Labs
Machine Learning Course
Fall 2021

EPFL
School of Computer and Communication Sciences
Nicolas Flammarion & Martin Jaggi
www.epfl.ch/labs/mlo/machine-learning-cs-433

Problem Set 08, Nov 09, 2021 (PyTorch Introduction)

Goals. The goal of this exercise is to

- introduce you to the PyTorch platform.
- discuss the vanishing gradient problem

Problem 1 (PyTorch Getting Started):

Tutorials. We (optionally) recommend using the following official tutorials:

- Deep Learning with PyTorch: a 60-minute Blitz
- Learning PyTorch with Examples

Setup, data, and sample code. Obtain the folder `labs/ex08` of the course github repository:

github.com/epfml/ML_course.

You can use PyTorch without any setup on **Google Colab** (recommended), or install it on your own computer. Google Colab gives you a free GPU to play with. You can upload the notebook from this exercise to Colab to get started.

Exercise 1: Torch Familiarize yourself with the basics of PyTorch through the tutorial.

Exercise 2: Basic Linear Regression

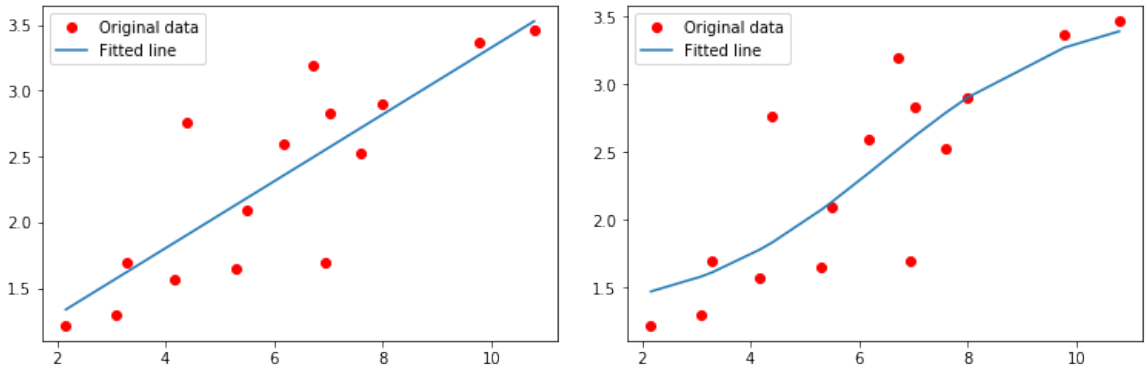
- Implement prediction and loss computation for linear regression using tensors in the `MyLinearRegression` class.
- Implement the gradient descent steps in the `train` function.
- **HINT:** don't forget to clear the gradients computed at previous steps.

Your output should be similar to that of Figure 1, left.

Exercise 3: NN package

- Re-implement Linear Regression as a sub-class of PyTorch's nn.Module but without using any pre-existing modules in the `nn` package. Make sure you obtain similar results to the previous example.
- **HINT:** Use `nn.Parameter` to declare your parameters.
- Re-Implement Linear Regression and MSE loss using only pre-existing modules from the `nn` package and without defining a new module. Does the result that you obtain differ from the previous one? If so, why?
- Combine two linear layers and a non-linearity (sigmoid or ReLU) layer to build a Multi-Layer Perceptron (MLP) with one hidden layer, in the `MLP` class. Find the optimal hyper-parameters for training it.

Your prediction using the MLP should be non-linear, and for a hidden size of 2 might look like Figure 1, right.



(a) Prediction with linear regression

(b) Prediction with MLP

Figure 1: Predictions made by various trained models.

Problem 2 (Vanishing Gradient Problem):

Over the past few years it has become more and more popular to use "deep" neural networks, i.e., networks with a large number of layers, sometimes counting in the hundreds. Emperically such networks perform very well but they pose new challenges, in particular in the training phase. One of the most challenging aspect is what is called the "vanishing gradient" problem. This refers to the fact that the gradient with respect to the parameters of the network tends to zero typically exponentially fast in the number of layers. This is a simple consequence of the chain rule which says that for a composite function $f(\boldsymbol{x}^{(0)}) = (f_{L+1} \circ f_L \circ \cdots f_2 \circ f_1)(\boldsymbol{x}^{(0)})$ the derivative has the form

$$f'(\boldsymbol{x}^{(0)}) = f'_{L+1}(\cdots)\, f'_L(\cdots) \cdots f'_1(\boldsymbol{x}^{(0)}).$$

The aim of this exercise is to explore this problem.

Consider a neural net as introduced in the course with L layers, $K = 3$ (i.e. number of nodes per layer), and the sigmoid function as activation layer (i.e. $\boldsymbol{x}^{(l)} = f_l(\boldsymbol{x}^{(l-1)}) = \phi((\boldsymbol{W}^{(l)})^\top \boldsymbol{x}^{(l-1)} + \boldsymbol{b}^{(l)})$). Assume that all weights $W^{(l)}_{i,j}$ are bounded, lets say $|W^{(l)}_{i,j}| \leq 1$. Consider a regression task where the output layer (i.e. f_{L+1}) has a single node representing a simple linear function with some bounded weights c_i, lets say $|c_i| \leq 1$. Hence the overall function, call it f is a scalar function on $\mathbb{R}^D$. Show that the derivative of this function with respect to the weight $W^{(1)}_{1,1}$ vanishes exponentially fast as a function of L at a rate of at least $(\frac{3}{4})^L$.