CS440 MP4
Team member: Mingren Feng (mingren3) ECE448, Q3;
Wenyao Jin (wenyaoj2) CS440, Q4
Ziyi Chen (ziyic2) CS440, R3
Date: 2019/04/22

Part 1: Q-learning (Snake)

1. During training phase, the the avg point gradually increase from zero to 23~25. The agent was going all possible states, until most of the state reaches Ne, and then it's trained for more sophisticated cases. During test phase, the agents is able to avoid awkward way of dying (when there is other action the can help it escape from dying), and the result are 22~25 which also varies for different check point. The training process is done by comparing state with previous state and using the formulate to update Q and N. Exploration is simply by using the formula and data in Q to determine the best case.
2. Used C = 40, Ne = 40, gamma = 0.7 that gives the best result. The test result is 22.7882 with max of 45 and min of 3 after about 40000 trains, the time is about 439.83s.
3. I also included the case when snake is two steps from wall to anticipate potential danger, and this makes the convergence a lot slower, the result is around 10 after 30000 trains, 16 after 40000 trains. After 100000 trains, the test gives 22.52 average points with Max of 50 and min of 2, it seems like this would be hard to get trained but gives higher max points.

Part 2: Deep Learning (MNIST Fashion)

1.Briefly describe any optimizations you performed on your functions for faster runtime.

We avoid using for-loop to do calculation, instead, we use broadcast, np.sum() and calculate on entire arrays as much as possible.

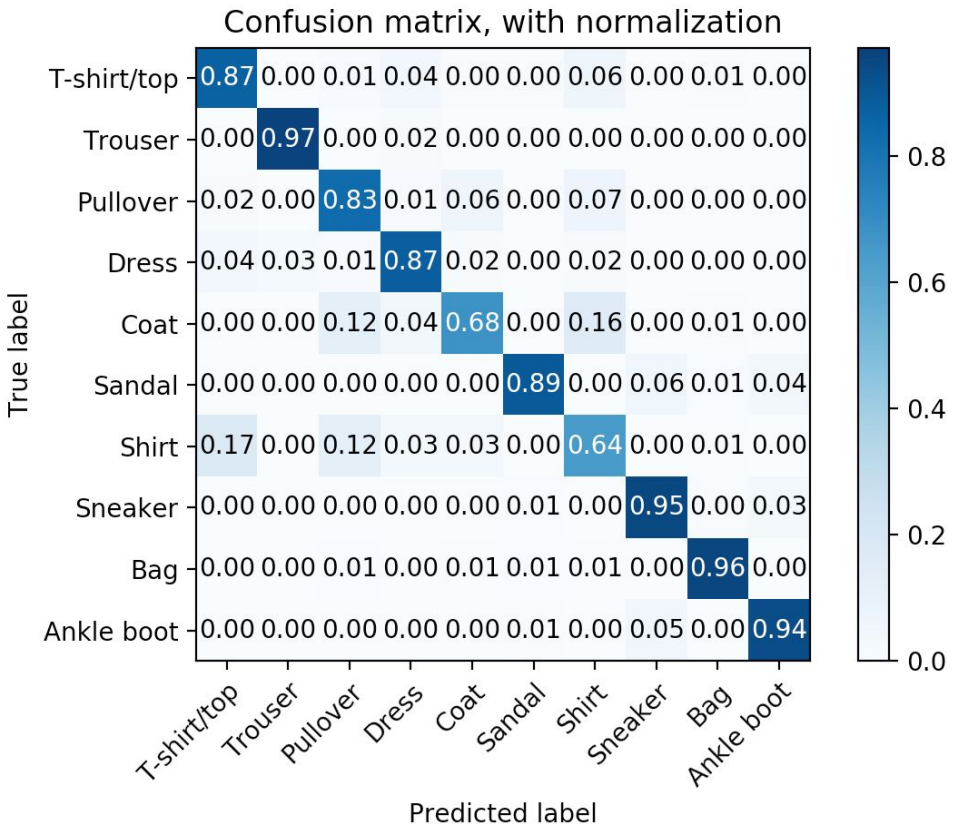
For `cross_entropy` function, we use np.sum() to calculate the sum of exp(Fik) and the sum of Fiyi, and apply array division to obtain loss. For dF, when calculating 1{j=fy}, we use array subtraction to avoid using IF-ELSE.

For `affine_backward` function, we use np.matmul() to calculate dA,dW and dB. Especially for calculating dB, we multiply dZ with np.ones() array to get dB, which can avoid for-loop.

2. Report Confusion Matrix, Average Classification Rate, and Runtime of Minibatch gradient for 10, 30, 50 epochs.

2.1. 10 epoches

Confusion Matrix



Average Classification Rate

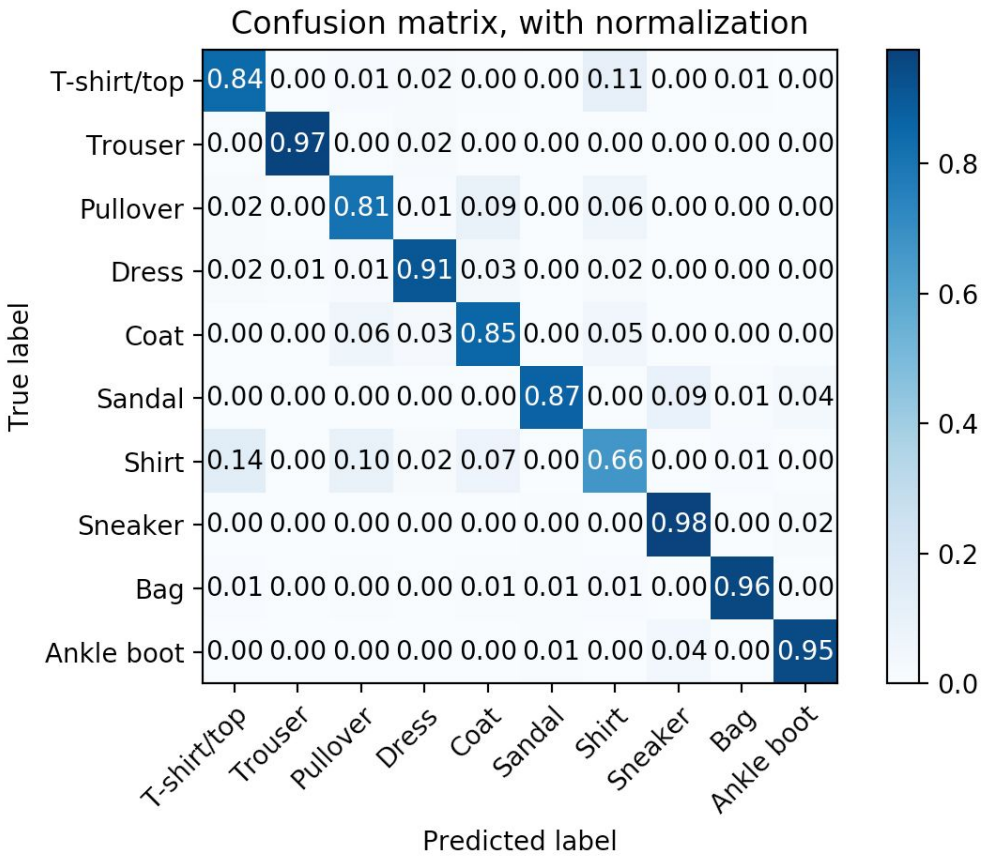
0.861

Runtime of minibatch gradient function

31.45s on EWS

2.2. 30 epoches

Confusion Matrix



Average Classification Rate

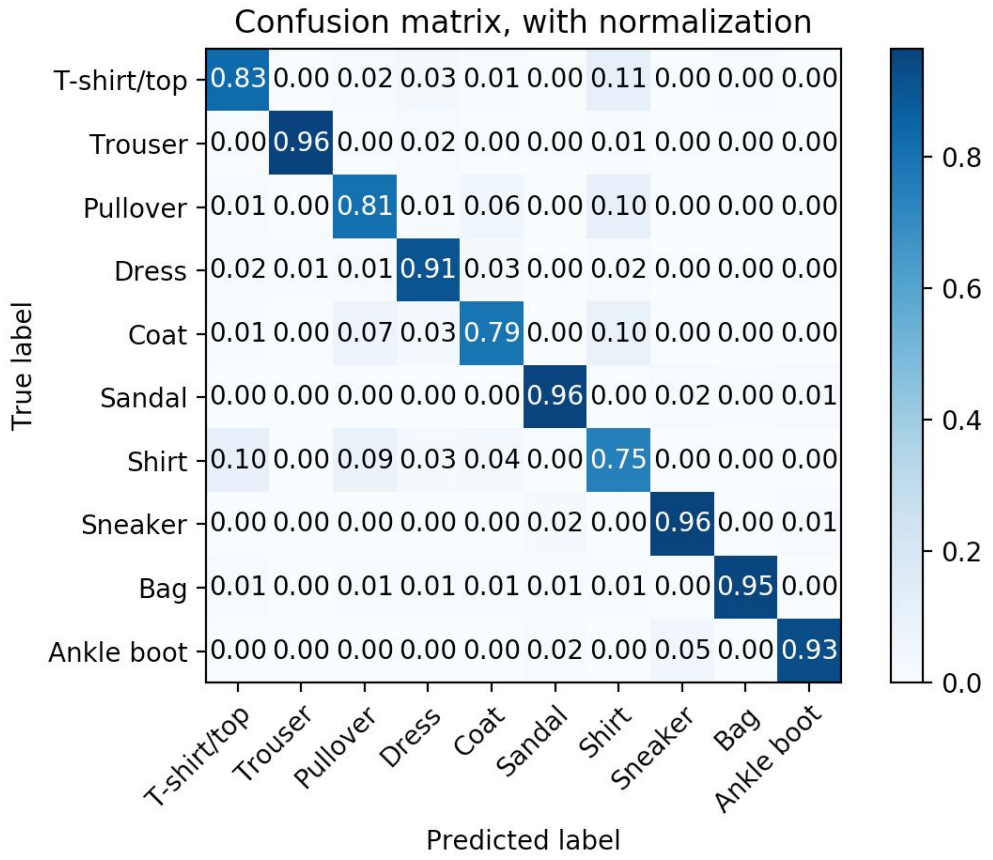
0.8803

Runtime of minibatch gradient function

90.50s on EWS

2.3. 50 epoches

Confusion Matrix



Average Classification Rate

0.8854

Runtime of minibatch gradient function

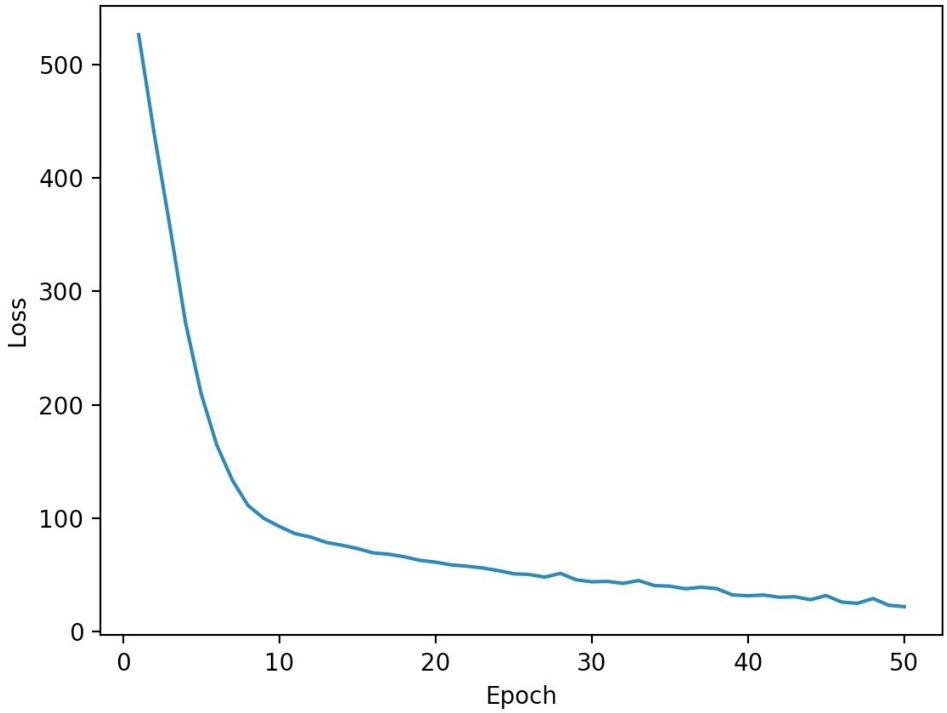
163.75s on EWS

3. Epoches VS losses for 50 epochs

Losses:

[526.6492140464547, 438.66935096487634, 357.2537095269296, 272.3515571537075, 209.63599961032196, 164.35503939537344, 133.23879106942567, 111.0452582977622, 99.70147055477715, 92.51447061951303, 86.16938396808094, 83.13118398699362, 78.42423295254994, 75.92072630703186, 72.9425505036083, 69.20141648267635, 67.93872347142805, 65.6864953910961, 62.5694550418263, 60.94133787126343, 58.546817862979175, 57.39321114933819, 55.872728620108205, 53.51598138642558, 50.69948059683583, 50.0539850630817, 47.73393993512668, 51.07275381324379,

45.369460396755656, 43.64212910299977, 43.95768251827847, 42.23117099744098, 44.71186968050424, 40.27875171387371, 39.768008943346416, 37.454539719932974, 38.83274907080152, 37.60223147794326, 32.02875167085795, 31.29394469466778, 31.89866407089859, 29.99058573681394, 30.38651107880154, 27.837679020023142, 31.43336329586559, 25.700854129527915, 24.589965356758967, 28.873116673703795, 22.904511048620936, 21.68907528879001]



4. Observations on trends & explanations

The loss decreases rapidly in first 10 epochs, then decreases slowly. At around the 28th epoch, the loss rebounds a little bit, and between the 30th epoch and 50th epoch, there will be “rebound” losses every several epochs.

The trend is as expected. The loss fluctuates in a tolerable range and is expected to converge.

In the beginning, the weights are far from correct and updated rapidly. Therefore the loss decreases quickly. After that, the weights are close to the trained values and updated more slowly than before. When it is close to convergence, the weights can be overcorrected each epoch and cause loss fluctuation.

Extra Credit

In this part, we use all online materials from this link:
https://github.com/MorvanZhou/PyTorch-Tutorial/blob/master/tutorial-contents/401_CNN.py

Convolution neural network is popular in the area of image identification. The complete convolution neural network can be divided into three main layers convolution layer, pooling layer and fully connected layer.

In the convolution layer, the filters will be used on the objective image. In this case, each part of the image should convolute with the filter and then the filter will move on the objective image in specific step size until the whole image is filtered by the filter. For instance, if we set the number of filters is 5. When all 5 filters finish their jobs, it will generate 5 new matrix which represent the dimension of the image. Then after relu function, they will enter the pooling layers. The objective of pooling layer is to reduce attributes and compress the image. We use maxpooling function to finish pooling layer, which will choose the most obvious attribute of one part of matrix. After finish convolution layer and pooling layer the output will be inputted into the full connection layer and get the final label.

In this part, we use pytorch to complete the CNN. At first we also shuffle the present dataset. We design three convolution layers in this part and set study rate as 0.001, batch size as 200. We set epoch as 30 in order to make it complete convergence. In the first convolution layer, because the image is black and white we set in_channel as 1 and set 16 filters. We set the size of filter as 5*5 and step size is 1. At last, in order to keep the size of image we set padding as 2. Then we use relu function as the activation function and use maxpooling function as pooling layer and set the matrix of maxpooling as 2*2.

In the second convolution layer, the input channel is 16 because of 16 filters in the first layer and the filters should be 32. We keep other attributes. Then in the third convolution layer, besides changing the input channel and output channel we change the size of matrix of maxpooling to 1*1 in order to keep the size of image as integer. Then we set the fully connection layer. The output shape should be 48*7*7 and the output label should be 10.

Then, the same as neural network, we forward it, calculate losses and backward it. After 30 epoches when it complete convergence, we get the result.

After 30 epochs, the results are as follow:

Confusion matrix

```
[[0.86       0.         0.03       0.015      0.         0.
  0.09       0.         0.005      0.        ]
```

```
 [0.         0.99014778 0.         0.00985222 0.         0.
  0.         0.         0.         0.        ]
 [0.02336449 0.         0.89719626 0.0046729  0.03271028 0.
  0.04205607 0.         0.         0.        ]
 [0.02105263 0.00526316 0.01052632 0.92105263 0.01578947 0.
  0.02631579 0.         0.         0.        ]
 [0.00456621 0.         0.06392694 0.0456621  0.82191781 0.
  0.05936073 0.         0.00456621 0.        ]
 [0.         0.         0.         0.         0.         0.97948718
  0.         0.02051282 0.         0.        ]
 [0.06598985 0.         0.05583756 0.01015228 0.05583756 0.
  0.81218274 0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         1.         0.         0.        ]
 [0.01546392 0.         0.00515464 0.00515464 0.         0.01030928
  0.         0.         0.96391753 0.        ]
 [0.         0.         0.         0.         0.         0.0106383
  0.         0.05851064 0.         0.93085106]]
avg_class_rate 0.9165

class_rate_per_class
[0.86,0.99014778,0.89719626,0.92105263,0.82191781,0.97948718,0.81218274,1.
,0.96391753,0.93085106]
```

Epoch VS Losses

```
Epoch:  0 | train loss: 0.4852 | test accuracy: 0.85
Epoch:  1 | train loss: 0.3028 | test accuracy: 0.88
Epoch:  2 | train loss: 0.3621 | test accuracy: 0.90
Epoch:  3 | train loss: 0.2138 | test accuracy: 0.90
Epoch:  4 | train loss: 0.1984 | test accuracy: 0.91
Epoch:  5 | train loss: 0.2297 | test accuracy: 0.90
Epoch:  6 | train loss: 0.2307 | test accuracy: 0.91
Epoch:  7 | train loss: 0.1562 | test accuracy: 0.92
Epoch:  8 | train loss: 0.1798 | test accuracy: 0.91
Epoch:  9 | train loss: 0.1642 | test accuracy: 0.91
Epoch:  10 | train loss: 0.2144 | test accuracy: 0.91
Epoch:  11 | train loss: 0.1827 | test accuracy: 0.92
Epoch:  12 | train loss: 0.1888 | test accuracy: 0.91
Epoch:  13 | train loss: 0.1595 | test accuracy: 0.91
Epoch:  14 | train loss: 0.1586 | test accuracy: 0.92
Epoch:  15 | train loss: 0.1733 | test accuracy: 0.91
Epoch:  16 | train loss: 0.1184 | test accuracy: 0.91
Epoch:  17 | train loss: 0.0692 | test accuracy: 0.92
Epoch:  18 | train loss: 0.1304 | test accuracy: 0.92
Epoch:  19 | train loss: 0.1166 | test accuracy: 0.92
Epoch:  20 | train loss: 0.1022 | test accuracy: 0.92
Epoch:  21 | train loss: 0.0805 | test accuracy: 0.92
Epoch:  22 | train loss: 0.0346 | test accuracy: 0.92
Epoch:  23 | train loss: 0.0862 | test accuracy: 0.92
Epoch:  24 | train loss: 0.0721 | test accuracy: 0.92
Epoch:  25 | train loss: 0.0400 | test accuracy: 0.92
Epoch:  26 | train loss: 0.0681 | test accuracy: 0.92
Epoch:  27 | train loss: 0.0515 | test accuracy: 0.92
Epoch:  28 | train loss: 0.0603 | test accuracy: 0.91
Epoch:  29 | train loss: 0.0497 | test accuracy: 0.92
```

Comparing the results of NN with CNN, we find that CNN provides higher accuracy overall and for each class. For some class the accuracy can be 100% with CNN. CNN takes fewer epochs to reach the highest accuracy than NN.