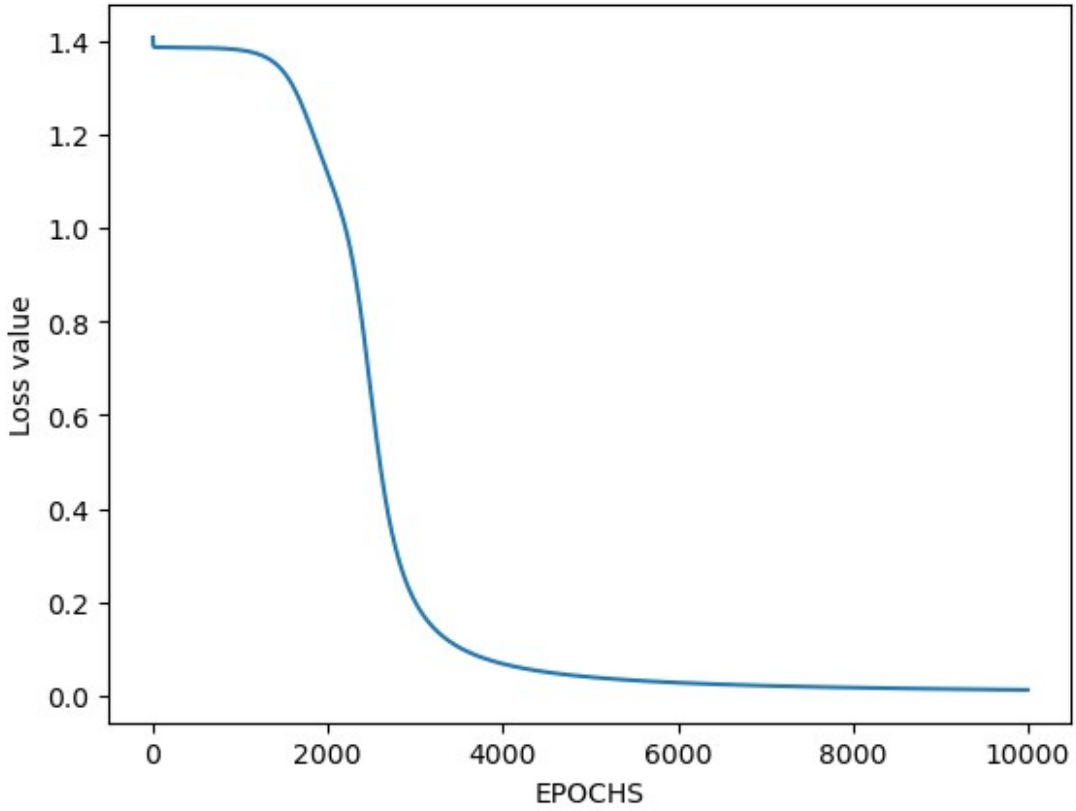
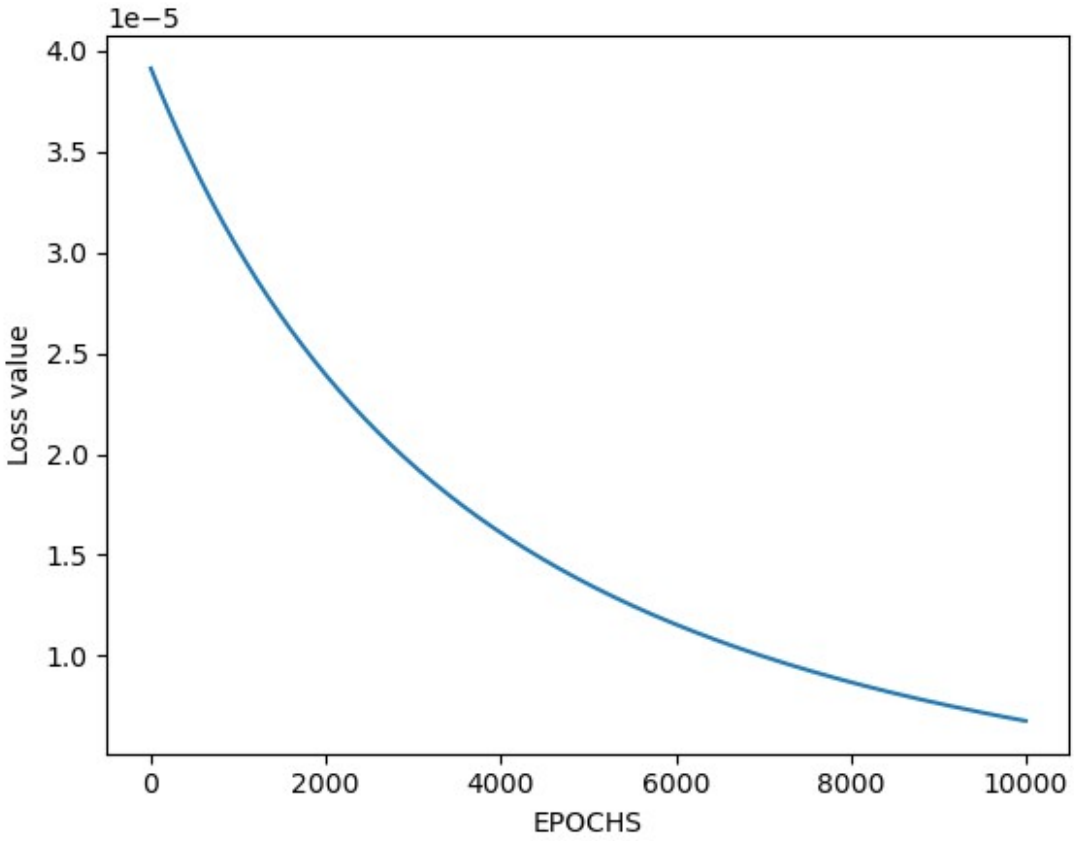
```python
import numpy as np
import matplotlib.pyplot as plt

class ML_perceptron:
    def __init__(self,input_size, hidden_size, output_size):
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.output_size = output_size
        self.w1 = np.random.rand(self.hidden_size, self.input_size)
        self.w2 = np.random.rand(self.output_size, self.hidden_size)
        self.b1 = np.zeros((self.hidden_size, 1))
        self.b2 = np.zeros((self.output_size, 1))
        self.activation_func = None
        self.loss_func = None

    def sigmoid(self,z):
        z= 1/(1+np.exp(-z))
        return z

    def ReLU(self,z):
        z=np.maximum(0,z)
        return z

    def tanh(self, z):
        z=np.tanh(z)
        return z

    def threshold(self, z):
        return (z > 0.5).astype(int)

    def forward_prop(self,x):
        z1 = np.dot(self.w1,x) + self.b1
        a1 = self.sigmoid(z1)
        z2 = np.dot(self.w2,a1) + self.b2
        a2 = self.sigmoid(z2)
        return z1,a1,z2,a2

    def back_prop(self,x,y,z1,a1,z2,a2):
        #For output layer
        dz2 = a2-y
        dw2 = np.dot(dz2,a1.T)/2
        dw2 = np.reshape(dw2,self.w2.shape)

        db2=np.sum(dz2,axis=1)
        db2 = np.reshape(db2,self.b2.shape)

        #For hidden layer
        dz1 = np.dot(self.w2.T,dz2)*a1*(1-a1)
        dw1 = np.dot(dz1,x.T)/2
        dw1 = np.reshape(dw1,self.w1.shape)
```

```
        db1=np.sum(dz1,axis=1,keepdims=True)
        db1 = np.reshape(db1,self.b1.shape)

        #Return corresponding gradient values
        return dw1,dw2,db1,db2

    def compile_func(self, activation_func, loss_func):
        if activation_func == 'sigmoid':
            self.activation_func = self.sigmoid
        elif activation_func == 'ReLU':
            self.activation_func = self.ReLU
        elif activation_func == 'tanh':
            self.activation_func = self.tanh
        elif activation_func == 'threshold':
            self.activation_func = self.threshold
        self.loss_func = loss_func


    def fit(self,x,y,epoch,lr):
        losses=[]
        for i in range(epoch):

            z1,a1,z2,a2 = self.forward_prop(x)
            if self.loss_func == 'binary_crossentropy':
                loss = -(1 / len(x)) * np.sum(y * np.log(a2) + (1 - y)
* np.log(1 - a2))
            elif self.loss_func == 'mse':
                loss = (1 / (2 * len(x))) * np.sum((y - a2)**2)
            losses.append(loss)
            dw1,dw2,db1,db2= self.back_prop(x,y,z1,a1,z2,a2)
            self.w2 = self.w2-lr*dw2
            self.w1 = self.w1-lr*dw1
            self.b2=self.b2-lr*db2
            self.b1=self.b1-lr*db1
        # We plot losses to see how network is doing
        plt.plot(losses)
        plt.xlabel("EPOCHS")
        plt.ylabel("Loss value")

        return self.w1,self.w2,self.b1,self.b2


    def predict(self, x):
        _, _, _, output = self.forward_prop(x)
        return (output > 0.5).astype(int)


input_size = 2
hidden_size = 2
```

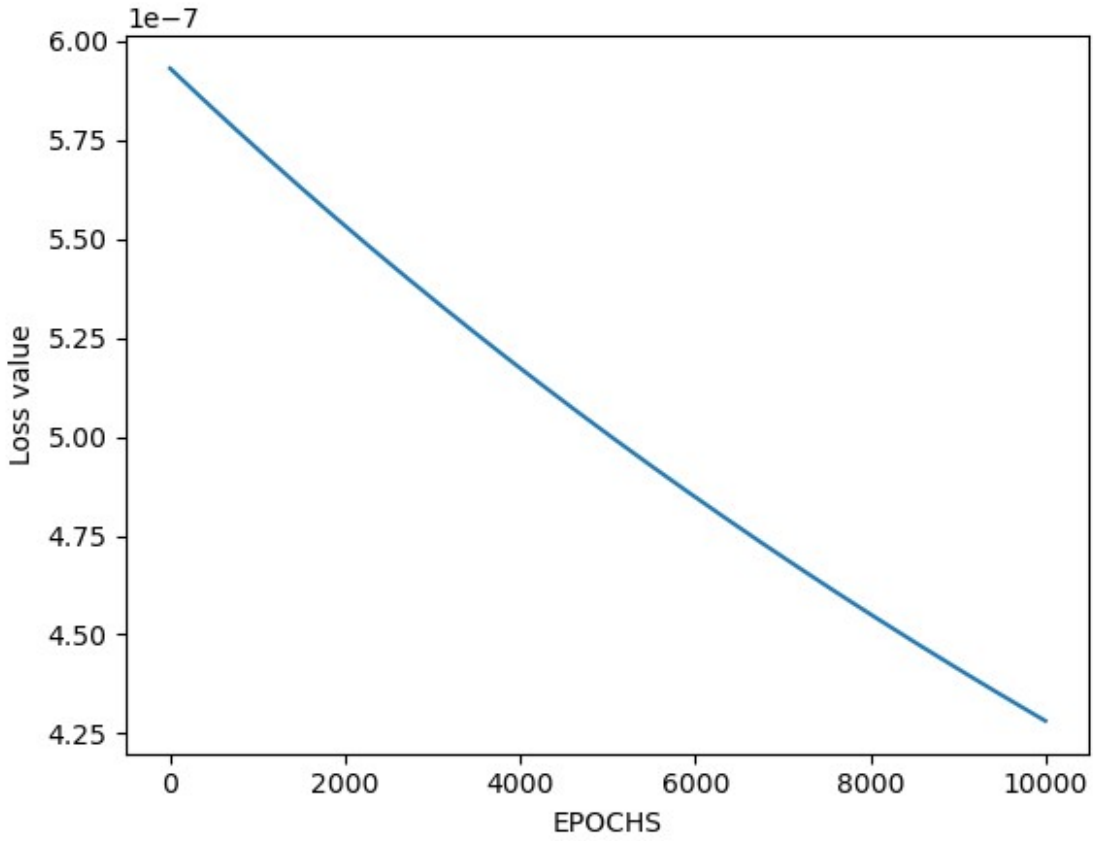
```
output_size = 1
epochs = 10000
learning_rate = 0.1
model= ML_perceptron(input_size, hidden_size, output_size)
model.compile_func(activation_func='sigmoid',
loss_func='binary_crossentropy')
x=np.array([[0,0,1,1],[0,1,0,1]])
y=np.array([[0,1,1,0]])
trained_params= model.fit(x,y,epochs,learning_rate)
```



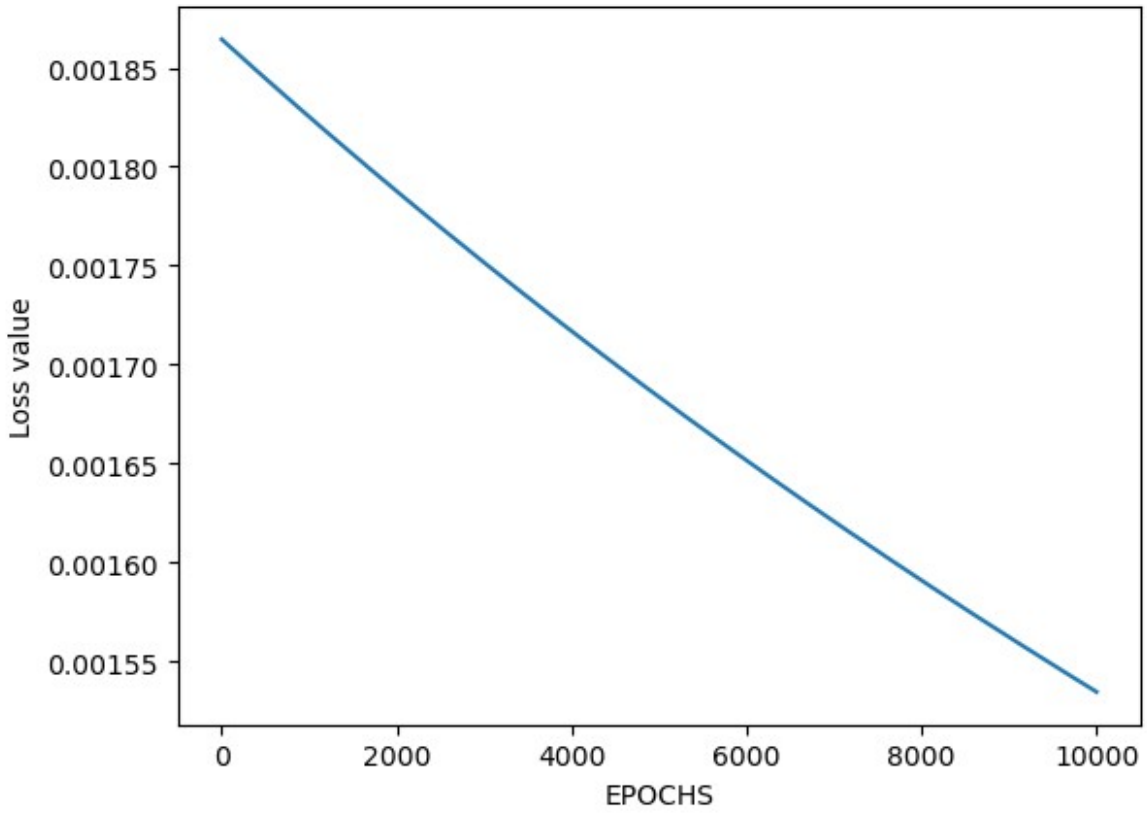

```
model.compile_func(activation_func='ReLU', loss_func='mse')
x=np.array([[0,0,1,1],[0,1,0,1]])
y=np.array([[0,1,1,0]])
trained_params= model.fit(x,y,epochs,learning_rate)
```




```
model.compile_func(activation_func='tanh', loss_func='mse')
x=np.array([[0,0,1,1],[0,1,0,1]])
y=np.array([[0,1,1,0]])
trained_params= model.fit(x,y,epochs,learning_rate)
```




```
model.compile_func(activation_func='threshold',
loss_func='binary_crossentropy')
x=np.array([[0,0,1,1],[0,1,0,1]])
y=np.array([[0,1,1,0]])
trained_params= model.fit(x,y,epochs,learning_rate)
```




```
# Test the model
prediction = model.predict(x)
print("Predictions:", prediction)
```

```
Predictions: [[0 1 1 0]]
```