MLP

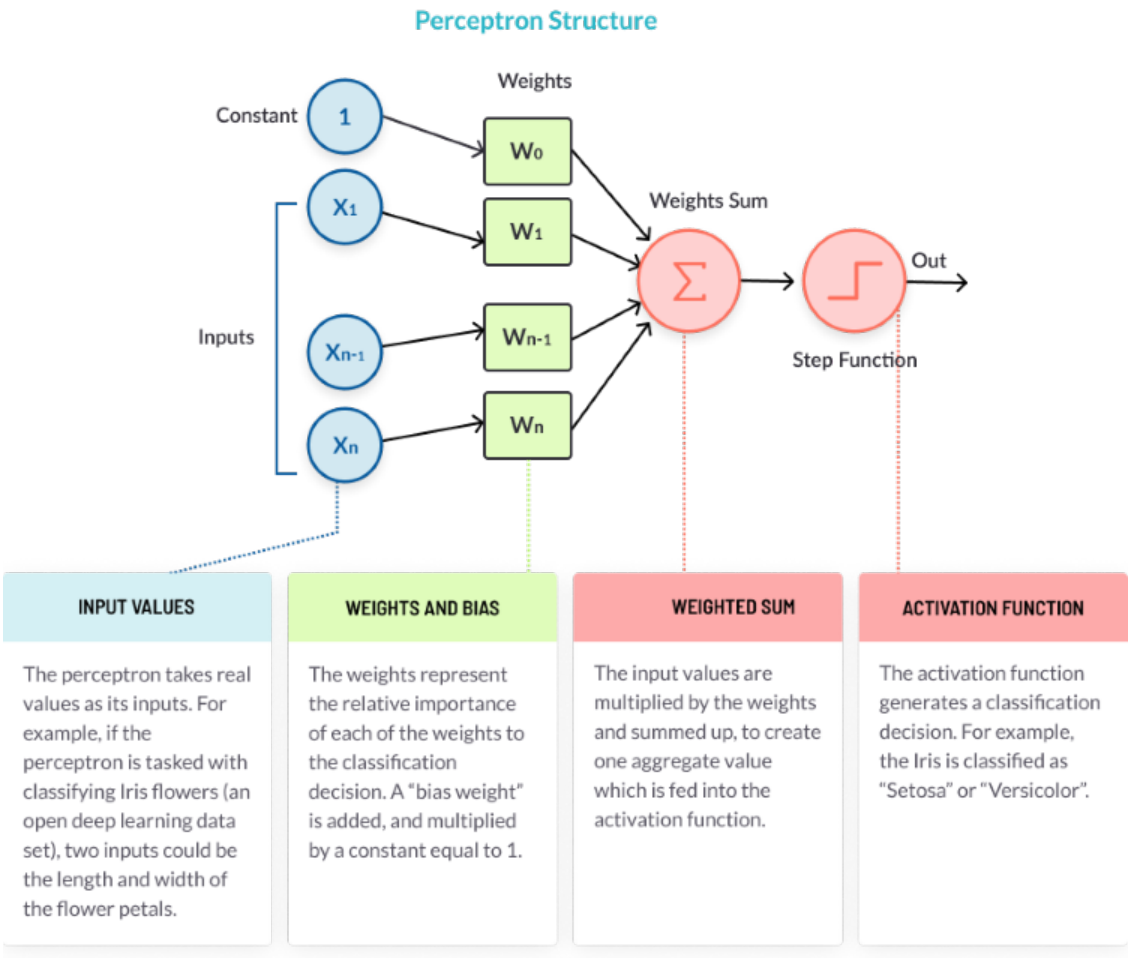
December 3, 2020

Thony Yan PID:3913880

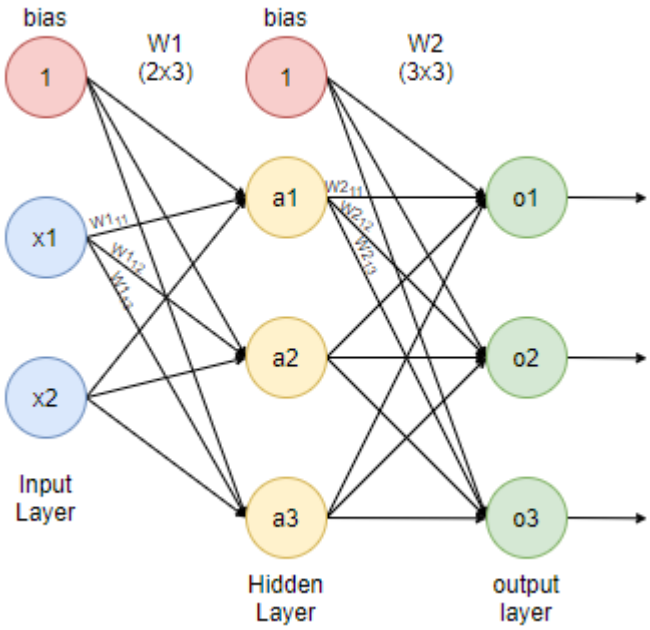
[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import copy
np.random.seed(0)
```

1 Multilayer Perceptron (MLP)

To understand a multilayer perceptron, we must see how a regular perceptron function. A perceptron is a very simple unit for learning machine. It does this by taking an input and multiplying it by their associated weights. The weights signify how important the input is. Now when you have multiple perceptrons, it forms a multilayer perceptron.[1]



A multilayer perceptron is a structure where many perceptrons are stacked to form different layers to solve relatively complex problems. A basic MLP typically has three types of layers, the input layer which are the features we want to predict, the output layer that are the results after passing though the MLP, and the hidden layer which are basically neural networks that sits between the input and output layer. Below is a simple MLP structure with two features, three neurons as the hidden layer, and three outputs in the output layer. Note: the bias in the layers store as a value of one that makes it possible for the activation function be able to adjust.

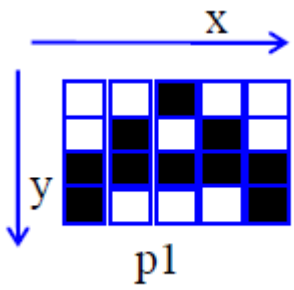


Now in this project I will implement the MLP structure in Python. After the creating of the, I will implement the training algorithm that occurs in an MLP. Which consists of the feed forward calculation, the backpropagation, and finally updating the weights so that the neural network learns from the features pass through the network.

1.1 The Data

First we must create the data to train on. We will be making an array of signed binary 1s that when arrange into a 5 by 5 matrix will display an image of the 5 vowels of the English alphabet *a ,e, i, o, u*. After creating the vowels we will create 4 more images from the original vowels but each image will have 1 pixel change (the 1 will turn to a -1 and vice versa).

In this following example we will see how it is suppose to look:



[2]:
```
p1 = np.array([-1,-1,1,-1,-1,-1,1,-1,1,-1,1,1,1,1,1,1,-1,-1,-1,1])
p6 = np.array([1,1,1,1,1,1,-1,-1,-1,-1,1,1,1,-1,-1,1,1,1,1,1])
p11 = np.array([-1,1,1,1,-1,-1,-1,1,-1,-1,-1,-1,1,-1,-1,-1,1,1,1,-1])
p16 = np.array([1,1,1,1,1,1,-1,-1,-1,1,1,-1,-1,-1,1,1,1,1,1,1])
```

```
p21 = np.array([1,-1,-1,-1,1,1,-1,-1,-1,1,1,-1,-1,-1,1,1,1,1,1,1])
```

[3]:
```
p1 = p1.reshape(4,5)
p6 = p6.reshape(4,5)
p11 = p11.reshape(4,5)
p16 = p16.reshape(4,5)
p21 = p21.reshape(4,5)
```

[4]:
```
p1.shape
```

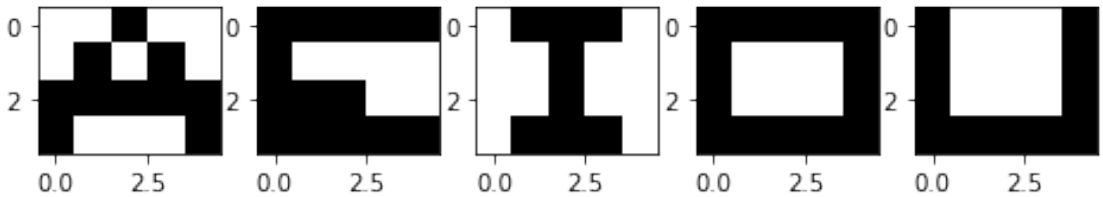
[4]:
```
(4, 5)
```

[5]:
```
base = [p1,p6,p11,p16,p21]
```

[6]:
```
def show(figs): # This function is use to show image
    img=plt.figure(figsize=(10, 10))
    for i in range(len(figs)):
        img.add_subplot(5, 5, i+1)
        plt.imshow(figs[i]-1, cmap='Greys')
```

In the following snippet of code we can see all 5 vowels display.

[7]:
```
img=plt.figure(figsize=(8, 8))
for i in range(1,6):
    img.add_subplot(5, 5, i)
    plt.imshow(base[i-1]-1, cmap='Greys')
```



Here we are getting the x and y coordinates to invert the pixel. Example we will get the original vowel A matrix and invert the pixel located at (2,1) to create a variation of A. Then we will repeat this process for (3,2) and so on.

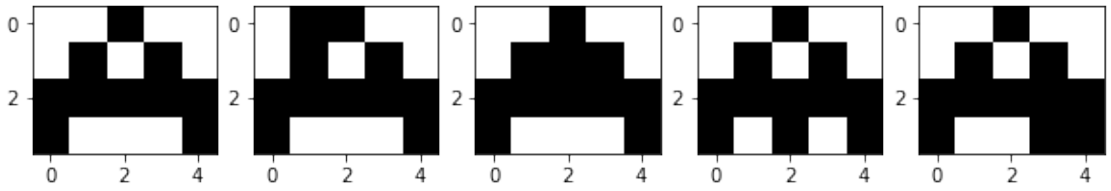
[8]:
```
a_mod = np.array([[2,1],[3,2],[3,4],[4,4]])
e_mod = np.array([[2,2],[4,2],[5,3],[4,3]])
i_mod = np.array([[1,1],[4,2],[1,4],[4,3]])
o_mod = np.array([[2,2],[4,2],[2,3],[4,3]])
u_mod = np.array([[2,1],[4,1],[2,3],[4,3]])
```

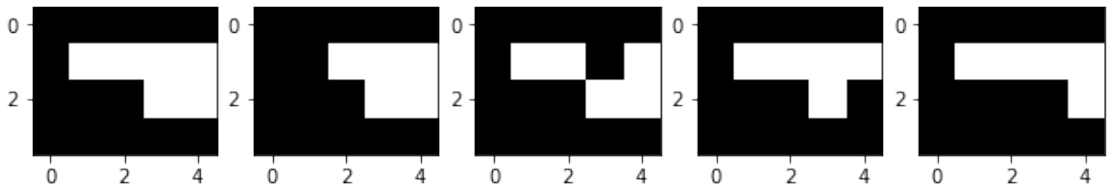
[9]:
```
mod = [a_mod,e_mod,i_mod,o_mod,u_mod]
```

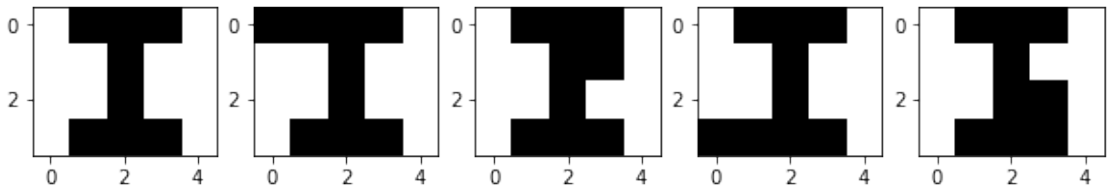
[10]:
```
def modify(base, mod):
    lst = []
    lst.append(base)
    for i in range(len(mod)):
        tmp = copy.deepcopy(base)
        tmp[mod[i][1]-1,mod[i][0]-1] *= -1
        lst.append(tmp)

    return lst
```

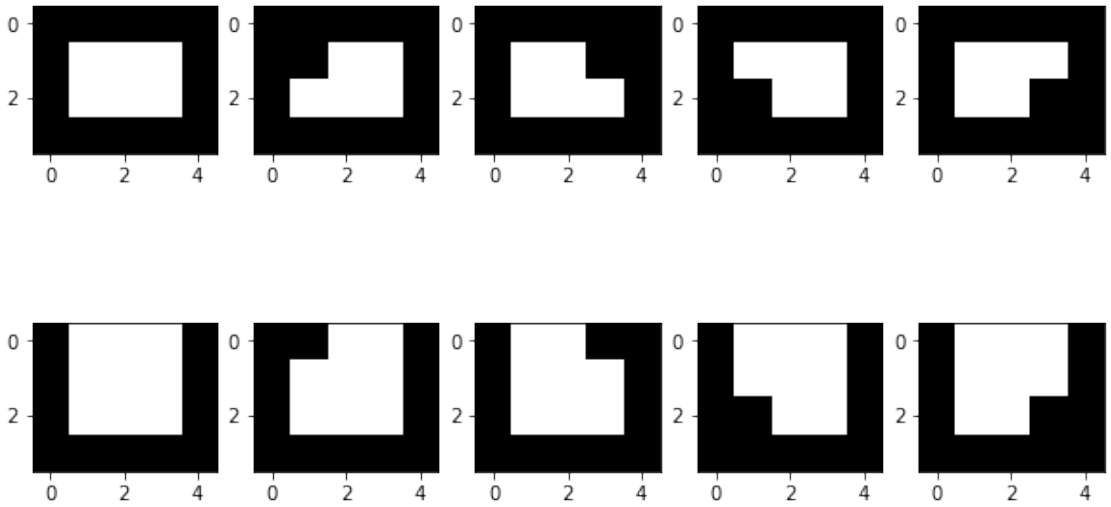
[11]:
```
a_test = modify(base[0], mod[0])
e_test = modify(base[1], mod[1])
i_test = modify(base[2], mod[2])
o_test = modify(base[3], mod[3])
u_test = modify(base[4], mod[4])
```

[12]:
```
show(a_test)
show(e_test)
show(i_test)
show(o_test)
show(u_test)
```









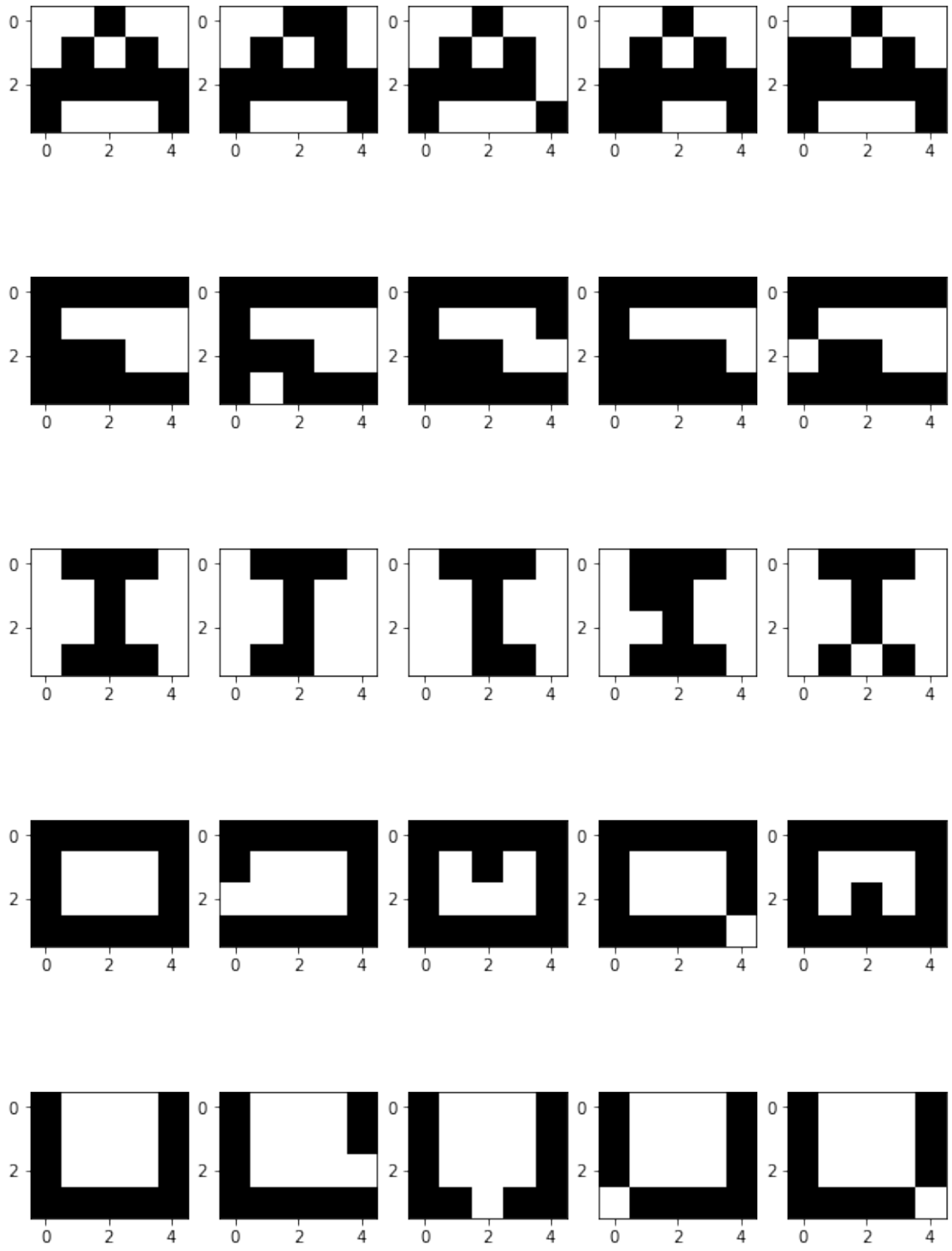
Above are the vowels and their 4 variations that we will use as the training set for our perceptron and adaline algorithm. Next we will be creating the test sets. In test set 1 we will still have the original vowels but now with a different 1 pixel variation.

[13]:
```
test_set = np.array([a_test,e_test,i_test,o_test,u_test])
```

[14]:
```
tset_mod1 = np.array([[ [4,1],[5,3],[2,4],[1,2] ],  #tset1 a
                      [ [2,4],[5,2],[4,3],[1,3] ],  #tset1 e
                      [ [4,4],[2,4],[2,2],[3,4] ],  #tset1 i
                      [ [1,3],[3,2],[5,4],[3,3] ],  #tset1 o
                      [ [5,3],[3,4],[1,4],[5,4] ]]) #tset1 u
```

[15]:
```
tset1_a = modify(base[0], tset_mod1[0])
tset1_e = modify(base[1], tset_mod1[1])
tset1_i = modify(base[2], tset_mod1[2])
tset1_o = modify(base[3], tset_mod1[3])
tset1_u = modify(base[4], tset_mod1[4])
```

[16]:
```
show(tset1_a)
show(tset1_e)
show(tset1_i)
show(tset1_o)
show(tset1_u)
```



[17]:
```
tset1 = np.array([tset1_a,tset1_e,tset1_i,tset1_o,tset1_u])
```

For test set 2 we will be grabbing the vowel variations from test set 1 and inverting 1

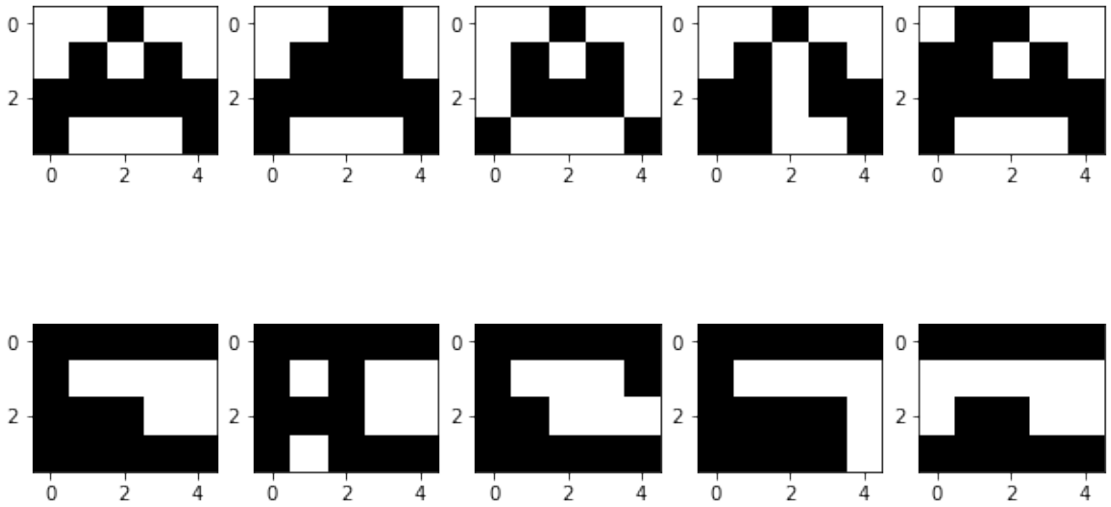
more pixel.

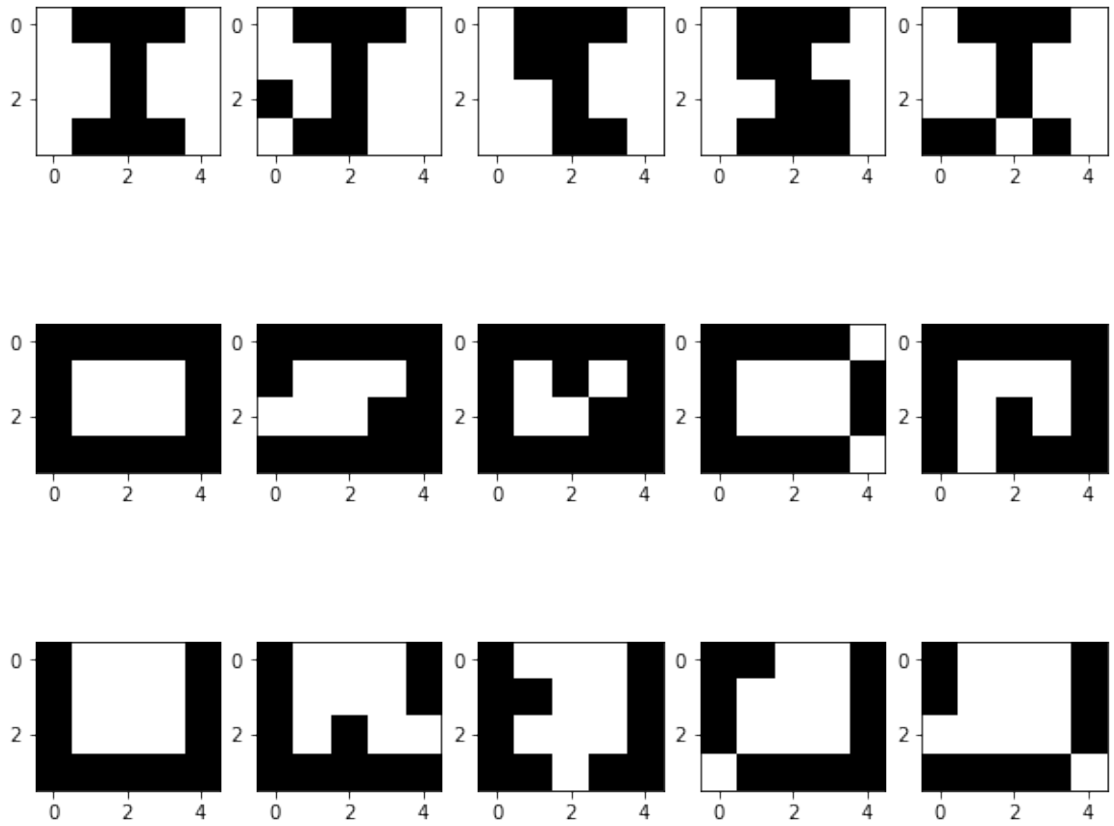
[18]:
```
tset_mod2 = np.array([[ [3,2],[1,3],[3,3],[2,1] ], #tset2 a
                      [ [3,2],[3,3],[5,4],[1,2] ], #tset2 e
                      [ [1,3],[2,2],[4,3],[1,4] ], #tset2 i
                      [ [4,3],[4,3],[5,1],[2,4] ], #tset2 o
                      [ [3,3],[2,2],[2,1],[1,3] ]])#tset2 u
```

[19]:
```
def modify2(base,mod):
    lst = []
    lst.append(base[0])
    for i in range(1,5):
        tmp = copy.deepcopy(base[i])
        tmp[mod[i-1][1]-1,mod[i-1][0]-1] *= -1
        lst.append(tmp)
    return lst
```

[20]:
```
tset2_a = modify2(tset1_a,tset_mod2[0])
tset2_e = modify2(tset1_e,tset_mod2[1])
tset2_i = modify2(tset1_i,tset_mod2[2])
tset2_o = modify2(tset1_o,tset_mod2[3])
tset2_u = modify2(tset1_u,tset_mod2[4])
```

[21]:
```
show(tset2_a)
show(tset2_e)
show(tset2_i)
show(tset2_o)
show(tset2_u)
```





[22]:
```
tset2 = np.array([tset2_a,tset2_e,tset2_i,tset2_o,tset2_u])
```

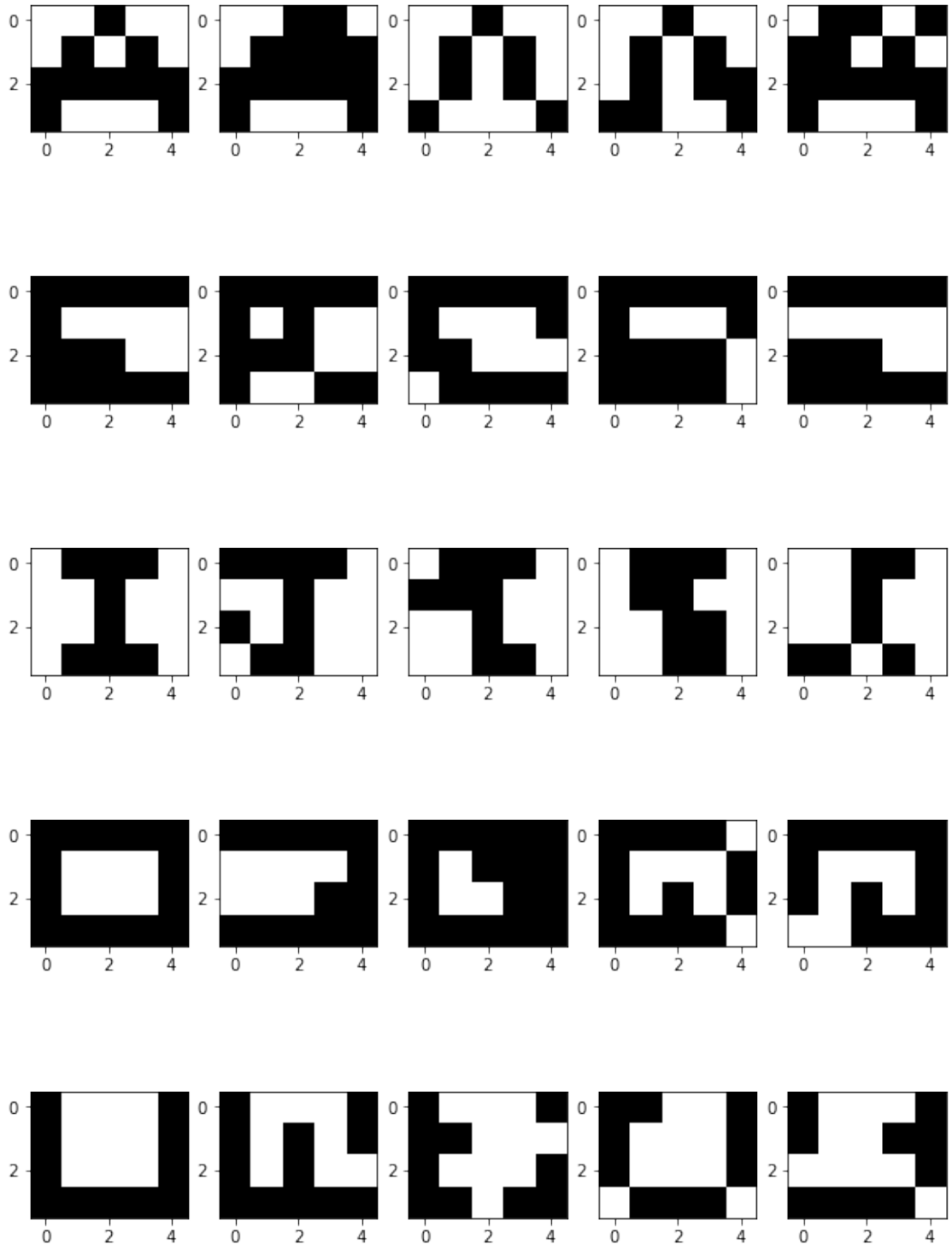
We will be using the same process to create test set 3 but using the variations of test set 2

[23]:
```
tset_mod3 = np.array([[ [5,2],[3,3],[1,3],[5,1] ], #tset3 a
                      [ [3,4],[1,4],[5,2],[1,3] ], #tset3 e
                      [ [1,1],[1,2],[2,4],[2,1] ], #tset3 i
                      [ [1,2],[4,2],[3,3],[1,4] ], #tset3 o
                      [ [3,2],[5,2],[5,4],[4,2] ]])#tset3 u
```

[24]:
```
tset3_a = modify2(tset2_a,tset_mod3[0])
tset3_e = modify2(tset2_e,tset_mod3[1])
tset3_i = modify2(tset2_i,tset_mod3[2])
tset3_o = modify2(tset2_o,tset_mod3[3])
tset3_u = modify2(tset2_u,tset_mod3[4])
```

[25]:
```
show(tset3_a)
show(tset3_e)
show(tset3_i)
show(tset3_o)
```

```
show(tset3_u)
```



[26]:
```
tset3 = np.array([tset3_a,tset3_e,tset3_i,tset3_o,tset3_u])
```

1.2 The Neural Network Training Cycle

1.2.1 Feedforward

Feedforward or the forward propagation is the calculation and storage of intermediate variables from the input layer all the way to the output layer. we first get the sum of the weights multiply by the inputs and adding the bias, then we apply the activation function to get the activation value and use that as the output to pass it to the next consecutive layer.

Forward Propagation

$$\mathbf{a}^0 = \mathbf{p},$$

$$\mathbf{a}^{m+1} = \mathbf{f}^{m+1}(\mathbf{W}^{m+1}\mathbf{a}^m + \mathbf{b}^{m+1}) \text{ for } m = 0, 1, \ldots, M-1,$$

$$\mathbf{a} = \mathbf{a}^M.$$

1.2.2 Back Propagation

Backpropagation is the method of calculating the gradient of the neural networks parameters. We do this by traversing the network in reverse order, meaning we are moving from the output layer to the input layer. By using partial derivatives of the parameters, $F^M(n^M)$, we can calculate the sensitivity to update the weights in the MLP.

Backward Propagation

$$\mathbf{s}^M = -2\dot{\mathbf{F}}^M(\mathbf{n}^M)(\mathbf{t}-\mathbf{a}),$$

$$\mathbf{s}^m = \dot{\mathbf{F}}^m(\mathbf{n}^m)(\mathbf{W}^{m+1})^T\mathbf{s}^{m+1}, \text{ for } m = M-1, \ldots, 2, 1,$$

where

$$\dot{\mathbf{F}}^m(\mathbf{n}^m) = \begin{bmatrix} \dot{f}^m(n_1^m) & 0 & \ldots & 0 \\ 0 & \dot{f}^m(n_2^m) & \ldots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & & \dot{f}^m(n_{S^m}^m) \end{bmatrix},$$

$$\dot{f}^m(n_j^m) = \frac{\partial f^m(n_j^m)}{\partial n_j^m}.$$

1.2.3 Weight update

After calculating all the sensitivity from the backpropagation we can begin updating the weights and bias from the network. We use alpha so we can slowly change the weights as dramatic changes to the weights will not produce desire results

Weight Update (Approximate Steepest Descent)

$$\mathbf{W}^m(k+1) = \mathbf{W}^m(k) - \alpha \mathbf{s}^m (\mathbf{a}^{m-1})^T,$$

$$\mathbf{b}^m(k+1) = \mathbf{b}^m(k) - \alpha \mathbf{s}^m.$$

[27]:
```
class Neural_Network:

    def __init__(self):
        self.weights = [] # weight matrices
        self.bias = [] # bias matrices
        self.activation = [] # activation functions
        self.z_val = [np.zeros(1)] # sum values of neurons. note: first value␣
↪suppose to be inputs
        self.a_val = [np.zeros(1)] # values after activation functions are␣
↪apply. note: first value suppose to be inputs
        self.sensitivity = [] # sensitivity or delta (derivatives)


    def add_layer(self, neurons: int, activation: str, input_shape=None):
        if input_shape is None:
            try:
                w = np.random.uniform(0,0.25,(self.weights[-1].
↪shape[1],neurons))
                self.weights.append(w)
            except IndexError:
                w = np.random.uniform(0,0.25,(1,neurons))
                self.weights.append(w)

        else:
            input = np.prod(input_shape)
            w = np.random.uniform(0,0.25,(input,neurons))
            self.weights.append(w)

        self.bias.append(np.random.rand(neurons))
        self.activation.append(self.activation_f(activation))
        self.z_val.append(np.zeros(neurons))
        self.a_val.append(np.zeros(neurons))
```

```
    @staticmethod
    def sigmoid(x, derivative: bool=False):
        z = 1/(1+np.exp(-x))
        if derivative:
            z = z * (1-z)
        return z

    @staticmethod
    def tanh(x, derivative: bool=False):
        z =  (1-np.exp(-2*x)) / (1+np.exp(-2*x))
        if derivative:
            z = (1 + z) * (1 - z)
        return z

    @staticmethod
    def linear(x, derivative: bool=False):
        if derivative:
            return np.array(1)
        return x

    def activation_f(self, activation_name: str):

        activation = {
            'linear' : self.linear,
            'sigmoid' : self.sigmoid,
            'tanh' : self.tanh
        }

        act = str.lower(activation_name)

        if act in activation:
            return activation[act]
        else:
            print("activation function not in record")

    @staticmethod
    def mse(error):

        mse = np.sum(error ** 2)
        return mse

    @staticmethod
    def error(target, output):
        error = (target - output)
        return error

    @staticmethod
```

```
    def hardlim(output, tresh=0.32):

        output[output>tresh] = 1
        output[output<tresh] = -1
        return output

    @staticmethod
    def max_arg(output):

        index = output.argmax()
        output.fill(-1)
        output[index] = 1
        return output

    def feedforward(self, x):

        self.a_val[0] = x
        self.z_val[0] = x

        for layer in range(len(self.weights)):

            z = np.dot(self.a_val[layer], self.weights[layer]) + self.
↪bias[layer]
            a = self.activation[layer](z)
            self.z_val[layer+1] = z
            self.a_val[layer+1] = a

    def back_propagate(self, error):
        error = -2 * error # -2 * (target-output)
        self.sensitivity = []
        for i in reversed(range(len(self.weights))):

            s = error * self.activation[i](self.z_val[i+1], derivative=True) #␣
↪-2 * FMnM * (t-a)
            self.sensitivity.insert(0,s)
            error = np.dot(s, self.weights[i].T)

    def update_weights(self, alpha=0.1):
        for i in range(len(self.weights)):
            sensitivity = self.sensitivity[i]
            a = self.a_val[i]

            sensitivity = sensitivity.reshape(sensitivity.shape[0],-1)
            a = a.reshape(a.shape[0],-1)

            sa = np.dot(sensitivity,a.T)
```

```
            self.weights[i] = self.weights[i] - (alpha * sa.T)
            self.bias[i] = self.bias[i] - (alpha * self.sensitivity[i])

    def train(self, x, y, alpha=0.01):

        self.feedforward(x)
        error = self.error(y, self.a_val[-1])
        self.back_propagate(error)
        self.update_weights(alpha)
        return self.mse(error)

    def threshold_predict(self,x,thresh=0.32):
        self.a_val[0] = x
        self.z_val[0] = x
        for layer in range(len(self.weights)):

            z = np.dot(self.a_val[layer], self.weights[layer]) + self.
↪bias[layer]
            a = self.activation[layer](z)
            self.z_val[layer+1] = z
            self.a_val[layer+1] = a

        return self.hardlim(a, thresh)

    def max_arg_predict(self,x):
        self.a_val[0] = x
        self.z_val[0] = x
        for layer in range(len(self.weights)):

            z = np.dot(self.a_val[layer], self.weights[layer]) + self.
↪bias[layer]
            a = self.activation[layer](z)
            self.z_val[layer+1] = z
            self.a_val[layer+1] = a

        return self.max_arg(a)
```

[28]:
```
model = Neural_Network()
model.add_layer(10, 'sigmoid', input_shape=(4,5))
model.add_layer(5, 'tanh')
```

The way we are expressing the target output are as follow.

Target component	For "**A** s" (p1 to p5)	For "**E** s" (p6 to p10)	For "**I** s" (p11 to p15)	For "**O** s" (p16 to p20)	For "**U** s" (p21 to p25)
t_1	**+1**	-1	-1	-1	-1
t_2	-1	**+1**	-1	-1	-1
t_3	-1	-1	**+1**	-1	-1
t_4	-1	-1	-1	**+1**	-1
t_5	-1	-1	-1	-1	**+1**

[29]:
```
training_set = a_test, e_test, i_test, o_test, u_test
training_set
```

[29]:
```
([array([[-1, -1,  1, -1, -1],
         [-1,  1, -1,  1, -1],
         [ 1,  1,  1,  1,  1],
         [ 1, -1, -1, -1,  1]]),
  array([[-1,  1,  1, -1, -1],
         [-1,  1, -1,  1, -1],
         [ 1,  1,  1,  1,  1],
         [ 1, -1, -1, -1,  1]]),
  array([[-1, -1,  1, -1, -1],
         [-1,  1,  1,  1, -1],
         [ 1,  1,  1,  1,  1],
         [ 1, -1, -1, -1,  1]]),
  array([[-1, -1,  1, -1, -1],
         [-1,  1, -1,  1, -1],
         [ 1,  1,  1,  1,  1],
         [ 1, -1,  1, -1,  1]]),
  array([[-1, -1,  1, -1, -1],
         [-1,  1, -1,  1, -1],
         [ 1,  1,  1,  1,  1],
         [ 1, -1, -1,  1,  1]])],
 [array([[ 1,  1,  1,  1,  1],
         [ 1, -1, -1, -1, -1],
         [ 1,  1,  1, -1, -1],
         [ 1,  1,  1,  1,  1]]),
  array([[ 1,  1,  1,  1,  1],
         [ 1,  1, -1, -1, -1],
         [ 1,  1,  1, -1, -1],
         [ 1,  1,  1,  1,  1]]),
  array([[ 1,  1,  1,  1,  1],
         [ 1, -1, -1,  1, -1],
         [ 1,  1,  1, -1, -1],
         [ 1,  1,  1,  1,  1]]),
  array([[ 1,  1,  1,  1,  1],
         [ 1, -1, -1, -1, -1],
         [ 1,  1,  1, -1,  1],
```

```
        [ 1,  1,  1,  1,  1]]),
 array([[ 1,  1,  1,  1,  1],
        [ 1, -1, -1, -1, -1],
        [ 1,  1,  1,  1, -1],
        [ 1,  1,  1,  1,  1]])],
[array([[-1,  1,  1,  1, -1],
        [-1, -1,  1, -1, -1],
        [-1, -1,  1, -1, -1],
        [-1,  1,  1,  1, -1]]),
 array([[ 1,  1,  1,  1, -1],
        [-1, -1,  1, -1, -1],
        [-1, -1,  1, -1, -1],
        [-1,  1,  1,  1, -1]]),
 array([[-1,  1,  1,  1, -1],
        [-1, -1,  1,  1, -1],
        [-1, -1,  1, -1, -1],
        [-1,  1,  1,  1, -1]]),
 array([[-1,  1,  1,  1, -1],
        [-1, -1,  1, -1, -1],
        [-1, -1,  1, -1, -1],
        [ 1,  1,  1,  1, -1]]),
 array([[-1,  1,  1,  1, -1],
        [-1, -1,  1, -1, -1],
        [-1, -1,  1,  1, -1],
        [-1,  1,  1,  1, -1]])],
[array([[ 1,  1,  1,  1,  1],
        [ 1, -1, -1, -1,  1],
        [ 1, -1, -1, -1,  1],
        [ 1,  1,  1,  1,  1]]),
 array([[ 1,  1,  1,  1,  1],
        [ 1,  1, -1, -1,  1],
        [ 1, -1, -1, -1,  1],
        [ 1,  1,  1,  1,  1]]),
 array([[ 1,  1,  1,  1,  1],
        [ 1, -1, -1,  1,  1],
        [ 1, -1, -1, -1,  1],
        [ 1,  1,  1,  1,  1]]),
 array([[ 1,  1,  1,  1,  1],
        [ 1, -1, -1, -1,  1],
        [ 1,  1, -1, -1,  1],
        [ 1,  1,  1,  1,  1]]),
 array([[ 1,  1,  1,  1,  1],
        [ 1, -1, -1, -1,  1],
        [ 1, -1, -1,  1,  1],
        [ 1,  1,  1,  1,  1]])],
[array([[ 1, -1, -1, -1,  1],
        [ 1, -1, -1, -1,  1],
```

```
        [ 1, -1, -1, -1,  1],
        [ 1,  1,  1,  1,  1]]),
 array([[ 1,  1, -1, -1,  1],
        [ 1, -1, -1, -1,  1],
        [ 1, -1, -1, -1,  1],
        [ 1,  1,  1,  1,  1]]),
 array([[ 1, -1, -1,  1,  1],
        [ 1, -1, -1, -1,  1],
        [ 1, -1, -1, -1,  1],
        [ 1,  1,  1,  1,  1]]),
 array([[ 1, -1, -1, -1,  1],
        [ 1, -1, -1, -1,  1],
        [ 1,  1, -1, -1,  1],
        [ 1,  1,  1,  1,  1]]),
 array([[ 1, -1, -1, -1,  1],
        [ 1, -1, -1, -1,  1],
        [ 1, -1, -1,  1,  1],
        [ 1,  1,  1,  1,  1]])])
```

[30]:
```
y_set = np.array([[1.,-1,-1,-1,-1],␣
↪[-1,1,-1,-1,-1],[-1,-1,1,-1,-1],[-1,-1,-1,1,-1],[-1,-1,-1,-1,1]])
```

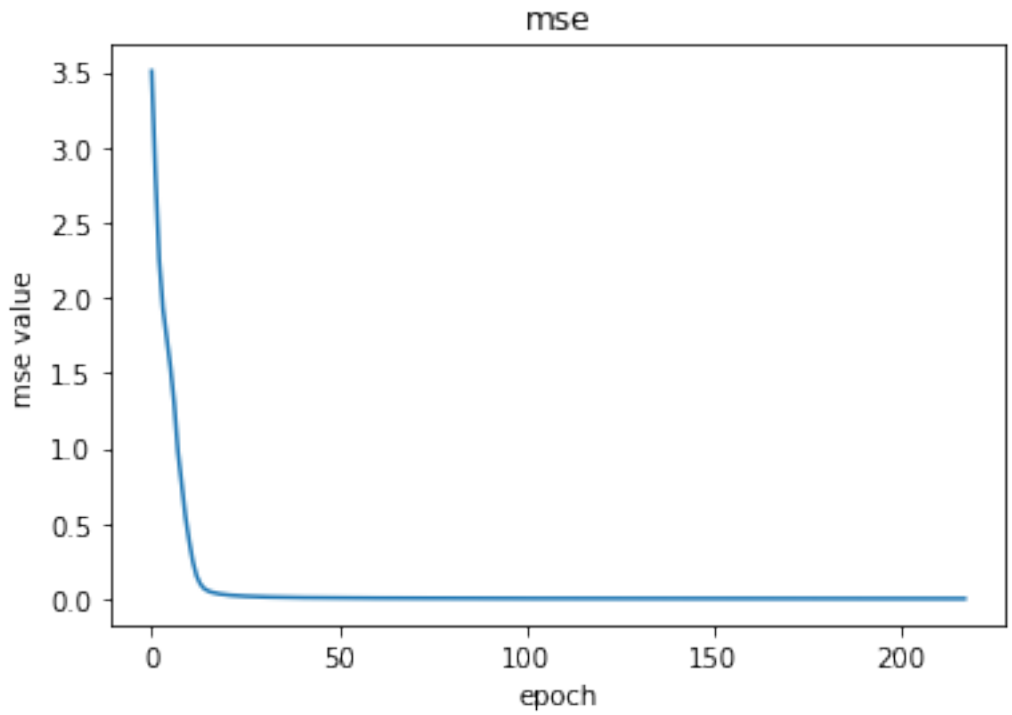
[31]:
```
def fit(model,epochs, alpha, dataset, targets, exit=0.01, input_l=20,␣
↪hidden_l=10, output_l=5):
    total_mse = []
    random_w1 = []
    random_w2 = []
    random_w3 = []
    bias1 = []
    bias2 = []
    random1 = np.random.randint(input_l)
    random2 = np.random.randint(hidden_l)
    random3 = np.random.randint(hidden_l)
    random4 = np.random.randint(output_l)
    random5 = np.random.randint(input_l)
    random6 = np.random.randint(hidden_l)
    bias_random1 = np.random.randint(hidden_l)
    bias_random2 = np.random.randint(output_l)
    for epoch in range(epochs):
        mse = 0
        random_w1.append(model.weights[0][random1][random2])
        random_w2.append(model.weights[1][random3][random4])
        random_w3.append(model.weights[0][random5][random6])
        bias1.append(model.bias[0][bias_random1])
        bias2.append(model.bias[1][bias_random2])

        for i in range(len(dataset)):
```

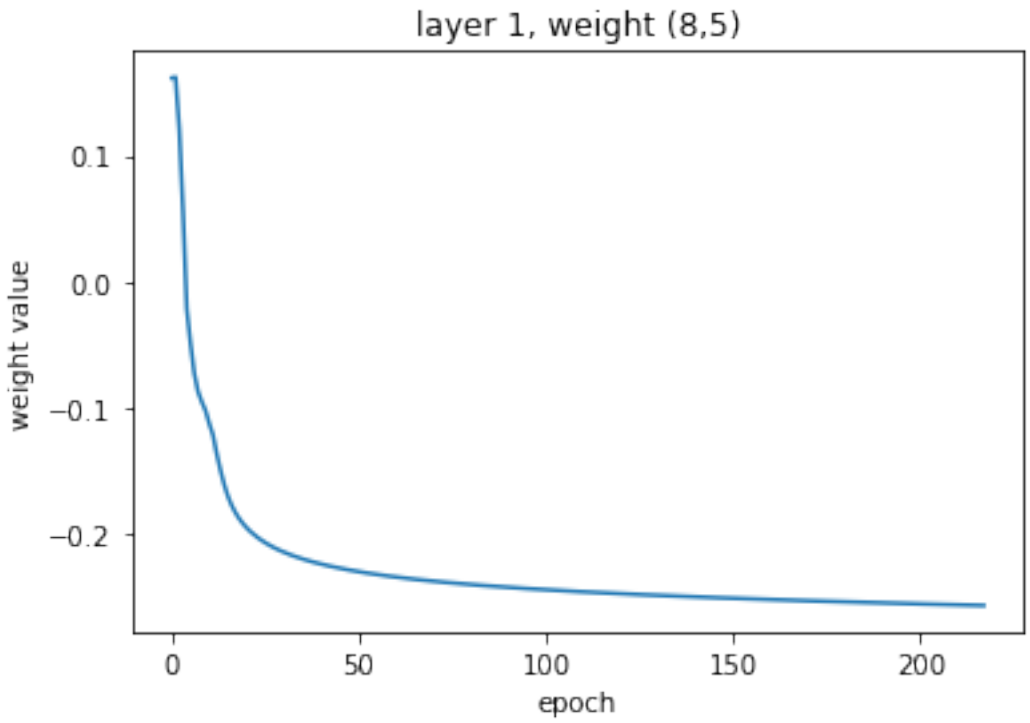
```
            sets = dataset[i]
            target = targets[i]
            for j in range(len(sets)):
                mse += model.train(sets[i].flatten(), target, 0.1)

        mse = (mse / 25.)
        total_mse.append(mse)
        if mse < exit:
            break

    plt.plot(total_mse)
    plt.xlabel('epoch')
    plt.ylabel('mse value')
    plt.title('mse')
    plt.show()

    plt.plot(random_w1)
    plt.xlabel('epoch')
    plt.ylabel('weight value')
    plt.title('layer 1, weight (' + str(random1) + ',' + str(random2) + ')')
    plt.show()

    plt.plot(random_w2)
    plt.xlabel('epoch')
    plt.ylabel('weight value')
    plt.title('layer 2, weight (' + str(random3) + ',' + str(random4) + ')')
    plt.show()

    plt.plot(random_w3)
    plt.xlabel('epoch')
    plt.ylabel('weight value')
    plt.title('layer 1, weight (' + str(random5) + ',' + str(random6) + ')')
    plt.show()

    plt.plot(bias1)
    plt.xlabel('epoch')
    plt.ylabel('bias value')
    plt.title('layer 1, bias (' + str(bias_random1) + ')')
    plt.show()

    plt.plot(bias2)
    plt.xlabel('epoch')
    plt.ylabel('bias value')
    plt.title('layer 2, bias (' + str(bias_random2) + ')')
    plt.show()

    print('\n number of epochs to reach an mse of ' , epoch)
```

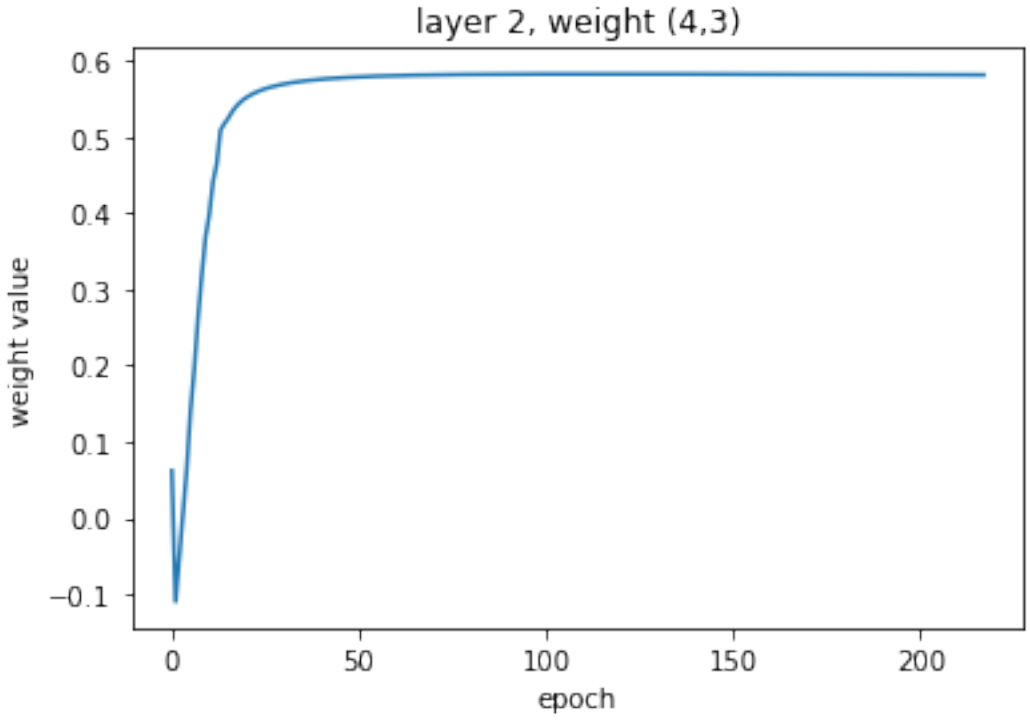
During training we want to minimize the error which is the loss function. In this experiments we will be using the Mean Squared Error as our loss function. By calculating the MSE at each epoch we can see how well the model is learning and use it as a metric to tweak hyper parameters for better testing

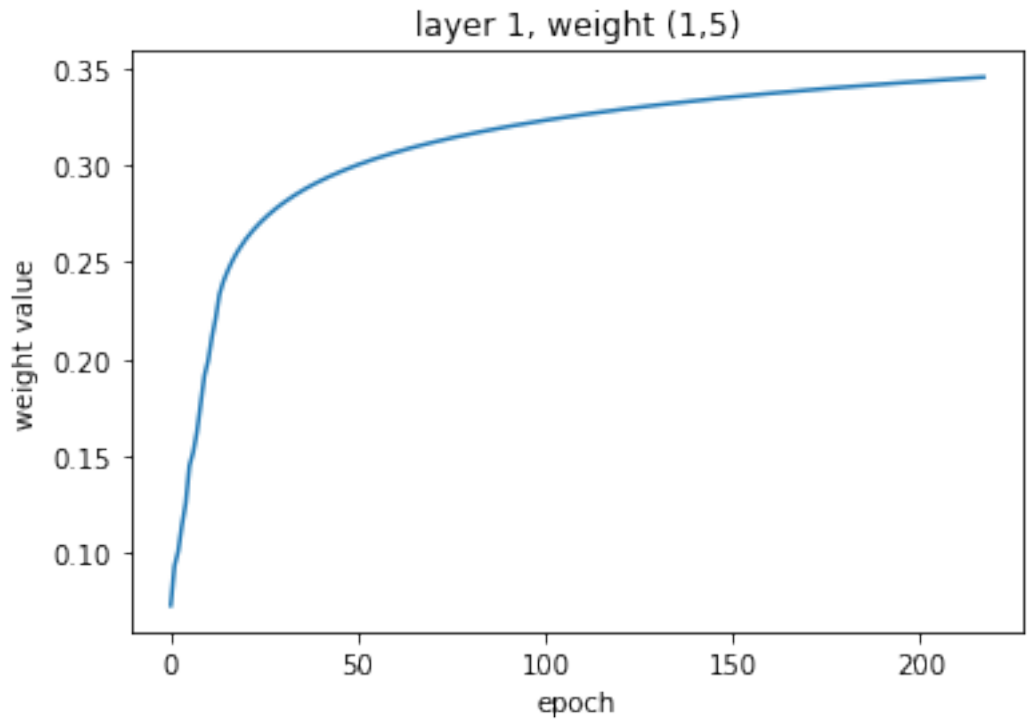
[32]:
```
model1 = Neural_Network()
model1.add_layer(10, 'sigmoid', input_shape=(4,5))
model1.add_layer(5, 'tanh')
```

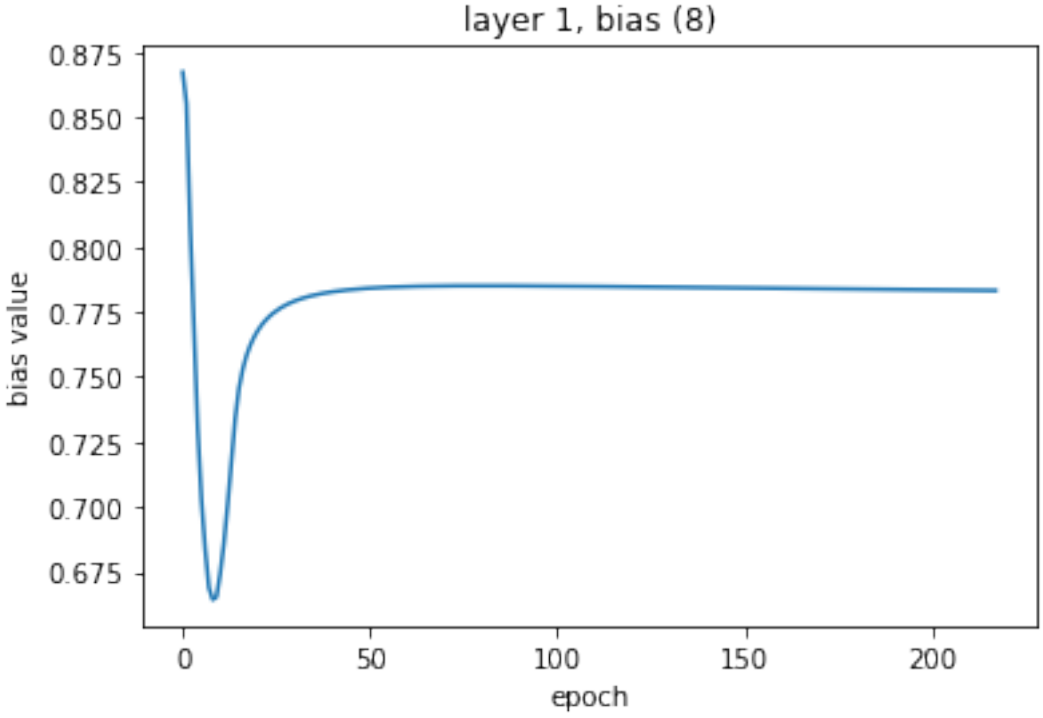
[33]:
```
fit(model1,epochs=250, alpha=0.1, dataset=training_set, targets=y_set, exit=0.
↪001)
```

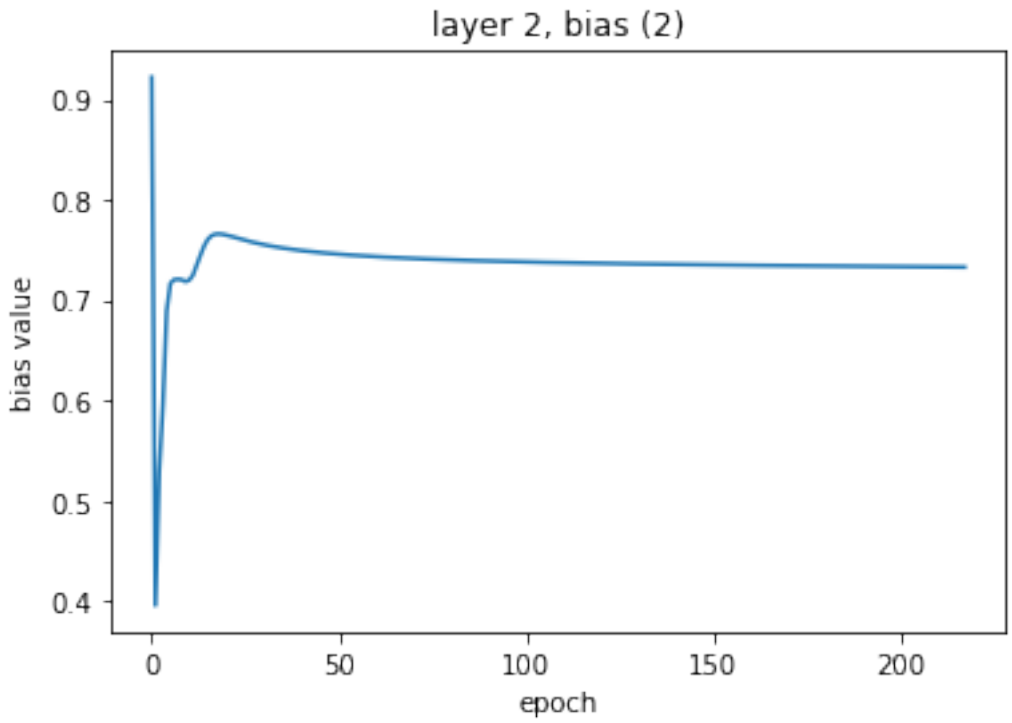



layer 1, weight (8,5)
weight value
0.1
0.0
−0.1
−0.2
0
50
100
150
200
epoch


layer 2, weight (4,3)
weight value
0.6
0.5
0.4
0.3
0.2
0.1
0.0
−0.1
0
50
100
150
200
epoch


layer 1, weight (1,5)
weight value
0.35
0.30
0.25
0.20
0.15
0.10
0
50
100
150
200
epoch

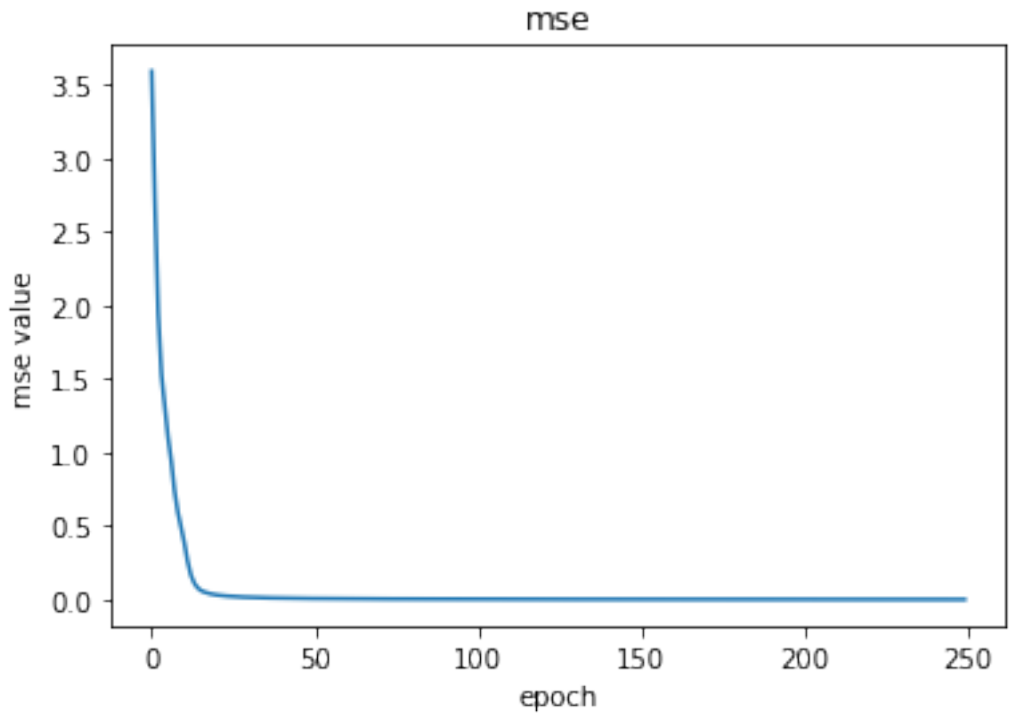

layer 1, bias (8)
bias value
0.875
0.850
0.825
0.800
0.775
0.750
0.725
0.700
0.675
0
50
100
150
200
epoch

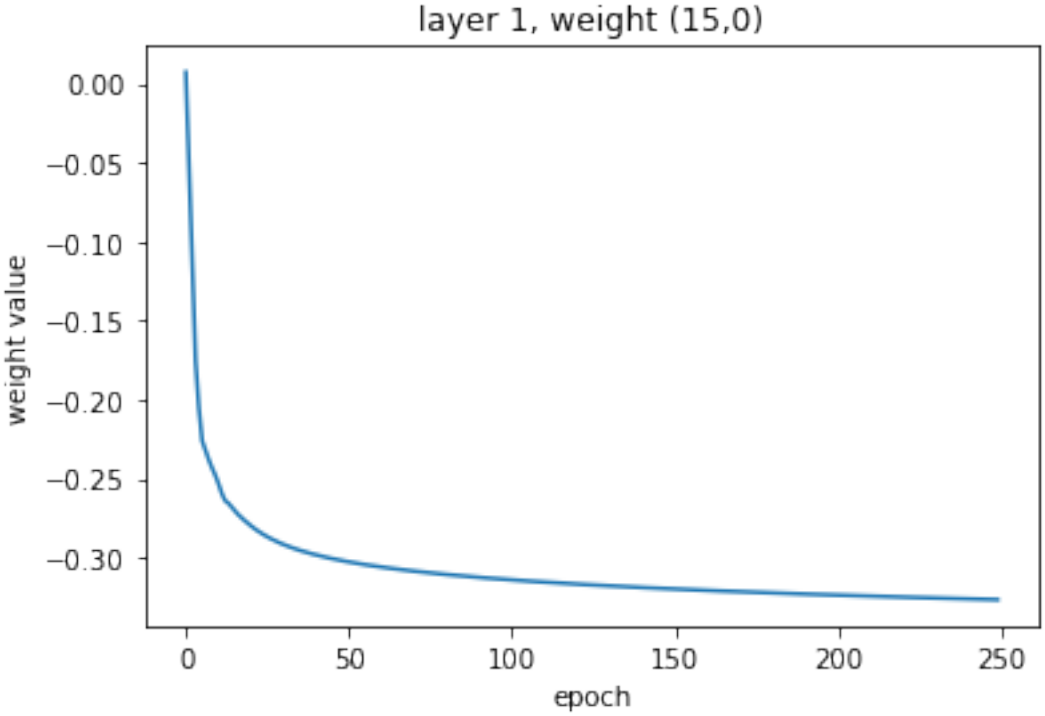


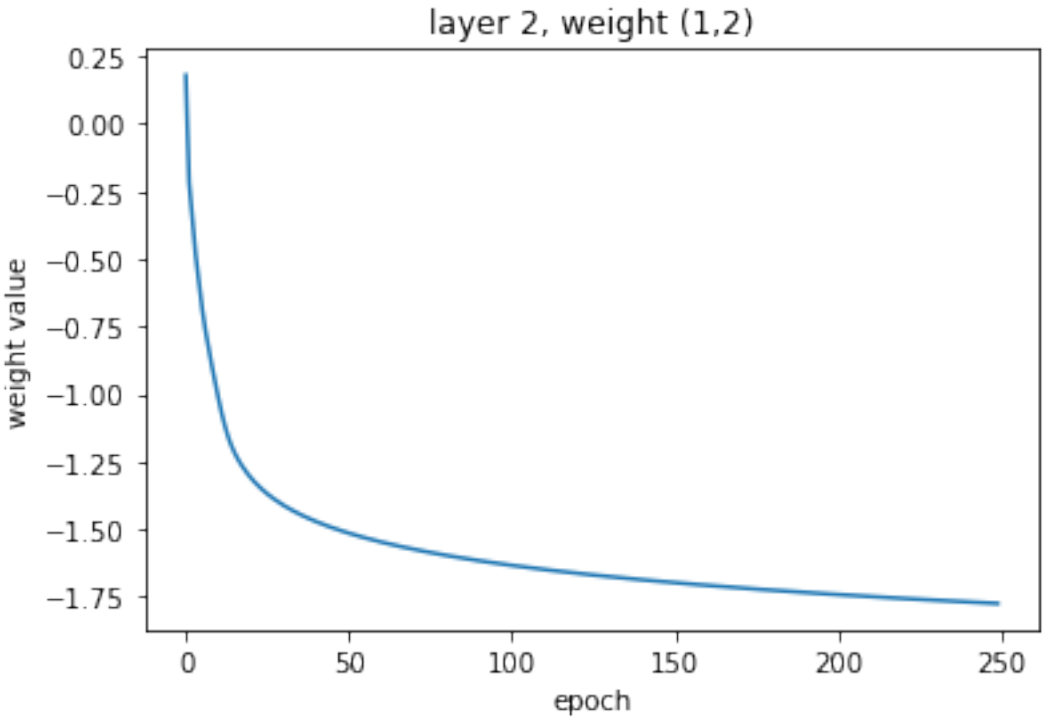

```
number of epochs to reach an mse of  217
```

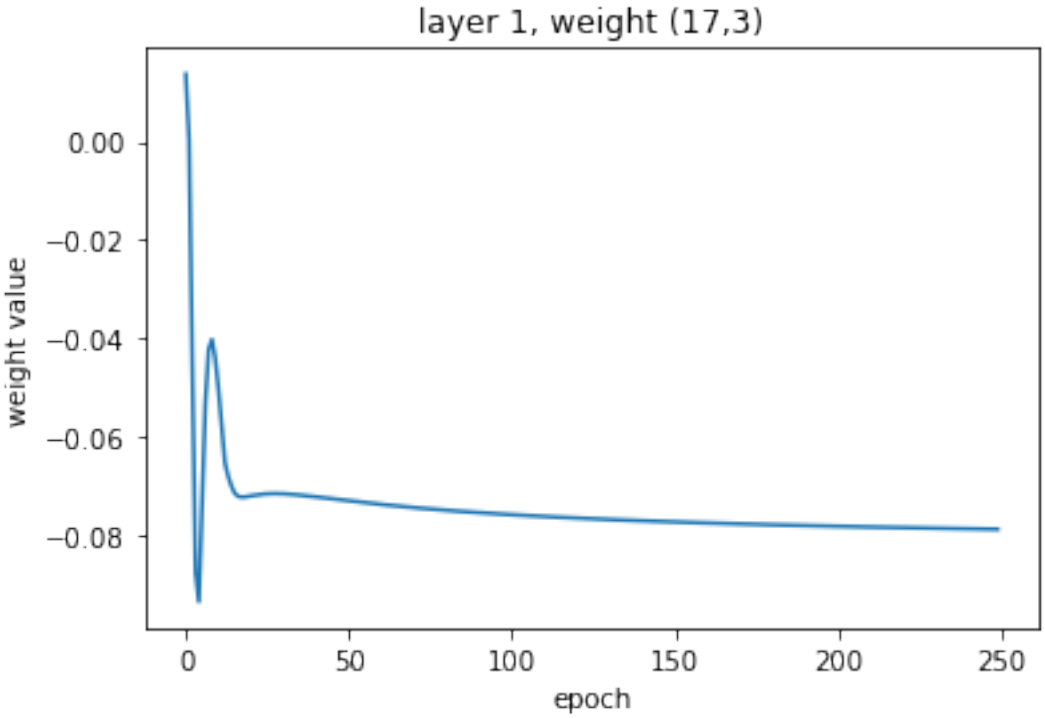
[34]:
```
model2 = Neural_Network()
model2.add_layer(10, 'sigmoid', input_shape=(4,5))
model2.add_layer(5, 'tanh')
```

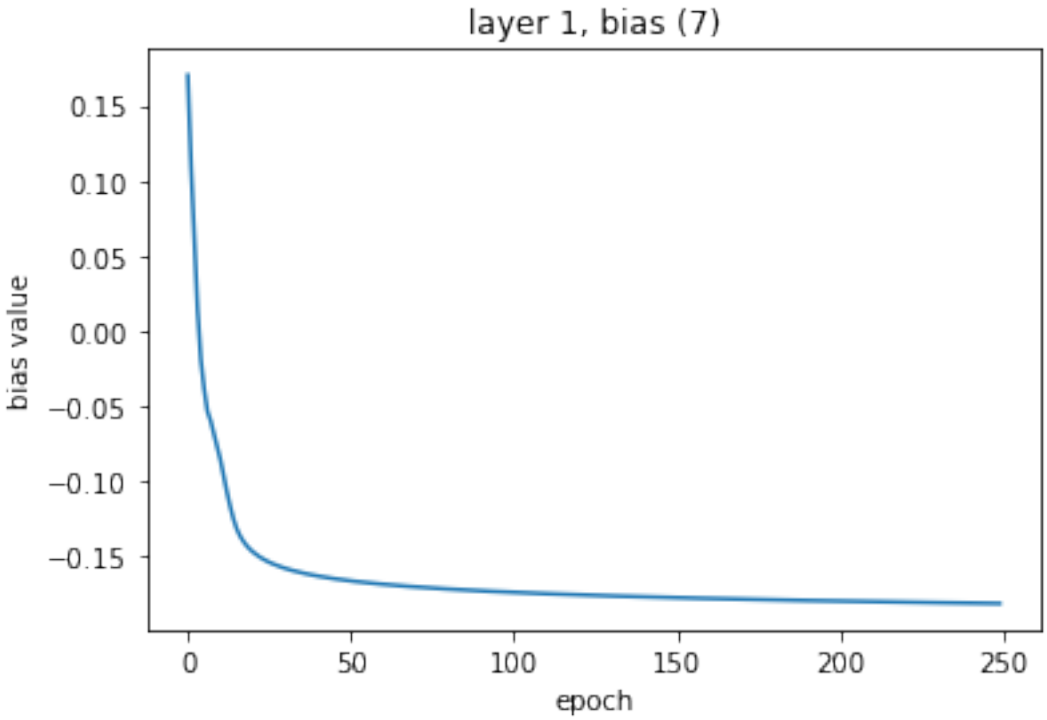
[35]:
```
fit(model2,epochs=250, alpha=0.01, dataset=training_set, targets=y_set, exit=0.
↪001)
```

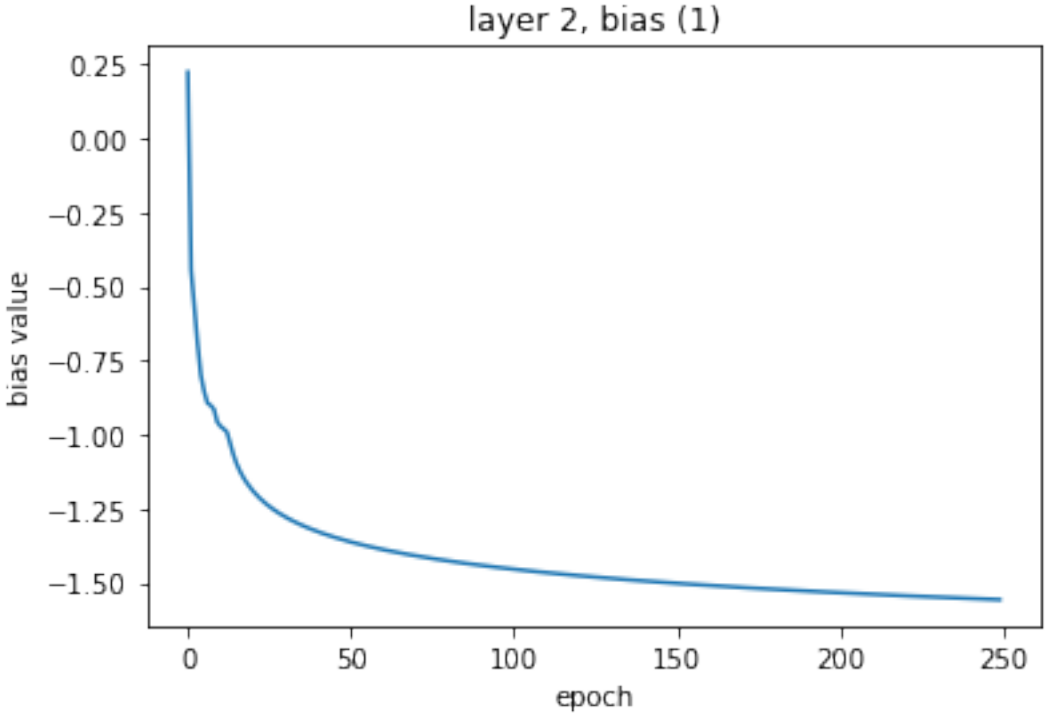

mse
mse value
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
50
100
150
200
250
epoch


layer 1, weight (15,0)
weight value
0.00
−0.05
−0.10
−0.15
−0.20
−0.25
−0.30
0
50
100
150
200
250
epoch


layer 2, weight (1,2)
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.25
−1.50
−1.75
weight value
0
50
100
150
200
250
epoch


layer 1, weight (17,3)
0.00
−0.02
−0.04
−0.06
−0.08
weight value
0
50
100
150
200
250
epoch

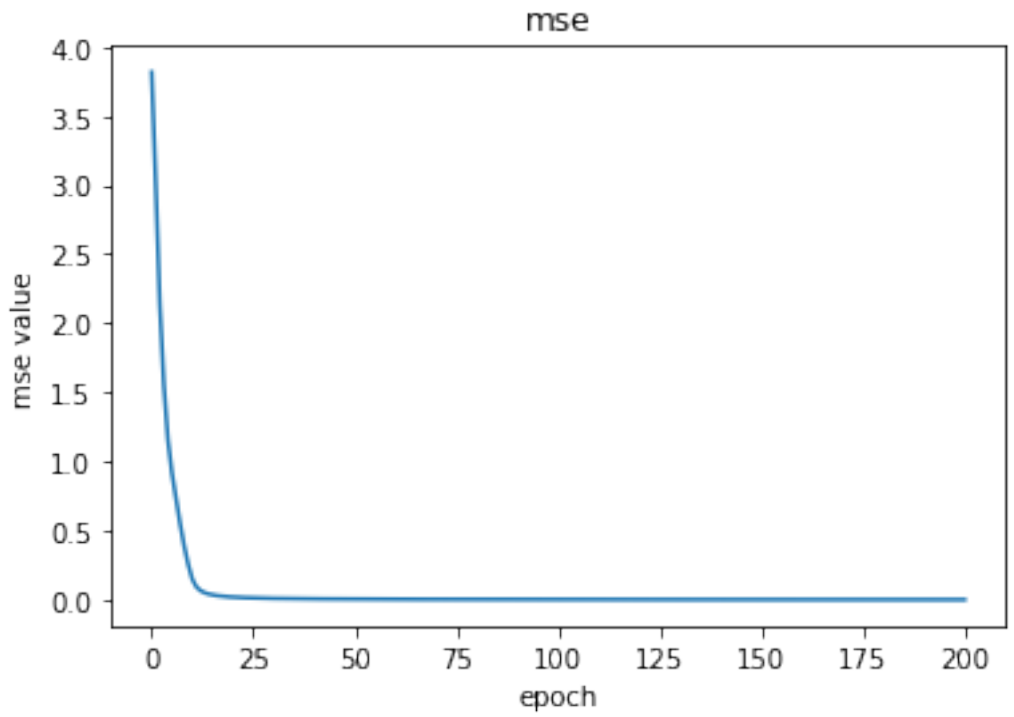

layer 1, bias (7)
bias value
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
0
50
100
150
200
250
epoch

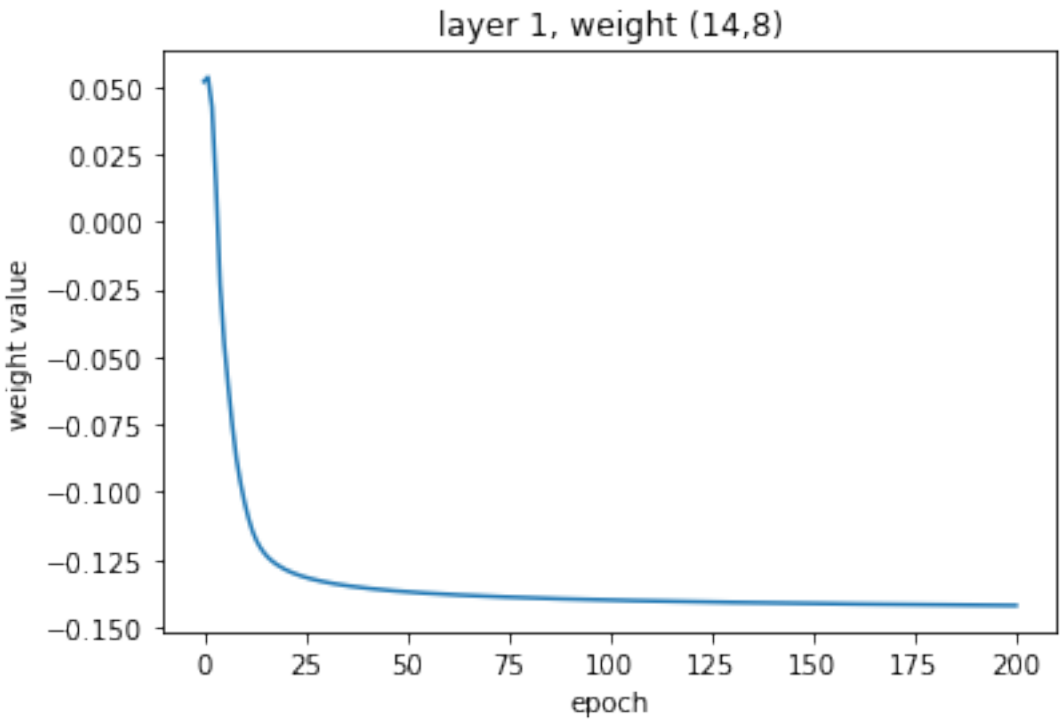

layer 2, bias (1)
bias value
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.25
−1.50
0
50
100
150
200
250
epoch

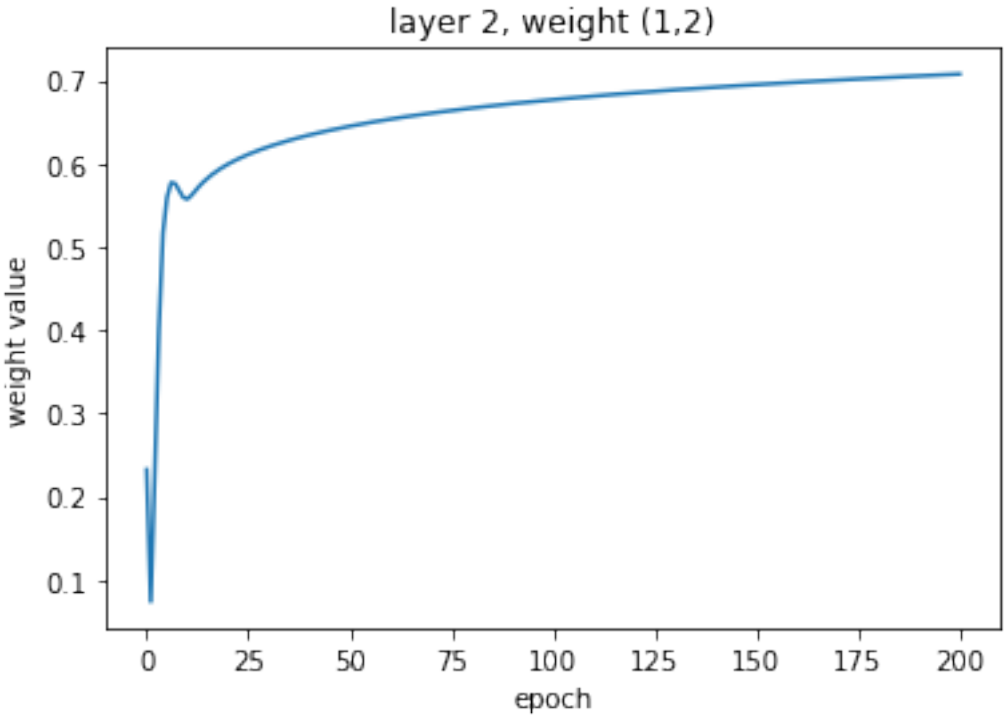
```
number of epochs to reach an mse of  249
```

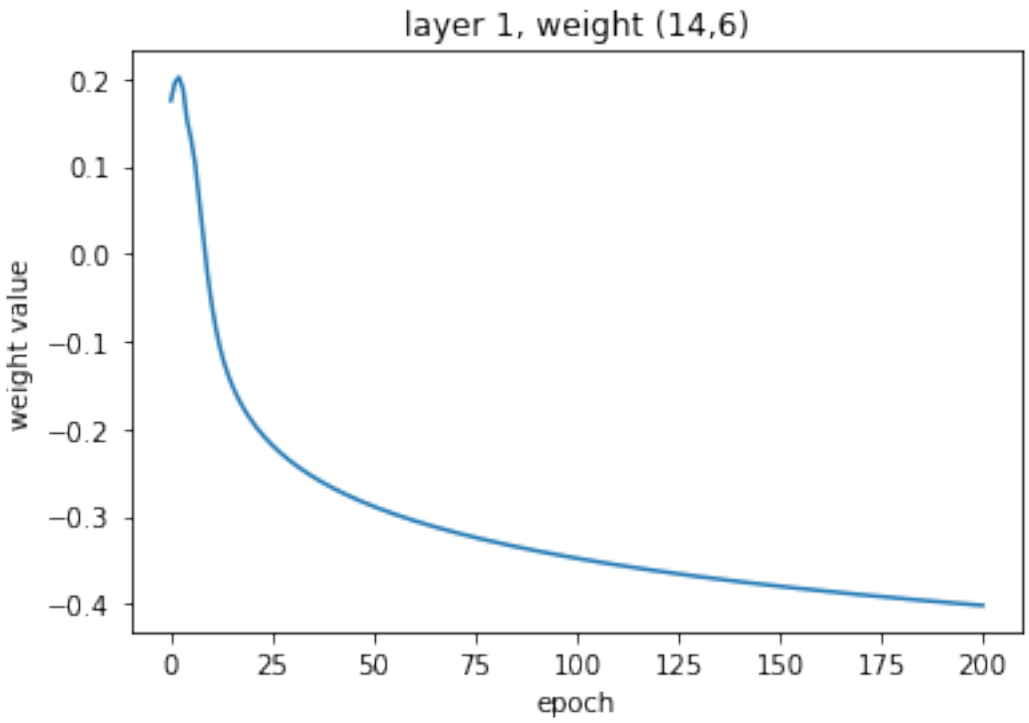
[36]:
```
model3 = Neural_Network()
model3.add_layer(10, 'sigmoid', input_shape=(4,5))
model3.add_layer(5, 'tanh')
```

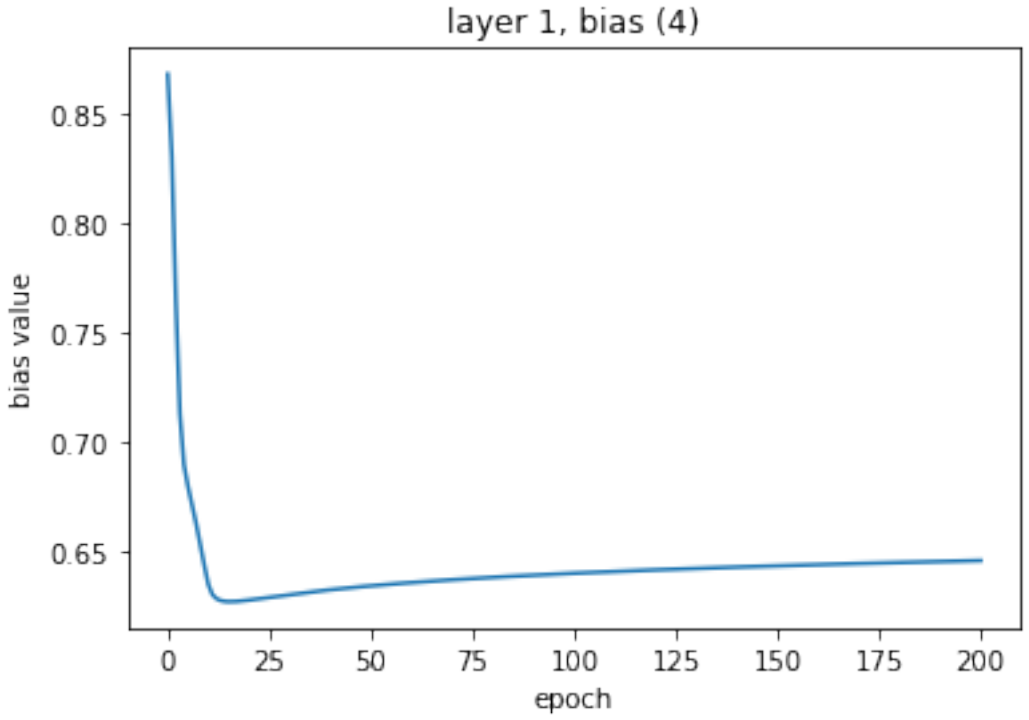
[37]:
```
fit(model3,epochs=250, alpha=0.001, dataset=training_set, targets=y_set, exit=0.
↪001)
```

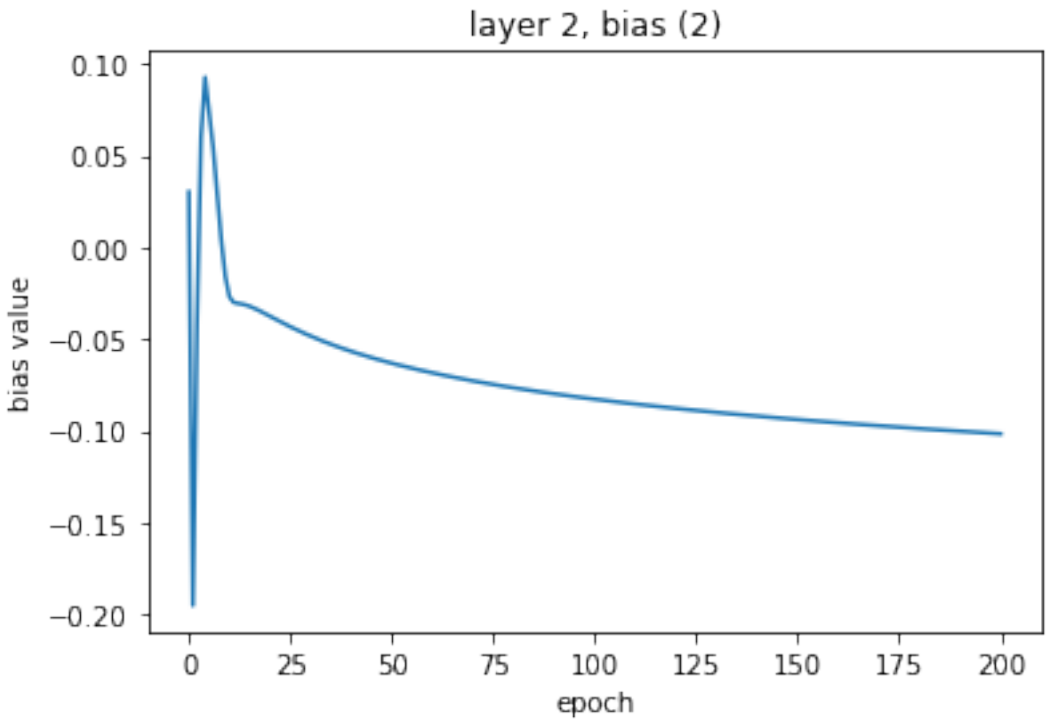



layer 1, weight (14,8)
weight value
0.050
0.025
0.000
−0.025
−0.050
−0.075
−0.100
−0.125
−0.150
0
25
50
75
100
125
150
175
200
epoch


layer 2, weight (1,2)
weight value
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
25
50
75
100
125
150
175
200
epoch


layer 1, weight (14,6)
weight value
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.4
0
25
50
75
100
125
150
175
200
epoch


layer 1, bias (4)
bias value
0.85
0.80
0.75
0.70
0.65
0
25
50
75
100
125
150
175
200
epoch




```
number of epochs to reach an mse of  200
```

Above we have tested three different alphas, 0.1, 0.01, and 0.001. I also use the MSE value to exit the training earlier as I believe an MSE less than 0.001 is enough training. This can always be change. From the results all the models were able to reach an MSE less than 0.001. But the 3rd model with an alpha=0.001 was able to reach the MSE threshold much quicker and we will be using it to test our test dataset.

We will start testing how the model will perform with the testing datasets. Now there are two ways I am deciding how to interpret the output values from the model. One is by using a threshold function. so if the output is greater than 0.32 it will be set to 1 otherwise -1. The other way is by making the biggest number in the outputs to 1 and set the rest to -1.

[38]:
```
correct = 0
for i in range(len(tset1)):
    sets = tset1[i]
    _y = y_set[i]
    for j in range(len(sets)):
        y = model3.threshold_predict(sets[i].flatten(), thresh=0.32)
        if np.array_equal(_y,y):
            correct += 1
        print(f'target = {_y}')
        print(f'     y = {y}')
        print('')
```

```
print(f'{(correct/25) * 100}% acc')
```

```
target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1. -1.  1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1. -1.  1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1. -1.  1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1. -1.  1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1. -1.  1. -1. -1.]
```

```
target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1.  1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1.  1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1.  1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1.  1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1.  1. -1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

100.0% acc
```

[39]:
```python
correct = 0
for i in range(len(tset2)):
    sets = tset2[i]
    _y = y_set[i]
    for j in range(len(sets)):
        y = model3.threshold_predict(sets[i].flatten(), thresh=0.32)
        if np.array_equal(_y,y):
            correct += 1
        print(f'target = {_y}')
        print(f'     y = {y}')
        print('')

print(f'{(correct/25) * 100}% acc')
```

```
target = [ 1. -1. -1. -1. -1.]
```

```
     y = [ 1. -1. -1. -1. -1.]

target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1. -1.  1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1. -1.  1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1. -1.  1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1. -1.  1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1. -1.  1. -1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1.  1. -1.]

target = [-1. -1. -1.  1. -1.]
```

```
     y = [-1. -1. -1.  1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1.  1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1.  1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1.  1. -1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

100.0% acc
```

[40]:
```
correct = 0
for i in range(len(tset3)):
    sets = tset3[i]
    _y = y_set[i]
    for j in range(len(sets)):
        y = model3.threshold_predict(sets[i].flatten(), thresh=0.32)
        if np.array_equal(_y,y):
            correct += 1
        print(f'target = {_y}')
        print(f'     y = {y}')
        print('')

print(f'{(correct/25) * 100}% acc')
```

```
target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]
```

```
target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1. -1.  1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1. -1.  1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1. -1.  1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1. -1.  1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1. -1.  1. -1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1.  1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1.  1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1.  1. -1.]
```

```
target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1.  1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1.  1. -1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

100.0% acc
```

[41]:
```python
correct = 0
for i in range(len(tset1)):
    sets = tset1[i]
    _y = y_set[i]
    for j in range(len(sets)):
        y = model3.max_arg_predict(sets[i].flatten())
        if np.array_equal(_y,y):
            correct += 1
        print(f'target = {_y}')
        print(f'     y = {y}')
        print('')

print(f'{(correct/25) * 100}% acc')
```

```
target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]
```

```
target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1. -1.  1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1. -1.  1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1. -1.  1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1. -1.  1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1. -1.  1. -1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1.  1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1.  1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1.  1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1.  1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1.  1. -1.]
```

```
target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

100.0% acc
```

[42]:
```
correct = 0
for i in range(len(tset2)):
    sets = tset2[i]
    _y = y_set[i]
    for j in range(len(sets)):
        y = model3.max_arg_predict(sets[i].flatten())
        if np.array_equal(_y,y):
            correct += 1
        print(f'target = {_y}')
        print(f'     y = {y}')
        print('')
print(f'{(correct/25) * 100}% acc')
```

```
target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]
```

```
target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1. -1.  1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1. -1.  1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1. -1.  1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1. -1.  1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1. -1.  1. -1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1.  1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1.  1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1.  1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1.  1. -1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]
```

```
target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

100.0% acc
```

[43]:
```
correct = 0
for i in range(len(tset3)):
    sets = tset3[i]
    _y = y_set[i]
    for j in range(len(sets)):
        y = model3.max_arg_predict(sets[i].flatten())
        if np.array_equal(_y,y):
            correct += 1
        print(f'target = {_y}')
        print(f'     y = {y}')
        print('')
print(f'{(correct/25) * 100}% acc')
```

```
target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]
```

```
target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1. -1.  1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1. -1.  1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1. -1.  1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1. -1.  1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1. -1.  1. -1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1.  1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1.  1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1.  1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1.  1. -1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]
```

```
target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

100.0% acc
```

As we can see from using our test set the model is doing well at classifying any test data we passed to it. It gets an accuracy of 100% for all test data that has some pixel distortion.

In this section we will be setting 20% of the of the weights in the model to 0 to see the impact of it, and see how well it still performs

[44]:
```
def destroy_weights(model, percent=0.2):
    for layers in range(len(model.weights)):
        n1 = model.weights[layers].shape[0]
        n2 = model.weights[layers].shape[1]
        twl = n1 * n2 # total weights in layer
        twl = int(twl* 0.2)
        destroyed = 0
        while(destroyed != twl):
            r1 = np.random.randint(low=0, high=n1)
            r2 = np.random.randint(low=0, high=n2)

            if model.weights[layers][r1][r2]!=0:
                model.weights[layers][r1][r2] = 0
                destroyed = destroyed + 1
            else:
                pass
```

[45]:
```
copy_model = copy.deepcopy(model3)
```

[46]:
```
copy_model.weights[0].shape
```

[46]:
```
(20, 10)
```

[47]:
```
destroy_weights(copy_model, percent=0.2)
```

[48]:
```
print(f"There are {np.count_nonzero(copy_model.weights[0]==0)} weights that are␣
 ↪zero from input to hidden layer")
print(f"There are {np.count_nonzero(copy_model.weights[1]==0)} weights that are␣
 ↪zero from hidden layer to output")
```

```
There are 40 weights that are zero from input to hidden layer
There are 10 weights that are zero from hidden layer to output
```

[49]:
```
correct = 0
for i in range(len(tset1)):
    sets = tset1[i]
    _y = y_set[i]
```

```
    for j in range(len(sets)):
        y = copy_model.threshold_predict(sets[i].flatten(), thresh=0.32)
        if np.array_equal(_y,y):
            correct += 1
        print(f'target = {_y}')
        print(f'     y = {y}')
        print('')

print(f'{(correct/25) * 100}% acc')
```

```
target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1. -1.  1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1. -1.  1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
```

```
     y = [-1. -1.  1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1. -1.  1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1. -1.  1. -1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1.  1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1.  1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1.  1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1.  1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1.  1. -1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

100.0% acc
```

[50]:
```
correct = 0
for i in range(len(tset2)):
    sets = tset2[i]
    _y = y_set[i]
    for j in range(len(sets)):
        y = copy_model.threshold_predict(sets[i].flatten(), thresh=0.32)
        if np.array_equal(_y,y):
```

```
            correct += 1
        print(f'target = {_y}')
        print(f'     y = {y}')
        print('')

print(f'{(correct/25) * 100}% acc')
```

```
target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
```

```
     y = [-1.  1. -1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1.  1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1.  1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1.  1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1.  1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1.  1. -1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

80.0% acc
```

[51]:
```python
correct = 0
for i in range(len(tset3)):
    sets = tset3[i]
    _y = y_set[i]
    for j in range(len(sets)):
        y = copy_model.threshold_predict(sets[i].flatten(), thresh=0.32)
        if np.array_equal(_y,y):
            correct += 1
        print(f'target = {_y}')
        print(f'     y = {y}')
```

```
        print('')

print(f'{(correct/25) * 100}% acc')
```

```
target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
```

```
     y = [-1.  1. -1. -1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1. -1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1. -1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1. -1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1. -1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1. -1. -1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

60.0% acc
```

[52]:
```
correct = 0
for i in range(len(tset1)):
    sets = tset1[i]
    _y = y_set[i]
    for j in range(len(sets)):
        y = copy_model.max_arg_predict(sets[i].flatten())
        if np.array_equal(_y,y):
            correct += 1
        print(f'target = {_y}')
        print(f'     y = {y}')
        print('')
print(f'{(correct/25) * 100}% acc')
```

```
target = [ 1. -1. -1. -1. -1.]
```

```
     y = [ 1. -1. -1. -1. -1.]

target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1. -1.  1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1. -1.  1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1. -1.  1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1. -1.  1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1. -1.  1. -1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1.  1. -1.]

target = [-1. -1. -1.  1. -1.]
```

```
     y = [-1. -1. -1.  1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1.  1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1.  1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1.  1. -1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

100.0% acc
```

[53]:
```
correct = 0
for i in range(len(tset2)):
    sets = tset2[i]
    _y = y_set[i]
    for j in range(len(sets)):
        y = copy_model.max_arg_predict(sets[i].flatten())
        if np.array_equal(_y,y):
            correct += 1
        print(f'target = {_y}')
        print(f'     y = {y}')
        print('')
print(f'{(correct/25) * 100}% acc')
```

```
target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [ 1. -1. -1. -1. -1.]
```

```
     y = [ 1. -1. -1. -1. -1.]

target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1.  1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1.  1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1.  1. -1.]

target = [-1. -1. -1.  1. -1.]
```

```
     y = [-1. -1. -1.  1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1.  1. -1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

80.0% acc
```

[54]:
```
correct = 0
for i in range(len(tset3)):
    sets = tset3[i]
    _y = y_set[i]
    for j in range(len(sets)):
        y = copy_model.max_arg_predict(sets[i].flatten())
        if np.array_equal(_y,y):
            correct += 1
        print(f'target = {_y}')
        print(f'     y = {y}')
        print('')
print(f'{(correct/25) * 100}% acc')
```

```
target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [ 1. -1. -1. -1. -1.]
```

```
     y = [ 1. -1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1.  1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1.  1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1.  1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1.  1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1.  1. -1.]

target = [-1. -1. -1. -1.  1.]
```

```
     y = [-1. -1. -1. -1.  1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

80.0% acc
```

1.2.4 Results 1

When setting 20% of the weights in the model to 0 it affected the accuracy of the testing dataset. When using the threshold method we see that it can classify the tset1 with 100% accuracy but when testing tset2 and tset3 we get 80% and 60% respectively.

When doing the biggest number method we see that the tset1 gets 100% accuracy while tset2 and tset3 gets 80% accuracy.

This is most likely due to the fact that the weights that were set to 0 were vital in the way on deciding what class its suppose to be.

setting 20% of the weights to zero. Basically 40% of the weights will be set to zero.

[55]:
```
destroy_weights(copy_model, percent=0.2)
```

[56]:
```
print(f"There are {np.count_nonzero(copy_model.weights[0]==0)} weights that are␣
 ↪zero from input to hidden layer")
print(f"There are {np.count_nonzero(copy_model.weights[1]==0)} weights that are␣
 ↪zero from hidden layer to output")
```

```
There are 80 weights that are zero from input to hidden layer
There are 20 weights that are zero from hidden layer to output
```

[57]:
```
correct = 0
for i in range(len(tset1)):
    sets = tset1[i]
    _y = y_set[i]
    for j in range(len(sets)):
        y = copy_model.threshold_predict(sets[i].flatten(), thresh=0.32)
        if np.array_equal(_y,y):
            correct += 1
        print(f'target = {_y}')
```

```
        print(f'     y = {y}')
        print('')

print(f'{(correct/25) * 100}% acc')
```

```
target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1. -1. -1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1. -1. -1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1. -1. -1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1. -1. -1. -1. -1.]
```

```
target = [-1. -1.  1. -1. -1.]
     y = [-1. -1. -1. -1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1.  1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1.  1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1.  1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1.  1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1.  1. -1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

80.0% acc
```

[58]:
```python
correct = 0
for i in range(len(tset2)):
    sets = tset2[i]
    _y = y_set[i]
    for j in range(len(sets)):
        y = copy_model.threshold_predict(sets[i].flatten(), thresh=0.32)
        if np.array_equal(_y,y):
            correct += 1
        print(f'target = {_y}')
        print(f'     y = {y}')
        print('')
```

```
print(f'{(correct/25) * 100}% acc')
```

```
target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1. -1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1. -1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1. -1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1. -1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1. -1. -1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]
```

```
target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1.  1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1.  1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1.  1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1.  1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1.  1. -1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

60.0% acc
```

[59]:
```python
correct = 0
for i in range(len(tset3)):
    sets = tset3[i]
    _y = y_set[i]
    for j in range(len(sets)):
        y = copy_model.threshold_predict(sets[i].flatten(), thresh=0.32)
        if np.array_equal(_y,y):
            correct += 1
        print(f'target = {_y}')
        print(f'     y = {y}')
        print('')

print(f'{(correct/25) * 100}% acc')
```

```
target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]
```

```
target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1. -1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1. -1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1. -1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1. -1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1. -1. -1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1. -1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1. -1. -1.]
```

```
target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1. -1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1. -1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1. -1. -1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

40.0% acc
```

[60]:
```
correct = 0
for i in range(len(tset1)):
    sets = tset1[i]
    _y = y_set[i]
    for j in range(len(sets)):
        y = copy_model.max_arg_predict(sets[i].flatten())
        if np.array_equal(_y,y):
            correct += 1
        print(f'target = {_y}')
        print(f'     y = {y}')
        print('')
print(f'{(correct/25) * 100}% acc')
```

```
target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]
```

```
target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1. -1.  1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1. -1.  1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1. -1.  1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1. -1.  1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1. -1.  1. -1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1.  1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1.  1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1.  1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1.  1. -1.]
```

```
target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1.  1. -1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

100.0% acc
```

[61]:
```
correct = 0
for i in range(len(tset2)):
    sets = tset2[i]
    _y = y_set[i]
    for j in range(len(sets)):
        y = copy_model.max_arg_predict(sets[i].flatten())
        if np.array_equal(_y,y):
            correct += 1
        print(f'target = {_y}')
        print(f'     y = {y}')
        print('')
print(f'{(correct/25) * 100}% acc')
```

```
target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]
```

```
target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1.  1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1.  1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1.  1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1.  1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1.  1. -1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]
```

```
target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

80.0% acc
```

[62]:
```
correct = 0
for i in range(len(tset3)):
    sets = tset3[i]
    _y = y_set[i]
    for j in range(len(sets)):
        y = copy_model.max_arg_predict(sets[i].flatten())
        if np.array_equal(_y,y):
            correct += 1
        print(f'target = {_y}')
        print(f'     y = {y}')
        print('')
print(f'{(correct/25) * 100}% acc')
```

```
target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [ 1. -1. -1. -1. -1.]
     y = [ 1. -1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]
```

```
target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1.  1. -1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1. -1.  1. -1. -1.]
     y = [-1.  1. -1. -1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1.  1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1.  1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1.  1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1.  1. -1.]

target = [-1. -1. -1.  1. -1.]
     y = [-1. -1. -1.  1. -1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]
```

```
target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

target = [-1. -1. -1. -1.  1.]
     y = [-1. -1. -1. -1.  1.]

80.0% acc
```

1.2.5 Results 2

When setting 20% of the weights in the model to 0 it affected the accuracy of the testing dataset. When using the threshold method we see that it can classify the tset1 with 80% accuracy but when testing tset2 and tset3 we get 60% and 40% respectively. This shows that we have heavily compromise the models when setting 40% of the weights to 0. Although it can handle a 1 bit pixel error when adding 2 or 3 pixel error it will struggle

When doing the biggest number method we see that the tset1 gets 100% accuracy while tset2 and tset3 gets 80% accuracy. This is very interesting as with this method the model did not suffer at all compare to the threshold method. This shows that depending on how you interpret the output out the end can influence the accuracy of the model.