A-Domain Prediction

-Adrita Das

Architectures

- **Model Architecture**

1. Input Layer: The input layer receives input sequences of amino acid types encoded as one-hot vectors. The input size is determined by the number of amino acid types, including the missing token and padding token.

2. LSTM Layer: The LSTM layer is responsible for capturing sequential dependencies in the input sequences. It takes the one-hot encoded input sequences and processes them through LSTM units. The hidden size determines the number of hidden units in the LSTM layer. The number of layers determines the depth of the LSTM architecture, and the dropout parameter controls the dropout probability between LSTM layers.

1. Fully Connected Layer: The output of the LSTM layer is passed through a fully connected layer. The hidden size of the LSTM layer is used as the input size for the fully connected layer, and the output size is determined by the number of output classes, obtained from the dataset label map.

Hyperparameters

- The model's hyperparameters used in the example code are as follows:
- Input_size : The number of amino acid types in the input sequences, including missing token and padding token
- Hidden_size :The number of units in LSTM layer. It was set to 256
- Num_layers : The number of LSTM layers in the model. It was set to 2.
- Dropout : The dropout probability of the LSTM layer. It was set to 0.2 here, i.e 20% of the LSTM units will be dropped randomly during training.
- Output_size : The number of output classes, obtained from the dataset label_map.

Criterion and Optimizer

- The criterion(loss function) and optimizer used for training the model are as follows:
- Criterion: Cross Entropy Loss.
- Optimizer : Adam
- Learning Rate : 0.01

Performance of the Different Models

- The performance of this model was at 28.7% accuracy. The performance of the previous LSTM model in which no dropout or regularization was applied, was significantly low.
- Adding regularization and increasing the number of layers, definitely boosted the model accuracy.
- Also, very complex models didn’t quite seem to be working with this dataset, as the data was very small.
- The change in the hyperparameters did not quite change the model performance. Increasing the hidden_size of the LSTM units did boost model performance.

Best Architecture and Hyperparameters

- I tried different architectures and hyperparameters to achieve the best performance.The final architecture has 2 layers LSTM layers. I used the following Hyperparameters:

1.Learning Rate: 0.01

2.Batch Size : 32

3.Number of epochs: 500

4.Optimizer:Adam

5.Loss: CrossEntropy

- I trained the model for 500 epochs.

BERT Model

- The model architecture provided appears to be a variant of the BERT (Bidirectional Encoder Representations from Transformers) model. Here's a description of the architecture in points:

1. BERT Model: The main component of the architecture is the BERT model, which is a transformer-based model for natural language processing tasks.
2. Embeddings Layer: The BERT model starts with an embeddings layer responsible for converting input tokens into fixed-dimensional embeddings. It includes:
    1. Word Embeddings: A trainable embedding layer that maps input tokens to continuous vectors of size 768.
    2. Position Embeddings: An embedding layer that encodes the position information of the tokens in the input sequence.
    3. Token Type Embeddings: An embedding layer that encodes the token types (e.g., sentence A vs. sentence B) for tasks involving multiple sequences.
    4. Layer Normalization: A layer normalization step to normalize the embeddings.
    5. Dropout: A dropout layer that randomly masks a fraction of the embeddings to prevent overfitting.

BERT Model

Transformer Encoder: The BERT model utilizes a stack of transformer encoder layers to capture contextual information from the input sequence. The encoder consists of 12 transformer layers, each referred to as BertLayer.

- Attention Mechanism: Each BertLayer includes a self-attention mechanism called BertSelfAttention. It performs attention computations using query, key, and value vectors derived from the input embeddings.
- Intermediate and Output Layers: The attention output is passed through a feed-forward neural network called BertIntermediate, which increases the dimensionality of the representation. The output is then processed by BertOutput, which applies another linear transformation and layer normalization.
- Layer Normalization and Dropout: Layer normalization and dropout are applied after each sub-layer in the transformer encoder for stabilization and regularization.
- Pooler Layer: The BertPooler layer aggregates the contextualized representations from the last transformer layer to create a fixed-size representation for downstream tasks. It uses a linear transformation followed by a hyperbolic tangent (tanh) activation function.

BERT Model Performance

- The BERT Model did not perform well on the dataset and gave an accuracy of 16.4% on the test dataset.