Lab 5 - LSTM for Text Classification

February 18

In Lab 3, you trained word embeddings using unsupervised learning cast as a binary classification problem (i.e. the skip-gram model with negative sampling). Last week, you used one-hot encoding, randomly initialized embeddings and pretrained GLoVe embeddings to each train a model for classifying movie reviews. This week, you will build an LSTM model to classify the same reviews and after training, extract the trained word embeddings.

1. Getting the Dataset

You will be using the imdb dataset from last week.

```
>>> import keras
>>> imdb = keras.datasets.imdb

# loading only the top 10000 words in the vocabulary like you did last week.
>>> VOCAB_SIZE = 10000
# we reserve indices 0 for '<PAD>', 1 for '<START>' and 2 for '<UNK>' i.e. words not in vocabulary
>>> INDEX_FROM = 3 # EDITED
>>> (train_data,train_labels),(test_data,test_labels) = imdb.load_data(num_words=VOCAB_SIZE,
index_from=INDEX_FROM)
```

Sanity check:

```
# Each instance in the training data is a list of word indices representing the  words in the review
>>> print('Sample review:', train_data[0])
# and each label is 1 if that review is positive, else 0
>>> print('\n Sample label:', test_labels[1])
```

```
Sample review: [1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 458, 4468,
66, 3941, 4, 173, 36, 256, 5, 25, 100, 43, 838, 112, 50, 670, 2, 9, 35,
480, 284, 5, 150, 4, 172, 112, 167, 2, 336, 385, 39, 4, 172, 4536, 1111,
17, 546, 38, 13, 447, 4, 192, 50, 16, 6, 147, 2025, 19, 14, 22, 4, 1920,
4613, 469, 4, 22, 71, 87, 12, 16, 43, 530, 38, 76, 15, 13, 1247, 4, 22,
17, 515, 17, 12, 16, 626, 18, 2, 5, 62, 386, 12, 8, 316, 8, 106, 5, 4,
2223, 5244, 16, 480, 66, 3785, 33, 4, 130, 12, 16, 38, 619, 5, 25, 124,
51, 36, 135, 48, 25, 1415, 33, 6, 22, 12, 215, 28, 77, 52, 5, 14, 407, 16,
82, 2, 8, 4, 107, 117, 5952, 15, 256, 4, 2, 7, 3766, 5, 723, 36, 71, 43,
```

```
530, 476, 26, 400, 317, 46, 7, 4, 2, 1029, 13, 104, 88, 4, 381, 15, 297,
98, 32, 2071, 56, 26, 141, 6, 194, 7486, 18, 4, 226, 22, 21, 134, 476, 26,
480, 5, 144, 30, 5535, 18, 51, 36, 28, 224, 92, 25, 104, 4, 226, 65, 16,
38, 1334, 88, 12, 16, 283, 5, 16, 4472, 113, 103, 32, 15, 16, 5345, 19,
178, 32]

Sample label: 1
```

2. Readying the Inputs for the LSTM

The movie reviews are of different lengths. Last week, the vectors for all the words in a given review were averaged so that each input to the neural network was an EMBED_SIZE vector. The LSTM can handle sequence data by taking input in the sequence in turn i.e. taking one review at a time. The standard is to feed it same sized inputs i.e. reviews of the same length. This is achieved by padding or truncating each review as needed to MAXIMUM_LENGTH. Keras provides a way to do this.

Using pad_sequences from keras.preprocessing.sequence, pad the train and test data.
>>> from keras.preprocessing.sequence import pad_sequences
>>> MAXIMUM_LENGTH = 500

Sanity Check
>>> print('Length of sample train_data before preprocessing:', len(train_data[0]))
>>> print('Length of sample train_data after preprocessing:', len(preprocessed_train_data[0]))

```
Length of sample train_data before preprocessing: 218
Length of sample train_data after preprocessing: 500
```

3. Building the Model

In this section, you will use the keras Sequential or Model API to build a model.

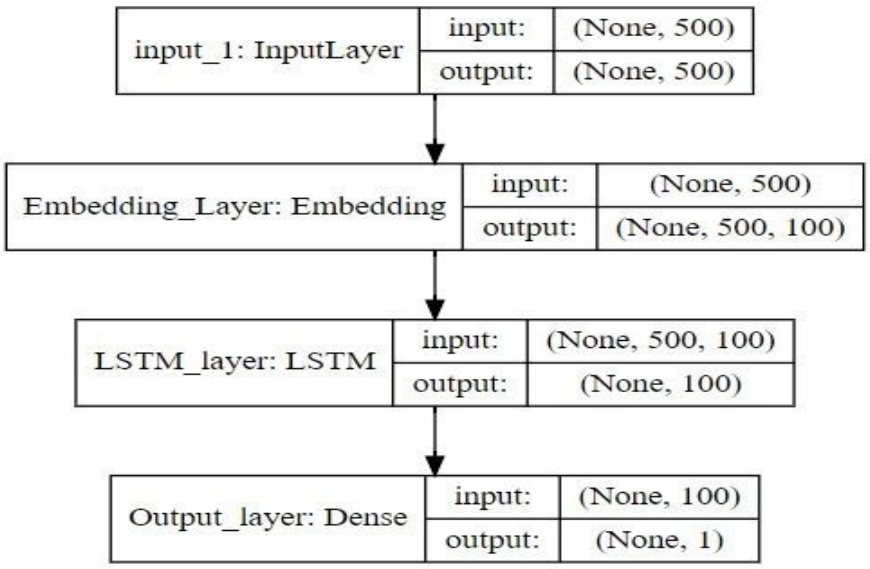
- The first "layer" in the architecture is a randomly initialized embedding layer, which turns each word in the review into an EMBED_SIZE dimension vector. Use EMBED_SIZE = 100.
- The second keras layer is an LSTM layer with 100 units and the **sigmoid (or tanh) activation function** https://keras.io/layers/recurrent/.
- The final keras layer is an output layer. Can you figure out what activation function will be appropriate for this?
- The model should be compiled with a 'binary_crossentropy' loss function, an 'adam' optimizer and since we care about the accuracy, set metrics=['accuracy'].

Sanity Check:

# view model summary
>>> print(model.summary())

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 500)               0
_________________________________________________________________
Embedding_Layer (Embedding)  (None, 500, 100)          1000000
_________________________________________________________________
LSTM_layer (LSTM)            (None, 100)               80400
_________________________________________________________________
Output_layer (Dense)         (None, 1)                 101
=================================================================
Total params: 1,080,501
Trainable params: 1,080,501
Non-trainable params: 0
_________________________________________________________________
```

# Visualizing the model Structure
>>> from keras.utils import plot_model
>>> from IPython.display import SVG
>>> from keras.utils import vis_utils
>>> SVG(vis_utils.model_to_dot(
model, show_shapes=True, show_layer_names=True).create(prog='dot', format='svg'))



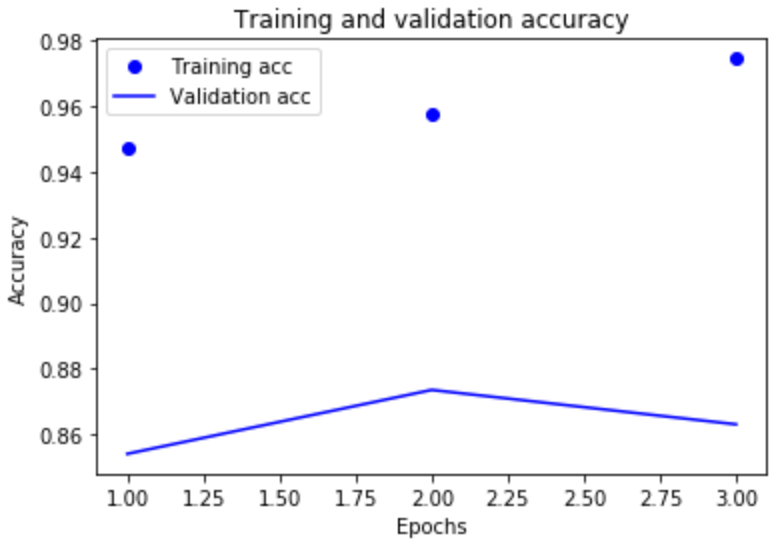
The layer names are optional and depending on what API you used and what parameters you specified, the Embedding layer dimensions should either be (None, None, 100) or (None, 500, 100). **If you use the Sequential API, you will not see the 'Input layer'.**

4. Training the Model

First, split the training data into training and validation data. You can use 2000 samples for validation and the rest for training.

Then, using this split dataset and the model.fit() function, train the model for 3 epochs with a batch size of 100. Using model.fit().history, plot the training and validation accuracy across the epochs. Your output should look like this.

```
Train on 23000 samples, validate on 2000 samples
Epoch 1/3
23000/23000 [==============================] - 218s 9ms/step - loss: 0.1427 - a
cc: 0.9473 - val_loss: 0.3602 - val_acc: 0.8540
Epoch 2/3
23000/23000 [==============================] - 215s 9ms/step - loss: 0.1188 - a
cc: 0.9577 - val_loss: 0.3555 - val_acc: 0.8735
Epoch 3/3
23000/23000 [==============================] - 218s 9ms/step - loss: 0.0749 - a
cc: 0.9747 - val_loss: 0.4016 - val_acc: 0.8630
```



Based on your plot, what do you think the optimal stopping point for the model should have been?

5. Evaluating the Model on the Test Data

Evaluate the model on the preprocessed test data. Print the test loss and accuracy using the line of code below.

Sanity Check:

```
>>> print('test_loss:', results[0], 'test_accuracy:', results[1])
25000/25000 [==============================] - 272s 11ms/step
test_loss: 0.4406837784147263 test_accuracy: 0.8486
```

6. Extracting the Word Embeddings

Extract the word embeddings from the embedding layer model using model.get_layer(layer_name).get_weights()[0] or model.layers[layer_number].get_weights()[0] where the layer number depends on the order in which the layers were added to the model. You can get details about the order from model.summary() or by calling model.layers.

Sanity Check:

```
>>> print('Shape of word_embeddings:', word_embeddings.shape)
Shape of word_embeddings: (10000, 100)
```

7. Visualizing the Reviews

In this section, we will view the effects of preprocessing on the dataset. Keras already preprocessed the dataset. In this preprocessed dataset, all the words have already been mapped to indices and were assigned these numbers in order of their frequency such that index 2 is the 2nd most frequent word in the dataset, and so on. **Create the word2idx dictionary to following lines of code**.

```
>>> word2idx = imdb.get_word_index()
# 9998 since we only used top 1000 words of including  '<PAD>', '<START>' and '<UNK>'
>>> word2idx = {k:(v+INDEX_FROM) for k,v in word2idx.items() if v < 9998}
>>> word2idx["<PAD>"] = 0
>>> word2idx["<START>"] = 1
>>> word2idx["<UNK>"] = 2
```

In the next line, create the idx2word map for all the words in the dataset

View a sample review text using the lines of code below:

```
>>> print(' '.join(idx2word[idx] for idx in train_data[0]))
```

```
<START> this film was just brilliant casting location scenery story direction e
veryone's really suited the part they played and you could just imagine being t
here robert <UNK> is an amazing actor and now the same being director <UNK> fat
her came from the same scottish island as myself so i loved the fact there was
a real connection with this film the witty remarks throughout the film were gre
at it was just brilliant so much that i bought the film as soon as it was relea
sed for <UNK> and would recommend it to everyone to watch and the fly fishing w
as amazing really cried at the end it was so sad and you know what they say if
you cry at a film it must have been good and this definitely was also <UNK> to
the two little boy's that played the <UNK> of norman and paul they were just br
illiant children are often left out of the <UNK> list i think because the stars
that play them all grown up are such a big profile for the whole film but these
children are amazing and should be praised for what they have done don't you th
ink the whole story was so lovely because it was true and was someone's life af
ter all that was shared with us all
```

# view the corresponding data
>>> print(train_data[0])

```
[1, 13, 21, 15, 42, 529, 972, 1621, 1384, 64, 457, 4467, 65, 3940, 3, 172, 35,
255, 4, 24, 99, 42, 837, 111, 49, 669, 2, 8, 34, 479, 283, 4, 149, 3, 171, 111,
166, 2, 335, 384, 38, 3, 171, 4535, 1110, 16, 545, 37, 12, 446, 3, 191, 49, 15,
5, 146, 2024, 18, 13, 21, 3, 1919, 4612, 468, 3, 21, 70, 86, 11, 15, 42, 529, 3
7, 75, 14, 12, 1246, 3, 21, 16, 514, 16, 11, 15, 625, 17, 2, 4, 61, 385, 11, 7,
315, 7, 105, 4, 3, 2222, 5243, 15, 479, 65, 3784, 32, 3, 129, 11, 15, 37, 618,
4, 24, 123, 50, 35, 134, 47, 24, 1414, 32, 5, 21, 11, 214, 27, 76, 51, 4, 13, 4
06, 15, 81, 2, 7, 3, 106, 116, 5951, 14, 255, 3, 2, 6, 3765, 4, 722, 35, 70, 4
2, 529, 475, 25, 399, 316, 45, 6, 3, 2, 1028, 12, 103, 87, 3, 380, 14, 296, 97,
31, 2070, 55, 25, 140, 5, 193, 7485, 17, 3, 225, 21, 20, 133, 475, 25, 479, 4,
143, 29, 5534, 17, 50, 35, 27, 223, 91, 24, 103, 3, 225, 64, 15, 37, 1333, 87,
11, 15, 282, 4, 15, 4471, 112, 102, 31, 14, 15, 5344, 18, 177, 31]
```

8. Visualizing the Word_Embeddings

Visualize the word embeddings for 10 of the words using pandas DataFrame like we did in lab 3.

Sanity check.
Your output should look like this.

```
                0         1         2         3         4         5   \
woods        0.043011  0.057863  0.016541  0.004355  0.008027  0.048465
hanging     -0.016311 -0.062583  0.046546  0.016145  0.039565 -0.020009
woody        0.021778 -0.037777 -0.030776  0.013886 -0.034078 -0.045006
arranged     0.030475 -0.021358 -0.020872  0.059222 -0.049794  0.062016
bringing     0.010466  0.028823  0.038332 -0.032471 -0.007872 -0.017367
wooden      -0.026852 -0.006252 -0.009357 -0.040450  0.018227  0.009683
errors       0.035765  0.017572  0.019605 -0.023359 -0.029060  0.045065
dialogs      0.036447 -0.021358  0.008285 -0.005219  0.022035  0.036900
kids         0.044978 -0.010639 -0.034521  0.040310  0.059508  0.003192
uplifting   -0.038223 -0.013133 -0.005411  0.019047  0.056864 -0.043192

                6         7         8         9     ...          90  \
woods        0.033225 -0.089236 -0.027951  0.049966  ...     0.119137
hanging      0.031527 -0.082795 -0.035165 -0.076627  ...     0.005154
woody        0.010373 -0.025600 -0.005656  0.023710  ...     0.001916
arranged     0.034865  0.022738 -0.011583  0.019224  ...    -0.032254
bringing    -0.005092 -0.059275  0.011880 -0.042273  ...    -0.009063
wooden      -0.012759  0.043928 -0.026520 -0.037386  ...     0.044405
errors       0.036590 -0.011786 -0.025057 -0.011152  ...     0.009108
dialogs     -0.025551 -0.002114  0.045453 -0.038898  ...     0.043944
kids         0.005469 -0.025277  0.033515  0.038443  ...     0.002067
uplifting   -0.013826  0.026510 -0.006041 -0.034192  ...     0.017940

                91        92        93        94        95        96  \
woods        0.032055  0.042732 -0.013019 -0.051505 -0.020607 -0.040440
hanging     -0.077567  0.022845 -0.046255  0.004053 -0.030766  0.020483
woody        0.048208 -0.014160  0.017805 -0.020808 -0.021415  0.019141
arranged    -0.034876 -0.016688  0.010584  0.045347 -0.059160  0.024925
bringing    -0.004028  0.045623  0.045389 -0.004814 -0.032158  0.026791
wooden       0.021138 -0.034470  0.014399  0.013768  0.003698  0.048629
errors      -0.007221 -0.003225  0.038257 -0.051599 -0.034307 -0.008337
dialogs      0.025793 -0.004957  0.040209 -0.039208  0.010382 -0.048758
kids         0.021802 -0.012816 -0.008089  0.013139  0.000401  0.017686
uplifting   -0.029102  0.040931  0.004778  0.040721  0.024920  0.026093

                97        98        99
woods        0.035259  0.035716  0.075549
hanging     -0.082727  0.037009  0.133169
woody        0.001701 -0.030822  0.023810
arranged    -0.047748  0.033776  0.007249
bringing     0.016461 -0.040839  0.051884
wooden      -0.020214  0.043088 -0.011488
errors       0.050736  0.030428 -0.015540
dialogs     -0.028492  0.009481 -0.054642
kids        -0.036334  0.018573  0.004151
uplifting   -0.025211  0.026499 -0.006089

[10 rows x 100 columns]
```

Plot the word embeddings using TSNE.

```
>>> from sklearn.manifold import TSNE
>>> import matplotlib.pyplot as plt
>>> import numpy as np

>>> tsne = TSNE(perplexity=3, n_components=2, init='pca', n_iter=5000, method='exact')
>>> np.set_printoptions(suppress=True)
# starting from the first word
```

```
>>> start = 3
>>> plot_only = 54
>>> T = tsne.fit_transform(word_embeddings[start:plot_only, :])
>>> labels = [idx2word[i] for i in range(start, plot_only)]
>>> plt.figure(figsize=(14, 8))
>>> plt.scatter(T[:, 0], T[:, 1])
>>> for label, x, y in zip(labels, T[:, 0], T[:, 1]):
        plt.annotate(label, xy=(x+1, y+1), xytext=(0, 0), textcoords='offset points',
        ha='right',va='bottom')
```

9. Questions

1. Create a new model that is a copy of the model step 3. To this new model, add two dropout layers, one between the embedding layer and the LSTM layer and another between the LSTM layer and the output layer. Repeat steps 4 and 5 for this model. What do you observe? How about if you train this new model for 6 epochs instead?
2. Experiment with compiling the model with batch sizes of 1, 32, len(training_data). What do you observe?
3. **(optional)** Can you retrain with a Bidirectional LSTM instead of an LSTM? What do you observe about the Bi-LSTM model at 3 epochs? What about at 6 epochs?