2021/22 Frist Term

INM702 Programming and Mathematics for Artificial Intelligence

Report – Task 3

Fashion MNIST Image Classification

By Alex Collins and Suen Chi Hang

[Alex.Collins@city.ac.uk](mailto:Alex.Collins@city.ac.uk)

[Chi.Suen@city.ac.uk](mailto:Chi.Suen@city.ac.uk)

**Building the model**

We built a NN class to run build the neural net model. The main workflow of using the model is as follows:

The model is initialised and at that point learning rate, regulization (None, “L1” or “L2”), and regularization alphs (None or float in [0,1]) and random seed are set. The random seed ensures that all output for each NN object is reproduceable. It sets an attribute as np.random.default\_rng(random\_seed) and all distributions are driven by that.

We then add layers using the .add() method. Arguments are nodes and activation function. A NN method is passed as the activation function: options are NN.ReLu, NN.sigmoid, NN.softmax. The last layer has to have nodes = number of classes and activation NN.softmax. For this specific task we could have automated the creation of the final layer but we left it as a manual input to keep the code flexible for other tasks.

The final part of preparing the model is to pass the data and other phyper-arameters used to set parameters. The .model() method accepts train and test features and labels, minibatch size (default None to use full dataset), scale used for initializing weights (options are “He” and “Xavier” – default is He as that is better for ReLu activation function which we use more), and finally verbose. If verbose is set to True then user sees a print out of neural net architecture. .model() initialises the weights and biases matrices as empty matrices of the correct size.

Graphical user interface, text, application, email

Description automatically generated

*Figure: example of initializing a model.*

Finally we are ready to fit! We run the .fit() method with the following arguments: epochs – max number of epochs to run; min-epochs – min number of epochs to run regardless of stopping criteria; patience – number of epochs to wait until improvement for stopping; metric – either “train” or “valid” – to choose whether to use training or validation loss as stopping metric; verbose – if True will print out summary information for each epoch.

Finally, we have built very basic loss visualisation. The .plot\_loss() method shows a chart of each mini-batch loss through time. It’s interesting to see the variation by batch. In the case of our example model, there was not much improvement on an already really good result.

|  |  |
| --- | --- |
| Text  Description automatically generated | Chart, scatter chart  Description automatically generated |

*Figure: First image is output of .fit(): fitting with early stop. In the case of mini-bath SGD, the model fits very well after first epoch and doesn’t really improce much afterwards. Second image is output of .plot\_error()*

**Tuning parameters systematically**

**Number of hidden layers**

First, we need to decide the basic structure for tuning, i.e. number of layers. Then we build helper function based on the determined basic structure. As in earlier trials sigmoid activation performs poorly, we simply keep using ReLu activation. We will add hidden layer(s) of 1024 nodes with ReLu activation. Results of final epoch by no. of hidden layers:

1: 

2: 

3: 

So let’s tune the no. of nodes for 2 hidden layers, which has highest test accuracy of 87.4%.

Table

Description automatically generated

2-hidden-layer NN with n1=2048, n2=256 has highest accuracy of 88.4%. Note that for n1=2048, accuracy seems to have decreasing trend with n2 beyond 256, which may justify us to stop the expensive looping thereafter. However, as 2-hidden-layer is much slower than 1-hidden-layer using python built functions for neural network, we will also explore different number of nodes for 1-hidden-layer.

|  |  |
| --- | --- |
| Text  Description automatically generated with low confidence | The best for 1-hidden-layer is 1024 nodes, with accuracy of 87.28%.  As it is quite close to the optimal of 2-hidden-layer NN and beat most of them, we choose the 1-hidden-layer NN of 1024 nodes for first layer after computation time is also taken into account. |

### Learning rate and batch size

As update of parameters is directly affected by learning rate and batch size, let's tune them together.

|  |  |
| --- | --- |
| Table  Description automatically generated  Best: learning rate 0.05, batch size 16  Larger batch size tends to go with higher learning rate to maximize accuracy. This is because for smaller batch size, the loss is more volatile and it is more suitable to use smaller learning rate for smaller batch size. | Chart, line chart  Description automatically generated |

Regularization

We have built two types of regularization: L1, L2, where L1 uses the sum of absolute weight as penalty added to loss, while L2 uses the sum of squares of weights as penalty, each multiplied by lambda, which acts as extent of regularization. The rationale is to prevent weights from being too big. L1 forces many weights to zero and L2 forces weights to move towards zero which helps prevent overfitting). We will tune regularization lambda (which is alpha in our code).

Text

Description automatically generatedText

Description automatically generated

As test accuracy remains the same with regularization L1 and L2, we will not use L1 or L2.

Stopping Criteria

Stopping criteria is used for saving computation time and more importantly, also avoids over-fitting which may occur with excessive training. We design the NN to stop when there is no significant improvement (default threshold 2.5%) of loss compared with previous p epochs, where p is patience in our code. We also allow user to specify min and max number of epochs, and select whether to use training or validation loss for monitoring improvement.

Results: test accuracy is unchanged at 88.76% for patience 1 to 9. It also remains unchanged at 86.23% for improvement threshold [0.1, 0.05, 0.025, 0.1, 0.05] for short patience of 2. (see supplementary Fig. 1 & 2)

Conclusion

|  |  |
| --- | --- |
| The optimal neural network structure is 2-layer(1024/10) and optimal parameters are: learning rate 0.05, batch size 16, ReLu activation for hidden layers Softmax activation for output layer, no regularization L1 or L2. The test accuracy is 88.76%. As shown by loss curve, there is some degree of over-fitting, though regularization L1, L2 and stopping criteria don’t help. | Chart, line chart  Description automatically generated |

Reflection

Supplementary Figures

Table

Description automatically generated Text

Description automatically generated

S-Fig. 1 S-Fig. 2

S-Fig. 3 S-Fig. 4

Reference