Artificial Neural Networks

Prof. Dr. Sen Cheng

Dec 16, 2019

Problem Set 11: Recurrent Neural Networks

Tutors: Filippos Panagiotou (filippos.panagiotou@rub.de), Sandhiya Vijayabaskaran(sandhiya.vijayabaskaran@rub.de)

1. **Predicting time series data.** Use Keras' *SimpleRNN* layer to predict data points of a ramping sine wave.

$$f(x) = sin(x) + 0.1x \tag{1}$$

   (a) In Keras, recurrent neural networks require a specific input shape (batch size, time steps, input length). Write a function *prepareData* that takes the number of time steps as a parameter, and prepares inputs x and labels $y = f(x)$ with appropriate shapes. Generate a dataset of size $N = 1000$ (with x linearly spaced in the range $[0, 100]$).

   (b) Build a sequential model with a hidden *SimpleRNN* layer (32 units, ReLU activation) and a fully connected output layer (1 unit, linear activation). Use mean squared error (MSE) as your loss function and Adam as your optimizer.

   (c) Prepare your data using one time step as input x. Split the data set $70/30$ into training and test sets. Train the model for 20 epochs. Plot the prediction for training and test sets.

   (d) Calculate and plot the squared errors for each data point. How does the squared error depend on the phase of the sine wave? Why does it show this behavior? Are there MSE differences between training and test set?

   (e) Repeat the previous two tasks using three time steps instead. Does the MSE differ? How does the squared error depend on the phase of the sine wave?

2. **Learning long-term dependencies.** In this exercise, we will compare a simple RNN to LSTMs in a simple text generation task. Train a network on the text file containing Goethe's *Faust* to predict the *next character* given a sequence of characters, i.e., a character-level RNN. The model is only predicting single characters and does not have any knowledge that words are units of language, or that words are combined into sentences.

   (a) Load the text file in lowercase and print out the length of the whole text. To get an idea of the dataset, also print the number of individual characters present in the text after sorting them in order (use the default python functions `sorted` and `set`).

   (b) To prepare the data, create two dictionaries that map characters to integers and vice-versa. Next, prepare the training data: split the data into sequences of length 50 that start at every third character of the text. The labels are the next character after each sequence.

   (c) To encode the characters use one-hot encoding. Use the dictionaries from the previous step to generate the one-hot encodings of the labels and also of the input sequences. That is, each input sequence is a sequence of the one-hot encodings of the individual characters in that sequence.

   (d) Build a Sequential model using an LSTM layer with 256 units as the hidden layer followed by an ouput Dense layer. How many units should the ouput layer have? What are the activation function and loss function that you would use, and why? Use Adam as the optimizer again and compile.

(e) Callbacks are a useful function in Keras that show relevant aspects of the model during training. The file `print_callback.py` provides a keras callback that prints the network output for a random sample after every epoch. Pass this callback to the fit function to see how the model does after every epoch. Train the model for 20 epochs. Save the weights after training. (Note : There is no train/test split. We train the model on the entire dataset to learn a probability distribution of characters in a sequence.)

(f) Test the model using a random input sample and look at the next 500 characters generated (approximately a paragraph). Look at the callback function provided for hints on how to do this. Analyze the features of the generated text.

(g) Repeat the training and test after replacing the LSTM layer with a SimpleRNN of 256 units. What are the qualitative differences you notice in the text generated by the two models?