perceptron-learning

November 10, 2024

[4]:
```
import numpy as np
import matplotlib.pyplot as plt
```

[5]:
```
def predicted(x1,x2,w1,w2):
    s=x1*w1+x2*w2
    if s>=0.5:
        return 1
    else:
        return 0
```

[6]:
```
def error(x1,x2,w1,w2,y):
    return y-predicted(x1,x2,w1,w2)
```

[7]:
```
def lossfun(x1,x2,w1,w2,y):
    sum=0
    for i in range(len(x1)):
        sum=sum+error(x1[i],x2[i],w1,w2,y[i])
    return sum
```

[8]:
```
def training(x1,x2,y,lr,iw1,iw2):
    w1=iw1
    w2=iw2
    losslist=[]
    for epoch in (range(100)):
        loss=lossfun(x1,x2,w1,w2,y)
        losslist.append(loss)
        for i in range(len(x1)):
            w1=w1+lr*error(x1[i],x2[i],w1,w2,y[i])*x1[i]
            w2=w2+lr*error(x1[i],x2[i],w1,w2,y[i])*x2[i]
    return w1,w2,losslist
```

[9]:
```
def linepoints(x1,w1,w2,th):
    x2=[]
    for i in range(len(x1)):
        x2.append((0.5-x1[i]*w1)/w2)
    return x2
```

1 OR function

[10]:
```
x1=[0,0,1,1]
x2=[0,1,0,1]
y=[0,1,1,1]
```

[11]:
```
w1,w2,losses=training(x1,x2,y,0.01,0.5,0.3)
```

[12]:
```
print(w1,w2)
```

```
0.5 0.5000000000000001
```

[13]:
```
print(losses)
```

```
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

[14]:
```
print(predicted(0,0,w1,w2))
```

```
0
```

[15]:
```
print(predicted(0,1,w1,w2))
```

```
1
```

[16]:
```
print(predicted(1,0,w1,w2))
```

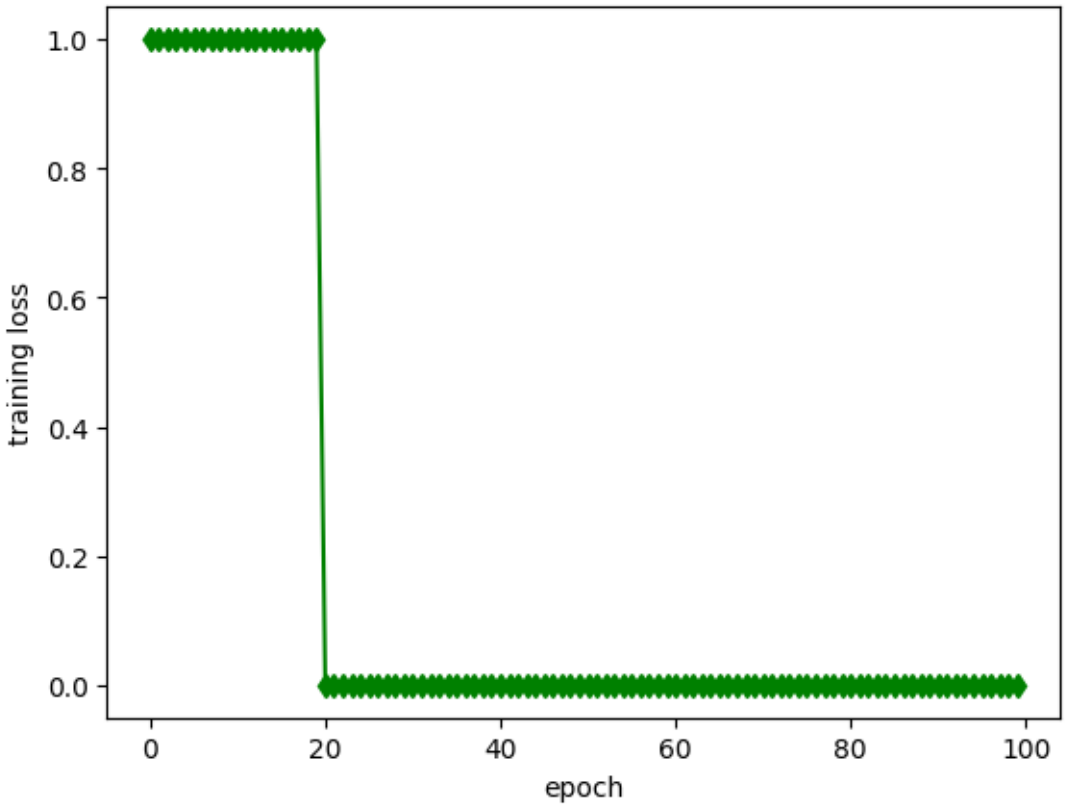
```
1
```

[17]:
```
print(predicted(1,1,w1,w2))
```

```
1
```

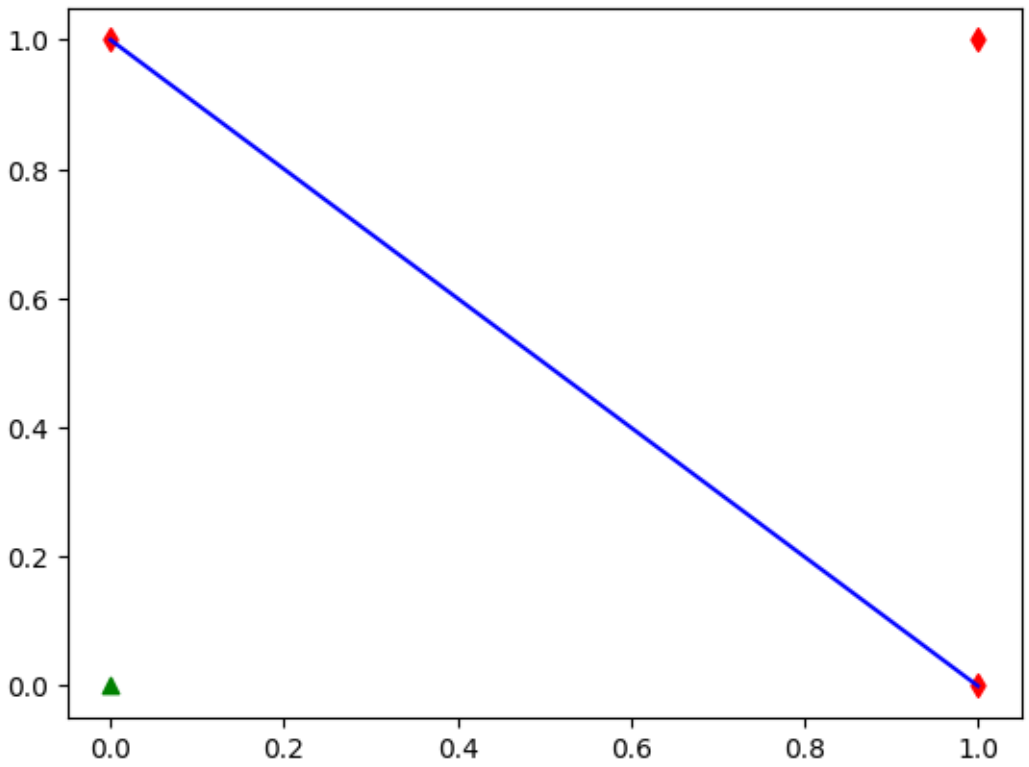
[18]:
```
epochs=[x for x in range(len(losses))]
```

[19]:
```
plt.plot(epochs,losses,color='g',marker='d')
plt.xlabel('epoch')
plt.ylabel('training loss')
plt.show()
```



[20]:
```
x2pred=linepoints(x1,w1,w2,0.5)
```

[21]:
```
plt.scatter(x1[0],x2[0],color='g',marker='^')
plt.scatter(x1[1:],x2[1:],color='r',marker='d')
plt.plot(x1,x2pred,color='b')
plt.show()
```



2 AND function

[22]:
```
x1=[0,0,1,1]
x2=[0,1,0,1]
y=[0,0,0,1]
```

[23]:
```
w1,w2,losses=training(x1,x2,y,0.01,0.6,0.7)
```

[24]:
```
print(w1,w2)
```

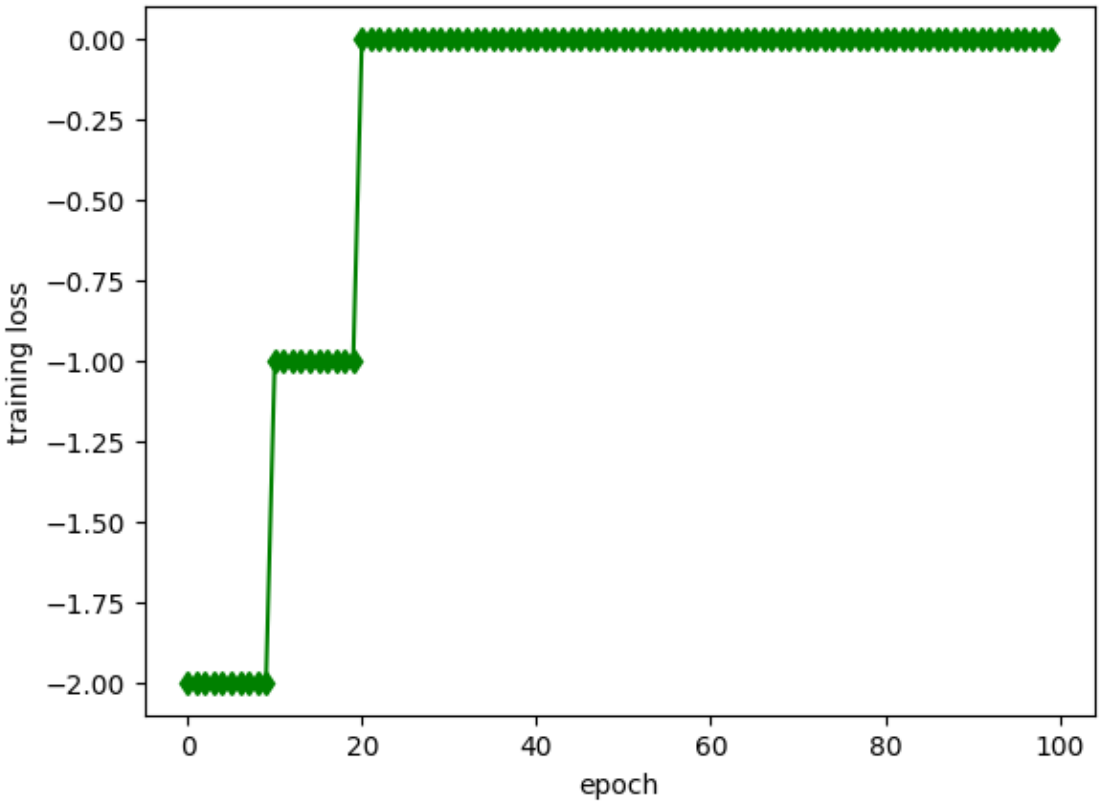
```
0.4999999999999999 0.4999999999999998
```

[25]:
```
print(losses)
```

```
[-2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

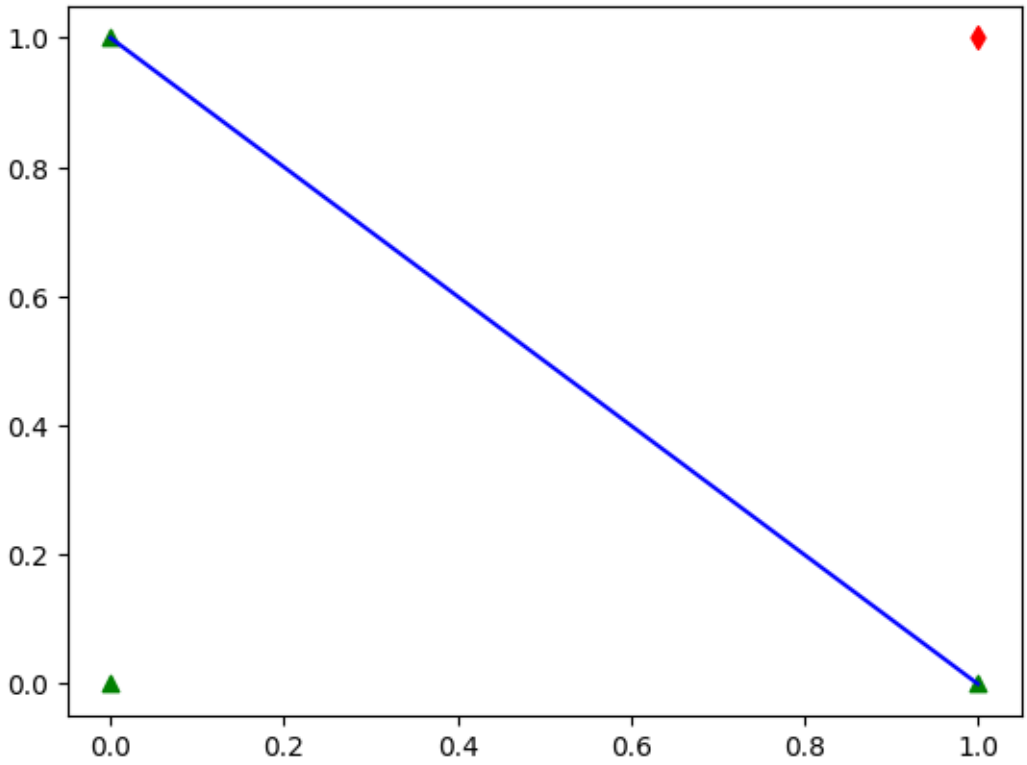
[26]:
```
epochs=[x for x in range(len(losses))]
```

[27]:
```
plt.plot(epochs,losses,color='g',marker='d')
plt.xlabel('epoch')
plt.ylabel('training loss')
plt.show()
```



[28]:
```
x2pred=linepoints(x1,w1,w2,0.5)
```

[29]:
```
plt.scatter(x1[0:3],x2[0:3],color='g',marker='^')
plt.scatter(x1[3],x2[3],color='r',marker='d')
plt.plot(x1,x2pred,color='b')
plt.show()
```



[32]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

[36]:
```
diabetes = load_diabetes()
```

[37]:
```
X = diabetes.data
y = (diabetes.target > 140).astype(int)
```

[39]:
```
x1 = X[:, 2]
x2 = X[:, 3]
```

[39]:
```
array([ 0.06169621, -0.05147406,  0.04445121, -0.01159501, -0.03638469,
       -0.04069594, -0.04716281, -0.00189471,  0.06169621,  0.03906215,
       -0.08380842,  0.01750591, -0.02884001, -0.00189471, -0.02560657,
       -0.01806189,  0.04229559,  0.01211685, -0.0105172 , -0.01806189,
       -0.05686312, -0.02237314, -0.00405033,  0.06061839,  0.03582872,
       -0.01267283, -0.07734155,  0.05954058, -0.02129532, -0.00620595,
        0.04445121, -0.06548562,  0.12528712, -0.05039625, -0.06332999,
```

```
-0.03099563,  0.02289497,  0.01103904,  0.07139652,  0.01427248,
-0.00836158, -0.06764124, -0.0105172 , -0.02345095,  0.06816308,
-0.03530688, -0.01159501, -0.0730303 , -0.04177375,  0.01427248,
-0.00728377,  0.0164281 , -0.00943939, -0.01590626,  0.0250506 ,
-0.04931844,  0.04121778, -0.06332999, -0.06440781, -0.02560657,
-0.00405033,  0.00457217, -0.00728377, -0.0374625 , -0.02560657,
-0.02452876, -0.01806189, -0.01482845, -0.02991782, -0.046085  ,
-0.06979687,  0.03367309, -0.00405033, -0.02021751,  0.00241654,
-0.03099563,  0.02828403, -0.03638469, -0.05794093, -0.0374625 ,
 0.01211685, -0.02237314, -0.03530688,  0.00996123, -0.03961813,
 0.07139652, -0.07518593, -0.00620595, -0.04069594, -0.04824063,
-0.02560657,  0.0519959 ,  0.00457217, -0.06440781, -0.01698407,
-0.05794093,  0.00996123,  0.08864151, -0.00512814, -0.06440781,
 0.01750591, -0.04500719,  0.02828403,  0.04121778,  0.06492964,
-0.03207344, -0.07626374,  0.04984027,  0.04552903, -0.00943939,
-0.03207344,  0.00457217,  0.02073935,  0.01427248,  0.11019775,
 0.00133873,  0.05846277, -0.02129532, -0.0105172 , -0.04716281,
 0.00457217,  0.01750591,  0.08109682,  0.0347509 ,  0.02397278,
-0.00836158, -0.06117437, -0.00189471, -0.06225218,  0.0164281 ,
 0.09618619, -0.06979687, -0.02129532, -0.05362969,  0.0433734 ,
 0.05630715, -0.0816528 ,  0.04984027,  0.11127556,  0.06169621,
 0.01427248,  0.04768465,  0.01211685,  0.00564998,  0.04660684,
 0.12852056,  0.05954058,  0.09295276,  0.01535029, -0.00512814,
 0.0703187 , -0.00405033, -0.00081689, -0.04392938,  0.02073935,
 0.06061839, -0.0105172 , -0.03315126, -0.06548562,  0.0433734 ,
-0.06225218,  0.06385183,  0.03043966,  0.07247433, -0.0191397 ,
-0.06656343, -0.06009656,  0.06924089,  0.05954058, -0.02668438,
-0.02021751, -0.046085  ,  0.07139652, -0.07949718,  0.00996123,
-0.03854032,  0.01966154,  0.02720622, -0.00836158, -0.01590626,
 0.00457217, -0.04285156,  0.00564998, -0.03530688,  0.02397278,
-0.01806189,  0.04229559, -0.0547075 , -0.00297252, -0.06656343,
-0.01267283, -0.04177375, -0.03099563, -0.00512814, -0.05901875,
 0.0250506 , -0.046085  ,  0.00349435,  0.05415152, -0.04500719,
-0.05794093, -0.05578531,  0.00133873,  0.03043966,  0.00672779,
 0.04660684,  0.02612841,  0.04552903,  0.04013997, -0.01806189,
 0.01427248,  0.03690653,  0.00349435, -0.07087468, -0.03315126,
 0.09403057,  0.03582872,  0.03151747, -0.06548562, -0.04177375,
-0.03961813, -0.03854032, -0.02560657, -0.02345095, -0.06656343,
 0.03259528, -0.046085  , -0.02991782, -0.01267283, -0.01590626,
 0.07139652, -0.03099563,  0.00026092,  0.03690653,  0.03906215,
-0.01482845,  0.00672779, -0.06871905, -0.00943939,  0.01966154,
 0.07462995, -0.00836158, -0.02345095, -0.046085  ,  0.05415152,
-0.03530688, -0.03207344, -0.0816528 ,  0.04768465,  0.06061839,
 0.05630715,  0.09834182,  0.05954058,  0.03367309,  0.05630715,
-0.06548562,  0.16085492, -0.05578531, -0.02452876, -0.03638469,
-0.00836158, -0.04177375,  0.12744274, -0.07734155,  0.02828403,
-0.02560657, -0.06225218, -0.00081689,  0.08864151, -0.03207344,
```

```
 0.03043966,  0.00888341,  0.00672779, -0.02021751, -0.02452876,
-0.01159501,  0.02612841, -0.05901875, -0.03638469, -0.02452876,
 0.01858372, -0.0902753 , -0.00512814, -0.05255187, -0.02237314,
-0.02021751, -0.0547075 , -0.00620595, -0.01698407,  0.05522933,
 0.07678558,  0.01858372, -0.02237314,  0.09295276, -0.03099563,
 0.03906215, -0.06117437, -0.00836158, -0.0374625 , -0.01375064,
 0.07355214, -0.02452876,  0.03367309,  0.0347509 , -0.03854032,
-0.03961813, -0.00189471, -0.03099563, -0.046085  ,  0.00133873,
 0.06492964,  0.04013997, -0.02345095,  0.05307371,  0.04013997,
-0.02021751,  0.01427248, -0.03422907,  0.00672779,  0.00457217,
 0.03043966,  0.0519959 ,  0.06169621, -0.00728377,  0.00564998,
 0.05415152, -0.00836158,  0.114509  ,  0.06708527, -0.05578531,
 0.03043966, -0.02560657,  0.10480869, -0.00620595, -0.04716281,
-0.04824063,  0.08540807, -0.01267283, -0.03315126, -0.00728377,
-0.01375064,  0.05954058,  0.02181716,  0.01858372, -0.01159501,
-0.00297252,  0.01750591, -0.02991782, -0.02021751, -0.05794093,
 0.06061839, -0.04069594, -0.07195249, -0.05578531,  0.04552903,
-0.00943939, -0.03315126,  0.04984027, -0.08488624,  0.00564998,
 0.02073935, -0.00728377,  0.10480869, -0.02452876, -0.00620595,
-0.03854032,  0.13714305,  0.17055523,  0.00241654,  0.03798434,
-0.05794093, -0.00943939, -0.02345095, -0.0105172 , -0.03422907,
-0.00297252,  0.06816308,  0.00996123,  0.00241654, -0.03854032,
 0.02612841, -0.08919748,  0.06061839, -0.02884001, -0.02991782,
-0.0191397 , -0.04069594,  0.01535029, -0.02452876,  0.00133873,
 0.06924089, -0.06979687, -0.02991782, -0.046085  ,  0.01858372,
 0.00133873, -0.03099563, -0.00405033,  0.01535029,  0.02289497,
 0.04552903, -0.04500719, -0.03315126,  0.097264  ,  0.05415152,
 0.12313149, -0.08057499,  0.09295276, -0.05039625, -0.01159501,
-0.0277622 ,  0.05846277,  0.08540807, -0.00081689,  0.00672779,
 0.00888341,  0.08001901,  0.07139652, -0.02452876, -0.0547075 ,
-0.03638469,  0.0164281 ,  0.07786339, -0.03961813,  0.01103904,
-0.04069594, -0.03422907,  0.00564998,  0.08864151, -0.03315126,
-0.05686312, -0.03099563,  0.05522933, -0.06009656,  0.00133873,
-0.02345095, -0.07410811,  0.01966154, -0.01590626, -0.01590626,
 0.03906215, -0.0730303 ])
```

[40]:
```
x1_train, x1_test, x2_train, x2_test, y_train, y_test = train_test_split(x1,␣
↪x2, y, test_size=0.2, random_state=42)
```

[41]:
```
def predicted(x1, x2, w1, w2):
    s = x1 * w1 + x2 * w2
    return 1 if s >= 0.5 else 0
```

[42]:
```
def error(x1, x2, w1, w2, y):
    return y - predicted(x1, x2, w1, w2)
```

[43]:
```
def lossfun(x1, x2, w1, w2, y):
    return sum(error(x1[i], x2[i], w1, w2, y[i]) for i in range(len(x1)))
```

[44]:
```
def training(x1, x2, y, lr, iw1, iw2):
    w1, w2 = iw1, iw2
    losslist = []
    for epoch in range(100):
        loss = lossfun(x1, x2, w1, w2, y)
        losslist.append(loss)
        for i in range(len(x1)):
            w1 += lr * error(x1[i], x2[i], w1, w2, y[i]) * x1[i]
            w2 += lr * error(x1[i], x2[i], w1, w2, y[i]) * x2[i]
    return w1, w2, losslist
```

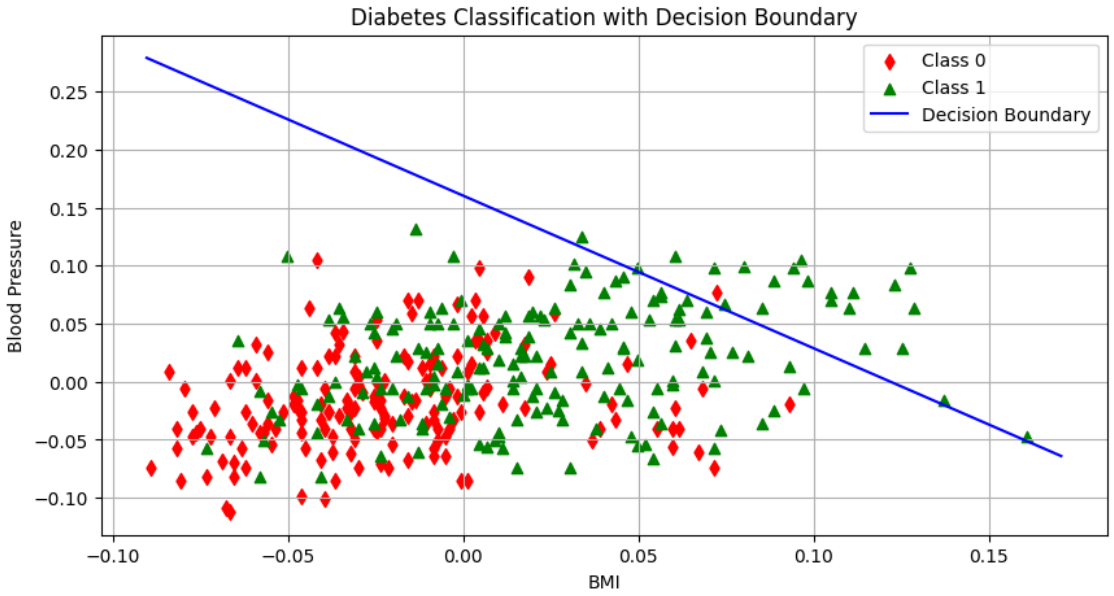
[45]:
```
def linepoints(x1, w1, w2, th):
    return [(0.5 - x * w1) / w2 for x in x1]
```

[46]:
```
initial_w1 = 0.5
initial_w2 = 0.3
learning_rate = 0.01
```

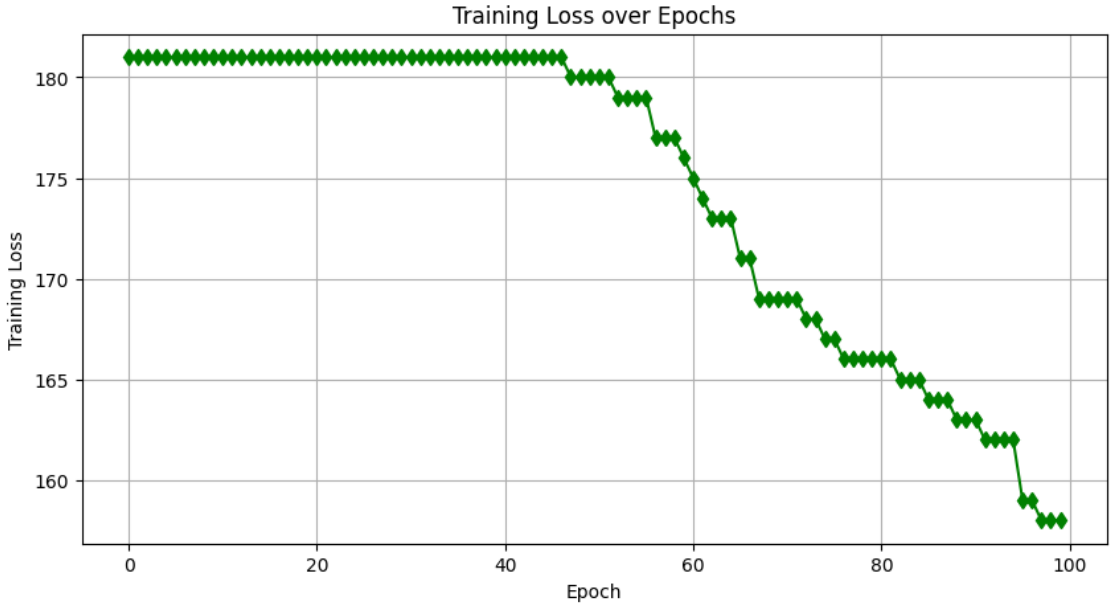
[47]:
```
w1, w2, losses = training(x1_train, x2_train, y_train, learning_rate,␣
 ↪initial_w1, initial_w2)
```

[48]:
```
x1_vals = np.linspace(min(x1), max(x1), 100)
x2pred = linepoints(x1_vals, w1, w2, 0.5)
```

[49]:
```
plt.figure(figsize=(10, 5))
plt.scatter(x1_train[y_train == 0], x2_train[y_train == 0], color='r',␣
 ↪marker='d', label='Class 0')
plt.scatter(x1_train[y_train == 1], x2_train[y_train == 1], color='g',␣
 ↪marker='^', label='Class 1')
plt.plot(x1_vals, x2pred, color='b', label='Decision Boundary')
plt.xlabel('BMI')
plt.ylabel('Blood Pressure')
plt.title('Diabetes Classification with Decision Boundary')
plt.legend()
plt.grid()
plt.show()
```



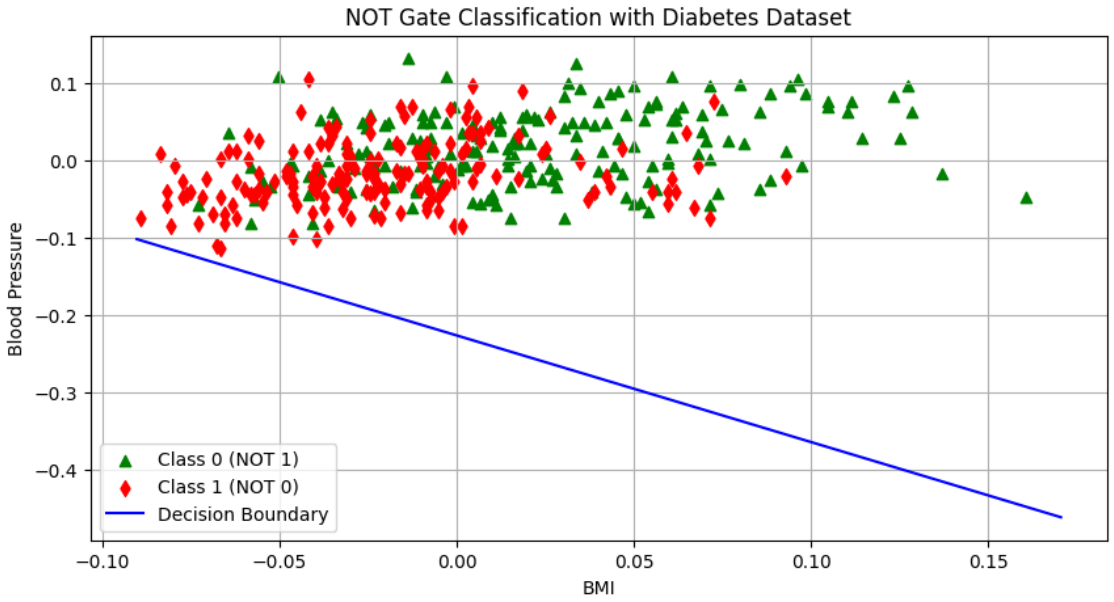
[50]:
```
epochs = range(len(losses))
plt.figure(figsize=(10, 5))
plt.plot(epochs, losses, color='g', marker='d')
plt.xlabel('Epoch')
plt.ylabel('Training Loss')
plt.title('Training Loss over Epochs')
plt.grid()
plt.show()
```

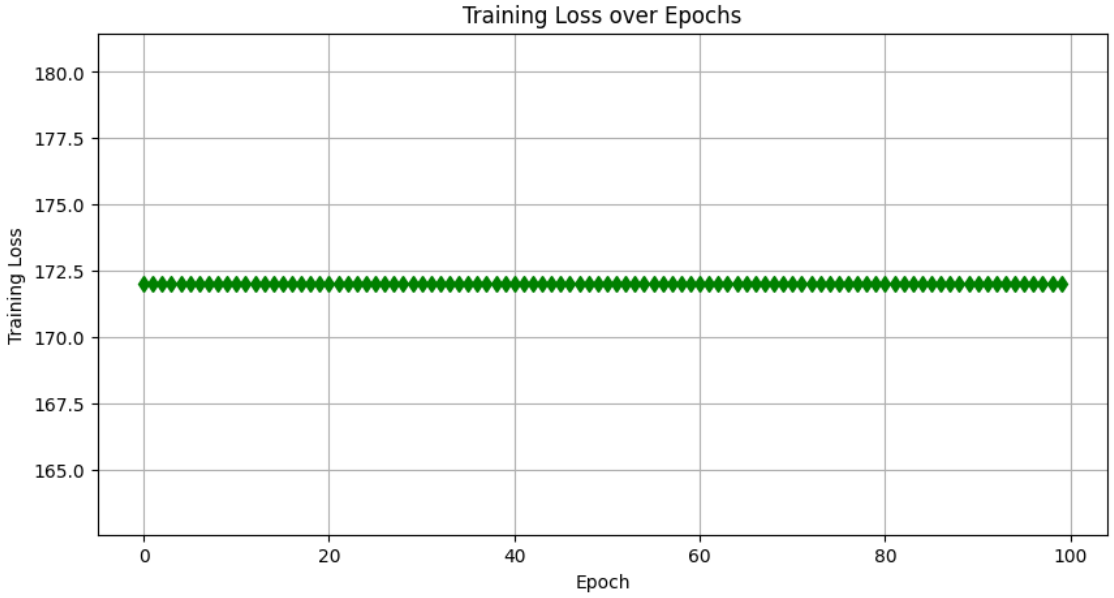


[57]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split

# Load the diabetes dataset
diabetes = load_diabetes()
X = diabetes.data
y = (diabetes.target > 140).astype(int)  # Binary classification based on target

# Invert the labels for NOT gate logic
y_not = 1 - y  # NOT gate: 0 becomes 1 and 1 becomes 0

# Select two features for visualization (e.g., BMI and Blood Pressure)
x1 = X[:, 2]  # BMI
x2 = X[:, 3]  # Blood Pressure

# Split the dataset into training and testing sets
x1_train, x1_test, x2_train, x2_test, y_train, y_test = train_test_split(x1,␣
 ↪x2, y_not, test_size=0.2, random_state=42)

def predicted(x1, x2, w1, w2):
    s = x1 * w1 + x2 * w2
    return 1 if s >= 0.5 else 0

def error(x1, x2, w1, w2, y):
    return y - predicted(x1, x2, w1, w2)

def lossfun(x1, x2, w1, w2, y):
    return sum(error(x1[i], x2[i], w1, w2, y[i]) for i in range(len(x1)))

def training(x1, x2, y, lr, iw1, iw2):
    w1, w2 = iw1, iw2
    losslist = []
    for epoch in range(100):
        loss = lossfun(x1, x2, w1, w2, y)
        losslist.append(loss)
        for i in range(len(x1)):
            w1 += lr * error(x1[i], x2[i], w1, w2, y[i]) * x1[i]
            w2 += lr * error(x1[i], x2[i], w1, w2, y[i]) * x2[i]
    return w1, w2, losslist

def linepoints(x1, w1, w2):
    return [(0.5 - x * w1) / w2 for x in x1]
```

```
# Train the model with specified initial weights
initial_w1 = 0.6
initial_w2 = 0.7
learning_rate = 0.01

w1, w2, losses = training(x1_train, x2_train, y_train, learning_rate,␣
 ↪initial_w1, initial_w2)

# Print the final weights
print("Final Weights: w1 =", w1, ", w2 =", w2)

# Calculate decision boundary points
x1_vals = np.linspace(min(x1), max(x1), 100)
x2pred = linepoints(x1_vals, w1, w2)

# Custom scatter plot
plt.figure(figsize=(10, 5))
plt.scatter(x1_train[y_train == 0], x2_train[y_train == 0], color='g',␣
 ↪marker='^', label='Class 0 (NOT 1)')  # Class 0
plt.scatter(x1_train[y_train == 1], x2_train[y_train == 1], color='r',␣
 ↪marker='d', label='Class 1 (NOT 0)')  # Class 1
plt.plot(x1_vals, x2pred, color='b', label='Decision Boundary')  # Decision␣
 ↪boundary
plt.xlabel('BMI')
plt.ylabel('Blood Pressure')
plt.title('NOT Gate Classification with Diabetes Dataset')
plt.legend()
plt.grid()
plt.show()

# Plot training loss over epochs
epochs = range(len(losses))
plt.figure(figsize=(10, 5))
plt.plot(epochs, losses, color='g', marker='d')
plt.xlabel('Epoch')
plt.ylabel('Training Loss')
plt.title('Training Loss over Epochs')
plt.grid()
plt.show()
```

```
Final Weights: w1 = -3.0509396524510692 , w2 = -2.2148083030892063
```


NOT Gate Classification with Diabetes Dataset
Blood Pressure
BMI
Class 0 (NOT 1)
Class 1 (NOT 0)
Decision Boundary
0.1
0.0
−0.1
−0.2
−0.3
−0.4
−0.10
−0.05
0.00
0.05
0.10
0.15


Training Loss over Epochs
Training Loss
Epoch
180.0
177.5
175.0
172.5
170.0
167.5
165.0
0
20
40
60
80
100