Predicting Protein Solubility from Amino Acid Sequences

Lubor Budaj, Matthew Dupraz, Anton Hosgood
Laboratory for Biomolecular Modeling, EPFL, Switzerland
Supervisor: Lucien F. Krapp

***Abstract*—Being able to predict protein solubility is applicable in many domains, but this task is difficult to perform without knowing environmental factors or the protein's physical structure. We use deep learning as well as traditional machine learning models to predict protein solubility given only its amino acid sequence. We evaluate these models using various metrics and find that classical machine learning approaches are still able to outperform more complicated deep learning models.**

I. Introduction

The solubility of a protein measures the ability of the protein to form a solution in water. Expressing soluble proteins is desired in research fields as well as in industrial sectors. However, given a protein's amino acid sequence, expressing it can require a significant investment of both time and capital. As such, being able to predict the solubility of a protein given its amino acid sequence is highly beneficial, as time and money would not be wasted expressing a protein which turns out being insoluble. In addition, the solubility of a protein depends on various factors, including pH, temperature, as well as the physical structure of the protein. These physical factors are either variable or complicated to determine, and so being able to predict a protein's solubility with only the amino acid sequence is desirable.

Several methods have already been deployed with mixed results. However, success has recently been observed in predicting the solubility of a protein given only its amino acid sequence using deep learning-based methods. Deep learning models are able to extract features themselves, so manual feature selection requiring existing domain knowledge should not be necessary for such methods to perform well, as opposed to other classical methods.

Predicting protein solubility is typically defined as a regression task, as solubility is not a discrete quantity. We convert this task into a classification task by setting a threshold on the solubility, after which we denote a protein as being soluble. Otherwise, the protein is insoluble. To tackle this classification problem, we chose to employ 1D convolutional neural networks (CNNs) and recurrent neural networks (RNNs), as these tools seemed to be the most adapted to handling sequences.

II. Exploratory Data Analysis and Data Processing

Our dataset[1] consists of 11,226 data entries containing amino acid sequences in FASTA format and a label of whether or not the protein is soluble or not. Solubility is typically represented as a real value between 0 and 1, but in order to have a classification task as opposed to a regression task, the solubility was set as a categorical variable. We observe that 66.8% of the proteins in our dataset are soluble, which means that our dataset is not balanced. This is accounted for by the criterion we used (BCEWithLogitsLoss), which takes as parameter the ratio of negative to positive samples.

The sequences in FASTA format can not be immediately fed into a neural network as they are stored as strings. In order to represent the sequences in such a way that they can be processed by a neural network, we use one-hot encoding. We observe 20 distinct FASTA characters in the data and encode each character with 20 bits. Concretely, a FASTA sequence of length N is transformed into a matrix of shape $(N, 20)$ with each row representing an element in the FASTA sequence and each column representing one of the characters. A 1 in a column indicates that the element is present in this position of the sequence. As a result, the sum of each row is equal to 1 as there is only one character in each position, and the sum of each column is the number of appearances of a character in a sequence. Each column is then passed as a separate channel containing a 1D sequence into the neural network.

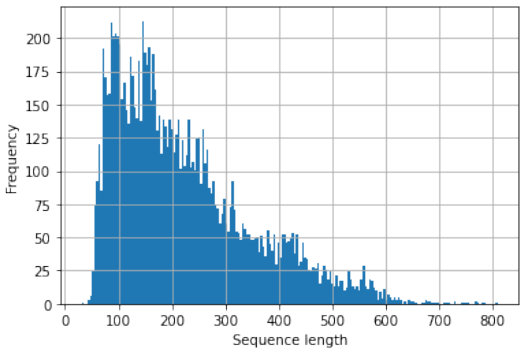

Fig. 1. Histogram of sequence lengths

The other challenge working with FASTA strings is the variable length of the sequences. We observe sequences in the dataset ranging from having 31 to 810 characters, with an average length of 223 characters. This can create issues as, for example, with a fixed CNN architecture producing a single output, the input may need to have a specific length or shape. We solve this by specifying a fixed length for all sequences. Sequences longer than the specified length are clipped to the

maximum length. Shorter sequences are padded with a specific character denoting padding. This character, when mapped to the one-hot encoding, corresponds to twenty 0 bits. As such, for the padding character all channels are set to 0 and the neural network should be able to treat padding differently in its classification.

Depending on the model, we implemented different padding strategies. The first padding implementation involved adding padding elements to the end of short sequences in order to increase their length to the predefined fixed length. The second padding involved adding an equal number of padding elements to either side of the sequence. In theory, the second strategy should perform better for CNNs as characters at the boundaries of the input are processed slightly differently than those covered fully by the convolution kernel. Adding padding characters to both sides will minimise this effect on the sequence itself. However, in our testing, the effect of this change was minimal and the choice of padding strategy is largely inconsequential. For the RNN model, we added padding elements to the beginning of the sequence. Using other padding schemes, recurrent layers could "forget" what it learned if it encountered a large succession of padding elements at the end of the sequence.

III. Models and Methods

We implemented our neural network architectures using PyTorch.

Our first model (Model 1) to try was a CNN followed by fully connected layers. The CNN aims to quickly reduce the number of channels down from 20 to create a lower-dimensional representation which is then fed to a neural network comprised of linear layers which outputs a final classification.

A second model (Model 2), in addition to convolutional and fully connected layers, uses gated recurrent units (GRUs) as part of an RNN. This mechanism allows our model to have a sort of memory which is used in the classification task.

The third model (Model 3) we tried involved first embedding the sequence residues into low-dimensional space with a linear transformation. The result is then fed into a CNN and fully connected layers. This model allowed us to look at the embedding after training and try to infer some information from it. One example of such an embedding obtained is show in Fig. 2. We could observe that in this case, the most hydrophobic residues are grouped together. It should be noted that this embedding varied greatly as we were training.

This observation suggested that the hydrophobicity of individual residues plays an important role in determining the protein's solubility.

Besides the neural networks, we tried to approach the problem with more classical tools, such as logistic regression or the Support Vector Machine (SVM). In order to use them, we had to extract a set of features from the sequences. We used residue occurrences and frequencies (this was separate as each sequence had different length), similarly occurrences and frequencies of pairs of consequent residues. For this we used the methods from scikit-learn.

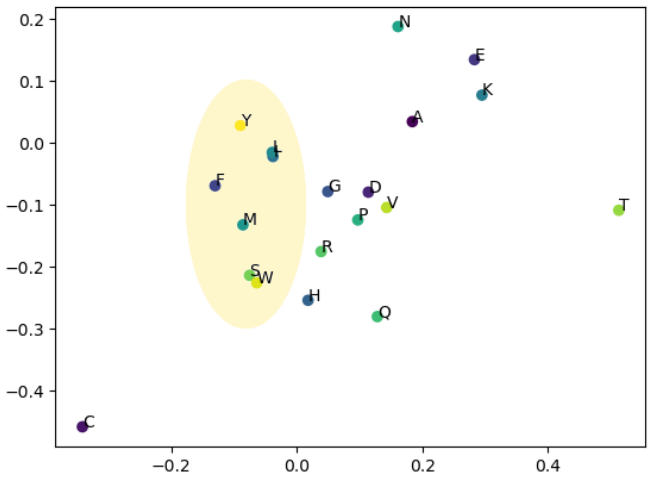

Fig. 2. Residue embedding into 2-dimensional space, with group of hydrophobic residues highlighted

IV. Results

To evaluate our models, we use a suite of metrics. From confusion matrices containing the number of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FP), we calculate the following metrics:

$$\begin{aligned}
\text{accuracy} &= \frac{TP+TN}{TP+TN+FP+FN} \\
\text{precision} &= \frac{TP}{TP+TN} \\
\text{recall} &= \frac{TP}{TP+FP} \\
\text{F1 score} &= 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}
\end{aligned}$$

We also use the predicted standard deviation and the Under the Receiver Operating Characteristic Curve (ROC AUC) score to evaluate our models.

To train the models we used the BCEWithLogits loss provided by PyTorch. However, this metric was only useful for monitoring the evolution of a given model. The loss was not necessarily correlated to the other scores.

Fig. 3 shows an example of evolution of the loss of a model when training and illustrates an important problem we had throughout the whole project. Most of the time our models were overfitting. As a potential solution we used L2 regularisation and dropout layers, but these techniques had only limited success. If we tried to fight overfitting too much, our models would not train at all. To tackle this issue we exported the weights of the model throughout the training so that we could keep those where the performance was best. Figures 4 and 5 show losses and metrics of model 2 during its training over 100 epochs. The values in the plots are averages over 10 runs with different seeds. We can see that the model

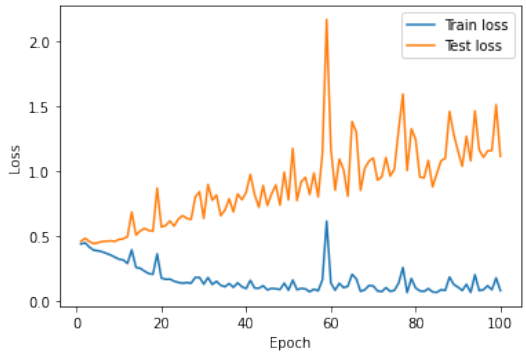

Fig. 3. Training and test losses for Model 1

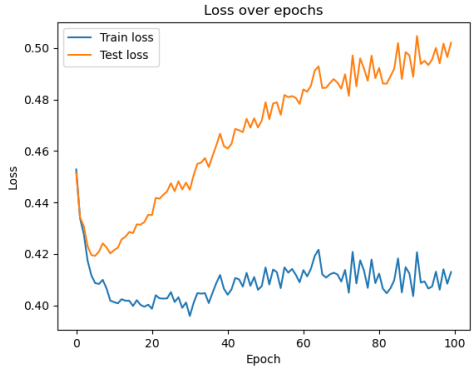

Fig. 4. Training and test losses for Model 2 (average over 10 runs)

is training only within the first 10 to 20 epochs, afterwards it begins to overfit. Consequently, we report the statistics after epoch 17, when the (average) ROC AUC was the highest. We treated the cases of Model 1 and Model 2 similarly.

TABLE I
Results obtained with different models (average over several runs)

Metric	ACC	PREC	REC	F1	ROC AUC
Model 1	0.660	0.766	0.738	0.747	0.661
Model 2	0.666	0.706	0.791	0.731	**0.723**
Model 3	0.653	0.827	0.713	0.755	0.631
Logistic (single)	0.696	0.867	0.715	0.798	0.721
Logistic (pairs)	0.705	0.829	0.733	0.798	0.714
SVM (single)	0.696	**0.925**	0.701	**0.809**	N/A
SVM (pairs)	**0.713**	0.846	**0.733**	0.808	N/A

When compared with logistic regression and the SVM, the neural networks were on average worse as seen in Table I. When logistic regression and the SVM were ran on occurrences of pairs of residues, the results were slightly better than just running on occurrences of single residues, but the difference was not so significant. Table I is missing ROC AUC values for the SVM, since the output of the model is the classification instead of probabilities (hence varying the threshold has no effect).

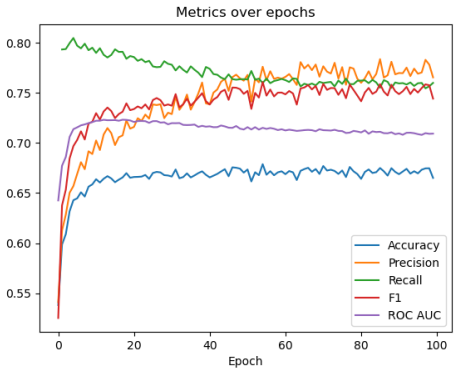

Fig. 5. Metrics for Model 2 (average over 10 runs)

V. Discussion

It was quite surprising to see that the simpler models (logistic regression and SVM) fared generally better than the deep learning models that we used. Indeed, we would expect that the convolutional neural networks would be capable of counting residues or pairs of residues. Perhaps one explanation would be that the performance of our models was thwarted by overfitting, which we were not able to successfully eliminate. Or perhaps the architectures we tried were not optimal for solving the solubility problem. However, it should be noted that we got the best ROC AUC with Model 2.

An interesting observation is that regression with pairs of residues performed just slightly better than using just single values. It would be interesting to take this one step further and consider occurrences of longer "words" of residues in the sequences, however this would become quickly computationally intensive. Our results indicate that the order in which the residues appear is important, but perhaps the data is not sufficient to determine solubility accurately. It came as a surprise that the results we obtained with the SVM are not very different from the results reported by the team working on NetSolP [1] – a deep learning model trained on the same dataset. Indeed, this might point at the fact that FASTA sequences do not provide enough information for solubility prediction after all. Another possible explanation is that the quality of the data we used is not very good. Finally, it might also be the case that the task itself is ill-defined – the definition of solubility is not so clear. As stated earlier, solubility depends on many factors and different sources might state different values for the solubility of the same protein as the experimental conditions may completely change the outcome of the measurement.

VI. Summary

We used and compared several models for the task of predicting protein solubility from their FASTA sequences. We designed several deep learning models, combining CNNs, RNNs and fully-connected layers. We also performed regression using more classical tools, such as logistic regression and the SVM. The deep learning methods we designed were

outperformed by the regression models. Our best performing model was the SVM ran on occurrences of pairs of residues, but we did not observe significant improvement over regression over occurrences of single residues.

REFERENCES

[1] Vineet Thumuluri, Hannah-Marie Martiny, Jose J Almagro Armenteros, Jesper Salomon, Henrik Nielsen, Alexander Rosenberg Johansen, NetSolP: predicting protein solubility in Escherichia coli using language models, Bioinformatics, Volume 38, Issue 4, 15 February 2022, Pages 941–946, https://doi.org/10.1093/bioinformatics/btab801