

# Dem4AI

Introdução à Inteligência Artificial

## 1. O que é Inteligência Artificial ?

- O que é?;
- Exemplos de aplicações;
- Exemplo de uma decisão;
- Fases da Inteligência Artificial;
- Neurónio Biológico x Artificial.

## 2. Perceptrão

- Conceitos de um Perceptrão;
- Erros do Sinal de Saída;
- Processo de Aprendizagem;
- Retropropagação (Backpropagation);
- Taxa de Aprendizagem;

## 3. Programa em Python

- Exemplo usando Programação (Python);
- Vídeo demonstrativo de uma rede neuronal.

# Capítulo 1.

---

O que é Inteligência  
Artificial ?

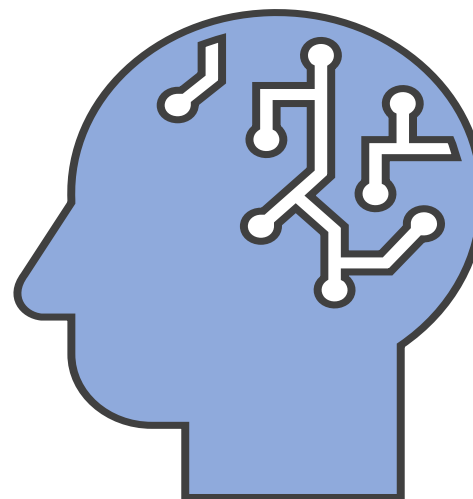
A Inteligência Artificial (IA) é a capacidade das máquinas executarem tarefas complexas que normalmente são desempenhadas por seres humanos.

O principal objetivo desta é executar funções de forma autónoma, recriando o raciocínio de tomada de decisão dos seres humanos através do uso de padrões, estatísticas e algoritmos.

O propósito da IA é resolver situações complexas, sem a necessidade da intervenção humana, facilitando e economizando o tempo.



Cérebro Humano



Inteligência Artificial

Usando como exemplo o xadrez, se um robô, sem qualquer programação ou percepção das jogadas normalmente utilizadas, assistir a diversas partidas diferentes, este irá encontrar sequências e definir padrões de forma a entender qual a melhor jogada a aplicar em cada situação.

Um exemplo onde se aplicou a Inteligência Artificial no xadrez, foi em maio de 1997, quando foi criado o supercomputador “Deep Blue”. Este, numa partida contra Garry Kasparov, o campeão mundial de xadrez na altura, conseguiu levar a vitória, tornando-se a primeira máquina a vencer um campeão de xadrez.



Deep Blue

**VS**



Garry Kasparov

Outro exemplo de utilização da Inteligência Artificial, que tem vindo a ser cada vez mais usada ultimamente, é a criação de imagens usando palavras.

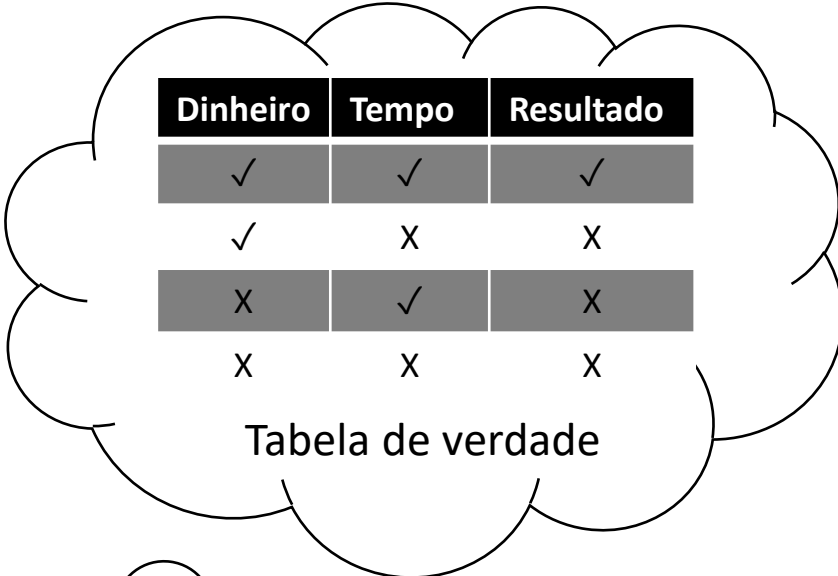
**Palavras utilizadas:**

Jogo de xadrez;  
Humano;  
Robô;  
Robô vs Humano.

**Midjourney**

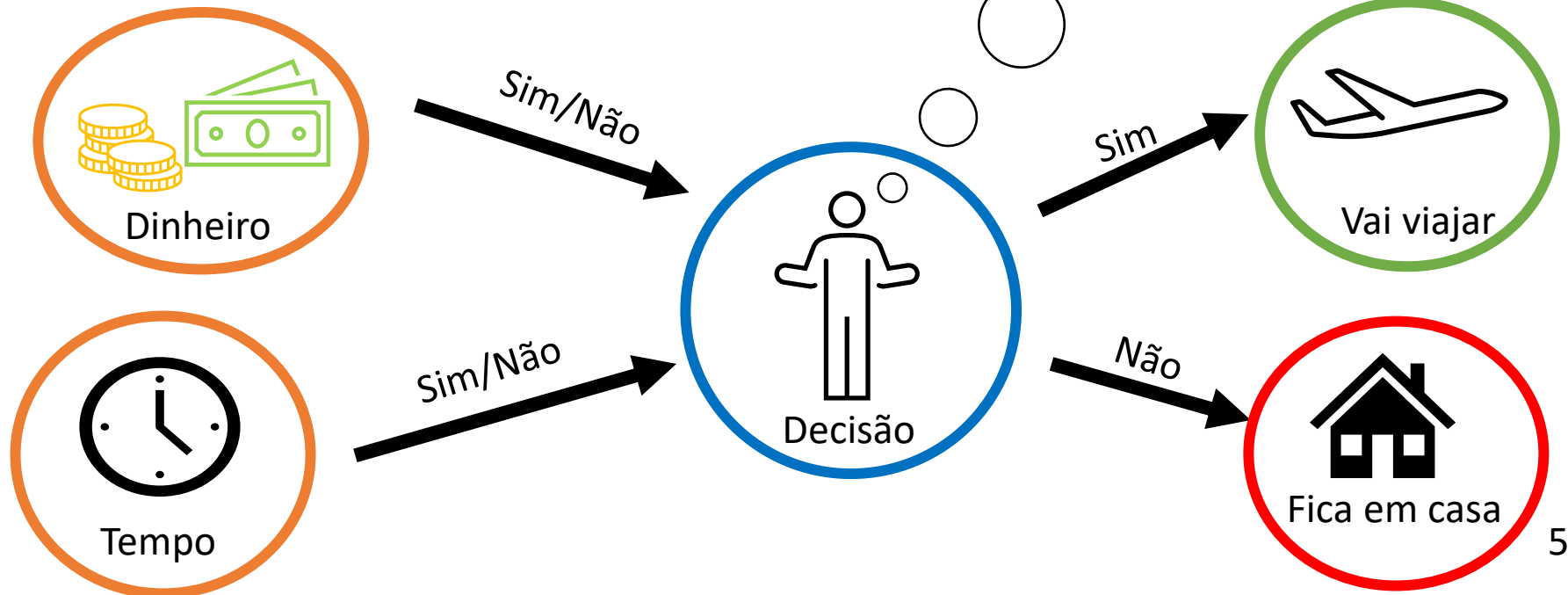


Tendo em conta o seguinte problema: Uma pessoa quer ir viajar para outro país, mas para tal necessita de dinheiro e tempo (entradas). Para tomarmos a decisão de viajar precisamos de saber se temos dinheiro e tempo para o podermos fazer. Se tivermos ambas então vamos viajar, se não tivermos uma delas, então não podemos fazê-lo.

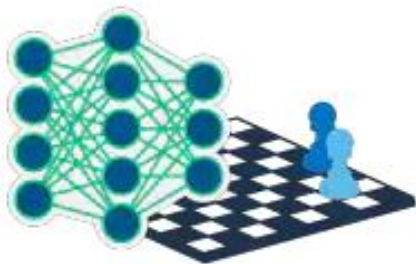


Dinheiro	Tempo	Resultado
✓	✓	✓
✓	X	X
X	✓	X
X	X	X

Tabela de verdade



**A Inteligência Artificial pode ser dividida em 3 categorias:**



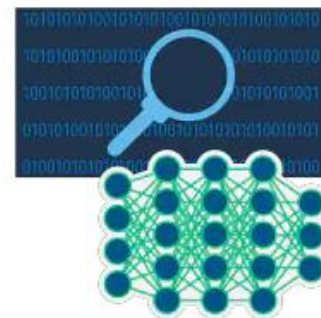
## **Inteligência Artificial**

Processa a informação e os dados e obtém uma resposta.



## **Machine Learning**

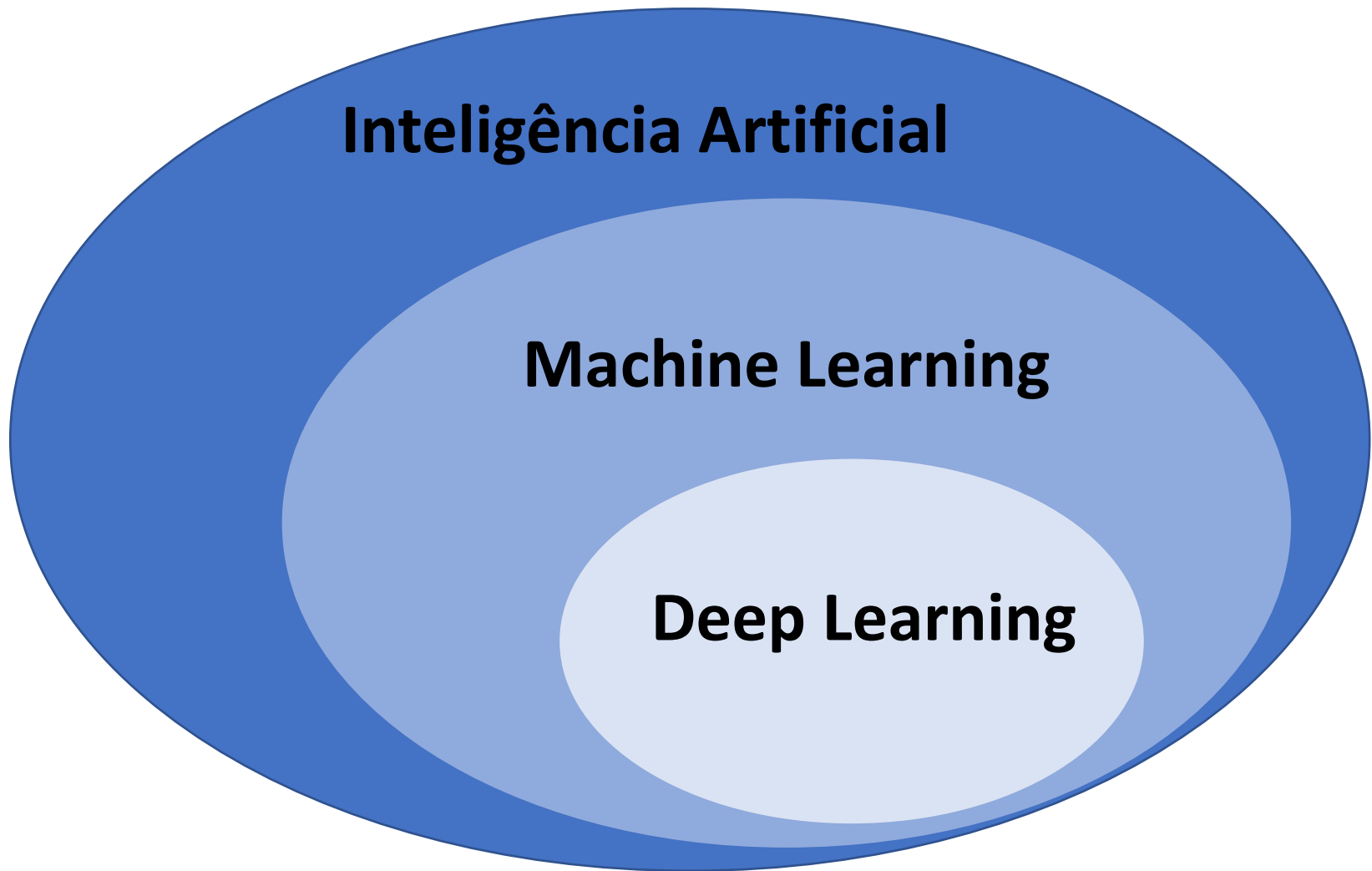
Reconhece padrões e aprende-os de forma a saber quando deve ser utilizada essa informação.



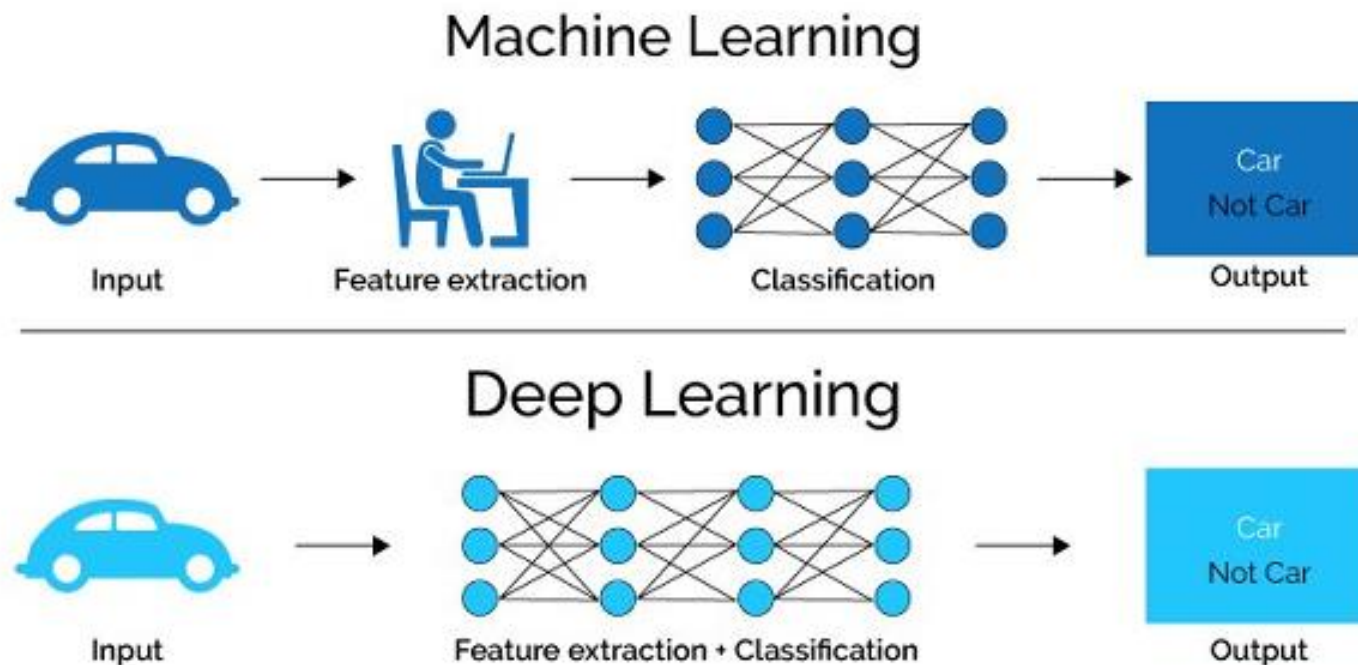
## **Deep Learning**

Prevê o que poderá acontecer no futuro.



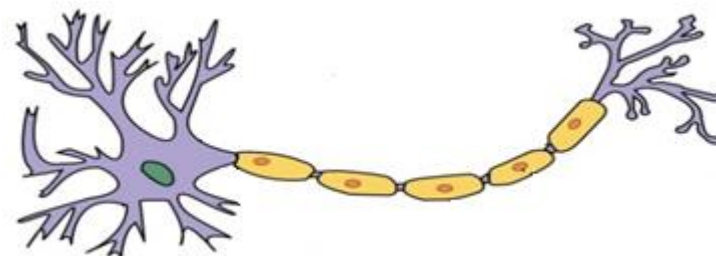


Na seguinte imagem, é possível perceber a diferença entre Machine Learning e Deep Learning:

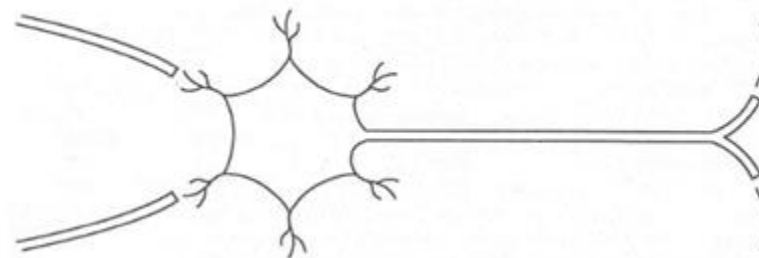


Uma das formas de comunicação dos neurónios biológicos é por impulsos nervosos, transmitindo a informação de um para os outros.

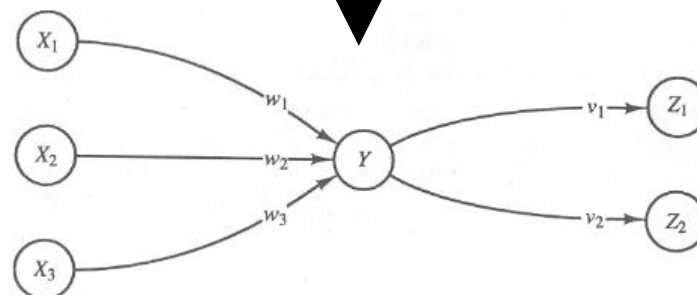
Os neurónio artificiais usam a mesma técnica mas neste caso o núcleo é “substituído” por um perceptrão.



Neurónio biológico



Misto



Neurónio artificial

# Capítulo 2.

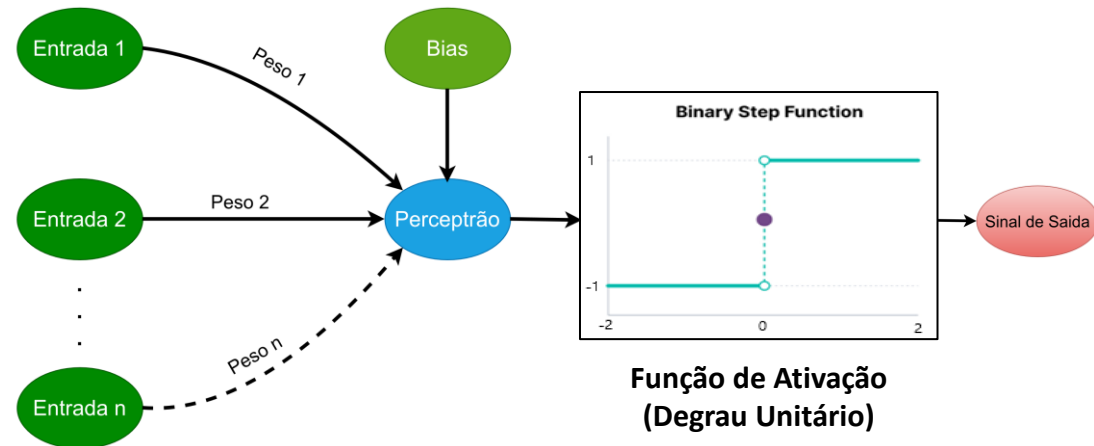
---

## Perceptrão

O Perceptrão é uma camada única de uma rede neuronal com uma saída binária.

Este consiste em 4 partes:

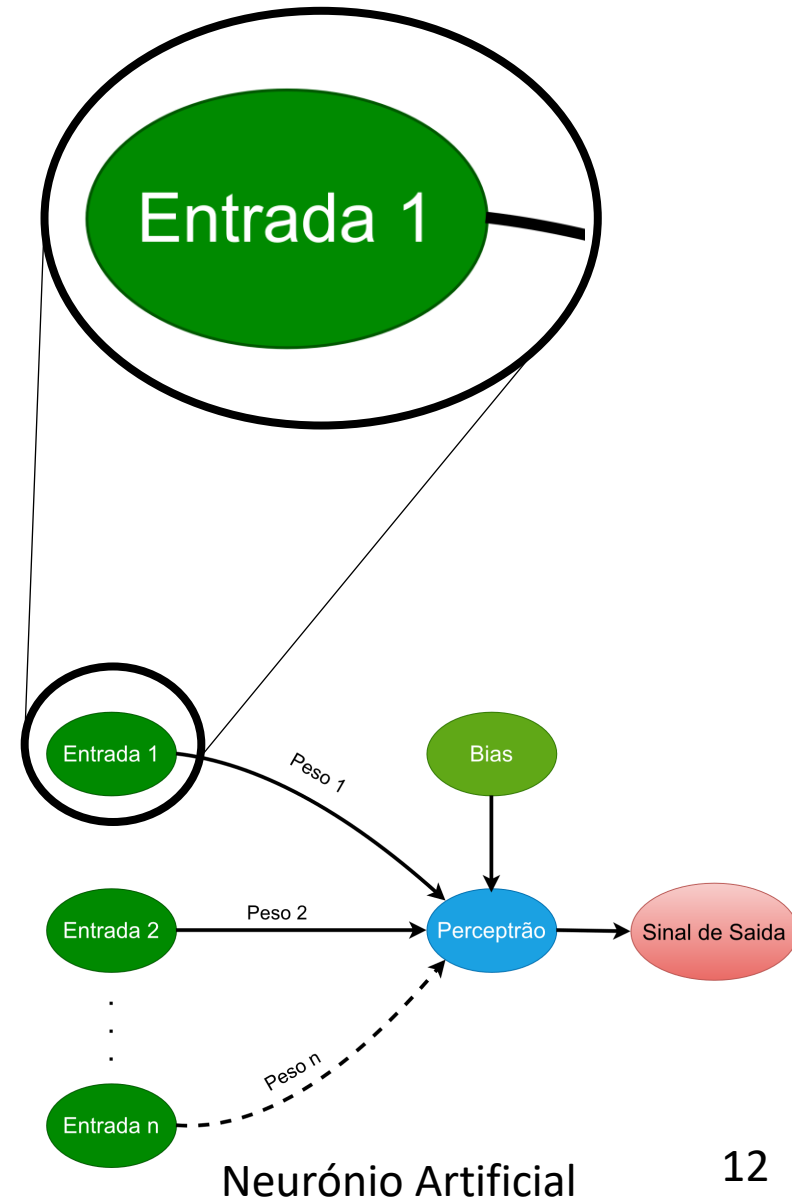
- Valores/camada de entrada;
- Pesos e bias;
- Soma Ponderada;
- Função de ativação.



Com diversas camadas de perceptrões, formamos uma Rede Neuronal.

Simplificando, as **entradas** são parâmetros/características que se relacionam com o que pretendemos classificar.

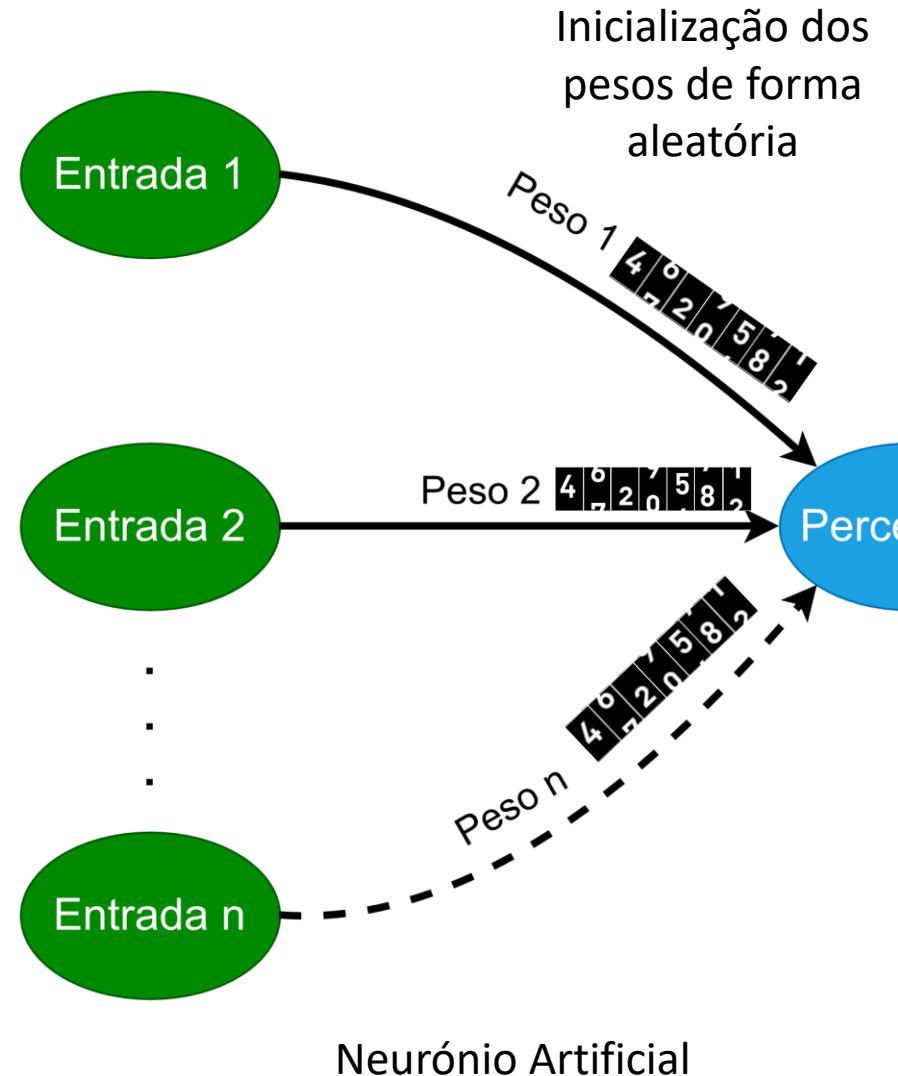
Um neurónio artificial pode ter uma ou diversas **entradas** dependendo da situação. Sendo que cada entrada vai ter uma importância (Peso) diferente.

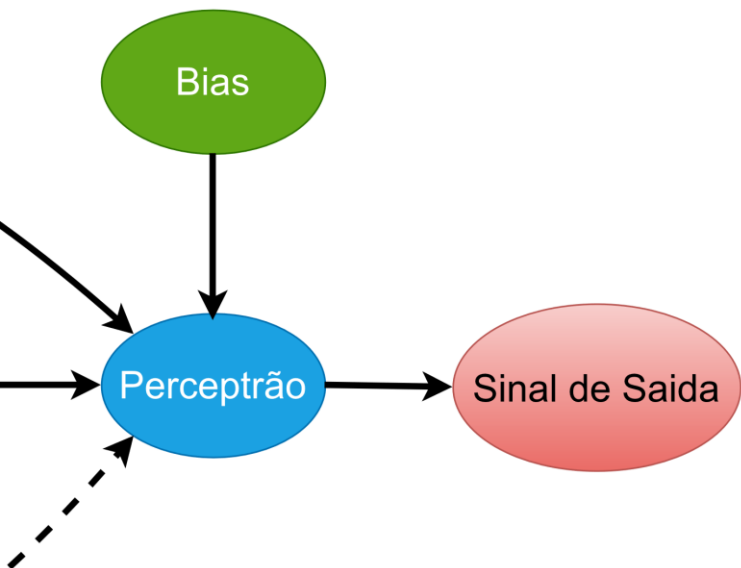


Como referido antes, cada entrada tem uma importância diferente para o resultado final, sendo que esta vai ser representada por um valor ao qual chamamos **peso**.

Estes **pesos** não tem um valor fixo, sendo diversas vezes alterado de forma a corrigir eventuais erros.

O **peso** não necessita de ser inteiro, podendo até ser negativo dependendo da situação.





Neurónio Artificial

O **Perceptrão** é a parte principal onde recebe o resultado final da **Entradas** e do **Pesos**.

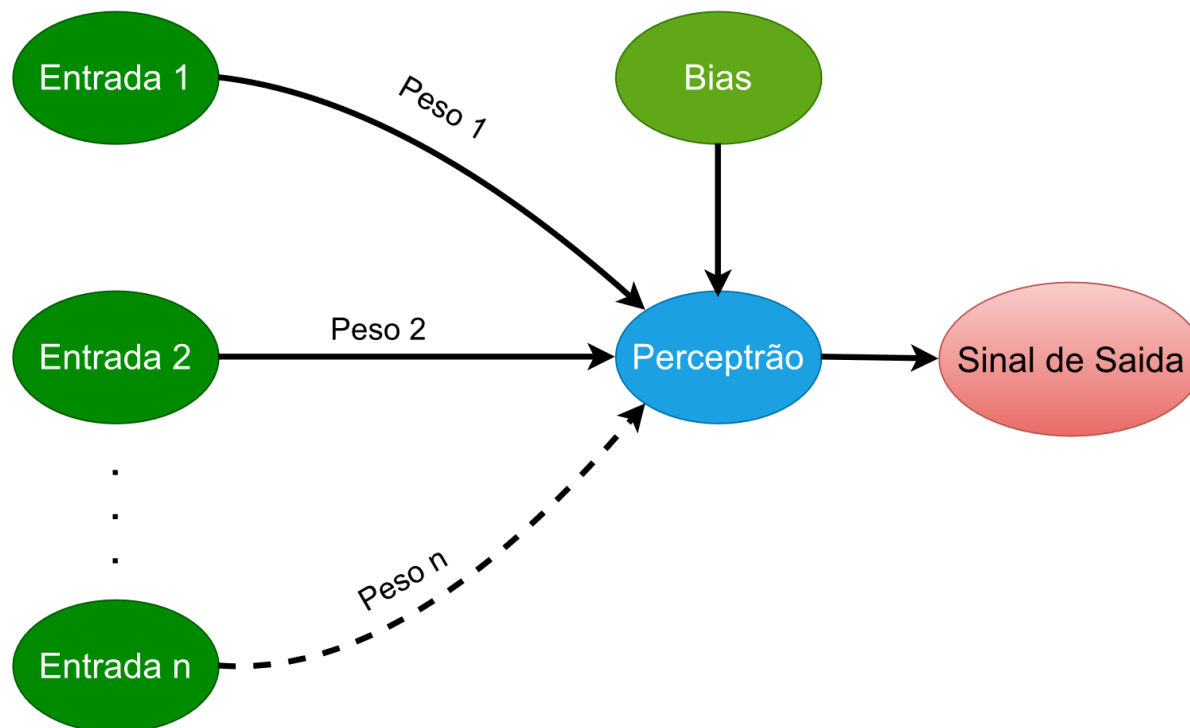
Mas não existe só um tipo de **Perceptrão**.

Por exemplo, este pode representar uma porta logica (AND ou OR), um comparador ou até mesmo um somador que é o mais usado.



O **Sinal de Saída** recebe o sinal do **Perceptrão** e a partir daí obtemos o resultado final.

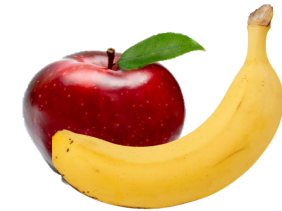
No caso de termos mais do que um **Perceptrão**, este primeiro pode ligar a outro com um peso associado a si, formando uma rede neuronal. Caso isso aconteça, o **Sinal de Saída** do primeiro passa a ser uma **entrada** do próximo **Perceptrão** e repete-se o processo.



Foi realizado um ensaio com 4 maçãs e 5 bananas escolhidas aleatoriamente. Retirando a massa em gramas e a forma atribuindo a classificação “1” à forma alongada e “0” à redonda.



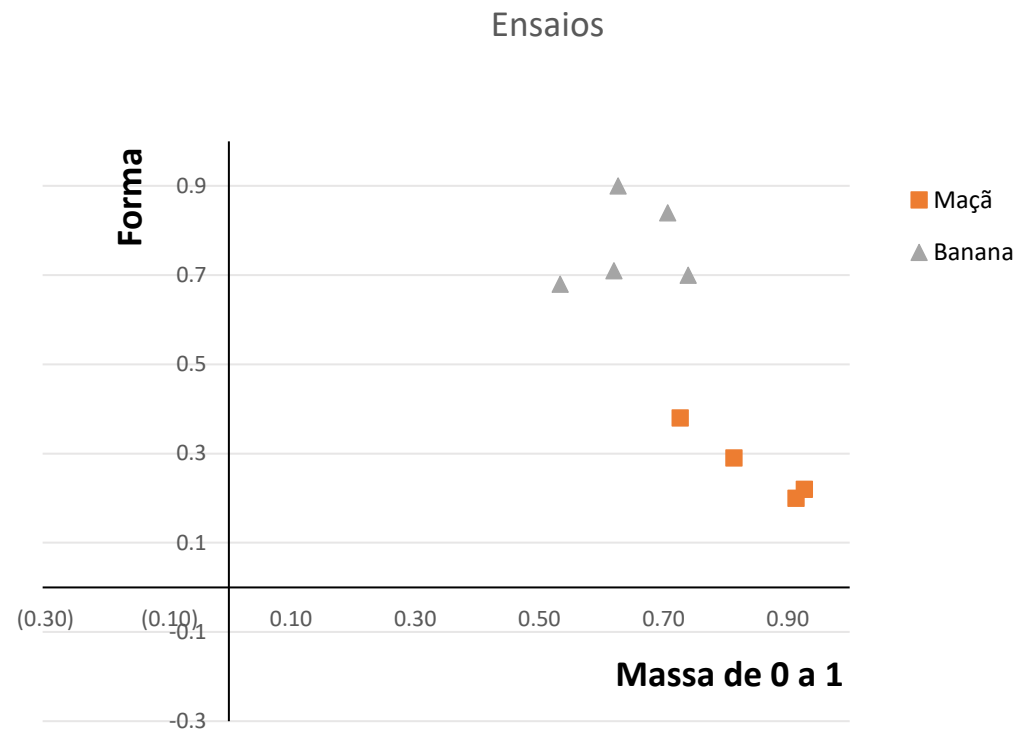
Os resultados obtidos no ensaio foram os seguintes:



Fruta	Massa (g)	Forma	Massa de 0 a 1
Maçã	139	0.22	0.93
Banana	111	0.7	0.74
Maçã	137	0.2	0.91
Banana	94	0.9	0.63
Banana	106	0.84	0.71
Maçã	109	0.38	0.73
Banana	80	0.68	0.53
Banana	93	0.71	0.62
Maçã	122	0.29	0.81



NOTA: 1 representa 150 gramas e 0 representa 0 gramas



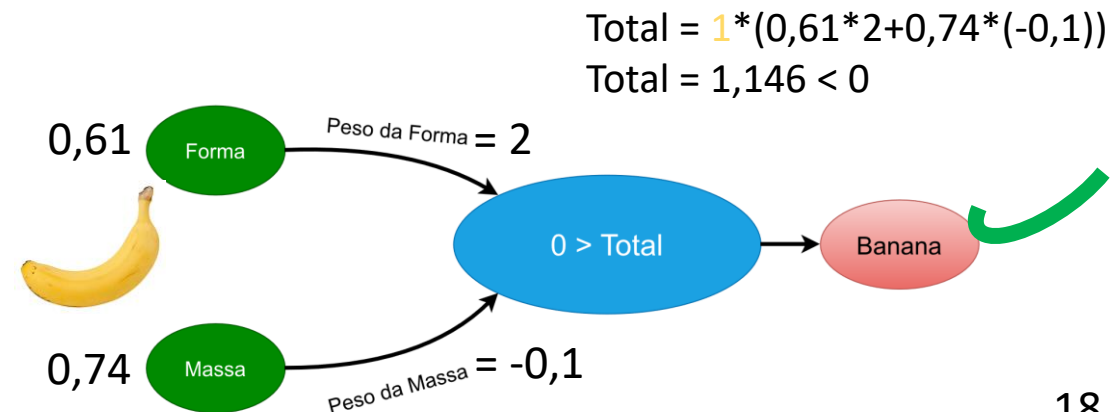
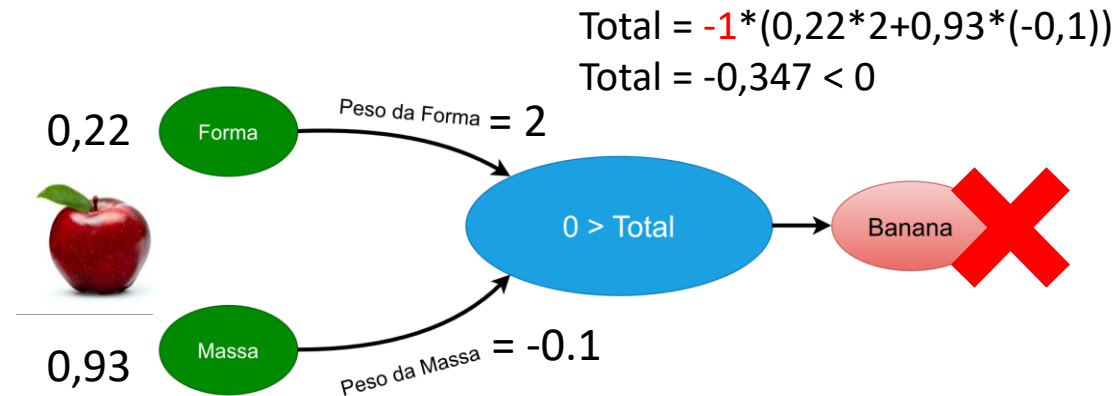
Através dos cálculos representados, é possível observar como o Perceptrão identifica se é banana ou se é maçã.

Os pesos da “Forma” e da “Massa” foram escolhidos aleatoriamente e é verificado se estão bem classificadas as amostras.

Assumindo que maçã é **-1** e banana é **1**.

Para verificar se os pesos estão corretos o total tem de ser superior a 0. (Explicação posteriormente)

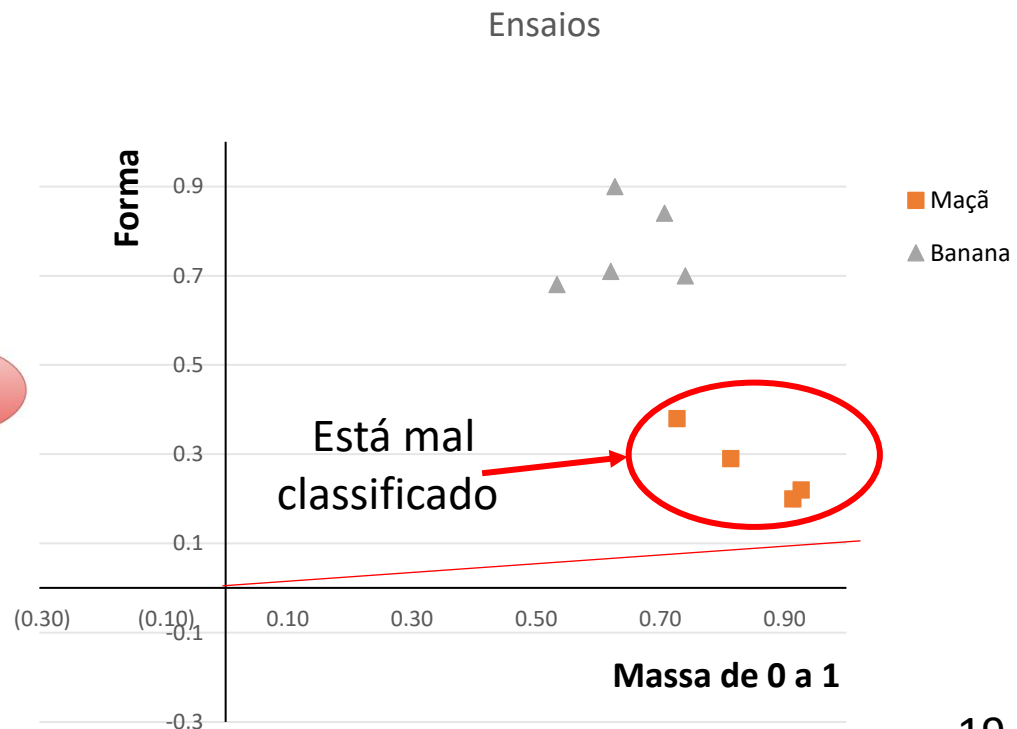
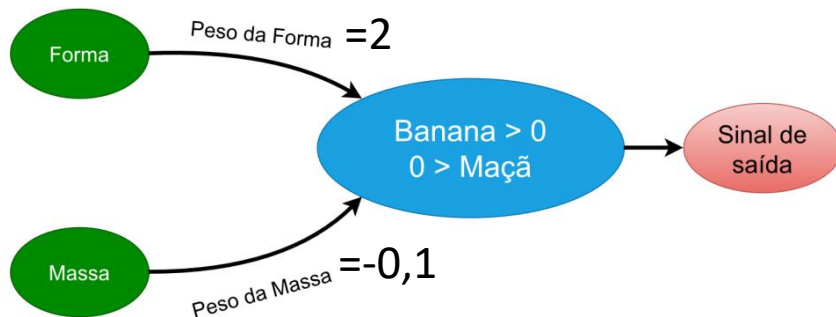
Fruta	Massa (g)	Forma	Massa de 0 a 1
Maçã	139	0.22	0.93
Banana	111	0.7	0.74



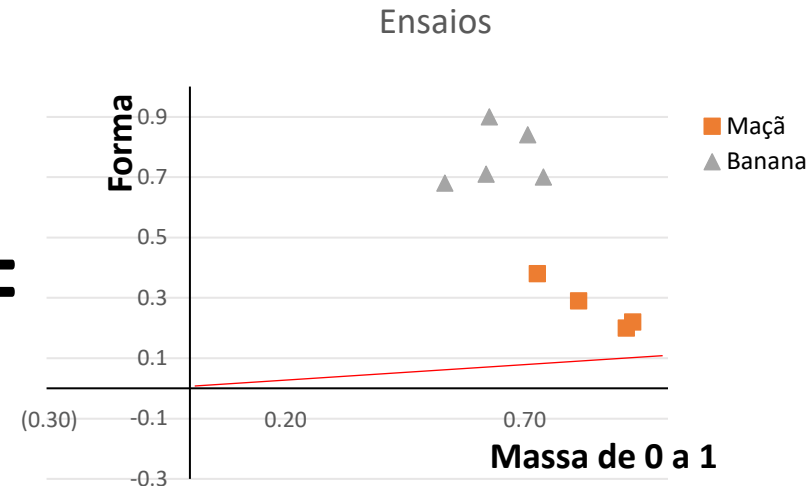
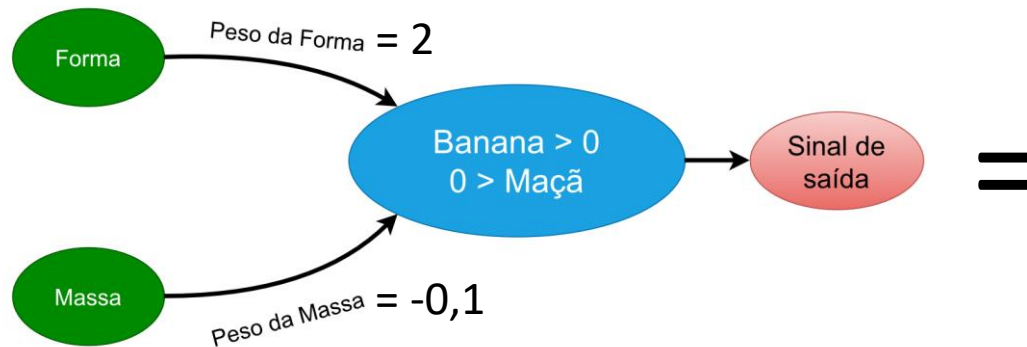
Ao observar o gráfico, verificamos que temos um problema. Todos os pontos que estão a cima da linha vermelha estão classificadas com banana, ou seja, existem maçãs mal classificadas.

A linha vermelha no gráfico está posicionada deste modo devido aos pesos das entradas.

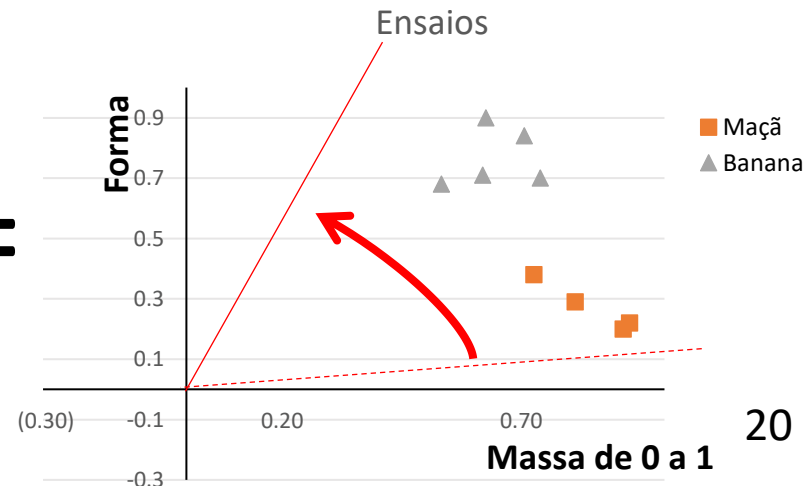
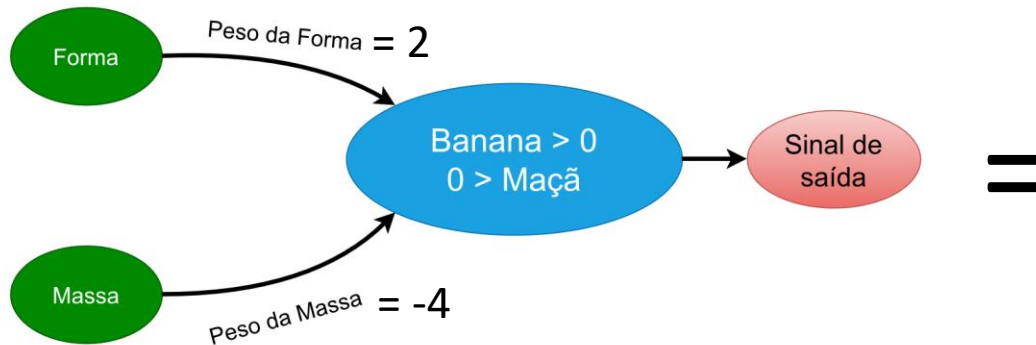
De maneira a corrigir este problema, é necessário atualizar os valores dos pesos.



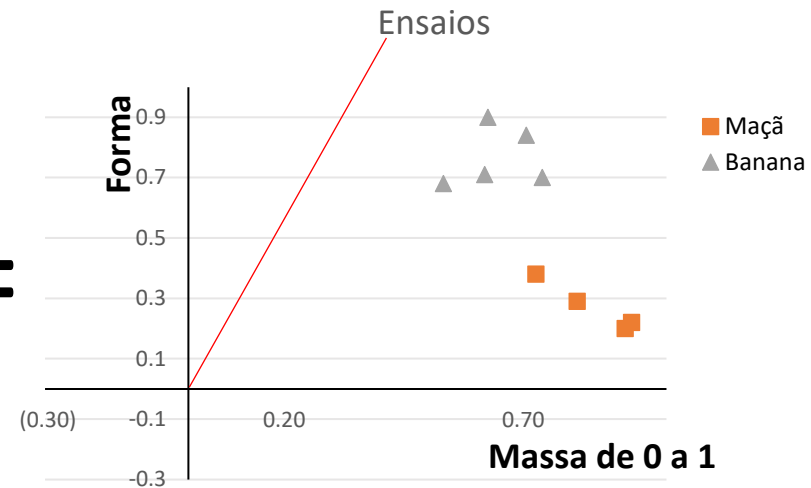
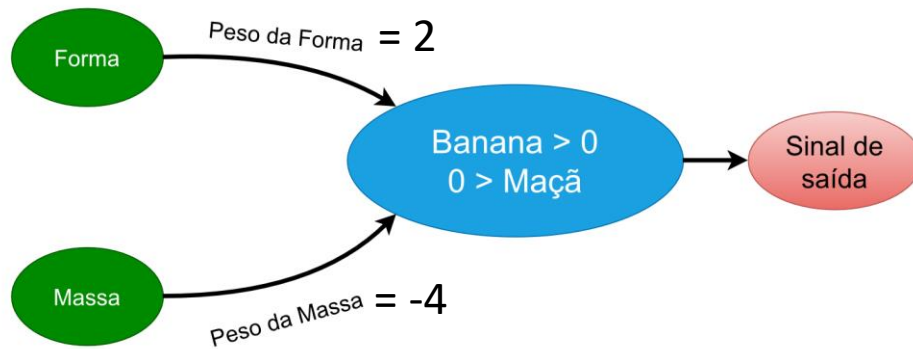
Problema: Há maçãs a ser consideradas bananas.



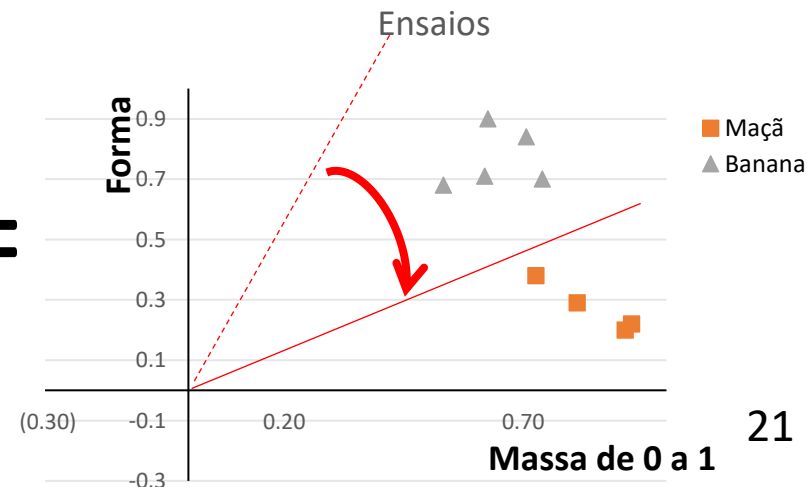
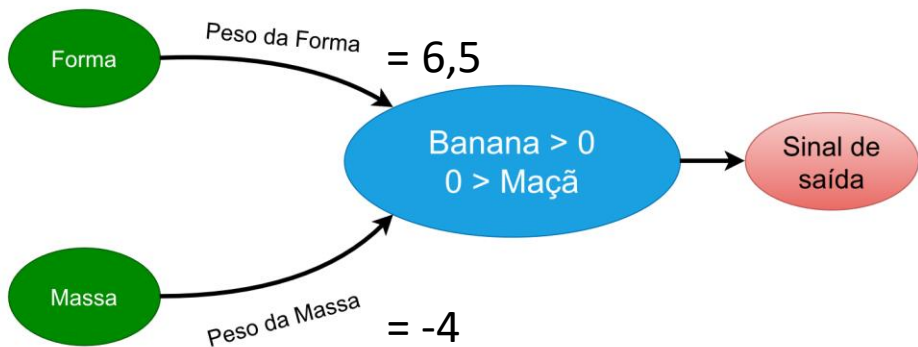
Ao experimentarmos mudar o **Peso da Massa**, verificamos que reduzindo o valor a reta tende a subir, que é o que nós pretendemos.



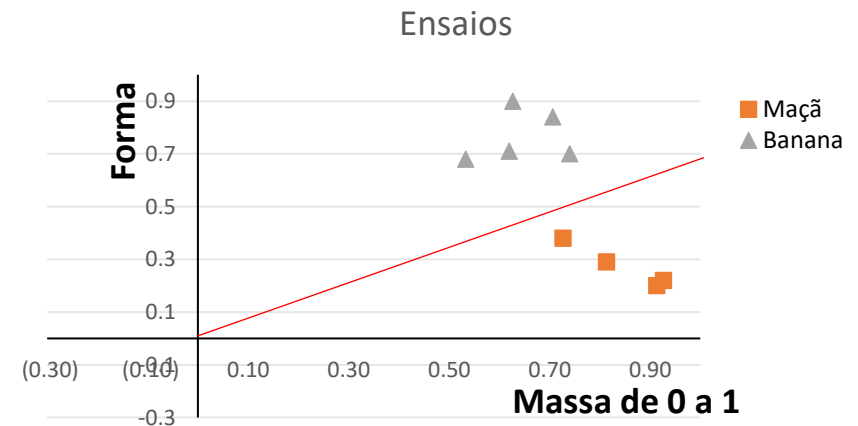
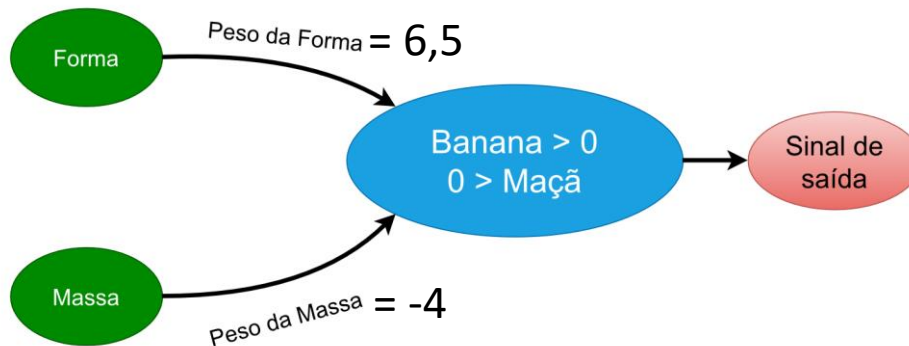
No entanto, se for demasiado pequeno, poderemos ter o problema inverso. Agora as bananas são classificadas como maçãs.



Experimentando mudar o **Peso da Forma**, verificamos que, ao aumentar o valor a reta tende a descer, que é o que nós precisamos agora.

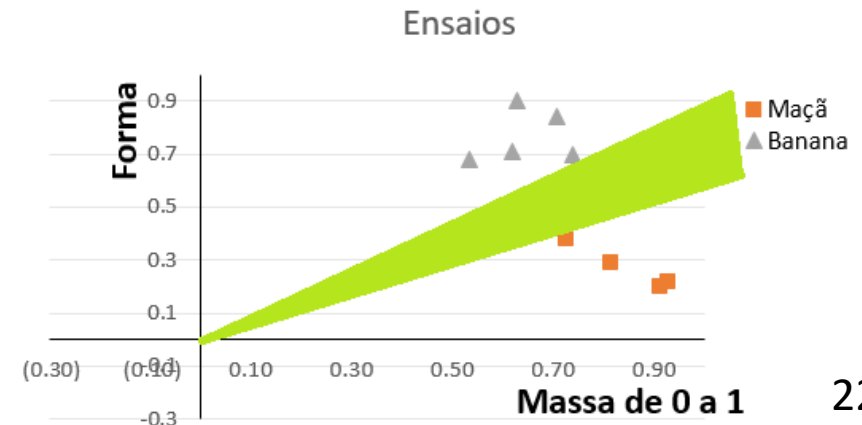


Assim já temos pesos adequados para esta situação, que classifica bem as maçãs e as bananas.



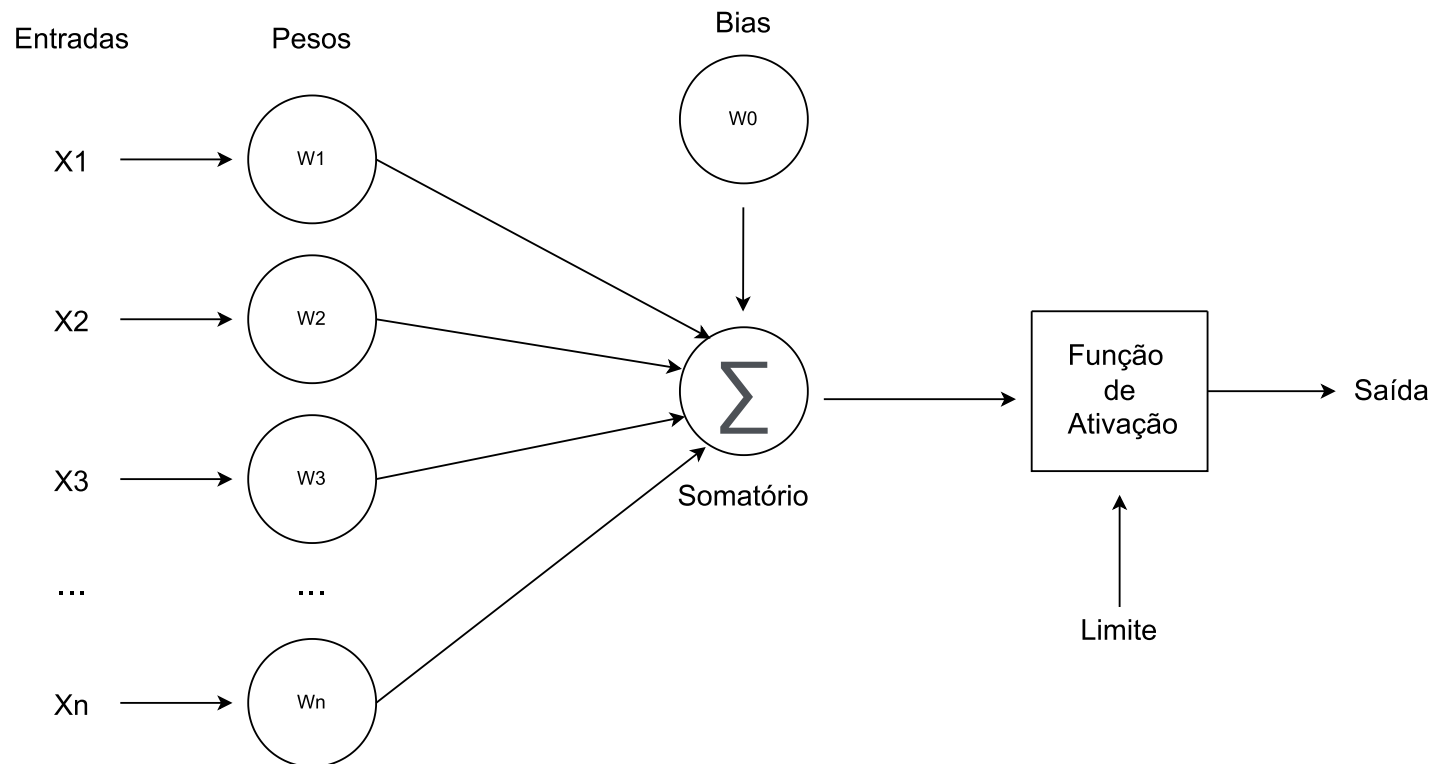
No entanto, estes pesos não são os únicos que funcionam para esta situação.

Qualquer valor dos pesos que “criariam” uma reta situada na região verde no gráfico ao lado separando as maçãs das bananas estaria correta para este conjunto de dados.

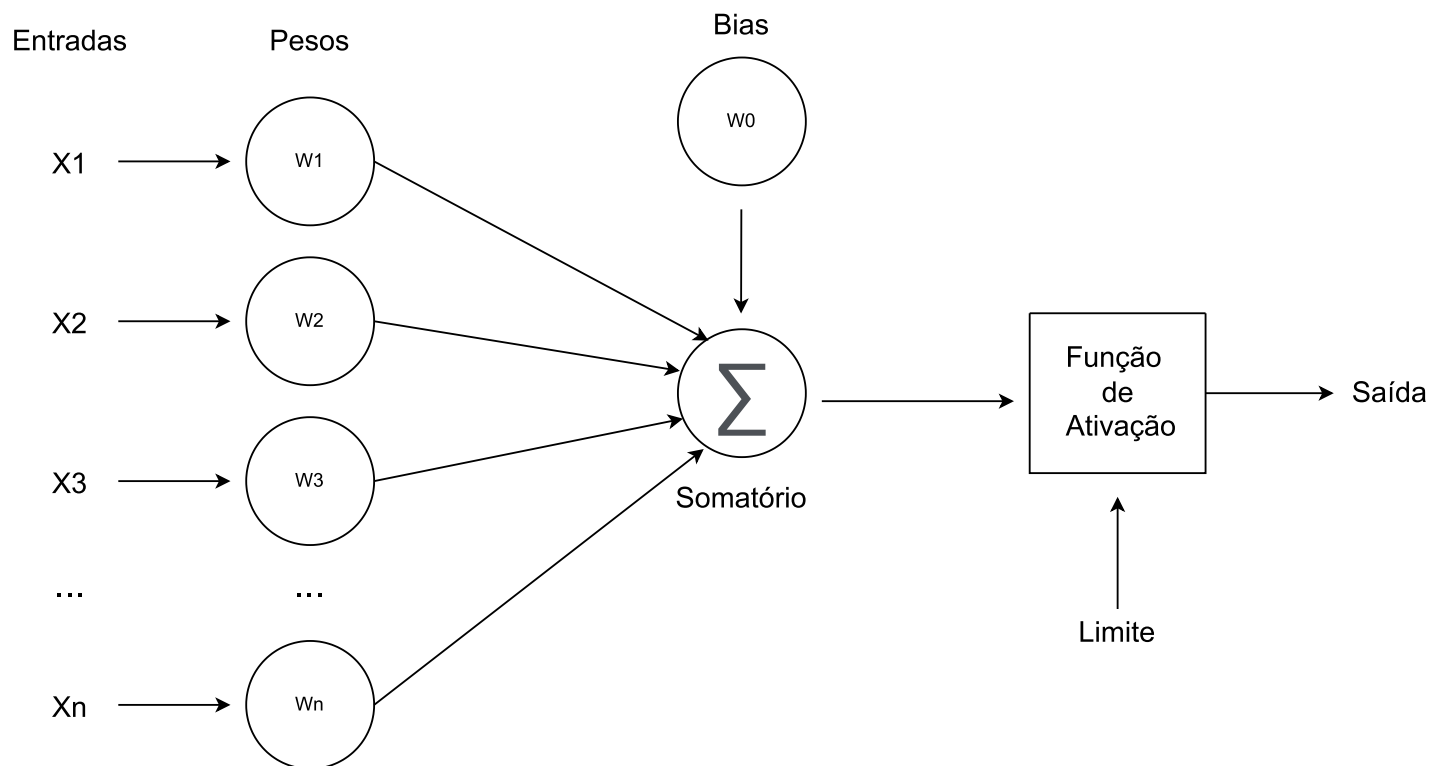




De forma simples, a sua função é “definir” se a Soma Ponderada supera um certo limite definido, de forma a “ativar” a saída do Perceptrão. Ou seja, se o limite for ultrapassado a saída é ativada.

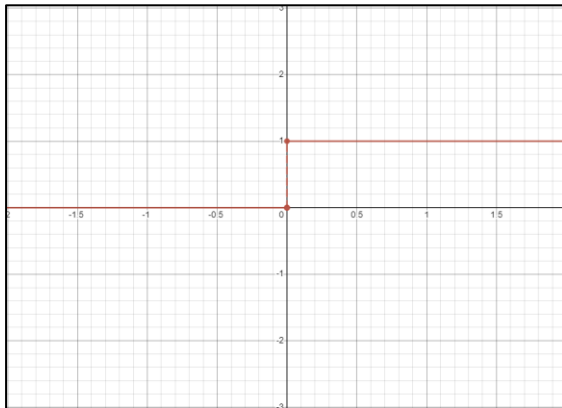


Existem diversas funções de ativação possíveis, a escolha de uma função vai ser influenciada pela situação. Atendendo a cada situação diferente, será usada a função de ativação que melhor se aplica de forma a obtermos o resultado que pretendemos.

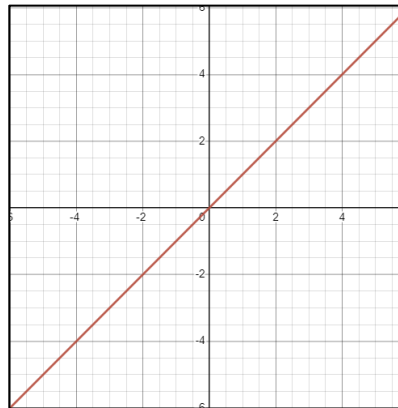


As Funções de Ativação podem ser divididas em 3 tipos:

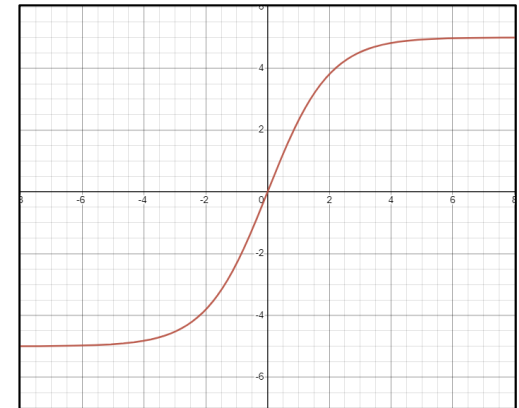
- Funções de degrau unitário;
- Funções lineares;
- Funções não-lineares (Ex: Sigmoide).



Degrau Unitário



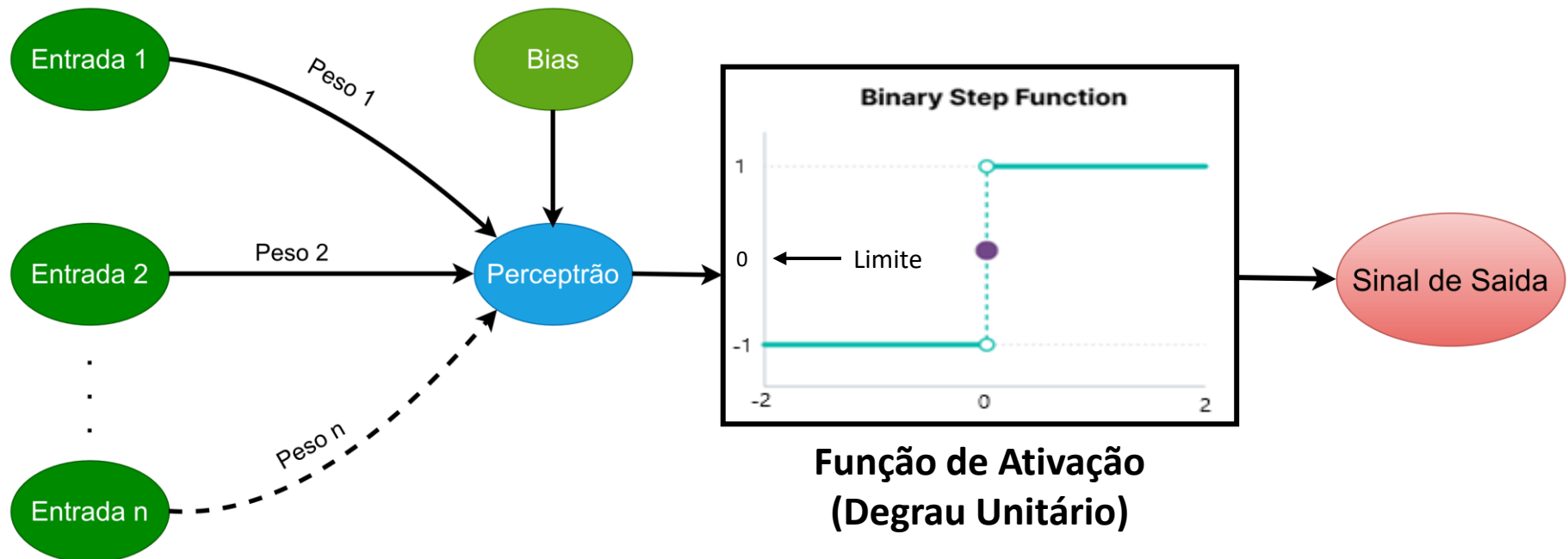
Função Linear  
( $x=y$ )



Sigmoid

## Exemplo

Analisando o exemplo anterior, das bananas e maçãs, este utiliza uma função de ativação de degrau unitário. Todos os valores que são superiores ao limite (0) são considerados como bananas (1) e todos os valores abaixo são considerados maçãs (-1).

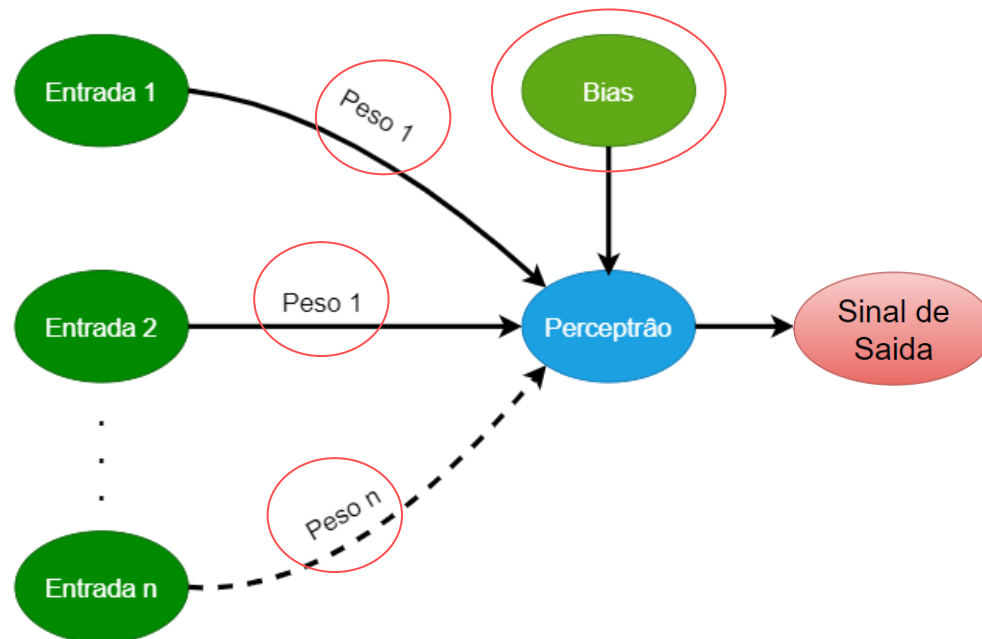


## O que são os Parâmetros?

Os parâmetros são coeficientes do modelo que são “escolhidos” pelo próprio modelo. Ou seja, o algoritmo, enquanto aprende, “otimiza” esses coeficientes (de acordo com o modelo) de modo a minimizar o erro.

Estes são conhecidos como:

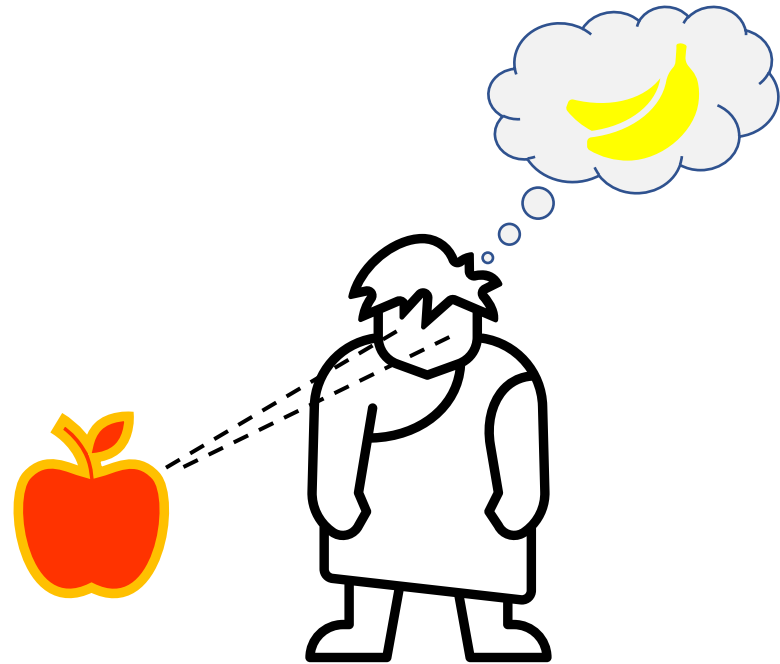
- Pesos
- Bias (Explicado posteriormente)



Todas as saídas tem um erro associado que pode ser reduzido alterando e enviando o erro para trás (feedback). Envia-se o erro “para trás” ajustando os pesos e o bias de modo a reduzi-lo.

Existem diferentes tipos de erros :

- Erro instantâneo;
- Erro quadrático médio;
- Erro absoluto médio;

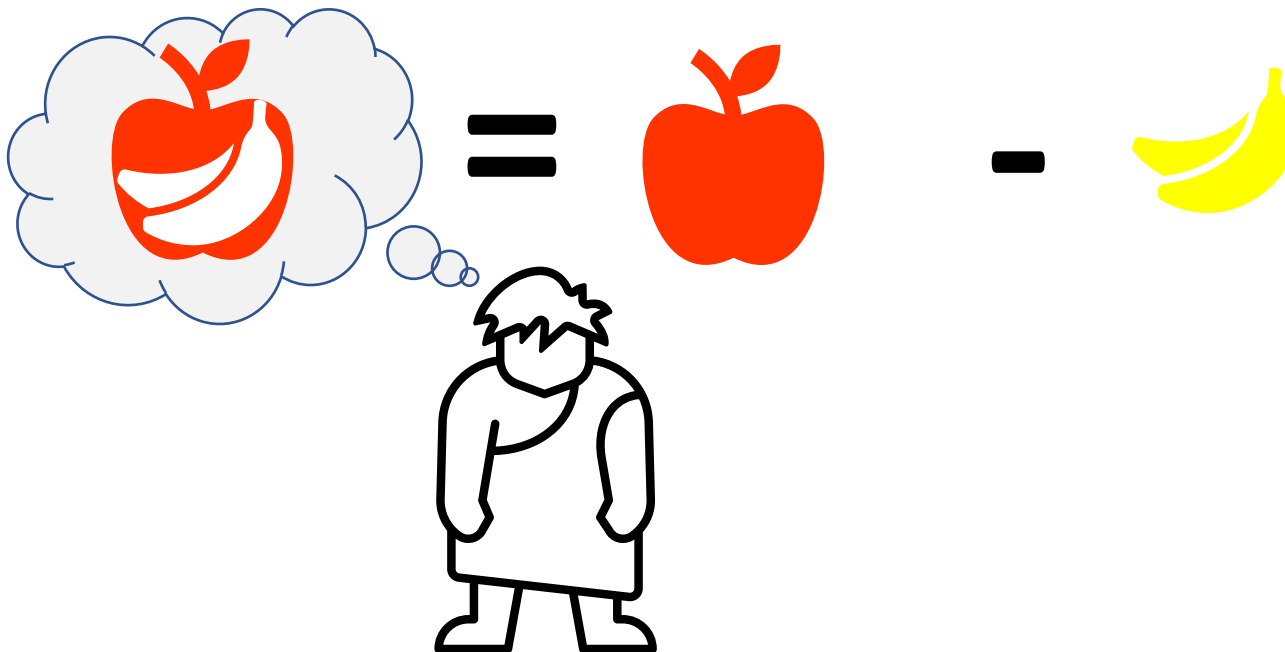


No entanto, iremos falar apenas do erro instantâneo por agora.

O erro instantâneo como o seu nome indica é no instante que acontece a ação.

Ele é calculado com a seguinte expressão.

**Erro instantâneo = Resposta correta – Sinal de saída**

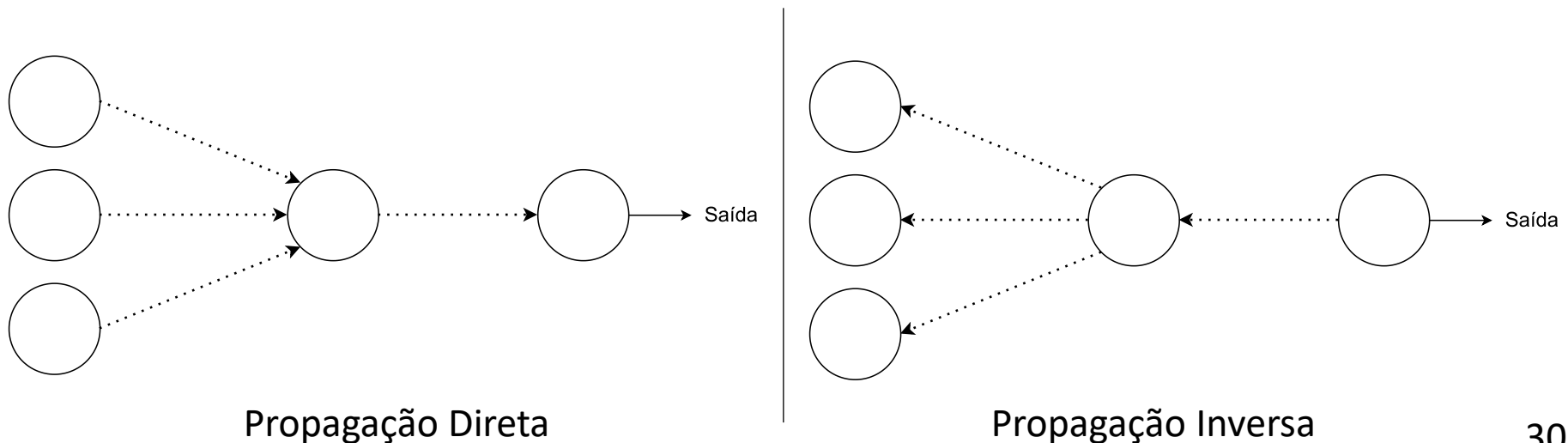


Podemos dividir o processo de aprendizagem em 2 fases:

## 1. Propagação

**1.1** - Propagação direta de um padrão (referindo que este é um algoritmo supervisionado e conhecemos as saídas) para gerar as ativações de saída da rede.

**1.2** - Propagação inversa das saídas pela rede neuronal usando as saídas reais para obter os erros de todos os neurónios.





## 2. Atualização dos pesos

Para cada um dos pesos:

**2.1** – Multiplica-se o seu erro de saída ( $x_i$ ) com a ativação da entrada ( $y_i$ ) para obter o **gradiente** de peso ( $x_i y_i$ ).

**2.2** - Subtraímos uma percentagem ( $\eta$ ) do gradiente desse peso.

A percentagem utilizada na etapa 2.2 influencia bastante a velocidade e a qualidade da aprendizagem do algoritmo e é chamada de "**taxa de aprendizagem**".

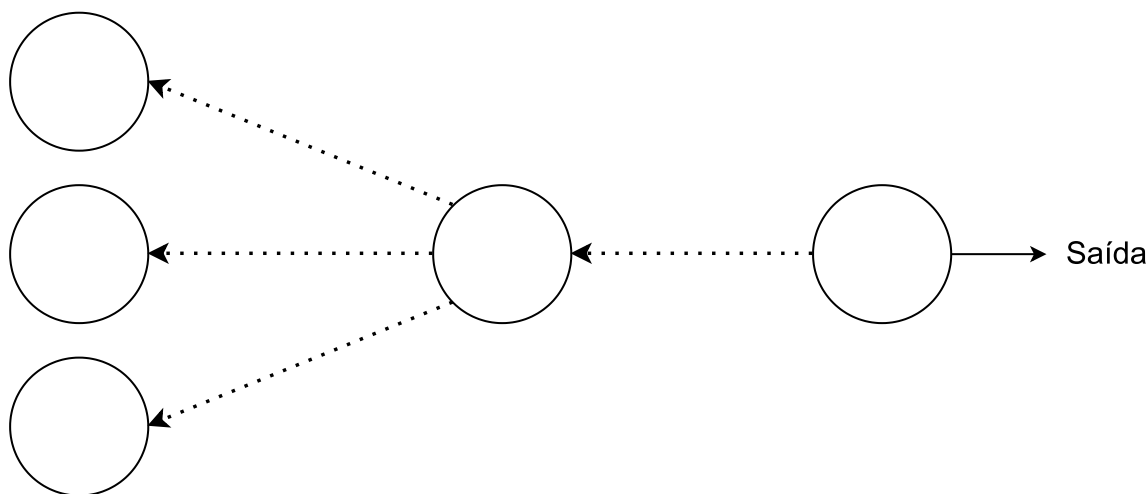
$$\text{"Novo" Peso} \longrightarrow \omega_j = \omega_j + \eta y_i x_i$$

↑  
Peso Inicial

A retropropagação é um algoritmo necessário para fazer melhorias quando se obtêm maus resultados na aprendizagem da máquina e da extração de dados.

Este algoritmo é usado para treinar efetivamente uma rede neuronal por meio de um método chamado regra da cadeia. Em termos simples, após cada passagem para frente numa rede, a retropropagação realiza uma passagem para trás (feedback) enquanto ajusta os parâmetros do modelo (pesos e bias).

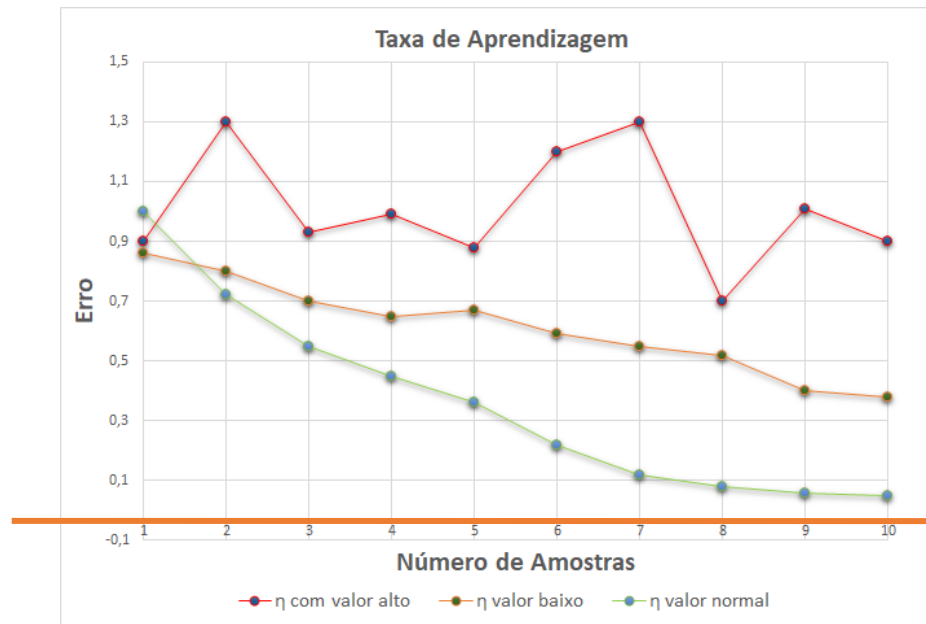
Ou seja, de forma a corrigir os nossos erros, quando o valor é diferente da rede de alimentação esperada, é considerado como um erro. Este algoritmo é definido para que o modelo altere os parâmetros (pesos e bias) cada vez que a saída não é a esperada.



A taxa de aprendizagem é representada pelo caracter  $\eta$ , e este valor é usado para se reduzir o erro da resposta obtida.

No entanto, um problema que pode ocorrer ao aumentar o valor, é que maior irão ser os “saltos” realizados e com isso poderá aumentar o muito o erro ou, se tudo correr bem, diminuir o erro rapidamente.

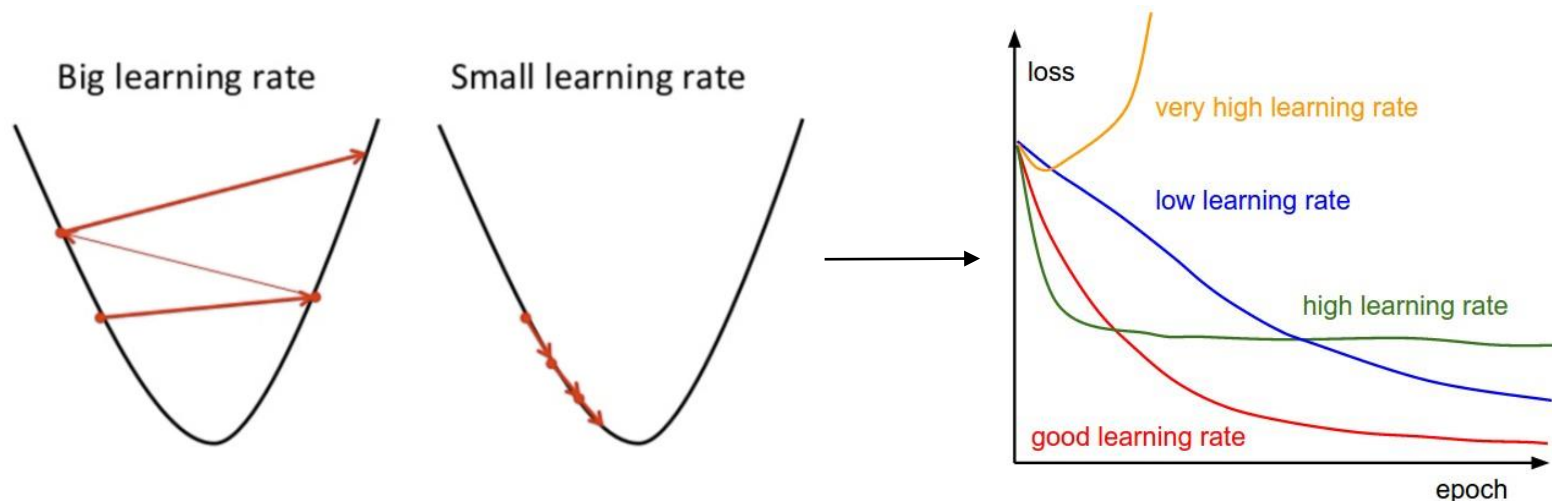
Já se o valor  $\eta$  for muito pequeno, são realizados “saltos” pequenos, logo o erro vai ser reduzido muito devagar até ter um erro quase zero que se chama a zona de saturação. Mas poderá haver situações em que não se reduza o erro por  $\eta$  ser demasiado pequeno.



Saturação

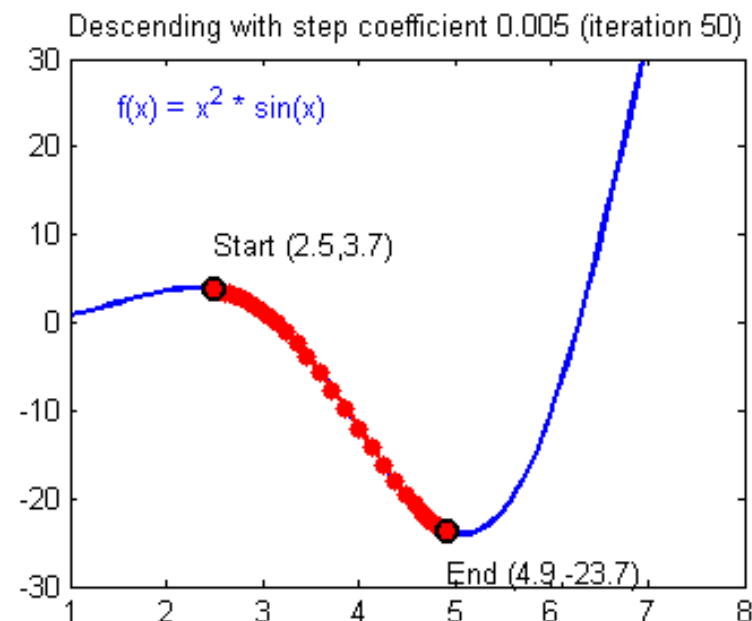
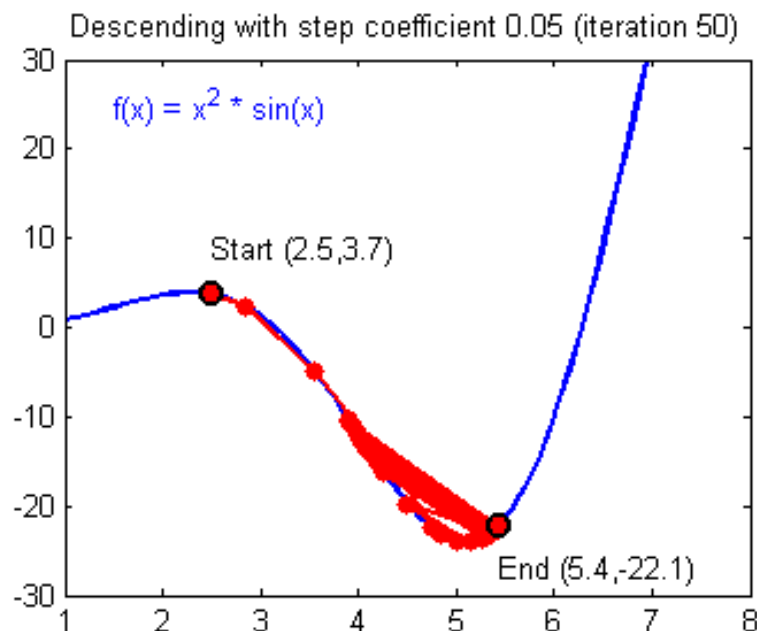
Tal como dito anteriormente, é necessário cautela com o valor da taxa de aprendizagem. Se usarmos uma taxa demasiado grande podemos nunca conseguir chegar aos resultados pretendidos.

Por isso, normalmente, é inicialmente utilizado um valor maior, **sendo diminuído ao longo do tempo de aprendizagem**, de forma a nos aproximarmos dos resultados pretendidos.



Como é mostrado na imagem, com valores muito altos, iremos dar “passos” muito grandes nos valores ao tentarmos corrigi-los, e se a taxa não for diminuída, o erro irá ser sempre muito grande.

Podemos ver duas situações. Na primeira, a taxa de aprendizagem é alta, dando "saltos" grandes, o que faz com que demore menos tempo mas que o erro de resposta obtido seja maior. Na segunda situação, a taxa de aprendizagem é mais adequada, dando saltos pequenos até chegar ao erro de resposta obtido mais baixo.



## Ajuste dos Pesos - Exemplo

De forma a ajustarmos os pesos para obtermos valores mais acertados, retiramos as coordenadas de todos os pontos (frutas) no gráfico de forma a calcularmos o erro e um novo valor para os pesos das entradas, caso os atuais não sejam os corretos.

Para cada fruta, se  $y_i f(x_i) \leq 0$ , então estamos perante uma classificação errada do tipo de fruta. As estimativas são obtidas através da equação:

$$y_i f(x_i) = y_i (\omega^T x_i)$$

No caso de verificarmos que a classificação do tipo de fruta está errada, usamos a seguinte equação de forma a corrigir os pesos:

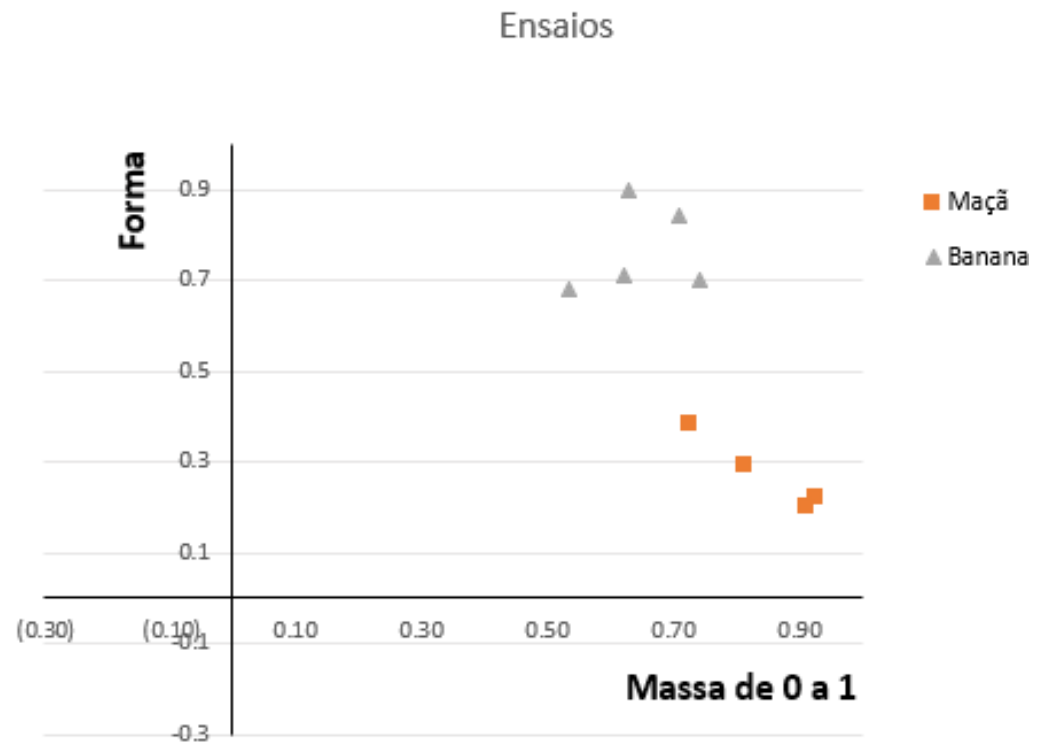
$$\omega_j = \omega_j + \eta y_i x_i$$

## Ajuste dos Pesos - Exemplo

Usando o exemplo anterior, e usando o degrau unitário com função de ativação, considerando as bananas como 1 e maçãs como -1, iniciamos\* com uma taxa de aprendizagem de 100% (i.e. valor  $\eta = 1$ ) e bias = 0.

Tendo frutas com as seguintes coordenadas:

	Y	X
Fruta	Forma	Massa de 0 a 1
Maçã	0.22	0.93
Banana	0.7	0.74
Maçã	0.2	0.91
Banana	0.9	0.63
Banana	0.84	0.71
Maçã	0.38	0.73
Banana	0.68	0.53
Banana	0.71	0.62
Maçã	0.29	0.81



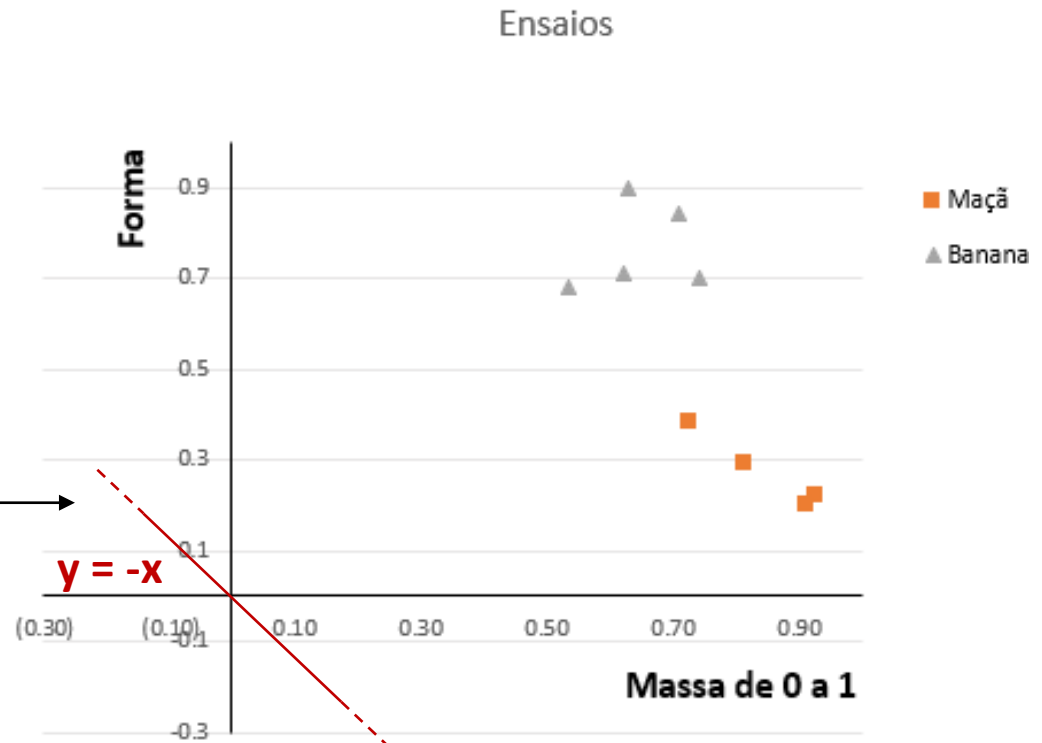
\*A cada 10 iterações temos  $\eta/10$ .

## Ajuste dos Pesos - Exemplo

Assumindo um número aleatório para cada peso, por exemplo, assumindo os dois pesos( $w$ ) com o valor de 1, calcula-se a função para desenhar a reta.

$$\begin{aligned}
 &\text{Peso 1} \quad \text{Peso 2} \\
 &\quad \downarrow \quad \quad \downarrow \\
 &0 = w_1 * x + w_2 * y \Leftrightarrow \\
 &\Leftrightarrow 0 = 1 * x + 1 * y \Leftrightarrow \\
 &\Leftrightarrow 0 = x + y \Leftrightarrow \\
 &\Leftrightarrow y = -x
 \end{aligned}$$

Verifica-se que estes pesos não são os mais adequados, visto que a reta não faz uma correta separação das frutas





## Ajuste dos Pesos

De forma a confirmarmos que a catalogação está errada usamos, então, a função:

$$y_i f(x_i) = y_1 (\omega_0 + \omega^T x_1)$$



Bias

Para a primeira fruta temos:

	Y	X
Maçã	0,22	0,93

$$y_1 f(x_1) = -1(0 + 1 \times 0.93 + 1 \times 0.22) = -1.15 < 0$$



Como o resultado obtido é menor que 0, então os valores dos pesos estão errados, logo recorreremos à expressão:

$$\omega_j = \omega_j + \eta y_i x_i$$

## “Novos” Pesos

Sendo assim, temos o seguinte:

$$\text{Peso 1} \longrightarrow \omega_1 = 1 + 0.1 \times -1 \times 0.93 = 0.907 \quad (\text{Contribuição da Massa})$$

$$\text{Peso 2} \longrightarrow \omega_2 = 1 + 0.1 \times -1 \times 0.22 = 0.978 \quad (\text{Contribuição da Forma})$$

Depois de calculados os novos pesos, repetimos todo o processo novamente com todas as frutas até obtermos a correta catalogação de todas elas e obtermos a reta do gráfico correta.

## Pesos “Finais”

Depois de feitos todos os cálculos para todas as frutas, obteríamos aproximadamente os seguintes valores para os pesos:

Peso 1  $\longrightarrow \omega_1 = -0.2058$  (Contribuição da Massa)

Peso 2  $\longrightarrow \omega_2 = 0.2643$  (Contribuição da Forma)

## Reta Final Obtida

Depois de verificarmos para todas as frutas, percebemos que todas estão bem catalogadas, sendo assim para vermos qual a reta correta usamos a equação usada inicialmente para obter a reta.

Peso 1      **Peso 2**

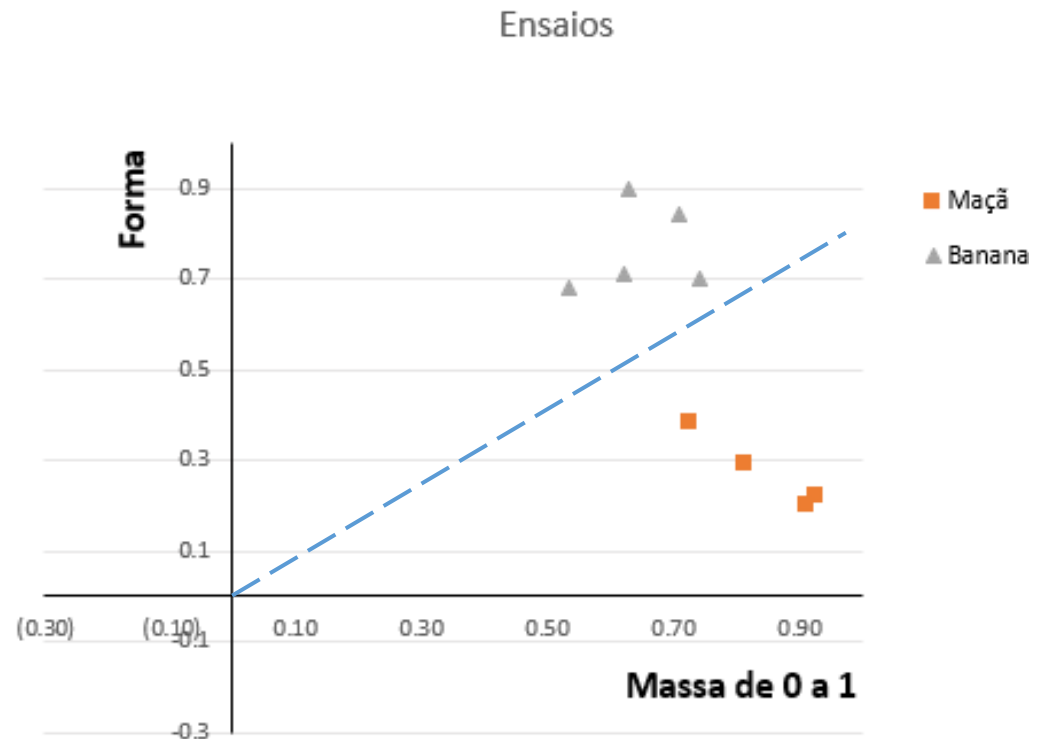
↓      ↓

$$0 = w_1 * x + w_2 * y \Leftrightarrow$$

$$\Leftrightarrow 0 = -0.2058 * x + 0.2643 * y \Leftrightarrow$$

$$\Leftrightarrow 0.2643 y = 0.2058 * x \Leftrightarrow$$

$$\Leftrightarrow y = 0.7787x$$

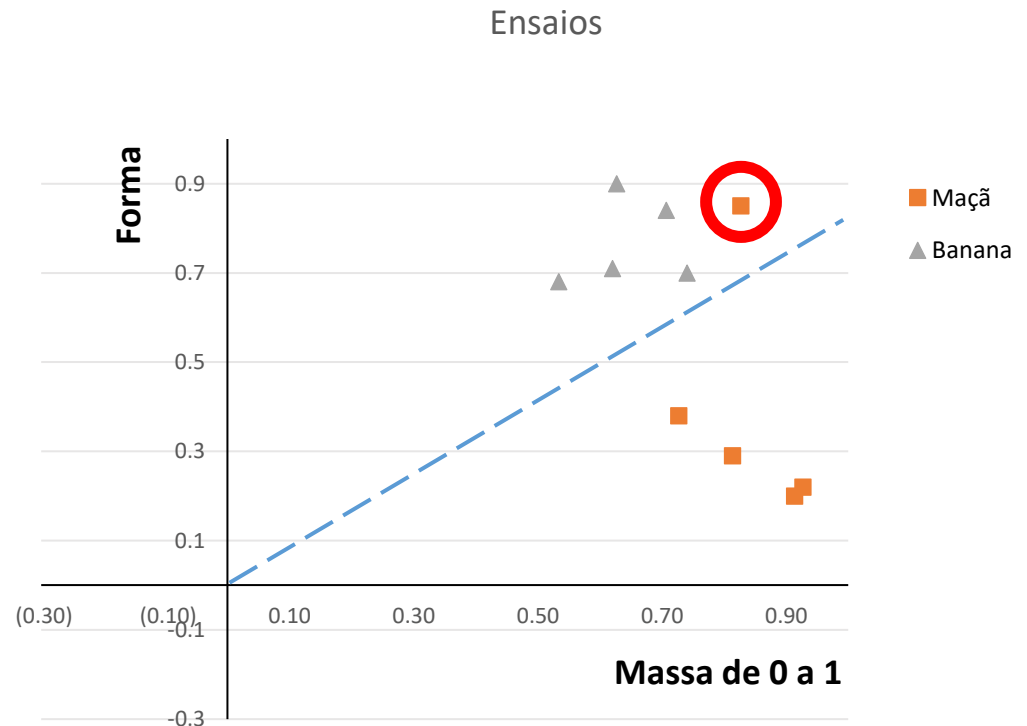


## Ajuste dos Pesos - Exemplo

Continuando o exemplo anterior, adicionamos uma nova maçã aos dados e fazemos de novo os cálculos.

	Y	X
Fruta	Forma	Massa de 0 a 1
Maçã	0.22	0.93
Banana	0.7	0.74
Maçã	0.2	0.91
Banana	0.9	0.63
Banana	0.84	0.71
Maçã	0.38	0.73
Banana	0.68	0.53
Banana	0.71	0.62
Maçã	0.29	0.81
Maçã	0.85	0.83

← Nova Maçã



## Pesos “Finais”

Depois de feitos todos os cálculos para todas as frutas, obteríamos aproximadamente os seguintes valores para os pesos:

Peso 1  $\longrightarrow \omega_1 = -0.263$  (Contribuição da Massa)

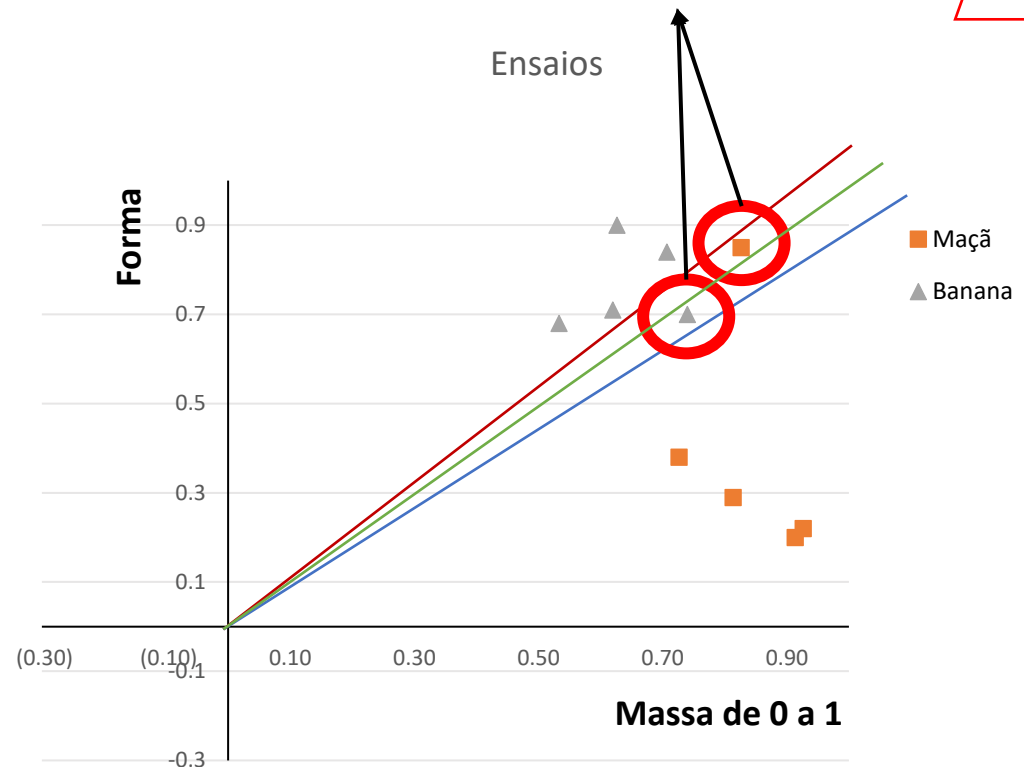
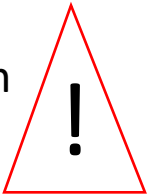
Peso 2  $\longrightarrow \omega_2 = 0.414$  (Contribuição da Forma)

Ainda assim, verificamos que existe sempre uma errada catalogação ou de uma maçã ou de uma banana, pois o nosso bias tem o valor 0.

## Problema!

Usando o bias com valor 0, não existe nenhuma reta que consiga separar todas as maçãs e bananas.

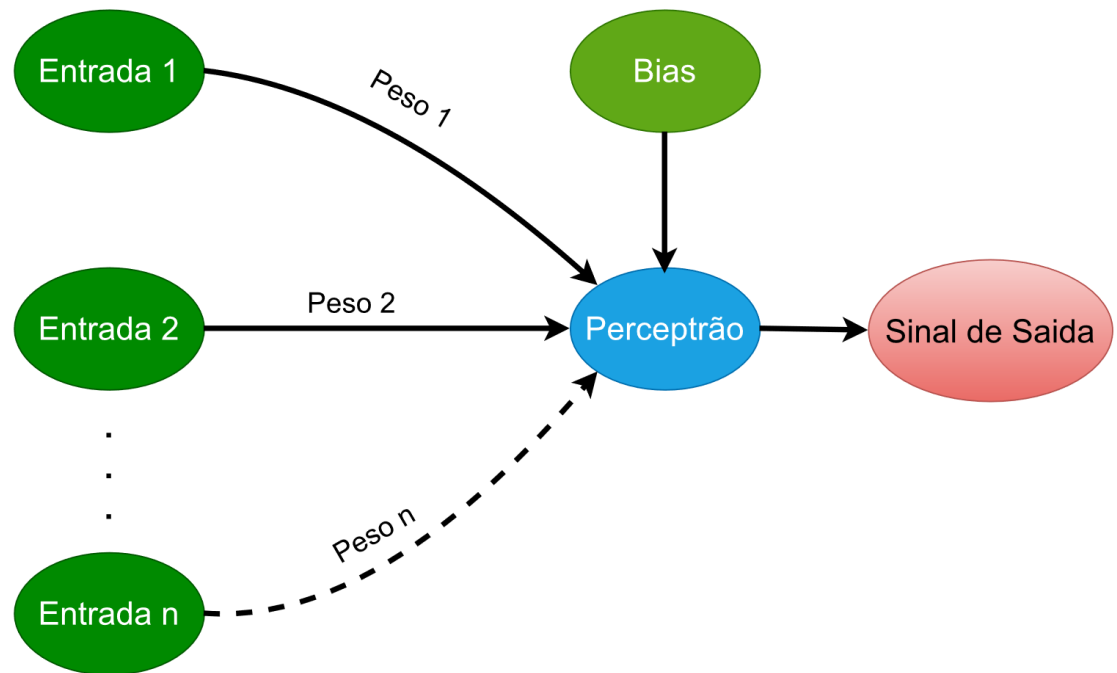
Estas duas frutas não podem ser bem classificadas com a mesma reta.



## O que é o Bias?

Por vezes, o **Bias** é confundido com uma simples entrada devido ao facto da sua função ser muito semelhante.

Na prática, o **Bias** é um valor somado ao somatório obtido pela multiplicação das entradas com os seus respetivos pesos.



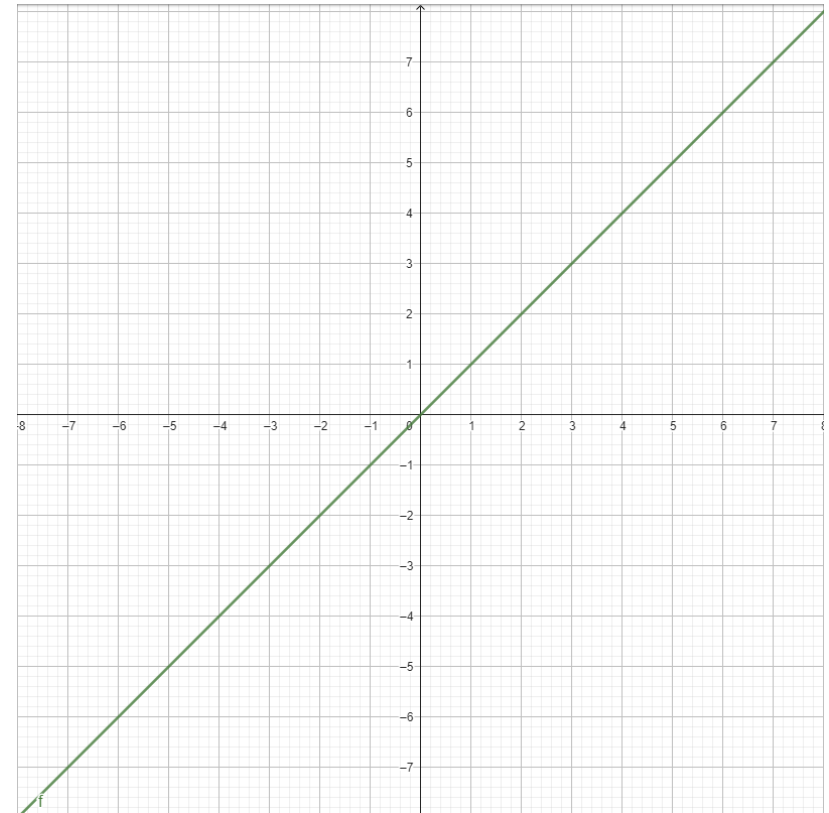
$$\text{Perceptrão} = [(Entrada1 * \text{Peso}1) + (Entrada2 * \text{Peso}2) + \dots + (Entrada n * \text{Peso} n)] + \text{Bias}$$



Uma maneira simples de entender o que é o bias é através de uma função linear.

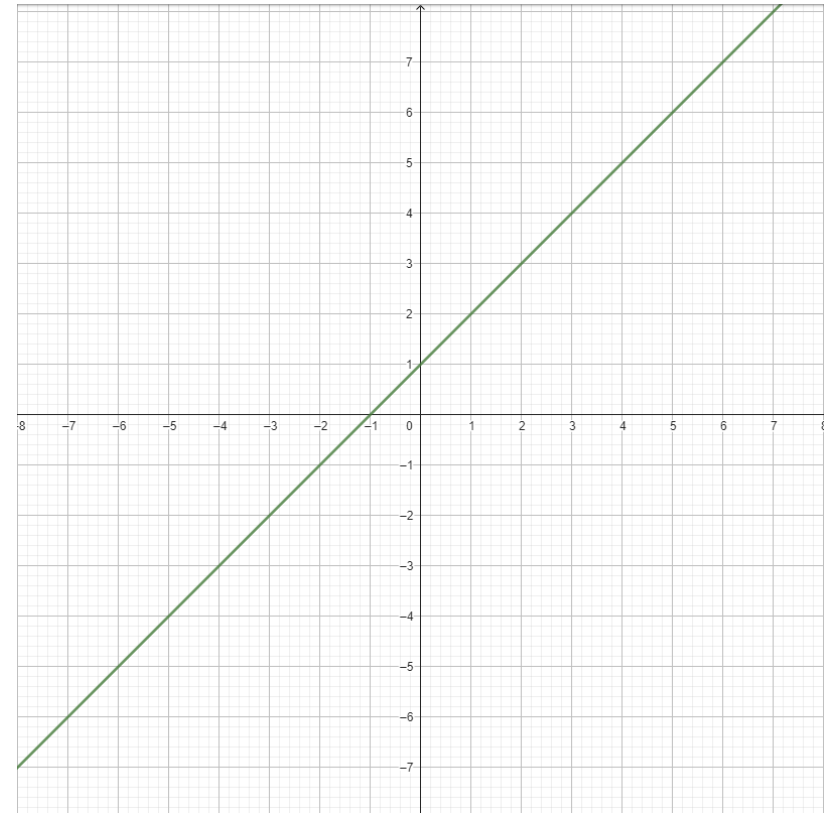
O bias é de alguma forma semelhante à constante **b** de uma função  $y = wx + \mathbf{b}$ .

Este permite que possamos mover a linha para cima e para baixo para ajustar melhor a previsão com os dados. Sem **b**, a linha passa sempre pela origem (0, 0) e podemos ter um resultado fora do pretendido.



Se tivermos  $b=1$ , por exemplo, alteramos a função, ajustando-a de forma a podermos obter resultados dentro dos valores pretendidos.

No caso de termos  $b$  negativo, então a Soma Ponderada deve produzir um valor positivo maior que  $|b|$  para “empurrar” a função de ativação acima do limite 0, de forma a colocar a saída a 1.

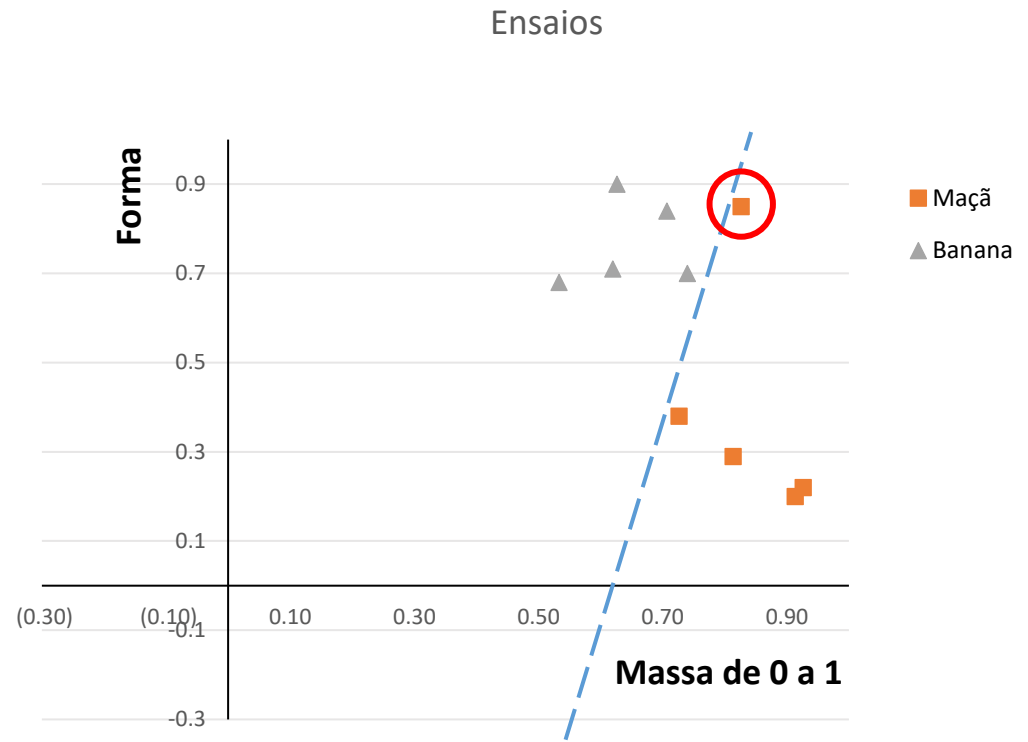


## Solução

Todas as bananas estão à esquerda da reta e as maçãs à direita.

Para conseguirmos separar corretamente todas as frutas necessitaríamos de uma reta algo parecida com esta.

Para tal, precisamos de encontrar o Bias que nos permite obter esta reta.



## Correção do Bias

De forma a obtermos o valor correto para o Bias usamos a mesma fórmula usada para os pesos:

$$\text{"Novo" Bias} \longrightarrow \omega_0 = 0 + 0.1 \times -1 \times 1 = -0.1$$

Agora, com o novo Bias, e usando os valores dos pesos corrigidos obtidos, verificamos se todas as frutas são catalogadas corretamente, usando a mesma equação de anteriormente. Caso não estejam, corrigimos os pesos.

Se ainda assim, com este Bias, não obtivermos os valores corretos, voltamos a corrigir o Bias e assim sucessivamente.

## Valores Finais

Depois de verificarmos para todas as frutas repetidamente, obtemos, aproximadamente os seguintes valores para os pesos e o bias no final de todas as correções:

$$\text{Peso 1} \longrightarrow \omega_1 = -0.3445 \quad (\text{Contribuição da Massa})$$

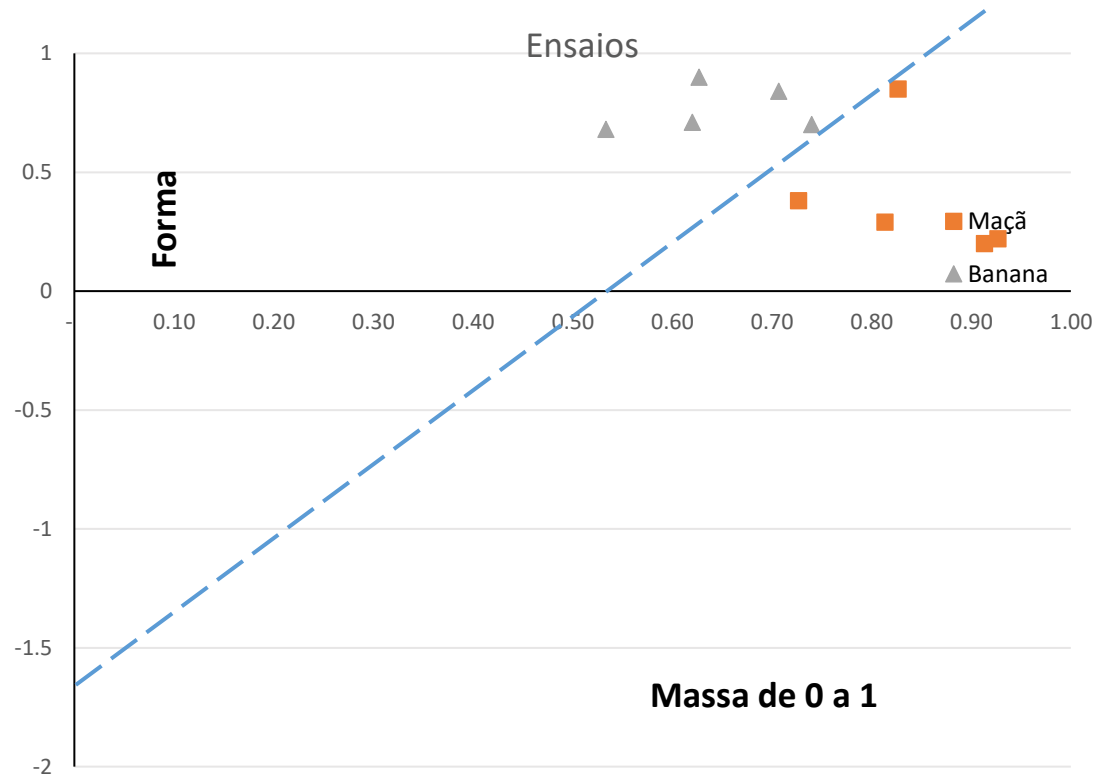
$$\text{Peso 2} \longrightarrow \omega_2 = 0.1118 \quad (\text{Contribuição da Forma})$$

$$\text{Bias Final} \longrightarrow \omega_1 = 0.19$$

## Reta Final Obtida

Depois de verificarmos para todas as frutas, percebemos que todas estão bem catalogadas, sendo assim para vermos qual a reta correta usamos a equação usada inicialmente para obter a reta.

$$\begin{aligned}
 & \text{Bias} \quad \text{Peso 1} \quad \text{Peso 2} \\
 & 0 = w_0 + w_1 * x + w_2 * y \Leftrightarrow \\
 & \Leftrightarrow 0 = 0.19 - 0.3445 * x + 0.1118 * y \Leftrightarrow \\
 & \Leftrightarrow 0.1118y = 0.3445 * x - 0.19 \Leftrightarrow \\
 & \Leftrightarrow y = 3.081x - 1.699
 \end{aligned}$$



# Capítulo 3.

---

## Programa em Python

De forma a facilitar o nosso trabalho e diminuir a possibilidade de erro humano, é criado um algoritmo usando programação em Python de forma a que este faça os cálculos automaticamente até obter os valores corretos dos pesos e do bias.

Dados

```
n = 10

X = np.array( [[0.93,0.22], [0.74,0.61], [0.83,0.85], [0.91,0.2], [0.63,0.9], [0.71,0.84], [0.73, 0.38], [0.53,0.68], [0.62,0.71], [0.81,0.29]] )

Y = np.array( [[-1], [1], [-1], [-1], [1], [1], [-1], [1], [1], [-1]] )

# Divide o dataset em 2, treino e teste

train_x, test_x = np.split(X, [ n*8//10])

train_labels, test_labels = np.split(Y, [n*8//10])

print("Features:\n",train_x[0:10])

print("Labels:\n",train_labels[0:10])
```

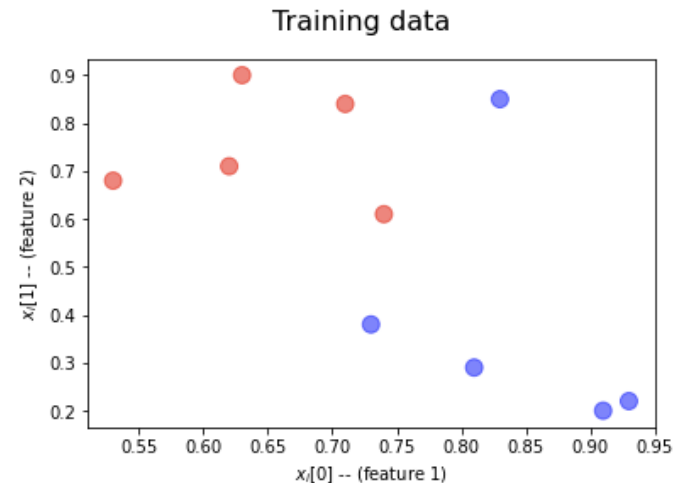
Divisão entre valores de teste e treino



# Definição do gráfico

```
def plot_dataset(supitle, features, labels):  
  
    # prepare the plot  
  
    fig, ax = pylab.subplots(1, 1)  
  
    fig.suptitle(supitle, fontsize = 16)  
  
    ax.set_xlabel('$x_i[0]$ -- (feature 1)')  
  
    ax.set_ylabel('$x_i[1]$ -- (feature 2)')
```

Definição do gráfico



Definição das bananas com cor vermelha e maçãs azul

```
    colors = ['r' if l>0 else 'b' for l in labels]  
  
    ax.scatter(features[:, 0], features[:, 1], marker='o', c=colors, s=100, alpha = 0.5)  
  
    fig.show()
```

```
plot_dataset('Training data', train_x, train_labels)
```

Divide as  
maçãs (neg)  
das bananas  
(pos)

```
pos_examples = np.array([ [t[0], t[1], 1] for i,t in enumerate(train_x)
                           if train_labels[i]>0])

neg_examples = np.array([ [t[0], t[1], 1] for i,t in enumerate(train_x)
                           if train_labels[i]<0])

print(pos_examples[0:10])

print(neg_examples[0:10])
```

Definição da  
função de  
treino

```
# Definição da função de treino do perceptrão

def train_all(positive_examples, negative_examples, num_iterations = 1000):

    num_dims = positive_examples.shape[1]

    print(num_dims)

    # Inicialização dos pesos a 0 para simplificar, idealmente teriam valores
aleatórios

    weights = np.zeros((num_dims,1))

    pos_count = positive_examples.shape[0]

    neg_count = negative_examples.shape[0]

    report_frequency = 10

    learning_rate=1;
```

Continuação  
da definição  
da função  
de treino

```
for i in range(num_iterations):  
  
    for j in range(pos_count):  
  
        # Pick one positive and one negative example  
  
        pos = positive_examples[j]  
  
        neg = negative_examples[j]  
  
        z = np.dot(pos, weights)  
  
        if z < 0: # positive example was classified as negative  
  
            weights = weights + learning_rate*pos.reshape(weights.shape)  
  
            z = np.dot(neg, weights)  
  
        if z >= 0: # negative example was classified as positive  
  
            weights = weights - learning_rate * neg.reshape(weights.shape)
```

# Função que mostra a alteração dos valores da precisão da classificação e dos pesos e bias periodicamente.

```
if i % report_frequency == 0:

    if learning_rate > 0.01:

        learning_rate= learning_rate/10

    pos_out = np.dot(positive_examples, weights)

    neg_out = np.dot(negative_examples, weights)

    pos_correct = (pos_out >= 0).sum() / float(pos_count)

    neg_correct = (neg_out < 0).sum() / float(neg_count)

    print("Iteration={}, learning_rate={}, pos correct={}, neg correct={}".format(i, learning_rate, pos_correct, neg_correct))

    print("weights={}".format(weights.transpose()))

    if pos_correct == 1 and neg_correct == 1:

        print("Iteration={}, learning_rate={}, pos correct={}, neg correct={}".format(i, learning_rate, pos_correct, neg_correct))

        print("weights={}".format(weights.transpose()))

    return weights
return weights

wts = train_all(pos_examples, neg_examples)

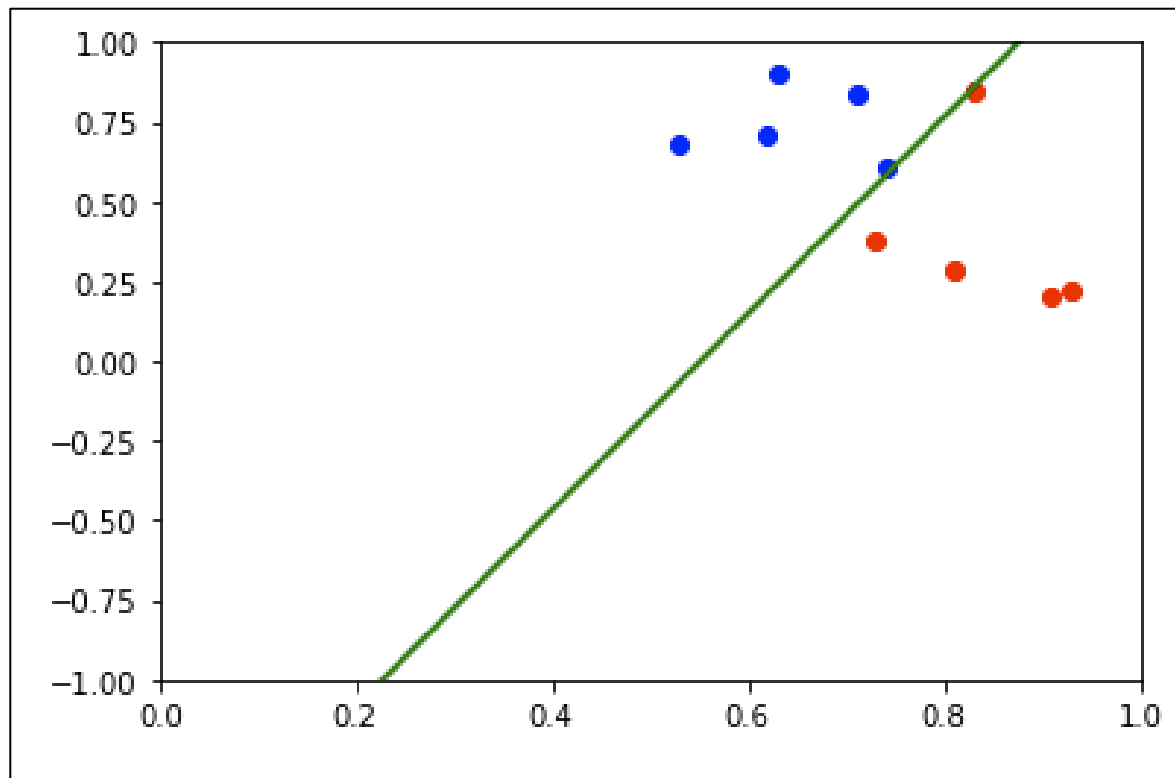
print(wts.transpose())
```

Continuação  
da definição  
da função  
de treino

Visualização  
do gráfico  
com a reta  
final

```
def plot_boundary(positive_examples, negative_examples, weights):  
    if np.isclose(weights[1], 0):  
        if np.isclose(weights[0], 0):  
            x = y = np.array([-6, 6], dtype = 'float32')  
        else:  
            y = np.array([-6, 6], dtype='float32')  
            x = -(weights[1] * y + weights[2])/weights[0]  
    else:  
        x = np.array([-6, 6], dtype='float32')  
        y = -(weights[0] * x + weights[2])/weights[1]  
        pylab.xlim(0, 1)  
  
        pylab.ylim(-1, 1)  
  
        pylab.plot(positive_examples[:,0], positive_examples[:,1], 'bo')  
        pylab.plot(negative_examples[:,0], negative_examples[:,1], 'ro')  
  
        pylab.plot(x, y, 'g', linewidth=2.0)  
  
        pylab.show()  
  
plot_boundary(pos_examples,neg_examples,wt)
```

Gráfico final com os pesos e bias atualizados para os valores corretos



É possível criar uma função que nos mostre a alteração da reta ao longo do tempo:

```
def train_all_graph(positive_examples, negative_examples, num_iterat
ions = 1000):

    num_dims = positive_examples.shape[1]

    print(num_dims)

    # Initialize weights.

    # We initialize with 0 for simplicity, but random initialization
    is also a good idea

    weights = np.zeros((num_dims,1))

    pos_count = positive_examples.shape[0]

    neg_count = negative_examples.shape[0]

    print(pos_count)

    print(neg_count)

    report_frequency = 10

    learning_rate=1;

    snapshots = []
```

Visualização da  
mudança da reta  
de classificação  
ao longo do  
tempo

Continuação da  
função

```
for i in range(num_iterations):  
    for j in range(pos_count):  
        # Pick one positive and one negative example  
        pos = positive_examples[j]  
        neg = negative_examples[j]  
  
        z = np.dot(pos, weights)  
  
        if z < 0: # positive example was classified as negative  
            weights = weights + learning_rate*pos.reshape(weights.  
shape)  
  
            z = np.dot(neg, weights)  
  
            if z >= 0: # negative example was classified as positive  
                weights = weights -  
learning_rate * neg.reshape(weights.shape)
```



Continuação  
da função

```
# Periodically, print out the current accuracy on all examples

    if i % report_frequency == 0:

        if learning_rate > 0.01:

            learning_rate= learning_rate/10

        pos_out = np.dot(positive_examples, weights)

        neg_out = np.dot(negative_examples, weights)

        pos_correct = (pos_out >= 0).sum() / float(pos_count)

        neg_correct = (neg_out < 0).sum() / float(neg_count)

        print("Iteration={}, learning_rate={}, pos correct={}, neg correct={}".forma
t(i,learning_rate,pos_correct,neg_correct))

        print("weights={}".format(weights.transpose()))

        snapshots.append((np.copy(weights),(pos_correct+neg_correct)/2.0))
        if pos_correct == 1 and neg_correct == 1:

            print("Iteration={}, learning_rate={}, pos correct={}, neg correct={}".format(
i,learning_rate,pos_correct,neg_correct))

            print("weights={}".format(weights.transpose()))

            snapshots.append((np.copy(weights),(pos_correct+neg_correct)/2.0))

        return np.array(snapshots)

return np.array(snapshots)
```

Continuação  
da função

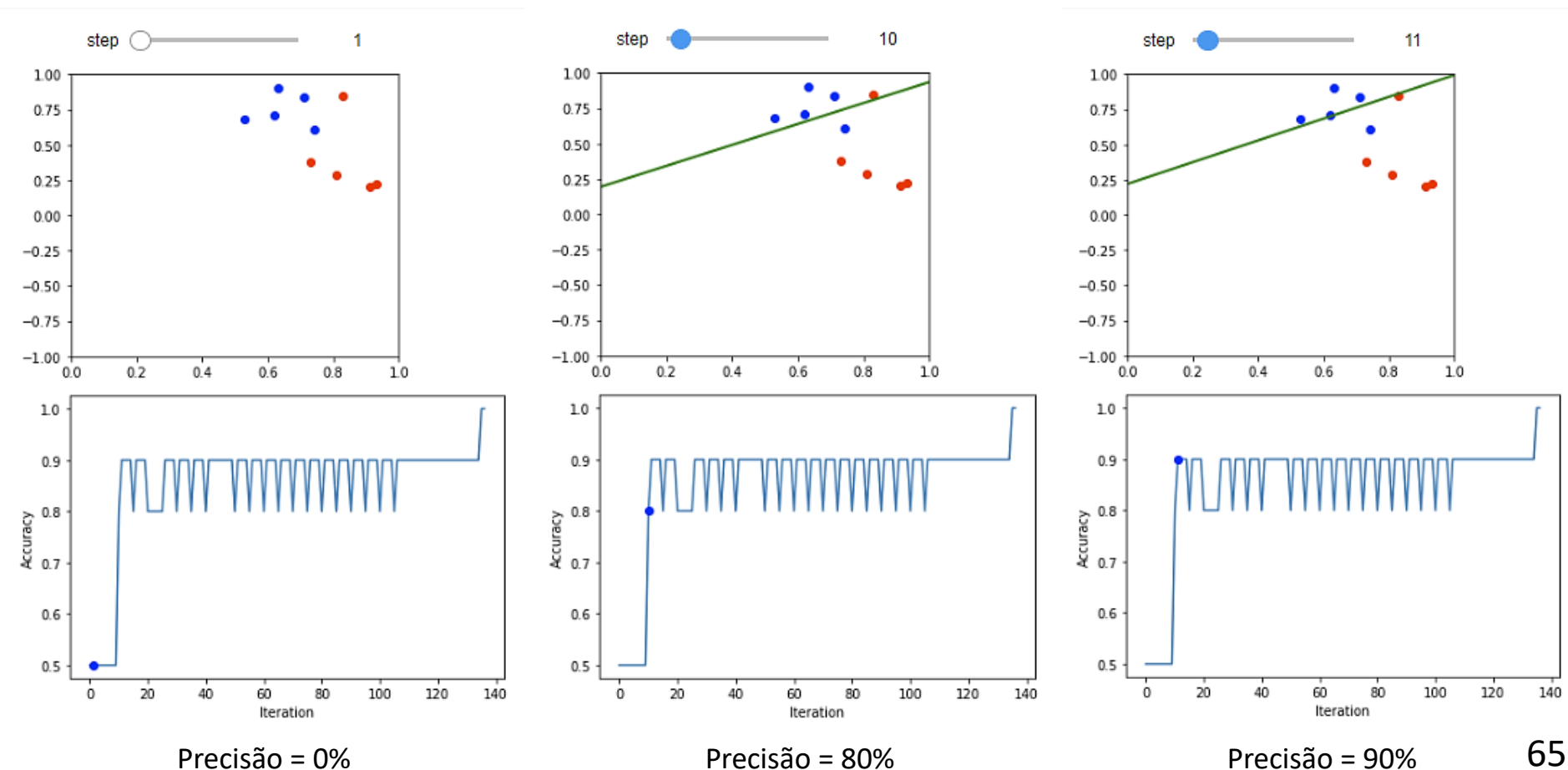
```
snapshots = train_all_graph(pos_examples,neg_examples)

def plotit_all(pos_examples,neg_examples,snapshots,step):
    fig = pylab.figure(figsize=(10,4))
    fig.add_subplot(1, 2, 1)
    plot_boundary(pos_examples, neg_examples, snapshots[step][0])
    fig.add_subplot(1, 2, 2)
    pylab.plot(np.arange(len(snapshots[:,1])), snapshots[:,1])
    pylab.ylabel('Accuracy')
    pylab.xlabel('Iteration')
    pylab.plot(step, snapshots[step,1], "bo")
    pylab.show()

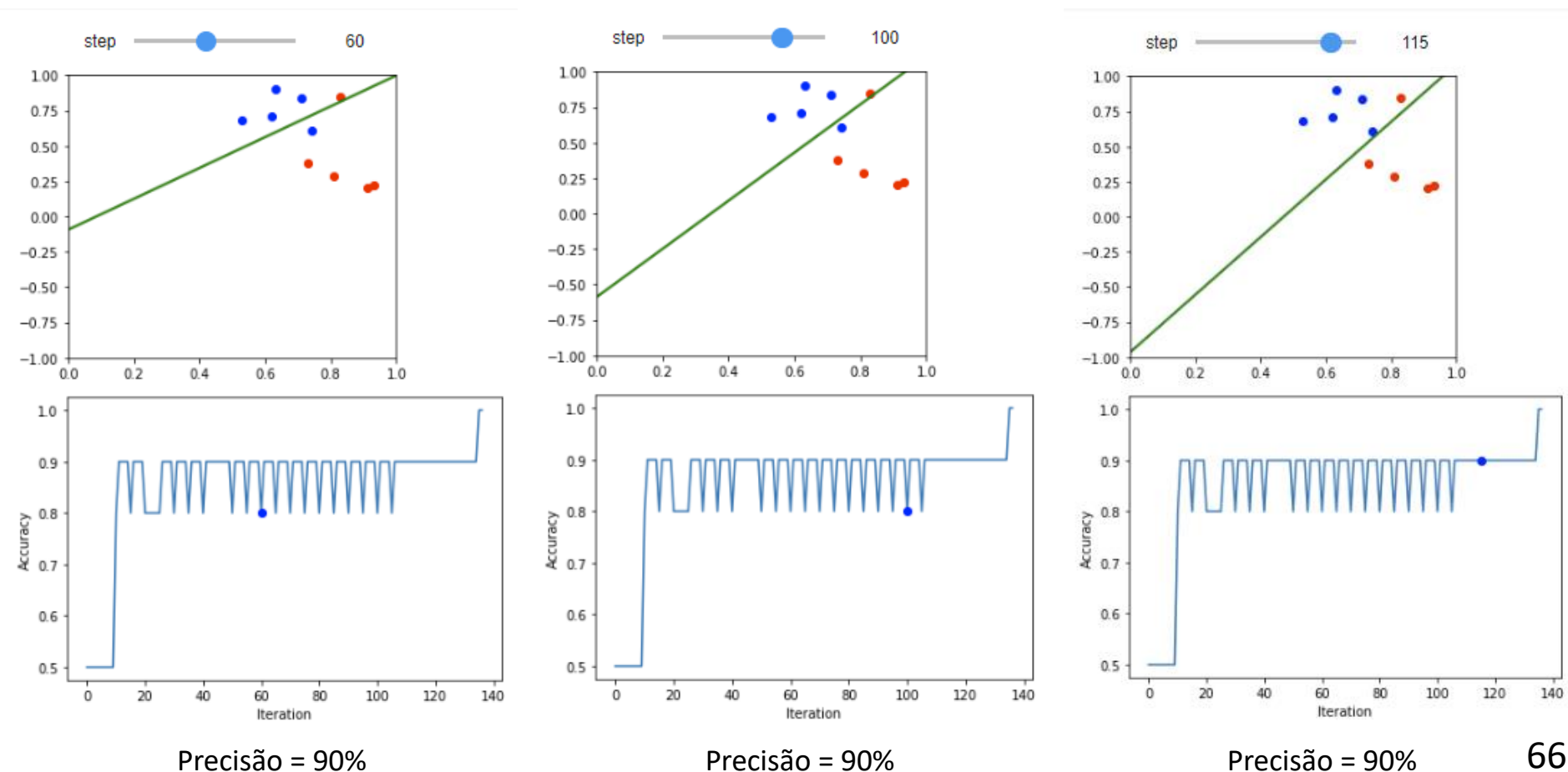
def pl1_all(step): plotit_all(pos_examples,neg_examples,snapshots,step)

interact(pl1_all, step=widgets.IntSlider(value=0, min=0, max=len(snapshots)-1))
```

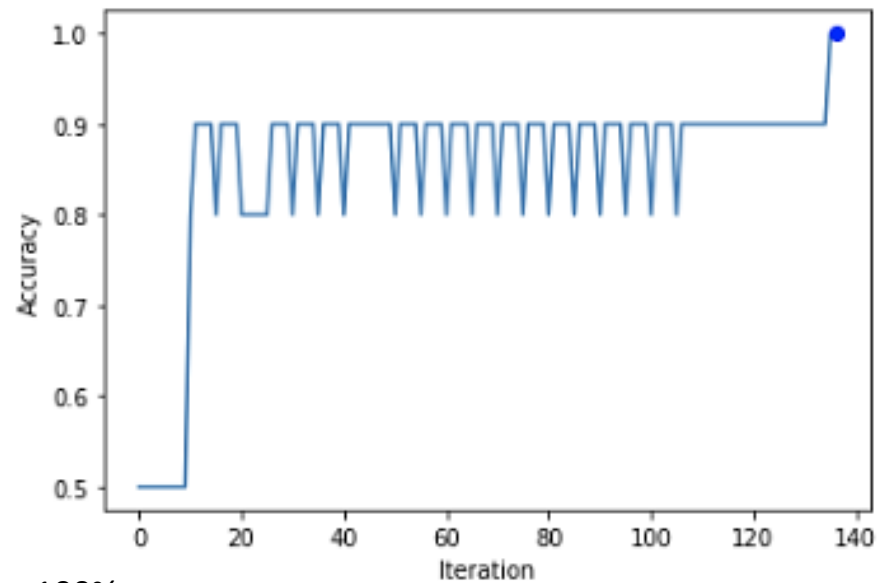
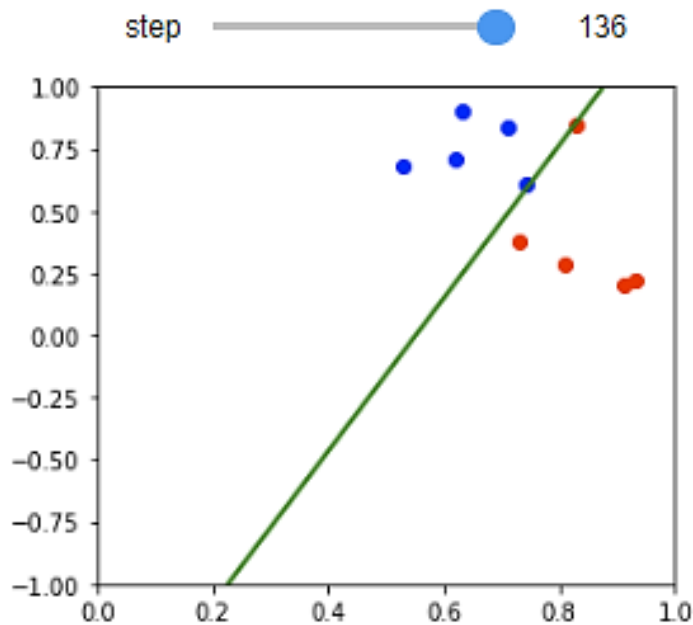
Com a função anterior, podemos obter os seguintes gráficos que nos permitem ver ao longo do tempo como a reta vai sendo alterada e qual a precisão da classificação.



Podemos ainda ver, que apesar da precisão não alterar durante um tempo (mantém-se a 90%), a reta vai alterando, aproximando-se da reta ideal.

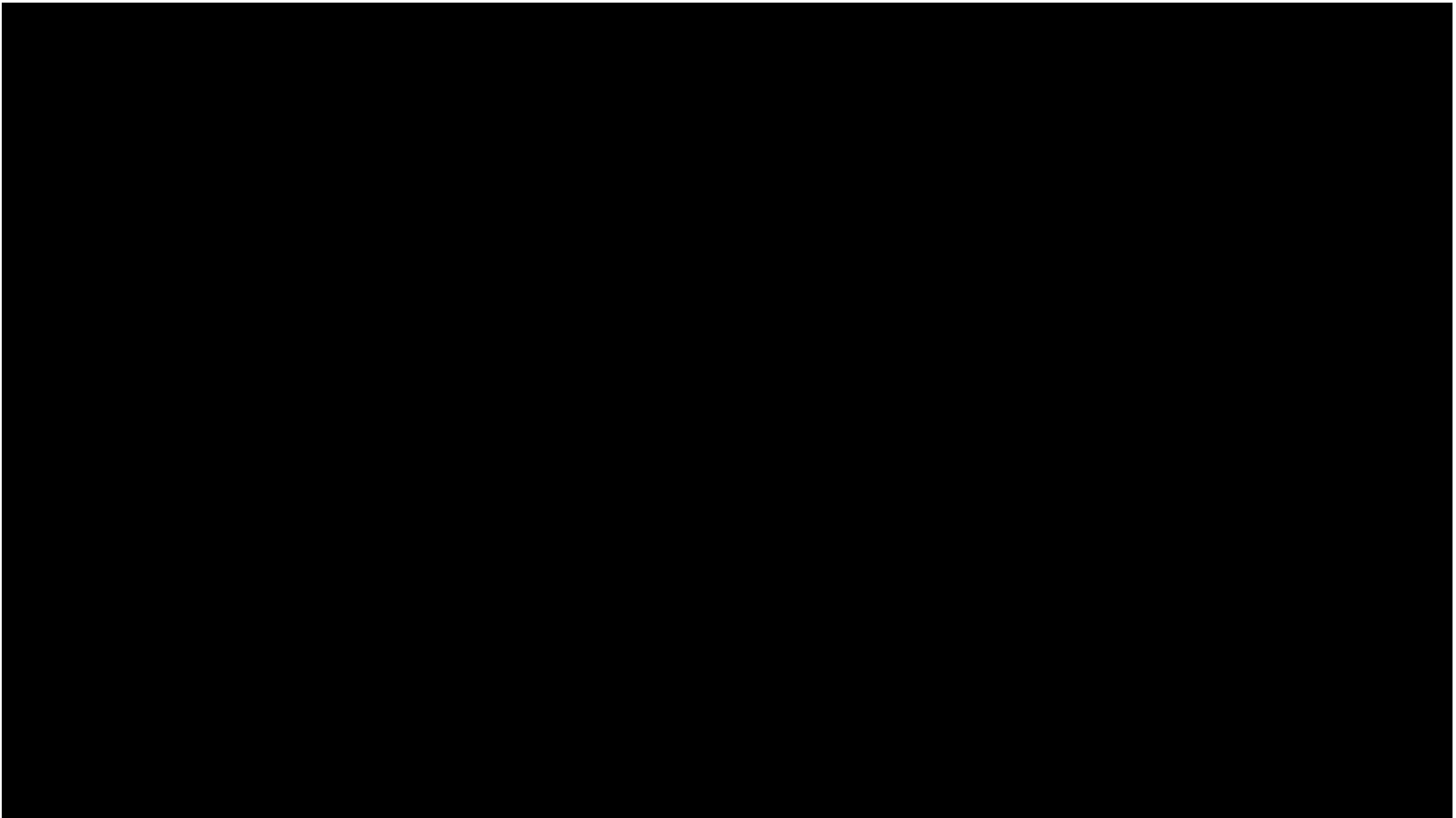


Se colocarmos para visualizar os gráficos finais, vemos o gráfico final e verificamos que tem uma precisão de 1 (100%), todas as frutas estão bem classificadas nesta situação.



Precisão = 100%

De seguida, é possível visualizar a alteração da reta ao longo do tempo, a cada 3 iterações.



# Rede Neuronal Profunda

## Dem4AI

- [https://pt.wikipedia.org/wiki/Garry\\_Kasparov](https://pt.wikipedia.org/wiki/Garry_Kasparov)
- [https://pt.wikipedia.org/wiki/Deep\\_Blue](https://pt.wikipedia.org/wiki/Deep_Blue)
- <https://discord.com/invite/midjourney>
- [https://www.sas.com/pt\\_br/insights/analytics/inteligencia-artificial.html](https://www.sas.com/pt_br/insights/analytics/inteligencia-artificial.html)
- <https://www.analyticsvidhya.com/blog/2020/02/cnn-vs-rnn-vs-mlp-analyzing-3-types-of-neural-networks-in-deep-learning/>
- <https://www.significados.com.br/neuronios/>
- [http://each.uspnet.usp.br/sarajane/wp-content/uploads/2015/05/2015\\_Perceptron\\_McCullochPitts\\_Adaline\\_MLP.pdf](http://each.uspnet.usp.br/sarajane/wp-content/uploads/2015/05/2015_Perceptron_McCullochPitts_Adaline_MLP.pdf)
- <https://ai.stackexchange.com/questions/11567/what-causes-a-model-to-require-a-low-learning-rate>
- <https://towardsdatascience.com/understanding-learning-rates-and-how-it-improves-performance-in-deep-learning-d0d4059c1c10>
- <https://analyticsindiamag.com/most-common-activation-functions-in-neural-networks-and-rationale-behind-it/>
- [https://colab.research.google.com/drive/1PlMueScbRsNeuCcWCm8\\_LuFbOH4rBEIf?usp=sharing](https://colab.research.google.com/drive/1PlMueScbRsNeuCcWCm8_LuFbOH4rBEIf?usp=sharing)
- <https://www.youtube.com/watch?v=aircAruvnKk>