Title page:

1. Shangyi Xiong sxiong13 cs440/R3
2. Junzhe Wu junzhew3 cs440/Q4
3. Yi Zhang yzhan230 cs440/R3

Part 1:

1. Briefly describe the implementation of your agent snake.
   - How does the agent act during train phase?

     The agent gets the previous action, previous state, previous points, current state and current points first, and then discretizes the current state and previous state.

     If the previous state and previous action are not None, the agent is going to calculate corresponding alpha, previous Q-value based on previous action and current max Q value.

     If the snake is not dead yet, current action will be chosen based on max f function. If the snake is dead, action will not make any effect, so just return 0.

   - How does the agent act during test phase?

     During test phase, the agent already generates a good Q-table, so there is no need to update the Q-table anymore. However, the agent still needs to discretizes the previous state and current state and decide an action. Another difference is that the action is not chosen based on f-value anymore. The best action is chosen based on max Q-value in the Q-table instead.

2. Use Ne, C (or fixed alpha?), gamma that you believe to be the best. After training has converged, run your algorithm on 1000 test games and report the average point.
   - Give the value of Ne, C (or fixed alpha) you believe to be the best.

     The best value for Ne is clearly 20 because it has significantly more average points after 10000 training than others, which means it learns fastest. I believe C is the best at 20.

Ne	C	Average points after 10000 training
20	20	24
20	40	22

20	60	24
20	80	22
40	40	14
60	40	12

Table 1. Average points after 10000 training for varying Ne and C

- Report the training convergence time.

  The training takes 445 seconds to converge.

- Report average point on 1000 test games.

  The average point on 1000 test games is 21.

3. Describe the changes you made to your MDP(state configuration, exploration policy and reward model), **at least make changes to state configuration**. Report the performance (the average points on 1000 test games). Notice that training your modified state space should give you at least 10 points in average for 1000 test games. Explain why these changes are reasonable, observe how snake acts after changes and analyze the positive and negative effects they have. **Notice again, make sure your submitted agent.py and q_agent.npy are without these changes and your changed MDP should not be submitted.**

   I added a state which is the relative position between the snake head and snake tail. The average points for 1000 test games are 22. There is a little improvement compared to before. This makes sense because the snake tries to avoid the rest of its body even if the body is not next to the tail because it knows that there is large possibility that between its head and its tail, there are some parts of its body, so it tries to avoid that path in advance.

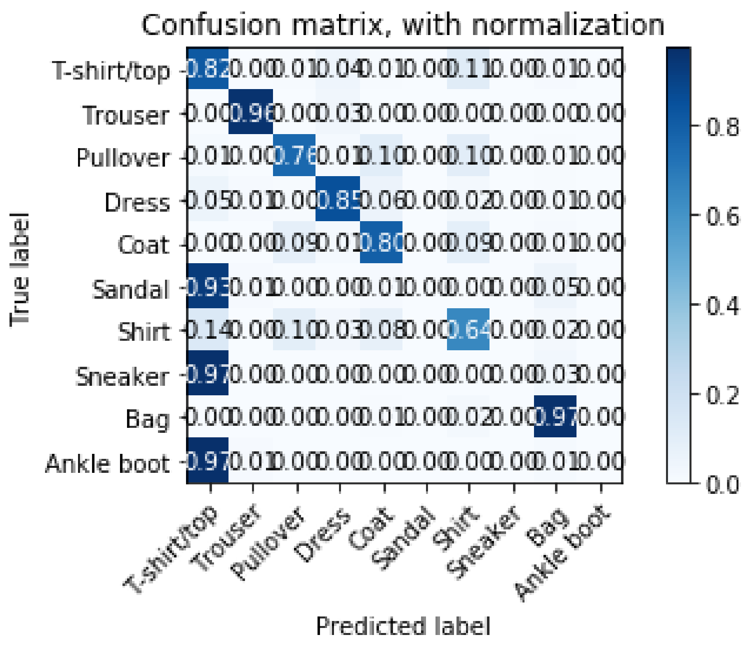
Part 2:

1. Briefly describe any optimizations you performed on your functions for faster runtime.
    1) I use many numpy functions such as **np.dot**, **np.sum** and **np.exp** as opposed to native python loops. And I transpose the matrixes in **affine_backward(dZ, cache)** in order to use the **np.dot** directly.
    2) I use broadcasting in **affine_forward(A, W, b)** as opposed to native python loops.
2. Report Confusion Matrix, Average Classification Rate, and Runtime of Minibatch gradient for 10, 30, 50 epochs. This means that you should have 3x3=9 total items in this section.

10 epochs:

1) Confusion Matrix

```
[[0.821 0.003 0.007 0.044 0.005 0.    0.112 0.    0.008 0.   ]
 [0.001 0.961 0.    0.033 0.003 0.    0.002 0.    0.    0.   ]
 [0.013 0.001 0.761 0.014 0.105 0.    0.1   0.    0.006 0.   ]
 [0.051 0.011 0.004 0.855 0.055 0.    0.019 0.    0.005 0.   ]
 [0.002 0.002 0.087 0.015 0.798 0.    0.088 0.    0.008 0.   ]
 [0.929 0.009 0.003 0.003 0.009 0.    0.001 0.    0.046 0.   ]
 [0.136 0.002 0.096 0.025 0.078 0.    0.644 0.    0.019 0.   ]
 [0.97  0.    0.    0.001 0.    0.    0.    0.    0.029 0.   ]
 [0.003 0.    0.001 0.003 0.006 0.    0.018 0.    0.969 0.   ]
 [0.974 0.009 0.001 0.001 0.    0.    0.    0.    0.015 0.   ]]
```



2) Average Classification Rate

   0.5809

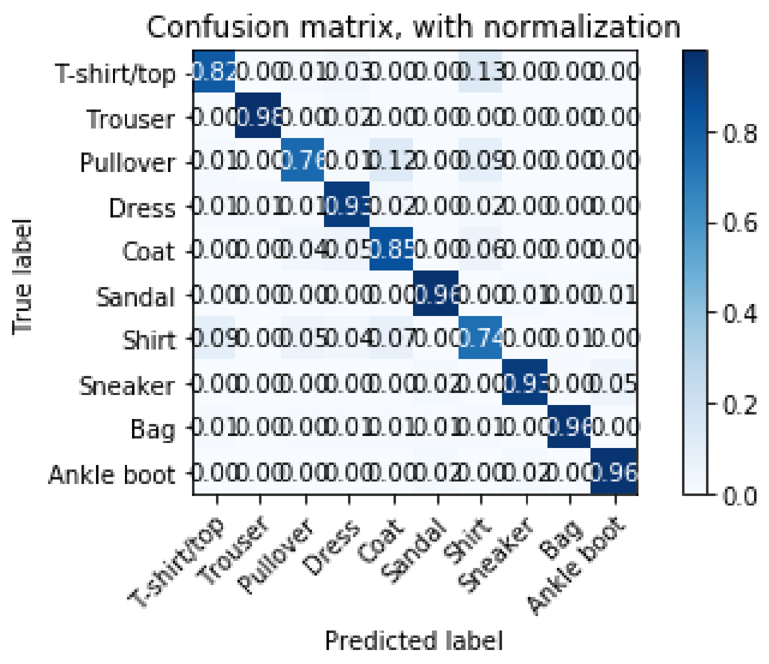
3) Runtime of your minibatch gradient function

   time cost 138.64681577682495 s

30 epochs:

1) Confusion Matrix

```
[[0.818 0.001 0.01  0.034 0.003 0.003 0.128 0.    0.003 0.   ]
 [0.001 0.976 0.002 0.018 0.002 0.    0.001 0.    0.    0.   ]
 [0.015 0.001 0.761 0.012 0.12  0.001 0.087 0.    0.003 0.   ]
 [0.011 0.009 0.005 0.933 0.023 0.003 0.016 0.    0.    0.   ]
 [0.002 0.    0.038 0.048 0.852 0.001 0.056 0.    0.003 0.   ]
 [0.003 0.    0.    0.001 0.    0.962 0.002 0.015 0.002 0.015]
 [0.09  0.002 0.054 0.036 0.074 0.    0.739 0.    0.005 0.   ]
 [0.    0.    0.    0.    0.    0.02  0.    0.927 0.004 0.049]
 [0.009 0.    0.004 0.005 0.007 0.006 0.008 0.001 0.959 0.001]
 [0.    0.    0.    0.    0.    0.021 0.    0.021 0.001 0.957]]
```

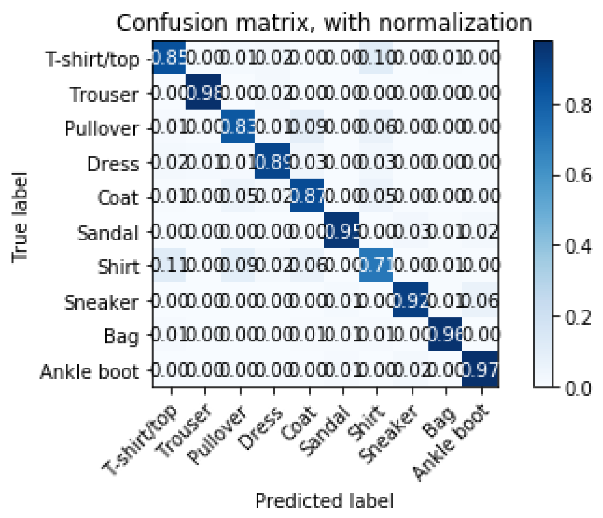


2) Average Classification Rate
   0.8884
3) Runtime of your minibatch gradient function
   time cost 389.3407070636749 s

50 epochs:

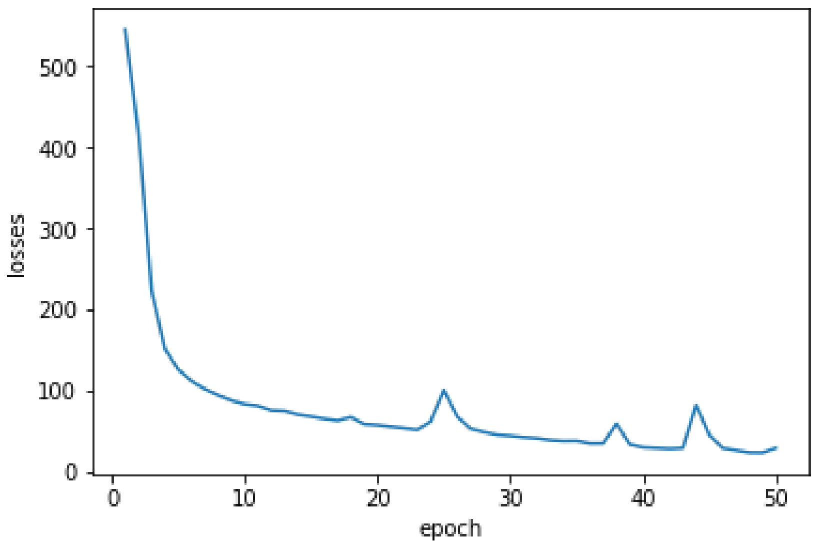
1) Confusion Matrix

```
[[0.852 0.002 0.012 0.018 0.002 0.003 0.104 0.    0.007 0.   ]
 [0.001 0.976 0.001 0.019 0.001 0.    0.001 0.    0.001 0.   ]
 [0.01  0.    0.826 0.008 0.093 0.    0.063 0.    0.    0.   ]
 [0.022 0.012 0.01  0.891 0.032 0.001 0.029 0.    0.003 0.   ]
 [0.007 0.    0.047 0.017 0.872 0.    0.054 0.    0.003 0.   ]
 [0.001 0.    0.    0.001 0.    0.946 0.    0.026 0.005 0.021]
 [0.111 0.001 0.088 0.022 0.063 0.    0.709 0.    0.006 0.   ]
 [0.    0.    0.    0.    0.    0.015 0.    0.918 0.006 0.061]
 [0.009 0.001 0.002 0.002 0.005 0.005 0.008 0.002 0.964 0.002]
 [0.    0.    0.    0.    0.    0.009 0.    0.021 0.    0.97 ]]
```



2) Average Classification Rate
   0.8924
3) Runtime of your minibatch gradient function
   time cost 668.3061945438385 s

3. The plot of epochs vs losses



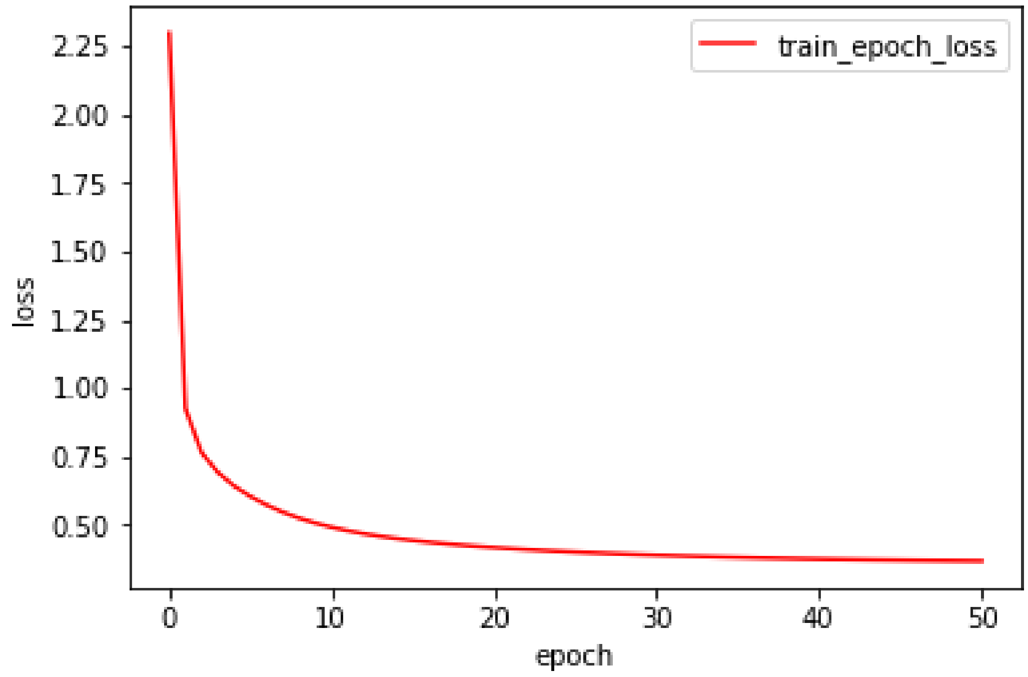
4. Describe any trends that you see. Is this expected or surprising? What do you think is the explanation for the observations?
   The loss decreases fast at first and then converges. It is expected. Since we use gradient descent here, at first the loss is huge and the gradient of w is also big, then the w has a great change and the loss also has a great change. When w is closed to the optimum, the gradient of w is very small, so the w and loss converge.

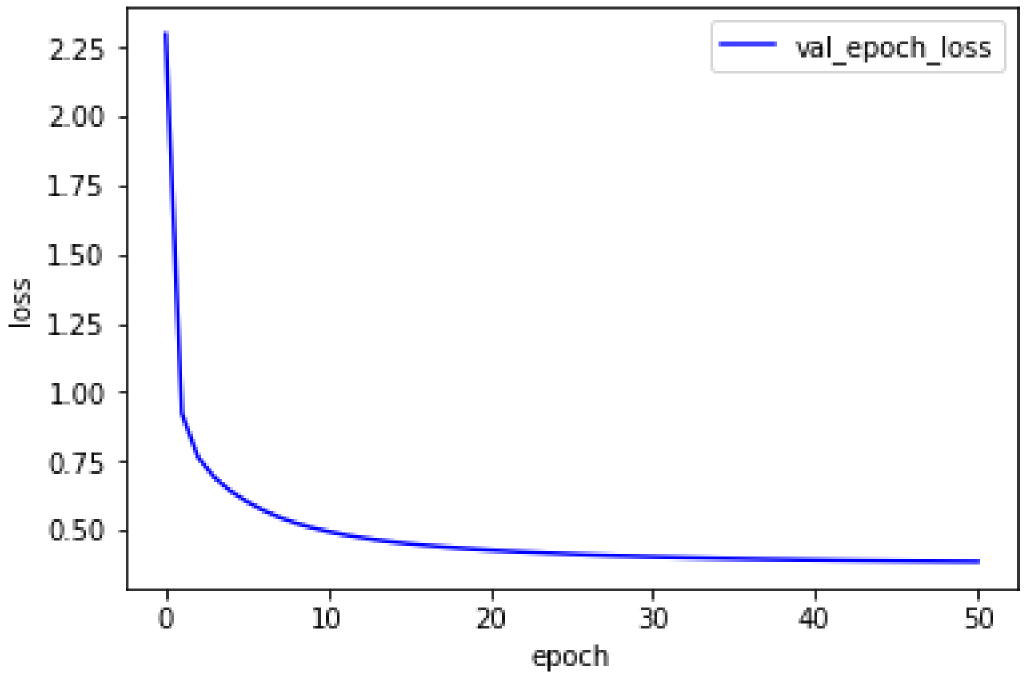
5. Extra credit:

I have made a code of CNN in the homework 2 of CS446, and I use this as a reference.

The website of CS446 and HW2:
http://mjt.cs.illinois.edu/courses/ml-s19/
https://courses.engr.illinois.edu/cs446/sp2019/secure/MJT/hw2.pdf

I use Pytorch to implement the CNN. In order to use CNN, I need to transfer the data in to the [x,28,28] format. CNN can extract the feature of each image and the feature is more and more accurate with the increasing number of convolution layer. I use 2 convolution layers and 2 fully connected layer here. And I also use GPU to speed up the calculation. The results are shown below:

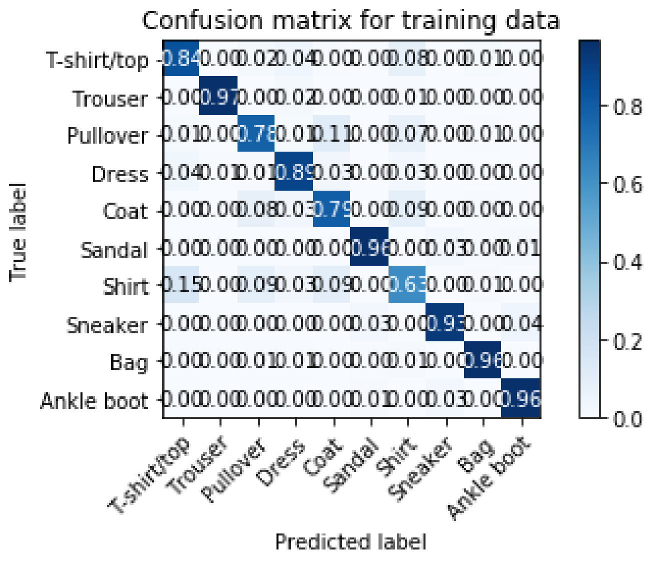




These 2 pictures show the change of training loss and test loss for 50 epoches.

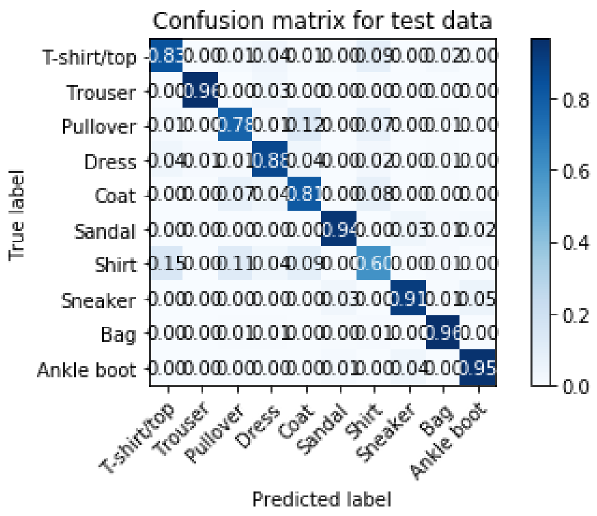
1. Confusion matrix for training data:

```
[[8.406e-01 1.400e-03 1.760e-02 4.240e-02 4.600e-03 1.000e-03 8.020e-02
  0.000e+00 1.220e-02 0.000e+00]
 [3.200e-03 9.652e-01 2.000e-03 2.120e-02 2.400e-03 4.000e-04 5.200e-03
  0.000e+00 4.000e-04 0.000e+00]
 [1.160e-02 1.000e-03 7.812e-01 1.220e-02 1.130e-01 1.000e-03 7.420e-02
  0.000e+00 5.800e-03 0.000e+00]
 [3.560e-02 7.200e-03 8.200e-03 8.872e-01 3.160e-02 4.000e-04 2.620e-02
  4.000e-04 3.200e-03 0.000e+00]
 [2.800e-03 2.200e-03 7.900e-02 3.020e-02 7.946e-01 6.000e-04 8.660e-02
  0.000e+00 4.000e-03 0.000e+00]
 [2.000e-04 0.000e+00 0.000e+00 8.000e-04 2.000e-04 9.594e-01 2.000e-04
  2.660e-02 4.000e-03 8.600e-03]
 [1.500e-01 2.400e-03 8.880e-02 3.080e-02 9.140e-02 0.000e+00 6.252e-01
  0.000e+00 1.120e-02 2.000e-04]
 [0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00 2.920e-02 0.000e+00
  9.284e-01 1.600e-03 4.080e-02]
 [4.000e-03 2.000e-04 7.200e-03 5.800e-03 3.200e-03 4.600e-03 1.000e-02
  3.200e-03 9.612e-01 6.000e-04]
 [2.000e-04 2.000e-04 0.000e+00 4.000e-04 0.000e+00 8.400e-03 4.000e-04
  2.600e-02 1.000e-03 9.634e-01]]
```



2. Confusion matrix for test data

```
[[0.835 0.001 0.013 0.039 0.005 0.004 0.085 0.    0.017 0.001]
 [0.002 0.964 0.001 0.027 0.002 0.    0.004 0.    0.    0.   ]
 [0.014 0.001 0.775 0.011 0.118 0.002 0.072 0.    0.007 0.   ]
 [0.036 0.011 0.011 0.877 0.037 0.    0.023 0.    0.005 0.   ]
 [0.002 0.    0.07  0.035 0.811 0.001 0.077 0.    0.004 0.   ]
 [0.    0.    0.002 0.001 0.    0.939 0.    0.034 0.008 0.016]
 [0.146 0.003 0.114 0.036 0.09  0.    0.602 0.    0.008 0.001]
 [0.    0.    0.    0.    0.    0.032 0.    0.913 0.005 0.05 ]
 [0.003 0.001 0.006 0.006 0.003 0.004 0.008 0.002 0.965 0.002]
 [0.    0.    0.    0.    0.    0.013 0.    0.037 0.    0.95 ]]
```



3. average classification rate
   Training: 0.87064
   Test: 0.8631
4. average classification rate per class
   Training: [0.8406, 0.9652, 0.7812, 0.8872, 0.7946, 0.9594, 0.6252, 0.9284, 0.9612, 0.9634]
   Test: [0.835, 0.964, 0.775, 0.877, 0.811, 0.939, 0.602, 0.913, 0.965, 0.95]

We can find that the performance of CNN is not better than the normal neural network. The average Classification Rate of normal neural network is 0.8924. I think it might because the number of convolution layer are not enough and I still do not find the appropriate hyperparameter.

Statement of Contribution:

The code of Part 1 is written by Shangyi Xiong.
The code of Part 2 is written by Junzhe Wu.
The code of extra credit is written by Yi Zhang.
The report is written by all three members.