Autoencoder for a Rapid Solution to Protein Design

Trenton Bricken

April 2019

1 Abstract

Approaches to protein design are either provable or heuristic with both approaches introducing significant tradeoffs between speed and optimality. This paper presents a new heuristic approach to protein design that uses continuous rather than discrete gradient descent for protein design that is potentially faster and more optimal that other heuristic approaches. This is made possible by the training of an autoencoder that can learn a high dimensional manifold of the relationships between protein sequence and structure. The successful mapping of a protein's sequence and tertiary structure into this latent manifold not only creates opportunities for protein design but also opens the door for exciting possibilities to learn new relationships between proteins and generate novel proteins either at random or related to a given example.

2 Introduction

While many methods for discrete optimization exist, continuous optimization using gradient descent is more efficient for finding minima. This is because we only have to evaluate the gradient at a single point to know the best direction rather than at many points around our current one such as in local beam search[1]. Moreover, with the rise of neural networks and the need for backpropagation to learn weights, many novel stochastic gradient descent optimizers[2] have been developed which are more capable of escaping saddle points in high dimensional spaces[3].

[1] Stuart Russell and Peter Norvig "Artificial Intelligence A Modern Approach", 3rd edition, Prentice Hall, Chapter 4.2, (2009)

[2] Examples include RMSProp and Adam Goodfellow et al. "Deep Learning" (2016) Ch. 8

[3] Contrary to previous assumptions, in non-convex high dimensional spaces there are many more saddle points than local minima. This is because, for a local minima to exist, the gradient needs to be in the same direction for every dimension which is highly unlikely. In fact, the expected ratio of the number of saddle points to local minima grows exponentially with number of dimensions (Goodfellow et al. "Deep Learning" (2016) Ch. 8). The success of deep learning models and their ability to converge has given empirical validation to this idea.

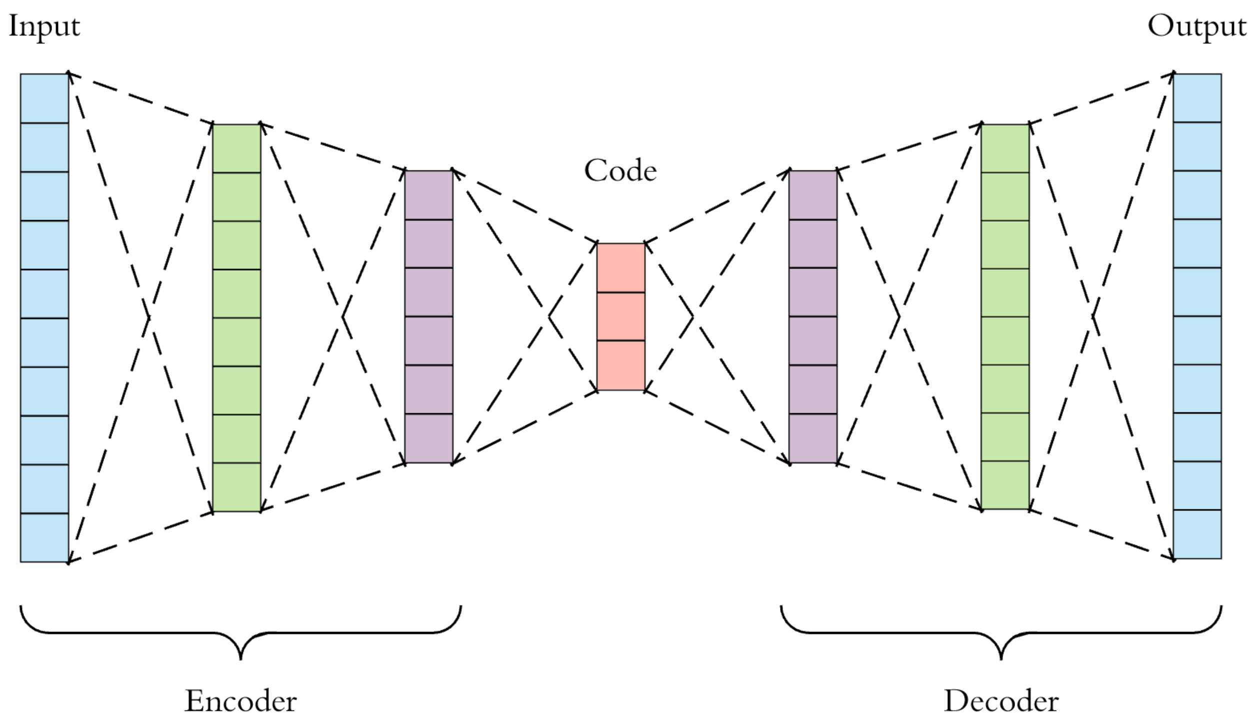

Figure 1: This is an example of an autoencoder architecture. Image taken from https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798

Gradient descent is not only capable of converging faster than its discrete counterparts but also naturally able to handle continuous backbone and rotamer conformations for protein design. However, the barrier to using gradient descent to design to a given protein structure lies in the fact that amino acids are discrete. A novel way to solve this problem is proposed through mapping the discrete amino acids into a continuous, latent space before using gradient descent for optimization in this latent space. Given the amount of complexity and high order interactions that are encoded inside a protein's structure and sequence, the most promising way to perform dimensionality reduction into the latent space without losing information is using an autoencoder[4]. Autoencoders are deep neural network architectures that perform supervised learning to train two neural networks: one network that encodes the data into a latent space, and another that learns to decode this latent space back into its original form. Assuming that the autoencoder can learn to encode and decode a protein's 3D structure and sequence with enough fidelity[5] we can select a 3D structure we

[4] Goodfellow et al. "Deep Learning" (2016) Ch. 14
[5] This is something that does not appear to have ever been attempted before

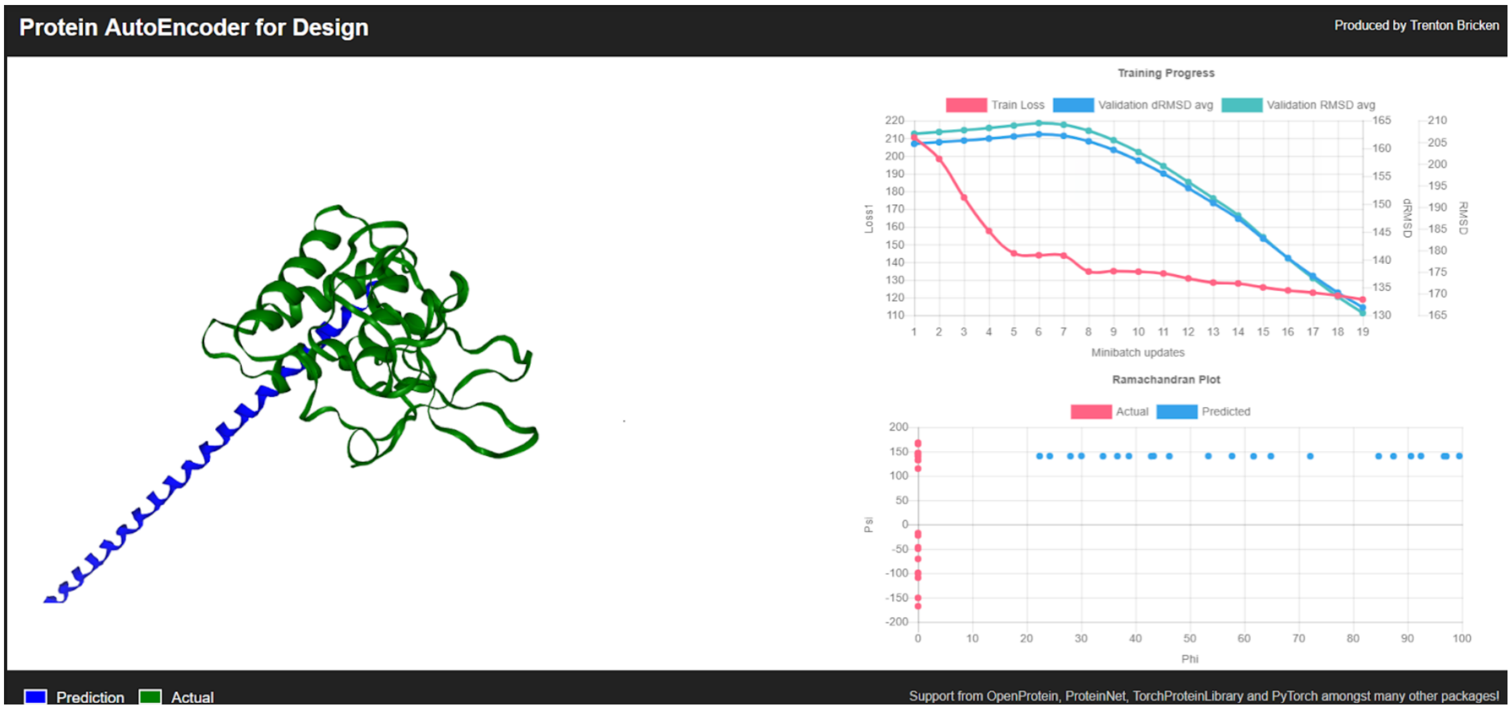
Protein AutoEncoder for Design
Produced by Trenton Bricken
Training Progress
Train Loss
Validation dRMSD avg
Validation RMSD avg
Loss1
dRMSD
RMSD
Minibatch updates
Ramachandran Plot
Actual
Predicted
Psi
Phi
Prediction
Actual
Support from OpenProtein, ProteinNet, TorchProteinLibrary and PyTorch amongst many other packages!

Figure 2: A screenshot of the web application used to visualize the neural network as it is being trained over time along with the predicted protein amino acids, structure, and Ramachandran plot in comparison to the actual protein to e generated. This screenshot was taken shortly after the network was initialized which is why the prediction is a very long alpha helix. NB. The Ramachandran plot in the bottom right had a bug that made it look so incorrect for the acutal structure that has since been fixed.

would like to have an amino acid sequence fold into, initialize our latent variables to a template structure or randomly, and then perform gradient descent on these latent variables to find a minima, evaluating the latent variables at each gradient descent step against a cost function. Upon convergence we can convert this latent space back into the sequence, rotamers, and backbone positions for a prediction of what sequence will fold into the desired structure.

3 Autoencoder Approach

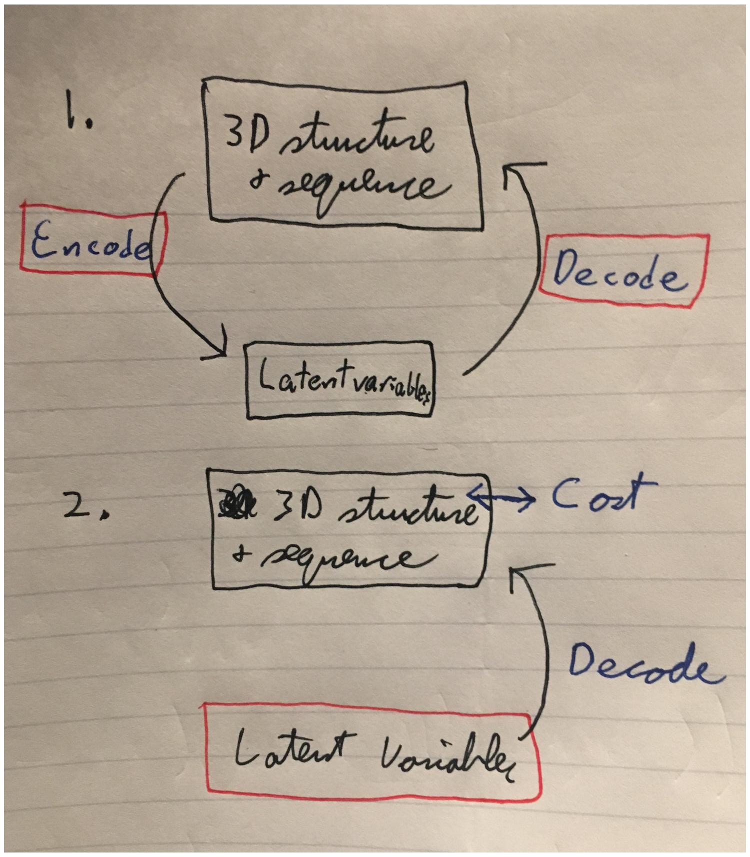

Figure 3: The two steps necessary for the autoencoder to be used to design proteins.

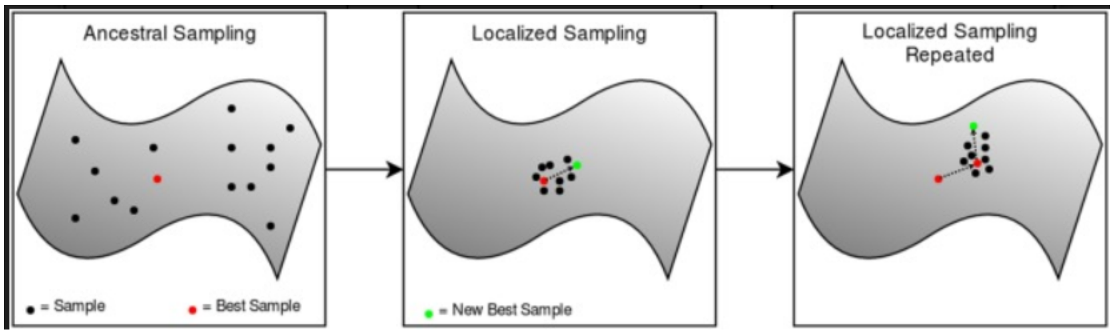

Figure 4: An example of sampling from a manifold and different techniques for doing so. Figure Greener et al. "Design of metalloproteins and novel protein folds using variational autoencoders" Sci Rep. 2018; 8: 16189

The two steps for making the protein design possible are explained below and reference Figure 3:

Step 1. We train an autoencoder to faithfully reproduce the inputted protein's 3D structure and sequence. The two components of the autoencoder, the encoder and decoder, are in red boxes in Figure 3 because they are what is being updated or "trained" through gradient descent via backpropagation. This training process learns a high dimensional manifold of "protein space", in particular high order patterns between sequences and structures.

Step 2. Once this manifold is learnt, we can use the autoencoder to convert a protein's structure and sequence into this latent space to make the sequence continuous. Therefore, the latent space is in a red box here as gradient descent is being performed on it while we freeze the decoder network weights, no longer training them and using the decoder only to translate the latent manifold into a predicted protein structure and sequence.

The exact neural network architectures used for the encoder and decoder networks are outlined in the Methods section later.

If the autoencoder can be trained to accurately encode and decode the inputted protein's sequence and structure this will be valuable beyond just applications to protein design. Two clear other applications are:

Firstly, for the generation of novel proteins that are either similar to an existing given example or are randomly generated from a part of the learned manifold. See Figure 4 for a visualization of what sampling from a manifold looks like and some of the different strategies that can be used. This novel protein generation could be achieved by putting an existing protein through the encoder to initialize the latent space and then sample from this latent space with some random noise to generate similar proteins or by initializing randomly and then generating proteins from the closest portion of the manifold.

Secondly, the latent space of different proteins can act as a sort of word embedding for them. Word embedding is a technique used in the natural language processing community to "embed" words as high dimensional vectors which can then represent relationships between different words. For example the word embedding technique word2vec[6] found that the Euclidean distance between the word embedding vectors for king and queen were very similar to the distances between the vectors for boy and girl and other gendered nouns more broadly. This word embedding done for proteins could find homology between proteins that combined both sequence and structural similarities allowing us to better identify patterns across the protein state space and refine our vocabulary of proteins. Initial results in this direction have recently been achieved for word embedding protein sequences with exciting results[7].

4 Results

During this project over 3,000 lines of code have been written in Python using the PyTorch[8] package to build the data pipeline and autoencoder neural network.

While the model and training process have been successfully built, it has proven to be very difficult to get the network to learn to reproduce its given input's sequence or tertiary structure, even when just trying to overfit the model on a very small subset of the data. After much experimentation with different network architectures and hyperparameters a few lessons have been learnt which may prove to be useful for readers planning to work on future projects using neural networks in general, and recurrent neural networks (used for the encoder and decoder networks of the autoencoder here) in particular.

The first lesson has been that the most important hyperparameter to tune is the learning rate. This is the coefficient applied to the gradients calculated through the use of backpropagation that will update each of the weights in the network. $w = w - \alpha w'$ where w are the current weights and w' are the gradients found through backpropagation. Upon building the first autoencoder that only handled protein sequences, the network failed to learn because the learning rate was not low enough. Not only should different learning rates be tried and a drop in the value generated by the loss function observed but also learning rates ranging from 0.1 all the way to 0.0001 should be attempted. Initially the author failed to try learning rates below 0.01 which is why the network failed to learn and time was spent trying to debug other problems that were not in fact the issue. The second lesson learnt was to start with a very small dataset and overfit it first before adding in any more complexity. A small dataset not only increases the rate at which the network can learn, shortening the feedback loop, but more importantly, a small dataset enables an assessment for whether or not the neural network is learning as it should as is able to obtain a loss

[6]Mikolov et al. Efficient Estimation of Word Representations in Vector Space, arXiv, (2013), https://arxiv.org/abs/1301.3781

[7]Alley et al. "Unified rational protein engineering with sequence-only deep representation learning", bioRxiv, (2019), https://www.biorxiv.org/content/10.1101/589333v1

[8]https://pytorch.org/

of approximately 0.0. The next lesson learnt was to think hard about where the biggest bottleneck in an underperforming network may be. The author spent much time concerned about the architecture of the decoder component of the autoencoder without realizing that the more pressing bottleneck was in the ability for the encoder to accurately encode into the latent space. Initially the encoder which used a Long Short Term Memory (LSTM) recurrent neural network architecture (explained further in the Methods section) was only taking the hidden state output from the last time step of the network and passing this on as the latent space. However, it is well known that recurrent networks struggle to capture information over long distances and therefore it would be very difficult to hold information in the network across up to 750 time steps. Realizing that the encoder architecture was the limiting factor, the solution to this problem came from the the word embedding of protein sequences paper[9] which, instead of taking only the last hidden state, averaged over each of the hidden states across every timestep for the encoder. A further problem has been the need for computation in the form of graphics processing units (GPUs) which can do matrix multiplication operations far more efficiently than less specialized central processing units (CPUs)[10] found in all personal computers. In order to train large networks with up to 100 million different parameters that must be learnt, it can take weeks even when using four or eight GPUs, each of which can be very expensive. Another lesson that also has to do with learning rates has been the need to alter the learning rate over time. Each dataset a neural network is trained on will have a different loss function landscape that gradient descent must traverse and this particular dataset and autoencoder has a highly convex and steep landscape where in order to continue reducing the loss the learning rate must be continuously shrunk during training enough so that it does not jump to a higher cost position but not so much that no further progress is made. The solution to this is the use of a learning rate scheduler but these are highly heuristic and can require much trial and error.

As a result of this problem the autoencoder has been unable to learn how to create any meaningful reproduction of the input protein yet. Even after more extensively trying different learning rates, if the model still fails to learn then experimentation with more powerful neural network architectures such as the state of the art BERT[11] (Bidirectional Encoder Representations from Transformers) encoder for sequence based data will be implemented to hopefully increase the model capacity and performance.

While the autoencoder itself has not yet borne fruit, during this project the author has also become an open source contributor to PyTorch[12], the OpenProtein project[13] and ProteinNet database[14]. These packages have been invaluable

[9] Alley et al. "Unified rational protein engineering with sequence-only deep representation learning", bioRxiv, (2019), https://www.biorxiv.org/content/10.1101/589333v1

[10] Goodfellow et al. "Deep Learning" (2016) Ch. 12

[11] Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", arXiv:1810.04805, (2018)

[12] https://discuss.pytorch.org/t/tensorflow-esque-bucket-by-sequence-length/41284

[13] https://github.com/OpenProtein/openprotein/pull/19

[14] https://github.com/aqlaboratory/proteinnet/issues/7

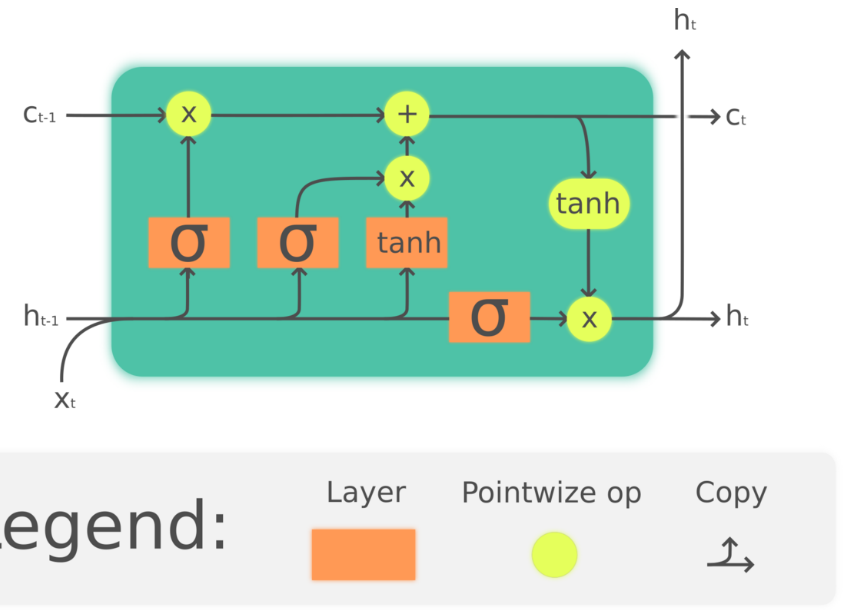

Figure 5: An LSTM cell at one timestep. The σ is the sigmoid activation function. h is the hidden state and c is the cell state which maintains "memory" across timesteps. Source: Wikipedia LSTM Cell

to progress in this project.

5 Methods

The encoder and decoder portions of the autoencoder were both built of Long Short Term Memory[15] (LSTM) recurrent neural network units with some fully connected layers also used in between them to create the latent space and make the final predictions from the decoder for the dihedral angles and amino acid at each position in the protein.

More specifically, the current architecture (which is very subject to change) receives as input the Cartesian coordinates of the backbone atoms and a one-hot encoding of the amino acids of the protein. The sequence data is entered into a bidirectional LSTM with 600 hidden units and the output at each time step is fed into another bidirectional LSTM also with 600 hidden units. The Cartesian coordinates are converted into dihedral angles before being fed into a separate but identically parameterized series of 2 LSTMs. The outputs from the two separate doubly stacked LSTMs for the sequence and tertiary structure are then concatenated together at each time step and fed into another bidirectional

[15]Hochreiter and Schmidhuber, "Long Short-Term Memory", Neural Computation 9(8):1735-1780, (1997)

LSTM with 300 hidden units. The outputs from this at each time step are taken and averaged together to produce a vector of size 600 (300 hidden units from the forward and 300 from the backward directions of the bidirectional LSTM). This is then put through a batch normalization layer and then a fully connected layer with 500 hidden units and an elu activation function. Then another fully connected layer also with 500 hidden units which outputs the latent space.

This is then fed into the decoder which first inputs the same latent space vector as its input as every timestep into a bidirectional LSTM with 600 hidden units. This is passed to another LSTM also with 600 units. The output at each timestep is then passed in two directions. The first direction goes through a fully connected hidden layer with 21 hidden units and a softmax activation function that converts this output into a probability density over the 20 amino acids and a padding character (this is explained later around data preprocessing). The second direction puts the LSTM output through a fully connected layer with 50 hidden units, then an elu activation function and another fully connected layer with 3 hidden units that outputs the dihedral angles for each amino acid's backbone atoms. These are then fed into the pNeRF algorithm[16] to convert them into Cartesian coordinates in a parallelized, GPU optimized, way. Together, these encoder and decoder networks making up the autoencoder have 44,858,224 parameters to train. The loss function being used is cross entropy loss between the predicted and actual sequences + mean squared error between the predicted and actual dihedral angles. Distance root mean squared deviation (dRMSD) was also previous incorporated, however this seemed to make the loss landscape exceedingly non-convex and steep and the authoer decided for now to try and train the network without using it[17].

For the data preprocessing, the dataset ProteinNet was used[18] which uses data from the Protein Data Bank (PDB)[19] with careful procedures for removing sequence homologous proteins. This dataset came with a mask for which amino acids had undetermined coordinates in each protein. In order to avoid having to try to somehow interpolate the position of each of these amino acids, any proteins that had missing data that was not located at either of the ends of the protein were removed. For those that did have missing atom coordinates only at the ends, these edge amino acids were removed from their sequence and tertiary structure in essence trimming the ends of the protein. Any proteins that were longer than 750 amino acids in length were also removed in order to reduce the length of long distance relationships that the recurrent neural networks had to capture. This resulted in a training dataset of approximately 77,000 proteins, a cross validation dataset of approximately 400 and a test dataset

[16]Mohammed AlQuraishi, "Parallelized Natural Extension Reference Frame: Parallelized Conversion from Internal to Cartesian Coordinates", Journal of Computational Chemistry, 2019, https://doi.org/10.1002/jcc.25772

[17]The OpenProtein project (https://github.com/OpenProtein/openprotein) which some code was taken from was able to obtain good results for the protein folding problem without using dRMSD

[18]Mohammed AlQuraishi "ProteinNet: a standardized data set for machine learning of protein structure" https://arxiv.org/pdf/1902.00249.pdf (2019)

[19]https://www.rcsb.org/

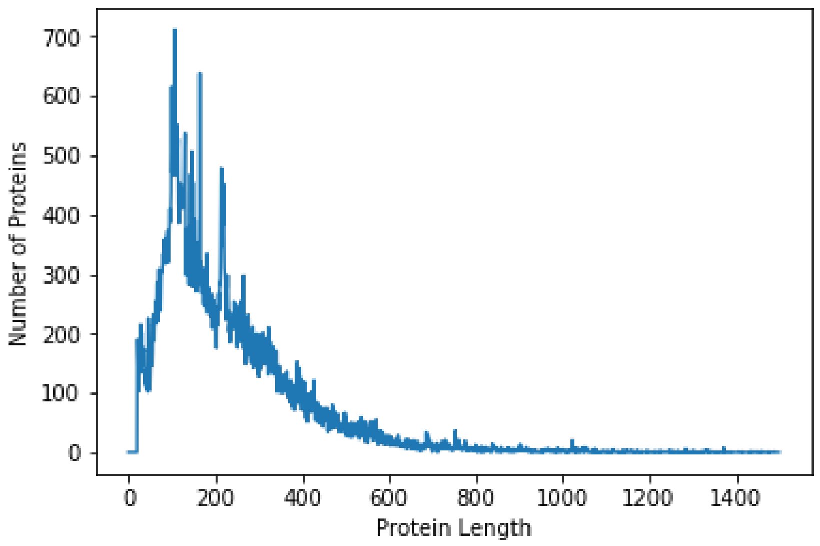

Figure 6: The distribution of protein lengths in the training dataset from ProteinNet.

of 23. In order to make training more efficient, minibatches of size 32 were used. This means that 32 proteins were selected from the dataset and their losses calculated and averaged over before performing gradient descent from this data alone. This leads to less accurate stochastic steps across the loss landscape but much faster ones (there are $\frac{n}{32}$ neural network weight updates (where n is the size of the dataset) rather than one on every iteration through the data. This often results in faster convergence. However, in order to generate a minibatch, each protein needs to have the same sequence length so that it can be simultaneously processed by the LSTM recurrent network. This means that proteins of a shorter length need to have a padding value added to the end of them. If the proteins are randomly selected to be used in a minibatch then much wasteful padding must be added to accommodate the longest sequence chosen[20]. To remove this inefficiency the author developed and refined (in collaboration with another PyTorch user) a new DataLoader that selects proteins of exactly the same length or, when not possible, of very similar lengths to reduce the total amount of padding used[21].

[20]This padding is masked from the loss function meaning that it does not effect the predictions made but it remains computationally wasteful

[21]https://discuss.pytorch.org/t/tensorflow-esque-bucket-by-sequence-length/41284

6 Discussion

The idea to apply an autoencoder to discrete biological sequence data in order to make it continuous and easier to manipulate is not novel. The author originally came across the idea from studying the Flu Forecaster project developed by Eric J. Ma where he used a variant of the traditional autoencoder called a variational autoencoder to convert flu sequences into continuous space so that he could then use a Gaussian Process to do time series predictions for how the flu was most likely to next mutate[22]. The results for the performance of his variational autoencoder were promising, however, doing the same for 3D structural protein data, including the rotamers, their angles, and the backbone coordinates, has been a much greater challenge. Moreover, the author decided to avoid using a variational autoencoder which tries to force the latent space to conform to a particular distribution (most often Gaussian) using Kullback-Leibler Divergence between the latent space and the chosen distribution as part of the loss function. This makes it easier to sample from the latent space and use the autoencoder in a generative way to create similar but novel samples not seen in the training data. However, Boris Sattarov et al. (2019)[23] argue that for the purposes of generating new small molecules (and we should also assume proteins), it is wrong to think of their latent space as being unimodally distributed and, therefore, the latent space should not be penalized to conform to a particular unimodal distribution. Instead, a more simple autoencoder without any penalty on the distribution of the latent space should be used. However, to prevent this autoencoder from simply learning the identity function, mapping its input to its output, the latent space is smaller than the size of the input forcing the network to learn an efficient encoding for the input information. This makes it harder to sample from the latent space but the aforementioned authors used Generative Topographic Mapping in order to accomplish this[24].

Fortunately, beyond just small molecules there has already been some exciting work in applying deep neural networks to protein folding and design. This existing work supports the feasibility of the future development of this project.

As brief summary of what has been recently accomplished in the literature: Deepmind's AlphaFold entry which won CASP13[25]. A major reason for their success was their application of a neural network and using it to output a continuous map of pairwise distances that could then be optimized via gradient descent. One of the 15 most downloaded arXiv papers of 2018[26] developed a new protein folding prediction algorithm that used a deep neural network without any evolutionary data, templates or multiple sequence alignments[27] . A

[22]https://fluforecaster.herokuapp.com/

[23]Boris Sattarov et al., De Novo Molecular Design by Combining Deep Autoencoder Recurrent Neural Networks with Generative Topographic Mapping, JCIM, 2019

[24]Boris Sattarov et al., De Novo Molecular Design by Combining Deep Autoencoder Recurrent Neural Networks with Generative Topographic Mapping, JCIM, 2019

[25] https://deepmind.com/blog/alphafold/

[26] https://rxivist.org/top/2018

[27]Mohammed AlQuraishi "End-to-end differentiable learning of protein structure" https://www.biorxiv.org/content/biorxiv/early/2018/08/29/265231.full.pdf (2018)

major reason behind the success of the network learning important protein features, such as secondary structure, was the invention of a "recurrent geometric network" that is able to transform dihedral angles to Cartesian coordinates, allowing for an end-to-end differentiable system. A good portion of the code for this paper has been used indirectly by me through the OpenProtein project[28].

Last year in Nature[29] a paper used a variational autoencoder for two different purposes: Firstly, To mutate proteins that previously could not bind to metals so that they now could. Secondly, to design proteins that fit a novel fold. However, this study is different from this project in many ways. To mention just two important differences: firstly, they used a variational autoencoder (VAE) that had an easier learning task as it did not encode the 3D Cartesian coordinates of a protein but instead only the protein topology into "Taylor's 'periodic table' of protein structures" which is a grammar for describing secondary structure but unsuitable for designing novel protein structures. Secondly, they did not perform the second step of Figure 3, using gradient descent on the latent space. Instead they used the generative properties of the VAE to sample proposed structures, evaluated them (using many different optimization and then scoring functions), resampled from the best, and repeated until they ended up with a high scoring structure (as is shown in Figure 4). Their VAE was also trained on a very small dataset and they used a basic VAE network architecture that could have benefitted from many further optimizations.

The Results section already mentioned some of the previous and still present problems with this project. However, while the author believes that with enough computational effort and architectural exploration this autoencoder can be a success, there are some longer term problems that may linger and be harder to solve: Firstly, there may not be enough data present for the autoencoder to generalize well. Secondly, there is a large bias in the PDB towards proteins discovered from X-Ray Crystallography[30] which also may limit the generalization abilities of the autoencoder. Thirdly, another possible barrier to generalization may be that currently only perfectly solved structures except for the ends of the protein are being used in the model, this means that proteins which are intrinsically disordered in any way are not included in the preprocessed dataset.

The next steps for this project will be to: perform more hyperparameter optimization particularly of the learning rate scheduler; implement BERT, the previously mentioned more powerful transformer based architecture; impute unknown amino acid backbone positions to increase the size of the dataset from approximately 77,000 to 120,000; work with longer proteins up to 2,000 amino acids in length; and work with rotamers rather than just the backbone atoms. This is more computationally expensive for conversion between dihedral

[28]https://github.com/OpenProtein/openprotein

[29]Greener et al. "Design of metalloproteins and novel protein folds using variational autoencoders" Sci Rep. 2018; 8: 16189

[30]These are charts showing the number of X-ray crystallography versus NMR solved PDB structures, respectively: https://www.rcsb.org/stats/growth/xray, https://www.rcsb.org/stats/growth/nmr showing an order of magnitude fewer NMR structures

angles and cartesian coordinates and the ProteinNet dataset currently lacks this rotamer data meaning the author would need to either switch datasets or use the same one but get all rotamer positions from the PDB for each of the proteins in the dataset.

Ultimately, the author has learned an immense amount from this project, remains excited about its theoretical potential, and believes that with enough continued work it can be successful and only constrained by the limits of what types of protein structures we currently have data for.

7 Bibliography

Papers, in order of reference by footnotes:

Stuart Russell and Peter Norvig "Artificial Intelligence A Modern Approach", 3rd edition, Prentice Hall, Chapter 4.2, (2009)

Goodfellow et al. "Deep Learning" (2016) Chapters 8,12,14

Mikolov et al. Efficient Estimation of Word Representations in Vector Space, arXiv, (2013), https://arxiv.org/abs/1301.3781

Alley et al. "Unified rational protein engineering with sequence-only deep representation learning", bioRxiv, (2019), https://www.biorxiv.org/content/10.1101/589333v1

Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", arXiv:1810.04805, (2018)

Hochreiter and Schmidhuber, "Long Short-Term Memory", Neural Computation 9(8):1735-1780, (1997)

Mohammed AlQuraishi, "Parallelized Natural Extension Reference Frame: Parallelized Conversion from Internal to Cartesian Coordinates", Journal of Computational Chemistry, (2019), https://doi.org/10.1002/jcc.25772

Mohammed AlQuraishi "ProteinNet: a standardized data set for machine learning of protein structure" https://arxiv.org/pdf/1902.00249.pdf (2019)

Boris Sattarov et al., De Novo Molecular Design by Combining Deep Autoencoder

Recurrent Neural Networks with Generative Topographic Mapping, JCIM, (2019)

Mohammed AlQuraishi "End-to-end differentiable learning of protein structure" https://www.biorxiv.org/content/biorxiv/early/2018/08/29/265231.full.pdf

(2018)

Greener et al. “Design of metalloproteins and novel protein folds using variational autoencoders” Sci Rep. 2018; 8: 16189

WWW links:

https://pytorch.org/

https://discuss.pytorch.org/t/tensorflow-esque-bucket-by-sequence-length/41284

https://github.com/OpenProtein/openprotein/pull/19

https://github.com/aqlaboratory/proteinnet/issues/7

https://www.rcsb.org/

https://fluforecaster.herokuapp.com/

https://deepmind.com/blog/alphafold/

https://rxivist.org/top/2018

https://github.com/OpenProtein/openprotein