Assignment 1: Neural Networks
Machine Learning

Deadline: Friday 19th Nov 2021, 21:00

Introduction

In this assignment, you will learn how to implement an Artificial Neural Network with Python and Numpy and train it on toy data. To get started with the assignment, you will need to use the starter code provided with this assignment. For each of the tasks below it should be clear that you have to implement the algorithms yourself, i.e. you cannot use a neural network library and you are not allowed to reuse code from any other sources. The code has to be accompanied with a report answering part 1 in PDF format (preferably, but not necessarily, LaTeX). Handwriting and scanned documents are not allowed.

Your report and source code must be archived in a file named "firstname.lastname.zip" and uploaded to the iCorsi website before the deadline expires. Your report should also contain your name.

Where to get help

We encourage you to use the tutorials to ask questions or to discuss exercises with other students. However, do not look at any source code written by others or share your source code with others. In case you need further help, please write on iCorsi or contact me at vincent.herrmann@idsia.ch.

Grading

The assignment consists of three parts totalling at 100 points. Deadline is hard: late submissions will result in 0 points, as well as hand written answers, handing in .doc/docx files, sharing solutions, plagiarism. Reusing each others code by changing the variable names, alignment of the code, or similar count as plagiarism and will result in failing the course for all the students involved.

1 The Perceptron (20 points)

Before working on a neural network we will study the perceptron; a linear classifier without an activation function, or a very simple single layer network as given in Figure 1.

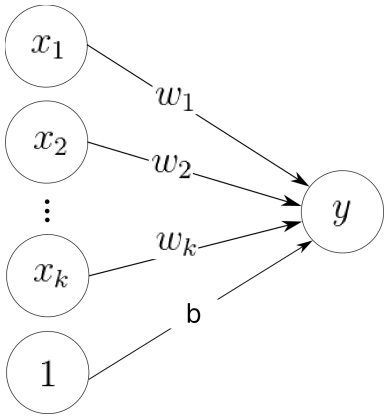

Figure 1: The linear perceptron with single output

$\boldsymbol{X}_{i,1}$	$\boldsymbol{X}_{i,2}$	$\boldsymbol{X}_{i,3}$	t_i
6	1	-11	0
-5	6	7	1
-5	-11	1	0
-11	3	5	0
-5	-2	1	1
5	-6	-7	1
2	0	2	1
4	-9	-11	0
9	-7	-9	0
-8	4	-8	1

Table 1: The training batch

You are given a batch of data of size n: $\mathcal{D} = \{(\boldsymbol{x}_1, t_1), (\boldsymbol{x}_2, t_2), \ldots, (\boldsymbol{x}_n, t_n)\}$, where input is represented as k-dimensional vectors $\boldsymbol{x}_i \in \mathbb{R}^k$, and the targets $t_i \in \mathbb{R}$ are scalars. Assume that the bias term $b \in \mathbb{R}$ is *not* part of weight matrix $\boldsymbol{W} \in \mathbb{R}^{k \times 1}$.

1. (1 point) We assemble the batch of n vectors x_i into a matrix $\boldsymbol{X} \in \mathbb{R}^{n \times k}$ to calculate multiple outputs in parallel. What is the dimension of the output vector $\boldsymbol{y}$ if $\boldsymbol{X}$ was the input?

2. (3 points) Write down the vectorized equation for the forward pass of the perceptron with input $\boldsymbol{X}$. Use $\mathbf{1}_n$ as a column vector of ones of length n (watch out for dimensions to match, your result $\boldsymbol{y}$ has to have the dimension you determined in 1.).

3. (3 points) Write down the vectorized equation for the mean square error (MSE) of the perceptron in terms of $\boldsymbol{y}$ and $\boldsymbol{t} \in \mathbb{R}^n$.

4. (4 points) Determine the derivative of the error function w.r.t. the weights.
   Hint: $\frac{d\boldsymbol{x}^T\boldsymbol{x}}{d\boldsymbol{x}} = 2\boldsymbol{x}$.

5. (3 points) Write down the equation of the weight update for one step of gradient descent.

6. (3 points) Suppose $k = 3$, $n = 10$, with a batch of data given in Table 1. The initial weights are $w_1 = -0.1$, $w_2 = -0.3$, $w_3 = 0.2$ and the bias weight is $b = 2$. Compute the weights after one step of gradient descent with learning rate of $\eta = 0.02$ (report their value up to 2 decimal points). *Hint: MATLAB/Octave/Python might come in handy here.*

7. (3 points) Learning in multi-layer neural networks is usually done with help of two methods: backpropagation and gradient descent. Describe briefly what role in the learning process each of the two has (should not be longer than 6 lines).

2 Simple Machine Learning Framework (50 points)

In this part we are going to implement a very simple machine learning framework. The goal is that at the end of this part, you will have a rough idea what is happening inside a real world ML framework (e.g. PyTorch, TensorFlow, JAX, etc). Using this framework you will be able to easily create arbitrary fully connected neural networks. It will have the following building blocks: linear layer, sequential layer, tanh and softmax activation functions, and cross-entropy loss. The skeleton is given, you will have to fill in the missing parts. Pay attention to the description in the skeleton file: it clarifies the task further, and also gives some useful hints. *You are not allowed to change the signature of the functions (name, the format of the arguments or output).* They will be auto-checked, so their interface should match.

You will test your framework on a synthetic two-class spiral dataset. You will use a simple, 3 layer network with tanh activation function on hidden layers and a softmax output layer. The output sizes are 50, 30, 2, respectively.

1. (15 points) Implement the activation functions and their derivatives in the code skeleton (*tanh*, *softmax*).
2. (10 points) Implement forward and backward pass for the linear layer.
3. (10 points) Implement forward and backward pass for the cross-entropy loss.
4. (5 points) Implement the sequential layer.
5. (10 points) Create the network and implement a single step of training.

Your task is to fill missing parts of the framework.py file. Run it with **Python 3** in order to verify whether your solution is correct. Make sure your code runs and does not crash with an exception.

3 Handwritten Digit Recognition (30 points)

In this part you will use your ML framework to train a network that can classify handwritten digits. The network is a very simple 2 layer network, with layer output sizes of 20 and 10 respectively. Again, the skeleton is given, and you have to fill in the missing parts. After this part, you will have a rough idea how to build a proper training pipeline. Pay attention to the description in the skeleton file: it clarifies the your task further, and also gives some useful hints. Reuse as much as you can from framework.py by importing the corresponding functions and classes (e.g the layers, the train_one_step() function, etc).

1. (5 points) Split the data into a training and a validation set.
2. (10 points) Implement the main training loop.
3. (10 points) Implement one function for testing the network on the validation set and one for testing it on the test set.
4. (5 points) Implement early stopping.

Your task is to fill missing parts of the minst.py file. Run it with Python 3 in order to verify whether your solution is correct. Note: The mnist.py and framework.py must be in the same folder for this to work. You will find lots of hints in the templates. Note: in case you return wrongly shaped vectors, the code might fail outside the part you had to fill, but this does not mean that your part is right and the template is wrong.