Tutorial 04: Fully Connected Neural Nets - Backpropagation (21P+7P)

The main goal of this week is to implement the backpropagation for last week's neural network. Further, we will write a training routine for the model on the already introduced MNIST database. The exercise will be based on and add to the previous forward propagation code of a fully connected neural network with two hidden layers (400 units and sigmoid activation each) and an output layer with 10 units and softmax activation for the classification of the MNIST data. You can either use your own code from Exercise 03 or the uploaded solution as a basis for this Exercise.
You can use Google Colab https://colab.research.google.com/ to upload and run your python notebooks.
Please submit your code by 19th Nov 23:59 to manuel.milling@informatik.uni-augsburg.de.
You can submit your solutions alone or in Teams of 2 (please indicate all names with the submission).

1 Error Term Last Layer: Cross-Entropy + Softmax (6P)

Implement a routine, which calculates the error term Δ^n of the last layer, i.e.,

$$\Delta^n_{jm} = \frac{\mathrm{d}L}{\mathrm{d}\hat{\boldsymbol{Y}}_{\boldsymbol{im}}} \frac{\mathrm{d}\hat{\boldsymbol{Y}}_{im}}{\mathrm{d}\boldsymbol{I}_{jm}}, \qquad i : \text{correctlabel}$$

for the cross-entropy loss function and the softmax activation function, given the output of the network at the last layer $\boldsymbol{H}$ and the correct labels $\boldsymbol{y}$.
Note: You can use the formulas in the handout for this exercise. The result should be a numpy array of shape (N_n, M)

2 Derivative of the Sigmoid Function (2P)

Implement a routine, which calculates the derivative of the element-wise sigmoid function in terms of the sigmoid function: **del_sigmoid(h)**, given the output **h** of the sigmoid function.
Note: You can use your solution from Exercise sheet 2 for this task. The result should have the same dimension as the input.

3 Backpropagation (6P)

Implement a class method that computes the gradient's components for all weights and biases, given the labels $\boldsymbol{y}$ for the training set, by repeatedly applying the recursive

formula for the backpropagation of the error Δ^i using the weights and the derivative of the activation function.
Note: Assume that the activation of all layers, as well as the weights and biases, are available as class variables from the forward propagation.

4 Optimisation step (3P)

Implement a class method which performs one Optimisation step for the network given a learning rate **learning_rate**, i.e. an update of the weights and biases by an amount of

$$-\text{learning\_rate} \cdot \nabla_{\boldsymbol{\Theta}} L$$

.
Note: Assume that the weights and biases, as well the components of the gradient are available via class variables.

5 Training Routine (4P)

Implement a training routine for the neural network by repeatedly calling the forward propagation, the backpropagation and the optimisation step for the training set (considering **all** training examples). Use 1 000 training steps (this can take up to a few hours) with a learning rate of 0.01. After each 100 training iterations print the cross-entropy loss **and** the accuracy on the training **and** the test set into a file. Your train and test accuracy should surpass 55 % during the training.
Note: You can try to increase the number of training iterations to 10 000 optimisation steps. This can take 12 hours and more but should push the results above 85%.

6 Bonus: Stochastic Gradient Descent (4P)

Familiarise yourself with the concept of stochastic gradient descent (mini-batch training) to increase the performance of your system. Use a mini-batch-size of 64 samples for each optimisation step and shuffle your training set after each epoch. Can you obtain good results faster?
Note: This exercise is a bonus, i.e., it is not necessary to solve this exercise to obtain 100 % on the sheet.

7 Bonus: Derivative of the Softmax Function (3P)

Show that the derivative of the softmax function for an input $\boldsymbol{x} \in \mathbb{R}^n$ can be written as:

$$\frac{\text{dsoftmax}(\boldsymbol{x})_i}{\text{d}\boldsymbol{x}_j} = \begin{cases} -\text{softmax}(\boldsymbol{x})_i \text{softmax}(\boldsymbol{x})_j, & i \neq j \\ \text{softmax}(\boldsymbol{x})_i (1 - \text{softmax}(\boldsymbol{x})_i), & i = j \end{cases}$$

You can use the latex layout from exercise sheet 2 for your submission.

Note: This exercise is a bonus, i.e., it is not necessary to solve this exercise to obtain 100 % on the sheet.