ADL Problem 1 Report

Zhenning Li, Qiyun WU, Brieg L'Hostis

1 Multiclass Problem Summary

For multiclassification, the best model has the architecture:

4 layers

$\Rightarrow F.relu(Conv2d(1, 16, 3, stride = 1))$

$\Rightarrow F.relu(Conv2d(16, 8, 3, stride = 1))$

$\Rightarrow F.relu(Linear(12 * 12 * 8, 32))$

$\Rightarrow F.softmax(Linear(32, 10))$;

Criterion is CrossEntropyLoss, optimizer is Adam, epochs are 20, batch size is 10.

This model has achieved an accuracy of 98.45% on validation set and 94.57% on test set.

In the training process to improve the numerical performance, we separately tried 2, 3, 4 hidden layers, and "Softmax", "Sigmoid", "Tanh", "ReLU" activation function in the given model which has 5, 10, 20, 50, 100, 200 neurons in each hidden layer with a learning rate of 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 5, 10. The best result (accuracy: 97.21%) has: Tanh activation function, learning rate=0.1, epochs=20, batch=10 in convolutional model with 4 hidden layers.

We did some research on the impact of the optimizer "SGD", "ADAM", "RMSprop": we tried to change the batch size, learning rate and epochs (10, 20, 50, 100, 200) on the given model and the convolutional model created (with 4 hidden layers), and the best model (accuracy: 97.6%) is the convolutional model with a learning rate of 1, epoch=20, batch size=10 using SGD optimizer.

Besides, we also did some research on impact of loss function: MSELoss and CrossEntropyLoss. We found that CrossEntropyLoss can yield an even better accuracy on convolutional model (98.45%) with learning rate of 0.001, Adam optimizer, 20 epochs, without final activation function and with a batch size of 10.

We chose the best model above we could find to run on test set, and we got an accuracy of 94.57%.

2 Regression Problem Summary

For regression, the best model has the architecture: 3 hidden linear layers (100 neurons per layer) with relu in the middle as activation function and without final activation function, learning rate=0.01, batch size=10, epochs=20. This model has a validation loss of 0.157 with MSELoss.

In the training process to improve the numerical performance, we researched the impact of mode architecture: we tried a model with 1, 2, 3 linear hidden layers, tried "None", "Sigmoid", "Tanh", "ReLU" four activation function separately with 5, 10, 20, 50, 100 neurons in each layer, and we found the model with 3 hidden layers with 100 neurons per layer, 0.01 learning rate, MSELoss criterion, SGD optimizer and 20 epochs with batch size of 10 can yield the best result (validation loss of 0.1567).

We researched the impact of optimizer: separately we tried "SGD", "ADAM", "RMSprop" these three optimizers with batch size 10, 20, 50, 100, 200, 400, learning rate of 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 5, 10 and number of epochs of 10, 20, 50, 100, 200, and we found the best lowest validation loss (0.1596) has: SGD optimizer with 0.01 learning rate, 200 epochs, 10 batch size and MSELoss criterion.

We also tried to change the loss function from MSELoss to self defined Gaussian likelihood loss function. With the best model parameters above, the loss error decreased from 0.9545 to 0.0672 with a loss of 0.099 on validation set.

On test set, we used Gaussian likelihood loss and got a loss on validation set of 0.1028 and 0.0128 on test set.

3 Discussion

For deep learning hyper-parameter tuning, it's not quite possible to use grid search, as in my classification problem, there will be 3*4*6*9*5=3240 combinations and usually for one trial the training time is long.

Among many of our trials, we found that they almost have very similar accuracy or validation loss, only trivial difference. For multi-classification problem, after using training and validation set, we found epochs=20 is relatively better, so the training time is very short, around 30 seconds with Google Colab Pro. Theoretically speaking, if we want to find the optimal parameters, it should take us 30*3240=27 hours to try every combination. Based on the patterns we found, it took us about 30 minutes to find our final hyper parameters.

We use a "dynamic programming"-like method to find our best parameters: In theory and based on experience, from most important parameters to "less" important are: learning rate, batch_size & loss function, optimizer, number of hidden layers and kernel size etc. In this ranking, epochs are not included, because epochs are something to be determined by watching the training and validation loss. Therefore, to get our optimal model, we first tried to fix other parameters (20-40 epochs, others based on several random trials) and find the best learning rate. Then, we fixed this learning rate, and found the best batch size and loss function that can yield to the highest accuracy. Then similarly, fix all the parameters that have been chosen, and one by one test the other parameters and the final model is done. To demonstrate it's the best model, we randomly changed learning rate etc., and saw indeed the accuracy was not as high as the best model.

We found that, in accordance with theory, learning rate changes the accuracy most, all other parameters fixed, a good learning rate can converge very fast and a bad can never converge. Among those which converge, the accuracy differed so largely. Then when we change other parameters, indeed the result did not change must, usually just a little improvement was achieved.

3.1 Specific influence of different parameters

Architectural choice: In theory, the more hidden layers, the more complex the model, the better (before overfitting problem happens). In multi-classification problem, the input image is very simple, so 3 hidden layers and 4 hidden layers model performed the best. 2 hidden layer model and 5 or more hidden layer models will achieve a quite low accuracy. Similarly, in regression model, 3 hidden layers performed the best, and the graph of performance with regards to number of layers is like a parabola with a negative coefficient: there must be a comparatively better number of layers for any problem.

Optimizer: For SGD, it is the simplest method with a certain learning rate, so in theory it would very probably yield a saddle point or local minimum, and it will cause the loss function to fluctuate very heavily. For RMSprop, it uses second order momentum to adjust the learning rate. For Adam, it combines SGD and RMSprop and has their advantages. In theory, Adam should perform better than RMSprop and SGD should perform the worst, and it is the case in first problem, but SGD performed better in the second one, but still, no much difference compared with Adam. I think this maybe because that the complexity of the image is not high, so that the loss function is rather smooth and "convex".

batch size: Normally, batch size within an appropriate range would not influence much to the model result, but a very small batch size may make the model learn too much abstractions and not converge to the minimal, and a very large batch size would make the training process quite slow. In theory, a larger batch size can yield a better result, but in our problems, we found a larger batch size (200, 500, 1000) can only yield an accuracy of 17.66%, 9.84% and 7.98% in classification problem. I think this is mainly due to the limited training data: the model stopped training before it learned much. We also find, that usually batch size of 5-20 would be a good choice, and selecting 1-5 is almost never good.

learning rate: This is the most important parameter in deep learning training. Given the number of epochs, we found that in classification problem for example, a larger learning rate have a better performance. In theory, we should first use a larger rate and get a result, and then based on the performance to decrease the rate until performance cannot be better. In our case, the best model has a learning rate of 5, which means the model "crazily" absorbs the negative gradient values. We think there are two reasons: firstly, the epochs we chose are too small, so that those small learning rate cannot reach a good accuracy before the training is stopped; secondly, our image loss function is quite convex (no much noise) so to get closer to the minimal point within limited epochs, a larger learning rate performs better.