Assignment 3: Sentiment Analysis with Convolutional Neural Networks

Filip Stefaniuk
`filipste@student.matnat.uio.no`

October 25, 2018

Introduction

In this assignment, I have implemented Convolutional Neural Network classifier based on Kim[1] 2014 and Ye Zang[2] 2015 work. I haven't achieved the accuracies listed in papers, but I suspect this is due to the fact that the dataset we have in this assignment is different. The authors of the mentioned papers used also the **phrases** part of the SST-2 dataset for training.
I have implemented two scripts:

- `train_model.py` - used for training
- `eval_on_test.py` - used for evaluation

Aditionally I have several functions splitted across the python modules:

- `data.py` - data preprocessing
- `emb.py` - loading embeddings and creation of embeddings layers
- `model.py` - functions for creating the models

All the experiments were run on abel with the scripts provided in the `scripts` directory. Source code, scripts, logs from experiments, results as json files, notebooks used to present the data and sourcecode of this report available in github repository[3]

1 Baseline sentiment analisys

I have build CNN model with hyperparameters listed in the assignment, namely:

1. categorical cross-entropy loss
2. pre-trained 300-D *word2vec* embeddings trained on Google News corpus, I used original version downloaded from the web.
3. sequence of static word embeddings
4. window sizes of 3, 4 and 5
5. 100-D filters
6. ReLU activation function
7. global 1-max pooling
8. dropout with dropout rate of 0.5

I have not restricted the vocabulary size resulting in having 13295 words in vocabulary. I have padded and truncated sentences to length of 50. As a result I received architecture listed below.

[1]Yoon Kim 2014
[2]Ye Zang et al. 2015
[3]https://github.uio.no/filipste/INF5820-Assignment3

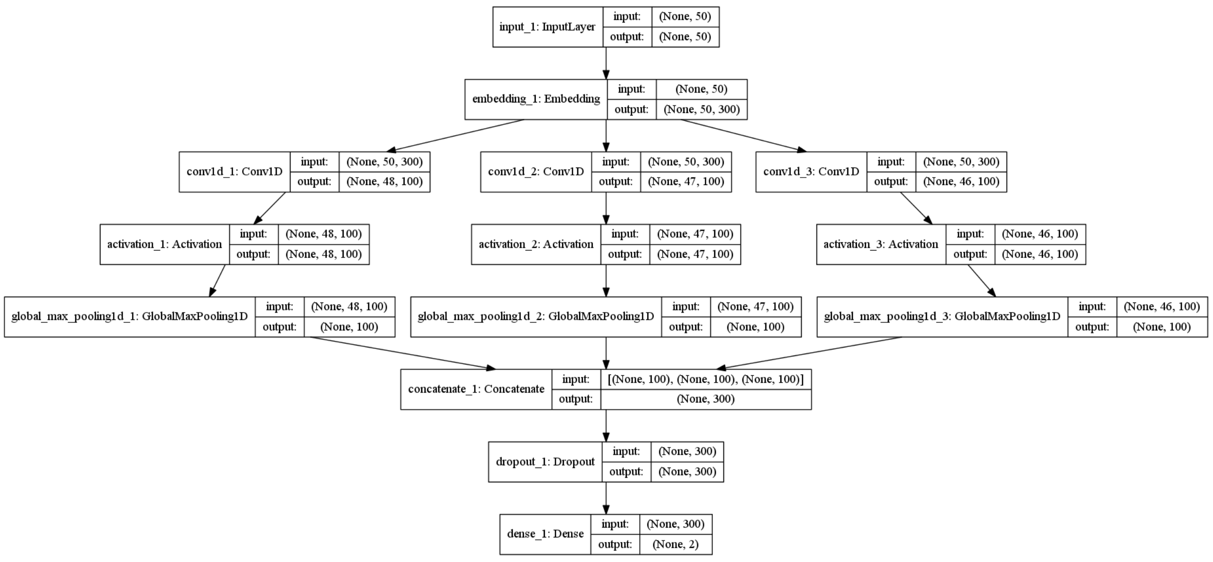

Figure 1: Baseline model architecture

I have trained the model using batch size of **256** and **adadelta** optimizer (same as in the papers).

2 Controlling the randomness

I have trained the model 10 times both with unset seed and seed set to **123**. I got the following results: Setting the seed doesn't change much, maybe because I am using GPU.

	min	max	mean	std
no seed	0.767	0.783	0.777	0.006
seed	0.776	0.799	0.789	0.007

3 Sensitivity analysis: tuning hyperparameters

When testing hyperparameters I was changing value of only one of them keeping the rest as in a baseline model. The authors of the *A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification* concluded that almost always 1-max pooling strategy outperformed all the other approches and dropout is the best regularization term. Because of that I decided that those two hyperparameters are not interesting to test and experimented with all the others. I set the same seed for all the experiments.

3.1 Activation function

I have tested multiple activation functions in convolutional layers. The *sigmoid* and *tanh* activations performed poorly. I tried different variations of *ReLU* activation namely *eLU* and *leakyReLU* but there was no improvement. Seems like the *ReLU* is the best activation function to use in this case, what is similar to the conclusions of authors of *A Sensitivity Analysys...* paper.

activation	accuracy
tanh	0.768485
sigmoid	0.665455
elu	0.780606
relu	**0.793939**
leakyrelu	0.789091

Figure 2: Accuracy with different activations

3.2 Convolutional window size

I have tested the same window sizes as in the previously mentioned paper, it seems that in this case there is not much of a difference when using different window sizes. Turns out that the best setting was either using the 3-4-5, 1-3-5-7 or simply 3 window size. Seems like window size of 3 is enough to capture necessairy information.

3.3 Number of filters per window size

Experimenting with the filter size, yielded results that the optimal size of the filter is 200.

window sizes	accuracy
1	0.787879
1-3-5-7	0.792727
14-15-16	0.753939
2-3-4	0.786667
2-3-4-5	0.773333
3	**0.796364**
3-4-5	0.790303
4-5-6	0.780606
5	0.773333
6-7-8-9	0.761212
7-8-9	0.761212

(a) Accuracy with different window sizes

	filter_size	accuracy
0	50	0.772121
1	100	0.786667
2	200	**0.791515**
3	300	0.790303
4	500	0.772121
5	1000	0.785455

(b) Accuracy with different filter sizes

3.4 Raw, lemmatized and POS tagged data

I have tried using POS tagged data, but it seems that the word embedding model with POS tags is not good. Using Google News model (*1.zip*) there was **9001** out of vocabulary tokens, so the model had very poor results.
To compare results when using lemmatized and non-lemmatized data I used models trained on Wikipedia and Gigaword (*17.zip, 18.zip, 19.zip 20.zip*). There was no clear improvement when using lemmatized data.

	f1	precision	recall
negative	0.679	0.659	0.701
positive	0.666	0.688	0.645
average	0.673	0.673	0.673

(a) Metrics when using model 1.zip

	raw	lemmatized
wiki+Gigaword (w2v)	0.655	0.661
wiki+Gigaword (glove)	0.753	0.765

(b) Comparison of using raw and lemmatized data

3.5 Multiple channels

I have tested model with multiple channels, one with static and the second with no-static word embeddings as in Kim 2014. Results are presented in section about influence of word embeddings.

4 Testing the influence of word embeddings

I have tested multiple word embeddings in 3 modes: **static**, **non-static** and **multichannel**. Fine tuning helps achieve better results, but the best scores were achieved when using **multichannel** mode. Fasttext turned out to be the best model.

	static	non-static	multichannel
wiki+Gigaword (w2v)	0.655	0.659	0.704
wiki+Gigaword (glove)	0.753	0.764	0.759
GoogleNews (w2v)	0.790	0.795	0.790
CommonCrawl 840B (glove)	0.790	0.794	0.804
wiki-news (fasttext)	**0.796**	**0.801**	**0.818**

Figure 5: Comparison of different word embeddings.

4.1 Inferring vectors for OOV words

Inferring OOV words further improved the results. Model trained with these embeddings achieved accuracy **0.818** in **static** mode.

	f1	precision	recall
negative	0.812	0.831	0.794
positive	0.824	0.807	0.842
average	0.818	0.819	0.818

Figure 6: Results after infering OOV words

4.2 Influence of vocabulary size

I have tested how changing the vocabulary size changes the number of the oov tokens. Experiments were performed on Google News *w2v*.

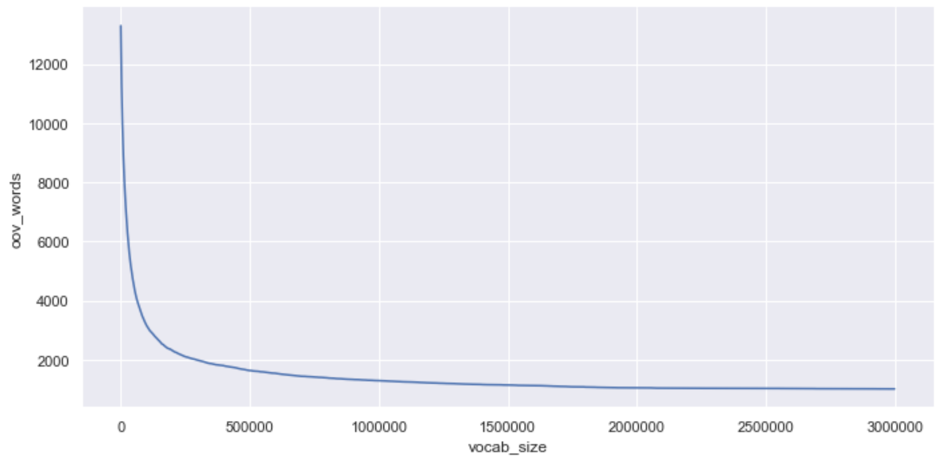

Figure 7: Results after infering OOV words

5 Evaluation of the best model