CAP 5619 - Deep and Reinforcement Learning

Project1 Hua Huang

1 Task I-Neural Network Design

In these 3 networks, reLU are adopted except for the 1st layer. Since reLU is generally recognised as the best choice for most of the application scenarios of Neural Network, it does not saturate when it is fired, and derivative is always 1 if fired. This choice of activation functions can guarantee in the backpropogation, the gradients can be propogated to the deeper layer and hence the network can be trained efficiently.

(1) Fully connected Neural Network. There are 4 layers(exclude the input layer). The 1st layer has 256 units, activation function is tanh. The 2nd and 3rd layer has 256 units, and the activation function is reLU, the last layer is a softmax layer, which output the scores for each of the 10 classes. In total, there are 199946 parameters.

(2) Locally connected with no weights shared in the first three layers, where each neural/input is connected to the neurons in a local neighbor in the next layer. There are 4 layers, the 1st layer has 8 filters, each filter has a size of $[3, 3]$, and the activation function is sigmoid. The 2nd and 3rd layer also has 8 filters, each of size $[3, 3]$, and the activation function is reLU. The last layer is a softmax layer, which output the scores for each of the 10 classes. In total, There are 166186 parameters.

(3) CNN. There are 4 layers, the 1st layer has 8 filters, each filter has a size of $[3, 3]$, and the activation function is sigmoid. The 2nd and 3rd layer also has 8 filters, each of size $[3, 3]$, and the activation function is reLU. The last layer is a softmax layer, which output the scores for each of the 10 classes. In total, There are 9258 parameters.

2 Techniques for Optimization

2.1 parameter initialization strategies

(1) Fully connected NN (MLP): The frist 3 layers are initialized as *Identity(gains=3.)*, the last layer is initialized as *TruncatedNormal(mean=0.0, stddev=0.05)*, the biases are initialized as *Constant(value=0.5)*.
To test the parameter choice, 3 *gains* of $[0.1, 3, 10]$ are tested, the performance is given in Fig.1, we can see *gains=3* is most appropriate among these 3, *gains=0.1* will lead to very slow learning, and *gains=10* is too fast, it learns within 10 epochs, after that, it never learns anymore.

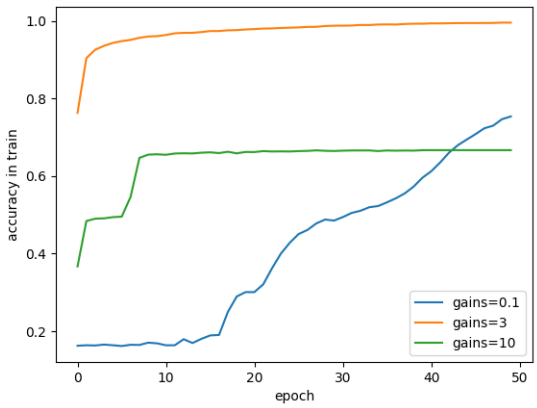


Figure 1: initialization in MLP

(2) Locally connected NN: All the 4 layers are initialized as *TruncatedNormal(mean=0.0, stddev=0.1)*, the biases are initialized as *Constant(value=0.5)*.
To test the parameter choice, 3 *stddev* of $[0.15, 0.1, 0.05]$ are tested, the performance is given in Fig.2, we can see *stddev=0.1* is most appropriate among these 3, *stddev=0.05* will lead to very slow learning, and *stddev=0.15* is too fast, it learns within 1 spoch, after that, it never learns anymore.

(3) CNN: All the 4 layers are initialized as *TruncatedNormal(mean=0.0, stddev=0.1)*, the biases are initialized as *Constant(value=0.5)*.

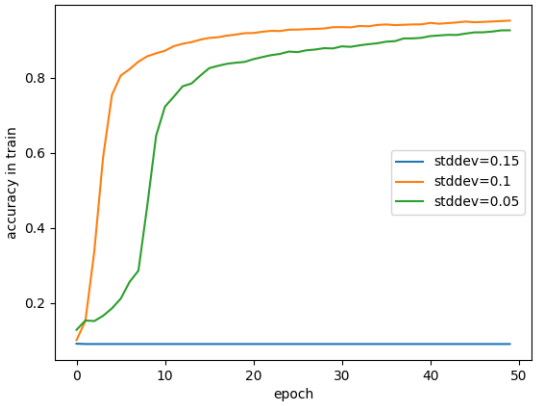


Figure 2: initialization in locally connected layer

To test the parameter choice, 3 *stddev* of $[1.0, 0.5, 0.02]$ are tested, the performance is given in Fig.3, we can see *stddev=0.5* is most appropriate among these 3, *stddev=0.02* will lead to very slow learning, and *stddev=1.0* is too fast, it learns within 1 spoch, after that, it never learns anymore.

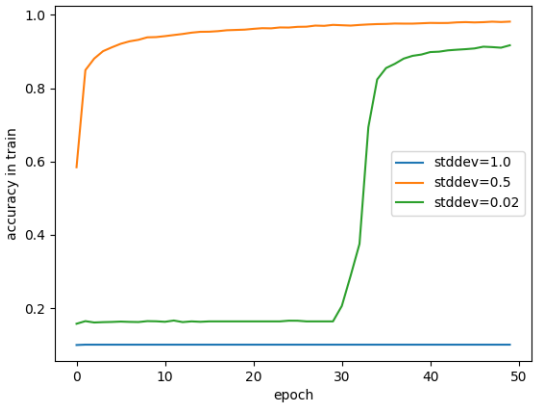


Figure 3: initialization in CNN

2.2 learning rate

(1) Fully connected NN (MLP): To test the parameter choice, 3 lr of $[0.25, 0.01, 0.00001]$ are tested, the performance is given in Fig.4, we can see *lr=0.01* is most appropriate among these 3, *lr=0.00001* will lead to very slow learning, and *lr=0.25* is too fast, it learns within 10 epochs, after that, it never learns anymore.

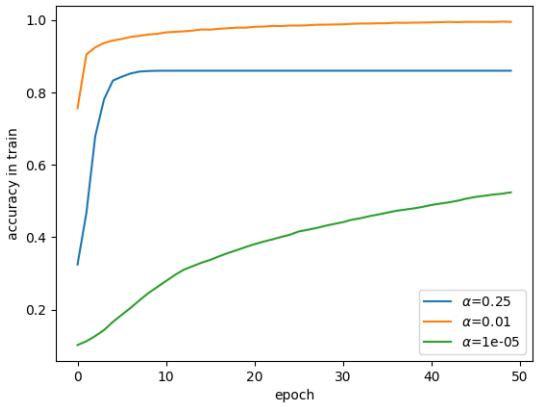


Figure 4: step size in MLP

(2) Locally connected NN: To test the parameter choice, 3 lr of $[0.5, 0.1, 0.01]$ are tested, the performance is given in Fig.5, we can see *lr=0.1* is most appropriate among these 3, *lr=0.01* will lead to very slow learning, and *lr=0.5* is too fast, it learns within 1 spoch, after that, it never learns anymore.

(3) CNN: To test the parameter choice, 3 lr of $[0.1, 0.01, 0.00001]$ are tested, the performance is given in Fig.6, we can see *lr=0.01* is most appropriate among these 3, *lr=0.00001* will lead to very slow learning, and *lr=0.1* is too fast, it learns within 1 spoch, after that, it never learns anymore.

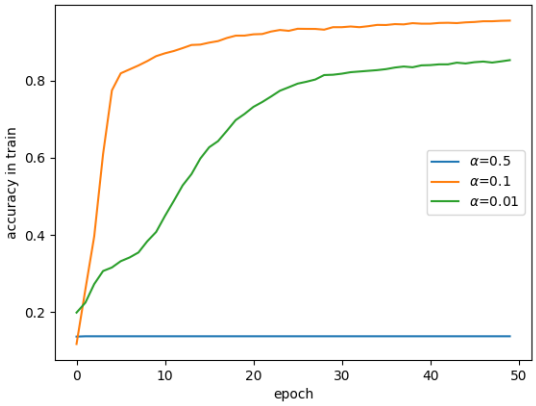


Figure 5: stepsize in locally connected layer

2.3 Batch size effects

If we do not consider the computational benefits, batch size is the smaller the better. Although large batch size will lead to a more accurate computation of the gradient, generally large batch size will lead to bad performance. The behind reason is guessed to be that since Neural Network is non-convex optimization, the noise(or we can say the inaccuracy) introduced by small--size batch learning can help the trainning escape the local optimum and hence obtain better performance.
(1) Fully connected NN (MLP): To test the parameter choice, 2 *batch size* of $[256, 8192]$ are tested, the performance is given in Fig.7, we can see *batch size=256* is more appropriate, and *batch size=8192* will lead to slow learning and bad final performance.

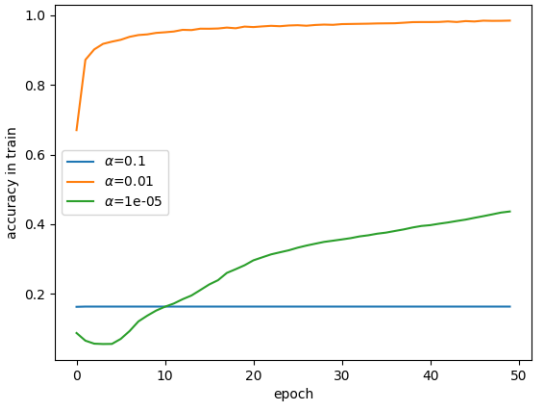


Figure 6: step size in CNN

(2) Locally connected NN: To test the parameter choice, 2 *batch size* of $[128, 2048]$ are tested, the performance is given in Fig.8, we can see *batch size=128* is more appropriate, and *batch size=2048* will lead to slow learning and bad final performance.

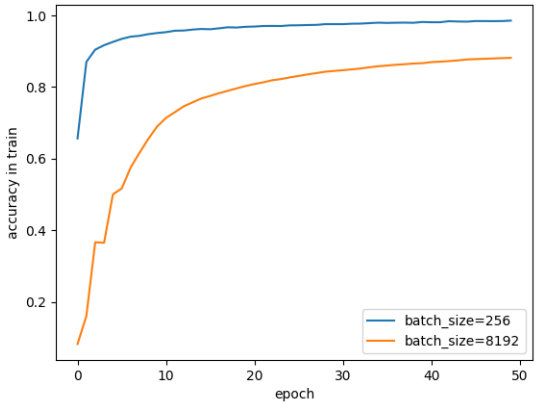


Figure 7: batch size in MLP

(3) CNN: To test the parameter choice, 2*batch size* of $[128, 4096]$ are tested, the performance is given in Fig.9, we can see *batch size=128* is more appropriate, and *batch size=4096* will lead to slow learning and bad final performance.

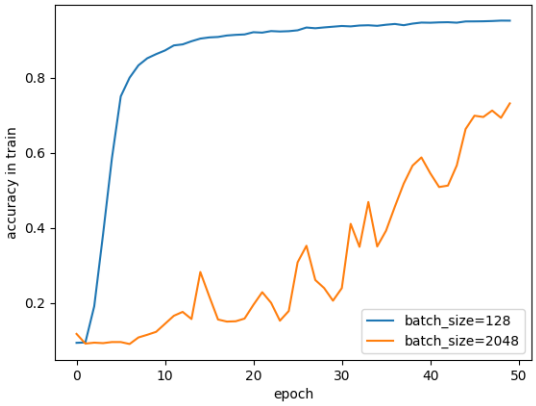


Figure 8: batch size in locally connected layer

2.4 Momentum

Three values $[0.5, 0.9, 0.99]$ of momentum are tested on these 3 net works. The momentum effects varies on the 3 networks:
For MLP, *momentum=0.9* will degrade the final performance, and it seems it arrives at plateau sooner then *momentum=0.5*. *momentum=0.99* leads to a failed learning.
For locally connected layer, all three momentum values leads to a failed learning. Since the momentum actually acts as a accumulation of prior gradients, we can naively understand it as equivalent to a increased batch size effect. The learning immediately falls to local minimum, and bad performance is observed.
For CNN, a very different behavior is observed. textitmomentum=0.9 has the best performance, *momentum=0.5* performs slightly worse, and *momentum=0.99* leads to a failed learning.

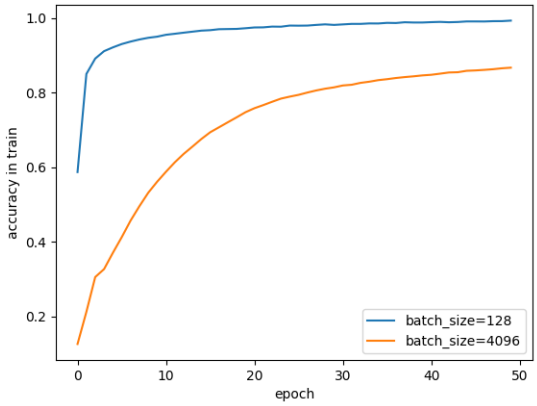
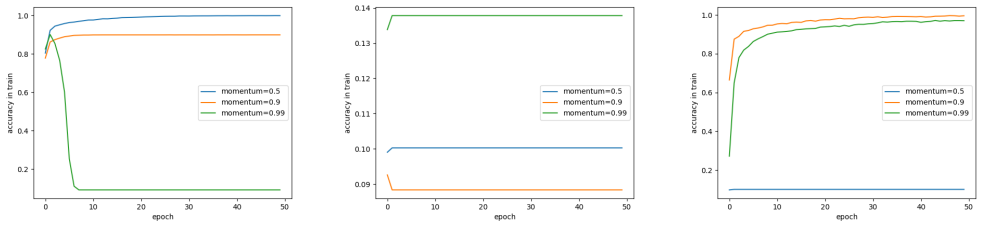


Figure 9: batch size in CNN



(a) MLP (b) Local (c) CNN

Figure 10: momentum effects

3 Techniques for improving generalization

3.1 ensemble networks

To improve the generalization performance, 6 CNN networks are trained independently. They have same architecture and differs in their initialization of their weights. Once they are trained, they are ensembled to predict the test set. As we can see in the table, the ensemble net has the highest test accuracy then all the 6 networks. Ensemble increase the final generalization capability.

Table 1: Test accuracy

Networks	1	2	3	4	5	6	ensemble
Accuracy	0.849	0.848	0.847	0.845	0.854	0.845	**0.856**

3.2 Dropout

During training, multiple realization of Dropout generates ensemble of networks, and these networks share hidden units in training. In such a way, it can generally improve the generation performance. (1) Fully connected NN (MLP): To test the parameter choice, 2 *dropouts* of $[0.2, 0.5]$ are tested, and in comparison, no dropout is also included. The performance is given in Fig.11, we can see Dropout degrade the trainning performance, but an appropriate chosen dropout probalbility of 0.2 does not harm the test performance, while dropout of 0.5 is too large and harms the trainning and generalization significantly.

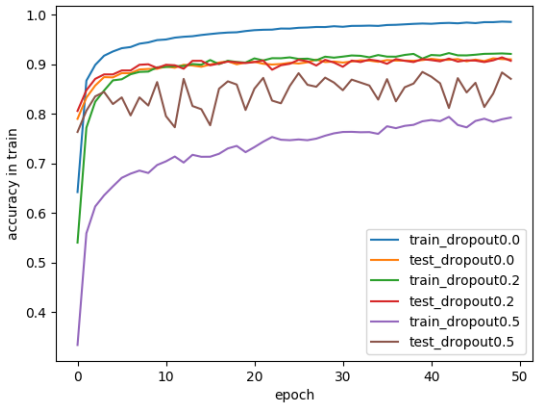


Figure 11: dropout effects in MLP

(2) Locally connected NN: To test the parameter choice, 2 *dropouts* of $[0.2, 0.6]$ are tested, and in comparison, no dropout is also included. The performance is given in Fig.12, we can see Dropout degrade the trainning performance, but an appropriate chosen dropout probalbility of 0.2 does not harm the test performance, while dropout of 0.6 is too large and harms the trainning and generalization.

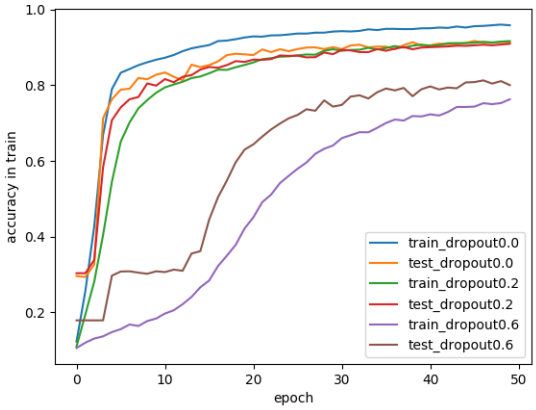


Figure 12: dropout effects in locally connected layer

(3) CNN: To test the parameter choice, $2 dropout$ of $[0.2, 0.6]$ are tested, and in comparison, no dropout is also included. The performance is given in Fig.13, we can see Dropout degrade the trainning performance, but an appropriate chosen dropout probalbility of 0.2 does not harm the test performance, while dropout of 0.6 is too large and harms the trainning and generalization, although it seems it still can catch up if we keep training.

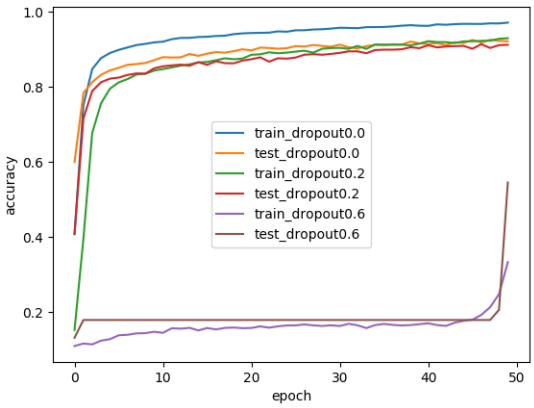


Figure 13: dropout effects in CNN

3.3 L1 regularization

L1 regularization is known as introducing sparsity in the network, it acts like feature selection and only keeps the significant weights.
(1) Fully connected NN (MLP): To test the parameter choice, 2 *regularization* of $[0.1, 0.2]$ are tested, and in comparison, no regularization is also included. The performance is given in Fig.14, we can see L1 regularization degrade the trainning performance, but an appropriate chosen L1 regularization of coefficient 0.1 does not harm the test performance, while L1 coefficient of 0.2 is too large and harms the trainning and generalization significantly. Eventually too many zeros are introduced in the network and it failed to train at last for coefficient of 0.2.

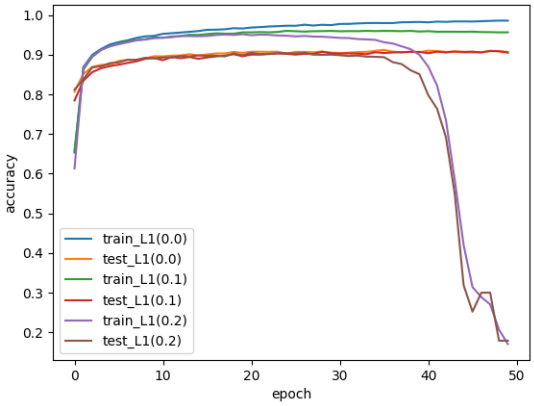


Figure 14: L1 regularization effects in MLP

(2) Locally connected NN: To test the parameter choice, 2 *regularization* of $[0.0005, 0.002]$ are tested, and in comparison, no regularization is also included. The performance is given in Fig.15, we can see L1 regularization degrade the test performance, even if we have a coefficient as small as 0.0005. Coefficient of 0.002 further decrease the final test accuracy. It seems L1 regularization is very dangerous for locally connected net work and can lead to disasters if we chose coefficients carelessly.

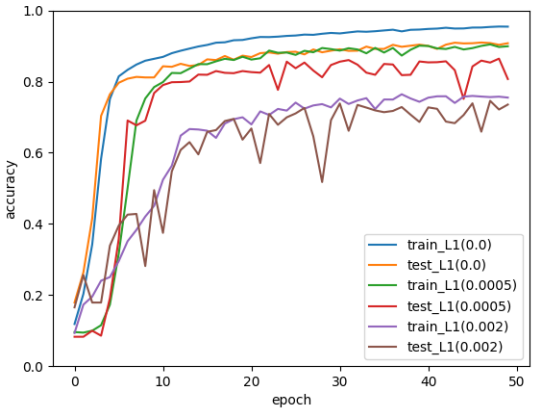


Figure 15: L1 regularization effects in locally connected layer

(3) CNN: To test the parameter choice, $2regularization$ of $[0.01, 0.02]$ are tested, and in comparison, no regularization is also included. The performance is given in Fig.16, we can see L1 regularization degrade the test performance. Coefficient of 0.02 further decrease the final test accuracy and slower the learning. But it's relatively robust to L1 regularization compared with locally connected layer.

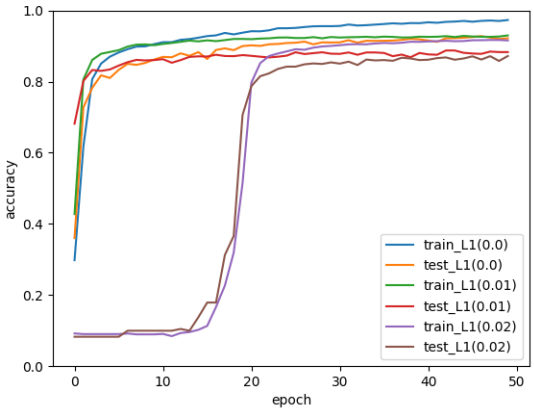


Figure 16: L1 regularization effects in CNN