

Redes Neurais

Prof. Luiz Alberto Bordignon

Neurônio artificial

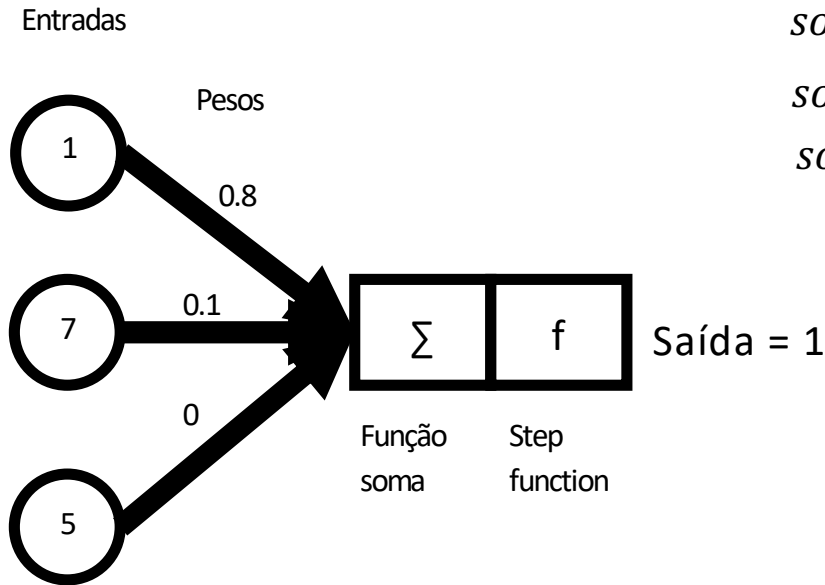
primeira parte

$$soma = \sum_{i=1}^n x_i * w_i$$

$$soma = (1 * 0.8) + (7 * 0.1) + (5 * 0)$$

$$soma = 0.8 + 0.7 + 0$$

$$soma = 1.5$$



segunda parte

Step function (função Degrau)

Maior ou igual a um = 1
Caso contrário = 0

processo básico de uma rede neural, multiplica as entradas pelos pesos, faz o somatório, e depois aplica uma função de ativação

Neurônio artificial

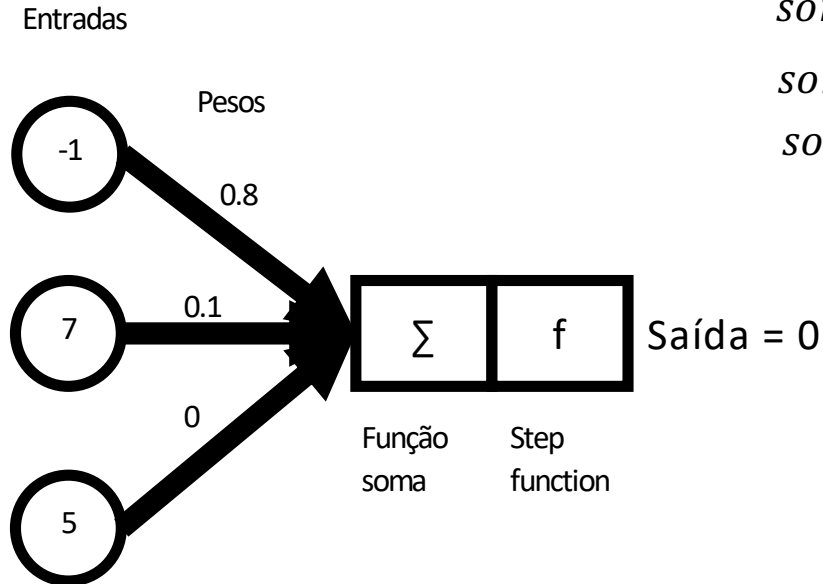
$$soma = \sum_{i=1}^n x_i * w_i$$

$$soma = (-1 * 0.8) + (7 * 0.1) + (5 * 0)$$

$$soma = -0.8 + 0.7 + 0$$

$$soma = -0.1$$

saída negativa, valor = 0

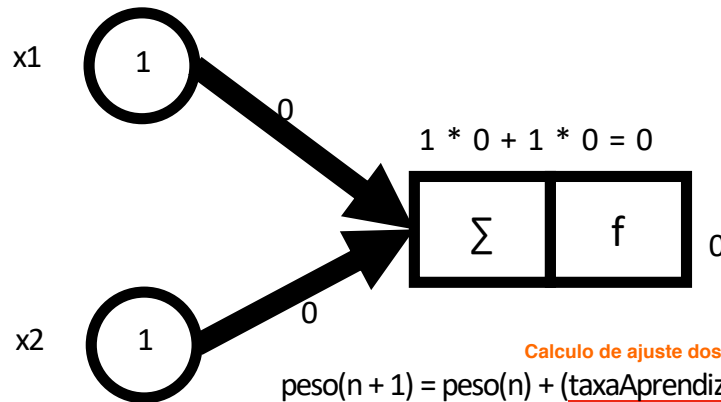
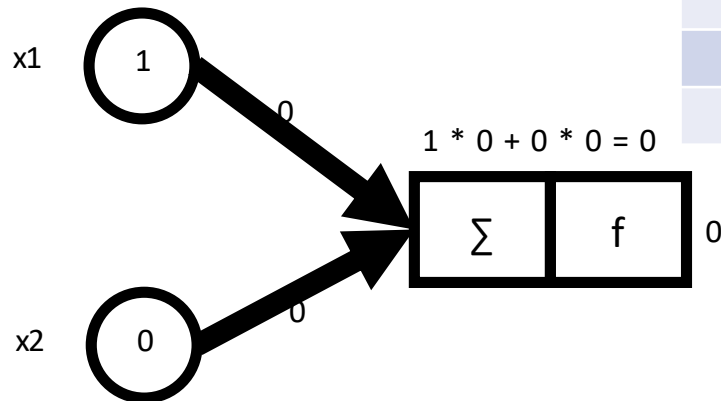
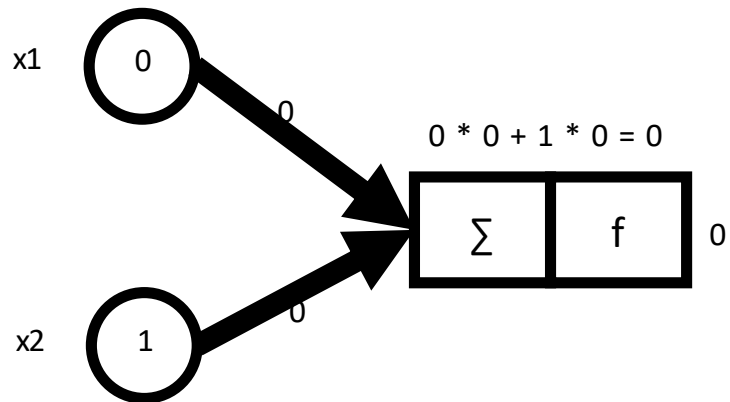
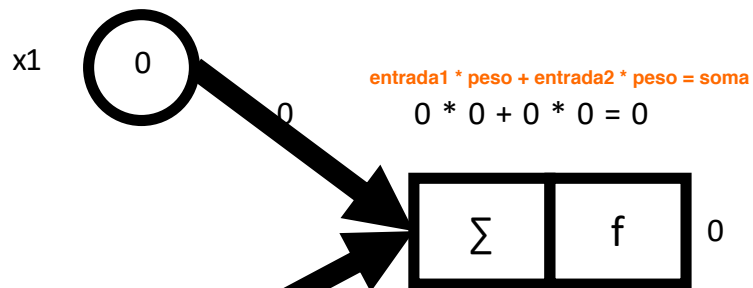


Exemplo

Operador E

x1	x2	Classe
0	0	0
0	1	0
1	0	0
1	1	1

Construir uma rede neural que consiga fazer a previsão do operador lógico AND.
Fazer a rede aprender a calcular pesos, para conseguir prever os valores de saída do operador.



x1	x2	Classe
0	0	0
0	1	0
1	0	0
1	1	1

Resp. Prevista
 Resp. Correta

$0 - 0 = 0$
 $0 - 0 = 0$
 $0 - 0 = 0$
 $1 - 0 = 1$

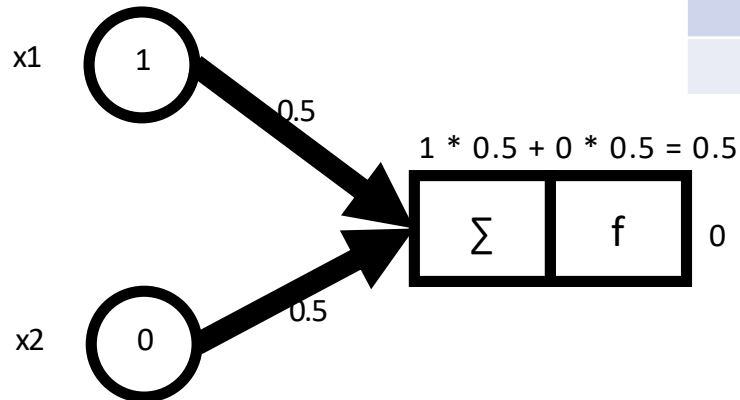
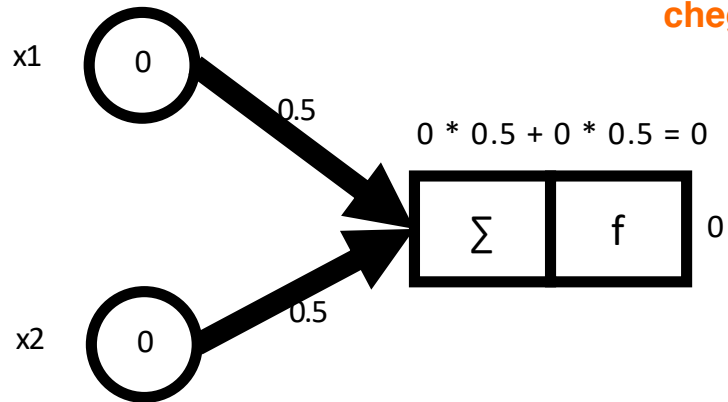
Calculo do erro
 somatório = 1

Calculo de ajuste dos pesos

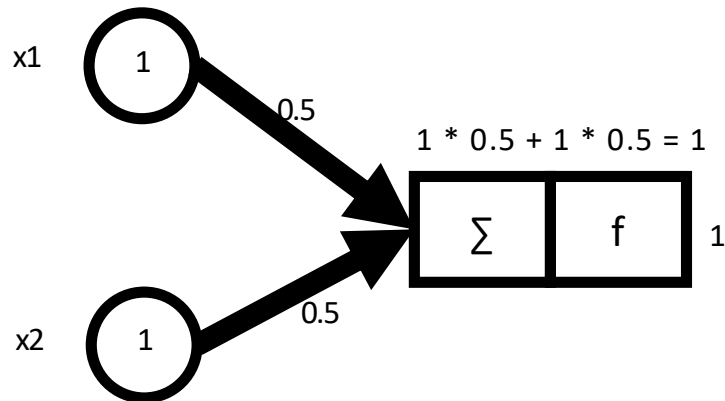
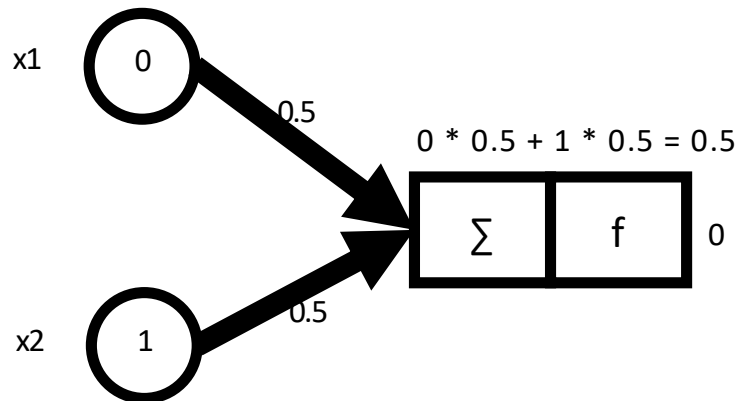
$\text{peso}(n+1) = \text{peso}(n) + (\text{taxaAprendizagem} * \text{entrada} * \text{erro})$
 $\text{peso}(n+1) = 0 + (0.1 * 1 * 1) = 0.1$

Se evoluirmos os ajustes,
chegaremos ao peso de 0.5

x1	x2	Classe
0	0	0
0	1	0
1	0	0
1	1	1



$0 - 0 = 0$
 $0 - 0 = 0$
 $0 - 0 = 0$
 $1 - 1 = 0$

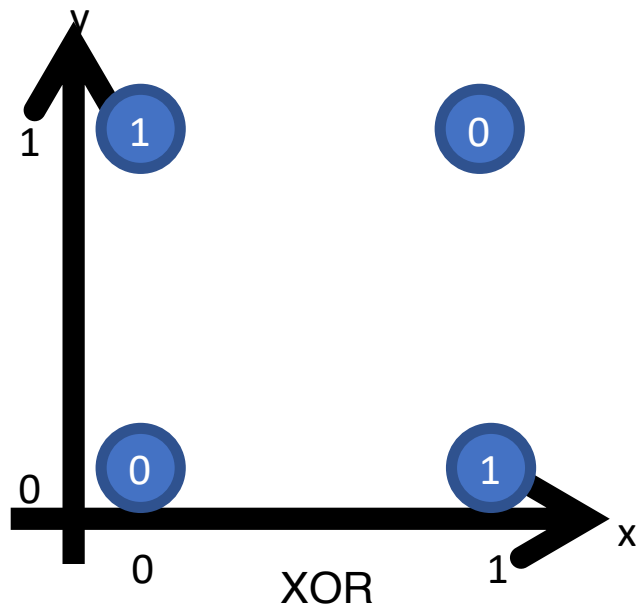
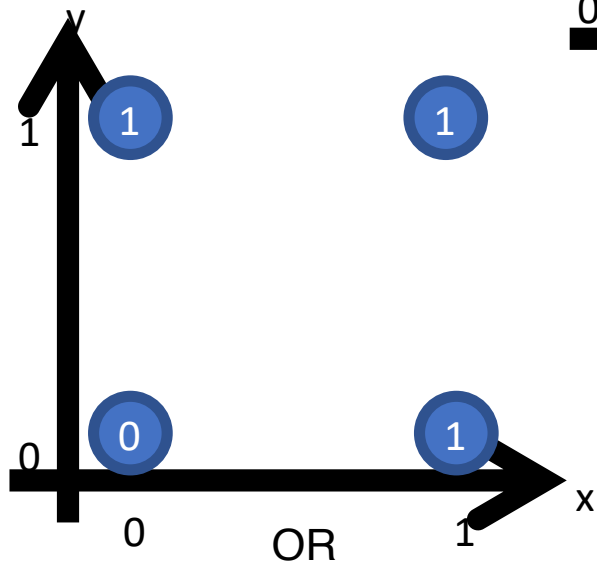
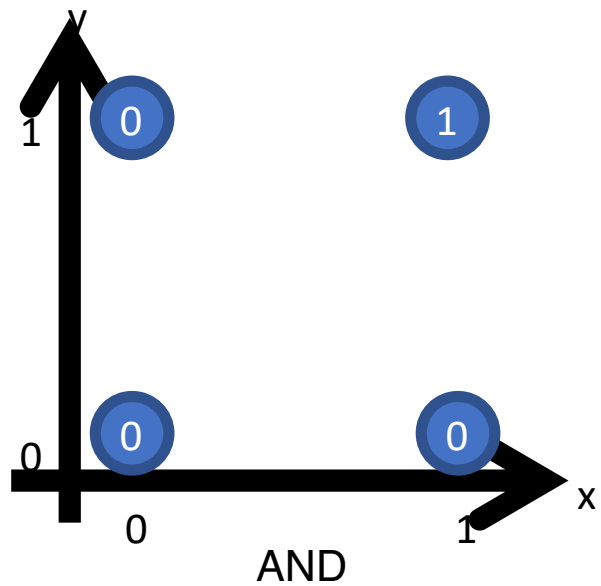


Objetivo

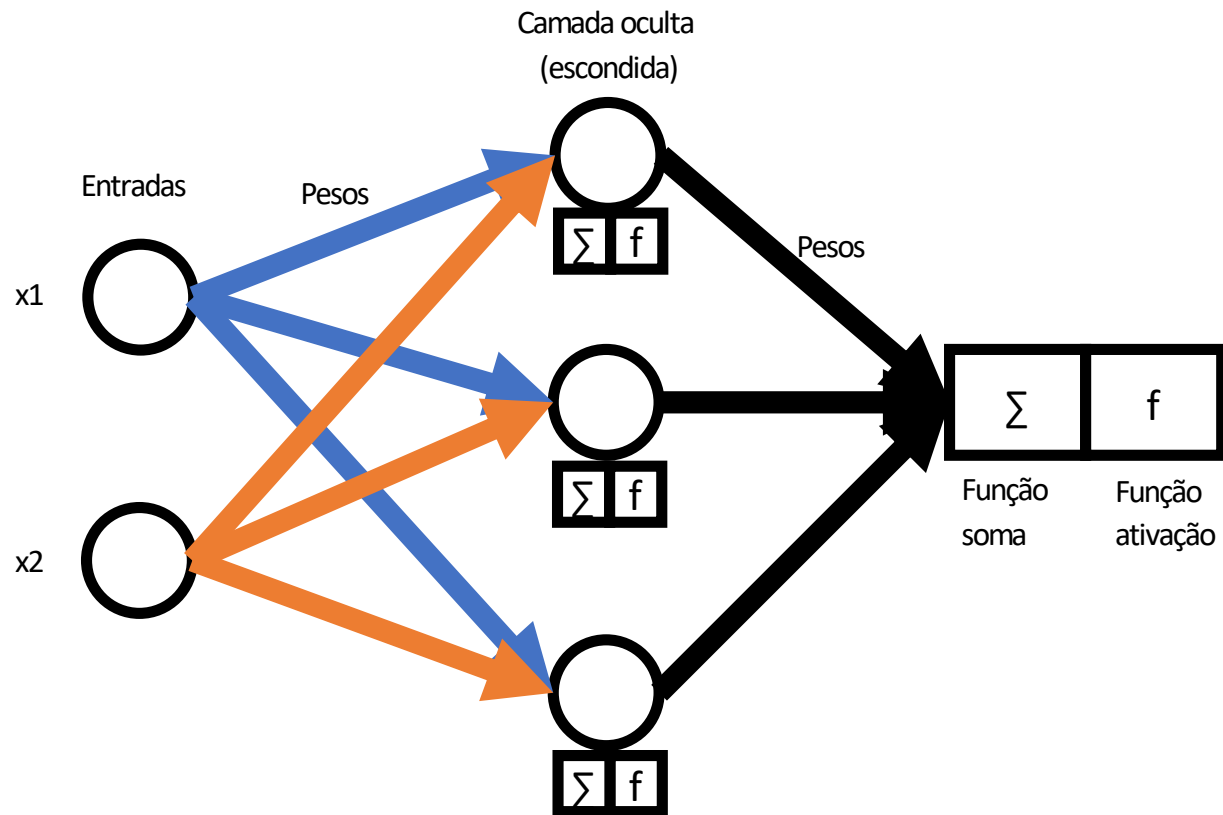
- A aprendizagem de uma rede neural é descobrir os melhores valores de peso para as camadas de neurônios.

Atividade

- Monte a tabela verdade dos operadores OR e XOR.
- Monte um rede neural que consiga prever os resultados desses operadores.
- Faça o calculo dos pesos dos neurônios.

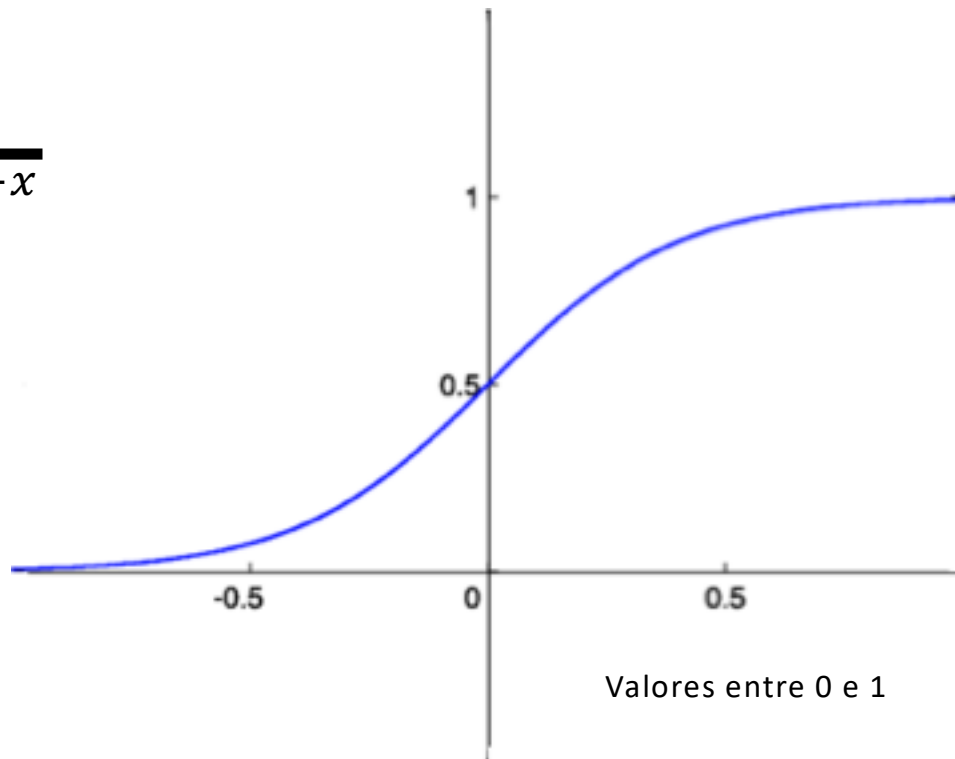


Redes multicamada (multilayer perceptron)



Sigmoid (função sigmoide)

$$y = \frac{1}{1 + e^{-x}}$$



Valores entre 0 e 1

Se X for alto o valor será aproximadamente 1

Se X for pequeno o valor será aproximadamente 0

Não retorna valores negativos

Operador XOR

x1	x2	Classe
0	0	0
0	1	1
1	0	1
1	1	0

Cálculos camada entrada
para camada oculta

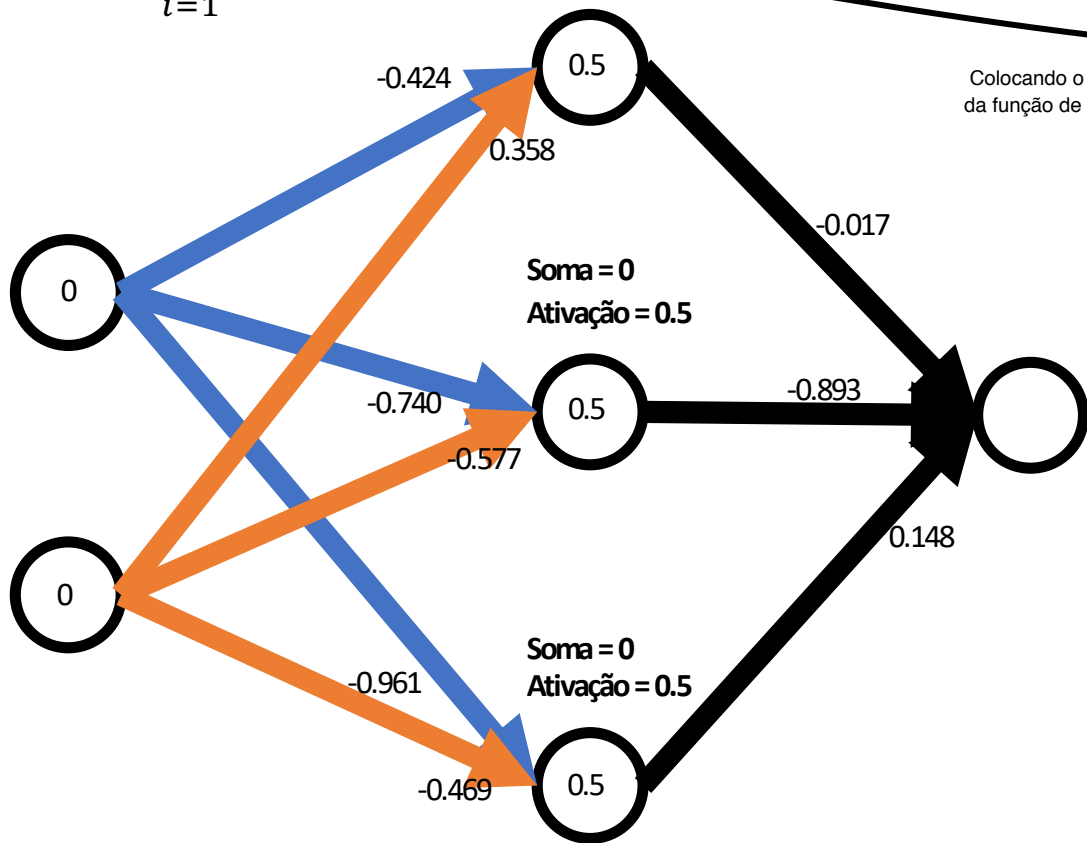
$$soma = \sum_{i=1}^n x_i * w_i$$

função de ativação

$$y = \frac{1}{1 + e^{-x}}$$

x1	x2	Classe
0	0	0
0	1	1
1	0	1
1	1	0

Colocando o valor da soma de cada neurônio da função de ativação, temos o valor de saída.

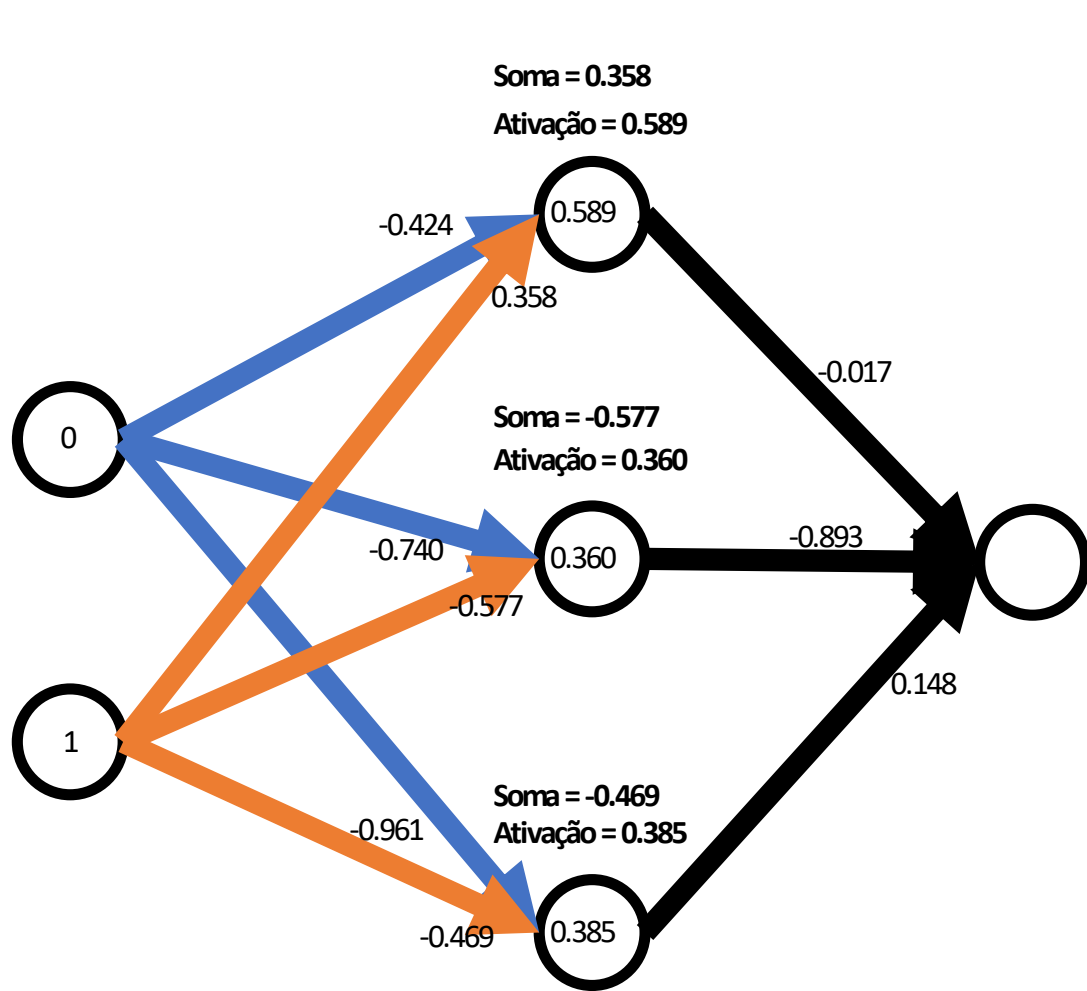


$$0 * (-0.424) + 0 * 0.358 = 0$$

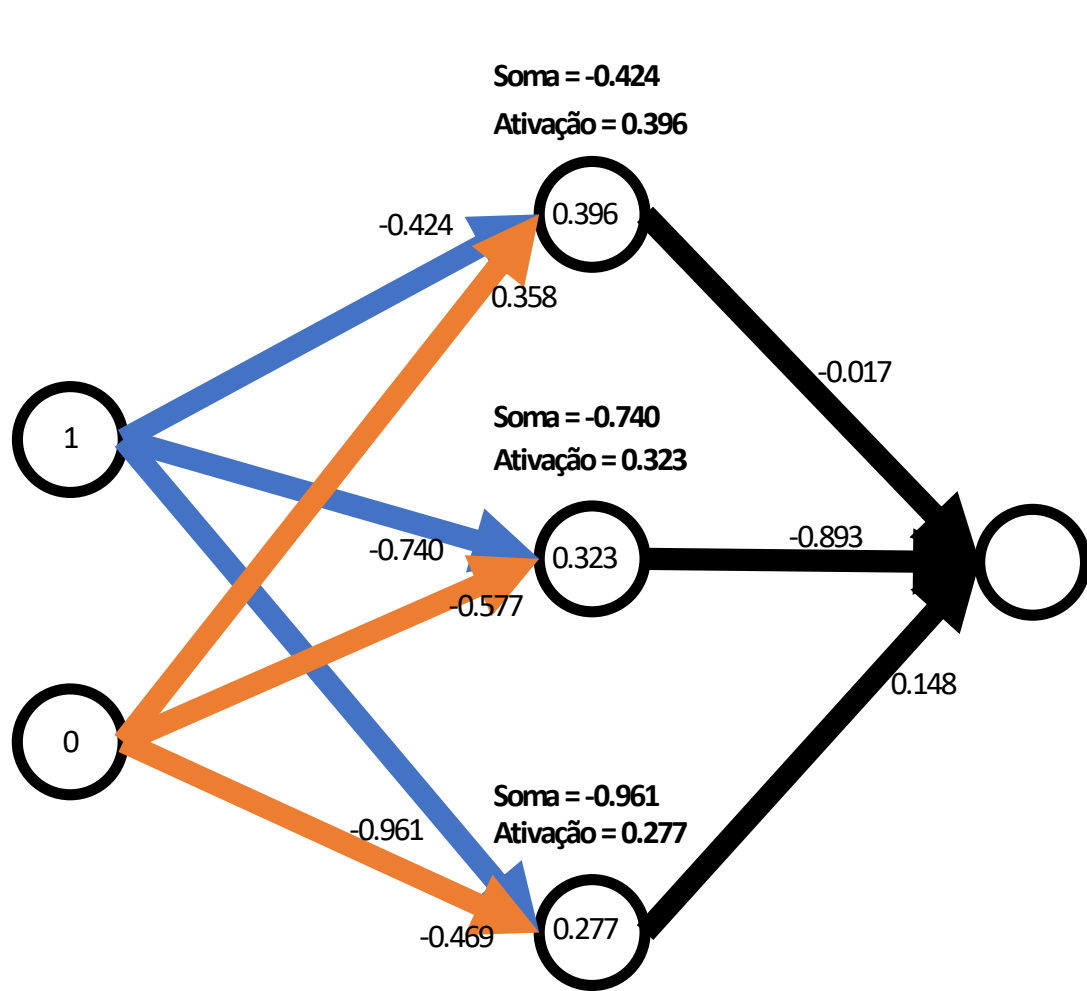
$$0 * (-0.740) + 0 * (-0.577) = 0$$

$$0 * (-0.961) + 0 * (-0.469) = 0$$

Os pesos são inseridos aleatoriamente



x1	x2	Classe
0	0	0
0	1	1
1	0	1
1	1	0

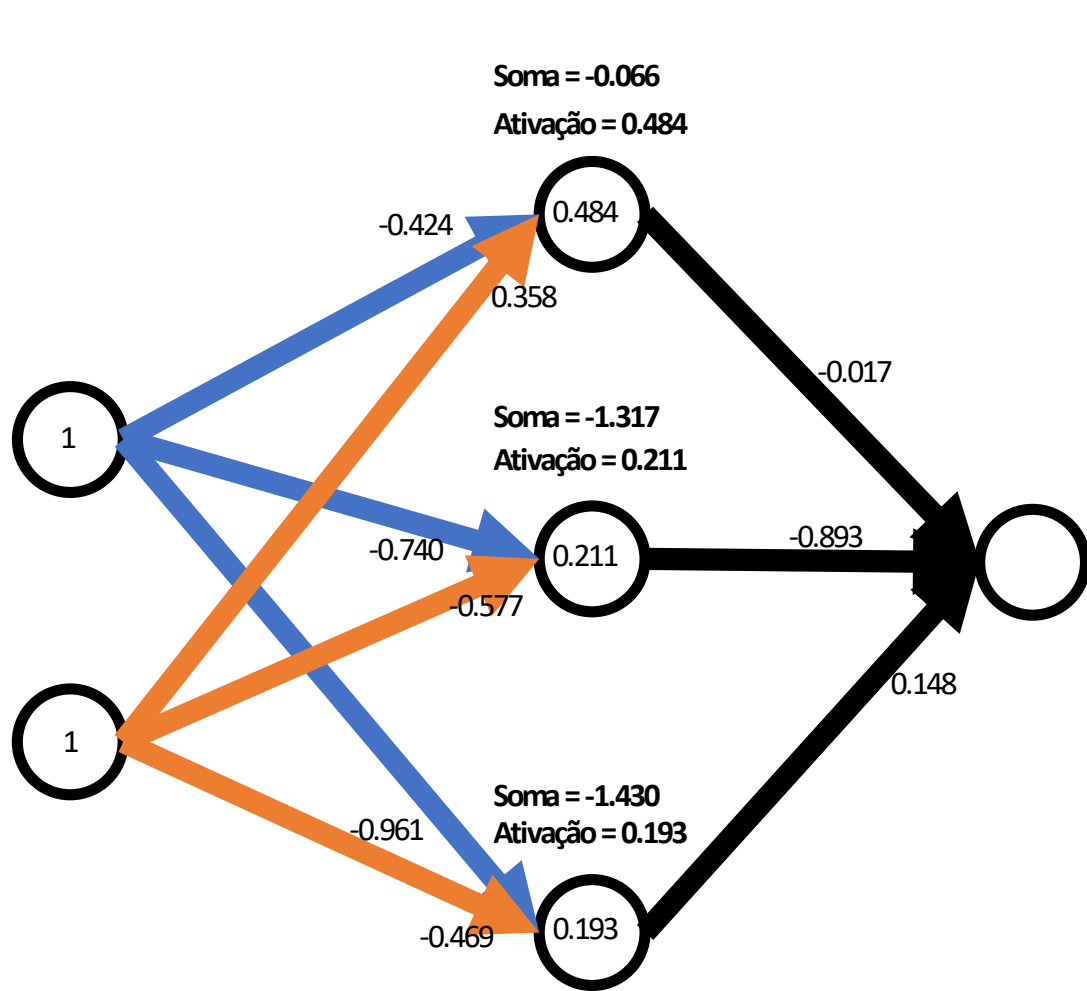


x1	x2	Classe
0	0	0
0	1	1
1	0	1
1	1	0

$$1 * (-0.424) + 0 * 0.358 = -0.424$$

$$1 * (-0.740) + 0 * (-0.577) = -0.740$$

$$1 * (-0.961) + 0 * (-0.469) = -0.961$$



x1	x2	Classe
0	0	0
0	1	1
1	0	1
1	1	0

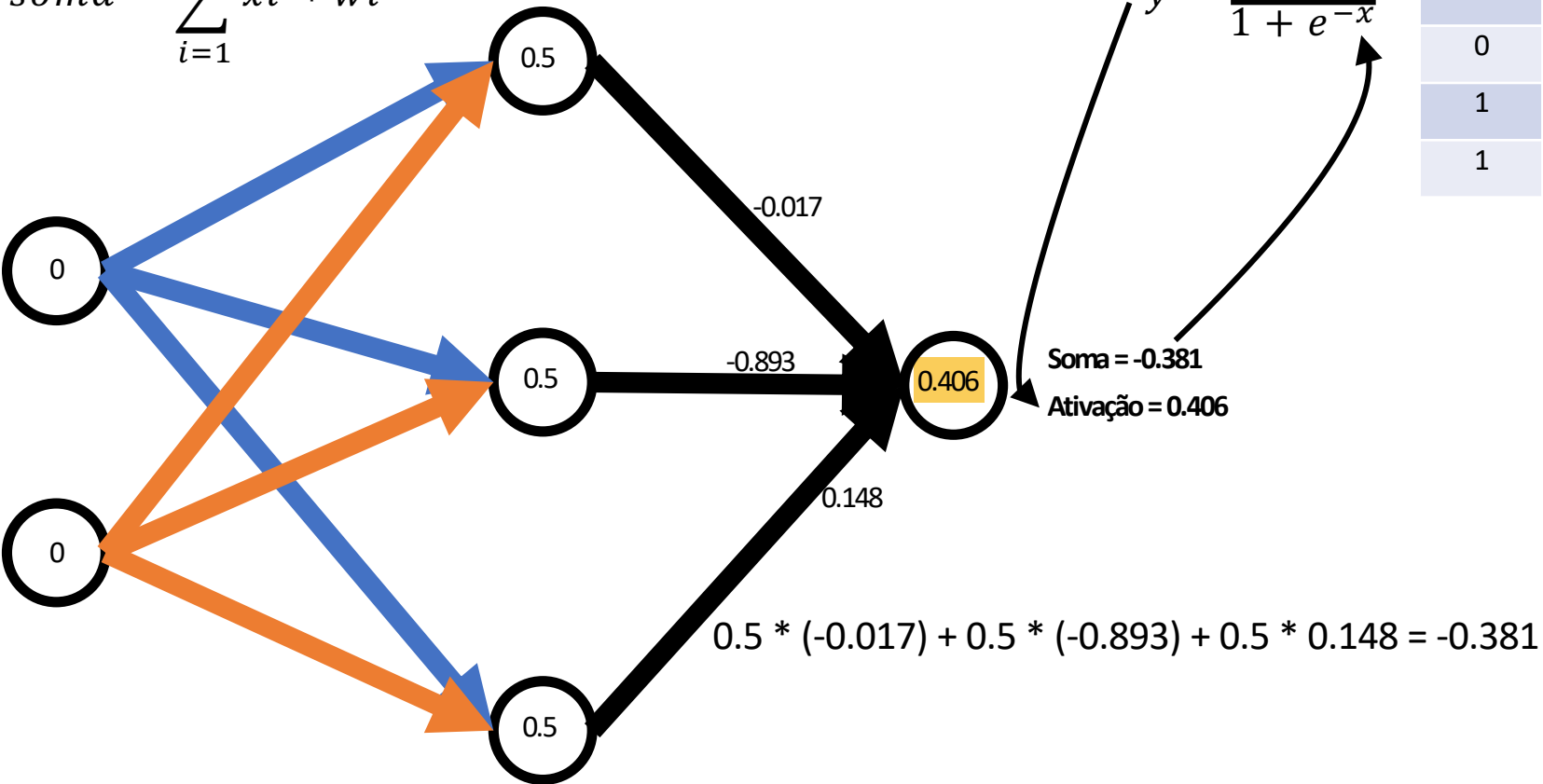
$$1 * (-0.424) + 1 * 0.358 = -0.066$$

$$1 * (-0.740) + 1 * (-0.577) = -1.317$$

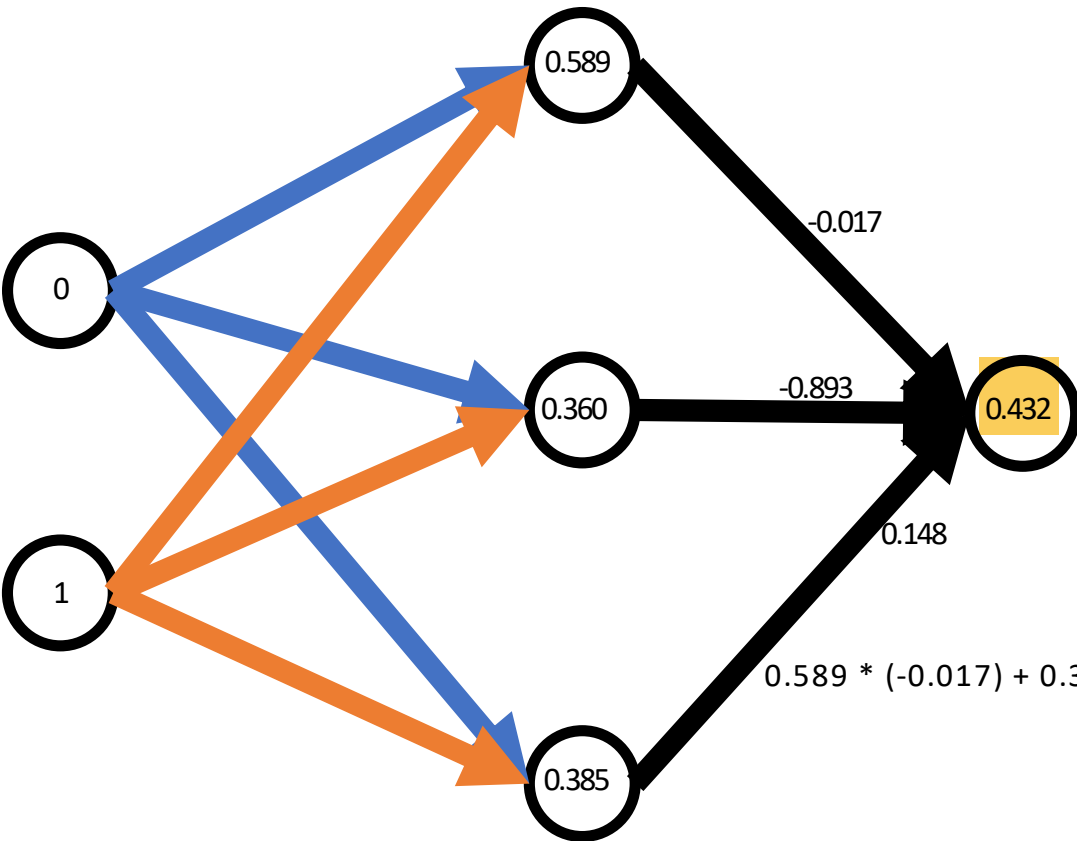
$$1 * (-0.961) + 1 * (-0.469) = -1.430$$

Cálculos camada oculta para
camada de saída

$$soma = \sum_{i=1}^n x_i * w_i$$



x1	x2	Classe
0	0	0
0	1	1
1	0	1
1	1	0

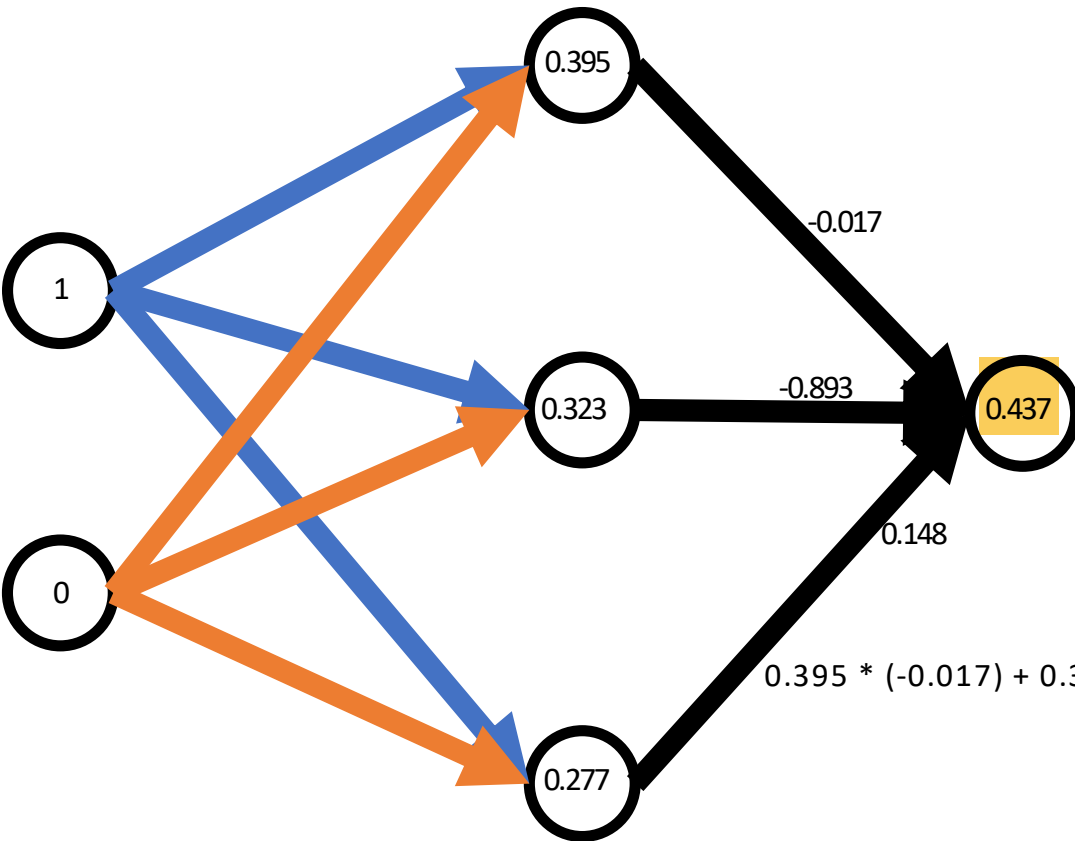


Soma = -0.274
Ativação = 0.432

$$0.589 * (-0.017) + 0.360 * (-0.893) + 0.385 * 0.148 = -0.274$$



x1	x2	Classe
0	0	0
0	1	1
1	0	1
1	1	0

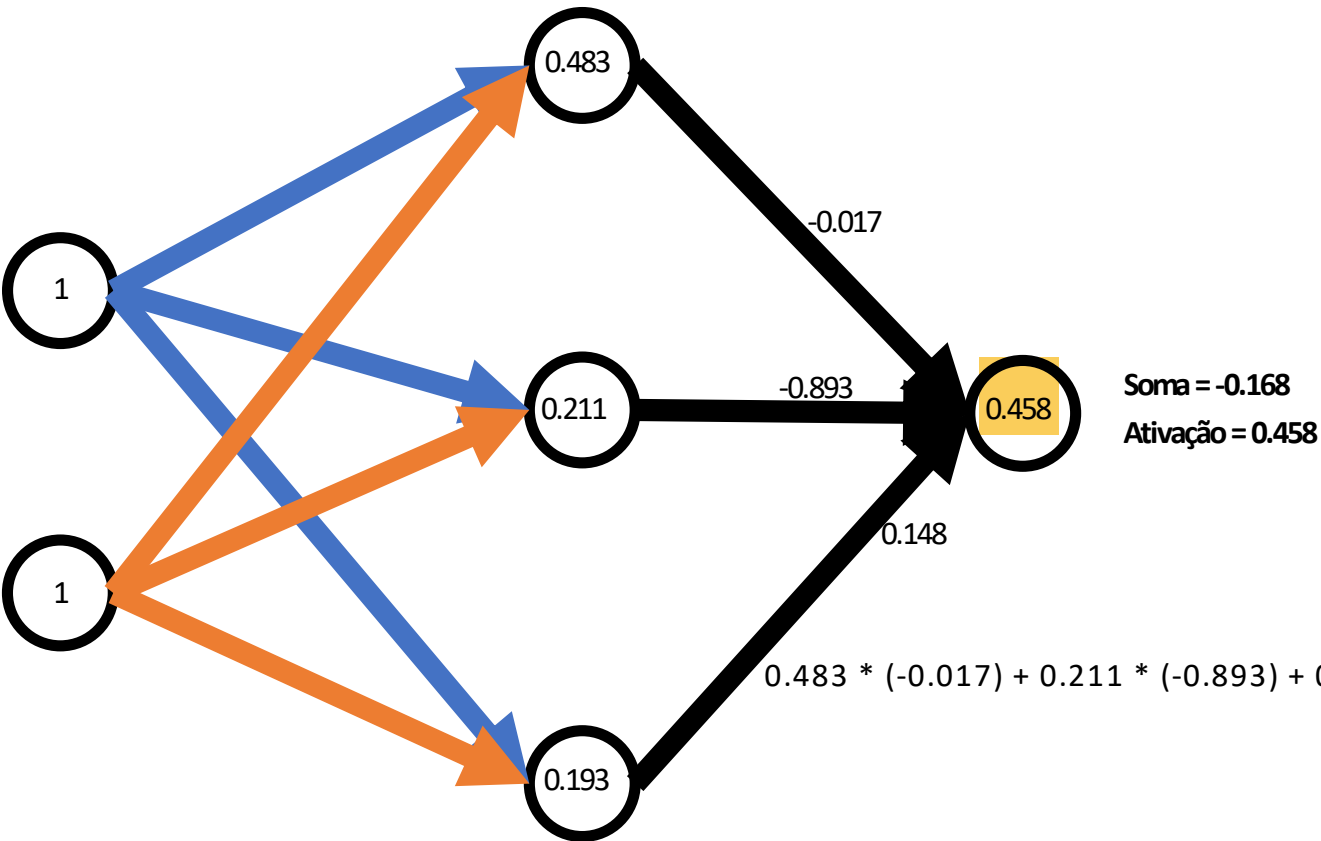


Soma = -0.254
Ativação = 0.437

$$0.395 * (-0.017) + 0.323 * (-0.893) + 0.277 * 0.148 = -0.254$$



x1	x2	Classe
0	0	0
0	1	1
1	0	1
1	1	0



x1	x2	Classe
0	0	0
0	1	1
1	0	1
1	1	0

Primeira rodada da rede esta pronta, ja temos os valores de previsão para os primeiros registros.

Erro

- Algoritmo mais simples
 - $\text{erro} = \text{respostaCorreta} - \text{respostaCalculada}$

x1	x2	Classe	Calculado	Erro
0	0	0	0.406	-0.406
0	1	1	0.432	0.568
1	0	1	0.437	0.563
1	1	0	0.458	-0.458

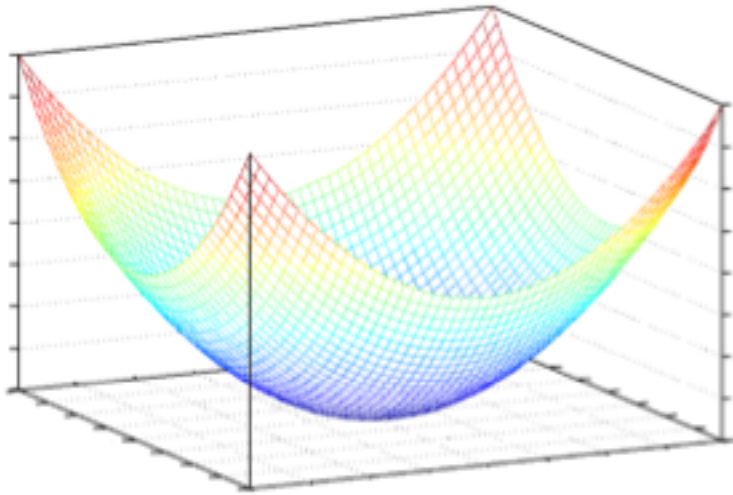
Média absoluta = 0.49

Gradiente

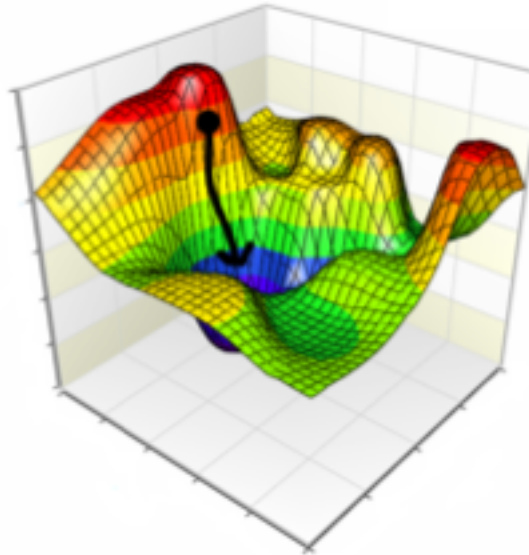
loss function, ou função de perda

$\min C(w_1, w_2 \dots w_n)$

Calcular a derivada parcial para
mover para a direção do gradiente

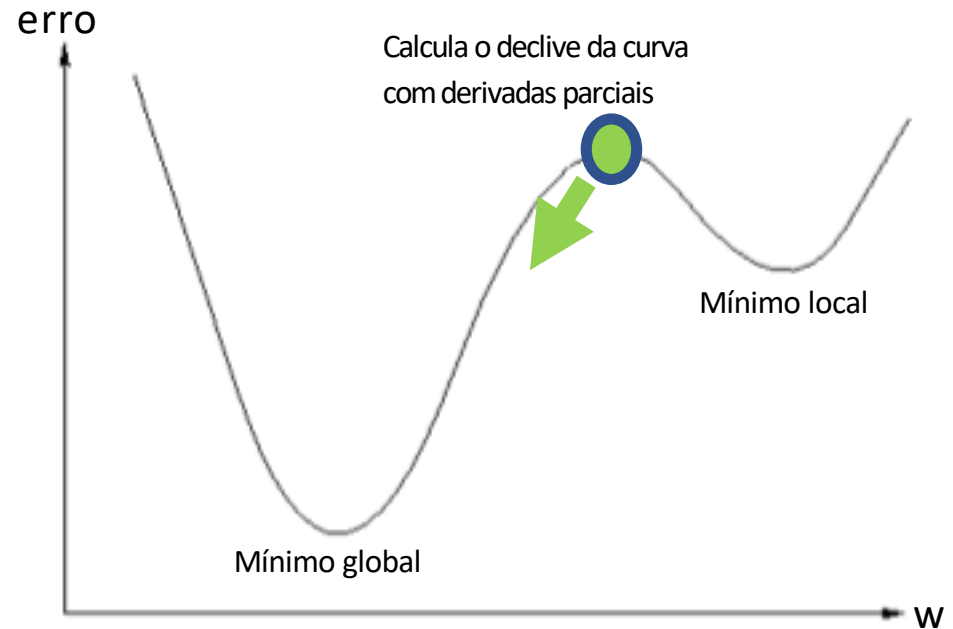


Algoritmo que tenta encontrar o
melhor valor de ajuste dos pesos



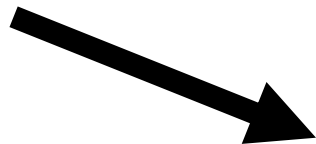
Gradiente

- Encontrar a combinação de pesos que o erro é o menor possível
- Gradiente é calculado para saber quanto ajustar os pesos



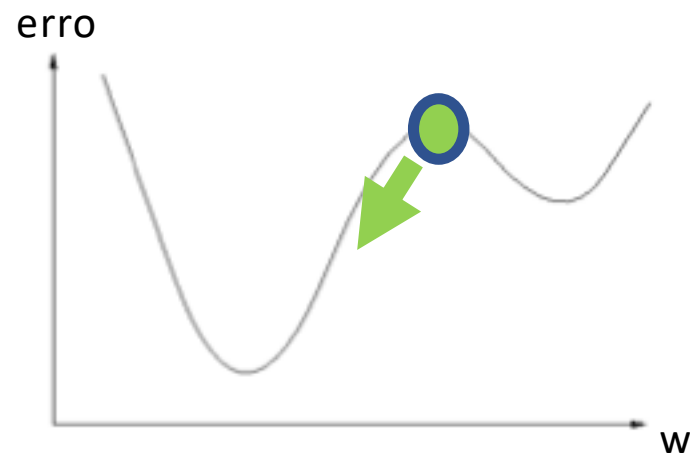
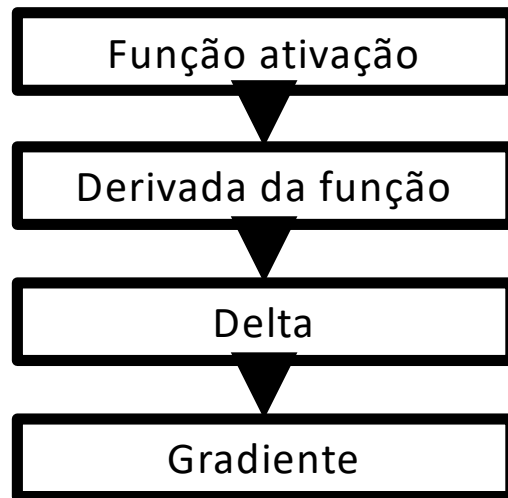
Gradiente (derivada)

$$y = \frac{1}{1 + e^{-x}}$$



$$d = y * (1 - y)$$

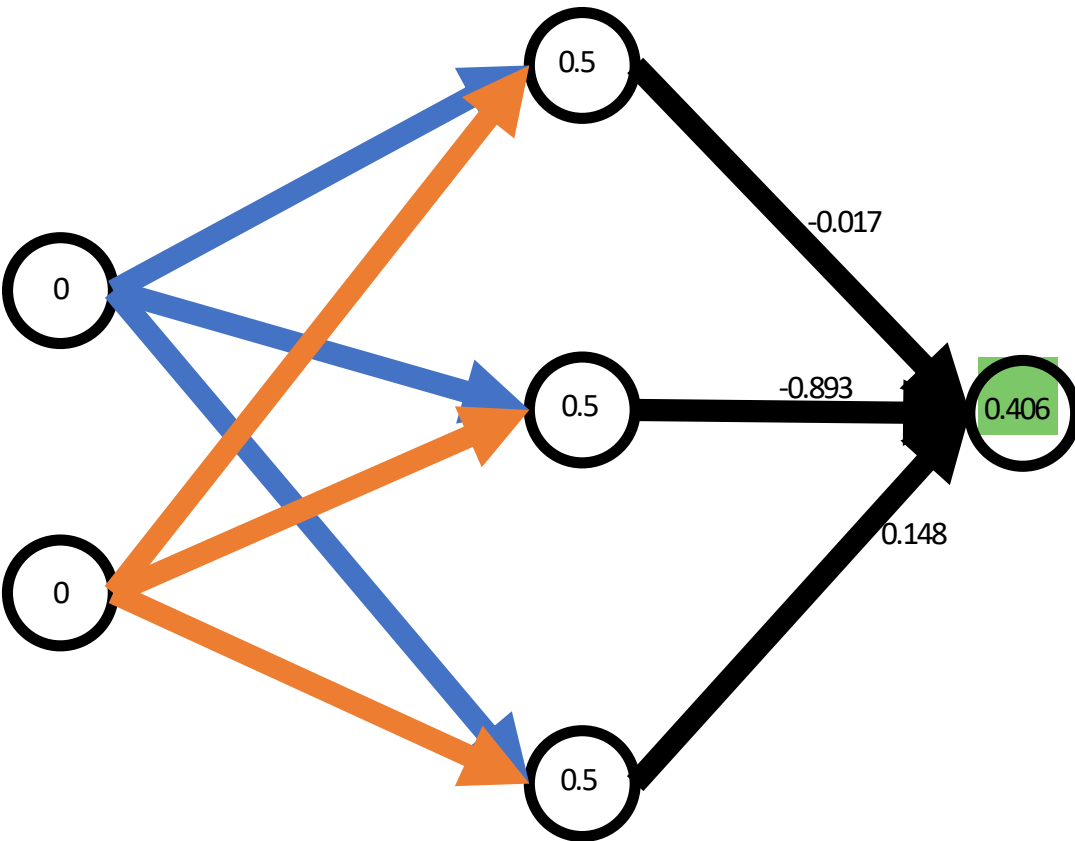
Cálculo do parâmetro Delta



Delta camada saída

$$\text{DeltaSaída} = \text{Erro} * \text{DerivadaSigmoide}$$

x1	x2	Classe
0	0	0
0	1	1
1	0	1
1	1	0



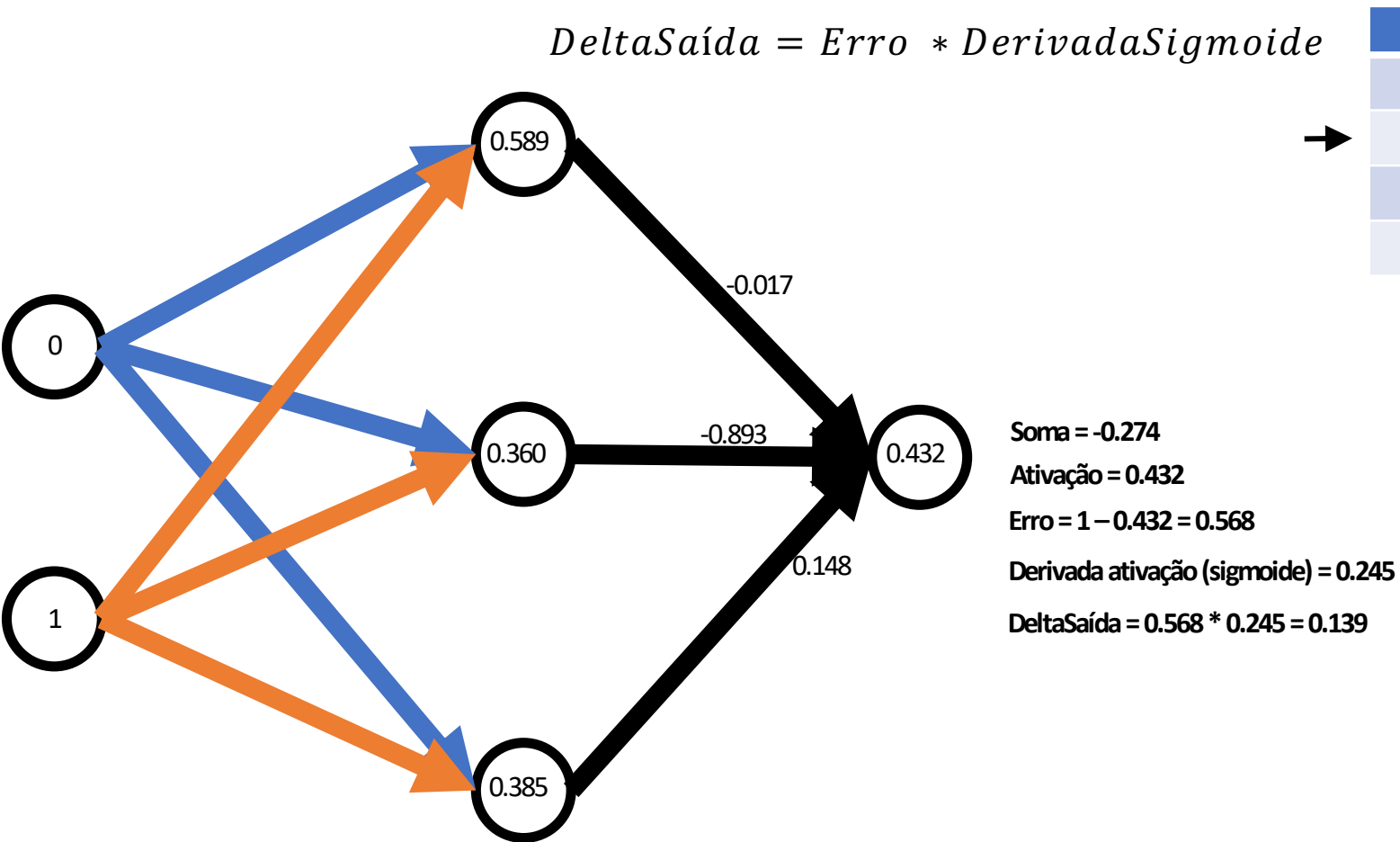
Soma = -0.381

Ativação = 0.406

Erro = 0 - 0.406 = -0.406

Derivada ativação (sigmoide) = 0.241

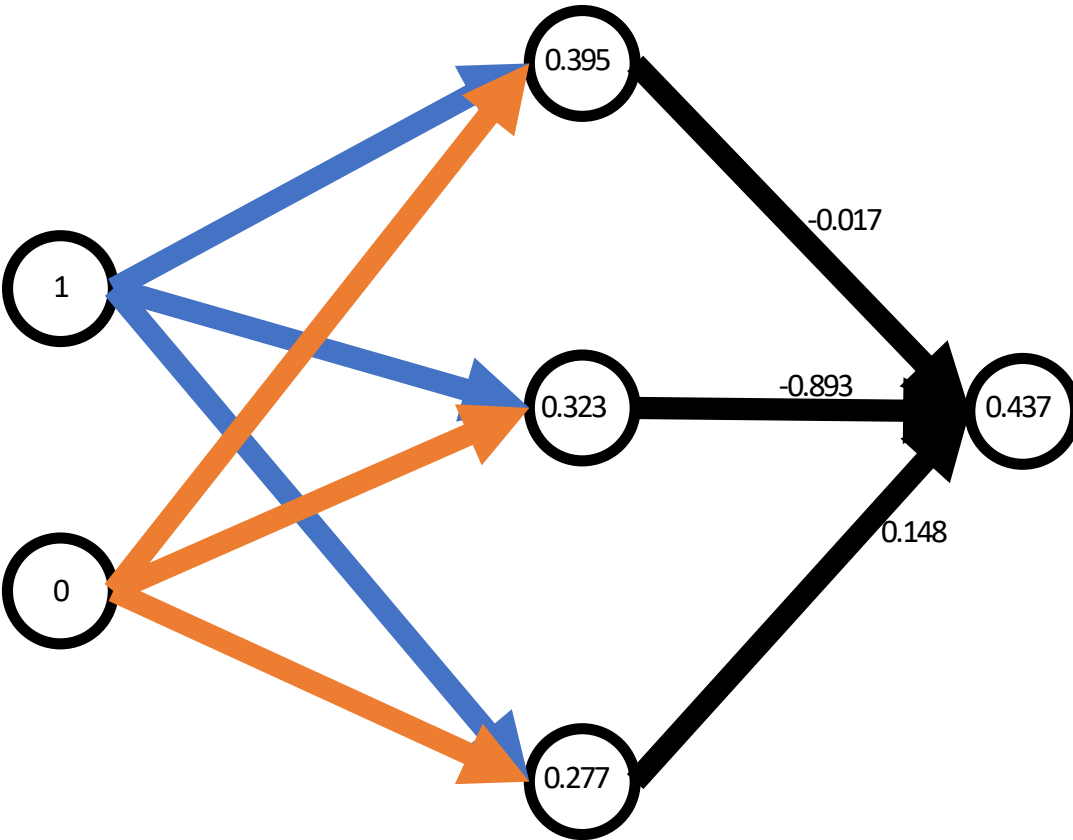
DeltaSaída = -0.406 * 0.241 = -0.098



→

x1	x2	Classe
0	0	0
0	1	1
1	0	1
1	1	0

$$\text{DeltaSaída} = \text{Erro} * \text{DerivadaSigmoide}$$



x1	x2	Classe
0	0	0
0	1	1
1	0	1
1	1	0

Soma = -0.254

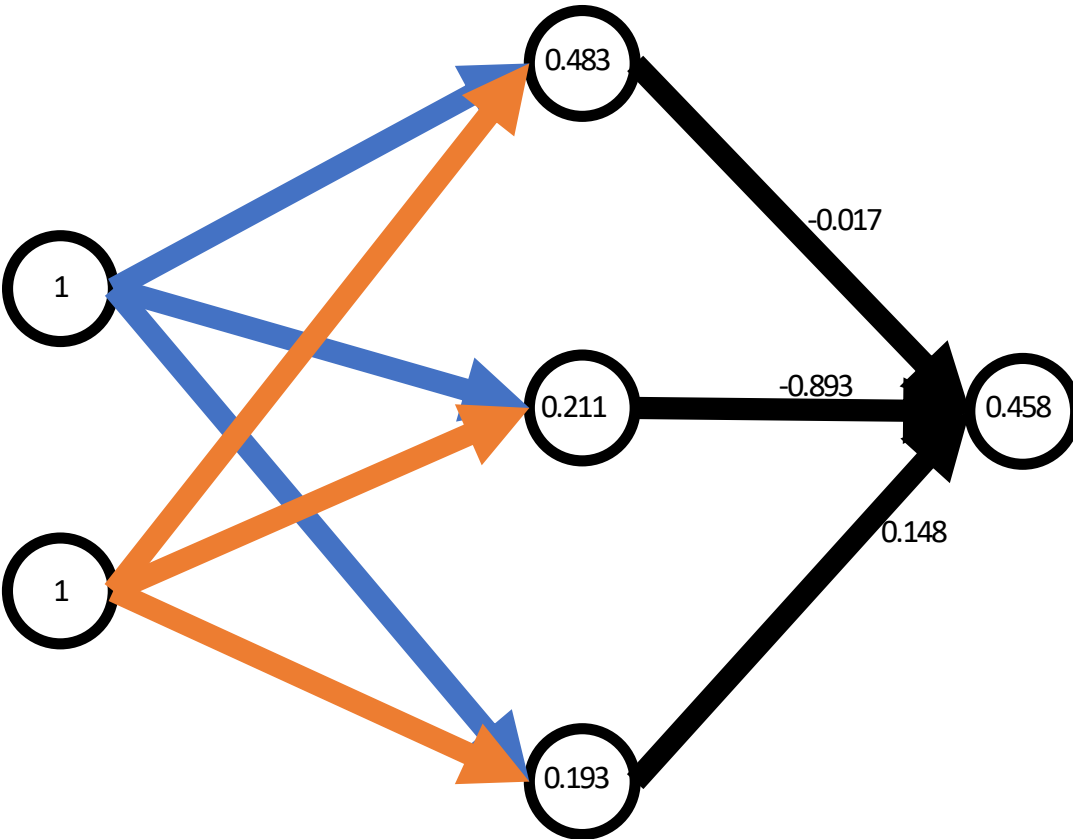
Ativação = 0.437

Erro = $1 - 0.437 = 0.563$

Derivada ativação (sigmoide) = 0.246

DeltaSaída = $0.563 * 0.246 = 0.139$

$$\text{DeltaSaída} = \text{Erro} * \text{DerivadaSigmoide}$$



x1	x2	Classe
0	0	0
0	1	1
1	0	1
1	1	0

Soma = -0.168

Ativação = 0.458

Erro = $0 - 0.458 = -0.458$

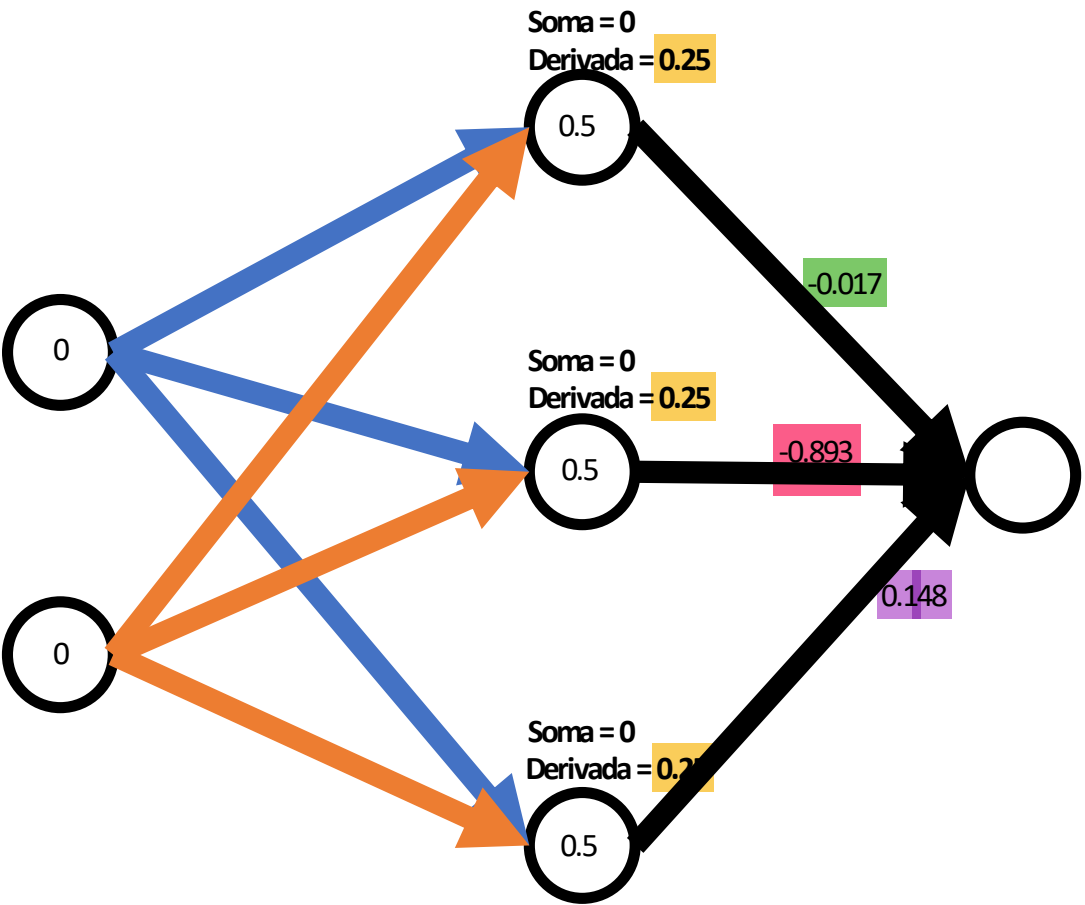
Derivada ativação (sigmoide) = 0.248

DeltaSaída = $-0.458 * 0.248 = -0.114$

Delta camada escondida

DeltaEscondida = DerivadaSigmoide * peso * DeltaSaída

x1	x2	Classe
0	0	0
0	1	1
1	0	1
1	1	0



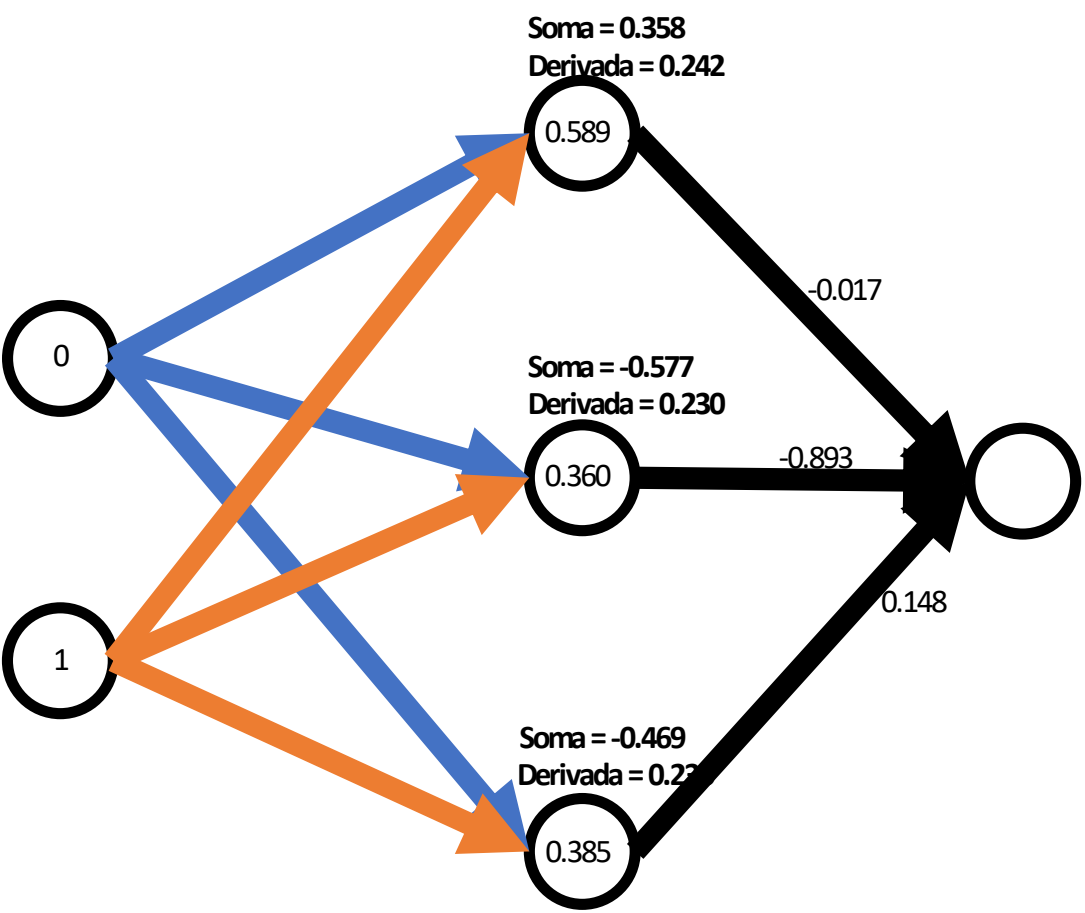
DeltaSaída = -0.098

0.25 * (-0.017) * (-0.098) = 0.000

0.25 * (-0.893) * (-0.098) = 0.022

0.25 * 0.148 * (-0.098) = -0.004

*DeltaEscondida = DerivadaSigmoide * peso * DeltaSaída*



x1	x2	Classe
0	0	0
0	1	1
1	0	1
1	1	0

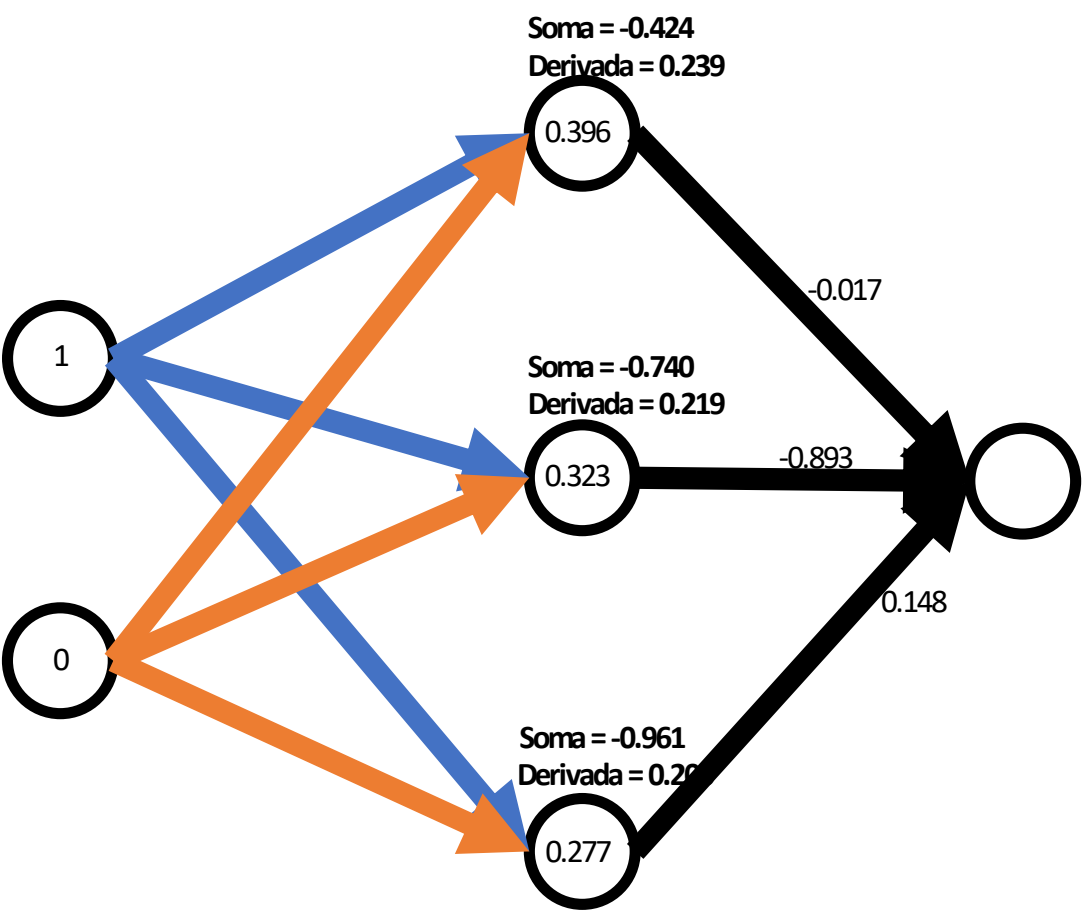
DeltaSaída = 0.139

$0.242 * (-0.017) * 0.139 = -0.001$

$0.230 * (-0.893) * 0.139 = -0.029$

$0.236 * 0.148 * 0.139 = 0.005$

*DeltaEscondida = DerivadaSigmoides * peso * DeltaSaída*



x1	x2	Classe
0	0	0
0	1	1
1	0	1
1	1	0

DeltaSaída = 0.139

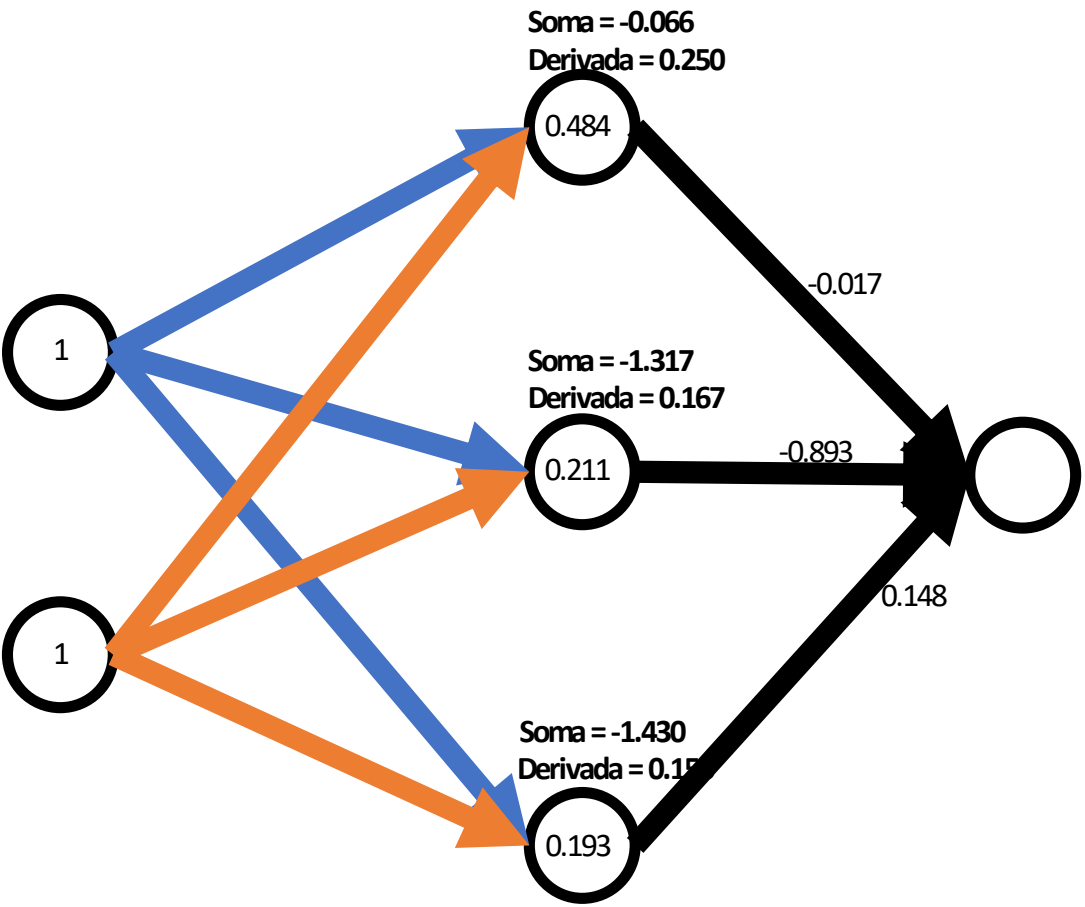
$0.239 * (-0.017) * 0.139 = -0.001$

$0.219 * (-0.893) * 0.139 = -0.027$

$0.200 * 0.148 * 0.139 = 0.004$

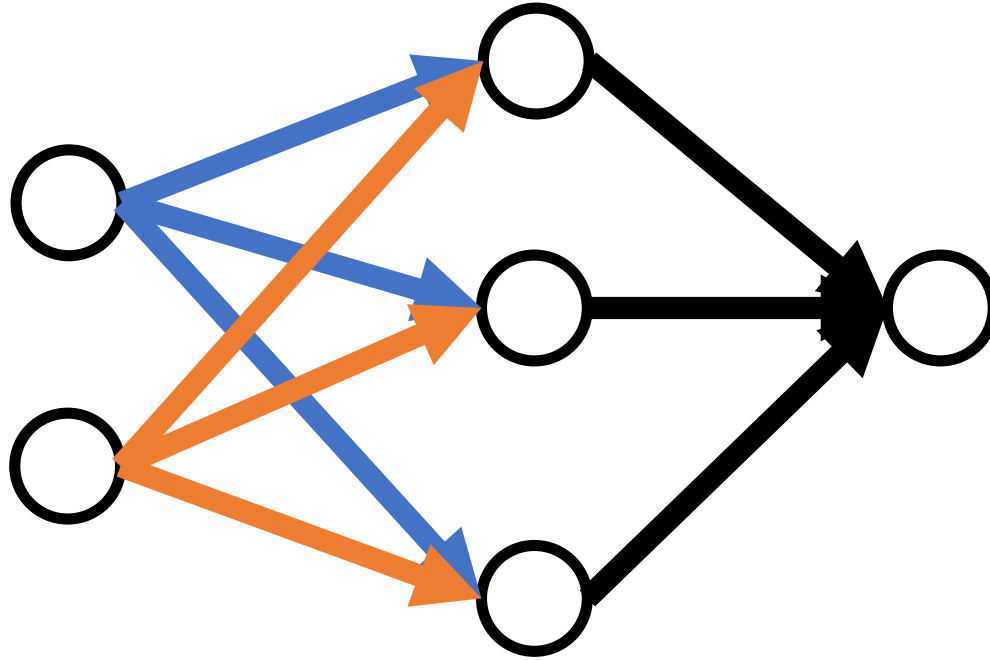
*DeltaEscondida = DerivadaSigmoide * peso * DeltaSaída*

x1	x2	Classe
0	0	0
0	1	1
1	0	1
1	1	0



$0.250 * (-0.017) * (-0.114) = 0.000$
 $0.167 * (-0.893) * (-0.114) = 0.017$
 $0.156 * 0.148 * (-0.114) = -0.003$

Formula de atualização dos pesos



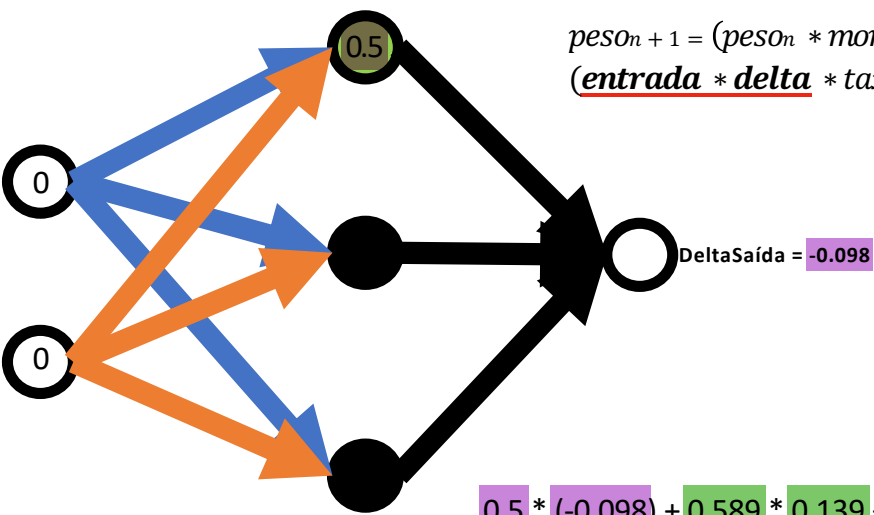
próximo peso

peso atual

$$peso_{n+1} = (peso_n * momento) + (entrada * delta * taxa\ de\ aprendizagem)$$

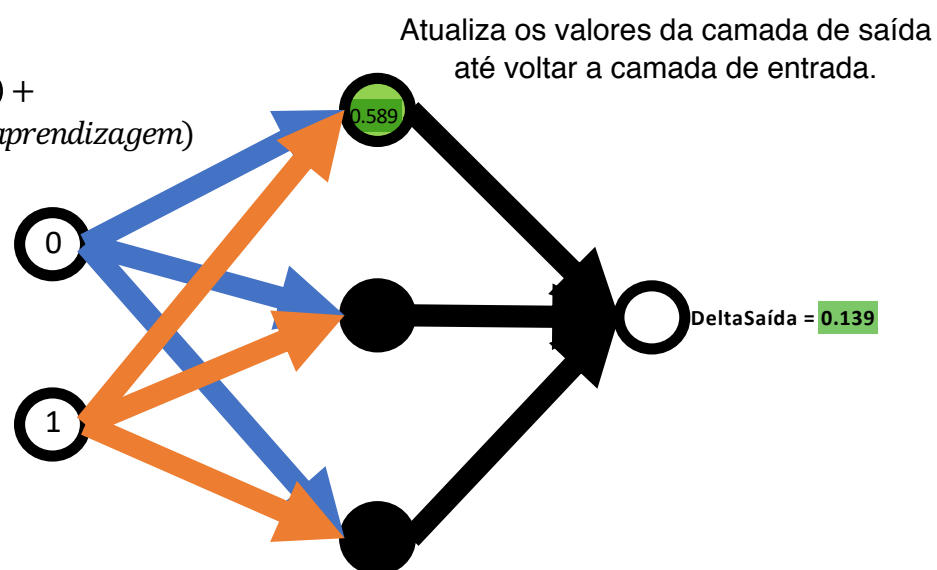
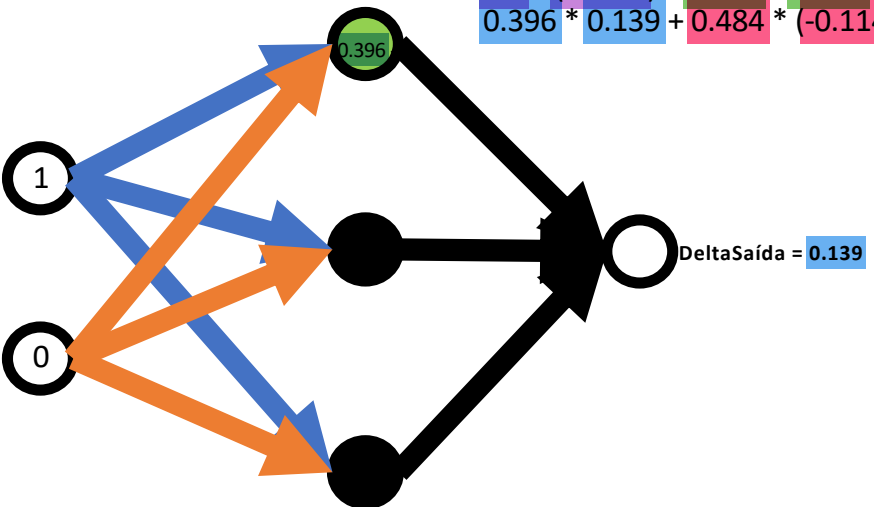
momento -> acelera o ajuste

Ajuste dos pesos da camada
oculta para a camada de saída

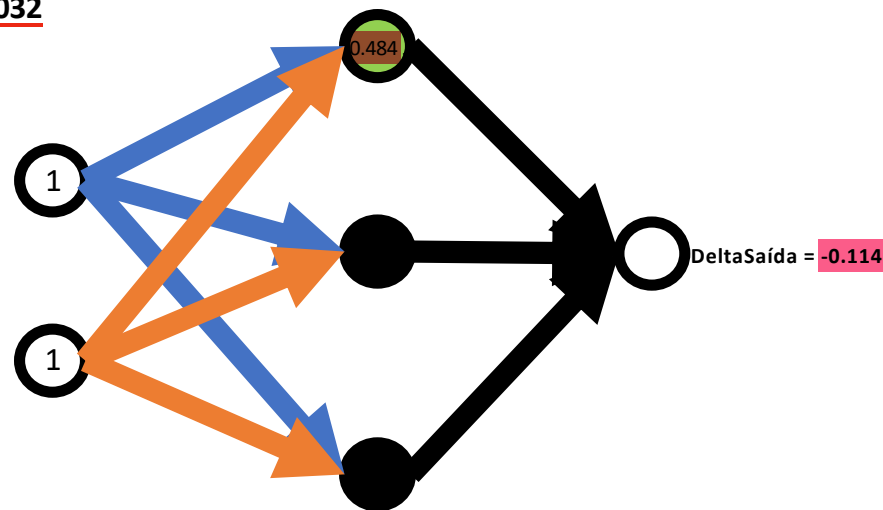


$$peso_{n+1} = (peso_n * momento) + (\textit{entrada} * \textit{delta} * \textit{taxa de aprendizagem})$$

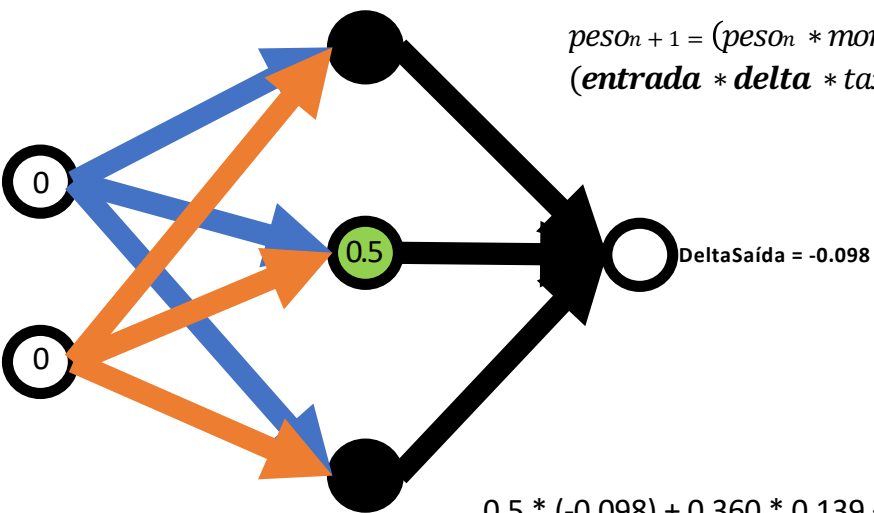
$$0.5 * (-0.098) + 0.589 * 0.139 + 0.396 * 0.139 + 0.484 * (-0.114) = \underline{0.032}$$



Atualiza os valores da camada de saída até voltar a camada de entrada.

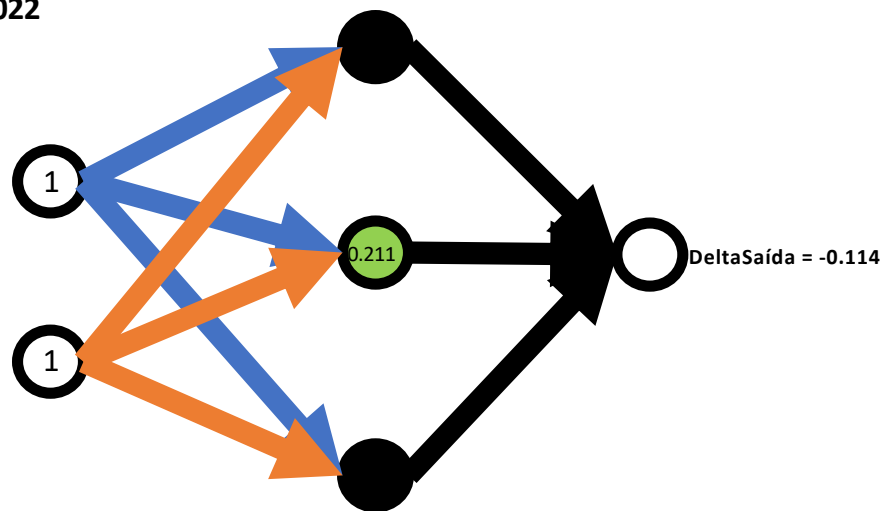
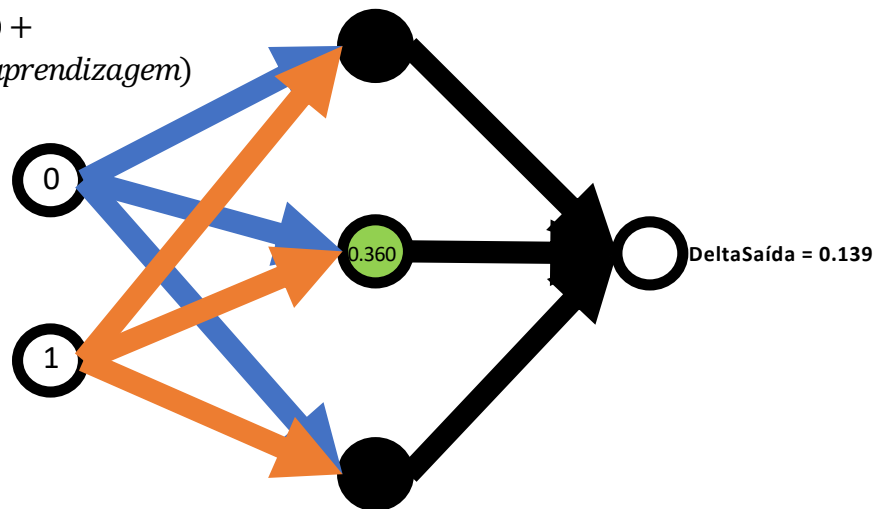
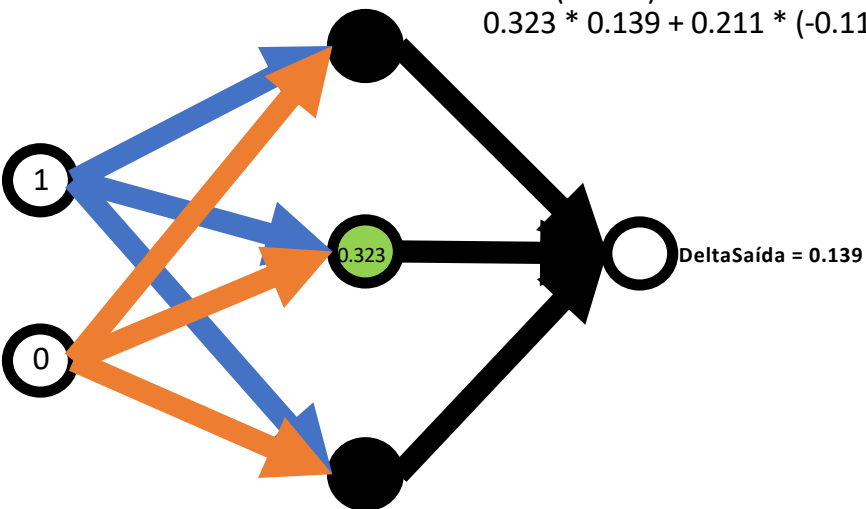


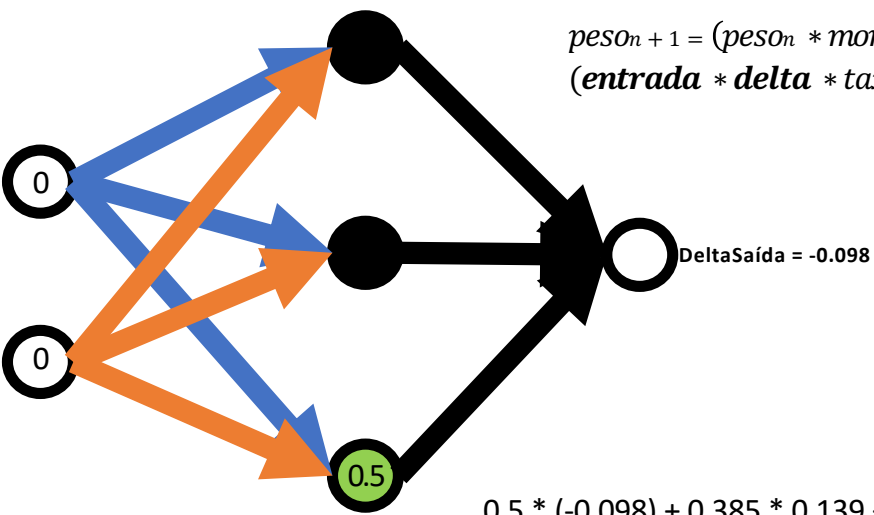
Algoritmo backpropagation



$$peso_{n+1} = (peso_n * momento) + (entrada * delta * taxa de aprendizagem)$$

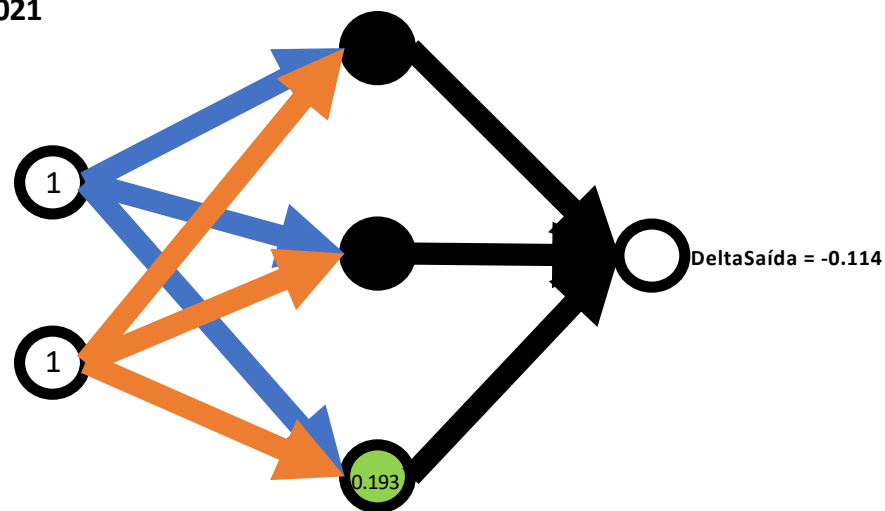
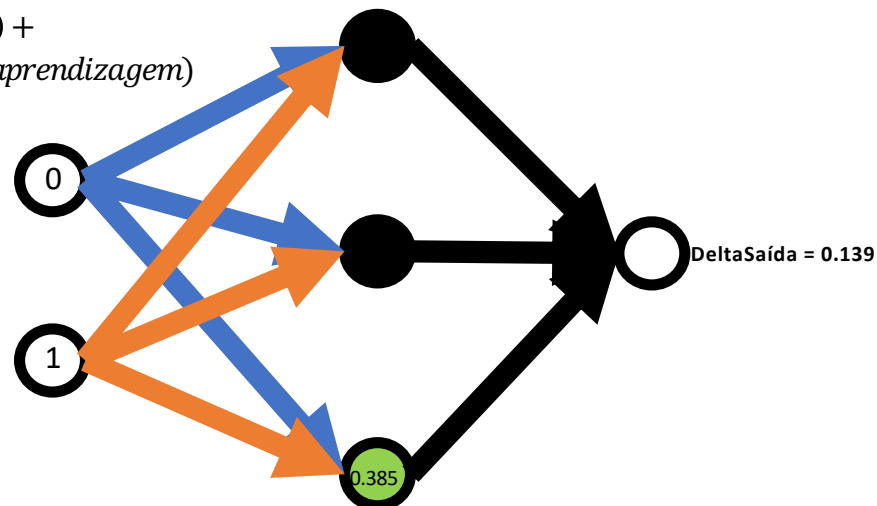
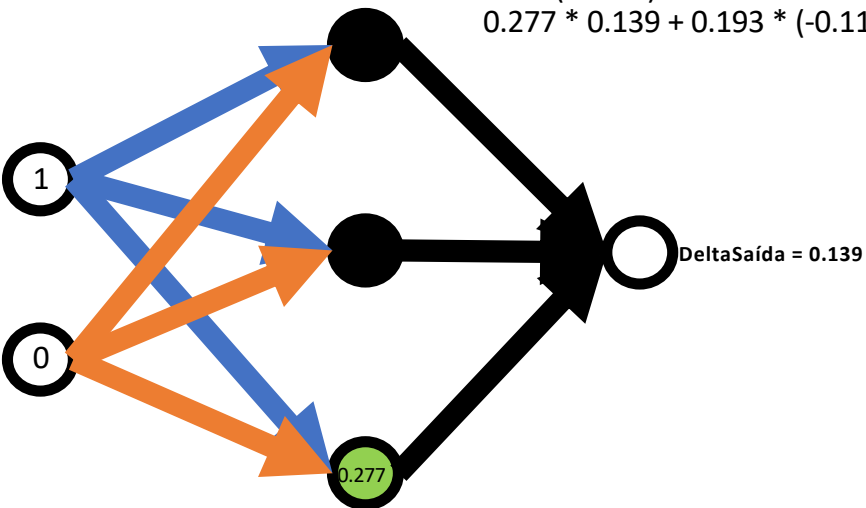
$$0.5 * (-0.098) + 0.360 * 0.139 + 0.323 * 0.139 + 0.211 * (-0.114) = 0.022$$





$$peso_{n+1} = (peso_n * momento) + (entrada * delta * taxa de aprendizagem)$$

$$0.5 * (-0.098) + 0.385 * 0.139 + 0.277 * 0.139 + 0.193 * (-0.114) = 0.021$$



Ajuste dos pesos da camada oculta

Taxa de aprendizagem = 0.3

Momento = 1

Entrada x delta

0.032

0.022

0.021

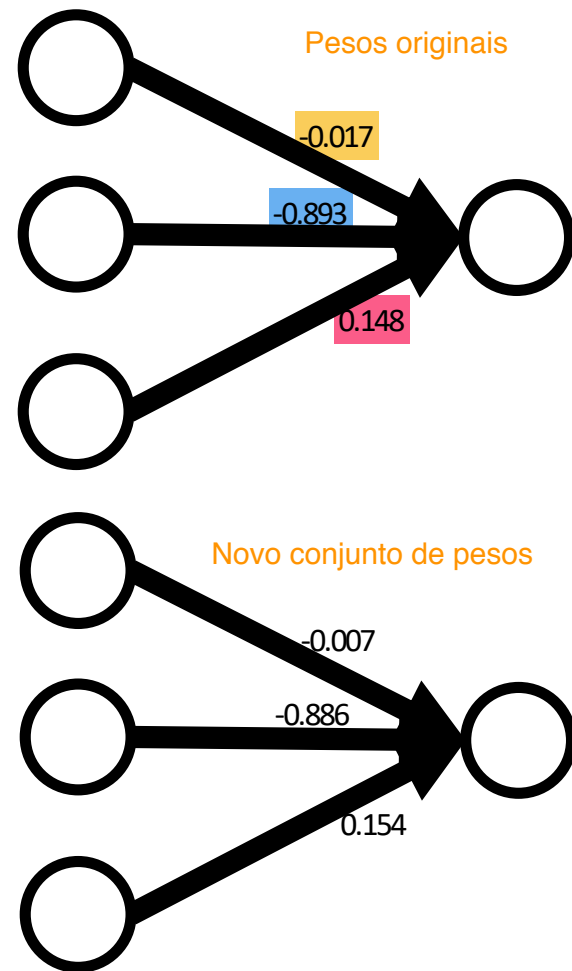
$$\underline{Peso_{n+1}} = (peso_n * momento) + (entrada * delta * taxa\ de\ aprendizagem)$$

Próximos pesos

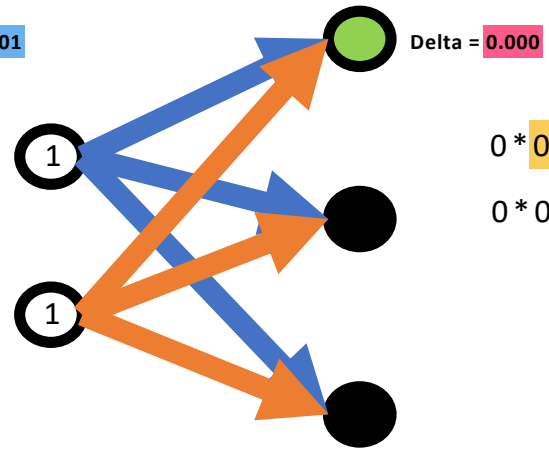
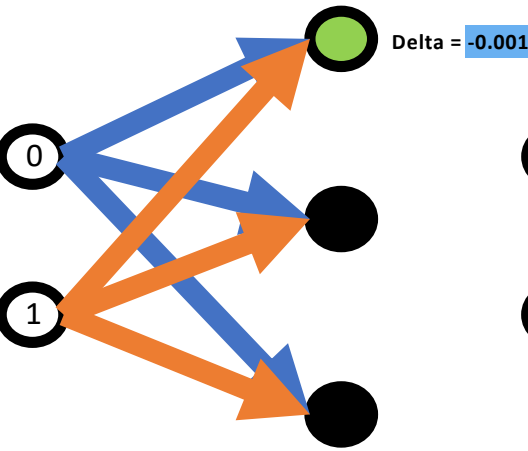
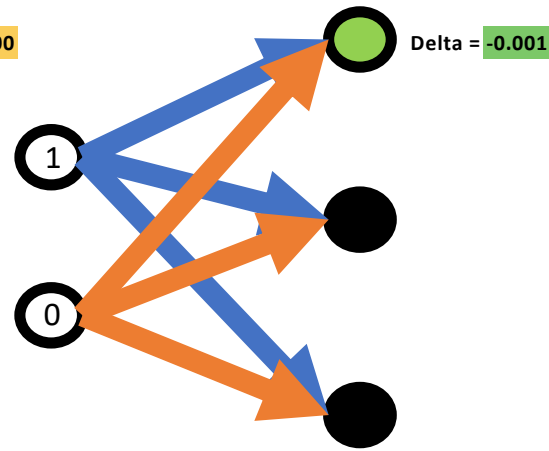
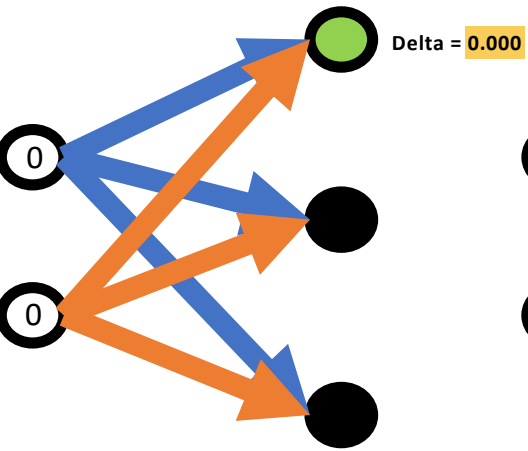
$$(-0.017 * 1) + 0.032 * 0.3 = \underline{-0.007}$$

$$(-0.893 * 1) + 0.022 * 0.3 = \underline{-0.886}$$

$$(0.148 * 1) + 0.021 * 0.3 = \underline{0.154}$$



Ajuste dos pesos da camada de
entrada para a camada oculta

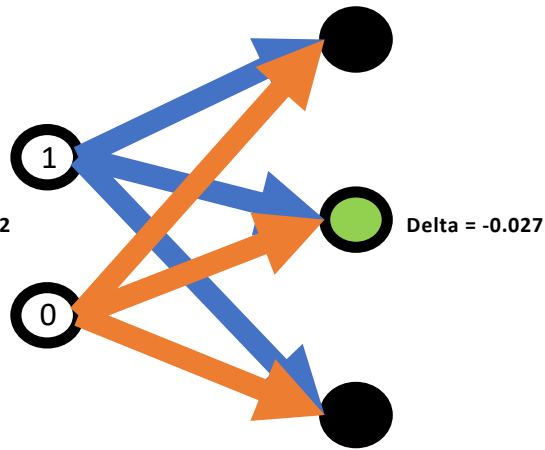
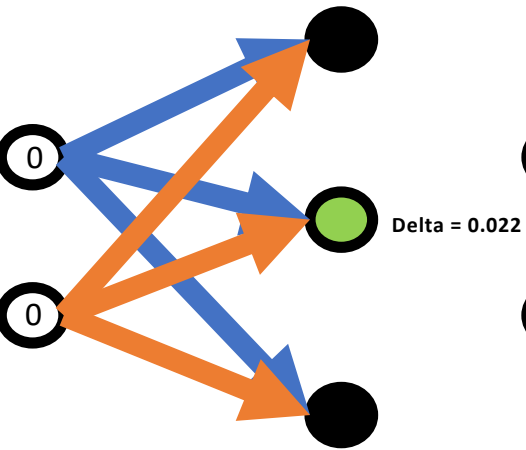


$$peson + 1 = (peson * momento) + (\textit{entrada} * \textit{delta} * \textit{taxa de aprendizagem})$$

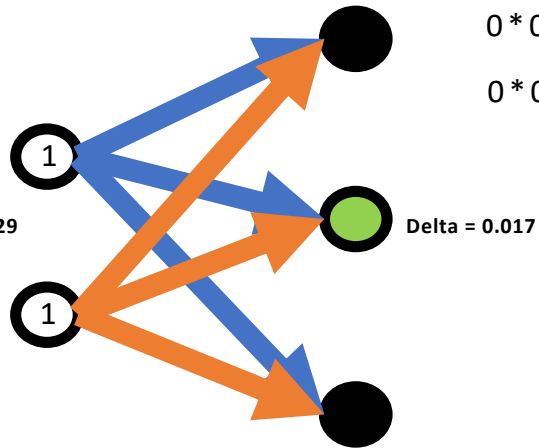
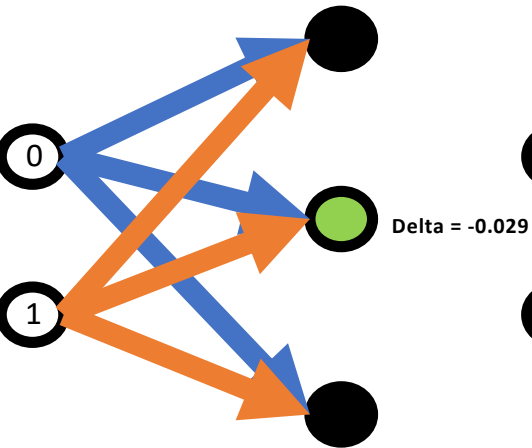
$$0 * 0.000 + 0 * (-0.001) + 1 * (-0.001) + 1 * 0.000 = -0.000 \quad \text{primeira entrada}$$

$$0 * 0.000 + 1 * (-0.001) + 0 * (-0.001) + 1 * 0.000 = -0.000 \quad \text{segunda entrada}$$

Arredondado!

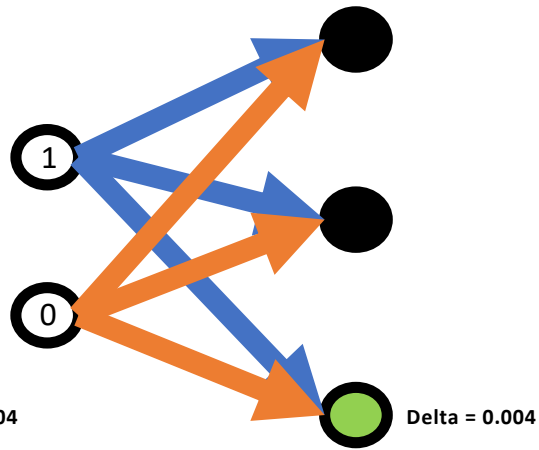
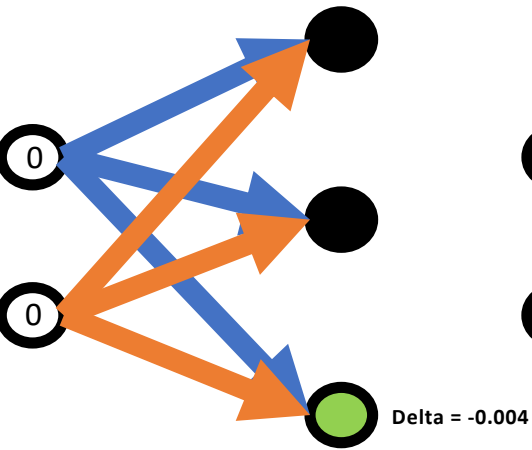


$$peson + 1 = (peson * momento) + (entrada * \mathbf{delta} * taxa \text{ de aprendizagem})$$

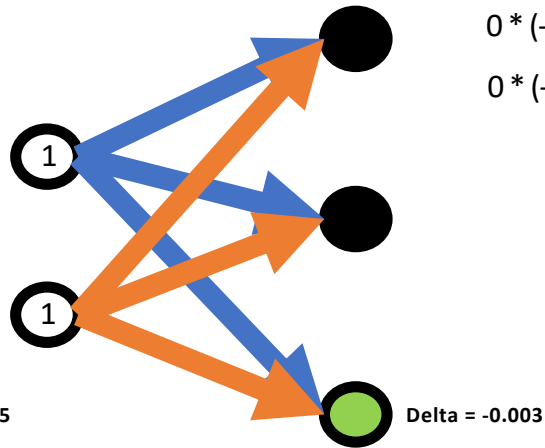
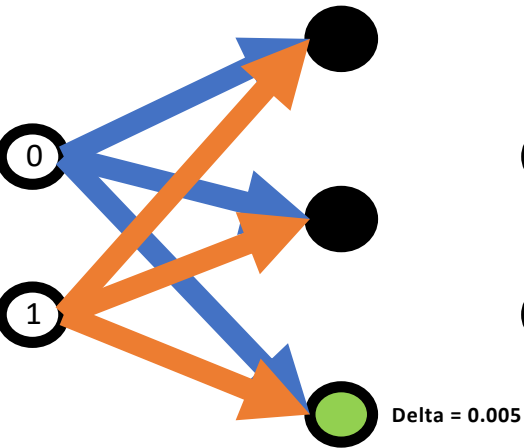


$$0 * 0.022 + 0 * (-0.029) + 1 * (-0.027) + 1 * 0.017 = -0.010$$

$$0 * 0.022 + 1 * (-0.029) + 0 * (-0.027) + 1 * 0.017 = -0.012$$



$$peso_{n+1} = (peso_n * momento) + (entrada * \mathbf{delta} * taxa\ de\ aprendizagem)$$



$$0 * (-0.004) + 0 * 0.005 + 1 * 0.004 + 1 * (-0.003) = 0.001$$

$$0 * (-0.004) + 1 * 0.005 + 0 * 0.004 + 1 * (-0.003) = 0.002$$

Taxa de aprendizagem = 0.3

Momento = 1

Entrada x delta

-0.000	-0.010	0.001
-0.000	-0.012	0.002

$$Peso_{n+1} = (peso_n * momento) + (entrada * delta * taxa\ de\ aprendizagem)$$

$$(-0.424 * 1) + (-0.000) * 0.3 = -0.424$$

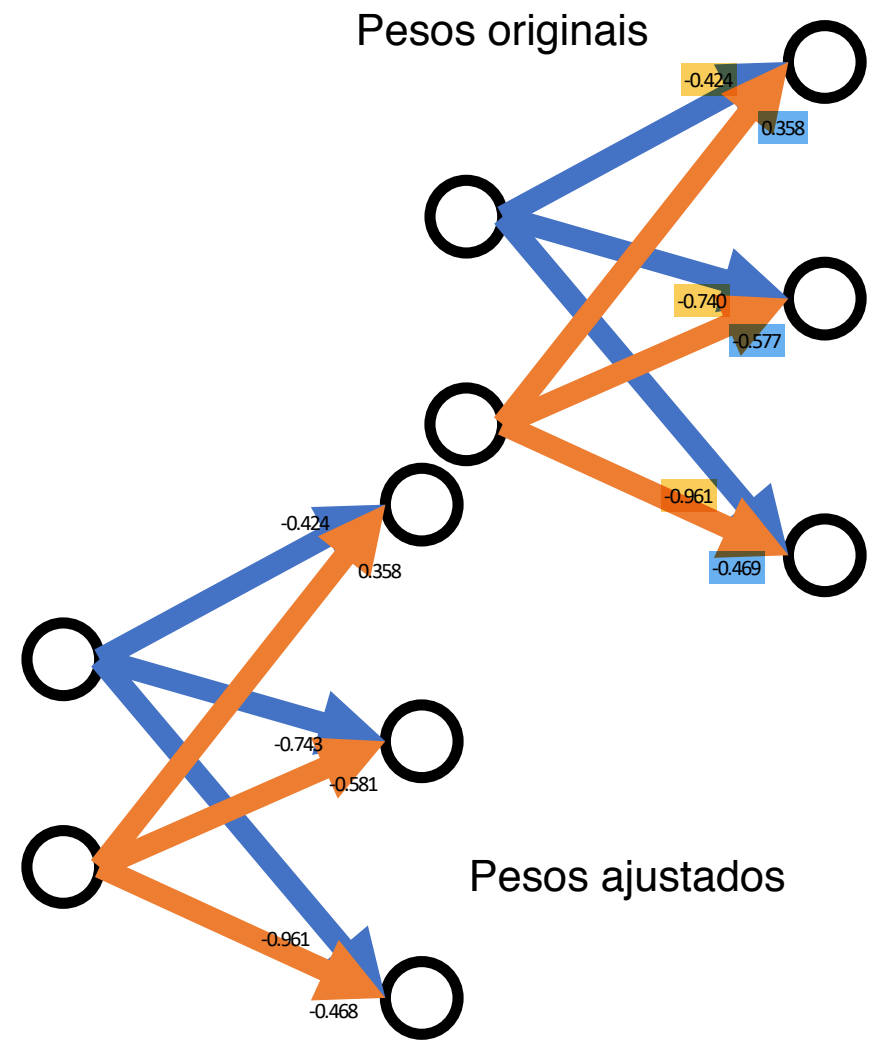
$$(0.358 * 1) + (-0.000) * 0.3 = 0.358$$

$$(-0.740 * 1) + (-0.010) * 0.3 = -0.743$$

$$(-0.577 * 1) + (-0.012) * 0.3 = -0.581$$

$$(-0.961 * 1) + 0.001 * 0.3 = -0.961$$

$$(-0.469 * 1) + 0.002 * 0.3 = -0.468$$



Com os valores do delta, faremos a atualização dos pesos

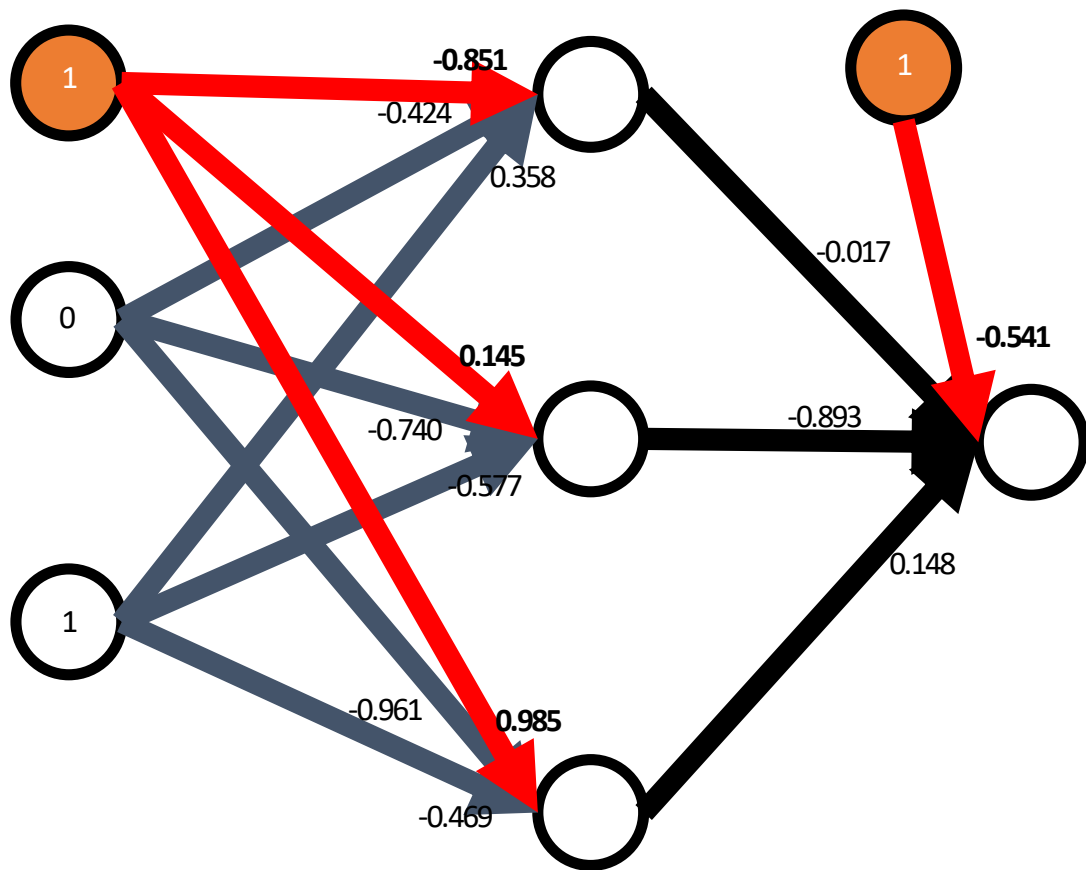
Bias

Valores diferentes mesmo se todas as entradas forem zero

Muda a saída com a unidade de bias

Normalmente os algoritmos de redes neurais, já tem essa camada de bias de forma transparente para a implementação

Ajuda a ajustar os valores de peso, dependendo das entradas.



Erro

- Algoritmo mais simples
 - $\text{erro} = \text{respostaCorreta} - \text{respostaCalculada}$

x1	x2	Classe	Calculado	Erro
0	0	0	0.406	-0.406
0	1	1	0.432	0.568
1	0	1	0.437	0.563
1	1	0	0.458	-0.458

Mean square error (MSE) e Root mean square error (RMSE)

$$MSE = \frac{1}{N} \sum_{i=1}^N (f_i - y_i)^2$$

$$RMSE = \sqrt{\frac{1}{N} \sum_{i=1}^N (f_i - o_i)^2}$$

x1	x2	Classe	Calculado	Erro
0	0	0	0.406	$(0 - 0.406)^2 = 0.164$
0	1	1	0.432	$(1 - 0.432)^2 = 0.322$
1	0	1	0.437	$(1 - 0.437)^2 = 0.316$
1	1	0	0.458	$(0 - 0.458)^2 = 0.209$

Soma = 1.011

MSE = 1.011 / 4 = 0.252

RMSE = 0.501

História do crédito	Dívida	Garantias	Renda anual	Risco
3	1	1	1	100
2	1	1	2	100
2	2	1	2	010
2	2	1	3	100
2	2	1	3	001
2	2	2	3	001
3	2	1	1	100
3	2	2	3	010
1	2	1	3	001
1	1	2	3	001
1	1	1	1	100
1	1	1	2	010
1	1	1	3	001
3	1	1	2	100

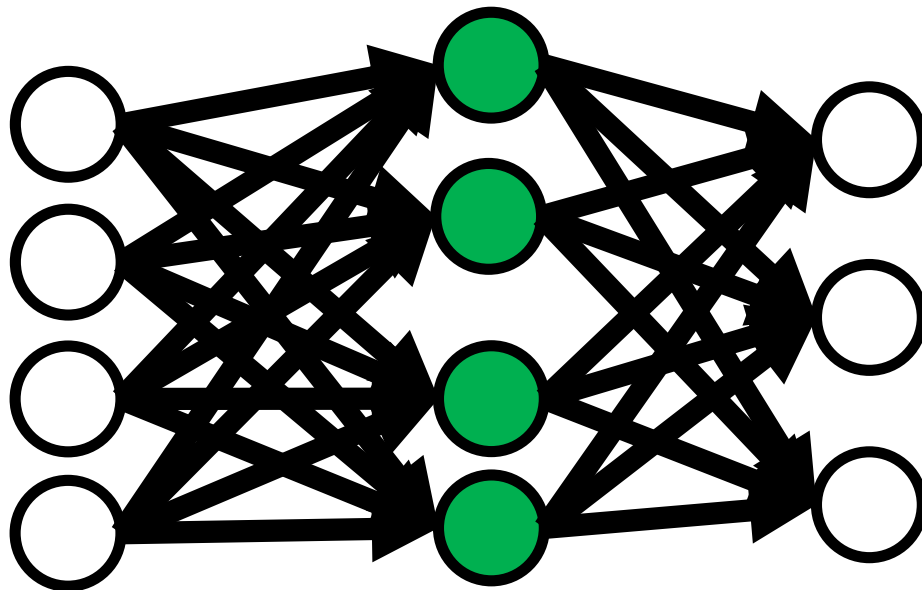
Calcula o erro para todos os registros e atualiza os pesos

Batch gradient descent

Calcula o erro para cada registro e atualiza os pesos

Stochastic gradient descent

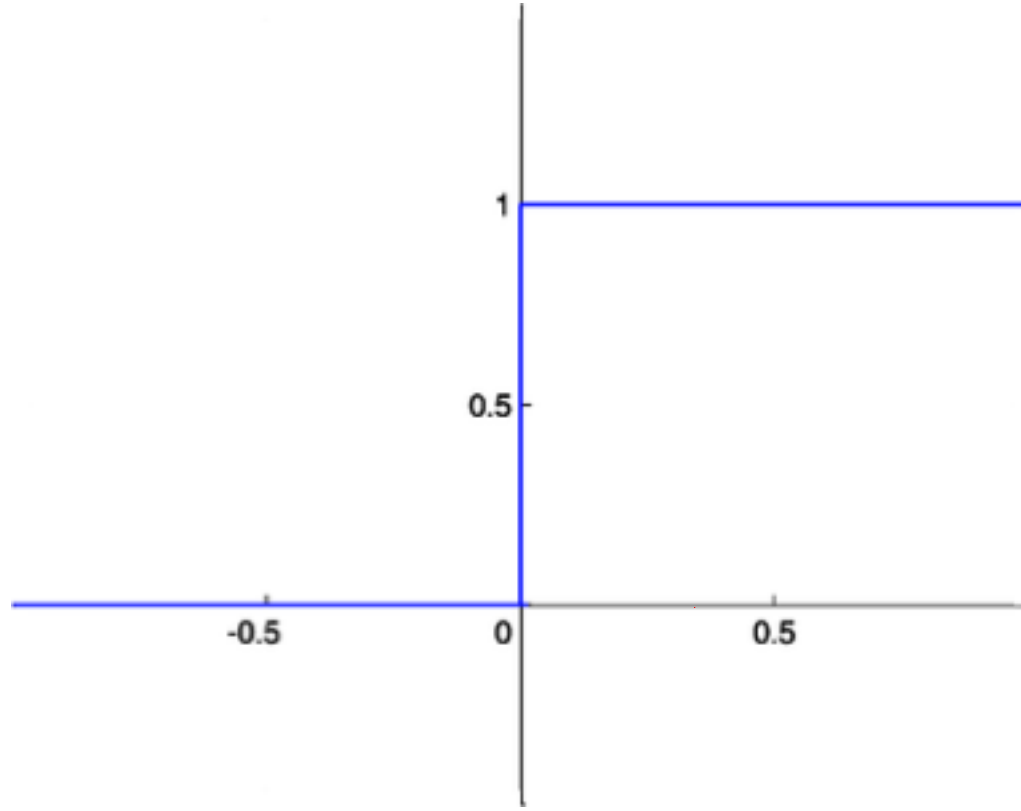
História do crédito	Dívida	Garantias	Renda anual	Risco
3	1	1	1	100
2	1	1	2	100
2	2	1	2	010
2	2	1	3	100
2	2	1	3	001
2	2	2	3	001
3	2	1	1	100
3	2	2	3	010
1	2	1	3	001
1	1	2	3	001
1	1	1	1	100
1	1	1	2	010
1	1	1	3	001
3	1	1	2	100



Gradient descent

- Stochastic
 - Ajuda a prevenir mínimos locais (superfícies não convexas)
 - Mais rápido (não precisa carregar todos os dados em memória)
- Mini batch gradient descent
 - Escolhe um número de registros para rodar e atualizar os pesos

Step (função degrau)

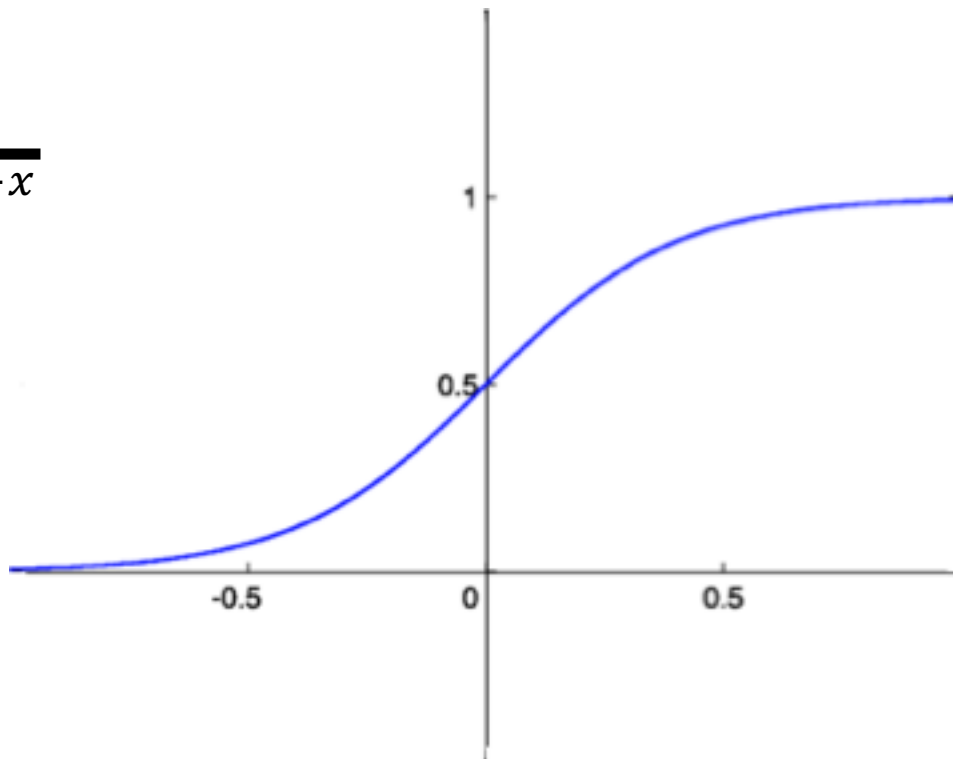


Valor 0 ou 1

Problemas linearmente separáveis

Sigmoid (função sigmoide)

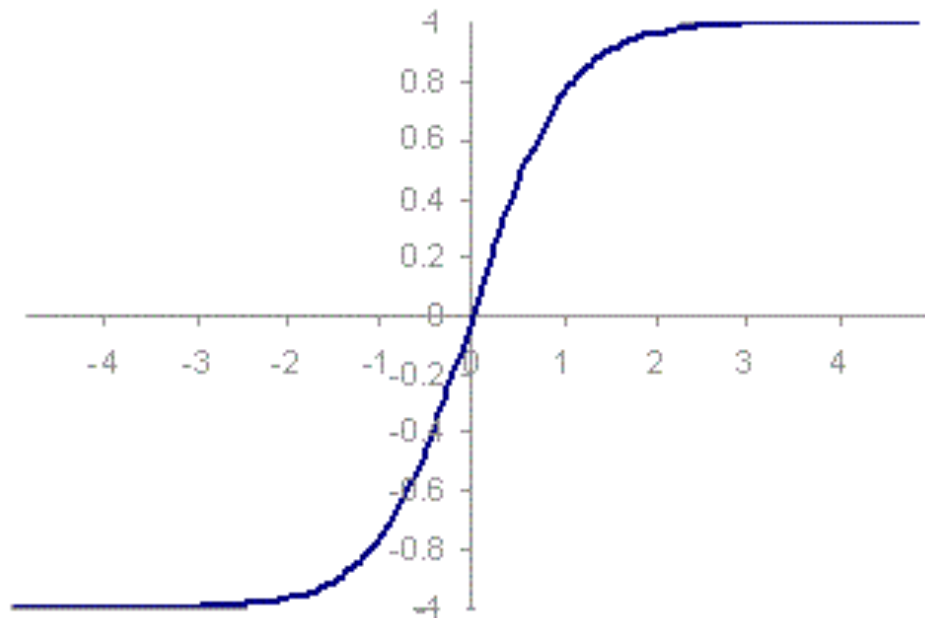
$$y = \frac{1}{1 + e^{-x}}$$



Valores entre 0 e 1

Hyperbolic tangent (função tangente hiperbólica)

$$Y = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$



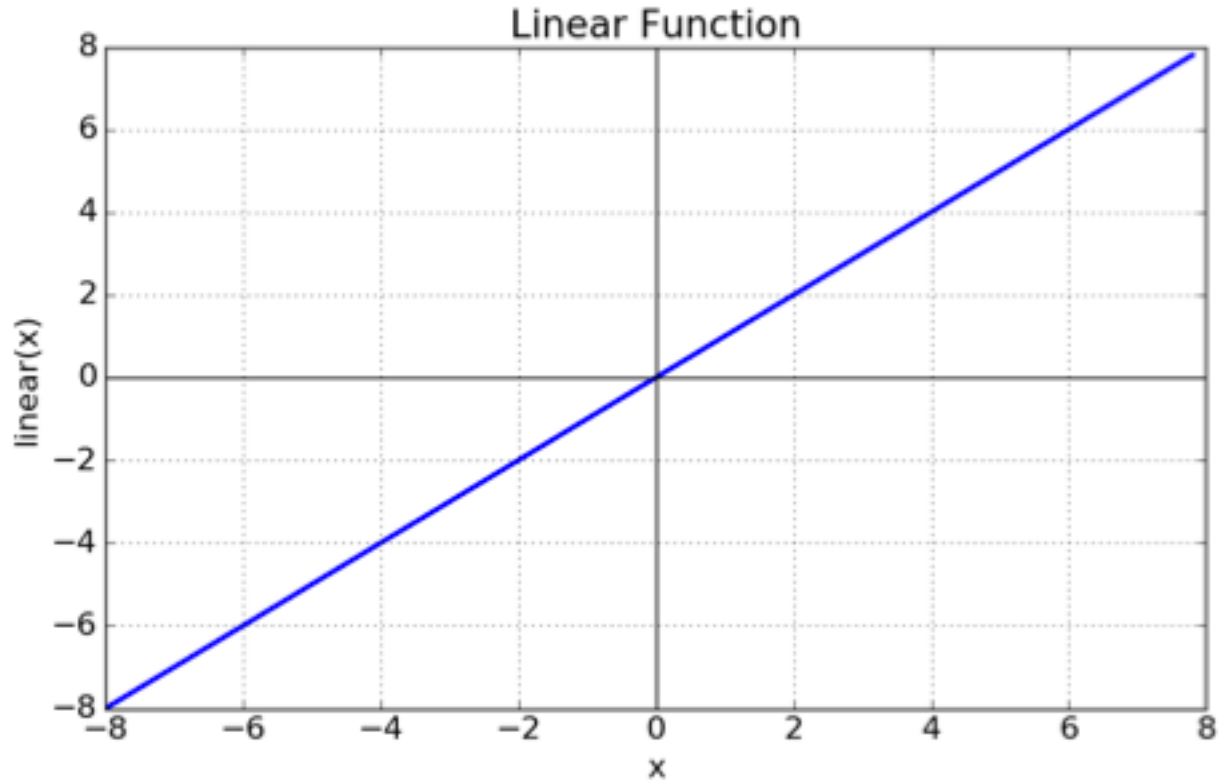
Valores entre -1 e 1

ReLU (rectified linear units)

$$Y = \max(0, x)$$



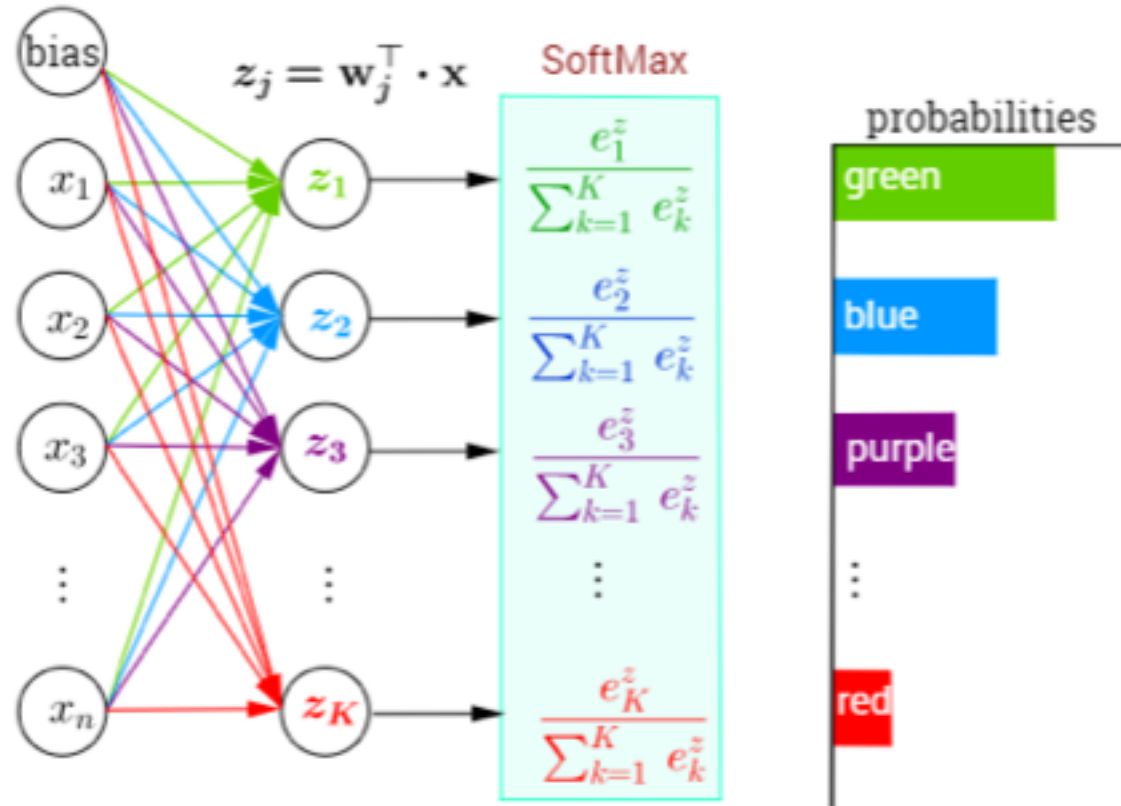
Linear



Problemas de regressão (previsão de números)

Softmax

$$Y = \frac{e(x)}{\sum e(x)}$$



Atividade

- Utilizando qualquer tipo de linguagem, implementar as funções de ativação:
 - Step;
 - Sigmoid;
 - Hyperbolic tangent;
 - ReLU;
 - Softmax;
- Fazer teste com valores aleatórios.