Introdução às Redes Neuronais

Um Percurso da Teoria à Prática

Diogo Freitas
diogo.freitas@iti.larsys.pt
@DiogoNTFreitas

18 de dezembro de 2020

GDG Madeira

ORDEM DOS ENGENHEIROS

Agenda

1. Introdução
2. Redes Neuronais Artificiais
3. *Forward Propagation*
4. *Backpropagation*
5. Algumas Regras de Polegar
6. *Overfitting* vs. *Underfitting*
7. *Pipeline* Geral

Introdução

Introdução

As **Redes Neuronais Artificiais (RNAs)** foram criadas para **permitir que as máquinas** se comportem como **agentes inteligentes**.

Elas *imitam* a forma de como os nossos **neurónios se interconectam**, bem como na **forma de como aprendemos** – inferência indutiva.

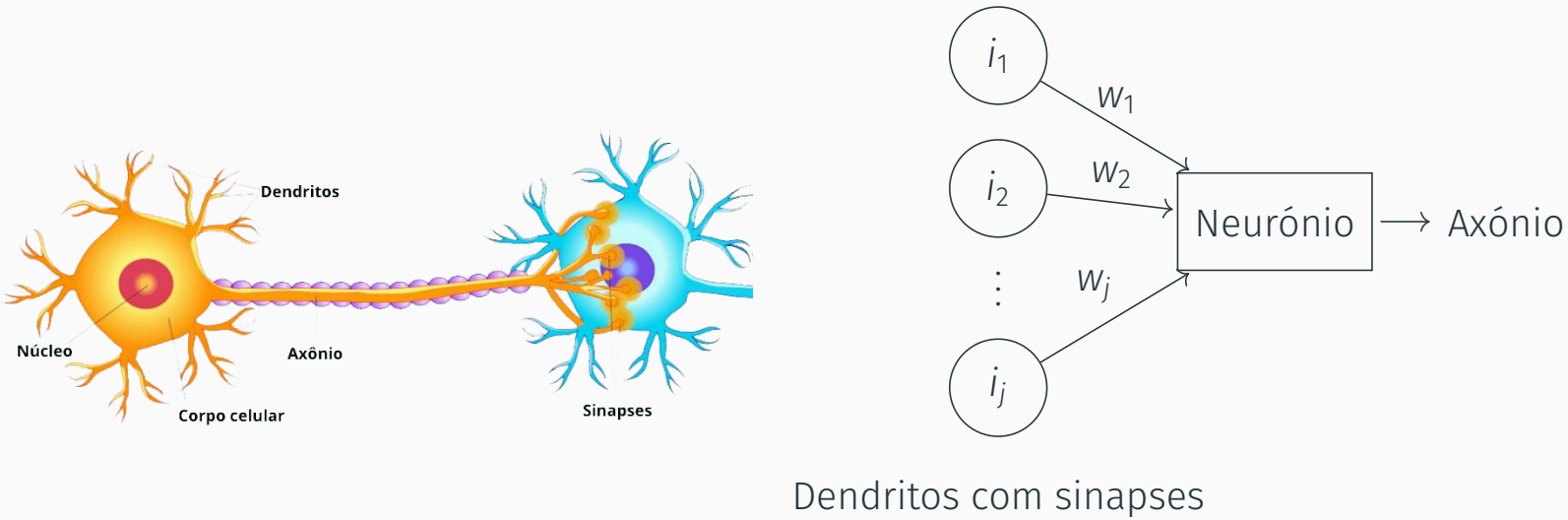


Figura 1: Rede neuronal biológica (esquerda) e rede neuronal artificial (direita).

Introdução (cont.)

McCulloch e Pitts são considerados os *pais* das RNAs.

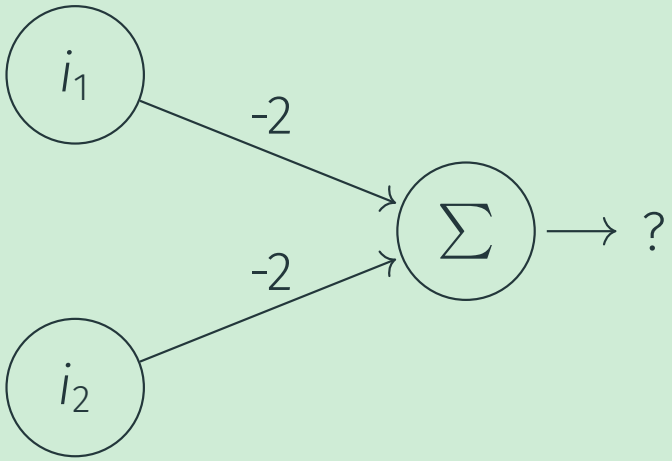
De acordo com este autores, os neurónios faziam **uma soma ponderada (w) das suas entradas (i)** para implementar qualquer função lógica.

Nos finais dos anos 50 e começo dos anos 60, **Rosenblatt introduziu o perceptrão**.

Nesse modelo, os neurónios faziam uma **soma ponderada** das suas entradas, mas o ***output*** **($\hat{y}$) era binário** quando comparado com um limiar, de tal forma que:

$$\hat{y} = \begin{cases} 0, \text{ se } \sum_j w_j i_j \leq \text{limiar}, \\ 1, \text{ caso contrário.} \end{cases} \tag{1}$$

Exemplo (Calcular o *output* de um perceptrão)



Primeiro caso: $i_1 = i_2 = 1$

$$\sum_{j=1}^{2} w_j i_j = w_1 i_1 + w_2 i_2$$

$$= (-2) \times 1 + (-2) \times 1 = -4.$$

Se considerarmos um limiar de -3, os *outputs* vêm:

Segundo caso: $i_1 = 1$ e $i_2 = 0$, obtemos -2.

Terceiro caso: $i_1 = 0$ e $i_2 = 1$, obtemos -2.

Quarto caso: $i_1 = i_2 = 0$, obtemos 0.

Caso	i_1	i_2	*Output*
1	1	1	1
2	1	0	0
3	0	1	0
4	0	0	0

Introdução (cont.)

Vamos, agora, **alterar um pouco a notação**, e considerar $b = -\text{limiar}$. Desta forma, o *output* de um perceptrão vem:

$$\hat{y} = \begin{cases} 0, \text{ se } \sum_j w_j i_j + b \leq 0, \\ 1, \text{ caso contrário.} \end{cases} \tag{2}$$

O termo b é conhecido como *bias*.

Se considerarmos $\mathbf{W} = [w_1, w_2, \cdots, w_j]$ e $\mathbf{I} = [i_1, i_2, \cdots, i_j]$, a Equação (2) pode ser reescrita, de tal forma que:

$$\hat{y} = \begin{cases} 0, \text{ se } \mathbf{W} \cdot \mathbf{I} + b \leq 0, \\ 1, \text{ caso contrário,} \end{cases} \tag{3}$$

onde $\cdot$ denota o produto interno.

Introdução (cont.)

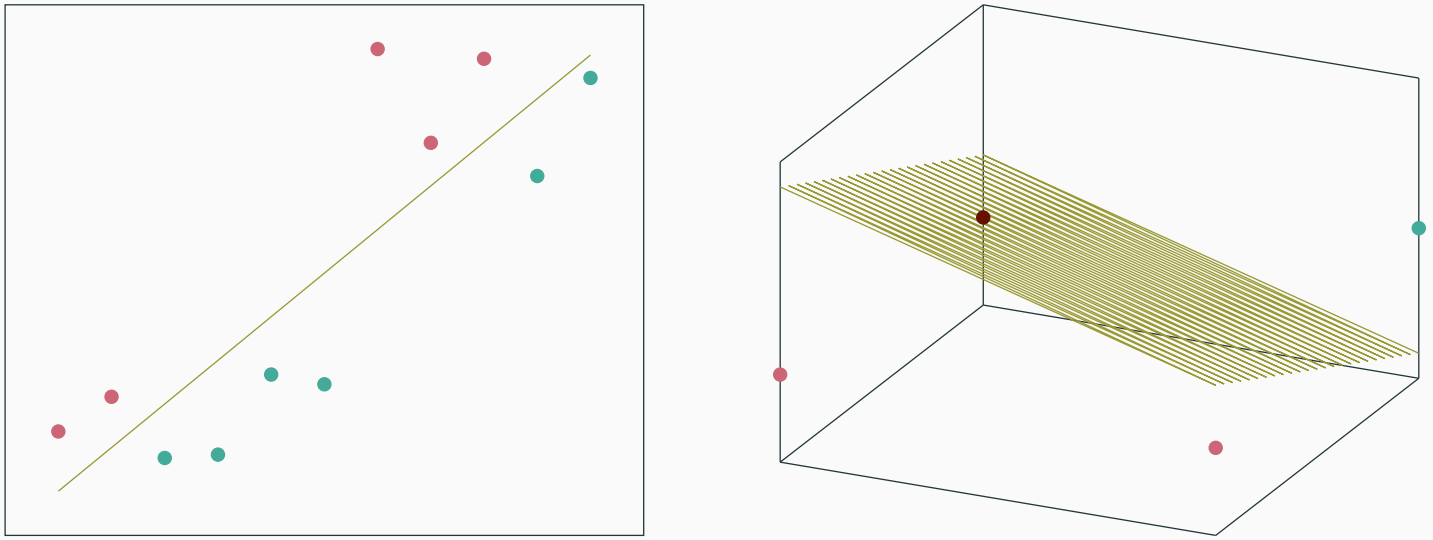


Figura 2: Recta (esquerda) e plano (direita) de regressão que se podem obter de um perceptrão.

Contudo, nós só sabemos as entradas e os *outputs* esperados.

É necessário **ajustar automaticamente os pesos e os *bias* de um perceptrão** – algoritmos de aprendizagem.

Introdução (cont.)

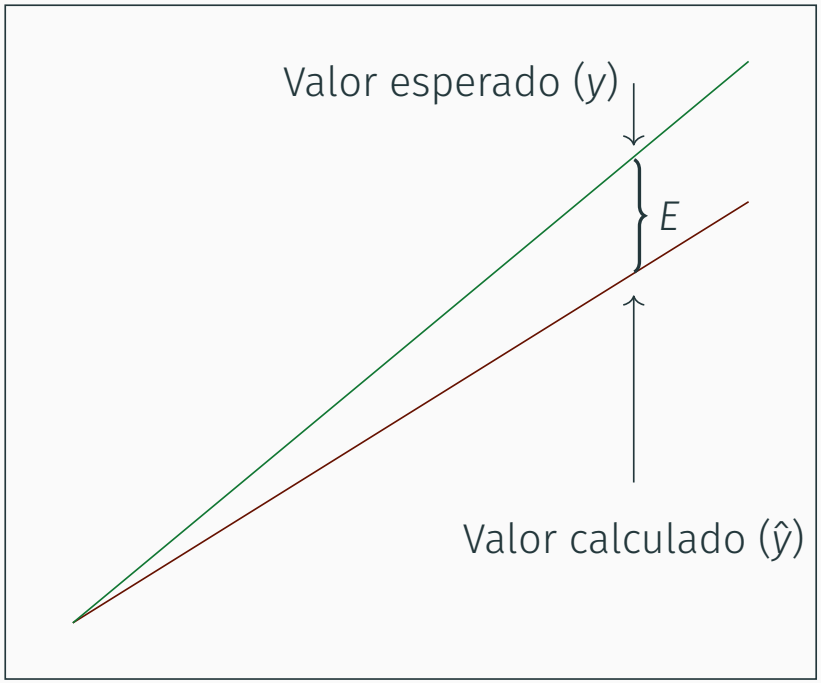


Figura 3: Visualização gráfica do cálculo do erro (E).

O erro (E) é dado por $E = y - \hat{y}$.

Este **erro é depois usado pelo algoritmo de aprendizagem** para ajustar a linha de modo a minimizar o erro.

Introdução (cont.)

Podemos escrever o *output* do percepção da seguinte forma:

$$y = (\mathsf{W} + \Delta\mathsf{W}) \cdot \mathsf{I}, \tag{4}$$

onde Δ denota um factor de mudança.

Assim,

$$\begin{aligned} E &= y - \hat{y} \\ &= (\mathsf{W} + \Delta\mathsf{W}) \cdot \mathsf{I} - \mathsf{W} \cdot \mathsf{I} \\ &= (\Delta\mathsf{W}) \cdot \mathsf{I}, \end{aligned} \tag{5}$$

e, desta forma,

$$\Delta\mathsf{W} = \frac{E}{\mathsf{I}}. \tag{6}$$

Contudo, o mais comum é considerar a seguinte equação, onde α é conhecido como *learning rate*:

$$\Delta\mathsf{W} = \alpha \left(\frac{E}{\mathsf{I}} \right). \tag{7}$$

Introdução (cont.)

Quando foram propostos, os **perceptrões despertaram um interesse significativo na comunidade científica**. Isto porque:

- Eram muito **semelhantes aos neurónios reais**, que só disparam quando a quantidade de excitação é alta;
- Existia um **algoritmo de aprendizagem simples** para ajustar os pesos.

Contudo, os perceptrões só podem ser usados em **problemas linearmente separáveis**, uma vez que a **soma ponderada é uma função linear**.

Introdução (cont.)

Para exemplificar este problema, vejamos, agora, a porta lógica XOR:

i_1	i_2	*Output*
1	1	0
1	0	1
0	1	1
0	0	0

Por esta razão, **surgem as RNAs** como **modelos computacionais que resultam da conexão de neurónios simples**.

Redes Neuronais Artificiais

Redes Neuronais Artificiais

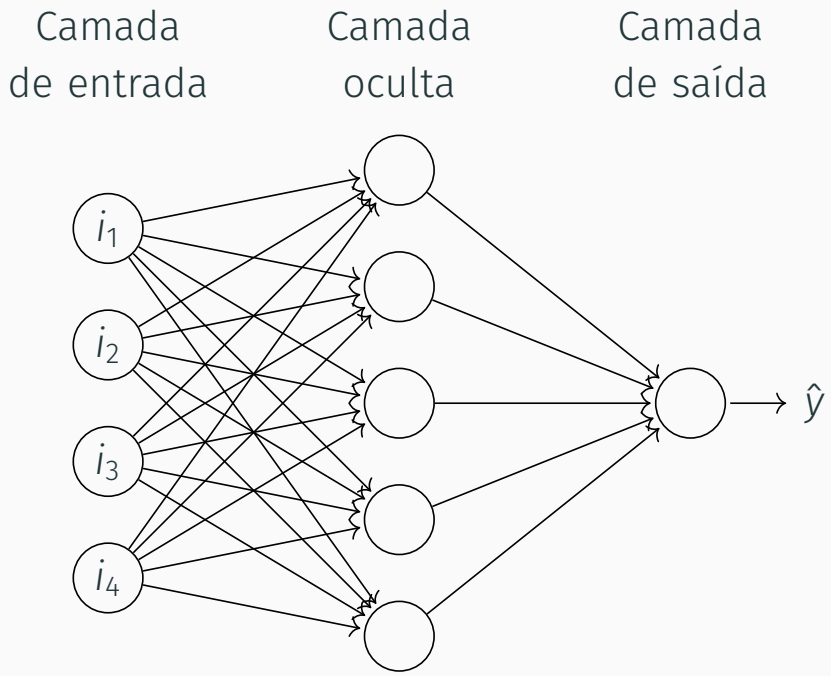


Figura 4: Arquitectura de uma rede neuronal artificial.

Numa RNA, os neurónios **não têm um limiar rígido, mas gradual**.

Isso significa que **pequenas mudanças nas entradas ou nos pesos de conexão levam a pequenas mudanças na saída**.

Redes Neuronais Artificiais (cont.)

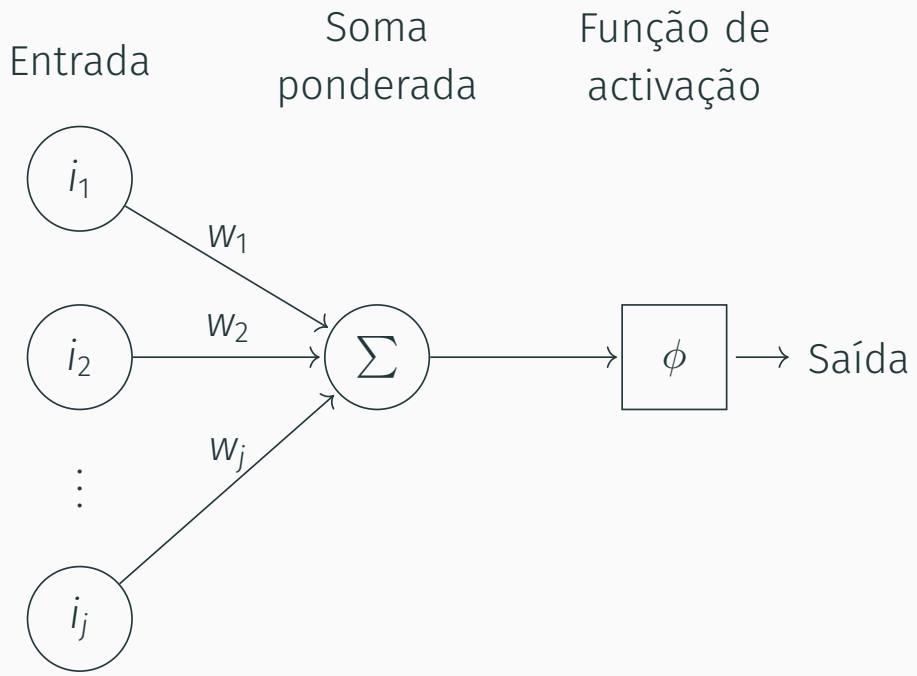


Figura 5: Arquitectura de um neurónio de uma RNA.

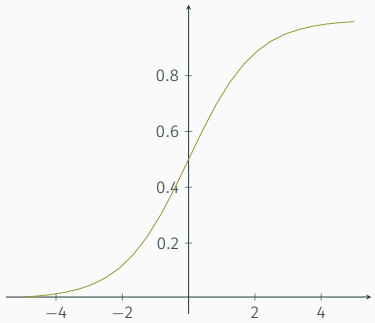
Os neurónios numa **RNA têm duas funções**:

- **Função de transferência** ($\mathsf{T} = \mathsf{W} \cdot \mathsf{I}$);
- **Função de activação** (ϕ).

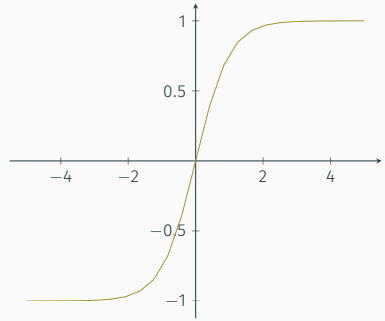
Redes Neuronais Artificiais (cont.)

Assim, a saída de um neurónio é calculada da seguinte forma:

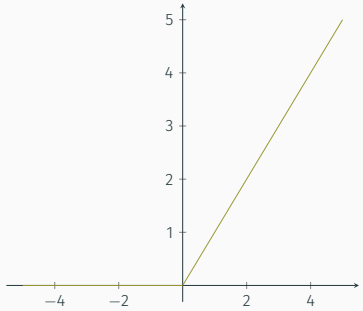
$$O_j = \phi(T_j). \tag{8}$$



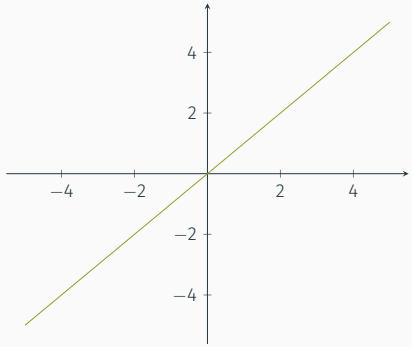
(a) Sigmoide.



(b) Tangente hiperbólica.



(c) Unidade Linear Retificada (ReLU).



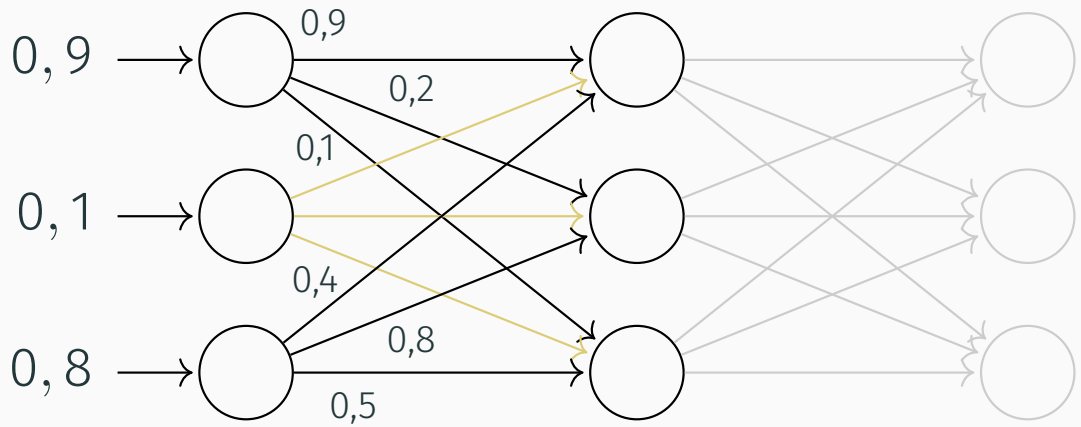
(d) Identidade.

Figura 6: Exemplos de funções de activação.

Forward Propagation

Forward Propagation

Agora que já sabemos a função de cada neurónio na rede, vamos estudar o conceito de *forward propagation*.



$$\mathbf{I} = \begin{bmatrix} 0,9 \\ 0,1 \\ 0,8 \end{bmatrix} \qquad \mathbf{W}_{\text{in\_oc}} = \begin{bmatrix} 0,9 & 0,3 & 0,4 \\ 0,2 & 0,8 & 0,2 \\ 0,1 & 0,5 & 0,6 \end{bmatrix}$$

Forward Propagation (cont.)

Agora, procedemos com o produto interno entre as matrizes **W** e **I**. Ou seja,

$$\mathbf{T}_{oc} = \mathbf{W}_{in\_oc} \cdot \mathbf{I} = \begin{bmatrix} 0,9 & 0,3 & 0,4 \\ 0,2 & 0,8 & 0,2 \\ 0,1 & 0,5 & 0,6 \end{bmatrix} \cdot \begin{bmatrix} 0,9 \\ 0,1 \\ 0,8 \end{bmatrix} = \begin{bmatrix} 1,16 \\ 0,42 \\ 0,62 \end{bmatrix},$$

e, desta forma, obtém-se o valor da **função de transferência**.

```
1 import numpy as np
2
3 I = np.array([ [0.9], [0.1], [0.8] ])
4 W_in_oc = np.array([ [0.9, 0.3, 0.4],
5                      [0.2, 0.8, 0.2],
6                      [0.1, 0.5, 0.6] ])
7
8 T_oc = np.dot(W_in_oc, I)
```

Forward Propagation (cont.)

Contudo, não nos podemos **esquecer da função de activação**.

Para este exemplo, **vamos considerar a função sigmoide**, dada por:

$$\phi(z) = \frac{1}{1 + e^{-z}} \tag{9}$$

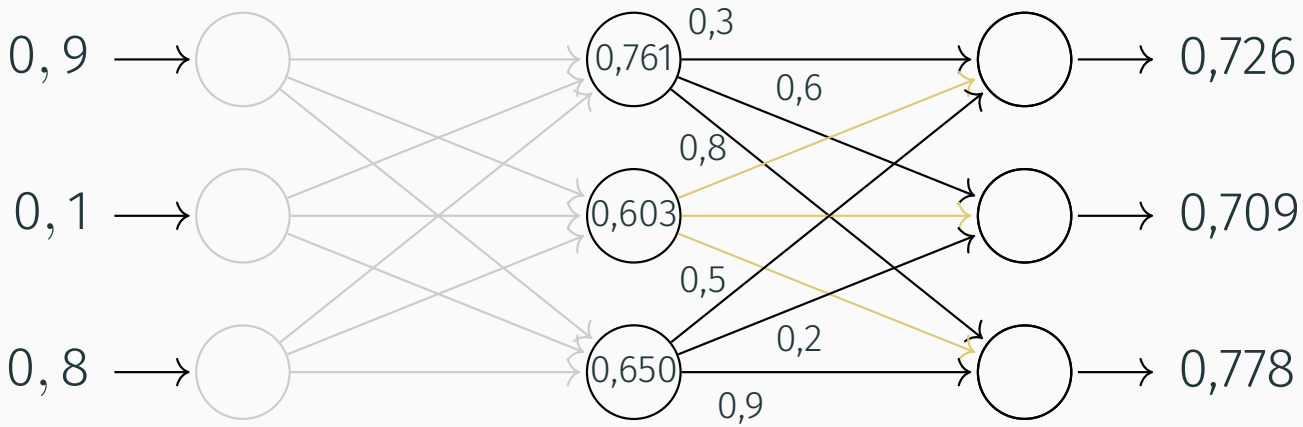
Finalmente, obtêm-se os valores de entrada da próxima camada:

$$\mathsf{O}_{oc} = \phi(\mathsf{T}_{oc}) = \frac{1}{1 + exp\left(-\begin{bmatrix} 1,16 \\ 0,42 \\ 0,62 \end{bmatrix}\right)} = \begin{bmatrix} 0,761 \\ 0,603 \\ 0,650 \end{bmatrix}. \tag{10}$$

```
1 O_oc = 1/(1 + np.exp(-T_oc))
```

Forward Propagation (cont.)

Este procedimento repete-se para a camada seguinte:



$$\mathsf{T}_{\text{ou}} = \begin{bmatrix} 0,3 & 0,7 & 0,5 \\ 0,6 & 0,5 & 0,2 \\ 0,8 & 0,1 & 0,9 \end{bmatrix} \cdot \begin{bmatrix} 0,761 \\ 0,603 \\ 0,650 \end{bmatrix} = \begin{bmatrix} 0,978 \\ 0,889 \\ 1,255 \end{bmatrix}$$

$$\mathsf{O}_{\text{ou}} = \phi(\mathsf{T}_{\text{ou}}) = \begin{bmatrix} 0,726 \\ 0,709 \\ 0,778 \end{bmatrix}.$$

Backpropagation

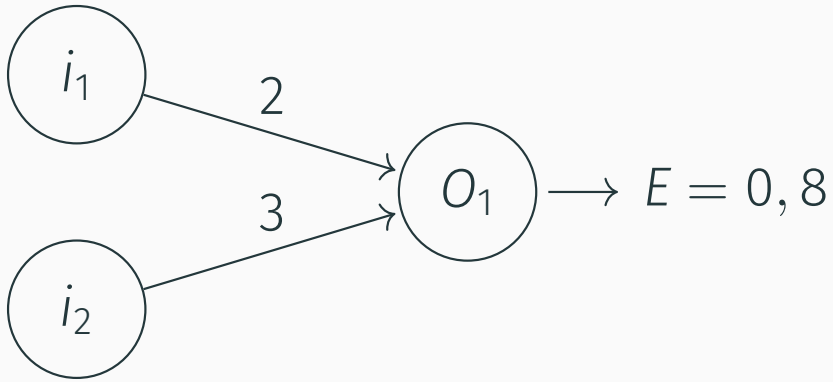
Backpropagation

O **algoritmo de *backpropagation***, ou em português *retropropagação*, **consiste em dois aspectos** fundamentais:

- **Cálculo do erro de saída e retropropagação** até aos neurónios de entrada;
- **Actualização dos pesos** por meio de um algoritmo de optimização.

Esta abordagem *treina* uma RNA para problemas simples.

Como se procede com a retropropagação do erro de saída?

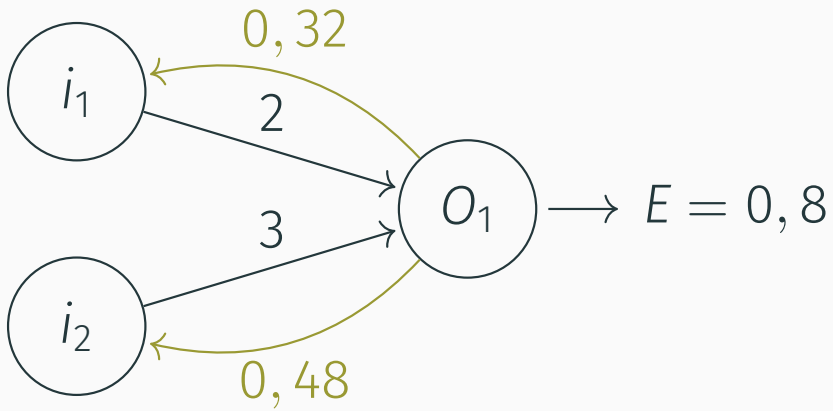


A ideia é **distribuir proporcionalmente o erro** de acordo com a contribuição de cada peso.

Neste caso,

$$\begin{aligned} E_{i_1} &= \frac{w_1}{w_1 + w_2} \times E = \frac{2}{2+3} \times 0,8 = 0,32, \\ E_{i_2} &= \frac{w_2}{w_1 + w_2} \times E = \frac{3}{2+3} \times 0,8 = 0,48. \end{aligned} \tag{11}$$

Como se procede com a retropropagação do erro de saída?



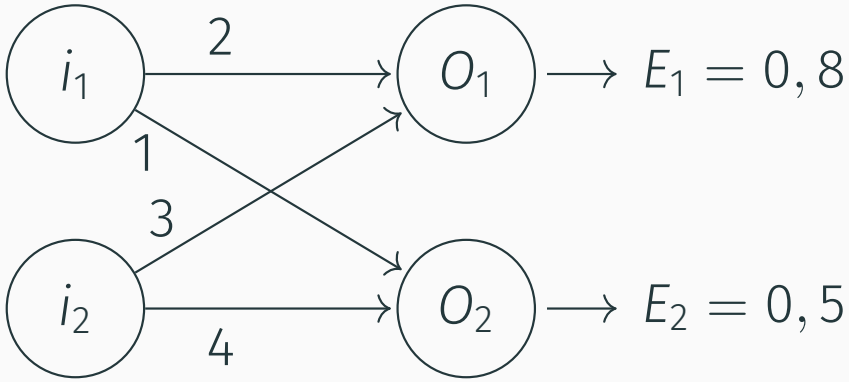
A ideia é **distribuir proporcionalmente o erro** de acordo com a contribuição de cada peso.

Neste caso,

$$\begin{aligned} E_{i_1} &= \frac{w_1}{w_1 + w_2} \times E = \frac{2}{2+3} \times 0,8 = 0,32, \\ E_{i_2} &= \frac{w_2}{w_1 + w_2} \times E = \frac{3}{2+3} \times 0,8 = 0,48. \end{aligned} \tag{11}$$

Backpropagation (cont.)

Vejamos, agora, como fazemos quando temos mais do que um neurónio de saída:

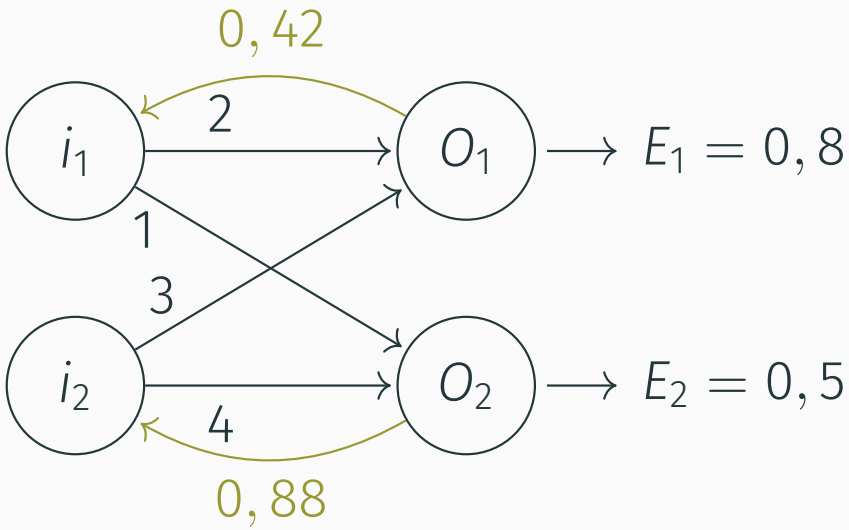


$$E_{i_1} = \frac{w_{11}}{w_{11}+w_{21}} \times E_1 + \frac{w_{12}}{w_{12}+w_{22}} \times E_2 = \frac{2}{2+3} \times 0,8 + \frac{1}{1+4} \times 0,5 = 0,42$$
$$E_{i_2} = \frac{w_{21}}{w_{11}+w_{21}} \times E_1 + \frac{w_{22}}{w_{12}+w_{22}} \times E_2 = \frac{3}{2+3} \times 0,8 + \frac{4}{1+4} \times 0,5 = 0,88 \quad (12)$$

Esta abordagem repete-se até chegarmos aos neurónios de entrada.

Backpropagation (cont.)

Vejamos, agora, como fazemos quando temos mais do que um neurónio de saída:



$$E_{i_1} = \frac{w_{11}}{w_{11}+w_{21}} \times E_1 + \frac{w_{12}}{w_{12}+w_{22}} \times E_2 = \frac{2}{2+3} \times 0,8 + \frac{1}{1+4} \times 0,5 = 0,42$$

$$E_{i_2} = \frac{w_{21}}{w_{11}+w_{21}} \times E_1 + \frac{w_{22}}{w_{12}+w_{22}} \times E_2 = \frac{3}{2+3} \times 0,8 + \frac{4}{1+4} \times 0,5 = 0,88 \quad (12)$$

Esta abordagem repete-se até chegarmos aos neurónios de entrada.

Como pode uma rede aprender com os erros?

Introduzimos, agora, o conceito de gradiente de uma função.

O gradiente de uma função $f(x, y, \cdots)$ é dado por:

$$\nabla f(x, y, \cdots) = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \\ \vdots \end{bmatrix}. \tag{13}$$

Em termos simples, o gradiente indica-nos em que **direcção devemos *andar* para aumentar** f.

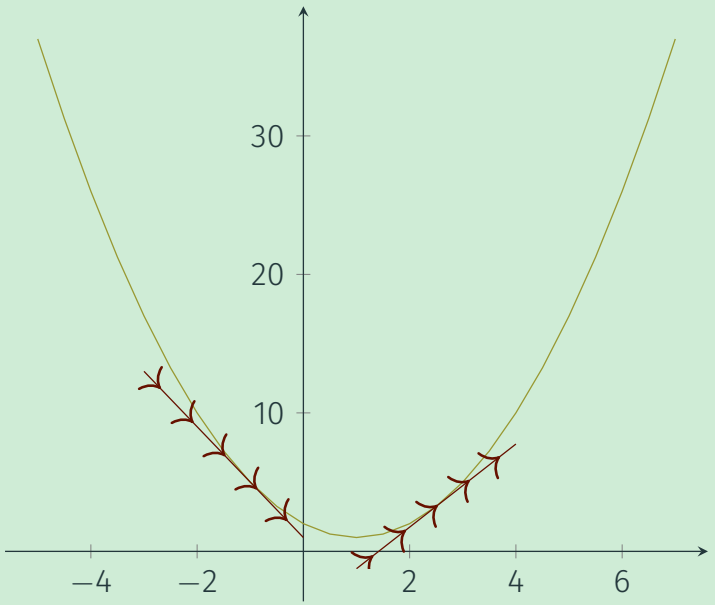
Exemplo (Cálculo do gradiente e interpretação gráfica)

Vamos considerar

$$f(x) = (x-1)^2 + 1.$$

O gradiente dessa função é, assim, dado por:

$$\nabla f(x) = \frac{\partial f}{\partial x} = 2(x-1).$$



Backpropagation (cont.)

O gradiente é usado num algoritmo de aprendizagem chamado método da descida mais rápida.

Neste algoritmo, o gradiente é usado para determinar a **direcção da descida mais rápida da função**.

É necessário, contudo, ter em consideração o **tamanho do passo a adoptar** (o tamanho do passo é, depois, multiplicado pelo *learning rate*).

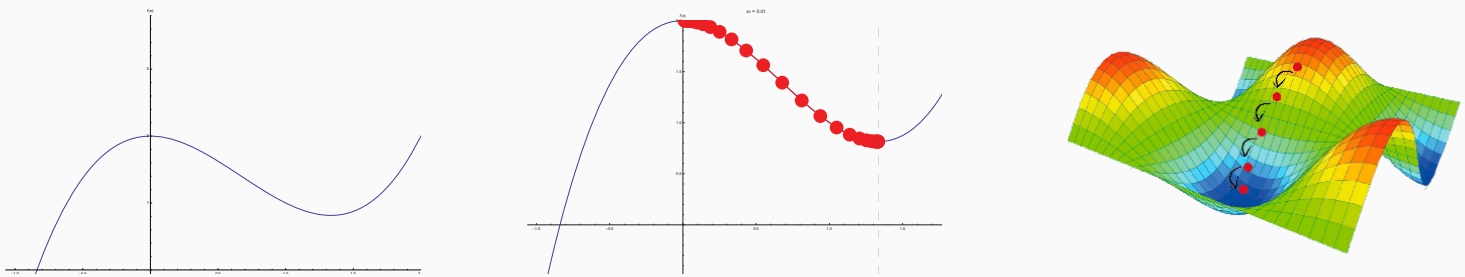


Figura 7: Função a minimizar (esquerda) e passos do método da descida mais rápida em 2-d e em 3-d (direita).

Backpropagation (cont.)

O **gradiente do erro** para um determinado peso é dado por:

$$\frac{\partial E}{\partial w_{jk}} = -E_k \times O_k \times (1 - O_k) \times O_j, \tag{14}$$

onde $E_k = (y_k - \hat{y}_k)^2$, $O_k = \text{sigmoide}\,(\mathbf{w} \cdot \mathbf{i})$ e O_j é o valor de saída do neurónio anterior.

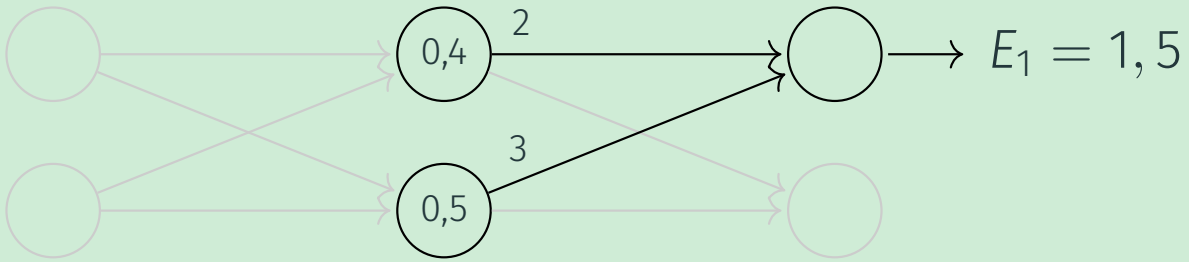
O **peso é, assim, actualizado** da seguinte forma:

$$w'_{jk} = w_{jk} - \alpha \times \frac{\partial E}{\partial w_{jk}}, \tag{15}$$

onde α é o *learning rate*, tal como vimos anteriormente.

Esta técnica é considerada um técnica de *supervised learning*.

Exemplo (Actualizar um peso de uma RNA)



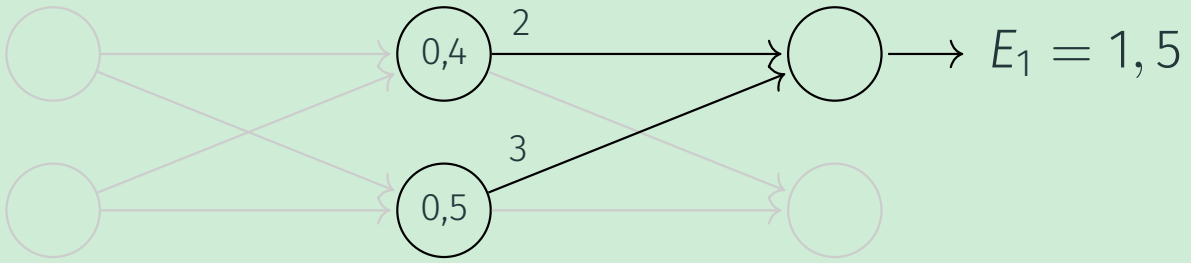
Vamos considerar $E_1 = 1,5$ e actualizar o peso w_{11}.

O somatório vem:

$$\sum_{j=1}^{2} w_{j1} i_1 = w_{11} \times i_1 + w_{21} \times i_1 = (2 \times 0,4) + (3 \times 0.5) = 2,3.$$

Aplicando a função sigmoide, $\text{sigmoide}(2,3) = 0,909 = O_1$.

Exemplo (Actualizar um peso de uma RNA – continuação)



Desta forma,

$$\frac{\partial E}{\partial w_{11}} = -E_1 \times O_1 \times (1 - O_1) \times O_{1\text{ (anterior)}}$$
$$= -1,5 \times 0,909 \times (1 - 0,909) \times 0,4 = -0,049.$$

Juntando tudo na equação final, e considerando um *learning rate* de 0,1, temos

$$w'_{11} = w_{11} - \alpha \times \frac{\partial E}{\partial w_{11}} = 2 - 0,1 \times (-0,049) = 2,005.$$

Backpropagation (cont.)

É possível **obter os pesos quase-óptimos** de uma RNA **sem recorrer ao cálculo de derivadas**.

Estes métodos constituem **técnicas tradicionais de optimização**, tais como:

- Enxame de partículas;
- Algoritmos genéticos;
- Arrefecimento simulado.

A vantagem destes algoritmos é que podemos **optimizar a arquitectura de uma RNA e os seus pesos**.

Backpropagation (cont.)

Vamos rever:

1. Os neurónios de entrada recebem os valores de entrada e os transmitem para a camada oculta seguinte.
2. Com base numa função matemática, cada neurónio oculto produz um valor e passa esse valor para a próxima camada oculta (se existir) ou para a camada de saída.
3. A camada de saída recebe as saídas da última camada oculta e calcula o valor previsto pelo modelo.
4. Este valor previsto é, então, comparado com o valor esperado, e o erro é usado para actualizar os pesos das conexões sinápticas, com base num algoritmo de optimização.

Algumas Regras de Polegar

Algumas Regras de Polegar

A escolha do número de neurónios ocultos e o número de camadas não é sabido.

Existem várias **regras de polegar para escolher o número de neurónios ocultos**, tais como:

- $\frac{2}{3}$ do número de neurónios de entrada;
- Entre o número de camadas de entrada e de saída;
- Menos do que o dobro do número de neurónios na camada de entrada;
- $\frac{1}{10}$ do número de exemplos usados para treinar a rede.

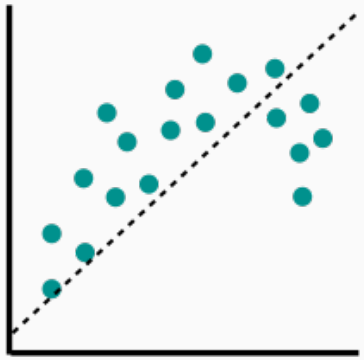
Overfitting vs. *Underfitting*

Overfitting vs. *Underfitting*

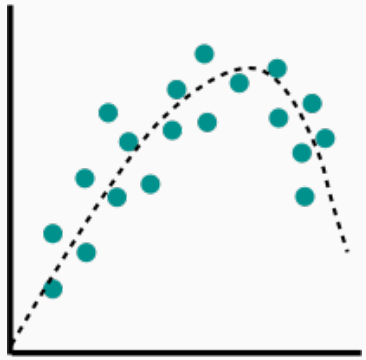
O **objectivo** de qualquer modelo de *machine learning* é **ter uma boa capacidade de generalização**.

Assim, surge o conceito de *overfitting* e de *underfitting*:

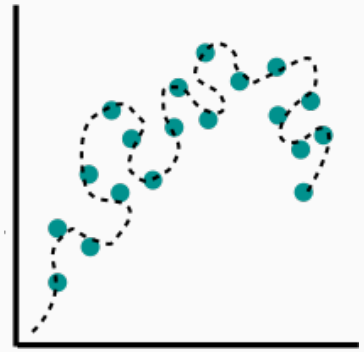
- ***Overfitting*** – quando o modelo ajusta-se muito bem ao conjunto de dados de treino.
- ***Underfitting*** – quando o modelo não se ajusta ao conjunto de dados de treino, nem de teste (modelo inadequado).



(a) *Underfitting*.



(b) Ponto óptimo.



(c) *Overfitting*.

Overfitting vs. *Underfitting* (cont.)

O **ponto ótimo** é ter um **equilíbrio entre *overfitting* e *underfitting***.

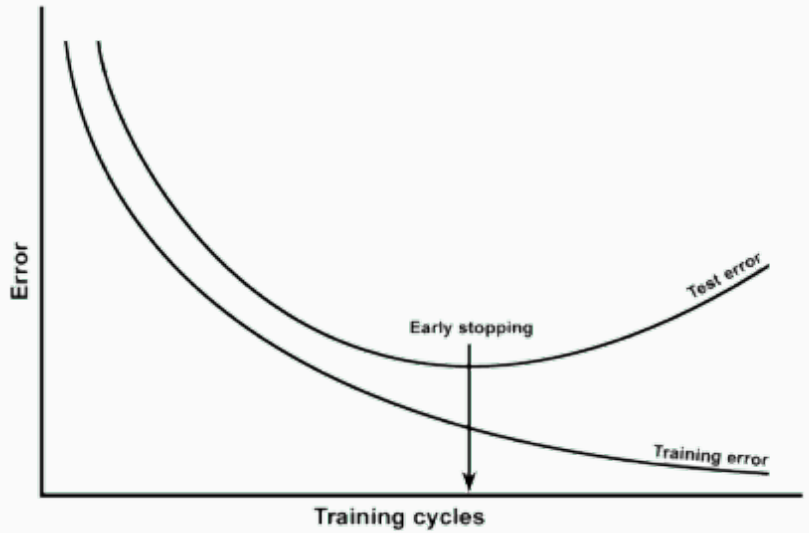
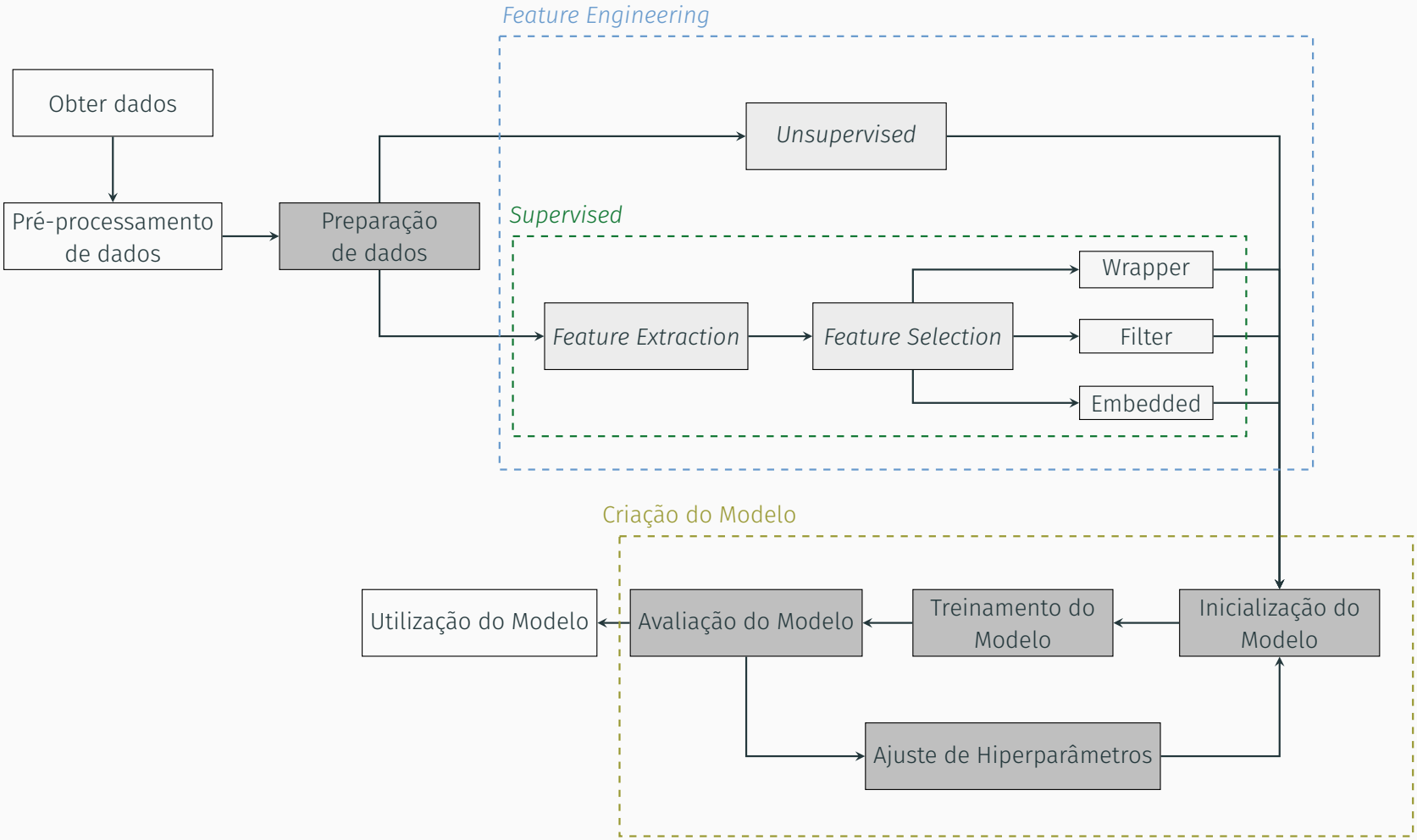


Figura 9: Ponto óptimo para paragem (antecipada) do treino.

Pipeline Geral

Pipeline Geral



Questões?

Bibliografia

I. Goodfellow, Y. Bengio, and A. Courville.

Deep Learning.

MIT Press, Cambridge, MA, EUA, 2016.

ISBN:978-026-203-561-3.

M. A. Nielsen.

Neural Networks and Deep Learning.

Determination Press, San Francisco, CA, USA, 2015.

ISBN:978-026-203-561-3.

A. Oliveira.

Mentes Digitais: A Ciência Redefinindo a Humanidade.

IST Press, Lisboa, Portugal, 2016.

ISBN:978-989-848-160-3.

T. Rashid.

Make Your Own Neural Network.

CreateSpace Independent Publishing Platform, Scotts Valley, CA, USA, 2016.

ISBN:978-153-082-660-5.

Introdução às Redes Neuronais

Um Percurso da Teoria à Prática

Diogo Freitas
diogo.freitas@iti.larsys.pt
@DiogoNTFreitas

18 de dezembro de 2020

GDG Madeira

ORDEM DOS ENGENHEIROS