Tutorial 09: Textgeneration with RNNs (20P)

The goal of this exercise sheet is to implement a text generator, as a basic example of natural language processing. We will thereby explore more recent RNN architectures including long short-term memory (LSTM) cells and gated recurrent units (GRUs).

Please submit your code and results by 06th Jan 23:59 to manuel.milling@informatik.uni-augsburg.de. You can submit your solutions alone or in Teams of 2 (please indicate all names with the submission).

Idea and Data Set

The text generator will be based on Nietzsche's book "Beyond Evil and Good" and will always try to predict one character based on the 40 previous character's in the text. The possibly occurring characters are assigned an ID and the prediction of characters is based on a classification task over these IDs. As there is an obvious temporal dependencies in written language (The probability of certain characters or words depends on the previously occuring charcters or words) RNNs seem to be especially useful in this context.
The download mechanism of the text is already implemented.

1 Data Preparation (4P)

Assign an integer ID to every occuring character in the text (57 overall). Split the text into "sentences" with a length of 40 characters each and save the character after each sentence as a label for the sentence. Consider every 3rd character in the text as a possible starting character for a sentence, such that the data $\mathbf{x}$ is in a one-hot representation and has a shape of (200285, 40, 57) and the labels $\mathbf{y}$ in a one-hot representation have a shape of (200285, 57).

2 Model(4P)

Implement a RNN, which takes a sentence as a sequential input. The first layer should have an LSTM unit whose hidden state $\mathbf{h}$ has 128 neurons. The output of the LSTM layer at the last time step should be fed into a fully connected layer with a number of neurons equal to the number of character IDs and a softmax activation. Use the cross-entropy as the loss function.

3 Sampling (4P)

Implement a method **sample(preds, temperature=1.0)** that rescales a probability distribution and samples from the final probability distribution. For this purpose, take

the logarithm of the input probability distribution **preds**, before dividing it by the **temperature**. The final probability distribution is obtained by applying the softmax function. You can sample from this distribution using the **np.random.multinomial** method.
Note: The goal of this method is to sample characters from our predictions not by taking the character with the highest probability as often used for classification problems, but using probability based sampling. The temperature is used to stretch or squash the probability function.

4 Callback (4P)

Implement a method which is called after each epoch during training. The method should take a random sentence (40 characters) from the text and take this sentence as the basis to generate a text of 100 characters. In order to do so, run your network iteratively and update your input sentence after every step with your previous prediction. Run the generation for different temperatures of 0.2, 0.5, 1.0, 1.2 and 1.5. Print your generated texts (no interpretation or similar needed for the submission). Train your network for 20 epochs.

5 Network Parameters (2P)

How many trainable parameters does the neural network from exercise 2 have? Explain your solution!

6 GRU (2P)

Replace the LSTM unit from exercise 2 with a GRU with an equally big number of hidden state neurons and run your experiment again. How many trainable parameters does the new network have? Explain your solution.