Assignment 4: Model Comparison for Text Classification
CS 584: Natural Language Processing
Spring 2021
Sarita Hedaya

Summary of method

(a) Preprocess the data

To get the data ready for training I performed the following steps:

- Removed book index and titles such as “Chapter I” from the top of each document
- Removed punctuations and end line markers from text.
- Tokenized each paragraph into a list of words
- Removed stop words as set by nltk package
- Embedded each word into a 50-dimensional vector using pre-trained GloVe word embeddings. These embeddings were trained using 6 billion words from Wikipedia and Gigaword 5. (https://nlp.stanford.edu/projects/glove/)
- Padded / truncated sequences to a length of 500 words, therefore each training example is 500,50.
- Removed very short samples, used this as a hyperparameter. Tested removing and keeping the very short samples.
- Used train_test_split from sklearn to split the data. Since there is class imbalance and the law of large numbers is not in our favor (dataset is only ~20,000 samples) it was necessary to ensure that the class distribution was the same for each of the training, validation, and test sets. For that, I used the `stratify` parameter in `train_test_split`.

(b) Build models

I used Keras Sequential to define both LSTM models and the CNN model. I used the appropriate loss function and output layer for each model.

(c) Backpropagation

Backpropagation is taken care of by the compiler using the selected optimizer and categorical cross entropy loss for part 1, binary cross entropy for part 2.

(d) Training the models

I used an iterative process to fine tune each of the three models:

1. The LSTM model with last hidden state as prediction
2. The LSTM model with all hidden states considered for prediction
3. The CNN model

I started with default values for all hyperparameters and the Adam optimizer. Documenting every run, I systematically changed the appropriate hyperparameters to maximize performance. See the hyperparameter search space in the next section.

(e) Compare models

The following tables document the improvement runs that I implemented (I recorded here only those that improved the val_loss, I ran more than 100 runs).
The results are recorded as follows:

- **Val_loss**: the lowest validation loss seen, not necessarily at the latest epoch.
- **Train_loss**: is the training loss at the epoch with lowest validation loss, this is useful when determining if the model is overfitting. If the train_loss and validation_loss are too far apart, the model is likely overfitting and dropout or model simplification should be considered.
- **train _acc:** accuracy on the training set at the epoch of lowest validation loss.
- **Val_acc:** accuracy on the validation set at the epoch of lowes validation loss.

Hyperparameter search space:

Class weights

All models with an asterisk in the tables represent models that were trained with class weights.

The balanced model for calculating the class weights equalizes the loss caused by each class by normalizing w.r.t. the presence of each class in the data:

```
wj=n_samples / (n_classes * n_samplesj)
```

W_0 = weight for “f.d.” = 20098 / (3*6055) = 1.106
W_1 = weight for “a.c.d” = 20098 / (3*2548) = 2.629
W_2 = weight for “j.a.” = 20098 / (3*11495) = 0.5828

Using class weights to increase the recall of the under-represented class caused instability in the model, since the loss was now more volatile to the presence or absence of class 1.

I found that in practice, setting the class weight for class 1 “a.c.d.” to 2.0 gave the best balance between precision and recall.

Optimizers

Adam, RMSProp, SGD. Learning rates: default, 0.001, 0.0001, 0.01

Loss functions

For part 1, since it's a classification problem with more than two classes I used categorical cross entropy loss and a 3 unit softmax activation function on the output layer. For part 2, I used binary cross entropy loss, and a sigmoid activation function on the last layer.

Number of LSTM layer units

5, 10, 25, 50, 100, 200

Number of Dense layer units

5, 10, 25, 50, 100, 200

Dropout regularization rate

0, 0.1, 0.2, 0.3, 0.4

Conv1D layer filters

16, 32, 64, 128

Conv1D layer kernel size

2, 3, 5, 7

Part 1: Document Classification

Results:

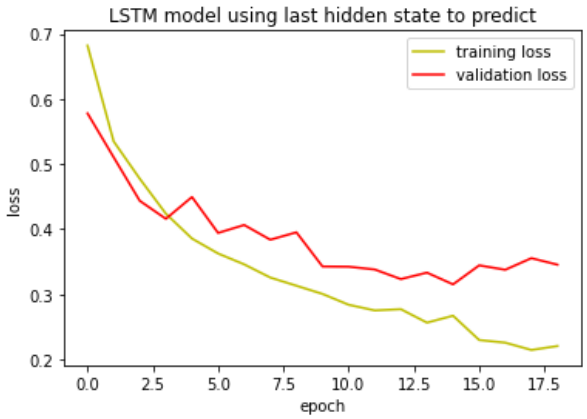
LSTM model with last hidden state as prediction

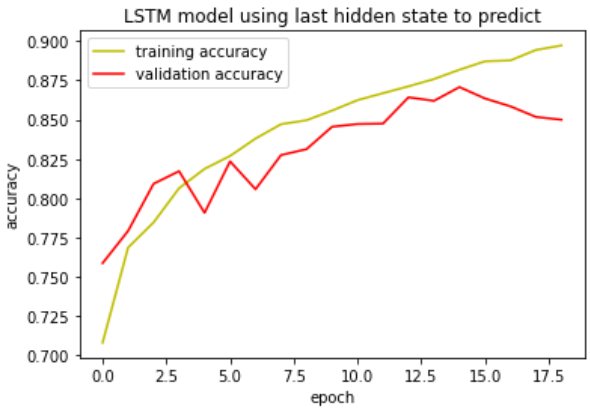
Run #	Lstm nodes	Dense layer nodes	Dropout	train_loss	val_loss	train_acc	val_acc
1	100	100	-	0.2894	0.3333	0.8804	0.8600
2	100	100	0.3	0.2977	0.3374	0.8757	0.8568
3	100	64	0.4	0.3102	0.3390	0.8711	0.8543
4	150	100	0.4	0.3075	0.3337	0.8714	0.8578
5 (*)	100	100	0.3,0.4	0.1520	0.3143	0.8917	0.8710
6 (*)	100	100	0,0.3	0.2455	0.3069	0.9018	0.8740
7 (*)	**100**	**100**	**0,0.4**	**0.2376**	**0.3040**	**0.9020**	**0.8775**
8	64	64	0, 0.3	0.2157	0.2911	0.9103	0.8835

(*) On the 5th model I introduced class weights since there is a class imbalance problem. As you can see, the training loss finally started decreasing below 0.28, but the validation loss did not catch up. I need to take measures to combat overfitting but there is definitely an improvement.

Complete classification report for top LSTM last hidden state model

	precision	recall	f1-score	support
0	0.97	0.94	0.95	592
1	0.58	0.70	0.63	260
2	0.92	0.89	0.90	1148
accuracy			0.88	2000
macro avg	0.82	0.84	0.83	2000
weighted avg	0.89	0.88	0.88	2000


LSTM model using last hidden state to predict
training loss
validation loss
loss
epoch
0.7
0.6
0.5
0.4
0.3
0.2
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

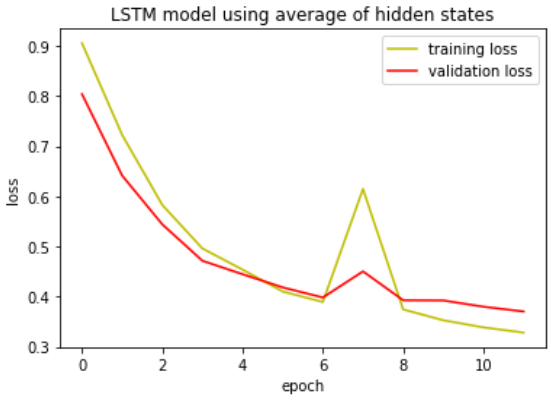

LSTM model using last hidden state to predict
training accuracy
validation accuracy
accuracy
epoch
0.900
0.875
0.850
0.825
0.800
0.775
0.750
0.725
0.700
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

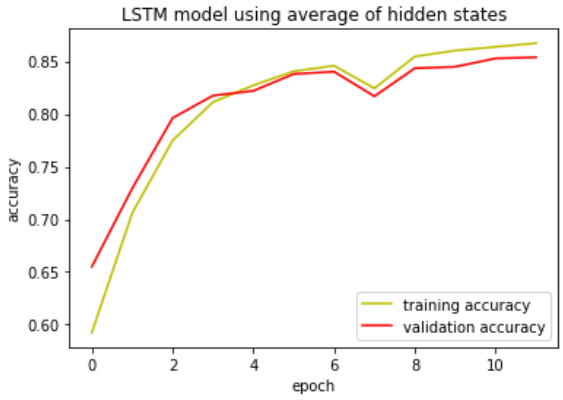
LSTM model with average of all hidden states for prediction

Run #	Lstm nodes	Dense layer nodes	Dropout	train_loss	val_loss	train_acc	val_acc
1	100	100	-	0.3015	0.3722	0.8772	0.8530
2	100	100	0.3	0.3324	0.3693	0.8648	0.8502
3 (*)	**100**	**64**	**0.3**	**0.1913**	**0.3406**	**0.8588**	**0.8568**
4 (*)	64	64	0.3	0.2503	0.4203	0.7965	0.7726
5 (*)	200	150	0.2	0.3422	0.3537	0.8271	0.8495

Complete classification report for top LSTM all hidden states model

	precision	recall	f1-score	support
0	0.96	0.93	0.95	592
1	0.69	0.58	0.63	260
2	0.89	0.94	0.92	1148
accuracy			0.89	2000
macro avg	0.85	0.82	0.83	2000
weighted avg	0.89	0.89	0.89	2000





CNN model

Run #	Filters, kernel size	Dense layer nodes	Dropout	train_loss	val_loss	train_ acc	val_acc	Precision class 1	recall 1	F1 score
1	64,3	16	-	0.2003	0.3888	0.918	0.8558			
2	64,3	16	0.2	0.2506	0.3656	0.896	0.8570			
3 (*) (+)	32,3	16	0.2	0.3835	0.4233	0.822	0.8230	0.45	0.83	0.58
4 (*) (+)	64,3	16	0.3,0.3	0.2843	0.3900	0.868	0.8310	0.45	0.85	0.59
5 (*) (^)	**64,3**	**16**	**0.3,0.3**	**0.2638**	**0.3768**	**0.872**	**0.8425**	**0.57**	**0.75**	**0.65**

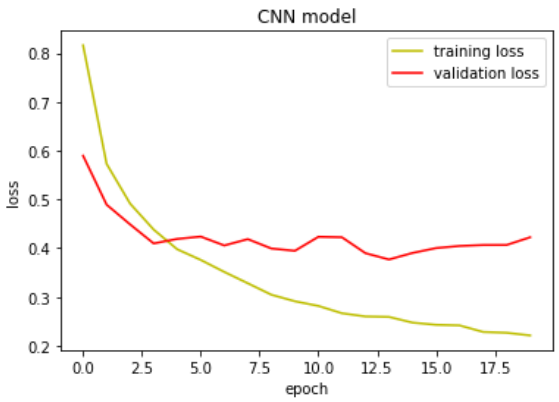
(+) even though the CNN model #3 had lower validation accuracy, it had significantly higher recall of the underrepresented class. Nonetheless the precision was very low, so it was as if the classifier was scared to classify class1 out of fear of severe punishment (class 1 is weighed 6x higher than class 2). Therefore, in model 5 (^) I lowered the class weight of class 1 from 2.63 to 2 to see if it would increase precision on class 1.

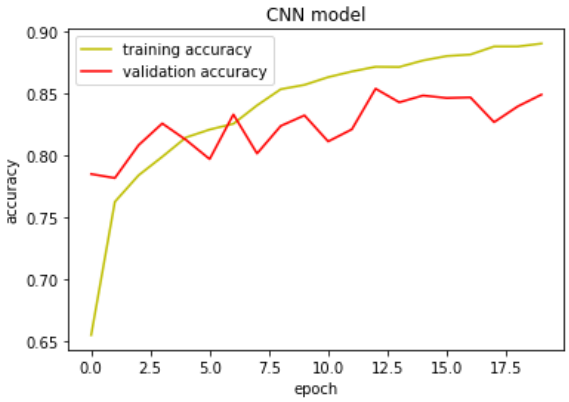
Complete classification report for top CNN model

```
                precision    recall  f1-score   support

           0       0.93      0.93      0.93       592
           1       0.57      0.75      0.65       260
           2       0.93      0.86      0.89      1148

    accuracy                           0.87      2000
   macro avg       0.81      0.85      0.82      2000
weighted avg       0.88      0.87      0.87      2000
```


CNN model
training loss
validation loss
loss
epoch
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


CNN model
training accuracy
validation accuracy
accuracy
epoch
0.90
0.85
0.80
0.75
0.70
0.65
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

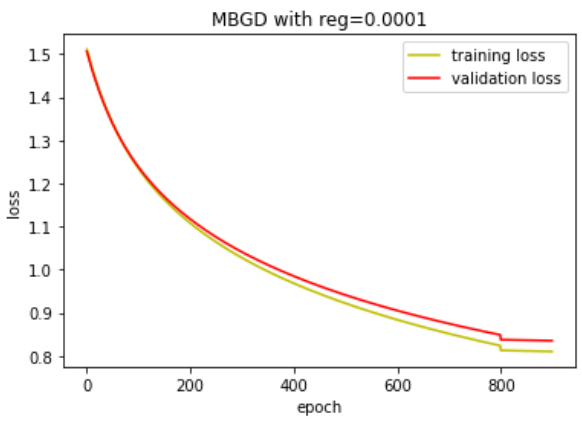
Logistic regression model

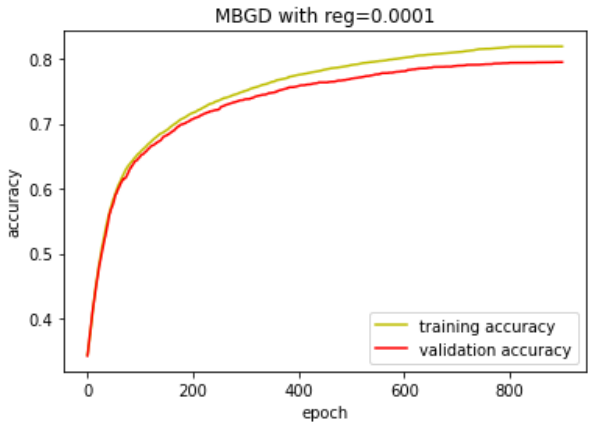
Using the code I wrote for assignment 1, I trained a logistic regression model optimized using mini-batch stochastic gradient descent. I fine tuned the model by experimenting with different regularization parameters. The table below summarizes my findings:

Run #	Regularization	train_loss	val_loss	train_acc	val_acc
1	0.01	0.9231	0.9503	0.7203	0.7085
2	0.001	0.8223	0.8457	0.8183	0.7938
3	**0.0001**	**0.8104**	**0.8354**	**0.8189**	**0.7948**

Complete classification report for Logistic regression model

	precision	recall	f1-score	support
0	0.75	0.78	0.76	592
1	0.74	0.25	0.37	260
2	0.81	0.91	0.86	1148
accuracy			0.79	2000
macro avg	0.77	0.65	0.67	2000
weighted avg	0.78	0.79	0.77	2000





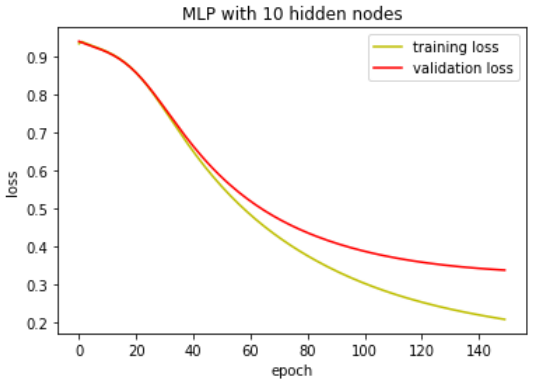
Multi Layer Perceptron

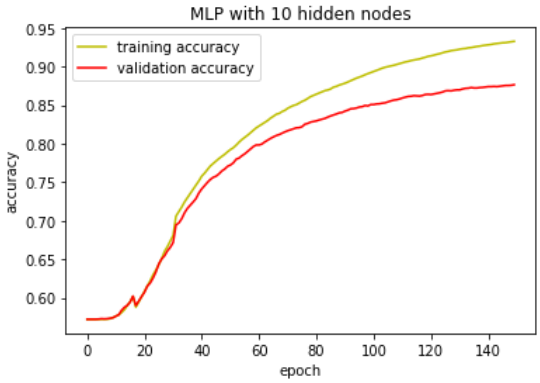
Using the code I wrote for assignment 1, I trained a multilayer perceptron model optimized using mini-batch stochastic gradient descent. I fine tuned the model by experimenting with different number of hidden nodes on the hidden layer. The table below summarizes my findings:

Run #	Nodes on hidden layer	filters	kernel_size	train_loss	val_loss	train_acc	val_acc
1	25	64	5	0.2155	0.3479	0.9326	0.8755
2	100	32	5	0.222	0.3508	0.9352	0.8685
3	50	64	3	0.2139	0.3564	0.9321	0.8680
4	10	64	3	0.2095	0.3392	0.9328	0.8765
5	5	64	3	0.1663	0.336	0.9514	0.8788
6(*)	**5**	**64**	**3**	**3285**	**3284**	**0.9421**	**0.8833**

(*) used L2 regularization which made the loss larger but the accuracy improved.

Complete classification report for top MLP model





	precision	recall	f1-score	support
0	0.88	0.84	0.86	592
1	0.76	0.61	0.68	260
2	0.90	0.96	0.93	1148

accuracy			0.88	2000
macro avg	0.85	0.80	0.82	2000
weighted avg	0.87	0.88	0.87	2000

Part 2: Sentiment Analysis

LSTM model with last hidden layer to predict

Run #	Lstm nodes	Dense layer nodes	Dropout	train_loss	val_loss	train_acc	val_acc
1	5	10	0.2,0.3	0.5491	0.5758	0.7358	0.7100
2	10	10	0.4,0.4	0.5248	0.5871	0.7582	0.7025
3	20	4	0,0.4	0.5135	0.5623	0.7574	0.7325
4(^)	10	10	0,0.5	0.5169	0.5804	0.7743	0.7120
5	20	20	0,0.4	0.5146	0.5931	0.7489	0.7200

(^) The models that have this symbol were trained only on those samples that were longer than 10 words.

Test set accuracy = 0.71

LSTM model with average of hidden layers to predict

Run #	Lstm nodes	Dense layer nodes	Dropout	train_loss	val_loss	train_acc	val_acc
1	40	64,16	0.3,0.2	0.6417	0.6316	0.7764	0.7125
2	100	64	0.3	0.7610	0.6598	0.7083	0.6275
3	100	64	0.4	0.6801	0.6681	0.7584	0.6875

(^) The models that have this symbol were trained only on those samples that were longer than 10 words.

Test set accuracy = 0.69

CNN model

Run #	filters	kernel_size	nodes on dense	dropout	train_loss	val_loss	train_acc	val_acc
1	128	3	32	0.4	0.3733	0.6338	0.8775	0.6225
2	64	3	32	0.4	0.3671	0.6399	0.8660	0.7100

None of the other parameters provided higher val_acc, therefore I did not include them in the above list.
I also tried to add a second Dense layer and Dropout layer before the output layer. This just caused additional overfitting.

Test set accuracy = 0.71