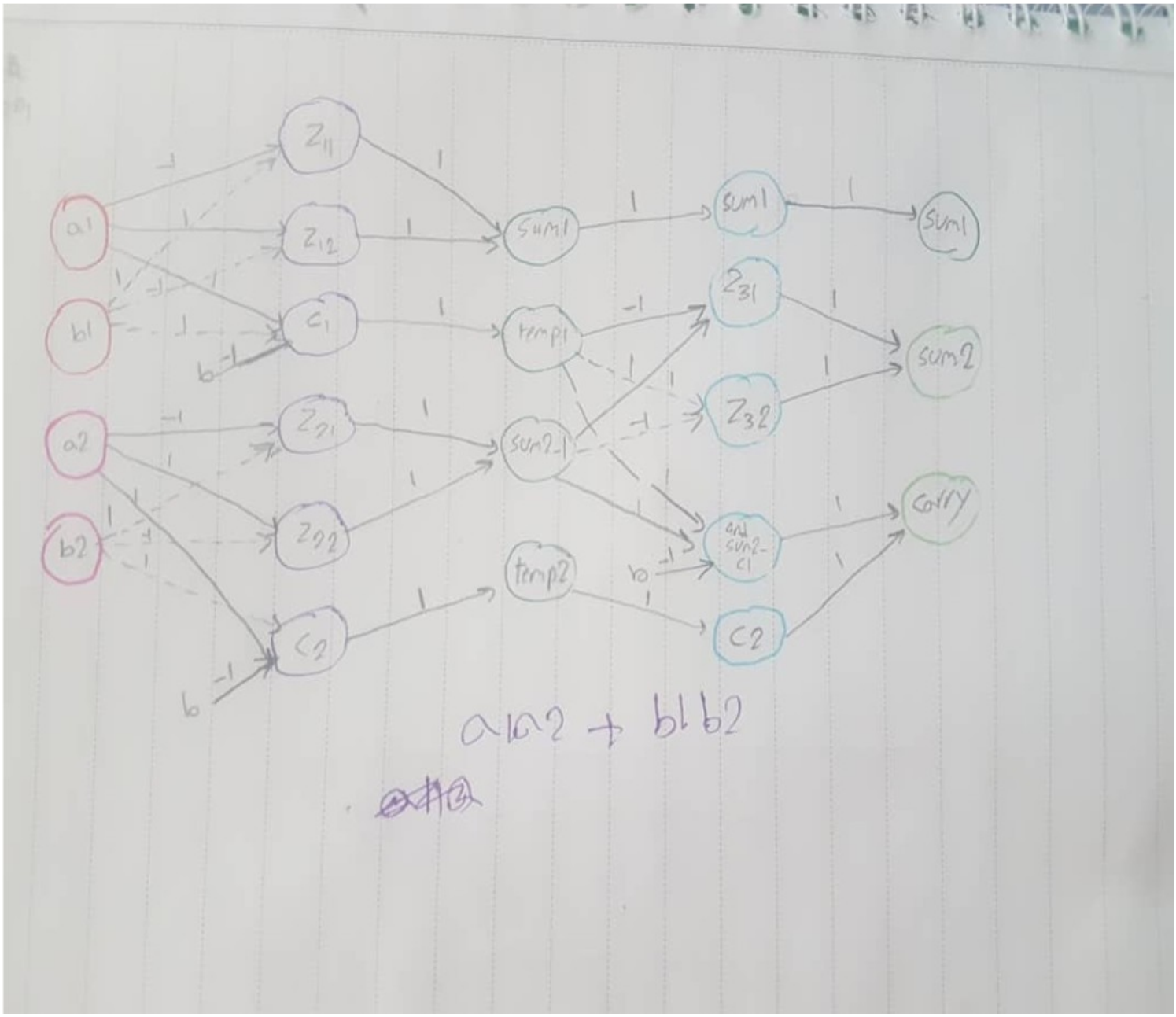
سؤال ۱:

در این سؤال یک شبکه عصبی ساده می‌خواهیم طراحی کنیم که یک عملیات جمع ۲ بیتی را انجام دهد
برای این کار باید تمام وزن ها و بایاس ها را پیدا کنیم و در این قسمت این عدد ها را باید بدون train کردن پیدا کنیم

شبکه عصبی طراحی شده به شکل زیر است:



کد مربوط به آن نیز آورده شده است:

```
def full_adder_1bit(a1, a2, b1, b2):
    bias = 1
    # Layer 1
    z11 = np.heaviside(np.sum(a1 * -1 + b1 * 1), 0)
    z12 = np.heaviside(np.sum(a1 * 1 + b1 * -1), 0)
    c1 = np.heaviside(np.sum(a1 * 1 + b1 * 1 + bias * -1), 0)

    z21 = np.heaviside(np.sum(a2 * -1 + b2 * 1), 0)
    z22 = np.heaviside(np.sum(a2 * 1 + b2 * -1), 0)
    c2 = np.heaviside(np.sum(a2 * 1 + b2 * 1 + bias * -1), 0)

    # Layer 2
    sum1 = np.heaviside(np.sum(z11 * 1 + z12 * 1), 0)
    sum2_1 = np.heaviside(np.sum(z21 * 1 + z22 * 1), 0)

    # Layer 3
    z31 = np.heaviside(np.sum(c1 * -1 + sum2_1 * 1), 0)
    z32 = np.heaviside(np.sum(c1 * 1 + sum2_1 * -1), 0)
    and_sum2_c1 = np.heaviside(np.sum(c1 * 1 + sum2_1 * 1 + -1), 0)
```

```
    # Layer 4
    sum2 = np.heaviside(np.sum(z31 * 1 + z32 * 1), 0)
    carry = np.heaviside(np.sum(c2 * 1 + and_sum2_c1 * 1), 0)

    return sum1, sum2, carry
```

که برای تست کردن آن داریم:

```
for bit11 in range(2):
    for bit12 in range(2):
        for bit21 in range(2):
            for bit22 in range(2):
                print("bit11:", bit11, " bit12:", bit12)
                print("bit21:", bit21, " bit22:", bit22)
                print(full_adder_1bit(bit11, bit12, bit21, bit22))
```

برای هر ۱۶ حالت خروجی را تست می کنیم:

```
bit11: 0  bit12: 0
bit21: 0  bit22: 0
Sum1:  0.0  Sum2:  0.0  Carry:  0.0
bit11: 0  bit12: 0
bit21: 0  bit22: 1
Sum1:  0.0  Sum2:  1.0  Carry:  0.0
bit11: 0  bit12: 0
bit21: 1  bit22: 0
Sum1:  1.0  Sum2:  0.0  Carry:  0.0
bit11: 0  bit12: 0
bit21: 1  bit22: 1
Sum1:  1.0  Sum2:  1.0  Carry:  0.0
bit11: 0  bit12: 1
bit21: 0  bit22: 0
Sum1:  0.0  Sum2:  1.0  Carry:  0.0
bit11: 0  bit12: 1
bit21: 0  bit22: 1
Sum1:  0.0  Sum2:  0.0  Carry:  1.0
bit11: 0  bit12: 1
bit21: 1  bit22: 0
Sum1:  1.0  Sum2:  1.0  Carry:  0.0
bit11: 0  bit12: 1
bit21: 1  bit22: 1
Sum1:  1.0  Sum2:  0.0  Carry:  1.0
```

```
bit11: 1  bit12: 0
bit21: 0  bit22: 0
Sum1:  1.0  Sum2:  0.0  Carry:  0.0
bit11: 1  bit12: 0
bit21: 0  bit22: 1
Sum1:  1.0  Sum2:  1.0  Carry:  0.0
bit11: 1  bit12: 0
bit21: 1  bit22: 0
Sum1:  0.0  Sum2:  1.0  Carry:  0.0
bit11: 1  bit12: 0
bit21: 1  bit22: 1
Sum1:  0.0  Sum2:  0.0  Carry:  1.0
bit11: 1  bit12: 1
bit21: 0  bit22: 0
Sum1:  1.0  Sum2:  1.0  Carry:  0.0
bit11: 1  bit12: 1
bit21: 0  bit22: 1
Sum1:  1.0  Sum2:  0.0  Carry:  1.0
bit11: 1  bit12: 1
bit21: 1  bit22: 0
Sum1:  0.0  Sum2:  0.0  Carry:  1.0
bit11: 1  bit12: 1
bit21: 1  bit22: 1
Sum1:  0.0  Sum2:  1.0  Carry:  1.0
```

و می‌بینیم که شبکه برای تمام حالت‌ها درست کار می‌کند

سؤال ۲:

در این سؤال با استفاده از الگوریتم preceptron آن را باری ۲ بار اپدیت حل کنیم
این حل در زیر آمده است«

$w_1 = 0.2 \quad w_2 = 0.7 \quad w_3 = 0.9 \quad b = -0.7$

$x_1 = 0 \quad x_2 = 0 \quad x_3 = 1 \quad t = -1$

$\alpha = 0.2$

① $net = W^k \times x^k + b = 0 + 0 + 0.9 - 0.7 = 0.2$

$h = \begin{cases} +1 & net \geq 0 \\ -1 & net < 0 \end{cases} \Rightarrow h_1 = 1$

$\Rightarrow error = t - h = -2 \rightarrow$ update $w_{new} = w_{old} + \alpha t x$

$w_{1new} = 0.2 + 0 \quad w_{2new} = 0.7 + 0$

$w_{3new} = 0.9 + 0.2 \times (-1) \times 1 = 0.7$

$b = -0.7 + (0.2 \times (-1)) = -0.9$

② $net = 0 + 0 + 0.7 - 0.9 = -0.2$

$\Rightarrow h = -1 \Rightarrow error = 0$

سؤال ۳:

در این سؤال ۲ سری داده, داریم و می‌خواهیم با استفاده از دو الگوریتم آن‌ها را از هم جدا کنیم
هر حالت را جدا حل می کنیم:

حالت الف ـ ۲۰۰ داده متقارن

در این حالت ابتدا داده‌ها را درست می کنیم:

```
# First Dataset 200 Point Equal 100 vs 100
    # Uncomment For First Part
valuesOfPositive = 1 + 0.5 * np.random.normal(0, 1, (2, 100))
valuesOfNegative = -1 + 0.5 * np.random.normal(0, 1, (2, 100))
```

سپس آن‌ها را در نمودار گذاشته و همه یانها را به همراه مقدار مورد نظر جواب در یک dataset می گذاریم:

```
plt.plot(valuesOfPositive[0], valuesOfPositive[1], "bo")
plt.plot(valuesOfNegative[0], valuesOfNegative[1], "ro")
plt.grid()
# plt.show()
dataset = []
for j in range(len(valuesOfPositive[0])):
    dataset.append([valuesOfPositive[0][j], valuesOfPositive[1][j], 1])

for j in range(len(valuesOfNegative[0])):
    dataset.append([valuesOfNegative[0][j], valuesOfNegative[1][j], -1])
```

و سپس یک بار آن را به کمک الگوریتم preceptron جدا می کنیم:
در حالت learning rate را ۰.۲ گذاشته‌ایم و برای ۵ epoch داریم:

```
l_rate = 0.2
n_epoch = 5
weights = train_weights_Preceptron(dataset, l_rate, n_epoch)
# weights = train_weights_Adaline(dataset, l_rate, n_epoch)
print(weights)
x = []
y = []
for i in range(len(dataset)):
    x.append((-weights[2]/weights[1]) * dataset[i][1] - weights[0])
    y.append(dataset[i][1])

plt.plot(y, x)
plt.show()
```

بعد از اجرای این کد نمودار به شکل زیر جدا می شود:

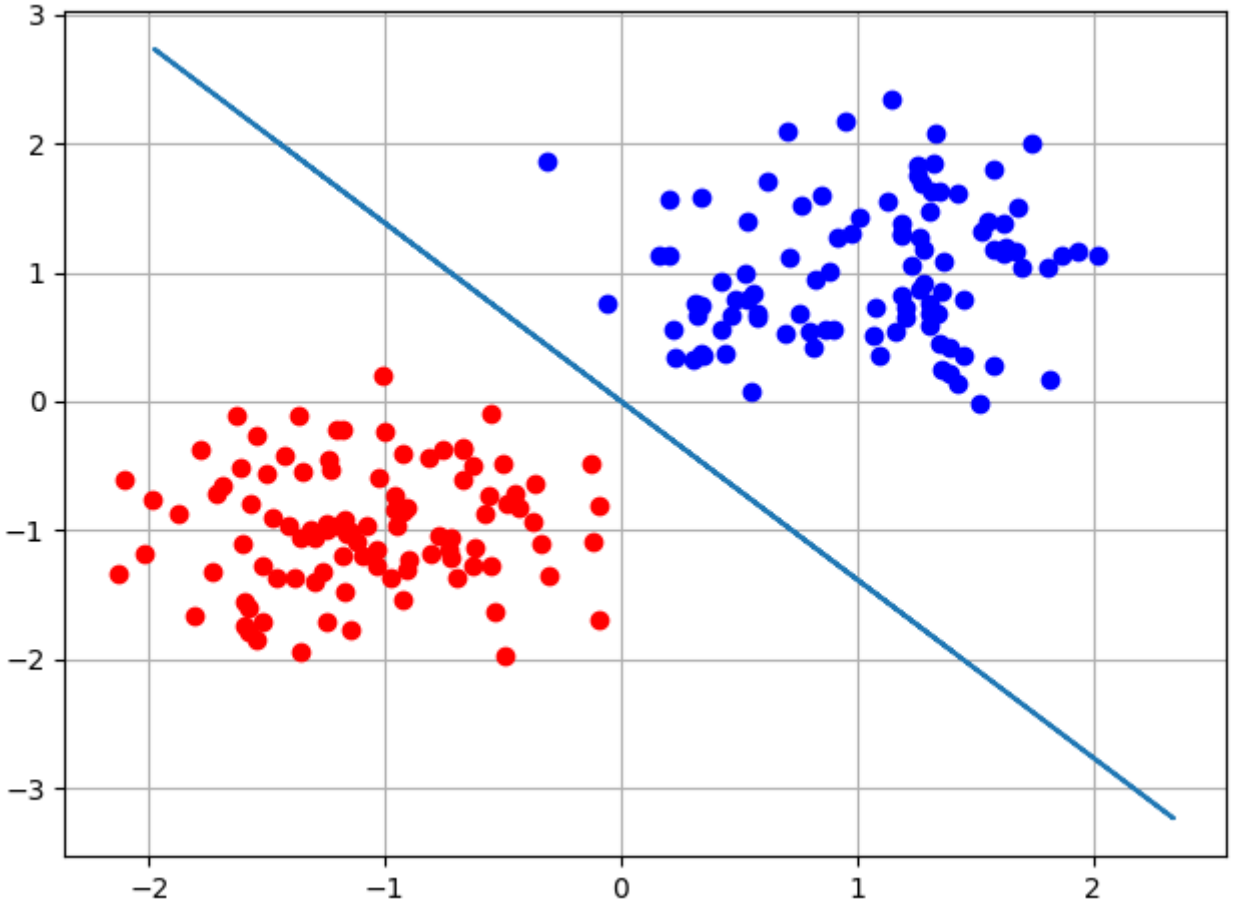


Illustration 1: Preceptron_Part1_Rate: 0.2

```
>epoch=0, lrate=0.200, error=4.000
>epoch=1, lrate=0.200, error=4.000
>epoch=2, lrate=0.200, error=0.000
>epoch=3, lrate=0.200, error=0.000
>epoch=4, lrate=0.200, error=0.000
[0.0, 0.35704376232739704, 0.4933275662714721]
```

همان‌طور که دیده می‌شود در ۳ epoch ارور صفر شده است و خط به درستی کشیده شده است

اگر learning rate را زیاد کنیم و به ۱ برسانیم داریم:

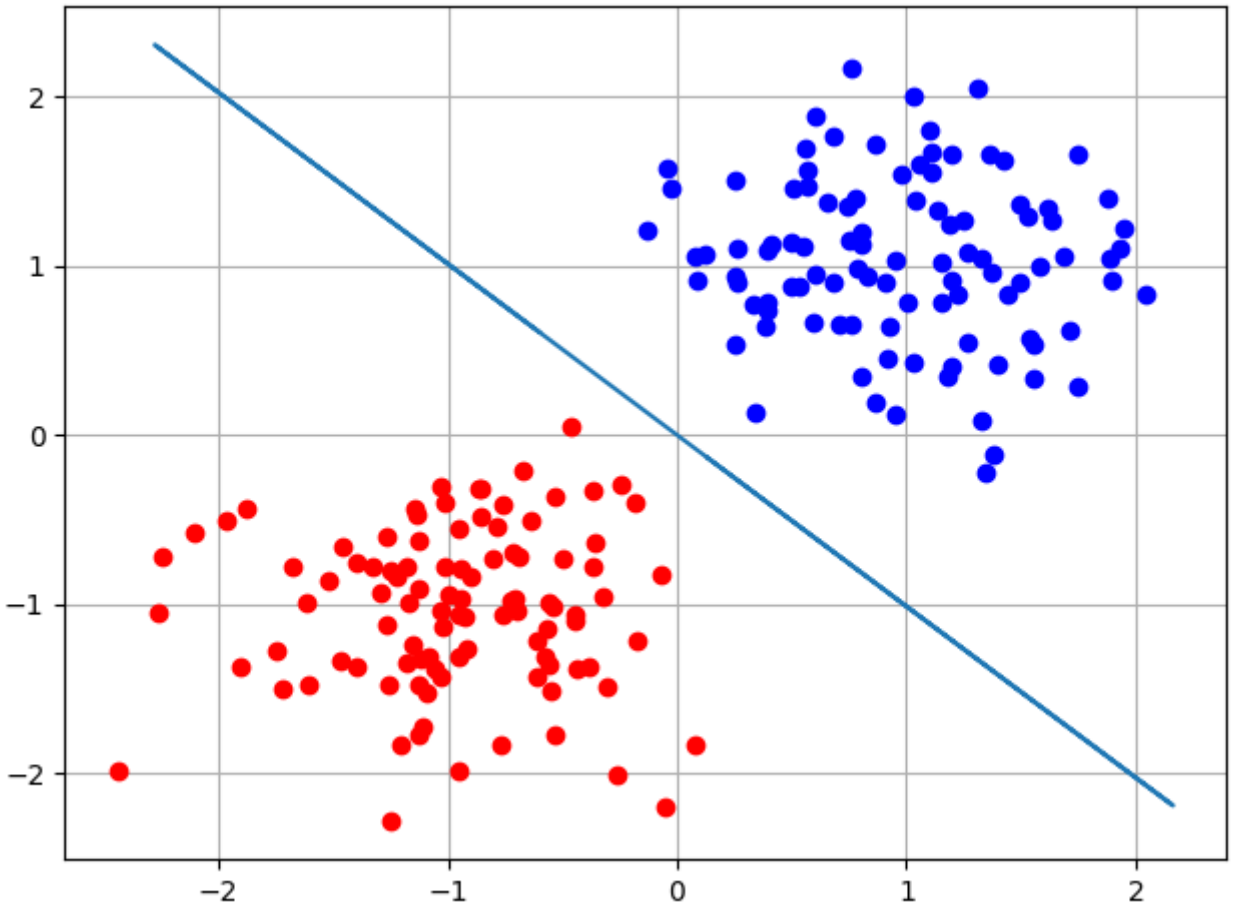


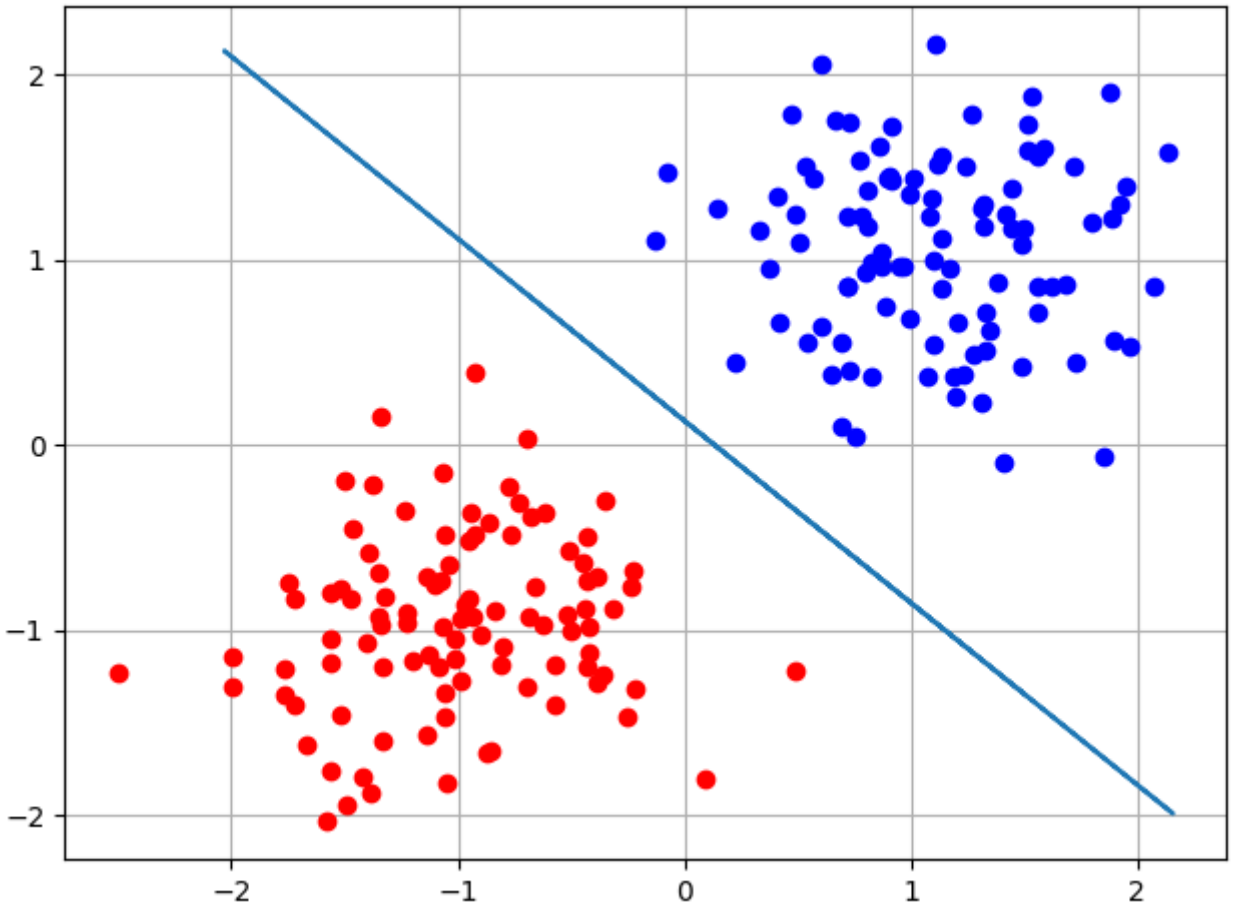
Illustration 2: Preceptron_Part1_Rate: 1

```
>epoch=0, lrate=1.000, error=4.000
>epoch=1, lrate=1.000, error=4.000
>epoch=2, lrate=1.000, error=0.000
>epoch=3, lrate=1.000, error=0.000
>epoch=4, lrate=1.000, error=0.000
[0.0, 1.4284673578192193, 1.4450876254614489]
```

که همان‌طور که دیده می‌شود باز هم جدا شده‌اند و باز ۲ epoch کافی بوده است

حال همین داده‌ها را با الگوریتم adaline انجام می‌دهیم و داریم: learning rate را ابتدا 0.01 گذاشته‌ایم و برای epoch 5 اینکار را انجام می دهیم:

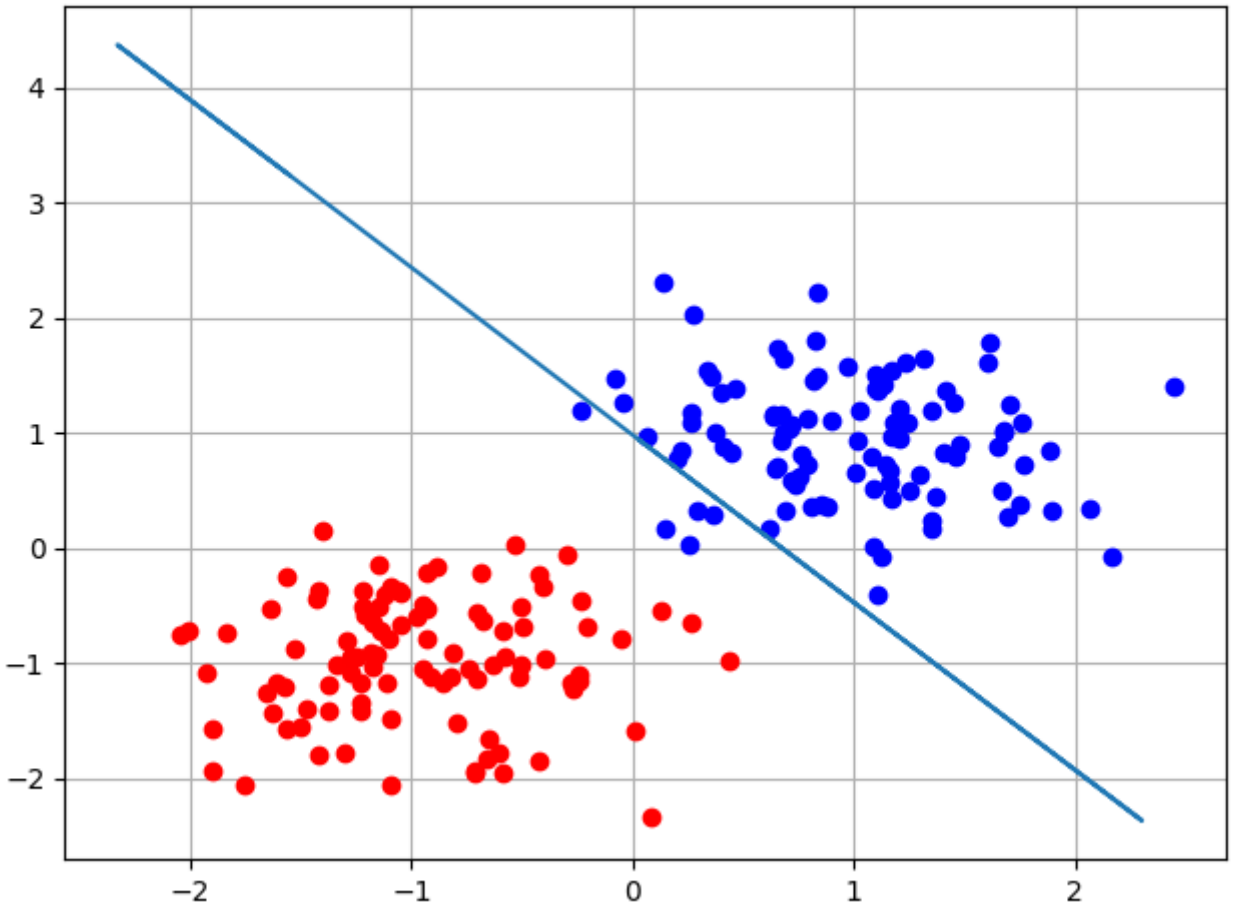
```
l_rate = 0.01
n_epoch = 10
# weights = train_weights_Preceptron(dataset, l_rate, n_epoch)
weights = train_weights_Adaline(dataset, l_rate, n_epoch)
print(weights)
x = []
y = []
for i in range(len(dataset)):
    x.append((-weights[2]/weights[1]) * dataset[i][1] - weights[0])
    y.append(dataset[i][1])

plt.plot(y, x)
plt.show()
```



همان‌طور که دیده می‌شود با این learning rate نقاط به درستی از هم جدا می‌شوند
حال اگر learning rate را به ۰.۵ افزایش دهیم داریم:

```
>epoch=0, lrate=0.500, error=-1.987
>epoch=1, lrate=0.500, error=0.000
>epoch=2, lrate=0.500, error=-0.000
>epoch=3, lrate=0.500, error=0.000
>epoch=4, lrate=0.500, error=-0.000
[-0.9933197433335279, -0.004557106302410465, -0.011032051957513948]
```

می‌بینیم به اینکه خطا کم شده است ولی نمودار :



و نتوانسته است جدا کند زیرا سریع تغییر پیدا کرده و فرصت کافی نداشته است

حالت ب ـ ۱۱۰ داده غیر متقارن

در این حالت نیز ابتدا داده‌های خود را درست می‌کنیم می‌دانیم این داده‌ها متقارن نیستند.

```
valuesOfPositive = 1 + 0.5 * np.random.normal(0, 1, (2, 100))
valuesOfNegative = -1 + 0.5 * np.random.normal(0, 1, (2, 10))
```

سپس ابتدا الگوریتم Perceptron را با 0.1 learning rate اجرا می‌کنیم و داریم:

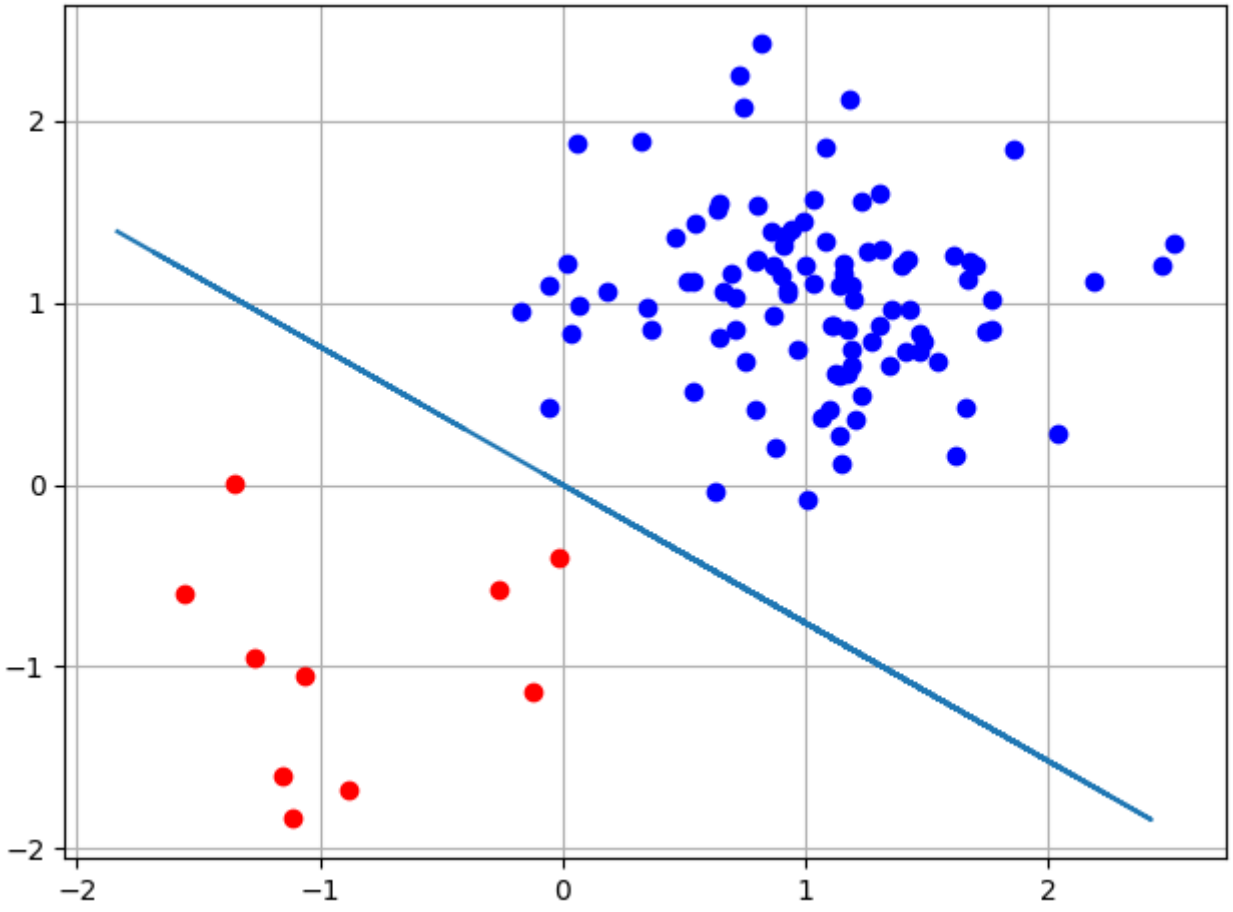


Illustration 3: Preceptron_Part2_Rate=0.1

و در آن داریم:

```
>epoch=0, lrate=0.100, error=4.000
>epoch=1, lrate=0.100, error=4.000
>epoch=2, lrate=0.100, error=0.000
>epoch=3, lrate=0.100, error=0.000
>epoch=4, lrate=0.100, error=0.000
[0.0, 0.1892975743994434, 0.14385062340762225]
```

می‌بینیم که در epoch ۲ به جواب رسیده است
حال اگر learning rate را به ۰.۵ تغییر دهیم داریم:

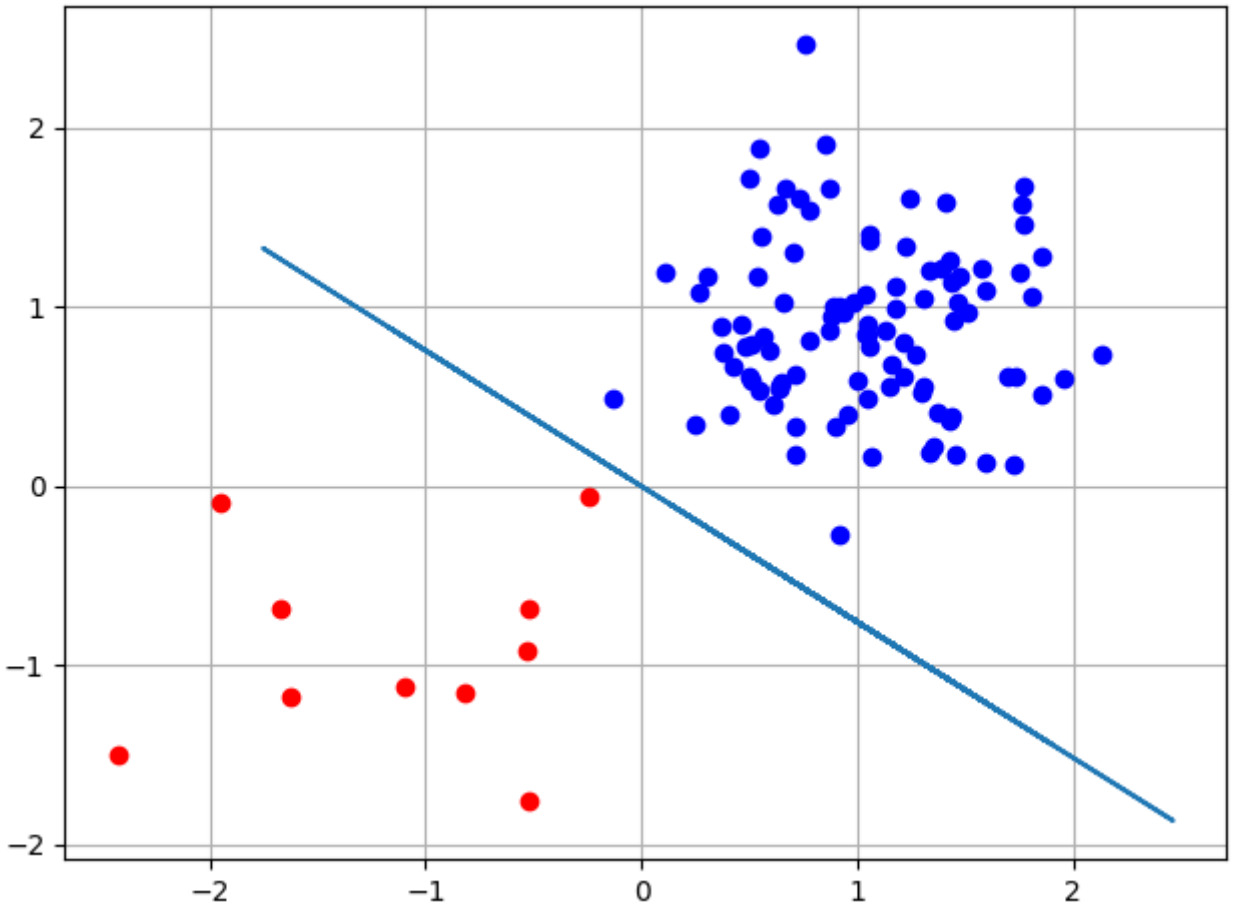


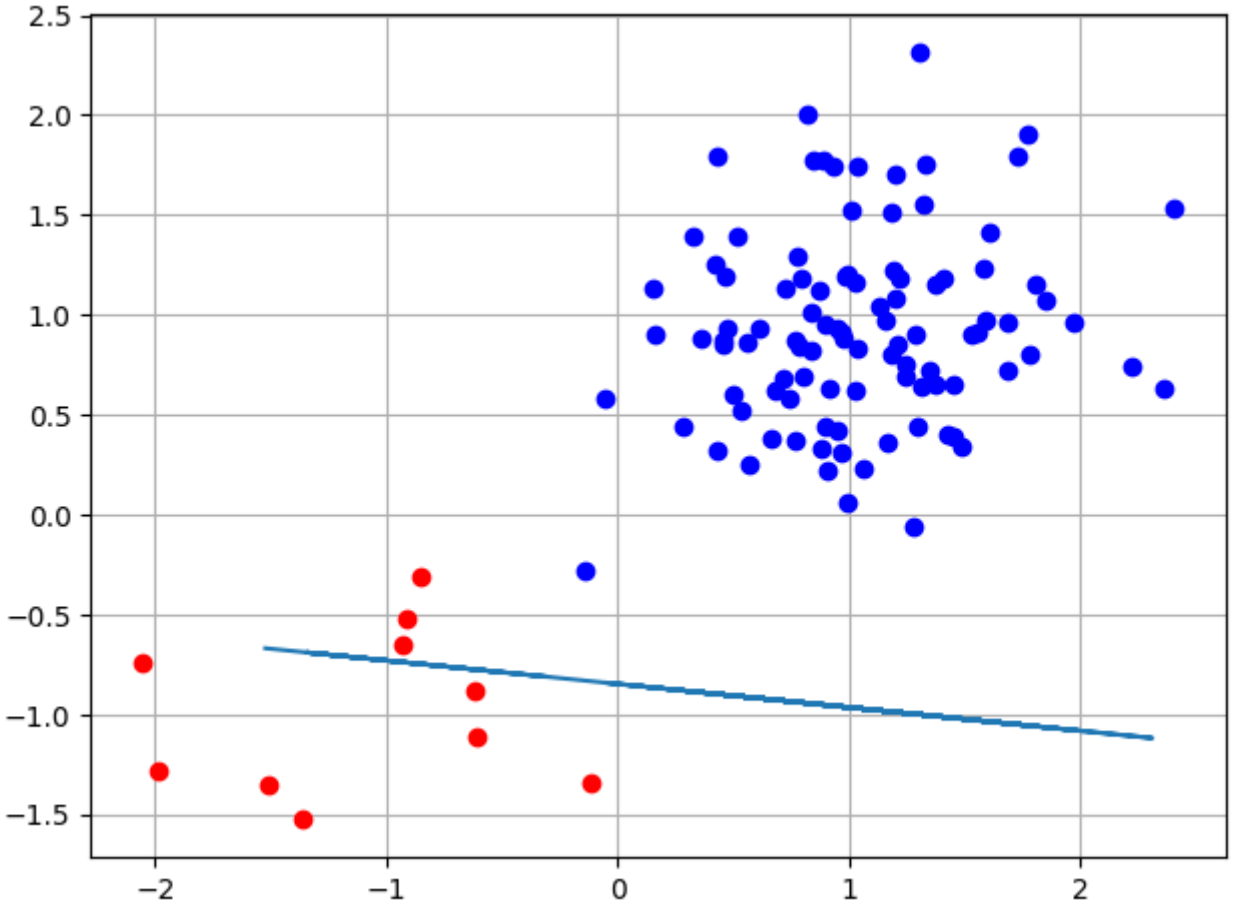
Illustration 4: Perceptron_Part2_Rate=0.5

و

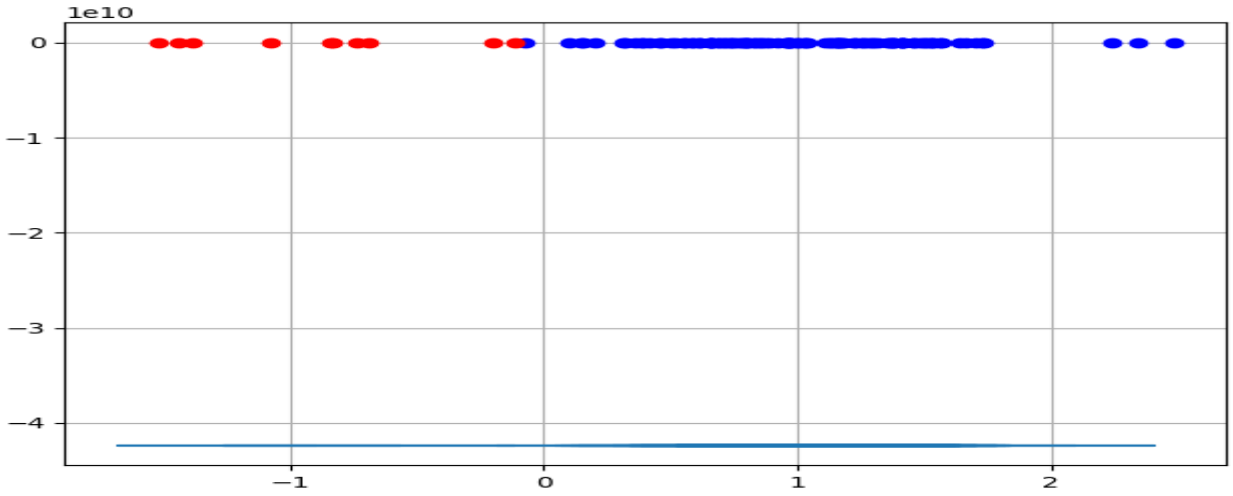
```
>epoch=0, lrate=0.500, error=4.000
>epoch=1, lrate=0.500, error=4.000
>epoch=2, lrate=0.500, error=0.000
>epoch=3, lrate=0.500, error=0.000
>epoch=4, lrate=0.500, error=0.000
[0.0, 0.7687379668211181, 0.5824300911936228]
```

که می‌بینیم باز هم موفق به جدا کردن نقاط شده است

حا همین کار را با adeline انجام می‌دهیم و داریم:
ابتدا با ۰.۱ learning rate اینکار را انجام می دهیم:



می‌بینیم که این الگوریتم موفق به جدا شده آن‌ها نشده است
و اگر این learning rate را به ۰.۶ زیاد کنیم داریم:



می‌بینیم که این الگوریتم در صورتی که تعداد نقاط متقارن نباشند نمی‌تواند نقاط را درست جدا کند

برای الگوریتم ها داریم؛
الگوریتم Preceptron را مانند زیر درست کرده ایم:

```python
# Make a prediction with weights
def predict_h(row, weights):
    activation = weights[0]
    for i in range(len(row) - 1):
        activation += weights[i + 1] * row[i]
    return 1.0 if activation >= 0.0 else -1.0

# Estimate Perceptron weights using stochastic gradient descent
def train_weights_Preceptron(train, l_rate, n_epoch):
    weights = [0.0 for i in range(len(train[0]))]
    for epoch in range(n_epoch):
        sum_error = 0.0
        for row in train:
            prediction = predict_h(row, weights)
            error = row[-1] - prediction
            if error != 0:
                # print(">epoch=%d, prediction=%d, error=%d, expected=%d" % (epoch, prediction, error, row[-1]))
                sum_error += error ** 2
                weights[0] = weights[0] + l_rate * row[-1]
                for i in range(len(row) - 1):
                    weights[i + 1] = weights[i + 1] + l_rate * row[-1] * row[i]
        print('>epoch=%d, lrate=%.3f, error=%.3f' % (epoch, l_rate, sum_error))
    return weights
```

و برای الگوریتم Adeline داریم:

```python
def predict_net(row, weights):
    activation = weights[0]
    for i in range(len(row) - 1):
        activation += weights[i + 1] * row[i]
    return activation

def train_weights_Adaline(train, l_rate, n_epoch):
    weights = [0.0 for i in range(len(train[0]))]
    for epoch in range(n_epoch):
        sum_error = 0.0
        for row in train:
            prediction = predict_net(row, weights)
            error = row[-1] - prediction
            if error != 0:
                # print(">epoch=%d, prediction=%d, error=%d, expected=%d" % (epoch, prediction, error, row[-1]))
                sum_error += error
                weights[0] = weights[0] + l_rate * error
                for i in range(len(row) - 1):
                    weights[i + 1] = weights[i + 1] + l_rate * error * row[i]
        print('>epoch=%d, lrate=%.3f, error=%.3f' % (epoch, l_rate, sum_error))
    return weights
```

مقایسه:

با کار های انجام شده دیده می‌شود که الگوریتم adeline وقتی تعداد نقاط متقارن و برابر نیست نمی‌تواند به درستی خطوط را جدا کند
این درحالی است که Preceptron موفق به انجام آن در هر learning rate ای شده است

الگوریتم Adeline برای رسیدن به جواب نهایی نیاز به تعداد epoch بیشتری دارد و دقیق‌تر وبهتر است زیرا خطا در آن صفر کامل نمی‌شود و هر سری وزن ها دقیق‌تر می‌شود ولی Preceptron برای این کار سریع‌تر به نتیجه می‌رسد زیرا خطای آن صفر می‌شود و دیگر وزن ها اپدیت نمی‌شوند و وزن های آن از Adeline ضعیف‌تر است

با زیاد کردن Learning Rate الگوریتم Adeline دچار خطا می‌شود زیرا می‌خواهد سریع به جواب برست ولی نمی‌شود پس در این الگوریتم نمیتوان rate را زیاد گذاشت و با زیاد کردن آن جواب غیر دقیق‌تر می‌شود و خط کشیده شده به نقاط نزدیک می‌شود و حتی ممکن است خراب شود
در الگوریتم Preceptron با زیاد کردن rate باز هم خط درست پیدا می‌شود ولی فاصله آن از یک سری از داده‌ها کم می‌شود یعنی robustness شبکه کم شده و دقت آن پایین می رود