DSC 275/475: Time Series Analysis and Forecasting (Fall 2022)

Project 2 – Feedforward Neural Networks

Total points: 30 (undergraduate students); 40 (graduate students; and with extra credit for undergraduate students)

Submitted by:

Aradhya Mathur

Lakshmi Nikhil Goduguluri

Instructions:

- You are welcome to work on this project individually or in teams (up to 2 members in each team max).
- If you plan to use PyTorch, a good resource is to review and modify the attached example code (we plan to review this example code in class as well) for feedforward networks. For this project, you will need to modify the following sections of the example code for each question:
    - Network configuration (number of layers and neurons) in the class "Net"
    - Activation function ("relu" is recommended)
    - Decision boundary limits in the function "plot_decision_boundary" as per each question
    - Learning rate "learning_rate"
    - Number of epochs and/or apply a stopping condition when the desired accuracy and/or loss is achieved
    - Optimizer: "torch.optim.SGD" or "torch.optim.Adam". The latter may provide faster convergence

Overview

In this project, you will create and train feedforward neural networks on synthetic, separable data. Although traditional machine learning tasks involve measuring performance on sets of data not seen by the network during training (e.g., test data), in this project we will mostly be concerned with architecture, hyperparameter and learning parameter tuning to maximize performance on the training set. This is an often-overlooked aspect of machine learning: practitioners seldom verify their algorithms have the right capacity for the data by examining their behavior on the training set. In this project, all performance metrics of interest will be computed on the training data.

In order to construct networks that are capable of performing the tasks being posed, you will need to adjust parameters such as learning rate, type of network activation function, number of layers and number of neurons per layer. Lastly, you will need to monitor the loss and the accuracy on the training set as the learning takes place.

Please make sure to hand in a pdf/Word/text document with the final version of your code as well as answers to the questions below. In addition, please submit the code file also. Points highlighted in red denote problems *required* for graduate students and *extra credit* optional submissions for undergraduate students.

1. Classification of XOR data (15 points)

At first sight, it may seem as if separating the XOR data is a simple task. However, due to the fact that the data is not linearly separable and that fitting the data requires non-trivial learning, the XOR problem has been a case study of interest on many topics related to training of feedforward networks. In the 1960s, Minsky and Papert's observations that perceptrons (neural network ancestors) were unable to fit the XOR data contributed to the rise of the first AI winter. XOR data makes for such an interesting case study that papers describing learning properties of networks trained on it are still being published to this day.

Create arrays containing the input data and the corresponding output labels for the XOR operator. Recall that XOR takes as input two binary variables, and outputs a 0/1 if they have the same/different value. Your XOR data should look like the following table:

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

In other words, the XOR training includes four, two-dimensional data samples with labels. Create and train a network with at least three hidden layers that separates the XOR data, that is, a network that gets 100% performance on the four training samples above. Monitor the accuracy on the training set as the training progresses.

(a) Plot the decision boundaries of the earliest network in the training process that achieves 100% accuracy by plotting the network outputs in a densely sampled region around [-0.5,1.5] × [-0.5,1.5]. *(5 points)*

Network:

```
class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(2, 2) # Layer 1
        self.fc2 = nn.Linear(2, 2) # Layer 2
        self.fc3 = nn.Linear(2, 2) #Layer 3
        #self.fc4 = nn.Linear(3, 3)
        self.fc5 = nn.Linear(2, 2)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = F.relu(self.fc3(x))
        #x = F.relu(self.fc4(x))
        x = self.fc5(x)
        return F.log_softmax(x)
        #return F.softmax(x)
```

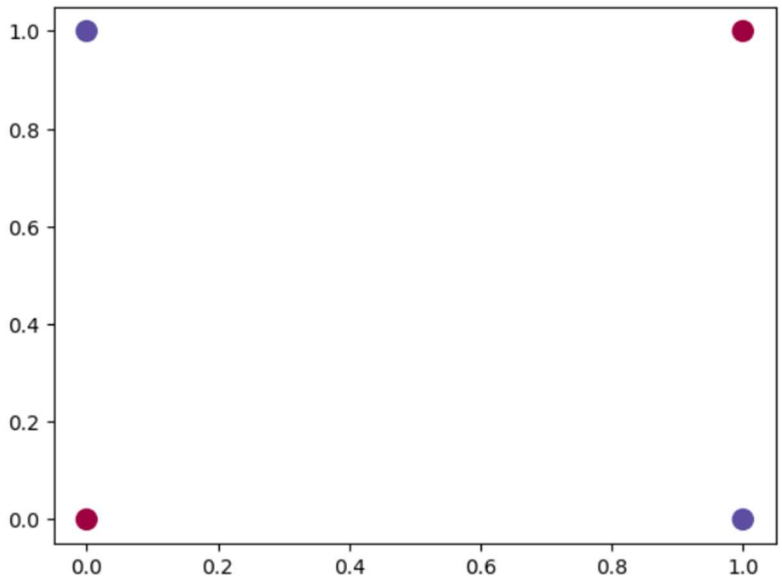

Fig 1: Scatter Plot

Results:

```
Training accuracy is  0.75
Epoch  60 Loss  0.6620653867721558
Training accuracy is  0.75
Epoch  70 Loss  0.6267451047897339
Training accuracy is  0.75
Epoch  80 Loss  0.5702000260353088
Training accuracy is  0.75
Epoch  90 Loss  0.5063083171844482
Training accuracy is  0.75
Epoch  100 Loss  0.4361491799354553
Training accuracy is  0.75
Epoch  110 Loss  0.35086917877197266
Training accuracy is  1.0
```

We got Accuracy 1 at Epoch 110, Loss is 0.3508

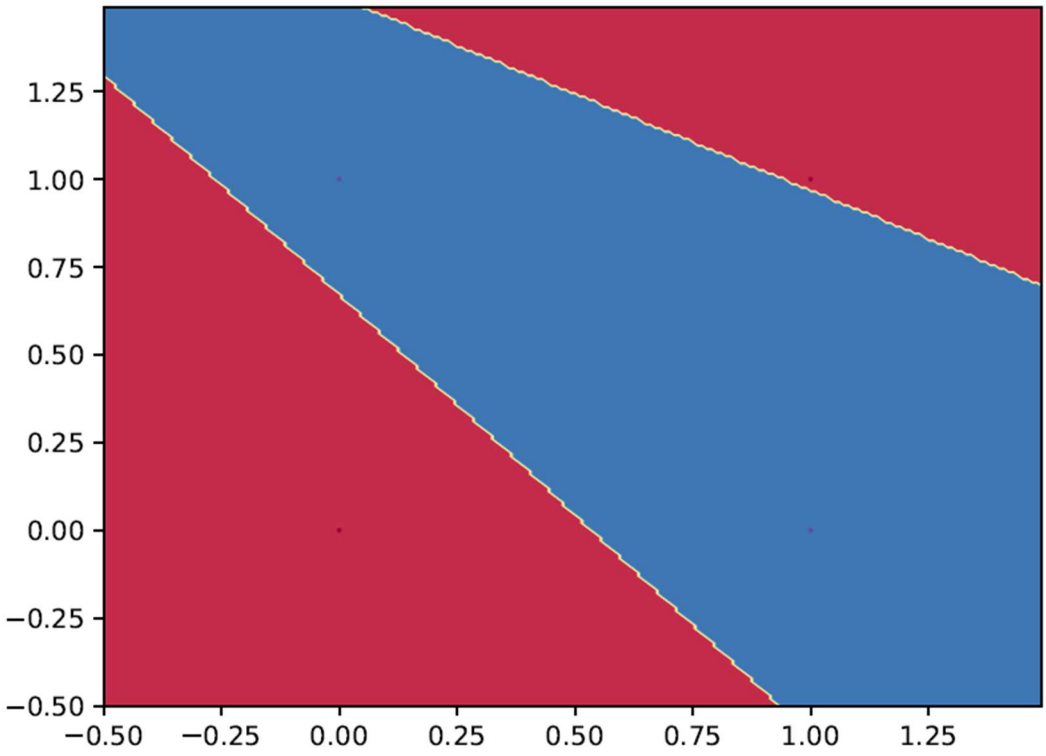

Fig 2. Decision Boundaries

(b) Plot the decision boundaries of a network after the loss falls below 1×10^{-4}. *(5 points)*

Network:

```
class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(2, 4)
        self.fc2 = nn.Linear(4, 4)
        self.fc3 = nn.Linear(4, 4)
        #self.fc4 = nn.Linear(4, 4)
        self.fc5 = nn.Linear(4, 2)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = F.relu(self.fc3(x))
        #x = F.relu(self.fc4(x))
        x = self.fc5(x)
        return F.log_softmax(x)
        #return F.softmax(x)
```

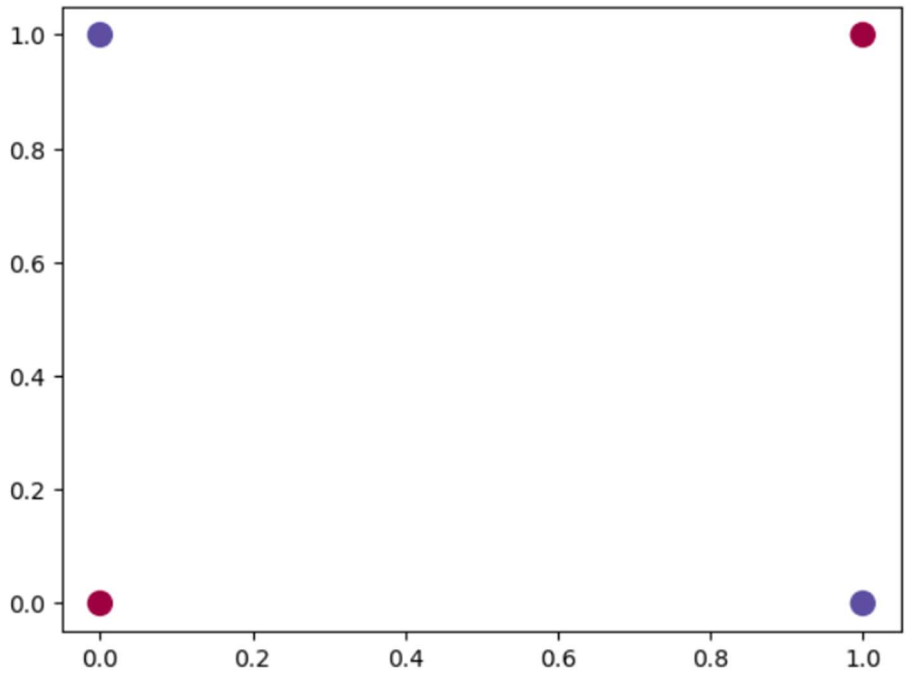

Fig 3: Scatter Plot

Results:

```
Epoch  260 Loss  0.0001663995499256998
Training accuracy is  1.0
Epoch  270 Loss  0.0001517395576229319
Training accuracy is  1.0
Epoch  280 Loss  0.0001388373784720897
Training accuracy is  1.0
Epoch  290 Loss  0.0001274844835279509
Training accuracy is  1.0
Epoch  300 Loss  0.0001174127610283903
Training accuracy is  1.0
Epoch  310 Loss  0.0001085328622139059
Training accuracy is  1.0
Epoch  320 Loss  0.0001005170488497242
Training accuracy is  1.0
Epoch  330 Loss  9.342491102870554e-05
```

We got desired output at Epoch 330.

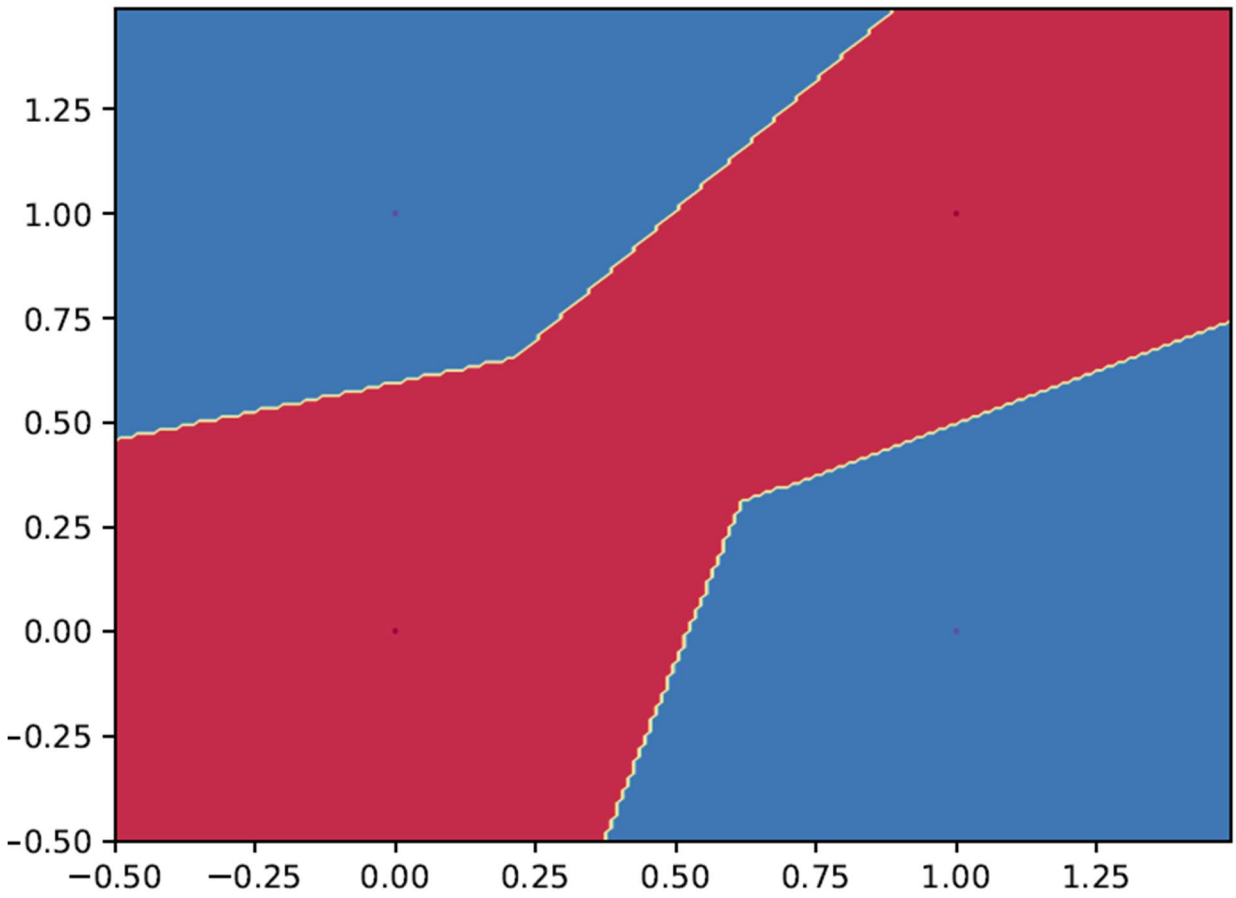

Fig 4: Decision Boundaries

(c) Gradually decrease the capacity of the network above. Find the smallest network that can still separate the data, i.e., find the <u>least</u> number of hidden layers and neurons that produces an accuracy of 1 on the training set? *(5 points) [A portion of the total points is allocated to your rank amongst your peers in achieving the smallest network]*

class Net(nn.Module):

We used 3 hidden layers (were able to get accuracy = 1 with less than 3 layers but chose to have 3 hidden layers as it was mentioned in the question). Only 2 neurons were required in the process.

Network:

```
class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(2, 2) #Hidden layer 1
        self.fc2 = nn.Linear(2, 2) #Hidden layer 2
        self.fc3 = nn.Linear(2, 2) #Hidden layer 3
        self.fc4 = nn.Linear(2, 2)
    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = F.relu(self.fc3(x))
        x = self.fc4(x)
        return F.log_softmax(x)
        #return F.softmax(x)
```

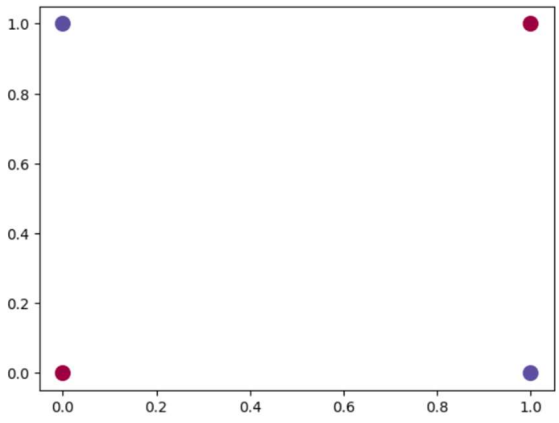

Fig 5: Scatter Plot

Results:

```
Epoch  0 Loss  0.7529647350311279
Training accuracy is  0.5
Epoch  10 Loss  0.6938625574111938
Training accuracy is  0.75
Epoch  20 Loss  0.6936883330345154
Training accuracy is  0.5
Epoch  30 Loss  0.6893919706344604
Training accuracy is  0.5
Epoch  40 Loss  0.682646632194519
Training accuracy is  0.5
Epoch  50 Loss  0.6749989986419678
Training accuracy is  0.5
Epoch  60 Loss  0.6615309715270996
Training accuracy is  0.5
Epoch  70 Loss  0.6377838850021362
Training accuracy is  0.75
Epoch  80 Loss  0.6046977639198303
Training accuracy is  0.75
Epoch  90 Loss  0.5577099919319153
Training accuracy is  0.75
Epoch  100 Loss  0.4990972578525543
Training accuracy is  0.75
Epoch  110 Loss  0.4278269112110138
Training accuracy is  0.75
Epoch  120 Loss  0.323006272315979
Training accuracy is  1.0
```

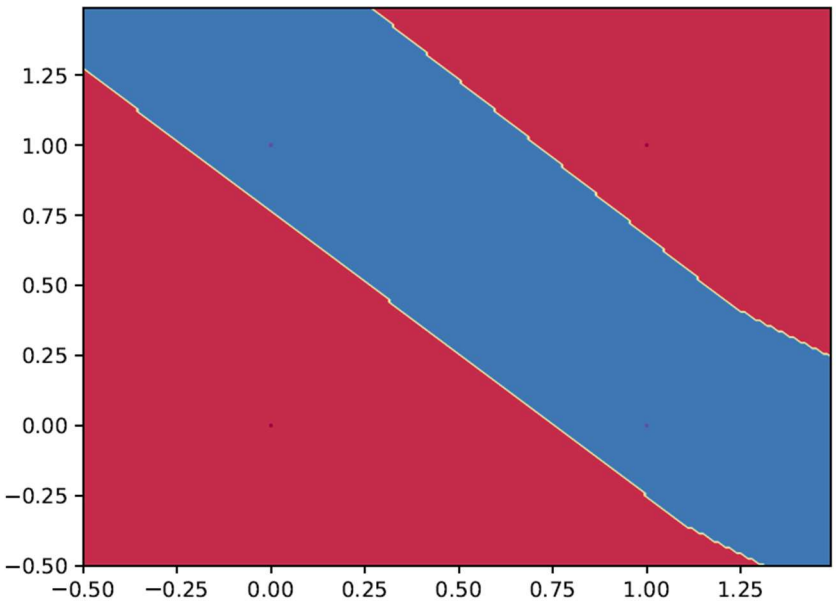

Fig 6: Decision Boundaries

2. Classification of Separable, Synthetic data (15 +10 points)

In this section, you will attempt to design a network that is able to classify synthetically created data that is still separable, that is, where no overlap exists between the classes in the native feature space. In this case, however, the tolerances (i.e., the smallest separation between samples in different classes) are much tighter than those seen in the XOR case.

(a) **File `Feedforward_Data_ellipse.csv` contains 13312 two-dimensional data points (feature values located in columns 1 and 2) and their respective binary label (labels located in column 3). Create and train a network that separates the data. Report your best loss and accuracy values.** Plot the decision boundaries of your best network by plotting the network outputs in a densely sampled region around [-1.0,1.0] $\times$ [-1.0,1.0]. Report the number of hidden layers, type of activation function and number of neurons per layer used. *(15 points). [A portion of the total points is allocated based on your rank amongst your peers in achieving the best loss and accuracy values]*

Network:

```
class Net(nn.Module):
  def __init__(self):
    super(Net, self).__init__()
    self.fc1 = nn.Linear(2, 256)
    self.fc2 = nn.Linear(256, 256)
    #self.fc3 = nn.Linear(256, 256)
    #self.fc4 = nn.Linear(128, 128)
    #self.fc5 = nn.Linear(128, 128)
    #self.fc6 = nn.Linear(128, 128)
    self.fc7 = nn.Linear(256, 2)
  def forward(self, x):
    x = F.relu(self.fc1(x))
    x = F.relu(self.fc2(x))
    #x = F.relu(self.fc3(x))
    #x = F.relu(self.fc4(x))
    #x = F.relu(self.fc5(x))
    #x = F.relu(self.fc6(x))
```

x = self.fc7(x)
return F.log_softmax(x)
#return F.softmax(x)

Built a 2 Hidden Layer network with 256 neurons in it.
Activation function: Relu
Used Adam optimizer
#self fc3,fc4,fc5 and fc6 are in comment and not used in the model

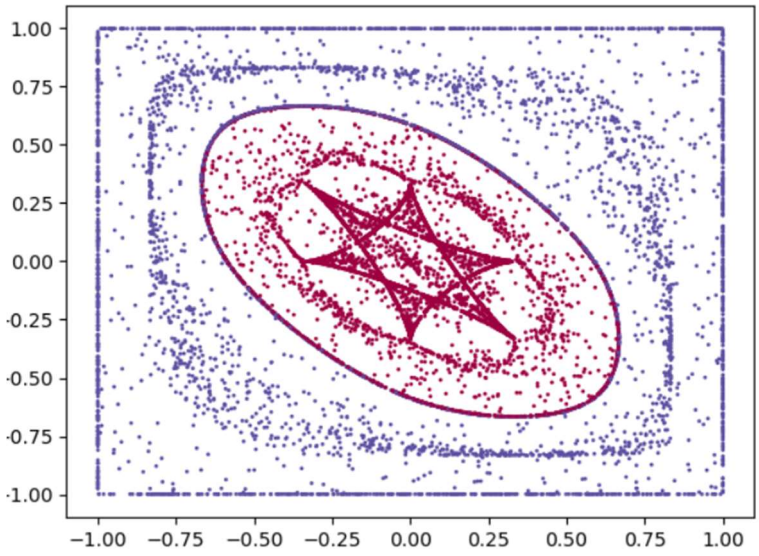

Fig 7: Scatter Plot

Results:

Best Loss and Accuracy

```
Epoch  5240 Loss  0.17049048840999603
Training accuracy is  0.9900082638419352
Epoch  5250 Loss  0.17164991796016693
Training accuracy is  0.9897077604988356

Epoch  5260 Loss  0.17355935275554657
Training accuracy is  0.9866276012320637
Epoch  5270 Loss  0.1743961125612259
Training accuracy is  0.9810682893847193
Epoch  5280 Loss  0.1730756014585495
Training accuracy is  0.9791150176545714
```

Epoch 5240 Loss 0.17049048840999603
Training accuracy is 0.9900082638419352

Training accuracy is 99.000% with Loss of 0.17049

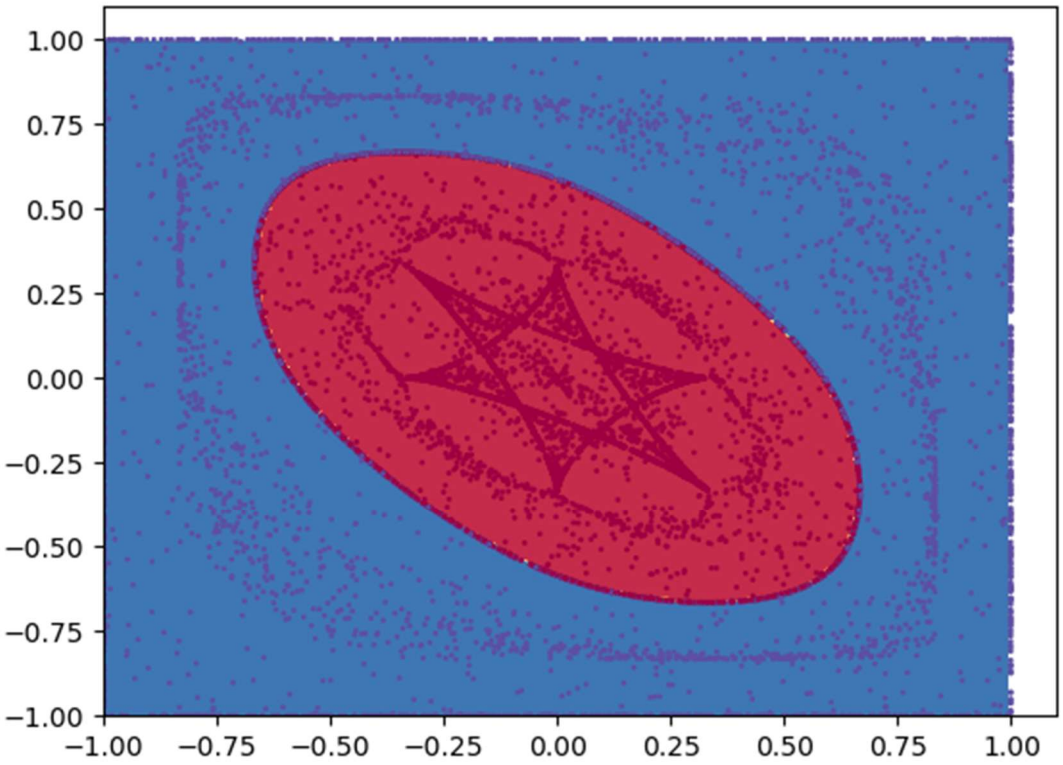

Fig 8: Decision Boundaries

(b) File `Feedforward_Data_hexa.csv` contains 13312 two-dimensional data points (feature values located in columns 1 and 2) and their respective binary label (labels located in column 3). Create and train a network that separates the data. Report your best loss and accuracy values. Plot the decision boundaries of your best network by plotting the network outputs in a densely sampled region around [-1.0,1.0] × [-1.0,1.0]. Report the number of hidden layers, type of activation function and number of neurons per layer used. *(10 points) [A portion of the total points is allocated based on your rank amongst your peers in achieving the best loss and accuracy values]*

Network:

```
class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(2, 32)
        self.fc2 = nn.Linear(32, 32)
        #self.fc3 = nn.Linear(256, 256)
        #self.fc4 = nn.Linear(128, 128)
        #self.fc5 = nn.Linear(128, 128)
        #self.fc6 = nn.Linear(128, 128)
        self.fc7 = nn.Linear(32, 2)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        #x = F.relu(self.fc3(x))
        #x = F.relu(self.fc4(x))
        #x = F.relu(self.fc5(x))
        #x = F.relu(self.fc6(x))
        x = self.fc7(x)
        return F.log_softmax(x)
        #return F.softmax(x)
```

Built a 2 Hidden Layer network with 32 neurons in it.
Activation function: Relu
Used Adam optimizer
#self fc3,fc4,fc5 and fc6 are in comment and not used in the model

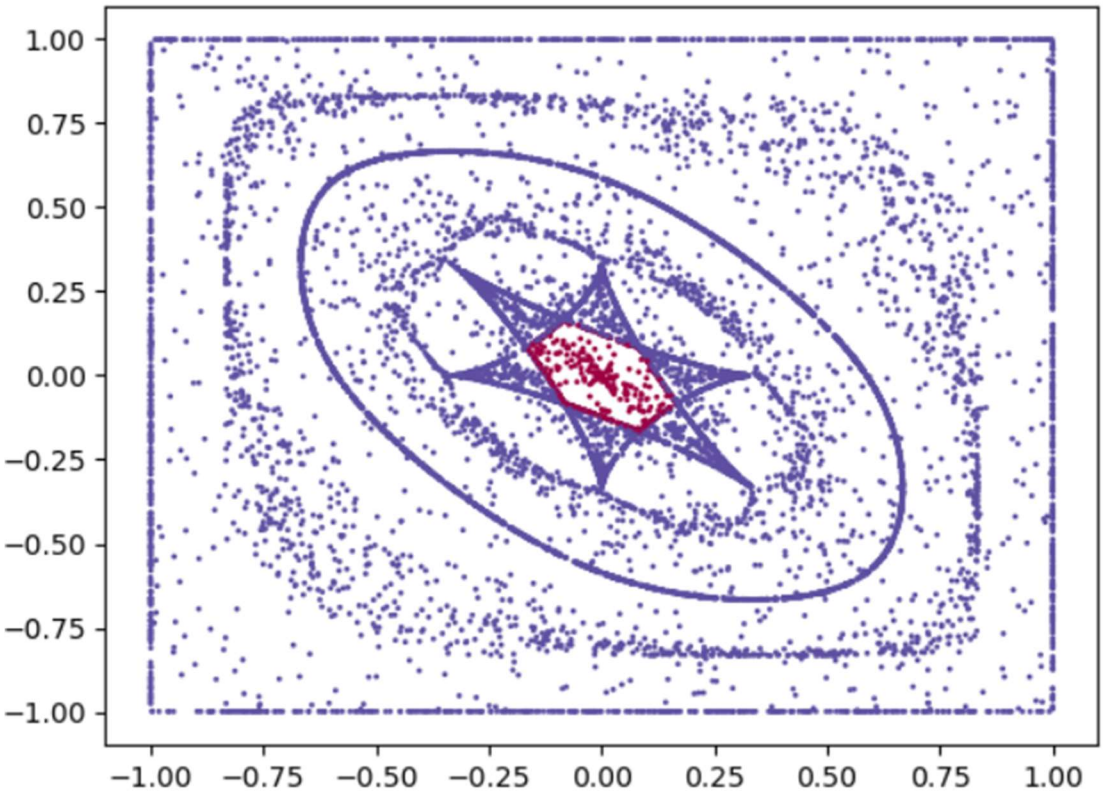

Fig 9: Scatter Plot

Results:

```
Epoch  9350 Loss  0.022177480161190033
Training accuracy is  0.9916610322289835
Epoch  9360 Loss  0.022875739261507988
Training accuracy is  0.9902336413492601
Epoch  9370 Loss  0.02817964181303978
Training accuracy is  0.9869281045751634
Epoch  9380 Loss  0.025631479918956757
Training accuracy is  0.9906843963639096
Epoch  9390 Loss  0.02212025597691536
Training accuracy is  0.9910600255427842
Epoch  9400 Loss  0.02463643252849579
```

```
Epoch  9350 Loss  0.022177480161190033 <br>
Training accuracy is  0.9916610322289835
```

Training accuracy of 99.166% with loss of 0.02217

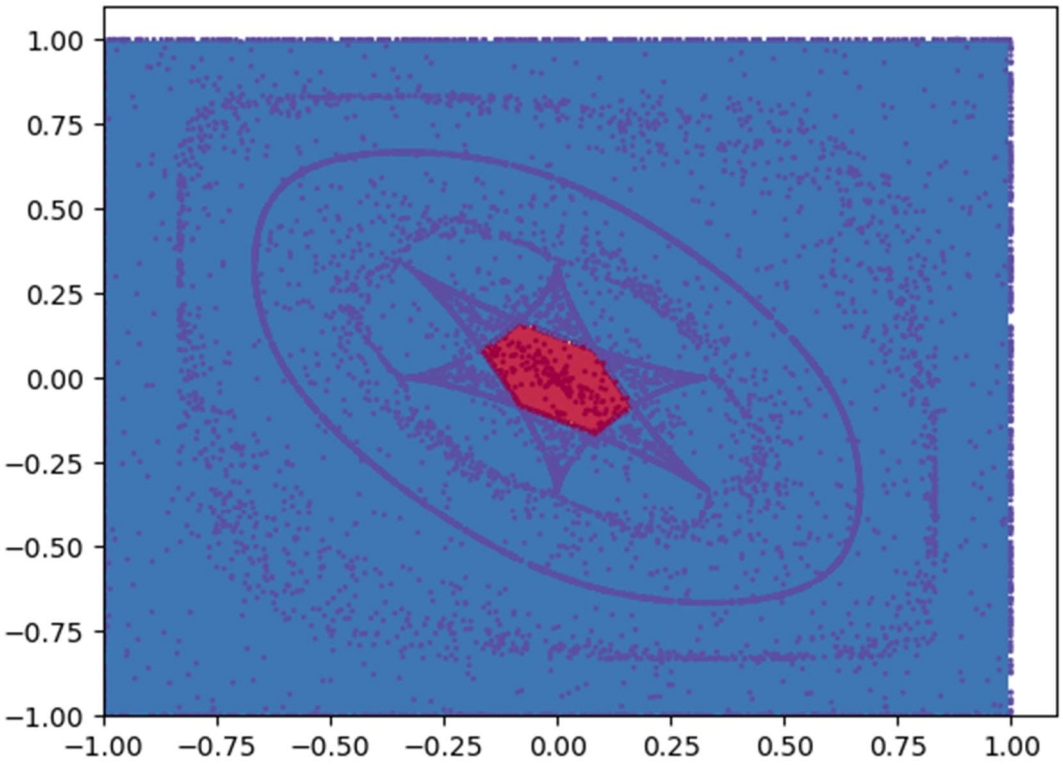

Fig 10: Decision Boundaries