Technological Institute of the Philippines	Quezon City - Computer Engineering
Course Code:	CPE 019
Code Title:	Emerging Technologies in CpE 2
2nd Semester	AY 2024 - 2025

ACTIVITY NO. 6.1	**Neural Networks**
Name	Calvadores, Kelly Joseph
Section	CPE32S3
Date Performed:	March 30, 2024
Date Submitted:	April 2, 2024
Instructor:	Engr. Roman M. Richard

Sigmoid function

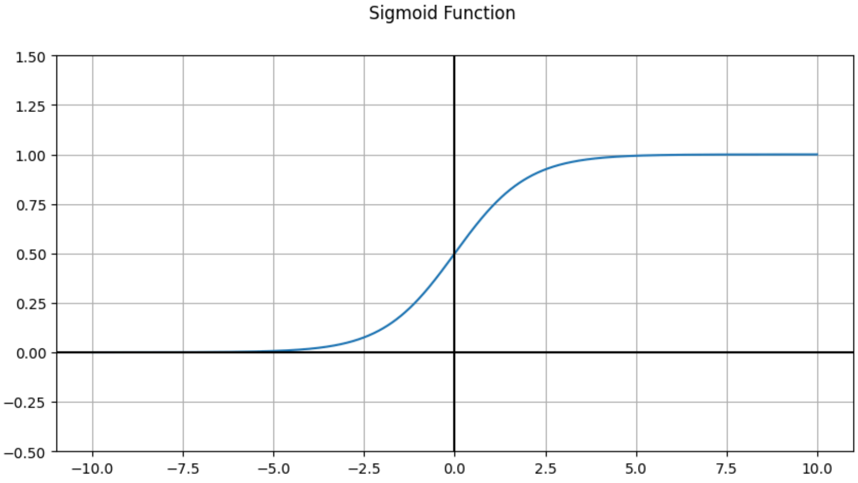
```
#Import Libraries
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

```
# Create sigmoid function
def SigFunc(x):

  return 1.0 / (1.0 + np.exp(-x))

#Create an array that will be used as dataset in this sigomoid function
Value = np.linspace(-10, 10, num=1000, dtype = np.float32)
Activate = SigFunc(Value)
print(Activate)
```

```
[4.53978682e-05 4.63158722e-05 4.72523934e-05 4.82079013e-05
 4.91826686e-05 5.01772083e-05 5.11918042e-05 5.22269656e-05
 5.32829981e-05 5.43604474e-05 5.54596190e-05 5.65810697e-05
 5.77251449e-05 5.88924158e-05 6.00832209e-05 6.12981676e-05
 6.25376124e-05 6.38021738e-05 6.50922593e-05 6.64084801e-05
 6.77512508e-05 6.91212408e-05 7.05188577e-05 7.19448071e-05
 7.33995184e-05 7.48837119e-05 7.63978387e-05 7.79426482e-05
 7.95186352e-05 8.11265418e-05 8.27668846e-05 8.44404785e-05
 8.61478256e-05 8.78897699e-05 8.96668498e-05 9.14799457e-05
 9.33296178e-05 9.52167829e-05 9.71419940e-05 9.91062261e-05
 1.01110170e-04 1.03154533e-04 1.05240331e-04 1.07368192e-04
 1.09539185e-04 1.11753950e-04 1.14013608e-04 1.16318843e-04
 1.18670796e-04 1.21070174e-04 1.23518184e-04 1.26015555e-04
 1.28563552e-04 1.31162931e-04 1.33814974e-04 1.36520524e-04
 1.39280892e-04 1.42096920e-04 1.44970021e-04 1.47901068e-04
 1.50891501e-04 1.53942252e-04 1.57054819e-04 1.60230178e-04
 1.63469842e-04 1.66774858e-04 1.70146857e-04 1.73586814e-04
 1.77096532e-04 1.80677001e-04 1.84330012e-04 1.88056685e-04
 1.91858882e-04 1.95737768e-04 1.99695234e-04 2.03732503e-04
 2.07851597e-04 2.12053696e-04 2.16340995e-04 2.20714704e-04
 2.25177035e-04 2.29729587e-04 2.34373903e-04 2.39112327e-04
 2.43946313e-04 2.48878234e-04 2.53909617e-04 2.59042892e-04
 2.64279690e-04 2.69622571e-04 2.75073166e-04 2.80634180e-04
 2.86307361e-04 2.92095443e-04 2.98000232e-04 3.04024608e-04
 3.10170493e-04 3.16440797e-04 3.22837586e-04 3.29363946e-04
 3.36022087e-04 3.42814659e-04 3.49744514e-04 3.56814475e-04
 3.64027248e-04 3.71385802e-04 3.78892990e-04 3.86551925e-04
 3.94365605e-04 4.02337173e-04 4.10469773e-04 4.18766722e-04
 4.27231338e-04 4.35866910e-04 4.44676960e-04 4.53665038e-04
 4.62834752e-04 4.72189626e-04 4.81733528e-04 4.91470215e-04
 5.01403643e-04 5.11537713e-04 5.21876500e-04 5.32424136e-04
 5.43184869e-04 5.54162834e-04 5.65362745e-04 5.76788676e-04
 5.88445575e-04 6.00337808e-04 6.12470321e-04 6.24847715e-04
 6.37475227e-04 6.50357688e-04 6.63500279e-04 6.76908356e-04
 6.90587156e-04 7.04542210e-04 7.18779105e-04 7.33303314e-04
 7.48121354e-04 7.63238175e-04 7.78660178e-04 7.94393476e-04
 8.10444530e-04 8.26819451e-04 8.43525166e-04 8.60568136e-04
 8.77955055e-04 8.95692792e-04 9.13788739e-04 9.32249939e-04
 9.51083843e-04 9.70297784e-04 9.89899389e-04 1.00989663e-03
 1.03029748e-03 1.05110998e-03 1.07234251e-03 1.09400356e-03
 1.11610151e-03 1.13864522e-03 1.16164389e-03 1.18510658e-03
 1.20904250e-03 1.23346120e-03 1.25837279e-03 1.28378649e-03
 1.30971300e-03 1.33616233e-03 1.36314507e-03 1.39067171e-03
 1.41875364e-03 1.44740171e-03 1.47662766e-03 1.50644255e-03
 1.53685862e-03 1.56788796e-03 1.59954268e-03 1.63183548e-03
 1.66477996e-03 1.69838755e-03 1.73267222e-03 1.76764815e-03
 1.80332863e-03 1.83972809e-03 1.87686074e-03 1.91474147e-03
 1.95338530e-03 1.99280749e-03 2.03302363e-03 2.07404979e-03
 2.11590179e-03 2.15859665e-03 2.20215111e-03 2.24658265e-03
 2.29190825e-03 2.33814633e-03 2.38531479e-03 2.43343250e-03
```

```
 2.48251902e-03 2.53259251e-03 2.58367369e-03 2.63578258e-03
 2.68893922e-03 2.74316501e-03 2.79848161e-03 2.85490998e-03
 2.91247317e-03 2.97119352e-03 3.03109409e-03 3.09219887e-03
 3.15453135e-03 3.21811601e-03 3.28297820e-03 3.34914355e-03
 3.41663742e-03 3.48548731e-03 3.55571904e-03 3.62736126e-03
 3.70044308e-03 3.77499033e-03 3.85103328e-03 3.92860221e-03
 4.00772644e-03 4.08843858e-03 4.17076936e-03 4.25475091e-03
 4.34041582e-03 4.42779809e-03 4.51693125e-03 4.60785022e-03
```

```
# Plot the sigmoid function

Fig = plt.figure(figsize = (10, 5))
Fig.suptitle('Sigmoid Function')
plt.plot(Value, Activate)
plt.grid(True, which = 'both')#//
plt.axhline(y = 0, color = 'k')
plt.axvline(x = 0, color = 'k')
plt.yticks()#//
plt.ylim([-0.5, 1.5]);
```



Choose any activation function and create a method to define that function.

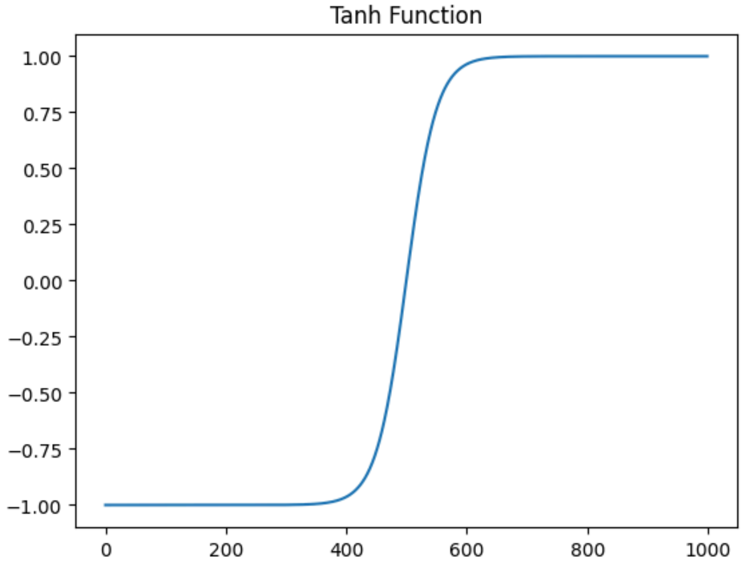
```
def TanhFunc(x):

  return np.tanh(x)

Activation = TanhFunc(Value)
print(Activation)
```

```
[-1.         -0.99999994 -0.99999994 -0.99999994 -0.99999994 -0.99999994
 -0.99999994 -0.99999994 -0.99999994 -0.99999994 -0.99999994 -0.99999994
 -0.99999994 -0.99999994 -0.99999994 -0.99999994 -0.99999994 -0.99999994
 -0.99999994 -0.99999994 -0.99999994 -0.99999994 -0.99999994 -0.99999994
 -0.99999994 -0.99999994 -0.99999994 -0.99999994 -0.99999994 -0.99999994
 -0.99999994 -0.99999994 -0.99999994 -0.99999994 -0.99999994 -0.99999994
 -0.99999994 -0.99999994 -0.99999994 -0.99999994 -0.99999994 -0.99999994
 -0.99999994 -0.99999994 -0.99999994 -0.99999994 -0.99999994 -0.99999994
 -0.99999994 -0.99999994 -0.99999994 -0.99999994 -0.99999994 -0.99999994
 -0.99999994 -0.99999994 -0.99999994 -0.99999994 -0.99999994 -0.99999994
 -0.99999994 -0.99999994 -0.99999994 -0.99999994 -0.99999994 -0.99999994
 -0.99999994 -0.9999999  -0.9999999  -0.9999999  -0.9999999  -0.9999999
 -0.9999999  -0.9999999  -0.9999999  -0.9999999  -0.9999999  -0.9999999
 -0.9999999  -0.9999999  -0.9999999  -0.9999999  -0.9999999  -0.9999999
 -0.9999999  -0.9999998  -0.9999998  -0.9999998  -0.9999998  -0.9999998
 -0.9999998  -0.9999998  -0.9999998  -0.9999998  -0.9999998  -0.99999976
 -0.99999976 -0.99999976 -0.99999976 -0.99999976 -0.99999976 -0.99999976
 -0.99999976 -0.99999976 -0.99999976 -0.99999976 -0.9999997  -0.9999997
 -0.9999997  -0.9999997  -0.9999997  -0.99999964 -0.99999964 -0.99999964
 -0.99999964 -0.9999996  -0.9999996  -0.9999996  -0.9999995  -0.9999995
 -0.9999995  -0.99999946 -0.99999946 -0.99999946 -0.9999994  -0.9999994
 -0.9999994  -0.99999934 -0.99999934 -0.9999993  -0.9999993  -0.9999992
 -0.9999992  -0.99999917 -0.9999991  -0.9999991  -0.99999905 -0.99999905
 -0.999999   -0.9999989  -0.99999887 -0.99999887 -0.9999988  -0.99999875
 -0.9999987  -0.9999986  -0.99999857 -0.9999985  -0.99999845 -0.9999984
 -0.99999833 -0.9999983  -0.99999815 -0.9999981  -0.99999803 -0.999998
 -0.99999785 -0.9999978  -0.9999977  -0.9999976  -0.9999975  -0.9999974
 -0.99999726 -0.9999972  -0.9999971  -0.99999696 -0.99999684 -0.99999666
 -0.99999654 -0.9999964  -0.99999624 -0.9999961  -0.99999595 -0.99999577
```

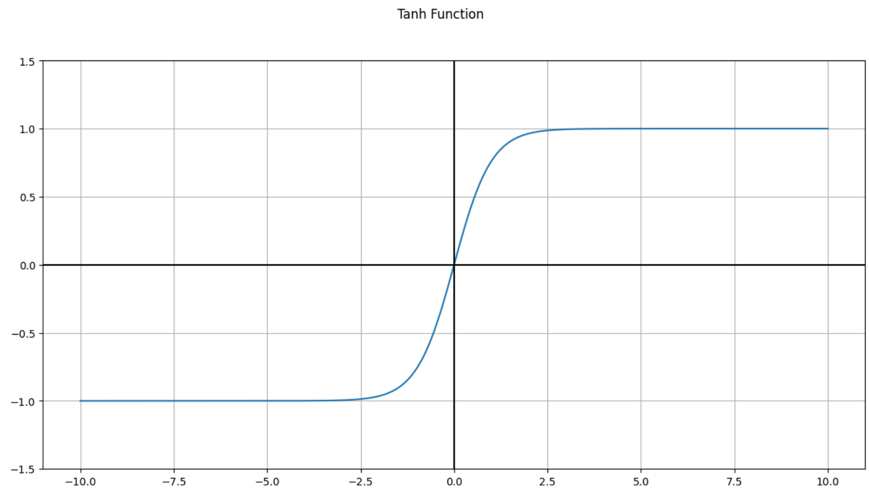
```
 -0.9999956  -0.9999954  -0.99999523 -0.99999505 -0.9999949  -0.99999464
 -0.99999446 -0.9999942  -0.999994   -0.99999374 -0.99999344 -0.9999932
 -0.9999929  -0.9999926  -0.9999923  -0.999992   -0.9999917  -0.99999136
 -0.999991   -0.99999064 -0.9999902  -0.99998987 -0.99998945 -0.99998903
 -0.99998856 -0.9999881  -0.99998766 -0.9999871  -0.9999866  -0.99998605
 -0.99998546 -0.9999849  -0.99998426 -0.9999836  -0.99998295 -0.99998224
 -0.9999815  -0.9999808  -0.99998    -0.9999792  -0.9999783  -0.99997747
 -0.9999765  -0.99997556 -0.99997455 -0.99997354 -0.99997246 -0.99997133
 -0.99997014 -0.9999689  -0.99996763 -0.9999663  -0.99996495 -0.9999635
 -0.99996203 -0.9999605  -0.9999589  -0.99995714 -0.9999554  -0.99995357
 -0.9999517  -0.9999497  -0.99994767 -0.9999455  -0.9999433  -0.999941
 -0.99993855 -0.99993604 -0.9999334  -0.99993074 -0.9999279  -0.99992496
 -0.99992186 -0.9999187  -0.99991536 -0.9999119  -0.9999083  -0.9999046
 -0.9999007  -0.9998966  -0.9998924  -0.999888   -0.9998834  -0.99987864
 -0.9998737  -0.9998685  -0.99986315 -0.99985754 -0.99985176 -0.9998457
 -0.99983937 -0.9998328  -0.999826   -0.99981886 -0.9998115  -0.9998038
 -0.9997958  -0.99978745 -0.99977875 -0.9997697  -0.99976027 -0.9997505
 -0.9997403  -0.9997297  -0.99971867 -0.99970716 -0.9996952  -0.9996828
 -0.9996698  -0.9996563  -0.99964225 -0.99962765 -0.99961245 -0.9995966
 -0.99958014 -0.999563   -0.99954516 -0.99952656 -0.99950725 -0.9994871
 -0.9994661  -0.99944437 -0.99942166 -0.99939805 -0.99937344 -0.99934787
 -0.9993212  -0.9992935  -0.99926466 -0.9992346  -0.9992033  -0.99917084
 -0.999137   -0.9991017  -0.99906504 -0.99902683 -0.99898714 -0.9989458
 -0.99890274 -0.9988579  -0.9988113  -0.9987627  -0.99871224 -0.9986597
 -0.99860495 -0.998548   -0.9984887  -0.99842703 -0.99836284 -0.998296
 -0.99822646 -0.99815404 -0.99807876 -0.9980003  -0.9979187  -0.9978338
 -0.9977454  -0.99765337 -0.99755764 -0.997458   -0.99735427 -0.9972463
 -0.99713403 -0.9970171  -0.99689543 -0.9967688  -0.99663705 -0.9964999
 -0.99635714 -0.9962086  -0.99605405 -0.9958932  -0.99572575 -0.9955515
```

Plot the activation function

```
plt.plot(Activation)
plt.title("Tanh Function")
plt.show()
```




```
Fig = plt.figure(figsize = (14, 7))
Fig.suptitle('Tanh Function')
plt.plot(Value, Activation)
plt.grid(True, which = 'both')#//
plt.axhline(y = 0, color = 'k')
plt.axvline(x = 0, color = 'k')
plt.yticks()#//
plt.ylim([-1.5, 1.5]);
```



Neurons as boolean logic gates

```
def logic_gate(w1, w2, b):

    return lambda x1, x2: SigFunc(w1 * x1 + w2 * x2 + b)

def test(gate):
    # Helper function to test out our weight functions.
    for a, b in (0, 0), (0, 1), (1, 0), (1, 1):
        print("{}, {}: {}".format(a, b, np.round(gate(a, b))))


or_gate = logic_gate(20, 20, -10)
test(or_gate)
```

```
0, 0: 0.0
0, 1: 1.0
1, 0: 1.0
1, 1: 1.0
```

Try to figure out what values for the neurons would make this function as an AND gate.

```
# Fill in the w1, w2, and b parameters such that the truth table matches
w1 = 20
w2 = 20
b = -30
and_gate = logic_gate(w1, w2, b)

test(and_gate)
```

```
0, 0: 0.0
0, 1: 0.0
1, 0: 0.0
1, 1: 1.0
```

Do the same for the NOR gate and the NAND gate.

NOR gate

```
w1_1 = -20
w2_1 = -20
b_1 = 10
nor_gate = logic_gate(w1_1, w2_1, b_1)

test(nor_gate)
```

```
0, 0: 1.0
0, 1: 0.0
1, 0: 0.0
1, 1: 0.0
```

NAND gate

```
w1_2 = -20
w2_2 = -20
b_2 = 25
nand_gate = logic_gate(w1_2, w2_2, b_2)

test(nand_gate)
```

```
0, 0: 1.0
0, 1: 1.0
1, 0: 1.0
1, 1: 0.0
```

Limitation of single neuron

```
# Make sure you have or_gate, nand_gate, and and_gate working from above!
def xor_gate(a, b):
    c = or_gate(a, b)
    d = nand_gate(a, b)
    return and_gate(c, d)
test(xor_gate)
```

```
0, 0: 0.0
0, 1: 1.0
1, 0: 1.0
1, 1: 0.0
```

Feedforward Networks

```
W_1 = np.array([[2,-1,1,4],[-1,2,-3,1],[3,-2,-1,5]])
W_2 = np.array([[3,1,-2,1],[-2,4,1,-4],[-1,-3,2,-5],[3,1,1,1]])
W_3 = np.array([[-1,3,-2],[1,-1,-3],[3,-2,2],[1,2,1]])
x_in = np.array([.5,.8,.2])
x_mat_in = np.array([[.5,.8,.2],[.1,.9,.6],[.2,.2,.3],[.6,.1,.9],[.5,.5,.4],[.9,.1,.9],[.1,.8,.7]])

def SMV(vec):# Soft Max Vector(SMV)
    return np.exp(vec)/(np.sum(np.exp(vec)))

def SMM(mat):# Soft Max Matrix
    return np.exp(mat)/(np.sum(np.exp(mat),axis=1).reshape(-1,1))

print('the matrix W_1\n')
print(W_1)
print('-'*30)
print('vector input x_in\n')
print(x_in)
print ('-'*30)
print('matrix input x_mat_in -- starts with the vector `x_in`\n')
print(x_mat_in)
```

```
the matrix W_1

[[ 2 -1  1  4]
 [-1  2 -3  1]
 [ 3 -2 -1  5]]
------------------------------
vector input x_in

[0.5 0.8 0.2]
------------------------------
matrix input x_mat_in -- starts with the vector `x_in`

[[0.5 0.8 0.2]
 [0.1 0.9 0.6]
```

```
 [0.2 0.2 0.3]
 [0.6 0.1 0.9]
 [0.5 0.5 0.4]
 [0.9 0.1 0.9]
 [0.1 0.8 0.7]]
```

```
#1. Get the product of array x_in and W_1 (z2)
z2 = np.dot(x_in, W_1)
print(z2)
#2. Apply sigmoid function to z2 that results to a2
a2 = SigFunc(z2)
print(a2)
#3. Get the product of a2 and z2 (z3)
z3 = np.dot(a2, z2)
print(z3)
#4. Apply sigmoid function to z3 that results to a3
a3 = SigFunc(z3)
print(a3)
#5. Get the product of a3 and z3 that results to z4
z4 = np.dot(a3, z3)
print(z4)
```

```
[ 0.8  0.7 -2.1  3.8]
[0.68997448 0.66818777 0.10909682 0.97811873]
4.507458871351723
0.9890938122523221
4.458299678635824
```

Apply soft_max_vec function to z4 that results to y_out

```
def SMV(vec):# Soft Max Vector(SMV)
    return np.exp(vec)/(np.sum(np.exp(vec)))

def SMM(mat):# Soft Max Matrix
    return np.exp(mat)/(np.sum(np.exp(mat),axis=1).reshape(-1,1))
```

```
y_out1 = SMV(z4)
#y_out2 = SMM(z4)
print(y_out1)
#print(y_out2)
```

```
1.0
```

```
## A one-line function to do the entire neural net computation

def nn_comp_vec(x):
    return SMV(SigFunc(SigFunc(np.dot(x,W_1)).dot(W_2)).dot(W_3))

def nn_comp_mat(x):
    return SMM(SigFunc(SigFunc(np.dot(x,W_1)).dot(W_2)).dot(W_3))
```

```
nn_comp_vec(x_in)
```

```
array([0.72780576, 0.26927918, 0.00291506])
```

```
nn_comp_mat(x_mat_in)
```

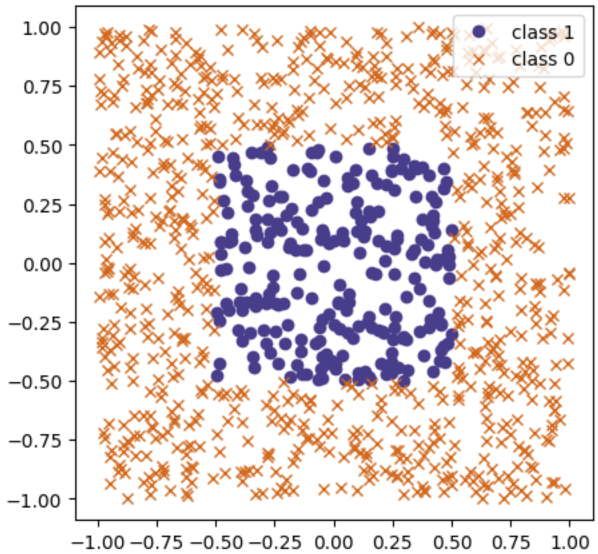
```
array([[0.72780576, 0.26927918, 0.00291506],
       [0.62054212, 0.37682531, 0.00263257],
       [0.69267581, 0.30361576, 0.00370844],
       [0.36618794, 0.63016955, 0.00364252],
       [0.57199769, 0.4251982 , 0.00280411],
       [0.38373781, 0.61163804, 0.00462415],
       [0.52510443, 0.4725011 , 0.00239447]])
```

Backpropagation

```
#Preliminaries
from __future__ import division, print_function
```

```
## This code below generates two x values and a y value according to different patterns
## It also creates a "bias" term (a vector of 1s)
## The goal is then to learn the mapping from x to y using a neural network via back-propagation

num_obs = 1000
x_mat_1 = np.random.uniform(-1,1,size = (num_obs,2))
x_mat_bias = np.ones((num_obs,1))
x_mat_full = np.concatenate( (x_mat_1,x_mat_bias), axis=1)

# PICK ONE PATTERN BELOW and comment out the rest.

# # Circle pattern
# y = (np.sqrt(x_mat_full[:,0]**2 + x_mat_full[:,1]**2)<.75).astype(int)

# # Diamond Pattern
# y = ((np.abs(x_mat_full[:,0]) + np.abs(x_mat_full[:,1]))<1).astype(int)

# # Centered square
y = ((np.maximum(np.abs(x_mat_full[:,0]), np.abs(x_mat_full[:,1])))<.5).astype(int)

# # Thick Right Angle pattern
# y = (((np.maximum((x_mat_full[:,0]), (x_mat_full[:,1])))<.5) & ((np.maximum((x_mat_full[:,0]), (x_mat_full[:,1])))>-.5)).astype(int)

# # Thin right angle pattern
# y = (((np.maximum((x_mat_full[:,0]), (x_mat_full[:,1])))<.5) & ((np.maximum((x_mat_full[:,0]), (x_mat_full[:,1])))>0)).astype(int)


print('shape of x_mat_full is {}'.format(x_mat_full.shape))
print('shape of y is {}'.format(y.shape))

fig, ax = plt.subplots(figsize=(5, 5))
ax.plot(x_mat_full[y==1, 0],x_mat_full[y==1, 1], 'ro', label='class 1', color='darkslateblue')
ax.plot(x_mat_full[y==0, 0],x_mat_full[y==0, 1], 'bx', label='class 0', color='chocolate')
# ax.grid(True)
ax.legend(loc='best')
ax.axis('equal');
```

```
shape of x_mat_full is (1000, 3)
shape of y is (1000,)
<ipython-input-30-5c19db4c51d6>:32: UserWarning: color is redundantly defined by the 'color' keyword argument and the fmt string "rc
  ax.plot(x_mat_full[y==1, 0],x_mat_full[y==1, 1], 'ro', label='class 1', color='darkslateblue')
<ipython-input-30-5c19db4c51d6>:33: UserWarning: color is redundantly defined by the 'color' keyword argument and the fmt string "b>
  ax.plot(x_mat_full[y==0, 0],x_mat_full[y==0, 1], 'bx', label='class 0', color='chocolate')
```



```
def loss_fn(y_true, y_pred, eps=1e-16):
    """
    Loss function we would like to optimize (minimize)
    We are using Logarithmic Loss
    http://scikit-learn.org/stable/modules/model_evaluation.html#log-loss
    """
    y_pred = np.maximum(y_pred,eps)
    y_pred = np.minimum(y_pred,(1-eps))
    return -(np.sum(y_true * np.log(y_pred)) + np.sum((1-y_true)*np.log(1-y_pred)))/len(y_true)


def forward_pass(W1, W2):
    global x_mat
    global y
    global num_
    # First, compute the new predictions `y_pred`
    z_2 = np.dot(x_mat, W_1)
    a_2 = SigFunc(z_2)
    z_3 = np.dot(a_2, W_2)
    y_pred = SigFunc(z_3).reshape((len(x_mat),))
    # Now compute the gradient
    J_z_3_grad = -y + y_pred
    J_W_2_grad = np.dot(J_z_3_grad, a_2)
    a_2_z_2_grad = SigFunc(z_2)*(1-SigFunc(z_2))
    J_W_1_grad = (np.dot((J_z_3_grad).reshape(-1,1), W_2.reshape(-1,1).T)*a_2_z_2_grad).T.dot(x_mat).T
    gradient = (J_W_1_grad, J_W_2_grad)

    # return
    return y_pred, gradient


def plot_loss_accuracy(loss_vals, accuracies):
    fig = plt.figure(figsize=(16, 8))
    fig.suptitle('Log Loss and Accuracy over iterations')

    ax = fig.add_subplot(1, 2, 1)
    ax.plot(loss_vals)
    ax.grid(True)
    ax.set(xlabel='iterations', title='Log Loss')

    ax = fig.add_subplot(1, 2, 2)
    ax.plot(accuracies)
    ax.grid(True)
    ax.set(xlabel='iterations', title='Accuracy');
```

Complete the pseudocode below

```
#### Initialize the network parameters

np.random.seed(1241)

W_1 = np.random.uniform(-1,1,size = (3,4))
W_2 = np.random.uniform(-1,1,size = (4))
num_iter = 1500
learning_rate = 0.001
x_mat = x_mat_full


loss_vals, accuracies = [], []
for i in range(num_iter):
    ### Do a forward computation, and get the gradient
    y_pred, (w_1Grad, w_2Grad) = forward_pass(W_1, W_2)

    ## Update the weight matrices
    W_1 = W_1 - learning_rate * w_1Grad
    W_2 = W_2 - learning_rate * w_2Grad

    ### Compute the loss and accuracy
    Loss = loss_fn(y, y_pred)
    loss_vals.append(Loss)

    Accuracy = np.sum((y_pred >= 0.5 ) == y) / num_obs
    accuracies.append(Accuracy)

    ## Print the loss and accuracy for every 200th iteration
    if i % 200:
      print(f"Iteration {i}: Loss {Loss:.4f}, Accuracy {Accuracy:.4f}")

plot_loss_accuracy(loss_vals, accuracies)
```

```
Iteration 1: Loss 0.5597, Accuracy 0.7540
Iteration 2: Loss 0.5593, Accuracy 0.7540
Iteration 3: Loss 0.5590, Accuracy 0.7540
Iteration 4: Loss 0.5589, Accuracy 0.7540
Iteration 5: Loss 0.5588, Accuracy 0.7540
Iteration 6: Loss 0.5587, Accuracy 0.7540
Iteration 7: Loss 0.5586, Accuracy 0.7540
Iteration 8: Loss 0.5585, Accuracy 0.7540
Iteration 9: Loss 0.5585, Accuracy 0.7540
Iteration 10: Loss 0.5584, Accuracy 0.7540
Iteration 11: Loss 0.5584, Accuracy 0.7540
Iteration 12: Loss 0.5584, Accuracy 0.7540
Iteration 13: Loss 0.5583, Accuracy 0.7540
Iteration 14: Loss 0.5583, Accuracy 0.7540
Iteration 15: Loss 0.5582, Accuracy 0.7540
Iteration 16: Loss 0.5582, Accuracy 0.7540
Iteration 17: Loss 0.5581, Accuracy 0.7540
Iteration 18: Loss 0.5581, Accuracy 0.7540
Iteration 19: Loss 0.5581, Accuracy 0.7540
Iteration 20: Loss 0.5580, Accuracy 0.7540
Iteration 21: Loss 0.5580, Accuracy 0.7540
Iteration 22: Loss 0.5579, Accuracy 0.7540
Iteration 23: Loss 0.5579, Accuracy 0.7540
Iteration 24: Loss 0.5579, Accuracy 0.7540
Iteration 25: Loss 0.5578, Accuracy 0.7540
Iteration 26: Loss 0.5578, Accuracy 0.7540
Iteration 27: Loss 0.5578, Accuracy 0.7540
Iteration 28: Loss 0.5577, Accuracy 0.7540
Iteration 29: Loss 0.5577, Accuracy 0.7540
Iteration 30: Loss 0.5576, Accuracy 0.7540
Iteration 31: Loss 0.5576, Accuracy 0.7540
Iteration 32: Loss 0.5576, Accuracy 0.7540
Iteration 33: Loss 0.5575, Accuracy 0.7540
Iteration 34: Loss 0.5575, Accuracy 0.7540
Iteration 35: Loss 0.5575, Accuracy 0.7540
Iteration 36: Loss 0.5574, Accuracy 0.7540
Iteration 37: Loss 0.5574, Accuracy 0.7540
Iteration 38: Loss 0.5574, Accuracy 0.7540
Iteration 39: Loss 0.5573, Accuracy 0.7540
Iteration 40: Loss 0.5573, Accuracy 0.7540
Iteration 41: Loss 0.5573, Accuracy 0.7540
Iteration 42: Loss 0.5572, Accuracy 0.7540
Iteration 43: Loss 0.5572, Accuracy 0.7540
Iteration 44: Loss 0.5572, Accuracy 0.7540
Iteration 45: Loss 0.5572, Accuracy 0.7540
Iteration 46: Loss 0.5571, Accuracy 0.7540
Iteration 47: Loss 0.5571, Accuracy 0.7540
Iteration 48: Loss 0.5571, Accuracy 0.7540
Iteration 49: Loss 0.5570, Accuracy 0.7540
Iteration 50: Loss 0.5570, Accuracy 0.7540
Iteration 51: Loss 0.5570, Accuracy 0.7540
Iteration 52: Loss 0.5569, Accuracy 0.7540
Iteration 53: Loss 0.5569, Accuracy 0.7540
Iteration 54: Loss 0.5569, Accuracy 0.7540
Iteration 55: Loss 0.5569, Accuracy 0.7540
Iteration 56: Loss 0.5568, Accuracy 0.7540
Iteration 57: Loss 0.5568, Accuracy 0.7540
Iteration 58: Loss 0.5568, Accuracy 0.7540
Iteration 59: Loss 0.5567, Accuracy 0.7540
Iteration 60: Loss 0.5567, Accuracy 0.7540
Iteration 61: Loss 0.5567, Accuracy 0.7540
Iteration 62: Loss 0.5566, Accuracy 0.7540
Iteration 63: Loss 0.5566, Accuracy 0.7540
Iteration 64: Loss 0.5566, Accuracy 0.7540
Iteration 65: Loss 0.5566, Accuracy 0.7540
Iteration 66: Loss 0.5565, Accuracy 0.7540
Iteration 67: Loss 0.5565, Accuracy 0.7540
Iteration 68: Loss 0.5565, Accuracy 0.7540
Iteration 69: Loss 0.5564, Accuracy 0.7540
Iteration 70: Loss 0.5564, Accuracy 0.7540
Iteration 71: Loss 0.5564, Accuracy 0.7540
Iteration 72: Loss 0.5564, Accuracy 0.7540
Iteration 73: Loss 0.5563, Accuracy 0.7540
Iteration 74: Loss 0.5563, Accuracy 0.7540
Iteration 75: Loss 0.5563, Accuracy 0.7540
Iteration 76: Loss 0.5562, Accuracy 0.7540
Iteration 77: Loss 0.5562, Accuracy 0.7540
Iteration 78: Loss 0.5562, Accuracy 0.7540
Iteration 79: Loss 0.5562, Accuracy 0.7540
Iteration 80: Loss 0.5561, Accuracy 0.7540
Iteration 81: Loss 0.5561, Accuracy 0.7540
Iteration 82: Loss 0.5561, Accuracy 0.7540
Iteration 83: Loss 0.5560, Accuracy 0.7540
Iteration 84: Loss 0.5560, Accuracy 0.7540
Iteration 85: Loss 0.5560, Accuracy 0.7540
Iteration 86: Loss 0.5560, Accuracy 0.7540
Iteration 87: Loss 0.5559, Accuracy 0.7540
Iteration 88: Loss 0.5559, Accuracy 0.7540
Iteration 89: Loss 0.5559, Accuracy 0.7540
Iteration 90: Loss 0.5558, Accuracy 0.7540
```

```
Iteration 91: Loss 0.5558, Accuracy 0.7540
Iteration 92: Loss 0.5558, Accuracy 0.7540
Iteration 93: Loss 0.5557, Accuracy 0.7540
Iteration 94: Loss 0.5557, Accuracy 0.7540
Iteration 95: Loss 0.5557, Accuracy 0.7540
Iteration 96: Loss 0.5557, Accuracy 0.7540
Iteration 97: Loss 0.5556, Accuracy 0.7540
Iteration 98: Loss 0.5556, Accuracy 0.7540
Iteration 99: Loss 0.5556, Accuracy 0.7540
Iteration 100: Loss 0.5555, Accuracy 0.7540
Iteration 101: Loss 0.5555, Accuracy 0.7540
Iteration 102: Loss 0.5555, Accuracy 0.7540
Iteration 103: Loss 0.5554, Accuracy 0.7540
Iteration 104: Loss 0.5554, Accuracy 0.7540
Iteration 105: Loss 0.5554, Accuracy 0.7540
Iteration 106: Loss 0.5553, Accuracy 0.7540
Iteration 107: Loss 0.5553, Accuracy 0.7540
Iteration 108: Loss 0.5553, Accuracy 0.7540
Iteration 109: Loss 0.5552, Accuracy 0.7540
Iteration 110: Loss 0.5552, Accuracy 0.7540
Iteration 111: Loss 0.5552, Accuracy 0.7540
Iteration 112: Loss 0.5551, Accuracy 0.7540
Iteration 113: Loss 0.5551, Accuracy 0.7540
Iteration 114: Loss 0.5551, Accuracy 0.7540
Iteration 115: Loss 0.5550, Accuracy 0.7540
Iteration 116: Loss 0.5550, Accuracy 0.7540
Iteration 117: Loss 0.5550, Accuracy 0.7540
Iteration 118: Loss 0.5549, Accuracy 0.7540
Iteration 119: Loss 0.5549, Accuracy 0.7540
Iteration 120: Loss 0.5549, Accuracy 0.7540
Iteration 121: Loss 0.5548, Accuracy 0.7540
Iteration 122: Loss 0.5548, Accuracy 0.7540
Iteration 123: Loss 0.5548, Accuracy 0.7540
Iteration 124: Loss 0.5547, Accuracy 0.7540
Iteration 125: Loss 0.5547, Accuracy 0.7540
Iteration 126: Loss 0.5546, Accuracy 0.7540
Iteration 127: Loss 0.5546, Accuracy 0.7540
Iteration 128: Loss 0.5546, Accuracy 0.7540
Iteration 129: Loss 0.5545, Accuracy 0.7540
Iteration 130: Loss 0.5545, Accuracy 0.7540
Iteration 131: Loss 0.5545, Accuracy 0.7540
Iteration 132: Loss 0.5544, Accuracy 0.7540
Iteration 133: Loss 0.5544, Accuracy 0.7540
Iteration 134: Loss 0.5543, Accuracy 0.7540
Iteration 135: Loss 0.5543, Accuracy 0.7540
Iteration 136: Loss 0.5543, Accuracy 0.7540
Iteration 137: Loss 0.5542, Accuracy 0.7540
Iteration 138: Loss 0.5542, Accuracy 0.7540
Iteration 139: Loss 0.5541, Accuracy 0.7540
Iteration 140: Loss 0.5541, Accuracy 0.7540
Iteration 141: Loss 0.5540, Accuracy 0.7540
Iteration 142: Loss 0.5540, Accuracy 0.7540
Iteration 143: Loss 0.5540, Accuracy 0.7540
Iteration 144: Loss 0.5539, Accuracy 0.7540
Iteration 145: Loss 0.5539, Accuracy 0.7540
Iteration 146: Loss 0.5538, Accuracy 0.7540
Iteration 147: Loss 0.5538, Accuracy 0.7540
Iteration 148: Loss 0.5537, Accuracy 0.7540
Iteration 149: Loss 0.5537, Accuracy 0.7540
Iteration 150: Loss 0.5536, Accuracy 0.7540
Iteration 151: Loss 0.5536, Accuracy 0.7540
Iteration 152: Loss 0.5535, Accuracy 0.7540
Iteration 153: Loss 0.5535, Accuracy 0.7540
Iteration 154: Loss 0.5534, Accuracy 0.7540
Iteration 155: Loss 0.5534, Accuracy 0.7540
Iteration 156: Loss 0.5533, Accuracy 0.7540
Iteration 157: Loss 0.5533, Accuracy 0.7540
Iteration 158: Loss 0.5532, Accuracy 0.7540
Iteration 159: Loss 0.5532, Accuracy 0.7540
Iteration 160: Loss 0.5531, Accuracy 0.7540
Iteration 161: Loss 0.5531, Accuracy 0.7540
Iteration 162: Loss 0.5530, Accuracy 0.7540
Iteration 163: Loss 0.5530, Accuracy 0.7540
Iteration 164: Loss 0.5529, Accuracy 0.7540
Iteration 165: Loss 0.5529, Accuracy 0.7540
Iteration 166: Loss 0.5528, Accuracy 0.7540
Iteration 167: Loss 0.5528, Accuracy 0.7540
Iteration 168: Loss 0.5527, Accuracy 0.7540
Iteration 169: Loss 0.5527, Accuracy 0.7540
Iteration 170: Loss 0.5526, Accuracy 0.7540
Iteration 171: Loss 0.5525, Accuracy 0.7540
Iteration 172: Loss 0.5525, Accuracy 0.7540
Iteration 173: Loss 0.5524, Accuracy 0.7540
Iteration 174: Loss 0.5524, Accuracy 0.7540
Iteration 175: Loss 0.5523, Accuracy 0.7540
Iteration 176: Loss 0.5522, Accuracy 0.7540
Iteration 177: Loss 0.5522, Accuracy 0.7540
Iteration 178: Loss 0.5521, Accuracy 0.7540
Iteration 179: Loss 0.5520, Accuracy 0.7540
Iteration 180: Loss 0.5520, Accuracy 0.7540
Iteration 181: Loss 0.5519, Accuracy 0.7540
```

```
Iteration 181: Loss 0.5519, Accuracy 0.7540
Iteration 182: Loss 0.5519, Accuracy 0.7540
Iteration 183: Loss 0.5518, Accuracy 0.7540
Iteration 184: Loss 0.5517, Accuracy 0.7540
Iteration 185: Loss 0.5517, Accuracy 0.7540
Iteration 186: Loss 0.5516, Accuracy 0.7540
Iteration 187: Loss 0.5515, Accuracy 0.7540
Iteration 188: Loss 0.5514, Accuracy 0.7540
Iteration 189: Loss 0.5514, Accuracy 0.7540
Iteration 190: Loss 0.5513, Accuracy 0.7540
Iteration 191: Loss 0.5512, Accuracy 0.7540
Iteration 192: Loss 0.5512, Accuracy 0.7540
Iteration 193: Loss 0.5511, Accuracy 0.7540
Iteration 194: Loss 0.5510, Accuracy 0.7540
Iteration 195: Loss 0.5509, Accuracy 0.7540
Iteration 196: Loss 0.5509, Accuracy 0.7540
Iteration 197: Loss 0.5508, Accuracy 0.7540
Iteration 198: Loss 0.5507, Accuracy 0.7540
Iteration 199: Loss 0.5506, Accuracy 0.7540
Iteration 201: Loss 0.5505, Accuracy 0.7540
Iteration 202: Loss 0.5504, Accuracy 0.7540
Iteration 203: Loss 0.5503, Accuracy 0.7540
Iteration 204: Loss 0.5502, Accuracy 0.7540
Iteration 205: Loss 0.5501, Accuracy 0.7540
Iteration 206: Loss 0.5500, Accuracy 0.7540
Iteration 207: Loss 0.5499, Accuracy 0.7540
Iteration 208: Loss 0.5499, Accuracy 0.7540
Iteration 209: Loss 0.5498, Accuracy 0.7540
Iteration 210: Loss 0.5497, Accuracy 0.7540
Iteration 211: Loss 0.5496, Accuracy 0.7540
Iteration 212: Loss 0.5495, Accuracy 0.7540
Iteration 213: Loss 0.5494, Accuracy 0.7540
Iteration 214: Loss 0.5493, Accuracy 0.7540
Iteration 215: Loss 0.5492, Accuracy 0.7540
Iteration 216: Loss 0.5491, Accuracy 0.7540
Iteration 217: Loss 0.5490, Accuracy 0.7540
Iteration 218: Loss 0.5489, Accuracy 0.7540
Iteration 219: Loss 0.5488, Accuracy 0.7540
Iteration 220: Loss 0.5487, Accuracy 0.7540
Iteration 221: Loss 0.5486, Accuracy 0.7540
Iteration 222: Loss 0.5485, Accuracy 0.7540
Iteration 223: Loss 0.5484, Accuracy 0.7540
Iteration 224: Loss 0.5483, Accuracy 0.7540
Iteration 225: Loss 0.5482, Accuracy 0.7540
Iteration 226: Loss 0.5481, Accuracy 0.7540
Iteration 227: Loss 0.5480, Accuracy 0.7540
Iteration 228: Loss 0.5478, Accuracy 0.7540
Iteration 229: Loss 0.5477, Accuracy 0.7540
Iteration 230: Loss 0.5476, Accuracy 0.7540
Iteration 231: Loss 0.5475, Accuracy 0.7540
Iteration 232: Loss 0.5474, Accuracy 0.7540
Iteration 233: Loss 0.5473, Accuracy 0.7540
Iteration 234: Loss 0.5471, Accuracy 0.7540
Iteration 235: Loss 0.5470, Accuracy 0.7540
Iteration 236: Loss 0.5469, Accuracy 0.7540
Iteration 237: Loss 0.5468, Accuracy 0.7540
Iteration 238: Loss 0.5466, Accuracy 0.7540
Iteration 239: Loss 0.5465, Accuracy 0.7540
Iteration 240: Loss 0.5464, Accuracy 0.7540
Iteration 241: Loss 0.5463, Accuracy 0.7540
Iteration 242: Loss 0.5461, Accuracy 0.7540
Iteration 243: Loss 0.5460, Accuracy 0.7540
Iteration 244: Loss 0.5459, Accuracy 0.7540
Iteration 245: Loss 0.5457, Accuracy 0.7540
Iteration 246: Loss 0.5456, Accuracy 0.7540
Iteration 247: Loss 0.5454, Accuracy 0.7540
Iteration 248: Loss 0.5453, Accuracy 0.7540
Iteration 249: Loss 0.5452, Accuracy 0.7540
Iteration 250: Loss 0.5450, Accuracy 0.7540
Iteration 251: Loss 0.5449, Accuracy 0.7540
Iteration 252: Loss 0.5447, Accuracy 0.7540
Iteration 253: Loss 0.5446, Accuracy 0.7540
Iteration 254: Loss 0.5444, Accuracy 0.7540
Iteration 255: Loss 0.5443, Accuracy 0.7540
Iteration 256: Loss 0.5441, Accuracy 0.7540
Iteration 257: Loss 0.5439, Accuracy 0.7540
Iteration 258: Loss 0.5438, Accuracy 0.7540
Iteration 259: Loss 0.5436, Accuracy 0.7540
Iteration 260: Loss 0.5435, Accuracy 0.7540
Iteration 261: Loss 0.5433, Accuracy 0.7540
Iteration 262: Loss 0.5431, Accuracy 0.7540
Iteration 263: Loss 0.5430, Accuracy 0.7540
Iteration 264: Loss 0.5428, Accuracy 0.7540
Iteration 265: Loss 0.5426, Accuracy 0.7540
Iteration 266: Loss 0.5424, Accuracy 0.7540
Iteration 267: Loss 0.5423, Accuracy 0.7540
Iteration 268: Loss 0.5421, Accuracy 0.7540
Iteration 269: Loss 0.5419, Accuracy 0.7540
Iteration 270: Loss 0.5417, Accuracy 0.7540
Iteration 271: Loss 0.5415, Accuracy 0.7540
Iteration 272: Loss 0.5414, Accuracy 0.7540
```

```
Iteration 273: Loss 0.5412, Accuracy 0.7540
Iteration 274: Loss 0.5410, Accuracy 0.7540
Iteration 275: Loss 0.5408, Accuracy 0.7540
Iteration 276: Loss 0.5406, Accuracy 0.7540
Iteration 277: Loss 0.5404, Accuracy 0.7540
Iteration 278: Loss 0.5402, Accuracy 0.7540
Iteration 279: Loss 0.5400, Accuracy 0.7540
Iteration 280: Loss 0.5398, Accuracy 0.7540
Iteration 281: Loss 0.5396, Accuracy 0.7540
Iteration 282: Loss 0.5394, Accuracy 0.7540
Iteration 283: Loss 0.5392, Accuracy 0.7540
Iteration 284: Loss 0.5389, Accuracy 0.7540
Iteration 285: Loss 0.5387, Accuracy 0.7540
Iteration 286: Loss 0.5385, Accuracy 0.7540
Iteration 287: Loss 0.5383, Accuracy 0.7540
Iteration 288: Loss 0.5381, Accuracy 0.7540
Iteration 289: Loss 0.5378, Accuracy 0.7540
Iteration 290: Loss 0.5376, Accuracy 0.7540
Iteration 291: Loss 0.5374, Accuracy 0.7540
Iteration 292: Loss 0.5371, Accuracy 0.7540
Iteration 293: Loss 0.5369, Accuracy 0.7540
Iteration 294: Loss 0.5367, Accuracy 0.7540
Iteration 295: Loss 0.5364, Accuracy 0.7540
Iteration 296: Loss 0.5362, Accuracy 0.7540
Iteration 297: Loss 0.5359, Accuracy 0.7540
Iteration 298: Loss 0.5357, Accuracy 0.7540
Iteration 299: Loss 0.5354, Accuracy 0.7540
Iteration 300: Loss 0.5352, Accuracy 0.7540
Iteration 301: Loss 0.5349, Accuracy 0.7540
Iteration 302: Loss 0.5347, Accuracy 0.7540
Iteration 303: Loss 0.5344, Accuracy 0.7540
Iteration 304: Loss 0.5342, Accuracy 0.7540
Iteration 305: Loss 0.5339, Accuracy 0.7540
Iteration 306: Loss 0.5336, Accuracy 0.7540
Iteration 307: Loss 0.5334, Accuracy 0.7540
Iteration 308: Loss 0.5331, Accuracy 0.7540
Iteration 309: Loss 0.5328, Accuracy 0.7540
Iteration 310: Loss 0.5325, Accuracy 0.7540
Iteration 311: Loss 0.5322, Accuracy 0.7540
Iteration 312: Loss 0.5320, Accuracy 0.7540
Iteration 313: Loss 0.5317, Accuracy 0.7540
Iteration 314: Loss 0.5314, Accuracy 0.7540
Iteration 315: Loss 0.5311, Accuracy 0.7540
Iteration 316: Loss 0.5308, Accuracy 0.7540
Iteration 317: Loss 0.5305, Accuracy 0.7540
Iteration 318: Loss 0.5302, Accuracy 0.7540
Iteration 319: Loss 0.5299, Accuracy 0.7540
Iteration 320: Loss 0.5296, Accuracy 0.7540
Iteration 321: Loss 0.5293, Accuracy 0.7540
Iteration 322: Loss 0.5289, Accuracy 0.7540
Iteration 323: Loss 0.5286, Accuracy 0.7540
Iteration 324: Loss 0.5283, Accuracy 0.7540
Iteration 325: Loss 0.5280, Accuracy 0.7540
Iteration 326: Loss 0.5277, Accuracy 0.7540
Iteration 327: Loss 0.5273, Accuracy 0.7540
Iteration 328: Loss 0.5270, Accuracy 0.7540
Iteration 329: Loss 0.5267, Accuracy 0.7540
Iteration 330: Loss 0.5263, Accuracy 0.7540
Iteration 331: Loss 0.5260, Accuracy 0.7540
Iteration 332: Loss 0.5257, Accuracy 0.7540
Iteration 333: Loss 0.5253, Accuracy 0.7540
Iteration 334: Loss 0.5250, Accuracy 0.7540
Iteration 335: Loss 0.5246, Accuracy 0.7540
Iteration 336: Loss 0.5242, Accuracy 0.7540
Iteration 337: Loss 0.5239, Accuracy 0.7540
Iteration 338: Loss 0.5235, Accuracy 0.7540
Iteration 339: Loss 0.5232, Accuracy 0.7540
Iteration 340: Loss 0.5228, Accuracy 0.7540
Iteration 341: Loss 0.5224, Accuracy 0.7540
Iteration 342: Loss 0.5221, Accuracy 0.7540
Iteration 343: Loss 0.5217, Accuracy 0.7540
Iteration 344: Loss 0.5213, Accuracy 0.7540
Iteration 345: Loss 0.5209, Accuracy 0.7540
Iteration 346: Loss 0.5205, Accuracy 0.7540
Iteration 347: Loss 0.5201, Accuracy 0.7540
Iteration 348: Loss 0.5198, Accuracy 0.7540
Iteration 349: Loss 0.5194, Accuracy 0.7540
Iteration 350: Loss 0.5190, Accuracy 0.7540
Iteration 351: Loss 0.5186, Accuracy 0.7540
Iteration 352: Loss 0.5182, Accuracy 0.7540
Iteration 353: Loss 0.5178, Accuracy 0.7540
Iteration 354: Loss 0.5173, Accuracy 0.7540
Iteration 355: Loss 0.5169, Accuracy 0.7540
Iteration 356: Loss 0.5165, Accuracy 0.7540
Iteration 357: Loss 0.5161, Accuracy 0.7540
Iteration 358: Loss 0.5157, Accuracy 0.7540
Iteration 359: Loss 0.5153, Accuracy 0.7540
Iteration 360: Loss 0.5148, Accuracy 0.7540
Iteration 361: Loss 0.5144, Accuracy 0.7540
Iteration 362: Loss 0.5140, Accuracy 0.7540
Iteration 363: Loss 0.5135, Accuracy 0.7540
```