Inteligência Artificial

Ciência da
Computação

Ciência da
Computação

Redes Neurais

ATITUS
EDUCAÇÃO

Redes Neurais

Origem: algoritmos que tentam imitar o cérebro.

Utilizado nos anos 1980 e 1990.

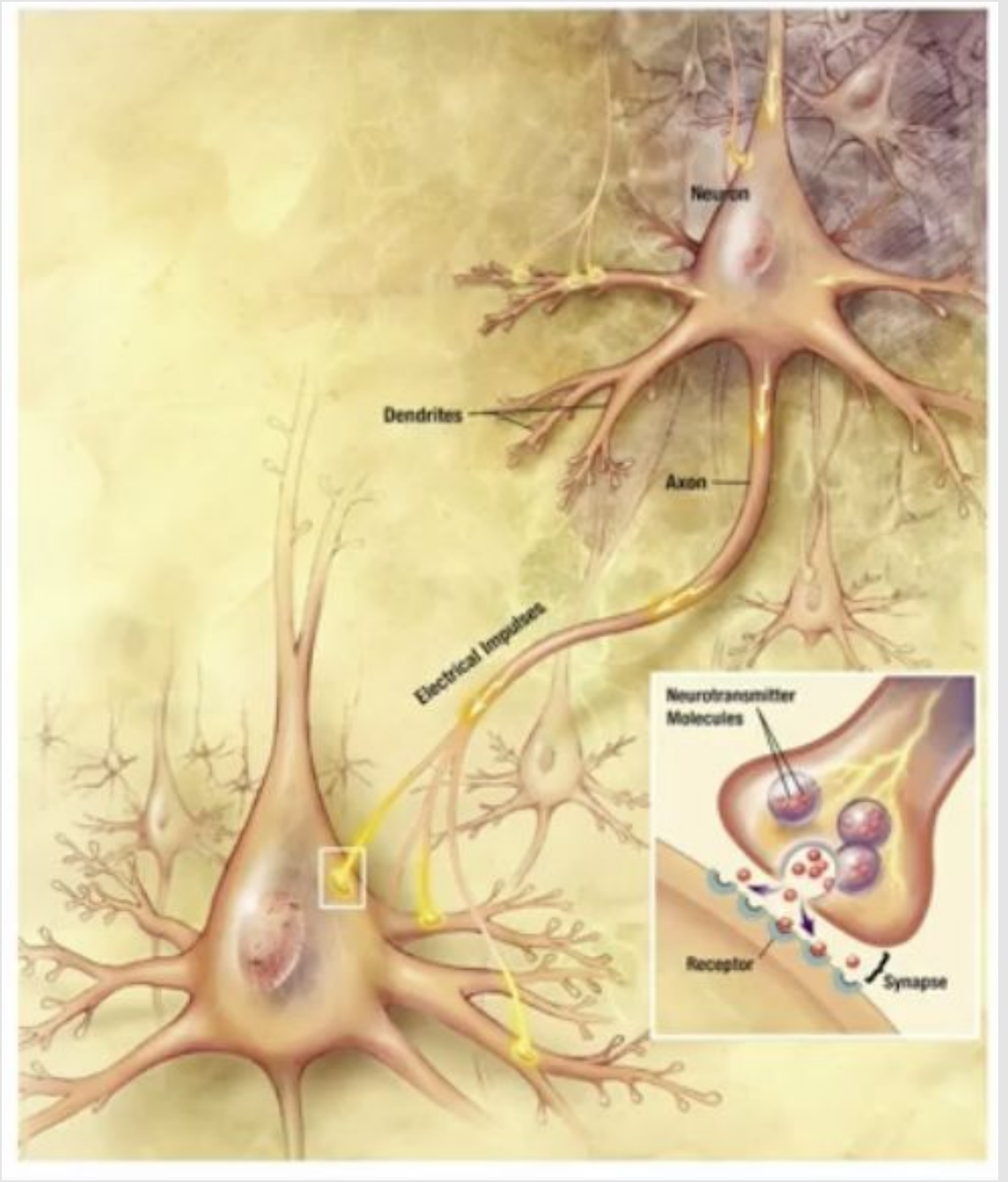
Após 15 anos sem uso voltou à evidência em 2005.

Reconhecimento de fala → Imagens → Texto

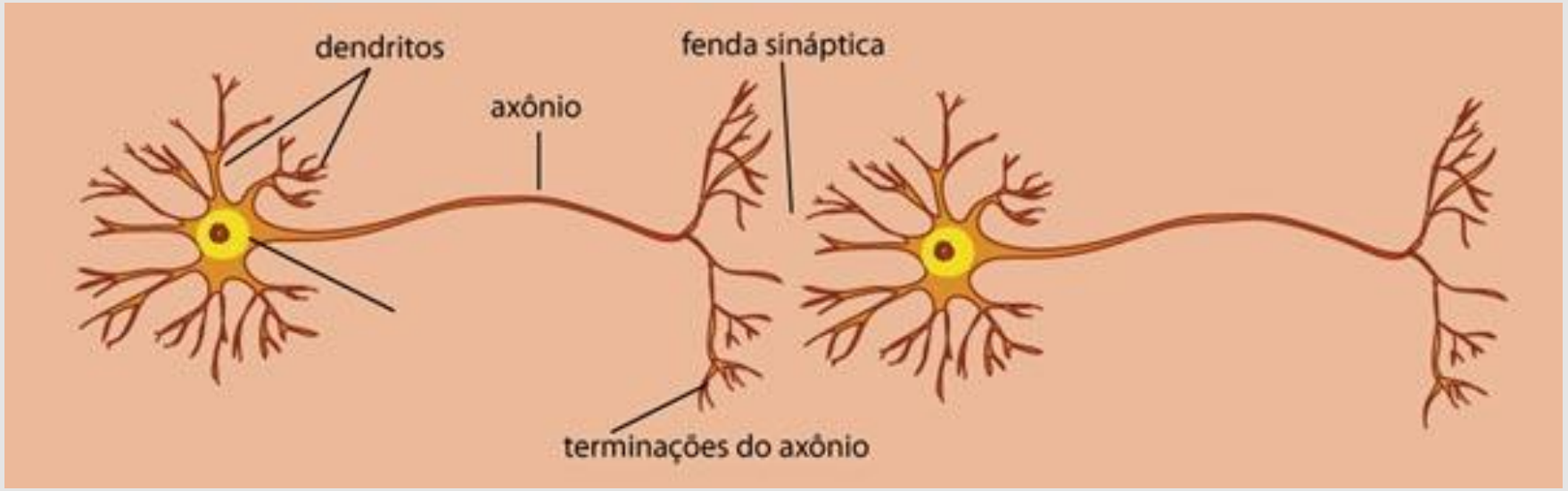
https://www.youtube.com/watch?v=hGDvvUNU-cw

https://www.youtube.com/shorts/BloWyn0QTrI

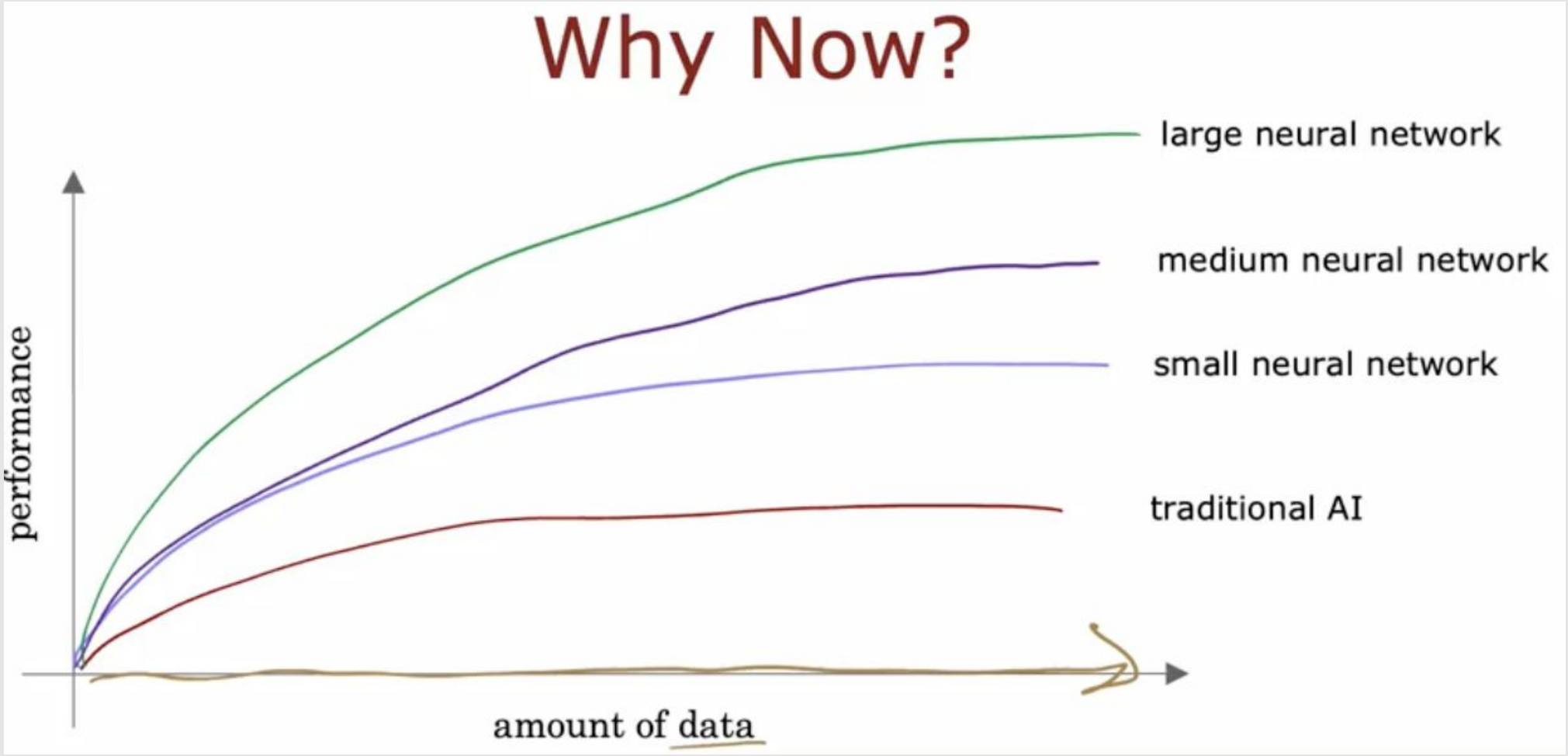
Redes Neurais



Redes Neurais



Redes Neurais



Redes Neurais

https://www.youtube.com/watch?v=TmPfTpjtdgg

https://www.youtube.com/watch?v=4vH86Jk2fjY

https://www.youtube.com/watch?v=g9apeWoNa-0

https://www.youtube.com/watch?v=yi5uGX1ovvc

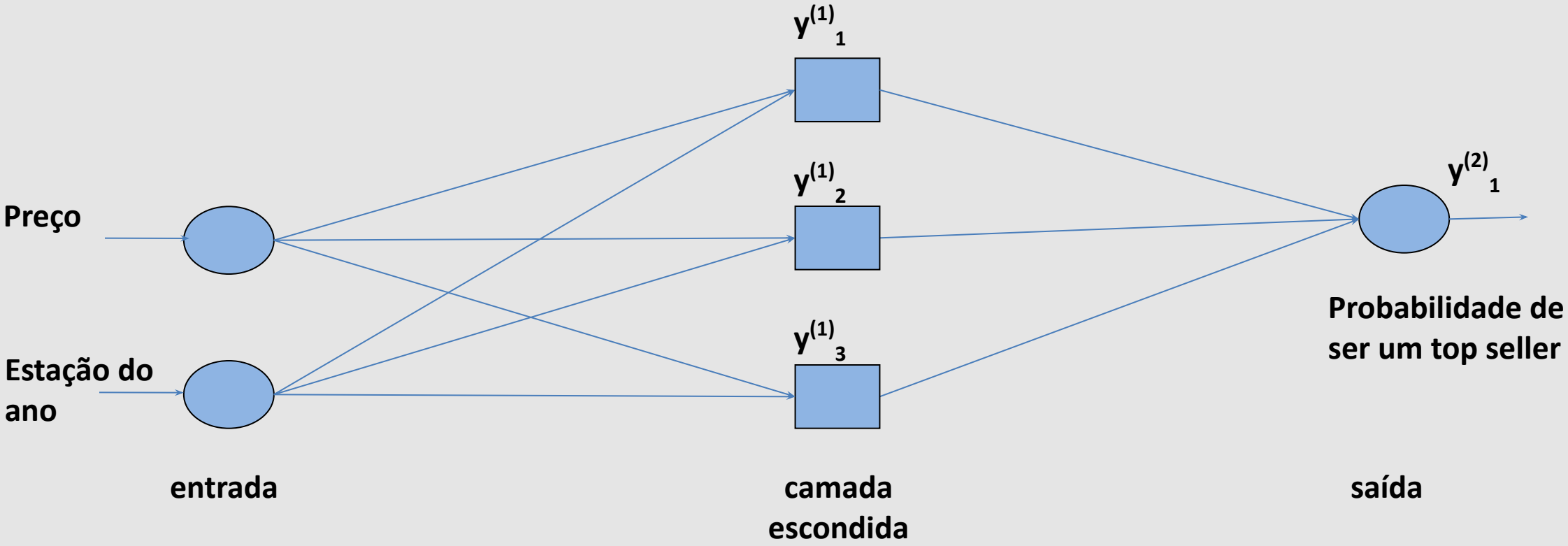
https://www.youtube.com/watch?v=e-uf510bXH0

https://www.youtube.com/watch?v=qtLyyGkdh_U

https://quickdraw.withgoogle.com/

https://notebooklm.google/

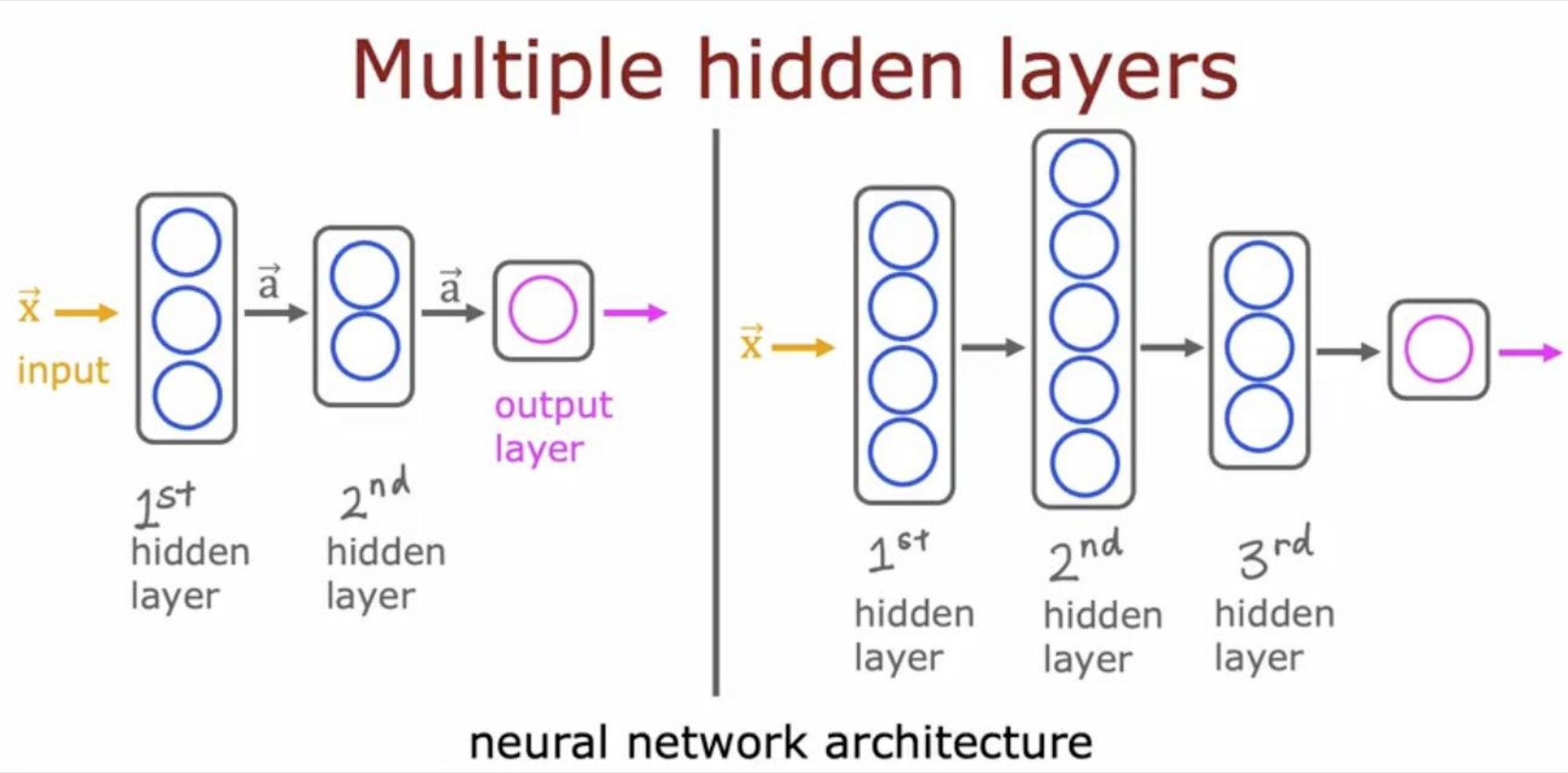
Redes Neurais: predição de demanda



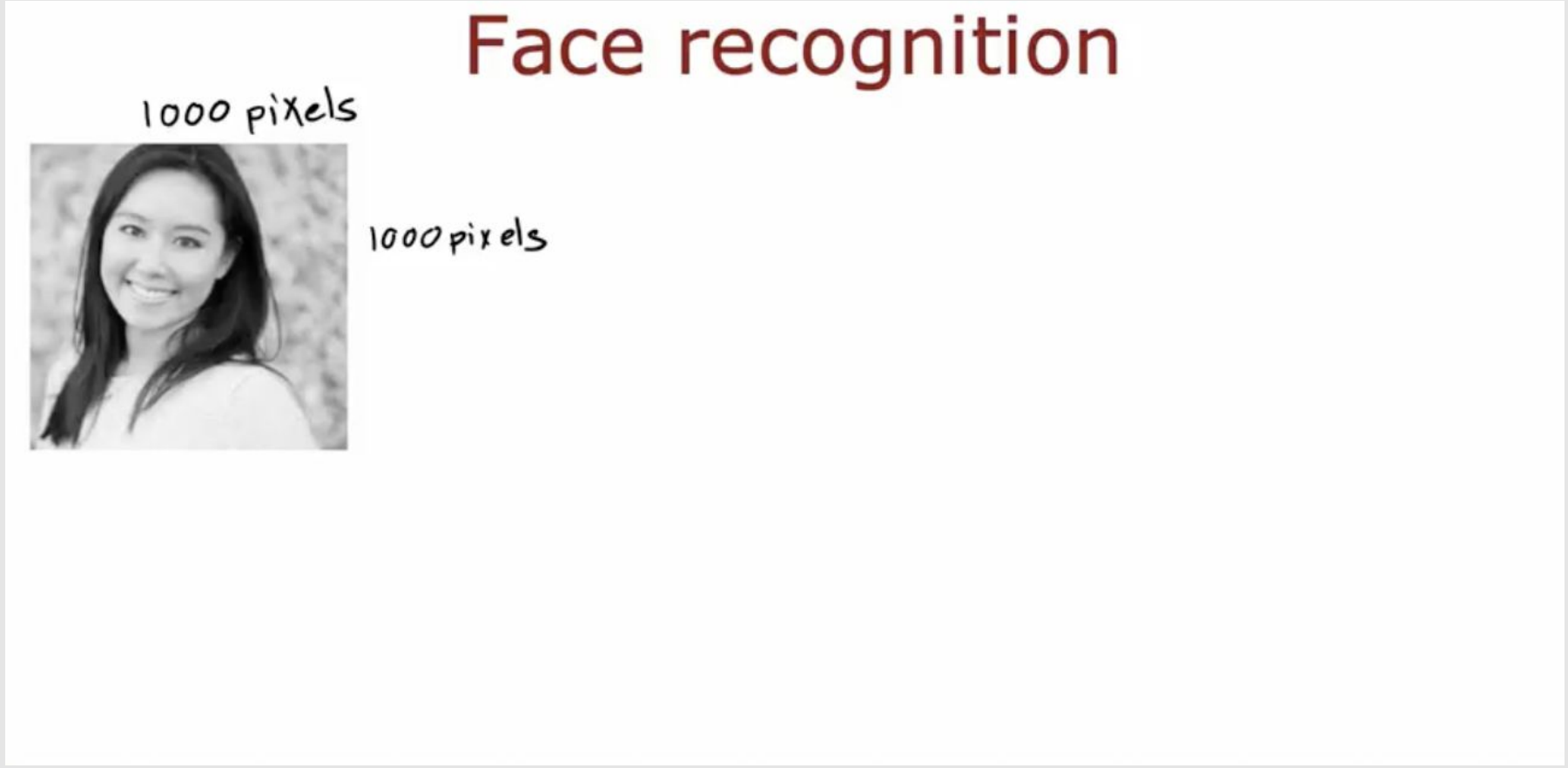
https://developers.google.com/machine-learning/crash-course/neural-networks/interactive-exercises?hl=pt-br

https://playground.tensorflow.org/

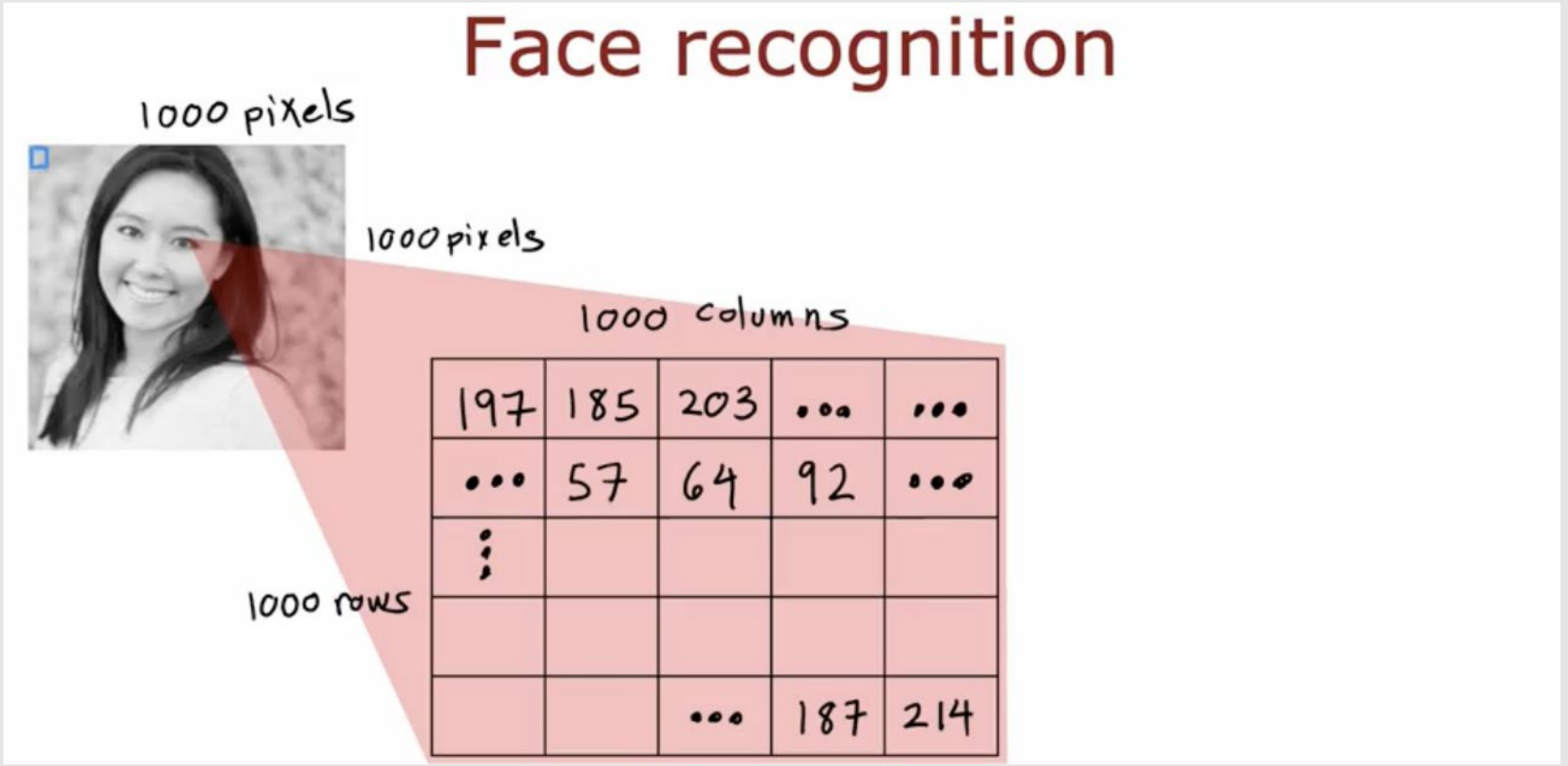
Redes Neurais: arquiteturas



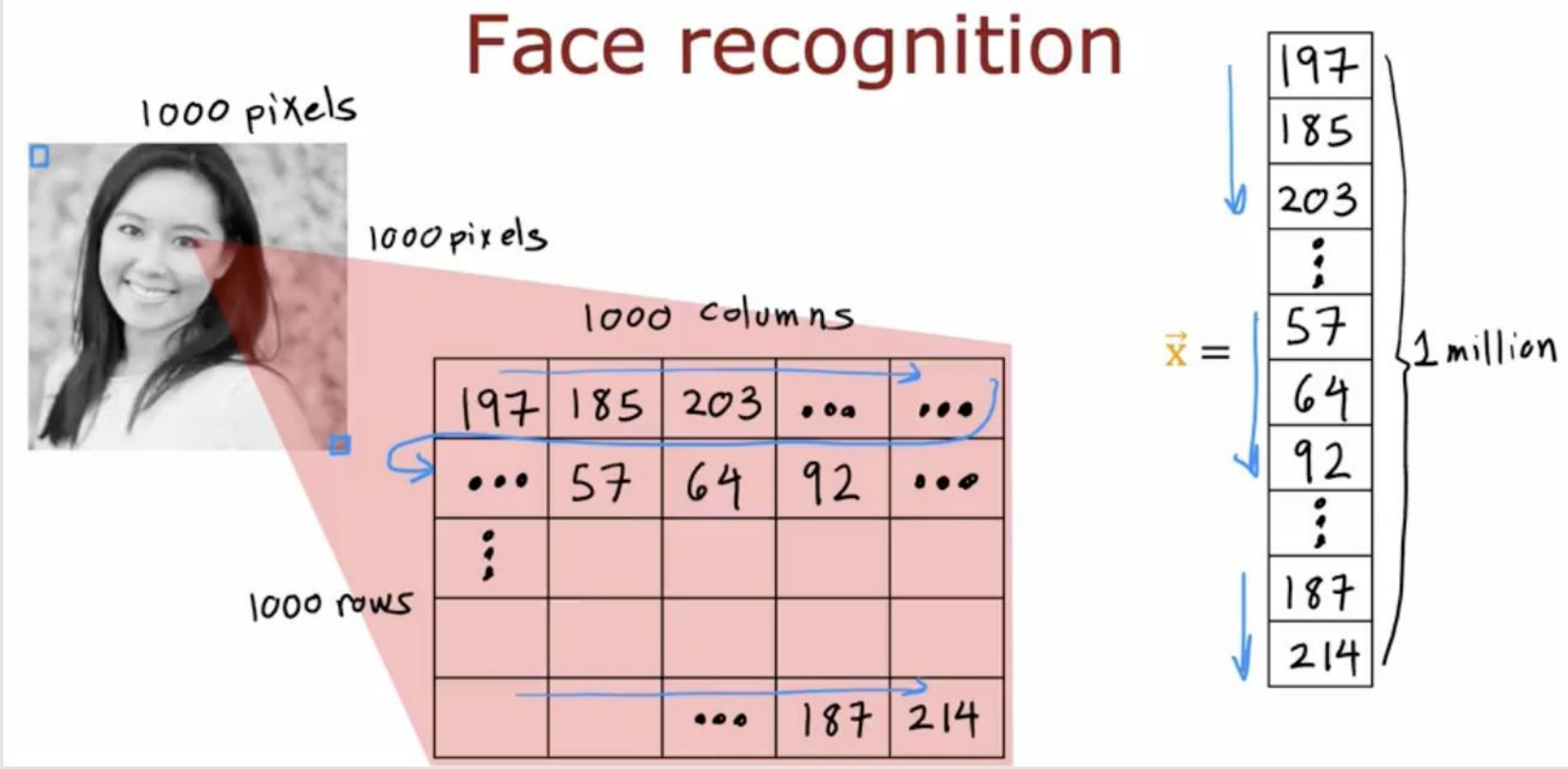
Redes Neurais: reconhecimento facial



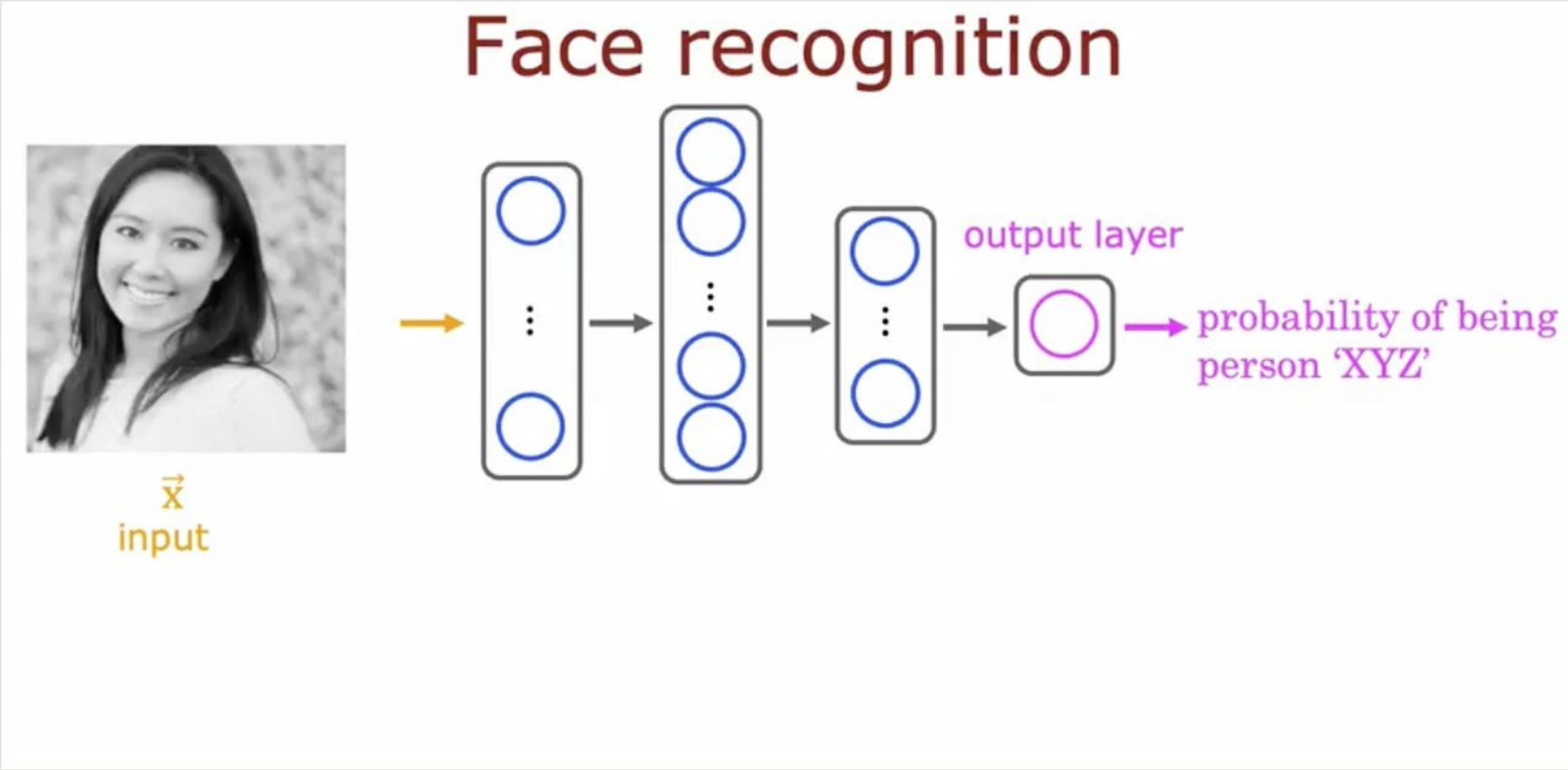
Redes Neurais: reconhecimento facial



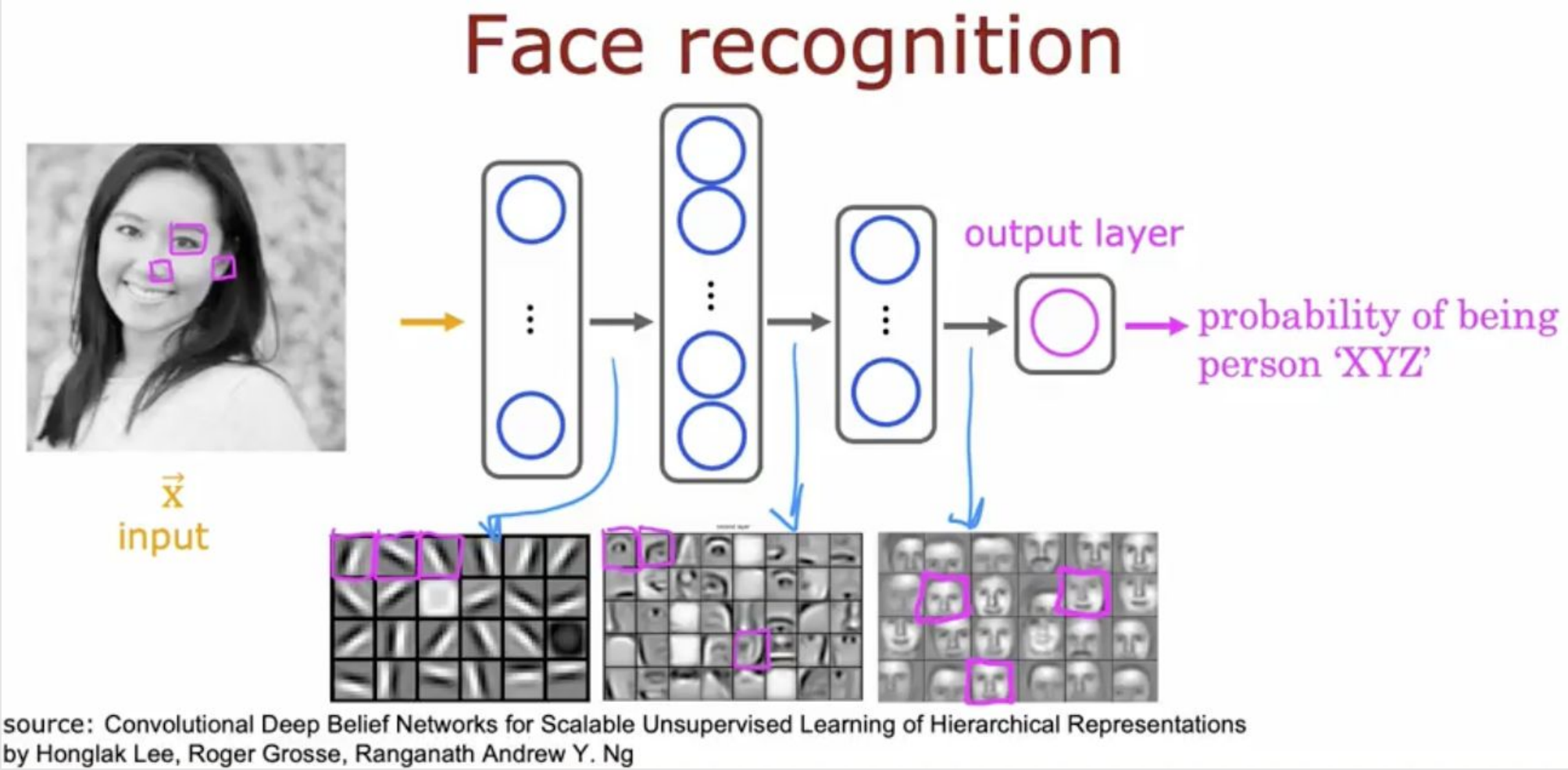
Redes Neurais: reconhecimento facial



Redes Neurais: reconhecimento facial

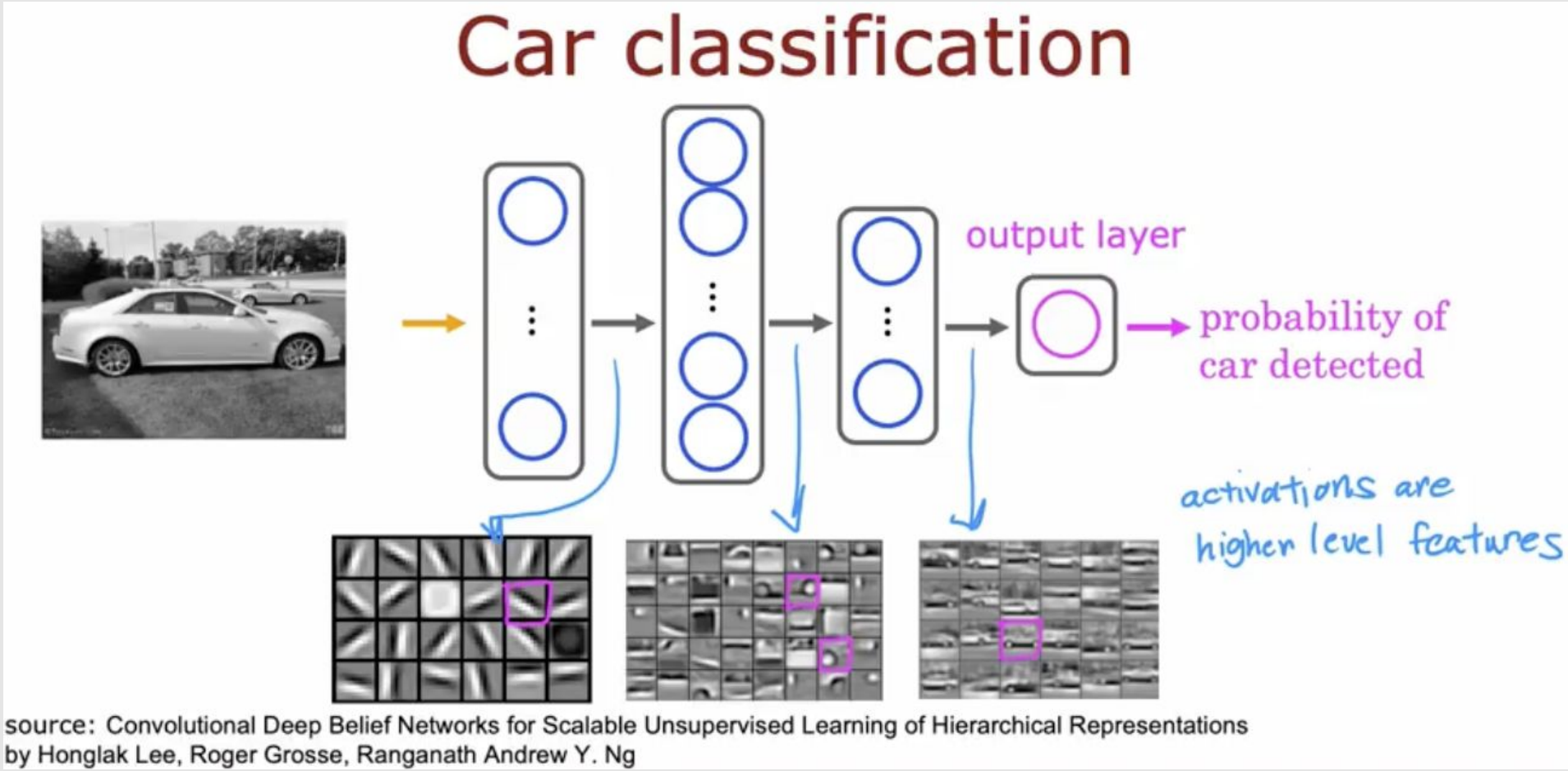


Redes Neurais: reconhecimento facial



source: Convolutional Deep Belief Networks for Scalable Unsupervised Learning of Hierarchical Representations by Honglak Lee, Roger Grosse, Ranganath Andrew Y. Ng

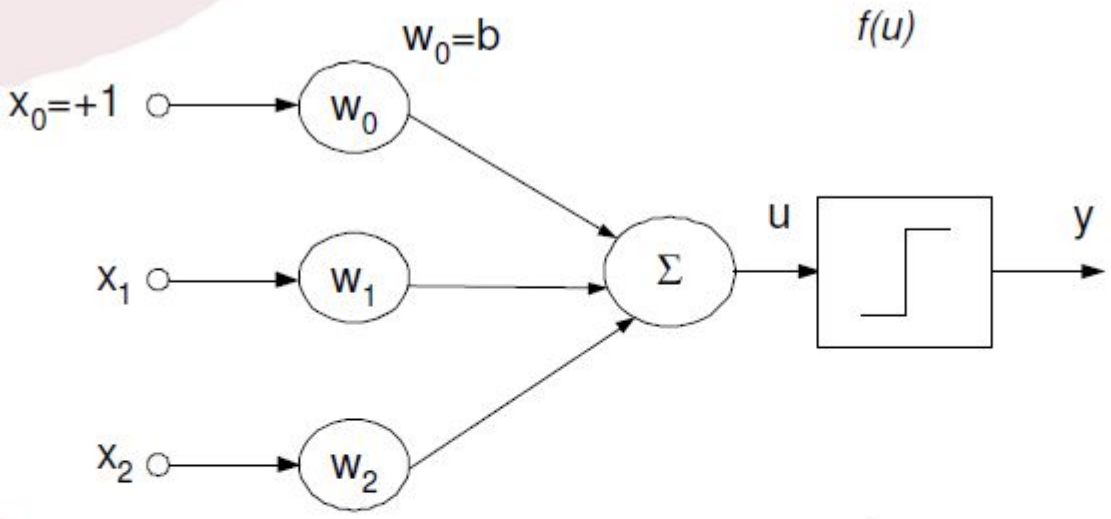
Redes Neurais: reconhecimento de itens



source: Convolutional Deep Belief Networks for Scalable Unsupervised Learning of Hierarchical Representations by Honglak Lee, Roger Grosse, Ranganath Andrew Y. Ng

Rede Neural Perceptron

•



$$y = f(w_1 x_1 + w_2 x_2 + w_0), \text{ sendo } \begin{cases} f(u) = 1 \text{ se } u \geq 0 \\ f(u) = 0 \text{ se } u < 0 \end{cases}$$

Com os parâmetros w_0, w_1 e w_2, a função *f(u)* separa o espaço de entradas em **duas regiões**, usando uma linha reta dada por:

$$w_1 x_1 + w_2 x_2 + w_0 = 0$$

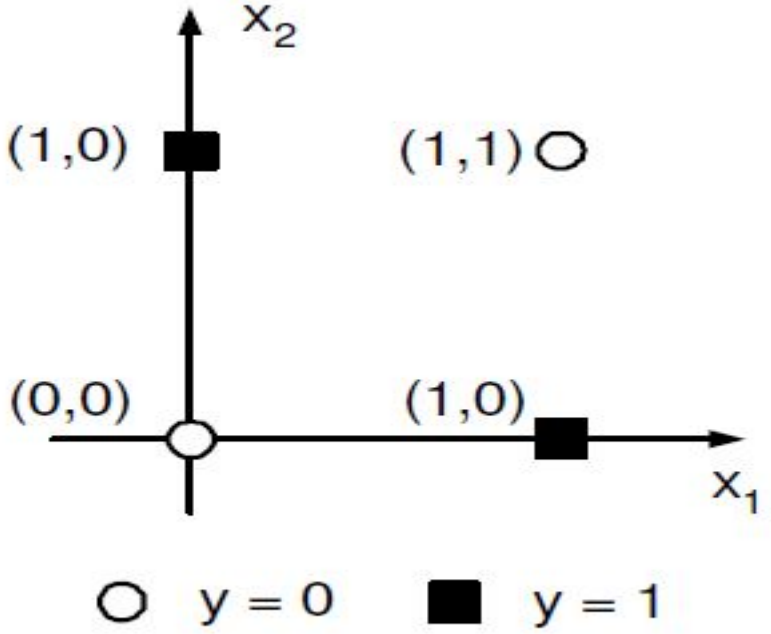
Rede Neural Perceptron

- O uso do bias permite que fixemos o valor de ***threshold*** adotado em nossa **função de ativação**, sendo necessário então atualizar somente os **pesos** e o **bias** na rede.
- Como o bias pode ser encarado como sendo o peso para um neurônio cuja entrada é sempre 1, percebe-se que a mesma **regra para atualização dos pesos** é válida também para a atualização do bias.

Rede Neural Perceptron

No caso do XOR, não existe uma única reta que divide os pontos (0,0) e (1,1) para um lado, e (0,1) e (1,0) do outro lado.

Função XOR



Conclui-se que um **neurônio do tipo Perceptron** não implementa uma **função ou-exclusivo** (constatado por Minsky & Papert, em 1969).

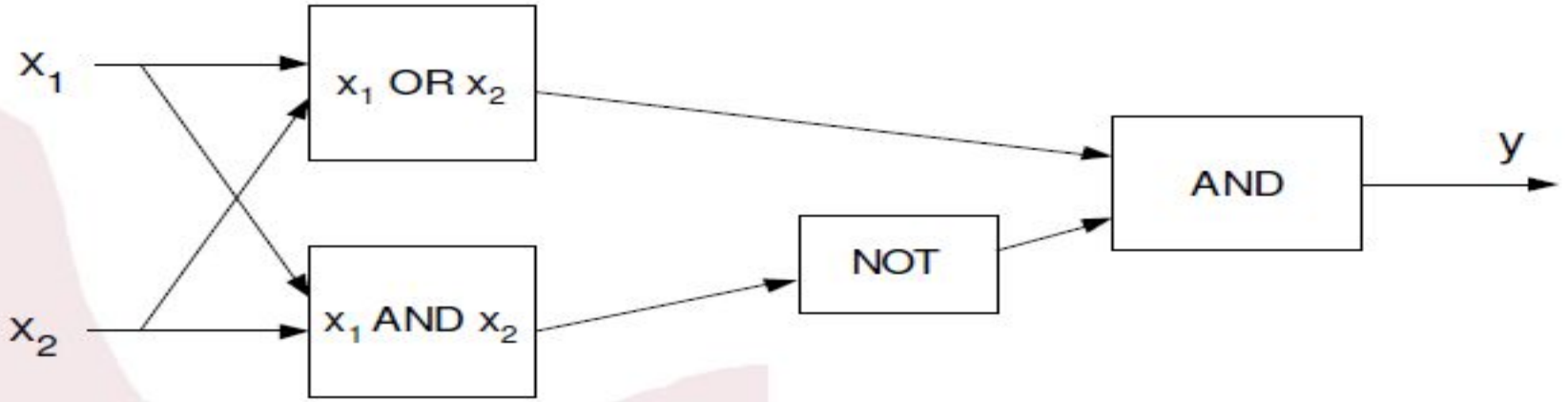
Rede Neural Perceptron

A função XOR está além da capacidade de um Perceptron simples.

Contudo, um **Perceptron simples pode implementar funções lógicas elementares**: AND, OR e NOT.

Assim, se uma função pode ser expressa como uma **combinação dessas funções lógicas elementares**, então essa função pode ser implementada usando mais neurônios.

Por exemplo, XOR pode ser expressa por: $(x_1$ or $x_2)$ and (not $(x_1$ and $x_2))$.

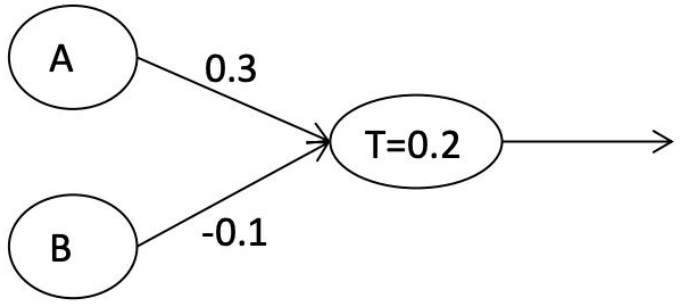


Rede Neural: treinando um neurônio

Operador And

A	B	Saída
0	0	0
0	1	0
1	0	0
1	1	1

Threshold = 0.2
Learning Rate = 0.1



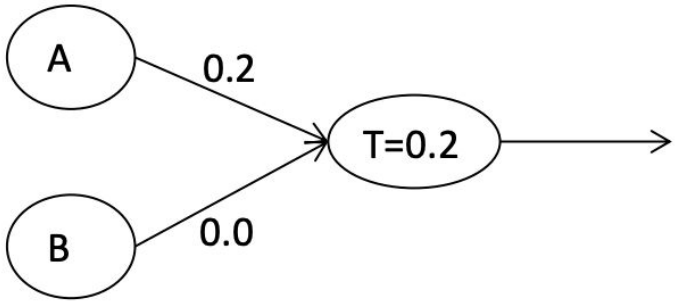
A	B	Somatório	Saída	Erro
0	0	(0*0.3)+(0*-0.1) = 0	0	0
0	1	(0*0.3)+(1*-0.1) = -0.1	0	0
1	0	(1*0.3)+(0*-0.1) = 0.3	1	-1
1	1	(1*0.3)+(1*-0.1) = 0.2	1	0

Rede Neural: treinando um neurônio

Operador And

A	B	Saída
0	0	0
0	1	0
1	0	0
1	1	1

Threshold = 0.2
Learning Rate = 0.1



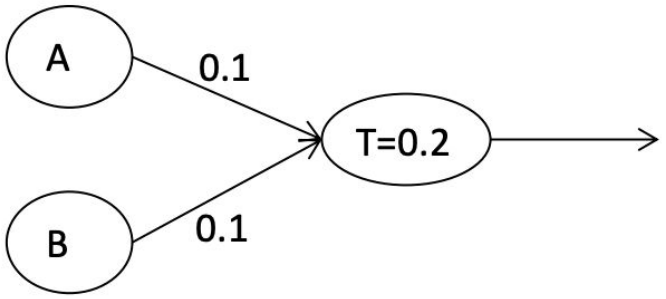
A	B	Somatório	Saída	Erro
0	0	(0*0.2)+(0*0.0) = 0	0	0
0	1	(0*0.2)+(1*0.0) = 0	0	0
1	0	(1*0.2)+(0*0.0) = 0.2	1	-1
1	1	(1*0.2)+(1*0.0) = 0.2	1	0

Rede Neural: treinando um neurônio

Operador And

A	B	Saída
0	0	0
0	1	0
1	0	0
1	1	1

Threshold = 0.2
Learning Rate = 0.1

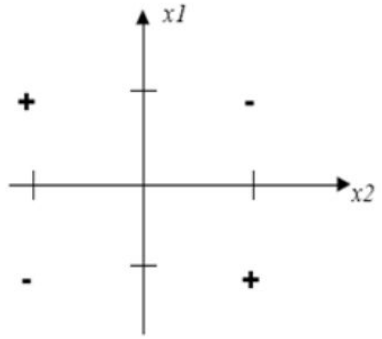


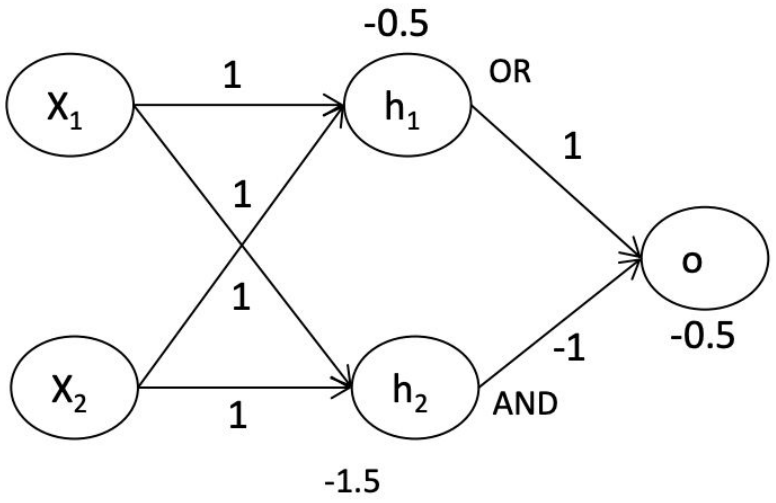
A	B	Somatório	Saída	Erro
0	0	(0*0.1)+(0*0.1) = 0	0	0
0	1	(0*0.1)+(1*0.1) = 0.1	0	0
1	0	(1*0.1)+(0*0.1) = 0.1	0	0
1	1	(1*0.1)+(1*0.1) = 0.2	1	0

Rede Neural: treinando um neurônio

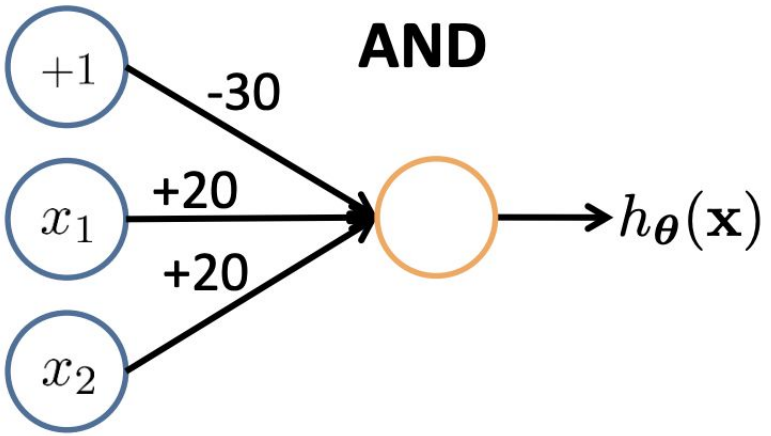
Operador XOR

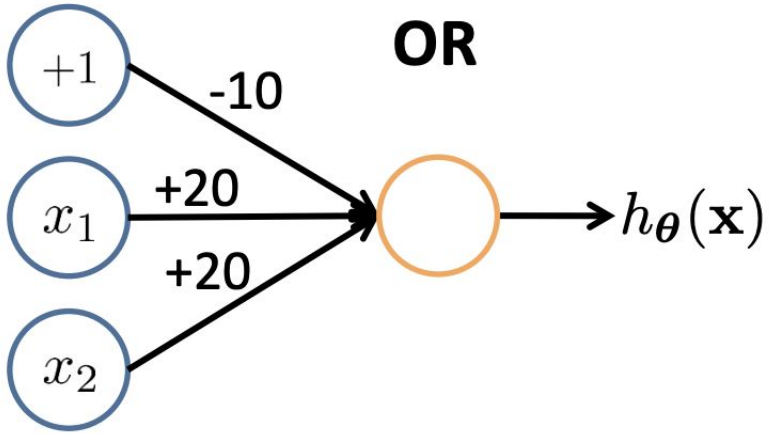
A	B	Saída
0	0	0
0	1	1
1	0	1
1	1	0

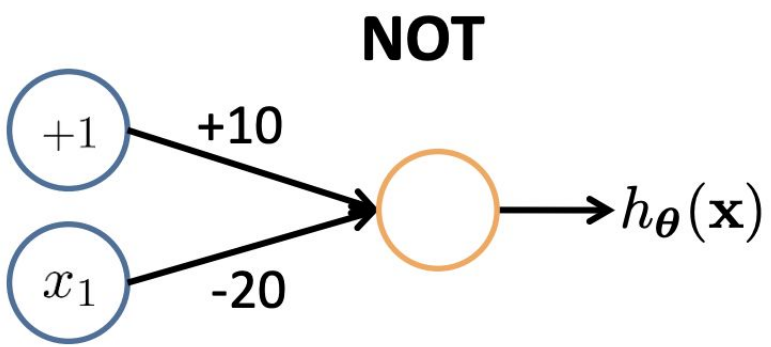


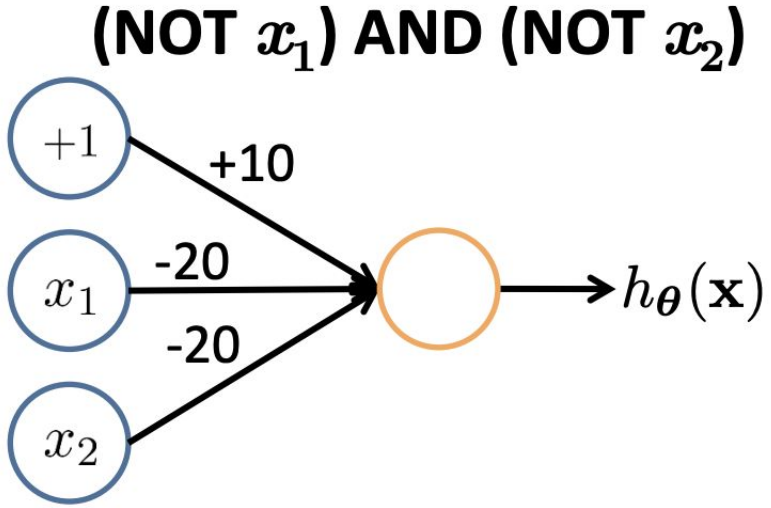


Rede Neural: treinando um neurônio

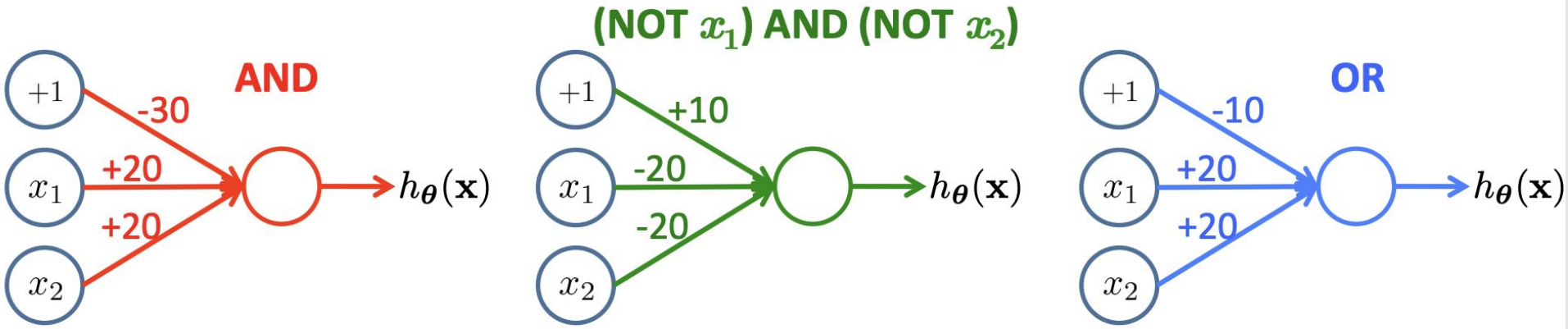


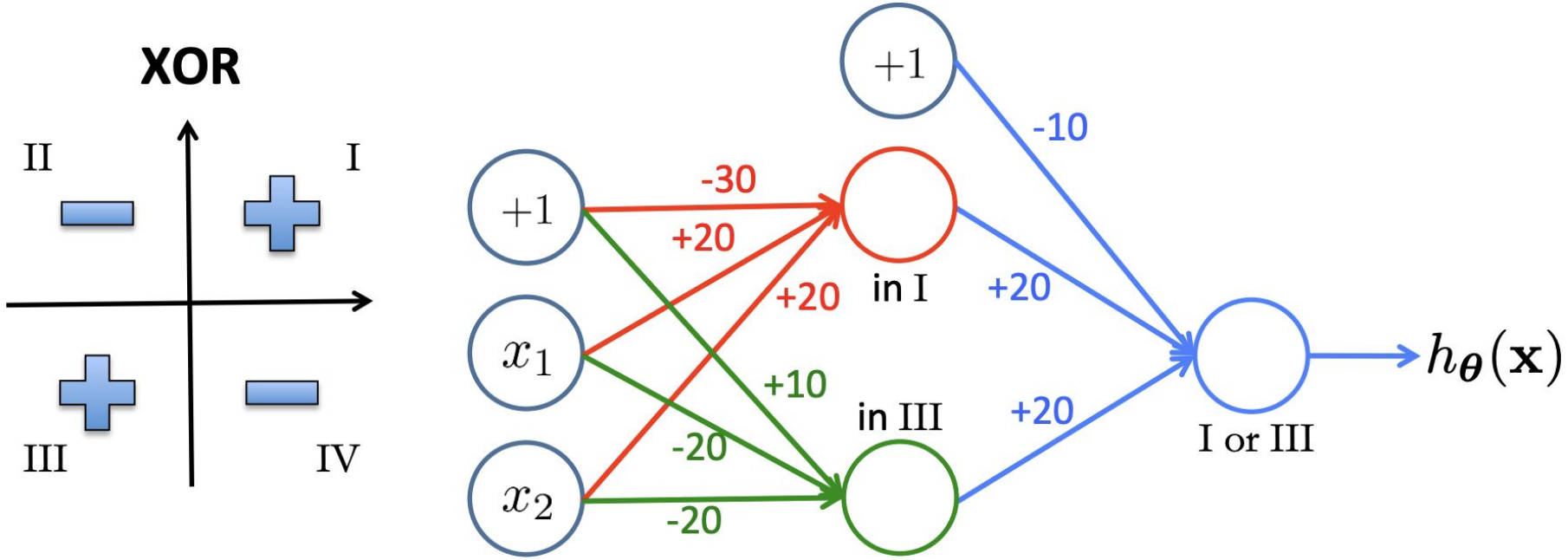






Rede Neural: treinando um neurônio





Rede Neural Perceptron

- Não linearidades são inerentes à maioria das situações e problemas reais.
- Não linearidades são incorporadas através:
  - De funções de ativação não lineares.
  - Da composição de sucessivas camadas de neurônios.
- MLP (*MultiLayer Perceptron*):
  - RNA composta por neurônios com funções de ativação sigmoidais nas camadas intermediárias.

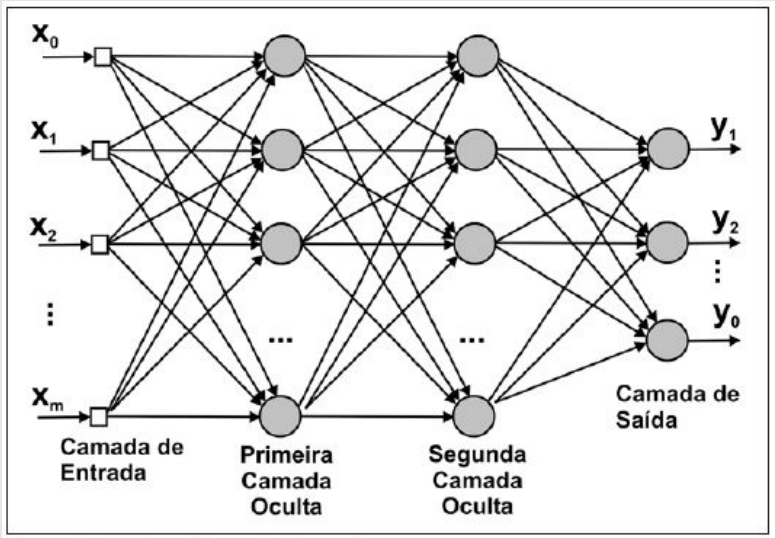


Figura 2 - Rede multilayer feedforward.

Rede Neural Perceptron

- Perceptron:
  - Aprendizado supervisionado e correção de erros
  - Ajustes no vetor de pesos
  - Saída desejada –> saída obtida

- MLP:
  - Aplicado somente à última camada.
  - Não há saídas desejadas para camadas intermediárias.
  - Como calcular ou estimar o erro das camadas intermediárias?

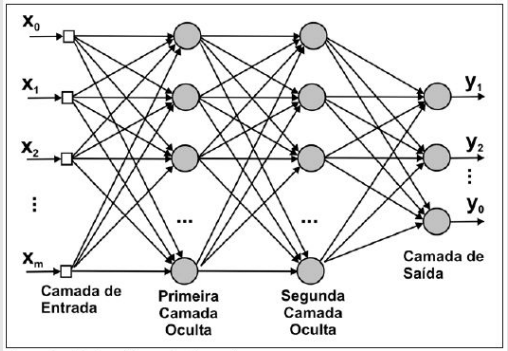


Figura 2 - Rede multilayer feedforward.

Rede Neural MLP

- Algoritmo Back-propagation
  - Década de 80. Novo “gás” para área de redes neurais.
- Gradiente descendente
  - Estimativa de erro das camadas intermediárias pelo **efeito** que estas causam no erro da camada de saída.
  - Erro da camada de saída **retroalimentado** para camadas intermediárias.
  - Ajustes dos pesos **proporcional** aos valores das conexões entre as camadas.

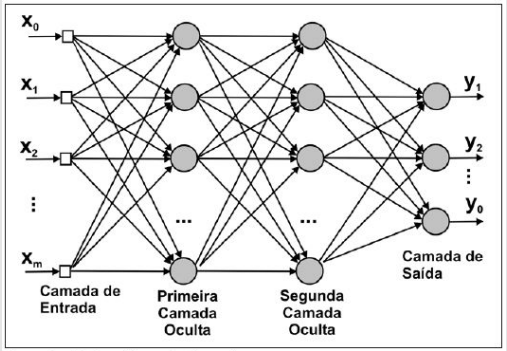


Figura 2 - Rede multilayer feedforward.

Rede Neural MLP

- Redes com duas camadas podem implementar qualquer função seja ela linearmente separável ou não [Cybenko,1989].

- A qualidade da aproximação obtida depende da complexidade da rede.

- Número de camadas, número de neurônios, funções de ativação

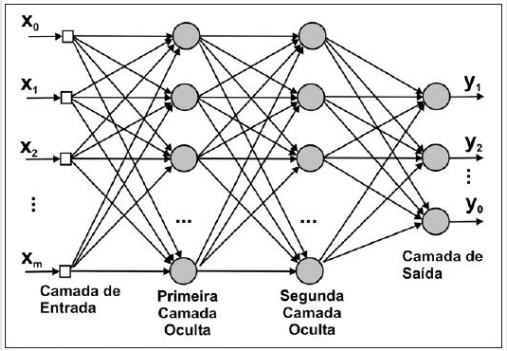


Figura 2 - Rede multilayer feedforward.

Rede Neural MLP

- Número de camadas
  - Maioria dos problemas práticos raramente precisam mais que duas camadas.
  - Primeira camada: cada neurônio contribui com retas para formação da superfície no espaço de entrada.
  - Segunda camada: cada neurônio combina as retas formando regiões convexas

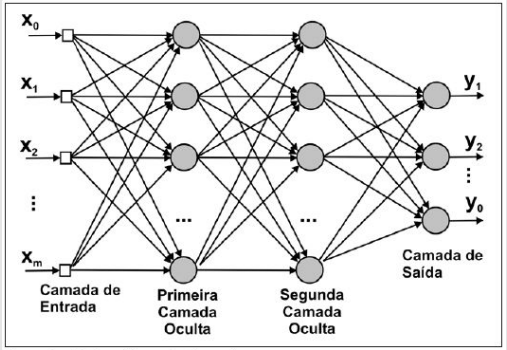
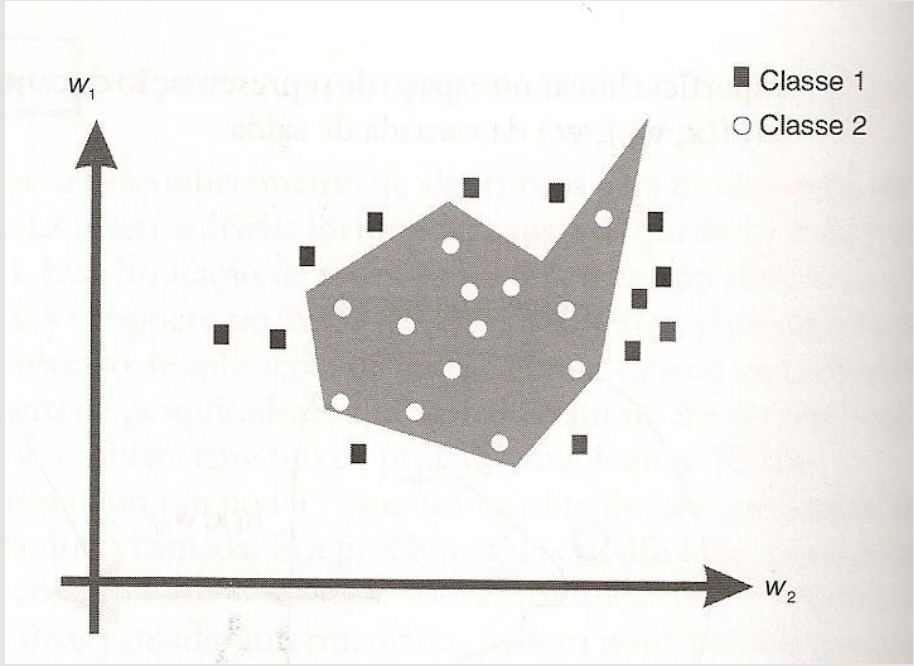


Figura 2 - Rede multilayer feedforward.

Rede Neural MLP



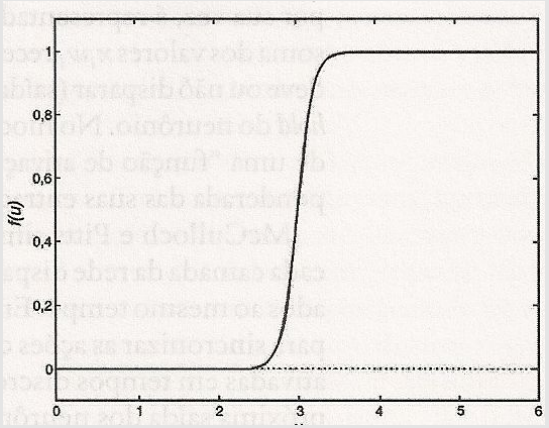
Rede Neural MLP

- Número de neurônios
  - Refere-se a capacidade de generalização da rede.
  - Quanto maior o número de neurônios, maior a capacidade de resolver problemas.
  - Não há na literatura definição formal acerca da quantidade de neurônios.
  - Empirismo: adiciona-se ou reduz-se de acordo com a medida de tolerância da rede.

Rede Neural MLP

- Funções de ativação
  - Sigmoidais nas camadas intermediárias e Lineares na camada de saída.
  - Semelhante a degrau, contudo possui região semilinear, que pode ser importante na aproximação de funções contínuas.

$$f(u) = \frac{1}{1 + e^{-\beta u}}$$



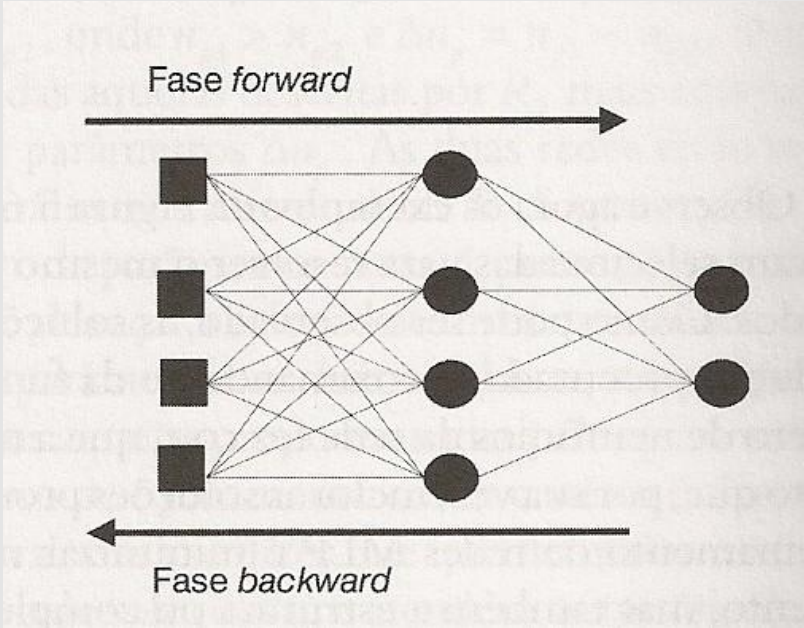
Rede Neural MLP

- Uma RNA é composta por:
  - Um conjunto de neurônios com capacidade de processamento.
  - Uma topologia de conexão que defina como estes neurônios estão conectados.
  - Uma regra de aprendizado.

- Redes MLP:
  - Diversos neurônios
  - Topologia de duas ou mais camada
  - Regra Delta Generalizada

Rede Neural MLP

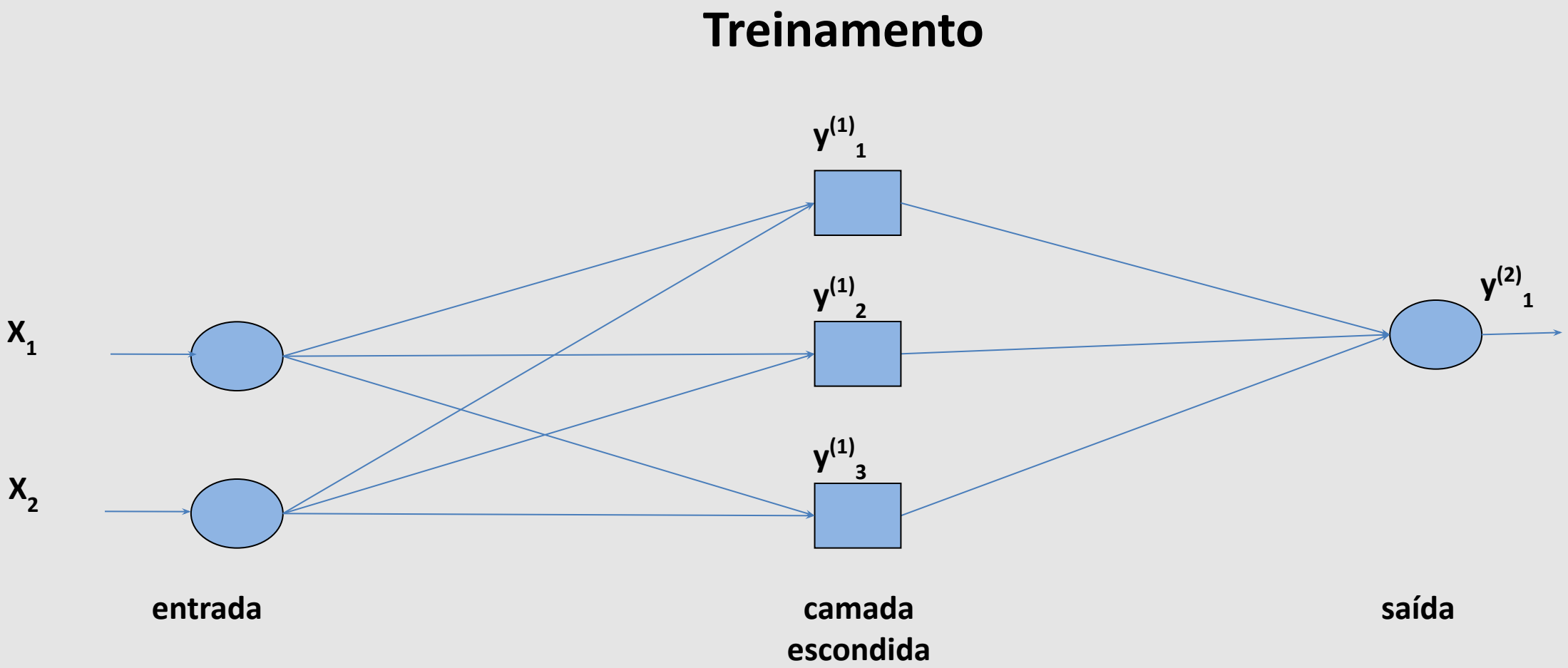
- Treinamento em duas fases:



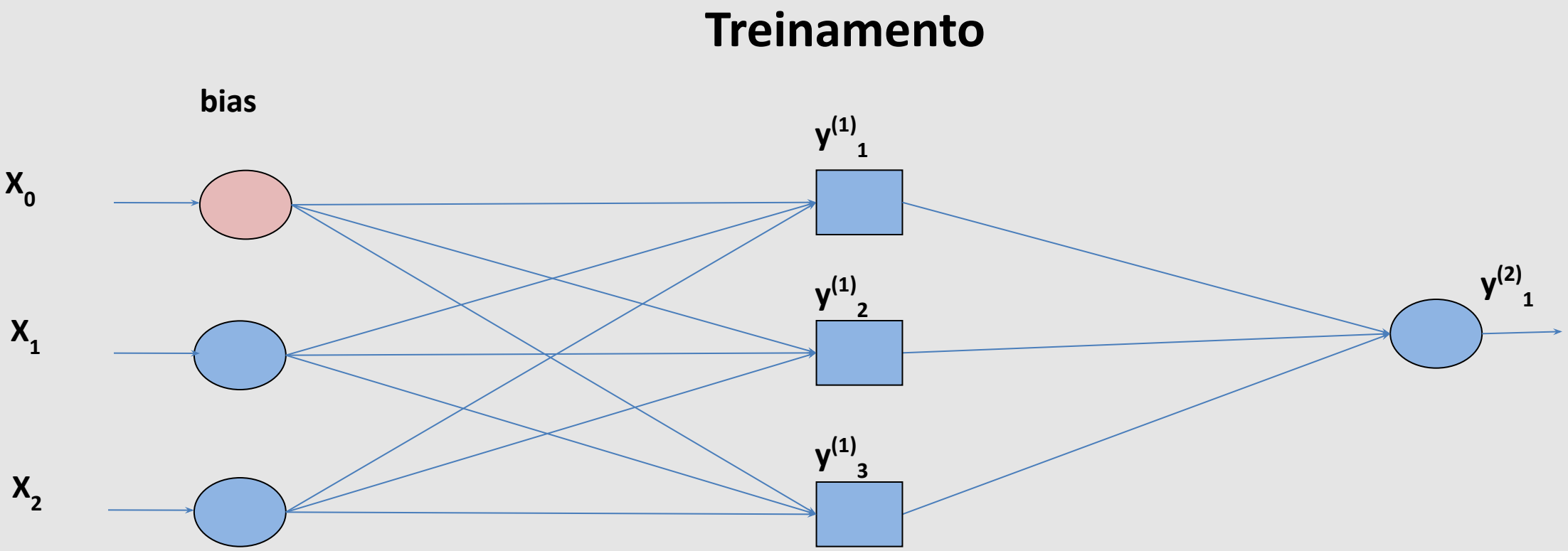
Rede Neural MLP

- ***Fase Forward***
  - Inicializar η;
  - Inicializar a matriz de pesos w com valores aleatórios;
  - Apresentar entrada à primeira camada da rede...
  - Após os neurônios da camada i calcularem seus sinais de saída, os neurônios da camada i + 1 calculam seus sinais de saída...
  - Saídas produzidas pelos neurônios da última camada são comparadas às saídas desejadas...
  - Erro para cada neurônio da camada de saída é calculado

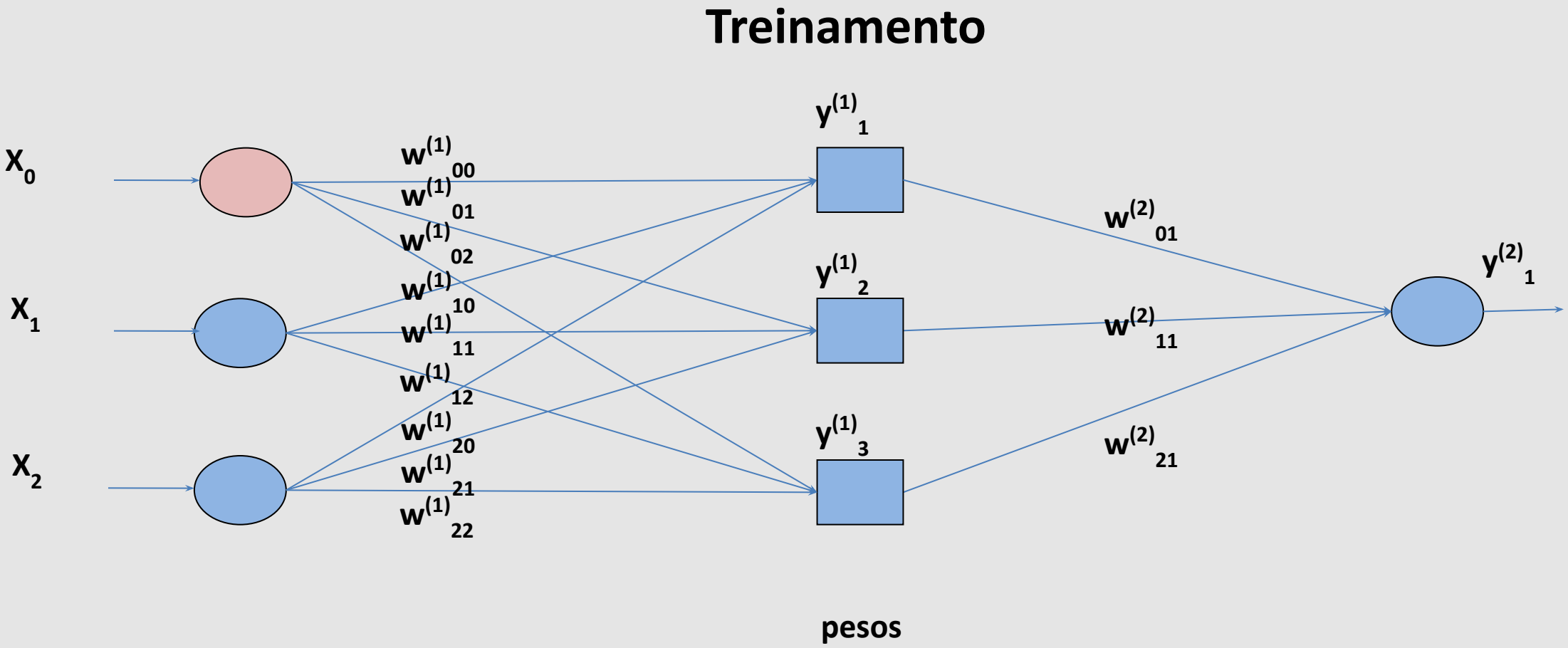
Rede Neural MLP



Rede Neural MLP



Rede Neural MLP

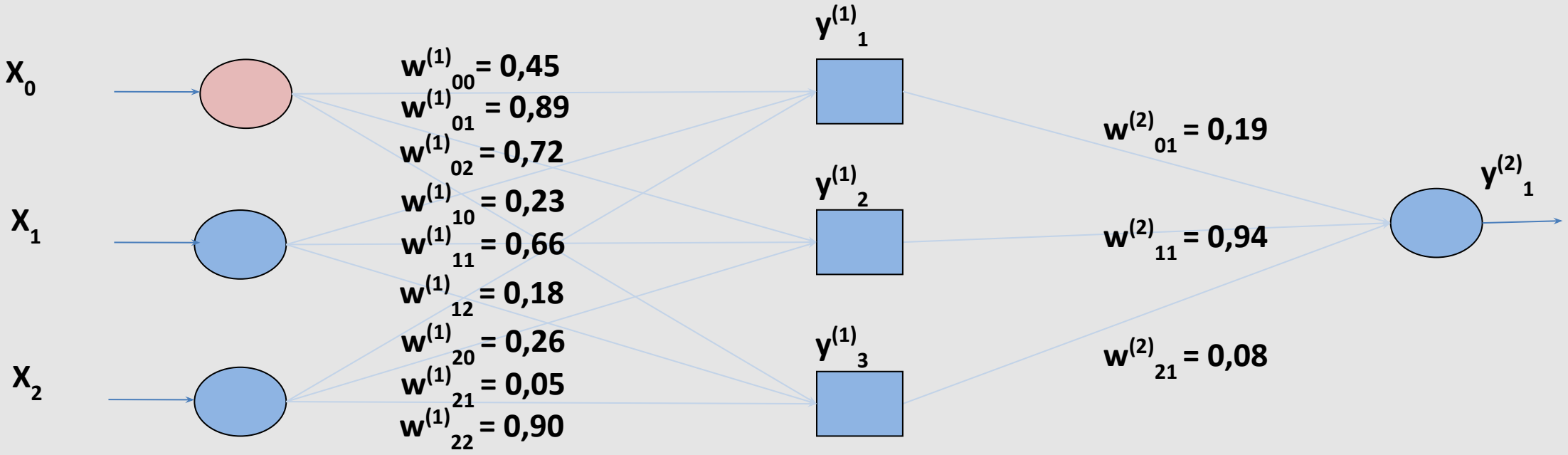


Rede Neural MLP

Treinamento

$\eta = 0,1$

1. Inicia todas as conexões com pesos aleatórios

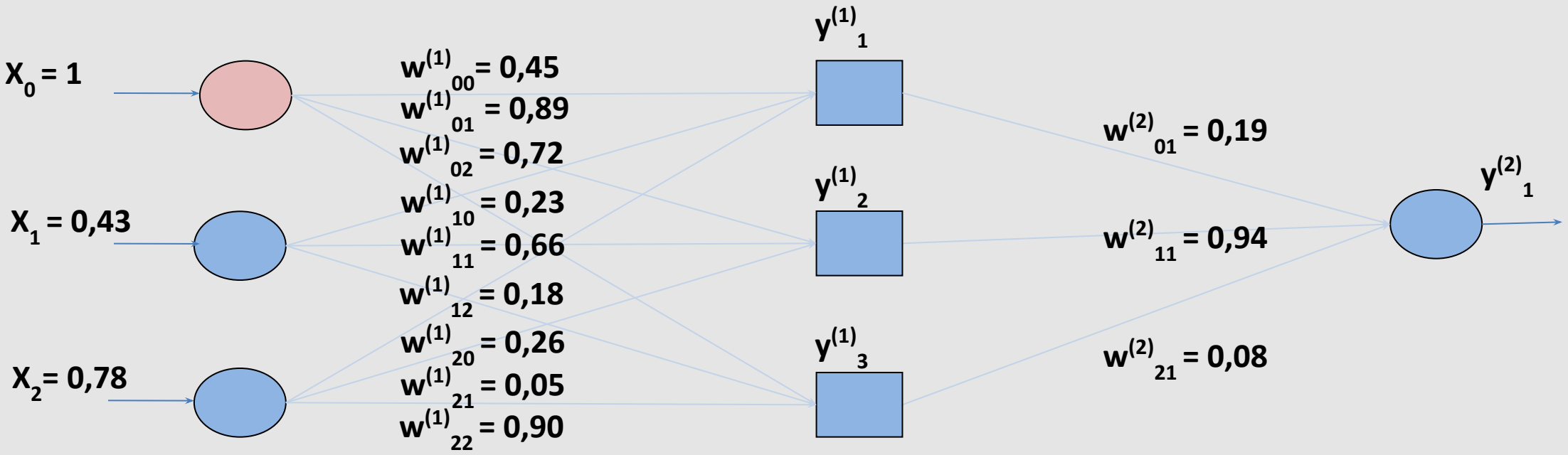


Rede Neural MLP

Treinamento

$\eta = 0,1$
$d = 1$

2. Para de entrada X é apresentado

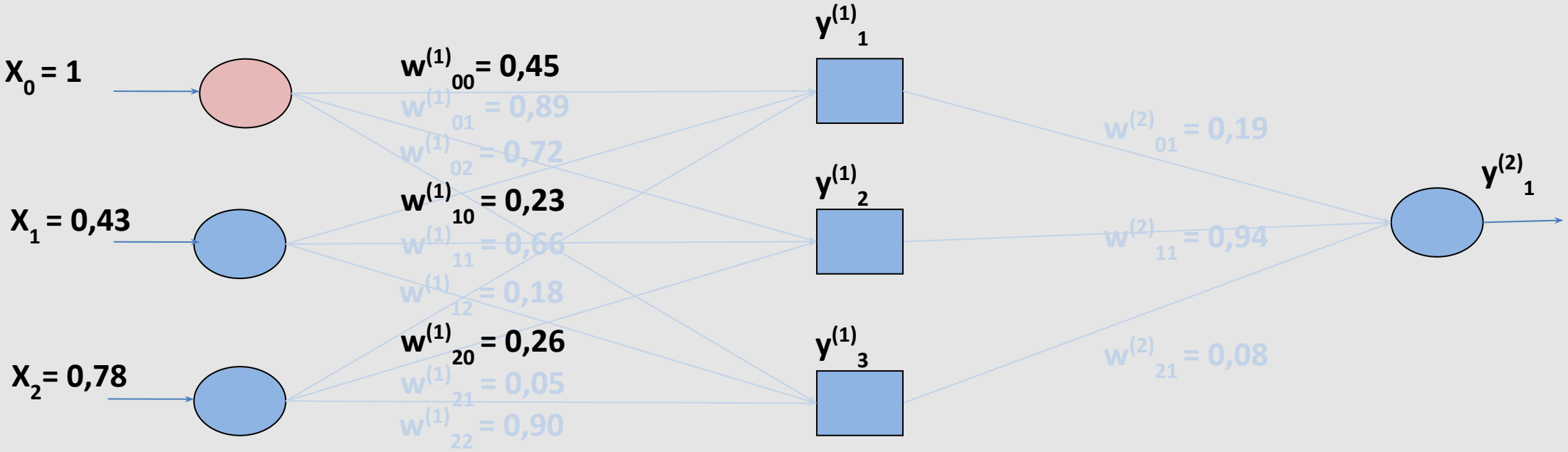


Rede Neural MLP

Treinamento

$\eta = 0,1$
$d = 1$

3. Calcula saída para 1ª camada



$u^{(1)}_1 = (1*0,45) + (0,43*0,23) + (0,78*0,26) = 0,7517$

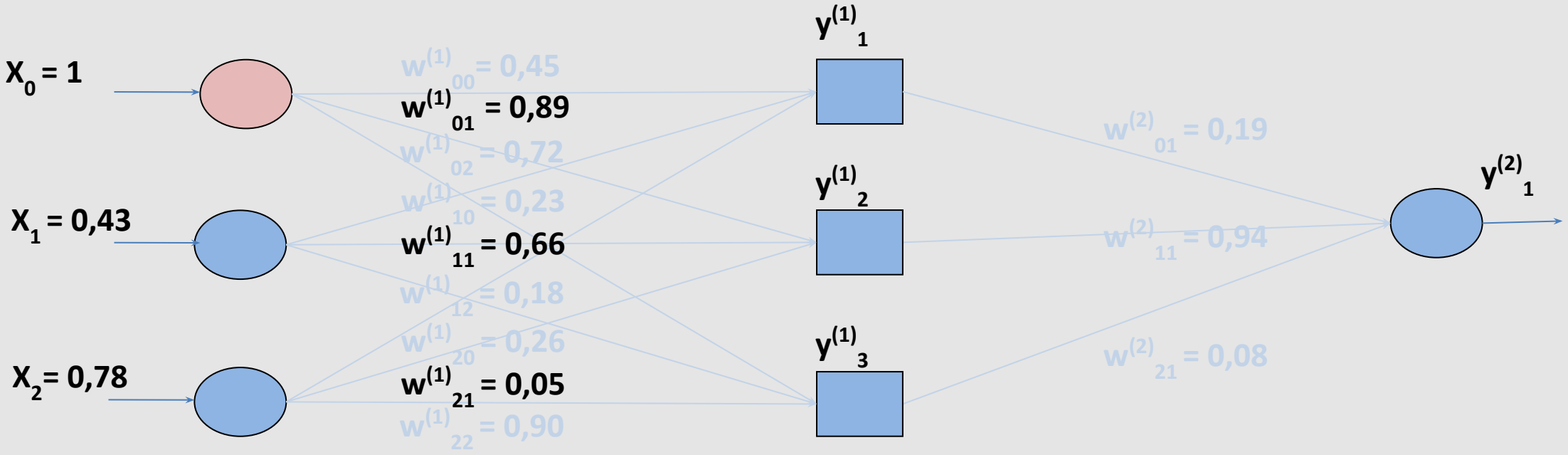
Rede Neural MLP

Treinamento

$\eta = 0,1$

$d = 1$

3. Calcula saída para 1ª camada



$u^{(1)}_1 = (1*0,45) + (0,43*0,23) + (0,78*0,26) = 0,7517$

$u^{(1)}_2 = (1*0,89) + (0,43*0,66) + (0,78*0,05) = 1,2128$

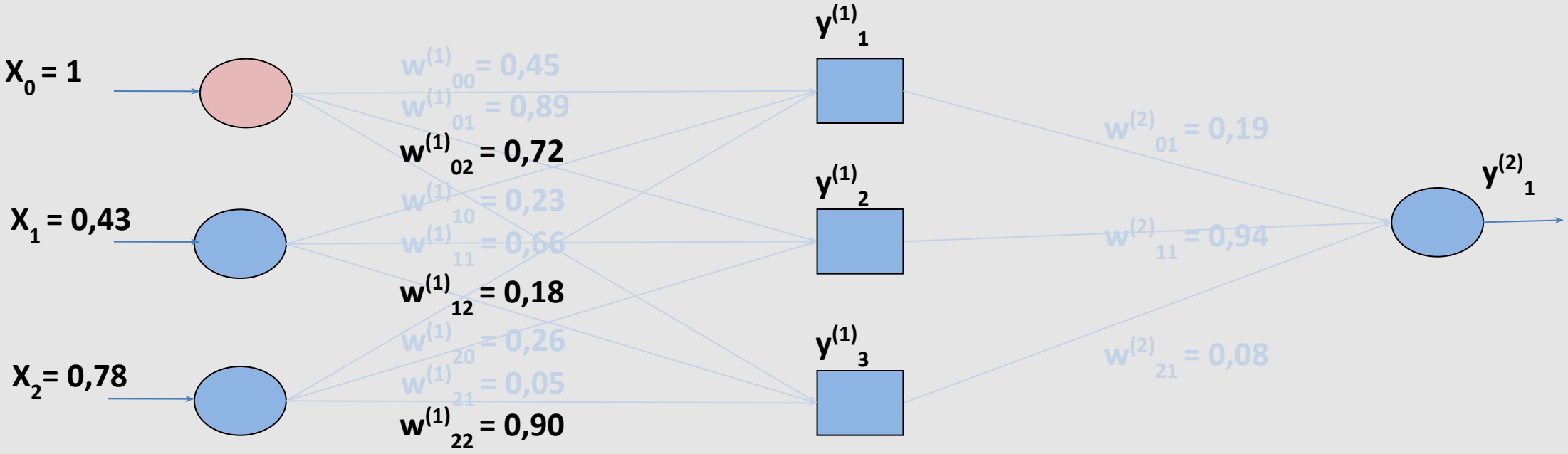
Rede Neural MLP

Treinamento

$\eta = 0,1$

$d = 1$

3. Calcula saída para 1ª camada



$u^{(1)}_1 = (1*0,45) + (0,43*0,23) + (0,78*0,26) = 0,7517$

$u^{(1)}_2 = (1*0,89) + (0,43*0,66) + (0,78*0,05) = 1,2128$

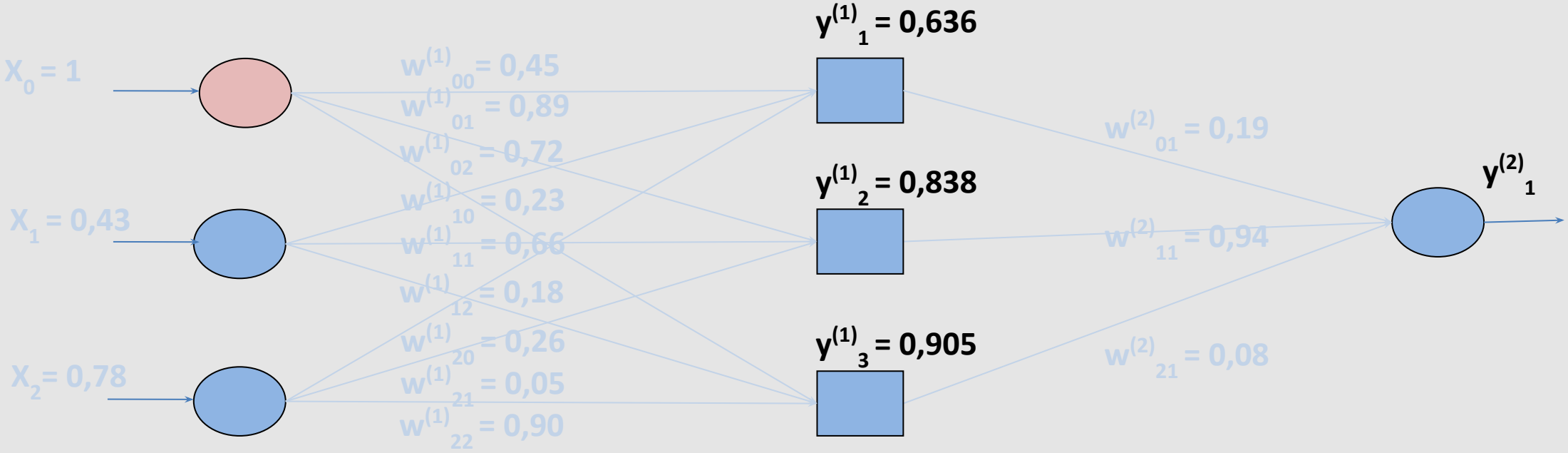
$u^{(1)}_3 = (1*0,72) + (0,43*0,18) + (0,78*0,90) = 1,4994$

Rede Neural MLP

Treinamento

$\eta = 0,1$
$d = 1$

4. Calcula saída da função de ativação



$u^{(1)}_1 = (1*0,45) + (0,43*0,23) + (0,78*0,26) = 0,7517$

$u^{(1)}_2 = (1*0,89) + (0,43*0,66) + (0,78*0,05) = 1,2128$

$u^{(1)}_3 = (1*0,72) + (0,43*0,18) + (0,78*0,90) = 1,4994$

$y^{(1)}_1 = TANH(u^{(1)}_1) = 0,636$

$y^{(1)}_2 = TANH(u^{(1)}_1) = 0,838$

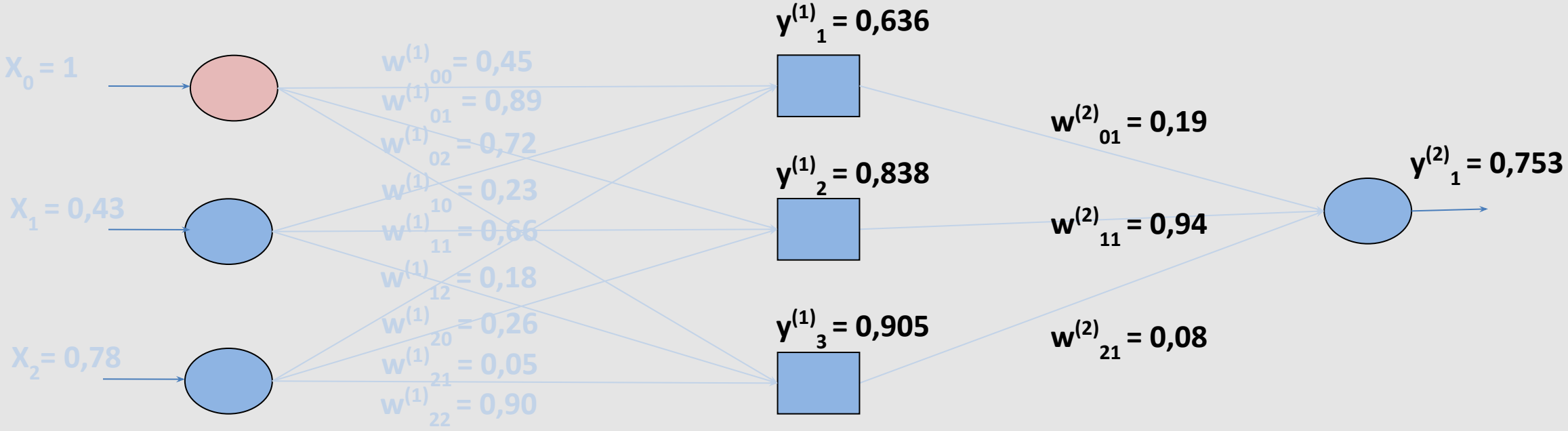
$y^{(1)}_3 = TANH(u^{(1)}_1) = 0,905$

Rede Neural MLP

$\eta = 0,1$

$d = 1$

5. Calcula saída para 2ª camada



$u^{(2)}_1 = (0,636*0,19) + (0,838*0,94) + (0,905*0,08) = 0,981$

$y^{(2)}_1 = TANH(u^{(2)}_1) = 0,753$

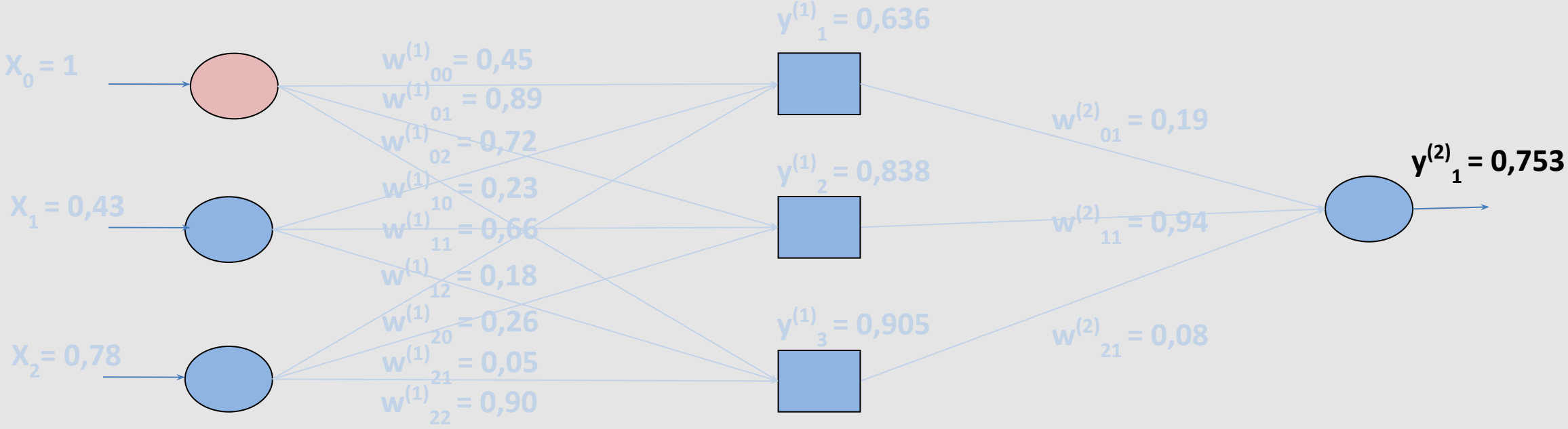
Rede Neural MLP

Treinamento

$\eta = 0,1$

$d = 1$

6. Calcula variação do erro



$$E(k) = 1/2 \sum_{i=0}^{n} (d_i(t) - y_i(t))^2$$

$$E(k) = ½ (0,247)^2 = 0,03$$

Rede Neural MLP

- ***Fase Backward***
  - A partir da última camada
    - O nó ajusta seu peso de modo a reduzir o seu erro
    - Nós das camadas anteriores tem seu erro definidos por:
      - Erros dos nós da camada seguinte conectados a ele ponderados pelos pesos das conexões entre eles

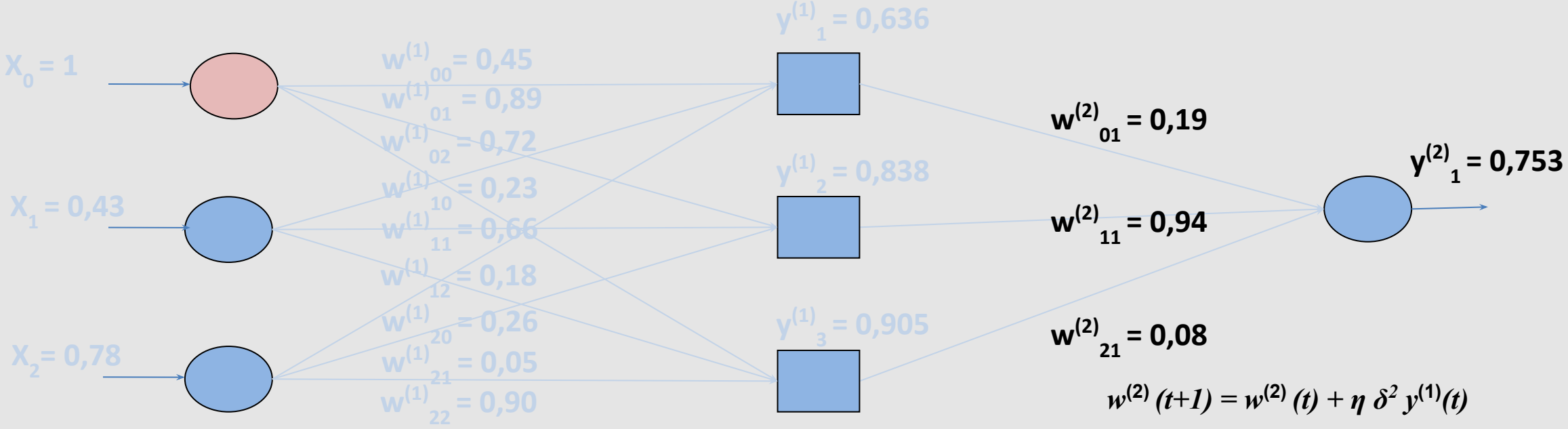
Rede Neural MLP

Treinamento

$\eta = 0,1$

$d = 1$

7. Calcula variação dos pesos da 2ª camada



$$w^{(2)}(t+1) = w^{(2)}(t) + \eta\, \delta^2\, y^{(1)}(t)$$

$$\delta^{(2)}(t) = (d(t) - y(t)) * y'^{(2)}_1$$

$$\delta^{(2)}(t) = 0,247 * \text{ATANH}(0,753)$$

$$\delta^{(2)}(t) = 0,247 * 0,981 = 0,2423$$

	$w^{(2)}(t)$	η	$\delta^{(2)}$	$x(t)$	$w^{(2)}(t+1)$
$w^{(2)}_{01}$	0.19	0.1	0.2423	0.636	0.2054
$w^{(2)}_{11}$	0.94	0.1	0.2423	0.838	0.9603
$w^{(2)}_{21}$	0.08	0.1	0.2423	0.905	0.1019

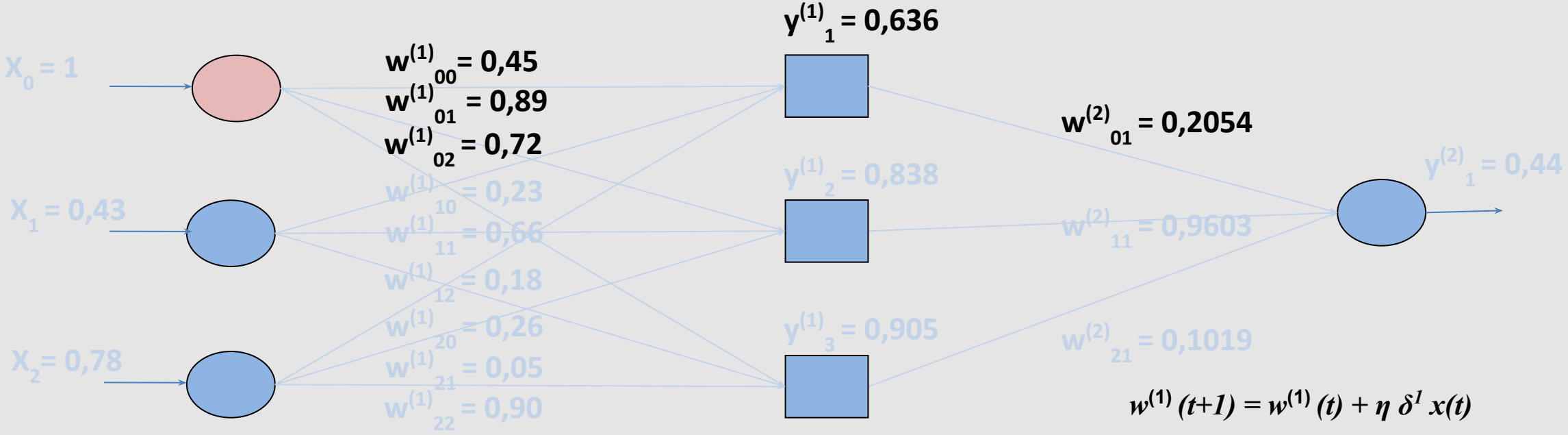
Rede Neural MLP

Treinamento

$\eta = 0,1$

$d = 1$

7. Calcula variação dos pesos da 1ª camada



$$\delta^{(1)}_1(t) = \left(\sum_{k=1}^{n} \delta^{(2)} * W_{kj}^{(2)}\right) * y'^{(1)}_1$$

$$\delta^{(1)}_1(t) = (0,2423 * 0,2054) * \text{ATANH}(0,636) = 0,0374$$

$$w^{(1)}(t+1) = w^{(1)}(t) + \eta\, \delta^1\, x(t)$$

	$w^{(1)}(t)$	η	$\delta^{(1)}$	$x(t)$	$w^{(1)}(t+1)$
$w^{(1)}_{01}$	0.45	0.1	0.0374	1	0.4537
$w^{(1)}_{11}$	0.89	0.1	0.0374	0.43	0.8916
$w^{(1)}_{21}$	0.72	0.1	0.0374	0.78	0.7229

ATITUS EDUCAÇÃO

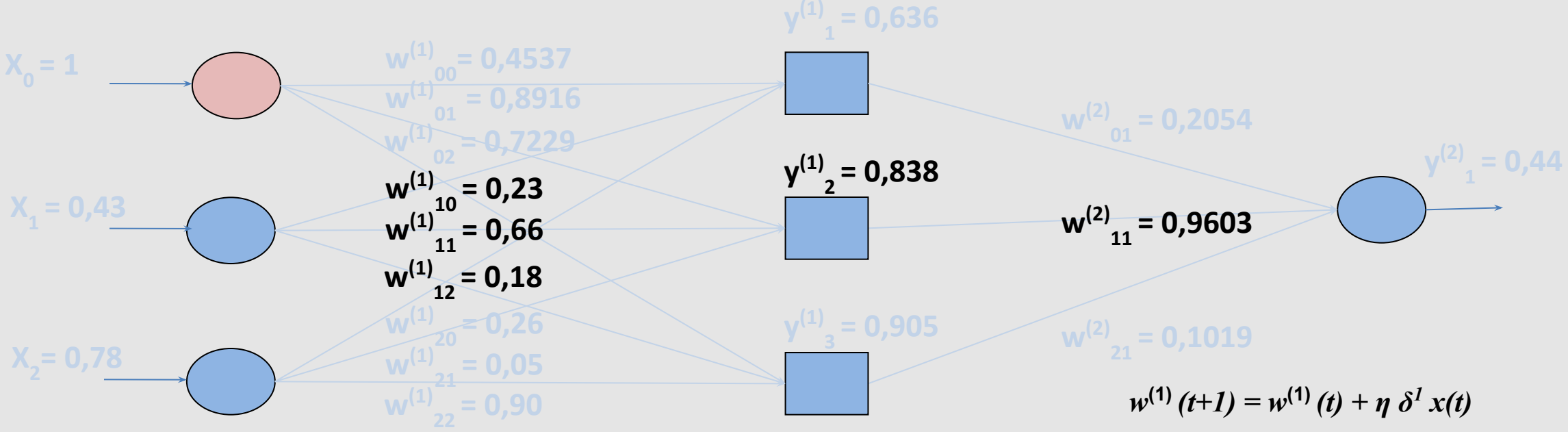
Rede Neural MLP

Treinamento

$\eta = 0,1$

$d = 1$

7. Calcula variação dos pesos da 1ª camada



$$w^{(1)}(t+1) = w^{(1)}(t) + \eta\, \delta^{1}\, x(t)$$

$$\delta^{(1)}{}_2(t) = \left(\sum_{k=1}^{n} \delta^{(2)} * W_{kj}^{(2)}\right) * y'^{(1)}{}_1$$

$$\delta^{(1)}{}_2(t) = (0,2423 * 0,9603) * \text{ATANH}(0,838) = 0,2821$$

	$w^{(1)}(t)$	η	$\delta^{(1)}$	$x(t)$	$w^{(1)}(t+1)$
$w^{(1)}{}_{01}$	0.23	0.1	0.2821	1	0.2582
$w^{(1)}{}_{11}$	0.66	0.1	0.2821	0.43	0.6721
$w^{(1)}{}_{21}$	0.18	0.1	0.2821	0.78	0.2020

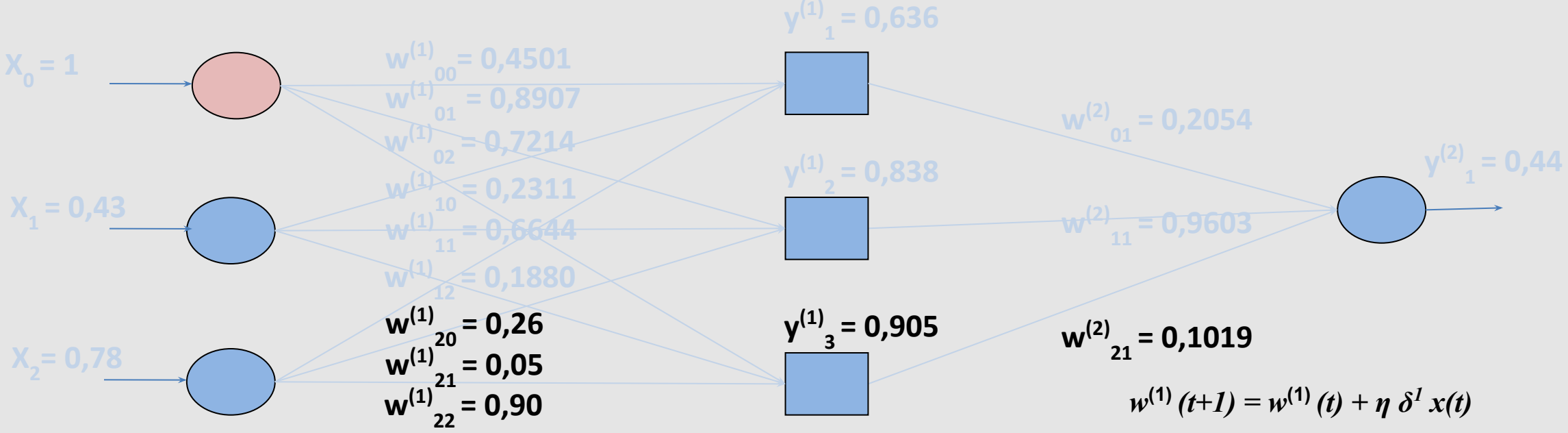
Rede Neural MLP

Treinamento

$\eta = 0,1$

$d = 1$

7. Calcula variação dos pesos da 1ª camada



$$\delta^{(1)}_3(t) = \left(\sum_{k=1}^{n} \delta^{(2)} * W_{kj}^{(2)}\right) * y'^{(1)}_1$$

$$\delta^{(1)}_3(t) = (0,2423 * 0,1019) * \text{ATANH}(0,905) = 0,0370$$

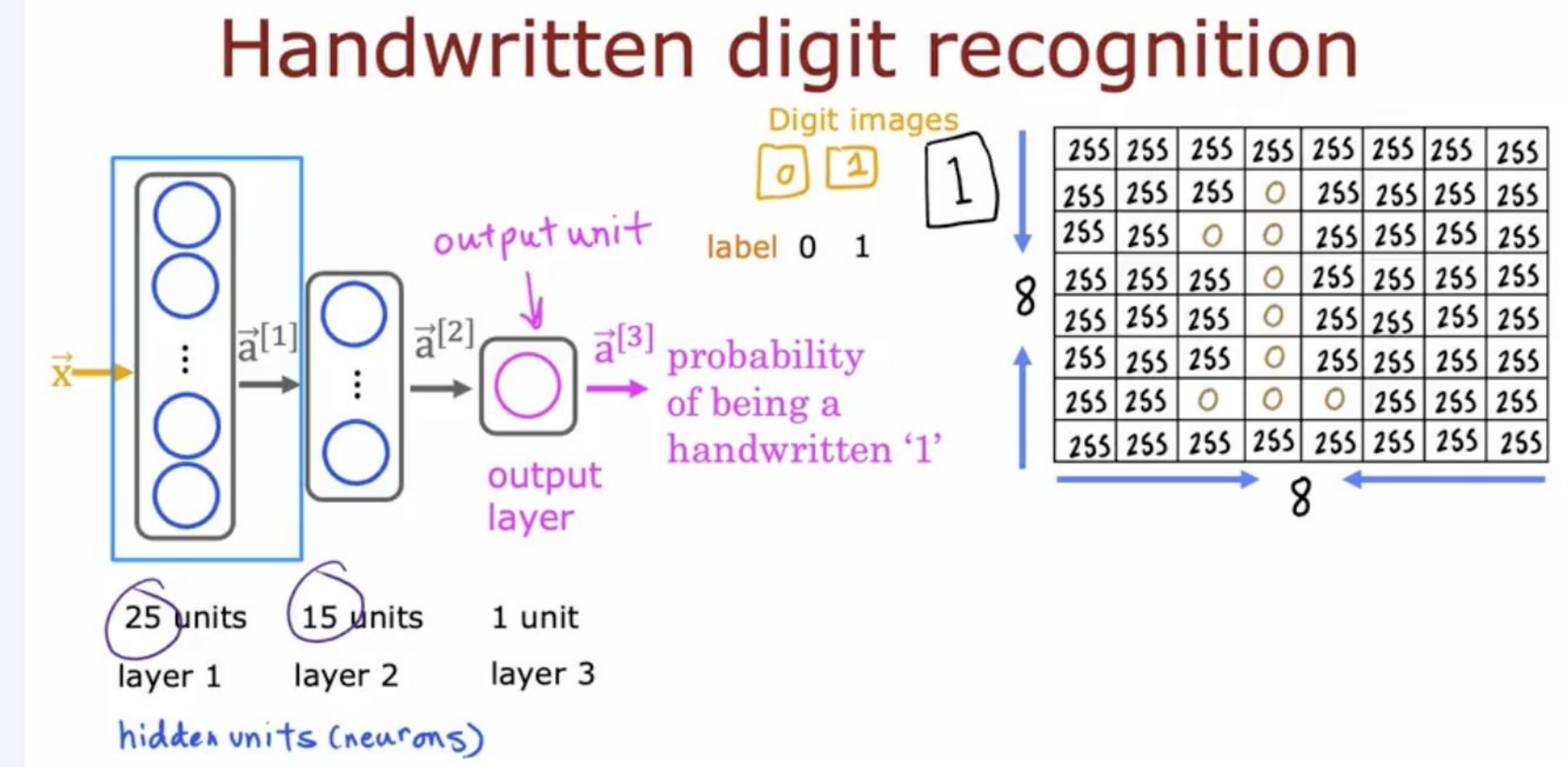
$$w^{(1)}(t+1) = w^{(1)}(t) + \eta\, \delta^{1}\, x(t)$$

	$w^{(1)}(t)$	η	$\delta^{(1)}$	$x(t)$	$w^{(1)}(t+1)$
$w^{(1)}_{01}$	0.26	0.1	0.0370	1	0.2637
$w^{(1)}_{11}$	0.05	0.1	0.0370	0.43	0.0515
$w^{(1)}_{21}$	0.90	0.1	0.0370	0.78	0.9028

Rede Neural MLP

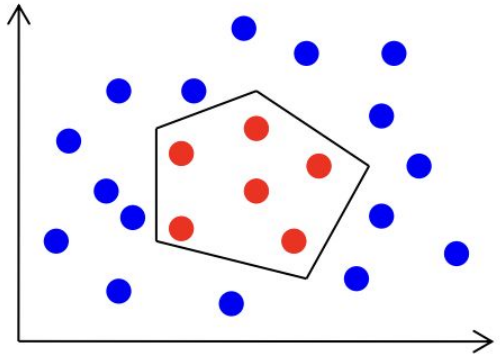
8. Repetir até k = número de interações desejada ou Erro = erro aceitável

Rede Neural MLP

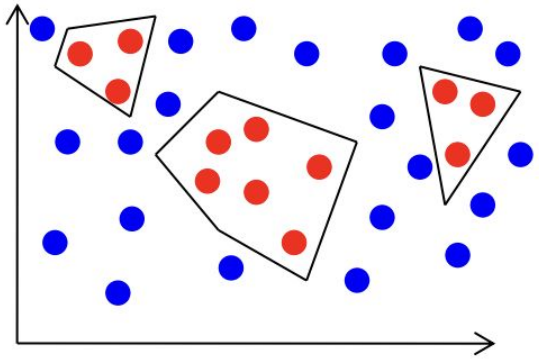


Rede Neural: treinando um neurônio

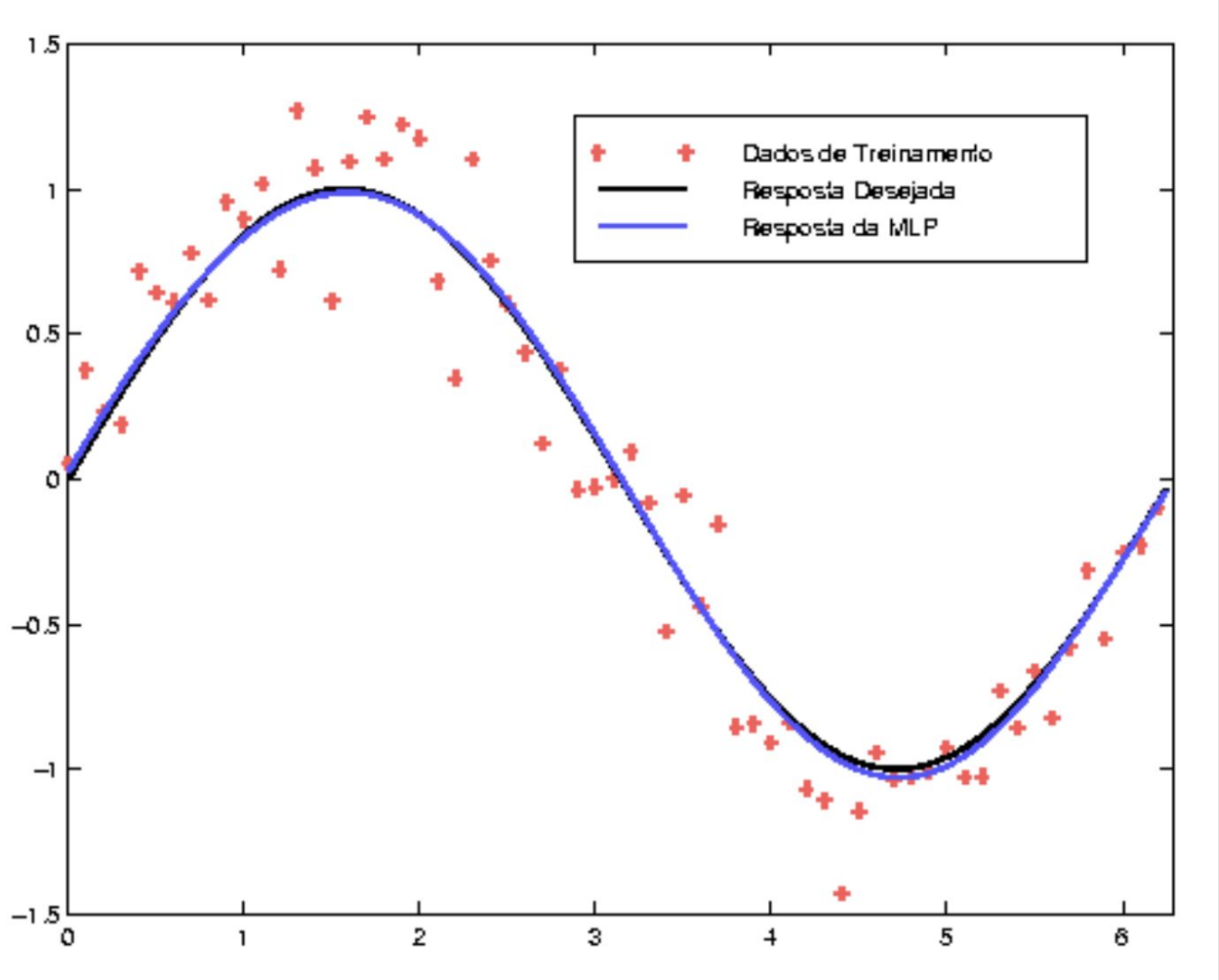
- Adicionar uma camada oculta a rede permite que a rede possa gerar uma função de convex hull.



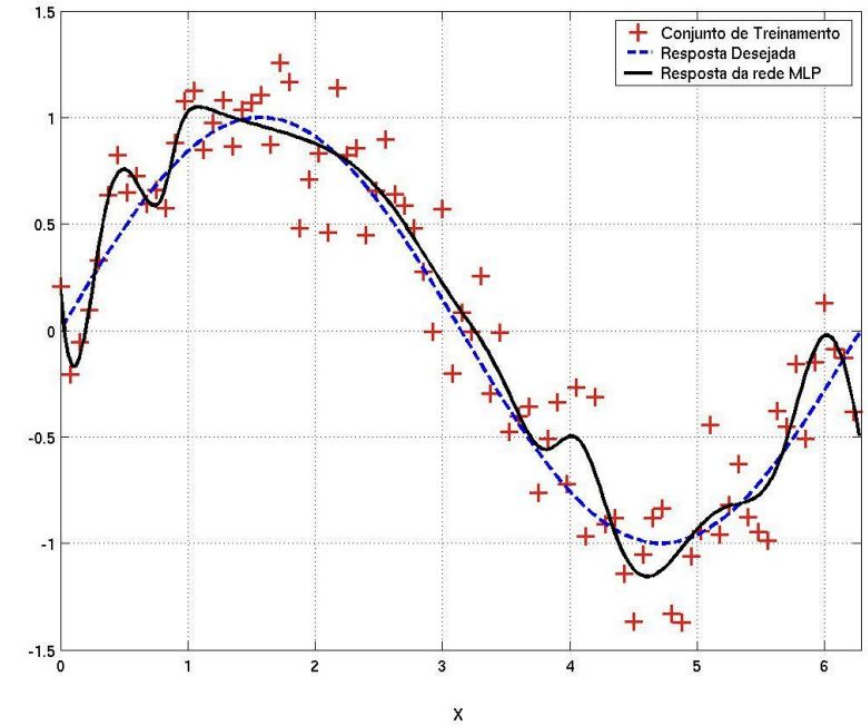
- Duas camadas ocultas permite a rede gerar um função com diferentes convex hulls.

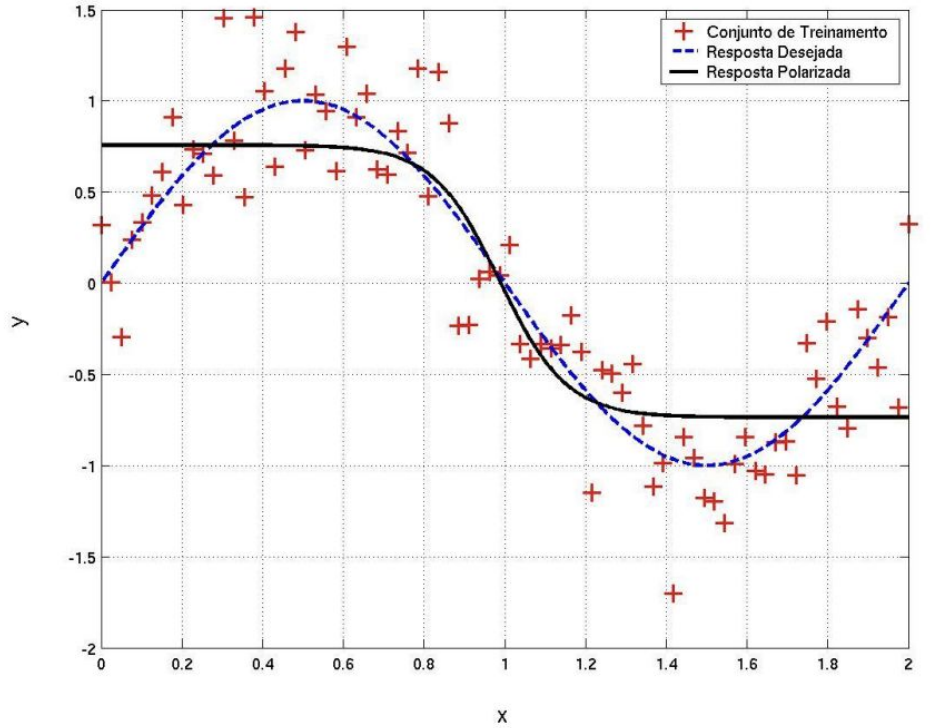


Rede Neural: exemplo



Rede Neural: exemplo





Overfitting
Variância

Underfitting
Vício

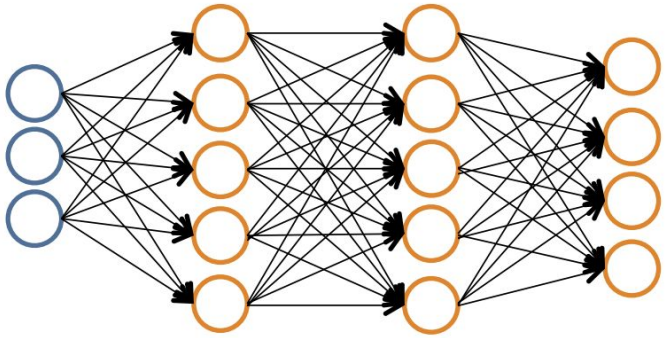
Rede Neural: múltiplas saídas

Pedestrian

Car

Motorcycle

Truck



$$h_\Theta(\mathbf{x}) \in \mathbb{R}^K$$

We want:

$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

when pedestrian

$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

when car

$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

when motorcycle

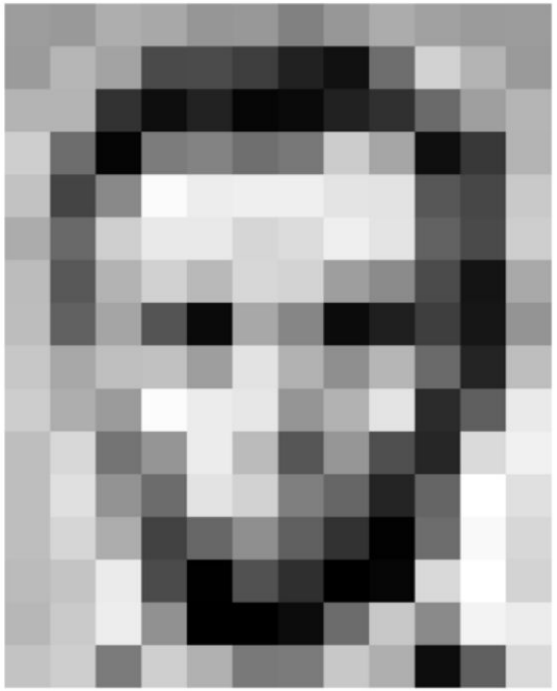
$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

when truck

Rede Neural: Imagens

What computers 'see': Images as Numbers

What you see



Input Image

What you both see

157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

Input Image + values

What the computer "sees"

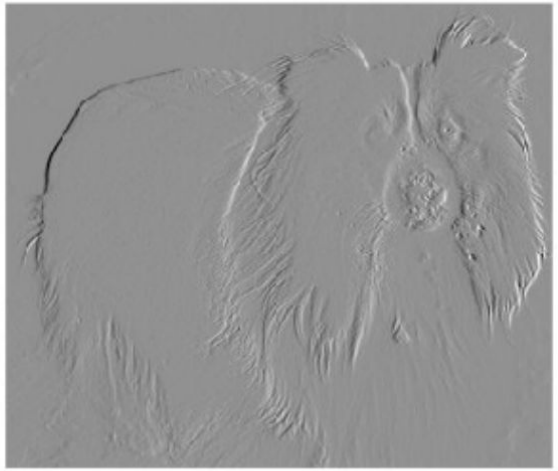
157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

Pixel intensity values
("pix-el"=picture-element)

Levin *Image Processing & Computer Vision*

Rede Neural: Imagens

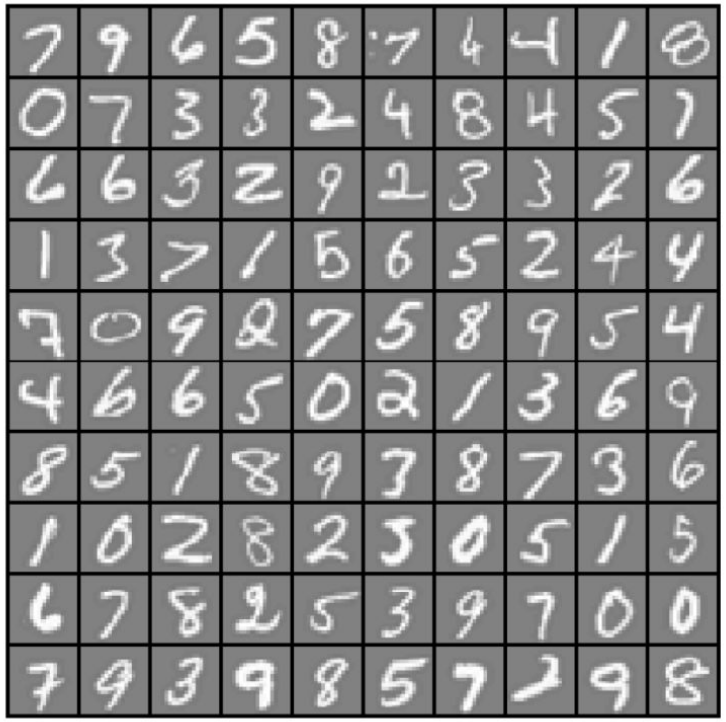
Input

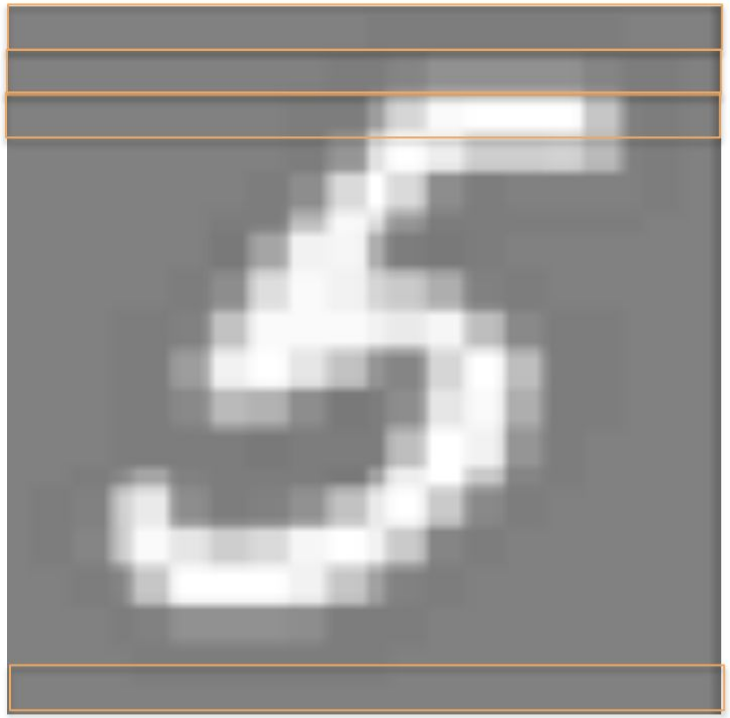


Output

1	-1

Filter



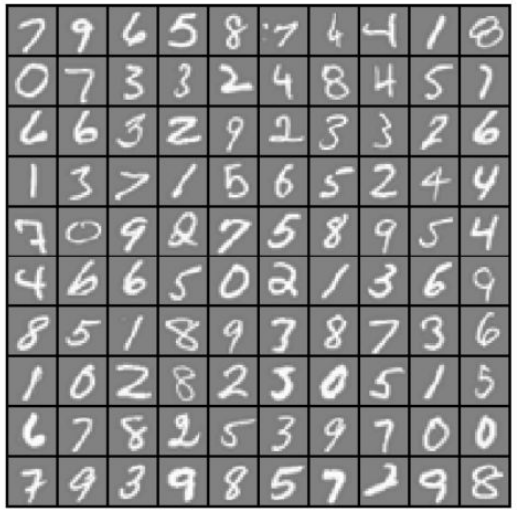


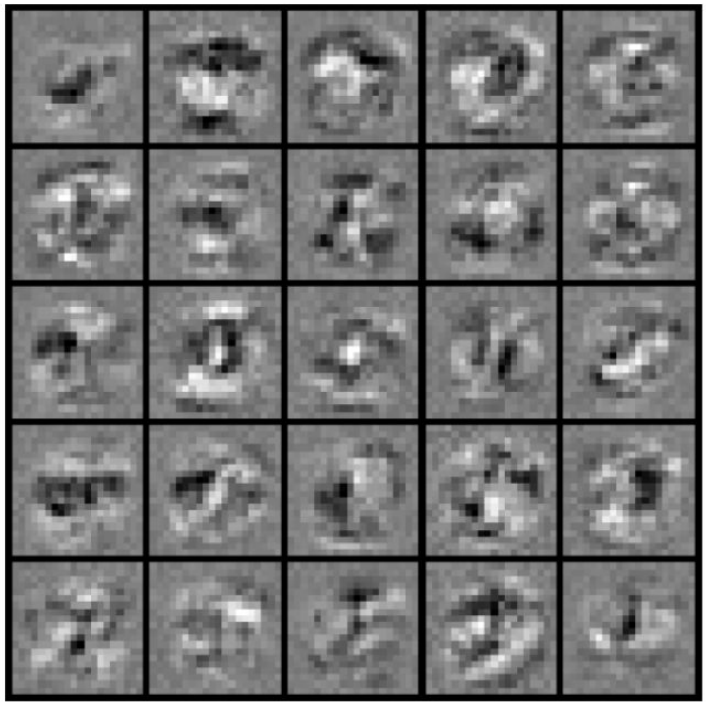
$x_1 \dots x_{20}$
$x_{21} \dots x_{40}$
$x_{41} \dots x_{60}$

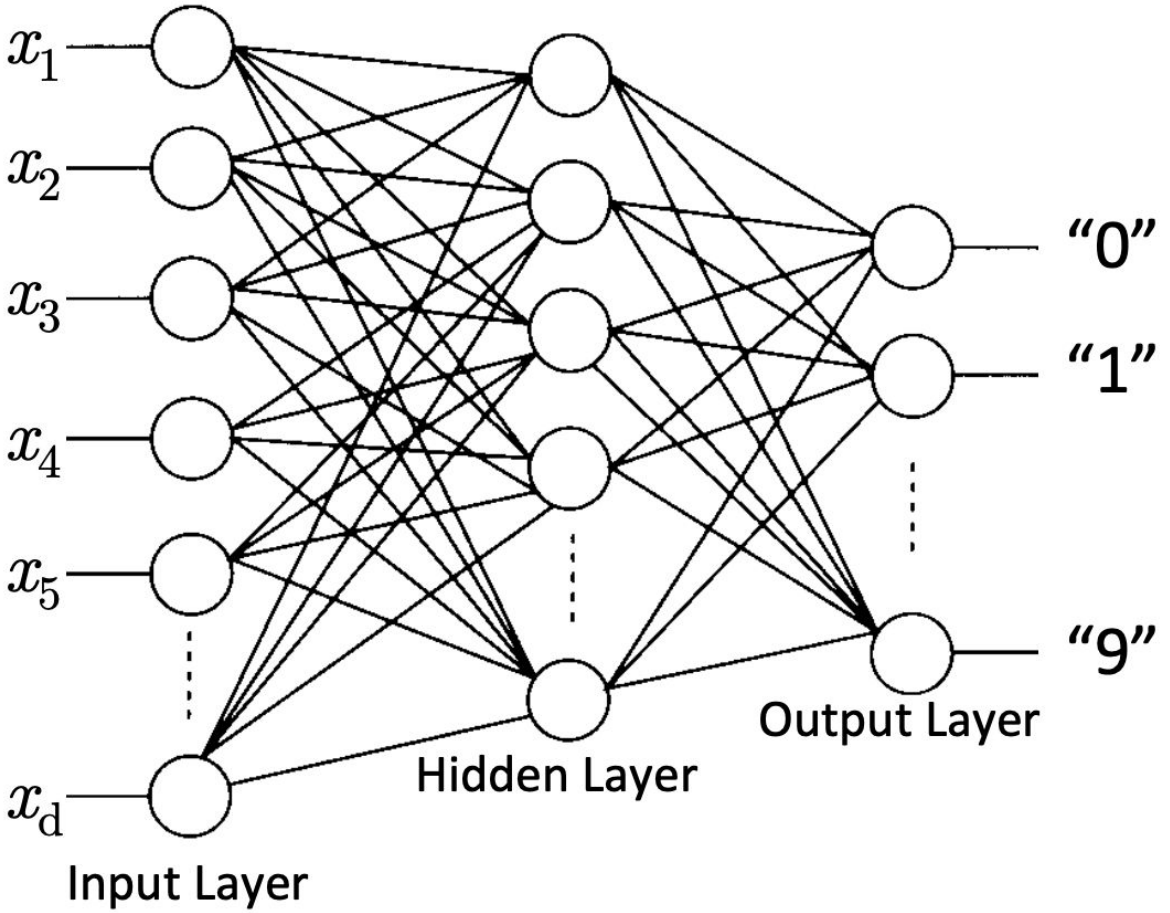
$x_{381} \dots x_{400}$

20 × 20 pixel images
d = 400 10 classes

Each image is “unrolled” into a vector $\mathbf{x}$ of pixel intensities







Visualization of Hidden Layer

Atividade

- Jupyter notebook
  - Google colab
  - Instalação local (de preferência Unix)
- Python (kernel)
- Exercício 1 (forward pass)
  - Varie os dados de entrada e os tamanhos da rede
  - Analise a arquitetura
  - Faça um debugging
  - Varie o número de amostras "n" na última célula. Qual o impacto?

Atividade

- Exercício 2 (modelo treinado)
  - Varie os atributos utilizados
  - Mude o tamano do conjunto de teste e a quantidade de épocas
  - Mude a função de ativação (relu, tanh, sigmoid)
- Exercício 3 (análise de rede)
  - Siga as instruções no notebook e responda as perguntas
- Exercício 4 (comparação)
  - Tente mudar o dataset
  - Compare os algoritmos

OBRIGADO

Ciência da
Computação

Passo Fundo / Porto Alegre / Ijuí