Protein Structure Prediction

FINAL PROJECT

Yomna Gamal | Bioinformatics | 3 February 2022

Introduction

Proteins are chains of chemical units, called amino acids, that fold to form three dimensional structures. The ability to predict a protein's structure from its amino acid sequence remain to be the most elusive yet rewarding challenges in computational biology. The structure determines the protein's function, which can be used to help understand life threatening diseases and accelerate drug discovery. However, experimental methods for solving a protein's structure are both time consuming and costly, and they only account for a small percentage of known protein sequences.

Earlier computational methods for predicting protein structure includes molecular dynamics (MD), which use physic-based equations to simulate the trajectory of a protein's folding process into a stable final 3D conformation. However, this method is computationally expensive and ineffective for larger proteins. Other approaches include using co-evolutional information to predict the residue-residue contact map, which can be used to guide structure prediction methods. With the help of deep learning architectures like convolutional neural networks, contact prediction remains to be the prevailing methods in structure prediction. However, because these method does not provide an explicit mapping from sequence to structure, they lack the ability to capture intrinsic information between the sequence and structure.

RGNs solve that issue, as it is an end-to-end differentiable model that jointly optimizes the relationships between protein sequences and structure. However, because it uses RNNs as internal representations, training can be both difficult and time consuming.

Model	Prediction Speed	Training Time
Rosetta, I-Tasser, Quark	Hours to days	N/A
Raptor X, DeepContact + CONFOLD	One to few hours	Hours
Recurrent geometric networks (RGNs)	Milliseconds	Weeks to months

Figure 1 Prediction and Training Speeds of Structure Prediction Methods

There is another solution (UTGN), it is a modification to the RGN architecture. Inspired by the recent successes of the transformer models in the NLP community, replacing the LSTMs in the RGN model with the encoder portion of the Universal Transformer (UT) as the internal representation. By doing so, the model is faster to train and it is contextually informed by all subsequent symbols. As a result, it is better at learning global dependencies among the amino acids than RNNs.

The UTGN operates by first taking a sequence of vector representation of the amino acids and applying the universal transformer architecture to iteratively refine a sequence of internal representations. Next, it uses the internal states to construct three torsional angles for each position, which is used to construct the 3D Cartesian structure.

So, the objective is to use deep learning to predict the 3D structure of a protein using just the amino acid sequence.

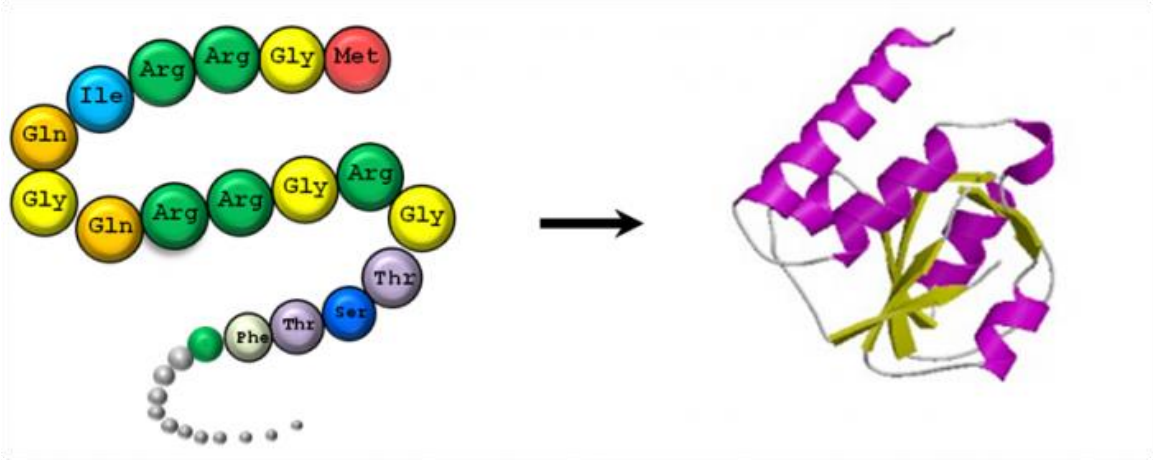


Figure 2 Amino acid to 3D structure

Methods

There are two methods, the first depends on RNN, the other method uses transformers instead of RNN and table shows comparison between both.

RNN	Transformers
Takes inputs sequentially, making it harder to parallelize	Take the entire sequence at once as an input
Have problems as exploding / vanishing gradient problems	Less susceptible to exploding / vanishing gradient problems
Unstable (different initialization produces very different results).	Better at creating internal states that considers global dependencies between input and outputs
Information fades as time step increases	

RECURRENT GEOMETRIC NETWORK (RGN)

End-to-End Differentiable Learning of Protein Structure paper introduce an approach based entirely on machine learning that predicts protein structure from sequence using a single neural network. The model achieves state-of-the-art accuracy when published (in 2019), and does not require co-evolution information or structural homologs. It is fast, making predictions in milliseconds.

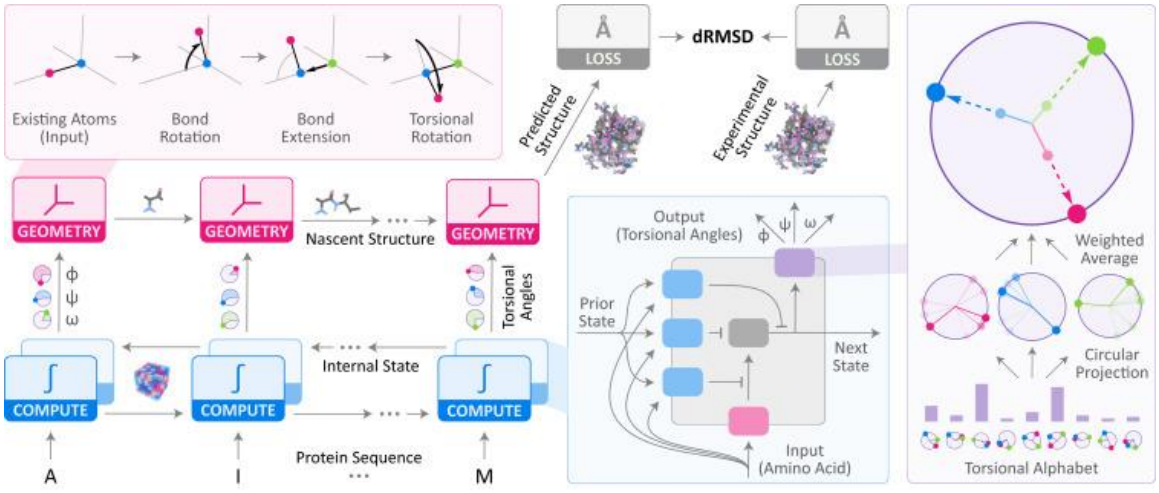
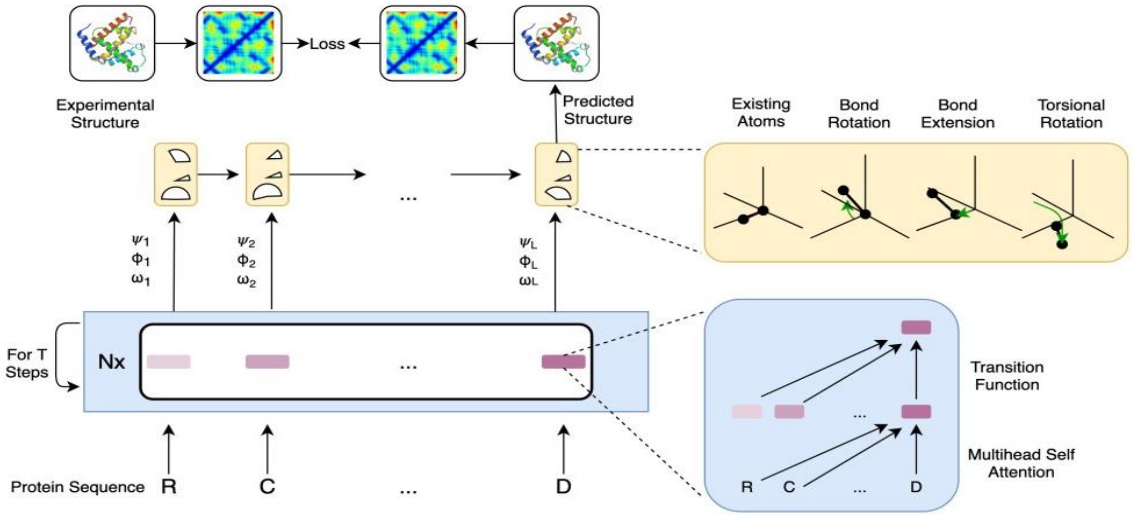


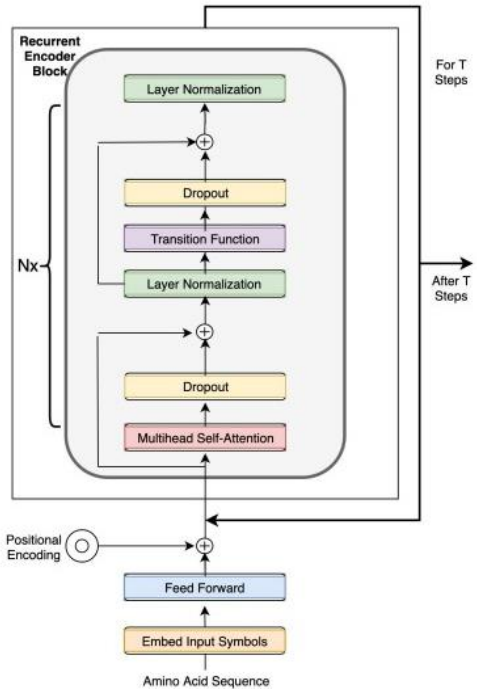
Figure 3 RGN

UNIVERSAL TRANSFORMING GEOMETRIC NETWORK (UTGN)

Keep everything from RGN architecture, except the way the internal states are represented. Replace the Bi-LSTM with a Universal Transformer Encoder to get faster and easier training with good results.



Represent amino acid sequence just like in the RGN.
Feed into a feed forward layer.
Add positional encodings to include information of the position of the amino acid.
Apply multi-head self-attention. At each position, the internal representation learns to pay different amounts of attention to previous inputs.
Next, apply a dropout to help prevent overfitting.
Residual connection to allow information flow to skip layers.
Layer normalization to allow better gradient flow.
Transition function can either be feed forward or separable convolution.
Dropout.
Residual connection.
Layer normalization.
The entire gray part called a layer. After passing through the gray area once, the internal representation will be passed through the same layer architecture for N times. Each layer has different weights. The layers together called the encoder architecture. Once that is done, the resulting states are passed back into the encoder architecture T times. This refines the states. Also apply ACT mechanism to better focus on more ambiguously used amino acids.

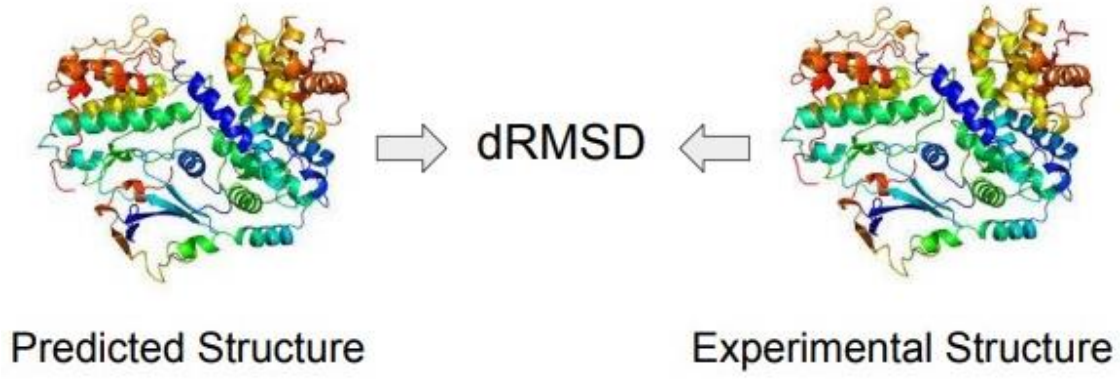


Loss Function

DRMSD LOSS

It is calculated by creating the distance map of predicted and experimental structure. Then find the distance between the distance maps. Perform backpropagation to update and optimize weights.

This loss function is translationally and rotationally invariant. It is differentiable.



Datasets

ProteinNet	SidechainNet
• Train / validation set contains all the PDB structures except: Structures less than 2 residues. >90% of structures were not resolved. • Contains mask records for missing residues in PDB structure. • Multiple logical chains are combined. • Methods for making TF records efficiently.	• It directly extends ProteinNet. • Methods for loading and batching SidechainNet data efficiently in PyTorch. • Methods for generating protein structure visualizations from model predictions.

Implementation

RGN

Its official code was using ProteinNet "for dataset" and implemented using TensorFlow.

The loss was used is distance-based root mean square deviation (dRMSD).

Since official code is old and I couldn't run it, I use another implementation from SidechainNet tutorial.

The other implementation was using SidechainNet "for dataset" and using PyTorch.

In this implementation we can use either RMSE or dRMSD, but dRMSD takes too long for training (3:30 hrs. for only 1 epoch) and due to limitation in using Colab GPU, it was hard to use it so I use RMSE.

UTGN

Its official code was a modification on the RGN official code. So it is also using ProteinNet "for dataset" and implemented using TensorFlow, with same loss used (dRMSD).

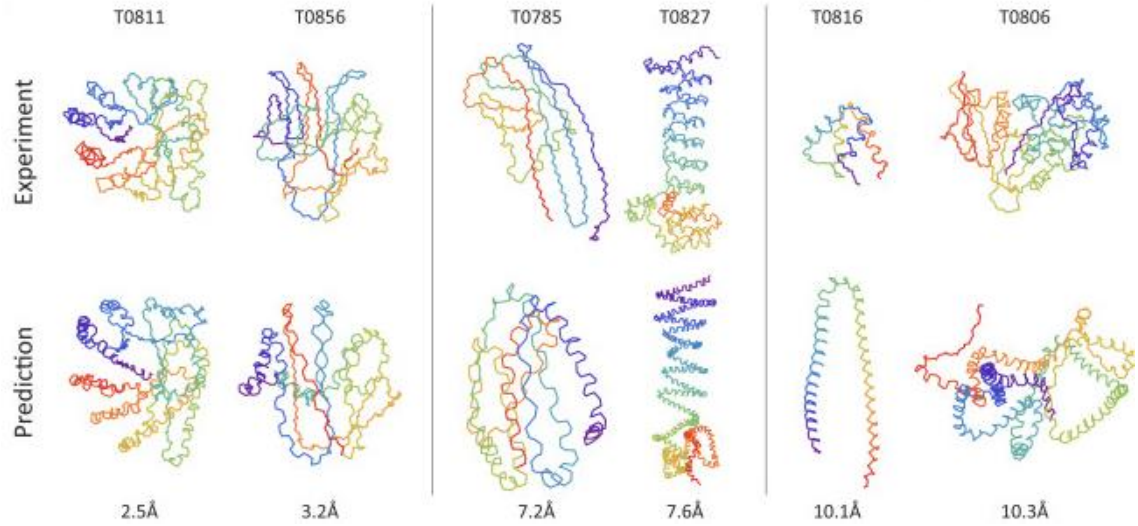
This implementation is full of bugs and after some modification it still didn't work correctly.

I tried to implement it from the beginning using PyTorch but unfortunately it still not working correctly.

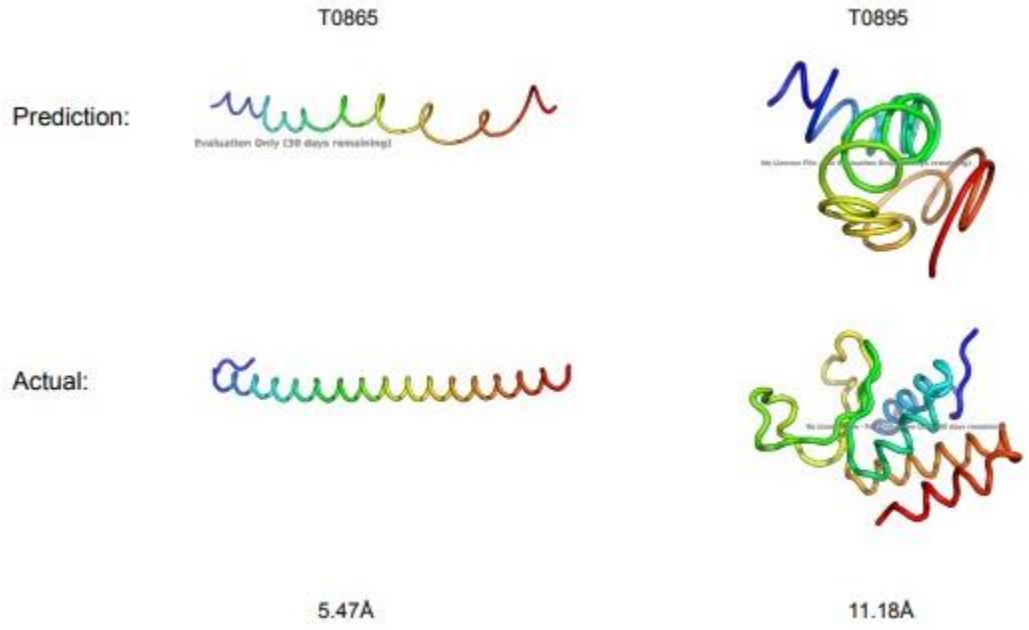
Results

OFFICIAL RESULTS

RGN



UTGN



Template Based Modeling

Model	dRMSD (Å)	TM score
RGN	17.8	0.200
UTGN-FF	17.6	0.198
UTGN-SepConv	**17.1**	**0.208**

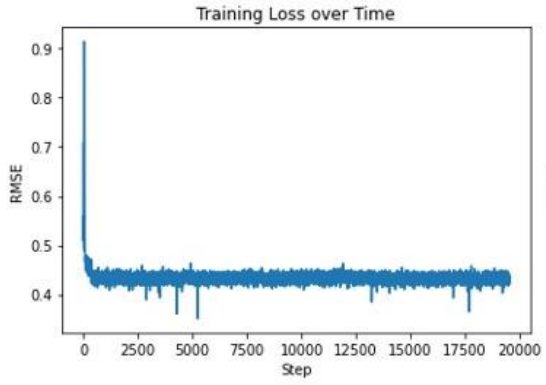
Free Modeling

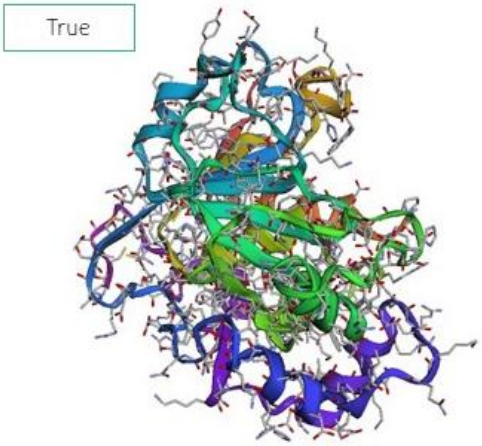
Model	dRMSD (Å)	TM score
RGN	19.8	0.181
UTGN-FF	19.4	0.174
UTGN-SepConv	**18.1**	**0.194**

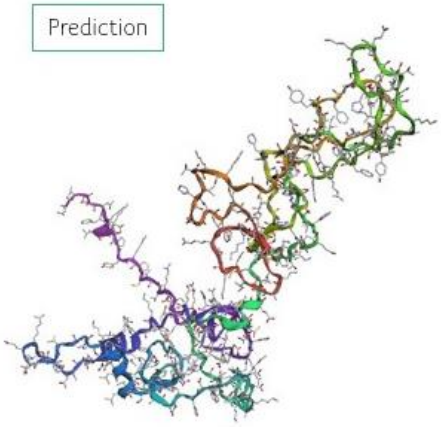
CURRENT RESULTS

Using RMSE as a loss function instead of dRMSD make results bad but it was the only solution to train faster as the epoch using dRMSD on CASP7 takes 3.5 hrs. and colab cant train model due to network interruption.

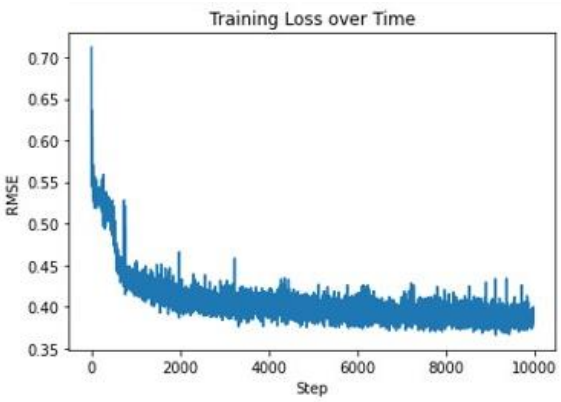
Sequences → Angles

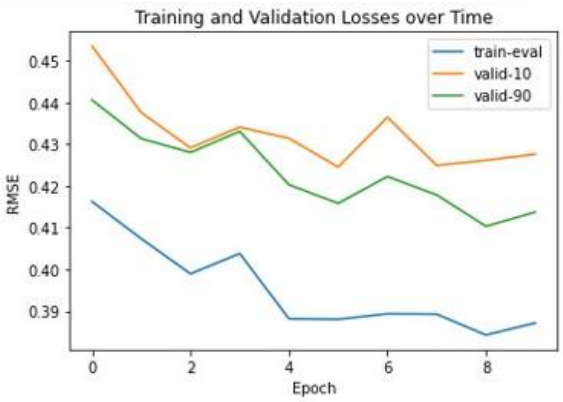


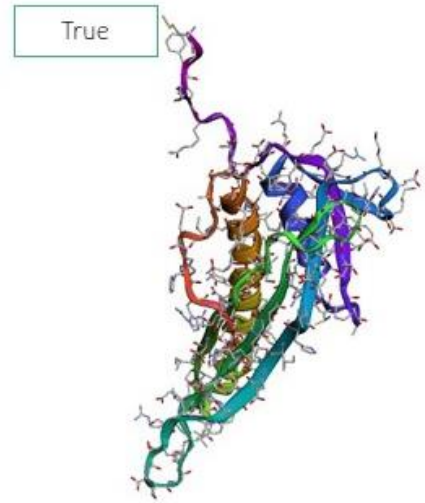


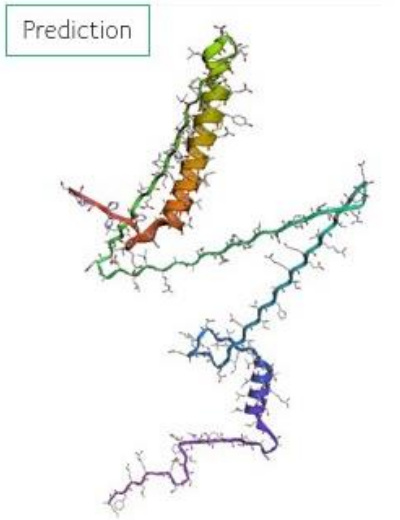


(Sequences, PSSMs, Secondary Structures, and Information Content) → Angles









Code

Final colab notebook, contains all work with PyTorch Models
https://colab.research.google.com/drive/1TbjKAcMeHbKz_mE5GnnVO53mJ5rVUj9z?usp=sharing

Old Colab notebook, contains trials made for working with official codes that uses TensorFlow
https://colab.research.google.com/drive/1PVHuGhD6pDsNCyW3g9WG7G_E2_lDQeBX?usp=sharing

Model weights and result of training
https://drive.google.com/drive/folders/1tjHLCoyx9Cp2bW4iVzQ-_WUKVg_6lCzi?usp=sharing

References

RGN paper: https://www.cell.com/cell-systems/fulltext/S2405-4712(19)30076-6
RGN official code: https://github.com/aqlaboratory/rgn
UTGN paper: https://arxiv.org/abs/1908.00723
UTGN official code: https://github.com/JinLi711/UTGN
Sidechainnet: https://github.com/jonathanking/sidechainnet
ProteinNet: https://github.com/aqlaboratory/proteinnet
Some for implementing UTGN:
https://pytorch.org/docs/stable/generated/torch.nn.TransformerEncoderLayer.html
https://github.com/andreamad8/Universal-Transformer-Pytorch
https://colab.research.google.com/github/pytorch/tutorials/blob/gh-pages/_downloads/transformer_tutorial.ipynb
https://colab.research.google.com/github/PytorchLightning/lightning-tutorials/blob/publication/.notebooks/course_UvA-DL/05-transformers-and-MH-attention.ipynb
https://pythonrepo.com/repo/minqukanq-transformer-pytorch-python-deep-learning