ECE 4984   Homework3

Problem 1

In order to formulate this problem as a MDP, I need to specify the set of states, the set of actions, the transition function and the reward function.

Set of states: The set of states is shown in below. There is a total of 33 states. The left column represents the energy that the robot current has and the top row represents the current position of the robot. For example, at S15, the robot is at top of the hill and has four units of energy left. The terminal states are S11, S12 and S33 because at those states, the robot has more than 10 units of energy.

	Bottom	Middle	Top
Energy: 0	S1: Bottom/0	S12: Middle/0	S23: Top/0
Energy: 1	S2: Bottom/1	S13: Middle/1	S24: Top/1
Energy: 2	S3: Bottom/2	S14: Middle/2	S25: Top/2
Energy: 3	S4: Bottom/3	S15: Middle/3	S26: Top/3
Energy: 4	S5: Bottom/4	S16: Middle/4	S27: Top/4
Energy: 5	S6: Bottom/5	S17: Middle/5	S28: Top/5
Energy: 6	S7: Bottom/6	S18: Middle/6	S29: Top/6
Energy: 7	S8: Bottom/7	S19: Middle/7	S30: Top/7
Energy: 8	S9: Bottom/8	S20: Middle/8	S31: Top/8
Energy: 9	S10: Bottom/9	S21: Middle/9	S32: Top/9
Energy: greater and equal 10	S11: Bottom/10	S22: Middle/10	S33: Top/10

Set of actions: At each state, there are only two actions that the robot can make. The robot can either turns on the motor and do not turns on the motor. However, at S1, S12, and S23, the robot has zero energy, therefore, the action could only be turns off. Therefore the set of actions is A = { (S1, turns off), (S2, turns on, not turns on), (S3, turns on, not turns on)……, (S12, turns off), …. , (S23, turns off)…(S33, turns on , turns off) }

Transition Functions: The transition functions is shown below. The energy in the following table represents the current energy that the robot has where energy is greater or equal to 0. In addition, when the robot reaches the terminal states, the transition function will then stays on that terminal state with 100% possibility and a reward of zero.

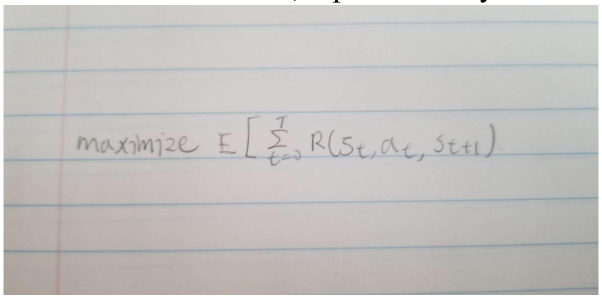
T(Top/ energy, turns on, Top/ energy + 3) = 0.7
T(Top/ energy, turns on, Middle/ energy) = 0.3
T(Top/ energy, turns off, Top/ energy + 4) = 0.5
T(Top/ energy, turns off, Middle/ energy + 1) = 0.5
T(Middle/ energy, turns on, Top/ energy + 3) = 0.4
T(Middle/ energy, turns on, Middle/ energy) = 0.4

T(Middle/ energy, turns on, Bottom/ energy -1) = 0.2
T(Middle/ energy, turns off, Middle/ energy + 1) = 0.5
T(Middle/ energy, turns off, Bottom/energy) = 0.5
T(Bottom/ energy, turns on, Bottom/energy - 1) = 0.3
T(Bottom/ energy, turns on, Middle/ energy) = 0.7
T(Bottom/ energy, turns off, Bottom/ energy) = 1

Reward Function: 1 for the terminal states and 0 for the rest of states.

Discount Factor: I set 1 as the discount factor. Since there are terminal conditions, therefore the value function will converge eventually.

Objective function: The following is the objective function. It means that we need to maximize the value of expected reward to go. The sigma ends when the robot reaches the terminal states , represented by T.



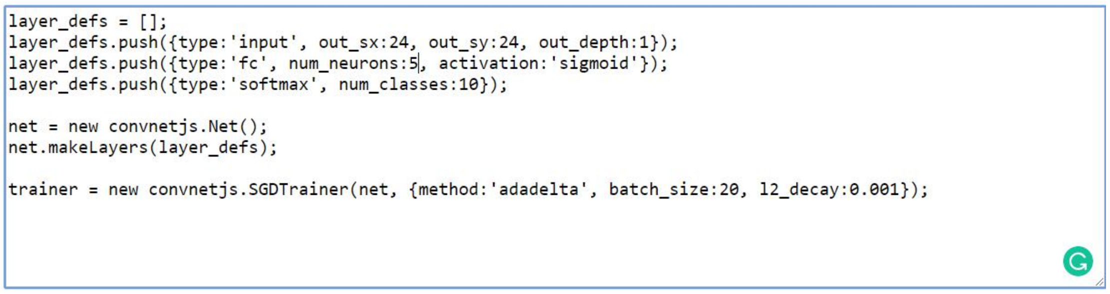
$$\text{maximize } E\left[\sum_{t=0}^{T} R(s_t, a_t, s_{t+1})\right.$$

Problem 2

The e parameter in e greedy policy indicates that the possibility of choosing something new or following the highest Q value. Therefore, if the e parameter is 0.1. That means there are 10 percent of chance that the robot will choose the action randomly and there are 90 percent of change that the robot will act following the highest Q value. Therefore, if the value is very small, the robot will have a big chance follow the highest Q value, and if the value is very big, then the robot will have a big change choose the next action randomly.

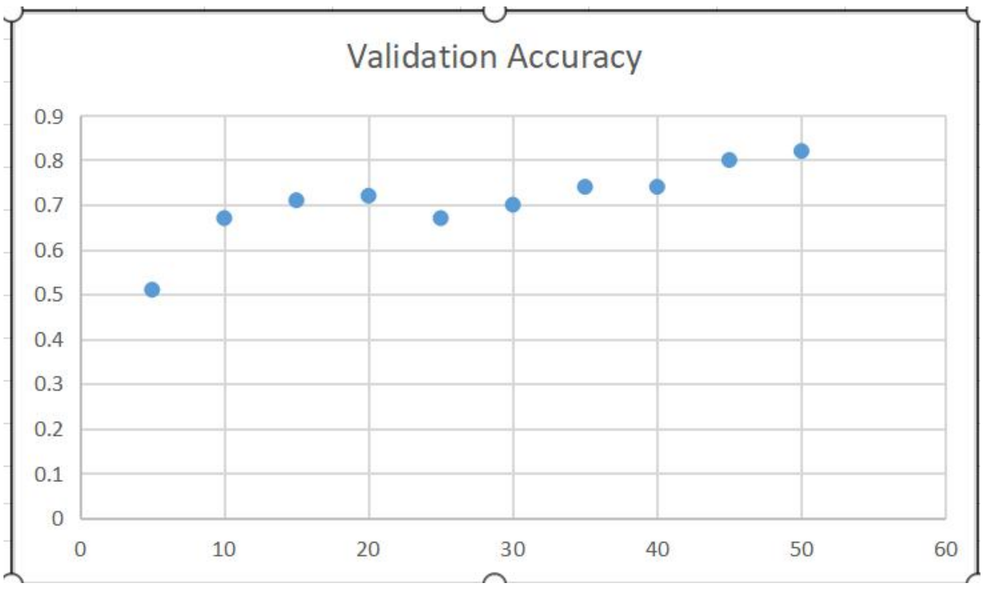
Problem 3

A: For this question, I used the given cod showing in the picture and the following shows the validation accuracy of a neutral network with one hidden layers, sigmoid activation and different numbers of neurons.



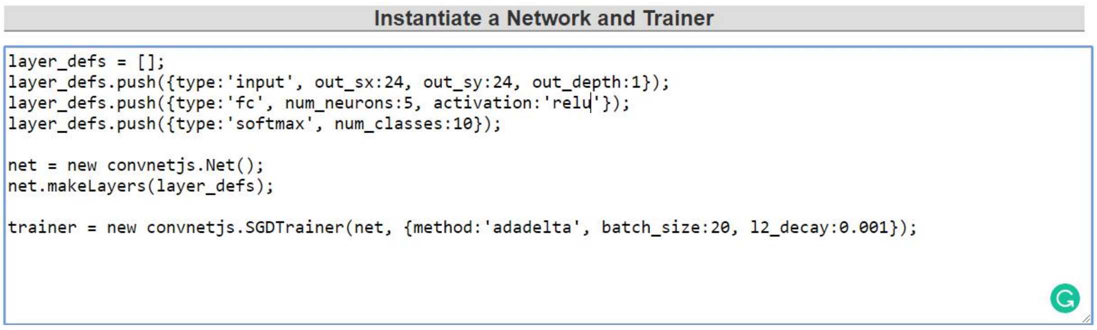
```
layer_defs = [];
layer_defs.push({type:'input', out_sx:24, out_sy:24, out_depth:1});
layer_defs.push({type:'fc', num_neurons:5, activation:'sigmoid'});
layer_defs.push({type:'softmax', num_classes:10});

net = new convnetjs.Net();
net.makeLayers(layer_defs);

trainer = new convnetjs.SGDTrainer(net, {method:'adadelta', batch_size:20, l2_decay:0.001});
```

Number of Neurons	Validation Accuracy
5	0.51
10	0.67
15	0.71
20	0.72
25	0.67
30	0.7
35	0.74
40	0.74
45	0.8
50	0.82



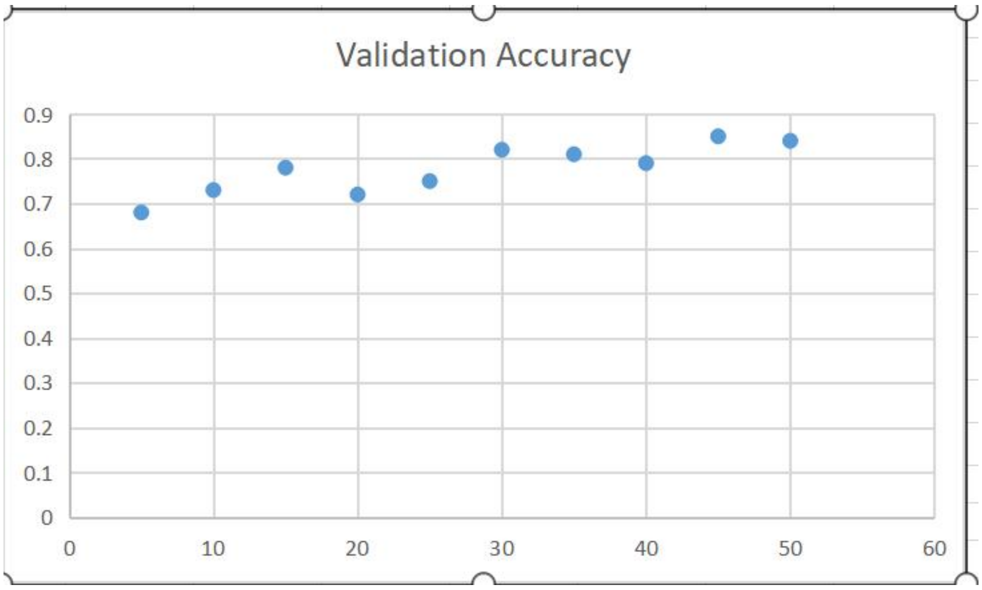
The is the plot of the validation accuracy table, the x-axis is the number of neurons and the y -axis is the validation accuracy.

B. For this question, I changed the activation function to relu, and the code is shown in below. The following table shows the validation accuracy of a neutral network with one hidden layers, relu activation and different numbers of neurons.



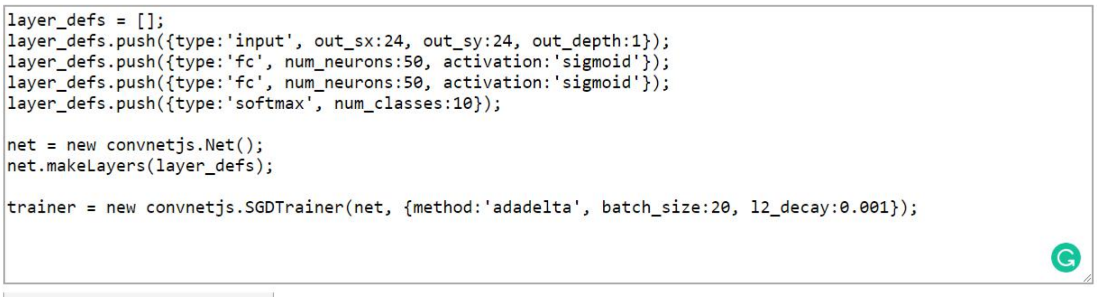
```
layer_defs = [];
layer_defs.push({type:'input', out_sx:24, out_sy:24, out_depth:1});
layer_defs.push({type:'fc', num_neurons:5, activation:'relu'});
layer_defs.push({type:'softmax', num_classes:10});

net = new convnetjs.Net();
net.makeLayers(layer_defs);

trainer = new convnetjs.SGDTrainer(net, {method:'adadelta', batch_size:20, l2_decay:0.001});
```

Number of Neurons	Validation Accuracy
5	0.68
10	0.73
15	0.78
20	0.72
25	0.75
30	0.82
35	0.81
40	0.79
45	0.85
50	0.84



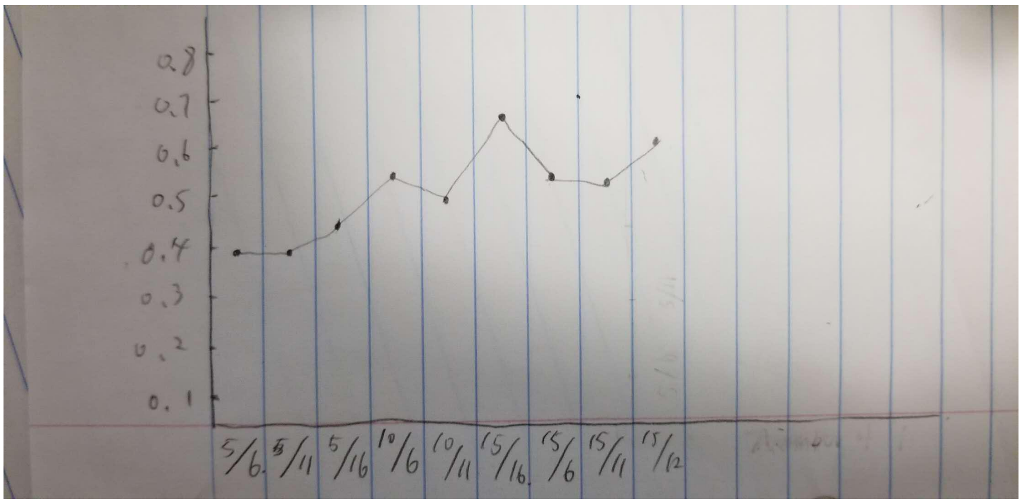
The is the plot of the validation accuracy table, the x-axis is the number of neurons and the y -axis is the validation accuracy.

C: For this question, I added one more layer and the code is shown below. The following table shows the validation accuracy of a neutral network with two hidden layers, sigmoid activation and different number of neurons.



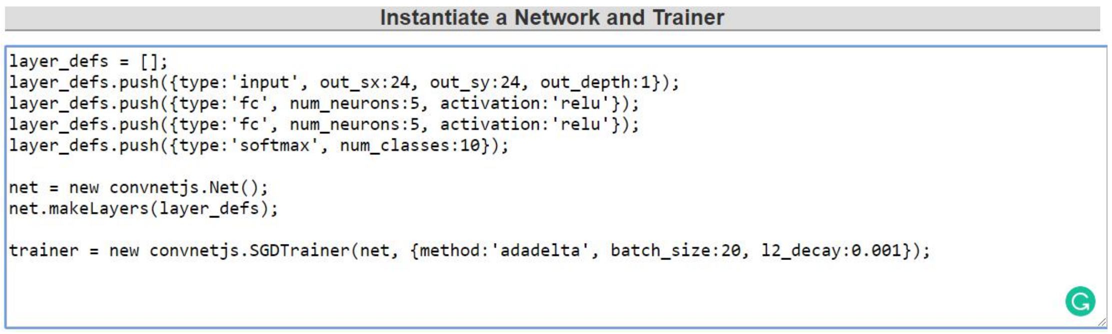
```
layer_defs = [];
layer_defs.push({type:'input', out_sx:24, out_sy:24, out_depth:1});
layer_defs.push({type:'fc', num_neurons:50, activation:'sigmoid'});
layer_defs.push({type:'fc', num_neurons:50, activation:'sigmoid'});
layer_defs.push({type:'softmax', num_classes:10});

net = new convnetjs.Net();
net.makeLayers(layer_defs);

trainer = new convnetjs.SGDTrainer(net, {method:'adadelta', batch_size:20, l2_decay:0.001});
```

Neurons of first layer	Neurons of second layer	Validation Accuracy
5	6	0.38
5	11	0.38
5	16	0.44
10	6	0.52
10	11	0.47
15	16	0.63
15	6	0.51
15	11	0.5
15	12	057



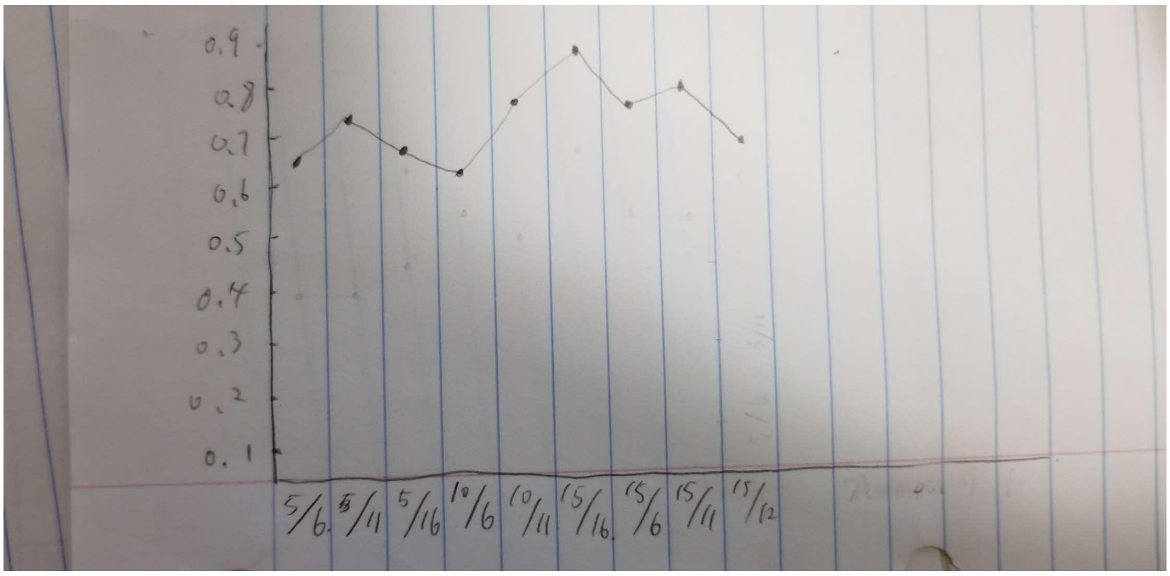
The above picture is the plot for the validation accuracy. The x-axis represents the number of neurons of both layer and the y-axis represents the validation accuracy.

D: For this question I modified the code in last question. I changed the activation function to relu which is shown in below. The following table shows the validation accuracy of a neutral network with two hidden layers, relu activation and different number of neurons.


Instantiate a Network and Trainer

```
layer_defs = [];
layer_defs.push({type:'input', out_sx:24, out_sy:24, out_depth:1});
layer_defs.push({type:'fc', num_neurons:5, activation:'relu'});
layer_defs.push({type:'fc', num_neurons:5, activation:'relu'});
layer_defs.push({type:'softmax', num_classes:10});

net = new convnetjs.Net();
net.makeLayers(layer_defs);

trainer = new convnetjs.SGDTrainer(net, {method:'adadelta', batch_size:20, l2_decay:0.001});
```

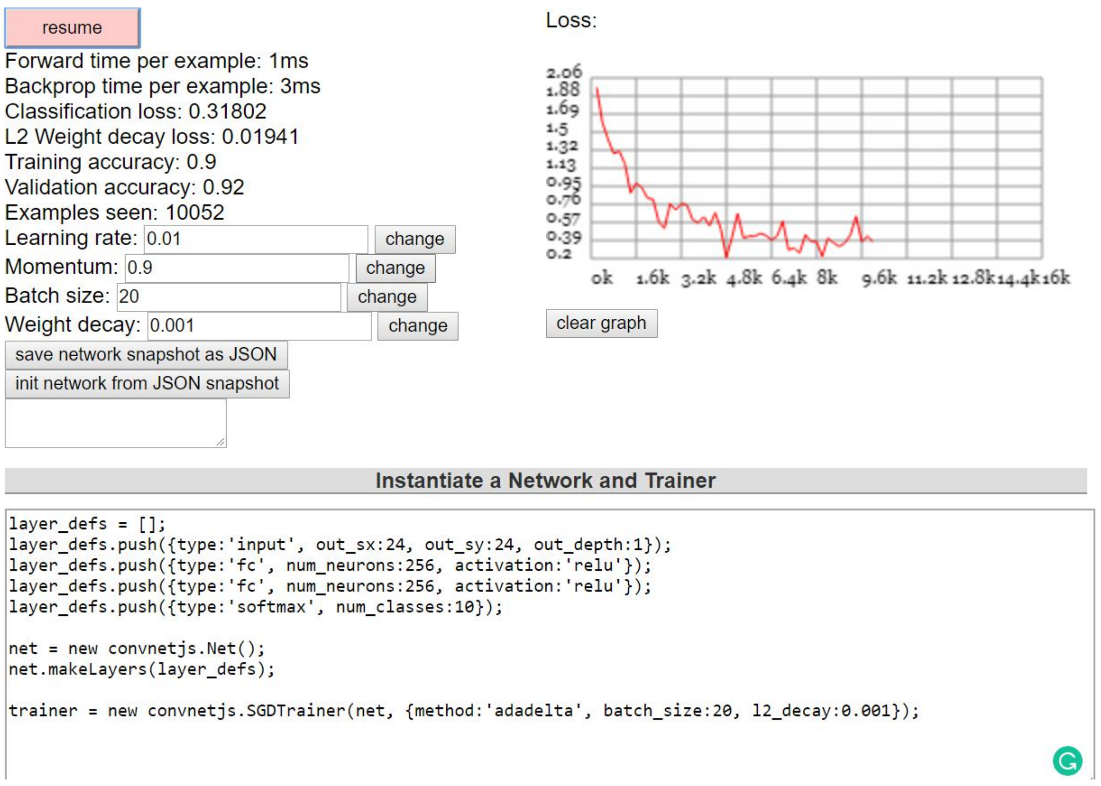
Neurons of first layer	Neurons of second layer	Validation Accuracy
5	6	0.65
5	11	0.72
5	16	0.66
10	6	0.61
10	11	0.74
15	16	0.87
15	6	0.71
15	11	0.75
15	12	0.65



The above picture is the plot for the validation accuracy. The x-axis represents the number of neurons of both layer and the y-axis represents the validation accuracy.

E:

The highest validation accuracy I got is 0.9. This is a fully connected neutral network with two hidden layers and 256 neurons for each layer. The activation function I used is relu.



Forward time per example: 1ms
Backprop time per example: 3ms
Classification loss: 0.31802
L2 Weight decay loss: 0.01941
Training accuracy: 0.9
Validation accuracy: 0.92
Examples seen: 10052
Learning rate: 0.01
Momentum: 0.9
Batch size: 20
Weight decay: 0.001

Instantiate a Network and Trainer

```
layer_defs = [];
layer_defs.push({type:'input', out_sx:24, out_sy:24, out_depth:1});
layer_defs.push({type:'fc', num_neurons:256, activation:'relu'});
layer_defs.push({type:'fc', num_neurons:256, activation:'relu'});
layer_defs.push({type:'softmax', num_classes:10});

net = new convnetjs.Net();
net.makeLayers(layer_defs);

trainer = new convnetjs.SGDTrainer(net, {method:'adadelta', batch_size:20, l2_decay:0.001});
```