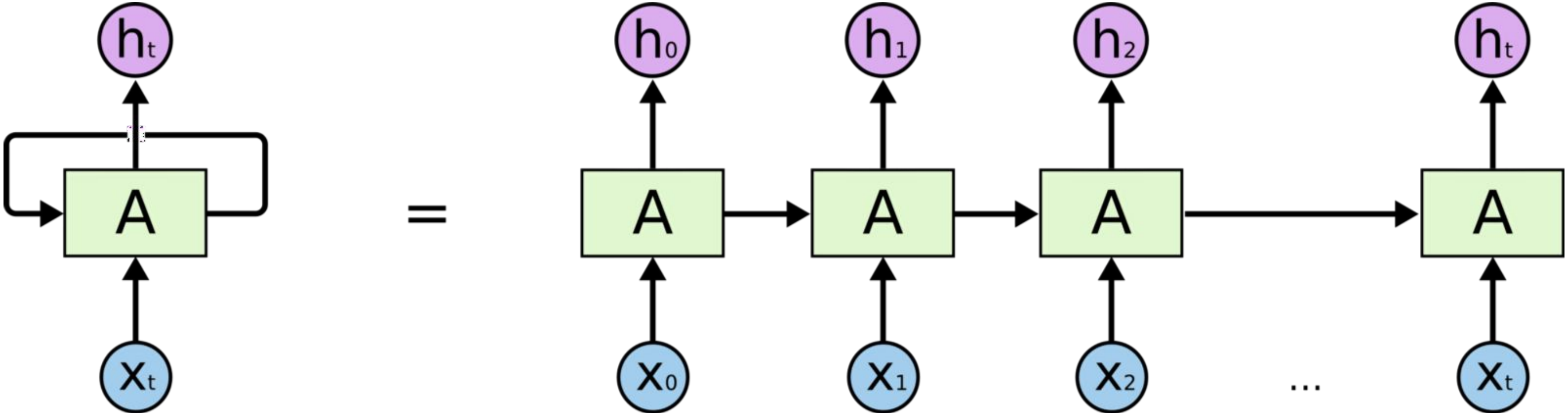
Twitter NLP

Ihor Volokhovych, MMAI, master’s 1st grade

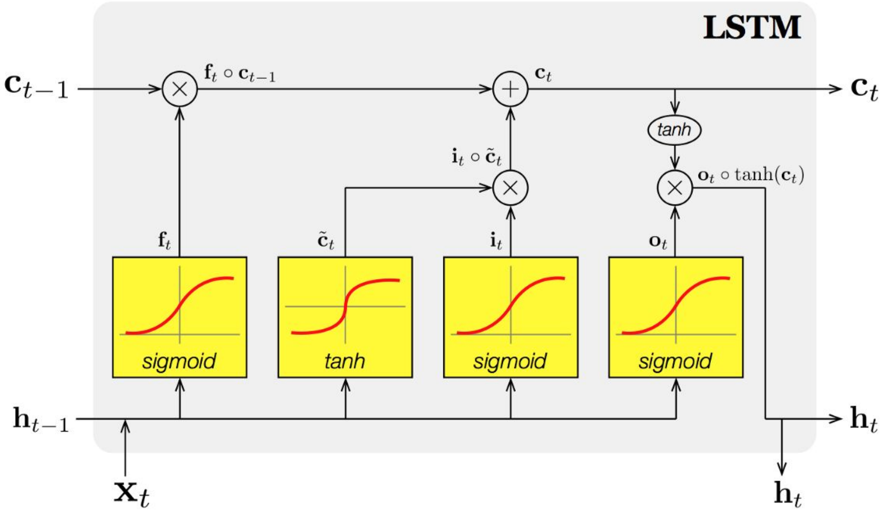
Task

Train model to predict and classify whether tweet is about some disaster or not

Recurrent Neural Networks (RNN)



Long Short Term Memory (LSTM)



Gating variables

$$\mathbf{f}_t = \sigma\left(\mathbf{W}_f[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_t\right)$$
$$\mathbf{i}_t = \sigma\left(\mathbf{W}_i[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_i\right)$$
$$\mathbf{o}_t = \sigma\left(\mathbf{W}_o[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_o\right)$$

Candidate (memory) cell state

$$\tilde{\mathbf{c}}_t = \tanh\left(\mathbf{W}_c[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_c\right)$$

Cell & Hidden state

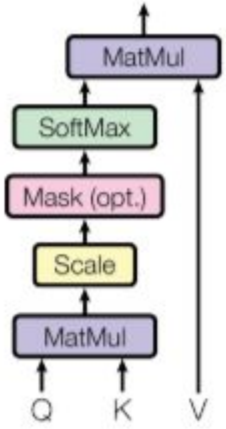
$$\mathbf{c}_t = \mathbf{f}_t \circ \mathbf{c}_{t-1} + \mathbf{i}_t \circ \tilde{\mathbf{c}}_t$$
$$\mathbf{h}_t = \mathbf{o}_t \circ \tanh(\mathbf{c}_t)$$

Components of LSTM:

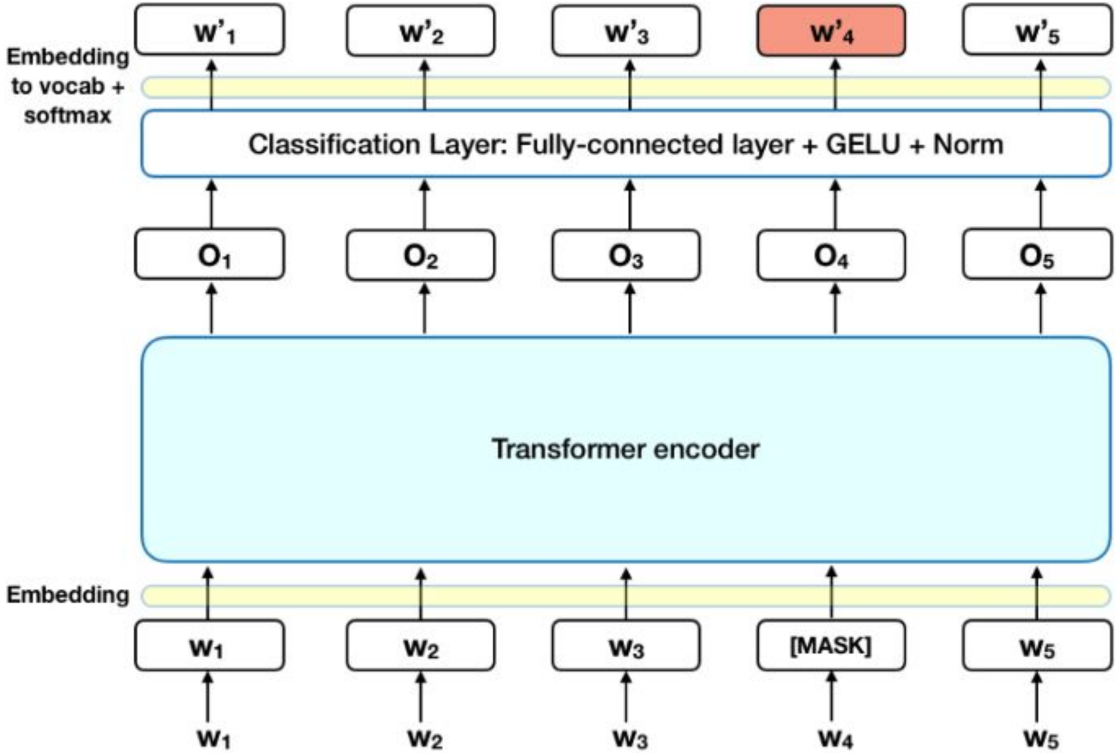
- Forget Gate “f” (a neural network with sigmoid)
- Candidate layer “C"(a NN with Tanh)
- Input Gate “I” (a NN with sigmoid)
- Output Gate “O”(a NN with sigmoid)
- Hidden state “H” (a vector)
- Memory state “C” (a vector)
- Inputs to the LSTM cell at any step are Xt (current input) , Ht-1 (previous hidden state) and Ct-1 (previous memory state).
- Outputs from the LSTM cell are Ht (current hidden state) and Ct (current memory state)

Self - attention

- The first step is multiplying each of the encoder input vectors with three weights matrices (W(Q), W(K), W(V)) that we trained during the training process. This matrix multiplication will give us three vectors for each of the input vectors: the key vector, the query vector, and the value vector.
- The second step in calculating self-attention is to multiply the Query vector of the current input with the key vectors from other inputs.
- In the third step, we will divide the score by the square root of dimensions of the key vector (dk). In the paper the dimension of the key vector is 64, so that will be 8. The reason behind that is if the dot products become large, this causes some self-attention scores to be very small after we apply softmax function in the future.
- In the fourth step, we will apply the softmax function on all self-attention scores we calculated wrt the query word (here first word).
- In the fifth step, we multiply the value vector on the vector we calculated in the previous step.
- In the final step, we sum up the weighted value vectors that we got in the previous step, this will give us the self-attention output for the given word.

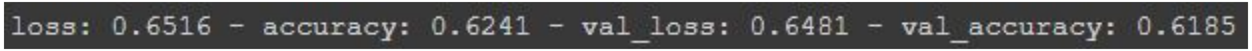


BERT

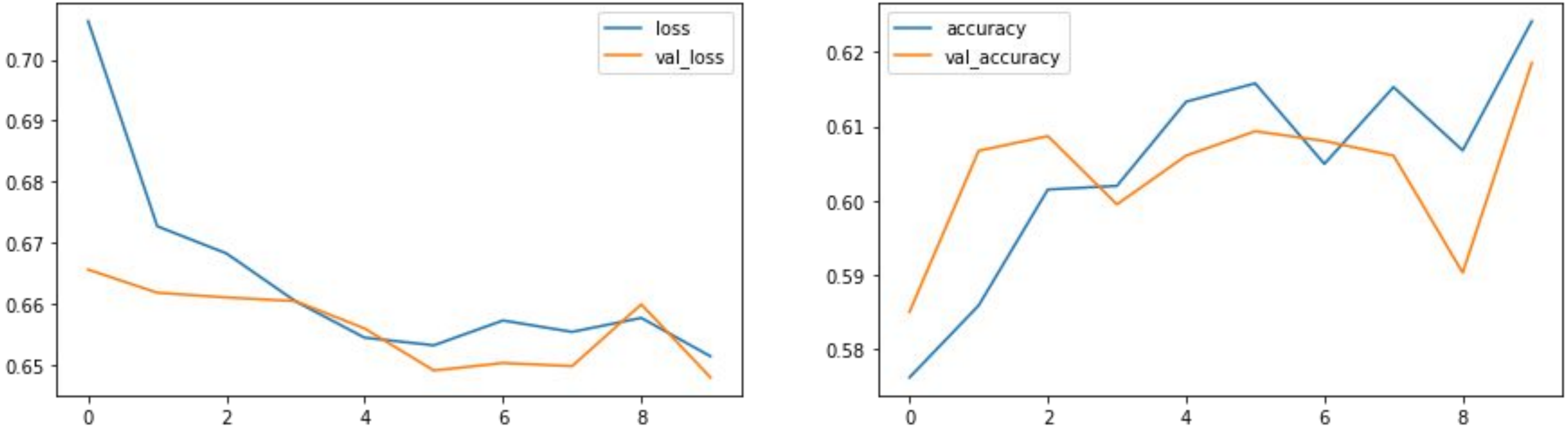

w'1
w'2
w'3
w'4
w'5
Embedding to vocab + softmax
Classification Layer: Fully-connected layer + GELU + Norm
O1
O2
O3
O4
O5
Transformer encoder
Embedding
w1
w2
w3
[MASK]
w5
w1
w2
w3
w4
w5

Results

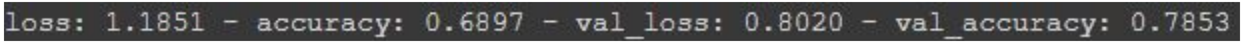
GRU (Average the embeddings for each word of the tweet and learn to classify using a feedforward NN model)



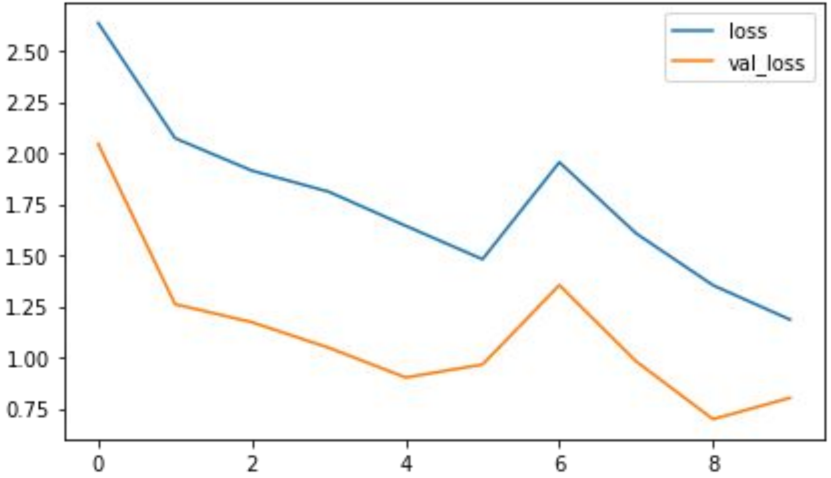
```
loss: 0.6516 - accuracy: 0.6241 - val_loss: 0.6481 - val_accuracy: 0.6185
```

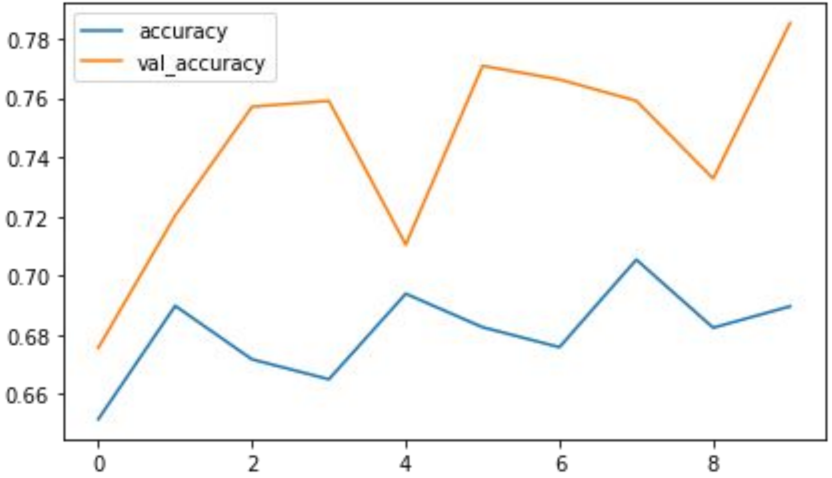


(Concatenated Embeddings Model)

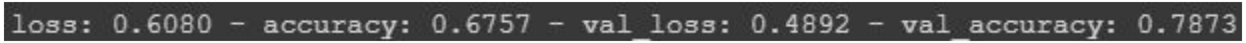


```
loss: 1.1851 - accuracy: 0.6897 - val_loss: 0.8020 - val_accuracy: 0.7853
```

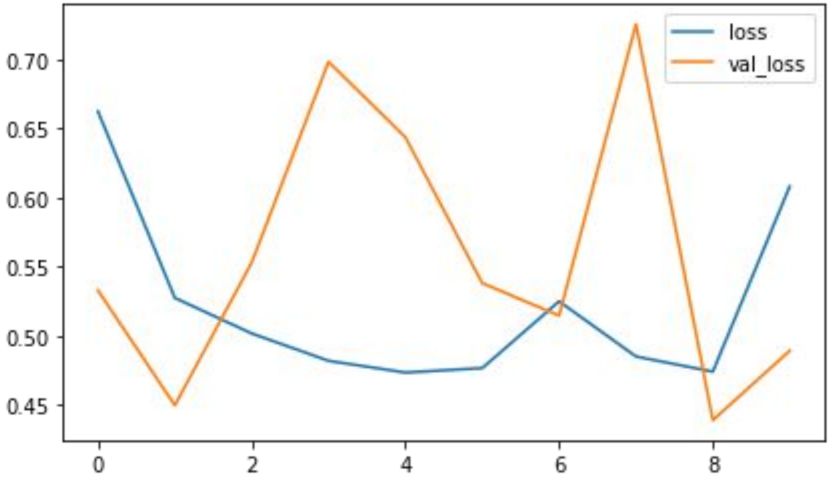


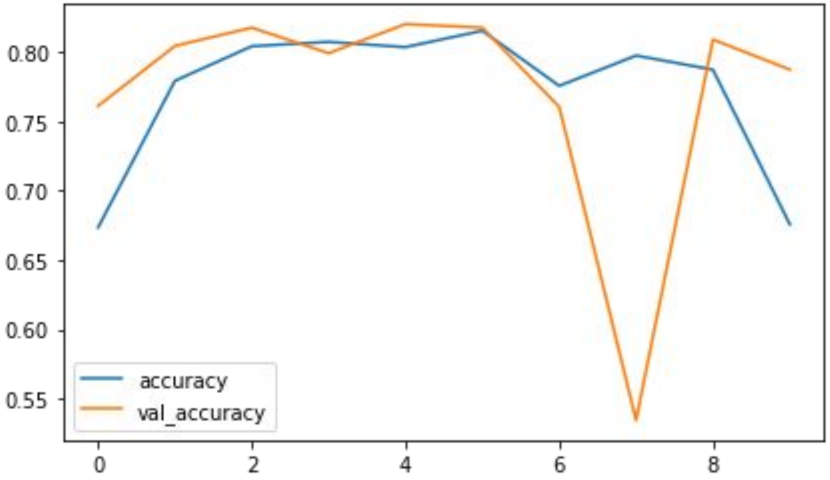


LSTM model (Using only the text feature)

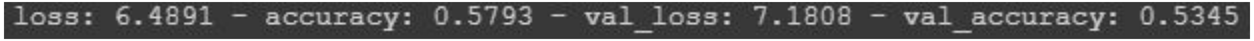


```
loss: 0.6080 - accuracy: 0.6757 - val_loss: 0.4892 - val_accuracy: 0.7873
```

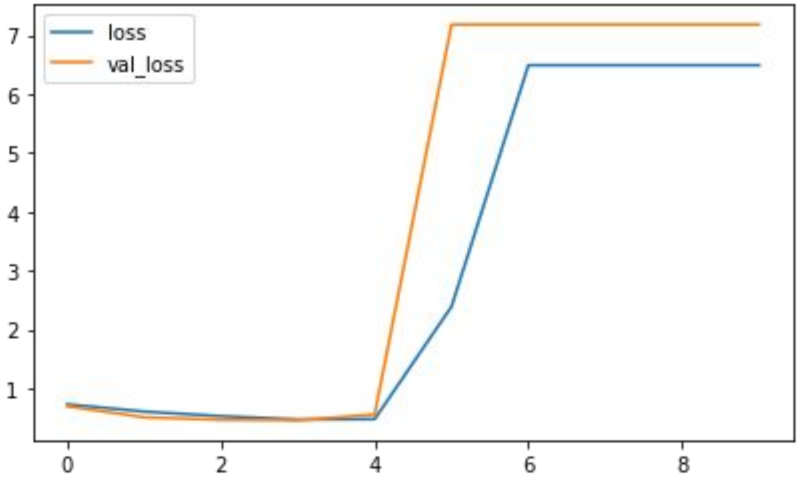


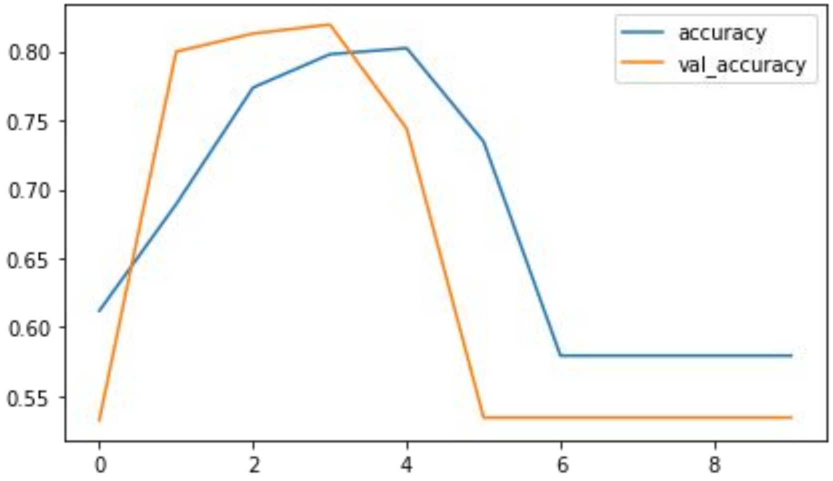


GRU model

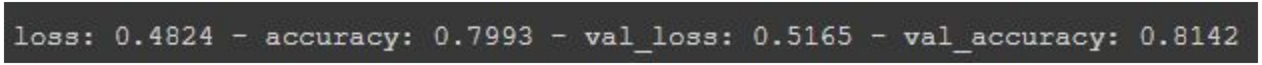


```
loss: 6.4891 - accuracy: 0.5793 - val_loss: 7.1808 - val_accuracy: 0.5345
```

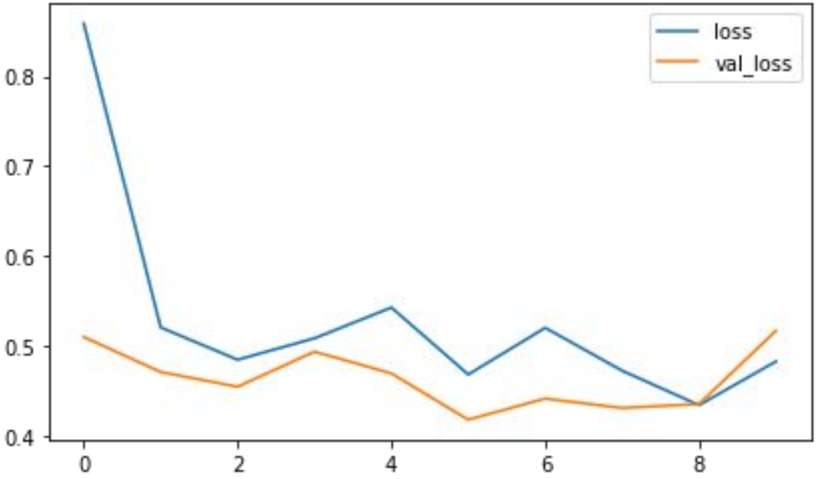


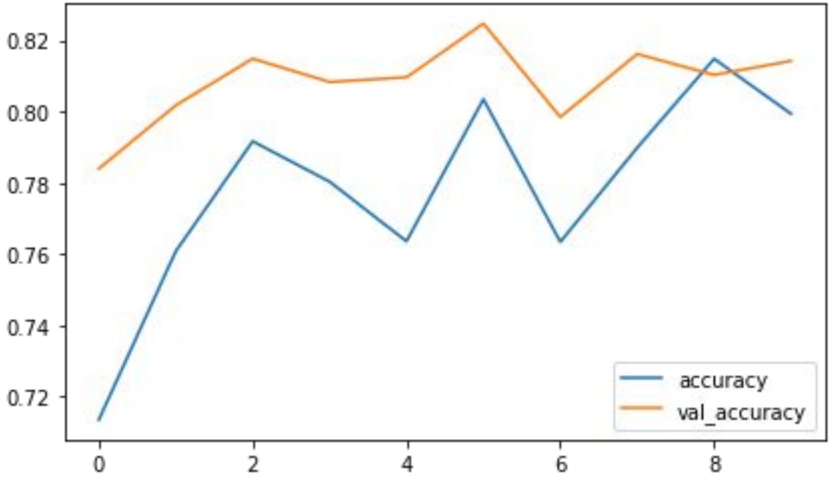


LSTM model with self-attention

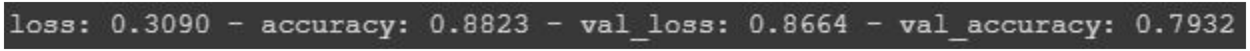


```
loss: 0.4824 - accuracy: 0.7993 - val_loss: 0.5165 - val_accuracy: 0.8142
```

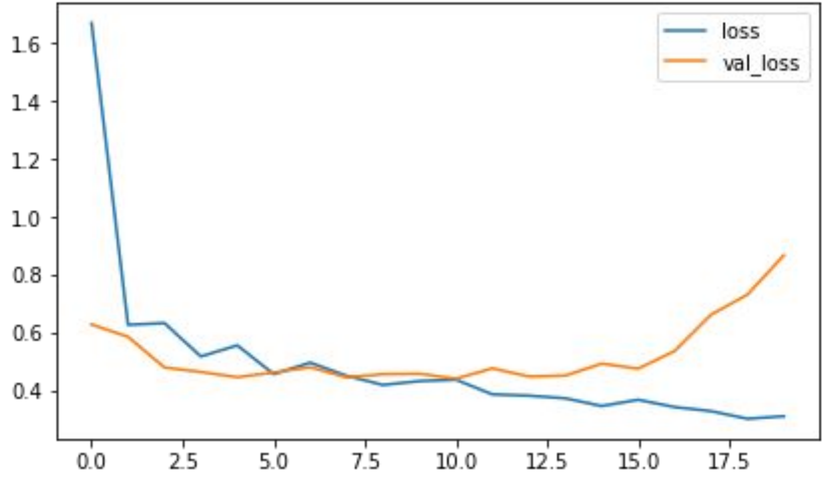


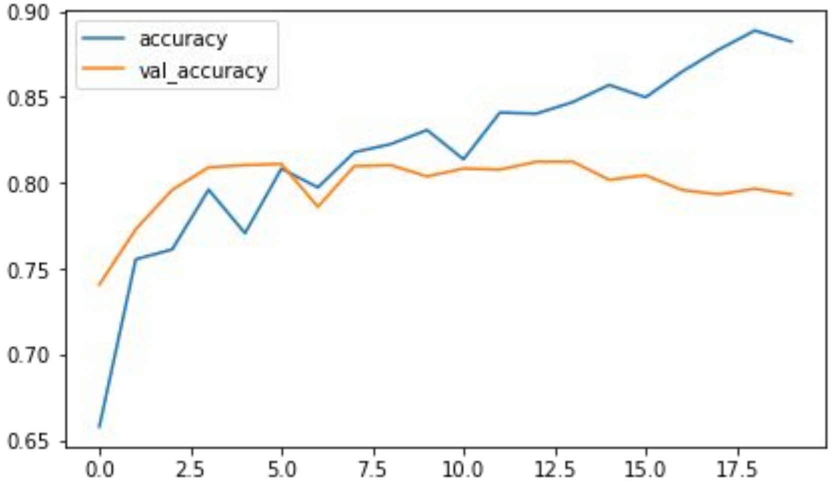


Non-linear LSTM model (with both seq and non-seq inputs)

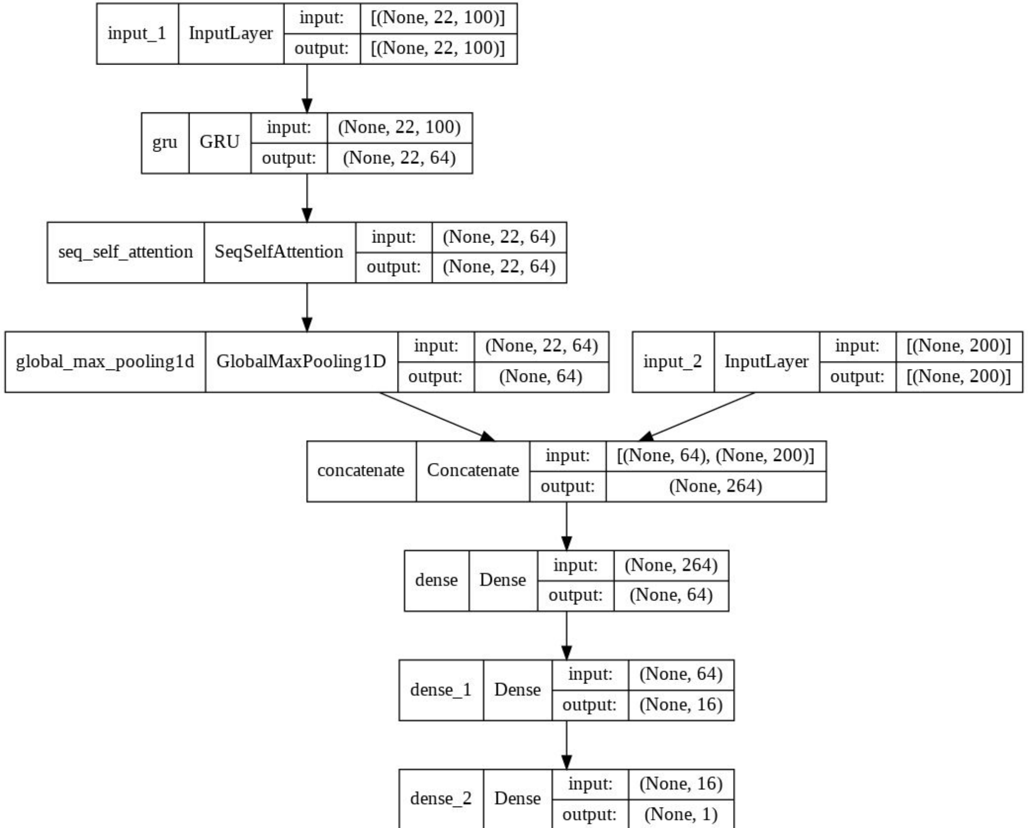


```
loss: 0.3090 - accuracy: 0.8823 - val_loss: 0.8664 - val_accuracy: 0.7932
```





Resulting model



input_1 | InputLayer | input: [(None, 22, 100)] | output: [(None, 22, 100)]

gru | GRU | input: (None, 22, 100) | output: (None, 22, 64)

seq_self_attention | SeqSelfAttention | input: (None, 22, 64) | output: (None, 22, 64)

global_max_pooling1d | GlobalMaxPooling1D | input: (None, 22, 64) | output: (None, 64)

input_2 | InputLayer | input: [(None, 200)] | output: [(None, 200)]

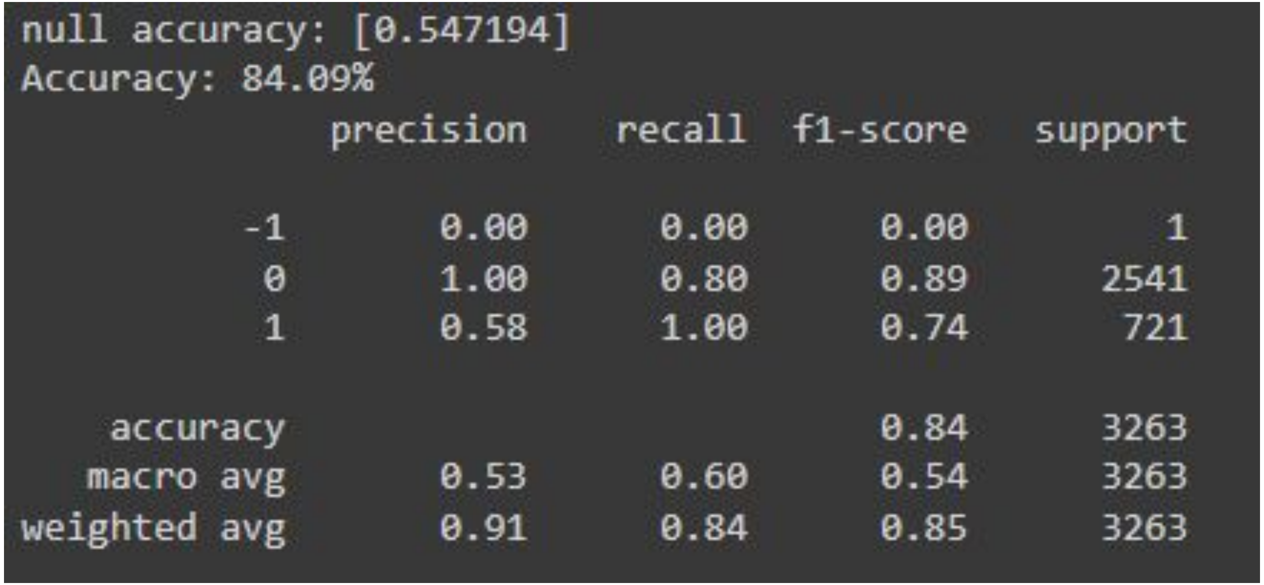
concatenate | Concatenate | input: [(None, 64), (None, 200)] | output: (None, 264)

dense | Dense | input: (None, 264) | output: (None, 64)

dense_1 | Dense | input: (None, 64) | output: (None, 16)

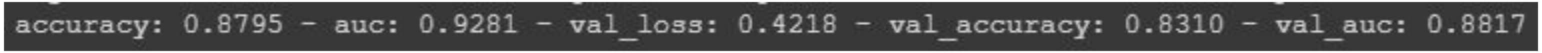
dense_2 | Dense | input: (None, 16) | output: (None, 1)

Prediction accuracy

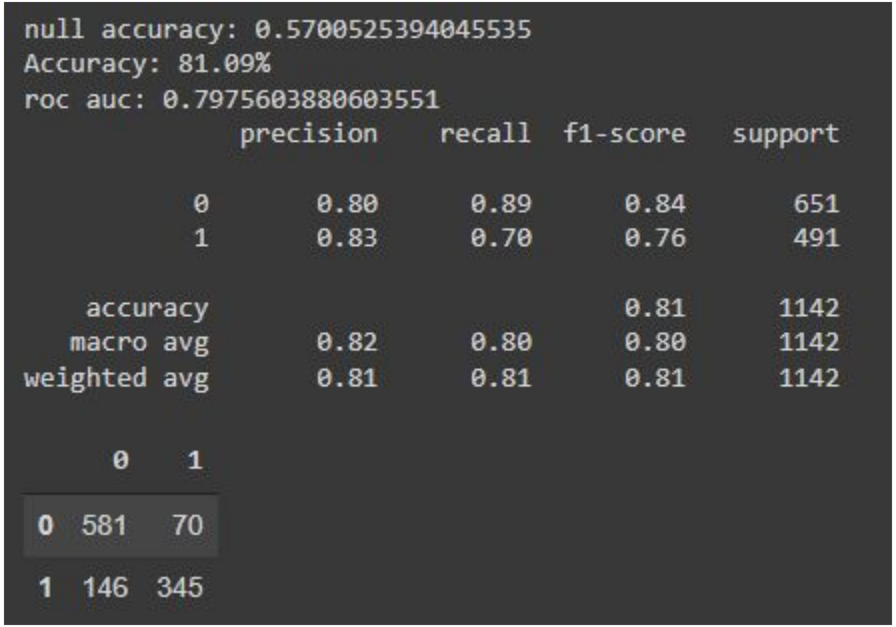


```
null accuracy: [0.547194]
Accuracy: 84.09%
              precision    recall  f1-score   support

          -1       0.00      0.00      0.00         1
           0       1.00      0.80      0.89      2541
           1       0.58      1.00      0.74       721

    accuracy                           0.84      3263
   macro avg       0.53      0.60      0.54      3263
weighted avg       0.91      0.84      0.85      3263
```

BERT



```
accuracy: 0.8795 - auc: 0.9281 - val_loss: 0.4218 - val_accuracy: 0.8310 - val_auc: 0.8817
```



```
null accuracy: 0.5700525394045535
Accuracy: 81.09%
roc auc: 0.7975603880603551
              precision    recall  f1-score   support

           0       0.80      0.89      0.84       651
           1       0.83      0.70      0.76       491

    accuracy                           0.81      1142
   macro avg       0.82      0.80      0.80      1142
weighted avg       0.81      0.81      0.81      1142
```

	0	1
0	581	70
1	146	345

Thanks!