Utilizing Recurrent Neural Networks for Temporal Data Generation and Prediction

ThaoVy Nguyen

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

Master of Science
in
Mathematics

Russell J. Hewett, Chair
Serkan Gugercin
Eileen R. Martin

May 12, 2021
Blacksburg, Virginia

Keywords: Neural Networks, LatentGAN, Falling Creek Reservoir

Utilizing Recurrent Neural Networks for Temporal Data Generation and Prediction

ThaoVy Nguyen

(ABSTRACT)

The Falling Creek Reservoir (FCR) in Roanoke is monitored for water quality and other key measurements to distribute clean and safe water to the community. Forecasting these measurements is critical for management of the FCR. However, current techniques are limited by inherent Gaussian linearity assumptions. Since the dynamics of the ecosystem may be nonlinear, we propose neural network-based schemes for forecasting. We create the LatentGAN architecture by extending the recurrent neural network-based ProbCast and autoencoder forecasting architectures to produce multiple forecasts for a single time series. Suites of forecasts allow for calculation of confidence intervals for long-term prediction. This work analyzes and compares LatentGAN's accuracy for two case studies with state-of-the-art neural network forecasting methods. LatentGAN performs similarly with these methods and exhibits promising recursive results.

Utilizing Recurrent Neural Networks for Temporal Data Generation and Prediction

ThaoVy Nguyen

(GENERAL AUDIENCE ABSTRACT)

The Falling Creek Reservoir (FCR) is monitored for water quality and other key measurements to ensure distribution of clean and safe water to the community. Forecasting these measurements is critical for management of the FCR and can serve as indicators of significant ecological events that can greatly reduce water quality. Current predictive techniques are limited due to inherent linear assumptions. Thus, this work introduces LatentGAN, a data-driven, generative, predictive neural network. For a particular sequence of data, LatentGAN is able to generate a suite of possible predictions at the next time step. This work compares LatentGAN's predictive capabilities with existing neural network predictive models. LatentGAN performs similarly with these methods and exhibits promising recursive results.

Acknowledgments

I would like to thank my advisor, Dr. Russell J. Hewett for mentoring me throughout this process. Your support for my research and in the next steps in my career are greatly appreciated.

I'd also like to acknowledge Dr. Cayelan Carey and Dr. Quinn Thomas for making this work possible. The open source datasets, FLARE, and GLM code allowed my work to analyze the FCR case study.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Predictive modeling is prevalent in many fields including ecology [8], dynamical systems [9, 10, 11, 12, 13, 14], and economics [15]. Small changes in predictions can lead to vastly different outcomes for decision makers [16, 17]. Even the most accurate process-driven models cannot exactly predict the physical world. However, data-based models have limitations of their own. Although observed data is representative of the world, accuracy, precision, and availability of instrumentation can decrease predictive accuracy.

In this study, we evaluate recent advances in nonlinear, data-driven forecasting models for ecological drivers in a freshwater reservoir. Current state-of-the-art lake and reservoir prediction models are limited by linear assumptions, however, relevant ecological processes have nonlinear relationships [8, 18, 19].

This work introduces the LatentGAN framework, a data-driven, generative, predictive recurrent neural network (RNN) model. This model combines the RNN-based autoencoder [4, 5] and RNN-based Generative Adversarial Network (GAN) [2, 6] architectures to generate suites of predictions given a single time series. LatentGAN relies on the prediction capabilities of RNNs [3, 20, 21], the ability to represent data in a reduced-dimensional space [1], and the generative capability of GANs [2, 7]. The short and long-term predictive capability of LatentGAN is analyzed through comparison of current predictive models and two specific case-studies.

1.1 Objectives

Many state-of-the-art models, such as the Ensemble Kalman Filter (EnKF), assume Gaussian statistics [8, 19]. Embedding inherent linear assumptions into models limits their predictive accuracy. Often, models, including the EnKF, are trained as single step predictors [4, 7, 8, 20, 21, 22]. These recursive models are able to take previous predictions as input and predict arbitrary time steps in advance [20]. As fewer measured data are used in recursive prediction, we expect uncertainty to increase. This property is observed in current EnKF methods and is expected to be retained in the LatentGAN model [8]. These shortfalls and the nonlinearity of common ecological and physical problems motivates us to consider a neural network (NN)-based approach.

LatentGAN is a data-driven NN architecture that aims to generate a suite of possible single step predictions given a time sequence of data. The architecture consists of a joint GAN and autoencoder. In particular, LatentGAN draws upon the generative architecture of the GAN to generate these suites of predictions as a nonlinear mapping [2]. The autoencoder depends upon the hypothesis of a reduced-dimension representation of data [1]. This allows LatentGAN to minimize data overfitting as well as recognize or exploit hidden correlations within the data [1]. The GAN is able to produce forecasts within the existing data distribution [2]. Although GANs were introduced in the generation of images and have only recently been studied in forecasting [2, 7, 23, 24], the generative capability of GANs have been shown to successfully perform image-to-image translation [25], generate music [26], and deblur images [27]. This work aims to further analyze the ability of GANs and autoencoders to be utilized as forecasting models in ecology. Ultimately, we aim to implement LatentGAN as both a process and data-driven model where process drivers are embedded through existing process-driven models, such as the General Lake Model (GLM) [18]. This study represents

the first steps in achieving a data and process-driven NN architecture for generation of time series predictions.

1.2 Falling Creek Reservoir

In this work, we consider a case study of the Falling Creek Reservoir (FCR) [8]. The FCR is a freshwater reservoir in Roanoke, VA and provides clean, safe water to consumers in the area [8]. The reservoir is the subject of a long-term study, and high-frequency data is collected from a single water column in the center of the reservoir [8, 28, 29]. This water column contains a suite of instruments set at depths spaced one meter apart [8]. Additional data are also being taken relevant to the FCR, including stream in and outflow data [30] and meteorlogical data [31, 32, 33].

Predicting key properties can inform lead scientists of significant upcoming events with respect to lake biology and chemistry [8]. For example, fall turnover is a specific ecological event that occurs when the temperature at the bottom of the reservoir is equivalent to the temperature at the top of the reservoir [8]. This causes sediment circulation and lowers water quality [8]. Accurate forecasts of these events allow reservoir managers to take action, such as modifying chemical treatments, to maintain water quality [8]. For example, it is known that algal blooms reduce water quality [8]. Although it is unknown what measurements may serve as predictive indicators for these blooms, NNs may be able to recognize implicit relationships within data that gives warning to these events.

1.2.1 Data

There are many data sets available regarding the FCR. These data are processed by Dr. Cayelan Carey and her lab at Virginia Tech. The most complete data set is taken at the water column in the center of the reservoir [28]. Current public data contains a high frequency data set of measurements for temperature, oxygen concentration, chlorophyll, blue-green algae phycoycanin, and fine dissolved organic matter at ten depths [28]. Measurements are recorded every ten minutes and data from July 6, 2018 to January 1, 2021 is publicly available [28]. The second data set is the weekly water column data set that consists of data during the non-winter months of the water column since 2013 [29]. Instruments, similar to that of the high-frequency water column data set, are gradually lowered through the water column and continuously record data [29]. The third data set is the meteorological data set with measurements taken every minute [32]. Finally, a sparse meteorlogical data set contains weekly measurements of various meteorological and ecological data [31, 33]. The usage of this abundant data is obstructed by the various frequencies of each data set. In this study, we focus on the first, high-frequency water column data set.

1.3 Thesis Organization

Existing literature is reviewed in Chapter 2. This review contains five main sections: time series forecasting, forecasting lake and reservoir dynamics, neural networks, neural networks for time series forecasting, and uncertainty in neural networks. The LatentGAN architecture and training phases are introduced in Chapter 3, and two case studies comparing existing NN forecasters and LatentGAN are presented in Chapter 4. Future extensions and applications of LatentGAN are included in the discussion of Chapter 5.

Chapter 2

Background and Literature Review

2.1 Time Series Forecasting

A time series is a sequence of data $\boldsymbol{x} = \{x_1, ..., x_T\}$ where $x_t \in \mathbb{R}^n$ for $t \in 1, ..., T$. Multiple *univariate* strategies for time series forecasting are defined with the goal to predict $\{x_{T+1}, x_{T+2}, ..., x_{T+h}\}$ given $\{x_1, x_2, ..., x_T\}$, that is to forecast h steps in advance [20, 21, 34, 35]. Specifically, univariate methods only utilize previous and present data to predict future data [35]. The simple *single-step strategy* specifies $h = 1$ in the general time series forecasting method [20, 21, 34]. In practice, the single-step strategy is often used [4, 7, 20, 21, 34].

The *direct strategy* consists of learning h models to predict each subsequent time step separately [34]. Thus, the i^{th} model predicts the data at step $T + i$ given $\boldsymbol{x}$, where $i = 1, ..., h$ [34]. The *multi-step strategy* requires a single model that predicts all h forecasts with a single prediction. The *recursive strategy* trains a single model that predicts a single step ahead and recursively takes the output as input to predict the next time step [34]. By applying this process h times, the model is able to predict h steps in advance.

Taieb et al. concludes that both the recursive and direct strategies perform similarly, with the recursive strategy slightly outperforming the direct strategy for the particular NN5 forecasting dataset [34]. In this work, we consider the recursive strategy because this strategy requires a single model and allows for the propagation of uncertainty with time. The single

model reduces computational training time. Additionally, the recursive strategy allows for arbitrary h, rather than a fixed h in the direct and multi-step strategies.

All four of the mentioned strategies can also be utilized as *multivariate* methods [35]. Multivariate methods consist of predicting future data given previous and current data, as well as additional data, or conditions, that may be relevant. Conditions can take the form of time of day, classifications, or any information that may inform the model. In this work, conditions will take the form of time of day, day of year, and relevant data transformations.

Predictive models can be described as process-driven, data-driven, or process- and data-driven. A process-driven model is one that only takes into account known relationships between the inputs and outputs of the model. For example, known differential equations can describe particular systems without observed data. Data-driven models are those that rely solely on observed data for prediction. Thus, the accuracy of this type of model depends on the accuracy of the observed data itself. Process- and data-driven models are able to combine both known processes along with observed data to produce informed predictions. In this work, we focus on data-driven models.

2.2 Forecasting Lake and Reservoir Dynamics

2.2.1 General Lake Model

The lake and reservoir ecosystem modeling community often relies on 1-D temperature modeling due to its known relationships with various complex ecological simulations [18]. The specific 1-D temperature model used by existing state-of-the-art lake and reservoir predictive models is the General Lake Model (GLM) [8]. The GLM is a process model that takes various meteorological data and the current center water column of temperatures as input,

and outputs the next day's water column of temperatures. The meteorological data can be observed *in situ* or from one of twenty-one National Oceanic and Atmospheric Association (NOAA) atmospheric models [8]. The GLM takes into account many ecological aspects of lakes and reservoirs, including solar radiation, inflow and outflow boundary conditions, and groundwater seepage [18]. Despite many factors being accounted for, the exact physical relationships between these inputs and the 1-D temperature output is not known. Ecological processes bridge both chemistry and biology. Thus, many processes within ecology are not known or are difficult to represent. Therefore, the GLM serves as a process estimate of the next day's temperature profile. The model utilizes a Lagrangian approach that causes the output measurement predictions to differ in depth from the input depths [18]. The Lagrangian approach allows for the defined layers in the water column to change in thickness due to ecological forces [18]. Because of this, a modified version of the GLM that keeps input measurement depths the same as the output measurement depths is considered key [8].

A shortfall of the GLM is its prediction frequency. The model only predicts the next day at midnight, however specific events, such as fall turnover, may occur at any time of day. Many bodies of water, including the FCR, are monitored at a much higher frequency and this abundance of data may bring specific relationships to the forefront.

2.2.2 Forecasting Lake and Reservoir Ecosystems

The current state-of-the-art in lake and reservoir forecasting is the Forecasting Lake and Reservoir Ecosystems (FLARE) model [8]. FLARE is an Ensemble Kalman Filter (EnKF) based approach [8]. The EnKF is a recursive, predictive model that allows for data assimilation and provides informed updates to the standard state space model [19]. The standard

state space model is given by

$$x_t = F_t x_{t-1} + w_t, \tag{2.1}$$

$$y_t = H_t x_t + v_t, \tag{2.2}$$

where x_t is the state vector at time t, y_t is the observation vector at time t, F_t is the state transition matrix at time t, H_t is the observation matrix at time t, w_t is the process noise vector in $\mathcal{N}(0, R)$ where R is the covariance of the process noise, and v_t is the observation noise vector in $\mathcal{N}(0, Q)$ where Q is the covariance of the observation noise [19, 36]. In the EnKF, an ensemble of k estimates of x_t are tracked. The statistics of this ensemble of predictions allows the EnKF method to yield confidence intervals and uncertainty quantifications for each prediction. In FLARE, x_t is length 29 while y_t is length 10 since the GLM outputs predictions at 29 different depths although the FCR only takes measurements at 10 depths.

EnKFs have the ability to perform data assimilation and improve the accuracy of the model with the availability of additional data [8]. By taking the GLM output as the state vector as the input to FLARE, the model is able to enhance the GLM's predictive capabilities [8]. In particular, FLARE uses the twenty-one NOAA atmospheric models to induce an ensemble of 21 observations and thus, statistics are estimated on the twenty-one member ensemble and a single prediction is given along with a confidence interval [8]. To assess FLARE's predictive capability, the null model is introduced as the predictive model that assumes no state or observation changes with time [8]. FLARE performs well and can predict fall turnover with reasonable accuracy compared to this null model [8]. As the FLARE model is run consecutively off of predicted values, uncertainty can be measured in time.

Although this model performs reasonably, the inherent linear assumptions of Kalman Filters (KFs) may prohibit the model from learning the underlying process. Additionally, KFs assume Gaussian statistics in the assumption of Gaussian noise, however this may not be

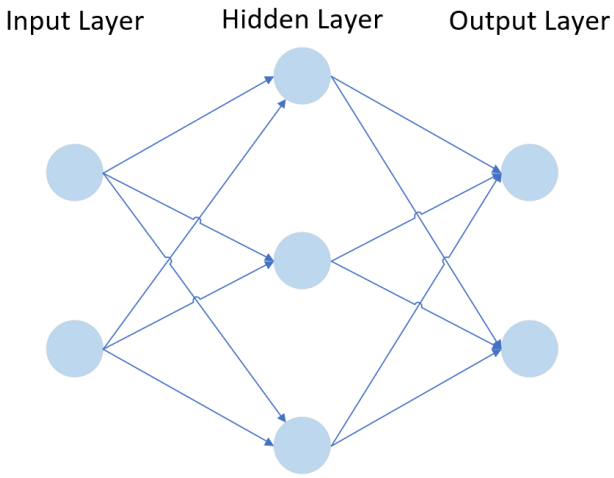


Figure 2.1: A simple neural network architecture.

the case for this ecological process. While nonlinear extensions to the KF exist [37], they are not applied in the FLARE model.

2.3 Neural Networks

We introduce the use of neural networks (NNs) to remove inherent linear and Gaussian assumptions in forecasting. NNs are the composition of nonlinear, activation functions, which take the form of sigmoids, tanh, or ReLU functions [38], along with linear mappings of the form [1, 3]

$$f(x) = Wx + b, \tag{2.3}$$

where W is a matrix mapping the input vector, $x \in \mathbb{R}^n$, from $\mathbb{R}^n$ to $\mathbb{R}^m$, and $b \in \mathbb{R}^m$ is an additional bias vector. The activation functions allow the network to observe nonlinear relationships within the data that would not appear with linear operators. An arbitrary k-layer neural network can be written in the form

$$N(x) = \sigma_k(...\sigma_2(f_2(\sigma_1(f_1(x))))...), \tag{2.4}$$

where f_i are linear mappings of the form given in equation (2.3) and σ_i are activation functions for the i^{th} layer. A simple NN architecture is depicted in Figure 2.1. The model contains 2 layers: the first takes data of dimension 2 as input and outputs data of dimension 3, the second takes input of dimension 3 and outputs data of dimension 2.

Neural networks can be *trained* to recognize relationships within the data, a process that requires large data sets. In practice, the network architecture, or sequence of linear mappings and activation functions, is customized for specific problems and data sets. These architectures are custom engineered depending on the problem at hand.

These architectures can be used on different types of problems, ranging from classification to data generation. A regression model seeks to map a single or multiple independent variables to a single dependent variable [3]. A classification model is a regression model in which the dependent variable is discrete [3]. To train NNs for regression or classification, input and output pairs of data are required [1, 3]. The training process aims to minimize the output of the model evaluated on the input data compared to the expected output [1, 3]. For example, a simple loss function for a NN regressor is given by

$$L_{NN} = ||y - N(x)||_2^2, \tag{2.5}$$

where N is the arbitrary NN, x is the input data, and y is the corresponding expected output data [1, 3]. Through training techniques, the weight matrix W and bias vector b in each function f_i, are learned. Training involves evaluation of a given loss function of a subset, or batch, of the training data set, computation of gradients through given optimizers, and adjusting W and b according to a the computed gradients and given learning rate [1, 3]. Training is completed after the evaluation of the loss function approaches an equilibrium. The number of times training is completed over the entire data set is denoted as the number

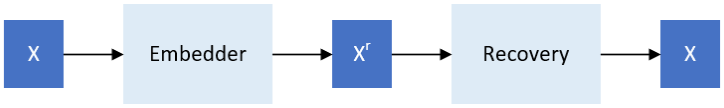


Figure 2.2: The autoencoder architecture [1].

of epochs the model is trained on [3]. After training is completed, NNs can be evaluated quickly.

2.3.1 Autoencoders

The autoencoder is a NN architecture aiming to represent data in a reduced-dimension space and consists of an embedder and a recovery NN [1, 4]. The embedder NN is a network that maps data into a reduced-dimension, different from the existing data space, $E : X \to H$, while the recovery NN maps data from the latent space back to the data space, $R : H \to X$ [1, 4]. The workflow of a basic autoencoder is shown in Figure 2.2. This architecture performs optimally when there exists a reduced-dimension such that the data space is more accurately represented [1]. The output of the autoencoder, $N(x) = R(E(x))$, is expected to be the same as the data itself. Therefore, the autoencoder loss function for training can be described by

$$L_{ER} = ||x - R(E(x))||_2^2, \tag{2.6}$$

where $x \in X$. The size of the reduced-dimension can be chosen with nested cross-validation, a common technique for searching hyperparameter spaces [21, 34, 39, 40, 41]. However, autoencoders can encounter problems when this latent dimension is chosen incorrectly [1]. If the latent dimension is larger than the data dimension, it is possible for the autoencoder to simply copy the input data to the output data without learning underlying relationships within the data itself [1]. If the latent dimension is too small, the network may not not

be able to learn meaningful relationships between data. Nested cross-validation reduces the probability of either occurrences, however it can be costly when searching large hyperparameter spaces.

Reduced Order Models and Autoencoders

Time series prediction has been studied in the dynamical systems and model reduction field [9, 10, 11, 12, 13, 14, 42, 43, 44]. A full order model (FOM) dynamical system is a system of differential equations dependent on an independent variable, t [9, 10, 11, 12, 13, 14, 42, 43, 44]. While solving FOMs can be computationally costly, reduced order models (ROMs) can increase efficiency given a calculable error [9, 10, 11, 12, 13, 14, 42, 43, 44]. A ROM is a reduced-dimension representation of a corresponding FOM. Common linear methods for forming ROMs include utilization of Singular Value Decomposition (SVD), Hankel singular values, and balanced truncation [9, 10, 11, 12, 13, 14, 42, 43, 44].

However, rather than utilizing linear transformations to map data to the reduced space, recent research has shown that utilizing an autoencoder gives better accuracy [42]. This allows the dynamical system to be solved in a reduced space along with a nonlinear mapping to the true data space [42]. Thus, the solution to a given FOM can be solved as a nonlinear model rather than a linear model. Although the technique of using an autoencoder to map a FOM to its corresponding ROM exhibits promising results, this method requires explicit differential equations governing the given system [42]. As mentioned in Section 2.2.1, many ecological processes are unknown or difficult to represent. Thus, utilizing traditional ROM and dynamical system techniques for time series data prediction will not be used in this work.

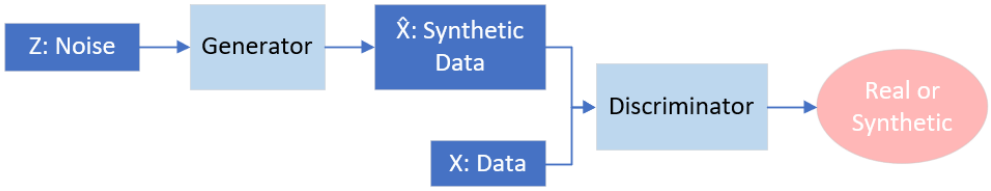


Figure 2.3: The GAN workflow [2].

2.3.2 Generative Adversarial Networks

Due to the requirement of large data sets for training, neural networks can have limitations in the absence of data. For example, lack of enough data can often cause overfitting. Training on synthetic data and testing on real data is common practice and can be useful when there are pre-existing data sets relevant to the problem at hand [6, 40]. In the presence of small data with no similar synthetic data sets, traditional neural network training is prone to overfitting. However, the generative adversarial network (GAN) architecture aims to create large synthetic data sets from small data sets [2]. This architecture consists of two NNs: the generator and the discriminator [2]. The generator, $G : Z \to X$, is a NN that takes random noise from a specific distribution, for example sampled from the normal distribution, to the data space, X [2]. The discriminator, $D : X \to [0, 1]$, is a NN that maps data to a value between 0 and 1 [2]. 0 indicates the presence of synthetic data, and 1 real data. The GAN workflow is depicted in Figure 2.3. A properly trained generator is able to sample from the distribution of the data set [2]. Therefore, the loss function for the GAN architecture becomes

$$L_{GAN} = \mathbb{E}_{x\sim p_{\text{data}}(x)}[\log(D(x))] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))], \tag{2.7}$$

where $p_z(z)$ is the distribution of noise [2]. The discriminator aims to minimize loss function (2.7), whereas the generator aims to maximize loss function (2.7) [2]. By minimizing the loss function, the discriminator correctly classifies real data as 1 in the first term of (2.7), and classifies synthetic data as 0 in the second term. By maximizing the loss function, the

generator, only affecting the second term of (2.7), aims for the discriminator to incorrectly classify generated data as real. Training is completed when the discriminator converges and can no longer differentiate between real and generated data.

Although the GAN architecture introduces a new method for fields with a lack of data, the architecture has shown common problems. When training a GAN, the generator may produce a single output that the discriminator cannot distinguish as synthetic, a phenomenon called *mode collapse* [2]. Mode collapse occurs when the generator collapses the input of noise distribution to a single, or multiple modes [2]. The loss function reaches a local minimum and does not map to the desired distribution of data. Mode collapse is difficult to overcome, but is often mitigated with more data or a smaller learning rate for the discriminator [45].

Another common shortfall occurs when the discriminator trains significantly faster than the generator, causing the generator to never converge [46]. The generator continues to generate arbitrary data not fitting the data distribution. This can be mitigated by lowering the learning rate of the discriminator or taking multiple steps for the generator for every discriminator step [45, 47]. Both of these methods allow the generator to train at a faster rate. Successfully overcoming these shortfalls of GANs is imperative, otherwise the network does not learn to produce realistic data.

Conditional Generative Adversarial Networks

The Conditional GAN (CGAN) was introduced as an improvement to the GAN [41]. By allowing both the generator and discriminator functions in the GAN network to take additional external features as input, the architecture can learn a distribution of data given these additional inputs [41]. These additional inputs can vary from observation time to

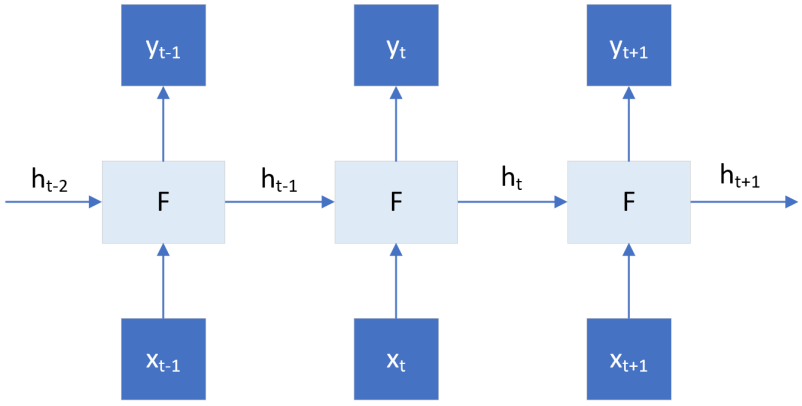


Figure 2.4: The recurrent neural network underlying mechanism [3].

classification labels. Thus, the loss becomes

$$L_{GAN} = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log(D(x, y))] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z, y)))], \tag{2.8}$$

where y consists of the additional external features corresponding to a specific x [41]. The additional input allows the GAN to better learn the distribution of data and mitigate the common problems of mode collapse and convergence. Often for classical GANs, the generator has more difficulty learning the data distribution. Thus, the discriminator is easily able to tell whether an input is real or synthetic. This additional input aids the generator in learning the distribution of data.

2.4 Neural Networks for Time Series Forecasting

2.4.1 Recurrent Neural Networks

A recurrent neural network (RNN) is a NN architecture that is well-suited for sequential data [3, 20, 21, 48]. The input time series vector $\boldsymbol{x}$ is concatenated with a hidden state

vector $h \in H$ where H is an arbitrary space. Thus, the linear mapping for an RNN is

$$f(x_t) = h_t = \begin{bmatrix} W_x & W_h \end{bmatrix} \begin{bmatrix} x_t \\ h_{t-1} \end{bmatrix} + b, \tag{2.9}$$

where $W_x \in \mathbb{R}^{m \times n}, W_h \in \mathbb{R}^{m \times m}$, and $b \in \mathbb{R}^m$ and are learned through training [3]. Recurrent networks iterate one time step at a time, and thus, an RNN can take sequences of any length as input [3]. Using the RNN output as input to subsequent layers, architectures are able to utilize information from previous time-steps without storing all prior data [3]. RNNs output into the hidden space, H, rather than the desired output space, Y. Thus, an additional mapping is required to obtain output in the desired space. Therefore, an arbitrary k-layer predictive RNN model takes the form

$$N(\boldsymbol{x}) = \sigma_k(f_k(...\sigma_1(f_1(\boldsymbol{x}))...)), \tag{2.10}$$

where f_i for $i = 1, ..., k-1$ are RNN layers, and f_k is a linear layer.

The workflow for a single layer RNN predictor is shown in Figure 2.4. In general h_0 is initialized to zeros. Beginning from the left side of Figure 2.4, the previous h_{t-2} vector and data x_{t-1} are taken as input to the recurrent layer, F, and outputs h_{t-1}. This h_{t-1} vector can then either be inputted to the additional linear mapping to produce the prediction, y_{t-1}, or is stored by the network to predict the next data in the sequence. This process follows for the middle and right section of Figure 2.4. The loss function for simple RNNs follows as equation (2.5). While training, the matrices W_x, W_h, and b are learned for h to learn long-term trends in data [3].

2.4.2 Gated Recurrent Unit

Specific sequences of RNNs optimize predictive capabilities. The Gated Recurrent Unit (GRU) is a specific combination of RNNs and activation functions that provides optimal usage for analysis of time series data [49]. The GRU consists of the following system:

$$r_t = \sigma(W_r \begin{bmatrix} x_t \\ h_{t-1} \end{bmatrix} + b_r), \tag{2.11}$$

$$z_t = \sigma(W_z \begin{bmatrix} x_t \\ h_{t-1} \end{bmatrix} + b_z), \tag{2.12}$$

$$n_t = \tanh(W_{nx}x_t + b_{nx} + r_t \odot (W_{nh}h_{t-1} + b_{nh})), \tag{2.13}$$

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot n_t, \tag{2.14}$$

where $W_r, W_z, W_{nx}, W_{nh}, b_r, b_z, b_{nx}$, and b_{nh} are learned , σ is the sigmoid activation function, and $\odot$ is the Hadamard product [49]. This formulation allow for r_t to act as a reset gate, allowing the system to remove any irrelevant information when calculating the next h_t, and z_t to act as the update gate, allowing the system to carry information from the previous hidden state to the next hidden state [49]. When r_t is close to zero, the n_t resets its dependence on h_{t-1} and relies on the data itself. Similar to the Long Short Term Memory (LSTM) architecture [50], the GRU allows models to retain long-term and short-term relationships between data [49, 51]. It has been shown that in forecasting, the LSTM and GRU achieve similar results [20, 51].

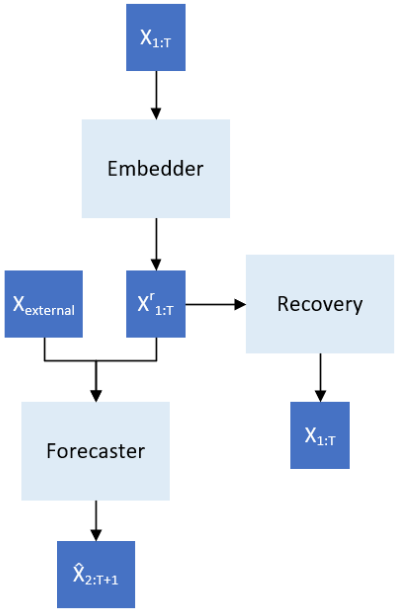


Figure 2.5: The autoencoder forecaster architecture [4].

2.4.3 Forecasting Architectures

By utilizing the RNN model of the form in equation (2.10), an arbitrary number of stacked RNNs can be utilized as forecasters. Nested cross-validation can be used to find the optimal number of layers for a given problem. Often, this model is utilized as the base case for comparison with new architectures [4, 21, 34].

An additional forecasting architecture is depicted in Figure 2.5, due to [4]. This architecture combines the autoencoder's inherent latent dimension with forecasting. By pre-training an autoencoder network on the data, then training a forecasting network to take encoded data as input, along with external features, and predict the next time step, the model is designed to forecast from the inherent latent space of the data [4].

Similar to the previous architecture discussed, the Auto-LSTM architecture pre-trains an autoencoder, then trains a forecaster within the reduced dimension [5]. The Auto-LSTM architecture is shown in Figure 2.6, where the forecaster is a single LSTM layer [5].

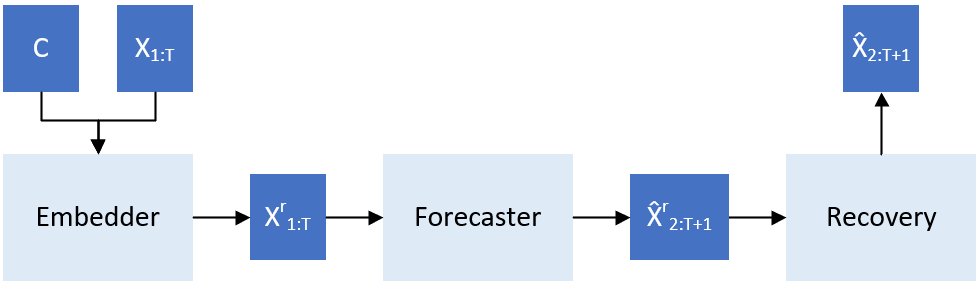


Figure 2.6: The forecaster architecture of the Auto-LSTM [5].

These architectures may encounter the same problems as there are for training autoencoders and RNNs. With the incorrect latent dimension, the network may not learn the optimal relationships within the data [1]. Since both methods require pre-training the autoencoder, in the case that the autoencoder fails to learn, the forecaster is unable to predict with incorrectly encoded data.

2.5 Uncertainty in Neural Networks

2.5.1 Evaluation Criteria

Uncertainty Quantification (UQ) in time series forecasting is important in applications for decision making processes [8, 16]. Depending on the NN architecture used, specific UQ techniques may be used. In the case of simple recursive NN forecasters, point prediction accuracy is computed. In this case, if there is uncertainty in the data itself, this error may be propagated in the prediction. However, this uncertainty may not be known, and therefore, for many point-predictive models, uncertainty may not be analyzed [17]. For point-predictors, many different error functions may be used as evaluation criteria, such as Mean Squared Error (MSE), Symmetric Mean Absolute Percentage Error (SMAPE), and Mean Absolute Squared Error (MASE). MSE calculates the distance between the prediction,

$\hat{y}_i$, and the actual data, y_i, given by

$$MSE(\hat{y}, y) = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2. \tag{2.15}$$

SMAPE is given by

$$SMAPE(\hat{y}, y) = \frac{100}{N}\sum_{i=1}^{N}\frac{|\hat{y}_i - y_i|}{(|\hat{y}_i| + |y_i|)/2}, \tag{2.16}$$

which uses the percentage errors to calculate accuracy [21]. Lastly for point prediction, MASE compares the prediction to the naïve forecast, which simply repeats the previous value as the forecast [21]. MASE is given by

$$MASE(\hat{y}, y) = \frac{\frac{1}{N}\sum_{i=1}^{N}|\hat{y}_i - y_i|}{\frac{1}{N-1}\sum_{i=2}^{N}|\hat{y}_i - \hat{y}_{i-1}|}. \tag{2.17}$$

Values for MASE less than 1 indicate better performance than the naïve forecast and values larger than 1 indicate worse performance than the naïve forecast.

The presence of multiple predictions for a particular time series allows for the computation of confidence intervals for UQ. Common methods, such as the Delta, Bayesian, and Mean-Variance Estimation Methods have an underlying Gaussian distribution assumption for the computation of confidence intervals as well as a required Hessian or gradient calculation [52]. Another common method for calculating confidence intervals is the *bootstrapping method* for NNs [52, 53]. This method consists of creating an ensemble of k trained recursive NN forecasters, $\Phi = \{\phi_1, ..., \phi_k\}$ [53]. Thus, the empirical variance of the forecasts for a particular input can be computed as

$$\sigma^2(x) = \frac{1}{k-1}\sum_{i=1}^{k}(\phi_i(x) - \phi_{\text{avg}}(x))^2, \tag{2.18}$$

where $\phi_{\text{avg}}(x)$ is the average of all k forecasts [53]. Thus, given the mean and variance of an ensemble, the standard Gaussian confidence interval may be computed by the probability density function. Although this method does not require the computation of gradients or Hessians, it does require the training of multiple forecasters and continues to assume Gaussian distributions [52, 53].

2.5.2 Neural Networks for Uncertainty Quantification

Additional NN architectures can give multiple forecasts for a single input. Rather than outputting the most probable outcome, models can learn to output distributions of possible predictions given initial inputs. This is beneficial in forecasting as giving certainty intervals rather than a single prediction can give more insight to future outcomes. In particular, the GAN architecture, which maps a known distribution of noise to a known distribution, with modifications is considered.

Using RNNs as the underlying generators, it has been shown that the CGAN and GAN architectures can generate time series data from known distributions [40, 48, 54, 55]. These promising results inspired additional research on using the CGAN and GAN architectures as forecasting models. Utilizing the GAN framework is a relatively new development in the forecasting field.

Generative Networks for Time Series

The time series Generative Adversarial Network (TimeGAN) architecture combines ideas from both the autoencoder and GAN architectures [6]. The autoencoder network remains the same with the embedding and recovery functions where $E : X \to H$ and $R : H \to X$. The GAN in the TimeGAN architecture consists of a generator that generates into the latent

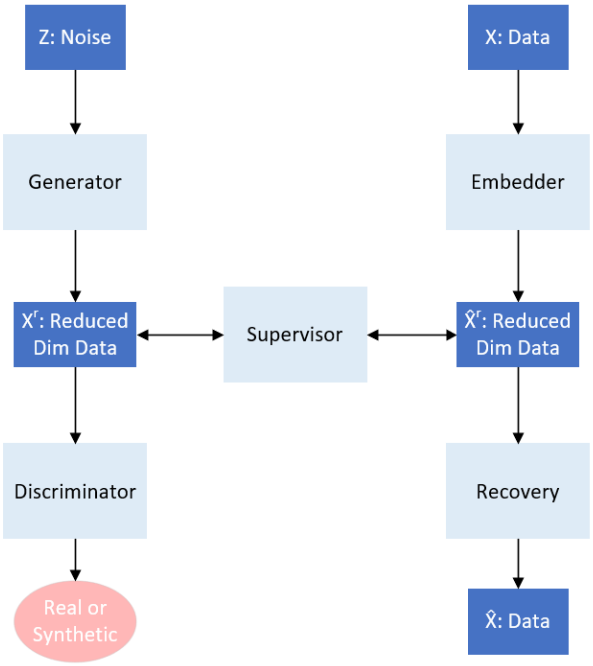


Figure 2.7: The TimeGAN architecture [6].

space rather than into the data space [6]. Therefore, the discriminator takes an element of the latent space as input and produces a value in $[0, 1]$. Thus, we denote the generator and discriminator NNs as $G : Z \to H$ and $D : H \to [0, 1]$. Within this latent space, an additional network, the supervisor, $S : H \to H$, performs operations in this optimal space [6]. It is expected that the autoencoder loss is similar to (2.6). Thus the new autoencoder loss for TimeGAN is given by [6],

$$L_{ER} = \sum_{t=1}^{T} ||x_t - R(E(x_t))||_2^2. \tag{2.19}$$

Similarly, the expected GAN loss in the TimeGAN architecture is given by

$$L_{GAN} = \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}(\boldsymbol{x})}} \left[\sum_{t=1}^{T} \log(D(S(E(x_t)))) + \sum_{t=1}^{T} \log(1 - D(S(G(h_{t-1}, z_t)))) \right], \tag{2.20}$$

where the generator aims to maximize loss function (2.20) and the discriminator aims to minimize loss function (2.20) [6]. The construction of this loss function follows similarly to

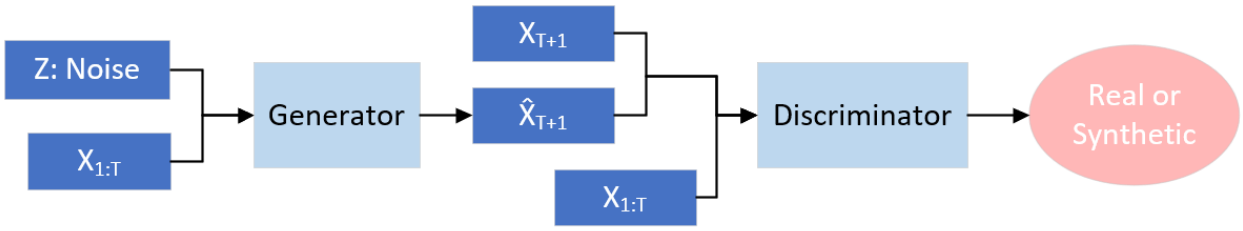


Figure 2.8: The ProbCast architecture [7].

(2.7). Lastly, within the latent dimension, it is expected that the generator generates data in the same distribution as the embedded data, therefore we have [6]

$$L_S = \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})} \left[\sum_{t=1}^{T} ||E(x_t) - G(h_{t-1}, z_t)||_2^2 \right]. \tag{2.21}$$

We observe that TimeGAN has similar advantages and disadvantages as the autoencoder and GAN architectures. TimeGAN may produce mode collapse, an overpowering discriminator, and incorrectly embedded data. However, embedding into this smaller latent space increases the ability of the GAN architecture to learn to produce feasible data. Additionally, the TimeGAN architecture does not address forecasting, as it aims to only create feasible sequences of data given a defined distribution of existing data. This work originally aimed to extend TimeGAN as a forecaster because of its promising results for matching time series distributions. However, differences in implementation [56] and the mathematical formulation [6] by the authors created obstacles in successfully implementing the architecture.

Time Series Forecasting with GANs

Recently, advancements have been made in forming GAN based architectures for forecasting. Koochali et. al. introduce the ProbCast model: a pre-trained forecaster that is then trained as a generator in a GAN [7]. The deterministic model is transferred to a generator by

maintaining the same architecture, except with the addition of a noise vector input. The model is then trained as a traditional GAN. This workflow is shown in Figure 2.8. The loss function for the first phase of training for ProbCast is given by [7]

$$L_{ProbCast,1} = ||G(\boldsymbol{x}) - x_{t+1}||_2^2. \tag{2.22}$$

The loss function for the second phase of training for ProbCast is given by [7]

$$L_{ProbCast,2} = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log(D(\boldsymbol{x}, x_{t+1}))] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(\boldsymbol{x}, G(\boldsymbol{x}, \boldsymbol{z})))]. \tag{2.23}$$

An advantage of ProbCast is that there remains an assumption of an existing latent dimension where the data is better represented. Inherently, recurrent functions map from a data space to a hidden space, then with evaluation of a linear mapping, the hidden representation is recovered into the data space. Additionally, if a pre-existing deterministic model exists as a forecaster, this architecture can utilize the forecaster without having to train one itself [7]. Disadvantages of ProbCast include the existing issues of GAN convergence and mode collapse. A similar version of ProbCast was introduced with similar structure, but only optimizes loss function (2.23). This model, ForGAN, focuses solely on the ability for the GAN learning technique to train a forecaster without comparisons with real and synthetic predictions [22]. The architecture is the same as that depicted in Figure 2.8.

Luo et. al. introduce using two generators and a single discriminator in a GAN architecture [23]. This allows one generator to take data as input, and another to only take noise as input [23]. They are jointly trained to forecast daily oil prices [23]. Luo et. al. conclude that this method improves forecasting accuracy as long as Adaptive Scales Continuous Wavelet Transformation is used to preprocess the data [23]. Lastly, Tan et. al. introduce MAD-GAN, a group of generators trained on a single discriminator [24]. This model takes advantage of

the GAN's tendency towards different modes by pushing each generator in the group to learn a specific mode of the forecast [24]. Although this model is able to learn multiple different forecasts, GAN training is computationally expensive and ensuring the correct number of generators in MAD-GAN introduces another hyperparameter to be found [24]. This architecture serves as an ensemble approach for NNs and allows for UQ to be performed. Results from the ProbCast, ForGAN, MAD-GAN, and two generator and one discriminator architectures prove the GAN architecture to be promising for time series prediction.

Chapter 3

Methodology

3.1 Latent Autoencoder

First, we introduce the Latent Autoencoder, a forecasting architecture. The architecture is depicted in Figure 2.6. This architecture is an adaption of the Auto-LSTM architecture, explained in Section 2.4.3 [5]. Rather than specify that the forecasting NN within the latent space is a single LSTM layer, the Latent Autoencoder allows the forecaster to be any sequence of RNNs [5]. Thus, the forecasting loss function for the Latent Autoencoder is given by

$$L_{\text{LatAuto}} = \sum_{t=1}^{T} ||x_{t+1} - R(F(E(x_t, c_t)))||_2^2, \tag{3.1}$$

where $\boldsymbol{x} = \{x_1, ..., x_{T+1}\}$ is the data sequence and $\boldsymbol{c} = \{c_1, ..., c_{T+1}\}$ is the corresponding condition. This architecture will have similar advantages and distadvantages to the architectures referenced in Section 2.4.3.

3.2 LatentGAN

Here, we introduce the LatentGAN architecture (Figure 3.1). With LatentGAN, we aim to build from the advantages of the architectures discussed in Chapter 2 while mitigating their disadvantages. LatentGAN is a forecasting GAN that is composed of an autoencoder and

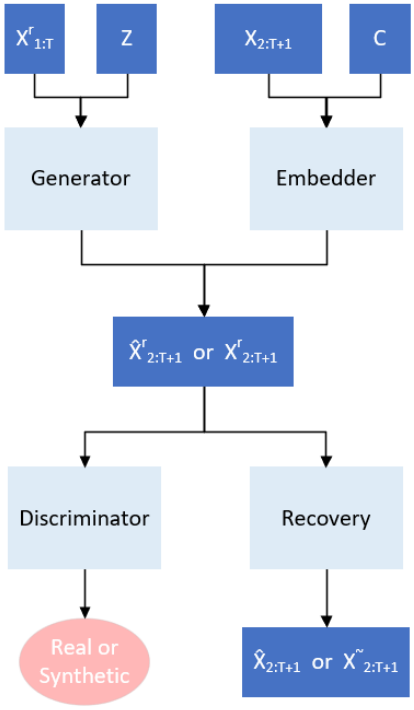


Figure 3.1: The LatentGAN architecture.

a predictive CGAN. Thus, the RNNs composing the architecture consist of the embedder, $E : X \times C \to X^r$, recovery, $R : X^r \to X$, generator, $G : X^r \times Z \to X^r$, and discriminator, $D : X^r \to [0, 1]$, functions.

Assembly and training of LatentGAN is a three phase process. First, the autoencoder is trained until convergence. This allows us to map data into a reduced space. However, the autoencoder may converge to a local minimum. To mitigate this risk, cross-validation is utilized to determine the optimal reduced dimension. Second, the generator is trained as a single step forecaster within the reduced dimension space. This allows the generator produce point predictions close to the expected value. Lastly, the generator and discriminator are trained as a GAN ensuring that the generator produces data within the prediction distribution.

3.2.1 Phase 1: Autoencoder

First, we analyze the autoencoder of the proposed model. The loss function is given by

$$L_{\text{Model ER}} = \sum_{t=1}^{T} ||x_t - R(E(x_t, c_t))||_2^2, \tag{3.2}$$

where $\boldsymbol{x}$ is a data sequence and $\boldsymbol{c}$ is the corresponding conditional input. This loss follows the loss of the standard autoencoder for RNNs in equation (2.19) and allows the model to learn the underlying data representation. In order to make meaningful predictions from the latent space, the model must be pre-trained with loss function (3.2). By training the remaining NN functions of the model on unchanging latent representations of data, we hypothesize that the model is able learn to forecast without the risk of the autoencoder failing to learn the latent representation correctly.

3.2.2 Phase 2: Forecaster

The second phase of training for LatentGAN is to train the generator as a shifted time series predictor within the latent space. Thus, the loss function for the predictor phase is given by

$$L_{\text{Prediction}} = \sum_{t=1}^{T} ||R(E(x_{t+1}, c_{t+1})) - R(G(E(x_t, c_t), z_t))||_2^2, \tag{3.3}$$

where $\boldsymbol{c}$ is the condition for the corresponding data $\boldsymbol{x}$, and $\boldsymbol{z} = \{z_1, ..., z_{T+1}\}$ is the zero vector. Loss function (3.3) compares the conditional generator with inputs corresponding to the time sequence $1 : T$ with the embedder evaluated on the data from $2 : T+1$. This allows the generator to function as a single step forecaster in the reduced-dimension space without any variance in the prediction. Another loss, similar to loss function (3.3), compares both of these evaluations within the latent space rather than in the data space, that is, removing

the recovery function from loss function (3.3). However, since this nonlinear model may not preserve expected properties within the underlying latent space, the data distribution within the known data space is expected to be more informative to the model.

Phase 2 of LatentGAN is an adaptation to the Latent Autoencoder introduced in Section 3.1. In particular, the forecaster in the Latent Autoencoder is specified as the generator in LatentGAN. Thus, results of Phase 2 of LatentGAN are dependent on results from the Latent Autoencoder.

3.2.3 Phase 3: Generative Adversarial Network

The third and final phase of training for LatentGAN is the GAN training phase. The loss function is given by

$$\begin{aligned} L_{\text{GAN}} =& \mathbb{E}_{x\sim p_{\text{data}(x)}, z\sim p(z)} \left[\sum_{t=1}^{T} \log(D(R(E(x_{t+1}, c_{t+1})))) + \sum_{t=1}^{T} \log(1 - D(R(G(E(x_t, c_t), z_t)))) \right] \\ &+ \mathbb{E}_{x\sim p_{\text{data}(x)}, z\sim p(z)} L_{\text{prediction}}, \end{aligned} \tag{3.4}$$

where $\boldsymbol{c}$ is the condition corresponding to data $\boldsymbol{x}$, and $\boldsymbol{z}$ is a random vector chosen from the uniform distribution from 0 to 1. These first two terms are similar to the corresponding terms in equation (2.20), the TimeGAN GAN loss function. Lastly, the loss function compares the distribution of equation (3.3) rather than the distance between the predicted and actual values due to the variability in future predictions. This allows the generator to generate a suite of predictions rather than fixate on a single point prediction.

3.2.4 Training

Thus, LatentGAN completes the three training phases separately and sequentially. Training all three phases separately allows the model to converge in each phase regardless of the other parts of the model. This method also minimizes interference of training between multiple functions. For example, training a generator to forecast on ill-represented data in the latent space due to an under-trained autoencoder may not produce informative predictions. Intuitively, phase 2 ensures LatentGAN's single point predictions are close to the expected value. Phase 3 allows LatentGAN to generate suites of predictions that are within distribution. We expect the pre-training of the generator function in phase 2 to prevent mode collapse as well as non-convergence.

Chapter 4

Results

4.1 Data Preprocessing

To improve training performance, data must be preprocessed before NN training [54]. Data is normalized to be within $[0, 1]$ by the common min-max scalar [20, 54, 56, 57]

$$x = \frac{x - \min(x)}{\max(x) - \min(x)}. \tag{4.1}$$

Re-scaling data to this interval reduces the number of epochs required for training due to the data being within a specific range optimized for the initialized weights of the model [20].

Traditional RNNs have been found to poorly recognize and predict seasonality within normalized datasets [21]. It is known that the FCR data exhibits both daily and yearly cyclic trends. Taieb, et al. compares the effects of different preprocessings of cyclic and seasonal data on the accuracy of predictive models [34]. Deseasonalization within forecasting is often performed by the X-11 method [21]. Another technique for deseasonalization is concatenating the following transformations with the original data [20]:

$$\tilde{x} = \sin\left(\frac{2\pi x}{\max(x)}\right), \tag{4.2}$$

$$\tilde{x} = \cos\left(\frac{2\pi x}{\max(x)}\right). \tag{4.3}$$

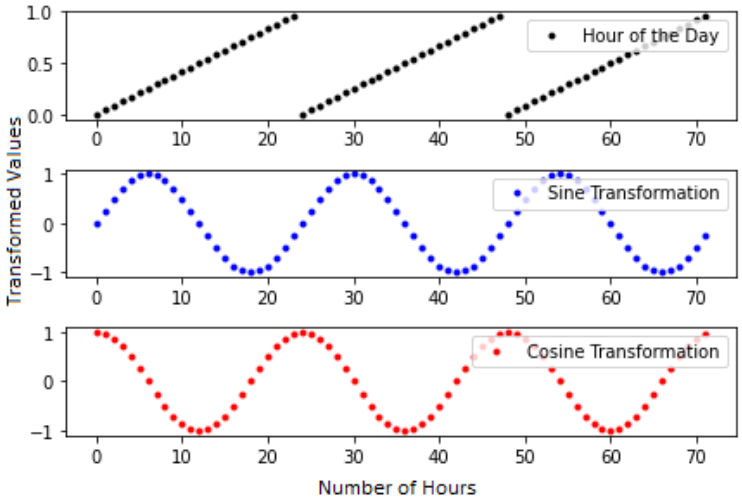


Figure 4.1: Deseasonalized hours of the day.

By performing both transformations on data, models are able to map distinct transformed values to each piece of data allowing the model to recognize relationships within cyclic data [20]. A simple example of deseasonalized hours of the day is shown in Figure 4.1. Equations (4.2) and (4.3) deseasonalize and transform the discontinuous hours of the day into continuous curves.

Detrending data consists of quantifying the overall trend of the time series data and subtracting this trend from the data, which centers the data around 0 [58]. This detrends the data by removing the long-term trend component. Although this method addresses forecasting outside of the normalized data range of $[0, 1]$, it does not consider the effect of changing long-term trends on the predictive capabilities of models. Additionally, retaining long-term trends may allow models to correlate data with conditions such as time. Since it is unknown whether ecological processes will continue to follow current trends, detrending will not be considered in this work.

4.2 Hyperparameters

We consider two forecasting studies using five forecasting models: a basic RNN, autoencoder forecaster, Latent Autoencoder, ProbCast, and LatentGAN. The basic RNN, autoencoder forecaster, Latent Autoencoder, and autoencoder of the LatentGAN are GRU RNNs based on our PyTorch adaptation of the original TimeGAN implementation [56]. They can be represented as NNs of the arbitrary RNN predictive model form in equation (2.10) where $k = 4$, the f_1, f_2, and f_3 layers are GRU layers, the f_4 layer is a linear layer, and the final σ_4 is a sigmoid activation function. To increase comparability, all models containing autoencoders consist of the same weights. The ProbCast model and GAN of LatentGAN are based on the implementation in ForGAN [59]. In these, the generator is also an arbitrary RNN predictive model where $k = 3$, f_1 is a GRU layer, f_2 and f_3 are linear layers, and the final σ_3 is a ReLU activation function. Similarly, the discriminator is an arbitrary RNN predictive model where $k = 2$, f_1 is a GRU layer, f_2 is a linear layer, and σ_2 is a ReLU activation function. The Adam optimizer is used with a learning rate of 0.001 and default beta values of $(0.9, 0.999)$ for all models.

4.3 Google Stock

Google stock data is available from 2004 to 2019 [6]. This data set consists of opening, high, low, close, and adjusted close daily prices, as well as the volume of stock [6]. The first 80% of data is used as the training set and the last 20% of data is used as the test set.

This data is not expected to yield seasonal or cyclic fluctuations, so the data is not pre-processed to include deseasonalization or time. Each model and each phase of training is completed with 100 epochs. Cross-validation was used to determine the optimal latent

Model	SMAPE	MASE
RNN	19.96	13.78
Autoencoder Forecaster	19.78	6.86
Latent Autoencoder	21.21	11.46
ProbCast	32.46	-
LatentGAN	23.74	7.74

Table 4.1: Table of model accuracies for Google stock data.

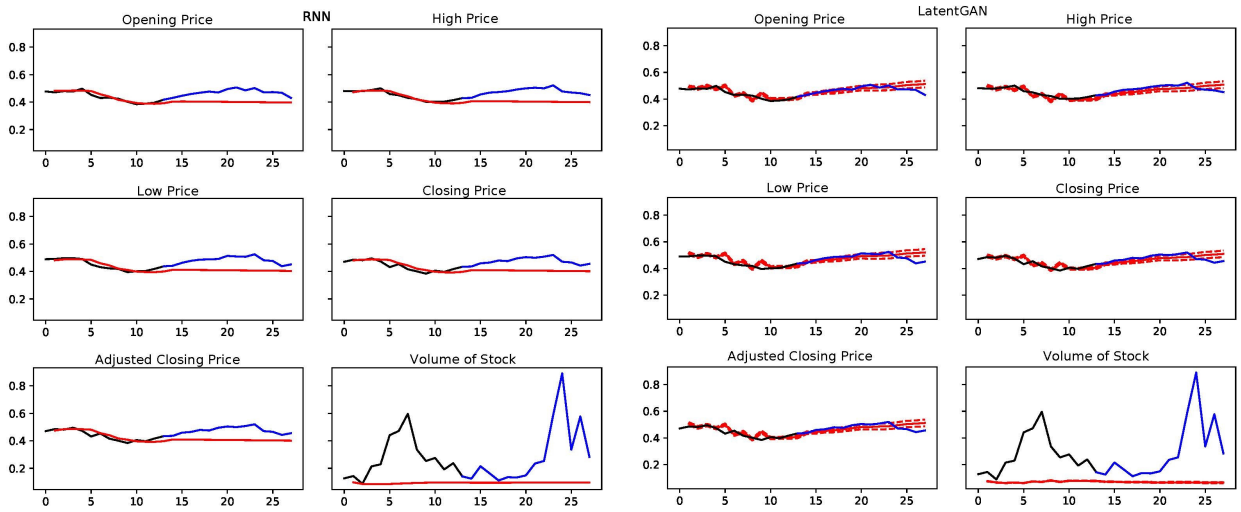
dimension size of 4. A sequence length of 14 was chosen to correspond to two weeks of data.

4.3.1 Comparative Analysis

To ensure that the data is well represented in the reduced space, we first evaluate the test accuracy of the autoencoder. The MSE test accuracy of the autoencoder is 7.67×10^{-3}. Overall, the autoencoder performs well with representing the Google stock data in the latent space.

Single Step Forecast

The test accuracies of each model for the Google stock data is given in Table 4.1. The autoencoder forecaster performs the most accurately as a single step predictor in both the SMAPE and MASE evaluation criteria. The RNN model performs second best of SMAPE and worst for MASE values. The LatentGAN model performs fourth in SMAPE and second in MASE. However, these large error values are likely due to inaccurate predictions in volume of stock.



(a) The classical RNN prediction for a single Google stock time series.

(b) The classical LatentGAN prediction for a single Google stock time series.

Figure 4.2: The classical RNN and LatentGAN prediction for a single Google stock time series.

Model	SMAPE	MASE
RNN	22.18	38.48
LatentGAN	21.74	12.28

Table 4.2: Table of error values corresponding to Figure 4.2.

Recursive Forecast

Figure 4.2 shows the true values and forecasts for an arbitrary time window. The RNN and LatentGAN predictions are depicted for a single time series. The average prediction of a 100 member ensemble and upper and lower bounds for the 95% confidence interval are depicted in the solid and dashed red line plots respectively for LatentGAN. We observe that the prediction of LatentGAN performs better than that of the RNN for all prices. However, both models perform very poorly for the volume of stock prediction. While the first five values are visually correlated, the volume of stock does not follow the same trend. The poor predictions in volume of stock likely cause the large error values in Tables 4.1 and 4.2.

4.4 Falling Creek Reservoir

The data utilized in the case study includes the single water column temperature data taken every ten minutes from July 6, 2018 to January 1, 2021 [28]. The data from July 6, 2018 to December 31, 2019 are used as training data, while the data from January 1, 2020 to January 1, 2021 are used as the test data. The data is further split into four week sequences of 6 hour interval data. The models are trained as recursive single step forecasters and are tested to predict two weeks of data.

Regarding the preprocessing procedures, there are four distinct sets of inputs considered: temperature data, temperature data concatenated with the deseasonalized temperature data, temperature data concatenated with the deseasonalized time, and temperature data concatenated with deseasonalized temperature data and deseasonalized time. The deseasonalized data are the transformations on temperature data performed by equations (4.2) and (4.3). The deseasonalized time are the transformations on day of the year as well as time of day performed by equations (4.2) and (4.3). Four instances of each model are trained to analyze the affect of each additional input on the predictive accuracy. With the four distinct inputs, we analyze twenty distinct models. To increase comparability, the embedding and recovery functions of each model containing these functions have the same weights. All models and phases are trained on 20 epochs, except phase 3 of LatentGAN, which is trained on 10 epochs. The optimal hidden dimension of 8 was chosen through cross-validation of the autoencoder. A sequence length of 56 was chosen to correspond to two weeks of data.

4.4.1 Comparative Analysis

Before analyzing the predictive capabilities of each model, the trained autoencoder is tested. Table 4.3 denotes the autoencoder's test accuracy for each given set of inputs. Conditional

Input	MSE
Data	4.2004×10^{-4}
Data + DS	8.3598×10^{-4}
Data + T	5.0664×10^{-4}
Data + DS + T	3.5404×10^{-3}

Table 4.3: Table of autoencoder accuracies for given data inputs.

Model	SMAPE	MASE
RNN	3.62 (2)	3.50
RNN: DS	4.11	3.77
RNN: T	3.33 (1)	3.17 (3)
RNN: DS,T	3.67 (3)	3.70
Autoencoder Forecaster	4.01	3.32 (4)
Autoencoder Forecaster: DS	4.96	5.40
Autoencoder Forecaster: T	3.89	4.59
Autoencoder Forecaster: DS,T	5.15	6.15
Latent Autoencoder	3.84	3.43
Latent Autoencoder: DS	5.01	6.07
Latent Autoencoder: T	3.74 (4)	3.70
Latent Autoencoder: DS,T	6.12	5.23
ProbCast	24.91	-
ProbCast: DS	27.02	-
ProbCast: T	45.26	-
ProbCast: DS,T	60.86	-
LatentGAN	4.77	1.45 (1)
LatentGAN: DS	6.24	5.92
LatentGAN: T	4.81	2.24 (2)
LatentGAN: DS,T	66.64	4.50

Table 4.4: Table of FCR test data set SMAPE and MASE values.

inputs to each model are designated by deseasonalized (DS), Time (T), and deseasonalized and time (DS + T). While the data, data + DS, and data + T autoencoders perform on the same order of magnitude, the data + DS + T autoencoder does not. However, these low test accuracies indicate that the data is well represented in the latent space.

Single Step Forecast

The test accuracies of each model with specified inputs are given in Table 4.4. The top four ranked models are denoted in the red parenthesis to the right of each error value. Despite research showing that both deseasonalization and time improve the accuracy of predicton, this is not the case for all of the single step models. For each model analyzed, the addition of deseasonalization decreases single step accuracy compared to the plain model. The addition of time does not reliably increase single step accuracy, and the effect of both inputs does not provide conclusive results across all models. Regarding overall rankings, the RNN with time input performs the most consistently in evaluation criteria. The classical LatentGAN model performs the best in MASE loss, and the LatentGAN model with time input is the second most accurate in MASE loss.

Recursive Forecast

To test the recursive prediction accuracy of LatentGAN, we perform two week recursive predictions and compare with the RNN model. A single experiment is shown in Figures 4.3, 4.4, 4.5, and 4.6 with corresponding error calculations in Tables 4.5, 4.6, 4.7, 4.8 respectively. This experiment was chosen arbitrarily by a random number generator. The figures depict the average of a 100 ensemble prediction as well as the upper and lower bounds for the 95% confidence intervals of prediction for LatentGAN in the solid and dashed red line plots

respectively. The error calculations for LatentGAN in the experimental tables are calculated with the average of the 100 ensemble prediction. Shown in Figure 4.3, although the classical RNN model performs among the best of all the models trained on single step predictions, we observe that the RNN is not able to predict well recursively. The LatentGAN model is able to predict two weeks in advance more accurately for the lower depths in Figure 4.3.

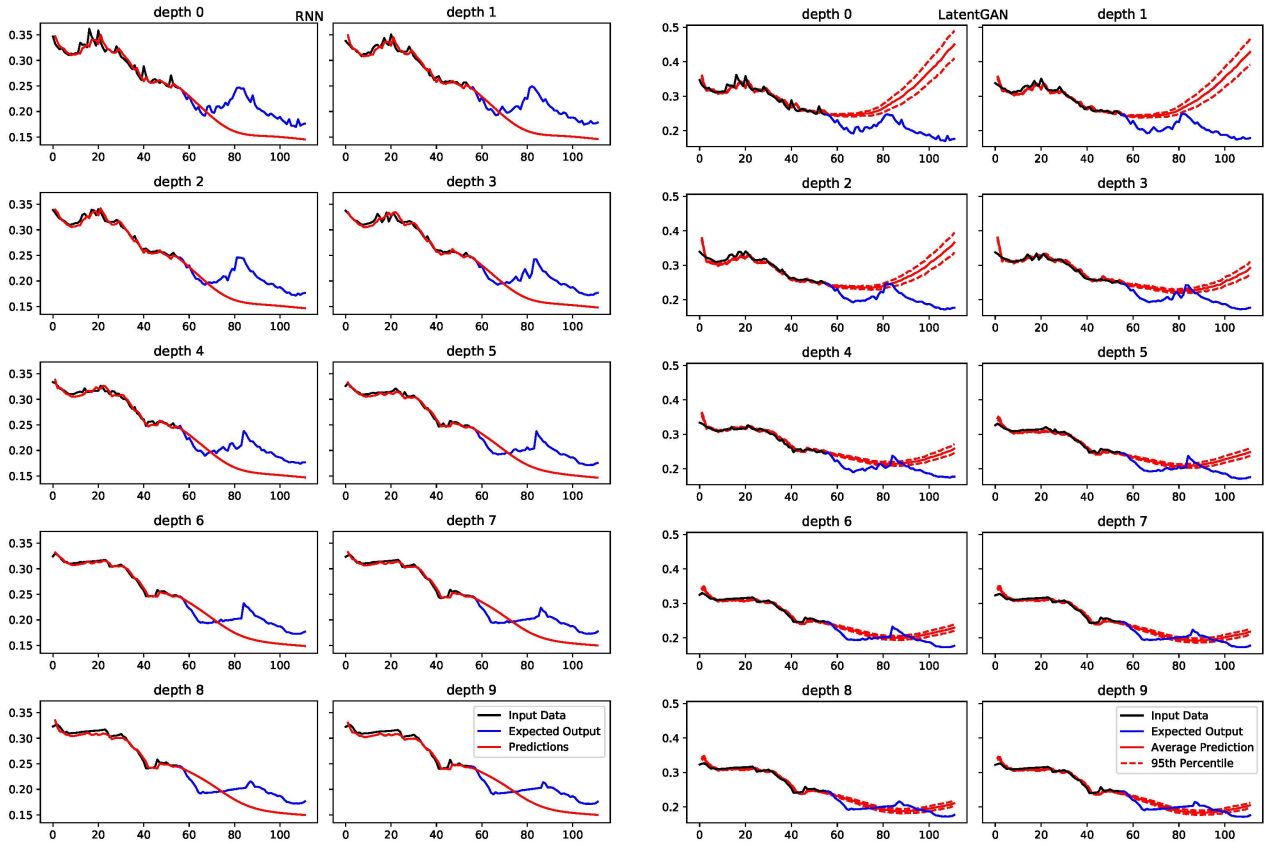
Figures 4.4 and 4.5 show that adding additional output for the RNN model improves accuracy. However, for the LatentGAN model, the data + DS input improves upper depth predictions, but deteriorates lower depth recursive predictions. For the time input experiment, we observe that for the first week of predictions, LatentGAN is able to predict well, but does not follow correct trends afterwards.

Notice that the scale on Figure 4.6(b) is much different than those of Figures 4.3, 4.4, and 4.5. However the data remains the same. The scale difference is due to the inaccurate prediction of the LatentGAN with data + DS + T input. Suprisingly, the addition of both DS and T inputs slightly improves recursive prediction accuracy for the RNN, and slightly deteriorates prediction accuracy for LatentGAN in Figures 4.4 and 4.5. In the data, data + DS , and data + T LatentGAN models, we observe the propagation of uncertainty with time as hypothesized also depicted in Figures 4.4 and 4.5. However, the predictions, even with 95% confidence intervals are not as accurate as would be desired. It is also observed that the addition of deseasonalization allows each model to better predict long-term trends in Figure 4.4, while the addition of time input allows each model to better predict the daily fluctuations in data in Figure 4.5.

Additional experiments are shown in Appendices A and B. Appendix A includes an experiment starting on October 12, 2020 at 12:00AM. It is known that fall turnover occurred for the FCR on November 2, 2020, corresponding to data from time steps 84 to 88 for all figures in Appendix A [60]. We observe that the simple RNN with specified inputs performs better

than all LatentGAN models with corresponding inputs except for the models with additional time input. The classical LatentGAN model performs slightly worse than the classical RNN, shown in Figure A.1 and Table A.1. The LatentGAN models with DS and DS + T input perform much worse than their respective RNN models observed in Figures A.2 and A.4 with respective error tables A.2 and A.4. However, we observe in Figure A.3 and Table A.3 that the LatentGAN model with time input performs better in SMAPE and very similarly in MASE with the respective RNN model. Overall the RNN + DS + T model performs the most accurately in evaluation criteria for the fall turnover experiment.

Appendix B includes an experiment starting on June 11, 2020 at 12:00 AM. This time period corresponds to summer stratification: when there are defined layers of temperature within the water column [8]. In Figures B.1, B.2, and B.3, the LatentGAN model outperforms or performs very similarly to the RNN model with corresponding input. Evaluation criteria values for these figures are found in tables B.1, B.2, and B.3.



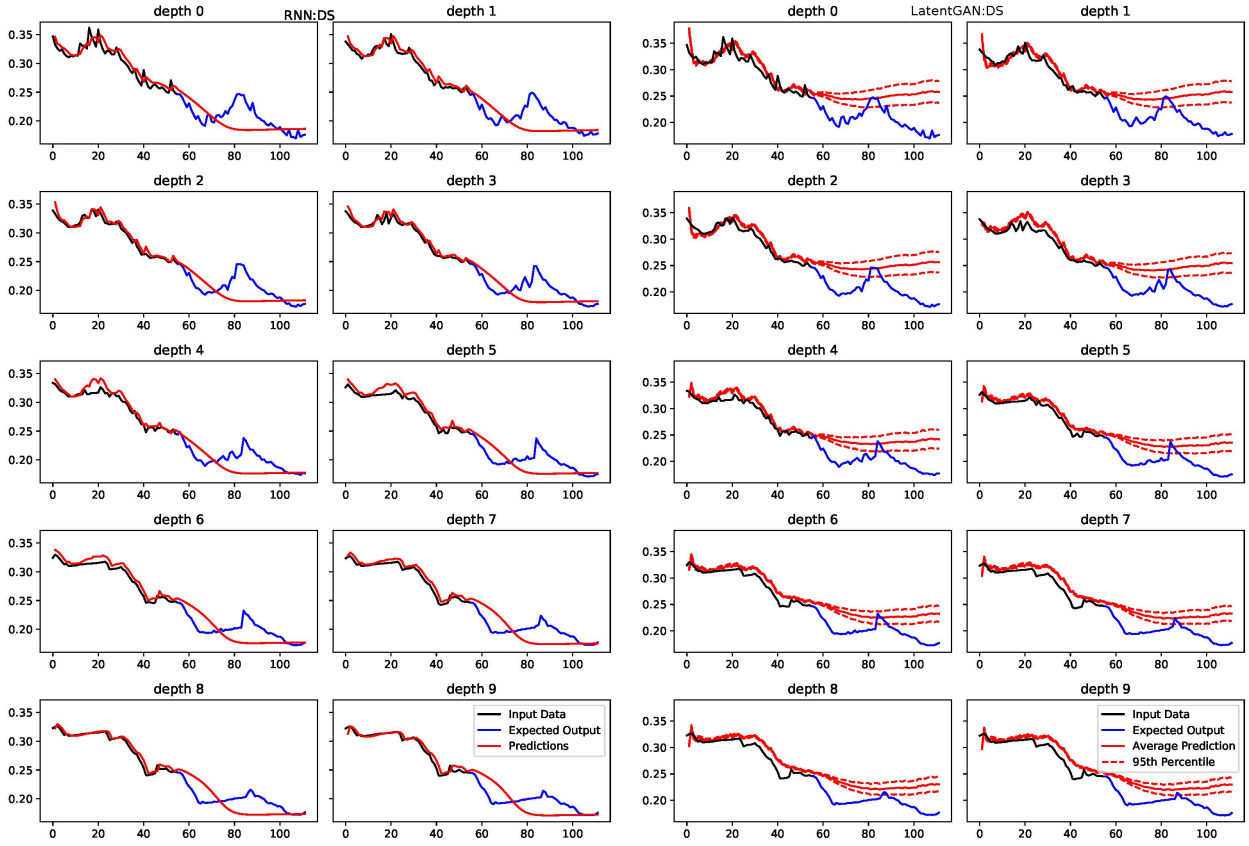
(a) Classical RNN 2 week predictions at all 10 depths of the water column.

(b) Classical LatentGAN 100 ensemble average 2 week prediction and 95% confidence interval at all 10 depths of the water column.

Figure 4.3: RNN and LatentGAN 2 week predictions beginning on November 23, 2020 at 6:30 PM.

Model	SMAPE	MASE
RNN	8.83	6.55
LatentGAN	9.72	7.59

Table 4.5: Table of error values corresponding to Figure 4.3.



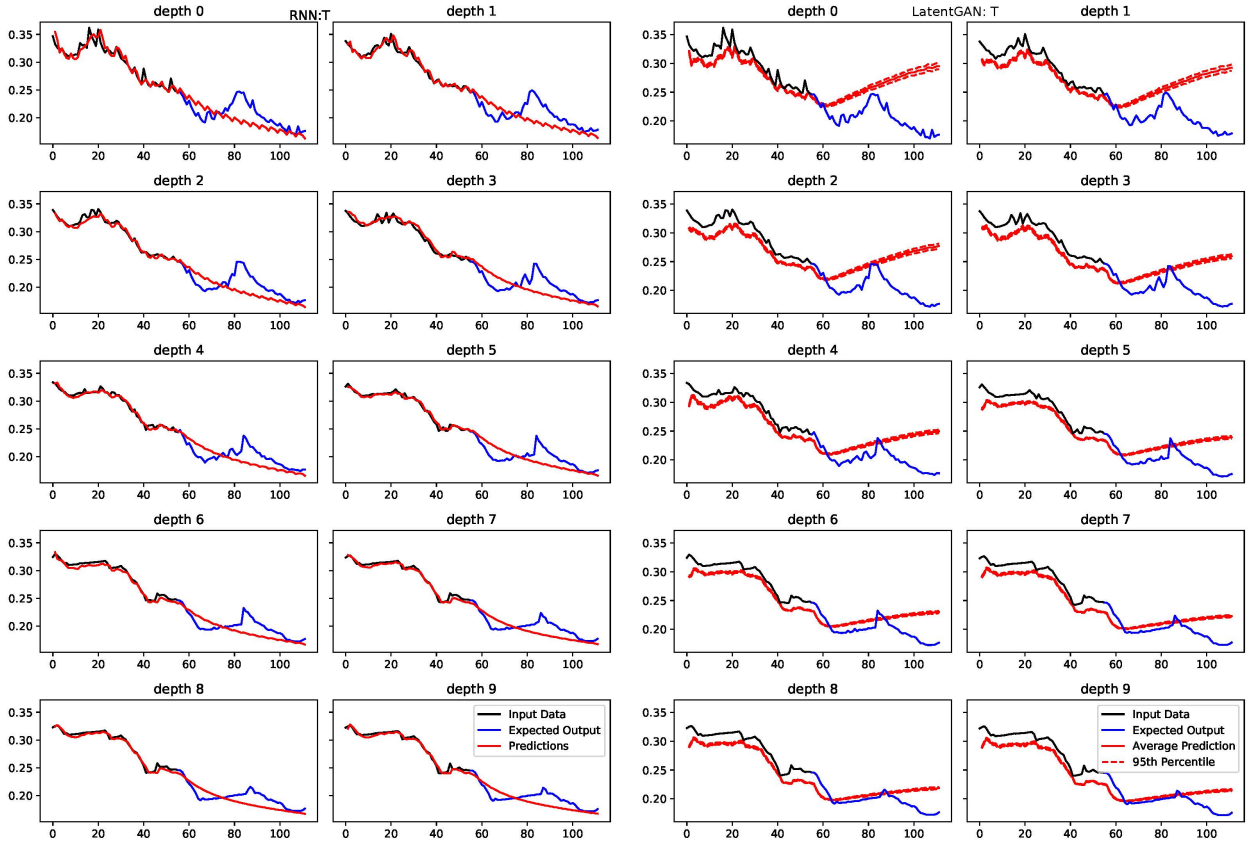
(a) RNN with data + DS input 2 week predictions at all 10 depths of the water column.

(b) LatentGAN with data + DS input 100 ensemble average 2 week prediction and 95% confidence interval at all 10 depths of the water column.

Figure 4.4: RNN and LatentGAN with data + DS input 2 week predictions beginning on November 23, 2020 at 6:30 PM.

Model	SMAPE	MASE
RNN	5.83	5.15
LatentGAN	10.54	10.04

Table 4.6: Table of error values corresponding to Figure 4.4.



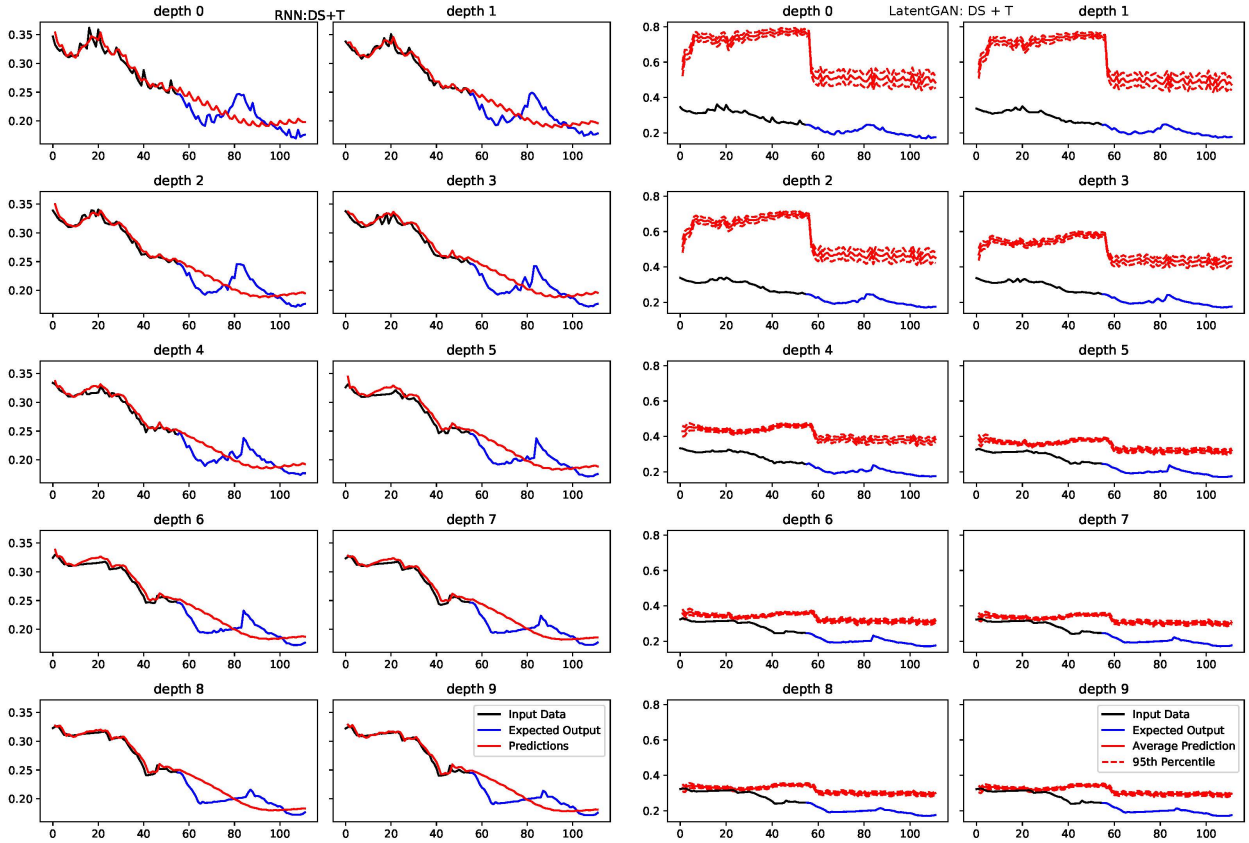
(a) RNN with data + T input 2 week predictions at all 10 depths of the water column.

(b) LatentGAN with data + T input 100 ensemble average 2 week prediction and 95% confidence interval at all 10 depths of the water column.

Figure 4.5: RNN and LatentGAN with data + T input 2 week predictions beginning on November 23, 2020 at 6:30 PM.

Model	SMAPE	MASE
RNN	4.28	3.55
LatentGAN	10.25	11.55

Table 4.7: Table of error values corresponding to Figure 4.5.



(a) RNN with data + DS + T input 2 week predictions at all 10 depths of the water column.

(b) LatentGAN with data + DS + T input 100 ensemble average 2 week prediction and 95% confidence interval at all 10 depths of the water column.

Figure 4.6: RNN and LatentGAN with data + DS + T input 2 week predictions beginning on November 23, 2020 at 6:30 PM.

Model	SMAPE	MASE
RNN	5.26	4.40
LatentGAN	52.00	20.90

Table 4.8: Table of error values corresponding to Figure 4.6.

Chapter 5

Discussion and Future Work

This work analyzes the advantages and disadvantages of existing generative and predictive NN architectures for time series data. The LatentGAN architecture is then introduced as a joint autoencoder and generative forecaster with three training phases. Results show that LatentGAN is a better single step forecaster than recursive forecaster, as well as being comparable with state-of-the-art single step forecasters. In the Google stock case study, LatentGAN outperforms the base case in point prediction for MASE and recursive predictions for both evaluation metrics. Regarding the FCR case study, LatentGAN performs slightly worse in SMAPE evaluation metric than that of the base case, and performs the best in MASE evaluation metric for single step predictions. Additionally, the quality of expected growth in uncertainty over time is observed in the FCR and Google stock case studies. However, average ensemble predictions leave room for improvement.

The poor results of LatentGAN can be accounted for in a number of unoptimized hyperparameters. While within a reasonable range, the number of GRU layers, training epochs, data frequency, and sequence length are among the unoptimized hyperparameters that may affect LatentGAN's ability to learn. Furthermore, the reduced dimension for the FCR case study was optimized for the autoencoder with no additional inputs, rather than performing cross-validation for all four sets of inputs. Future work in optimization of hyperparameters could improve predictive results. Along with optimized parameters, sensitivity analysis may be performed to give insight to the reliability and sensitivity of each parameter, as well as

the robustness of results. Additionally, the available training data for the FCR case study is limited to less than two years of data. Thus, when the model encounters a specific time period that only has a single data point to compare with, the model may overfit to this single data point. The meteorological FCR datasets have also been omitted from the case study. Utilization of these data sets as well as longer data sets in terms of time may be required for GAN training.

Although LatentGAN is shown to be comparable with state-of-the-art NN forecasting techniques, it has yet to be compared to forecasting techniques outside of the NN community. Future work with comparisons of LatentGAN with FLARE, the GLM, and linear reduction models may give insight to the linearity and nonlinearity of the case studies.

Additionally, the GAN architecture, while able to match data distributions, does not guarantee that inputs are intelligently chosen to represent the mapped distribution. The implementation of Markov Chain Monte–Carlo sampling in the noise vector input space may better map the noise distribution to the data distribution [61]. Given a specific number of samples to be chosen, the sampling method chooses a random sample, and rejects or accepts the sample based on the posterior value of that sample [61]. Choosing more informed noise vectors would allow LatentGAN to better map the distribution of data.

Improvements to the LatentGAN architecture include transforming the model to be both data and process-driven. Current state-of-the-art lake and reservoir modeling draws upon both data and process-driven models. Physics Informed Neural Networks (PINNs) are a class of process-driven NNs that utilize known differential equations that govern a particular problem along with data to train time series predictors [62, 63]. Further extensions of LatentGAN may enhance the architecture into a data and process-driven model by including process-driven inputs in LatentGAN's prediction: a similar idea to PINNs. Adding GLM outputs as inputs to LatentGAN for the FCR case study would extend the model into a

data and process-driven model. This would give the LatentGAN model insight to known ecological processes without explicitly embedding them. Many fields have existing process-driven models that are not exact and contain linear assumptions, and thus have room for nonlinear improvements [8, 18, 64].

LatentGAN has proven to successfully produce single step predictions for both case studies compared to existing architectures. However, the current architecture does not produce reliable long-term results according to evaluation criteria. This work serves as the first steps in utilizing GANs as predictors in the ecology field. To apply the current LatentGAN architecture to an additional case study, availability of large time series data sets and the usage of cross-validation would be required to determine optimal hyperparameters prior to training. These steps would ensure the most accurate results in both single step and recursive prediction.

In practice, ecologists can utilize results from LatentGAN to predict future ecological events. By analyzing the 95% prediction confidence intervals, ecologists and reservoir management can make informed decisions to preserve water quality. For example, if the confidence intervals for each depth in the FCR case study overlap, this may indicate a possibility for the occurrence of spring or fall turnover. Improving LatentGAN's accuracy would increase confidence in predictions and better inform decision-makers and ecologists about upcoming ecological events.

Bibliography

[1] I. J. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA: MIT Press, 2016, http://www.deeplearningbook.org.

[2] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," *Neural Information Processing Systems*, Jun. 2014.

[3] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, *Dive into Deep Learning*, Mar. 2020, https://d2l.ai.

[4] L. Zhu and N. Laptev, "Deep and confident prediction for time series at Uber," *2017 Institute of Electrical and Electronics Engineers International Conference on Data Mining Workshops (ICDMW)*, Nov. 2017. [Online]. Available: http://dx.doi.org/10.1109/ICDMW.2017.19

[5] A. Gensler, J. Henze, B. Sick, and N. Raabe, "Deep learning for solar power forecasting — an approach using autoencoder and LSTM neural networks," in *2016 Institute of Electrical and Electronics Engineers International Conference on Systems, Man, and Cybernetics (SMC)*, Oct. 2016, pp. 002 858–002 865.

[6] J. Yoon, D. Jarrett, and M. van der Schaar, "Time-series generative adversarial networks," in *Advances in Neural Information Processing Systems*, Nov. 2019, pp. 5508–5518.

[7] A. Koochali, A. Dengel, and S. Ahmed, "If you like it, GAN it. Probabilistic multivariate times series forecast with GAN," *ArXiv:2005.01181 [cs.LG]*, May 2020.

[8] R. Q. Thomas, R. J. Figueiredo, V. Daneshmand, B. J. Bookout, L. K. Puckett, and C. C. Carey, "A near-term iterative forecasting system successfully predicts reservoir hydrodynamics and partitions uncertainty in real time," *Water Resources Research*, vol. 56, no. 11, Sep. 2020. [Online]. Available: https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2019WR026138

[9] A. C. Antoulas, *Approximation of large-scale dynamical systems*. Philadelphia: Society for Industrial and Applied Mathematics, 2005.

[10] A. C. Antoulas, C. A. Beattie, and S. Güğercin, *Interpolatory methods for model reduction*. Philadelphia: Society for Industrial and Applied Mathematics, 2020.

[11] U. Baur, P. Benner, and L. Feng, "Model order reduction for linear and nonlinear systems: a system-theoretic perspective," *Archives of Computational Methods in Engineering*, vol. 21, no. 4, pp. 331–358, Aug. 2014.

[12] P. Benner, M. Ohlberger, A. Cohen, and K. Willcox, *Model Reduction and Approximation*. Philadelphia: Society for Industrial and Applied Mathematics, 2017.

[13] A. Quarteroni, A. Manzoni, and F. Negri, *Reduced Basis Methods for Partial Differential Equations: An Introduction*. Cham: Springer, 2015, vol. 92.

[14] G. Scarciotti and A. Astolfi, "Data-driven model reduction by moment matching for linear and nonlinear systems," *Automatica*, vol. 79, pp. 340–351, Jan. 2017.

[15] P. Andreini, C. Izzo, and G. Ricco, "Deep dynamic factor models," *ArXiv:2007.11887 [econ.EM]*, Jul. 2020. [Online]. Available: https://arxiv.org/abs/2007.11887

[16] J. H. Kwakkel, W. E. Walker, and V. A. W. J. Marchau, "From predictive modeling to exploratory modeling: How to use non-predictive models for decisionmaking under

deep uncertainty," in *Proceedings of the 25th Mini-EURO Conference on Uncertainty and Robustness in Planning and Decision Making (URPDM2010)*, Apr. 2010.

[17] A. Gaba, D. G. Popescu, and Z. Chen, "Assessing uncertainty from point forecasts," *Management Science*, vol. 65, no. 1, pp. 90–106, Jan. 2019.

[18] M. R. Hipsey, L. C. Bruce, C. Boon, B. Busch, C. C. Carey, D. P. Hamilton, P. C. Hanson, J. S. Read, E. de Sousa, M. Weber, and L. A. Winslow, "A general lake model (GLM 3.0) for linking with high-frequency sensor data from the global lake ecological observatory network (GLEON)," *Geoscientific Model Development*, vol. 12, no. 1, pp. 473–523, Jan. 2019. [Online]. Available: https://gmd.copernicus.org/articles/12/473/2019/

[19] G. Evensen, "Sequential data assimilation with a nonlinear quasi-geostrophic model using Monte Carlo methods to forecast error statistics," *Journal of Geophysical Research: Oceans*, vol. 99, no. C5, pp. 10 143–10 162, May 1994. [Online]. Available: https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/94JC00572

[20] G. Petneházi, "Recurrent neural networks for time series forecasting," *ArXiv:1901.00069 [cs.LG]*, Jan. 2019. [Online]. Available: http://arxiv.org/abs/1901.00069

[21] H. Hewamalage, C. Bergmeir, and K. Bandara, "Recurrent neural networks for time series forecasting: Current status and future directions," *International Journal of Forecasting*, vol. 37, no. 1, pp. 388–427, Jan. 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0169207020300996

[22] A. Koochali, P. Schichtel, S. Ahmed, and A. Dengel, "Probabilistic forecasting of sensory data with generative adversarial networks - ForGAN," *ArXiv:1903.12549 [cs.LG]*, Mar. 2019. [Online]. Available: http://arxiv.org/abs/1903.12549

[23] Z. Luo, J. Chen, X. J. Cai, K. Tanaka, T. Takiguchi, T. Kinkyo, and S. Hamori, "Oil price forecasting using supervised GANs with continuous wavelet transform features," in *24th International Conference on Pattern Recognition*. Institute of Electrical and Electronics Engineers, Aug. 2018, pp. 830–835.

[24] J. Tan, G. Qi, D. Sun, W. Li, H. Yang, and Z. Li, "Forecasting renewable energy generation scenarios based on multi-agent diverse GANs," in *2020 Institute of Electrical and Electronics Engineers Sustainable Power and Energy Conference (iSPEC)*, Nov. 2020, pp. 180–186.

[25] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," *2017 Institute of Electrical and Electronics Engineers International Conference on Computer Vision (ICCV)*, pp. 2242–2251, Oct. 2017.

[26] L. Yang, S. Chou, and Y. Yang, "MidiNet: A convolutional generative adversarial network for symbolic-domain music generation using 1D and 2D conditions," *ArXiv:1703.10847 [cs.SD]*, Mar. 2017. [Online]. Available: http://arxiv.org/abs/1703.10847

[27] O. Kupyn, V. Budzan, M. Mykhailych, D. Mishkin, and J. Matas, "DeblurGAN: Blind motion deblurring using conditional adversarial networks," *2018 Institute of Electrical and Electronics Engineers/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8183–8192, Dec. 2018.

[28] C. C. Carey, Woelmer, Whitney M., A. S. L. Lewis, A. Breef-Pilz, D. W. Howard, and B. J. Bookout, "Time series of high-frequency sensor data measuring water temperature, dissolved oxygen, pressure, conductivity, specific conductance, total dissolved solids, chlorophyll a, phycocyanin, and fluorescent dissolved organic matter at

discrete depths in Falling Creek Reservoir, Virginia, USA in 2018-2020," 2021. [Online]. Available: https://portal.edirepository.org/nis/mapbrowse?packageid=edi.271.5

[29] C. C. Carey, A. S. L. Lewis, R. P. McClure, A. B. Gerling, S. Chen, A. Das, J. P. Doubek, D. W. Howard, M. E. Lofton, K. D. Hamre, and H. L. Wander, "Time series of high-frequency profiles of depth, temperature, dissolved oxygen, conductivity, specific conductivity, chlorophyll a, turbidity, ph, oxidation-reduction potential, photosynthetic active radiation, and descent rate for Beaverdam Reservoir, Carvins Cove Reservoir, Falling Creek Reservoir, Gatewood Reservoir, and Spring Hollow Reservoir in Southwestern Virginia, USA 2013-2020," 2021. [Online]. Available: https://portal.edirepository.org/nis/mapbrowse?packageid=edi.200.11

[30] C. C. Carey, A. G. Hounshell, M. E. Lofton, F. Birgand, B. J. Bookout, R. S. Corrigan, A. B. Gerling, R. P. McClure, and W. M. Woelmer, "Discharge time series for the primary inflow tributary entering Falling Creek Reservoir, Vinton, Virginia, USA 2013-2021," 2021. [Online]. Available: https://portal.edirepository.org/nis/mapbrowse?packageid=edi.202.7

[31] R. P. McClure, M. E. Lofton, S. Chen, K. M. Krueger, J. C. Little, and C. C. Carey, "Methane ebullition and diffusion rates, turbulence, water temperature, and water depth data from Falling Creek Reservoir (Virginia, USA) in the ice-free period during 2016-2019," 2020. [Online]. Available: https://portal.edirepository.org/nis/mapbrowse?packageid=edi.440.2

[32] C. C. Carey, A. Breef-Pilz, B. J. Bookout, M. E. Lofton, and R. P. McClure, "Time series of high-frequency meteorological data at Falling Creek Reservoir, Virginia, USA 2015-2020," 2021. [Online]. Available: https://portal.edirepository.org/nis/mapbrowse?packageid=edi.389.5

[33] C. C. Carey, "Ice cover data for Falling Creek Reservoir, Vinton, Virginia, USA for 2013-2021," 2021. [Online]. Available: https://portal.edirepository.org/nis/mapbrowse?packageid=edi.456.3

[34] S. B. Taieb, G. Bontempi, A. F. Atiya, and A. Sorjamaa, "A review and comparison of strategies for multi-step ahead time series forecasting based on the NN5 forecasting competition," pp. 7067–7083, Jan. 2012. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417412000528

[35] C. Chatfield, *Time-series forecasting*. Boca Raton, Florida: CRC press, 2000.

[36] J. Y. Li, S. Ambikasaran, E. F. Darve, and P. K. Kitanidis, "A Kalman filter powered by-matrices for quasi-continuous data assimilation problems," *Water Resources Research*, vol. 50, no. 5, pp. 3734–3749, Apr. 2014.

[37] S. F. Schmidt, "Computational techniques in Kalman filtering," *Theory and Applications of Kalman Filtering*, 1970, AGARDograph 139, Tech. Rep., NATO Advisory Group for Aerospace Research and Development.

[38] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the International Conference on Machine Learning*, 2010.

[39] C. G. Atkeson, A. W. Moore, and S. Schaal, *Locally Weighted Learning*. Dordrecht: Springer Netherlands, 1997, pp. 11–73. [Online]. Available: https://doi.org/10.1007/978-94-017-2053-3_2

[40] C. Esteban, S. L. Hyland, and G. Rätsch, "Real-valued (medical) time series generation with recurrent conditional GANs," *ArXiv:1706.02633 [stat.ML]*, Jun. 2017. [Online]. Available: https://arxiv.org/abs/1706.02633

[41] M. Mirza and S. Osindero, "Conditional generative adversarial nets," *ArXiv:1411.1784 [cs.LG]*, Nov. 2014. [Online]. Available: http://arxiv.org/abs/1411.1784

[42] Y. Kim, Y. Choi, D. Widemann, and T. Zohdi, "A fast and accurate physics-informed neural network reduced order model with shallow masked autoencoder," *ArXiv:2009.11990 [math.NA]*, Sep. 2020. [Online]. Available: https://arxiv.org/abs/2009.11990

[43] S. Gugercin and A. C. Antoulas, "A survey of model reduction by balanced truncation and some new results," *International Journal of Control*, vol. 77, no. 8, pp. 748–766, Apr. 2004. [Online]. Available: https://doi.org/10.1080/00207170410001713448

[44] A. Antoulas, D. Sorensen, and Y. Zhou, "On the decay rate of Hankel singular values and related issues," *Systems Control Letters*, vol. 46, no. 5, pp. 323–342, Mar. 2002. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167691102001470

[45] M. Pasini, "10 lessons I learned training GANs for a year," Jul. 2019. [Online]. Available: https://towardsdatascience.com/10-lessons-i-learned-training-generative-adversarial-networks-gans-for-a-year-c9071159628

[46] M. Arjovsky and L. Bottou, "Towards principled methods for training generative adversarial networks," *ArXiv:1701.04862 [stat.ML]*, Jan. 2017. [Online]. Available: https://arxiv.org/abs/1701.04862

[47] J. Hui, "GAN - ways to improve GAN performance," Jun. 2018. [Online]. Available: https://towardsdatascience.com/gan-ways-to-improve-gan-performance-acf37f9f59b

[48] O. Mogren, "C-RNN-GAN: continuous recurrent neural networks with adversarial training," *ArXiv:1611.09904 [cs.AI]*, Nov. 2016. [Online]. Available: http://arxiv.org/abs/1611.09904

[49] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder–decoder for statistical machine translation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1724–1734. [Online]. Available: https://www.aclweb.org/anthology/D14-1179

[50] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, pp. 1735–80, Dec. 1997.

[51] J. Chung, Ç. Gülçehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *ArXiv:1412.3555 [cs.NE]*, Dec. 2014. [Online]. Available: http://arxiv.org/abs/1412.3555

[52] H. M. D. Kabir, A. Khosravi, M. A. Hosen, and S. Nahavandi, "Neural network-based uncertainty quantification: A survey of methodologies and applications," *Institute of Electrical and Electronics Engineers Access*, vol. 6, pp. 36 218–36 234, Jun. 2018.

[53] J. Carney, P. Cunningham, and U. Bhagwan, "Confidence and prediction intervals for neural network ensembles," in *International Joint Conference on Neural Networks*, vol. 2, Aug. 1999, pp. 1215–1218 vol.2.

[54] Z. Lin, A. Jain, C. Wang, G. Fanti, and V. Sekar, "Using GANs for sharing networked time series data," *Proceedings of the ACM Internet Measurement Conference*, Oct. 2020. [Online]. Available: http://dx.doi.org/10.1145/3419394.3423643

[55] P. Stinis, T. Hagge, A. M. Tartakovsky, and E. Yeung, "Enforcing constraints for interpolation and extrapolation in generative adversarial networks," *Journal of Computational Physics*, vol. 397, p. 108844, Jul. 2019.

[56] J. Yoon, "TimeGAN code," GitHub, Jul. 2020, Accessed Jul. 2, 2020. [Online]. Available: https://github.com/jsyoon0823/TimeGAN

[57] X. Wu and Y. Wang, "Extended and unscented Kalman filtering based feedforward neural networks for time series prediction," *Applied Mathematical Modelling*, vol. 36, no. 3, pp. 1123 – 1131, Jul. 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0307904X11004501

[58] G. Zhang and M. Qi, "Neural network forecasting for seasonal and trend time series," *European Journal of Operational Research*, vol. 160, no. 2, pp. 501–514, Aug. 2003. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0377221703005484

[59] A. Koochali, "ForGAN code," GitLab, Jul. 2020, Accessed Jul. 2, 2020. [Online]. Available: https://git.opendfki.de/koochali/forgan

[60] M. Dhillon, "Virginia Tech pioneers smart reservoirs," Dec. 2020. [Online]. Available: https://roanoke.com/news/local/virginia-tech-pioneers-smart-reservoirs/article_b7b7330e-35a5-11eb-a443-b3f621f19df6.html

[61] D. van Ravenzwaaij, P. Cassey, and S. Brown, "A simple introduction to markov chain Monte–Carlo sampling," *Psychonomic Bulletin Review*, vol. 25, Mar. 2016.

[62] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *Journal of Computational Physics*, vol. 378, pp. 686–707, Feb. 2019.

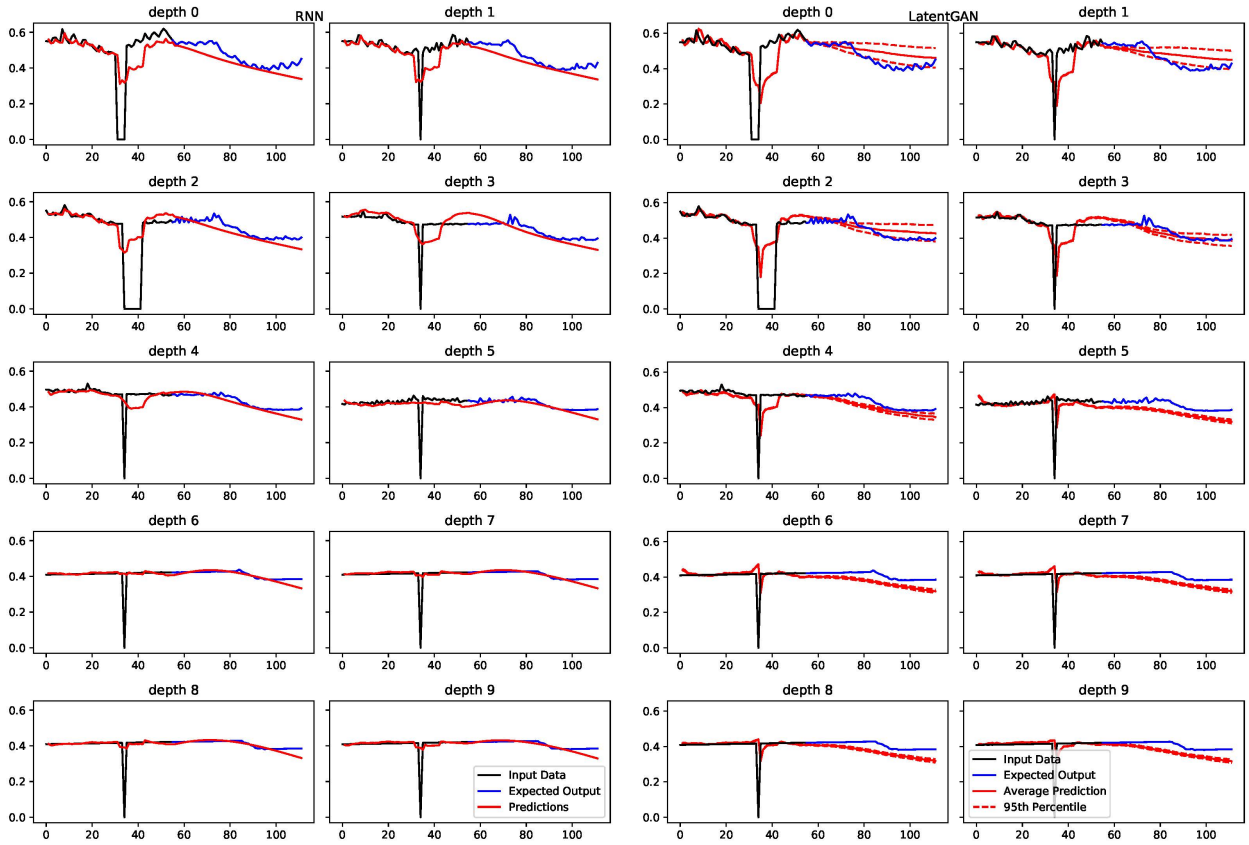
[63] Y. Yang and P. Perdikaris, "Adversarial uncertainty quantification in physics-informed neural networks," *Journal of Computational Physics*, vol. 394, pp. 136–152, May 2019.

[64] G. E. P. Box and D. A. Pierce, "Distribution of residual autocorrelations in autoregressive-integrated moving average time series models," *Journal of the American Statistical Association*, vol. 65, no. 332, pp. 1509–1526, Apr. 1970. [Online]. Available: https://amstat.tandfonline.com/doi/abs/10.1080/01621459.1970.10481180

Appendices

Appendix A

Fall Turnover Experiment



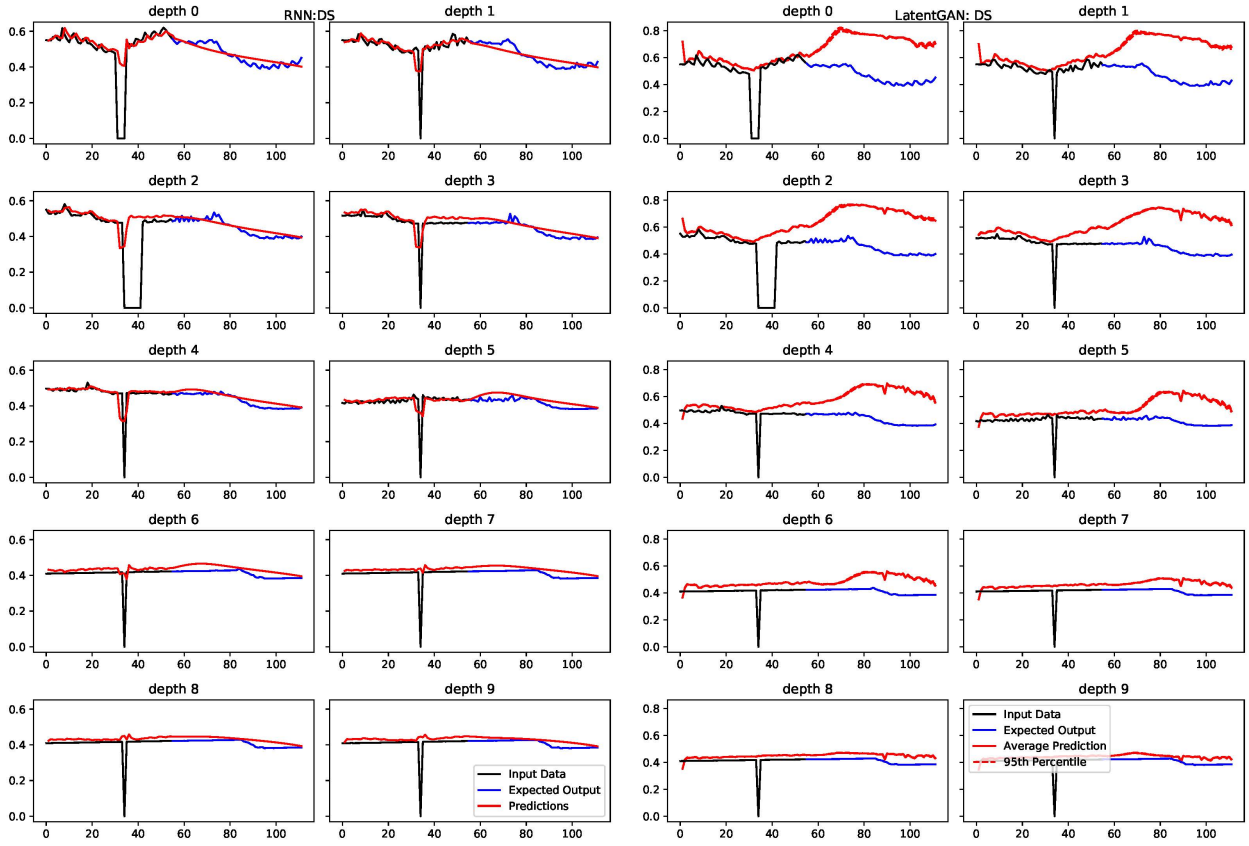
(a) Classical RNN 2 week predictions at all 10 depths of the water column.

(b) Classical LatentGAN 100 ensemble average 2 week prediction and 95% confidence interval at all 10 depths of the water column.

Figure A.1: RNN and LatentGAN 2 week predictions beginning on October 12, 2020 at 12:00 AM.

Model	SMAPE	MASE
RNN	4.75	1.65
LatentGAN	6.93	2.50

Table A.1: Table of error values corresponding to Figure A.1.



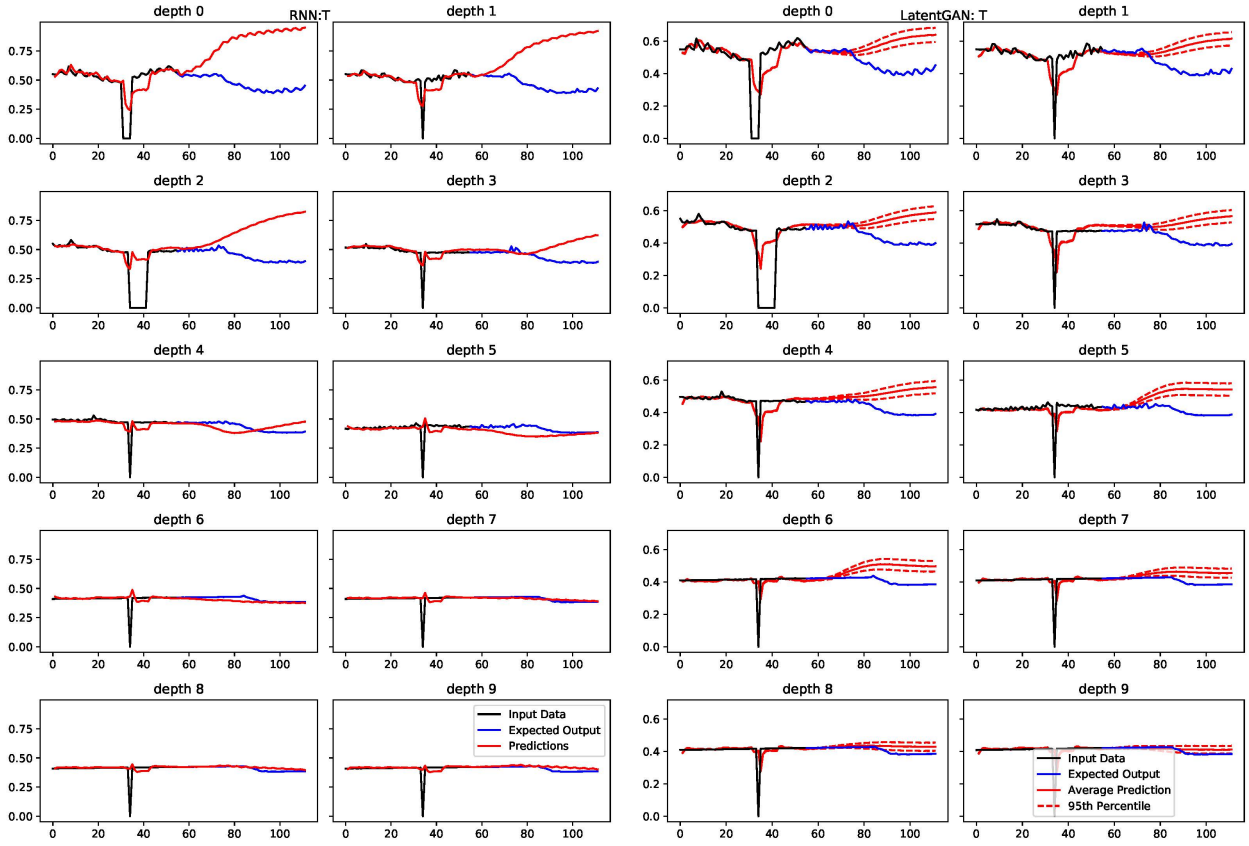
(a) RNN with data + DS input 2 week predictions at all 10 depths of the water column.

(b) LatentGAN with data + DS input 100 ensemble average 2 week prediction and 95% confidence interval at all 10 depths of the water column.

Figure A.2: RNN and LatentGAN with data + DS input 2 week predictions beginning on October 12, 2020 at 12:00 AM.

Model	SMAPE	MASE
RNN	4.54	1.72
LatentGAN	19.19	5.74

Table A.2: Table of error values corresponding to Figure A.2.



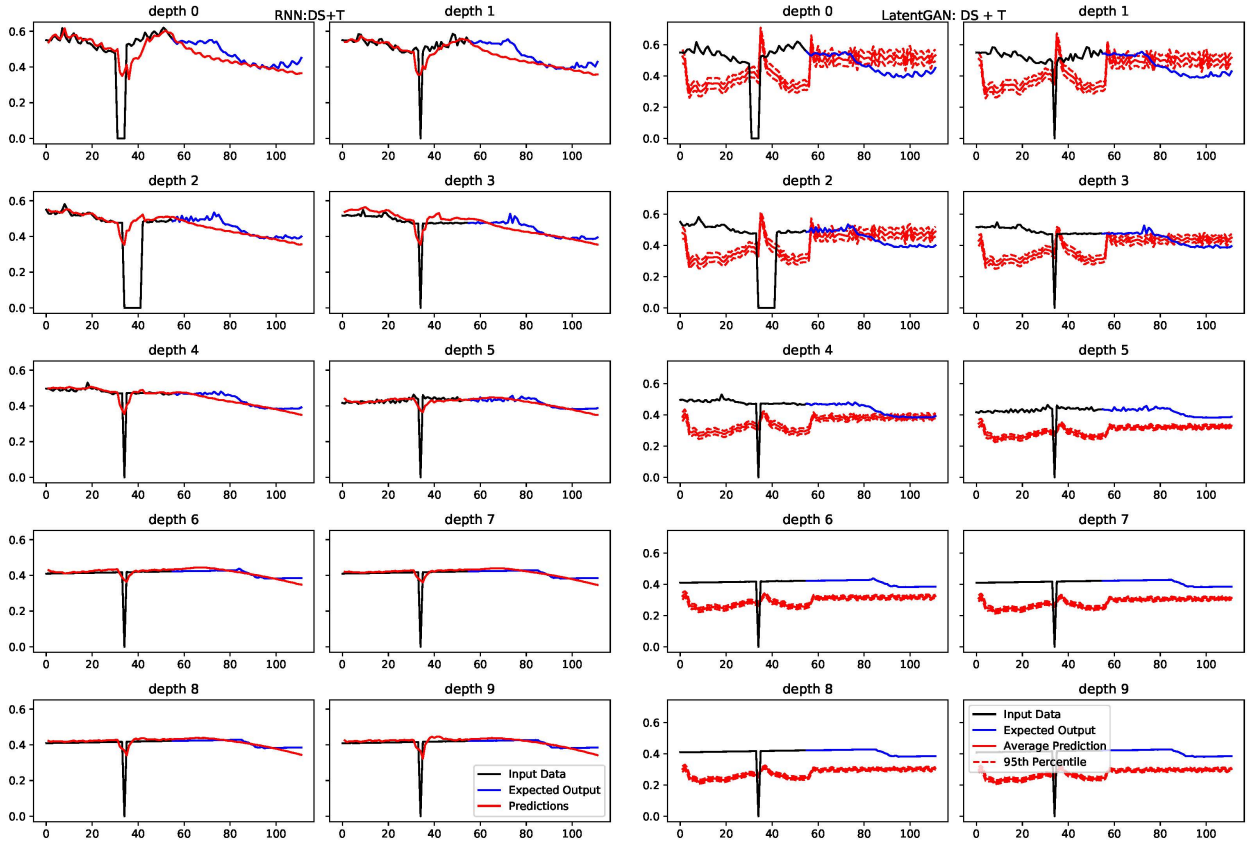
(a) RNN with data + T input 2 week predictions at all 10 depths of the water column.

(b) LatentGAN with data + T input 100 ensemble average 2 week prediction and 95% confidence interval at all 10 depths of the water column.

Figure A.3: RNN and LatentGAN with data + T input 2 week predictions beginning on October 12, 2020 at 12:00 AM.

Model	SMAPE	MASE
RNN	11.47	4.04
LatentGAN	9.86	4.09

Table A.3: Table of error values corresponding to Figure A.3.



(a) RNN with data + DS + T input 2 week predictions at all 10 depths of the water column.

(b) LatentGAN with data + DS + T input 100 ensemble average 2 week prediction and 95% confidence interval at all 10 depths of the water column.

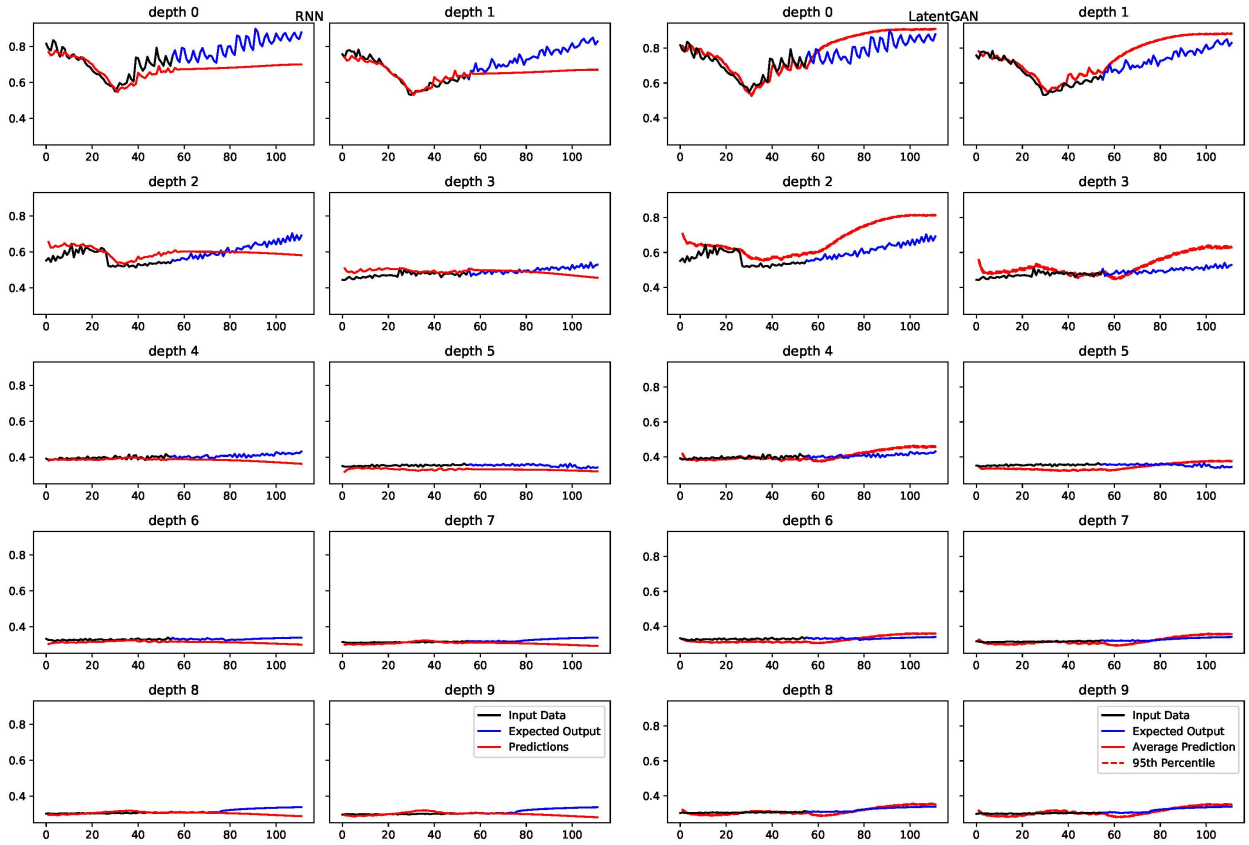
Figure A.4: RNN and LatentGAN with data + DS + T input 2 week predictions beginning on October 12, 2020 at 12:00 AM.

Model	SMAPE	MASE
RNN	4.07	1.46
LatentGAN	31.01	7.52

Table A.4: Table of error values corresponding to Figure A.4.

Appendix B

Summer Stratification Experiment



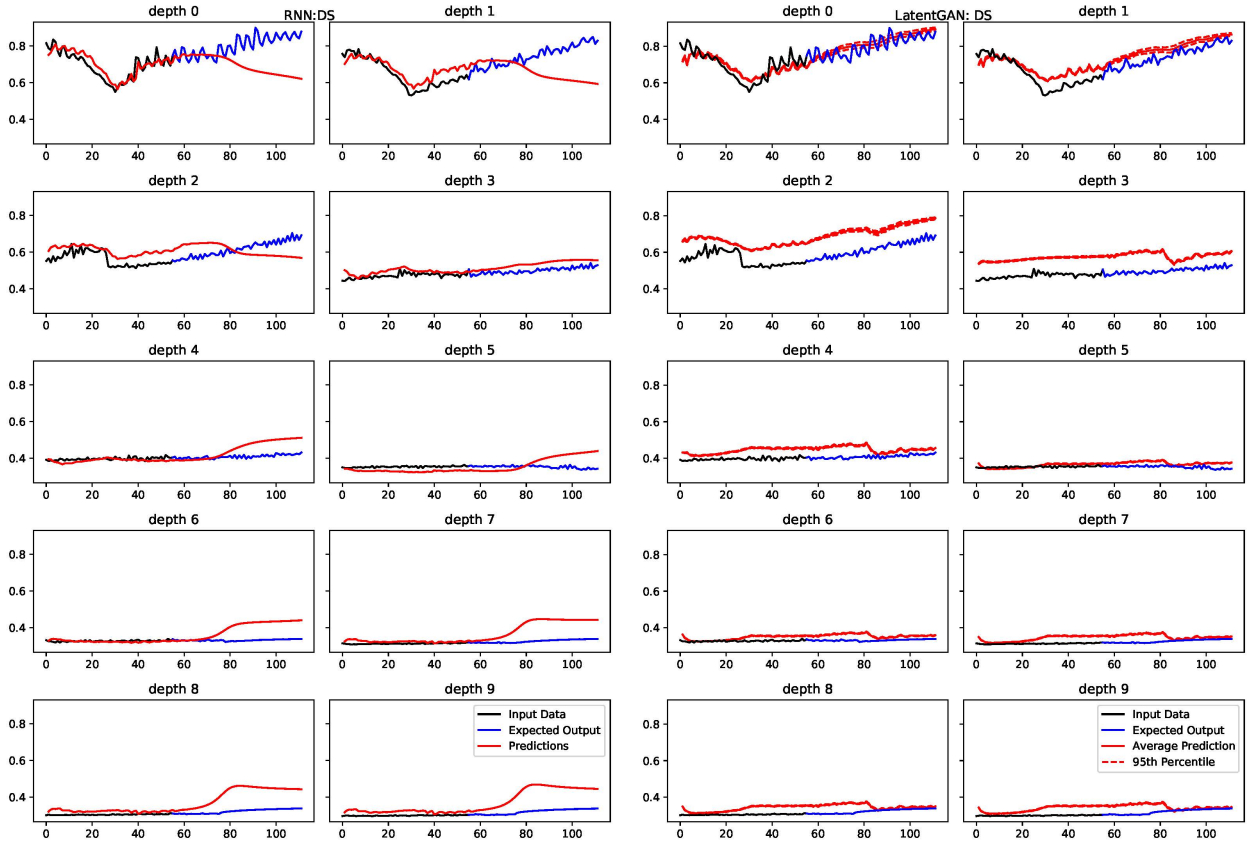
(a) Classical RNN 2 week predictions at all 10 depths of the water column.

(b) Classical LatentGAN 100 ensemble average 2 week prediction and 95% confidence interval at all 10 depths of the water column.

Figure B.1: RNN and LatentGAN 2 week predictions beginning on June 11, 2020 at 12:00 AM.

Model	SMAPE	MASE
RNN	6.10	13.12
LatentGAN	6.15	8.39

Table B.1: Table of error values corresponding to Figure B.1.



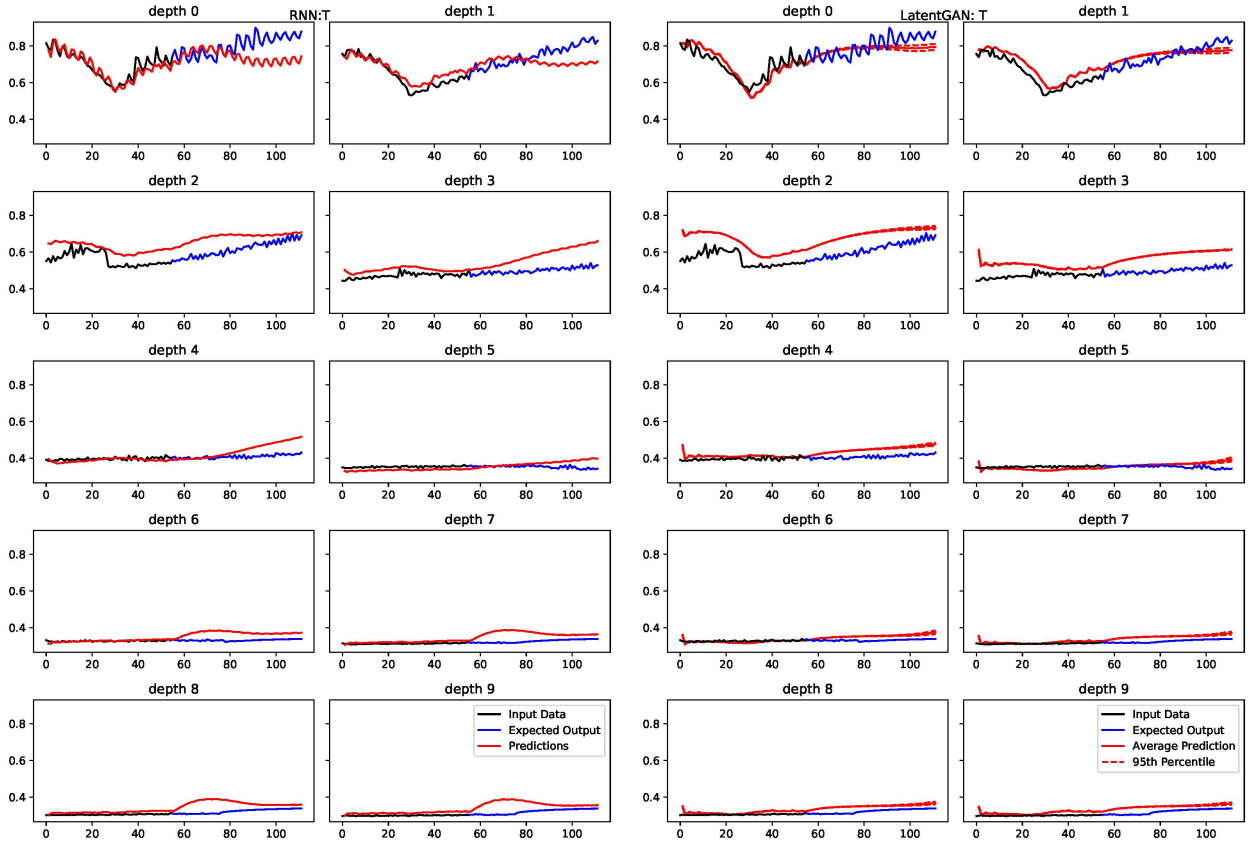
(a) RNN with data + DS input 2 week predictions at all 10 depths of the water column.

(b) LatentGAN with data + DS input 100 ensemble average 2 week prediction and 95% confidence interval at all 10 depths of the water column.

Figure B.2: RNN and LatentGAN with data + DS input 2 week predictions beginning on June 11, 2020 at 12:00 AM.

Model	SMAPE	MASE
RNN	10.35	13.88
LatentGAN	9.10	10.17

Table B.2: Table of error values corresponding to Figure B.2.



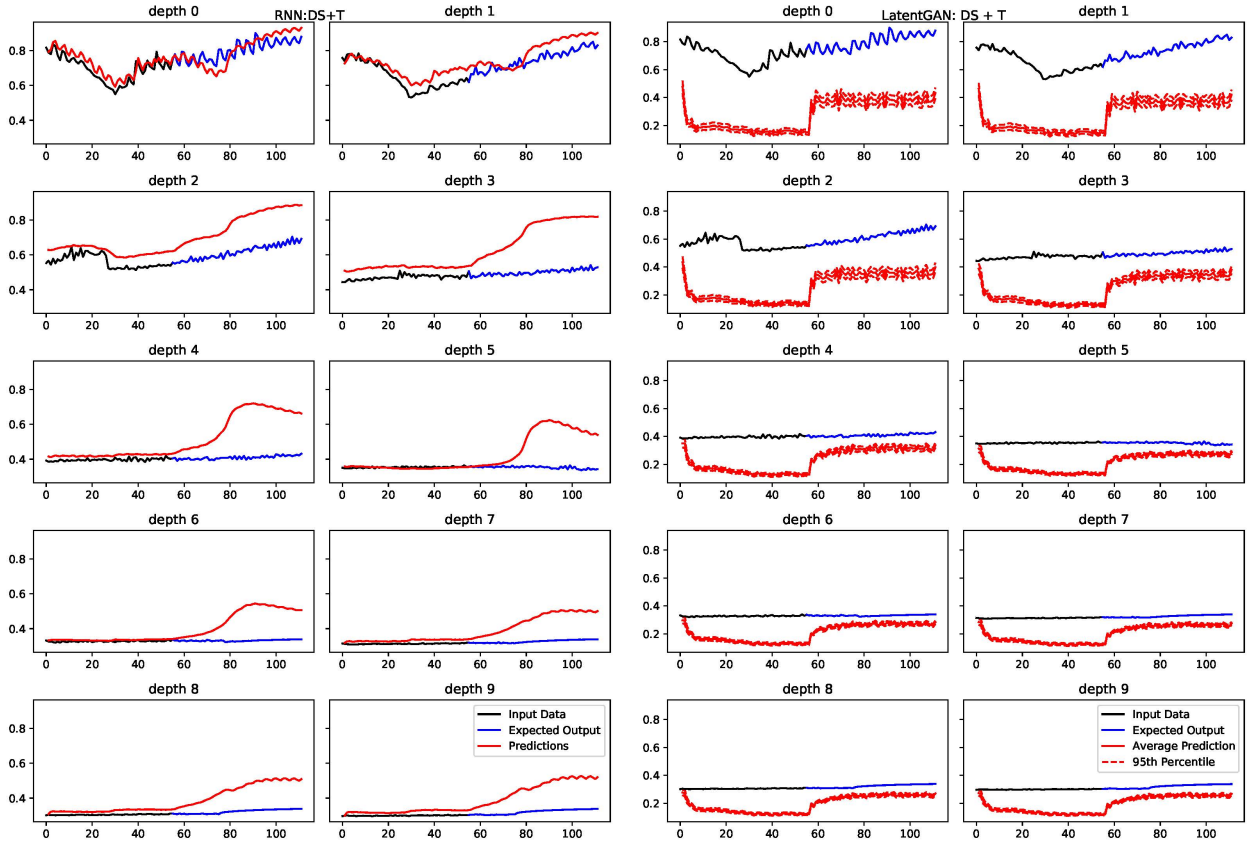
(a) RNN with data + T input 2 week predictions at all 10 depths of the water column.

(b) LatentGAN with data + T input 100 ensemble average 2 week prediction and 95% confidence interval at all 10 depths of the water column.

Figure B.3: RNN and LatentGAN with data + T input 2 week predictions beginning on June 11, 2020 at 12:00 AM.

Model	SMAPE	MASE
RNN	7.62	12.95
LatentGAN	6.90	11.85

Table B.3: Table of error values corresponding to Figure B.3.



(a) RNN with data + DS + T input 2 week predictions at all 10 depths of the water column.

(b) LatentGAN with data + DS + T input 100 ensemble average 2 week prediction and 95% confidence interval at all 10 depths of the water column.

Figure B.4: RNN and LatentGAN with data + DS + T input 2 week predictions beginning on June 11, 2020 at 12:00 AM.

Model	SMAPE	MASE
RNN	16.59	20.17
LatentGAN	65.52	17.92

Table B.4: Table of error values corresponding to Figure B.4.