National Tsing Hua University
Fall 2023 11210IPT 553000
Deep Learning in Biomedical Optical Imaging
Homework 2

WEN CHI CHEN[1]

[1]*Master Degree Student, High-energy Optics& Electronics Laboratory (HOPE Lab), Laser & Nonlinear Optics Laboratory, Institute of Photonics Technologies, National Tsing Hua University.*

Student ID: 110066515

1. Introduction

There are two model introduced in the course: Binary Cross Entropy Loss Function (BCE loss) & Cross Entropy Loss Function (CE loss). The main difference between them is the connection activation function from the last hidden layer to the output layer. The Figure 1 [1] is the diagram of CE loss. The activation function is Softmax(), which output the sum of probabilities of each possible case to be 1, For example, suppose we want to handwrite the numbers 0~9. After inputting an image of the number, we perform distributed Softmax regression. Finally, we need to output the probability that the image belongs to 0~9 individually, and the sum of the individual probabilities is 1. The Softmax regression provide the ability of multi-classification to CE loss. However, the Sigmoid regression in BCE only gives the BCE distinguish whether it is number 9 or not, for example.

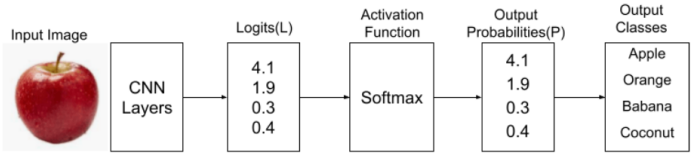

Fig 1. CE loss diagram

2. Model Introduction

The experiment setup is listed in the following Table 1 to 3. First of all, I change the training set split point to be 900 in the code of Lab 3, so the number of training set will be 1800, and the number of test set will be 200. The reason I changed this parameter is to give more training example for the loss function training since there is no training backward propagation relation about the validation set. For my perspective, reduce the number of validation set will not cause lot of detect on the entire training model.

Second, about the training batch size, I modify the training batch size to be 128. From the class video (Figure2. [2]), the batch size will affect a lot the training accuracy. If the batch size is too small, even though the training time will be small for each iteration, the gradient path will be extremely noisy; if batch size is too large, it might consume too much GPU memory and cause the training time be extended. Therefore, I choose the batch size to be 128 and find that my training curve become smoother and the accuracy result still saturate under the same epoch under proper learning rate. The related equations are shown in course video.

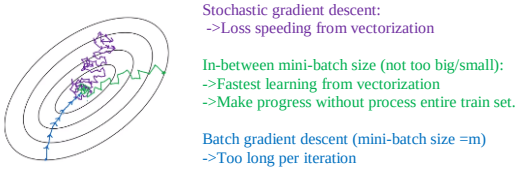

Fig 2. Batch size comparison.

Third, about the hidden layer and activation function design, I just simply set 3 layer both connected with ReLU function, and there is no dropout term, since I don't want to increase the complexity of my model; therefore, for the CE loss function output layer, there are only 2 classification.

Last, the Learning model, I use the model that TA provided, which is Adam for optimizer, StepLR for the learning rate decay. I find that *Adam Optimizer* combines the advantages of *stochastic gradient decent* and *Momentum*, it finds the gradient slope efficiently but with gradient direction guidance, so *Adam* does "deviation correction" of parameters, so that each learning rate will have a certain range, which will make the update of parameters more stable. And for the learning rate of StepLR, it will be discussed in tuning hyperparameters part.

Table 1. Training Model parameters

Dataset	
# training set	1800
# validation set	200
# test set	400
Training batch size	128
Test batch size	32
Epoch	30

Learning Model	
Loss Function	BCE/CE
Optimizer	Adam
lr	3e-6
LR model	StepLR
step_size	4
gamma	0.4

Hidden Layers & Activation functions	
# Layers	3
Layer 1	
# nodes	64
Activation function	ReLU
Dropout	0
Layer 2	
# nodes	64
Activation function	ReLU
Dropout	0
Layer 3	
# nodes	64
Activation function	ReLU
Dropout	0
Output classification	1 for BCE 2 for CE

3. Comparison of Performance between Binary Cross Entropy Loss Function (BCE loss) & Cross Entropy Loss Function (CE loss)

Following Fig 3 & 4 are the accuracy and loss output of my BCE & CE loss model. The performance of my BCE loss & CE loss models gets the training accuracy: ~94% & 92%, training loss: ~0.37 & 0.37, validation accuracy: ~90% & 87%, validation loss: ~0.37 & 0.35, the test accuracy for my best try is 83% and 82.25%, shown in following Table 2.

Table 2. Training Performance Comparison

Performance of BCE & CE loss		
	BCE	CE
training accuracy	94%	92%
training loss	0.37	0.37
validation accuracy	90%	87%
validation loss	0.37	0.35
test accuracy	83%	82.25%

The result implies that under my setting, in the terms of loss function, the BCE loss trains the similar performance compared with that of CE loss, since they have similar training

loss value: 0.37. About the validation loss, BCE loss get the value equal to that of its train loss, but CE's validation loss gets slightly reduction.

And In the terms of accuracy, the BCE loss gets all the advantage over CE loss. From these points of views, I would like to say **BCE loss plays a slightly better training role** in my model. Meanwhile, if I look into the training curves as well and compare them, there are 3 reasons support my perspective:

1. BCE gets simply higher accuracy on three datasets.
2. The accuracy & loss curves are more stable in BCE model
3. Even though the accuracy curve of BCE looks like overfitting, the smooth loss curve tells me it still works in the control.
4. Accuracy & loss curves of CE have the better outline; however, they contain too lot of fluctuation at higher iteration.

Note that the reversion in loss curve of CE only replies that the validation sets is better trainable compared to training set under the my CE loss setting.

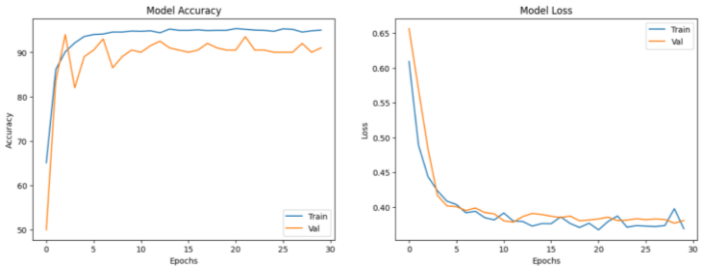

Fig 3. Performance of BCE loss

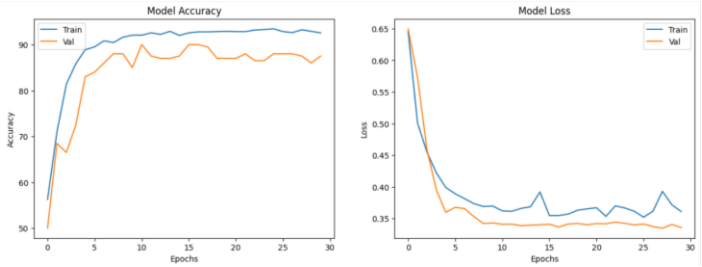

Fig 4. Performance of CE loss

4. Performance between Different Hyperparameters

optimizer = optim.Adam(model.parameters(), lr=3e-6)
lr_scheduler = StepLR(optimizer, step_size=4, gamma=0.4)

In my report, I would like to tune these two parameters labeled in red letter: for the *StepLR* learning rate, the tuning parameter is shown in Figure 5[2]. And the tuning values & final test accuracy are shown in Table 3.

Table 2. Hyperparameters Tuning Table

Hyperparameters Tuning			
	lr	Step_size	Test accuracy
Sample 1	3e-6	4	83%
Sample 2	3e-5	4	80.25%
Sample 3	3e-7	4	77%
Sample 4	3e-6	8	74.75%
Sample 5	3e-6	2	75.25%

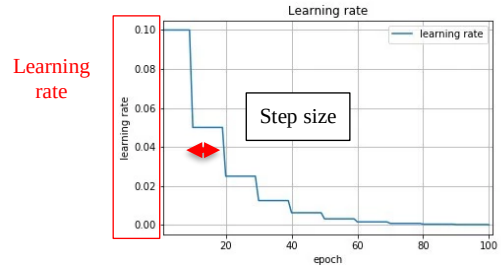

Fig 5. StepLR[2]

Sample 1
lr = 3e-6
Step_size = 4

Sample 2
lr = 3e-5
Step_size = 4

Sample 3
lr = 3e-7
Step_size = 4

Sample 4
lr = 3e-6
Step_size = 8

Sample 5
lr = 3e-6
Step_size = 2

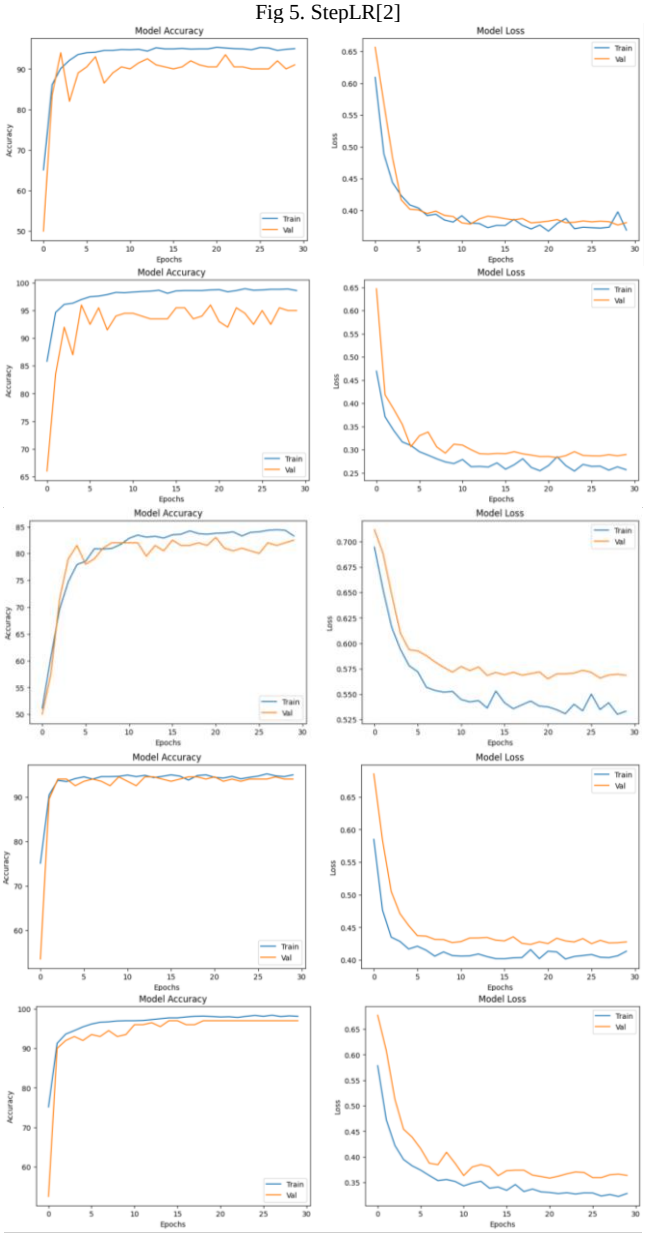

Fig 6. Performance of Hyperparameters tuning

For the first hyperparameter: lr (learning rate), I change it from my best model value: 3e-6 to 3e-5 & 3e-7, the two tuning performance are shown in sample 2 & 3 in Figure 6. The loss curves imply that for the larger learning rate, training model might overfit due to the steeper slope at the beginning and the larger fluctuation at higher iteration. By the way, the overfitted model usually get training accuracy reach >98% in my model, as shown in sample 2 in Figure 6; however, for the lower learning rate, the training model might be underfitted. As shown in sample 3 in Figure 6, the accuracy stops at 85% (cannot be higher since the learning rate decay) and the loss curve get larger value compared to that of overfitted one. The accuracies follow my opinion, for the lr = 3e-5, overfitted one, the test accuracy (80.25%) has a large gap between the training accuracy (98%); for the lr = 3e-7, underfitted one, the test accuracy (77%) is the lowest accuracy even though it's close to training accuracy (85%).

For the second hyperparameter: step_size, I change it from my best model value: 4 to 8 & 2, the lower the value the faster the decay of learning rate. Therefore, for step_size = 8, the decay of learning rate is slower compare to my best model, so the loss curve is steeper at the beginning due to higher learning rate at low iteration; for step_size = 2, the decay of learning rate is faster, so the loss curve is less steep. The two hyperparameters tuning output are shown in sample 4 & 5 in Figure 6.

5. Conclusion

In this report, I first compare the difference between BCE and CE loss and their practice performance difference, then the hyperparameter tuning reveals that the learning rate have a role to play in underfitting/just good/overfitting training model.

Taking into account the multifaceted aspects and complexities of the matter at hand, the BCE loss runs better performance due to there being only 1 classification: (pneumonia or not), but CE is mainly used in multi-classification. If BCE loss cannot have better performance in our case (pneumonia or not), why should it exist?

And about the hyperparameter, the larger the learning rate, the larger the possibility to overfit the mode; however, if the learning rate is too small, it might be a underfitted mode. Therefore, choose a proper start learning rate(lr) and decay parameters (step_size, gamma) is a crucial trade-off of the learning rate curve. For example, if it is a high starting learning rate and a high decay rate, then the first few steps of the gradient descent are vital since it must find the proper gradient path at the beginning or the model will be stuck in a local minimum. Another example is low starting learning rate and a low decay rate, such a model will usually be overfitted since the lower starting learning rate make it easily stuck near its random beginning.

6. References

References

1. https://androidkt.com/what-is-categorical-cross-entropy-loss-function-in-keras/
2. https://www.youtube.com/watch?v=-_4Zi8fCZO4&list=PLkDaE6sCZn6Hn0vK8co82zjQtt3T2Nkqc&index=16&ab_channel=DeepLearningAIK.
3. https://hasty.ai/docs/mp-wiki/scheduler/steplr