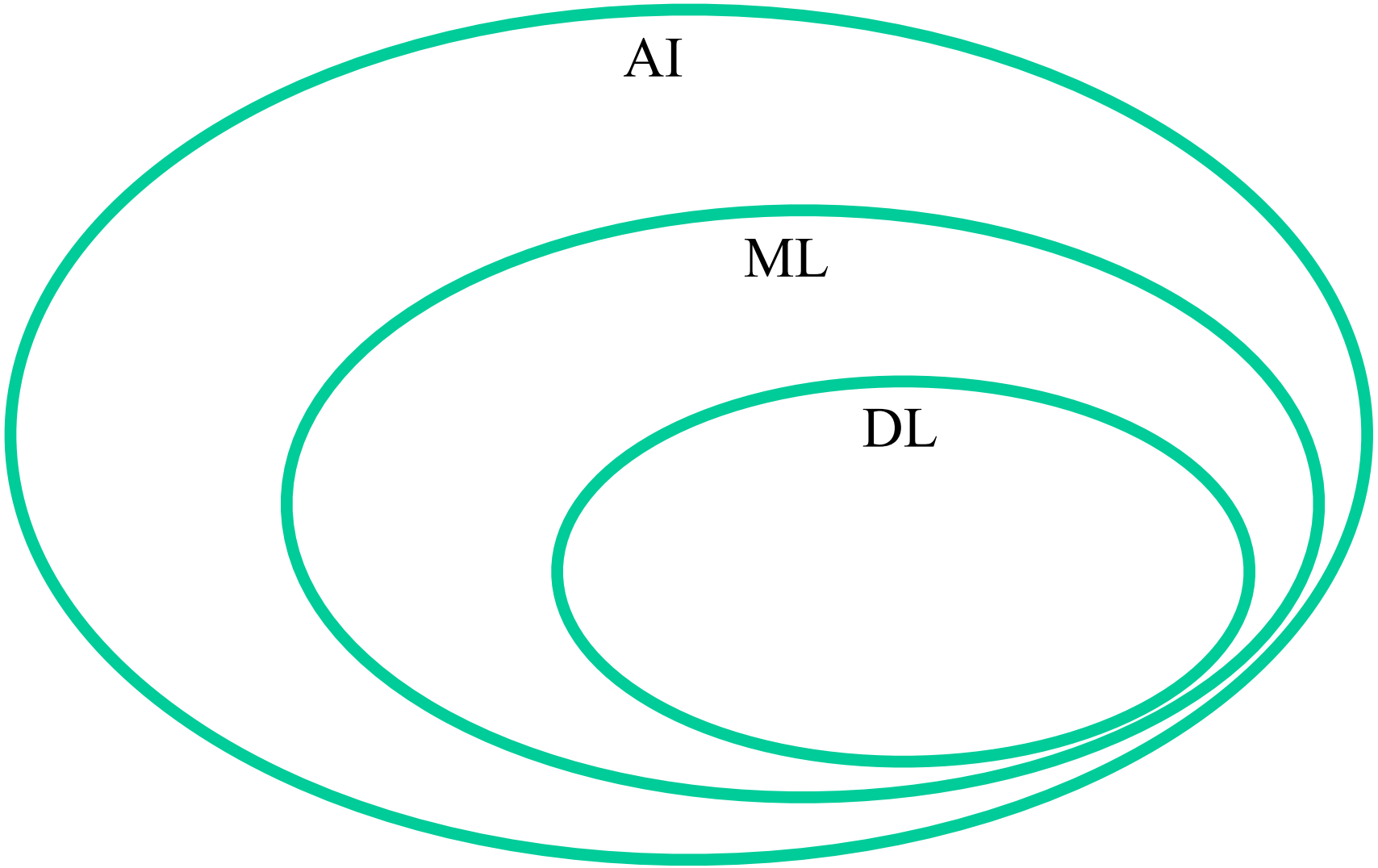
Aplicações de Inteligência Artificial
Parte IV

Hitoshi Nagano, Ph.D.

Redes Neurais Artificiais

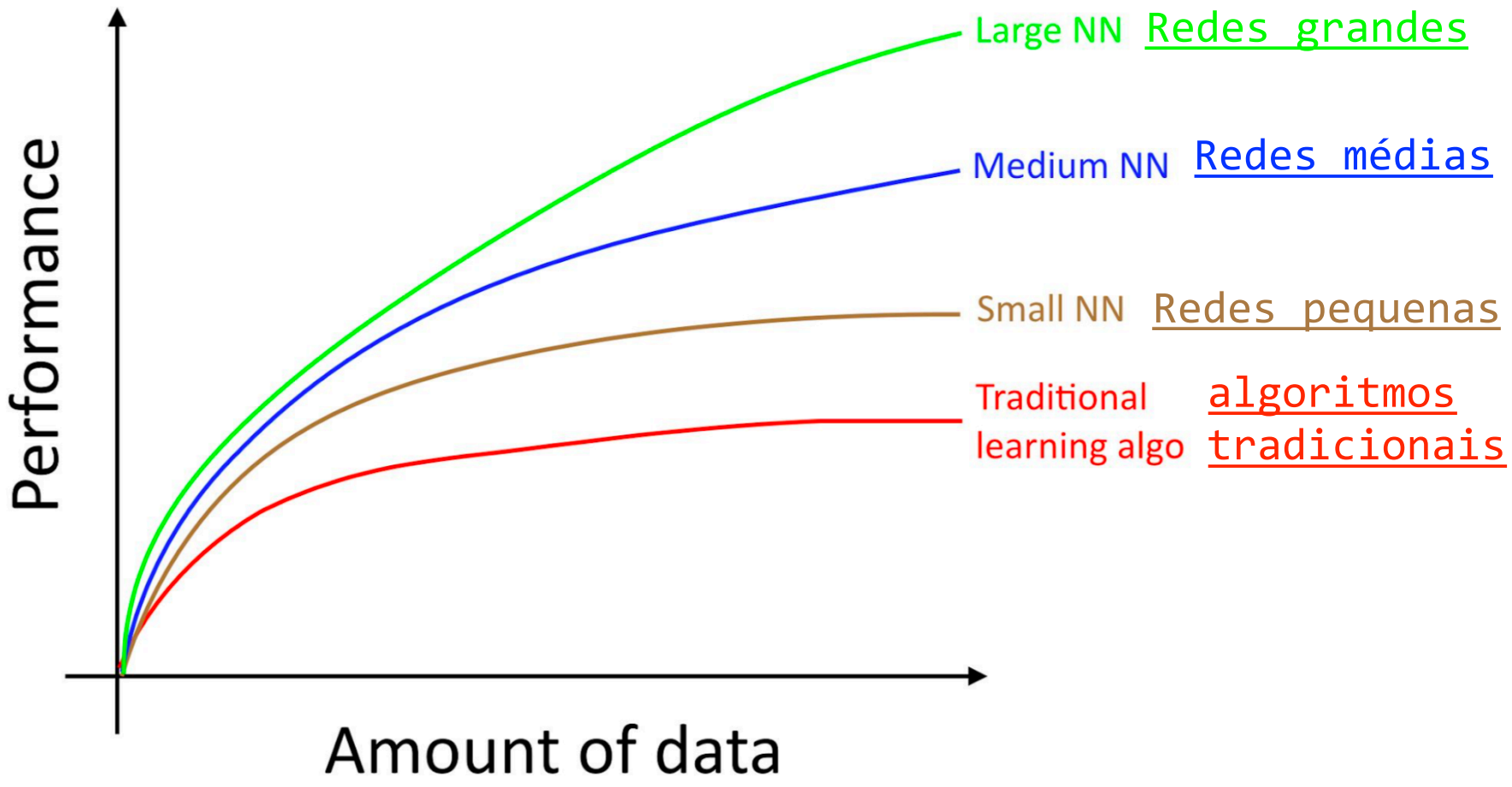
referencias

- livros
  http://neuralnetworksanddeeplearning.com
  http://www.deeplearningbook.org/lecture_slides.html
- cursos
  http://cs231n.github.io/
  https://web.stanford.edu/class/cs224n/
  https://www.coursera.org/specializations/deep-learning
- blogs
  http://colah.github.io/


AI
ML
DL

sistemas ML e não-ML

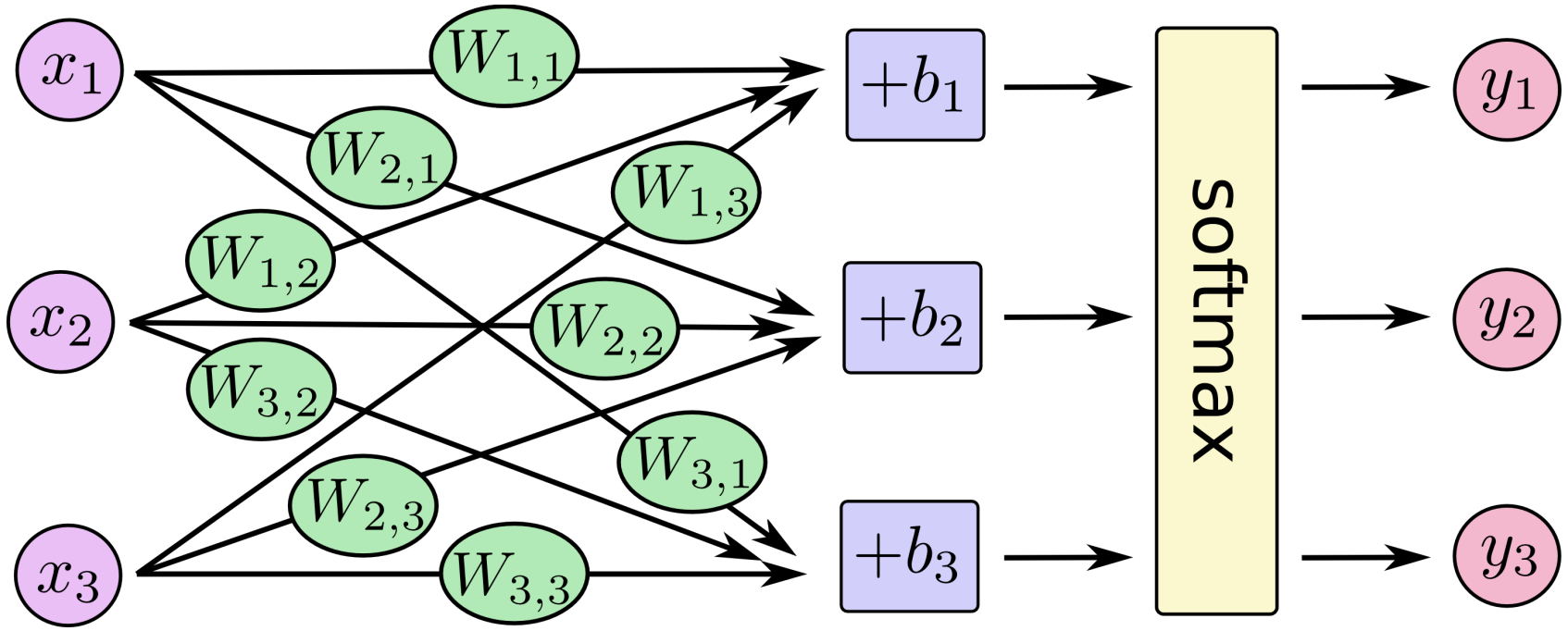
Performance



roteiro

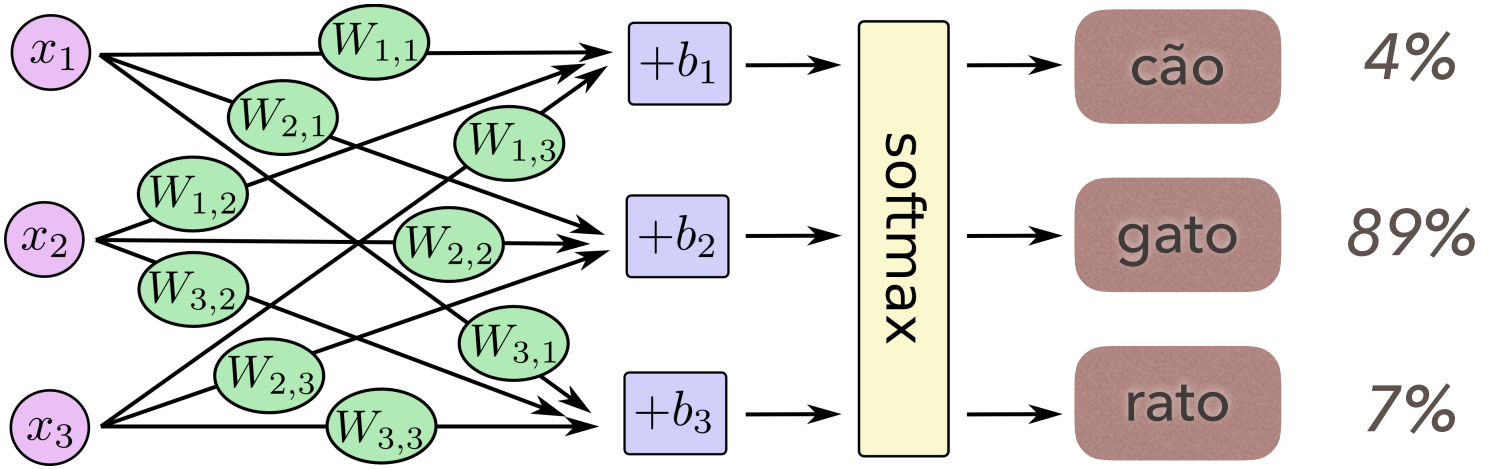
- arquitetura
- forward propagation
- backpropagation
- otimização
- tensorflow & keras

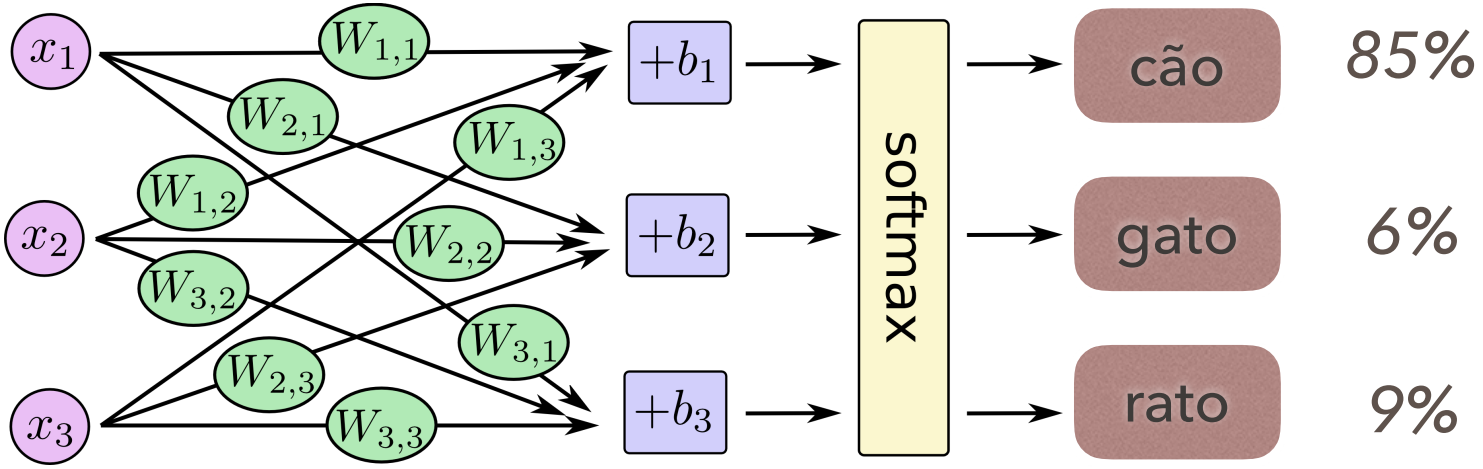
arquitetura



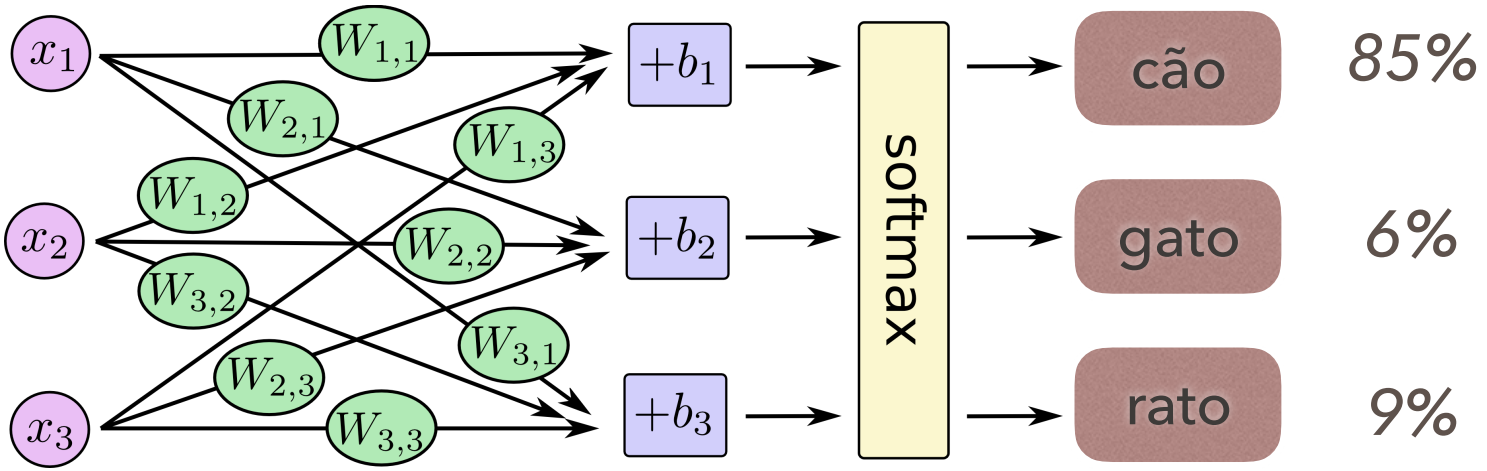
https://www.tensorflow.org/get_started/mnist/beginners

qual é o problema?





qual é o problema?



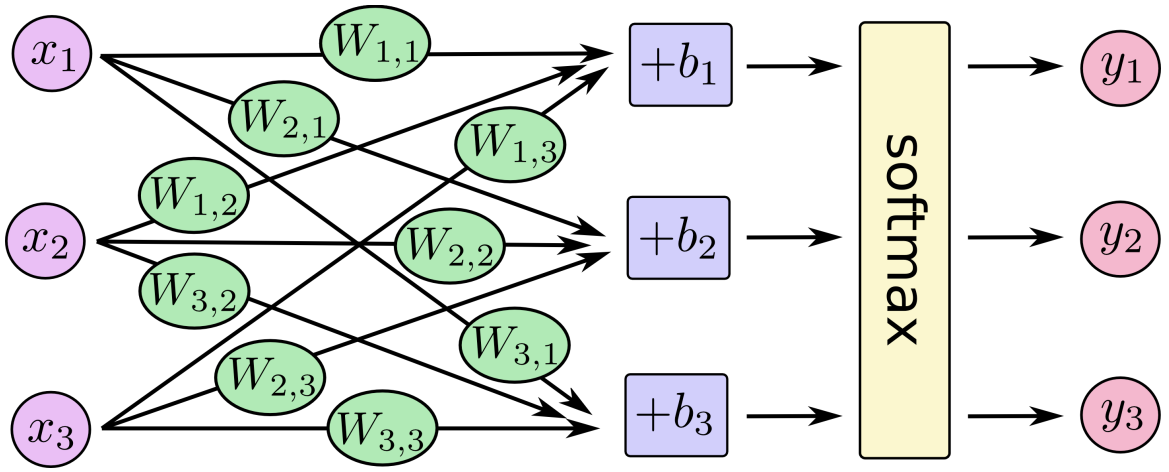
x

	pixel1	pixel2	pixel3	...	pixel n
🐱	2	133	25	...	77
🐶	3	3	55	...	89
🐭	254	255	253	...	255
😾	8	8	10	...	21

y

cão	gato	rato
0	1	0
1	0	0
0	0	1
0	1	0

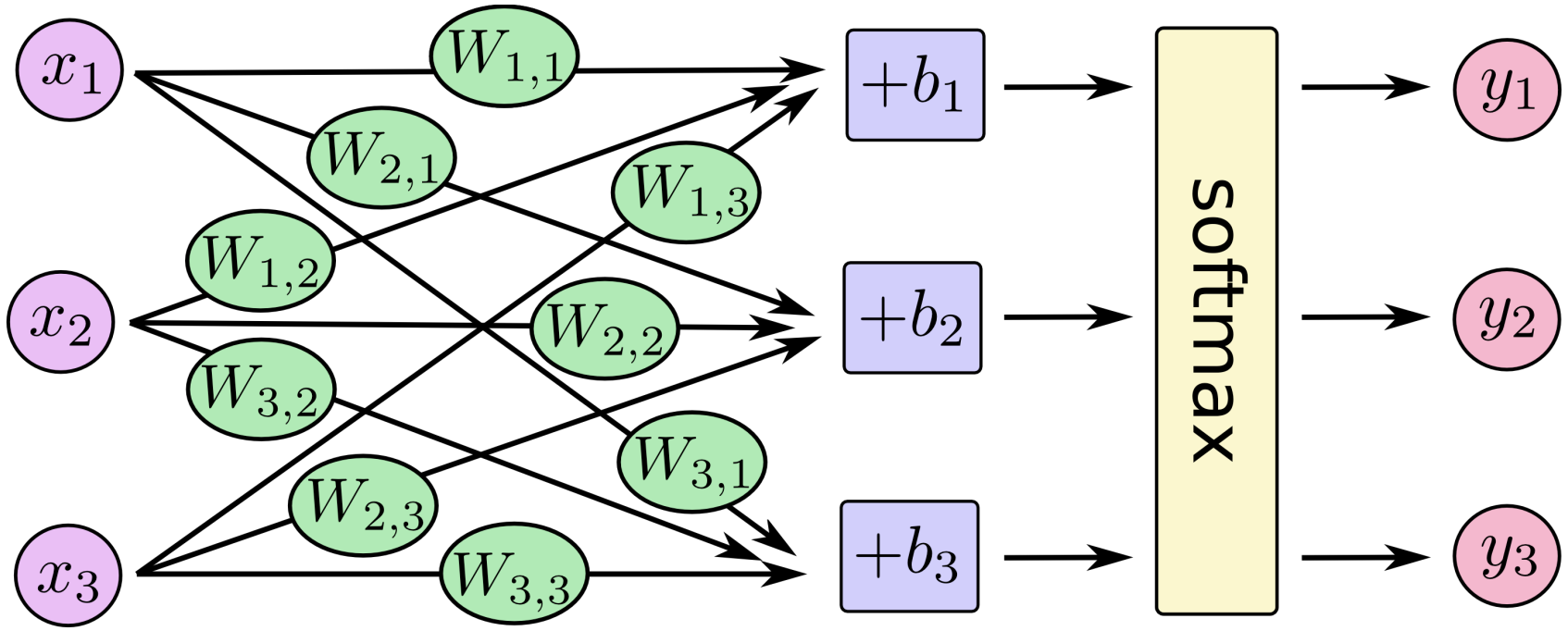
formulação

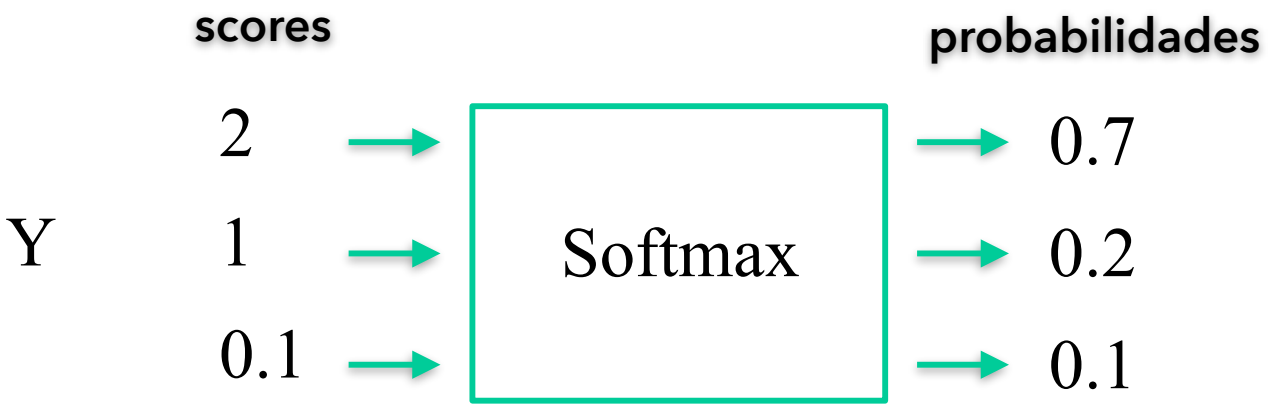


$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax}\left(\begin{bmatrix} W_{1,1} & W_{1,2} & W_{1,3} \\ W_{2,1} & W_{2,2} & W_{2,3} \\ W_{3,1} & W_{3,2} & W_{3,3} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \right)$$

$$y = softmax(\boldsymbol{W}^T x + \boldsymbol{b})$$

E… o que é esse softmax?



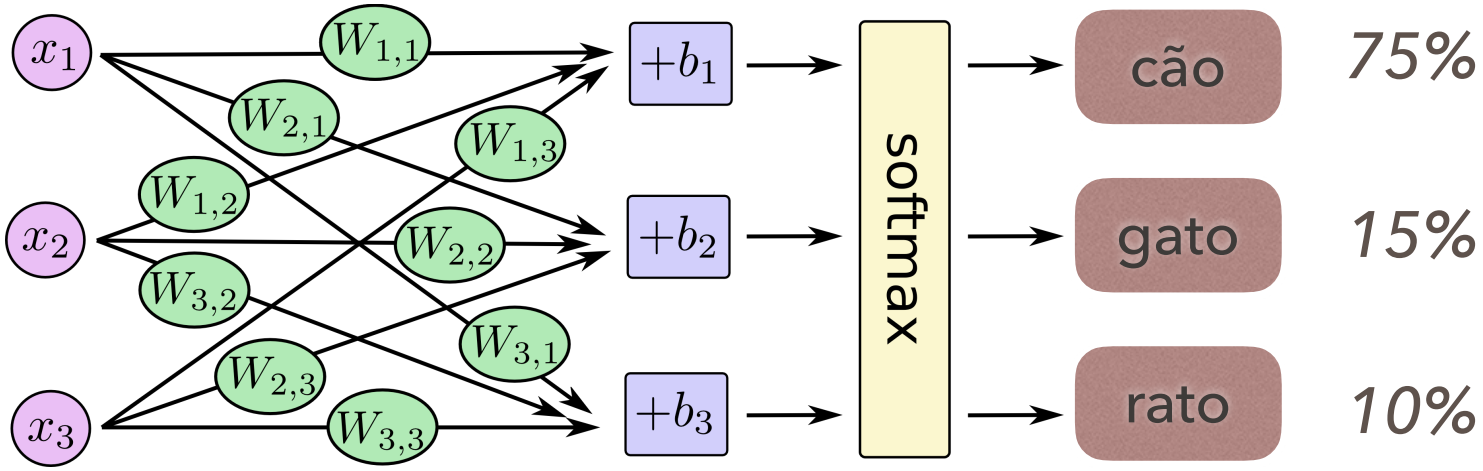


neuralnetworksanddeeplearning.com/chap3.html#softmax

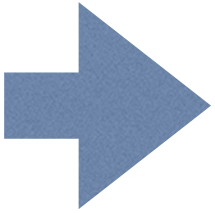
	scores	e^scores	probabilidades
y_0	2	7,389	0,67
y_1	1	2,718	0,25
y_2	-0,1	0,905	0,08
fator de normalização ➜		11,012	

um "bom" modelo...

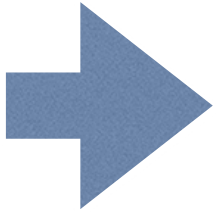
ESSE MODELO É "BOM"?



erra menos
acerta mais

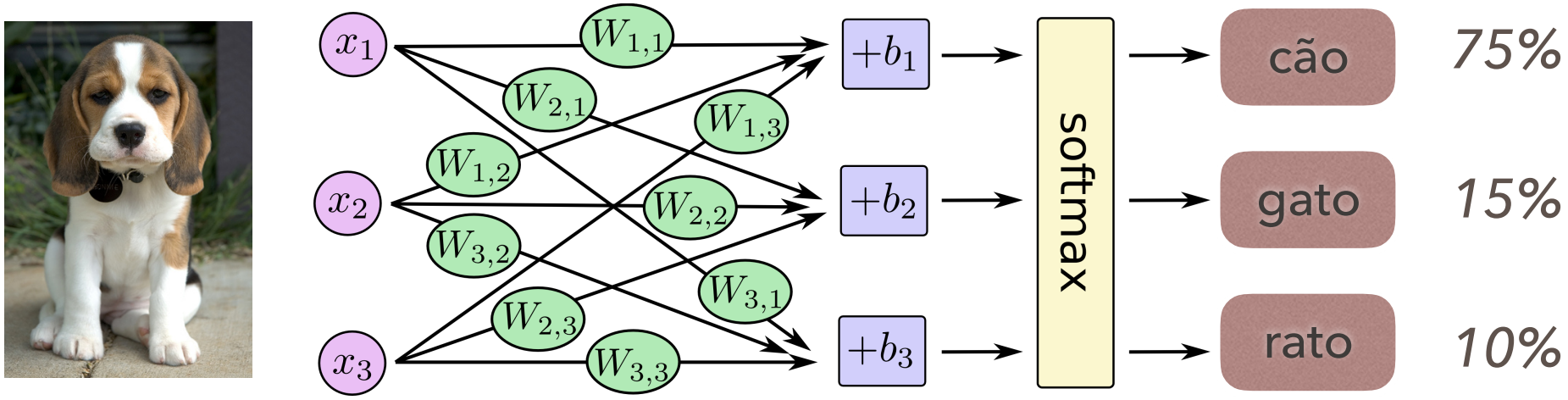


"bons"
W & b



"bom"
modelo

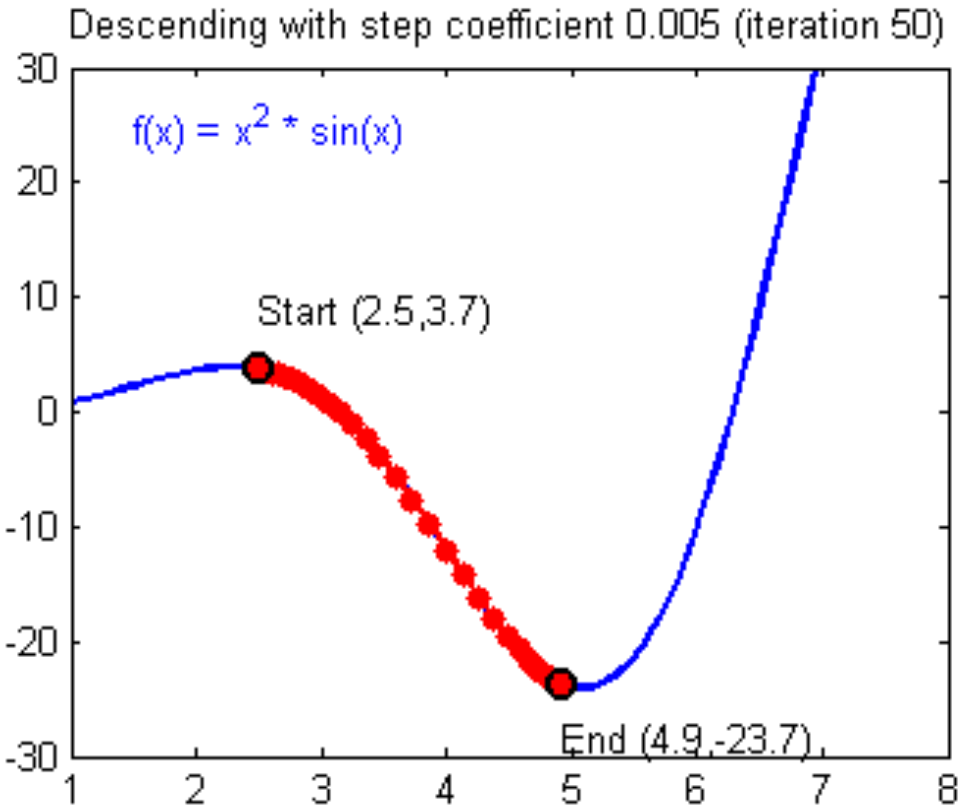
como achar “bons” pesos?

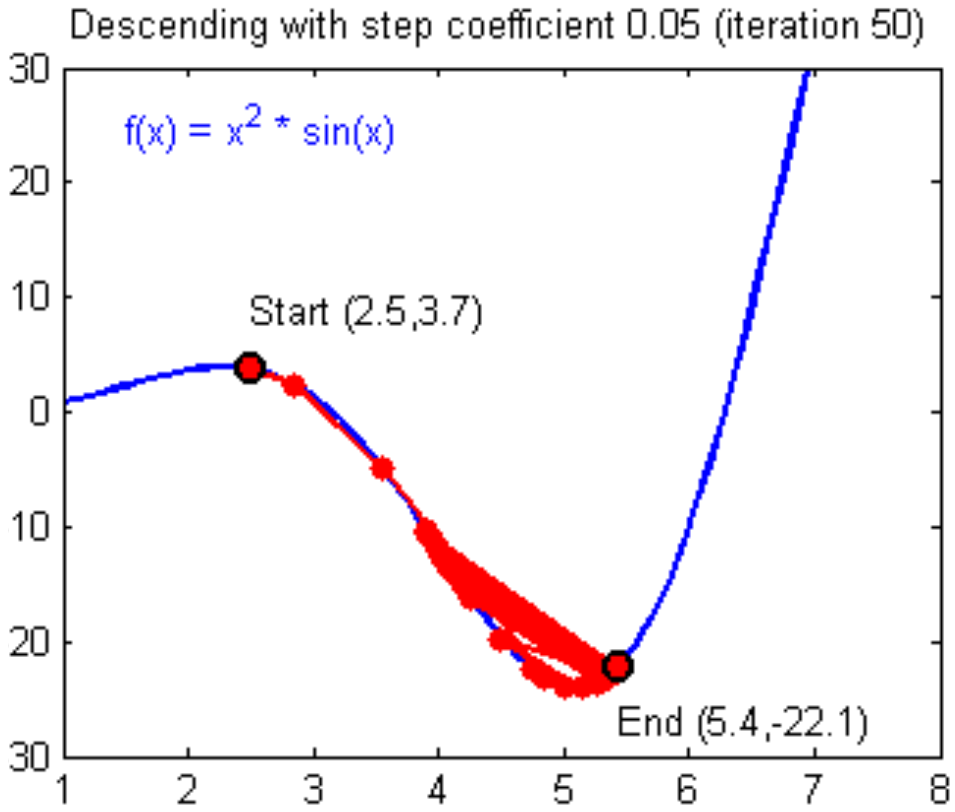


GRADIENT DESCENT

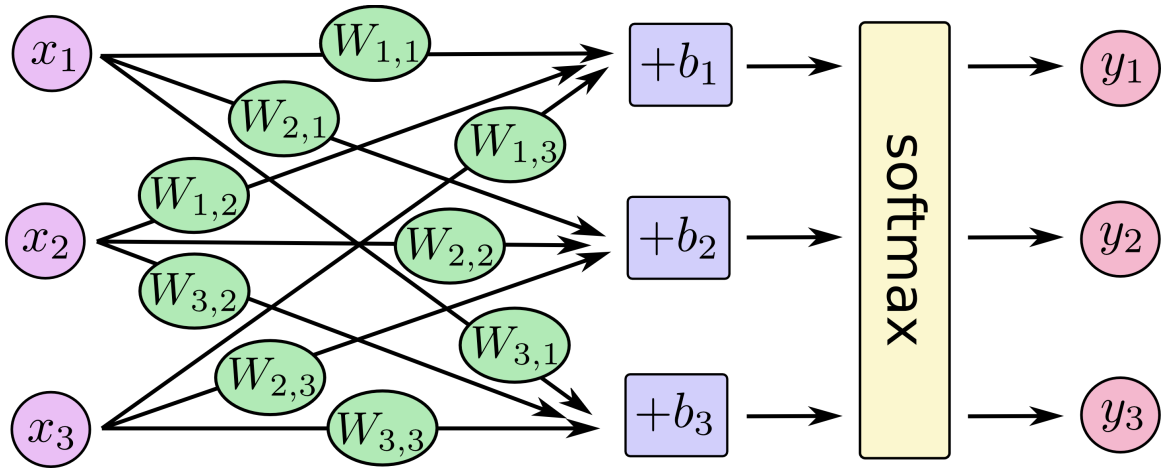
HTTP://WWW.KDNUGGETS.COM/2017/04/SIMPLE-UNDERSTAND-GRADIENT-DESCENT-ALGORITHM.HTML

https://docs.google.com/spreadsheets/d/1TFHcFix5zN5ikqFkJC6mlXg83LxGJAeVJl3BddvYsPM/edit?usp=sharing


Descending with step coefficient 0.005 (iteration 50)
f(x) = x^2 * sin(x)
Start (2.5,3.7)
End (4.9,-23.7)
30
20
10
0
-10
-20
-30
1
2
3
4
5
6
7
8


Descending with step coefficient 0.05 (iteration 50)
f(x) = x^2 * sin(x)
Start (2.5,3.7)
End (5.4,-22.1)
30
20
10
0
10
20
30
1
2
3
4
5
6
7
8

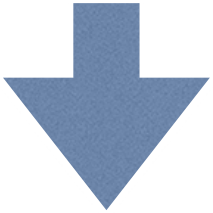
como quantificar "BOM"???



	x					***y***			***y_pred***		
	pixel1	pixel2	pixel3	...	pixel n	cão	gato	rato	cão	gato	rato
	2	133	25	...	77	0	1	0	0,06	0,85	0,09
	3	3	55	...	89	1	0	0	0,77	0,20	0,03
	254	255	253	...	255	0	0	1	0,32	0,15	0,53
	8	8	10	...	21	0	1	0	0,40	0,20	0,30

como quantificar "BOM"???

x					*y*			*y_pred*		
	pixel1	pixel2	pixel3	...	pixel n	cão	gato	rato	cão	gato	rato
	2	133	25	...	77	0	1	0	0,06	0,85	0,09
	3	3	55	...	89	1	0	0	0,77	0,20	0,03
	254	255	253	...	255	0	0	1	0,32	0,15	0,53
	8	8	10	...	21	0	1	0	0,40	0,20	0,30



CROSS-ENTROPY

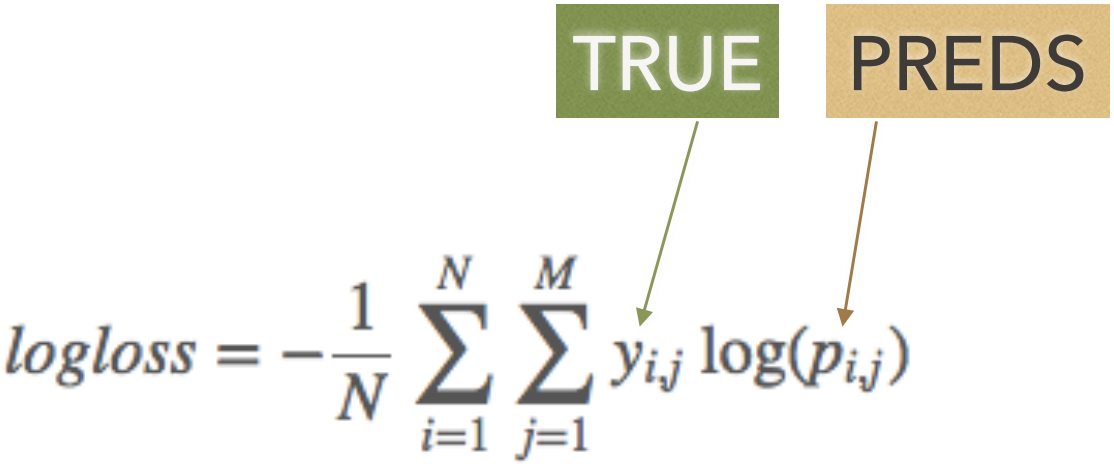
- 2 médicos, Paulo e Georgia prestando o exame do conselho de medicina.
- 2 questões, Q1 e Q2. Cada questão com uma descrição de sintomas com três alternativas de doença: doença A, doença B e doença C.
- O gabarito é
  Q1: doença C
  Q2: doença B
- Paulo lê o enunciado e pensa nas seguintes probabilidades:

	Q1	Q2
doença A	20%	30%
doença B	30%	40%
doença C	50%	30%

  ==========> Paulo acerta 2/2
- Georgia lê o enunciado e pensa nas seguintes probabilidades:

	Q1	Q2
doença A	0%	5%
doença B	10%	95%
doença C	90%	0%

  ==========> Georgia acerta 2/2
- Ambos passam no exame, mas… QUEM É O “MELHOR” MÉDICO?


TRUE
PREDS
$logloss = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{i,j}\log(p_{i,j})$

PREDS

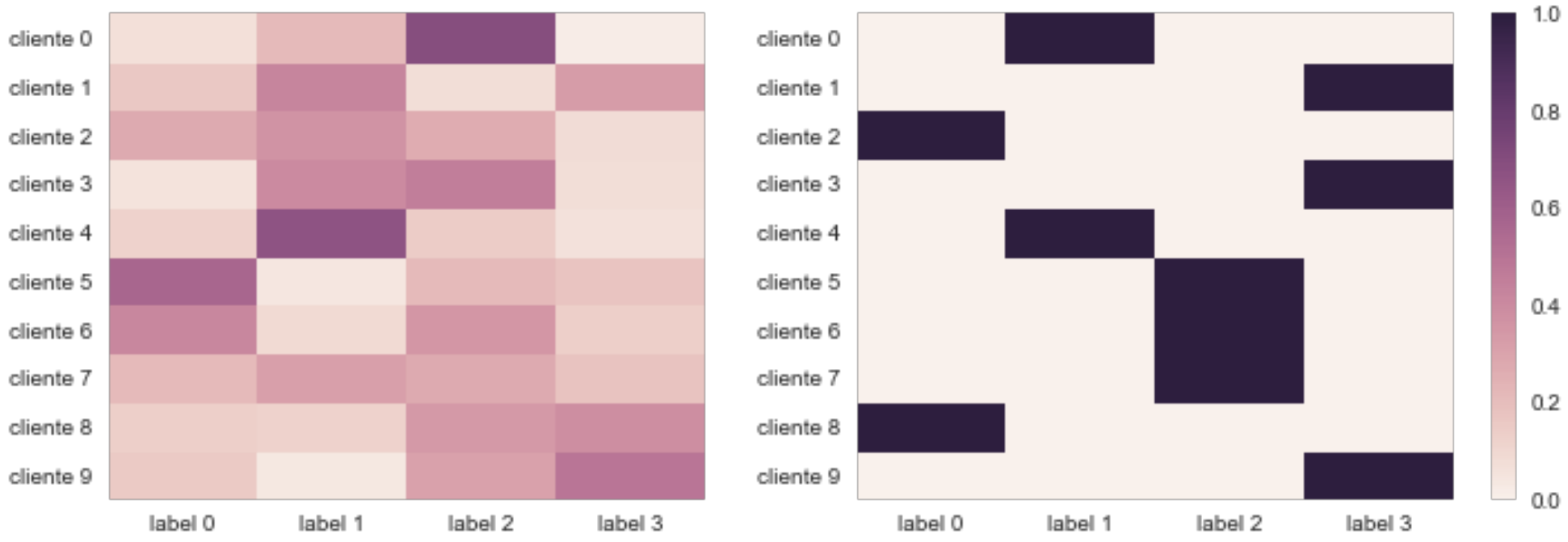
	label 0	label 1	label 2	label 3
cliente 0	0,069	0,213	0,698	0,020
cliente 1	0,164	0,426	0,081	0,329
cliente 2	0,278	0,370	0,269	0,083
cliente 3	0,055	0,405	0,460	0,080
cliente 4	0,124	0,666	0,145	0,066
cliente 5	0,569	0,045	0,214	0,172
cliente 6	0,421	0,091	0,353	0,135
cliente 7	0,215	0,323	0,279	0,182
cliente 8	0,139	0,123	0,347	0,390
cliente 9	0,155	0,036	0,311	0,498

TRUE

	label 0	label 1	label 2	label 3
cliente 0	0	1	0	0
cliente 1	0	0	0	1
cliente 2	1	0	0	0
cliente 3	0	0	0	1
cliente 4	0	1	0	0
cliente 5	0	0	1	0
cliente 6	0	0	1	0
cliente 7	0	0	1	0
cliente 8	1	0	0	0
cliente 9	0	0	0	1

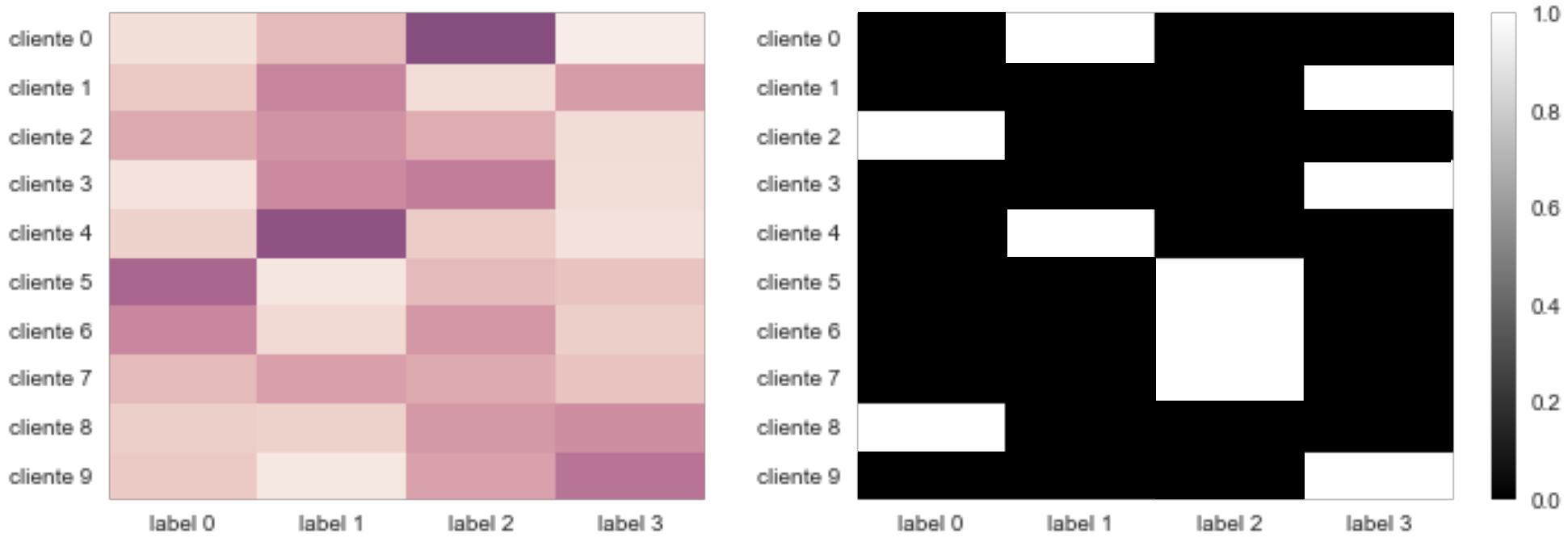
PREDS

TRUE


cliente 0
cliente 1
cliente 2
cliente 3
cliente 4
cliente 5
cliente 6
cliente 7
cliente 8
cliente 9
label 0
label 1
label 2
label 3
cliente 0
cliente 1
cliente 2
cliente 3
cliente 4
cliente 5
cliente 6
cliente 7
cliente 8
cliente 9
label 0
label 1
label 2
label 3
1.0
0.8
0.6
0.4
0.2
0.0

PREDS

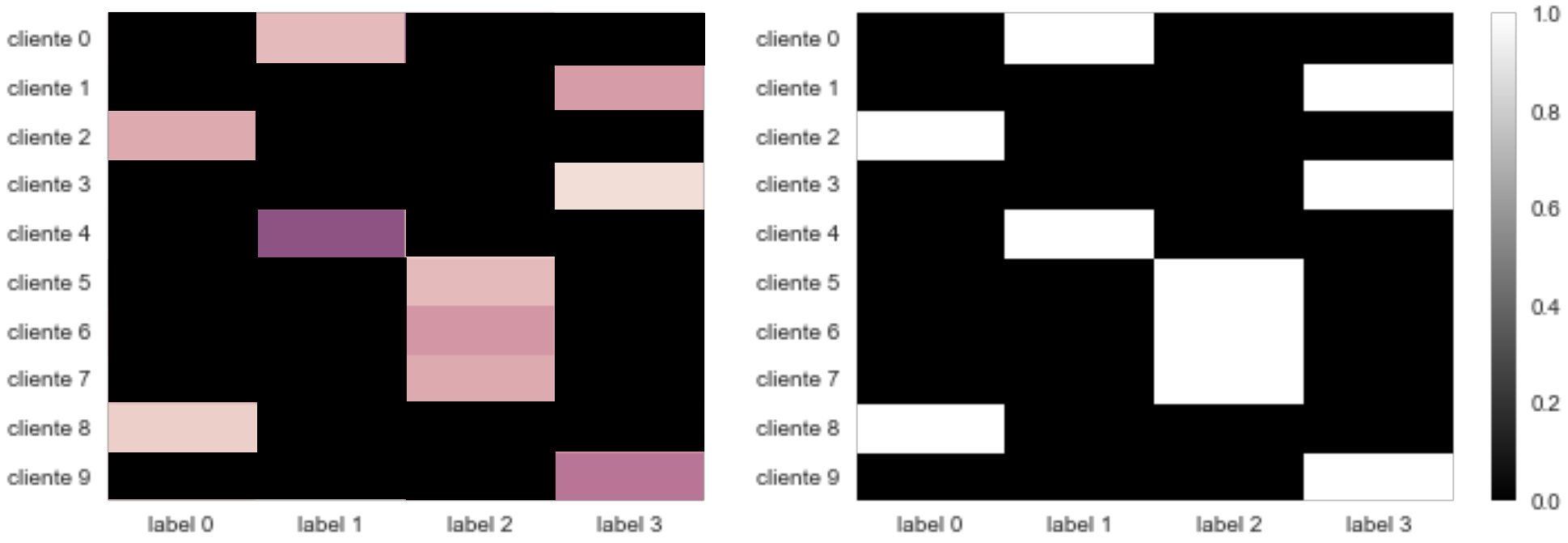
TRUE



$$logloss = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{i,j}\log(p_{i,j})$$

PREDS

TRUE



$$logloss = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{i,j}\log(p_{i,j})$$

terminologia

- loss function
- cost function
- objective function
- Qual a diferença?
  - Loss - uma única observação
  - Cost - todos as observações
  - Objective - uma função a otimizar durante o treinamento
- ref https://stats.stackexchange.com/questions/179026/objective-function-cost-function-loss-function-are-they-the-same-thing

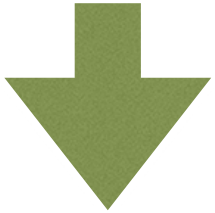
como quantificar "BOM"???

y_gold

cão	gato	rato
0	1	0
1	0	0
0	0	1
0	1	0

y_pred

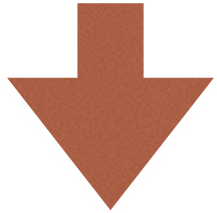
cão	gato	rato
0,06	0,85	0,09
0,77	0,20	0,03
0,32	0,15	0,53
0,40	0,20	0,40



CROSS-ENTROPY MENOR

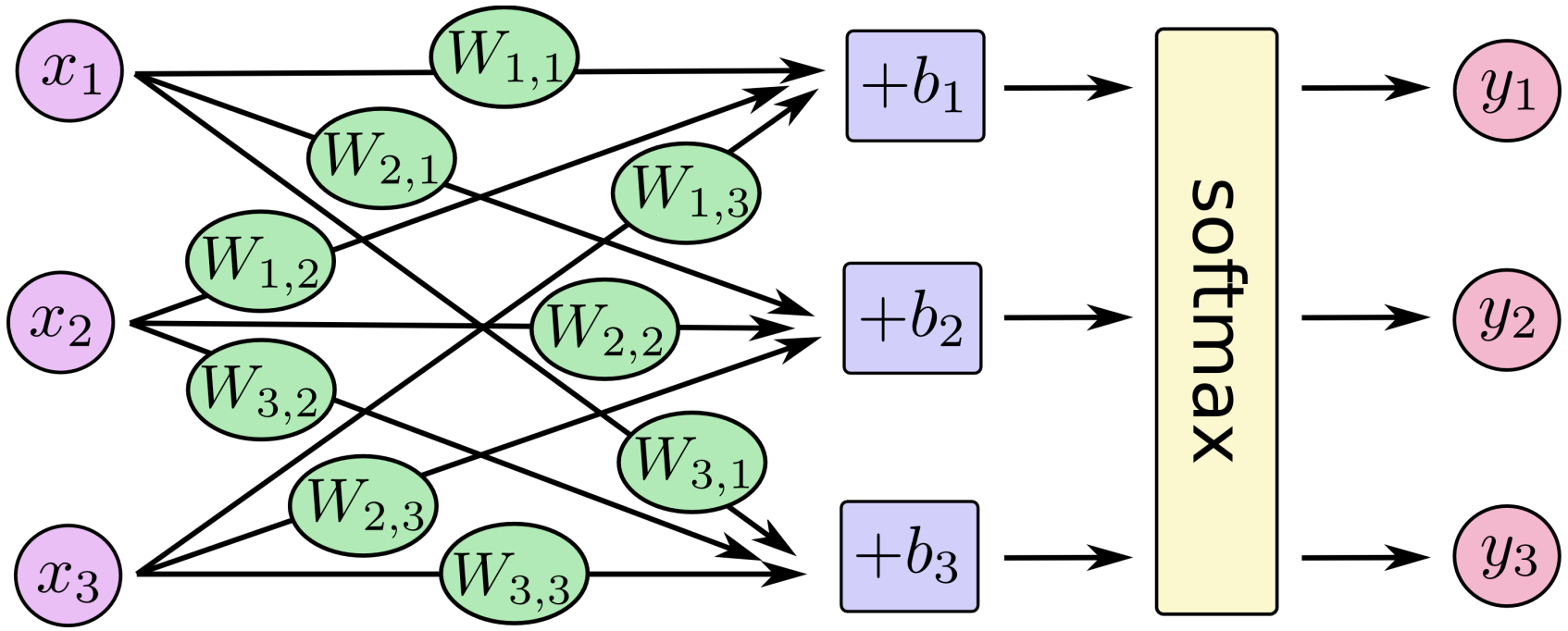
y_pred

cão	gato	rato
0,60	0,15	0,25
0,24	0,20	0,56
0,32	0,03	0,65
0,40	0,15	0,45



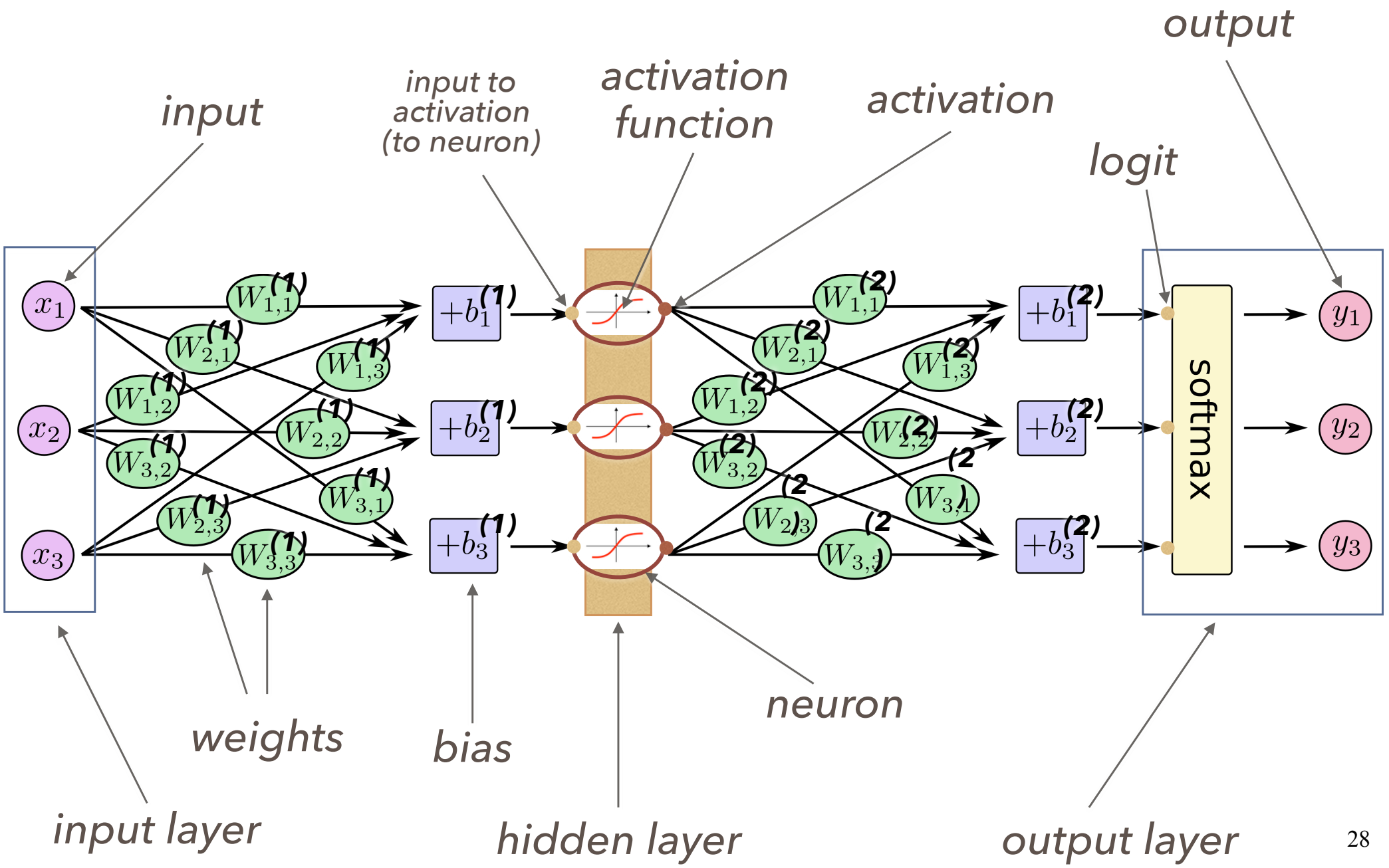
CROSS-ENTROPY MAIOR

MULTINOMIAL LOGISTIC CLASSIFICATION



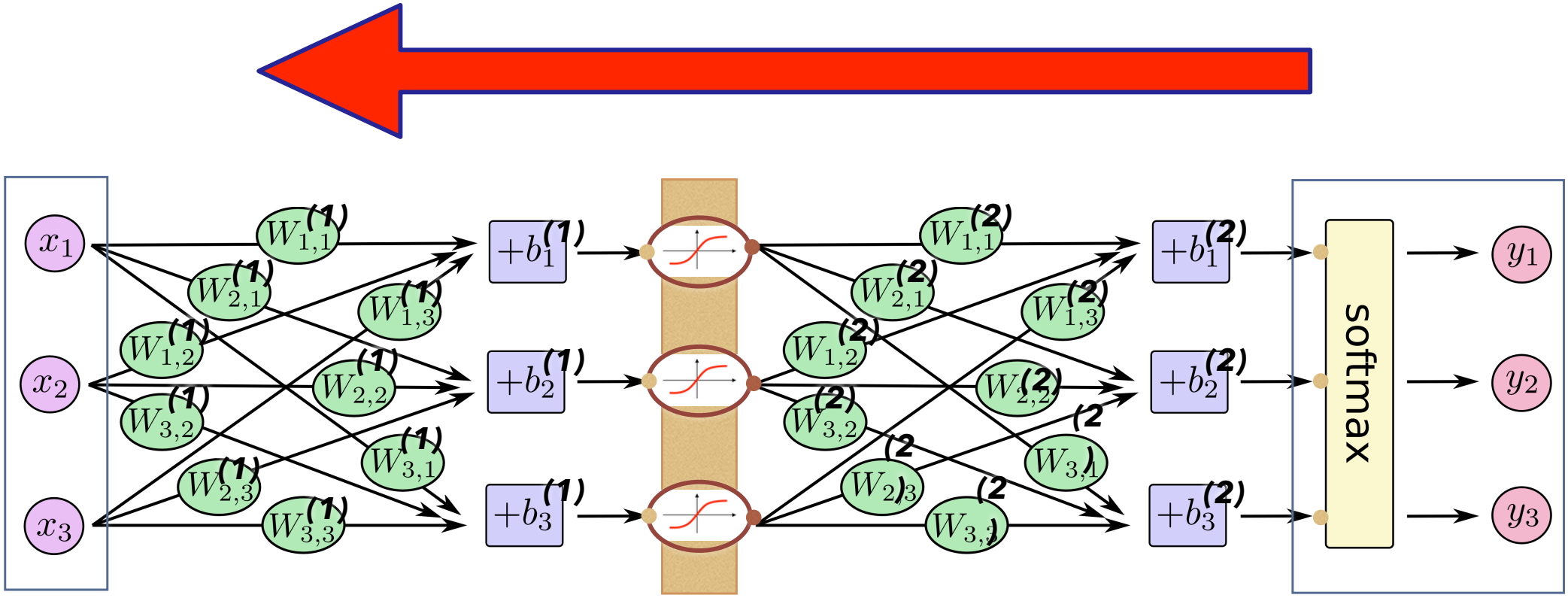
Este setup também é conhecido por
MULTINOMIAL LOGISTIC CLASSIFICATION

terminologia



Backpropagation...

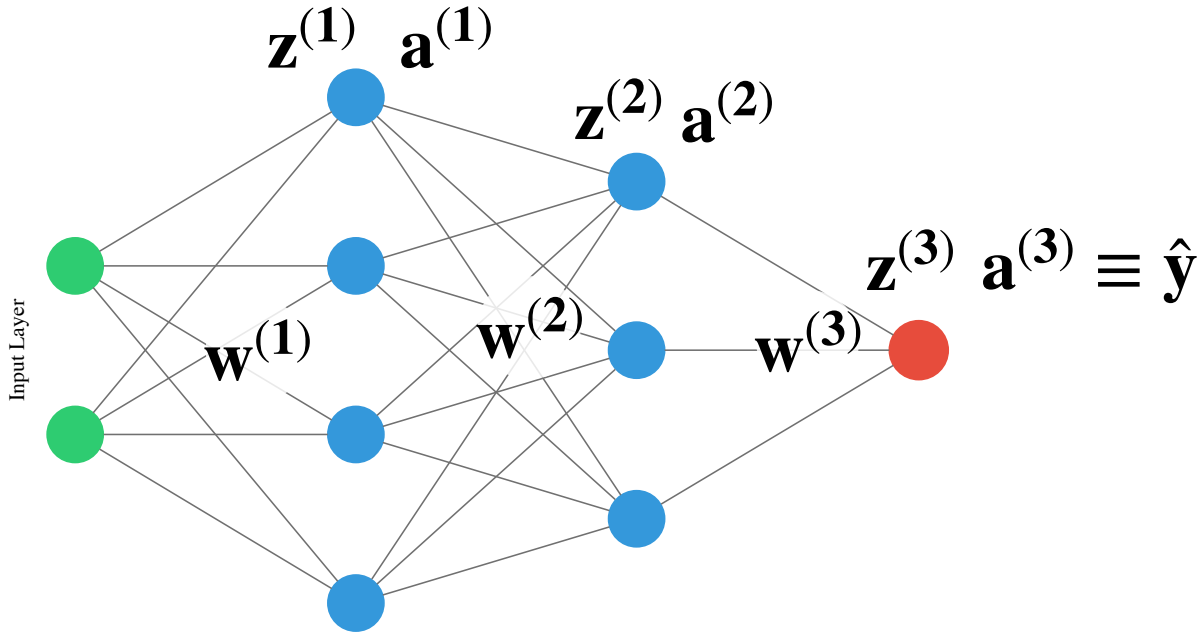
... técnica para obter os gradientes da função de custo (e.g., logloss) em todos os pesos (W & b) da rede



MLP
DEDUÇÃO & IMPLEMENTAÇÃO

forward propagation

obs	x_1	x_2	y
#1	5	2	0
#2	1	0	1
#3	3	1	1

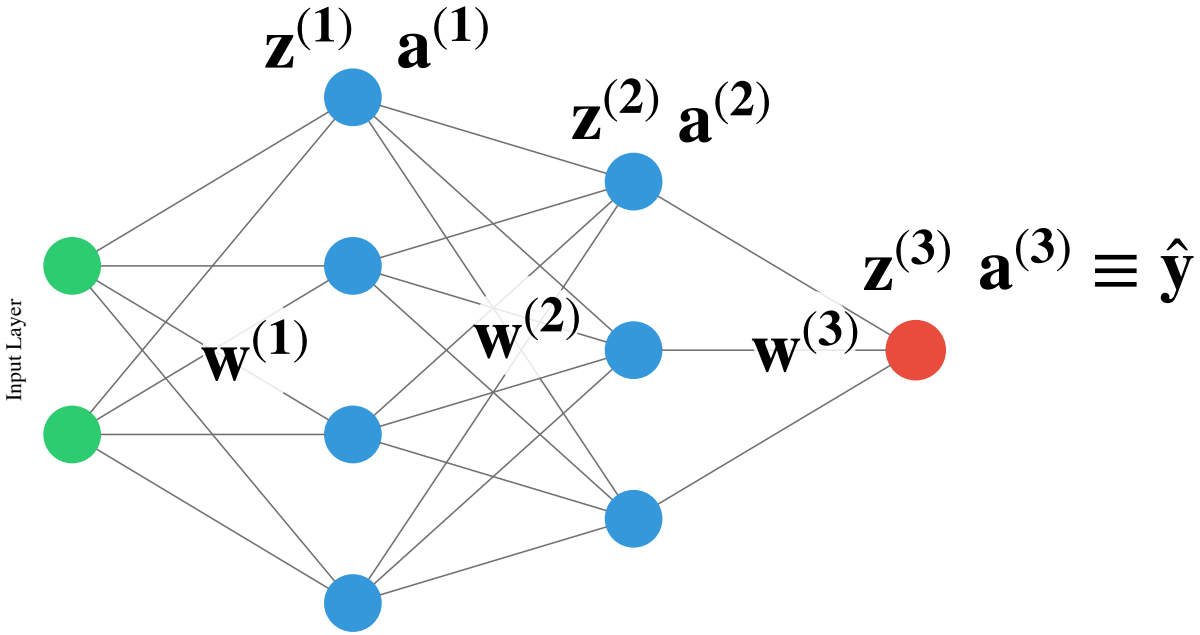


$$\mathbf{z}^{(1)} = \mathbf{X}\mathbf{w}^{(1)} = \begin{bmatrix} 5 & 2 \\ 1 & 0 \\ 3 & 1 \end{bmatrix} \begin{bmatrix} w_{11}^{(1)} & w_{12}^{(1)} & w_{13}^{(1)} & w_{14}^{(1)} \\ w_{21}^{(1)} & w_{22}^{(1)} & w_{23}^{(1)} & w_{24}^{(1)} \end{bmatrix}$$

$$= \begin{bmatrix} 5w_{11}^{(1)} + 2w_{21}^{(1)} & 5w_{12}^{(1)} + 2w_{22}^{(1)} & 5w_{13}^{(1)} + 2w_{23}^{(1)} & 5w_{14}^{(1)} + 2w_{24}^{(1)} \\ 1w_{11}^{(1)} + 0w_{21}^{(1)} & 1w_{12}^{(1)} + 0w_{22}^{(1)} & 1w_{13}^{(1)} + 0w_{23}^{(1)} & 1w_{14}^{(1)} + 0w_{24}^{(1)} \\ 3w_{11}^{(1)} + 1w_{21}^{(1)} & 3w_{12}^{(1)} + 1w_{22}^{(1)} & 3w_{13}^{(1)} + 1w_{23}^{(1)} & 3w_{14}^{(1)} + 1w_{24}^{(1)} \end{bmatrix}$$

forward propagation

obs	x_1	x_2	y
#1	5	2	0
#2	1	0	1
#3	3	1	1

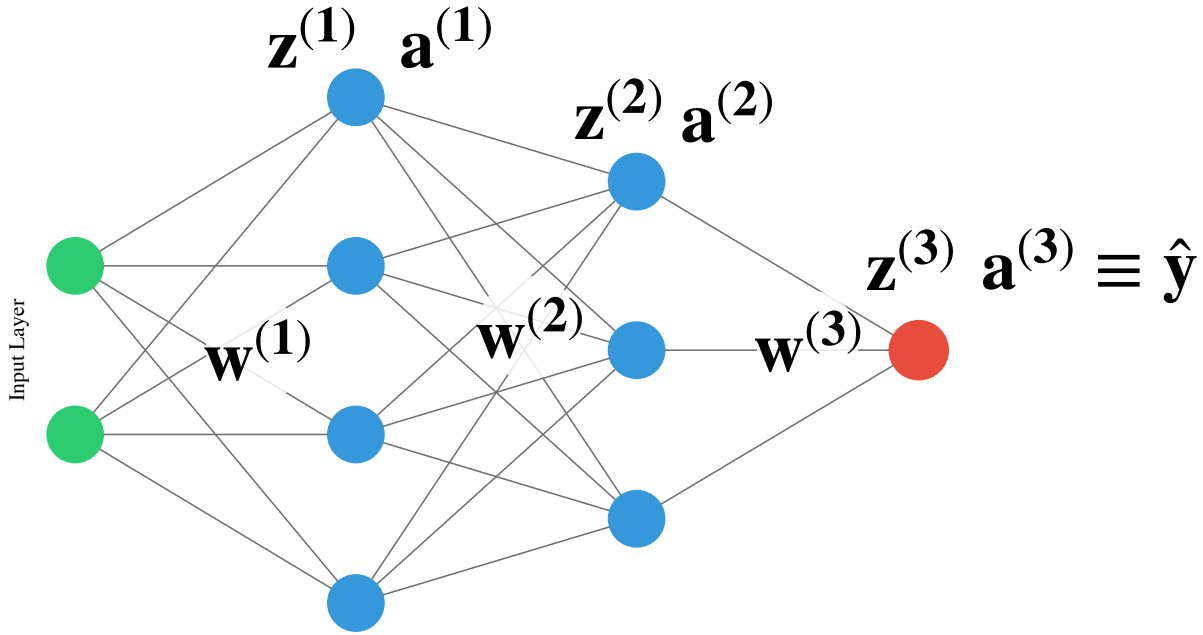


$$\mathbf{z}^{(1)} = \mathbf{X}\mathbf{w}^{(1)} = \begin{bmatrix} 5w_{11}^{(1)} + 2w_{21}^{(1)} & 5w_{12}^{(1)} + 2w_{22}^{(1)} & 5w_{13}^{(1)} + 2w_{23}^{(1)} & 5w_{14}^{(1)} + 2w_{24}^{(1)} \\ 1w_{11}^{(1)} + 0w_{21}^{(1)} & 1w_{12}^{(1)} + 0w_{22}^{(1)} & 1w_{13}^{(1)} + 0w_{23}^{(1)} & 1w_{14}^{(1)} + 0w_{24}^{(1)} \\ 3w_{11}^{(1)} + 1w_{21}^{(1)} & 3w_{12}^{(1)} + 1w_{22}^{(1)} & 3w_{13}^{(1)} + 1w_{23}^{(1)} & 3w_{14}^{(1)} + 1w_{24}^{(1)} \end{bmatrix}$$

$$\mathbf{a}^{(1)} = \sigma(\mathbf{z}^{(1)}) = \begin{bmatrix} \sigma(5w_{11}^{(1)}) + 2w_{21}^{(1)}) & \sigma(5w_{12}^{(1)} + 2w_{22}^{(1)}) & \sigma(5w_{13}^{(1)} + 2w_{23}^{(1)}) & \sigma(5w_{14}^{(1)} + 2w_{24}^{(1)}) \\ \sigma(1w_{11}^{(1)}) + 0w_{21}^{(1)}) & \sigma(1w_{12}^{(1)} + 0w_{22}^{(1)}) & \sigma(1w_{13}^{(1)} + 0w_{23}^{(1)}) & \sigma(1w_{14}^{(1)} + 0w_{24}^{(1)}) \\ \sigma(3w_{11}^{(1)}) + 1w_{21}^{(1)}) & \sigma(3w_{12}^{(1)} + 1w_{22}^{(1)}) & \sigma(3w_{13}^{(1)} + 1w_{23}^{(1)}) & \sigma(3w_{14}^{(1)} + 1w_{24}^{(1)}) \end{bmatrix}$$

forward propagation

obs	x_1	x_2	y
#1	5	2	0
#2	1	0	1
#3	3	1	1



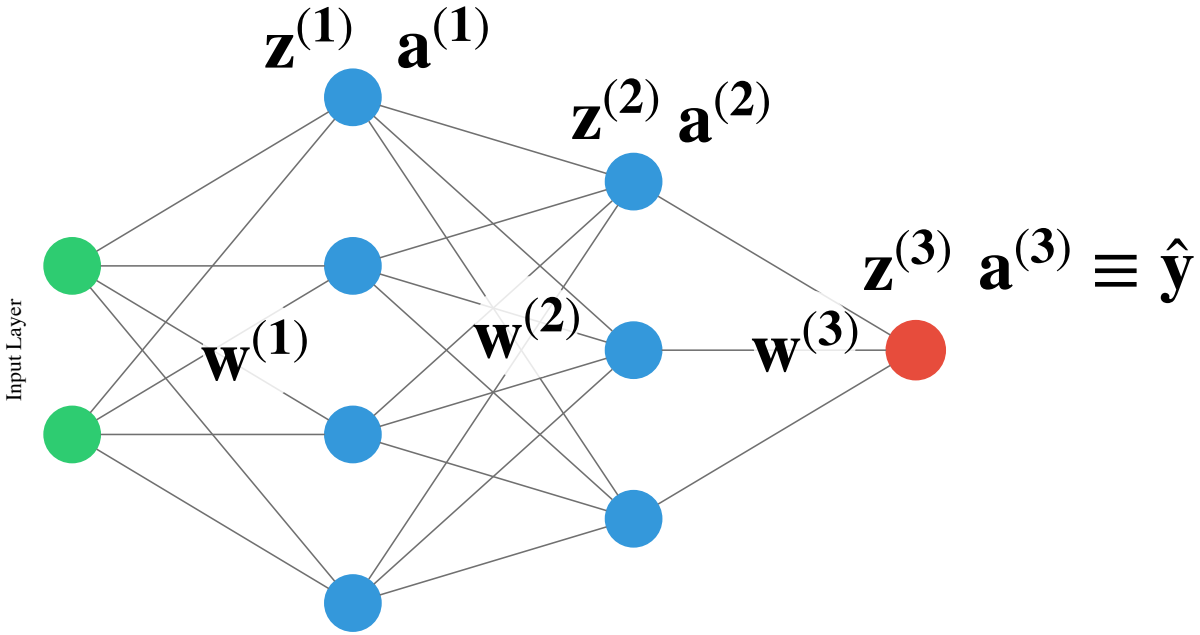
$$\mathbf{z}^{(1)} = \mathbf{X}\mathbf{w}^{(1)} = \begin{bmatrix} 5 & 2 \\ 1 & 0 \\ 3 & 1 \end{bmatrix} \begin{bmatrix} w_{11}^{(1)} & w_{12}^{(1)} & w_{13}^{(1)} & w_{14}^{(1)} \\ w_{21}^{(1)} & w_{22}^{(1)} & w_{23}^{(1)} & w_{24}^{(1)} \end{bmatrix}$$

$$\mathbf{z}^{(1)} = \mathbf{X}\mathbf{w}^{(1)} = \begin{bmatrix} \bullet & \bullet \\ \bullet & \bullet \\ \bullet & \bullet \end{bmatrix} \begin{bmatrix} \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet \end{bmatrix} = \begin{bmatrix} \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet \end{bmatrix}$$

(3 obs x 2 sinais) (3 obs x 4 sinais)

forward propagation

obs	x_1	x_2	y
#1	5	2	0
#2	1	0	1
#3	3	1	1



$$\mathbf{z}^{(1)} = \mathbf{X}\mathbf{w}^{(1)} = \begin{bmatrix} \cdot & \cdot \\ \cdot & \cdot \\ \cdot & \cdot \end{bmatrix} \begin{bmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{bmatrix} = \begin{bmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{bmatrix}$$

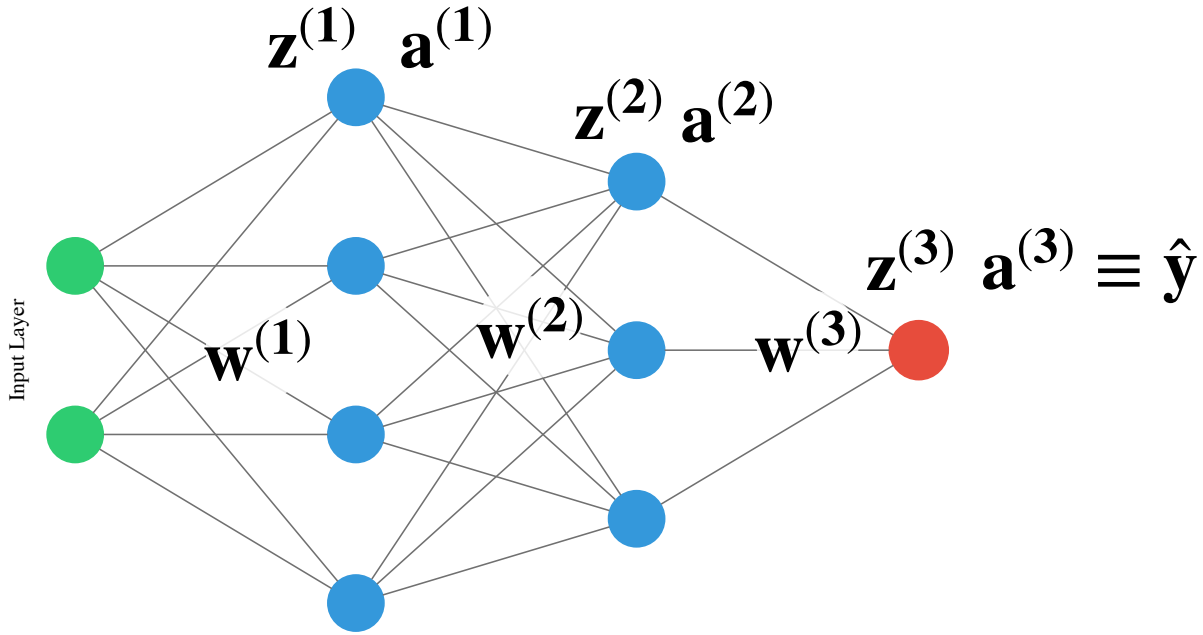
(3 obs x 2 sinais) (3 obs x 4 sinais)

$$\mathbf{a}^{(1)} = \sigma(\mathbf{z}^{(1)}) = \sigma\left(\begin{bmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{bmatrix}\right) = \begin{bmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{bmatrix}$$

(3 obs x 4 sinais) (3 obs x 4 sinais)

forward propagation

obs	x_1	x_2	y
#1	5	2	0
#2	1	0	1
#3	3	1	1



$$\mathbf{z}^{(2)} = \mathbf{a}^{(1)}\mathbf{w}^{(2)} = \begin{bmatrix} \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet \end{bmatrix} \begin{bmatrix} \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet \end{bmatrix} = \begin{bmatrix} \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet \end{bmatrix}$$

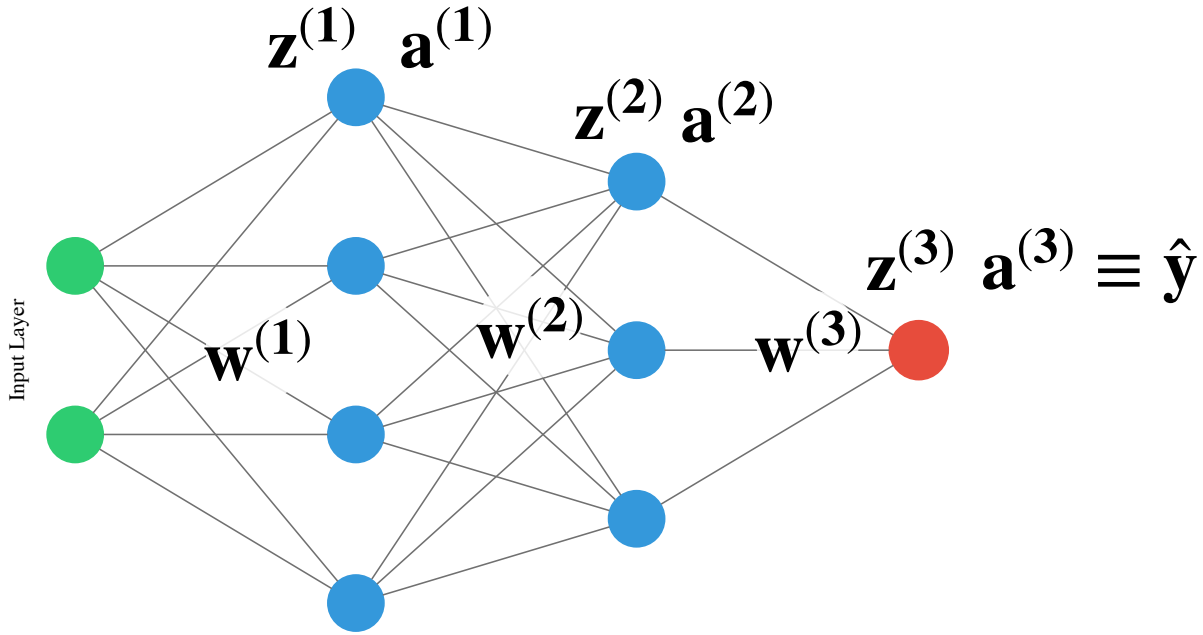
(3 obs x 3 sinais)

$$\mathbf{a}^{(2)} = \sigma(\mathbf{z}^{(2)}) = \sigma\left(\begin{bmatrix} \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet \end{bmatrix} \right) = \begin{bmatrix} \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet \end{bmatrix}$$

(3 obs x 3 sinais) (3 obs x 3 sinais)

forward propagation

obs	x_1	x_2	y
#1	5	2	0
#2	1	0	1
#3	3	1	1



$$\mathbf{z}^{(3)} = \mathbf{a}^{(2)}\mathbf{w}^{(3)} = \begin{bmatrix} \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet \end{bmatrix} \begin{bmatrix} \bullet \\ \bullet \\ \bullet \end{bmatrix} = \begin{bmatrix} \bullet \\ \bullet \\ \bullet \end{bmatrix}$$

(3 obs x 1 sinal)

$$\mathbf{a}^{(3)} = \sigma(\mathbf{z}^{(3)}) = \sigma\left(\begin{bmatrix} \bullet \\ \bullet \\ \bullet \end{bmatrix} \right) = \begin{bmatrix} \bullet \\ \bullet \\ \bullet \end{bmatrix}$$

(3 obs x 1 sinal) (3 obs x 1 sinal)

forward propagation

$$\mathbf{z}^{(1)} = \mathbf{X}\mathbf{w}^{(1)} =$$

(3 obs x 2 sinais) (3 obs x 4 sinais)

$$\mathbf{a}^{(1)} = \sigma(\mathbf{z}^{(1)}) = \sigma(\quad) =$$

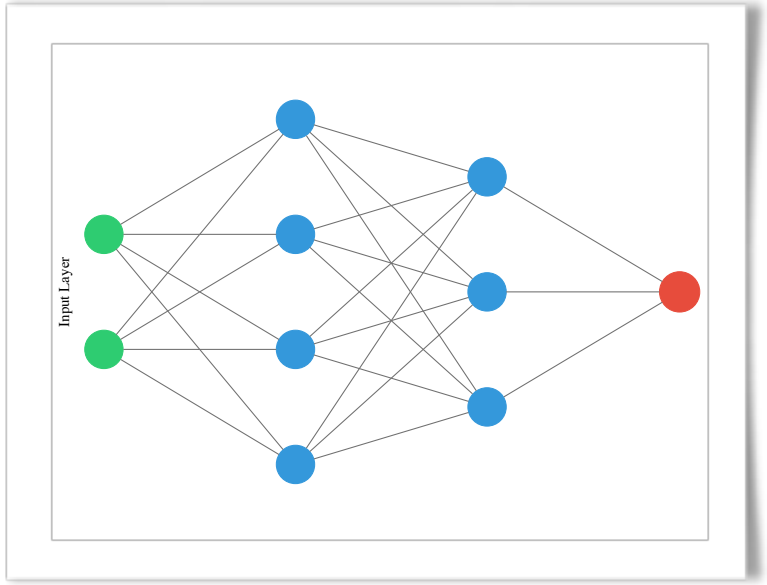
(3 obs x 4 sinais) (3 obs x 4 sinais)

$$\mathbf{z}^{(2)} = \mathbf{a}^{(1)}\mathbf{w}^{(2)} =$$

(3 obs x 3 sinais)

$$\mathbf{a}^{(2)} = \sigma(\mathbf{z}^{(2)}) = \sigma(\quad) =$$

(3 obs x 3 sinais) (3 obs x 3 sinais)



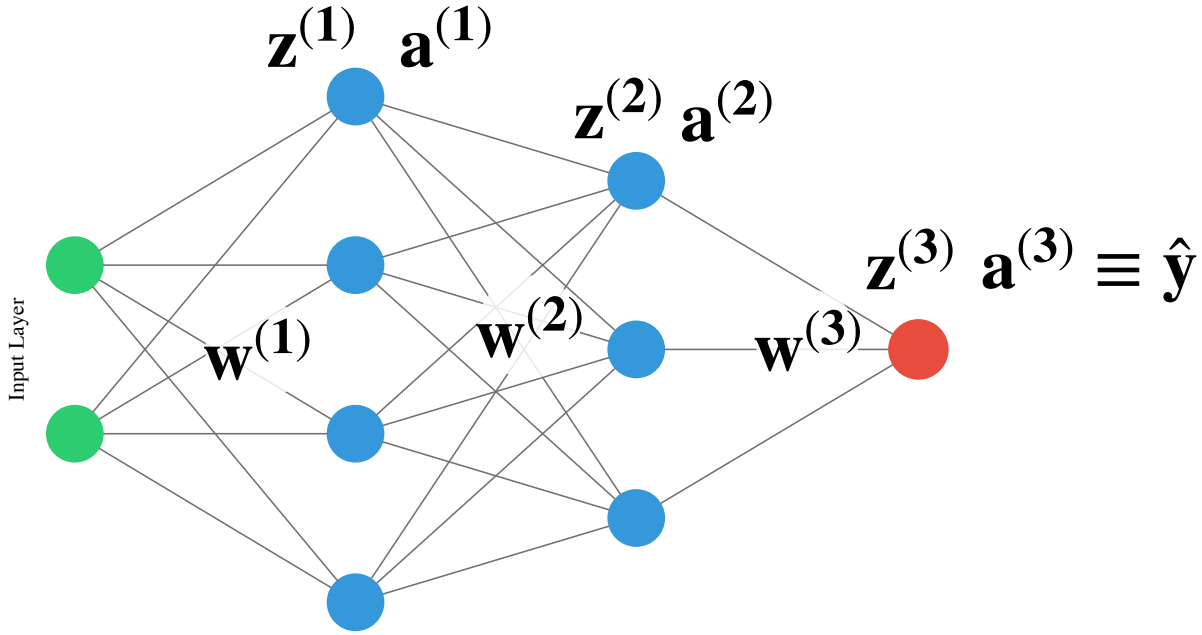
$$\mathbf{z}^{(3)} = \mathbf{a}^{(2)}\mathbf{w}^{(3)} =$$

(3 obs x 1 sinal)

$$\mathbf{a}^{(3)} = \sigma(\mathbf{z}^{(3)}) = \sigma(\quad) =$$

(3 obs x 1 sinal) (3 obs x 1 sinal)

função de custo

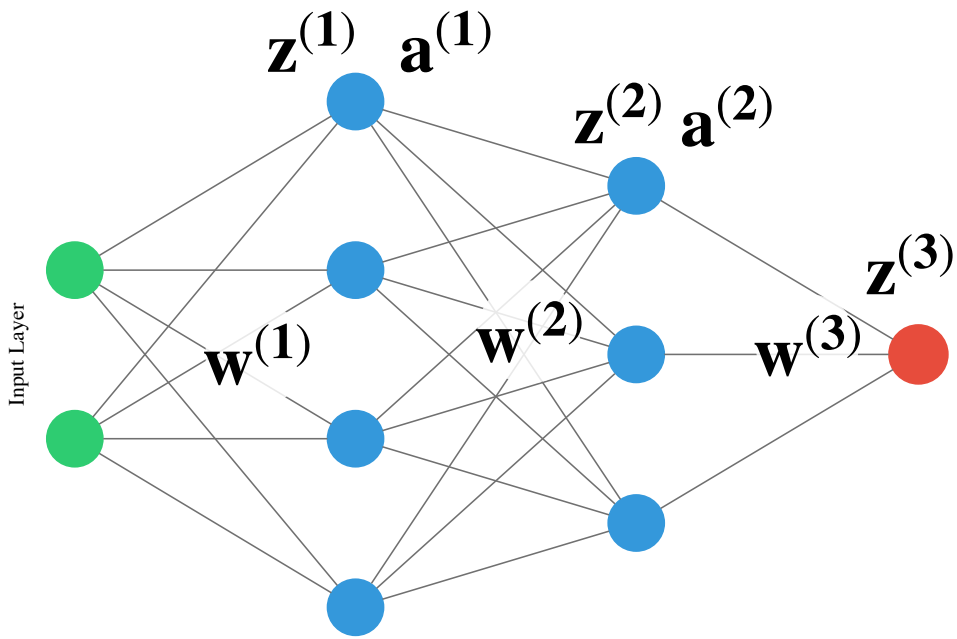


obs	x_1	x_2	y	$\hat{y}$
#1	5	2	**0**	**0.2**
#2	1	0	**1**	**0.9**
#3	3	1	**1**	**0.6**

$$\mathbf{J} = \sum_{i=1}^{3} -(y_i \log \hat{y}_i + (1 - y_i)\log(1 - \hat{y}_i))$$

$$\mathbf{J} = \sum_{i=1}^{3} l_i$$

função de custo



$$\mathbf{J} = \sum_{i=1}^{3} -(y_i \log \hat{y}_i + (1 - y_i)\log(1 - \hat{y}_i))$$

obs	x_1	x_2	y	$\hat{y}$
#1	5	2	**0**	**0.2**
#2	1	0	**1**	**0.9**
#3	3	1	**1**	**0.6**

$$\frac{\partial \mathbf{J}}{\partial \mathbf{w}^{(3)}} = - \begin{bmatrix} a_{11}^{(2)} & a_{21}^{(2)} & a_{31}^{(2)} \\ a_{12}^{(2)} & a_{22}^{(2)} & a_{32}^{(2)} \\ a_{13}^{(2)} & a_{23}^{(2)} & a_{33}^{(2)} \end{bmatrix} \begin{bmatrix} y_1 - \hat{y}_1 \\ y_2 - \hat{y}_2 \\ y_3 - \hat{y}_3 \end{bmatrix} = - \mathbf{a}^{(2)\mathbf{T}}(\mathbf{y} - \hat{\mathbf{y}}) = \mathbf{a}^{(2)\mathbf{T}}\delta^{(3)},$$

(3 sinais x 3 obs) (3 obs x 1 sinal)

onde $\delta^{(3)} \triangleq -(\mathbf{y} - \hat{\mathbf{y}})$

gradientes - back propagation

$$\frac{\partial \mathbf{J}}{\partial \mathbf{w}^{(3)}} = -\mathbf{a}^{(2)^{\mathbf{T}}}(\mathbf{y} - \hat{\mathbf{y}}) \qquad = \mathbf{a}^{(2)^{\mathbf{T}}}\delta^{(3)}$$

$$\frac{\partial \mathbf{J}}{\partial \mathbf{w}^{(2)}} = \mathbf{a}^{(1)^{\mathbf{T}}}\left(\delta^{(3)}\mathbf{w}^{(3)^{\mathbf{T}}} \odot \mathbf{a}^{(2)} \odot (1 - \mathbf{a}^{(2)})\right) \qquad = \mathbf{a}^{(1)^{\mathbf{T}}}\delta^{(2)}$$

$$\frac{\partial \mathbf{J}}{\partial \mathbf{w}^{(1)}} = \mathbf{X}^{\mathbf{T}}\left(\delta^{(2)}\mathbf{w}^{(2)^{\mathbf{T}}} \odot \mathbf{a}^{(1)} \odot (1 - \mathbf{a}^{(1)})\right) \qquad = \mathbf{X}^{\mathbf{T}}\delta^{(1)}$$

gradientes

$$\frac{\partial J}{\partial w^{(3)}} = -\,a^{(2)^T}(y - \hat{y}) = a^{(2)^T}\delta^{(3)} = \underset{\text{(3 sinais x 3 obs)}}{\begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix}} \underset{\text{(3 obs x 1 sinal)}}{\begin{bmatrix} \cdot \\ \cdot \\ \cdot \end{bmatrix}} = \underset{\text{(3 sinais)}}{\begin{bmatrix} \cdot \\ \cdot \\ \cdot \end{bmatrix}}$$

$$\frac{\partial J}{\partial w^{(2)}} = a^{(1)^T}\left(\delta^{(3)} w^{(3)^T} \odot a^{(2)} \odot (1 - a^{(2)})\right)$$

$$= \begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix} \left(\begin{bmatrix} \cdot \\ \cdot \\ \cdot \end{bmatrix} \begin{bmatrix} \cdot & \cdot & \cdot \end{bmatrix} \odot \begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix} \odot \left(1 - \begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix} \right) \right)$$

$$\frac{\partial J}{\partial w^{(1)}} = X^T \left(\delta^{(2)} w^{(2)^T} \odot a^{(1)} \odot (1 - a^{(1)})\right)$$

$$= \begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix} \left(\begin{bmatrix} \cdot & \cdot \\ \cdot & \cdot \\ \cdot & \cdot \end{bmatrix} \begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix} \odot \begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix} \odot \left(1 - \begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix} \right) \right)$$

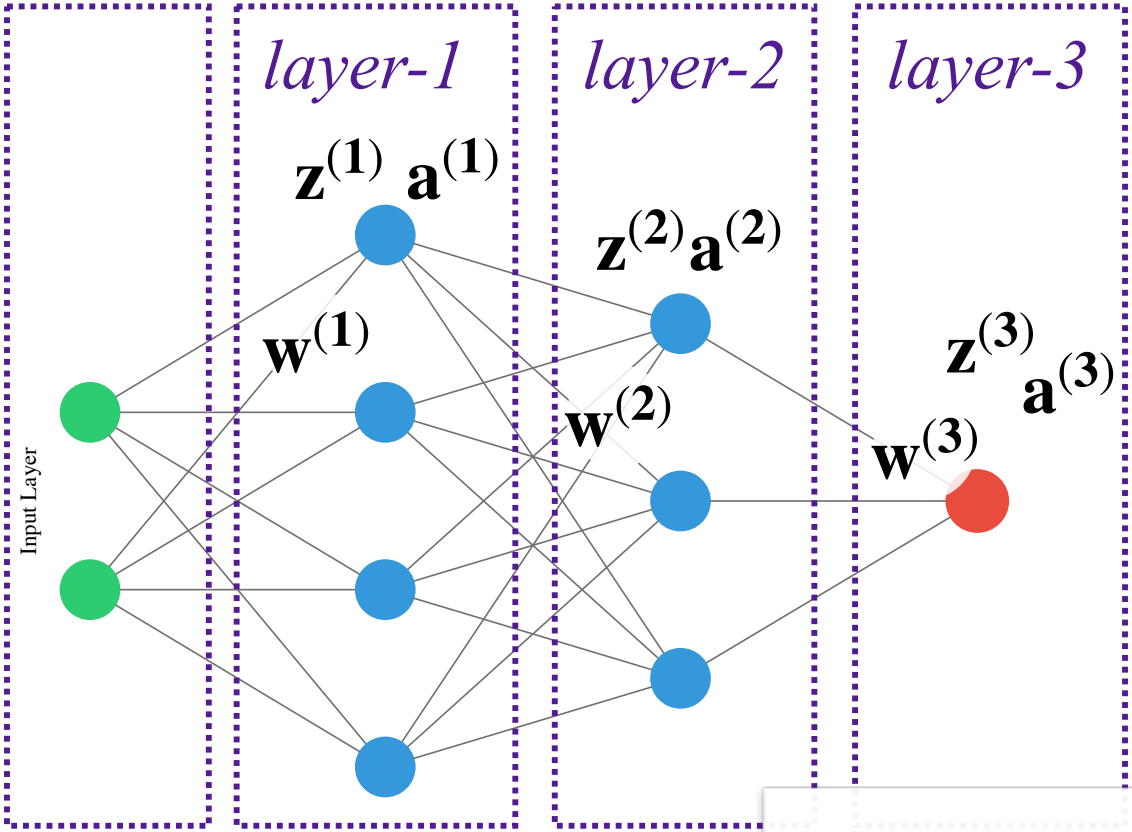
gradiente descendente - máximo declive

$$\mathbf{w}^{(3)} \leftarrow \mathbf{w}^{(3)} - \rho \frac{\partial \mathbf{J}}{\partial \mathbf{w}^{(3)}}$$

$$\mathbf{w}^{(2)} \leftarrow \mathbf{w}^{(2)} - \rho \frac{\partial \mathbf{J}}{\partial \mathbf{w}^{(2)}}$$

$$\mathbf{w}^{(1)} \leftarrow \mathbf{w}^{(1)} - \rho \frac{\partial \mathbf{J}}{\partial \mathbf{w}^{(1)}}$$

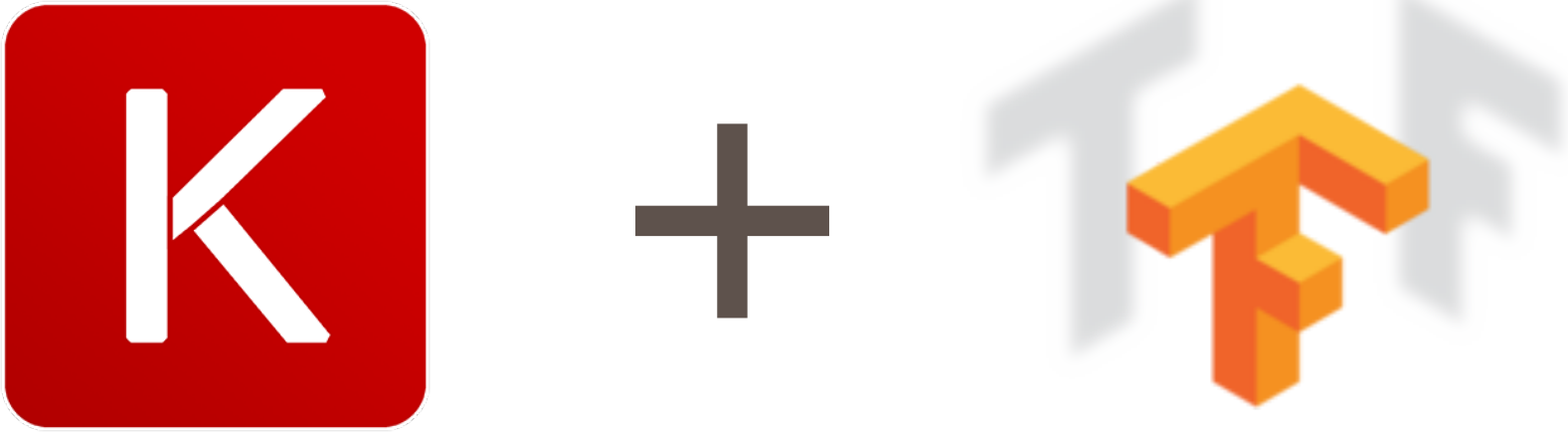
implementação



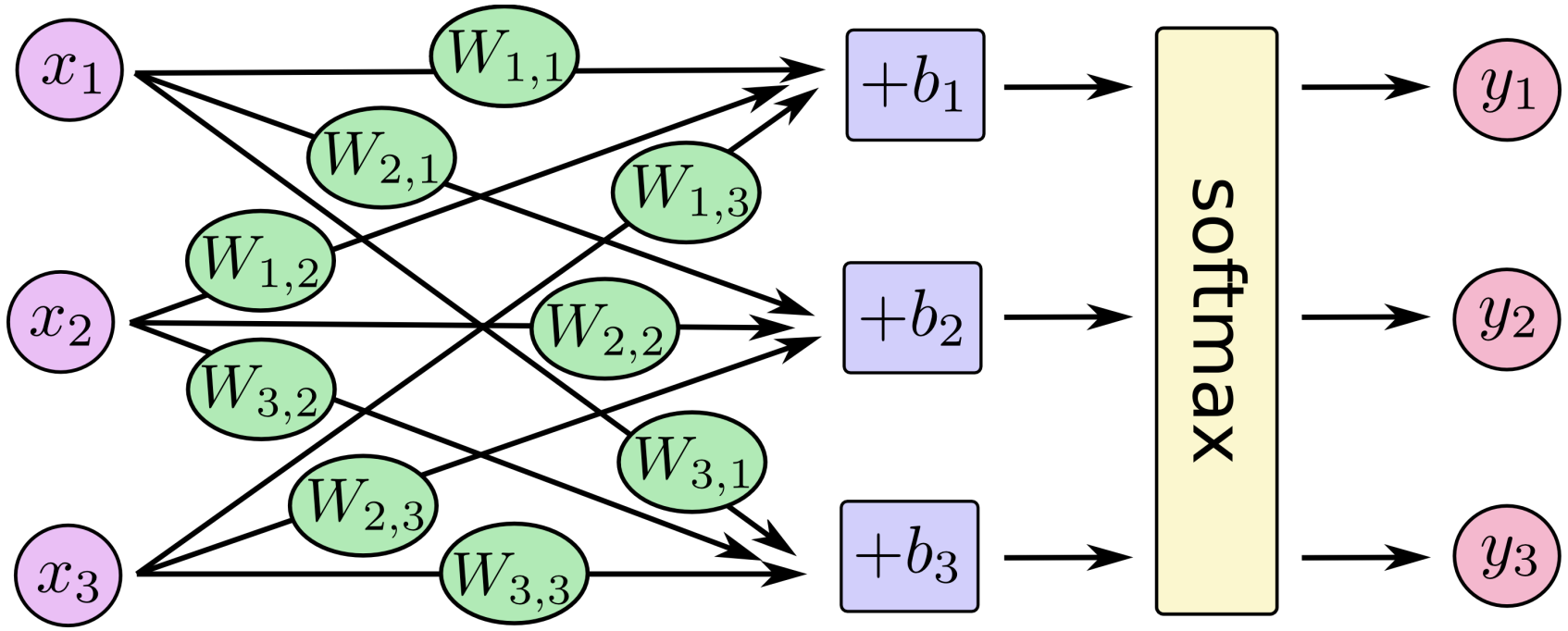

```
class Layer(object):
    def __init__(self, n_prev, n_out):
        """
        n_prev: # unids da camada anterior
        n_out : # unids da camada
        """
```

```
class Network(object):
    def __init__(self, input_dim,
                 hidden_layer_dims,
                 output_dim = 1):
        """
        input_dim : dim entrada
        hidden_layer_dims : lista com dims
                            camadas escondidas
        output_dim : dim ultima camada
        """
```

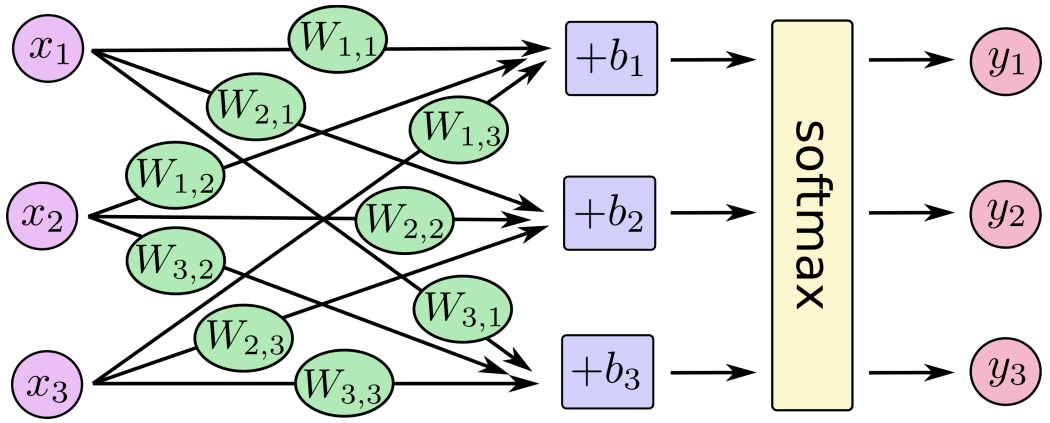
frameworks deep learning



o que precisamos definir?

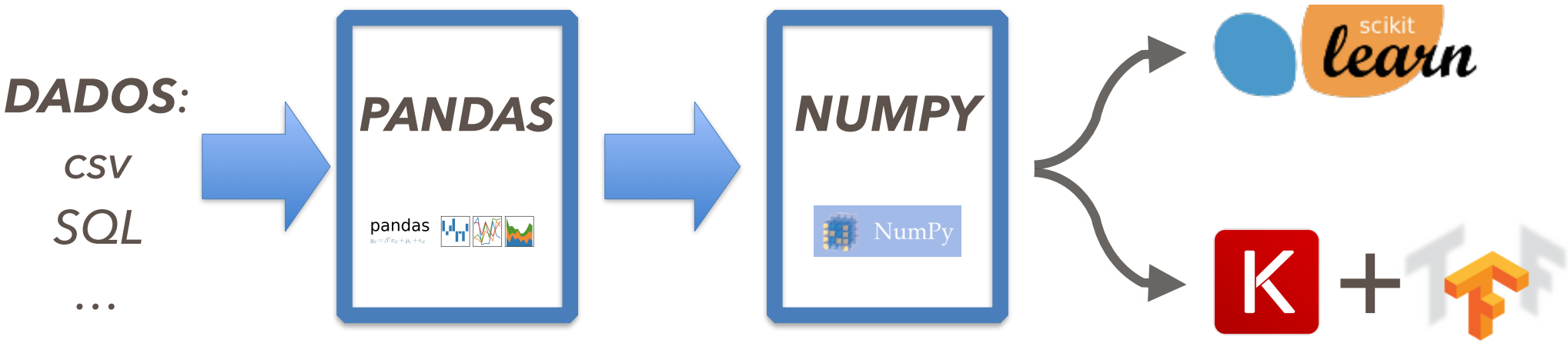


parâmetros e variáveis BASICOS



- arquitetura:
  - dimensão da entrada
  - dimensão da saída (numero de classes)
  - dimensões escondidas - # de pesos & bias
  - função de ativação
- operação:
  - tamanho do batch
  - qtd de epochs
- objetivos:
  - função de custo & métrica
  - otimizador

ferramentas Python



indices
slicing, indexing
limpeza
get_dummies
merge, concat
groupby

algebra linear
obj. multi-dimensionais
multiplicação matrizes
prod. interno
transform. linear

roteiro keras

- importar módulos
  - ➡ `Sequential, Dense, Dropout,…`
- processar X
  - ➡ `reshape(. . . .)`
  - ➡ `astype('. .')`
  - ➡ `Minmaxscaler()`
- processar y
  - ➡ `y: to_categorical(. ., 10).  # 10 labels`
- criar modelos
  - ➡ `model = Sequential()`
  - ➡ `model.add(. . .)`
- compilar
  - ➡ `model.compile(loss = '. .', optimiser = '. .', metrics = ['. .'])`
- treinar
  - ➡ `model.fit(X, y, batch_size = .., epochs = .., verbose = ..)`
- predizer ou avaliar
  - ➡ `model.predict(X_test)`
  - ➡ `model.evaluate(X_test, Y_test)`

roteiro keras

```
def run_keras(X_train, X_test, y_train, y_test):

    from keras import Sequential
    from keras.layers import Dense
    from keras.optimizers import SGD

    model = Sequential()
    model.add(Dense(10, activation='sigmoid', input_dim = X_train.shape[1]))
    model.add(Dense(5, activation='sigmoid'))
    model.add(Dense(1, activation='sigmoid'))
    sgd = SGD(lr = 0.5)
    model.compile(optimizer = sgd,
                  loss='binary_crossentropy',
                  metrics=['accuracy'])

    model.fit(x = X_train, y = y_train, epochs = 1000, batch_size = len(X_train),
              validation_data = (X_test, y_test), verbose = 0)

    y_pred_kr = model.predict(X_test)

    print('Log loss - Keras:', log_loss(y_test, y_pred_kr))

    return model
```

roteiro keras

```
In [1]: model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 10)                210
_________________________________________________________________
dense_2 (Dense)              (None, 5)                 55
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 6
=================================================================
Total params: 271
Trainable params: 271
Non-trainable params: 0
_________________________________________________________________
```

```
In [2]: X_train.shape[1]
Out[2]: 20
```

LSTM

LSTM

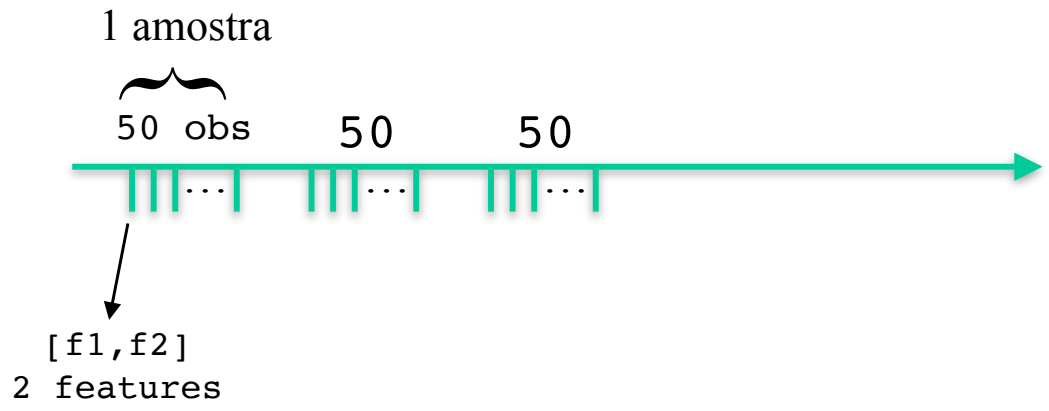
- `model.add(LSTM(32), input_shape = (50, 2))`
- input (s, t, f) precisa ser 3D
  - ➡ s: # amostras no batch
  - ➡ t: # observações = # passos no tempo
  - ➡ f: # features numa observação
- precisa ser 3D
  - `[[[1], [1], [0], [2]],`
    `[[1], [1], [0], [2]]]`
  - 2 amostras
    4 observações em cada amostra
    1 feature em cada observação

LSTM

- precisa ser 3D
  - [[[1, 1, 0, 2]],
    [[1, 1, 0, 2]]]
  - 2 amostras
    1 observação em cada amostra
    4 features em cada observação

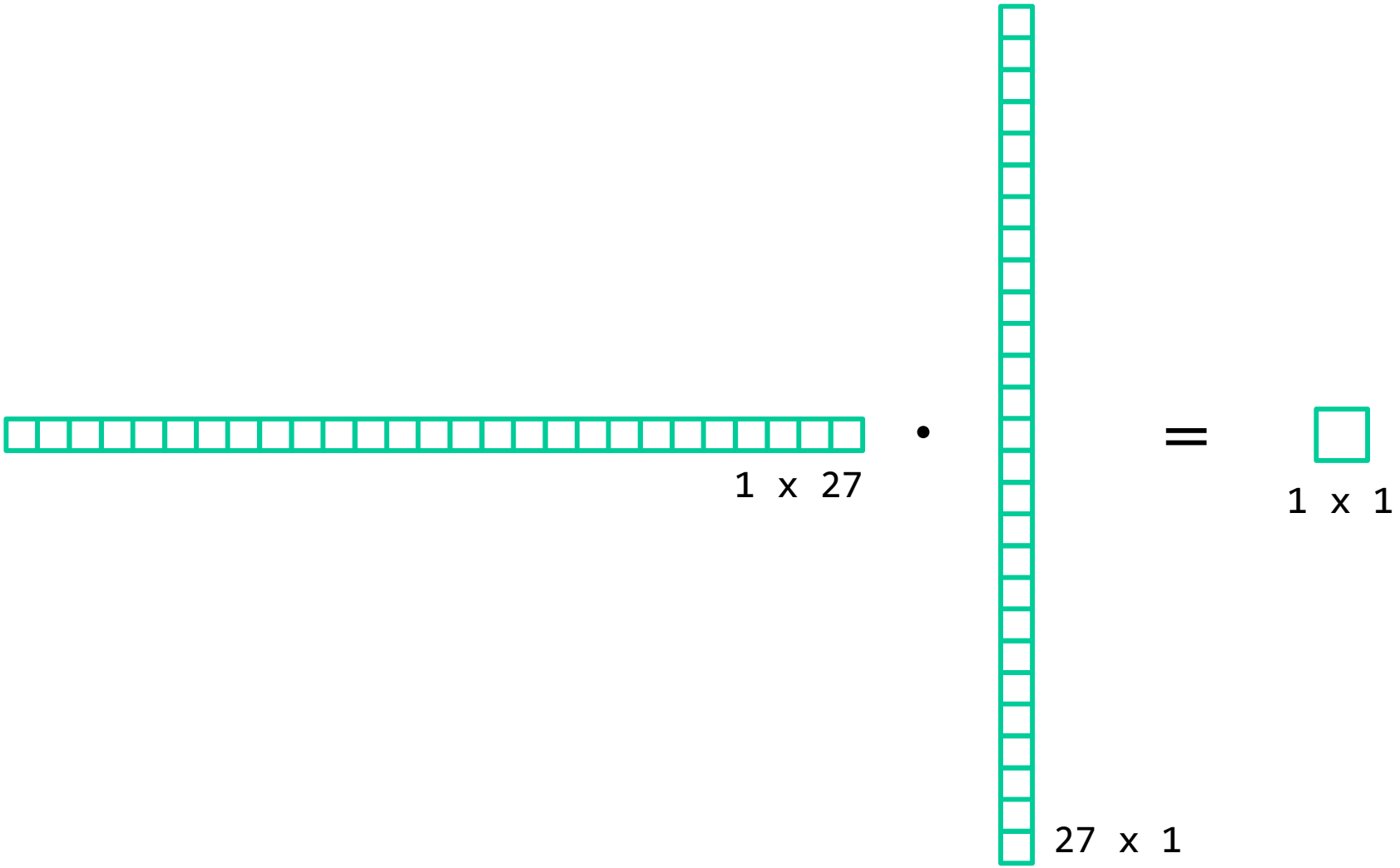
LSTM

- `model = Sequential()`
  `model.add(LSTM(32, input_shape=(50, 2)))`
  `model.add(Dense(1))`
- `input_shape=(50, 2)`
  - ➡ tupla indica (#obs, #features)
  - ➡ 50 observações, ou 50 passos no tempo em 1 uma observação
  - ➡ 2 features em cada observação

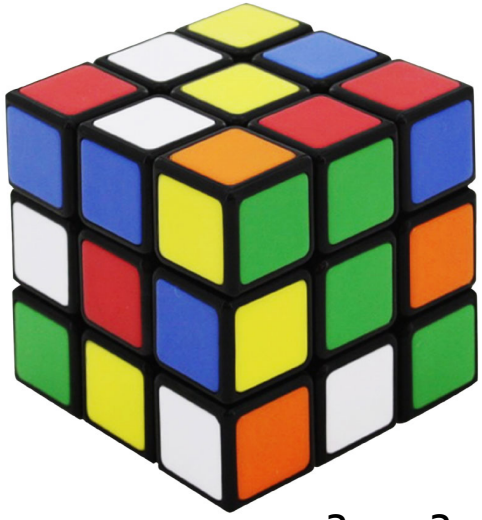


Convnet

dot product



dot product



3 x 3 x 3

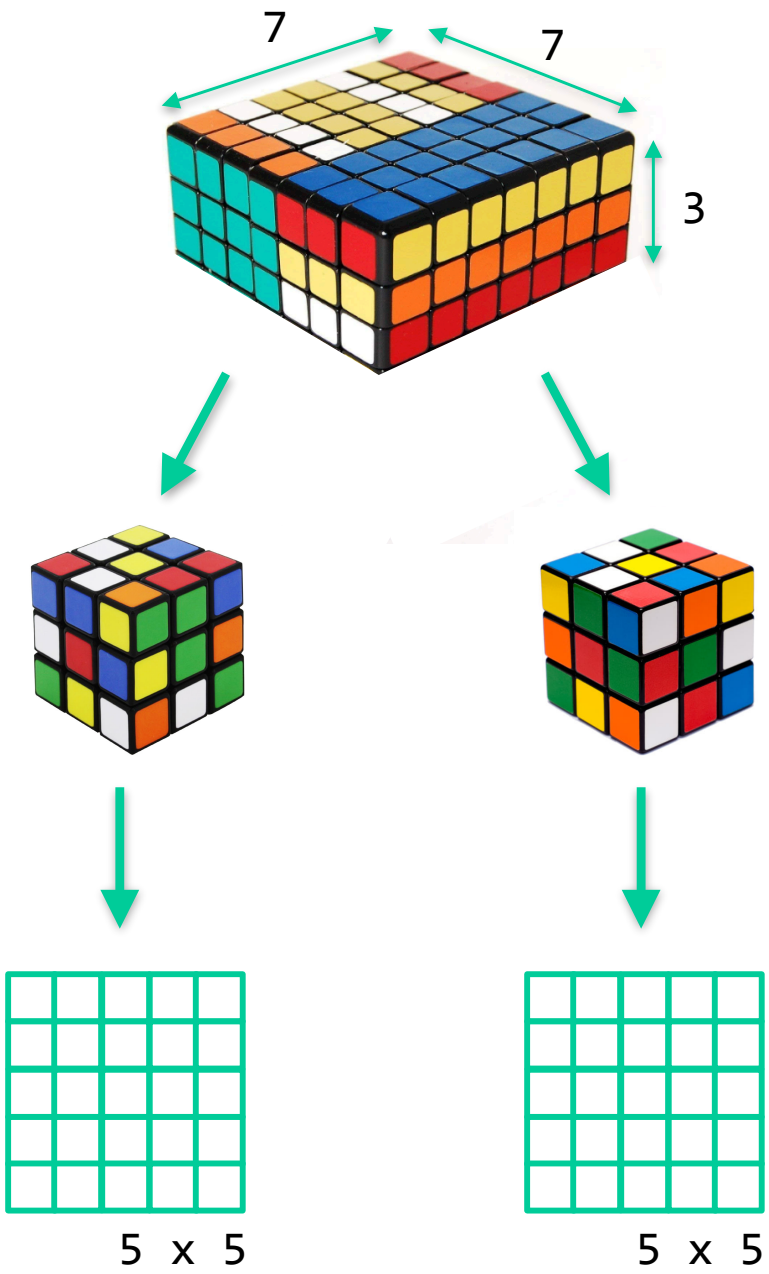
•

3 x 3 x 3

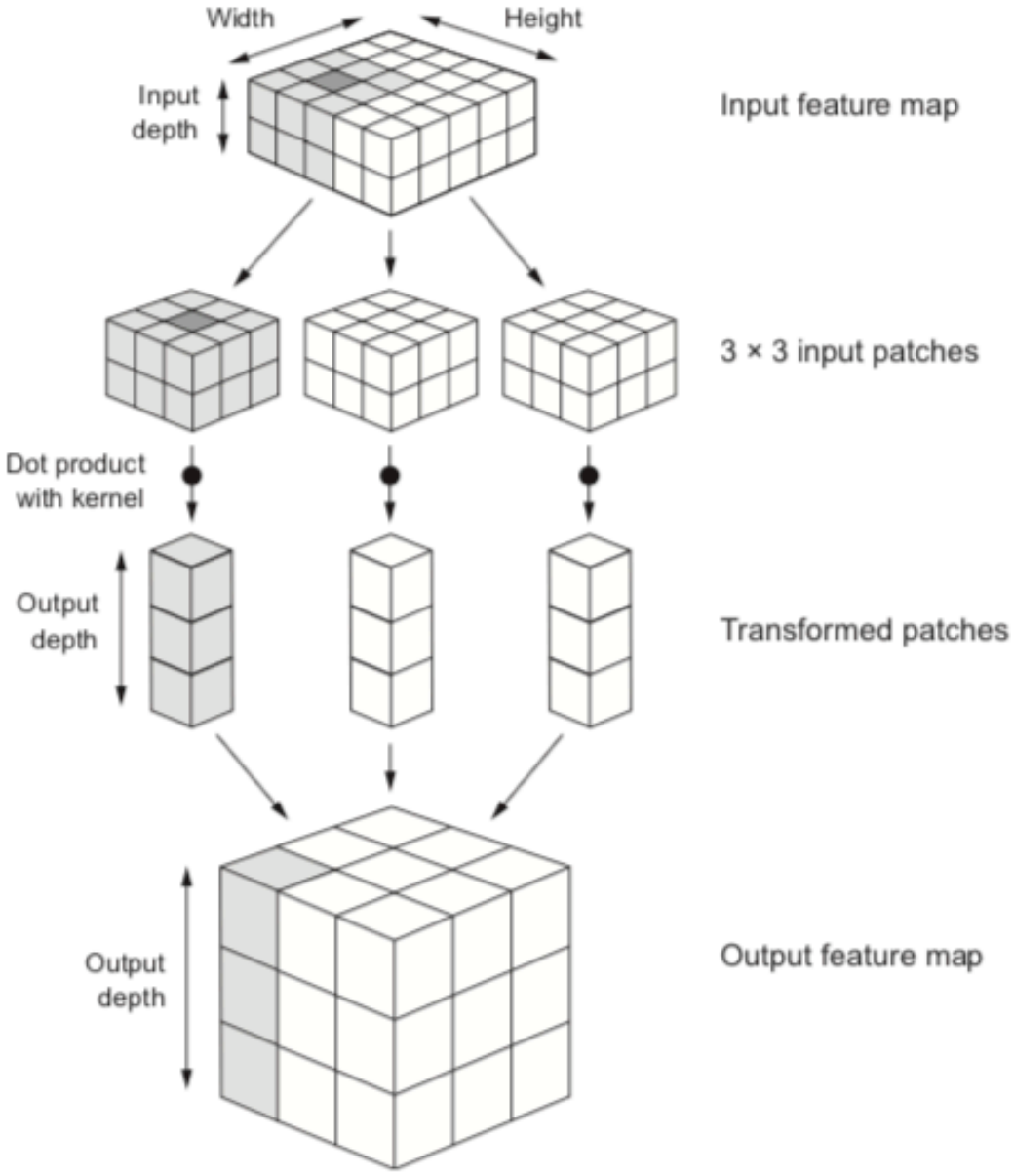
=

1 x 1

convolução

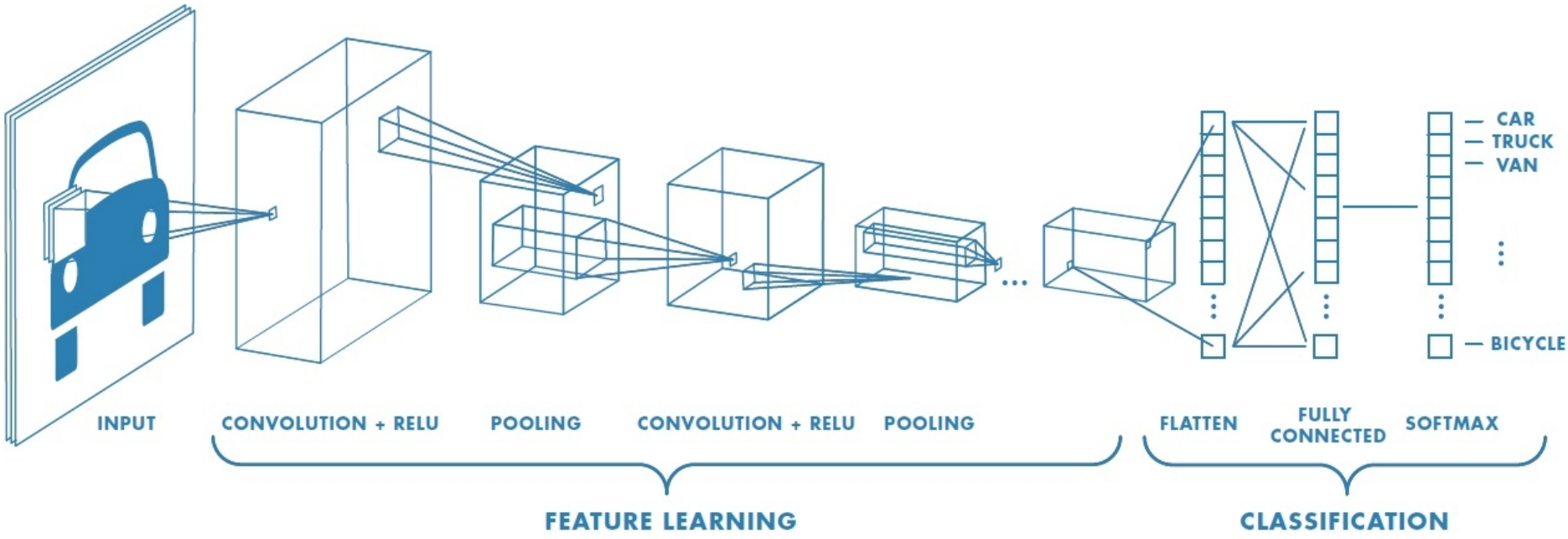


convnets



fonte: Francois Chollet - Deep_Learning_with_Python

convnets



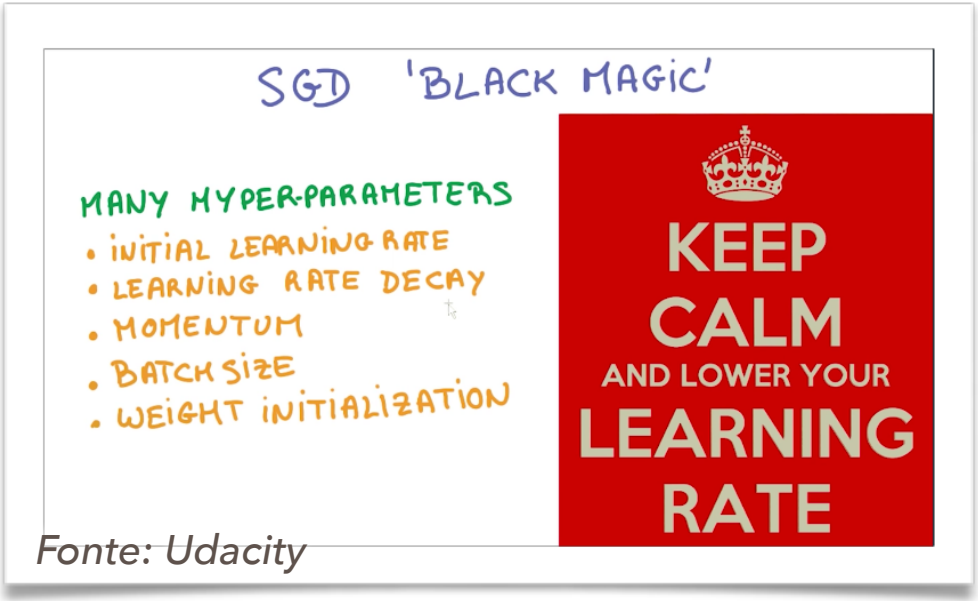
fonte: https://github.com/llSourcell/Convolutional_neural_network/blob/master/convolutional_network_tutorial.ipynb

roteiro keras

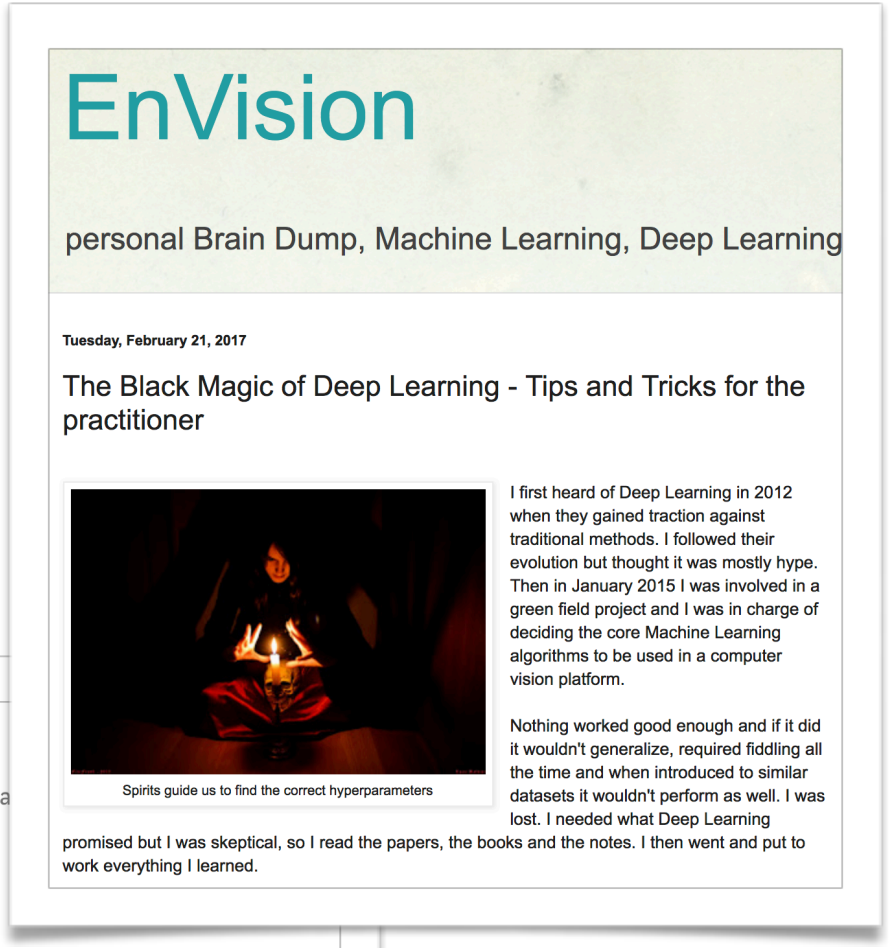
- importar módulos
  - `Sequential, Dense, Dropout,…`
- processar X
  - `reshape(. . . .)`
  - `astype(‘. .’)`
  - `Minmaxscaler()`
- processar y
  - `y: to_categorical(. ., 10).  # 10 labels`
- criar modelos
  - `model = Sequential()`
  - `model.add(. . .)`
- compilar
  - `model.compile(loss = ‘. .’, optimiser = ‘. .’, metrics = [‘. .’])`
- treinar
  - `model.fit(X, y, batch_size = .., epochs = .., verbose = ..)`
- predizer ou avaliar
  - `model.predict(X_test)`
  - `model.evaluate(X_test, Y_test)`

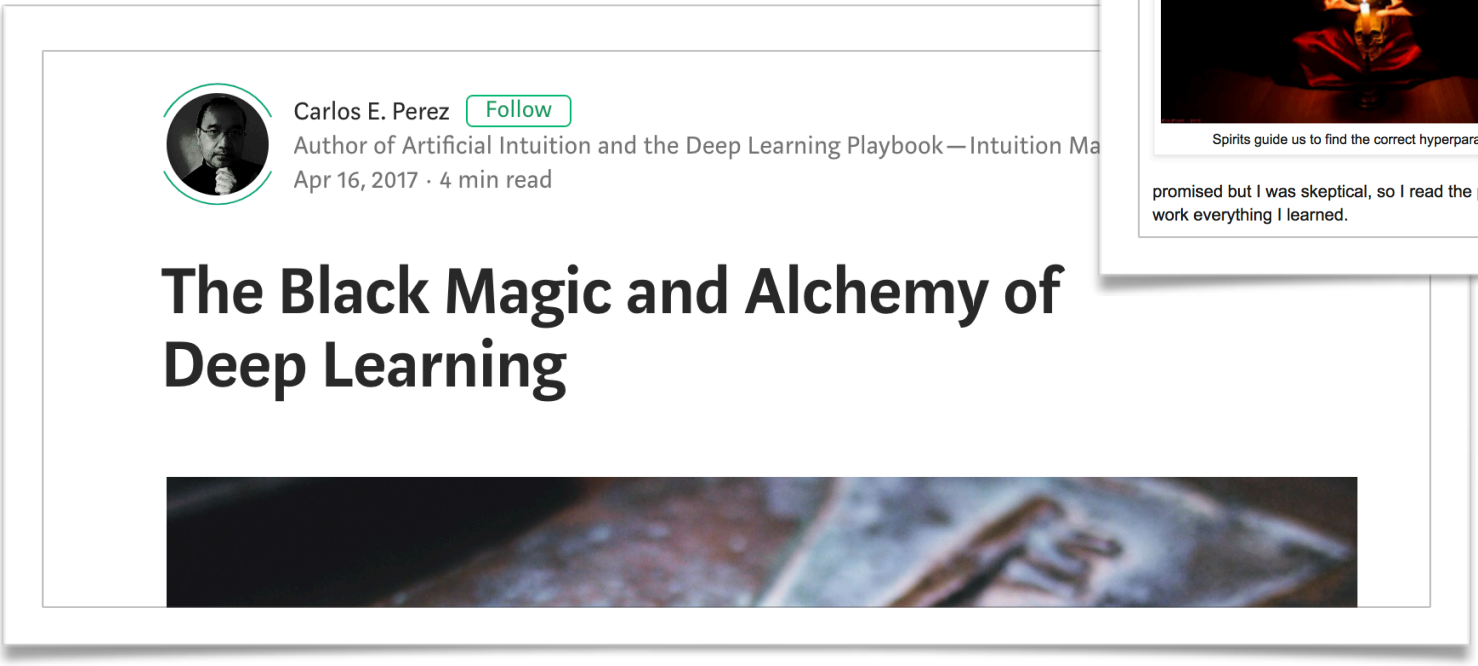
Mão NA MASSA

black magic

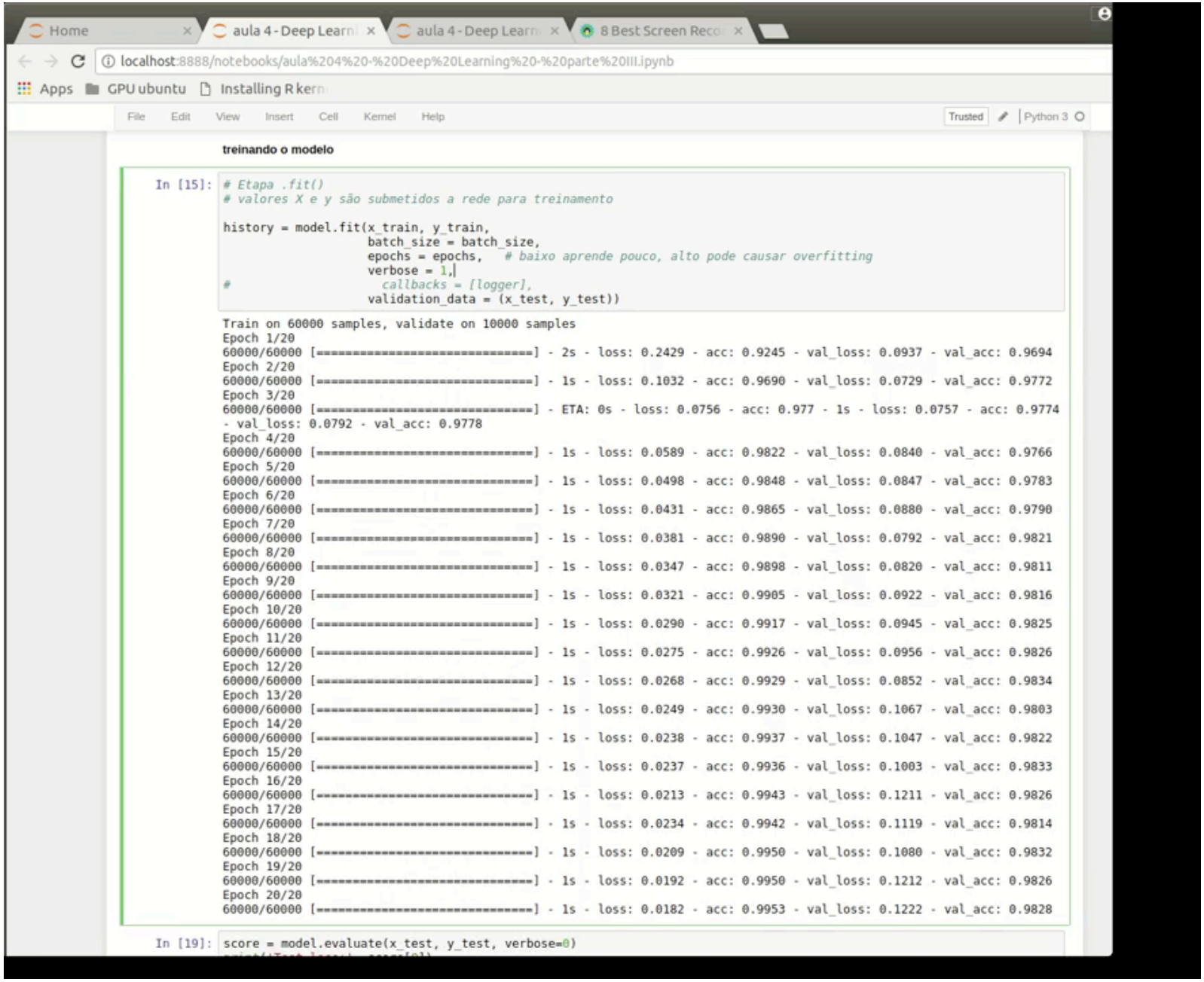


Fonte: Udacity





CPU vs. GPU


Home
aula 4 - Deep Learn
aula 4 - Deep Learn
8 Best Screen Reco
localhost:8888/notebooks/aula%204%20-%20Deep%20Learning%20-%20parte%20III.ipynb
Apps
GPU ubuntu
Installing R kern
File
Edit
View
Insert
Cell
Kernel
Help
Trusted
Python 3
treinando o modelo
In [15]:
# Etapa .fit()
# valores X e y são submetidos a rede para treinamento
history = model.fit(x_train, y_train,
batch_size = batch_size,
epochs = epochs, # baixo aprende pouco, alto pode causar overfitting
verbose = 1,
# callbacks = [logger],
validation_data = (x_test, y_test))
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==========] - 2s - loss: 0.2429 - acc: 0.9245 - val_loss: 0.0937 - val_acc: 0.9694
Epoch 2/20
60000/60000 [==========] - 1s - loss: 0.1032 - acc: 0.9690 - val_loss: 0.0729 - val_acc: 0.9772
Epoch 3/20
60000/60000 [==========] - ETA: 0s - loss: 0.0756 - acc: 0.977 - 1s - loss: 0.0757 - acc: 0.9774
- val_loss: 0.0792 - val_acc: 0.9778
Epoch 4/20
60000/60000 [==========] - 1s - loss: 0.0589 - acc: 0.9822 - val_loss: 0.0840 - val_acc: 0.9766
Epoch 5/20
60000/60000 [==========] - 1s - loss: 0.0498 - acc: 0.9848 - val_loss: 0.0847 - val_acc: 0.9783
Epoch 6/20
60000/60000 [==========] - 1s - loss: 0.0431 - acc: 0.9865 - val_loss: 0.0880 - val_acc: 0.9790
Epoch 7/20
60000/60000 [==========] - 1s - loss: 0.0381 - acc: 0.9890 - val_loss: 0.0792 - val_acc: 0.9821
Epoch 8/20
60000/60000 [==========] - 1s - loss: 0.0347 - acc: 0.9898 - val_loss: 0.0820 - val_acc: 0.9811
Epoch 9/20
60000/60000 [==========] - 1s - loss: 0.0321 - acc: 0.9905 - val_loss: 0.0922 - val_acc: 0.9816
Epoch 10/20
60000/60000 [==========] - 1s - loss: 0.0290 - acc: 0.9917 - val_loss: 0.0945 - val_acc: 0.9825
Epoch 11/20
60000/60000 [==========] - 1s - loss: 0.0275 - acc: 0.9926 - val_loss: 0.0956 - val_acc: 0.9826
Epoch 12/20
60000/60000 [==========] - 1s - loss: 0.0268 - acc: 0.9929 - val_loss: 0.0852 - val_acc: 0.9834
Epoch 13/20
60000/60000 [==========] - 1s - loss: 0.0249 - acc: 0.9930 - val_loss: 0.1067 - val_acc: 0.9803
Epoch 14/20
60000/60000 [==========] - 1s - loss: 0.0238 - acc: 0.9937 - val_loss: 0.1047 - val_acc: 0.9822
Epoch 15/20
60000/60000 [==========] - 1s - loss: 0.0237 - acc: 0.9936 - val_loss: 0.1003 - val_acc: 0.9833
Epoch 16/20
60000/60000 [==========] - 1s - loss: 0.0213 - acc: 0.9943 - val_loss: 0.1211 - val_acc: 0.9826
Epoch 17/20
60000/60000 [==========] - 1s - loss: 0.0234 - acc: 0.9942 - val_loss: 0.1119 - val_acc: 0.9814
Epoch 18/20
60000/60000 [==========] - 1s - loss: 0.0209 - acc: 0.9950 - val_loss: 0.1080 - val_acc: 0.9832
Epoch 19/20
60000/60000 [==========] - 1s - loss: 0.0192 - acc: 0.9950 - val_loss: 0.1212 - val_acc: 0.9826
Epoch 20/20
60000/60000 [==========] - 1s - loss: 0.0182 - acc: 0.9953 - val_loss: 0.1222 - val_acc: 0.9828
In [19]:
score = model.evaluate(x_test, y_test, verbose=0)

CPU vs. GPU