Project 2 of Deep Learning: Mini deep-learning framework

Yann Bouquet, 273827, yann.bouquet@epfl.ch, Master in Data Science,
Thomas Fry, 280922, thomas.fry@epfl.ch, Master in Computational Science and Engineering,
EPFL

***Abstract*—The project is the design of a mini deep learning framework that uses only Pytorch's tensor operations without needing autograd or neural network modules from Pytorch. After defining all the modules implemented, we analyse the performance of models created and trained with this framework by randomly generating points in a 2D space.**

I. Introduction

This is the design of a mini deep learning framework which would enable to create a Multilayer Perceptron with ReLU or Tanh activation and a training process with Stochastic Gradient Descent on a Mean Squared Error Loss so that any autograd process or neural network modules offered by Pytorch can be disposed of. Are provided the diagram of the architecture and some mathematical explanations about all the modules. Finally a training and a test sets are generated, on which the networks created thanks to this framework will be trained for classification.

II. Modules

The unit (Module) of the framework consist of linear layers (Linear) with non-linear activation function (Tanh, ReLU) handling forward and backward propagation. The training of this module is based on a loss function (LossMSE) and an optimization algorithm (SGD). The unit (Sequential) is to append and handle several layers for a network.

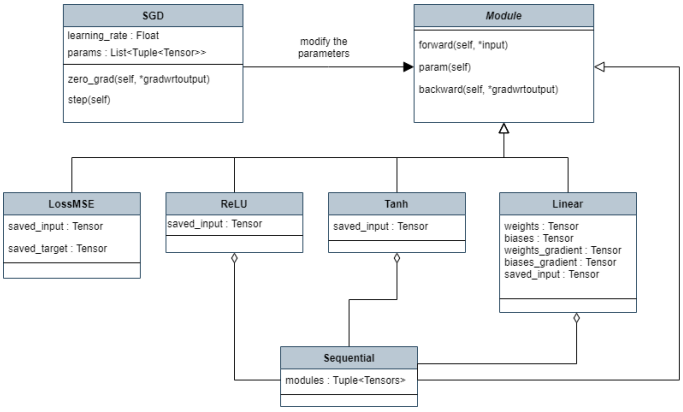

Fig. 1: Class diagram representing the framework

A. Mean Squared Error

1) Forward: The loss function used in this framework is the Mean Squared Error, defined as :

$$l = MSE(y, \hat{y}) = \frac{1}{N}\sum_{n=1}^{N}(y_n - \hat{y}_n)^2$$

with $y \in \mathbb{R}^N$ is the target and $\hat{y} \in \mathbb{R}^N$ is the output of the model.

Moreover batches of samples should be considered as input of the modules. For $Y \in \mathbb{R}^{K,N}$ a matrix of K targets with N features and $\hat{Y} \in \mathbb{R}^{K,N}$ the equally shaped output, the MSE Loss becomes:

$$l = MSE(Y, \hat{Y}) = \frac{1}{KN}\sum_{n=1}^{N}\sum_{k=1}^{K}(Y_{k,n} - \hat{Y}_{k,n})^2 \qquad (1)$$

2) Backward: As a consequence the derivative of the MSE according to equation (1) is the following:

$$\frac{\partial l}{\partial \hat{Y}} = \frac{2}{KN}(\hat{Y} - Y)$$

B. ReLU

1) Forward: The first considered activation function is the ReLU activation. Activation functions are element-wise operations. So for $x \in \mathbb{R}$, ReLU is defined as:

$$\sigma(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

During the forward, ReLU is called on $s = wx + b$ where x is the input of a layer, w the weights and b the bias of the linear layer:

$$x^{layer} = \sigma(s) = \sigma(w^{layer-1}x^{layer-1} + b^{layer-1})$$

2) Backward: The derivative of ReLU is:

$$\sigma'(x) = sign(\sigma(x)) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

thus if $s^{layer} = w^{layer}x^{layer-1} + b^{layer}$ then :

$$\frac{\partial \sigma}{\partial w^{layer}} = x^{layer-1}\sigma'(s^{layer})$$

C. Tanh

Tanh, similarly to ReLu, is also an activation function defined as:

$$\sigma(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

$$\sigma'(x) = 1 - \sigma^2(x)$$

D. Full Connected Layer

1) Initialization: The weight initialization followed a normal distribution, the $Xavier$ initialization $\mathcal{N}(0, \sqrt{\frac{2}{N_i+N_o}})$, where N_i and N_o are the number of features for input and output respectively. Likewise biases were initialized with a normal distribution $\mathcal{N}(0, 1)$.

2) Forward: During the forward propagation, considering a linear layer $layer$, we have :

$s \in \mathbb{R}^J$, $w \in \mathbb{R}^{J,I}$, $x \in \mathbb{R}^I$, $b \in \mathbb{R}^J$, so that the output of the layer is:

$$s = wx + b$$

If input and output are batches of size K, there is:

$S \in \mathbb{R}^{K,J}$, $W \in \mathbb{R}^{J,I}$, $X \in \mathbb{R}^{K,I}$, $B \in \mathbb{R}^{K,J}$ where a line of B, $B_i = b \ \forall i \in [1, K] \cap \mathbb{N}$

$$S = XW^t + B$$

3) Backward:

$$s = wx + b$$

Let l the loss, for the weights:

$$\frac{\partial l}{\partial W^{layer}} = \frac{\partial l}{\partial S^{layer}} (X^{layer-1})^T$$

with batches:

$$\frac{\partial l}{\partial W^{layer}} = (\frac{\partial l}{\partial S^{layer}})^T X^{layer-1}$$

Let l the loss, for the biases:

$$\frac{\partial l}{\partial B^{layer}} = \frac{\partial l}{\partial S^{layer}}$$

with batches:

$$\frac{\partial l}{\partial B^{layer}} = \sum_{k=1}^{K} \frac{\partial l}{\partial S_k^{layer}}$$

Backward for the layer $layer$-1, return:

$$\frac{\partial l}{\partial X^{layer-1}} = (W^{layer})^T (\frac{\partial l}{\partial S^{layer}})$$

$$\frac{\partial l}{\partial X^{layer-1}} = (\frac{\partial l}{\partial S^{layer}}) W^{layer}$$

E. Stochastic Gradient Descent

The 'Vanilla' Gradient Descent, with w a weight of the network and η the learning rate of the algorithm, updates w by:

- computing g the gradient of w according to the batch with the backward method
- subtracting $w \leftarrow w - \eta g$

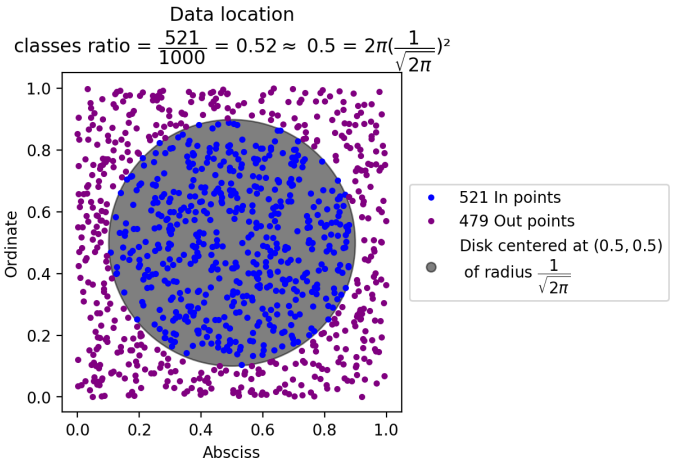

Fig. 2: Example of a training set distribution

III. Data

In order to test the framework, a network will be applied on a train and a test set of 1, 000 points sampled uniformly in $[0, 1]^2$, each with a label 0 if outside the disk centered at (0.5, 0.5) of radius $\frac{1}{\sqrt{2\pi}}$, and 1 inside. As a consequence the probability for a point to be in the circle is equals to 0.5. This distribution gives an unbiased set to the network.

IV. Results

The model used to test the framework consists of two input units, two output units, three hidden layers of 25 units. The performances are given for two version of this model: one with ReLU activations between the linear layers, the other with Tanh activations. We run the training and testing process with 5000 epochs and a batch size equal to 100. The learning rate chosen for the SGD algorithm is 10^{-2}.

1) Validation: Before applying the models on the test set, let us analyze them on a validation set taken from the training set. For the validation we randomly selected 20% of the generated points for the validation set and kept the other 80% as the training subset.

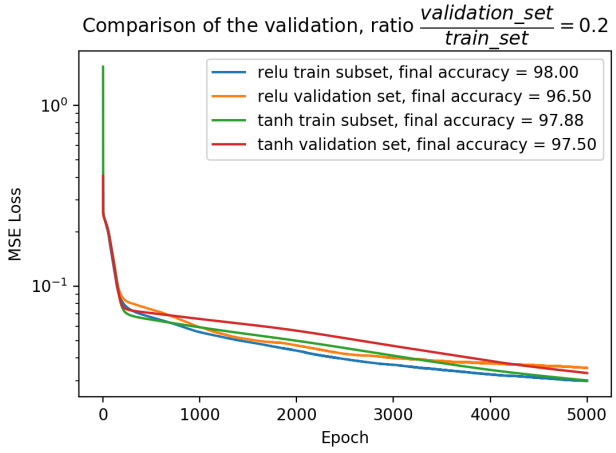

Fig. 3: Performance of the models through the epochs on the training and validation sets

Tanh model performs better on both the training and the test set.

Models	Mean train error	Standard deviation train accuracy	Mean test error	Standard deviation test accuracy
ReLU	1.12	0.61	1.77	0.65
Tanh	0.84	0.55	1.17	0.65

TABLE I: Comparison between ReLU and Tanh

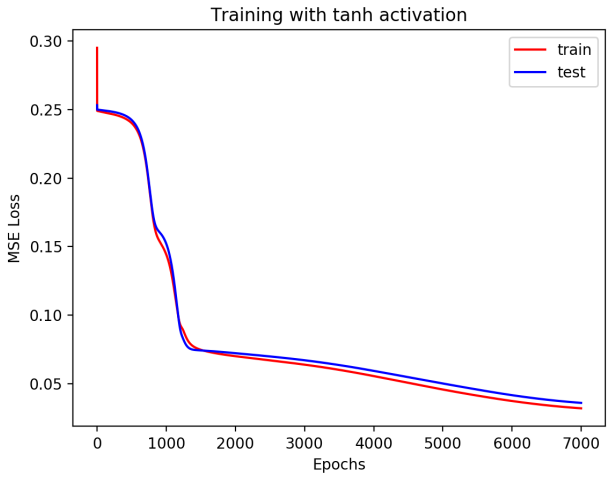

Fig. 4: Training process for the full connected layer with Tanh activation

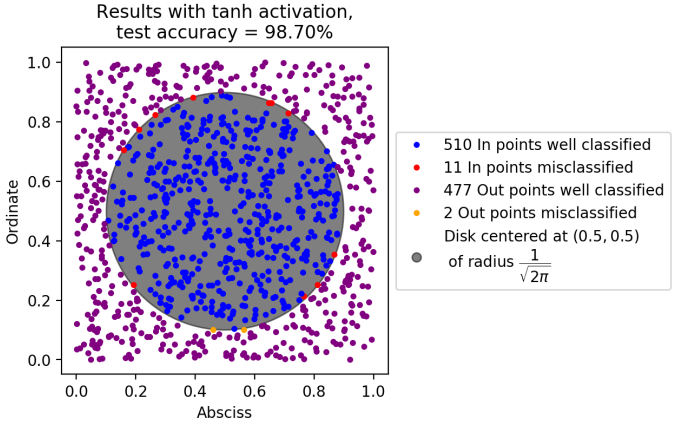

Fig. 5: Classification of the Tanh model

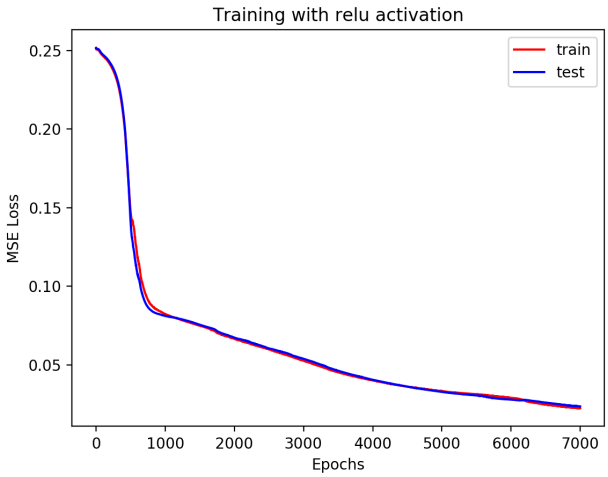

Fig. 6: Training process for the full connected layer with ReLU activation

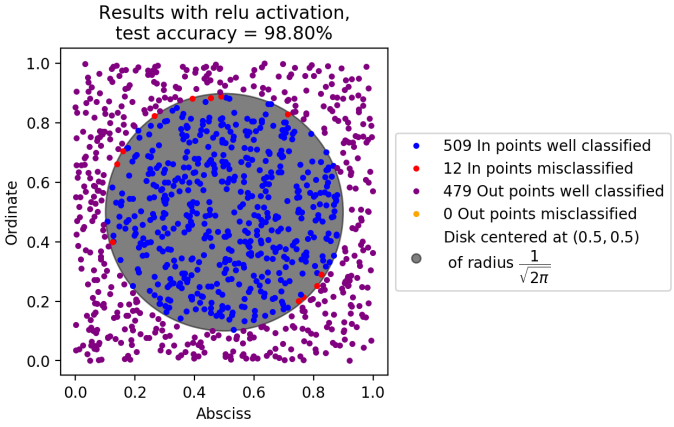

Fig. 7: Classification of the ReLU model

ReLU performs better on the train set, but it shows some overfitting. Both models are very efficient.

V. Conclusion

This project helped us learn to build a neural network from scratch, combining full connected layers, running forward and backward passes, training the model for classification on a specific dataset and optimizing its parameters with SGD for MSE. We compared different activation functions and their performances. This project gave a better understanding of the calculations behind deep learning. Finally the framework is functional, efficient, easy-to-handle and modular for improvements to come as new optimizers, auxiliary losses or criterions.