CSE143 - A3 Writeup

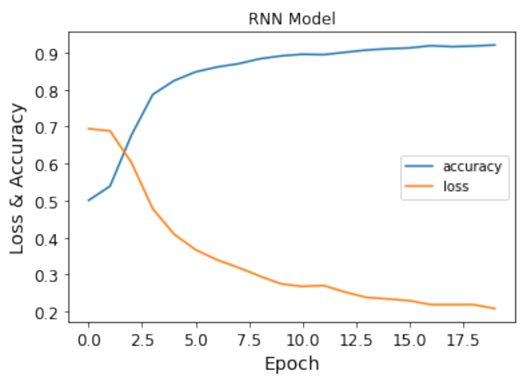
Joey Babcock, Shane Sawyer, Qilong Guo, Joel Indurkar

June 10 2022

1 Programming: Text Classification with RNNs

1.1 Report the accuracy of your model on the training and test sets

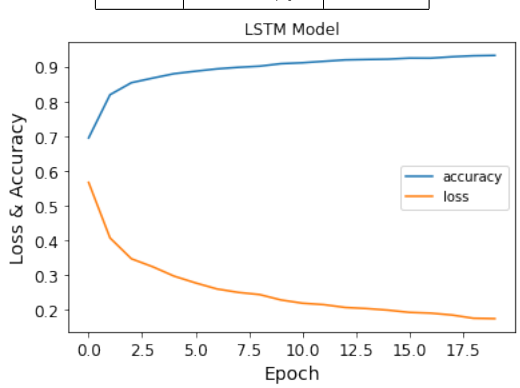
RNN		
	Accuracy	Loss
train	96.57%	0.1117
test	73.61%	0.7601



2 Programming: Text Classification with LSTMs

2.1 Report your LSTM-based model's training, development, and test set accuracy (following the same experimental procedure as in section 1.1).

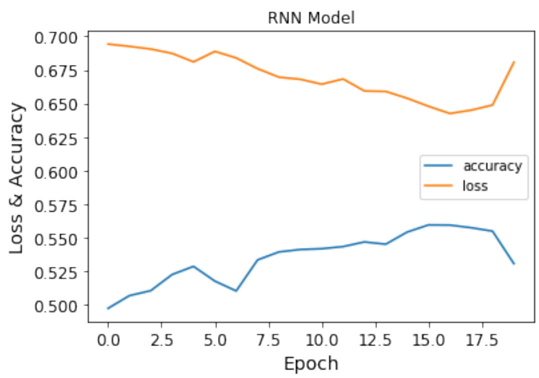
LSTM		
	Accuracy	Loss
train	96.20%	0.1187
test	73.24%	0.7367



2.2 We might expect the LSTM model to have better accuracy than the simple RNN model on longer sequences. Does this hold for your models? Provide evidence and/or examples to demonstrate.

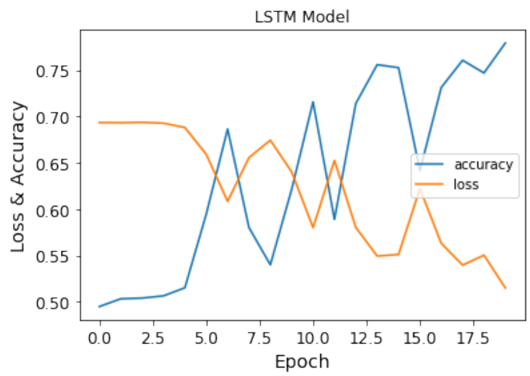
In order to test how the models perform on longer sequences, the models were trained on longer text sequences than before. Additionally, the same hyperparameters were used. While the models could perform much better with some tuning of the hyperparameters, it is still clear which model performs the best on longer sequences.

RNN		
	Accuracy	Loss
train	55.87%	0.6461
test	50.31%	0.7086



The RNN model tends to forget information between each epoch. While training the model, it was able to make some progress in classifying longer sequences but simply fails to make substantial progress as training continued. As seen by the accuracy on the test data, the RNN model fails to classify unseen long sequences any better than random chance.

LSTM		
	Accuracy	Loss
train	79.56%	0.4911
test	75.30%	0.5615



On the other hand, the LSTM model is able to perform much better on longer sequences. Similar to the RNN model, it tends to forget information between training. But, as training continued the model clearly is able to learn from the data. Although the LSTM model is not very accurate on classifying the test data, it is much more accurate the the RNN model.

3 Theory: Deriving the Viterbi Algorithm

Let, for every possible value of y_j, $v_j(y_j) = \max_{y_1,\dots,y_{j-1}} \sum_{i=1}^{j} s(\boldsymbol{x}, i, y_{i-1}, y_i)$. Show that

$$v_j(y_j) = \max_{y_{j-1}} \Big[s(\boldsymbol{x}, j, y_{j-1}, y_j) + v_{j-1}(y_{j-1}) \Big]$$

We see that

$$v_j(y_j) = \max_{y_1,\dots,y_{j-1}} \sum_{i=1}^{j} s(\boldsymbol{x}, i, y_{i-1}, y_i)$$

$$v_j(y_j) = \max_{y_{j-1}} \Big[s(\boldsymbol{x}, j, y_{j-1}, y_j) + \max_{y_1,\dots,y_{j-2}} \sum_{i=1}^{j-1} s(\boldsymbol{x}, i, y_{i-1}, y_i) \Big]$$

$$v_j(y_j) = \max_{y_{j-1}} \Big[s(\boldsymbol{x}, j, y_{j-1}, y_j) + v_{j-1}(y_{j-1}) \Big]$$

Algorithm 11 The Viterbi algorithm. Each $s_m(k, k')$ is a local score for tag $y_m = k$ and $y_{m-1} = k'$.

for $k \in \{0, \dots K\}$ **do**
  $v_1(k) = s_1(k, \Diamond)$
for $m \in \{2, \dots, M\}$ **do**
  for $k \in \{0, \dots, K\}$ **do**
    $v_m(k) = \max_{k'} s_m(k, k') + v_{m-1}(k')$
    $b_m(k) = \operatorname{argmax}_{k'} s_m(k, k') + v_{m-1}(k')$
$y_M = \operatorname{argmax}_k s_{M+1}(\blacklozenge, k) + v_M(k)$
for $m \in \{M-1, \dots 1\}$ **do**
  $y_m = b_m(y_{m+1})$
return $\boldsymbol{y}_{1:M}$

Let $n = M$ and $m = K$. The first for loop take $O(m)$ time. Inside the nested for loop, the max and argmax operations take a total of $O(m)$ time inside a loop that iterates $O(m)$ times. These operations are nested inside another for loop that takes $O(n)$ time, so the nested for loops take $O(nm^2)$ time. Lastly, the final argmax and for loop take $O(m)$ and $O(n)$ time respectively. Thus, the final runtime of the Viterbi algorithm is $O(nm^2)$.

3.1 Coding the Viterbi Algorithm

Using "model.simple"

	Precision	Recall	F_1
dev	72.26%	39.45%	51.03
test	66.90%	32.85%	44.06

After training a model using stochastic gradient descent for 10 epochs, "model.iter7" was chosen because it has the best performance on the dev data.

	Precision	Recall	F_1
dev	84.32%	72.18%	77.78
test	77.40%	59.83%	67.49

The weights can be seen in VITERBI/data/model.sorted.txt