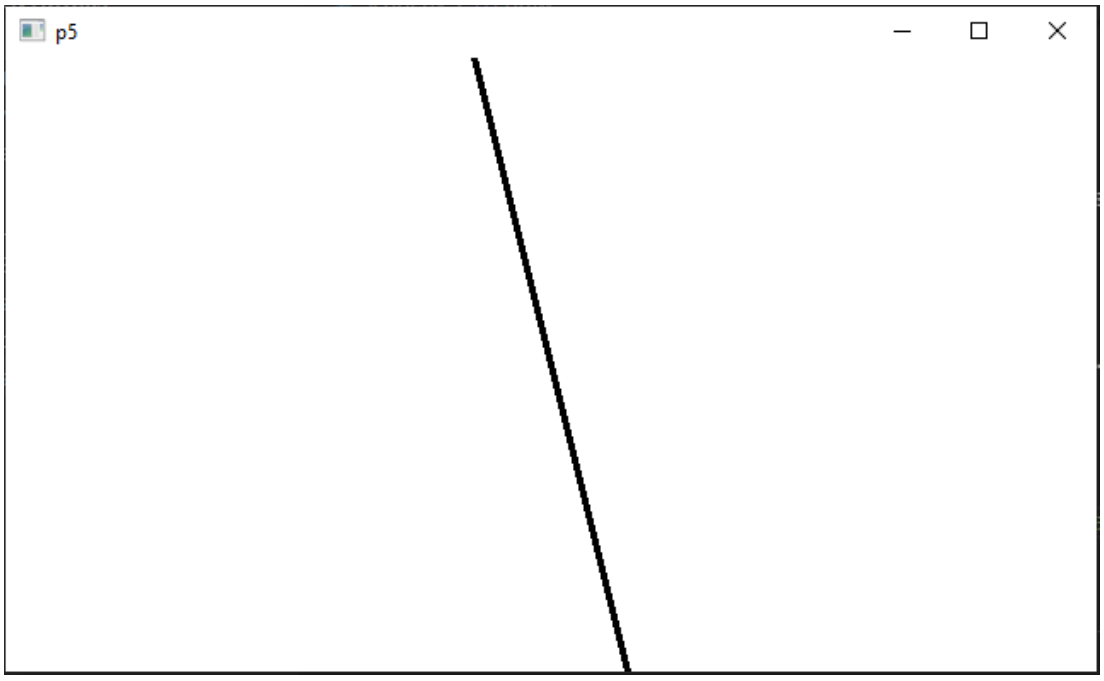
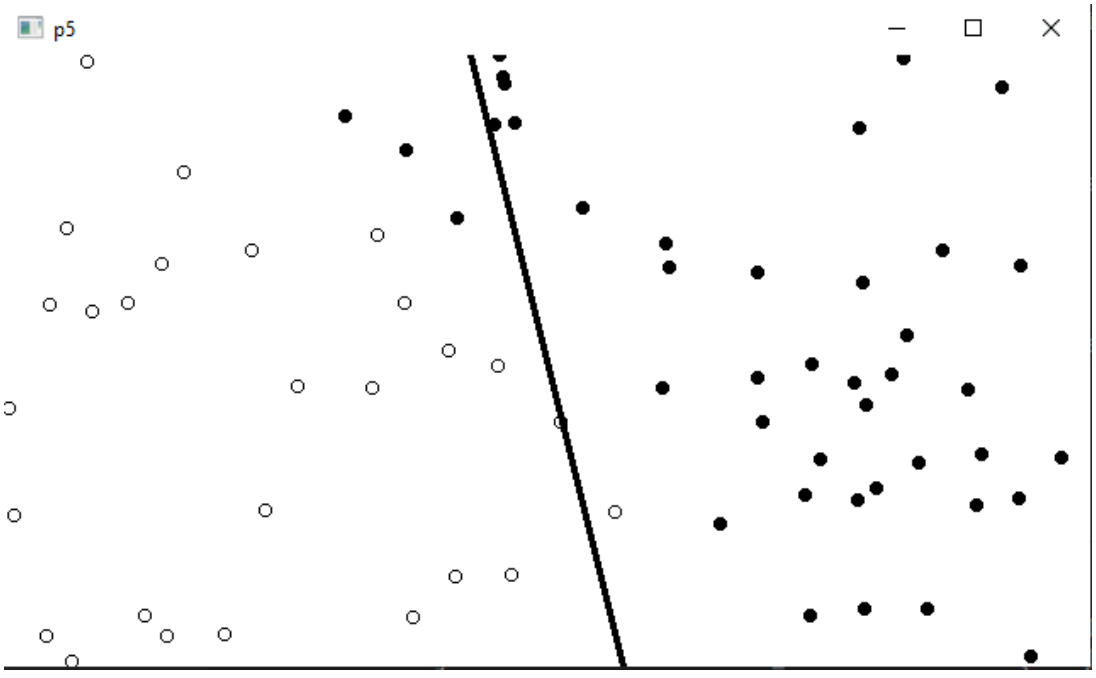
Perceptron

Para este projeto, vamos criar uma rede neural artificial bastante simples, que será composta de um único neurônio, um perceptron. O objetivo do exercício é ensinar o neurônio a descobrir uma equação que separa nossa tela em duas metades, veja a imagem abaixo:



O único elemento dessa imagem é uma linha no meio que separa a nossa tela em duas metades, o nosso perceptron vai tentar separar uma série de pontos neste plano bidimensional entre pontos a esquerda e a direita da linha, mais ou menos como está na imagem abaixo, porém, observe que nesta imagem o perceptron está aprendendo a classificar os itens, e portanto, acaba por erroneamente marcar pontos escuros do lado esquerdo da linha, apesar do fato que estes deveriam estar sem preenchimento.



Bom, entendido o nosso problema, vamos começar a escrever código!

O perceptron

Como aprendemos na nossa aula sobre redes neurais, um perceptron é um elemento computacional que recebe um ou mais inputs, que quando multiplicados pelos weights correspondentes geram um output. Vamos descrever um pouco o nosso perceptron, e depois vamos ver o código resultante:

- Classe: Perceptron
  - Propriedades:
    1. weights, que são os pesos dos inputs;
    2. l_const que é a constante de aprendizado, ou o quão rápido o perceptron aprende;
  - Construtor (__init__):
    - No construtor vamos definir os pesos iniciais como um valor aleatório entre -1 e 1;
  - feed_forward(inputs):
    - Nesta função vamos calcular a soma dos inputs e passar o valor para a função activate;
  - activate(soma):
    - Aqui vamos retornar -1 se a soma dos inputs for negativa, +1 se for positiva;
  - train(inputs, resposta):
    - A função de treinamento do nosso perceptron, vamos passar alguns inputs e a resposta esperada;
    - Ajustamos os weights de acordo com a resposta;

O trecho abaixo apresenta o código fonte para o nosso perceptron respeitando a descrição acima.

```
import random
from p5 import *

class Perceptron:
  weights = []
  l_constant = 0.01

  def __init__(self, weights):
    self.weights = [random.uniform(-1, 1) for weight in range(weights)]

  def feed_forward(self, inputs = []):
    sum = 0
    for index, value in enumerate(self.weights):
      sum += inputs[index] * self.weights[index]
    return self.activate(sum)

  def activate(self, sum):
    result = -1 if sum < 0 else 1
    return result

  def train(self, inputs = [], desired = 0):
    guess = self.feed_forward(inputs)
    error = desired - guess
    for index, weight in enumerate(self.weights):
      self.weights[index] += self.l_constant * error * inputs[index]
```

Treinamento

Para treinar o nosso perceptron, vamos precisar criar uma classe para abrigar os nossos itens de treinamento, ou melhor dizendo, as nossas observações. Esta classe vai receber os inputs de treino, e um valor de resposta esperado, vamos utilizar essa resposta para verificar se nosso perceptron acertou ou não, e com isso, ajustar os pesos de cada input.

```
class TrainingItem:
  inputs = []
  answer = 0

  def __init__(self, x, y, a):
    self.inputs = [x, y, 1]
    self.answer = a
```

A função alvo

Como dito anteriormente, o objetivo do exercício é treinar um perceptron para identificar uma função matemática, vamos definir essa função matemática da seguinte forma:

```
def f(x):
        return 4*x
```

Uma função matemática simples, que será utilizada para desenhar a linha que separa os dois campos do nosso plano cartesiano

Visualizando os resultados

Agora sim, vamos utilizar a biblioteca de desenho p5 para abrir uma janela e desenhar a linha e os pontos no nosso plano cartesiano, mas antes disso, precisamos definir algumas variáveis importantes:

- O nosso perceptron, que vamos chamar de p1;
- A largura da janela;
- A altura da janela;
- O set de treinamento, que consistirá de 2000 itens;
- A quantidade de vezes que o nosso perceptron já foi treinado;

O código que atende essas 5 variáveis está descrito abaixo:

```
p1 = Perceptron(3)
width = 640
height = 360
training_set = [1 for i in range(500)]
training_count = 0
```

P

ara que a biblioteca p5 funcione corretamente, precisamos definir duas funções: setup e draw. Vamos começar com setup(), nesta função, precisamos definir o tamanho da janela que vamos abrir no nosso computador, vamos aproveitar para criar o nosso training set com itens de treinamento verdadeiros, já que inicializamos essa lista com 2000 números 1.

```
def setup():
  size(width, height)
  for i in range(500):
    x = random.randint(-width/2, width/2)
    y = random.randint(-height/2, height/2)
    answer = -1 if y < f(x) else 1
    training_set[i] = TrainintItem(x, y, answer)
```

Agora, vamos para a função draw(), que é executada frame a frame, atualizando o canvas com os desenhos que indicarmos dentro dela, é nesta função que vamos treinar o nosso neuronio, já que ela é executada 60 vezes por segundo. Vamos utilizar os métodos ellipse e line para desenhar os pontos e a linha de corte no plano bidimensional. No final, a função draw vai ficar desse jeito:

```
def draw():
  global training_count
  background(255)
  translate(width/2, height/2)
  p1.train(training_set[training_count].inputs, training_set[training_count].answer)
  training_count = (training_count + 1)
  for i in range(training_count):
    stroke(0)
    stroke_weight(1)
    guess = p1.feed_forward(training_set[i].inputs)
    if guess > 0:
      no_fill()
    else:
      fill(0)
    ellipse(training_set[i].inputs[0], training_set[i].inputs[1], 8, 8)
  stroke_weight(4)
  line(-640, f(-640), 640, f(640))
```

Para finalizar, basta chamar o método run, e pronto! Podemos observar nosso perceptron aprendendo a interpretar os dados e colorir os pontos de acordo com as suas observações.

```
run()
```

Ao final do projeto, o código fonte ficará da seguinte maneira:

```
import random
from p5 import *

class Perceptron:
  weights = []
  l_constant = 0.01

  def __init__(self, weights):
    self.weights = [random.uniform(-1, 1) for weight in range(weights)]

  def feed_forward(self, inputs = []):
    sum = 0
    for index, value in enumerate(self.weights):
      sum += inputs[index] * self.weights[index]
    return self.activate(sum)

  def activate(self, sum):
    result = -1 if sum < 0 else 1
    return result

  def train(self, inputs = [], desired = 0):
    guess = self.feed_forward(inputs)
    error = desired - guess
    for index, weight in enumerate(self.weights):
      self.weights[index] += self.l_constant * error * inputs[index]

class TrainintItem:
  inputs = []
  answer = 0

  def __init__(self, x, y, a):
```

```
        self.inputs = [x, y, 1]
        self.answer = a

def f(x):
    return 4*x

p1 = Perceptron(3)
width = 640
height = 360
training_set = [1 for i in range(500)]
training_count = 0

def setup():
    size(width, height)
    for i in range(500):
        x = random.randint(-width/2, width/2)
        y = random.randint(-height/2, height/2)
        answer = -1 if y < f(x) else 1
        training_set[i] = TrainintItem(x, y, answer)

def draw():
    global training_count
    background(255)
    translate(width/2, height/2)
    p1.train(training_set[training_count].inputs, training_set[training_count].answer)
    training_count = (training_count + 1)
    for i in range(training_count):
        stroke(0)
        stroke_weight(1)
        guess = p1.feed_forward(training_set[i].inputs)
        if guess > 0:
            no_fill()
        else:
            fill(0)
        ellipse(training_set[i].inputs[0], training_set[i].inputs[1], 8, 8)
    stroke_weight(4)
    line(-640, f(-640), 640, f(640))

run()
```

Finalizamos por aqui a nossa introdução as redes neurais artificiais, esperamos que você tenha aprendido bastante conosco, e estamos ansiosos para os próximos capítulos, até mais!