DD[:,1]))
my_array = np.concatenate((month_CP, y_CP, y_FEM_CP, y_CP_pred_DD, y_CP_pred_PINN, y_CP_pred_SSNN), axis=1)

DOY_plot = pd.DataFrame(my_array, columns=['month', 'y_test', 'y_FEM', 'y_test_pred_DD', 'y_test_pred_PI', 'y_test_pred_SS'])

y_test_doy = (DOY_plot.groupby('month', as_index=False)['y_test'].mean())
y_test_FEM_doy = (DOY_plot.groupby('month', as_index=False)['y_FEM'].mean())
y_test_pred_doy_DD = (DOY_plot.groupby('month', as_index=False)['y_test_pred_DD'].mean())
y_test_pred_doy_PI = (DOY_plot.groupby('month', as_index=False)['y_test_pred_PI'].mean())
y_test_pred_doy_SS = (DOY_plot.groupby('month', as_index=False)['y_test_pred_SS'].mean())

y_test_doy_std = (DOY_plot.groupby('month', as_index=False)['y_test'].std())
y_test_FEM_doy_std = (DOY_plot.groupby('month', as_index=False)['y_FEM'].std())
y_test_pred_doy_std_DD = (DOY_plot.groupby('month', as_index=False)['y_test_pred_DD'].std())
y_test_pred_doy_std_PI = (DOY_plot.groupby('month', as_index=False)['y_test_pred_I'].std())
y_test_pred_doy_std_SS = (DOY_plot.groupby('month', as_index=False)['y_test_S'].std())

plt.figure(figsize=(12,10))
p1 = plt.errorbar(y_test_doy[:,0], y_test_doy[:,1], xerr=0, markerfacecolor='black',
        yerr=y_test_doy_std[:,1], fmt='.', ecolor='black', capsize=3,
plt.plot(y_test_doy[:,0], y_test_doy[:,1], color='black', linewidth=3)
p2=plt.errorbar(y_test_FEM_doy[:,0], y_test_FEM_doy[:,1], xerr=0, yerr=y_test_FEM_doy_std[:,1], fmt='o', ecolor='grey', capsize=3, markerfacecolor='grey', markeredgecolor='grey')
plt.plot(y_test_FEM_doy[:,0], y_test_FEM_doy[:,1], color='grey', linewidth=3)
p3=plt.errorbar(y_test_pred_doy_DD[:,0], y_test_pred_doy_DD[:,1], xerr=0, yerr=y_test_pred_doy_std_DD[:,1], fmt='x', ecolor='blue', capsize=3, markerfacecolor='blue', markeredgecolor='blue')
plt.plot(y_test_pred_doy_DD[:,0], y_test_pred_doy_DD[:,1], color='blue', linewidth=3)
p4=plt.errorbar(y_test_pred_doy_PI[:,0], y_test_pred_doy_PI[:,1], xerr=0, yerr=y_test_pred_doy_std_PI[:,1], fmt='s', ecolor='green', capsize=3, markerfacecolor='green', markeredgecolor='green')
plt.plot(y_test_pred_doy_PI[:,0], y_test_pred_doy_PI[:,1], color='green', linewidth=3)
p5=plt.errorbar(y_test_pred_doy_SS[:,0], y_test_pred_doy_SS[:,1], xerr=0, yerr=y_test_pred_doy_std_SS[:,1], fmt='.', ecolor='red', capsize=3, markerfacecolor='red', markeredgecolor='red')
plt.plot(y_test_pred_doy_SS[:,0], y_test_pred_doy_SS[:,1], color='red', linewidth=3)

plt.xlabel('Month', fontsize='15')
plt.ylabel('Manure temperature($^\circ$C)', fontsize='15')
plt.xticks([1,2,3,4,5,6,7,8,9,10,11,12], fontsize=15)
plt.yticks(fontsize=15)
plt.legend(handles=[p1, p2, p3, p4, p5], labels=['Actual MT in SCP', 'FEM', 'DD', 'PI', 'SS'])
'Predicted MT in SCP by FEHT',
'Predicted MT in SCP by DDNN',
'Predicted MT in SCP by PINN',
'Predicted MT in SCP by SSNN'],
loc='best',bbox_to_anchor=(1,1),frameon=True, fontsize='15')
plt.savefig("SCP_month_ALL.jpg",bbox_inches='tight',dpi=250)
plt.show()

# print monthly averages

table={
'Month':y_test_doy[:,0],
'Actual MT':y_test_doy[:,1],
'FEM':y_test_FEM_doy[:,1],
'DDNN':y_test_pred_doy_DD[:,1],
'PINN':y_test_pred_doy_PI[:,1],
'SSNN':y_test_pred_doy_SS[:,1]}
print(tabulate(table, headers='keys', tablefmt='fancy_grid'))


table1={
'Month':y_test_doy[:,0],
'Actual MT':(y_test_doy[:,1]/y_test_doy[:,1]),
'FEM':(y_test_FEM_doy[:,1]/y_test_doy[:,1]),
'DDNN':(y_test_pred_doy_DD[:,1]/y_test_doy[:,1]),
'PINN':(y_test_pred_doy_PI[:,1]/y_test_doy[:,1]),
'SSNN':(y_test_pred_doy_SS[:,1]/y_test_doy[:,1])
print(tabulate(table1, headers='keys', tablefmt='fancy_grid'))


table2={
'Month':y_test_doy[:,0],
'Actual MT':1-(y_test_doy[:,1]/y_test_doy[:,1]),
'FEM':1-(y_test_FEM_doy[:,1]/y_test_doy[:,1]),
'DDNN':1-(y_test_pred_doy_DD[:,1]/y_test_doy[:,1]),
'SSNN':1-(y_test_pred_doy_SS[:,1]/y_test_doy[:,1])}
'PINN': 1 - (y_test_pred_doy_PI[:, 1] / y_test_doy[:, 1]),
'SSNN': 1 - (y_test_pred_doy_SS[:, 1] / y_test_doy[:, 1])

print(tabulate(table2, headers='keys', tablefmt='fancy_grid'))

print(1 - (y_test_pred_doy_PI[:, 1] / y_test_pred_doy_DD[:, 1]))
np.random.seed(111)
tf.random.set_seed(111)

my_array = np.concatenate((DOY_CP, y_CP, y_FEM_CP, y_CP_pred_DDNN,
y_CP_pred_PINN, y_CP_pred_SSNN, AAT_CP), axis=1)

DOY_plot = pd.DataFrame(my_array, columns=['DOY', 'y_test', 'y_FEM', 'y_test_pred_DD',
'y_test_pred_PI', 'y_test_pred_SS', 'AAT'])

y_test_doy = (DOY_plot.groupby('DOY', as_index=False)['y_test'].mean())
y_test_FEM_doy = (DOY_plot.groupby('DOY', as_index=False)['y_FEM'].mean())
y_test_pred_doy_DD = (DOY_plot.groupby('DOY', as_index=False)['y_test_pred_DD'].mean())
y_test_pred_doy_PI = (DOY_plot.groupby('DOY', as_index=False)['y_test_pred_PI'].mean())
y_test_pred_doy_SS = (DOY_plot.groupby('DOY', as_index=False)['y_test_pred_SS'].mean())
AAT_doy = (DOY_plot.groupby('DOY', as_index=False)['AAT'].mean())

fig = plt.figure()
fig.set_size_inches(15, 10)
axs = fig.subplots(nrows=2, ncols=2)

axs[0, 0].plot(y_test_doy[:, 0], y_test_doy[:, 1], color='k')
axs[0, 0].plot(y_test_doy[:, 0], y_test_FEM_doy[:, 1], color='b')
axs[0, 0].plot(y_test_doy[:, 0], AAT_doy[:, 1], color='k', linestyle='')

axs[0, 0].set_title('Model: FEHT')
axs[0, 1].plot(y_test_doy[:,0],y_test_doy[:,1], color='k')
axs[0, 1].plot(y_test_doy[:,0],y_test_pred_doy_DD[:,1], color='b')
axs[0, 1].plot(y_test_doy[:,0],AAT_doy[:,1], color='k', linestyle=':')
axs[0, 1].set_title('Model: DDNN')

axs[1, 0].plot(y_test_doy[:,0],y_test_doy[:,1], color='k')
axs[1, 0].plot(y_test_doy[:,0],y_test_pred_doy_PI[:,1], color='b')
axs[1, 0].plot(y_test_doy[:,0],AAT_doy[:,1], color='k', linestyle=':')
axs[1, 0].set_title('Model: PINN')

axs[1, 1].plot(y_test_doy[:,0],y_test_doy[:,1], color='k',
               label='Actual manure temperature in SCP')
axs[1, 1].plot(y_test_doy[:,0],y_test_pred_doy_SS[:,1], color='b',
               label='predicted manure temperature in SCP')
axs[1, 1].plot(y_test_doy[:,0],AAT_doy[:,1], color='k', linestyle=':',
               label='Ambient air temperature in SCP')
axs[1, 1].set_title('Model: SSNN')

lines = []
labels = []
for ax in fig.axes:
    axLine, axLabel = ax.get_legend_handles_labels()
    lines.extend(axLine)
    labels.extend(axLabel)
fig.legend(lines, labels,loc = 'lower center',fontsize=16)
for ax in fig.axes:
    ax.set(xlabel='DOY', ylabel='Manure temperature($\circ$C)')
for item in ([ax.title, ax.xaxis.label, ax.yaxis.label] +
    ax.get_xticklabels() + ax.get_yticklabels()):
    item.set_fontsize(14)
plt.subplots_adjust(left=0.1,
    bottom=0.18,
    right=0.9,
    top=0.9,
    wspace=0.4,
    hspace=0.4)
plt.savefig("CP_DOY_each.jpg",bbox_inches='tight',dpi=250)
plt.show()

#box_plots
np.random.seed(111)
tf.random.set_seed(111)
my_array = np.concatenate((DOY_CP,y_CP,y_FEM_CP,y_CP_pred_DDNN
y_CP_pred_PINN,y_CP_pred_SSNN,AAT_CP),axis=1)
DOY_plot= pd.DataFrame(my_array, columns = ['DOY','y_test','y_FEM','y_test_pred_DD',
'y_test_pred_PI','y_test_pred_SS','AAT'])
y_test_doy=(DOY_plot.groupby('DOY', as_index=False)['y_test'].mean())
y_test_FEM_doy=(DOY_plot.groupby('DOY', as_index=False)['y_FEM'].mean())
y_test_pred_doy_DD=(DOY_plot.groupby('DOY', as_index=False)['y_test_pred_DD'].mean())
y_test_pred_doy_PI=(DOY_plot.groupby('DOY', as_index=False)['y_test_pred_PI'].mean())
y_test_pred_doy_SS=(DOY_plot.groupby('DOY', as_index=False)['y_test_pred_SS'].mean())

257
AAT_doy=(DOY_plot.groupby('DOY', as_index=False)['AAT'].mean())
x=DOY_plot['DOY']
fig,ax = plt.subplots(figsize=(9,7))
medianprops = {'color': 'black', 'linewidth': 2}
boxprops = {'color': 'black', 'linestyle': '-'}
whiskerprops = {'color': 'black', 'linestyle': '-'}
capprops = {'color': 'black', 'linestyle': '-'}
flierprops = {'color': 'black', 'marker': 'x'}
bp = ax.boxplot(x=[y_test_doy[:,1],y_test_FEM_doy[:,1],y_test_pred_doy_DD[:,1],y_test_pred_doy_PI[:,1],y_test_pred_doy_SS[:,1]],
positions=[1,2,3,4,5])
ax.set(xlabel='Actual and predicted manure temperature in SCP', ylabel='Manure temperature($^\circ$C)')
plt.savefig("CP_DOY_bp_each.jpg",bbox_inches='tight',dpi=250)
plt.show()
np.random.seed(111)
tf.random.set_seed(111)
my_array = np.concatenate((DOY_CP,y_CP,y_FEM_CP,y_CP_pred_DDNN,
y_CP_pred_PINN,y_CP_pred_SSNN,AAT_CP),axis=1)
DOY_plot= pd.DataFrame(my_array, columns = ['DOY','y_test','y_FEM','y_test_pred_DD','y_test_pred_PI','y_test_pred_SS','AAT'])
y_test_doy=(DOY_plot.groupby('DOY', as_index=False)['y_test'].mean())
y_test_FEM_doy=(DOY_plot.groupby('DOY', as_index=False)['y_FEM'].mean())
y_test_pred_doy_DD=(DOY_plot.groupby('DOY', as_index=False)['y_test_pred_DD'].mean())
y_test_pred_doy_PI=(DOY_plot.groupby('DOY', as_index=False)['y_test_pred_PI'].mean())
y_test_pred_doy_SS=(DOY_plot.groupby('DOY', as_index=False)['y_test_pred_SS'].mean())
AAT_doy=(DOY_plot.groupby('DOY', as_index=False)['AAT'].mean())
y_test_diff = y_test_doy - y_test_doy
y_test_FEM_doy_diff = y_test_doy - y_test_FEM_doy
y_test_pred_doy_DD_diff = y_test_doy - y_test_pred_doy_DD
y_test_pred_doy_PI_diff = y_test_doy - y_test_pred_doy_PI
y_test_pred_doy_SS_diff = y_test_doy - y_test_pred_doy_SS

fig = plt.figure()
fig.set_size_inches(12, 6)
axs = fig.subplots(nrows=2, ncols=2)
axs[0, 0].plot(y_test_doy[:, 0], y_test_FEM_doy_diff[:, 1], color='b')
axs[0, 0].set_title('Model: FEHT')

axs[0, 1].plot(y_test_doy[:, 0], y_test_pred_doy_DD_diff[:, 1], color='b')
axs[0, 1].set_title('Model: DDNN')

axs[1, 0].plot(y_test_doy[:, 0], y_test_pred_doy_PI_diff[:, 1], color='b')
axs[1, 0].set_title('Model: PINN')

axs[1, 1].plot(y_test_doy[:, 0], y_test_pred_doy_SS_diff[:, 1], color='b', label='difference between predicted and actual manure temperature in SCP')
axs[1, 1].set_title('Model: SSNN')

lines = []
labels = []
for ax in fig.axes:
    ax.set(xlabel='DOY', ylabel='Manure temperature($^\circ$C)')
axLine, axLabel = ax.get_legend_handles_labels()
lines.extend(axLine)
labels. Extend(axLabel)
for item in ([ax.title, ax.xaxis.label, ax.yaxis.label] +
    ax.get_xticklabels() + ax.get_yticklabels()):
    item.set_fontsize(14)

fig.legend(lines, labels,
    loc = 'upper right')
plt.subplots_adjust(left=0.1,
    bottom=0.1,
    right=0.73,
    top=0.9,
    wspace=0.5,
    hspace=0.5)
plt.savefig("diff_DOY_each_SCP.jpg",bbox_inches='tight',dpi=250)
plt.show()

plt.figure(figsize=(12,10))
plt.plot(y_test_doy[:,0],y_test_FEM_doy_diff[:,1], color='k',label='FEHT')
plt.plot(y_test_doy[:,0],y_test_pred_doy_DD_diff[:,1], color='k',linestyle='--', label='DDNN')
plt.plot(y_test_doy[:,0],y_test_pred_doy_PI_diff[:,1], color='b',label='PINN')
plt.plot(y_test_doy[:,0],y_test_pred_doy_SS_diff[:,1], color='k',linestyle=':', label='SSNN')
plt.plot(y_test_doy[:,0],y_test_diff[:,1], color='r')
plt.xlabel('DOY', fontsize=25)
plt.ylabel('Residuals in SCP (actual MT - predicted MT)', fontsize=25)
plt.legend(fontsize=25)
plt.xticks(fontsize=25)
plt.yticks(np.arange(-20, 15, 5), fontsize=25)
plt.savefig("diff_DOY_SCP.jpg", bbox_inches='tight', dpi=250)
plt.show()

# stats difference
print('Mean difference between actual MT and FEM predictions in SCP: ',
     np.average(y_test_FEM_doy_diff[:,1]))
print('minimum difference between actual MT and FEM predictions in SCP: ',
     np.min(y_test_FEM_doy_diff[:,1]))
print('maximum difference between actual MT and FEM predictions in SCP: ',
     np.max(y_test_FEM_doy_diff[:,1]))
print('Mean difference between actual MT and DDNN predictions in SCP: ',
     np.average(y_test_pred_doy_DD_diff[:,1]))
print('minimum difference between actual MT and DDNN predictions in SCP: ',
     np.min(y_test_pred_doy_DD_diff[:,1]))
print('maximum difference between actual MT and DDNN predictions in SCP: ',
     np.max(y_test_pred_doy_DD_diff[:,1]))
print('Mean difference between actual MT and PINN predictions in SCP: ',
     np.average(y_test_pred_doy_PI_diff[:,1]))
print('minimum difference between actual MT and PINN predictions in SCP: ',
     np.min(y_test_pred_doy_PI_diff[:,1]))
print('maximum difference between actual MT and PINN predictions in SCP: ',
     np.max(y_test_pred_doy_PI_diff[:,1]))
np.max(y_test_pred_doy_PI_diff[:,1])
print('Mean difference between actual MT and SSNN predictions in SCP:',
np.average(y_test_pred_doy_SS_diff[:,1]))
print('minimum difference between actual MT and SSNN predictions in SCP:',
np.min(y_test_pred_doy_SS_diff[:,1]))
print('maximum difference between actual MT and SSNN predictions in SCP:',
np.max(y_test_pred_doy_SS_diff[:,1]))

my_array = np.concatenate((month_CP,y_CP,y_FEM_CP,y_CP_pred_DDNN,
y_CP_pred_PINN,y_CP_pred_SSNN),axis=1)
DOY_plot= pd.DataFrame(my_array, columns = ['month','y_test','y_FEM','y_test_pred_DD',
'y_test_pred_PI','y_test_pred_SS'])

y_test_doy=(DOY_plot.groupby('month', as_index=False)['y_test'].mean())
y_test_FEM_doy=(DOY_plot.groupby('month', as_index=False)['y_FEM'].mean())
y_test_pred_doy_DD=(DOY_plot.groupby('month', as_index=False)['y_test_pred_DD'].mean())
y_test_pred_doy_PI=(DOY_plot.groupby('month', as_index=False)['y_test_pred_PI'].mean())
y_test_pred_doy_SS=(DOY_plot.groupby('month', as_index=False)['y_test_pred_SS'].mean())

y_test_diff=y_test_doy-y_test_doy
y_test_FEM_doy_diff= y_test_doy-y_test_FEM_doy
y_test_pred_doy_DD_diff=y_test_doy-y_test_pred_doy_DD
y_test_pred_doy_PI_diff=y_test_doy-y_test_pred_doy_PI
y_test_pred_doy_SS_diff=y_test_doy-y_test_pred_doy_SS

plt.figure(figsize=(12,6))
```python
plt.plot(y_test_doy[:,0], y_test_FEM_doy_diff[:,1], color='k', label='FEHT')
plt.plot(y_test_doy[:,0], y_test_pred_doy_DD_diff[:,1], color='k', linestyle='--', label='DDNN')
plt.plot(y_test_doy[:,0], y_test_pred_doy_PI_diff[:,1], color='b', label='PINN')
plt.plot(y_test_doy[:,0], y_test_pred_doy_SS_diff[:,1], color='k', linestyle=':', label='SSNN')
plt.plot(y_test_doy[:,0], y_test_diff[:,1], color='r')
plt.xlabel('Month')
plt.ylabel('Actual-predicted MT in SCP')
plt.yticks([-10,-5,0,5,10])
plt.legend()
plt.savefig("diff_Month_SCP.jpg", bbox_inches='tight', dpi=250)
plt.show()

# stats difference
print('Mean difference between actual MT and FEM predictions in SCP: ',
      np.average(y_test_FEM_doy_diff[:,1])))
print('minimum difference between actual MT and FEM predictions in SCP: ',
      np.min(y_test_FEM_doy_diff[:,1])))
print('maximum difference between actual MT and FEM predictions in SCP: ',
      np.max(y_test_FEM_doy_diff[:,1])))
print('Mean difference between actual MT and DDNN predictions in SCP: ',
      np.average(y_test_pred_doy_DD_diff[:,1])))
print('minimum difference between actual MT and DDNN predictions in SCP: ',
      np.min(y_test_pred_doy_DD_diff[:,1])))
print('maximum difference between actual MT and DDNN predictions in SCP: ',
      np.max(y_test_pred_doy_DD_diff[:,1])))
```

263
np.max(y_test_pred_doy_DD_diff[:,1]))
print('Mean difference between actual MT and PINN predictions in SCP:',
np.average(y_test_pred_doy_PI_diff[:,1]))
print('minimum difference between actual MT and PINN predictions in SCP:',
np.min(y_test_pred_doy_PI_diff[:,1]))
print('maximum difference between actual MT and PINN predictions in SCP:',
np.max(y_test_pred_doy_PI_diff[:,1]))
print('Mean difference between actual MT and SSNN predictions in SCP:',
np.average(y_test_pred_doy_SS_diff[:,1]))
print('minimum difference between actual MT and SSNN predictions in SCP:',
np.min(y_test_pred_doy_SS_diff[:,1]))
print('maximum difference between actual MT and SSNN predictions in SCP:',
np.max(y_test_pred_doy_SS_diff[:,1]))

Testing on FCT dataset

np.random.seed(111)
tf.random.set_seed(111)

#Predicted MT in the FCT by the models
y_VTA_pred_DDNN=DDNN.evaluate(X_VTA)
y_VTA_pred_PINN=PINN.evaluate(X_VTA)
y_VTA_pred_SSNN=SSNN.evaluate(X_VTA)

#evaluation parameters from the FEM when tested on the data of the FCT
print('Mean absolute error from testing the FEM on FCT:',
metrics.mean_absolute_error(y_VTA,y_FEM_VTA))
print('Root mean square error from testing the FEM on FCT:',
metrics.mean_squared_error(y_VTA,y_FEM_VTA))
np.sqrt(metrics.mean_squared_error(y_VTA,y_FEM_VTA)))
print('R2 from testing the FEM on FCT:', metrics.r2_score(y_VTA,y_FEM_VTA))

#evaluation parameters from the DDNN when tested on the data of the FCT
print('Mean absolute error from testing the DDNN on FCT:',
      metrics.mean_absolute_error(y_VTA,y_VTA_pred_DDNN))
print('Root mean square error from testing the DDNN on FCT:',
      np.sqrt(metrics.mean_squared_error(y_VTA,y_VTA_pred_DDNN)))
print('R2 from testing the DDNN on FCT:', metrics.r2_score(y_VTA,y_VTA_pred_DDNN))

#evaluation parameters from the PINN when tested on the data of the FCT
print('Mean absolute error from testing the PINN on FCT:',
      metrics.mean_absolute_error(y_VTA,y_VTA_pred_PINN))
print('Root mean square error from testing the PINN on SCP:',
      np.sqrt(metrics.mean_squared_error(y_VTA,y_VTA_pred_PINN)))
print('R2 from testing the PINN on SCP:', metrics.r2_score(y_VTA,y_VTA_pred_PINN))

#evaluation parameters from the SSNN when tested on the data of the FCT
print('Mean absolute error from testing the SSNN on FCT:',
      metrics.mean_absolute_error(y_VTA,y_VTA_pred_SSNN))
print('Root mean square error from testing the SSNN on SCP:',
      np.sqrt(metrics.mean_squared_error(y_VTA,y_VTA_pred_SSNN)))
print('R2 from testing the SSNN on SCP:', metrics.r2_score(y_VTA,y_VTA_pred_SSNN))
np.random.seed(111)
tf.random.set_seed(111)
fig = plt.figure()
```python
fig.set_size_inches(12, 8)
axes = fig.subplots(nrows=2, ncols=2)

axes[0, 0].scatter(y_VTA, y_FEM_VTA, marker='o', c='k')
axes[0, 0].plot([0, 30], [0, 30], linestyle='--', c='grey')
axes[0, 1].scatter(y_VTA, y_VTA_pred_DDNN, marker='o', c='k')
axes[0, 1].plot([0, 30], [0, 30], linestyle='--', c='grey')
axes[1, 0].scatter(y_VTA, y_VTA_pred_PINN, marker='o', c='k')
axes[1, 0].plot([0, 30], [0, 30], linestyle='--', c='grey')
axes[1, 1].scatter(y_VTA, y_VTA_pred_SSNN, marker='o', c='k')
axes[1, 1].plot([0, 30], [0, 30], linestyle='--', c='grey')

axes[0, 0].set_title('Model: FEHT')
axes[0, 1].set_title('Model: DDNN')
axes[1, 0].set_title('Model: PINN')
axes[1, 1].set_title('Model: SSNN')

for ax in fig.axes:
    ax.set(xlabel='Actual Manure temperature in FCT ($^\circ$C)',
            ylabel='Predicted manure temperature in FCT ($^\circ$C)')

plt.subplots_adjust(left=0.1,
                     bottom=0.1,
                     right=0.9,
                     top=0.9,
                     wspace=0.5,
                     hspace=0.5)

plt.savefig("AcvPr_VTA_each.jpg", bbox_inches='tight', dpi=250)
plt.show()

np.random.seed(111)
tf.random.set_seed(111)
```
my_array = np.concatenate((DOY_VTA,y_VTA,y_FEM_VTA,y_VTA_pred_DDNN, 
y_VTA_pred_PINN,y_VTA_pred_SSNN),axis=1)
DOY_plot= pd.DataFrame(my_array, columns = ['DOY','y_test','y_FEM', 
'y_test_pred_DD','y_test_pred_PI','y_test_pred_SS'])

doy=(DOY_plot['DOY'])
y_test_doy=(DOY_plot['y_test'])
y_test_FEM_doy=(DOY_plot['y_FEM'])
y_test_pred_doy_DD=(DOY_plot['y_test_pred_DD'])
y_test_pred_doy_PI=(DOY_plot['y_test_pred_PI'])
y_test_pred_doy_SS=(DOY_plot['y_test_pred_SS'])

plt.figure(figsize=(18,10))
p1=plt.plot(y_test_doy, linestyle='-',c='black', linewidth=4)
p2=plt.plot(y_test_FEM_doy, linestyle=':',c='black', linewidth=3)
p3=plt.plot(y_test_pred_doy_DD, linestyle='-',c='grey', linewidth=3)
p4=plt.plot(y_test_pred_doy_PI, linestyle='-',c='blue', linewidth=3)
p5=plt.plot(y_test_pred_doy_SS, linestyle='--',c='black', linewidth=3)

plt.xlabel('Day', fontsize='15')
plt.ylabel('Manure temperature($\circ$C)', fontsize='15')
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.legend(handles=
[p1[0],p2[0],p3[0],p4[0],p5[0]],labels=['Actual MT in FCT',
'Predicted MT in FCT by FEHT',
'Predicted MT in FCT by DDNN'],
plt.savefig("VTA_DOY_ALL.jpg",bbox_inches='tight',dpi=250)
plt.show()
np.random.seed(111)
tf.random.set_seed(111)
my_array = np.concatenate((DOY_VTA,y_VTA,y_FEM_VTA,y_VTA_pred_DDNN,y_VTA_pred_PINN,y_VTA_pred_SSNN),axis=1)
DOY_plot= pd.DataFrame(my_array, columns = ['DOY','y_test','y_FEM','y_test_pred_DD','y_test_pred_PI','y_test_pred_SS'])
y_test_doy=(DOY_plot.groupby('DOY', as_index=False)['y_test'].mean())
y_test_FEM_doy=(DOY_plot.groupby('DOY', as_index=False)['y_FEM'].mean())
y_test_pred_doy_DD=(DOY_plot.groupby('DOY', as_index=False)['y_test_pred_DD'].mean())
y_test_pred_doy_PI=(DOY_plot.groupby('DOY', as_index=False)['y_test_pred_PI'].mean())
y_test_pred_doy_SS=(DOY_plot.groupby('DOY', as_index=False)['y_test_pred_SS'].mean())
y_test_doy_std=(DOY_plot.groupby('DOY', as_index=False)['y_test'].std())
y_test_FEM_doy_std=(DOY_plot.groupby('DOY', as_index=False)['y_FEM'].std())
y_test_pred_doy_std_DD=(DOY_plot.groupby('DOY', as_index=False)['y_test_pred_DD'].std())
y_test_pred_doy_std_PI=(DOY_plot.groupby('DOY', as_index=False)['y_test_pred_PI'].std())
y_test_pred_doy_std_SS=(DOY_plot.groupby('DOY', as_index=False)['y_test_pred_SS'].std())

my_array = np.concatenate((month_VTA,y_VTA,y_FEM_VTA, y_VTA_pred_DDNN,y_VTA_pred_PINN,y_VTA_pred_SSNN),axis=1)
DOY_plot= pd.DataFrame(my_array, columns = ['month','y_test','y_FEM','y_test_pred_DD','y_test_pred_PI','y_test_pred_SS'])
y_test_doy = (DOY_plot.groupby('month', as_index=False)['y_test'].mean())
y_test_FEM_doy = (DOY_plot.groupby('month', as_index=False)['y_FEM'].mean())
y_test_pred_doy_DD = (DOY_plot.groupby('month', as_index=False)['y_test_pred_DD'].mean())
y_test_pred_doy_PI = (DOY_plot.groupby('month', as_index=False)['y_test_pred_PI'].mean())
y_test_pred_doy_SS = (DOY_plot.groupby('month', as_index=False)['y_test_pred_SS'].mean())

y_test_doy_std = (DOY_plot.groupby('month', as_index=False)['y_test'].std())
y_test_FEM_doy_std = (DOY_plot.groupby('month', as_index=False)['y_FEM'].std())
y_test_pred_doy_std_DD = (DOY_plot.groupby('month', as_index=False)['y_test_pred_DD'].std())
y_test_pred_doy_std_PI = (DOY_plot.groupby('month', as_index=False)['y_test_pred_PI'].std())
y_test_pred_doy_std_SS = (DOY_plot.groupby('month', as_index=False)['y_test_pred_SS'].std())

plt.figure(figsize=(12,10))
p1=plt.errorbar(y_test_doy[:,0], y_test_doy[:,1], xerr=0, markerfacecolor='black',
                yerr=y_test_doy_std[:,1], fmt='.', ecolor='black', capsize=3,
                markeredgecolor='black')
plt.plot(y_test_doy[:,0], y_test_doy[:,1], color='black', linewidth=3)
p2=plt.errorbar(y_test_FEM_doy[:,0], y_test_FEM_doy[:,1], xerr=0,
                yerr=y_test_FEM_doy_std[:,1], fmt='o', ecolor='grey',
                capsize=3, markerfacecolor='grey', markeredgecolor='grey')
plt.plot(y_test_FEM_doy[:,0], y_test_FEM_doy[:,1], color='grey', linewidth=3)
p3=plt.errorbar(y_test_pred_doy_DD[:,0], y_test_pred_doy_DD[:,1], xerr=0,
                yerr=y_test_pred_doy_std_DD[:,1], fmt='x', ecolor='blue',
                capsize=3, markerfacecolor='blue', markeredgecolor='blue')
plt.plot(y_test_pred_doy_DD[:,0], y_test_pred_doy_DD[:,1], color='blue', linewidth=3)

269
p4=plt.errorbar(y_test_pred_doy_PI[:,0],y_test_pred_doy_PI[:,1],xerr=0,
            yerr=y_test_pred_doy_std_PI[:,1],fmt='s', ecolor='green',
            capsize=3, markerfacecolor='green', markeredgecolor='green')
plt.plot(y_test_pred_doy_PI[:,0],y_test_pred_doy_PI[:,1],color='green',linewidth=3)
p5=plt.errorbar(y_test_pred_doy_SS[:,0],y_test_pred_doy_SS[:,1],xerr=0,
            yerr=y_test_pred_doy_std_SS[:,1],fmt='.', ecolor='red',
            capsize=3, markerfacecolor='red', markeredgecolor='red')
plt.plot(y_test_pred_doy_SS[:,0],y_test_pred_doy_SS[:,1],color='red',linewidth=3)

plt.xlabel('Month', fontsize='15')
plt.ylabel('Manure temperature($^\circ$C)', fontsize='15')
plt.xticks([1,2,3,4,5,6,7,8,9,10,11,12],fontsize=15)
plt.yticks(fontsize=15)
plt.legend(handles=[p1,p2,p3,p4,p5],labels=['Actual MT in SCT',
                'Predicted MT in SCT by FEHT',
                'Predicted MT in SCT by DDNN',
                'Predicted MT in SCT by PINN',
                'Predicted MT in SCT by SSNN'],
            loc='best',bbox_to_anchor=(1,1),frameon=True, fontsize='15')
plt.savefig("CT_month_ALL.jpg",bbox_inches='tight',dpi=250)
plt.show()

# print monthly averages

# print monthly averages

table={'Month':y_test_doy[:,0],'Actual MT':y_test_doy[:,1],'FEHT':y_test_FEM_doy[:,1],
...
print(tabulate(table, headers='keys', tablefmt='fancy_grid'))

table1={'Month':y_test_doy[:,0],
       'Actual MT':(y_test_doy[:,1]/y_test_doy[:,1]),
       'FEM':(y_test_FEM_doy[:,1]/y_test_doy[:,1]),
       'DDNN':(y_test_pred_doy_DD[:,1]/y_test_doy[:,1]),
       'PINN':(y_test_pred_doy_PI[:,1]/y_test_doy[:,1]),
       'SSNN':(y_test_pred_doy_SS[:,1]/y_test_doy[:,1])}
print(tabulate(table1, headers='keys', tablefmt='fancy_grid'))

table2={'Month':y_test_doy[:,0],
       'Actual MT':1-(y_test_doy[:,1]/y_test_doy[:,1]),
       'FEM':1-(y_test_FEM_doy[:,1]/y_test_doy[:,1]),
       'DDNN':1-(y_test_pred_doy_DD[:,1]/y_test_doy[:,1]),
       'PINN':1-(y_test_pred_doy_PI[:,1]/y_test_doy[:,1]),
       'SSNN':1-(y_test_pred_doy_SS[:,1]/y_test_doy[:,1])}
print(tabulate(table2, headers='keys', tablefmt='fancy_grid'))

print(1-(y_test_pred_doy_PI[:,1]/y_test_pred_doy_DD[:,1]))

np.random.seed(111)
tf.random.set_seed(111)

my_array = np.concatenate((DOY_VTA,y_VTA,y_FEM_VTA,y_VTA_pred_DDNN,y_VTA_pred_PINN,y_VTA_pred_SSNN,AAT_VTA),axis=1)

DOY_plot= pd.DataFrame(my_array, columns = ['DOY','y_test','y_FEM','y_test_pred_DD','y_test_pred_PI','y_test_pred_SS','AAT'])
y_test_doy=(DOY_plot.groupby('DOY', as_index=False)['y_test'].mean())
['y_test'].mean()
y_test_FEM_doy=(DOY_plot.groupby('DOY', as_index=False)['y_FEM'].mean())
['y_FEM'].mean()
y_test_pred_doy_DD=(DOY_plot.groupby('DOY', as_index=False)['y_test_pred_DD'].mean())
['y_test_pred_DD'].mean()
y_test_pred_doy_PI=(DOY_plot.groupby('DOY', as_index=False)['y_test_pred_PI'].mean())
['y_test_pred_PI'].mean()
y_test_pred_doy_SS=(DOY_plot.groupby('DOY', as_index=False)['y_test_pred_SS'].mean())
['y_test_pred_SS'].mean()
AAT_doy=(DOY_plot.groupby('DOY', as_index=False)['AAT'].mean())
['AAT'].mean()

fig = plt.figure()
fig.set_size_inches(15, 10)
axs = fig.subplots(nrows=2, ncols=2)
axs[0, 0].plot(y_test_doy[:,0],y_test_doy[:,1], color='k')
axs[0, 0].plot(y_test_doy[:,0],y_test_FEM_doy[:,1], color='b')
axs[0, 0].plot(y_test_doy[:,0],AAT_doy[:,1], color='k', linestyle=':')
axs[0, 0].set_title('Model: FEHT')

axs[0, 1].plot(y_test_doy[:,0],y_test_doy[:,1], color='k')
axs[0, 1].plot(y_test_doy[:,0],y_test_pred_doy_DD[:,1], color='b')
axs[0, 1].plot(y_test_doy[:,0],AAT_doy[:,1], color='k', linestyle=':')
axs[0, 1].set_title('Model: DDNN')
axs[1, 0].plot(y_test_doy[:,0],y_test_doy[:,1], color='k')
axs[1, 0].plot(y_test_doy[:,0],y_test_pred_doy_PI[:,1], color='b')
axs[1, 0].plot(y_test_doy[:,0],AAT_doy[:,1], color='k', linestyle=':')
axs[1, 0].set_title('Model: PINN')

axs[1, 1].plot(y_test_doy[:,0],y_test_doy[:,1], color='k', label='Actual manure temperature in FCT')
axs[1, 1].plot(y_test_doy[:,0],y_test_pred_doy_SS[:,1], color='b', label='predicted manure temperature in FCT')
axs[1, 1].plot(y_test_doy[:,0],AAT_doy[:,1], color='k', linestyle=':', label='Ambient air temperature in FCT')
axs[1, 1].set_title('Model: SSNN')

lines = []
labels = []
for ax in fig.axes:
    axLine, axLabel = ax.get_legend_handles_labels()
    lines.extend(axLine)
    labels.extend(axLabel)
    ax.set(xlabel='DOY', ylabel='Manure temperature($^\circ$C)')
    for item in ([ax.title, ax.xaxis.label, ax.yaxis.label] +
                 ax.get_xticklabels() + ax.get_yticklabels()):
        item.set_fontsize(14)
fig.legend(lines, labels, loc = 'lower center',fontsize=16)
plt.subplots_adjust(left=0.1, bottom=0.18,
plt.savefig("VTA_DOY_each.jpg",bbox_inches='tight',dpi=250)
plt.show()

# box plots
np.random.seed(111)
tf.random.set_seed(111)
my_array = np.concatenate((DOY_VTA,y_VTA,y_FEM_VTA,y_VTA_pred_DDNN,
y_VTA_pred_PINN,y_VTA_pred_SSNN,AAT_VTA),axis=1)
DOY_plot= pd.DataFrame(my_array, columns = ['DOY','y_test','y_FEM','y_test_pred_DD','y_test_pred_PI','y_test_pred_SS','AAT'])

y_test_doy=(DOY_plot.groupby('DOY', as_index=False),['y_test'].mean())
y_test_FEM_doy=(DOY_plot.groupby('DOY', as_index=False),['y_FEM'].mean())
y_test_pred_doy_DD=(DOY_plot.groupby('DOY', as_index=False),['y_test_pred_DD'].mean())
y_test_pred_doy_PI=(DOY_plot.groupby('DOY', as_index=False),['y_test_pred_PI'].mean())
y_test_pred_doy_SS=(DOY_plot.groupby('DOY', as_index=False),['y_test_pred_SS'].mean())
AAT_doy=(DOY_plot.groupby('DOY', as_index=False),['AAT'].mean())

274
x=DOY_plot['DOY']
fig, ax = plt.subplots(figsize=(9,7))
medianprops = {'color': 'black', 'linewidth': 2}
boxprops = {'color': 'black', 'linestyle': '-'}
whiskerprops = {'color': 'black', 'linestyle': '-'}
capprops = {'color': 'black', 'linestyle': '-'}
flierprops = {'color': 'black', 'marker': 'x'}
bp = ax.boxplot(x=[y_test_doy[:,1], y_test_FEM_doy[:,1],
y_test_pred_doy_DD[:,1], y_test_pred_doy_PI[:,1], y_test_pred_doy_SS[:,1]],
positions=[1, 2, 3, 4, 5])
plt.xticks([1, 2, 3, 4, 5], ['Actual MT', 'Model: FEHT', 'Model: DDNN',
'Model: PINN', 'Model: SSNN'])
ax.set(xlabel='Actual and predicted manure temperature in FCT',
ylabel='Manure temperature($^\circ$C)')
plt.savefig("VTA_DOY_bp_each.jpg", bbox_inches='tight', dpi=250)
plt.show()
np.random.seed(111)
tf.random.set_seed(111)
my_array = np.concatenate((DOY_VTA, y_VTA, y_FEM_VTA,
y_VTA_pred_DDNN, y_VTA_pred_PINN, y_VTA_pred_SSNN, AAT_VTA), axis=1)
DOY_plot = pd.DataFrame(my_array, columns = ['DOY', 'y_test', 'y_FEM',
'y_test_pred_DD', 'y_test_pred_PI', 'y_test_pred_SS', 'AAT'])

y_test_doy = (DOY_plot.groupby('DOY', as_index=False)['y_test'].mean())
y_test_FEM_doy = (DOY_plot.groupby('DOY', as_index=False)['y_FEM'].mean())
y_test_pred_doy_DD = (DOY_plot.groupby('DOY', as_index=False)['y_test_pred_DD'].mean())
y_test_pred_doy_PI = (DOY_plot.groupby('DOY', as_index=False)['y_test_pred_PI'].mean())
y_test_pred_doy_SS = (DOY_plot.groupby('DOY', as_index=False)['y_test_pred_SS'].mean())
y_test_pred_doy_DD=DOY_plot.groupby('DOY', as_index=False)['y_test_pred_DD'].mean()

y_test_pred_doy_PI=DOY_plot.groupby('DOY', as_index=False)['y_test_pred_PI'].mean()

y_test_pred_doy_SS=DOY_plot.groupby('DOY', as_index=False)['y_test_pred_SS'].mean()

AAT_doy=DOY_plot.groupby('DOY', as_index=False)['AAT'].mean()

y_test_diff=y_test_doy-y_test_doy

y_test_FEM_doy_diff= y_test_doy-y_test_FEM_doy

y_test_pred_doy_DD_diff=y_test_doy-y_test_pred_doy_DD

y_test_pred_doy_PI_diff=y_test_doy-y_test_pred_doy_PI

y_test_pred_doy_SS_diff=y_test_doy-y_test_pred_doy_SS

fig = plt.figure()
fig.set_size_inches(12, 6)

axs = fig.subplots(nrows=2, ncols=2)

axs[0, 0].plot(y_test_doy[:,0],y_test_FEM_doy_diff[:,1], color='b')
axs[0, 0].set_title('Model: FEHT')

axs[0, 1].plot(y_test_doy[:,0],y_test_pred_doy_DD_diff[:,1], color='b')
axs[0, 1].set_title('Model: DDNN')

axs[1, 0].plot(y_test_doy[:,0],y_test_pred_doy_PI_diff[:,1], color='b')
axs[1, 0].set_title('Model: PINN')

axs[1, 1].plot(y_test_doy[:,0],y_test_pred_doy_SS_diff[:,1], color='b',

276
label='difference between predicted and actual manure temperature in FCT')
axs[1, 1].set_title('Model: SSNN')

lines = []
labels = []
for ax in fig.axes:
    axLine, axLabel = ax.get_legend_handles_labels()
    lines.extend(axLine)
    labels.extend(axLabel)
fig.legend(lines, labels,
          loc = 'upper right')
for ax in fig.axes:
    ax.set(xlabel='DOY', ylabel='Manure temperature($^\circ$C)')
plt.subplots_adjust(left=0.1,
          bottom=0.1,
          right=0.73,
          top=0.9,
          wspace=0.5,
          hspace=0.5)
plt.savefig("diff_DOY_each_FCT.jpg",bbox_inches='tight',dpi=250)
plt.show()
plt.plot(y_test_doy[:,0], y_test_pred_doy_PI_diff[:,1], color='b', label='PINN')
plt.plot(y_test_doy[:,0], y_test_pred_doy_SS_diff[:,1], color='k', linestyle=':', label='SSNN')
plt.plot(y_test_doy[:,0], y_test_diff[:,1], color='r')
plt.xlabel('DOY', fontsize=25)
plt.ylabel('Residuals in FCT (actual MT - predicted MT)', fontsize=25)
plt.legend(fontsize=25)
plt.xticks(fontsize=25)
plt.yticks(np.arange(-10, 20, 5), fontsize=25)
plt.savefig("diff_DOY_FCT.jpg",bbox_inches='tight',dpi=250)
plt.show()

# stats difference
print('Mean difference between actual MT and FEM predictions in FCT:',
    np.average(y_test_FEM_doy_diff[:,1]))
print('minimum difference between actual MT and FEM predictions in FCT:',
    np.min(y_test_FEM_doy_diff[:,1]))
print('maximum difference between actual MT and FEM predictions in FCT:',
    np.max(y_test_FEM_doy_diff[:,1]))
print('Mean difference between actual MT and DDNN predictions in FCT:',
    np.average(y_test_pred_doy_DD_diff[:,1]))
print('minimum difference between actual MT and DDNN predictions in FCT:',
    np.min(y_test_pred_doy_DD_diff[:,1]))
print('maximum difference between actual MT and DDNN predictions in FCT:',
    np.max(y_test_pred_doy_DD_diff[:,1])
np.max(y_test_pred_doy_DD_diff[:,1]))
print('Mean difference between actual MT and PINN predictions in FCT:',
    np.average(y_test_pred_doy_PI_diff[:,1]))
print('minimum difference between actual MT and PINN predictions in FCT:',
    np.min(y_test_pred_doy_PI_diff[:,1]))
print('maximum difference between actual MT and PINN predictions in FCT:',
    np.max(y_test_pred_doy_PI_diff[:,1]))
print('Mean difference between actual MT and SSNN predictions in FCT:',
    np.average(y_test_pred_doy_SS_diff[:,1]))
print('minimum difference between actual MT and SSNN predictions in FCT:',
    np.min(y_test_pred_doy_SS_diff[:,1]))
print('maximum difference between actual MT and SSNN predictions in FCT:',
    np.max(y_test_pred_doy_SS_diff[:,1]))

my_array = np.concatenate((month_VTA,y_VTA,y_FEM_VTA,
y_VTA_pred_DDNN,y_VTA_pred_PINN,y_VTA_pred_SSNN),axis=1)
DOY_plot= pd.DataFrame(my_array, columns = ['month','y_test','y_FEM',
'y_test_pred_DD','y_test_pred_PI','y_test_pred_SS'])

y_test_doy=(DOY_plot.groupby('month', as_index=False)[
'y_test'].mean())

y_test_FEM_doy=(DOY_plot.groupby('month', as_index=False)[
'y_FEM'].mean())

y_test_pred_doy_DD=(DOY_plot.groupby('month', as_index=False)[
'y_test_pred_DD'].mean())

y_test_pred_doy_PI=(DOY_plot.groupby('month', as_index=False)[
'y_test_pred_PI'].mean())

y_test_pred_doy_SS=(DOY_plot.groupby('month', as_index=False)[
'y_test_pred_SS'].mean())
['y_test_pred_SS'].mean()

y_test_diff=y_test_doy-y_test_doy
y_test_FEM_doy_diff= y_test_doy-y_test_FEM_doy
y_test_pred_doy_DD_diff=y_test_doy-y_test_pred_doy_DD
y_test_pred_doy_PI_diff=y_test_doy-y_test_pred_doy_PI
y_test_pred_doy_SS_diff=y_test_doy-y_test_pred_doy_SS

plt.figure(figsize=(12,6))
plt.plot(y_test_doy[:,0],y_test_FEM_doy_diff[:,1], color='k',label='FEHT')
plt.plot(y_test_doy[:,0],y_test_pred_doy_DD_diff[:,1], color='k',linestyle='--',
label='DDNN')
plt.plot(y_test_doy[:,0],y_test_pred_doy_PI_diff[:,1], color='b',
label='PINN')
plt.plot(y_test_doy[:,0],y_test_pred_doy_SS_diff[:,1], color='k',linestyle=':',
label='SSNN')
plt.plot(y_test_doy[:,0],y_test_diff[:,1], color='r')
plt.xlabel('Month')
plt.ylabel('Actual-predicted MT in FCT ')
plt.yticks([-10,-5,0,5,10,15,20])
plt.legend()
plt.savefig("diff_Month_FCT.jpg",bbox_inches='tight',dpi=250)
plt.show()

#stats difference
print('Mean difference between actual MT and FEM predictions in FCT:',

280
np.average(y_test_FEM_doy_diff[:,1])
print('minimum difference between actual MT and FEM predictions in FCT:',
    np.min(y_test_FEM_doy_diff[:,1]))
print('maximum difference between actual MT and FEM predictions in FCT:',
    np.max(y_test_FEM_doy_diff[:,1]))
print('Mean difference between actual MT and DDNN predictions in FCT:',
    np.average(y_test_pred_doy_DD_diff[:,1]))
print('minimum difference between actual MT and DDNN predictions in FCT:',
    np.min(y_test_pred_doy_DD_diff[:,1]))
print('maximum difference between actual MT and DDNN predictions in FCT:',
    np.max(y_test_pred_doy_DD_diff[:,1]))
print('Mean difference between actual MT and PINN predictions in FCT:',
    np.average(y_test_pred_doy_PI_diff[:,1]))
print('minimum difference between actual MT and PINN predictions in FCT:',
    np.min(y_test_pred_doy_PI_diff[:,1]))
print('maximum difference between actual MT and PINN predictions in FCT:',
    np.max(y_test_pred_doy_PI_diff[:,1]))
print('Mean difference between actual MT and SSNN predictions in FCT:',
    np.average(y_test_pred_doy_SS_diff[:,1]))
print('minimum difference between actual MT and SSNN predictions in FCT:',
    np.min(y_test_pred_doy_SS_diff[:,1]))
print('maximum difference between actual MT and SSNN predictions in FCT:',
    np.max(y_test_pred_doy_SS_diff[:,1]))

fig, ax = plt.subplots(figsize=(18,10))
medianprops = {'color': 'black', 'linewidth': 2}
y4=[-20,15]
plt.plot(x4, y4, c='k',linestyle='--')

plt.xticks([1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16],
           ['FEHT','DDNN','PINN','SSNN','FEHT','DDNN','PINN','SSNN','FEHT','DDNN','PINN','SSNN','FEHT','DDNN','PINN','SSNN'],fontsize=15,rotation=45)
plt.yticks([-19,-15,-10,-5,0,5,10,15],fontsize=15)
plt.margins(x=0, y=0)
ax.set(xlabel='Model', ylabel='The difference between the actual and predicted MT ($\circ$C)'
ax.xaxis.get_label().set_fontsize(15)
ax.yaxis.get_label().set_fontsize(15)
ax.text(2.3,13, 'Test 1', style='italic',fontsize=15)
ax.text(6.3,13, 'Test 2', style='italic',fontsize=15)
ax.text(10.3,13, 'Test 3', style='italic',fontsize=15)
ax.text(14,13, 'Average', style='italic',fontsize=15)

plt.savefig("overall_error.jpg",bbox_inches='tight',dpi=250)
plt.show()
Appendix C

Third Appendix: codes for Chapter 5

C.1 Compartmental process-based model code

#Import all the libraries required
#pandas for retrieving the data from the excel sheet
import pandas as pd
#tensorflow is for defining all the neural networks operations
import tensorflow as tf
#numpy and scipy for performing the calculations and the data management
import numpy as np
from scipy.sparse import diags, lil_matrix
from scipy.sparse.linalg import spsolve
from numpy import any
#matplot is for data plotting and results visualization
import matplotlib as mpl
import matplotlib.pyplot as plt
from matplotlib.ticker import MultipleLocator, FormatStrFormatter, MaxNLocator
#time for computing the run time
import time
import datetime
# Math library for calculations
import math
from math import pi

# Importing and visualizing data
df = pd.DataFrame(pd.read_excel('Datasets/IN_dataset.xlsx', sheet_name = 'Data'))

# Indexing the data and time
df = df.set_index('D/T')

# Resampling the data on hourly basis
df = df.resample('H').mean()

# Inplace N/A values with the mean
df.fillna(df.mean(), inplace = True)

df.head(3)

<table>
<thead>
<tr>
<th>D/T</th>
<th>Julian Day</th>
<th>Month</th>
<th>AAT</th>
<th>WS</th>
<th>WD</th>
<th>RH</th>
<th>SR</th>
<th>RF</th>
<th>Agitation</th>
<th>Crust</th>
<th>Ag</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>2008-09-16 18:00:00</td>
<td>259.770833</td>
<td>9.0</td>
<td>20.080</td>
<td>1.39479</td>
<td>283.666283</td>
<td>52.70</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2008-09-16 19:00:00</td>
<td>259.812500</td>
<td>9.0</td>
<td>20.620</td>
<td>0.85000</td>
<td>293.350000</td>
<td>47.38</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2008-09-16 20:00:00</td>
<td>259.854167</td>
<td>9.0</td>
<td>21.585</td>
<td>1.05000</td>
<td>267.150000</td>
<td>46.62</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>D/T</th>
<th>SR</th>
<th>RF</th>
<th>Agitation</th>
<th>Crust</th>
<th>Ag</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>2008-09-16 18:00:00</td>
<td>775.45</td>
<td>0.000417</td>
<td>0.0</td>
<td>0.0</td>
<td>99.30</td>
<td></td>
</tr>
<tr>
<td>2008-09-16 19:00:00</td>
<td>683.70</td>
<td>0.000417</td>
<td>0.0</td>
<td>0.0</td>
<td>99.20</td>
<td></td>
</tr>
<tr>
<td>2008-09-16 20:00:00</td>
<td>553.90</td>
<td>0.000417</td>
<td>0.0</td>
<td>0.0</td>
<td>99.15</td>
<td></td>
</tr>
</tbody>
</table>

NH3_g/m2/s  NH3_mg/m2/d
def calculate_storage_size(rain, freeboard, runoff_coefficient, runoff_area, wash_water_volume, storage_days, number_of_cows, manure_production, manure_density, bedding_mass, bedding_density, waste_water_density, depth_0, total_height):
    # this function calculates the area of the manure storage structure
    volume_reduction_factor = 0.3
    runoff_volume = runoff_coefficient * runoff_area
    bedding_volume = (
        volume_reduction_factor * number_of_cows * bedding_mass * storage_days / 
        bedding_density
    )
    waste_water_volume = wash_water_volume / waste_water_density * storage_days
    total_storage_volume = (
        number_of_cows * manure_production * storage_days / manure_density + 
        waste_water_volume + bedding_volume + runoff_volume
    )
    manure_height = total_height - depth_0 - rain - freeboard
    surface_area = total_storage_volume / manure_height
    return surface_area

def soil_temperature(Soildepth, t_diff, soilparam):
    # Ta = average soil temperature (C)
    # A0 = annual amplitude of the surface soil temperature (C)
    # Soildepth = soil depth from surface to bottom of the tank (m)
# d = damping depth (m)
# Dh = thermal diffusivity of soil (m^2/day) = k/Cs
# Tbot = temperature at bottom of the manure storage
Ta, A0, Dh = soilparam

omega = 2 * pi / 365*24  # 1/hour

d = (2 * Dh / omega) ** 0.5

cos_factor = math.cos((2 * pi * (t_diff) / 365*24) - (Soildepth / d) - (pi / 2))
sin_factor = math.sin((2 * pi * (t_diff) / 365*24) - (Soildepth / d) - (pi / 2))

Tbot = Ta + A0 * (math.exp(-Soildepth / d)) * sin_factor  # C
return Tbot

def manure_depth_calculator(Area, MD, BD, WaterIN, ManureIN, BeddingIN, ManureLA, PRECIP, Tair, Wind, RH, Depth, Evaporationparam, dz):

    # Constants
    a, h0, ce = Evaporationparam
    rd = 287.04

    ur = Wind * (2 / h0) ** a

    temperature_surface = 5.0 + Tair * 0.75

    e_ta = 0.61078 * np.exp((Tair * 17.269) / (237.3 + Tair))
    e_a = RH * e_ta / 100
    e_s = 0.61078 * np.exp((17.269 * temperature_surface) / (237.3 + temperature_surface))

    evaporation_flux = 0.622 / (rd * (temperature_surface + 273.15)) * (e_s - e_a) * ur * ce

    water_evaporation = evaporation_flux * Area * 3600  # unit: [kg/hr]
# Change of manure volume per hour

delta_volume = ((ManureIN / MD)/24 + (BeddingIN / BD)/24 + 
\( (WaterIN / 1000)/24 + (PRECIP * Area / 100) - (water_evaporation / 1000) \)) # unit: [m^3/hr]

volume_remain = np.cumsum(delta_volume) - (np.cumsum(delta_volume) * 
\ManureLA / 100) # unit: [m^3/hr]

# Change of manure depth per day

delta_depth = (delta_volume - (delta_volume * ManureLA/100)) / Area # unit:[m/hr]
depth = round(delta_depth * 1 / dz) / (1 / dz)

manure_depth = Depth + delta_depth

return depth, manure_depth

def FDsolver1Dheat(n,Height,Htdelta,dz,Thermalparam,Tbot,Tsurface,dt,Told):

# 1D Finite Difference Solver for Heat Transfer in a manure storage
# This function takes following inputs
#=========================================================================
# Height -- Total height of manure in tank,[m]
# Htdelta -- Height of newly added manure layer,[m]
# dz -- z-direction discretization length,[m]
# Qg -- Heat generation of stored manure,[W/m^3]
# Parameters -- Array of manure parameters
# Tbot -- Temperature at bottom of the storage,[C]
# Tsurface -- Surface temperature of manure,[C]
# Told -- Initial value for old temperature field,[C]

288
# dt -- Size of time step,[sec]
# nt -- Number of time step computed,[days]

# Initial uniform temperature profile

# Call aFDSolver function

# Physical and thermal properties of manure
kappa, density, cp, Qg = Thermalparam

# Numerical parameters
nz= round(Height/dz)+1 # Number of nodes in z-direction
nznew = round(Htdelta/dz) # Numner of new nodes in z-direction
q = Qg # Uniform heat generation [W/m^3]

# initial uniform temperature profile
Tint = Tsurface

# construcion of coefficient matrix A
s = kappa*dt/((dz**2)*density*cp)
A = lil_matrix((nz,nz))
for x in range (1,nz-1):
    A[x,x-1] = -s
    A[x,x] = (1+2*s)
    A[x,x+1] = -s

# Bottom boundary: fixed temperature
A[0,0] = 1
A[nz-1,nz-1] = 1
# Set ICs for the simulation

\[ T = \text{np.zeros}([nz,1]) \]

# T(1:nz) = Told

if n==0:
    T[1:nz-1] = Tint
elif Told.shape > T.shape:
    T = Told[:len(T)-1]
else:
    T[0:len(Told)] = Told  # portion of the matrix with old T
    T[len(Told):nz] = Tint  # portion of the matrix with new T

# Solve the matrix for different time step

time = 0  # starting time

# compute rhs for center domain

rhs = \text{np.zeros}([nz,1])

for i in range (1,nz-1):
    rhs[i] = T[i] + q/density/cp*dt

# top and bottom

rhs[0] = Tbot  # Fixed temperature at bottom
rhs[nz-1] = Tsurface  # Fixed temperature at the surface

# compute temperature at new time step

T_vector = \text{np.linalg.lstsq}(A.toarray(),rhs,rcond=None)[0]

Tnew = T_vector

# Update the T field

T = Tnew

# increase time

time = time + dt
def manureTAN(n, mTANold, mTANnew, Ntemp, mONold, mONnew, nz, nznew, MineralizationParam):
    
    Calculates concentration of TAN at each node

    1. Mineralization of organic N to TAN
    2. Remaining organic nitrogen after mineralization
    3. Concentration TAN at each node

    Parameters:
    mTANold -- conc. of TAN remaining at end of previous hour (kg/m^3)
mTANnew -- conc. of TAN generated by new manure (kg/m^3)
Ntemp -- manure temperature at each time step (C)
mONold -- conc. of organic nitrogen remaining at end of previous hour (kg/m^3)
mONnew -- conc. of organic nitrogen generated by new manure (kg/m^3)
nz -- total number of nodes (unitless)
nznew -- number of new nodes added (unitless)
    MineralizationParam -- parameters for organic nitrogen mineralization,
a tuple with:
        theta -- temperature coefficient
        kON20 -- rate constant at temp 20C (1/day)

    Returns:
    mTAN -- conc. of TAN generated (kg/m^3)
ON -- organic nitrogen remaining after mineralization (kg/m^3)
if n == 0:
    mTANold = mTANold*np.ones((nz,1))
    mONold = mONold*np.ones((nz,1))

# Unpack mineralization parameters
theta, kON20 = Mineralizationparam
kON20 = kON20/24 #c converter unit: 1/d -> 1/hr

# Create vectors for creating new mON and TAN concentration profile
mON = np.zeros([nz,1]) # concentration of organic nitrogen in manure
TANtot = np.zeros([nz,1]) # concentration of total TAN in manure

# Create vectors for creating new concentration profile
if len(mONold) > len(mON):
    mON = mONold[:len(mON)-1]
else:
    mON[0:len(mONold)] = mONold
    # portion of the vector with old mON
mON[len(mONold):nz]= mONnew
    # portion of the vector with new mON

# Create vectors for creating new concentration profile
if len(mTANold) > len(TANtot):
    TANtot = mTANold[:len(mON)-1]
    mTAN = TANtot
    # Total TAN after mineralization [kg N/m^-3]
ON = mON

# Organic Nitrogen remaining after mineralization [kg/m^3]
else:
    TANtot[0:len(mTANold)] = mTANold
    # portion of the vector with old TAN
    TANtot[len(mTANold):nz]= mTANnew
    # portion of the vector with new TAN
    kON = kON20 * theta**(Ntemp-20)
    # correction of rate of mineralization for temperature
    mTANgen = kON * mON
    # TAN produced from [kg N/m^3]
    mTAN = TANtot + mTANgen
    # Total TAN after mineralization [kg N/m^3]
    ON = mON - mTANgen
    # Organic Nitrogen remaining after mineralization [kg/m^3]

return mTAN, ON

def FDsolver1Dmass(n, Height, dz, Diffusionparam, Cold, J, dt):
    # Calculate number of nodes in z-direction
    nz = round(Height/dz) + 1

    # Physical properties of manure
    D = Diffusionparam #unit: [m^2/s]

    # Create coefficient matrix using scipy.sparse.diags
    s = D * dt / dz**2
    A = diags([-s, 1+2*s, -s], [-1, 0, 1], shape=(nz, nz)).tocsr()
A[0,1] = -2*s  # Bottom boundary: zero flux
A[-1,-2] = -2*s  # Top boundary: changing flux

# Initialize concentration field and storage
C = np.zeros((nz,1))
C[0:len(Cold)] = Cold

# Solve for different time steps

time = 0

    # Compute right hand side vector
rhs = np.zeros((nz,1))
rhs[1:-1] = C[1:-1]
rhs[0] = C[0] - 2*s*dz/D  # bottom
rhs[-1] = C[-1] + J*2*s*dz/D  # top

    # Solve for new concentration field
Cnew = np.linalg.solve(A.toarray(), rhs)

    # Update concentration field
C = Cnew

    # Increase time
    time += dt

return C
def ammoniaemission(Tliq,Tai,P,pH,CTAN,WindS,WindH,z0,CAIR):
# "ammoniaemission" function calculates NH3 emission (flux) from the manure surface
#
#
#=========================================================================
uz = WindS
z = WindH

U8 = uz*(np.log(8/z0))/np.log(z/z0) # wind speed at 8m height
Ka = 10**(0.0897-(2729/(Tliq+273.15))) # D dissociation constant for NH3 Jayaweera and Mikkelsen (1990)
F = 1/(1+(10**(-pH)/Ka)) # Fraction of free ammonia

Dh2oo2 = (7.28236*(10**-15)*(Tliq+273.15))/(np.exp((1622/(Tliq+273.15))-12.40581)) #Diffusivity of oxygen in water (m2/s)
Dairh2o = (3.00123*(10**-8)*(Tai+273.15)**1.75)/(25.5231*P) # Diffusivity of water vapor in air (m2/s)
Dh2onh3 = (6.14526*(10**-15)*(Tliq+273.15))/(np.exp((1622/(Tliq+273.15))-12.40581)) #Diffusivity of NH3 in water(m2/s)
Dairnh3 = ((3.05519*10**-8)*(Tai+273.15)**1.75)/(26.8288*P) # Diffusivity of ammonia in air (m2/s)

kL =((1.6761*10**-6)*np.exp(-0.236*U8))*(Dh2onh3/Dh2oo2)**0.57 # Mass transfer coefficient in the liquid phase(m/s)
kG =((5.1578*10**-5)+ (1.954*10**-3)*U8)*(Dairnh3/Dairh2o)**0.67 # Mass transfer coefficient in the gas phase (m/s)

H =((2.39*10**5)/(Tliq+273.15))*np.exp(-4151/(Tliq+273.15))# Henry's constant
KL = (kL*H*kG)/(H*kG+kL) # overall mass transfer coefficient (m/s)

NH3em = KL*(F*CTAN-CAIR)*10**3 # ammonia flux (kg/m^2/s)
return NH3em,Ka,F

#=========================================================================
# "ManureStorageEmission"- simulates heat transfer, biochemical reactions,
# mass transfer and emission of gases from a manure storage

295
# This driver performs following tasks
# 1. Set all the parameters
# 2. Call following functions
# i. FDsolver1Dheat - 1D Finite Difference Solver for Heat Transfer
# ii. FDsolver1Dmass - 1D Finite Difference Solver for Mass Transfer
#=========================================================================
# INPUTS
#=========================================================================
## Load weather data and time rom an external excel file
#-------------------------------------------------------------------------
# Date and Time
start_time = df.index[0]
end_time = df.index[-1]
num_hours = int(abs(start_time - end_time).total_seconds() / 3600)
num_days = int(abs(start_time - end_time).total_seconds() / (24 * 3600))

#weather data
Tair = df['AAT'] # ambient air temperature: unit [C]
PRECIP = df['RF'] # cummulative precipitation: unit [cm]
Wind = df['WS'] # Average wind velocity: unit [m/s]
RH = df['RH'] # Relative humidity: unit [%]

Agitation = df['Agitation'] #Agitation of manure that preceeds manure removal

## Herd and manure management constants

# 296
NAU = 3450 # dairy cows, each weighing 635 kg (1 animal unit = AU)
MW = 67 # manure produced per cow, [kg/day] (MWPS, 2000, MWPS-7)
B = 3.78 # chopped straw per cow, [kg/day] (MWPS, 1993, MWPS-18)
WW = 0 # wash per cow, [kg/day]
MD = 993 # density of manure, [kg/m^3] (MWPS, 2000, MWPS-7)
BD = 128 # density of bedding, [kg/m^3] (MWPS, 2000, MWPS-7)
WD = 1000 # density of water, [kg/m^3]

## Dimensions and properties of the storage

Totalheight = 4.8 # total depth allocated for the storage [m]
Depth0 = 0.3 # Depth of the residual manure [m]
Freeboard = 0.6 # Freeboard (24 inch)
Soildepth = 4.6 # soil depth from surface to bottom of the tank [m]
Rain = 0.6 # annual rainfall [m]
S25y = 0.154 # 25-year, 24-hour storm [m]
Runoffarea = 4046 # runoff area, [m^2] # approx. 1 acre 4046
sdays = 81 # number of days manure is stored [days]

## Initial conditions/ manure characteristics

mONnew = 1.387 # initial ON conc. in manure, [kg/m^3]
mTANnew = 1.089 # initial TAN conc. in manure, [kg/m^3]
pH = 7.2 # manure pH
CAIR = 0 # Ammonia conc. in air, [kg/m^3]
J = 0 # initial NH3 flux [kg/m^2/s]
## Create input data vectors for simulation period

dayManure = NAU*MW  # kg, total mass of manure moved to storage

dayBedding = NAU*B   # kg, total mass of bedding material moved to storage

ManureIN = dayManure  # manure

BeddingIN = dayBedding # bedding material

WaterIN = WW         # waste water

# accounting for manure removal events from the storage

ManureLA = []

for Ag in Agitation:
    if Ag == 0:
        ManureLA.append(0)
    else:
        ManureLA.append(95)

# Parameters for heat and mass transfer of stored manure

# Size of mesh dz

dz = 0.1 # y-direction discretization length: unit[m]

# Physical and thermal properties of manure

Thermalparam = [0.6814, MD, 1992, 1.2]

Thermalparam[0] = Thermal conductivity, [W/m/C], (Nayyeri et al., 2009)
Thermalparam[1] = Density, [kg/m$^3$], (MWPS, 1997)
Thermalparam[2] = Heat capacity, [J/kg/C], (Nayyeri et al., 2009)
Thermalparam[3] = Internal heat generation, [W/m$^3$], (Baral et al., 2013)

Diffusionparam = 2.5$\times$10$^{-9}$

Diffusionparam[0] = Diffusion coefficient of ammonia [m$^{-2}$/s$^{-1}$]
(Muck and Steenhuis, 1982)

for Tair1 in Tair:
    Tair1 = max(Tair1, 0.01)

Tsurface = 5.0 + 0.75*Tair

# Surface temperature of manure: unit[C],
(Preud' homme and Stefan, 1993), based on daily air temperature
Told = Tsurface[0]

# Initial temperature value for "Told" variable: unit[C],
("Told"-old temperature profile of manure)

# Parameters for soil temperature calculations
soilparam = [13, 28, 0.08]

soilparam(1) = Ta, average soil temperature (C)
soilparam(2) = A0, annual amplitude of the surface soil temperature (C)
soilparam(3) = Dh, thermal diffusivity of soil (m$^{-2}$/day)

# Control parameters for solver (FDSolver1Dheat & FDSolver1Dmass)
dt = 3600  # Size of a time step in seconds: unit [sec]
nt = 1     # Number of time steps 1 hr
Store = 24 # store data for every 24 steps (i.e., daily)

#-----------------------------------------------
# Parameters for organic nitrogen mineralization
#-----------------------------------------------
Mineralizationparam = [1.2, 0.06]
# Mineralizationparam(1)= Temperature coefficent (Zhang etal.,2005)
# Mineralizationparam(2)= Mineralization rate constant [1/day] (Zhang etal.,2005)

#-----------------------------------------------
# Parameters for NH3 emission
#-----------------------------------------------
z0 = 1*(10**-3)  # roughness height [m]
P = 1            # atmospheric pressure [atm]
WindH = 1.5      # anemometer height at which wind speed was measured [m]

#-----------------------------------------------
# Input parameters for evaporation
#-----------------------------------------------
Evaporationparam = [0.14, WindH, 2.81*(10**-3)]
"""
Evaporationparam[0] = parameter depends on surrounding terrain
Evaporationparam[1] = standard height at which wind speed is measured,[m]
Evaporationparam[2] = bulk aerodynamic transfer coeffcient (Ham, 1999)
# initialize matrices/vectors for storing data

maxznodene = 400
# max # of nodes on z direction
zMark = np.zeros([1,num_hours])
# contains # of z-nodes for each day (z-nodes change every day)
Tstore = np.zeros([maxznodene,num_hours]) # matrix to store Temp data for each day
Cstore = np.zeros([maxznodene,num_hours])
NH3flux = np.zeros([num_hours,1])
NH3fluxgN = np.zeros([num_hours,1])
NH3emission = np.zeros([num_hours,1])
CTANsurf = np.zeros([num_hours,1])
MHeight = np.zeros([num_hours,1])
Tbottom = np.zeros([num_hours,1])

# Call storage size funstion
Area = calculate_storage_size(Rain, Freeboard, S25y, Runoffarea, WW, sdays,
NAU, MW, MD, B, BD, WD, Depth0, Totalheight)

#Hourly loop
manure_depth_n = Depth0
#initial manure depth
for n in range(num_hours):
    # Call manure_depth function
    depth, manure_depth = manure_depth_calculator(Area, MD, BD, WaterIN,
                                                ManureIN, BeddingIN, ManureLA[n], PRECIP, Tair, Wind, RH, manure_depth_n, Evaporation)

    manure_depth_n = manure_depth[n]
    # Total depth of manure at nth time step (at ith day)
    depth_n = depth[n]
    # Height change of manure at nth time step (at ith day)
    Tsurface_n = Tsurface[n]
    # Manure surface temperature at nth time step (at ith day)

    ## Call soil temperature function
    time_diff_n = n
    Tbot_n = soiltemperature(Soildepth,time_diff_n,soilparam)
    Tbottom[n] = Tbot_n

    # Call FDsolver1Dheat function to calculate the temperature at each layer
    T, nz, nznew = FDsolver1Dheat(n, manure_depth_n, depth_n, dz, Thermalparam,
                                 Tbot_n, Tsurface_n, dt, Told)
    Told = T
    # Temperature profile of manure at end of the ith time step

    # initial ON and TAN in the manure
    if n == 0:
        mTANold = mTANnew # unit: [kg N/m^3]
mONold = mONnew # unit: [kg N/m^3]

# Call the manureTAN function to calculate the TAN generated at each node
for j in range(nz):
    mTAN, ON = manureTAN(n, mTANold, mTANnew, T, mONold, mONnew, nz, nznew,
                          Mineralizationparam)

# update manure ON and TAN values
mONold = ON
Cold = mTAN

# Call FDsolver1Dmass function for diffusion calculations
C = FDsolver1Dmass(n, manure_depth_n, dz, Diffusionparam, Cold, J, dt)
mTANold = C.reshape(nz, 1)

# Call ammoniaemission function
WindS = Wind[n]
Tliq = T[-1] # manure temperature of the surface layer/element
CTAN = C[-1] # TAN concentration of the surface layer/element
Tai = Tair[n]
NH3em, Ka, F = ammoniaemission(Tliq, Tai, P, pH, CTAN, WindS, WindH, z0, CAIR)

# NH3 emission from the manure surface
if NH3em < 0:
    NH3em = 0
\[ \text{NH3fluxgN}[n] = \text{NH3em} \times 24 \times 3600 \times (10^{\text{**3}}) \quad \# \text{NH3 flux (g N/m}^2\text{/day)} \]

```python
if n%24 == 0 and n!=0:
    av_daily = np.mean(NH3fluxgN[n-24:])
    day = n/24
    print(f'The average daily ammonia flux is \{av_daily \times 10^{3:.2f}\} mg N/m^2/day in day \{int(day)\}''

NH3fluxgN.reshape(7687,)
plt.rcParams["font.family"] = "Arial"
plt.figure(figsize=(14,6))
plt.plot(df.index, df['NH3_mg/m2/d']/1000, c = 'gray')
plt.plot(df.index[1:], NH3fluxgN/1000, c = 'black')
plt.xlabel("Date", fontsize = 15)
plt.ylabel("Ammonia $g/m^2/d$", fontsize = 15)
plt.xticks(fontsize =15)
plt.yticks(fontsize =15)
plt.savefig("PBM_without_HPT.eps", dpi=1200)
plt.show()
```
C.2 The physics-informed LSTM model code

# Import all the libraries required

# pandas for retrieving the data from the excel sheet
import pandas as pd

# PyTorch is for defining all the neural networks operations
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.autograd import Variable
from pprint import pformat

# numpy and scipy for performing the calculations and the data management
import numpy as np
from scipy.sparse import diags, lil_matrix
from scipy.sparse.linalg import spsolve
# Optuna for hyperparameters tuning

```python
import optuna

# scikit-learn for data processing and evaluation
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# Matplotlib is for data plotting and results visualization
import matplotlib.pyplot as plt

# Time for computing the run time
import time
import datetime

# Math library for calculations
import math
from math import pi

# Importing and visualizing data
```

df = pd.DataFrame(pd.read_excel('Datasets/IN_dataset.xlsx', sheet_name = 'Data'))

# Indexing the date and time
```
df = df.set_index('D/T')

# Resampling the data on hourly basis
```
df = df.resample('H').mean()

# Inplace N/A values with the mean
```
df.fillna(df.mean(), inplace = True)
df.head(3)

<table>
<thead>
<tr>
<th>Julian Day</th>
<th>Month</th>
<th>AAT</th>
<th>WS</th>
<th>WD</th>
<th>RH</th>
</tr>
</thead>
<tbody>
<tr>
<td>D/T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2008-09-16</td>
<td>18:00:00</td>
<td>259.770833</td>
<td>9.0</td>
<td>20.080</td>
<td>1.39479</td>
</tr>
<tr>
<td>2008-09-16</td>
<td>19:00:00</td>
<td>259.812500</td>
<td>9.0</td>
<td>20.620</td>
<td>0.85000</td>
</tr>
<tr>
<td>2008-09-16</td>
<td>20:00:00</td>
<td>259.854167</td>
<td>9.0</td>
<td>21.585</td>
<td>1.05000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SR</th>
<th>RF</th>
<th>Agitation</th>
<th>Crust</th>
<th>Ag</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>D/T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2008-09-16</td>
<td>18:00:00</td>
<td>775.45</td>
<td>0.000417</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>2008-09-16</td>
<td>19:00:00</td>
<td>683.70</td>
<td>0.000417</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>2008-09-16</td>
<td>20:00:00</td>
<td>553.90</td>
<td>0.000417</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NH3_g/m2/s</th>
<th>NH3_mg/m2/d</th>
</tr>
</thead>
<tbody>
<tr>
<td>D/T</td>
<td></td>
</tr>
<tr>
<td>2008-09-16</td>
<td>18:00:00</td>
</tr>
<tr>
<td>2008-09-16</td>
<td>19:00:00</td>
</tr>
<tr>
<td>2008-09-16</td>
<td>20:00:00</td>
</tr>
</tbody>
</table>

#Visualize the ammonia flux with the agitation data
fig, ax1 = plt.subplots(figsize = (8,5))

ax1.plot(df['NH3_mg/m2/d'], c = 'blue')
ax1.set_ylabel('NH3 flux (mg/m2/d)', color = 'blue')

ax2 = ax1.twinx()
ax2.plot(df['Agitation'], c = 'k', lambda1 = 0.8, linestyle = ':')
ax2.set_ylabel('Agitation', color = 'k')

ax1.set_xlabel('Date')
plt.show()

#defining the data used in the model

""

Features used in the model
Month - The month of the measurement
AAT - Ambient air temperature (C)
WS - Wind speed at 1.5 m above the surface (m/s)
WD - Wind direction at 1.5 m above the surface (degrees)
RH - Relative humidity (%)
RF - cumulative rainfall (m)
Ag - manure agitation: 1 means the manure was agitated, 0 means it was not

Model target value
NH3\_mg/m^2/d - the ammonia emissions in mg/m^2/d

# a function to split the features (Month, AAT, WS, WD, RH, RF, Ag) and the target value
def feature_label_split(df, target_col):
    y = df[[target_col]]
    X = df.drop(columns=[target_col])
    return X, y

# a function to split the data into training, validation, and testing sets.
def train_val_test_split(df, target_col, test_ratio):
    val_ratio = test_ratio / (1 - test_ratio)
    X, y = feature_label_split(df, target_col)
    X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                     test_size=test_ratio, shuffle=True)
    X_train, X_val, y_train, y_val = train_test_split(X_train, y_train,
                                                   test_size=val_ratio, shuffle=True)
    return X_train, X_val, X_test, y_train, y_val, y_test

# splitting the data into features and target, and training, validation, and testing
X, y = feature_label_split(df_data, 'NH3\_mg/m2/d')
X_train, X_val, X_test, y_train, y_val, y_test = train_val_test_split(df_data, 'NH3_mg/m2/d', 0.2)
n_features = X_train.shape[1]

print("number of features %d " %n_features)
print("Shapes of data X, Y: ",X.shape, y.shape)
print("Shapes of training X, Y: ",X_train.shape, y_train.shape)
print("Shapes of validation X, Y: ",X_val.shape, y_val.shape)
print("Shapes of testing X, Y: " ,X_test.shape, y_test.shape)

number of features 7
Shapes of data X, Y: (7688, 7) (7688, 1)
Shapes of training X, Y: (4612, 7) (4612, 1)
Shapes of validation X, Y: (1538, 7) (1538, 1)
Shapes of testing X, Y: (1538, 7) (1538, 1)

#standardize the data by features scaling and target scaling
from sklearn.preprocessing import StandardScaler

scaler_features = StandardScaler()
scaler_target = StandardScaler()

#fitting the scaler_features function using the training features
X_train_arr = scaler_features.fit_transform(X_train)

#scaling the validation and testing features using scaler_features
X_val_arr = scaler_features.transform(X_val)
X_test_arr = scaler_features.transform(X_test)
#fitting the scaler_target function on the training NH3 flux
y_train_arr = scaler_target.fit_transform(y_train.values)

#scaling the validation and testing NH3 flux data using scaler_target
y_val_arr = scaler_target.transform(y_val.values)
y_test_arr = scaler_target.transform(y_test.values)

#convert all the array to tensors for the RNN
train_features = torch.Tensor(X_train_arr)
train_targets = torch.Tensor(y_train_arr)
val_features = torch.Tensor(X_val_arr)
val_targets = torch.Tensor(y_val_arr)
test_features = torch.Tensor(X_test_arr)
test_targets = torch.Tensor(y_test_arr)

#merging the features and the target values into train, val, and
test datasets for training, validation, and testing
train = torch.utils.data.TensorDataset(train_features, train_targets)
val = torch.utils.data.TensorDataset(val_features, val_targets)
test = torch.utils.data.TensorDataset(test_features, test_targets)

#importing and visualizing data
df_Kupper = pd.DataFrame(pd.read_excel('Datasets/Kupper_dataset.xlsx',
                                      sheet_name = 'Data'))

#Indexing the dat and time
df_Kupper = df_Kupper.set_index('D/T')

#resampling the data on hourly basis
df_Kupper = df_Kupper.resample('H').mean()

#inplace N/A values with the mean
df_Kupper.fillna(df_Kupper.mean(), inplace = True)
df_Kupper.head(3)
df_Kupper_data = df_Kupper[['Month','AAT','WS','WD','RH','RF', 'Agitation', 'NH3_mg/m2/d']]

X_Kupper, y_Kupper = feature_label_split(df_Kupper_data, 'NH3_mg/m2/d')
X_Kupper_arr = scaler_features.transform(X_Kupper)
y_Kupper_arr = scaler_target.transform(y_Kupper.values)

#convert all the array to tensors for the RNN
test_Kupper_features = torch.Tensor(X_Kupper_arr)
test_Kupper_targets = torch.Tensor(y_Kupper_arr)

#merging the features and the target values into test dataset
test_Kupper = torch.utils.data.TensorDataset(test_Kupper_features, test_Kupper_targets)
def physics_informed_predictions(x, Diffusionparam, KON20, init_ON, pH):
    x = x.view([x.shape[0],x.shape[2]])
    batch_size = x.shape[0]

    #transform the scaled training features back for the PBM
    x = scaler_features.inverse_transform(x)

    df = pd.DataFrame(x, columns = ['Month','AAT','WS','WD','RH','RF', 'Agitation'])

    #-----------------------------------------------
    ## Herd and manure management constants
    #-----------------------------------------------
    ## Herd and manure management constants
NAU = 3450  # dairy cows, each weighing 635 kg (1 animal unit = AU)
MW = 67    # manure produced per cow, [kg/day] (MWPS,2000, MWPS-7)
B = 3.78   # chopped straw per cow, [kg/day] (MWPS,1993, MWPS-18)
WW = 0     # wash per cow, [kg/day]
MD = 993   # density of manure, [kg/m^3] (MWPS,2000, MWPS-7)
BD = 128   # density of bedding, [kg/m^3] (MWPS,2000, MWPS-7)
WD = 1000  # density of water, [kg/m^3]

## dimensions and properties of the storage
Totalheight = 4.8  # total depth allocated for the storage [m]
Depth0 = 0.3       # Depth of the residual manure [m]
Freeboard = 0.6    # Freeboard (24 inch)
Soildepth = 4.6    # soil depth from surface to bottom of the tank (m)
Rain = 0.6         # annual rainfall [m]
S25y = 0.154       # 25-year, 24-hour storm [m]
Runoffarea = 4046  # runoff area, [m^2] # approx. 1 acre 4046
sdays = 81         # number of days manure is stored [days]

## Initial conditions/ manure characteristics
mONnew = init_ON   # initial ON conc. in manure, [kg/m^3]
mTANnew = 0.78 * mONnew # initial TAN conc. in manure, [kg/m^3]
#pH = 7.2 # manure pH
CAIR = 0           # Ammonia conc. in air, [kg/m^3]
J = 0              # initial NH3 flux [kg/m^2/s]
## Create input data vectors for simulation period

dayManure = NAU*MW # kg, total mass of manure moved to storage

dayBedding = NAU*B # kg, total mass of bedding material moved to storage

ManureIN = dayManure # manure

BeddingIN = dayBedding # bedding material

WaterIN = WW # waste water

batch_size = x.shape[0]

# weather data

Tair = df['AAT'] # ambient air temperature: unit [°C]

PRECIP = df['RF'] # cumulative precipitation: unit [cm]

Wind = df['WS'] # Average wind velocity: unit [m/s]

RH = df['RH'] # Relative humidity: unit [%]

Agitation = df['Agitation'] # Agitation of manure that precedes manure removal

# accounting for manure removal events from the storage(#)

ManureLA = []

for Ag in Agitation:
    if Ag == 0:
        ManureLA.append(0)
    else:
        ManureLA.append(95)
# Parameters for heat and mass transfer of stored manure

# Size of mesh dz

dz = 0.01 #y-direction discretization length: unit[m]

# Physical and thermal properties of manure

Thermalparam = [0.6814, MD, 1992, 1.2]

Thermalparam[0] = Thermal conductivity, [W/m/C], (Nayyeri et al., 2009)
Thermalparam[1] = Density, [kg/m^3], (MWPS, 1997)
Thermalparam[2] = Heat capacity, [J/kg/C], (Nayyeri et al., 2009)
Thermalparam[3] = Internal heat generation, [W/m^3], (Baral et al., 2013)

#Diffusionparam 2.5*10**-9 Diffusion coefficient of ammonia [m^2/s^-1]
# (Muck and Steenhuis, 1982)

for Tair1 in Tair:
    Tair1 = max(Tair1, 0.01)

Tsurface = 5.0 + 0.75*Tair

# manure surface temperature: unit[C], (Preud'homme and Stefan, 1993)
Told = Tsurface[0]

# Initial temperature value for "Told" variable: unit[C]
# "Told" represents the old temperature profile of manure
# Parameters for soil temperature calculations
soilparam = [13, 28, 0.08]

soilparam(1) = Ta, average soil temperature (C)
soilparam(2) = A0, annual amplitude of the surface soil temperature (C)
soilparam(3) = Dh, thermal diffusivity of soil (m^2/day)

# Control parameters for solver (FDSolver1Dheat & FDSolver1Dmass)
dt = 3600 # Size of a time step in seconds: unit [sec]

# Parameters for organic nitrogen mineralization
Mineralizationparam = [1.2, KON20]
# Mineralizationparam[0]= Temperature coefficient (Zhang etal.,2005)

# Parameters for NH3 emission
z0 = 1*(10**-3) # roughness height [m]
P = 1 # atmospheric pressure [atm]
WindH = 1.5 # anemometer height at which wind speed was measured [m]

# Input parameters for evaporation

316
Evaporationparam = [0.14, WindH, 2.81*(10**-3)]

""
Evaporationparam[0] = parameter depends on surrounding terrain
Evaporationparam[1] = standard height at which wind speed is measured,[m]
Evaporationparam[2] = bulk aerodynamic transfer coefficient (Ham, 1999)
""

## initialize matrices/vectors for storing data

## Call storage size funstion
Area = calculate_storage_size(Rain, Freeboard, S25y, Runoffarea, WW, sdays,
NAU, MW, MD, B, BD, WD, Depth0, Totalheight)

#Hourly loop
for n in range(batch_size):
    # Call manuredepth function
    depth, manure_depth = manure_depth_calculator(Area, MD, BD, WaterIN, ManureIN, BeddingIN,
    ManureLA, PRECIP, Tair, Wind, RH, Depth0, Evaporationparam, dz)

    manure_depth_n  = manure_depth[n]
# Total depth of manure at nth time step (at ith day)
\[
\text{depth}_n = \text{depth}[n]
\]
# Height change of manure at nth time step (at ith day)
\[
\text{Tsurface}_n = \text{Tsurface}[n]
\]
# Manure surface temperature at nth time step (at ith day)

## Call soiltemperature function
\[
\text{time}_\text{diff}_n = n
\]
\[
\text{Tbot}_n = \text{soiltemperature(Soildepth, time}_\text{diff}_n, \text{soilparam)}
\]
\[
\text{Tbottom}[n] = \text{Tbot}_n
\]

# Call FDsolver1Dheat function to calculate the temperature at each layer
\[
\text{T}, \text{nz}, \text{nznew} = \text{FDsolver1Dheat(n, manure_depth}_n, \text{depth}_n, \text{dz, Thermalparam, Tbot}_n, \text{dt, Told)}
\]
\[
\text{Told} = \text{T}
\]
# Temperature profile of manure at end of the ith time step

# initial ON and TAN in the manure
if n == 0:
    \[
    \text{mTANold} = \text{mTANnew} \# \text{unit: [kg N/m}^{-3}]\n    \]
    \[
    \text{mONold} = \text{mONnew} \# \text{unit: [kg N/m}^{-3}]\n    \]

# Call the manureTAN function to calculate the TAN generated at each node
for j in range (nz):
    \[
    \text{mTAN}, \text{ON} = \text{manureTAN(n, mTANold, mTANnew, T, mONold, mONnew, nz, nznew,}
    \]
    \[
    \text{Mineralizationparam)}
    \]
#update manure ON and TAN values

mONold = ON
Cold = mTAN

# Call FDsolver1Dmass function for diffusion calculations
C = FDsolver1Dmass(n, manure_depth_n, dz, Diffusionparam * 10**-9, Cold, J, dt)
mTANold = C.reshape(nz, 1)

# Call ammoniaemission function
WindS = Wind[n]
Tliq = T[-1]  # manure temperature of the surface layer/element
CTAN = C[-1]  # TAN concentration of the surface layer/element
Tai = Tair[n]
NH3em, Ka, F = ammoniaemission(Tliq, Tai, P, pH, CTAN, WindS, WindH, z0, CAIR)

# NH3 emission from the manure surface
NH3fluxgN[n] = NH3em * 84000 * 10**3  # NH3 flux (mg N/m^2/day)

# scale the NH3 flux calculated by the process-based model and convert
# it into torch tensor for the RNN model
NH3fluxgN = NH3fluxgN.astype('float32')
NH3fluxgN = scaler_target.transform(NH3fluxgN)
NH3fluxgN = torch.from_numpy(NH3fluxgN)
return NH3fluxgN

class Optimization_PI:
    ""

Optimization_PI class that entails:
1. initializing the instance variables using __init__ function
2. define the training steps which in the train_step function, which are:
   a. training the model the model
   b. calculating the predicted output by the model
   c. calculating the loss between the predicted and target values
   d. optimized and update the model parameters
3. defining model training as follows:
   a. repeat the following steps for n_epochs
   b. split the training data into batchs based on the batch size
   c. run the train_step function with physics informed loss function
   d. calculate the average training loss for each batch
   e. use backward propagation to update model parameters (weights and biases)
   f. follow same steps for validation dataset but without backward propagation
4. define the evaluate function that tests the model
on the test dataset with no back propagation

```
def __init__(self, model, loss_fn, optimizer):
    self.model = model
    self.loss_fn = loss_fn
    self.optimizer = optimizer
    self.train_losses = []
    self.val_losses = []

def train_step(self, x, y, lambda1, NH3fluxgN):
    # Sets model to train mode
    self.model.train()

    # Makes predictions
    yhat = self.model(x)

    # Computes loss
    loss =
    lambda1* self.loss_fn(yhat,y) + (1 - lambda1) * self.loss_fn(yhat,NH3fluxgN)

    # Computes gradients
    loss.backward()

    # Updates parameters and zeroes gradients
```
self.optimizer.step()
self.optimizer.zero_grad()

# Returns the loss
return loss.item()

def train(self, batch_size, n_epochs, n_features, lambda1, Diffusionparam,
          KON20, init_ON, pH):
    model_path = f'{self.model}'

    train_loader = torch.utils.data.DataLoader(train, batch_size=batch_size,
                                              shuffle=False, drop_last=True)
    val_loader = torch.utils.data.DataLoader(val, batch_size=batch_size,
                                              shuffle=False, drop_last=True)

    for epoch in range(1, n_epochs + 1):
        batch_losses = []
        for x_batch, y_batch in train_loader:
            x_batch = x_batch.view([batch_size, -1, n_features])
            y_batch = y_batch
            NH3fluxgN = physics_informed_predictions(x_batch, Diffusionparam,
                                                     KON20, init_ON, pH)
            loss = self.train_step(x_batch, y_batch, lambda1, NH3fluxgN)
            batch_losses.append(loss)
        training_loss = np.mean(batch_losses)
        self.train_losses.append(training_loss)
with torch.no_grad():
    batch_val_losses = []
    batch_PBM_losses = []
    for x_val, y_val in val_loader:
        x_val = x_val.view([batch_size, -1, n_features])
        y_val = y_val
        self.model.eval()
        yhat_val = self.model(x_val)
        NH3fluxgN_val = physics_informed_predictions(x_val, Diffusionparam, KON20, init_ON, pH)
        val_loss = self.loss_fn(yhat_val, y_val).item()
        PBM_losses = self.loss_fn(NH3fluxgN_val, y_val).item()
        batch_val_losses.append(val_loss)
        batch_PBM_losses.append(PBM_losses)
    validation_loss = np.mean(batch_val_losses)
    PBM_loss = np.mean(batch_PBM_losses)
    self.val_losses.append(validation_loss)
    if (epoch <= 10) | (epoch % 10 == 0):
        print(f"[\{epoch}/{n_epochs}] Training loss: {training_loss:.4f} \t Validation loss: {validation_loss:.4f} \t Process-based model loss: {PBM_loss:.4f}"
        )
#torch.save(self.model.state_dict(), model_path)
return (validation_loss+PBM_loss)/2
def evaluate(self, test, batch_size, n_features):
    test_loader = torch.utils.data.DataLoader(test, batch_size=batch_size,
                                              shuffle=False, drop_last=True)
    with torch.no_grad():
        predictions = []
        values = []
        for x_test, y_test in test_loader:
            x_test = x_test.view([batch_size, -1, n_features])
            y_test = y_test
            self.model.eval()
            yhat = self.model(x_test)
            predictions.append(yhat.detach().numpy())
            values.append(y_test.detach().numpy())
    return predictions, values

def plot_losses(self, n_epochs):
    plt.figure( figsize = (5,5))
    plt.plot(np.arange(0,n_epochs),self.train_losses, label="Training loss",
             c = 'k')
    plt.plot(self.val_losses, label="Validation loss", c = 'gray')
    plt.legend()
    plt.xlabel('Number of epochs')
    plt.ylabel('scaled loss value')
    plt.title("Losses")
    plt.show()
plt.close()

class Optimization:
    
    Optimization class follows the same steps as Optimization_PI but with using MSE as loss function instead of physics-informed loss
    
    def __init__(self, model, loss_fn, optimizer):
        self.model = model
        self.loss_fn = loss_fn
        self.optimizer = optimizer
        self.train_losses = []
        self.val_losses = []

    def train_step(self, x, y):
        # Sets model to train mode
        self.model.train()

        # Makes predictions
        yhat = self.model(x)

        # Computes loss
        loss = self.loss_fn(y, yhat)

        # Computes gradients
        loss.backward()
# Updates parameters and zeroes gradients
self.optimizer.step()
self.optimizer.zero_grad()

# Returns the loss
return loss.item()

def train(self, batch_size, n_epochs, n_features=1):
    train_loader = torch.utils.data.DataLoader(train, batch_size=batch_size,
                                             shuffle=False, drop_last=True)
    val_loader = torch.utils.data.DataLoader(val, batch_size=batch_size,
                                             shuffle=False, drop_last=True)

    for epoch in range(1, n_epochs + 1):
        batch_losses = []
        for x_batch, y_batch in train_loader:
            x_batch = x_batch.view([batch_size, -1, n_features])
            y_batch = y_batch
            loss = self.train_step(x_batch, y_batch)
            batch_losses.append(loss)
            training_loss = np.mean(batch_losses)
            self.train_losses.append(training_loss)
            with torch.no_grad():
                batch_val_losses = []
for x_val, y_val in val_loader:
    x_val = x_val.view([batch_size, -1, n_features])
    y_val = y_val
    self.model.eval()
    yhat = self.model(x_val)
    val_loss = self.loss_fn(y_val, yhat).item()
    batch_val_losses.append(val_loss)
validation_loss = np.mean(batch_val_losses)
self.val_losses.append(validation_loss)

if (epoch <= 10) | (epoch % 10 == 0):
    print(f"[{epoch}/{n_epochs}] Training loss: {training_loss:.4f} \t Validation loss: {validation_loss:.4f}"
return validation_loss

def evaluate(self, test, batch_size, n_features):
    test_loader = torch.utils.data.DataLoader(test, batch_size=batch_size,
                                              shuffle=False, drop_last=True)
    with torch.no_grad():
        predictions = []
        values = []
        for x_test, y_test in test_loader:
            x_test = x_test.view([batch_size, -1, n_features])

327
y_test = y_test
self.model.eval()
yhat = self.model(x_test)
predictions.append(yhat.detach().numpy())
values.append(y_test.detach().numpy())
return predictions, values

def plot_losses(self, n_epochs):
    plt.figure( figsize = (5,5))
    plt.rcParams["font.family"] = "Arial"
    plt.plot(np.arange(0,n_epochs),self.train_losses, label="Training loss", c = 'k')
    plt.plot(self.val_losses, label="Validation loss", c = 'gray')
    plt.legend()
    plt.xlabel('Number of epochs', fontsize = 15 )
    plt.ylabel('Scaled loss value', fontsize = 15)
    plt.title("Losses")
    plt.show()
    plt.close()

class LSTMModel(nn.Module):
    def __init__(self, input_dim, hidden_dim, layer_dim, output_dim, dropout_prob):
        super(LSTMModel, self).__init__()
        # Defining the number of layers and the nodes in each layer
        self.hidden_dim = hidden_dim
        self.layer_dim = layer_dim
# LSTM layers
self.lstm = nn.LSTM(
    input_dim, hidden_dim, layer_dim, batch_first=True)

# Dropout layer
#self.dropout = nn.Dropout(p=dropout_prob)

# Fully connected layer
self.fc = nn.Linear(hidden_dim, output_dim)

def forward(self, x):
    # Initializing hidden state for first input with zeros
    h0 = torch.zeros(self.layer_dim, x.size(0), self.hidden_dim).requires_grad_()

    # Initializing cell state for first input with zeros
    c0 = torch.zeros(self.layer_dim, x.size(0), self.hidden_dim).requires_grad_()

    # We need to detach as we are doing truncated backpropagation through time
    # If we don't, we'll backprop all the way to the start even after going through
    # Forward propagation by passing in the input, hidden state, and cell state
    # into the model out, (hn, cn) = self.lstm(x, (h0.detach(), c0.detach()))

    #out = self.dropout(out)
# Reshaping the outputs in the shape of (batch_size, seq_length, hidden_size)
# so that it can fit into the fully connected layer
out = out[:, -1, :]

# Convert the final state to our desired output shape (batch_size, output_dim)
out = self.fc(out)

return out

# Obtain hyperparameters for this trial
def suggest_hyperparameters(trial):
    # Neural Network Hyperparameters
    # Obtain the batch size
    batch_size = trial.suggest_int("batch_size", 32, 128, step=8)
    # Obtain the learning rate on a logarithmic scale
    lr = trial.suggest_float("lr", 1e-4, 1e-1, log=True)
    # Obtain the dropout ratio in a range from 0.0 to 0.9 with step size 0.1
    dropout = trial.suggest_float("dropout", 0.0, 0.5, step=0.1)
    # Obtain number of hidden neurons in each layer
    hidden_dim = trial.suggest_int("hidden_dim", 8, 64, step=8)
    # Obtain number of hidden layers
    #layer_dim = trial.suggest_int("layer_dim", 1, 3, step=1)
    # Obtain number of hidden layers in the linear layer
    #hidden1 = trial.suggest_int("hidden1", 1, 4, step=8)

    #================================================================
    # Process based model hyperparameters
lambda1 = trial.suggest_float("lambda1", 0.5, 0.9, step=0.1)
Diffusionparam = trial.suggest_float("Diffusionparam", 1.5, 3, step=0.5)
KON20 = trial.suggest_float("KON20", 0.006, 0.06, step=0.002)
init_ON = trial.suggest_float("initial ON", 1, 3.5, step=0.5)
pH = trial.suggest_float("pH", 6.5, 7.5, step=0.5)

print(f"Suggested hyperparameters: \n{pformat(trial.params)}")
return batch_size, lr, dropout, hidden_dim, lambda1, Diffusionparam,
        KON20, init_ON, pH

def inverse_transform(scaler, df, columns):
    """
    To transform the scaled data back to its original value
    """
    for col in columns:
        df[col] = scaler.inverse_transform(df[col])
    return df

def format_predictions(predictions, values, df_test, scaler):
    """
    Formatting the predicted and target outputs to calculate
    the model performance metrics
    """
    vals = np.concatenate(values, axis=0).ravel()
preds = np.concatenate(predictions, axis=0).ravel()
df_result = pd.DataFrame(data={"value": vals, "prediction": preds}, index=df_test.head(len(vals)).index)
df_result = df_result.sort_index()
df_result = inverse_transform(scaler, df_result, [["value", "prediction"]])
return df_result

def calculate_metrics(df):
    return {'mae' : mean_absolute_error(df.value, df.prediction),
            'rmse' : mean_squared_error(df.value, df.prediction) ** 0.5,
            'r2' : r2_score(df.value, df.prediction)}

def objective(trial):
    """
    This function connects the other functions, it retrieves the
    values of the hyperparameters and use them for train the
    model
    """
    batch_size, lr, dropout, hidden_dim, lambda1, Diffusionparam, \
            KON20, init_ON = suggest_hyperparameters(trial)
    input_dim = len(X_train.columns)
    output_dim = 1
    hidden_dim = hidden_dim
    layer_dim = 1
    batch_size = batch_size
    dropout = dropout

    # define the number of epochs (typically 500)
model = LSTMModel(input_dim, hidden_dim, layer_dim, output_dim, dropout)
loss_fn = nn.MSELoss(reduction="mean")
optimizer = optim.Adam(model.parameters(), lr=lr)

#to run the model with MSE as loss function
opt = Optimization(model=model, loss_fn=loss_fn, optimizer=optimizer)
val_mse = opt.train(batch_size=batch_size, n_epochs=n_epochs,
n_features=input_dim)

#to run the model with the physics-informed loss function
opt = Optimization_PI(model=model, loss_fn=loss_fn, optimizer=optimizer)
val_mse = opt.train(batch_size=batch_size, n_epochs=n_epochs,
n_features=input_dim, lambda1 = lambda1, Diffusionparam = Diffusionparam,
KON20 = KON20, init_ON = init_ON)

opt.plot_losses(n_epochs)
predictions, values = opt.evaluate(test, batch_size=1, n_features=input_dim)
df_result = format_predictions(predictions, values, X_test, scaler_target)
result_metrics = calculate_metrics(df_result)
print(f'results metrics {result_metrics}')
plt.plot(df_result.index, df_result['value'], label = 'Actual NH3 flux')
plt.plot(df_result.index, df_result['prediction'], label = 'Predicted NH3 flux')
plt.legend()
plt.show()
return val_mse

eval(open("PBM_functions.py").read())

if __name__ == '__main__':
    n_epochs = 30
    study_name = 'study0330_100tial_30epochs'
    study = optuna.create_study(study_name=study_name,
                                storage='sqlite:///example.db',direction="minimize")
    val_mse = study.optimize(objective, n_trials=100)

    pruned_trials = [t for t in study.trials if t.state ==
                     optuna.trial.TrialState.PRUNED]
    complete_trials = [t for t in study.trials if t.state ==
                       optuna.trial.TrialState.COMPLETE]

    print("Study statistics: ")
    print(" Number of finished trials: ", len(study.trials))
    print(" Number of pruned trials: ", len(pruned_trials))
    print(" Number of complete trials: ", len(complete_trials))

    print("Best trial:")
    trial = study.best_trial

    print(" Value: ", trial.value)

    print(" Params: ")
    for key, value in trial.params.items():
        print(" {}: {}".format(key, value))
trial.report(val_mse, n_epochs)

# Handle pruning based on the intermediate value.
if trial.should_prune():
    raise optuna.exceptions.TrialPruned()

loaded_study = optuna.load_study(study_name="study0330_100tial_30epochs",
        storage="sqlite:///example.db")

fig = optuna.visualization.plot_slice(loaded_study)
fig.update_layout(plot_bgcolor='white')
fig.update_xaxes(
    mirror=True,
    ticks='outside',
    showline=True,
    linecolor='black',
    color = 'black',
    gridcolor='lightgrey'
)

fig.update_yaxes(
    mirror=True,
    ticks='outside',
    showline=True,
    linecolor='black',
    color = 'black',
    gridcolor='lightgrey'
)

fig.update_traces(marker=dict(color="black"))
Developing the model using the optimum hyperparameters

```python
def objective(trial):
    
    """
    This function connects the other functions, it retrieves the
    values of the hyperparameters and use them for train the
    model
    """
    hidden_dim = trial.params['hidden_dim']
    lr = trial.params['lr']
    batch_size = trial.params['batch_size']
    dropout = trial.params['dropout']
    Diffusionparam = trial.params['Diffusionparam']
    KON20 = trial.params['KON20']
    lambda1 = trial.params['lambda1']
    init_ON = trial.params['init_ON']
    pH = trial.params['pH']

    input_dim = len(X_train.columns)
    output_dim = 1
    layer_dim = 1

    # define the number of epochs (typically 500)

    model = LSTMMModel(input_dim, hidden_dim, layer_dim, output_dim, dropout)
```
loss_fn = nn.MSELoss(reduction="mean")
optimizer = optim.Adam(model.parameters(), lr=lr)

""
#to run the model with MSE as loss function
opt = Optimization(model=model, loss_fn=loss_fn, optimizer=optimizer)
val_mse = opt.train(batch_size=batch_size, n_epochs=n_epochs,
                     n_features=input_dim)
""

#to run the model with the physics-informed loss function
opt = Optimization_PI(model=model, loss_fn=loss_fn, optimizer=optimizer)
val_mse = opt.train(batch_size=batch_size, n_epochs=n_epochs,
                     n_features=input_dim, lambda1 = lambda1, Diffusionparam = Diffusionparam,
                     KON20 = KON20, init_ON = init_ON, pH = pH)

opt.plot_losses(n_epochs)

#predictions for the test dataset from IN
predictions, values = opt.evaluate(test, batch_size=1, n_features=input_dim)
df_result = format_predictions(predictions, values, X_test, scaler_target)
result_metrics = calculate_metrics(df_result)
print(f'results metrics {result_metrics}')
plt.figure(figsize=(14,6))
plt.rcParams['font.family'] = "Arial"
plt.plot(df_result.index, df_result['value']/1000,
         label = 'Actual ammonia flux', c = 'gray')
plt.plot(df_result.index, df_result['prediction']/1000,
```python
plt.legend()
plt.xlabel("Date", fontsize = 15)
plt.ylabel("Ammonia \( g/m^2/d \)", fontsize = 15)
plt.xticks(fontsize =15)
plt.yticks(fontsize =15)
plt.savefig("PIRNN_HPT.eps", dpi=1200)
plt.show()

#predictions for the test dataset from Kupper
predictions_K, values_K = opt.evaluate(test_Kupper, batch_size=1,
                                       n_features=input_dim)
df_result_K = format_predictions(predictions_K, values_K, X_Kupper, scaler_target)
result_metrics_K = calculate_metrics(df_result_K)
print(f'results metrics for kupper testing {result_metrics_K}')
plt.figure(figsize=(12,6))
plt.rcParams["font.family"] = "Arial"
plt.plot(df_result_K.index, df_result_K['value']/1000,
         label = 'Actual ammonia flux', c = 'gray')
plt.plot(df_result_K.index, df_result_K['prediction']/1000,
         label = 'Predicted ammonia flux', c = 'k')
plt.legend()
plt.xlabel("Date", fontsize = 15)
plt.ylabel("Ammonia \( g/m^2/d \)", fontsize = 15)
plt.xticks(fontsize =15)
plt.yticks(fontsize =15)
```
plt.savefig("PIRNN_HPT_Kupper.eps", dpi=1200)
plt.show()

return val_mse

exec(open("PBM_functions.py").read())

if __name__ == "__main__":
    n_epochs = 100
    study = optuna.create_study(direction="minimize")
    val_mse = study.optimize(objective, n_trials=1)

    pruned_trials = [t for t in study.trials if t.state ==
                     optuna.trial.TrialState.PRUNED]
    complete_trials = [t for t in study.trials if t.state ==
                       optuna.trial.TrialState.COMPLETE]

    print("Study statistics: ")
    print(" Number of finished trials: ", len(study.trials))
    print(" Number of pruned trials: ", len(pruned_trials))
    print(" Number of complete trials: ", len(complete_trials))

    print("Best trial:")
    trial = study.best_trial

    print(" Value: ", trial.value)

    print(" Params: ")
for key, value in trial.params.items():
    print("{}: ").format(key, value)

trial.report(val_mse, n_epochs)

# Handle pruning based on the intermediate value.
if trial.should_prune():
    raise optuna.exceptions.TrialPruned()

[1 2023-04-07 08:19:20,075] A new study created in memory with name: no-name-00
[1/100] Training loss: 0.7726 Validation loss: 0.6947 Process-based model loss:0.9976
[2/100] Training loss: 0.7502 Validation loss: 0.6828 Process-based model loss:0.9976
[3/100] Training loss: 0.7433 Validation loss: 0.6824 Process-based model loss:0.9976
[4/100] Training loss: 0.7345 Validation loss: 0.6769 Process-based model loss:0.9976
[5/100] Training loss: 0.7249 Validation loss: 0.6672 Process-based model loss:0.9976
[6/100] Training loss: 0.7180 Validation loss: 0.6614 Process-based model loss:0.9976
[7/100] Training loss: 0.7126 Validation loss: 0.6571 Process-based model loss:0.9976
[8/100] Training loss: 0.7067 Validation loss: 0.6525 Process-based model loss:0.9976
[9/100] Training loss: 0.7009 Validation loss: 0.6494 Process-based model loss:0.9976
[10/100] Training loss: 0.6937 Validation loss: 0.6396 Process-based model loss:0.9976
[20/100] Training loss: 0.6562 Validation loss: 0.6106 Process-based model loss:0.9976
[30/100] Training loss: 0.6324 Validation loss: 0.5950 Process-based model loss:0.9976
[40/100] Training loss: 0.6155 Validation loss: 0.5883 Process-based model loss:0.9976
[50/100] Training loss: 0.6025 Validation loss: 0.5825 Process-based model loss:0.9976
[60/100] Training loss: 0.5876 Validation loss: 0.5840 Process-based model loss:0.9976
[70/100] Training loss: 0.5746 Validation loss: 0.5851 Process-based model loss:0.9976
[80/100] Training loss: 0.5633 Validation loss: 0.5873 Process-based model loss:0.9976
[90/100] Training loss: 0.5548 Validation loss: 0.5932 Process-based model loss:0.9976

340
[100/100] Training loss: 0.5462  Validation loss: 0.5968  Process-based model loss: 0.9976
Trial 0 finished with value: 0.7971918197969596 and parameters: {}. Best is trial 0 with value: 0.7971918197969596.

Study statistics:

Number of finished trials: 1
Number of pruned trials: 0
Number of complete trials: 1

Best trial:

Value: 0.7971918197969596

Params:

exec(open("PBM_functions.py").read())

if __name__ == "__main__":
    n_epochs = 50
    study = optuna.create_study(direction="minimize")
    val_mse = study.optimize(objective, n_trials=1)

    pruned_trials = [t for t in study.trials if t.state ==
optuna.trial.TrialState.PRUNED]
complete_trials = [t for t in study.trials if t.state ==
    optuna.trial.TrialState.COMPLETE]

print("Study statistics: ")
print(" Number of finished trials: ", len(study.trials))
print(" Number of pruned trials: ", len(pruned_trials))
print(" Number of complete trials: ", len(complete_trials))

print("Best trial:")
trial = study.best_trial

print(" Value: ", trial.value)

print(" Params: ")
for key, value in trial.params.items():
    print(" {}: {}".format(key, value))
    trial.report(val_mse, n_epochs)

# Handle pruning based on the intermediate value.
if trial.should_prune():
    raise optuna.exceptions.TrialPruned()

[I 2023-04-07 15:15:40,114] A new study created in memory with name: no-name-00
[1/50] Training loss: 0.7730  Validation loss: 0.6983  Process-based model loss:0.9976
[2/50] Training loss: 0.7492  Validation loss: 0.6887  Process-based model loss:0.9976
[3/50] Training loss: 0.7426  Validation loss: 0.6841  Process-based model loss:0.9976
[4/50] Training loss: 0.7321  Validation loss: 0.6778  Process-based model loss: 0.9976
[5/50] Training loss: 0.7225  Validation loss: 0.6693  Process-based model loss: 0.9976
[6/50] Training loss: 0.7156  Validation loss: 0.6629  Process-based model loss: 0.9976
[7/50] Training loss: 0.7107  Validation loss: 0.6569  Process-based model loss: 0.9976
[8/50] Training loss: 0.7051  Validation loss: 0.6493  Process-based model loss: 0.9976
[9/50] Training loss: 0.6999  Validation loss: 0.6437  Process-based model loss: 0.9976
[10/50] Training loss: 0.6947  Validation loss: 0.6375  Process-based model loss: 0.9976
[20/50] Training loss: 0.6530  Validation loss: 0.6061  Process-based model loss: 0.9976
[30/50] Training loss: 0.6258  Validation loss: 0.6019  Process-based model loss: 0.9976
[40/50] Training loss: 0.6043  Validation loss: 0.6000  Process-based model loss: 0.9976
[50/50] Training loss: 0.5899  Validation loss: 0.5974  Process-based model loss: 0.9976

![Losses graph](image-url)
The PostScript backend does not support transparency; partially transparent artists will be rendered opaque.