Problem 10.1 (Multilayer Neural Network and Backpropagation) **(30 P.)**

The goal of this exercise is to implement a generic multilayer neural network and the backpropagation algorithm. The implementation of the optimization algorithm (mini-batch stochastic gradient descent), a scaffold for the neural network and a test script will be provided although you can use your own implementations.

Generic multilayer neural network in this context means that you can easily change the number of hidden layers, the number of nodes per layer, the number of inputs and outputs, the error function and the activation function of the output layer. It should be easy to add another type of layer in the future. In this exercise we will only solve regression problems with fully conntected layers but in the next week we will solve a classification problem with a type of layer which is called *convolutional layer*.

We will use a very simple test function to evaluate the implementation:

$$f : [0, 1] \rightarrow [-1, 1]$$
$$f(x) = \sin(2\pi x)$$

a) Write a software implementation of the forward and backward propagation equations for a generic multilayer neural network with rectified linear unit (ReLU, $g(a) = \max(0, a)$) as activation function for the hidden layers. You only have to modify the file `multilayer_neural_network.py` for this exercise. You can use the provided test script `test_gradient.py` to check your implementation. Note that you **must not** modify this file. (20 P.)

b) Draw 200 samples (x^n, t^n) from the test function f, where $x^n \sim U(0, 1)$ (which means that x^n will be drawn uniformly from the interval $[0, 1]$) and $t^n \sim \mathcal{N}(f(x^n), 0.1)$, i.e. add normally distributed noise with standard deviation 0.1 to the value of the function f. Train the neural network with linear output activation functions $g(a) = a$ and sum of squared errors $E = \frac{1}{2}\sum_n ||y^n - t^n||_2^2$ as error function on these samples. We recommend to use two hidden layers with 50 and 20 hidden nodes and a standard deviation of 0.01 for the normal distribution from which you draw the initial values of the weights. The optimization algorithm should use a batch size of 16, the learning rate `alpha` should be 0.1, the momentum `eta` should be 0.5 and the number of training epochs can be set to 150. Plot the samples, the true test function f in the interval $[0, 1]$ and the prediction of the neural network in the interval $[0, 1]$. You can use the script `train_sine.py` for this exercise. (10 P.)

Problem 10.2 (Learning Inverse Dynamics) **(30 P.)**

We will now use the implementation of the neural network to solve a robotic problem: we learn the inverse dynamics of the Sarcos robot arm with seven degrees of freedom (DOF). The mapping that we want to learn has 21 inputs (state of the robot arm: positions, velocities, and accelerations of 7 joints) and 7 outputs (torques of the joints that are required to produce the state of the robot arm). See http://www.gaussianprocess.org/gpml/data/ for more details on the dataset. Since the inverse dynamics problem can be regarded as a regression problem, we will again use linear activation functions in the output layer and sum of squared errors as error function.

We provide the file `sarcos.py` that provides the function `download_sarcos()` to download the dataset, the function `load_sarcos(dataset)` to load either the training set or the test set and the function `nMSE` that computes the normalized mean squared error. You can implement your solutions for this exercise based on the code that has been provided with `train_sarcos.py`.

Note: Before we train a model on the dataset, we must to normalize the targets so that they have a mean of zero and a standard deviation of one. You can use e.g. `sklearn.preprocessing.StandardScaler`. Note that you must use the same scaling for the training and the test set! The function `nMSE` also assumes that the targets have been normalized.

a) Train a linear model (e.g. linear regression from scikit learn or a neural network without hidden layer) on the training set. Report the normalized MSE ($nMSE = \frac{1}{N}||y_f^n - t_f^n||_2$ with normalized outputs, i.e. the mean of t_f is 0 and the standard deviation 1) on the training set and on the test set for each component f of the output. (5 P.)

b) Train a neural network with 50 hidden units in one hidden layer on the training set. Use `epochs=100, batch_size=32, alpha=0.005, eta=0.5` for the optimization algorithm `MBSGD`. Report the nMSE for each component on the training set and on the test set. (5 P.)

c) Improve the results that you obtained. You are allowed to change the following parameters

- number of hidden layers
- number of hidden nodes
- batch size (`batch_size`)
- initial learning rate (`alpha`)
- learning rate decay (`alpha_decay`)
- minimum learning rate (`min_alpha`)
- momentum (`eta`)
- momentum increment (`eta_inc`)
- maximum momentum (`max_eta`)
- optional: preprocessing and feature generation

Read the documentation of MBSGD to understand all of these parameters. You must not increase the number of epochs. Of course you could decrease the number of epochs for some preliminary tests.

Apart from that, you are allowed to do anything to solve this task. However, **you must explain your approach and your solution in detail** (why did you use method A and not method B, what kind of metric did you use, ...). Here are some hints:

- do not use the test set during the optimization of the parameters
- training the model is computationally expensive
- scikit learn >= 0.15 provides various classes and functions that can help you to do model selection, e.g. `sklearn.learning_curve.*` (learning curves), `sklearn.grid_search.*` (model selection), `sklearn.cross_validation.*` (cross-validation generators); you can combine other cross-validation generators than k-fold cross-validation with learning curves and model selection algorithms; these classes and functions have been moved to the module `sklearn.model_selection` in version >= 0.18
- `MiniBatchSGD` implements the `sklearn.base.BaseEstimator` interface and, hence, integrates perfectly with scikit learn

(20 P.)

On the hand-in date, **25.01.2017**, you must hand-in the following: [1]

a) a text file stating how much time you (all together) used to complete this exercise sheet
b) your solutions / answers / code

for problem 10.1 and 10.2.

[1] upload via StudIP (if there are problems with the upload contact me **beforehand**: krell@uni-bremen.de)