ML_HW4

Javad Kavian
810100103

July 28, 2024

Question 1

A

There are several reasons for which we use activation functions in neural networks:

1. Introducing non linearity to model:No matter how much we increase depth of the network, all the neurons do linear operations and if we don't add non linearity using activation functions, the model will not be able to represent non linear data
2. Introducing Sparsity: Some activation functions, like ReLU, introduce sparsity in the network by setting negative values to zero. This can lead to more efficient representations and faster computations, as only a subset of neurons are active at any given time.
3. Biological Inspiration: Many activation functions are inspired by biological neurons. For example, the sigmoid function mimics the firing rate of a biological neuron, providing a smooth transition between different activation states.

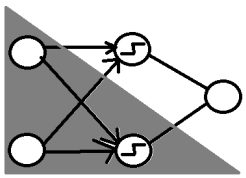
B

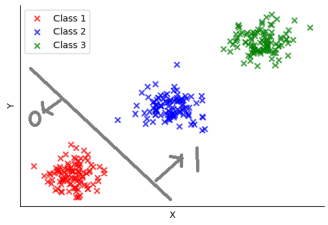
The problem of sigmoid activation function is the problem of vanishing gradients.The value of sigmoid function changes very smoothly in very high or very short value and thus the gradients are close to zero which which leads to slow convergence of the model.One activation function which can solve this problem is the **ReLU** activation function which only activates for positive inputs which helps to prevent the vanishing of the gradients.

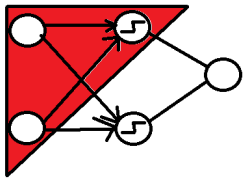
C

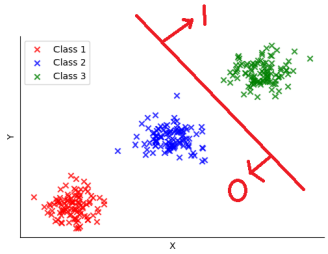
As we can see, classes can be seperated with two lines;thus we can use two neoruns to in the hidden layer to figure out whether the samples are located

under the line or above it using **step** activation function and finally use a neuron in the output layer to distinguish the class analyzing the position of samples with respect to the two lines mentioned earlier:





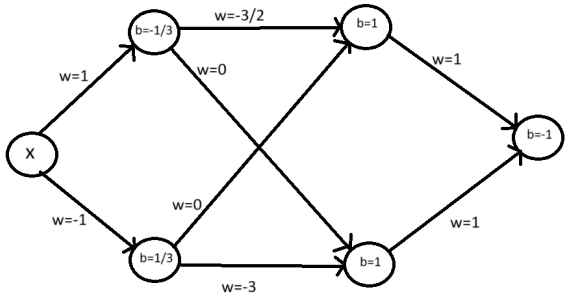




and finally the output neorun can simply add the output of the two neurons of the hidden layer with $weight = 1$ and add it with a $bias = 1$ to get the index of the class of the data

C

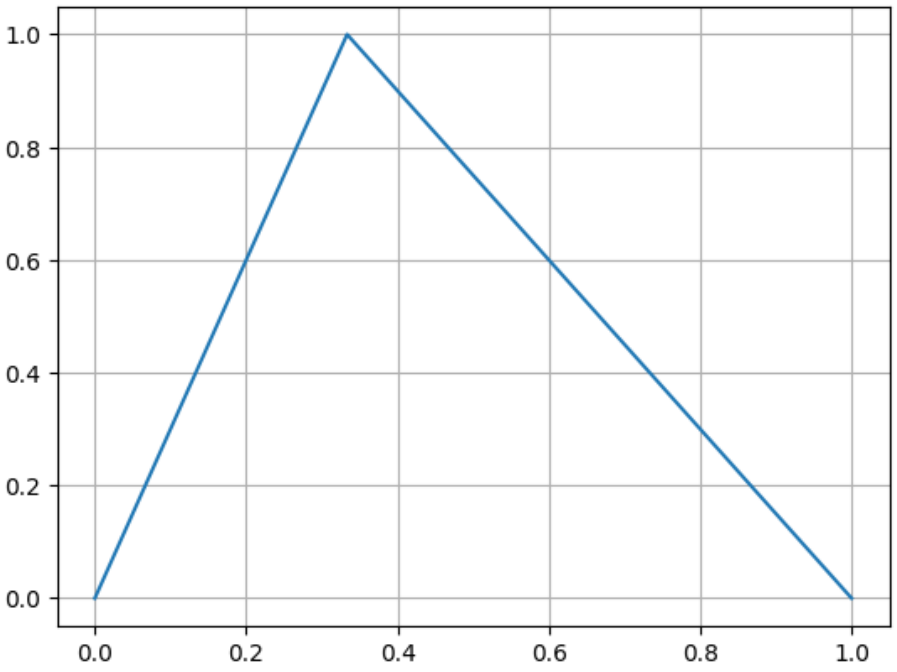
As we see, the function is constructed of two lines;thus we need two hidden layers.the first layer simply infers whether the input is less than $\frac{1}{3}$ or not and the second layer constructs the lines.The calculated weights and bias are as follows:



and with the following python code, we can verify its truthness:

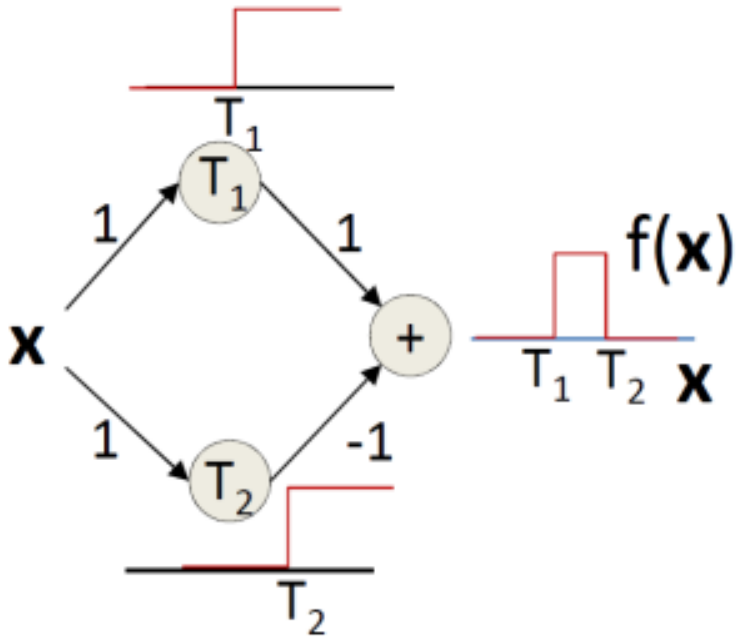
```
import numpy as np
import matplotlib.pyplot as plt
def ReLU(x):
    return np.maximum(x, 0)

def N1(x):
    return ReLU(x-1/3)


def N2(x):
    return ReLU(-x + 1/3)

def N3(n1, n2):
    return ReLU((-3/2)*n1 + 1)

def N4(n1, n2):
    return ReLU(-3*n2 + 1)

def N5(n3, n4):
    return ReLU(n3+n4 - 1)

x = np.linspace(0, 1, 100)
n1 = N1(x)
n2 = N2(x)
n3 = N3(n1, n2)
n4 = N4(n1, n2)
y = N5(n3, n4)

plt.plot(x, y)
plt.grid()
plt.show()
```

which gives as output the following figure:

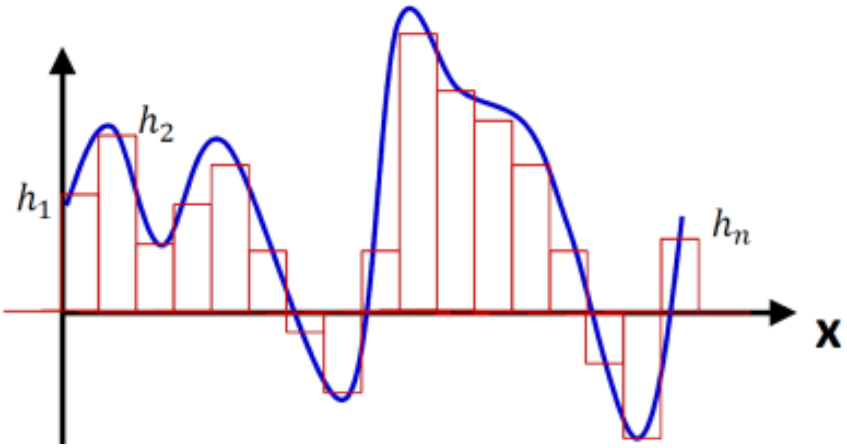


D

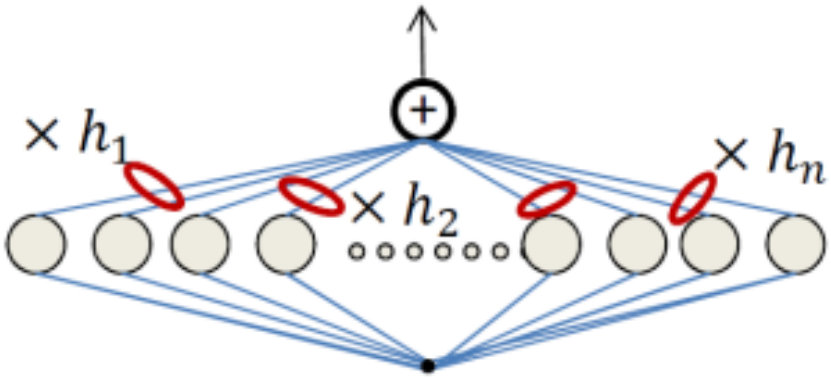
An MLP with one hidden layer and a summing output unit can generate a square pulse:



We can divide any arbitrary continous function to many number of square pulses:

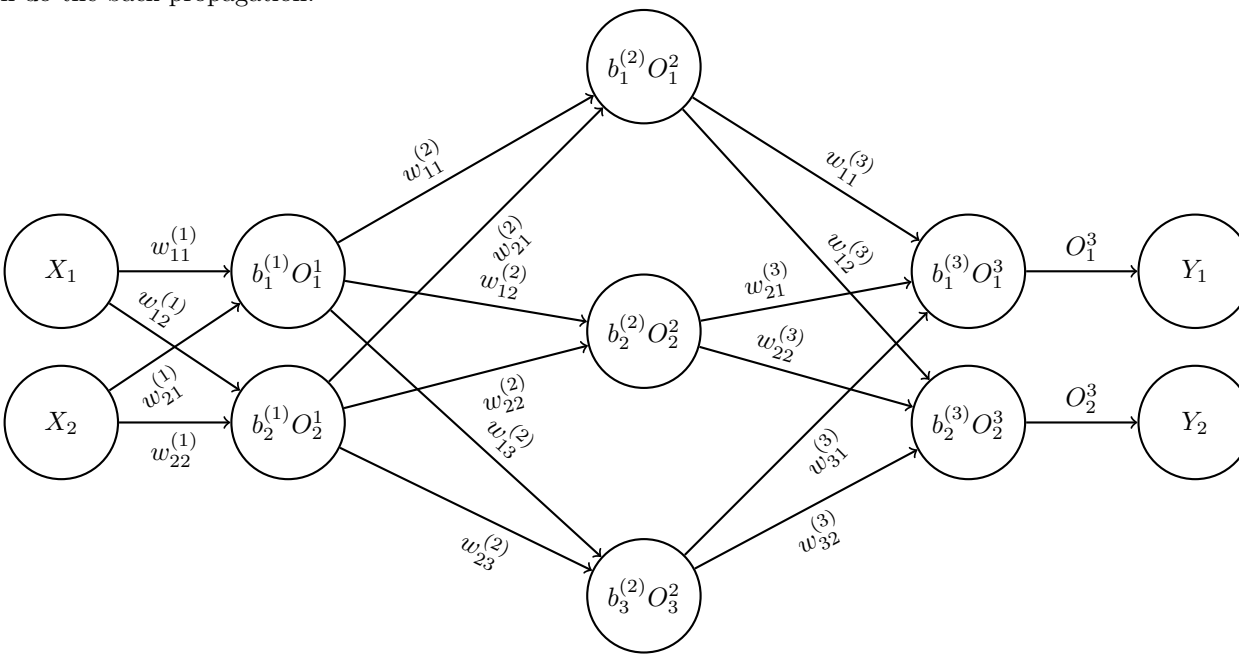


and we can model each of these square pulses with the MLP unit shown in figure.The final MLP would be something like:



Question 2

In the following network, we first do a forward pass to calculate the error and then do the back propagation:



Forward

Layer 1

$$O_1^1 = \frac{1}{1 + e^{-(X_1 w_{11}^{(1)} + X_2 w_{21}^{(1)} + b_1^{(1)})}}$$

$$O_2^1 = \frac{1}{1 + e^{-(X_1 w_{12}^{(1)} + X_2 w_{22}^{(1)} + b_2^{(1)})}}$$

Layer 2

$$O_1^2 = \frac{1}{1 + e^{-(O_1^1 w_{11}^{(2)} + O_2^1 w_{21}^{(2)} + b_1^{(2)})}}$$

$$O_2^2 = \frac{1}{1 + e^{-(O_1^1 w_{12}^{(2)} + O_2^1 w_{22}^{(2)} + b_2^{(2)})}}$$

$$O_3^2 = \frac{1}{1 + e^{-(O_1^1 w_{13}^{(2)} + O_2^1 w_{23}^{(2)} + b_3^{(2)})}}$$

Layer 3

$$z_1 = O_1^2 w_{11}^{(3)} + O_2^2 w_{21}^{(3)} + O_3^2 w_{31}^{(3)} + b_1^{(3)}$$

$$z_2 = O_1^2 w_{12}^{(3)} + O_2^2 w_{22}^{(3)} + O_3^2 w_{32}^{(3)} + b_2^{(3)}$$

$$O_1^3 = \frac{e^{z_1}}{e^{z_1} + e^{z_2}}$$

$$O_2^3 = \frac{e^{z_2}}{e^{z_1} + e^{z_2}}$$

Loss Calculation

Here we are using cross entropy loss which is formulated as below in this example:

$$\mathcal{L}oss = Y_1 \log O_1^3 + Y_2 \log O_2^3$$

Back Propagation

We start calculating the derivative of loss with respect to output of layers and using chaing rule, go back untill we reach the first layer:

$$\frac{\partial \mathcal{L}oss}{\partial O_1^3} = \frac{Y_1}{O_1^3} \tag{1}$$

$$\frac{\partial \mathcal{L}oss}{\partial O_2^3} = \frac{Y_2}{O_2^3} \tag{2}$$

$$\frac{\partial \mathcal{L}oss}{\partial z_1} = \frac{\partial \mathcal{L}oss}{\partial O_1^3} \frac{\partial O_1^3}{\partial z_1} = \frac{Y_1}{O_1^3} \frac{e^{z_1+z_2}}{(e^{z_1} + e^{z_2})^2} \tag{3}$$

$$\frac{\partial \mathcal{L}oss}{\partial z_2} = \frac{\partial \mathcal{L}oss}{\partial O_2^3} \frac{\partial O_2^3}{\partial z_2} = \frac{Y_2}{O_2^3} \frac{e^{z_1+z_2}}{(e^{z_1} + e^{z_2})^2} \tag{4}$$

$$\frac{\partial \mathcal{L}oss}{\partial O_1^2} = \frac{\partial \mathcal{L}oss}{\partial z_1}\frac{\partial z_1}{\partial O_1^2} + \frac{\partial \mathcal{L}oss}{\partial z_2}\frac{\partial z_2}{\partial O_1^2}$$

$$= \frac{\partial \mathcal{L}oss}{\partial z_1} w_{11}^{(3)} + \frac{\partial \mathcal{L}oss}{\partial z_2} w_{12}^{(3)} \tag{5}$$

and with the same method, we have:

$$\frac{\partial \mathcal{L}oss}{\partial O_2^2} = \frac{\partial \mathcal{L}oss}{\partial z_1} w_{21}^{(3)} + \frac{\partial \mathcal{L}oss}{\partial z_2} w_{22}^{(3)} \tag{6}$$

$$\frac{\partial \mathcal{L}oss}{\partial O_3^2} = \frac{\partial \mathcal{L}oss}{\partial z_1} w_{31}^{(3)} + \frac{\partial \mathcal{L}oss}{\partial z_2} w_{32}^{(3)} \tag{7}$$

we have already calcualted the value of $\frac{\partial \mathcal{L}oss}{\partial z_1}$ and $\frac{\partial \mathcal{L}oss}{\partial z_2}$ in equations (3) and (4).Now lets move one layer back:

$$\frac{\partial \mathcal{L}oss}{\partial O_1^1} = \frac{\partial \mathcal{L}oss}{\partial O_1^2}\frac{\partial O_1^2}{\partial O_1^1} + \frac{\partial \mathcal{L}oss}{\partial O_2^2}\frac{\partial O_2^2}{\partial O_1^1} + \frac{\partial \mathcal{L}oss}{\partial O_3^2}\frac{\partial O_3^2}{\partial O_1^1}$$

$\frac{\partial \mathcal{L}oss}{\partial O_1^2}$, $\frac{\partial \mathcal{L}oss}{\partial O_2^2}$ and $\frac{\partial \mathcal{L}oss}{\partial O_3^2}$ can be obtained from equations (5), (6) and (7).The other derivative terms have sigmoid form whose derivative looks like : $g(z)(1 - g(z))$.thus the above equation can be wrriten as:

$$\frac{\partial \mathcal{L}oss}{\partial O_1^1} = \frac{\partial \mathcal{L}oss}{\partial O_1^2}(w_{11}^{(2)} O_1^2(1-O_1^2)) + \frac{\partial \mathcal{L}oss}{\partial O_2^2}(w_{12}^{(2)} O_2^2(1-O_2^2)) + \frac{\partial \mathcal{L}oss}{\partial O_3^2}(w_{13}^{(2)} O_3^2(1-O_3^2)) \tag{8}$$

now that we have $\frac{\partial \mathcal{L}oss}{\partial O_1^1}$, we can easily calculate $\frac{\partial \mathcal{L}oss}{\partial w_{11}^{(1)}}$ and $\frac{\partial \mathcal{L}oss}{\partial b_1^{(1)}}$:

$$\frac{\partial \mathcal{L}oss}{\partial O_1^1} = \frac{\partial \mathcal{L}oss}{\partial O_1^1}\frac{\partial O_1^1}{\partial w_{11}^{(1)}} = \frac{\partial \mathcal{L}oss}{\partial O_1^1}[X_1 O_1^1(1 - O_1^1)] \tag{9}$$

$$\frac{\partial \mathcal{L}oss}{\partial b_1^{(1)}} = \frac{\partial \mathcal{L}oss}{\partial O_1^1}\frac{\partial O_1^1}{\partial b_1^{(1)}} = \frac{\partial \mathcal{L}oss}{\partial O_1^1}[O_1^1(1 - O_1^1)] \tag{10}$$

and also the value of $\frac{\partial \mathcal{L}oss}{\partial O_1^1}$ is calcualted from equation (8).

Stochastic gradient descent

Stochastic gradient descent is a gradient based optimization algorithm in which the objective is to optimize some weights according to a loss function and a number of samples.Given n samples as $x_1, x_2, \ldots, x_n$ and the target variables $y_1, y_2, \ldots, y_n$ and loss function $\mathcal{L}$, the algorithm works as below:

for i=1 to n:

$$w =: w - \eta \mathcal{L}(y_i, f(x_i))x_i$$

Newton-Raphson method

Suppose we want to optimize the function f;given an arbitrary point θ, we update θ to the intersection of the tangent of function in point θ and x-axis and do this in many epochs untill we reach the optimum point:

$$\theta =: \theta - \frac{f(\theta)}{f'(\theta)}$$

Question 3

A

Translational invariance in Convolutional Neural Networks refers to the ability of the network to recognize objects in an image regardless of their position and angle. This is achieved through convolutional layers which share weights and apply in different parts of the objective and can consider spatial features of that.

B

1. Convolutional Layers:In CNNs, the same set of filters are applied across the entire input. This means that the learned features (such as edges or textures) are detected regardless of their position or angle. When a filter detects a specific pattern in one part of the image, it can detect the same pattern in any other part due to the shared weights.

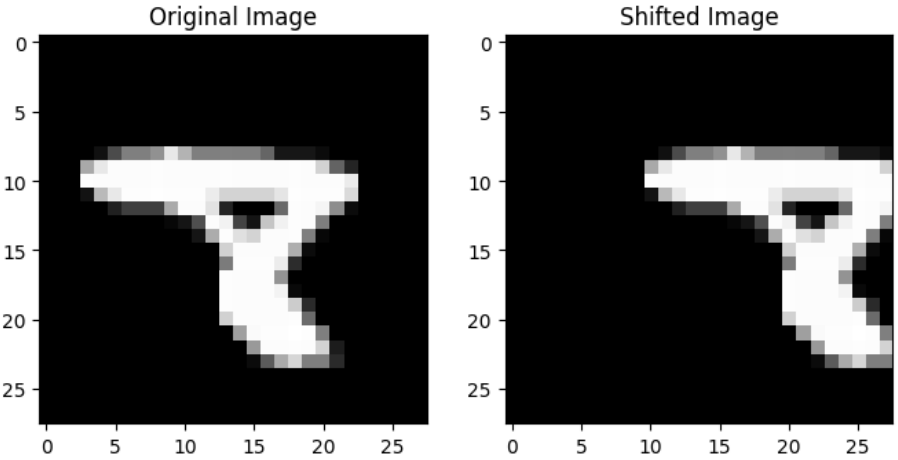
2. Pooling Layers:Pooling layers (typically max pooling or average pooling) reduce the spatial dimensions of the feature maps, which helps in achieving translational invariance. Max pooling, for instance, selects the maximum value within a pooling window, ensuring that the most prominent features are preserved even if their positions vary slightly.

C

Codes of this section are in Q3.ipynb file.The MLP designed with the following architecture got 93.08% accurcy:

```
    fc1 = nn.Linear(784, 300)
    fc2 = nn.Linear(300, 50)
    softmax_layer = nn.Linear(50, 10)
```

Moving forward and implementing LeNet and training it for 30 eopchs (runned the training cell 3 times), the model got 98.5% accuracy.But the power of CNNs is not limited to high accuracy and now we want to test the transition invariancy of them by applying a transformation on an image to see whether the model can still predict it or not. Here is the original image and its shifted image whose label is **8**:

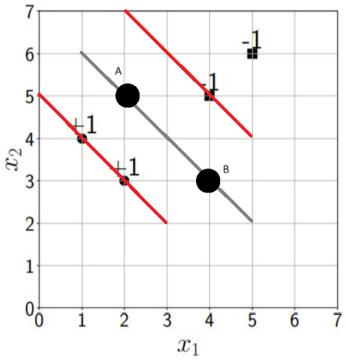


but as we can see in the notebook file, the MLP recognized it as 7 while CNN could easily classifiy it correctly.

Question 4

A

If we draw support vectors as shown in the below figure, the decision bound will be the gray line;considering two arbitrary points A and B, we can calculate weights and bias:



$$X_2 - X_2^A = \frac{X_2^A - X_2^B}{X_1^A - X_1^B}(X_1 - X_1^A)$$

$$\Rightarrow X_2 + X_1 - 7 = 0 \tag{11}$$

Now to verify our answer, we have two approaches.As we know, solving SVM problem is actually solving a convex optimization problem.To solve with this

method, we used the library **cvxpy** in the section **Quadratic form** of the Q4.ipynb file.Another method is to use the **Sklearn** library which has the SVM classifier as SVC module. This method is implemented in the section **Sklearn** of the Q4.ipynb file.Both of this methods gave the following answer:

$$W = [-0.5, -0.5], bias = 3.49$$

If we multiply the equation (11) with -0.5, we have:

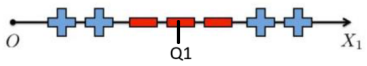
$$-0.5X_1 - 0.5X_2 + 3.5 = 0$$

which is the same as solved by python.

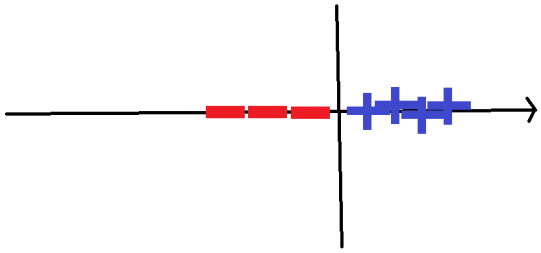
B

DB 1

Consider the point Q_1:

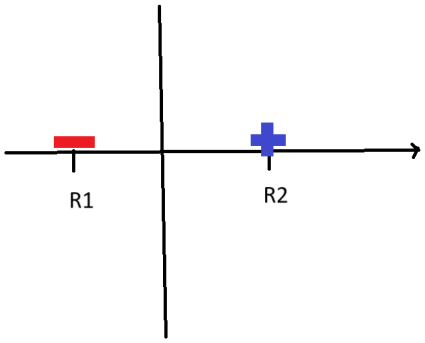


If we consider the transformation: $\phi(X) = |X - Q_1|$, the transformed data will be like the following figure which is linearly seperable:



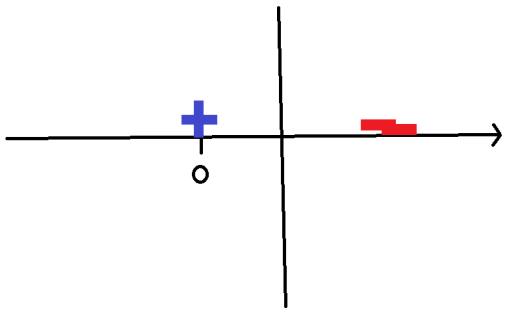
DB 2

As the datas have circular form, we can easily use the transformation:$\phi(X_1, X_2) = X_1^2 + X_2^2$.If we consider radius of negative samples as R_1 and radius of positive samples as R_2, the transfromed data will look like the following figure which is linearly seperable:



DB 3

We can use a combination of two previous databases to transform these samples to a space which are linearly seperable.If we consider radius of positive samples as R, then the transformed data will look like the following figure which is linearly seperable under the transformation:$|X_1^2 + X_2^2 - R|$



Question 5

Kernel is a function which gives as output the dot product of points in a transformed space without the need to have the transition function explicitly.In many machine learning models, kernel trick is used to avoid complexity of the trans-

formed space.It is a technique that allows machine learning algorithms to operate in a high-dimensional, implicit feature space without ever computing the coordinates of the data in that space. Instead, it relies on computing the inner products between the images of all pairs of data in the feature space, directly in the input space using the kernel function.

If K is a valid kernel, there exists a transfromation ϕ in a way that we have:

$$K(x,y) =< \phi(x), \phi(y) >$$

$$K(x,x) =< \phi(x), \phi(x) >$$

$$K(y,y) =< \phi(y), \phi(y) >$$

By applying the **Cauchy-Schwarz inequality** to the transformed points, we have:

$$| < \phi(x), \phi(y) > |^2 \leq < \phi(x), \phi(x) > . < \phi(y), \phi(y) >$$

replacing the values with kernel values, we have:

$$K(x,y)^2 \leq K(x,x)K(y,y)$$

Now we want to calculate the average in the transformed space.we have:

$$||\mu_\phi||^2 = \mu_\phi^T \mu_\phi$$

$$\mu_\phi = \frac{1}{Q}\sum_{i=1}^{Q} x_i$$

$$\mu_\phi^T = \frac{1}{Q}\sum_{i=1}^{Q} x_i^T$$

$$\Rightarrow \mu_\phi^T \mu_\phi = \frac{1}{Q^2}(\sum_{i=1}^{Q} x_i^T)(\sum_{i=1}^{Q} x_i)$$

$$= \frac{1}{Q^2}\sum_{m=1}^{Q}\sum_{n=1}^{Q} x_m^T . x_n$$

$$= \frac{1}{Q^2}\sum_{m=1}^{Q}\sum_{n=1}^{Q} < x_m, x_n >$$

$$= \frac{1}{Q^2}\sum_{m=1}^{Q}\sum_{n=1}^{Q} K(x_m, x_n)$$

$$\Rightarrow ||\mu_\phi|| = \frac{1}{Q}\sqrt{\sum_{m=1}^{Q}\sum_{n=1}^{Q} K(x_m, x_n)}$$

Question 6

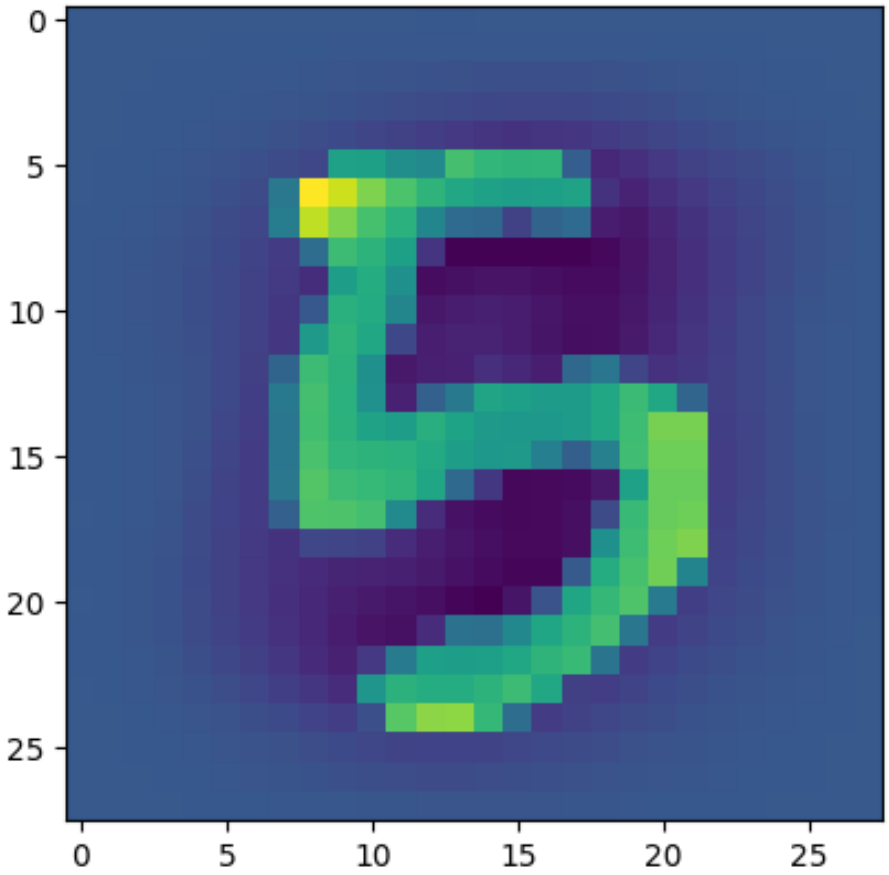
Codes of this section are stored in Q6.ipynb file.At first we divided data into train and test with 80-20 ratio;then from train data, we extracted validation data in a way that about 100 samples are driven from each label.This validation data is used to find optimal parameters of SVM classifier.

To find optimal parameters, we used gridSearch with accuracy score and fitting the grid model to validation data, we got the following scores and parameters:

kernel	accuracy	C	gamma
polynomial	85	0.1	0.1
linear	86	0.1	0.0001
rbf	88	10	0.0001

Table 1: grid search result

As shown in the above table, the best model is the rbf which has 4% training error and 6% test error which is equal to 94% accuracy.Comparing it with logistic regression, we see that it has 91% accuracy on test set data and the following image is missclassified by it while it is correctly classified by SVM:


0
5
10
15
20
25
0
5
10
15
20
25