CS5590 – Foundation of Machine Learning
Assignment - 3

Tushar K Raysad (BM21MTECH14004)

1. Neural Networks:

(a)The XOR function (exclusive or) returns true only when one of the arguments is true and another is false. Otherwise, it returns false. Show that a two-layer perceptron (a perceptron with one hidden layer) can solve the XOR problem. Submit a figure and a network diagram (with associated weights).

(b) x, y, and z are inputs with values -2, 5, and -4 respectively. You have a neuron q and neuron f with functions:

$$q = x - y$$

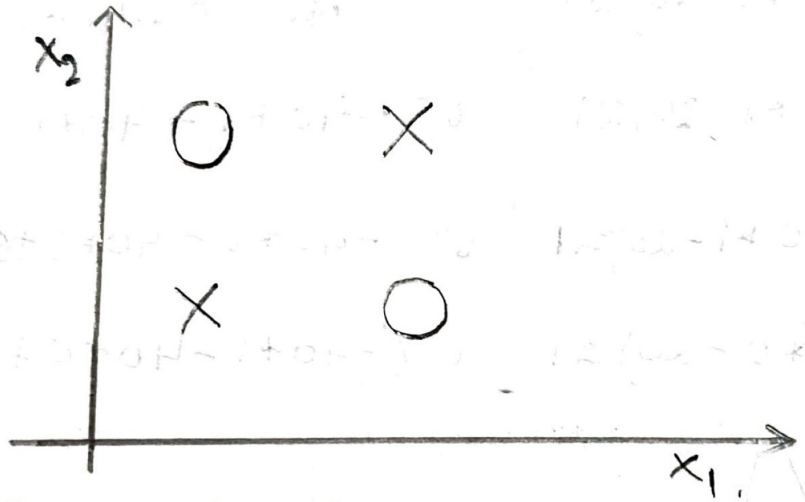
$$f = q * z$$

Show the graphical representation, and compute the gradient of f with respect to x, y, and z.

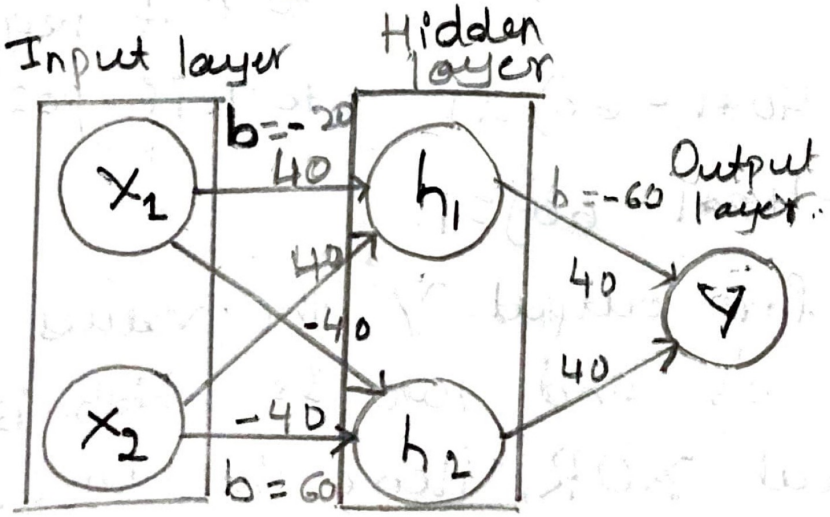
a. →

Solving XOR with two layer perceptron. [A perceptron with one hidden layer] :-

Let us consider the following diagram to be classified.



A linear classifier cannot classify this with one hyperplane. Therefore we consider the following network diagram:



Two Layer Perceptron

Let us consider above two layer perceptron to solve the classification problem [XOR problem]
Let weights be as shown in the network and b is the bias given to neurons.

Then the equations will be:

[For h_1]

$\sigma(40*0 + 40*0 - 20) \simeq 0$

$\sigma(40*1 + 40*1 - 20) \simeq 1$

$\sigma(40*0 + 40*1 - 20) \simeq 1$

$\sigma(40*1 + 40*0 - 20) \simeq 1$

[For h_2]

$\sigma(-40*0 - 40*0 + 60) \simeq 1$

$\sigma(-40*1 - 40*1 + 60) \simeq 0$

$\sigma(-40*0 - 40*1 + 60) \simeq 1$

$\sigma(-40*1 - 40*0 + 60) \simeq 1$

[For Y]

$\sigma(40*0 + 40*1 - 60) \simeq 0$

$\sigma(40*1 + 40*0 - 60) \simeq 0$

$\sigma(40*1 + 40*1 - 60) \simeq 1$

$\sigma(40*1 + 40*1 - 60) \simeq 1.$

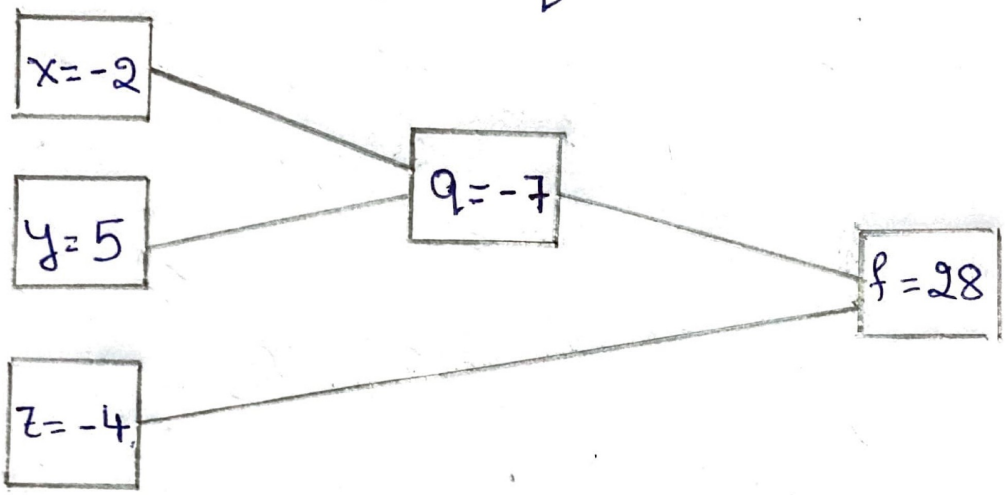
σ = Sigmoid function which equates the value to 0 if negative, to 1 if positive.

Therefore for output Y the values obtained are two 0's and two 1's which is output of logical XOR. Hence the classification is done.

b.

→ Graphical representation of the functions, $q = x - y$

$f = q * z$.



Gradient of f with respect to x, y and z,

$$\frac{df}{dx} = \frac{df}{dq} \cdot \frac{dq}{dx} = z \cdot 1 = -4$$

$$\frac{df}{dy} = \frac{df}{dq} \cdot \frac{dq}{dy} = z \cdot (1) = 4$$

$$\frac{df}{dz} = \frac{df}{dq} \cdot \frac{dq}{dz} = z \cdot 0 = 0.$$

Therefore gradients of f w.r.t x, y and z are −4, 4 and 0 respectively.

2. Neural Networks:

The extension of the cross-entropy error function for a multi-class classification problem is given by:

$$E(\mathbf{w}) = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{kn} \ln y_k(\mathbf{x}_n, \mathbf{w})$$

where K is the number of classes, N is the number of data samples, and t_n is a one-hot vector which designates the expected output for a data sample x_n (note that a one-hot vector has a 1 in the correct class' position and zeroes elsewhere, e.g. t_n = [0, 0, 1, 0, · · · , 0] for the ground truth of the 3rd class). The network outputs $y_k(x_n,w) = p(t_k = 1|x)$ are given by the softmax activation function:

$$y_k(\mathbf{x}, \mathbf{w}) = \frac{\exp(a_k(\mathbf{x}, \mathbf{w}))}{\sum_j \exp(a_k(\mathbf{x}, \mathbf{w}))}$$

which satisfies $0 \leq y_k \leq 1$ and $\Sigma_k y_k = 1$, where 'a_k's are the pre-softmax activations of the output layer neurons (also called logits). Show that the derivative of the above error function with respect to the activation a_k for an output unit having a logistic sigmoid activation function is given by:

$$\frac{\partial E}{\partial a_k} = y_k - t_k$$

→ The extension of the cross entropy error function for a multiclass classification problem:

$$E(w) = -\sum_{n=1}^{N}\sum_{k=1}^{N} t_{kn} \ln y_k(x_n, w)$$

This can be written as

$$E(w) = -\sum_{k=1}^{N} t_k \ln y_k + (1-t_k)\ln(1-y_k)$$

Differentiating w.r.t y_k,

$$\frac{\partial E(w)}{\partial y_k} = -\left[\frac{t_k}{y_k} + \frac{(1-t_k)(-1)}{(1-y_k)}\right]$$

$$\frac{\partial E(w)}{\partial y_k} = \left[-\frac{t_k}{y_k} + \frac{1-t_k}{1-y_k}\right] = \frac{1-t_k}{1-y_k} - \frac{t_k}{y_k} \quad \text{— ①.}$$

We have,

$$y_k(x, w) = \frac{\exp(a_k(x, w))}{\sum_j \exp(a_k(x, w))}$$

Differentiating w.r.t a_k

$$\frac{\partial y_k}{\partial a_k} = \frac{\sum_j \exp(a_k(x, w)).\exp(a_k) - (\exp(a_k))^2}{\left(\sum_j \exp(a_k(x, w))\right)^2}$$

$$= \frac{\exp(a_k)}{\sum_j \exp(a_k)} - \left(\frac{\exp(a_k)}{\sum_j \exp(a_k)}\right)^2.$$

$$= y_k - y_k^2$$

$$\frac{\partial y_k}{\partial a_k} = y_k(1 - y_k) \quad \text{---} \quad (2)$$

Eqn (1) can be written as,

$$\frac{\partial E(w)}{\partial y_k} = \frac{y_k(1-t_k) - t_k(1-y_k)}{y_k(1-y_k)}$$

$$\frac{\partial E(w)}{\partial y_k} = \frac{y_k - t_k}{y_k(1-y_k)}$$

From (2) we have $\frac{\partial E(w)}{\partial y_k} = \frac{y_k - t_k}{\partial y_k / \partial a_k}$.

$$\therefore \frac{\partial E(w)}{\partial y_k} \cdot \frac{\partial y_k}{\partial a_k} = y_k - t_k.$$

$$\therefore \boxed{\frac{\partial E(w)}{\partial a_k} = y_k - t_k}$$ Hence proved.

3. Ensemble Methods:

Consider a convex function f(x) = x^2. Show that the average expected sum-of-squares error

$$E_{AV} = \frac{1}{M}\sum_{m=1}^{M}\mathbb{E}_x[(y_m(x) - f(x))^2]$$

of the members of an ensemble model and the expected error

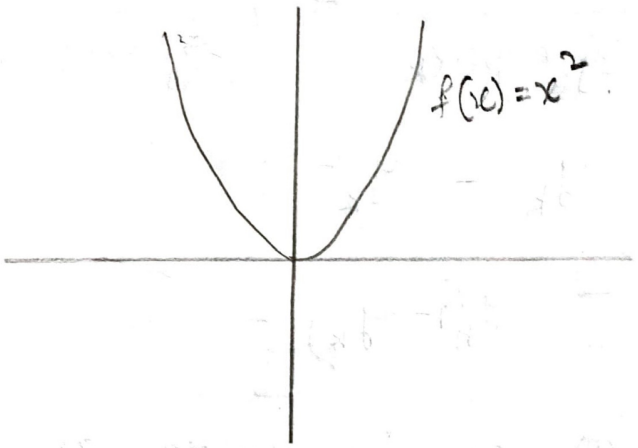
$$E_{ENS} = \mathbb{E}_x[\frac{1}{M}\sum_{m=1}^{M}(y_m(x) - f(x))^2]$$

of the ensemble satisfy:

$E_{ENS} \leq E_{AV}$

Show further that the above result holds for any error function E(y), not just sum-of-squares, as long as it is convex in y. (Hint: The only tool you may need is the Jensen's inequality. Read up about it, and use it!)

→ We have $f(x) = x^2$ which is a convex function,



Given that $f(x)$ is a convex function, we have

$$f\left(\sum_{m=1}^{M} \lambda_m x_m\right) \leq \sum_{m=1}^{M} \lambda_m f(x_m) \quad \text{—— (1)}$$

where $\lambda_m \geq 0$ and $\sum_m \lambda_m = 1$ for any set of points x_i.
If we take λ_i as the probability distribution over variable x. then above inequality can be written as

$$f(E(x)) \leq E(f(x)).$$

This is known as Jensen's inequality.

Now consider, $E_{AV} = \frac{1}{M} \sum_{m=1}^{M} E_x\left[(y_m(x) - f(x))^2\right]$

taking $\frac{1}{M}$ inside the sum and expectation operator outside the sum we get.

$$E_{AV} = E_x\left[\sum_{m=1}^{M} \frac{1}{M} (y_m(x) - f(x))^2\right]$$

Comparing $\frac{1}{M}$ to λ_m and $(y_m(x) - f(x))^2$ with $f(x_m)$ in eqn (1), we get.

$$\left(\sum_{m=1}^{M} \frac{1}{M} \sum_{m=1}^{M} y_m(x) - f(x)\right)^2 \leq \sum_{m=1}^{M} \frac{1}{M} (y_m(x) - f(x))^2$$

Since this holds for all values of x, it will hold good for expectation over x. Hence.

$$E_{ENS} \leq E_{AV}.$$

4. Random Forests:

(a)**Write your own random forest classifier (this should be relatively easy, given you have written your own decision tree code) to apply to the Spam dataset [data, information]. Use 30% of the provided data as test data and the remaining for training. Compare your results in terms of accuracy and time taken with Scikitlearn's built-in random forest classifier. (Note that you can't use in-built decision tree functions to implement your code. You can modify your decision tree code of the Assignment 1, or code a new one, to implement a random forest. You can however use the inbuilt train test split of sklearn to divide the data into train and test.)**

(b)Explore the sensitivity of Random Forests to the parameter m (the number of features used for best split).

(c)**Plot the OOB (out-of-bag) error (you have to find what this is, and read about it!) and the test error against a suitably chosen range of values for m. (Use your implementation of random forest to perform this analysis.)**

***(a)*Accuracy of random forest using sklearn with 3 trees = 0.944967414916727**

Accuracy of random forest from scratch with 3 trees= 0.9021739130434783

Time taken for random forest classifier using sklearn = 5.7 sec.
Time taken for random forest classifier built from scratch = 9 minutes (3 minutes for each tree).

***(b)*When number of features considered : 16**

Accuracy of random forest from scratch with 3 trees= 0.803623188405797

When number of features considered : 7

Accuracy of random forest from scratch with 3 trees= 0.7130434782608696

When number of features considered : 30

Accuracy of random forest from scratch with 3 trees = 0.8384057971014492

(c) Out of bag error with 1 tree = 0.12463768115942031

Out of bag error with 3 trees = 0.07608695652173902

Out of bag error with 5 trees = 0.07246376811594202

Test Error = 1 - Test Accuracy

Test Error when Number of features considered is 16 = 0.19637682

Test Error when Number of features considered is 7 = 0.28695653

Test Error when Number of features considered is 30 = 0.16159421

5. Gradient Boosting:

In this question, we will explore the use of pre-processing methods and Gradient Boosting on the popular Lending Club dataset. You are provided with two files: loan train.csv and loan test.csv. The dataset is almost as provided by the the original source, and you may have to make the necessary changes to make it suitable for applying ML algorithms. (If required, you can further divide loan train.csv into a validation set for model selection.) Your efforts will be to pre-process the data appropriately, and then apply gradient boosting to classify whether a customer should be given a loan or not. The target attribute is in the column loan status, which has values "Fully Paid" for which you can assign +1 to, and "Charged off" for which you can assign -1 to. The other records with loan status values "Current" (in both train and test) are not relevant to this problem. You can see this link to know more about the different attributes on the dataset (but please use the provided data, there are several versions of the dataset online.) Your tasks are to do the following:

(a) Pre-process the data as needed to apply the classifier to the training data (you are free to use pandas or other relevant libraries. Note that test data should not be used for pre-processing in any way, but the same pre-processing steps can be used on test data. Some steps to consider:

- **Check for missing values, and how you want to handle them (you can delete the records, or replace the missing value with mean/median of the attribute - this is a decision you must make. Please document your decisions/choices in the final submitted report.)**

- **Check whether you really need all the provided attributes, and choose the necessary attributes. (You can employ feature selection methods, if you are familiar; if not, you can eyeball.)**

- **Transform categorical data into binary features, and any other relevant columns to suitable datatypes**

- **Any other steps that help you perform better.**

(b) Apply gradient boosting using the function sklearn.ensemble.GradientBoostingClassifier for training the model. You will need to import sklearn, sklearn.ensemble, and numpy. Your effort will be focused on predicting whether or not a loan is likely to default.

- **Get the best test accuracy you can, and show what hyperparameters led to this accuracy. Report the precision and recall for each of the models that you built.**

- **In particular, study the effect of increasing the number of trees in the classifier.**

- **Compare your final best performance (accuracy, precision, recall) against a simple decision tree built using information gain. (You can use sklearn's inbuilt decision tree function for this.)**

(a) Pre-processing.

- **All the columns having attributes "NA" are deleted from both the test dataset and training dataset.**
- **In the column "term", word "months'' is removed and the values are converted to integer type.**
- **In the column "int_rate" the sign "%" is removed from all the values and then converted to float type.**
- **As mentioned in the question the target attribute is in the column loan status, which has values "Fully Paid" is assigned to +1 , and "Charged off" and "Charged Off" is assigned to -1 .And the other records with loan status values "Current" (in both train and test)are dropped as they are not relevant to this problem.**
- **The column "url" is deleted as it is irrelevant.**
- **Columns having categorical attributes are separated from columns having numerical attributes to convert them into binary type.**
- **Columns having both integer and string type data such as "issue_date", "issue_month", "last_credit_pull_date", "last_credit_pull_month" is converted integer type.**
- **Columns having all similar values are dropped.**

(b)

- `number of decision trees = 500 ,`
  `accuracy=0.9993415908810337`

	precision	recall
-1	1.00	1.00
1	1.00	1.00

- **The hyperparameters led to this accuracy are the default parameters of the GradientBoostingClassifier class of sklearn, except for n_estimators. List of hyperparameters and their values: earning_rate=0.1 (shrinkage). n_estimators=500 (number of trees). max_depth=3. min_samples_split=2. min_samples_leaf=1. subsample=1.0.**

- **number of decision trees = 25**
  accuracy = 0.9857207522324184

```
              precision    recall

          -1       1.00      0.90
           1       0.98      1.00
```

number of decision trees = 50
accuracy = 0.9920990905724044

```
              precision    recall

          -1       1.00      0.94
           1       0.99      1.00
```

number of decision trees = 100
accuracy = 0.9969960083947162

```
              precision    recall

          -1       1.00      0.98
           1       1.00      1.00
```

number of decision trees = 500
accuracy = 1.0

```
              precision    recall

          -1       1.00      1.00
           1       1.00      1.00
```

```
number of decision trees = 1000
accuracy = 1.0
                precision    recall

          -1       1.00      1.00
           1       1.00      1.00
```

- **A simple decision tree built using information gain-**

```
   accuracy = 1.0
               precision    recall

          -1       1.00      1.00
           1       1.00      1.00
```