EE 559:Deep Learning-Miniproject 2

Yu-Ting Huang[a], Shengzhao Xia[a], Tianyang Dong[b]
[a]*Institute of Electrical Engineering, EPF Lausanne, Switzerland*
[b]*Institute of Mathematics, EPF Lausanne, Switzerland*

I. Introduction

The objective of this project is to design a mini deep learning framework using only PyTorchs tensor operations and the standard math library. In this project, we implemented a simple framework which could run the forward, backward passes and solve the aimed 2-class points classification problem with a high accuracy. It has 8 basic modules: Linear, Sequential, Relu, Tanh, Sigmoid, Batch Normalization, MSE, SGD.

II. Modules

In this section, we introduce the structures of the modules in our framework and mathematical principles of its forward, backward propagation process.

We define N as data size, n and m as the dimensions of input and output data of one layer. $X_{N\times n}$ and $Y_{N\times m}$ are defined as the input and output matrix of the layer. Each row of them represents one example, while each column represents one feature, so we represent their components as X_{ij} and Y_{ij} respectively. L is defined as the final loss of the network.

A. Linear

Linear layer is a fully connected layer and it applies a linear transformation to the incoming data from previous layer. Linear layer holds two parameters, weight $W_{n\times m}$ and bias $B_{1\times m}$, which should be updated in optimization process. In Pytorch, tensor $B_{1\times m}$ will be broadcast to N same rows in matrix computation, we use $B_{N\times m}$ to represent the broadcast matrix.

1. Forward Propagation:

Each neuron in a fully connected layer have connections to all neurons in the previous layer. So its activation functions can be computed with a matrix multiplication followed by a bias offset:

$$Y_{N\times m} = X_{N\times n}W_{n\times m} + B_{N\times m}$$

2. Initialization:

Proper initialization is an effective way for preventing gradient from vanishing exponentially with the depth, because it can help maintain proper statistics of the activations and derivatives. The variance of input should be initialized according to the type of the activation function.

- He initialization: This method is recommended for layers with a ReLU activation function. It multiplies the activations by a corrective gain of 2.

$$\mathbb{V}(W_{n\times m}) = \frac{2}{n}$$

- Xavier initialization: This method is recommended for layers with a tanh activation. It reaches a compromise between the controls of the variance of activations and variance of the gradient.

$$\mathbb{V}(W_{n\times m}) = \frac{2}{n} + \frac{2}{m}$$

3. Backward Propagation:

According to chain rule, gradients with respect to parameters and input can be calculated as following:

$$\frac{\partial L}{\partial X_{N\times n}} = \frac{\partial L}{\partial Y_{N\times m}}\frac{\partial Y_{N\times m}}{\partial X_{N\times n}} = \frac{\partial L}{\partial Y_{N\times m}}W^T_{n\times m}$$

$$\frac{\partial L}{\partial W_{n\times m}} = \frac{\partial L}{\partial Y_{N\times m}}\frac{\partial Y_{N\times m}}{\partial W_{n\times m}} = X^T_{N\times n}\frac{\partial L}{\partial Y_{N\times m}}$$

$$\frac{\partial L}{\partial B_{N\times m}} = \frac{\partial L}{\partial Y_{N\times m}}\frac{\partial Y_{N\times m}}{\partial B_{N\times m}} = \frac{\partial L}{\partial Y_{N\times m}}\mathbb{I}_{m\times m}$$

B. Activation Function

1)ReLu

The Rectied Linear Unit Activation Function (ReLU) is the most popular activation function, which fixes the gradient vanishing problem, see Fig 1a.

1. Forward Propagation:

$$Y_{ij} = ReLu(X_{ij}) = \begin{cases} X_{ij}, & X_{ij} > 0 \\ 0, & Others \end{cases}$$

2. Backward Propagation: There is no parameters in ReLU layer, so chain rule is not needed. We can calculated it as following:

$$(\frac{\partial Y}{\partial X})_{ij} = (\frac{\partial ReLu(X)}{\partial X})_{ij} = \begin{cases} 1, & X_{ij} > 0 \\ 0, & Others \end{cases}$$

2)Tanh

Tanh is the hyperbolic tangent function, which will not change the sign of input but compress it from range (-∞, +∞) to range (-1, 1), see Fig 1b.

1. Forward Propagation:

$$Y_{ij} = Tanh(X_{ij}) = \frac{e^{X_{ij}} - e^{-X_{ij}}}{e^{X_{ij}} + e^{-X_{ij}}}$$

2. Backward Propagation: The gradient w.r.t input is:

$$(\frac{\partial Y}{\partial X})_{ij} = (\frac{\partial Tanh(X)}{\partial X})_{ij} = \frac{4}{(e^{X_{ij}} + e^{-X_{ij}})^2}$$

3)Sigmoid

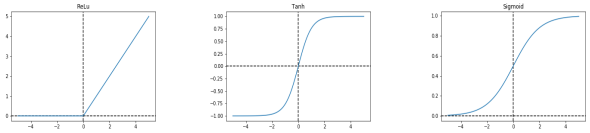
Similar to Tanh, a sigmoid function is a bounded, differentiable function that is defined for all real input values and has a non-negative derivative at each point, see Fig 1c.

1. Forward Propagation:

$$Y_{ij} = Sigmoid(X_{ij}) = \frac{1}{1 + e^{-X_{ij}}}$$

2. Backward Propagation:

$$(\frac{\partial Y}{\partial X})_{ij} = (\frac{\partial Sigmoid(X)}{\partial X})_{ij} = \frac{e^{-X_{ij}}}{(1 + e^{-X_{ij}})^2}$$



(a) ReLu (b) Tanh (c) Sigmoid

Fig. 1: Activation functions

C. Loss function

1)Mean Squared Error

MSE measures the average of the squares of the errors between the output of network and the ground truth.

1. Forward Propagation: Here, we define Y_i^{pred} and Y_i^{label} as the prediction of network and ground truth label.

$$L = \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{m}(Y_{ij}^{pred} - Y_{ij}^{label})^2$$

2. Backward Propagation: As a sum of quadratic function, its derivative can be easily calculated as below.

$$\frac{\partial L}{\partial Y_{ij}^{pred}} = \frac{2}{N}\sum_{i=1}^{N}\sum_{j=1}^{m} Y_{ij}^{pred} - Y_{ij}^{label}$$

2)Cross Entropy Loss

Cross entropy loss indicates the difference between predicated distribution of prediction and the ground truth. Cross entropy is always followed by softmax activation as the last layer, we combine them together in loss.

1. Forward Propagation:

$$Y_{ij}^{pred} = softmax(X_{ij}) = \frac{e^{X_{ij}}}{\sum_{l=1}^{m} e^{X_{il}}}$$

$$L = CrossEntropy(Y) = \sum_{i=1}^{N}\sum_{j=1}^{m} Y_{ij}^{label} log(Y_{ij}^{pred})$$

2. Backward Propagation: Since the output layer has only two nodes, we only calculate the simplest case, when $m = 2$.

$$(\frac{\partial L}{\partial X})_{ij} = \sum_{l=1}^{m} \frac{Y_{il}^{label}}{Y_{il}^{pred}} \frac{\partial Y_{il}^{pred}}{\partial X_{ij}}$$

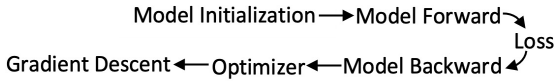


Fig. 2: Pipeline

III. Implementation

The pipeline of our framework is shown in Fig. 2. The entire model above consists of 2 input units, 1 or 2 output units and three hidden linear layers with 25 points. We utilize ReLU, Tanh, Sigmoid functions as the activation function, MSE and Cross-Entropy loss as loss function and stochastic gradient descent as the optimizer.

Following this pipeline, we train our model according to the following setting.

- **Epoch:** Size 200.
- **Mini-batches:** Size 100.
- **Optimizer:** SGD with learning rate 0.01.

IV. Framework Testing

A. Data generation

The data set is composed of 1000 points and sampled uniformly, randomly, and independently from $[0, 1]^2$. The points are labeled as 1 if they lie inside the circle with radius $\frac{1}{\sqrt{2\pi}}$ centered in (0.5, 0.5), and labeled as 0 otherwise. The testing dataset contains 500 points whihc are sampled in the same way.

B. Network Building and Comparison

1. Network Building

The model consists of one input layer with 2 units, three hidden layers with 25 units, and one output layer with 2 units. To investigate the performance of different activation functions and loss functions, we construct several combinations of network modules. They are respectively *Tanh + MSE, ReLU + MSE, Tanh + Cross Entropy, ReLU + Cross Entropy*. The activation function after the output layer is sigmoid in all modules, so we don't illustrate it above. Normally, people will not add an activation function after output layer if using mean squared error; nonetheless, we add sigmoid activation for MSE because we are doing classification task here.

In Module ReLU + MSE, we found that ReLU activation will not limit the positive orthant, which leads to gradient explosion. Therefore, we introduce two methods to solve this problem, which are parameter initialization and simplified batch normalization.

1) Parameter Initialization: As mentioned above, we use He initialization method to initialize our parameters.

2) Simplified Batch Normalization: To increase the stability of a neural network, batch normalization normalizes the output of the previous activation layer by subtracting the batch mean and dividing by the batch standard deviation. The concept of batch normalization comes from the fact that if an algorithm learned an mapping from X to Y , and if the distribution of X changes, then we might need to retrain the learning algorithm by trying to keep the distribution of X same as the distribution

of Y. Batch normalization prevents the hidden unit values from shifting around. It allows each layer of a network to learn regardless of the influence of other layers.

There are two learnable parameters in the original Batch Normalization[1]. However, since we only train on a toy dataset, we don't implement the scale and shift part in the algorithm namely, there is no parameter to train in our simplified batch normalization.

2. Network Comparison

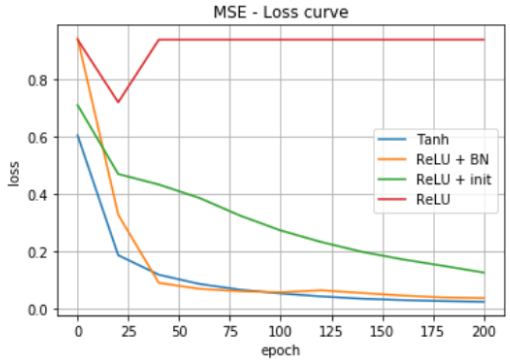


Fig. 3: MSE Loss Log

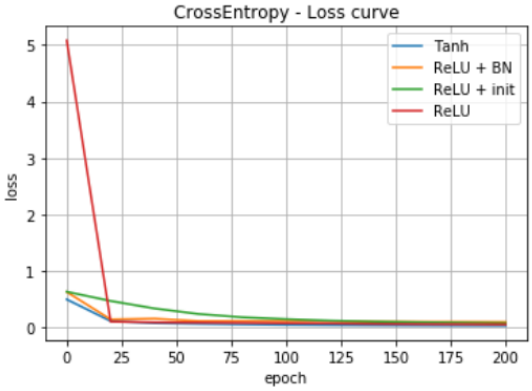


Fig. 4: Cross Entropy Loss Log

To compare the performance between modules, we plot the log loss for MSE and Cross Entropy in Fig. 3 and 4. However, the scale of loss is different and it is hard to compare the performance based on loss. Therefore, we run 10-fold cross validation to obtain a more confidential result and evaluate the model based on number of errors. Using simplified batch normalization and parameter initialization to prevent gradient explosion, now we will compare 8 modules together. Note that the moduel ReLU + MSE has training error with mean equals to 340.6, and standard deviation equals to 119.4. We do not plot this failing model in the figure to avoid outlier. In the following part, we will explain some observations from Fig. 5 and 6.

1). People usually use cross entropy loss for the classification problem. We can see the advantages of cross entropy loss over mean squared error by comparing between models using ReLU + MSE and ReLU + CrossEntropy. As stated before, combination of ReLU and MSE will lead to gradient explosion. On the contrary, cross entropy almost performs the best among our models. This result comes from the fact that the number of errors of correctly-classified points is not zero under MSE, and zero under cross entropy loss.

2). Simplified batch normalization and parameter initialization makes the combination of ReLU and MSE trainable; however, compared to the performance of the other models, the mean and standard deviation of number of errors are both larger, which means their performances are unstable, especially the parameter initialization.

3). Simplified batch normalization and parameter initialization increase the number of errors when using cross entropy. In our opinion, it is resulted from the fact that we don't implement the scale and shift in batch normalization. Furthermore, although good initialization creates a good start point, it is too weak to guarantee success.

4). The combination of Tanh and ReLU performs the best in our example. However, this might be different if we have a different task.

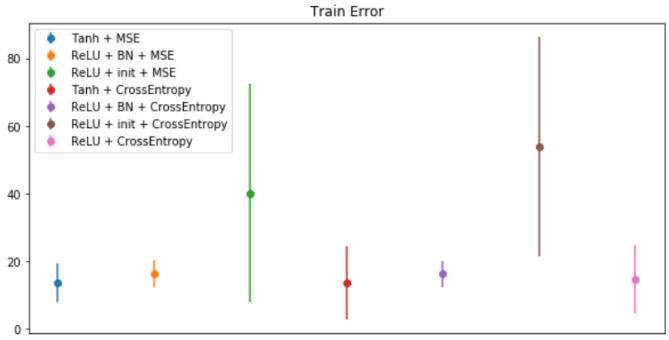


Fig. 5: Train Error

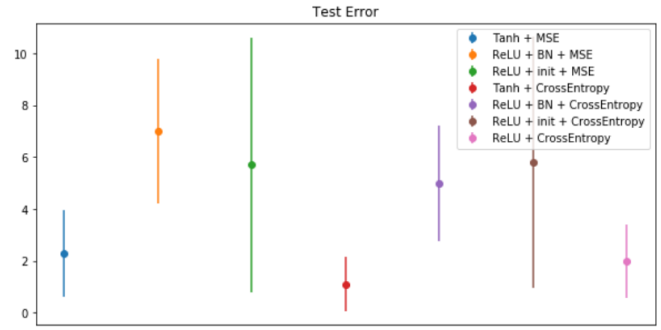


Fig. 6: Test Error

REFERENCES

[1] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.