CS5691: PRML Assignment #2

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

Teammate 1: Varun Gumma **Roll number:** CS21M070
Teammate 2: Uday Sai Vemula **Roll number:** CS21M066

- 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.

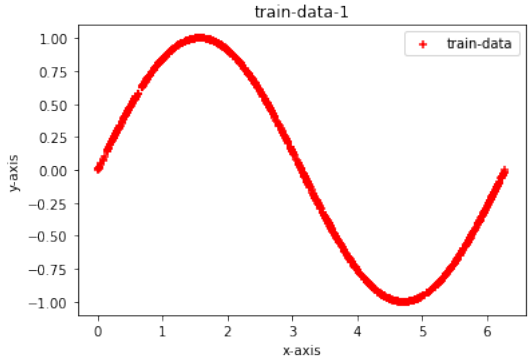
1. **[ANN]** In this Question, you will code a single layer ANN with Sigmoid Activation function and appropriate loss function from scratch. Train the ANN for the Dataset1 and Dataset2

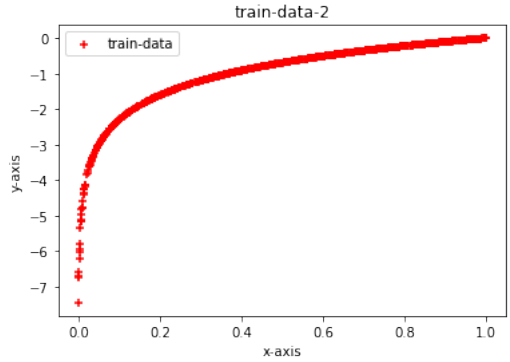
   NOTE: Test Data should not be used for training.

   NOTE 2: You need to code from scratch.

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

Solution:

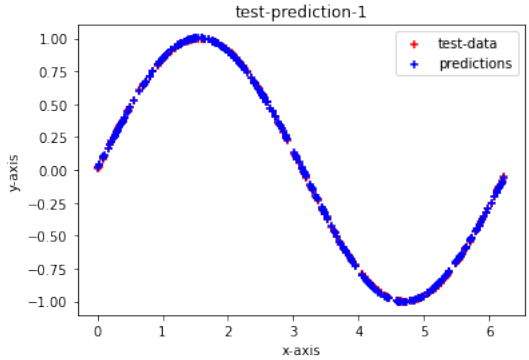




(b) (1 mark) **For data set 1** : (1) Write the number of nodes in the hidden layer and learning rate used (2) Plot Test Data and prediction on Test Data in the same graph with different colors and appropriate legend.

Solution:

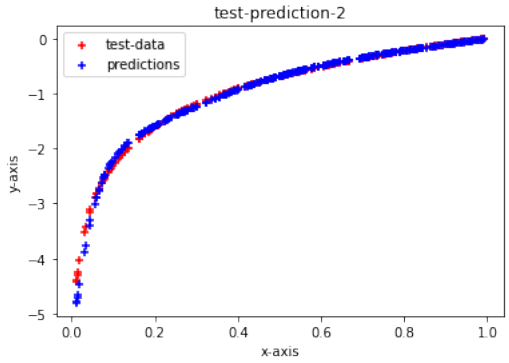
1. Hidden units: `20`
2. Learning rate: `0.0002`
3. Test data prediction:



(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. Hidden units: `30`
2. Learning rate: `0.00025`
3. Test data prediction:



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

Solution:

	Avg. Train Error	Avg. Test Error
Dataset-1	`0.0`	`0.0`
Dataset-2	`0.007`	`0.002`

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

Solution: As target values are real and continuous, we use Mean Squared Error loss, i.e. $L = \sum_{n=1}^{N} \frac{1}{2N}(\hat{y}_n - y_n)^2$ to perform regression.

(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: Link: Colab Notebook ANN
Hyperparameters tuned: `Random Seed`, `iterations`, `learning_rate`, `hidden_nodes`.
Hyperparam values - `Random Seed`: 42, `iterations`: 100000 for both datasets, `hidden_nodes`: 20 for first, 30 for second, `learning_rate`: $2 \cdot 10^{-4}$ for first, $2.5 \cdot 10^{-4}$ for second.

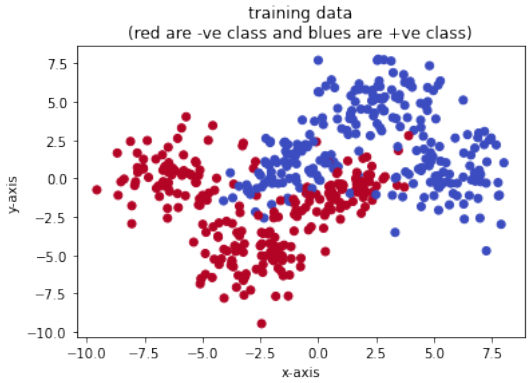
2. **[AdaBoost]** In this question, you will code the AdaBoost algorithm. Follow the instructions in this Jupyter Notebook for this question. Find the dataset for the question here.

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.

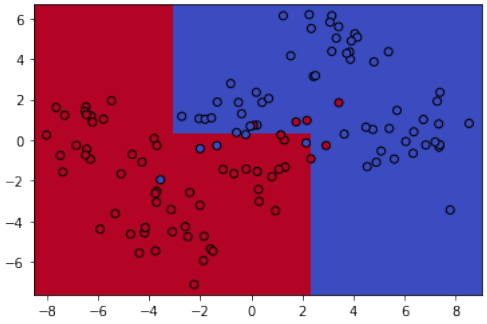
(a) (1 mark) Plot the training data.

Solution:



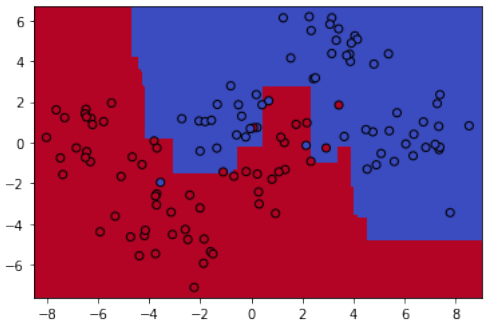
(b) (1 mark) **For training with k=5** : (1) Plot the learnt decision surface. (2) Write down the test accuracy.

Solution: Test accuracy: `0.917`



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

Solution: Test accuracy: `0.942`



(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: Link: Colab Notebook AdaBoost
Hyperparameters tuned: `Random Seed`, `criterion`, `max_features`.

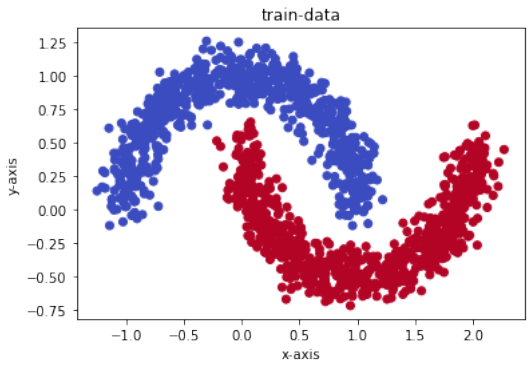
Hyperparam values - `Seed`: 1, `criterion`: 'entropy', `max_features`: None

We tuned these parameters alone for the internal Decision Tree classifier as for a decision stump that we are using most the parameters like, `max_depth`, `min_samples_leaf`, `min_samples_split`, `max_leaf_nodes` etc are either irrelevant or fixed. We did not tune the `class_weight` hyperparameter as the dataset was fairly balanced. We simply changed the values of `criterion` and `max_features` by hand (as they were so few) and kept those which gave the best accuracy on the test set. We preferred to keep `max_features` as None as it analyses all features (not just a subset of them) before making the appropriate split.

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: No, the data is not linearly separable.

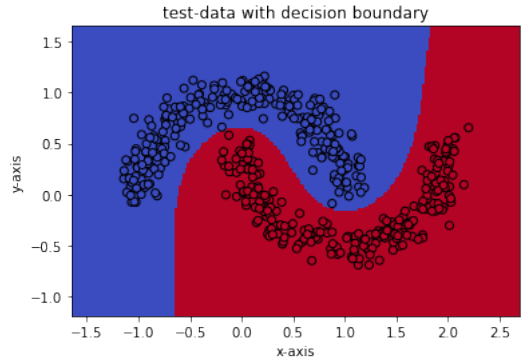


(b) (1 mark) Using *sklearn.svm* (read the documentation here), bulid a classifier that classifies the data points in the testing data set using the Radial Basis Function (RBF) kernel.How do you tune the involved hyperparameters?

Solution: We split the given training data (80 : 20) into new train and validation data respectively and used the new train data to fit the model (with varying hyperparameters). We evaluated its performance on the validation data and chose those hyperparameters for the final model which gave that best accuracy on the validation data. Since we are using RBF kernel, the major hyperparameters we could tune were `C`, `gamma`, `shrinking`. The rest of the hyperparameters for the SVC class either deal with multi-class classification or `poly/linear` kernels. We did not tune for the hyperparameter `class_weight` as the dataset is fairly balanced.

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

Solution: Test Accuracy: `1.0`



(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: Link: Colab Notebook Kernel
Hypterparameters tuned: `Random Seed`, `C`, `gamma`, `shrinking`
Hyperparam values - `Random Seed`: 42, `C`: 1.0, `gamma`: 'scale', `shrinking`: True

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: The probability wrong answer is same as the probability of independent correct trials being less than $N/2$. It follows a binomial distribution where probability of being correct (in a single trail) equals $0.5 + \delta$. $\therefore P(incorrect) = P(\#incorrect\ trials \geq \frac{N}{2}) = \sum_{i=0}^{N/2} \binom{N}{i} \cdot (0.5+\delta)^i \cdot (0.5-\delta)^{N-i}$. Let X_i be an indicator random variable which is 1 when the i^{th} trial is a incorrect else 0. $\mathbb{E}(X_i) = (0.5 - \delta) \cdot 1 + 0 = (0.5 - \delta)$ and $\sum_{i=1}^{N} X_i$ gives the number incorrect trials.

$$
\begin{aligned}
P(incorrect) &= P(\sum_{i=1}^{N} X_i \geq \frac{N}{2}) \\
&= P(\sum_{i=1}^{N} (X_i - (0.5 - \delta)) \geq \frac{N}{2} - (0.5 - \delta)N) \\
&\text{(adding 0.5 - } \delta \text{ to both sides of inequality)} \\
&= P(\sum_{i=1}^{N} (X_i - \mathbb{E}(X_i)) > N\delta) \\
&= P((A_N - \mathbb{E}(A_N)) \geq N\delta) \quad (A_N = \sum_{i=1}^{N} X_i) \\
&\leq e^{-2N\delta^2} \quad \text{(Hoeffding's Inequality)}
\end{aligned}
$$

As we want the above result to be less than ϵ, i.e. $P(incorrect) \leq e^{-2N\delta^2} < \epsilon$, we have $P(correct) \geq 1 - \epsilon$ and $2N\delta^2 > \ln(\epsilon^{-1})$ or $N > \frac{1}{2}\delta^{-2} \ln(\epsilon^{-1})$.

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: The error function here is squared-error and hence the expression would be $E = \frac{1}{2}\sum_{i=1}^{N}(t_n - f_m(\mathbf{x}_n))^2$. $f_m(\mathbf{x}) = \frac{1}{2}\sum_{i=1}^{m}\alpha_l y_l(\mathbf{x}) = \frac{1}{2}\sum_{i=1}^{m-1}\alpha_l y_l(\mathbf{x}) + \frac{1}{2}\alpha_m y_m(\mathbf{x}) = f_{m-1}(\mathbf{x}) + \frac{1}{2}\alpha_m y_m(\mathbf{x})$. $\therefore E = \frac{1}{2}\sum_{i=1}^{N}(t_n - f_m(\mathbf{x}_n))^2 = \frac{1}{2}\sum_{i=1}^{N}((t_n - f_{m-1}(\mathbf{x}_n)) - \frac{1}{2}\alpha_m y_m(\mathbf{x}_n))^2$. Let's denote $t_n - f_{m-1}(\mathbf{x}_n)$ by $r_{m-1}(\mathbf{x}_n)$. $\therefore E = \frac{1}{2}\sum_{i=1}^{N}(r_{m-1}(\mathbf{x}_n) - \frac{1}{2}\alpha_m y_m(\mathbf{x}_n))^2$.

To get minimum error when fitting classifier y_m, we minimize the error w.r.t $y_m(\mathbf{x}_n)$. $\therefore \frac{dE}{dy_m(\mathbf{x}_n)} = -\frac{\alpha_m}{2}(r_{m-1}(\mathbf{x}_n) - \frac{1}{2}\alpha_m y_m(\mathbf{x})) = 0$. $\therefore r_{m-1}(\mathbf{x}_n) - \frac{1}{2}\alpha_m y_m(\mathbf{x}_n) = 0$ or $y_m(\mathbf{x}_n) = \frac{2}{\alpha_m}(r_{m-1}(\mathbf{x}_n)) = \frac{2}{\alpha_m}(t_n - f_{m-1}(\mathbf{x}_n))$. This means, to get the best overall prediction we should have $y_m(\mathbf{x}_n)$ as $\frac{2}{\alpha_m}(t_n - f_{m-1}(\mathbf{x}_n))$ or fit y_m to $\frac{2}{\alpha_m}(t - f_{m-1}(\mathbf{x}))$.

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: Here, let's assume the 5 weights from left to right are w_1, w_2, w_3, w_4 and w_5. The pre-activations for all the layers (hidden and output) from left to right are a_1, a_2, a_3, a_4, a_5 and their corresponding activations are z_1, z_2, z_3, z_4, z_5 respectively.

1. For the given weights and biases:
   $x = 0.5$
   $a_1 = b_1 + w_1 x = -0.5 + 1 * 0.5 = 0$, $z_1 = g(a_1) = 0.5$
   $a_2 = b_2 + w_2 z_1 = -0.5 + 1 * 0.5 = 0$, $z_2 = g(a_2) = 0.5$
   $a_3 = b_3 + w_3 z_2 = -0.5 + 1 * 0.5 = 0$, $z_3 = g(a_3) = 0.5$
   $a_4 = b_4 + w_4 z_3 = -0.5 + 1 * 0.5 = 0$, $z_4 = g(a_4) = 0.5$
   $a_5 = z_5 = \hat{y} = w_5 z_4 = 0.5$

Since we have linear/identity activation function for output layer, we are performing regression and the MSE is given by $L(y, \hat{y}) = \frac{1}{2}(\hat{y} - y)^2$. As g represents the sigmoid function, $g'(a_i) = \frac{\partial z_i}{\partial a_i} = g(a_i) \cdot (1 - g(a_i)) = z_i(1 - z_i)$. Further, $\frac{\partial a_i}{\partial z_{i-1}} = w_i$.

(a) $\delta_5 = \frac{\partial L}{\partial a_5} = \frac{\partial L}{\partial \hat{y}} = (\hat{y} - y) = -\frac{1}{2}$

(b) $\delta_4 = \frac{\partial L}{\partial a_4} = \frac{\partial L}{\partial a_5}\frac{\partial a_5}{\partial z_4}\frac{\partial z_4}{\partial a_4} = (\hat{y} - y)w_5 z_4(1 - z_4) = -\frac{1}{2^3}$

(c) $\delta_3 = \frac{\partial L}{\partial a_3} = \frac{\partial L}{\partial a_5}\frac{\partial a_5}{\partial z_4}\frac{\partial z_4}{\partial a_4}\frac{\partial a_4}{\partial z_3}\frac{\partial z_3}{\partial a_3}\frac{\partial a_3}{\partial w_3} = (\hat{y} - y)w_5 z_4(1 - z_4)w_4 z_3(1 - z_3) = -\frac{1}{2^5}$

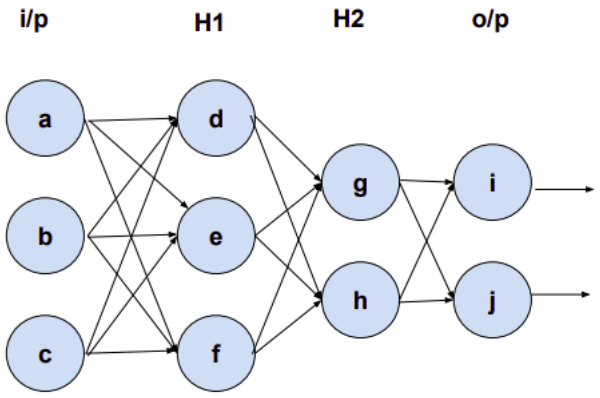
(d) $\delta_2 = \frac{\partial L}{\partial a_2} = \frac{\partial L}{\partial a_5}\frac{\partial a_5}{\partial z_4}\frac{\partial z_4}{\partial a_4}\frac{\partial a_4}{\partial z_3}\frac{\partial z_3}{\partial a_3}\frac{\partial a_3}{\partial z_2}\frac{\partial z_2}{\partial a_2}\frac{\partial a_2}{\partial w_2} = (\hat{y}-y)w_5 z_4(1-z_4)w_4 z_3(1-z_3)w_3 z_2(1-z_2) = -\frac{1}{2^7}$

(e) $\delta_1 = \frac{\partial L}{\partial a_1} = \frac{\partial L}{\partial a_5}\frac{\partial a_5}{\partial z_4}\frac{\partial z_4}{\partial a_4}\frac{\partial a_4}{\partial z_3}\frac{\partial z_3}{\partial a_3}\frac{\partial a_3}{\partial z_2}\frac{\partial z_2}{\partial a_2}\frac{\partial a_2}{\partial z_1}\frac{\partial z_1}{\partial a_1} = (\hat{y} - y)w_5 z_4(1 - z_4)w_4 z_3(1 - z_3)w_3 z_2(1 - z_2)w_2 z_1(1 - z_1) = -\frac{1}{2^9}$

$\therefore$ ratio of gradient updates ($\delta_5 : \delta_4 : \delta_3 : \delta_2 : \delta_1$ cancelling out $(\hat{y} - y)$) is
$1 : w_5 z_4(1 - z_4) : w_5 z_4(1 - z_4)w_4 z_3(1 - z_3) : w_5 z_4(1 - z_4)w_4 z_3(1 - z_3)w_3 z_2(1 - z_2) : w_5 z_4(1 - z_4)w_4 z_3(1 - z_3)w_3 z_2(1 - z_2)w_2 z_1(1 - z_1) = 1 : \frac{1}{4} : \frac{1}{16} : \frac{1}{64} : \frac{1}{256} = 256 : 64 : 16 : 4 : 1$

2. The gradient update magnitude decays exponentially as we move across the chain away from the output. With every hidden node, the magnitude becomes $\frac{1}{4^{th}}$ with the ratio being $256 : 64 : 16 : 4 : 1$

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: In the question it is given that there are 2 hidden layers, 1 input and 1 output layer. Let's denote the input layer as H_0 and output layer as H_L and hidden layers as H_1, H_2. The equation of activation is $H_i = g(W_i H_{i-1} + B)$ where W_i is weight matrix between the layers i and $i-1$ and H_{i-1} is the input vector to the layer i and B is the bias. $g(x)$ is the activation function. Since no bias is given in the question, let us assume it as 0 for the entire solution. Here, we forward propagate all input vectors together to simultaneously get all the outputs. **Note**: all values are taken precisely till 3 decimals, so answers might differ in decimal values.

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

$\implies H_1 = \sigma(W_1 H_0 + B)$

assuming that bias is zero

$\implies H_1 = \sigma(W_1 H_0)$

The three input vectors given are $V_1 = \begin{bmatrix} 0.2 \\ 1 \\ 3 \end{bmatrix}$, $V2 = \begin{bmatrix} 2.5 \\ 3 \\ 7 \end{bmatrix}$, $V3 = \begin{bmatrix} 0.75 \\ -2 \\ 3 \end{bmatrix}$

$\implies H_{11} = \sigma(W_1 V_1), \qquad H_{12} = \sigma(W_1 V_2), \qquad H_{13} = \sigma(W_1 V_3)$

Let's write all the three vectors combined into a matrix to make computation easy.

$$\implies H_1 = \sigma(W_1 \begin{bmatrix} V_1 & V_2 & V_3 \end{bmatrix})$$

$$= \sigma\left(\begin{bmatrix} W_{ad} & W_{bd} & W_{cd} \\ W_{ae} & W_{be} & W_{ce} \\ W_{af} & W_{bf} & W_{cf} \end{bmatrix} \begin{bmatrix} 0.2 & 2.5 & 0.75 \\ 1 & 3 & -2 \\ 3 & 7 & 3 \end{bmatrix} \right)$$

$$= \sigma\left(\begin{bmatrix} 0.5 & 1.3 & 0 \\ 0.9 & -1.4 & 0.3 \\ -2 & 0.75 & -1.25 \end{bmatrix} \begin{bmatrix} 0.2 & 2.5 & 0.75 \\ 1 & 3 & -2 \\ 3 & 7 & 3 \end{bmatrix} \right)$$

$$= \sigma\left(\begin{bmatrix} 1.4 & 5.15 & -2.225 \\ -0.32 & 0.15 & 4.375 \\ -3.4 & -11.5 & -6.75 \end{bmatrix} \right)$$

$\sigma(x) = \frac{1}{1+e^{-x}}$. To get the sigmoid of the whole matrix, we apply sigmoid function to each element of the matrix.

$$\therefore H_1 = \begin{bmatrix} 0.802 & 0.994 & 0.097 \\ 0.420 & 0.537 & 0.987 \\ 0.032 & 0.000 & 0.001 \end{bmatrix}$$

Similarly, we get $H_2 = \sigma(W_2 H_1)$

$$\begin{aligned} \implies H_2 &= \sigma(W_2 H_1) \\ &= \sigma\left(\begin{bmatrix} W_{dg} & W_{eg} & W_{fg} \\ W_{dh} & W_{eh} & W_{fg} \end{bmatrix} \begin{bmatrix} 0.802 & 0.994 & 0.097 \\ 0.420 & 0.537 & 0.987 \\ 0.032 & 0.000 & 0.001 \end{bmatrix} \right) \\ &= \sigma\left(\begin{bmatrix} 1 & 2.5 & 3 \\ 3 & -2 & 1.25 \end{bmatrix} \begin{bmatrix} 0.802 & 0.994 & 0.097 \\ 0.420 & 0.537 & 0.987 \\ 0.032 & 0.000 & 0.001 \end{bmatrix} \right) \\ &= \sigma\left(\begin{bmatrix} 1.948 & 2.336 & 2.567 \\ 1.606 & 1.908 & -1.681 \end{bmatrix} \right) \\ &= \begin{bmatrix} 0.875 & 0.911 & 0.928 \\ 0.832 & 0.870 & 0.156 \end{bmatrix} \end{aligned}$$

For the final output layer, $H_L = W_L H_2$ (because no activation function is given in the question to apply for output layer)

$$\begin{aligned} \implies H_L &= W_L H_2 \\ &= \begin{bmatrix} W_{gi} & W_{hi} \\ W_{gj} & W_{hj} \end{bmatrix} \begin{bmatrix} 0.875 & 0.911 & 0.928 \\ 0.832 & 0.870 & 0.156 \end{bmatrix} \\ &= \begin{bmatrix} 2.5 & 0.5 \\ -1.5 & -0.25 \end{bmatrix} \begin{bmatrix} 0.875 & 0.911 & 0.928 \\ 0.832 & 0.870 & 0.156 \end{bmatrix} \\ &= \begin{bmatrix} 2.603 & 2.712 & 2.398 \\ -1.520 & -1.585 & -1.431 \end{bmatrix} \end{aligned}$$

If sigmoid activation function is used for both hidden layers, then final output vectors will be $\begin{bmatrix} 2.603 \\ -1.520 \end{bmatrix}, \begin{bmatrix} 2.712 \\ -1.585 \end{bmatrix}, \begin{bmatrix} 2.398 \\ -1.431 \end{bmatrix}$

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

$\implies H_1 = tanh(W_1 H_0 + B)$

assuming that bias is zero

$\implies H_1 = tanh(W_1 H_0)$

three input vectors are given $V_1 = \begin{bmatrix} 0.2 \\ 1 \\ 3 \end{bmatrix}$, $V2 = \begin{bmatrix} 2.5 \\ 3 \\ 7 \end{bmatrix}$, $V3 = \begin{bmatrix} 0.75 \\ -2 \\ 3 \end{bmatrix}$

$\implies H_{11} = tanh(W_1 V_1), \qquad H_{12} = tanh(W_1 V_2), \qquad H_{13} = tanh(W_1 V_3)$
Let's write all the three vectors combine in a matrix form to make computation easy.

$$\begin{aligned}
\implies H_1 &= tanh(W_1 \begin{bmatrix} V_1 & V_2 & V_3 \end{bmatrix}) \\
&= tanh\left(\begin{bmatrix} W_{ad} & W_{bd} & W_{cd} \\ W_{ae} & W_{be} & W_{ce} \\ W_{af} & W_{bf} & W_{cf} \end{bmatrix} \begin{bmatrix} 0.2 & 2.5 & 0.75 \\ 1 & 3 & -2 \\ 3 & 7 & 3 \end{bmatrix} \right) \\
&= tanh\left(\begin{bmatrix} 0.5 & 1.3 & 0 \\ 0.9 & -1.4 & 0.3 \\ -2 & 0.75 & -1.25 \end{bmatrix} \begin{bmatrix} 0.2 & 2.5 & 0.75 \\ 1 & 3 & -2 \\ 3 & 7 & 3 \end{bmatrix} \right) \\
&= tanh\left(\begin{bmatrix} 1.4 & 5.15 & -2.225 \\ -0.32 & 0.15 & 4.375 \\ -3.4 & -11.5 & -6.75 \end{bmatrix} \right)
\end{aligned}$$

$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$. To get the tanh of the whole matrix, we apply tanh function to each element of the matrix.

$$\therefore H_1 = \begin{bmatrix} 0.885 & 0.999 & -0.976 \\ -0.309 & 0.148 & 0.999 \\ -0.997 & -0.999 & -0.999 \end{bmatrix}$$

Similarly, we get $H_2 = tanh(W_2 H_1)$

$$\begin{aligned}
\implies H_2 &= tanh(W_2 H_1) \\
&= tanh\left(\begin{bmatrix} W_{dg} & W_{eg} & W_{fg} \\ W_{dh} & W_{eh} & W_{fg} \end{bmatrix} \begin{bmatrix} 0.885 & 0.999 & -0.976 \\ -0.309 & 0.148 & 0.999 \\ -0.997 & -0.999 & -0.999 \end{bmatrix} \right) \\
&= tanh\left(\begin{bmatrix} 1 & 2.5 & 3 \\ 3 & -2 & 1.25 \end{bmatrix} \begin{bmatrix} 0.885 & 0.999 & -0.976 \\ -0.309 & 0.148 & 0.999 \\ -0.997 & -0.999 & -0.999 \end{bmatrix} \right) \\
&= tanh\left(\begin{bmatrix} -2.878 & -1.628 & -1.475 \\ 2.026 & 1.452 & -6.174 \end{bmatrix} \right) \\
&= \begin{bmatrix} -0.993 & -0.925 & -0.900 \\ 0.965 & 0.896 & -0.999 \end{bmatrix}
\end{aligned}$$

For the final output layer, $H_L = W_L H_2$ (because no activation function is given in the

question to apply for output layer)

$$\begin{aligned}
\implies H_L &= W_L H_2 \\
&= \begin{bmatrix} W_{gi} & W_{hi} \\ W_{gj} & W_{hj} \end{bmatrix} \begin{bmatrix} -0.993 & -0.925 & -0.900 \\ 0.965 & 0.896 & -0.999 \end{bmatrix} \\
&= \begin{bmatrix} 2.5 & 0.5 \\ -1.5 & -0.25 \end{bmatrix} \begin{bmatrix} -0.993 & -0.925 & -0.900 \\ 0.965 & 0.896 & -0.999 \end{bmatrix} \\
&= \begin{bmatrix} -2 & -1.864 & -2.749 \\ 1.248 & 1.163 & 1.599 \end{bmatrix}
\end{aligned}$$

If tanh activation function is used for both hidden layers, then final output vectors will be $\begin{bmatrix} -2 \\ 1.248 \end{bmatrix}, \begin{bmatrix} -1.864 \\ 1.163 \end{bmatrix}, \begin{bmatrix} -2.749 \\ 1.599 \end{bmatrix}$

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

$\implies H_1 = \sigma(W_1 H_0 + B)$

assuming that bias is zero

$\implies H_1 = \sigma(W_1 H_0)$

three input vectors are given $V_1 = \begin{bmatrix} 0.2 \\ 1 \\ 3 \end{bmatrix}$, $V2 = \begin{bmatrix} 2.5 \\ 3 \\ 7 \end{bmatrix}$, $V3 = \begin{bmatrix} 0.75 \\ -2 \\ 3 \end{bmatrix}$

$\implies H_{11} = \sigma(W_1 V_1), \qquad H_{12} = \sigma(W_1 V_2), \qquad H_{13} = \sigma(W_1 V_3)$

Let's write all the three vectors combine in a matrix form to make computation easy.

$$\begin{aligned}
\implies H_1 &= \sigma(W_1 \begin{bmatrix} V_1 & V_2 & V_3 \end{bmatrix}) \\
&= \sigma\left(\begin{bmatrix} W_{ad} & W_{bd} & W_{cd} \\ W_{ae} & W_{be} & W_{ce} \\ W_{af} & W_{bf} & W_{cf} \end{bmatrix} \begin{bmatrix} 0.2 & 2.5 & 0.75 \\ 1 & 3 & -2 \\ 3 & 7 & 3 \end{bmatrix} \right) \\
&= \sigma\left(\begin{bmatrix} 0.5 & 1.3 & 0 \\ 0.9 & -1.4 & 0.3 \\ -2 & 0.75 & -1.25 \end{bmatrix} \begin{bmatrix} 0.2 & 2.5 & 0.75 \\ 1 & 3 & -2 \\ 3 & 7 & 3 \end{bmatrix} \right) \\
&= \sigma\left(\begin{bmatrix} 1.4 & 5.15 & -2.225 \\ -0.32 & 0.15 & 4.375 \\ -3.4 & -11.5 & -6.75 \end{bmatrix} \right)
\end{aligned}$$

$\sigma(x) = \frac{1}{1+e^{-x}}$. To get the sigmoid of the whole matrix, we apply sigmoid function to each element of the matrix.

$$\therefore H_1 = \begin{bmatrix} 0.802 & 0.994 & 0.097 \\ 0.420 & 0.537 & 0.987 \\ 0.032 & 0.000 & 0.001 \end{bmatrix}$$

Similarly, we get $H_2 = tanh(W_2 H_1)$

$$\begin{aligned}
\implies H_2 &= tanh(W_2 H_1) \\
&= tanh\left(\begin{bmatrix} W_{dg} & W_{eg} & W_{fg} \\ W_{dh} & W_{eh} & W_{fg} \end{bmatrix} \begin{bmatrix} 0.802 & 0.994 & 0.097 \\ 0.420 & 0.537 & 0.987 \\ 0.032 & 0.000 & 0.001 \end{bmatrix}\right) \\
&= tanh\left(\begin{bmatrix} 1 & 2.5 & 3 \\ 3 & -2 & 1.25 \end{bmatrix} \begin{bmatrix} 0.802 & 0.994 & 0.097 \\ 0.420 & 0.537 & 0.987 \\ 0.032 & 0.000 & 0.001 \end{bmatrix}\right) \\
&= tanh\left(\begin{bmatrix} 1.948 & 2.336 & 2.567 \\ 1.606 & 1.908 & -1.681 \end{bmatrix}\right)
\end{aligned}$$

$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$. To get the tanh of the whole matrix, we apply tanh function to each element of the matrix.

$$\therefore H_2 = \begin{bmatrix} 0.960 & 0.981 & 0.988 \\ 0.922 & 0.956 & -0.932 \end{bmatrix}$$

For the final output layer, $H_L = W_L H_2$ (because no activation function is given in the question to apply for output layer)

$$\begin{aligned}
\implies H_L &= W_L H_2 \\
&= \begin{bmatrix} W_{gi} & W_{hi} \\ W_{gj} & W_{hj} \end{bmatrix} \begin{bmatrix} 0.960 & 0.981 & 0.988 \\ 0.922 & 0.956 & -0.932 \end{bmatrix} \\
&= \begin{bmatrix} 2.5 & 0.5 \\ -1.5 & -0.25 \end{bmatrix} \begin{bmatrix} 0.960 & 0.981 & 0.988 \\ 0.922 & 0.956 & -0.932 \end{bmatrix} \\
&= \begin{bmatrix} 2.861 & 2.930 & 2.004 \\ -1.670 & -1.710 & -1.249 \end{bmatrix}
\end{aligned}$$

If sigmoid activation function is used for H_1 hidden layer and tanh is used for H_2 hidden layer, then final output vectors will be $\begin{bmatrix} 2.861 \\ -1.670 \end{bmatrix}, \begin{bmatrix} 2.930 \\ -1.710 \end{bmatrix}, \begin{bmatrix} 2.004 \\ -1.249 \end{bmatrix}$

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

$\implies H_1 = tanh(W_1 H_0 + B)$

assuming that bias is zero

$\implies H_1 = tanh(W_1 H_0)$

three input vectors are given $V_1 = \begin{bmatrix} 0.2 \\ 1 \\ 3 \end{bmatrix}$, $V_2 = \begin{bmatrix} 2.5 \\ 3 \\ 7 \end{bmatrix}$, $V_3 = \begin{bmatrix} 0.75 \\ -2 \\ 3 \end{bmatrix}$

$\implies H_{11} = tanh(W_1 V_1), \qquad H_{12} = tanh(W_1 V_2), \qquad H_{13} = tanh(W_1 V_3)$

Let's write all the three vectors combine in a matrix form to make computation easy.

$$\begin{aligned}
\implies H_1 &= tanh(W_1 \begin{bmatrix} V_1 & V_2 & V_3 \end{bmatrix}) \\
&= tanh\left(\begin{bmatrix} W_{ad} & W_{bd} & W_{cd} \\ W_{ae} & W_{be} & W_{ce} \\ W_{af} & W_{bf} & W_{cf} \end{bmatrix} \begin{bmatrix} 0.2 & 2.5 & 0.75 \\ 1 & 3 & -2 \\ 3 & 7 & 3 \end{bmatrix}\right) \\
&= tanh\left(\begin{bmatrix} 0.5 & 1.3 & 0 \\ 0.9 & -1.4 & 0.3 \\ -2 & 0.75 & -1.25 \end{bmatrix} \begin{bmatrix} 0.2 & 2.5 & 0.75 \\ 1 & 3 & -2 \\ 3 & 7 & 3 \end{bmatrix}\right) \\
&= tanh\left(\begin{bmatrix} 1.4 & 5.15 & -2.225 \\ -0.32 & 0.15 & 4.375 \\ -3.4 & -11.5 & -6.75 \end{bmatrix}\right)
\end{aligned}$$

$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$. To get the tanh of the whole matrix, we apply tanh function to each element of the matrix.

$$\therefore H_1 = \begin{bmatrix} 0.885 & 0.999 & -0.976 \\ -0.309 & 0.148 & 0.999 \\ -0.997 & -0.999 & -0.999 \end{bmatrix}$$

Similarly, we get $H_2 = ReLU(W_2 H_1)$

$$\begin{aligned}
\implies H_2 &= tanh(W_2 H_1) \\
&= ReLU\left(\begin{bmatrix} W_{dg} & W_{eg} & W_{fg} \\ W_{dh} & W_{eh} & W_{fg} \end{bmatrix} \begin{bmatrix} 0.885 & 0.999 & -0.976 \\ -0.309 & 0.148 & 0.999 \\ -0.997 & -0.999 & -0.999 \end{bmatrix}\right) \\
&= ReLU\left(\begin{bmatrix} 1 & 2.5 & 3 \\ 3 & -2 & 1.25 \end{bmatrix} \begin{bmatrix} 0.885 & 0.999 & -0.976 \\ -0.309 & 0.148 & 0.999 \\ -0.997 & -0.999 & -0.999 \end{bmatrix}\right) \\
&= ReLU\left(\begin{bmatrix} -2.878 & -1.628 & -1.475 \\ 2.026 & 1.452 & -6.174 \end{bmatrix}\right)
\end{aligned}$$

$ReLU(x) = \max(0, x)$. To get the ReLU of the whole matrix, we apply ReLU function to each element of the matrix.

$$\therefore H_2 = \begin{bmatrix} 0 & 0 & 0 \\ 2.026 & 1.452 & 0 \end{bmatrix}$$

For the final output layer, $H_L = W_L H_2$ (because no activation function is given in the

question to apply for output layer)

$$\begin{aligned}
\implies H_L &= W_L H_2 \\
&= \begin{bmatrix} W_{gi} & W_{hi} \\ W_{gj} & W_{hj} \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 2.026 & 1.452 & 0 \end{bmatrix} \\
&= \begin{bmatrix} 2.5 & 0.5 \\ -1.5 & -0.25 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 2.026 & 1.452 & 0 \end{bmatrix} \\
&= \begin{bmatrix} 1.013 & 0.726 & 0 \\ -0.506 & -0.363 & 0 \end{bmatrix}
\end{aligned}$$

If tanh activation function is used for hidden layer H1 and ReLU for H2, then final output vectors will be $\begin{bmatrix} 1.013 \\ -0.506 \end{bmatrix}, \begin{bmatrix} 0.726 \\ -0.363 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

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: The decision tree for with minimum height does the following: checks truth of x_1 ($x_1 = 0/1$) in the first level, checks truth of x_2 in the second level (2 nodes) and so on, i.e. it will be a full-blown binary tree with the i^{th} level checking the truth value of x_i. $\therefore$ such a tree of height h will have 2^h leaf nodes.

To find the majority vote for any data point, we have to parse/analyse atleast half the input. $\therefore$ in the best case, we can make a decision about the majority with the help of the first $\frac{m}{2}$ features itself (eg. $\frac{m}{2}$ $1s$ followed by $\frac{m}{2}$ $0s$) or vice-versa. As analysing the first half is sufficient, the decision tree of will have a height of $\frac{m}{2}$ and $2^{\frac{m}{2}}$ leaves. In the worst case, we might need to parse the whole input before can find the majority (eg. any random input other the type mentioned above). Hence, the tree needs to have all m levels (height is m) to check all m features and will have a 2^m leaves. $\therefore$ the range of leaves is $[2^{\frac{m}{2}}, 2^m]$