EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Computer Science Department
Chair of Cognitive Modeling

Winter semester 2019/2020

Advanced Neural Networks

Exercise sheet 01

Release: October 24, 2019 Deadline: November 14, 2019

General remarks:

- Download the file `exercisesheet01.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.
- When questions arise start a forum discussion in ILIAS or contact us via email:
  Dania Humaidan (`dania.humaidan@uni-tuebingen.de`)
  Matthias Karlbauer (`matthias.karlbauer@uni-tuebingen.de`)
  Sebastian Otte (`sebastian.otte@uni-tuebingen.de`)
  Mahdi Sadeghi (`mahdi.sadeghi@uni-tuebingen.de`)

Exercise 1 Multilayer Perceptrons [50 points]

(a) Implementation Back-Propagation [30 points]

The mentioned archive contains the class `MultiLayerPerceptron`. This class is an incomplete implementation for multilayer perceptrons but already provides important data structures and such. Your task is to implement the method `backwardPass`, which computes the gradient based on a previously computed network activation. First, compute the δ_j for each neuron

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

and, afterward, the partial derivatives for all weights

$$\frac{\partial E}{\partial w_{ij}} = x_i \delta_j. \tag{2}$$

Further, complete the methode `trainStochastic`. This method should realize a stochastic gradient descent with momentum term. Follow the source code remarks.

(b) XOR [10 points]

Test your implementation using the class `MLPXOR`. Find suitable parameters (number hidden layer, hidden layer size, learning rate, momentum rate, bias on/off). The epoch error (the average Root Mean Square Error (RMSE) over all training examples) should be smaller than 0.01 within 10 000 epochs of training (an epoch refers to the one time processing of all training samples). Detail and briefly discuss the results of your investigations.

(c) Geometry [10 points]

Experiment with the class `MLPGeometry`. Again, find suitable parameters, such that the foreground points (two bubbles) are visually sufficiently separated from the background points. Detail and briefly discuss the results of your investigations and add a screenshot of a successful training run. Is this classification problem from a theoretical perspective solvable by a one-layered network (with input- and output layer only)? Give an explanation.

Exercise 2 Recurrent Neural Networks [50 points]

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

Analogously to Exercise 1, there is a class `RecurrentNeuralNetwork`, which give 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

$$\delta_j^t := \frac{\partial E}{\partial net_j^t} \tag{3}$$

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

$$\frac{\partial E}{\partial w_{ij}} = \sum_{t=1}^{T} \begin{cases} x_i^t \delta_j^t & \text{if } (i,j) \text{ is a feed-forward connection} \\ x_i^t \delta_j^{t+1} & \text{if } (i,j) \text{ is a recurrent connection} \end{cases} \tag{4}$$

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: 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.