Übung 4: RNNs - Report

Dataset

For this task, I decided to use the complete collection of Shakespeare's work, which I found on the following GitHub repository: `https://github.com/cedricdeboom/character-level-rnn-datasets`. The corpus contains, in one dataset (compiled from the repository's owners), all plays of Williams Shakespeare present on the Project Gutenberg's website, one after the other, in random order. The dataset was originally thought for a character-lever RNN training, but it seems that could work well for language modeling, too. For this reason, after that I gave a look to the entire file, it seemed to me that its structure could work well for the my training (e.g.: the fact that every poem is clearly marked and divided from the previous and next ones; the name of the speaking character/person is signalized; one verse is equal to one text line; no special characters is present; no numbers; …). The dataset's dimension is 4.4 MB, and it contains 156'771 tokens.

First training

The first training was ran with default parameters, and resulted in a perplexity value of 110.39.

Hyperparameters

I tried to change the following hyperparameters: number of epochs, batch size, vocabulary size, number of hidden layers. Furthermore, I wanted to implement an early stopping mechanism as described on this website: `https://chrisalbon.com/deep_learning/keras/neural_network_early_stopping/`. Before going farther, a premise as to be made. A good tuning relies on a great number of runs and attempts, after which one can check his results and, if it is the case, try to change something; and then re-do the training and re-check again, and so on after a satisfactory result is achieved. I could not proceed this way since I didn't had too much time and resources (i.e. I wasn't sure about how many credits were spent after every training step, and I already had to ask Mathias for extra money…), so I just tried to use a "one shot try" to see if I was able to improve my results (however I'm aware that in a real life situation it can be possible too that one has only limited resources to work with).

Coming back to our discussion, the number of epochs defines "the number of times all of the training vectors are used once to update the weights. For batch training, all of the training samples pass through the learning algorithm simultaneously in one epoch before weights are updated" (`https://nl.mathworks.com/matlabcentral/answers/62668-what-is-epoch-in-neural-network`). Like almost every parameter, it is not said that the greater the number of epoch and the better the results we'll achieve: too less epochs will drove to an underfitting danger, but a too big number could mislead the network to be too much attached to a determined "text structure", too much similar to the original one. Since in my sample created by `romanesco` after the first training (where the default number of epoch was 10) I still got some mistakes in the poems structures (e.g.: paragraph endings with comma and not a period), I will try with a higher number of epoch, i.e. 15.

The batch size "controls how often to update the weights of the network" (`https://machinelearningmastery.com/tune-lstm-hyperparameters-keras-time-series-forecasting/`). I wasn't able to find out the batch size of the first training, but I saw on GitHub that for the new version of `romanesco` Mathias set a default batch size of 64 for a vocabulary size of 10'000. Since I assume to work with a vocabulary dimension of 8'000, I'll set up this parameter to 50.

I downloaded the JSON file containing the vocabulary after the first training (`vocab.json`). I counted the total number of objects contained (10'001) but I wasn't able to sort this list following the numerical values, so that it would be possible to have an idea of how many tokens were for example under a threshold of 100 counts (or, better, 1'000). So I tried to assume that I will keep only 8'000 tokens, an the remaining 2'000 words I assume appear too rarely to be considered

I then lowered the hidden layer size from 1'500 to 1'000 in the `const.py` file. I wasn't sure about this parameter, because I'm not sure that – like in the case of the number of used hidden layers – a high value can better learn the language model but at the same time can result in a longer training time, with the risk of over fitting, too; I assumed that these points are partly valid for the hidden layer size, too. Moreover, on the Internet I read a quote where it is said that, as a rule of thumb, "the optimal size of the hidden layer is usually between the size of the input and size of the output layers" (cited on: `https://gist.github.com/DavidIsrawi/6c45744c12a4f8fc08bd5b8f7f9e06d8`); if it were so, will it mean that I need to higher the size, assuming that the input layer is equal to the vocabulary size (i.e., a dimension of 8'000). As I have already said, since I wasn't sure I preferred to lower the value to 1'000.

Second training

The second training achieved a perplexity value of 97.84 on the dev set.