Tutorial 03: Fully Connected Neural Nets - Forward Propagation (20P)

In this exercise we will build a neural network (in particular a *fully connected neural network with 2 hidden layers*) from scratch using numpy routines. We will only consider the forward pass this week, i.e., which prediction $\hat{\boldsymbol{y}}$ our network gives for a certain input $\boldsymbol{X}$. Next week, we will implement the learning algorithm for the network using backpropagation.
Please submit your code by 11th Nov 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).

MNIST

Download the data https://megastore.uni-augsburg.de/get/I3mqqwBdZz/.
We will evaluate our system on the MNIST database, one of the most prominent machine learning databases. MNIST is a large corpus (60 000 examples in train, and 10 000 in test) containing pictures of hand-written digits and their according labels. The goal of any machine learning algorithm on the MNIST corpus is to classify the pictures to the correct digit i.e., MNIST provides a 10-class classification problem. http://yann.lecun.com/exdb/mnist/
The loading of the MNIST database is already implemented. The training data **trainx** is a numpy array of shape (60 000, 784) containing 60 000 training examples (data points) with 784 pixel values (features), flattened from the original 28×28 pixels, each. The training labels **trainy** are in the form of a numpy array of shape (60000,), and contain the correct digit (class number) for every training example. The test data **testx** and test labels **testy** are of the same dimension according shapes, but only for 10 000 data points.

1 Sigmoid Function (1P)

Implement a routine, which takes a numpy array as input and returns the sigmoid function

$$\sigma(x) = \frac{1}{1 + e^{-x}},$$

for all elements of the input.

2 Forward Propagation for One layer(2P)

Implement forward propagation through one layer of a fully connected neural network given the input **data**, the weight matrix **W**,the bias **b** and the activation function

activation.
Note: In python you can give a python routine as a parameter to another routine. Do not use the parentheses „()“ when doing so.

3 Neural Network Initialisation (3P)

Implement a python class for a 2-layer fully connected neural network. The constructor should initialize the parameters (weights and biases) of the neural network.
The shapes of the parameters depend on the numbers of the input neurons (**n_input**), of the two hidden layer neurons (**n_hidden1**, **n_hidden2**) and of the output neurons **n_out**.
Randomly initialize the network parameters using a normal distribution (**numpy.random.randn**).
*Note: If you want to obtain results consistent to the ones given below, use a random seed of **42**, and the following order of initialisation: $\boldsymbol{W}^1, \boldsymbol{b}^1, \boldsymbol{W}^2, ...$*

4 Softmax (2P)

Implement a routine that takes a two-dimensional numpy array as input and calculates the softmax function for „each row“, i.e. the second axis of the numpy array. The softmax function of $\boldsymbol{X} \in \mathbb{R}^n$ is defined as:

$$\text{softmax}(\boldsymbol{X})_i = \frac{\exp(\boldsymbol{X}_i)}{\sum_{j=0}^{n} \exp(\boldsymbol{X}_j)}$$

5 Full Forward Propagation (2P)

Implement the full forward propagation of the neural network as a class routine. The forward propagation should consist of three steps: From the input to the first hidden layer, from the first hidden to the second hidden layer, each with sigmoid activation and from the second hidden to the output layer with softmax activation.
Note: Use the weights and biases of the class that were previously initialised

6 Cross-Entropy(3P)

Implement the cross entropy for the classification problem

$$\text{crossentropy}(\boldsymbol{P}) = -log(\boldsymbol{P}_i), \qquad i : \text{correct label}$$

as a loss function, given the logits of the predicition **prediction_logits** indicating the calculated probabilities for each class and the labels **labels** indicating the correct label. Average the cross-entropy over all data points.
*Note: For the MNIST example **prediction_logits** is a numpy array with shape: (number of data points, number of possible classes) and **labels** is a numpy array with shape (number of data points,).*

7 Accuracy (2P)

Implement a rountine that calculates the accuracy

$$\frac{\#\text{ correct classifications}}{\#\text{ data points}}$$

by taking the class with maximum probability in the logits of the prediction **prediction_logits** and comparing it to the actual correct class labels **labels**.

8 Test your network (2P)

Create an instance of your network for the MNIST classification task. Use 400 neurons for both hidden layers. Calculate the cross-entropy loss and the accuracy of your randomly initialized network on the training data of the MNIST database.
Note: If you used 42 as a random seed and, you should obtain a loss of 20.019544264127 25 and an accuracy of 0.11021666666666667

9 Trainable Parameters (2P)

Which are the parameters we can tune to improve the performance of our network? How many trainable parameters (scalars) does our network have in total?

10 Evaluation Metrics (2P)

Why did we implement two different evaluation metrics of our system (cross-entropy and accuracy)? What are the main differences between the two and why can't/shouldn't we use them interchangeably?