EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Computer Science Department
Neuro-Cognitive Modeling

Summer Term 2022

Recurrent and Generative Neural Networks

Exercise Sheet 02

Release: May 13, 2022 Deadline: June 2, 2022 (23:59 pm)

General remarks:

- Download the file `exercisesheet03.zip` from the lecture site (ILIAS). This archive contains files (Java classes), which are required for the exercises.
- All relevant equations can be found in the corresponding lecture slides. Ideally, the exercises should be completed in teams of two students. Larger teams are not allowed.
- **Add a brief documentation (pdf) to your submission**. The documentation should contain protocols of your experiments, parameter choices, and a discussion of the results.
- When questions arise start a forum discussion in ILIAS or contact us via email: Sebastian Otte (`sebastian.otte@uni-tuebingen.de`)

Exercise 1 - Recurrent Neural Networks [50 points]

(a) Implementation Back-Propagation Through Time [20 points]

Analogously to ExerciseSheet 01, there is a class `RecurrentNeuralNetwork`, which gives a implementation skeleton of a recurrent neural network (RNN). Implement the method `backwardPass` accordingly. Note that you now have to compute the gradient for a sequence (of length T) and not only for a single pattern. First, compute all

$$\delta_j^t := \frac{\partial E}{\partial net_j^t} \tag{1}$$

with $\delta_h^{T+1} := 0$. You can check the already implemented method `forwardPass` to understand how to handle the sequence buffers. Then, for each weight compute the respective

$$\frac{\partial E}{\partial w_{ij}} = \sum_{t=1}^{T} \begin{cases} x_i^t \delta_j^t & \text{if } (i,j) \text{ is a feed-forward connection} \\ x_i^t \delta_j^{t+1} & \text{if } (i,j) \text{ is a recurrent connection} \end{cases} \tag{2}$$

The code of the gradient descent (method `trainStochastic`) can be reused from Exercise 1 with minimal effort.

Hint: Do not forget to call the method `reset` (which zeros the activities and deltas) at the right time.

(b) Trajectory Generation [20 points]

In the class `RNNTrajectory` you can find a prepared experiment for a trajectory generation, which can be used to test your implementation. Again, find suitable parameters but don't use biases here. Document your investigations and add a screenshot of a successful training run.

Tricky (bonus): Adapt your implementation such that you can force a trained RNN by a specific initialization of the hidden activations (with no external input) to start its output sequence from an arbitrary point $p \in [-1, 1]^2$. Hint: Think sequentially and consider that BPTT can also be used to determine, which activations are required to produce a desired output.

(c) Gradient Measurement [10 points]

In this task you will identify a significant problem of RNNs empirically. Set up an RNN with one input, one hidden, and one output neuron (no biases). Initialize the weights randomly with std. dev. 0.1. Create an input sequence of 100 time steps in which each value is 1.0. The learning target is just a single 1.0 presented at the very last time step (do not use 0 as target for the remaining time-steps).

Input:	1	1	1	1	1	$\cdots$	1	1
Target:								1

First, call `forwardPass` and then `backwardPass`. Investigate the evolution of δ_h^t reversely in time. Discuss your observations and indicate possible consequences for sequence learning with RNNs.