# Deep Learning Assignment 4 in FS 2022

## Multi-Output Networks and Batch Processing

Microsoft Forms Document:
https://forms.office.com/r/nHKeCdSmTz

Manuel Günther

Distributed: Friday, March 18, 2022
Discussed: Friday, March 25, 2022

The goal of this exercise is to get to know some regularization techniques when implementing deep learning methods. For this purpose, we select a dataset the contains data in different formats, some binary ($x_d \in \{-1, 1\}$) and some numerical ($x_d \in \mathbb{N}$). As target values, this dataset contains three numerical outputs, so, $\vec{t} \in \mathbb{R}^3$ for each sample. These target values should be approximated with a two-layer multi-output network that we will train with the $\mathcal{J}^{L_2}$ loss.

# 1 Dataset

The dataset of our choice for today is the Student Performance estimation dataset that was collected in Portugal in two different schools and with two different subjects, i.e., math and Portuguese (the mother tongue). The dataset contains many different inputs such as a binary representation of the school, the gender, family sizes and alike, as well as numerical representations of age, travel time, and alcohol consumption. The dataset also includes some categorical data, which we skip in this assignment. See https://archive.ics.uci.edu/ml/datasets/Student+Performance for more information on the dataset.

**Task 1: Dataset Loading** The first task deals with the loading of the dataset. Due to the difficulty of the task, most of the implementation is provided. The implementation is very literal and, therefore, hopefully readable, while maybe not the most efficient. The bias value for $x_0 = 1$ is also already included. You just need to make sure that the data $(X, T)$ is returned in the desired format.

**Test 1: Assert Valid Data** Call the function to load the dataset of the math course (``mat''). Make sure that the dataset is in the correct dimensions for further processing, i.e., $\mathbf{X} \in \mathbb{R}^{(D+1)\times N}$ and $\mathbf{T} \in \mathbb{R}^{O\times N}$. Also assure that all target data is in range $t \in [0, 20]$.

**Task 2: Input Data Normalization** Since the data is in different input regimes, we want to standardize the data. For this purpose, we need to compute the mean and the standard deviation of the data for each input dimension. Then, we implement a function to perform the standardization of the data using the previously computed mean and standard deviation.

**Task 3: Batch Processing** In order to run stochastic gradient descent, we need to split our dataset into batches of a certain batch size $B$. Implement a function that yields one batch of size $B$ at a time, and circulates the dataset afresh when all data is consumed. Make sure that both the inputs and the targets are yielded.

**Test 2: Test your Batches** Design some test cases that assure that your batch generation function works as expected. You might want to design your own dataset, rather than relying on the dataset from Task 1. Call the batch function in a loop and assure that the yielded batches are as expected.

## 2 Gradient Descent

To train a two-layer multi-output regression network, we need to implement some functions. The network output is computed in three steps:

1. Compute network activation for a batch of inputs $\mathbf{X}$: $\mathbf{A} = \mathbf{W}^{(1)}\mathbf{X}$
2. Call the activation function element-wise: $\mathbf{H} = g(\mathbf{A})$. Here, we rely on the tanh activation function. Assure that the hidden neuron bias $\mathbf{H}_{0,:}$ is set appropriately.
3. Compute the output $\mathbf{Y}$ of the batch: $\mathbf{Y} = \mathbf{W}^{(2)}\mathbf{H}$.

**Task 4: Implement Network Output** Implement the output of the network for a given batch $X$, and given parameters $\Theta = (\mathbf{W}^{(1)}, \mathbf{W}^{(2)})$ using `numpy` operations. Make sure that the function returns both the output $\mathbf{Y}$ and the output of the hidden units $\mathbf{H}$ since we will need these in during gradient descent.

**Task 5: Loss Function** Implement a loss function that returns the squared loss $\mathcal{J}^{L_2} = \frac{1}{B}\|\mathbf{Y} - \mathbf{T}\|_F^2$ for given network outputs $\mathbf{Y}$ and target values $\mathbf{T}$.

**Task 6: Gradient Implementation** Implement a function that computes and returns the gradient for a given batch $(\mathbf{X}, \mathbf{T})$, the given network outputs $\mathbf{Y}$ and $\mathbf{H}$ as well as current parameters $\Theta = (\mathbf{W}^{(1)}, \mathbf{W}^{(2)})$. Make sure to compute the gradient with respect to both weight matrices. Implement the function using the fast version provided in the lecture.

**Task 7: Iterative Gradient Descent** Implement gradient descent for a given number of 10'000 epochs (not batches!) using given initial parameters $\Theta$ and a given batch size $B$, as well as a learning rate of $\eta = 0.001$. Make sure that the network output $\mathbf{Y}$ and $\mathbf{H}$ are computed only once per batch. Compute and store the obtained loss value (of the current batch) after each epoch. How can you know whether your current batch is the last one of the current epoch?

**Task 8: Run Gradient Descent Twice** Select an appropriate number of hidden neurons $K$. Initialize the weight matrices $\mathbf{W}^{(1)}$ and $\mathbf{W}^{(2)}$ using the Xavier initialization method as described in the lecture. Run the gradient descent twice, one using the normal gradient descent, and once stochastic gradient descent with $B = 16$. How can you achieve this by not modifying the implementation of the gradient descent function from Task 7?

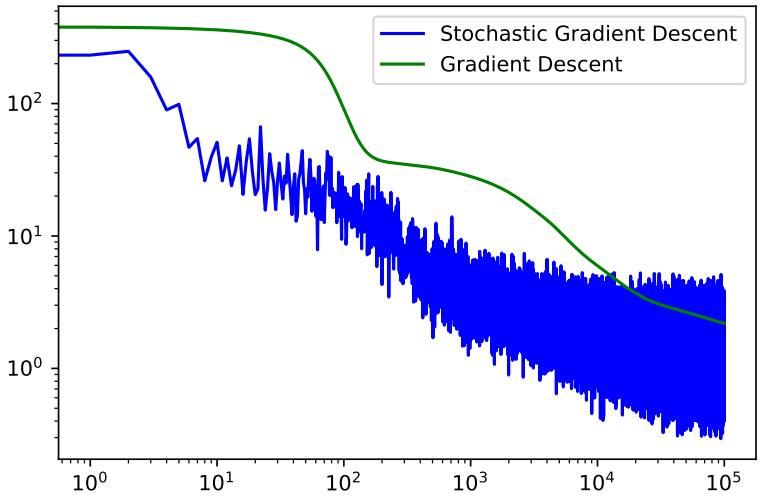


Figure 1: Exemplary loss progression as might be obtained on Task 9.

# 3 Evaluation

Finally, we want to evaluate how the learning process went and what the network has actually learned. For the former, we will plot the loss values obtained during training. For the latter, we define one specific sample of our own, and we evaluate the impact of several factors on the performance.

**Task 9: Loss Progression Plot** We want to plot the learning process of our network. For this purpose, plot the losses that were obtained during our training as line plots. First plot the loss values achieved by stochastic gradient descent, then the one achieved through gradient descent. Use logarithmic axes wherever you see a fit. An exemplary loss progression plot can be found in Fig. 1.

**Task 10: Example Evaluation** Design an input sample such that it would represent an individual. You can select a specific example, but you can also imagine a student. Please refer to `https://archive.ics.uci.edu/ml/datasets/Student+Performance` on possible values, and the implementation in Tasks 1 and 2 on how to generate an input sample $\vec{x}$ for our network. Compute the scores that your student would likely get by asking the network, using the parameters $\Theta$ optimized with stochastic gradient descent.

**Task 11: Influence of Data Dimensions** For some dimensions in the input feature $\vec{x}$, we want to test how different input values for this dimension would influence the outcome. Particularly, we test:

- Gender at index $d = 2$: change between male (1) and female ($-1$)
- Additional classes at index $d = 14$: change between yes (1) and no ($-1$)
- Romantic relations at index $d = 19$: change between yes (1) and no ($-1$)
- Weekday alcohol consumption at index $d = 23$: vary in range $[1, 6]$.