Assignment 1

Name : Kartik Srinivas

Roll No : ES20BTECH11015

The file structure is Modular. The general Utility functions, such as the ones for getting Linear or Non - Linearly separable datasets can be found here `Lib/Utils/Dataset.py` and `Lib/Utils/utility.py` Feature maps have been defined in `Lib/Utils/feature_map.py`

Requirements:

```
import numpy as np
from tqdm import tqdm # for progress bar
```

Section 1 Overall file to be run is `Perceptron.ipynb`

The Main idea behind the perceptron is present in `Lib/Utils/Models/Perceptron.py` The rest can be seen in the plot, The spearability is implemented as the split distance of the normal distribution that generates the data, It is quite visible that the number of iterations decreases as the separability gets high.

Section 2 Overall file to be run is `HingeClassifier.ipynb` The loss implemented is the hinge Loss. The hinge loss has an optimization problem that will converge. The hinge loss can be seen as a 'soft' version of the perceptron loss. Using Logistic loss however theoretically will yield no convergence(for linearly separable data) (if w^* is an optimal solution then so is $2w^*$ but the loss is lesser for the latter.) Now, in order to see the gradients, one can see the `Lib/Utils/Models/SVM_non_reg.py` file, that contains the main information about the hinge loss update. Here is a short version

$$w' = w - \alpha \frac{\partial L}{\partial w}$$

$$L = \sum_{i=1}^{n} \max(0, 1 - y_i(w^T \phi(X_i)))$$

$$L = \sum_{i|y_i w^T \phi(X_i)<1}^{n} 1 - y_i(w^T \phi(X_i))$$

Basically the the classifier is only built by points that are within the margin of $y_i W^t \phi(X_i) = 1$. From this we get

$$w' = w + \alpha \sum_{i|y_i w^T \phi(X_i)<1}^{n} y_i \phi(X_i)$$

This algorithm is tremendously similar to the perceptron, only that the learning rate is smoothening the growth of the weights.

Section 3

Overall file to be run is `NeuralNet.ipynb` The overall picture that is needed can be seen directly from the `Lib/Utils/Models/Neural_net.py` file. The layers are defined in `Lib/Gates/Layers.py` losses are defined in `Lib/Losses/Loss.py` and the Solver has been implemented in `Lib/Solvers/GD.py`. The batch sampler has been defined in `Lib/Utils/batch_sampler.py` The loss used is a NLL + Softmax (which is equivalent to binary cross entropy for two classes)

Observations: The decision boundary that has been learnt is giving us 100% classification accuracy on the 0 contour level. Unlike the first two methods, the NN is able to learn a Non linear decision boundary, because of its complicated feature map