In [1]:
```python
# Seyun Kim
# ECE472 Deep Learning Homework 2
```

In [2]:
```python
'''
Number of samples: 1000
iterations: 5000
learning_rate = 0.4
layers = [2, 30, 40, 30, 1]

First I used tanh as my activation function but changed to a sigmoid as I thought it might be easier fo
r me to work on. Still,
I could have used tanh as activation function and pass its outcome to sigmoid to get a result that rang
es from 0 to 1.

I first tried 1000 examples, 100 iterations, layers of [2 20 30 20 1] and 0.1 learning_rate. The result
 was about 50-60%
correct data labels. For better accuracy, I tried changing the layers, examples but the loss didn't dec
rease as much.
Then, I figured that the number of iterations were too small so I increased it to 1000 and the layer to
 [2 20 30 40 30 1]. The
result was better but not good enough so I increased the number of iterations to 5000. The result is sh
own below. Among anything else,
the number of iterations affected the accuracy the most.
'''
```

Out[2]:
```
"\nNumber of samples: 1000\niterations: 5000\nlearning_rate = 0.4\nlayers = [2, 30, 40, 30, 1]\n\nFir
st I used tanh as my activation function but changed to a sigmoid as I thought it might be easier for
me to work on. Still,\nI could have used tanh as activation function and pass its outcome to sigmoid
to get a result that ranges from 0 to 1. \n\nI first tried 1000 examples, 100 iterations, layers of
[2 20 30 20 1] and 0.1 learning_rate. The result was about 50-60% \ncorrect data labels. For better a
ccuracy, I tried changing the layers, examples but the loss didn't decrease as much. \nThen, I figure
d that the number of iterations were too small so I increased it to 1000 and the layer to [2 20 30 40
30 1]. The \nresult was better but not good enough so I increased the number of iterations to 5000. T
he result is shown below. Among anything else,\nthe number of iterations affected the accuracy the mo
st.\n"
```

In [3]:
```python
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from matplotlib import cm
```

In [4]:
```python
# Parameters
N = 500
iterations = 5000
learning_rate = 0.4
layers = [2, 30, 40, 30, 1]
```

In [5]:
```python
# Data generation
def twospirals(n_points):
    n = np.sqrt(np.random.rand(n_points,1)) * 600 * (2*np.pi)/360
    d1x = -np.cos(n)*n + np.random.rand(n_points,1) * np.random.rand()
    d1y = np.sin(n)*n + np.random.rand(n_points,1) * np.random.rand()
    return (np.vstack((np.hstack((d1x,d1y)),np.hstack((-d1x,-d1y)))),
            np.hstack((np.zeros(n_points),np.ones(n_points))))
```

In [6]:
```python
# Data generated
# x: [2*N, 2]
# y: [2*N, 1]
x, y = twospirals(N)
```

In [7]:
```python
y_target = y.reshape([-1,1])
```

In [8]:
```python
# Model parameter
w = {}
b = {}
l2w = {}
l2b = {}
l2 = {}
l2norm = 0
#Initializing weights and biases according to the layers and computing their L2 norm
for i in range(0, len(layers)-1):
    w[i] = tf.Variable(tf.random.normal([layers[i], layers[i+1]], 0, 1, tf.float32))
    l2w[i] = tf.reduce_sum(tf.square(w[i]))
    b[i] = tf.Variable(tf.zeros([layers[i+1], 1]))
    l2b[i] = tf.reduce_sum(tf.square(b[i]))
#Merging L2 norm of weights and biases
for j in range(0, len(layers)-1):
    l2norm = l2norm + l2w[j] + l2b[j]
    j=j+2
```

In [9]:
```python
# Multi-Layer Perceptron model(sigmoid function used)
def f(inputs, w, b):
    y_hat = tf.sigmoid(tf.add(tf.matmul(tf.cast(inputs, tf.float32), w[0]), tf.transpose(b[0])))
    for i in range(1, len(layers)-1):
        outputs = tf.add(tf.matmul(tf.cast(y_hat, tf.float32), w[i]), tf.transpose(b[i])) # z
        y_hat = tf.sigmoid(tf.cast(outputs, tf.float32)) # f
    return y_hat
    # f: probability of example belonging to spiral 1
    # (1000,1)
```

In [10]:
```python
# BCE loss for one example
def loss(y_i, y_h):
    return -tf.multiply(tf.cast(y_i, tf.float32), tf.math.log(y_h))-tf.multiply((1-tf.cast(y_i, tf.floa
t32)), tf.math.log(1-y_h))
```

In [11]:
```python
# BCE average loss and L2 penalty for all examples
def cost(y_target, y_hat, l2norm):
    return tf.reduce_mean(loss(tf.cast(y_target, tf.float32), tf.cast(y_hat, tf.float32)), 0)+l2norm
```

In [12]:
```python
# Computes derivatives of cost w.r.t to the model parameters and updates them
# Returns updated weights and biases
def gparam(x, w, b, y_target, layers, l2norm, learning_rate):
    dC_dw = {}
    dC_db = {}
    with tf.GradientTape(persistent = True) as g:
        g.watch(w)
        g.watch(b)
        y_hat = f(x,w,b)
        c = cost(y_target, y_hat, l2norm)
    dC_dw = g.gradient(c, w)
    dC_db = g.gradient(c, b)
    W = {}
    B = {}
    W[0] = tf.math.subtract(w[0], learning_rate * dC_dw[0])
    B[0] = tf.math.subtract(b[0], learning_rate * dC_db[0])

    for i in range(1, len(layers)-1):
        W[i] = tf.math.subtract(w[i], learning_rate * dC_dw[i])
        B[i] = tf.math.subtract(b[i], learning_rate * dC_db[i])

    '''
    original function:
    for i in range(0, len(layers)-1):
        w[i] = tf.math.subtract(w[i], learning_rate*dC_dw[i])
    error -> function traced cannot alter the structre of input argument
    '''

    return W, B
```

In [13]:
```python
@tf.function
def forward(x, w, b, iterations, y_target, layers):
    for i in range(0, iterations):
        w, b = gparam(x, w, b, y_target, layers, l2norm, learning_rate)
    W = w
    B = b
    p_1 = f(x,w,b)
    return p_1, W, B
```

In [14]:
```python
p_1, W, B = forward(x, w, b, iterations, y_target, layers)
```

In [15]:
```python
# Data for generating contour
# Returns true if the possibility of a point belonging to spiral 1 is greater than 0.5 and false otherw
ise
@tf.function
def boundary(new_x, w, b):
    return f(new_x, w, b) > 0.5
```

In [16]:
```python
# Contour setup
n_plot = 600
xgrid = np.linspace(-3.2*np.pi, 3.2*np.pi, n_plot, dtype = np.float32)
ygrid = xgrid
xplot, yplot = np.meshgrid(xgrid,ygrid)
xx = np.reshape(xplot, (-1,1))
yy = np.reshape(yplot, (-1,1))
cont = np.concatenate((xx,yy),1)
```

In [17]:
```python
# Generate contour z-values
contz = boundary(cont, W, B)
```

In [18]:
```python
#Plot spirals 1 and 2 and contour map
#If p_1 is greater than 0.5, mark it spiral 1

plt.figure()
plt.contourf(xgrid, ygrid, np.reshape(contz,(n_plot,n_plot)))
for i in range(0, 2*N):
    if p_1[i] > 0.5:
        plt.plot(x[i,0], x[i,1], '.r')
    else:
        plt.plot(x[i,0], x[i,1], '.b')
plt.title("Binary Classification of Spirals")
plt.xlabel("x")
plt.ylabel("y")
plt.legend(("Spiral 1", "Spiral 2"))
plt.show()
```