**Data Mining & Machine Learning – Assignment 3**

(20% of total grade)

\* If two homework submissions are found to be similar to each other, both submissions will receive 0.

\* Homework solutions must be submitted through Canvas. If you have multiple files, please include all files as one zip file.

\* For coding assignments, it is strongly recommended to use **Jupytor notebook** and submit **.ipynb** file.

\* Answers with **math expressions** and **graphs** can be handwritten and scanned.

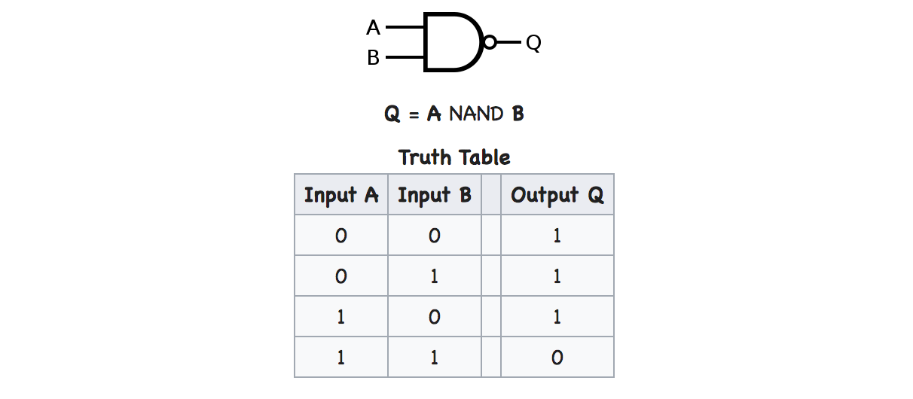
\* If you find any **typo/error** in the assignment, let me know.

1. [5 pts] In a decision tree, we can use ‘Information gain’ or ‘Gini’ as the measure in attribute selection.

When class distribution is very skewed (e.g., most class values are 0 or 1), explain in detail which one is better. Explain this using the formula of Information Gain and Gini.

2. [5 pts] When a problem is non-linear separable, the Perceptron Training Rule never stops, while the Delta Rule converges to a local optimum and stops. Explain the reason in detail using the update rule of each method.

3. [5 pts] (Refer to p. 9-10 in the slides) Implement the following NAND table using a perceptron.

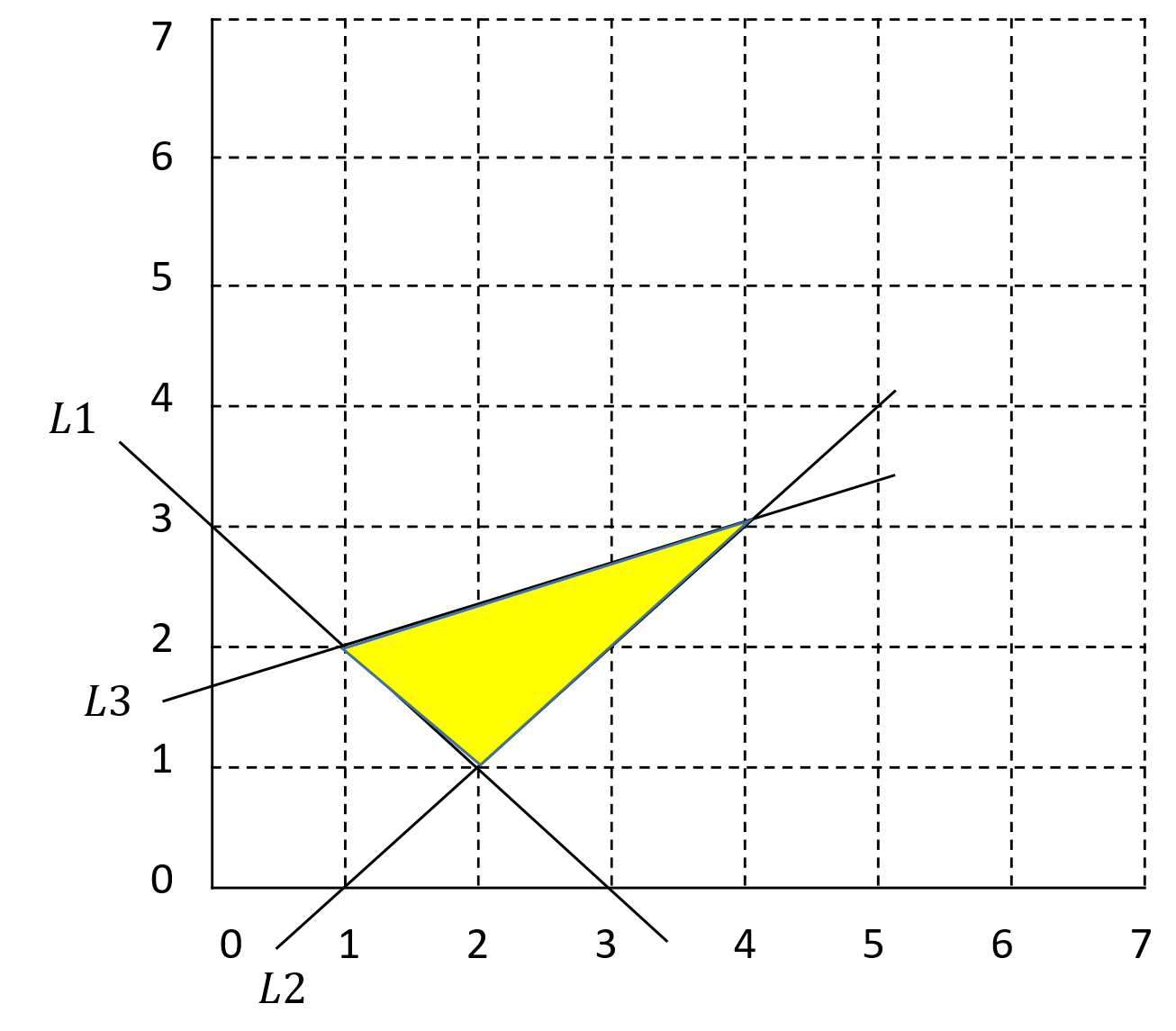
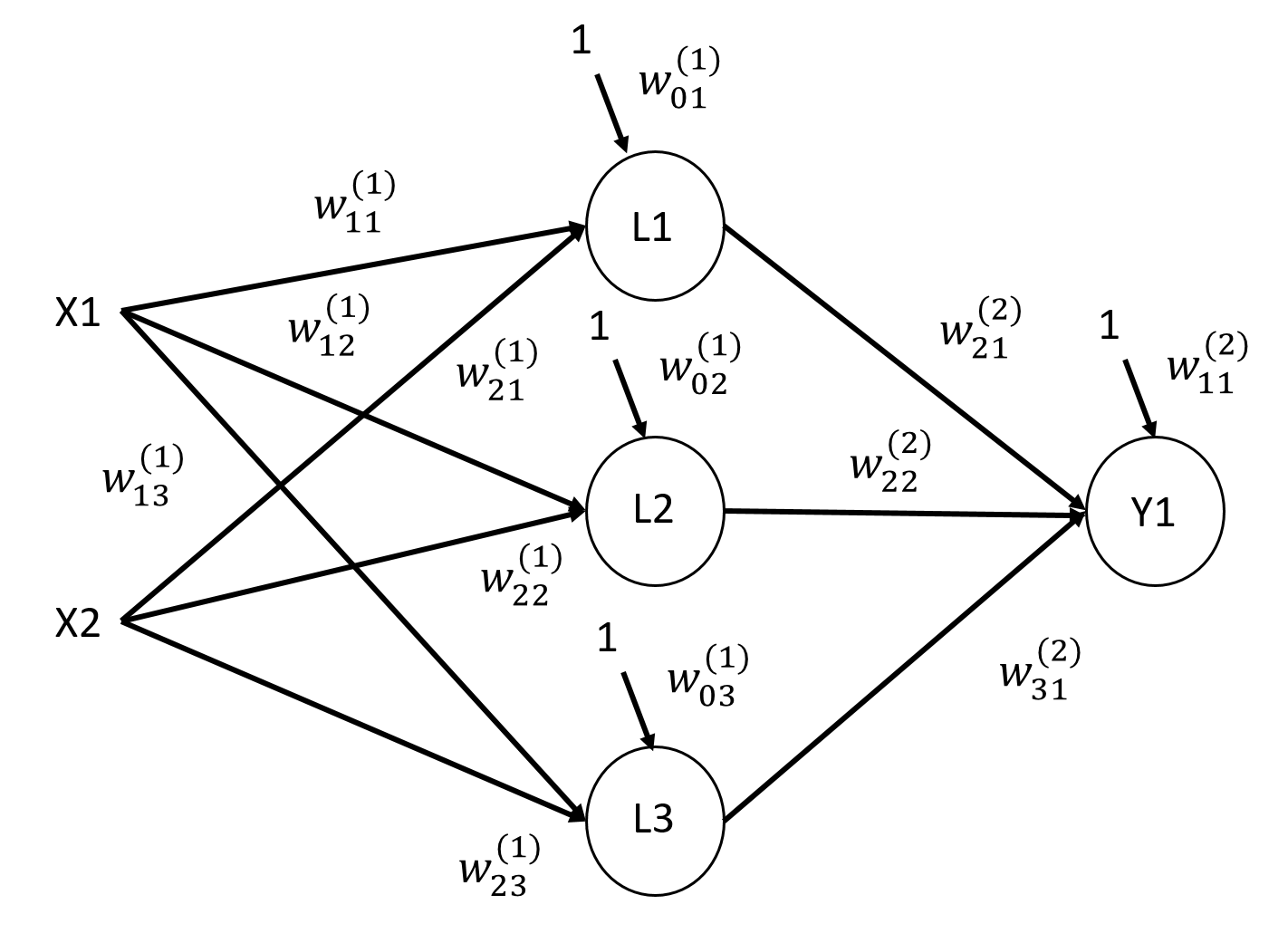


4. [3 pts] Explain why deep layer has much smaller # of parameter than shallow one using the following formula.

# parameters ≈ ⋅ depth

5. (Refer to p. 6-7 in the slides) Universal Approximation Theorem (for classification)

It is known that a neural network with one (or two hidden) layers can represent any classification boundary. Refer to the following classification boundary and a two layer neural network.

(a) (b)

The yellow region in figure (a) is a classification boundary, and we are going to represent the boundary using two layers neural network in figure (b). In other words, if a data point (x1, x2) falls within the yellow region, its target value is 1, 0 otherwise.

In figure (b), each (hidden/output) node is a perceptron using a step function. L1, L2 and L3 represents the linear boundary in figure (a), respectively. For example, if input data lies above L1 line, its output is 1, otherwise 0. If it lies above L2 line, output is 1, otherwise 0. if it lies below L3 line, output is 1, otherwise 0.

1) [3 pts] Show the formula of lines L1, L2, and L3, respectively.

2) [3 pts/ea] Define all parameter values (weights and activation function) of perceptron L1, L2 and L3, respectively. Explain how each perceptron works using these parameters. You have to explain using these parameters in detail.

L1:

L2:

L3:

3) [5 pts] Define all parameter values (weights and activation function) of perceptron Y1. Explain how Y1 is able to find the boundary region.

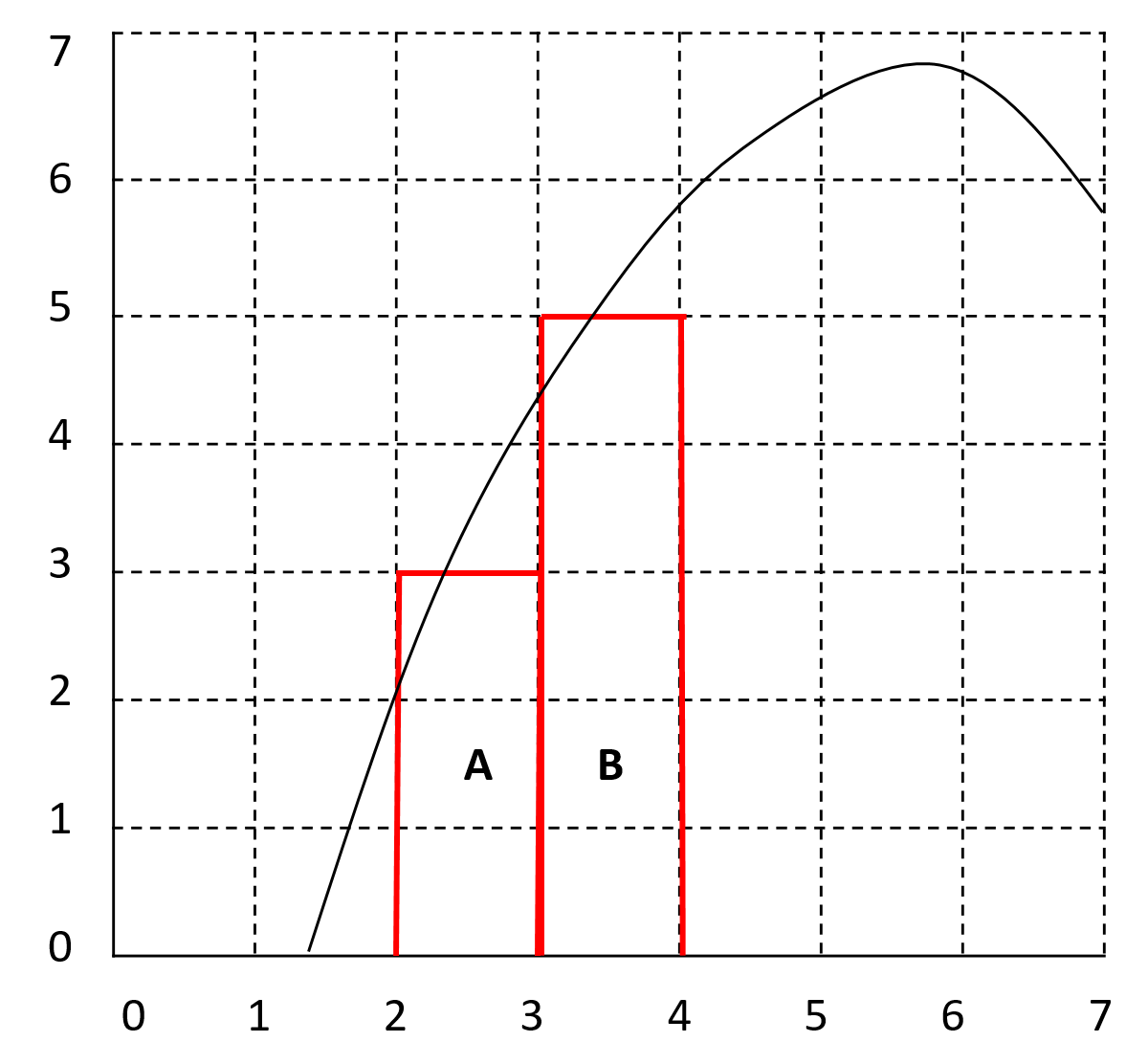
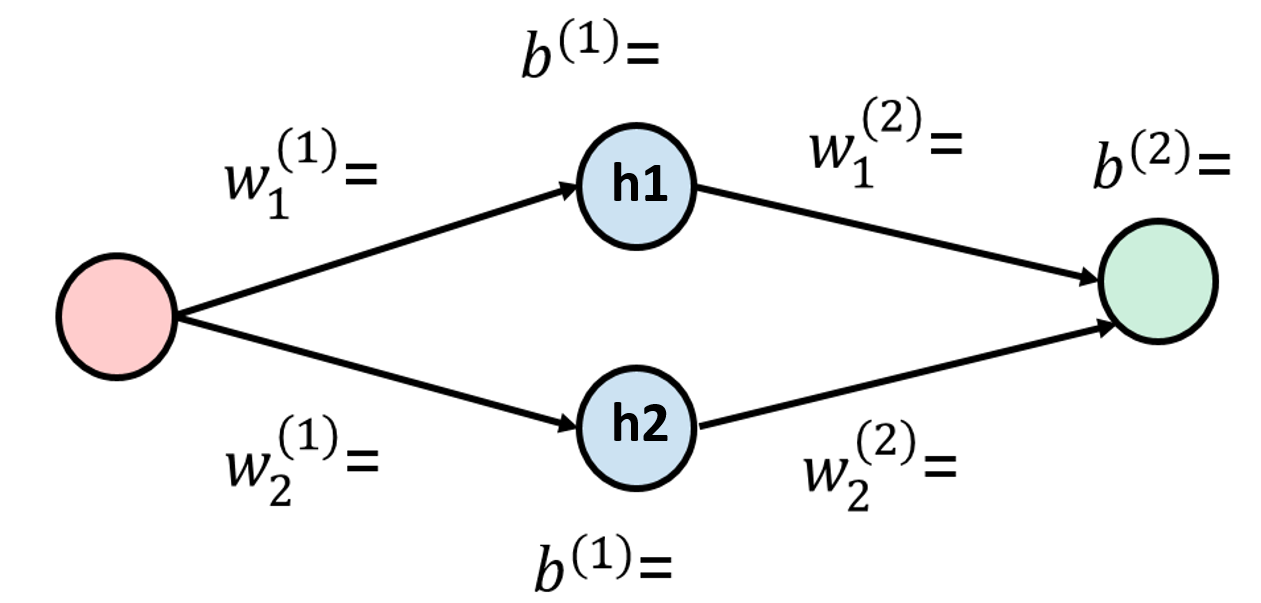
4) [5 pts] Show the classification values of input (x1=2, x2=2) and (x1=4, x2=1), respectively. Show the derivations.

6. (Refer to p. 11-13 in the slides) Universal Approximation Theorem (Regression)

It is also known that a neural network with one layer can represent any continuous (regression) functions.

Refer to the following function and a one layer neural network. Activation function of hidden nodes is sigmoid and output node uses linear activation function.

We want to approximate the function in figure (a) using red bars (e.g., A and B).

(a) (b)

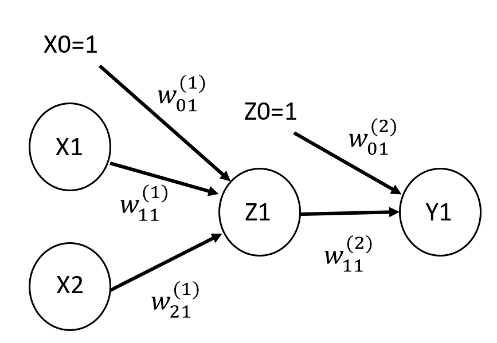
1) [4 pts/ea] Now we want to implement the red bar “A” using a neural network in figure (b). Show all the parameters of hidden nodes h1 & h2 (e.g., the weights, bias, slope of curve), respectively. Show the exact shape of output of each hidden node.

h1:

h2:

2) [4 pts] Show all the parameters of output node. Explain in detail how the function is implemented using this network.

3) [4 pts] Show the output value of network given an input 2.5.

7. Backpropagation algorithm

With the following neural network, we perform backpropagation using stochastic gradient descent.

Input data are x = [ [1 0] [0 1] [1 1] ] and its corresponding target y=[0 1 1]. All weight values are initialized to 0.1 and learning rate is 0.2.

We use sum of squared error (not mean of squared error) as the error function(). Each node in hidden/output has two functions and (refer to p. 22 in the slides) and (activation function) is the sigmoid function.

1) [6 pts] Given x = [1 0] and y = 0, Compute the output of Z1 and Y1. Show the derivations.

2) [4 pts] Show the chain rule formula of

3) [6 pts] Using the value of Z1 and Y1, compute (refer to p. 26)

4) [4 pts] Update

5) [Bonus 8 pts] Repeat above process for updating

8. Given the image and filter

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **2** | **4** | **5** | 9 | 7 |
| **6** | **3** | **1** | 4 | 6 |
| **4** | **2** | **7** | 7 | 2 |
| 6 | 5 | 9 | 9 | 8 |
| 8 | 4 | 3 | 6 | 5 |

|  |  |  |
| --- | --- | --- |
| 2 | 3 | 1 |
| 1 | 0 | 4 |
| 4 | 8 | 4 |

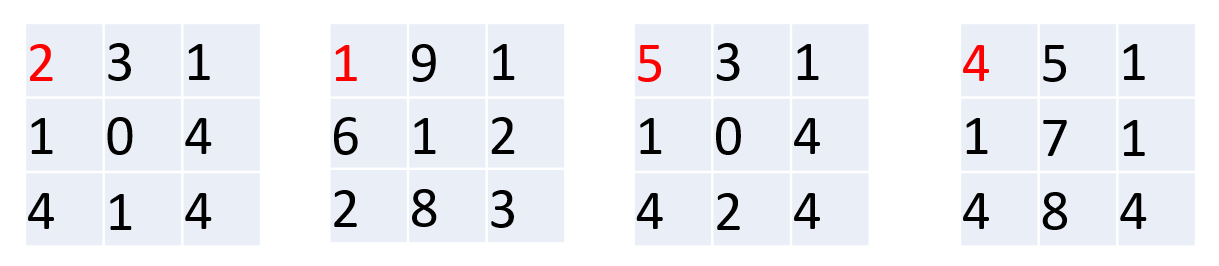
1) [4 pts] Show the output of feature map node using red region and filter (bias=0) using ReLU activation function

2) [3 pts] Compute the size of feature map. (stride=2, no padding).

3) [3 pts] When there are 10 feature maps, what is the total number of parameters?

9. We have the following multi-channel data.

channel 1 channel 2 channel 3 channel 4



1) [3 pts] Show an example of 1X1 convolution filter.

2) [3 pts] compute the output of upper left corner (red) position using above 1X1 filter.

3) [3 pts] What is the advantage of using 1X1 convolution. Compare it with traditional Convolutional filter.

10. [5 pts] In convolutional layer, one issue is to determine the proper filter size. Can we find the proper filter size automatically in CNN? Explain one possible way of computing filter size automatically.

11. Suppose you developed a CNN network that successfully identifies the images of ‘dog’ and ‘cat’. Now we want to make the neural network classify the images of ‘duck’ as well.

1) [3 pts] Explain the approach of building the CNN from scratch.

2) [6 pts] Now in transfer learning approach:

2-1) Explain source domain and target domain in this approach.

2-3) Explain the process of applying transfer learning in this task.

3) [3 pts] Explain whether approach in 2) is better than approach 1).

**\* For coding assignments:**

1) Don’t change the basic program code (unless you have my permission).

2) Never use any ready-made library to implement algorithms (except for bonus questions)

3) Don’t put entire program in one cell in Jupyter. Instead, for each sub questions, show the corresponding program code in .ipynb and explain it.

4) Don’t copy from other sources

5) HWs violating these guidelines will get zero.

12. (coding) perceptron

# For each subquestion, show the program code and explain it in .ipynb

import numpy as np

import matplotlib.pyplot as plt

import random

import pandas as pd

#import dataset

df = pd.read\_csv('iris.csv', header=None)

# select the first 100 rows with only the first 2 features (petal length & width)

# X=[[first feature value of 1st data & second feature value of 1st data] […..]]

# y=[target values]

# y value is -1 if target=setosa, y=-1, otherwise 1

# e.g.,

# X=[[5.1 1.4] [5.1 3. ] … [5.7 4.1]]

# y=[-1 -1 … -1 1 … 1]

#

‘’’

[2 pts]

YOUR WORK HERE: 12-1)

# do the following:

# create X and y as above

# show the first 5 contents of X and y

‘’’

class Perceptron():

# initialize learning rate and number of iterations

def \_\_init\_\_(self, lrate=0.1, no\_iter=50):

self.lrate = lrate

self.no\_iter = no\_iter

# using X & y, update ww parameters

def fit(self, X, y):

# initialze weights ww to random value (-1,1)

self.ww = [random.uniform(-1.0, 1.0) for \_ in range(1+X.shape[1])] #randomly initialize weights

# initilize list\_error=[]

# keeps track of the errors per iteration for graph plotting

self.list\_errors = []

#iterate over labelled dataset updating weights for each features accordingly

for \_ in range(self.no\_iter):

cur\_error = 0

for xx, label in zip(X, y):

‘’’

[8 pts]

YOUR WORK HERE: 12-2)

# update weights using *wi* ← *wi* + Δ*wi* where Δ*wi* = η(*ti* – *oi*)*xi*

# Refer to p. 14-15

# X: parameter array, y: label

#

# compute delta Δ*wi*

# update self.ww

# compute cur\_error. cur\_error is 0 if delta=0, 1 otherwise

# append cur\_error into list\_errors

‘’’

return self

#compute the net input i.e sum of X and the weights plus the bias value

def net\_input(self, X):

return np.dot(X, self.ww[1:]) + self.ww[0]

#predict a classification for a sample of features X

def predict(self, X):

‘’’

[6 pts]

YOUR WORK HERE: 12-3)

# do the following

# implement step function of perceptron

# output = 1 if ∑i=0 wixi >0 , = -1 otherwise

# return 1 or -1 depending on ∑i=0 wixi

‘’’

# create a new perceptron & initialize parameters

# you can change no\_iter

# train perceptron using data X, y

model = Perceptron(no\_iter=10)

model.fit(X, y)

‘’’

[4 pts]

YOUR WORK HERE: 12-3)

# plot errors against number of iterations using list\_errors

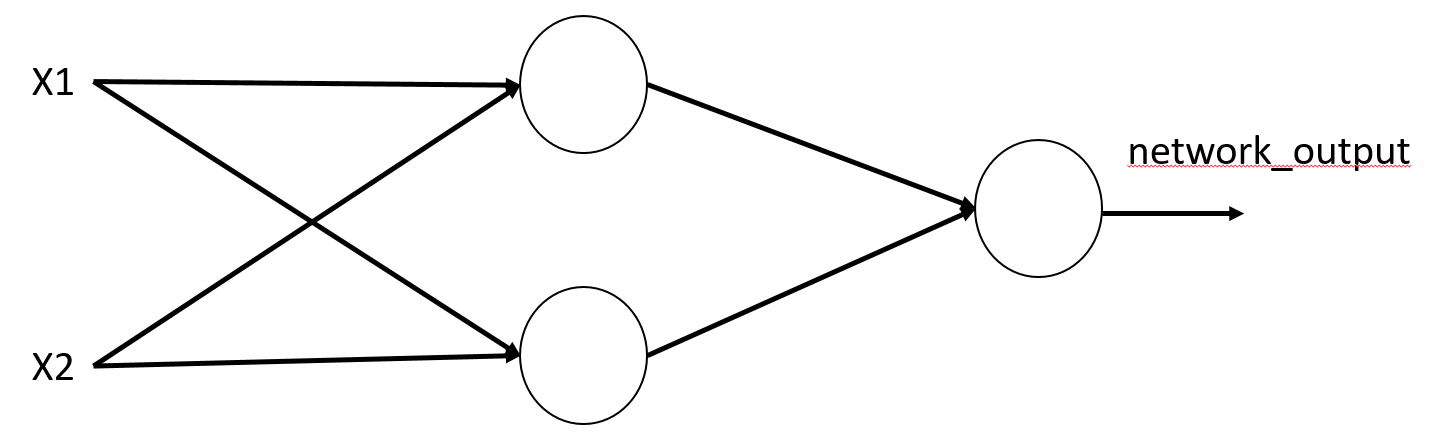
‘’’

13. (coding) Backpropagation

# Don’t change the program code.

# For each subquestion, complete the code and explain it.

We have the following neural network.



Both hidden nodes and output node uses sigmoid function.

**(Program code)**

import numpy as np

def forward(X):

# Feedforward

# we assume there is a dummy input X0. Therefore, no need of bias notation.

hidden\_input = np.dot(X, hidden\_weights)

hidden\_output = sigmoid(hidden\_input)

output\_input = np.dot(hidden\_output, output\_weights)

output\_output = sigmoid(output\_input)

return hidden\_output, output\_output

# Activation function: Sigmoid

def sigmoid(x):

‘’’

[2 pts]

YOUR WORK HERE 13-1)

# compute the value of sigmoid function and return it

‘’’

# Derivative of Sigmoid

def sigmoid\_derivative(x):

‘’’

[2 pts]

YOUR WORK HERE 13-2)

# compute the derivative of sigmoid function and return it

‘’’

# XOR dataset is used as a training data

‘’’

[2 pts]

YOUR WORK HERE 13-3)

Instead of using bias, we assume there a dummy variable x0=1.

1) Change the following X data accordingly (in green)

2) change input\_nodes values (in green)

‘’’

X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]]) # Input features (x1, x2)

Y = np.array([[0], [1], [1], [0]]) # Target values

#

# Construct a Neural Network

#

# number of nodes in each layer

input\_nodes = 2 # Number of input nodes

hidden\_nodes = 2 # Number of hidden nodes

output\_nodes = 1 # Number of output nodes

# Randomly initialize weights and biases between (0,1)

# hidden\_weights: weights connecting input layer nodes to hidden layer nodes

# output\_weights: weights connecting hidden layer nodes to output node

hidden\_weights = np.random.uniform(size=(input\_nodes, hidden\_nodes))

output\_weights = np.random.uniform(size=(hidden\_nodes, output\_nodes))

# Learning rate. You can change this

lrate = 0.1

# Number of iteration. You can change this

no\_iter = 10000

# Backpropagation training

for iter in range(no\_iter):

# forward step

# hidden\_output: outputs of hidden layer nodes

# output\_output: output of output layer node

hidden\_output, output\_output = forward(X)

# Compute error

error = Y - output\_output

# Backpropagation

‘’’

[8 pts]

YOUR WORK HERE 13-4)

# Refer to p. 26

# compute error

# implement the following for output weight update

‘’’

‘’’

[10 pts]

YOUR WORK HERE 13-5)

# Refer to p. 30

# since we have only one output node k=1 from the formula in p. 30

# implement the following for hidden weight update

‘’’

# Print error every 100 iter for monitoring

if iter % 100 == 0:

print(f"Epoch {iter} Error is {np.mean(np.abs(error))}")

# predict using training data

\_, nn\_output = forward(X)

print(nn\_output)

14. [Bonus 12 pts] Implement Question 13 with two output nodes (e.g., output\_nodes = 2).

Each output node represents the target value 0 and 1, respectively.