ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Deep Learning
EE-559

Implementing from scratch a mini deep-learning framework

Groupe 36:
Victor Faraut
Martial Bernard-Michel
Mickaël Bressieux

May 11^{th} **2018**

1 Introduction

The goal of this project was to implement from scratch a framework for deep-learning usage like Pytorch using only the math library and already implemented function for Tensor operations. This means that we had to implement modules capable of building networks with connected layers, applying a certain activation function, running the forward and backward pass of a certain transformation, optimizing its parameters.

In our case, we implemented the Tanh and ReLU activation function, the Mean Square error criterion, the Linear transformation and a Sequential container of modules.

We'll first introduce what were the project statements, then the structure of the framework and the mathematical formulas on which it is based. We'll finish with a quick analysis of our results.

2 Project statements

2.1 Dataset

The training and testing set are composed of 1'000 points each, sampled uniformly in $[0, 1]^2$. As for the classes, if the point is outside the disk of radius $1/\sqrt{2\pi}$ and centered at $[0.5, 0.5]$, then it is labeled 0, else it is labeled 1.

In our implementation, the target is a matrix of two column. The index of the column containing 1 is the label of the point.

2.2 Network

The network must have two input units, two output units and three hidden layers of 25 units.

2.3 Module function

- **forward**: should get as input a Tensor containing the data to transform and return also a Tensor containing the transformed data.
- **backward**: should get as input a Tensor containing the gradient of the loss with regard to the module's output, accumulate the gradient with regard to the parameters and return a Tensor containing the gradient of the loss with regard to the module's input.
- **parameters**: should return a list of pairs containing the parameter Tensor and gradient Tensor of the same size. Needs to be empty if the module is parameterless.

3 Explanation of the Framework

3.1 Base Module

This class is inherited by all the other classes (except Parameter of course). It is the common point between the other classes. This is why, the functions unique to the future classes that inherits 'Module' needs to be re-implemented in the latter but the functions common to all of them are already implemented here.

3.1.1 Add_parameter

Regarding the $'add\_parameter'$ function, we differentiate two types of input. The first one, 'Parameter', represents the case when you want to initialize the parameter in the current Module. The second one, 'list', represents the case when you want to add all the parameters of a module to another. This is the case for example when we want to add the parameters of a Linear module to the Sequential module.

3.1.2 Optimizer

The optimization step we implemented for this project is based on the Stochastic Gradient Descent which corresponds to the following formula:

$$param^{(l)} \leftarrow param^{(l)} - learning\_rate * \frac{\partial(loss)}{\partial(param^{(l)})} \tag{1}$$

3.2 Criterion

The criterion we're using to train our model is the Mean Square Error. The forward pass is the actual computation of the MSE with the following formula:

$$MSE = \sum_{i=0}^{n} (Output - Target)^2 \tag{2}$$

The backward pass is based on the derivative of the previous formula with regard to the output. This gives:

$$\frac{\partial(MSE)}{\partial(Output)} = 2 * (Output - Target) \tag{3}$$

3.3 Activation functions

3.3.1 ReLU

The Rectified Linear Unit is based on the simple formula:

$$x \mapsto max(0, x) \tag{4}$$

Again, the backward pass is based on the derivative:

$$dx = \begin{cases} 0 \text{ if x} < 0 \\ 1 \text{ otherwise} \end{cases} \tag{5}$$

3.3.2 TanH

The forward pass returns simply the hyperbolic tangent of the input:

$$x \mapsto tanh(x) \tag{6}$$

The derivative of tanh is used for the backward pass and is given by the formula:

$$d\{tanh\}(x) = 1 - tanh^2(x) \tag{7}$$

3.4 Linear Module

We put a counter of Linear object in this class in order to name properly the parameters ('weights0' for the first object, 'weights1' for the second, etc...). This is exclusively used for distinguishing the parameters of the different modules when debugging.

3.4.1 Init_parameters

The weights and bias are initialized with a normal distribution around 0 and with a standard deviation of 'epsilon', value given by the user. In our case, we used $\epsilon = 0.1$. After that, their corresponding gradient are put to 0. Those parameters are then added to the list 'param' of the Linear object.

3.4.2 Forward pass

A linear transformation is given by the formula:

$$y = x \cdot W + b \tag{8}$$

with W the weights and b the bias. The size of the weights and the bias depends on the size of input and output feature given by the user.

3.4.3 Backward pass

First, the backward pass computes the derivative of the loss with regard to the activations:

$$\frac{\partial(l)}{\partial(x^{(l)})} = \frac{\partial(l)}{\partial(s^{(l+1)})} \cdot (w^{(l+1)})^T \tag{9}$$

Then, it computes the derivative of the loss with regard to the parameters:

$$\frac{\partial(l)}{\partial(w^{(l)})} = (x^{(l-1)})^T \cdot \frac{\partial(l)}{\partial(s^{(l)})} \tag{10}$$

$$\frac{\partial(l)}{\partial(b^{(l)})} = \frac{\partial(l)}{\partial(s^{(l)})} \tag{11}$$

This last equation is implemented by multiplying the derivative of the loss with regard to the output by a matrix with the same number of row but only one column. This allows to have the proper size in order to accumulate the gradients.

3.5 Sequential Container

This module is used to create models with connected layers. It conserves the order given by the user. It creates the objects and then add their parameters to the Sequential module.

The forward pass is simply the succession of the forward pass of the objects of the sequential. The same goes for the backward pass.

4 Main Function

The data generated for the train and test is normalized before being used.

We use different parameters (that can be modified by the user) in order to train the model, they are the following:

- mini_batch_size: Size of the batch of data trained at the same time.
- nb_epochs: Number of times the model is trained.
- learning_rate: value indicating how important the new beliefs are over the old parameters.
- standard_deviation: standard deviation used to initialize the parameters of the Linear objects.
- hidden_layer: number of input features and number of output features for the middle layer.

In order to test all the modules implemented, our model is a sequential with three linear layers, each linear layers being separated by a different activation function.

The error computation is a simple comparison between the index of the column with the maximum value of the output and of the target.

5 Results

On average, those parameters and model result in an error of 2.5% on the training dataset and an error of 3.0% for the testing dataset (over 10 training sessions). The best results we obtained in this configuration are 0.9% and 1.9% for the training and testing dataset respectively.

The following figure shows the plot of the loss at each iteration of the training process in this configuration:

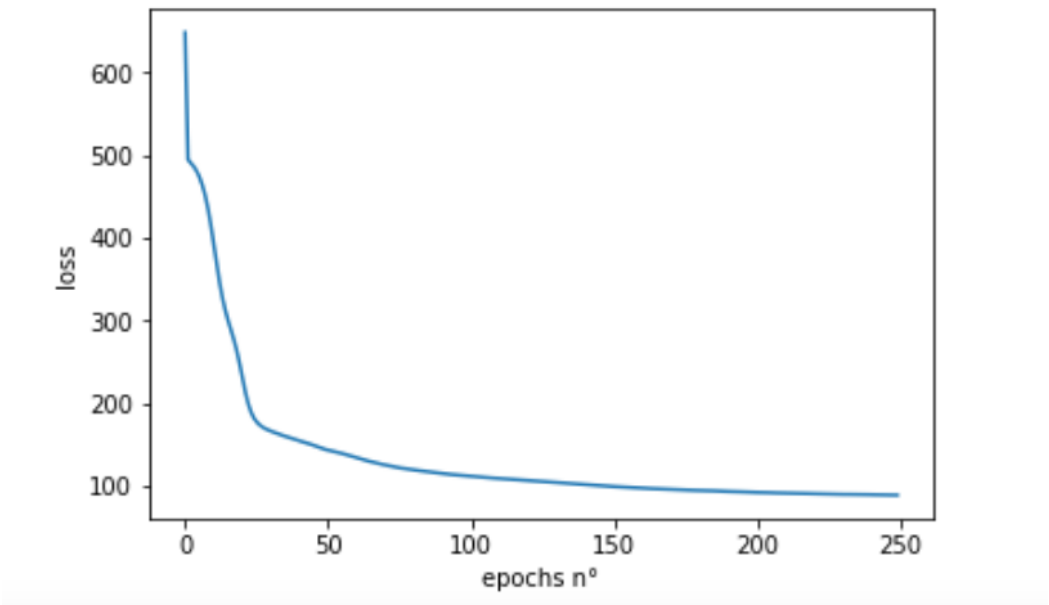

Figure 1: Plot of the loss at each iteration of the training process

The initial loss value is 648.19 and the final one is 88.62. The resulting errors are 2.2% and 3.1% for respectively the training and testing dataset.

6 Conclusion

This project was very challenging and at least as much interesting, especially since we are not very accustomed to this language (no python course in the Micro-engineering and Mechanical-engineering). The most difficult part of it was to make the classes as modular as possible. It really helped us to sharpen the knowledge we got from the course. In the end, we are very satisfied with the results we got.