

Redes Neurais Artificiais - Perceptron Simples e Multilayer Perceptron

Profa. Dra. Sarajane Marques Peres

Março de 2020

Disciplina: Inteligência Artificial
Bacharelado em Sistemas de Informação
<http://www.each.usp.br/si>

Linha do Tempo

Nicolas Rashevsky

Biólogo teórico: lidera a Comunidade de Modelagem Neural na **Universidade de Chicago**

1938

McCulloch and Pitts

McCulloch → psiquiatra e neuroanatomista: 20 anos estudando como representar um evento do sistema nervoso
Pitts → matemático

1943

Primeiro modelo formal de um neurônio, seguindo a lei "ou tudo ou nada".

Uma rede de tais unidades seria capaz de computar qualquer função computável.

Nascimentos das disciplinas:
• Redes Neurais Artificiais
• Inteligência Artificial

Influenciou Von Neumann – EDVAC e ENIAC

Norbert Wiener

Escreveu o livro **Cybernetics**: conceitos de controle, comunicações e processamento de sinal

1948



Hebb

Escreveu o livro **The organization of Behavior**: regra de aprendizado para modificação sináptica

1949

Postulado do aprendizado: a efetividade de uma variável sináptica entre dois neurônios é fortalecida por repetidas ativações de um neurônio por outros, envolvidos naquela sinapse.

McCulloch, W.S., W. Pitts (1943) A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*. vol 5, pp. 115-133.

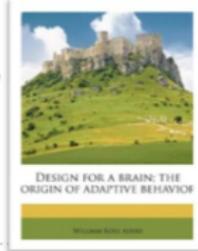
Linha do Tempo

Willian Ross Ashby:
escreve o livro *Design for a Brain: The Origin of Adaptive Behavior*

1952

Defende que o comportamento adaptativo não é nato, mas sim aprendido, e que ele pode mudar para melhor

Interpreta os aspectos dinâmicos de um organismo vivo como uma máquina



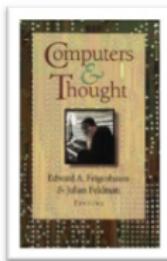
Minsky: escreve sua tese de doutorado na Universidade de Princeton.
Theory of Neural-Analog Reinforcement Systems and Its Application to the Brain-Modelo Problem.

1954

Cunhou o termo "redes neurais" em 1961 no artigo "Steps Toward Artificial Intelligence".

Minsky, M. L. (1961) Steps Toward Artificial Intelligence. Proceedings of the Institute of Radio Engineers, vol. 49, pp 8-30.

Gabor:
Pioneiro na teoria da comunicação, propôs a ideia de Filtros Adaptativos Não Lineares.



Feigenbaum, E. A. and Feldman, J.,eds. Computer and Thought. pp 406-450. McGraw-Hill

Linha do Tempo

Rochester, Holland, Haibt e Duda:
primeira tentativa de simular computacionalmente o postulado de aprendizado de Hebb

Uttley: resolve um problema de classificação binário.
Caianiello: desenvolve uma análise formal para modelo.

Taylor: propõe os primeiros conceitos de memória associativa
Steinbuch: matrix de aprendizagem

Rosemblatt: PERCEPTRON um novo método de aprendizado supervisionado (para reconhecimento de padrões)

1950s



Inserem o conceito de **inibição**.

Influenciam **Uttley**.

Matriz de aprendizagem: rede de *switches* interpostos entre arrays de sensores (*receptors*) e de motores (*effectors*)



Linha do Tempo

Widrow and Hoff:
proposição do *Least Mean Square (LMS) Algorithms*

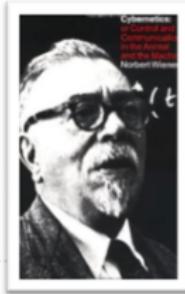
1960

Este algoritmo foi usado para formular a **ADALINE** (Adaptive Linear Element)

Segunda edição do livro
Cybernetics: inclui conceitos de aprendizado e auto organização; algumas menções relacionadas a mecânica estatística

1961

30 anos mais tarde
Hopfield sedimenta a relação entre mecânica estatística e aprendizado



Nilsson:
publica o livro: *Learning Machine*

1967

Discussão dos conceitos de:
•separabilidade linear
•hyper superfícies

Minsky e Papert: usam a matemática para demonstrar que existem limites fundamentais no que Perceptrons de uma única camada podem computar.

1969

Em uma breve seção da publicação, os autores falam sobre Perceptrons Multi-Camadas, mas colocam que não existe razão para assumir que as limitações do Perceptron de uma única camada poderiam ser resolvidas pela versão de múltiplas camadas.



Linha do Tempo

Alguns estudos sobre memória associativa.

Anderson, Kohonen e Nakano são alguns pesquisadores que investiram nesta área

1970s

Principal contribuição: memória de matriz de correlação baseada na **regra de aprendizado do PRODUTO INTERNO**.

Estudos referentes ao problema do Perceptron (computação de camadas escondidas) foram desenvolvidos.

Muitos deles sob a ótica de redes neurais recorrentes (atratores), hoje conhecidas como **Redes de Hopfield** (estabelecidas de fato nos anos 80).

Surge os **Mapas Auto Organizáveis** usando aprendizado competitivo.

von del Malsburg foi o primeiro a demonstrar a auto-organização

Grossberg estabeleceu de fato suas idéias sobre **Adaptive Resonance Theory (ART)**

Kohonen estabelece de fato a credibilidade dos Mapas Auto Organizáveis em 1982.

Linha do Tempo

O atraso dos anos 70

Segundo um pesquisador chamado Cowan, as razões foram:

- tecnológica: não havia computadores pessoais ou estações de trabalho para experimentação (Gabor e sua equipe precisaram de seis anos para construir um filtro com recursos analógicos);
- psicológica e financeira: o trabalho de Minsky e Papert certamente desencorajou pesquisadores e agências de fomento.

Linha do Tempo

Barto, Sutton e Anderson:
estabelecem de fato o **Aprendizado por Reforço**

Rumelhart, Hinton e Williams:
desenvolvem o **Back-Propagation Algorithm**

Broomhead e Lowe:
apresentam as redes neurais com **Radial Basis Functions (RBF)**

1983

1986

1988

• • •



Uma história cheia de descobertas independentes e de "descobertas" de trabalhos antigos que já haviam reportado alguma estratégia ou algoritmo.

Minsky - obituário by Wordsmith - um robô

Marvin Lee Minsky, 88, passed away January 24, 2016 in Boston, Massachusetts of cerebral hemorrhaging.

Born August 9, 1927 in New York City, New York, to parents Fannie Reiser and Henry Minsky, Marvin Minsky was known for his pioneer contribution to the field of artificial intelligence (AI). After graduating from Phillips Academy, Minsky attended Harvard University, graduating with a BA in Mathematics in 1950. He continued his education at Princeton University, ultimately graduating with a PhD in Mathematics in 1954.

Some of Minsky's greatest accomplishments include founding the MIT Computer Science and Artificial Intelligence Laboratory in 1959 and authoring many groundbreaking books in the field of artificial intelligence, including Perceptrons. He won many notable awards in his field of study, including the Turing Award in 1969.

Minsky is survived by his wife Gloria Minsky; three children, Margaret Minsky, Julie Minsky, and Henry Minsky.

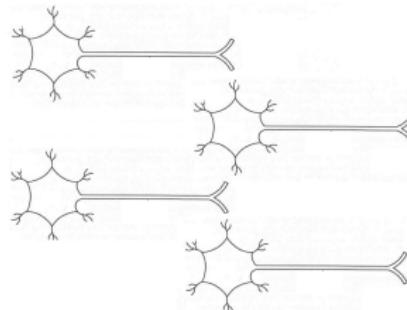
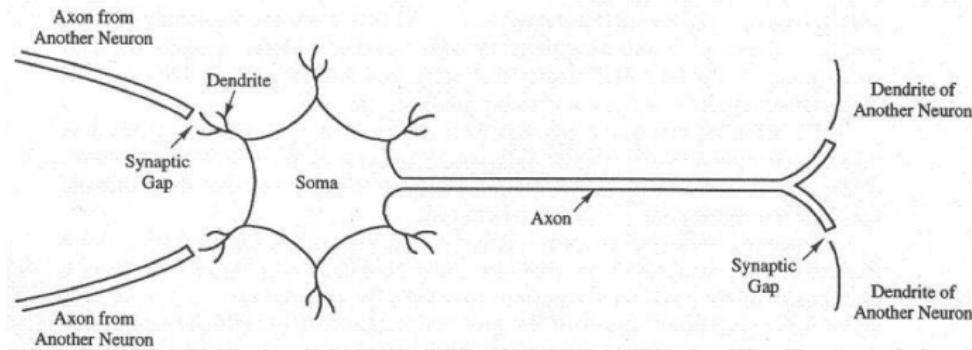
<https://automatedinsights.com/wordsmith>

Minsky - curiosidade

One of the earliest of his many collaborations was with the educationist Seymour Papert, the inventor of the Logo educational programming system widely used in schools. Minsky devised the “turtle” mechanism - a small robot equipped with a stylus - which extended Logo’s programming possibilities enormously, and enabled schoolchildren to draw spirals and complex patterns with simple commands. Minsky and Papert co-wrote Perceptrons (1969), an important early textbook that explained the potential and limits of early AI technology.

Minsky was a sociable individual and inspired many colleagues and students, who moved on to populate AI research and industry. Another of his collaborators, Ray Kurzweil, became a pioneer in the commercialisation of speech recognition and other technologies. A former student, Danny Hillis, founded the Thinking Machines Corporation in 1983. Minsky also had a long friendship with the physicist [Richard Feynman](#); and even helped Stanley Kubrick with his 1968 film *2001: A Space Odyssey*, for which he advised (somewhat optimistically) on the capabilities of the HAL computer.

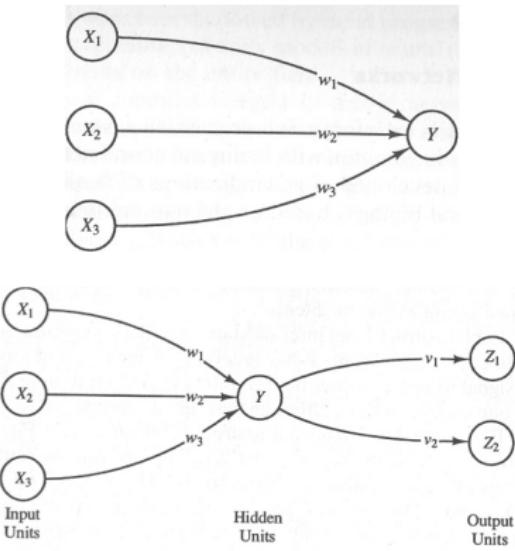




Neurônio e Rede Neural - Fausett

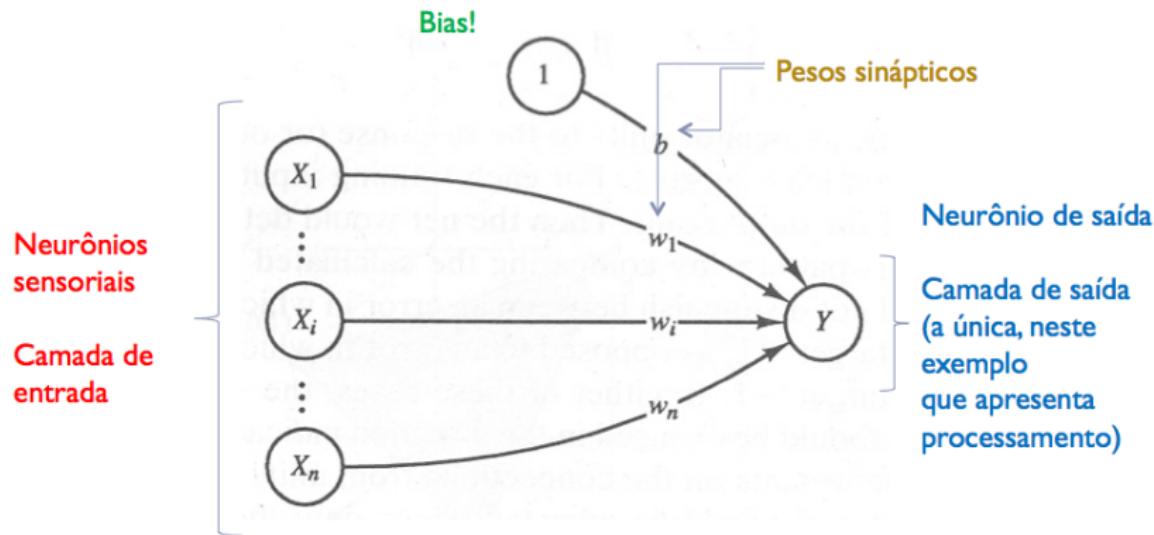


Existem diferentes tipos de neurônio: McCulloch-Pitts, Perceptron, Winner-Take-All, Rectified Linear Unit, Spiking Neurons ... Cada um deles segue procedimentos próprios e usa diferentes tipos de função para calcular sua ativação/saída.





Arquitetura





O processamento realizado por um neurônio j do tipo Perceptron segue as seguintes regras:

$$y_{\text{in}_j} = b_j + \sum_i x_i w_{ij}$$

$$y_j = f(y_{\text{in}_j})$$

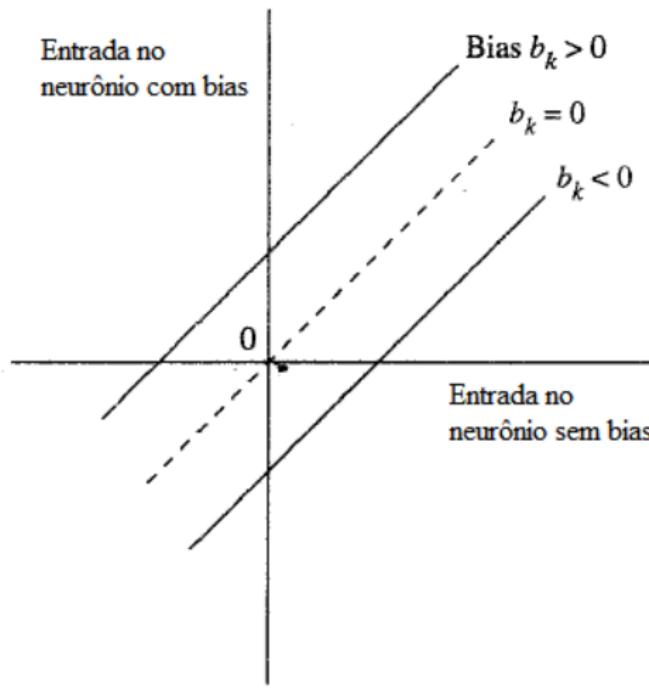
em que

- x_i são as entradas para o neurônio j ,
- w_{ij} são os pesos nas conexões entre cada entrada i e o neurônio j ,
- b_j é um *bias*,
- y_{in_j} é a entrada total no neurônio j ,
- $f(y_{\text{in}_j})$ é uma função de ativação e
- y_j é a saída no neurônio j .

Perceptron Simples - Haykin



Como resultado da existência do bias, o gráfico $v_k \times u_k$ não passa mais pela origem.



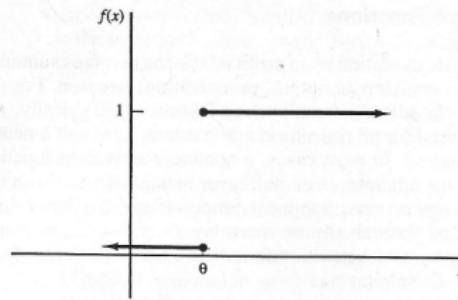


Função *step* (ou *threshold*) binária

$$f(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$$

Função *step* (ou *threshold*) binária com limiar

$$f(x) = \begin{cases} 1 & \text{if } x \geq \theta \\ 0 & \text{if } x < \theta \end{cases}$$





Função *step* bipolar

$$f(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ -1 & \text{if } x < 0 \end{cases}$$

Função *step* bipolar com limiar

$$f(x) = \begin{cases} 1 & \text{if } x \geq \theta \\ -1 & \text{if } x < \theta \end{cases}$$



O *bias*

O *bias* age exatamente como um peso em uma conexão de uma unidade cuja ativação é sempre 1. Aumentar o *bias* aumenta a entrada para o neurônio (e vice-versa). Se um *bias* é usado, tipicamente aplica-se a função *step* com decisão em 0.

O limiar

Se um limiar θ é usado na função de ativação *step*, é comum não usar o *bias*, pois para uma função *step* o *bias* e o limiar se tornam equivalentes.



Vamos usar como exemplo um neurônio Y do tipo Perceptron que possui as entradas x_1 , x_2 e 1, com seus respectivos pesos sinápticos w_1 , w_2 e b .

Assumindo a função *step* com decisão em 0

O limite entre os valores que x_1 e x_2 devem assumir para que o neurônio responda positivamente ou negativamente é a reta

$$b + x_1 w_1 + x_2 w_2 = 0,$$

ou^a,

$$x_2 = -\frac{w_1}{w_2} x_1 - \frac{b}{w_2}.$$

ou seja, para que o neurônio responda com saídas positivas, $b + x_1 w_1 + x_2 w_2 \geq 0$.

^a Na realidade, as operações que são de fato executadas não possuem as divisões, portanto não nos preocupamos com $w_2 = 0$.



Assumindo a função *step* com decisão em θ

O limite entre os valores que x_1 e x_2 devem assumir para que o neurônio responda positivamente ou negativamente é a reta

$$x_1 w_1 + x_2 w_2 = \theta,$$

ou^a,

$$x_2 = -\frac{w_1}{w_2} x_1 + \frac{\theta}{w_2}.$$

ou seja, para que o neurônio responda com saídas positivas, $x_1 w_1 + x_2 w_2 \geq \theta$.

^a Na realidade, as operações que são de fato executadas não possuem as divisões, portanto não nos preocupamos com $w_2 = 0$.

Ou seja, *bias* e limiar possuem o mesmo papel: determinam a altura da reta que separa as regiões de respostas positivas das regiões de respostas negativas.

Superfície de decisão do neurônio - Estude!!!

Neurônio

b	w1	w2
1,5	3	2

Reta

a	b
-1,5	-0,75

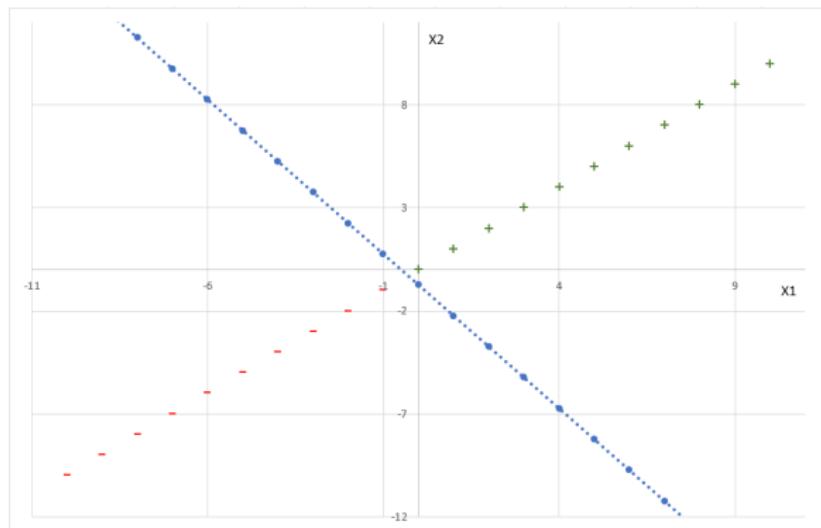
Definição do problema (entradas e saídas desejadas)

x1	x2	y	polaridade
-10	-10	-48,5	-
-9	-9	-43,5	-
-8	-8	-38,5	-
-7	-7	-33,5	-
-6	-6	-28,5	-
-5	-5	-23,5	-
-4	-4	-18,5	-
-3	-3	-13,5	-
-2	-2	-8,5	-
-1	-1	-3,5	-
0	0	1,5	+
1	1	6,5	+
2	2	11,5	+
3	3	16,5	+
4	4	21,5	+
5	5	26,5	+
6	6	31,5	+
7	7	36,5	+
8	8	41,5	+
9	9	46,5	+
10	10	51,5	+

Pontos para traçar a reta obtida após o treinamento

x1	x2
-10	14,25
-9	12,75
-8	11,25
-7	9,75
-6	8,25
-5	6,75
-4	5,25
-3	3,75
-2	2,25
-1	0,75
0	-0,75
1	-2,25
2	-3,75
3	-5,25
4	-6,75
5	-8,25
6	-9,75
7	-11,25
8	-12,75
9	-14,25
10	-15,75

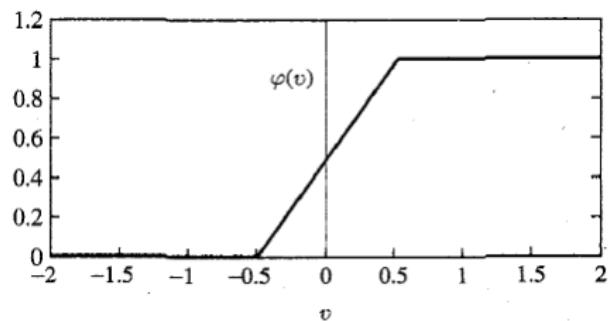
Superfície de decisão do neurônio - Estude!!!





Função linear por partes

$$\phi(v) = \begin{cases} 1 & \text{if } v \geq \frac{1}{2} \\ v & \text{if } -\frac{1}{2} < v < \frac{1}{2} \\ -1 & \text{if } v < -\frac{1}{2} \end{cases}$$

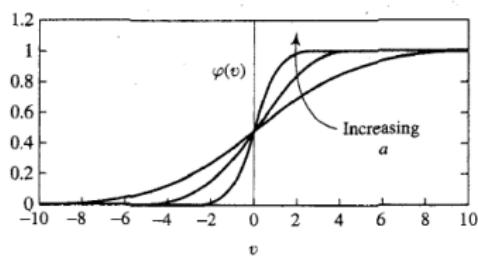




Função sigmoide

É a função mais comum e é um balanço entre um comportamento linear e um não linear. Um exemplo é a função logística: $\phi(v) = \frac{1}{1+\exp(-\alpha v)}$, em que α é um parâmetro de inclinação.

No limite ($\alpha \rightarrow \infty$) a função sigmóide se torna a função *threshold*. E a função sigmóide é derivável em todos os seus pontos - o que é importante para uma rede neural, como veremos mais à frente.



Função de saídas bipolares

É uma função de saída bipolar, desejável para expressar antissimetria com respeito à origem. Um exemplo é a função tangente hiperbólica.



Um modelo neuronal consiste de um combinador linear seguido de um limitador (a função de ativação). O nó soma do modelo neuronal computa uma combinação linear de entradas aplicadas para suas sinapses, e também incorpora um bias aplicado externamente. A soma resultante, isto é, o campo local induzido, é aplicada ao limitador. Então, o neurônio produz uma saída +1 se o limitador é positivo, -1 se ele é negativo.

A meta de um perceptron é classificar corretamente um conjunto de estímulos x_1, x_2, \dots, x_m em uma de duas classes \mathcal{C}_1 ou \mathcal{C}_2 .

Vejamos o problema do AND



Considere que a meta da rede neural é classificar cada padrão (dado ou exemplo) de entrada como **pertencente** ou **não pertencente** a uma classe particular.

- a **pertinência** é representada pela saída positiva no neurônio (+1)
- a **não pertinência** é representada pela saída negativa no neurônio (-1)

O problema pode ser representado com entradas binárias ou entradas bipolares, como mostrado abaixo:

$$1, 1 \rightarrow 1$$

$$1, 0 \rightarrow -1$$

$$0, 1 \rightarrow -1$$

$$0, 0 \rightarrow -1$$

$$1, 1 \rightarrow 1$$

$$1,-1 \rightarrow -1$$

$$-1, 1 \rightarrow -1$$

$$-1,-1 \rightarrow -1$$

O problema da porta lógica AND - Fausett



$$1, 1 \rightarrow 1$$

$$1, 0 \rightarrow -1$$

$$0, 1 \rightarrow -1$$

$$0, 0 \rightarrow -1$$

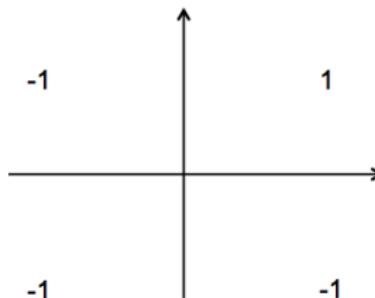
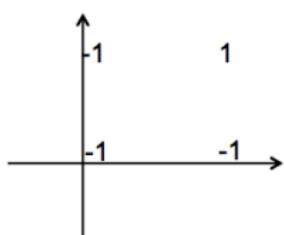
$$1, 1 \rightarrow 1$$

$$1, -1 \rightarrow -1$$

$$-1, 1 \rightarrow -1$$

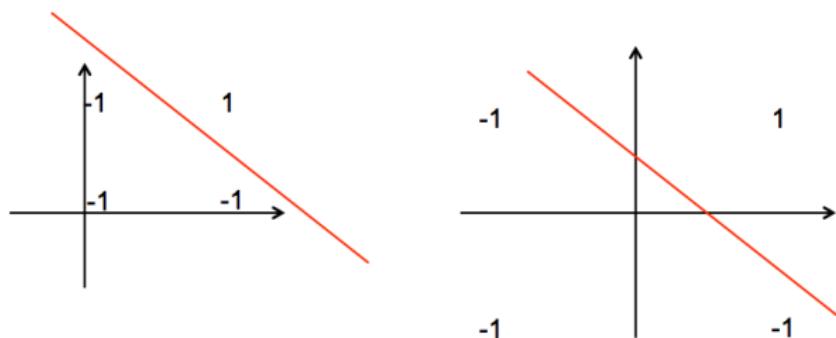
$$-1, -1 \rightarrow -1$$

Uma interpretação geométrica também é possível, e bastante pertinente para o nosso exemplo.





Para resolver esse problema de classificação (ou para implementar a porta lógica AND), precisamos estabelecer uma superfície (neste caso, uma reta) de decisão.



Ou seja, se usarmos um neurônio do tipo Perceptron para resolver esse problema, o neurônio representará essa reta de decisão. **Como?**



Um neurônio Y do tipo Perceptron modelado para o problema do AND terá:

- duas entradas, x_1 e x_2 , representando os valores possíveis de entrada na porta lógica AND. Uma das duas representações (binária ou bipolar) devem ser usadas;
- uma entrada sempre ativa, se o bias for usado;
- pesos sinápticos, w_1 e w_2 , para ponderar os sinais de cada uma das entradas, e um peso b se o bias estiver sendo usado;
- uma função de ativação que permita duas saídas (-1 e 1), sendo a função *step* uma função adequada (com decisão em 0 se o bias estiver sendo usado, ou com decisão em θ).

O problema é ???

Se nós conhecemos os valores que x_1 e x_2 podem assumir e se nós conhecemos os valores que Y deveria assumir para cada um dos pares de valores de entrada, nosso problema é encontrar os valores de w_1 , w_2 e b (se o bias estiver ser usado) que permitam posicionar a reta de forma que as respostas desejadas para Y sejam obtidas, considerando a função de ativação de Y .



Treinamento

Processo iterativo por meio do qual os valores para os pesos sinápticos são obtidos.
Trata-se da implementação de um processo de aprendizado indutivo.

Raciocínio indutivo

Observa: $p(a,b)$, $p(a,d)$, $p(d,e)$, $p(d,g)$, $p(e,f)$, $a(a,e)$, $a(d,f)$

Aprende: $p(X,Y) \wedge p(Y,Z) \rightarrow a(X,Z)$

Aprendizado supervisionado

Quando o processo de aprendizado é guiado pelas respostas desejadas, diz-se que ele é supervisionado.



- Inicialize os pesos e *bias* – por simplicidade, inicialize-os com 0 ou com números aleatórios no intervalo $[0, 1]$ ou $[-1, 1]$;
- Determine a **taxa de aprendizado** α como um número no intervalo $(0, 1]$;
- Enquanto condição de parada é falsa, faça
 - Para cada par de treinamento $s : t$, faça
 - Determine as ativações das unidades de entrada: $x_i = s_i$;
 - Compute a resposta do neurônio de saída;
 - Atualize os pesos e *bias* se ocorrer um erro para o par de treinamento atual;
 - Teste a condição de parada: se nenhum peso mudou na **época**, pare; senão continue



Compute a resposta do neurônio de saída

$$y_in_j = b_j + \sum_i x_i w_{ij}$$

$$y_j = f(y_in_j)$$

sendo que $f(y_in_j)$ pode ser a função step binária ou bipolar, a depender de como modelamos o problema, ou uma outra função alternativa que mantenha as propriedades de comportamento do neurônio Perceptron.

Função usada no exemplo (teste de mesa) do livro da Fausett (páginas 61 – 76 - CONSULTE):

$$y = \begin{cases} 1 & \text{if } y_in > \theta \\ 0 & \text{if } -\theta \leq y_in \leq \theta \\ -1 & \text{if } y_in < -\theta \end{cases}$$



Atualize os pesos e *bias* se ocorrer um erro para o par de treinamento atual

Se $y \neq t$,

$$w_i(\text{new}) = w_i(\text{old}) + \alpha t x_i,$$

$$b(\text{new}) = b(\text{old}) + \alpha t,$$

senão

$$w_i(\text{new}) = w_i(\text{old}),$$

$$b(\text{new}) = b(\text{old}).$$

Algoritmo para treinamento do Perceptron Simples - Estude!!!

Analisando tx_i na regra de aprendizado. Olhando para representação e respostas bipolares. Se:

- $y = 1$ e $t = 1$: não há alteração nos pesos
- $y = -1$ e $t = -1$: não há alteração nos pesos
- $y = 1$ e $t = -1$: o sinal que entra no neurônio está alto (acima de 0 ou de θ), e precisa diminuir.

Como $t = -1$ a alteração nos pesos será

- **negativa** para neurônios de entrada que possuem valores positivos ($x_i = 1$), diminuindo o valor do peso que vai então enfraquecer a entrada positiva na próxima iteração.
- **positiva** para neurônios de entrada que possuam valores negativos ($x_i = -1$), aumentando o valor do peso que vai então fortalecer a entrada negativa na próxima iteração.
- $y = -1$ e $t = 1$: o sinal que entra no neurônio está baixo (abaixo de 0 ou de θ), e precisa aumentar.

Como $t = 1$ a alteração nos pesos será

- **positiva** para neurônios de entrada que possuem valores positivos ($x_i = 1$), aumentando o valor do peso que vai então fortalecer a entrada positiva na próxima iteração.
- **negativa** para neurônios de entrada que possuam valores negativos ($x_i = -1$), diminuindo o valor do peso que vai então enfraquecer a entrada negativa na próxima iteração.



Algoritmo para treinamento do Perceptron Simples - Estude!!!

Estudando o caso (apenas verificando toda as possibilidades e olhando para a alteração de pesos a fim de entender o seu efeito):

- $y = 1$ e $t = -1$: o sinal de entrada no neurônio **está alto** (acima de 0 ou de θ), e **precisa diminuir**. Como $t = -1$ a alteração nos pesos será
 - **negativa** para neurônios de entrada que possuem valores positivos ($x_i = 1$), diminuindo o valor do peso que vai então enfraquecer a entrada positiva na próxima iteração.
 - **positiva** para neurônios de entrada que possuam valores negativos ($x_i = -1$), aumentando o valor do peso que vai então fortalecer a entrada negativa na próxima iteração.

a = 1							
saída (y)	saída esperada (t)	entrada (x)	peso atual (w(old))	y-in (agora)	alteração (a t x)	peso novo (w(new))	y_in (depois)
1	-1	1	0,5	0,5	-1	-0,5	-0,5
			-0,5	-0,5	-1	-1,5	-1,5
		-1	0,5	-0,5	1	1,5	-1,5
			-0,5	0,5	1	0,5	-0,5

a = 0,2							
saída (y)	saída esperada (t)	entrada (x)	peso atual (w(old))	y-in (agora)	alteração (a t x)	peso novo (w(new))	y_in (depois)
1	-1	1	0,5	0,5	-0,2	0,3	0,3
			-0,5	-0,5	-0,2	-0,7	-0,7
		-1	0,5	-0,5	0,2	0,7	-0,7
			-0,5	0,5	0,2	-0,3	0,3

Algoritmo para treinamento do Perceptron Simples - Estude!!!

Veja uma simulação ilustrativa em:

<https://www.youtube.com/watch?v=vGwemZhPlsA>.

Faça o teste de mesa para a porta lógica AND e para a porta lógica OR.



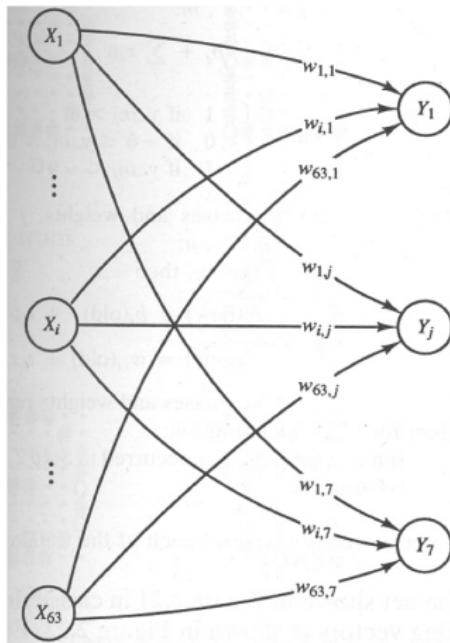
- Admita os pesos e *bias* obtidos no procedimento de aprendizado.
 - Para cada padrão de teste s , faça
 - Determina as ativações das unidades de entrada: $x_i = s_i$;
 - Compute a resposta do neurônio de saída;



Redes neural artificial

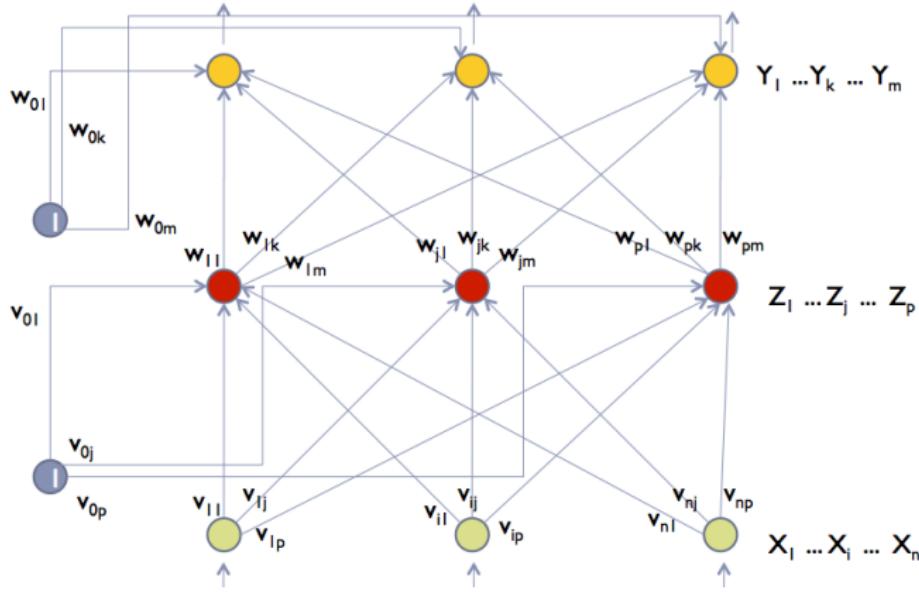
É um sistema de processamento de informação que tem algumas características em comum com as redes neurais biológicas.

- São desenvolvidas como uma generalização de modelos matemáticos da cognição humana e da biologia neural, com base nos seguintes princípios:
 - O processamento da informação ocorre em vários (muitos) elementos simples chamados neurônios.
 - Sinais são transmitidos entre neurônios por meio de links (conexões).
 - Cada link (conexão) tem um peso associado, pelo qual – em uma rede neural típica – sinais são transmitidos.
 - Cada neurônio aplica uma função de ativação (usualmente não linear) em sua entrada para determinar seu sinal de saída.



Uma rede neural de camada única com neurônios do tipo Perceptron.

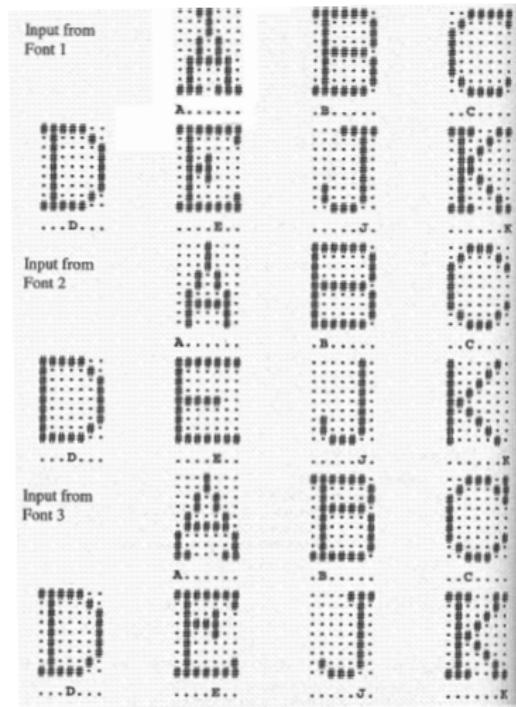
Multiplas camadas - Fausett



Arquitetura para um Multilayer Perceptron (MLP).

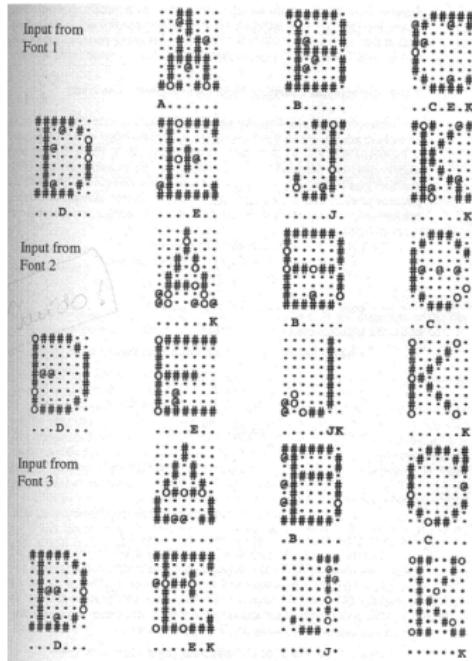


... como um problema de classificação resolvido com uma rede neural Perceptron Simples: **conjunto de treinamento**.



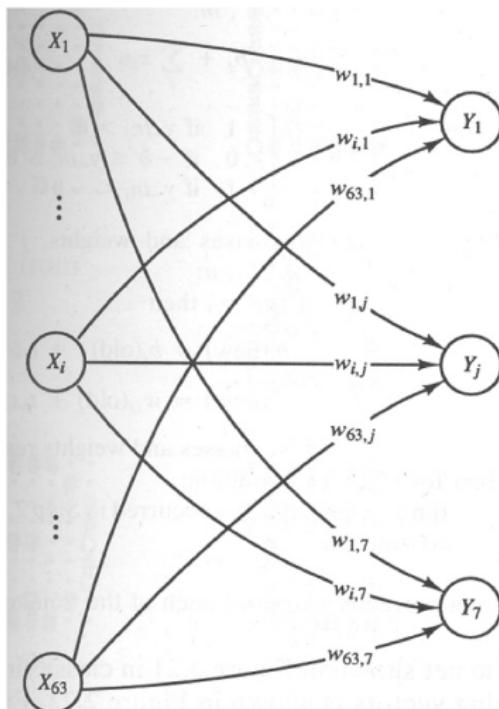


... como um problema de classificação resolvido com uma rede neural Perceptron Simples: **conjunto de teste**.





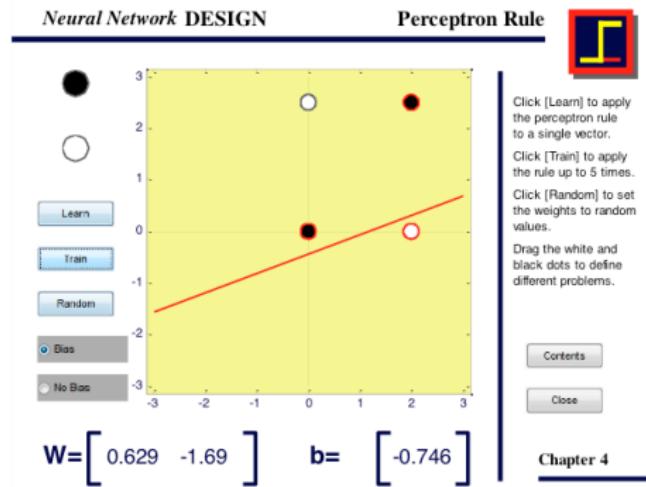
... como um problema de classificação resolvido com uma rede neural Perceptron Simples: **arquitetura**.



Exemplos no Matlab

Simulações no Matlab:

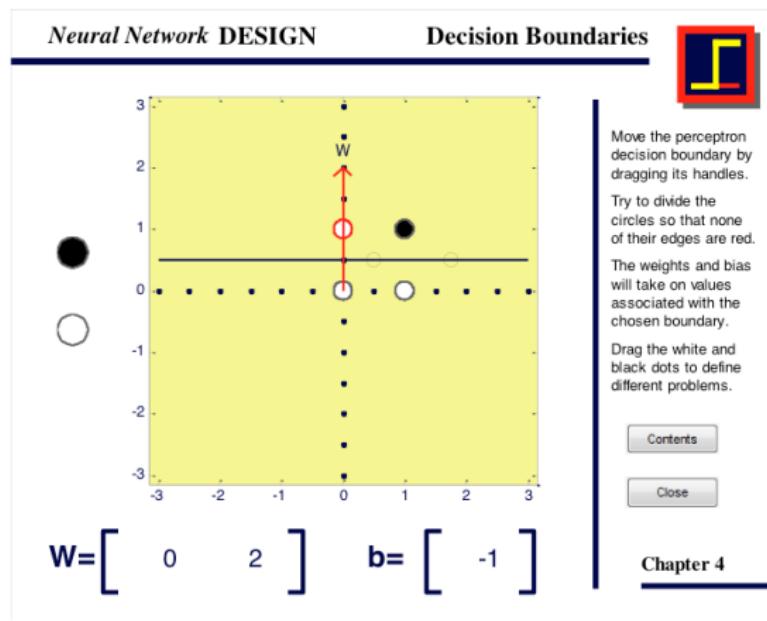
- nnd: carrega uma interface gráfica para chamar diferentes simulações;
- nnd4db: mostra a plotagem da reta de divisão do espaço e os valores dos pesos e bias referentes à reta.



Exemplos no Matlab

Simulações no Matlab:

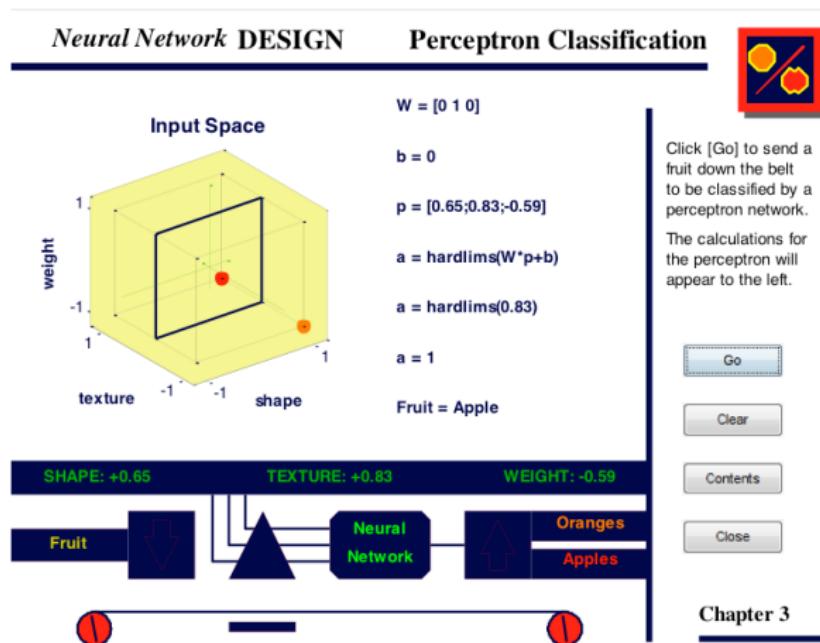
- nnd4pr: mostra a plotagem da reta de divisão do espaço e os valores dos pesos e bias referentes a ela durante o processo de treinamento.



Exemplos no Matlab

Simulações no Matlab:

- nnd3pc: classificador de maçãs e laranjas (espaço de características tri-dimensional)

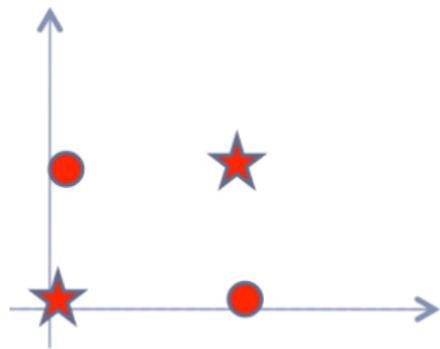


Separabilidade Linear - Fausett



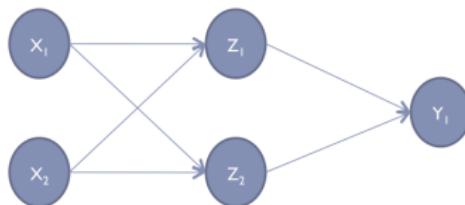
O neurônio do tipo Perceptron, com função de ativação *step* (linear), como estudado até aqui, só é capaz de resolver problemas linearmente separáveis, como a porta lógica AND e a porta lógica OR.

O que fazer, por exemplo, para resolver a porta lógica XOR?





As camadas escondidas extraem, progressivamente, informações dos dados de entradas. Veja o que acontece com os dados do problema XOR quando passam por uma camada escondida em uma rede multicamadas com neurônios McCulloch-Pitts. Pesos adequados precisam ser atribuídos para as sinapses. Vamos abstrair esse detalhe.



Entradas para a rede, e ações que deverão ser executadas na aplicação da rede.

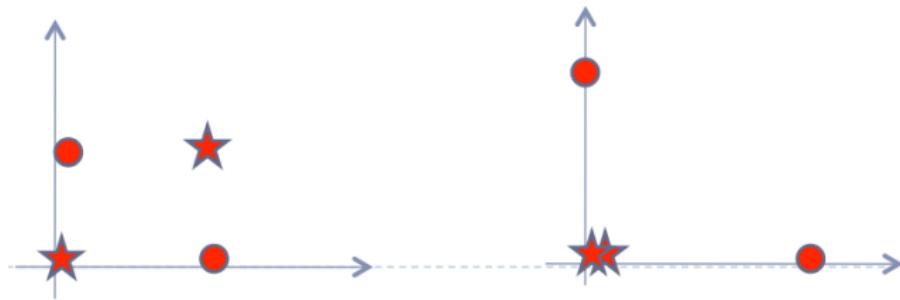
x1	x2	z1	z2	Y
0	0			
0	1			
1	0			
1	1			



Após a aplicação da rede.

x1	x2	z1	z2	Y
0	0	0	0	0
0	1	0	1	1
1	0	1	0	1
1	1	0	0	0

Efeito da camada escondida: **neste caso**, transformou o problema é um problema linearmente separável.





Uma rede neural Multilayer Perceptron (MLP) é tipicamente composta de:

- um conjunto de neurônios sensoriais (ou nós fonte) que constitui a **camada de entrada** da rede;
- uma ou mais **camadas escondidas** de neurônios (Perceptron) que fazem processamento de sinal a partir de funções de ativação não lineares (diferenciáveis em todos os seus pontos);
- a **camada de saída** da rede com neurônios (Perceptron) que fazem processamento de sinal a partir de funções de ativação não lineares ou funções de ativação lineares (diferenciáveis em todos os seus pontos).

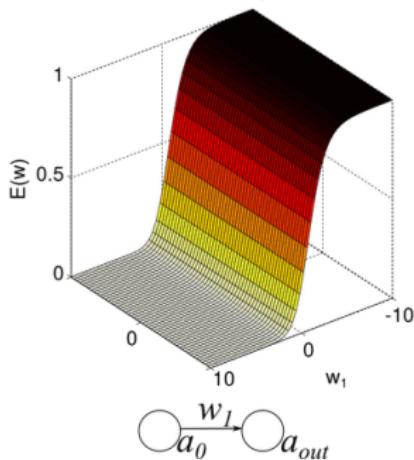
Supervisão, com minimização do erro

É uma rede neural com treinamento supervisionado, comumente treinada com o algoritmo de aprendizado supervisionado backpropagation (retro-propagação do erro) – um algoritmo baseado na regra de aprendizado de correção do erro (regra delta generalizada).

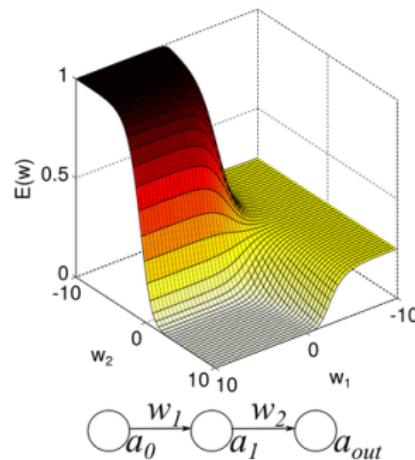
Superfície de erro ...



Error Surface: 1-layer Network



Error Surface: 2-layer Network

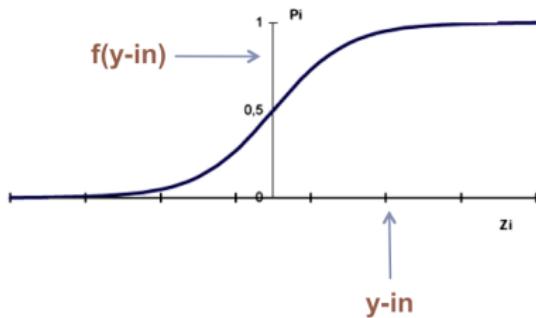


<https://theclevermachine.wordpress.com/2014/09/11/a-gentle-introduction-to-artificial-neural-networks/>



Multilayer Perceptron - Fausett

O modelo de cada neurônio na rede possui uma função de ativação, geralmente não linear, diferenciável em todos os seus pontos. Comumente é usada uma função logística (sigmoide binária, ou sigmóide bipolar ou tangente hiperbólica).



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

$$f'(x) = f(x)[1 - f(x)]$$

Multilayer Perceptron - Fausett



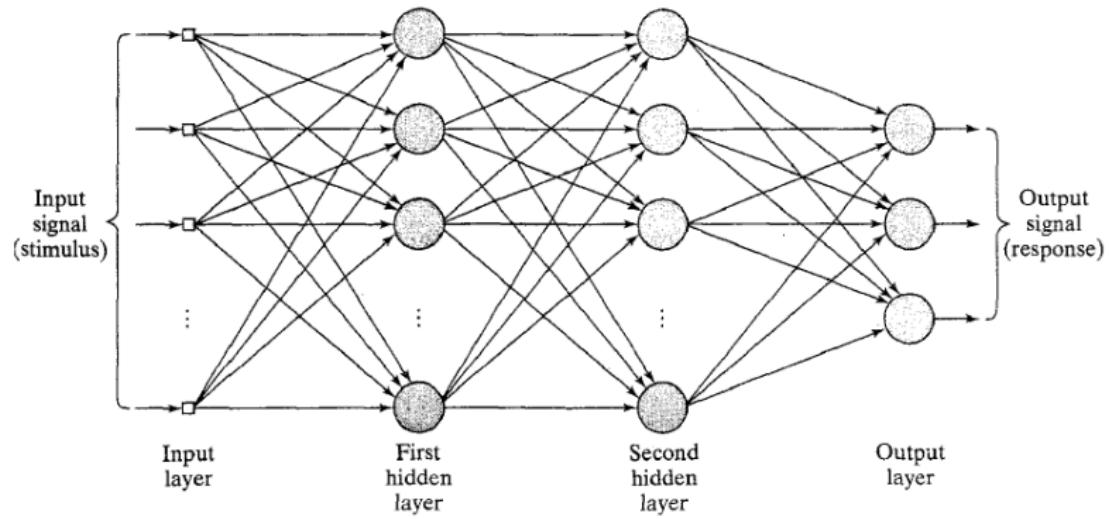
O treinamento envolve três estágios:

- a passagem (feedforward) dos dados de treinamento;
- o cálculo e retropropagação do erro associado à cada neurônio;
- o ajuste de pesos.

Os neurônios fazem dois tipos de processamento:

- o processamento clássico: ativação do neurônio mediante entradas e uma função de ativação;
- o cálculo da informação de erro - computação do gradiente

Multilayer Perceptron - arquitetura - Haykin



O algoritmo Backpropagation - Haykin

O sinal de erro na saída do **neurônio j** na iteração¹ n (considerando um dado de treinamento) é:

$$e_j(n) = d_j(n) - y_j(n).$$

O seu valor instantâneo do erro é

$$\frac{1}{2} e_j^2(n).$$

Assim, o valor instantâneo de erro $\mathcal{E}(n)$ (total) é obtido somando todos os erros na camada de saída:

$$\mathcal{E}(n) = \frac{1}{2} \sum_{j \in C} e_j^2(n)$$

em que o conjunto C inclui todos os neurônios da camada de saída. **São os únicos neurônios visíveis na rede.** O erro quadrado médio é:

$$\mathcal{E}_{av} = \frac{1}{N} \sum_{n=1}^N \mathcal{E}(n).$$

¹ N é o tamanho do conjunto de treinamento e n é o n^{th} dado do conjunto.

O algoritmo Backpropagation - Haykin

$$\mathcal{E}_{av} = \frac{1}{N} \sum_{n=1}^N \mathcal{E}(n).$$

Esse erro é uma função de todos os parâmetros livres da rede (pesos e bias). \mathcal{E}_{av} é a **função custo** para um conjunto de treinamento, e é uma medida de desempenho do aprendizado.

Objetivo

O objetivo do aprendizado é minimizar essa função custo ajustando os parâmetros livres da rede.

Para isso, considere:

- alteração de pesos “padrão a padrão”;
- uma época como a apresentação completa do conjunto de treinamento;
- ajuste de peso feito de acordo com os erros computados para cada padrão apresentado à rede.

O algoritmo Backpropagation - Haykin

Sabendo que o campo local induzido de cada **neurônio j** é:

$$v_j(n) = \sum_{i=0}^m w_{ji}(n)y_i(n),$$

com $y_i(n)$ sendo a saída do neurônio i da camada anterior, ou o sinal vindo da camada de entrada, e m sendo o número de entradas do **neurônio j** . A saída do neurônio j é

$$y_j(n) = \phi(v_j(n)),$$

e a correção $\Delta w_{ji}(n)$ é aplicada ao peso $w_{ji}(n)$, a qual é proporcional à derivada parcial $\frac{\partial \mathcal{E}(n)}{\partial w_{ji}(n)}$. % ... estamos guiando a busca pelo mínimo custo.

$$\frac{\partial \mathcal{E}(n)}{\partial w_{ji}(n)} = \frac{\partial \mathcal{E}(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{ij}(n)}.$$

O algoritmo Backpropagation - Haykin

Dado que

$$\mathcal{E}(n) = \frac{1}{2} \sum_{j \in C} e_j^2(n)$$

a derivada com respeito a $e_j(n)$ é:

$$\frac{\partial \mathcal{E}(n)}{\partial e_j(n)} = \textcolor{red}{e_j(n)}$$

Dado que

$$e_j(n) = d_j(n) - y_j(n).$$

a derivada com respeito a $y_j(n)$ é:

$$\frac{\partial e_j(n)}{\partial y_j(n)} = \textcolor{red}{-1}.$$

O algoritmo Backpropagation - Haykin

Dado que

$$y_j(n) = \phi(v_j(n)),$$

a derivada com respeito a $v_j(n)$ é

$$\frac{\partial y_j(n)}{\partial v_j(n)} = \phi'_j(v_j(n))$$

Dado que

$$v_j(n) = \sum_{i=0}^m w_{ji}(n)y_i(n),$$

a derivada com respeito a $w_{ji}(n)$ é

$$\frac{\partial v_j(n)}{\partial w_{ji}(n)} = y_i(n)$$

O algoritmo Backpropagation - Haykin

Aplicando tudo em $\frac{\partial \mathcal{E}(n)}{\partial w_{ji}(n)}$ temos:

$$\begin{aligned}\frac{\partial \mathcal{E}(n)}{\partial w_{ji}(n)} &= e_j(n) * (-1) * \phi'_j(v_j(n)) * y_i(n) \\ &= -e_j(n) * \phi'_j(v_j(n)) * y_i(n)\end{aligned}$$

Assim, a correção $\Delta w_{ji}(n)$ aplicada a $w_{ji}(n)$ é, pela regra Delta:

$$\Delta w_{ji}(n) = -\eta \frac{\partial \mathcal{E}(n)}{\partial w_{ji}(n)}$$

sendo que η é a taxa de aprendizado, e o sinal negativo indica que a direção de ajuste dos pesos reduz $\mathcal{E}(n)$.

O algoritmo Backpropagation - Haykin

Então, das duas últimas equações temos que:

$$\Delta w_{ji}(n) = \eta \delta_j(n) y_i(n)$$

em que o **gradiente local** $\delta_j(n)$ é

$$\delta_j(n) = -\frac{\partial \mathcal{E}(n)}{\partial v_j(n)}$$

$$\begin{aligned} &= -\frac{\partial \mathcal{E}(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \\ &= -(e_j(n) * (-1) * \phi'_j(v_j(n))) \\ &= e_j(n) * \phi'_j(v_j(n)) \end{aligned}$$

Ou seja: o gradiente local $\delta_j(n)$ para o neurônio de saída j é igual ao produto do erro $e_j(n)$ pela função de ativação no campo induzido do neurônio ($\phi'_j(v_j(n))$).

O algoritmo Backpropagation - Haykin

Existem dois casos distintos para aplicação da regra de ajuste de pesos:

- 1 o caso do neurônio de saída para o qual existe uma saída desejada;
- 2 o caso do neurônio escondido para o qual não existe uma saída desejada, mas que tem uma responsabilidade parcial na produção do erro cometido na camada de saída.

Caso 1: o neurônio j é um neurônio de saída

É possível calcular $e_j(n)$ e a aplicação da regra para encontrar Δw_{ji} é direta.

Caso 2: o neurônio j é um neurônio escondido

O sinal de erro para o neurônio escondido teria que ser determinado recursivamente em termos dos sinais de erro de todos os neurônios aos quais o neurônio escondido está diretamente conectado.

O algoritmo Backpropagation - Haykin

O gradiente local $\delta_j(n)$ para um neurônio escondido j é:

$$\begin{aligned}\delta_j(n) &= -\frac{\partial \mathcal{E}(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \\ &= -\frac{\partial \mathcal{E}(n)}{\partial y_j(n)} \phi'_j(v_j(n)).\end{aligned}$$

Observe que a função custo é derivada em relação à saída, pois não existe um “erro local” a partir do qual a função custo possa ser derivada. Então, para calcular a derivada parcial $\frac{\partial \mathcal{E}(n)}{\partial y_j(n)}$, considera-se:

$$\mathcal{E}(n) = \frac{1}{2} \sum_{k \in C} e_k^2(n),$$

em que k é um neurônio na cada de saída, e $k \in C$ significa considerar todos os neurônios na camada de saída.

O algoritmo Backpropagation - Haykin

Derivando os dois lados da equação $\mathcal{E}(n) = \frac{1}{2} \sum_{k \in C} e_k^2(n)$ com respeito a $y_j(n)$, temos

$$\frac{\partial \mathcal{E}(n)}{\partial y_j(n)} = \sum_k e_k(n) \frac{\partial e_k(n)}{\partial y_j(n)}$$

e resolvendo $\frac{\partial e_k(n)}{\partial y_j(n)}$ temos

$$\frac{\partial e_k(n)}{\partial y_j(n)} = \sum_k e_k(n) \frac{\partial e_k(n)}{\partial v_k(n)} \frac{\partial v_k(n)}{\partial y_j(n)}$$

O algoritmo Backpropagation - Haykin

Dado que

$$e_k(n) = d_k(n) - y_k(n) = d_k(n) - \phi_k(v_k(n))$$

então, a derivada com respeito a $v_k(n)$ é

$$\frac{\partial e_k(n)}{\partial v_k(n)} = -\phi'_k(v_k(n)).$$

Dado que

$$v_k(n) = \sum_{j=0}^m w_{kj}(n)y_j(n)$$

em que m é o número de entradas em k , derivando com respeito a $y_j(n)$

$$\frac{\partial v_k(n)}{\partial y_j(n)} = w_{kj}(n).$$

O algoritmo Backpropagation - Haykin

Substituindo em

$$\begin{aligned}\frac{\partial e_k(n)}{\partial y_j(n)} &= \sum_k e_k(n) \frac{\partial e_k(n)}{\partial v_k(n)} \frac{\partial v_k(n)}{\partial y_j(n)} \\ &= \sum_k \delta_k(n) * -\phi'_k(v_k(n)) * w_{kj}(n) \\ &= - \sum_k \delta_k(n) w_{kj}(n)\end{aligned}$$

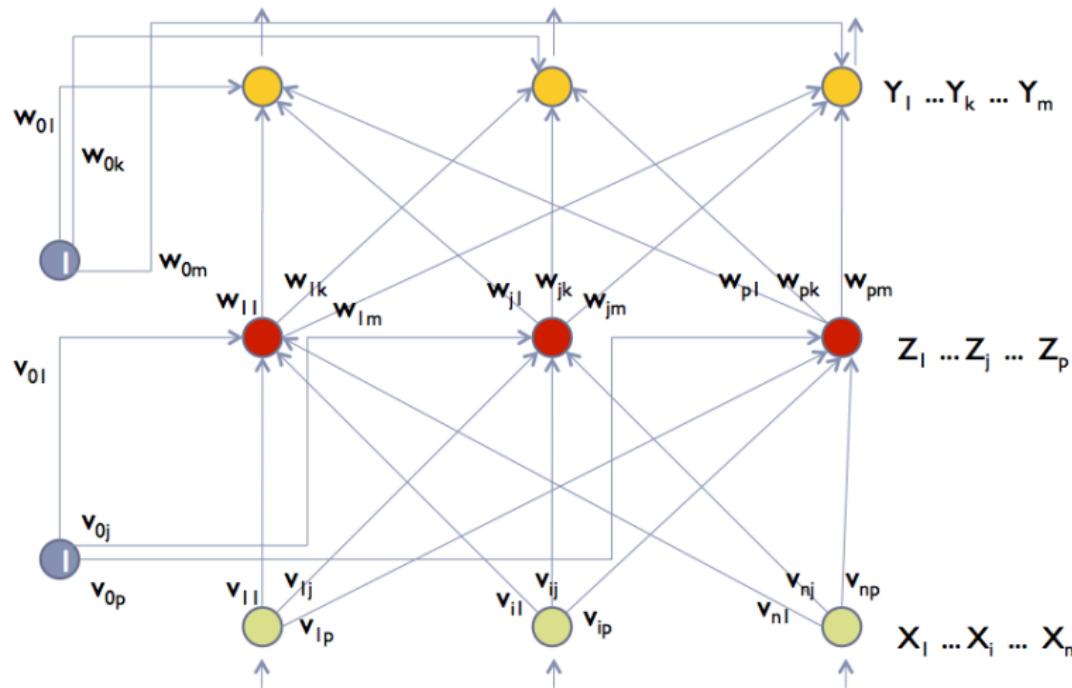
Finalmente, lembrando que:

$$\delta_j(n) = -\frac{\partial \mathcal{E}(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)}$$

e substituindo as derivadas parciais, temos:

$$= -(- \sum_k \delta_k(n) w_{kj}(n)) * \phi'_j(v_j(n)) \quad (1)$$

$$= \sum_k \delta_k(n) w_{kj}(n) * \phi'_j(v_j(n)) \quad (2)$$





- **Passo 0:** Inicializa pesos, bias, taxa de aprendizado, número de épocas, etc.
- **Passo 1:** Enquanto a condição de parada é falsa, execute mais uma época.
 - **Passos 3, 4, 5:** estágio feedforward;
 - **Passos 6, 7:** estágio de retro-propagação do erro;
 - **Passo 8:** estágio de atualização de pesos;
- **Passo 9:** Teste condição de parada (erro, taxa de aprendizado ou número de épocas).



- **Passo 3:** Cada unidade de entrada ($X_i, i = 1..n$) recebe um sinal de entrada x_i e o dissipa para todas as unidades na próxima camada.
- **Passo 4:** Cada unidade escondida ($Z_j, j = 1..p$) soma suas entradas ponderadas, aplica a função de ativação para computar seu sinal de saída, e o envia para as unidades da próxima camada.
- **Passo 5:** Cada unidade de saída ($Y_k, k = 1..m$) soma suas entradas ponderadas, aplica a função de ativação para computar seu sinal, de saída.



- **Passo 3:** Cada unidade de entrada ($X_i, i = 1..n$) recebe um sinal de entrada x_i e o dissipa para todas as unidades na próxima camada.
- **Passo 4:** Cada unidade escondida ($Z_j, j = 1..p$) soma suas entradas ponderadas, aplica a função de ativação para computar seu sinal de saída, e o envia para as unidades da próxima camada.

$$z_{\text{in}_j} = v_{0j} + \sum_{i=1}^n x_i v_{ij} \text{ e } z_j = f(z_{\text{in}_j})$$

- **Passo 5:** Cada unidade de saída ($Y_k, k = 1..m$) soma suas entradas ponderadas, aplica a função de ativação para computar seu sinal, de saída.

$$y_{\text{in}_k} = w_{0k} + \sum_{j=1}^p z_j w_{jk} \text{ e } y_k = f(y_{\text{in}_k})$$



- **Passo 6:** Cada unidade de saída ($Y_k, k = 1..m$) considera sua saída e a saída esperada para o dado de entrada para então computar o termo de informação de erro δ_k . Então calcula a correção de pesos e bias (Δw_{jk} e Δw_{0k}) e envia o termo de correção de erro para a camada abaixo (anterior).
- **Passo 7:** Cada unidade de saída ($Z_j, j = 1..p$) soma suas entradas δ_k (as informações de erro vindas da camada acima (posterior)).



- **Passo 6:** Cada unidade de saída ($Y_k, k = 1..m$) considera sua saída e a saída esperada para o dado de entrada para então computar o termo de informação de erro δ_k . Então calcula a correção de pesos e bias (Δw_{jk} e Δw_{0k}) e envia o termo de correção de erro δ_k para a camada abaixo (anterior).

$$\delta_k = (t_k - y_k) f'(y_{\text{in}_k})$$

$$\Delta w_{jk} = \alpha \delta_k z_j \text{ e } \Delta w_{0k} = \alpha \delta_k$$

- **Passo 7:** Cada unidade de saída ($Z_j, j = 1..p$) soma suas entradas δ_k (as informações de erro vindas da camada acima (posterior)).

$$\delta_{\text{in}_j} = \sum_{k=1}^m \delta_k w_{jk} \text{ e } \delta_j = \delta_{\text{in}_j} f'(z_{\text{in}_j})$$

$$\Delta v_{ij} = \alpha \delta_j x_i \text{ e } \Delta v_{0j} = \alpha \delta_j$$

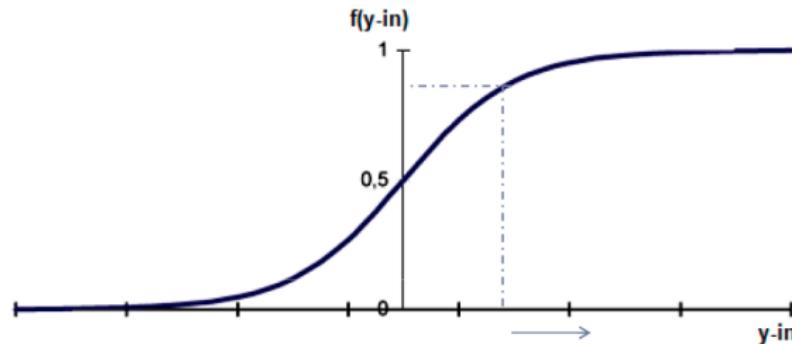


- **Passo 8:** Cada unidade de saída ($Y_k, k = 1..m$) altera seu bias e seus pesos ($j = 0..p$). Cada unidade escondida ($Z_j, j = 1..p$) altera seu bias e seus pesos ($i = 0..n$).

$$w_{jk}(\text{new}) = w_{jk}(\text{old}) + \Delta w_{jk}$$

$$v_{ij}(\text{new}) = v_{ij}(\text{old}) + \Delta v_{ij}$$

Multilayer Perceptron - treinamento - **Estude!!**

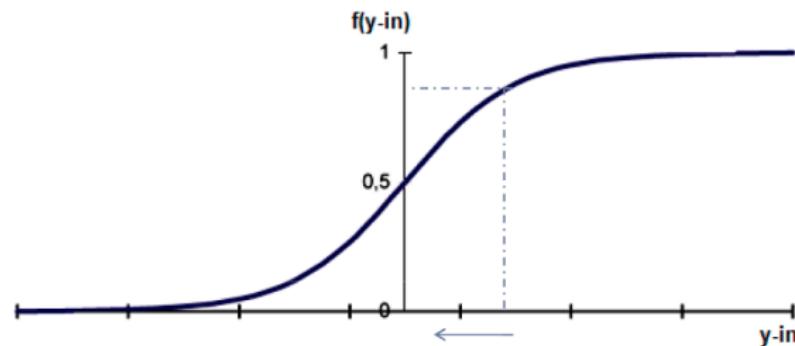


$x_{1..n}$	$y\text{-in}$	$f(y\text{-in})$	$Y(\text{desejado})$	$Y(\text{desejado}) - f(y\text{-in})$	$f'(y\text{-in})$
$x_{1..n}$	1,5	$1/(1+e^{-1,5}) = 0,81$	1	$1 - 0,81 = 0,19$ (positivo)	$0,81 * (1-0,81) = 0,14$ (positivo)

O valor esperado é maior que o obtido. É preciso que a entrada do neurônio seja mais forte para que o valor da ativação do neurônio seja mais alto.

A derivada indica a direção de crescimento da função. Combinando o valor da derivada ao erro, tem-se um passo na direção de correção do erro.

Multilayer Perceptron - treinamento - **Estude!!**

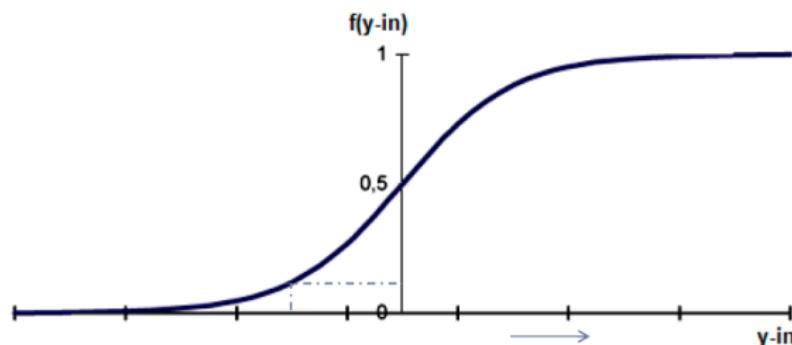


$x_{1..n}$	$y\text{-in}$	$f(y\text{-in})$	$Y\text{(desejado)}$	$Y\text{(desejado)} - f(y\text{-in})$	$f'(y\text{-in})$
$x_{1..n}$	1,5	$1/(1+e^{-1,5}) = 0,81$	0	$0 - 0,81 = -0,81$ (negativo)	$0,81*(1-0,81) = 0,14$ (positivo)

O valor esperado é menor que o obtido. É preciso que a entrada do neurônio seja mais fraca para que o valor da ativação do neurônio seja mais baixo.

A derivada indica a direção de crescimento da função. Combinando o valor da derivada ao erro, tem-se um passo na direção de correção do erro.

Multilayer Perceptron - treinamento - **Estude!!**

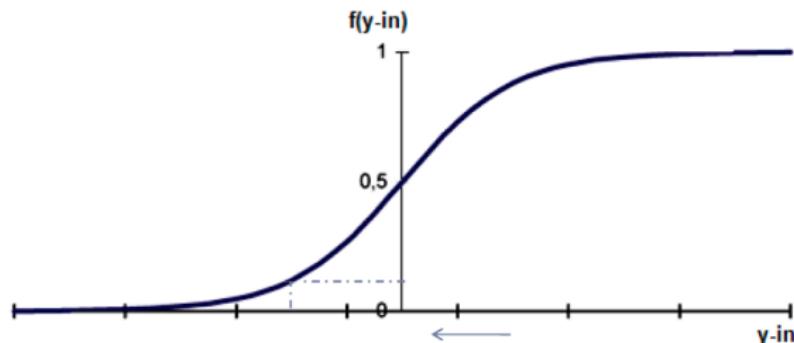


$x_{1..n}$	$y\text{-in}$	$f(y\text{-in})$	$Y(\text{desejado})$	$Y(\text{desejado}) - f(y\text{-in})$	$f'(y\text{-in})$
$x_{1..n}$	-1,5	$1/(1+e^{(-1,5)}) = 0,18$	1	$1 - 0,18 = 0,82$ (positivo)	$0,18 * (1 - 0,18) = 0,14$ (positivo)

O valor esperado é maior que o obtido. É preciso que a entrada do neurônio seja mais forte para que o valor da ativação do neurônio seja mais alto.

A derivada indica a direção de crescimento da função. Combinando o valor da derivada ao erro, tem-se um passo na direção de correção do erro.

Multilayer Perceptron - treinamento - Estude!!

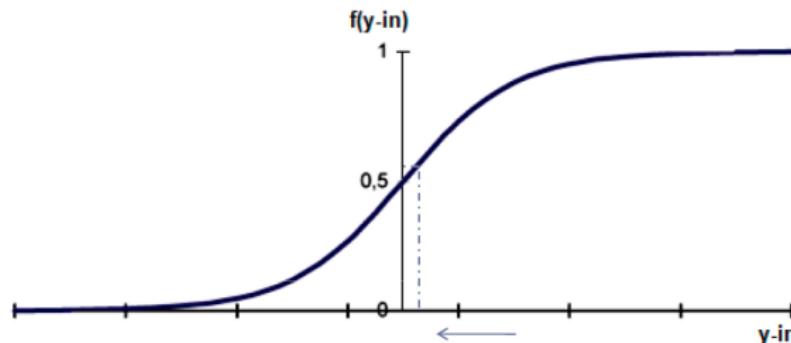


$x_{1..n}$	$y\text{-in}$	$f(y\text{-in})$	$Y(\text{desejado})$	$Y(\text{desejado}) - f(y\text{-in})$	$f'(y\text{-in})$
$x_{1..n}$	-1,5	$1/(1+e^{-(-1,5)}) = 0,18$	0	$0 - 0,18 = -0,18$ (negativo)	$0,18*(1-0,18) = 0,14$ (positivo)

O valor esperado é menor que o obtido. É preciso que a entrada do neurônio seja mais fraca para que o valor da ativação do neurônio seja mais baixo.

A derivada indica a direção de crescimento da função. Combinando o valor da derivada ao erro, tem-se um passo na direção de correção do erro.

Multilayer Perceptron - treinamento - **Estude!!**

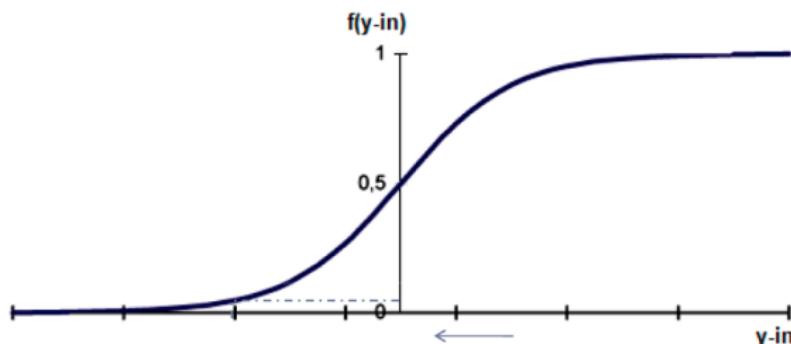


$x_{1..n}$	$y\text{-in}$	$f(y\text{-in})$	$Y(\text{desejado})$	$Y(\text{desejado}) - f(y\text{-in})$	$f'(y\text{-in})$
$x_{1..n}$	0,2	$1/(1+e^{-0,2}) = 0,54$	0	$0 - 0,54 = -0,54$ (negativo)	$0,54 \cdot (1-0,54) = 0,24$ (positivo)

O valor esperado é menor que o obtido. É preciso que a entrada do neurônio seja mais fraca para que o valor da ativação do neurônio seja mais baixo.

A derivada indica a direção de crescimento da função. Combinando o valor da derivada ao erro, tem-se um passo na direção de correção do erro.

Multilayer Perceptron - treinamento - **Estude!!**



$x_{1..n}$	$y\text{-in}$	$f(y\text{-in})$	$Y\text{(desejado)}$	$Y\text{(desejado)} - f(y\text{-in})$	$f'(y\text{-in})$
$x_{1..n}$	-2	$1/(1+e^{(-2)}) = 0,11$	0	$0 - 0,11 = -0,11$ (negativo)	$0,11*(1-0,11) = 0,09$ (positivo)

O valor esperado é menor que o obtido. É preciso que a entrada do neurônio seja mais fraca para que o valor da ativação do neurônio seja mais baixo.

A derivada indica a direção de crescimento da função. Combinando o valor da derivada ao erro, tem-se um passo na direção de correção do erro.



Multilayer Perceptron - aplicação

- **Passo 0:** Considere os pesos e bias obtidos no algoritmo de treinamento
- **Passo 1:** Para cada dado de entrada
 - **Passo 2:** Para $i = 1..n$: determine a ativação da iunidade de entrada x_i ;
 - **Passo 3:** Para $j = 1..p$:

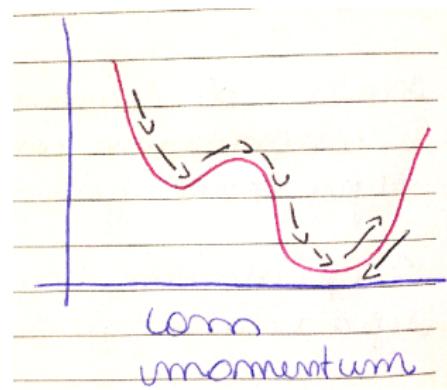
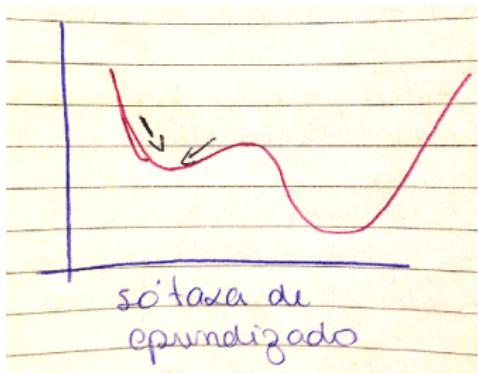
$$z_{\text{in}_j} = v_{0j} + \sum_{i=1}^n x_i v_{ij}$$

$$z_j = f(z_{\text{in}_j})$$

- **Passo 4:** Para $k = 1..m$:

$$y_{\text{in}_k} = w_{0k} + \sum_{j=1}^p z_j w_{jk}$$

$$y_k = f(y_{\text{in}_k})$$



A parada antecipada - Haykin

