Redes Neurais

Thaís de Almeida Ratis Ramos

Baseado nas aulas de Thaís Gaudencio e Yuri Malheiros - UFPB

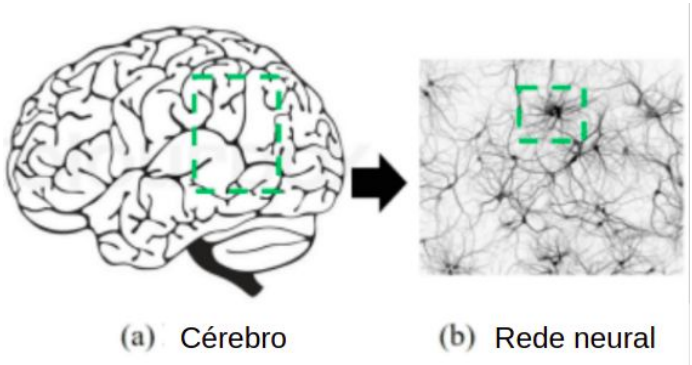
Inspiração biológica

- O cérebro é o principal órgão associado à inteligência e aprendizagem
- O cérebro é composto por uma rede complexa de aproximadamente 100 bilhões de neurônios interconectados
- Existem mais de 500 trilhões de conexões entre neurônios no cérebro humano
- Mesmo as maiores redes neurais artificiais de hoje não chegam perto do cérebro humano

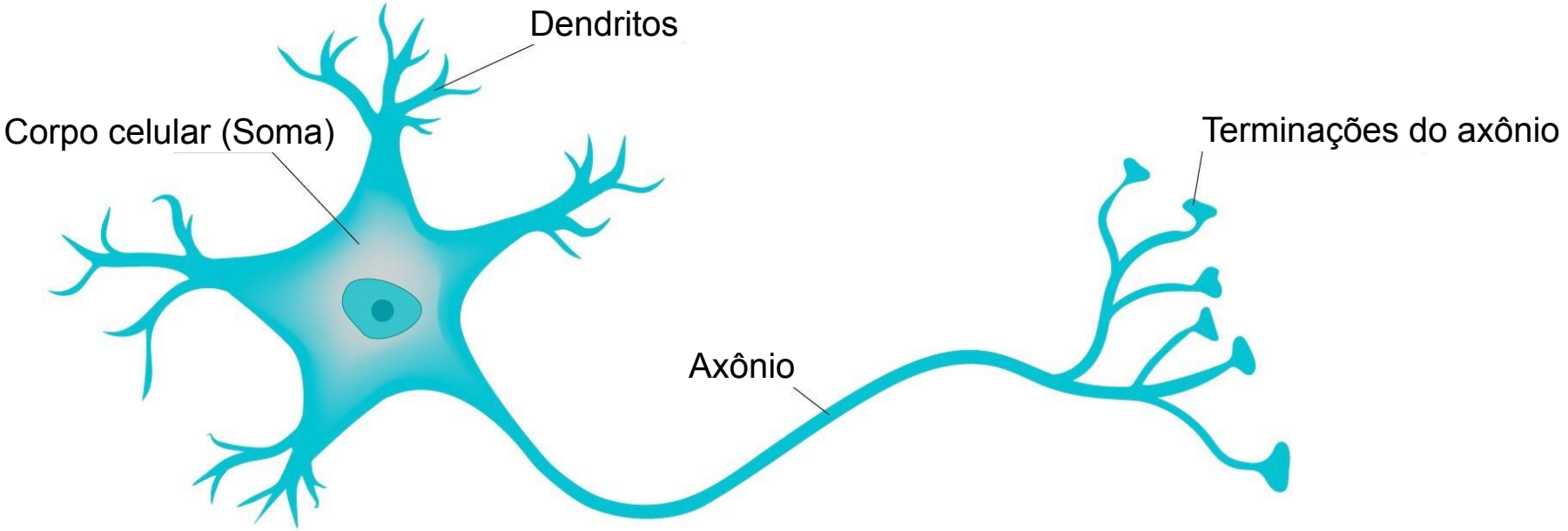
Inspiração biológica

A unidade básica de uma rede neural é o neurônio artificial

Neurônios artificiais são modelados como neurônios biológicos do cérebro, nos quais são estimulados por entradas



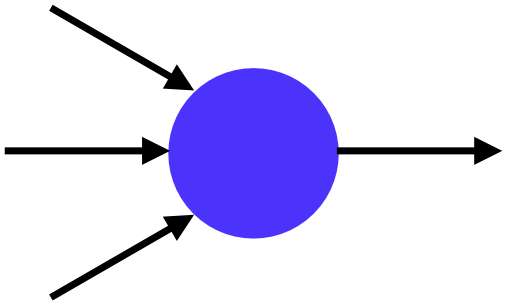
Neurônio

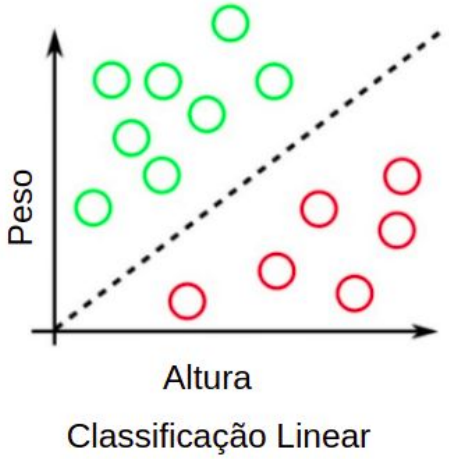

Dendritos
Corpo celular (Soma)
Terminações do axônio
Axônio

Perceptron

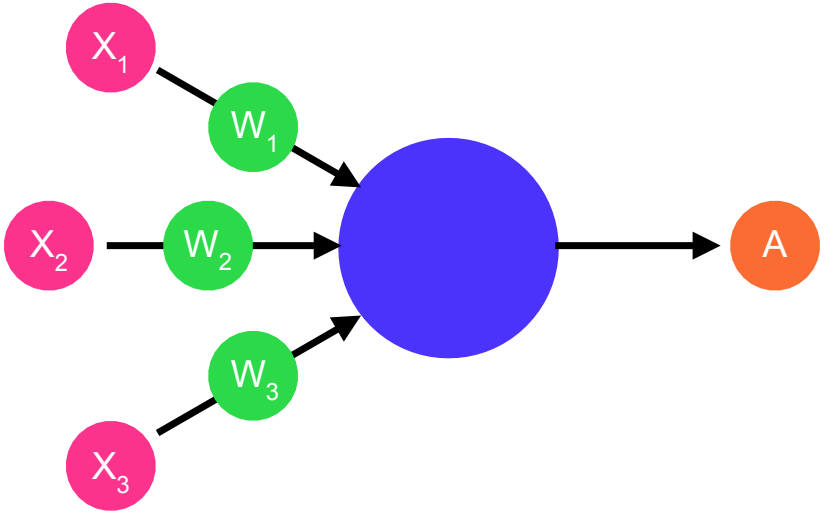
Em 1943, McCullock e Pitts propuseram um modelo de neurônio artificial: **perceptron**

Modelo linear utilizado para classificação binária

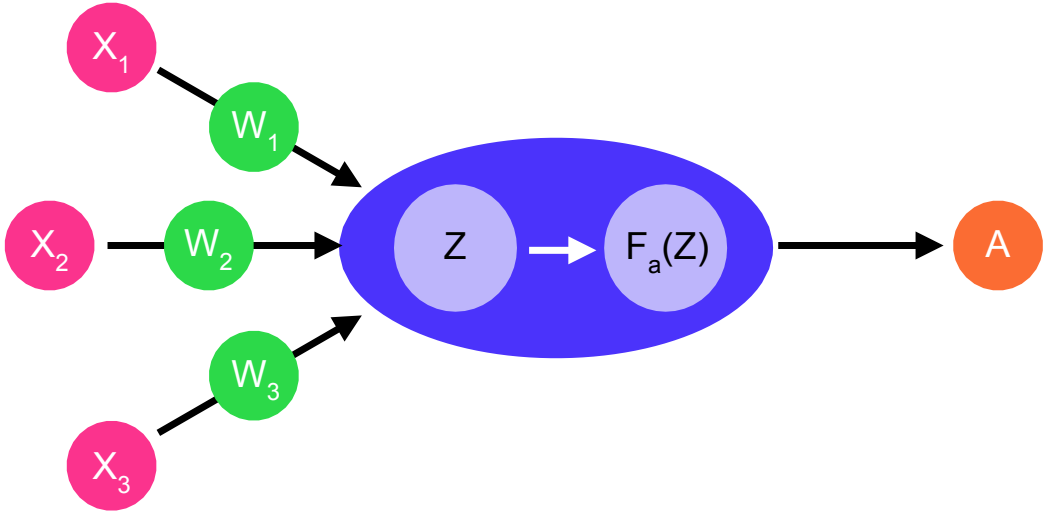




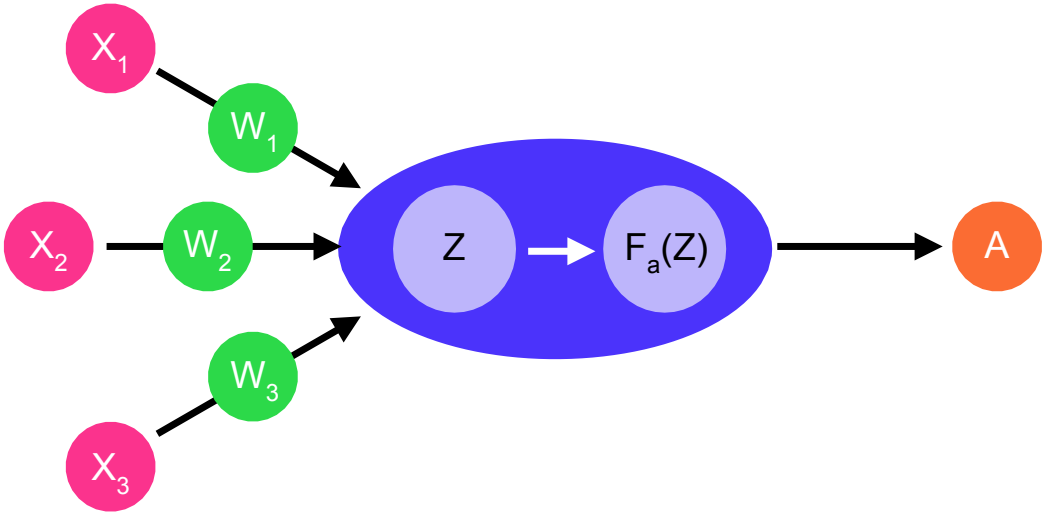
Perceptron



Perceptron

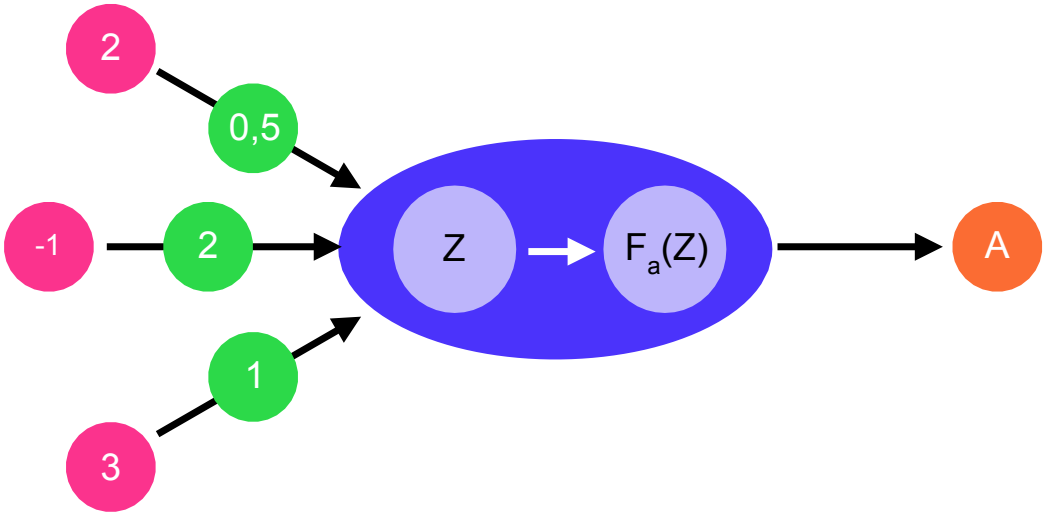


Perceptron



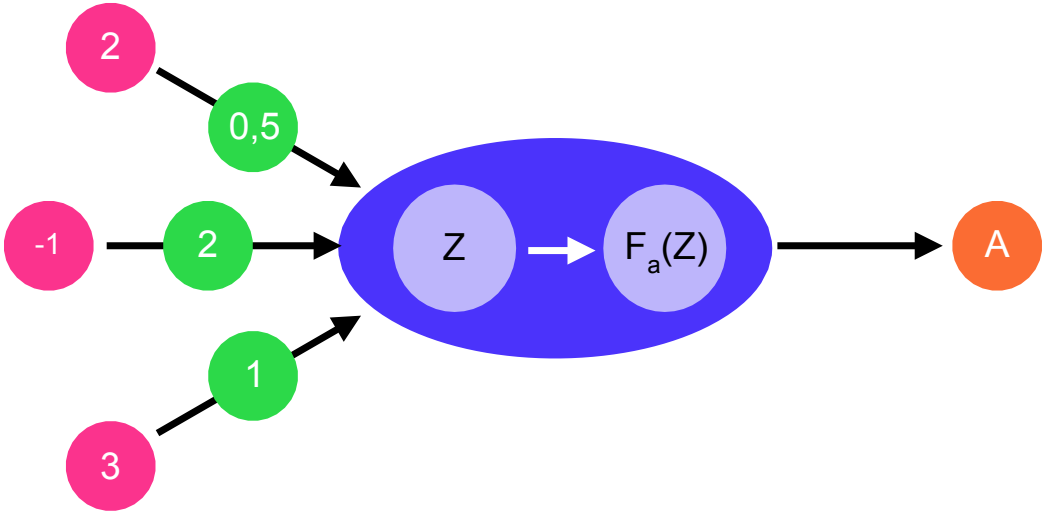
$$Z = W_1 \times X_1 + W_2 \times X_2 + W_3 \times X_3$$

Perceptron



Z = 0,5 x 2 + 2 x -1 + 1 x 3 = 2

Perceptron



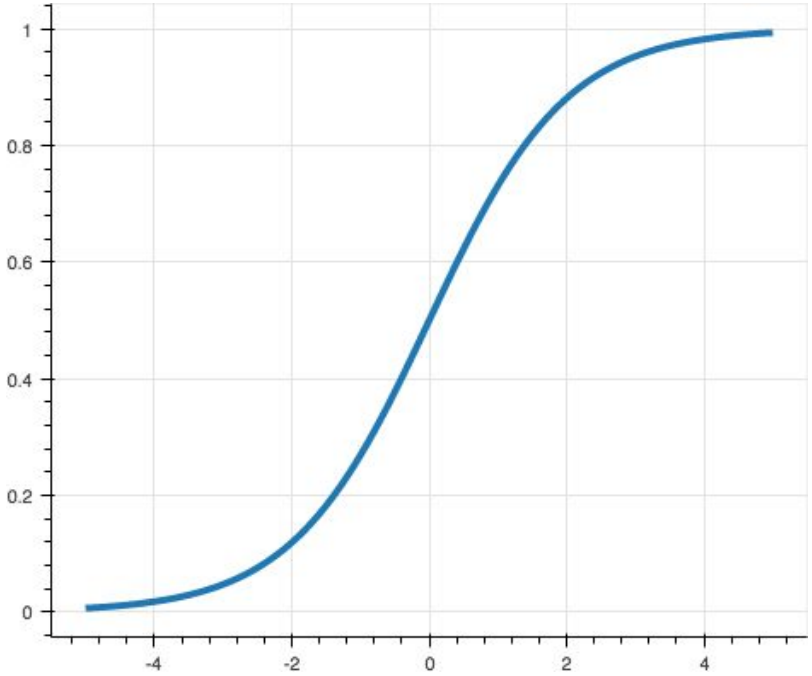
A = $F_a(2)$

Funções de ativação

Várias funções podem ser usadas como função de ativação (F_a)

Função Sigmóide:

$$f_a(z) = \frac{1}{1 + e^{-z}}$$

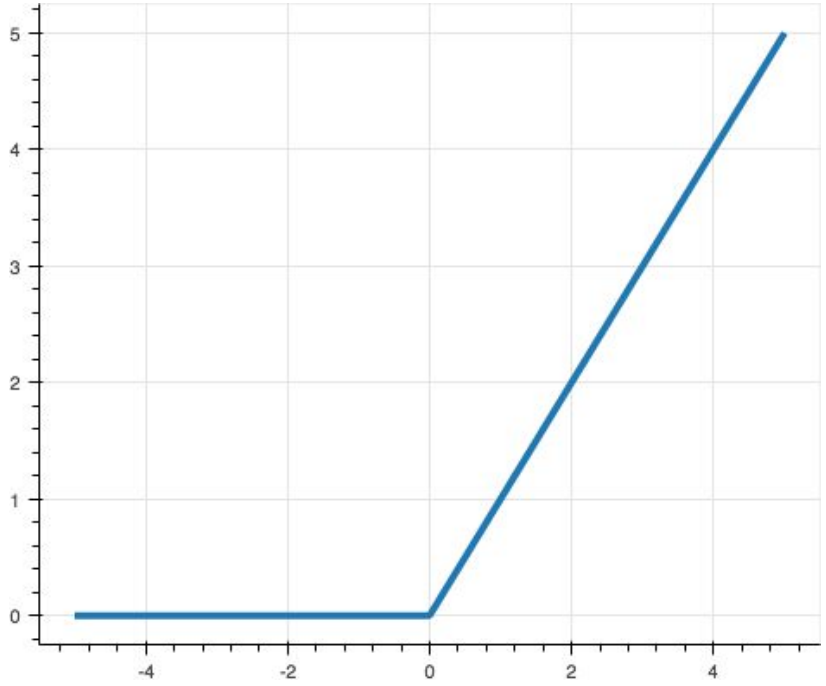


Funções de ativação

Várias funções podem ser usadas como função de ativação (F_a)

Função ReLU:

$f_a(z) = \max(0,z)$



Perceptron

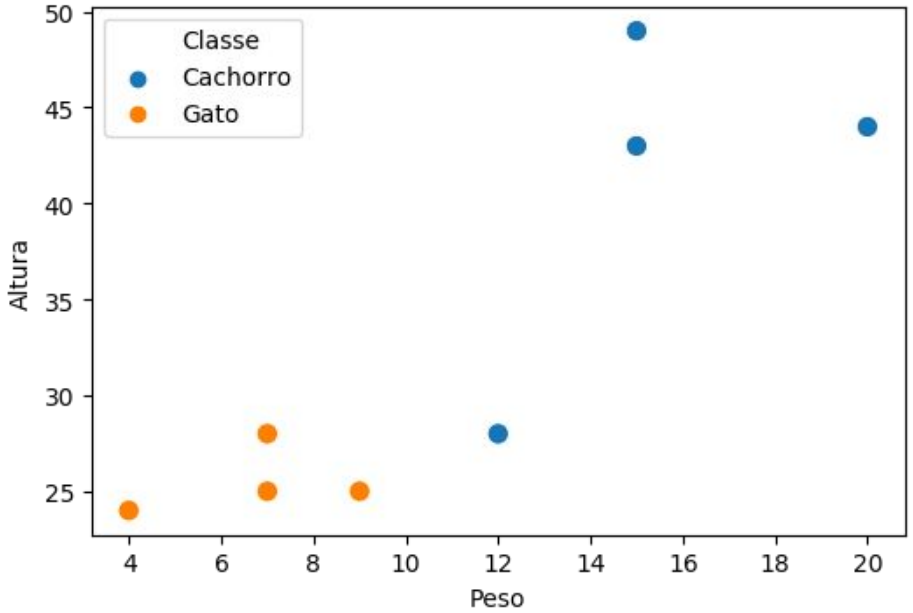
Podemos usar o neurônio artificial para diferentes problemas

Ele nos fornece uma saída de acordo com as entradas

O resultado do processamento das entradas para fornecer uma saída é determinado pela função de ativação e pelos pesos

Exemplo

Gatos e cachorros:



	Peso	Altura	Classe
0	20	44	Cachorro
1	15	43	Cachorro
2	12	28	Cachorro
3	15	49	Cachorro
4	7	28	Gato
5	7	25	Gato
6	9	25	Gato
7	4	24	Gato

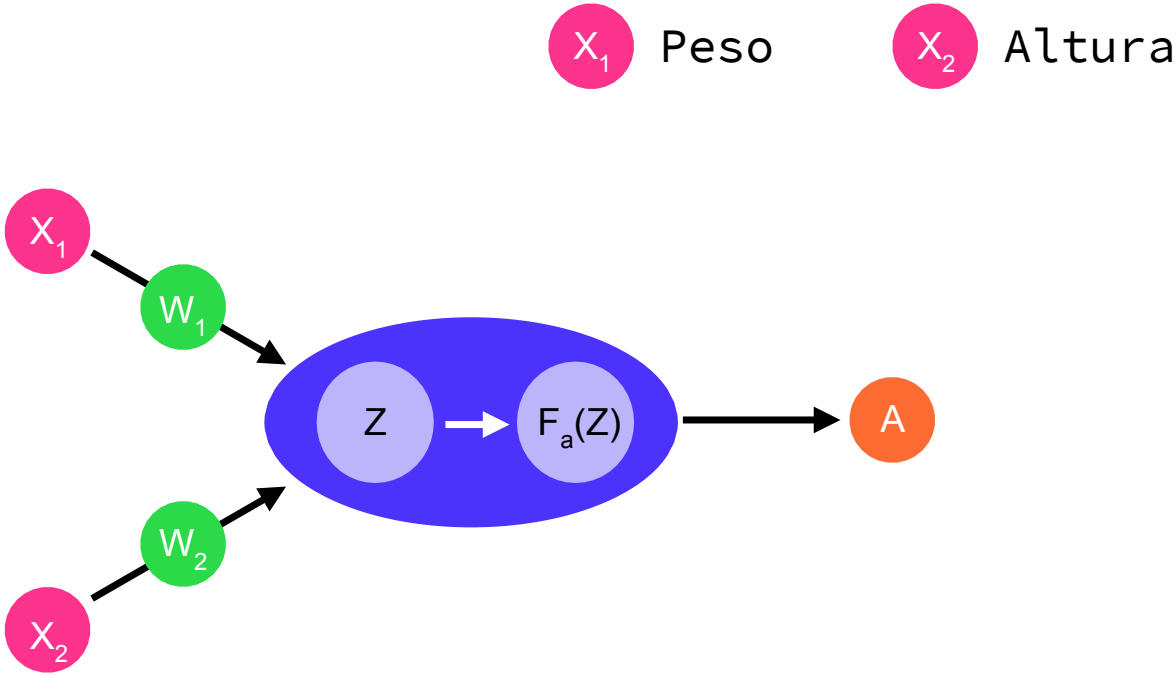
Exemplo

Dada a altura e o peso, vamos fazer o neurônio ter:

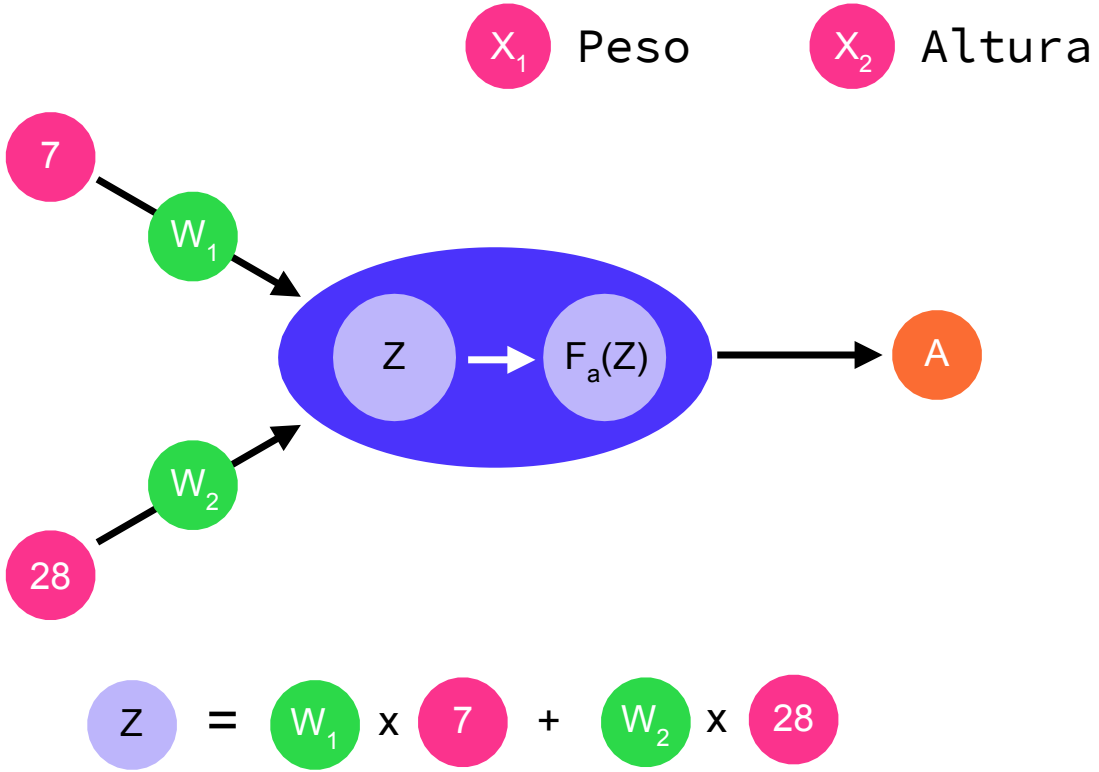
Saída **0** se for um gato

Saída **1** se for um cachorro

Exemplo



Exemplo



Treinando uma rede Neural

Quais são os valores de **w**?

Podemos tentar encontrá-los manualmente, porém este é um processo complexo e muitas vezes inviável

Treinando uma rede Neural

Como não sabemos os pesos iniciais, vamos iniciá-los aleatoriamente

$w_1 = 0,25$

$w_2 = 0,66$

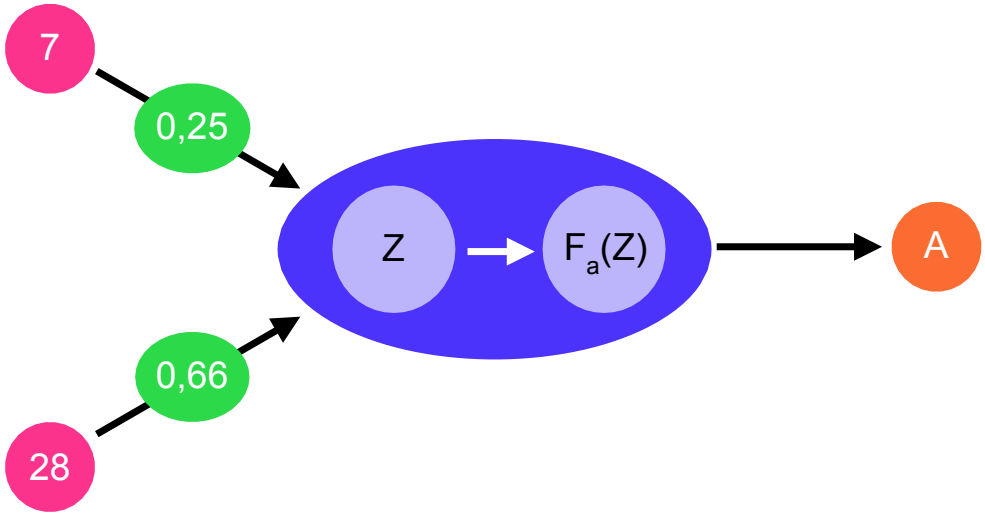
Em seguida, vamos calcular a saída para

$x_1 = 7$

$x_2 = 28$

Treinando uma rede Neural

Saída 0: gato
Saída 1: cachorro



$$Z = 0{,}25 \times 7 + 0{,}66 \times 28 = 20{,}23$$

$$A = F_a(20{,}23) = 0{,}99$$

Treinando uma rede Neural

Esperado	Obtido	Ajuste	Esperado - Obtido
0	0	-	0
0	1	Diminuir W	-1
1	0	Aumentar W	1
1	1	-	0

Treinando uma rede Neural

Para isso, eu preciso saber o erro, ou seja a diferença entre o valor esperado e o valor obtido

Quanto maior o erro, maior precisa ser o ajuste dos valores de **w**

Então, vamos atualizar w_1 da seguinte forma:

$$w_1 \leftarrow w_1 + \eta\,(esperado - obtido)\,x_1$$

Taxa de aprendizagem: controla o tamanho da atualização

Treinando uma rede Neural

Cada ajuste vai melhorando um pouco a rede neural

Vamos ajustar os pesos para cada um dos exemplos dos dados de treinamento

$$w_1 \leftarrow w_1 + \eta(esperado - obtido)x_1$$

$$w_2 \leftarrow w_2 + \eta(esperado - obtido)x_2$$

Treinando uma rede Neural

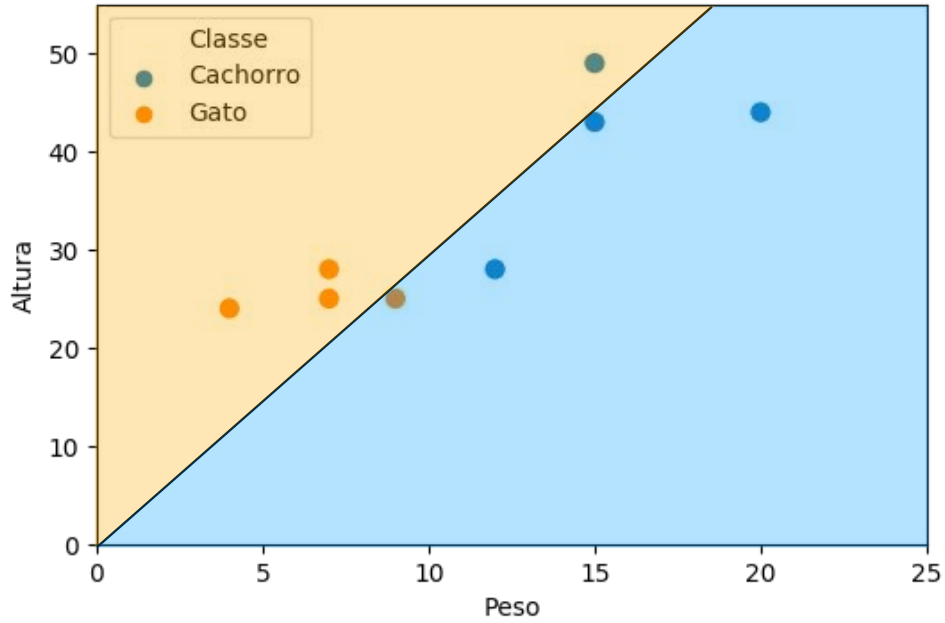
Em muitos casos, fazer uma atualização para cada exemplo pode não ser suficiente

Por isso, costumamos atualizar para cada exemplo e em seguida refazer o processo várias vezes

Esse número de vezes é chamado de **épocas**

Exemplo

Temos como resultado:



Treinando uma rede Neural

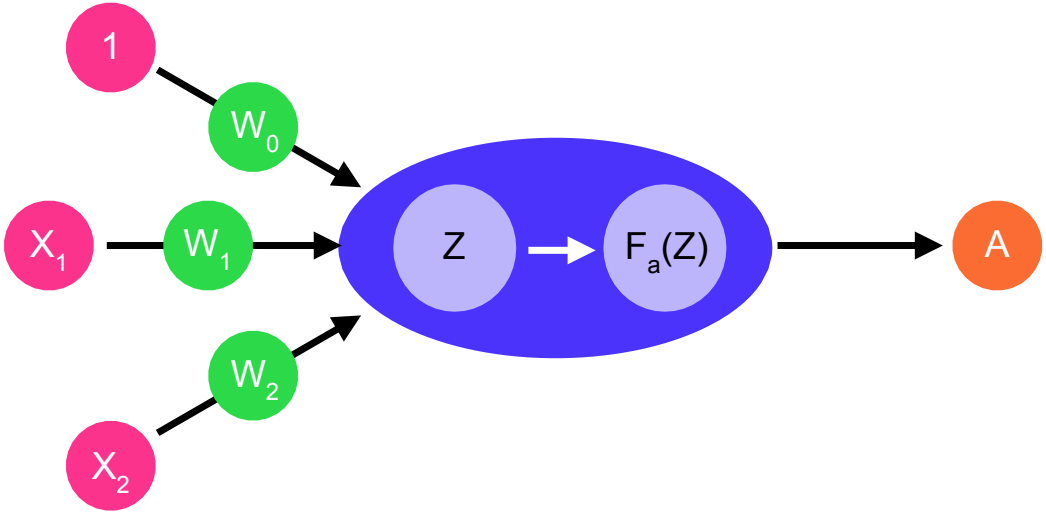
Usando o neurônio com a estrutura que fizemos, sempre teremos uma fronteira linear partindo da origem

Para permitir que a fronteira possa se deslocar no eixo X, vamos adicionar uma entrada extra no neurônio chamada de **viés**

O viés geralmente recebe 1 como entrada

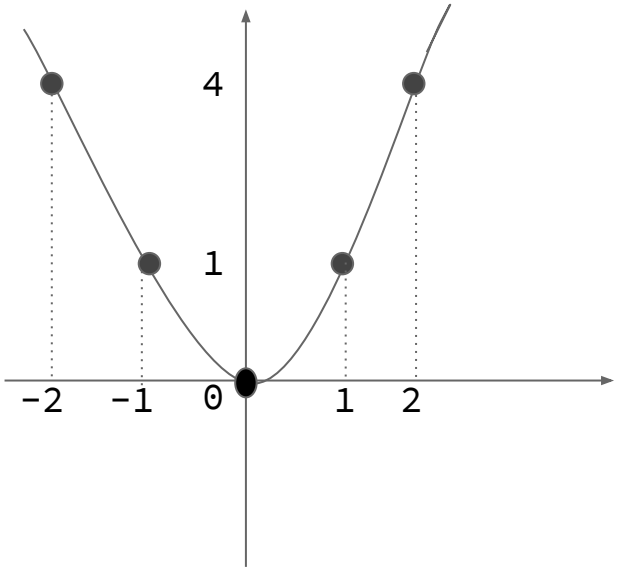
O peso dessa entrada segue o comportamento das outras

Treinando uma rede Neural



Viés (bias)

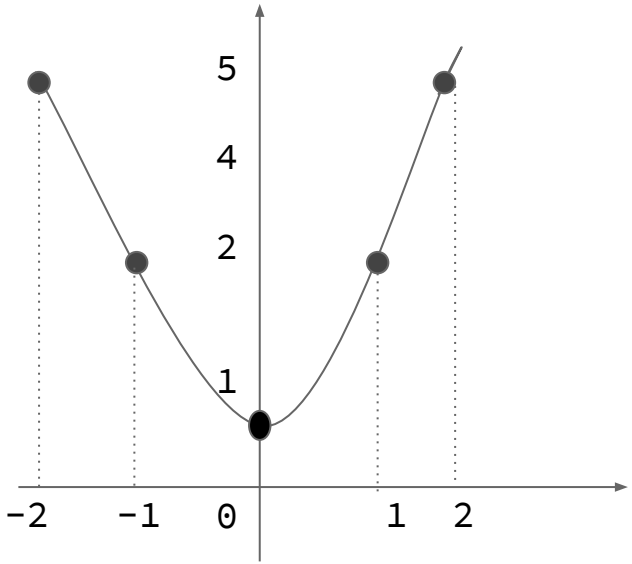
b = 0



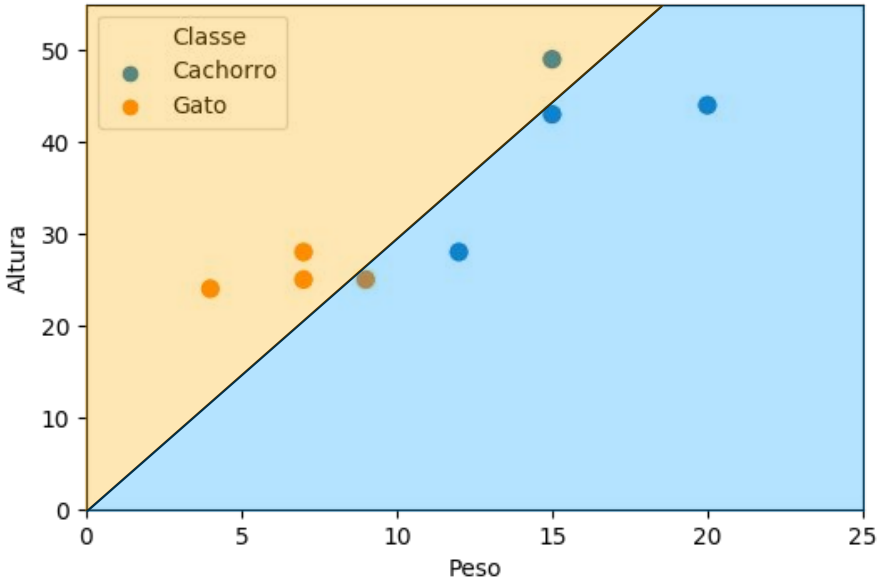
$f(x) = x^2+b$

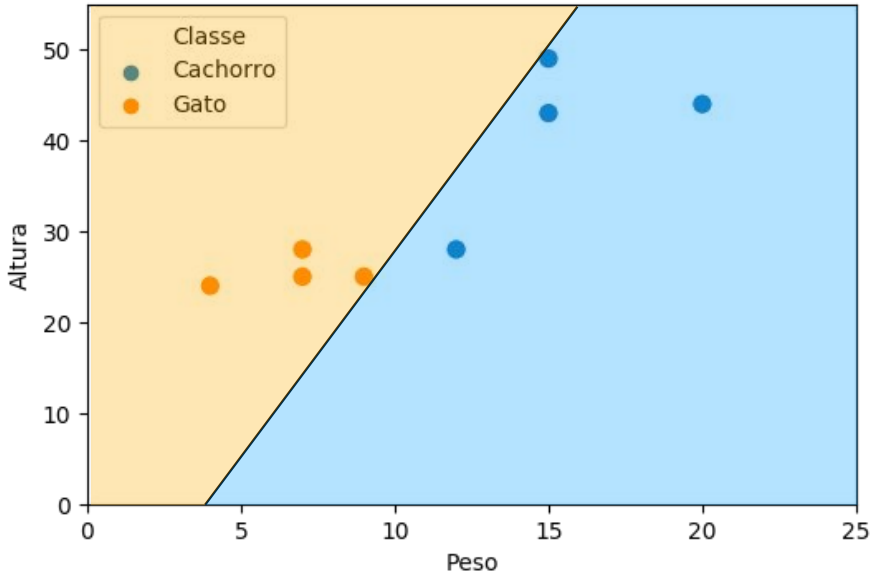
x	f(x) b=0	f(x) b=1
-2	4	5
-1	1	2
0	0	1
1	1	2
2	4	5

b = 1



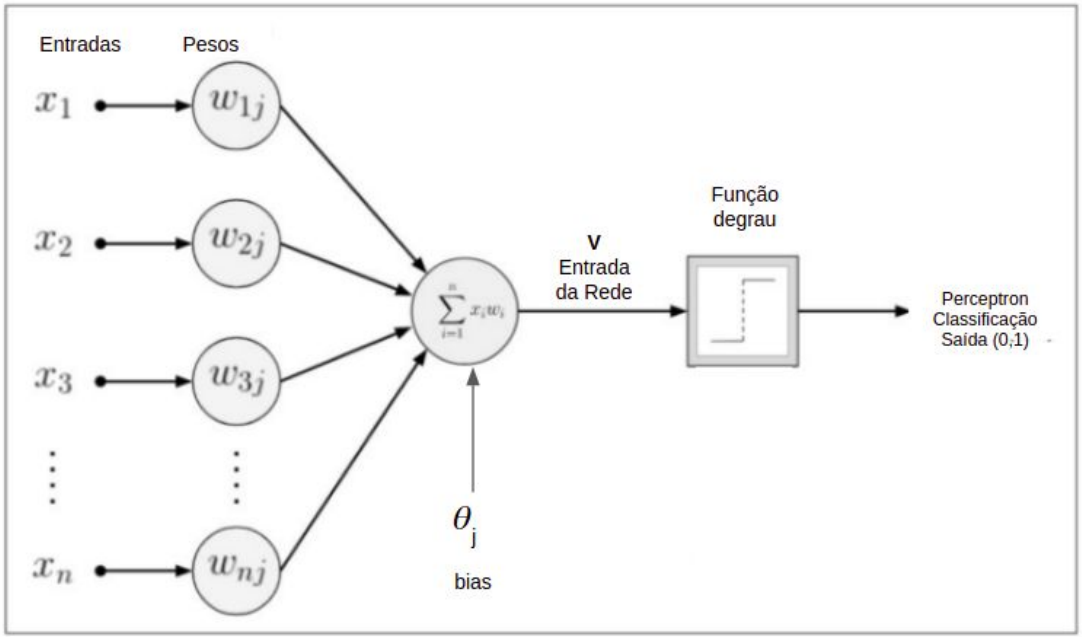
Viés (Bias)


Classe
Cachorro
Gato
Altura
Peso
0
10
20
30
40
50
0
5
10
15
20
25


Classe
Cachorro
Gato
Altura
Peso
0
10
20
30
40
50
0
5
10
15
20
25

ESTRUTURA DO PERCEPTRON

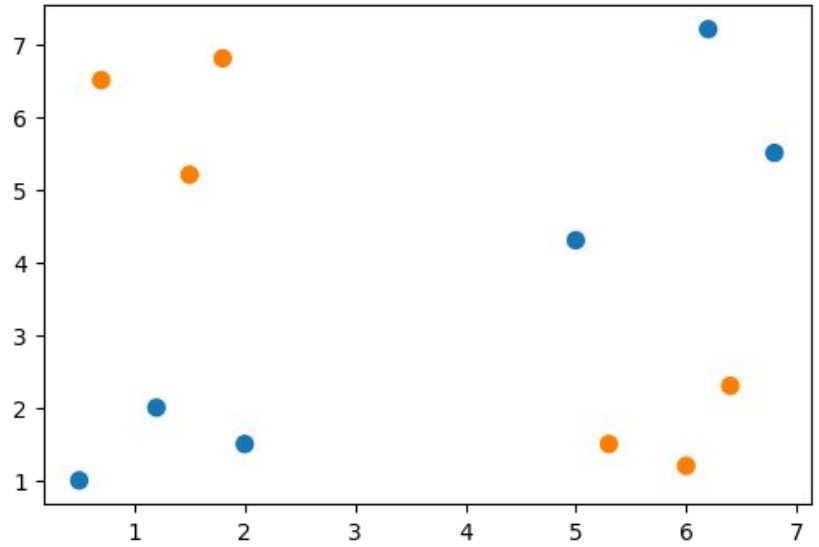
Função de ativação de Heaviside: $f(v) = \begin{cases} 0 & v < 0 \\ 1 & v >= 0 \end{cases}$



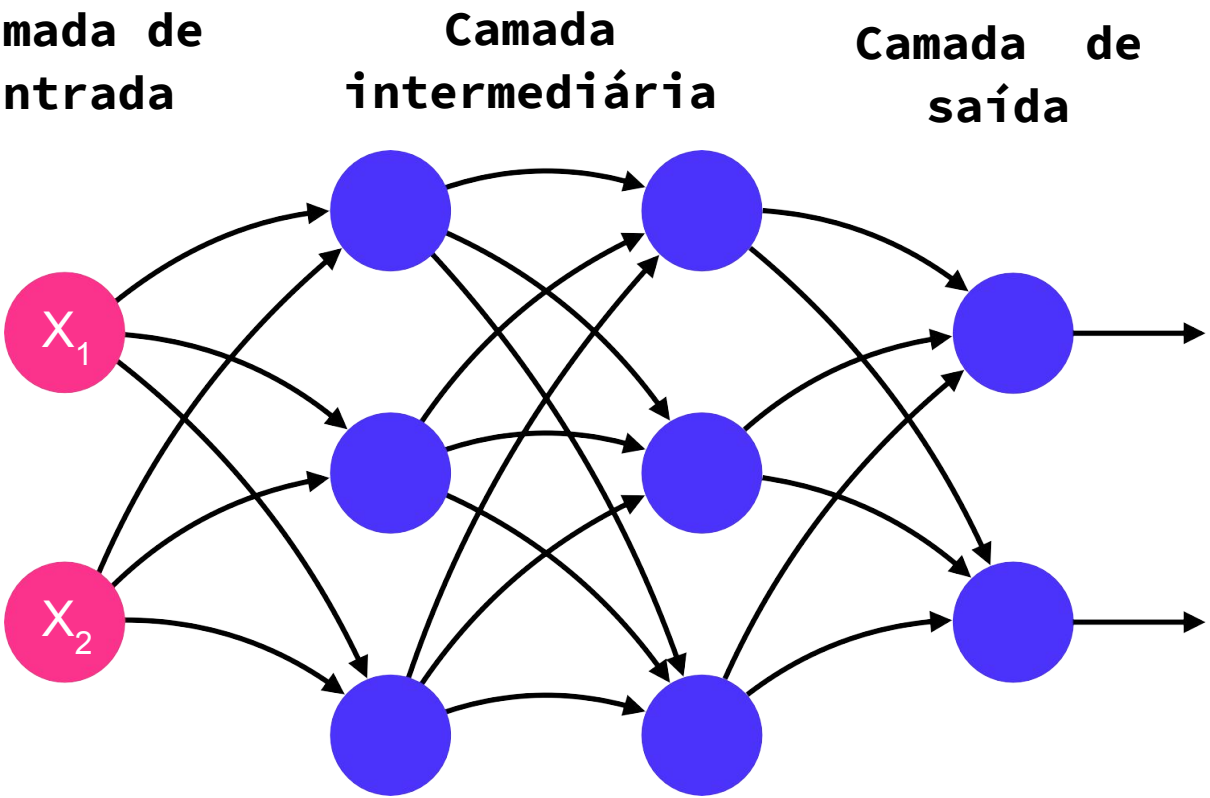
Treinando uma rede Neural

Como vimos, o perceptron possui a limitação de criar apenas fronteiras lineares

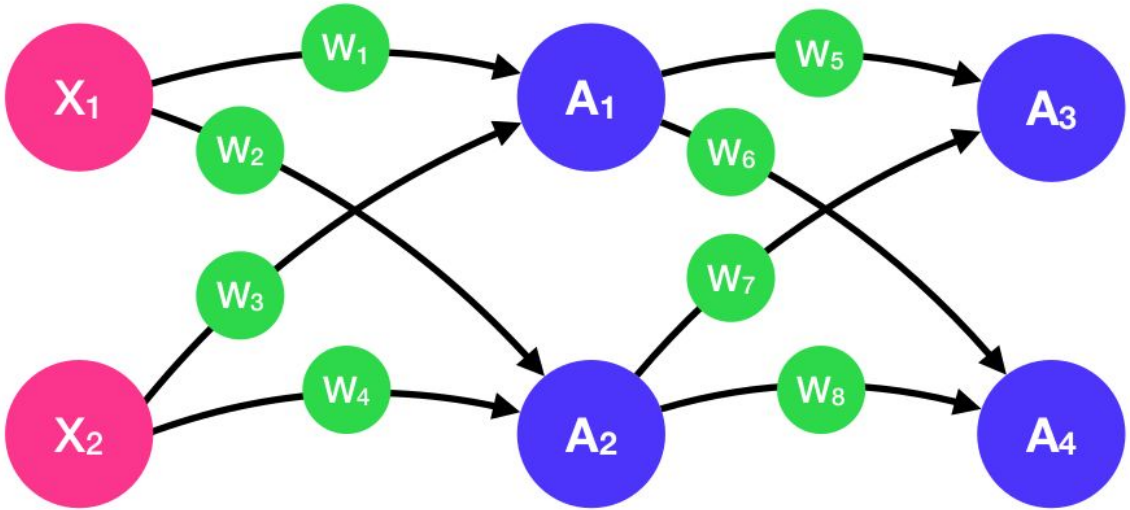
Ele não consegue classificar bem um conjunto de dados dessa forma:



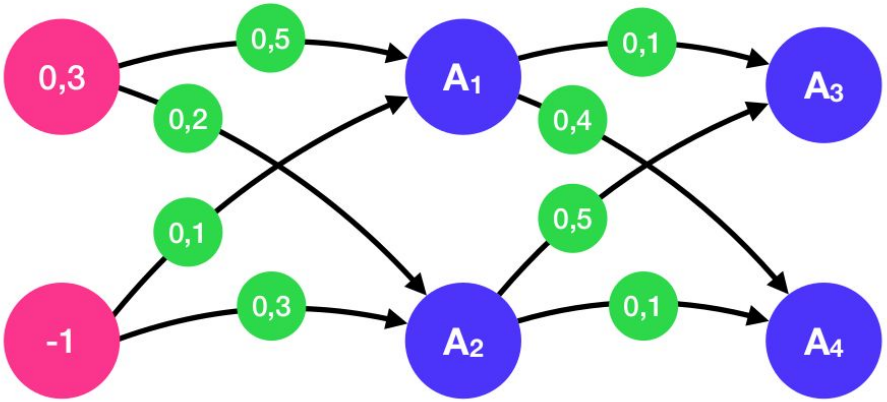
Rede neural


Camada de entrada
Camada intermediária
Camada de saída
X1
X2

Rede neural



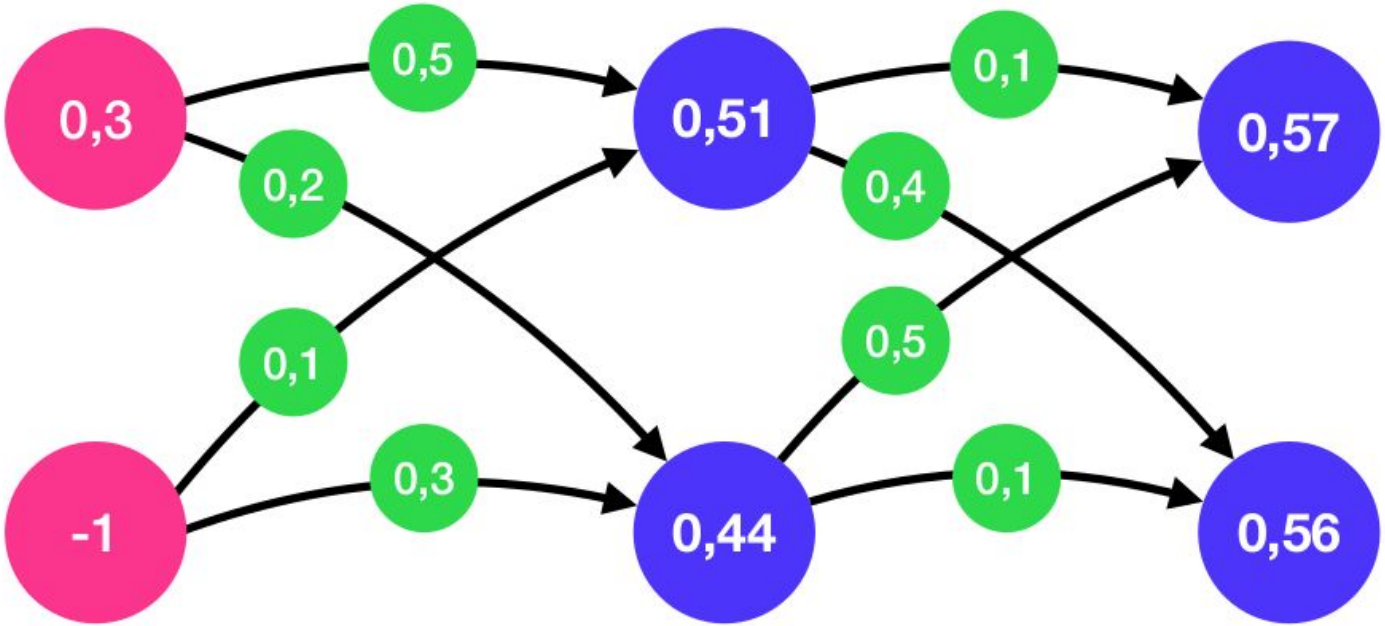
Rede neural



$Z_1 = 0{,}5 \times 0{,}3 + 0{,}1 \times -1 = 0{,}049$

$A_1 = F_a(0{,}049) = 0{,}51$

Rede neural



Treinando uma Rede neural

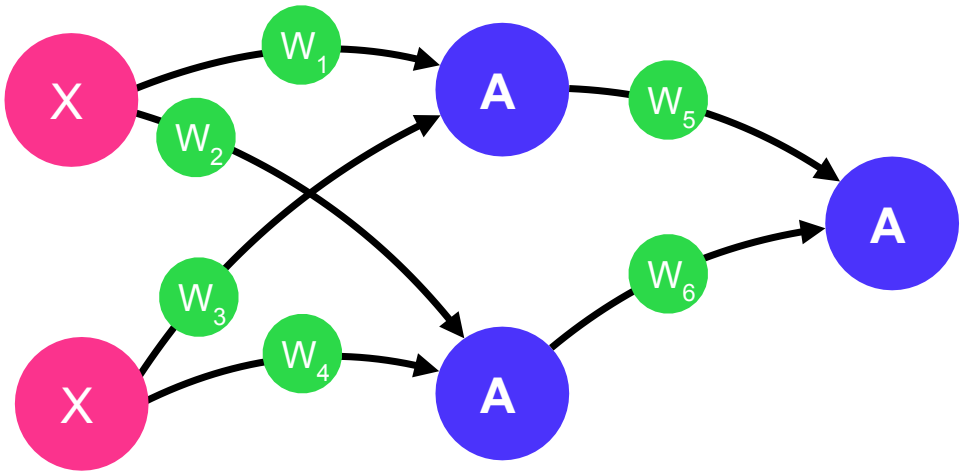
O algoritmo é semelhante ao que fizemos para apenas um neurônio

Iniciamos os pesos de forma aleatória

Entramos com os dados de treinamento e comparamos o resultado da rede com o resultado esperado

Ajustamos os pesos de acordo com o erro

Treinando uma Rede neural



O valor de A_3 é a saída da rede. Podemos compará-lo diretamente com o resultado dos dados de treinamento

E para A_1 e A_2? Quais são os valores esperados? Não temos esses dados

Treinando uma Rede neural

Para treinar uma rede com multicamadas usamos o algoritmo backpropagation

Nele, vamos definir uma função que mede o erro da saída da rede

Também conseguimos calcular através do gradiente da função de erro, em qual sentido devemos ajustar os pesos para que o erro diminua

Correção de erros

Calculamos o erro para um exemplo através da função:

$$e_k = d_k - y_k$$

Onde:

e - Sinal de erro
d - Saída desejada apresentada durante o treinamento
y - Saída real da rede

Aprendizado por Correção de erros

- O processo de aprendizado por correção de erros utiliza algoritmos para caminhar sobre a curva de erros, com o intuito de alcançar o menor valor de erro possível, o mínimo global
- Muitas vezes o algoritmo não alcança este mínimo global, atingindo o que chamamos de mínimo local. Caso este erro alcançado seja desfavorável, é necessário recomeçar o processo de aprendizagem

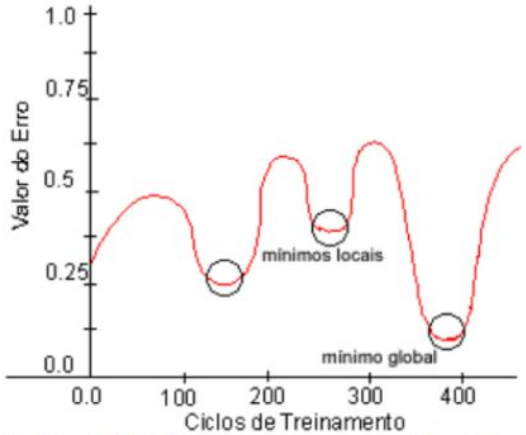


Gráfico de uma possível superfície de erro mostrando os mínimos locais e o mínimo global

Aprendizado por Correção de erros

Para correção do erro, os pesos devem ser ajustados de forma a aproximar a saída real à desejada

$$\Delta wi(n) = \eta\, e(n)\, xi(n)$$

Onde:

△wi(n) - Valor de ajuste a ser acrescido ao peso wi;

n - Taxa de aprendizagem (constante positiva);

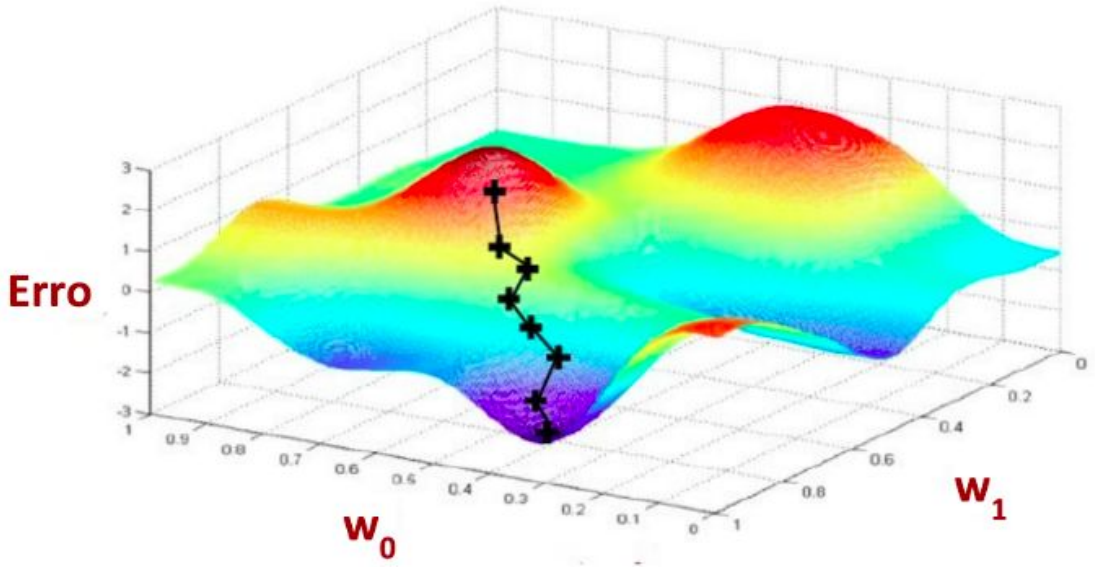
e(n) - Valor do erro;

xi - Valor da entrada

Encontrar o vetor (matriz) de pesos sinápticos (w*) que minimizem o erro (e) entre a saída da rede neural e a saída desejada

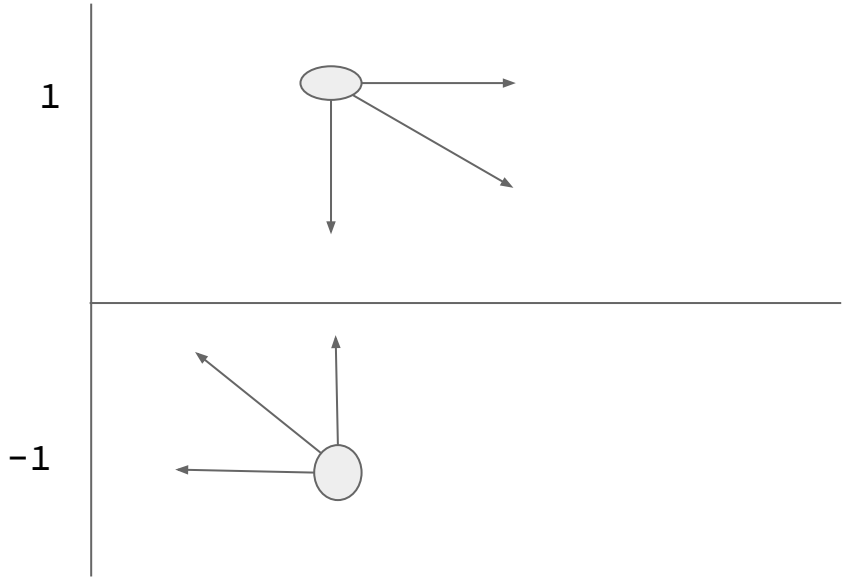
Método do Gradiente

Vetor que indica o sentido e a direção onde a função crescerá mais rápido



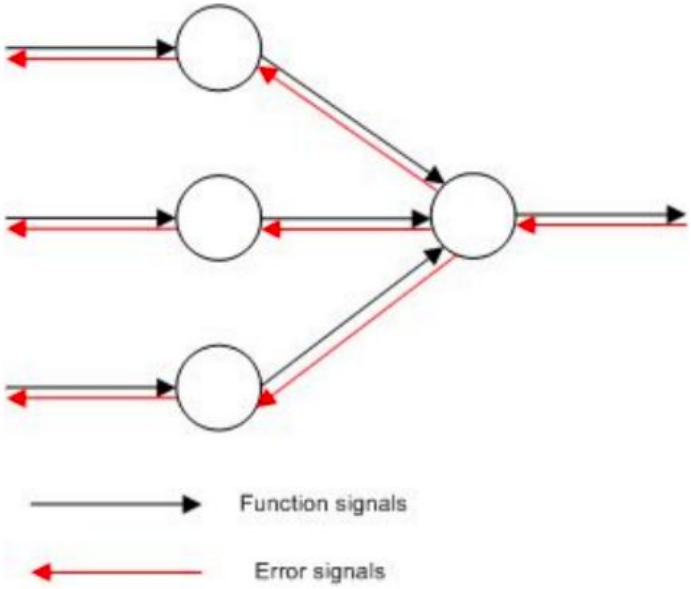
Método do Gradiente

Vetor que indica o sentido e a direção onde a função crescerá mais rápido



Backpropagation

Sinal de erro se origina em um neurônio de saída e se propaga para trás (camada por camada) através da rede



Correção dos pesos - Regra delta

$$\Delta w_{ji}(n) = \eta * \delta j(n) * y_i(n)$$

- $\Delta w_{ji}(n)$: Correção de peso
- η: Taxa de aprendizagem
- $\delta j(n)$: Gradiente local
- $y_i(n)$: Sinal de entrada do neurônio *j*

Gradiente Local

Camada de saída

$$\delta_j(n) = e_j * \varphi'(v_j(n))$$

Gradiente Local

Sinal de erro na saída de *j*

Derivada da função de ativação no neurônio j

Camada oculta

$$\delta_j(n) = \varphi'\left(v_j(n)\right) \sum_k \delta_k(n) w_{kj}(n)$$

Gradiente Local

Derivada associada ao neurônio j

Soma ponderada dos gradientes locais da camada seguinte k

Backpropagation

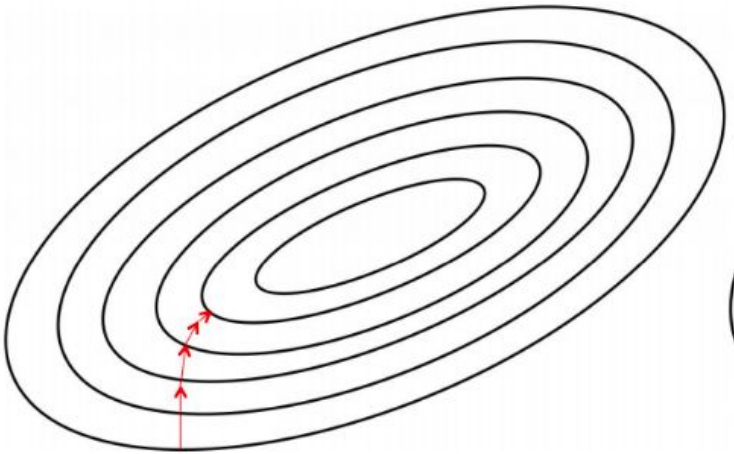
```
function backpropagation-algorithm
    (network, training_examples, learning_rate)
    network <- initialize_weights(randomly)
    start loop
        for each example in training-examples do
            // Computação para frente (Propagação)
            network_out = rna_output(network, example)

            // Computacao para trás (Retropropagação)
            example_err = actual_out - network_out
            local_gradient = calc_gradient(example_err)
```

Backpropagation

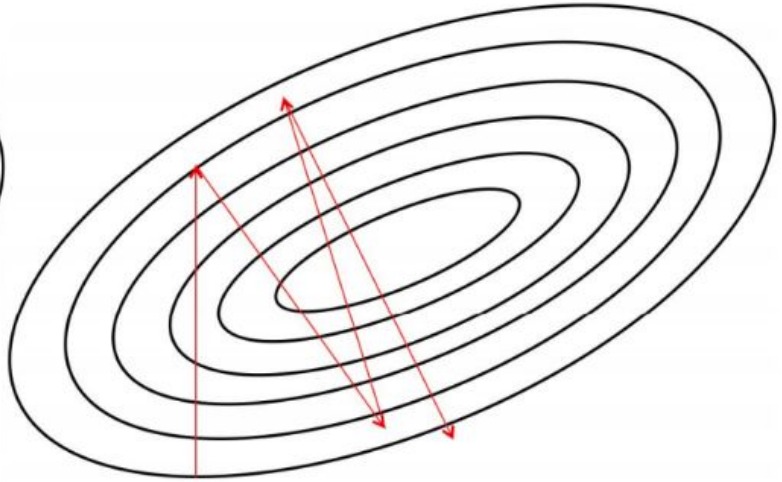
```
        Atualiza os pesos dos neurônios da camada de
        saída j (regra delta)

        for each previous-layer in network do
            Compute o erro em cada nó
            Atualiza os pesos dos neurônios da camada
            (regra delta)
        end for

    end for
end loop quando rede convergir
return network
```

Taxa de aprendizagem

η muito pequeno
Aprendizagem e convergência lenta



η muito grande
Rede instável



Regra delta com termo do momentum

Permite aumentar a taxa de aprendizado, evitando risco de instabilidade

Leva em consideração a variação dos pesos da iteração anterior:

- Variação de pesos com o mesmo sinal: Acelera a descida
- Variação de pesos com sinais opostos: Reduz o ajuste nos pesos (estabiliza o algoritmo)

Pode evitar que o processo de aprendizagem termine em um mínimo local raso

$$\Delta w_{ji}(n) = \alpha \, \Delta w_{ji}(n-1) + \eta \delta_j(n) y_i(n)$$

Constante do momento

Variação de pesos da última iteração

Regra delta

$$0 \leq \alpha < 1$$

Modos de treinamento: Estocástico, por lote e mini-lote

Modos de treinamento

Baseado na forma de atualização dos pesos, temos 3 modos de treinamento:

- Estocástico - Ajuste de pesos é realizado após a apresentação de cada exemplo
- Por lote (batch) - Ajuste de pesos é realizado após a apresentação de todo os exemplos à rede (fim da época)
- Mini-lote (mini-batch) - Ajuste de pesos é realizado após a apresentação de um subconjunto de exemplos

Estocástico

Vantagens:

- Ajuda a escapar de mínimos locais
- Mais simples de implementar
- Pode tirar vantagens de dados (exemplos) redundantes

Desvantagens:

- Estimar o erro baseado em um único exemplo não é uma boa aproximação do erro real
- Treinamento muito lento
- Mais difícil provar teoricamente que o algoritmo converge

Por lote (batch)

Vantagens:

- Estimativa precisa do gradiente
- Convergência mais rápida sob condições simples
- Mais fácil de paralelizar

Desvantagens:

- Pode ficar preso em mínimos locais

Por mini-lote (mini-batch)

Bom balanço entre o modo estocástico e o modo por lote:

- Convergência mais rápida (modo por lote)
- Evita mínimos locais

Critérios de parada

Existem alguns critérios razoáveis para a convergência:

- Vetor gradiente alcançar um limiar suficiente pequeno
- Taxa de variação do erro muito pequena entre as épocas (ex: menor que 1%)
- Rede apresenta um bom desempenho de generalização, ou seja, funciona bem com um outro conjunto de exemplos (conjunto de validação)