Mini Deep Learning Framework without Autograd

Kang Fu, Hangqi Li, Junhong Li
EPFL Switzerland

***Abstract*—The aim of this report is to develop a mini deep learning framework without using the PyTorch module of autograd. We can use basic Tensor module of PyTorch to construct the layers and optimizers of neural network, which can be used to solve numerous machine learning problems. In this report, the framework is used to solve a simple binary classification task and evaluate this framework with accuracy metrics.**

I. Introduction

The neural network implemented in this report combines multiple layers and use the forward and backward pass to update the parameters, which is optimized by stochastic gradient descent. After that, this model is tested on a 1000-point dataset with classification task. In the following section, we will cover the structure of the framework and explain the mathematical concepts of the neural network. Finally the performance of the framework will be evaluated.

II. Structure of the Framework

The framework consists of different modules such like parameter, linear module, sequential module, activation functions, loss function and optimizer. The weight and bias with their gradients are stored in parameters. All modules are classes inherited from the same base class Module with a constructor, forward, backward methods. The forward method should get an input tensor and output the simple linear calculation. The backward method gets the tensors of the gradient with respect to the output as input and return the tensors of the gradient with respect to the modules' input.

The linear class implements the linear layer. Then output will be applied to activation layers, defined by different activation class. The repeated process forms the forward process. Then the backward method takes the gradient of loss and propagates it backward of layers in the reversed order.

The optimizer class (SGD) implements the stochastic gradient descent to minimise the loss function defined by MSEloss class. In every epoch, the bias and weight will update with respect to the learning rate and the gradients.

III. Layers

The layer is the connection between the neurons. There are three kinds of layers in neural network. The input layer and output layers are specified by users. The hidden layer will perform a black box connecting the input and output. In this framework, fully connected layers are used, which means all the inputs from one layer are connected to every activation unit of the next layer.

A linear layer takes a batch of samples as input and outputs the transformed values by weights and biases. In this layer, the initial weights and biases are chosen on the uniform basis. The function of the linear layer is to calculate the values of each neuron. The output is the input multiplied by weights and then add the biases in the forward process. The gradients are computed in the backward process.

The sequential class is used to connect different layers (linear layers and activation layers in this project) in a sequential way.

IV. Activation Functions

The activation function in neural network defines the output of a node given an input or set of inputs. It can introduce the non-linearity to the network. Each layer without activation function is equivalent to matrix multiplication. Even if you stack several layers, it is nothing more than a matrix multiplication and and it can not have a good performance. There are many activation functions and in this project, Rectifier Linear Unit (ReLU), Sigmoid and Hyperbolic Tangent (Tanh) are covered in the framework.

1) ReLU: ReLU is an activation function defined as the positive part of its argument. If the input is positive, the output will be identical as the input, otherwise it outputs 0. During the backward process, it will keep the positive parts.

$$ReLU(x) = \begin{cases} 0 & x < 0 \\ x & x \geq 0 \end{cases}$$

$$\frac{d(ReLU(x))}{dx} = \begin{cases} 0 & x < 0 \\ 1 & x \geq 0 \end{cases}$$

2) Sigmoid: The Sigmoid activation function maps all the inputs to the range of 0 and 1. Since the output is in the range [0,1], it is frequently used in the binary classification task with a threshold in the sigmoid function. There are different drawbacks of Sigmoid: one is that it has a slow respond to changes of the input when the input is too large or small because the gradient are near zero. Thus, it will cause the vanishing gradient problems and make it hard to converge.

$$\sigma(x) = \frac{1}{1+e^{-x}}$$
$$\frac{d(\sigma(x))}{dx} = \sigma(x) * (1 - \sigma(x))$$

3) Tanh: Tanh activation function is also like logistic sigmoid function but can have a better performance. The range of the Tanh function is from -1 to 1. The advantage is it has a fast response to the change of input when the input is near 0. Similar to sigmoid function, this activation function is mainly used classification between two classes.[1] But it could also have the vanishing

gradient problem. The difference is that the derivatives of the Tanh are larger than the derivatives of the sigmoid, which means the cost function minimizing speed is faster than the sigmoid (see Figure 1).

$$Tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$
$$\frac{d(Tanh(x))}{dx} = 1 - Tanh(x)^2$$

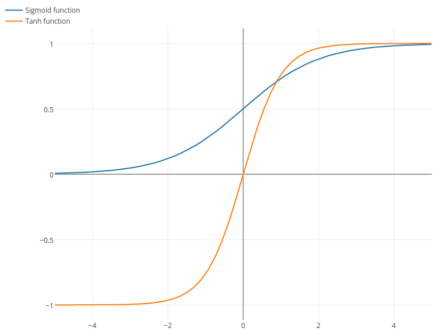

Fig. 1: sigmoid and tanh

V. Loss Function

The loss function is a metric that will be optimized during the training process, which is generally composed of the difference of the predicted value and target value. The goal of the neural network is to minimise the loss and increase the accuracy. In this project, mean square loss function is constructed, which is the mean square of the difference of the predicted value and target value.

$$L(y, x) = \sum_{i=1}^{N} (y_i - x_i)^2$$
$$\frac{d(L(y_i, x_i))}{dx_i} = 2 * (y_i - x_i)$$

VI. Optimizer

Optimizer is used to optimize the parameters of the neural network to minimise the loss function by updating the weights and biases. In this framework, stochastic gradient descent (SGD) algorithm is used. SGD picks one batch size of data point from the whole dataset, and repeats it until the whole dataset has calculated at each iteration to reduce the computations enormously.

$$\theta = \theta - \eta * \nabla L(\theta; x_i, y_i)$$

where θ is the parameter to be optimised and ∇ is the learning rate.

VII. Dataset

The dataset is generated by a uniform distributions. The training and testing sets are 1000 points uniformly distributed in $[0, 1]^2$. Points that are outside a circle of radius $1/\sqrt{2 * \pi}$ centered at (0.5,0.5) are labelled as 0, while inside the circle are labelled as 1. In order to compute the target label, we can use the sign function to get the binary label. The formula should be the following:

$$sign(\frac{1}{2*\pi} - ((x - 0.5)^2 + (y - 0.5)^2))$$

The area of circle is 0.5 and it means there is 0.5 probability of a random point labelling 1. Therefore the dataset are balanced and the accuracy is a proper metric to evaluate the performance of our framework.

VIII. Model Training

The model has 2 inputs, 1 output and 3 hidden layers of 25 units. We have proposed three models with different activation layers. The three model are demonstrated as Figure 2. As for the model 3, when all the activation function are Sigmoid function, it cannot converge in the end. So the Tanh is chosen as the first activation function in the model 3. The possible reasons have been discussed in the previous comparison of activation functions.

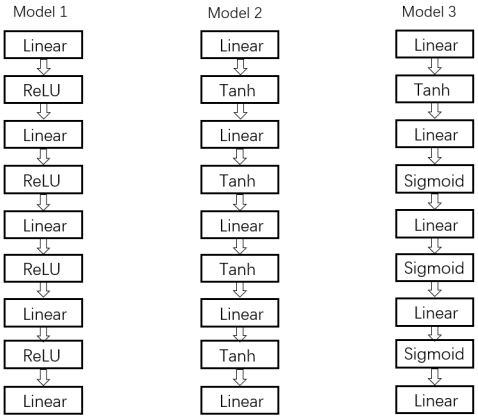

Fig. 2: The model demonstration

The network are trained in 250 epochs and the batch size is 50. In the framework, the learning rate is not constant. When the training begins, the learning rate is 0.1. After 100 epochs, the learning rate becomes 0.03. Then after 175 epoch, the learning rate becomes 0.01. The learning rate controls how quickly the model is adapted to solve the problem. Generally, a large learning rate allows the model to learn faster, at the cost of accuracy. It will arrive a local minimal, not the global minimal. A smaller learning rate may allow the model to learn a more optimal or even globally optimal set of weights but may take significantly longer time to train. Here we combine both advantages of them to make "adaptive" learning rates.[2]

The table 1 has shown the train accuracy, test accuracy and training time. It can be seen that model 1 and model 2 can achieve a high training accuracy and test accuracy while model 3 have a relative bad performance because it will take much longer time and large epochs for sigmoid activation functions to converge, which is around 1400 epoch. Figure 3, 4 and 5 has displayed the predicted output.

Models	Train Accuracy	Test Accuracy	Train Time (Test in VM)
Model 1	0.989	0.978	0.9 min
Model 2	0.968	0.959	0.79 min
Model 3	0.51	0.517	0.7 min

TABLE I: Comparison between different models

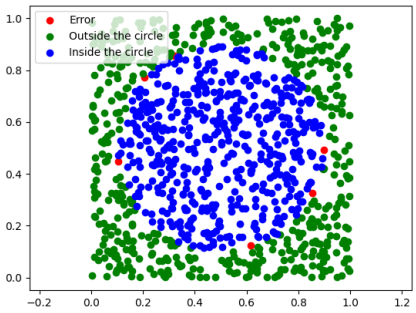

Fig. 3: The model 1 prediction result

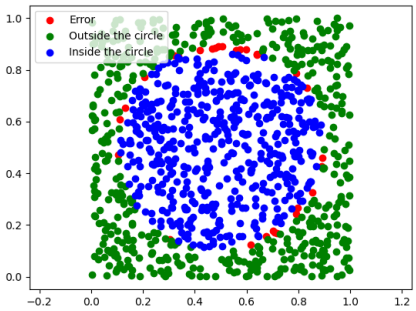

Fig. 4: The model 2 prediction result

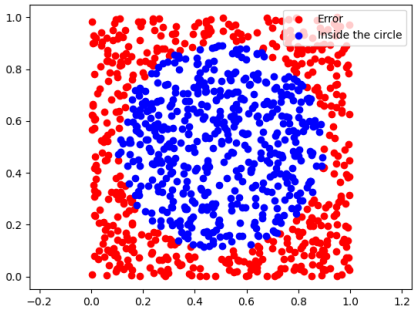

Fig. 5: The model 3 prediction result

IX. Conclusion

In conclusion, the proposed framework can basically solve the binary classification task with high accuracy. Although the Sigmoid function will take a much longer epoch than RELU, Tanh to converge. In the future, the framework can be improved by adding more layers such as pooling, more parameter initializing methods like Xavier Initialization, and other loss functions like cross-entropy or softmax, etc.

References

[1] S. Sharma, "Activation functions in neural networks," https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6, Sep 2017.

[2] J. Brownlee, "Understand the impact of learning rate on neural network performance," https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-ne Jan 2019.