Deep Learning – Mini project 2

Alexandre Di Piazza, Elodie Adam, Clémentine Lévy-Fidel

I. Introduction

The aim of this project was to implement from scratch a mini deep learning framework allowing to construct simple neural networks. Only the tensor operations provided by Pytorch and the basic math library were allowed. A simple framework must provide at least the following functionalities:

- a module allowing to build networks combining fully connected layers with different activation functions
- run the forward and backward passes to update the weights and bias during training
- optimize parameters with SGD for MSE or MAE

A mini deep learning framework is usually composed of theses different modules: operators, optimizers and loss functions. In order to be able to test the model, a function that trains the model, tests it and allows to appreciate its performance was implemented.

II. Data

The training and testing input dataset is composed of 1, 000 points sampled uniformly in $[0, 1]^2$. The label associated to this dataset is 0 if the point is outside the disk centered at (0.5, 0.5) of radius $1/\sqrt{2\pi}$, and 1 if inside, as shown in Figure 1.

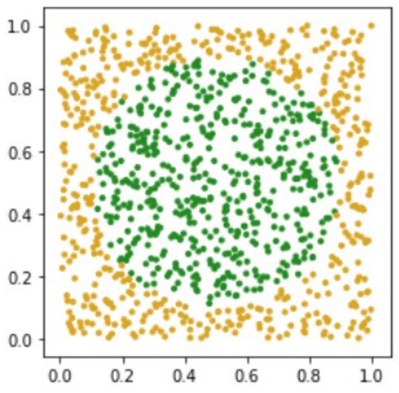

Fig. 1. Visualization of the dataset

III. Modules

Our model consists of a sequential container of different operators and activation function. It is implemented within the class Model. This class and ll operator, optimizer and loss function classes can be found in the file *framework.py*. To launch a simulation, run *test.py*. It trains a model once and prints the training accuracy at each epoch as well as the number of true positives, true negatives, false positives and false negatives on the test data set after the model has been trained.

IV. Operators

The following formulas were used for the different activation functions:

1) ReLU:
   - Forward pass:
$$x \rightarrow max(0, x)$$
   - Backward pass:
$$x \rightarrow heaviside(x) = \begin{cases} 0 \text{ if } x < 0 \\ 1 \text{ if } x \geq 1 \end{cases}$$
2) Tanh:
   - Forward pass:
$$x \rightarrow \frac{2}{1 + \exp(-2 * x))} - 1$$
   - Backward pass:
$$x \rightarrow 1 - tanh(x)^2$$
3) Sigmoid:
   - Forward pass:
$$x \rightarrow sigmoid(x) = \frac{1}{1 + \exp -x}$$
   - Backward pass:
$$x \rightarrow 1(1 - x)$$

In order to connect the input and output of all the hidden layers, a module Linear was implemented.

V. Optimizer

The optimizer used in the framework is a stochastic gradient descent (SGD) algorithm, which is one of the most common method to optimize neural networks. It consists in updating the parameters of the network in the opposite direction of the loss function with respect to the parameters. The size of the steps used to reach a local minimum is defined by the learning rate. At each step, the following formula is used to update the weights:

$$w_{s+1} = w_s - \gamma \cdot \nabla L(w_s)$$

with γ the learning rate and $\nabla L(w_s)$ the gradient of the loss function with respect to the parameters.

VI. Loss Functions

The framework allows the user to choose between two loss functions: MAE and MSE. MAE is defined in the following manner:

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_{pred} - y_{true}|$$

And MSE is given by the following expression:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_{pred} - y_{true})^2$$

VII. Architecture

The network implemented in the test.py file is composed of two input units, one output unit and three hidden layers of 25 units. The layers are sequenced in the following manner:

$$Linear(2,25) \rightarrow ReLU \rightarrow Linear(25,25)$$
$$\rightarrow Tanh \rightarrow Linear(25,25) \rightarrow ReLU \rightarrow Linear(25,1)$$

The chosen loss function for training was MSE, the learning rate was 0.01 and the training time selected was of 50 epochs.

VIII. Results

The model was trained and the training error was calculated at each epoch, as shown in Fig. 2. The final accuracy obtained at the last epoch was of 94%, after 9 seconds of training time.

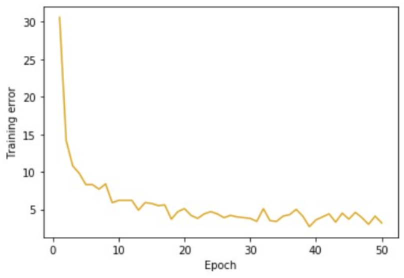

Fig. 2. Train error evolution

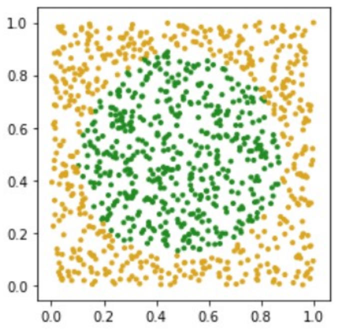

Fig. 3. Prediction obtained on the test dataset

Once the model was trained, we used it over the same data set shown in Fig. 1 and it gave the prediction shown in Fig. 3. We can assert that the model performs remarkably well at reproducing the boundary.

IX. Discussion

The mini framework implemented allows to explore different networks architecture using combinations of simple modules. It permits to have a better understanding of how the different layers are connected, how the backward/forward pass works or how the parameters are updated with SGD. The performance of the model using the modules implemented from scratch was relatively good. In order to offer a more complete deep learning framework, some more modules could be implemented in the future such as Adam optimizer or Logarithmic loss.