Q & A for the Final Exam

by Baikjin Jung

Gary Geunbae Lee
CSED342 - Artificial Intelligence

June, 2022

1. **(20 points)** Consider the following grid world game:
This is a stationary MDP with an infinite horizon. The agent can only be in one of the six locations. It gets the reward (or punishment) written in a particular cell when it leaves the cell. It gets a reward of 10 for leaving the bottom-middle square and a reward of -100 (punishment of 100) for leaving the top-left square. In each iteration of the game, the agent has to choose a direction to move. The agent can choose to move either up, down, left, or right. There is a 0.8 probability that it will move in that direction and a 0.1 probability that it will move in either of the neighboring directions. For example, if the agent wants to move up, there is a 0.8 probability that it will move up, a 0.1 probability that it will move left, and a 0.1 probability that it will move right. If the agent bumps into a wall, it stays in its current location and does not get any reward (or punishment).

-100		
	+10	

Figure 1: The given grid world.

   (a) **(8 points)** Perform one step of value iteration and show the resulting value function for each state.

-10	0	0
0	+10	0

Figure 2: The right answer.

No partial Credit.

(b) **(12 points)** What is the value in the top-left state after performing another step of value iteration?
We take the maximum value of the four possible actions:

$$\begin{cases} \text{left:} & 0.8 \times (0 + (-10)) + 0.1 \times (-100 + 0) + 0.1 \times (-100 + 0) = -28 \\ \text{up:} & 0.8 \times (0 + (-10)) + 0.1 \times (-100 + 0) + 0.1 \times (-100 + 0) = -28 \\ \text{down:} & 0.8 \times (-100 + 0) + 0.1 \times (-100 + 0) + 0.1 \times (0 + (-10)) = -91 \\ \text{right:} & 0.8 \times (-100 + 0) + 0.1 \times (-100 + 0) + 0.1 \times (0 + (-10)) = -91 \end{cases}$$

Therefore, the answer is -19.

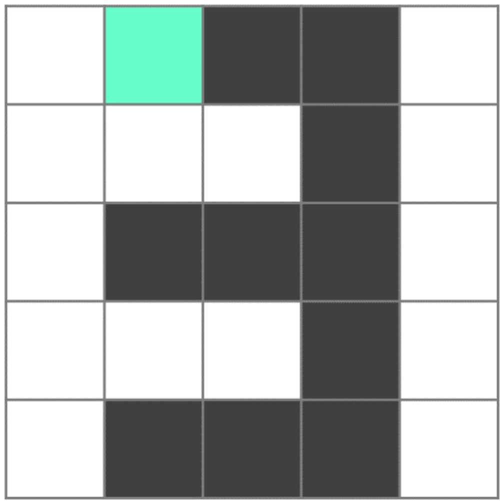
Partial Credit: 3 points for each direction.

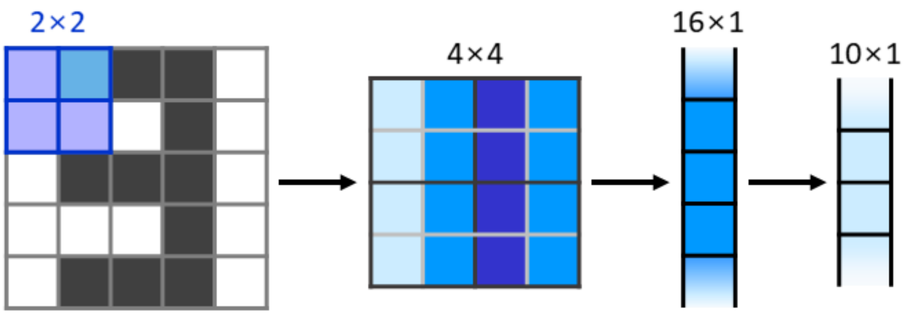
2. **(20 points)** Provided that one is faced with a task to classify a given image of a handwritten digit into one of ten classes representing integer values from 0 to 9, the go-to approach of these days would be using a convolutional neural network (CNN). Let's say that we are given a 5×5 pixel image of number 3, for example (the aquamarine pixel is what we are specially interested in.).
Our CNN consists of two layers in the following order:

- a convolution layer with 2×2 filter, which is followed by a vectorization procedure (just changing the form),
- and a fully-connected layer that maps a 4-dimensional column vector to a 10-dimensional column vector.

The figure below depicts the above-mentioned layers. Other details are as follows:

- in the original image x, a colored pixel has a value of 1, and a white pixel has a value of 0;
- the true label $y = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^{\mathrm{T}}$;





- the predicted label $\hat{y} = \begin{bmatrix}0 & 0 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0\end{bmatrix}^{\mathrm{T}}$;
- all the weight parameters (W) of the fully-connected layer have a value of 1.
- the output of the convolution layer

$$\forall n, m \in [1, 4] : c[n, m] = \sum_{u=1}^{2} \sum_{v=1}^{2} x[n + (u - 1), m + (v - 1)] \cdot k[u, v];$$

- the output of the fully-connected layer $\hat{y} = \sigma(W \times c)$, where

$$\sigma(z) = \frac{1}{1 + e^{-z}};$$

- the loss

$$L = \frac{1}{2} \sum_{i=1}^{10} (\hat{y}(i) - y(i))^2.$$

(a) **(10 points)** Calculate $\dfrac{\partial L}{\partial c}$. Refer to the derivatives below if needed.

$$\frac{d}{dz}[f(g(z))] = f'(g(z)) \cdot g'(z) \tag{1}$$

$$\frac{d}{dz}\left[\frac{1}{x}\right] = -\frac{1}{x^2} \tag{2}$$

$$\frac{d}{dz}\left[e^z\right] = e^z \tag{3}$$

$$\begin{aligned}
\frac{\partial L}{\partial c} &= \sum_{i=1}^{10} \frac{\partial L}{\partial \hat{y}(i)} \cdot \frac{\partial \hat{y}(i)}{\partial c} \\
&= \left(\sum_{i=1}^{10} (\hat{y}(i) - y(i)) \cdot \hat{y}(i)(1 - \hat{y}(i)) \right) \\
&= -0.125
\end{aligned}$$

Partial Credit:

- Differentiation of the mean squared error: 5 points,
- Differentitation of the sigmoid function: 5 points.

(b) **(10 points)** Calculate the back-propagated gradient for $k[1,2]$ when the upper left corner of the filter is on $x[1,1]$, which corresponds to the aquamarine pixel. For differentiation, refer to the derivatives above.

$$\begin{aligned}
\nabla k[1,2] &= \frac{\partial L}{\partial k[1,2]} \\
&= \sum_{i=1}^{10} \frac{\partial L}{\partial \hat{y}(i)} \cdot \frac{\partial \hat{y}(i)}{\partial k[1,2]} \\
&= \left(\sum_{i=1}^{10} (\hat{y}(i) - y(i)) \cdot \frac{\partial \hat{y}(i)}{\partial c} \right) \cdot \frac{\partial c}{\partial k[1,2]} \\
&= \left(\sum_{i=1}^{10} (\hat{y}(i) - y(i)) \cdot \hat{y}(i)(1 - \hat{y}(i)) \right) \cdot W_{i,\cdot} \cdot \frac{\partial c}{\partial k[1,2]} \\
&= \left(\sum_{i=1}^{10} (\hat{y}(i) - y(i))\hat{y}(i)(1 - \hat{y}(i)) \right) \cdot x[1,2] \\
&= -0.125
\end{aligned}$$

No partial Credit.

3. **(20 points)** Use the k-means algorithm and Euclidean distance to cluster eight data points into $k = 3$ clusters. The distance matrix based on the Euclidean distance is given in the table below. The coordinates of the data points are:

$$x^{(1)} = (2, 8), \quad x^{(2)} = (2, 5), \quad x^{(3)} = (1, 2), \quad x^{(4)} = (5, 8),$$
$$x^{(5)} = (7, 3), \quad x^{(6)} = (6, 4), \quad x^{(7)} = (8, 4), \quad x^{(8)} = (4, 7).$$

	$x^{(1)}$	$x^{(2)}$	$x^{(3)}$	$x^{(4)}$	$x^{(5)}$	$x^{(6)}$	$x^{(7)}$	$x^{(8)}$
$x^{(1)}$	0	3	6.0828	3	7.0711	5.6569	7.2111	2.2361
$x^{(2)}$	3	0	3.1623	4.2426	5.3852	4.1231	6.0828	2.8284
$x^{(3)}$	6.0828	3.1623	0	7.2111	6.0828	5.3852	7.2801	5.8310
$x^{(4)}$	3	4.2426	7.2111	0	5.3852	4.1231	5	1.4142
$x^{(5)}$	7.0711	5.3852	6.0828	5.3852	0	1.4142	1.4142	5
$x^{(6)}$	5.6569	4.1231	5.3852	4.1231	1.4142	0	2	3.6056
$x^{(7)}$	7.2111	6.0828	7.2801	5	1.4142	2	0	5
$x^{(8)}$	2.2361	2.8284	5.8310	1.4142	5	3.6056	5	0

(a) **(2 points)** Suppose that we initialize the centroids with k randomly chosen data points. Let's assume that those points are $\mu^{(1)} \leftarrow x^{(3)}$, $\mu^{(2)} \leftarrow x^{(4)}$, and $\mu^{(3)} \leftarrow x^{(6)}$. Perform one iteration and assign each data point to the closest cluster.

$$c^{(1)} \leftarrow C^{(2)}$$
$$c^{(2)} \leftarrow C^{(1)}$$
$$c^{(3)} \leftarrow C^{(1)}$$
$$c^{(4)} \leftarrow C^{(2)}$$
$$c^{(5)} \leftarrow C^{(3)}$$
$$c^{(6)} \leftarrow C^{(3)}$$
$$c^{(7)} \leftarrow C^{(3)}$$
$$c^{(8)} \leftarrow C^{(2)}$$

No partial Credit.

(b) **(4 points)** Next, move the centroids.

$$\mu^{(1)} \leftarrow \frac{1}{2}\left(x^{(2)} + x^{(3)}\right) = (1.5, 3.5)$$
$$\mu^{(2)} \leftarrow \frac{1}{3}\left(x^{(1)} + x^{(4)} + x^{(8)}\right) = (3.67, 7.67)$$
$$\mu^{(3)} \leftarrow \frac{1}{3}\left(x^{(5)} + x^{(6)} + x^{(7)}\right) = (7, 3.67)$$

No partial Credit.

(c) **(7 points)** Calculate the loss function before the first iteration

$$J_0\left(c^{(1)}, \cdots, c^{(m)}, \mu^{(1)}, \cdots, \mu^{(k)}\right) = \frac{1}{m}\sum_{i=1}^{m}\left\|x^{(i)} - \mu^{(c^{(i)})}\right\|_2^2,$$

where $c^{(i)}$ is the cluster assigned to the i^{th} data point by the algorithm, and the added quantity on the right-hand side is the square of the (given) Euclidean distance between two data points.

$$\begin{aligned} J_0 &= \frac{1}{8}\left(3^2 + 3.1623^2 + 1.4142^2 + 2^2 + 1.4142^2\right) \\ &= 3.375 \end{aligned}$$

No partial Credit.

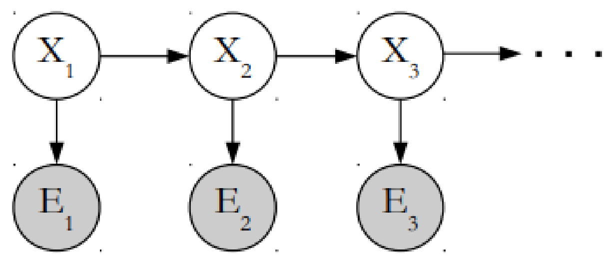
(d) **(7 points)** Calculate the loss function after the first iteration

$$J_1\left(c^{(1)}, \cdots, c^{(m)}, \mu^{(1)}, \cdots, \mu^{(k)}\right) = \frac{1}{m}\sum_{i=1}^{m}\left\|x^{(i)} - \mu^{(c^{(i)})}\right\|_2^2.$$

$$\begin{aligned} J_1 &= \frac{1}{8}\left(2.9 + 2.5 + 2.5 + 1.9 + 0.44 + 1.11 + 1.11 + 0.56\right) \\ &= 1.625 \end{aligned}$$

No partial Credit.

Q4) Most Likely Estimates in HMMs



The Viterbi algorithm finds the most probable sequence of hidden states $X_{1:T}$, given a sequence of observations $e_{1:T}$. Throughout this question you may assume there are no ties. Recall that for the canonical HMM structure, the Viterbi algorithm performs the following dynamic programming computations:

$$m_t[x_t] = P(e_t|x_t) \max_{x_{t-1}} P(x_t|x_{t-1}) m_{t-1}[x_{t-1}]$$

* Note about dynamic programming: dynamic programming is essentially a recursive relation in which the current value is defined as a function of previously computed values. In this case, the value at time t is defined as a function of the values at time t – 1.

(a) [5 points]

For the HMM structure above, which of the following probabilities are maximized by the sequence of states returned by the Viterbi algorithm? Pick **all** correct option(s).

1) $P(X_{1:T})$

2) $P(X_T|e_T)$

3) $P(X_{1:T}|e_{1:T})$

4) $P(X_{1:T}, e_{1:T})$

5) $P(X_1)P(e_1|X_1)\prod_{t=2}^{T} P(e_t|X_t)P(X_t|X_{t-1})$

6) $P(X_1)\prod_{t=2}^{T} P(X_t|X_{t-1})$

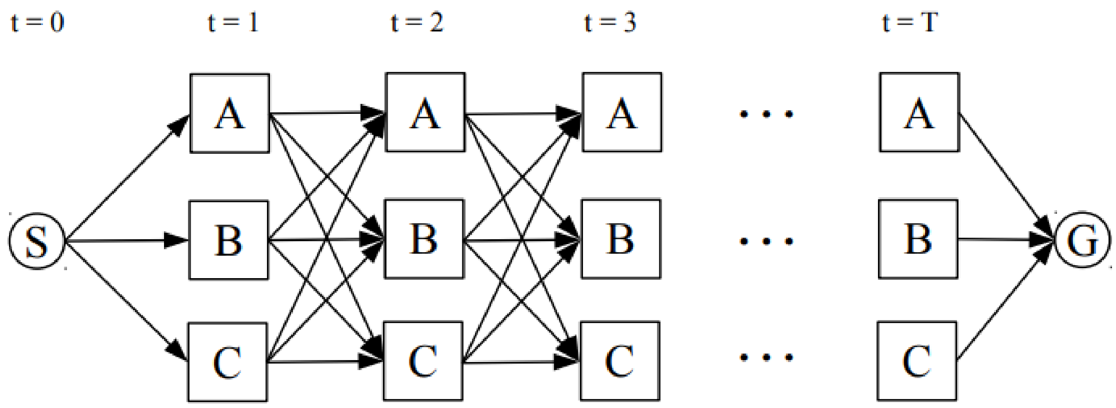
7) None of the above

Answer: 3, 4, 5

The sequence of states returned by the Viterbi Algorithm maximizes the conditional $= P(X_{1:T}|e_{1:T})$. Since, $P(e_{1:T})$ is just a constant, $P(X_{1:T}, e_{1:T}) \propto P(X_{1:T}|e_{1:T})$. Hence, the full joint $P(X_{1:T}, e_{1:T})$ is also maximized. The third option $P(X_1)P(e_1|X_1)\prod_{t=2}^{T} P(e_t|X_t)P(X_t|X_{t-1}) == P(X_{1:T}, e_{1:t})$ due to the conditional independences implied by the HMM structure; therefore, this is also maximized.

(b) 5 points each

Consider an HMM structure like the one in part (a) above. Say for all time steps t, the state Xt can take on one of the three values {A, B, C}. Then, we can represent the state transitions through the following directed graph, also called a Trellis Diagram.



We wish to formulate the most probable sequence of hidden state query as a graph search problem. Note in the diagram above, dummy nodes S and G have been added to represent the start state and the goal state respectively. Further, the transition from the starting node S to the first state X_1 occurs at time step t = 0; transition from X_T (the last HMM state) to the goal state G occurs at time step t = T.

Definition: Let $w^t_{Y \to Z}$, be the cost of the edge for the transition from state Y at time *t* to state Z at time *t+1*. For example, $w^1_{A \to B}$ is the cost of the edge for transition from state A at time 1 to state B at time 2.

(i) For which **one** of the following values for the weights $w^t_{Y \to Z}$, $1 \leq t < T$, would the minimum cost path be exactly the same as most likely sequence of states computed by the Viterbi algorithm?

(a) $w^t_{Y \to Z} = -P(X_{t+1} = Z | X_t = Y)$

(b) $w^t_{Y \to Z} = -P(e_{t+1} | X_{t+1} = Z) P(X_{t+1} = Z | X_t = Y)$

(c) $w^t_{Y \to Z} = -\log(P(X_{t+1} = Z | X_t = Y))$

(d) $w^t_{Y \to Z} = -\log(P(e_{t+1} | X_{t+1} = Z) P(X_{t+1} = Z | X_t = Y))$

(e) $w^t_{Y \to Z} = \frac{1}{P(X_{t+1} = Z | X_t = Y)}$

(f) $w^t_{Y \to Z} = \frac{1}{P(e_{t+1} | X_{t+1} = Z) P(X_{t+1} = Z | X_t = Y)}$

Answer: d

We want the solution to maximize the joint $P(X_{1:T}, e_{1:T}) = P(X_1)P(e_1|X_1)\prod_{t=2}^{T} P(e_t|X_t)P(X_t|X_{t-1})$.
Since, a search algorithm **minimizes** the cost, we want to pose this problem as a minimization problem. Hence, we can equivalently say that we want to minimize $\frac{1}{P(X_1)P(e_1|X_1)\prod_{t=2}^{T} P(e_t|X_t)P(X_t|X_{t-1})}$.
A search algorithm in its native form can only work with **additive** costs. Therefore, to turn the above products of probabilities into sums, we take the log.

Hence, we wish to minimize :

$$\log\left(\frac{1}{P(X_1)P(e_1|X_1)\prod_{t=2}^{T} P(e_t|X_t)P(X_t|X_{t-1})}\right) = -\log\left(P(X_1)P(e_1|X_1)\prod_{t=2}^{T} P(e_t|X_t)P(X_t|X_{t-1})\right)$$

$$= -\log\left(P(X_1)P(e_1|X_1)\right) - \sum_{t=2}^{T} \log\left(P(e_t|X_t)P(X_t|X_{t-1})\right).$$

If for $t > 1$, the edge cost is set to $-\log P(e_{t+1}|X_{t+1} = Z)P(X_{t+1} = Z|X_t = Y)$, we get the above as the total cost of the path.

(ii) The initial probability distribution of the state at time t = 1 is given P(X1 = Y), Y $\in$ {A, B, C}. Which **one** of the following should be the value of $w^0_{S\rightarrow Y}$, Y $\in$ {A, B, C} - these are the cost on the edges connecting S to the states at time t = 1?

(a) $w^0_{S\rightarrow Y} = -P(X_1 = Y)$ **(b)** $w^0_{S\rightarrow Y} = -P(e_1|X_1 = Y)P(X_1 = Y)$

(c) $w^0_{S\rightarrow Y} = -\log(P(X_1 = Y))$ **(d)** $w^0_{S\rightarrow Y} = -\log(P(e_1|X_1 = Y)P(X_1 = Y))$

(e) $w^0_{S\rightarrow Y} = \frac{1}{P(X_1 = Y)}$ **(f)** $w^0_{S\rightarrow Y} = \frac{1}{P(e_1|X_1 = Y)P(X_1 = Y)}$

Answer: d

(iii) Which **one** of the following should be the value of $w^T_{Y \to G}$, $Y \in \{A, B, C\}$ – these are the cost on the edges connecting the states at the last time step t = T to the goal state G?

(a) $w^T_{Y \to G} = -P(X_T = Y)$

(b) $w^T_{Y \to G} = -P(e_T|X_T = Y)P(X_T = Y)$

(c) $w^T_{Y \to G} = -\log(P(X_T = Y))$

(d) $w^T_{Y \to G} = -\log(P(e_T|X_T = Y)(P(X_T = Y))$

(e) $w^T_{Y \to G} = \dfrac{1}{P(X_T = Y)}$

(f) $w^T_{Y \to G} = \dfrac{1}{P(e_T|X_T = Y)P(X_T = Y)}$

(g) $w^T_{Y \to G} = \alpha,\ \alpha \in \mathbb{R}:$ (some constant)

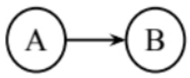
Answer: g

As long as the costs on all the three edges : $w^T_{A \to G}, w^T_{B \to G}, w^T_{C \to G}$ is the same, we will get the optimal answer. Note, it does not matter if the constant α is postive or negative because the total probability is only scaled by a positive constant $= e^{\alpha}$ (that is, the total cost of the path is (`sum of all log probabilities`) $+ w^T_{Y \to G} (== \alpha)$. When you exponentiate this to get the probabilities, you get $e^{\text{sum of log probabilities}} * e^{\alpha}$ — which is just a scaling by some positive number).

Q5) ML: Short Question and Answer Problems [20 points]

(a) Parameter Estimation and Smoothing

For the Bayes' net drawn on the left, A can take on values +a, -a, and B can take values +b and -b. We are given samples (on the right), and we want to use them to estimate P(A) and P(B|A).



$(-a,\ +b)$	$(-a,\ +b)$	$(-a,\ -b)$	$(-a,\ -b)$
$(-a,\ -b)$	$(-a,\ -b)$	$(-a,\ +b)$	
$(-a,\ +b)$	$(-a,\ -b)$	$(+a,\ +b)$	

(i) **[5 points]** Compute the maximum likelihood estimates for P(A) and P(B|A), and fill in the 2 tables below ((1) ~(6)).

A	**P(A)**
+a	(1)
-a	(2)

A	**B**	**P(B|A)**
+a	+b	(3)
+a	-b	(4)
-a	+b	(5)
-a	-b	(6)

Answer:

(1): 1/10 (2): 9/10

(3): 1/1 (4): 0/1 (5): 4/9 (6): 5/9

(ii) [5 points] Compute the estimates for P(A) and P(B|A) using Laplace smoothing with strength k=2, and fill in the 2 tables below ((1) ~ (6)).

A	**P(A)**
+a	(1)
-a	(2)

A	**B**	**P(B|A)**
+a	+b	(3)
+a	-b	(4)
-a	+b	(5)
-a	-b	(6)

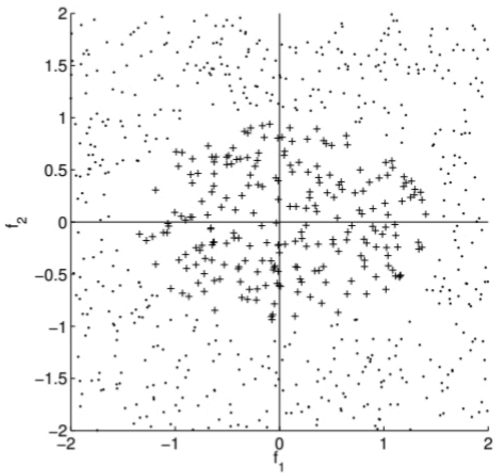
Answer:

(1): 3/14 (2): 11/14

(3): 3/5 (4): 2/5 (5): 6/13 (6): 7/13

(b) [5 points] Linear Separability

You are given samples from 2 classes (. And +) with each sample being described by 2 features f_1 and f_2. These samples are plotted in the following figure. You observe that these samples are not *linearly* separable using just these 2 features. Pick the minimal set of features (choices) below that you could use alongside f_1 and f_2, to *linearly* separate samples from the 2 classes.



Choice A) $f_1 < 1.5$ Choice B) $f_1 > -1.5$ Choice C) $f_2 < -1$

Choice D) $f_1 > 1.5$ Choice E) $f_2 < 1$ Choice F) $f_2 > -1$

Choice G) $f_1 < -1.5$ Choice H) $f_2 > 1$ Choice I) f_1^2

Choice J) f_2^2 Choice K) $|f_1 + f_2|$

Choice L) Even using all these features alongside f_1 and f_2 will not make the samples linearly separable.

Answer: Choice I and Choice J

(c) [5 points] Perceptrons

In this question, you will perform perceptron updates. You have 2 classes, +1 and -1, and 3 features f0, f1, f2 for each training point. The +1 class is predicted if $w \cdot f > 0$ and the -1 class is predicted otherwise.

You start with the weight vector w = [1 0 0]. In the table below, do a perceptron update for each of the given samples. If the w vector does not change, write "No Change", otherwise write down the new w vector for (i) ~ (iii).

f0 f1 f2	Class	Updated w
1 7 8	-1	(i)
1 6 8	-1	(ii)
1 9 6	+1	(iii)

Answer:

(i): [0 -7 -8]

(ii): Unchanged (no misclassification)

(iii): [1 2 -2]