L665 ML for NLP Spring 2018
Carlos Sathler (cssathler@gmail.com)

Assignment 3

Task 1.b – Sentence Level Classification with CNN

I chose to do sentence level classification using a simplified version of Yoon Kim's approach described in the paper "Convolutional Neural Networks for Sentence Classification" [1]. Kim Yoon describes multiple experiments with a number of datasets and model variations. I only used one model, CNN-rand, and one dataset, the movie review dataset "MR" [2]. All my solutions for this Assignment 3 were implemented with Jupyter notebooks using Keras with TensorFlow backend. The notebooks were pushed to GitHub and I include later in this report links to the various notebooks I created.

Dataset description:
The "movie review" dataset consists of 10,662 movie reviews in the form of sentences, half of which are positive reviews and half negative. The dataset is available in two files, one with positive reviews, and the other with negative reviews. My code reads the two files, concatenates the rows from the two files and assigns a flag = 1 for positive reviews and flag = 0 for negative reviews. The resulting dataset therefore consists of 10,662 rows each with two columns: a review and a flag.

Architecture and implementation:
Notebook "assignment3_task1_1.ipynb" implements a simplified version of Kim's CCN-rand experiment. I used multiple filters but with no variation to window size across the filters. Kim stated his filters had varying sizes, but due to performance limitations (I don't have free access to GRUs) I had to simplify my design. I tested two window sizes (3 and 5) on a couple filter sizes, and settled with window size of 3 words, and 128 filters. Another difference between my implementation and the one described in Kim Yoon's paper was the dimension of my word vectors. Kim Yoon reports using word vectors with dimension = 300, which generates too many parameters given my computing resources, so I chose much smaller vectors with dimension = 64. In my model each sentence was padded to length = 51, which was the max length of my tokenized sentence sequences.

As I said before, Kim Yoon describes a CNN with "multiple filter widths and feature maps". For my solution I followed the guidelines described in the class notes from Stanford class "CS231n: Convolutional Neural Networks for Visual Recognition". The CNN design guidelines from the class notes are in line with Kim Yoon's design. They can be summarized as:

"INPUT -> [[CONV -> RELU]*N -> POOL?]*M -> [FC -> RELU]*K -> FC
where the "*" indicates repetition, and the POOL? Indicates an optional pooling layer" [3].

I tried architectures with M = 1 and 2, and K varying between 1 and 3. Best results were obtained with N = M = K = 1, which is what I used.

Not much hyper parameter tuning was possible given the fact that I don't have access to GRUs, and my code was taking a long time to complete. Besides window size, I experimented with a few combinations of batch sizes and epochs, but once I started using early stopping (see below) it became clear 3 to 5 epochs were enough to train my models.

L665 ML for NLP Spring 2018
Carlos Sathler (cssathler@gmail.com)

Training procedure:
Kim Yoon used cross validation to train his model on the "movie review" dataset. He states in his paper that for this dataset "there was no standard train/test split and thus a 10-fold CV was used" [3, p 3]. For that reason I did not used static partitioning of the dataset for training. I followed Kim Yoon's approach and trained my model using 10 fold cross validation. I used dropout on the output of the convolution layer and the fully connected layer. To further reduce overfitting I used regularization in the output layer, and Keras early stopping with patience = 1. It appears I would have had better results using patience = 2 but that was making my code take too long to complete, so I decided to settle with patience = 1.

The loss function used during training was Keras' "binary-crossentropy". Optimizer was Adam. My final implementations use batch size = 128.

Evaluation methods and results:
I used the accuracy metric to evaluate my results, so I could compare them with Kim Yoon's. My final result was obtained by calculating average accuracy after running my model 10 times against 10 training/testing folds of the dataset. In other words, after using 10 fold cross validation. My final result (0.7347) is worse than the one reported by Kim Yoon (0.761). But I'm not convinced his reported result was obtained using the same method that I used. On page 3 of his paper Kim states, "For datasets without a standard dev set we randomly select 10% of the training data as the dev set." [3, p 3]. I too obtained accuracy higher than 0.7347 for certain folds during training. For example, in one split my accuracy was 0.7627, superior to the one reported in Kim Yoon's paper. At any rate, it should be expected that my results would be inferior considering my word vectors were much smaller and considering I only used one window size value.

Code: https://github.com/csathler/IU_MSDS/blob/master/NLP/assignment3/assignment3_task1_1.ipynb

Task 1.c – Extending the approaches for sentence level classification with CNN

Here are a *few examples* on how to extend Kim Yoon's approach by incorporating NLP features to his model:

1. Vectorize lemmas and/or stems derived from sentence tokens, and add these representations as channels to the input; feed this multi-channel dataset to a 2D convolutional CNN architecture.
2. Create BOW representation of NLP features extracted from the text, feed the BOW vector to perceptron layers (dense layers), and concatenate the output to dense layer output from CNN, on axis 0, then feed resulting tensor to 2D convolutional CNN architecture.
3. Create BOW representation of NLP feature(s) extracted from the text, feed it to one or more dense layers, and concatenate the output of MLP layers to the output of original CNN, on axis 1, then feed resulting tensor to MLP.
4. Create vectorized representation of NLP feature(s) extracted from the text, feed it to CNN architecture, then concatenate the output of the CNN layers to the output of the original CNN, on axis 1; feed resulting tensor to MLP.

For my implementation of Task 1.c I chose approach 4. Notebook assignment3_task1_2.ipynb implements my solution. Specifically, I created a network with two branches that process (1) a vectorized representation of words, and (2) a vectorized representation of POS tags. I chose a convolution with window size=3 for both words and POS tags convolutional layer, followed by max pooling and perceptron (dense) layers. The branch that processed word vectors output a tensor of dim=16. The branch that processed POS tag vectors output a tensor of dim=1. These tensors were concatenated on axis = 1 and feed to MLP. Training strategy was

unchanged and results were not improved. A higher dimension (>1) for the output of POS tag MLP layer yielded even worse results (<0.721). I was not able to improve the model by adding POS tag features to the input.

Below is the architecture of my CNN extended model. Branch 1 (words) is shown in red font, branch 2 (POS tag) in blue. The yellow highlight shows the output of branch 2. By changing the dimension of the output of branch 2 I tried to tune how much to extend the original implementation (task 1.b) with POS tag information. In the end, I found that the higher the dimension of the output for branch 2, the worse the results for the model. The best result was obtained with dim=1. Still, this result was inferior to the original "pure" CNN implementation.

Layer (type)	Output Shape	Param #	Connected to
input_67 (InputLayer)	(None, 51)	0	
input_68 (InputLayer)	(None, 51)	0	
embedding_67 (Embedding)	(None, 51, 64)	1247936	input_67[0][0]
embedding_68 (Embedding)	(None, 51, 32)	1600	input_68[0][0]
conv1d_66 (Conv1D)	(None, 51, 128)	1572992	embedding_67[0][0]
conv1d_67 (Conv1D)	(None, 51, 64)	196672	embedding_68[0][0]
dropout_185 (Dropout)	(None, 51, 128)	0	conv1d_66[0][0]
dropout_187 (Dropout)	(None, 51, 64)	0	conv1d_67[0][0]
max_pooling1d_66 (MaxPooling1D)	(None, 26, 128)	0	dropout_185[0][0]
max_pooling1d_67 (MaxPooling1D)	(None, 26, 64)	0	dropout_187[0][0]
flatten_66 (Flatten)	(None, 3328)	0	max_pooling1d_66[0][0]
flatten_67 (Flatten)	(None, 1664)	0	max_pooling1d_67[0][0]
dense_152 (Dense)	(None, 16)	53264	flatten_66[0][0]
dense_153 (Dense)	(None, 1)	1665	flatten_67[0][0]
dropout_186 (Dropout)	(None, 16)	0	dense_152[0][0]
dropout_188 (Dropout)	(None, 1)	0	dense_153[0][0]
concatenate_33 (Concatenate)	(None, 17)	0	dropout_186[0][0] dropout_188[0][0]
dense_154 (Dense)	(None, 16)	288	concatenate_33[0][0]
dropout_189 (Dropout)	(None, 16)	0	dense_154[0][0]
dense_155 (Dense)	(None, 1)	17	dropout_189[0][0]

Total params: 3,074,434
Trainable params: 3,074,434
Non-trainable params: 0

Code: https://github.com/csathler/IU_MSDS/blob/master/NLP/assignment3/assignment3_task1_2.ipynb

L665 ML for NLP Spring 2018
Carlos Sathler (cssathler@gmail.com)

Task 2 – Text Classification with RNN

Data: Same as task 1.

Architecture and implementation:
I chose a LSTM implementation of RNN for this task. In order to compare results with the CNN solution from task 1, I chose very same input for this task. Notebook "assignment3_task2_1.ipynb" contains the "pure" LSTM implementation that corresponds to the "pure" CNN implemented for task 1.b. I used the very same input to allow for like to like performance comparison between the two models. A single LSTM layer with 100 nodes followed by one perceptron layer ("dense" layer) implements my solution. The output of the dense layer is a tensor of dim=16, the same as dimension used in the CNN from task 1.b.

Training procedure, evaluation methods and results:
I used the very same training procedure and evaluation methods described for task 1.b. The only difference was in early stopping. The performance for the LSTM was much faster than the CNN, so my programs was ending quickly enough with patience = 1. Therefore I trained my final model with patience = 2.

Results from LSTM were superior to results from CNN: 0.7393 vs. 0.7347

Extended approaches:
I experimented with 3 extensions, which I describe below. Training procedure and evaluation methods are the same as the "pure" LSTM implementation. Best results were obtained by combining LSTM on word vectors with small feature set created from running a CNN on vectorized POS tags. The best solution (0.4705) was obtained with "Extension 2".

Extension 1: Increased word vectors, from dim=64 to dim=128
This extension implemented with notebook "assignment3_task2_2.ipynb" surprisingly did not improve results relative to the first LSTM. Result = 0.7385 was however superior to CNN from task 1.b.

Extension 2: Two-branch NN, with use of POS tag features
This extension, implemented with notebook "assignement3_task2_3.ipynb", is similar to the extension used on task 1.c. Two branches are created and in this case branch one is the "pure" LSTM. Branch 2 is the same branch used on task 1.c, described before; it uses POS tag sequences as input. Result = 0.4705 was the best result obtained in the assignment.

Extension 3: Two-branch NN, with word vectors only:
In this extension, implemented with notebook "assignement3_task2_4.ipynb", I tried to improve results by processing word vectors using LSTM and CNN. If POS tag convolution improved the result of the "pure" LSTM, perhaps convolution of word vectors would improve results even further. The result, 0.7281, was disappointing, in fact underperforming all other implementations in this assignment.

Code:
LSTM without extensions:
https://github.com/csathler/IU_MSDS/blob/master/NLP/assignment3/assignment3_task2_1.ipynb

L665 ML for NLP Spring 2018
Carlos Sathler (cssathler@gmail.com)

Extension 1: Increased word vectors, from dim=64 to dim=128
https://github.com/csathler/IU_MSDS/blob/master/NLP/assignment3/assignment3_task2_2.ipynb

Extension 2: Two branch NN, with use of POS tag features
https://github.com/csathler/IU_MSDS/blob/master/NLP/assignment3/assignment3_task2_3.ipynb

Extension 3: Two branch NN, with word vectors only:
https://github.com/csathler/IU_MSDS/blob/master/NLP/assignment3/assignment3_task2_4.ipynb

Summary

My CNN results were reasonably close to the one reported in Kim Yoon's paper, for dataset "movie review" and CNN-rand model variation, considering the resources I had available for this assignment. My best result for this sentence classification task was obtained when using vectorized representation of words as input for a LSTM layer, and by later combining the output from that layer to the output of a convolutional layer fed vectorized representations of POS tags. This solution was described in Task 2, extension 2 above.

References:

[1] Kim, Yoon. "Convolutional neural networks for sentence classification." *arXiv preprint arXiv:1408.5882* (2014).

[2] Pang, Bo, and Lillian Lee. "Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales." *Proceedings of the 43rd annual meeting on association for computational linguistics*. Association for Computational Linguistics, 2005.

[3] CS231n Convolutional Neural Networks for Visual Recognition, cs231n.github.io/convolutional-networks/.