MLBA Assignment 1 - Group 80

README

● **Input Files:**
The program takes paths to the training data as the user input. We have used the files provided on Kaggle as our training and testing data. For training data: train_data.csv For testing data: test_data.csv
● **Output Files:**
The program outputs a CSV file Group_80_Predictions.csv containing the predicted binary class labels (0,1) for the training data. The name of the prediction file submitted on Kaggle was different, though.
Prediction file submitted on Kaggle: cnn_att_3.csv
● **Steps To Run the Python File:**
Make sure that all dependencies and libraries used by this program are installed on your system. A list of the required libraries are:
➔ Numpy
➔ Pandas
➔ Sklearn
➔ TensorFlow

Use the following steps to run the code:

Dependencies:

1. Sklearn
2. Tensorflow
3. Numpy
4. Pandas

Command Line Options:

1. Type the command "**python3 group80_code.py <train_csv_path> <test_csv_path>**" in the same directory as the notebook
2. Change the file paths to your own file paths. I have used the Google Drive paths as I saved my data there
3. Install all dependencies mentioned above.
4. After this, you can run all the cells u will get a CSV file as the output "**group80_predictions_output.csv**"

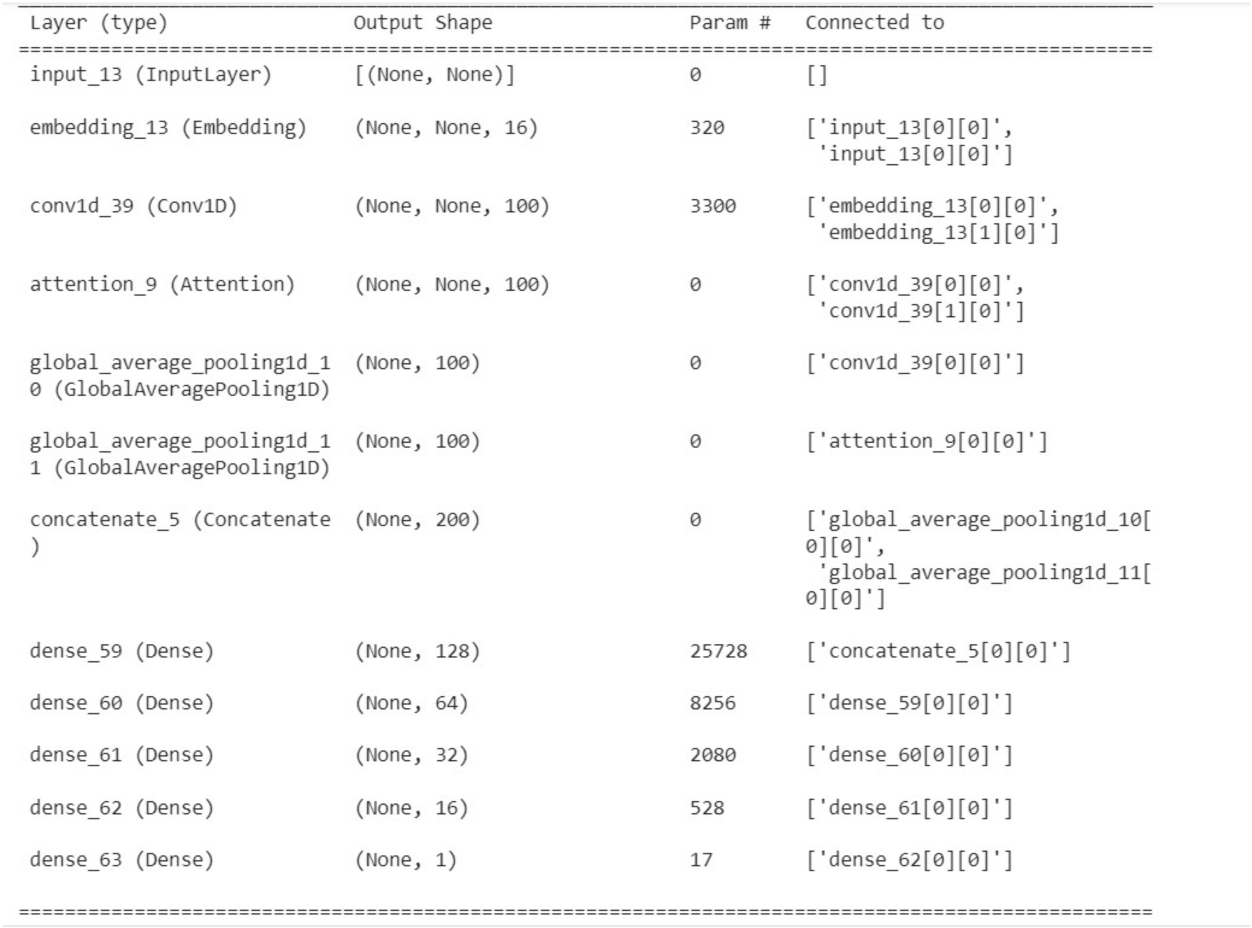
Why Run?

To predict the labels for the protein sequences.

Model Info:

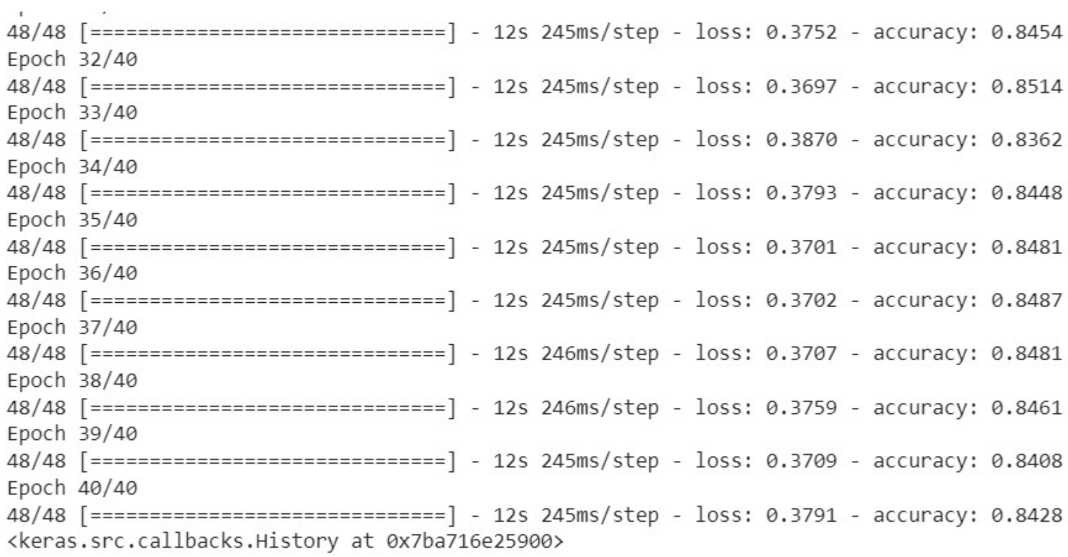
We used a CNN with attention. The CNN is a 1D CNN so it even checks the context of the neighboring Amino Acids by adding weights to it while making the embedding. Further, we use Self-attention to weigh the importance of each of the embedding going in. All these weights are learned by the model on its own. Finally, we add a few Dense Layers and an Output. We use Binary Cross entropy loss as the task is that of Binary Classification and the final layer has "sigmoid" activation.

Model Summary



```
 Layer (type)                Output Shape                 Param #   Connected to
==================================================================================================
 input_13 (InputLayer)       [(None, None)]               0         []

 embedding_13 (Embedding)    (None, None, 16)             320       ['input_13[0][0]',
                                                                     'input_13[0][0]']

 conv1d_39 (Conv1D)          (None, None, 100)            3300      ['embedding_13[0][0]',
                                                                     'embedding_13[1][0]']

 attention_9 (Attention)     (None, None, 100)            0         ['conv1d_39[0][0]',
                                                                     'conv1d_39[1][0]']

 global_average_pooling1d_1  (None, 100)                  0         ['conv1d_39[0][0]']
 0 (GlobalAveragePooling1D)

 global_average_pooling1d_1  (None, 100)                  0         ['attention_9[0][0]']
 1 (GlobalAveragePooling1D)

 concatenate_5 (Concatenate  (None, 200)                  0         ['global_average_pooling1d_10[
 )                                                                  0][0]',
                                                                     'global_average_pooling1d_11[
                                                                    0][0]']

 dense_59 (Dense)            (None, 128)                  25728     ['concatenate_5[0][0]']

 dense_60 (Dense)            (None, 64)                   8256      ['dense_59[0][0]']

 dense_61 (Dense)            (None, 32)                   2080      ['dense_60[0][0]']

 dense_62 (Dense)            (None, 16)                   528       ['dense_61[0][0]']

 dense_63 (Dense)            (None, 1)                    17        ['dense_62[0][0]']

==================================================================================================
```

Run Summary



```
48/48 [==============================] - 12s 245ms/step - loss: 0.3752 - accuracy: 0.8454
Epoch 32/40
48/48 [==============================] - 12s 245ms/step - loss: 0.3697 - accuracy: 0.8514
Epoch 33/40
48/48 [==============================] - 12s 245ms/step - loss: 0.3870 - accuracy: 0.8362
Epoch 34/40
48/48 [==============================] - 12s 245ms/step - loss: 0.3793 - accuracy: 0.8448
Epoch 35/40
48/48 [==============================] - 12s 245ms/step - loss: 0.3701 - accuracy: 0.8481
Epoch 36/40
48/48 [==============================] - 12s 245ms/step - loss: 0.3702 - accuracy: 0.8487
Epoch 37/40
48/48 [==============================] - 12s 246ms/step - loss: 0.3707 - accuracy: 0.8481
Epoch 38/40
48/48 [==============================] - 12s 246ms/step - loss: 0.3759 - accuracy: 0.8461
Epoch 39/40
48/48 [==============================] - 12s 245ms/step - loss: 0.3709 - accuracy: 0.8408
Epoch 40/40
48/48 [==============================] - 12s 245ms/step - loss: 0.3791 - accuracy: 0.8428
<keras.src.callbacks.History at 0x7ba716e25900>
```

Further models tried:

1. Convolutional Neural Network + Attention
2. Random Forest
3. Transformer
4. Easy Classifier
5. Bagging

The best accuracy given was **Convolutional Neural Network + Attention.**