MÜ Übung 4 **Tannon Kew**
Training a Recurrent Neural Network

For this exercise I worked with a data set of President Trump's tweets. The motivation for choosing this dataset was seeing an example of an automatic tweet generator, trained on the same data. With this exercise, I wanted to see if romanesco could generate similar language after training.

The dataset contains 33,491 tweets from the @realDonaldTrump twitter account. In order to pre-process the data for training the RNN, I first tokenized the dataset at sentence level (producing 62,364 sentences) and then at token level. For the first process I used NLTK's sentence tokenizer and for the second I used NLTK's TweetTokenizer, which is designed specifically to handle difficult tokens which are commonly used on Twitter, such as '@username' and web addresses. During the pre-processing, consideration was also given to normalising the dataset and removing problem characters such as Emojis or implementing truecasing, however, given the nature of the dataset, and the role played by Emojis and all-caps on twitter, I decided not to do these pre-processing steps. The dataset was then split into training, dev and test sets (90% – 5% – 5%). This resulted in a training set of 56,128 sentences, and 3,118 in the dev and training set.

In training the RNN, I attempted many separate trainings, each time adjusting small parameters to see how the results differed. As a first step, I used separate training and test set of Trump's tweets, split at 90% – 10% and trained a model with the default parameters to see how the system performed on the data. This baseline reached a perplexity of 45.66 in training and scored 77.28 on the test set. My aim was then to make adjustments in order to improve on these scores.

To do this, I first experimented with adjusting the learning rate from 0.0001 to 0.001 in order to speed up the training process, however this did not improve on the baseline results. After abandoning adjusting the learning rate, I reduced the size of the hidden layer from 1500 to 1000, as common sizes for hidden units range from 512 to 2560[1] and I felt that the original size was rather large for the size of the dataset considering the hidden layer size doubles as the embedding size. Additionally, I reduced the batch size, in order to allow for more iterations within each epoch. Here, I used a batch size of 8, which is evenly divisible by the number of sentences in my dataset.

In order to find the optimum performance and avoid overfitting, I added an early stopping feature to the romanesco code. This feature utilised the dev set to get a perplexity score after each iteration on the training data. If the perplexity started to increase, it would exit the training early. Here, I found it necessary to set the threshold of increasing perplexities to 3 in a row. Testing showed that a higher threshold was less effective in stopping the training, as the dev set perplexity fluctuations measured were so miniscule and would reset the counter before reaching the threshold.

As a result, with these final parameters and the implementation of early stopping, the model reached a perplexity of 51.77 in training after exiting due increasing scores on the dev set after 13 epochs. However, despite the changes made, this model only scored a perplexity of 77.69 on the test set, 0.41 above the baseline.

[1] https://svail.github.io/rnn_perf/