Assignment 1: Neural Networks
Machine Learning

Deadline: Friday 20 Oct 2017, 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 latex based report in the PDF format.

Your report and source code must be archived in a file named "firstname.lastname" and uploaded to the iCorsi website before the deadline expires. Your report should also contain your name. Late submissions will be subject to a point penalty according to USI regulations.

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. Violation of that rule will result in 0 points for all students involved.

Grading

The assignment consists of three parts totalling at 100 points. Bonus points are not necessary to achieve the maximal grade. You will be awarded 5 bonus points for a well-presented report (clear writing, annotated equations), as well as, a good programming style (clear code, useful comments, simple to execute). Further bonus points are elaborated in the text below.

1 The Perceptron (25 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 (if you will) as given in figure 1.

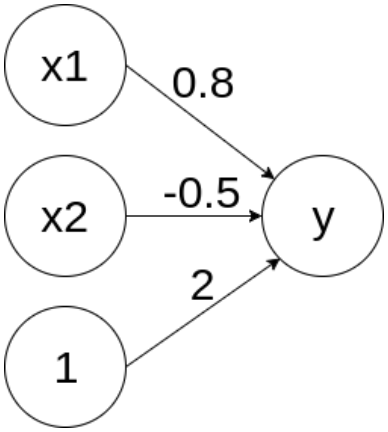

Figure 1: The perceptron with the initial weight values to be used.

Table 1: The training set consists of these points

x_1	1	6	3	4	3	1	6	7	6	10	4
x_2	8	2	6	4	1	6	10	7	11	5	11
$target$	1	1	1	1	1	1	-1	-1	-1	-1	-1

1. (10 points) Write down the vectorized equations for the forward pass of our perceptron using W for the weights, the mean squared error of our prediction and the target, and determine the derivative of the error with respect to the weights. Compute the new weight values after one step using a learning rate of 0.02 using Gradient Descent.

2. (2.5 points) Write down the equation of Gradient Descent and explain why it can be used as a Learning Algorithm. Provide a positive and negative argument.

3. (2.5 points) Implement the MSE (function *MSE*) and its derivative (function *dMSE*) in the code skeleton.

4. (5 points) Implement the function *forward* and *backward* in the Perceptron class. Implement a function to perform a single full batch gradient step (function *train_one_step*)

5. (5 points) Implement the *run_part1* function which trains the perceptron for 15 steps (use a learning rate of 0.02) and plot the final result using the *plot_boundary* function.

2 A Neural Network (50 points)

In this part, we are going to implement and train a Multi-Layer Perceptron i.e. a three-layer deep fully connected neural network and train it on the two-class spiral dataset

using the MSE cost function. For this part we generate the data using the *twospirals(n_points=120, noise=1.6, twist=420)* function. After every perceptron/neuron, we will use the hyperbolic tangent (tanh) activation function apart from the output neuron where we will use the sigmoid as the activation function. The network resembles the network in figure 2 in structure. The number of neurons from the first to the last layer should be 20, 15, 1.

1. (5 points) Implement the activation functions and their derivatives in the code skeleton (*sigmoid*, *dsigmoid*, *tanh*, *dtanh*).

2. (15 points) Write down the forward pass, derive the delta rules, and write down the derivatives of the weights using the deltas.

3. (5 points) Implement the functions *forward* and *backward* of the Neural Network class.

   Hint: Complete the gradient_check function which computes the gradient numerically to check if your forward and backward implementation is computing the correct gradient.

4. (15 points) Split the data into a train set and a test set, and plot the train and test data. Explain your reason for the chosen ratio. Initialize the weights randomly and find a good learning rate to train the network until convergence. Plot the learning rate of at least 5 different learning rates on both sets. Plot the boundary of the model which converged the most on the training data (it should achieve an MSE train set error of less than 0.02).

5. (10 points) Explain if this is a good model for the data. If not, find a better model and plot its boundary. Explain this is can be done.

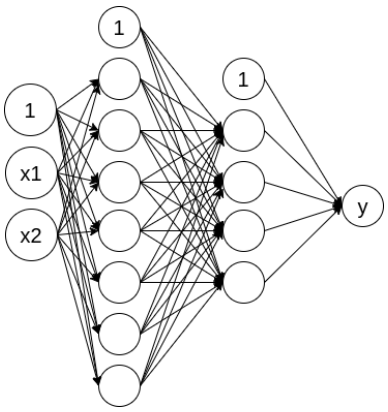

Figure 2: The neural network of this part is a directed acyclic graph, or more precisely, a fully connected neural network with three layers.

3 Further Improvements (25 points)

In this third part, your task is to improve upon the model of part two. Explain intuitively (or mathematically if possible) the problem you try to attack, implement your improvement, and analyse the two models empirically. Simply increasing the width of your network is not going to earn you the maximum number of points. You have to deliver at least one specific extension. Your analysis should be thorough. Make sure to show the benefits of your extensions. More than one extension will, to some extent, be awarded bonus points. Here are a few ideas:

- Improve Gradient Descent to be stochastic and use heuristics such as an adaptive learning rate and momentum for faster convergence.
- Train a deep network with more than 20 layers using skip connections.
- Use a different activation function and choose a good initialization.
- Perform multi-classification (2 outputs with one hot targets) using the softmax activation function and change the loss function.
- Provide a faster implementation through the use of parallelism, clever implementation, or GPU acceleration (without the use of deep learning libraries).

If you are not sure if your idea/extension is good enough go and speak to your TA.

Bonus points: If you submit your trained model we will evaluate it on some held out test data. If your model is among the three best it will be rewarded with 15 bonus points. In order to compete, you must implement the two competition functions. One function should load the submitted weights and compute the accuracy given our own data set while the other should train the network from scratch. The winner is defined by max accuracy. **For this challenge the train data has to be generated using *twospirals(250, noise=0.6, twist=800).*** The best networks will be retrained in order to prevent cheating. Submissions that fail to be retrained will be disqualified. The submission which cannot be executed will be disqualified.

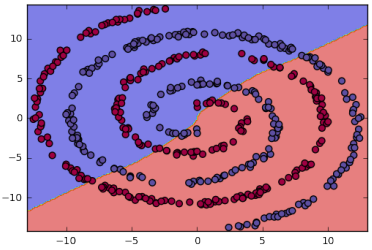

Figure 3: The spiral data for the challenge.