Project 2 Part 2: Using Recurrent Neural Networks for Regression

Marcus Brenscheidt
Heinrich Heine University
marcus.brenscheidt@uni-duesseldorf.de

Bastian Berndt
Heinrich Heine University
bastian.berndt@uni-duesseldorf.de

Tobias Uelwer
Heinrich Heine University
tobias.uelwer@uni-duesseldorf.de

Abstract

This paper will show Recurrent Neural Networks (RNNs) used for regression on real world time series data. We will apply it to two different examples.

1 Introduction to the data and RNNs

Both datasets will be split into training and test data, with the test dater always beeing the last 10% of the data.

1.1 Dataset 1

The first dataset we use is simply a time series of the function $x \cdot \sin(x)$ consisting of 1000 data points evenly distributed over [0,1] as x and their corresponding functionvalues as y.

1.2 Dataset 2

The second dataset is a real world time series. It contains 468 monthly observations on athmosperic CO2 concetrations expressed in parts per million (ppm) and reported in the preliminary 1997 SIO manometric mole fraction scale from the Mauna Loa, a volcano on Hawaii. They range from the year of 1959 to 1997. This dataset is interesting if one wants to predict an volcanic eruption.

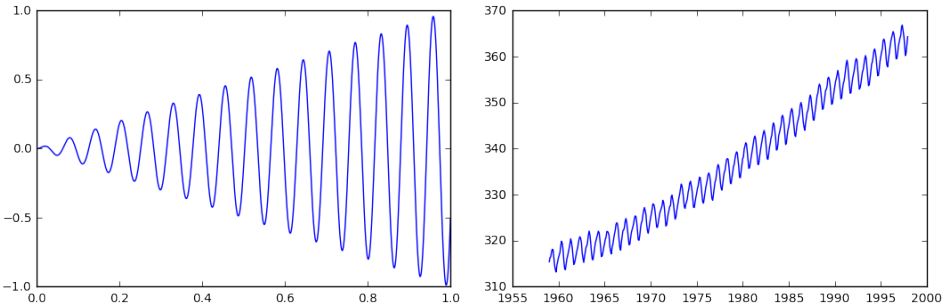


Figure 1: Left: x sinx; Right: CO2 at Mauna Loa

1.3 On RNNs

Recurrent Neural Networks are networks making use of sequential information by working in timesteps. We assume that inputs are dependent on previous ones, like the next word in a sentence depends on the ones before it. The main architecture used is the LSTM (Long Short-Term Memory). It consists of multiple stacked LSTM cells, each using information from previous cells to compute its own output and hand it to the next one, such that information is prolonged in the network. Additionally to that there is a cell state, which chains through all LSTM cells, each cell being able to add or remove information from the cell state. This way even longterm dependecies can be learned by the network.

2 Architecture and Hyperparameters

This section has already been subject in the previous report and will be done shorter here.

2.1 Architecture

The architecture is pretty simple. We basically use only LSTM cells on both datasets, even though we have the possibily to add dense layers at the end of the architecture, because we found they don't really give any additional accuracy. For the number of stacked LSTM cells we chose 10 for the first one and ??????????? for the second one. Adding to many cells leads to overfitting, so we rather choose less. As optimizer we use Adagrad and as loss the Squared Error.

Insert Visualization of Architecture here!

2.2 Effect of Iteration Count

As long as we didn't converge, more itearations means we get closer to the desired minimum of the loss function. We found that 20000 training steps were fine for the first dataset.

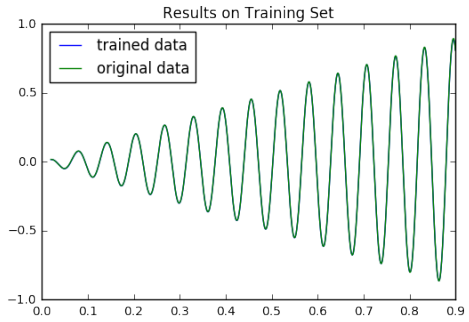
2.3 Effect of Timesteps

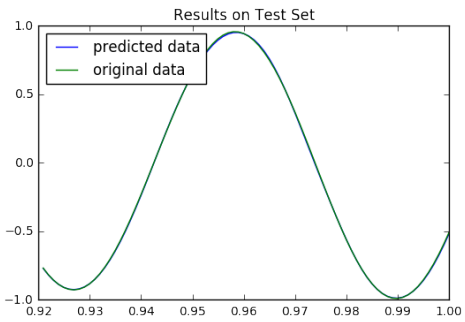
As timesteps we consider the number of datapoints we are able to look back at. This behaves similar to the number of stacked cells, making this too high might make us look at data, that is mostly independent from what we're looking at right now, while choosing it to low, removes valuable information. Another important aspect is the increase in computation time. Obviously looking at more data takes us longer to train the network.

3 Performance

3.1 Dataset 1

On the first dataset we achieved mean squared errors of around 0.000030 on the test data and <0.000003 on the training data. Here are some plots showing how close we got with just few seconds of training time:





3.2 Dataset 2

Missing