Determination of protein folds from amino acid sequence

Andreas Larsson, andlar4@kth.se

October 2024

1 Abstract

Protein folding is a complex problem, being part of research for more than 50 years. This project aims to evaluate how well deep-learning models can predict the secondary structure sequence merely from the amino acid sequence. The dataset used is publicly available on Kaggle. The deep-learning models developed was mainly LSTMs and encoder Transformers, utilizing the PyTorch package. One Support vector machine was also developed using scikit-learn. All models utilized a sliding window to predict the secondary structure. The best performing model was an encoder transformer with dropout, an embedding size of 256 and hidden size of 512 achieving a validation accuracy of 88%. Several conclusions could be seen from the experimental runs. LSTMs seemed to have more stable performance throughout epochs when using dropout, and saw great improvements when increasing the hidden layer size. The transformer did well with dropout, reducing the overfitting of the model, and an increase of attention heads and encoder layers saw small improvements in the model. The most improvement was seen when increasing the embedding layer size and hidden layer size.

2 Background and Motivation

Determining protein folds has long been a complex problem, usually involving time-consuming processes and has been part of research for more than 50 years [1]. This project aims to use deep-learning methods to investigate how much this problem can be simplified. The models will be trained on amino acid sequences and output the secondary structure each amino acid participates in. The project also aims to compare several different architectures, beginning with a standard Support vector machine (SVM) to serve as a benchmark to the latter deep-learning models.

The deep-learning models applied will be variants of RNN:s called LSTMs (Long short-term memory) and GRU as well as Encoder transformers. These models in general show promise in this field as for example RNNs provide a way to capture long term dependencies better than convolutional neural networks, which is beneficiary when determining β-sheets [2]. This mechanism also exists in the transformer architecture with the attention heads, as it allows the model to learn from several places simultaneously. A big part of the project will consist of finetuning these models, and attempting to find ideal hyperparameters such as batch size, hidden layer size, and learning rate. Different grades of regularization (such as dropout) will be tried. All the deep-learning models will be developed using the PyTorch package while the SVM will utilize scikit-learn.

3 Dataset Summary

The dataset used is publicly available at Kaggle but requires an account in order to download. It consists of several different columns, however, only 2 of them are relevant to this project. The ”seq” column is used as input for all models, as it contains the amino acid sequence. The ”sst3” column is used as the labels, where each character is the corresponding secondary structure the amino acid in the same position is a part of. "sst3" is a simplification of the "sst8" column and it was picked as it was believed initially that 8 different labels would make the model struggle to learn.

The dataset was therefore processed by only using the two previously mentioned columns and only considering amino acid chains with lengths between 20 and 50. Thereafter 5000 sequences with corresponding secondary structure labels were randomly chosen and a train/validation split was performed with 80% used for training and 20% for validation and testing.

4 Method

As the secondary structure is not solely reliant on the single amino acid, but also its neighboring amino acids, a sliding window is fed as inputs to the models. The size of the sliding window was set to 15, meaning that for each position in the chain the 7 amino acids before and the 7 after was also used as the input. In the cases where an amino acid was placed close to the edge, the sequence was padded with enough amino acids noted X, which generally means any amino acid, but in this project denoted the beginning or end of a sequence. The parts of the sequences containing X was then not considered when training and evaluating the model. Each sequence were then one-hot encoded and used as input to the models.

A SVM was then constructed using the scikit-learn package, using a RBF kernel, $C = 0.1$ and $\gamma = scale$. The other baseline models were then implemented using the PyTorch package, starting off with a vanilla RNN to initially investigate the performance of a basic deep learning architecture to the SVM. Three seperate LSTMs were then setup, comparing how the amount of layers would impact the performance. It was then chosen to proceed with the two-layer LSTM, as it performed quite likeworthy with the three-layer LSTM but had significantly shorter runtime. A two-layer GRU model was then tried, but chosen to move away from as it was worse than the two-layer LSTM in every way. For all the RNN baseline models the hidden size was set to 128, the learning rate to 0.001 and the batch size to 64. Lastly, the encoder transformer was set up, using a embedding size of 128, hidden size of 256, batch size 32 and 4 encoder layers. The results of all the *baseline* models can be seen in table 1. All the deep-learning models used the optimizer Adam.

The first experimental runs were then performed on the two layer LSTM. In table 2 all the performed runs can be seen. The varying parameters differed between runs among: learning rate, dropout, batch size and hidden size. The experimental runs then moved on to the encoder transformer. All the runs performed on the encoder transformer can be seen in table 3 and table 4. The parameters varied included: Embedding size, hidden size, dropout, number of attention heads and encoder layers. For all runs performed in the project the models were trained during 50 epochs.

5 Results

In table 1 the result for the models considered as the baselines can be seen. The highest validation accuracies was achieved with the two layer LSTM. It can also be clearly seen that all deep-learning models significantly outperform the more simplistic machine learning model (SVM).

Table 1: Overview of all tried experiments and the accuracies of the baseline models. More experimental runs will be presented in following tables.

Architecture	Structure	Validation accuracy	Training accuracy
SVM	Baseline	80.5%	-
Vanilla RNN	Baseline	84.8%	89%
LSTM	One-layer	86.8%	95%
LSTM	Two-layer	87.3%	95%
LSTM	Three-layer	87.1%	95%
GRU	Baseline	87%	95%
Encoder transformer	Baseline	86.9%	95.4%

In table 2 the results for all the experimental runs with the two-layer LSTM is visualised. Please note that the training accuracies are an approximation from a learning curve, and is meant as a visualisation of how overfitted the model is.

Table 2: The different variations tried with the two-layer LSTM.

Hidden size	Learning rate	Dropout	Batch size	Validation accuracy	Training Accuracy
128	0.01	0	64	86.7%	93%
128	0.1	0	64	41-48%	-
128	0.001	0	16	87.1%	95%
128	0.001	0	32	87.2%	95%
128	0.001	0	128	87.1%	95%
128	0.001	0.1	64	87%	95%
128	0.001	0.2	64	87.1%	95%
128	0.001	0.3	64	87.2%	95%
128	0.001	0.4	64	86.9%	95%
128	0.001	0.5	64	86.8%	95%
64	0.001	0.3	64	86.1%	94%
256	0.001	0.3	64	87.4%	95%
512	0.001	0.3	64	87.8%	95%

In table 3 the intitial experiments for the transformer can be seen. When good parameters was believed to be found, the embedding and hidden size were varied, which can be seen in table 4.

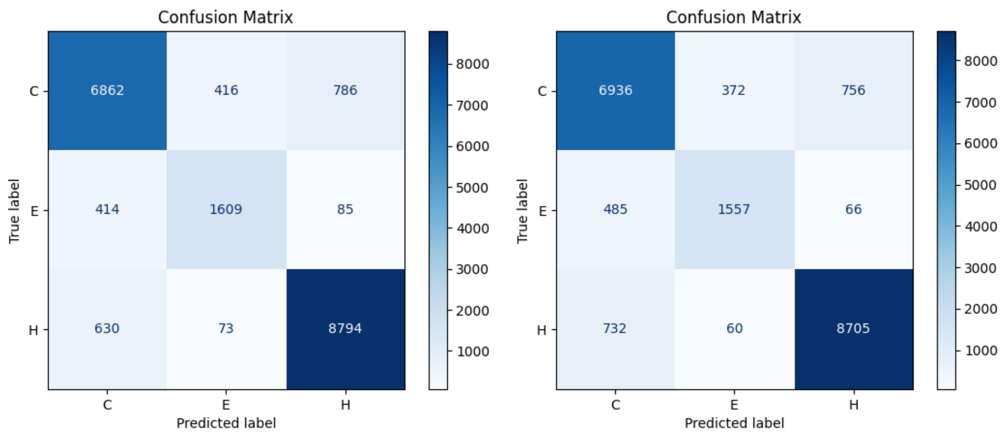
Table 3: The different variations tried with the encoder transformer. Learning rate was 0.0001 for all runs, the batch size was set to 32, embedding size 128 and hidden size 256.

Encoder layers	Dropout	Attention heads	Validation accuracy	Training Accuracy
1	0	8	86.7%	95.4%
2	0	8	86.8%	95.3%
3	0	8	86.7%	95%
4	0	2	86.7%	95.4%
4	0	4	86.8%	95.5%
4	0.1	8	87.2%	95%
4	0.2	8	87.4%	94.9%
4	0.3	8	87.5%	94.5%
4	0.4	8	87.5%	94.1%
4	0.5	8	87.4%	93.3%

Table 4: The different variations tried with the encoder transformer after determining the earlier hyperparameters. Learning rate was 0.0001 for all runs, the batch size was set to 32, dropout 0.5, encoder layers 4 and 8 attention heads were used.

Embedding size	Hidden size	Validation accuracy	Training Accuracy
256	256	87.8%	93.7%
64	256	86.8%	91.7%
128	128	87.4%	92.2%
128	512	87.7%	94.1%
256	512	88%	94.4%

In figure 1 the confusion matrix for the best performing transformer and the best performing LSTM can be seen. The labeling is C for coil, E for β-strand and H for α-helix.



(a) Best performing transformer

(b) Best performing LSTM

Figure 1: Confusion matrices for the two best performing models

6 Conclusion and Discussion

This project have evaluated how well deep-learning models can predict the secondary structure from an amino acid sequence. Generally the project has been a success with the best performing model reaching 88% validation accuracy. Most deep learning models reached at least 87%.

To some surprise the SVM performed relatively well, reaching 80.5%. The results are however expected, as all the deep learning models greatly outperform the SVM, with the vanilla RNN being the worst, reaching 84.8%, see table 1.

From the RNNs the LSTM showed the most promise. The two- and three-layer one performed quite likeworthy, but the training time were substantially shorter for the two layer one. Running experiments a learning rate of 0.001 seemed appropriate, as for example the learning rate of 0.1 didn't work at all. Batch size had no significant effect on the performance, and therefore 64 was kept. Dropout had only a little bit of effect on the model, it did however seem to be more stable when applying some regularization such as dropout. Therefore a dropout of 0.3 was selected for the final runs with the LSTM. Lastly the size of the hidden layer was experimented with. Increasing the size had an immediate effect on performance, reaching a result as high as 87.8% with a hidden layer size of 512. It did however not come without consequences as the training time increased significantly.

The transformer architecture saw several parameters being important to the performance. The amount of encoder layers played a somewhat small role, seeing some better performance with 4 encoder layers. The same was the case with attention heads, seeing small improvements with 8 attention heads instead of 4 and 2. Some improvements performance-wise could be seen when using dropout, but most importantly it reduced the overfitting of the model, therefore 0.5 was used as dropout. Lastly, the embedding layer size and hidden layer size were varied. It was evident that an increase in both saw great performance improvement, reaching a validation accuracy of 88%. As with the LSTM though, this came with the downside of the training time increasing significantly.

Table 1 shows the confusion matrices for the best performing transformer and the best performer LSTM. It can clearly be seen that the model learns the coils and helices with relative ease, and has a high accuracy for those. It does however seem that it struggles more with the β-strands, which might be a cause of there being a lower amount of those in the data, which can be seen in the confusion matrices. The matrices also shows that the models are very likeworthy, which is to be expected since there is a very small difference in terms of accuracy.

6.1 Further work and Limitations

This project has a few limitations. For example, only sequences between 20 and 50 residues long has been considered, therefore the capturing of long term dependencies might not be completely accurate. It should also be noted that fine tuning of hyper-parameters of the models could be tried to infinity, thus this project could keep going endlessly, and therefore it had to be restricted to the parameters mentioned and tried in this project.

It would therefore be interesting to build upon this project, as several other methods could be tried. For example other optimizers could be tried, other regularization techniques and the use of learning rate schedulers. Another interesting change would be to not exclude padded sequences as this might develop the model to learn where the edge of a sequence is. It could also be an idea to try out the more complex ”sst8” labels, as this likely would prove more challenging to the models, as well as varying the size of the sliding window.

7 Data and Code Availability

The data and code will be made fully available at https://github.com/andlar4/DDLS-project.

8 Acknowledgements

This project has been conducted with the help of chatGPT. Most of the vanilla architectures has been developed through chatGPT, whereas the finetuning and experimental runs have been further developed from the vanilla models. The chat history for SVM and RNNs can be seen here, and the chat history for the transformer development can be seen here.

9 References

[1] Jumper, J., Evans, R., Pritzel, A. et al. Highly accurate protein structure prediction with AlphaFold. Nature 596, 583–589 (2021). https://doi.org/10.1038/s41586-021-03819-2

[2] Ismi, D.P., Pulungan, R. Afihayati. Deep learning for protein secondary structure prediction: Pre and post- AlphaFold. Computational and Structural Biotechnology Journal 20, 6271-6286 (2022). https://doi.org/10.1016/j.csbj.2022.11.012