LSTM and Recurrent Neural Nets

Mario Dohr

October 25, 2022

1 Simple Recurrent Networks

In contrast to a simple feedforward network a recurrent network has a loop. That is, in a RNN the output of a previous timestep is again feed into the network. This enables RNNs to "remember" past inputs and makes them more suitable for processing time series than simple FNNs. Accordingly RNNs map an input sequence $(x(t))_{t=1}^{T}$ to an output sequence $(\tilde{y}(t))_{t=1}^{T}$.

In principle FNNs can also process sequences of size T, but this requires T to be constant.

By contrast RNNs can process arbitrarily long sequences with constant parameters and can memorize information over long distances. RNNs allow for *temporal generalization*. That means that learning to store important information at a time step can be generalized to store this information also if it appears at time steps that are never seen during training. For example an important input that is seen in the training set at time step 5,7,8,9 but never at time step 6, is generalized by RNNs to time step 6 while feedforward networks fail.

1.1 Jordan Network

In a Jordan network the output $\tilde{y}(t-1)$ at time stept $t-1$ is fed back as input at time t.

$$s(t) = W^T x(t) + R^T \tilde{y}(t)$$
$$a(t) = f(s(t))$$
$$\tilde{y}(t) = g(V^T a(t))$$

where W are the input weights, R the recurrent weights and V are the output weights, which are all collected in the parameter w. g and f are activation functions.

W, R and V does not depend on the time step. We use the same weights at all time steps this is called *weight sharing*.

1.2 Elman Network

The Elman network is a modification of the Jordan network. It feeds in the hidden units of time step $t-1$ as input to time step t.

$$s(t) = W^T x(t) + a(t-1)$$
$$a(t) = f(s(t))$$
$$\tilde{y}(t) = g(V^T a(t))$$

1.3 Full recurrent network

Is a modification of the Elman network. We add parameter R for the recurrent parameters.

$$s(t) = W^T x(t) + R^T a(t-1)$$
$$a(t) = f(s(t))$$
$$\tilde{y}(t) = g(V^T a(t))$$

1.4 other types

ergänzen

- ARMA
- NARX
- Time delay

1.5 Summary

- RNNs implement loops to memorize information over time. The architectures differ in which value (hidden state, output) from the previous time step is fed in in the next time step and where is it fed in.
- The hidden states contains more information than the output
- Look at the different abstraction levels of the inputs and the units they are connected to for the recurrent layer.

2 Learning algorithms for RNNs

2.1 Empirical riskminimization via gradient descent

As in FNNs we use gradient descent to minimize the empirical risk.

$$R_{emp}\Big(\{y(1:T)^{(n)}, \tilde{y}(1:T)^{(n)}\}_{n=1}^{N}\Big) = \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T} L(y(t)^{(n)}, \tilde{y}(t)^{(n)})$$

$$= \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T} L(y(t)^{(n)}, g(a(0), x(1:t)^{(n)}; w))$$

where $y(1:T)$ is a sequence of lenth T.

2.2 Backprobagation through time (BPTT)

ergänzen

2.2.1 BPTT for the fully connected network

The gradient $\nabla_w L$ decomposes into three parts, according to the weight matrices W, R and V.

We start with the output weights. v_{ik} connects the $i-th$ input unit with the $k-th$ output unit, because we multiply with V^T. So we get

$$\frac{\partial L}{\partial v_{ik}} = \sum_{t=1}^{T} \frac{\partial L(y(t), \tilde{y}(t))}{\partial \tilde{y}(t)} \frac{\partial \tilde{y}(t)}{\partial v_{ik}} = \sum_{t=1}^{T} \frac{\partial L(y(t), \tilde{y}(t))}{\partial \tilde{y}_k(t)} \frac{\partial \tilde{y}_k(t)}{\partial v_{ik}} \tag{2.1}$$

The second equation comes from the fact, that $\tilde{y}_k = v_i \cdot a$, where v_i is the $i-th$ row vector of V^T and a is an activation of the last layer.

The term $\frac{\partial L}{\partial \tilde{y}_k(t)}$ is the derivative with respect to the $k-th$ component of the model output. We will call this term $e_k(t)$. To calculate this term we only need to know the loss function. If we choose the halved squared error we get the following:

$$L(y(t), \tilde{y}(t)) = \frac{1}{2}\|y(t) - \tilde{y}(t)\|^2 \text{ then } e_k(t) = \frac{\partial L}{\partial \tilde{y}_k(t)} = \tilde{y}_k(t) - y_k(t). \tag{2.2}$$