Practical work 02 – 28/02/2023
Gradient Descent

Objectives

The main objectives of this Practical Work for Week 2 are the following :

a) Get more experienced in python, particularly with numpy.

b) Refresh or deepen your maths skills (multi-variate calculus)

c) Implement gradient descent for the perceptron model and then apply it to MNIST. Study some of the main features.

Submission

— **Deadline** : Tuesday 7 March, 3pm

— **Format** :

  — Exercise 1 (Numpy)
    — Jupyter notebook `numpy-tutorial-stud.ipynb` completed with your solutions.

  — Exercise 2 (Sigmoid Function) :
    — Jupyter Notebook with the sigmoid function and the plot (incl. derivative).
    — 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 3 (Gradient Descent Implementation) :
    — Jupyter Notebook `MNIST_binary_classifier_stud.ipynb` completed with your solutions.
    — The answers to the questions either in a small pdf-report or in the Jupyter Notebook.

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 Optional : Numpy in a Nutshell

This exercise is to become more familiar with `numpy`. Read the content of the jupyter notebook `numpy-tutorial-stud.ipynb` that you will find on Moodle. Pay special attention to the *broadcasting* section that allows to gain significant speedup when processing large numpy arrays. Regarding this, it is usually more efficient to use *broadcasting* instead of for loops in your code.

At the end of the tutorial, you have to complete some manipulations of images stored by arrays.

Exercise 2 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

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

(c) Compute the first and second derivative of

$$\zeta(z) = -\log(\sigma(-z)) \qquad \log : \text{Natural Logarithm}$$

Compute the asymptotes for $z \to \pm\infty$. Create a plot of ζ.
(Hint : For $z \to +\infty$ you may consider the limit of the function $\zeta(z)/z$.)
Remark : This function is also referred to as softplus, a smooth alternative of $x^+ = \max(0, x)$, the *rectifier*.

(d) Implement the sigmoid function in a Jupyter Notebook. Make it works such that you can pass numpy arrays of arbitrary shape and the function is applied element-wise. Plot the sigmoid function and its derivative by using matplotlib.

(f) Show that the function

$$c_1(x) = (\sigma(x) - 1)^2$$

is non-convex. (Hint : Consider the second derivative.)
Explain in which situations (initial settings) optimising $c_1(x)$ with gradient descent may become difficult. For the explanation create a plot. Note that this exercise should give an intuition on why mean-square error loss is less suited for classification problems.

(g) Compute the first and second derivative of the function

$$c_2(x) = -\left(y \log(\sigma(w \cdot x)) + (1-y)\log(1 - \sigma(w \cdot x))\right)$$

with respect to $w \in \mathbb{R}$ and for given $y \in \{0, 1\}$. Show that c_2 is convex.

Exercise 3 Gradient Descent for Perceptron

Implement gradient descent learning for the single perceptron and analyse the results. Do this on the basis of the Jupyter Notebook `MNIST_binary_classifier_stud.ipynb`. Do this by only using numpy functionality (scikit learn is only used for loading the data and splitting into train and test sets). The sections of code that you need to implement are marked again 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 MNIST Original for all what follows.
   — filtering the data for the digits of interest ('1' and '7') so that a binary classification problem is obtained
   — splitting the data into a train and a test set

(b) Study the class `GradientDescent` and implement (from top to bottom) the normalisation of training and test data (`normalise_data`), MSE and CE cost (`cost_funct`), associated gradients (`grad_cost`), the prediction (`predict`) and the update step for the weights and the bias (`update`). Pay attention to keep the same matrix dimension as the dummy implementations.

(c) Run the training by optimising on cross-entropy cost and plot the learning curves : Cost, Error Rate, Learning Speed and convince yourself that you obtain curves similar as seen in the class.

(d) Analyse the dependency of the final error rate on the number of epochs. What is the goal of the learning and how many epochs make sense ? (Choose here the learning rate $\alpha = 0.5$.)

(e) Analyse the dependency on the learning rate by trying different values, e.g.

$$0.01, 0.05, 0.1, 0.5, 1.0, 1.5, 2.0, 5.0, 10.0.$$

Determine the reasonable number of epochs to learn for each learning rate. Describe what you observe. How large can you choose the learning rate until the learning breaks down ?

(f) Plot a histogram of the weights finally obtained from learing. A strong peak at zero remains. Why ? You may understad this when plotting the weights as an image. Also compare this image with the misclassified test images. Try to explain.

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.