CS5691: PRML Assignment #2

Instructor : Prof. B. Ravindran
Topics: ANN,Ensemble Method,Kernel and SVM. **Deadline:** 12th November 2021

Teammate 1: Sanjanaa G V **Roll number:** CS21M057
Teammate 2: Gudivada Harsha Vardhan **Roll number:** CS21M021

- This assignment has to be completed in teams of 2. Collaborations outside the team are strictly prohibited.
- Be precise with your explanations. Unnecessary verbosity will be penalized.
- Check the Moodle discussion forums regularly for updates regarding the assignment.
- Type your solutions in the provided LaTeX template file.
- We highly recommend using `Python 3.6+` and standard libraries like `numpy, Matplotlib, pandas`. You can choose to use your favourite programming language however the TAs will only be able to assist you with doubts related to Python.
- You are supposed to write your own algorithms, any library functions which implement these directly are strictly off the table. Using them will result in a straight zero on coding questions, `import` wisely!
- **Please start early and clear all doubts ASAP.**
- Please note that the TAs will **only** clarify doubts regarding problem statements. The TAs won't discuss any prospective solution or verify your solution or give hints.
- Post your doubt only on Moodle so everyone is on the same page.
- Posting doubts on Moodle that reveals the answer or gives hints may lead to penalty
- **Only one team member will submit the solution**
- For coding questions paste the link to your Colab Notebook of your code in the LaTeX solutions file as well as embed the result figures in your LaTeX solutions. Make sure no one other than TAs have access to the notebook. And do not delete the outputs from notebook before final submission . Any update made to notebook after deadline will result in standard late submission penalty.
- Late submission per day will attract a penalty of 10 percent of the total marks.

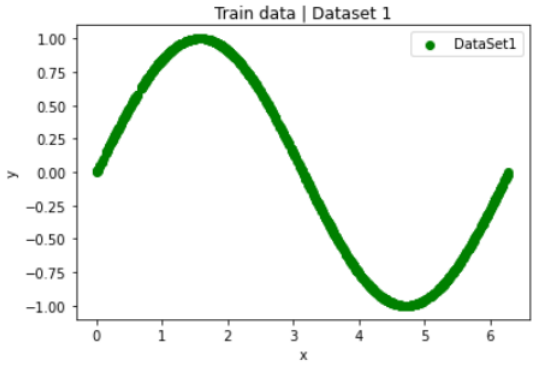
1. **[ANN]** In this Question, you will code a single layer ANN with Sigmoid Activation function and appropriate loss function from scratch. Train the ANN for the Dataset1 and Dataset2

   NOTE: Test Data should not be used for training.

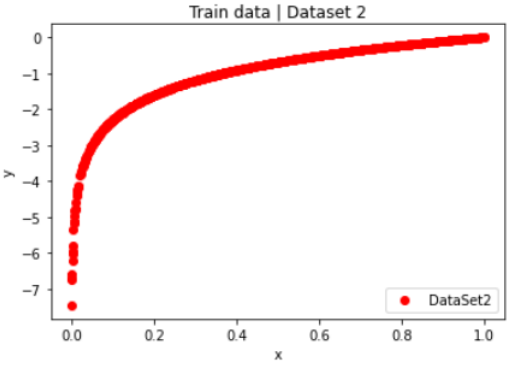
   NOTE 2: You need to code from scratch.

(a) (1 mark) Plot the training Data for Dataset1 and Dataset2.

Solution: The training data on the first dataset is as follows:



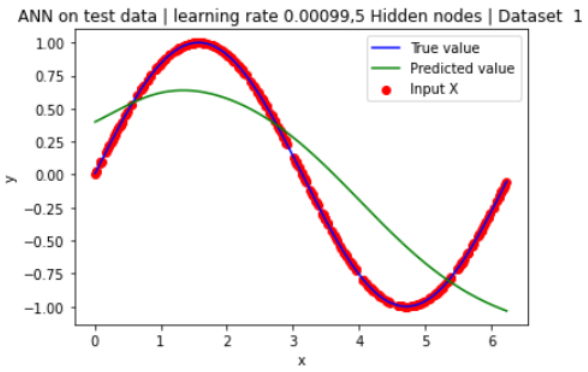
The training data on the second dataset is as follows:



(b) (1 mark) **For data set 1** : (1) Write the number of nodes in the hidden layer and learning rate used (2) Plot Test Data and prediction on Test Data in the same graph with different colors and appropriate legend.

Solution: (1) Hyperparameters used for dataset 1:
Learning rate: 0.00099
Number of hidden Nodes: 5

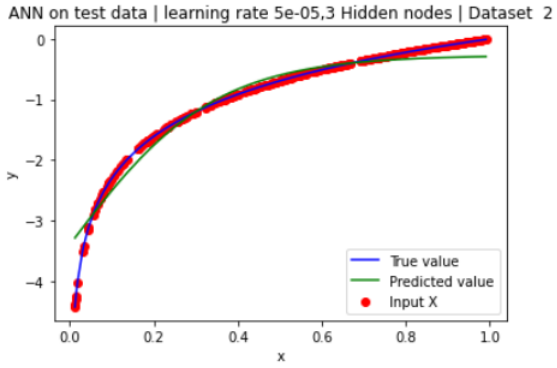
(2) Test Data and prediction on test data for the dataset 1 is as follows:



(c) (1 mark) **For data set 2** : (1) Write the number of nodes in the hidden layer and learning rate used (2) Plot Test Data and prediction on Test Data in the same graph with different colors and appropriate legend.

Solution: (1) Hyperparameters used for dataset 2:
Learning rate: 0.00005
Number of hidden Nodes: 3

(2) Test Data and prediction on test data for the dataset 2 is as follows:



(d) (1 mark) For each Dataset write average training Loss and average Test Loss.

Solution: For Dataset 1:
- Training loss : 0.157475
- Test loss : 0.141293

For Dataset 2:
- Training loss : 0.126865
- Test loss : 0.0342839

(e) (1 mark) What Loss function did you use and why?

Solution: We have used standard mean square loss to calculate the training and test loss. This is because, we are given with a regression problem and mean square error is a common loss function for regression problems as mean square error tells us how close a regression line is to a set of points. It does this by taking the distances from the points to the regression line (these distances are the "errors") and squaring them. The squaring is necessary to remove any negative signs. It also gives more weight to larger differences. The lower the MSE, the better the forecast.

$$MSE = \frac{1}{n}\sum(y_{true} - y_{pred})^2$$

where n is the number of training examples.

(f) (3 marks) Paste the link to your Colab Notebook of your code. Make sure that your notebook is private and give access to all the TAs. Your Notebook must contain all the codes that you used to generate the above results. **Note :** Do not delete the outputs.

Solution: Click here to view the Colab Notebook of the code: Colab Notebook

2. [**AdaBoost**] In this question, you will code the AdaBoost algorithm. Follow the instructions in this Jupyter Notebook for this question. Find the dataset for the question here.

   NOTE: Test data should not be used for training.

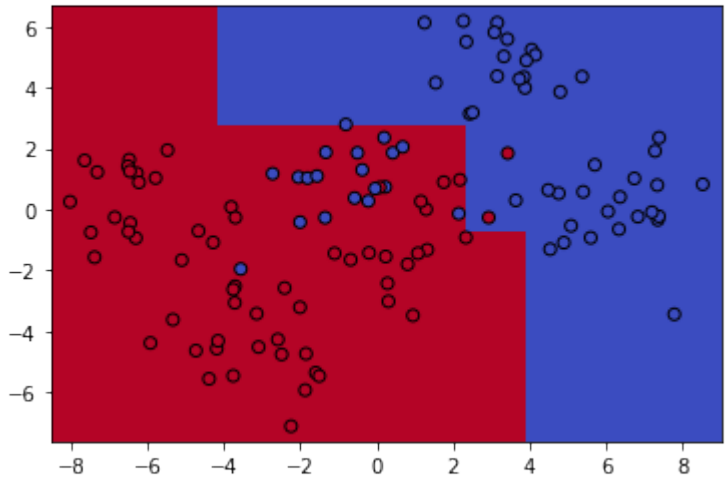
   NOTE 2: You need to code from scratch. You can use the starter notebook though :) . Make a copy of it in your drive and start.

   (a) (1 mark) Plot the training data.

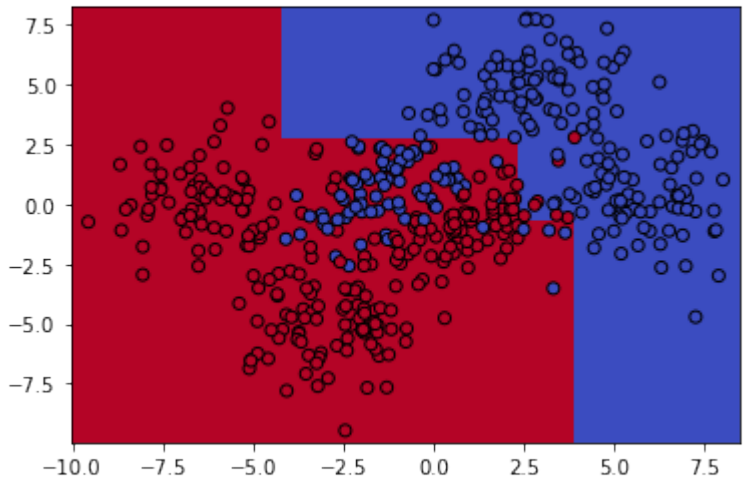
Solution:

(b) (1 mark) **For training with k=5** : (1) Plot the learnt decision surface. (2) Write down the test accuracy.

Solution: 1 (a). Learnt decision surface on test data:



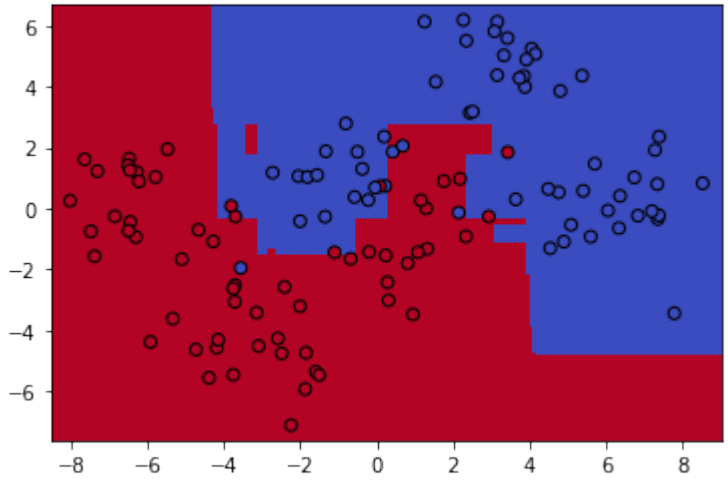
1 (b). Learnt decision surface on train data:



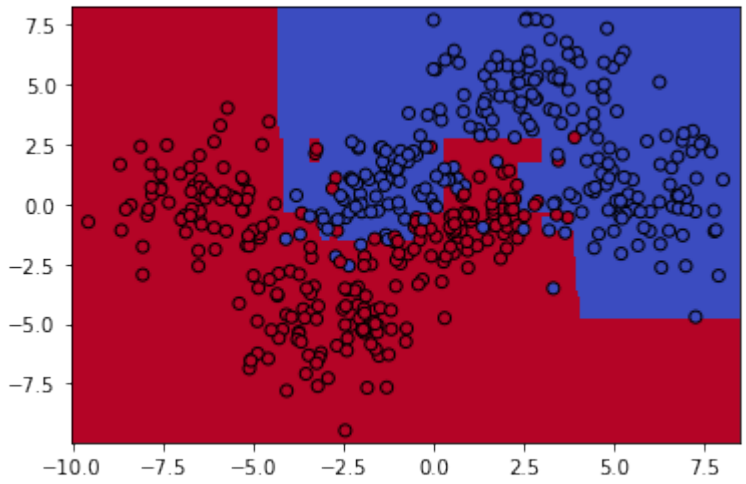
2. Test accuracy: 83.33333333333333
Test Accuracy is defined as the percentage of correct predictions for the test data. It can be calculated by dividing the number of correct predictions by the number of total predictions and the result multiplied by 100 to get the percentage.

(c) (1 mark) **For training with k=100** : (1) Plot the learnt decision surface. (2) Write down the test accuracy.

Solution: 1(a). Learnt decision surface on test data:



1(b). Learnt decision surface on train data:



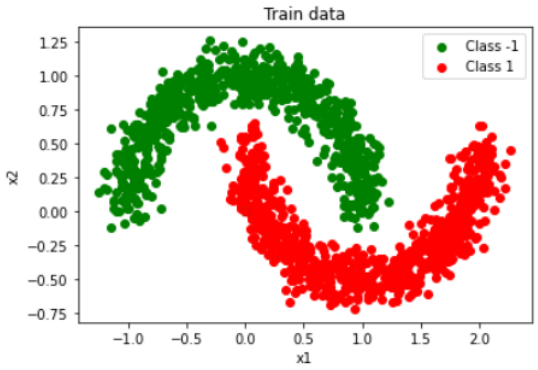
2. Test accuracy: 91.66666666666667
Test Accuracy is defined as the percentage of correct predictions for the test data. It can be calculated by dividing the number of correct predictions by the number of total predictions and the result multiplied by 100 to get the percentage.

(d) (3 marks) Paste the link to your Colab Notebook of your code. Make sure that your notebook is private and give access to all the TAs. Your notebook must contain all the codes that you used to generate the above results. **Note :** Do not delete the outputs.

Solution: Click here to view the Colab Notebook of the code: Colab Notebook

3. **[Kernel]** Consider *Dataset_Kernel_Train.npy* and *Dataset_Kernel_Test.npy* for this question. Each row in the above matrices corresponds to a labelled data point where the first two entries correspond to its x and y co-ordinate, and the third entry $\in \{-1, 1\}$ indicates the class to which it belongs. Find the dataset for the question here. **NOTE: Test data should not be used for training.**

(a) (1 mark) Plot the training data points and indicate by different colours the points belonging to the different classes. Is the data linearly separable?



Solution:
The above figure shows the plot of the training data points. Green points indicate class -1 and red points indicate class 1. As seen above the data is **not linearly separable.**

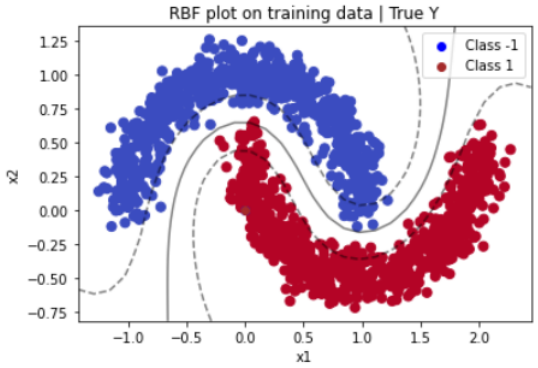
(b) (1 mark) Using *sklearn.svm* (read the documentation here), bulid a classifier that classifies the data points in the testing data set using the Radial Basis Function (RBF) kernel.How do you tune the involved hyperparameters?

Solution: For a radial basis function kernel, the parameters C and $gamma$ of the function sklearn.svm.svc() can be varied to tune the hyperparameters. C is the parameter used to tune the Regularization parameter, trades off the misclassification of training examples against simplicity of the decision surface. The strength of the regularization is inversely proportional to C. $gamma$ is the kernel coefficient. If $gamma =' scale'$ (default) is passed then it uses $1/(n_features * X.var())$ as value of gamma and if 'auto', it uses $1/n_features$. The larger gamma is, the closer other examples must be to be affected.

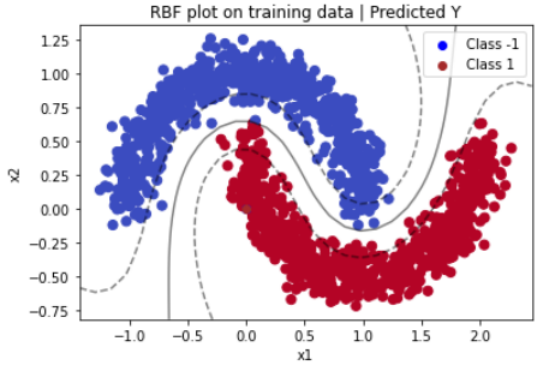
Reference: https://scikitlearn.orgstablemodulessvm.html#svm-kernels

(c) (1 mark) Plot the separating curve and report the accuracy of prediction corresponding to the tuned hyperparameters.

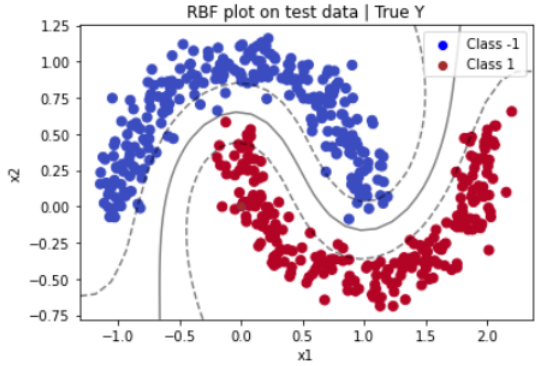
Solution: Tuned hyperparameters: C = 1, gamma = 'scale'.
Plot on train data with the true y:



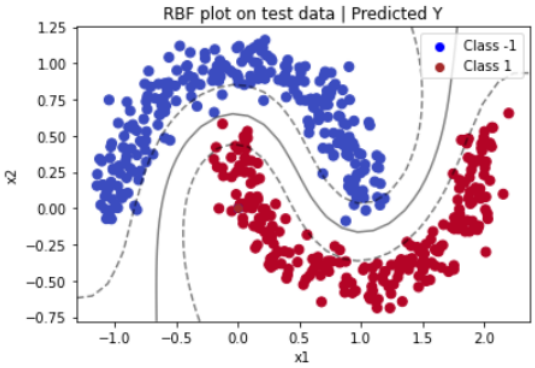
Plot on train data with the predicted y:



Plot on test data with the true y:



Plot on test data with the predicted y:



Accuracy of test dataset = 1.
Accuracy of train dataset = 0.9993.
(Formula used for calculating Accuracy = $\frac{Number\ \ of\ \ correct\ \ predictions}{Total\ \ number\ \ of\ \ predictions}$)

(d) (3 marks) Paste the link to your Colab Notebook of your code. Make sure that your notebook is private and give access to all the TAs. Your notebook must contain all the codes that you used to generate the above results. **Note :** Do not delete the outputs.

Solution: Click here to view the Colab Notebook of the code: Colab Notebook

4. (2 marks) **[Ensemble of randomised algorithms]** Imagine we have an algorithm for solving some decision problem (*e.g.*, is a given number p a prime?). Suppose that the algorithm makes a decision at random and returns the correct answer with probability $\frac{1}{2} + \delta$, for some $\delta > 0$, which is just a bit better than a random guess. To improve the performance, we run the algorithm N times and take the majority vote. Show that for any $\epsilon \in (0, 1)$, the answer is correct with probability $1 - \epsilon$, as long as $N > (1/2)\delta^{-2} \ln(\epsilon^{-1})$.

Hint 1: Try to calculate the probability with which the answer is not correct i.e. when the majority votes are not correct.

Hint 2: What value of N will you require so that the above probability is less than ϵ. Rearrange Inequalities :-)

Solution: We want a lower bound on N when the answer is correct with probability at least $1 - \epsilon$. This is equivalent to the lower bound on N when the probability with which the answer is not correct is less than ϵ. We will use this second approach because there are many concentration inequalities which can help us in finding tight upper bounds on probabilities.
As this is a majority vote algorithm, for the algorithm to give final wrong answer, we need more than half runs of the algorithm each to give wrong answer. So, let us consider X_i as an indicator random variable for the event that the algorithm returns wrong answer.

$$X_i = \begin{cases} 1 & \text{if algorithm returns wrong answer} \\ 0 & \text{otherwise} \end{cases}$$

So, we want a lower bound on N such that:

$P\{\sum_{i=1}^{N} X_i > \frac{N}{2}\} < \epsilon$

To bring this inequality closer to the popular concentration inequalities, let us subtract the expectation of the random variable on either side of the inequality inside the probability.

We know that the expectation of an indicator random variable is just the probability of the event. It is given that the probability of the correct answer is $\frac{1}{2} + \delta$. So, the probability of the incorrect answer will be $\frac{1}{2} - \delta$. This will be the expectation to be subtracted on both sides of the inequality inside the probability.

$P\{\sum_{i=1}^{N}(X_i - (\frac{1}{2} - \delta)) > \frac{N}{2} - \sum_{i=1}^{N}(\frac{1}{2} - \delta)\} < \epsilon$

$P\{\sum_{i=1}^{N}(X_i - (\frac{1}{2} - \delta)) > \frac{N}{2} - \frac{N}{2} + \delta N\} < \epsilon$

$$P\{\sum_{i=1}^{N}(X_i - (\frac{1}{2} - \delta)) > \delta N\} < \epsilon \tag{1}$$

Expression on the left side of the outer inequality is similar to Hoeffding's inequality. There are multiple versions of Hoeffding's inequality. However, to fit in this context, we will use Hoeffding's inequality for general bounded random variables and stated in terms of sum which is as follows:

$P\{\sum_{i=1}^{N}(X_i - E(X_i)) \geq t\} \leq e^{\frac{-2t^2}{\sum_{i=1}^{N}(M_i - m_i)^2}}$

In our context, t will be δN. And $X_i \epsilon [M_i, m_i]$. So, in this context, $M_i = 1, m_i = 0$ as X_i is an indicator random variable. So,$\sum_{i=1}^{N}(M_i - m_i)^2$ will be equal to N. And as discussed earlier, the expectation of the indicator random variable is the probability of the event. So, after substitution, the Hoeffding's inequality can be simplified as:

$P\{\sum_{i=1}^{N}(X_i - (\frac{1}{2} - \delta)) \geq \delta N\} \leq e^{\frac{-2(\delta N)^2}{N}}$

$$P\{\sum_{i=1}^{N}(X_i - (\frac{1}{2} - \delta)) \geq \delta N\} \leq e^{-2\delta^2 N} \tag{2}$$

Observing equation(1) and equation (2), if we can can get an lower bound on N for the inequality $e^{-2\delta^2 N} < \epsilon$, then, from the transitive property of inequalities, we can get a lower bound on N in equation(1).So,

$$e^{-2\delta^2 N} < \epsilon \tag{3}$$

$\implies -2\delta^2 N < \ln \epsilon$

$\implies 2\delta^2 N > -\ln \epsilon$

$\implies N > (\frac{1}{2})\delta^{-2} \ln(\epsilon^{-1})$

Thus, using the transitive property of inequalities on equations (1), (2), (3), we got a lower bound on N in the equation (1)

And equation (1) is the probability with which the answer is not correct is less than ϵ. And this is equivalent to when the answer is correct with probability at least $1-\epsilon$. So, the lower bound of $N : N > (\frac{1}{2})\delta^{-2} \ln(\epsilon^{-1})$ applies here too.

5. (2 marks) **[Boosting]** Consider an additive ensemble model of the form $f_m(\mathbf{x}) = \frac{1}{2}\sum_{l=1}^{m} \alpha_l y_l(\mathbf{x})$, where y_l's are individual models and α_l's are weights. Show that the sequential minimization of the sum-of-squares error function for the above model trained in the style of boosting (i.e. y_m is trained after accounting the weaknesses of f_{m-1}) simply involves fitting each new base classifier y_m to the residual errors $t_n - f_{m-1}(\mathbf{x}_n)$ from previous model.

Solution: Sum-of-squares error function is the sum of the squares of the errors between the predictions $y(x_n, w)$ for each data point x_n and the corresponding target value. It is:

$E(w) = \frac{1}{2}\sum_{n=1}^{N}\{y(x_n, w) - t_n\}^2$

Here, in this model, $y(x_n, w) = f_m(x_n) = \frac{1}{2}\sum_{l=1}^{m} \alpha_l y_l(x_n)$

$E = \frac{1}{2}\sum_{n=1}^{N}\{f_m(x_n) - t_n\}^2$

As $f_m(x_n)$ is an additive ensemble model of the form $f_m(x_n) = \frac{1}{2}\sum_{l=1}^{m} \alpha_l y_l(x_n)$, the error function can be also written as:

$$E_1 = \frac{1}{2}\sum_{n=1}^{N}\{f_{m-1}(x_n) + \frac{1}{2}\alpha_m y_m(x_n) - t_n\}^2 \tag{4}$$

We need to show the sequential minimization of the function in above equation (1) is equal to fitting each new base classifier y_m to the residual errors $t_n - f_{m-1}(\mathbf{x}_n)$ from previous model.

Fitting each new base classifier y_m to the residual errors $t_n - f_{m-1}(\mathbf{x}_n)$ from previous model

is nothing but reducing/minimizing the difference between the residual errors and the new base classifier's output.

Residual errors from previous model is given by $t_n - f_{m-1}(\mathbf{x}_n)$

Base classifier's output is $\frac{1}{2}\alpha_m y_m(x_n)$

Hence, the error function of this second part - involving difference between the residual errors and the new base classifier's output, which later need to be minimized can be written as:

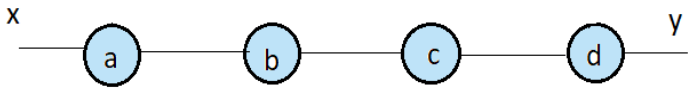
$$E_2 = \frac{1}{2}\sum_{n=1}^{N}\{(t_n - f_{m-1}(x_n)) - \frac{1}{2}\alpha_m y_m(x_n)\}^2 \tag{5}$$

We need to prove that minimization of the functions in equation (1) and equation (2) are equivalent. This can be shown if we show that functions in equation (1) and equation (2) are equal. Consider equation (1). As the term inside the summation is a square, we can multiply all the terms with -1, with no change in the value. After multiplying all the terms with -1 and rearranging the terms, we get the equation (1) as:

$$E_1 = \frac{1}{2}\sum_{n=1}^{N}\{t_n - f_{m-1}(x_n) - \frac{1}{2}\alpha_m y_m(x_n)\}^2 \tag{6}$$

Hence, we show that as these both error functions are equal, their minimization are equivalent. Finally, minimization of the sum-of-squares error function for the above model trained in the style of boosting simply involves fitting each new base classifier to the residual errors from previous model.

6. (2 marks) **[Backpropagation]** We are trying to train the following chain like neural network with back-propagation. Assume that the transfer functions are sigmoid activation functions i.e. $g(x) = \frac{1}{1+e^{-x}}$. Let the input $x = 0.5$, the target output $y = 1$, all the weights are initially set to 1 and the bias for each node is -0.5.



(a) Give an expression that compares the magnitudes of the gradient updates for weights (δ) across the consecutive nodes.
(b) How does the magnitude of the gradient update vary across the network/chain as we move away from the output unit?

Solution: (a) Expression that compares the magnitudes of the gradient updates for weights (δ) on the hidden units across consecutive nodes is given by,

$$\delta_j = \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k} \frac{\partial a_k}{\partial a_j}$$

where the unit k is adjacent to unit j while traversing from kth unit to jth unit in the back propagation, a_j is the feed forward weighted sum of its inputs,

$$a_j = \sum_i w_{ji} z_i$$

where w_{ji} is the weight between the ith and jth nodes and z_i is the activation of a unit that sends a connection to unit j,

$$z_j = h(a_j)$$

where h() is the nonlinear activation function.
Making use of the above equations, the backpropagation formula for the hidden layers can be given as,

$$\delta_j = h'(a_j) \sum_k w_{kj} \delta_k$$

However, for the output units,

$$\delta_k = y_k - t_k$$

where δ is the error signal at the output, t_k is the true value of the kth output and y_k is the predicted value at the same unit.

(b)Calculating Backpropagation along the network:
(i) Forward propagation:
Let a represent the linear combination between the weights and the biases,

$$a_i = w_i * z_{i-1} + b_i \; and \; a_0 = x$$

where z_{i-1} is the activation of a unit that sends a connection to unit i, w_i is the weight associated with this connection and b_i is the bias associated with it.

$$z_i = h(a_i)$$

Given, input $x = 0.5 \implies a_0 = 0.5$
Initially,
$w_1 = w_2 = w_3 = w_4 = w_5 = 1$
$b_1 = b_2 = b_3 = b_4 = -0.5$
We are given that the hidden layers have a sigmoid activation,

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

Performing feed forward propagation,
- between input and node 'a':

$$a_1 = a_0 w_1 + b_1 = 0.5 - 0.5 = 0$$

$$z_1 = \sigma(a_1) = \frac{1}{1 + e^{-a_1}} = \frac{1}{1 + e^0} = \frac{1}{2} = 0.5$$

- between node 'a' and node 'b':

$$a_2 = a_1 w_2 + b_2 = 0.5 - 0.5 = 0$$

$$z_2 = \sigma(a_2) = \frac{1}{1 + e^{-a_2}} = \frac{1}{1 + e^0} = \frac{1}{2} = 0.5$$

- between node 'b' and node 'c':

$$a_3 = a_2 w_3 + b_3 = 0.5 - 0.5 = 0$$

$$z_3 = \sigma(a_3) = \frac{1}{1 + e^{-a_3}} = \frac{1}{1 + e^0} = \frac{1}{2} = 0.5$$

- between node 'c' and node 'd':

$$a_4 = a_3 w_4 + b_4 = 0.5 - 0.5 = 0$$

$$z_4 = \sigma(a_4) = \frac{1}{1 + e^{-a_4}} = \frac{1}{1 + e^0} = \frac{1}{2} = 0.5$$

- between node 'd' and node 'y':

$$a_5 = a_4 w_5 + b_5 = 0.5 - 0.5 = 0$$

$$z_5 = \sigma(a_5) = \frac{1}{1 + e^{-a_5}} = \frac{1}{1 + e^0} = \frac{1}{2} = 0.5$$

Performing Backpropagation with the above values in mind:
- between node 'y' and node 'd':
Since this is the output layer, and we are given that the target output at y, $t = 1$.

$$\delta_5 = da_5 = y - t = z_5 - t = 0.5 - 1 = -0.5$$

$$dw_5 = da_5 * z_4 = -0.5 * 0.5 = -0.25$$

$$db_5 = da_5 = -0.5$$

- between node 'd' and node 'c':
For hidden layers,

$$\delta_j = h'(a_j) \sum_k w_{kj} \delta_k$$

$$\delta_4 = da_4 = w_5 * da_5 * \sigma'(a_4) = 1 * (-0.5) * \sigma'(0) = -0.5 * 0.25 = -0.125$$

where,

$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$

$$\sigma'(0) = \sigma(0)(1 - \sigma(0)) = 0.5 * 0.5 = 0.25$$

$$dw_4 = da_4 * z_3 = -0.125 * 0.5 = -0.0625$$

$$db_4 = da_4 = -0.125$$

- between node 'c' and node 'b':

$$\delta_3 = da_3 = w_4 * da_4 * \sigma'(a_3) = 1 * (-0.125) * \sigma'(0) = -0.125 * 0.25 = -0.03125 = \frac{-1}{32}$$

$$dw_3 = da_3 * z_2 = -0.03125 * 0.5 = -0.015625 = \frac{-1}{64}$$

$$db_3 = da_3 = -0.03125 = \frac{-1}{32}$$

- between node 'b' and node 'a':

$$\delta_2 = da_2 = w_3 * da_3 * \sigma'(a_2) = 1 * (\frac{-1}{32}) * \sigma'(0) = \frac{-1}{32} * 0.25 = -0.0078125 = \frac{-1}{128}$$

$$dw_2 = da_2 * z_1 = \frac{-1}{128} * 0.5 = -0.015625 = \frac{-1}{64}$$

$$db_3 = da_3 = -0.0039 = \frac{-1}{256}$$

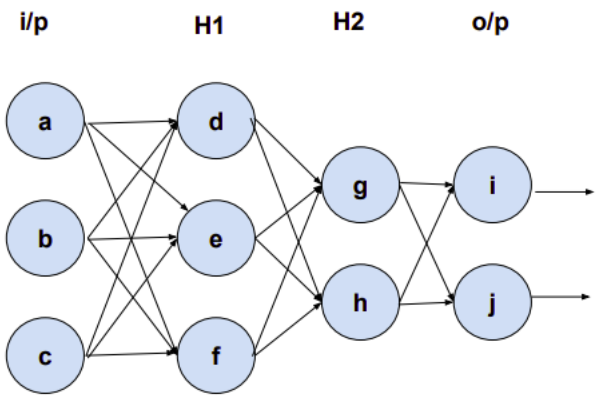
- between node 'a' and input layer, 'x':

$$\delta_1 = da_1 = w_2 * da_2 * \sigma'(a_1) = 1 * \frac{-1}{128}) * \sigma'(0) = \frac{-1}{128} * 0.25 = -0.00195 = \frac{-1}{512}$$

$$dw_1 = da_1 * x = \frac{-1}{512} * 0.5 = -0.0009765 = \frac{-1}{1024}$$

$$db_1 = da_1 = -0.00195 = \frac{-1}{512}$$

7. (2 marks) **[NN & Activation Functions]** The following diagram represents a feed-forward network with two hidden layers.



A weight on connection between nodes x and y is denoted by w_{xy}, such as w_{ad} is the weight on the connection between nodes a and d. The following table lists all the weights in the network:

$w_{ad} = 0.5$	$w_{be} = -1.4$	$w_{cf} = -1.25$	$w_{eh} = -2$	$w_{gj} = -1.5$
$w_{ae} = 0.9$	$w_{bf} = 0.75$	$w_{dg} = 1$	$w_{fg} = 3$	$w_{hi} = 0.5$
$w_{af} = -2$	$w_{cd} = 0$	$w_{dh} = 3$	$w_{fh} = 1.25$	$w_{hj} = -0.25$
$w_{bd} = 1.3$	$w_{ce} = 0.3$	$w_{eg} = 2.5$	$w_{gi} = 2.5$	

Find the output of the network for the following input vectors:
$V_1 = [0.2, 1, 3]$, $V2 = [2.5, 3, 7]$, $V3 = [0.75, -2, 3]$

(a) If *sigmoid* activation function is used in both H1 & H2

(b) If *tanh* activation function is used in both H1 & H2

(c) If *sigmoid* activation is used in H1 and *tanh* in H2

(d) If *tanh* activation is used in H1 & ReLU activation in H2

Please provide all steps and explain the same.

Solution: Choice of activation function for the output layer is not given in the question. So, we can treat the output layer without any activation functions. In that case, inputs arrived at the output layers will also be the final outputs.Then, this is like a multi-regression output problem. However, we can also have a optional interpretation. Generally, activation function in the output layer is determined by the nature of the data and the assumed distribution of target variables. From the diagram, we see that there is more than one target variable. So, it can be either a multi regression problem with two target variables, hence two output nodes. Then, we can use a linear function as the activation function in the output layer. This is equivalent to having no function in the output layer. Else, it can also be classification problem involving two classes. Then, we can use Softmax activation function in the output layer outputting probabilities. Let us calculate both the cases below:

(a)
In this, we use *sigmoid* activation function in both the hidden layers.
$sigmoid(x) = \frac{1}{1+e^{-x}}$

(i) Considering input V_1 :

Input to the node d in $H1$: $a * w_{ad} + b * w_{bd} + c * w_{cd}$
Input to the node e in $H1$: $a * w_{ae} + b * w_{be} + c * w_{ce}$
Input to the node f in $H1$: $a * w_{af} + b * w_{bf} + c * w_{cf}$
For faster calculation, these equations can be written in matrix form.

$$\begin{bmatrix} d\_input & e\_input & f\_input \end{bmatrix} = \begin{bmatrix} a & b & c \end{bmatrix} \begin{bmatrix} w_{ad} & w_{ae} & w_{af} \\ w_{bd} & w_{be} & w_{bf} \\ w_{cd} & w_{ce} & w_{cf} \end{bmatrix}$$

$$\begin{bmatrix} d\_input & e\_input & f\_input \end{bmatrix} = \begin{bmatrix} 1.4 & -0.32 & -3.4 \end{bmatrix}$$

Applying the *Sigmoid* activation functions at the nodes d, e, f, we get the following outputs from those nodes.
$\begin{bmatrix} d\_output & e\_output & f\_output \end{bmatrix} = \begin{bmatrix} 0.80218389 & 0.42067575 & 0.03229546 \end{bmatrix}$

Similar to above, using the weights between $H1$ and $H2$, calculating inputs to the nodes in $H2$:
$\begin{bmatrix} g\_input & h\_input \end{bmatrix} = \begin{bmatrix} 1.95075965 & 1.6055695 \end{bmatrix}$

Applying the *Sigmoid* activation functions at the nodes g, h, we get the following outputs from those nodes.
$\begin{bmatrix} g\_output & h\_output \end{bmatrix} = \begin{bmatrix} 0.87552945 & 0.83279536 \end{bmatrix}$

Similar to above, using the weights between $H2$ and the output layer, calculating inputs to the nodes in the output layer
$\begin{bmatrix} i\_input & j\_input \end{bmatrix} = \begin{bmatrix} 2.60522131 & -1.52149302 \end{bmatrix}$

Now, we can calculate the final outputs from the output layer in two different ways depending on the type of problem and the respective activation function.

Type 1: Multi-regression problem. We can then use a linear/identity (also called no activation) function. The, final outputs from i and j will be the same as their respective inputs.
$\begin{bmatrix} i\_output & j\_output \end{bmatrix} = \begin{bmatrix} 2.60522131 & -1.52149302 \end{bmatrix}$

Type 2: Classification problem. We can then use a Softmax activation function. The, final outputs from i and j will then be probabilities, with the majority probability predicting the class.

$Softmax(x_i) = \frac{e^{x_i}}{\sum_{k=1}^{no.ofclasses} e^{x_k}}$
$\begin{bmatrix} i\_output & j\_output \end{bmatrix} = \begin{bmatrix} 0.98412042 & 0.01587958 \end{bmatrix}$

(ii) Considering input V_2 :

Doing similar calculations as above,
$\begin{bmatrix} d\_input & e\_input & f\_input \end{bmatrix} = \begin{bmatrix} 5.15 & 0.15 & -11.5 \end{bmatrix}$

$\begin{bmatrix} d\_output & e\_output & f\_output \end{bmatrix} = \begin{bmatrix} 0.994234034 & 0.537429845 & 0.0000101299910 \end{bmatrix}$

$\begin{bmatrix} g\_input & h\_input \end{bmatrix} = \begin{bmatrix} 2.33783904 & 1.90785508 \end{bmatrix}$

$\begin{bmatrix} g\_output & h\_output \end{bmatrix} = \begin{bmatrix} 0.91196274 & 0.87077798 \end{bmatrix}$

$\begin{bmatrix} i\_input & j\_input \end{bmatrix} = \begin{bmatrix} 2.71529585 & -1.58563861 \end{bmatrix}$

Now, we can calculate the final outputs from the output layer in two different ways depending on the type of problem and the respective activation function.

Type 1: Multi-regression problem. We can then use a linear/identity (also called no activation) function. The, final outputs from i and j will be the same as their respective inputs.
$\begin{bmatrix} i\_output & j\_output \end{bmatrix} = \begin{bmatrix} 2.71529585 & -1.58563861 \end{bmatrix}$

Type 2: Classification problem. We can then use a Softmax activation function. The, final outputs from i and j will then be probabilities, with the majority probability predicting the class.
$\begin{bmatrix} i\_output & j\_output \end{bmatrix} = \begin{bmatrix} 0.98662542 & 0.01337458 \end{bmatrix}$

(iii) Considering input V_3 :

Doing similar calculations as above,
$\begin{bmatrix} d\_input & e\_input & f\_input \end{bmatrix} = \begin{bmatrix} -2.225 & 4.375 & -6.75 \end{bmatrix}$

$\begin{bmatrix} d\_output & e\_output & f\_output \end{bmatrix} = \begin{bmatrix} 0.09752784 & 0.98756835 & 0.00116951 \end{bmatrix}$

$\begin{bmatrix} g\_input & h\_input \end{bmatrix} = \begin{bmatrix} 2.56995724 & -1.6810913 \end{bmatrix}$

$\begin{bmatrix} g\_output & h\_output \end{bmatrix} = \begin{bmatrix} 0.92890287 & 0.15695102 \end{bmatrix}$

$\begin{bmatrix} i\_input & j\_input \end{bmatrix} = \begin{bmatrix} 2.40073269 & -1.43259206 \end{bmatrix}$

Now, we can calculate the final outputs from the output layer in two different ways depending on the type of problem and the respective activation function.

Type 1: Multi-regression problem. We can then use a linear/identity (also called no activation) function. The, final outputs from i and j will be the same as their respective inputs.
$\begin{bmatrix} i\_output & j\_output \end{bmatrix} = \begin{bmatrix} 2.40073269 & -1.43259206 \end{bmatrix}$

Type 2: Classification problem. We can then use a Softmax activation function. The, final outputs from i and j will then be probabilities, with the majority probability predicting the class.
$\begin{bmatrix} i\_output & j\_output \end{bmatrix} = \begin{bmatrix} 0.97882071 & 0.02117929 \end{bmatrix}$

(b)
In this, we use $tanh$ activation function in both the hidden layers. So, calculating similarly to above:
$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

(i) Considering input V_1 :

Doing similar calculations as above,
$\begin{bmatrix} d\_input & e\_input & f\_input \end{bmatrix} = \begin{bmatrix} 1.4 & -0.32 & -3.4 \end{bmatrix}$

$\begin{bmatrix} d\_output & e\_output & f\_output \end{bmatrix} = \begin{bmatrix} 0.88535165 & -0.30950692 & -0.99777493 \end{bmatrix}$

$\begin{bmatrix} g\_input & h\_input \end{bmatrix} = \begin{bmatrix} -2.88174044 & 2.02785013 \end{bmatrix}$

$\begin{bmatrix} g\_output & h\_output \end{bmatrix} = \begin{bmatrix} -0.99373934 & 0.96594329 \end{bmatrix}$

$\begin{bmatrix} i\_input & j\_input \end{bmatrix} = \begin{bmatrix} -2.0013767 & 1.24912318 \end{bmatrix}$

Now, we can calculate the final outputs from the output layer in two different ways depending on the type of problem and the respective activation function.

Type 1: Multi-regression problem. We can then use a linear/identity (also called no activation) function. The, final outputs from i and j will be the same as their respective inputs.
$\begin{bmatrix} i\_output & j\_output \end{bmatrix} = \begin{bmatrix} -2.0013767 & 1.24912318 \end{bmatrix}$

Type 2: Classification problem. We can then use a Softmax activation function. The, final outputs from i and j will then be probabilities, with the majority probability predicting the class.
$\begin{bmatrix} i\_output & j\_output \end{bmatrix} = \begin{bmatrix} 0.03730893 & 0.96269107 \end{bmatrix}$

(ii) Considering input V_2 :

Doing similar calculations as above,
$\begin{bmatrix} d\_input & e\_input & f\_input \end{bmatrix} = \begin{bmatrix} 5.15 & 0.15 & -11.5 \end{bmatrix}$

$\begin{bmatrix} d\_output & e\_output & f\_output \end{bmatrix} = \begin{bmatrix} 0.99993274 & 0.14888503 & -1.0 \end{bmatrix}$

$\begin{bmatrix} g\_input & h\_input \end{bmatrix} = \begin{bmatrix} -1.62785468 & 1.45202814 \end{bmatrix}$

$\begin{bmatrix} g\_output & h\_output \end{bmatrix} = \begin{bmatrix} -0.92575546 & 0.89609318 \end{bmatrix}$

$\begin{bmatrix} i\_input & j\_input \end{bmatrix} = \begin{bmatrix} -1.86634206 & 1.16460989 \end{bmatrix}$

Now, we can calculate the final outputs from the output layer in two different ways depending on the type of problem and the respective activation function.

Type 1: Multi-regression problem. We can then use a linear/identity (also called no activation) function. The, final outputs from i and j will be the same as their respective inputs.
$\begin{bmatrix} i\_output & j\_output \end{bmatrix} = \begin{bmatrix} -1.86634206 & 1.16460989 \end{bmatrix}$

Type 2: Classification problem. We can then use a Softmax activation function. The, final outputs from i and j will then be probabilities, with the majority probability predicting the class.
$\begin{bmatrix} i\_output & j\_output \end{bmatrix} = \begin{bmatrix} 0.04604699 & 0.95395301 \end{bmatrix}$

(iii) Considering input V_3 :

Doing similar calculations as above,
$\begin{bmatrix} d\_input & e\_input & f\_input \end{bmatrix} = \begin{bmatrix} -2.225 & 4.375 & -6.75 \end{bmatrix}$

$\begin{bmatrix} d\_output & e\_output & f\_output \end{bmatrix} = \begin{bmatrix} -0.9769125 & 0.99968313 & -0.99999726 \end{bmatrix}$

$\begin{bmatrix} g\_input & h\_input \end{bmatrix} = \begin{bmatrix} -1.47769645 & -6.18010031 \end{bmatrix}$

$\begin{bmatrix} g\_output & h\_output \end{bmatrix} = \begin{bmatrix} -0.90103551 & -0.99999143 \end{bmatrix}$

$\begin{bmatrix} i\_input & j\_input \end{bmatrix} = \begin{bmatrix} -2.75258448 & 1.60155112 \end{bmatrix}$

Now, we can calculate the final outputs from the output layer in two different ways depending on the type of problem and the respective activation function.

Type 1: Multi-regression problem. We can then use a linear/identity (also called no activa-

tion) function. The, final outputs from i and j will be the same as their respective inputs.
$\begin{bmatrix} i\_output & j\_output \end{bmatrix} = \begin{bmatrix} -2.75258448 & 1.60155112 \end{bmatrix}$

Type 2: Classification problem. We can then use a Softmax activation function. The, final outputs from i and j will then be probabilities, with the majority probability predicting the class.
$\begin{bmatrix} i\_output & j\_output \end{bmatrix} = \begin{bmatrix} 0.01269043 & 0.98730957 \end{bmatrix}$

(c)
In this, we use *sigmoid* activation function in $H1$ and *tanh* in $H2$. So, calculating similarly to above:

(i) Considering input V_1 :

Doing similar calculations as above,
$\begin{bmatrix} d\_input & e\_input & f\_input \end{bmatrix} = \begin{bmatrix} 1.4 & -0.32 & -3.4 \end{bmatrix}$

$\begin{bmatrix} d\_output & e\_output & f\_output \end{bmatrix} = \begin{bmatrix} 0.80218389 & 0.42067575 & 0.03229546 \end{bmatrix}$

$\begin{bmatrix} g\_input & h\_input \end{bmatrix} = \begin{bmatrix} 1.95075965 & 1.6055695 \end{bmatrix}$

$\begin{bmatrix} g\_output & h\_output \end{bmatrix} = \begin{bmatrix} 0.96037844 & 0.92250262 \end{bmatrix}$

$\begin{bmatrix} i\_input & j\_input \end{bmatrix} = \begin{bmatrix} 2.8621974 & -1.67119331 \end{bmatrix}$

Now, we can calculate the final outputs from the output layer in two different ways depending on the type of problem and the respective activation function.

Type 1: Multi-regression problem. We can then use a linear/identity (also called no activation) function. The, final outputs from i and j will be the same as their respective inputs.
$\begin{bmatrix} i\_output & j\_output \end{bmatrix} = \begin{bmatrix} 2.8621974 & -1.67119331 \end{bmatrix}$

Type 2: Classification problem. We can then use a Softmax activation function. The, final outputs from i and j will then be probabilities, with the majority probability predicting the class.
$\begin{bmatrix} i\_output & j\_output \end{bmatrix} = \begin{bmatrix} 0.98937003 & 0.01062997 \end{bmatrix}$

(ii) Considering input V_2 :

Doing similar calculations as above,
$\begin{bmatrix} d\_input & e\_input & f\_input \end{bmatrix} = \begin{bmatrix} 5.15 & 0.15 & -11.5 \end{bmatrix}$

$\begin{bmatrix} d\_output & e\_output & f\_output \end{bmatrix} = \begin{bmatrix} 0.994234034e & 0.537429845 & 0.0000101299910 \end{bmatrix}$

$\begin{bmatrix} g\_input & h\_input \end{bmatrix} = \begin{bmatrix} 2.33783904 & 1.90785508 \end{bmatrix}$

$\begin{bmatrix} g\_output & h\_output \end{bmatrix} = \begin{bmatrix} 0.98153368 & 0.9569049 \end{bmatrix}$

$\begin{bmatrix} i\_input & j\_input \end{bmatrix} = \begin{bmatrix} 2.93228666 & -1.71152675 \end{bmatrix}$

Now, we can calculate the final outputs from the output layer in two different ways depending on the type of problem and the respective activation function.

Type 1: Multi-regression problem. We can then use a linear/identity (also called no activation) function. The, final outputs from i and j will be the same as their respective inputs.
$\begin{bmatrix} i\_output & j\_output \end{bmatrix} = \begin{bmatrix} 2.93228666 & -1.71152675 \end{bmatrix}$

Type 2: Classification problem. We can then use a Softmax activation function. The, final outputs from i and j will then be probabilities, with the majority probability predicting the class.
$\begin{bmatrix} i\_output & j\_output \end{bmatrix} = \begin{bmatrix} 0.99047074 & 0.00952926 \end{bmatrix}$

(*iii*) Considering input V_3 :

Doing similar calculations as above,
$\begin{bmatrix} d\_input & e\_input & f\_input \end{bmatrix} = \begin{bmatrix} -2.225 & 4.375 & -6.75 \end{bmatrix}$

$\begin{bmatrix} d\_output & e\_output & f\_output \end{bmatrix} = \begin{bmatrix} 0.09752784 & 0.98756835 & 0.00116951 \end{bmatrix}$

$\begin{bmatrix} g\_input & h\_input \end{bmatrix} = \begin{bmatrix} 2.56995724 & -1.6810913 \end{bmatrix}$

$\begin{bmatrix} g\_output & h\_output \end{bmatrix} = \begin{bmatrix} 0.98835186 & -0.93300303 \end{bmatrix}$

$\begin{bmatrix} i\_input & j\_input \end{bmatrix} = \begin{bmatrix} 2.00437813 & -1.24927703 \end{bmatrix}$

Now, we can calculate the final outputs from the output layer in two different ways depending on the type of problem and the respective activation function.

Type 1: Multi-regression problem. We can then use a linear/identity (also called no activation) function. The, final outputs from i and j will be the same as their respective inputs.
$\begin{bmatrix} i\_output & j\_output \end{bmatrix} = \begin{bmatrix} 2.00437813 & -1.24927703 \end{bmatrix}$

Type 2: Classification problem. We can then use a Softmax activation function. The, final outputs from i and j will then be probabilities, with the majority probability predicting

the class.
$\begin{bmatrix} i\_output & j\_output \end{bmatrix} = \begin{bmatrix} 0.96280423 & 0.03719577 \end{bmatrix}$

(d)
In this, we use $tanh$ activation function in $H1$ and $ReLU$ in $H2$. So, calculating similarly to above:
$ReLU(x) = max(0.0, x)$

(i) Considering input V_1 :

Doing similar calculations as above,
$\begin{bmatrix} d\_input & e\_input & f\_input \end{bmatrix} = \begin{bmatrix} 1.4 & -0.32 & -3.4 \end{bmatrix}$

$\begin{bmatrix} d\_output & e\_output & f\_output \end{bmatrix} = \begin{bmatrix} 0.88535165 & -0.30950692 & -0.99777493 \end{bmatrix}$

$\begin{bmatrix} g\_input & h\_input \end{bmatrix} = \begin{bmatrix} -2.88174044 & 2.02785013 \end{bmatrix}$

$\begin{bmatrix} g\_output & h\_output \end{bmatrix} = \begin{bmatrix} 0.0 & 2.02785013 \end{bmatrix}$

$\begin{bmatrix} i\_input & j\_input \end{bmatrix} = \begin{bmatrix} 1.01392506 & -0.50696253 \end{bmatrix}$

Now, we can calculate the final outputs from the output layer in two different ways depending on the type of problem and the respective activation function.

Type 1: Multi-regression problem. We can then use a linear/identity (also called no activation) function. The, final outputs from i and j will be the same as their respective inputs.
$\begin{bmatrix} i\_output & j\_output \end{bmatrix} = \begin{bmatrix} 1.01392506 & -0.50696253 \end{bmatrix}$

Type 2: Classification problem. We can then use a Softmax activation function. The, final outputs from i and j will then be probabilities, with the majority probability predicting the class.
$\begin{bmatrix} i\_output & j\_output \end{bmatrix} = \begin{bmatrix} 0.82066915 & 0.17933085 \end{bmatrix}$

(ii) Considering input V_2 :

Doing similar calculations as above,
$\begin{bmatrix} d\_input & e\_input & f\_input \end{bmatrix} = \begin{bmatrix} 5.15 & 0.15 & -11.5 \end{bmatrix}$

$\begin{bmatrix} d\_output & e\_output & f\_output \end{bmatrix} = \begin{bmatrix} 0.99993274 & 0.14888503 & -1.0 \end{bmatrix}$

$\begin{bmatrix} g\_input & h\_input \end{bmatrix} = \begin{bmatrix} -1.62785468 & 1.45202814 \end{bmatrix}$

$\begin{bmatrix} g\_output & h\_output \end{bmatrix} = \begin{bmatrix} 0.0 & 1.45202814 \end{bmatrix}$

$\begin{bmatrix} i\_input & j\_input \end{bmatrix} = \begin{bmatrix} 0.72601407 & -0.36300704 \end{bmatrix}$

Now, we can calculate the final outputs from the output layer in two different ways depending on the type of problem and the respective activation function.

Type 1: Multi-regression problem. We can then use a linear/identity (also called no activation) function. The, final outputs from i and j will be the same as their respective inputs.
$\begin{bmatrix} i\_output & j\_output \end{bmatrix} = \begin{bmatrix} 0.72601407 & -0.36300704 \end{bmatrix}$

Type 2: Classification problem. We can then use a Softmax activation function. The, final outputs from i and j will then be probabilities, with the majority probability predicting the class.
$\begin{bmatrix} i\_output & j\_output \end{bmatrix} = \begin{bmatrix} 0.74819734 & 0.25180266 \end{bmatrix}$

(iii) Considering input V_3 :

Doing similar calculations as above,
$\begin{bmatrix} d\_input & e\_input & f\_input \end{bmatrix} = \begin{bmatrix} -2.225 & 4.375 & -6.75 \end{bmatrix}$

$\begin{bmatrix} d\_output & e\_output & f\_output \end{bmatrix} = \begin{bmatrix} -0.9769125 & 0.99968313 & -0.99999726 \end{bmatrix}$

$\begin{bmatrix} g\_input & h\_input \end{bmatrix} = \begin{bmatrix} -1.47769645 & -6.18010031 \end{bmatrix}$

$\begin{bmatrix} g\_output & h\_output \end{bmatrix} = \begin{bmatrix} 0.0 & 0.0 \end{bmatrix}$

$\begin{bmatrix} i\_input & j\_input \end{bmatrix} = \begin{bmatrix} 0.0 & 0.0 \end{bmatrix}$

Now, we can calculate the final outputs from the output layer in two different ways depending on the type of problem and the respective activation function.

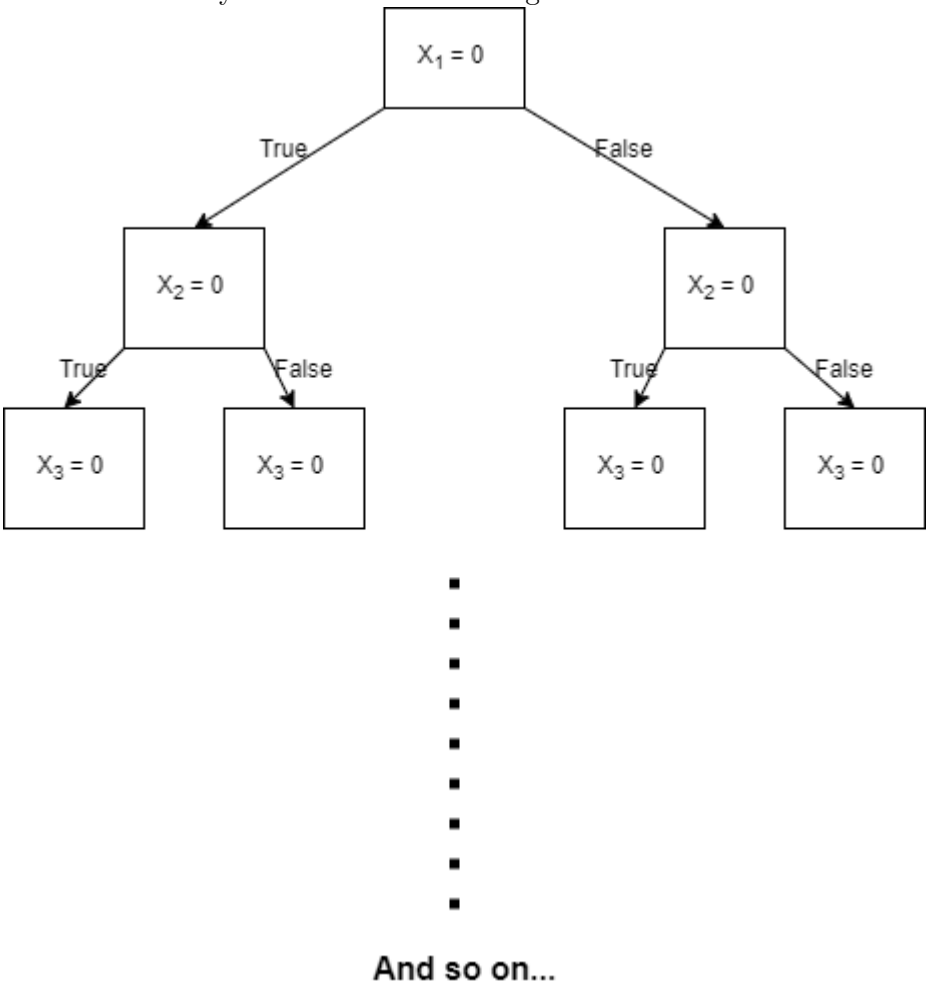
Type 1: Multi-regression problem. We can then use a linear/identity (also called no activation) function. The, final outputs from i and j will be the same as their respective inputs.
$\begin{bmatrix} i\_output & j\_output \end{bmatrix} = \begin{bmatrix} 0.0 & 0.0 \end{bmatrix}$

Type 2: Classification problem. We can then use a Softmax activation function. The, final outputs from i and j will then be probabilities, with the majority probability predicting the class.If probabilities are equal, we can decide the class randomly or based on convention.
$\begin{bmatrix} i\_output & j\_output \end{bmatrix} = \begin{bmatrix} 0.5 & 0.5 \end{bmatrix}$

8. (2 marks) **[Decision Trees]** Consider a dataset with each data point $x \in \{0,1\}^m$, i.e., x is a binary valued feature vector with m features, and the class label $y \in \{+1,-1\}$. Suppose the true classifier is a majority vote over the features, such that
$y = sign\Big(\sum_{i=1}^{m}(2x_i - 1)\Big)$

where x_i is the i^{th} component of the feature vector. Suppose you build a binary decision tree with minimum depth, that is consistent with the data described above. What is the range of of number of leaves which such a decision tree will have?

Solution: Binary decision tree describing the data above:



We create the binary decision tree as follows. At each level, the nodes are split into 2 children based on the truth value of the binary valued feature.Features can be chosen in any order. However, as each feature has only 2 possible values, a feature is never repeated. Hence, each feature corresponds to one level in the binary tree. We know that the number of leaves in a binary tree depends on the height of the binary tree. Here, as each level corresponds to one feature, height of the tree will be the number of features considered in constructing the decision tree. In the worst case, all the features need to be considered in

constructing the decision tree.In that worst case, the decision tree will be a perfect binary tree. Number of leaves in a perfect binary tree is 2^h where h is the height. Here, height is the number of features considered in constructing the tree. In the worst case, all the m features are considered. So, the maximum number of leaves is 2^m.

However, in the best case, only half of the features are enough to classify using the tree. This is because, in the best case, if the first half of the features have the same value, then no more further splitting of nodes is necessary. As this is a majority vote over the features, class label can be found. Hence, in the best case, only $\frac{m}{2}$ features can be used for constructing the tree. So, as shown earlier, minimum number of leaves in this best case will be $2^{\frac{m}{2}}$.

So, range of number of leaves is $2^{\frac{m}{2}}$ to 2^m where m is the total number of features.