Learning from Data – Week 4
Assignment 4:
Neural Networks and Deep Learning

General remarks

In this assignment you will train both neural networks and word embeddings. The first exercise will allow you to experiment with different neural network architectures with the goal of making the best possible system for the task of noun-noun compound classification. This will be organised as a intra-student competition during the lab. The results from this competition will be shown continuously on the big screen, and the winner will get a small prize. The rest of the assignment consists of training your own set of embeddings and doing a short exploration of those.

What you have to do:

- Send in at least one and at most **five** evaluation attempts during the live evaluation in the lab. If you cannot attend the lab, send in your best evaluation run as part of your Nestor submission, and you will get back how your system ranks compared to others.
- Upload on Nestor the code with your best model (see Exercise 4.1.4). Note that the final system you hand in does not have to be the one you complete during the lab.
- Fill in a Google Form where you will answer a few questions: `https://goo.gl/forms/LXNrcTzvGnnxlPBr2`. When filling this in, please refer to this pdf for more details on the questions, instructions and clarifications.

Deadline: Monday, October 1, 23.59.

Exercise 4.1 – Practice

Noun-noun compound classification is the task of identifying the semantic relation which holds within, e.g., 'leaf blower' (OBJECTIVE), 'police abuse' (SUBJECT), 'shoe box' (CONTAIN). The semantic relations (within brackets) can be made more clear by paraphrasing the semantic relations. A 'leaf blower' is a blower *for/of* leaves, hence the OBJECTIVE label. 'Police abuse' is abuse *by* the police, hence the SUBJECT label. A 'shoe box' is a box *containing* a shoe, hence the CONTAIN label. Note that these labels vary significantly between tag sets, and that they are subject to debate. This is an important task for many higher-level NLP problems,

such as machine translation. The data set we will use contains roughly 19000 annotated noun-noun compounds. You will be given labelled training data, and unlabelled test data on which we will evaluate your systems.

4.1.1 Features

Word embeddings have been found to yield excellent results for the task of noun-noun compound classification. The features used here are a concatenation of two 300-dimensional GloVe embeddings - one for each noun in the compound under consideration. Altering these features by, e.g., including word embeddings from other sources would be one way of increasing system performance. However, since the goal of this exercise is to experiment with neural network architectures and parameters, the features you use are **not** to be changed as part of this exercise.

4.1.2 Basic system

The Python code in `LFDassignment4.py` contains code for a very basic neural network, a multi-layer perceptron, which should help you get started. Make sure to specify your run number using the `-r` argument. The script also includes an option of showing a confusion matrix on your development data, using the `-cm` flag. Before changing anything, make sure that you've installed everything properly by running the system as is. What score do you get on the development/validation data?

4.1.3 Competition

The goal of this exercise is to achieve the highest possible performance on the **test set**. Use the following procedure when experimenting with your neural network:

1. Make a change in the architecture or parameter values of your neural network
2. Evaluate this on the development (or with cross-validation)
3. Note down which parameter you changed and how this affected your score
4. If your score increases sufficiently, run your system on the test set and send us your output
5. If your score decreases, change something else (you may want to reset the previous change)

Make note of the steps you take and the procedure you use. This will be the basis for your answers to the questions in the Google form. Document your Python code appropriately.

You can submit your system's output by uploading the file to a Dropbox Requests folder at https://www.dropbox.com/request/UGuGLktsIOYIhRov0pjy using the following filename format: `test_predicted_runX.npy`, where `X` is the number of the run you send in. Make sure you enter your name in the upload form. You do **not** need a Dropbox account to upload your file.

The scores of your systems on the test set will be shown continuously on the big screen. The designer of the best scoring system at the end of the lab will be rewarded with a small prize! Scoring will end at: **14:45:00 sharp!**

4.1.4 Architecture choices

Describe the architecture you've used in your best system, in terms of all parameter settings (e.g. layers, optimisation algorithm, number of neurons, activation functions, dropout, other regularisation, initialization, etc.). Why did you make these parameter choices? Which parameters affected your results the most? Describe the method you used to choose parameters, as noted in the section 4.1.3.

Exercise 4.2 – Training Your Own Word Embeddings

This exercise is similar to the exercise on word embeddings of last week, with one important difference: you'll have to train your own word embeddings. In this exercise, however, we'll use the original word2vec toolkit, which you have used last week already.[1]

The content of a set of embeddings is determined by two factors: the corpus its trained on, and the training procedure. By choosing your own corpus and training settings, the embeddings should be quite different from the ones you explored last week.

First, choose your corpus. It shouldn't be too small, at least 1 million words, but preferably over 5 million words. Generally, bigger is better here. The more different the corpus is from the GoogleNews corpus, the more interesting your results will be. So, try and find a corpus from a different time period, genre, or language. If you use a language that is not Dutch or English, make sure to provide us with translations for the words you test, so we can understand what's going on.

Second, train your word embeddings using the `word2vec` program in the word2vec toolkit. Just running `./word2vec` will give you an overview of the arguments it takes. You should specify the input corpus, the output file you want, and that the vectors should be saved in binary mode. You are free to change all other parameters, but default parameters should work well in most cases. Just make sure the vocabulary size of your embeddings is reasonable: if it's too large, increase the value of `-min-count`, if it's too small, decrease the value of `min-count`. Depending on the size of your corpus, vocabulary and the number and speed of CPUs used, training the embeddings might take a while.

Once you've trained your embeddings, check the most similar words for the same 5 target words you've tested in Assignment 3 using the `distance` tool. You can also choose 5 new words. In that case, make sure to test them in the GoogleNews embeddings too. Report on interesting, surprising, and/or amusing differences you encounter and try to relate these outcomes to the corpus you used.

[1] In other settings, the gensim toolkit for Python is likely the best option for you to use when training your own embeddings.