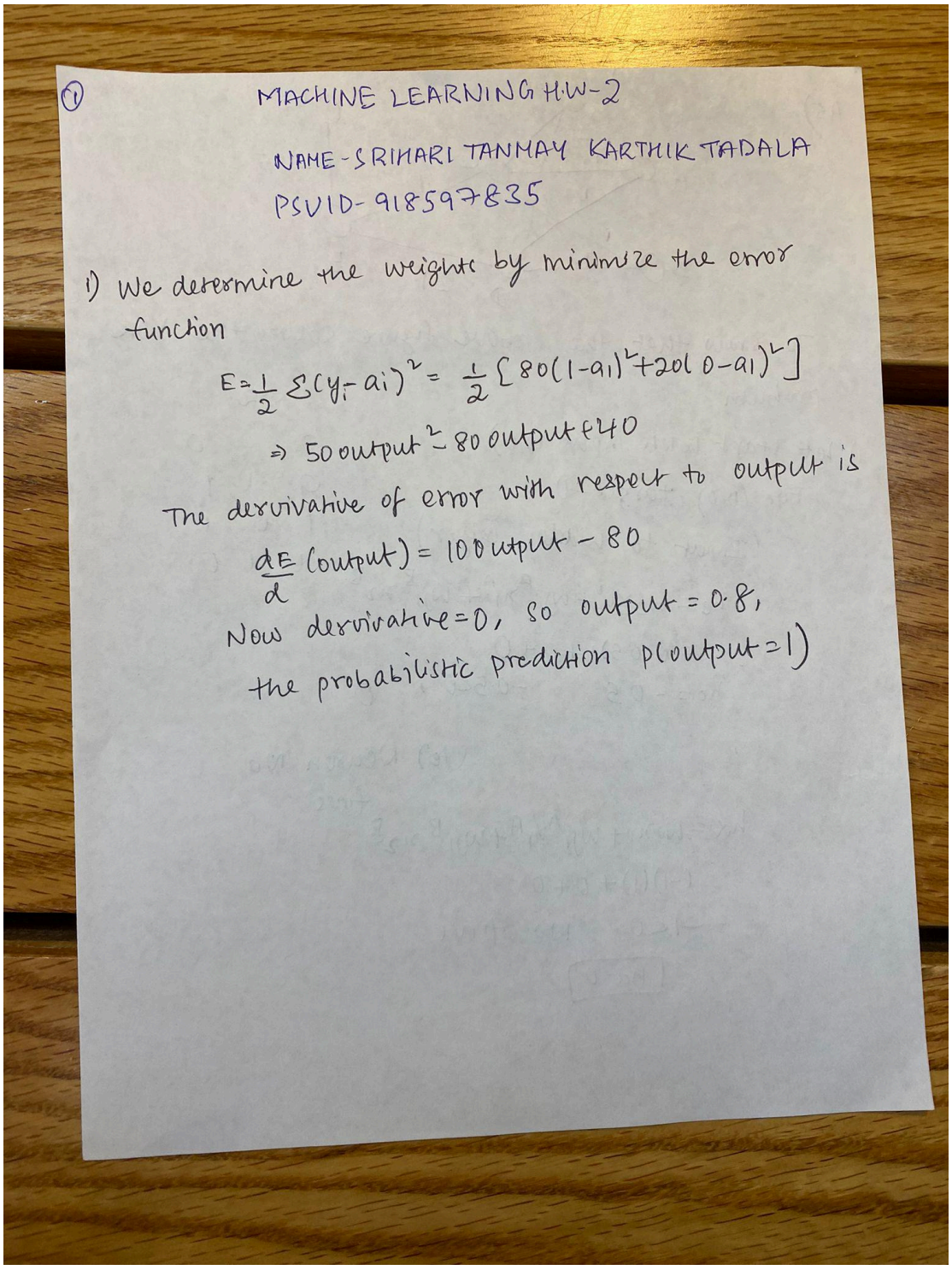
Machine Learning,
Spring 2024 Homework #2
SRIHARI TANMAY KARTHIK TADALA

Solution 1 -



① MACHINE LEARNING H.W-2

NAME - SRIHARI TANMAY KARTHIK TADALA
PSVID - 918597835

i) We determine the weights by minimize the error function

$$E = \frac{1}{2}\sum(y_i - a_i)^2 = \frac{1}{2}[80(1-a_1)^2 + 20(0-a_1)^2]$$

$$\Rightarrow 50\,output^2 - 80\,output + 40$$

The derivative of error with respect to output is

$$\frac{dE}{d}(output) = 100utput - 80$$

Now derivative = 0, so output = 0.8,
the probabilistic prediction p(output = 1)

Solution 2-

The order in which training samples are presented to a neural network during its training phase is pivotal, as it greatly impacts the learning process. When the network is subjected to the complete set of training examples repeatedly, for example, 1000 times, it gains an advantage by acquiring knowledge from each example during every cycle. This approach guarantees a well-rounded learning experience by incorporating data from the entire dataset.

In contrast, when the network is trained by repeatedly presenting each example 1000 times before moving on to the next, it has a tendency to focus on learning from the current example and neglect the previous ones. The network's concentrated approach can result in the loss of knowledge acquired from previous examples, which is referred to as catastrophic forgetting. Consequently, the network's weights are primarily modified according to the most recent examples, which may compromise its capacity to precisely classify earlier samples.

Hence, the sequence and presentation of training examples significantly impact the network's capacity to acquire and preserve information throughout the dataset.

Solution 3-

1) Networks of perceptrons with linear activation functions are unsatisfactory because a single perceptron can replicate the functionality of an entire network with linear activations. The redundancy of the network arises from the fact that both a single perceptron and a network have the ability to compute a weighted sum of inputs, which is the fundamental operation of linear activation.

2) When every layer in a network carries out a linear transformation, the overall result, regardless of the number of layers, is the same as applying a single linear transformation to the input. This suggests that increasing the number of layers does not improve the network's capacity to represent intricate patterns that go beyond simple linear relationships.

3) The utilization of linear activation functions hinders networks from effectively solving intricate problems that display non-linear connections, such as the XOR problem, image

recognition, or language processing. This constraint arises from their fundamental linear characteristic, which restricts them to linear approximations and makes them inadequate for tasks that require the representation of non-linearities.

4) The utilization of linear activation functions limits the network's capacity to learn, limiting its ability to adapt to complex data patterns and diverse datasets. This greatly limits the range of issues that these networks can effectively address.

Perceptron networks with linear activation functions are considered simplistic because they are naturally unable to capture nonlinear dynamics in data. They have limited utility in tasks that are straightforward and linearly separable, which are relatively rare in practical situations. In order to effectively model complex phenomena and data that display non-linear relationships, it is essential to incorporate non-linear activation functions. This allows neural networks to fully utilize their potential and adaptability.

Solution 4-

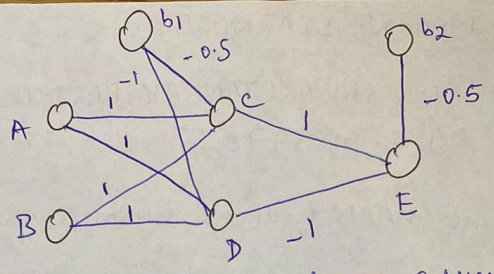
Overfitting is prone to happen under specific circumstances, including:

1) Insufficient Training Data: The probability of overfitting rises as the size of the training dataset decreases. With a reduced number of data points, the model may prioritize memorizing individual instances rather than comprehending general patterns. This can result in poor performance when faced with unfamiliar data, as the model learns the specific noise or irregularities present in the training set.

2) Increased Model Complexity: A model that has a significant number of parameters is more susceptible to overfitting. The reason for this is that a complex model possesses the capacity to accurately conform to the training data, capturing not only the fundamental patterns but also the incidental variations and unpredictability. Consequently, although it may perform exceptionally well on the training data, its capacity to apply that knowledge to new data is compromised, resulting in overfitting.

Solution 5-

A5)



To show that the above figure solves the XOR problem

XOR		
0	0	0
0	1	1
1	0	1
1	1	0

1) let start with input (0,0)

For (0,0), target = 0

(Input to hidden layer)

$h_c = w_0 x_0' + w_{ji}^A x_1^A + w_{ji}^B x_2^B$

$= (-0.5)1 + 0 + 0$

$h_c = -0.5 \quad -0.5 < 0, \quad 0$

(ie) Neuron No fires.

$h_d = w_0 x_0 + w_{ji}^A x_1^A + w_{ji}^B x_2^B$

$= (-1)(1) + 0 + 0$

$= -1 < 0$ No fires

$\boxed{h_d = 0}$

(2)

Output y(E) to hidden layer

$w_0 x_0 + w_{kj} x_i^C + w_{kj} x_2^D$

$= (-0.5)1 + 0 + 0 = -0.5 < 0 \Rightarrow 0$

$y(E) = 0$

Since the output $y(E) = 0$ and target = 0 matches and it satisfies for input (0,0)

2) For input (0,1) and target = 1

$hc = w_0 x_0^{bl} + w_{ij}^{A} x_1^A + w_{ji}^{B} x_2^B$

$= (-0.5)(1) + 0 + 1(1)$

$= -0.5 + 1 \Rightarrow 0.5$ $\quad 0.5 > 0$

hc = 1 | Neuron fires

$hd = w_0 x_0^{bi} + w_{ji}^{A} x_1^A + w_{ij}^{B} x_2^B$

$= (-1)(1) + 0 + 1 \Rightarrow -1 + 1 \Rightarrow 0 \geq 0$

$hd \Rightarrow 0$ | Neuron fires

hidden to output layer

$y(E) = (-0.5)1 + 1 + (-1)(0)$

$= -0.5 + 1 + 0 = +0.5$

Neuron fires

$y(E) = 1$ satisfies target

3. For input (1,0) & target 1

$h_c = (-0.5)(1) + 1(1) + 0$

$= -0.5 + 1 \Rightarrow 0.5 > 0$ Neuron fires

$h_c = 1$

$h_c = (-1)(1) + (1)(1) + 0 = 0 \leq 0$ No fire

$h_d = 0$

$y(E) = (-0.5)(1) + (1) + 0$

$= 0.5 \Rightarrow 0.5 > 0$

$y(E) = 1$

hence the output got from input (1,0) is 1

hence the matches the target = 1

4. For input (1,1) and target = 0

$h_c = (-0.5)(1) + 1 + 1 \Rightarrow -0.5 + 1 + 1$

$\Rightarrow 1.5$

Neuron fires.

$h_d = (-1)(1) + 1 + 1$

$\Rightarrow -1 + 1 + 1 \Rightarrow 1$

$y(E) = (-0.5)(1) + (-1)$

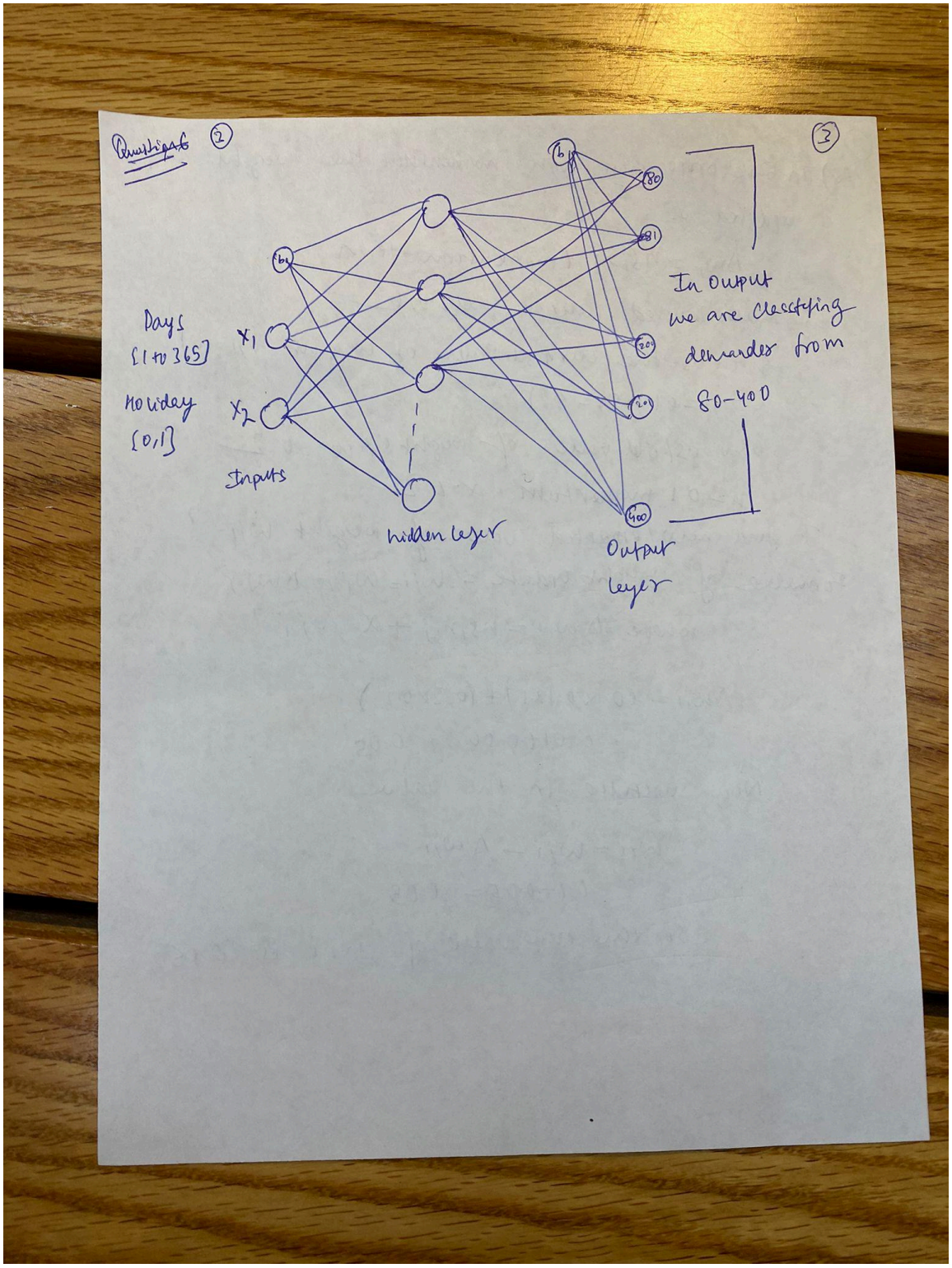
$= -0.5$

$= -0.5 < 0$

$y(E) = 0$

Hence the ~~output~~ input (1,1) the output is 0 & target given was also 0 so the target given input satisfies & matches. ∴ Hence the given fisine & weight satisfies or solves the problem of XOR.

Solution 6-



6 1)

1. Data Gathering: Collect historical data: Gather the daily demand statistics from the past 5 years, including daily values and any relevant attributes.

2. Data Processing and Normalization: Normalization is essential to streamline the data for effective model training.

• Input Features: Allocate a range of features to the input, including: o Days (1 to 365) to encompass a full year, and then duplicate for the following years,

Holiday Indicator: Use a binary input to represent holidays, with 0 indicating no holiday and 1 indicating a holiday.

Additional functionalities can be incorporated as required.

• Hidden Neurons: The ideal number of hidden neurons is determined by experimenting and testing different counts. Initially, the model is configured with 5 concealed layers, and subsequent modifications can be made depending on its performance.

• The number of output neurons ranges from 80 to 400, and they are used to classify electricity demand patterns over the last 5 years. For instance, when the 200th neuron is activated, it indicates the categorization of electricity demand for the upcoming 5 days.

3. Data Division: Divide the dataset into separate sets for training, testing, and validation.

4. Training and Evaluating the Model: Begin by training the model using data from the initial 4-year period. Afterwards, assess the model's accuracy by using the data from the fifth year to evaluate its performance.

5. Optimizing the Learning Rate: It is recommended to use a learning rate within the range of 0.001 to 0.01. Commence with a learning rate of 0.001, modifying it as necessary depending on the training results of the model.

6 2) To optimize your MLP-based system for electricity demand prediction by incorporating weather forecast data, adhere to the following procedure:

1) Increase the number of input neurons by incorporating daytime and nighttime temperatures as additional input features, with a temperature range of -50 to 50 degrees Celsius.

2) Standardize Weather Data: Ensure that weather inputs are rescaled to match the existing features in order to achieve optimal performance of the model.

3) Modify the input layer of the model to incorporate these supplementary weather characteristics.

4) Enhance Data Sets: Integrate weather-related inputs into both the training and testing datasets.

5) Enhance Model Training: Utilize the enriched dataset to train the MLP model, incorporating the effects of weather on electricity demand.

6. 3) The success or limitations of a system in predicting electricity consumption are influenced by various factors that affect its effectiveness.

Key Factors for Success:

• Data Integrity: Ensuring the accuracy of historical demand data and weather forecasts, as well as including relevant features, enhances the effectiveness of predictions.

• Model Design: An appropriately configured MLP architecture is essential for detecting complex patterns in electricity demand and the corresponding weather conditions.

• Feature Management: The careful selection and processing of important features, particularly weather information, improves the accuracy of the model.

• Overfitting Prevention: Utilizing regularization techniques aids in mitigating overfitting, thereby preserving the model's ability to generalize.

Possible constraints:

• Unforeseen external factors: Demand changes caused by unexpected events such as natural disasters, policy alterations, or economic fluctuations may be difficult to predict, which can impact the accuracy of the model.

• Timeframe for Forecasting: Multilayer Perceptrons (MLPs) perform exceptionally well in predicting outcomes in the short term, but their accuracy in making long-term projections may be less dependable.

• Aberrant Weather Conditions: The model may struggle to accurately forecast demand in the presence of extreme weather scenarios that deviate significantly from historical data.

• Data Quality Concerns: Insufficient data quality, which includes the absence or inaccuracy of historical and weather data, can have a negative impact on the accuracy of forecasts.

• Overfitting can occur when a model is excessively complex or improperly tuned, leading to inaccurate forecasts.

Solution 7-

1) Data Preprocessing and Normalization: Standardize all inputs by transforming them to have a mean of zero and a standard deviation of one. This process helps in preparing the data for neural network training.

• Input Neuron Configuration: The inputs will consist of a diverse range of features, as specified:
The term "Week" is used to represent a numerical value ranging from 1 to 52, which indicates the specific week of the year. The daytime temperature ranges from -50 to 50 degrees Celsius.

Seasons are divided into four categories: Spring, Summer, Autumn, and Winter, and they need to be properly normalized.

The presence of an epidemic is indicated by a binary input: 1 for presence and 0 for absence.

• Optimal number of hidden neurons: The most suitable number of hidden neurons is determined through experimentation and fine-tuning. Initially, there are 5 concealed layers, but more layers can be investigated if needed.

• Output Neurons: The number of output neurons is determined by the capacity of the geriatric ward, where each neuron may represent a group of 10 beds.

2) Data Division: Following the process of normalization, the dataset should be partitioned into separate groups for Training, Testing, and Validation.

3) Model Training and Evaluation: Commence with data from the initial four years for preliminary training. Afterwards, use the data from the last year to evaluate the accuracy and performance of the model.

4) Learning Rate Adjustment: Begin with initial learning rates ranging from 0.001 to 0.01, and make necessary adjustments to improve training results.The effectiveness of the system depends on various factors, such as the data's quality and quantity, the choice of features, the model's design, and the complexity of the patterns in bed demand.

Solution 8-

(4)

Ans.) In back propogation with momentum the weights are updated as

$\Delta w_{ji} = \eta \delta_j x_{ji} + \text{momentum term}$

Q) In multi layered neural network

given us the current value of weight $w_{ji} = 0.1$

$x_{ji} = 0.1$, $\delta_j = 0.1$

previous/old values of weights $\Delta w_i = 0.2$

$\eta = 0.1$ momentum, $\alpha = 0.2$

To find new/updated value of weight w_{ji} - ?

Formulae of weight update = $w_{ji} = w_{ji} - \Delta w_{ji}$

where $\Delta w_{ji} = \eta \delta_j x_{ij} + \alpha w_{ji}^{n-1}$

$\Delta w_{ji} = (0.1 \times 0.1 \times 1) + (0.2 \times 0.2)$

$= 0.01 + 0.04 = 0.05$.

Now substitue in the value

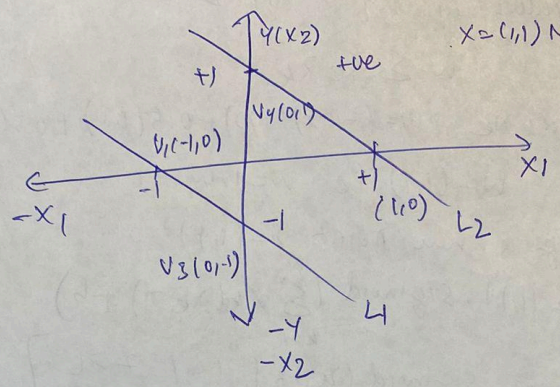
$w_{ji} = w_{ji} - \Delta w_{ji}$

$= 0.1 - 0.05 = 0.05$

So the new value of w_{ji} is 0.05.

Solution 9 -

A9. a)



X=(1,1) Newpoint)

Given 4 supports values

	x_1	x_2	class	α
v_1	-1	0	-1	$\alpha_1 = -0.5$
v_2	1	0	1	$\alpha_2 = 0.5$
v_3	0	-1	-1	$\alpha_3 = -0.5$
v_4	0	1	1	$\alpha_4 = 0.5$

Bias=0

The equation of a blue line $w_1x_1 + w_2x_2 + b = 0$

Now to find w_1 & w_2, the formulae to find vector weights is $w = \sum_{k \in y} (\alpha_k x_k)$

Training examples

(4)

We have values of α_1 & x_1, x_2

$$w = \sum \alpha_k x_k$$

$$w = \sum \alpha_k x_k \; ; \; w = (-0.5)(-1,0) + 0.5(1,0) + 0.5(0,1)$$

$$w = (1,1) \text{ ie } w_1 = 1, w_2 = 1$$

a) Classifying a new point $x = (1,1)$

$$h = (1,1), \text{sigmoid}\left(\sum_{k=1}^{m} \alpha_k (x_k \cdot x) + b\right)$$

where $\text{sigmoid}(z_k) = \begin{cases} 1 & \text{if } z > 0 \\ -1 & \text{if } z \leq 0 \end{cases}$

$$h(1,1) \rightarrow \text{sigmoid}\{[-0.5(-1,0)(1,1)] + [0.5(1,0)(1,1)] + [-0.5(0,-1)(1,1)] + [0.5(0,1)(1,1)] + 0$$

$$h(1,1) \Rightarrow \text{sigmoid}[2]$$

$$= 1$$

$\text{Sgmd}(2) = 1$ if $z > 0$

$z > 0$

This means that new input $x = (1,1)$ is on the positive side of the boundary

$$h(1,1) = 1$$

(3)

a)b) The equation of sepration line is

$$w_1 x_1 + w_2 x_2 + b = 0$$

putting values of w_1 & w_2

$$1(x_1) + 1(x_2) + 0 = 0$$

$$x_1 + x_2 = 0 \quad \boxed{x_2 = -x_1}$$

finding the value of x_2 when $x_1 = 0$

$$x_2 = -x_1 \qquad (x_1 = 0, x_2 = 0)$$

putting $x_1 = 0, x_2 = 0$

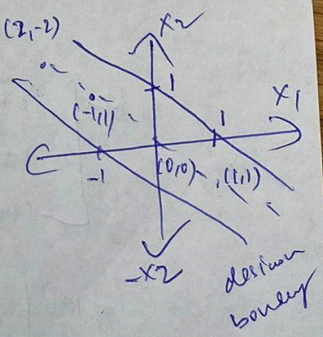
$x_1 = 1, x_2 = -1$

$x_1 = 2, x_2 = -2$

$x_1 = -1, x_2 = 1$

$x_1 = -2, x_2 = 2$

we get the decision boundary on seperating line as



so from the above is the decision boundary sepration line crossing from center when $x_1 = 0, x_2 = 0$

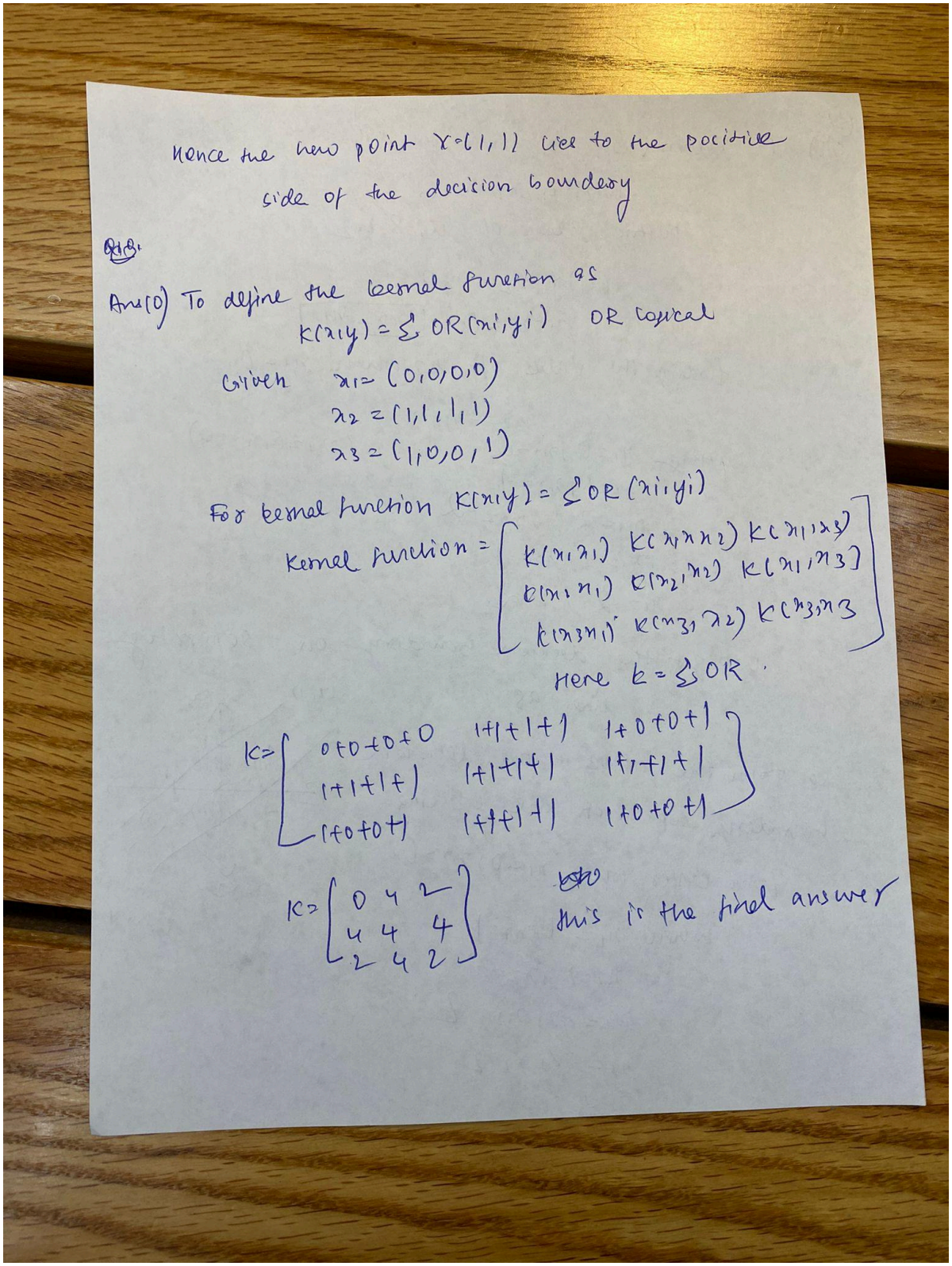
when $x_1 = -1, x_2 = 1$

$x_2 = 1, x_2 = -1$

$x_1 = 2, x_2 = -2$

$x_1 = -2, x_2 = 2$

Solution 10-



Hence the new point $x=(1,1)$ lies to the positive side of the decision boundary

Q10.

Ans(10) To define the kernel function as

$K(x,y) = \sum OR(x_i, y_i)$ OR logical

Given $x_1 = (0,0,0,0)$

$x_2 = (1,1,1,1)$

$x_3 = (1,0,0,1)$

For kernel function $K(x,y) = \sum OR(x_i, y_i)$

Kernel function $= \begin{bmatrix} K(x_1,x_1) & K(x_1,x_2) & K(x_1,x_3) \\ K(x_2,x_1) & K(x_2,x_2) & K(x_1,x_3) \\ K(x_3,x_1) & K(x_3,x_2) & K(x_3,x_3) \end{bmatrix}$

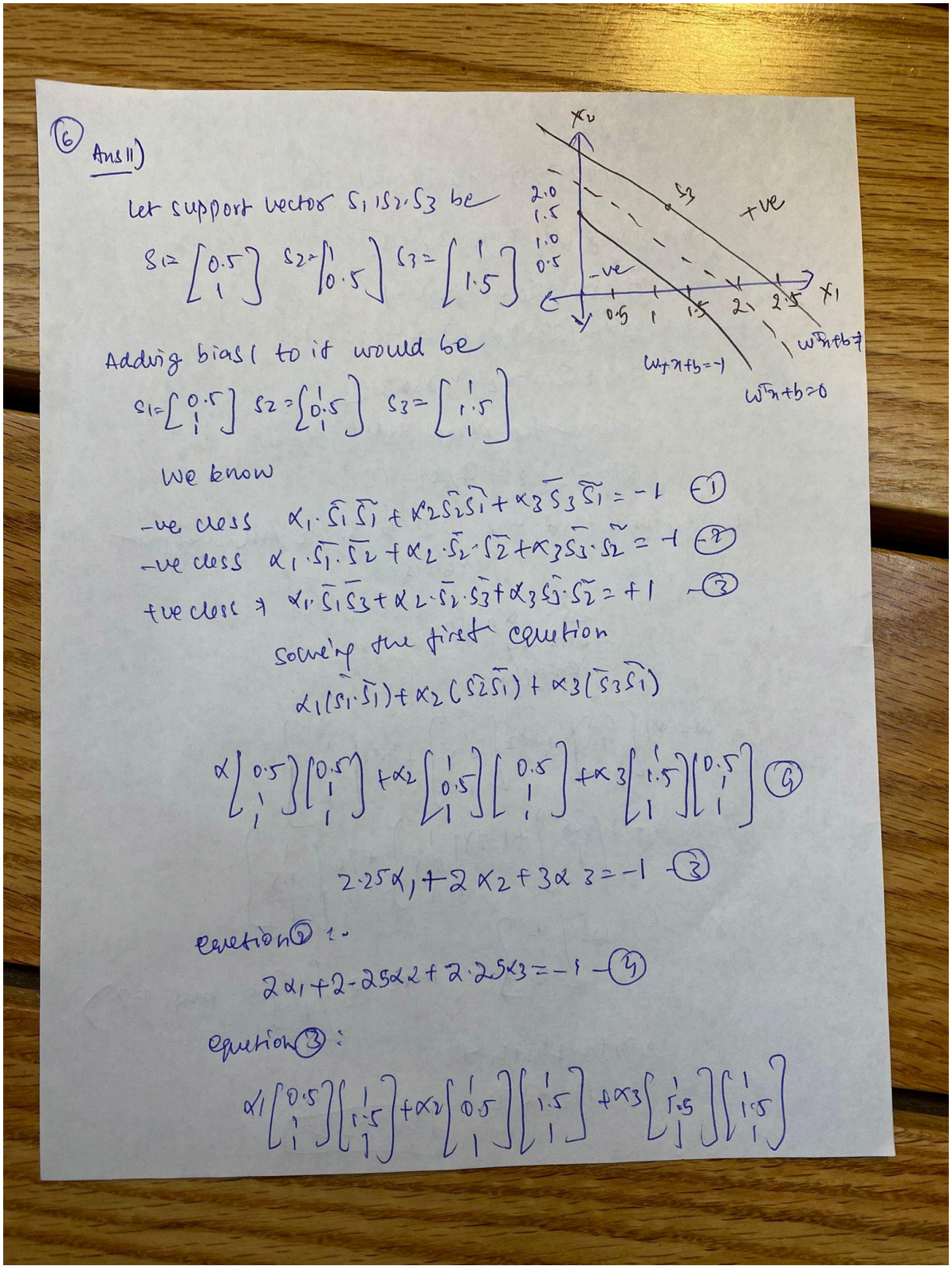
Here $k = \sum OR$.

$$K = \begin{bmatrix} 0+0+0+0 & 1+1+1+1 & 1+0+0+1 \\ 1+1+1+1 & 1+1+1+1 & 1+1+1+1 \\ 1+0+0+1 & 1+1+1+1 & 1+0+0+1 \end{bmatrix}$$

$$K = \begin{bmatrix} 0 & 4 & 2 \\ 4 & 4 & 4 \\ 2 & 4 & 2 \end{bmatrix}$$

this is the final answer

Solution 11 & 12 -



(6) Ans 11)

Let support vector s_1, s_2, s_3 be

$$s_1 = \begin{bmatrix} 0.5 \\ 1 \end{bmatrix} \quad s_2 = \begin{bmatrix} 1 \\ 0.5 \end{bmatrix} \quad s_3 = \begin{bmatrix} 1 \\ 1.5 \end{bmatrix}$$

Adding bias 1 to it would be

$$s_1 = \begin{bmatrix} 0.5 \\ 1 \\ 1 \end{bmatrix} \quad s_2 = \begin{bmatrix} 1 \\ 0.5 \\ 1 \end{bmatrix} \quad s_3 = \begin{bmatrix} 1 \\ 1.5 \\ 1 \end{bmatrix}$$

We know

-ve class $\alpha_1 \cdot \tilde{s_1}\tilde{s_1} + \alpha_2 \tilde{s_2}\tilde{s_1} + \alpha_3 \tilde{s_3}\tilde{s_1} = -1$ — (1)

-ve class $\alpha_1 \cdot \tilde{s_1}\cdot\tilde{s_2} + \alpha_2 \cdot \tilde{s_2}\cdot\tilde{s_2} + \alpha_3 \tilde{s_3}\cdot\tilde{s_2} = -1$ — (2)

+ve class $\alpha_1 \tilde{s_1}\tilde{s_3} + \alpha_2 \cdot \tilde{s_2}\cdot\tilde{s_3} + \alpha_3 \tilde{s_3}\cdot\tilde{s_2} = +1$ — (3)

Solving the first equation

$$\alpha_1(\tilde{s_1}\cdot\tilde{s_1}) + \alpha_2(\tilde{s_2}\tilde{s_1}) + \alpha_3(\tilde{s_3}\tilde{s_1})$$

$$\alpha \begin{bmatrix} 0.5 \\ 1 \\ 1 \end{bmatrix}\begin{bmatrix} 0.5 \\ 1 \\ 1 \end{bmatrix} + \alpha_2 \begin{bmatrix} 1 \\ 0.5 \\ 1 \end{bmatrix}\begin{bmatrix} 0.5 \\ 1 \\ 1 \end{bmatrix} + \alpha 3 \begin{bmatrix} 1 \\ 1.5 \\ 1 \end{bmatrix}\begin{bmatrix} 0.5 \\ 1 \\ 1 \end{bmatrix} \quad (4)$$

$$2.25\alpha_1 + 2\alpha_2 + 3\alpha_3 = -1 \quad (5)$$

equation (4) :-

$$2\alpha_1 + 2.25\alpha_2 + 2.25\alpha_3 = -1 \quad (4)$$

equation (5) :

$$\alpha_1 \begin{bmatrix} 0.5 \\ 1 \\ 1 \end{bmatrix}\begin{bmatrix} 1 \\ 1.5 \\ 1 \end{bmatrix} + \alpha_2 \begin{bmatrix} 1 \\ 0.5 \\ 1 \end{bmatrix}\begin{bmatrix} 1 \\ 1.5 \\ 1 \end{bmatrix} + \alpha_3 \begin{bmatrix} 1 \\ 1.5 \\ 1 \end{bmatrix}\begin{bmatrix} 1 \\ 1.5 \\ 1 \end{bmatrix}$$

(7)

Equation of line sepration is $x_1 + x_2 = 2$ | here 2 is intercept & slope m = 1

$\boxed{x_1 = 1 - x_2}$

Now putting the value of x_1 & we get x_2

$x_1 = 2, x_2 = 0$	$x_1 = 1.5, x_2 = 0.5$
$x_1 = 1, x_2 = 1$	$x_1 = 0, x_2 = 2$

Also, the line of equation with maximum margin is $\boxed{x_1 - x_2 = 2}$

Ans12) For the polynomial kernal function

$$k(x, z) = [(x \cdot z) + 1]^d$$

To show that:

$$k(x, z) = \phi(x) \cdot \phi(z)$$

where $\phi(x) = \{1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1 x_2\}$

Given $k(x, z) = [(x_1 x_2) \cdot (z_1 \cdot z_2)^2 + 1]^2$

$$\Rightarrow [x_1 z_1 + x_2 z_2 + 1]^2$$

$$k(x, z) = x_1^2 z_1^2 + x_2^2 z_2^2 + 1 + 2x_1 z_1 x_2 z_2 + 2x_1 z_1 - 2x_2 z_2 \quad \text{--- (1)}$$

Now we need to show that $K(x, z) = [\phi(x) \cdot \phi(z)]$

where $\phi(x) = (1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1 x_2)$

similarly $\phi(z) = (1, \sqrt{2}z_1, \sqrt{2}z_2, z_1^2, z_2^2, \sqrt{2}z_1 z_2)$

putting $\phi(z)\phi(x)$ on the RHS of the equation we get

$$[\phi(x)\cdot\phi(z)] = [(1+\sqrt{2}x_1+\sqrt{2}x_2+x_1^2+x_2^2+\sqrt{2}x_1x_2)(1+\sqrt{2}z_1+\sqrt{2}z_2+z_1^2+z_2^2+\sqrt{2}z_1z_2)]$$

$$= 1+\sqrt{2}x_1\sqrt{2}z_1+\sqrt{2}x_2\sqrt{2}z_2+x_1^2z_1^2+\sqrt{2}x_1x_2\sqrt{2}z_1z_2$$

$$= 1+2x_1z_1+2x_2z_2+x_1^2z_1^2+2x_1x_2z_1z_2+x_2^2z_2^2+2x_1z_1x_2z_2 \quad \text{--- (2)}$$

Since eq (1) & eq (2) are same we have proved that

$$K(x,z) = [\phi(x)\cdot\phi(z)]$$

when showing both left hand side $[K(x,z)]$ & right hand right side $[\phi(x)\cdot\phi(z)]$ seperately, we got same answer for both, hence it is proved.

Solution 13-

Observation and comment: _

The optimal learning rate, n=0.1, achieved the best result with 11 steps to reach the global minimum. _ The optimal learning rate, n=0.01, achieved the best result with 129 steps to reach the global minimum.

The observation for the learning rate n=0.001 is that, despite taking 500 steps, it did not reach the global minimum. However, it came very close, with a value of f(x,y)=2.31.
This suggests that a higher learning rate leads to a faster convergence to the global minimum.

```
import random

 def calculate_f(x, y):

        return 52 - 4*x + 2*x**2 + 24*y + 3*y**2 # Derivative of f(x,y) with respect to x
def derivative_x(x):

        return 4*x - 4
 # Derivative of f(x,y) with respect to y

def derivative_y(y):

        return 6*y + 24


 def gradient_descent(x_init, y_init, num_steps, learning_rate):

          x, y = x_init, y_init
         for step in range(num_steps):
        x -= learning_rate * derivative_x(x)
         y -= learning_rate * derivative_y(y)
         z = calculate_f(x, y)
         print(f"STEP[{step}]: f(x,y) => f({round(x,2)},{round(y,2)}) ==> {round(z,2)}")
         learning_rate = 0.001
         num_steps = 500
         x_initial = random.randint(-10, 10)
         y_initial = random.randint(-10, 10)
         print(f"\nInitial random points: \n({x_initial},{y_initial})") gradient_descent(x_initial,
        y_initial, num_steps, learning_rate) #Main Program Ends
```

```
learning_rate = 0.001
 num_steps = 500
 x_initial = random.randint(-10, 10)
 y_initial = random.randint(-10, 10)
 print(f"\nInitial random points: \n({x_initial},{y_initial})") gradient_descent(x_initial,
y_initial, num_steps, learning_rate) #Main Program Ends
```