Final Take-home Exam Solutions

Shuchang Liu (sl1471)

Question 1 – Localization

a) The probability at G?

Solution:

There are in total 695 white cells. With no prior knowledge, the probability of being in any one of them is simply 1/695. Thus, the probability of being at G is 1/695.

b) Give a (short) sequence of moves that will with probability at least 1/2 end at G?

Solution:

I transform the problem as this, for each of the 695 positions, I assign 1 person. Then the question is now asking for a sequence such that applying all the actions in the sequence, more than half (, namely, 348) of the population are moved to target G. This can be further divided into two phases:

1. Apply an action sequence Seq_1 that gathers at least 348 people in one position T.
2. Apply another sequence Seq_2 that move those people in T to target G.

The combination of $Seq_1 + Seq_2$ is then a solution to the problem.

The solution is generated through my code in *localize.py*. One can repeat my process by calling in command line:

- python localize.py

Then move all the people by pressing arrow keys or WSAD keys, the corresponding action sequence is output in the terminal. The sequence I discovered is saved in *results/q1b_sequence.txt*, and the length of the sequence is 110. The corresponding result is shown as Figure 2. As you can see, there are 387 people moved to G. Since 387/695 > 1/2, the solution condition is satisfied.

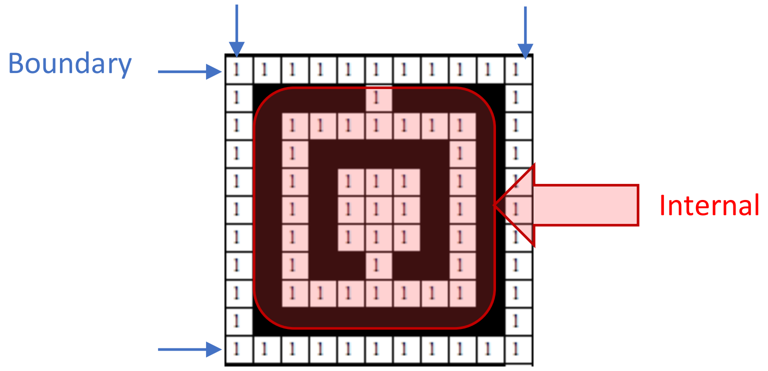


Figure 1 - Outer most boundary positions of a subgraph is indicated by blue and internal positions are covered in red.

The intuition behind this sequence I apply is that, for each of the subgraph shown as Figure 1, applying Right and Down for multiple times will gather those people on the boundary (the outer most positions) to a corner in the graph. Then the obviously next step is to move this group of people from the corner to the target position G. Since each subgraph has 40 of those people, combining 9 such subgraphs will gather 9*40=360 people which is more than half of the population.

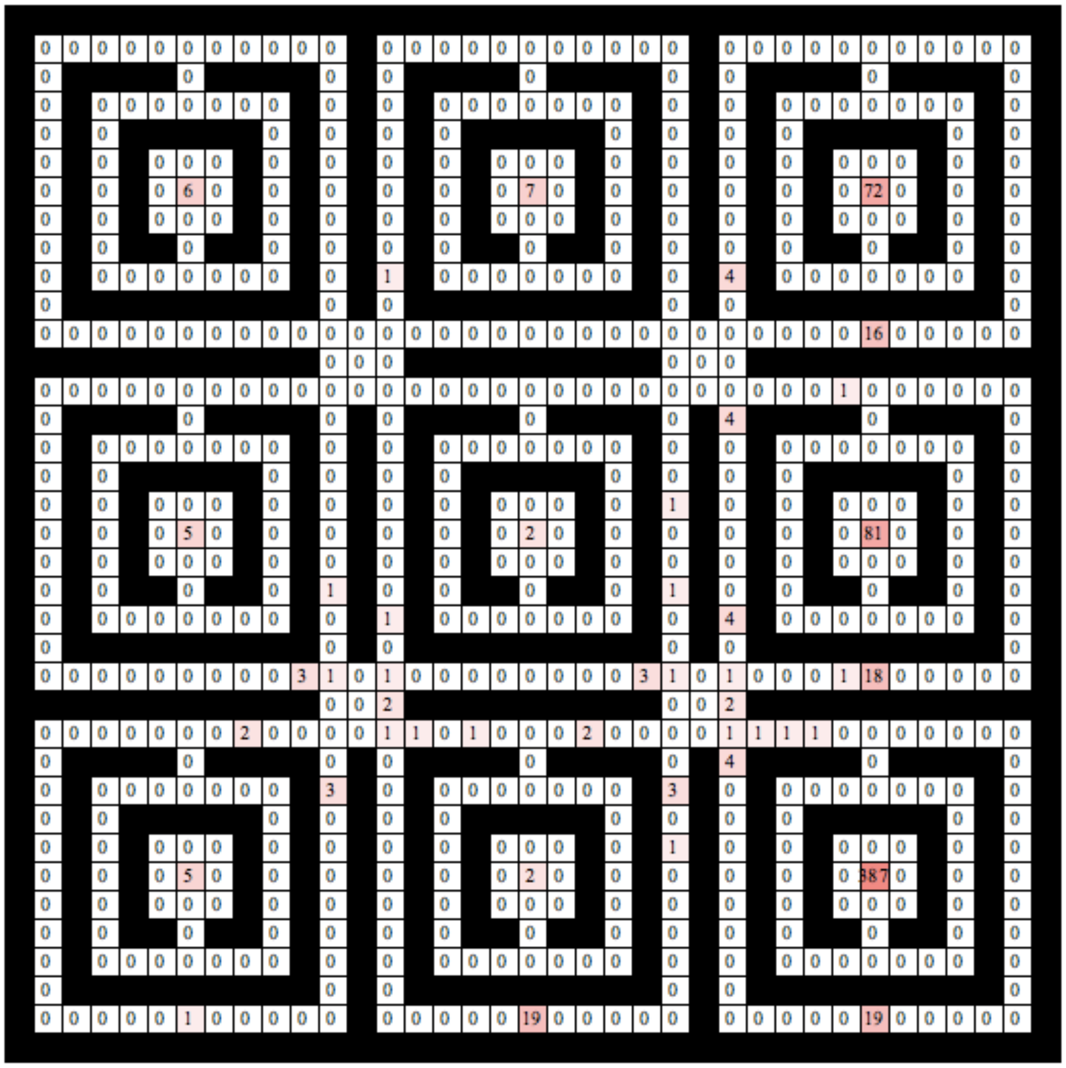


Figure 2 - The result for question 1.b

c) Find the shortest possible sequence of moves that guarantees you will end up at G regardless of where you start.

Solution:

The result should be exactly Figure 3 where everyone is at G. Currently the shortest sequence I observe is a sequence of length 171 which is saved in file results/p1c_sequence.py. In terms of the intuition, the sequence first empty the internal (as shown in Figure 1) of each of the 9 subgraphs, then everyone is on the boundary. Then it gathers all of them downward and rightward until all 695 people are standing in the same cell. Then move them to the target based on the shortest path from that location to G.

However, in implementation, it is better to have a systematic way. On possible approach is the greedy best approach. In my guess, the solution algorithm may look at all the possible 4 actions each step, and simply takes the one that generates the estimated least cost or best rewards.

Here are some possible choices for estimation of cost/rewards:

1. The sum of shortest path: each person no matter where it is, it knows its shortest path to G. Simply collect this report from everyone and take the sum.
2. The number of people gathered: compare the number of groups or the average size of groups before and after the action.
3. The average Manhattan distance to G.

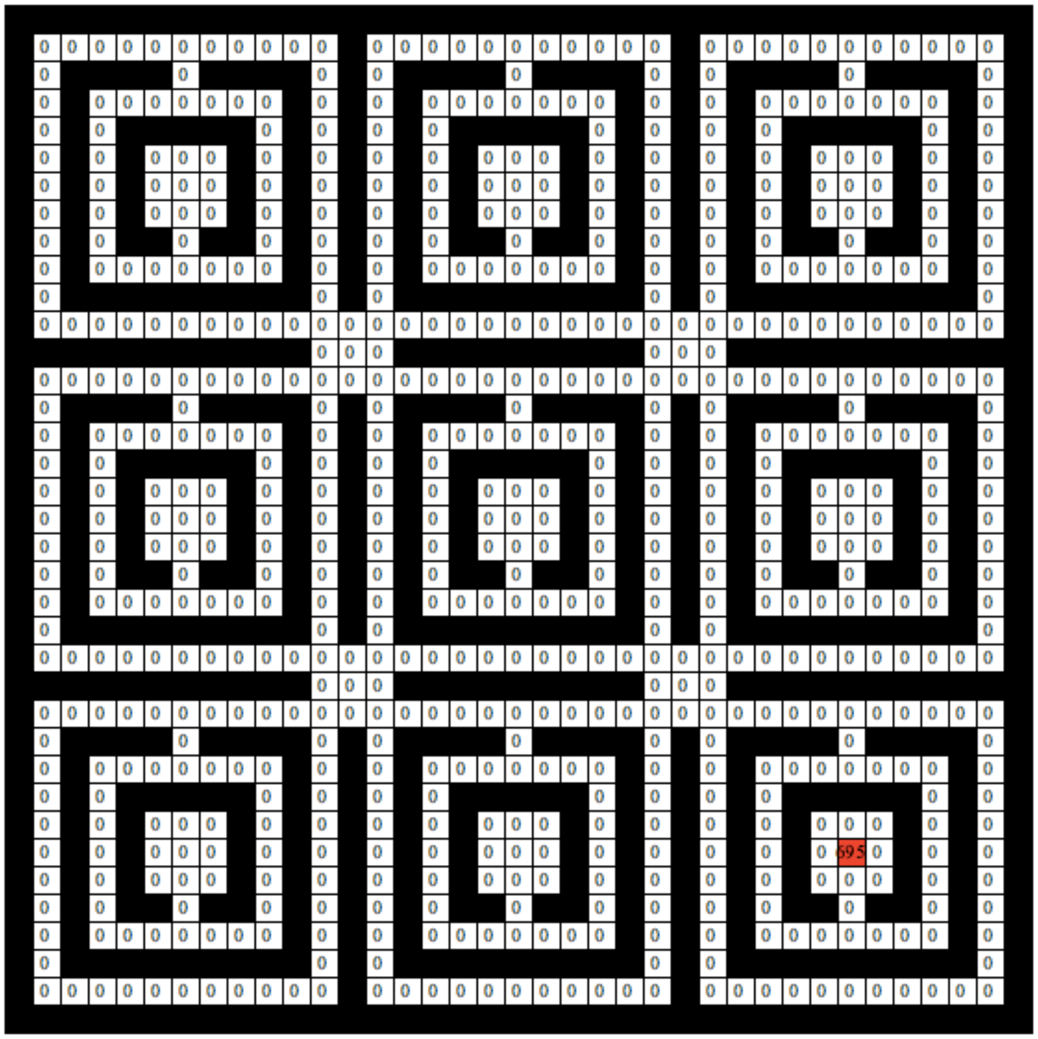


Figure 3 - Result of applying sequence of problem 1.c), everyone is at G.

d) Part 1) 5 blocked cells, move LEFT, 5 blocked cells, move LEFT, 5 blocked cells. Indicate, for each cell the final probability of you being in that cell.

<u>Solution:</u>

In my implementation, the result is visualized with color and numbers indicating the probability. In terms of the color, the lighter the red, the lower the probability. In terms of the number, it shows how many people are there in that position, if the position violates the observation, then the number is set to 0.

The result of this question is shown as Figure 4 where you can see that all the possible cells have 1 person. And the total number of possible cells are 128, then each possible cell has probability 1/128.

To repeat this experiment: i) click "reset simulator" in control panel. ii) enter 5 for the number of surrounding blocks. iii) click "filter" in control panel. iv) focus on the maze window and press either "LEFT" or "A" key to move everyone left. v) click "filter" again. vi) focus on the maze window and press either "LEFT" or "A" key again. vii) click "filter" and see the result.

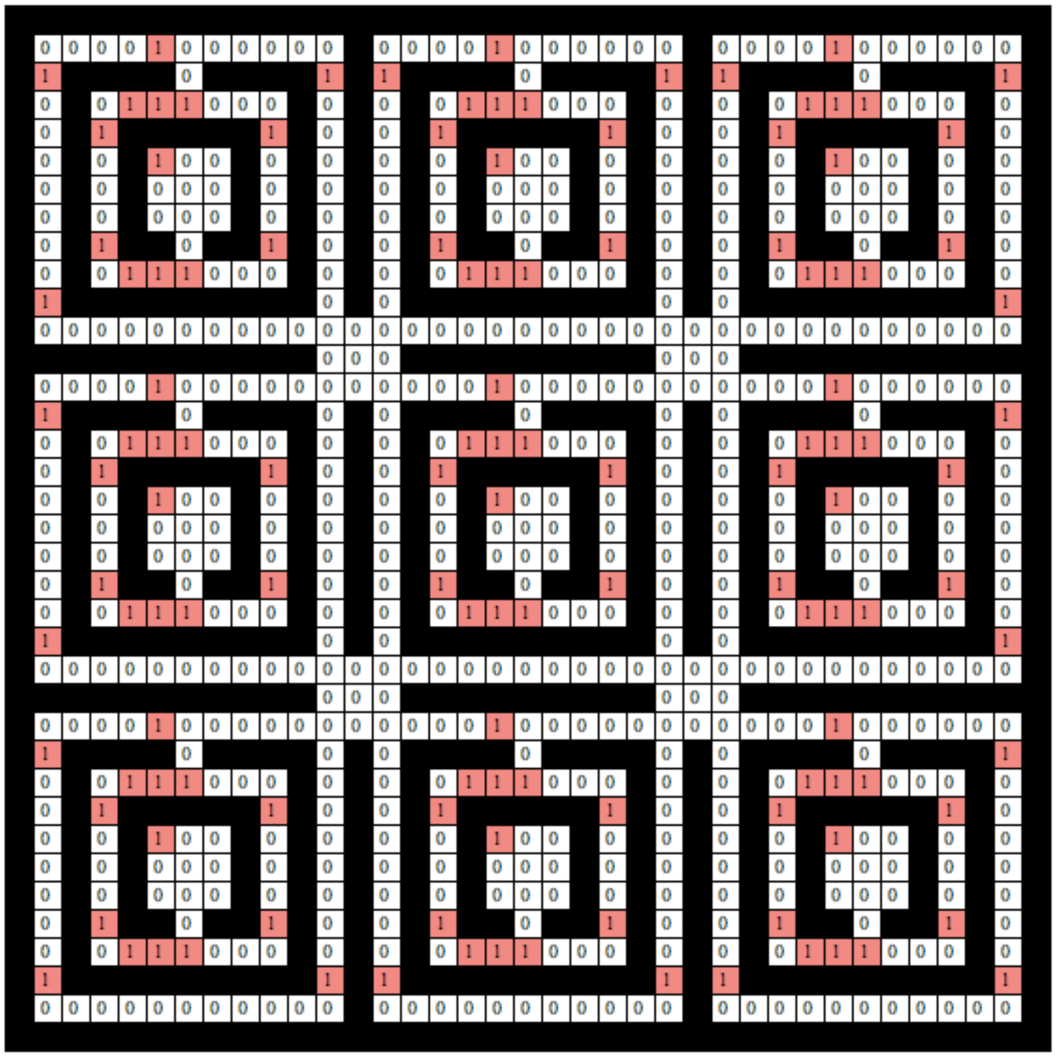


Figure 4 - Result of possible cells and their probabilities for question 1.d.1)

Part 2) Write an algorithm to take a sequence of observations and a sequence of actions and returns the cell you are most likely to be in.

Solution:

Using my implementation, one can specify the sequence of observations in the file *observations.txt* and the sequence of action in the file *actions.txt*. Then go back to the control panel and click "load and test filter", the program will automatically apply the filtering based on your specification in these two files, and returns the result in the maze window.

For example, along with this report, in the code folder, the observations and action are generated by the path shown in Figure 5 - The actual movement. The corresponding sequence of actions:

[Right, Right, Right, Right, Right, Down, Down, Right, Right, Right]

The corresponding sequence of observation: [3, 4, 6, 6, 5, 5, 2, 5, 5, 5, 6]

All the four resulting possible positions are shown in the bottom image of Figure 6.

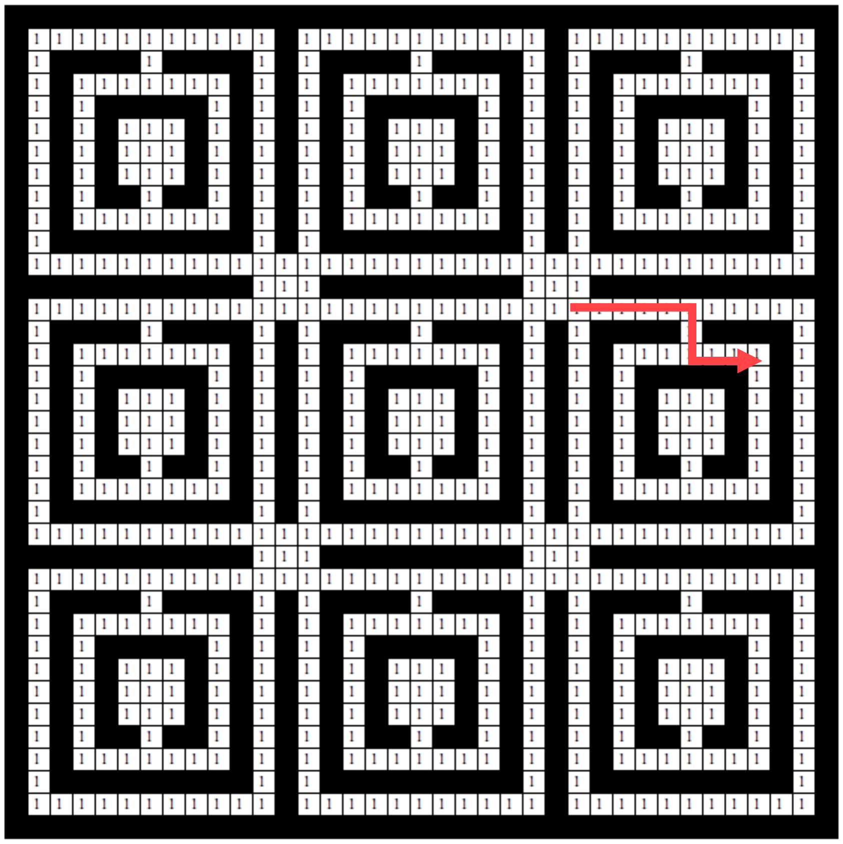


Figure 5 - The actual movement

Question 2

Possible states:

New, Used$_1$, Used$_2$, Used$_3$, Used$_4$, Used$_5$, Used$_6$, Used$_7$, Used$_8$, Dead

Rules:

1. **In states except $\boldsymbol{Dead}$, you can USE. In states not New, you can REPLACE.**
2. **REPLACE sends the machine state to $\boldsymbol{New}$ with probability 1 at cost 250.**
3. **In state New, USE gives a reward of 100, and transitions to state $\boldsymbol{Used_1}$ with probability 1. For $\boldsymbol{i = 1, \dots, 7}$, in state $\boldsymbol{Used_i}$, USE yields a reward of $\boldsymbol{100 - 10 * i}$ and with probability $\boldsymbol{0.1 * i}$ the machine transitions to state $\boldsymbol{Used_{i+1}}$, otherwise stay. In state $\boldsymbol{Used_8}$, USE yield a reward of 20, and transitions to state $\boldsymbol{Dead}$ with probability 0.8, otherwise stay.**
4. **The only action available in state Dead is REPLACE**
5. **At every time step, all future rewards (and costs) are discounted at a factor of $\boldsymbol{\beta = 0.9}$.**

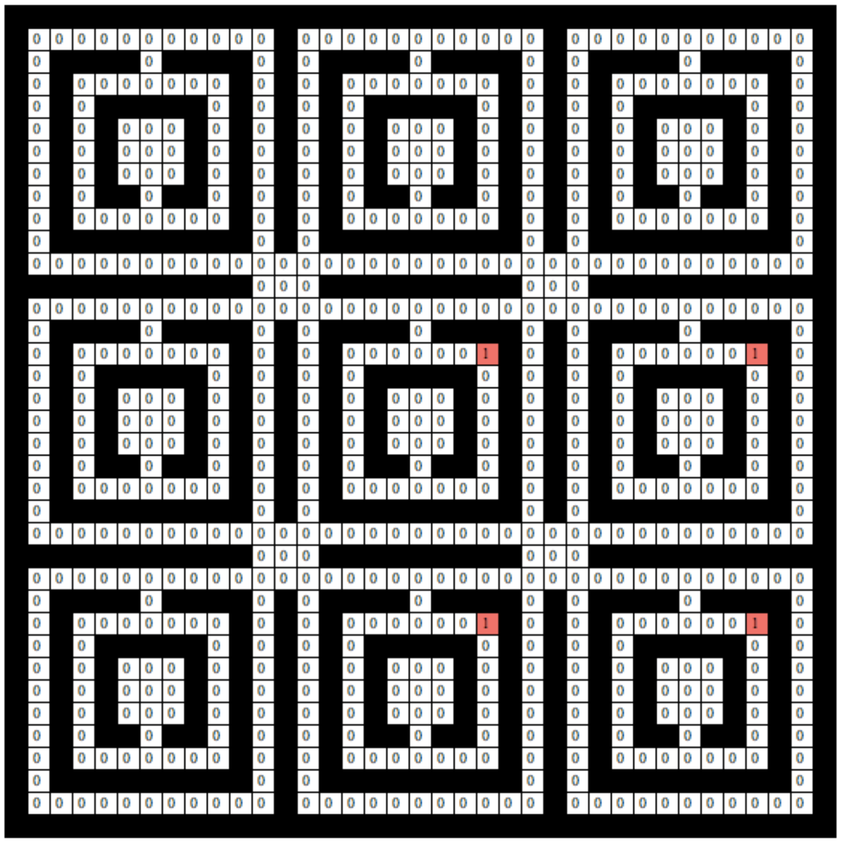


Figure 6 - The target path and the result of all possible ending positions

Question 2 Problems:

a) **For each of the 10 states, what is the optimal utility (long term expected discounted value available in that state)?**

<u>Solutions:</u>

The corresponding Markov Decision Process is shown in Figure 7.

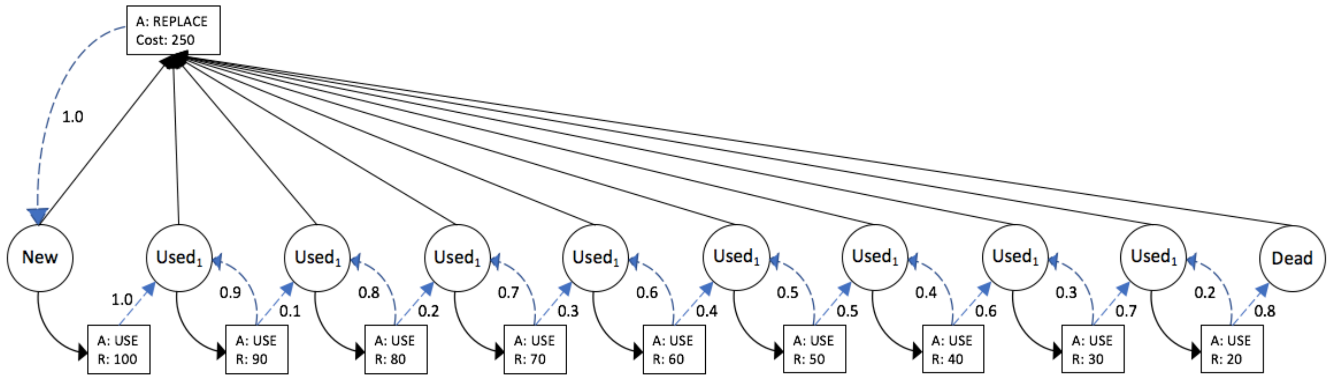


Figure 7 - The MDP correspond to problem 2

I used Value Iteration to converge to the optimal utilities:

```
Optimal utilities:
State(New): 800.531609876
State(Used1 - Used8):[778.3684554178556, 643.2222947711231, 556.1235696440165, 502.836002845536,
475.84600363598634, 470.4784488884467, 470.4784488884467, 470.4784488884467]
State(Dead): 470.478448888
```

In my implementation, I used the following update function as the variation of bellman equation:

$$U_{i+1}(s) = \max_{\alpha}\left(R(s,\alpha) + \beta \sum\nolimits_{s'} T(s,\alpha,s')U(s')\right)$$

To repeat this experiment, in command line type:

- python markov.py

Then type 1 and press Enter to select the first experiment. The initial utilities are set to all zeros, and the tolerance factor is $\varepsilon = 10^{-10}$. The number of iterations used to converge is 280.

b) What is the optimal policy that gives you this optimal utility. (best action in each state)

Solution:

This correspond to the same solution of part a), the optimal policies are:

```
Optimal policies:
Policy(New): USE
Policy(Used1 - Used8):['USE', 'USE', 'USE', 'USE', 'USE', 'REPLACE', 'REPLACE', 'REPLACE']
Policy(Dead): REPLACE
```

To repeat this experiment, again, type:

- python markov.py

Then type 1 and press Enter to select the first experiment.

c) New machines are expensive. Open a store selling used machines. Equal chance of being in $\boldsymbol{Used_1}$ and $\boldsymbol{Used_2}$. Free: no one would buy New machines but everyone comes. Sell at 250, no one would buy them. What is the highest price you could sell your used machines while ensuring that users would buy them?

Solution:

A rational user will understand the results generated from part a) and part b) so the reasonable price for them is simply:

$$price = \begin{cases} 250 * \dfrac{U_*(Used1)}{U_*(New)}, & s = Used1 \\ 250 * \dfrac{U_*(Used2)}{U_*(New)}, & s = Used2 \end{cases}$$

The overall price is then:

$$0.5 * 250 * \frac{U_*(Used1)}{U_*(New)} + 0.5 * 250 * \frac{U_*(Used1)}{U_*(New)} \approx 192.6$$

d) **For different values of $\boldsymbol{\beta}$, the utility or value of being in certain states will change. However, the optimal policy may not. Compare the optimal policy for $\boldsymbol{\beta = 0.1, 0.3, 0.5, 0.7, 0.9, 0.99}$ etc. Is there a policy that is optimal for all sufficiently large $\boldsymbol{\beta}$? What do you make of it?**

Solution:

I tested $\beta = 0.1, \beta = 0.3, \beta = 0.5$, and $\beta_i = 1 - 0.3^i$ for $i = [1, 2, 3, \dots, 8]$. Their corresponding results are given in Table 1 - Selected discount and its corresponding optimal policies.. As you can see, increasing the β, will generate a set of increased utilities, and the number of steps needed to convergence is also increasing. However, the optimal policy seems to stop changing after β_3, which indicate that, there exist a certain threshold $\beta_* \in (\beta_2, \beta_3]$, for any $\beta \geq \beta_*$, the optimal policy is fixed as:

[U, U, U, U, R, R, R, R, R, R]

To obtain the threshold, one can use bisection method to find an ε-approximation in O($\log_2 \frac{\beta_3 - \beta_2}{\varepsilon}$) steps.

Table 1 - Selected discount and its corresponding optimal policies.

Discount β_i	Optimal Policy (R = REPLACE, U = USE)	Optimal Utilities (All numbers rounded to integers)	#Step
0.1	[U, U, U, U, U, U, U, U, U, R]	[110, 100, 89, 77, 66, 55, 44, 31, 1, -239]	14
0.3	[U, U, U, U, U, U, U, U, U, R]	[138, 128, 113, 98, 83, 68, 51, 26, -32, -208]	26
0.5	[U, U, U, U, U, U, U, U, U, R]	[189, 178, 156, 133, 110, 85, 56, 16, -47, -156]	44
$1 - 0.3$	[U, U, U, U, U, U, U, U, U, R]	[303, 290, 247, 203, 161, 118, 22, 37, -1, -38]	84
$1 - 0.3^2$	[U, U, U, U, U, U, R, R, R, R]	[878, 855, 712, 624, 573, 550, 549, 549, 549, 549]	314
$1 - 0.3^3$	[U, U, U, U, R, R, R, R, R, R]	[2654, 2625, 2429, 2355, 2333, 2333, 2333, 2333, 2333, 2333]	1079
$1 - 0.3^4$	[U, U, U, U, R, R, R, R, R, R]	[8540, 8509, 8297, 8232, 8221, 8221, 8221, 8221, 8221, 8221]	3626
$1 - 0.3^5$	[U, U, U, U, R, R, R, R, R, R]	[28140, 28109, 27891, 27830, 27822, 27822, 27822, 27822, 27822, 27822]	12132
$1 - 0.3^6$	[U, U, U, U, R, R, R, R, R, R]	[93467, 93436, 93217, 93156, 93149, 93149, 93149, 93149, 93149, 93149]	39614
$1 - 0.3^7$	[U, U, U, U, R, R, R, R, R, R]	[311222, 311190, 310971, 310911, 310904, 310904, 310904, 310904, 310904, 310904]	127200
$1 - 0.3^8$	[U, U, U, U, R, R, R, R, R, R]	[1037071, 1037039, 1036820, 1036760, 1036753, 1036753, 1036753, 1036753, 1036753, 1036753]	414470

Question 3 – Classification

a) **Construct a model to classify class A or B, and train it. What does it predict for each of the unlabeled images. Give details of the model, its training, and the final result. Do the predictions make sense?**

Solution:

The method used here is decision tree. The code is available in *classification.py*.

During training / construction, feature selection is done according to information gain. It terminates until every leaf node is pure. The feature of each record is a vector of 25 binary variables correspond to a reshape of the original image. The label is binary too, 1 represents class B, 0 represents class A.

During prediction, my code provides the flexibility of setting the threshold T ($0.5 < T \leq 1$, the certainty to of a prediction). Given a feature X, as it goes through the branches of the model tree, it will immediately return a guess if a class label, at the node, has higher probability than the threshold.

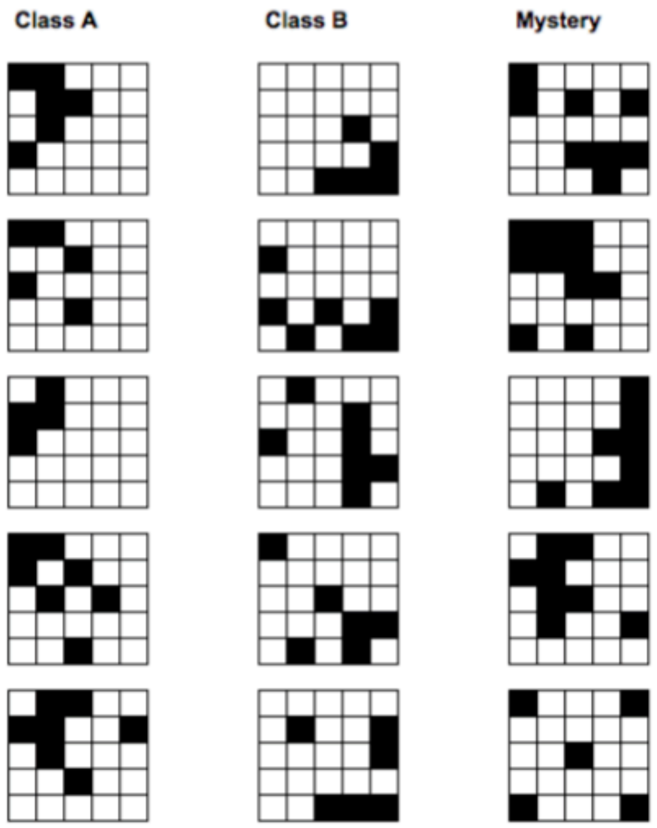


Figure 8 - Datasets of question 3

The resulting model is shown as Figure 9.

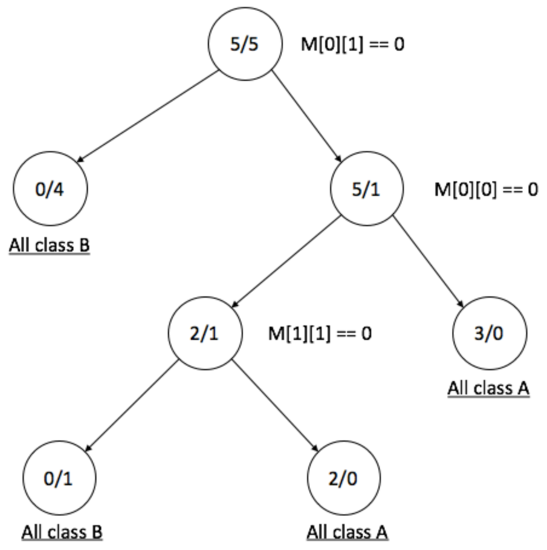
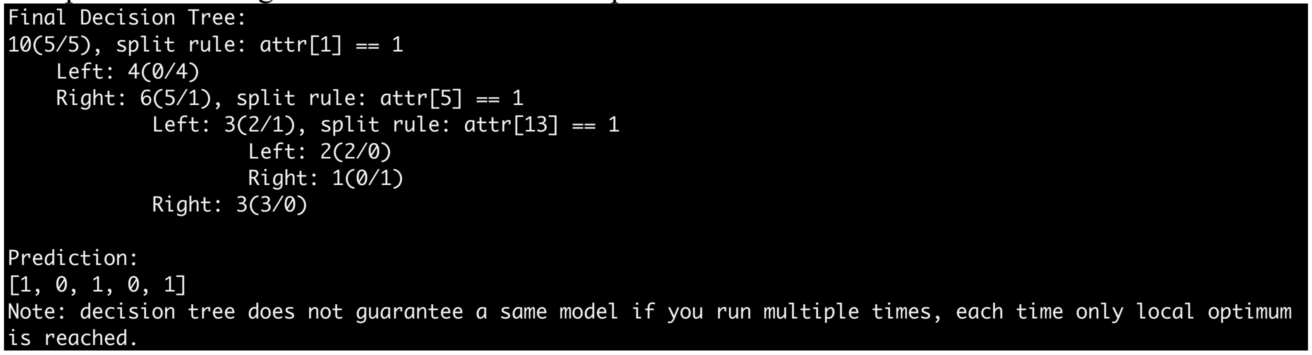


Figure 9 - Resulting decision tree of question 3

The prediction along with the tree structure output are:



```
Final Decision Tree:
10(5/5), split rule: attr[1] == 1
    Left: 4(0/4)
    Right: 6(5/1), split rule: attr[5] == 1
            Left: 3(2/1), split rule: attr[13] == 1
                    Left: 2(2/0)
                    Right: 1(0/1)
            Right: 3(3/0)

Prediction:
[1, 0, 1, 0, 1]
Note: decision tree does not guarantee a same model if you run multiple times, each time only local optimum
is reached.
```

I also tested for multiple times to approximate the performance of a random forest, there are multiple different results existed:

[1,0,1,0,1], [1,0,1,1,0], [1,0,1,0,0], [1,0,1,1,1]

But based on the majority vote, it is more likely to have:

[1,0,1,0,1]

The results are reasonable, it distinguishes based on the structure of the top left corner of the image. While class A has more characteristics in this area, class B performs on other regions.

To repeat the experiment for this part, type in command line:

- python classification.py

Then press 1 and Enter to start experiment 1

b) The data provided is quite small, and overfitting is a serious risk. What steps can you take to avoid it?

Solution:

Overfitting happens when the model is too complex for the given data. The solution to overfitting, in general, are listed below:

1. Include more data in training
2. Use regularization on model complexity / parameters
3. Model selection based on evaluation or risk management.

Clearly, evaluation based model selection is preferred. The reason for this is the fact that there is no more data available, and that there the model parameters of decision tree method is hard to regularize. One can chose simple Hold-out method or 5-fold cross-validation that generates 5 models, and the result is then given by the majority vote over all predictions. No matter which evaluation you use, it should be tested for multiple rounds and the model with the lowest validation error will be chosen.

c) Construct and train a second type of model. Specify its details. How do its prediction compare to the first model?

Solution:

Now I want to compare the perceptron model to the decision tree method. Intuitively, perceptron works like regression, but if one sets a cutting threshold during prediction, it will perform like one.

In my implementation, I constructed a single hidden layer perceptron. The network structure is given in Figure 10. While the output node uses sigmoid activation function, the hidden layer uses the RELU method. Interestingly, the input data is always 0 or 1, this means that the RELU will perform like linear model. My construction take advantage of the robustness of RELU and the bounding effect of sigmoid. The initial weights are generated randomly between 0 and 1 to eliminate symmetric updating.

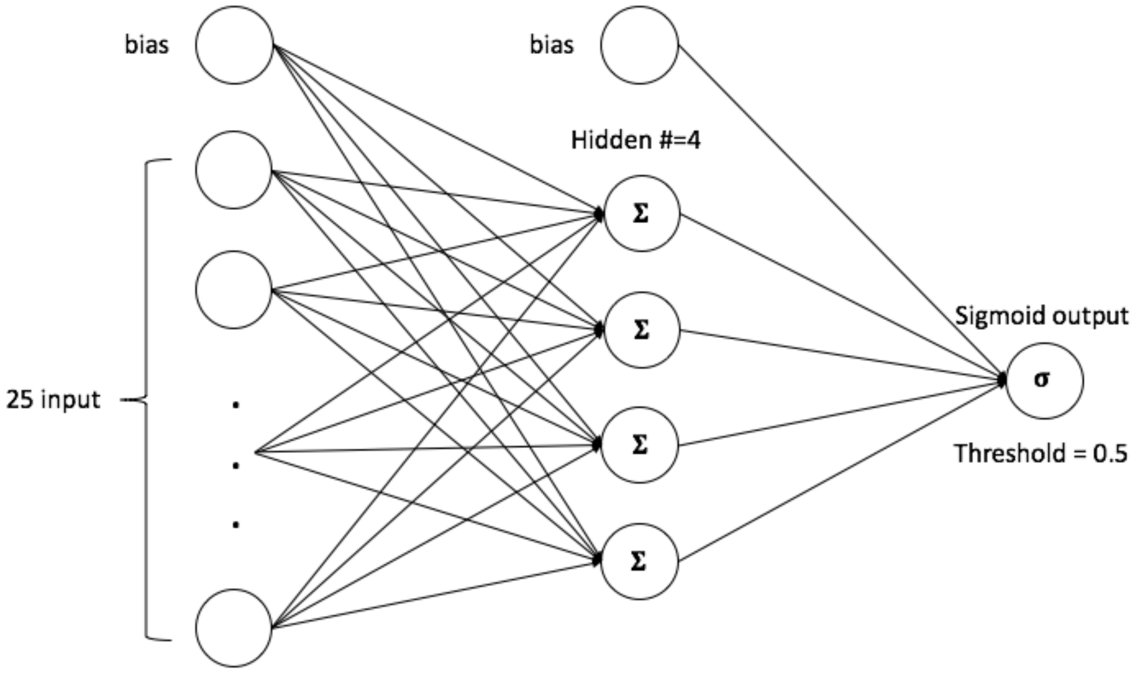
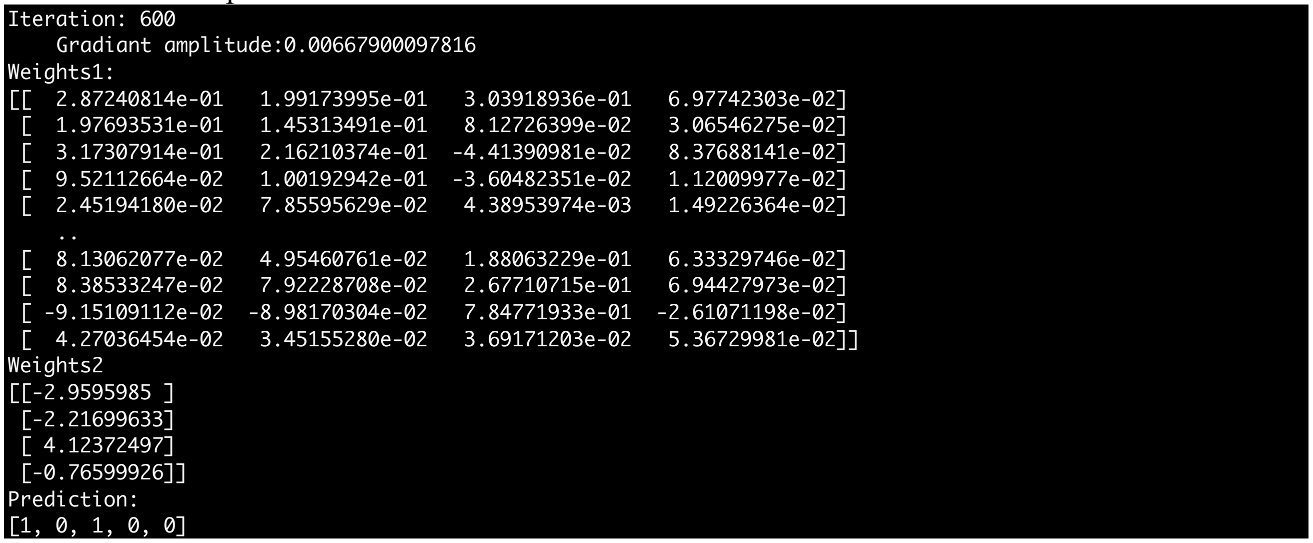


Figure 10 - Neural network structure in the implementation for question 3.c)

The result of the prediction is shown below:



```
Iteration: 600
    Gradiant amplitude:0.00667900097816
Weights1:
[[  2.87240814e-01   1.99173995e-01   3.03918936e-01   6.97742303e-02]
 [  1.97693531e-01   1.45313491e-01   8.12726399e-02   3.06546275e-02]
 [  3.17307914e-01   2.16210374e-01  -4.41390981e-02   8.37688141e-02]
 [  9.52112664e-02   1.00192942e-01  -3.60482351e-02   1.12009977e-02]
 [  2.45194180e-02   7.85595629e-02   4.38953974e-03   1.49226364e-02]
    ..
 [  8.13062077e-02   4.95460761e-02   1.88063229e-01   6.33329746e-02]
 [  8.38533247e-02   7.92228708e-02   2.67710715e-01   6.94427973e-02]
 [ -9.15109112e-02  -8.98170304e-02   7.84771933e-01  -2.61071198e-02]
 [  4.27036454e-02   3.45155280e-02   3.69171203e-02   5.36729981e-02]]
Weights2
[[-2.9595985 ]
 [-2.21699633]
 [ 4.12372497]
 [-0.76599926]]
Prediction:
[1, 0, 1, 0, 0]
```

This result shows the most frequent output when the number of node in the hidden layer is set to 6. The most frequent output for each of the hidden layer setting is given in Table 2.

Table 2 - The number of hidden layer and the corresponding output

# nodes in hidden layer	Output
2	[1,1,1,1,1]
3	[1,0,1,0,1]
4	[1,0,1,0,1]
8	[1,0,1,0,1]
16	[1,0,1,0,1]
32	[1,0,1,0,1]
64	[1,0,1,0,1]

As you can see, for the number of nodes no smaller than 3, the model is complex enough to generate the same result of decision tree. Compare to decision trees, this perceptron implementation is much more stable and robust. However, one must consider the well-known disadvantage of perceptron model, which is its trouble in interpretability.

To repeat the experiment for this part, type in command line:

- python classification.py

Then press 3 and Enter to start experiment 3

Question 4

Pick one of the following. Identify some of the properties and abilities it would need. Identify at least five issues that would be faced in trying to implement or build such a thing, and possible algorithmic solutions for each based on the material covered in class. Credit will be given for thoroughness and detail.

1) **ChefBot**
2) **ArtForgerBot**
3) **CharadesBot**
4) **SheepdogBot**
5) **PharmacistBot**
6) **SearchAndRescueBot**
7) **ArtForgeryDetectorBot**
8) **FashionTrendsPredictorBot**
9) **DrugTreatmentDesignerBot**
10) **MovieGenreCategorizationBot**
11) **HumanConsciousnessBot**

<u>Solution:</u>

HumanConsciousnessBot

The problem intuitive is an artificial intelligence that can determine whether a human is conscious or not. Consciousness in general describe the state or quality of awareness, or, being aware of an external object or something within oneself. The question itself has been asked for as long as there have been humans and there is a wide spread, if less than universal, consensus that an adequate account of mind requires a clear understanding of it and its place in nature [1]. In this brand domain, there are several types or notions of consciousness:

1. Sentience: capable of sensing and responding to its world.
2. Wakefulness: conscious only if it were awake and normally alert.
3. Self-consciousness: not only aware but also aware that they are aware.

And clearly the second notion is the most related notion to our question. However, determining if a human is conscious or not is still an extremely diverse and huge problem, since the number of existing tests and well-defined models has been sufficiently large. For example, projects like NASA uses trackers to monitor the mental states of astronauts, hospitals use medical sensors and devices to detect the wellness of a patient's brain. In artificial intelligence, it is often interested to determine if a person is asleep or awake based on simple streaming data of from some body sensor. One typical example is the sleep detection algorithm used in apple watch. Now let's consider this specific problem.

Problem Definition: Suppose given a wearable device that monitors one of your physical sign like heart rate, I am asked to program an AI that tells if the person is awake or asleep in real time.

Initial Proposal: The basic idea is to consider this as a binary classification problem that takes the input X and the label $y \in \{awake, asleep\}$. X consists of M examples of N consecutive physical signal collected from the sensor, for example a typical ECG [2] there is a clear difference between a sleeping sequence and a waking sequence. The solution first record a length-N subsequence of the stream as features, then uses typical classifiers to train the model or predict the label. Possible choices for classification model are random forests, SVM, naïve Bayes, and neural networks.

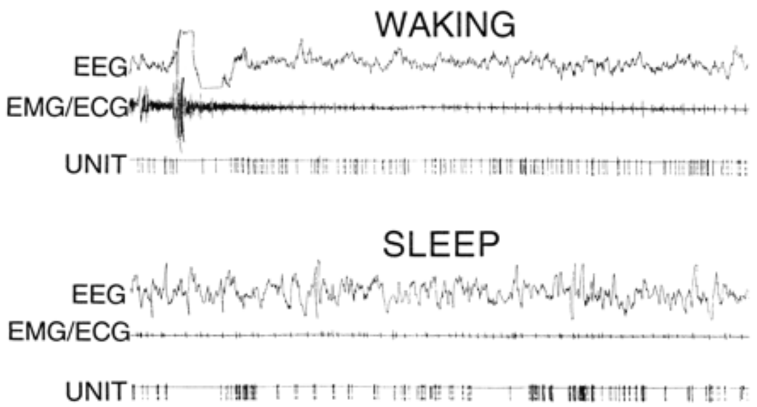


Figure 11 - Different EEG/ECG patterns between waking and sleep [https://www.semel.ucla.edu/sites/all/files/sleep-research/3500/J_ Neurosci_ -- Siegel et al_ 16 (10) 3500 Figure 3a.htm]

Possible issues: Here I listed some of the issues that may occur during implementation of those classifiers:

1. The feature space is usually large: for a very small sequence, wake and sleep patterns are ambiguous and hard to characterize. Thus, the window size is usually very large. To deal with large feature space, one can try feature selection or feature reduction. This is especially useful when choosing a fast algorithm is impossible. While feature selection uses correlation analysis to get rid of irrelevant or redundant features, feature reduction methods like PCA will projects the whole input data to a smaller space without much information loss.

2. The physical signals are more likely to be continuous values rather than categorical values that suits well with random forest. Thus, training a random forest (decision trees) or even a single decision tree might not be promising.
3. The randomness of the data may cause a huge problem. This randomness not only comes from the noise from the physical world (background noise, device malfunction, and weather condition), but also from the nature of the data source. In streaming data, pattern recognition algorithms have to consider the characteristics brought by the nature of time series: the trend, the cyclic behaviors, and the periodical behaviors. Trend changes the amplitude of the data but it may not change the pattern. Cyclic behaviors or periodical patterns can happen in any time, this means that the patterns shown in Figure 12 – The different time series that represents the same pattern can essentially represents the same thing. This means naïve Bayes may not be able to handle such variance.

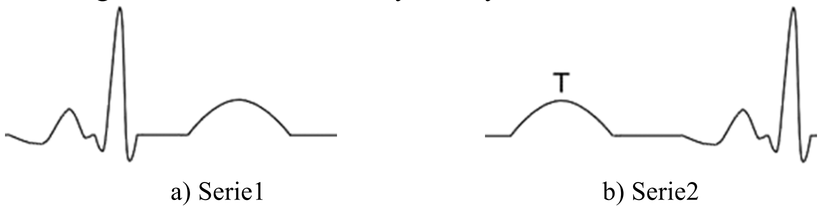


Figure 12 – The different time series that represents the same pattern

4. The size of the data is always an issue when doing supervised learning. This include both under-fitting and overfitting problems. Consider you are given the set of ECG sequence for Kelly labeled with either asleep or awake. After training, the model is 100% correct distinguish the state of Kelly's life circle. However, chances are, in the future, Kelly may change her living style such that the physical signal is totally altered. Another critical question is that what is the accuracy of applying this model to the data from Jack? Thus, if the goal is to generalize the common feature of the wake/sleep states of all human beings, the training data must be sufficiently large and diverse. If the data size is fixed and there is no guarantee of future data, evaluation methods like cross validation should applied to get a better general model.
5. There is also another issue of determine the correct window to extract features. This consists of two sub-problems: find the best place to put a window, and choose the best window length N. Finding the best place to put a window can be determined by probabilistic detection of state transfer like that we have seen in CoinBot. On the other hand, if the best N to distinguish sleep and awake signal is not known, then we need a way discover or approximate such an N. This can be solved by evaluation comparison, more specifically, choosing the N that generates the lowest average validation error.

References:

[1] "Consciousness" Merriam-Webster. Retrieved June 4, 2012

[2] Electrocardiography, https://en.wikipedia.org/wiki/Electrocardiography