Ecole Polytechnique Fédérale de Lausanne

Deep Learning

EE-559

Project 2 Mini deep-learning framework

Authors:
Ertuğrul Mert
De Lima Carvalho Luis
Gaiffe Raphaël

May 22, 2020

1 Introduction

The goal of this project is to implement a mini deep learning framework without using autograd or the neural-network modules given with PyTorch. The deep learning framework should provide the tools to build networks, run the forward and backward passes and optimize the parameters using SGD for MSE.

2 Structure and modules

The framework consists of a group of modules inheriting a model parent and a few functions. As suggested we used the *class Module* as a template for all of our modules and those are:

1. Activation functions hyperbolic tangent (*Tanh*), rectified linear unit (*ReLu*), leaky rectified linear unit (*LeakyReLu*) and the sigmoid activation function (*Sigmoid)*
2. Mean squared error loss (*LossMSE*) and mean absolute error loss (*LossMAE*)
3. Stochastic gradient descent optimizer (*SGD*)
4. Fully connected layer creator (*Linear*)
5. A module to execute a sequence of modules for the creation of the model (*Sequential*)

The test file *test.py* is where we generate the data, normalize it, split it, train the model and eventually test it. We opted to implement three helper functions inside test.py instead of adding it to framework because we wanted the framework to be focused on deep learning operations and not a general helpers.py file. These functions are :

1. *generate_norm_data*: it generates a dataset sampled from a uniform distribution within $[0, 1]^2$ and normalizes it
2. *labels_conversion*: the generated targets are either 0 or 1 and here we convert it to a tensor of labels of respectively [1,-1] and [-1,1] that we can easily retrieve 0 or 1 from the index or the maximum value.
3. *data_split*: it splits the data into a training, validating and testing data. The partition of each group is inside the function as constants and not as parameters to ensure a proper division.

Concerning storage, the forward passes cache values as instance variables so that it is easier and more convenient when doing the backward pass. Each layer sends back it's gradient calculation to be multiplied with the earlier layer's gradient.

2.1 Activation Functions

The main activation functions were implemented with the following formulas using tensor operations from Pytorch. Here are the formulas used for hyperbolic tangent:

$$\text{forward} : x \mapsto \frac{2}{1 + e^{-2x}} - 1$$

$$\text{backward} : x \mapsto \frac{4}{(e^{-x} + e^{x})^2}$$

Here are the formulas used for ReLU:

$$\text{forward} : x \mapsto max(0, x)$$

$$\text{backward} : x \mapsto \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases}$$

2.2 Fully Connected Layers and Sequence

The goal was to generate a model of two inputs, two outputs and three hidden layers

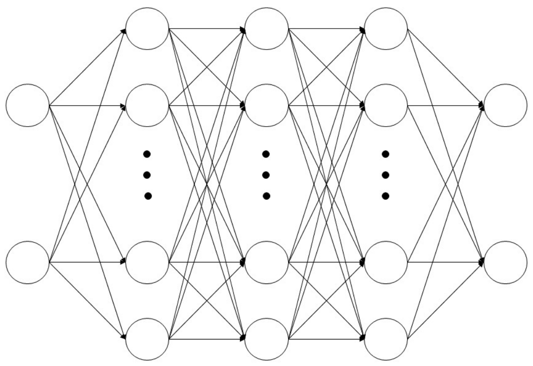


Figure 1: Model layers representation

The Sequential module allows us to create a model by using the Linear module to create the layers in combination with one of the two activation functions. The following sequence creates one layer of two inputs, one of two outputs and a hidden layer with N units (25 in our project) with their respective chosen activation functions:

$$\text{Linear(2,N)} \rightarrow \text{Tanh()} \rightarrow \text{Linear(N,2)} \rightarrow \text{ReLu()}$$

2.3 Optimizer

In order to reduce the burden of calculation, the iterative stochastic gradient descent is used to update the modules parameters minimizing the loss. The SGD uses has a learning rate parameter η. The optimizer does a so called "step" after the backward pass, iterating through the parameters of each module and updating them according to the gradient with respect to them weighted by the learning rate. The step is as follows:

$$w_{t+1} = w_t - \eta \nabla \mathscr{L}(w_t) \tag{1}$$

A *zero_ grad()* method was implemented in the SGD module in order to clear the gradients of the layer before every backward pass.

2.4 Loss

The loss function used in this project is the Mean Squared loss and here is the formula used for its calculation and also its derivative:

$$MSE(\hat{y}, y) = \frac{1}{N_{\text{out}}} \sum_{i=1}^{N_{\text{out}}} (\hat{y}_i - y_i)^2 \tag{2}$$

$$MSE'(\hat{y}, y) = 2 \times (\hat{y} - y) \tag{3}$$

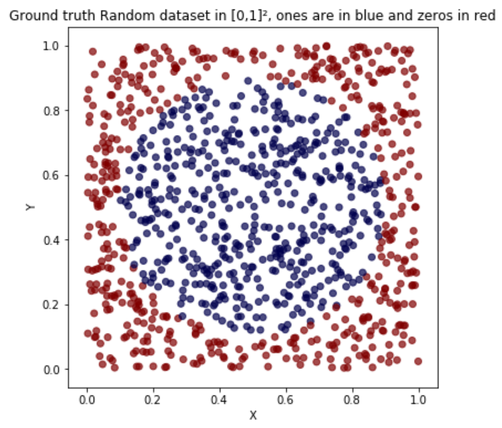
We also implemented the mean absolute error loss (MAE) with the following formulas:

$$\text{MAE}(\hat{y}, y) = \frac{\sum_{i=1}^{n} \hat{y}_i - y_i}{n} \tag{4}$$

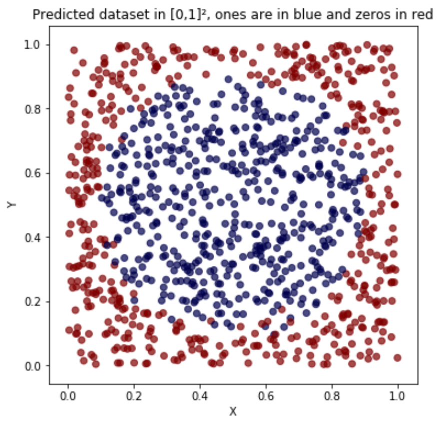
$$\text{MAE}'(\hat{y}, y) = \begin{cases} +1, & \hat{y} > y \\ -1, & \hat{y} < y \end{cases} \tag{5}$$

3 Conclusion

We will now conclude with results from the prediction of our model here below:



(a) Ground truth



(b) Prediction

Figure 2: Comparison of the dummy dataset with the prediction or our model. Around 96% accuracy

We also tested different learning rates and their impact on the accuracy of predictions according to the number of epochs:

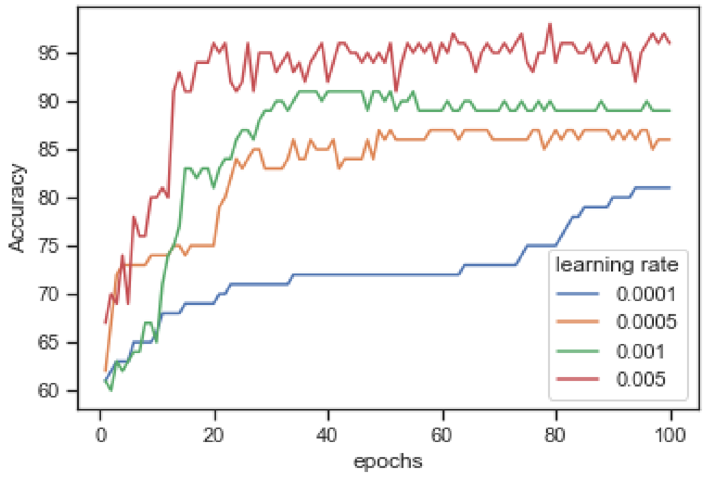


Figure 3: Accuracy of validation data in function of the learning rate

Overall this project showed us how useful PyTorch actually is with all the already implemented helper functions and very convenient methods. It was indeed a nourishing experience to better understand what is happening behind PyTorch and also the course itself.