Exercises
**Deep Learning**
Fall 2018

**Machine Learning Institute**
Dept. of Computer Science, ETH Zürich
**Fernando Pérez-Cruz**
Web http://www.da.inf.ethz.ch/teaching/2018/DeepLearning/

# Series Friday, Nov 16, 2018
# (Deep Learning, Exercise series 8)

**Problem 1 (Word Embedding):**

Recall $\boldsymbol{z}_w$ and $\boldsymbol{x}_v$ denote the embedding of the predictor word $w$ and the predicted word $v$, respectively.

a. **Classification model for the co-occurrence:** We would like to model the co-occurrence of words $v$ and $w$ as a classification problem. Let $D_{v,w} \in \{0, 1\}$ be a random variable that is one when $v$ and $w$ have co-occurred and is zero otherwise. Propose a conditional probability distribution model for $P(D_{v,w}|\boldsymbol{x}_v, \boldsymbol{z}_w)$ using logistic regression model. Hint: use inner product as a similarity measure (more similar the words are, the more probable the co-occurrence is); then, use logistic function to make a valid conditional probability distribution.

b. **Derivation of the likelihood and log-likelihood function:** Suppose that we are given two sets of pairs of words: set $\mathcal{S}$ containing co-occurred pairs of words, and $\bar{\mathcal{S}}$ the pairs of words that have not co-occurred (this set is called the set of negative samples). Using the conditional probability distribution, which is driven in the last part, derive the likelihood $P(\mathcal{S}, \bar{\mathcal{S}}|\boldsymbol{z}_{w_i}s, \boldsymbol{x}_{v_i}s)$ and it logarithm $\log(P(\mathcal{S}, \bar{\mathcal{S}}|\boldsymbol{z}_{w_i}s, \boldsymbol{x}_{v_i}s))$.

c. **Stochastic approximation:** Relate the log-likelihood function $\log(P(\mathcal{S}, \bar{\mathcal{S}}|\boldsymbol{z}_w, \boldsymbol{x}_v))$ driven in part b. to the following objective (in page 10 of slide 8):

$$L(\mathcal{S}, \bar{\mathcal{S}}; \boldsymbol{x}_{v_i}s, \boldsymbol{z}_{w_i}s) = \sum_{(i,j)\in\mathcal{S}} \log \sigma(\boldsymbol{x}_{v_i}^\top \boldsymbol{z}_{w_j}) + k \sum_i \mathbb{E}_{w\sim P_n,(v_i,w)\in\bar{\mathcal{S}}} \left[\sigma(-\boldsymbol{x}_{v_i}^\top \boldsymbol{z}_w)\right]$$

where $p_n(w) = p(w)^\alpha$ and $p(w)$ is the relative word frequency. Note that $\sigma(\beta) = 1/(1 + \exp(-\beta))$ is the logistic function. Is the above objective easier to optimize compared to $\log(P(\mathcal{S}, \bar{\mathcal{S}}|\boldsymbol{z}_{w_i}s, \boldsymbol{x}_{v_i}s))$? why? What is the role of $\alpha$ in the sampling?

**Problem 2 (Practical: Word Embedding):**

Download the code skeleton word_embedding.ipynb from the course website and complete the implementation of the negative sampling.

**Problem 3 (Practical: CNN Sentiment Classification):**

The goal of this exercise is to implement a CNN model for sentiment classification. The model is taken from [2] and shown in Figure 1. The first layer embeds words into low-dimensional vectors. The next layer performs convolutions over the embedded word vectors using multiple filter sizes. For example, sliding over 3, 4 or 5 words at a time. Next, we max-pool the result of the convolutional layer into a long feature vector, add dropout regularization, and classify the result using a softmax layer. We will use only one channel for the convolution.

**Dataset** We provide a large set of training tweets (*twitter-dataset*), one tweet per line. All tweets have been pre-processed so that all words (tokens) are lower-cased and separated by a single whitespace. Tweets in `*.pos.txt` files are positive tweets and those in `*.neg.txt` files are negative ones. We suggest you start testing your code with the smaller files and move to the `full` files once you are confident that your model is doing what it should.

**Task** We provide the training loop (*train.py*) and data helper script (*data-helper.py*), as well as a skeleton for the CNN model (*text-cnn.py*). You should complete the code and fill in the gaps in the CNN model. If

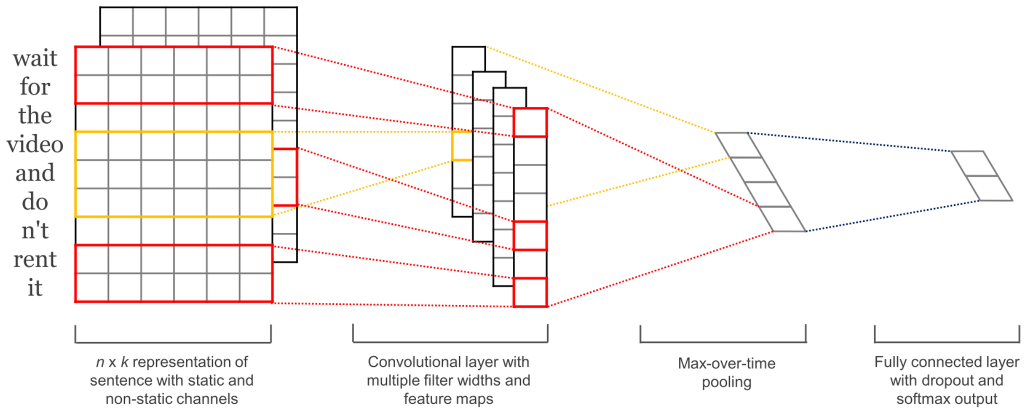
wait
for
the
video
and
do
n't
rent
it
*n* x *k* representation of sentence with static and non-static channels
Convolutional layer with multiple filter widths and feature maps
Max-over-time pooling
Fully connected layer with dropout and softmax output

Figure 1: CNN model for sentiment classification (from [2])

your code is correct, the accuracy on the dev set should go from 0.5 to about and 0.82 after 25K steps as show in Figure 2.

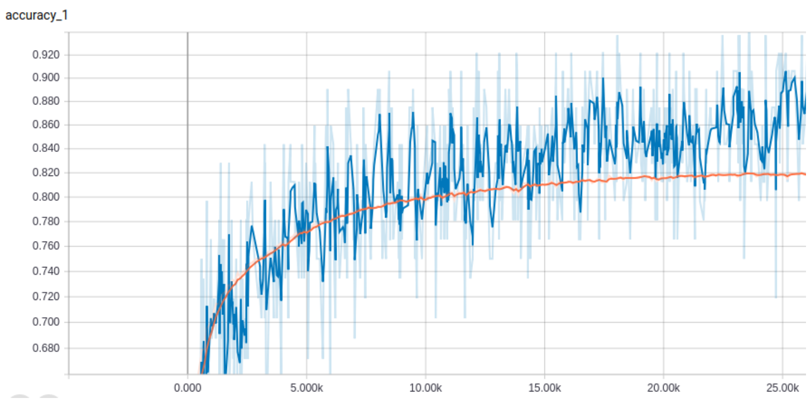

Figure 2: Classification accuracy for training (in blue) and dev (in orange) sets after 25K iterations

Then perform the following experiments:

1. Investigate the impact of the pretrained word embeddings by commenting out the indicated line in the train script. Run with the same parameters and compare the results.
2. Change the parametrization of the convolution, e.g. the *number of filters* and *their width*. What gives the best results? When is the model saturated?
3. Investigate the impact of the regularization by setting the regularitazion weight (*lambda* parameter) to some very large and very small values. What do you observe?
4. Study the influence of the dropout parameter and consider different dropout rates. How does this helps for regularization?
5. Explore the influence on the word embedding's size.

## References

[1] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[2] Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.