Sheet 5: Supervised learning

Released: 06/27/23; Submit until: 07/10/23 (**20 Points**)

On this sheet we will implement a small neural network and use it for pattern recognition, a task that can be used for instance to classify measurement data and detect phase transitions. However, here we will use the back-propagation method to realize a version of an optimizer and apply it to train a network to identify hand-written numbers. For that purpose, the training and test datasets are obtained from the `MNIST` database (loading the data requires `Python`, so for this sheet it might be the preferred programming language).

Problem 1 A neural network class (10 Points)

We aim to learn features from grayscale images that are represented by $\mathbf{X} \in \mathbb{V}_{\mathbb{R}}^{28\times 28}$ matrices, fixing the number of nodes on the input layer to $N_I = 28^2 = 784$. The output layer should signal the detected feature, i.e., which number between 0 and 9 is shown on the input image yielding $N_O = 10$ output nodes. We then consider a single hidden layer which is represented by $N_H \in \mathbb{N}$ nodes. The input layer is connected to the hidden layer via a weight matrix $\mathbf{A} \in \mathbb{V}_{\mathbb{R}}^{N_I \times N_H}$, the hidden layer is connected to the output layer via a weight matrix $\mathbf{B} \in \mathbb{V}_{\mathbb{R}}^{N_H \times N_O}$, and we use a sigmoid activation function $\sigma(x) = \frac{1}{1+e^{-x}}$. A vectorized input signal $\underline{x} \in \mathbb{R}_I^N$ is then mapped to the output signal $\underline{y} \in \mathbb{R}_O^N$ via an intermediate signal $\underline{m} \in \mathbb{R}^{N_H}$ in the hidden layer, evaluating successively

$$m_j = \sigma(\sum_{i=0}^{N_I-1} A_{ij} x_i) \ , \quad y_k = \sigma(\sum_{j=0}^{N_H-1} B_{jk} m_j) \ .$$

(1.a) **(6P)** Write a neural network class which wraps the described architecture. It should implement methods to update weight matrices $\mathbf{A}, \mathbf{B}$ as well as a function that evaluates the output signal $\underline{y}$, given an input signal $\underline{x}$. Using the evaluation function, implement a method that computes for a given input $\underline{x}$ and an expected feature $t \in [0, 9]$ the cost function

$$C(\mathbf{A}, \mathbf{B}) = \frac{1}{2} |\underline{y} - \underline{t}|^2 \ ,$$

where $\underline{t} = 0.99 \underline{e}_t + 0.01 \sum_{t' \neq t} \underline{e}_{t'}$ is the feature vector. Note that we avoided the choice of a canonical unit vector $\underline{e}_t$ as feature vector. This will be usefull when training the network, since exact 0's and 1's may cause the gradient optimization to fail. It may also be useful to write two helper functions, which either propagate an input signal $\underline{x}$ up to a certain layer or back-propagate an output signal $\underline{y}$ up to a certain layer and return the corresponding, propagated signals.

(1.b) **(4P)** We want to train our neural network by minimizing the cost function with respect to all parameters A_{ij}, B_{jk}. For that purpose, we use a back-propagation scheme of the errors per layer with a fixed learning rate $\eta > 0$. In one training iteration, we update the parameters by shifting them in the direction which minimizes the errors:

$$B_{jk} \longleftarrow B_{jk} - \eta \frac{\partial C}{\partial B_{jk}} \quad A_{ij} \longleftarrow A_{ij} - \eta \frac{\partial C}{\partial A_{ij}} , \tag{1}$$

beginning at the output layer. The derivatives are evaluated using the chain rule. In order to determine the weights of the nth weight matrix $\mathbf{M}^n$, we then propagate the input signal $\underline{x}$ from the input layer to the $(n-1)$th layer yielding the propagated input $\underline{m}^{n-1}$, using the old weight matrices. We furthermore propagate the target feature vector $\underline{t}$ as well as the output vector $\underline{y}$ from the output layer to the $(n+1)$th layer yielding the back-propagated vectors $\underline{t}^{n+1}$ and $\underline{y}^{n+1}$, using the updated weight matrices. The gradient in the nth layer $\mathbf{M}^n$ is then given by

$$\frac{\partial C}{\partial M_{jk}} = (t_k^{n+1} - y_k^{n+1}) \cdot \sigma\left(\sum_{j'} M_{j'k}^n m_{j'}^{n-1}\right) \cdot \left(1 - \sigma\left(\sum_{j'} M_{j'k}^n m_{j'}^{n-1}\right)\right) m_j^{n-1} \tag{2}$$

where the sums are over the input dimension of the nth weight matrix $\mathbf{M}^n$. Write an update method, which performs the back-propagation steps for each layer and replaces the weight matrices. Try to reuse intermediate results to improve the performance.

Problem 2 Training and confusion **(10 Points)**

We now want to train our neural network, which means that we need a measure for the quality of the predicted features. This can be provided by the confusion matrix $\mathbf{K} \in \mathbb{V}_{\mathbb{N}}^{N_O \times N_O}$. The matrix elements $K(t, y)$ count the number of generated output features y (here we take as output feature the output node with the highest weight in the output signal), given a certain expected target feature t. Thus, on the main diagonal $\mathbf{K}$ contains the number of successfully predicted features, while the off-diagonal elements keep track of false predictions.

(2.a) **(3P)** Write a method which continuously updates the confusion matrix when propagating an input signal through your network. Furthermore write methods which compute the total success-rates for each feature $s = \sum_t K(t,t) / \sum_{t,y} K(t,y)$, as well as the so-called precision rates per feature $p(t) = K(t,t) / \sum_y K(t,y)$ and the recall rates per output $r(y) = K(y,y) / \sum_t K(t,y)$.

(2.a) **(3P)** Download the `MNIST` datasets from the lecture's home directory: `/project/cip/2023-SS-NQP/mnist`, you can load the data in python using numpy's `load` routine. These are grayscale images which have pixel values $p(x, y) \in [0, 255]$. Write a method to rescale these values to lie in the interval $[0.01, 0.99]$ to improve the stability of the gradient evaluations during back propagation.

(2.c) **(4P)** Use a number of $N_H = 100$ hidden nodes and a learning rate $\eta = 0.1$ to train your neural network on the training datasets by randomly choosing N_T images and use its target feature to update the neural network weight matrices (you can initialize these matrices with normally distributed numbers). For different values of N_T, use all test datasets and evaluate the confusion matrices and plot the success, precision and recall rates as function of N_T.