COSC4337_Manual Neural Network

1 Manual Neural Network

1.0.1 Quick Note on Super() and OOP

[1]:
```
class SimpleClass():

    def __init__(self,str_input):
        print("SIMPLE"+str_input)
```

[2]:
```
class ExtendedClass(SimpleClass):

    def __init__(self):
        print('EXTENDED')
```

[3]:
```
s = ExtendedClass()
```

```
EXTENDED
```

[4]:
```
class ExtendedClass(SimpleClass):

    def __init__(self):

        super().__init__(" My String")
        print('EXTENDED')
```

[5]:
```
s = ExtendedClass()
```

```
SIMPLE My String
EXTENDED
```

1.1 Operation

[6]:
```
class Operation():
    """
```

```
    An Operation is a node in a "Graph". TensorFlow will also use this concept␣
↪of a Graph.

    This Operation class will be inherited by other classes that actually␣
↪compute the specific
    operation, such as adding or matrix multiplication.
    """

    def __init__(self, input_nodes = []):
        """
        Intialize an Operation
        """
        self.input_nodes = input_nodes # The list of input nodes
        self.output_nodes = [] # List of nodes consuming this node's output

        # For every node in the input, we append this operation (self) to the␣
↪list of
        # the consumers of the input nodes
        for node in input_nodes:
            node.output_nodes.append(self)

        # There will be a global default graph (TensorFlow works this way)
        # We will then append this particular operation
        # Append this operation to the list of operations in the currently␣
↪active default graph
        _default_graph.operations.append(self)

    def compute(self):
        """
        This is a placeholder function. It will be overwritten by the actual␣
↪specific operation
        that inherits from this class.

        """

        pass
```

1.2 Example Operations

1.2.1 Addition

[7]:
```
class add(Operation):

    def __init__(self, x, y):

        super().__init__([x, y])

```

```
    def compute(self, x_var, y_var):

        self.inputs = [x_var, y_var]
        return x_var + y_var
```

1.2.2 Multiplication

[8]:
```
class multiply(Operation):

    def __init__(self, a, b):

        super().__init__([a, b])

    def compute(self, a_var, b_var):

        self.inputs = [a_var, b_var]
        return a_var * b_var
```

1.2.3 Matrix Multiplication

[9]:
```
class matmul(Operation):

    def __init__(self, a, b):

        super().__init__([a, b])

    def compute(self, a_mat, b_mat):

        self.inputs = [a_mat, b_mat]
        return a_mat.dot(b_mat)
```

1.3 Placeholders

[10]:
```
class Placeholder():
    """
    A placeholder is a node that needs to be provided a value for computing the␣
↪output in the Graph.
    """

    def __init__(self):

        self.output_nodes = []

        _default_graph.placeholders.append(self)
```

1.4 Variables

[11]:
```
class Variable():
    """
    This variable is a changeable parameter of the Graph.
    """

    def __init__(self, initial_value = None):

        self.value = initial_value
        self.output_nodes = []


        _default_graph.variables.append(self)
```

1.5 Graph

[12]:
```
class Graph():


    def __init__(self):

        self.operations = []
        self.placeholders = []
        self.variables = []

    def set_as_default(self):
        """
        Sets this Graph instance as the Global Default Graph
        """
        global _default_graph
        _default_graph = self
```

1.6 A Basic Graph

$$z = Ax + b$$

With A=10 and b=1

$$z = 10x + 1$$

Just need a placeholder for x and then once x is filled in we can solve it!

[13]:
```
g = Graph()
```

[14]:
```
g.set_as_default()
```

[15]:
```
A = Variable(10)
```

[16]:
```
b = Variable(1)
```

[17]:
```
# Will be filled out later
x = Placeholder()
```

[18]:
```
y = multiply(A,x)
```

[19]:
```
z = add(y,b)
```

1.7 Session

[20]:
```
import numpy as np
```

1.7.1 Traversing Operation Nodes

[21]:
```
def traverse_postorder(operation):
    """
    PostOrder Traversal of Nodes. Basically makes sure computations are done in
    the correct order (Ax first , then Ax + b). Feel free to copy and paste␣
↪this code.
    It is not super important for understanding the basic fundamentals of deep␣
↪learning.
    """

    nodes_postorder = []
    def recurse(node):
        if isinstance(node, Operation):
            for input_node in node.input_nodes:
                recurse(input_node)
        nodes_postorder.append(node)

    recurse(operation)
    return nodes_postorder
```

[22]:
```
class Session:

    def run(self, operation, feed_dict = {}):
        """
          operation: The operation to compute
          feed_dict: Dictionary mapping placeholders to input values (the data)␣
↪
        """

        # Puts nodes in correct order
        nodes_postorder = traverse_postorder(operation)

        for node in nodes_postorder:
```

```
            if type(node) == Placeholder:

                node.output = feed_dict[node]

            elif type(node) == Variable:

                node.output = node.value

            else: # Operation

                node.inputs = [input_node.output for input_node in node.
↪input_nodes]


                node.output = node.compute(*node.inputs)

            # Convert lists to numpy arrays
            if type(node.output) == list:
                node.output = np.array(node.output)

        # Return the requested node value
        return operation.output
```

[23]:
```
sess = Session()
```

[24]:
```
result = sess.run(operation=z,feed_dict={x:10})
```

[25]:
```
result
```

[25]: 101

[26]:
```
10*10 + 1
```

[26]: 101

** Looks like we did it! **

[27]:
```
g = Graph()

g.set_as_default()

A = Variable([[10,20],[30,40]])
b = Variable([1,1])

x = Placeholder()

y = matmul(A,x)
```

```
z = add(y,b)
```

[28]:
```
sess = Session()
```

[29]:
```
result = sess.run(operation=z,feed_dict={x:10})
```

[30]:
```
result
```

[30]:
```
array([[101, 201],
       [301, 401]])
```

1.8 Activation Function

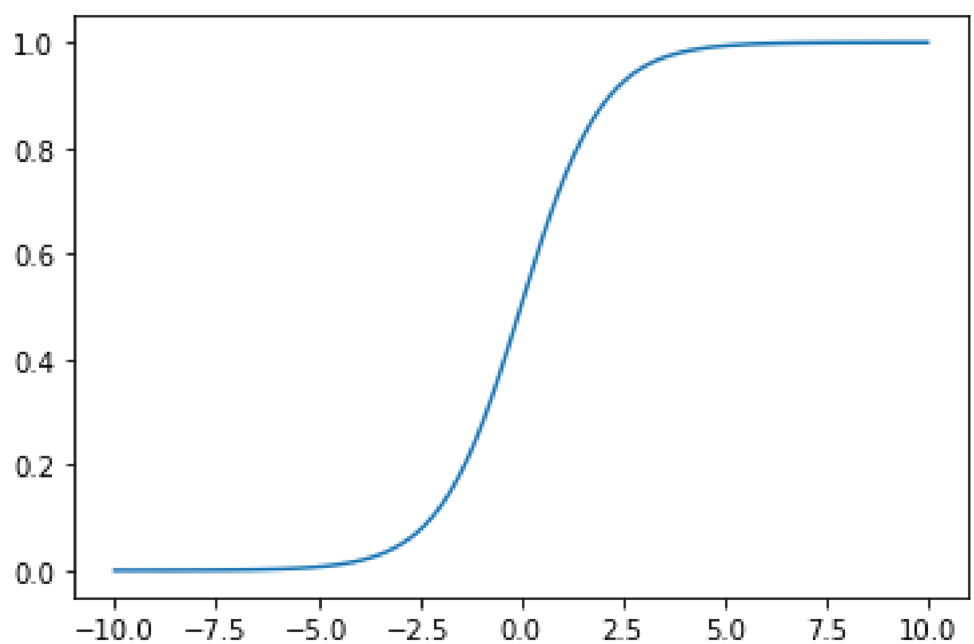
[31]:
```
import matplotlib.pyplot as plt
%matplotlib inline
```

[32]:
```
def sigmoid(z):
    return 1/(1+np.exp(-z))
```

[33]:
```
sample_z = np.linspace(-10,10,100)
sample_a = sigmoid(sample_z)
```

[34]:
```
plt.plot(sample_z,sample_a)
```

[34]:
```
[<matplotlib.lines.Line2D at 0x212d8295d08>]
```



Sigmoid as an Operation

[35]:
```
class Sigmoid(Operation):


    def __init__(self, z):

        # a is the input node
        super().__init__([z])

    def compute(self, z_val):

        return 1/(1+np.exp(-z_val))
```

1.9 Classification Example

[36]:
```
from sklearn.datasets import make_blobs
```

[37]:
```
data = make_blobs(n_samples = 50,n_features=2,centers=2,random_state=75)
```

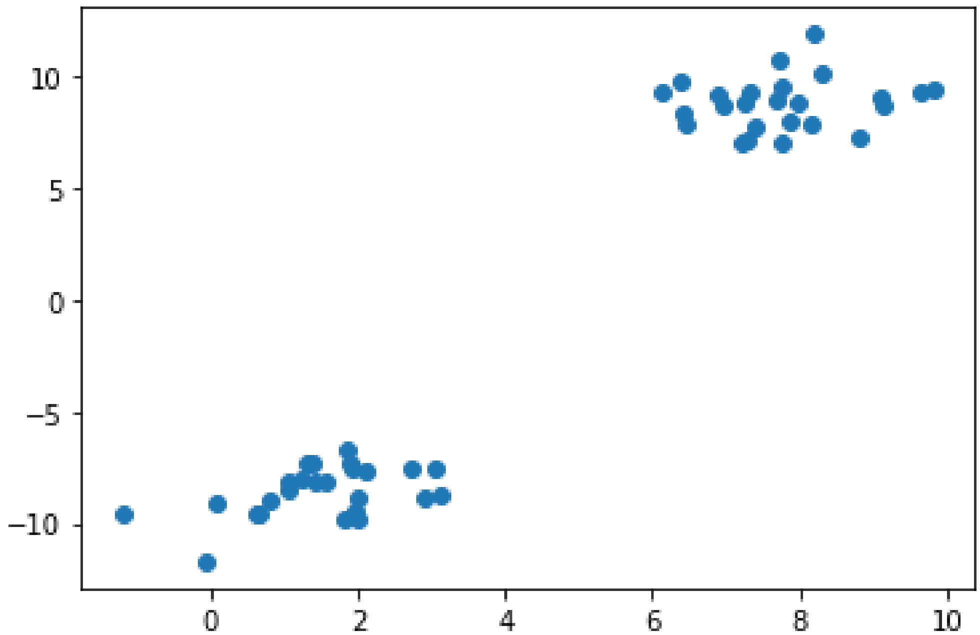
[38]:
```
data
```

[38]:
```
(array([[ 7.3402781 ,  9.36149154],
        [ 9.13332743,  8.74906102],
        [ 1.99243535, -8.85885722],
        [ 7.38443759,  7.72520389],
        [ 7.97613887,  8.80878209],
        [ 7.76974352,  9.50899462],
        [ 8.3186688 , 10.1026025 ],
        [ 8.79588546,  7.28046702],
        [ 9.81270381,  9.46968531],
        [ 1.57961049, -8.17089971],
        [ 0.06441546, -9.04982817],
        [ 7.2075117 ,  7.04533624],
        [ 9.10704928,  9.0272212 ],
        [ 1.82921897, -9.86956281],
        [ 7.85036314,  7.986659  ],
        [ 3.04605603, -7.50486114],
        [ 1.85582689, -6.74473432],
        [ 2.88603902, -8.85261704],
        [-1.20046211, -9.55928542],
        [ 2.00890845, -9.78471782],
        [ 7.68945113,  9.01706723],
        [ 6.42356167,  8.33356412],
        [ 8.15467319,  7.87489634],
        [ 1.92000795, -7.50953708],
        [ 1.90073973, -7.24386675],
        [ 7.7605855 ,  7.05124418],
```

```
       [ 6.90561582,   9.23493842],
       [ 0.65582768,  -9.5920878 ],
       [ 1.41804346,  -8.10517372],
       [ 9.65371965,   9.35409538],
       [ 1.23053506,  -7.98873571],
       [ 1.96322881,  -9.50169117],
       [ 6.11644251,   9.26709393],
       [ 7.70630321,  10.78862346],
       [ 0.79580385,  -9.00301023],
       [ 3.13114921,  -8.6849493 ],
       [ 1.3970852 ,  -7.25918415],
       [ 7.27808709,   7.15201886],
       [ 1.06965742,  -8.1648251 ],
       [ 6.37298915,   9.77705761],
       [ 7.24898455,   8.85834104],
       [ 2.09335725,  -7.66278316],
       [ 1.05865542,  -8.43841416],
       [ 6.43807502,   7.85483418],
       [ 6.94948313,   8.75248232],
       [-0.07326715, -11.69999644],
       [ 0.61463602,  -9.51908883],
       [ 1.31977821,  -7.2710667 ],
       [ 2.72532584,  -7.51956557],
       [ 8.20949206,  11.90419283]]),
array([1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1,
       1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1,
       1, 0, 0, 0, 0, 1]))
```

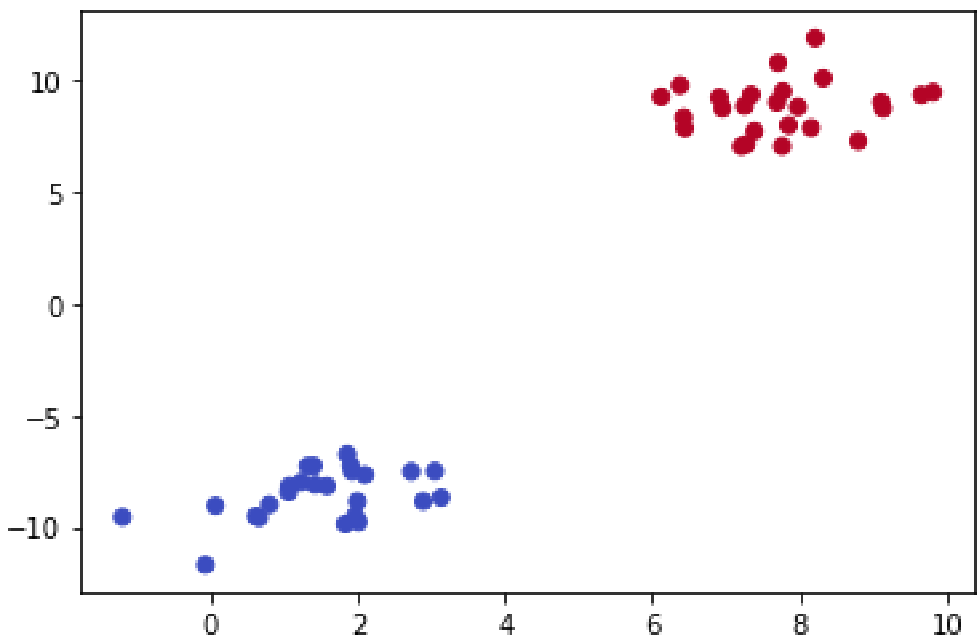
[39]:
```
features = data[0]
plt.scatter(features[:,0],features[:,1])
```

[39]: <matplotlib.collections.PathCollection at 0x212dde03a08>



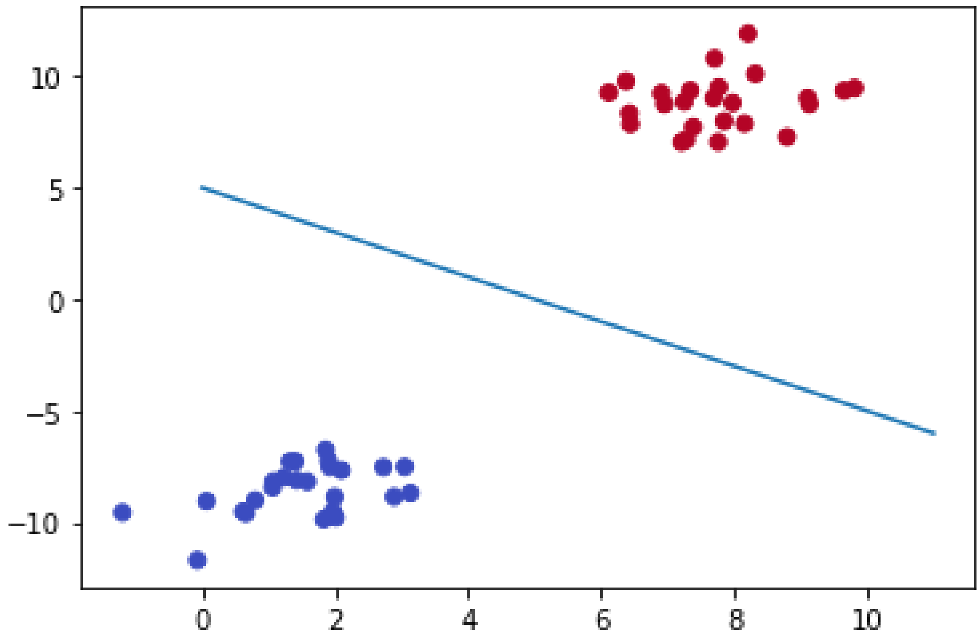
[40]:
```
labels = data[1]
plt.scatter(features[:,0],features[:,1],c=labels,cmap='coolwarm')
```

[40]: <matplotlib.collections.PathCollection at 0x212dde6a508>



[41]:
```python
# DRAW A LINE THAT SEPERATES CLASSES
x = np.linspace(0,11,10)
y = -x + 5
plt.scatter(features[:,0],features[:,1],c=labels,cmap='coolwarm')
plt.plot(x,y)
```

[41]: [<matplotlib.lines.Line2D at 0x212ddea6908>]



1.10 Defining the Perceptron

$$y = mx + b$$

$$y = -x + 5$$

$$f1 = mf2 + b, m = 1$$

$$f1 = -f2 + 5$$

$$f1 + f2 - 5 = 0$$

1.10.1 Convert to a Matrix Representation of Features

$$w^T x + b = 0$$

$$\begin{pmatrix} 1, 1 \end{pmatrix} f - 5 = 0$$

Then if the result is > 0 its label 1, if it is less than 0, it is label=0

1.10.2 Example Point

Let's say we have the point f1=2 , f2=2 otherwise stated as (8,10). Then we have:

$$(1, 1) \begin{pmatrix} 8 \\ 10 \end{pmatrix} + 5 =$$

[42]:
```
np.array([1, 1]).dot(np.array([[8],[10]])) - 5
```

[42]:
```
array([13])
```

Or if we have (4,-10)

[43]:
```
np.array([1,1]).dot(np.array([[4],[-10]])) - 5
```

[43]:
```
array([-11])
```

1.10.3 Using an Example Session Graph

[44]:
```
g = Graph()
```

[45]:
```
g.set_as_default()
```

[46]:
```
x = Placeholder()
```

[47]:
```
w = Variable([1,1])
```

[48]:
```
b = Variable(-5)
```

[49]:
```
z = add(matmul(w,x),b)
```

[50]:
```
a = Sigmoid(z)
```

[51]:
```
sess = Session()
```

[52]:
```
sess.run(operation=a,feed_dict={x:[8,10]})
```

[52]:
```
0.999997739675702
```

[53]:
```
sess.run(operation=a,feed_dict={x:[0,-10]})
```

```
[53]: 3.059022269256247e-07
```

2 Great Job!