Deep learning for generating peptide helices

Xuezhi Xie
Computer Science
University of Toronto
xuezhi.xie@mail.utoronto.ca

Zi Han Zhao
Electrical & Computer Engineering
University of Toronto
simon.zhao@mail.utornto.ca

Abstract

Peptide drugs have shown advantages over traditional drugs since they have high binding affinity, low toxicity and precise specificity, especially for dextrorotatory (D) peptides. One major challenge in D-peptide drugs design is that the limitation of helix structures causes difficulty to fulfill the protein folding constraints. We here propose a Generative Adversarial Networks that utilized Convolution Neural Networks (CNN) as the feature extractors. The model is trained with real protein structures and output the helix structures to enrich the searching databases. Our model has shown its ability to generate correct helix structure and its potentials to be further utilized in D-peptide design.

1 Introduction

Peptide drugs are a fast developing class of therapeutics and they have shown better potentials over traditional small molecular drugs since they are natural candidates for targeting receptors with peptide ligands, they are more specific and relatively easy to synthesize. One major obstacle of peptide drugs is that they are easier to be destroyed by proteases, which leads to low bioavailability[7]. One of the most effective method to solve this is to design the dextrorotatory(D)-peptide drugs, which is more stable compared to the common-type peptides. However, current approaches to design D-peptide drugs all have their major drawbacks, especially for the method proposed by Micheal and our lab on 2018[7], they suffered a lot due to not enough helix structures meeting the folding constraints. In order to solve this, we propose a method to generate more helices structures which could possibly enrich the searching database for d-peptide drug design.

The main novelty in our project is that we proposed a novel generative model architectures which used CNN as the feature extractors. Unlike other current models utilizing discriminative models to do prediction, our model can be beneficial especially when input data is not available.

2 Related works

Computational prediction of protein secondary structures are significant for protein design. There are some related works predicting protein secondary structures.

Real - SPINE is the first method to predict the phi angles using position specific scoring matrix(PSSM), which is a popular evolutionary encoding matrix in bioinformatics. In addition, they applied shifting transformation on angles to decrease the impact of the periodic propriety of the angles so that probabilities of the shifted angles are close to zero around boundary angles[6].

Spider is another popular software in bioinformatics to predict protein secondary structure. They proposed another periodicity handling method which uses sin cos value to encode an angle. The sin and cos value could be directly transformed into angles by the following equation 1. This way to encode angle could make it varying more smoothly. We also use this method to encode all angles for

34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

our dataset. Spider 3 is an updated version using long short-term memory(LSTM) to predict protein secondary structures. The model architecture involves two LSTM layers and two fully-connected layers.

$$\theta = \tan^{-1}[\frac{\sin\theta}{\cos\theta}] \tag{1}$$

3 Data

The source data we are using is coming from Protein Data Bank (PDB). PDB is the key resource for structural biologist, and it stores most big molecular structures information including protein. As shown in Figure 1, the leftest picture is the screenshot of PDB file. It contains all the 3d coordinates for each atom and their residues classes. Using this, we can visualize the PDB as the protein structure shown in the middle. But due to our method focuses on helix structures instead of full protein structures, therefore, a shell script is used to capture the helix structures from PDB databases first to construct our helix databases. The rightest picture is a representation showing how helix structures can be represented by angles.

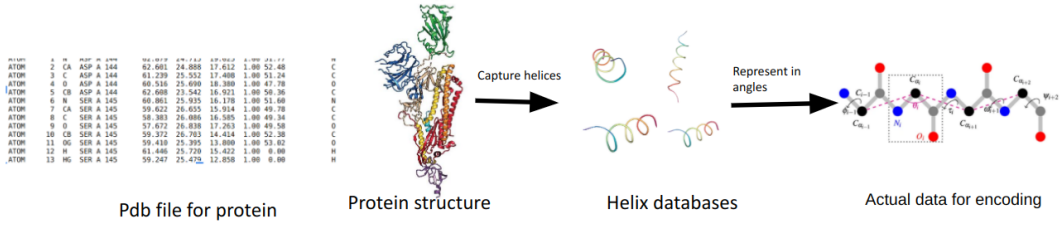


Figure 1: Data Encoding Process

4 Model

Our proposed solution for generating helix structures is to build a WGAN-GP (Wasserstein Generative Adversarial Networks with Gradient Penalty) model, an improved version of our baseline DCGAN (Deep Convolutional GAN) model. It consists of two parts: a discriminator D to classify given data samples as `"real"` or `"fake"`, and a generator G to generate data samples given a noise random variable input `z`. G is trained using real data sample `x` to generate samples as real as possible, trying to trick D; while D's goal is to identify fake data generated by G. During training, they compete against each other to optimize on both functionalities.

As shown in Figure 2, G and D of WGAN-GP are two convolutional neural networks. The FC (Fully connected) and convolutional 1d layers are used to resize the inputs and outputs to make them fit into the main layer; and the main layers, consisting of CNN blocks, are used to extract and analyze abstract features of input data [4].

In comparison, our baseline model DCGAN uses an additional sigmoid layer as the output layer in D since it outputs a probability of `"real"` instead of WGAN-GP's scalar score. Although both models check the similarity between generated and real data distributions, there are major differences between their loss functions and weight updating methods. DCGAN uses JS-Divergence as loss function – the cross-entropy between the two data distributions [8]; whereas WGAN-GP uses Wasserstein distance, a linear method for checking the distance between data distributions [8]. The refactored loss functions for G and D are shown in equation 2. Here, compared with other models, using the Wasserstein distance with gradient penalty can effectively help with convergence and avoid mode collapse, a form of GAN failure that occurs when G figures out a particular way to generate data to trick D [9]. When that happens, G will fall into a limited space with low variety and won't be able to learn complex real data, leading to the output data always being the same type.

$$\begin{cases} \text{D loss} = & D(G(z)) - D(x) + \lambda * \textit{Gradient Penalty} \\ & \text{where } \textit{Gradient Penalty} = (\|\nabla_{G(z)} D(G(z))\|_2 - 1)^2 \\ \text{G loss} = & -D(G(z)) \end{cases} \tag{2}$$

For optimization of our model, we tried tuning different parameters and different test results are compared. For Gradient Penalty weight λ, we found that a λ too small can cause mode collapse and

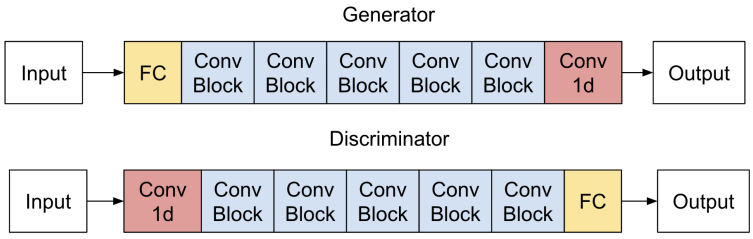


Figure 2: WGAN-GP structures

non-convergence and a λ too large leads to overfitting; hence λ is set to 10. For number of hidden units and number of CNN block layers, we found that when hidden units are over 256 and CNN layers are over 6, loss of G becomes overfitting since feature dimension is too high. However as the more CNN layers and hidden units, the better their abstract feature analysing performance, we set the number of hidden units to 128 and number of CNN layer to 5.

5 Results

As the purpose of our model is to generate peptide helix structures, the number of valid helix structures generated and their qualities are used to measure model performance. The raw PDB dataset was first encoded into angle representation via a Python script as mentioned in the Data section. Then after 30 iterations of training, real data was used to calculate loss and weights were updated for both G and D. Lastly, when training was done, using a fixed noise z' as input, we tested our WGAN-GP model against the baseline DCGAN model, and results are as follows.

For the quality of the generated data, FID score, which represents the distance between the inception feature vectors of the real data and generated data, is used as the metric [5]; and pre-trained InceptionV3 model is used to extract the inception features of data [1]. As shown in Table 1, for well-trained WGAN-GP, the final FID score is around 5.3; but for baseline DCGAN, FID can be as high as 91. Since lower FID score means better data quality, this verifies that the inception quality of WGAN-GP is better than that of the baseline model. Further, WGAN-GP has a lower tolerance range than DCGAN, meaning its generated data quality is more stable than DCGAN's.

For the number of valid helix structures generated, or the helices count, we performed a manual count on the iteration with the lowest FID score. As seen in 1st column of Table 1, out of the 20 structures generated after each iteration, for DCGAN, 8 are valid helices during its top performing iteration 3100; whereas for WGAN-GP, all 20 are valid during its top performing iteration 6100. This 40% versus 100% validity demonstrates the convergence nature of WGAN-GP after training.

Aside from the quantitative analysis, we also did some visual inspection on the Generator-generated data after decoding them. We implemented a visualization method using nglview [2], and the 3D generated helix structure is captured as screenshots in Table 2. As seen from the figures, structures generated by DCGAN haven't evolve much as training progresses; while for WGAN-GP, the helix structure becomes clearer and better developed as more iterations are running.

Model	Helices count/Total	FID Score	Tolerance
DCGAN	8/20	91.3773	$\pm$56.1193
WGAN-GP	20/20	5.3454	$\pm$1.6397

Table 1: Helices count and FID score of WGAN-GP and DCGAN

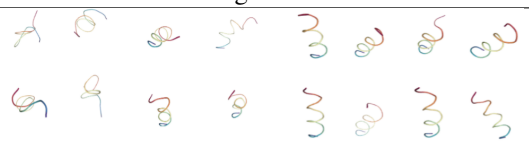
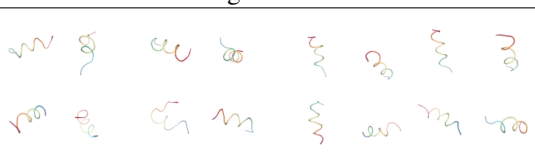
WGAN-GP generated results				DCGAN generated results			
							
iteration =100	iteration =500	iteration =5100	iteration =6100	iteration =100	iteration =500	iteration =3100	iteration =5000

Table 2: Helix Structure screenshots generated by WGAN-GP and DCGAN

6 Discussion

In this project, we designed a novel WGAN-GP model with CNN as feature extractor to generate peptide helices. Compared to the baseline model - DCGAN, its main distinction is using the Wasserstein distance with a penalty term to solve the gradient vanishing problem, which can slow down or even halt the learning process [8]. Our experiment have demonstrated WGAN-GP model can converge easier compared with with our baseline model - DCGAN. This indeed helped to avoid the model collapse issue during training, which is one major difficulty in training GAN model.

Figure 3 indicates the Generator and Discriminator losses for both WGAN-GP and DCGAN. Although the Discriminator loss for both converges to zero during training, Generator loss oscillates between 0 and 10 with no visible convergence trend for DCGAN, while it begins to decrease and approaches 1 after 20000 iterations for WGAN-GP. The difference in loss performance is caused by gradient vanishing [3, 9]: when the Discriminator reaches the optimal, it almost always predict `"real"` for real sample data and `"fake"` for generated sample data, causing the Generator loss to flatten and a zero gradient to be used for update even though the Generator has not reached optimal. To counter this problem, WGAN-GP uses Wasserstein distance with a gradient penalty term, as shown in equation 2. It penalizes the model when the gradient is away from its target norm value 1 [8]. By dealing with gradient vanishing, the penalty term also helps with non-converging results. As shown in Figure 4, Wasserstein distance converges to around 2 in the end, confirming the convergence of the loss function of our model.

To summarize, compared with standard DCGAN, our WGAN-GP model performs better in terms of FID score, loss convergence and stability.

Our conclusion is consistent with that of the paper and our final results meet the expected performance. However, a limitation on the current model is that it's costly and time-consuming to manually judge whether a large number of the generated structure are valid helices or not, so possible future work is to design a scoring function regarding helix structures. Some other future works would include using the generated helix structures in current D-peptide design method.

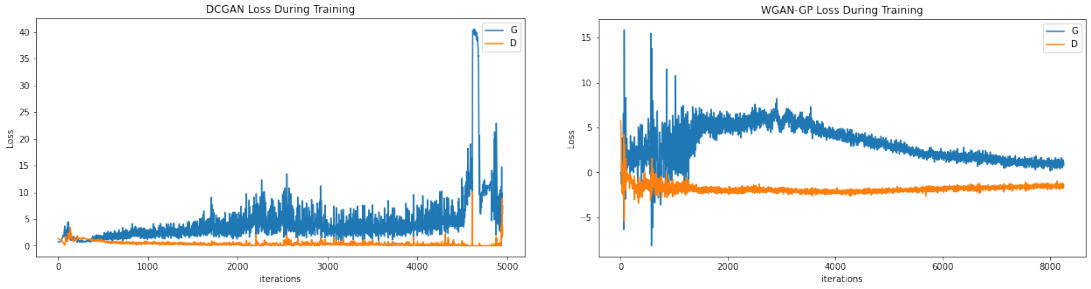


Figure 3: Generator and Discriminator loss of WGAN-GP and DCGAN

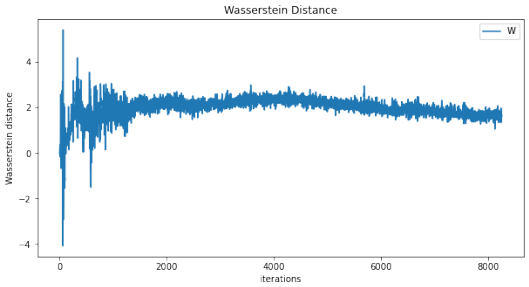


Figure 4: Wasserstein distance of WGAN-GP

7 Attributions

All group members contributed equally. Xuezhi Xie worked on WGAN-GP architecture, data-prepossessing and visualization. Zi Han Zhao took care of the baseline model, and model performance evaluation.

References

[1] Keras documentation: Inceptionv3. https://keras.io/api/applications/inceptionv3/. Accessed on 2020-12-16.

[2] Nglview 2.1.0 documentation. http://nglviewer.org/nglview/latest/api.html#. Accessed on 2020-12-16.

[3] M. Arjovsky and L. Bottou. Towards principled methods for training generative adversarial networks, 2017.

[4] J. Brownlee. How do convolutional layers work in deep learning neural networks? https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/, Apr 2019. Accessed on 2020-12-16.

[5] J. Brownlee. How to implement the frechet inception distance (fid) for evaluating gans. https://jonathan-hui.medium.com/gan-how-to-measure-gan-performance-64b988c47732, 2019. Accessed on 2020-12-16.

[6] O. Dor and Y. Zhou. Achieving 80% ten-fold cross-validated accuracy for secondary structure prediction by large-scale training. *Proteins: Structure, Function, and Bioinformatics*, 66(4): 838–845, 2007.

[7] M. Garton, S. Nim, T. A. Stone, K. E. Wang, C. M. Deber, and P. M. Kim. Method to generate highly stable d-amino acid analogs of bioactive helical peptides using a mirror image of the entire pdb. *Proceedings of the National Academy of Sciences*, 115(7):1505–1510, 2018.

[8] J. Hui. Gan — wasserstein gan & wgan-gp. https://jonathan-hui.medium.com/gan-wasserstein-gan-wgan-gp-6a1a2aa1b490, 2018. Accessed on 2020-12-16.

[9] L. Weng. From gan to wgan. https://lilianweng.github.io/lil-log/2017/08/20/from-GAN-to-WGAN.html, 2017. Accessed on 2020-12-16.