Project 2 - Mini deep-learning framework

Pengkang GUO, Xiaoyu LIN
École Polytechnique Fédérale de Lausanne, Switzerland

I. Introduction

The task of this project is to implement a mini deep learning framework using only Pytorch's tensor operation and standard math library to deepen the understanding of neural network mechanisms. The framework must be able to build a network that combines the fully connected layer and the activation ($Tanh$ and $ReLU$) layer and train the network using mini-batch Stochastic Gradient Descent (SGD). We use this framework to build and train a multi-layer perceptron (MLP) network to test the framework. We are sharing the code used to run the experiments in this report at https://github.com/DoubleMuL/DeepLearningProject2.

II. Methodology

In this section, we will discuss the fundamental components in our MLP network in detail. A complex deep network always contains several blocks for different functions, and each block is specially designed with different layers and activation functions. However, for this simple binary classification task, we use a simple MLP model, which only contains fully connected linear layers and two types of activation functions: $ReLU$ and $Tanh$.

A. Linear layer

The linear layer takes three input arguments: the number of input dimension n, output dimension m and a Boolean to tell whether the bias is added. It applies a linear transform on input data. As we will see in Section III-B, we will use mini-batch SGD algorithm for optimization. Suppose the batch size is N, the input data is $\mathbf{X} \in \mathbb{R}^{N \times n}$, the output of the linear layer is given by:

$$\text{Linear}(\mathbf{X}) = \mathbf{X}\mathbf{W}^T + \mathbf{b} \tag{1}$$

where $\mathbf{W} \in \mathbb{R}^{m \times n}$ is the weight matrix and, $\mathbf{b} \in \mathbb{R}^m$ is bias. In our simple MLP model, we only have those two types of parameters. As for initialization, the $\mathbf{W}$ and $\mathbf{b}$ are initialized by He initialization [1] and zero vector $\mathbf{0}$, respectively:

$$\mathbf{W}^{(0)} \sim N(0, \frac{2}{\sqrt{m}}) \tag{2}$$

$$\mathbf{b}^{(0)} = \mathbf{0}^m \tag{3}$$

B. Activation function

The linear layer achieves good performance when modeling the linear relationship between input and output data, but it is not powerful enough to deal with no-linear model, which is more common in applications. To solve such a problem, activation functions are introduced. In this section, we will introduce two activation functions: $ReLU$ and $Tanh$.

Rectified linear unit ($ReLU$)

The ReLU function gives the positive part of the input, and replace the negative elements by zeros:

$$ReLU(x) = \begin{cases} 0, x < 0 \\ x, x \geq 0 \end{cases} \tag{4}$$

Hyperbolic tangent ($Tanh$)

The Tanh function has a similar shape with logistic sigmoid function, but its output takes value in the interval $(-1, 1)$, so its output is zero-centered:

$$Tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{5}$$

C. Architecture

In this project, our simple MLP model only contains four linear layers (three hidden layers) with the $ReLU$ activation function in between. The final linear layer transforms the number of dimensions to the final output dimension, and it is followed by a $Tanh$ function as a prediction layer. The layout of our MLP model is shown in Figure 1.

III. Implementation

Before implementing and training the model, we need to define and implement loss function and optimizer. In this project, we use Mean Square Error(MSE) as loss function, and mini-batch stochastic gradient descent (SGD). To calculate gradient w.r.t each parameter, we introduce backpropagation algorithm.

A. MSE loss

MSE loss is one of the most commonly used and basic loss functions in machine learning. In this project, supposing the output of the model given input $\mathbf{x}$ is $\mathbf{y} = M(\mathbf{x})$, where $\mathbf{y} \in (-1, 1)^2$, and its' corresponding

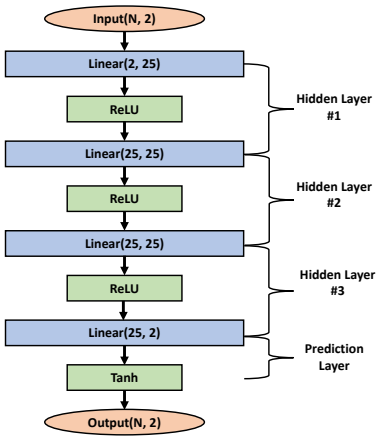

Figure 1. Model architecture layout

label is $\hat{\mathbf{y}}$. The loss of mini-batch $\mathbf{X}$ with size N is given below:

$$Loss(\mathbf{X}) = \frac{1}{N}\sum_{i=1}^{N}||\mathbf{y}_i - \hat{\mathbf{y}}_i||^2 = \frac{1}{N}\sum_{i=1}^{N}||M(\mathbf{x}_i) - \hat{\mathbf{y}}_i||^2 \tag{6}$$

B. mini-batch SGD

Like MSE loss, mini-batch SGD is also a very popular and basic algorithm. It is an optimizer to seek the best parameters for the model by minimizing the loss function. In our code, we don't write a separate SGD module, instead, we merge the SGD algorithm in the training part of our code. The principle of SGD is simple: calculate the gradient of loss w.r.t each parameter. For each mini-batch, update the parameter towards the opposite direction of the gradient. At t-th update:

$$\Theta^{(t)} = \Theta^{(t-1)} - \frac{\eta}{N}\sum_{i=1}^{N}\nabla Loss_i(\Theta^{(t-1)}) \tag{7}$$

where Θ represents all parameters in the model and η represents step size (learning rate). To find the gradient, we need to introduce Backpropagation.

C. Backpropagation

Backpropagation is a widely used algorithm in training feed-forward neural networks for supervised learning. It is a very efficient algorithm to compute the gradient. The foundation of backpropagation is Chain Rule:

Case 1: $y = g(x),\ z = h(y)$

$$\frac{\mathrm{d}z}{\mathrm{d}x} = \frac{\mathrm{d}z}{\mathrm{d}y}\frac{\mathrm{d}y}{\mathrm{d}x} \tag{8}$$

Case 2: $x = g(s),\ y = h(s),\ z = k(x, y)$

$$\frac{\mathrm{d}z}{\mathrm{d}s} = \frac{\partial z}{\partial x}\frac{\mathrm{d}x}{\mathrm{d}s} + \frac{\partial z}{\partial y}\frac{\mathrm{d}y}{\mathrm{d}s} \tag{9}$$

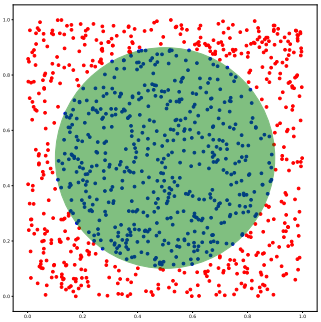

Figure 2. Input data points. Points with target 1 are shown in blue, else red

The forward pass has given in Section III. In this section we discuss the backward pass. The backward pass of an activation function is its derivative, and no parameter is introduced: For $ReLU$:

$$\frac{\mathrm{d}(ReLU(x))}{\mathrm{d}(x)} = \begin{cases} 0, x < 0 \\ 1, x > 0 \end{cases} \tag{10}$$

For $Tanh$:

$$\frac{\mathrm{d}(Tanh(x))}{\mathrm{d}(x)} = 1 - Tanh^2(x) \tag{11}$$

For linear layer, two parameters are introduced, so we have:

$$\frac{\partial \mathrm{Liner}(\mathbf{X})}{\partial \mathbf{W}^T} = \mathbf{X}^T \tag{12}$$

$$\frac{\partial \mathrm{Liner}(\mathbf{X})}{\partial \mathbf{b}} = \mathbf{1}^m \tag{13}$$

$$\frac{\partial \mathrm{Liner}(\mathbf{X})}{\partial \mathbf{X}} = \mathbf{W} \tag{14}$$

For MSE loss:

$$\frac{\mathrm{dLoss}(\mathbf{X})}{\mathrm{d}(M(\mathbf{X}))} = \frac{2}{N}[M(\mathbf{X}) - \hat{\mathbf{Y}}] \tag{15}$$

IV. Experiment

In this section, we employ our model on a simple data set and show its performance. Then, we compared our framework with PyTorch.

A. Data set

The input data are 2-D points uniformly distributed in square $[0, 1]^2$. If the point is inside the circle centered at $(0.5, 0.5)$ with radius $\frac{1}{\sqrt{2\pi}}$, it is labeled 1, and 0 outside. Figure 2 illustrates the distribution of input data and target.

Since the output of our model has 2 dimensions, we should convert the original target into a model-accepted label. To fit the final $Tanh$ layer, we convert it into a 2-D label $[1, -1]$ if the original target is 0, and $[-1, 1]$ if the original target is 1. When predicting, we convert

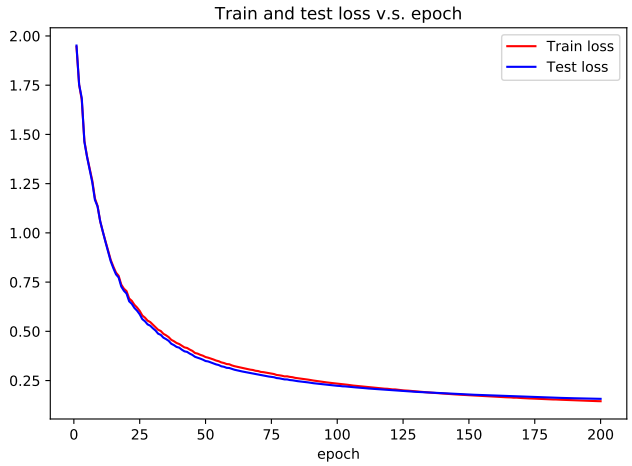

Figure 3. MSE loss on train and test set v.s. train epoch

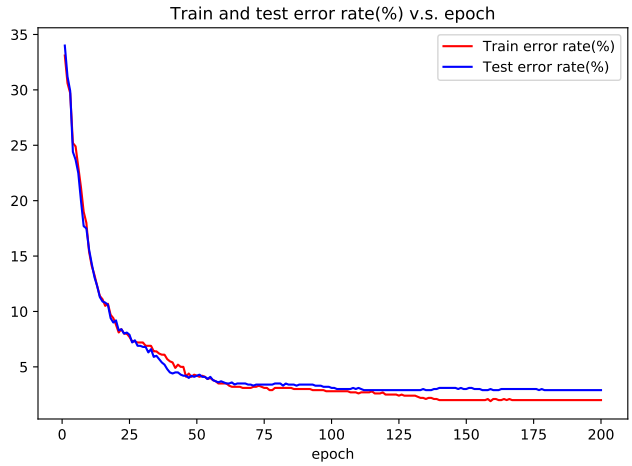

Figure 4. Error rate on train and test set v.s. train epoch

them back by finding the index of the maximum item in model output as a prediction.

B. Experiment setup

We randomly generate 1000 pieces of data for the training set and the test set respectively, and normalize all the data by using the mean and standard deviation of the train data. For training, we use mini-batch SGD with the learning rate $\eta = 0.001$ and the batch size $N = 10$. Finally, we train our model on the train set with 200 epochs.

C. Baseline & Evaluation metrics

PyTorch is one of the most widely used open-source machine learning frameworks that accelerates the path from research prototyping to production deployment. We conduct comparisons with it.

To evaluate the performance of two frameworks, we use two common metrics: error rate and training time.

D. Results

We train the MLP model using our framework and record the loss and error rate on the train or test set. The results are shown in Figure 3 and Figure 4 respectively. Both loss and error rates decrease as the epoch increasing.

To reduce the randomness of our results, we estimate the performance of our framework and PyTorch through 20 rounds each. For PyTorch, we use the same model architecture and experiment setup. The results are given in Table I.

Table I
Comparison with PyTorch

	Error rate		Training time	
	mean	std	mean	std
our Framework	1.89%	2.52E-03	16.82s	4.73E-01
PyTorch	2.36%	3.88E-03	21.71s	2.93E+00

V. Analysis

From the results in Table I, our framework overcomes the PyTorch framework on both error rate and training time in this specific task and model. For training time, PyTorch is a more complex framework designed for more complex architecture and functions. However, our simple framework is specifically designed for the simple MLP model. Therefore, PyTorch may do more extra calculations leading to larger time consumption. As for error rate, it might be caused by different parameter initialization methods between our framework and PyTorch.

VI. Conclusion

In this project, we implement a mini deep learning framework and use it to build a multi-layer perceptron network to deal with a simple two-class classification problem. Comparing the performance of our framework with that of Pytorch, it turns out to be doing well. In the process of implementation, we have a deeper understanding of the basic deep learning algorithms and their implementation methods, including forward propagation, backpropagation, mini-batch SGD and so on.

References

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV), ser. ICCV '15. USA: IEEE Computer Society, 2015, p. 1026–1034. [Online]. Available: https://doi.org/10.1109/ICCV.2015.123