Recurrent Neural Networks

Introduction

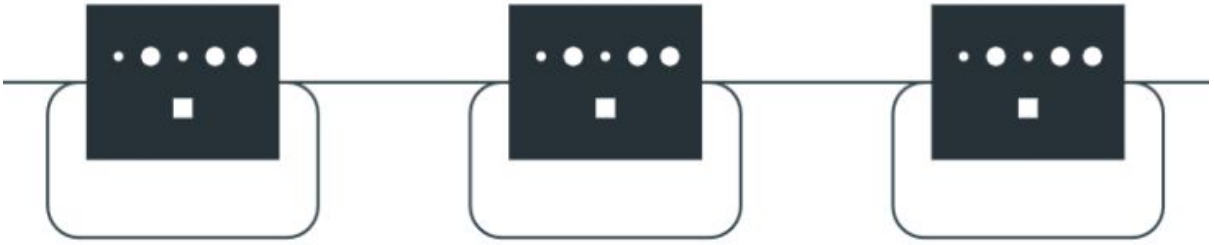
With this exercise, you will learn about recurrent neural networks (RNNs). In contrast to fully connected or convolutional neural networks, RNNs operate on sequential data. Thus, they are used for tasks such as translation, image captioning, and next element prediction. Looking at a text-prediction task, one objective for the network could be to continue a text snippet given just a part of it. This way, one can generate diverse text snippets from a given starting point.

Input

The input to a recurrent neural network can be sequential data. The following examples will be based on periodic functions and simple text snippets. For other tasks, such as image captioning, which does not include sequential input data, the network only gets an input value for the first time step, and then has to recurrently build a caption by outputting one word at a time, which, again, makes it a sequential task.
This way, recurrent neural networks can be used for tasks with sequential input and non-sequential output (many-to-one), such as next element prediction. Tasks with sequential input and sequential output (many-to-many), such as translation. As well as tasks with non-sequential input and sequential output (one-to-many), such as image captioning.
As the general concepts for these tasks do not differ, we focus on next element prediction in this application document.

Network

A recurrent neural network is built up in a similar way as convolutional or fully connected neural networks, where multiple layers are connected to extract information from the input. The great speciality of recurrent neural networks is that they also use information from previous time steps for their predictions. This is visualized by loops in their network glyphs.
To achieve this, they have weights not only for how information is processed throughout layers, but also for how information from previous time steps is processed. The network first builds up its internal state before making a prediction. To make further predictions, as when generating text or continuing a function, one can feed the network its own prediction values, to again build up the internal state with the same number of time steps as at the beginning, where known datapoints were used.
During trianing, this means that the error is not only backpropagated through the layers, but also through time, to update the weights responsible for determining how information from previous time steps is processed. This can be thought of as the error flowing backwards through the loops in the network, as well as through the lines connecting the layers.



Training

Training Steps

Forward Direction

To teach the network how to continue a sequence, one needs to first give the network data it can learn from. The network then makes predictions based on this data, suggesting how it currently thinks, the sequence would continue.

Validation Step

After a prediction is made on the training data, we can calculate the prediction error. This is done by comparing the prediction with what would actually be correct. This is also referred to as comparing the prediction to the ground truth. This difference is then used as the loss, which is important for the backward step.

Backward Direction

After making predictions and calculating the loss, the network weights get updated. This is done by backpropagating the error through the network, and for RNNs, also through time. Here, the network learns, how weights for individual layers (backpropagation through network), but also for individual timesteps of the input function (backpropagation through time), should be changed to the predictions.

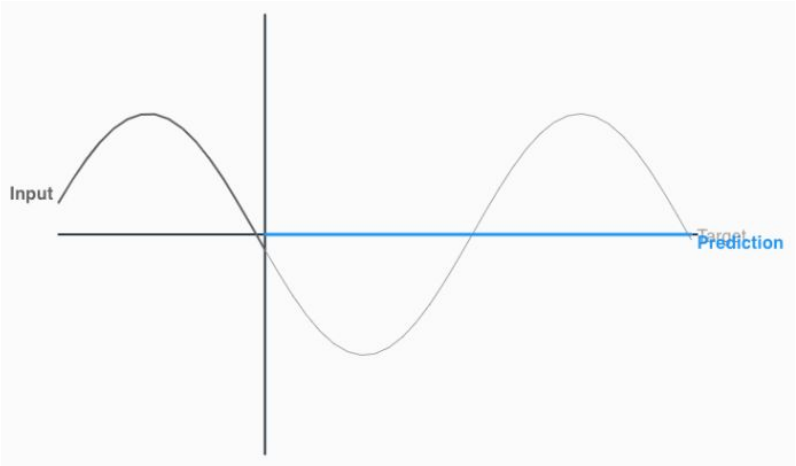
Parameters

As with other types of neural networks, there are different parameters that can be manipulated during training. In the following, you will learn how a few of these parameters can influence the training success for the example application of function continuation.

Learning Rate

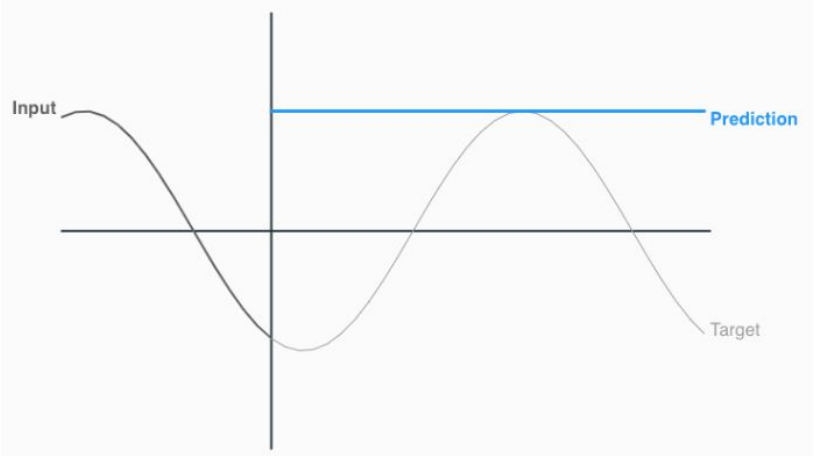
Low Learning Rate

When the learning rate is too low, one can see that the model prediction is not good, even after 20 epochs. Indicating a learning process that is too slow.



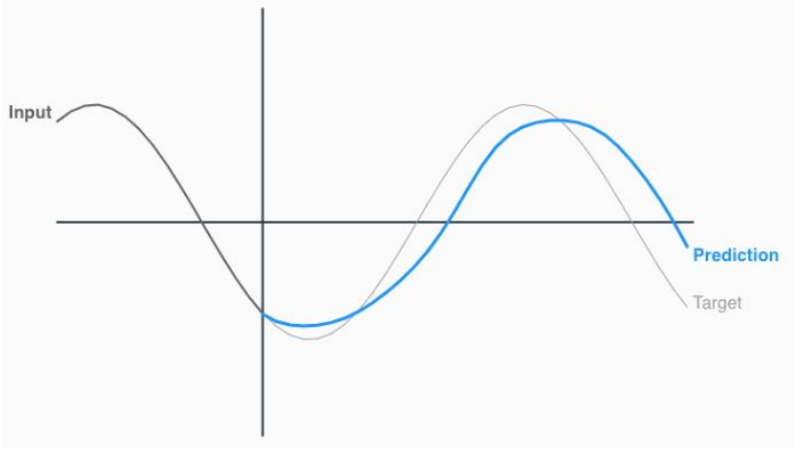
High Learning Rate

By contrast, when the learning rate is too high, the model prediction is also not good after 20 epochs, indicating overshooting of optima during training.



Medium Learning Rate

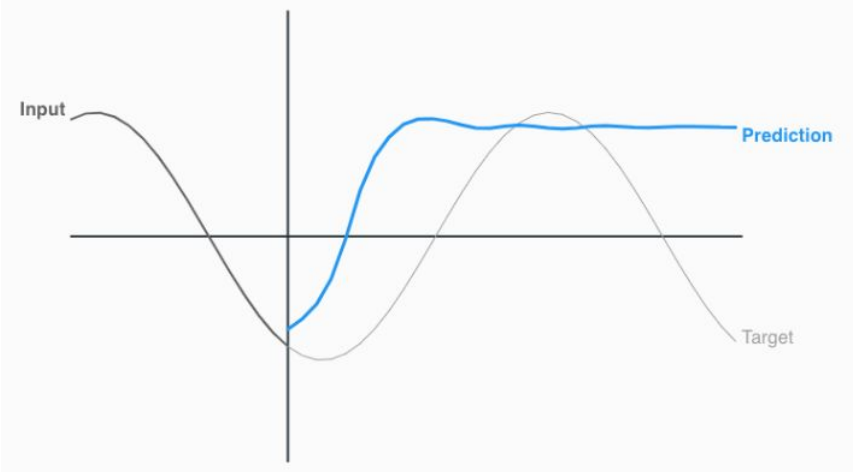
When the learning rate is selected appropriately, the model finds an optimal solution fast, even after 10 epochs.



Batch Size

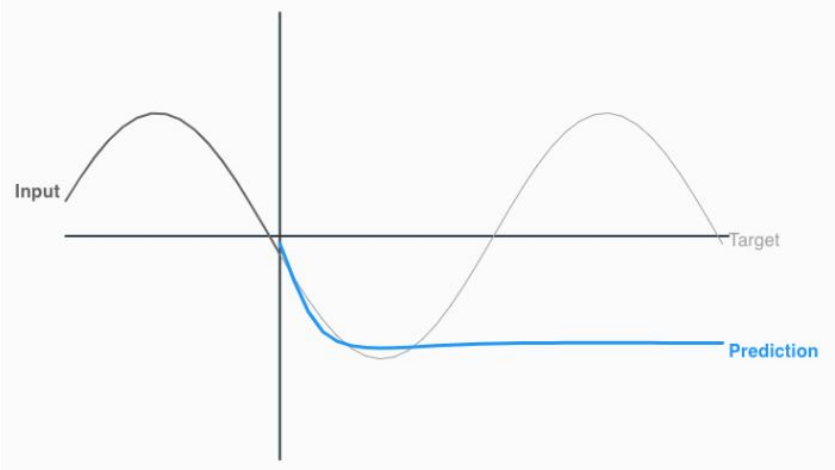
Low Batch Size

When the batch size is low, the model is learning slowly, not being converged even after 20 epochs.



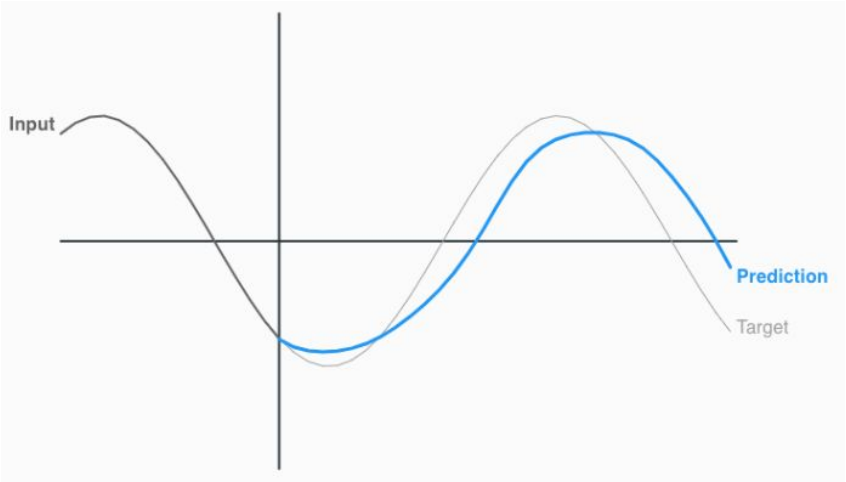
High Batch Size

By contrast, when the batch size is too high, the loss is averaged over too many samples, thus, also not converging after 20 epochs.



Medium Batch Size

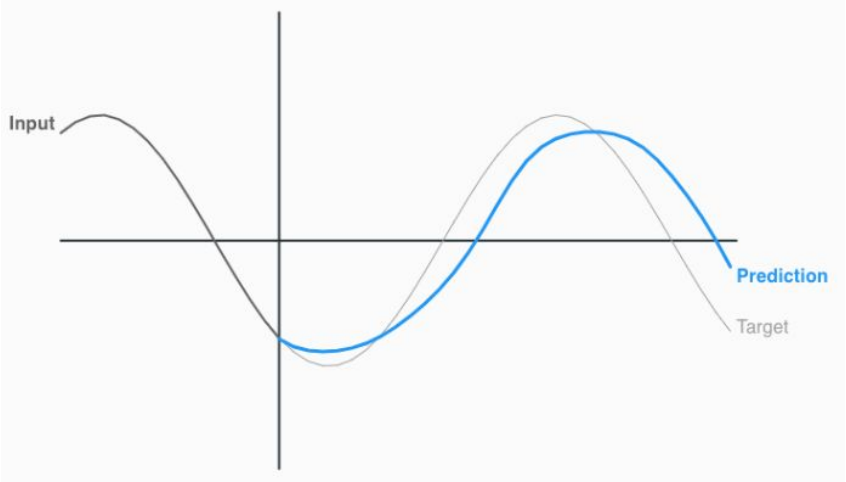
Again, when the batch size is selected appropriately, the model finds an optimal solution fast, even after 10 epochs.



Noise

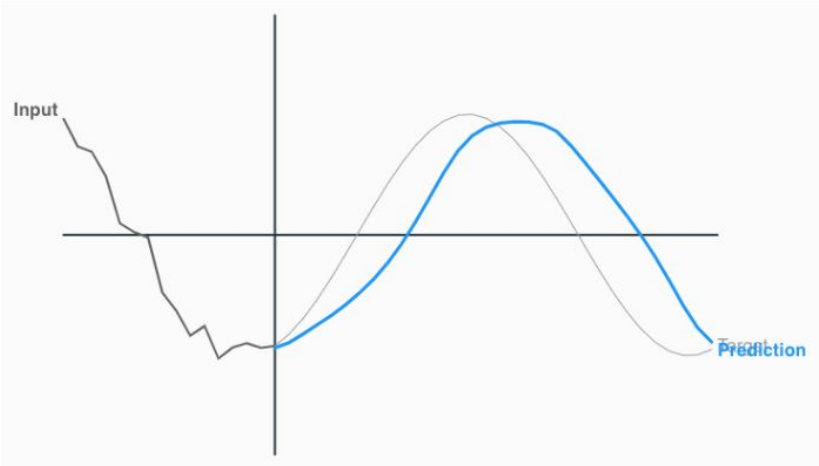
No Noise

Without noise in the training data, it is easy for the model to learn. However, this is not a very realistic scenario, as real-life data is often not as smooth.



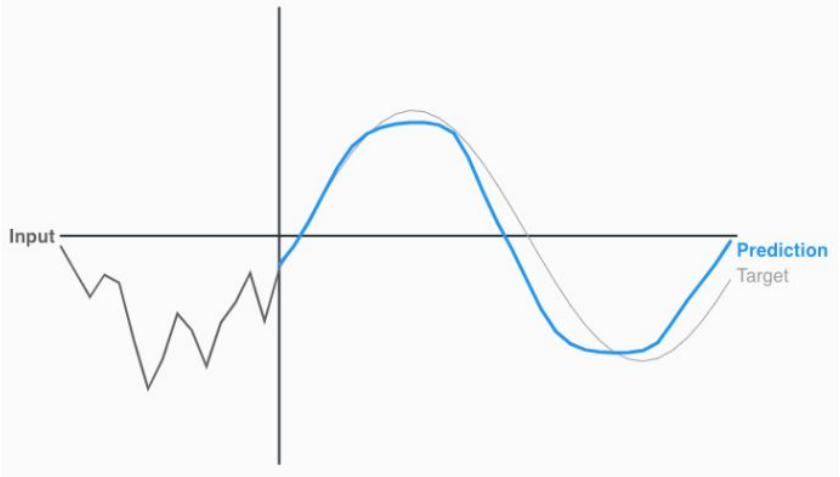
Some Noise

Adding noise reflects a more realistic scenario. As this is harder to learn, it takes longer for the modele to converge.



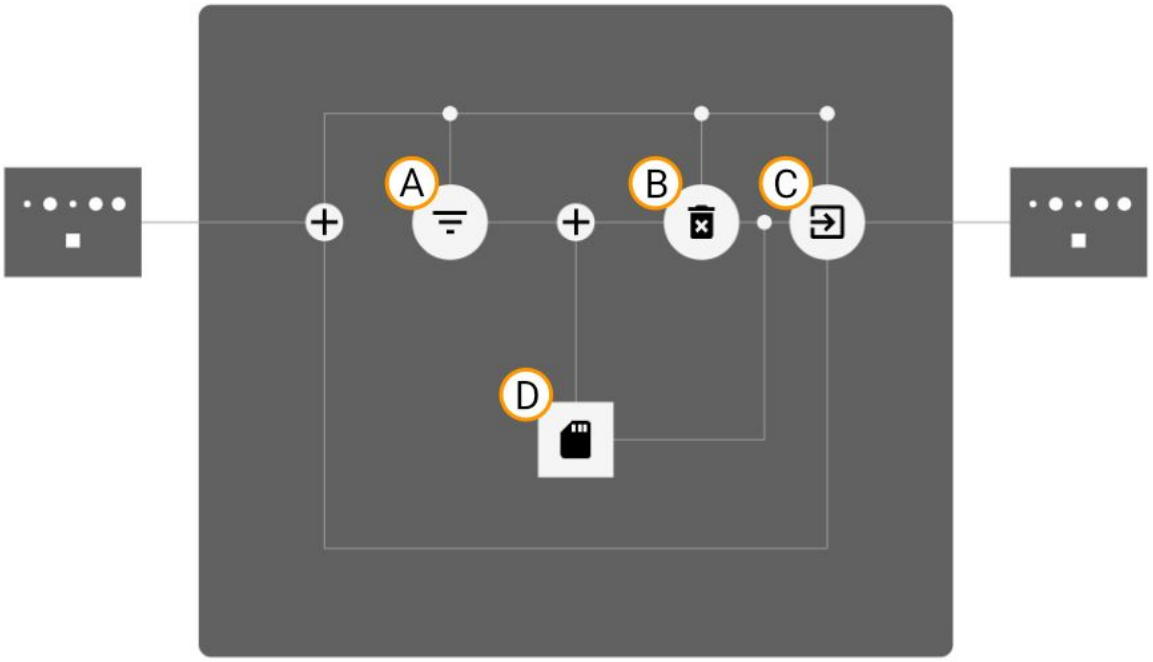
High Noise

Noise can also be too much for the model to handle. When the noise is too much, the model might not converge at all.



LSTM Cells

An LSTM cell consists of different components, which process the data that is fed into the network. We will explain these components in the following.



Data for a cell can come from previous layers, or directly be the input to the network. Upon arrival of new data, this data is combined with the previous activation of the cell for further computations.
The aforementioned combination of input data and previous activation is then fed to the three gates that a LSTM cell has. They are called Input Gate (A), Forget Gate (B), and Output Gate (C). All of these Gates also interact with the Cell State (D).

Cell Elements

Cell State

The cell state is the heart of any LSTM cell. By having a cell state, and deciding how to update it based on the filtered input and previous state, LSTM cells are able to capture long-term dependencies.

$$c^t = \text{filtered\_input} + \text{filtered\_state}$$

Symbols:

$$c^t : \text{the cell state at timestep t}$$

Input Gate

The input gate is used as a measure for what part of the input is used for updating the cell state with new information. It only lets through information that might be helpful for future predictions of the cell.

$$i^t = sigmoid(W_{ix}x^t + W_{ia}a^{t-1} + b_i)$$

It is used to calculate the filtered input to update the internal cell state.

$$\text{filtered\_input} = i^t \circ tanh(W_{cx}x^t + W_{ca}a^{t-1} + b_c)$$

Symbols:

i^t : the input gate

x^t : the input from the previous cell at time step t

a^{t-1} : the activation of this cell from the previous time step

W : trainable weight parameters

b : trainable bias parameters

Forget Gate

The forget gate determines what part of the old cell state can be forgotten.

$$f^t = sigmoid(W_{fx}x^t + W_{fa}a^{t-1} + b_f)$$

It is used to calculate the filtered state to update the internal cell.

$$\text{filtered\_state} = f^t \circ c^{t-1}$$

Symbols:

f^t : the forget gate

x^t : the input from the previous cell at time step t

a^{t-1} : the activation of this cell from the previous time step

c^{t-1} : the cell state from the previous time step

W : trainable weight parameters

b : trainable bias parameters

Output Gate

The output gate is responsible for filtering what part of the current cell state is used as the output activation of the cell.

$$o^t = sigmoid(W_{ox}x^t + W_{oa}a^{t-1} + b_o)$$

Using the output gate, one can then compute the cell activation, which is used as input to the next cell, as well as recurrently in the same cell at the next time step.

$$a^t = o^t \circ tanh(c^t)$$

Symbols:

o^t : the output gate

a^t : the activation of this cell for the current timestep

x^t : the input from the previous cell at time step t

a^{t-1} : the activation of this cell from the previous time step

c^t : the current cell state

W : trainable weight parameters

b : trainable bias parameters

Cell Steps

The functionality of a cell can roughly described in four steps that explain how the gates and memory is working.

Layer Input

To inform the gates of the cell about which information is relevant to be integrated into the cell state, what aprt of the previous cell state can be forgotten, and what part of the cell state is output as the cell activation, LSTM cells use both, the input to this cell, and the activation of this cell from the previous time step. This combined input information is referred to as the layer input.

Gate Activation

Using the information that is combined as the layer input, all three gates use their own weights and biases to compute the gate activation, which determines what part of information is used for updating the cell state and calculating the cell activation. The input gate filters what part of the layer input is used to update the cell state with new information. The forget gate filters what part of the cell state can be forgotten. The output gate filters what part of the cell state is used to compute the cell activation.

Cell Update

Using the filters of the input gate and the forget gate, the cell state is updated with a mix of new information and information from the old cell state. The input gate is used in combination with the layer input to determine what new information gets added to the cell state. The forget gate is used in combination with the previous cell state to determine what information from the previous cell state can be forgotten.

Output

Finally, the cell needs to output an activation value, which can be used as an input to the next cell, or as a prediction output.It is also used recurrently as an input to this cell for the next time step. The activation is computed as filtered information from the cell state. Here, the output gate is used for this filtering.