Practical work 01 – 20/02/2024
Gradient Descent

Objectives

The main objectives of this Practical Work for Week 1 are the following :

a) Recap basic usage of python and particularly with numpy.

b) Implement gradient descent for the perceptron model and then apply it to Fashion-MNIST data.

Submission

— **Deadline** : Tuesday 27 February, 3pm

— **Format** :

  — Exercise 1 (Sigmoid Function) :

    — Maths calculations either in a pdf with your handwritten solutions or calculations all in the Jupyter Notebook (by using latex). Don't just state the final results but expand on how you obtained them.

  — Exercise 2 (Gradient Descent Implementation) :

    — Jupyter Notebook `generalised_perceptron_stud.ipynb` completed with your solutions.

  Submission of all files in a single zip-file using the naming convention (for team of two students #1, #2) :

  `family name_given name #1- family name_given name #2.zip`

Exercise 1 Sigmoid Function

(a) Compute the derivative of the sigmoid function

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

(b) Show that the derivative fullfills the equation

$$\sigma'(z) = \sigma(z) \cdot (1 - \sigma(z))$$

Exercise 2 Gradient Descent for Perceptron

Implement gradient descent learning for the generalised perceptron and analyse the results. Do this on the basis of the Jupyter Notebook `generalised_perceptron_stud.ipynb` . Use **numpy** functionality only. The sections of code that you need to implement are marked with

START YOUR CODE

END YOUR CODE

Proceed as follows :

(a) Work your way through the preprocessing steps in the notebook consisting of
   — loading the data : Use Fashion-MNIST for all what follows.
   — plotting the data (`plot_tiles`).
   — filtering the data for the classes of interest, splitting the data into a train and a test set as well as normalising it (`prepare_data`).

(b) Complete the function (`prepare_data`) with correct splitting of the data into a train and test set as well as the data normalisation.

(c) Study the class `NeuralNetwork` and implement (from top to bottom) the sigmoid activation function (`activation_function`), the gradient calculation (`back_propagate`) and the cost function (`cost_funct`). Both cost and gradients have to be implemented for MSE and CE cost.

(d) Run the training (chosing an appropriate learning rate and number of epochs) both on MSE and CE and determine the matrices for any combinations of binary classifications possible (c.f. Table 1 in `TSM-DeLearn_Lecture-Notes`).

(e) For MSE cost only do 10 successive training runs and determine the average and standard deviation of the final error rates obtained (c.f. Table 2 in `TSM-DeLearn_Lecture-Notes`).

Hints :

— Keep an eye on the shapes of the arrays (as used in the dummy implementation).

— In case of problem you may want to try using PyCharm debugger to analyse problems.