# Natural Language Understanding with Distributed Representations - Assignment 3
# Due Nov 4 (11:59 PM), 2016

In this assignment, you will play with the code from Tensorflow implementation of RNN [1] and think about the LSTM architecture [1]. You can also refer to other packages resources if you use them (e.g. the theano tutorial [2]). Besides, take a look at **this great article** for an introduction to recurrent neural networks and LSTMs in particular.

## 1 Bag of Words

Read the short description of the bag of words model here [3]. This is a very common way to represent text (it is often surprisingly effective too).

- Explain in your own words some potential shortcomings of a bag of words model. Provide an example to illustrate. No more than a 4-5 sentences are expected but you may write more if you think it is necessary (0.5 points)
- Starting from the LSTM model for sentiment analysis in the Theano tutorial, what is the simpliest modification to LSTM architecture which might allow us to recover (learn) the bag of words classification model through gradient descent? (Modify LSTM so that it does exactly the same thing as BoW) You do not need to write code here, just a description is expected. (0.5 points)

## 2 RNN Language Model

### 2.1 Train a language model (2 points)

In this assignment you will train a recurrent neural network on a challenging task of language modeling. The goal of the problem is to fit a probabilistic model which assigns probabilities to sentences. It does so by predicting next words in a text given a history of previous words. For this purpose you will use the Penn Tree Bank (PTB) dataset [4], which is a popular benchmark for measuring quality of these models, whilst being small and relatively fast to train. You will reproduce the results from Zaremba et al., 2014 [2], which achieves very good results on the PTB dataset.

You will use the starter code provided Tensorflow tutorial (also in the zip file on piazza). The data required for this assignment is in the 'data' directory of the PTB dataset from Tomas Mikolov's webpage [5].

The dataset is already preprocessed and contains overall 10000 different words, including the end-of-sentence marker and a special symbol (<unk>) for rare words. We convert all of them in the reader.py to unique integer identifiers to make it easy for the neural network to process.

To run the code:

```
python ptb_word_lm.py --data_path=/path/to/simple-examples/data/ --model small
```

[1]https://www.tensorflow.org/versions/r0.11/tutorials/recurrent/index.html

[2] http://deeplearning.net/tutorial/lstm.html

[3]https://en.wikipedia.org/wiki/Bag-of-words model

[4]http://www.cis.upenn.edu/%7Etreebank/

[5]http://www.fit.vutbr.cz/ imikolov/rnnlm/simple-examples.tgz

### 2.2 Which gate is the most important for this task? (2 points)

Unless you are quite familiar with LSTMs, you might find the gating architecture in the model a little bit ad-hoc. In this section, we will try to get a little insight into the relative importance of each of the gates for our task.

- Modify the code (rnn_cell.py, which is just a copy of tf.nn.rnn_cell) in the model so you omit first the input gate $I_t$, the forget gate $F_t$, and finally the output gate $O_t$(i.e they are always 1). Which gate was the most important for the empirical performance? Which gate was the least important?

For this problem, you should hand in your training/validation curves and the test results. Train a language model on the task, using the development set for early stopping. Report the best perplexity on the test set.

### 2.3 Gated Recurrent Unit (2 points)

In this section, you will replace the LSTM architecture used in the model with the Gated Recurrent Unit (see reference here [3]). Try your hand at tuning these models and see what works best.

- Which model performs better on the test set? Which model converges faster?

For this problem, you should hand in your training/validation curves, the test results and training times.

### 2.4 Setup evaluation

An RNN language model trained on such a small dataset and vocabulary won't be spectacular, so we won't bother with a full-fledged similarity or analogy evaluation. Instead, you are required to build a simple function, which grades the model on how well it captures ten easy/simple similarity comparisons (cosine similarity). The function returns a score between 0 and 10. Random embeddings can be expected to get a score of 5. In particular, you will compute the score based on the following logic:

```
def score(model):
    score = 0
    score += model.similarity('a', 'an') > model.similarity('a', 'document')
    score += model.similarity('in', 'of') > model.similarity('in', 'picture')
    score += model.similarity('nation', 'country') >  model.similarity('nation', 'end')
    score += model.similarity('films', 'movies') > model.similarity('films', 'almost')
    score += model.similarity('workers', 'employees') > model.similarity('workers', 'movies')
    score += model.similarity('institutions', 'organizations') >
             model.similarity('institution', 'big')
    score += model.similarity('assets', 'portfolio') > model.similarity('assets', 'down')
    score += model.similarity("'", ",") > model.similarity("'", 'quite')
    score += model.similarity('finance', 'acquisition') > model.similarity('finance', 'seems')
    score += model.similarity('good', 'great') > model.similarity('good', 'minutes')
    return score
```

in which $model$ is your trained language model, and $model.similarity(word1, word2)$ is some function that retrieves the word embedding and compute the cosine similarity of 2 words.

- Finish $model$ and $model.similarity(word1, word2)$ in the evaluation mentioned above, and report the score of your model (1 point);
- Use dimension reduction algorithms such as t-SNE to visualize your word vectors. (1 point).

## 3 Write-up

You should write a short report (3 pages) describing your experiments and results. (1 points)

Please email your code and report to dl4nlp2016@gmail.com. There is no need to hand in a hard copy. If you do not use the starter code, please include a script to run the code titled train (e.g train.py, train.lua etc).

## References

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

[2] Zaremba, Wojciech, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization. arXiv preprint arXiv:1409.2329, 2014.

[3] Kyunghyun Cho, Bart Van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. EMNLP, 2014.