

Introdução às Redes Neuronais

Um Percurso da Teoria à Prática

Diogo Nuno Freitas
diogo.freitas@staff.uma.pt

Inteligência Artificial

11 de janeiro de 2022



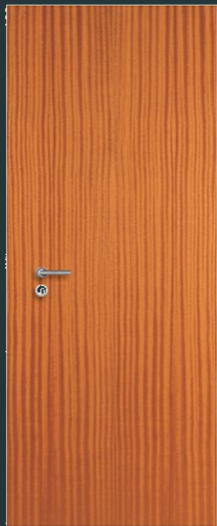
Motivação

O que é uma *porta*?

Não se refira ao termo informático de *porta*.

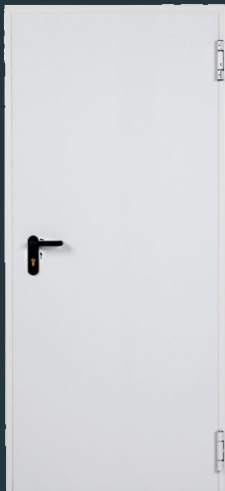
O que é uma *porta*?

Não se refira ao termo informático de *porta*.



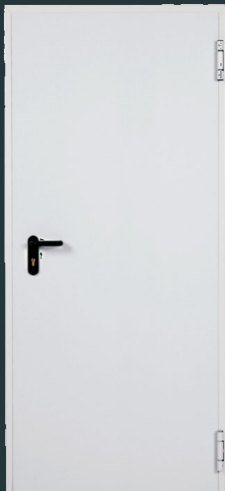
O que é uma *porta*?

Não se refira ao termo informático de *porta*.



O que é uma *porta*?

Não se refira ao termo informático de *porta*.



O que é uma porta? (cont.)

Vamos **desenvolver** um **algoritmo** para detectar o objecto *porta*.

```
1 def is_door(object):  
2     if object.has_handle():  
3         if object.has_keyhole():  
4             if object.shape == 'square':  
5                 if object.height >= 2 and object.width >= 0.8:  
6                     return True  
7     return False
```

Este algoritmo segue o paradigma da **programação tradicional**:



Vejamos outros exemplos de portas...

Vejamos outros exemplos de portas...



Vejamos outros exemplos de portas...



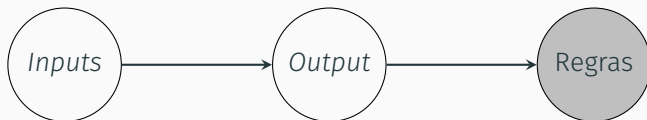
Vejamos outros exemplos de portas...



O que é uma porta? (cont.)

Nesses exemplos, o nosso algoritmo **não vai retornar a resposta correcta.**

Uma solução (a mais recomendada) passa por utilizar **aprendizagem automática**:



Permite um sistema **aprender com a experiência, i.e., o sistema escreve automaticamente as regras.**

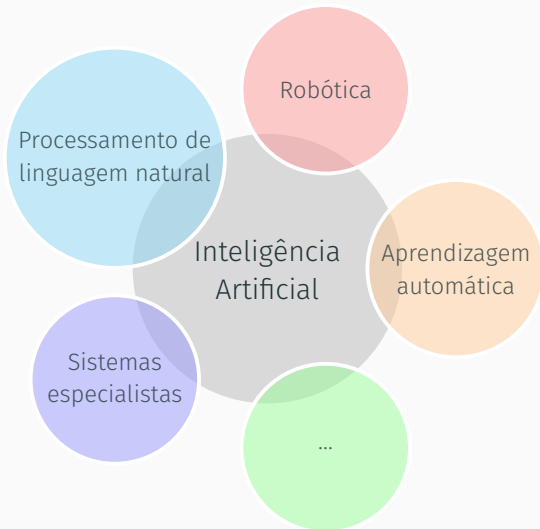
As **redes neurais artificiais** podem ser usadas para **atingir a aprendizagem automática** de um sistema.

Agenda

1. Introdução
2. Redes Neurais Artificiais
3. *Forward Propagation*
4. *Backpropagation*
5. Classificação vs. Regressão
6. *Overfitting* vs. *Underfitting*
7. Algumas Regras de Polegar
8. *Unsupervised Learning*
9. *Pipeline* Geral

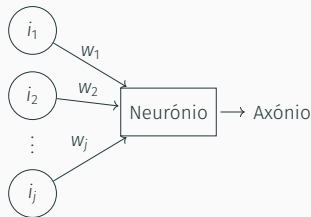
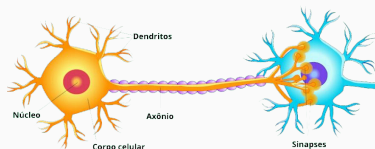
Introdução

Onde estamos no panorama da Inteligência Artificial?



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

Elas *imitam* a forma de como os nossos **neurônios** se interconectam, bem como **forma de como nós aprendemos**—**inferência indutiva**.



Dendrites com sinapses

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

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

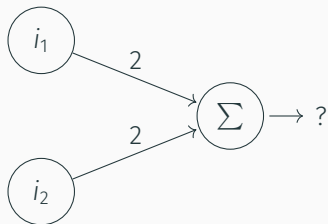
De acordo com estes autores, os neurónios faziam **uma soma ponderada (w) das suas entradas (i)** para implementar qualquer função lógica.

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

Os perceptrões faziam uma **soma ponderada** das suas entradas, mas o ***output* (\hat{y}) era binário**, de tal forma que:

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

Exemplo (Calcular o *output* de um perceptrão:)



Primeiro caso: $i_1 = i_2 = 4$

$$\sum_{j=1}^2 w_j i_j = w_1 i_1 + w_2 i_2$$
$$= 2 \times 4 + 2 \times 4 = 16.$$

Se considerarmos um limiar de 2, 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

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

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

O termo b é conhecido como *bias*.

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

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

onde \cdot denota o produto interno.

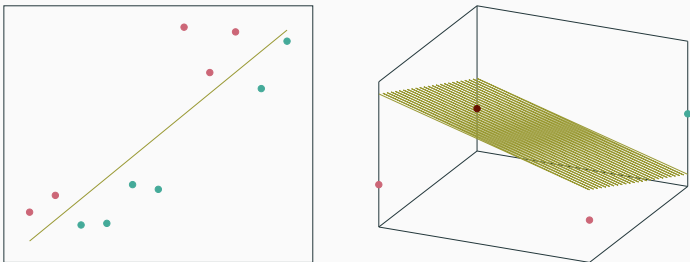


Figura 2: Recta (esquerda) e plano (direita) de regressão que se podem obter de um perceptrão.

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**.

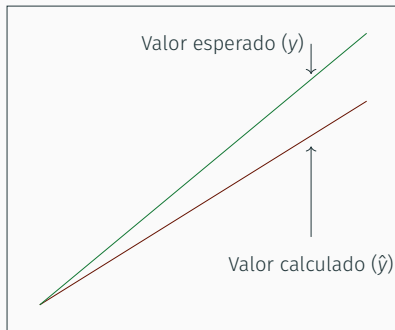


Figura 3: Visualização gráfica do cálculo do erro (E).

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.

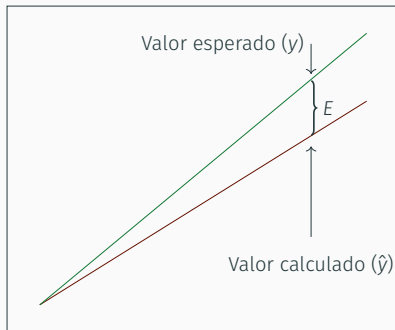


Figura 3: Visualização gráfica do cálculo do erro (E).

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.

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

$$y = (\mathbf{W} + \Delta\mathbf{W}) \cdot \mathbf{I}, \quad (4)$$

onde Δ denota um factor de mudança.

Assim,

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

e, desta forma,

$$\Delta\mathbf{W} = \frac{E}{\mathbf{I}}. \quad (6)$$

Contudo, o mais comum é considerar a seguinte equação, onde α é conhecido como *learning rate* (ou **tamanho do passo**):

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

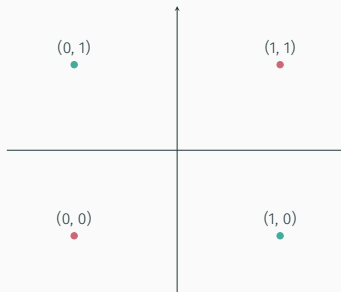
Quando foram propostos, os **perceptrões** despertaram um interesse **significativo na comunidade científica**. Isto porque:

- 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**.

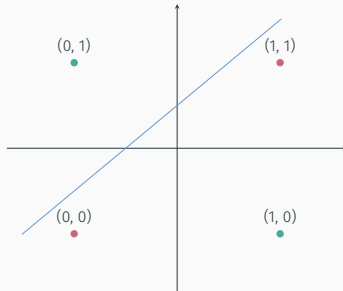
Para exemplificar este problema, vejamos, agora, a porta lógica XOR:

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



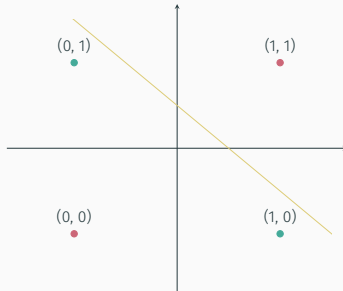
Para exemplificar este problema, vejamos, agora, a porta lógica XOR:

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



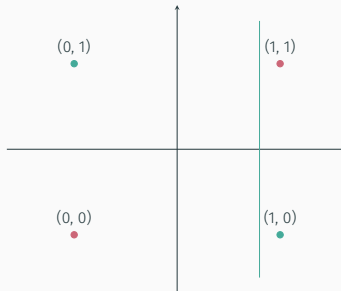
Para exemplificar este problema, vejamos, agora, a porta lógica XOR:

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



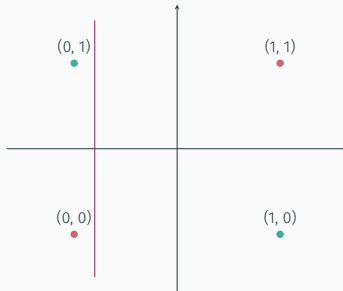
Para exemplificar este problema, vejamos, agora, a porta lógica XOR:

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



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 Neurais Artificiais

Redes Neurais Artificiais

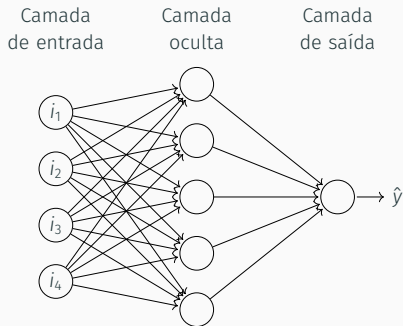


Figura 4: Arquitetura de uma rede neuronal artificial.

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.**

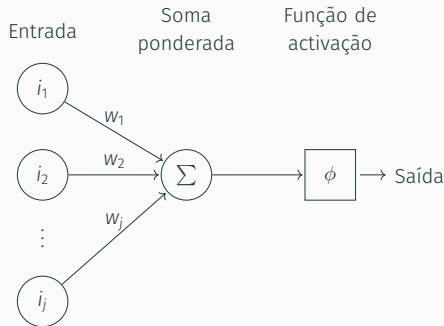


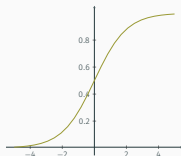
Figura 5: Arquitectura de um neurónio de uma RNA.

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

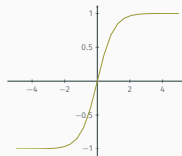
- Função de transferência ($T = W \cdot I$);
- Função de activação (ϕ).

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

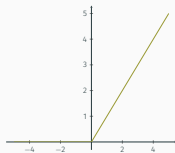
$$O_j = \phi(T_j). \quad (8)$$



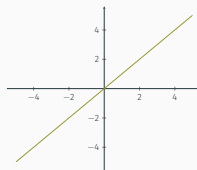
(a) Sigmóide.



(b) Tangente hiperbólica.



(c) Unidade Linear Retificada (ReLU).



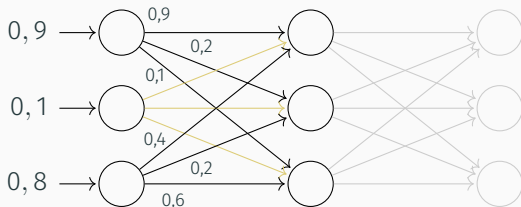
(d) Identidade.

Figura 6: Exemplos de funções de activação.

Forward Propagation

Forward Propagation

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



$$I = \begin{bmatrix} 0,9 \\ 0,1 \\ 0,8 \end{bmatrix}.$$

$$W_{in_oc} = \begin{bmatrix} 0,9 & 0,3 & 0,4 \\ 0,2 & 0,8 & 0,2 \\ 0,1 & 0,5 & 0,6 \end{bmatrix}.$$

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)
```

Contudo, não nos podemos esquecer da função de activação.

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

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

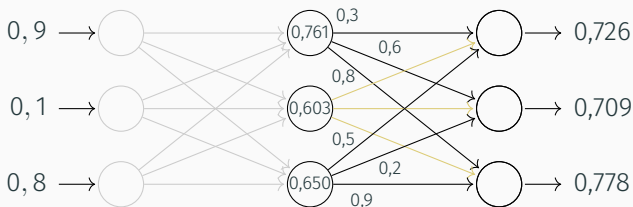
Finalmente, obtêm-se os valores de entrada da próxima camada:

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

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

Forward Propagation (cont.)

Este procedimento repete-se para a camada seguinte:



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

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

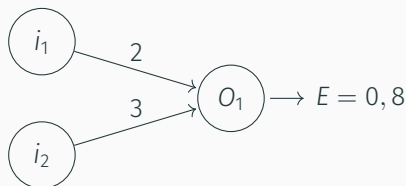
Backpropagation

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?

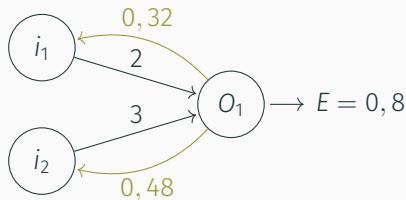


A ideia é **distribuir proporcionalmente o erro** de acordo com a contribuição de cada peso.

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?

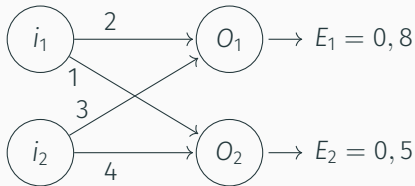


A ideia é **distribuir proporcionalmente o erro** de acordo com a contribuição de cada peso.

Neste caso,

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

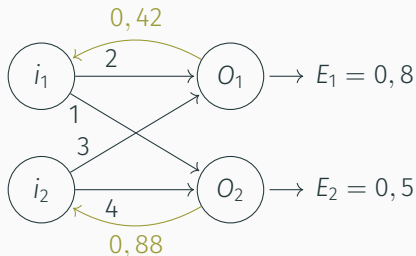
Vejamos, agora, como fazemos quando temos mais do que um neurônio de saída:



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

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

Vejamos, agora, como fazemos quando temos mais do que um neurônio 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 $f(x, y, \dots)$ é dado por:

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

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

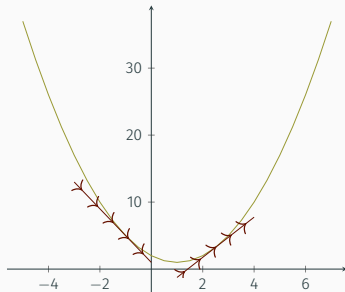
Exemplo (Cálculo do gradiente e interpretação gráfica:)

Vamos considerar

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

O gradiente dessa função é, assim, dado por:

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



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** (i.e., o *learning rate*).

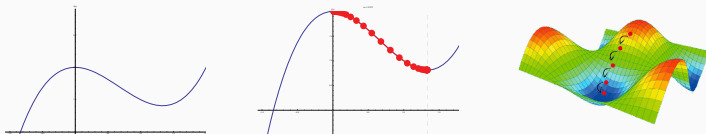


Figura 7: Função a minimizar (esquerda) e passos do método da descida mais rápida em 2-d e em 3-d (direita).

O **gradiente do erro** para um determinado peso entre a última camada oculta e a camada de saída é dado por:

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

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

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

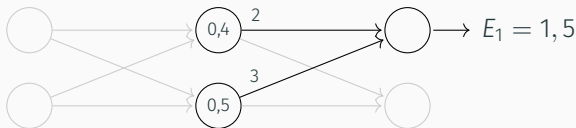
$$w'_{jk} = w_{jk} - \alpha \times \frac{\partial E}{\partial w_{jk}}, \quad (15)$$

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

O *learning rate* mede a deslocação (passo) ao longo da direcção $\frac{\partial E}{\partial w_{jk}}$.

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

Exemplo (Actualizar um peso de uma RNA:)



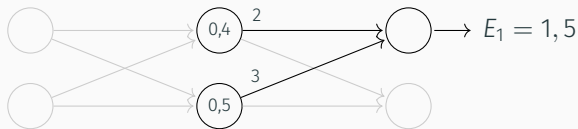
Vamos considerar $E_1 = 1,5$ e actualizar o peso w_{11} .

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 sigmóide, $\text{sigmóide}(2,3) = 0,909 = O_1$.

Exemplo (Atualizar um peso de uma RNA—continuação:)



Desta forma,

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

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

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

Para os restantes pesos, a expressão é semelhante, e é dada por

$$\frac{\partial E}{\partial w_{jk}} = -e_k \times O_k \times (1 - O_k) \times o_j,$$

onde e_k é o erro que vem da retro-propagação.

Contudo, no caso genérico, têm-se

$$\frac{\partial E}{\partial w_{jk}} = -e_k \times \frac{\partial}{\partial w_{jk}} \phi \left(\sum_j w_{jk} \cdot o_j \right) \times o_j,$$

onde ϕ é, tal como vimos, uma qualquer função de activação diferenciável.

Na prática, o método da descida mais rápida **não é utilizado directamente para o treino de uma RNA.**

Como alternativa, podemos usar o método do **gradiente descendente estocástico** (SGD).

O SGD utiliza uma **aproximação do gradiente** da função calculado a partir de um **subconjunto de dados (*batch*) seleccionado aleatoriamente (*mini-batch*).**

Uma **passagem por todos os *mini-batches*** de um conjunto de dados designa-se por ***época***.

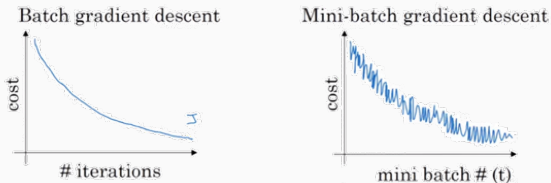


Figura 8: Diferença entre o método da descida e o gradiente descendente estocástico.

É preciso ter em atenção que:

- *mini-batches* com um **menor número de elementos** faz com que o algoritmo precise de **mais iterações para convergir**.
- *mini-batches* com um **maior número de elementos** faz com que o cálculo do gradiente seja **mais exacto**, contudo **mais lento**.

É, ainda, 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**.

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.

Classificação vs. Regressão

Classificação vs. Regressão

Classificação e regressão **são problemas semelhantes**.

Em termos gerais, a classificação tenta **rotular** uma combinação de características. Por outro lado, a regressão tenta estimar uma **quantidade**.

Exemplos:

- **Regressão:** previsão do preço de uma casa, da altura de uma pessoa, do preço de acções...
- **Classificação:** detecção de correio electrónico não solicitado, reconhecimento de algarismos, reconhecimento de espécies...

Classificação vs. Regressão (cont.)

Apesar de problemas de classificação lidarem com variáveis discretas, a saída da rede neuronal é contínua.

Esta saída é vista como uma **probabilidade de uma dada combinação de características pertencer a uma dada classe.**

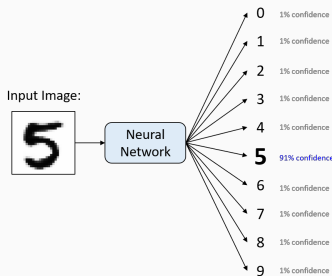


Figura 9: Exemplo de um problema de classificação.

Como podemos estimar a capacidade do nosso modelo de classificação?

		Classe Prevista		Total
		p	n	
Classe Real	p'	Verdadeiro Positivo	Falso Negativo	p'
	n'	Falso Positivo	Verdadeiro Negativo	n'
Total		P	N	

Uma das medidas mais comuns é a *accuracy*, e é dada por:

$$ACC = \frac{VP + VN}{P + N}. \quad (16)$$

Como podemos estimar a capacidade do nosso modelo de regressão?

Existem várias métricas, mas as mais comuns são o erro quadrático médio (MSE) e a raiz do erro quadrático médio (RMSE).

Estas métricas são definidas da seguinte forma:

$$\begin{aligned} \text{MSE} &= \frac{1}{n} \sum_{k=1}^n (y_k - \hat{y}_k)^2 \quad \text{e} \\ \text{RMSE} &= \sqrt{\text{MSE}} = \sqrt{\frac{1}{n} \sum_{k=1}^n (y_k - \hat{y}_k)^2}, \end{aligned} \tag{17}$$

onde n é o número de amostras.

Um vez que os modelos de **classificação** têm como valor de saída uma probabilidade, a **função de activação da camada de saída tem de estar no intervalo $[0, 1]$** .

Para esse caso, um exemplo de função de activação a usar é a **função sigmóide**.

Para o caso da **regressão**, a última camada tem de possuir a **função de activação de identidade**, de modo a não restringir a **saída da rede ao domínio da função de activação**.

Na maioria dos casos, para as **camadas ocultas**, qualquer função de **activação** poderá ser usada.

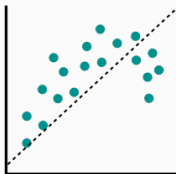
Overfitting vs. Underfitting

Overfitting vs. Underfitting

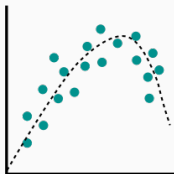
O **objectivo** de qualquer modelo de *machine learning* é **ter uma boa capacidade de generalização**.

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

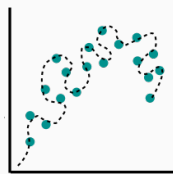
- **Overfitting**—quando o modelo ajusta-se muito bem ao conjunto de dados de treino.
- **Underfitting**—quando o modelo não se ajusta ao conjunto de dados de treino, nem de teste (modelo inadequado).



(a) Underfitting.



(b) Ponto óptimo.



(c) Overfitting.

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

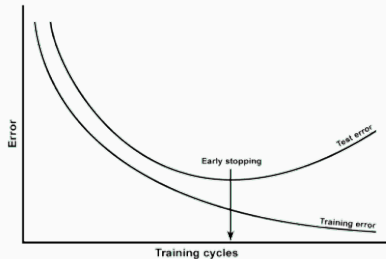


Figura 11: Ponto ótimo para paragem (antecipada) do treino.

Algumas Regras de Polegar

O número de neurónios na camada oculta e o número de camadas não é sabido.

Existem várias regras de polegar para escolher o número de neurónios na camada oculta, tais como:

- $\frac{2}{3}$ do número de neurónios de entrada;
- 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.

Unsupervised Learning

Unsupervised Learning

Anteriormente, disponibilizávamos à rede exemplos de mapeamento entre as entradas e as saídas.

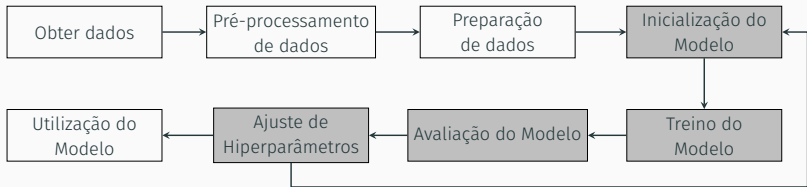
No caso do *unsupervised learning*, esse mapeamento não é fornecido à rede. A rede é, assim, obrigada a aprender a classificar os dados com base nos padrões ou nos *clusters* que encontrar.

Exemplos de tarefas com *unsupervised learning*:

- **Clustering:** quando os dados são agrupados com base na sua semelhança. Por exemplo, as redes *self-organizing map*.
- **Redução de dimensionalidade:** quando, de todas as características dos dados, apenas são mantidas algumas. Por exemplo, *autoencoder*.

Pipeline Geral

Pipeline Geral



Vamos avançar para a parte prática!

🔗 [https://github.com/Dntfreitas/
Dntfreitas.github.io](https://github.com/Dntfreitas/Dntfreitas.github.io)

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 Nuno Freitas
diogo.freitas@staff.uma.pt

Inteligência Artificial

11 de janeiro de 2022

