



Mini-Curso: Introdução à Inteligência Artificial com Redes Neurais Artificiais

Parte 1

Professor José Francisco Pessanha

06 de dezembro de 2024

15:30 – 18:00, sala RAV62, 6º andar, bloco F

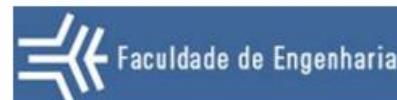
https://homeprojextransicaoenergetica.netlify.app/_site/eventos

Projeto de Extensão

TRANSIÇÃO ENERGÉTICA: vantagens e desafios técnicos das energias renováveis para o equilíbrio entre custos, segurança e mudanças climáticas



Departamento de
Estatística



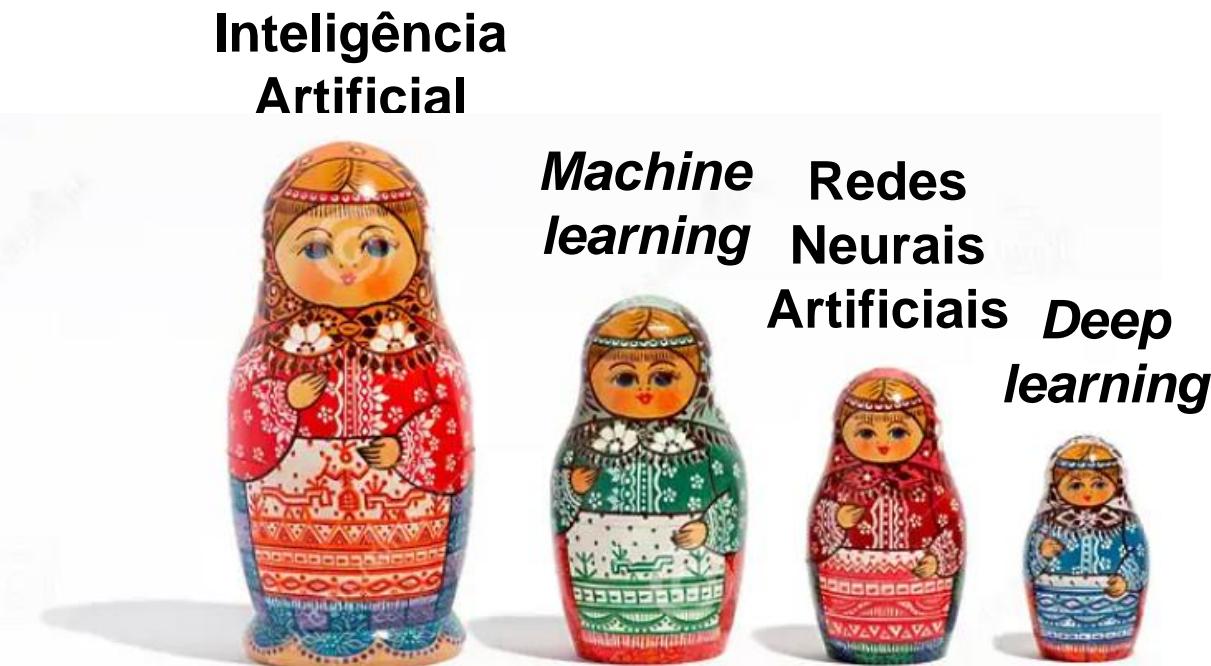
ELE
Dept. de Eng. Elétrica



Inteligência Artificial

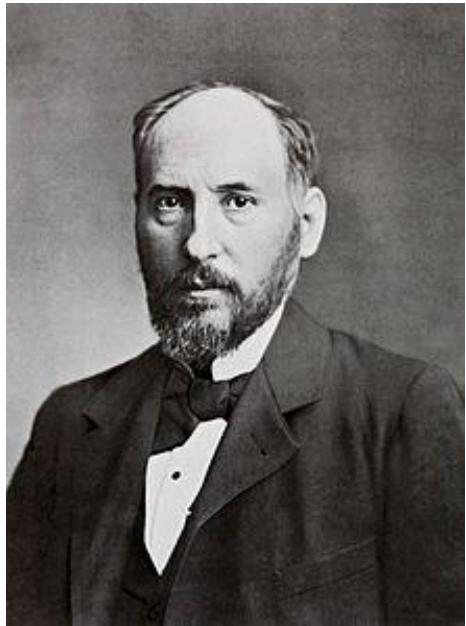
A inteligência artificial pode ser definida como o ramo da ciência da computação que se ocupa da automação do comportamento inteligente (Luger, 2013).

- **Paradigma conexionista:** visa construir um sistema que simule um **sistema inteligente capaz de aprender, assimilar, errar e aprender com seus erros** (Redes Neurais Artificiais, Deep Learning).



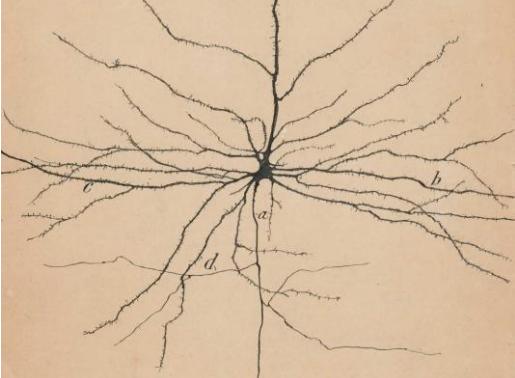
Redes Neurais Artificiais - RNA

- **Modelo de aprendizado de máquina inspirado na estrutura neural de organismos inteligentes e que adquirem conhecimento através da experiência.**
- **As redes neurais artificiais estão no centro nevrálgico do aprendizado de máquina (GÉRON, 2021)**
- **Permite construir sistemas inteligentes, i.e., com capacidade de aprender, assimilar, errar e aprender com seus erros.**
- **Diversas arquiteturas de redes**
- **Redes Neurais Supervisionadas: *redes multilayer perceptron MLP* em problemas de regressão e classificação**
- **Redes Neurais Não Supervisionadas: *self organizing map SOM* na análise de agrupamentos (clustering)**



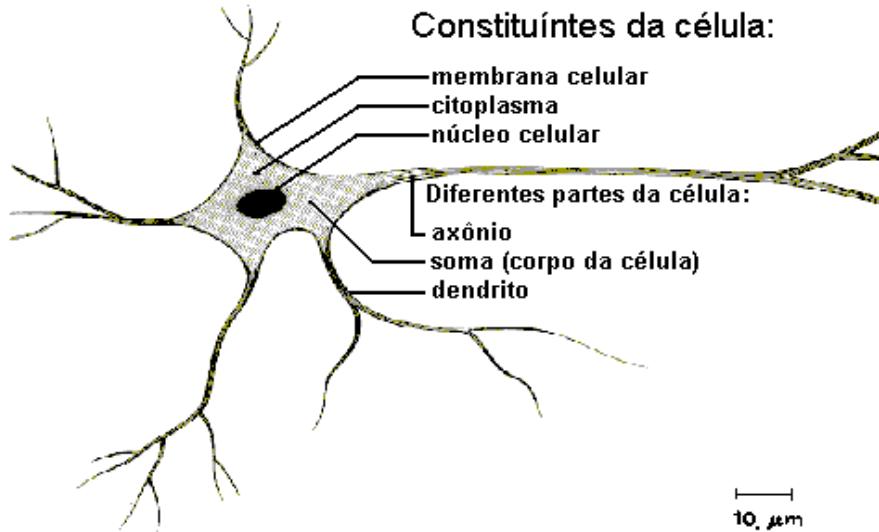
Santiago Ramón y Cajal
1852 – 1934

Nobel de Medicina 1906
Pai da Neurociência



Neurônio (final do século XIX)

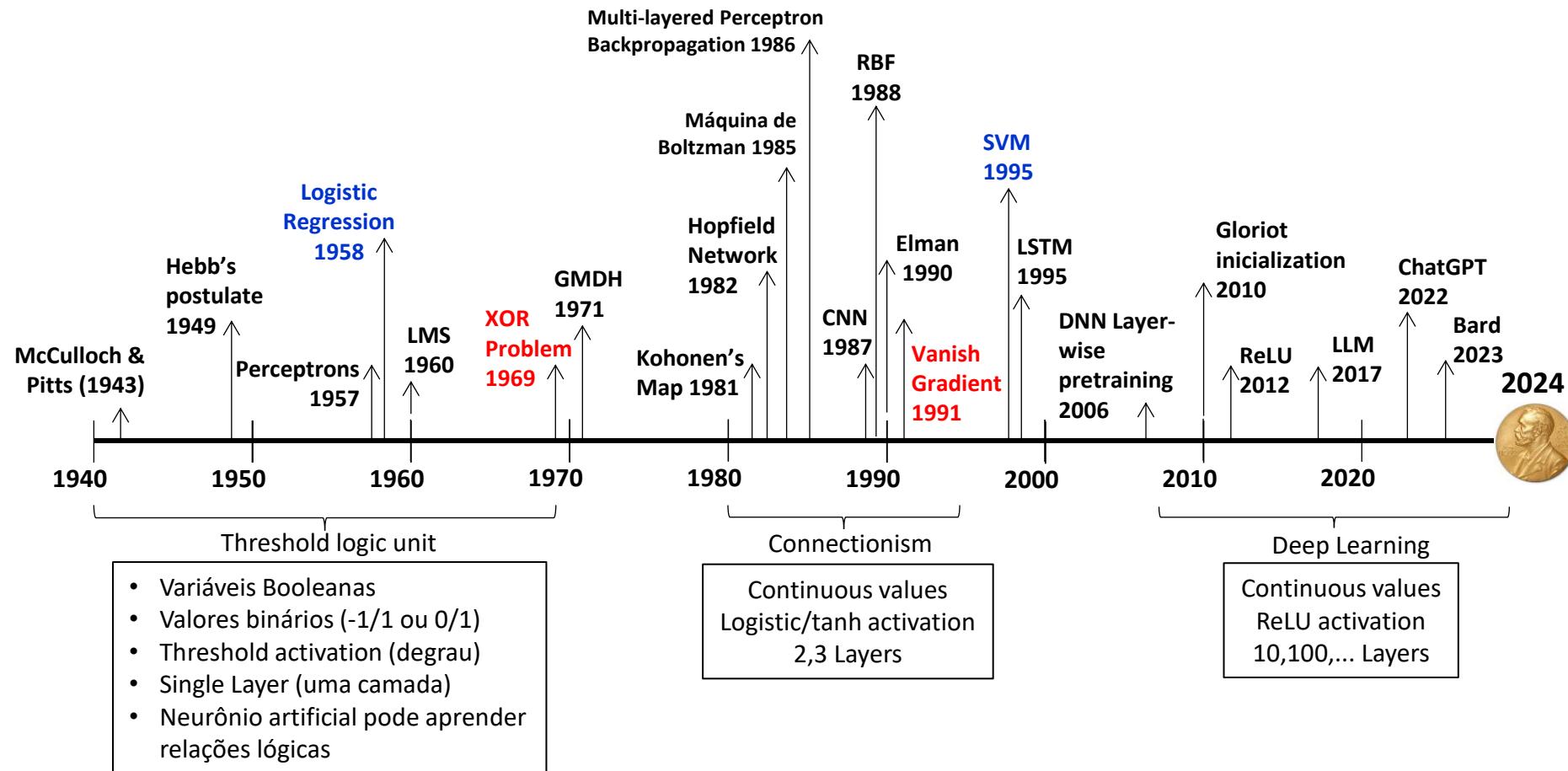
Teoria do Neurônio: O sistema nervoso é composto por células individuais, chamadas neurônios, que se comunicam entre si através de sinapses.



Work

Our bodies are controlled by the interaction between the brain and the nervous system, which extends throughout the body. Camillo Golgi's discovery in the 1870s that nerve cells could be colored using silver nitrate opened up new opportunities for their study. Santiago Ramón y Cajal began using this method in 1887 and achieved many groundbreaking results in the years that followed. This included proving that each nerve cell is an independent entity and nerve synapses transfer nerve impulses from one cell to another.

Marcos do desenvolvimento das redes neurais artificiais



BULLETIN OF
MATHEMATICAL BIOPHYSICS
VOLUME 5, 1943

Neurônio Artificial (1943)

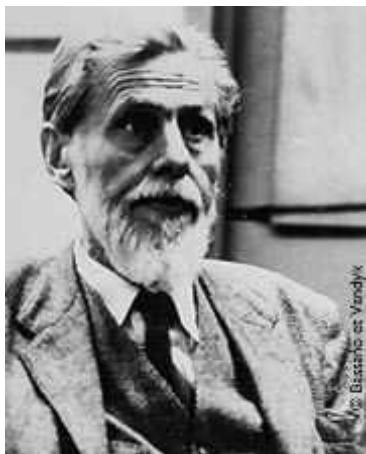
A LOGICAL CALCULUS OF THE
IDEAS IMMANENT IN NERVOUS ACTIVITY

WARREN S. MCCULLOCH AND WALTER PITTS

FROM THE UNIVERSITY OF ILLINOIS, COLLEGE OF MEDICINE,
DEPARTMENT OF PSYCHIATRY AT THE ILLINOIS NEUROPSYCHIATRIC INSTITUTE,
AND THE UNIVERSITY OF CHICAGO

Because of the “all-or-none” character of nervous activity, neural

[A logical calculus of the ideas immanent in nervous activity | Bulletin of Mathematical Biology](#)



Warren McCulloch
1898-1969



Walter Pitts
1923-1969

McCulloch & Pitts Publish the First Mathematical Model of a Neural Network

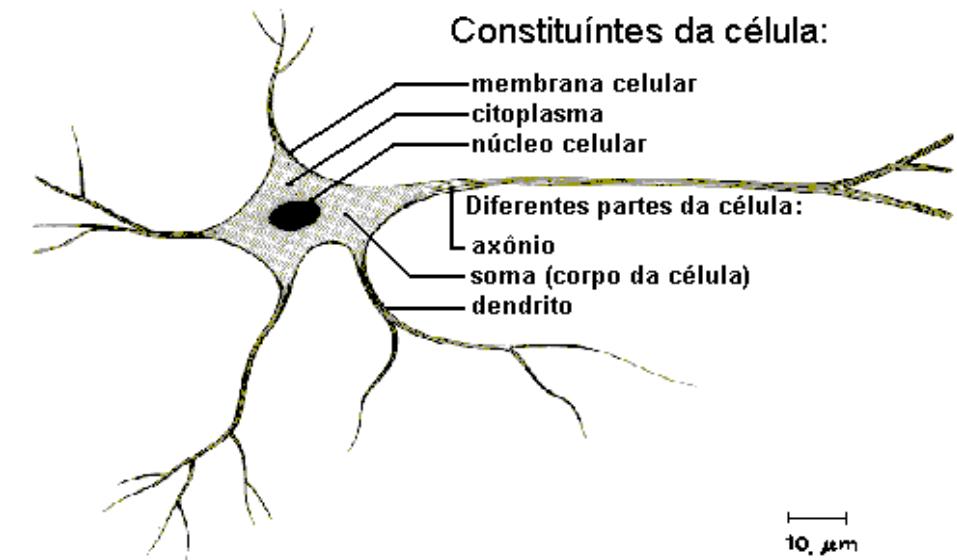
[McCulloch & Pitts Publish the First Mathematical Model of a Neural Network : History of Information](#)

Primeiro modelo computacional do neurônio

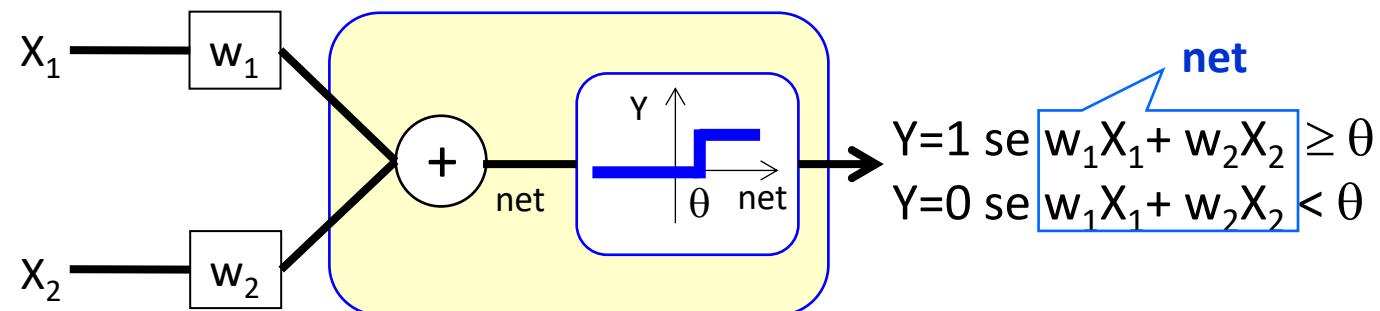
Neurônio Artificial (1943)

Neurônio biológico: Os impulsos recebidos por um neurônio são processados, e atingindo um dado limiar (*threshold* θ), o neurônio dispara, produzindo uma substância neurotransmissora que flui do corpo celular para o axônio que se conecta a um dendrito de outro neurônio.

O objetivo dos neurônios é processar sinais que transportam informações (SEJNOWSKI, 2019)



Neurônio artificial de McCulloch e Pitts



Primeiro modelo computacional do neurônio
(*Threshold Logic Unit* – TLU)

Caso com n variáveis de entrada

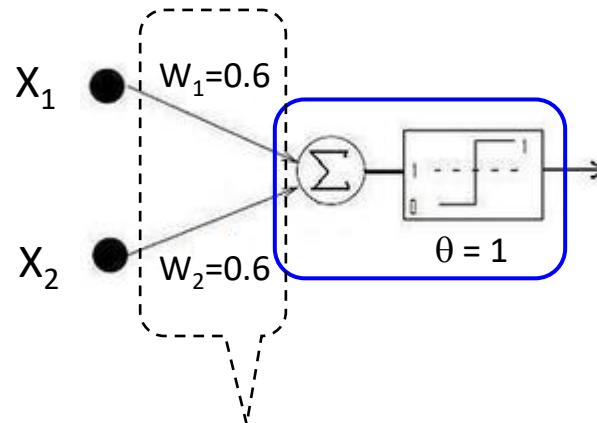
$$net = \sum_{i=1}^n w_i x_i$$

$$\begin{aligned} y = g(net) &= 1 & \text{se } g(net) \geq \theta \\ &= 0 & \text{se } g(net) < \theta \end{aligned}$$

Funções Booleanas

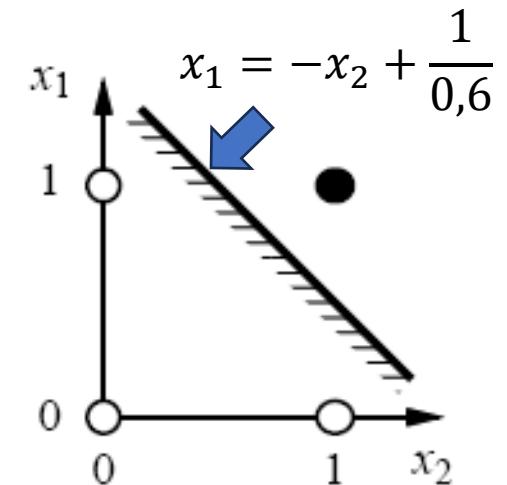
Regra E

Entrada X1	Entrada X2	Saída Y
0	0	0
0	1	0
1	0	0
1	1	1



Se $0,6X_1 + 0,6X_2 \geq 1 \rightarrow Y=1$
Se $0,6X_1 + 0,6X_2 < 1 \rightarrow Y=0$

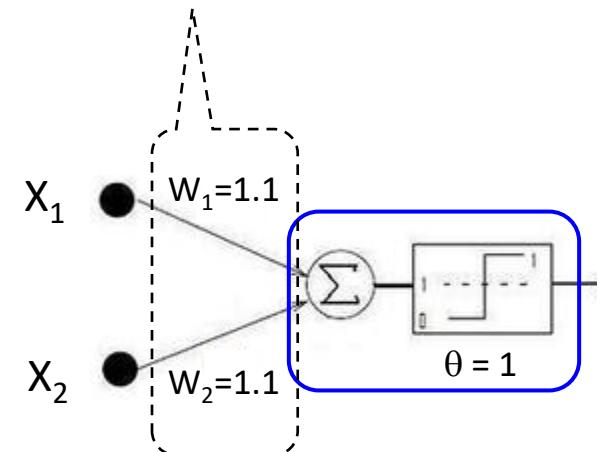
Pesos sinápticos fixados manualmente



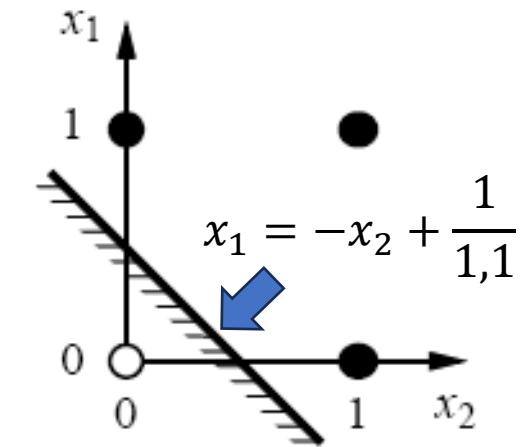
Classes linearmente separáveis

Regra OU

Entrada X1	Entrada X2	Saída Y
0	0	0
0	1	1
1	0	1
1	1	1



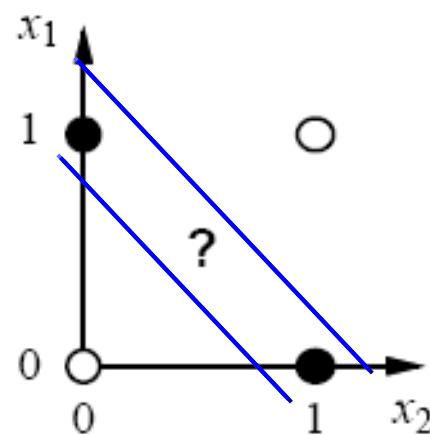
Se $1,1X_1 + 1,1X_2 > 1 \rightarrow Y=1$
Se $1,1X_1 + 1,1X_2 \leq 1 \rightarrow Y=0$



Funções Booleanas

Regra XOR

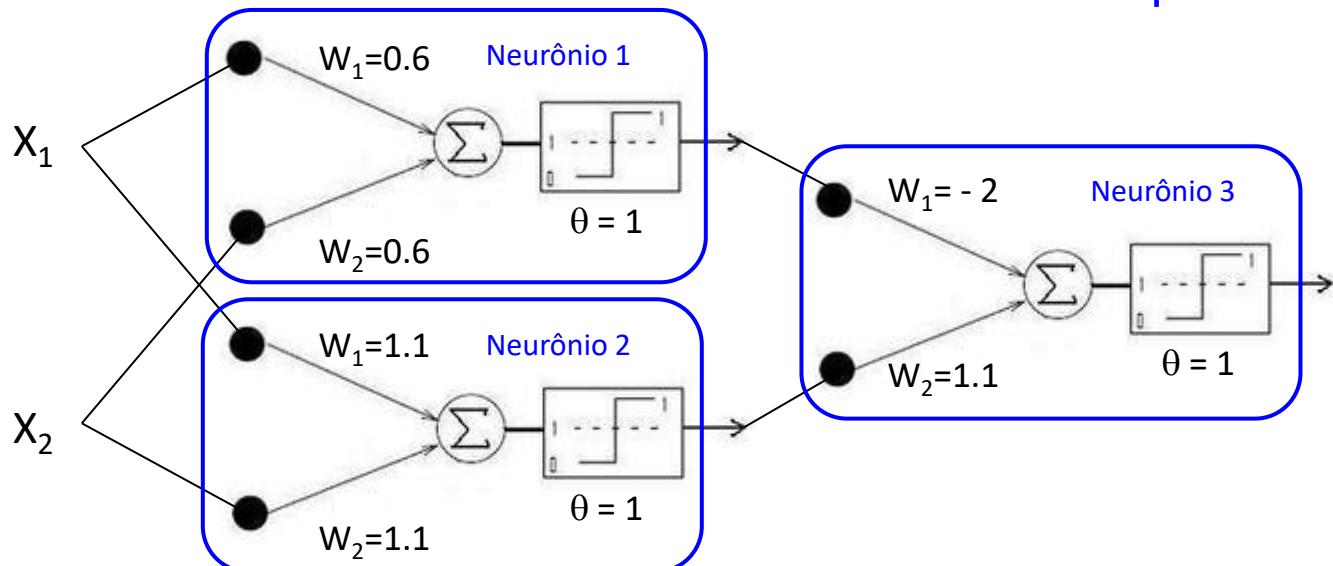
Neurônio 1	Neurônio 2	Saída
0	0	0
0	1	1
0	1	1
1	1	0



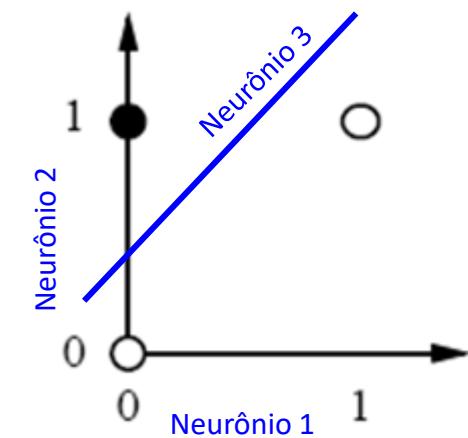
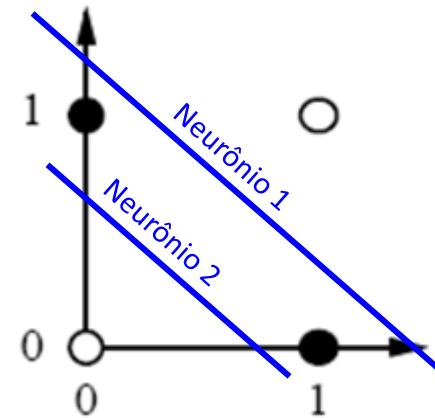
No caso do XOR, não existe uma única reta que separe os pontos $(0,0)$ e $(1,1)$ para um lado, e $(0,1)$ e $(1,0)$ do outro lado.

Um único neurônio artificial só funciona com classes linearmente separáveis

Rede com três neurônios



Funções booleanas mais complexas podem se implementadas por redes de múltiplos neurônios.





Donald O. Hebb
1904 - 1985

The Organization of Behavior

A NEUROPSYCHOLOGICAL THEORY

D. O. HEBB
McGill University

https://pure.mpg.de/rest/items/item_2346268_3/component/file_2346267/content

Aprendizagem neuronal (1949)

Primeiro a propor uma lei de adaptação das sinapses durante o processo de aprendizagem no livro *The Organization of Behavior* (1949).

A informação está armazenada nas conexões sinápticas entre dois neurônios e o aprendizado ocorre pela alteração das conexões com base em padrões repetidos de ativação.

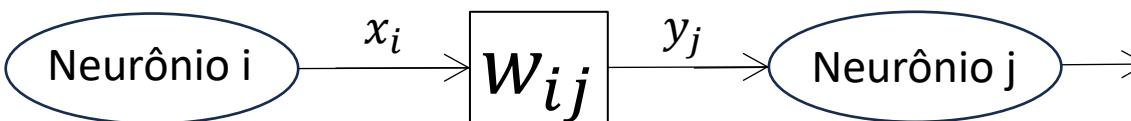
A conexão entre dois neurônios é reforçada se ambos são ativados simultaneamente. Se os dois neurônios são ativados assincronamente então a sinapse entre eles deve ser enfraquecida.

"Neurônios que disparam juntos, conectam-se juntos"

correlação + sinapse fortalecida (w aumenta)

correlação - sinapse enfraquecida (w diminui)

O aprendizado envolve incrementos Δw nas sinapses. Armazenamento distribuído da memória.



Atualização das sinapses
durante o aprendizado

$$w_{ij}(\text{novo}) = w_{ij}(\text{velho}) + \eta x_i y_j \quad \Delta w_{ij}$$

$0 < \eta < 1$ taxa de aprendizagem



Frank Rosenblatt
1928 -1971

"A criação de máquinas dotadas de qualidades humanas há muito tempo representa uma área fascinante da ficção científica. No entanto, estamos prestes a testemunhar o surgimento de uma máquina exatamente desse tipo: capaz de perceber, reconhecer e identificar o que a rodeia sem qualquer treinamento ou controle por parte do ser humano."

Implementação do primeiro modelo de neurônio artificial: o Perceptron (1958). Rede neural com camada única.

Saída binária (-1 OU +1) é a função degrau da soma dos estímulos (entradas -1 OU +1), ponderados por pesos com valores reais.

Pesos sinápticos adaptáveis.

Método de treinamento supervisionado para ajuste dos pesos sinápticos.

Perceptron (1958)

[Professor's perceptron paved the way for AI – 60 years too soon | Cornell Chronicle](#)

Vol. VI, No. 2, Summer 1958

research trends

CORNELL AERONAUTICAL LABORATORY, INC., BUFFALO 21, NEW YORK

The Design of an

Intelligent AUTOMATON

by FRANK ROSENBLATT

Introducing the perceptron — A machine which senses, recognizes, remembers, and responds like the human mind.

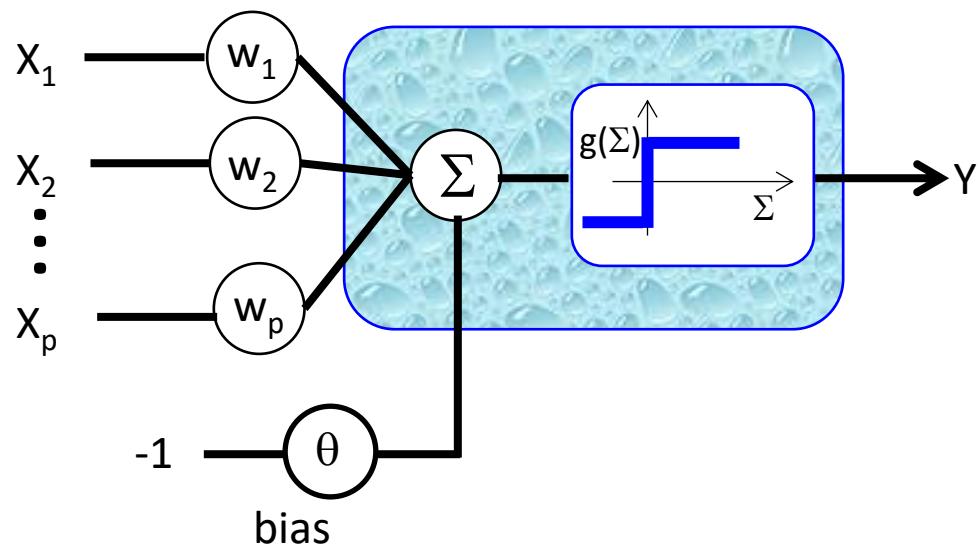
Perceptron (1958)

Saída binária: -1 ou +1

Função de ativação: Degrau

Regra de propagação: $w_1X_1 + \dots + w_pX_p - \theta$

(1958)
F. Rosenblatt



$$Y = g(w_1X_1 + \dots + w_pX_p - \theta) = g(w^T X)$$

Se $w_0 + w_1X_1 + \dots + w_pX_p \geq \theta \rightarrow Y=1$ (pulso)

Se $w_0 + w_1X_1 + \dots + w_pX_p < \theta \rightarrow Y=-1$ (não pulso)

The perceptron: a probabilistic model
for information storage and organization in the brain
Psychological Review 65:386–408

If we are eventually to understand the capability of higher organisms for perceptual recognition, generalization, recall, and thinking, we must first have answers to three fundamental questions:

1. How is information about the physical world sensed, or detected, by the biological system?
2. In what form is information stored, or remembered?
3. How does information contained in storage, or in memory, influence recognition and behavior?

and the stored pattern. According to this hypothesis, if one understood the code or "wiring diagram" of the nervous system, one should, in principle, be able to discover exactly what an organism remembers by reconstructing the original sensory patterns from the "memory traces" which they have left, much as we might develop a photographic negative, or translate the pattern of electrical charges in the "memory" of a digital computer. This hypothesis is appealing in its simplicity and ready intelligibility, and a large family of theoretical brain

Aprendizagem do Perceptron: Treinamento supervisionado

Valores iniciais são atribuídos aleatoriamente aos pesos sinápticos w e limiar θ

Para uma vetor de entrada $X^d = (x_1^d, \dots, x_i^d, \dots, x_p^d)$ deseja-se uma saída y^d

A saída do perceptron pode ser $y = -1$ ou $y = +1$

Rosenblatt empregou a regra de Hebb para alterar os pesos sinápticos w .

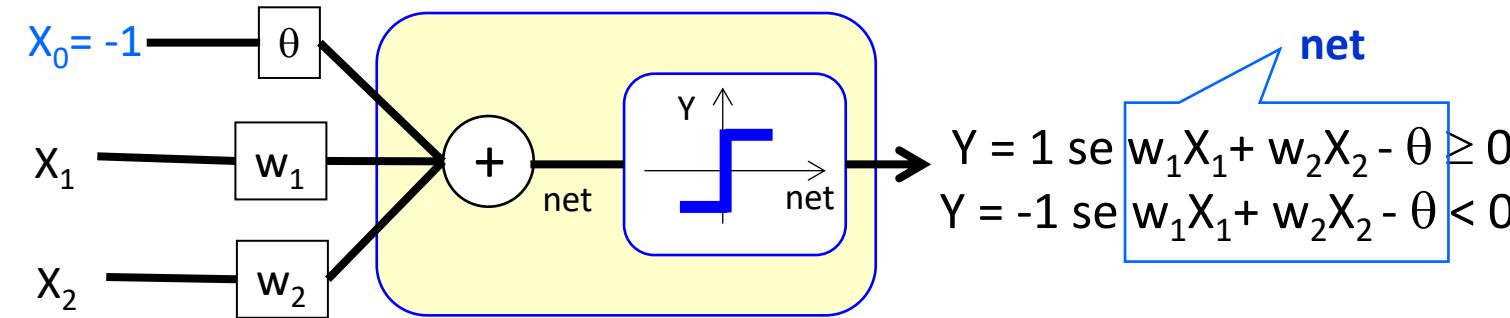
$$\theta^{novo} = \theta^{velho} + \Delta\theta \quad \Delta\theta = \eta(y^d - y)$$
$$w_i^{novo} = w_i^{velho} + \Delta w_i \quad \Delta w_i = \eta(y^d - y)x_i^d$$

Erro

Para $y_d = y$ o erro é zero, logo não há necessidade de modificar os pesos sinápticos $\Delta w = 0$.

$$\left. \begin{array}{l} \text{Para } y^d = 1 \text{ e } y = -1, \text{ erro} = 2 \rightarrow \Delta w_i = 2\eta x_i^d \\ \text{Para } y^d = -1 \text{ e } y = 1, \text{ erro} = -2 \rightarrow \Delta w_i = -2\eta x_i^d \end{array} \right\} \Delta w_i = 2\eta y^d x_i^d$$

Treinamento supervisionado



Considere $\eta=1$ e pesos sinápticos (w_1, w_2) e bias (θ) inicializados com valores nulos, ou seja, $(\theta, w_1, w_2)^T = (0,0,0)^T$.

Aprendizagem dos pesos sinápticos:

$$\begin{pmatrix} \theta_{novo} \\ w_{1,novo} \\ w_{2,novo} \end{pmatrix} = \begin{pmatrix} \theta_{velho} \\ w_{1,velho} \\ w_{2,velho} \end{pmatrix} + \eta \begin{pmatrix} X_{0,i} \\ X_{1,i} \\ X_{2,i} \end{pmatrix} \boxed{(Y_i^d - Y_i)}$$

erro

$$w(novo) = w(velho) + \eta x_i (y_i^d - y_i)$$

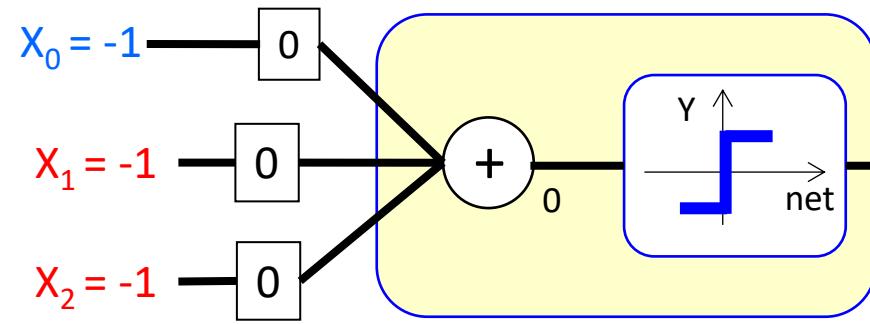
Δw_{ij}

Exemplos de treinamento

Entradas			Saída
X_0	X_1	X_2	Y
-1	-1	-1	-1
-1	-1	1	-1
-1	1	-1	-1
-1	1	1	1

Exemplos de treinamento

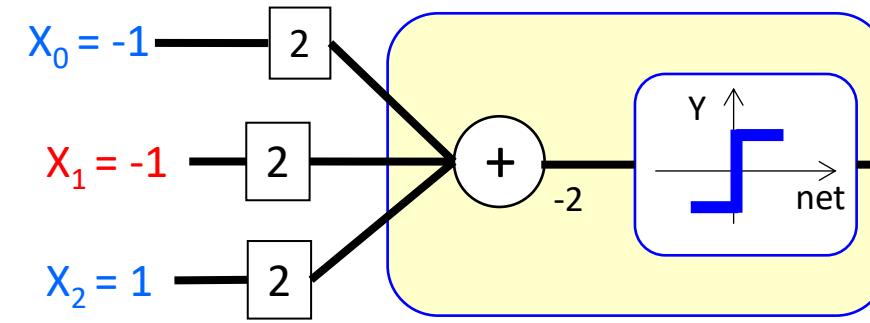
Entradas			Saída
x_0	x_1	x_2	y
-1	-1	-1	-1
-1	-1	1	-1
-1	1	-1	-1
-1	1	1	1



$$Y = 1$$

$$\text{erro} = -1 - (1) = -2$$

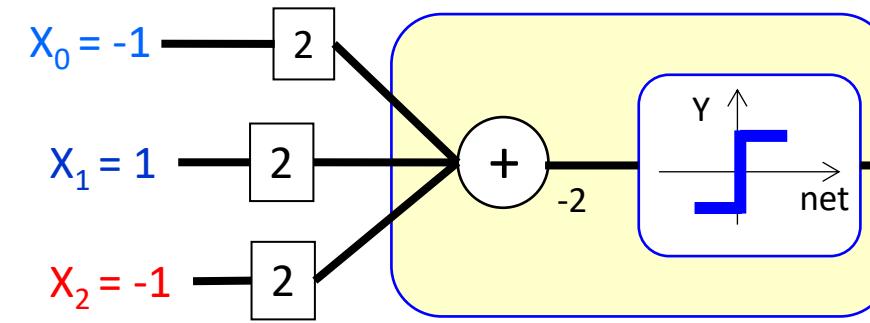
$$w(\text{novo}) = (0 \ 0 \ 0)^T + (-1 \ -1 \ -1)^T(-2) = (2 \ 2 \ 2)$$



$$Y = -1$$

$$\text{erro} = -1 - (-1) = 0$$

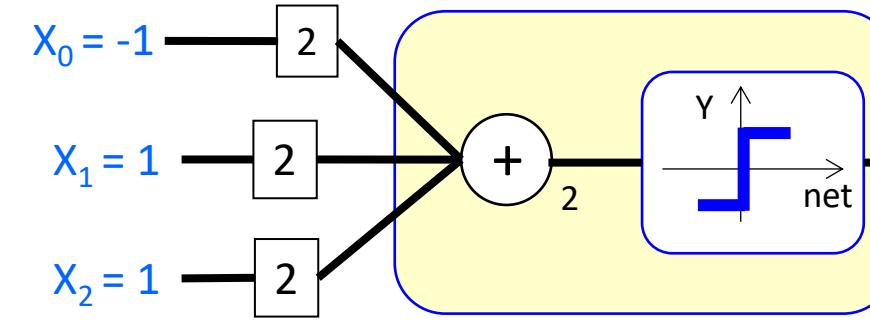
$$w(\text{novo}) = (2 \ 2 \ 2)^T + (-1 \ -1 \ 1)^T(0) = (2 \ 2 \ 2)$$



$$Y = -1$$

$$\text{erro} = -1 - (-1) = 0$$

$$w(\text{novo}) = (2 \ 2 \ 2)^T + (1 \ -1 \ 1)^T(0) = (2 \ 2 \ 2)$$

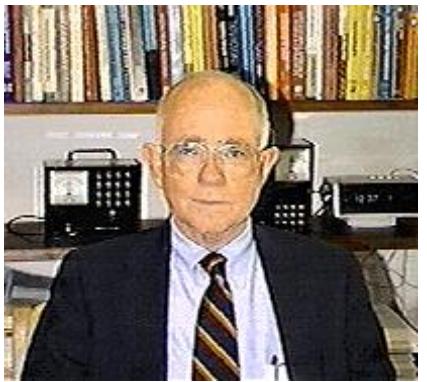


$$Y = 1$$

$$\text{erro} = 1 - (1) = 0$$

$$w(\text{novo}) = (2 \ 2 \ 2)^T + (1 \ 1 \ 1)^T(0) = (2 \ 2 \ 2)$$

ADALINE - ADAptive LInear NEuron (1960)



Bernard Widrow
1929

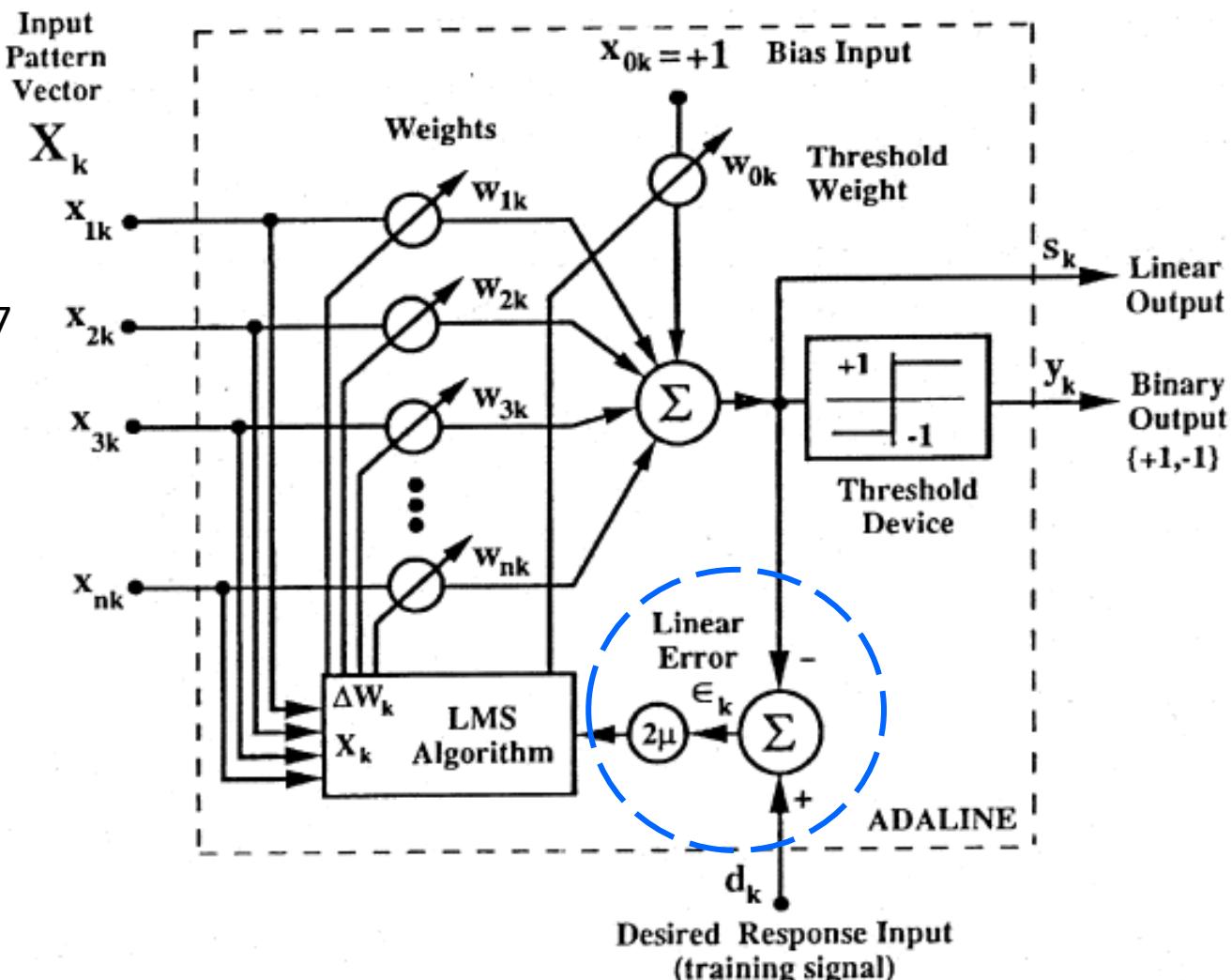


Marcian Edward "Ted" Hoff, Jr. 1937
Inventor do Intel 4004 (1971)



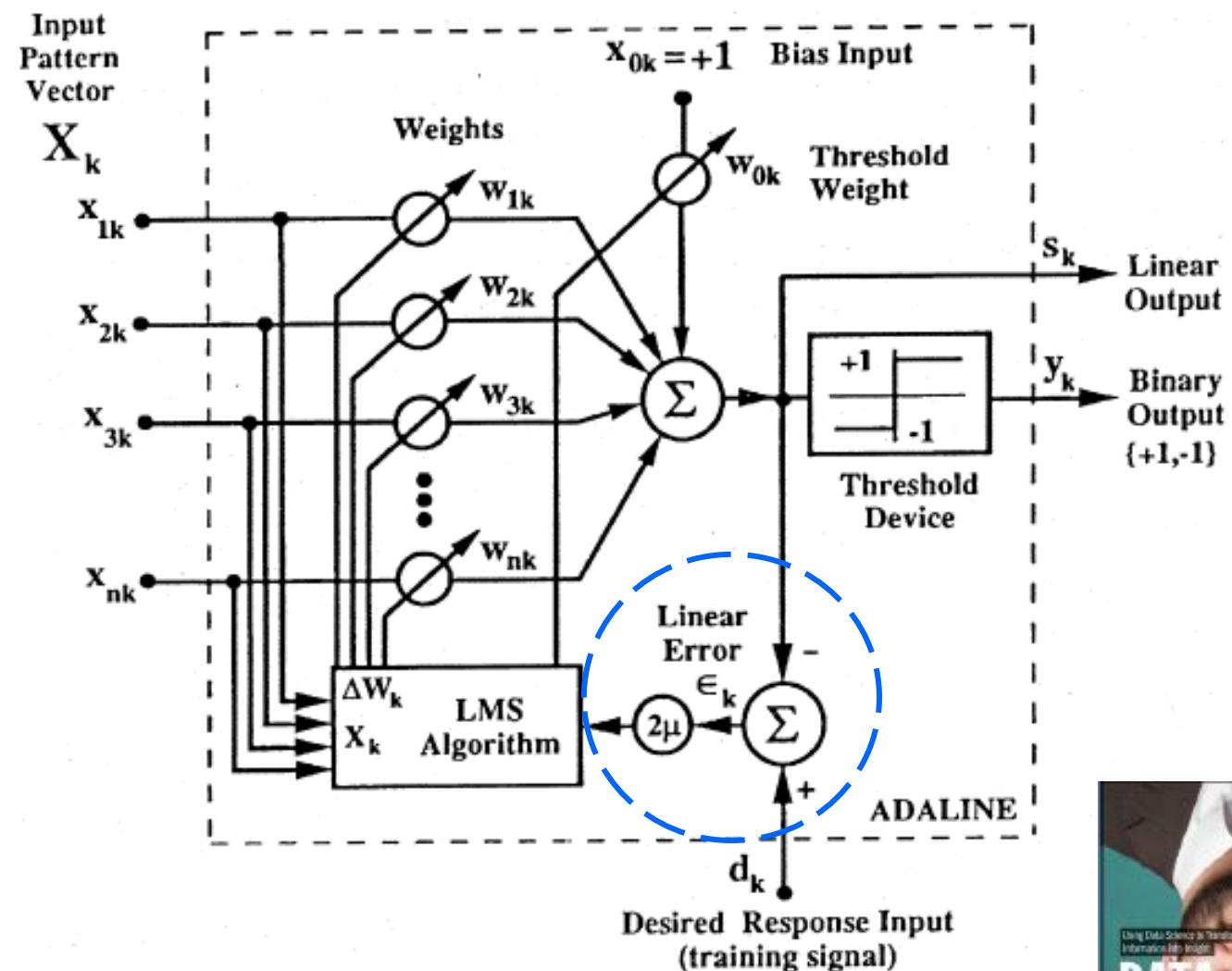
Algoritmo *Least Mean Square (LMS)* para ajustar os pesos sinápticos de forma a minimizar o erro entre a saída desejada e a saída do filtro (Regra Delta).

O ADALINE é similar ao Perceptron, porém emprega um neurónio com função de ativação linear.



Widrow & Lehr (1990)

Regra Delta de Widrow ou Método do Gradiente



Widrow & Lehr (1990)

Algoritmo de treinamento supervisionado ajusta os pesos $w=(w_0, \dots, w_p)$ com o objetivo de minimizar o erro quadrático médio $E(w)$ entre a saída da rede $y=w^T x$ e a saída desejada y^d .

$$w = (w_1, w_2, \dots, w_p) \quad E(w) = \sum_{i=1}^n \left(\sum_{j=0}^p w_j x_{i,j} - y_i^d \right)^2$$

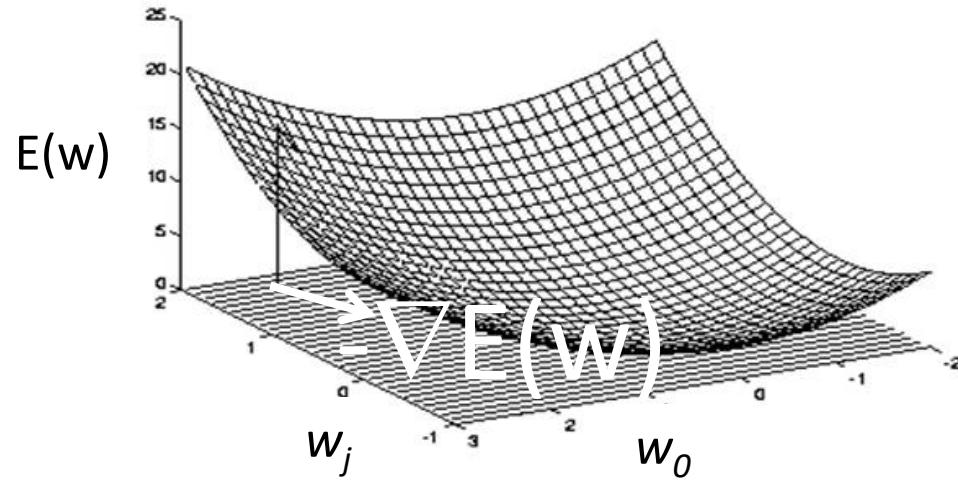
saída da rede para o i-ésimo exemplo

saída desejada do i-ésimo exemplo



Modelo de regressão linear, o avô da Inteligência Artificial Supervisionada (FOREMAN)

Regra Delta de Widrow ou Método do Gradiente



$$\underset{w=(w_1, w_2, \dots, w_p)}{\text{Min}} \quad E(w) = \sum_{i=1}^n \left(\sum_{j=0}^p w_j x_{i,j}^d - y_i^d \right)^2$$

saída da rede para o i-ésimo exemplo

saída desejada do i-ésimo exemplo

Pesos w são inicializados aleatoriamente.

$$w = \begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_p \end{pmatrix}$$

Regra de atualização dos pesos. $w^{novo} = w^{velho} + \Delta w$

$$\Delta w = \begin{pmatrix} \Delta w_0 \\ \Delta w_1 \\ \vdots \\ \Delta w_p \end{pmatrix} = -\eta \nabla E(w^{velho})$$

Gradiente da função erro quadrático médio

$$\nabla E(w) = \left(\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \dots, \frac{\partial E}{\partial w_p} \right)$$

Regra Delta de Widrow ou Método do Gradiente

Função objetivo

$$\underset{w=(w_1, w_2, \dots, w_p)}{\text{Min}} \quad E(w) = \sum_{i=1}^n \left(\sum_{j=0}^p w_j x_{i,j}^d - y_i^d \right)^2$$

saída da rede para o i-ésimo exemplo

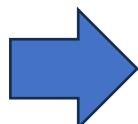
saída desejada do i-ésimo exemplo



Elementos do gradiente de $E(w)$

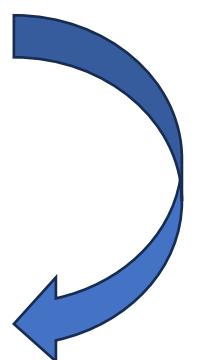
$$\frac{\partial E(w)}{\partial w_j} = 2 \sum_{i=1}^n x_{i,j}^d \left(\sum_{j=0}^p w_j^{velho} x_{i,j}^d - y_i^d \right)$$

δ_i = erro no i-ésimo exemplo



j-ésimo elemento de Δw

$$\Delta w_j = -\eta \frac{\partial E(w^{velho})}{\partial w_j} = -\eta \sum_{i=1}^n x_{i,j}^d \delta_i$$



Atualiza j-ésimo elemento de w

$$w_j^{novo} = w_j^{velho} - \eta \sum_{i=1}^n x_{i,j}^d \delta_i$$

Regra Delta X Regra de Hebb

Modificação dos pesos realizada iterativamente para reduzir a diferença (delta δ) entre a saída desejada e a saída da rede a assim minimizar o erro médio quadrático (treinamento supervisionado).

Para cada padrão de treinamento apresentado à rede, um novo ajuste nos pesos é realizado

$$\text{Regra Delta} \quad w^{novo} = w^{velho} - \eta X_i^d \delta_i$$

Padrão de entrada $X_i^d = (1, x_{i,1}^d, \dots, x_{i,p}^d)$ **Erro** $\delta_i = \sum_{j=0}^p w_j^{velho} x_{i,j}^d - y_i^d$

A regra Delta é uma variante da regra de Hebb

$$\text{Regra de Hebb} \quad w^{novo} = w^{velho} + 2\eta y_i^d X_i^d$$

modificação proporcional a saída

Minsky e Papert (1969)

No livro *Perceptrons: an Introduction to Computational Geometry*, Minsky e Papert mostram que a capacidade dos *perceptrons* é limitada: os *perceptrons* só podem separar categorias linearmente separáveis.

Por exemplo, com apenas um *perceptron* não é possível resolver problemas não linearmente separáveis, como o operador XOR.

O problema do XOR pode ser resolvido com a adição de uma camada intermediária de processadores.

Minsky e Papert consideraram a possibilidade de generalizar *perceptrons* de camada única para múltiplas camadas, mas duvidavam de que houvesse uma maneira de treinar esses *perceptrons* (SEJNOWSKI, 2019).

Ninguém na década de 1960 sabia como treinar uma rede com uma única camada escondida (oculta) entre as camadas de entrada e saída (SEJNOWSKI, 2019).

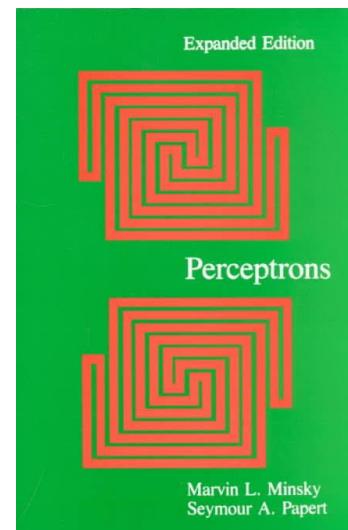
Como relacionar os pesos sinápticos das entradas dos neurônios da camada escondida com o erro da rede? (KELLEHER, 2019).



Marvin Minsky
1927 - 2016



Seymour Aubrey Papert
1928 - 2016



[Minsky-and-Papert-Perceptrons.pdf](#)

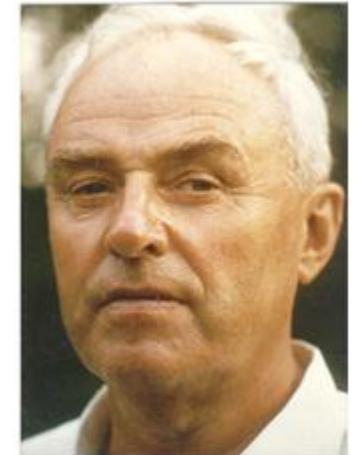
A capa do livro mostra um problema que o *perceptron* não consegue resolver, o *perceptron* não consegue distinguir os dois labirintos

Group Method Data Handling – GMDH (1971)

Redes neurais polinomiais introduzidas por Ivakhnenko em 1971.

O GMDH é uma rede neural capaz de relacionar m variáveis independentes x_1, \dots, x_m com uma única variável dependente y por meio de um polinômio de alta ordem.

$$y = a + \sum_{i=1}^m b_i x_i + \sum_{i=1}^m \sum_{j=1}^m c_{ij} x_i x_j + \sum_{i=1}^m \sum_{j=1}^m \sum_{k=1}^m d_{ijk} x_i x_j x_k + \sum_{i=1}^m \sum_{j=1}^m \sum_{k=1}^m \sum_{l=1}^m e_{ijkl} x_i x_j x_k x_l + \dots$$



Alexey G. Ivakhnenko
1913-2007

Na GMDH cada neurônio tem duas variáveis de entrada x_i e x_j e uma variável de saída y relacionadas por meio de um polinômio com a seguinte especificação:

$$y = A + Bx_i + Cx_j + Dx_i^2 + Ex_j^2 + Fx_i x_j$$

Modelo de regressão linear, o avô da Inteligência Artificial Supervisionada (FOREMAN)

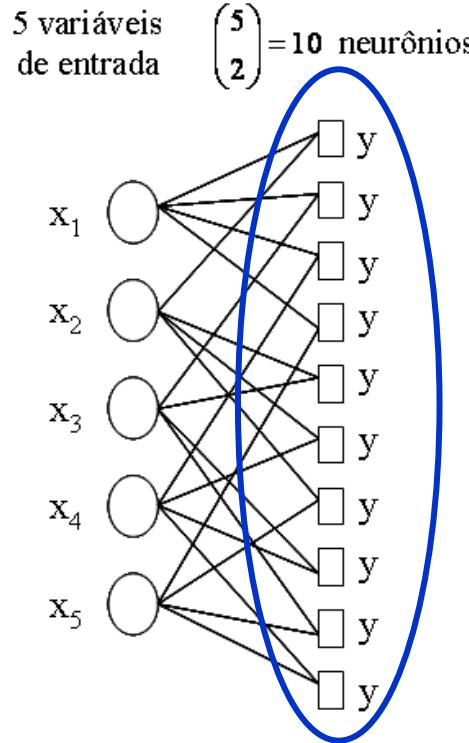
Dada uma amostra com n observações de uma variável dependente y e m variáveis independentes x_1, \dots, x_m , o algoritmo de aprendizagem da GMDH constrói a estrutura da rede gradativamente até que a configuração ótima seja alcançada.

Rede auto-organizável: determina automaticamente o número de camadas escondidas e o número de neurônios em cada camada.

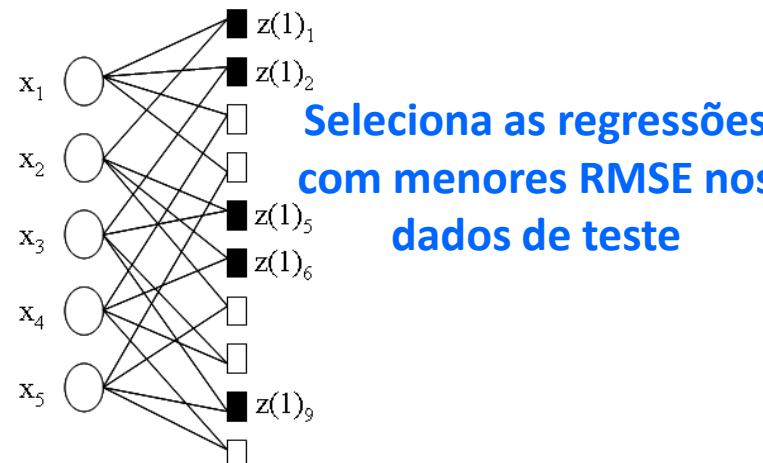
Rede com 8 camadas,
primeiro exemplo de
rede neural profunda

Group Method Data Handling - GMDH

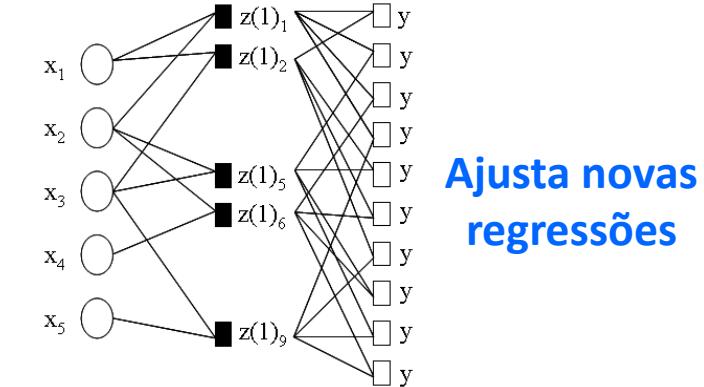
1) adição da primeira camada



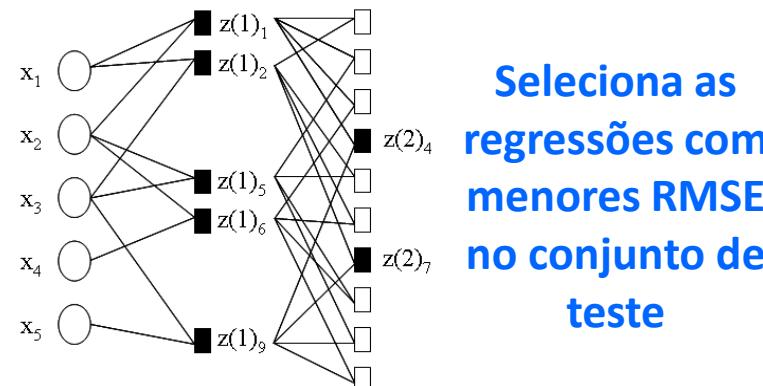
2) melhores regressões polinomiais



3) adição da segunda camada

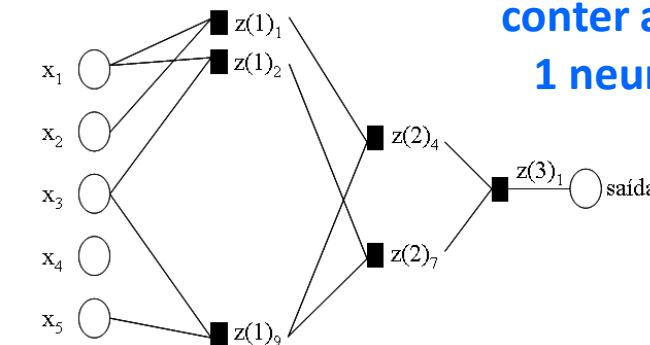


4) melhores regressões polinomiais



5) Configuração final

A camada de saída deve conter apenas 1 neurônio



Previsão da velocidade do vento 1 hora à frente

Parque eólico na Galícia – Espanha

- Capacidade instalada 17,56 MW
- 24 aerogeradores
- 5 tecnologias e 9 fabricantes diferentes
- Terreno complexo
- Velocidade média anual 6,41 m/s
- Produção anual da ordem de 33MWh



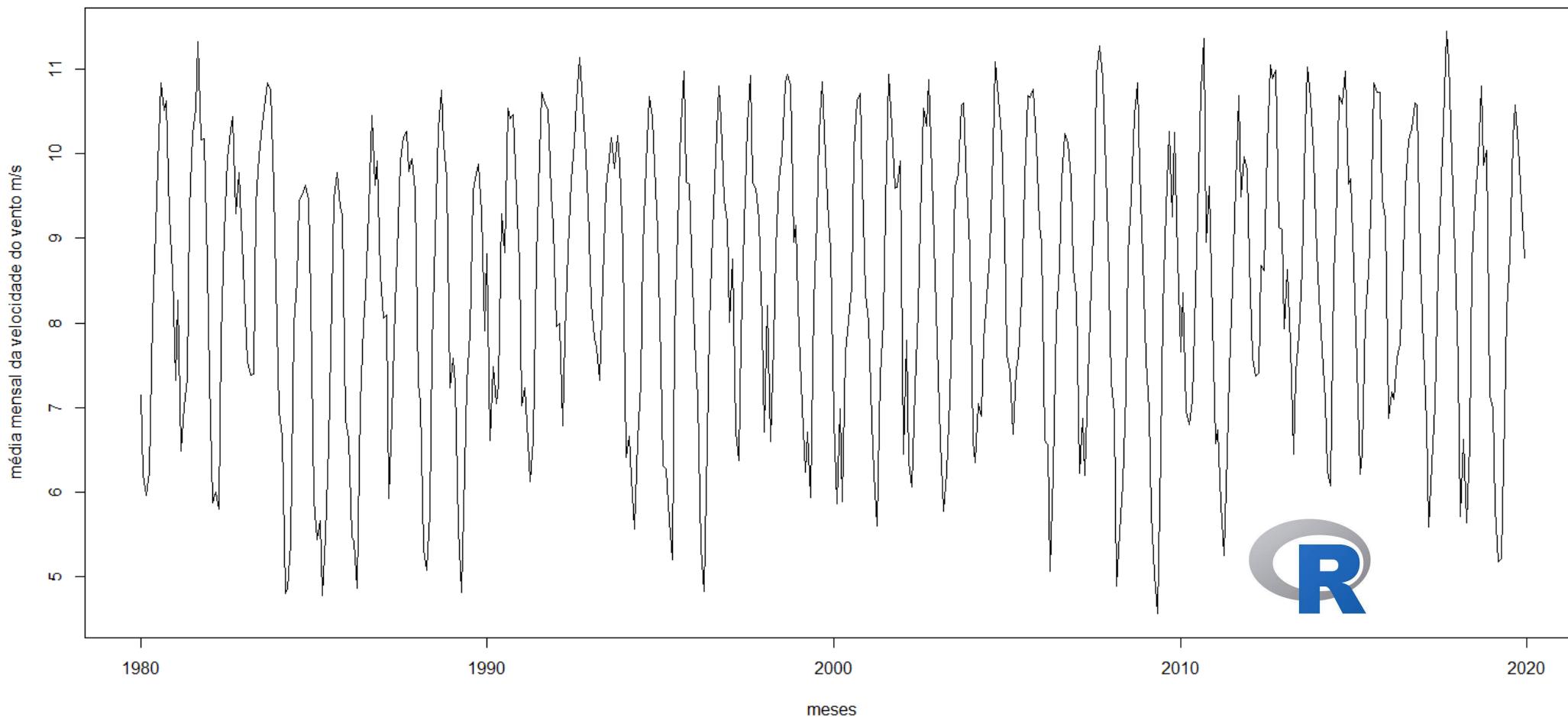
Previsão da média mensal da velocidade do vento com o pacote GMDH

Médias mensais das reanálises da velocidade do vento em Acaraú II 230 kV (MERRA 2, [Renewables.ninja](https://renewables.ninja))



Renewables.ninja

480 médias
mensais de 1980/1
até 2019/12



Previsão da média mensal da velocidade do vento com o pacote GMDH

```
arquivo="c:/ventos/AcarauII230_mensal_merra.csv"
dados=read.csv2(arquivo,header=F,dec=". ",sep=";")
serie=ts(dados[,3],start=c(1980,1),freq=12)
insample=serie[1:456]
outsample=serie[457:480] # 24 últimas observações para comparar com as previsões 24 passos à frente
```



```
library(GMDH)
```

```
Z=insample
```

```
serie=ts(Z,start=c(1980,1),freq=12)
```

```
for (i in 1:5{
```

```
    previsao=fcast(serie, method = "GMDH", input = 12, layer = 12, f.number = 5)
```

```
    Z=c(Z,previsao$mean[1:5])
```

```
    serie=ts(Z,start=c(1980,1),freq=12)
```

```
}
```

```
apenas_previsoes=tail(Z,25)[-25] # retorna a 25ª previsão
```

```
minimo=min(outsample, apenas_previsoes)
```

```
maximo=max(outsample,apenas_previsoes)
```

```
plot(outsample,ylim=c(minimo,maximo),pch=20,col="black",xlab="meses",ylab="m/s")
```

```
lines(apenas_previsoes,col="blue",lwd=2)
```

```
MAPE=mean(abs(outsample- apenas_previsoes)/outsample)
```

```
print(MAPE*100)
```

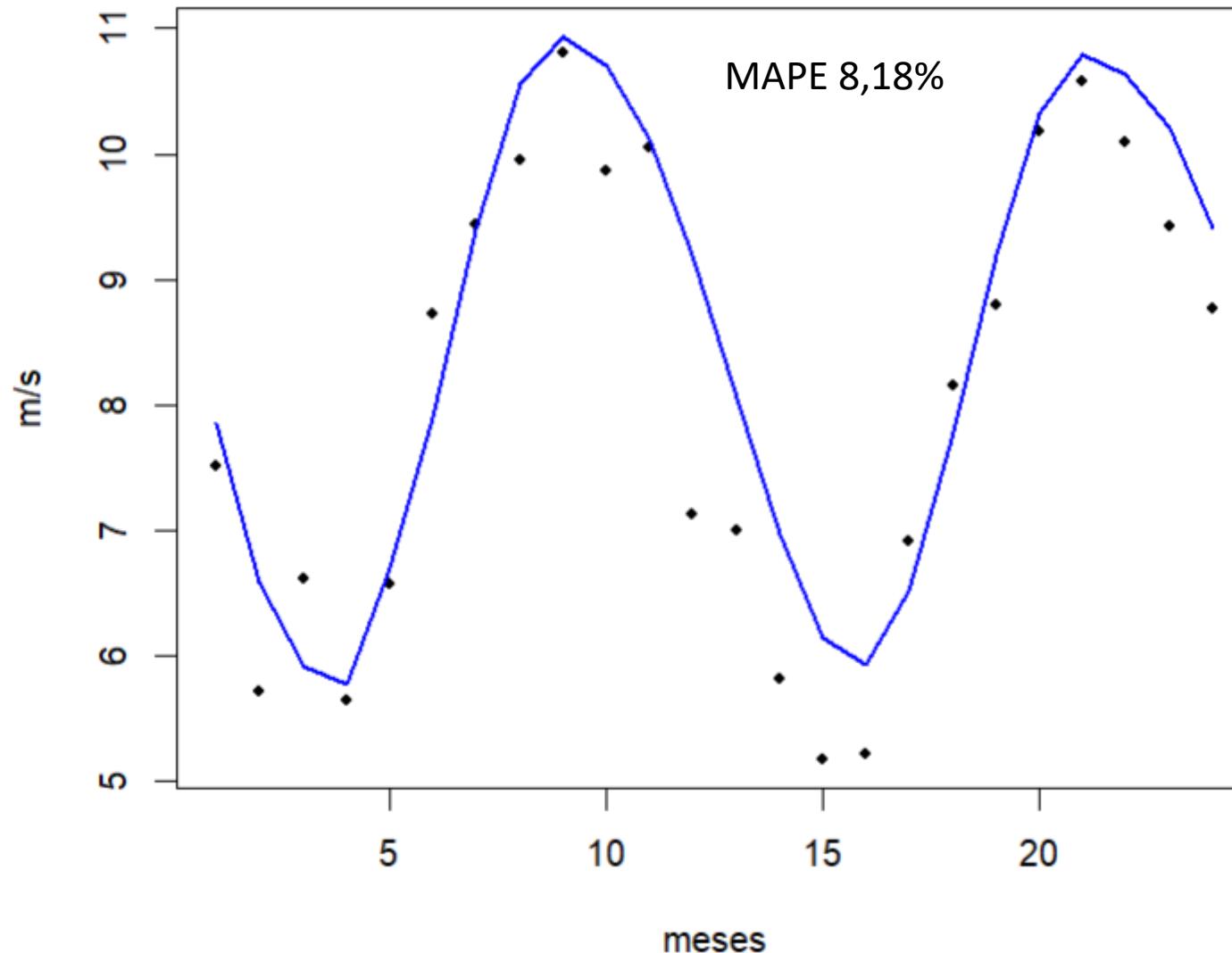
Faz gráfico

Faz gráfico e calcula o
Mean Absolute
Percentage Error (MAPE)

$$MAPE = \frac{1}{n} \sum_{t=1}^n \frac{|observado_t - previsto_t|}{observado_t} \times 100\%$$

Previsão da média mensal da velocidade do vento com o pacote GMDH

Previsões mensais até 24 meses à frente



Retropropagação do erro (1974)

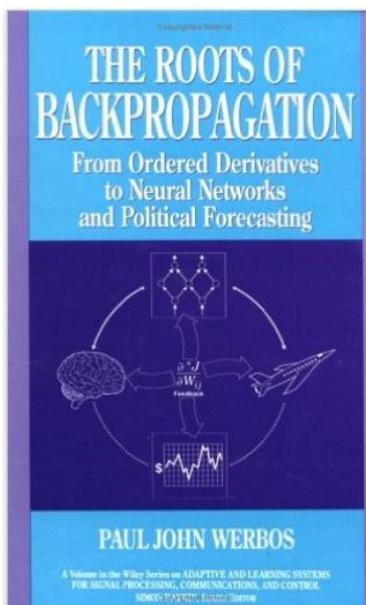
Paul Werbos Introduziu o método de retropropagação do erro (*error backpropagation*).



Paul J. Werbos
1947

Tese de doutorado em Harvard: *Beyond regression: new tools for prediction and analysis in the behavioral sciences* [Beyond regression: new tools for prediction and analysis in the behavioral sciences](#)

Simon Haykin, the editor of this series, offered to publish my 1974 Ph.D. thesis, *Beyond Regression*, because this thesis has become something of a classic reference in the neural network and engineering world. The thesis is now recognized as the original source of *backpropagation*, which is now the most widely used algorithm by far in the neural network world. Also, the thesis *communicates* that algorithm to an audience with *no prior understanding* of neural networks;



Rede de Hopfield (1982)

Neural networks and physical systems with emergent collective computational abilities

<https://pmc.ncbi.nlm.nih.gov/articles/PMC346238/pdf/pnas00447-0135.pdf>



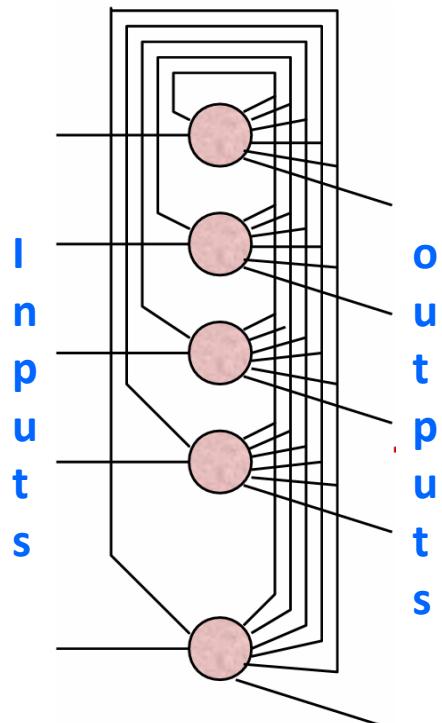
John Hopfield
1933

J. J. HOPFIELD

Division of Chemistry and Biology, California Institute of Technology, Pasadena, California 91125; and Bell Laboratories, Murray Hill, New Jersey 07974

Contributed by John J. Hopfield, January 15, 1982

- Rede neural recorrente com uma camada e com neurônios totalmente conectados (*perceptrons* com saída -1 e +1)
- As conexões possuem pesos simétricos ($w_{ij} = w_{ji}$) e não há conexões de um neurônio para ele mesmo $w_{ii}=0$ (condições para a estabilidade da rede neural)
- Armazena padrões nos pesos.
- Utilizada em problemas de memória associativa (recuperar dados danificados)
- Um rede com N neurônios pode armazenar $0,138N$ padrões binários, logo para armazenar 10 padrões seriam necessários 74 neurônios



Máquina de Boltzman (1985)

O algoritmo de aprendizado da máquina de Boltzman conseguia **aprender a resolver problemas que exigem unidades ocultas**, o que contrariava a opinião de Marvin Minsky, Seymour Papert e da maioria dos pesquisadores da área de que era impossível treinar uma rede multicamadas e superar as limitações do perceptron (SEJNOWSKI, 2019).

COGNITIVE SCIENCE 9, 147-169 (1985)

A Learning Algorithm for Boltzmann Machines*

DAVID H. ACKLEY

GEOFFREY E. HINTON

*Computer Science Department
Carnegie-Mellon University*

TERRENCE J. SEJNOWSKI

*Biophysics Department
The Johns Hopkins University*



David H. Ackley



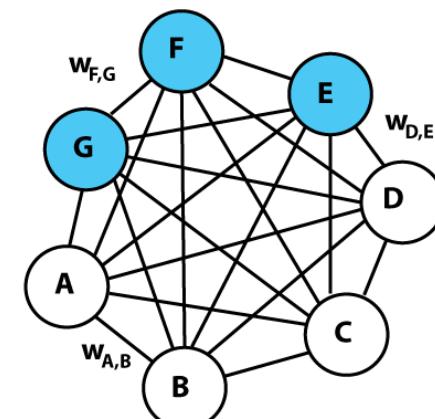
Geoffrey E. Hinton

1947



1947

**3 unidades visíveis
e 4 ocultas**



Rumelhart, Hinton e Williams (1986)

Redescobriram e desenvolveram o método de retropropagação do erro para o ajuste dos pesos sinápticos.

O método de retropropagação do erro permite treinar eficientemente redes com camadas intermediárias (*Multilayer Perceptron MLP*).



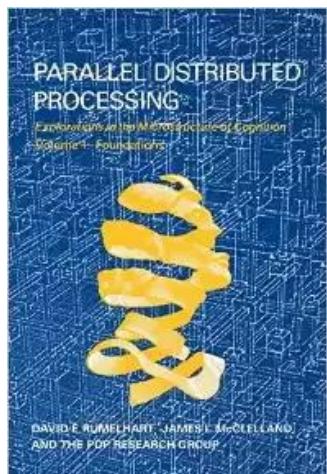
David Rumelhart
1942 - 2011



Geoffrey E. Hinton
1947



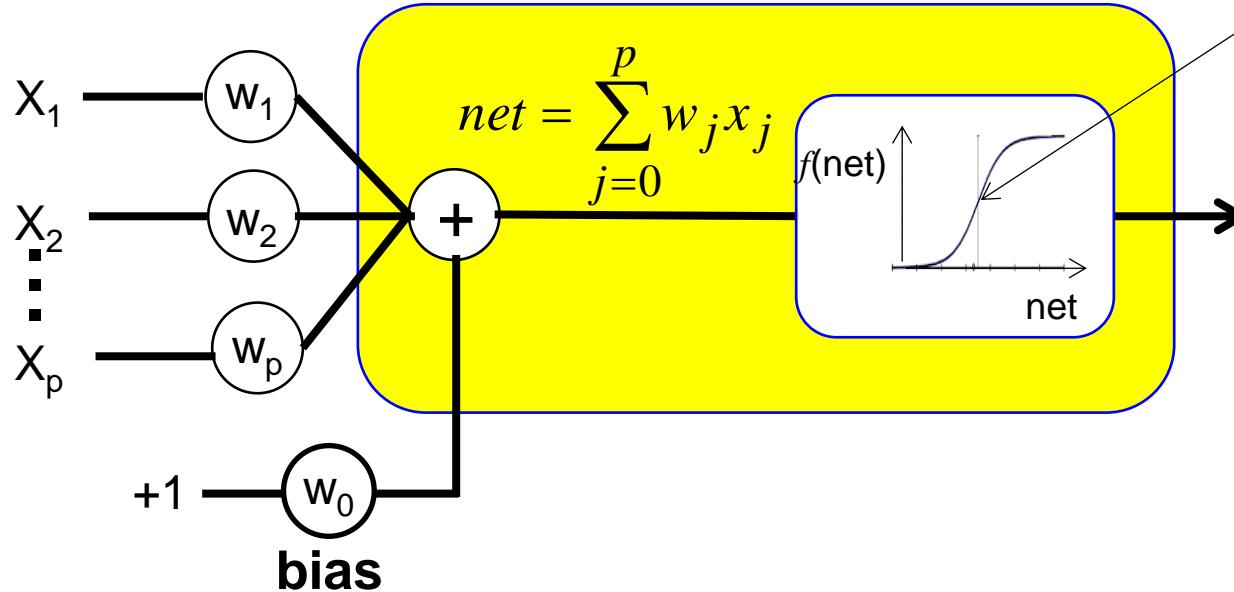
Ronald J. Williams
1945-2024



Rumelhart, D. E., Hinton, G. E. & Williams, R. J. in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*. Vol. 1: *Foundations* (eds Rumelhart, D. E. & McClelland, J. L.) 318–362 (MIT, Cambridge, 1986)

All the knowledge is in the connections
David Rumelhart

Neurônio Artificial

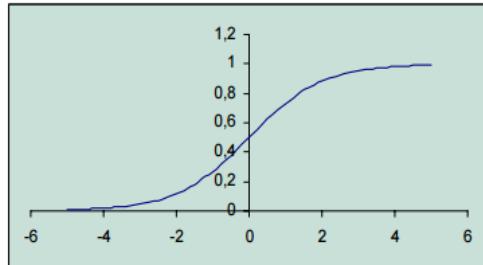


Função de ativação derivável

Embora os neurônios com função sigmoide sejam mais biologicamente plausíveis do que os neurônios com função tangente hiperbólica, estes últimos apresentam melhor desempenho no treinamento de redes neurais
[glorot11a.pdf](#) multicamadas

Funções de ativação típicas

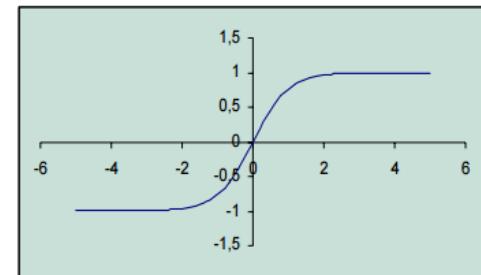
$f(net) = net$ linear



$f(net) = \frac{1}{1 + \exp(-net)}$ sigmóide

tangente hiperbólica

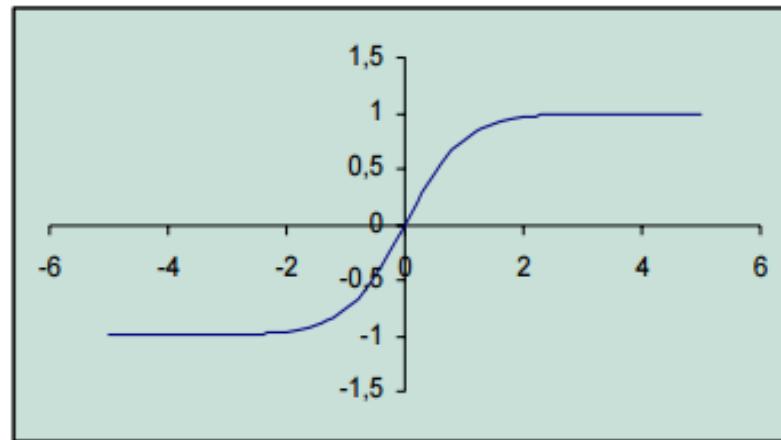
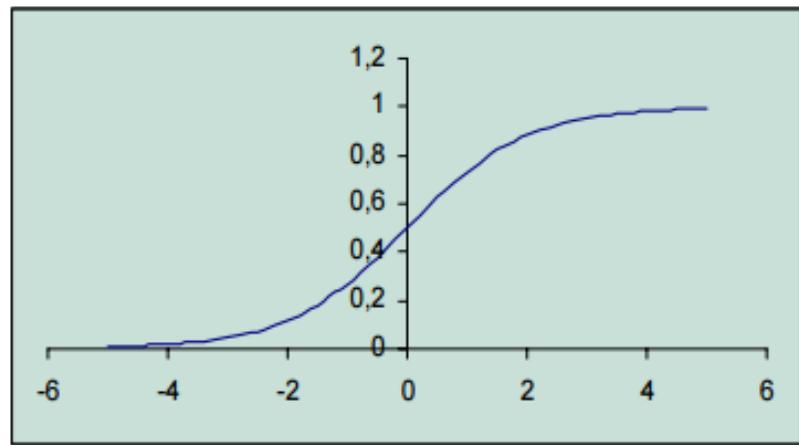
$f(net) = \frac{\exp(net) - \exp(-net)}{\exp(net) + \exp(-net)}$



Xavier Glorot

Normalização do padrão de entrada e saída

Uma característica das funções sigmoide e tangente hiperbólica é a saturação, ou seja, para valores grandes de argumento, a função opera numa região de saturação.



É importante portanto trabalhar com valores de entrada que estejam contidos num intervalo que não atinjam a saturação, por exemplo: $[0, 1]$ ou $[-1, +1]$.

Multilayer Perceptron (MLP)

É a arquitetura de RNA mais difundida, eficaz e mais fácil de modelar redes complexas

A retropropagação do erro é uma variação da regra delta, apropriada para redes multi-camadas: a regra delta generalizada.

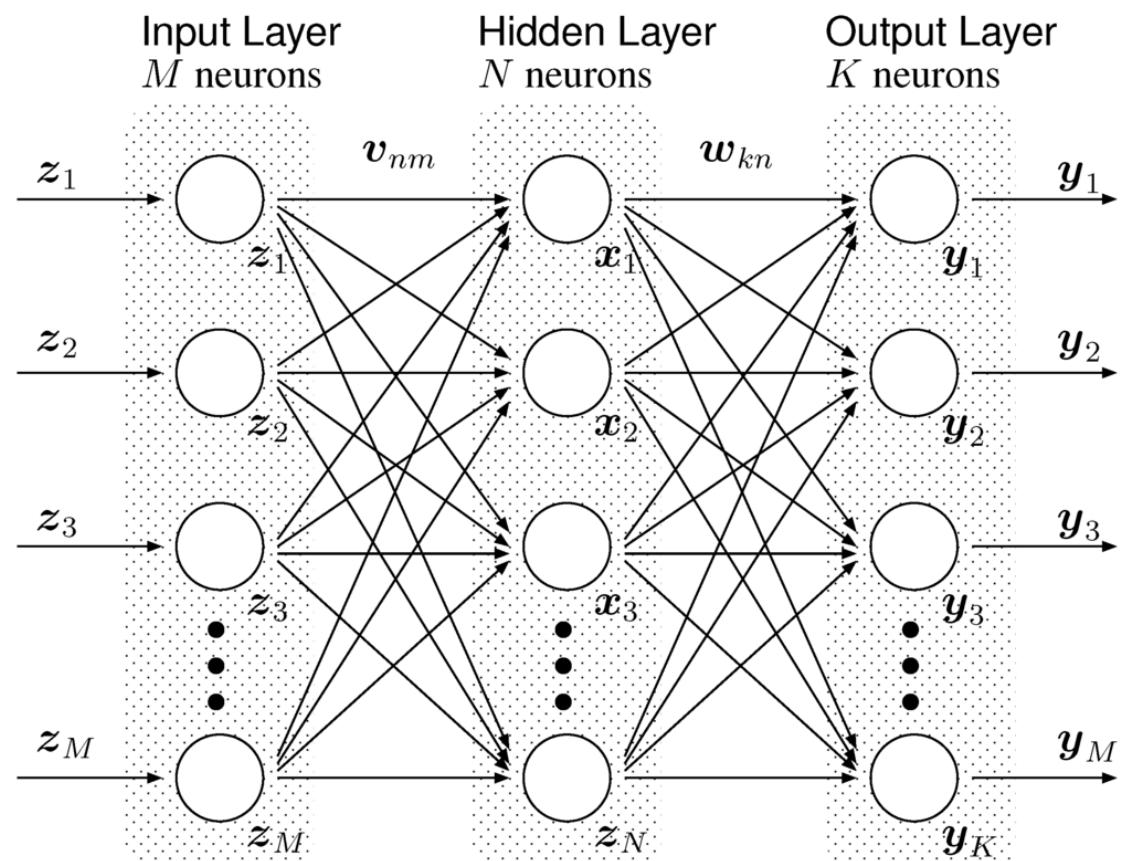
A regra delta padrão essencialmente implementa um gradiente descendente no quadrado da soma do erro para funções de ativação lineares.

Na regra delta generalizada são utilizadas funções de ativação não lineares, diferenciáveis e não decrescentes (sigmóide ou tangente hiperbólica). A função degrau não se enquadra nesse requisito.

A superfície do erro pode não ser tão simples, as redes ficam sujeitas aos problemas de mínimos locais.

Rede MLP típica com três camadas:

- uma camada de entrada
- uma camada escondida
- uma camada de saída



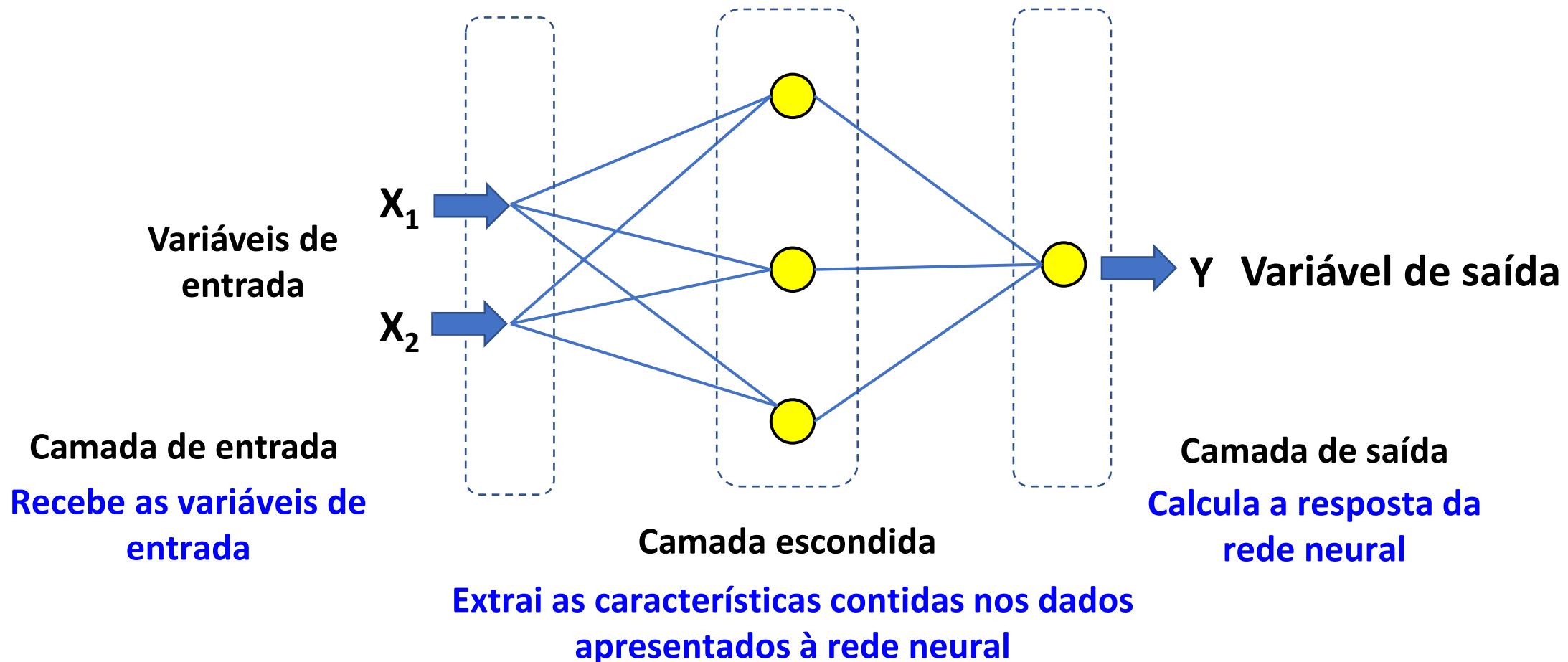
Teorema de Kolmogorov-Nielsen

Dada uma função contínua arbitrária $f: [0,1]^n \rightarrow \mathbb{R}^m$, $f(x)=y$, existe sempre para f uma implementação exata de uma RNA de três camadas, sendo a camada de entrada um vetor de dimensão n , a camada oculta composta por $2n+1$ neurônios e a camada de saída com m neurônios representando as m componentes de y

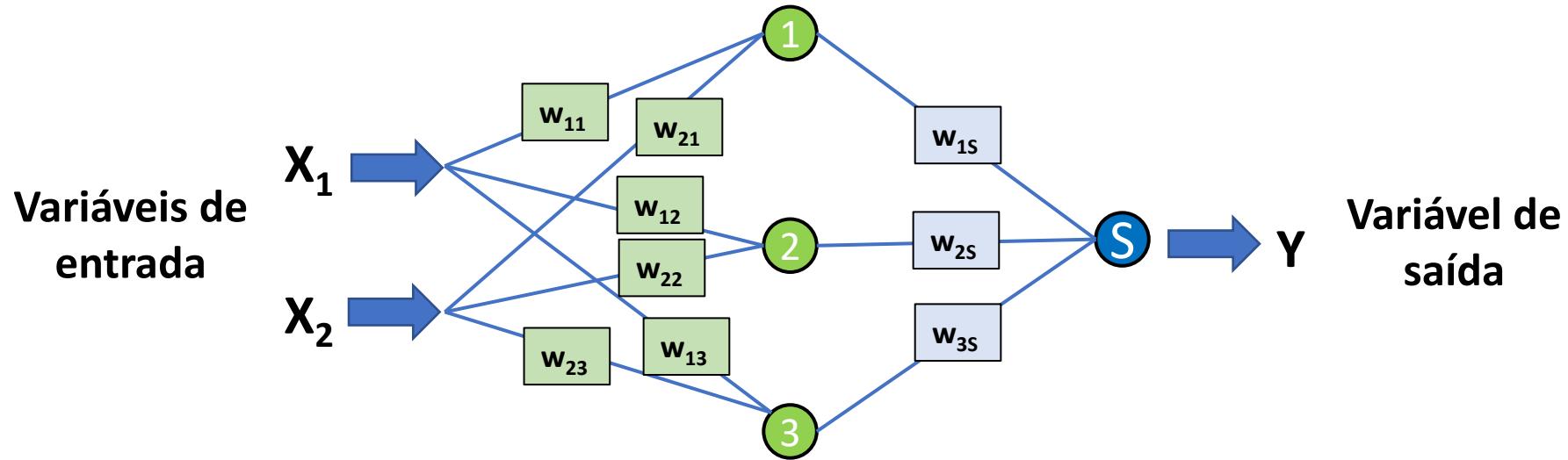
RNA como um aproximador universal de funções

Multilayer Perceptron - MLP

Rede Neural Artificial com três camadas



Multilayer Perceptron - MLP



$$Y = g[w_{1S}f_1(w_{11}X_1 + w_{21}X_2) + w_{2S}f_2(w_{12}X_1 + w_{22}X_2) + w_{3S}f_3(w_{13}X_1 + w_{23}X_2)]$$

$$Y = \mathbf{g} \left(\sum_{k=1}^3 \mathbf{w}_{kS} f_k \left(\sum_{j=1}^2 \mathbf{w}_{jk} x_j \right) \right)$$

Uma RNA é uma regressão não linear



Aprendizado supervisionado – Retropropagação do erro

- Cálculo dos pesos sinápticos de uma rede neural artificial por meio de um processo iterativo.
- A rede aprende com exemplos de pares entrada/saída desejada.
- Os pesos sinápticos são inicializados de forma aleatória
- Para cada entrada (estímulo) a rede neural gera uma resposta. Quando a resposta não coincide com a saída desejada existe um erro que necessita ser corrigido.
- Retropropagação do erro: o erro na resposta da rede é usado para corrigir os pesos sinápticos em todos os neurônios da rede neural.
- O aprendizado é resultado de apresentação repetitiva de pares entrada/saída.
- O processo de aprendizagem é repetido até que um critério de parada seja satisfeito

Aprendizado supervisionado – Retropropagação do erro

O ajuste dos pesos sinápticos (w) se dá pelo cálculo do gradiente da função custo e pela execução da retropropagação do erro, um processo de otimização em duas fases (*forward e backward*), onde os pesos são definidos de forma a minimizar a soma dos quadrados dos erros.

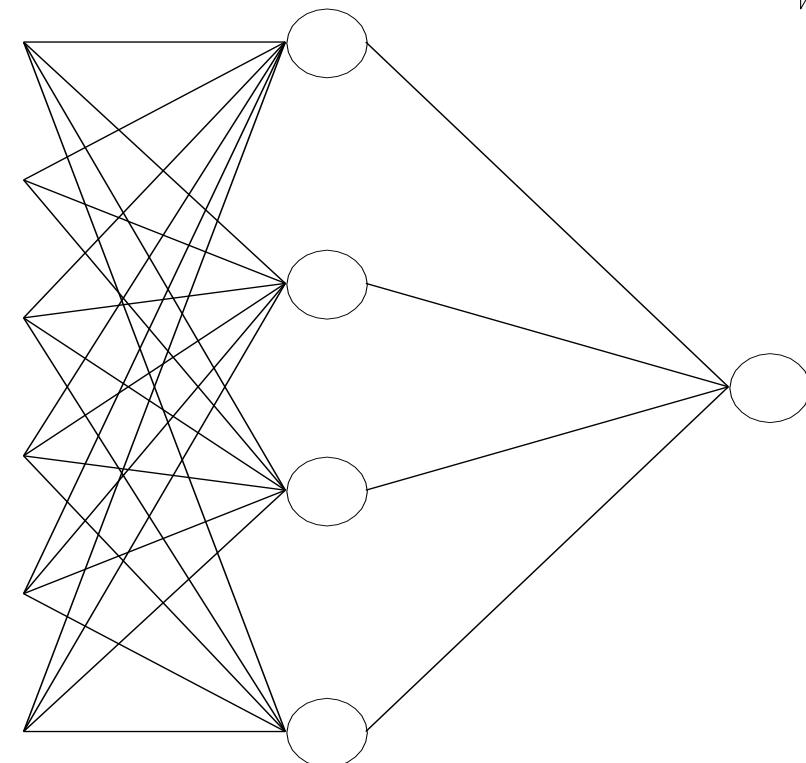
$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}$$
$$\Delta w_{ij} = -\eta \frac{\delta E}{\delta w_{ij}}$$

Taxa de aprendizagem

Gradiente

padrão de entrada

FASE FORWARD: Propaga a entrada através das camadas ocultas até a saída



pesos
ajustados



FASE BACKWARD: Propaga os erros de volta até a camada de entrada

Aprendizado supervisionado – Retropropagação do erro

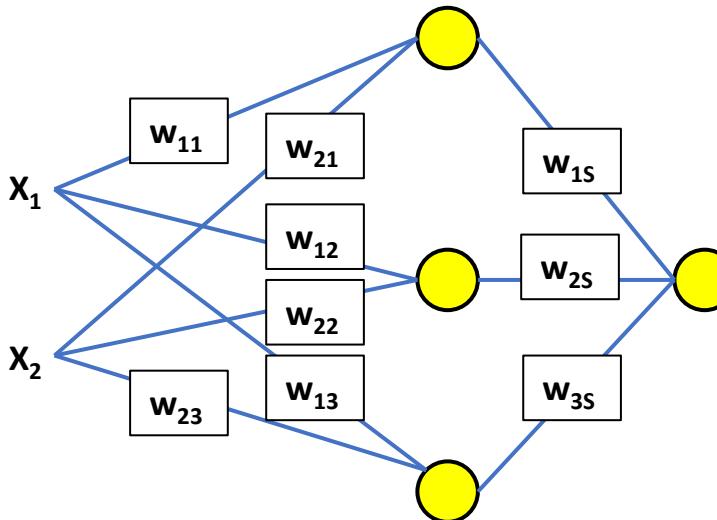
1. Pesos sinápticos
inicializados
aleatoriamente

2. Propaga a entrada X através das camadas ocultas até a saída

padrão de
entrada

FASE FORWARD

resposta
da rede Y



$$Y = g\left(\sum_{k=1}^3 w_{kS} f_k \left(\sum_{j=1}^2 w_{jk} x_j\right)\right)$$

pesos
ajustados

FASE BACKWARD

erro = Y - Saída desejada

3. Retropropagação do erro para
atualização dos pesos

$$w_{ij}(\text{novo}) = w_{ij}(\text{atual}) + \Delta w_{ij}$$

Aprendizado supervisionado – Retropropagação do erro

Atualização dos pesos: local ou por lote?

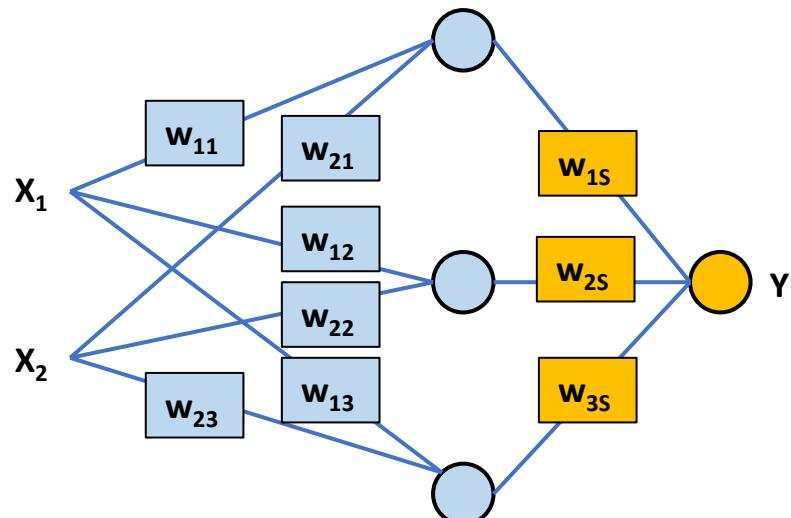
Local ou *online*:

- A atualização dos pesos é feita imediatamente após a apresentação de cada exemplo de treinamento.
- Requer um menor armazenamento para cada conexão, e apresenta menos possibilidade de convergência para um mínimo local.

Lote ou *off-line*:

- A atualização dos pesos só é feita após a apresentação de todos os exemplos de treinamento, ou seja, uma época.
- O ajuste relativo a cada apresentação de um exemplo é acumulado.
- Fornece uma melhor estimativa do vetor gradiente.

Aprendizado supervisionado – Retropropagação do erro



Soma dos quadrados dos erros

Resposta
da rede \mathbf{Y}

$$E = \frac{1}{2} \sum_{k=1}^N \left[g\left(\sum_{k=1}^3 w_{ks} f\left(\sum_{j=1}^2 w_{jk} x_{ij} \right) \right) - y_i^d \right]^2$$

Soma dos
quadradados
dos erros

O treinamento da rede neural consiste em ajustar os pesos sinápticos (W) de forma a minimizar a soma dos quadrados dos erros
(Problema de Otimização)

A retropropagação do erro é uma implementação da regra da cadeia

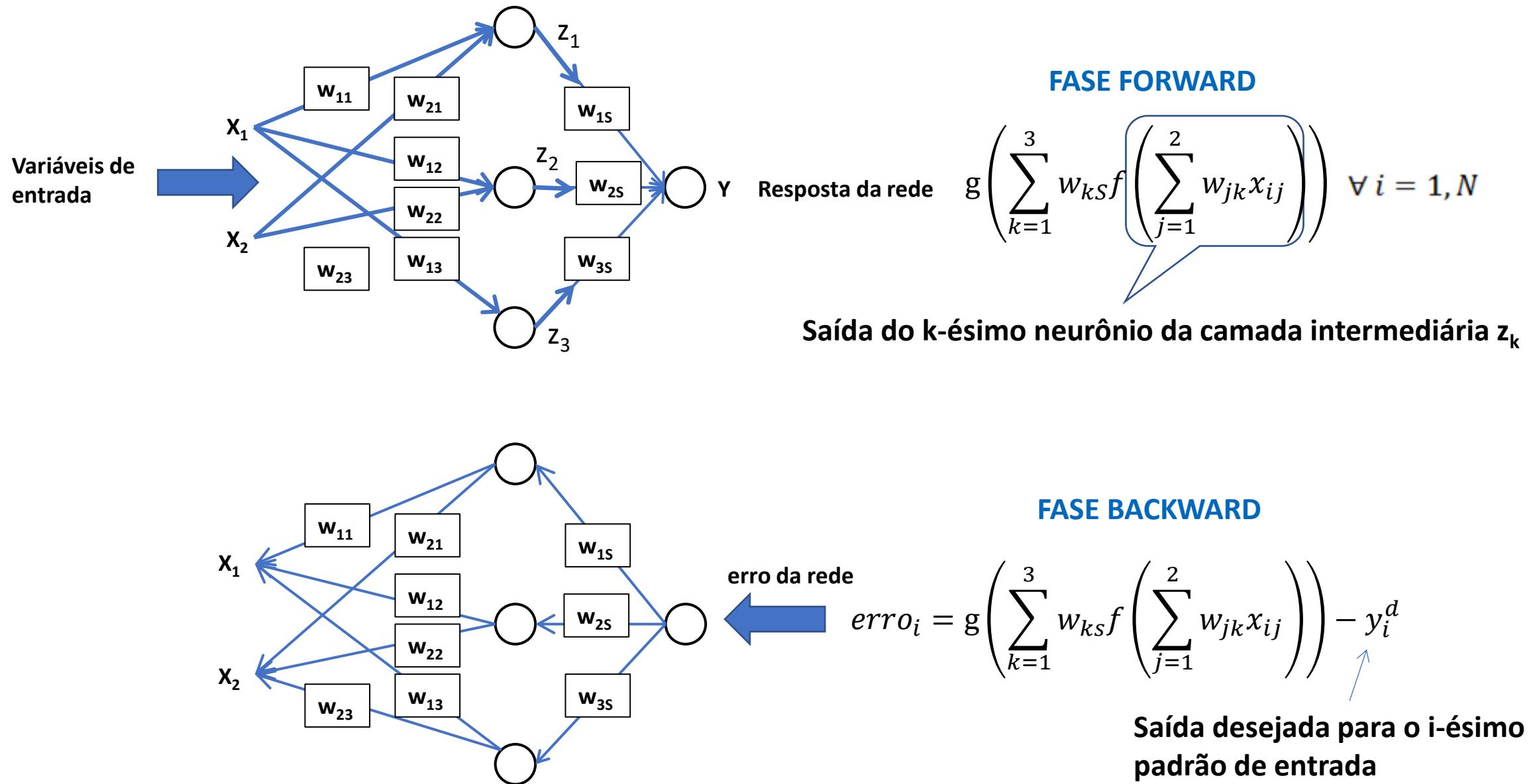
Gradiente no treinamento online

$$\frac{\partial E}{\partial w} = \frac{\partial E}{\partial g} \frac{\partial g}{\partial w} = \left[g\left(\sum_{k=1}^3 w_{ks} f\left(\sum_{j=1}^2 w_{jk} x_{ij} \right) \right) - y_i^d \right] g' = \text{erro}_i \times g'$$

erro

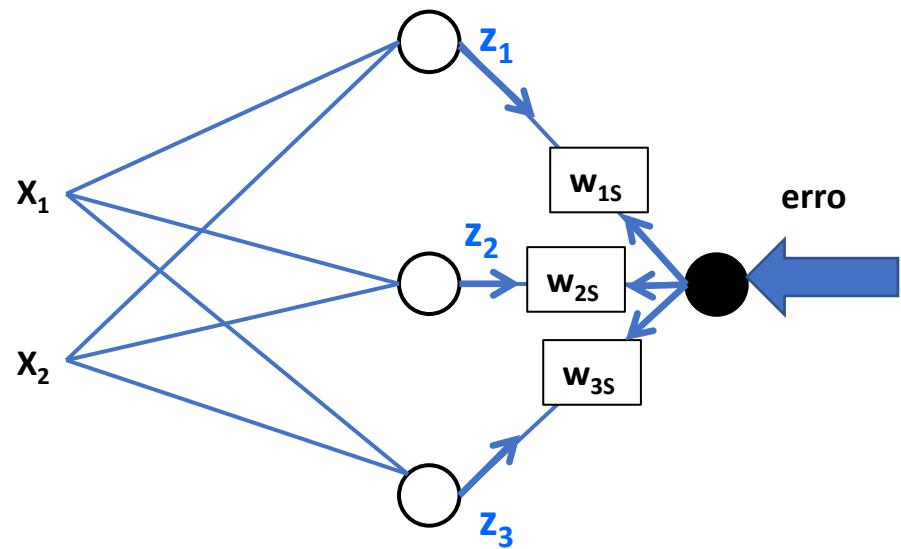
Saída desejada para o i -ésimo padrão de entrada

Aprendizado supervisionado – Retropropagação do erro



Aprendizado supervisionado – Retropropagação do erro

FASE BACKWARD: Atualização dos pesos das sinapses que chegam no neurônio da camada de saída



$$\Delta w_{1s} = -\eta \cdot \mathbf{z}_1 \cdot erro_i \cdot g'\left(\sum_{k=1}^3 w_{ks} \cdot z_k\right)$$

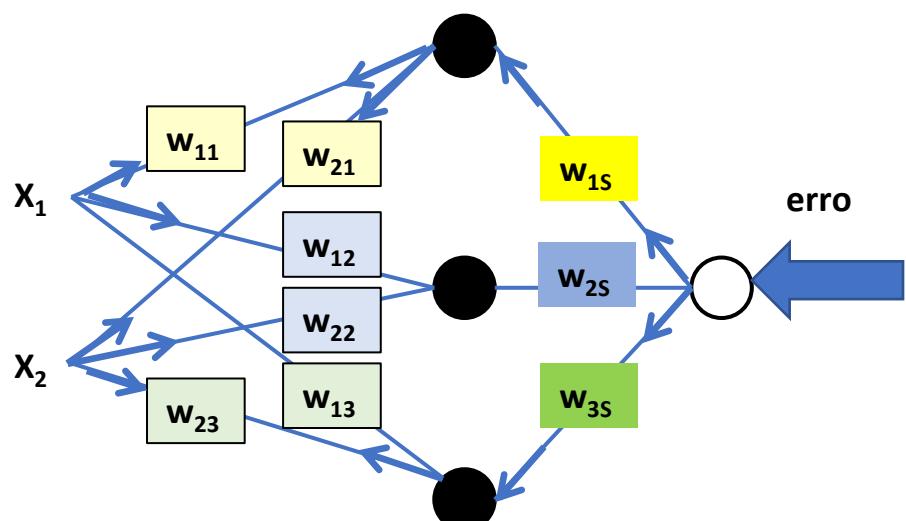
$$\Delta w_{2s} = -\eta \cdot \mathbf{z}_2 \cdot erro_i \cdot g'\left(\sum_{k=1}^3 w_{ks} \cdot z_k\right)$$

$$\Delta w_{3s} = -\eta \cdot \mathbf{z}_3 \cdot erro_i \cdot g'\left(\sum_{k=1}^3 w_{ks} \cdot z_k\right)$$

$$w_{ks}(\text{novo}) = w_{ks}(\text{atual}) + \Delta w_{ks} \quad \forall k=1,3$$

Aprendizado supervisionado – Retropropagação do erro

FASE BACKWARD: Atualização dos pesos das sinapses que chegam nos neurônios da camada escondida



$$w_{ij}(\text{novo}) = w_{ij}(\text{atual}) + \Delta w_{ij} \quad \forall i=1,3; j=1,2$$

$$\Delta w_{11} = -\eta \cdot x_{i1} \cdot f'\left(\sum_{j=1}^2 w_{j1} \cdot x_{ij}\right) \cdot w_{1S} \cdot erro_i \cdot g'\left(\sum_{k=1}^3 w_{ks} \cdot z_k\right)$$
$$\Delta w_{21} = -\eta \cdot x_{i2} \cdot f'\left(\sum_{j=1}^2 w_{j1} \cdot x_{ij}\right) \cdot w_{1S} \cdot erro_i \cdot g'\left(\sum_{k=1}^3 w_{ks} \cdot z_k\right)$$

$$\Delta w_{12} = -\eta \cdot x_{i1} \cdot f'\left(\sum_{j=1}^2 w_{j2} \cdot x_{ij}\right) \cdot w_{2S} \cdot erro_i \cdot g'\left(\sum_{k=1}^3 w_{ks} \cdot z_k\right)$$
$$\Delta w_{22} = -\eta \cdot x_{i2} \cdot f'\left(\sum_{j=1}^2 w_{j2} \cdot x_{ij}\right) \cdot w_{2S} \cdot erro_i \cdot g'\left(\sum_{k=1}^3 w_{ks} \cdot z_k\right)$$

$$\Delta w_{13} = -\eta \cdot x_{i1} \cdot f'\left(\sum_{j=1}^2 w_{j3} \cdot x_{ij}\right) \cdot w_{3S} \cdot erro_i \cdot g'\left(\sum_{k=1}^3 w_{ks} \cdot z_k\right)$$
$$\Delta w_{23} = -\eta \cdot x_{i2} \cdot f'\left(\sum_{j=1}^2 w_{j3} \cdot x_{ij}\right) \cdot w_{3S} \cdot erro_i \cdot g'\left(\sum_{k=1}^3 w_{ks} \cdot z_k\right)$$

Aprendizado supervisionado – Retropropagação do erro

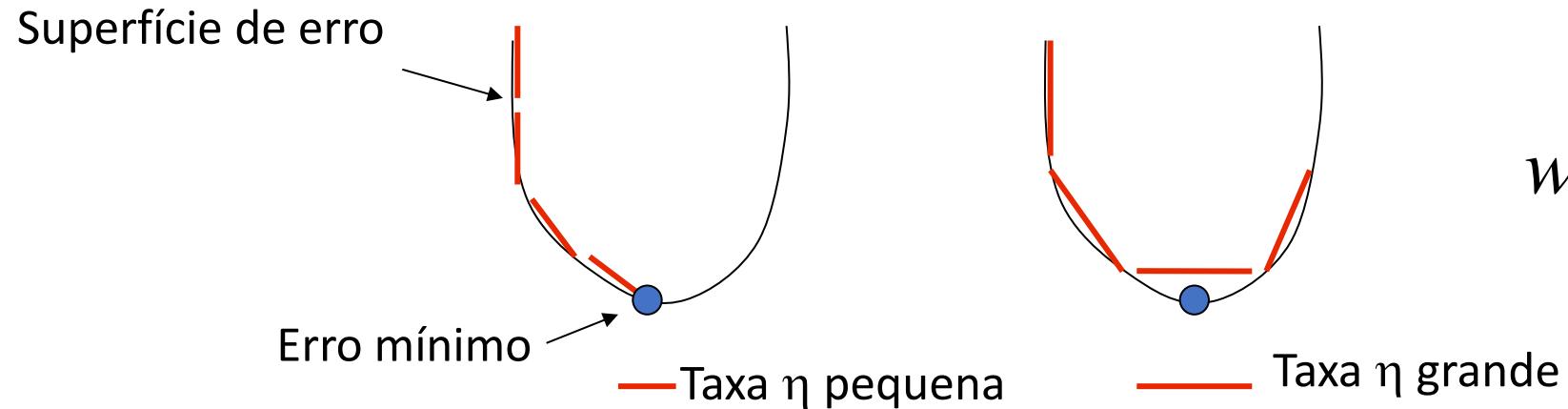
O processo de minimização da função erro não apresenta convergência garantida para o mínimo global.

Critérios de parada:

- 1) Quando o total de iterações atinge o **máximo número de iterações** pré-definido. Este critério de parada não leva em conta o estado do processo iterativo.
- 2) Quando a norma euclidiana da estimativa do vetor gradiente atinge um valor suficientemente pequeno. $\|\nabla E(w)\| \leq \varepsilon$
- 3) Quando a variação do erro quadrático médio de uma época para outra atingir um valor suficientemente pequeno. $\Delta E(w) \leq \varepsilon$
- 4) Quando o erro quadrático médio atingir um valor suficientemente pequeno. $E(w) \leq \varepsilon$

Aprendizado supervisionado – Retropropagação do erro

Taxa de aprendizagem variável



$$w_{ij}^{novo} = w_{ij}^{velho} - \eta \frac{\partial E}{\partial w_{ij}}$$

A taxa de aprendizagem $0 < \eta < 1$.

Valores pequenos para η implicam em longo tempo para convergência

Valores grandes implicam na oscilação do algoritmo entre soluções na vizinhança da solução ótimo, sem no entanto convergir.

Uma solução consiste em modificar a taxa de aprendizagem ao longo do treinamento, iniciando com um valor grande, mas gradualmente reduzido. $\eta^{t+1} = \eta^1 \left(\frac{1}{t} \right)$

Aprendizado supervisionado – Retropropagação do erro

Uma maneira de aumentar a taxa de aprendizado sem levar à oscilação é incluir o momento, um termo adicional expresso pela soma ponderada das direções de buscas anteriores.

Introduz um efeito de inércia no ajuste dos pesos.

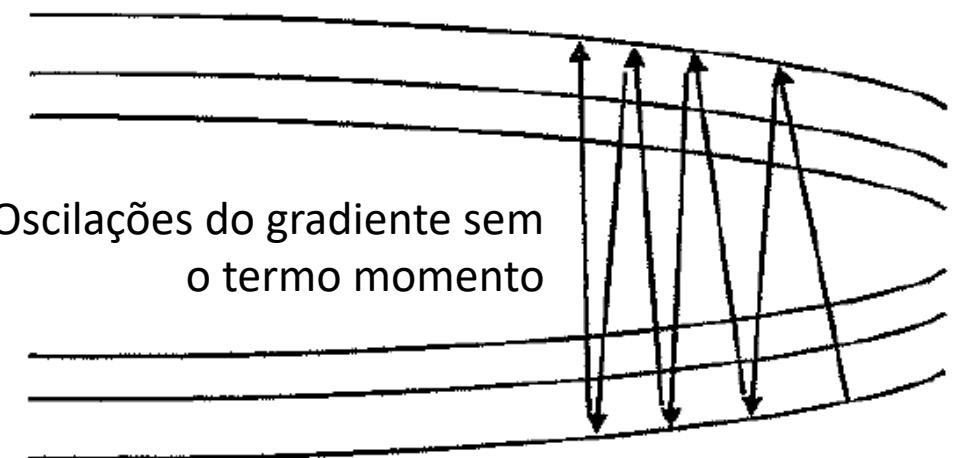
$$\Delta w_{ij}^{atual} = -\eta \frac{\partial E}{\partial w_{ij}} + \alpha \Delta w_{ij}^{anterior}$$

Os ajustes mais recentes nos pesos exercem influência no ajuste atual.

O momento ajuda a amortecer as oscilações ao redor da solução ótima.

Ajuda a aumentar a taxa de aprendizagem no inicio do aprendizado

É uma estratégia para evitar mínimos locais da função erro



Oscilações do gradiente sem
o termo momento

Masters (1993)

Aplicação: Modelagem da curva de potência

Energy Systems (2022) 13:983–1010
<https://doi.org/10.1007/s12667-021-00449-5>

ORIGINAL PAPER

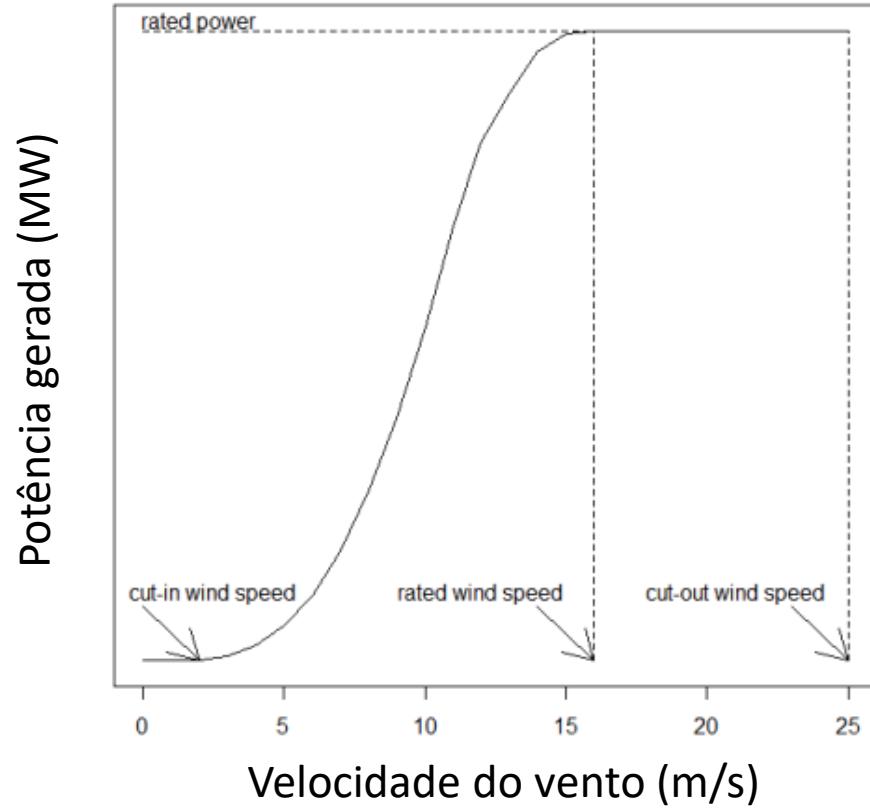


Revisiting the modeling of wind turbine power curves using neural networks and fuzzy models: an application-oriented evaluation

Guilherme A. Barreto¹ · Igor S. Brasil² · Luis Gustavo M. Souza²

A modelagem da curva de potência da turbina eólica a partir de dados medidos é essencial para prever a geração de energia dos parques eólicos.

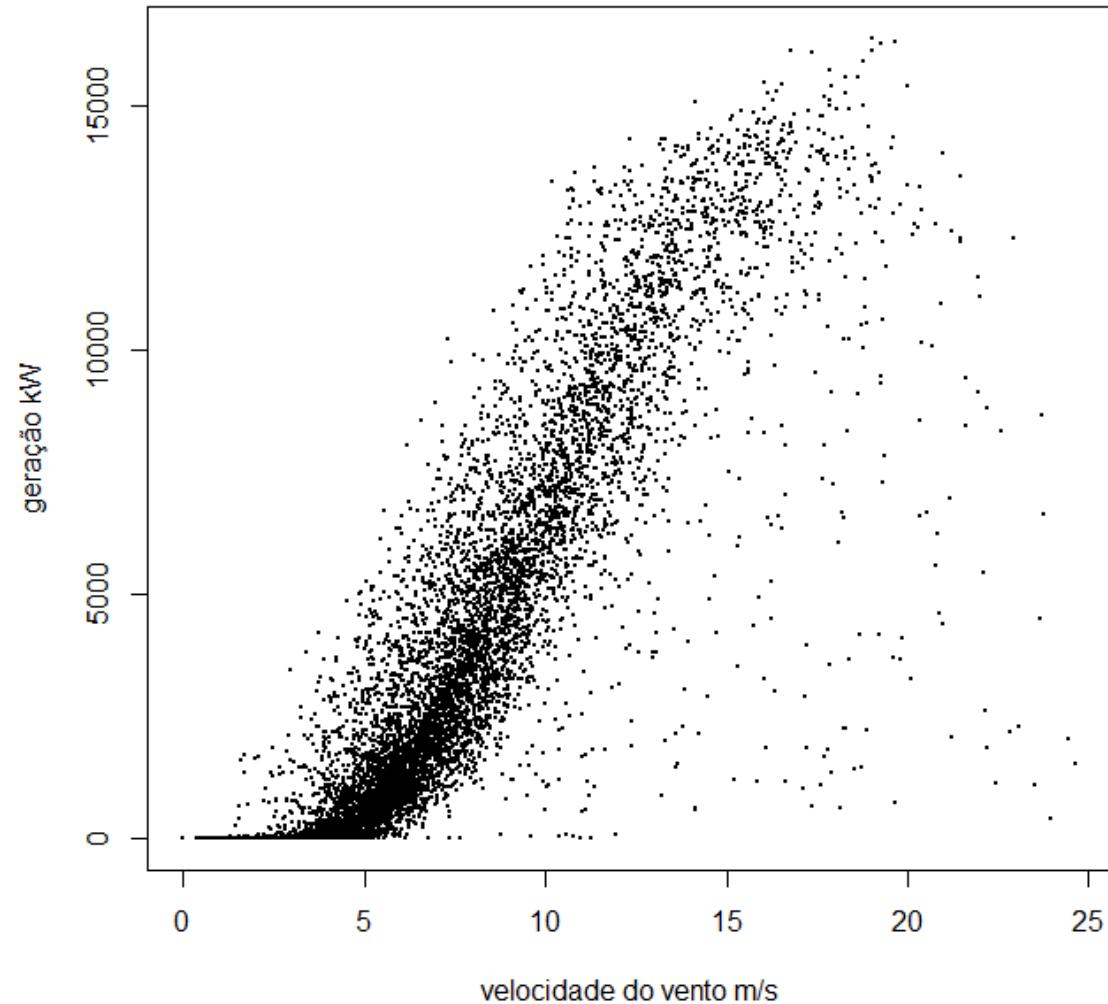
A regressão polinomial é comumente a primeira escolha para esse propósito, mas existem outras alternativas mais poderosas baseadas em redes neurais e algoritmos fuzzy, por exemplo.



Curva de potência de um parque eólico

Parque eólico na Galícia – Espanha

- Capacidade instalada 17,56 MW
- 24 aerogeradores
- 5 tecnologias e 9 fabricantes diferentes
- Terreno complexo
- Velocidade média anual 6,41 m/s
- Produção anual da ordem de 33MWh



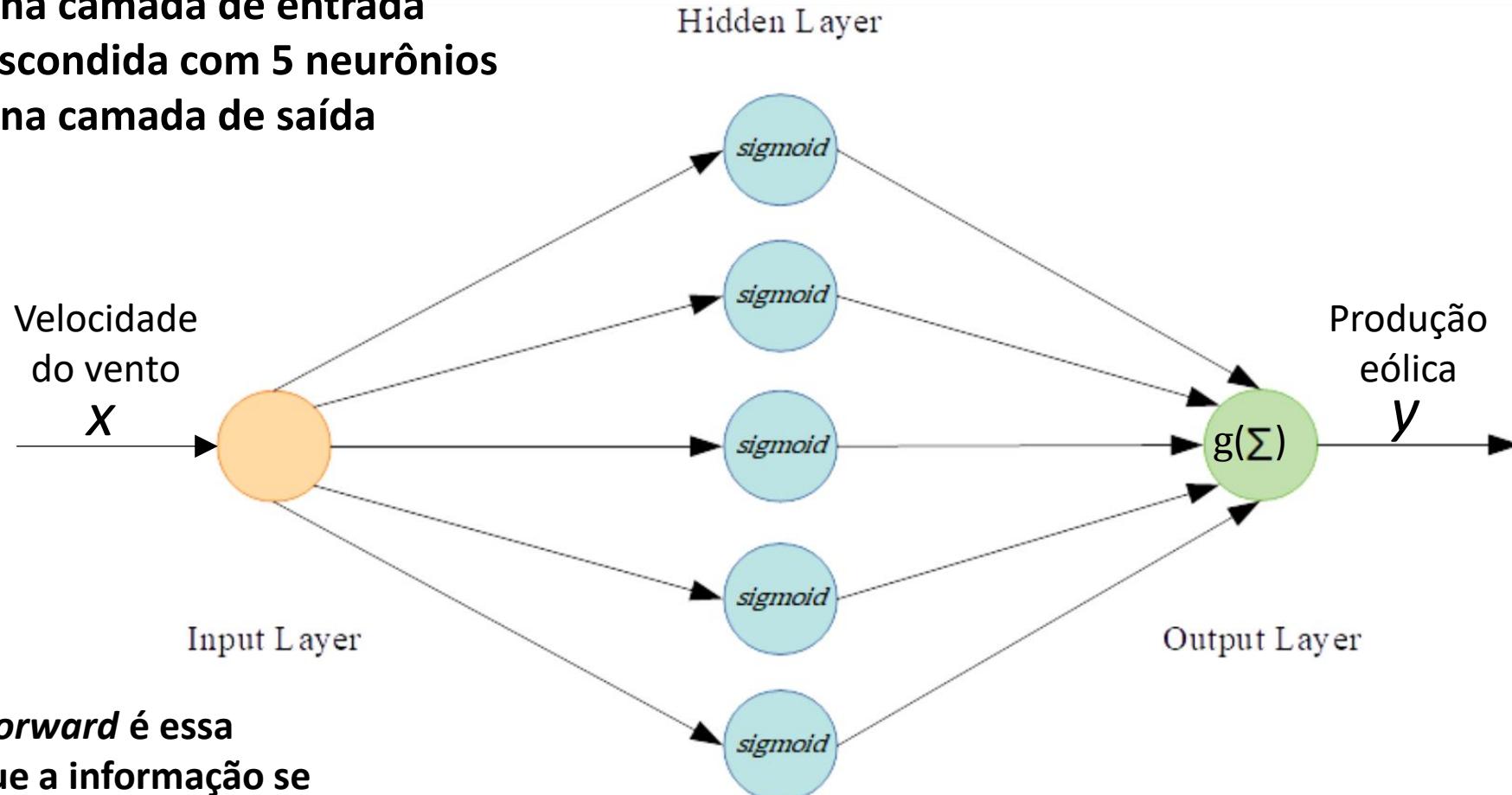
Queremos encontrar uma curva que descreva o valor esperado da produção eólica em função da velocidade do vento

$$E(\text{produção}|\text{velocidade do vento}) = f(\text{velocidade do vento})$$

RNA com uma camada escondida

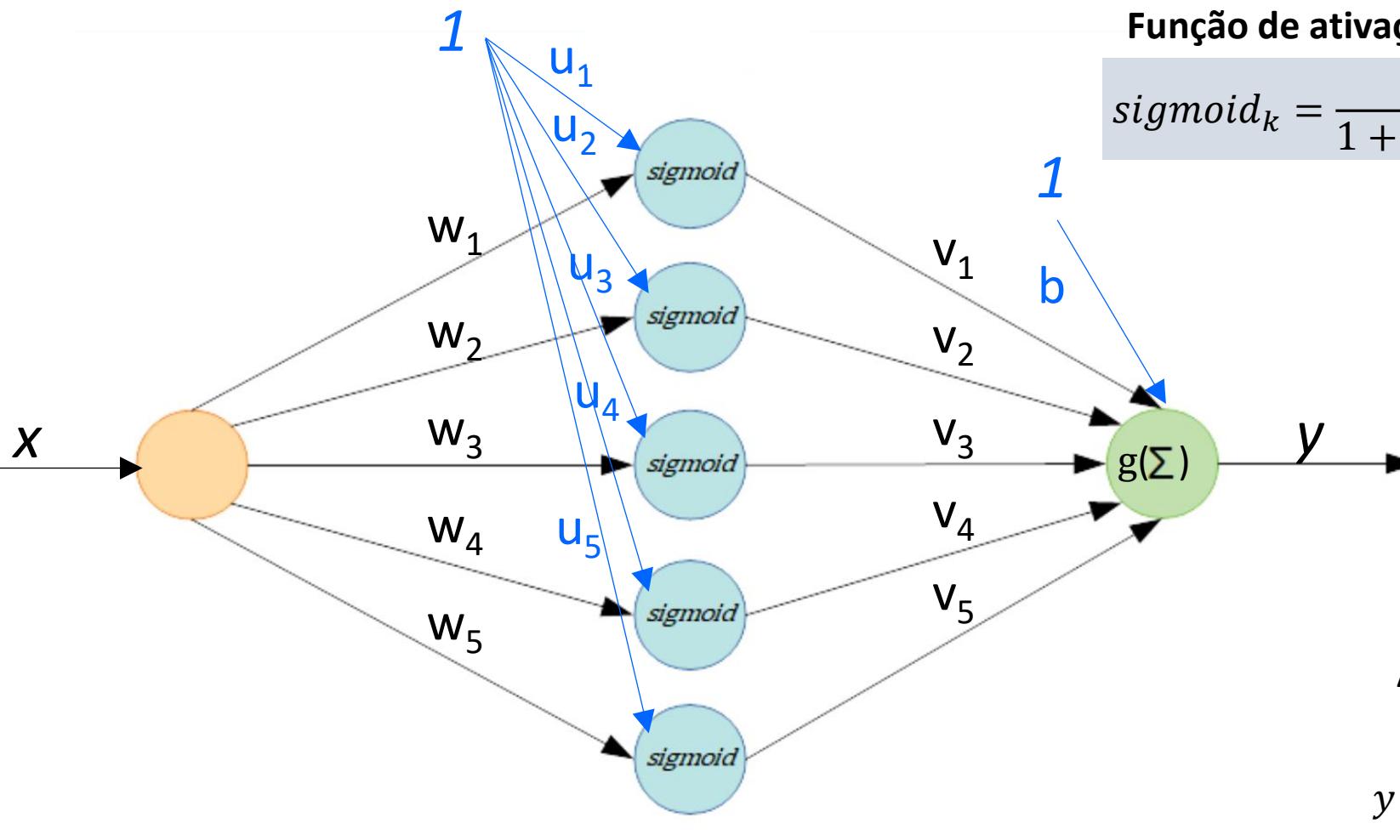
Multi-Layer Perceptron (MLP)

- Rede *feedforward* com 3 camadas
- 1 neurônio na camada de entrada
- 1 camada escondida com 5 neurônios
- 1 neurônio na camada de saída



uma rede *feedforward* é essa estrutura em que a informação se propaga "para frente" sem loops

Funções de ativação dos neurônios



Função de ativação dos neurônios da camada escondida

$$\text{sigmoid}_k = \frac{1}{1 + \exp(-\text{net}_k)} = \frac{1}{1 + \exp[-(u_k + w_k x)]}$$

Saída da RNA

$$y = g(\text{NET}) = g\left(b + \sum_{k=1}^5 v_k \text{sigmoid}_k\right)$$

A RNA é uma função que mapeia x em y

$$y = g\left(b + \sum_{k=1}^5 \frac{v_k}{1 + \exp[-(u_k + w_k x)]}\right)$$

b, u, v, w são parâmetros a serem ajustados, i.e., aprendidos

Treinamento da RNA

Soma dos quadrados dos erros (função custo) nos n exemplos de treinamento

$$E = \frac{1}{2} \sum_{i=1}^n (\mathbf{y}_i - \mathbf{y}_i^d)^2 \rightarrow E = \frac{1}{2} \sum_{i=1}^n (\mathbf{g}(\text{NET}_i) - \mathbf{y}_i^d)^2 \rightarrow \frac{1}{2} \sum_{i=1}^n \left(\mathbf{g} \left(\mathbf{b} + \sum_{k=1}^5 v_k \text{sigmoid}_{i,k} \right) - \mathbf{y}_i^d \right)^2$$

↑ Saída da RNA ↑ Saída desejada
 Erro do i-ésimo exemplo de treinamento

função de u_k e w_k

$$E = \frac{1}{2} \sum_{i=1}^n \left(\mathbf{g} \left(\mathbf{b} + \sum_{k=1}^5 \frac{v_k}{1 + \exp[-(u_k + w_k x_i)]} \right) - \mathbf{y}_i^d \right)^2$$

Objetivo: encontrar valores para $\mathbf{b}, \mathbf{u}, \mathbf{v}, \mathbf{w}$ que minimizem a função custo

$$\underset{\mathbf{b}, \mathbf{u}, \mathbf{v}, \mathbf{w}}{\text{Min}} \quad \frac{1}{2} \sum_{i=1}^n \left(\mathbf{g} \left(\mathbf{b} + \sum_{k=1}^5 \frac{v_k}{1 + \exp[-(u_k + w_k x_i)]} \right) - \mathbf{y}_i^d \right)^2$$

Otimização pelo método do gradiente descendente

Vamos precisar das derivadas parciais da função custo

$$w_{ij}^{novo} = w_{ij}^{velho} - \eta \frac{\partial E}{\partial w_{ij}}$$

$$E = \frac{1}{2} \sum_{i=1}^n (\mathbf{y}_i - y_i^d)^2 \rightarrow E = \frac{1}{2} \sum_{i=1}^n (\mathbf{g}(NET_i) - y_i^d)^2 \rightarrow \frac{1}{2} \sum_{i=1}^n \left(\mathbf{g}\left(\mathbf{b} + \sum_{k=1}^5 v_k sigmoid_{i,k}\right) - y_i^d \right)^2$$

$$b^{novo} = b^{velho} - \eta \left. \frac{\partial E}{\partial b} \right|_{b^{velho}}$$

$$\frac{\partial E}{\partial b} = \frac{\partial E}{\partial g} \frac{\partial g}{\partial NET} \frac{\partial NET}{\partial b} = \sum_{i=1}^n (g(NET_i) - y_i^d) \times g' \times 1 = \sum_{i=1}^n erro_i \times g'$$

$$v_k^{novo} = v_k^{velho} - \eta \left. \frac{\partial E}{\partial v_k} \right|_{v_k^{velho}}$$

$$\frac{\partial E}{\partial v_k} = \frac{\partial E}{\partial g} \frac{\partial g}{\partial NET} \frac{\partial NET}{\partial v_k} = \sum_{i=1}^n (NET_i - y_i^d) \times g' \times sigmoid_{i,k} = \sum_{i=1}^n erro_i \times g' \times sigmoid_{i,k}$$

A retropropagação do erro é uma implementação da regra da cadeia

Otimização pelo método do gradiente descendente

Vamos precisar das derivadas parciais da função custo

$$E = \frac{1}{2} \sum_{i=1}^n (\mathbf{y}_i - y_i^d)^2 \rightarrow E = \frac{1}{2} \sum_{i=1}^n (g(NET_i) - y_i^d)^2 \rightarrow \frac{1}{2} \sum_{i=1}^n \left(g\left(b + \sum_{k=1}^5 v_k \text{sigmoid}_{i,k}\right) - y_i^d \right)^2$$

$$u_k^{novo} = u_k^{velho} - \eta \left. \frac{\partial E}{\partial u_k} \right|_{u_k^{velho}}$$

$$\begin{aligned} \frac{\partial E}{\partial u_k} &= \frac{\partial E}{\partial g} \frac{\partial g}{\partial NET} \frac{\partial NET}{\partial \text{sigmoide}} \frac{\partial \text{sigmoide}}{\partial u_k} = \sum_{i=1}^n (g(NET_i) - y_i^d) \times g' \times v_k \times \text{sigmoid}_{i,k} (1 - \text{sigmoid}_{i,k}) \\ &= \sum_{i=1}^n \text{erro}_i \times g' \times v_k \times \text{sigmoid}_{i,k} (1 - \text{sigmoid}_{i,k}) \end{aligned}$$

$$w_k^{novo} = w_k^{velho} - \eta \left. \frac{\partial E}{\partial w_k} \right|_{w_k^{velho}}$$

$$\begin{aligned} \frac{\partial E}{\partial w_k} &= \frac{\partial E}{\partial g} \frac{\partial g}{\partial NET} \frac{\partial NET}{\partial \text{sigmoide}} \frac{\partial \text{sigmoide}}{\partial w_k} = \sum_{i=1}^n (g(NET_i) - y_i^d) \times g' \times v_k \times \text{sigmoid}_{i,k} (1 - \text{sigmoid}_{i,k}) x_i \\ &= \sum_{i=1}^n \text{erro}_i \times g' \times v_k \times \text{sigmoid}_{i,k} (1 - \text{sigmoid}_{i,k}) x_i \end{aligned}$$

A retropropagação do erro é uma implementação da regra da cadeia

$$\frac{1}{1 + \exp[-(u_k + w_k x_i)]}$$

Incrementos aos pesos

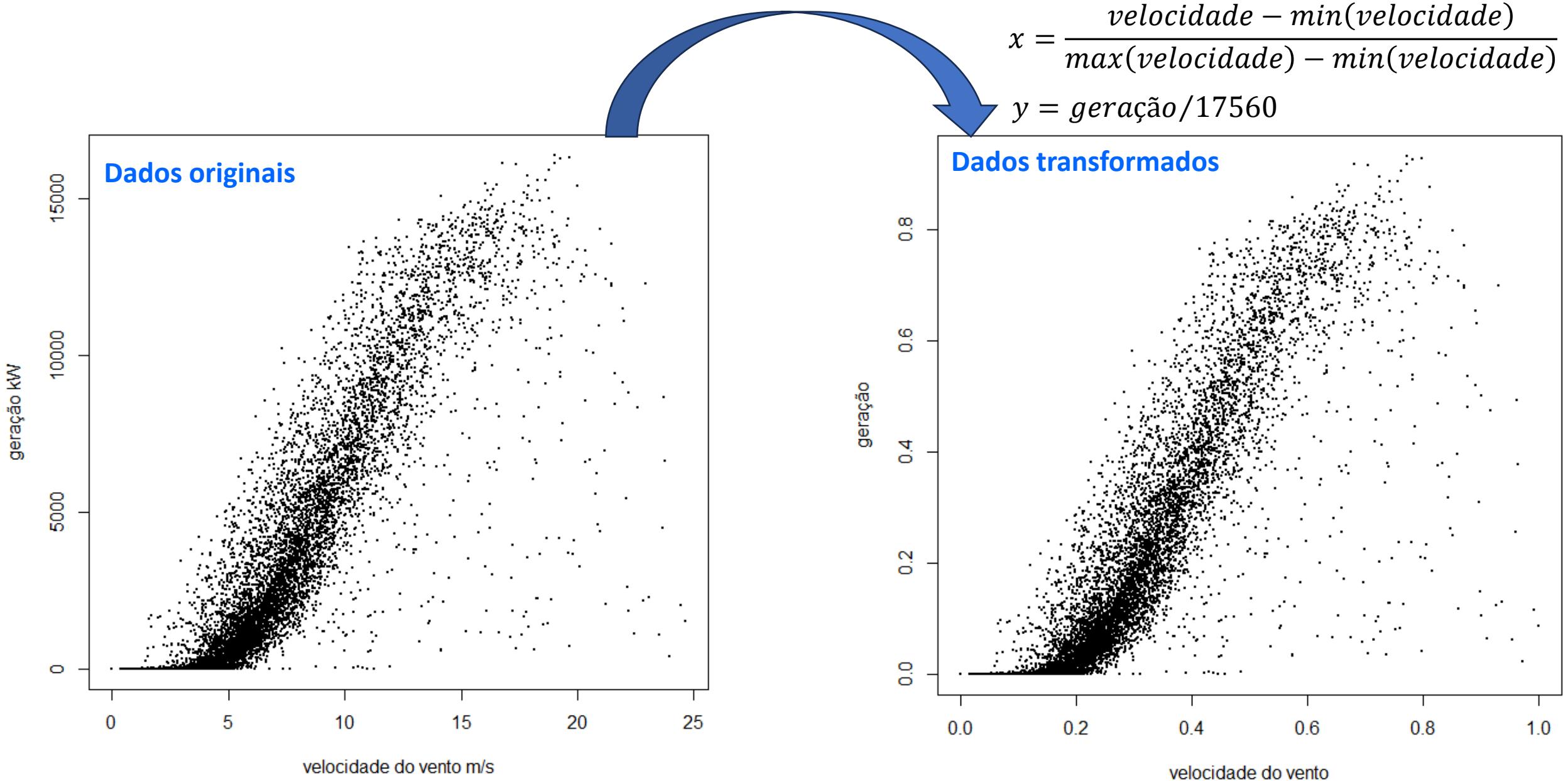
$$\frac{\partial E}{\partial b} = \sum_{i=1}^n \text{erro}_i \times g' \rightarrow \Delta b = \text{erro}_i \times g'$$

$$\frac{\partial E}{\partial v_k} = \sum_{i=1}^n \text{erro}_i \times g' \times \text{sigmoid}_{i,k} \rightarrow \Delta v_k = \text{erro}_i \times g' \times \text{sigmoid}_{i,k}$$

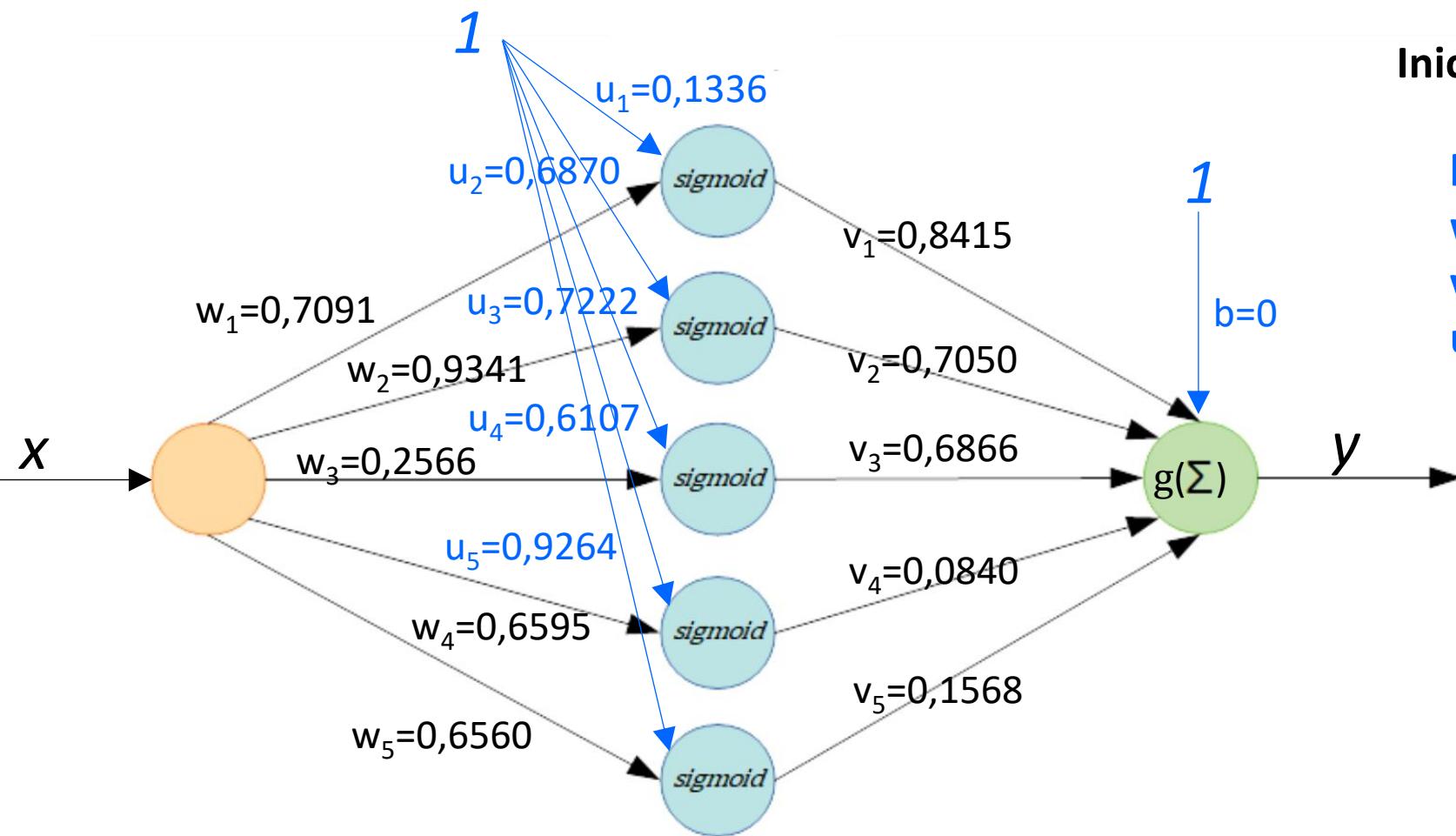
$$\frac{\partial E}{\partial u_k} = \sum_{i=1}^n \text{erro}_i \times g' \times v_k \times \text{sigmoid}_{i,k} (1 - \text{sigmoid}_{i,k}) \rightarrow \Delta u_k = \text{erro}_i \times g' \times v_k \times \text{sigmoid}_{i,k} (1 - \text{sigmoid}_{i,k})$$

$$\frac{\partial E}{\partial w_k} = \sum_{i=1}^n \text{erro}_i \times g' \times v_k \times \text{sigmoid}_{i,k} (1 - \text{sigmoid}_{i,k}) x_i \rightarrow \Delta w_k = \text{erro}_i \times g' \times v_k \times \text{sigmoid}_{i,k} (1 - \text{sigmoid}_{i,k}) x_i$$

Conjunto de treinamento



Inicialização do treinamento

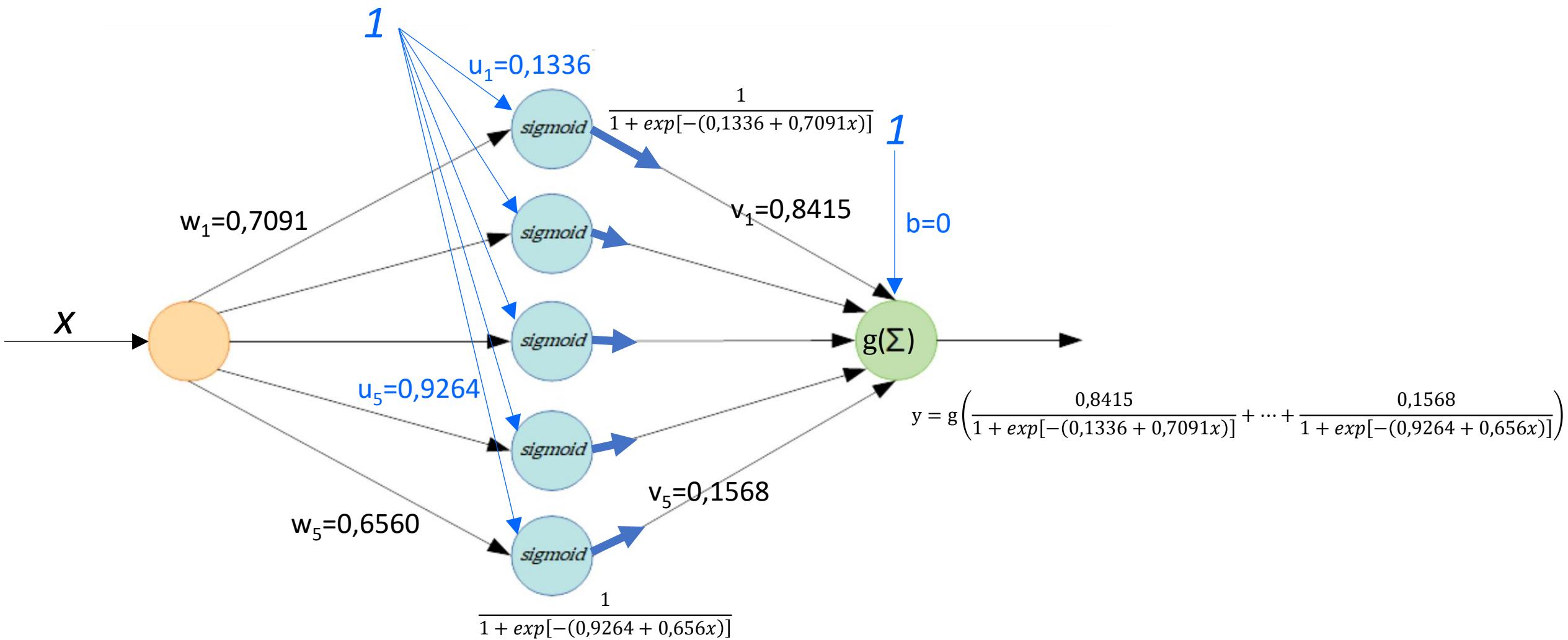


Inicialização aleatória dos pesos

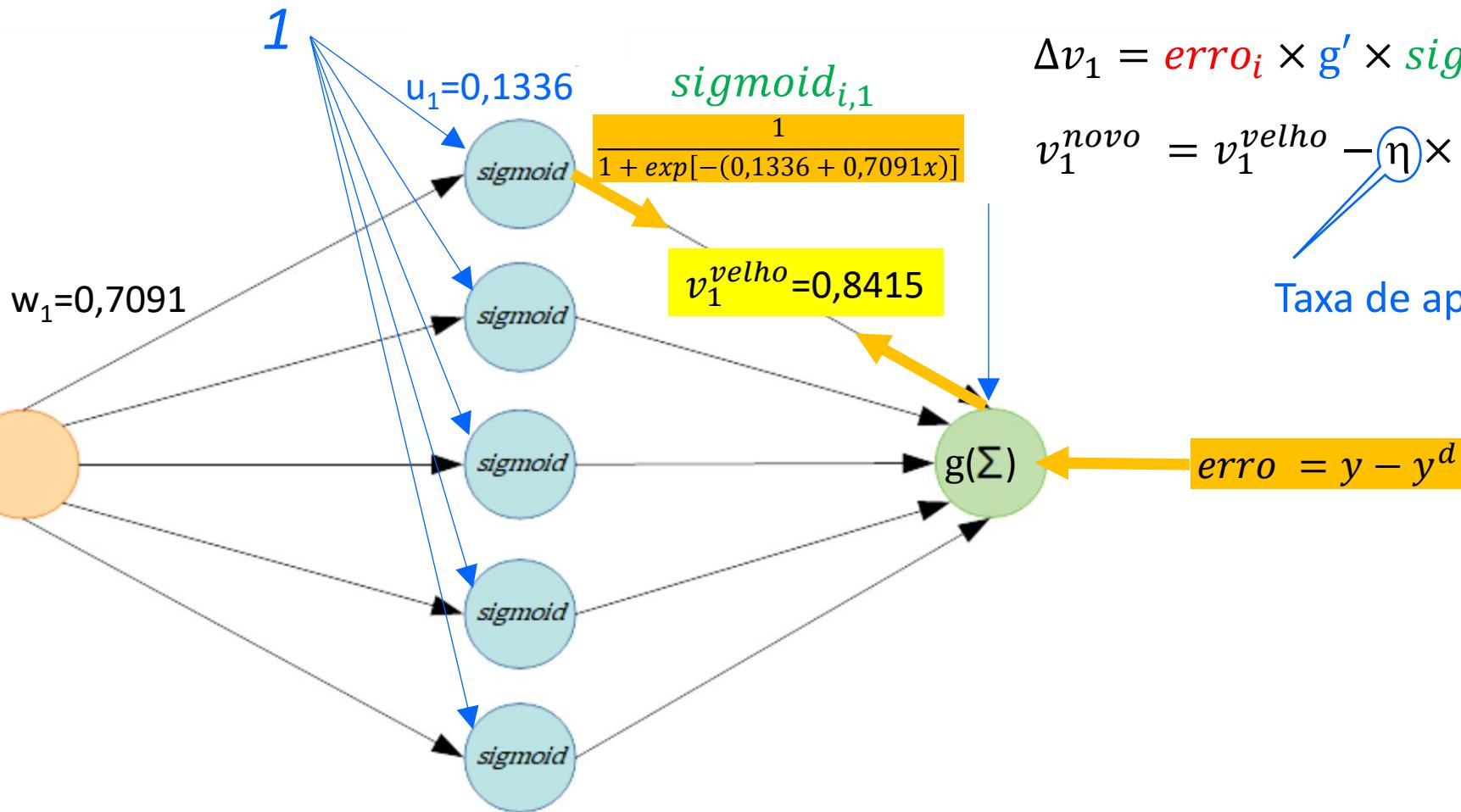
$b <- 0$
 $w <- \text{runif}(5,0,1)$
 $v <- \text{runif}(5,0,1)$
 $u <- \text{runif}(5,0,1)$



Fase Forward



Fase Backward



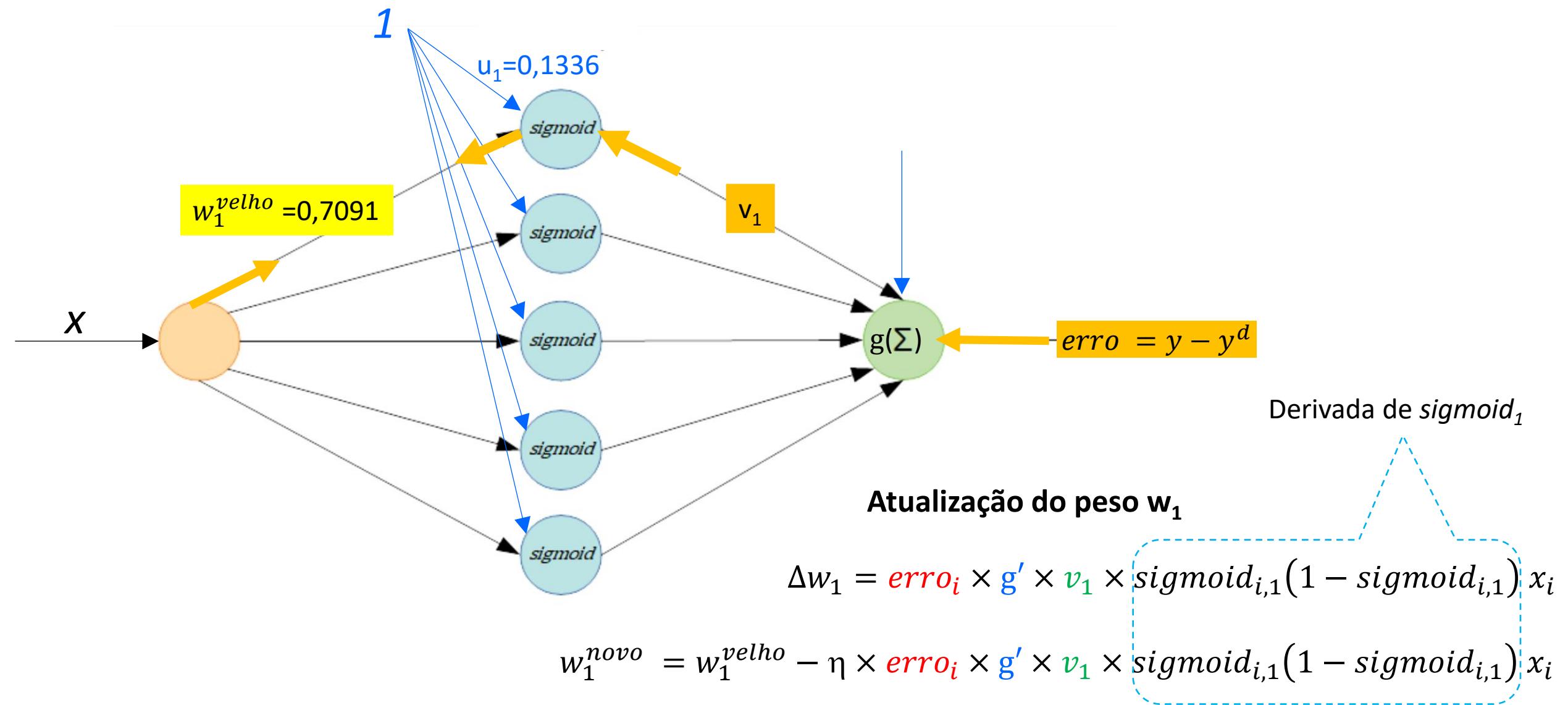
Atualização do peso v_1

$$\Delta v_1 = \text{erro}_i \times g' \times \text{sigmoid}_{i,1}$$

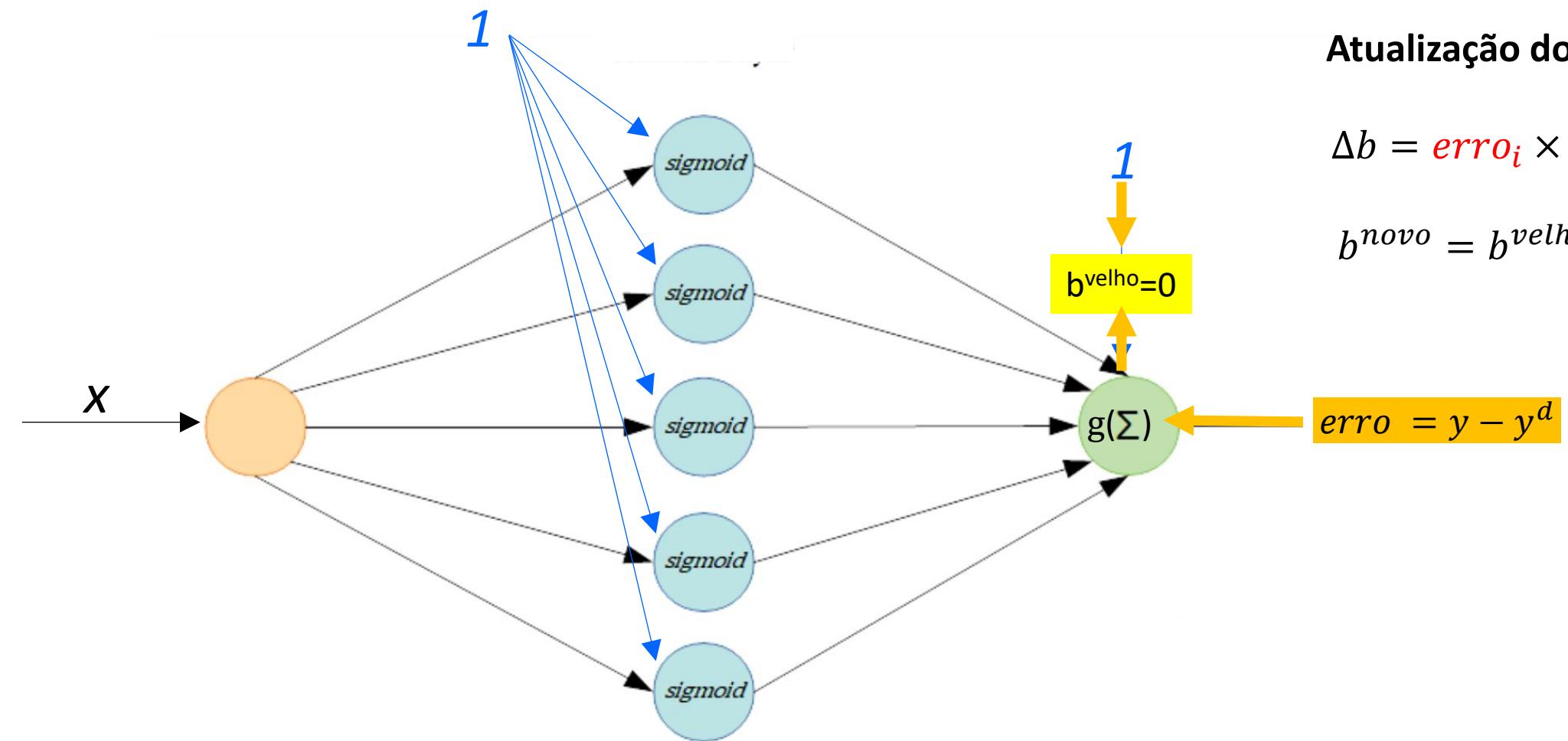
$$v_1^{\text{novo}} = v_1^{\text{velho}} - \eta \times \text{erro}_i \times g' \times \text{sigmoid}_{i,1}$$

Taxa de aprendizagem

Fase Backward



Fase Backward



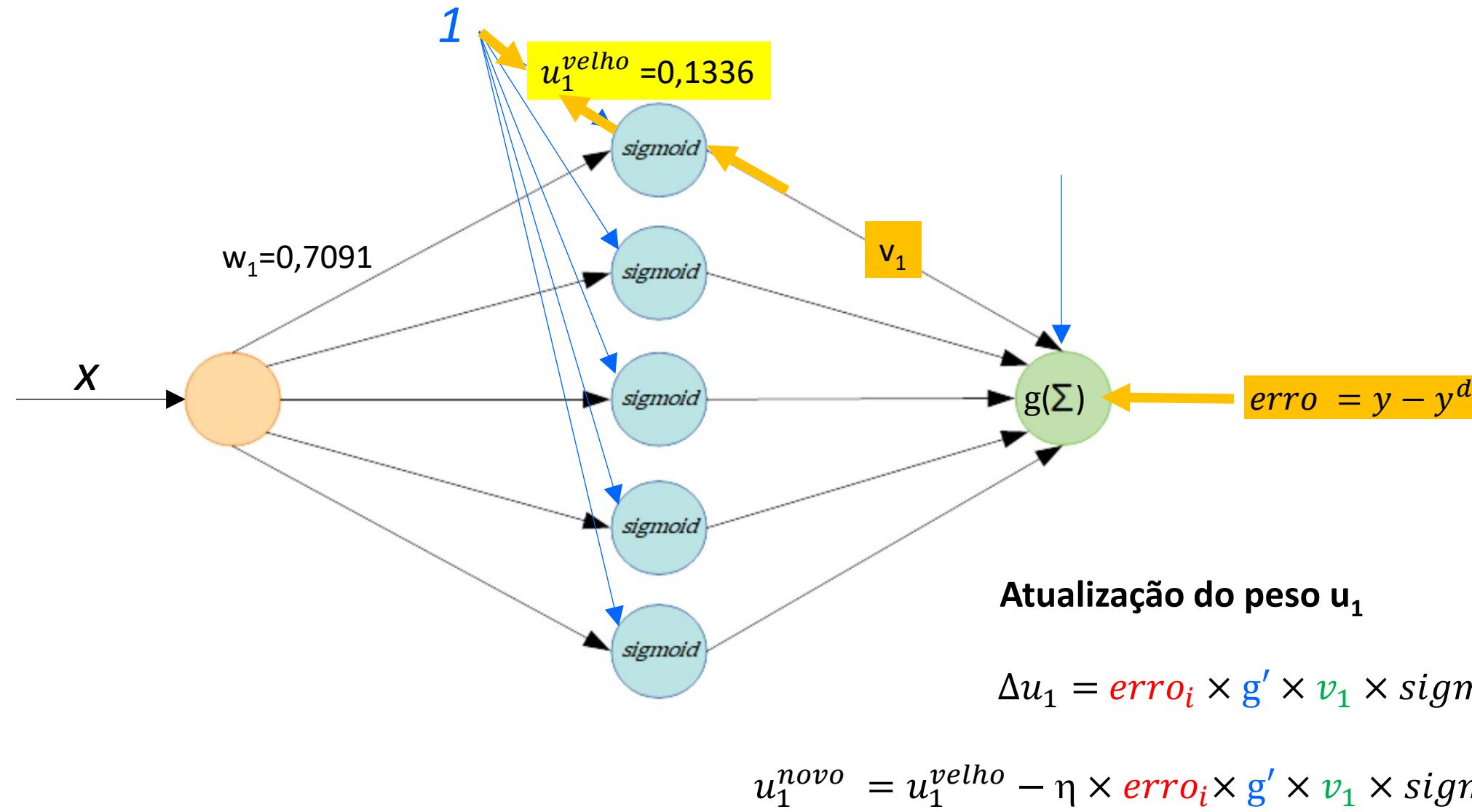
Atualização do peso b

$$\Delta b = \text{erro}_i \times g'$$

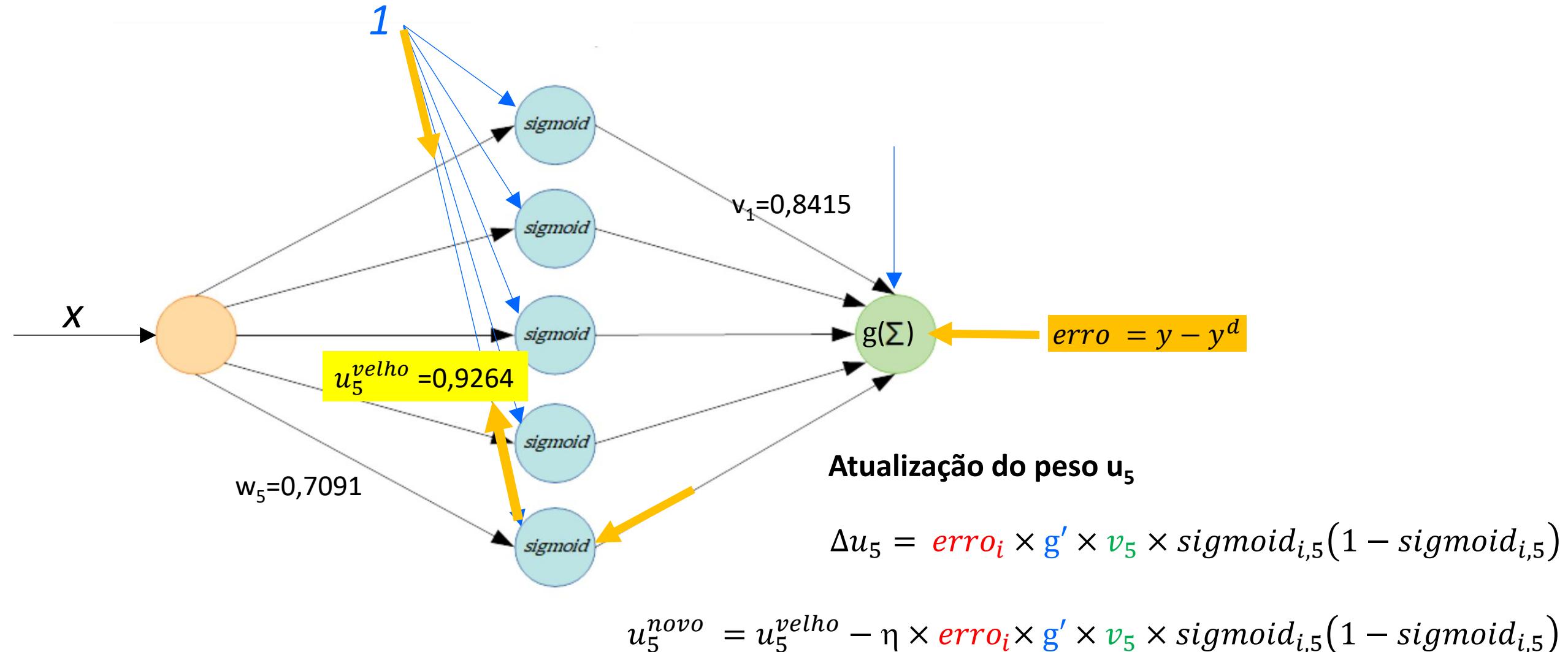
$$b^{\text{novo}} = b^{\text{velho}} - \eta \times \text{erro}_i \times g'$$

$$\text{erro} = y - y^d$$

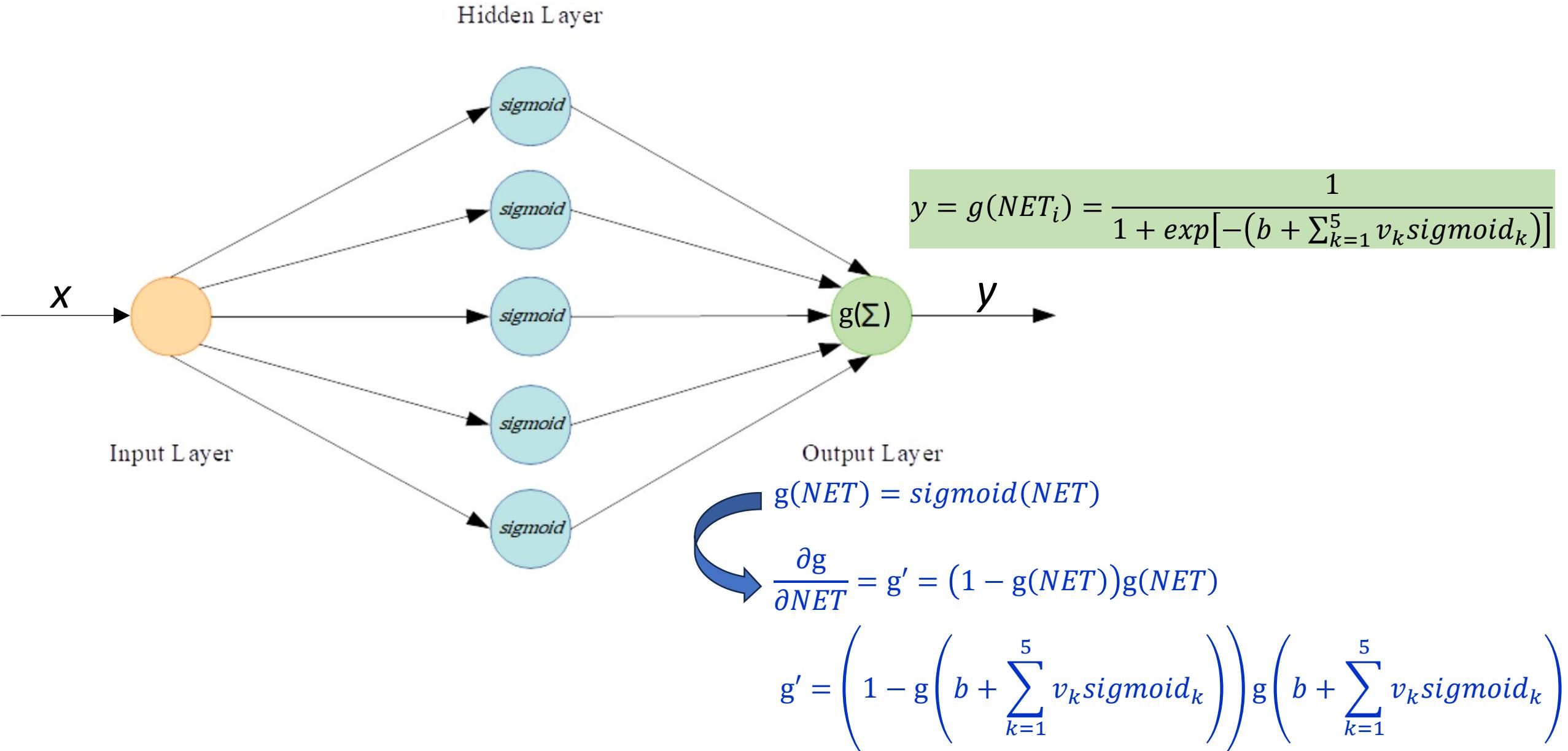
Fase Backward



Fase Backward



Fazendo $g(NET) = \text{sigmoide}(NET)$ no neurônio de saída



Implementação no R com função de ativação sigmoide no neurônio de saída

```
eta <- 0.01 # taxa de aprendizagem  
epocas <- 10000 # numero de épocas de treinamento
```

```
for (j in 1:epocas){      Treinamento da RNA  
  for (i in 1:500){  
    treina(trainSet[i,1],trainSet[i,2])  
  }  
}
```

```
estimativas=c()      Saída da rede para os exemplos do  
for (i in 1:500) {    conjunto de treinamento  
  estimativas = c( estimativas, saida(trainSet[i,1]))  
}
```

$$y = \frac{1}{1 + \exp[-(b + \sum_{k=1}^5 v_k sigmoid_k)]}$$

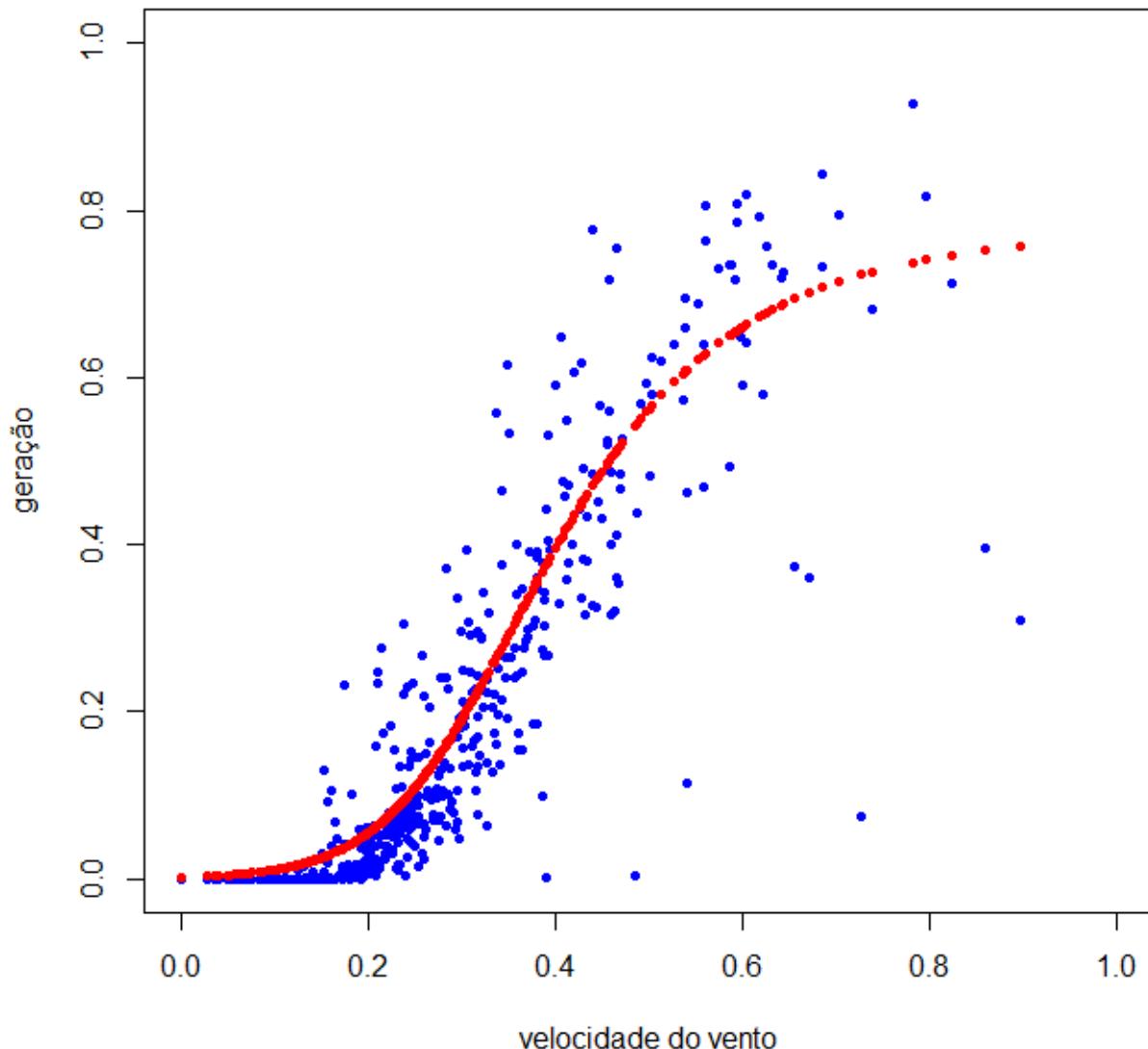
x	y
7921	0.31452922
6444	0.20657468
865	0.59780844
6527	0.15219156
8872	0.30275974
3560	0.02597403
8249	0.23051948
10	0.68425325
8080	0.31412338
4195	0.12134740

10 primeiros registros de *trainSet*
Conjunto de treinamento
formado por 500 exemplos

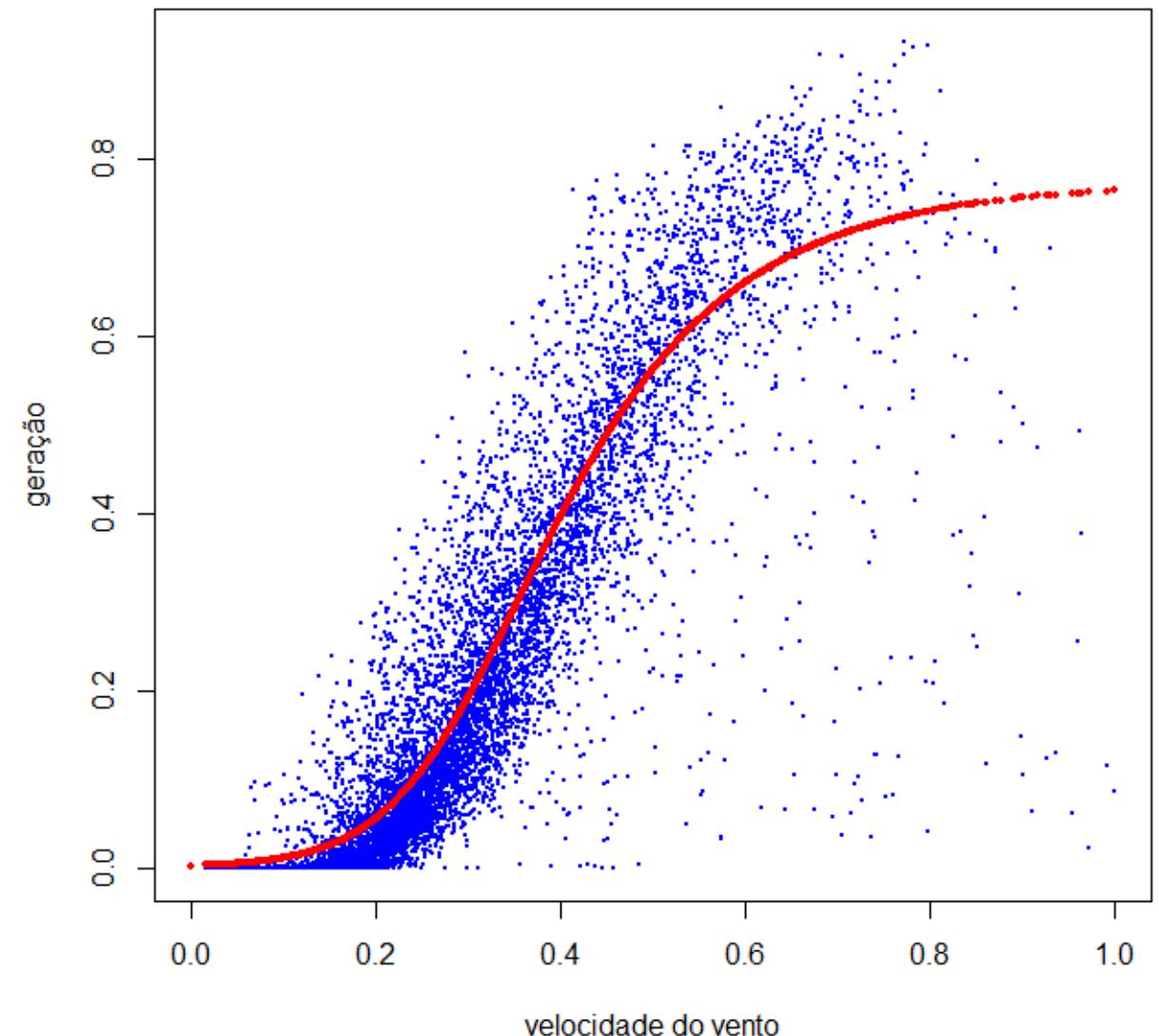


Implementação no R com função de ativação sigmoide no neurônio de saída

Resultados da RNA para o conjunto de treinamento



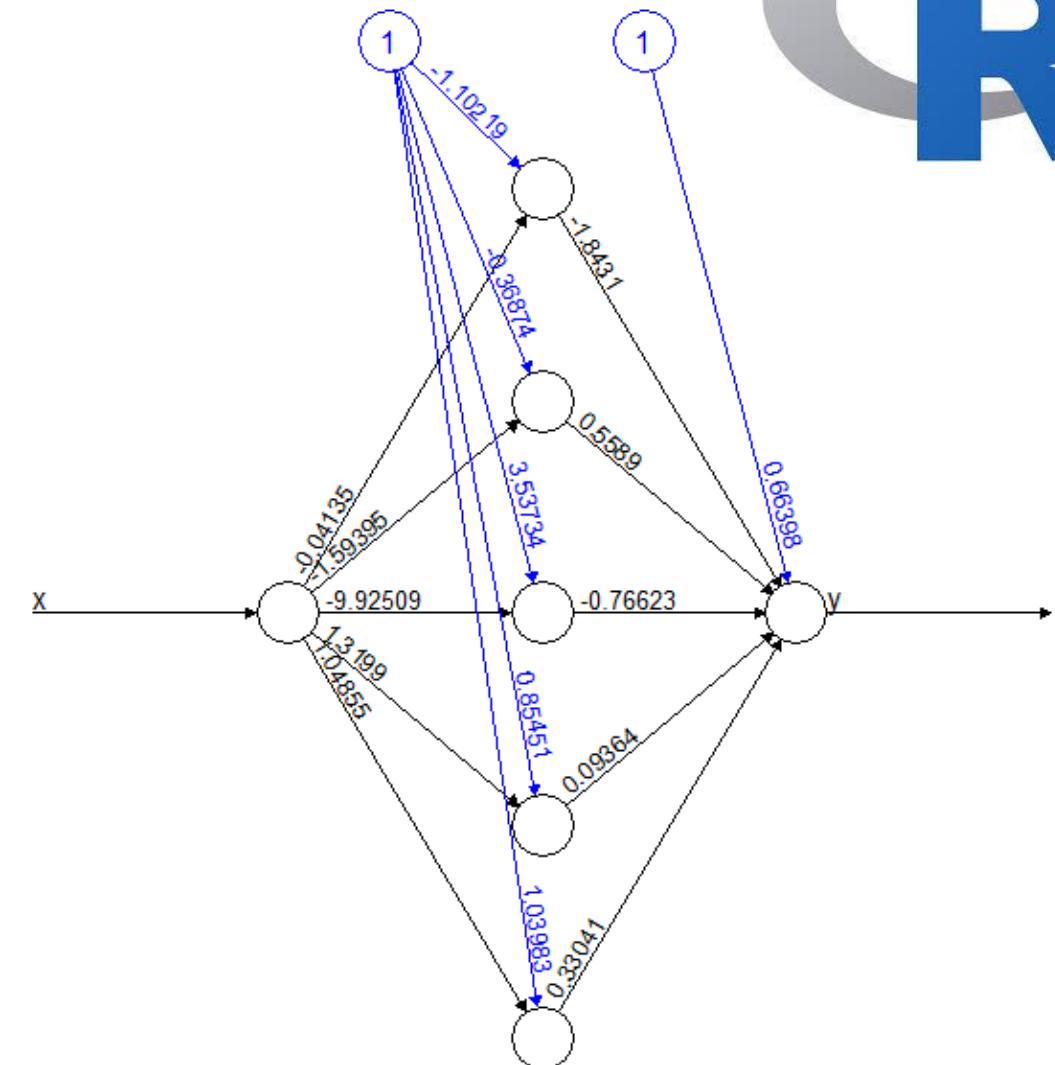
Resultados da RNA para o conjunto de dados



Exemplo com o pacote “neuralnet”



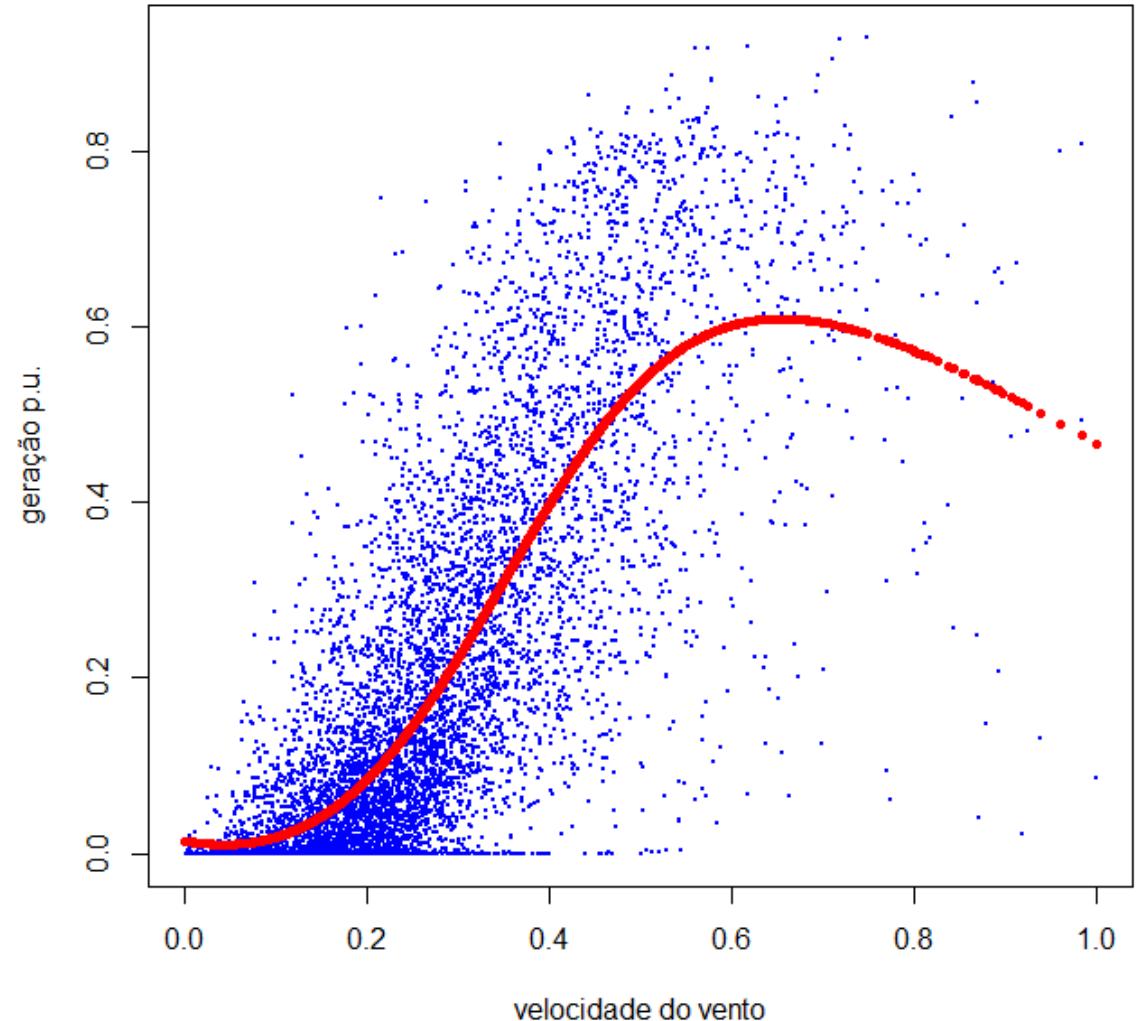
```
# leitura de dados
arquivo="c:/ventos/dados.csv"
dados=read.csv2(arquivo,header=F,dec=",")  
  
# padroniza dados (x = velocidade , y = geração)
x=(dados[,1]-min(dados[,1]))/(max(dados[,1])-min(dados[,1]))
y=dados[,2]/17560
nobs=length(x) # número de observações
amostras=data.frame(x,y)  
  
library(neuralnet)
# separa amostras de treinamento (insample) e outsample
indice_treina=sample(nobs,round(0.2*nobs))
dados_insample=amostras[indice_treina,]
dados_outsample=amostras[-indice_treina,]  
  
# treina rede neural para obter curva de potência
# rede com 5 neurônios na camada escondida
# função logística no neurônio de saída
rede=neuralnet( y ~ x, hidden=5, data=dados_insample,act.fct="logistic")
plot(rede) # diagrama da rede neural
```



Error: 16.358009 Steps: 15390

Exemplo com o pacote “neuralnet”

```
windows()  
estimativa=predict(redede,dados_outsample)  
plot(dados_outsample[,1],dados_outsample[,2],col="blue",pch=20,cex=0.5,xlab="velocidade do vento",ylab="geração p.u.")  
points(dados_outsample[,1],estimativa,col="red",pch=20)
```



Exemplo com o pacote “neuralnet”



```
arquivo="C:/VENTOS/dados_RNA.csv"dados=read.csv2(arquivo,header=F,dec=",",sep=";")
```

```
angulo=dados[,1]
```

```
velo=dados[,2]
```

```
# coordenadas cartesianas da velocidade do vento
```

```
U=velo*cos(angulo*pi/180)
```

```
V=velo*sin(angulo*pi/180)
```

```
x1=(U-min(U))/(max(U)-min(U))
```

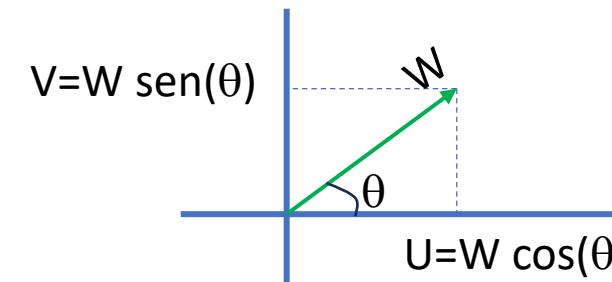
```
x2=(V-min(V))/(max(V)-min(V))
```

```
y=dados[,3]/17600 # produção
```

```
amostras=data.frame(x1,x2,y)
```

```
nobs=length(y)
```

$$\text{Produção} = f(U, V)$$



```
# separa amostras de treinamento (insample) e outsample
```

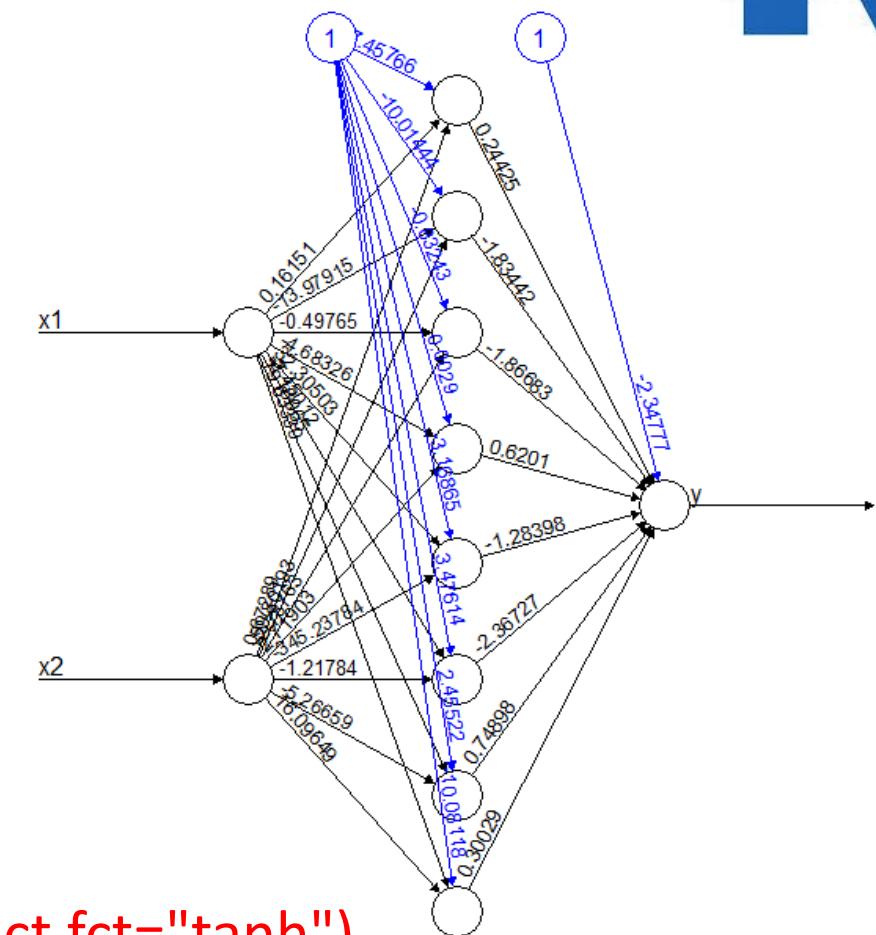
```
indice_treina=sample(nobs,round(0.1*nobs))
```

```
dados_insample=amostras[indice_treina,]
```

```
dados_outsample=amostras[-indice_treina,]
```

```
# treina rede neural
```

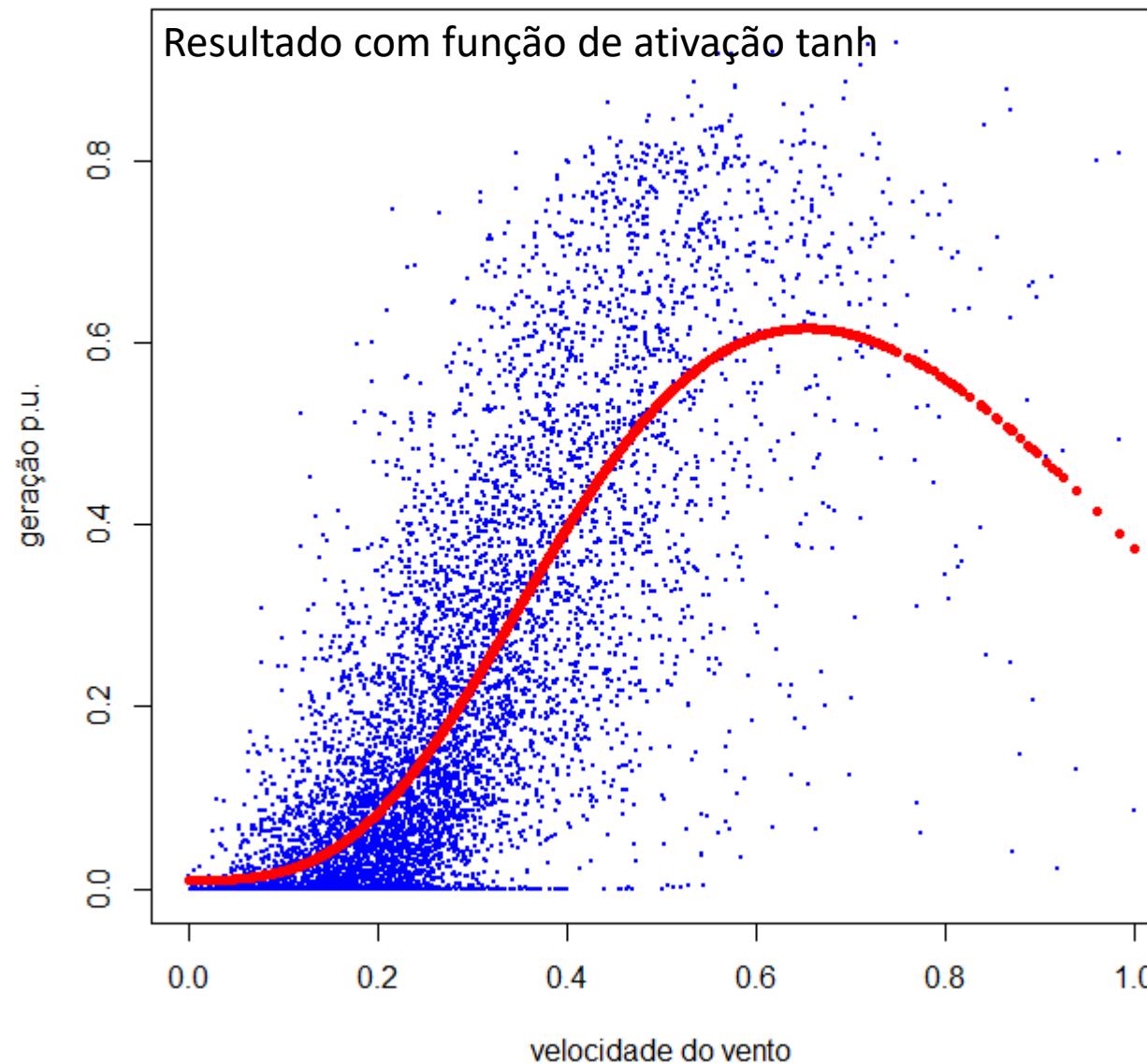
```
rede=neuralnet(y~x1+x2,hidden=8,data=dados_insample,act.fct="tanh")
```



Error: 3.155227 Steps: 72575

Exemplo com o pacote “neuralnet”

```
rede=neuralnet(y~x,hidden=5,data=dados_insample,act.fct="tanh")
```



Previsão de séries temporais com RNA

- Uma série temporal é um conjunto de observações ordenadas no tempo
- Informação fundamental para fazer previsões
- Previsão é muito difícil, especialmente se for sobre o futuro (Niels Bohr)



Niels Bohr
1885 - 1962

*Previsão de séries temporais de velocidade do vento
utilizando Redes Neurais Artificiais e Métodos
Estatísticos na região de Arraial do Cabo - RJ*

Ricardo Teixeira¹, Diego da Silva¹, Harold de Mello Junior¹, Leonardo Forero¹, Antonio Lima¹, Karla Figueiredo²

¹Departamento de Engenharia Elétrica

²Departamento de Informática e Ciência da Computação
Universidade do Estado do Rio de Janeiro (UERJ)

Rio de Janeiro, Brasil

{ricardo.mauroct7021, lemos.diego.07, harold.dias}@gmail.com,
leofome@hotmail.com, antoniolima@uerj.br, karla.figueiredo@gmail.com

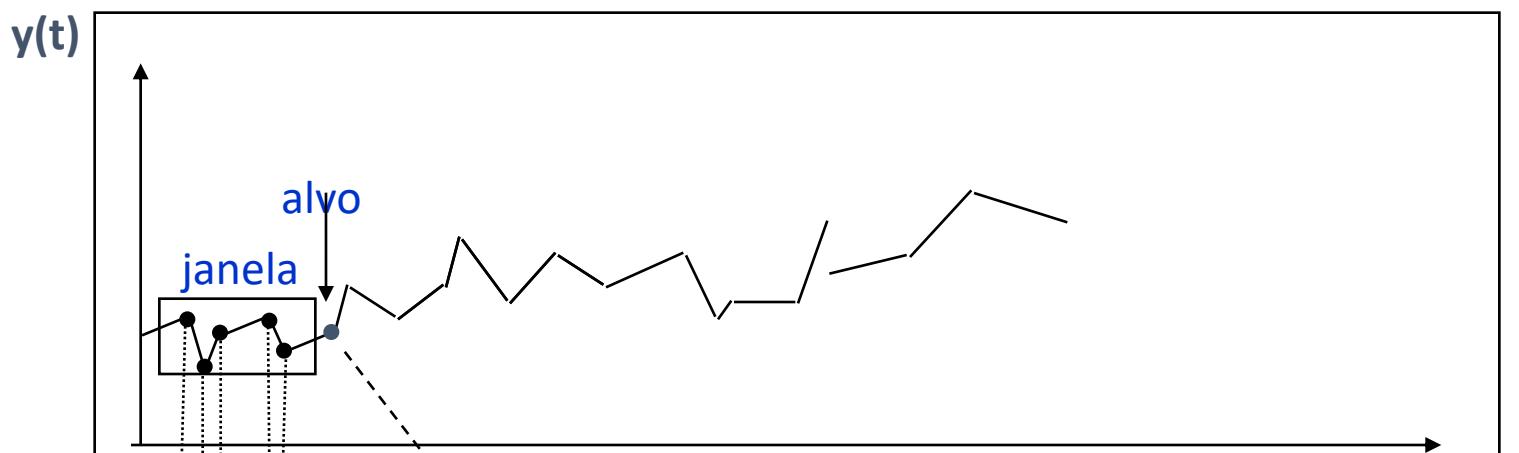
[CBIC2019-101.pdf](#)

Previsão de séries temporais com RNA

- Considere um modelo univariado: $y(t) = f(y(t-1), \dots, y(t-5))$
- Considere previsão 1 passo à frente

A previsão depende dos cinco últimos valores passados (janela) da própria série temporal

Geração dos padrões de treinamento da RNA



Entradas
da rede
formada pelos
5 valores
passados
janela de
entrada

janela de saída
(alvo)

5 valores
passados

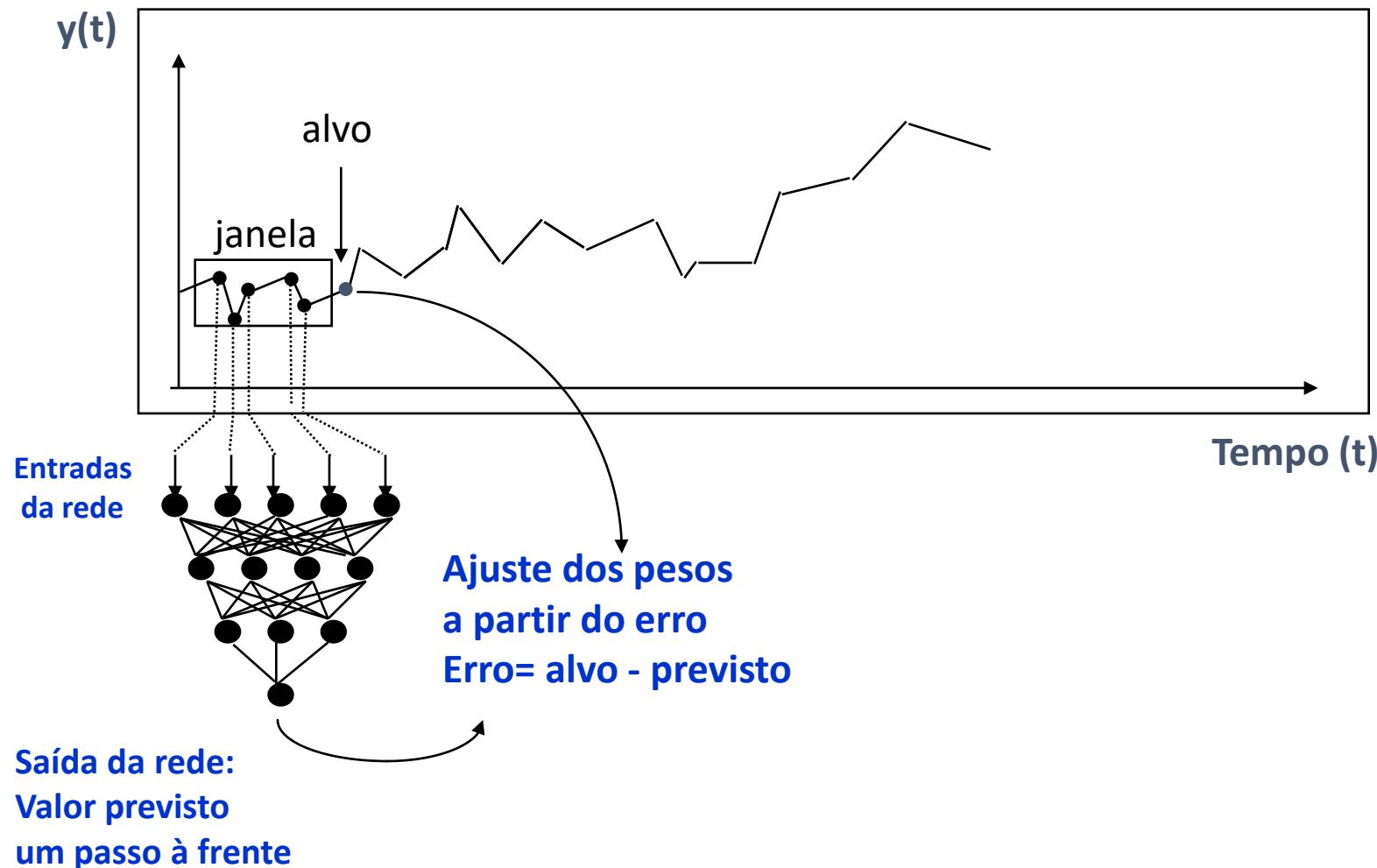
Entrada : y_1, y_2, y_3, y_4, y_5

Saída
Desejada
corresponde ao
valor da série
1 passo à
frente

Saída: y_6

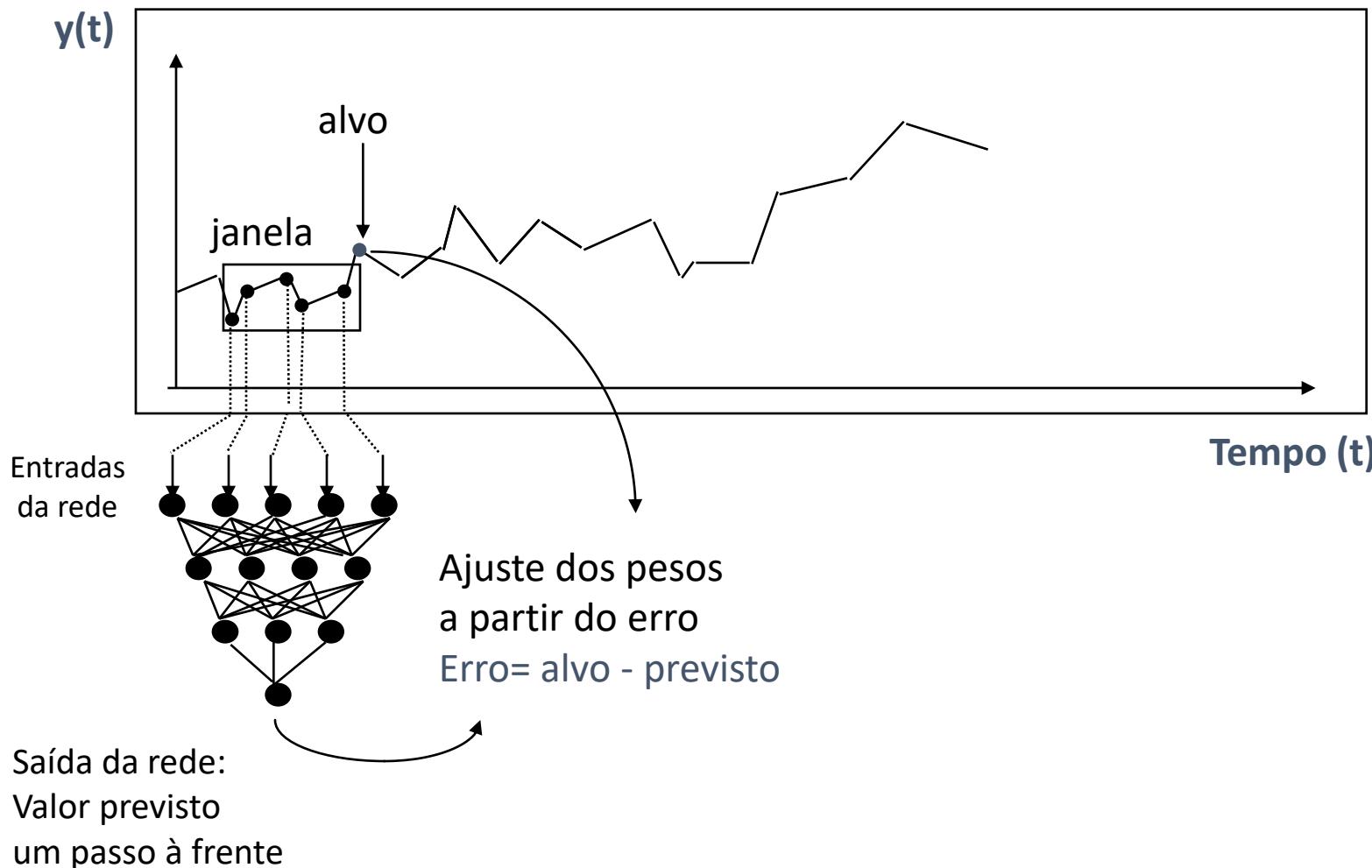
Previsão de séries temporais com RNA

- Treina a rede com o exemplo



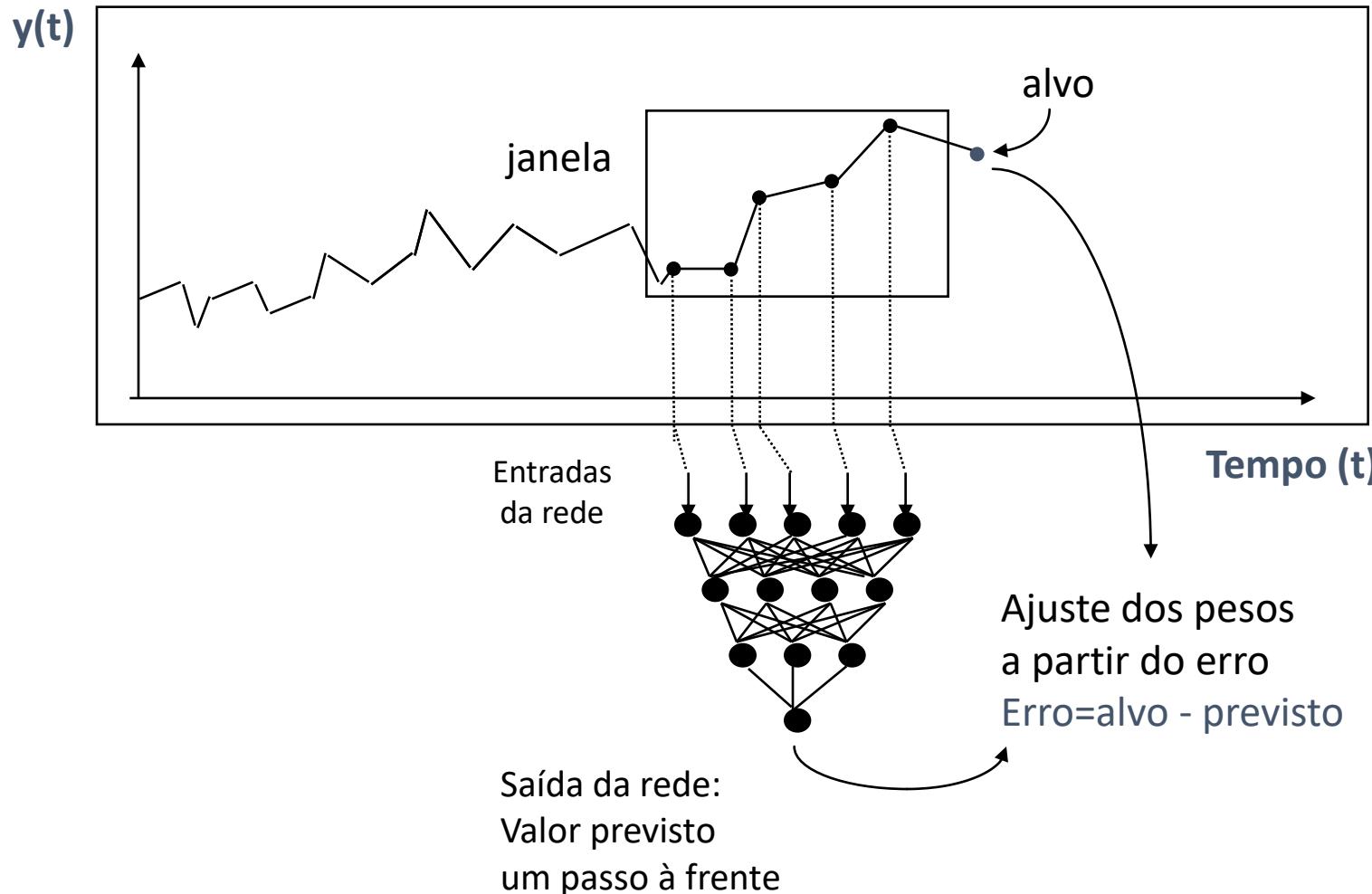
Previsão de séries temporais com RNA

- Move as janelas de entrada e saída 1 passo à frente
- Forma novos padrões entrada/saída
- Treina a rede



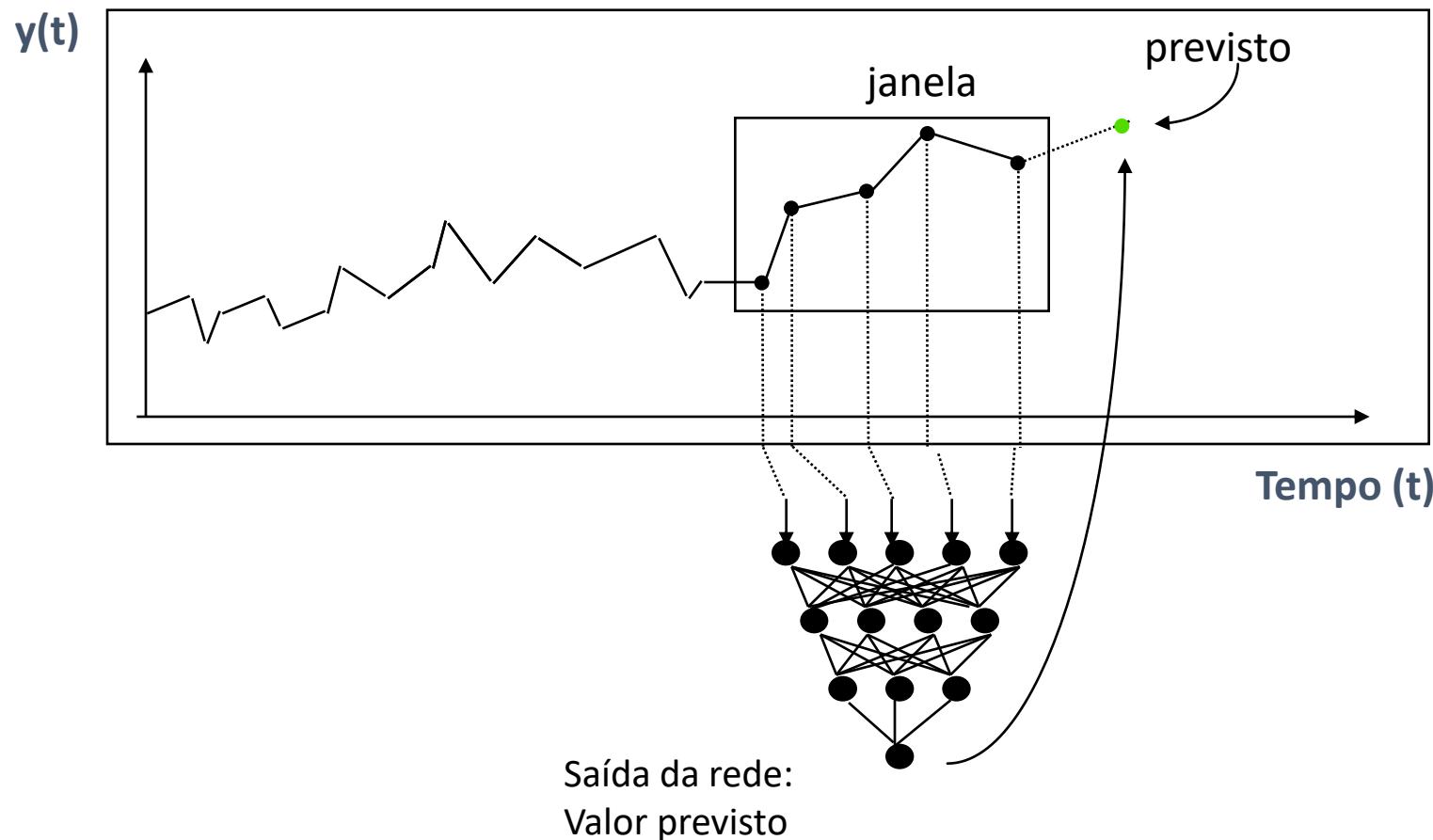
Previsão de séries temporais com RNA

- Move as janelas de entrada e saída 1 passo à frente até o final da série
- Forma novos padrões entrada/saída a cada passo
- Treina a rede a cada passo



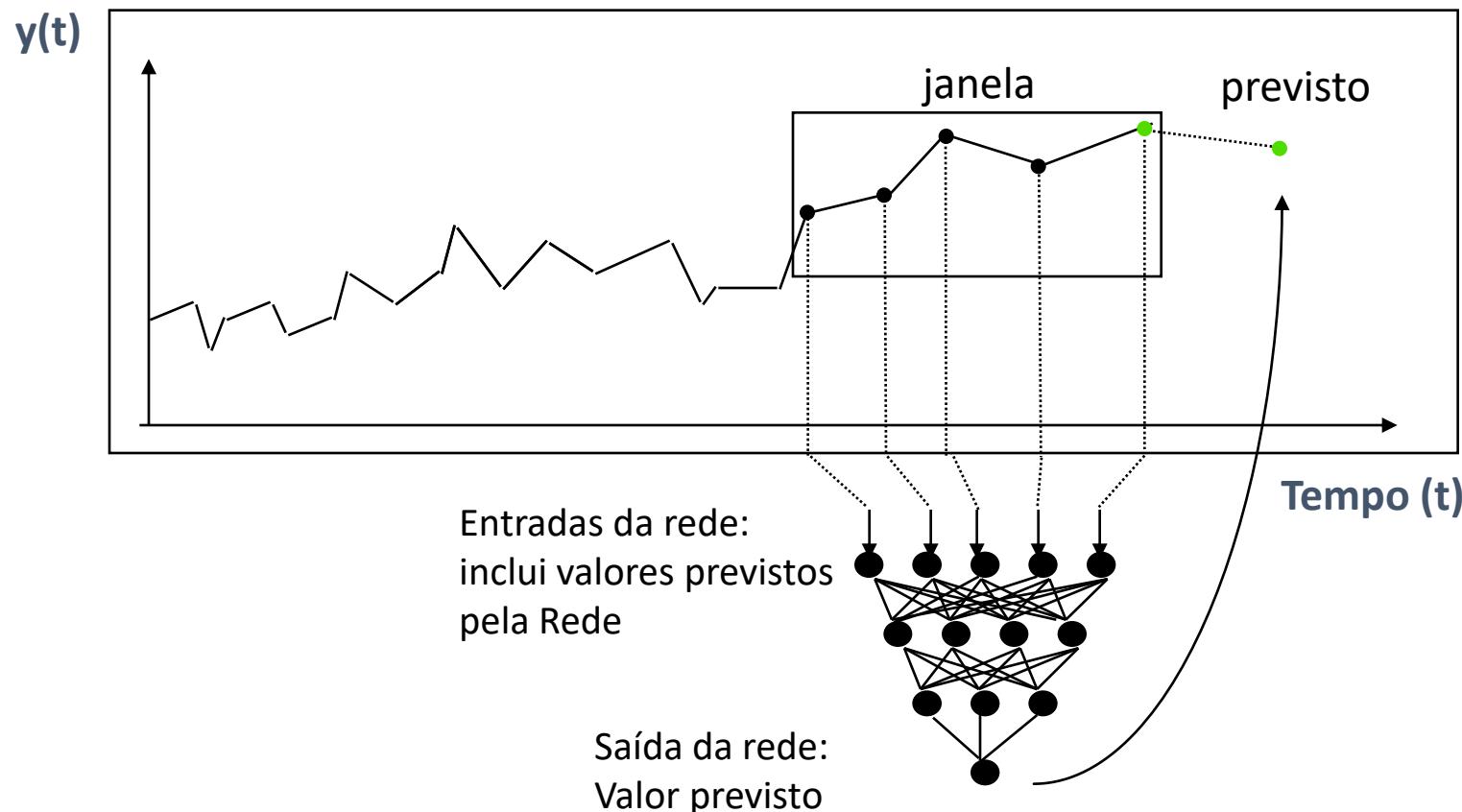
Previsão de séries temporais com RNA

Previsão 1 passo à frente



Previsão de séries temporais com RNA

Previsão 2 passos à frente com realimentação da previsão



Previsão da média mensal da velocidade do vento com RNA

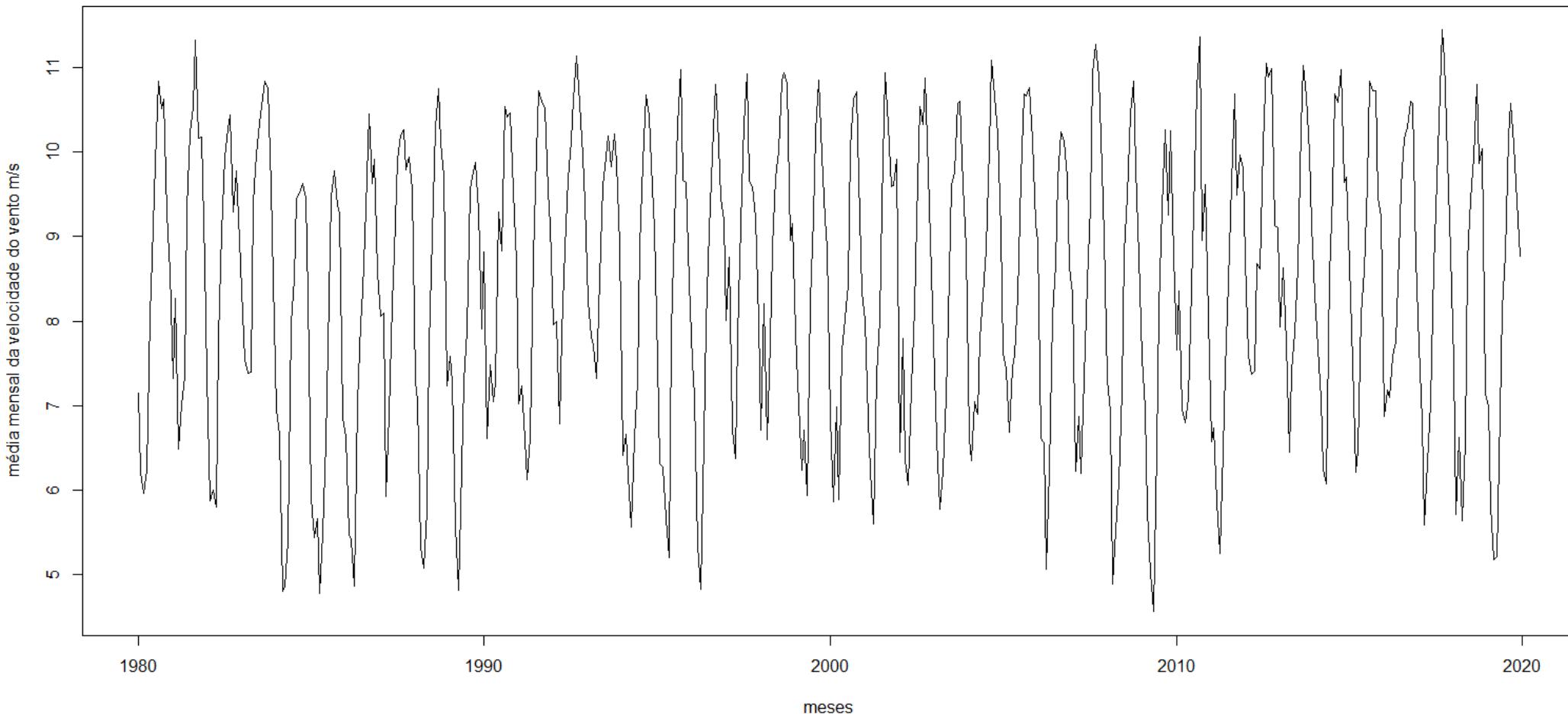
Médias mensais das reanálises da velocidade do vento em Acaraú II 230 kV (MERRA 2)

([Renewables.ninja](https://renewables.ninja))

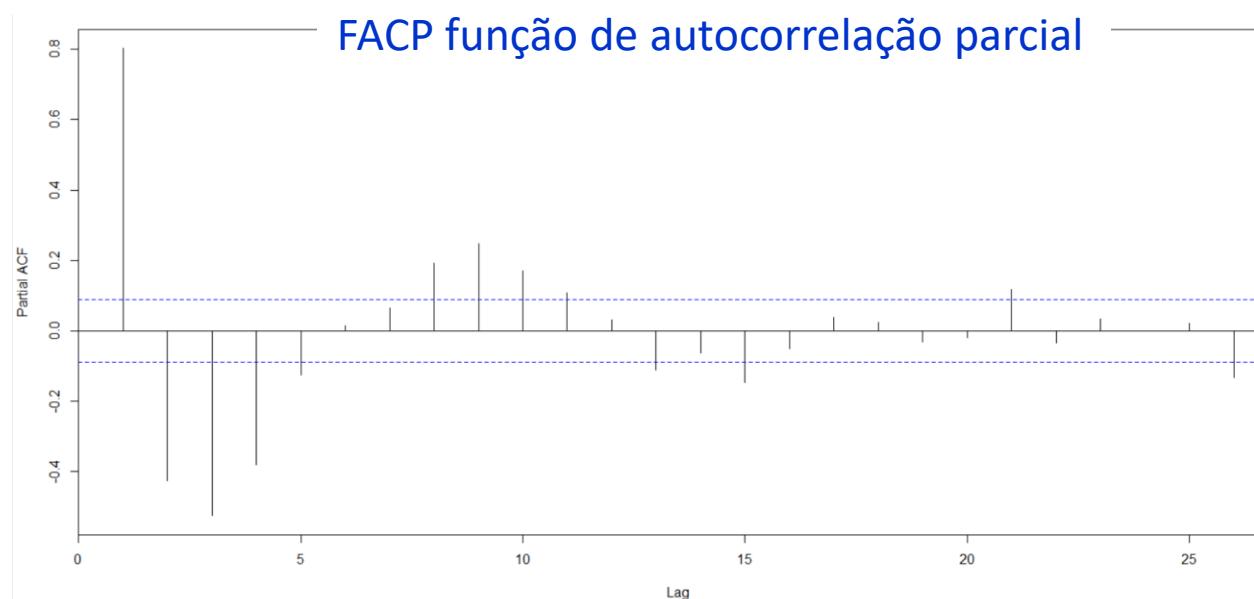
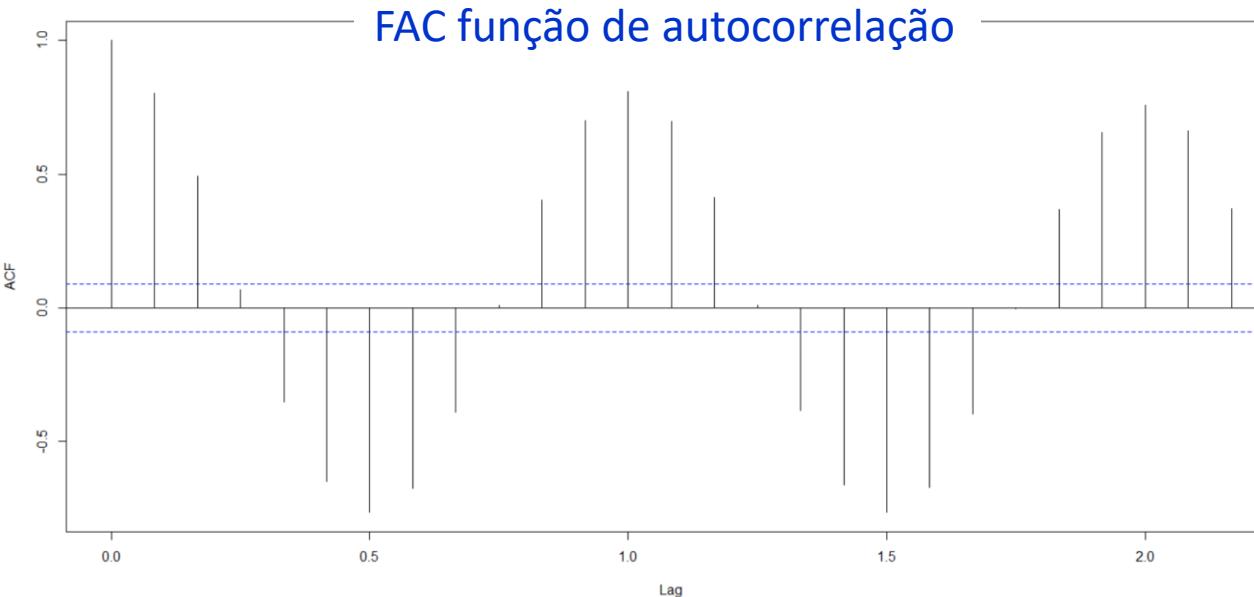


Renewables.ninja

480 médias
mensais de 1980/1
até 2019/12



Previsão da média mensal da velocidade do vento com RNA



A FAC e FACP exibem a estrutura de autocorrelação da série temporal, uma informação fundamental na seleção das variáveis de entrada da rede neural

Previsão da velocidade média mensal no mês m em função das velocidades nos meses $m-1, m-2, m-3, m-4, m-5, m-6, m-7, m-11, m-12$ e $m-13$

Inicialmente, vamos considerar redes com uma camadas escondida

$$\begin{aligned} \text{num. neurônios na camada escondida} &= \\ (\text{num. entradas} + \text{num. saída}) \times 2/3 & \\ (\text{BEYSOLOW II, 2017}) \end{aligned}$$

Neste caso $(10 + 1) \times 2/3 = 7,33 \cong 8$

Previsão da média mensal da velocidade do vento com RNA

```
lags=c(1,2,3,4,5,6,7,11,12,13)
```

```
insample=serie[1:456] # Seleciona conjunto insample (dados para treinamento da rede)
```

```
outsample=serie[457:480] # Seleciona conjunto outsample (dados para testar ou validar o modelo)
```

```
media=mean(insample) # Média da série insample
```

```
dp=sd(insample) # Desvio padrão da série insample
```

```
serie_padronizada=(insample-media)/dp # Série padronizada
```

```
Inputs=c() Inicializa janelas de entrada
```

```
Targets=c() Inicializa janelas de saída (alvo)
```

```
for (i in max(lags+1):(length(serie_padronizada))){
```

```
    Inputs=rbind(Inputs,serie_padronizada[i-lags])
```

```
    Targets=c(Targets, serie_padronizada[i])
```

```
}
```

Percorre a série temporal padronizada para formar as janelas de entrada/saída



Previsão da média mensal da velocidade do vento com RNA

```
nData=dim(Inputs)[1] # Número de exemplos de treinamento ou pares entrada/saída  
pData=dim(Inputs)[2] # Número de variáveis na janela de entrada  
dados = data.frame(Inputs,Targets)/100 # Unifica as janelas de entrada e saída em data.frame e divide por 100  
colnames(dados)=c(paste("X",seq(1,pData,1),sep=""),"Y") # atribui nomes às variável (Target Y)
```

	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	Y
1	-0.006754762	0.000785056	0.004944942	0.012990878	0.012280810	0.014192759	0.008496378	-0.014902300	-0.013354961	-0.007815664	-0.001137467
2	-0.001137467	-0.006754762	0.000785056	0.004944942	0.012990878	0.012280810	0.014192759	-0.013267522	-0.014902300	-0.013354961	-0.011732362
3	-0.011732362	-0.001137467	-0.006754762	0.000785056	0.004944942	0.012990878	0.012280810	-0.002279677	-0.013267522	-0.014902300	-0.008599837
4	-0.008599837	-0.011732362	-0.001137467	-0.006754762	0.000785056	0.004944942	0.012990878	0.001732909	-0.002279677	-0.013267522	-0.006631289
5	-0.006631289	-0.008599837	-0.011732362	-0.001137467	-0.006754762	0.000785056	0.004944942	0.008496378	0.001732909	-0.002279677	0.005182361
6	0.005182361	-0.006631289	-0.008599837	-0.011732362	-0.001137467	-0.006754762	0.000785056	0.014192759	0.008496378	0.001732909	0.010592857

```
library(nnet) # Ativa pacote nnet  
set.seed(1513) # fixa semente do gerador de números aleatórios  
neuron=8 # número de neurônios na camada escondida  
rede = nnet(Y~.,data=dados,size=neuron,linout=T,maxit=100000) # treina a RNA
```

O target Y é função de todas as variáveis em dados

Função de ativação linear no neurônio de saída

PACOTE NNET

- Redes com apenas uma camada escondida
- Neurônios da camada escondida com função de ativação sigmoide



Previsão da média mensal da velocidade do vento com RNA

Z=insample

```
entrada=((Z[length(Z)+1-lags]-media)/dp)/100 # Cria padrão de entrada para prever 1 mês à frente a partir do último dado insample
```

```
nomesvariaveis= paste("X",seq(1,length(entrada),1),sep="")
```

```
names(entrada)= nomesvariaveis
```

```
apenas_previsoes=c()
```

```
for (horizonte in 1:24){ # Previsão até 24 meses à frente com realimentação
```

```
    previsao = as.numeric(predict(rede,t(entrada)))
```

```
    previsao = (previsao*100)*dp+media
```

```
    Z=c(Z,previsao) # Realimenta a previsão
```

```
    apenas_previsoes =c(apenas_previsoes,previsao)
```

```
    entrada=((Z[length(Z)+1-lags]-media)/dp)/100
```

```
    names(entrada)= nomesvariaveis
```

Cria padrão de entrada
para a próxima previsão

```
}
```

```
minimo=min(outsample, apenas_previsoes)
```

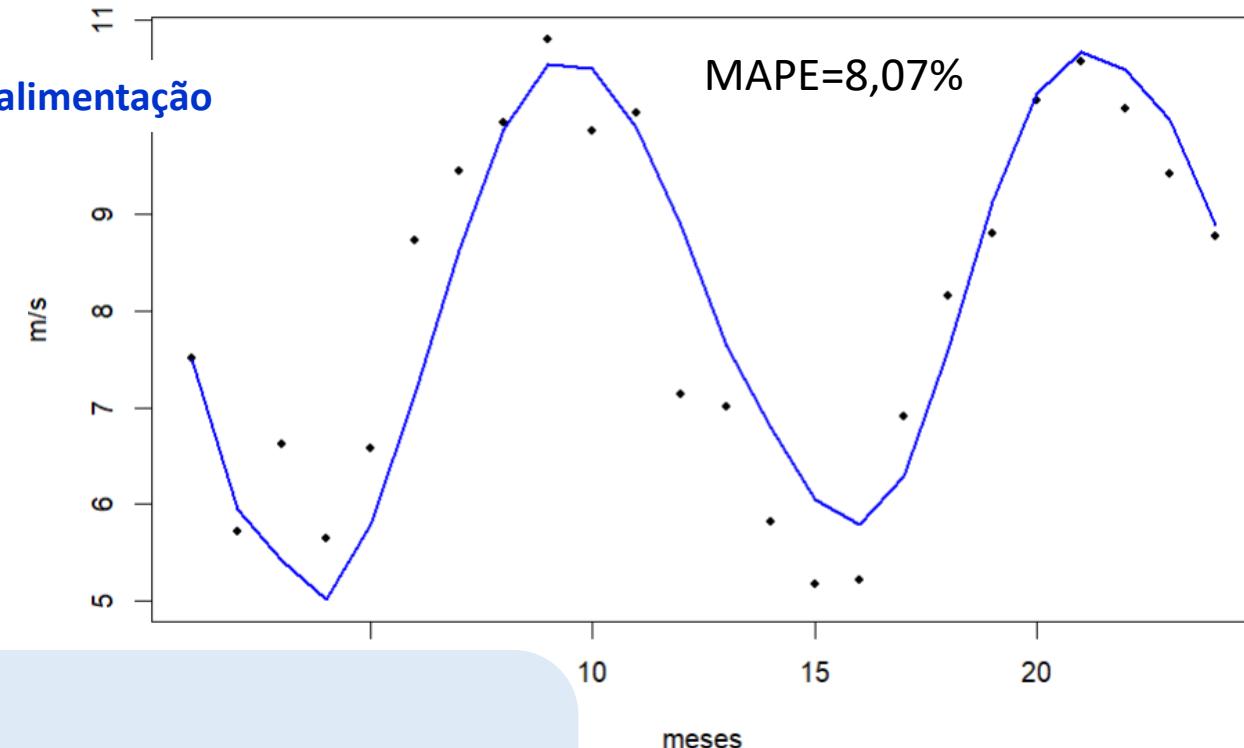
```
maximo=max(outsample, apenas_previsoes)
```

```
plot(outsample,ylim=c(minimo,maximo),pch=20,col="black",xlab="meses",ylab="m/s")
```

```
lines(apenas_previsoes,col="blue",lwd=2)
```

```
MAPE=mean(abs(outsample- apenas_previsoes)/outsample)
```

```
print(MAPE*100)
```



Faz gráfico e calcula o
Mean Absolute
Percentage Error (MAPE)

$$MAPE = \frac{1}{n} \sum_{t=1}^n \frac{|observado_t - previsto_t|}{observado_t} \times 100\%$$

Previsão com o pacote Neuralnet

O pacote Neuralnet permite a função de ativação *tanh* e pode treinar redes com mais de uma camada escondida. O pacote oferece uma função para desenhar a rede neural.

Vamos usar os mesmos exemplos do exercício anterior

	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	Y
1	-0.006754762	0.000785056	0.004944942	0.012990878	0.012280810	0.014192759	0.008496378	-0.014902300	-0.013354961	-0.007815664	-0.001137467
2	-0.001137467	-0.006754762	0.000785056	0.004944942	0.012990878	0.012280810	0.014192759	-0.013267522	-0.014902300	-0.013354961	-0.011732362
3	-0.011732362	-0.001137467	-0.006754762	0.000785056	0.004944942	0.012990878	0.012280810	-0.002279677	-0.013267522	-0.014902300	-0.008599837
4	-0.008599837	-0.011732362	-0.001137467	-0.006754762	0.000785056	0.004944942	0.012990878	0.001732909	-0.002279677	-0.013267522	-0.006631289
5	-0.006631289	-0.008599837	-0.011732362	-0.001137467	-0.006754762	0.000785056	0.004944942	0.008496378	0.001732909	-0.002279677	0.005182361
6	0.005182361	-0.006631289	-0.008599837	-0.011732362	-0.001137467	-0.006754762	0.000785056	0.014192759	0.008496378	0.001732909	0.010592857

```
library(neuralnet) # Ativa pacote neuralnet
```

Função de ativação sigmoide (logistic)
nos neurônios da camada escondida

```
set.seed(1513) # fixa semente do gerador de números aleatórios
```

PACOTE
NEURALNET

```
neuron=8 # número de neurônios na camada escondida
```

```
rede = neuralnet(Y~.,data=dados,hidden=neuron,stepmax=100000,rep=100,linear.output=T)
```

treina a rede neural

O target Y é função de todas as variáveis em dados

Número máximo de iterações do treinamento

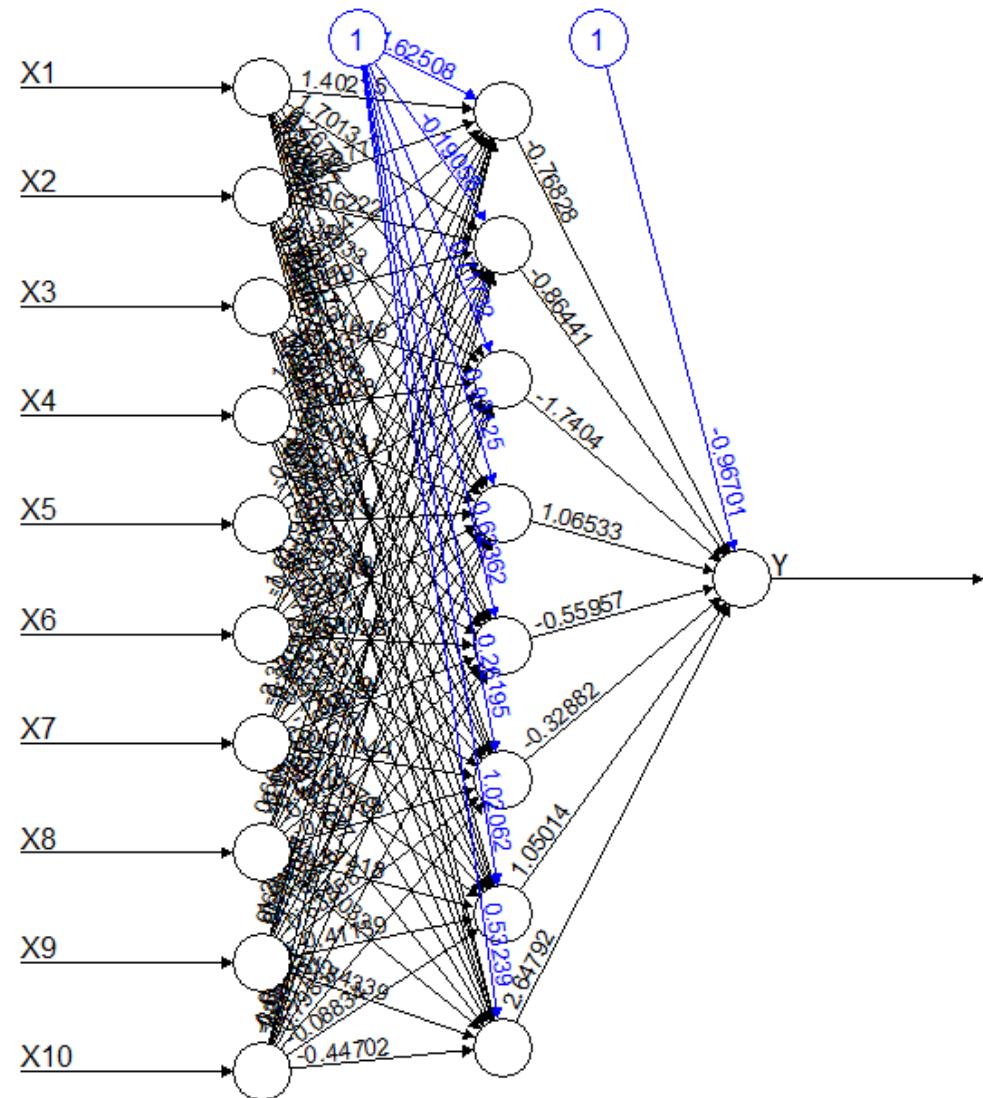
Número de repetições do treinamento

Função de ativação linear no neurônio de saída



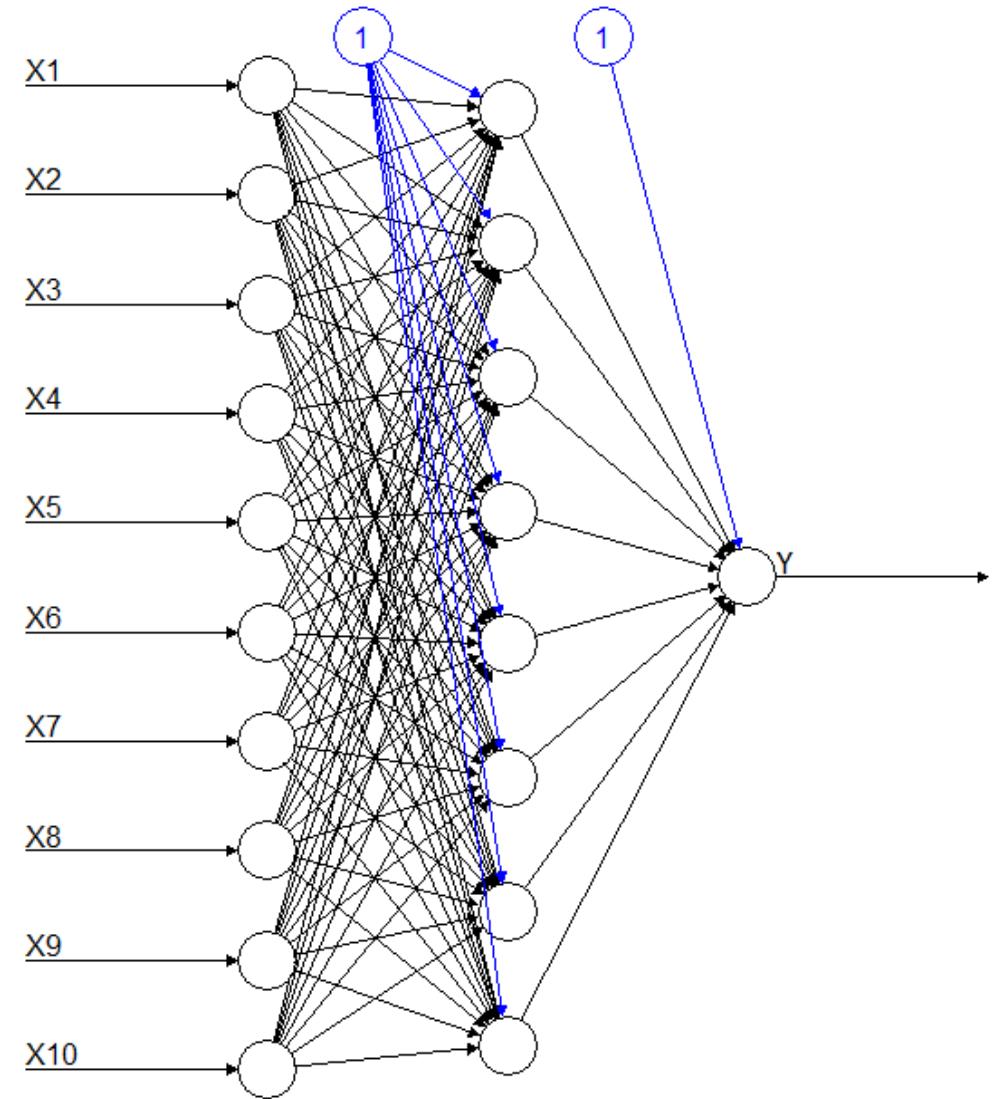
Previsão com o pacote Neuralnet

`plot(redes, rep = "best")`



Error: 0.0002204 Steps: 27

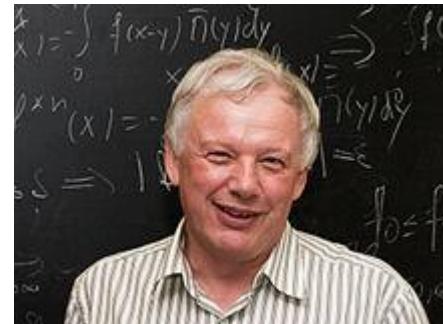
`plot(redes, rep = "best", show.weights = F)`



Error: 0.0002204 Steps: 27

Redes RBF – Radial Basis Function (1988)

- Proposta por Broomhead e Lowe em 1988
- Função de ativação gaussiana nos neurônios da camada oculta
- Função de ativação linear no neurônio da camada de saída, a resposta da rede é uma combinação linear de funções de base radial das entradas.
- Aprendizagem rápida
- Utilizada nas mesmas tarefas da MLP



David S. Broomhead

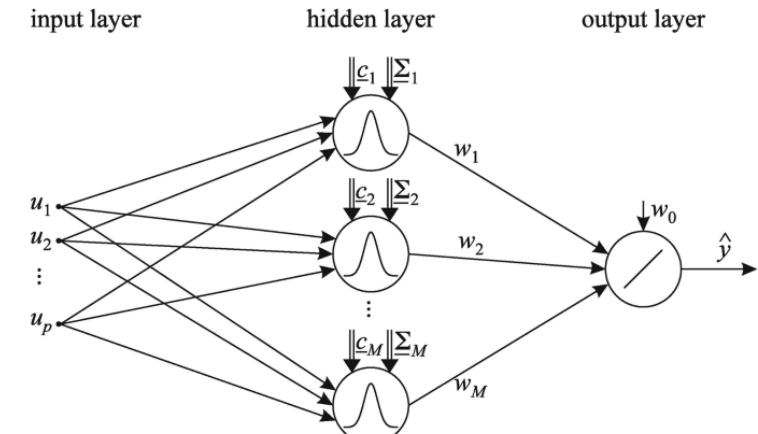
1950 - 2014



David Lowe

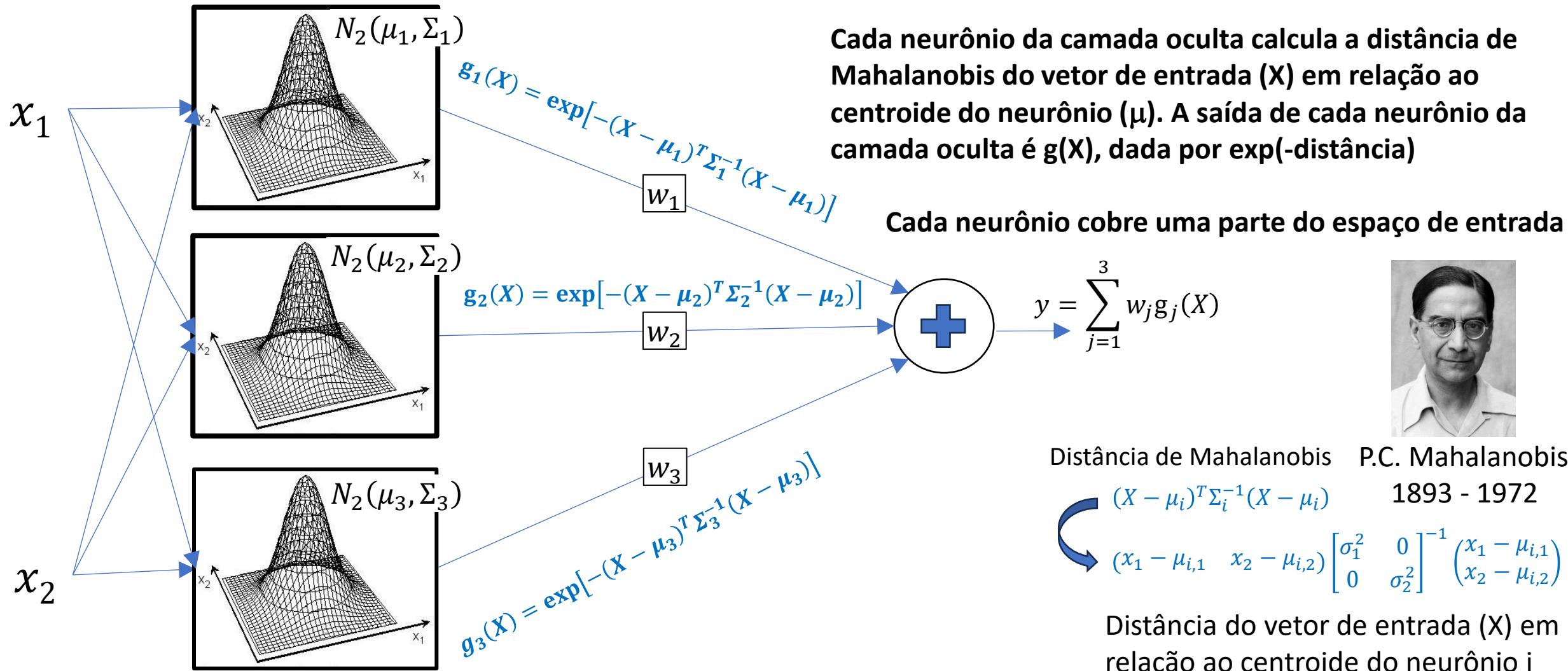
RADIAL BASIS FUNCTIONS, MULTI-VARIABLE
FUNCTIONAL INTERPOLATION AND ADAPTIVE NETWORKS

Authors: D S Broomhead and D Lowe



Redes RBF – Radial Basis Function (1988)

Neste exemplo, em cada neurônio da camada oculta, a função de ativação é uma normal bivariada, caracterizada por um centroide μ e matriz de covariâncias Σ .



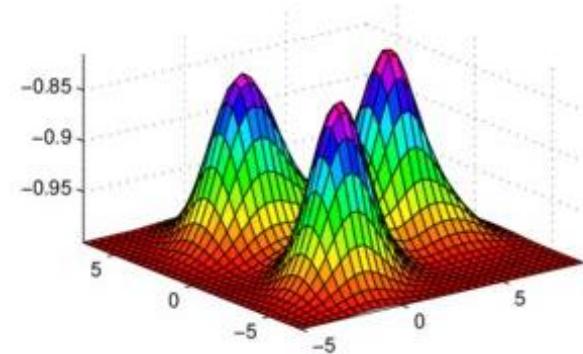
Redes RBF – Radial Basis Function (1988)

Treinamento em duas etapas:

Etapa 1) Obtenção dos parâmetros dos neurônios da camada oculta (treinamento não supervisionado).

1.1) Segmentação (*clustering*) do conjunto formada pelos n exemplos de treinamento em k clusters ($k < n$).
Em geral, a segmentação é realizada pelo método K-Means .

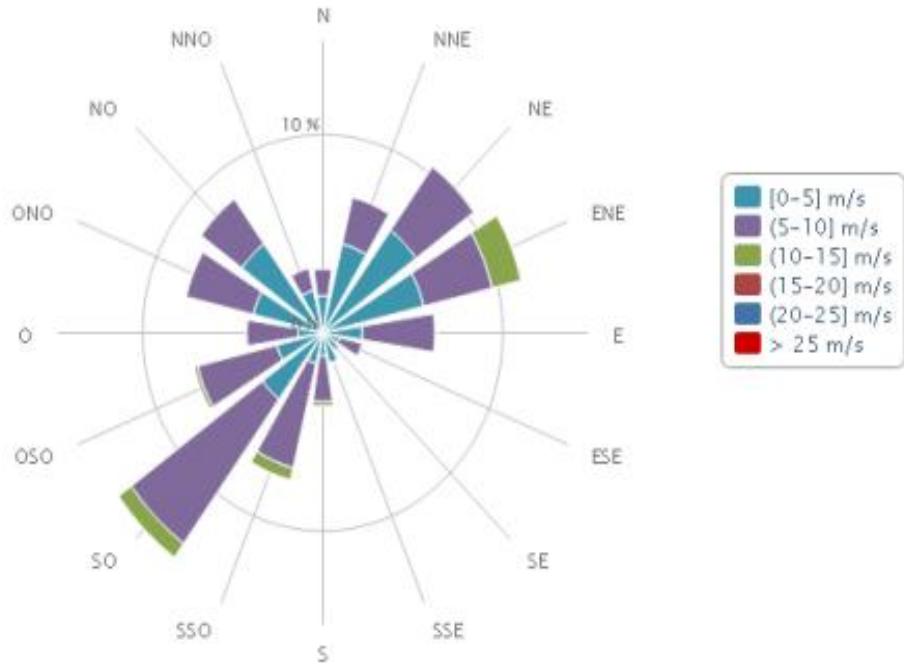
1.2) Cada cluster corresponde à um neurônio da RBF, calcule o centroide (vetor de médias) e a matriz de covariâncias em cada cluster) e a matriz de covariâncias.



Etapa 2) Ajuste dos pesos que conectam a camada oculta com a camada de saída (treinamento supervisionado). Pode ser utilizado o método da retropropagação do erro ou mínimos quadrados com a matriz pseudo inversa

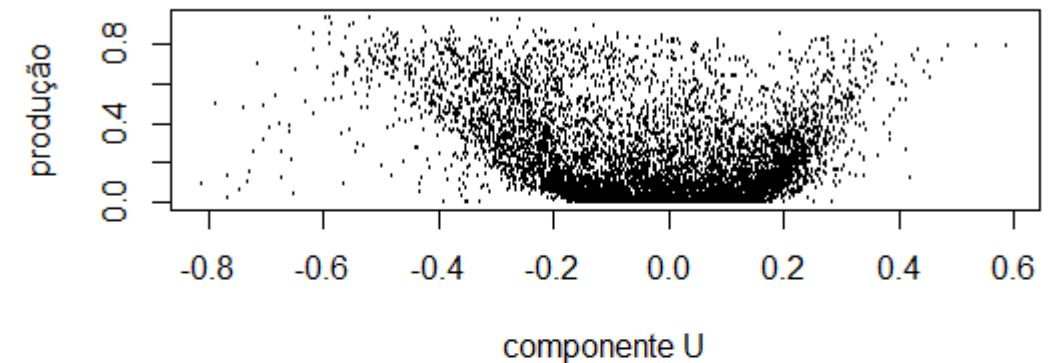
Introdução da direção do vento na curva de potência da geração eólica

Direção do vento



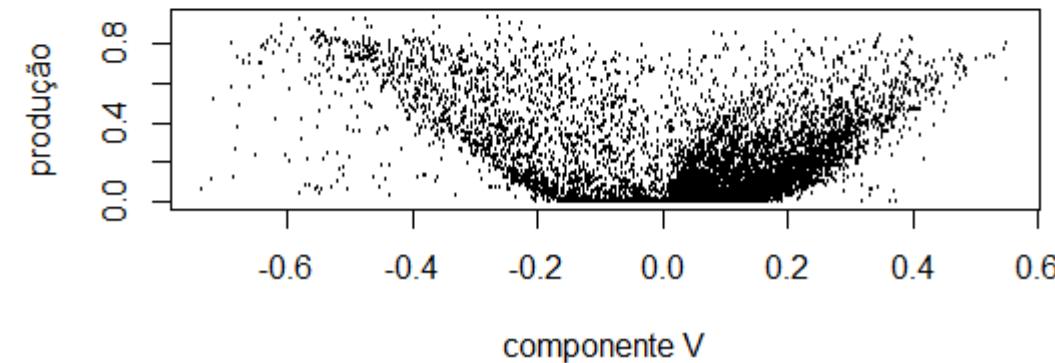
$U = \text{velocidade do vento} \times \cos(\text{ângulo})$

Componente U da velocidade do vento e produção

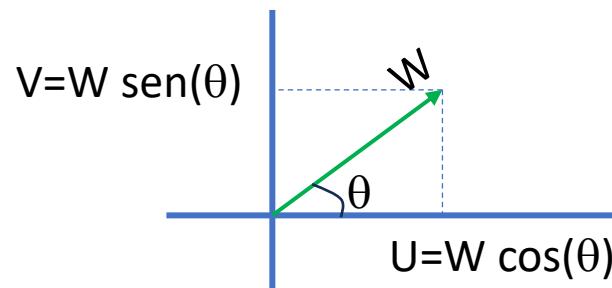


$V = \text{velocidade do vento} \times \sin(\text{ângulo})$

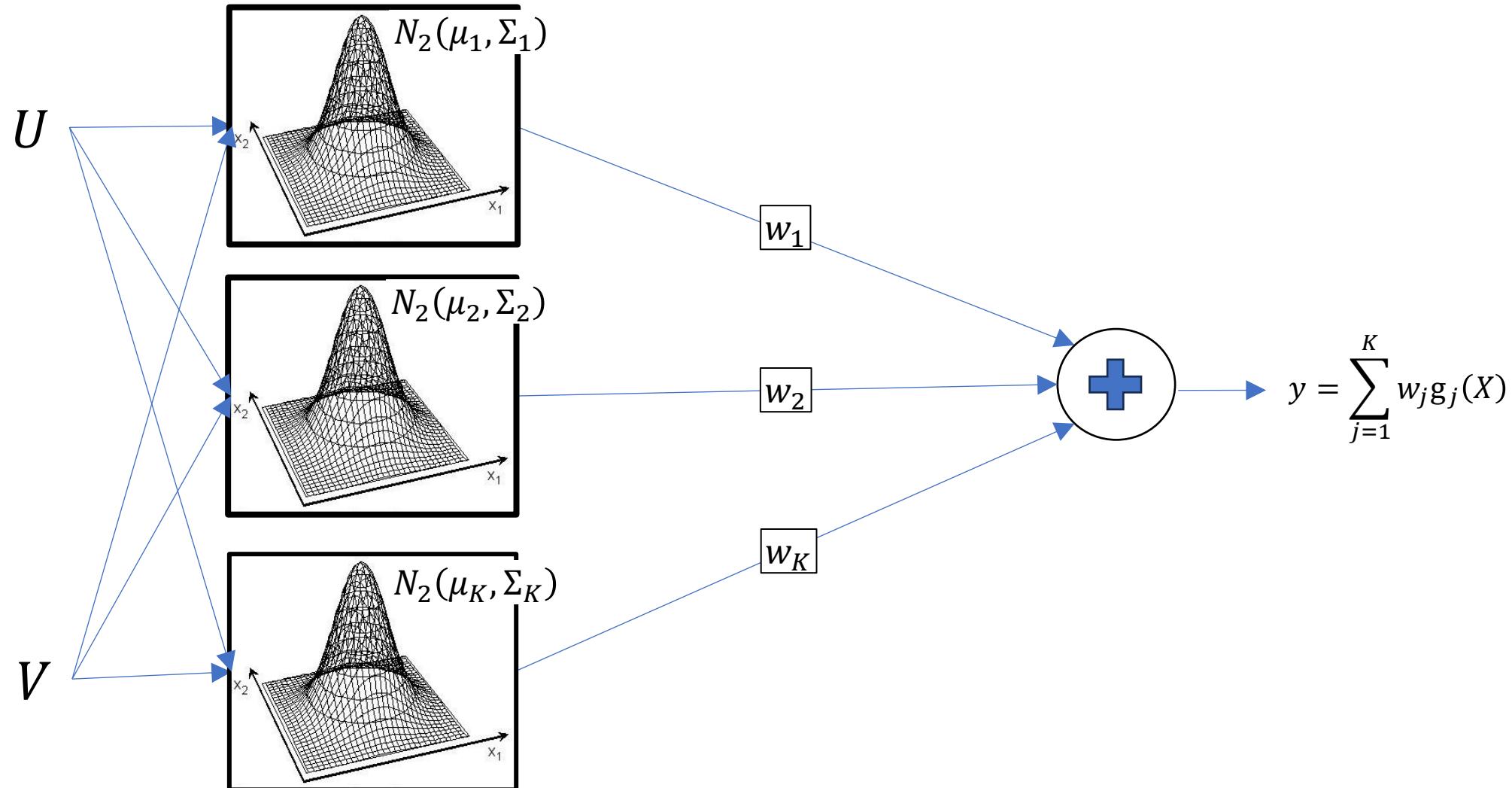
Componente V da velocidade do vento e produção



$$\text{Produção} = f(U, V)$$



Introdução da direção do vento na curva de potência da geração eólica



Introdução da direção do vento na curva de potência da geração eólica

Conjunto de treinamento formado por 500 exemplos



	U	V	Produção (Y)
	2.5286435	1.519362	0.005847727
	3.6994636	1.494680	0.027026136
	-0.5445933	5.181459	0.133814205
	3.1917530	3.544801	0.046067045
	-5.1500177	2.624065	0.067867045
	2.5280838	-10.139590	0.366028977
	-1.8302469	5.986468	0.080525000
	-6.6014288	7.867276	0.393845455
	-0.9210488	4.333194	0.001781250
	0.8712584	12.459575	0.606994318

xtrain

ytrain

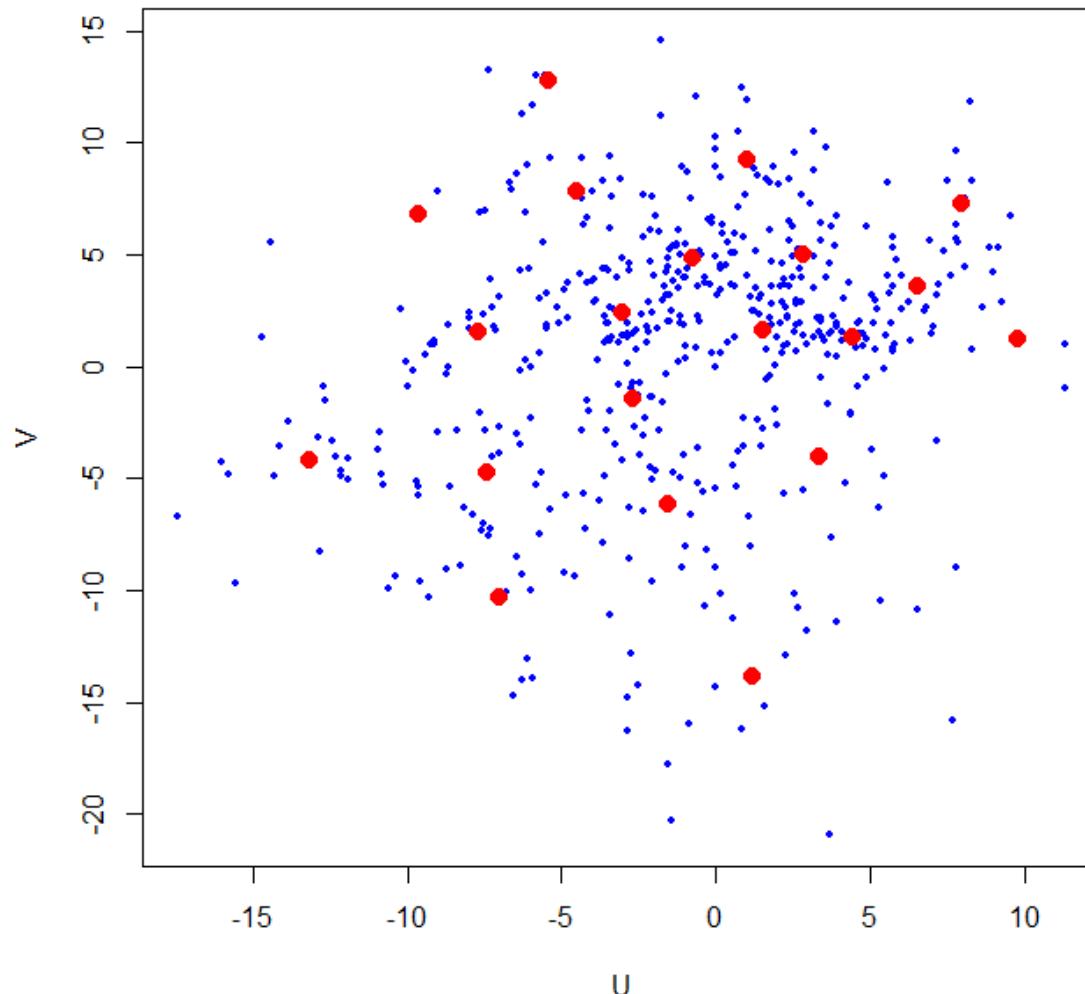
Introdução da direção do vento na curva de potência da geração eólica

500 exemplos de treinamento

Centroides (μ) dos 20 clusters
identificados pelo K-Means

	U	V
1	-7.3955242	-4.716588
2	9.7783295	1.242591
3	6.5230128	3.596768
4	1.5132041	1.632928
5	-1.5281472	-6.128467
6	-5.4416012	12.798555
7	-0.7242383	4.849191
8	-13.1378013	-4.177583
9	-7.0068999	-10.297100
10	-3.0064396	2.411146
11	-9.6135331	6.786963
12	4.4192822	1.289376
13	1.0342729	9.270349
14	-7.7090439	1.529768
15	7.9704430	7.257606
16	-4.4973780	7.805598
17	3.3446898	-4.007371
18	2.8423315	4.972383
19	-2.6902724	-1.472040
20	1.1666161	-13.871615

```
set.seed(0)  
kmeans_result <- kmeans(xtrain, centers = 20)  
centers <- kmeans_result$centers
```



Introdução da direção do vento na curva de potência da geração eólica

Neste caso considerou-se uma mesma matriz de covariância diagonal (Σ) para todos os *clusters*

$$\Sigma = \begin{bmatrix} \sigma^2 & & \\ & \ddots & \\ & & \sigma^2 \end{bmatrix}$$

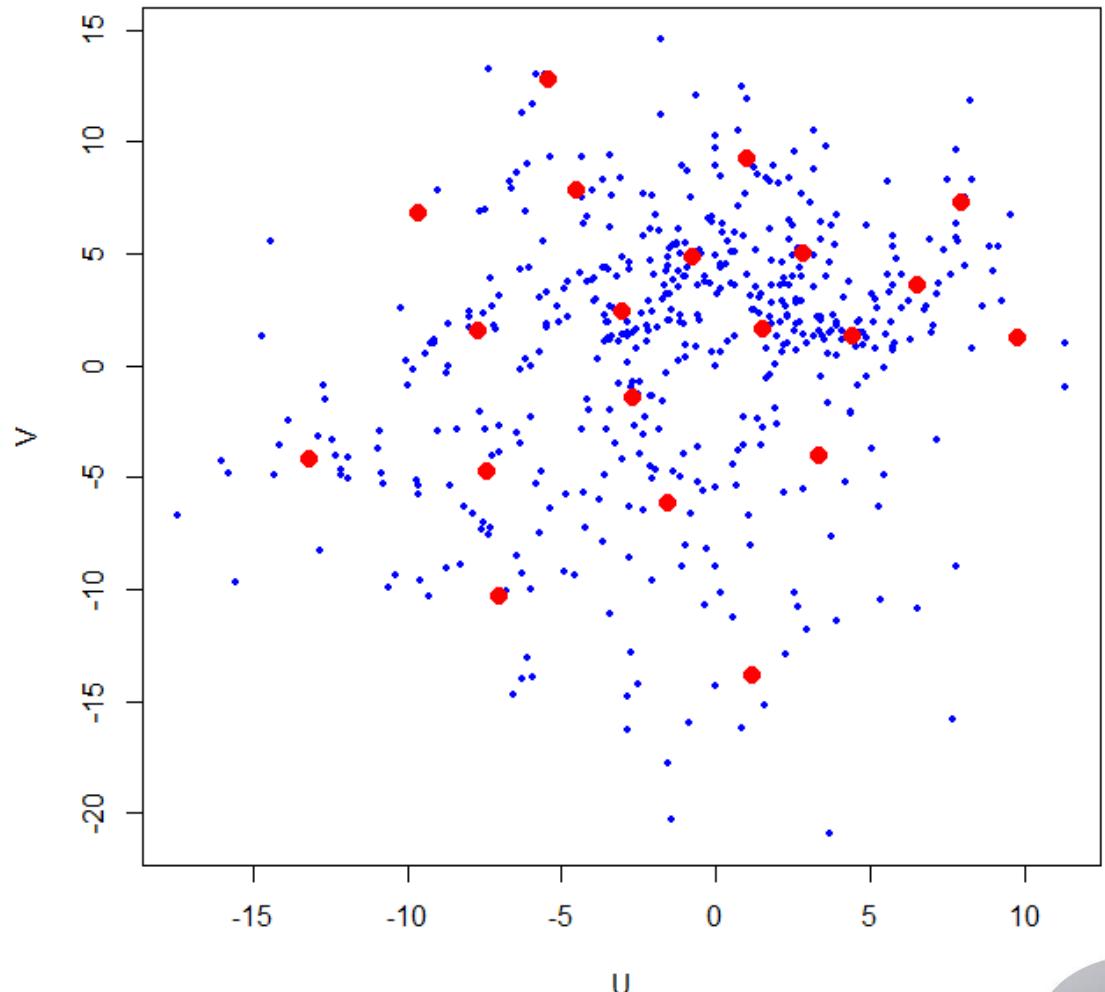
```
library(stats)
```

```
d_max <- max(as.matrix(dist(centers)))
```

```
sigma <- d_max / sqrt(2 * nrow(centers))
```

$$\sigma = \frac{d_{max}}{\sqrt{2 \times 20}}$$

Matriz de distâncias entre os centroides



Introdução da direção do vento na curva de potência da geração eólica

Define a função de base radial

```
rbf <- function(x, c, s) {exp(-sum((x - c)^2) / (2 * s^2))}
```

$$\exp \left[-\frac{1}{2} (U - \mu_1 \quad V - \mu_2) \begin{bmatrix} \sigma^2 & 0 \\ 0 & \sigma^2 \end{bmatrix}^{-1} \begin{pmatrix} U - \mu_1 \\ V - \mu_2 \end{pmatrix} \right]$$
$$\exp \left[-\left(\frac{(U - \mu_1)^2}{2\sigma^2} + \frac{(V - \mu_2)^2}{2\sigma^2} \right) \right]$$

Cálculo das saídas dos neurônios da camada oculta

```
G <- matrix(0, nrow = nrow(xtrain), ncol = nrow(centers))
```

Inicializa matriz G de dimensões 500 x 20

```
for (i in 1:nrow(xtrain)) {  
  for (j in 1:nrow(centers)) {  
    G[i, j] <- rbf(xtrain[i, ], centers[j, ], sigma)  
  }  
}
```

Saída do neurônio j para o i-ésimo exemplo de treinamento

Cálculo dos pesos da camada de saída

```
library(MASS)  
W <- ginv(G) %*% ytrain
```

$$y_{train_i} = \sum_{j=1}^{20} w_j g_j(x_{train_i}) \quad \forall i=1,500$$
$$Y_{500 \times 1} = G_{500 \times 20} W_{20 \times 1} \rightarrow W = \boxed{(G^T G)^{-1} G^T Y}$$

ginv(R)

Introdução da direção do vento na curva de potência da geração eólica

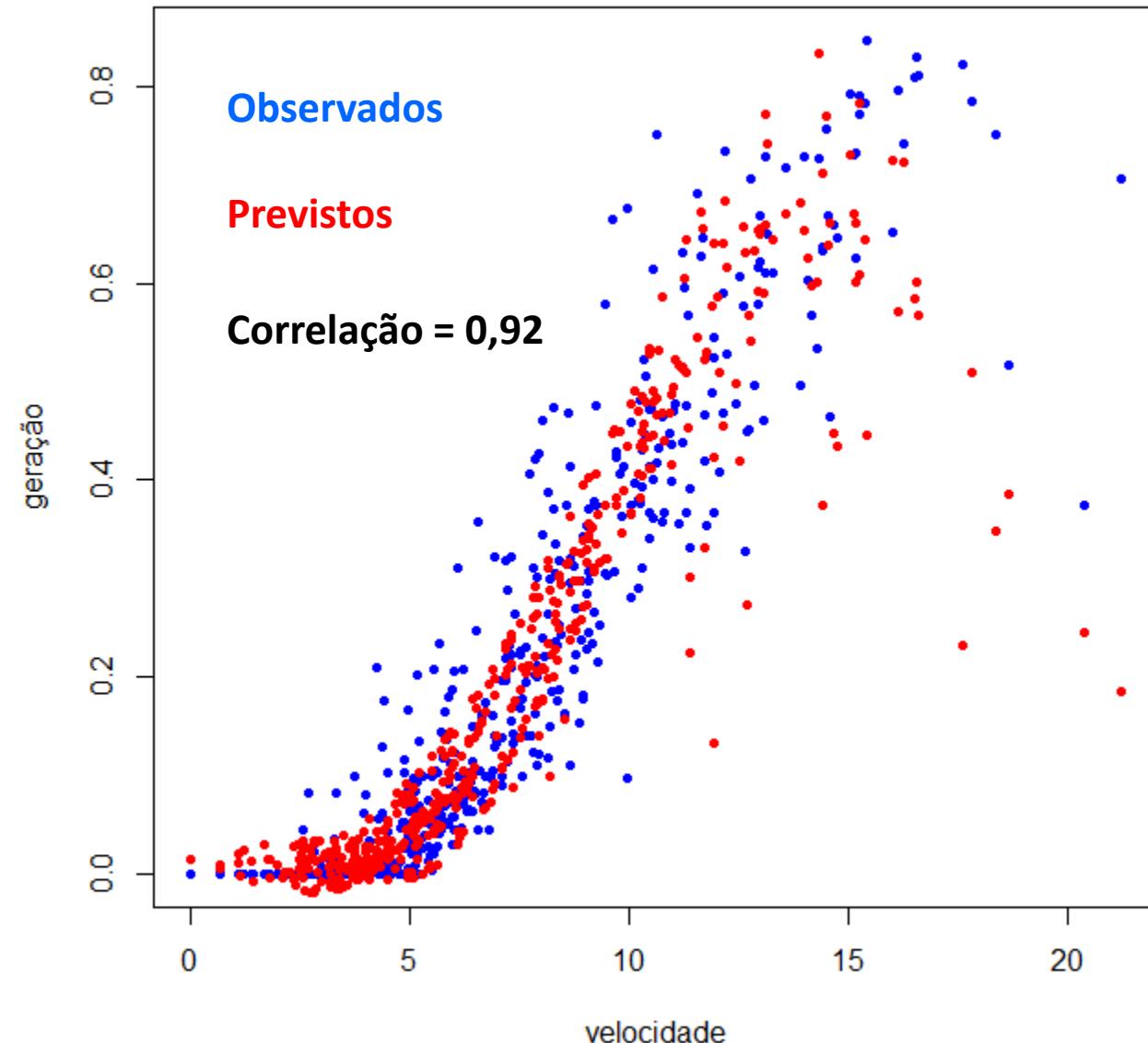
Calcula resposta da rede RBF já treinada para um ou mais vetores de entrada X

```
rbf_network <- function(X, centers, sigma, W) {  
  G <- matrix(0, nrow = nrow(X), ncol = nrow(centers))  
  for (i in 1:nrow(X)) {  
    for (j in 1:nrow(centers)) {  
      G[i, j] <- rbf(X[i, ], centers[j, ], sigma)  
    }  
  }  
  return(G %*% W)
```

$$y = \sum_{j=1}^K w_j g_j(X)$$

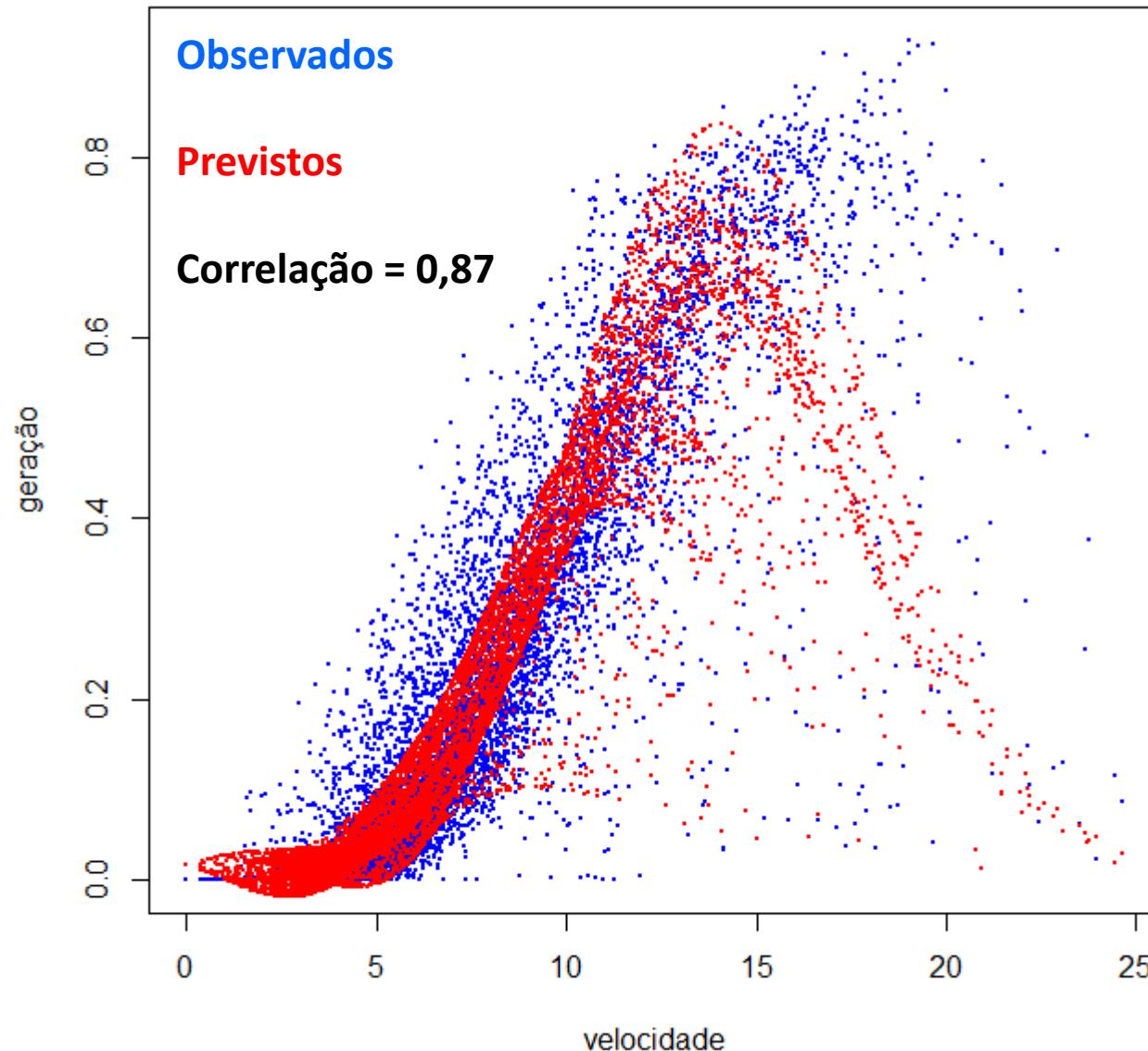
Calcula a previsão para o conjunto de treinamento

```
y_pred <- rbf_network(xtrain, centers, sigma, W)
```



Introdução da direção do vento na curva de potência da geração eólica

Resultado para todo conjunto de dados



Redes Neurais Recorrentes (RNN)

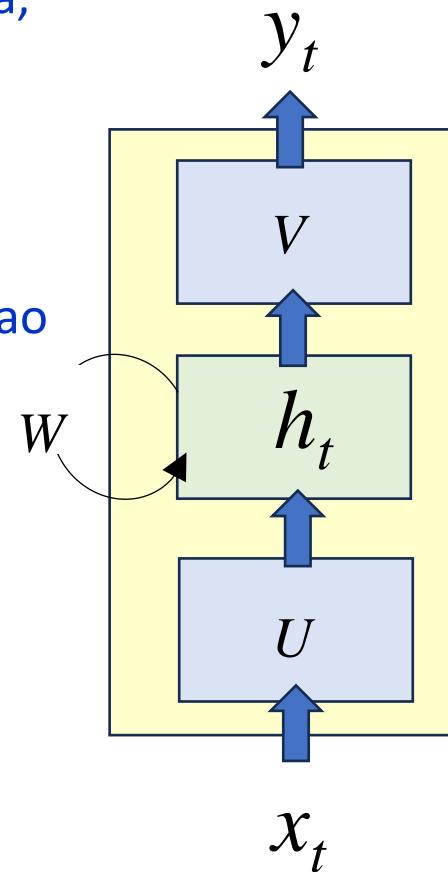
- O tipo mais comum de rede neural artificial projetado para lidar com dados de séries temporais é a Rede Neural Recorrente (RNN) (GERRISH, 2019).
- Em uma RNN, a saída em um dado instante depende de elementos anteriores na sequência, refletindo as dependências temporais dos dados.
- Unidades recorrentes mantêm um estado oculto (h) que armazena informações sobre entradas anteriores em uma sequência. Essas unidades podem "lembrar" informações de etapas anteriores ao retroalimentar seu estado oculto, permitindo capturar dependências ao longo do tempo
- A RNN considera a dependência sequencial entre os elementos ao processar informações.

$$h_t = f(W \cdot h_{t-1} + U \cdot x_t)$$

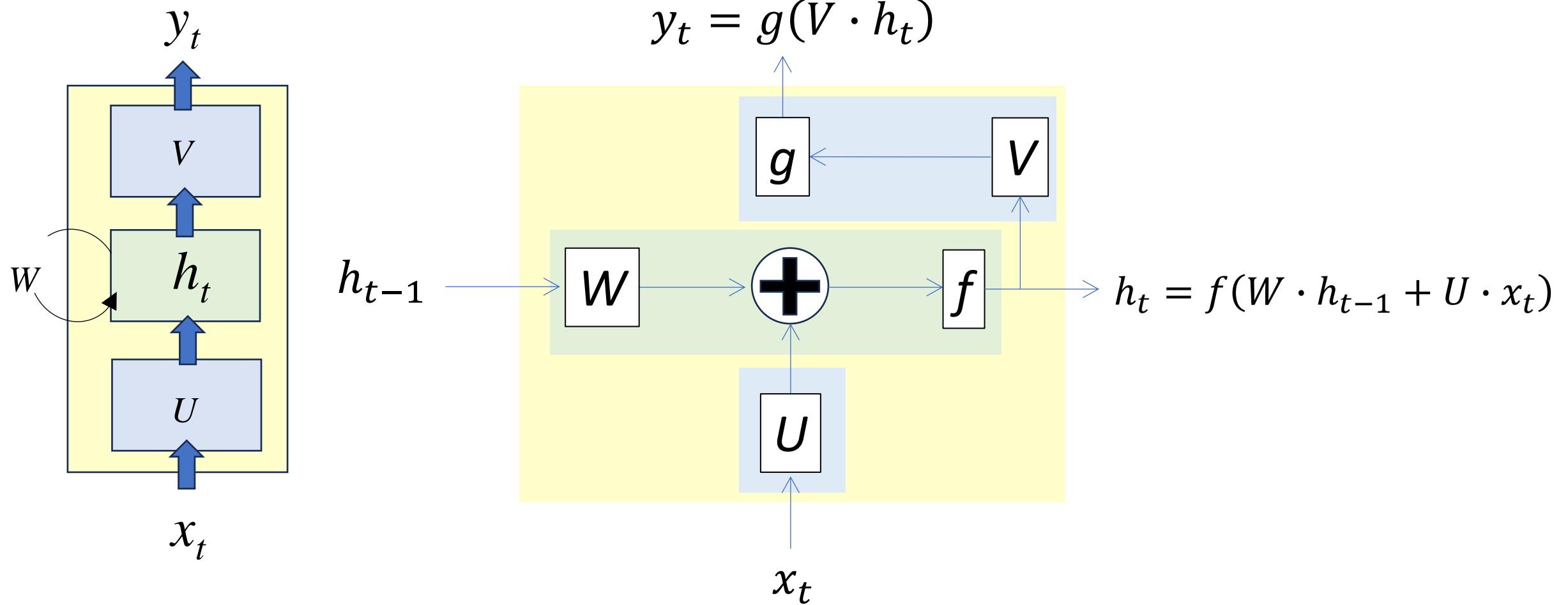
$$y_t = g(V \cdot h_t)$$

A recorrência dos estados latentes h permite que a RNN capture a estrutura de dependência temporal presente em séries temporais

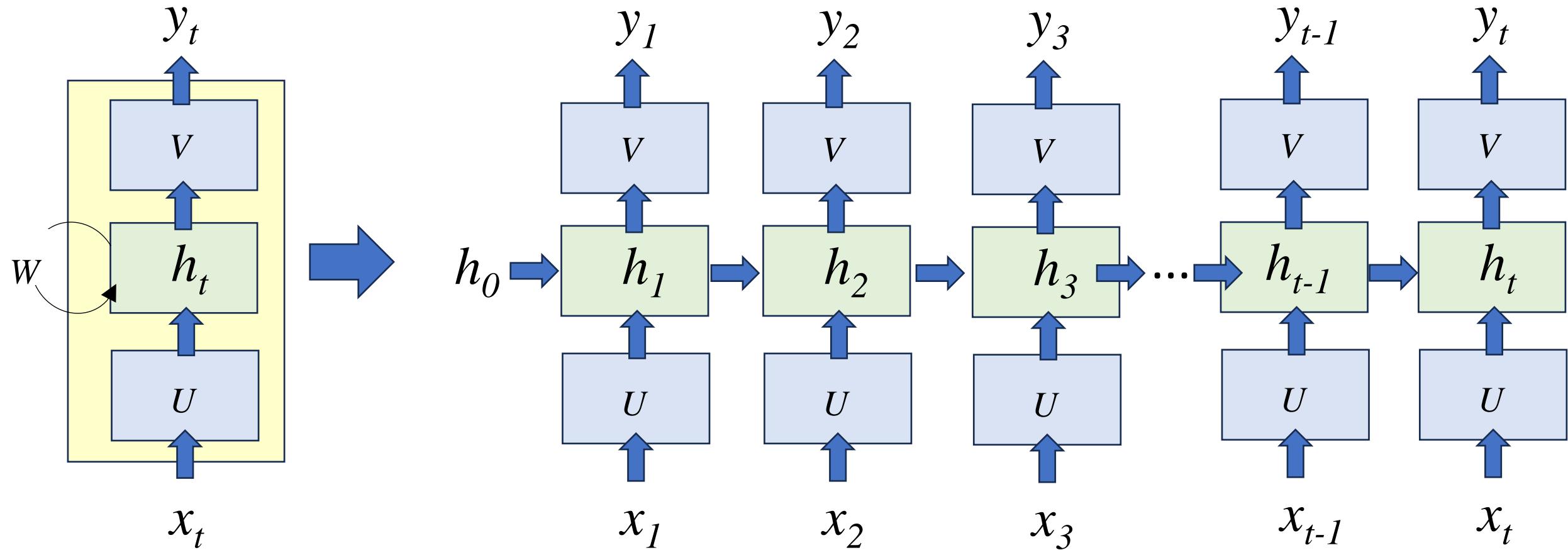
- Em uma RNN, o compartilhamento de pesos (U, V, W) garante que os mesmos parâmetros sejam aplicados a cada passo da sequência, preservando a estrutura temporal dos dados



Redes Neurais Recorrentes (RNN)

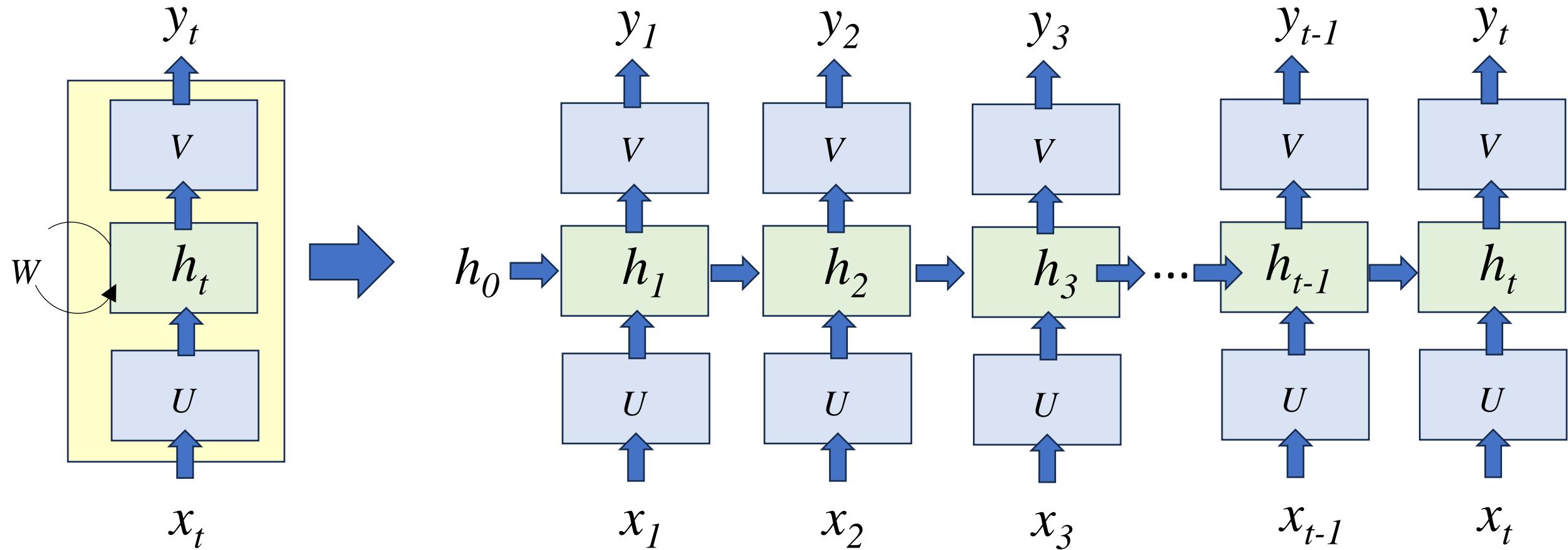


Desdobrando a RNN (*Unfolding*)



A cada intervalo de tempo t o estado e a saída da rede são atualizados

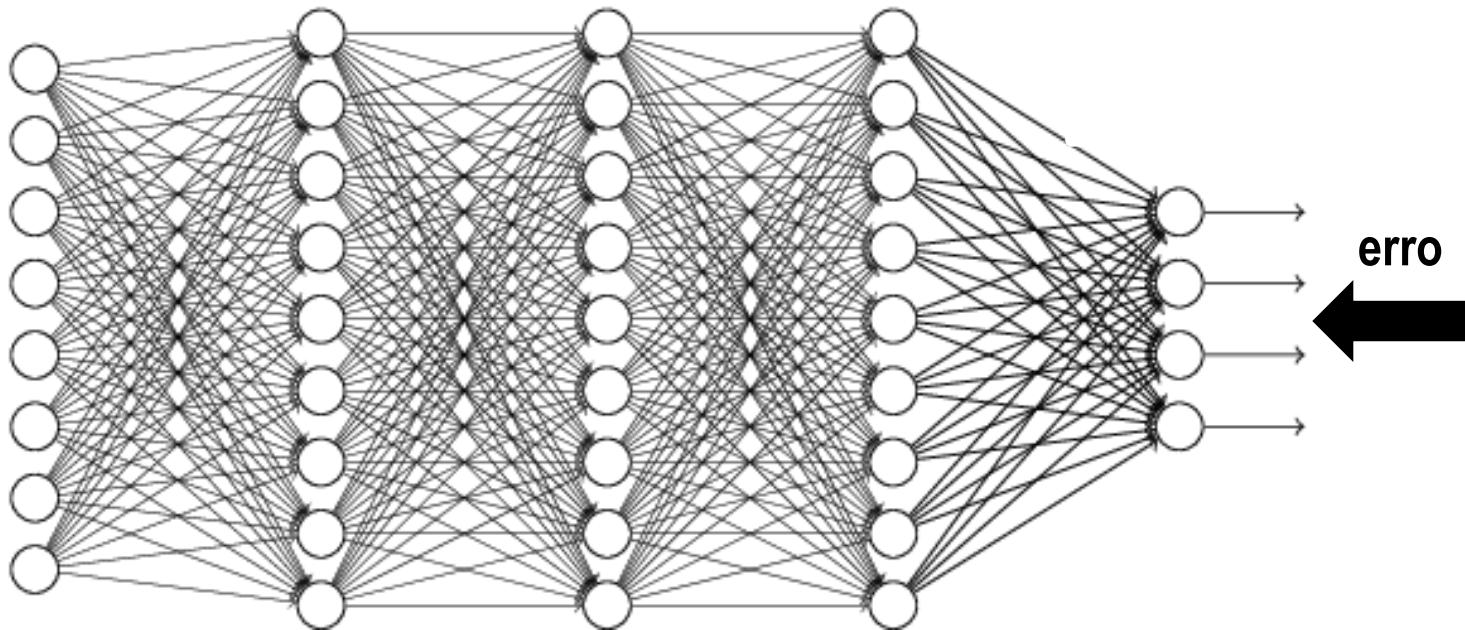
Desdobrando a RNN (*Unfolding*)



Treinamento da rede por retropropagação do erro através do tempo (a retropropagação é feita em cada ponto do tempo), mas para rede com muitos camadas o gradiente pode desaparecer ou explodir.

Isso ocorre porque é difícil capturar dependências de longo prazo devido ao gradiente multiplicativo, que pode diminuir ou aumentar exponencialmente em relação ao número de camadas.

Problema do desaparecimento do gradiente (1991) (*Vanishing Gradient Problem*)



Sepp Hochreiter
1967

O sinal do erro torna-se cada vez menor a medida que é retropropagado

Problema frequente em redes profundas ou com muitas camadas

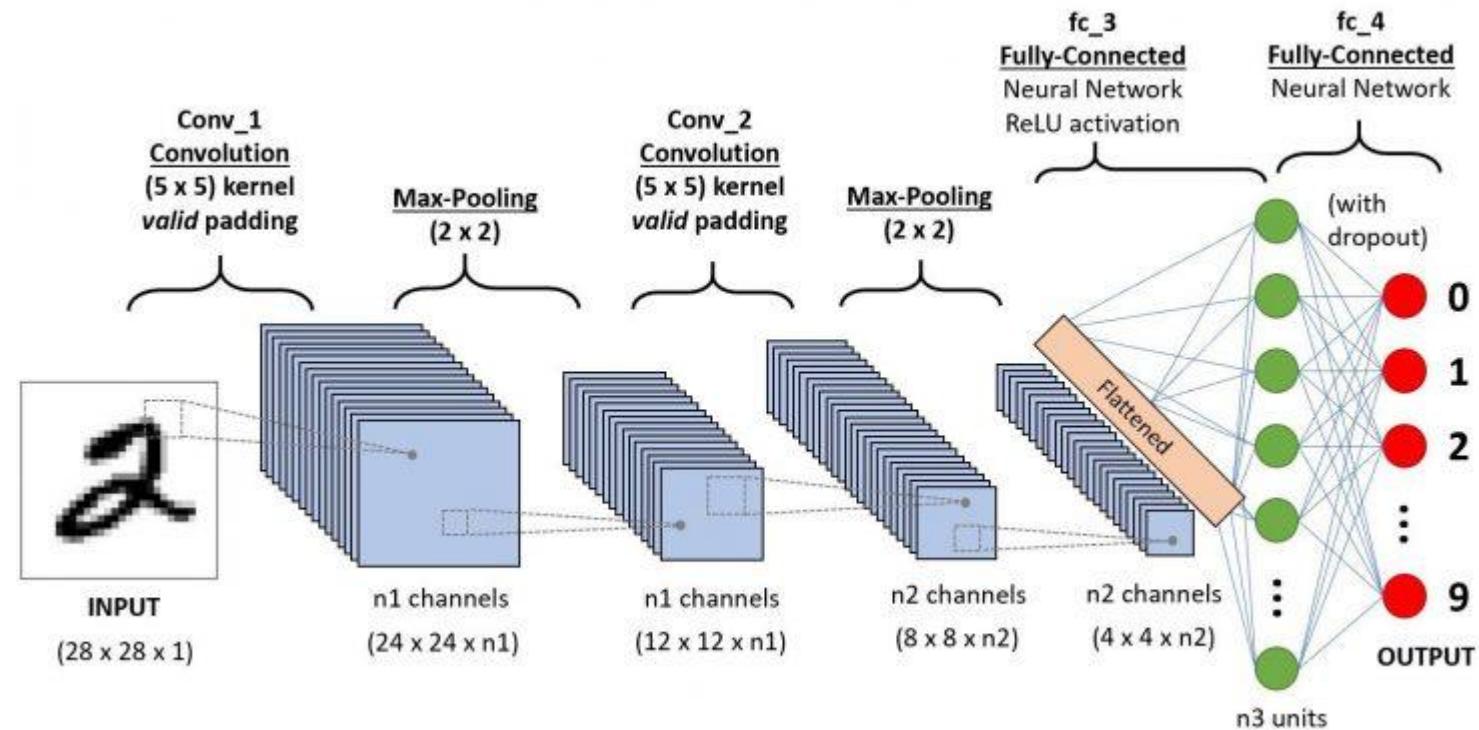
Redes convolucionais – CNN (1999)



Yann LeCun 1960

Conjunto de treinamento

000000000000000000
111111111111111111
222222222222222222
333333333333333333
444444444444444444
555555555555555555
666666666666666666
777777777777777777
888888888888888888
999999999999999999



<https://datasciencepr.com/convolutional-neural-network/>

Soluções para o problema do desaparecimento do gradiente

Soluções (Kelleher, 2019):

- **Inicialização dos pesos (Glorot initialization 2010):** [glorot10a.pdf](#)
 - O desaparecimento do gradiente é mitigado quando as variâncias dos gradientes são semelhantes em cada uma das camadas da rede.
 - As variâncias dos gradientes são relacionadas com as variâncias dos pesos.
 - A inicialização de Glorot é desenhada de tal forma que todas as camadas terão variâncias semelhantes em termos de ativações na fase *forward* e dos gradientes na fase *backward*.
 - Regra heurística de inicialização dos pesos para atender este objetivo.

$$w \sim U\left(-\frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}\right)$$

pesos entre as camadas j e $j+1$
 n_j é o número de neurônios na camada j

Understanding the difficulty of training deep feedforward neural networks

Xavier Glorot
DIRO, Université de Montréal, Montréal, Québec, Canada

Yoshua Bengio
DIRO, Université de Montréal, Montréal, Québec, Canada

- **Mudança da função de ativação para ReLus (Rectified Linear Units) e suas variants:**
 - Funções sigmoide e tanh exacerbam o problema do desaparecimento do gradiente

Deep Sparse Rectifier Neural Networks

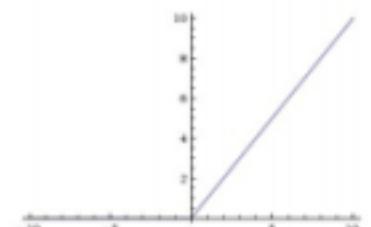
Xavier Glorot
DIRO, Université de Montréal
Montréal, QC, Canada
glorotxa@iro.umontreal.ca

Antoine Bordes
Heudiasyc, UMR CNRS 6599
UTC, Compiègne, France
and
DIRO, Université de Montréal
Montréal, QC, Canada
antoine.bordes@hds.utc.fr

Yoshua Bengio
DIRO, Université de Montréal
Montréal, QC, Canada
bengioy@iro.umontreal.ca

[glorot11a.pdf](#)

ReLU $\max(0, x)$



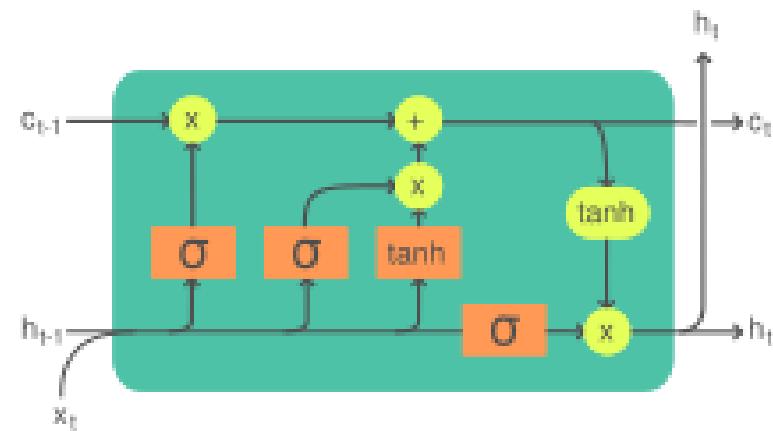
Redes LSTM (1997)



Sepp Hochreiter
1967



Jürgen Schmidhuber
1963



A célula LSTM pode processar dados sequencialmente e manter seu estado oculto ao longo do tempo

Legend:

Layer Componentwise Copy Concatenate

LONG SHORT-TERM MEMORY

NEURAL COMPUTATION 9(8):1735–1780, 1997

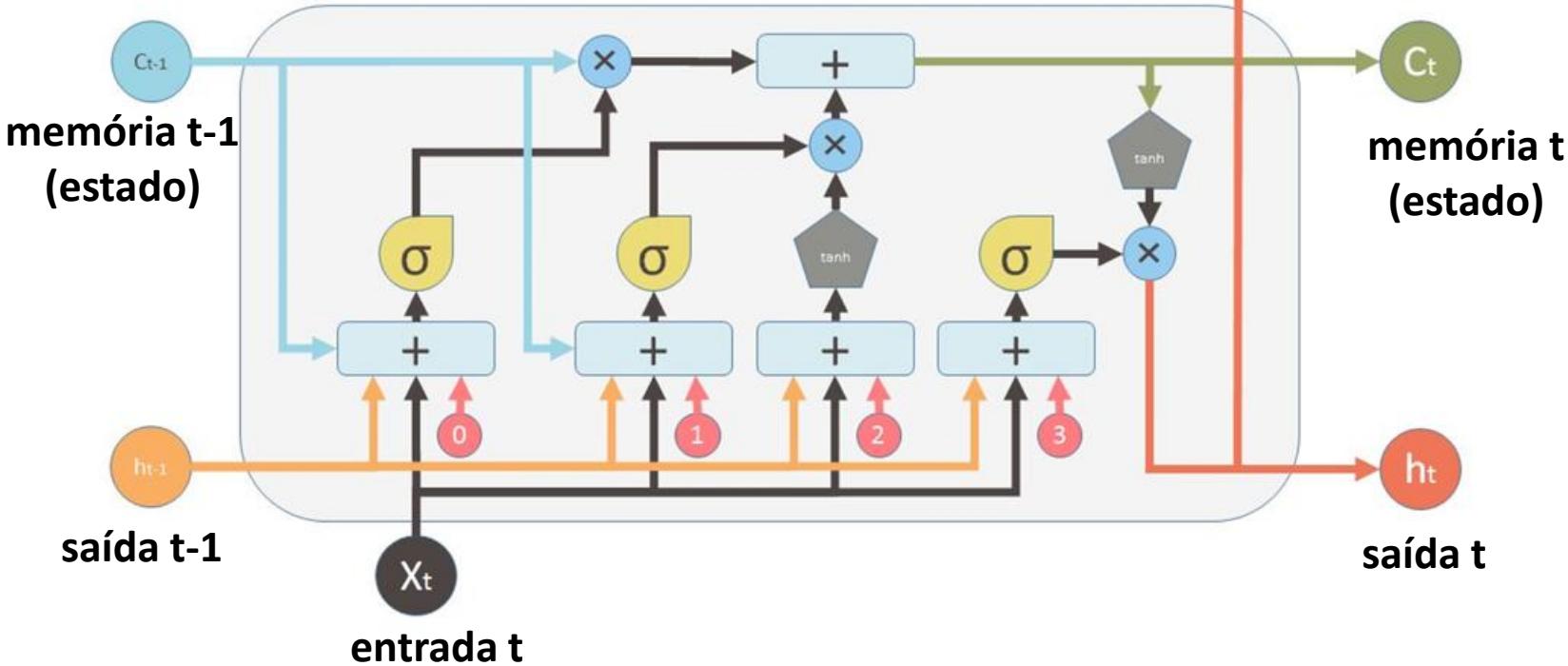
Sepp Hochreiter
Fakultät für Informatik
Technische Universität München
80290 München, Germany
hochreit@informatik.tu-muenchen.de
<http://www7.informatik.tu-muenchen.de/~hochreit>

<https://www.bioinf.jku.at/publications/older/2604.pdf>

Jürgen Schmidhuber
IDSIA
Corso Elvezia 36
6900 Lugano, Switzerland
juergen@idsia.ch
<http://www.idsia.ch/~juergen>

- Evolução das redes neurais recorrentes (RNN)
- Processa dados sequenciais
- Processamento de linguagem natural
- Tradução
- Legendagem de imagens
- Reconhecimento de escrita
- Conversão de vídeo para texto

Célula LSTM



Inputs:

X_t Input vector

C_{t-1} Memory from previous block

h_{t-1} Output of previous block

outputs:

C_t Memory from current block

h_t Output of current block

Nonlinearities:

σ Sigmoid

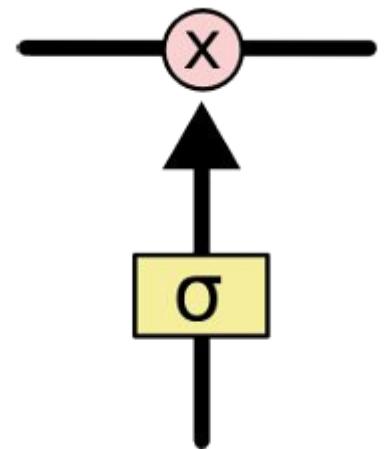
\tanh Hyperbolic tangent

Vector operations:

\times Element-wise multiplication

$+$ Element-wise Summation / Concatenation

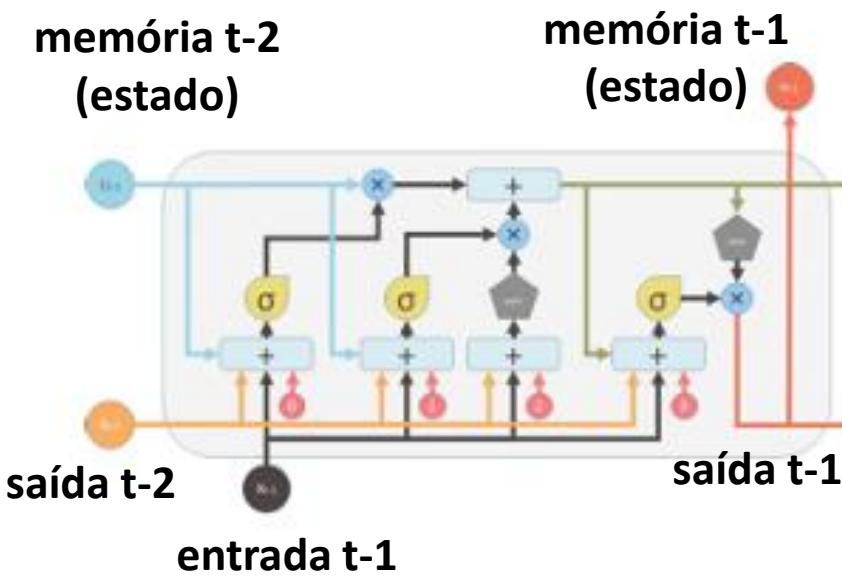
Bias: 0



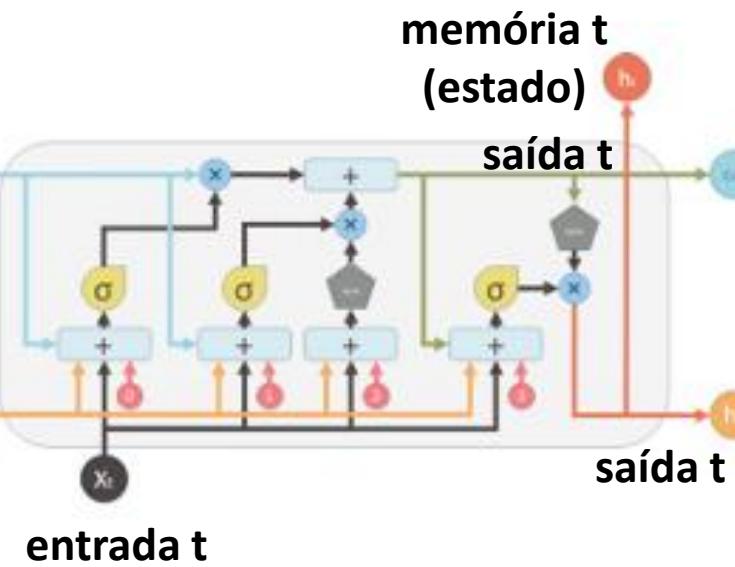
Gate controla o fluxo de informação da memória

Desdobramento da LSTM

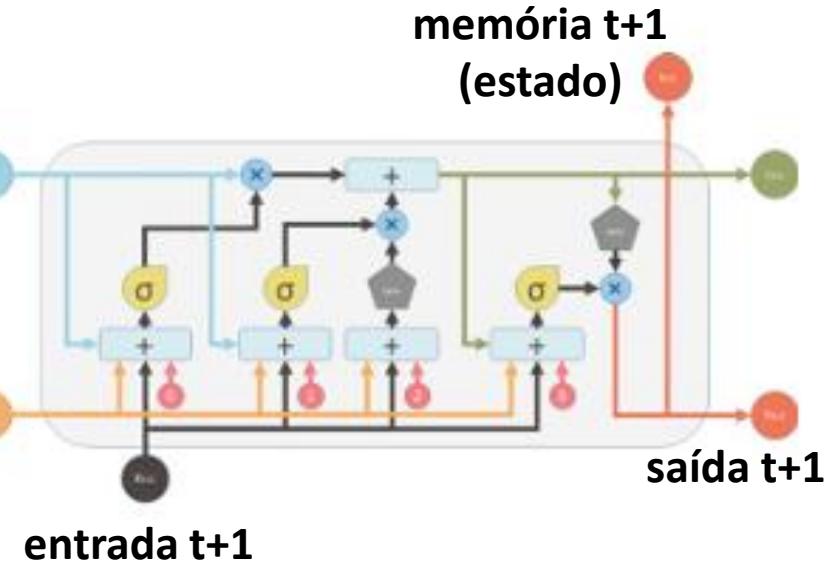
Tempo T-1



Tempo T

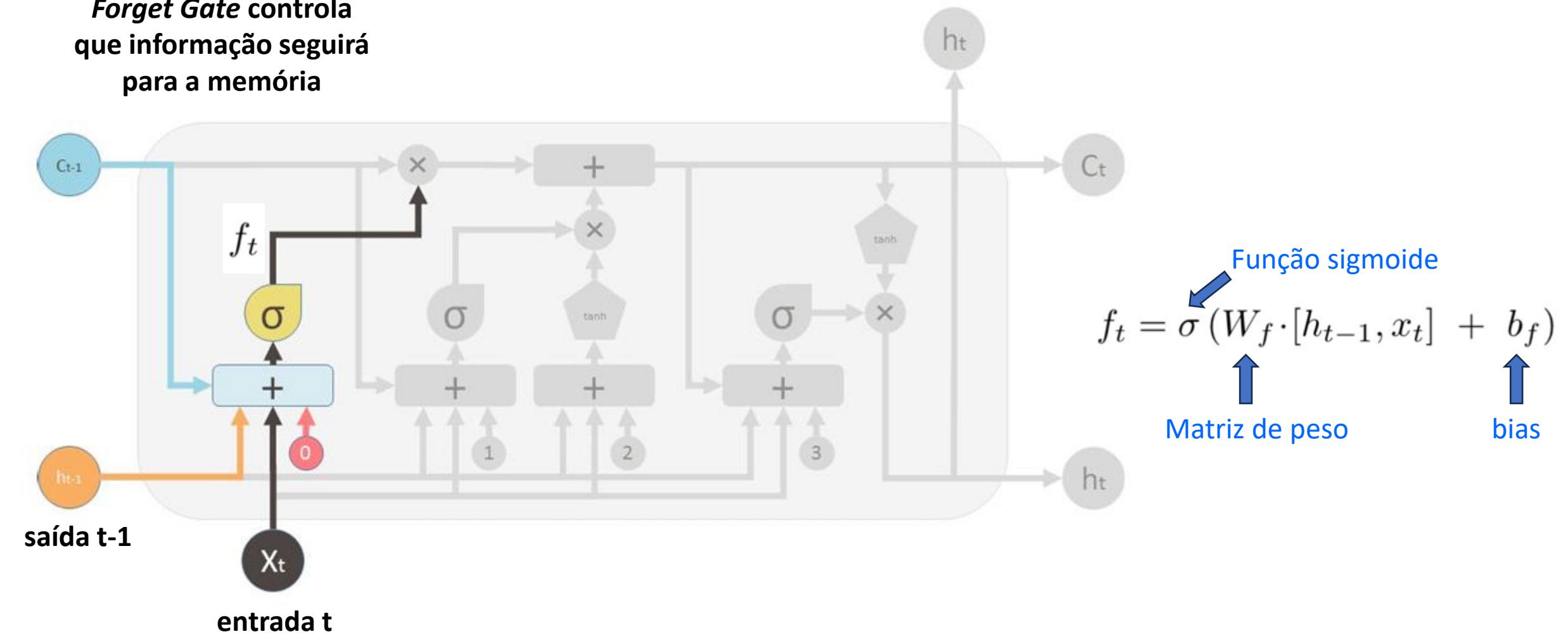


Tempo T+1



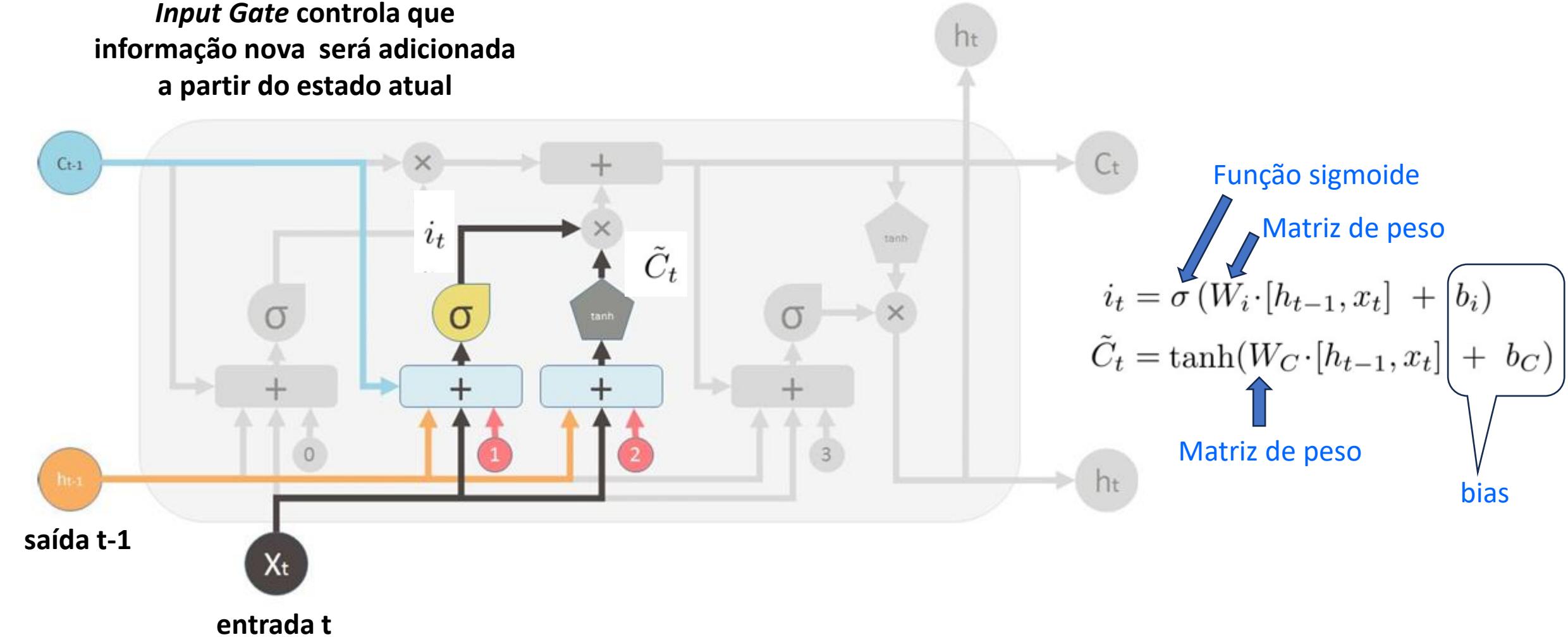
Célula LSTM – *Forget gate*

Forget Gate controla
que informação seguirá
para a memória



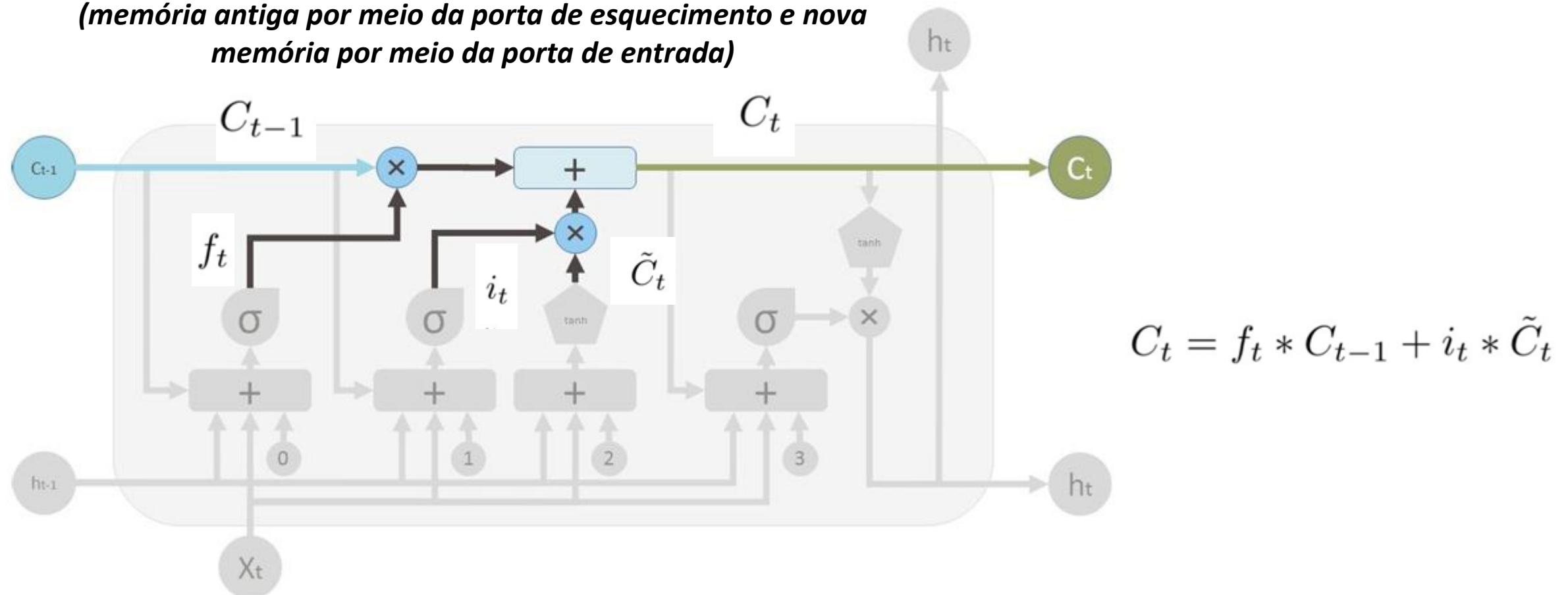
Célula LSTM – *Input gate*

Input Gate controla que informação nova será adicionada a partir do estado atual



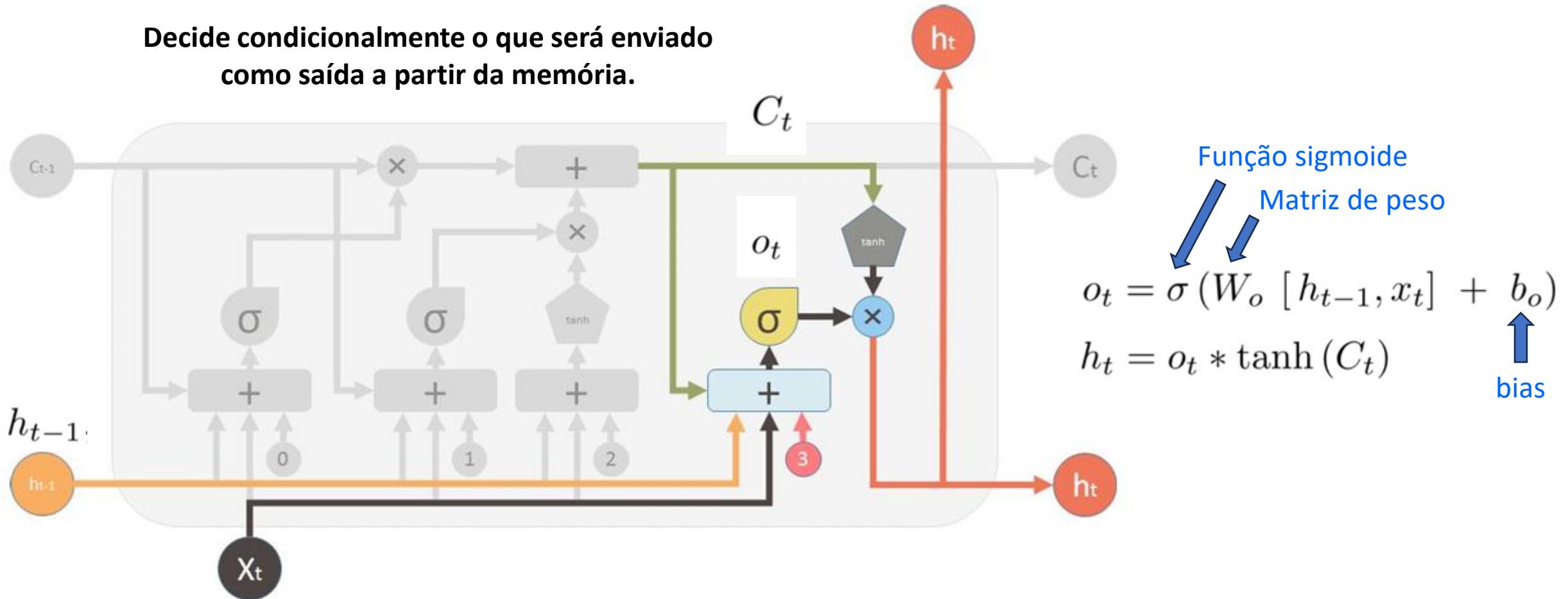
Célula LSTM – Atualiza a memória

O vetor de estado da célula agrupa os dois componentes
(memória antiga por meio da porta de esquecimento e nova
memória por meio da porta de entrada)

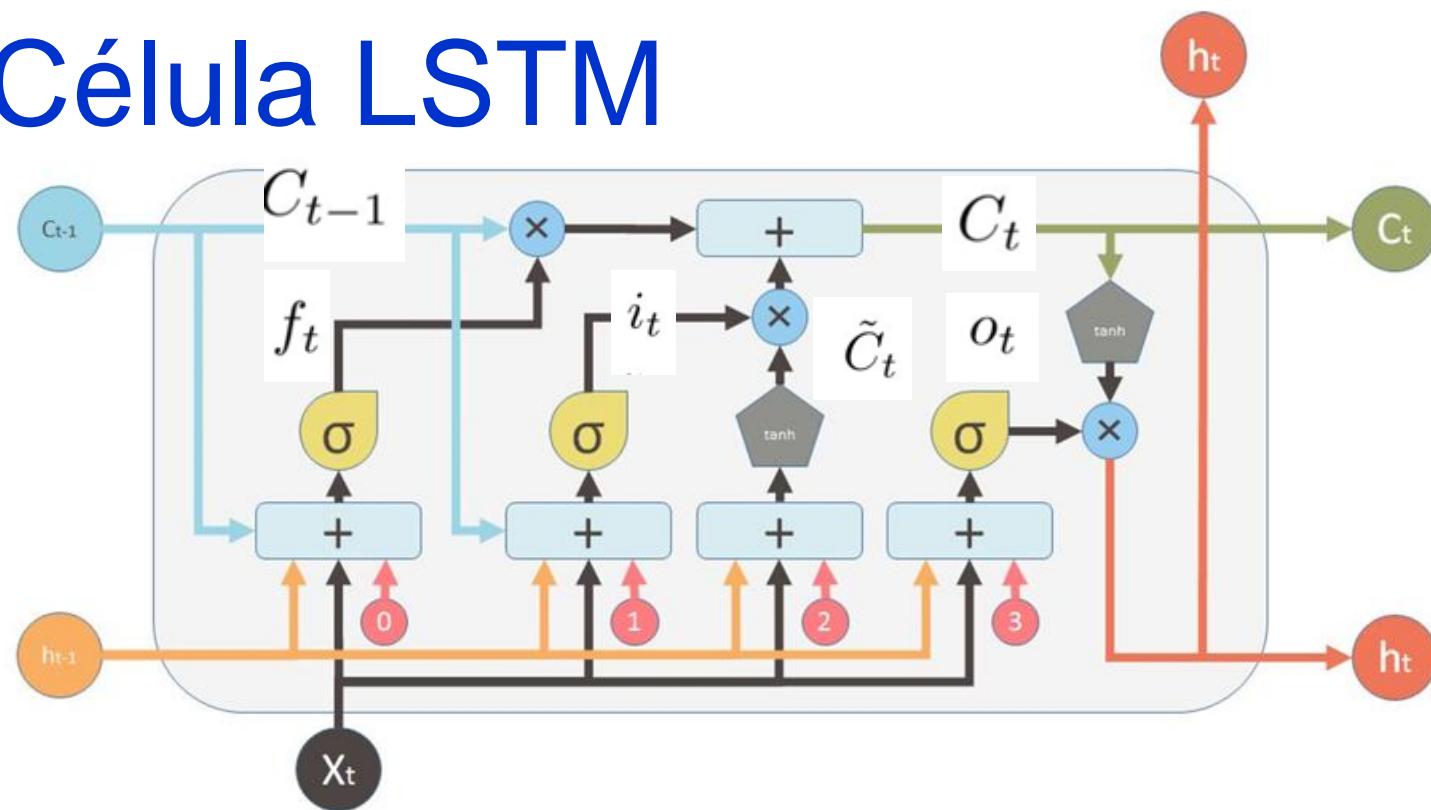


Célula LSTM – *Output Gate*

Decide condicionalmente o que será enviado como saída a partir da memória.



Célula LSTM



Inputs:

x_t Input vector

C_{t-1} Memory from previous block

h_{t-1} Output of previous block

outputs:

C_t Memory from current block

h_t Output of current block

Nonlinearities:

σ Sigmoid

\tanh Hyperbolic tangent

Vector operations:

\times

$+$

Element-wise multiplication

Element-wise Summation / Concatenation

Bias: 0

$$f_t = \sigma (W_f \cdot [h_{t-1}, x_t] + b_f)$$

$$i_t = \sigma (W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh (W_C \cdot [h_{t-1}, x_t] + b_C)$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

$$o_t = \sigma (W_o \cdot [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh (C_t)$$

[LSTM for Production: Part 1](#)

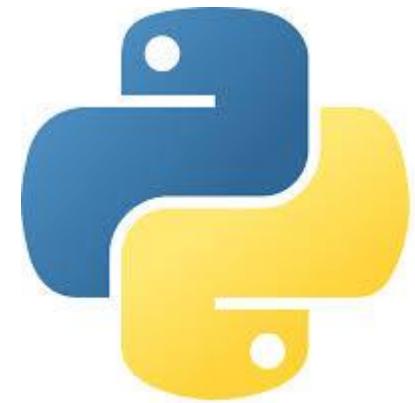
Treinamento da LSTM

Retropropagação do erro através do tempo

Que pesos são aprendidos?

- matrizes W e *bias* b no *input*, *output* e *forget gates*
- *Input* da camada com *tanh*

Implementação da LSTM em Python



Untitled4.ipynb ★

Arquivo Editar Ver Inserir Ambiente de execução

+ Código + Texto

[5] `import numpy as np
import tensorflow as tf
from tensorflow import keras
import pandas as pd
import seaborn as sns
from pylab import rcParams
import matplotlib.pyplot as plt
from google.colab import drive`

[7] `drive.mount('/content/drive')` **Leitura do arquivo no Google drive**

Mounted at /content/drive

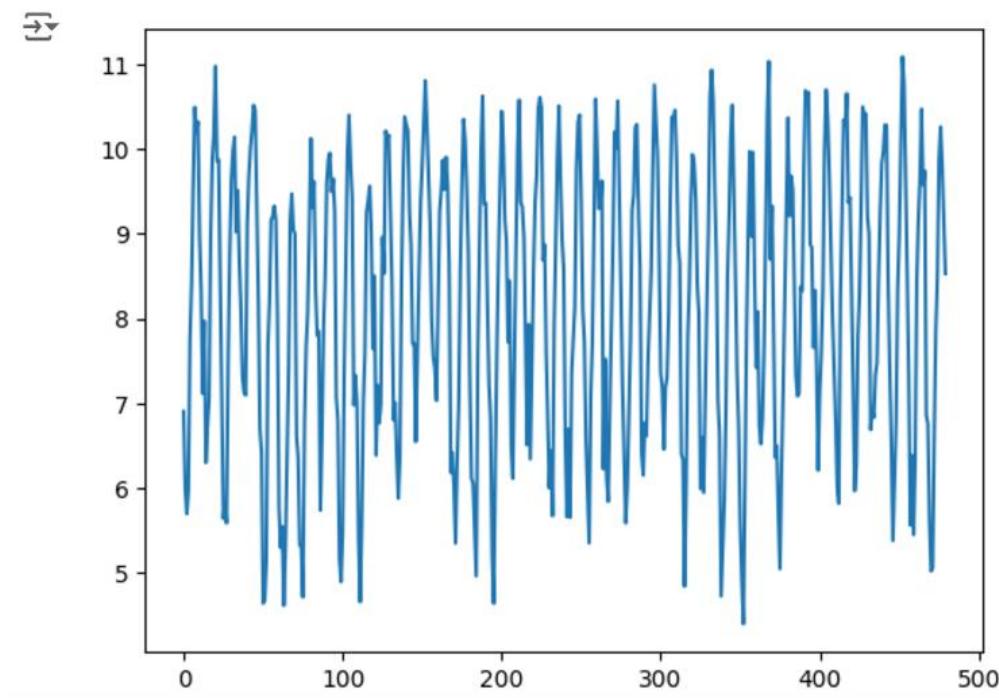
[8] `file_path = '/content/drive/My Drive/AcarauII69_mensal_MERRA.csv'
df = pd.read_csv(file_path,sep=";")`

[] `sns.set(style='whitegrid', palette='muted', font_scale=1.5)
RANDOM_SEED = 42
np.random.seed(RANDOM_SEED)
tf.random.set_seed(RANDOM_SEED)`

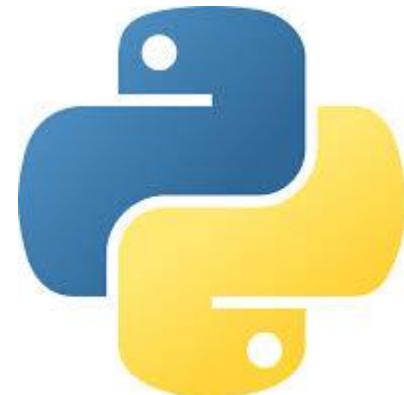
Exemplo da previsão da média mensal da velocidade do vento em Acaraú II 230 kV
Código disponível em

[Time Series Forecasting Using LSTM - Scaler Topics](#)

▶ `data_time = np.arange(0,480, 1)
ventos_values = df.vento
plt.plot(data_time, ventos_values, label='vento');`



Implementação da LSTM em Python



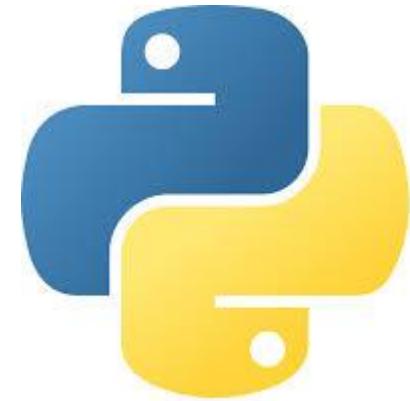
```
[43] data_full = pd.DataFrame(dict(vento=ventos_values), index=data_time, columns=['vento'])  
data_full.head()  
  
len_train = int(len(data_full) * 0.90)      Separa em treinamento e teste  
len_test = len(data_full) - len_train  
train, test = data_full.iloc[0:len_train], data_full.iloc[len_train:len(data_full)]
```



```
def gen_data(X, y, num_steps=1):      Monta padrões de entrada X (valores passados) e saída y  
    Xs, ys = [], []  
    for i in range(len(X) - num_steps):  
        Xs.append(X.iloc[i:(i + num_steps)].values)  
        ys.append(y.iloc[i + num_steps])  
    return np.array(Xs), np.array(ys)
```

```
num_steps = 12  Usa os últimos 12 meses para prever o próximo mes  
trainX, trainY = gen_data(train, train.vento, num_steps)  
testX, testY = gen_data(test, test.vento, num_steps)
```

Implementação da LSTM em Python



```
[48] lstm_model = keras.Sequential()  
      lstm_model.add(keras.layers.LSTM(128, input_shape=(trainX.shape[1], trainX.shape[2])))  
      lstm_model.add(keras.layers.Dense(1))  
      lstm_model.compile(loss='mean_squared_error', optimizer=keras.optimizers.Adam(0.001))
```

Configura treinamento da rede LSTM

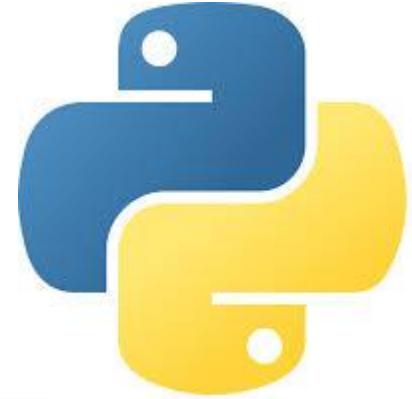


```
callbacks=[tf.keras.callbacks.EarlyStopping(monitor='loss', patience=3)]
```

```
history = lstm_model.fit(  
    trainX, trainY,  
    epochs=30,  
    batch_size=16,  
    validation_split=0.1,  
    shuffle=False,  
    callbacks=[tf.keras.callbacks.EarlyStopping(monitor='loss', patience=3)],  
)
```

Treinamento da rede LSTM

Implementação da LSTM em Python



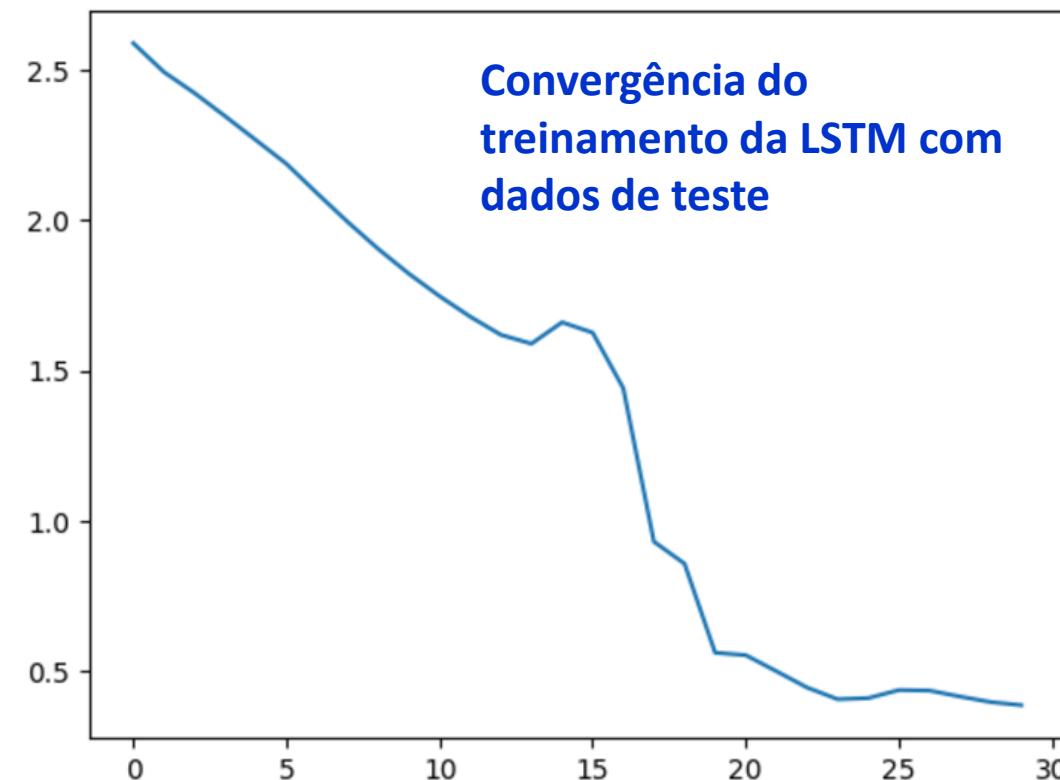
```
plt.plot(history.history['loss'], label='train')
```

```
[<matplotlib.lines.Line2D at 0x7a69595abc8e0>]
```

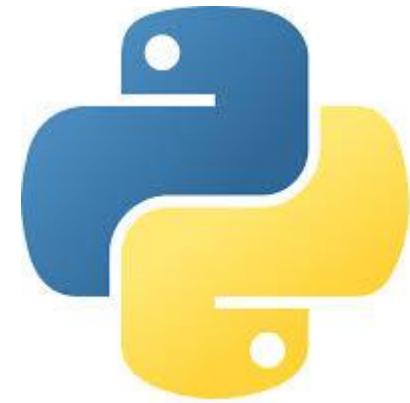


```
[38] plt.plot(history.history['val_loss'], label='test')
```

```
[<matplotlib.lines.Line2D at 0x7a69599034f0>]
```

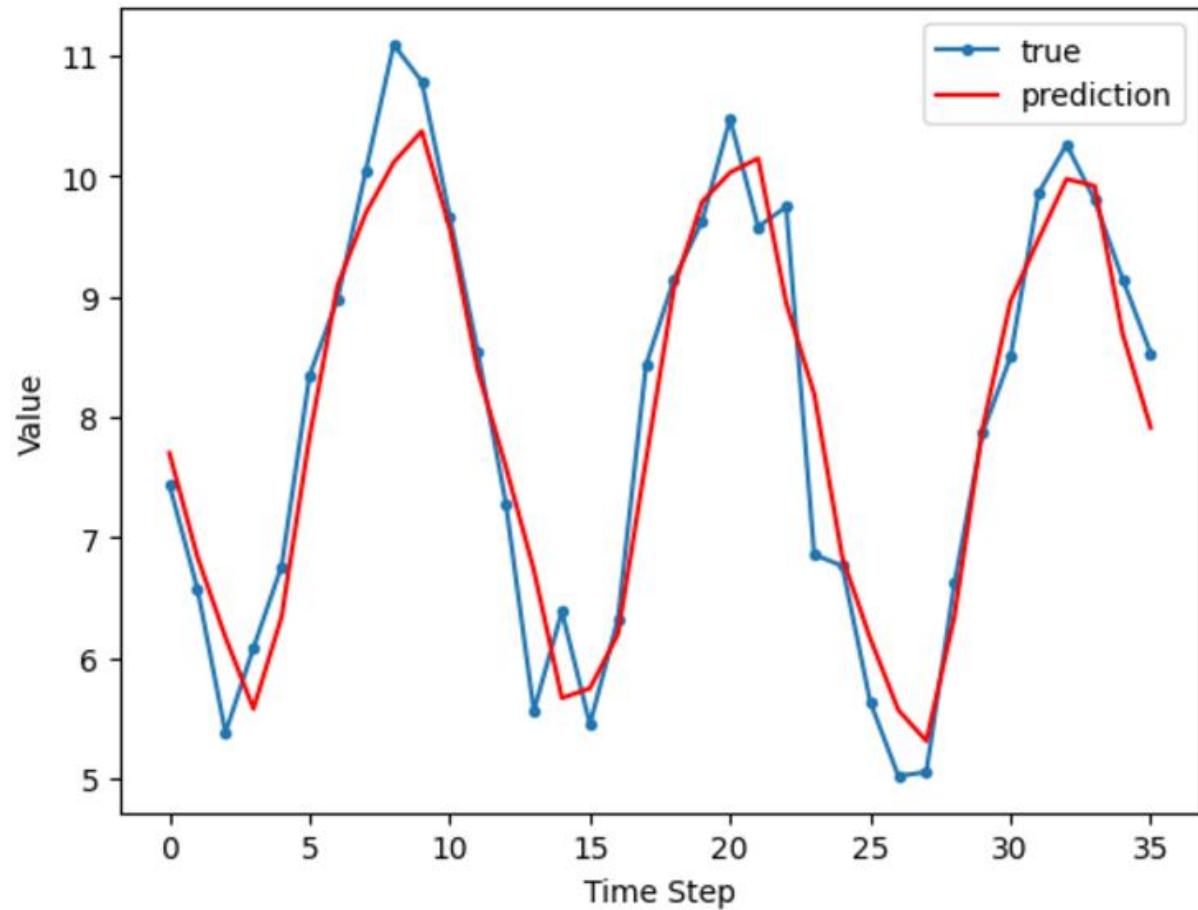


Implementação da LSTM em Python



Previsão 1 passo à frente

```
▶ y_pred = lstm_model.predict(testX)
    plt.plot(testY, marker='.', label="true")
    plt.plot(y_pred, 'r', label="prediction")
    plt.ylabel('Value')
    plt.xlabel('Time Step')
    plt.legend()
    plt.show();
```



Considerações finais

Foram apresentadas algumas arquiteturas de redes neurais artificiais para treinamento supervisionado: MLP, RBF, GMDH, RNN e LSTM

Aplicações apresentadas: ajuste de funções e previsão

As redes neurais artificiais ocupam um lugar de destaque na ciência de dados.

As redes neurais artificiais são bastante flexíveis e funcionam como uma ferramenta genérica para modelagem em diferentes tipos de problemas em *business intelligence* e mineração de dados.

Os recursos disponibilizados pelo R e Python facilitam a implementação das redes neurais artificiais

Os códigos disponibilizados na apresentação podem ser facilmente adaptados para outras aplicações.

Referências bibliográficas

GERRISH, S. How smart machine think, MIT Press 2019.

GÉRON, A. Mão à obra: aprendizado de máquina com Scikit-Learn, Keras & TensorFlow, 2^a edição, Rio de Janeiro: Alta Books, 2021.

KELLEHER, J.D. Deep Learning, MIT Press, 2019.

SEJNOWSKI, T.J. A revolução do aprendizado profundo, Rio de Janeiro: Alta Books, 2019.

WIDROW, B., LEHR, M.A. 30 Years of adaptative neural networks: perceptron, Madaline and backpropagation, Proceedings of the IEEE, vol. 78, no. 9, pp. 1415-1442, Sept.1990.