Assignment 8

Welcome to the eighth assignment of the lecture *2D Vision and Deep Learning*. **Please read all instructions carefully!** This assignment covers some basic concepts of neural networks. Submission is due on Monday, January 18th, 2021 at 2pm. Please note that late assignments will receive zero (0) marks, so you are strongly encouraged to start the assignment early. If you have any questions please contact Tim Heußler (`theussle@students.uni-mainz.de`).

Exercise 1 (4 points)**.** Show that a multi-layer perceptron always collapses into a single-layer neural network, regardless of how many hidden layers it contains, if all activation functions are identities, i.e. if

$$f(x) = x$$

holds for all activation functions.

Exercise 2 (4 points)**.** Given the function

$$f(x, y) = \frac{(x-2)^2}{4} + (y-4)^2$$

1. (1 point) Draw some isolines of f and describe the *zero isoline* of f.
2. (1 point) You are at point $(x, y) = (2.4, 4.2)$. Along which direction can you climb up the function fastest?
3. (2 point) You are at point $(x, y) = (2.4, 4.2)$. Along which two directions could you move down fastest if the maximum allowed slope is 0.2?

Exercise 3 (2 points)**.** Suppose you have a neural network with 7 output neurons that provide the output vector $\mathbf{o} = (17, 4, 42, 15, 8, 32, 27)^\top$. Normalize the output vector into a probability vector using the softmax function.

Exercise 4 (2 points)**.** Use *PyTorch* to implement a neural network that fits $\cos(x)$ in the range $[-\pi, \pi]$ with a sixth-order polynomial.

Exercise 5 (4 points)**.** Use *PyTorch* to implement a MLP (`https://pytorch.org`) with *one hidden layer* to learn the MNIST dataset (`http://yann.lecun.com/exdb/mnist/`).

Exercise 6 (6 points)**.** Create a deep neural network with five hidden layers of 100 neurons, Xavier initialization and RELU activation functions using *PyTorch*.

1. (3 points) Use Adam optimization and L_2-regularization to train the network on the MNIST dataset. Save checkpoints at regular intervals as well as the final model so you can reuse it later.

2. (1 point) Tune the hyperparameters using *cross-validation* and see what precision you can achieve.

3. (1 point) Add *Batch Normalization* and compare the learning curves. Is it converging faster than before? Does it produce a better model?

4. (1 point) Is the model overfitting the training set? Try adding *dropout* to every layer. Does it help?