Protein Function Annotation with Neural Networks

Author: Kyrylo Stadniuk

Introduction

This project addresses protein function prediction using deep learning. The task is to classify proteins into selected Gene Ontology (GO) biological process terms based solely on sequence data. I frame it as a multilabel classification problem and evaluate different architectures—LSTMs, CNNs, and transformers—on their ability to learn from amino acid sequences. The models are trained and validated on curated UniProt data, with performance measured via AUC and accuracy.

Protein Sequence Encoding Pipeline `pipeline.py`

This pipeline preprocesses protein sequence data for supervised multi-label classification based on Gene Ontology (GO) biological process annotations.

Selected GO Terms

- `GO:0007165` – Signal transduction
- `GO:0006468` – Protein phosphorylation
- `GO:0008284` – Positive regulation of cell population proliferation
- `GO:0043066` – Negative regulation of apoptotic process

Data Sources

Annotations: `uniprot_sprot_exp.txt` – UniProt proteins with experimental GO annotations.
Sequences: `uniprot_sprot_exp.fasta` – Corresponding protein sequences in FASTA format.

Preprocessing Steps

1. Label Mapping

- Filter GO annotations to the Biological Process (P) ontology.
- Build a binary multi-label vector per protein indicating presence/absence of each selected GO term.

2. Sequence Encoding

- Convert amino acid sequences into integer sequences using a 20-letter vocabulary (ACDEFGHIKLMNPQRSTVWY).
- Unknown amino acids and padding use a reserved index (PAD_ID = 20).
- All sequences are truncated or padded to a fixed length (SEQ_LEN = 1104).

Final Output

- **X.npy**: array of shape (N, SEQ_LEN) containing integer-encoded sequences.
- **y.npy**: array of shape (N, 4) containing binary multi-label vectors.

Model Architectures `models.py`

This file defines four deep learning models for multilabel protein function prediction based on amino acid sequences. All models use an `Embedding` layer for sequence representation and are trained using binary cross-entropy with sigmoid activation.

- **`cnn_model_1`** : A simple 1D convolutional network with a single convolution layer (`Conv1D`), followed by global max pooling and dense layers. Effective for detecting local patterns in sequences.
- **`cnn_inception`** : A CNN with inception-style architecture using multiple convolution filters of different kernel sizes (3, 5, 9). Feature maps are concatenated and followed by dense layers. Captures multi-scale motifs.
- **`transformer_model`** : Incorporates a single-layer Transformer encoder (`keras-nlp`) after embedding, followed by global average pooling and dense layers. Designed to capture long-range dependencies.
- **`lstm_model`** : Uses stacked `LSTM` layers to model sequential dependencies, with batch normalization and dropout for regularization. Suited for temporal pattern recognition in sequences.

All models are compiled with the Adam optimizer and evaluated using accuracy and AUC, which is more appropriate for multilabel classification.

Training

I spent half the day trying to train an LSTM model, but it just wouldn't learn — it latched onto a single class and predicted it almost every time. Turns out the issue wasn't with the model itself but with how I structured the dataset: I had incorrectly assigned each protein to just one ontology, even though proteins can have multiple GO annotations. Once I converted the task into a proper multilabel classification problem, the models finally started to train.

After diving into LSTMs and getting excited about how powerful they're supposed to be, I was disappointed to see them underperform. Despite experimenting with several architectures, LSTMs trained painfully slowly and never produced good results — maybe I missed something, but I couldn't track down the issue.

Remembering how effective CNNs were for exon prediction in a previous assignment, I gave them a shot. That decision paid off: CNNs trained quickly and reached solid performance. They turned out to be the most reliable baseline across the architectures I tested.

As for Transformers — as expected, they were the slowest to train. I experimented with different learning rates (0.01, 0.001, 0.0001), but the AUC just oscillated around 0.65 without much improvement. Again, I suspect there might have been a mistake on my end, but I didn't manage to resolve it in time.

Paradoxically, methods that are supposed to shine with sequential data underperform compared to CNNs.

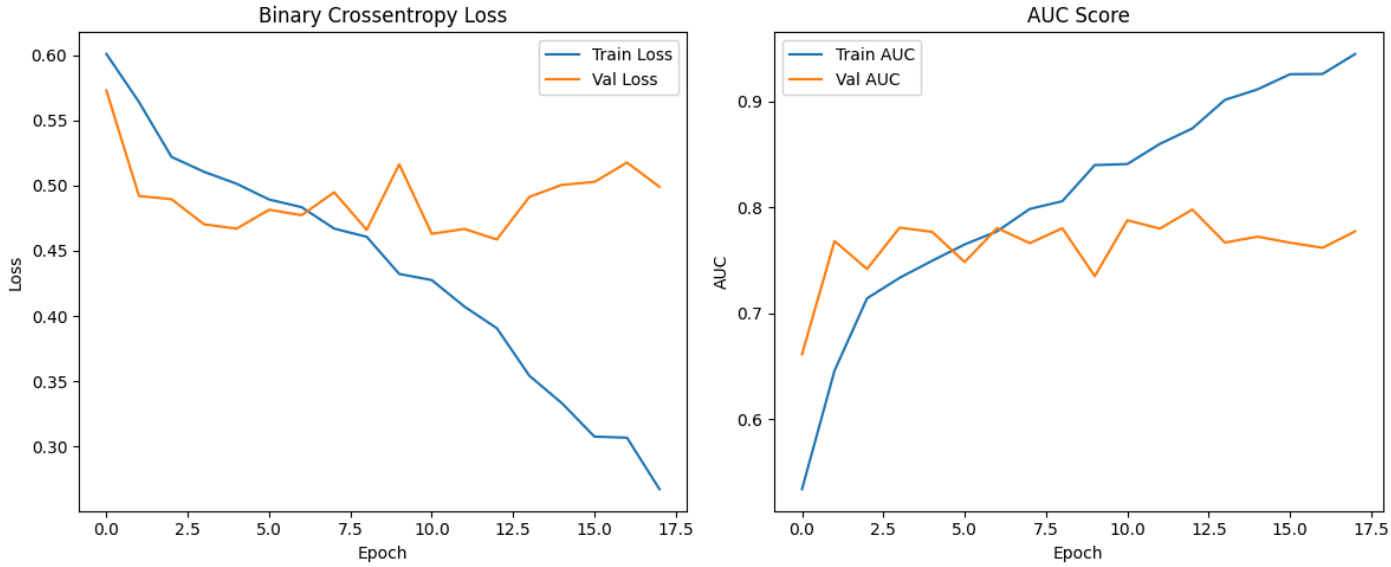

Figure: Example Inception CNN training curves. The model was trained after 12 epochs. It stopped early after 17th epoch.

I intended to explore the pretrained models, but as the example script with the example data ran for 3 hours, I did not use it with this data.

Results

Model	Test AUC	Test Accuracy	Test Loss
Inception	**0.808**	**0.561**	**0.462**
CNN	0.784	0.5496	0.4983
	Validation AUC	**Validation Accuracy**	**Validation Loss**
LSTS	0.5247	0.3023	0.5993
Transformer	0.6692	0.3836	0.556

The Inception-style CNN achieved the best performance with a test AUC of 0.808 and accuracy of 0.561, outperforming a simpler CNN baseline (AUC 0.784). Both models were trained and evaluated on a held-out test set. In contrast, the LSTM and transformer models were only evaluated on the validation set due to time constraints. LSTM showed poor performance (AUC 0.525), likely due to training issues, while the transformer reached a moderate AUC of 0.669. Overall, CNN-based models trained faster and performed better, making them the most effective choice for this task.

Conclusion

In this assignment I was working on protein function annotation using deep learning, focusing on classifying proteins by selected GO terms. I tested LSTM-based RNNs, CNNs, and transformers. Despite the sequential nature of protein data, LSTMs trained slowly and underperformed. Transformers showed unstable training and moderate AUC (~0.65), likely due to limited compute. In contrast, CNNs—especially a multi-kernel Inception variant—trained quickly and achieved the best results (AUC ~0.75).

While pre-trained protein language models weren't used due to time limits, they remain a promising direction. Overall, this work shows that properly framed multilabel CNNs can be effective and efficient for protein sequence classification.