Designing a Mini Deep Learning Framework through Various Activations and Optimizers

Bohan WANG
bohan.wang@epfl.ch

Ke WANG
k.wang@epfl.ch

Anqi Hou
anqi.hou@epfl.ch

***Abstract*—The objective of this project is to design a mini deep learning framework using only python operations, including only pytorch tensor operations and standard math library, without advanced tools from the deep learning frameworks.**

I. Introduction

Respecting the spirit of the project to design the deep learning framework without using tools from deep learning tools, this report describes how this project designs the deep learning network, with the following properties:

- incorporating different optimization algorithms, including SGD, Adam, RMSProp, to train the network and comparing their performance on the same classification task.
- using cross-validation to optimize the respective super-parameters for all three optimization algorithms.
- providing different choices activation function to use
- visualizing the result for training and prediction to evaluate the model performance

As an outline for this report, in Section II we will briefly explain the modules and functions we designed for this project; in Section III we will explain our approach for training the network and cross validation; in Section IV we will discuss the results of different optimizers and model structures we used in the project.

II. Module Description

In this section the different modules used in our project are explained.

A. Model module

The Model module (in the codes it is called Module) is the basic module to design our neural network. More specific modules including Linear module, Sequential module, and the modules for loss function and activation functions, all inherited from the this module. It has 6 basic functions.

inti_params: initialize the parameters for the network
forward: forward pass for the network
backward: backward pass for the network
param: contains the parameters and their gradients for the network
update: function to update the parameters by gradient descent
zero_grad: initialize the gradients to zeros

1) Linear module: The linear module defines the module for one layer in the network. This module contains the following functions:
inti_params: initialize the parameters for this layer according a given initialization method (zero initialization or Xavier initialization).
forward: calculate the output for this layer to be forwarded to next layer.
backward: calculate the gradient of this layer to be backwared to previous layer
param: returns the parameters for this layer, as well as their gradients
update: update the parameters for this layer by the step of gradient descent
update_adam: update the parameters for this layer by the step of Adam optimizer
update_rmsprop: update the parameters for this layer by the step of RMSProp
zero_grad: initialize the gradients of this layer to zero

2) Sequential module: The sequential module is a module used to connect the linear modules, and builds a neural network. It includes the following functions:
inti_params: initialize the parameters for each layer of the network
forward: forward the outputs from first layer to the final layer
backward: calculate and backward the gradients for each layer
param: returns the parameters (and their gradients) for each layer
update: update the parameters of each layer by the step of gradient descent
update_adam: update the parameters of each layer by the step of Adam optimizer
update_rmsprop: update the parameters of each layer by the step of RMSProp
zero_grad: initialize the gradients of each layer to zero

3) 'Xavier' type initialization: The weights and bias are initialized with Gaussian distribution centered at 0. The variance of Gaussian distribution for in a layer l is calculated by below formula.

$$\mathrm{V}\left(w^{(l)}\right) = \mathrm{V}\left(b^{(l)}\right) = \frac{2 * gain}{N_{l-1} + N_l} \tag{1}$$

B. Loss modules

In our framework we use Mean Squared Error (MSE) as the loss function.

1) Loss_MSE module: calculates the MSE of the given label and predictions, containing the following functions:

forward: calculates the MSE based on given label and prediction

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

backward: calculate the gradient of MSE loss with respect to prediction

$$\frac{2}{N}(y_{pred} - y_{label}) \tag{3}$$

C. Activation function modules

Three activation functions are incorporated in the project, including ReLU, Tanh and Sigmoid function.

1) ReLU module: calculates the ReLU activation for the given input, with the following usable functions:

forward: calculates the ReLU activation

$$\max(0, x) \tag{4}$$

backward: calculates the gradient of the ReLU activation with respective to its input

$$\begin{cases} 1, x > 0 \\ 0, o.w. \end{cases} \tag{5}$$

update_adam and **update_rmsprop:** these two functions are put here for integrity but are passed since ReLU module has no parameters to update

2) Tanh module: calculates the Tanh activation for the given input, with the following usable functions:

forward: calculates the Tanh activation

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{6}$$

backward: calculates the gradient of the Tanh activation with respective to its input

$$1 - tanh(x)^2 \tag{7}$$

update_adam and **update_rmsprop:** passed for the same reason as ReLU

3) Sigmoid module: calculates the Sigmoid activation for the given input, with the following usable functions:

forward: calculates the Sigmoid activation

$$sigmoid(x) = \frac{1}{1 + e^{-x}} \tag{8}$$

backward: calculates the gradient of the Sigmoid activation with respective to its input

$$sigmoid(x) \cdot (1 - sigmoid(x)) \tag{9}$$

update_adam and **update_rmsprop:** passed for the same reason as ReLU

III. Training and Cross Validation

A. Training process

In our project, we defined a function for training the neural network. The input of this function determines the loss, super-parameters and optimizer in the training process.

For each iteration, this function calculates the neural networks output through forward, the gradients of the network through backward, and the loss. Then, according to the given optimization algorithm, update the parameters of each layer. This process is iterated until maximum iteration number is reached. For every 10 iterations, the training loss is printed.

In the training process, the three different optimization algorithms take different training steps.

SGD updates the parameters by subtracting their gradients multiplied by the learning rate.

$$p_t = p_{t-1} - lr * \bigtriangledown p_{t-1} \tag{10}$$

Adam updates the parameters by the following rule:

1. Set m_0, v_0 = 0
2. For t = 1,..., iterate

$$\begin{cases} m_t = \beta_1 * m_{t-1} + (1 - \beta_1) * \bigtriangledown p_{t-1} \\ v_t = \beta_2 * v_{t-1} + (1 - \beta_2) * (\bigtriangledown p_{t-1})^2 \\ \widehat{m_t} = m_t / (1 - \beta_1^t) \\ \widehat{v_t} = v_t / (1 - \beta_2^t) \\ p_t = p_{t-1} - lr * \widehat{m_t} / (\sqrt{\widehat{v_t}} + \epsilon) \end{cases} \tag{11}$$

RMSProp updates the parameters by the following rule:

1. Set ρ_0 = 0
2. For t = 1,..., iterate

$$\begin{cases} r_t = \rho * r_{t-1} + (1 - \rho) * \bigtriangledown^2 p_t \\ p_{t+1} = p_t - lr * \bigtriangledown p_t / (\sqrt{r_t + \epsilon}) \end{cases} \tag{12}$$

B. Cross Validation

In our project, cross validation is performed to find the optimal super-parameters for the three optimization algorithms and different acitivation functions we used.

For SGD, the super-parameters learning rate and mini-batch size are optimized.

For Adam, the super-parameters learning rate, mini-batch size, β_1, β_2 and ϵ are optimized.

For RMSProp, the super-parameters learning rate, mini-batch size, ρ and ϵ are optimized.

Here, we use a 5-fold cross-validation for each combination of the activation functions and optimization algorithms. Since the data are generalized randomly and independently, we did not explicitly shuffle the data before cross validation.

Our cross validation function iterates over each combination of the super-parameters, using which to train the neural network individually.

The optimal super-parameters found by the cross-validation is shown in the TABLE I.

	SGD		ADAM					RMSPROP			
	lr	minibatch	lr	β_1	β_2	ϵ	minibatch	lr	minibatch	ρ	ϵ
ReLU	0.001	1	0.0001	0.9	0.999	1e-9	20	0.0001	20	0.9	1e-9
Tanh	0.01	1	0.0001	0.8	0.999	1e-9	10	0.001	20	0.9	1e-9
Sigmoid	0.01	20	0.0001	0.8	0.99	1e-9	20	0.0001	20	0.9	1e-9

TABLE I: The optimal parameters for each model found by cross validation

IV. Results and Discussions

After cross validation, we use the respective optimal super-parameters to train the neural network given in Fig.1. The results of different optimization algorithms using the optimal super-parameters are presented in this section.

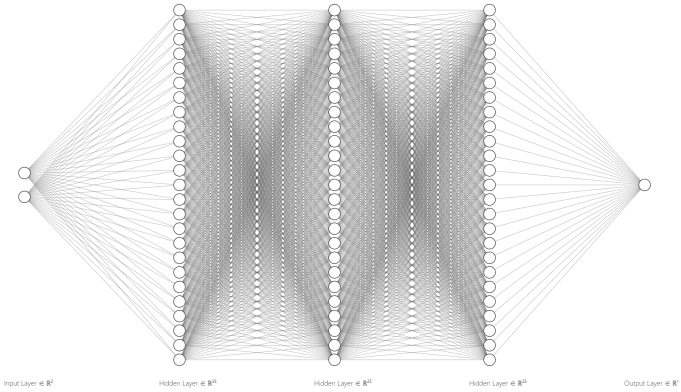

Fig. 1: Neural network structure

First, we generate 1000 training samples and 1000 test samples, sampled uniformly in $[0, 1]^2$. The points outside the disk centered at (0.5, 0.5) of radius $1/\sqrt{(2\pi)}$ are labeled with 0, and points inside the disk are labeled with 1.

Next, with the modules we defined and introduced in Section II, we construct a neural network with the following structure through the sequential module.

Then, with the optimized super-parameters, we run SGD, Adam and RMSProp individually using three activation functions (ReLU, Tanh, Sigmoid) to train the neural network.

Fig.2 presents the averaged classification accuracy on test set over 10 runs for each combination of the acivation function and optimizer. Here the error bar represents the standard error of the ten runs. From this figure we can see, the test accuracy for all the 9 neural networks surpassed at least 96.9%. SGD with Tanh activation outperforms other neural networks, with its mean accuracy being 98.49%. The optimal choices for the model structure may probably be SGD with ReLU or Tanh, since they have a high accuracy with relatively low variance.

Fig.3 compares the training loss for each neural network structure. The shadows in the figure represents the standard error of each point. Generally, Adam and RMSProp converge faster than SGD, and they also have a lower training loss compared with SGD. Adam with ReLU activation and RMSProp with ReLU activation have the lowest training loss. It is interesting to observe that, although SGD with ReLU and SGD with Tanh have the highest training losses, they actually have the best and 3rd best test accuracy, which reveals that we can not judge the performance of a neural network simply by its training loss.

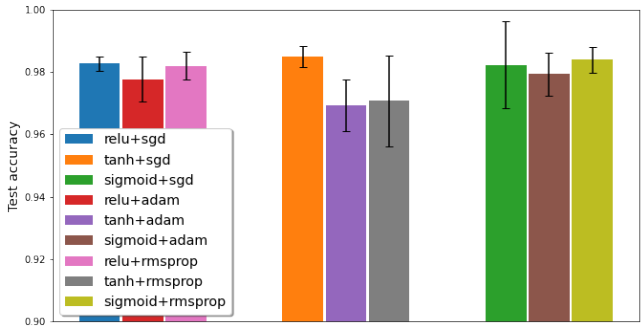

Fig. 2: Test accuracy of various neural networks

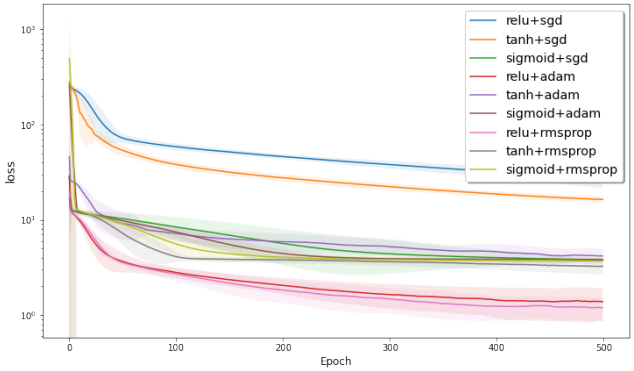

Fig. 3: Training loss of various neural networks

V. Conclusion

In this project, we desined a neural network using without utilizing the tools from deep learning libraries. For the classification problem, the performance of three optimization algorithms, including SGD, Adam, RMSProp are compared using different activation functions. To improve the model performance, a cross validation to optimize the super-parameters for each algorithm is performed. In the final result, SGD with ReLU and SGD with Tanh are very good candidates for the optimal model structure, for their high test accuracy and relatively low standard error.