Deep Learning Based Proteomic Language Modelling for in-silico Protein Generation

Nitin Kesavan Nair

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

Computer Engineering

Jianhua Xuan, Co-chair

Frank O. Aylward, Co-chair

A. Lynn Abbott

August 13, 2020

Blacksburg, Virginia

Keywords: Protein, Deep Learning, Recurrent Neural Network, Auto-Regressive Models

Copyright 2020, Nitin Kesavan Nair

Deep Learning Based Proteomic Language Modelling for in-silico Protein Generation

Nitin Kesavan Nair

(ABSTRACT)

A protein is a biopolymer of amino acids that encodes a particular function. Given that there are 20 amino acids possible at each site, even a short protein of 100 amino acids has 20^{100} possible variants, making it unrealistic to evaluate all possible sequences in sequence level space. This search space could be reduced by considering the fact that billions of years of evolution exerting a constant pressure has left us with only a small subset of protein sequences that carry out particular cellular functions. The portion of amino acid space occupied by actual proteins found in nature is therefore much smaller than that which is possible [24]. By examining related proteins that share a conserved function and common evolutionary history (heretofore referred to as protein families), it is possible to identify common motifs that are shared. Examination of these motifs allows us to characterize protein families in greater depth and even generate new "in silico" proteins that are not found in nature, but exhibit properties of a particular protein family. Using novel deep learning approaches and leveraging the large volume of genomic data that is now available due to high-throughput DNA sequencing, it is now possible to examine protein families in a scale and resolution that has never before been possible. By using this abundance of data to learn high dimensional representations of amino acids sequences, in this work, we show that it is possible to generate novel sequences from a particular protein family. Such a deep sequential model-based approach has great value for bioinformatics and biotechnological applications due to its rapid sampling abilities.

Deep Learning Based Proteomic Language Modelling for in-silico Protein Generation

Nitin Kesavan Nair

(GENERAL AUDIENCE ABSTRACT)

Proteins are one of the most important functional biological elements. These are composed of amino acids which link together to form different shapes which might encode a particular function. These proteins may act independently or might form "complexes" to have a particular function. Therefore, understanding them is of utmost importance. Due to the fact that there are 20 amino acids even a protein sequence fragment of length 5 can have more than 3 million different combinations. Given, that proteins are generally 1000 amino acids long, looking at all the possibilities is next to impossible. In this work, by leveraging the "deep learning" paradigm and the vast amount of data available, we try to model these proteins and generate new proteins belonging to a specific "protein family." This approach has great value for bioinformatics and biotechnological applications due to its rapid sampling abilities.

Acknowledgments

I would like to firstly express my sincerest gratitude to my advisor Dr. Frank Aylward. His constant guidance and support has been key in the completion of this work. I would also like to thank Dr. Jason Xuan and Dr. Lynn Abbott for serving on my committee as Co-Chair and member respectively. Their support and guidance always served as a motivation to strive for better. I would also like thank my labmates whose love for science has instilled in me a deep appreciation and curiousness for the biological sciences.

I'm thankful to the Department of Biological Sciences for hosting me and for the Bradley Department of Electrical and Computer Engineering for allowing me to pursue projects that were collaboratory in nature. I would also like to express my gratitude to the Virginia Tech Graduate School for their continued assistance.

Finally, I am ever thankful to my family without whose support and confidence, pursuing a graduate degree would not have been possible.

Contents

List of Figures ix

List of Tables xii

1 Introduction 1

1.1 Motivation 1

1.2 Contribution 2

2 Background 4

2.1 Deep Neural Networks 4

2.2 Sequential Models 4

2.2.1 RNN 5

2.2.2 LSTM 6

2.2.3 GRU 7

2.2.4 Generative Adversarial Network 7

2.3 Model Architectures 8

2.3.1 Char-RNN 8

2.3.2 Transformer 8

2.3.3 BERT 10

2.3.4 GPT-2 11

2.3.5 Reformer 11

2.3.6 styleGAN 13

2.4 Loss Function 13

2.4.1 MSE 14

2.4.2 Cross-Entropy 14

2.5 Optimizers 14

2.5.1 Adam 14

2.5.2 AdamW 15

2.5.3 Adamax 15

2.6 Sampling Techniques 15

2.6.1 Top-K 16

2.6.2 Top-P 16

2.6.3 Gumbel-Softmax 16

3 Methodology **17**

3.1 Complete Genome Generation 17

3.1.1 Tokenizer 17

3.1.2 GPT-2 18

3.1.3 BERT 19

3.1.4 styleGAN 19

3.2 Protein Family Generation 19

3.2.1 Data Sampling 20

3.2.2 Tokenizer 21

3.2.3 Evaluation Strategy 21

3.2.4 char-RNN 22

3.2.5 char-RNN-taxo 23

3.2.6 char-RNN-gumbel 25

3.2.7 Reformer 25

4 Results and Discussion **27**

4.1 Complete Genome Generation 27

4.1.1 GPT-2 27

4.1.2 styleGAN 28

4.1.3 BERT 30

4.2 Protein Family Generation 31

4.2.1 char-RNN 32

4.2.2 Reformer 33

4.2.3 char-RNN-taxo 35

4.2.4 char-RNN-gumbel 36

4.2.5 char-RNN-gumbel-taxo 37

5 Summary and Future Work **51**

Bibliography **52**

Appendix A First Appendix **59**

A.1 Train-Validation Plots 59

List of Figures

2.1 RNN 5

2.2 Vanilla LSTM 7

2.3 LSTM Equations 7

2.4 GRU Equations 8

2.5 Char-RNN Architecture 9

2.6 Transformer Architecture 10

2.7 styleGAN Generator Architecture 13

3.1 Sample pre-processed genome 18

3.2 Codon to Color Mapping 20

3.3 UMAP plot of network embeddings of GTDB terms found using Node2Vec . 24

3.4 Char-RNN-taxo Architecture 25

3.5 Gumbel-Softmax distribution (Source: Figure 1 [20]) 26

4.1 GPT-2 Sample Output 28

4.2 Samples from styleGAN Generator 29

4.3 styleGAN Sample Output 30

4.4 BERT Training Statistics 30

4.5 Example of a generated protein using 2L_LSTM 33

4.6 BLASTp of generated protein shown in Fig 4.5 34

4.7 Amino Acid Embeddings learnt by 2L_LSTM 35

4.8 Amino Acid Similarity Based on Physio-Chemical Properties (Source: Figure 1a [30]) 36

4.9 2L_LSTM: Phylogenetic Tree of Natural and Synthetic Sequences 37

4.10 Phylogentic Tree Creation 38

4.11 2L_LSTM: t-SNE plot of both natural and synthetic sequence embedding . 39

4.12 Example of generated protein using Reformer model 39

4.13 BLASTp of generated protein shown in Fig 4.12 40

4.14 BLASTp of generated protein shown in Fig 4.12 41

4.15 Amino Acid Embeddings learnt by the Reformer model 42

4.16 Reformer: Attention Visualization at conserved motif NADFDG 43

4.17 Example of generated protein using 2L_LSTM_taxo 44

4.18 BLASTp of generated protein shown in Fig 4.17 44

4.19 Amino Acid Embeddings learnt by 2L_LSTM_taxo 45

4.20 2L_LSTM_taxo: Phylogenetic Tree of Natural and Synthetic Sequences . . 45

4.21 Example of generated protein using 2L_LSTM_Gumbel 46

4.22 BLASTp of generated protein shown in Fig 4.21 47

4.23 Amino Acid Embeddings learnt by 2L_LSTM_Gumbel 47

4.24 2L_LSTM_Gumbel: Phylogenetic Tree of Natural and Synthetic Sequences 48

4.25 Example of generated protein using 2L_LSTM_Gumbel_Taxo 48

4.26 BLASTp of generated protein shown in Fig 4.25 49

4.27 Amino Acid Embeddings learnt by 2L_LSTM_Gumbel_Taxo 49

4.28 2L_LSTM_Gumbel_Taxo: Phylogenetic Tree of Natural and Synthetic Sequences 50

A.1 2L_LSTM: Train-Validation Loss Plot 59

A.2 Reformer: Train-Validation Loss Plot 60

A.3 2L_LSTM_taxo: Train-Validation Loss Plot 61

A.4 2L_LSTM_Gumbel: Train-Validation Loss Plot 62

A.5 2L_LSTM_Gumbel_Taxo: Train-Validation Loss Plot 63

List of Tables

4.1 Test Loss 37

Chapter 1

Introduction

1.1 Motivation

Proteins are biopolymers composed of amino acids. These amino acids have different electrochemical properties due to which changing even one amino acid in the protein could result in a different resultant protein structure which may affect its original function. Proteins in both prokaryotic and eukaryotic systems play a wide range of functions. This may range from translation to transportation of materials. Therefore, understanding them is of utmost importance. Due to the decrease in the cost of DNA sequencing and the rise in its throughput, it is now possible to look at the world in a resolution that up until recently has been impossible. This abundance of data makes it possible to use the deep learning paradigms which have shown great promise in computer vision and NLP. The data-driven paradigm helps avoid the need to find good protein candidates in the protein sequence space. This is necessary as even a short protein of 100 amino acids has 20^{100} possible variants, making it unrealistic to evaluate all possible sequences in protein sequence space. Also, due to the fact that evolution has been acting on these sequences, only a subset of these combinations in the 'sequence space' would even be viable given the multiple external factors that it encounters [24].

But producing such de novo biological sequences are extremely desirable due to its numerous applications. For example, due to the fact that proteins are key catalysts in biological reac-

tions, the ability to generate new genes or proteins belonging to a known class would have variety of applications in numerous fields like enzymology, industrial methods, and medicine. Indeed, various methods do exist currently which helps generate such novel protein [4] (this method won a nobel prize for its contribution). This work, focuses on using generative machine learning models to produce novel biological sequence variants; the advantage of these approaches is that they leverage the large amount of research that has been performed in recent decades to develop deep learning models, which are effective at capturing complex patterns that may elude detection using manual curation or traditional bioinformatic methods. Several approaches have recently used generative machine learning methods to generate DNA sequences that bind to protein [19], sequences that contain splice sites or known regulatory motifs [29], and even antibodies [45]. In this work, by examining related proteins that share a conserved function and common evolutionary history (heretofore referred to as protein families), I show that it is possible to identify common motifs that are shared. This enables the deep learning model to generate "in silico" proteins that have not yet been found in nature, and provides useful benchmarking that will inform future studies seeking to generate novel biological sequences.

1.2 Contribution

In this work I:

- Build generative deep learning models, such as GPT and BERT, to generate complete genomes at the nucleotide level. These efforts were useful in my initial efforts to build generative machine learning models of biological sequences, but they were ultimately not successful owing to the large size of genomes and their complexity.

- I then moved on to design generative deep learning models of protein sequences, since these datasets are substantially smaller. Here I built autoregressive models based on RNNs and transformers to generate proteins belonging to a particular family. These models were successful in generating biologically-feasible protein sequences, as illustrated by their successful inclusion in phylogenetic and homology-based analyses that are typically used in bioinformatic protein analysis.

- Test if network embeddings of taxonomy could be used to generate protein sequences from a particular phylum. This was successful in that proteins from a given bacterial phylum could be successfully generated to an extent.

- Evaluate the properties of the amino acid embeddings learnt. This analysis demonstrated that the models successfully learned key properties of amino acids, as demonstrated by their recapitulation of patterns observed in the well-known physiochemical properties of amino acids.

- Evaluate the interpretability of attention-based models to understand common motif generation. Through the visualization of attention around the common motifs (which the model learnt to conserve highly like in the natural sequences) I revealed that local information is mostly what the model uses to decide the amino acids in the motifs.

Chapter 2

Background

2.1 Deep Neural Networks

Inspired by brain neurons, [52] put forward a computational model of the neuron. This soon became the basis of various machines/classifiers like the [46]. Although it was shown to not have the ability to model non-linear functions (could only model linearly separable functions), [46] later inspired a multi-layer version called the multi-layer perceptron which did have that ability. In the past decade, owing to the leap in parallel computing capability, various deep learning based networks have shown great promise in various fields like computer vision and natural language processing. Various methodological innovations in the field such as transfer learning have allowed wide spread usage of these networks due to them having learnt good reusable representations. This scaling up of hardware resources and the data digested having shown exceptional performance in CV and NLP has immense potential for use in unravelling the mysteries in biology.

2.2 Sequential Models

Sequence data is ubiquitous. Therefore, building models that will help us get an understanding of their sequential compositionality is a priority. There exist multiple computational models that vary drastically in their receptive field, the ease of training, reusability and the

resources needed to train. In this work we focus on using deep learning based sequential models. Such models allow us to avoid explicit feature engineering.

2.2.1 RNN

Leveraging the idea of parameter sharing, sequential models like RNNs are capable of modelling sequences of varying lengths as they share the same parameters across time-steps. They can also model very long sequences without taking the penalty of larger and larger parameter sizes as they only keep a single state vector that updates itself as a representation of the past. Although some networks like ones based on 1D convolutions can also apply parameter sharing, the "recurrent" formulation allows for long range dependencies larger than its counterparts.

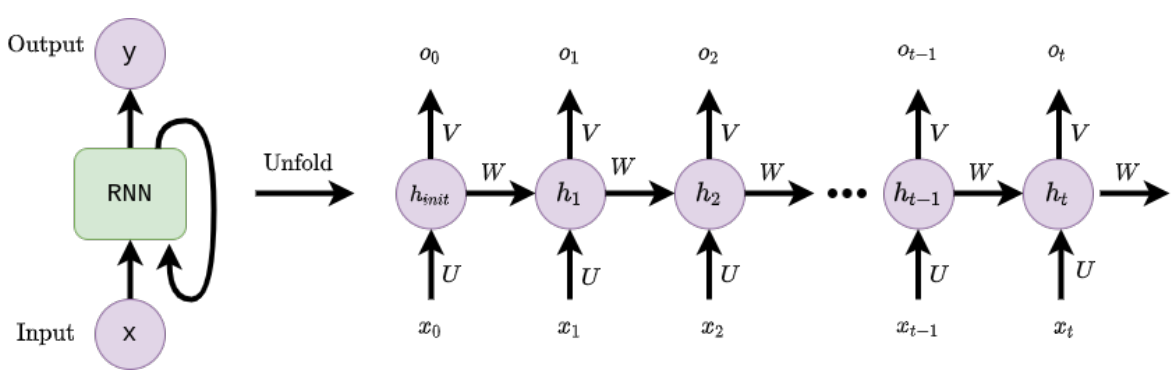


Figure 2.1: RNN Unrolled with respect to time. U is the input-hidden weight matrix, W is the hidden-hidden weight matrix and V is the hidden-output weight matrix

The update equation of the RNN is given below.

$$
\begin{aligned}
h_t &= f\left(Ux_t + W_{t-1}\right) \text{ where, } f \text{ is a non-linear function} \\
o_t &= Vh_t \\
h_{t+1} &= Wh_t \\
h_{init} &= \mathbf{0}
\end{aligned}
\tag{2.1}
$$

Although hidden layers are initialized to zeros, other methods like random initialization and so forth exist [59] which could better the performance of the RNN model. Care has to be taken during training to avoid gradient explosion (proper gradient clipping) and vanishing gradient problem. These issues are why better formulations of RNNs like the ones given below are formulated.

2.2.2 LSTM

Long Short Term Memory (LSTM) is a type of RNN designed to alleviate the long-term dependency issues usually found in vanilla RNNs. It achieves so by using gating mechanisms to allow information to pass through long paths without the gradient vanishing using the cell state matrix. The LSTM uses multiple gates to decide to either conserve the data or modify it. 'Forget gate layer' is used to decide what information in the cell state path is to be forgotten (f_t). The 'input gate layer'(i_t), computes the new value that will be written in the particular states that are to be rewritten which are determined by the new cell state vector $\widetilde{C}_t$. The 'output gate layer' (o_t) along with the new cell state C_t computed from $\widetilde{C}_t$ are then used to determine the hidden state.

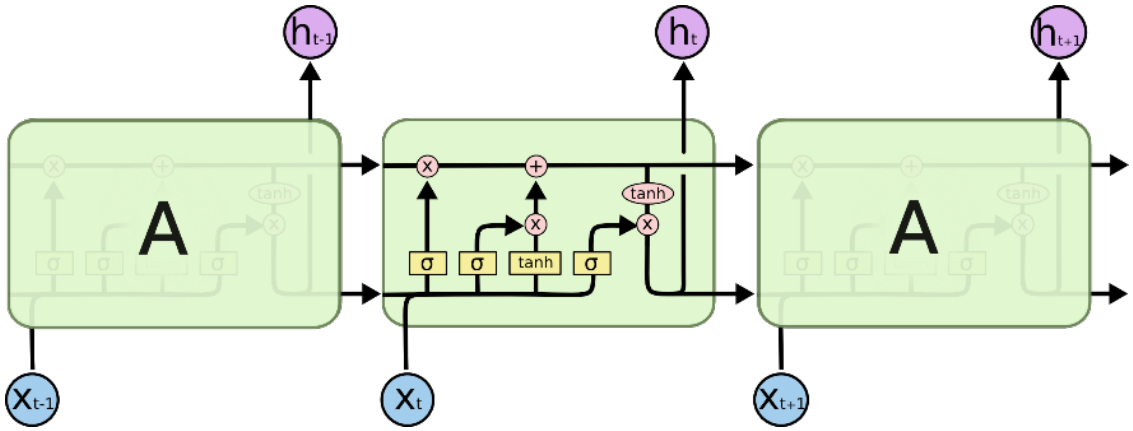


Figure 2.2: Vanilla LSTM (Source: [38])

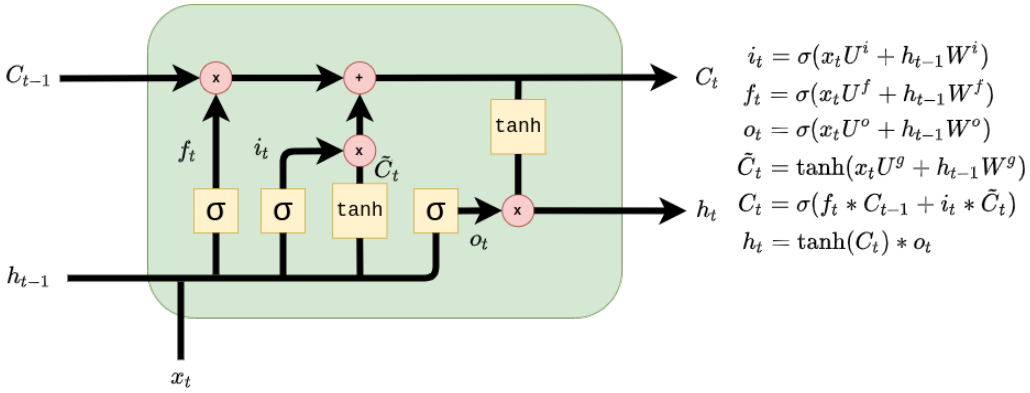


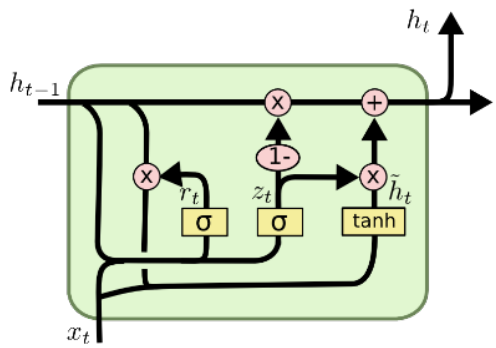
Figure 2.3: LSTM Update Equations

2.2.3 GRU

Gated Recurrent Unit (GRU) is a variant of the LSTM which combine forget and input gates as seen in LSTMs to a single "update gate". This unit only merges the cell state and the hidden state resulting in a simpler model with certain advantages as given below.

2.2.4 Generative Adversarial Network

Generative Adversarial Networks (GANs) [12] are a class of generative models which are trained in an 'adversarial' manner. This means that the adversary (discriminator) is tasked



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Figure 2.4: GRU Update Equations (Source: [38])

with predicting if the input to it is 'fake' or 'real' while the 'generator' keeps trying to fool the discriminator by producing outputs as close as the 'real' ones (thereby learning the distribution of the target objects).

2.3 Model Architectures

2.3.1 Char-RNN

These RNNs if trained properly are considered to be turing-complete [50], due to which they have an "unreasonable effectiveness" to generate any arbitrary "programs" [21]. Therefore, they are an excellent candidate to generate proteins at the amino acid level as shown in 2.5. The goal of this model is to keep predicting the next amino acid, thereby learning the "program" which could generate certain proteins.

2.3.2 Transformer

Although certain RNN based models theoretically should have excellent performance in modelling long sequences, they fail to do so in practice [40]. Also, although transfer learning

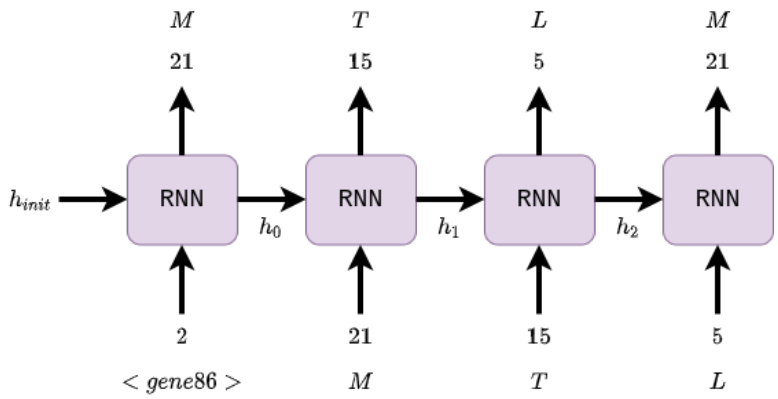


Figure 2.5: Char-RNN Architecture

paradigms have revolutionized the computer vision domains, RNN based works in NLP have not been able to produce representations that are transferable. This has created a need to introduce a class of model that is robust to train and more importantly has the ability to scale/parallelize, which RNN-based models lag in inherently due to their design. Attention-based models like transformers [56] allow exactly this while also allowing interpretability, something early deep learning-based models did not have. Also, transformers due to the way it is constructed is not tasked with encoding all of the information onto the hidden vector. This allows it to learn and keep more unique features with ease.

The attention mechanism present in the transformer model is tasked with mapping a query (Q) and a set of key-value pairs (K, V) to an output. These vectors once operated on as shown in 2.2.

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}}) \tag{2.2}$$

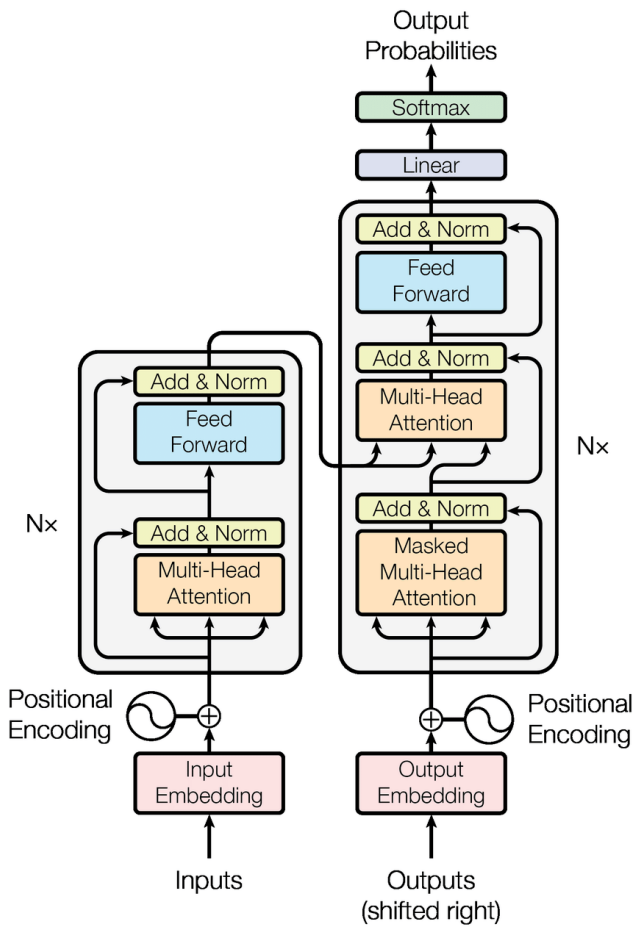


Figure 2.6: Transformer Architecture (Source: Figure 1 [56])

2.3.3 BERT

Bidirectional Encoder Representations from Transformers (BERT) [8] is a language model that is based on the transformer architecture as explained in Section 2.3.2. BERT, using the "cloze task" [54], builds a model called masked language model (MLM). This MLM objective in BERT allows it to condition words on both left and right context without indirectly leaking the information. This task also allows for the multiple transformer encoders to train well. Apart from MLM, BERT also has a next sentence prediction task it optimizes for. These together allow for pretrained-BERT to perform well in task specific objectives such as question answering, sentence pair classification, and sentence tagging with just one

additional output layer achieving state-of-the-art models during the release of [8]. The pretrained BERT model apart from its use in fine-grained tasks as mentioned above can also be used as a way of getting contextualized word embeddings. Depending on the task, different combinations of the hidden vectors along the path of the token can be used as the contextual word embedding.

2.3.4 GPT-2

GPT-2 [43] is a language model trained by OpenAI. This model unlike BERT (Section 2.3.3) uses just the transformer decoder blocks (BERT only uses the encoder block). GPT-2 unlike BERT is an auto-regressive model (similar to RNNs) whose objective is to predict the next token given just the left context.

2.3.5 Reformer

Vanilla transformers, although shown to work exceptionally well, are seriously restricting with respect to pretraining from scratch due to their resource requirements. This is especially true in cases where the sequences are long. This is due to the fact that self-attention computation in the transformers is quadratic with respective to sequence length [56]. This means that more efficient transformer implementations are needed. Reformers [26] are one such efficient implementation. Reformer achieves this efficiency and performance through the following steps:

- Better self-attention formulation
  - Local Self-Attention: This allows for reducing the asymptotic memory complexity of the inner dot-product $\boldsymbol{QK}^T$ from $\mathcal{O}(n^2)$ to $\mathcal{O}(n * l_c)$ where n is the sequence

length and l_c is the chunk length.

  - LSH Self-Attention: Uses [3] to simulate global self-attention in an efficient manner. This exploits the fact that the softmax applied to $\boldsymbol{QK}^T$ only weighs few 'value vectors' with weights significantly larger than 0 for each 'query vector'. Therefore computing over the keys with high cosine similarity with the query is needed.

- Chunking feed-forward layers: The transfomer architecture generally has two feed-forward (ff) layers following the self-attention layer. Here, the intermediate embedding size of the ff layers is usually bigger than the hidden size of the model. Trading memory for speed [26] employs chunking the linear layers computation processing only one chunk at a time. This avoids having to store the whole intermediate tensor, thereby drastically reducing the memory footprint.

- Reversible residual layers: Uses reversible residual layers from [11] which allows for an implementation where storing in the activation of the forward pass is not needed, thereby saving memory during training.

- Axial positional encodings: To avoid having to store large positional encoding matrices, this positional encoding strategy stores it as two factorized embedding vectors.

Although the best efficient transformer implementation during this work's implementation was [26] recent works have shown better implementations like [57][53] and few have even shown these linear transformer's relationship to RNNs [23].

2.3.6 styleGAN

The styleGAN model [22] improves the generator capability by having the ability to automatically learn attributes at different levels of scale (fine to coarse grained). The generator design proposed in [22] allows for better disentanglement of the latent factors of variations.

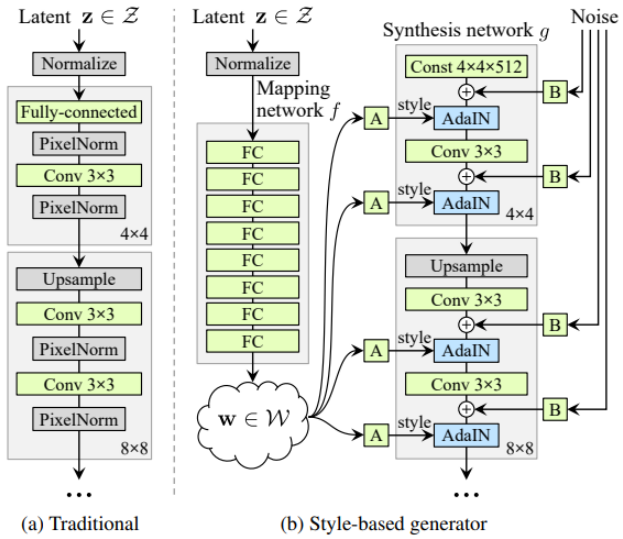


Figure 2.7: styleGAN Generator Architecture (Source: Figure 1 [22])

2.4 Loss Function

Neural networks learn to map the given inputs to the possible outputs. Therefore, when trained using gradient-based methods a mathematical representation of the "cost" incurred in predicting is needed. This is termed as the loss function. Various loss functions exist given the job at hand. A few loss functions which are used in this work are discussed below.

2.4.1 MSE

Mean squared error (MSE) measures squared L2 norm between the actual value and the estimated value with respect to the parameters of the model.

2.4.2 Cross-Entropy

This loss function allows us to measure the difference in the probability distribution over the true label and the predicted probability by the model.

2.5 Optimizers

The goal of the optimizer is to find the gradient of training samples with respect to the loss (Section 2.4) computed. The gradients are calculated in an iterative fashion so as to find the global optimum point in the non-convex loss landscape. The gradient computed here allows for back-propagation [47] to minimize the loss of the neural network model. The selection of the best optimizer for the task at hand is a tricky one. This is because the performance of the optimizer, although task-aware, is also highly dependent on the hyperparameter selection budget [5]. Various optimizers exist, a few of which are explained below.

2.5.1 Adam

Adam [25] is a first-order gradient-based optimizer. This optimizer can be looked at as a combination of RMSprop [55] and stochastic gradient descent with momentum [41]. This optimizer allows for both fast changing weights of certain parameters and the more slow changing weights of certain others to change accordingly to find the optimal weight. More

importantly this optimizer compared to AdaGrad [9][35] and RMSprop [55] offered significantly lower training time. Although soon shown to be not robust [58], Adam fell out of favor until various bugs like weight decay implementation bug [31] and update rule bug [44] (allowing for optimal point convergance) were discovered. The performance of this optimizer depends highly on the hyperparameter choice.

2.5.2 AdamW

The objective of this optimizer is to decouple the weight decay from the gradient-based update in Adam [31]. Due to the fact that weights are changed adaptively, L2 regularization formulation in Adam [25] leads to "weights with large historic parameter and/or gradient amplitudes being regularized less". Therefore, decoupling the weight decay from the gradient-based update is needed. This is the major reason for the formulation of AdamW.

2.5.3 Adamax

This is a variant of Adam based on the infinity norm [25]. This optimizer is useful when learning sparsely updated parameters like embeddings.

2.6 Sampling Techniques

The output of the models (neural language modeling task) give a distribution over the labels. Therefore, it is necessary to select one from the multinomial distribution given. Depending on the strategy of "selection", the quality of the resultant output of the generative models varies drastically.

2.6.1 Top-K

In this method, a selection is made from the top 'k' probability values. The value 'k' is a hyperparameter that has to found.

2.6.2 Top-P

In this strategy called "nucleus sampling" (or top-p) the goal is to "sample from the dynamic nucleus of tokens containing the vast majority of the probability mass" [17]. This allows for omitting the tail end of the distribution. This strategy also allows for avoiding repetitive loops usually found in text generation.

2.6.3 Gumbel-Softmax

This strategy is slightly different from the previous two. Here, the parameters needed for selection strategy are learnt with the probability distribution. Gumbel-softmax can be looked at as a differentiable formulation of argmax. Such a strategy means the objective performed both at training and inference is as close as possible.

Chapter 3

Methodology

3.1 Complete Genome Generation

In these experiments the goal was to see if it is possible to learn how to generate in-silico complete genomes at the nucleotide sequence level. A selection of more than 500 complete genomes of bacteria and archaea representing a broad phylogenetic range was obtained from NCBI. These genomes were then run through Prodigal [18] to predict all the protein-coding regions in the genomes. This information was used to tag the intergenic regions (start & stop), gene regions (start & stop) as well as the whole genome region (start & stop) as shown in Fig 3.1. These tags are custom tokens defined in the tokenizer as explained in Section 3.1.1.

3.1.1 Tokenizer

The nucleotide sequence has to be first converted to a list of integers for it to be used in all the methods given below due to the way the loss function is defined. To achieve this, a tokenizer is built using sentencepiece [28]. The unigram model [27] is used to create a vocabulary of size 100. The tokenizer selects the best possible k-mers of various sizes in an unsupervised fashion. The tokenizer is also configured with custom symbols which denote the following:

```
<genome>
<inter>GTTGA....</inter>
<gene>ATG.......</gene>
<inter>GATGA....</inter>
<gene>ATG.......</gene>
.
.
.
</genome>
```

Figure 3.1: Sample pre-processed genome

- Genome start and stop ('<genome>', '</genome>')
- Gene start and stop ('<gene>', '</gene>')
- Intergenic region start and stop ('<inter>', '</inter>')

The nucleotide sequences with the custom tags are given as input to the tokenizer, both during the training of tokenizer and during the usage of the tokenizer in the experiments described below. The locations of the tags are determined with Prodigal as explained in Section 3.1.

3.1.2 GPT-2

A GPT-2 model as explained in Section 2.3.4 is trained using the 'transformer-lm' repository (https://github.com/lopuhin/transformer-lm/commit/fa3f529ff300a30cd984ea72a4e23b525b6e3f52). The specific model trained was the GPT-2 (small) which has 117M parameters. The model specifically has a vocab size of 100, 12 layers, 768 as the embedding size, 12 attention heads and 1024 as the casual mask size (therefore the max size of sequence it can produce is limited to 1024). The model is trained with a batch size of 4 which is

definitely not ideal. The goal of the model is simple which is to predict the next token given the past tokens. The input to the model are genomes pre-processed to look like Fig 3.1.

3.1.3 BERT

A BERT model as explained in Section 2.3.3 is used to model the complete genomes at the nucleotide level. The source code used to train the model can be found at https://github.com/yoheikikuta/bert-japanese/commit/613c376feb8d102585cd8a3609d02ce41a02cbbd. The data tagged like as shown in Fig 3.1 is split into lines that are 512 tokens long. The model is trained with a vocabulary size of 100, 12 layers, 768 as the embedding size, 12 attention heads and with gelu [16] as the activation function.

3.1.4 styleGAN

In this experiment a nucleotide sequence is converted to an image of resolution 256x256. This is achieved by mapping every nucleotide single-letter [6] present in the data (complete genomes obtained from NCBI) to a color. The mapping of nucleotide single-letter to a color is done as shown in Fig 3.2. During the sequence to image conversion, alternate lines are flipped so that the spatial closeness is preserved. These images are used to build a syleGAN model [22].

3.2 Protein Family Generation

In these experiments, the goal is to generate amino acid sequences of DNA-directed RNA polymerase β' subunit (RNApol). RNA polymerase is a protein complex that is active during the process of transcription. Its structural and functional properties are studied extensively

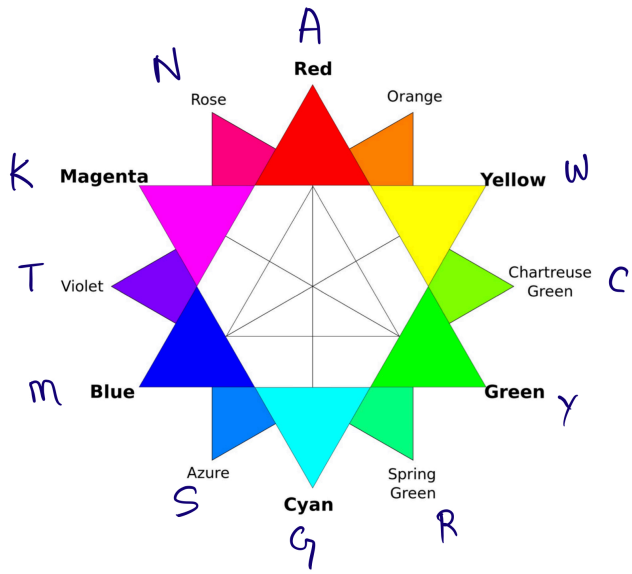


Figure 3.2: Codon to Color Mapping

across the three domains of life. Due to its function it is conserved pretty well allowing it to be used as a good candidate to construct phylogenetic trees. This allows us to study the relationship between different species. It also allows us to analyze the phylogenetic relationships between the proteins coming from the models (synthetic proteins) and those coming from nature. The models described below are written in PyTorch from scratch. The data is served using dataloaders which preprocess '*.faa' files during use, negating the need to keep the entire preprocessed corpora in memory.

3.2.1 Data Sampling

The goal of this dynamic programming based sampling strategy is to make sure that we create a dataset that is comprised of protein sequences which are taxonomic-ally diverse. In order to achieve that we start with a large set of protein sequences coming from a particular family (COG0086). This is the large dataset from which we want to create a diverse subset. To achieve this we first start with creating a tree using the GTDB taxonomic information [39]

of all the protein sequences present. The tree described thus contains nodes which are GTDB taxonomic terms and has a corresponding count (number of occurrences of the term in the whole dataset). Each depth level in the tree corresponds to a particular taxonomic rank, starting from 'Domain' (root of the tree). This tree is then used to pick samples which have a high taxonomic diversity (number of GTDB terms in the child node) and samples which have low occurrence (sampling without replacement). This ensures that no one taxonomic rank dominates the dataset.

3.2.2 Tokenizer

As explained in Section 3.1.1 the deep learning models need an integer representation of the sequence for training. In the experiments described below, amino acids or combinations of amino acids along with custom symbols as described below counting up to 29/50 different tokens are considered vocabulary. This is obtained in an unsupervised manner by training a unigram model [27] using sentencepiece [28].

3.2.3 Evaluation Strategy

Two different strategies were used to evaluate whether the proteins produced were biologically realistic (i.e. can be considered a member of the protein family). First, proteins were compared to the NCBI RefSeq [37] database to identify sequence similarity with reference proteins. This comparison is done using blastp [2] which is one of the most widely used bioinformatics tools, and is considered the 'gold standard' for protein-protein comparison. In this analysis, a high score (greater than 50) and low E-Value (less than $10 - 2$) are considered to represent biologically meaningful relationships.

Secondly, synthetic proteins were evaluated by including them in a phylogenetic tree [10]

together with reference proteins. In this analysis, proteins are considered biologically realistic if they are interspersed with real proteins in the tree. Unrealistic sequences would form a distinct long branch away from real sequences.

3.2.4 char-RNN

The objective of this experiment is to generate RNApol one token at a time (this could be an amino acid or a combination of amino acids) using the charRNN model architecture shown in Fig 2.5. The data is sampled using the approach described in Section 3.2.1 to create a taxonomically diverse dataset. This diverse dataset is split into train/test/validation with 0.8/0.1/0.1 size ratio, respectively. Various RNN-based PyTorch models as mentioned in 2.2 are trained using this dataset. The best performing model out of this is then selected for further analysis. The goal of the RNN-based model (+ fully connected layer) is to learn a probability distribution over the tokens for the next token given all the tokens in the past (learns a distribution over the vocabulary given the past tokens). This is achieved by minimizing the cross-entropy betweeen the predicted labels and the ground truth (one-hot vector).

This distribution is used to pick a token during generation. A temperature parameter is used to adjust the uniqueness of the sampled token [1]. The optimal temperature value is found by trial and error. The generation process starts with a seed value. This seed value is then used to predict the next token. The hidden vector at this stange is passed onto the next time step as this alone contains all the necessary information of the past. The prediction is terminated when a ”stop codon” is encountered.

This process is repeated multiple times to produce several proteins. These proteins along with the naturally occuring proteins are aligned together using ClustalW [51]. Once these multiple sequence alignements are found, fasttree [42] is used to create the approximately

maximum-likelihood trees (phylogenetic trees). The synthetic sequences are also run through blastp [32] to get sequence-specific scores. These scores are then used to determine the quality of the proteins generated.

3.2.5 char-RNN-taxo

In this experiment, the prediction of the next amino acid is conditioned on the embedding of the phylum that sequence belongs to. Before we explain how this conditioning is achieved, it is important to first explain the procedure to generate taxonomic embeddings. The phylum embeddings are generated using Node2Vec [14]. This is obtained by first building a directed tree from the GTDB taxonomy [39] where the first two children correspond to the Archaea and Bacteria domains. The rest of the taxonomy is added in such a way that every level of the graph corresponds to a taxonomic rank, where the first level corresponds to domain and the last corresponds to genus. This graph is given as input to node2vec to obtain an embedding vector for every ontology term (Onto2Vec). The exploration and exploitation parameters (p and q) are adjusted along with the walk length. The optimal values are found by trial and error. Using this, a Node2Vec model is trained. The model checkpoint with the best validation accuracy is picked as the model from which the taxonomic embeddings are extracted. The resultant node embeddings in 2 dimensions (achieved through UMAP) is shown in Fig 3.3. The top left cluster corresponds to all Archaea-related taxonomy terms and the other, all Bacteria-related taxonomy terms. The network embeddings of phylum terms are used along with the sequence information to obtain the probability distribution of the next token over the vocabulary as shown in Fig 3.4.

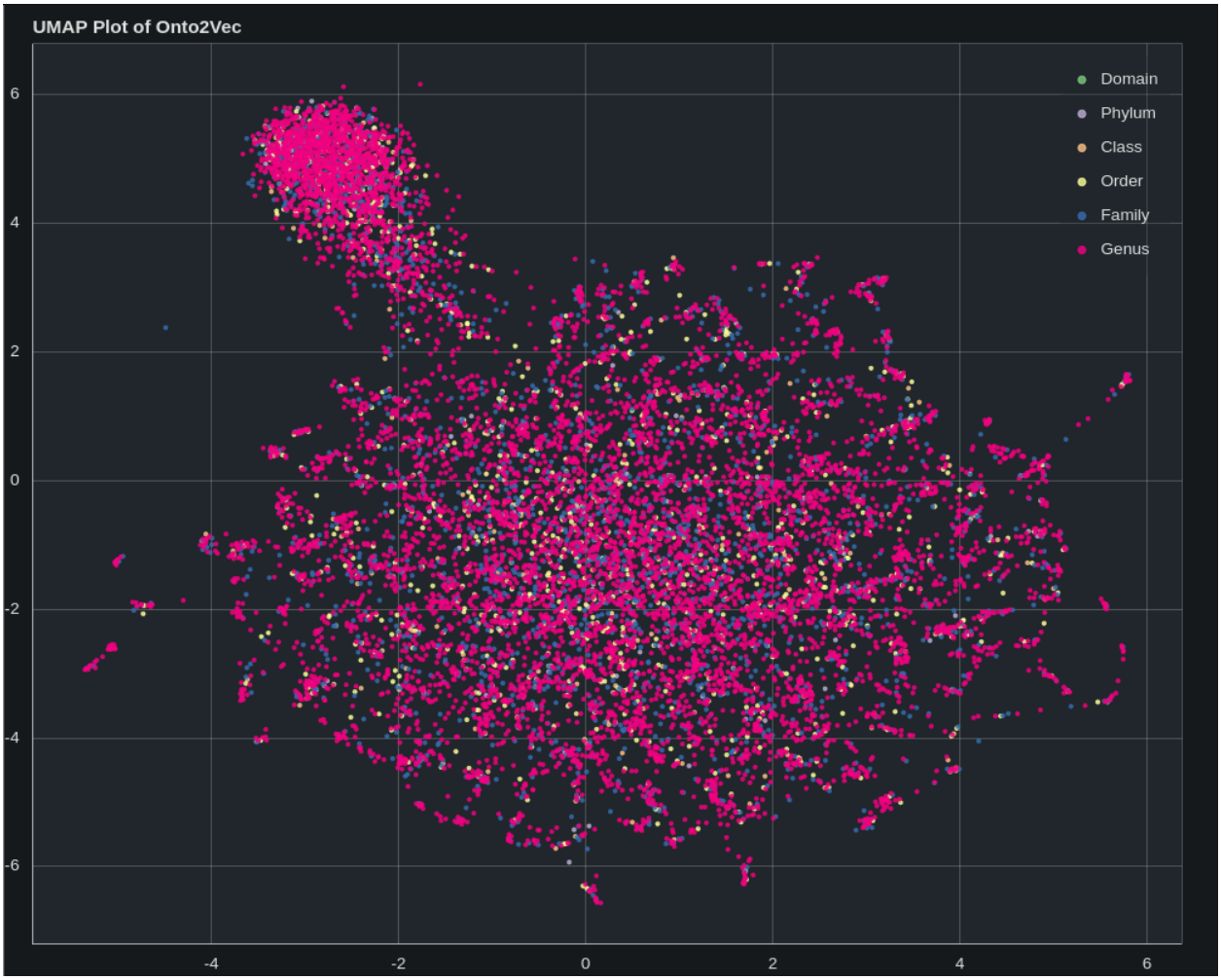


Figure 3.3: UMAP plot of network embeddings of GTDB terms found using Node2Vec

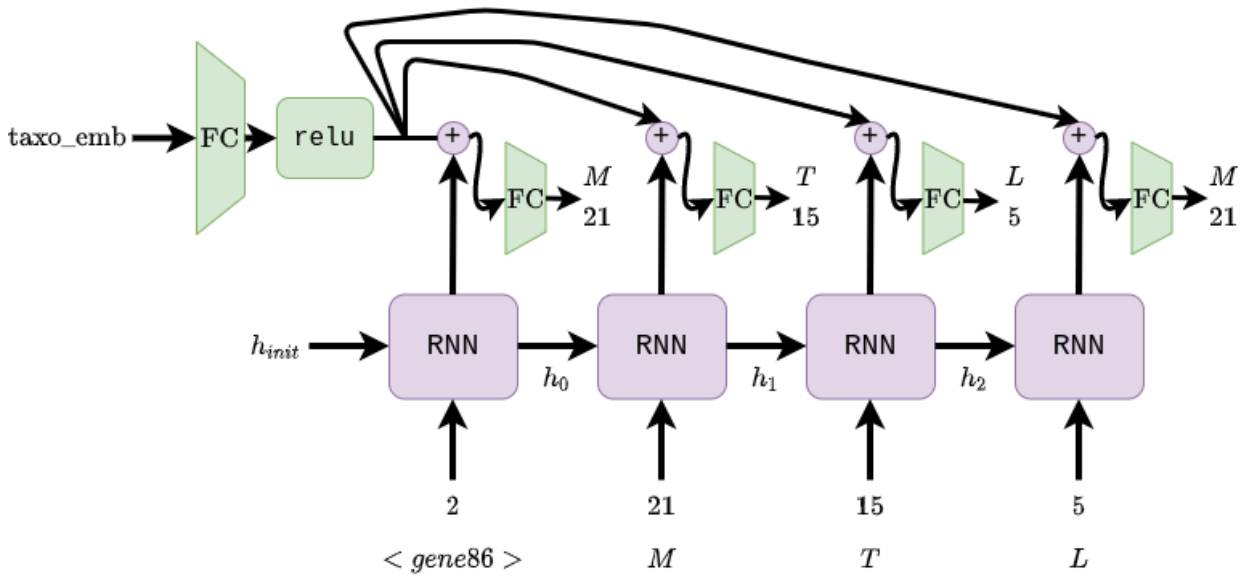


Figure 3.4: Char-RNN-taxo Architecture

3.2.6 char-RNN-gumbel

Gumbel-Max trick [15][33] provides a way to interpolate between discrete one-hot-encoded categorical distributions and continuous categorical densities as shown in Fig 3.5. In this experiment a gumbel-softmax [20] is used to simulate the selection of a token. The rest of the architecture is exactly like the plain char-RNN architecture as explained in 3.2.4. This formulation allows for making both the training and evaluation phase more similar than the above experiments. The parameter τ is found in multiple ways such as starting from a large tau value followed by annealing and making tau a parameter that is learnable.

3.2.7 Reformer

In this experiment, tokens are predicted one at a time through the use of an auto-regressive model based on reformers [26]. Here, a model which is 6 layers deep with 4 attention heads is trained. The model has a dimension size of 256 except for the embedding vectors which

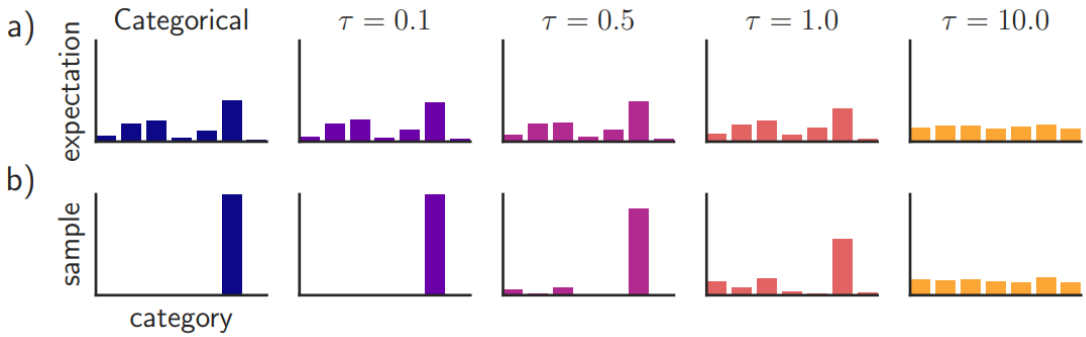


Figure 3.5: Gumbel-Softmax distribution (Source: Figure 1 [20])

are of length 128. The bucket size and number of hashes are 64 and 6 respectively. The position encoding method chosen is axial position encoding. The model is trained with a causal mask so that the model can only attend to tokens in the past, thus bringing in a sense of direction.

Chapter 4

Results and Discussion

4.1 Complete Genome Generation

The efficacy of generating complete genomes was not up to mark. The individual experiments performed had varying levels of success. These are enumerated in the sections below. One of the major challenges faced in all the experiments below were resource limitations due to the fact that self-attention used in vanilla transformers is quadratic with sequence length (the average length of sequences was 2k) [56]. This meant that lower batch sizes had to be used, which reduced the chance of finding the optimal loss/minima within reasonable time.

4.1.1 GPT-2

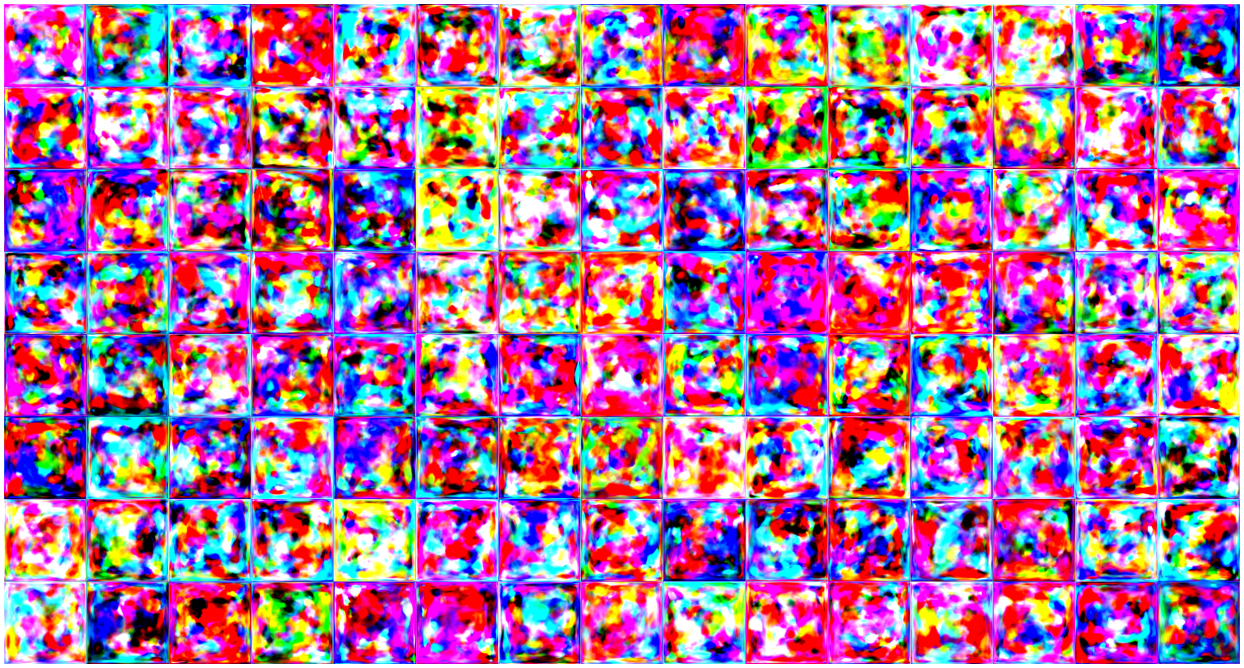
Using the transformer decoding layers (GPT-2) had the best success of the complete genome experiments conducted. This model was able to understand the simple structure of interspersed intergenic and gene-coding regions. A sample output generated by the model showing such structure is shown in Figure 4.1. But, unfortunately the gene-coding regions generated by the model didn't produce relevant protein coding genes when evaluated on BLASTn. The loss of the model is still decreasing and is now at 2.98.

<genome><inter>CCACTGTTCGATCCGAGTCGCCCTCACCGGTGCGGGGCCGATGCTACAATTGGGGCCG
</inter><gene>ATGCTCGCATTCCTGCTCGCGTTGGCGCTCGTTGCCCTCGTTCTCACCGACGCCGACGC
CTCCTCGGCCGGAGCCCGGAAGACCCGCGCATCCGATGTGTCGGTGGCCCAGGCGGTCCTCGACCGCCGGGAG
GATCTGGTCGAGGCCGTGGCCCTGATGCGCGGCACCGACGCACCGAAGGAGCCCCGCGCCCGCCAGCGCATCG
AGAGGCCGGGACGCTCGCGGAGGCGCAGGTCCAGCTCTCCGAGGTGGAGCGCCGGCTCCAGGAGGAGCTGGAG
CGCCGCCGGCGCGACGAGGAGGAGCGGGGCACCCGCGCGCTCCTCGCCGAGCTCGCGCCGGTCGATCGGCTCG
ATCGCTTCGACGAGGACGAGGGCCTCACGCCGGAGCTCCGGGCGAGCGCGCTCGAGGATCTGCGCAGGCAGAT
CCTGATGGACTCGGACGACCACGACCCGCGGCGTCAGTTCCAGGCCGGGATCTTCAAGGAGCTCGTGTTCCAC
GAGGTCGGCCCGCTCGGGAAGCTCGCCGCGAGGCTCGAGGAGTCGGCCCGCGCGACCGGCGTCCCGCGCTCGG
CGCTGGGCTCGGGCGCCGTCTCCGGCGTCGTCCTCTTCCTGGGCGTCCTCGCGGTCGGCGTGGCCGGCGGGTA
CCTGATCCGTTCTCCCGGCAAGGCGGCCGCCATCGCGAGGAGGGTCGCTCTCATCGGCGTCGTCGTCCTCGCG
CTCACGATGATCTACCGGCTCGCCCCGTCGTCGGACGATCCCGAGCTCGTCGACGAGGTCGCCGGCGCCGAGA
CGATCGGCTCCACGGATCGGAACGTGCCCTCGGCCCGCTCGGTGCTCGAGCTCCTGCGGGTCCGGGAAGCCCG
CGCGGAGCTCACGGAGGGCGATGAGGTCTCGGGGGAGGCCGTCGAGCTCGCGCTCGACAAGGTGAACGACGAG
CTCTCCGCGGGAAACGCCATGCTCCTGCGCAGCGCCCGCGCCTCGCTCACCTCCGTCGATCTGCGCTCCGGCG
ACGTCGAGAGCGGCGCGTCCATCGCCGTCGCGATGGAGGCGCAGGCCGAGAAGCAGCGGATCAACGAGGATCT
CCGGAAGATCCGGCAGATCGGGGGCAGCGGCGAGGACGTCACCAAGAGCGCCATGGATCAGGTGAGCGAGCTG
AACAAGATGTCGGTGACGGCGCCGACGATCCGGGTCATCGTGCACATGGTCGAGAACACCATCCGCACGCTCA
ACAGCCGCACGACGGCCGCCTCCCTCACCGAGATCCGGTCGAGCGCCGCGGGAGACGGCCGCAAGGCCGCCAA
CATCCTCGAGGTCTTCGCGGACGTCACCCGCAATGCGGTGAAATCGACCGAGGCCAAGGCCCGCATCGCCCAG
GTGCAGCTCGATCTCACCCGGTCCATCCGCGGCCTGATCGAGGCGGTGACGATCAACGCCGCCTCCGAGCACA
CCAACGTCACCAGGCAGATCGCCGCCGGTGCCGCCTCGGCCAAGGAAGCGGCCGAGGAGGTCGGCAAGGCGGT
CGCGGGCATCGGGGAGACCGTCACCAAGGTGGAGGAGGACAGCCGCCTCGCCGACGAGATCGCTGCGATGAAC
CGGCAGGTGCCGGGCGCCAACCACGGGCAGATCGTCTACACCCTGGTCCAGCAGATCCAGCGCGCCTCGTCCG
ACGCGGCCCGGCTCCTGGCGGAGGTGCGGGCGAACGCCGTCGACGCCACGCCGCTGAGCGGCCAGATCCTCGC
CGCCATCGAGGCGGCTTCCGCCCGGGGCGCGGGCCACATCGTCCGGCTCCTGGACACCGACGGCCTGGTCCGC
TGGAACGTGGCCAAGGAGGATCGCGAGGAGCGGGCGATCATCGCCGCGGCCGCGGTGTACACGTTCCCCTCGG
ATCGGGACGCGCTCCTGAAGGTCGGCTACTTCGACCTGGCGAACCGCGACCGGCAGCAGAAGGTGGCCGAGCA
GATCAACGGCTCGAGCGAGAAGATCCAGGCCGAGGAGGCCGAGAAGAAGGCCGACGAGCTGACCCAGGACGAG
CTGATGGCCTTCTCGTTCGGCCTGGGTCCGGGCCTGCTCACGATAGGCGCTGTGATCTTCGGGTTCGTCATGA
TGAGCCCGGTGATGCGTCGCCCGCCCCACGTGACGCACCTGCCGGCGCCGCGGGGCGAGGAGCGC
TGAC</gene><inter>GCCG</inter><gene>ATGGACGCCGTCGCCCTCGACCTGGTCCGGCCGACC
GCCGAAGAGATCGTCCTCCGGGTGGACACCTCCGCGGAGGCGATCGACCGCCATCTCGC

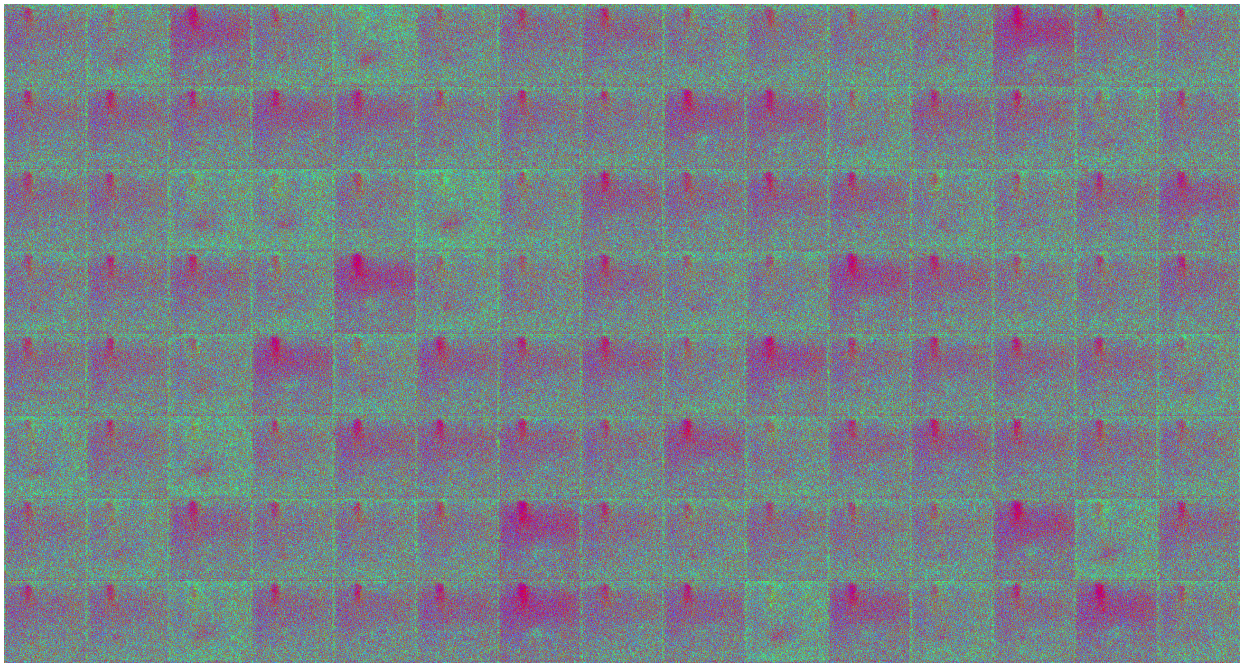
Figure 4.1: GPT-2 Sample Output

4.1.2 styleGAN

The styleGAN was trained with 1 GPU into producing better and better output (visually) (Figure 4.2a 4.2b) but the decoded (from image) nucleotide sequence produced no relevant protein when run through BLASTn (Figure 4.3). A better output could be produced if the model was trained with higher batch sizes on a large amount of GPUs in parallel like how the authors intended. But this was not possible due to resource unavailability. Few samples at different epochs during training are shown in Figure 4.2.



(a) Samples at Epoch 0



(b) Samples at Epoch 6926

Figure 4.2: Samples from styleGAN Generator

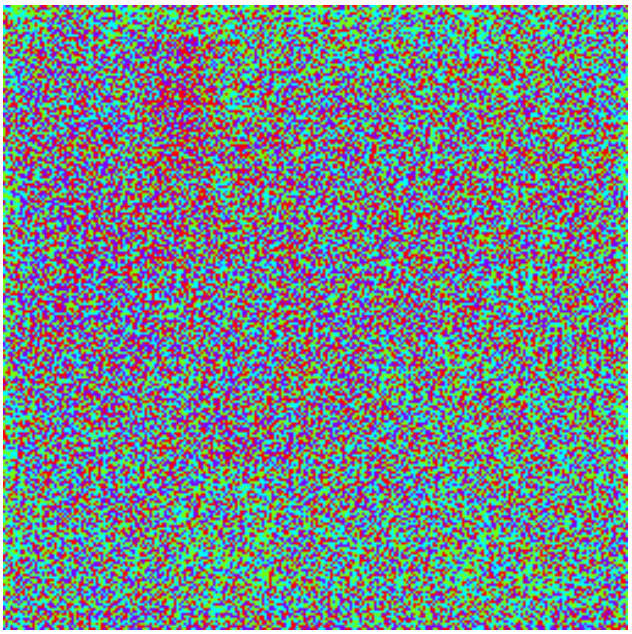


Figure 4.3: styleGAN Sample Output

4.1.3 BERT

The goal of this model isn't to generate a sequence as such but to produce high quality representations. The final results are shown in Fig 4.4. Due to truncation of the the sequence to 512 tokens, long term dependency information might have been lost. This could be one reason for the bad performance of the model. The transformer could also be switched to transformer-XL [7] but this wasn't done due to resource limitations.

```
I1123 19:55:59.755087 140477709883200 run_pretraining.py:488] ***** Eval results *****
I1123 19:55:59.755134 140477709883200 run_pretraining.py:490]   global_step = 5000000
I1123 19:55:59.758524 140477709883200 run_pretraining.py:490]   loss = 3.2472062
I1123 19:55:59.758663 140477709883200 run_pretraining.py:490]   masked_lm_accuracy = 0.2959375
I1123 19:55:59.758767 140477709883200 run_pretraining.py:490]   masked_lm_loss = 2.7834735
I1123 19:55:59.758859 140477709883200 run_pretraining.py:490]   next_sentence_accuracy = 0.765
I1123 19:55:59.758891 140477709883200 run_pretraining.py:490]   next_sentence_loss = 0.46373343
```

Figure 4.4: BERT Training Statistics

4.2 Protein Family Generation

The goal of the experiments, as mentioned earlier, is to see if it is possible to generate proteins from a particular protein family. The experiments conducted below are trying to generate the DNA-directed RNA polymerase β' subunit. The results of these experiments were drastically different from that of the complete genome generation experiments, in a sense that they were able to generate high quality proteins with ease. The output of the models discussed below were benchmarked/evaluated using the strategy mentioned in 3.2.3. The choice of char-RNN model was motivated due to its high efficacy [] in generating sequences in various fields. Althoug the charRNN model trained and was able to generate high quality proteins (as seen by its excellent blastp score), it struggled in generating diverse outputs (taxonomic diversity). This might be due to mode collapse during the generation process. Therefore, ways to tackle this had to be devised.

The Reformer provides a way to allow the model to learn representations of the sequence without being forced to confine it within one vector. It also, due to its interpretability allows one to where in the past the model is looking at in order to make the decision for the next protein. This although had excellent blastp scores struggled with mode collapse.

One strategy is to see if providing explicit taxonomic information along with the sequence helps. As is shown detail below, such information does help the model (charRNN-taxo) being able to generate taxonomically diverse proteins. This can be seen from the phylogenetic analysis done (synthetic sequences more evenly spread among reference proteins compared to charRNN). But, this model, due to the class imbalance in the training data (some phyla dominated the dataset) was not able to fully learn good quality sequence-phyla relationship (especially the phyla which are least represented).

Another strategy would be to improve the selection process (token selection) by trying to learn to pick a token as explained in 3.2.6. This was implemented creating the model

charRNN-gumbel. This although benchmarked better than charRNN according to the blastp and tree-based evaluation. Overall, these five approaches produced biologically-reasonable protein sequences, and therefore provide a framework for future deep learning approaches that seek to simulate biological data.

4.2.1 char-RNN

The model is trained and samples generated as explained in 3.2.4. Although, gradient clipping helped [40][13], The training of the LSTM model was still difficult to train due to exploding values. 50 sequences were obtained which one of which is shown in Fig 4.5. The sampling strategy used was top-p with 0.9 cut-off and a temperature of 0.7. These sequences once aligned and trimmed along with the natural sequences, was used to built a ML (maximum-likelihood) based tree (phylogeny). This tree is shown in Fig 4.9. Few of the output sequences are run through blastp the run of which is shown in Fig 4.6. The metrics to note here in the blast output was the exceptionally high 'total score' and 'query cover'. To explore how the model has managed to perform well, the token embeddings of just the amino acids are extracted from the model. These high dimensional vectors are then reduced to a two dimensional vector using a dimensionality reduction technique UMAP [34]. This technique among having other advantages allow for the use of different distance metric. In this work, cosine similarity measure is used as the distance metric. The resultant output is plotted as shown in Fig 4.7. From the UMAP plot, one can clearly see the the similarity between 4.7 and 4.8. This shows that the model was able to learn physio-chemical similarities without being explicitly given such information.

The hidden state of the LSTM is designed in such a way that is stores all the information in the past. Therefore, the next experiment was to see if the hidden state of the charRNN model after processing the entire sequence could be used to create a phylogeny tree. Hidden

state of few sequences were extracted to be used in distance based phylogeny tree creation methods like neighbour joining [48] and UPGMA [36]. The results of which are shown in Fig 4.10. The resultant output 4.10b 4.10c does fair well when compared to the traditional (ML-based) tree creation method as shown in Fig 4.10a. More analysis is needed to show the efficacy of this alignment-free phylogenetic tree creation method but the results so far definitely shows promise. A similar 2D plot of the high dimensional hidden state vector of natural and synthetic sequences show how they are related to each other Fig 4.11.

```
<gene86>MDELKGLFKKVGQEEDFDAIKISLASPDKIRSWSHGEVKKPETINYRTFKPERDGLFCAK
IFGPVKDYECNCGKYKRMKHRGIVCEKCGVEVIQSKVRRERMGHIDLASPVAHIWFLKSLPSRIGLLL
DMTLKDIERVLYFEAFVVTDPGMTELKRGELLTEEEYLDAMEEHGDEFDALMGAEAIRRLLEKIDLEQ
LATELRKELRNEKSEAKRKKLAKRLKVVEAFRHSGNRPEWMILDVVPVIPPELRPMVQLDGGRFATSD
LNDLYRRVINRNNRLKRLLELGAPEIIVRNEKRMLQEAVDALIDNGRRGKAVTGAGNRPLKSLSDMLK
GKQGRFRQNLLGKRVDYSGRSVIVVGPELKLHQCGLPKIMALELFKPFIISKLIERGFVHTVKSAKRM
VERERPEVWDILEEVIKEHPVLLNRAPTLHRLGIQAFEPVLVEGKALQLHPLVCTAFNADFDGDQMAV
HLPLSIEAQAEARLLMLSTNNILSPAHGRPLVTPTQDMVLGAFYLTVESDGVEGEGSRYRDVEEAERA
YQFGAVGLHARVRVRLPGKLIDTTVGRVLFNQILPDDMGFYNEVLNKKQLENLIGRTIKLAGNKETAK
FLDELKDLGFEFATRAGTTISIDDLVVPEEKRDILEGAEKEVRRIEKQYRRGLITDDERYQKVIDIWT
EATDDVAQVLMDSLDPFNSIYMMANSGARGNIDQIRQIAGMRGLMADPSGKTIDLPIKANFREGLTVL
EYFISTHGARKGLADTALRTADSGYLTRRLVDVAQDVIVREEDCGTDEGIWMSAVVDGGETVEPLGER
ILGRTALDDILDPVTGEVIVEANEMIDEELADRIEQAGIEKVRIRSVLTCRAKRGVCAKCYGRNLATG
RLVEIGEAVGVIAAQSIGEPGTQLTMRTFHIGGTASRVAEQSTLEAKNDGIVKFENLRTVTNRDGDLV
AMNRHGELVYLDDNGRERERYSLPVGATLAVKDGAKVEKGDVLARWDPFTQPIITEVGGKVRFGDVIE
GVTMQEQTDEVTGLSSLVILDPKQRGGAGKDLRPMVKLVDANGKDLYLPGTDEPAYYLLPVGAHISVE
PGANVQPGDVLAKIPRETTKTKDITGGLPRVAELFEARKPKEHAVVTEIDGVVEFGKDTKGKRKIVVQ
PEMGEPHEYLIPKGKHISVQEGDFVKAGDPLMDGTSNPHDILSVLGEKELQKYLVDEIQQVYRLQGVK
INDKHIEIIVRQMLRRVRVEDVGDTDFLSGEQVSKKKFREENERVLAEGGRPAVARPLLLGITKASLS
TDSFLSAASFQETTRVLTDAAIKGKVDYLRGLKENVIIGKLIPAGTGMDYYKEIRIETKSRTRDAVMP
VMEE*</gene86>
```

Figure 4.5: Example of a generated protein using 2L_LSTM

4.2.2 Reformer

The model is trained and samples generated as explained in 3.2.7. The top-p sampling strategy was used with p=0.8 and temp=0.7. A sample generated in such manner is shown in Fig 4.12 which is then analyzed using blastp, the result of which is shown in 4.14. The inference time of this model was significantly larger than the charRNN based methods even though the quality was comparable. This means that further code-base inference optimization should be done to make it faster (the code-base used is an active development repo so performance

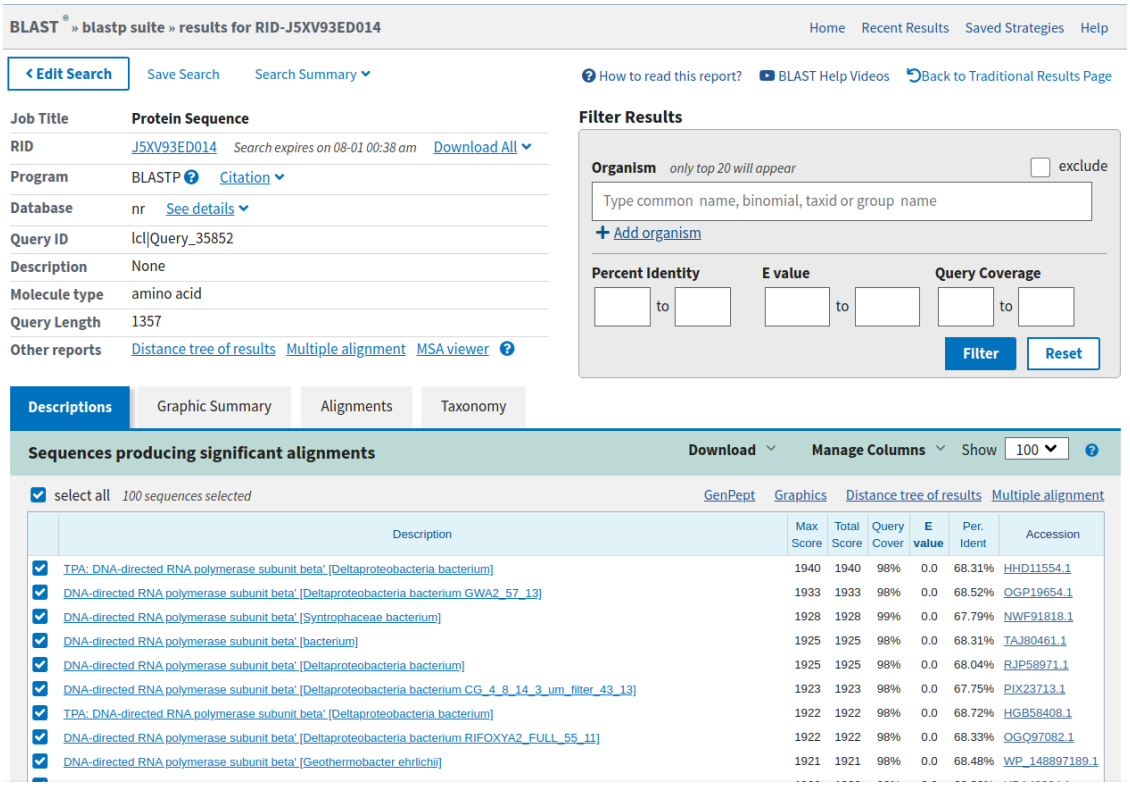


Figure 4.6: BLASTp of generated protein shown in Fig 4.5

might be better in newer versions). The reformer could also be switched out to other efficient transformer architectures like sinkhorn [53] or [57] to enable better train/inference time performance (both sequence quality and inference time). The vocabulary embeddings are reduced to a 2D vector and plotted (Fig 4.16). This model too captures the physio-chemical properties of the amino acids like as shown in 4.8. One of the main advantages of the attention based model is that it enables interpretability. Looking at the attention weights enable us to get an idea of what the model is looking at when deciding to generate the probability over the vocabulary. This attention over a conserved motif [49] is analyzed. This catalytic core (which has a highly conserved motif "NADFDG") is responsible for coordinating a magnesium ion due which it is highly preserved. This motif was found in a data subset using a straight-forward subsequence search written in numba. The attention around this region is

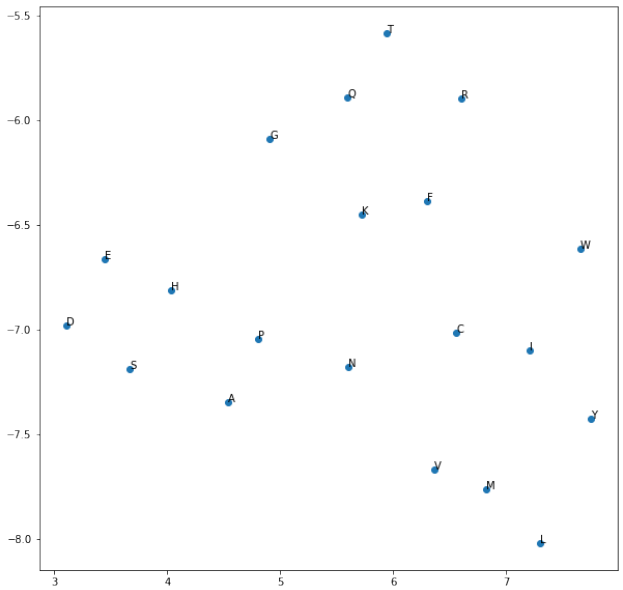


Figure 4.7: Amino Acid Embeddings learnt by 2L_LSTM

visualized in Fig 4.16.

4.2.3 char-RNN-taxo

The model is trained with cross-entropy loss like all the above models as mentioned in 3.2.5. Samples from the model were generated with the same parameters as the char-RNN (top-p sampling with p=0.9 and temperature=0.7). One sequence generated in such manner is shown in Fig 4.17 along with the blastp output of it in Fig 4.18. These in-silico sequences along with their natural counterparts are aligned, trimmed then made to create a ml-tree the output of which is shown in Fig 4.20. The phylums used in creation of the tree were selected at random for all the gtdb phylums present in the dataset. The idea of this model was that instead of the model being tasked to learn the diversity of RNAPol among all the phylums implicitly, providing a phylum embedding would provide some sort of explicit nudge to learning that relationship. Although, for few phylums it was able to generate its respective RNAPol (found using blastp), for some it wasn't able to. This might be due to

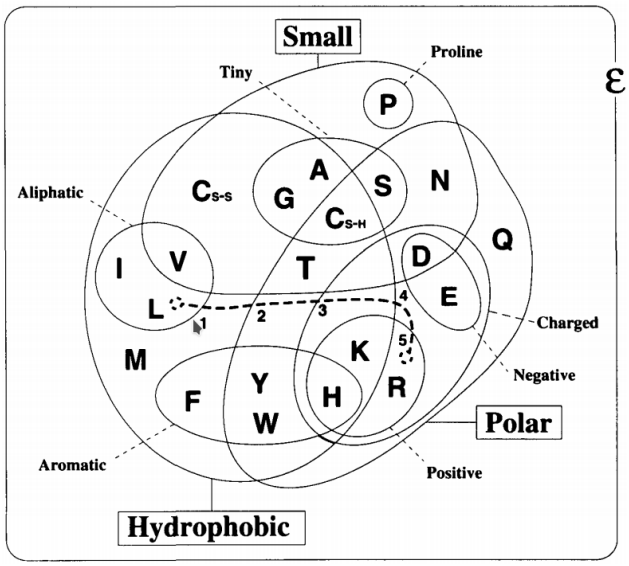


Figure 4.8: Amino Acid Similarity Based on Physio-Chemical Properties (Source: Figure 1a [30])

the fact the class distribution of phylums were extremely skewed (high class imbalance). The amino acid embeddings learnt for this model is shown in Fig 4.19.

4.2.4 char-RNN-gumbel

The model is trained and samples are generated as mentioned in 3.2.6. Samples from the model were generated with the same parameters as the char-RNN (top-p sampling with p=0.9 and temperature=0.4). One sequence generated in such manner is shown in Fig 4.21. This sample is also run through blastp whose output is shown in Fig 4.22. Although, the model generated high quality proteins belonging to the protein family, the idea of making the model learn the idea of selection is something that have to be worked on. Making τ a function of the past sequence is definitely a way to go due to the fact that proteins generally have varying conservation level at the sequence level. A phylogeny is nonetheless created as shown in Fig 4.24 and the amino acid embeddings visualized in Fig 4.23.

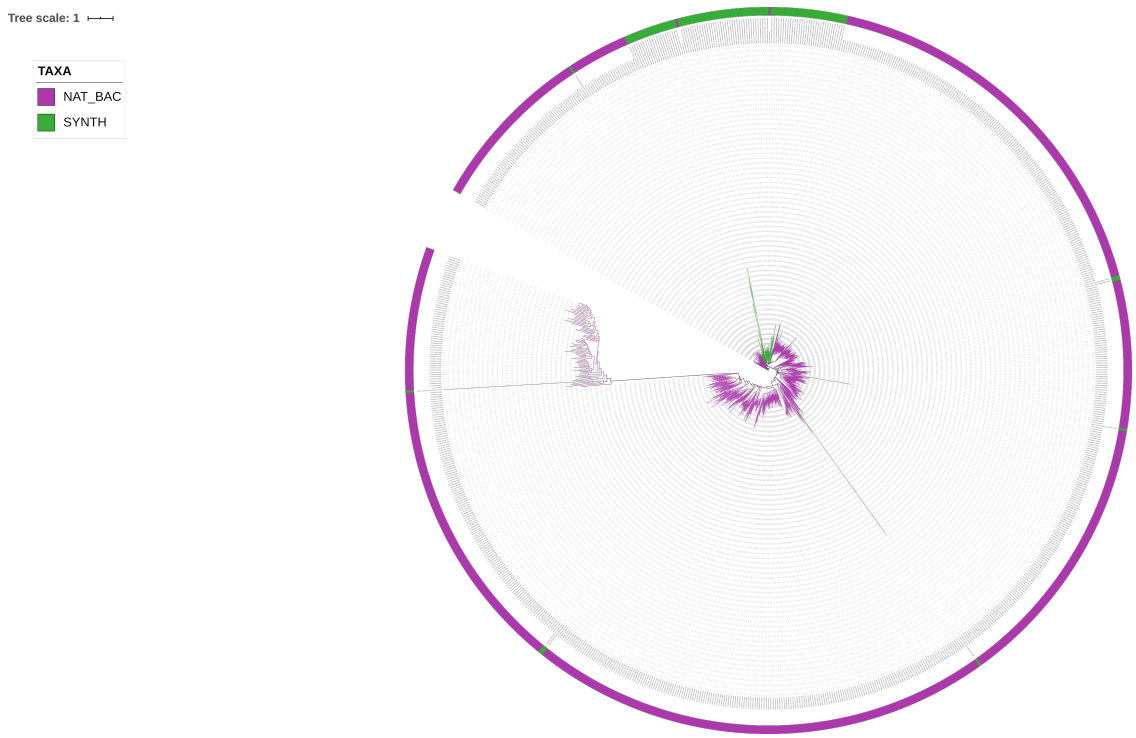


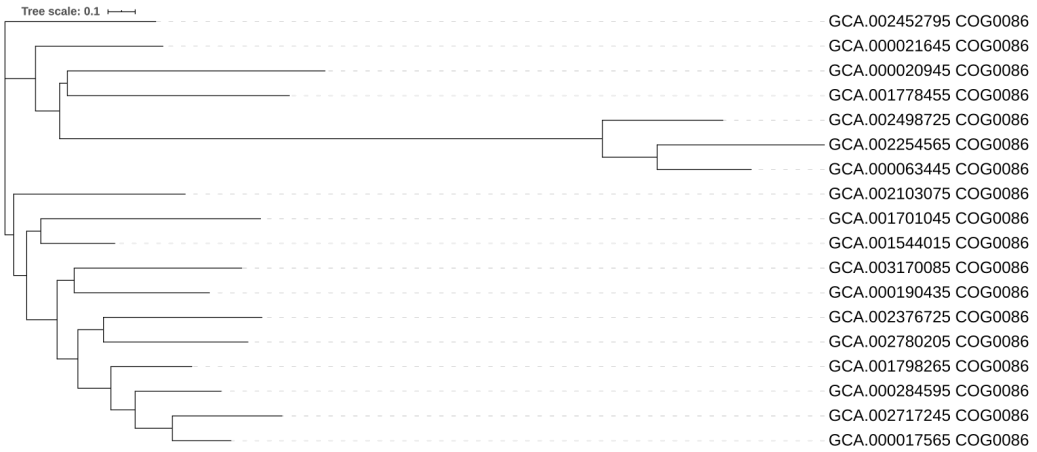
Figure 4.9: 2L_LSTM: Phylogenetic Tree of Natural and Synthetic Sequences

2L_LSTM	1.298
Reformer	1.430
2L_LSTM_Taxo	1.294
2L_LSTM_Gumbel	1.506
2L_LSTM_Gumbel_Taxo	1.793

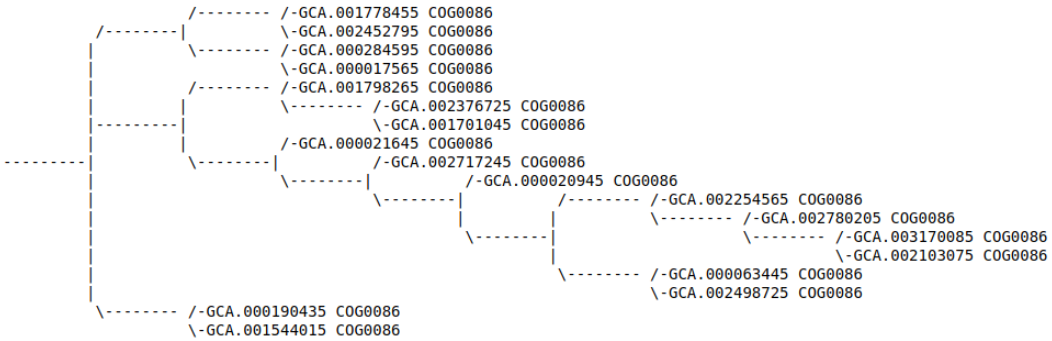
Table 4.1: Test Loss

4.2.5 char-RNN-gumbel-taxo

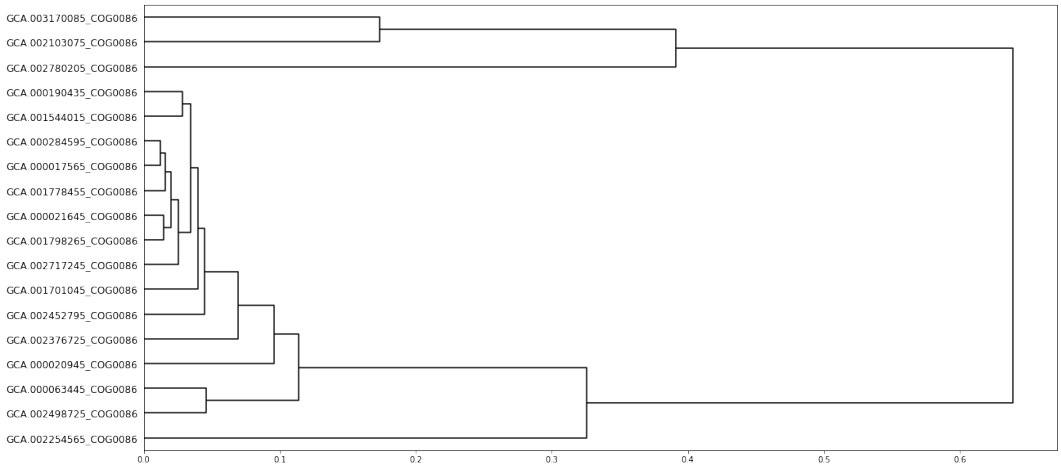
This model combines both the char-RNN-gumbel and char-RNN-taxo into one. The output wasn't drastically different. The different outputs as shown in the above experiments are repeated and are shown in Fig 4.25 4.26 4.28 4.27.



(a) Phylogenetic Tree created using traditional pipeline



(b) Phylogenetic Tree created using NJ method from sequence hidden representations



(c) Phylogenetic Tree created using UPGMA method from sequence hidden representations

Figure 4.10: Phylogentic Tree Creation

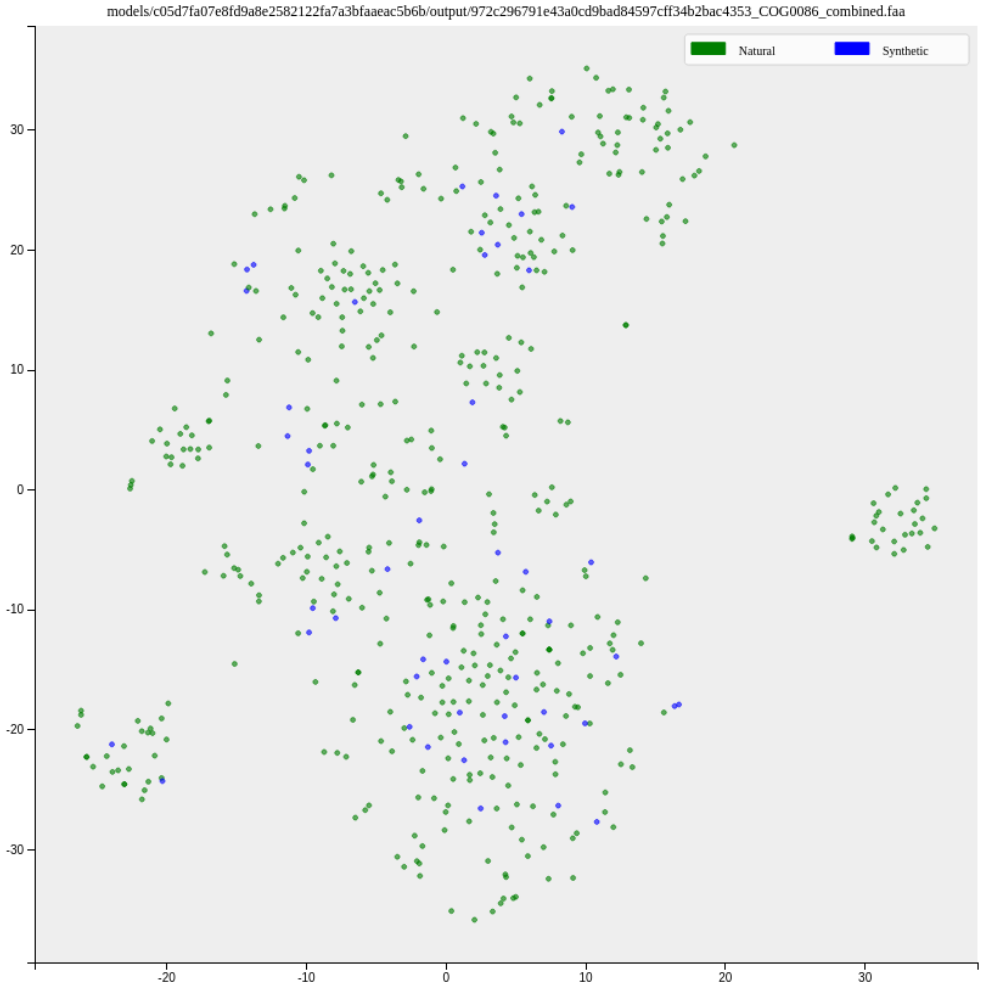


Figure 4.11: 2L_LSTM: t-SNE plot of both natural and synthetic sequence embedding

```
<gene86>MLDVNNFDSIRIGLASPEQIRSWSYGEVKKPETINYRTLRPERDGLFCERIFGPT
KDWECHCGKYKRVRYKGIVCDRCGVEVTRSKVRRERMGHIELAAPVSHIWYFKGIPSRMGLIL
DMSPRALEEVIYFAAYVVIDPGDTPLMKKQLLTETEYREYREKYGNAFKAGMGAEAIKELLLE
EIDLDKMAELRKELKEVSGQRRNRAIKRLEVVEAFRKSGNLPSGNRPEWMVLDVIPVIPPDLR
PMVQLDGGRFATSDLNDLYRRVINRNNRLKRLLDLGAPDIIVRNEKRMLQEAVDALIDNGRRG
RPVTGPGNRPLKSLSDMLKGKQGRFRQNLLGKRVDYSGRSVIVVGPELKLHQCGLPKEMALEL
FKPFVMKRLVDLGHNIKSAKRMVERVRPEVWDVLEEVIKEHPVLLNRAPTLHRLGIQAFEPVL
VEGRAIQIHPLVCTAFNADFDGDQMAVHVPLSAEAQAEARLLMLSAHNILNPANGRPVTTPTQ
DMVLGCYYLTVIADGAKGEGRKKGEGMVFASPEEVRAYDHGAVGLHAKVKVYDTGELIETTVG
RVIFNEAIPEDLGFVNKVNKLNKLVAQIIDHHRLGNDTARFLDGIKDAGFKYATKSGITISVS
VDDIKVPPEKKEILAKAEEVEKQYQRGLITDGERYNKVIDIWNEATEAVTEQVTEAMFREMKK
DQGFNPIFMMADSGARGNISQLRQLAGMRGLMADPSGRIIDLPIKANFREGLTVLEYFISTHG
ARKGLADTALRTADSGYLTRRLVDVAQDVIVREEDCGTTEGITVRAITEGDEV</gene86>
```

Figure 4.12: Example of generated protein using Reformer model

BLAST® » blastp suite » results for RID-K8EU5V7X014

Home Recent Results Saved Strategies Help

< Edit Search Save Search Search Summary

How to read this report? BLAST Help Videos Back to Traditional Results Page

Job Title	Protein Sequence
RID	K8EU5V7X014 *Search expires on 08-14 02:56 am* Download All
Program	BLASTP Citation
Database	nr See details
Query ID	lcl|Query_29597
Description	None
Molecule type	amino acid
Query Length	801
Other reports	Distance tree of results Multiple alignment MSA viewer

Filter Results

Organism *only top 20 will appear* exclude

Type common name, binomial, taxid or group name

+ Add organism

Percent Identity to **E value** to **Query Coverage** to

Filter Reset

Descriptions Graphic Summary Alignments Taxonomy

Sequences producing significant alignments Download Manage Columns Show 100

select all *100 sequences selected* GenPept Graphics Distance tree of results Multiple alignment

Description	Max Score	Total Score	Query Cover	E value	Per. Ident	Accession
DNA-directed RNA polymerase subunit beta' [Thermincola potens]	1318	1318	100%	0.0	80.93%	WP_013119201.1
DNA-directed RNA polymerase subunit beta' [Thermincola ferriacetica]	1317	1317	100%	0.0	80.93%	WP_052218136.1
DNA-directed RNA polymerase subunit beta' [Firmicutes bacterium]	1305	1305	100%	0.0	80.05%	NPV26985.1
DNA-directed RNA polymerase subunit beta' [Desulfotomaculum hydrothermale]	1299	1299	100%	0.0	79.24%	WP_008410845.1
DNA-directed RNA polymerase subunit beta' [Desulfofundulus australicus]	1298	1298	100%	0.0	80.05%	WP_073164616.1
DNA-directed RNA polymerase subunit beta' [Desulfofundulus thermocisternus]	1298	1298	100%	0.0	80.05%	WP_027357638.1
DNA-directed RNA polymerase subunit beta' [Calderihabitans maritimus]	1297	1297	100%	0.0	80.86%	WP_088553520.1
DNA-directed RNA polymerase subunit beta' [Desulfotomaculum ferrireducens]	1296	1296	100%	0.0	78.99%	WP_077712918.1
DNA-directed RNA polymerase subunit beta' [Zhaonella formicivorans]	1296	1296	100%	0.0	79.14%	TDF71151.1
DNA-directed RNA polymerase subunit beta' [Desulfohalotomaculum alkaliphilum]	1294	1294	100%	0.0	79.31%	WP_031517421.1
DNA-directed RNA polymerase subunit beta' [Desulfotomaculum putei]	1293	1293	100%	0.0	78.97%	WP_073240195.1
DNA-directed RNA polymerase subunit beta' [Desulfotomaculum nigrificans]	1290	1290	100%	0.0	78.55%	WP_003545469.1
TPA: DNA-directed RNA polymerase subunit beta' [Desulfotomaculum sp.]	1289	1289	100%	0.0	80.05%	HHW42278.1
DNA-directed RNA polymerase subunit beta' [Desulfotomaculum nigrificans]	1288	1288	100%	0.0	78.55%	WP_013809479.1
TPA: DNA-directed RNA polymerase subunit beta' [Syntrophomonadaceae bacterium]	1287	1287	100%	0.0	78.55%	HHW39857.1
TPA: DNA-directed RNA polymerase subunit beta' [Clostridia bacterium]	1286	1286	100%	0.0	79.14%	HHU32330.1
DNA-directed RNA polymerase subunit beta' [Desulfallas thermosapovorans]	1286	1286	100%	0.0	79.31%	WP_166512796.1
DNA-directed RNA polymerase subunit beta' [Desulfofundulus thermosubterraneus]	1286	1286	100%	0.0	80.67%	WP_072871213.1
DNA-directed RNA polymerase subunit beta' [Clostridia bacterium]	1285	1285	100%	0.0	78.42%	WP_174600721.1
DNA-directed RNA polymerase subunit beta' [Desulfofundulus salinum]	1285	1285	100%	0.0	80.67%	WP_121450356.1

Figure 4.13: BLASTp of generated protein shown in Fig 4.12

BLAST® » blastp suite » results for RID-K8EU5V7X014

Home Recent Results Saved Strategies Help

< Edit Search Save Search Search Summary

How to read this report? BLAST Help Videos Back to Traditional Results Page

Job Title	Protein Sequence
RID	K8EU5V7X014 *Search expires on 08-14 02:56 am* Download All
Program	BLASTP Citation
Database	nr See details
Query ID	lcl|Query_29597
Description	None
Molecule type	amino acid
Query Length	801
Other reports	Distance tree of results Multiple alignment MSA viewer

Filter Results

Organism *only top 20 will appear* exclude

Type common name, binomial, taxid or group name

+ Add organism

Percent Identity to **E value** to **Query Coverage** to

Filter Reset

Descriptions Graphic Summary Alignments Taxonomy

Sequences producing significant alignments Download Manage Columns Show 100

select all *100 sequences selected* GenPept Graphics Distance tree of results Multiple alignment

Description	Max Score	Total Score	Query Cover	E value	Per. Ident	Accession
DNA-directed RNA polymerase subunit beta' [Thermincola potens]	1318	1318	100%	0.0	80.93%	WP_013119201.1
DNA-directed RNA polymerase subunit beta' [Thermincola ferriacetica]	1317	1317	100%	0.0	80.93%	WP_052218136.1
DNA-directed RNA polymerase subunit beta' [Firmicutes bacterium]	1305	1305	100%	0.0	80.05%	NPV26985.1
DNA-directed RNA polymerase subunit beta' [Desulfotomaculum hydrothermale]	1299	1299	100%	0.0	79.24%	WP_008410845.1
DNA-directed RNA polymerase subunit beta' [Desulfofundulus australicus]	1298	1298	100%	0.0	80.05%	WP_073164616.1
DNA-directed RNA polymerase subunit beta' [Desulfofundulus thermocisternus]	1298	1298	100%	0.0	80.05%	WP_027357638.1
DNA-directed RNA polymerase subunit beta' [Calderihabitans maritimus]	1297	1297	100%	0.0	80.86%	WP_088553520.1
DNA-directed RNA polymerase subunit beta' [Desulfotomaculum ferrireducens]	1296	1296	100%	0.0	78.99%	WP_077712918.1
DNA-directed RNA polymerase subunit beta' [Zhaonella formicivorans]	1296	1296	100%	0.0	79.14%	TDF71151.1
DNA-directed RNA polymerase subunit beta' [Desulfohalotomaculum alkaliphilum]	1294	1294	100%	0.0	79.31%	WP_031517421.1
DNA-directed RNA polymerase subunit beta' [Desulfotomaculum putei]	1293	1293	100%	0.0	78.97%	WP_073240195.1
DNA-directed RNA polymerase subunit beta' [Desulfotomaculum nigrificans]	1290	1290	100%	0.0	78.55%	WP_003545469.1
TPA: DNA-directed RNA polymerase subunit beta' [Desulfotomaculum sp.]	1289	1289	100%	0.0	80.05%	HHW42278.1
DNA-directed RNA polymerase subunit beta' [Desulfotomaculum nigrificans]	1288	1288	100%	0.0	78.55%	WP_013809479.1
TPA: DNA-directed RNA polymerase subunit beta' [Syntrophomonadaceae bacterium]	1287	1287	100%	0.0	78.55%	HHW39857.1
TPA: DNA-directed RNA polymerase subunit beta' [Clostridia bacterium]	1286	1286	100%	0.0	79.14%	HHU32330.1
DNA-directed RNA polymerase subunit beta' [Desulfallas thermosapovorans]	1286	1286	100%	0.0	79.31%	WP_166512796.1
DNA-directed RNA polymerase subunit beta' [Desulfofundulus thermosubterraneus]	1286	1286	100%	0.0	80.67%	WP_072871213.1
DNA-directed RNA polymerase subunit beta' [Clostridia bacterium]	1285	1285	100%	0.0	78.42%	WP_174600721.1
DNA-directed RNA polymerase subunit beta' [Desulfofundulus salinum]	1285	1285	100%	0.0	80.67%	WP_121450356.1

Figure 4.14: BLASTp of generated protein shown in Fig 4.12

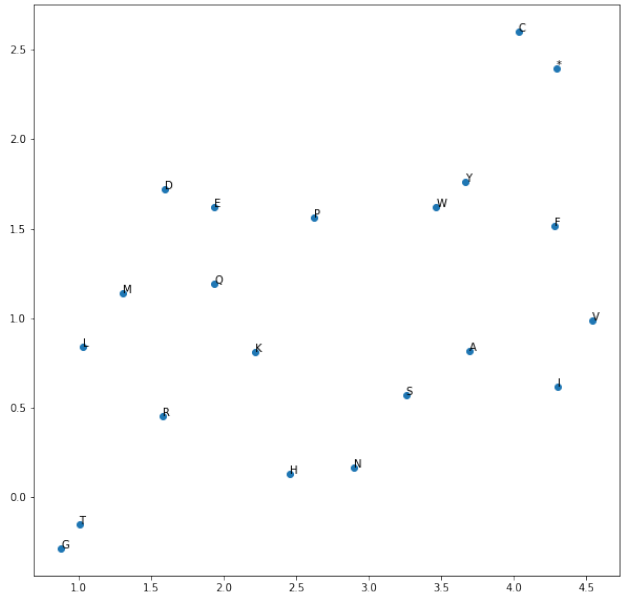


Figure 4.15: Amino Acid Embeddings learnt by the Reformer model

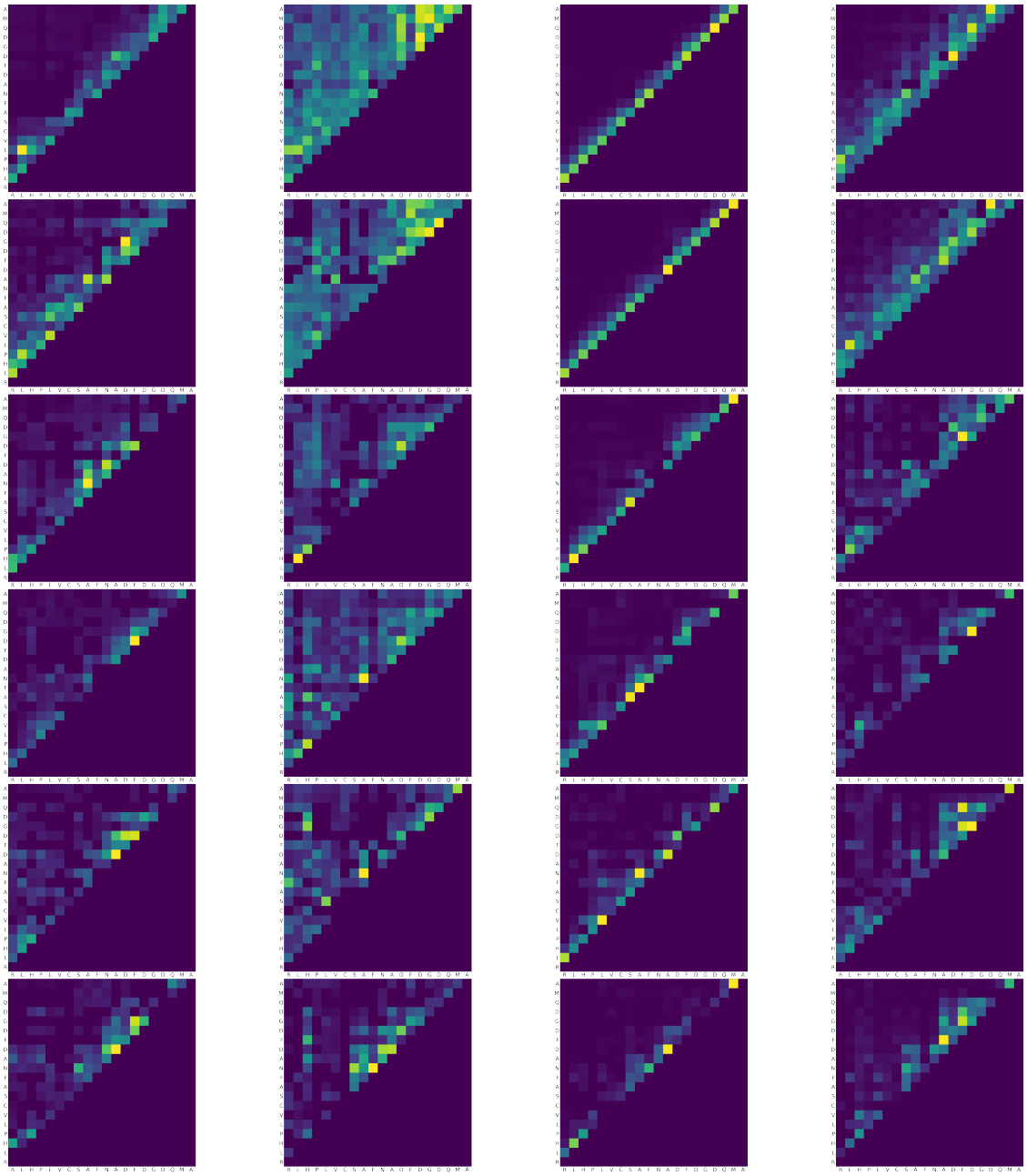


Figure 4.16: Reformer: Attention Visualization at conserved motif NADFDG

```
<gene86>MLDVNNFESMKIGLASPEQIRAWSHGEVKKPETINYRTLKPEKDGLFCERIFGPTRDWEC
HCGKYKRIRYKGIVCDRCGVEVTRSKVRRERMGHIELAAPVSHIWYFKGIPSRLGLLLDMSPRALEKV
IYFASYVVLDPGETPLNKKQLLTEQEYRELIERFGNRFQAGMGAEAIKELLEKIDLDGIKHELEKELKGT
SGQRRVRAIKRLEVVETFRKSGNRPEWMILDALPVIPPDLRPMVQLEGGRFATSDLNDLYRRVINRNN
RLKRLLDLGAPDIIVRNEKRMLQEAVDALIDNGRRGRPVTGPGNRPLKSLSDMLKGKQGRFRQNLLG
KRVDYSGRSVIVVGPQLKMYQCGLPKEMALELFKPFVMKKLVDKGLAHNIKSAKRMVERVRPEVWD
VLEEVIQEHPVLLNRAPTLHRLGIQAFEPVLVEGRAIQIHPLVCTAYNADFDGDQMAVHAPLSGEAQVE
ARLLMLSSNNILSPANGQPIATPSQDMVLGCYYLTAAKKGAKGKGAVFASPEEAILAFESGEVSLHTK
IKVRPEGEFINVTTTAVDSIVDTTVGRLLFNSVLPDEVPFVNDVLKKTALGDLVSQVIAEFNPKLSVKTV
DLLKNLGFTYATQAGASIGIDDVVIPSSKKDIIEKAHDEVAEIEGRFKRGIITDGERYNKVIDVWTHTTNE
VADAMMEALDHLEGGGRPFNPVYMMIDSGARGSKDQIKQLGGMRGLMAKPQKKMTGQLGEIIETP
ITANFREGLSVLEYFISTHGARKGLADTALKTADAGYLTRRLVDVAQEVIITEHDCGTTLGIDMSAIKEGE
EVVEPLSDRILGRVAAEDIYDPITNEVLVAAGEELDEEVASKIAKAGVDKVKIRSVLTCRSRYGVCAKCY
GRNLATGKMIDIGESVGVIAAQSIGEPGTQLTMRTFHTGGVAGDDITQGLPRVVELFEARSPKGQAVIA
EIDGVISIIEEEGDRRIRIKLTNAAKDDVYTVYVVSTRATKVVTGDSVAKGQPLTDGAIDPEKVLRIKGPP
AVQFYLLDEVQRVYRDQGVNVNDKHIEIIVRQMLKKVKVEDAGDTDLLPGATINLFDFENENEEVAAS
GKKPVEASPVLLGVTKAALATDSFISAASFQETTRVLTEAAIKGKKDELVGLKENVIIGKLIPAGTGFKK
YHNLGVETDEEGQEEALALPEPVSTAEKTEATSLLAAPEEAAQK*</gene86>
```


Figure 4.17: Example of generated protein using 2L_LSTM_taxo

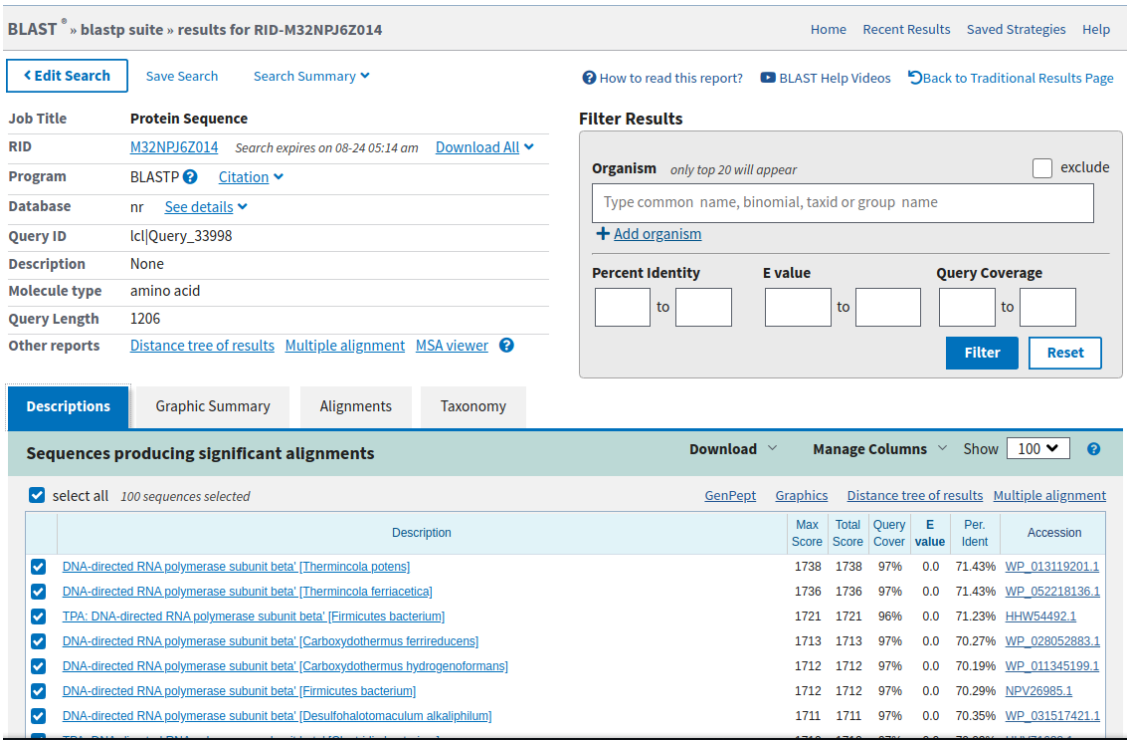


Figure 4.18: BLASTp of generated protein shown in Fig 4.17

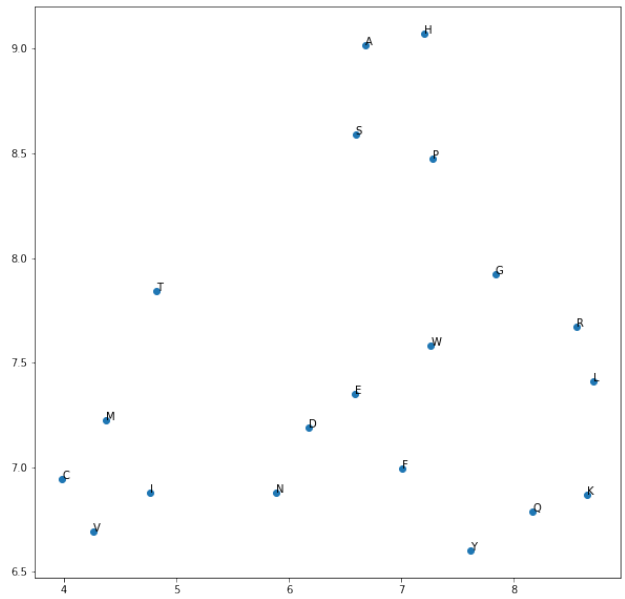


Figure 4.19: Amino Acid Embeddings learnt by 2L_LSTM_taxo

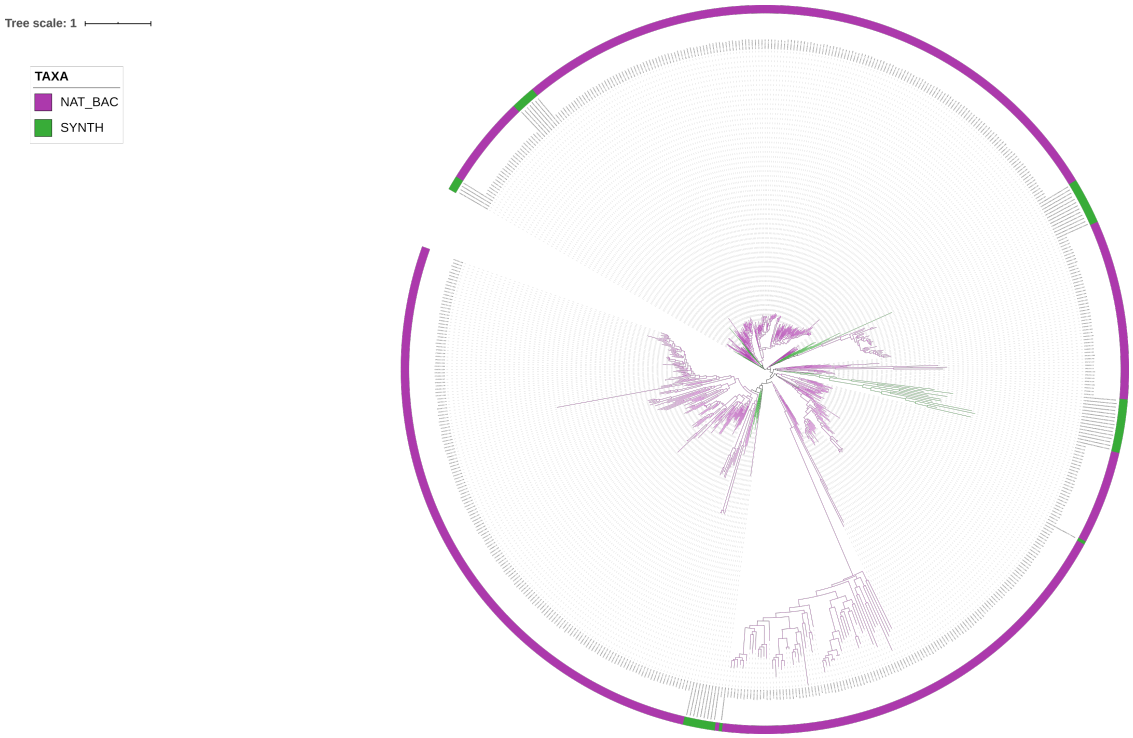


Figure 4.20: 2L_LSTM_taxo: Phylogenetic Tree of Natural and Synthetic Sequences

```
<gene86>MKALLDLFKQVNQEEQFDAIKVGIASPEKIRSWSYGEVKKPETINYRTFKPERDG
LFCAKIFGPIKDYECLCGKYKRLKHRGVICEKCGVEVTLTKVRRERMGHIELASPVAHIWFLK
SLPSRIGLILDMTLRDIERILYFESYIVVDPGMTPLERGQLLTEDQYFSALEEYGDDFDAKMGA
EAVQELLMEIDLESEAEKLRSEMSATKSEKKLKKLKKLLKRLKVIEGFIESRNKSENPIWMVL
EVIPVIPPELRPLVPLDGGRFATSDLNDLYRRVINRNNRLKRLLELNAPDIIVRNEKRMLQESV
DALLDNGRRGRAITGSNKRPLKSLADMIKGKQGRFRQNLLGKRVDYSGRSVIVVGPTLRLHQ
CGLPKKMALELFKPFIFSKLQLRGLATTIKAAKKMVERGEAVVWDILEEVIREHPVMLNRAP
TLHRLGIQAFEPVLIEGKAIQLHPLVCTAFNADFDGDQMAVHVPLSLEAQLEARVLMMSTNN
ILSPANGKPIIVPSQDIVLGLYYLTRERVNAKGEGMVFADIEEVHRAYDNGEVDIHAKIKVRIRE
VVVEDGDIVTLGRVITKDDVLPPGAIIFEELNRKEVGKLVSFDVLDTVVNDSGEIIELTLDEAGEL
VTVESLDEIKELGFKYATMAGISIAVSDMVIPPEKKEIISDARKEVKEIEQQYSSGLVTQGERYN
KVVDIWGRAGDKISDLMQKQFTDGSKNKSYNNIYMMANSGARGSIAQIRQLAGMRGLMAK
PSGEIIETPITANFREGLNVLQYFISTHGARKGLADTALKTANSGYLTRRLVDVAQDLVVTEIDC
GTDEGLTMTPLIEGGDVVEPLGERILGRVAAEDVYHPESQAVVVAAGTLIDEDVVDRVEAAGV
DEVKIRTVLTCDTKYGICAKCYGRDLGRGTLVNVGEAVGVIAAQSIGEPGTQLTMRTFHIGGAA
SRAAAASNVQIKNSGTVRLHNLKYVSKNNVKISVASGTVVVTSRSGELAIIDEKGRERERHKV
PYGATLAVKDGQKVSKGDVVARGPRATAGSDLTEGSVNPHDILKVKGPAAVQEYLVNEVQEV
YRLQGVKINDKHFEVIVRQMLRKVEITNSGDSEFLPGEQVERSEVLDENDRLLAEGGRPVTAG
ETLVAGTDLPAAGKVVTAGSKGTEGTYNTYVSTYVEAYEAVELGKTLITELGIDRIKVRSVLTCE
TRRGVCAKCYGRNLATGRLVEVGEAVGIIAAQSIGEPGTQLTMRTFHTGGVAGDDITQGLPRVE
ELFEARKPKGLAPIAEVSGVVSFGKETKGKRRLIITPKDDGIEPYELIVKKKSKLIVTDDNGRER
ERYKVPYGATLNVKDGQKVKAGEVLARIPQESSKTKDITGGLPRVAELFEARRPKDAAIIAEIS
GTVSFGKDTKGKRRLVITPVDGEVYEYEIPKGKKIVVTENDRVKKGDKITEGAINPHDILEILGV
EALARYIVNEVQDVYRLQGVKINDKHIEVIVRQMLRRVRITDPGDTKLLPGDQVDRNEVLDEN
QKVLAEGKEPATARHLLAGTTGAILSKAGAVLATPGDLVKAGEVLTKEADDLVEPYGADKVQV
RSVLTCKSRHGICAKCYGRDLGRGSLVNIGEAVGVIAAQSIGEPGTQLTMRTFHIGGAASRAA
AASSIQVKNKGSVRLKNVKFVNSTNVRLVDLKGKEVVITSRGAEVVVGRTKEIKIIDKTGKKRE
YLVPKDKNINVHEGEVVNKGDYIVDGPADPHDILRIRGVEALARYLVNEVQEVYRLQGVKINDK
HIEVIVRQMLRRVEITDAGDSNLLPGELVERYEFEEENDKVIAEGKRPAPAQARPLLLGITKASL
ATESFISAASFQETTRVLTEAAVSGKVDDLRGLKENVIVGRLIPAGTGLAYHRERRRRRAKGPP
APAESEPAEAEAEEIAEAAAEAAAAALLENDADDALLDGADA*</gene86>
```


Figure 4.21: Example of generated protein using 2L_LSTM_Gumbel

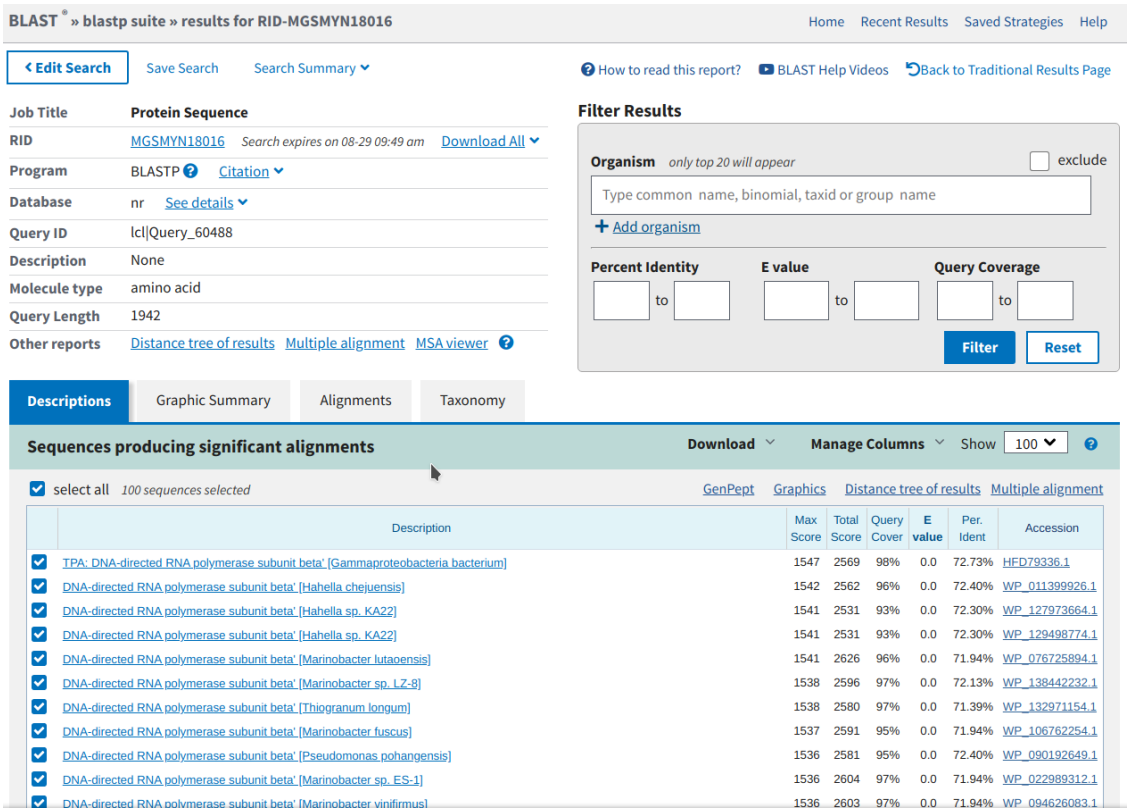


Figure 4.22: BLASTp of generated protein shown in Fig 4.21

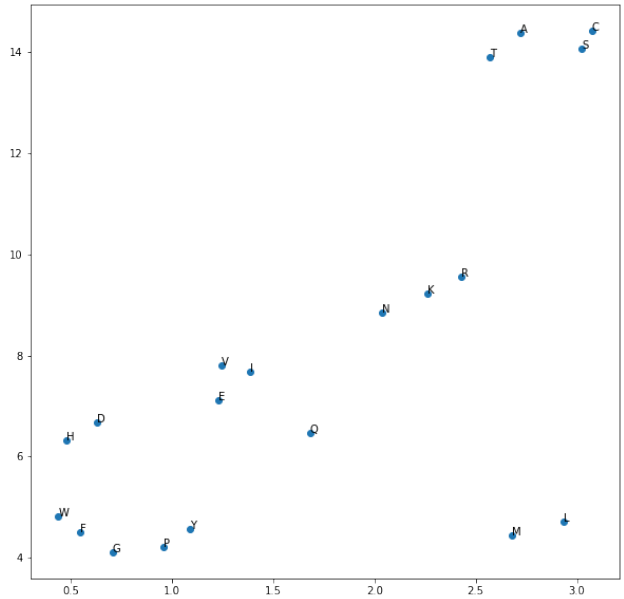


Figure 4.23: Amino Acid Embeddings learnt by 2L_LSTM_Gumbel

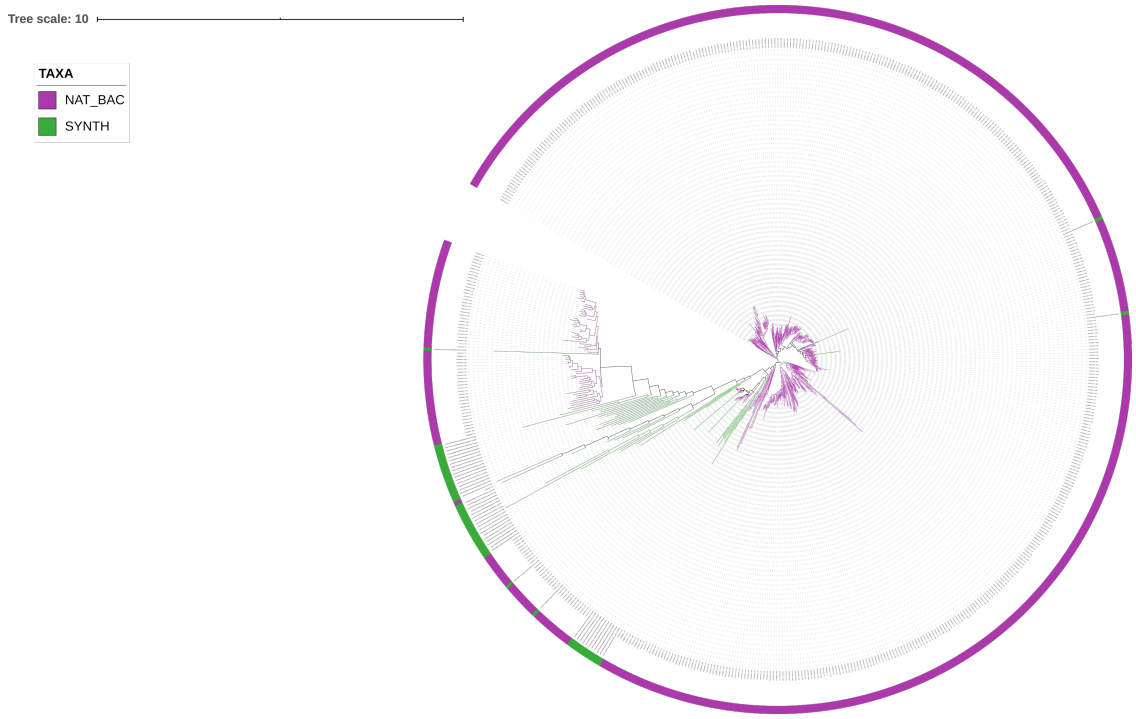


Figure 4.24: 2L_LSTM_Gumbel: Phylogenetic Tree of Natural and Synthetic Sequences


```
<gene86>MINVNKRNSQKININFEAIKLALASPEDILSWSYGEVTKPETINYRTFKPERDGLFCERIF
GPVKDYECHCGKYKGIRYKGVICDRCGVEVTHSVVRRERMGHIELAVPVVHTWYLRSTPSRIGLILNLS
VRELERVLYYESYIITEPGSTPLKRKQLLSEKEYREYREKYGAFFTAGMGAEAVKKLLEEIDLDREKEK
LKKEIASVNSITRKKKATKRLKVVEAFRKSGNRPEWMILLVVPVLPPDLRPLVPLDGGRFATSDLNDLY
RRVINRNNRLKRLLDLNAPDIIVRNEKRMLQEAVDALFDNGRRGRVITGSNKRPLKSLSDMIKGKQGRF
RQNLLGKRVDYSGRSVIVVGPELRLHQCGLPKKMALELFKPFIFSKLELRGLASTIKAAKKMVEREGPE
VWDILEEVIREHPVLLNRAPTLHRLGIQAFEPILVEGKAIQIHPLVCAAYNADFDGDQMAVHVPLSVEA
QMEAHTLMLATNNILSPAHGRPIAMPSQDMVLGLYYLTKSRPGAKGEGKIFSSPEEILIAYELGKIDLQ
ALIKVRIEDKLIETTVGRILFNEVLPEEVPFVNEIMDKKALGSIVSICYQELGNQRTVKMLDGIKELGY
NYATKAGISISIEDMKVPTDKPKLIAEADAKVEKVEKQYKRGLISENERYNKVIDIWNKATDEVTKAVT
QNFDRMNPVYIMAKSGARGNMQQIRQLAGMRGLMADPSGRIIELPIKANFREGLTVLEYFISTHGARKG
LADTALKTADSGYLTRRLVDVAQDVIVNEEDCGTEKGIEVNPLVEGGEIIEPLRDRILGRVSVDDVYDP
MTGEIIVPANEEITEEKAQAIDKAGIKSVKIRSVLTCKTRRGVCSKCYGRDLGRGQLVNIGEAVGVIAA
QSIGEPGTQLTMRTFHIGGAASKGAEQSNFEARYAGKVKFENLSTVKNQTGEIVVLSRQGELSLIDESG
RERERYQIPYGANLKVKEGERVEPGEVLAEWDPYTNIILTERSGKVKYGDIIEGITMQEEIDEVTGLSR
KVIIEHKGEKLQPQIIIKDDDGVTLHPYIPIPARLKVEEGEEVEAGDPLTKGPIDPHDILRIKGAEAVQ
KYLVMEVQKVYRLQGVEINDKHIEIIVRQMLRKVRVELLGDTDLLPGSLVDVHQLLEVNERVKIEGKSP
AELKPILLGITKASLATESFISAASFQETTRVLTEASIAGKVDPLFGLKENVIVGRLIPAGTGLPRYQD
ERRKKPALEPPAAAEPVEEG*</gene86>
```


Figure 4.25: Example of generated protein using 2L_LSTM_Gumbel_Taxo

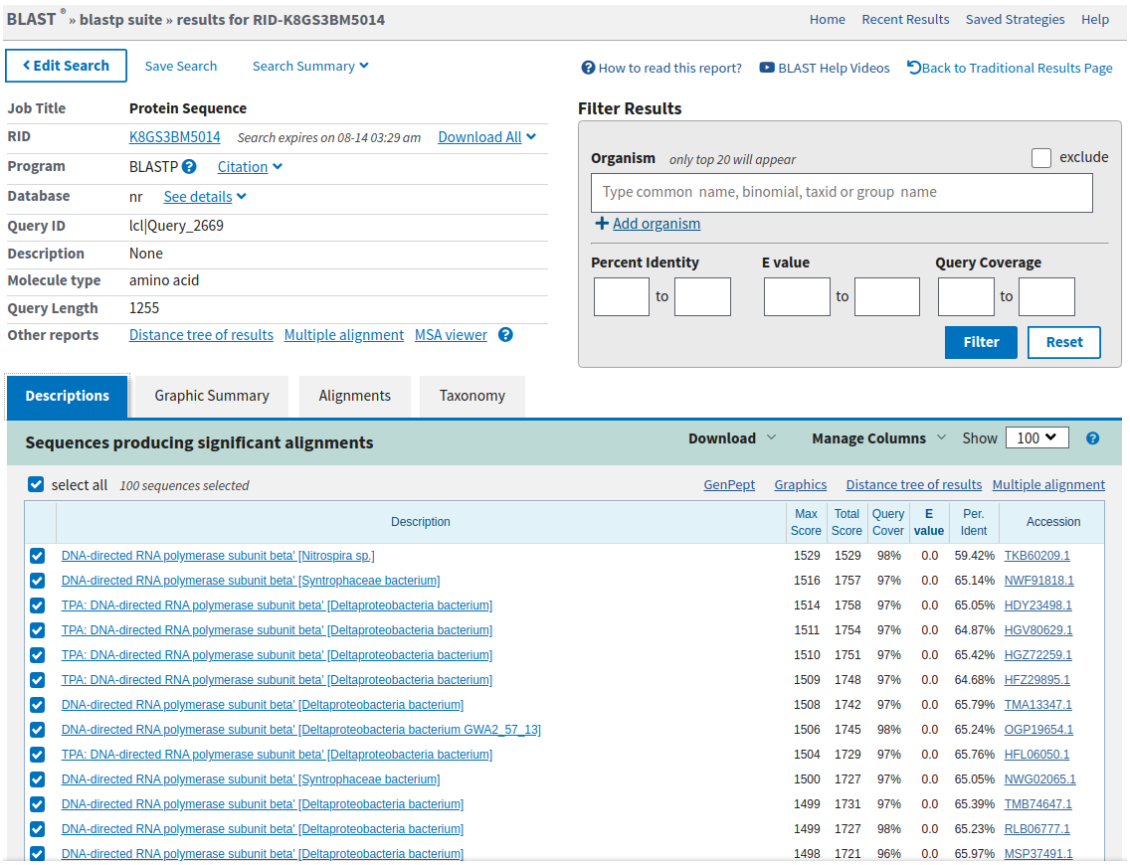


Figure 4.26: BLASTp of generated protein shown in Fig 4.25

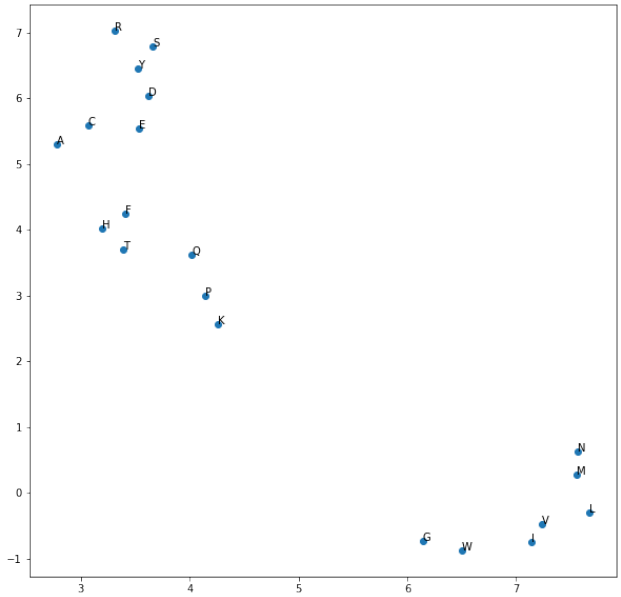


Figure 4.27: Amino Acid Embeddings learnt by 2L_LSTM_Gumbel_Taxo

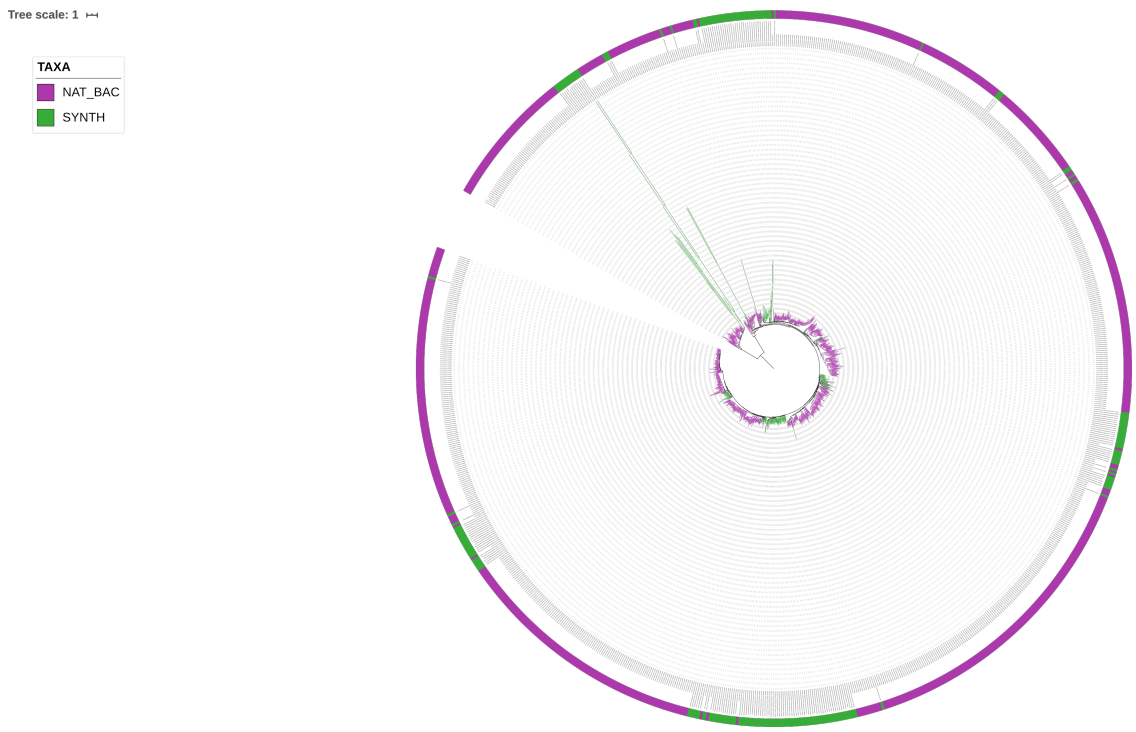


Figure 4.28: 2L_LSTM_Gumbel_Taxo: Phylogenetic Tree of Natural and Synthetic Sequences

Chapter 5

Summary and Future Work

Through careful designing of both data and model, deep learning based causal auto-regressive models were able to generate high quality proteins from a particular protein family. These protein generated should be looked at as protein candidates rather than actual proteins as the feasibility of the proteins have not been tested in the wet-lab experimental setting. But, looking at the fact that the model is able to learn amino acid physio-chemical properties as well as looking at the quality of the phylogenetic trees generated, the proteins predicted/sampled should be well-informed samples. The more interpretable attention-based reformer models open up multiple avenues where the interpretable insights could prove valuable for many tasks. Although, this work has shown the promise of applying deep learning for protein generation, the following tasks given below could help in extending its use even further.

- Build models for proteins from different functional groups
- Build and analyze a reformer model pretrained on all protein types
- Multi-task learning where one of the decoders predicts the protein secondary structure
- Make the gumbel-softmax parameter τ dependent on the past sequence information
- Improve/Modify the charRNN-taxo model to enable out of distribution sampling (generate proteins of phyla not in the dataset)

Bibliography

[1] David H Ackley, Geoffrey E Hinton, and Terrence J Sejnowski. "A learning algorithm for Boltzmann machines". In: *Cognitive science* 9.1 (1985), pp. 147–169.

[2] Stephen F Altschul et al. "Gapped BLAST and PSI-BLAST: a new generation of protein database search programs". In: *Nucleic acids research* 25.17 (1997), pp. 3389–3402.

[3] Alexandr Andoni et al. "Practical and optimal LSH for angular distance". In: *Advances in neural information processing systems*. 2015, pp. 1225–1233.

[4] Frances H Arnold. "Design by directed evolution". In: *Accounts of chemical research* 31.3 (1998), pp. 125–131.

[5] Dami Choi et al. "On empirical comparisons of optimizers for deep learning". In: *arXiv preprint arXiv:1910.05446* (2019).

[6] Athel Cornish-Bowden. "Nomenclature for incompletely specified bases in nucleic acid sequences: recommendations 1984." In: *Nucleic acids research* 13.9 (1985), p. 3021.

[7] Zihang Dai et al. "Transformer-xl: Attentive language models beyond a fixed-length context". In: *arXiv preprint arXiv:1901.02860* (2019).

[8] Jacob Devlin et al. "Bert: Pre-training of deep bidirectional transformers for language understanding". In: *arXiv preprint arXiv:1810.04805* (2018).

[9] John Duchi, Elad Hazan, and Yoram Singer. "Adaptive subgradient methods for online learning and stochastic optimization." In: *Journal of machine learning research* 12.7 (2011).

[10] Joseph Felsenstein and Joseph Felenstein. *Inferring phylogenies*. Vol. 2. Sinauer associates Sunderland, MA, 2004.

[11] Aidan N Gomez et al. "The reversible residual network: Backpropagation without storing activations". In: *Advances in neural information processing systems*. 2017, pp. 2214–2224.

[12] Ian Goodfellow et al. "Generative adversarial nets". In: *Advances in neural information processing systems*. 2014, pp. 2672–2680.

[13] Alex Graves. "Generating sequences with recurrent neural networks". In: *arXiv preprint arXiv:1308.0850* (2013).

[14] Aditya Grover and Jure Leskovec. "node2vec: Scalable feature learning for networks". In: *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. 2016, pp. 855–864.

[15] Emil Julius Gumbel. *Statistical theory of extreme values and some practical applications: a series of lectures*. Vol. 33. US Government Printing Office, 1948.

[16] Dan Hendrycks and Kevin Gimpel. "Gaussian error linear units (gelus)". In: *arXiv preprint arXiv:1606.08415* (2016).

[17] Ari Holtzman et al. "The curious case of neural text degeneration". In: *arXiv preprint arXiv:1904.09751* (2019).

[18] Doug Hyatt et al. "Prodigal: prokaryotic gene recognition and translation initiation site identification". In: *BMC bioinformatics* 11.1 (2010), p. 119.

[19] Jinho Im, Byungkyu Park, and Kyungsook Han. "A generative model for constructing nucleic acid sequences binding to a protein". In: *BMC genomics* 20.13 (2019), pp. 1–13.

[20] Eric Jang, Shixiang Gu, and Ben Poole. "Categorical reparameterization with gumbel-softmax". In: *arXiv preprint arXiv:1611.01144* (2016).

[21] Andrej Karpathy. "The unreasonable effectiveness of recurrent neural networks, 2015". In: *URL http://karpathy. github. io/2015/05/21/rnn-effectiveness* (2016).

[22] Tero Karras, Samuli Laine, and Timo Aila. "A style-based generator architecture for generative adversarial networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2019, pp. 4401–4410.

[23] Angelos Katharopoulos et al. "Transformers are RNNs: Fast Autoregressive Transformers with Linear Attention". In: *arXiv preprint arXiv:2006.16236* (2020).

[24] Stuart A Kauffman et al. *The origins of order: Self-organization and selection in evolution*. Oxford University Press, USA, 1993.

[25] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[26] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. "Reformer: The efficient transformer". In: *arXiv preprint arXiv:2001.04451* (2020).

[27] Taku Kudo. "Subword regularization: Improving neural network translation models with multiple subword candidates". In: *arXiv preprint arXiv:1804.10959* (2018).

[28] Taku Kudo and John Richardson. "Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing". In: *arXiv preprint arXiv:1808.06226* (2018).

[29] Johannes Linder et al. "A Generative Neural Network for Maximizing Fitness and Diversity of Synthetic DNA and Protein Sequences". In: *Cell Systems* (2020).

[30] Craig D Livingstone and Geoffrey J Barton. "Protein sequence alignments: a strategy for the hierarchical analysis of residue conservation". In: *Bioinformatics* 9.6 (1993), pp. 745–756.

[31] Ilya Loshchilov and Frank Hutter. "Decoupled weight decay regularization". In: *arXiv preprint arXiv:1711.05101* (2017).

[32] Thomas Madden. "The BLAST sequence analysis tool". In: *The NCBI Handbook [Internet]. 2nd edition*. National Center for Biotechnology Information (US), 2013.

[33] Chris J Maddison, Daniel Tarlow, and Tom Minka. "A* sampling". In: *Advances in Neural Information Processing Systems*. 2014, pp. 3086–3094.

[34] Leland McInnes, John Healy, and James Melville. "Umap: Uniform manifold approximation and projection for dimension reduction". In: *arXiv preprint arXiv:1802.03426* (2018).

[35] H Brendan McMahan and Matthew Streeter. "Adaptive bound optimization for online convex optimization". In: *arXiv preprint arXiv:1002.4908* (2010).

[36] Charles D Michener and Robert R Sokal. "A quantitative approach to a problem in classification". In: *Evolution* 11.2 (1957), pp. 130–162.

[37] Nuala A O'Leary et al. "Reference sequence (RefSeq) database at NCBI: current status, taxonomic expansion, and functional annotation". In: *Nucleic acids research* 44.D1 (2016), pp. D733–D745.

[38] Christopher Olah. "Understanding lstm networks, 2015". In: *URL http://colah. github. io/posts/2015-08-Understanding-LSTMs* (2015).

[39] Donovan H Parks et al. "A complete domain-to-species taxonomy for Bacteria and Archaea". In: *Nature Biotechnology* (2020), pp. 1–8.

[40] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. "On the difficulty of training recurrent neural networks". In: *International conference on machine learning*. 2013, pp. 1310–1318.

[41] Boris T Polyak. "Some methods of speeding up the convergence of iteration methods". In: *USSR Computational Mathematics and Mathematical Physics* 4.5 (1964), pp. 1–17.

[42] Morgan N Price, Paramvir S Dehal, and Adam P Arkin. "FastTree 2–approximately maximum-likelihood trees for large alignments". In: *PloS one* 5.3 (2010), e9490.

[43] Alec Radford et al. "Language models are unsupervised multitask learners". In: *OpenAI Blog* 1.8 (2019), p. 9.

[44] Sashank J Reddi, Satyen Kale, and Sanjiv Kumar. "On the convergence of adam and beyond". In: *arXiv preprint arXiv:1904.09237* (2019).

[45] Adam J Riesselman et al. "Accelerating Protein Design Using Autoregressive Generative Models". In: *bioRxiv* (2019), p. 757252.

[46] Frank Rosenblatt. *The perceptron, a perceiving and recognizing automaton Project Para*. Cornell Aeronautical Laboratory, 1957.

[47] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. "Learning representations by back-propagating errors". In: *nature* 323.6088 (1986), pp. 533–536.

[48] Naruya Saitou and Masatoshi Nei. "The neighbor-joining method: a new method for reconstructing phylogenetic trees." In: *Molecular biology and evolution* 4.4 (1987), pp. 406–425.

[49] Ludovic Sauguet. "The extended "two-barrel" polymerases superfamily: structure, function and evolution". In: *Journal of molecular biology* 431.20 (2019), pp. 4167–4183.

[50] Hava T Siegelmann. "Computation beyond the Turing limit". In: *Science* 268.5210 (1995), pp. 545–548.

[51] Fabian Sievers et al. “Fast, scalable generation of high-quality protein multiple sequence alignments using Clustal Omega”. In: *Molecular systems biology* 7.1 (2011), p. 539.

[52] C Stephen. “Kleene. Representation of events in nerve nets and finite automata”. In: *Automata studies* (1956).

[53] Yi Tay et al. “Sparse Sinkhorn Attention”. In: *arXiv preprint arXiv:2002.11296* (2020).

[54] Wilson L Taylor. ““Cloze procedure”: A new tool for measuring readability”. In: *Journalism quarterly* 30.4 (1953), pp. 415–433.

[55] T. Tieleman and G. Hinton. *Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude*. COURSERA: Neural Networks for Machine Learning. 2012.

[56] Ashish Vaswani et al. “Attention is all you need”. In: *Advances in neural information processing systems*. 2017, pp. 5998–6008.

[57] Sinong Wang et al. “Linformer: Self-Attention with Linear Complexity”. In: *arXiv preprint arXiv:2006.04768* (2020).

[58] Ashia C Wilson et al. “The marginal value of adaptive gradient methods in machine learning”. In: *Advances in neural information processing systems*. 2017, pp. 4148–4158.

[59] Hans-Georg Zimmermann, Christoph Tietz, and Ralph Grothmann. “Forecasting with recurrent neural networks: 12 tricks”. In: *Neural Networks: Tricks of the Trade*. Springer, 2012, pp. 687–707.

Appendices

Appendix A

First Appendix

A.1 Train-Validation Plots

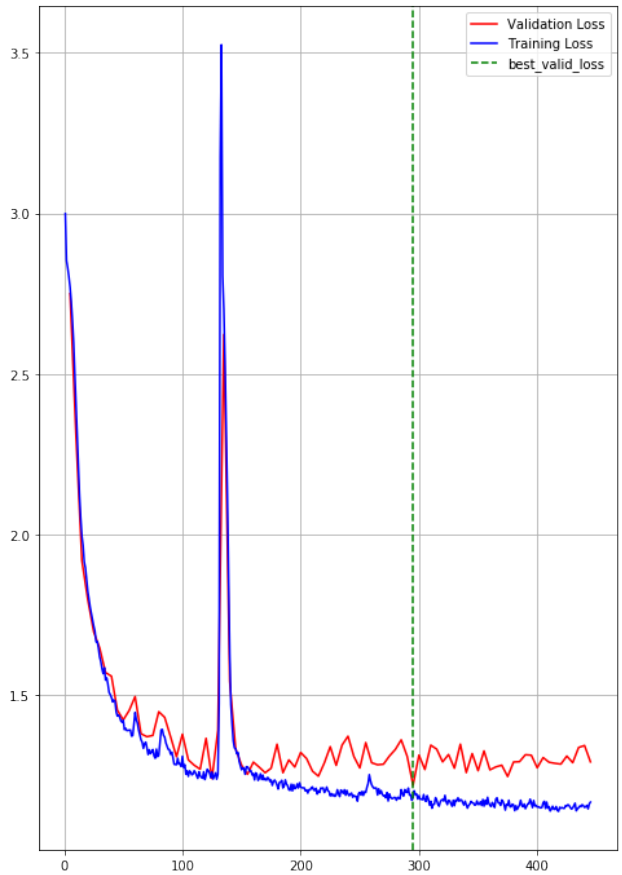


Figure A.1: 2L_LSTM: Train-Validation Loss Plot

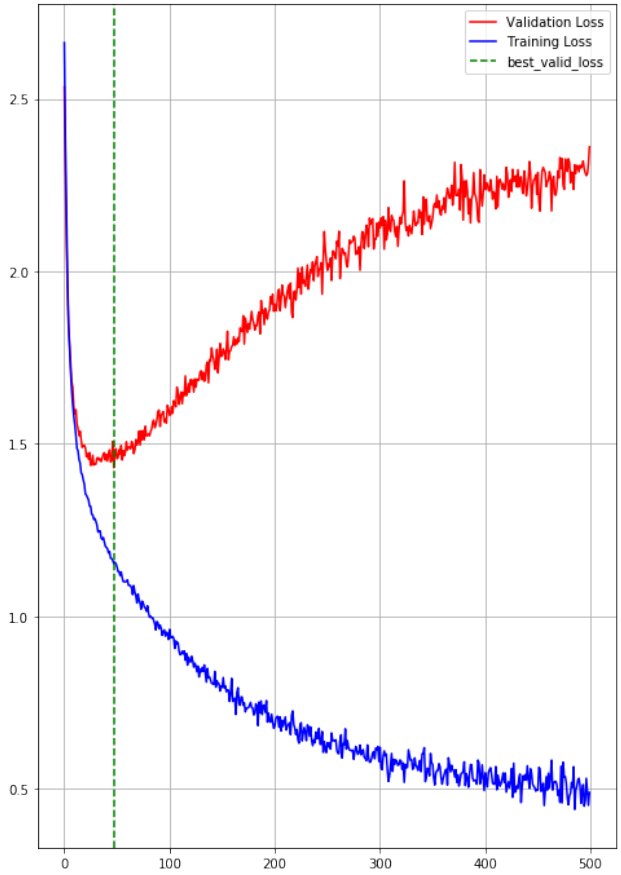


Figure A.2: Reformer: Train-Validation Loss Plot

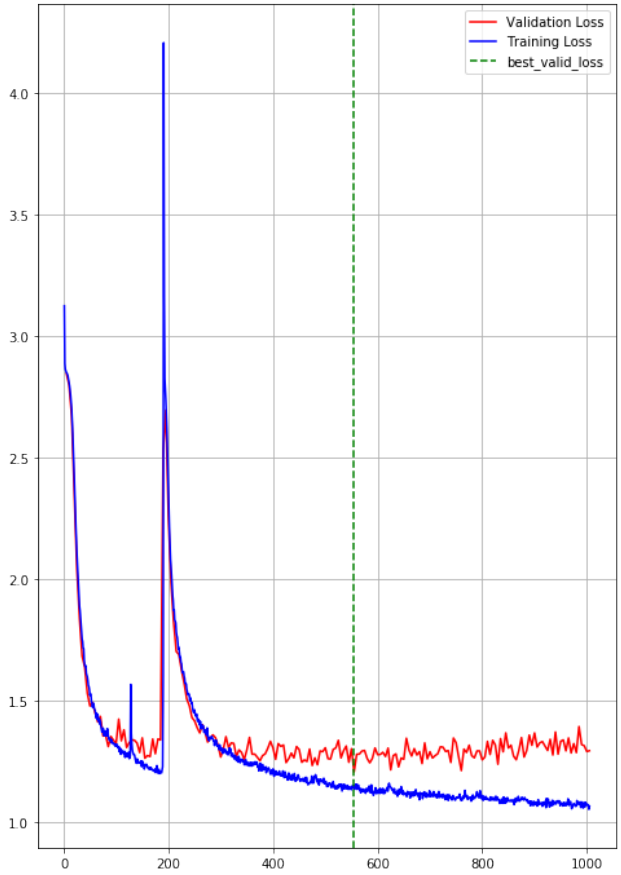


Figure A.3: 2L_LSTM_taxo: Train-Validation Loss Plot

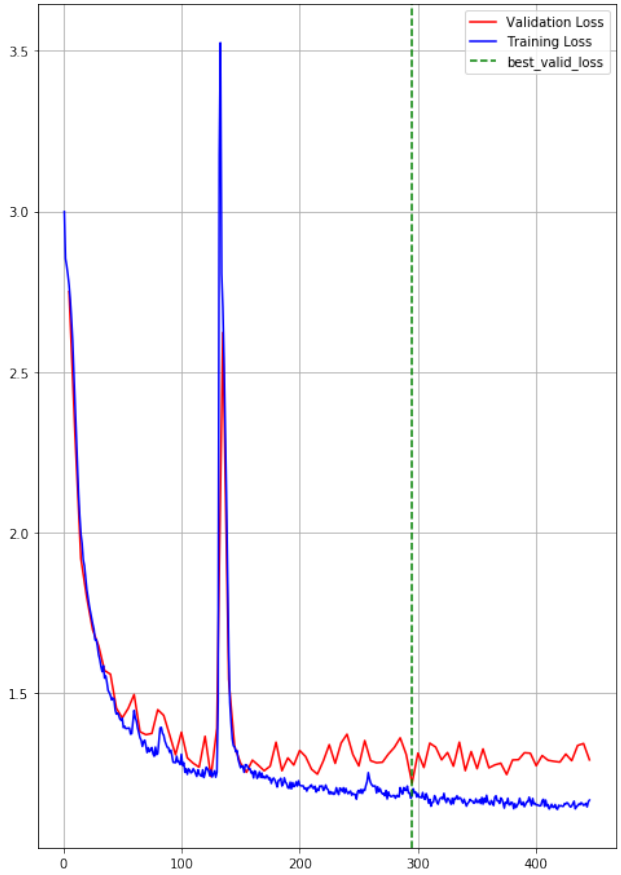


Figure A.4: 2L_LSTM_Gumbel: Train-Validation Loss Plot

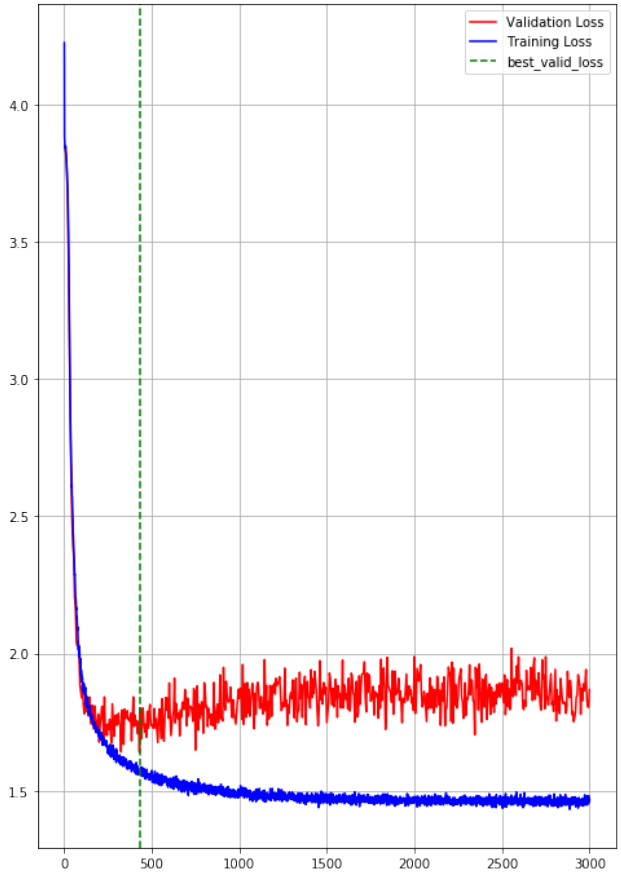


Figure A.5: 2L_LSTM_Gumbel_Taxo: Train-Validation Loss Plot