# Learning from Data – Assignment 4: Neural Networks & Word Embeddings

## General remarks

This assignment has to be completed **individually**. During the lab, we will have a **small live competition**! See below for details. The best performing students get a bonus on their grade.

This assignment is meant to get you acquainted with the basics of neural networks: by implementing an MLP and experimenting with word embeddings. First, you are provided with (the code for) a basic neural classifier and will have to make adaptations to that. Second, you will load existing word embeddings and perform small experiments on them.

What you have to hand in:

- A report that answers the questions. This time, you don't have to format it as a research paper, just answer the questions! Both Word and Latex are fine, as long as you submit a PDF. There's also no need to submit your code, as it will likely be very similar to the script you start out with.

**Deadline: Monday, October 4th, 10.59**

## Data and Software

For this assignment, we will be using a set of Named Entity Disambiguation data, extracted from the OntoNotes corpus. It consists of single-word named entities, labelled as belonging to one of five different categories: geo-political entity (GPE), person (PERSON), organisation (ORG), date (DATE) and cardinal number (CARDINAL). The data is in the file `NE_train.txt` on Nestor.

**Note:** In addition to scikit-learn, we are using the Python deep learning library Keras in this week and next week's assignments. If you work on your own machine, make sure to install Keras first.

For those of you not familiar with Keras, there is a very quick overview here, and in general the excellent documentation at `https://keras.io` should help you out.

## Exercise 4.1 – Using a neural network for named entity disambiguation

You are given four files for this assignment:

- `NE_train.txt`: a corpus of single-word named entities, and their named entity labels. Each training example is on one line, which consists of a word, followed by its label, and separated by a tab.
- `NE_test_input.txt`: the test set of single-word named entities. The gold standard labels are only available to us.
- `glove_filtered.json`: a Python dictionary containing the 300-dimensional word vectors, taken from here: `http://nlp.stanford.edu/projects/glove/`. Provided as a json-file, so it's quicker to read in.

  **Note:** for effiency, this set of embeddings contains only lower-cased, single words that occur in the named entity data set. You cannot use these embeddings for assignment 4.2!
- `lfd_assignment4.py`: a Python script, similar to the ones from previous assignments, that does basic neural network classification with (sort of) default parameters, and reads in the data and word embeddings. This is the file you have to experiment with!

### 4.1.1 Live Competition

Currently, the script you're given uses a neural network with 1 layer to perform 5-class classification. For each input word, it uses the Glove embeddings as features. It reaches a performance of around 70% accuracy. This seems good, but can surely be improved!

We will have a nice **live competition** during the lab! You have to experiment with the neural network classifier and see if you can push its performance. We have a held out test set on which we will measure your performance. The input for this test set can be downloaded from Nestor (`NE_test_input.txt`). You get **at most 5** submissions to get a score on this test set. Your highest score will count towards the competition. We will show the current leaderboard live in class.

To submit a system, send your output file with predictions (1 prediction per line) to this Dropbox folder:

`https://www.dropbox.com/request/a5oMH7HIHp8AlhwlY2dl`

There is no need to make an account, but you have to add your name (we will show this on the leaderboard) and your email. Please name your submission `model_{run_number}` (no extension). Dropbox automatically uses your name to rename the file (yes really), so no need to add that in the filename already.

To remind you, here are some things you can experiment with:

- Number of epochs
- Learing rate

- Batch size
- Number of layers
- Number of nodes of the hidden layers
- Activation function
- Optimizer function
- Loss function
- Addition of dropout (and the percentage)

Obviously, do not use any algorithms from previous week, or any other functionality we have not discussed in class (yet). You might want to check out the Keras documentation to get up to speed.

### 4.1.2 Code and Answering Questions

Of course, you can work on the assignment longer than just during the lab. You can even try to optimize your model a bit more, potentially trying different settings or architectures. For the assignment, you have to answer these questions related to neural networks:

**NN-1:** First things first: what are the majority class baseline scores? Given these scores, do you think your model reached a good performance?

**NN-2:** Please explain in detail what dropout is and how it works. Did you experiment with using it? Where did you add it in your model? Did it improve performance?

**NN-3:** How did you decide when to stop training (a single model)? Do you think this worked well? Do you think there could be better methods? Why is it important to not train for too long?

**NN-4:** With what hyperparameters did you experiment? What did you find to be the most important ones? How did you try to find the most optimal values?

**NN-5:** Please describe your final model and how exactly you arrived there. Be specific! Describe both the architecture (number of layers, dropout, activation) as well as the hyperparameter settings (learning rate, number of layers/nodes, dropout percentage). If some settings lead to improvement, please also mention specific accuracies.

**NN-6:** Look at a number of mistakes your model made on the test set (your own one). What were the most surprising errors the model made? Is there anything in particular that stands out to you? Do you think there's room for improvement?

Again, no need to structure it like a research paper, just answer the questions!

## Exercise 4.2 – Word Embeddings

In class, you were given examples of the power of word embeddings. For example, they are used to output the most similar words given a certain input word. Word embeddings can also do analogies, e.g. given the input *Paris France Berlin*, it will provide *Germany* as the top answer. In the following exercises, you'll have to explore this, and try to come up with interesting similarities and analogies.

The two tools, and the trained word embeddings, are available in `/net/shared/word2vec`. The word embeddings are in the file called `GoogleNews-vectors-negative300.bin`, the similarity tool is called `distance`, and the analogy tool is called `word-analogy`. To start the tool, just type: `./distance GoogleNews-vectors-negative300.bin`. This might take a short while to load, and requires at least 4.5 GB of free RAM.

This works on the LWP machines. If you want to work from your laptop, you can either login to karora (`ssh s1234567@karora.let.rug.nl`) or download and install the software yourself. Simply follow these steps, either on your laptop or karora:

```
git clone https://github.com/tmikolov/word2vec
cd word2vec
make
wget -c "https://s3.amazonaws.com/dl4j-distribution/GoogleNews-vectors-negative300.bin.gz"
gzip -d GoogleNews-vectors-negative300.bin.gz
```

For a Mac, you can check out installation instructions here:

`https://github.com/William-Yeh/word2vec-mac`

### 4.2.1 Similarities

If you try out some inputs in the distance tool, and look at the most similar words, you'll start to notice some oddities. If you type *Groningen*, for example, you'll see that it doesn't just return other Dutch cities, but also Dutch and foreign football teams, and a Dutch university. This is because *Groningen* is strongly associated with not just the city itself, but also its university and football team.

**WE-1:** for this exercise, try out a number of different words in the similarity tool, and see if anything unexpected comes up, similarly to the example described above. For at least 5 words, report the input word, what you expected to be the most similar words, and, if the results are different than expected, what might be the cause of that. You don't need to give the full list of most similar words, just the interesting ones is sufficient. Interesting words to try are ones which have multiple meanings, or which have strong connotations.

### 4.2.2 Novel analogies

In class, we saw that word embeddings can capture certain relations. For example, it can capture the male-female relation, by coming up with the vector for *queen*, given the input *man woman king*. The same goes for countries and their capitals, and countries and their demonyms.

**WE-2:** in this exercise, try to come up with 3 **new** relations that word embeddings might be able to capture, and test these out using the analogy tool. Give the relations you tried to capture, 3 test cases (e.g. for the male-female analogy, try king-queen, actor-actress and monk-nun), and whether they were successful or not. If the relation was not captured successfully, which is often the case, try and come up with a possible explanation. Of course, try to have at least 1 original and succesful analogy!