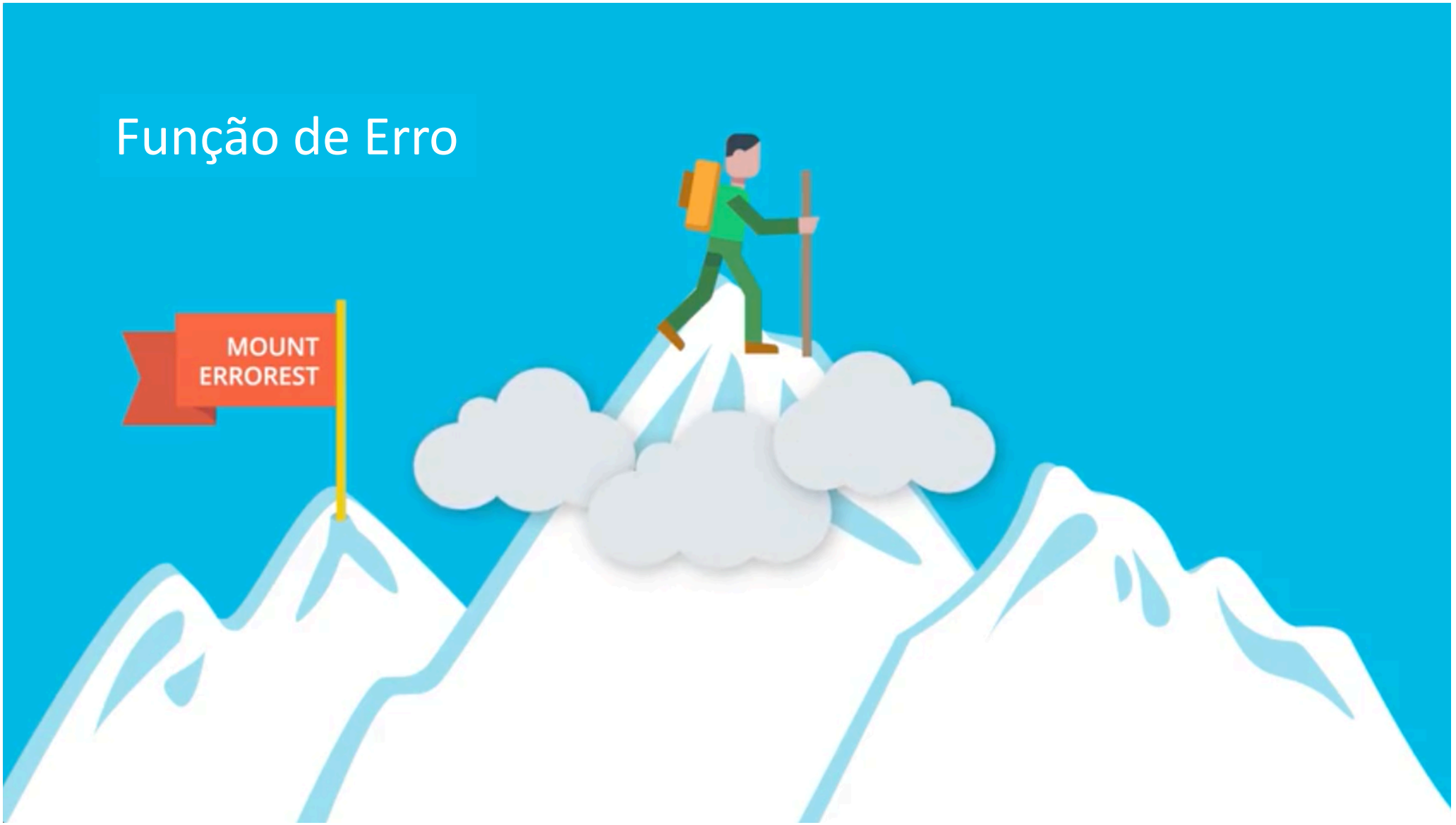
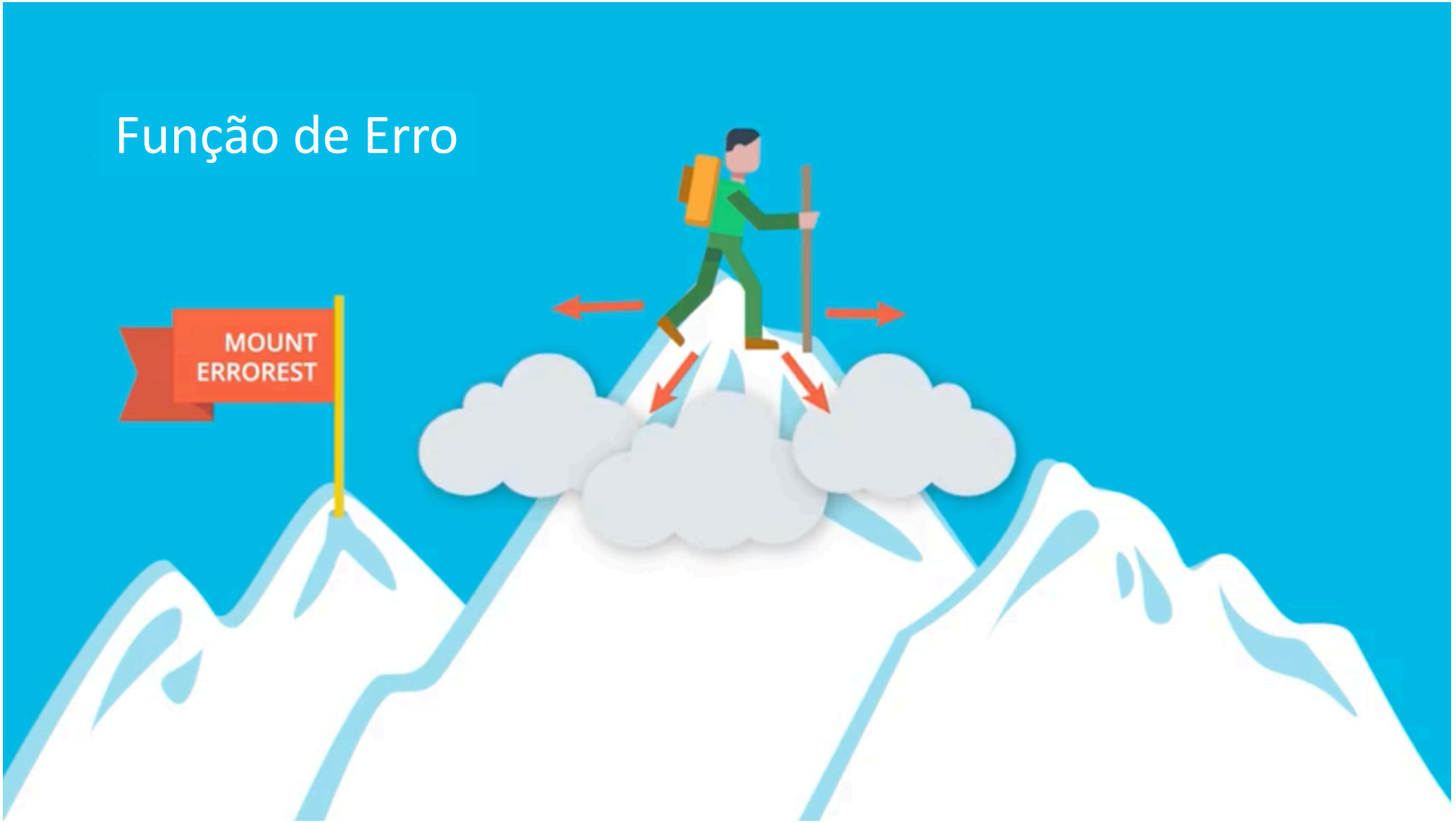
FILIPE ROLIM CORDEIRO (FILIPE.ROLIM@GMAIL.COM)

Função de Erro

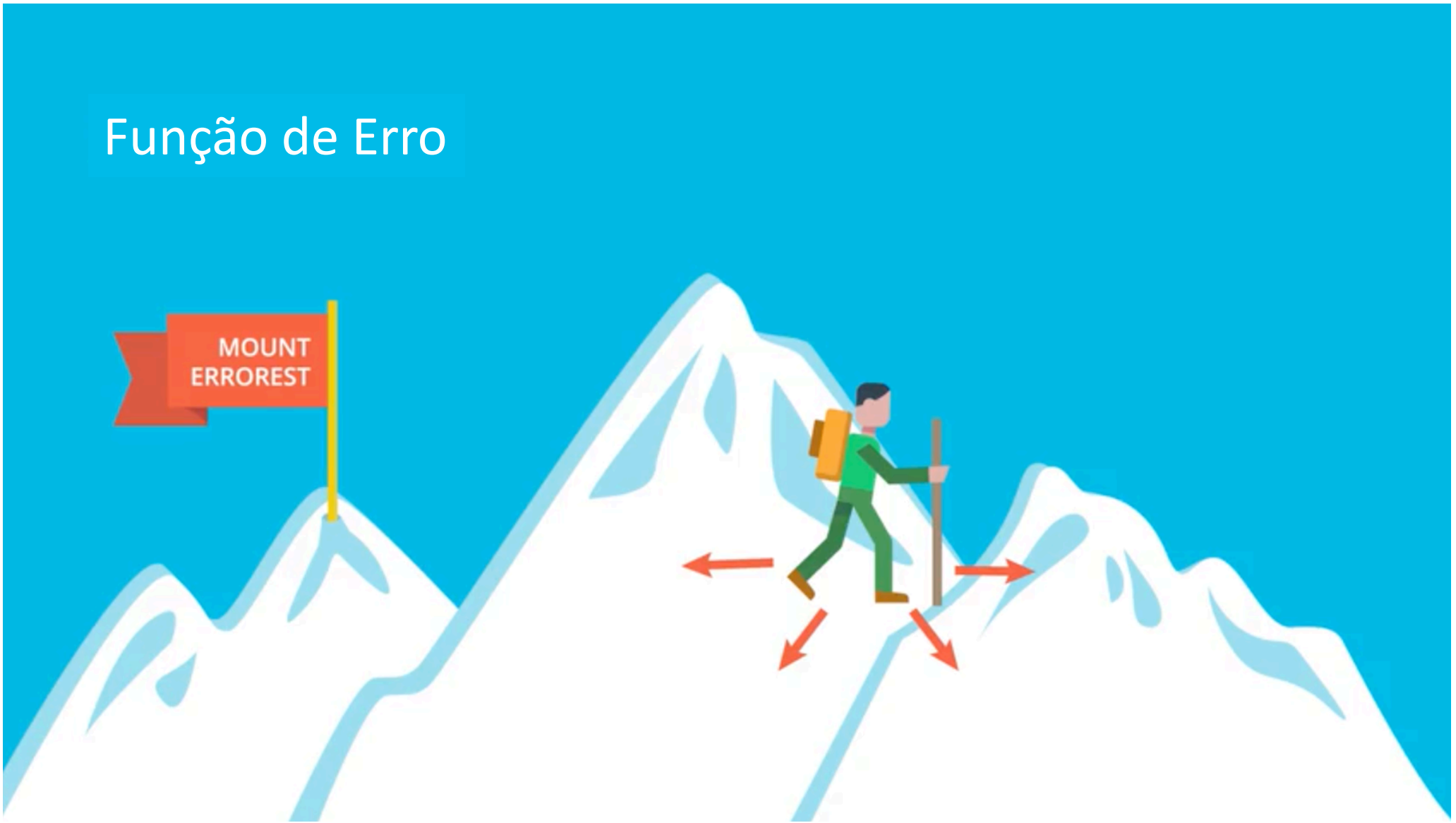
Função de Erro



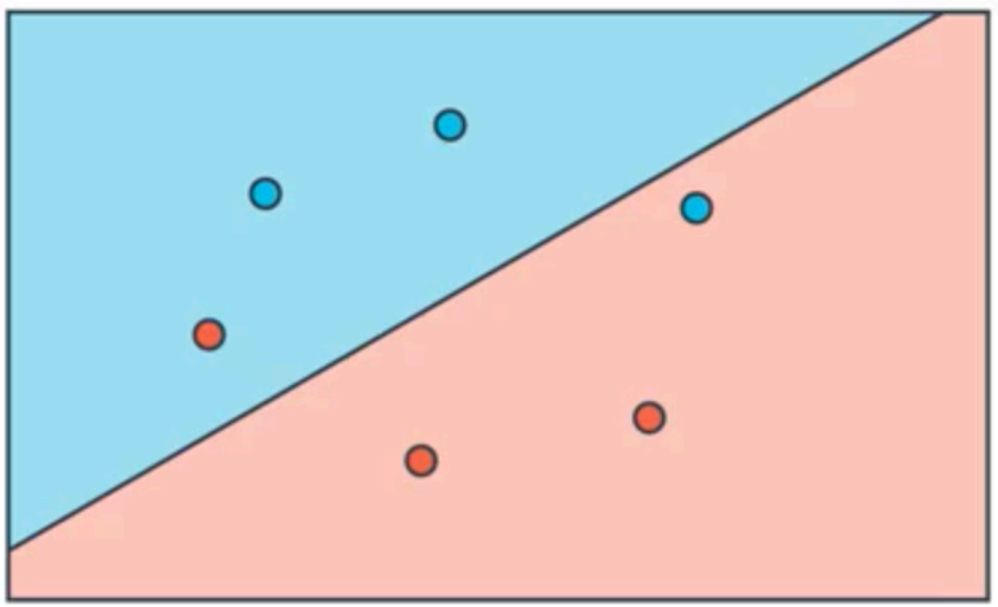
Função de Erro



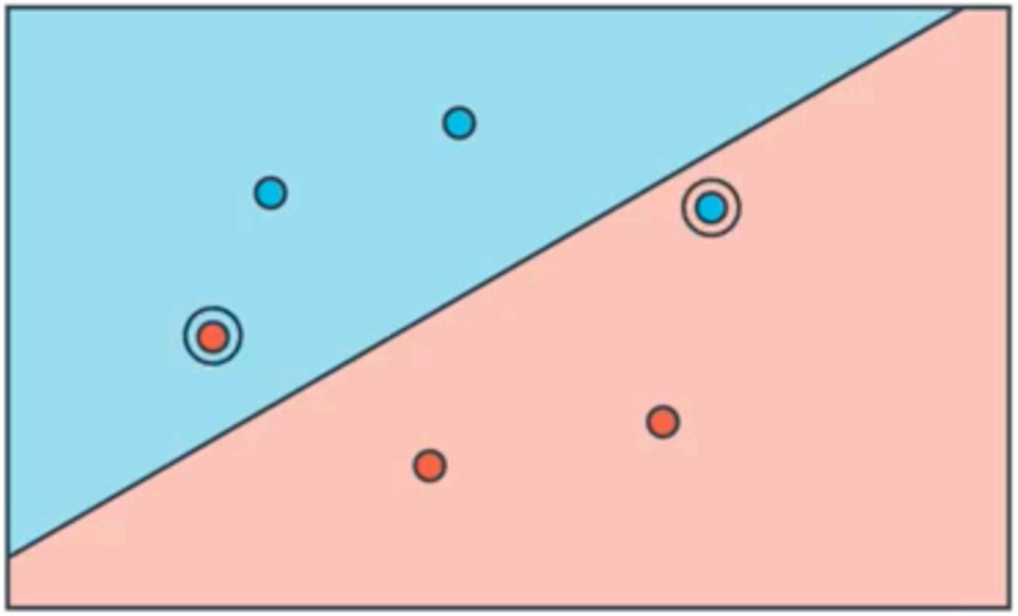
Função de Erro



- O quão errado é o modelo?

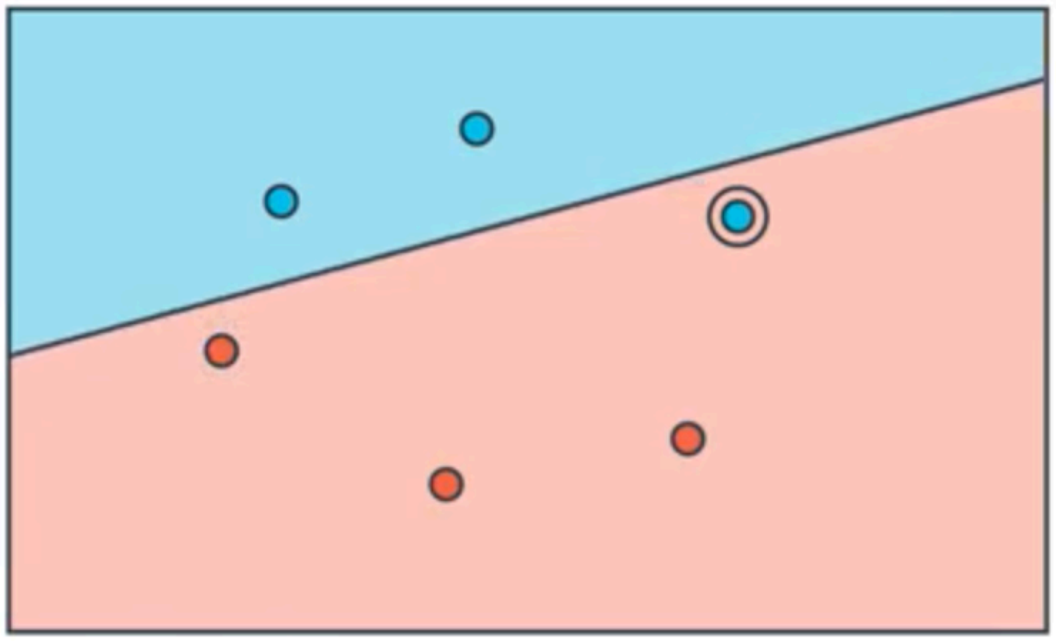


- O quão errado é o modelo?



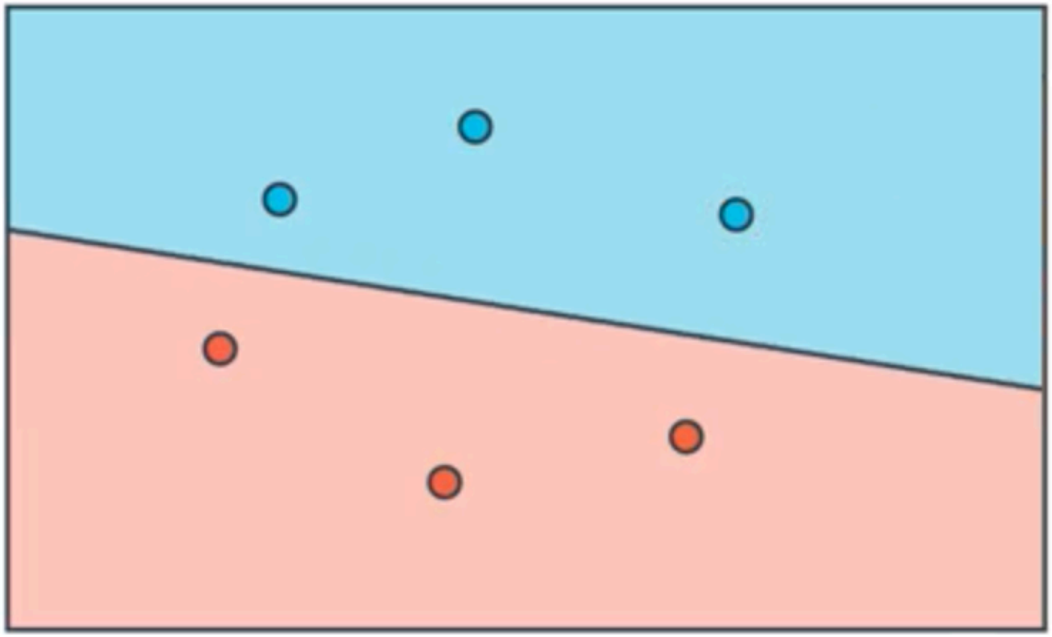
2 erros

- O quão errado é o modelo?



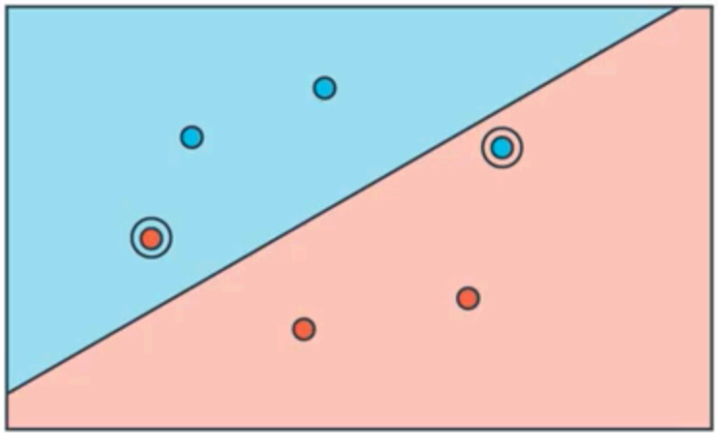
1 erro

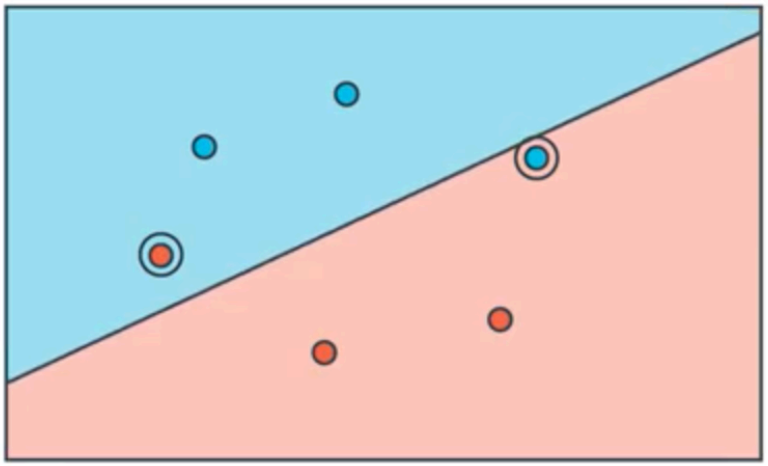
- O quão errado é o modelo?

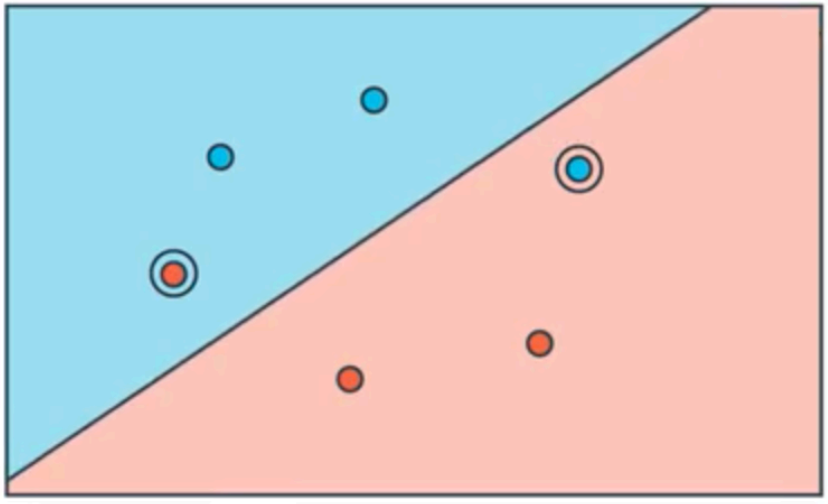


0 erros

- Problema!

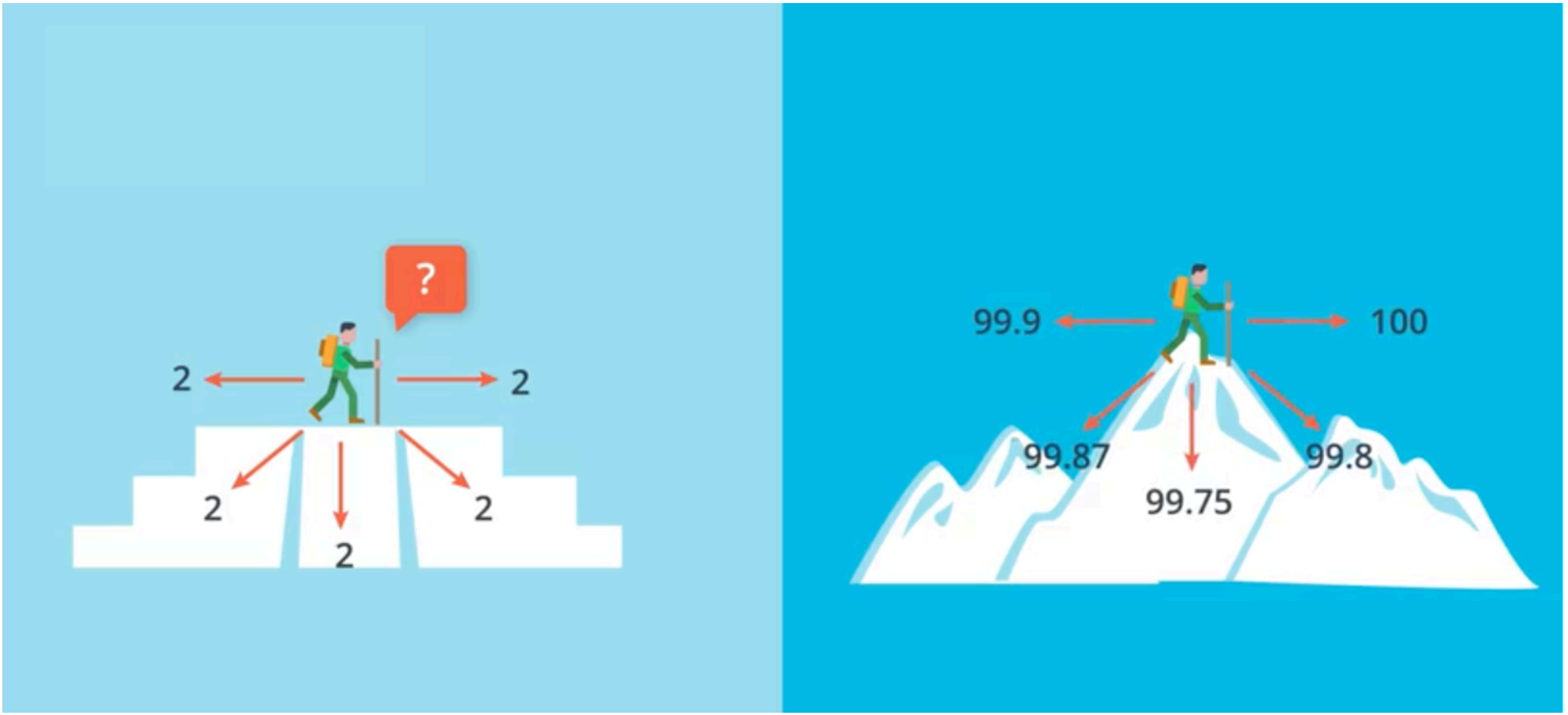




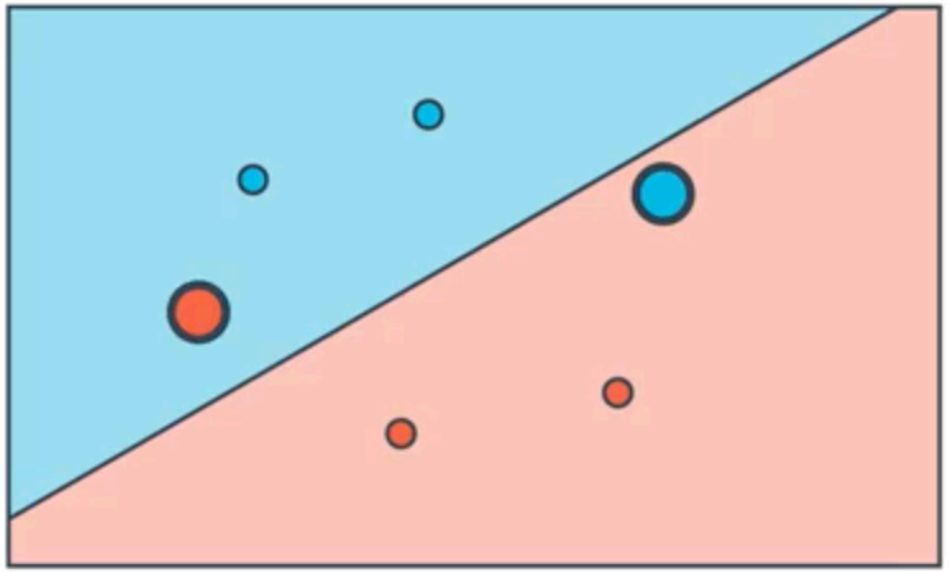


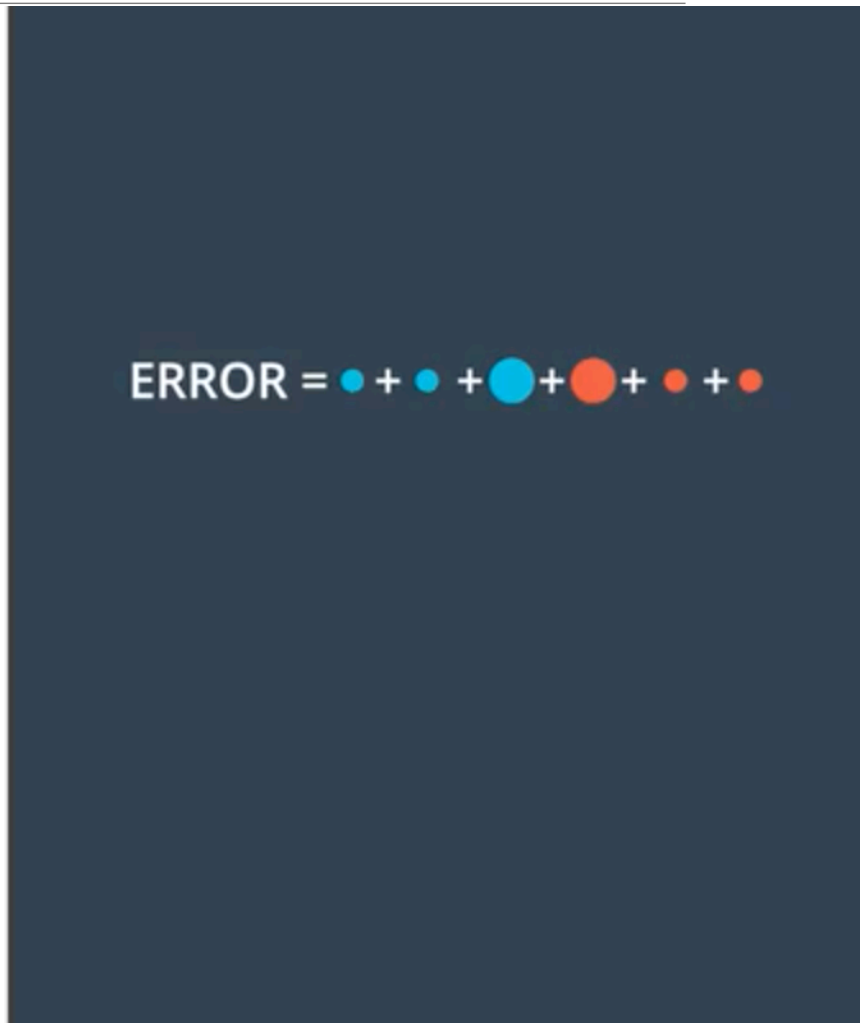
2 erros

Discreto vs Contínuo

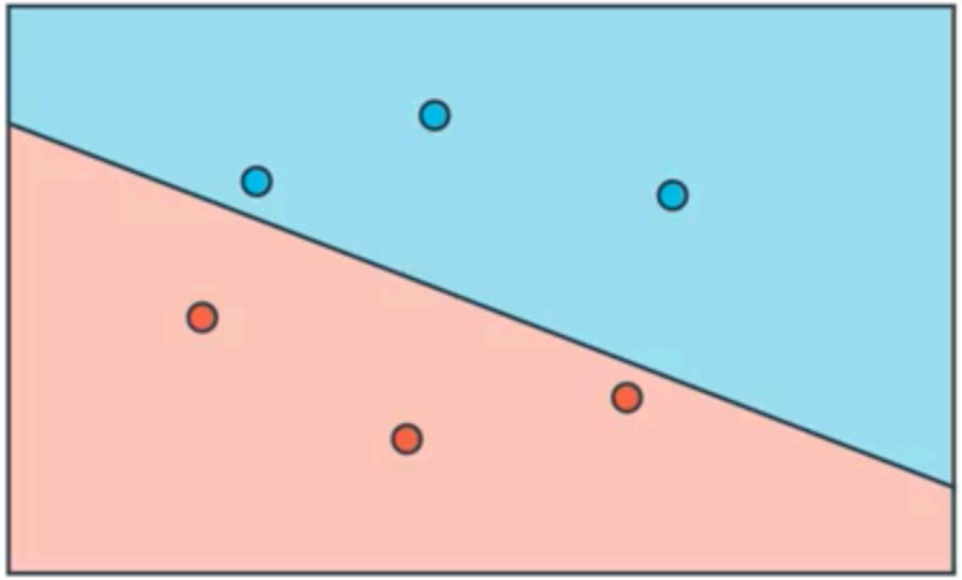


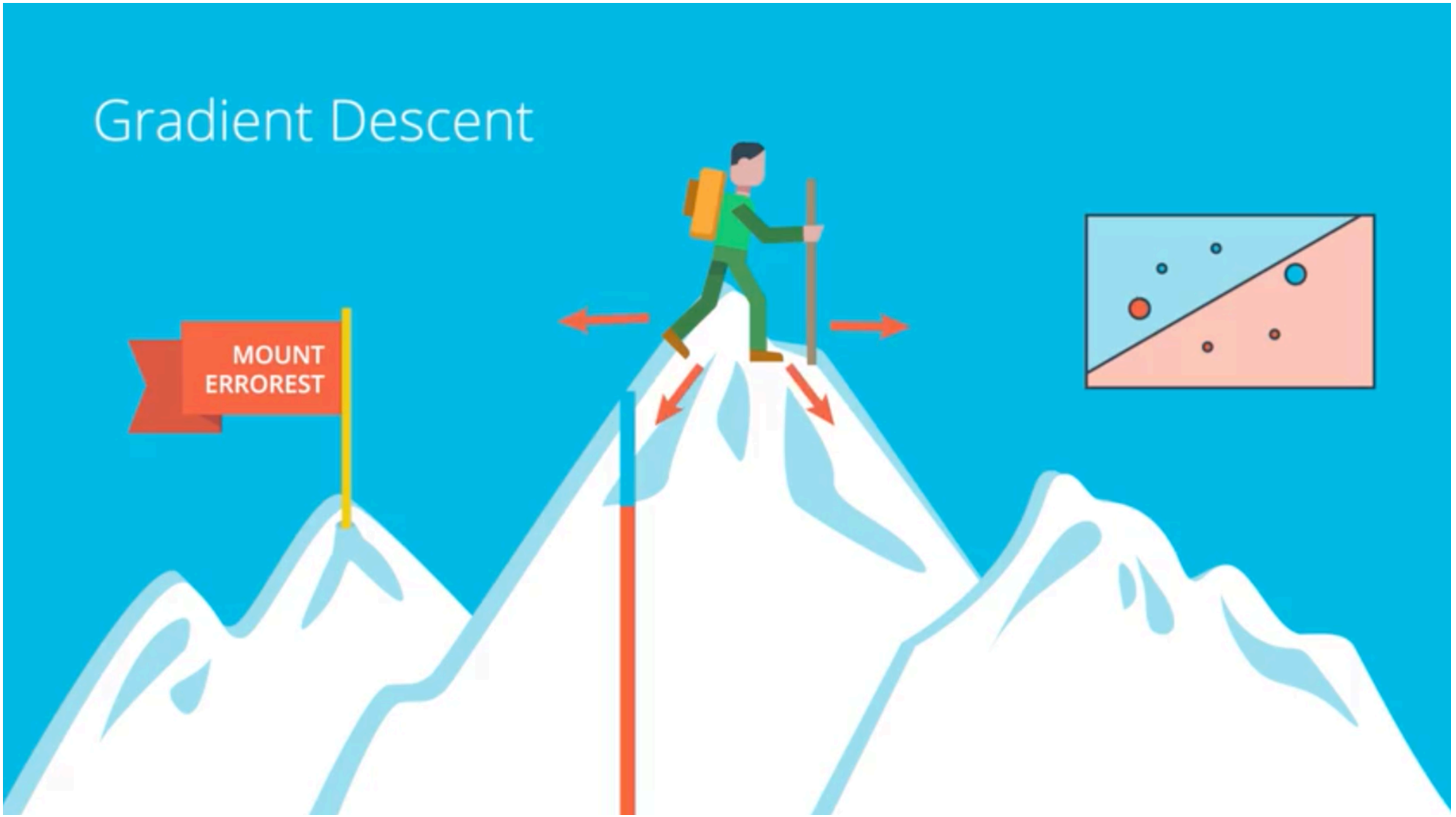
Função de Erro

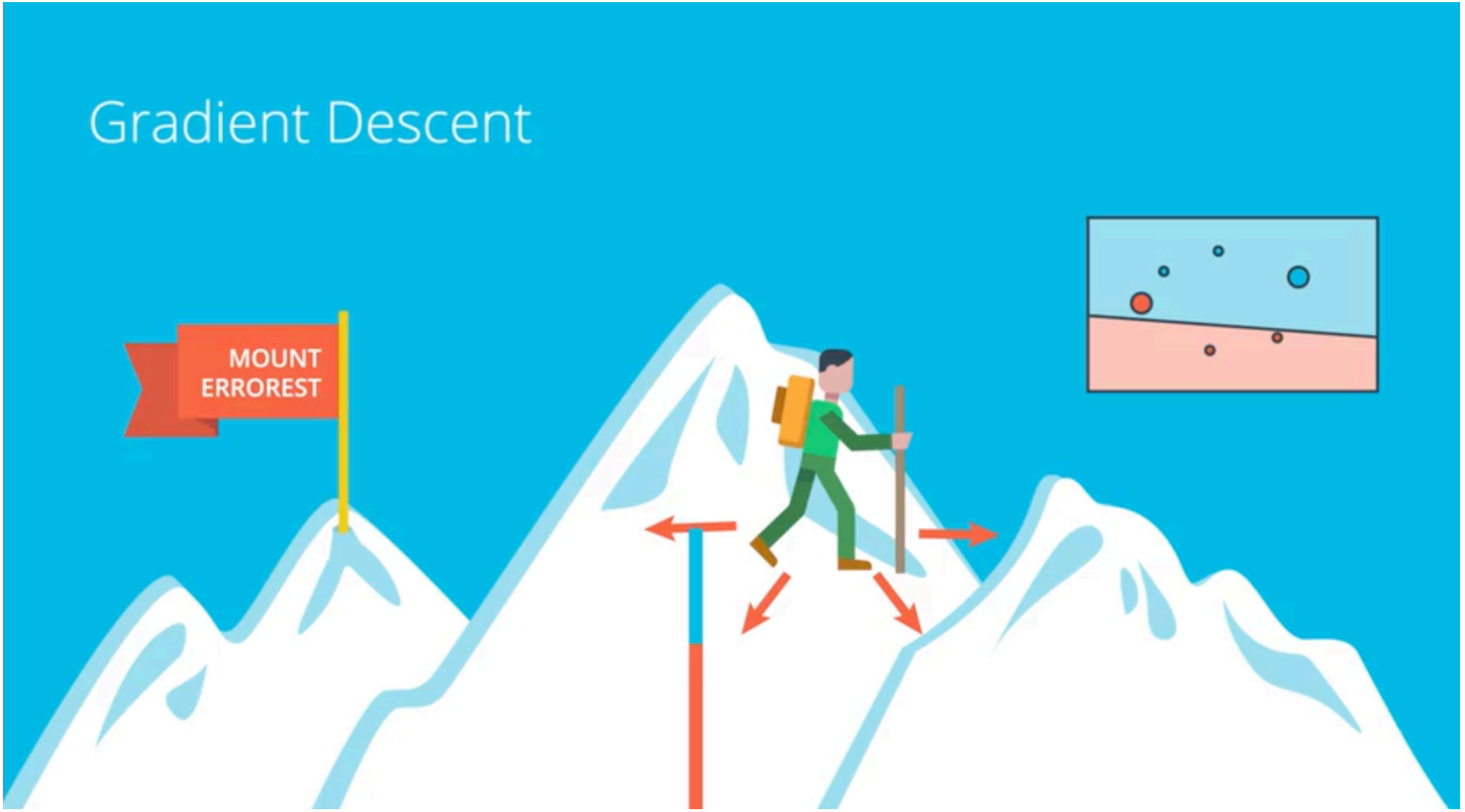


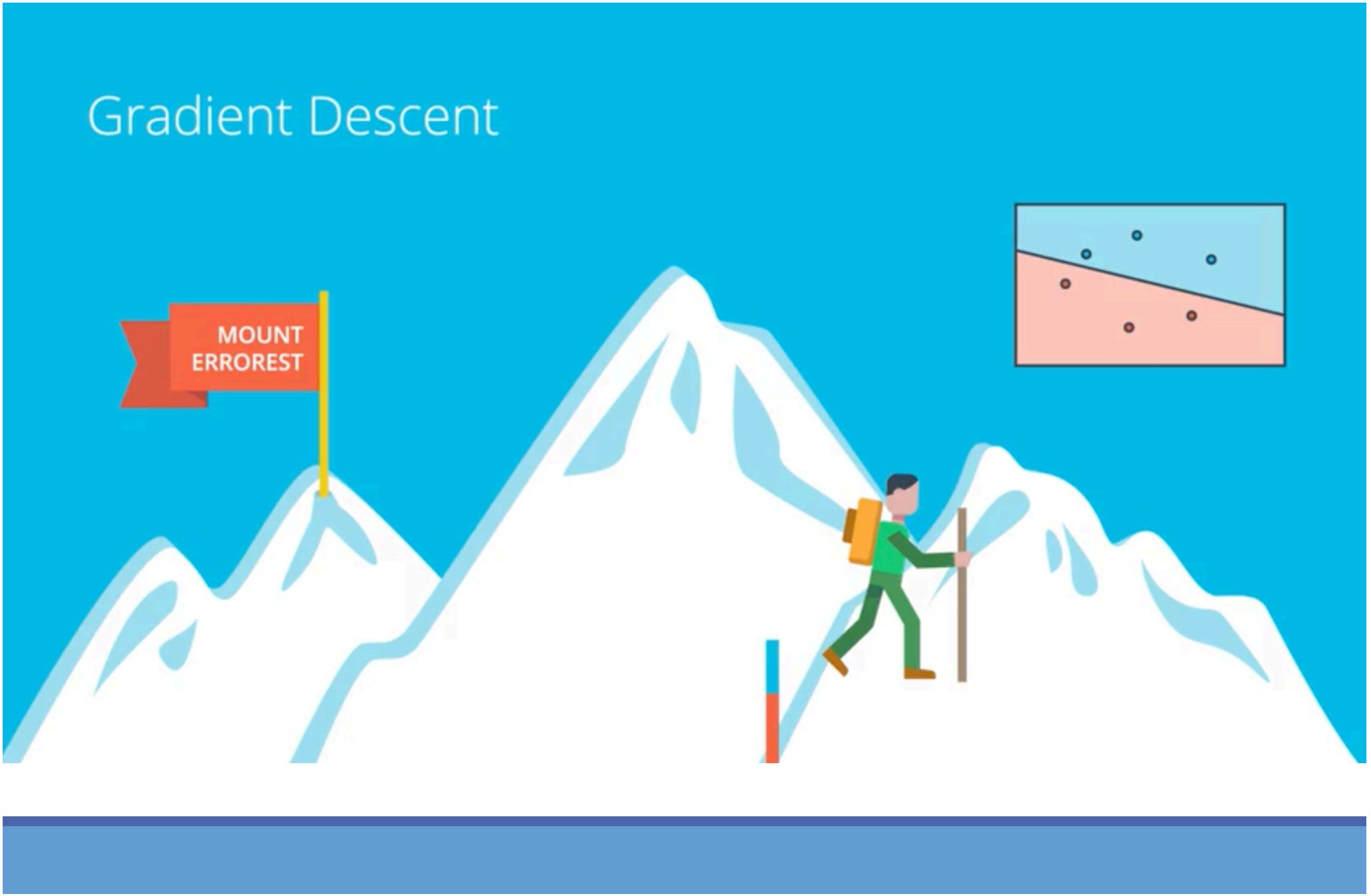


Função de Erro

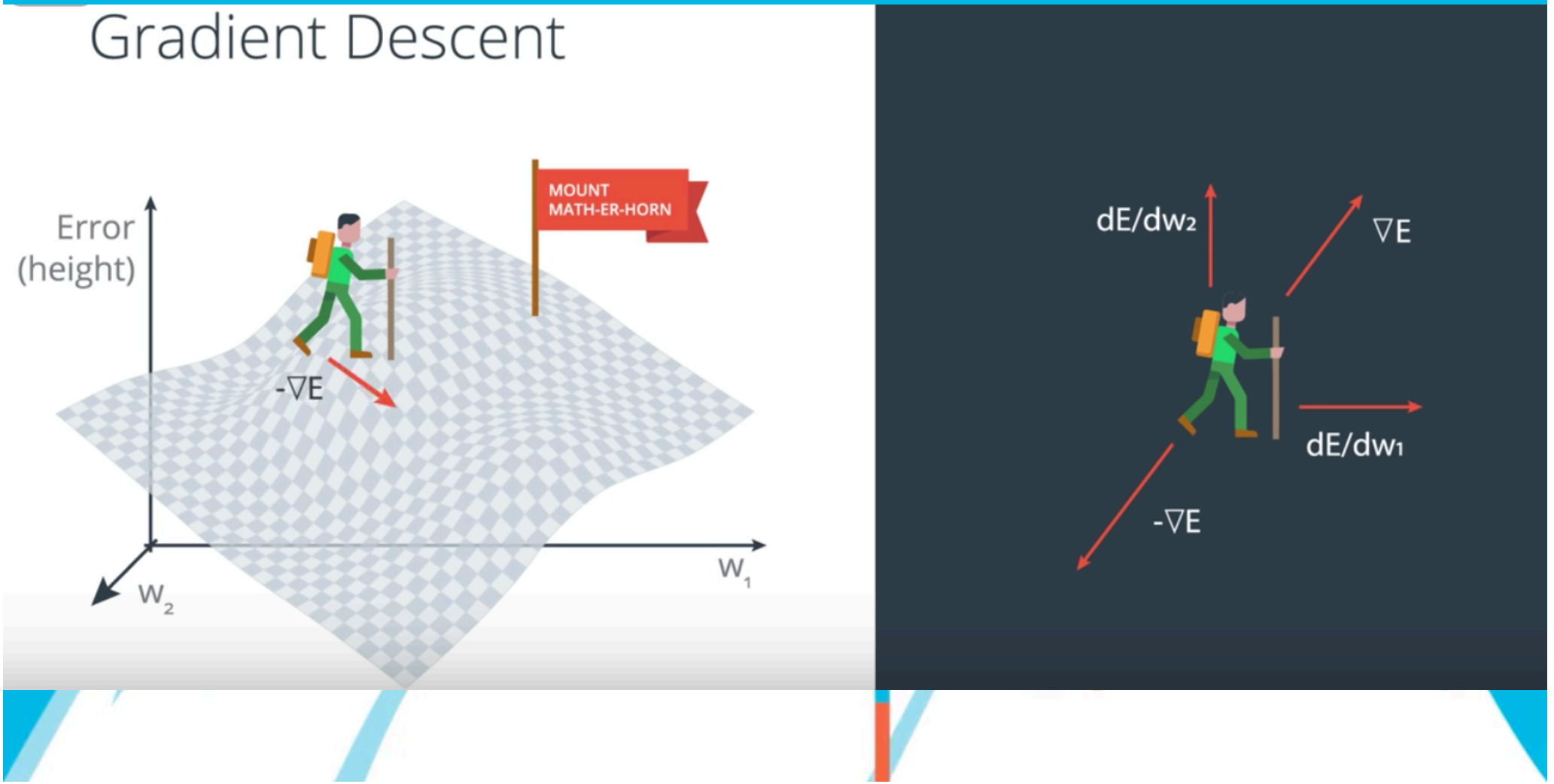



Gradient Descent
MOUNT
ERROREST

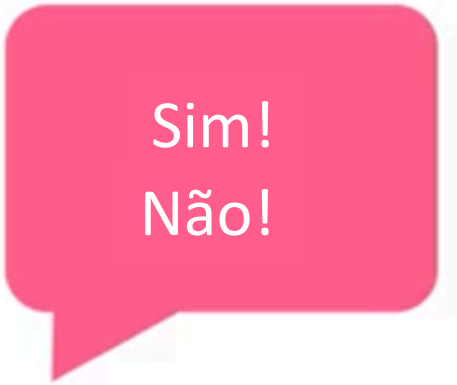

Gradient Descent
MOUNT
ERROREST


Gradient Descent
MOUNT
ERROREST

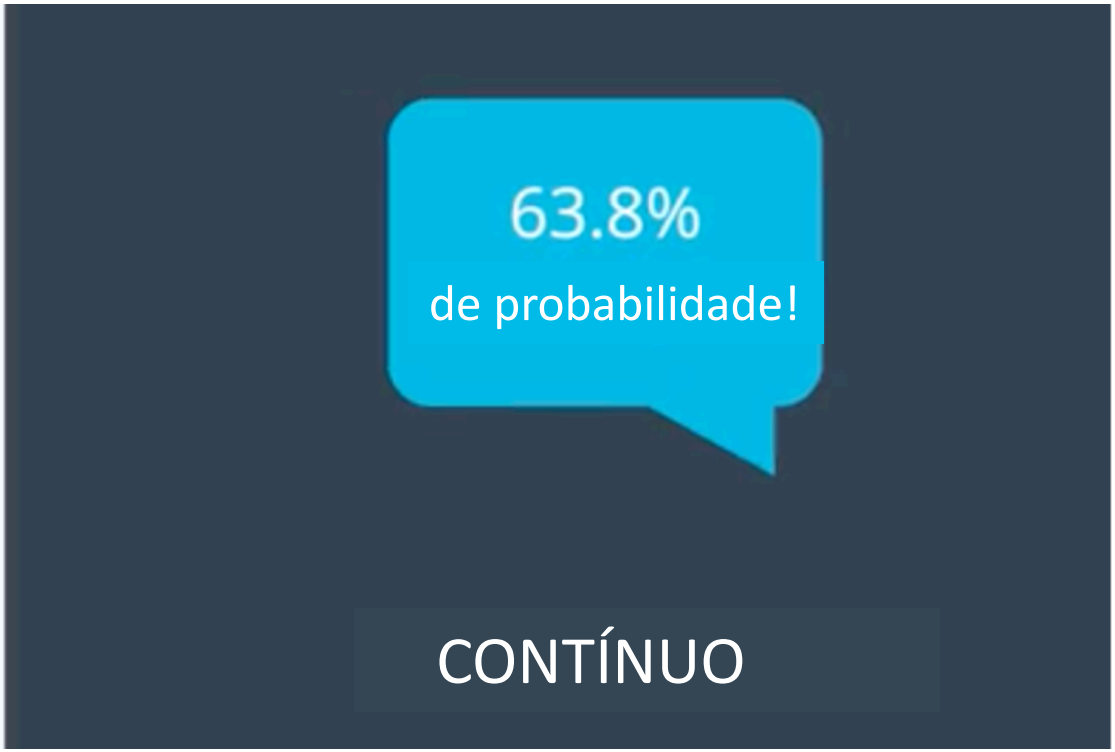
Gradient Descent



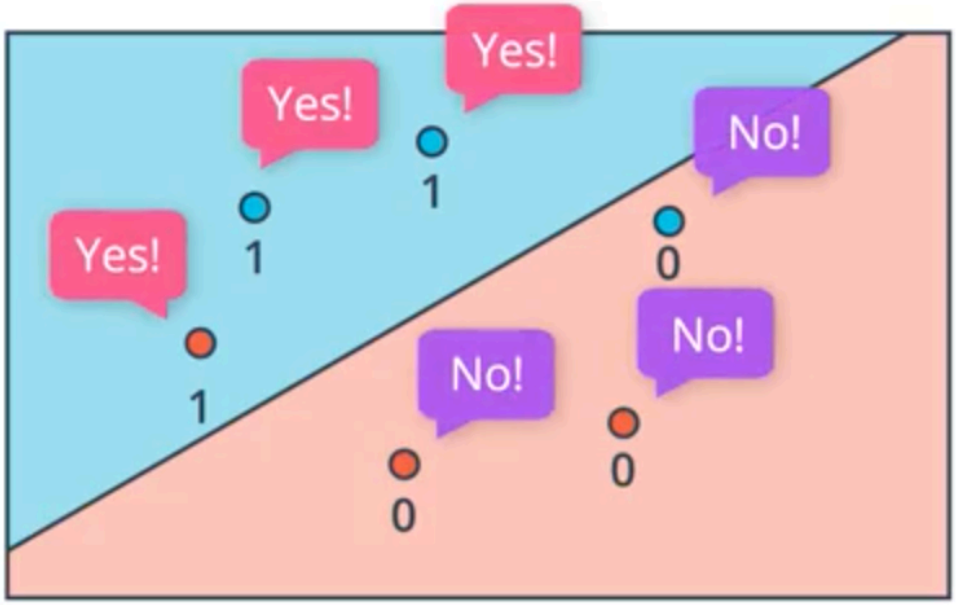
Predições – Discreta vs Contínua



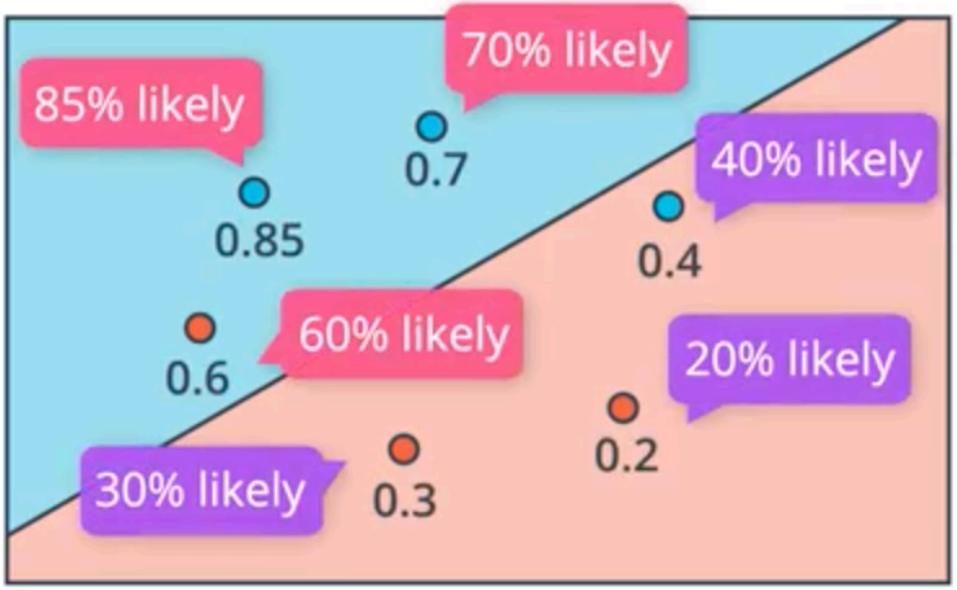
DISCRETO



Predições

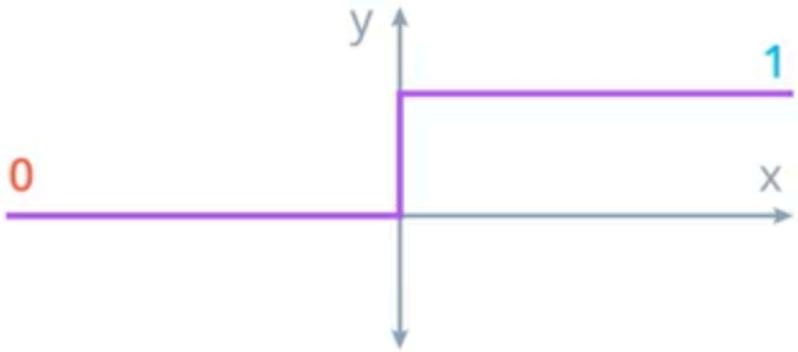


DISCRETO



CONTÍNUO

Funções de Ativação



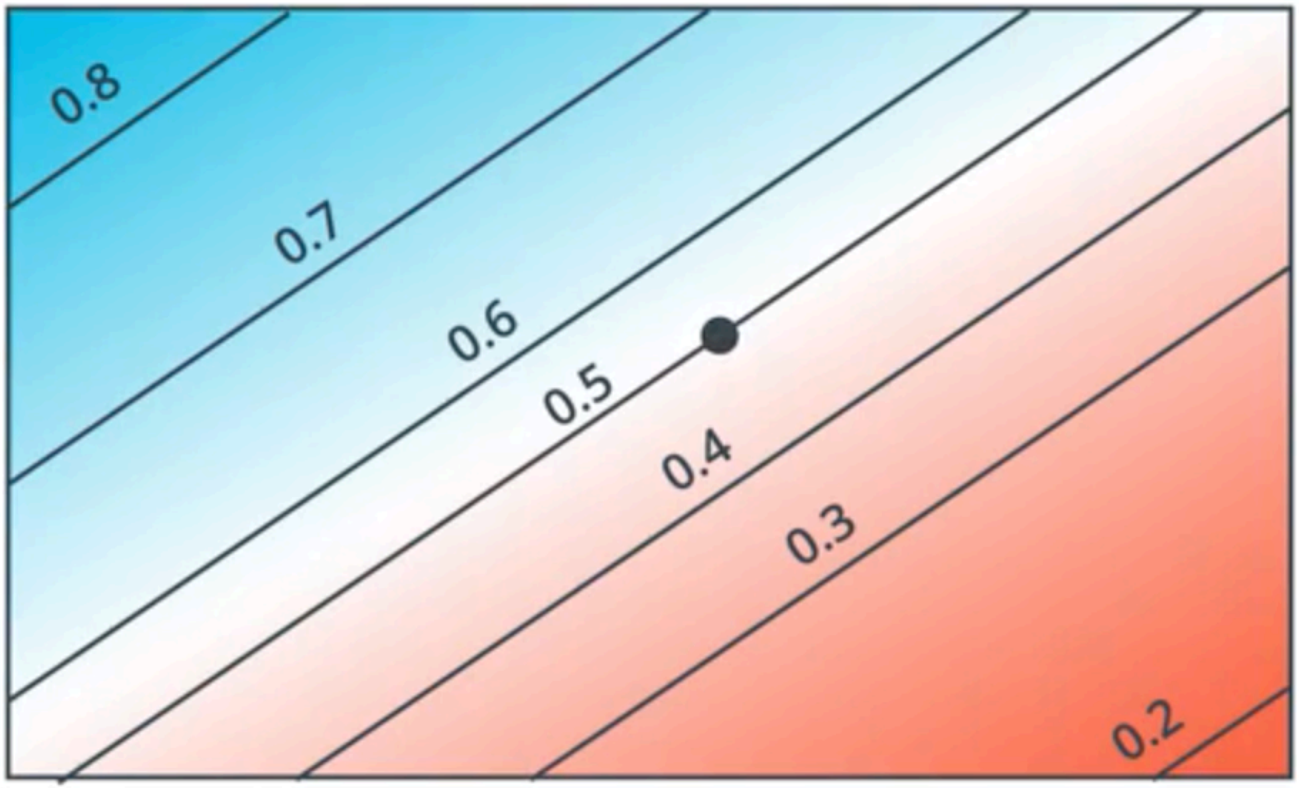
DISCRETO:
Função Degrau

$$y = \begin{cases} 1 \text{ if } x \geq 0 \\ 0 \text{ if } x < 0 \end{cases}$$

CONTÍNUO
Função Sigmoide

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

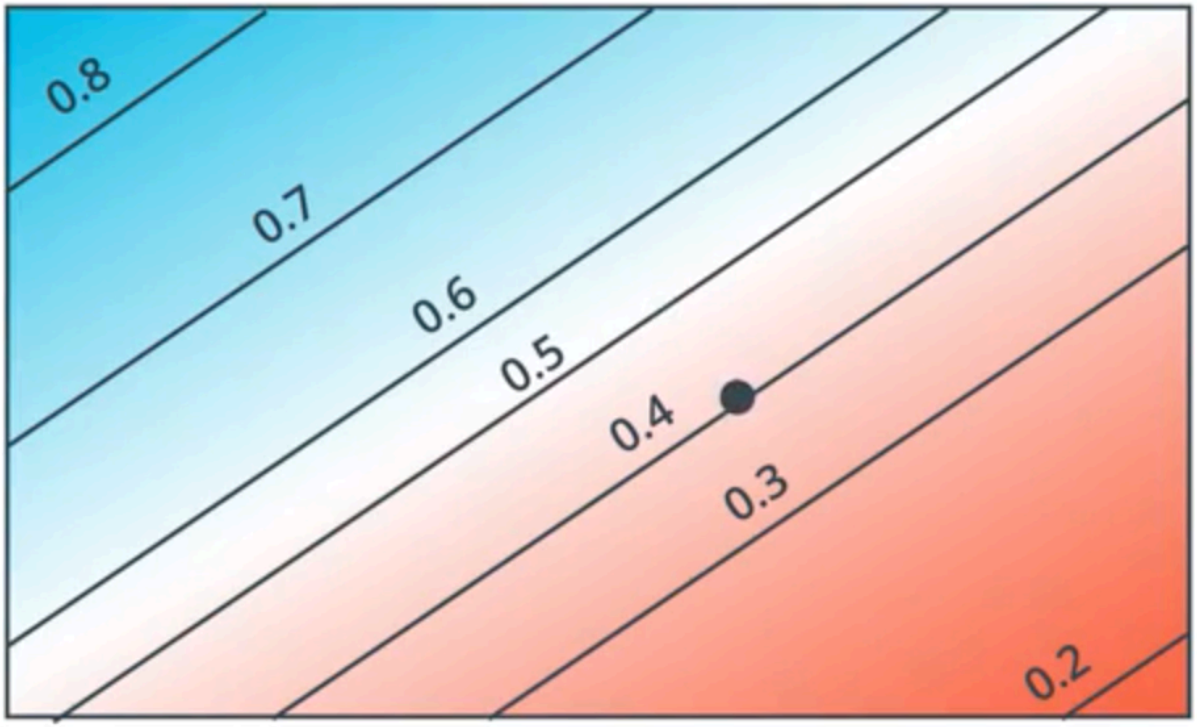
Predições



P(BLUE) = 0.5

P(RED) = 0.5

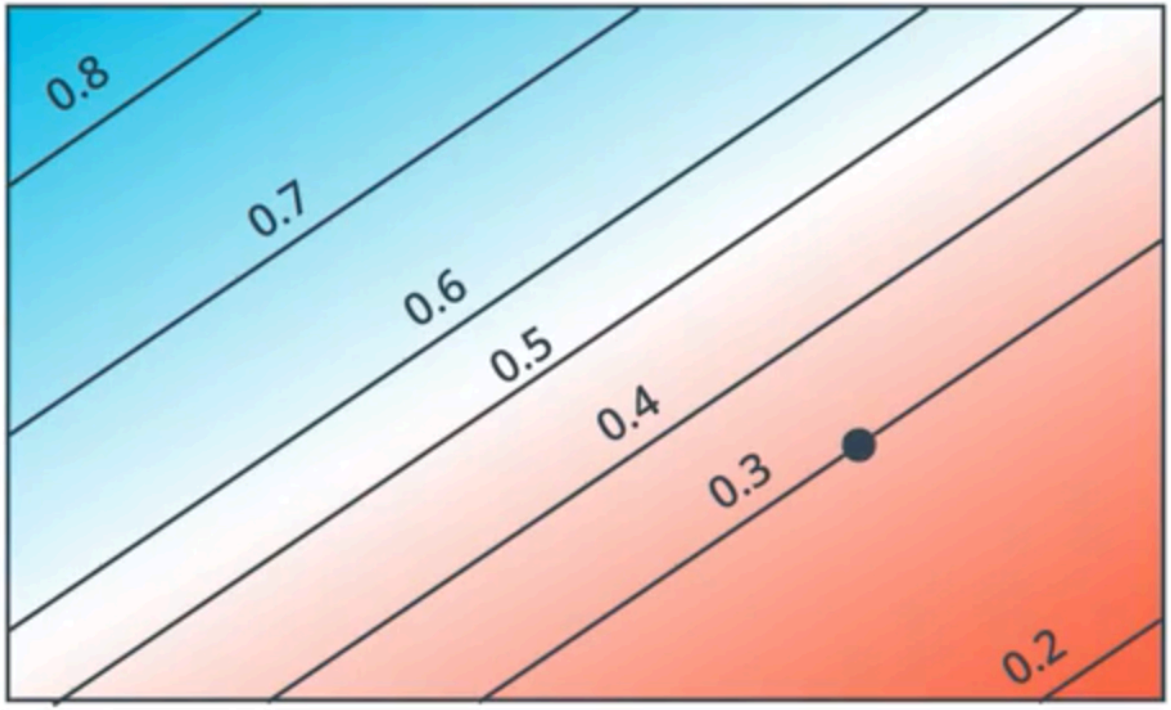
Predições



P(BLUE) = 0.4

P(RED) = 0.6

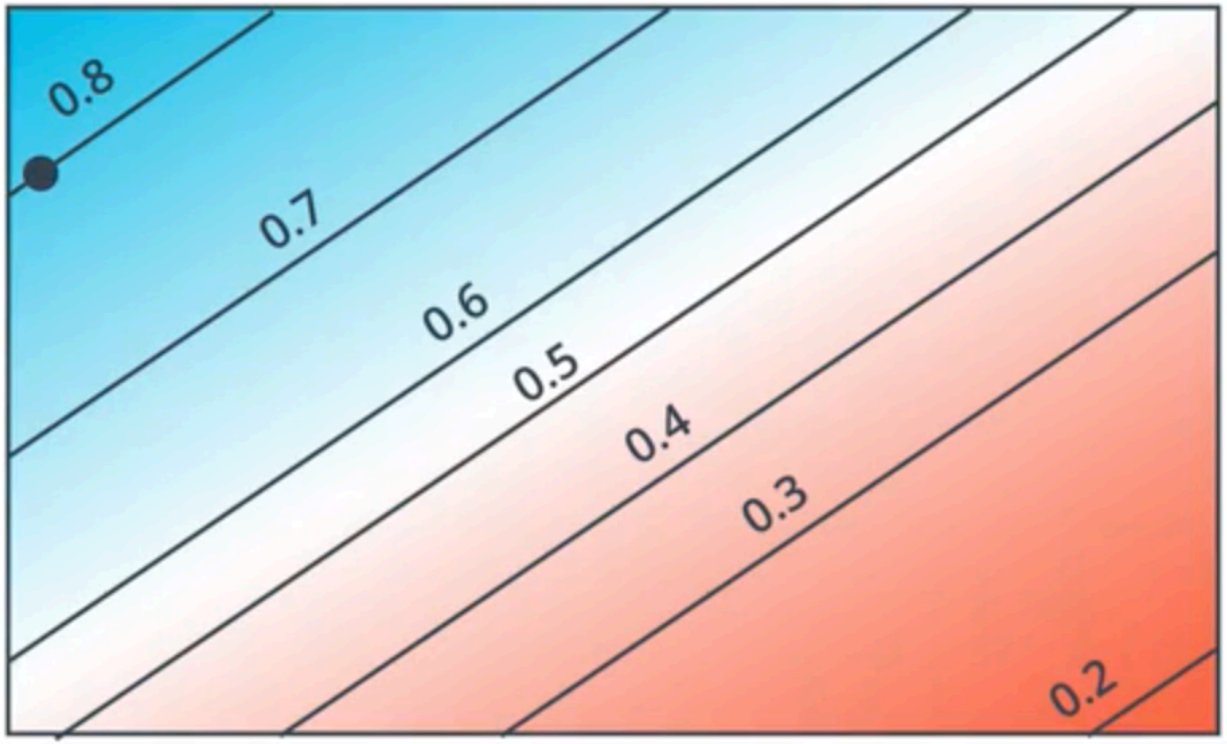
Predicões



P(BLUE) = 0.3

P(RED) = 0.7

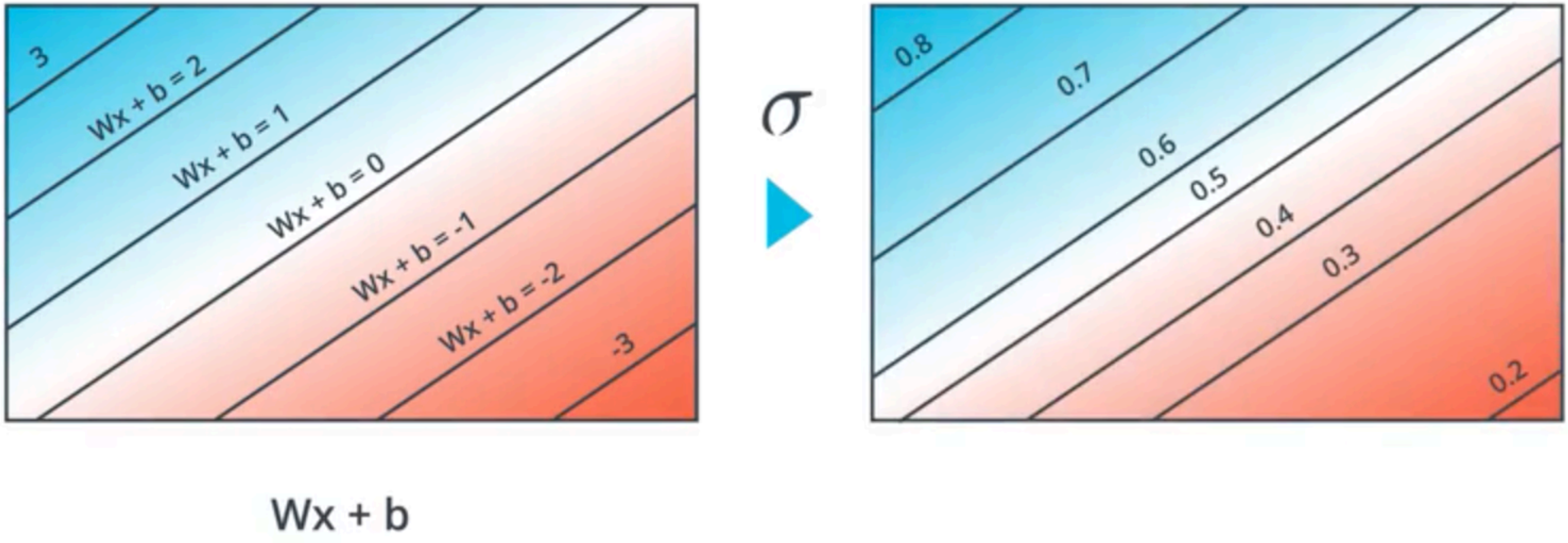
Predições



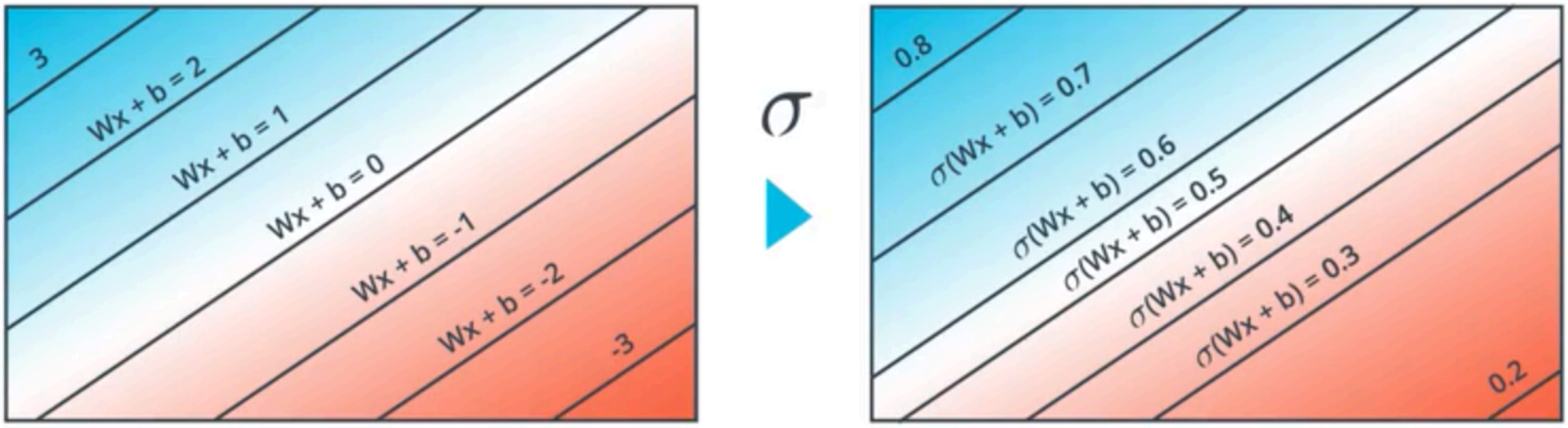
P(BLUE) = 0.8

P(RED) = 0.2

Predições

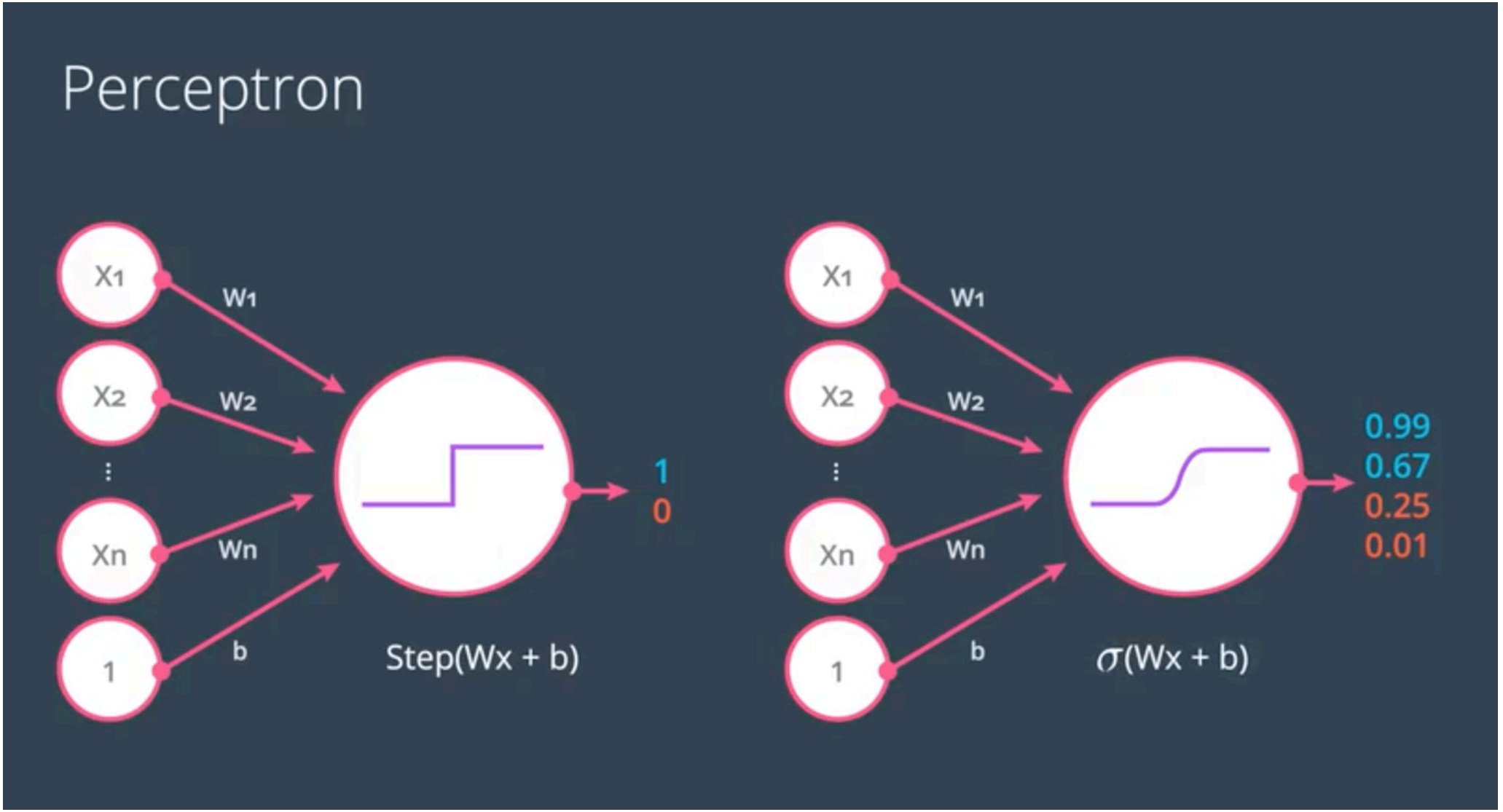


Predições

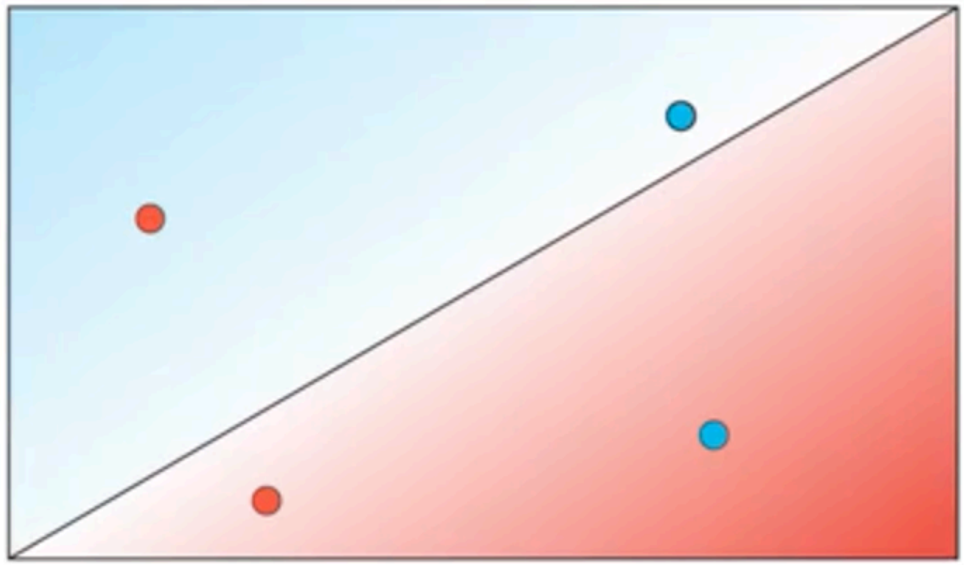


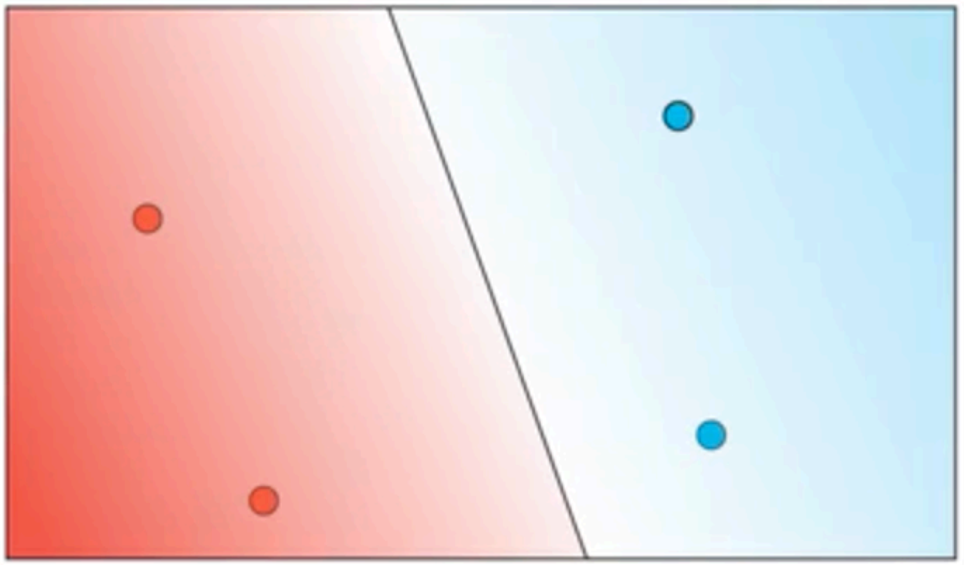
$Wx + b$

$\hat{y} = \sigma(Wx + b)$

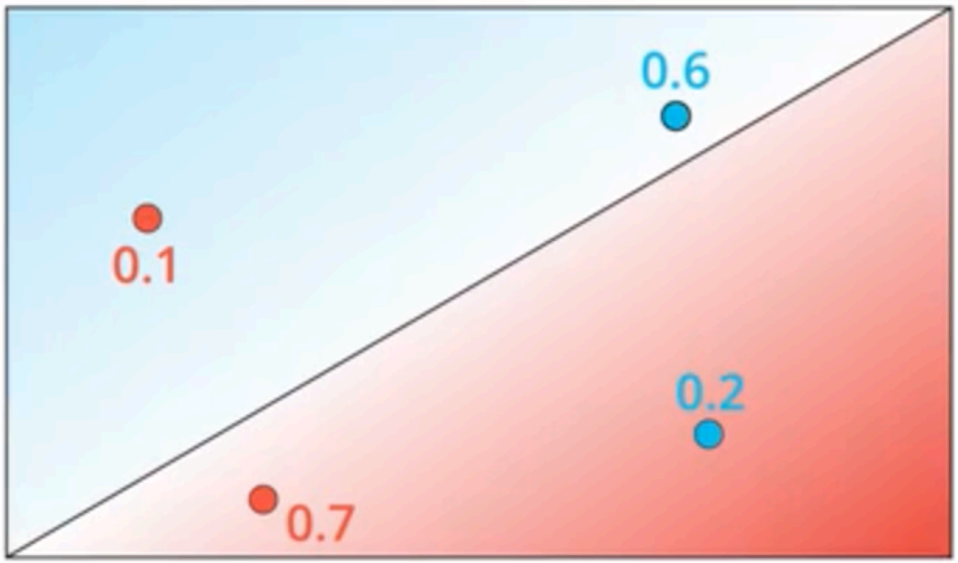

Perceptron
X1
X2
Xn
1
W1
W2
Wn
b
Step(Wx + b)
1
0
X1
X2
Xn
1
W1
W2
Wn
b
σ(Wx + b)
0.99
0.67
0.25
0.01

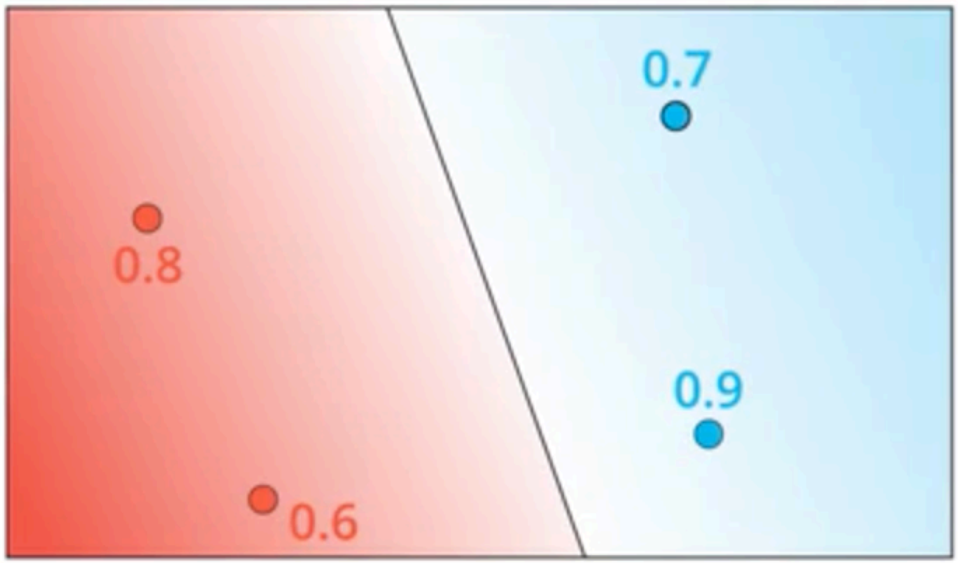
Qual desses modelos é melhor?



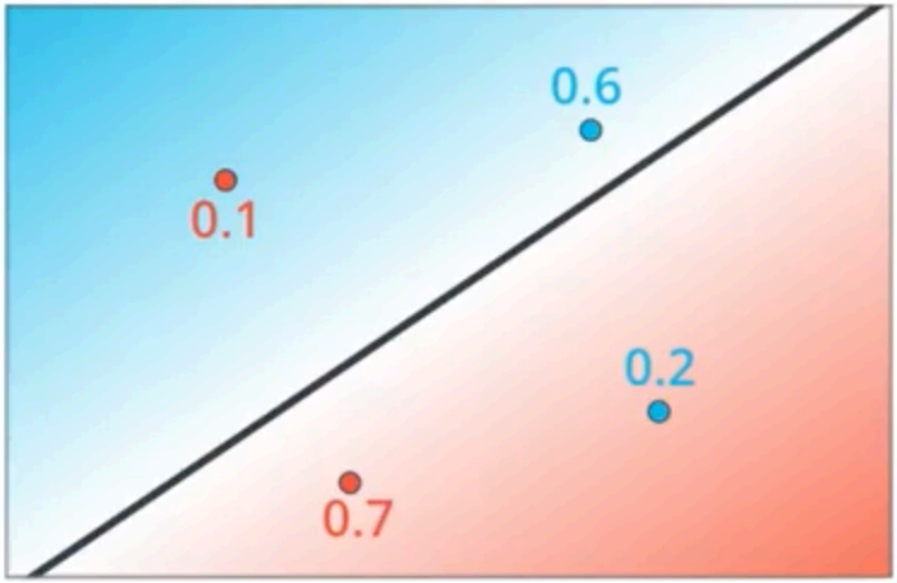


Qual desses modelos é melhor?

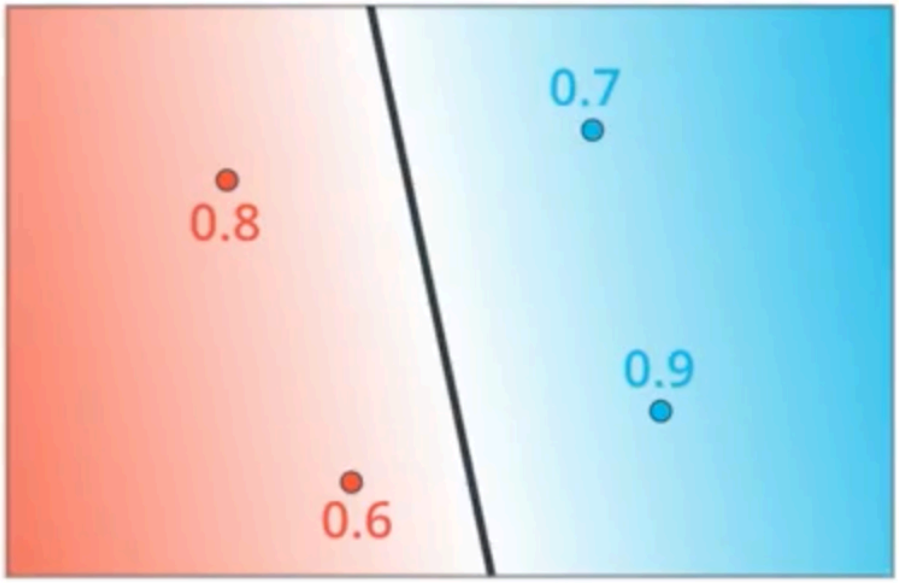




Qual desses modelos é melhor?

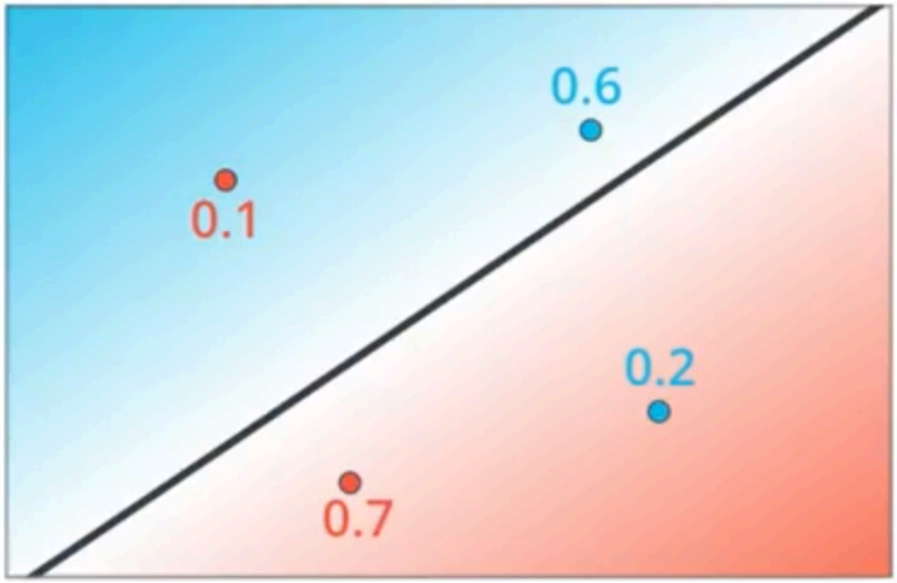


0.6*0.2*0.1*0.7 = 0.0084

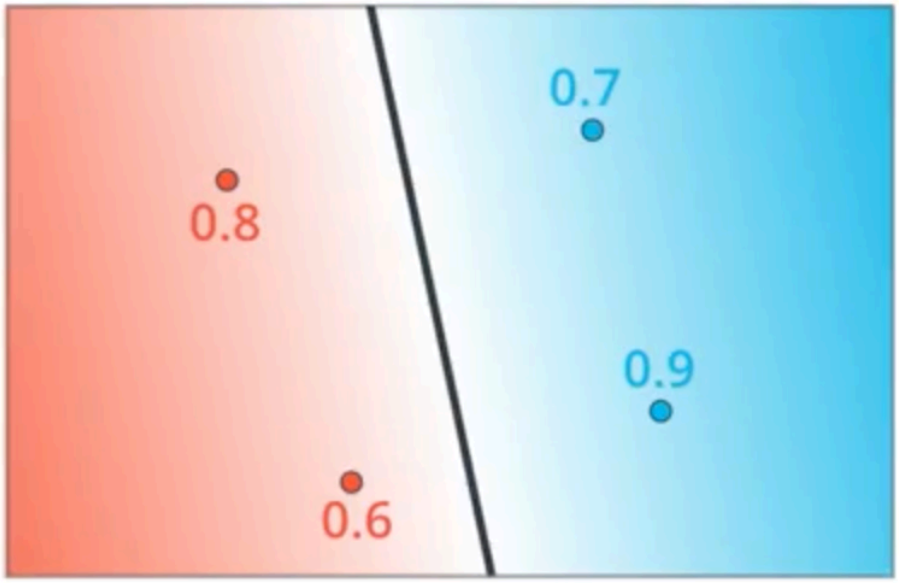


0.7*0.9*0.8*0.6 = 0.3024

Qual desses modelos é melhor?



0.6*0.2*0.1*0.7 = 0.0084



0.7*0.9*0.8*0.6 = 0.3024

Máxima Verossimilhança (Maximum Likelihood)

Problema

$0.6*0.2*0.1*0.7 = 0.0084$

$0.7*0.9*0.8*0.6 = 0.3024$

Qual Função usar?

- sin ○
- cos ○
- log ○
- exp ○

Problema

$0.6*0.2*0.1*0.7 = 0.0084$

$0.7*0.9*0.8*0.6 = 0.3024$

Qual Função usar?

- sin ○
- cos ○
- log ●
- exp ○

$\log(ab) = \log(a) + \log(b)$

Produtos

$0.6 * 0.2 * 0.1 * 0.7 = 0.0084$

$0.7 * 0.9 * 0.8 * 0.6 = 0.3024$

Produtos

$$0.6 * 0.2 * 0.1 * 0.7 = 0.0084$$

$$\ln(0.6) + \ln(0.2) + \ln(0.1) + \ln(0.7)$$

$$-0.51 \quad -1.61 \quad -2.3 \quad -0.36$$

$$0.7 * 0.9 * 0.8 * 0.6 = 0.3024$$

$$\ln(0.7) + \ln(0.9) + \ln(0.8) + \ln(0.6)$$

$$-0.36 \quad -0.1 \quad -.22 \quad -0.51$$

Produtos

$0.6 * 0.2 * 0.1 * 0.7 = 0.0084$

$\ln(0.6) + \ln(0.2) + \ln(0.1) + \ln(0.7)$

$-0.51 \quad -1.61 \quad -2.3 \quad -0.36$

$-\ln(0.6) - \ln(0.2) - \ln(0.1) - \ln(0.7) = 4.8$

$0.51 \quad 1.61 \quad 2.3 \quad 0.36$

$0.7 * 0.9 * 0.8 * 0.6 = 0.3024$

$\ln(0.7) + \ln(0.9) + \ln(0.8) + \ln(0.6)$

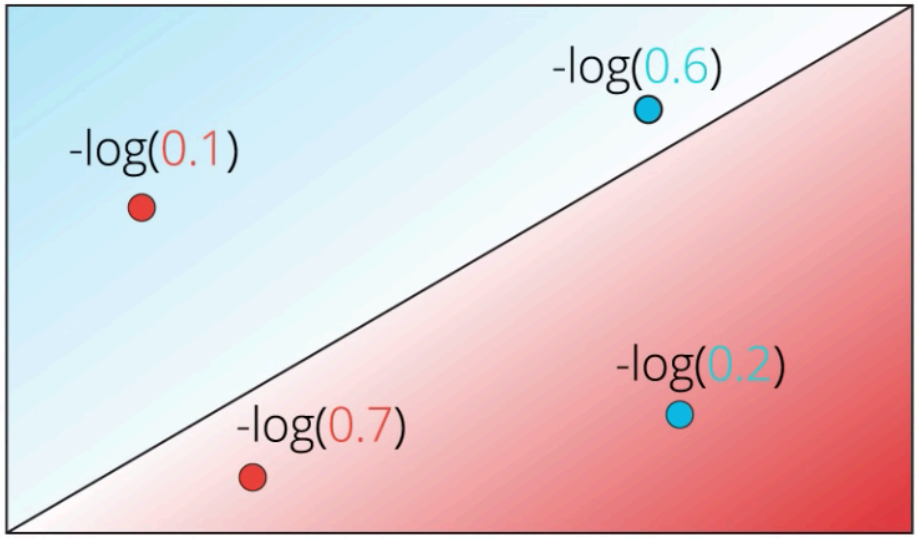
$-0.36 \quad -0.1 \quad -.22 \quad -0.51$

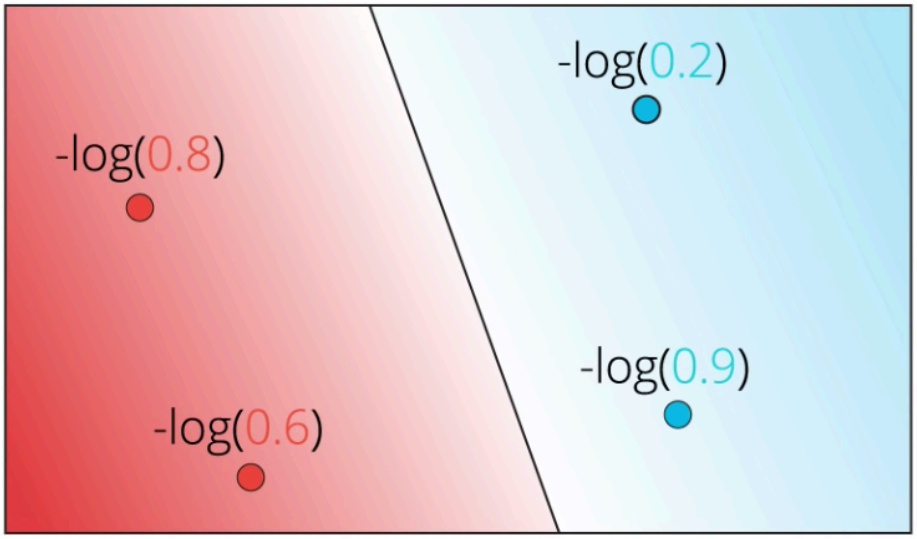
$-\ln(0.7) - \ln(0.9) - \ln(0.8) - \ln(0.6) = 1.2$

$0.36 \quad 0.1 \quad .22 \quad 0.51$

Cross Entropy

Cross Entropia





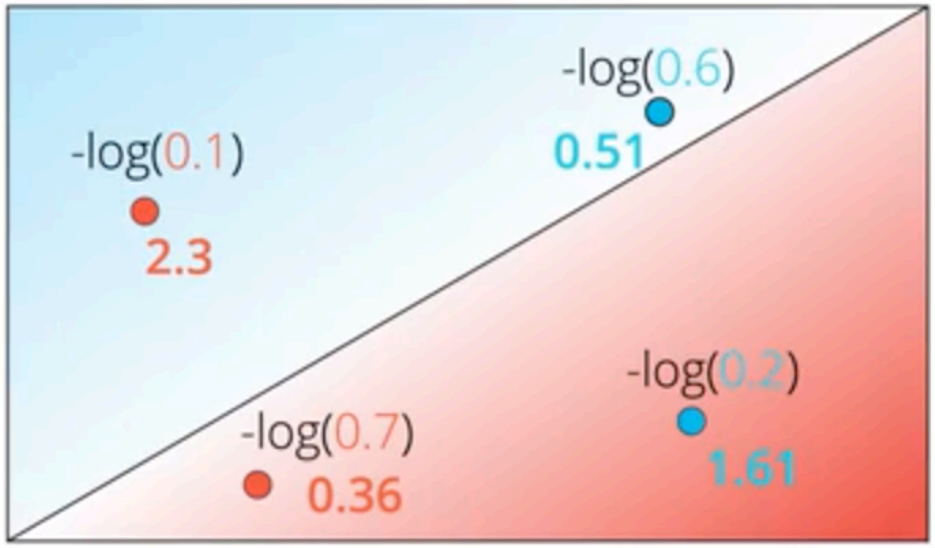
0.6 * 0.2 * 0.1 * 0.7 = 0.0084

-log(0.6) - log(0.2) - log(0.1) - log(0.7) = 4.8

0.7 * 0.9 * 0.8 * 0.6 = 0.3024

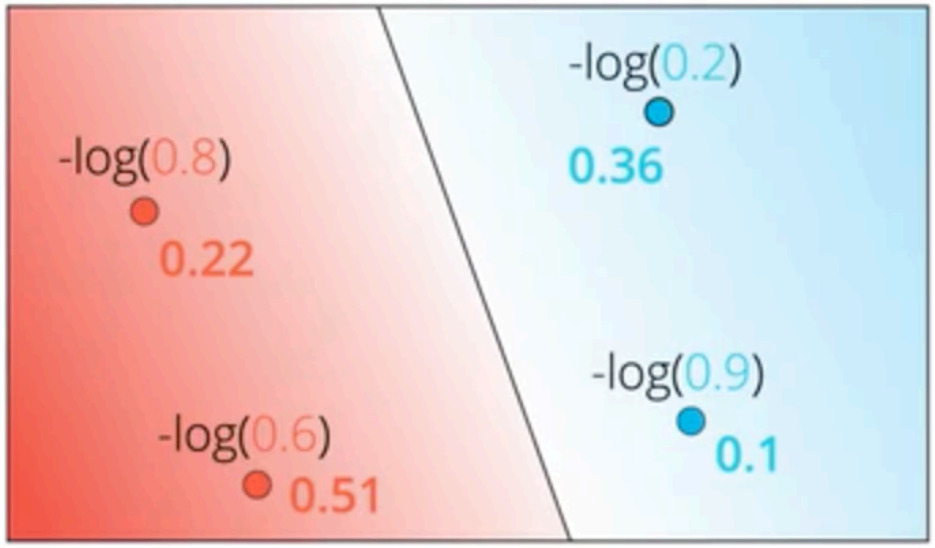
-log(0.7) - log(0.9) - log(0.8) - log(0.6) = 1.2

Cross Entropia



$0.6 * 0.2 * 0.1 * 0.7 = 0.0084$

$-\log(0.6) - \log(0.2) - \log(0.1) - \log(0.7) = 4.8$

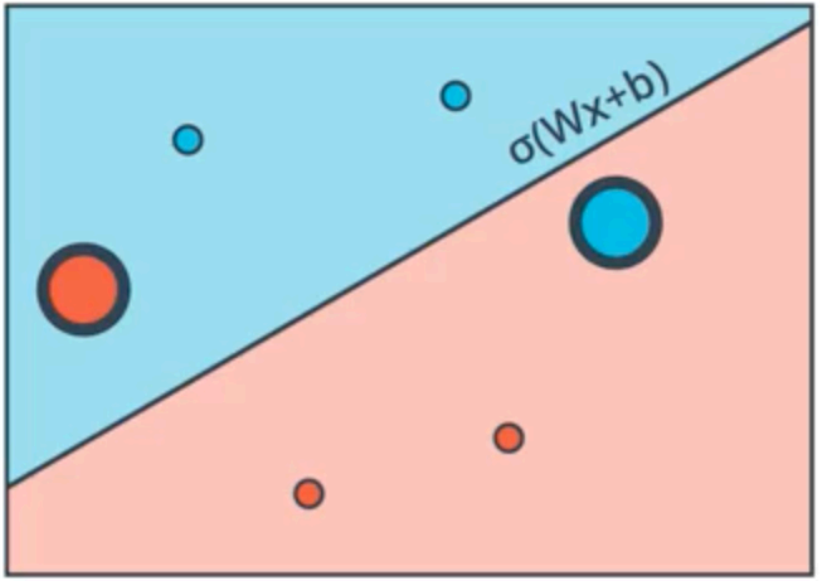


$0.7 * 0.9 * 0.8 * 0.6 = 0.3024$

$-\log(0.7) - \log(0.9) - \log(0.8) - \log(0.6) = 1.2$

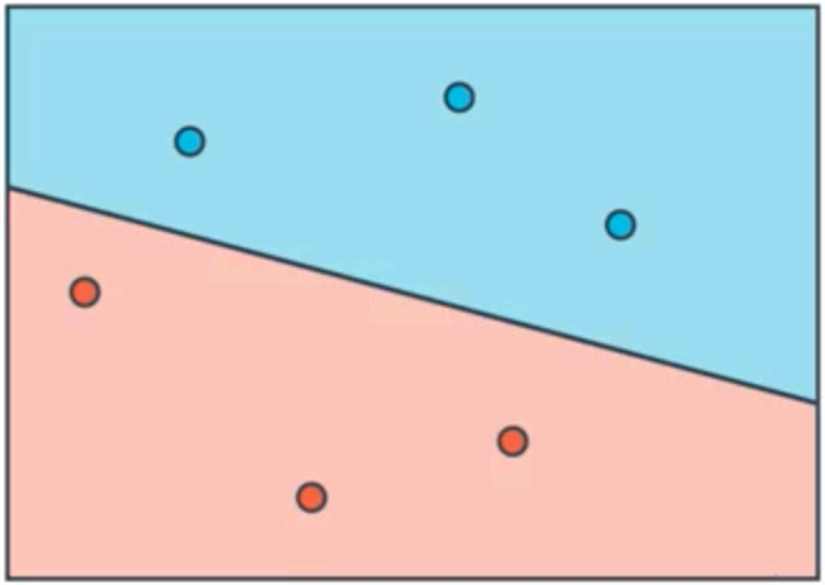
Objetivo: Minimizar a Cross Entropia

Objetivo: Minimizar Função de Erro



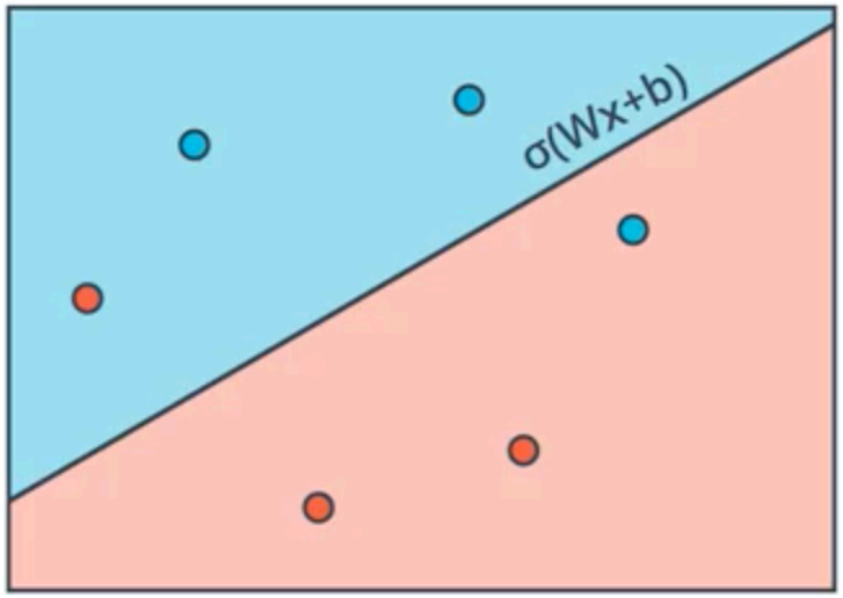
$$E(W,b) = -\frac{1}{m}\sum_{i=1}^{m} (1-y_i)\ln(1-\sigma(Wx^{(i)}+b)) + y_i \ln(\sigma(Wx^{(i)}+b))$$

Objetivo: Minimizar Função de Erro



$$E(W,b) = -\frac{1}{m}\sum_{i=1}^{m} (1-y_i)\ln(1-\sigma(Wx^{(i)}+b)) + y_i \ln(\sigma(Wx^{(i)}+b))$$

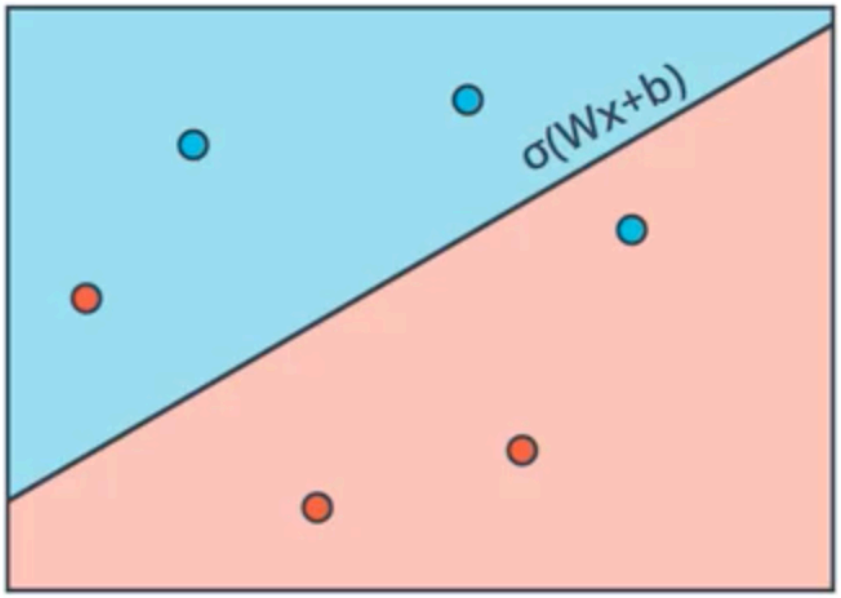
Algoritmo Gradiente Descendente



1. Comece com pesos aleatórios: :

   $w_1, \ldots, w_n, b$

Algoritmo Gradiente Descendente

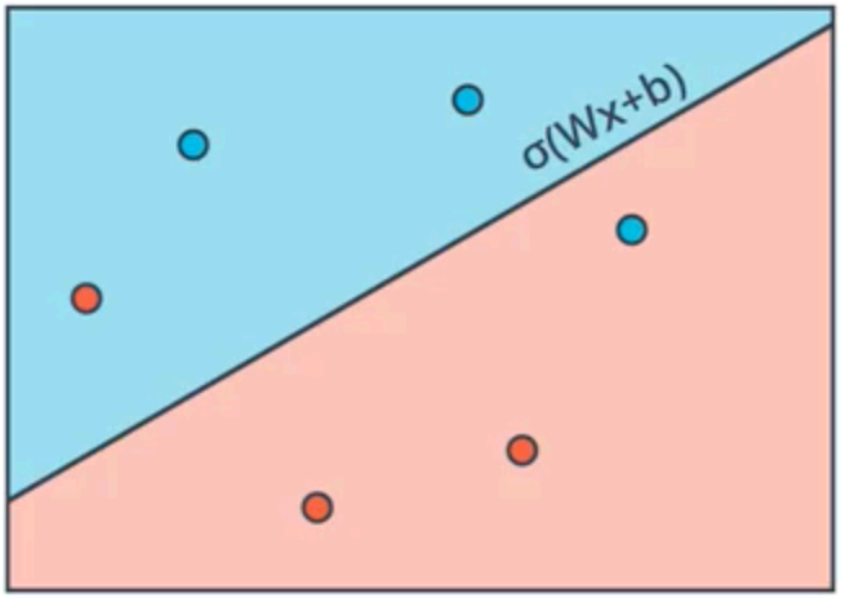


1. Comece com pesos aleatórios: :

   $w_1, \ldots, w_n, b$

2. Para cada ponto $(x_1, \ldots, x_n)$:
   2.1 Para i=1...n
      2.1.1 Atualiza $w_i' \leftarrow w_i - \alpha \, \partial E / \partial w_i$
      2.1.2 Atualiza b'$\leftarrow b - \alpha \, \partial E / \partial b$

Algoritmo Gradiente Descendente

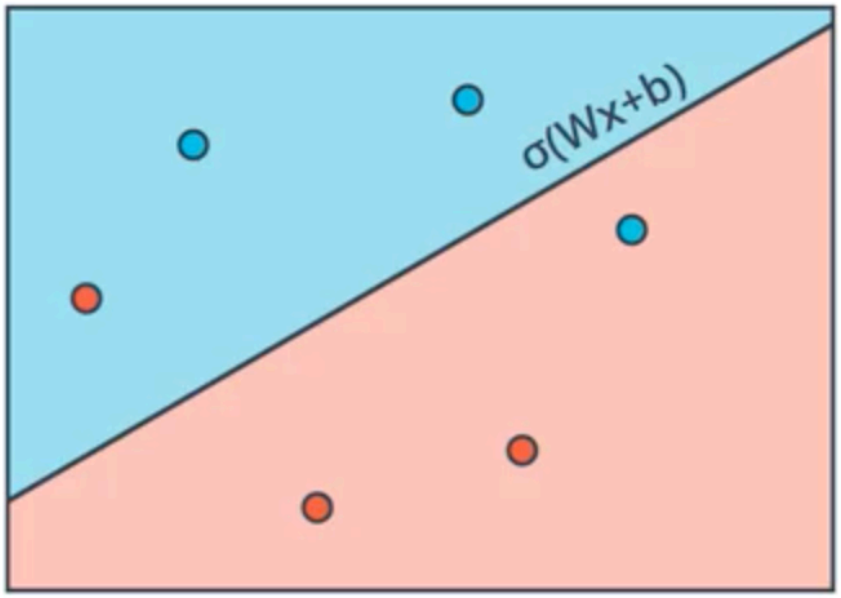


1. Comece com pesos aleatórios: :

   $w_1, \ldots, w_n, b$

2. Para cada ponto $(x_1, \ldots, x_n)$:
   2.1 Para i=1...n
      2.1.1 Atualiza $w_i' \leftarrow w_i - \alpha\ (\hat{y} - y)x_i$
      2.1.2 Atualiza b'$\leftarrow b - \alpha(\hat{y} - y)$

Algoritmo Gradiente Descendente



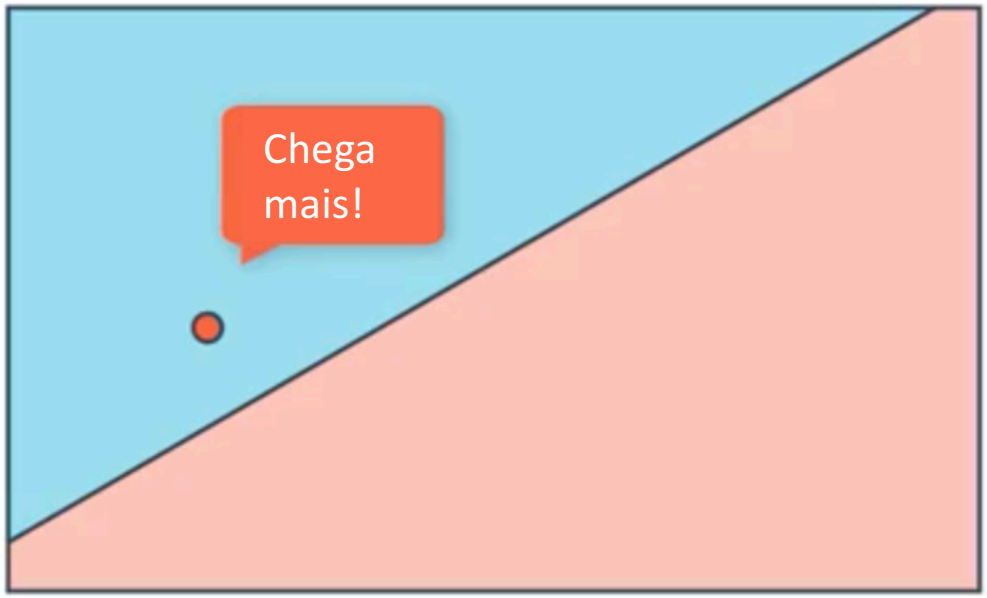
1. Comece com pesos aleatórios: :

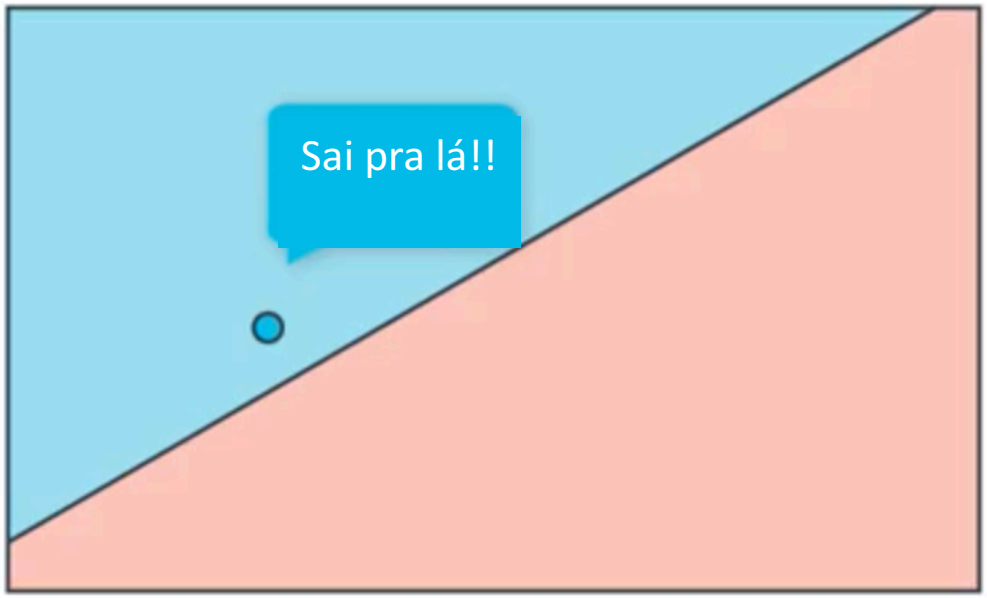
   $w_1, \ldots, w_n, b$

2. Para cada ponto $(x_1, \ldots, x_n)$:
   2.1 Para i=1...n
      2.1.1 Atualiza $w_i' \leftarrow w_i - \alpha\ (\hat{y} - y)x_i$
      2.1.2 Atualiza b'$\leftarrow b - \alpha(\hat{y} - y)$

3. Repetir até erro ser pequeno

Perceptron vs Gradiente Descendente



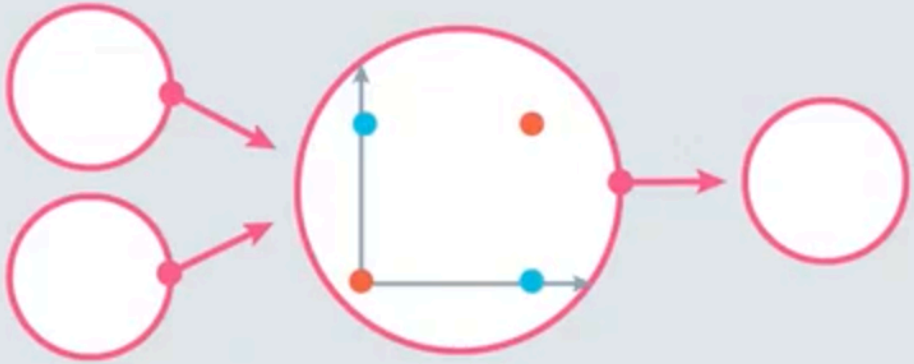


Exercício

- Implementar algoritmo de Gradiente Descendente.
  - ex_gradient_descent.ipynb

XOR Perceptron

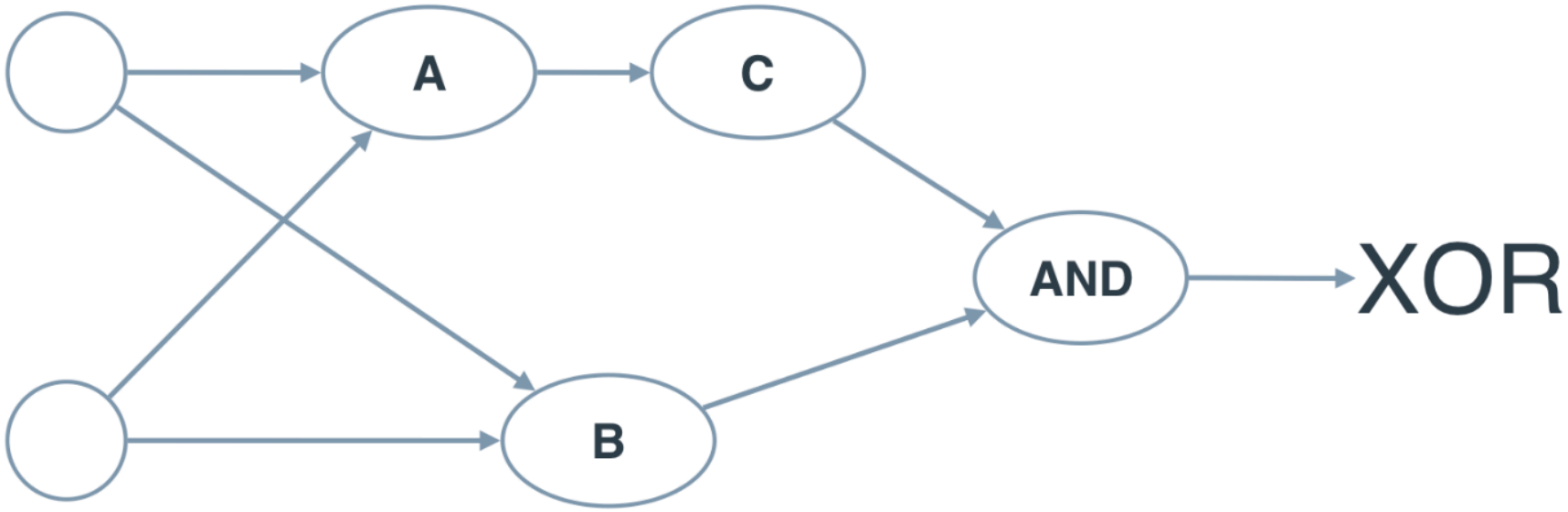
IN	IN	OUT
✓	✓	✗
✓	✗	✓
✗	✓	✓
✗	✗	✗



IN	IN	OUT
1	1	0
1	0	1
0	1	1
0	0	0

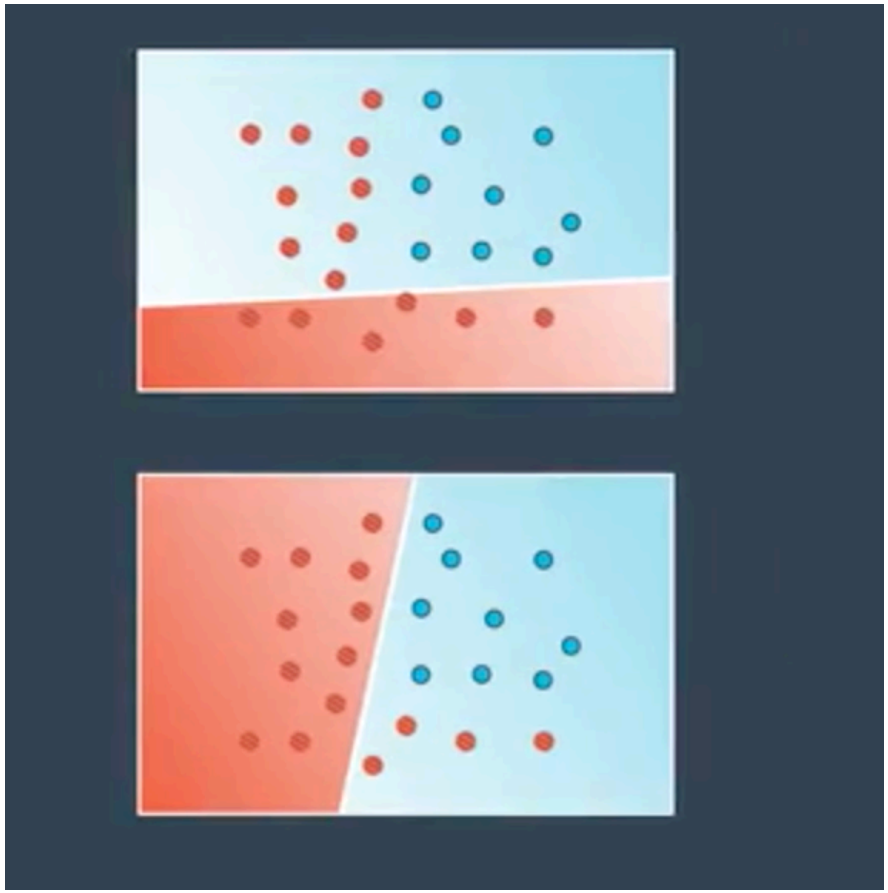
Quiz – Construir um XOR MultiLayer Perceptron

- A rede neural abaixo contêm 3 perceptrons, A, B, e C. O último (AND) foi fornecido. A entrada da rede neural é através do primeiro nó. A saída é pelo último nó.
- O perceptron multi-camadas abaixo calcula a operação XOR. Cada perceptron é um operação lógica de AND, OR e NOT. Diga os operações de cada perceptron para calcular o XOR.

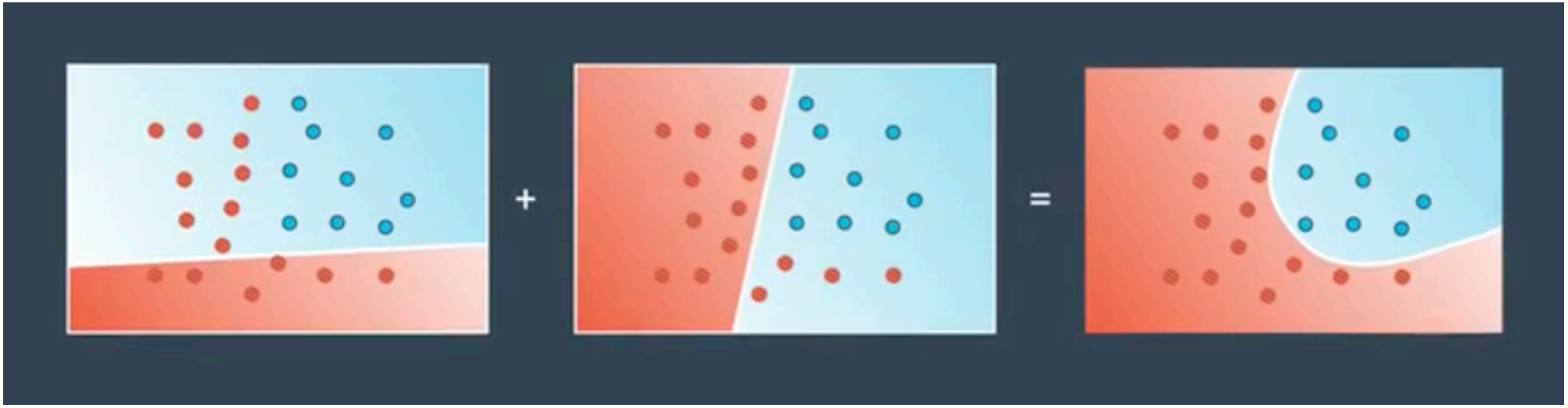


Regiões Não Lineares

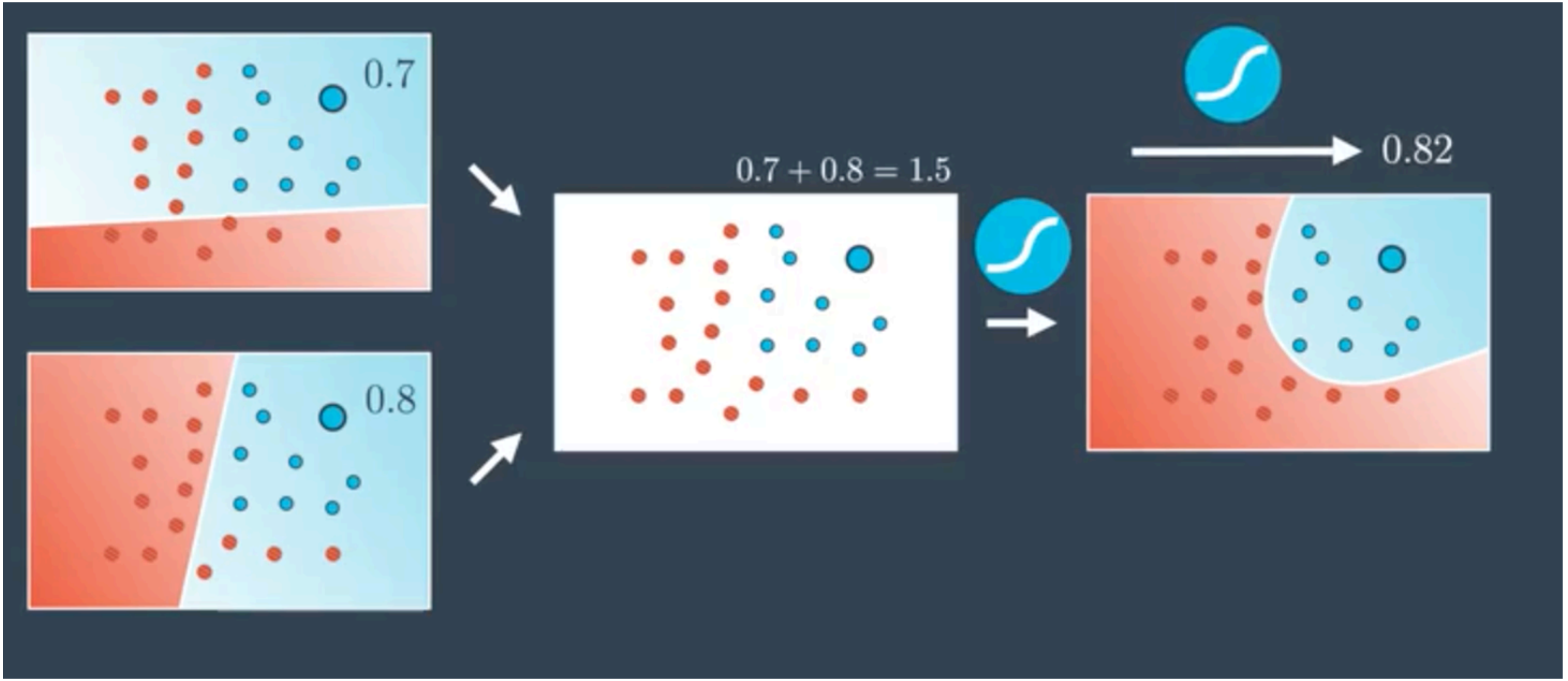
Combinando Regiões



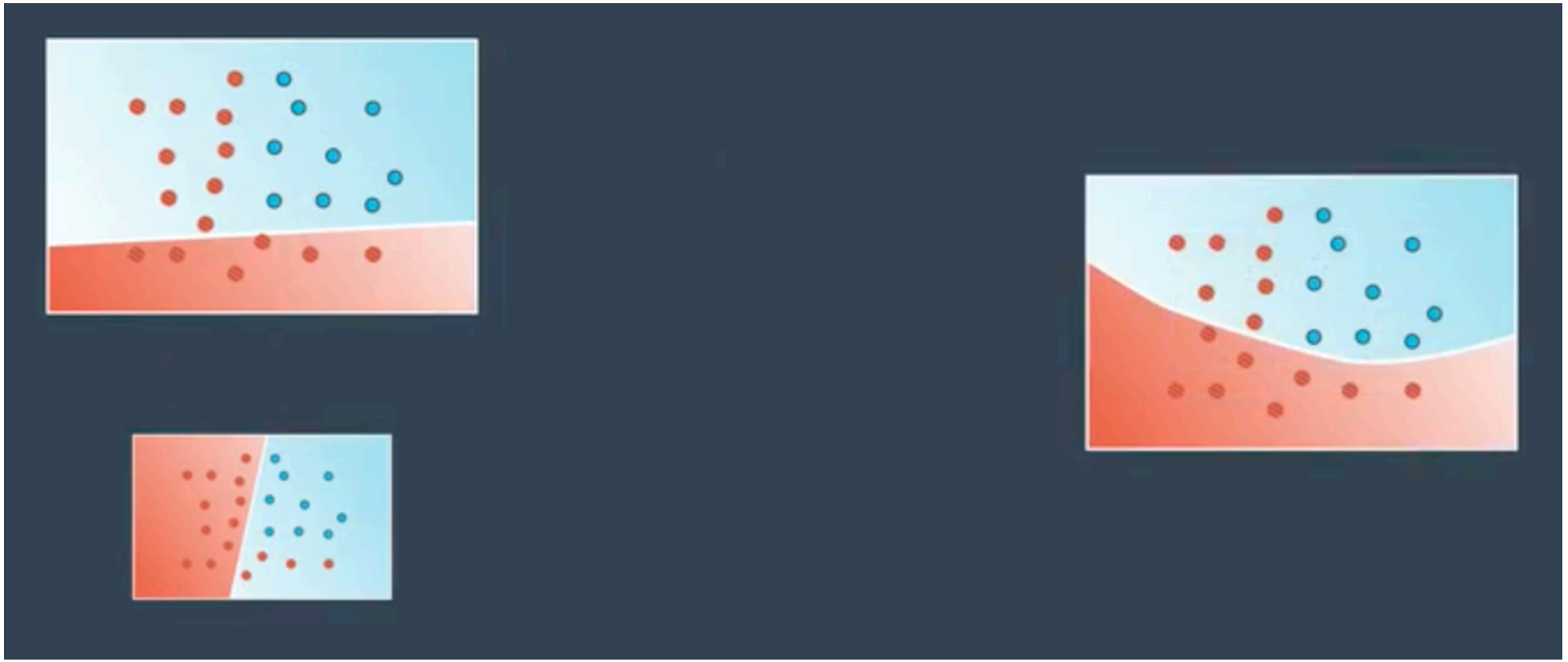
Combinando Regiões



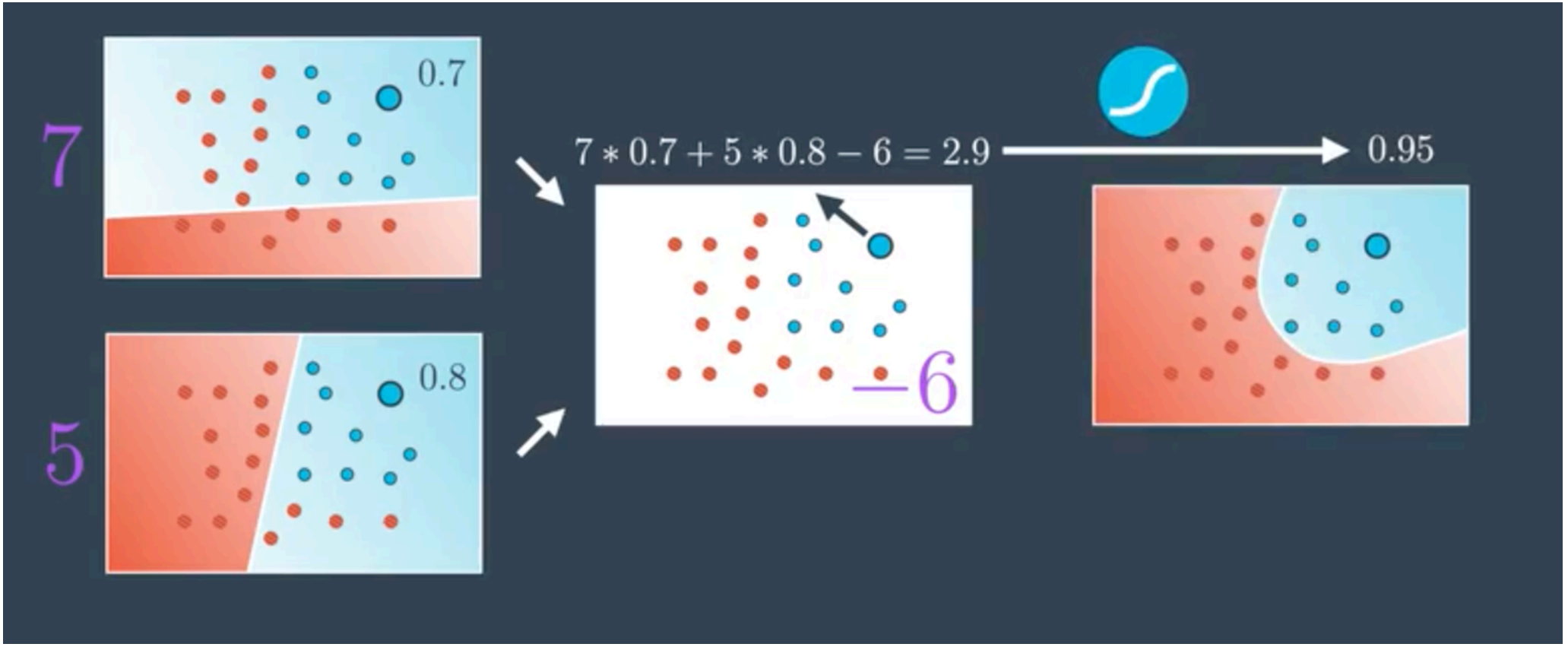
Combinando Regiões



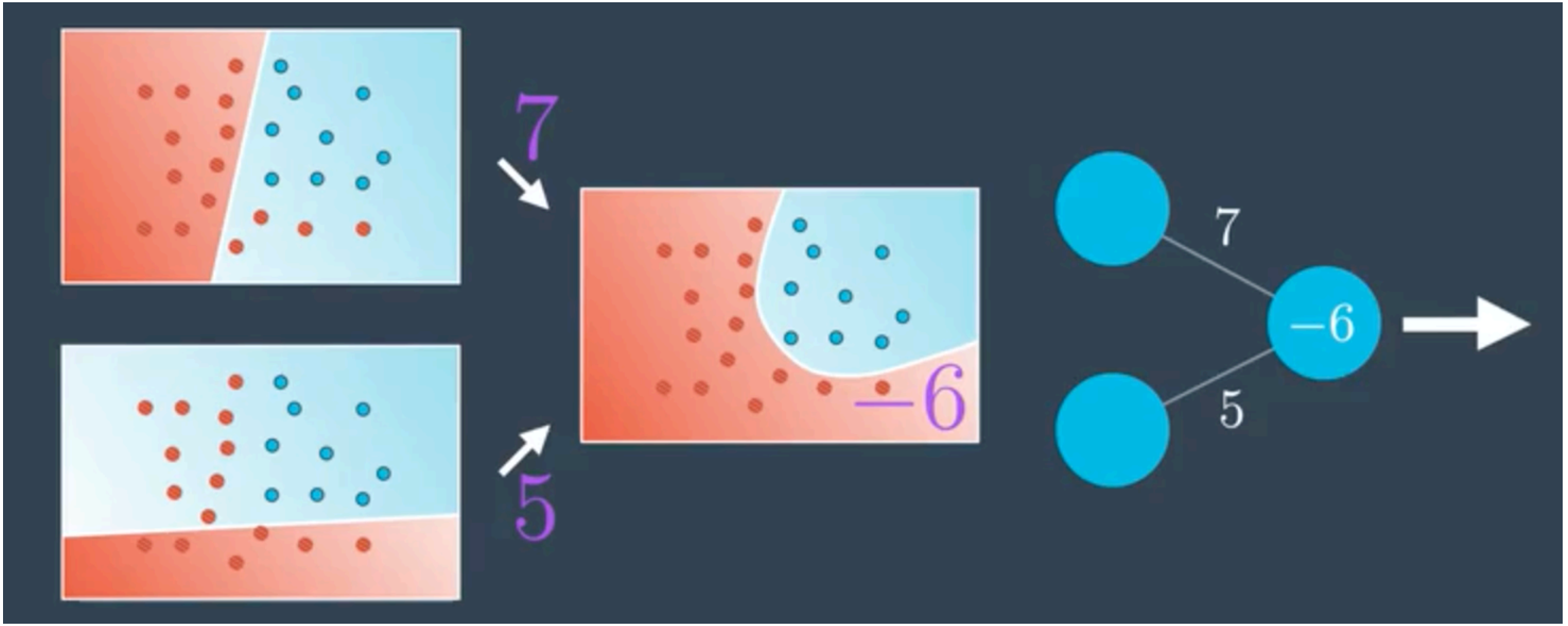
Combinando Regiões



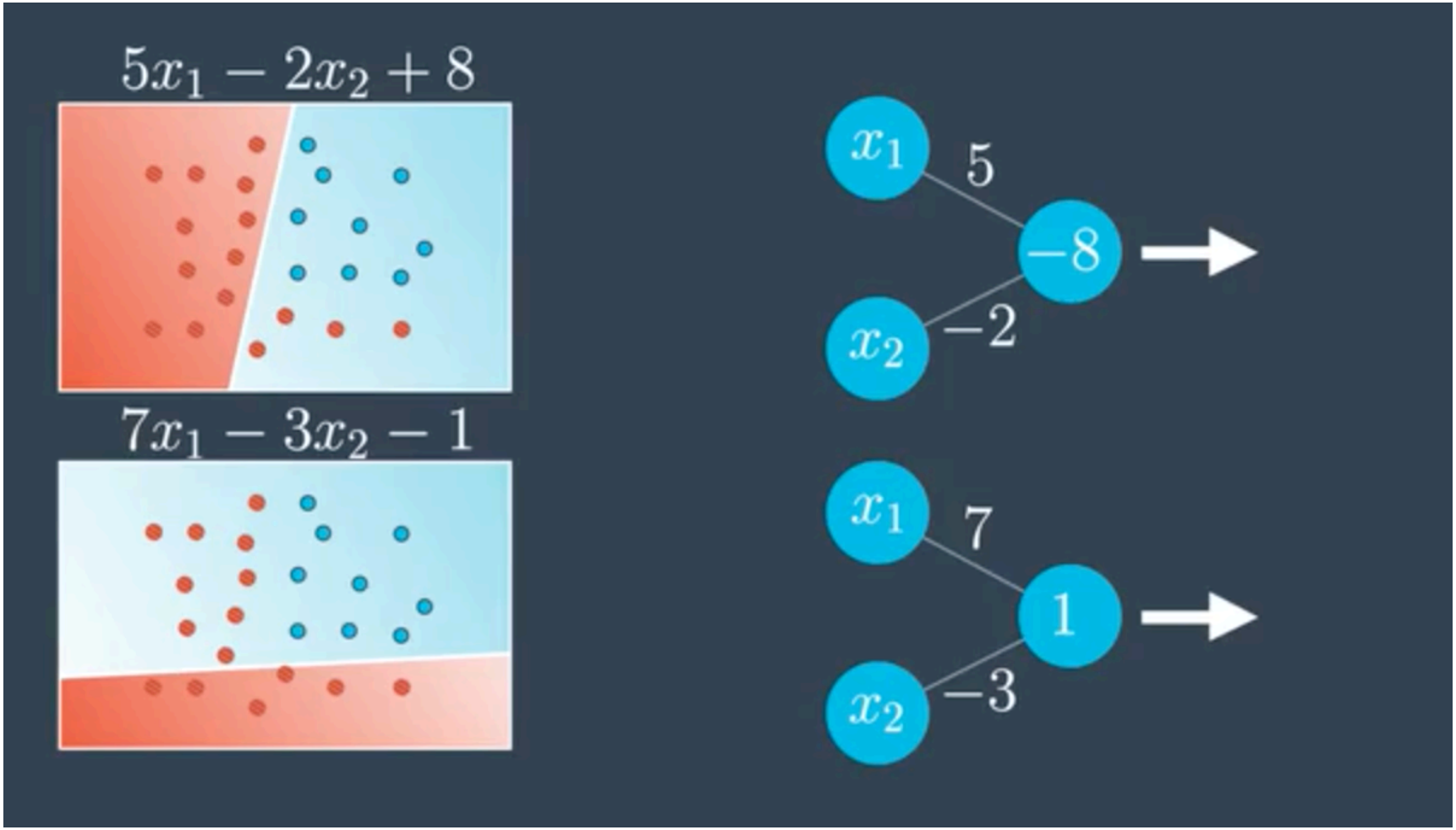
Combinando Regiões



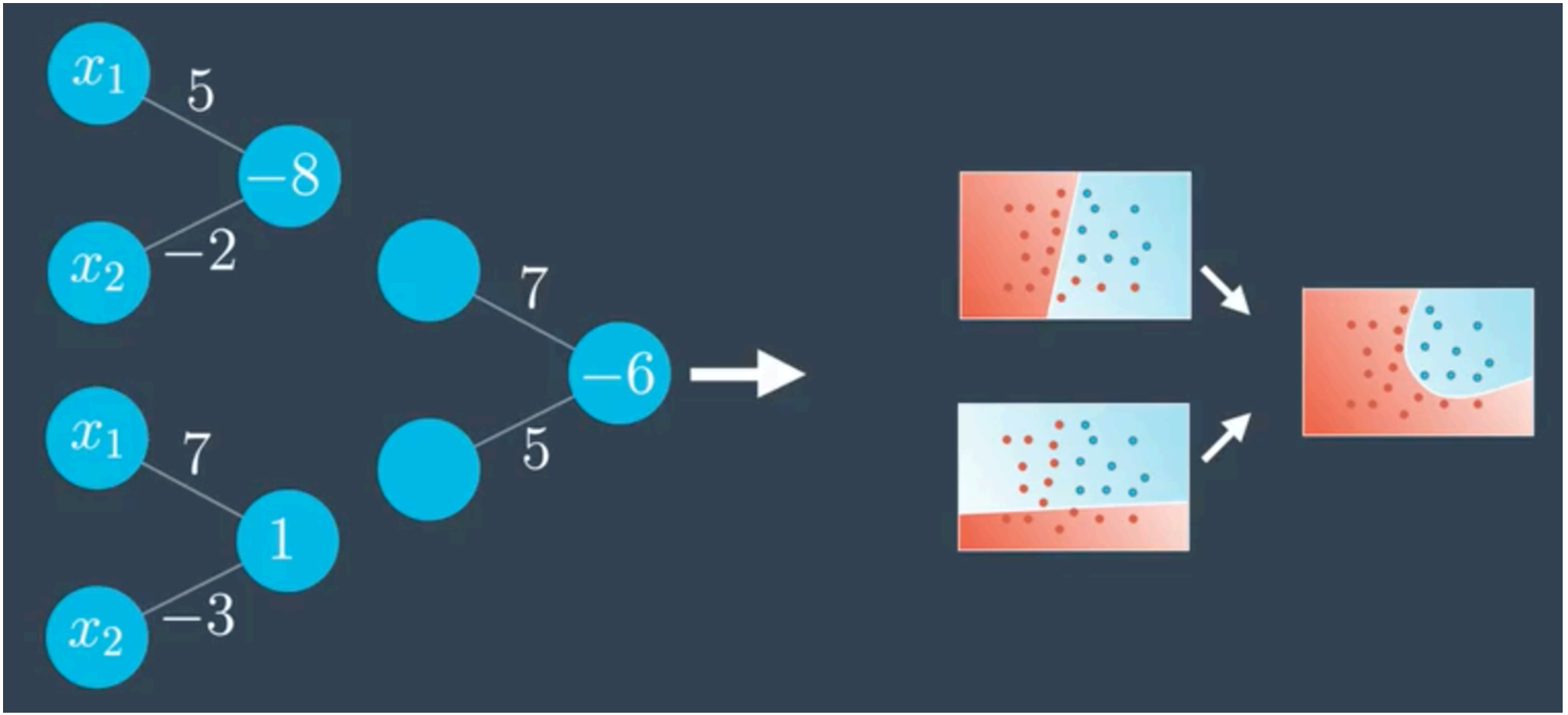
Combinando Regiões



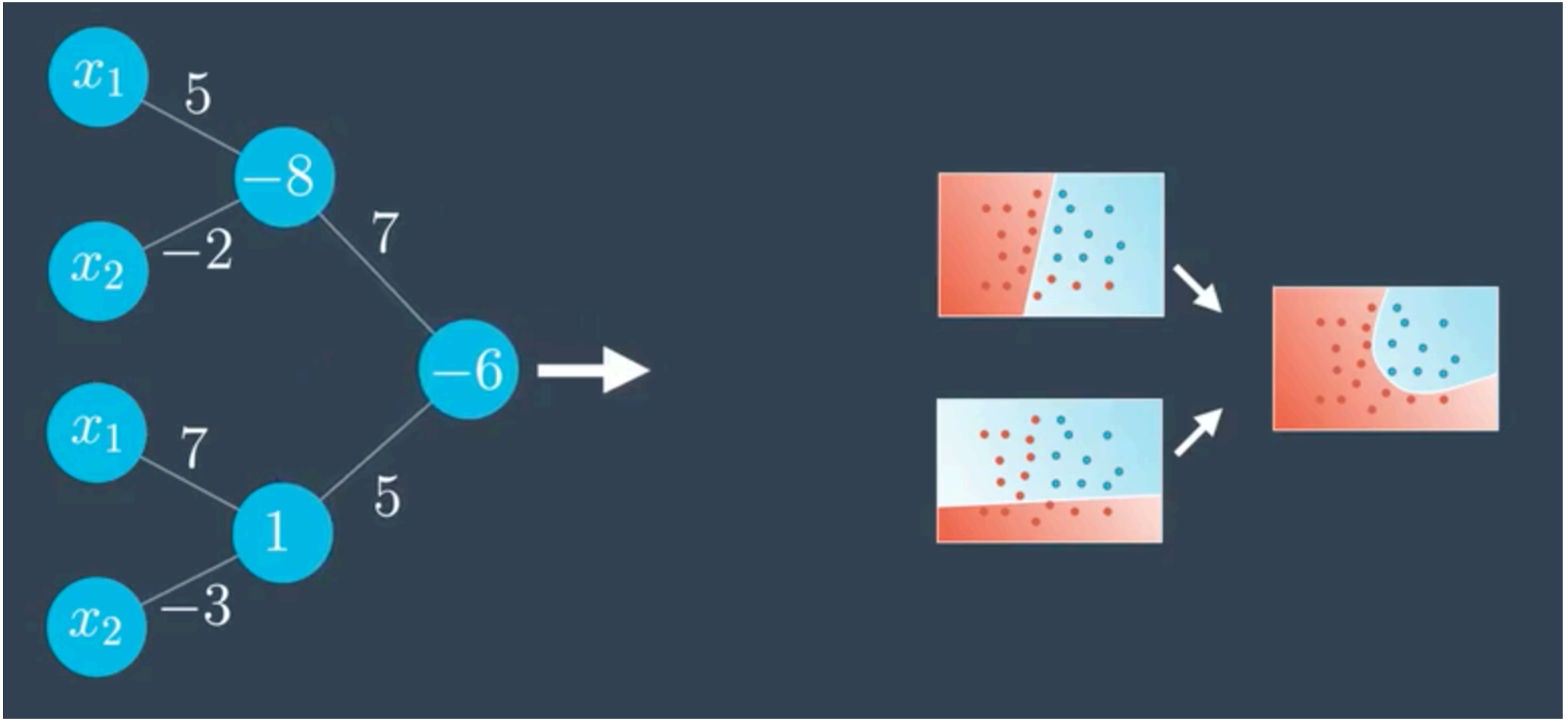
Combinando Regiões



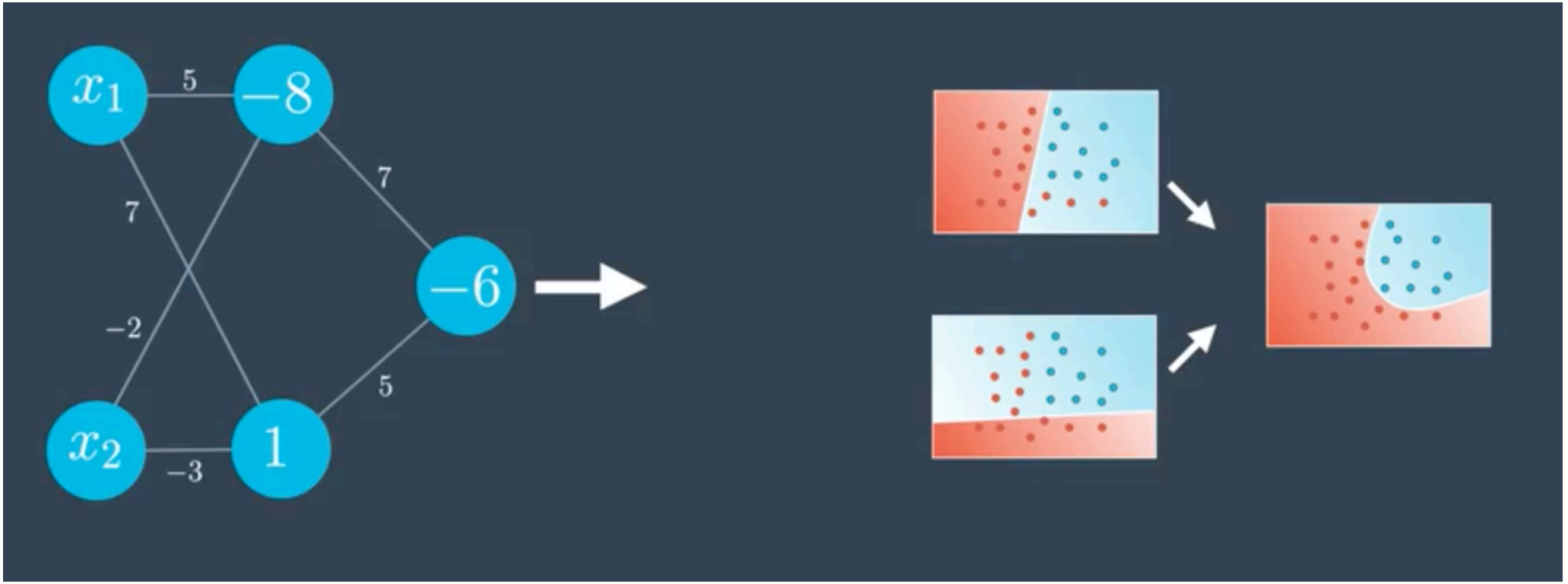
Combinando Regiões



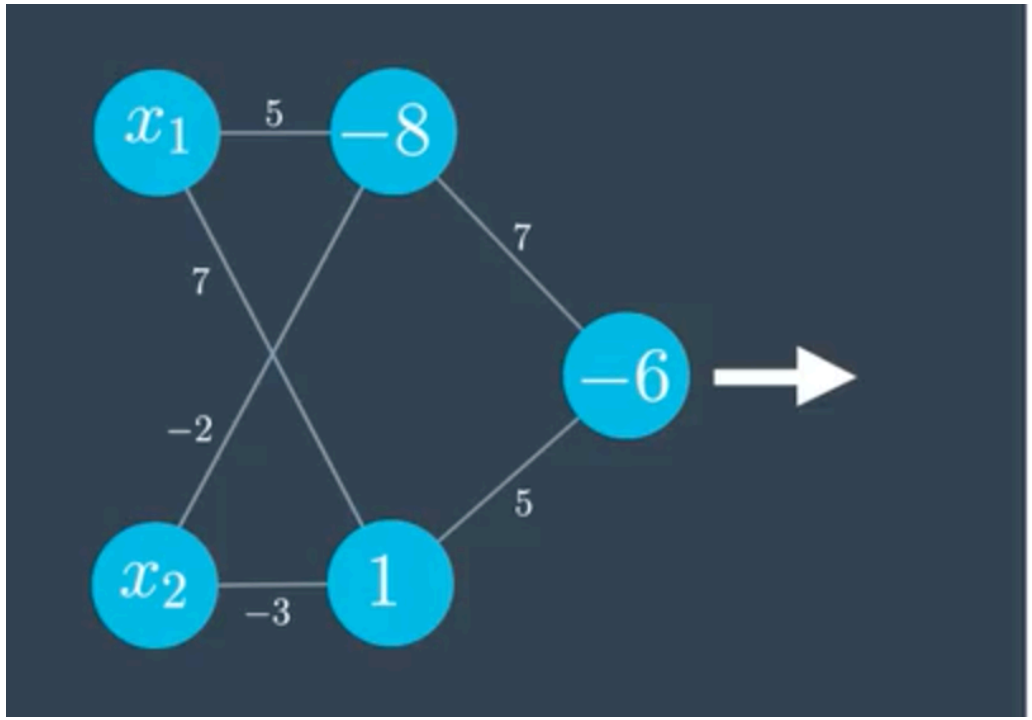
Combinando Regiões

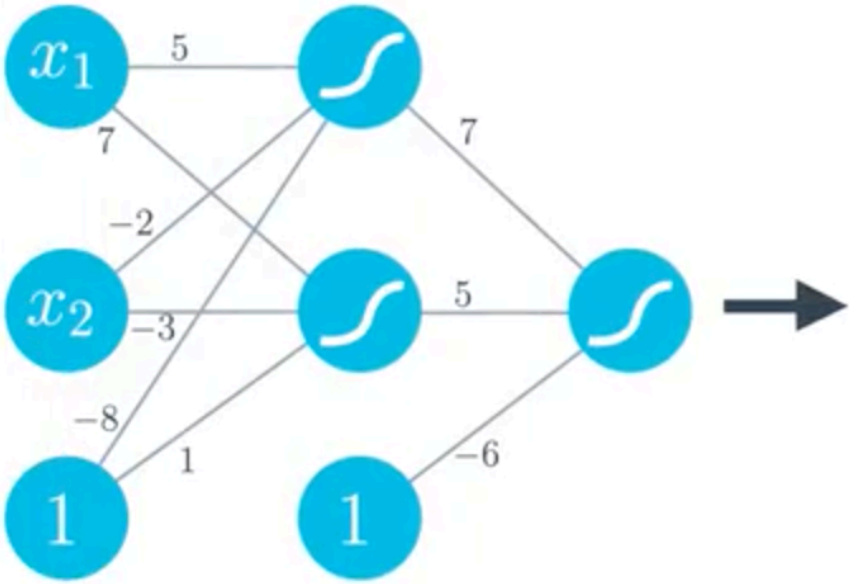


Combinando Regiões

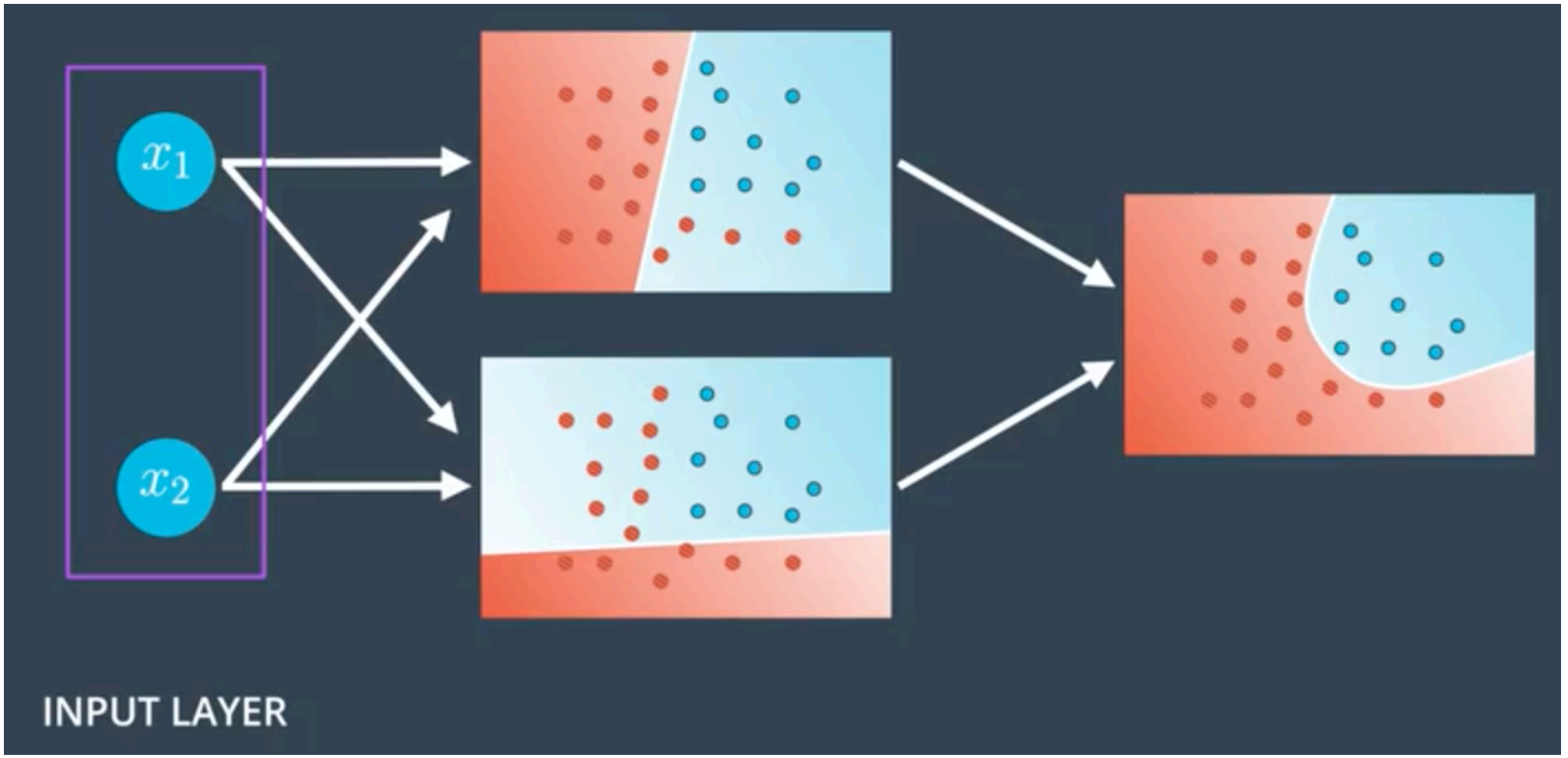


Combinando Regiões

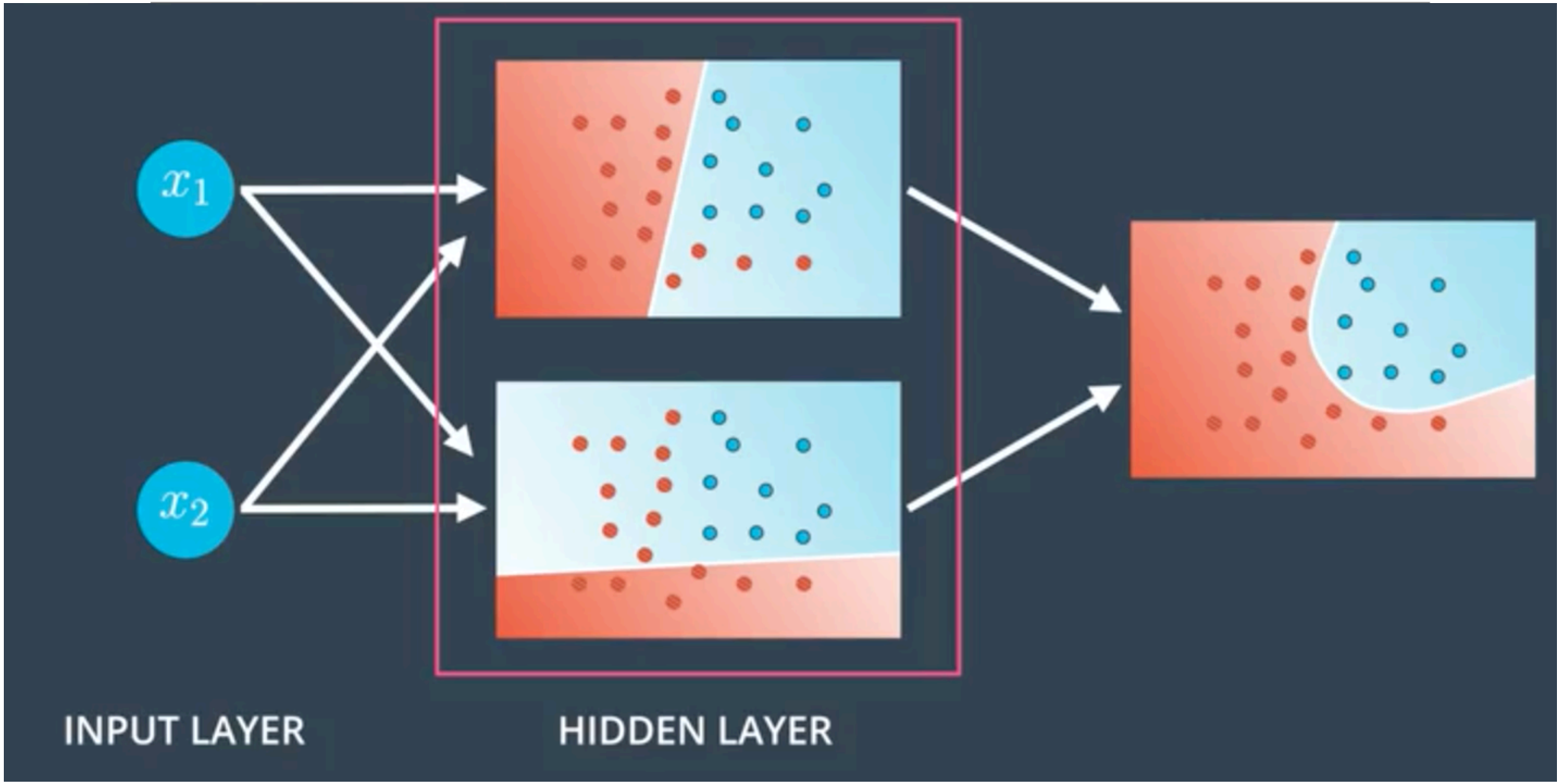




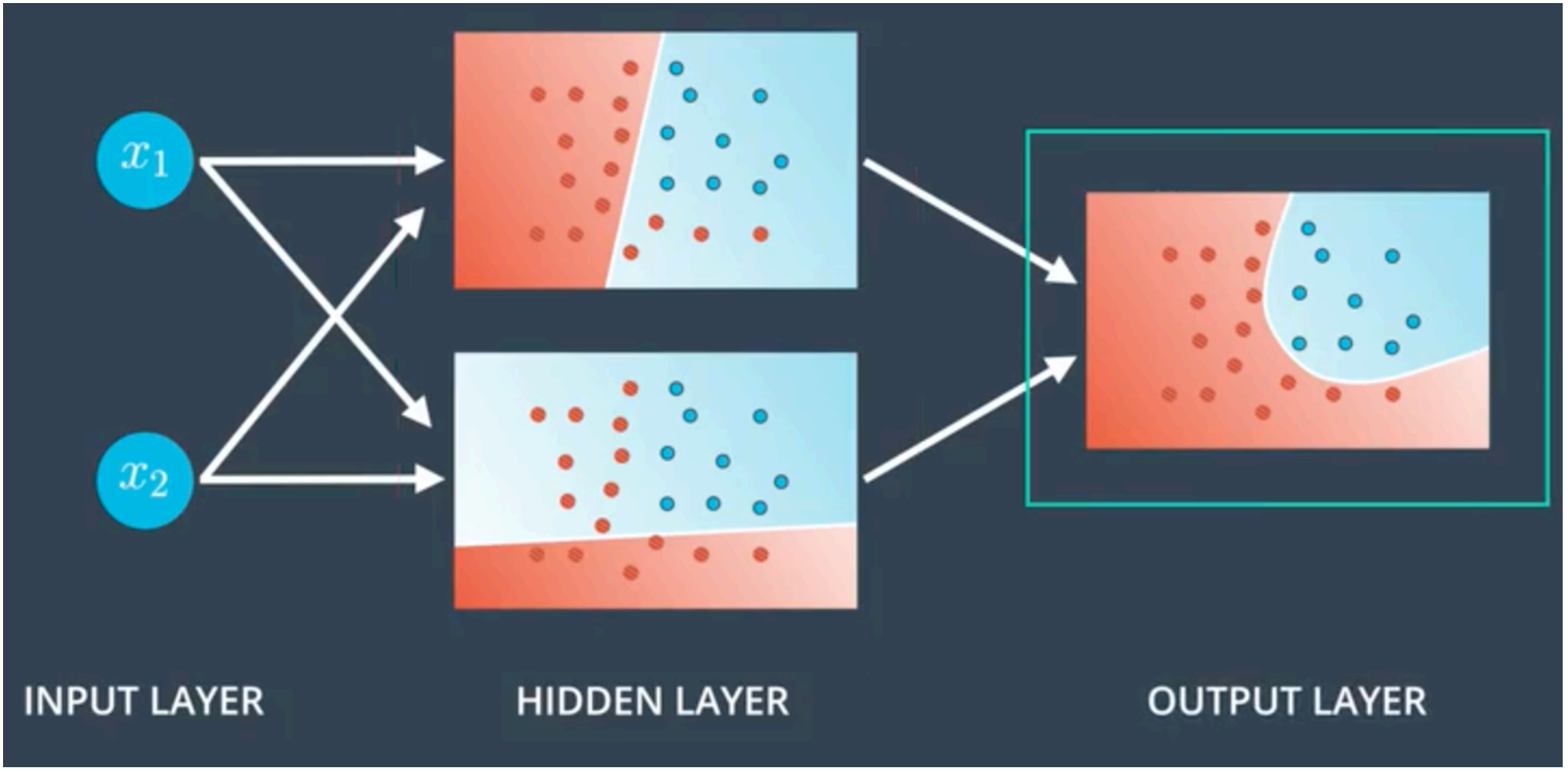
Multilayer Perceptron (MLP)

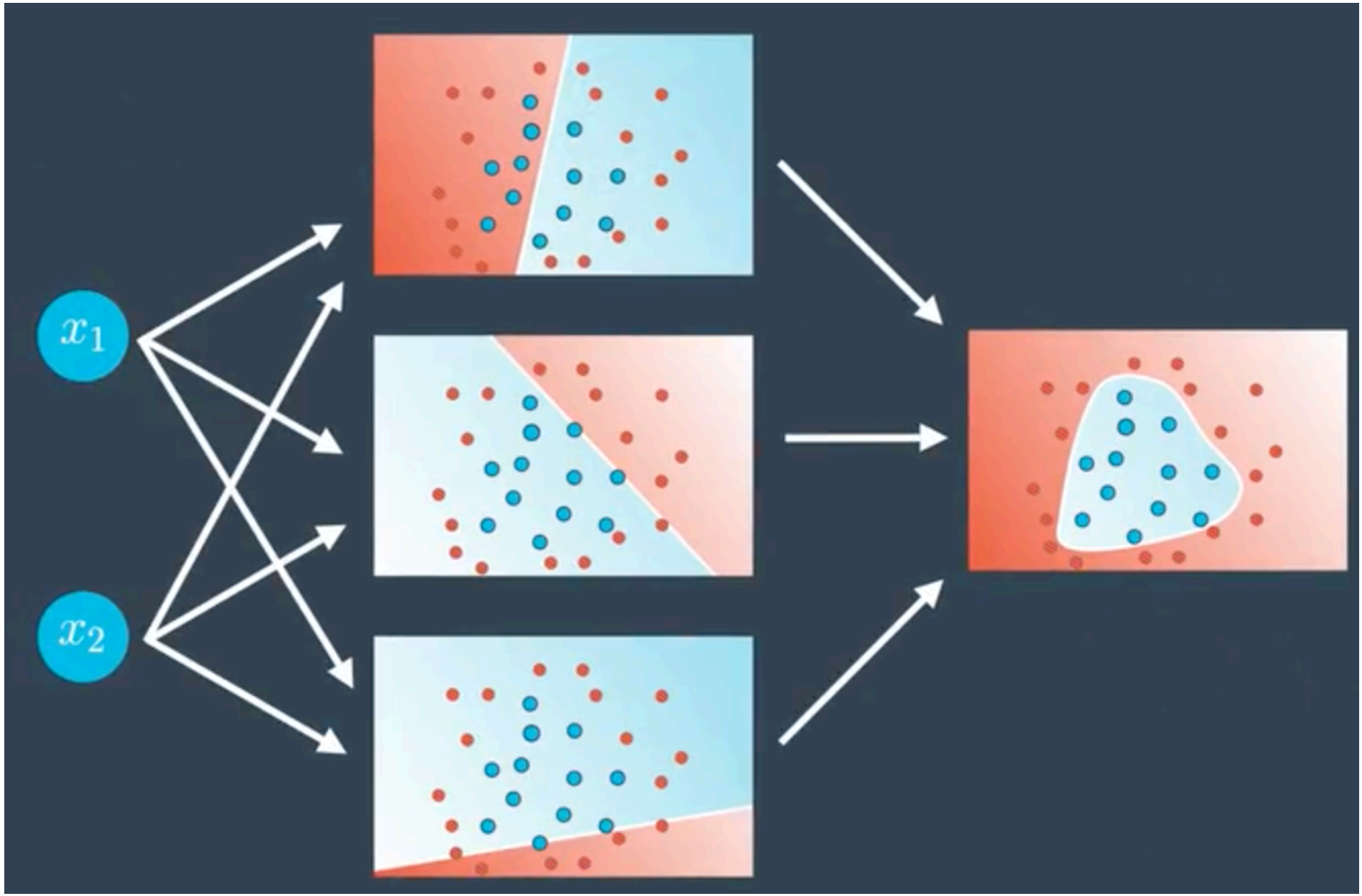


Multilayer Perceptron (MLP)

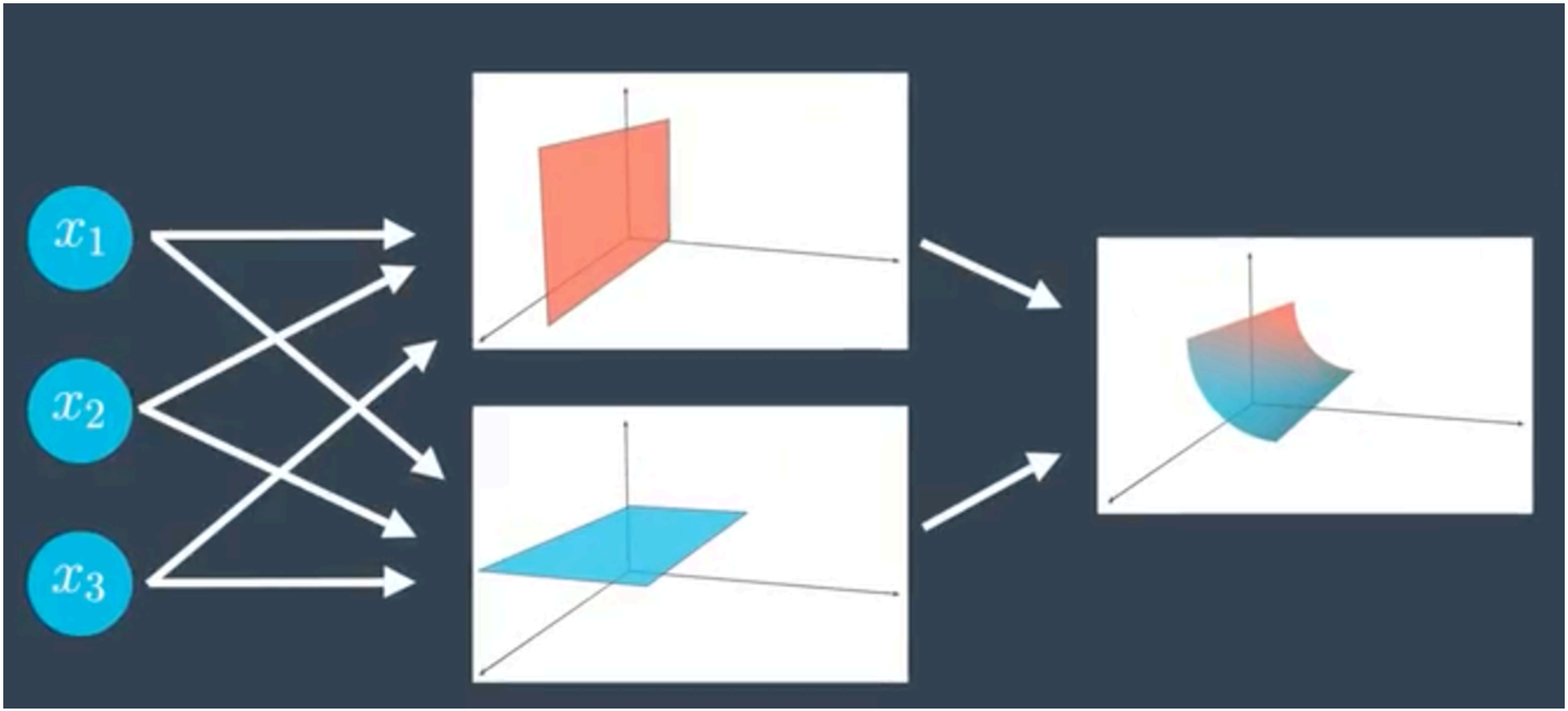


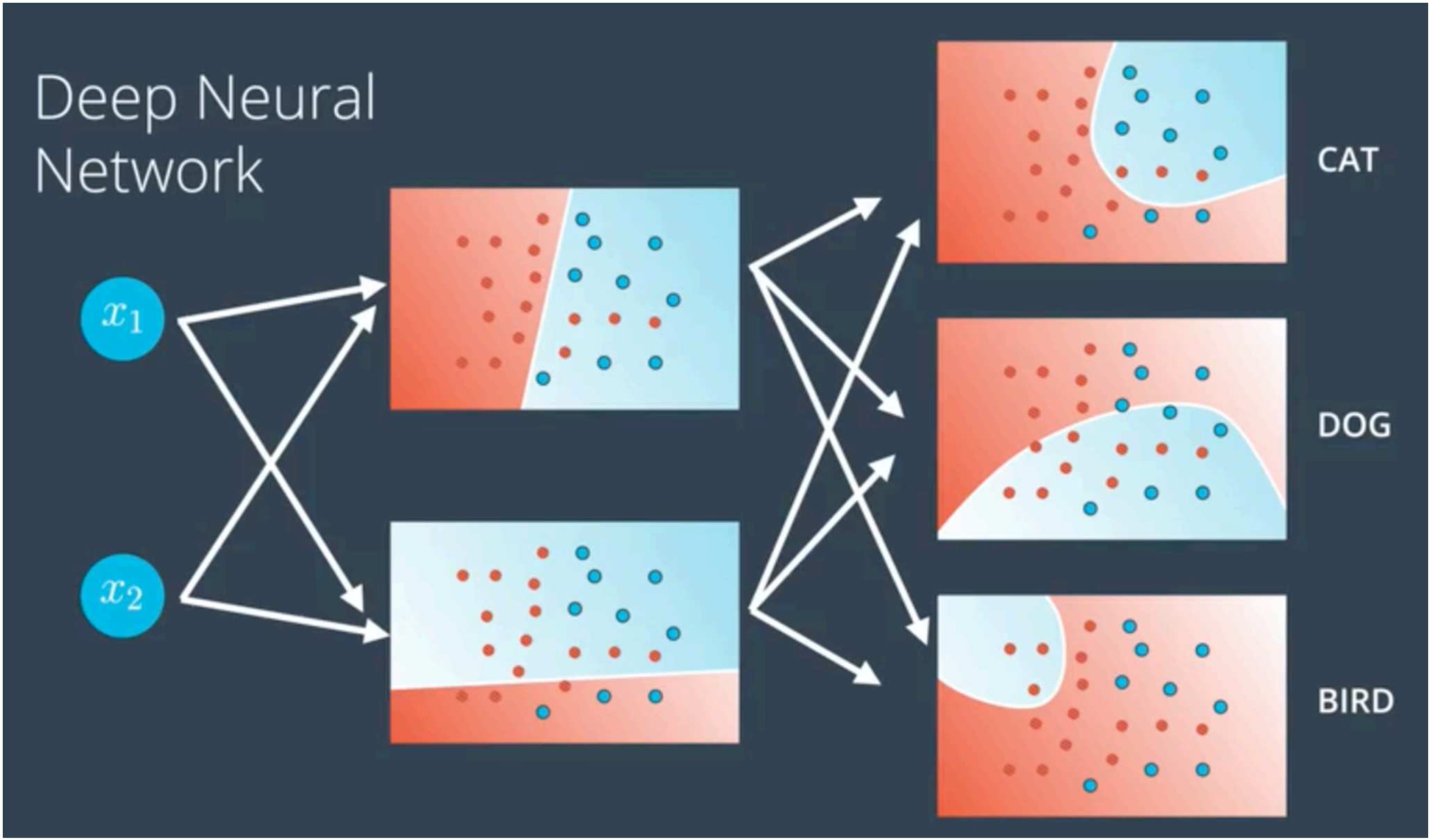
Multilayer Perceptron (MLP)

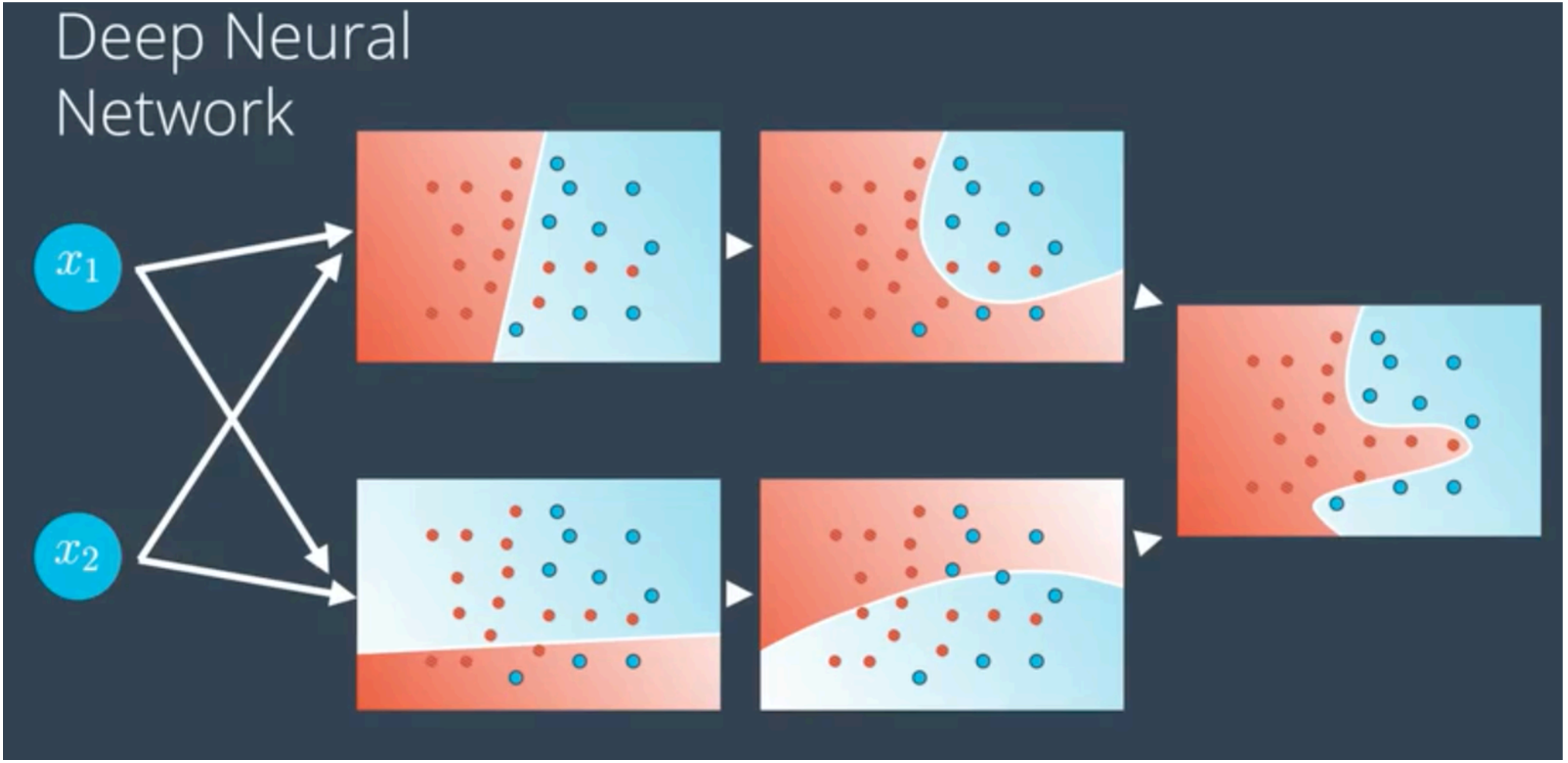


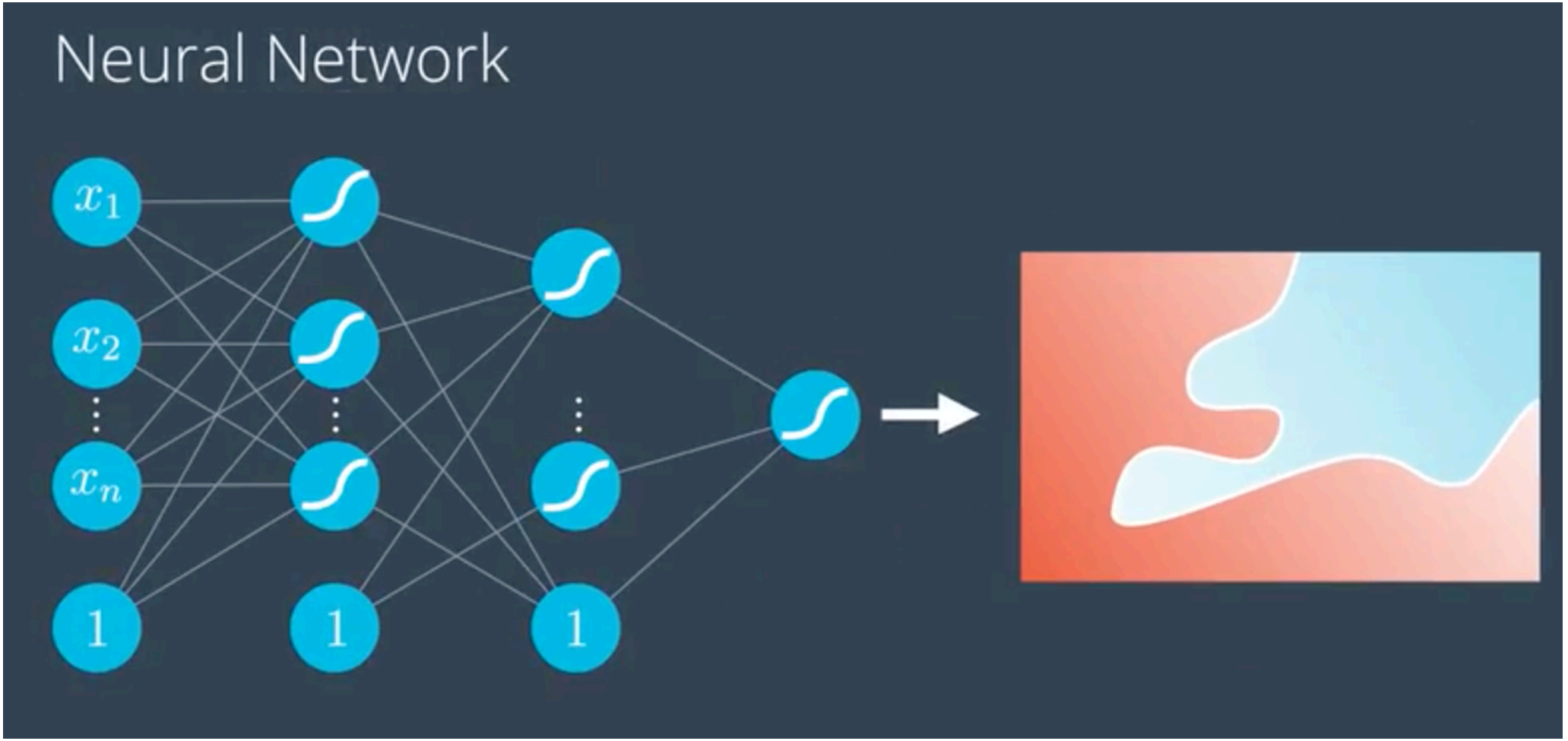

x_1
x_2

Multilayer Perceptron (MLP)



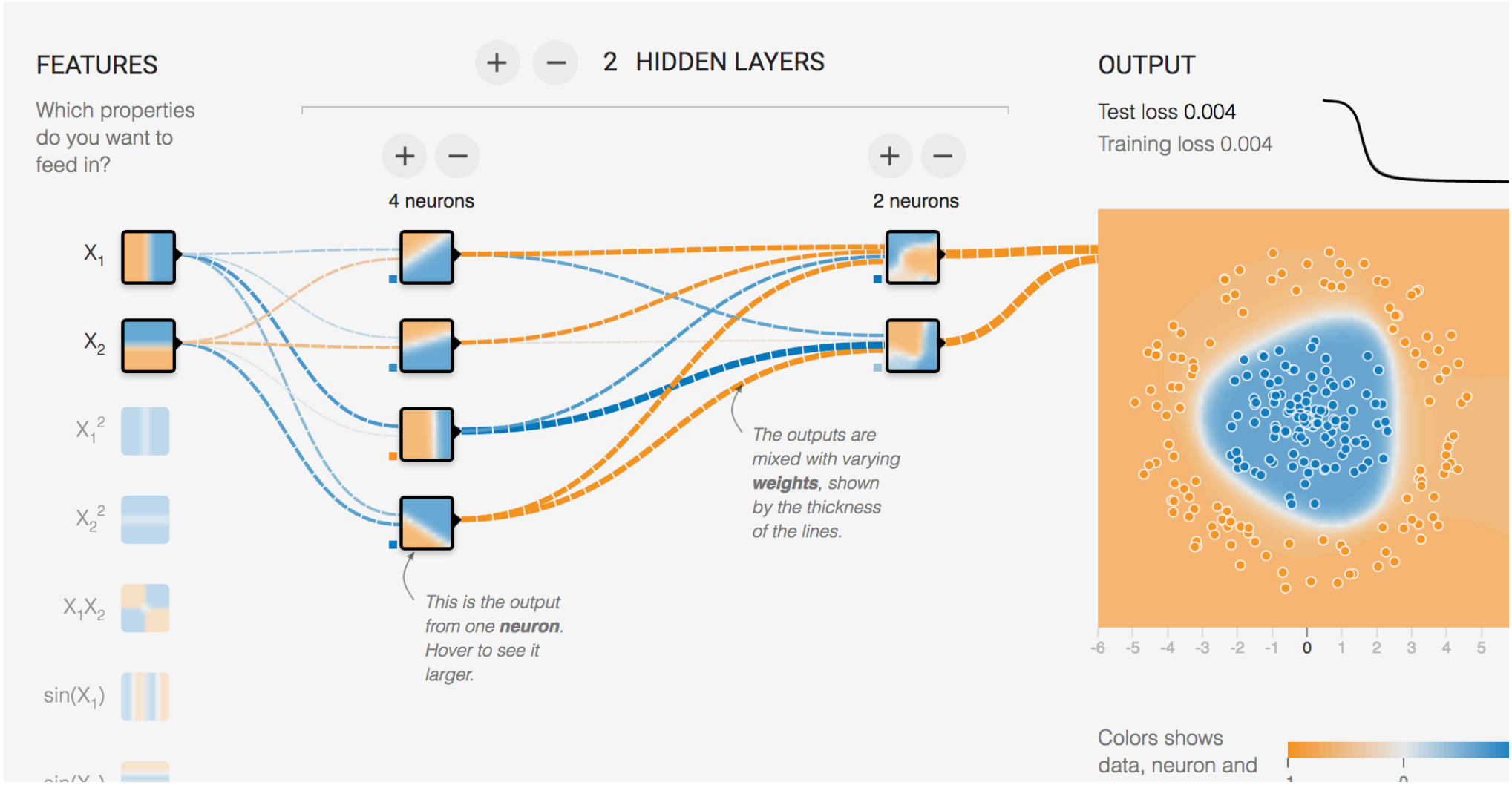

Deep Neural Network
x_1
x_2
CAT
DOG
BIRD


Deep Neural Network
x_1
x_2

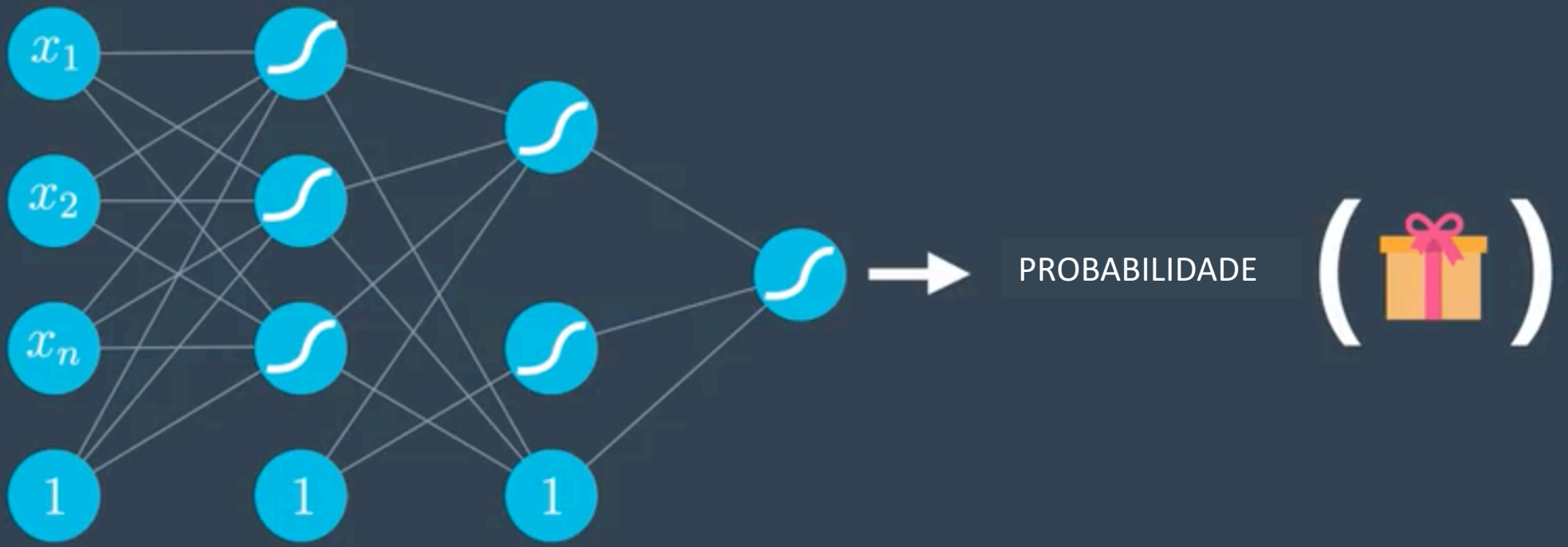

Neural Network
x_1
x_2
x_n
1
1
1

Exercicio

- http://playground.tensorflow.org



Classificação Binária



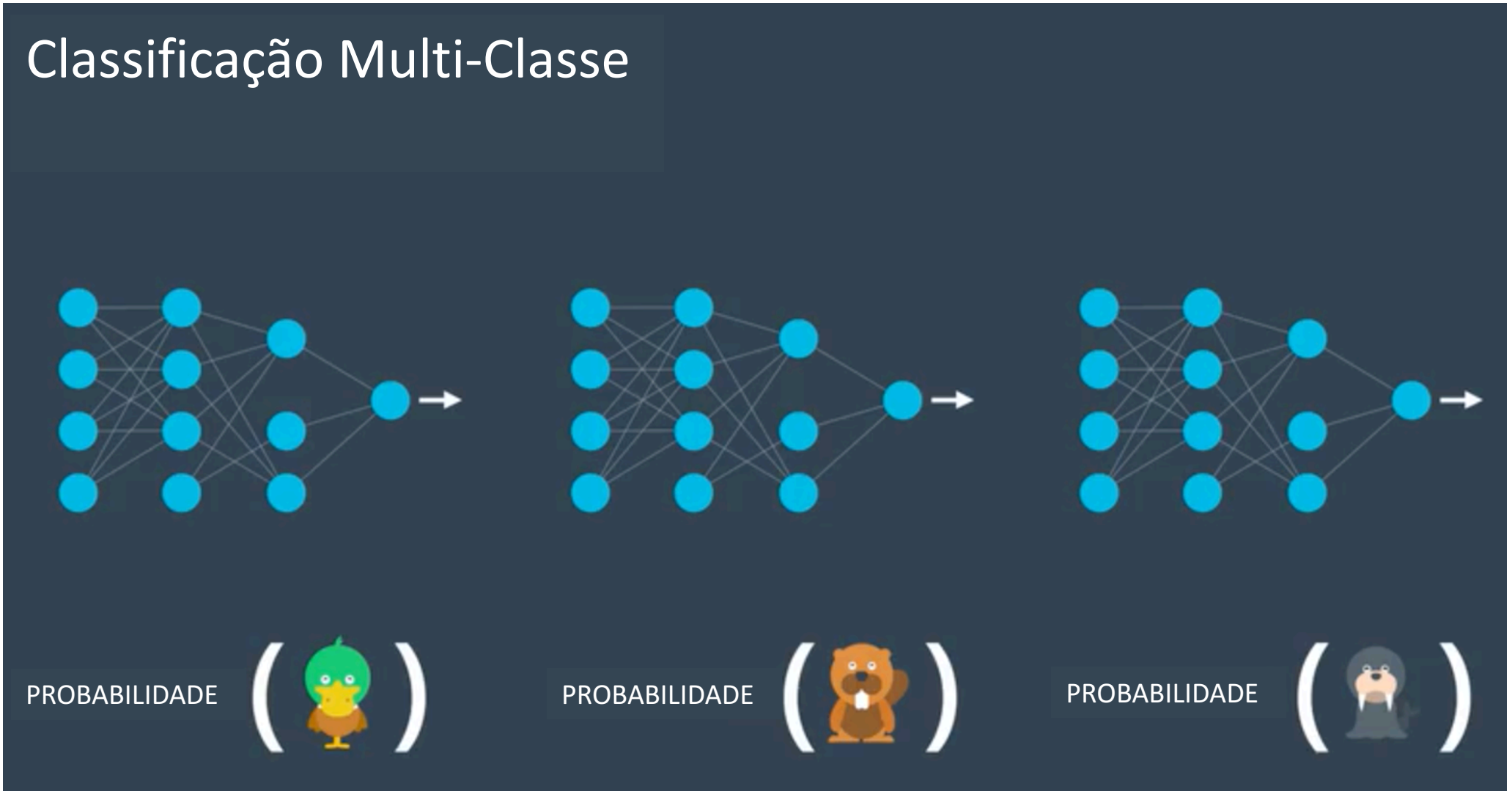
Classificação Multi-Classe

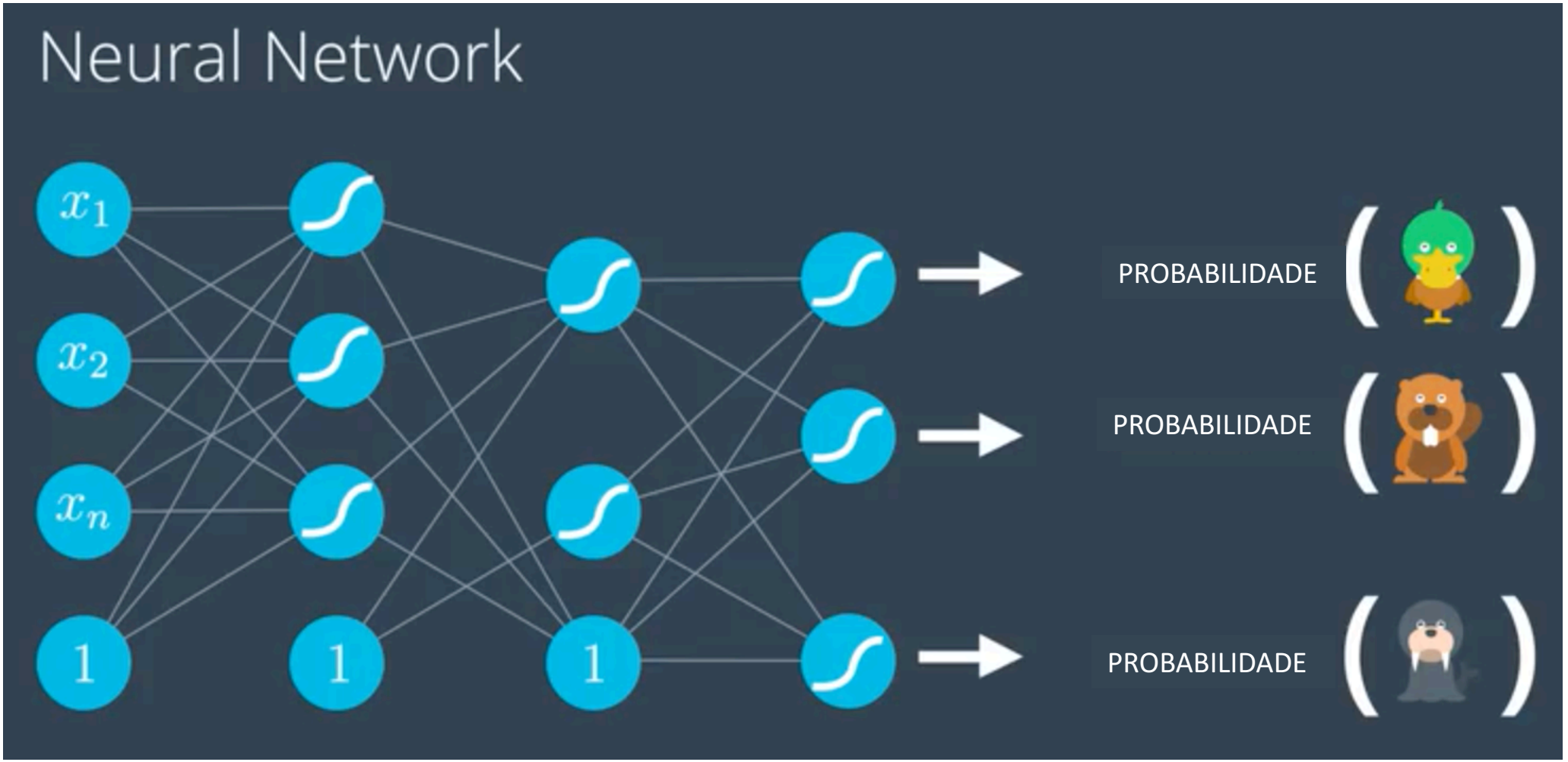
Pato

Castor

Morsa

Classificação Multi-Classe




Neural Network
x_1
x_2
x_n
1
1
1
PROBABILIDADE
PROBABILIDADE
PROBABILIDADE

Classificação Multi-Classe

P(Pato)=
0.67

P(Castor)=
0.24

P(Morsa)=
0.09

Função Softmax

$$P(classe\ i) = \frac{e^{z_i}}{e^{z_1} + \cdots + e^{z_n}}$$

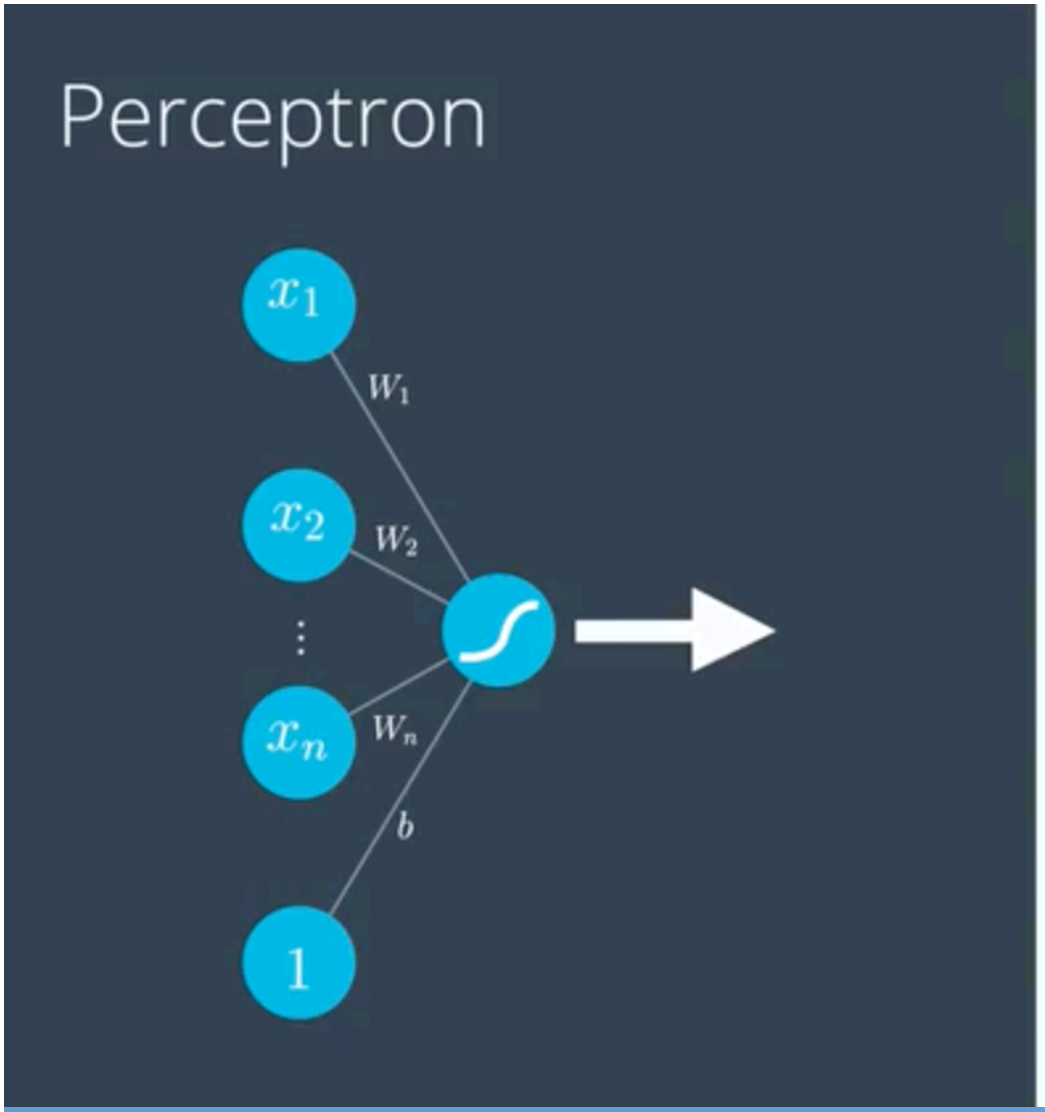
z_i= resultado do somatório do neurônio, antes de aplicar a função de ativação

$$\frac{e^2}{e^2 + e^1 + e^0} = 0.67$$ P(Pato)

$$\frac{e^1}{e^2 + e^1 + e^0} = 0.24$$ P(Castor)

$$\frac{e^0}{e^2 + e^1 + e^0} = 0.09$$ P(Morsa)

Revisando

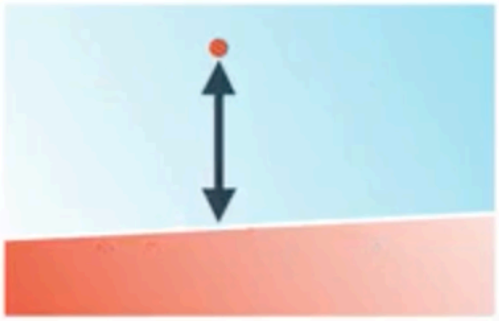


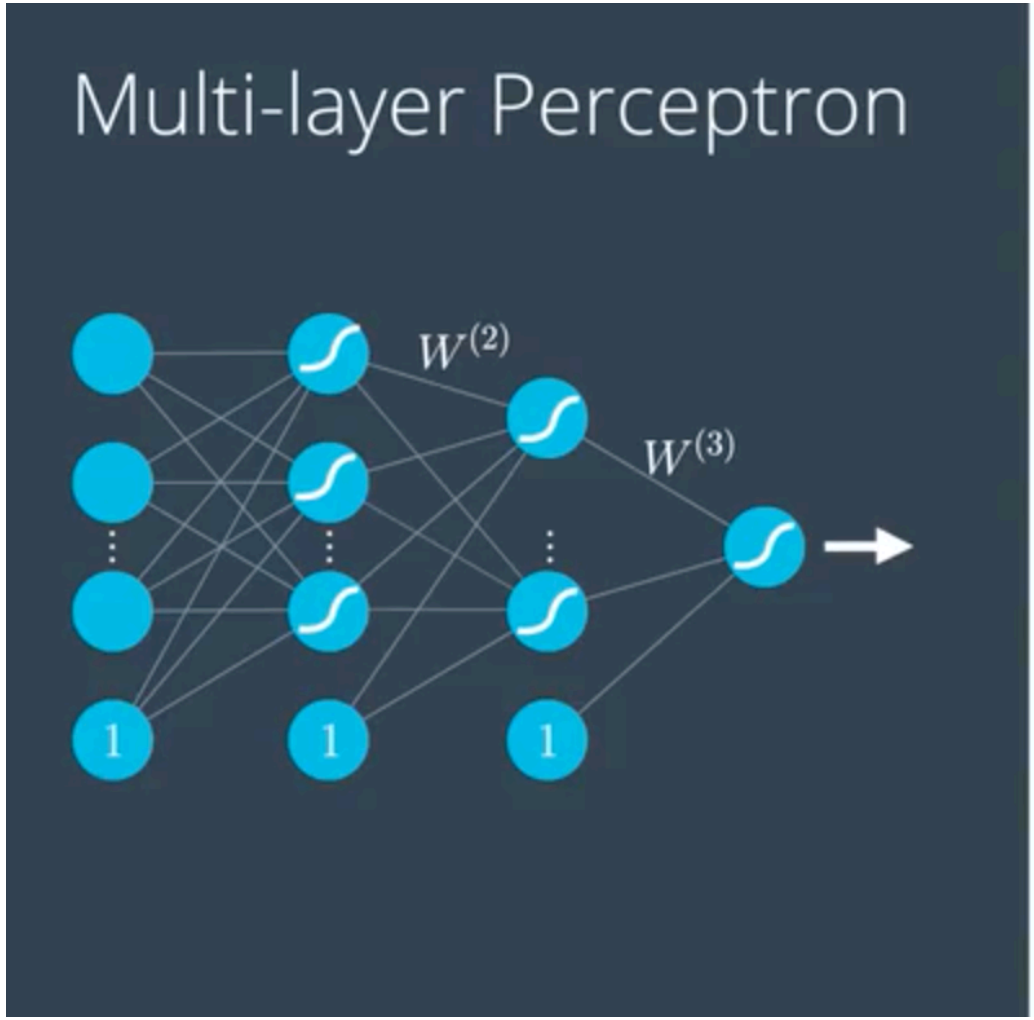
PREDIÇÃO

$$\hat{y} = \sigma(Wx + b)$$

FUNÇÃO DE ERRO

$$E(W) = -\frac{1}{m}\sum_{i=1}^{m} y_i \ln(\hat{y}_i) + (1 - y_i)\ln(1 - \hat{y}_i)$$



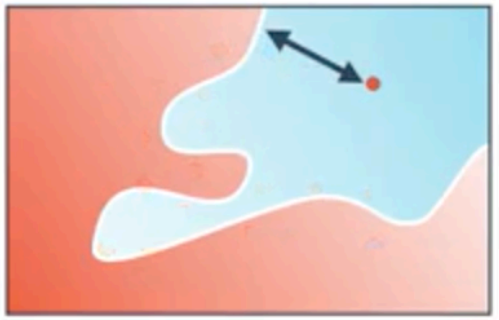


PREDIÇÃO

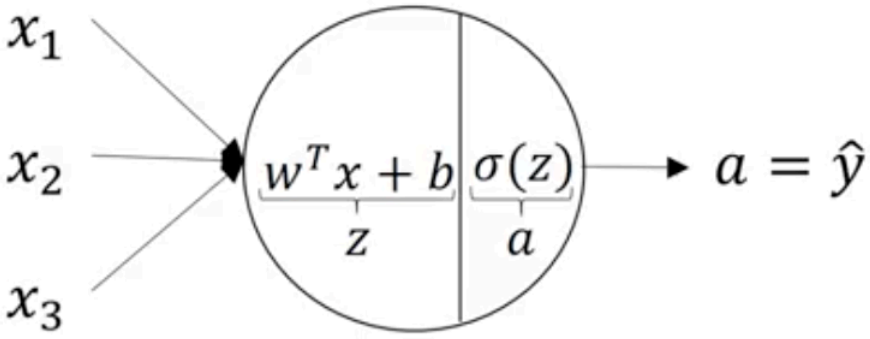
?

FUNÇÃO DE ERRO

$$E(W) = -\frac{1}{m}\sum_{i=1}^{m} y_i \ln(\hat{y}_i) + (1 - y_i)\ln(1 - \hat{y}_i)$$

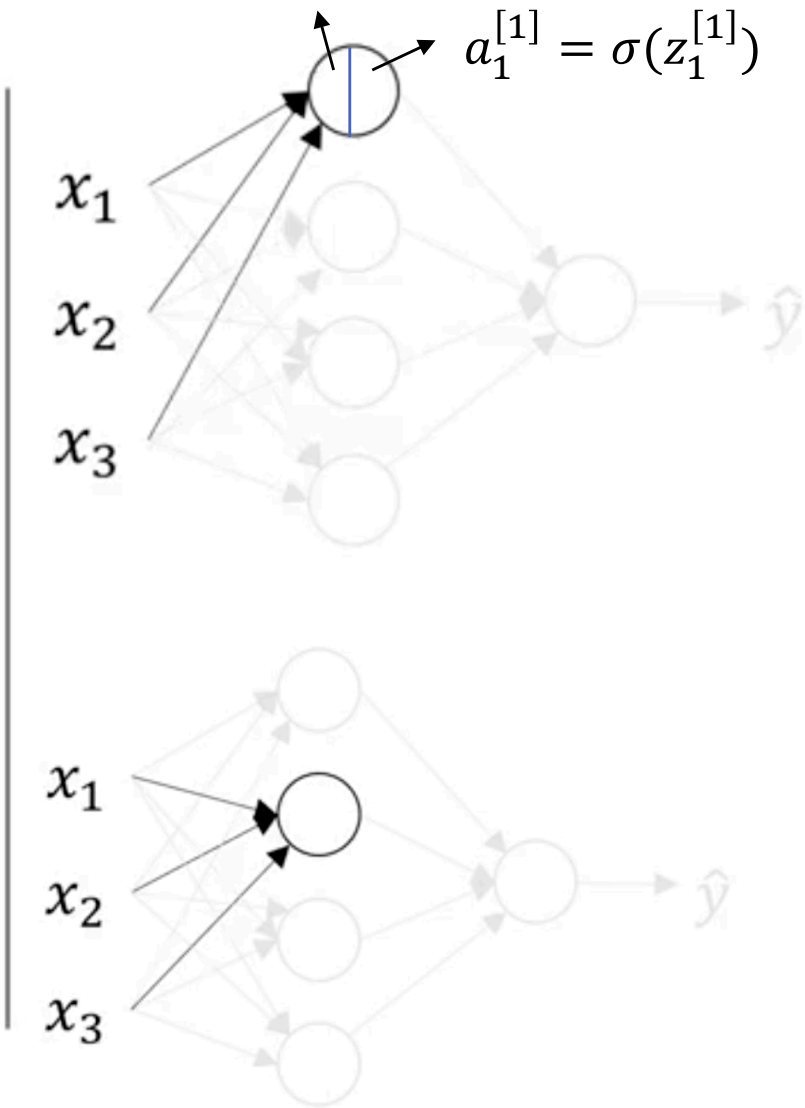


Representação

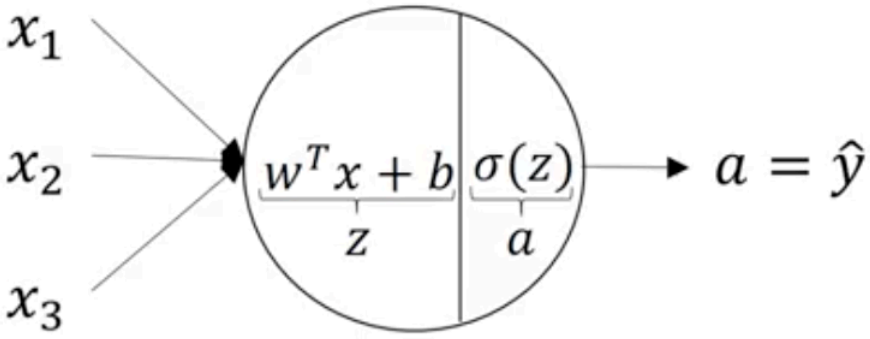


$$z = w^T x + b$$

$$a = \sigma(z)$$

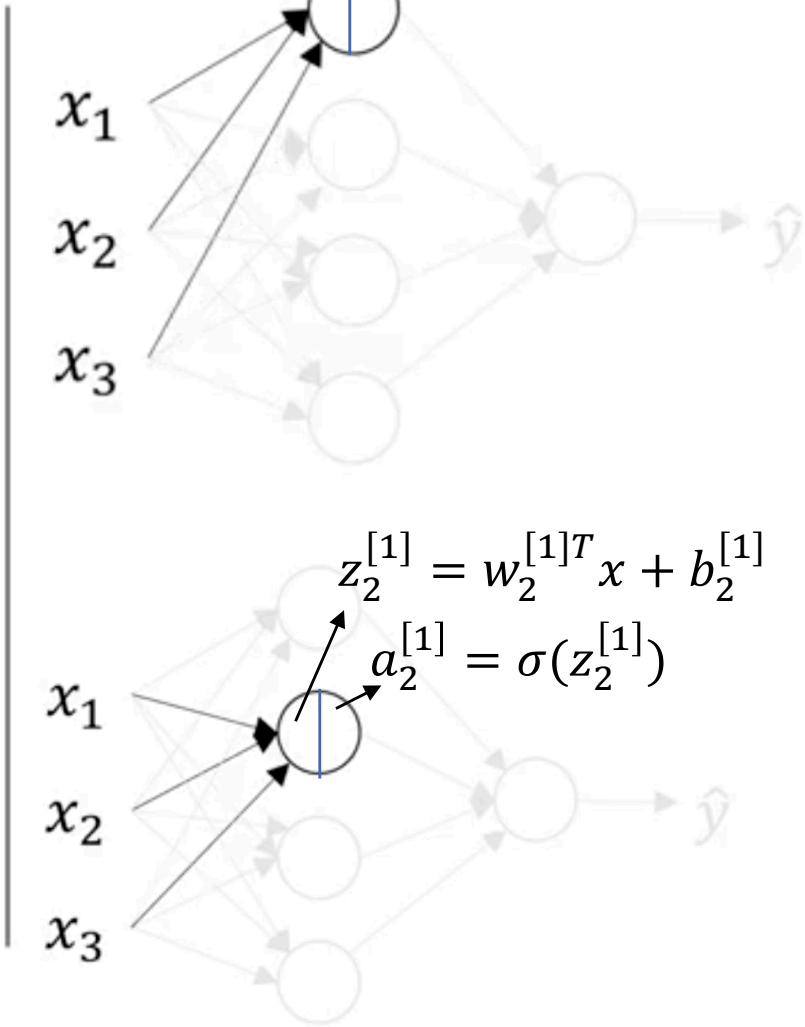


Representação

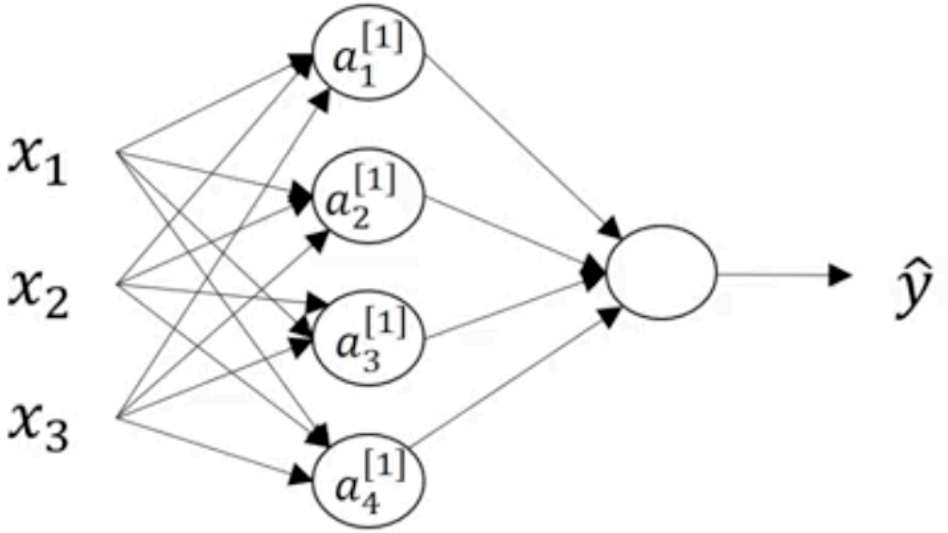


$$z = w^T x + b$$

$$a = \sigma(z)$$

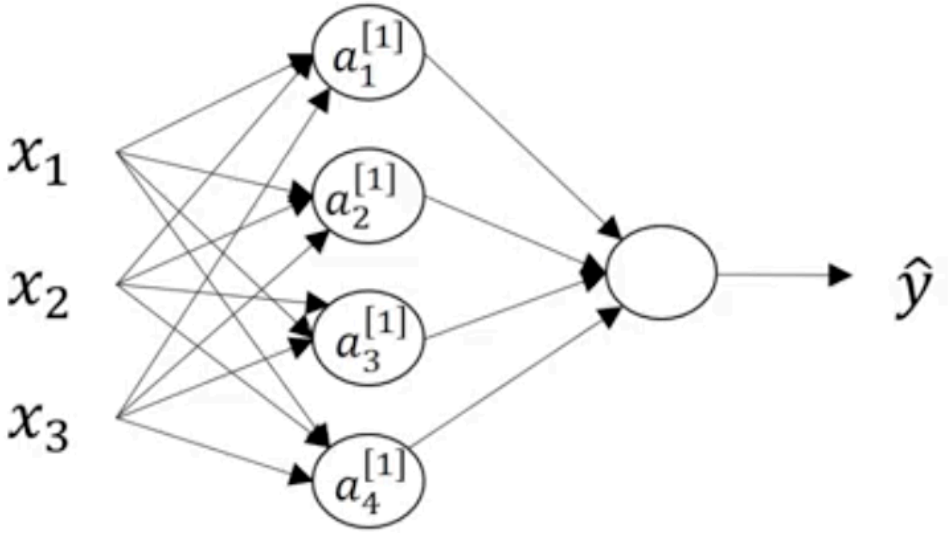


Representação



$$z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]}, a_1^{[1]} = \sigma(z_1^{[1]})$$
$$z_2^{[1]} = w_2^{[1]T} x + b_2^{[1]}, a_2^{[1]} = \sigma(z_2^{[1]})$$
$$z_3^{[1]} = w_3^{[1]T} x + b_3^{[1]}, a_3^{[1]} = \sigma(z_3^{[1]})$$
$$z_4^{[1]} = w_4^{[1]T} x + b_4^{[1]}, a_3^{[1]} = \sigma(z_4^{[1]})$$

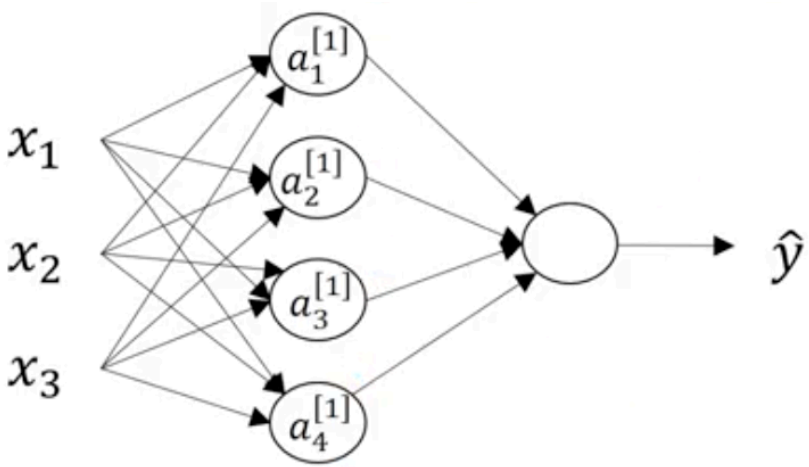
Representação



$$z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]}, a_1^{[1]} = \sigma(z_1^{[1]})$$
$$z_2^{[1]} = w_2^{[1]T} x + b_2^{[1]}, a_2^{[1]} = \sigma(z_2^{[1]})$$
$$z_3^{[1]} = w_3^{[1]T} x + b_3^{[1]}, a_3^{[1]} = \sigma(z_3^{[1]})$$
$$z_4^{[1]} = w_4^{[1]T} x + b_4^{[1]}, a_3^{[1]} = \sigma(z_4^{[1]})$$

$$Z^{[1]} = \underbrace{\begin{bmatrix} - & w_1^{[1]T} & - \\ - & w_2^{[1]T} & - \\ - & w_3^{[1]T} & - \\ - & w_4^{[1]T} & - \end{bmatrix}}_{W^{[1]}} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \underbrace{\begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix}}_{b^{[1]}} = \begin{bmatrix} w_1^{[1]T} x + b_1^{[1]} \\ w_2^{[1]T} x + b_2^{[1]} \\ w_3^{[1]T} x + b_3^{[1]} \\ w_4^{[1]T} x + b_4^{[1]} \end{bmatrix} = \begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \end{bmatrix}$$

Representação



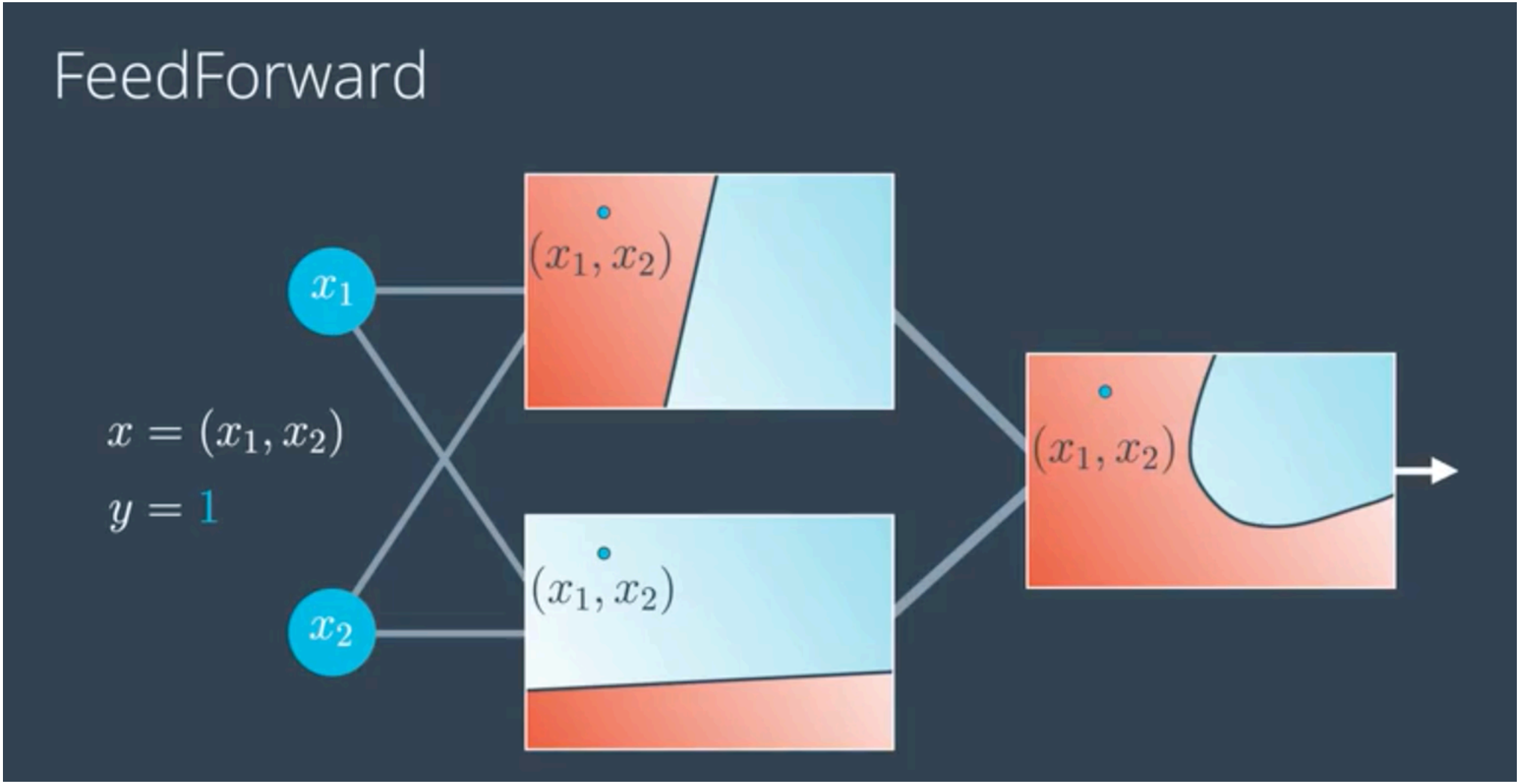
Dada a entrada x:

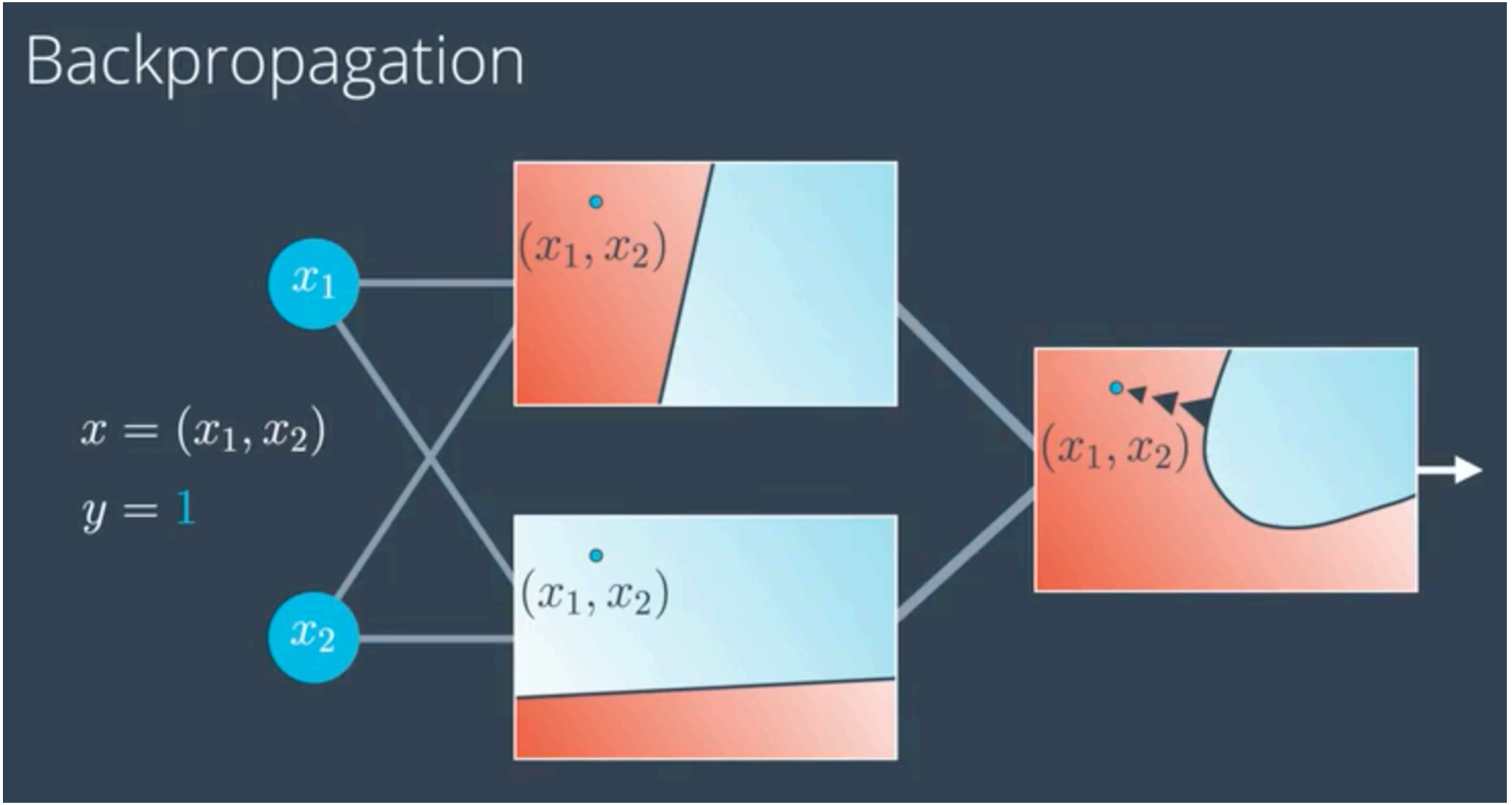
$$z^{[1]} = w^{[1]}x + b^{[1]}$$

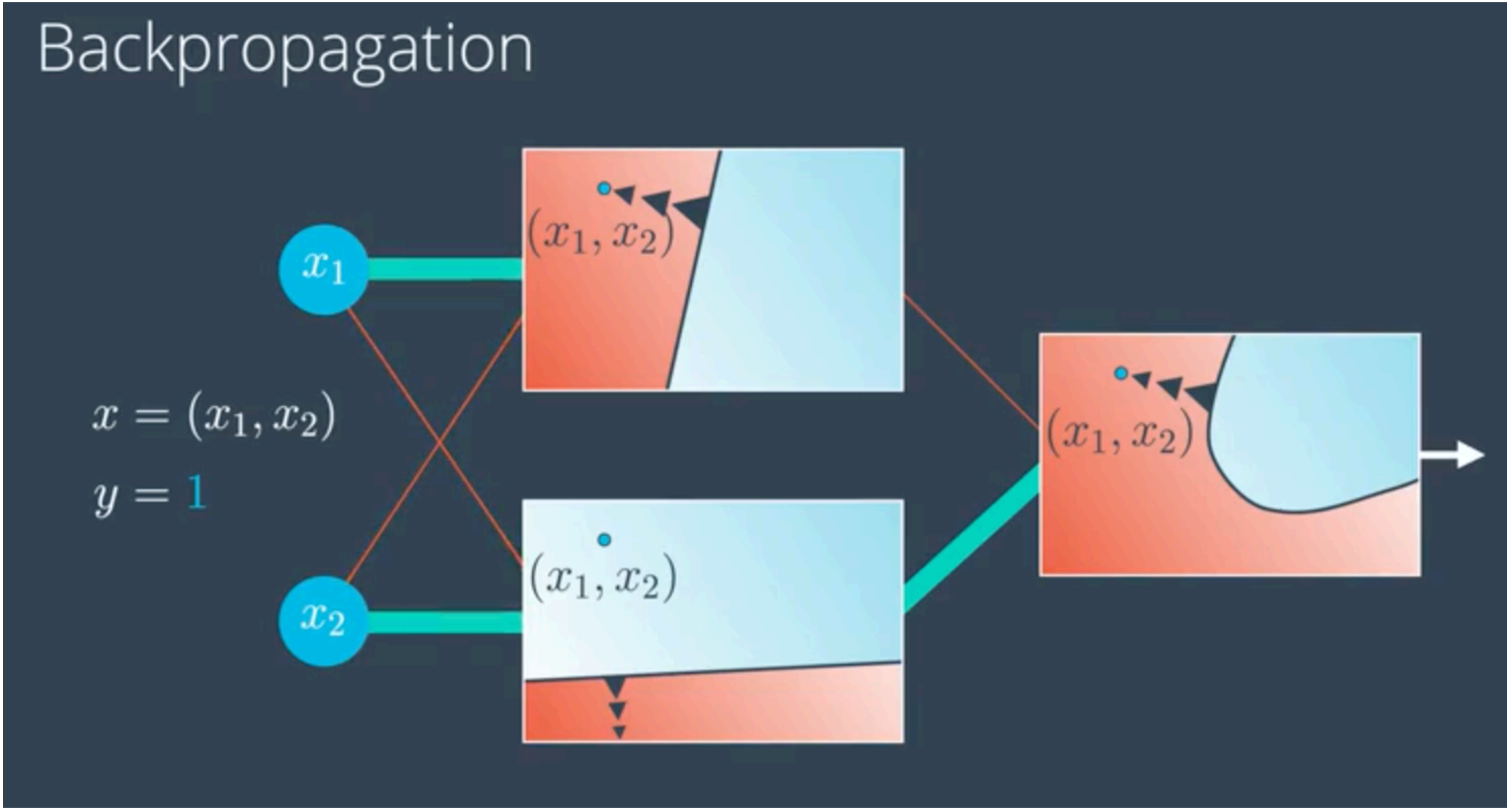
$$a^{[1]} = \sigma(z^{[1]})$$

$$z^{[2]} = w^{[2]}x + b^{[2]}$$

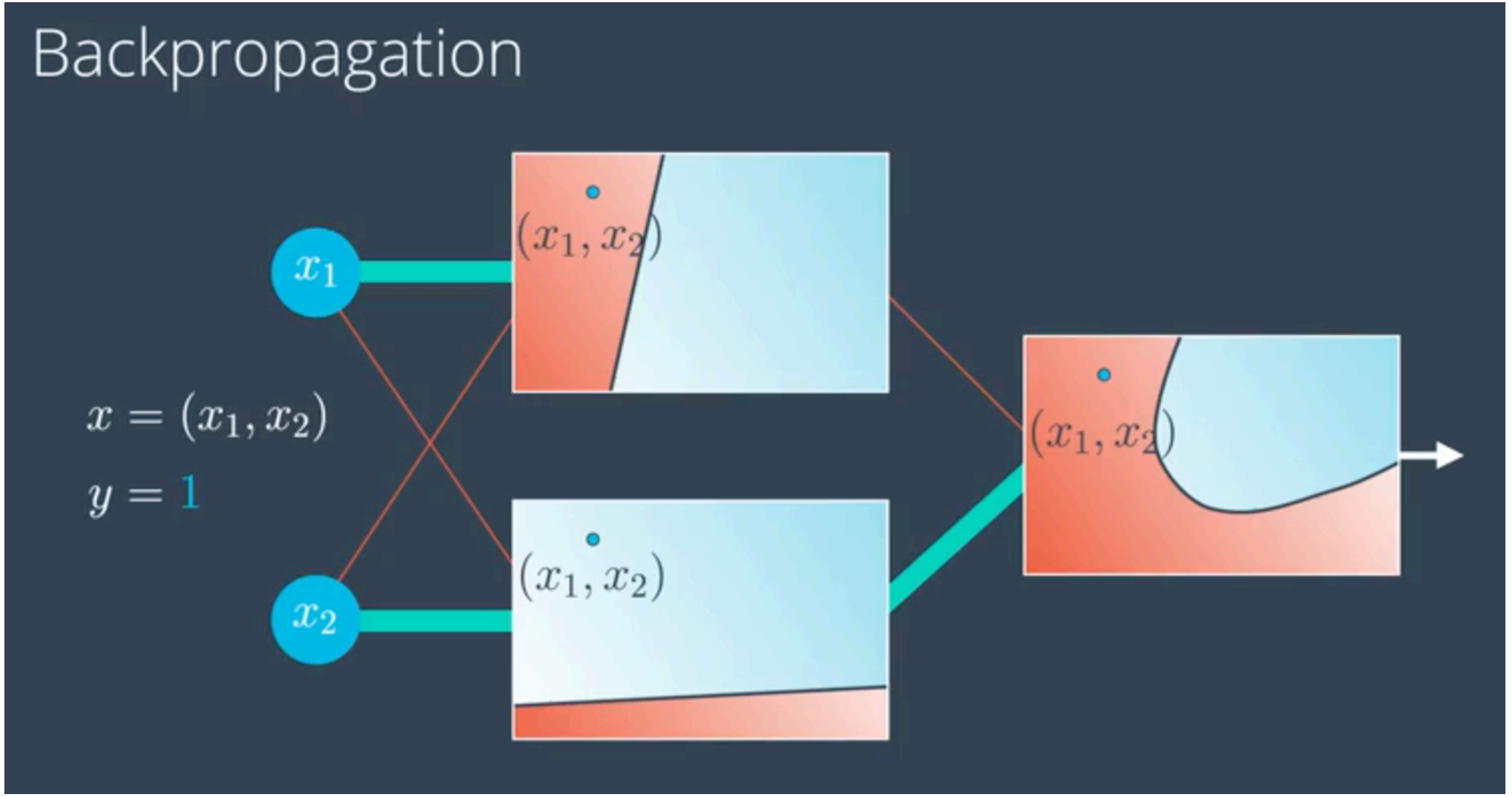
$$a^{[2]} = \sigma(z^{[2]})$$


FeedForward
x_1
x_2
$x = (x_1, x_2)$
$y = 1$
(x_1, x_2)
(x_1, x_2)
(x_1, x_2)


Backpropagation
x_1
x_2
$x = (x_1, x_2)$
$y = 1$
(x_1, x_2)
(x_1, x_2)
(x_1, x_2)


Backpropagation
$x = (x_1, x_2)$
$y = 1$
x_1
x_2
(x_1, x_2)
(x_1, x_2)
(x_1, x_2)

Backpropagation



Exercício

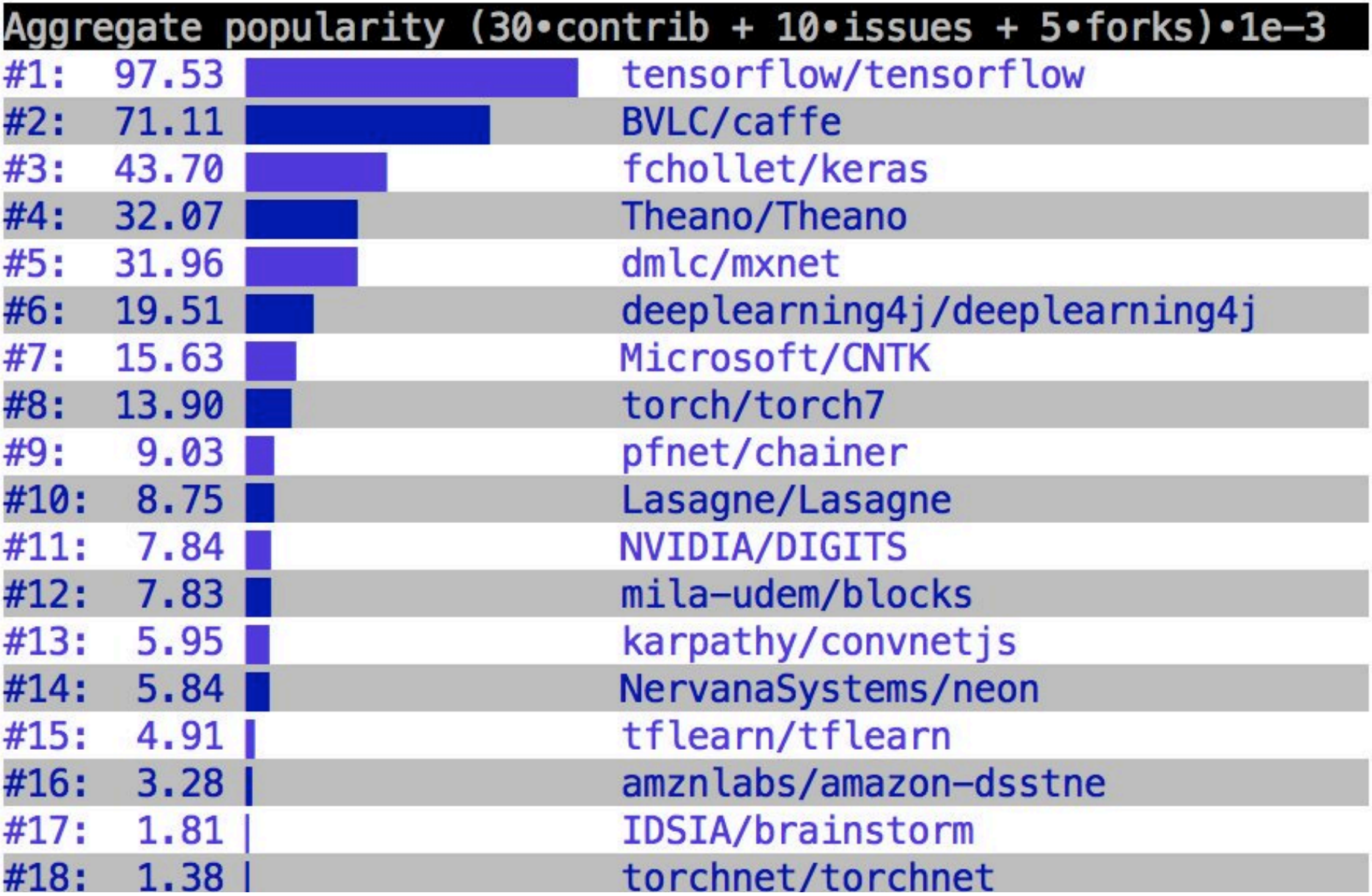
- MLP_code.ipynb

MLP com Keras

Frameworks

DEEPLEARNING4J

Popularidade



	Languages	Tutorials and training materials	CNN modeling capability	RNN modeling capability	Architecture: easy-to-use and modular front end	Speed	Multiple GPU support	Keras compatible
Theano	Python, C++	++	++	++	+	++	+	+
Tensor-Flow	Python	+++	+++	++	+++	++	++	+
Torch	Lua, Python (new)	+	+++	++	++	+++	++	
Caffe	C++	+	++		+	+	+	
MXNet	R, Python, Julia, Scala	++	++	+	++	++	+++	
Neon	Python	+	++	+	+	++	+	
CNTK	C++	+	+	+++	+	++	+	

Keras: High Level Wrapper

Keras é uma camada em cima do *Tensorflow*, que facilita alguns procedimentos

Também suporta *Theano* como backend

```
from keras.models import Sequential
from keras.layers.core import Dense, Activation
from keras.optimizers import SGD

N, D, H = 64, 1000, 100

model = Sequential()
model.add(Dense(input_dim=D, output_dim=H))
model.add(Activation('relu'))
model.add(Dense(input_dim=H, output_dim=D))

optimizer = SGD(lr=1e0)
model.compile(loss='mean_squared_error',
              optimizer=optimizer)

x = np.random.randn(N, D)
y = np.random.randn(N, D)
history = model.fit(x, y, nb_epoch=50,
                    batch_size=N, verbose=0)
```

Keras: High Level Wrapper

Define modelo como uma sequência de camadas

```
from keras.models import Sequential
from keras.layers.core import Dense, Activation
from keras.optimizers import SGD

N, D, H = 64, 1000, 100

model = Sequential()
model.add(Dense(input_dim=D, output_dim=H))
model.add(Activation('relu'))
model.add(Dense(input_dim=H, output_dim=D))

optimizer = SGD(lr=1e0)
model.compile(loss='mean_squared_error',
              optimizer=optimizer)

x = np.random.randn(N, D)
y = np.random.randn(N, D)
history = model.fit(x, y, nb_epoch=50,
                    batch_size=N, verbose=0)
```

Keras: High Level Wrapper

Define otimizador

```
from keras.models import Sequential
from keras.layers.core import Dense, Activation
from keras.optimizers import SGD

N, D, H = 64, 1000, 100

model = Sequential()
model.add(Dense(input_dim=D, output_dim=H))
model.add(Activation('relu'))
model.add(Dense(input_dim=H, output_dim=D))

optimizer = SGD(lr=1e0)
model.compile(loss='mean_squared_error',
              optimizer=optimizer)

x = np.random.randn(N, D)
y = np.random.randn(N, D)
history = model.fit(x, y, nb_epoch=50,
                    batch_size=N, verbose=0)
```

Keras: High Level Wrapper

Constrói modelo,
especifica função de perda

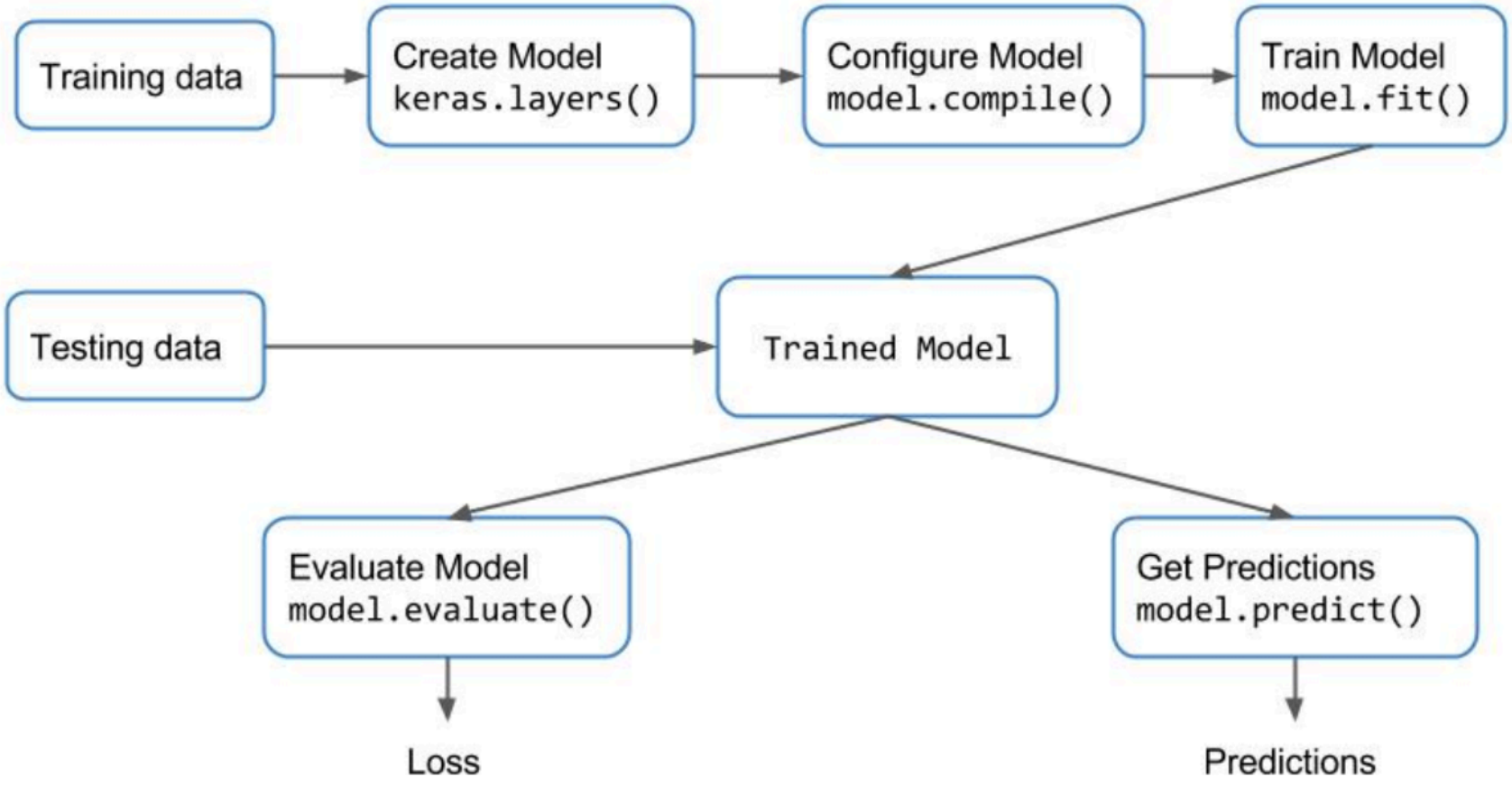
```
from keras.models import Sequential
from keras.layers.core import Dense, Activation
from keras.optimizers import SGD

N, D, H = 64, 1000, 100

model = Sequential()
model.add(Dense(input_dim=D, output_dim=H))
model.add(Activation('relu'))
model.add(Dense(input_dim=H, output_dim=D))

optimizer = SGD(lr=1e0)
model.compile(loss='mean_squared_error',
              optimizer=optimizer)

x = np.random.randn(N, D)
y = np.random.randn(N, D)
history = model.fit(x, y, nb_epoch=50,
                    batch_size=N, verbose=0)
```

Keras: High Level Wrapper

```
from keras.models import Sequential
from keras.layers.core import Dense, Activation
from keras.optimizers import SGD

N, D, H = 64, 1000, 100

model = Sequential()
model.add(Dense(input_dim=D, output_dim=H))
model.add(Activation('relu'))
model.add(Dense(input_dim=H, output_dim=D))

optimizer = SGD(lr=1e0)
model.compile(loss='mean_squared_error',
              optimizer=optimizer)

x = np.random.randn(N, D)
y = np.random.randn(N, D)
history = model.fit(x, y, nb_epoch=50,
                    batch_size=N, verbose=0)
```

Treina o modelo

Fluxograma Keras



MLP com Keras

- código minist_keras.ipynb

One-hot Encoding

ANIMAL	VALOR
	?
	?
	?
	?
	?

ANIMAL	PATO	CASTOR	MORSA?
	1	0	0
	0	1	0
	1	0	0
	0	0	1
	0	1	0

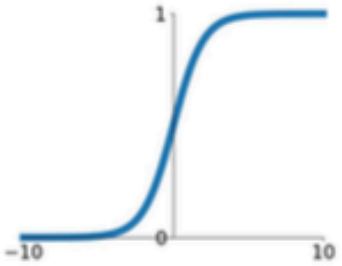
TODO

- Cifar10_keras.ipynb

Funções de Ativação

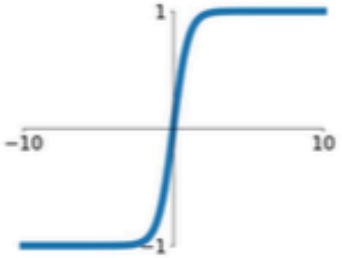
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



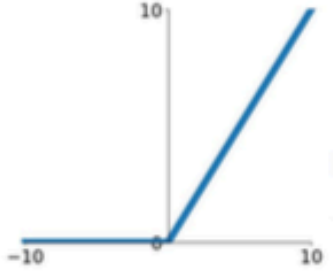
tanh

$$\tanh(x)$$



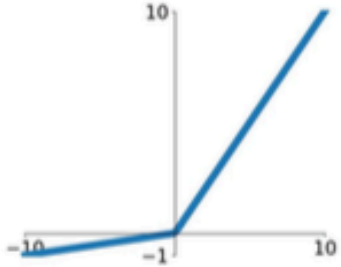
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

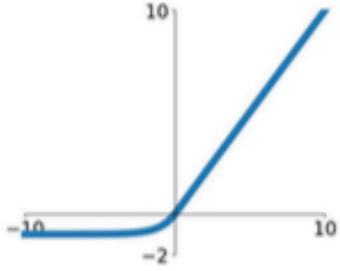


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Histórico das funções de ativação

Name	Formula	Year
none	$y = x$	-
sigmoid	$y = \frac{1}{1+e^{-x}}$	1986
tanh	$y = \frac{e^{2x}-1}{e^{2x}+1}$	1986
ReLU	$y = \max(x, 0)$	2010
(centered) SoftPlus	$y = \ln(e^x + 1) - \ln 2$	2011
LReLU	$y = \max(x, \alpha x), \alpha \approx 0.01$	2011
maxout	$y = \max(W_1 x + b_1, W_2 x + b_2)$	2013
APL	$y = \max(x,0) + \sum_{s=1}^{S} a_i^s \max(0, -x + b_i^s)$	2014
VLReLU	$y = \max(x, \alpha x), \alpha \in 0.1, 0.5$	2014
RReLU	$y = \max(x, \alpha x), \alpha = \text{random}(0.1, 0.5)$	2015
PReLU	$y = \max(x, \alpha x), \alpha$ is learnable	2015
ELU	$y = x$, if $x \geq 0$, else $\alpha(e^x - 1)$	2015


IDEAS
Think?
good
click
Inspiration
CHECK LIST:
- SEO
- 3G / 4G
cloud
SHARE