PROJECT REVIEW

TOPIC:

- *Solving the* ***TWO SPIRAL PROBLEM***

  using Stochastic Gradient Descent....

COURSE CODE: EEE1020

COURSE NAME: ENGINEERING OPTIMIZATION

Project members and Project's guide

- **Project Guide :**
- Mr. Raghunathan T

- **Team Members:**
- 1. Swarup Tripathy , 19BEE0167
- 2. Adwyetya Tripathy , 19BEI0060
- 3. Aman Srivastava , 19BEE0146

ABSTRACT

- Stochastic gradient descent (often abbreviated SGD) is an iterative method for optimizing an objective function with suitable smoothness properties .
- The succeeding paper presents the process of training a neural network using Stochastic Gradient Descent to solve the two-spiral problem.

Contents

- Introduction
- Project Background
- Details of the project
- References

Introduction

It can be regarded as a stochastic approximation of gradient descent optimization, since it replaces the actual gradient by an estimate

There are a few downsides of the gradient descent algorithm. For instance when it comes to complex problems especially like a logistic regression problem that uses 23000 genes

In high-dimensional optimization problems this reduces the computational burden, achieving faster iterations in trade for a lower convergence rate

In stochastic gradient descent, the true gradient of is approximated by a gradient at a single example.

The algorithm then sweeps through the training set and performs the above update for each training example.

Several passes can be made over the training set until the algorithm converges.

PROJECT

BACKGROUND:

- The two-spiral problem is a classification task that consists of deciding in which of two interlocking spiral-shaped regions a given coordinate lies.
- The interlocking spiral shapes are chosen for this problem because they are not linearly separable.
- Thus the two-spiral task became a well-known benchmark for binary classification and since it had visual appeal, it was convenient to use in pilot studies

Let's Walk through the Code

```
import conx as cx
import math

def spiral_xy(i, spiral_num):
    """
    Create the data for a spiral.

    Arguments:
        i runs from 0 to 96
        spiral_num is 1 or -1
    """
    ϕ = i/16 * math.pi
    r = 6.5 * ((104 - i)/104)
    x = (r * math.cos(ϕ) * spiral_num)/13 + 0.5
    y = (r * math.sin(ϕ) * spiral_num)/13 + 0.5
    return (x, y)

def spiral(spiral_num):
    return [spiral_xy(i, spiral_num) for i in range(97)]
```

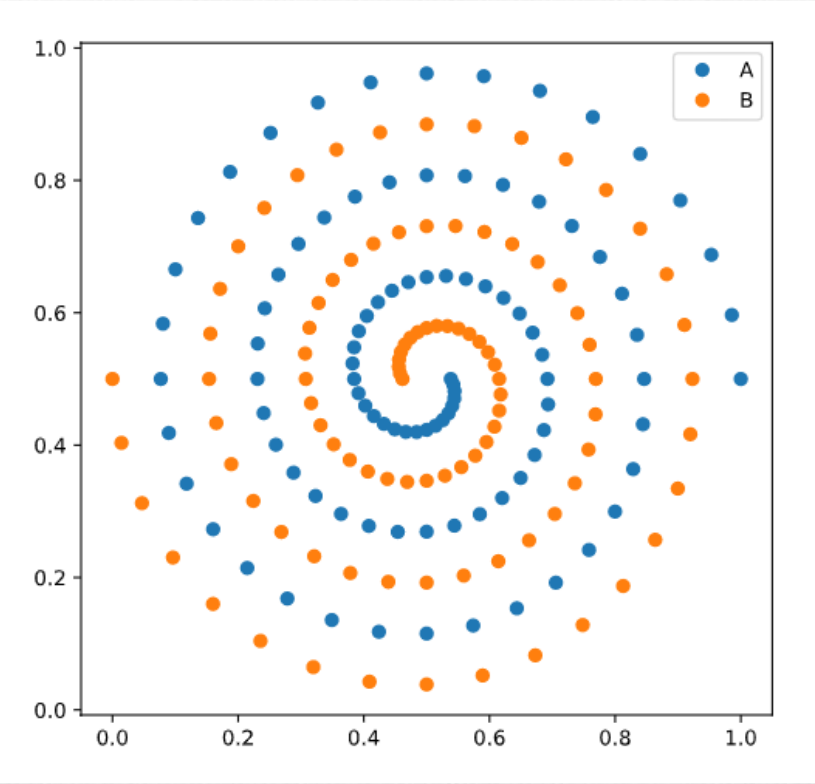
Making the Dataset first.

Here we are importing the libraries for the model where the most important is the Conx library.

And then we are defining a function where we determine the structure of the 2 spirals.

```
a = ["A", spiral(1)]
b = ["B", spiral(-1)]

cx.scatter([a,b])
```



Here we are finally creating our 2 spirals consisting of 194 data points which are distributed equally and are rotated by π radians relative to each other.

```
net = cx.Network("Two-Spirals")
net.add(
    cx.Layer("input", 2),
    cx.Layer("hidden1", 5, activation="sigmoid"),
    cx.Layer("hidden2", 5, activation="sigmoid"),
    cx.Layer("hidden3", 5, activation="sigmoid"),
    cx.Layer("output", 2, activation="softmax")
)
net.connect("input", "hidden1")
net.connect("input", "hidden2")
net.connect("input", "hidden3")
net.connect("input", "output")
net.connect("hidden1", "hidden2")
net.connect("hidden1", "hidden3")
net.connect("hidden1", "output")
net.connect("hidden2", "hidden3")
net.connect("hidden2", "output")
net.connect("hidden3", "output")
net.build_model()
```

Construction and use of Conx Network

1. Imported Conx
2. Creating the Network
3. Adding the Desired Layers
4. Connecting the Layers
5. Compiling the network with a loss function
6. Creating the Dataset
7. Training the Network
8. Testing the Network

```
net.summary()
```

```
Model: "model_12"
_________________________________________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
=================================================================================================
input (InputLayer)           (None, 2)                 0
_________________________________________________________________________________________________
hidden1 (Dense)              (None, 5)                 15          input[0][0]
_________________________________________________________________________________________________
concatenate_1 (Concatenate)  (None, 7)                 0           input[0][0]
                                                                   hidden1[0][0]
_________________________________________________________________________________________________
hidden2 (Dense)              (None, 5)                 40          concatenate_1[0][0]
_________________________________________________________________________________________________
concatenate_2 (Concatenate)  (None, 12)                0           input[0][0]
                                                                   hidden1[0][0]
                                                                   hidden2[0][0]
_________________________________________________________________________________________________
hidden3 (Dense)              (None, 5)                 65          concatenate_2[0][0]
_________________________________________________________________________________________________
concatenate_3 (Concatenate)  (None, 17)                0           input[0][0]
                                                                   hidden1[0][0]
                                                                   hidden2[0][0]
                                                                   hidden3[0][0]
_________________________________________________________________________________________________
output (Dense)               (None, 2)                 36          concatenate_3[0][0]
=================================================================================================
Total params: 156
Trainable params: 156
Non-trainable params: 0
```

```
net.dataset.load([(xy, [1, 0]) for xy in spiral(1)] +
                 [(xy, [0, 1]) for xy in spiral(-1)])
```

```
def schedule(start, end, num_steps):
    step = (end - start) / (num_steps - 1)
    current = start
    values = []
    for i in range(num_steps):
        values.append(current)
        current += step
    return values
```

So we start loading the dataset by defining a variable "xy" which loops through each and every datapoint of the 2 spirals.

Then we define a function called schedule to form the 2 matrices while determining the 'start' ← Starting Point, 'end' ← Ending Point and 'num_steps' ← Number of Steps.

```
schedule(0.001, 0.002, 10)
```

```
[0.001,
 0.0011111111111111111,
 0.0012222222222222222,
 0.0013333333333333333,
 0.0014444444444444444,
 0.0015555555555555555,
 0.0016666666666666666,
 0.0017777777777777776,
 0.0018888888888888887,
 0.002]
```

```
schedule(0.5, 0.95, 10)
```

```
[0.5,
 0.55,
 0.6000000000000001,
 0.6500000000000001,
 0.7000000000000002,
 0.7500000000000002,
 0.8000000000000003,
 0.8500000000000003,
 0.9000000000000004,
 0.9500000000000004]
```

The 2 Matrices formed

Initialised with start , end and number of steps

```
for lr, m in zip(schedule(0.001, 0.002, 10),
                 schedule(0.5, 0.95, 10)):
    net.compile(error="categorical_crossentropy", optimizer='sgd', lr=lr, momentum=m)
    net.train(100, report_rate=10, batch_size=16, accuracy=1.0, tolerance=0.4, verbose=0)
```

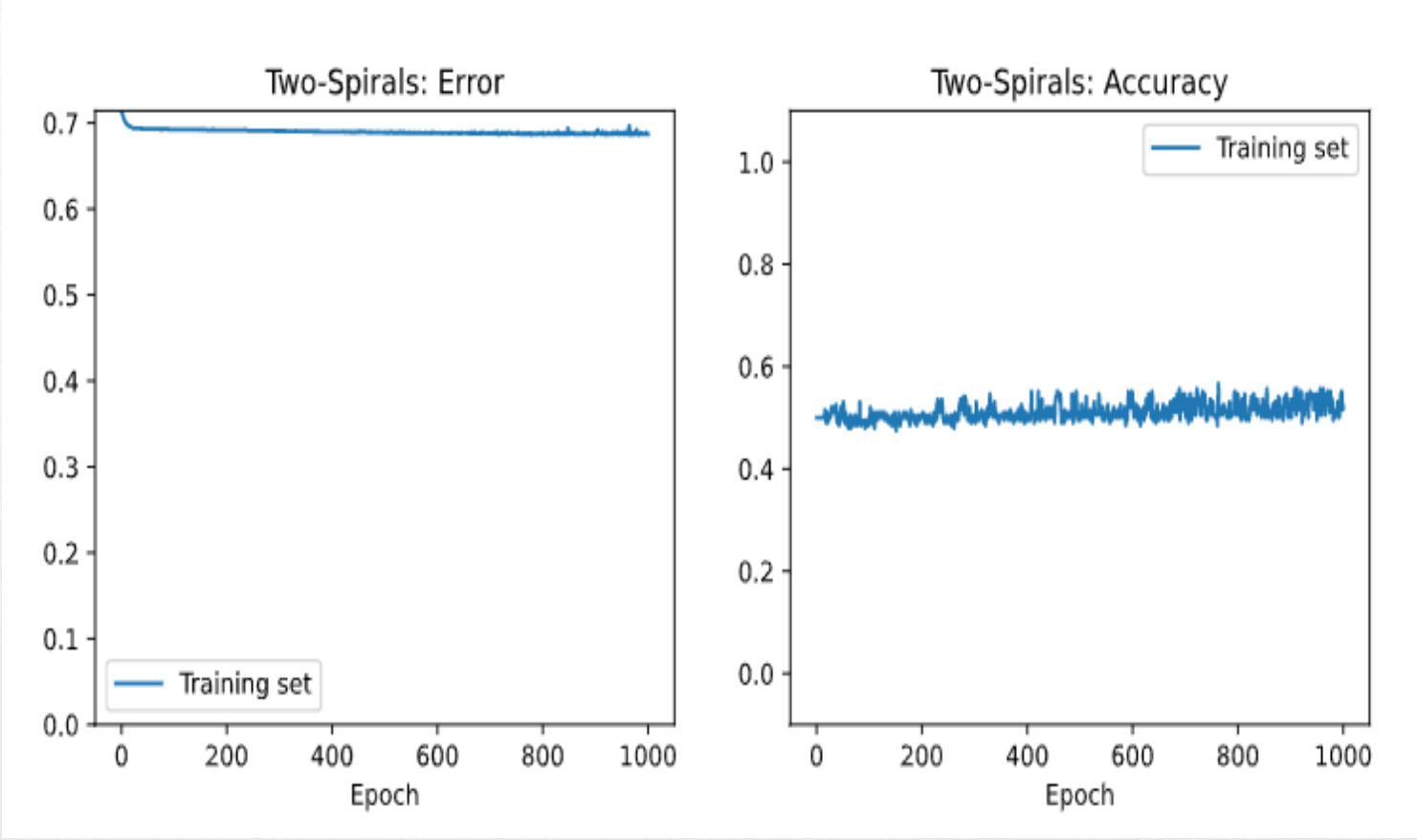
Compiling and training

Zip function → merges the same index values to the same group in python.

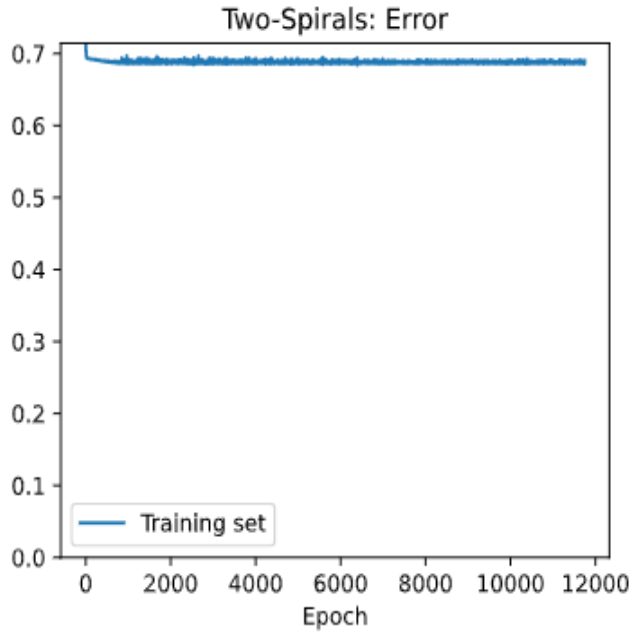
Categorical_Crossentropy → Loss function designed to quantify the difference between the 2 probability distribution.

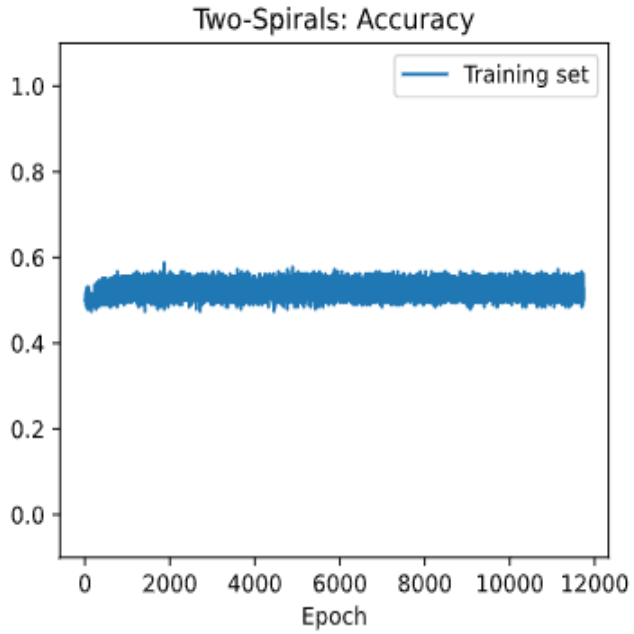
Batch_size → Number of samples to be processed before the model gets updated.

The following Output from previous Code



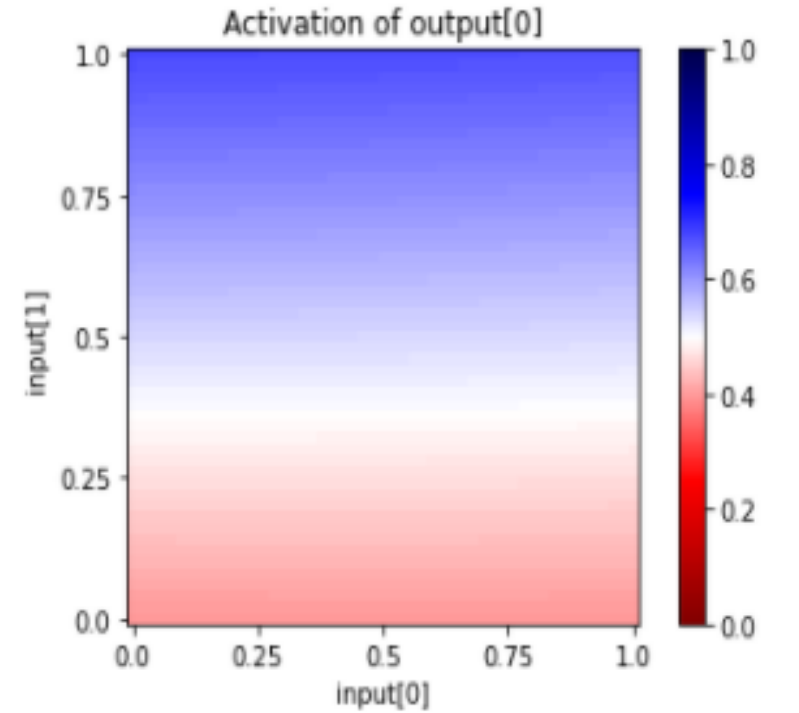
```
net.train(20000, report_rate=10, batch_size=16, accuracy=1.0, tolerance=0.4)
```






```
Interrupted! Cleaning up...
============================================================
       |  Training |  Training
Epochs |     Error |  Accuracy
------ | --------- | ---------
#11740 |   0.69030 |   0.52577
```

```
net.plot_activation_map()
```



Activation maps are just a visual representation of these activation numbers at various layers of the network as a given image progresses through as a result of various linear algebraic operations.

```
import conx as cx
import copy
```

```
RESOLUTION = 50
```

```
def make_picture(res):
    matrix = [[0.0 for i in range(res)]
              for j in range(res)]
    for x,y in spiral(1):
        x = min(int(round(x * res)), res - 1)
        y = min(int(round(y * res)), res - 1)
        matrix[1 - y][x] = 0.5
    for x,y in spiral(-1):
        x = min(int(round(x * res)), res - 1)
        y = min(int(round(y * res)), res - 1)
        matrix[1 - y][x] = 0.5
    return matrix
```

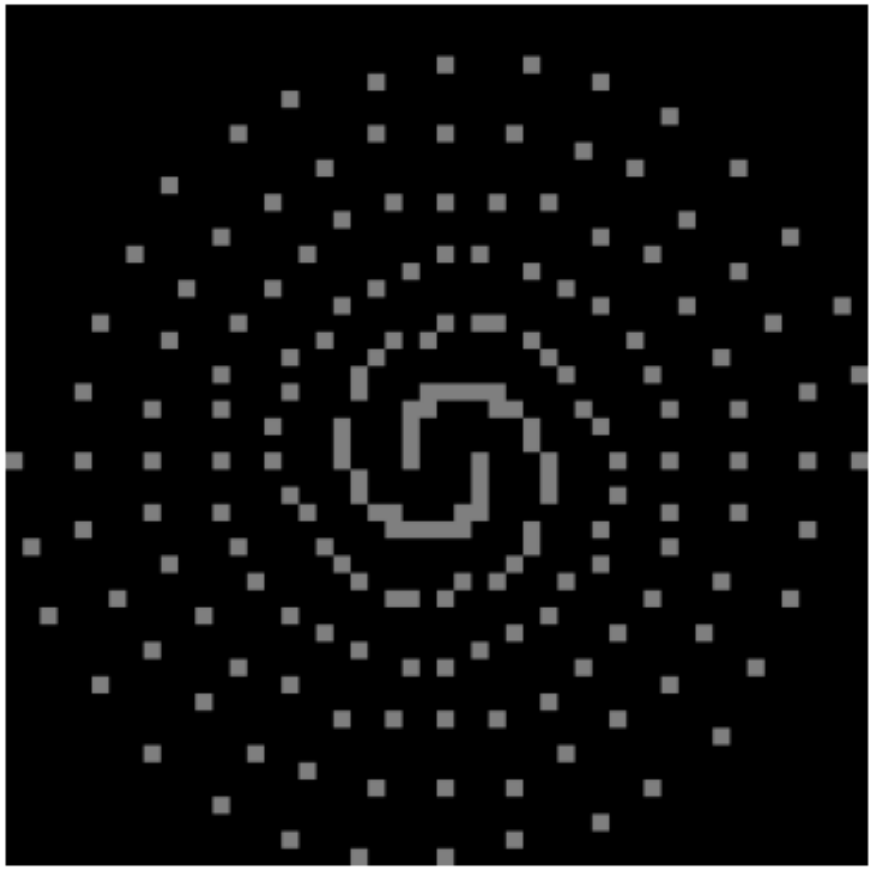
```
matrix = make_picture(RESOLUTION)
```

```
cx.array_to_image(matrix, shape=(RESOLUTION,RESOLUTION,1)).resize((400,400))
```

Picture Based Approach

In this formulation, we create "images" for each input and use a Convolutional layer.

```
cx.array_to_image(matrix, shape=(RESOLUTION,RESOLUTION,1)).resize((400,400))
```



```
def make_data(res):
    data = []
    for x,y in spiral(1):
        x = min(int(round(x * res)), res - 1)
        y = min(int(round(y * res)), res - 1)
        inputs = copy.deepcopy(matrix)
        inputs[1 - y][x] = 1.0
        inputs = cx.reshape(inputs,(50,50,1))
        data.append([inputs, [0, 1]])
    for x,y in spiral(-1):
        x = min(int(round(x * res)), res - 1)
        y = min(int(round(y * res)), res - 1)
        inputs = copy.deepcopy(matrix)
        inputs[1 - y][x] = 1.0
        inputs = cx.reshape(inputs,(50,50,1))
        data.append([inputs, [1, 0]])
    return data
```

```
data = make_data(RESOLUTION)
```

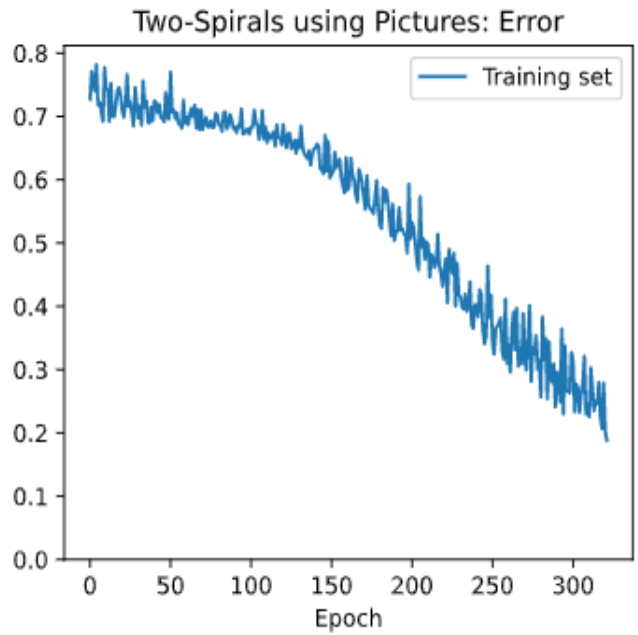
```
net = cx.Network("Two-Spirals using Pictures")
net.add(
    cx.ImageLayer("input", (RESOLUTION, RESOLUTION), 1),
    cx.Conv2DLayer("conv2d", 2, 4),
    cx.FlattenLayer("flatten"),
    cx.Layer("output", 2, activation="softmax")
)
net.connect()
net.compile(error="categorical_crossentropy", optimizer="rmsprop")
```

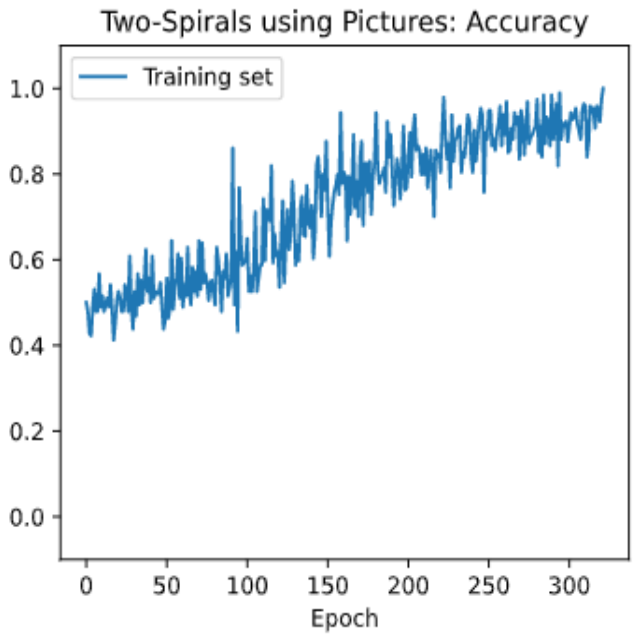
```
net.dataset.load(data)
```

Creating the simplest form of a Conv2DLayer network

Then again we proceed with the basic Conx network construction with connecting and compiling the network.

```
net.train(1000, accuracy=1.0, report_rate=10)
```





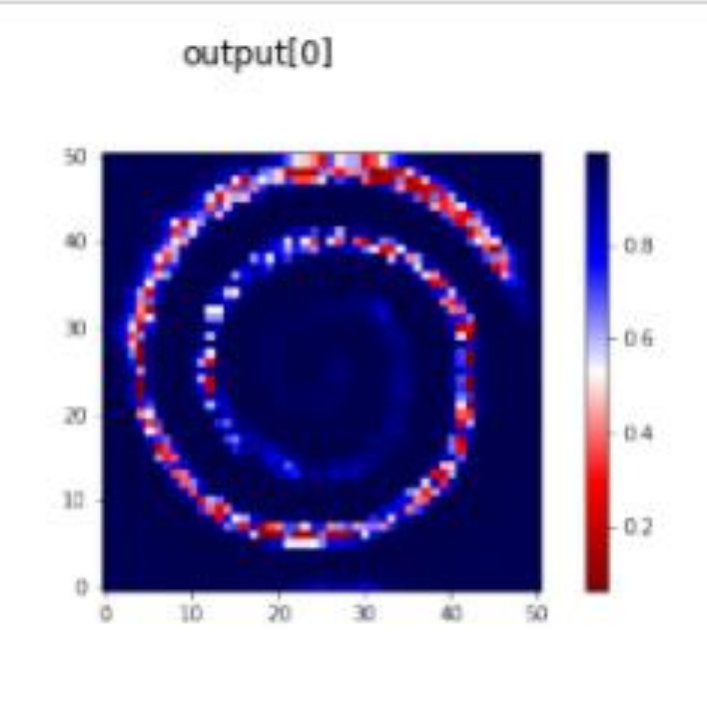

```
=======================================================
        |  Training |  Training
Epochs  |     Error |  Accuracy
------  | --------- | ---------
#  321  |   0.18812 |   1.00000
```

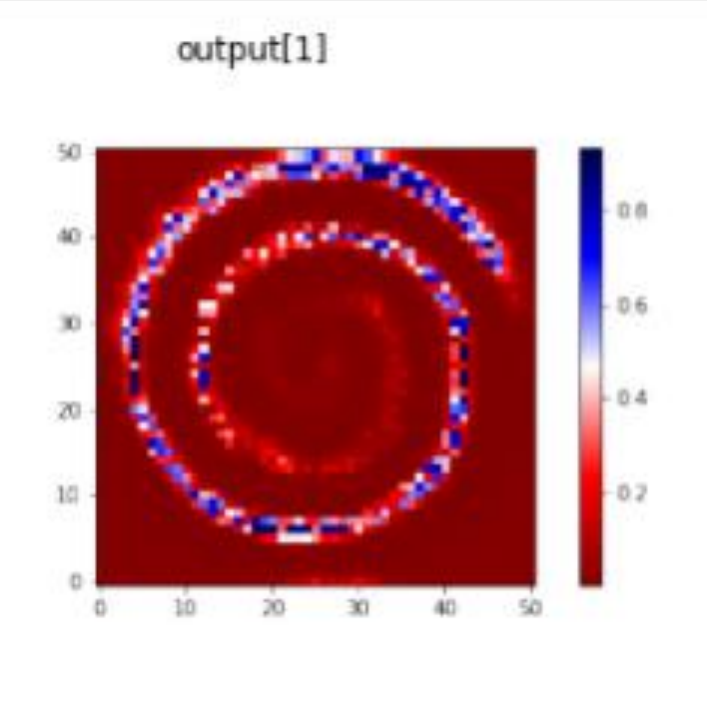
```
def test0(x, y, res=RESOLUTION):
    x = min(int(round(x * res)), res - 1)
    y = min(int(round(y * res)), res - 1)
    inputs = copy.deepcopy(matrix)
    inputs[1 - y][x] = 1.0
    inputs = cx.reshape(inputs,(50,50,1))
    return net.propagate(inputs)[0]

def test1(x, y, res=RESOLUTION):
    x = min(int(round(x * res)), res - 1)
    y = min(int(round(y * res)), res - 1)
    inputs = copy.deepcopy(matrix)
    inputs[1 - y][x] = 1.0
    inputs = cx.reshape(inputs,(50,50,1))
    return net.propagate(inputs)[1]

cx.view([cx.heatmap(test0, format="image"), cx.heatmap(test1, format="image")],
        labels=["output[0]","output[1]"], scale=7.0)
```

A heat map is a data visualization technique that shows magnitude of a phenomenon as color in two dimensions.





The Final Output

As you can see that we were able to achieve the separation of the 2 different spirals with the help of the neural network and we know that this is pixelated due to less training time in terms of epochs so if we tend to train for more time then it would result to much smoother spiral distinction.

References

•Engelbrecht, Andres (2007). *Computational Intelligence: An Introduction*, John Wiley & Sons.

•Sopena, J.M., Romero, E. and Alquezar, R. (1999). "Neural networks with periodic and monotonic activation functions: a comparative study in classification problems". (In: *ICANN Ninth International Conference on Artificial Neural Networks*.)

•Lang, K.J. and Witbrock, M.J. (1988). "Learning to Tell Two Spirals Apart".(In: *Proceedings of the 1988 Connectionist Models Summer School)*

•https://glowingpython.blogspot.com/2017/04/solving-two-spirals-problem-with-keras.html

•https://conx.readthedocs.io/en/latest/Two-Spirals.html