Hyperparmaters　電機一　翁挺瑋

Most machine learning algorithms involve “hyperparameters” which are variables set before actually optimizing the model's parameters. Setting the values of hyperparameters can be seen as model selection, choosing which model to use from the hypothesized set of possible models. Hyperparameters are often set by hand, selected by some search algorithm, or optimized by some “hyper-learner”. Neural networks can have many hyperparameters, including those which specify the structure of the network itself and those which determine how the network is trained. In this project, we downloaded the fashion MNIST dataset and show the model by optimizing hyperparameters.

Considerations

The time required to train and test a model can depend upon the choice of its hyperparameters. A hyperparameter is usually of continuous or integer type, leading to mixed-type optimization problems. The existence of some hyperparameters is conditional upon the value of others, e.g. the size of each hidden layer in a neural network can be conditional upon the number of layers.

Tunability

Most performance variation can be attributed to just a few hyperparameters. The tunability of an algorithm, hyperparameter, or interacting hyperparameters is a measure of how much performance can be gained by tuning it. For an LSTM, while the learning rate followed by the network size are its most crucial hyperparameters, whereas batching and momentum have no significant effect on its performance.Although some research has advocated the use of mini-batch sizes in the thousands, other work has found the best performance with mini-batch sizes between 2 and 32.

Robustness

An inherent stochasticity in learning directly implies that the empirical hyperparameter performance is not necessarily its true performance. Methods that are not robust to simple changes in hyperparameters, random seeds, or even different implementations of the same algorithm cannot be integrated into mission critical control systems without significant simplification and robustification.

Reinforcement learning algorithms, in particular, require measuring their performance over a large number of random seeds, and also measuring their sensitivity to choices of hyperparameters. Their evaluation with a small number of random seeds does not capture performance adequately due to high variance. Some reinforcement learning methods, e.g. DDPG (Deep Deterministic Policy Gradient), are more sensitive to hyperparameter choices than others.

Optimization

Hyperparameter optimization finds a tuple of hyperparameters that yields an optimal model which minimizes a predefined loss function on given test data. The objective function takes a tuple of hyperparameters and returns the associated loss.

Learning Rate

Learning rate is the ratio of the weights adjustment with respect to the loss gradient. Higher learning rate enables the model to reach the local maximum faster, yet it could possibly skip some local maximum since each step it takes is too big. However, for this test we conclude that higher learning rates are better.
Testing 5 learning rates (0.0001, 0.001, 0.002, 0.01, 1.0) with the following parameters fixed:
Batch size: 512
Transfer function: relu
Number of epochs: 10

Optimization algorithm: nadam
Weight initialization: random uniform

Transfer Function

Next, we look into the effect of different transfer functions. The transfer functions are monotonically increasing continuous functions applied to the weighted input and simulate the behavior of a biological neuron for the neuron network. Different functions display different properties that fit for certain types of data. We study the exponential linear units (elu), linear, rectified linear unit (relu), scaled exponential linear units, sigmoid and tanh functions, keeping others contant as follow:
Batch size: 512
Number of epochs: 10
Optimization algorithm: nadam
Learning rates: 0.002
Weight initialization: random uniform
We find that most of the functions performs similarly expect for the linear function, which is expected. The traditional sigmoid function gets a slight upper hand here.

Batch Size

Batch size is the number of samples being propagated through the network with an iterative computation. Smaller batch size takes less memory and higher efficiency yet rougher gradient estimation.
Here we test different sizes of 20, 30, 50, 100 and 200.
Fixed hyper parameters:
Transfer function: sigmoid
Number of epochs: 10
Optimization algorithm: nadam
Learning rates: 0.002
Weight initialization: random uniform

And we find the when they are close, the 100 batch sizes give a better result.

Epoch

Then we take consideration of different number of epochs. Epochs denote the number of cycles we scan through all the sample, or how many time we train the model. While it seems the more times we train the model the better the result, if we train the model excessively (by setting the number of epochs too high), not only we consume more computational resources, the model also just “memorizes” the result instead of making meaningful learning which damages its predictive ability. We test the number of epoch from 1 to 50.
Fixed hyper parameters:
Batch size: 100
Transfer function: sigmoid
Optimization algorithm: nadam
Learning rates: 0.002
Weight initialization: random uniform
And we find the number of 38 gives the best result.

Optimization algorithm

Optimization algorithm are what the model applied to look for global maximum for accuracy. We tried stochastic gradient descent (sgd), adagrad, adadelta, adaptive moment estimation (adam) and nesterov-accelerated adaptive moment estimation (nadam).
Different optimization algorithms
Testing 5 different optimization algorithms
Fixed hyper parameters:
Batch size: 100
Transfer function: sigmoid
Learning rates: 0.002
Number of epochs: 38
Weight initialization: random uniform

And we find the nadam fits best here.

Weight initialization Method

Finally, we look into the weight initialization methods. In the construction a neural network we try to find out how the input Xs affect the output Y and each X has their own weights on the effect. Instead of setting all the weights to be zero at the beginning, we set them to a small amount in order to kick start the learning process of the network. And the weight initialization method is how we set thpse small amount initially.

We are testing 8 different weight initializations (glorot normal, glorot uniform, he normal, he uniform, lecun normal, lecun uniform, random uniform and uniform.

Fixed hyper parameters:

Batch size: 100

Transfer function: sigmoid

Learning rates: 0.002

Number of epochs: 38

Optimization algorithm: nadam

And we find that the lecun normal method gives the best accuracy.

Conclusion

We find in this project the best combination of hyperparameters are

Batch size: 100

Transfer function: sigmoid

Learning rates: 0.002

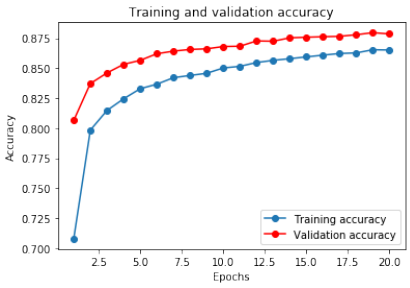
Number of epochs: 38

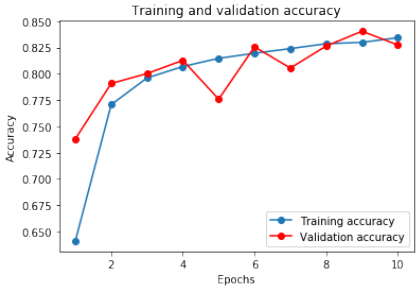
Optimization algorithm: nadam

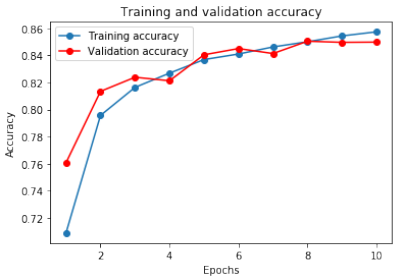
Weight initialization: lecun normal

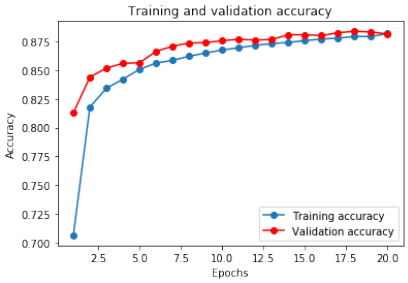
Which gives us a 0.894 accuracy. While the procedure we used in search for the optimization of hyperparameters is indeed a primitive and inefficient, we have learnt the principle underlying and achieved a better understanding of the hyperparameters which equips us well in preparation for the further study of machine learning.

Graph


Training and validation accuracy
Accuracy
Epochs
Training accuracy
Validation accuracy


Training and validation accuracy
Accuracy
Epochs
Training accuracy
Validation accuracy


Training and validation accuracy
Accuracy
Epochs
Training accuracy
Validation accuracy


Training and validation accuracy
Accuracy
Epochs
Training accuracy
Validation accuracy