Practical work 02 – 26/9/2019
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** : Wednesday 2 October, 10am

— **Format** :

  — Exercise 1 (Numpy)
    — Jupyter notebook `numpy-tutorial-stud.ipynb` completed with your solutions.
  — Exercise 2 (Sigmoid Function) :
    — pdf with your handwritten solutions.
    — Jupyter Notebook with the sigmoid function and the plot (incl. derivative).
  — Exercise 3 (Gradient Descent Implementation) :
    — Jupyter Notebook `MNIST_Binary_PW-stud.ipynb` completed with your solutions.
    — Small pdf-report with plots and the answers to the questions.

Exercice 1 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.

Exercice 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 work 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.

Exercice 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_Light_Binary_PW-stud.ipynb`. Do this by only using numpy functionality (scikit learn only 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
   — rescale the input features to lie between $[0, 1]$

(b) Implement the sigmoid function, CE and MSE cost, associated update rules (`step_CE` and `step_MSE`) and the `optimise` function. Make sure that your implementation works for images of arbitrary shapes.

(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 ? Also check the learning speed (length of the gradient) and interpret what you see.

(f) Plot the weights finally obtained from learning as image and compare it with the misclassified test images. Try to explain.

(g) Optional : Learn binary classification for all the pairs of digits (using reasonable values for the learning rate and the number of epochs). Compare the finally obtained error rates. Which pairs of digits work specifically well - which ones rather bad ? Interpret these findings. Are some of the problems linearly separable ?

Hints :

— Keep an eye on the shapes of the arrays (as passed into the functions and as used within the functions (and declared in the function descriptions).

— Run the tests (kind of unit tests) marked with `## TEST ##` which should not raise an Assertion Error.

— Possibly, add `assert()` statements in the code.

Exercice 4 Optional : Review Questions

a) Explain why normalisation is beneficial.

b) In what sense are optimisation techniques important for machine learning problems ?

c) Describe in what problems gradient descent is applicable. In what problems will it necessarily lead to a unique solution ? Describe what can go wrong in more general problems and why.

d) Why is learning with MSE cost considered less suitable for classification problems ?

e) What may happen if the learning rate is chosen too large ?

f) Why is it possible that the test error starts increasing after some epochs of training ?

g) Is it becoming more or less difficult to reach small values of the cost function if we have more training data ?

Exercice 5 Optional : Reading Assignment

Read the start of the Section 4.3 on *Gradient-Based Optimization* in the *Deep Learning* book by Ian Goodfellow et al (see https ://www.deeplearningbook.org/contents/numerical.html). Students that have learned multi-variate calculus will find interesting reading also in Section 4.3.1.