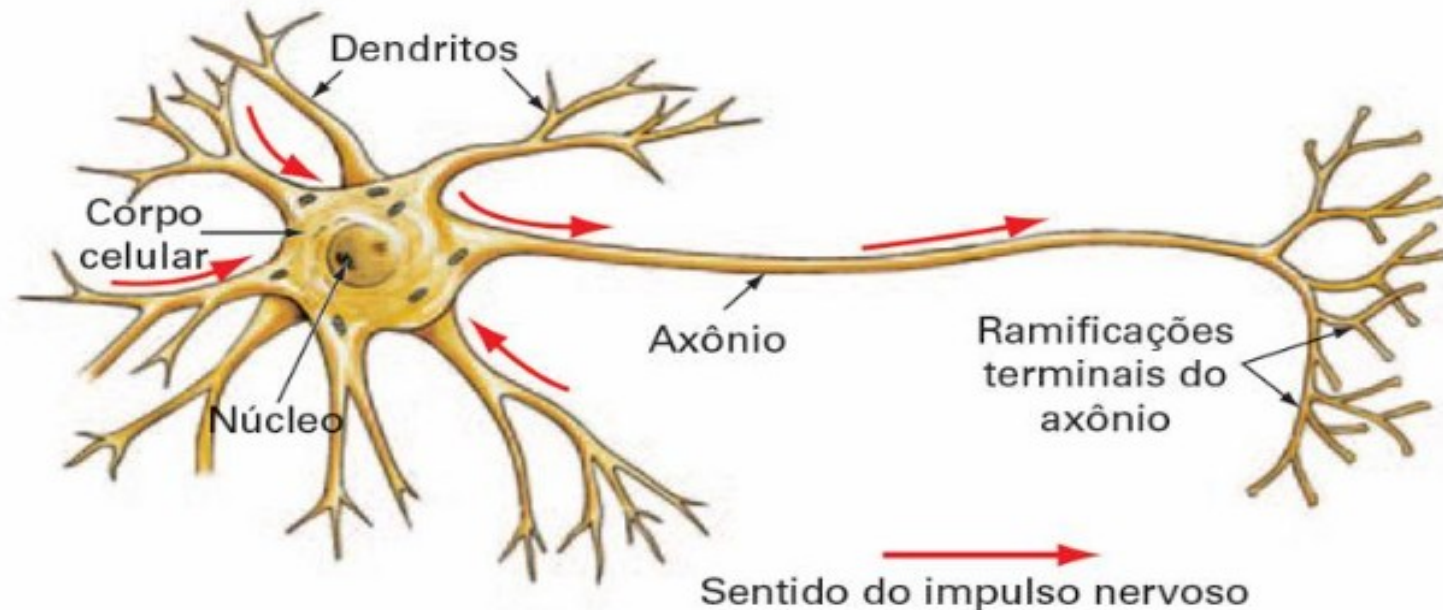


Introdução às Redes Neurais Artificiais [ANN]

O Neurônio Biológico

O neurônio é a unidade básica do cérebro humano, sendo uma célula especializada na transmissão de informações, pois nelas estão introduzidas propriedades de excitabilidade e condução de mensagens nervosas. O neurônio é constituído por 3 partes principais: a soma ou corpo celular, do qual emanam algumas ramificações denominadas de dendritos, e por uma outra ramificação descendente da soma, porém mais extensa, chamada de axônio. Nas extremidades dos axônios estão os nervos terminais, pelos quais é realizada a transmissão das informações para outros neurônios. Esta transmissão é conhecida como sinapse.



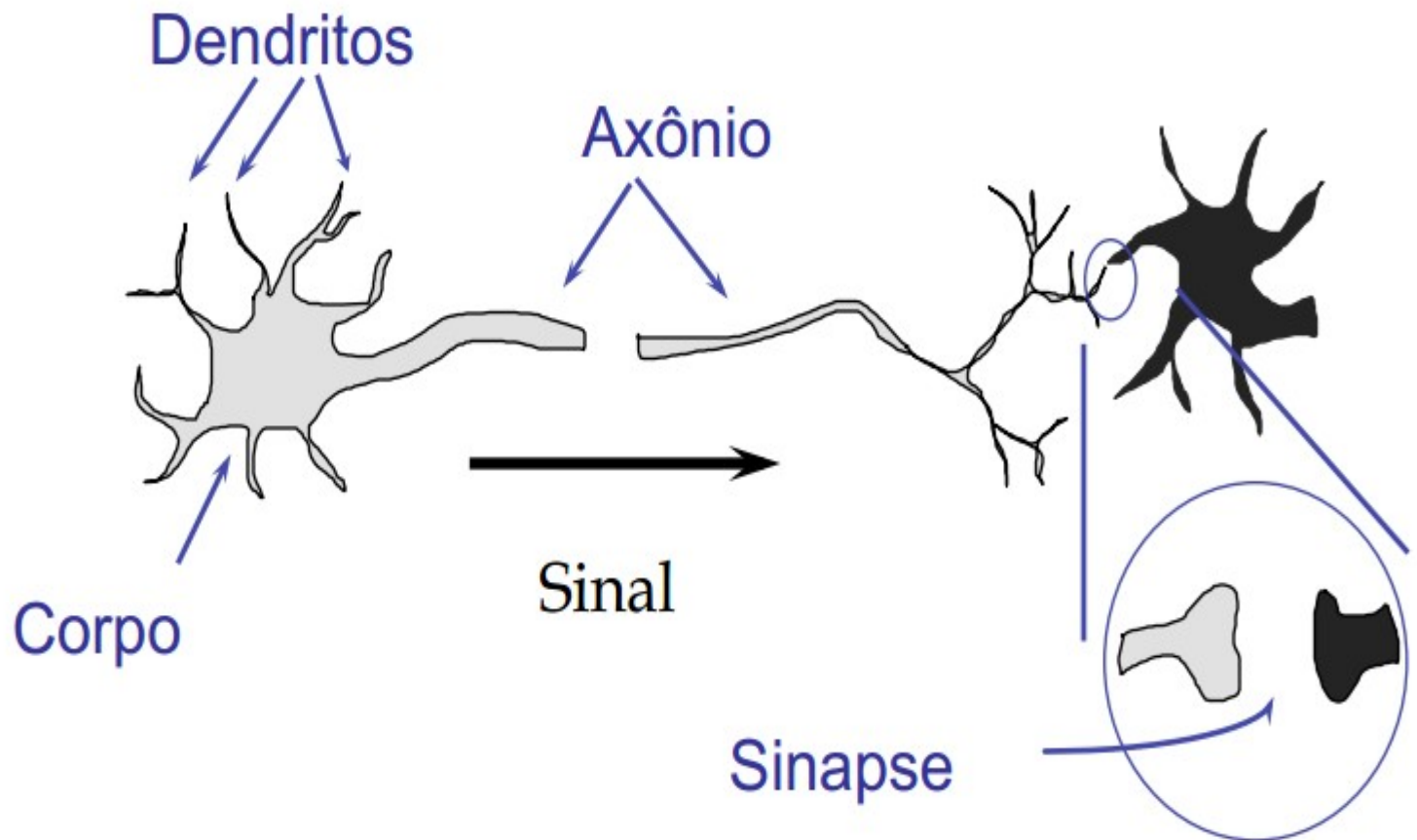


Redes Neurais Artificiais

- Sistemas distribuídos inspirados no cérebro humano
 - Compostas por várias unidades de processamento ("neurônios")
 - Interligadas por um grande número de conexões ("sinapses")
- Arquitetura e aprendizado

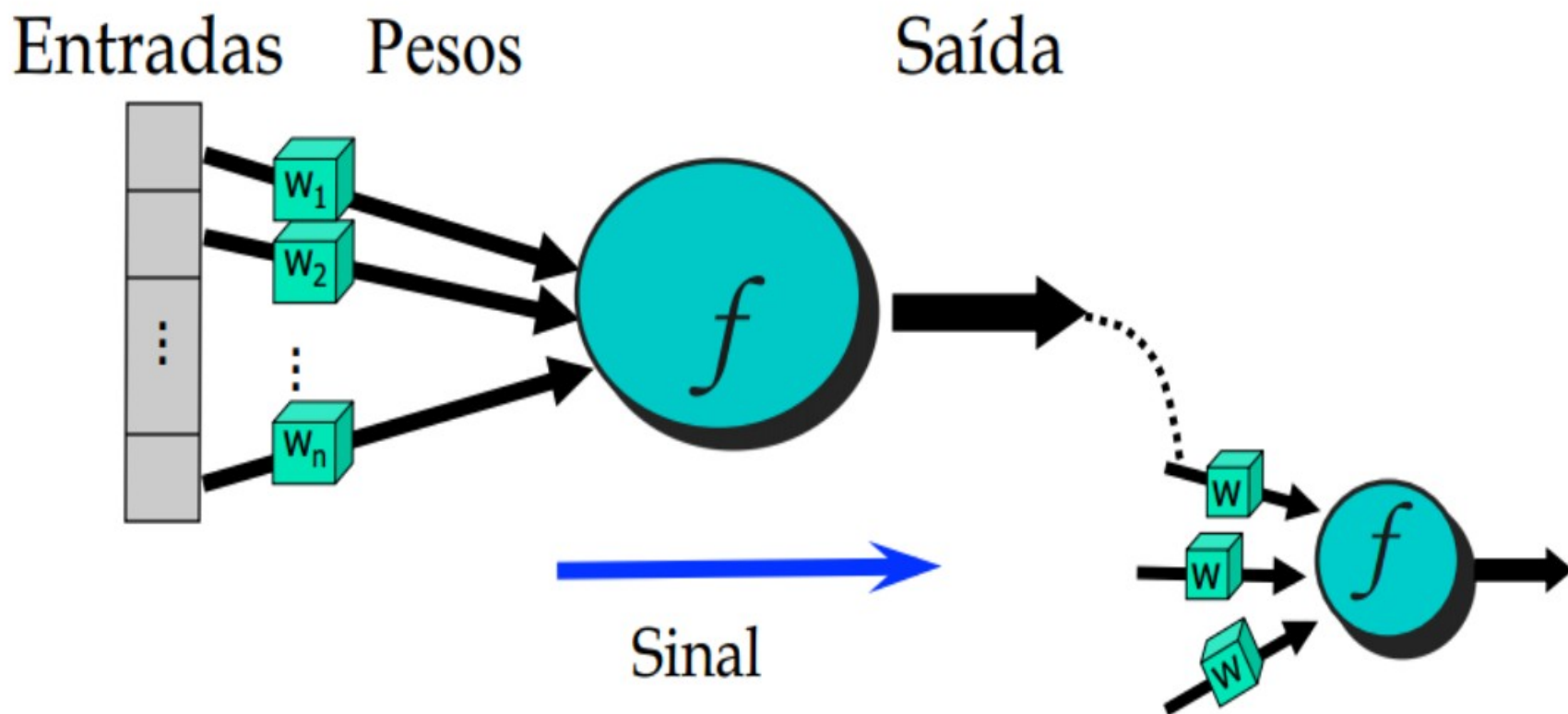
Neurônio natural

- Um neurônio simplificado:



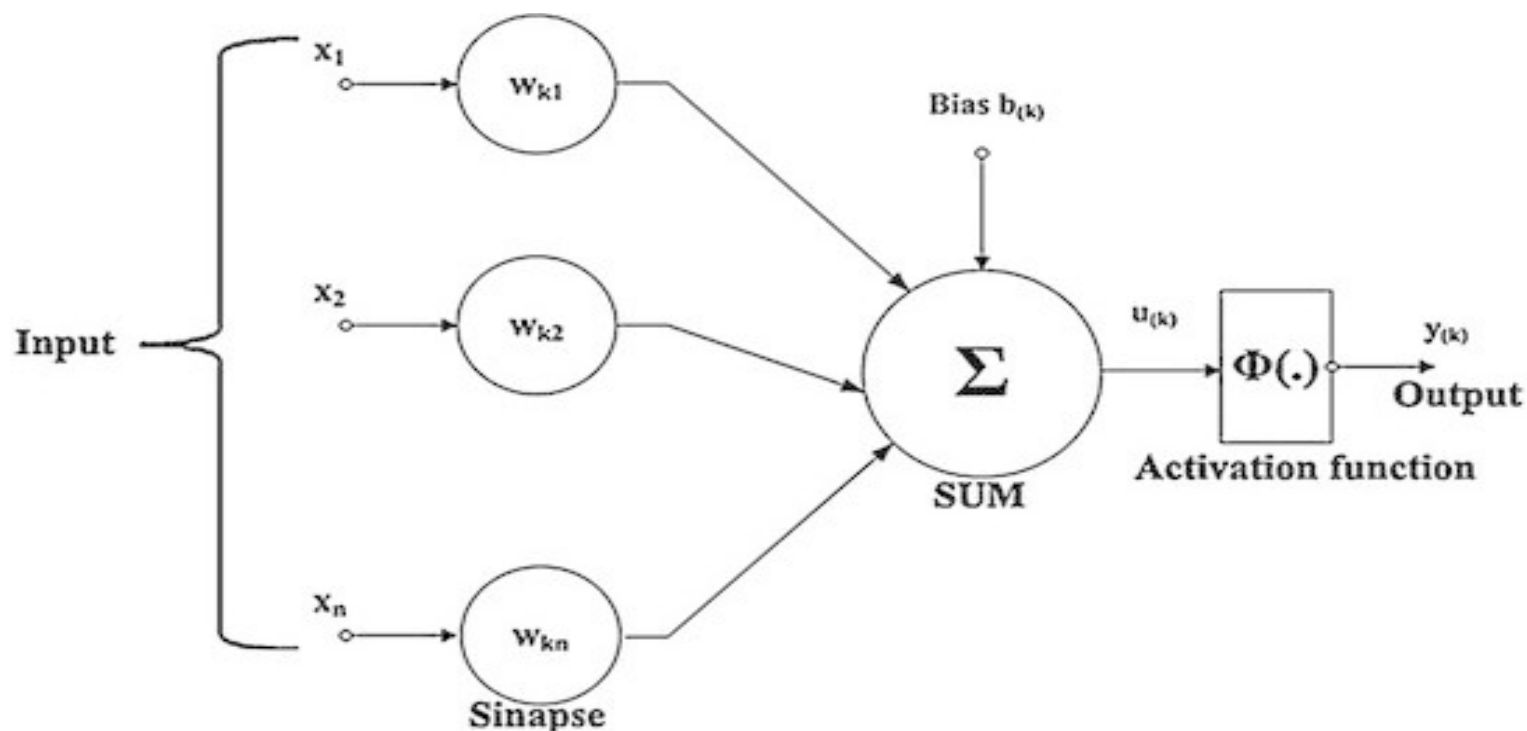
Neurônio artificial

- Modelo de um neurônio abstrato:



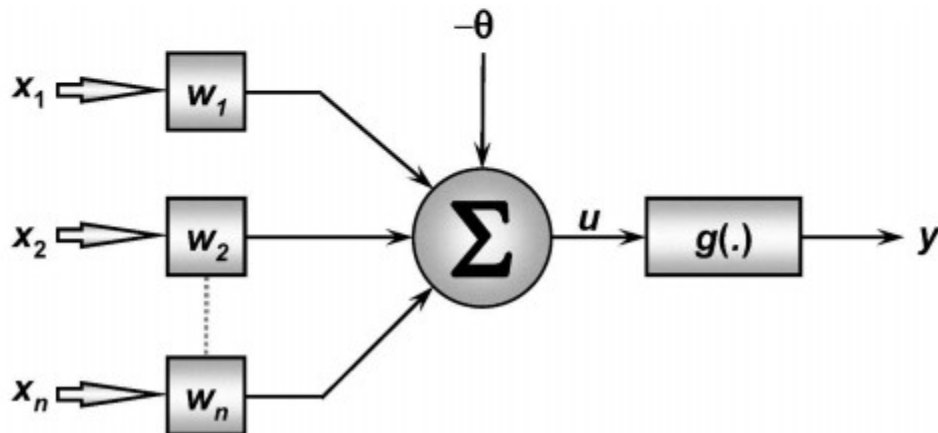
O Neurônio Matemático

A partir da estrutura e funcionamento do neurônio biológico, pesquisadores tentaram simular este sistema em computador. O modelo mais bem aceito foi proposto por Warren McCulloch e Walter Pitts em 1943, o qual implementa de maneira simplificada os componentes e o funcionamento de um neurônio biológico. Em termos simples, um neurônio matemático de uma rede neural artificial é um componente que calcula a soma ponderada de vários inputs, aplica uma função e passa o resultado adiante.



Redes Neurais Artificiais

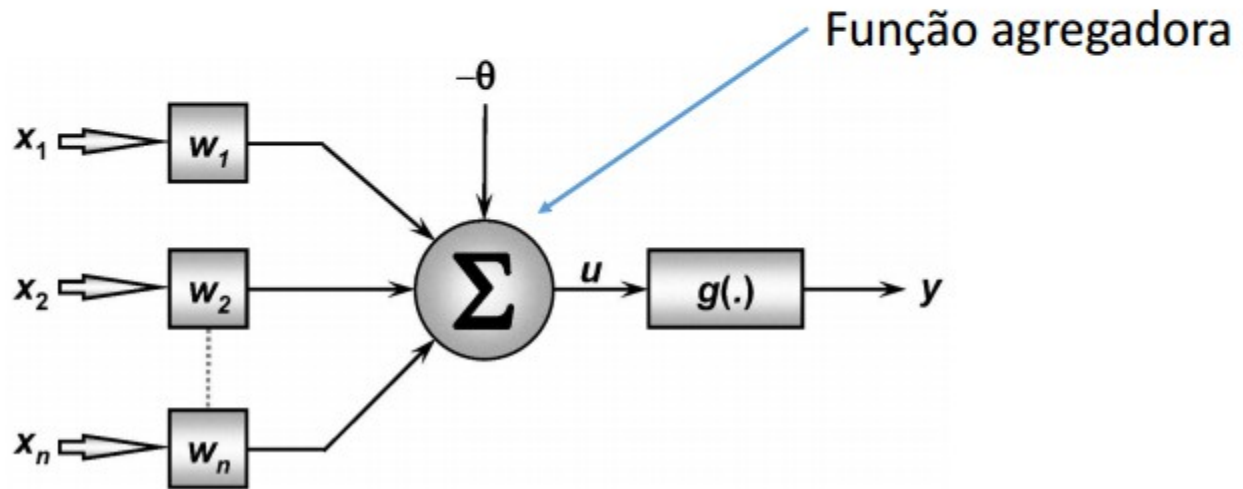
- Os sinais de entrada $\{x_1, x_2, x_n\}$ são ponderados por $\{w_1, w_2, w_n\}$.
- Existe um produto do sinal de entrada pelo peso que pondera.
- Isso é dado como entrada para o neurônio.



- Os sinais são informações do problema.
- Os pesos sinápticos $\{w_1, w_2, w_n\}$ ponderam a entrada.
- Para cada uma das entradas, tem o peso para ponderar.

Redes Neurais Artificiais

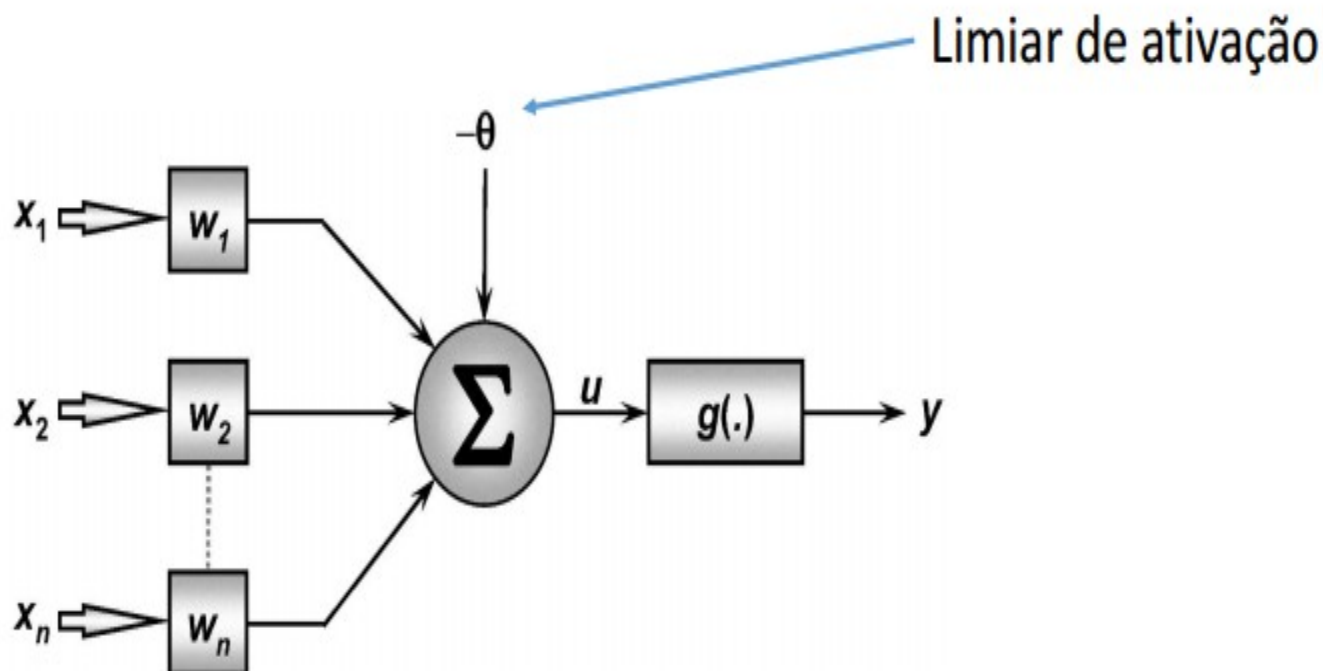
- A função agregadora recebe todos os sinais e realiza a soma dos produtos dos sinais.



- O neurônio deixa passar ou inibe um determinado sinal ou até mesmo altera a saída de acordo com a entrada.

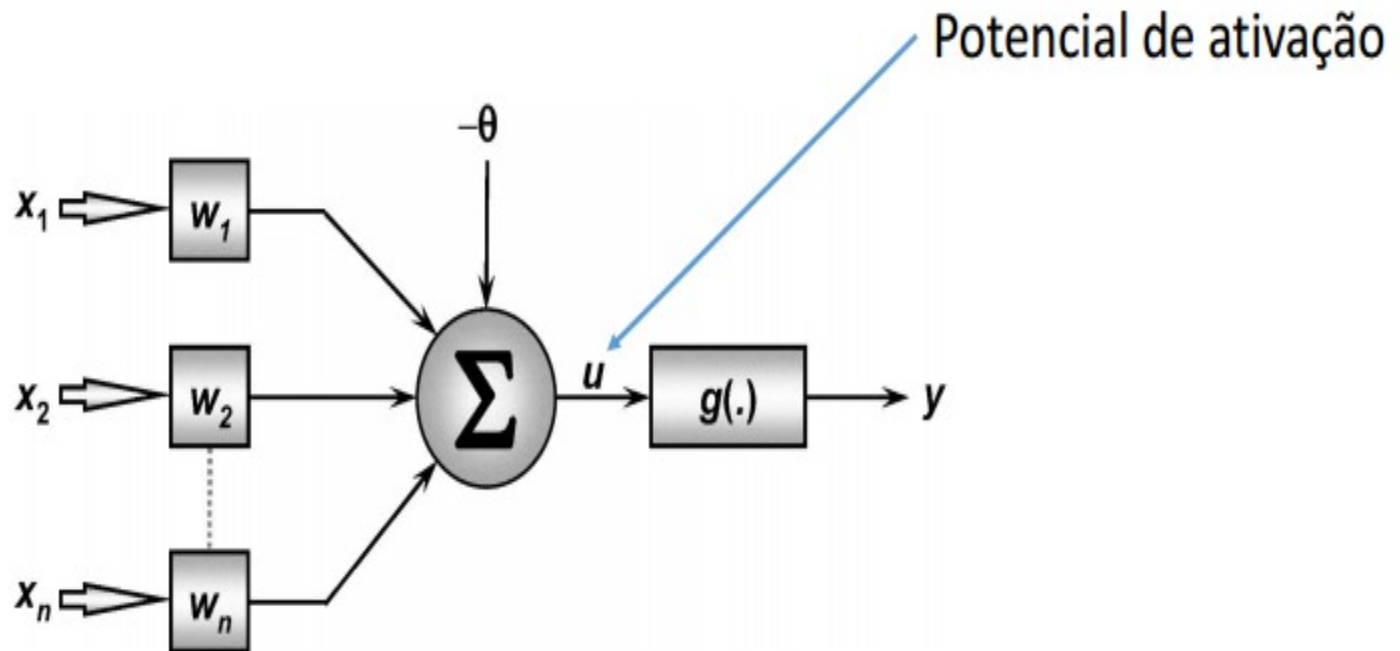
Redes Neurais Artificiais

- O limiar é uma constante (geralmente é ponderada) que vai indicar um limiar para o sinal passar ou não.



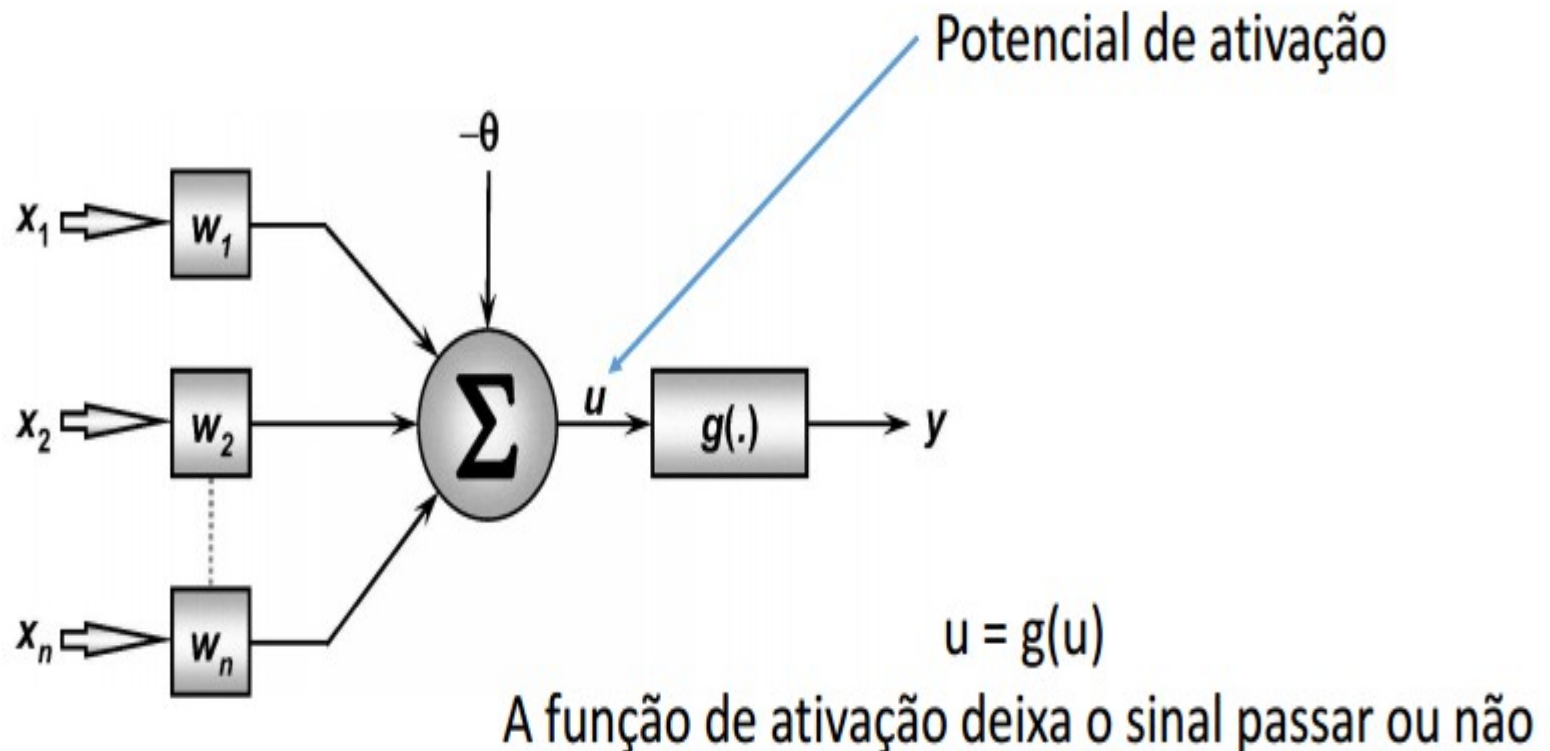
Redes Neurais Artificiais

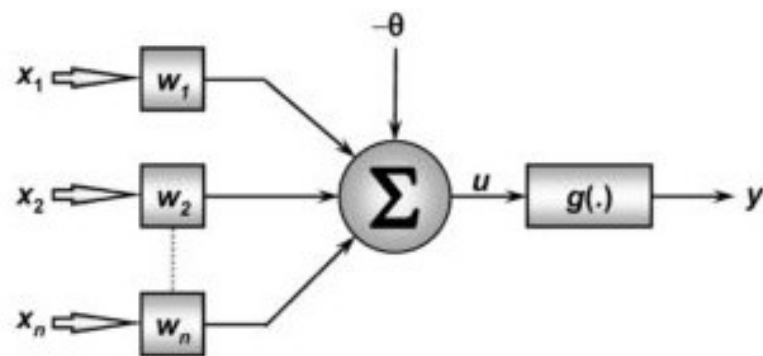
- “g” é a função de ativação. O valor “u” é dado como entrada para a função de ativação.



Redes Neurais Artificiais

- Potencial de ativação: $u = \sum_{i=1}^n w_i * x_i - \theta$ Somatório do produto das entradas

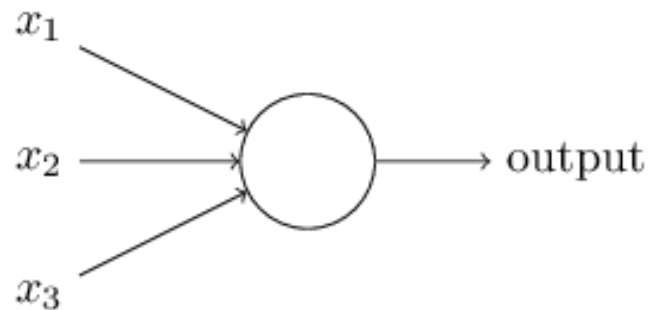




- Sinais de entrada $\{ X_1, X_2, \dots, X_n \}$: São os sinais externos normalmente normalizados para incrementar a eficiência computacional dos algoritmos de aprendizagem. São os dados que alimentam seu modelo preditivo.
- Pesos sinápticos $\{ W_1, W_2, \dots, W_n \}$: São valores para ponderar os sinais de cada entrada da rede. Esses valores são aprendidos durante o treinamento.
- Combinador linear $\{ \Sigma \}$: Agregar todos sinais de entrada que foram ponderados pelos respectivos pesos sinápticos a fim de produzir um potencial de ativação.
- Limiar de ativação $\{ \Theta \}$: Especifica qual será o patamar apropriado para que o resultado produzido pelo combinador linear possa gerar um valor de disparo de ativação.
- Potencial de ativação $\{ u \}$: É o resultado obtido pela diferença do valor produzido entre o combinador linear e o limiar de ativação. Se o valor for positivo, ou seja, se $u \geq 0$ então o neurônio produz um potencial excitatório; caso contrário, o potencial será inibitório.
- Função de ativação $\{ g \}$: Seu objetivo é limitar a saída de um neurônio em um intervalo valores.
- Sinal de saída $\{ y \}$: É o valor final de saída podendo ser usado como entrada de outros neurônios que estão sequencialmente interligados.

O Perceptron

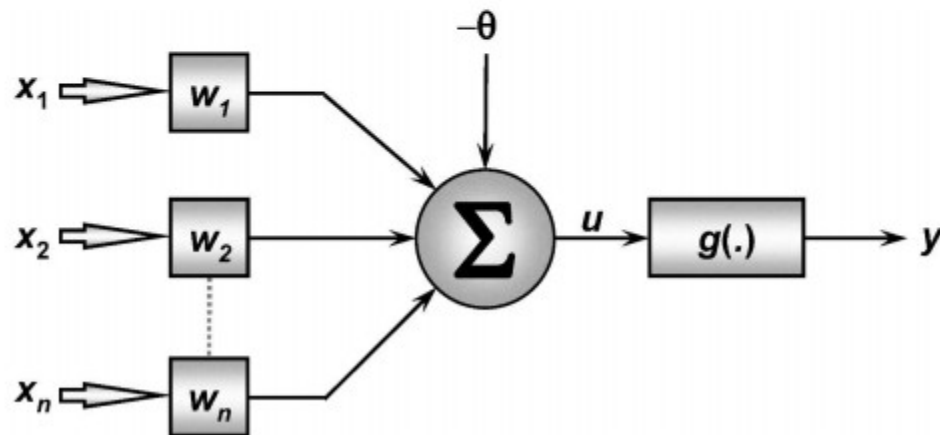
O Modelo Perceptron foi desenvolvido nas décadas de 1950 e 1960 pelo cientista Frank Rosenblatt, inspirado em trabalhos anteriores de Warren McCulloch e Walter Pitts. Hoje, é mais comum usar outros modelos de neurônios artificiais, mas o Perceptron permite uma compreensão clara de como funciona uma rede neural em termos matemáticos, sendo uma excelente introdução.



$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{threshold} \end{cases}$$

Rede Perceptron

- Rede muito simples.
- Possui apenas um neurônio.

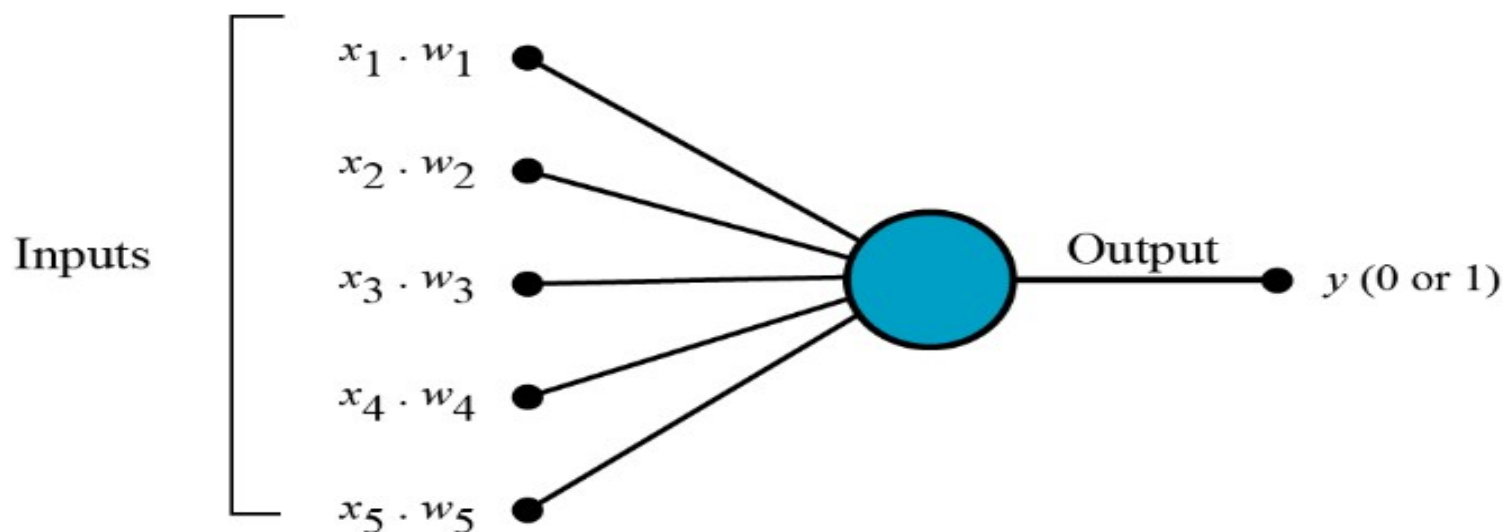


- Se o nosso “g” é a função degrau bipolar, então a saída será 1 ou -1.
- Será 1 se o somatório dos produtos for ≥ 0 .
- Será -1 se o somatório dos produtos for < 0 .

$$y = \begin{cases} 1, & \text{se } \left(\sum_{i=1}^n w_i * x_i - \theta \right) \geq 0 \Leftrightarrow w_1 * x_1 + w_2 * x_2 - \theta \geq 0 \\ -1, & \text{se } \left(\sum_{i=1}^n w_i * x_i - \theta \right) < 0 \Leftrightarrow w_1 * x_1 + w_2 * x_2 - \theta < 0 \end{cases}$$

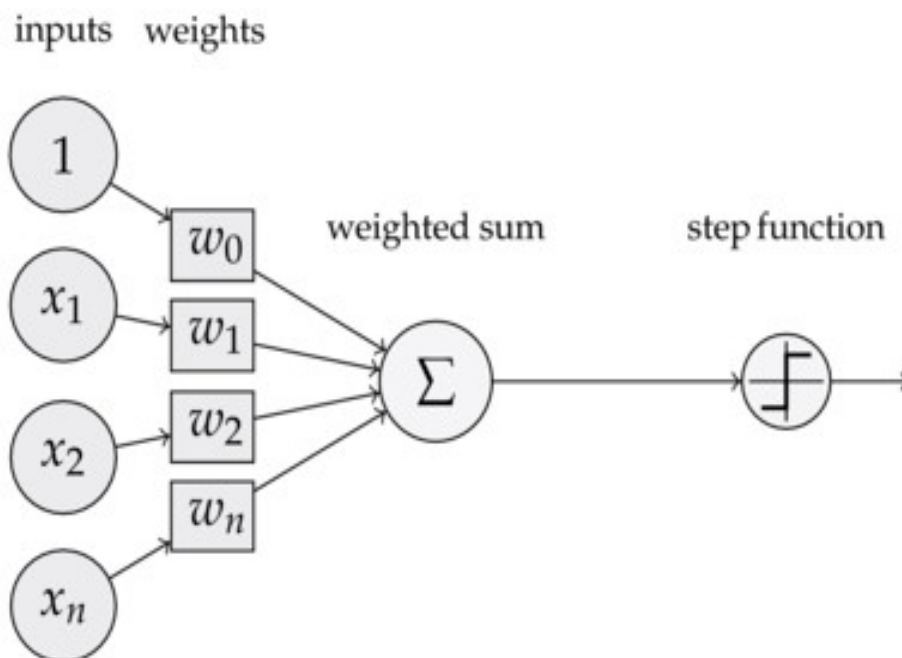
Vamos simplificar a maneira como descrevemos os Perceptrons. No limite de condição $\sum w_j x_j > \text{threshold}$ podemos fazer duas mudanças de notação para simplificá-lo. A primeira mudança é escrever $\sum w_j x_j$ como um produto (dot product), $w \cdot x \equiv \sum w_j x_j$, onde w e x são vetores cujos componentes são os pesos e entradas, respectivamente. A segunda mudança é mover o threshold para o outro lado da equação e substituí-lo pelo que é conhecido como o viés (bias) do Perceptron, ou $b \equiv -\text{threshold}$. Usando o viés em vez do threshold, a regra Perceptron pode ser reescrita:

$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$



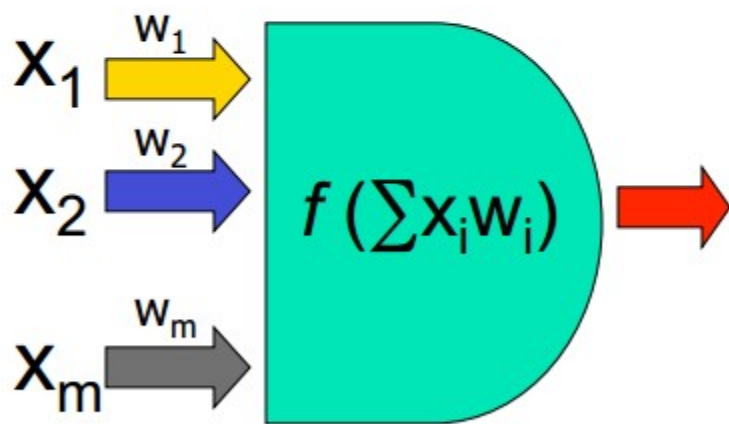
Neurônios

Nesta introdução, para exemplificar a construção e o treinamento de um sistema moderno de IA, realizaremos uma simples tarefa de visão computacional, na qual usaremos uma rede neural bem simples para reconhecer dígitos escritos. Em termos técnicos, será uma tarefa de OCR (Optical Character Recognition). Mas, antes de entendermos o que são e como treinar redes neurais, precisamos falar sobre seu componente mais básico: os neurônios.



Perceptron

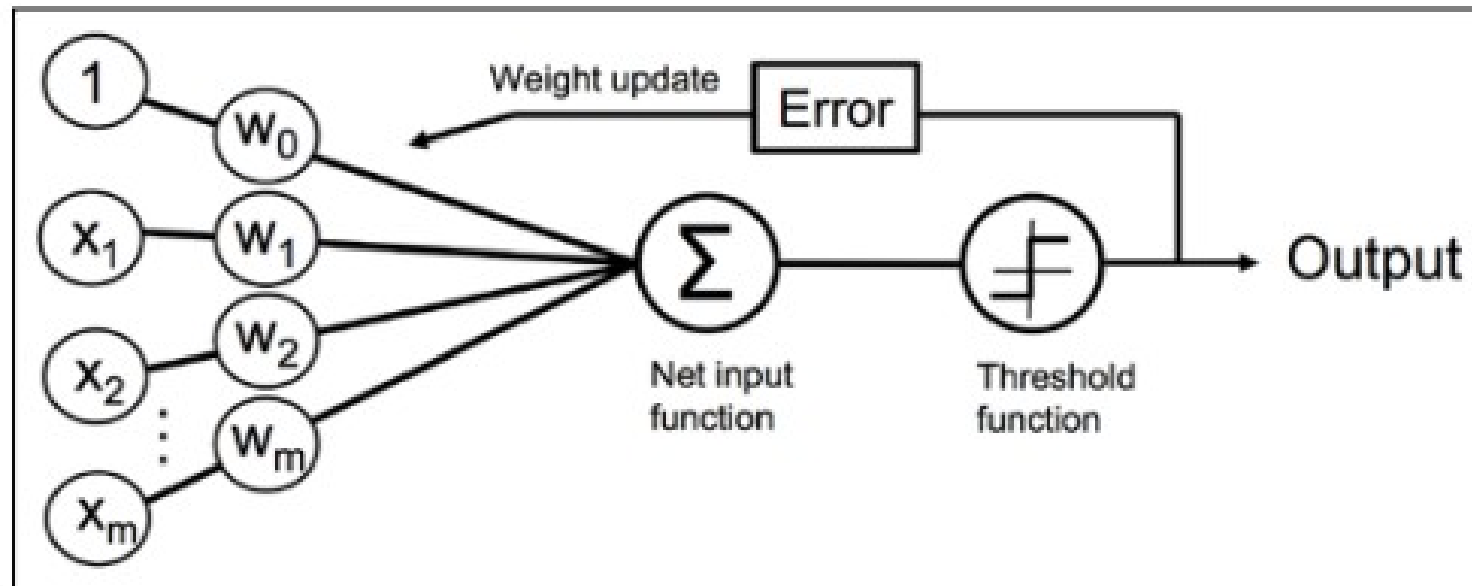
- Primeira rede - Rosemblat, 1958
 - Modelo de neurônio de McCulloch-Pitts
- Treinamento
 - Supervisionado
 - Correção de erro
 - $w_i(t) = w_i(t-1) + \Delta w_i$
 - $\Delta w_i = \eta x_i \delta$
 - $\Delta w_i = \eta x_i (y - f(x))$
- Teorema de convergência



The formal definition of an artificial neuron

More formally, we can put the idea behind **artificial neurons** into the context of a binary classification task where we refer to our two classes as 1 (positive class) and -1 (negative class) for simplicity. We can then define a decision function ($\phi(z)$) that takes a linear combination of certain input values \mathbf{x} and a corresponding weight vector \mathbf{w} , where z is the so-called net input $z = w_1x_1 + \dots + w_mx_m$:

$$\mathbf{w} = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$$



Now, if the net input of a particular sample $\mathbf{x}^{(i)}$ is greater than a defined threshold θ , we predict class 1, and class -1 otherwise. In the perceptron algorithm, the decision function $\phi(\cdot)$ is a variant of a **unit step function**:

$$\phi(z) = \begin{cases} 1 & \text{if } z \geq \theta \\ -1 & \text{otherwise} \end{cases}$$

For simplicity, we can bring the threshold θ to the left side of the equation and define a weight-zero as $w_0 = -\theta$ and $x_0 = 1$ so that we write z in a more compact form:

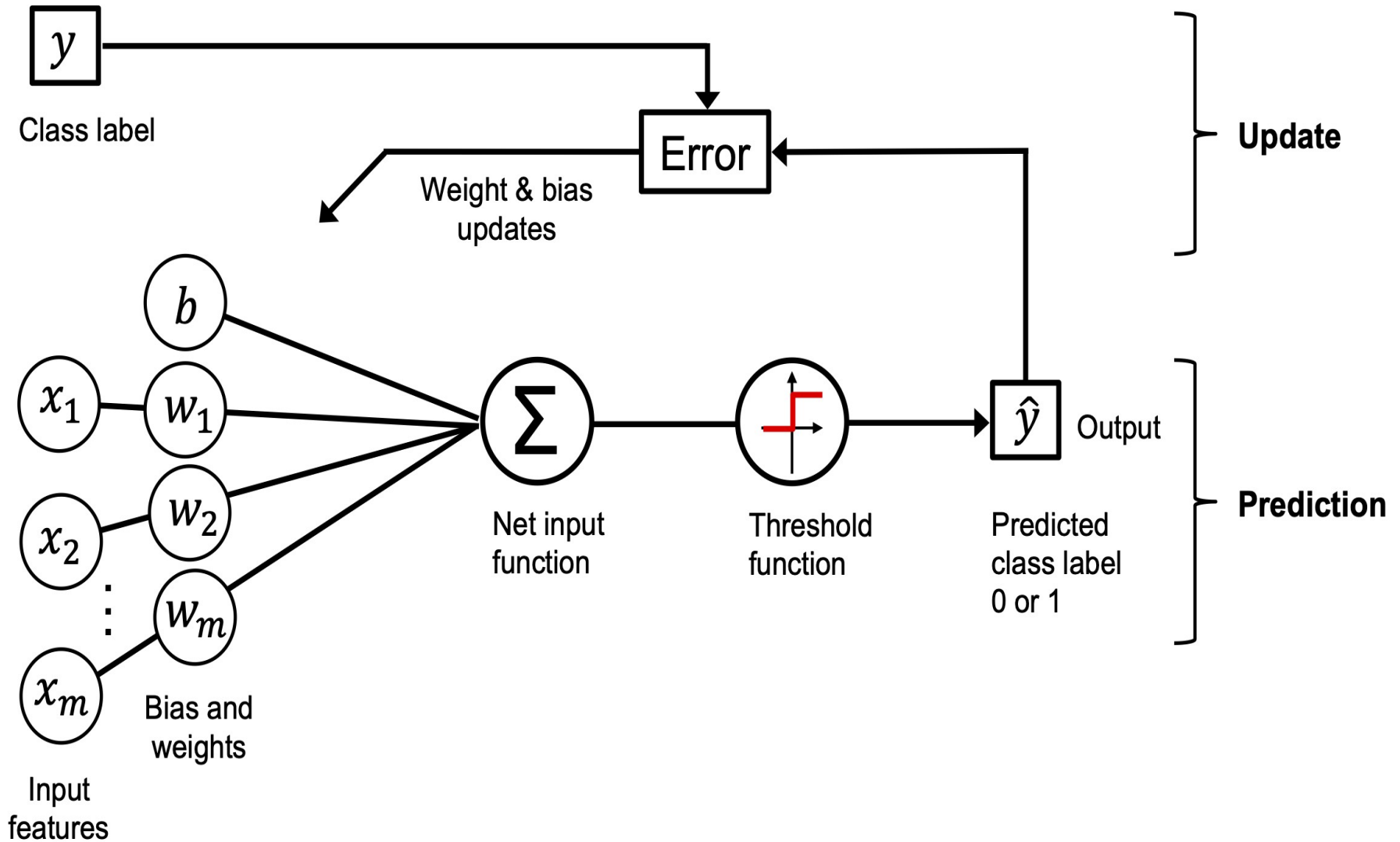
$$Z = w_0 x_0 + w_1 x_1 + \dots + w_m x_m = \mathbf{w}^T \mathbf{x}$$

And:

$$\phi(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

In machine learning literature, the negative threshold, or weight, $w_0 = -\theta$, is usually called the **bias unit**.

Perceptron



The perceptron learning rule

The whole idea behind the MCP neuron and Rosenblatt's *thresholded* perceptron model is to use a reductionist approach to mimic how a single neuron in the brain works: it either *fires* or it doesn't. Thus, Rosenblatt's initial perceptron rule is fairly simple and can be summarized by the following steps:

1. Initialize the weights to 0 or small random numbers.
2. For each training sample $\mathbf{x}^{(i)}$:
 - a. Compute the output value \hat{y} .
 - b. Update the weights.

Here, the output value is the class label predicted by the unit step function that we defined earlier, and the simultaneous update of each weight w_j in the weight vector \mathbf{w} can be more formally written as:

$$w_j := w_j + \Delta w_j$$

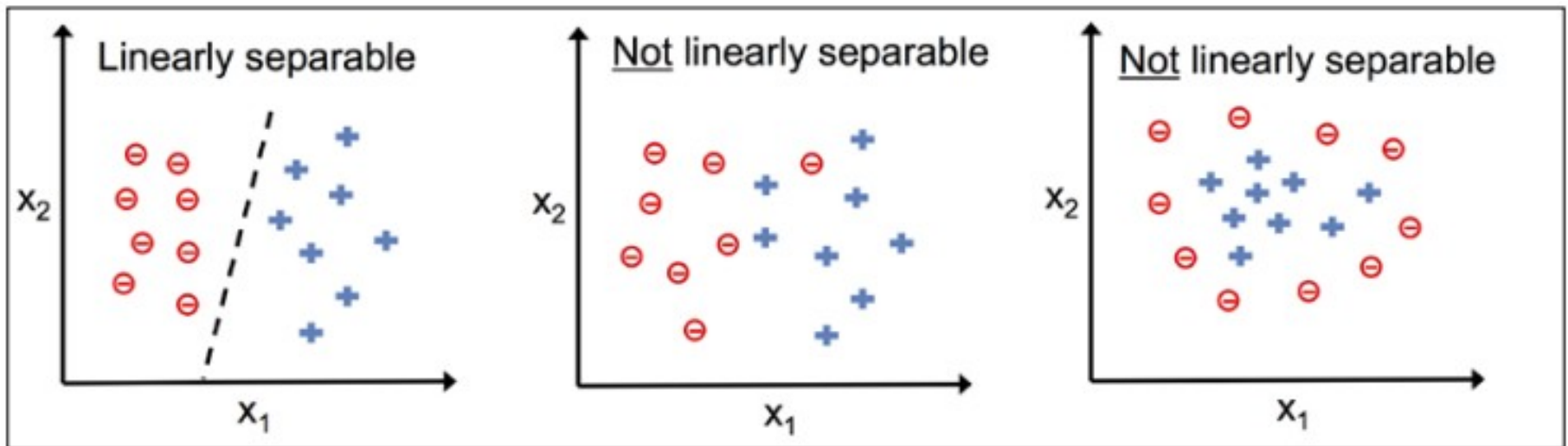
The value of Δw_j , which is used to update the weight w_j , is calculated by the perceptron learning rule:

$$\Delta w_j = \eta (y^{(i)} - \hat{y}^{(i)}) x_j^{(i)}$$

Where η is the **learning rate** (typically a constant between 0.0 and 1.0), $y^{(i)}$ is the **true class label** of the i th training sample, and $\hat{y}^{(i)}$ is the **predicted class label**.

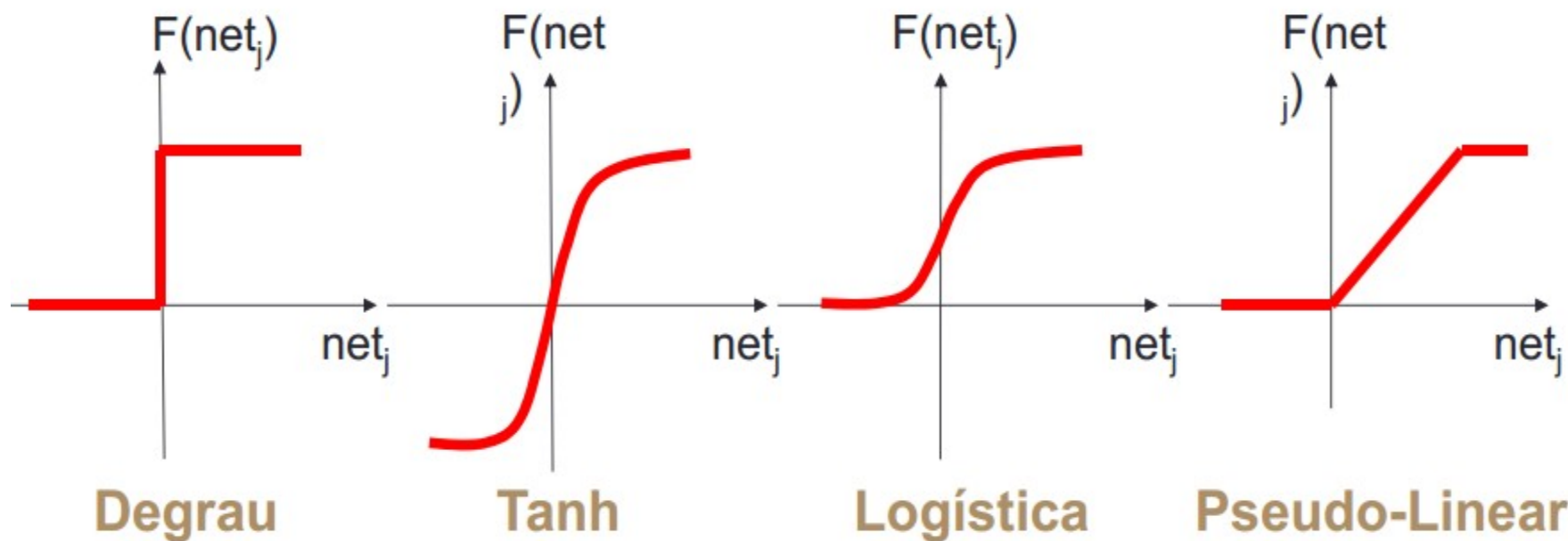
Perceptron

It is important to note that the convergence of the perceptron is only guaranteed if the two classes are linearly separable and the learning rate is sufficiently small. If the two classes can't be separated by a linear decision boundary, we can set a maximum number of passes over the training dataset (**epochs**) and/or a threshold for the number of tolerated misclassifications – the perceptron would never stop updating the weights otherwise:



Funções de Ativação

É a função que determina o nível de ativação do Neurônio Artificial: $s_j = F(\text{net}_j)$



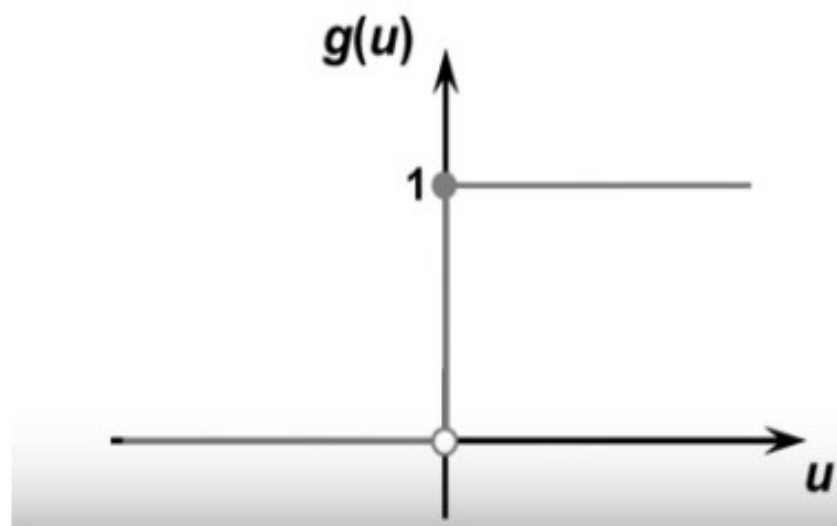
Redes Neurais Artificiais

- Existem algumas funções de ativação:
 - Função degrau

$$g(u) = \begin{cases} 1 & \text{se } (u \geq 0) \\ 0 & \text{se } (u < 0) \end{cases}$$

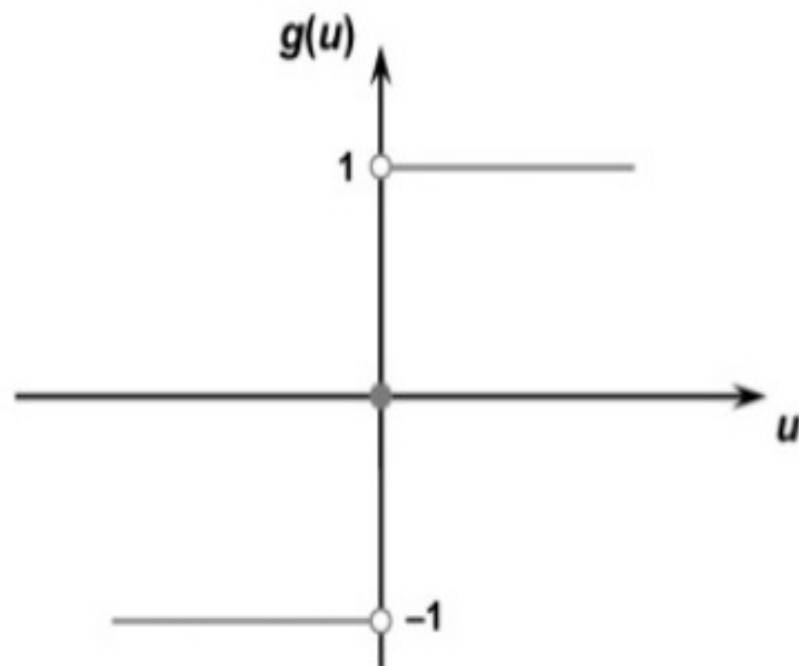
1 significa que houve ativação

0 significa que não houve ativação



Redes Neurais Artificiais

- Função degrau bipolar



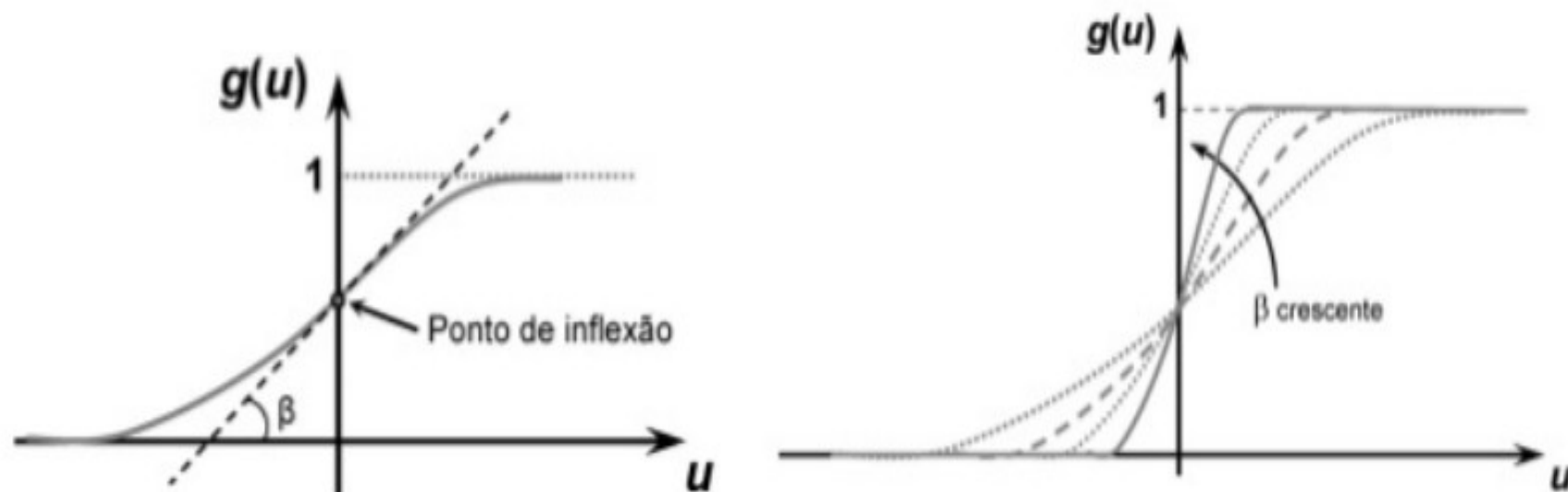
$$g(u) = \begin{cases} 1 & \text{se } (u > 0) \\ 0 & \text{se } (u = 0) \\ -1 & \text{se } (u < 0) \end{cases}$$

$$g(u) = \begin{cases} 1 & \text{se } (u \geq 0) \\ -1 & \text{se } (u < 0) \end{cases}$$

Redes Neurais Artificiais

- Função logística $g(u) = \frac{1}{1 + e^{-\beta u}}$

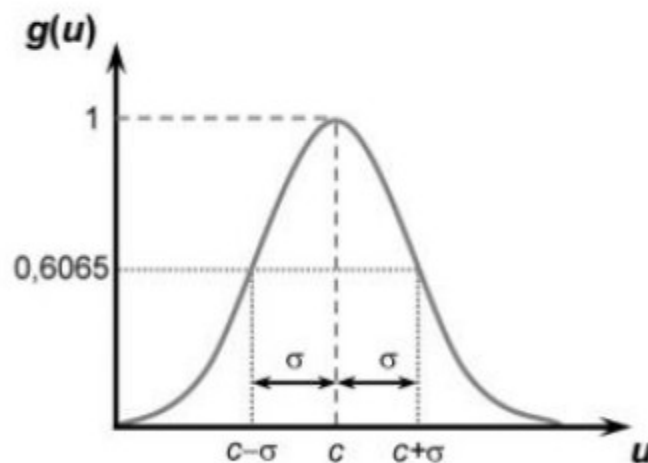
$e = 2,718281 \Rightarrow$ (Número de Euler)
 $\beta =$ constante de inclinação



Os limites de saída são de 0 a 1.

Redes Neurais Artificiais

- Função Gaussiana $g(u) = e^{-\frac{(u-c)^2}{2\sigma^2}}$



$e = 2,718281 \Rightarrow$ Número de Euler
 $\sigma \Rightarrow$ Desvio Padrão
 $c \Rightarrow$ Centro da Função Gaussiana

Muito utilizada em estatística e reconhecimento de padrões.

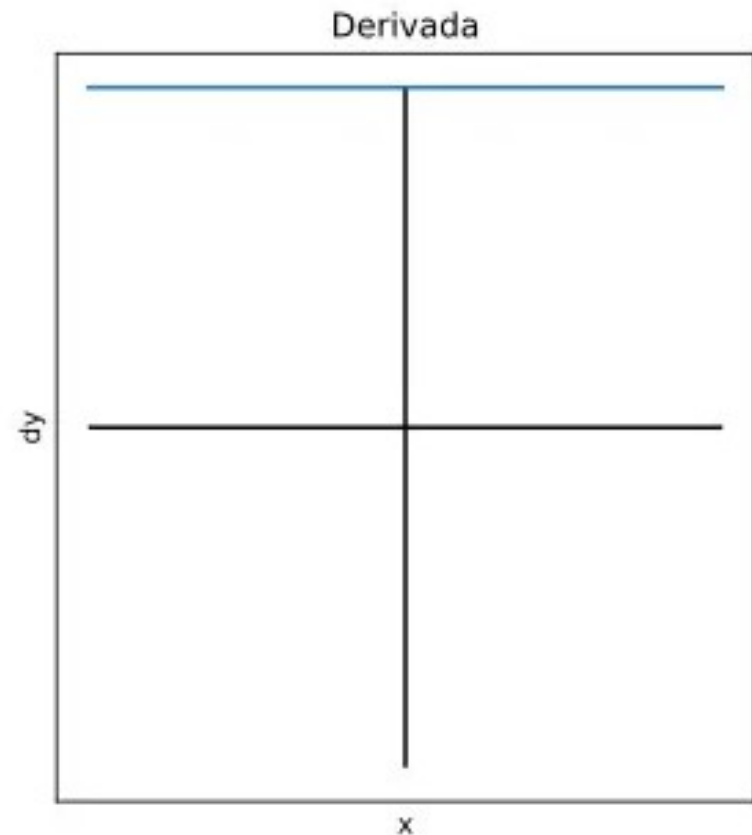
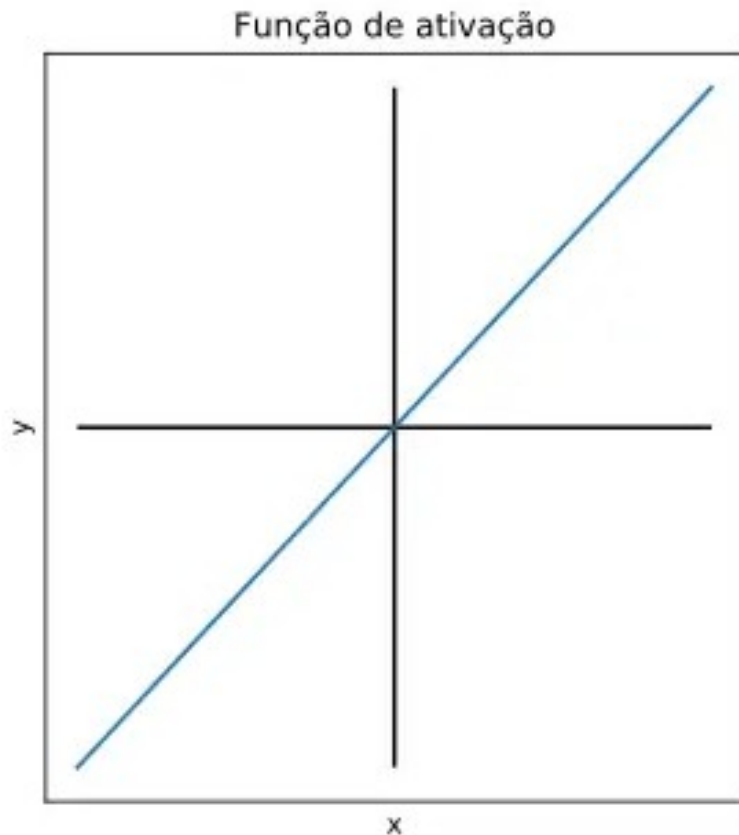
A ideia é concentrar as entradas no intervalo do meio.

Os sinais fora do desvio padrão vão tender a 0.

Principais funções de ativação

Linear

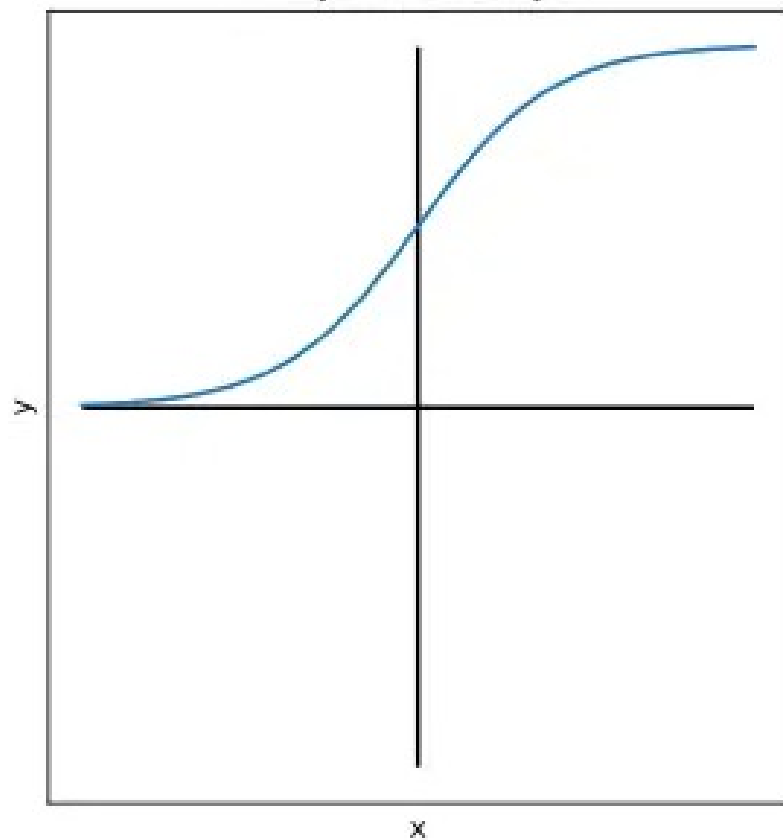
A função linear apenas aplica um fator de multiplicação ao valor que recebe.



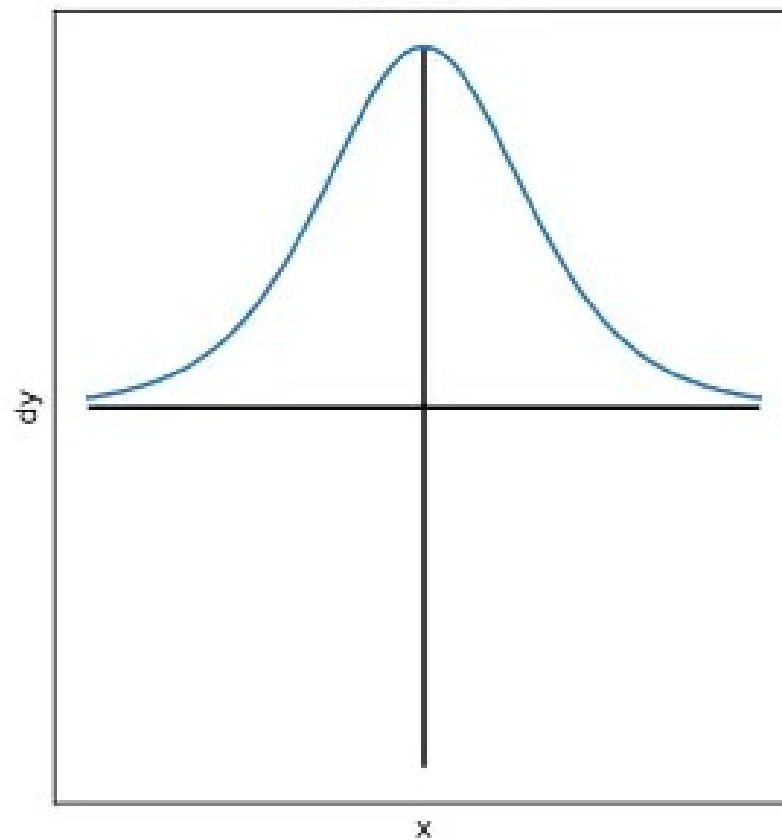
Sigmoide

A função logística ou sigmoide produz valores no intervalo $[0, 1]$.

Função de ativação



Derivada



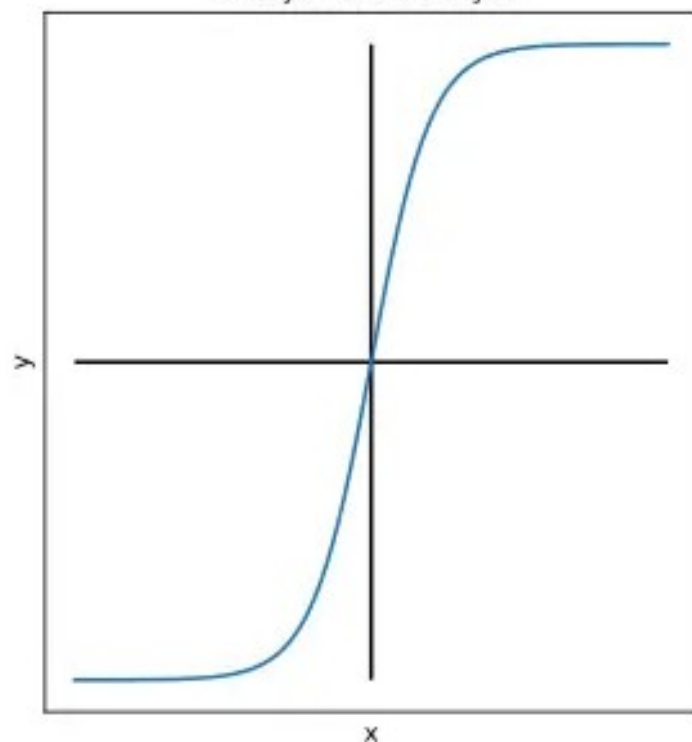
Softmax

Softmax é uma generalização da função sigmoide para casos não-binários. Ela não costuma ser aplicada às camadas escondidas da rede neural, mas sim na camada de saída de problemas de classificação multiclasse, já que sua característica é produzir valores no intervalo $[0, 1]$ onde sua soma é igual a 1. Ou seja, num problema com 3 classes, por exemplo, a função softmax vai produzir 3 valores, que somam 1, onde cada valor representa a probabilidade da instância pertencer a uma das 3 possíveis classes.

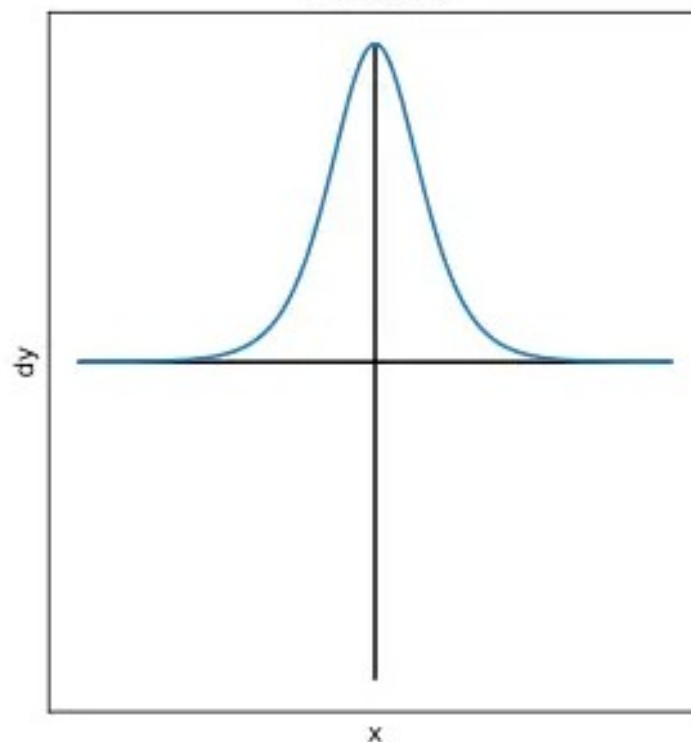
Tangente hiperbólica (tanh)

Produz resultados no intervalo $[-1, 1]$.

Função de ativação



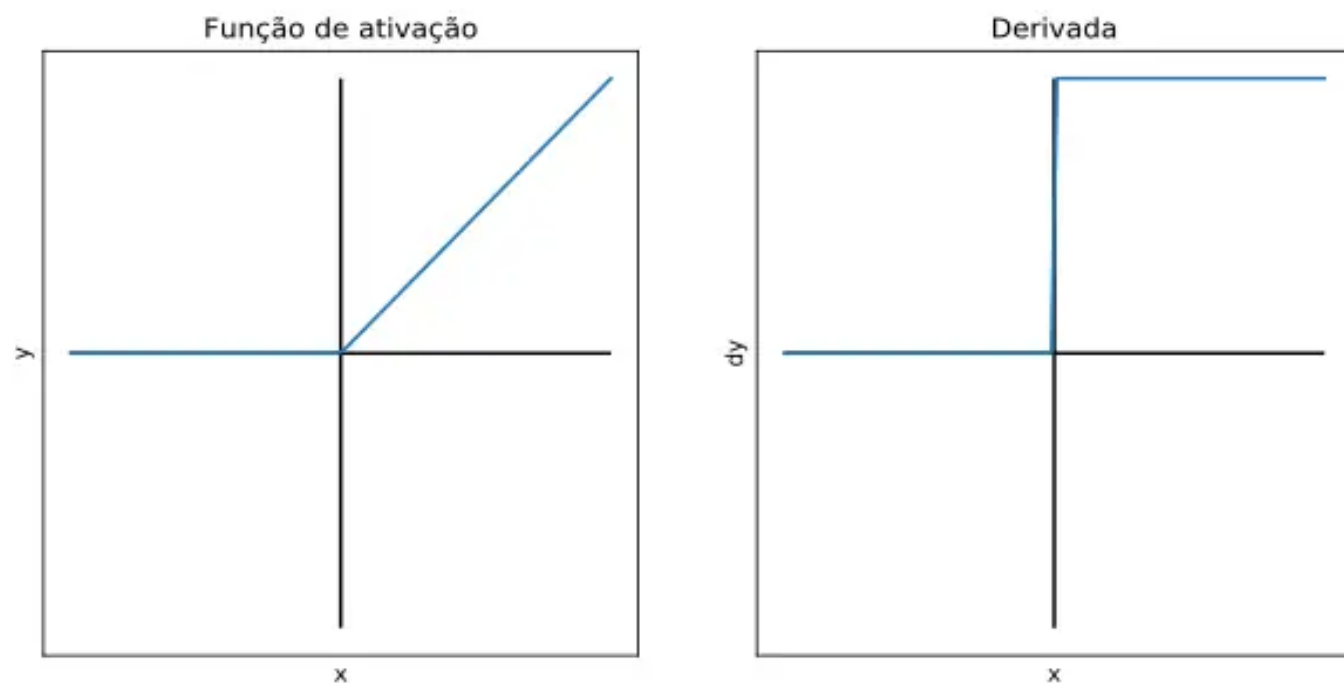
Derivada



A tangente hiperbólica tem as mesmas vantagens e ainda resolve um dos problemas da função sigmoide, sendo centrada em zero. Entretanto, sua derivada também converge a zero, e mais rapidamente.

ReLU

ReLU é uma abreviação para *rectified linear unit*, ou unidade linear retificada. Ela produz resultados no intervalo $[0, \infty[$.



A função ReLU retorna 0 para todos os valores negativos, e o próprio valor para valores positivos. É uma função computacionalmente leve, entretanto não é centrada em zero. Como seu resultado é zero para valores negativos, ela tende a “apagar” alguns neurônios durante um passo *forward*, o que aumenta a velocidade do treinamento, mas por outro pode fazer com que esses neurônios “morram” e não aprendam nada se eles só receberem valores negativos. Além disso, ela pode produzir ativações explodidas, já que não possui um limite positivo. Mesmo com suas limitações, a função ReLU é hoje uma das funções de ativação mais utilizadas no treinamento de redes neurais, e não costuma ser utilizada na camada de saída.

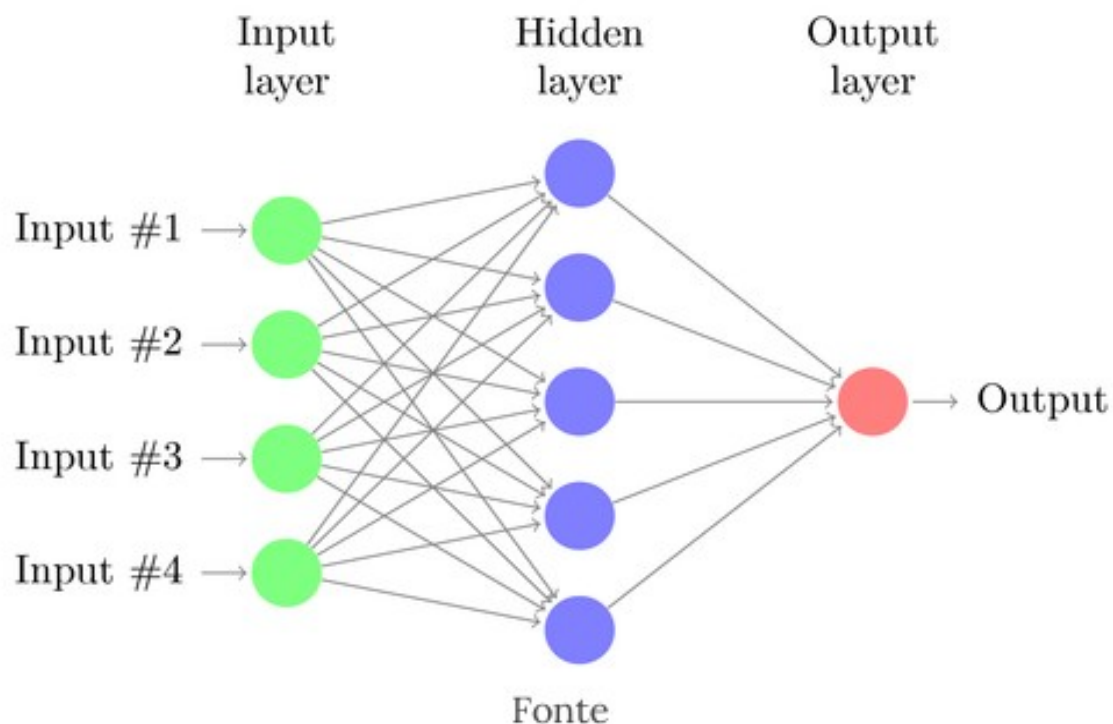


Rede Multi-Layer Perceptron

- Arquitetura de RNA mais utilizada
 - Uma ou mais camadas intermediárias de neurônios
- Funcionalidade (teórica)
 - Uma camada intermediária: qualquer função contínua ou Booleana
 - Duas camadas intermediárias: qualquer função
- Originalmente treinada com o algoritmo *Backpropagation*

Redes Neurais Artificiais

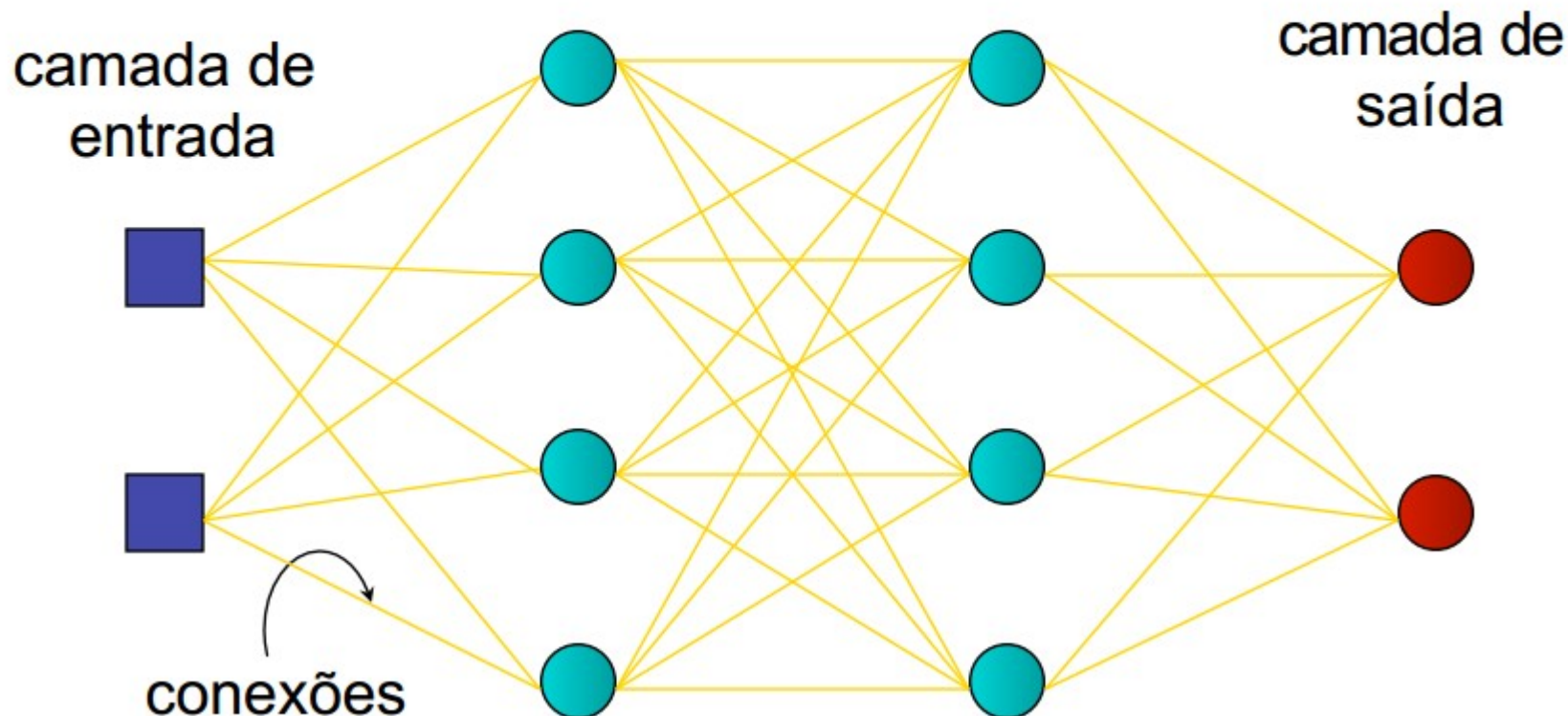
Infelizmente, os neurônios são bastante limitados. Em aprendizado de máquina, queremos que um algoritmo possa aprender qualquer tipo de padrão presente nos dados, mas isso não é possível com um simples neurônio. Por isso, construímos as redes neurais, que são simplesmente vários neurônios conectados. Pense nos neurônios como blocos de Lego e nas redes neurais como estruturas que montamos empilhando esses blocos de Lego. Dependendo da tarefa, uma estrutura pode se mais útil do que outra. No entanto, aqui, vamos considerar apenas a estrutura mais simples e mais comum de rede neural, o modelo de **redes neurais *feedforward* densas**.





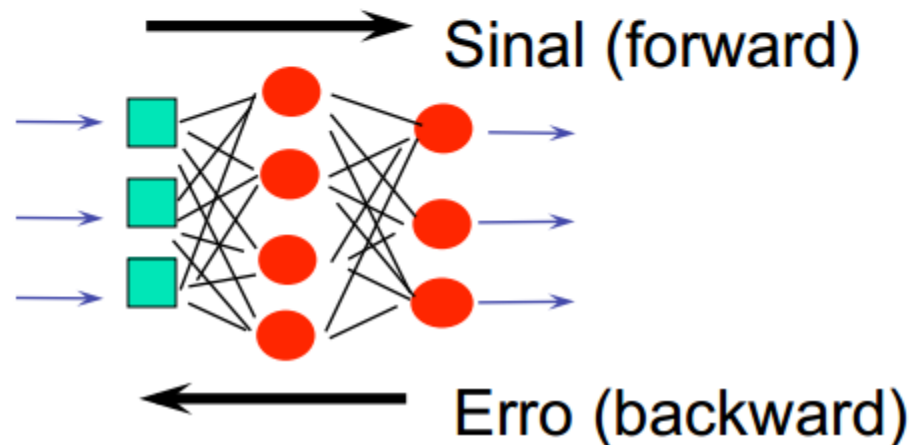
MLP e Backpropagation

camadas intermediárias



Backpropagation

- Treina a rede com pares entrada-saída
 - Cada vetor de entrada é associado a uma saída desejada
- Treinamento em duas fases, cada uma percorrendo a rede em um sentido
 - Fase forward
 - Fase backward





Treinamento

Iniciar todas as conexões com valores aleatórios $\in [a,b]$

Repita

erro = 0;

Para cada par de treinamento (X, y)

Para cada camada $k := 1$ a N

Para cada neurônio $j := 1$ a M_k

Calcular a saída $f_{kj}(net)$

Se $k = N$

Calcular soma dos erros de seus neurônios;

Se erro $> \epsilon$

Para cada camada $k := N$ a 1












Para cada neurônio $j := 1$ a M_k

Atualizar pesos;

Até erro $< \epsilon$ (ou número máximo de ciclos)

A mostly complete chart of Neural Networks

©2016 Fjodor van Veen - asimovinstitute.org

-  Backfed Input Cell
-  Input Cell
-  Noisy Input Cell
-  Hidden Cell
-  Probabilistic Hidden Cell
-  Spiking Hidden Cell
-  Output Cell
-  Match Input Output Cell
-  Recurrent Cell
-  Memory Cell
-  Different Memory Cell
-  Kernel
-  Convolution or Pool

Perceptron (P)



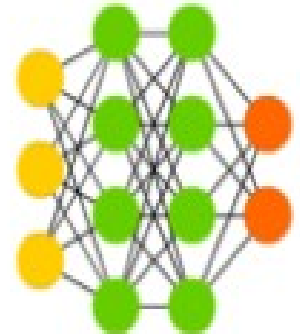
Feed Forward (FF)



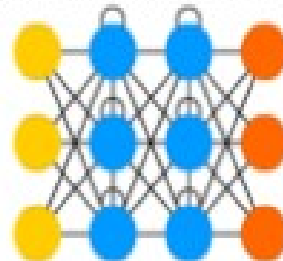
Radial Basis Network (RBF)



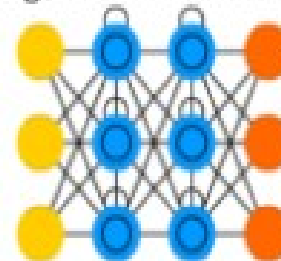
Deep Feed Forward (DFF)



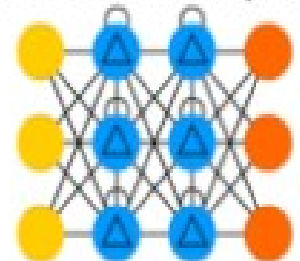
Recurrent Neural Network (RNN)



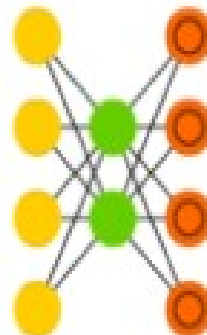
Long / Short Term Memory (LSTM)



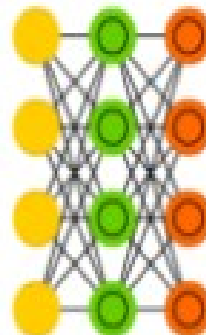
Gated Recurrent Unit (GRU)



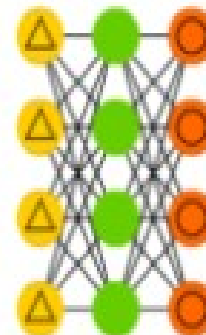
Auto Encoder (AE)



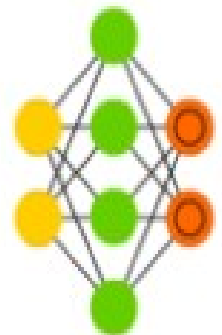
Variational AE (VAE)



Denoising AE (DAE)



Sparse AE (SAE)





Redes neurais profundas (RNP)

- Redes neurais MLP em geral têm 1 ou 2 camadas intermediárias
 - Redes neurais rasas
- Poucas camadas tornam difícil extrair função que represente os dados
- Uso de backpropagation em redes com muitas camadas leva a soluções pobres
 - Problema de atribuição de erro



RNs profundas

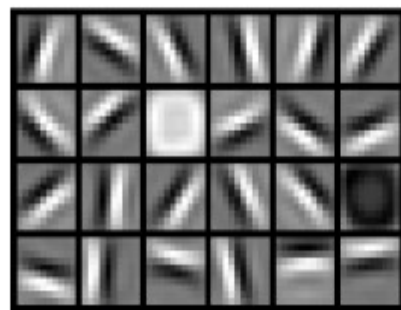
- RNs rasas
 - Características extraídas manualmente (por especialistas) ou por técnicas de extração
- RN profundas
 - Características extraídas hierarquicamente por algoritmos de aprendizado
 - Não supervisionado
 - Pode usar dados não rotulados
 - Semi-supervisionado

RNs profundas

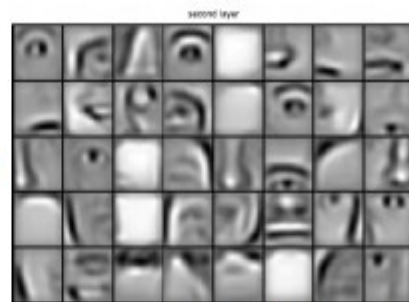
- Extração de características
 - Inicialmente características simples
 - Nível crescente de abstração
 - Cada camada aplica transformação não linear às características recebidas da camada anterior



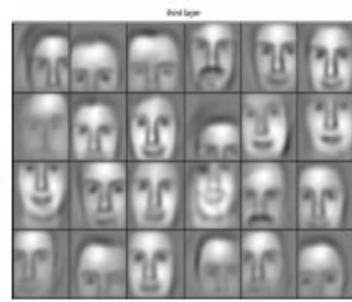
pixels



arestas



partes de objetos



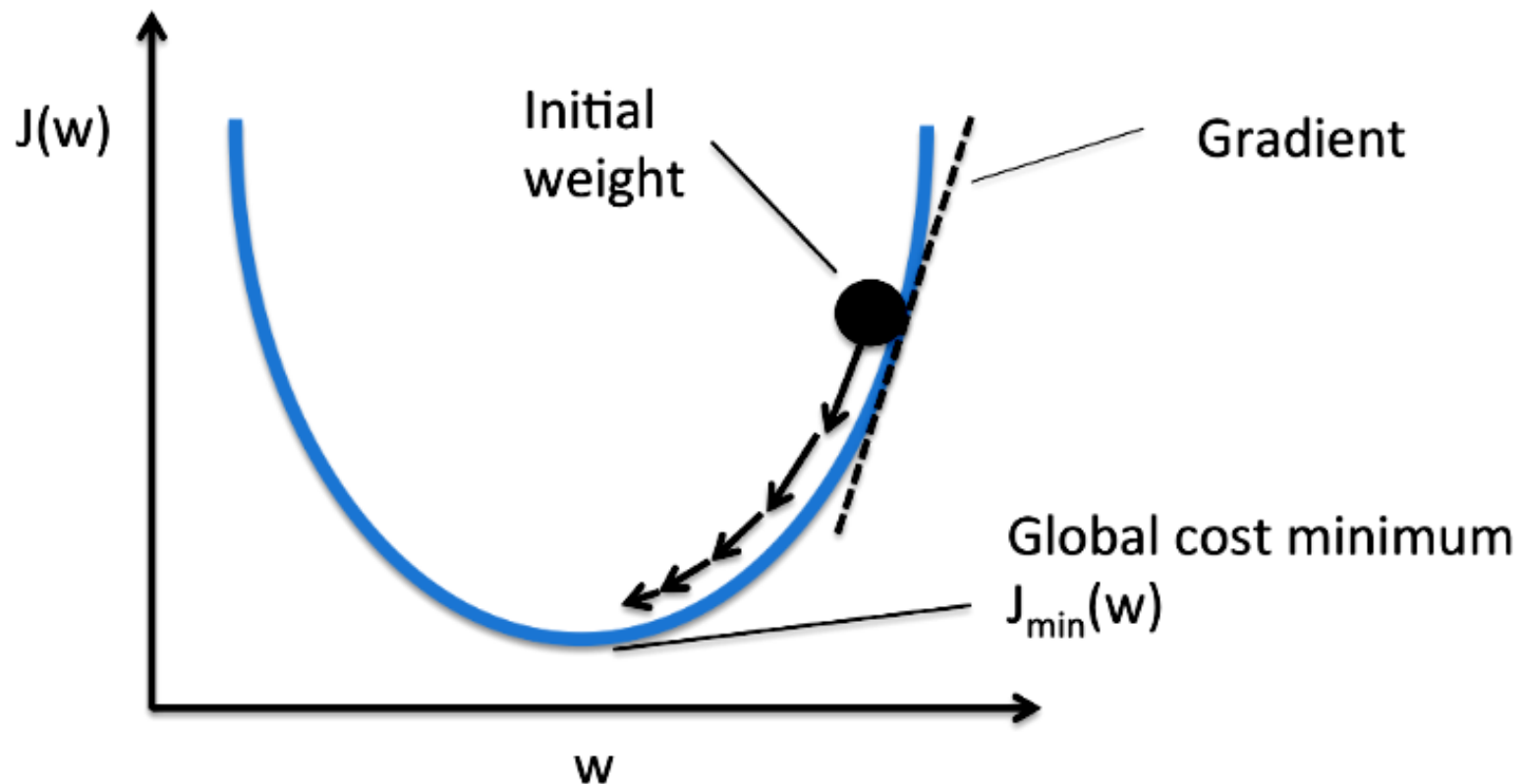
objetos



Principais RNs profundas

- Redes neurais profundas (RNP)
- Redes credais profundas (RCP)
- Redes autocodificadoras profundas (RAP)
- Redes neurais convolucionais profundas (RNCP)

Para iniciar o treinamento, vamos chutar alguns valores para os w s de cada neurônio. Em seguida, observaremos a previsão da rede neural em alguns dados, que, muito provavelmente, será péssima. Dessa forma, os w s iniciais serão associados a um alto custo ou a uma **região elevada na superfície de custo**. No treinamento então, vamos atualizar os w s de maneira iterativa, de forma a diminuir o custo. Isso é feito com a técnica de gradiente descendente estocástico, que pode ser entendida como uma descida na superfície de custo de uma tarefa de otimização.



Fonte

Métricas



Acurácia

- Quantos exemplos foram corretamente classificados
 - Avalia erro nas classes igualmente
- Pode não ser adequada para dados desbalanceados
 - Pode prejudicar desempenho para classe minoritária
 - Geralmente mais interessante que a classe majoritária



Classificação binária

- Classe de interesse é a classe positiva
- Dois tipos de erro:
 - Classificação de um exemplo N como P
 - Falso positivo (alarme falso)
 - Ex.: Diagnosticado como doente, mas está saudável
 - Classificação de um exemplo P como N
 - Falso negativo
 - Ex.: Diagnosticado como saudável, mas está doente



Desempenho preditivo

- Matriz de confusão (tabela de contingência) pode ser utilizada para distinguir os erros
 - Base de várias medidas
 - Pode ser utilizada com 2 ou mais classes

Classe verdadeira	Classe predita			
	1	2	3	
	1	25	0	5
	2	10	40	0
	3	0	0	20



Exemplo

- Matriz de confusão para 200 exemplos divididos em 2 classes

Classe verdadeira	Classe predita	
	p	n
P	70	30
N	40	60



Classe verdadeira	Classe predita	
	p	n
P	VP	FN
N	FP	VN



Medidas de avaliação

$$\text{Taxa de FP (TFP)} = \frac{FP}{FP + VN}$$

(Alarmes falsos)

Erro do tipo I

Classe verdadeira	Classe predita	
	p	n
P	VP	FN
N	FP	VN

$$\text{Taxa de FN (TFN)} = \frac{FN}{VP + FN}$$

Erro do tipo II

Classe verdadeira	Classe predita	
	p	n
P	VP	FN
N	FP	VN



Medidas de avaliação

$$\text{Taxa de FP (TFP)} = \frac{FP}{FP + VN}$$

(Alarmes falsos)

Custo

Classe verdadeira	Classe predita	
	p	n
P	VP	FN
N	FP	VN

$$\text{Taxa de VP (TVP)} = \frac{VP}{VP + FN}$$

Benefício

Classe verdadeira	Classe predita	
	p	n
P	VP	FN
N	FP	VN



Exemplo

$$\frac{VP}{VP + FN}$$

$$\frac{FP}{FP + VN}$$

■ Avaliação de 3 classificadores

Classe verdadeira	Classe predita	
	p	n
	P	N
P	20	30
N	15	35

Classificador 1
TVP =
TFP =

Classe verdadeira	Classe predita	
	p	n
	P	N
P	70	30
N	50	50

Classificador 2
TVP =
TFP =

Classe verdadeira	Classe predita	
	p	n
	P	N
P	60	40
N	20	80

Classificador 3
TVP =
TFP =



Exemplo

$$\frac{VP}{VP + FN}$$

$$\frac{FP}{FP + VN}$$

■ Avaliação de 3 classificadores

Classe verdadeira	Classe predita	
	p	n
P	20	30
N	15	35

Classificador 1
TVP = 0.4
TFP = 0.3

Classe verdadeira	Classe predita	
	p	n
P	70	30
N	50	50

Classificador 2
TVP = 0.7
TFP = 0.5

Classe verdadeira	Classe predita	
	p	n
P	60	40
N	20	80

Classificador 3
TVP = 0.6
TFP = 0.2



Medidas de avaliação

- Medidas frequentemente utilizadas

$$\text{TFP} = \frac{FP}{FP + VN}$$

(Erro tipo I)

$$\text{TFN} = \frac{FN}{VP + FN}$$

(Erro tipo II)

$$\text{Precisão} = \frac{VP}{VP + FP}$$

$$\text{Especificidade} = \frac{VN}{VN + FP} = 1 - \text{TFP}$$

$$\text{TVP} = \frac{VP}{VP + FN}$$

$$\text{Acurácia} = \frac{VP + VN}{VP + VN + FP + FN}$$

$$\text{Sensibilidade}$$
$$\text{Revocação (Recall)}$$

$$\text{Medida-F1} = \frac{2}{1/\text{prec} + 1/\text{rev}}$$



Revocação X Precisão

- Revocação (*recall*)

- Porcentagem de exemplos positivos classificados como positivos $\frac{VP}{VP + FN}$
 - Nenhum exemplo positivo é deixado de fora

- Precisão

- Porcentagem de exemplos classificados como positivos que são realmente positivos $\frac{VP}{VP + FP}$
 - Nenhum exemplo negativo é incluído



Sensibilidade X Especificidade

- Sensibilidade

- Porcentagem de exemplos positivos classificados como positivos
 - Igual a revocação

$$\frac{VP}{VP + FN}$$

- Especificidade

- Porcentagem de exemplos negativos classificados como negativos
 - Nenhum exemplo negativo é deixado de fora

$$\frac{VN}{VN + FP}$$



Medidas de avaliação

- Medida-F

- Média harmônica ponderada da precisão e da revocação

$$\frac{(1 + \alpha) \times (prec \times rev)}{\alpha \times prec + rev}$$

- Medida-F1

- Precisão e revocação têm o mesmo peso

$$\frac{2 \times (prec \times rev)}{prec + rev} = \frac{2}{1/prec + 1/rev}$$



Exemplo

- Seja um classificador com a seguinte matriz de confusão, definir:
 - Acurácia
 - Precisão
 - Revocação
 - Especificidade

Classe verdadeira	Classe predita	
	p	n
	P	70
N	40	60



Exemplo

$$\text{Acurácia} = \frac{VP + VN}{VP + VN + FP + FN}$$

$$\text{Precisão} = \frac{VP}{VP + FP}$$

$$\text{Revocação} = \frac{VP}{VP + FN}$$

$$\text{Especificidade} = \frac{VN}{VN + FP}$$

		Predito	
		p	n
Verdadeiro	P	VP	FN
	N	FP	VN
		p	n
	P	70	30
	N	40	60



Exemplo

$$\text{Acurácia} = \frac{VP + VN}{VP + VN + FP + FN} = (70 + 60) / (70 + 30 + 40 + 60) = 0.65$$

$$\text{Precisão} = \frac{VP}{VP + FP} = 70 / (70 + 40) = 0.64$$

$$\text{Revocação} = \frac{VP}{VP + FN} = 70 / (70 + 30) = 0.70$$

$$\text{Especificidade} = \frac{VN}{VN + FP} = 60 / (40 + 60) = 0.60$$

		Predito	
		p	n
Verdadeiro	P	VP	FN
	N	FP	VN

		p	
		p	n
Verdadeiro	P	70	30
	N	40	60