Deep Learning for NLP

Word and sentence embeddings

Sentence Classification with LSTMs/ConvNets

Flavie Vampouille MSc in DS&BA

CentraleSupelec

ESSEC Business School

Note: All items used for answering the questions lie in the txt files attached.

Part 3. WORD AND SENTENCES EMBEDDINGS WITH WORD2VEC

QUESTION 1 Run the **"first part"** (you don't need to finish the training of the model). From the INFO that is shown while training :

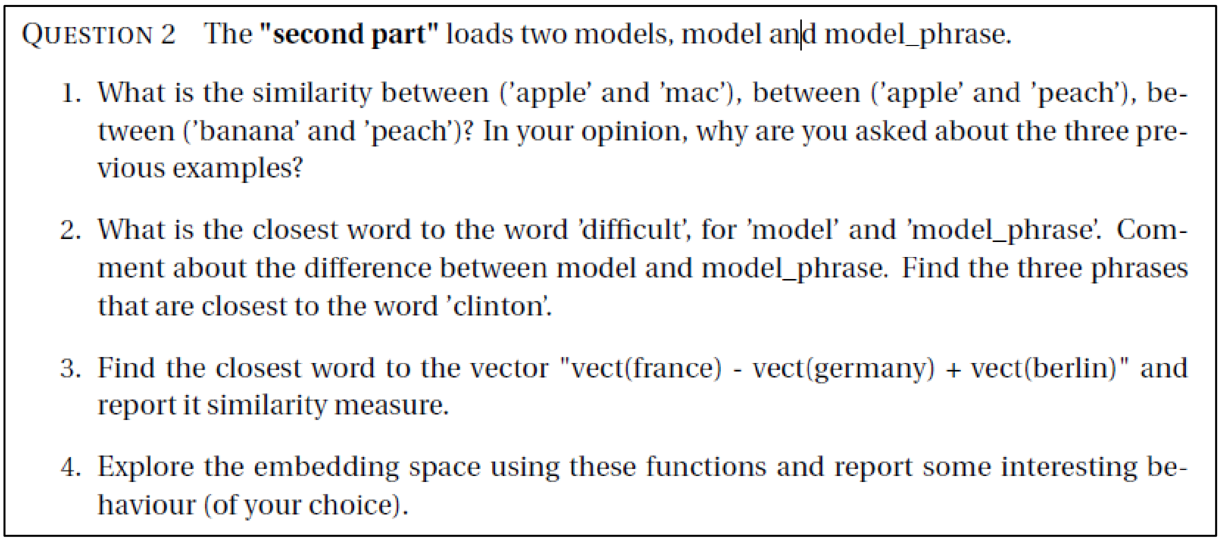
1. What is the total number of raw words found in the corpus?
2. What is the number of words retained in the word2vec vocabulary (with default min_count = 5)?

Q1.1

After running task 1, the total number of raw words found in the corpus is 17 005 207.

Q1.2

The number of words retained in the word2vec vocabulary with default min_count = 5 is 71 290.



QUESTION 2 The **"second part"** loads two models, model and model_phrase.

1. What is the similarity between ('apple' and 'mac'), between ('apple' and 'peach'), between ('banana' and 'peach')? In your opinion, why are you asked about the three previous examples?
2. What is the closest word to the word 'difficult', for 'model' and 'model_phrase'. Comment about the difference between model and model_phrase. Find the three phrases that are closest to the word 'clinton'.
3. Find the closest word to the vector "vect(france) - vect(germany) + vect(berlin)" and report it similarity measure.
4. Explore the embedding space using these functions and report some interesting behaviour (of your choice).

Q2.1

Words	*"apple" and "mac"*	*"apple" and "peach"*	*"banana" and "peach"*
Similarity	*0.567861632452*	*0.178399832237*	*0.688715470006*

We studied the similarities between these pairs of words to show that the similarity depends of the context. The word "apple" can refer both to the computer brand or the fruit and in our case, we can see that it is more often associated to the computer brand and has very low similarity with another fruit name. In reverse, fruit's names that are not ambiguous have a big similarity.

Q2.2

Model used:	model	model_phrase
Closest word to "difficult"	*'easy' 0.7628731727600098*	*'very_difficult' 0.870441734790802*

The model "model" is trained on single words (unigram) and so it returns "easy" as being the word with the most similarity to "difficult". The model "model_phrase" being trained on two-consecutives words (bigram) it returns "very_difficult" as being the closer to "difficult".

The three phrases that are the closest to the word "clinton" are

Closest word	"bush"	"Reagan"	"gore"
Similarity	*0.8667015433311462*	*0.859365701675415*	*0.8575333952903748*

Q2.3

The closest word to the vector "vect(france) -vect(germany) +vect(berlin)" is “paris” with a similarity measure of 0.757699728012085.

Q2.4

The closest words to the vector “vec(nasa) +vec(image)” are

'landsat'	'satellite'	'photo'	‘gif’	'voyager'	‘iss’	‘hipparcos’	‘gemini’
0.6949	0.6498	0.6519	0.6453	0.6433	0.6350	0.6095	0.5949

We can see that if some are really relevant, others are just generic terms like ‘photo’ or ‘gif’. However all the words are really connected to Nasa images. This is probably due to the fact that the term “nasa” is really specific.

When I used most common words, the relation between the closest words and the words studied is less evident. For example, the closest words to the vector “vec(sky) +vec(animal)” are:

‘bird’	‘eye’	‘giant’	‘creatures’	‘flesh’	‘creature’	‘herd’	‘skin’
0.6866	0.6548	0.6414	0.6376	0.6349	0.6340	0.6276	0.6275

Except the first one, the others even if they are related to my terms were quite unexpected. They all stay in the domain of animal parts or description but the relation with the sky is not so easy to see.

As the word “insect” doesn’t appear in the above list but is related to bird which I expected to be the main subject I try to remove it to see if there will be any difference. The closest words to the vector “vec(sky) +vec(animal) -vec(insect)” are

‘heavens’	‘heaven’	‘planet’	‘sun’	‘dead’	‘moon’	‘darkness’	‘shining’
0.6082	0.5848	0.5502	0.5309	0.5248	0.5165	0.5084	0.5083

We can see that removing the word “insect” leads to remove everything related to birds and animals and that we obtained a whole new relation, more related to biblical or planetary ideas.

QUESTION 3 The **"third part"** proposes a way to construct sentence embeddings from word embeddings, by simply taking the mean of the word embeddings contained in the sentence. The "avgword2vec" vector of a sentence is computed :

$$\boxed{\text{avgword2vec(s)} = \frac{\sum_{i=1}^{S} \alpha_i \text{word2vec}(w_i)}{\sum_{i=1}^{S} \alpha_i}} \tag{3.1}$$

with $\alpha_i = 1, \forall i = 1, \ldots, S$, the weights of the word vectors.

1. Fill the blanks of the "cosine_similarity" function. Report the closest sentence to the sentence with idx "777", and their similarity score.
2. Fill the blanks of the "most_5_similar" function. Report the 5 closest sentences to the sentence with idx "777", and the associated similarity scores.

Q3.1

The sentence “a girl trying to get fit for a bodybuilding competition .” is most similar to “gymnasts get ready for a competition .” with a similarity score of : 0.902949750423.

Q3.2

The sentence “a girl trying to get fit for a bodybuilding competition .” is most similar to:

“gymnasts get ready for a competition .” with similarity 0.90294975

“a woman is getting ready to perform a song for the audience .” with similarity 0.89009732

“a runner in a competition want to go to the finish line .” with similarity 0.85553652

“men working to build a fence for customers .” with similarity 0.8514716

“a man prepares to give a speech for a television audience .” with similarity 0.84947604

QUESTION 4 (OPTIONAL) In the **"fourth part"**, we are going to define the weights α_i as the IDF scores of each word. The TF-IDF is a classical method to construct a representation of a document. TF-IDF is composed of TF (Term Frequency) and IDF (Inverse Document Frequency). In our case, documents are just sentences, and the term frequency of each word inside a sentence is not very informative (they usually all appear once or twice max). But the IDF score is informative, and we are going to use them to embed a sentence using a weighted average of word2vec embeddings.

$$\boxed{\text{IDF\_avgword2vec(s)} = \frac{\sum_{i=1}^{S} \text{idf}(w_i).\text{word2vec}(w_i)}{\sum_{i=1}^{S} \text{idf}(w_i)}} \tag{3.2}$$

1. Implement the "IDF" function. Report the IDF score of the word "the", the word "a", and the word "clinton".
2. Implement the "avg_word2vec_idf" (very small modification from "avg_word2vec" function) using the "word2idf" dictionary. Report the closest sentence to sentence with idx 777.

Q4.1

IDF score of the word "the" : 0.867762351618

IDF score of the word "a" : 0.473266274881

The word “clinton” doesn’t appear in the text, hence we can’t calculate the IDF.

Q4.2

With avg_word2vec_idf function, the query : “a girl trying to get fit for a bodybuilding competition .” is most similar to

“gymnasts get ready for a competition” with a score of: 0.897237539291

“a woman is getting ready to perform a song for the audience” with a score of: 0.800492525101

“a runner in a competition want to go to the finish line” with a score of: 0.795241653919

“the men are trying to fix a car” with a score of: 0.783501505852

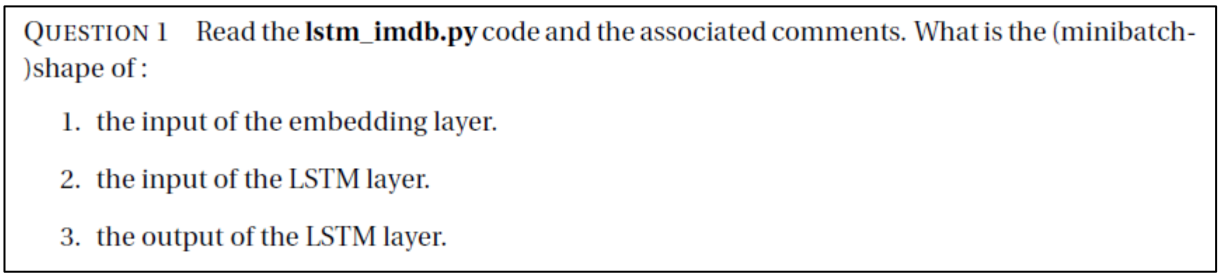
“a dog is going to get wet” with a score of: 0.780033588409

The most similar sentence to sentence with idx 777 is the same with both avg_word2vec and avg_word2vec_idf functions, but with different scores of similarity, the score being lower with IDF method:

- 0.90294975 with avg_word2vec
- 0.898546099663 with avg_word2vec_idf

As in question 3.2, I look at the five most similar sentences and we can see that it diverges after the three first sentences.

Part 4. SIMPLE LSTM FOR SEQUENCE CLASSIFICATION



QUESTION 1 Read the **lstm_imdb.py** code and the associated comments. What is the (minibatch-)shape of :

1. the input of the embedding layer.
2. the input of the LSTM layer.
3. the output of the LSTM layer.

Q1.1

The input of the embedding layer is of shape: (15000, 32), because it is (vocab_size , embed_dim).

Q1.2

The input of the LSTM layer is the output of the embedding layer, of shape (None, None, 32).

Q1.3

The output of the LSTM layer is of shape (None, 64).

```
Layer (type)                  Output Shape          Param #        Connected to
====================================================================================
embedding_1 (Embedding)      (None, None, 32)      480000         embedding_input_1[0][0]
____________________________________________________________________________________
lstm_1 (LSTM)                (None, 64)            24832           embedding_1[0][0]
____________________________________________________________________________________
```

QUESTION 2

1. From the output of the **lstm_imdb.py** script, report the number of parameters of the model with the standard set of hyper-parameters. Report also the number of sentences in the training set. In standard statistics, a rule of thumb is to have less parameters than samples in your dataset. How do you think it's possible to train a model that has so many parameters compared to this number of samples? (we expect one key word)

The number of parameters of the model with the standard set of hyper-parameters is 504 897.

The number of sentences in the training set is 28 000.

To train a model that has a lot of parameters and only a few number of samples the best way is to drop some parameters (dropout method).

QUESTION 3 The word embeddings are fed to the LSTM, which outputs a sentence embedding. This sentence embedding is then fed to a classifier, that outputs the prediction of the label.

1. For a single sentence, the LSTM has states $h_1, \dots, h_T$ where T is the number of words in the sentence. The sentence embeddings that is fed to the classifier is thus computed as $f(h_1, \dots, h_T)$. What is the exact form of $f(h_1, \dots, h_T)$ used in the python script (look at the keras doc of the lstm function)?

The default parameters in keras lstm fuction are:

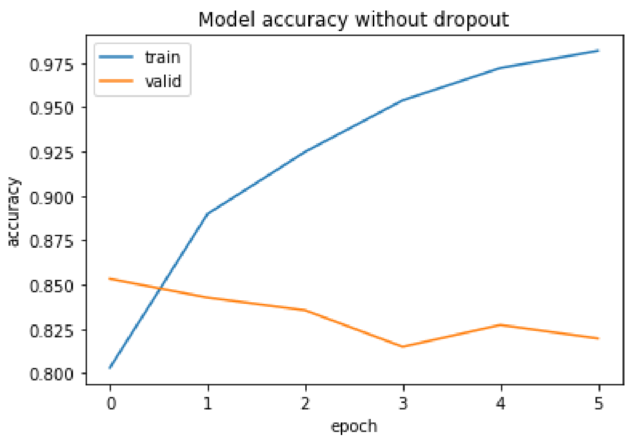
```
keras.layers.recurrent.LSTM(output_dim, init='glorot_uniform', inner_init='orthogonal',
forget_bias_init='one', activation='tanh', inner_activation='hard_sigmoid',
W_regularizer=None, U_regularizer=None, b_regularizer=None, dropout_W=0.0,
dropout_U=0.0)
```

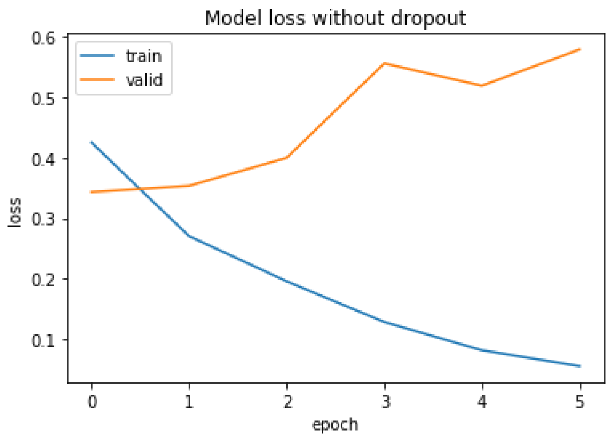
hence the exact form of $f(h_1, \dots, h_T)$ is by default 'tanh', the hyperbolic tangent. It can possibly be change to 'softplus', 'Softsign', 'relu', 'sigmoid', 'hard_sigmoid' or 'linear'.

QUESTION 4 Run the model with and without dropout (training should take less than 10 minutes).

1. For each, plot the evolution of the train and valid accuracy per epoch, and write the test errors that you obtain. You can use the python package "matplotlib" that's already installed (import matplotlib). Matplotlib tutorial.

<u>Without dropout</u>



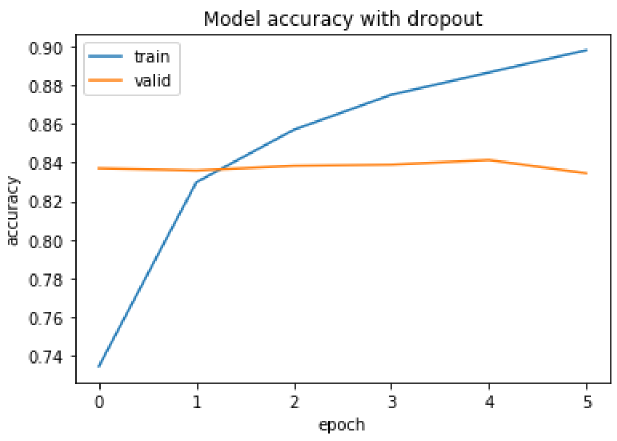


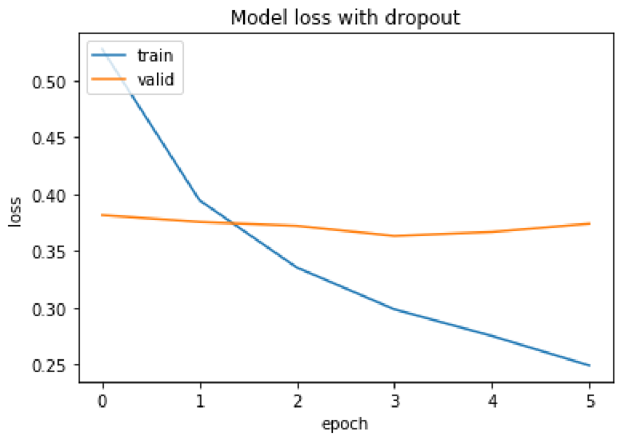
Test accuracy: 0.82

Test loss: 0.582743918196

We can see that the model overfit as the accuracy on the validation set decrease with the number of epoch.

With dropout





Test accuracy: 0.8358

Test loss: 0.377364609194

When we add a dropout on the parameters of the models, we obtain a better score for the accuracy on the training set.

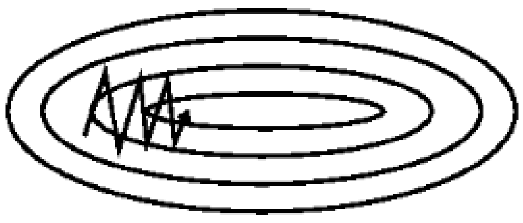
QUESTION 5

1. Explain what is the difference between SGD and Adam. (no more than 3-4 lines) (more info here : Optimization methods).

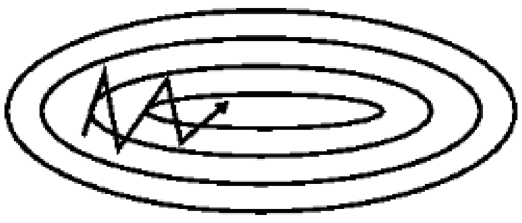
SGD is a step by step algorithm used to reach a minimum. It provides good convergence but can stay stuck a long time in an area if the surfaces curves are more steeply in one dimension than in the other (like a ravine) and so the execution time could be long.

Adam (Adaptive Moment Estimation) changes the learning rate for each parameter at each step (using exponentially decaying average on both past gradients and past squared gradients). This leads to increase the learning rate if the gradient points toward the minimum and decrease it if the gradient change direction (oscillate).

Hence **the convergence will be faster with Adam than with SGD without momentum and there will be less oscillations.**



SGD without momentum



Adam

Source: http://sebastianruder.com/optimizing-gradient-descent/index.html#adam

Part 5. SIMPLE CONVNET FOR SEQUENCE CLASSIFICATION

QUESTION 1 Fill the gaps of **cnn_imdb.py**. Run the script, and report the results (test loss and test error) that you obtain.

The tests loss and accuracy I obtained after running the cnn_imdb.py script are:

- Test loss = 0.357824989335
- Test accuracy = 0.842866666667

QUESTION 2

1. In **cnn_imdb.py**: what is the input and output shape of Convolution1D?

In cnn_imdb.py, the input shape of Convolution1D is (80,16) and its output shape is (78,250) according to the Built model printing in the consol.

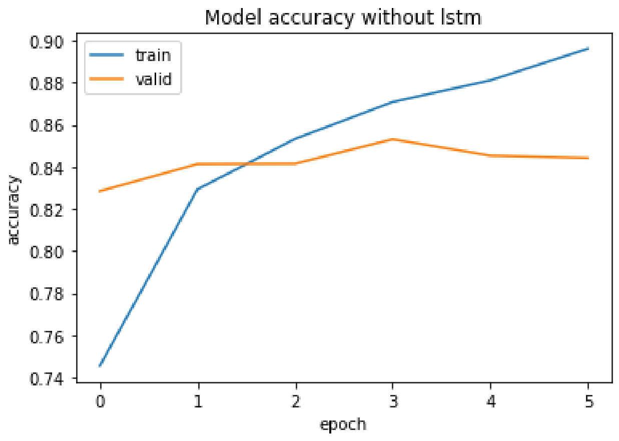
Layer (type)	Output Shape	Param #	Connected to
embedding_10 (Embedding)	(None, 80, 16)	80000	embedding_input_10[0][0]
convolution1d_2 (Convolution1D)	(None, 78, 250)	12250	embedding_10[0][0]

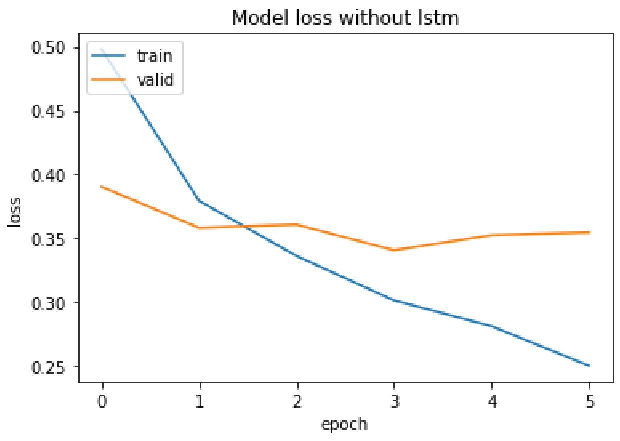
QUESTION 3 (+1.5 points)

1. In a new **cnn_lstm_imdb.py** script that you create (and include in your assignment), build a model where on top of the convolution, you have an LSTM. It means that the input of the LSTM will be the output of your ConvNet (yes, it's possible, and usually done in the litterature). Run the model with the best parameters you find (sufficiently small so that you have time to run the model, limiting over/under fitting). Report your best results.

In cnn_lstm_imdb.py script I use the cnn_imdb.py script in which I add a LSTM layer on top of the convolution. As in the previous script I tried it with and without convolution.

Results with cnn_imdb.py script, without LSTM layer

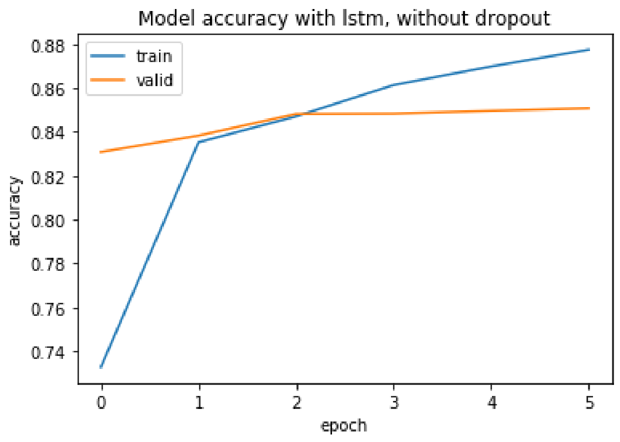


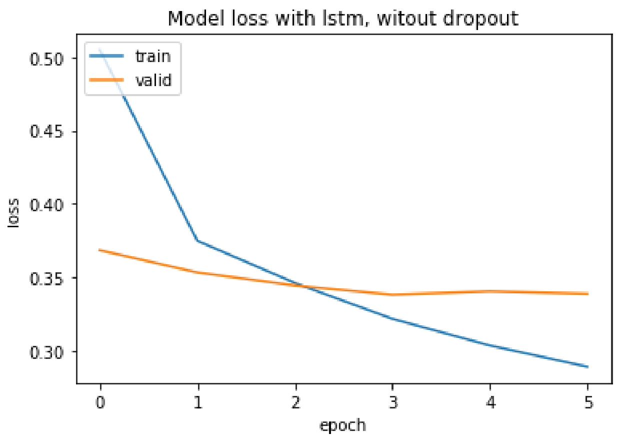


Test accuracy: 0.842866666667

Test loss: 0.357824989335

Results with cnn_imdb.py script, with LSTM layer and without dropout

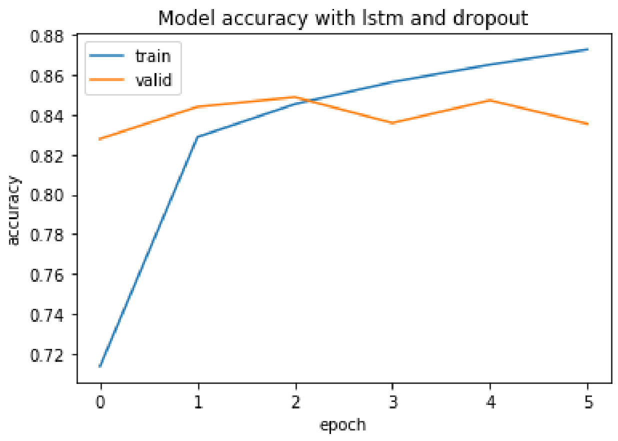


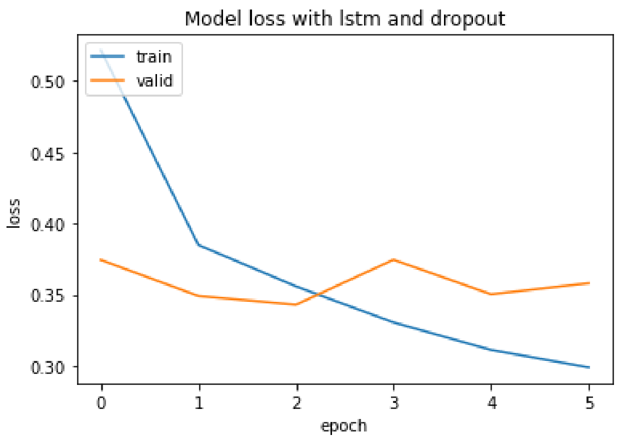


Test accuracy: 0.8478

Test loss: 0.35015932436

Results with cnn_imdb.py script, with LSTM layer and with dropout





Test accuracy: 0.8394

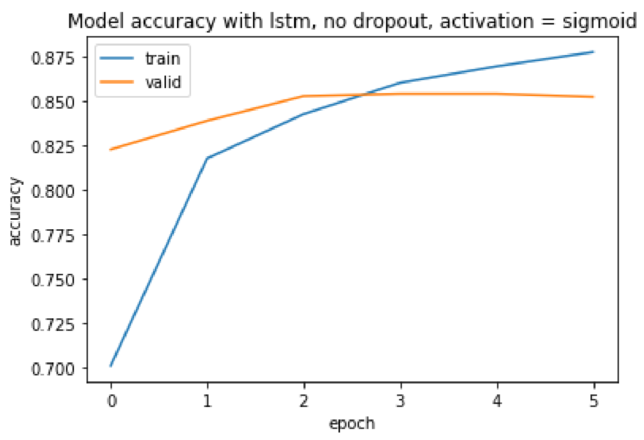
Test loss: 0.363586441755

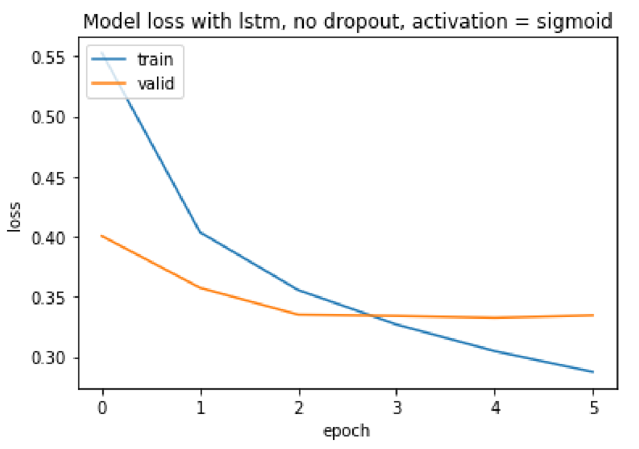
<u>Best results</u>

- We can see that the cnn model doesn't overfit too much, hence when we add a lstm layer with dropout the model tends to underfit and the accuracy is only 0.8394 which is lower than the accuracy with only cnn model of 0.8428.
- When we add a lstm layer without dropout, we obtain the best accuracy of 0.8394.

I then try different activation function in the LSTM layer.

<u>Results with cnn_imdb.py script, with LSTM layer, no dropout, activation = 'sigmoid'</u>

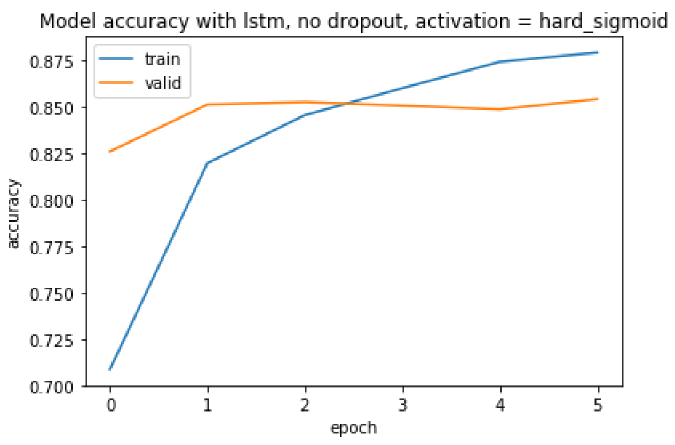


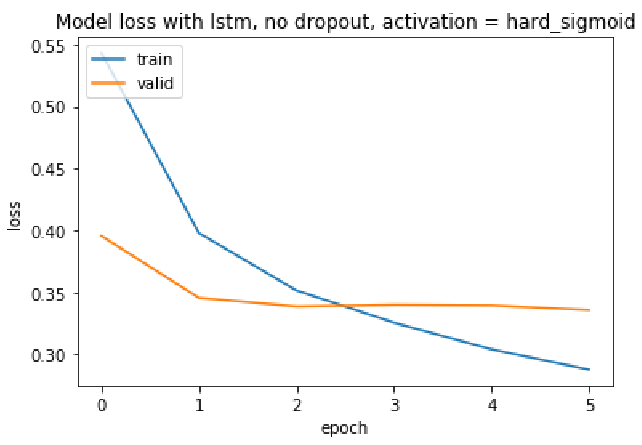


Test accuracy: 0.845866666667

Test loss: 0.346548287169

<u>Results with cnn_imdb.py script, with LSTM layer, no dropout, activation = 'hard_sigmoid'</u>





Test accuracy: 0.847133333333

Test loss: 0.34394287672

Best result

The best results I obtain with my code are for a LSTM layer without dropout and with the activation function 'tanh'.

I have with these parameters the best accuracy score:

- **Test accuracy: 0.8478**
- **Test loss: 0.3501**

Or with LSTM layer without dropout and with the activation function 'hard_sigmoid'.

I have with these parameters the lower loss score:

- **Test accuracy: 0.8471**
- **Test loss: 0.3439**