Tertiary Structure Prediction via Structure Matching and In-painting

Dimitri Leggas
Columbia University
ddl2133@columbia.edu

Kendal Sandridge
Columbia University
ks3311@columbia.edu

Deniz Ulcay
Columbia University
du2147@columbia.edu

Abstract

In this project, we address the problem of predicting the tertiary structure of a protein given its sequence of amino acids and the corresponding secondary structure classes (Q8 structures). While this question has been considered for decades, only recently have neural networks been applied to the problem. The formulation employed herein is to construct a map from the two input sequences, amino acids and secondary structures, to a distance matrix containing the distances between every pair of residues in the folded protein. We construct incomplete distance matrices by matching the amino acid sequences with the sequences of proteins with a known structure. These matrices are completed by finding the closest distance matrix produced by a generative adversarial network.

Introduction

Proteins are polymers of amino acids that form the basis of life as we know it. Almost all biological processes are mediated by proteins, which act as antibodies, messengers, enzymes, structural elements, and transport and storage. The complexity of the underlying sample space, given the possibility of 20 residues at every position, is lessened by the repetitive pattern of torsion angles and hydrogen bonds. These relatively few structural features makeup the so-called secondary protein structure. In this project, we take the secondary structure for granted and address the problem of predicting tertiary structure, i.e. predicting how protein is folded in three-dimensional space.

Rather than threading, which is the recognition of folds in the protein structure, as in many previous works [2], we define the problem as learning a map from the input space of two length n sequences (the amino acid and Q8 sequences, respectively) to a $n \times n$ matrix containing the pairwise distances between all residues in the folded structure. From this distance matrix, it is a straightforward optimization problem to embed the distances into $\mathbb{R}^3$ [5]. This solution allows for the actual visualization of the protein. While machine learning approaches have been used to attack this problem, very few have used deep learning as we do here. A notable exception to this trend is the recent announcement of AlphaFold by Google DeepMind [1].

The approach in this project is inspired by recent work by Anand and Huang on recovering complete tertiary structure from a partially-known structure that is either corrupted or has missing parts [4]. Although we do not have such partial distance matrices *a priori*, we construct incomplete distance matrices by querying a custom BLAST [3] database of protein sequences with known folded structures. After training a generative adversarial network (GAN), we have a generator that allows us to predict the missing distances so that we have a realistic distance matrix. These matrices can then be scaled based on an estimation of the structure's mean and variance.

A good solution to the problem of tertiary structure prediction would be widely applicable. Successfully prediction would not only allow protein identification, but would lead to an improved methodology for the design and synthesis of custom proteins. This has myriad applications in medicine, in particular in drug development [6].

Previous Work

As mentioned, most progress on this problem has been made by threading, or the analysis of folds in the protein. This approach frequently involves experimentally obtained data coupled with computer-based predictions. Rather than attempting this task *de novo*, many threading algorithms use a template-based approach. These algorithms consist of four main steps: (i) find a similar protein to the target that has a known tertiary structure; (ii) align the target's sequence with the known template; (iii) satisfy the spatial restraints of the template; (iv) construct unaligned regions.

Each of the steps in template-based methods are difficult. Machine learning approaches, in particular use of Hidden Markov Models, have been successful in addressing template identification. Methods from molecular dynamics are necessary for the refinement of templates [18].

To date, these template-based methods are the most successful in predicting tertiary structure. The Critical Assessment of protein Structure Prediction, or CASP, is a collec-

tive experiment that has taken place every two years since 1994. In 2016, at the time of CASP12, the final report indicated that template-based methods out performed all others on the tasks of structure prediction [2]. The work put into CASP12, of course, preceded the wide availability of deep neural network libraries. CASP13 includes submissions that utilize deep learning, like the aforementioned AlphaFold. The approach taken by DeepMind employs a neural network to predict distances between residues and fold angles [1]. In this work we will focus entirely on computing the inter-residue distances in the folded structure from the primary and secondary structure sequences.

This work would not be possible without BLAST and the recent work of Anand and Huang on distance matrix in-painting. Anand and Huang's work is in the context of protein design and not structure prediction [4], but we are nonetheless able to apply their methodology to the problem at hand after constructing partial matrices according to alignments found using BLAST.

Sequence alignments have been used to infer information about protein tertiary structure in numerous other previous works [15] [17]. This is due to the identical tertiary structures of matching helix sequences and correlated mutation information that can be inferred from sequence alignments. Constrained by the size of our training data, we were not able to use BLAST's alignment findings to predict protein residue-residue contacts as discussed in [15] [17] yet these alignments still revealed valuable information.

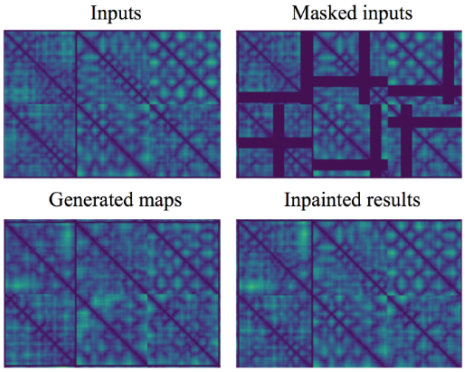

Figure 1. These images highlight the ability of Anand and Huang's method to produce realistic distance matrices from incomplete data. For more details see our Methods or the original paper [4].

The inferred partial structures are then used analogously to the masked inputs used by Anand and Huang. The particulars of both of these methods are addressed in a more detail in our Methods section.

Problem Formulation

We inherit a clear problem formulation which allows us to circumvent the issues of folding and templates and focus our energy on predicting inter-residue distances. Given a length n sequence of amino acids, and the corresponding length n sequence of n secondary structure classes, we predict the symmetric $n \times n$ distance matrix that where the ij^{th} entry gives the distance between residue i and residue j in the folded protein. We give one example.

Inputs:

- Amino acid sequence: LECHNQQSSQTPTTTGC SGGETNCYKKRWRDHRGYRTERGCGCPSVK NGIEINCCTTDRCNN
- Secondary structure: LEEELLLTTLLLLEEELLTT LLLEEEEEEEETTEEEEEEEESLLLLLTTLEEEE ESSTTLLL

Output (distance matrix):

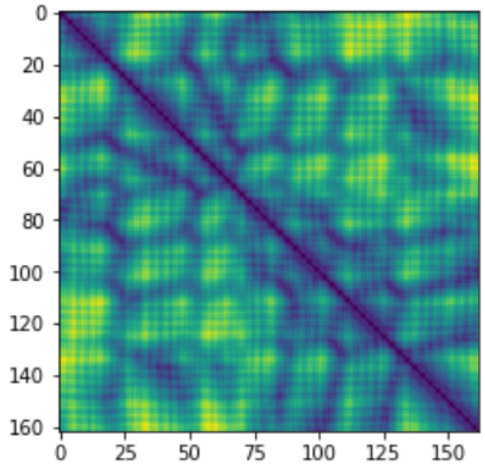

Figure 2. The dark blue pixels represent values close to 0, and lighter green pixels represent larger values.

Methods

Broadly speaking, we construct incomplete distance matrices by matching the amino acid sequences with the sequences of proteins with a known structure. These matrices are completed by finding the closest distance matrix produced by a generative adversarial network. More specifically, we begin by summarizing our pipeline:

(i) Create a database of amino acid and secondary structure sequences from the training data. Using BLAST, we find matches between the test and training sequences. Whenever the matches are good, we copy the

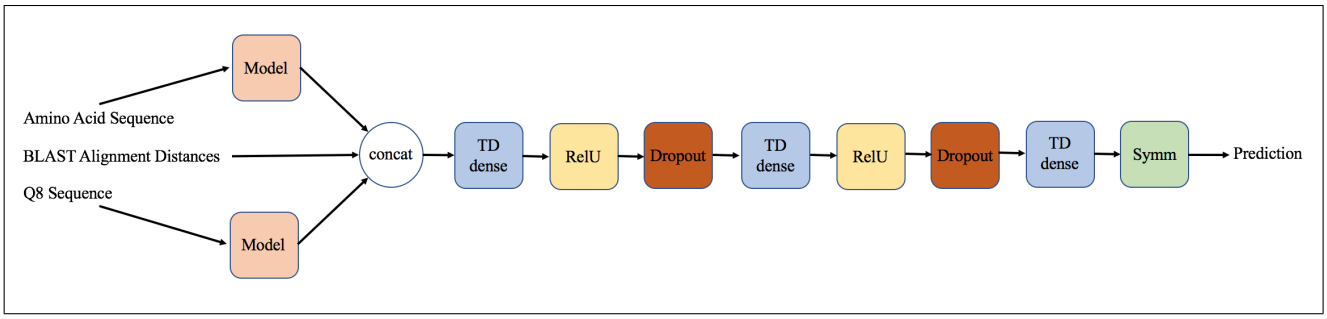

Figure 3. Overall architecture of the end-to-end model. Amino acid and Q8 sequences are inputted to the implemented model in parallel. The outputs of these parallel layers are concatenated with the distance information obtained using BLAST and passed onto the fully-connected layers before the output is symmetrized by our custom Symmetrize (Symm) layer.

corresponding portion of the training sequence's distance matrix to form an initial hypothesis for the test sequence's distance matrix.

(ii) Train an end-to-end network that extracts features from the two input sequences with convolutional and recurrent layers and takes into account the stitched hypothesis matrix from (i) in order to predict the entire distance matrix. We then paste the stitched regions over the top of this filled out prediction.

(iii) Train a generative adversarial network (GAN) to generate realistic distance matrices of size 160×160, 320×320, and 691×691 (691 is the maximum length of a sequence). Using gradient descent, we can find the input that produces the distance matrix most similar to the hypothesis generated by (ii).

(iv) Using the Auto Scikit Learn library, we build two strong models to predict the mean and variance of distances of a sequence, respectively. Using the mean and variance predicted by these models, we re-scale the output from (iii) .

Matching Structure Using BLAST

We used BLASTs protein sequence alignment program, BLASTP, for both training and post-processing in order to locate sequence alignments with our training sequences. We utilized the BLAST library offered by NCBI to create our own training and test databases and queried sequences for BLASTP to search over our databases. We used no external database for our sequence alignments. We then used these matches to infer corresponding tertiary structure information from the training target data.

For training, getting the alignments with all sequences in the training set for a sequence would have resulted in a perfect alignment with the sequence itself. Since this would have led to an extreme case of overfitting, we created a unique database for each training sequence that consisted of all training samples except the sequence itself. We then determined the Levenshtein distance between the alignment pairs and sorted the alignments according to their Levenshtein distance to the aligned segment of the original sequence. After that, we stitched the corresponding distance information from these alignments onto an $L \times L$ matrix (L being sequence length) based on their Levenshtein distance to ensure that more successful alignments would be given priority. We did this by iteratively replacing overlapping areas with tertiary structure information from more successful alignments.

To evaluate our information gain from these stitched matrices, we concatenated them to the outputs of the parallel model blocks of our baseline sequence architecture. The added pairwise features of some residues allowed the network to infer information about other residues too and lowered our validation loss by 14%. As observed in Figure 4, the distance predictions made by the end-to-end model with BLAST information were noticeably more in sync with the stitches.

For post-processing, to infer information about the tertiary structure of our test sequences, we queried them for BLASTP to search over our test database that consists of all training sequences. After our model made its predictions on the test data, we stitched the residue-residue distance information from sequence alignments we obtained with BLASTP onto our predictions. Same as in the stitching process for training, we stitched the distance information based on the Levenshtein distance of alignment pairs.

Estimating Distances from Sequences and Matches

We initially used an end-to-end approach, which proved to be quite weak for estimating inter-residue distances *de novo* (see the milestone report). However, we found that we could slightly improve these results by also taking advantage of the matched structures found using BLAST. Furthermore, we reasoned that we could take advantage of this additional structural information during the in-painting stage.

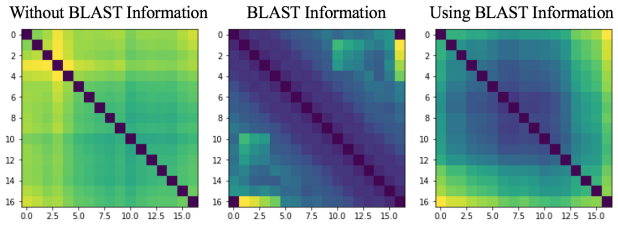

Figure 4. These distance matrices convey the information gain from using BLAST. Given the distance information of alignments in the training data, the end-to-end model can make more accurate predictions.

Our baseline approach is to apply vanilla sequence models to the problem. The overall architecture makes use of neural networks that are known to perform well in the setting of secondary structure prediction in order to learn representations of the amino acid and secondary structure. These features are then concatenated and passed through fully-connected layers in order to make the final predictions. See Figure 3 for a diagram of this architecture.

Our sub-model, which we used for feature extractions, came from our class's recent work on secondary structure prediction [8]. One model we used was the Bidirectional GRU with Convolutional Blocks. This model consists of convolutional layers in parallel followed by three bidirectional GRU layer and finally several fully-connected layers. Input of the model is either an embedded representation of the primary or secondary protein structure sequences. In order to detect local correlations in the data, this model uses three parallel convolution layers. Kernel sizes of these convolution layers are: 3, 5, 7. They all use same padding and use 64 channels each. All three are followed by RelU activation and batch normalization. This model then uses three stacked bidirectional GRU layers with an L2 recurrent regularizer to detect global correlations in the data. Each GRU unit has 64 units which means the bidirectional layer has 128 units with the inverse direction considered. Dropout at rate 0.1 is used after the layer. For the optimizer, this model uses Nadam optimizer, which is Adam optimizer with Nesterov momentum. For the loss function, we used mean squared error.

Applying In-paining to Structure Completion

In order to complete the predictions made by the pattern matching with BLAST, we required a generator. Using Deep Convolutional GAN (DCGAN) by Radford et al. [13] we produced realistic distance matrices of sizes 160×160, 320×320, and 691×691.

We use most of the hyperparameter settings employed by the original paper. We use a Leaky ReLU in both the generator and the discriminator, with the α parameter set to 0.2. We did not use batch normalization, because we realized that because of the way our model was written, the real and generated images were being normalized together. We apply a hyperbolic activation at the end of the generator. Both the generator and discriminator were trained using the Adam optimizer, and following Anand we set the learning rate to 0.0002, $\beta_1 = 0.5$, and $\beta_2 = 0.9$ [4]. Unable to obtain stable results with the vanilla GAN loss, we train with the improved Wasserstein loss [11]. Finally, we initialize our noise according to a Gaussian distribution as suggested by Soumith Chintala [7].

Now, given a matrix from the preceding layer of the pipeline, we can make a better prediction using the in-painting method. Our approach follows Yeh et al. [16] as well as Anand and Huang [4]. The procedure is to find the distance matrix produced by the generator that minimizes a certain loss, defined shortly. This loss consists of context term and a prior term. The context term takes into account where the holes are by giving larger weight to known distances that are near holes. The prior term is a measure of how realistic the discriminator thinks the generated distance matrix is.

Before proceeding with the definitions of the losses, a clarification needs to be made. Anand and Huang deal with images that have explicit holes. Indeed, we produce such images by stitching together distances corresponding to BLAST matches. However, as discussed above, we paste these stitches over a prediction of the entire structure. We treat any region without a BLAST match to be part of a hole, as the end-to-end predictions are relatively weak. However, the loss, as we will see shortly, takes into account the entire "corrupt" distance matrix, therefore it is useful to have these predictions be part of that hypothesis matrix.

We now define the losses used. Our terminology follows Yeh et al. [16]. Let $\mathbf{y}$ be the corrupted image and $\mathbf{M}$ of the same size be a binary mask denoting the holes; by holes we mean as defined above. We seek $\hat{\mathbf{z}}$ according to

$$\hat{\mathbf{z}} = \arg\min_{\mathbf{z}} \{\mathcal{L}_c(\mathbf{z}|\mathbf{y}, \mathbf{M}) + \mathcal{L}_p(\mathbf{z})\}$$

where $\mathcal{L}_c$ and $\mathcal{L}_p$ respectively denote the context and prior losses just mentioned. In order to define the context loss, we need a weight matrix given by:

$$W(i) = \begin{cases} \sum_{j \in N(i)} \frac{1-M_j}{|N(i)|} & M(i) \neq 0 \\ 0 & M(i) = 0 \end{cases}$$

As discussed briefly above, this gives weight to known distances surrounded by many blank distances. We choose a neighborhood width and height of 7. We are now ready to define the context loss:

$$\mathcal{L}_c(\mathbf{z}|\mathbf{y}, \mathbf{M}) = ||\mathbf{W} \odot (G(\mathbf{z}) - \mathbf{y})||_1$$

While the context loss tries to force the generated distance matrix to be close to incomplete one, the prior loss

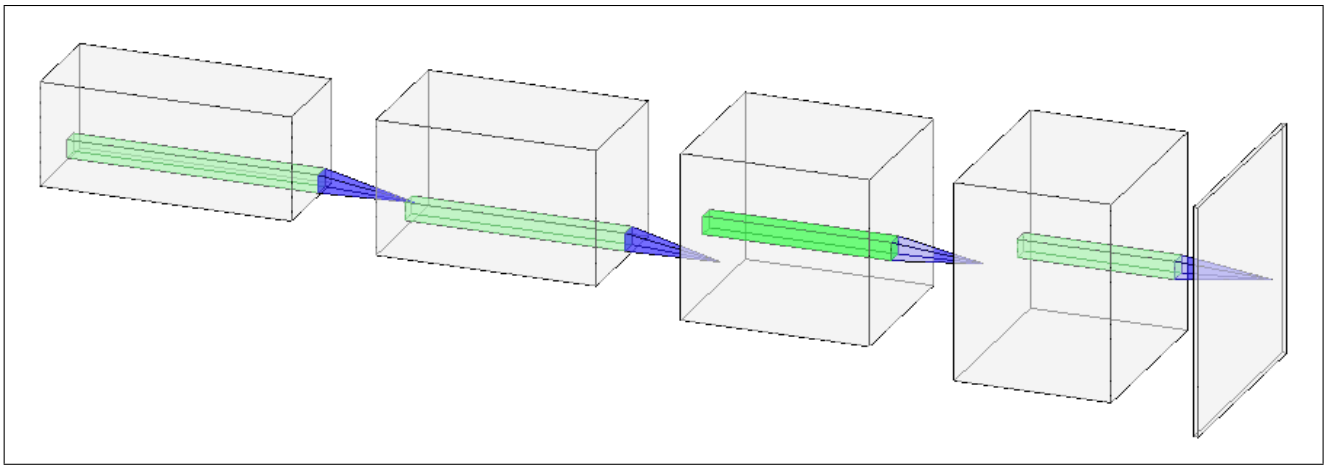

Figure 5. The architecture of our generator. We follow the Deep Convolutional GAN developed by Radford et al. [13]. Each layer is a Transposed 2D convolution. The filters decrease according to 256, 128, 64, 32, 1; we use a filter size of 2×2 at each layer; and we apply a Leaky ReLU activation with $\alpha = 0.2$ after every layer, except the final layer where we use the hyperbolic tangent activation. This figure was generated using the NN-SVG by Alexander Lenail [12].

penalizes unrealistic images. It is defined by

$$\mathcal{L}_p(\mathbf{z}) = \lambda \log(1 - D(G(\mathbf{z})))$$

Following both Yeh et al. and Anand and Huang, we set $\lambda = 0.003$. We find $\hat{\mathbf{z}}$ via gradient descent.

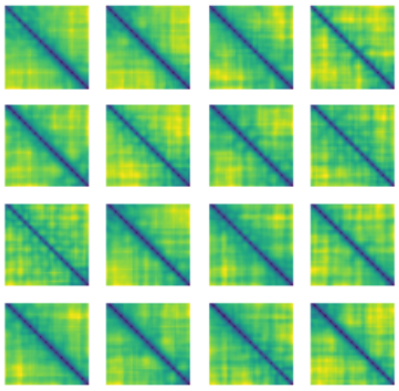

Figure 6. These (unscaled) distance matrices were produced by our trained generator after 800 epochs. The generated results are ultimately scaled to have the predicted mean and standard deviation of distances.

Appropriately Scaling the Distances

We used the Auto Scikit Learn library [9] to develop two robust regressors for the mean and standard deviation of the distance matrix for a given input sequence. We re-scale all of our predictions so that they have the predicted mean and standard deviation. This is done via the following equation:

$$x' = \mu' + (x - \mu) \cdot \frac{\sigma'}{\sigma}$$

where x, x' represent the original values and new predicted distances, μ, μ' represent the original mean and the predicted (desired) mean, and σ, σ' represent original standard deviation and the predicted (desired) standard deviation.

Results

We will now exhibit examples of our prediction matrices at different stages of the pipeline. These matrices are all submitted on the shared Google Drive for tertiary structure prediction and on our `GitHub`.

First, we show an example of the BLAST results and how they are stitched over the improved end-to-end predictions.

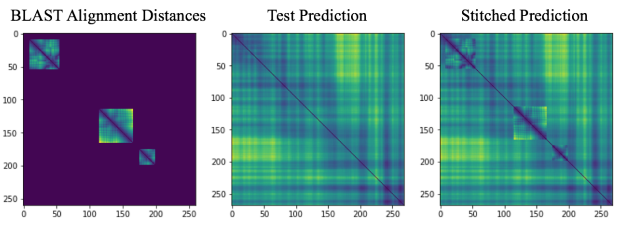

Figure 7. Distance information is inferred from training sequences BLAST finds alignments with. After a prediction, alignment distances are stitched onto the predicted matrix.

After supplementing our end-to-end model with residue-residue distances obtained from BLAST alignments, there

was a noticeable improvement in the model's understanding of the target space. As shown in Figure 8, we decided to stitch the residue-residue distances of alignments in the training data onto the end-to-end model's test predictions. While the end-to-end predictions do not have the same clearly defined structure as desired, it is visible in Figure 7 that there is alignment in the underlying structure. Green portions align with green portions, and blue with blue.

The coverage that the stitches provide in the distance matrix hypotheses vary greatly. A few test sequences had no matches in our database of training examples. Other, smaller test sequences are covered almost completely by the matches.

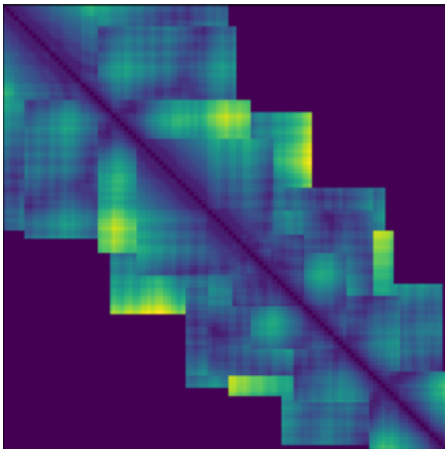

Figure 8. Some of the distance matrix hypotheses were better than others...

Problems and Barriers to Success

While we were successful in using BLAST to construct strong hypotheses for the distance matrices and in improving our end-to-end results, we are still dealing with some problems with our GAN training. By the time that we got stable results, we realized that regardless of the noise input, our generator yielded identical distance matrices.

We tried several methods for overcoming our mode collapse problem, many of which were suggested in the GAN training tutorial by Soumith Chintala in [7]. These include:

- Use noisy labels (e.g. positive label values chosen from $[0.8, 1.2]$ instead of being all 1)
- Generate noise according to a Gaussian distribution instead of the uniform distribution
- Change or remove batch normalization
- Replace ReLU in the generator with Leaky ReLU as in the discriminator

While these changes helped to introduce some variation in the generated distance matrices, after looking more closely at the generated matrices they still appear to have the same global structure. We also looked into implementing minibatch discrimination as introduced by Salimans et al [14]. Minibatch discrimination passes images to the discriminator in batches as opposed to individually to give the discriminator extra context. Unfortunately, we were unable to obtain good results in any of our attempts. It is noted by Ian Goodfellow that this method is prone to error if the implementation deviates slightly from the authors' code [10], and we were unable to make their code compatible with our code.

The issue with the GAN propagated throughout the rest of our pipeline. We were unable to get gradient descent for the in-painting to converge. This most likely owes to the fact that our generator does not cover the entire space of distance matrices as it should. It should also be noted that we implemented this gradient descent ourselves, as the Keras gradient function `K.gradients` incorrectly returned `[None]` despite the fact that our defined loss function is differentiable. There is some documentation that suggests that this is a Keras/tensorflow issue. (See: `https://github.com/keras-team/keras/issues/8478`).

Predictions on Test Data

We present now the quality of predictions made at various stages of the pipeline. (We will fill out this table after our meeting with Prof. Drori).

Predictions	
Pipeline Stage	MSE on test sequences
Stitches	–
Stitches on E2E	–
Stitches scaled	–
Stitches on E2E scaled	–

Figure 9. Our prediction error on the test sequences.

Final Comments

While our overall pipeline did not execute exactly as planned, we feel that we are very close to getting the strong predictions that we believe this methodology can produce. Given that we changed directions at our postponed milestone only a week ago, we are pleased to have made it most of the way to the goal since then. At least we have improved our predictions from the milestone.

References

[1] Alphafold: Using ai for scientific discovery. https://deepmind.com/blog/alphafold/.

[2] Twelth meeting on the critical assesment of techniques for protein structure prediction. *Proteins*, 86, 2016.

[3] S. Altschul, W. Gish, W. Miller, E. Myers, and D. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215:403–410, 1990.

[4] N. Anand and P. Huang. Generative modeling for protein structures. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 7504–7515. Curran Associates, Inc., 2018.

[5] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3, 2011.

[6] A. Breda, N. Fonseca Valadares, O. Norberto de Souza, and R. C. Garratt. Protein structure, modelling and applications. *Bioinformatics in Tropical Disease Research*, 2007.

[7] S. Chintala. How to train a gan? tips and tricks to make gans work. https://github.com/soumith/ganhacks.

[8] I. Drori, I. Dwivedi, P. Shrestha, J. Wan, Y. Wang, Y. He, A. Mazza, H. Krogh-Freeman, D. Leggas, K. Sandridge, L. Nan, K. Thakoor, C. Joshi, S. Goenka, C. Keasar, and I. Pe'er. High quality prediction of protein q8 secondary structure by diverse neural network architectures. *NIPS2018*, 2018.

[9] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter. Efficient and robust automated machine learning. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2962–2970. Curran Associates, Inc., 2015.

[10] I. J. Goodfellow. NIPS 2016 tutorial: Generative adversarial networks. *CoRR*, abs/1701.00160, 2017.

[11] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville. Improved training of wasserstein gans. *CoRR*, abs/1704.00028, 2017.

[12] A. Lenail. Nn-svg. http://alexlenail.me/NN-SVG/AlexNet.html.

[13] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *CoRR*, abs/1511.06434, 2015.

[14] T. Salimans, I. J. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. Improved techniques for training gans. *CoRR*, abs/1606.03498, 2016.

[15] S. Seemayer, M. Gruber, and J. Soding. Ccmpred–fast and precise prediction of protein residue-residue contacts from correlated mutations. *Bioinformatics*, 30:3128–3130, 2014.

[16] R. A. Yeh, C. Chen, T. Lim, M. Hasegawa-Johnson, and M. N. Do. Semantic image inpainting with perceptual and contextual losses. *CoRR*, abs/1607.07539, 2016.

[17] H. Zhang, Q. Zhang, F. Ju, J. Zhu, S. Sun, Y. Gao, Z. Xie, M. Deng, W.-M. Zheng, and D. Bu. Predicting protein inter-residue contacts using composite likelihood maximization and deep learning. *CoRR*, abs/1809.00083, 2018.

[18] Y. Zhang. Progress and challenges in protein structure prediction. *Current Opinion in Structural Biology*, 2008.