MASTER OF SCIENCE IN ENGINEERING

Teachers: J. Hennebert, M. Melchior
Assistants: Y.-I. Beffa, S. Hackstein, O. Zayene

Practical Work 02 – 3/3/2022
Gradient Descent

Objectives

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

a) Get more experienced in python, numpy and in particular with pytorch.

b) Implement the linear regression model by using pytorch tensors. Use the closed-form formulas and gradient descent.

c) Use gradient descent for learning multinomial logistic regression applied to MNIST with manual implementation of the gradient. Study the behavior of the learning with different settings for the learning rate and the batch size.

d) Use gradient descent for learning binary and multinomial logistic regression applied to MNIST with autograd.

e) Refresh or deepen some of your maths skills (calculus).

Submission

— **Deadline** : Wednesday 16 March, noon

— **Format** :

For Exercises 1-3 jupyter notebooks completed with your solutions and answers.

For Exercise 4+6 (optional) a pdf with your maths calculations.

In the different notebooks, you will find sections marked with
YOUR CODE (START)

YOUR CODE (END)

You will need to complete those to make the notebooks work.

Exercise 1 Gradient Descent (GD) for Linear Regression

Implement gradient descent learning for the linear regression example of week 1. Do this on the basis of the Jupyter Notebook `pw02_lin_regression_GD_stud.ipynb`.

This includes the following steps :

a) Implement the solutions of the normal equations (see PW01) by using pytorch tensors.

b) Implement (full batch) GD for linear regression - by using pytorch tensors. Don't use automatic differentiation, but implement the formulas for the gradient explicitly.

c) Make sure that the your GD descent algorithm works correctly by checking that the solution gradually approaches the correct solution as computed in the first step (within numerical precision). Plot estimated MSE cost vs the number of epochs with matplotlib ("loss curve").

d) Play with different learning rates. Compare the trainings on the basis of the associated loss curves. Describe what you see and interpret what is going on.

e) What is the maximal learning rate that you can use ? What happens when choosing a larger learning rate ?

Exercise 2 Binary Classification with Logistic Regression

In this exercise you model the spine dataset (downloadable from kaggle) with (binary) logistic regression. Since there is no closed-form solution in this case, you apply gradient descent to find a solution. Don't use pytorch's autograd functionality to compute the gradients in this exercise - compute gradients explicitly.

This includes the following steps :

a) Download the dataset from kaggle (see the link in the notebook). Load it into a pandas dataframe (see the code in the notebook).

b) This (tabular) data needs to be normalized. More on normalization later in the course - again suitable code is given in the notebook.

c) Complete the code for the implementation of the methods `predict`, `cost`, `gradient_cost`, `accuracy`. As a test, just invoke the method by suitable dummy values.

d) Implement (full batch) GD for linear regression - by using pytorch tensors. Don't use automatic differentiation, but implement the formulas for the gradient explicitly. Plot estimated MSE cost vs the number of epochs with matplotlib ("loss curve").

e) Run the training for different learning rates and explore how the training changes with different learning rates. How would you determine a suitable learning ?

More details are found in the Jupyter Notebook `pw02_binary_classification_GD_stud.ipynb`.

Remarks :

— For two class classification you don't need the softmax function, but just the sigmoid function that outputs the probability of observing the class with label Í. In principle, you

could also use $softmax$ with two classes. Nevertheless, we think that it is instructive to go through and implement the formulas for standard binary logistic regression.

— We focus here on getting the model trained without doing so-called hyper-parameter tuning and evaluating the performance of the trained model on new data. We will do that in the next exercise.

Exercise 3 Gradient Descent for MNIST Classification

Implement gradient descent learning for the multinomial logistic regression model and analyse the results. Do this on the basis of the Jupyter Notebook `pw02_softmax_classification_stud.ipynb`. In a first part, don't use pytorch's autograd functionality to compute the gradients - rather compute them explicitly. In a second part, see how you can implement the same model and its training with full-fledge pytorch functionality including autograd.

Proceed as follows :

This includes the following steps :

a) Load the MNIST dataset by clicking through the first cells and see how to depict some samples. Please note that a training and a test dataset are loaded. Make sure that only the training set will be used for training and the test dataset just to evaluate the performance (by checking the accuracy).

b) Implement the multinomial logistic regression model by completing the code in the functions `linear_trsf`, `softmax`, `predict`, `loss_ce`, `cost_ce`.

c) Implement the mini-batch gradient descent training loop configured by the number of epochs, batch size and learning rate.

d) Run a test with `nepochs=10`, `nbatch=64`, `lr=0.01`.

e) For comparison implement the same model by using a model class (inheriting from *torch.nn.Module* and by using the layer functionality in the package *torch.nn* (see lecture). Implement the training loop also for this model.

f) Run the same test with `nepochs=10`, `nbatch=64`, `lr=0.01` also with this implementation.

g) Compare different settings for the learning rate and batch size and qualitatively characterise the behaviour.

h) What is your favorite combination (learning rate, batch size, number of epochs) ?

Hints :

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

— For each implemented function, run a small test with dummy input tensors (of the required shape) and check whether it outputs a tensor with expected shape and no 'nan'.

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

Exercise 4 Optional : Review Questions

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

b) Describe what problems gradient descent can be applied to. In what problems will gradient descent lead to a unique solution ? Describe what can go wrong in more general problems and why.

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

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

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

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

Exercise 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, p.80-84).

Exercise 6 Optional : Calculus Training

(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.
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 for $x \neq 0$.