MLP

December 1, 2020

Thony Yan PID:3913880

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import copy
```

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.

[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)
```

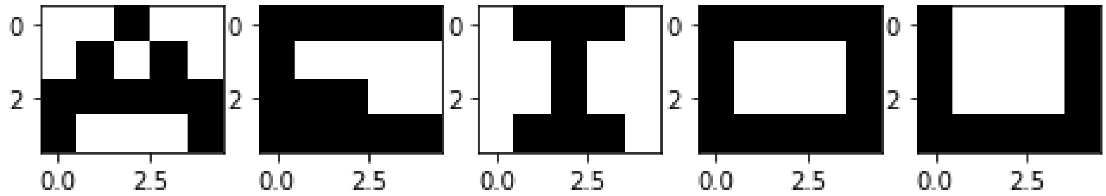
[4]:
```
p1.shape
```

[4]: (4, 5)

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

[6]:
```python
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')
```

[7]:
```python
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')
```



[8]:
```python
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]])
```

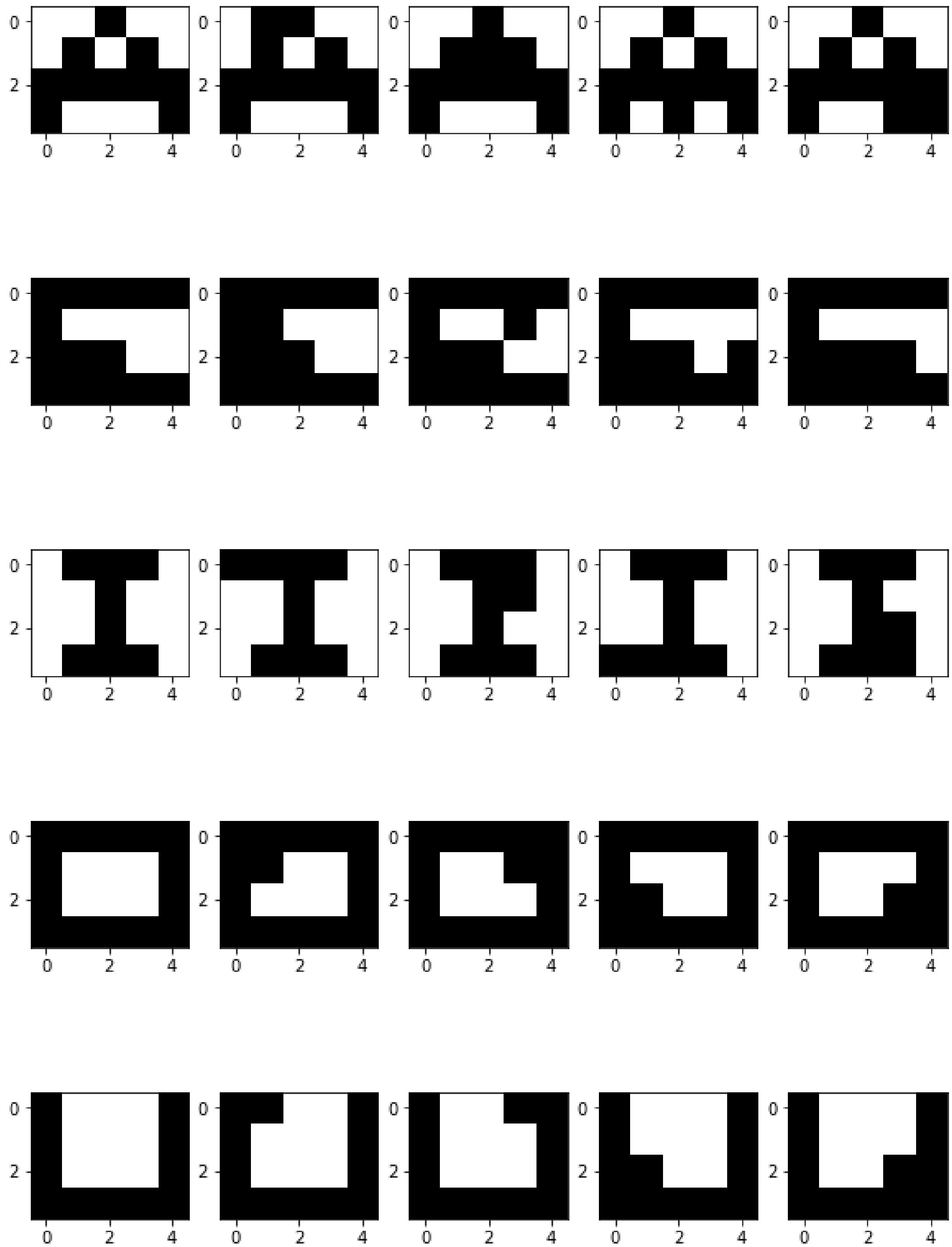
[9]:
```python
mod = [a_mod,e_mod,i_mod,o_mod,u_mod]
```

[10]:
```python
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]:
```python
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]:
```python
show(a_test)
show(e_test)
show(i_test)
```

```
show(o_test)
show(u_test)
```

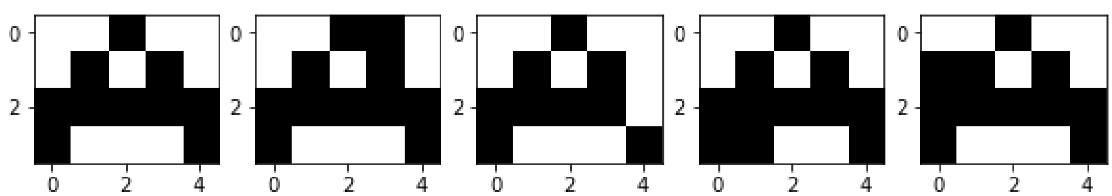


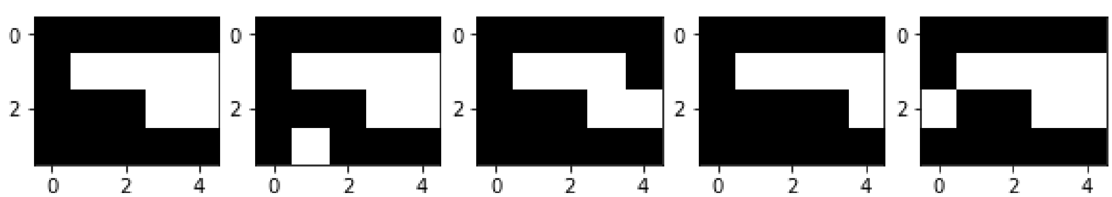
[13]:
```python
test_set = np.array([a_test,e_test,i_test,o_test,u_test])
```

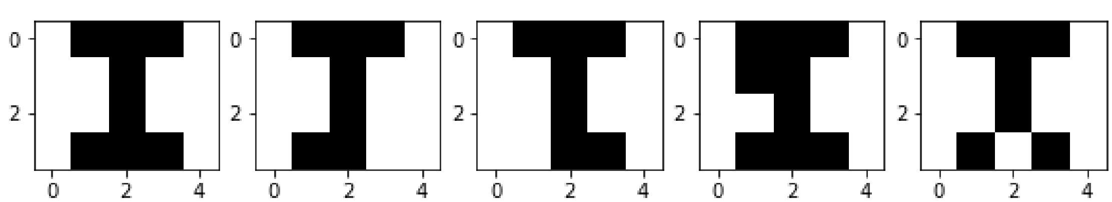
[14]:
```python
tset_mod1 = np.array([[ [4,1],[5,3],[2,4],[1,2] ],  #tset1 a
                      [ [2,4],[5,2],[4,3],[1,3] ],  #tset1 e
                      [ [4,4],[2,4],[2,2],[3,4] ],  #tset1 i
                      [ [1,3],[3,2],[5,4],[3,3] ],  #tset1 o
                      [ [5,3],[3,4],[1,4],[5,4]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

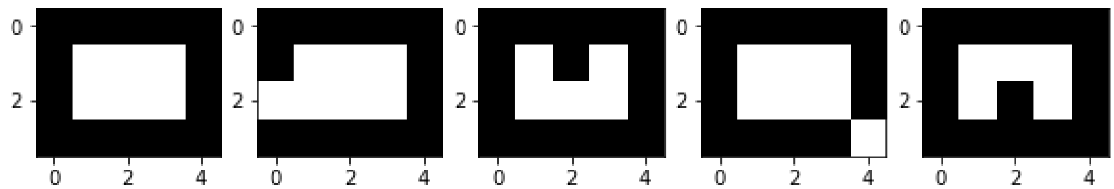
[15]:
```python
tset1_a = modify(base[0], tset_mod1[0])
tset1_e = modify(base[1], tset_mod1[1])
tset1_i = modify(base[2], tset_mod1[2])
tset1_o = modify(base[3], tset_mod1[3])
tset1_u = modify(base[4], tset_mod1[4])
```

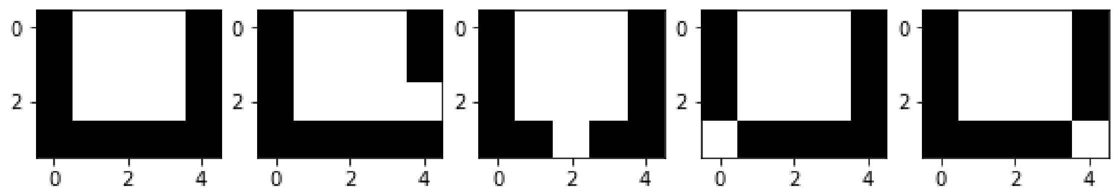
[16]:
```python
show(tset1_a)
show(tset1_e)
show(tset1_i)
show(tset1_o)
show(tset1_u)
```











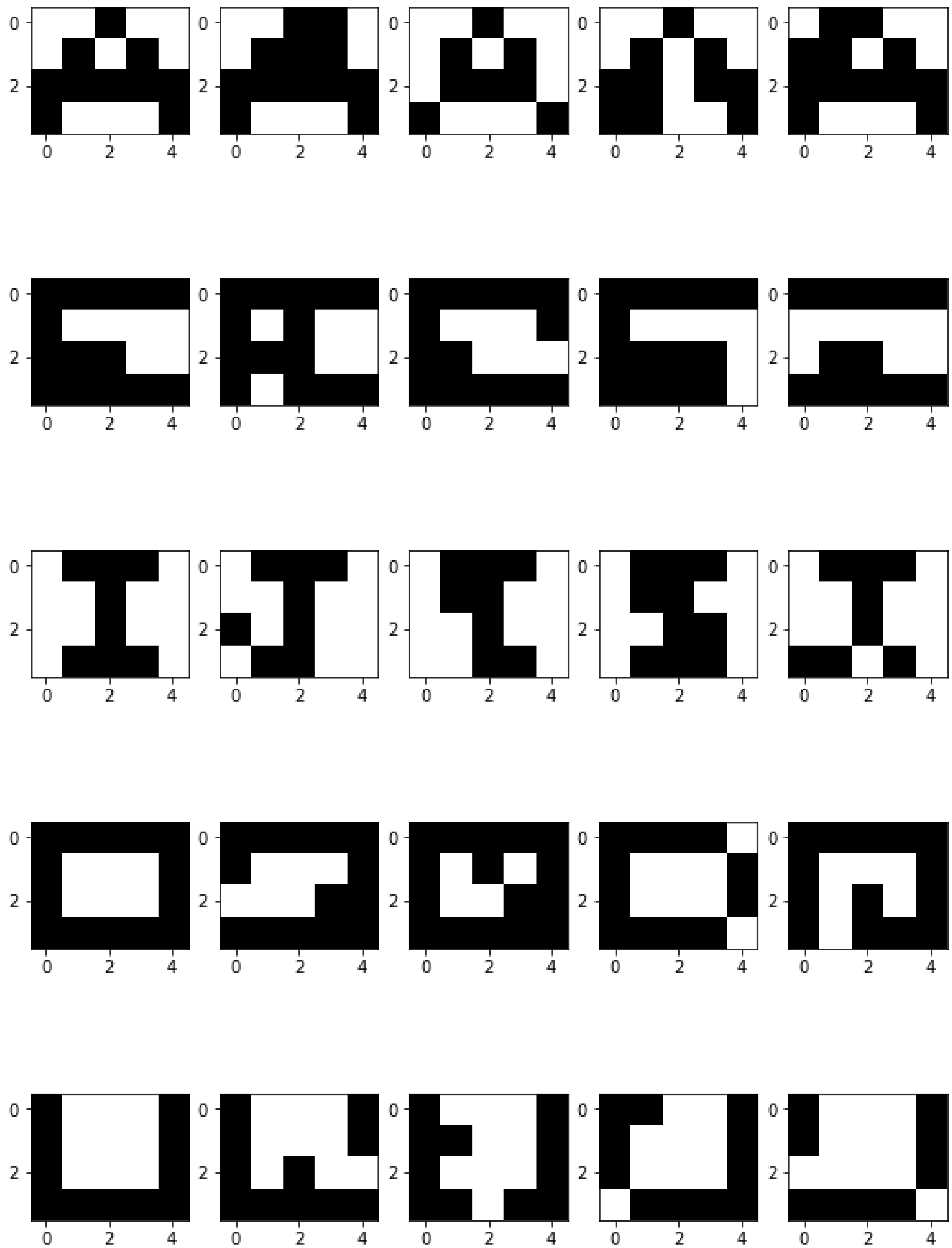
[17]:
```
tset1 = np.array([tset1_a,tset1_e,tset1_i,tset1_o,tset1_u])
```

[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)
```

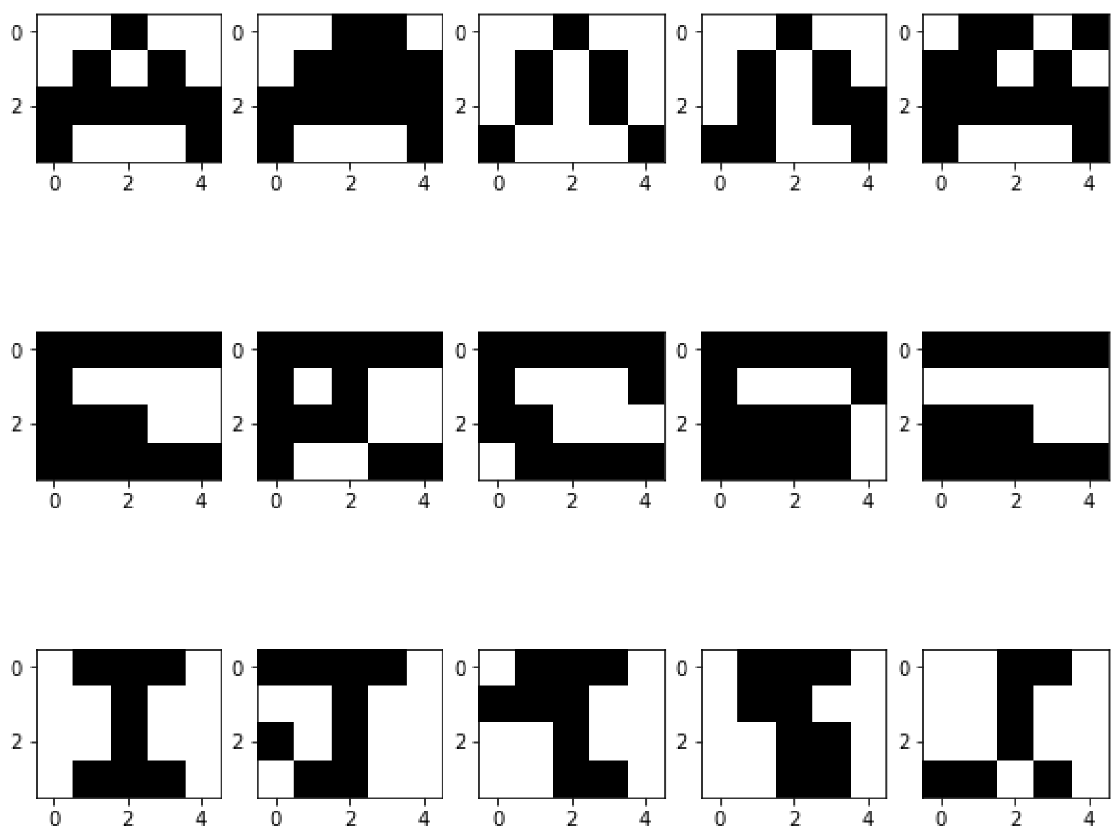


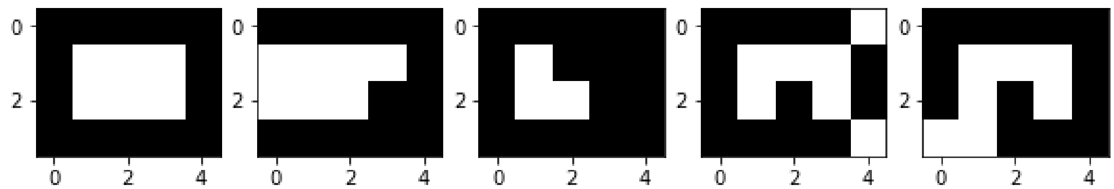
[22]:
```
tset2 = np.array([tset2_a,tset2_e,tset2_i,tset2_o,tset2_u])
```

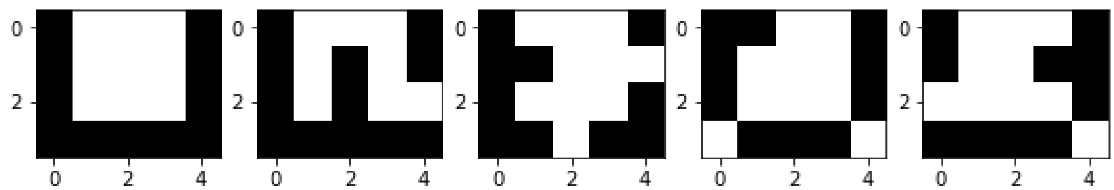
[23]:
```
tset_mod3 = np.array([[ [5,2],[3,3],[1,3],[5,1] ], #tset3 a
                      [ [3,4],[1,4],[5,2],[1,3] ], #tset3 e
                      [ [1,1],[1,2],[2,4],[2,1] ], #tset3 i
                      [ [1,2],[4,2],[3,3],[1,4] ], #tset3 o
                      [ [3,2],[5,2],[5,4],[4,2] ]])#tset3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

[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
```

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

    @staticmethod
    def hardlim(output, tresh=0.35):

        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 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

        print(self.max_arg(a))
```

[28]:
```
y = np.array([1.,-1.,-1.,-1.,-1.])
inputs = np.array(a_test[0].flatten(), dtype=np.float)
inputs
```

[28]:
```
array([-1., -1.,  1., -1., -1., -1.,  1., -1.,  1., -1.,  1.,  1.,  1.,
        1.,  1.,  1., -1., -1., -1.,  1.])
```

[29]:
```
model = Neural_Network()
```

[30]:
```
model.add_layer(10, 'sigmoid', input_shape=(4,5))
model.add_layer(5, 'tanh')
```

[31]:
```
print(model.weights[0].shape)
print(model.weights[1].shape)
print(model.bias[0].shape)
print(model.bias[1].shape)
```

```
(20, 10)
(10, 5)
(10,)
(5,)
```

[32]:
```
model.feedforward(inputs)
```

[33]:
```
model.a_val
```

[33]:
```
[array([-1., -1.,  1., -1., -1., -1.,  1., -1.,  1., -1.,  1.,  1.,  1.,
         1.,  1.,  1., -1., -1., -1.,  1.]),
 array([0.64644493, 0.77080004, 0.71885806, 0.47398656, 0.49598641,
        0.68281305, 0.80415187, 0.70227379, 0.49979034, 0.50440735]),
 array([0.82932298, 0.71183437, 0.49825489, 0.8272358 , 0.92051989])]
```

[34]:
```
model.z_val
```

[34]:
```
[array([-1., -1.,  1., -1., -1., -1.,  1., -1.,  1., -1.,  1.,  1.,  1.,
         1.,  1.,  1., -1., -1., -1.,  1.]),
 array([ 6.03448760e-01,  1.21283419e+00,  9.38804238e-01, -1.04147782e-01,
        -1.60547223e-02,  7.66729747e-01,  1.41244865e+00,  8.58149054e-01,
        -8.38620464e-04,  1.76298564e-02]),
 array([1.1859641 , 0.89089269, 0.54698203, 1.17931596, 1.59242219])]
```

[35]:
```
error = model.error(y, model.a_val[-1])
error
```

[35]:
```
array([ 0.17067702, -1.71183437, -1.49825489, -1.8272358 , -1.92051989])
```

[36]:
```
s = model.activation[1](model.z_val[2], derivative=True) * error
st = s.reshape(s.shape[0],-1)
st
```

[36]:
```
array([[ 0.05328936],
       [-0.84443391],
       [-1.12630123],
       [-0.57682351],
       [-0.29315418]])
```

[37]:
```
a = model.a_val[1]
a = a.reshape(a.shape[0],-1)
a
```

[37]:
```
array([[0.64644493],
       [0.77080004],
       [0.71885806],
       [0.47398656],
```

```
        [0.49598641],
        [0.68281305],
        [0.80415187],
        [0.70227379],
        [0.49979034],
        [0.50440735]])
```

[38]:
```
Wnew = np.dot(st,a.T)
```

[39]:
```
Wnew.T.shape
```

[39]:
```
(10, 5)
```

[40]:
```
model.weights[1].shape
```

[40]:
```
(10, 5)
```

[41]:
```
model.weights[1]
```

[41]:
```
array([[0.11950973, 0.18322987, 0.16974096, 0.24460128, 0.00776375],
       [0.24393673, 0.24799534, 0.0645592 , 0.07194402, 0.22945671],
       [0.14866324, 0.01806038, 0.08104931, 0.21332171, 0.13551904],
       [0.20450198, 0.23894001, 0.13669623, 0.10112259, 0.22700083],
       [0.04870516, 0.1889276 , 0.18740806, 0.19722554, 0.13696383],
       [0.11336842, 0.08430546, 0.07300146, 0.16405793, 0.20022777],
       [0.07881726, 0.09847161, 0.00097491, 0.21052535, 0.23203238],
       [0.04374802, 0.13588868, 0.05316806, 0.16723089, 0.12771972],
       [0.17482176, 0.17594354, 0.01899663, 0.0926024 , 0.18774543],
       [0.05607704, 0.00191615, 0.03879513, 0.0239213 , 0.15089376]])
```

[42]:
```
model.back_propagate(error)
```

[43]:
```
model.sensitivity[1].shape
```

[43]:
```
(5,)
```

[44]:
```
model.bias[1].shape
```

[44]:
```
(5,)
```

[45]:
```
model.weights[0][14]
```

[45]:
```
array([0.12231431, 0.22553703, 0.11454857, 0.10709523, 0.15177883,
       0.02874252, 0.17559411, 0.11504398, 0.02185554, 0.11890129])
```

[46]:
```
model.update_weights()
```

[47]:
```
print(model.weights[0].shape)
print(model.weights[1].shape)
print(model.bias[0].shape)
print(model.bias[1].shape)
```

```
(20, 10)
(10, 5)
(10,)
(5,)
```

[48]:
```
a = model.a_val
t = np.dot(inputs, model.weights[0])
t
```

[48]:
```
array([-0.44441062, -0.01526231, -0.19635479, -0.97025093, -0.65137729,
       -0.35530978,  0.42253502, -0.02501467, -0.48456651, -0.14728095])
```

[49]:
```
model = Neural_Network()
model.add_layer(10, 'sigmoid', input_shape=(4,5))
model.add_layer(5, 'tanh')
```

[50]:
```
training_set = a_test, e_test, i_test, o_test, u_test
training_set
```

[50]:
```
([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]])])
```

[51]:
```
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]])
```

[52]:
```
epoch = 100
total_mse = []
random_w1 = []
random_w2 = []
random_w3 = []
bias1 = []
bias2 = []
random1 = np.random.randint(20)
random2 = np.random.randint(10)
random3 = np.random.randint(10)
random4 = np.random.randint(5)
bias_random1 = np.random.randint(10)
bias_random2 = np.random.randint(5)
```

```
for i in range(epoch):
    mse = 0
    random_w1.append(model.weights[0][random1][random2])
    random_w2.append(model.weights[1][random3][random4])
    bias1.append(model.bias[0][bias_random1])
    bias2.append(model.bias[1][bias_random2])

    for i in range(len(training_set)):
        set = training_set[i]
        target = y_set[i]
        for j in range(len(set)):
            #show(set)
            #print(target)
            mse += model.train(set[i].flatten(), target, 0.1)

    mse = (mse / 25.)
    total_mse.append(mse)
```

[53]:

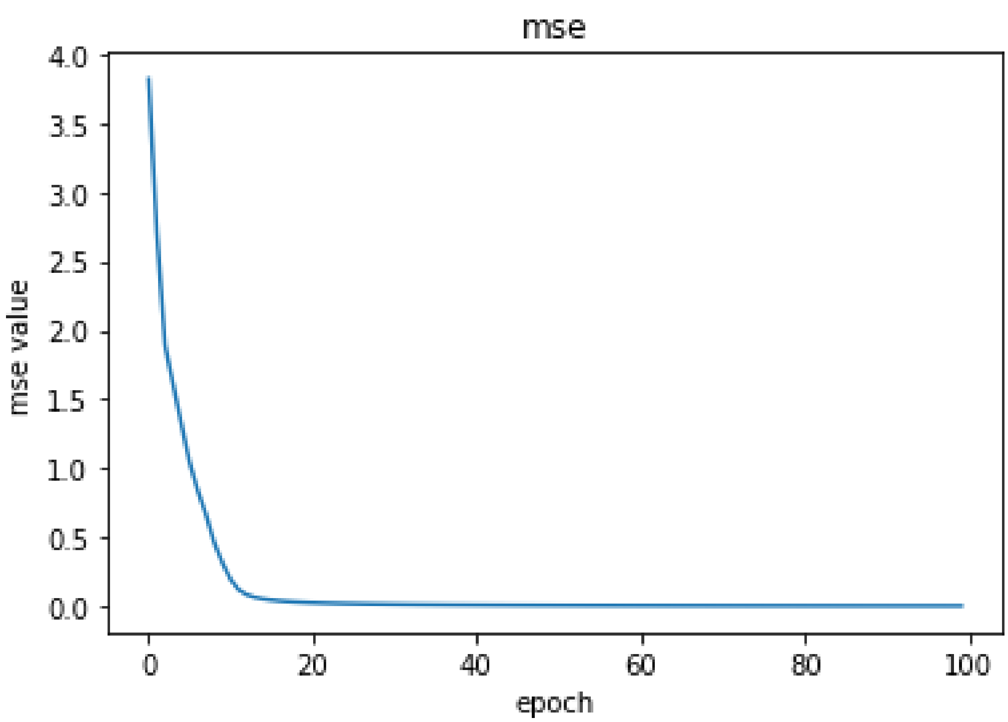
```
for i in range(len(training_set)):
    set = training_set[i]
    for j in range(len(set)):
        model.predict(set[i].flatten())

    print('')
```

```
[ 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.]

[-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.]

[-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.]

[-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.]

[-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.]
```

[54]:
```
plt.plot(total_mse)
plt.xlabel('epoch')
plt.ylabel('mse value')
plt.title('mse')
```

[54]: `Text(0.5, 1.0, 'mse')`



[55]:
```
total_mse
```

[55]:
```
[3.8211213883162003,
 2.7071518334550024,
 1.8893904706959035,
 1.6039784667644765,
 1.3175539621239252,
```

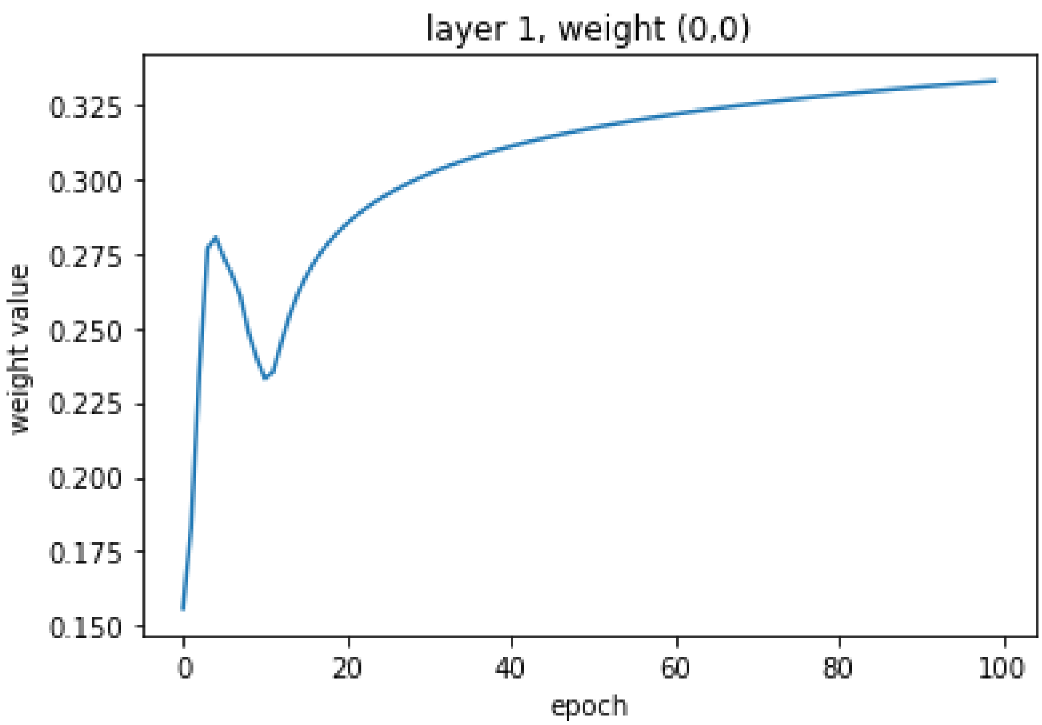
```
1.0415226763511125,
0.8362625761041579,
0.6625045051351494,
0.4596836361835347,
0.3139281848563958,
0.19321463459002722,
0.11871550781376981,
0.0829245109363495,
0.0644069842837835,
0.052924155867083794,
0.04503251774196979,
0.039236653034676994,
0.03478137661027706,
0.03124119146053134,
0.02835632404623788,
0.025958233578812995,
0.02393230490262448,
0.02219763855100262,
0.02069538316346409,
0.019381642212162903,
0.018222974953682556,
0.01719344038586795,
0.01627259625661048,
0.01544410949799902,
0.014694769600787446,
0.01401377430197083,
0.013392203418477213,
0.012822625246174746,
0.012298798011964339,
0.011815440563446283,
0.011368054215989014,
0.010952782891921347,
0.010566302264046248,
0.010205731109108378,
0.009868559840141499,
0.00955259245032961,
0.009255899017901428,
0.00897677659445303,
0.00871371679815177,
0.008465378807085222,
0.008230566730605689,
0.008008210551996277,
0.0077973500014224826,
0.007597120846432854,
0.0074067431873669895,
0.007225511423648126,
0.007052785619087886,
```

```
0.006887984043758454,
0.006730576709525491,
0.006580079748135459,
0.006436050506460667,
0.0062980832543945466,
0.0061658054179430846,
0.006038874264041949,
0.005916973975146645,
0.005799813061167118,
0.005687122064226569,
0.005578651518314888,
0.005474170131419882,
0.00537346316234696,
0.005276330968335102,
0.005182587702869972,
0.005092060145885884,
0.005004586650920946,
0.004920016195812674,
0.004838207525250477,
0.004759028374984271,
0.004682354768762483,
0.004608070380170594,
0.004536065952489847,
0.004466238770516563,
0.004398492178995108,
0.004332735142936956,
0.0042688818456384885,
0.004206851320681442,
0.004146567114612924,
0.004087956977363366,
0.004030952577779046,
0.003975489241925093,
0.003921505712061925,
0.0038689439244155147,
0.003817748804054935,
0.003767868075361048,
0.0037192520867219926,
0.003671853648225785,
0.003625627881240201,
0.0035805320788772437,
0.0035365255764347476,
0.003493569630993294,
0.0034516273094228876,
0.003410663384122512,
0.0033706442358770795,
0.0033315377632717907,
0.003293313298153558,
```

0.003255941526674066]

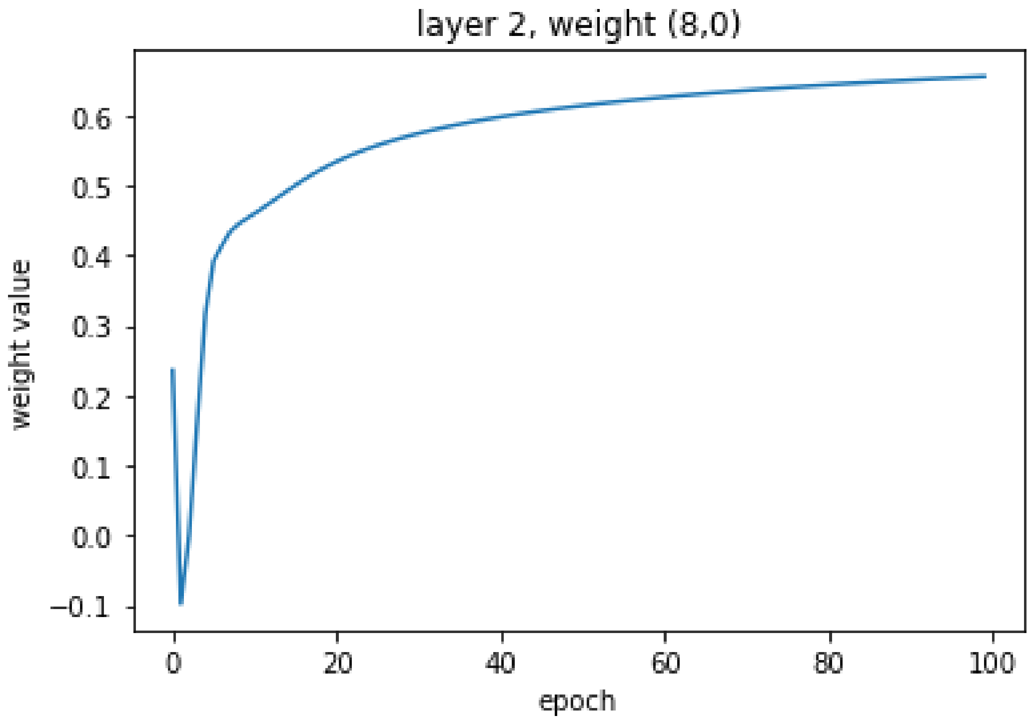
[56]:
```
plt.plot(random_w1)
plt.xlabel('epoch')
plt.ylabel('weight value')
plt.title('layer 1, weight (' + str(random1) + ',' + str(random2) + ')')
```

[56]: Text(0.5, 1.0, 'layer 1, weight (0,0)')



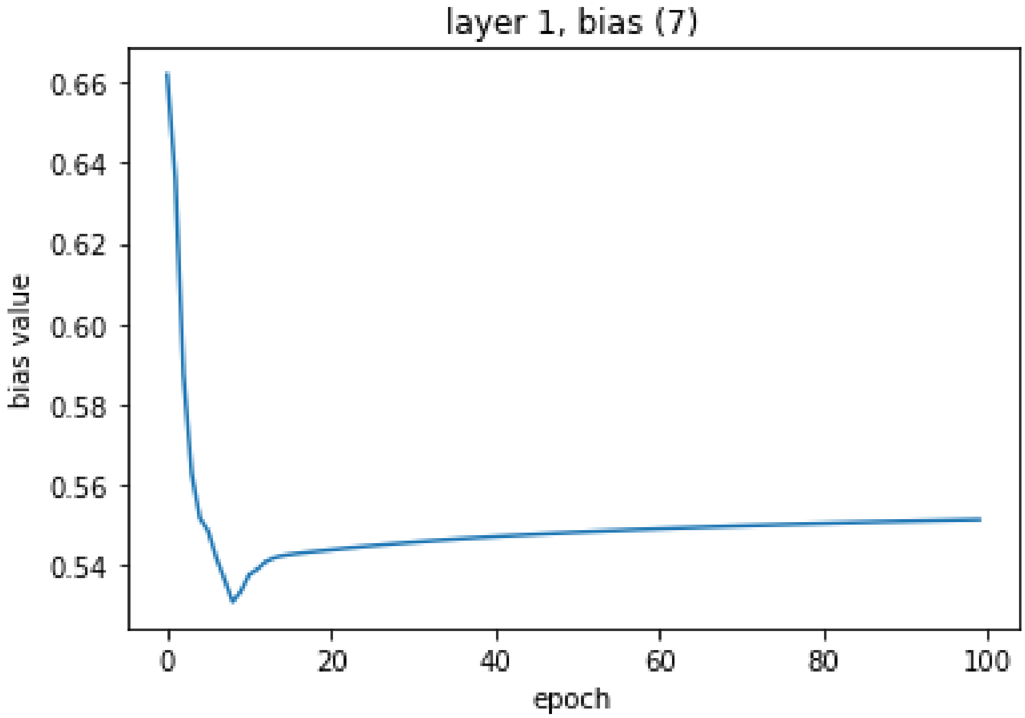
[57]:
```
plt.plot(random_w2)
plt.xlabel('epoch')
plt.ylabel('weight value')
plt.title('layer 2, weight (' + str(random3) + ',' + str(random4) + ')')
```

[57]: Text(0.5, 1.0, 'layer 2, weight (8,0)')



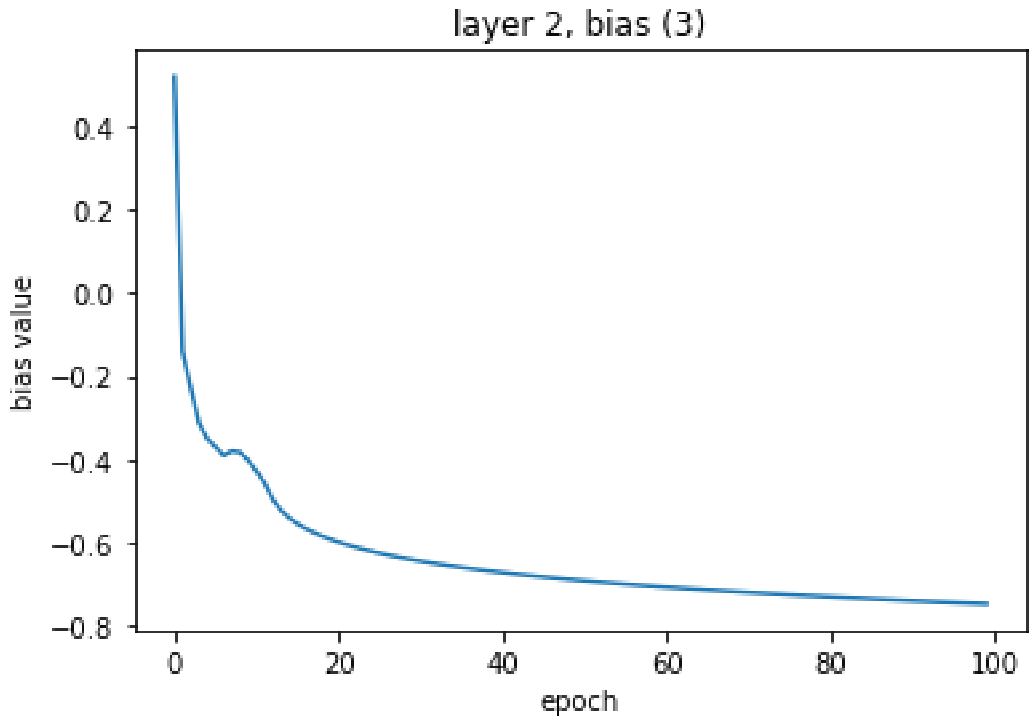
[58]:
```
plt.plot(bias1)
plt.xlabel('epoch')
plt.ylabel('bias value')
plt.title('layer 1, bias (' + str(bias_random1) + ')')
```

[58]: Text(0.5, 1.0, 'layer 1, bias (7)')



[59]:
```
plt.plot(bias2)
plt.xlabel('epoch')
plt.ylabel('bias value')
plt.title('layer 2, bias (' + str(bias_random2) + ')')
```

[59]:
```
Text(0.5, 1.0, 'layer 2, bias (3)')
```



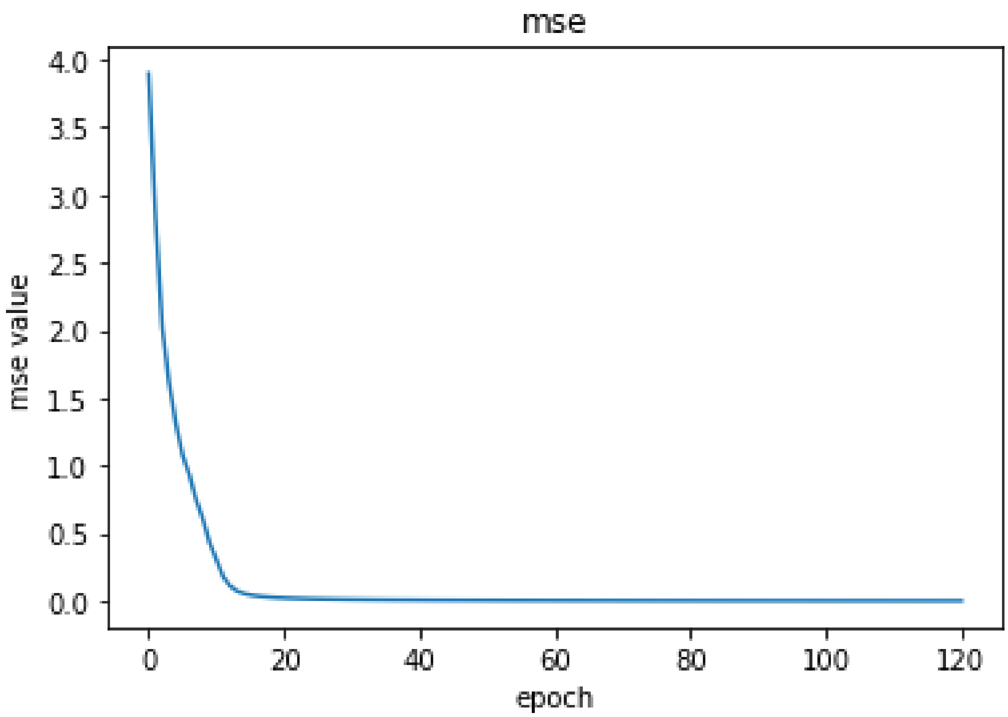
[60]:
```
def fit(model,epochs, alpha, dataset, targets, exit=0.01, input_l=20,␣
↪hidden_l=10, output_l=5):
    total_mse = []
    random_w1 = []
    random_w2 = []
    random_w3 = []
    bias1 = []
    bias2 = []
    random1 = np.random.randint(input_l)
    random2 = np.random.randint(hidden_l)
    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])
```

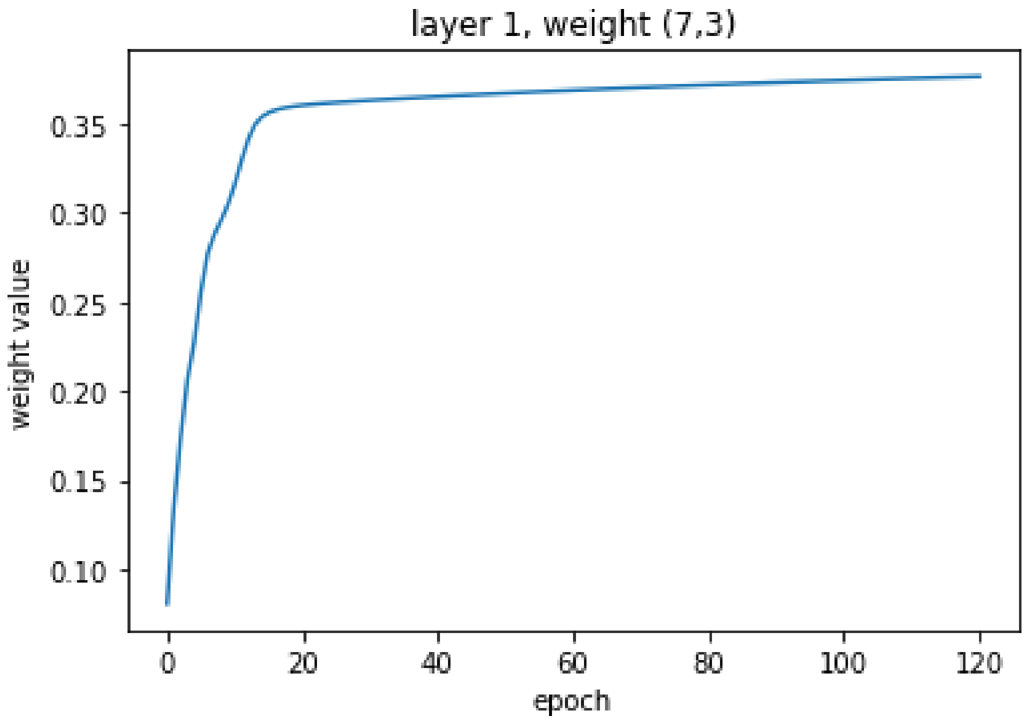
```python
        for i in range(len(dataset)):
            sets = dataset[i]
            target = targets[i]
            for j in range(len(sets)):
                mse += model.train(sets[i].flatten(), target, 0.1)

        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()
```

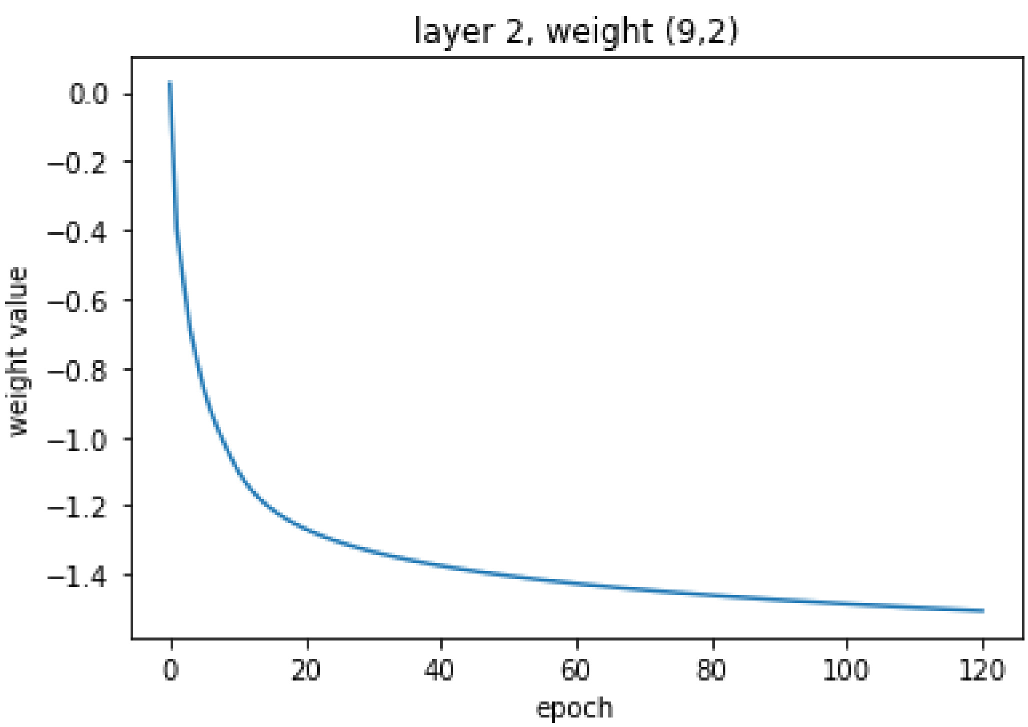
```
    print('\n number of epochs to reach an mse of ' , epoch)
```

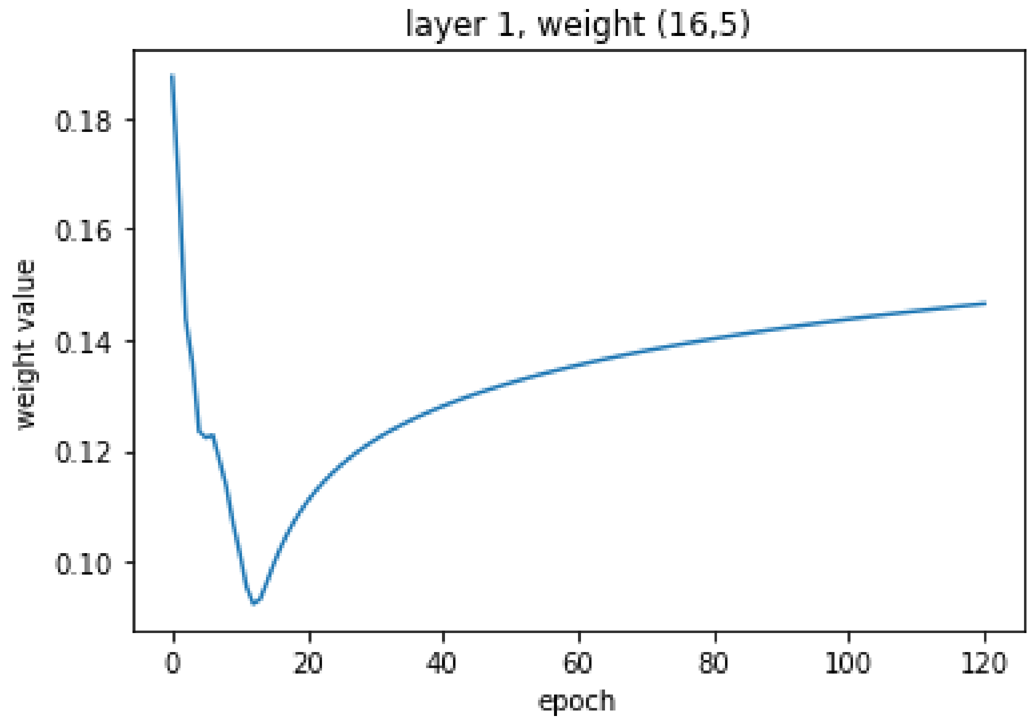
[61]:
```
model1 = Neural_Network()
model1.add_layer(10, 'sigmoid', input_shape=(4,5))
model1.add_layer(5, 'tanh')
```

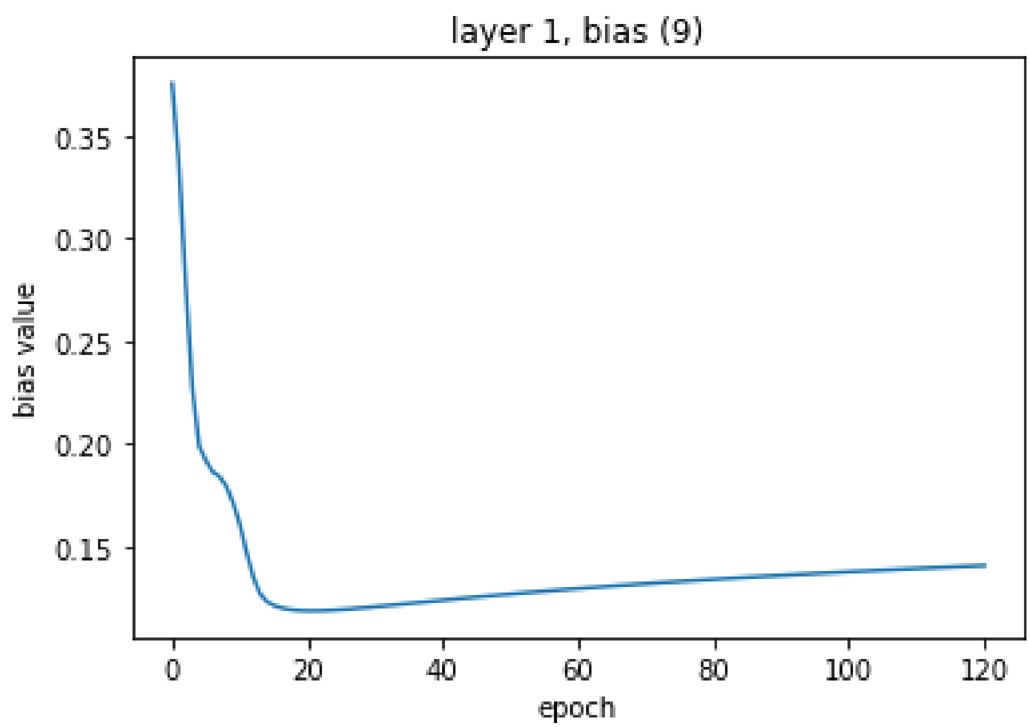
[62]:
```
fit(model1,epochs=500, alpha=0.1, dataset=training_set, targets=y_set, exit=0.
 ↪002)
```

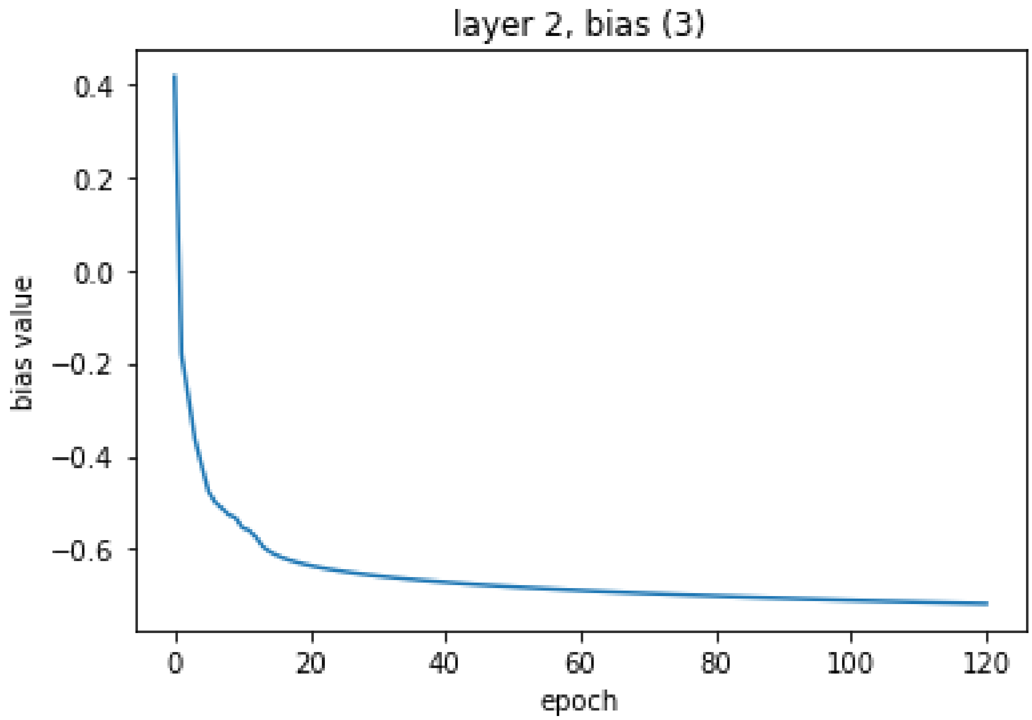



layer 1, weight (7,3)
weight value
0.10
0.15
0.20
0.25
0.30
0.35
0
20
40
60
80
100
120
epoch


layer 2, weight (9,2)
weight value
0.0
−0.2
−0.4
−0.6
−0.8
−1.0
−1.2
−1.4
0
20
40
60
80
100
120
epoch


layer 1, weight (16,5)
weight value
0.18
0.16
0.14
0.12
0.10
0
20
40
60
80
100
120
epoch

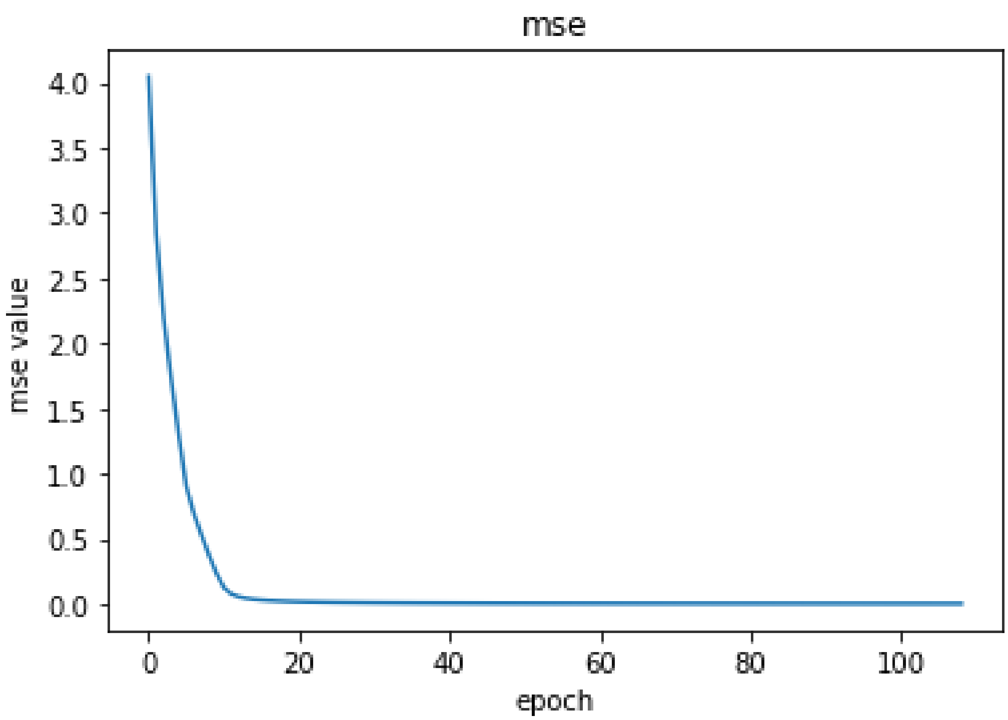

layer 1, bias (9)
bias value
0.35
0.30
0.25
0.20
0.15
0
20
40
60
80
100
120
epoch

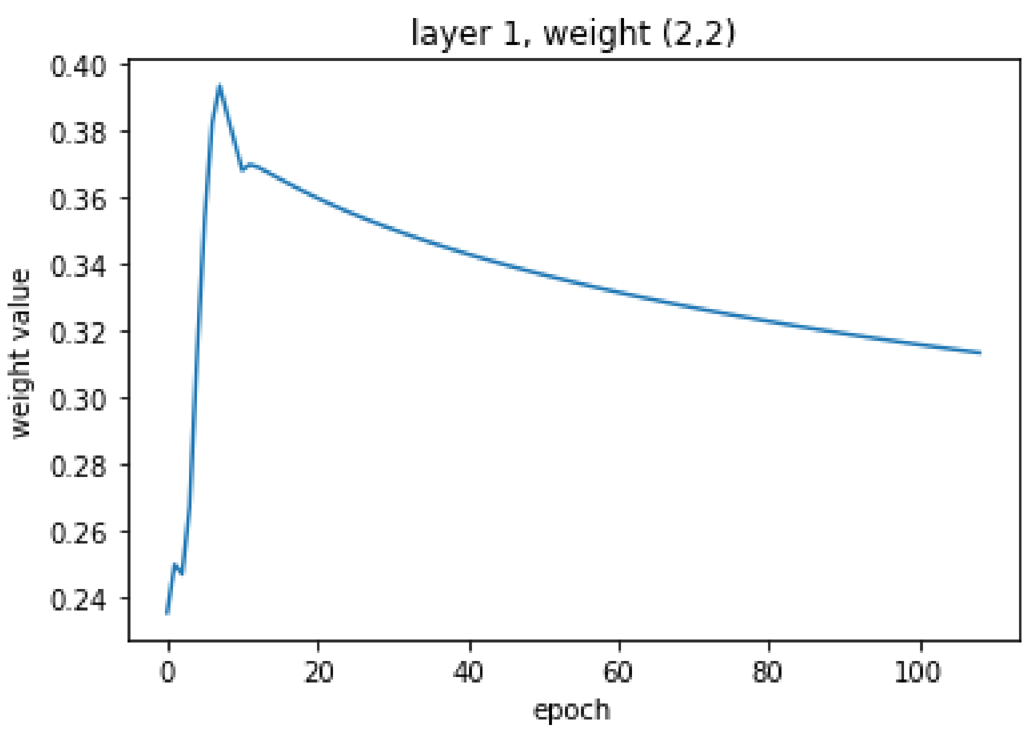


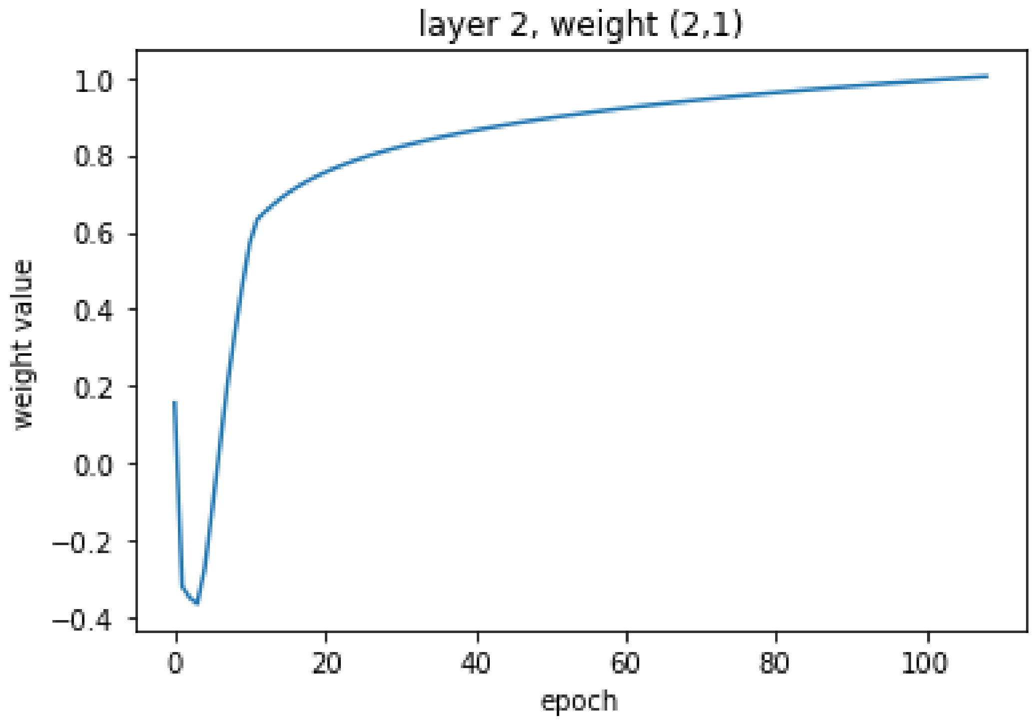

```
number of epochs to reach an mse of  120
```

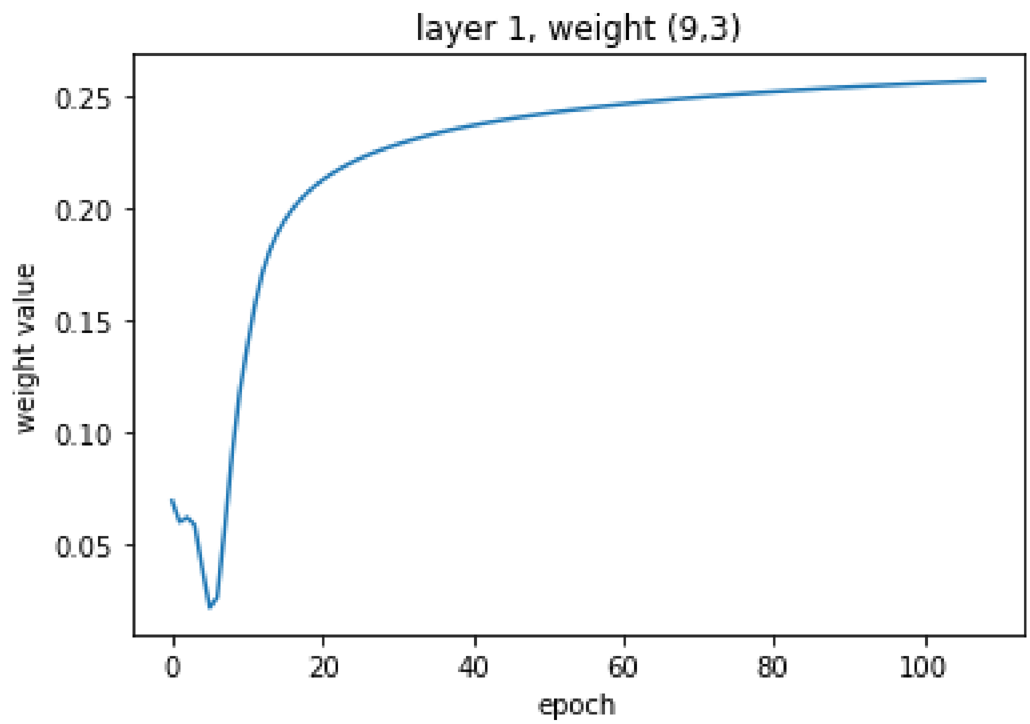
[63]:
```
model2 = Neural_Network()
model2.add_layer(10, 'sigmoid', input_shape=(4,5))
model2.add_layer(5, 'tanh')
```

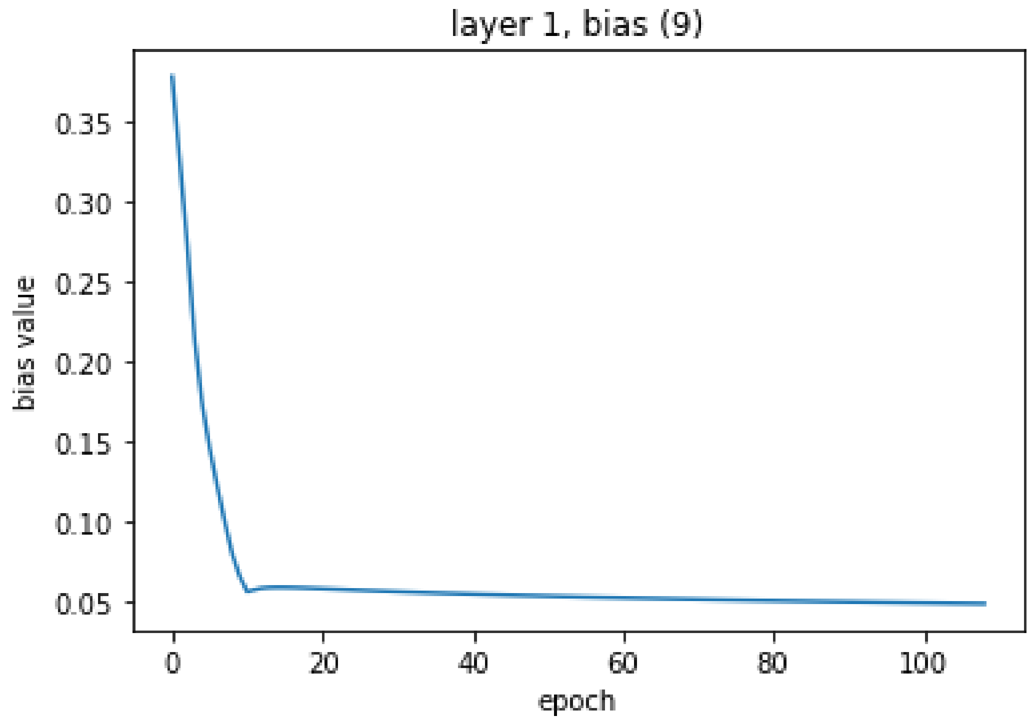
[64]:
```
fit(model2,epochs=500, alpha=0.01, dataset=training_set, targets=y_set, exit=0.
↪002)
```


mse
mse value
epoch
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
20
40
60
80
100

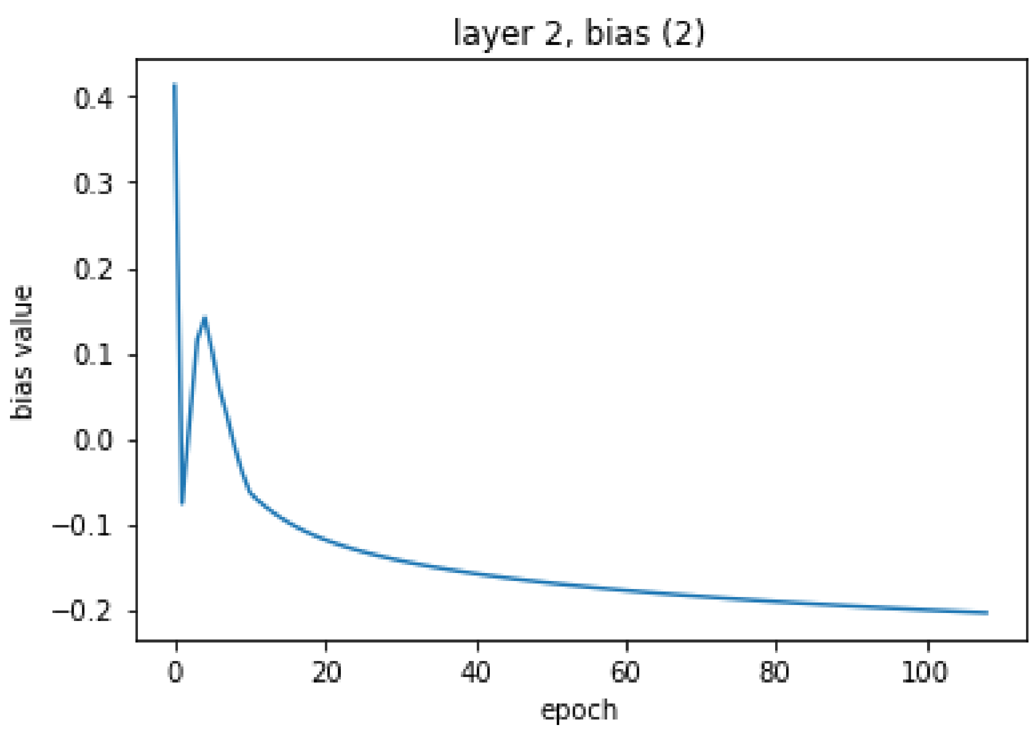

layer 1, weight (2,2)
weight value
epoch
0.40
0.38
0.36
0.34
0.32
0.30
0.28
0.26
0.24
0
20
40
60
80
100


layer 2, weight (2,1)
weight value
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
0
20
40
60
80
100
epoch


layer 1, weight (9,3)
weight value
0.25
0.20
0.15
0.10
0.05
0
20
40
60
80
100
epoch



layer 1, bias (9)
bias value
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0
20
40
60
80
100
epoch

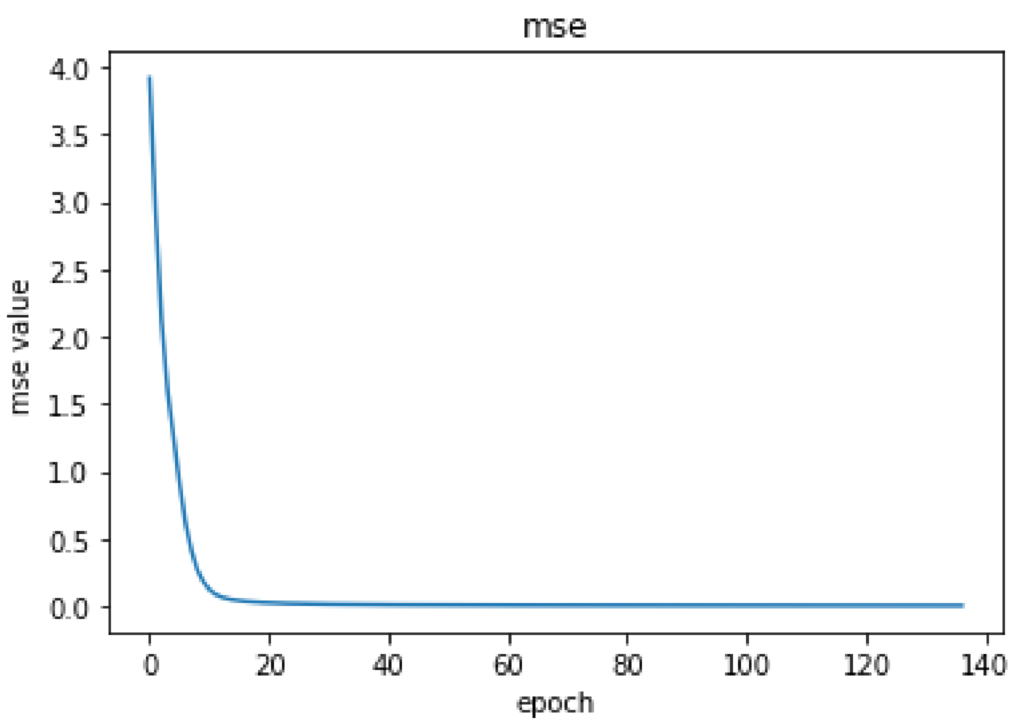


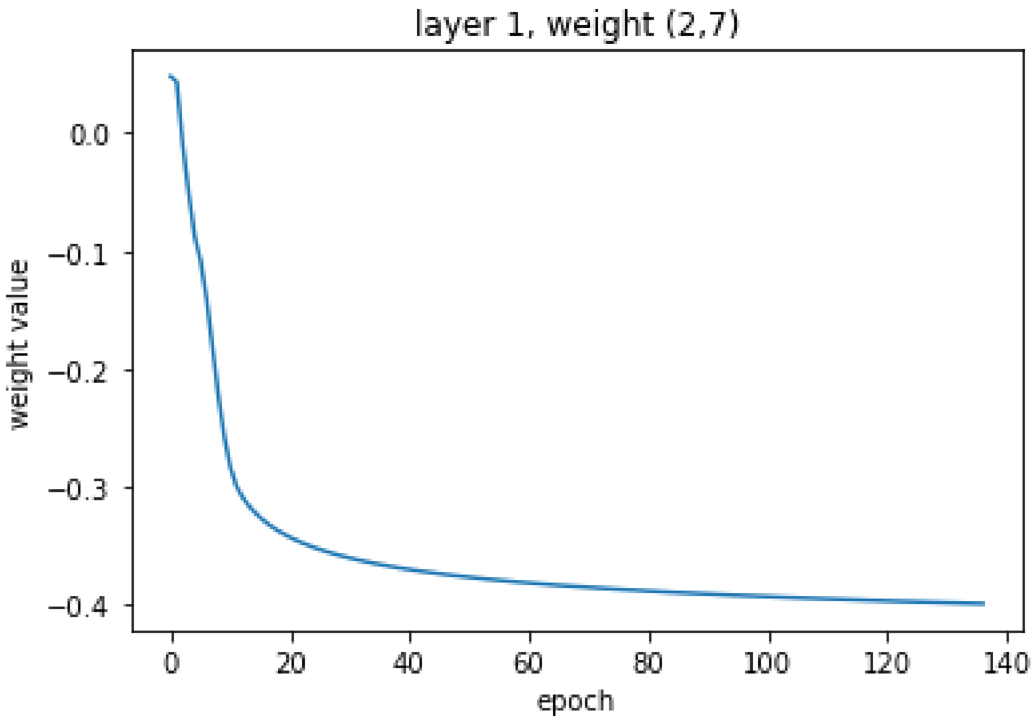
layer 2, bias (2)
bias value
0.4
0.3
0.2
0.1
0.0
−0.1
−0.2
0
20
40
60
80
100
epoch

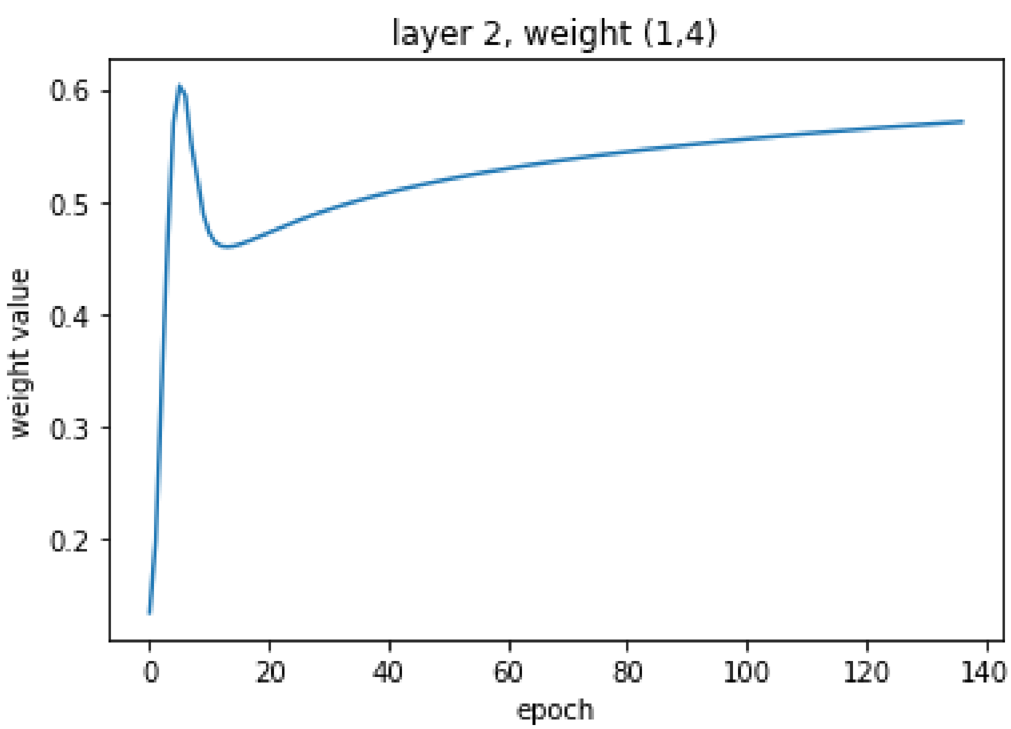
```
number of epochs to reach an mse of  108
```

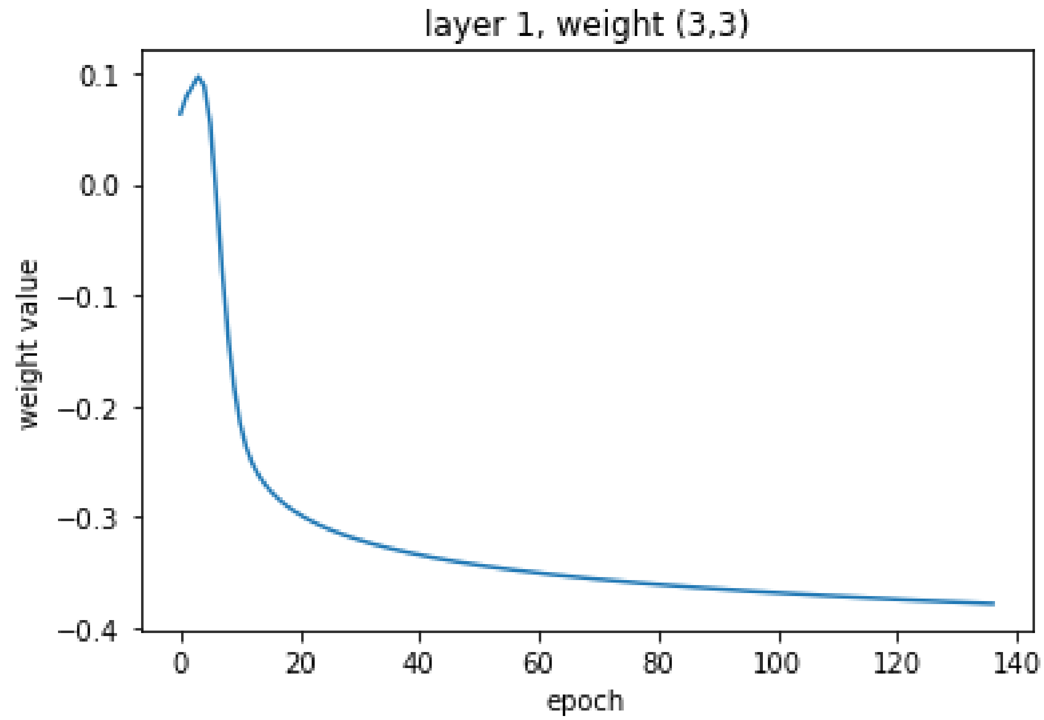
[65]:
```
model3 = Neural_Network()
model3.add_layer(10, 'sigmoid', input_shape=(4,5))
model3.add_layer(5, 'tanh')
```

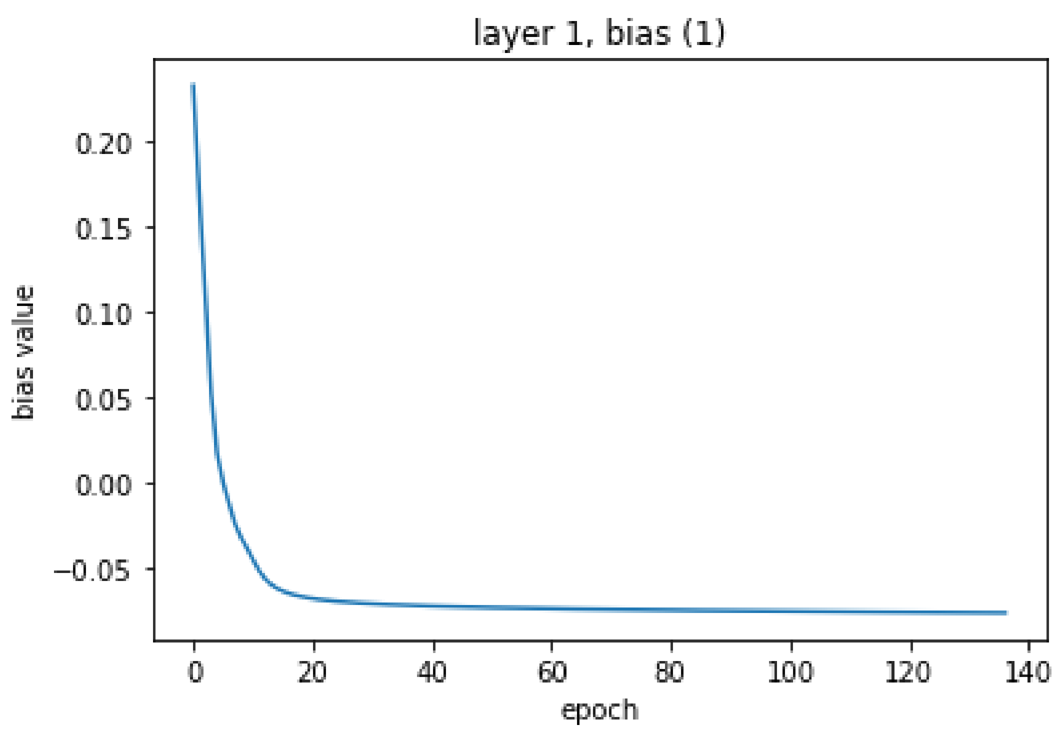
[66]:
```
fit(model3,epochs=500, alpha=0.001, dataset=training_set, targets=y_set, exit=0.
↪002)
```

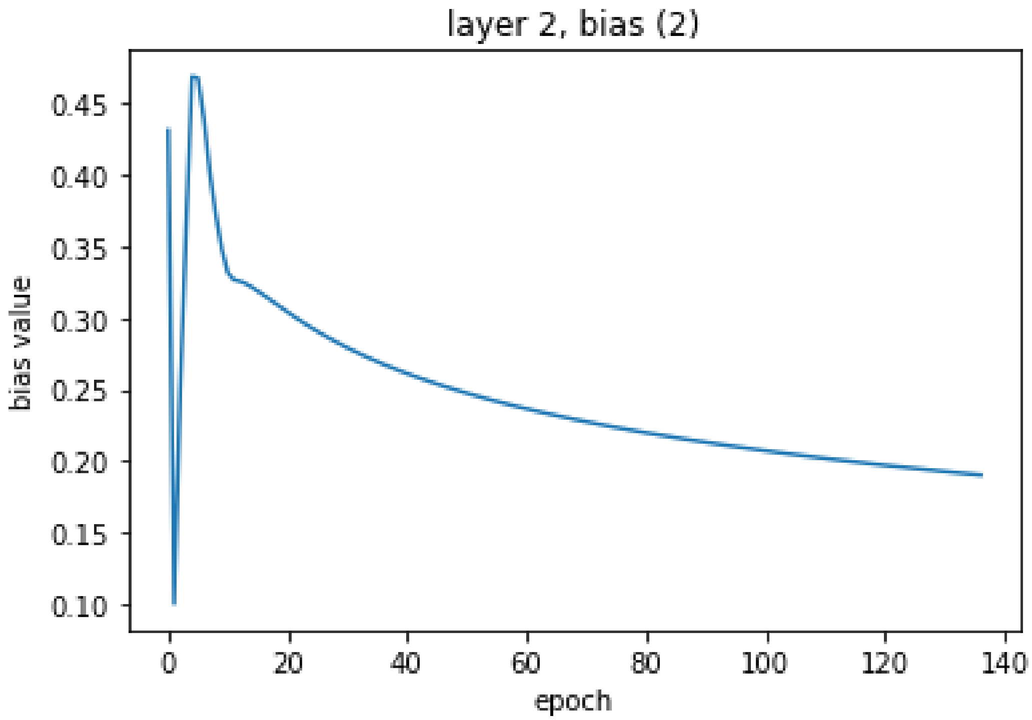



layer 1, weight (2,7)
0.0
−0.1
−0.2
−0.3
−0.4
weight value
0
20
40
60
80
100
120
140
epoch


layer 2, weight (1,4)
0.6
0.5
0.4
0.3
0.2
weight value
0
20
40
60
80
100
120
140
epoch


layer 1, weight (3,3)
weight value
0.1
0.0
−0.1
−0.2
−0.3
−0.4
0
20
40
60
80
100
120
140
epoch


layer 1, bias (1)
bias value
0.20
0.15
0.10
0.05
0.00
−0.05
0
20
40
60
80
100
120
140
epoch




```
number of epochs to reach an mse of  136
```

[67]:
```
for i in range(len(tset1)):
    sets = tset1[i]
    for j in range(len(sets)):
        model1.predict(sets[i].flatten())

    print('')
```

```
[ 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.]

[-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.]

[-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.]

[-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.]

[-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.]
```

[68]:

```python
for i in range(len(tset2)):
    sets = tset2[i]
    for j in range(len(sets)):
        model1.predict(sets[i].flatten())

    print('')
```

```
[ 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.]

[-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.]

[-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.]

[-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.]
```

```
[-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.]
```

[69]:
```
for i in range(len(tset3)):
    sets = tset3[i]
    for j in range(len(sets)):
        model1.predict(sets[i].flatten())

    print('')
```

```
[ 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.]

[-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.]

[-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.]

[-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.]

[-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.]
```

[70]:
```
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
```

[71]:
```
copy_model1 = copy.deepcopy(model2)
copy_model2 = copy.deepcopy(model2)
```

[72]:
```
copy_model1.weights[0].shape
```

[72]: (20, 10)

[73]:
```
destroy_weights(copy_model1, percent=0.2)
```

[74]:
```
print(f"There are {np.count_nonzero(copy_model1.weights[0]==0)} weights that␣
 ↪are zero from input to hidden layer")
print(f"There are {np.count_nonzero(copy_model1.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
```

[75]:
```
for i in range(len(tset1)):
    sets = tset1[i]
    for j in range(len(sets)):
        copy_model1.predict(sets[i].flatten())

    print('')
```

```
[ 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.]

[-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.]

[-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.]

[-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.]

[-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.]
```

[76]:
```
for i in range(len(tset2)):
    sets = tset2[i]
    for j in range(len(sets)):
        copy_model1.predict(sets[i].flatten())

    print('')
```

```
[ 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.]

[-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.]

[-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.]
```

```
[-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.]

[-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.]
```

[77]:
```
for i in range(len(tset3)):
    sets = tset3[i]
    for j in range(len(sets)):
        copy_model1.predict(sets[i].flatten())

    print('')
```

```
[ 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.]

[-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.]

[-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.]

[-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.]

[-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.]
```

[78]:
```
destroy_weights(copy_model1, percent=0.2)
```

[79]:
```
print(f"There are {np.count_nonzero(copy_model1.weights[0]==0)} weights that␣
↪are zero from input to hidden layer")
print(f"There are {np.count_nonzero(copy_model1.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
```

[80]:
```
for i in range(len(tset1)):
    sets = tset1[i]
    for j in range(len(sets)):
        copy_model1.predict(sets[i].flatten())

    print('')
```

```
[ 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.]

[-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.]

[-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.]

[-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.]

[-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.]
```

[81]:
```
for i in range(len(tset2)):
    sets = tset2[i]
    for j in range(len(sets)):
        copy_model1.predict(sets[i].flatten())

    print('')
```

```
[ 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.]

[-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.]

[-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.]

[-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.]

[-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.]
```

[82]:
```
for i in range(len(tset3)):
    sets = tset3[i]
    for j in range(len(sets)):
        copy_model1.predict(sets[i].flatten())

    print('')
```

```
[ 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.]

[-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.]

[-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.]

[-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.]

[-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.]
```