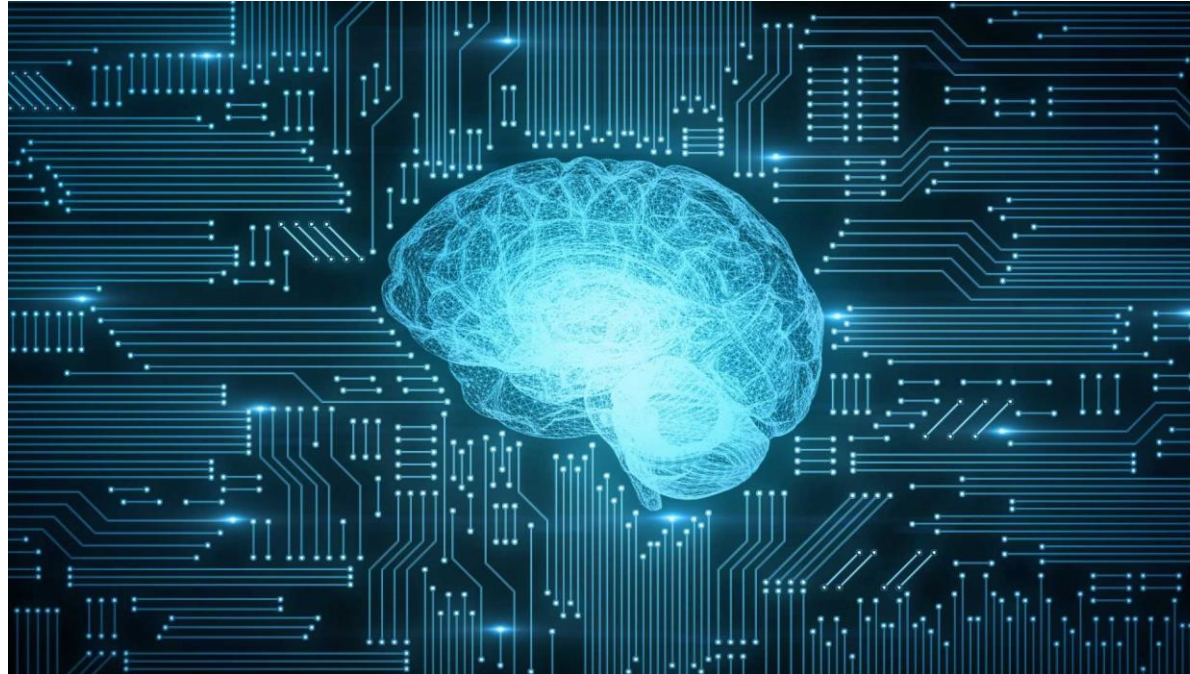


INTELIGÊNCIA COMPUTACIONAL



PROF. Dr. EDSON C. KITANI
2022

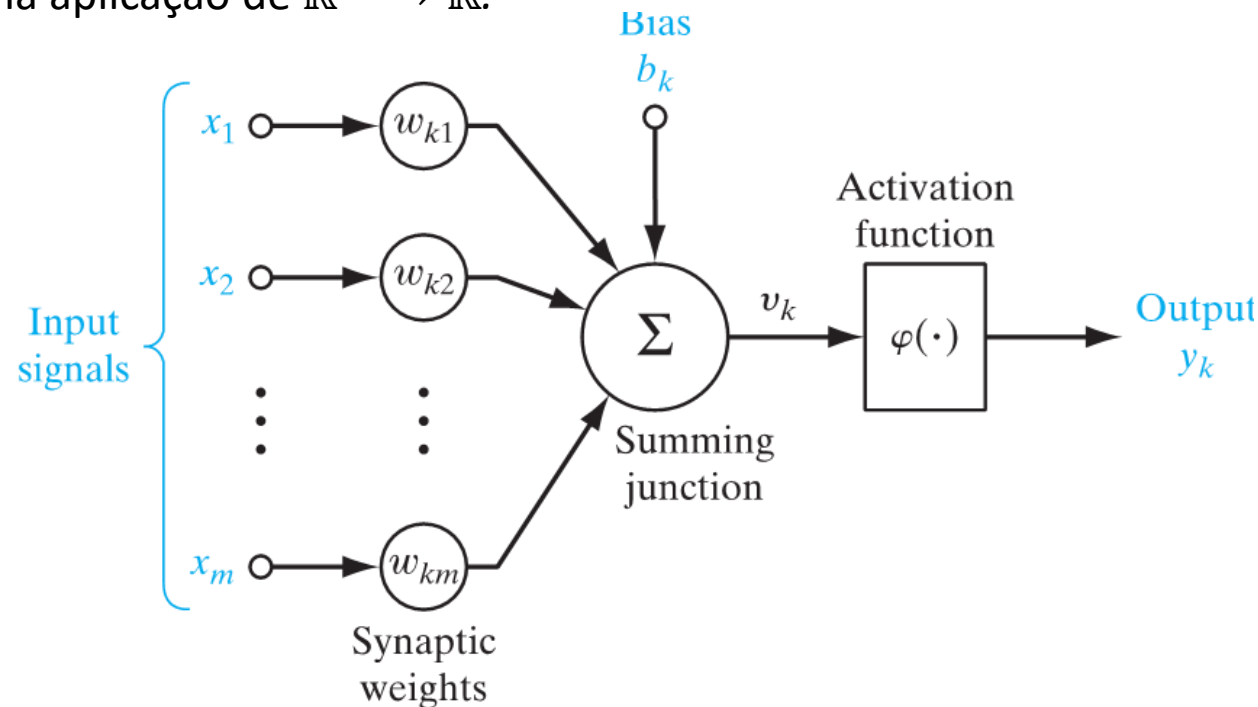
Livros Texto

- Tom Mitchell, **Machine Learning**, McGraw Hill, 1997
- Simon Haykin. **Neural Networks: A comprehensive foundation** 2nd Ed. 1999, Prentice Hall
- Richard Duda, Peter Hart, David Stork. **Pattern Recognition** 2nd Ed. 2001, John Wiley & Sons
- Trevor Hastie, Robert Tibshirani, Jerome Friedman, **The Elements of Statistical Learning**, Springer 2nd Ed.
- Ian Goodfellow, Yoshua Bengio, Aaron Courville, **Deep Learning**, MIT Press 2016
- Satish Kumar, **Neural Networks A classroom approach**, McGraw Hill, 2004
- John Hertz, Anders Krogh, Richard Palmer, **Introduction to the theory of neural computation**, Westview Press, 1991
- Teuvo Kohone, **Self-Organizing Maps**, Springer 3rd Ed. 2001
- Charu C. Aggarwal, **Neural Networks and Deep Learning**, Springer, 2018
- Christopher M. Bishop, **Neural Networks for Pattern Recognition**, Oxford, 2008
- Kevin P. Murphy, **Machine Learning: A Probabilistic Perspective**, MIT PRESS 2012
- Ivan Nunes Silva, et al. , **Redes Neurais Artificiais para Engenharia e Ciências Aplicadas**, Artliber, 2a Ed. 2016

- **Perceptron** (Haykin, Cap. 3, Duda Cap. 5, Mitchel Cap. 5)
- **MLP** (Nunes et al., Cap. 5)

Neurônio Artificial

Na aula anterior vimos que o neurônio artificial toma um vetor de entrada $X = (x_1, x_2, \dots, x_n)$ e faz o produto interno com uma matriz de pesos $W = (w_1, w_2, \dots, w_n)$ somado com um *bias* $b \in \mathbb{R}$. O resultado do produto interno mais o bias gera um $v_k \in \mathbb{R}$. Esse resultado intermediário é então apresentado a uma função de ativação $\varphi(\cdot)$, que produzirá a saída y_k . Basicamente, o neurônio é uma aplicação de $\mathbb{R}^n \rightarrow \mathbb{R}$.

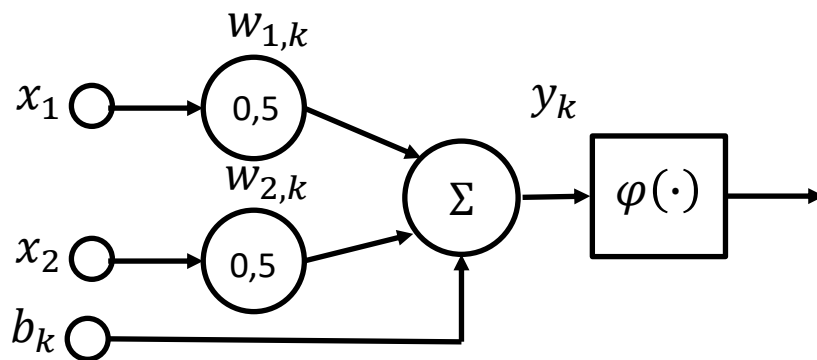


Vamos analisar em detalhes algumas funções de ativações, pois será ela que dará a natureza não linear no comportamento do Perceptron.

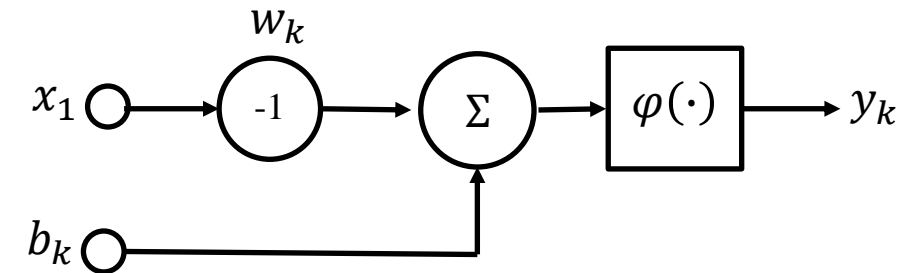
Função de Ativação Binária

Um neurônio com função de ativação binário pode representar funções binárias simples tais como operações AND, OR e NOT, para $X \in \{0, 1\}$ e considerando que $b_k = 0$. Este conceito nasceu com o primeiro neurônio artificial descrito por McCulloch e Pitts.

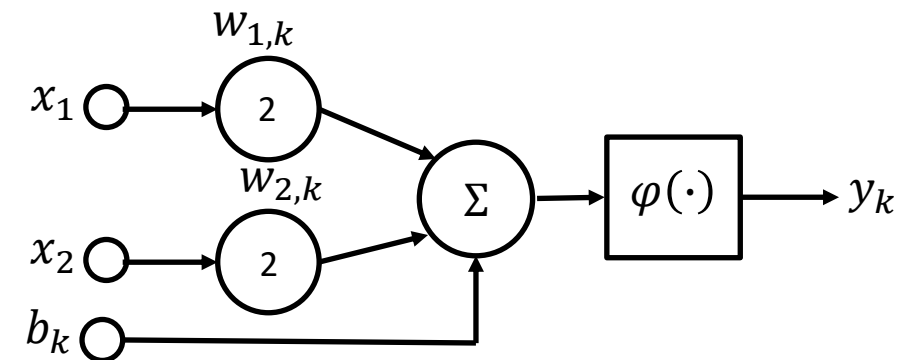
$$y_k = \varphi_k(v_k) = \begin{cases} 1 & \text{se } \sum_{i=1}^n w_{k,i}x_i + w_0x_0 \geq 0 \\ 0 & \text{se } \sum_{i=1}^n w_{k,i}x_i + w_0x_0 < 0 \end{cases}$$



Função AND



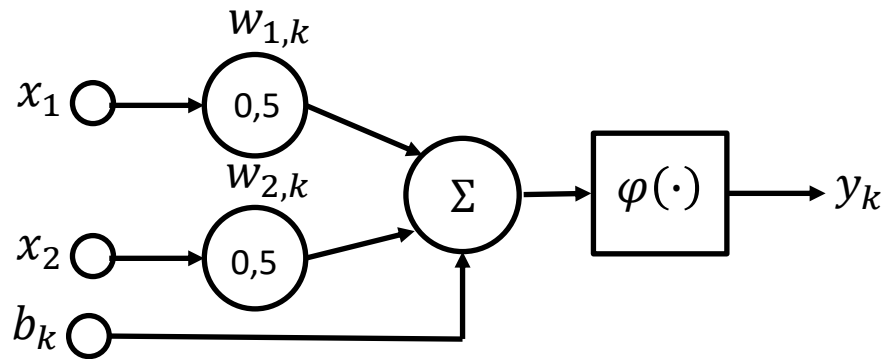
Função NOT



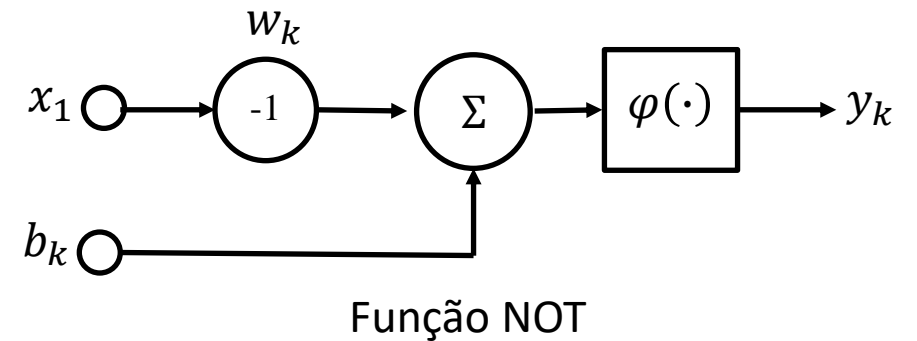
Função OR

Um neurônio com função de ativação binário pode representar funções binárias simples tais como operações AND, OR de 2 entradas e NOT, para $X \in \{0, 1\}$, considerando que $b_k = -1$ para as funções AND e OR e $b_k = 0$ para a função NOT. Para um número maior de entrada lembre-se de recalculer os pesos.

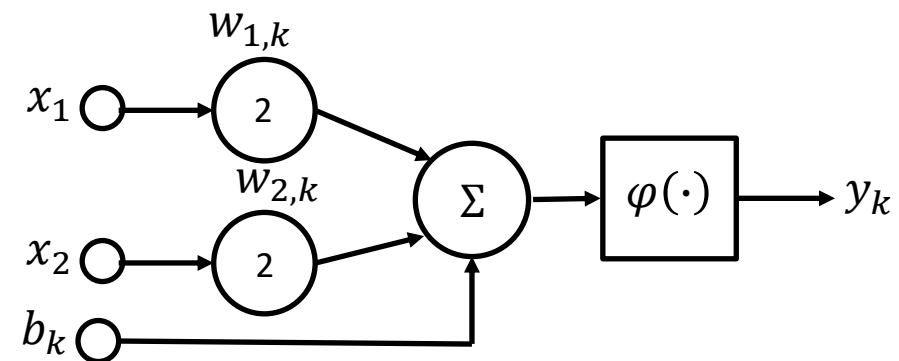
$$y_k = \varphi_k(v_k) = \begin{cases} 1 & \text{se } \sum_{i=1}^n w_{k,i} x_i + w_0 \geq 0 \\ 0 & \text{se } \sum_{i=1}^n w_{k,i} x_i + w_0 < 0 \end{cases}$$



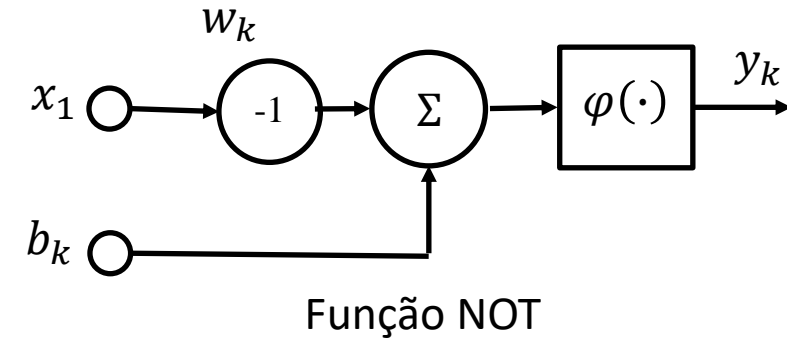
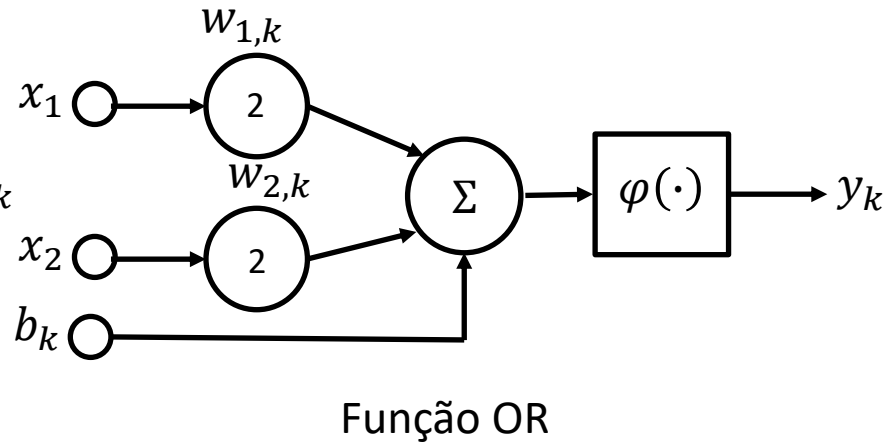
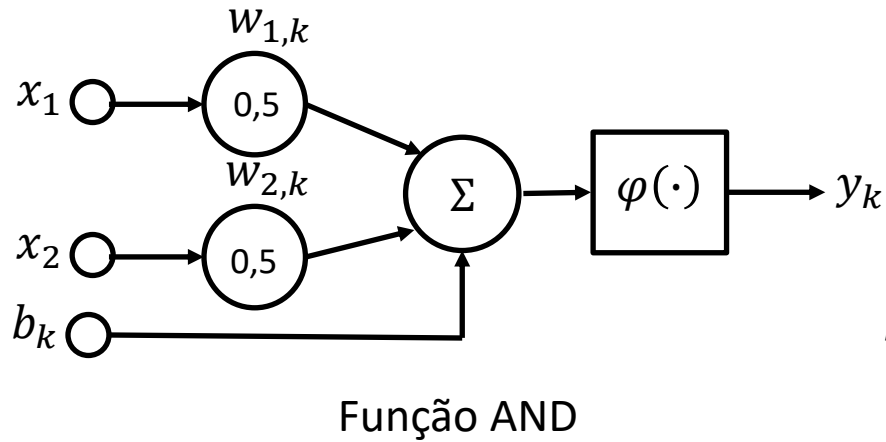
Função AND



Função NOT



Função OR



X1	X2	bk	Yk
0	0	-1	0
0	1	-1	0
1	0	-1	0
1	1	-1	1

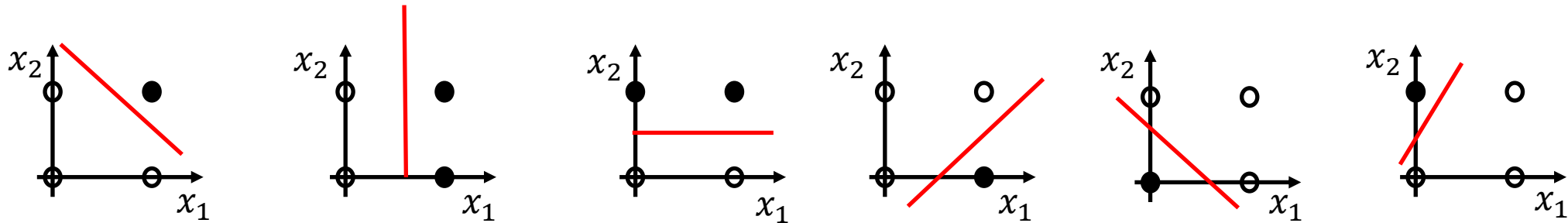
X1	X2	bk	Yk
0	0	-1	0
0	1	-1	1
1	0	-1	1
1	1	-1	1

X1	bk	Yk
0	0	1
1	0	0

“A inteligência é equivalente ao cálculo de predicados que por sua vez pode ser implementado por funções booleanas. Por outro lado, o sistema nervoso é composto de redes de neurônios, que com as devidas simplificações, tem a capacidade de básica de implementar estas funções booleanas.” (McCulloch & Pitts, apud Zolt L. Kovacs – 2006.

Neurônio de McCulloch e Pitts – Modelo Booleano

Considerando um espaço de variáveis x_1 e x_2 booleanos, podemos ter no máximo 16 possíveis funções que serão linearmente separáveis, mas somente duas delas não.



Padrões Booleanos linearmente separáveis

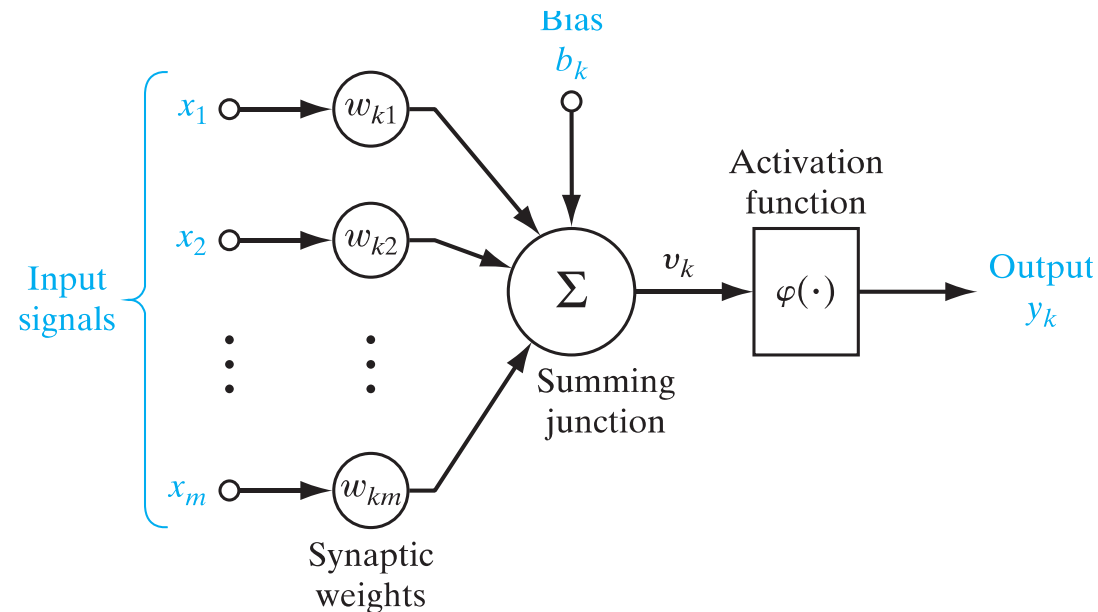


Padrões Booleanos não linearmente separáveis

Perceptron

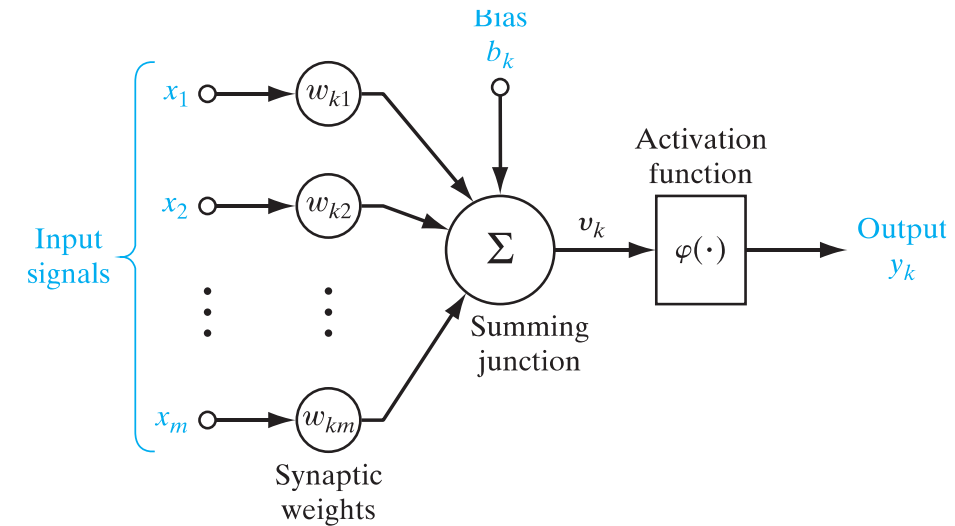
Conforme discutido na aula anterior, o Perceptron, idealizado pelo Rosemblatt (1958) é a forma mais simples de representação de um neurônio artificial.

Ele faz o produto interno entre o vetor de características (covariáveis) das amostras de entradas com o vetor de pesos, gerando o escalar $v_k = \langle x_k, w_k \rangle$. O resultado é então projetado no espaço de saída através de uma **função de ativação** $\varphi(\cdot)$.



Vamos detalhar melhor cada um dos parâmetros de um neurônio artificial.

- Sinal de entrada $X = \{x_1, x_2, \dots, x_n\}$ é o padrão de entrada que será usado para treinar ou estimar uma resposta
- Pesos sinápticos $W = \{w_1, w_2, \dots, w_n\}$ são valores usados para ponderar a importância de cada elemento do vetor de entrada.
- Combinador linear $\sum xw$ que soma o produto das entradas pelos respectivos ponderadores para gerar v_k .
- Limiar de ativação b_k (bias) que define o ponto na qual o resultado do combinador linear irá gerar um resultado para disparar uma respostas na saída.
- Função de ativação $\varphi(\cdot)$ limita a saída do neurônio conforme o sinal de entrada.

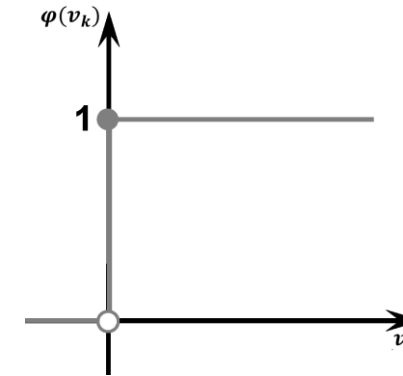


Extraído do Haykin, 3 ed.

A função de ativação limita a faixa de operação da saída do neurônio, e pode ser classificadas como:

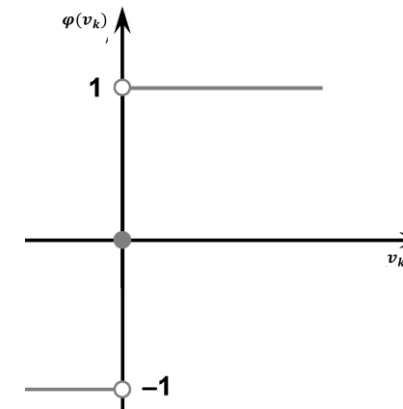
Função de Ativação Degrau ou Heavyside / Hard Limiter

$$\varphi(v_k) = \begin{cases} 1, & \text{se } v_k \geq 0 \\ 0, & \text{se } v_k < 0 \end{cases}$$



Função de Ativação Degrau bipolar

$$\varphi(v_k) = \begin{cases} 1, & \text{se } v_k > 0 \\ 0, & \text{se } v_k = 0 \\ -1, & \text{se } v_k < 0 \end{cases}$$

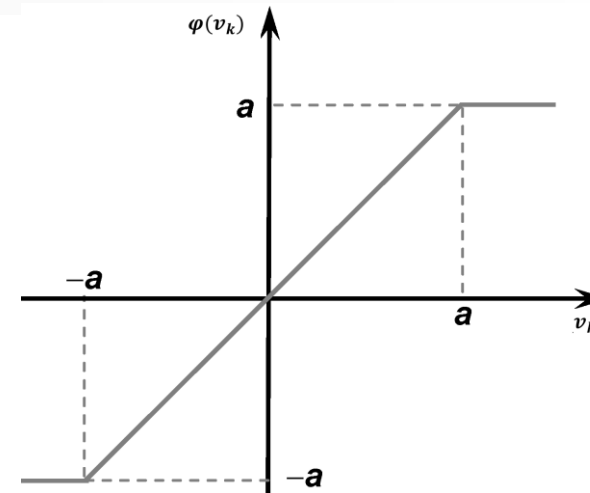


Extraído e Adaptado de Ivan Nunes et al., 2 ed.

Função de Ativação Rampa Simétrica

Para valores definidos no intervalo $[-a, a]$.

$$\varphi(v_k) = \begin{cases} 1, & \text{se } v_k > a \\ 0, & \text{se } -a \leq v_k \leq a \\ -1, & \text{se } v_k < -a \end{cases}$$

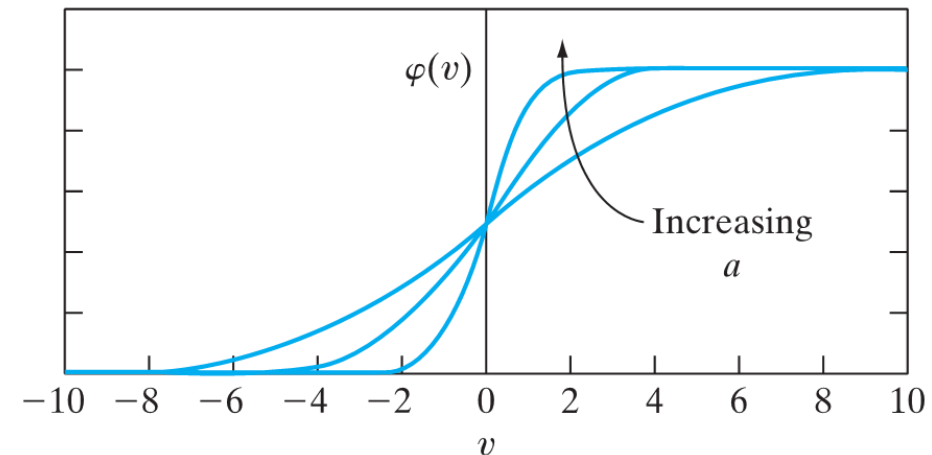


Extraído e Adaptado de Ivan Nunes et al., 2 ed.

Função de Ativação Sigmoidal ou Logística

Onde a_k é uma constante para o nível de inclinação da reta.

$$y_k = \varphi_k(v_k) = \frac{1}{1 + e^{-(a_k v_k)}}$$



Função de Ativação Tangente Hiperbólica

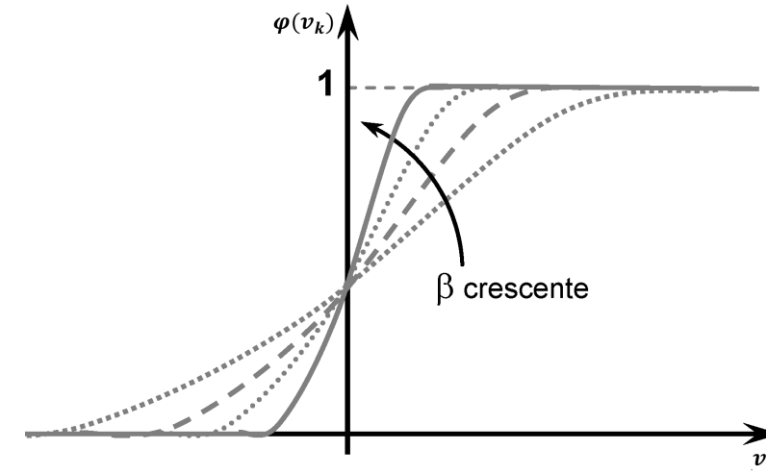
$$y_k = \varphi_k(v_k) = \tanh(a_k x_k) \text{ tal que } y_k \in (-1, 1)$$

Que pode ser derivado como:

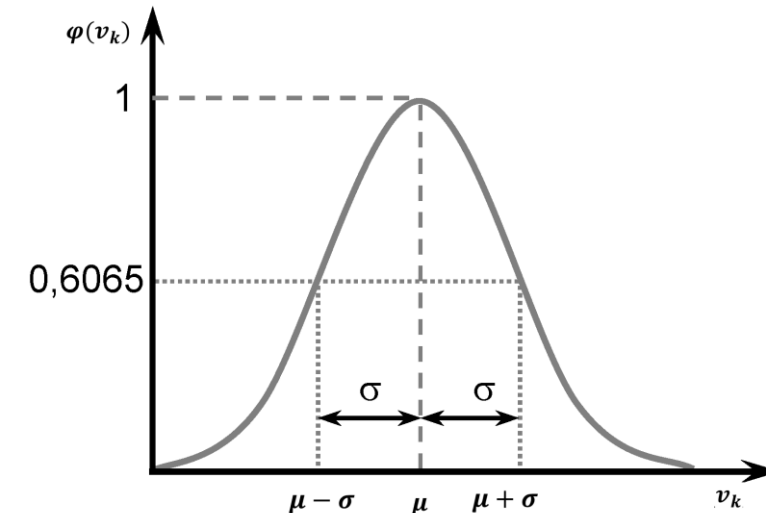
$$y_k = \varphi_k(v_k) = \frac{1 - e^{-(a_k v_k)}}{1 + e^{-(a_k v_k)}}$$

Função de Ativação Gaussiana

$$y_k = \varphi_k(v_k) = \exp\left(-\frac{(x_k - \mu_k)^2}{2(\sigma_k)^2}\right) \text{ tal que } y_k \in (0, 1)$$



Extraído e Adaptado de Ivan Nunes et al., 2 ed.



Função de Ativação Estocástica

Esta função será visto em detalhes na rede de Hopfield.

$$y_k = \varphi_k(v_k) = \begin{cases} +1 & \text{com probabilidade } P(x_k) \\ -1 & \text{com probabilidade } 1 - P(x_k) \end{cases} \text{ tal que } y_k \in (-1, 1) \text{ ou } y_k \in (0, 1)$$

$$\text{Sujeito a } P(x_k) = \begin{cases} 0 & \text{se } x_k \rightarrow -\infty \\ 1 & \text{se } x_k \rightarrow +\infty \end{cases}$$

Perceptron

Como a saída do neurônio é dependente da soma ponderada das entradas, podemos dizer que $y(X)$ é uma função discriminante. Se fizermos com que $y(\varphi(\cdot)) = 0$ encontraremos a reta de divide o espaço de entrada em duas regiões.

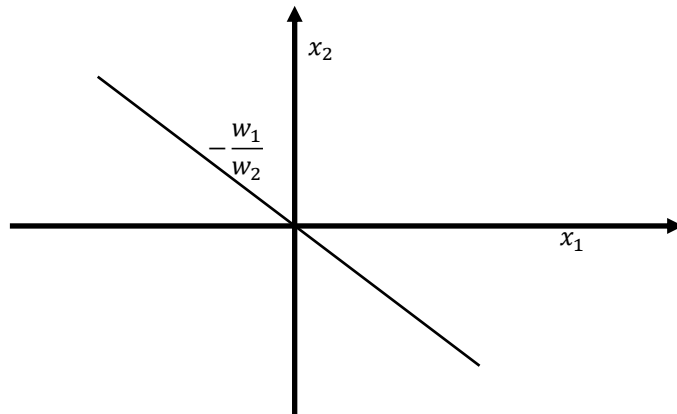
$$w_0x_0 + w_1x_1 + w_2x_2 = 0$$

e

$$x_2 = -\frac{w_1}{w_2}x_1 - \frac{w_0}{w_2}$$

$$y_k = \varphi_k(v_k) = \begin{cases} 1 & \text{se } \sum_{i=1}^n w_{k,i}x_i + w_0 \geq 0 \\ 0 & \text{se } \sum_{i=1}^n w_{k,i}x_i + w_0 < 0 \end{cases}$$

Assim, para $w_1, w_2 = 1$ e $w_0 = 0$, teremos o espaço de entrada dividido como:



Perceptron – Feed Forward

Até o momento estudamos o neurônio para uma camada com a capacidade para separar o espaço de entrada em duas regiões ou classes. Concluimos então que o neurônio é uma aplicação de $\mathbb{R}^{n+1} \rightarrow \mathbb{R}$ através do produto interno entre X e W .

Classificação de Padrões Multiclasse

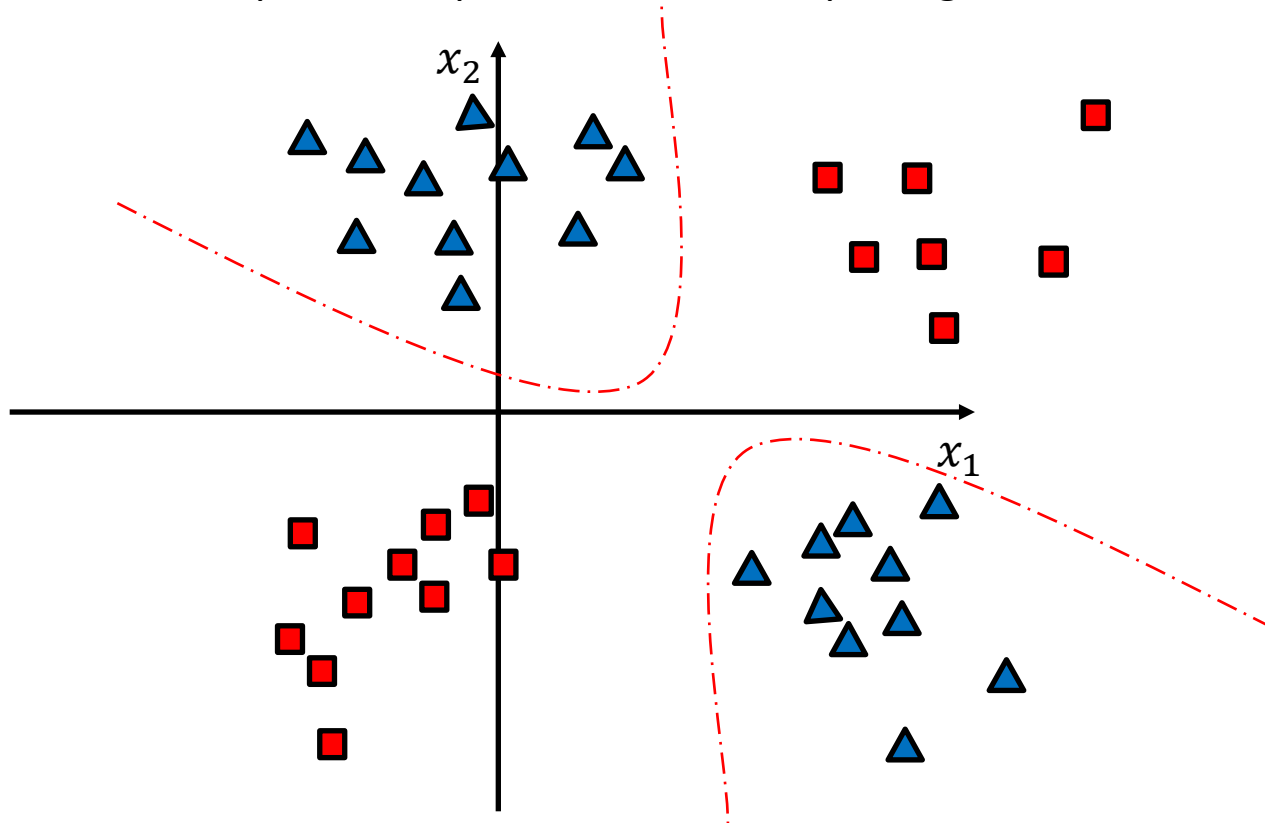
Vamos considerar um conjunto finito de \mathcal{X} elementos com Q padrões distintos $\{X_1, X_2, \dots, X_Q\}$ e vamos assumir que os padrões de \mathcal{X} são classificados de tal maneira que pertençam a apenas um conjunto c_j .

Se a classificação divide o espaço \mathcal{X} em subconjuntos $\{\mathcal{X}_1, \mathcal{X}_2, \dots, \mathcal{X}_C\}$ são linearmente separáveis se e somente se existirem funções discriminantes y_1, y_2, \dots, y_C , tal que $y_i(X) > y_j(X) \quad \forall X \in \mathcal{X}, i, j = 1, \dots, C \text{ e } i \neq j$.

A definição acima pode ser compreendida para o caso de $C = 2$, onde temos duas funções discriminantes y_1 e y_2 que criam uma dicotomia no espaço \mathcal{X} , criando os subespaços \mathcal{X}_1 e \mathcal{X}_2 somente se temos uma função discriminante geral $y(X) = y_1(X) - y_2(X)$ tal que $y(X) > 0 \quad \forall X \in \mathcal{X}_1$ e $y(X) < 0 \quad \forall X \in \mathcal{X}_2$, sendo assim linearmente separáveis, caso exista um hiperplano que mantenha os elementos de \mathcal{X}_1 de um lado e os elementos de \mathcal{X}_2 do outro lado.

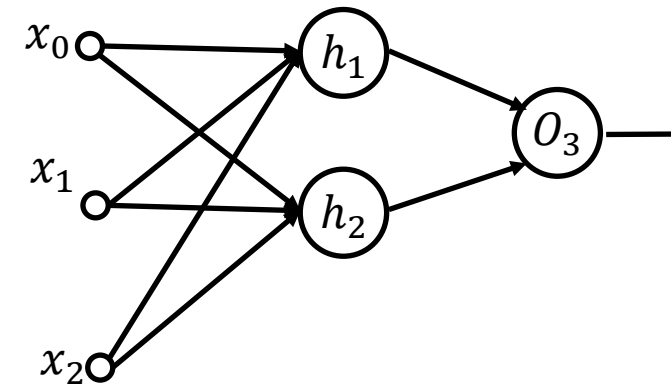
Problema do Exclusive Or

A maior crítica aos neurônios artificiais foi a incapacidade de resolver problemas não lineares, como o caso abaixo. Contudo, o problema pode ser resolvido por álgebra booleana.



$$y = x_1 \oplus x_2$$

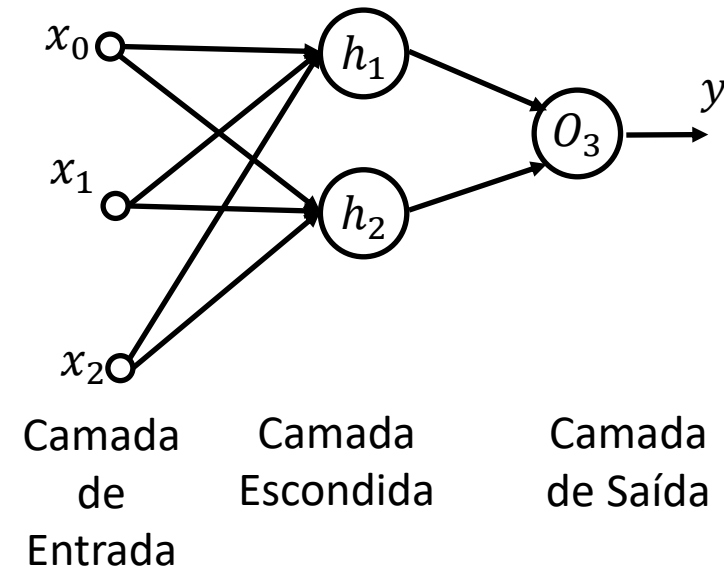
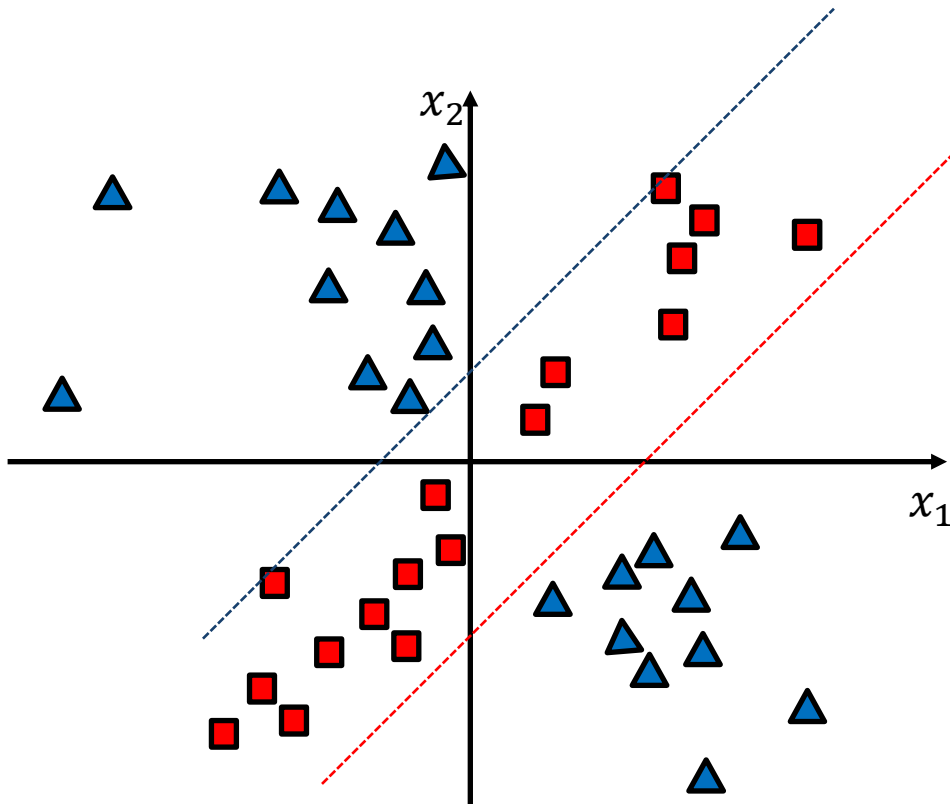
$$y = (\bar{x}_1 x_2) + (x_1 \bar{x}_2)$$



Observe que o neurônio o_3 une o dois meio espaços que dividem as classes.

Problema do Exclusive Or

Observe que a ideia do Feed Forward, depois que a rede está treinada, é apenas seguir com os sinais da entrada para saída. Como cada neurônio cria um hiperplano no espaço entrada, precisaremos de 2 neurônios para realizar a separação do espaço dos dados abaixo.



TREINAMENTO DO PERCEPTRON

Princípio de Aprendizado de Hebb

O Princípio de Aprendeizado de Hebb (PAH) é baseado no conceito de que o processo de aprendizado é um processo local, na qual a intensidade das conexões sinápticas é modificada apenas pelos erros observáveis e locais. Dessa maneira, o valor do i-ésimo peso w_i ser adaptado, levando-se em conta o valor do peso w_{i-1} que produziu o erro $(y_i - \hat{y}_i)$ na qual:

$$\Delta w_i = \eta(y_i - \hat{y}_i) \cdot x_i$$

$$w_i^{Novo} = w_i^{Velho} + \Delta w_i$$

Logo, por essa regra, a mudança no i-ésimo parâmetro depende apenas de produto da i-ésima entrada pelo erro produzido na saída por essa entrada.

O fator η representa a “taxa de aprendizado”, mas nada mais é do que um fator de aproximação que reduz a velocidade de convergência do neurônio.

Perceptron – Gradiente Descendente

Conceito do Método de Correção dos Pesos por LMS (Least Minimum Square)

O conceito do LMS foi utilizado nos primeiros treinamentos do Perceptron e baseia-se na da ideia da utilização dos valores instantâneos para correção do erro ou função do custo.

$$E(w) = \frac{1}{2} erro^2(t)$$

Onde $erro^2(t)$ é o erro medido no tempo (t) . Quando diferenciamos $E(w)$ e relação ao vetor de pesos w , temos:

$$\frac{\partial E(w)}{\partial w} = erro(n) \frac{\partial erro(t)}{\partial w}$$

Na qual:

$$erro(t) = y(t) - x^T(t)w(t)$$

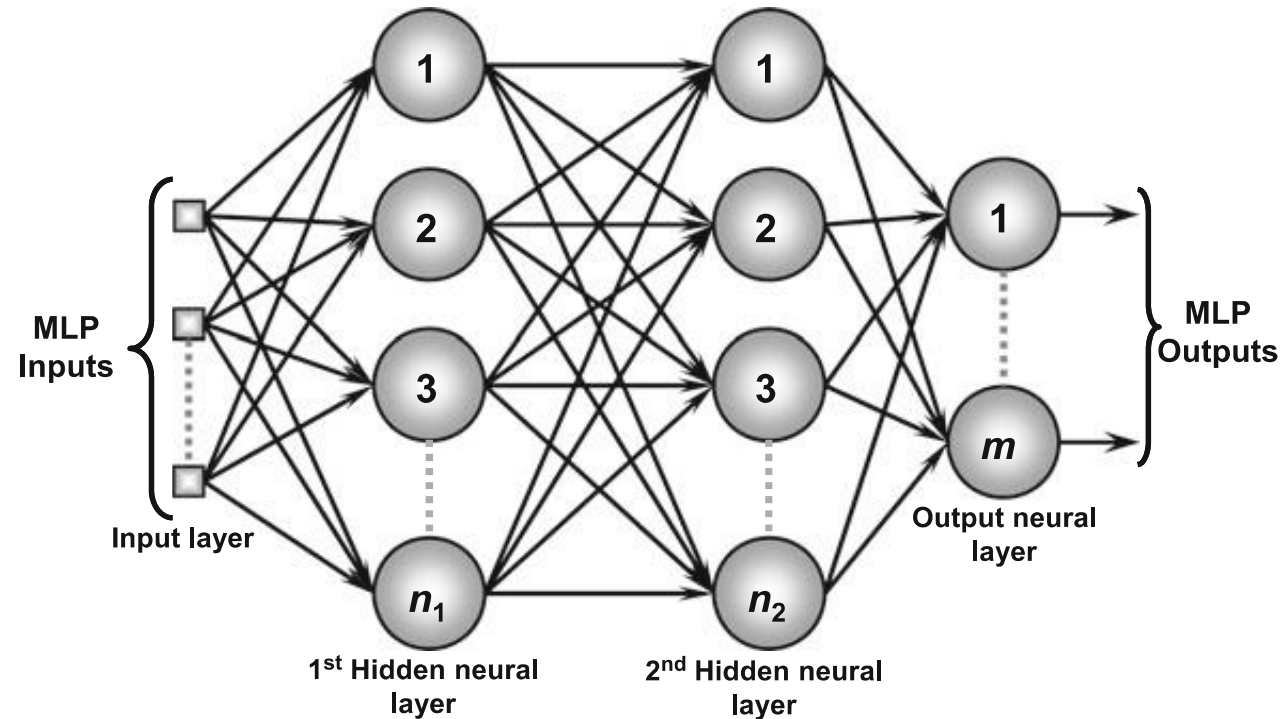
Finalmente, após a derivada parcial em relação a w , temos:

$$W^{t+1} = W^t + \eta \left(\sum_{k=1}^N (Y_k - \hat{Y}_k) X_k \right)$$

MULTILAYER PERCEPTRON

Perceptron Multicamadas (MLP)

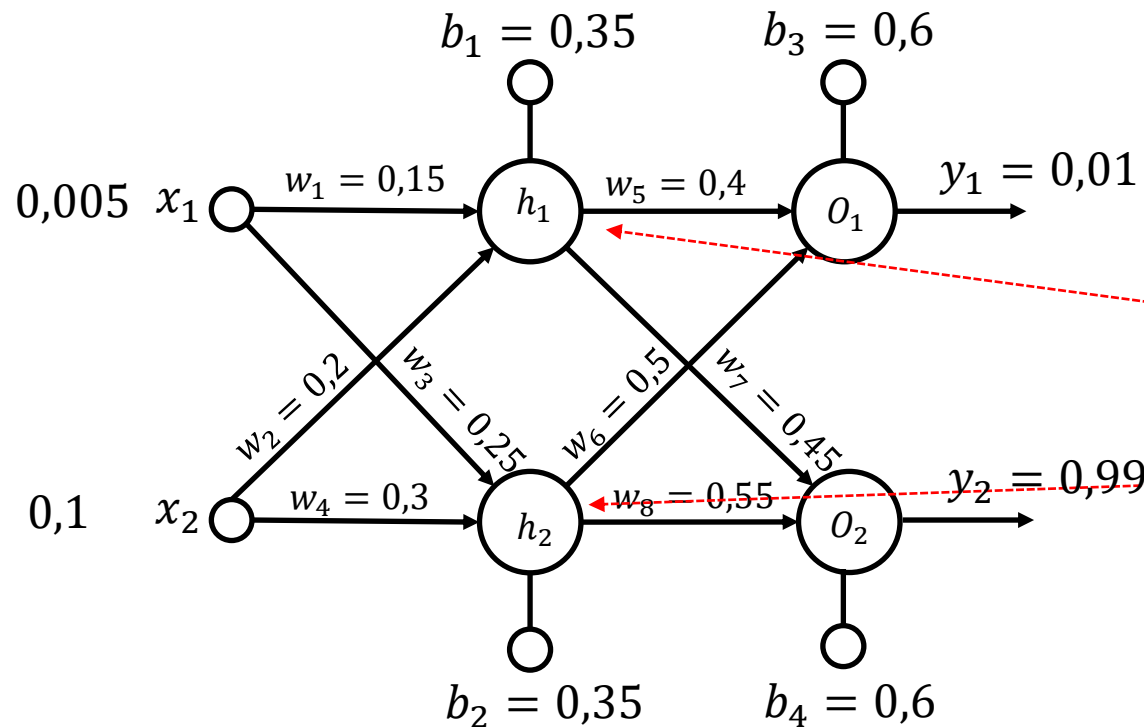
Como vimos, o Perceptron sozinho não consegue tratar problemas não lineares. Contudo, um Perceptron que possua múltiplas camadas pode dividir o espaço de entrada de maneiras mais complexas e não linear, resolvendo a limitação do Perceptron linear de uma camada.



Apesar do conceito das MLPs já fosse conhecida no tempo de Roseblatt, não havia um conceito fundamental de aprendizado para esse tipo de rede.

O algoritmo de Backpropagation foi fundamental para permitir a implementação das MLPs, e até hoje o conceito ainda é usado. E base principal desse algoritmo é o gradiente descendente.

Exemplo da Propagação dos Sinais num MLP de 2 Camadas – Execute manualmente a propagação



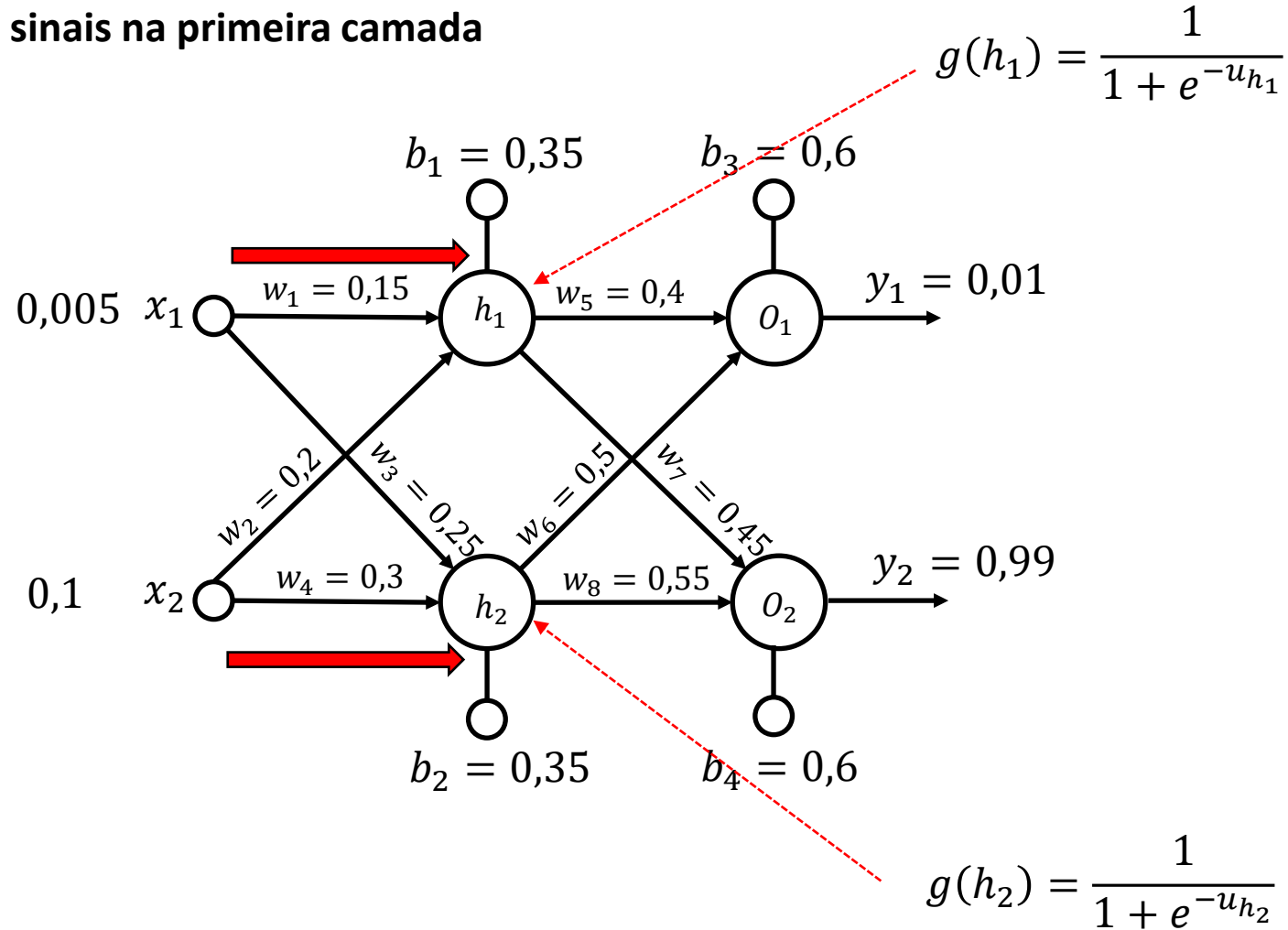
Considere a função de Ativação Sigmoidal da camada escondida como sendo:

$$g(h_1) = g(u_{h_1}) = \frac{1}{1 + e^{-u_{h_1}}}$$

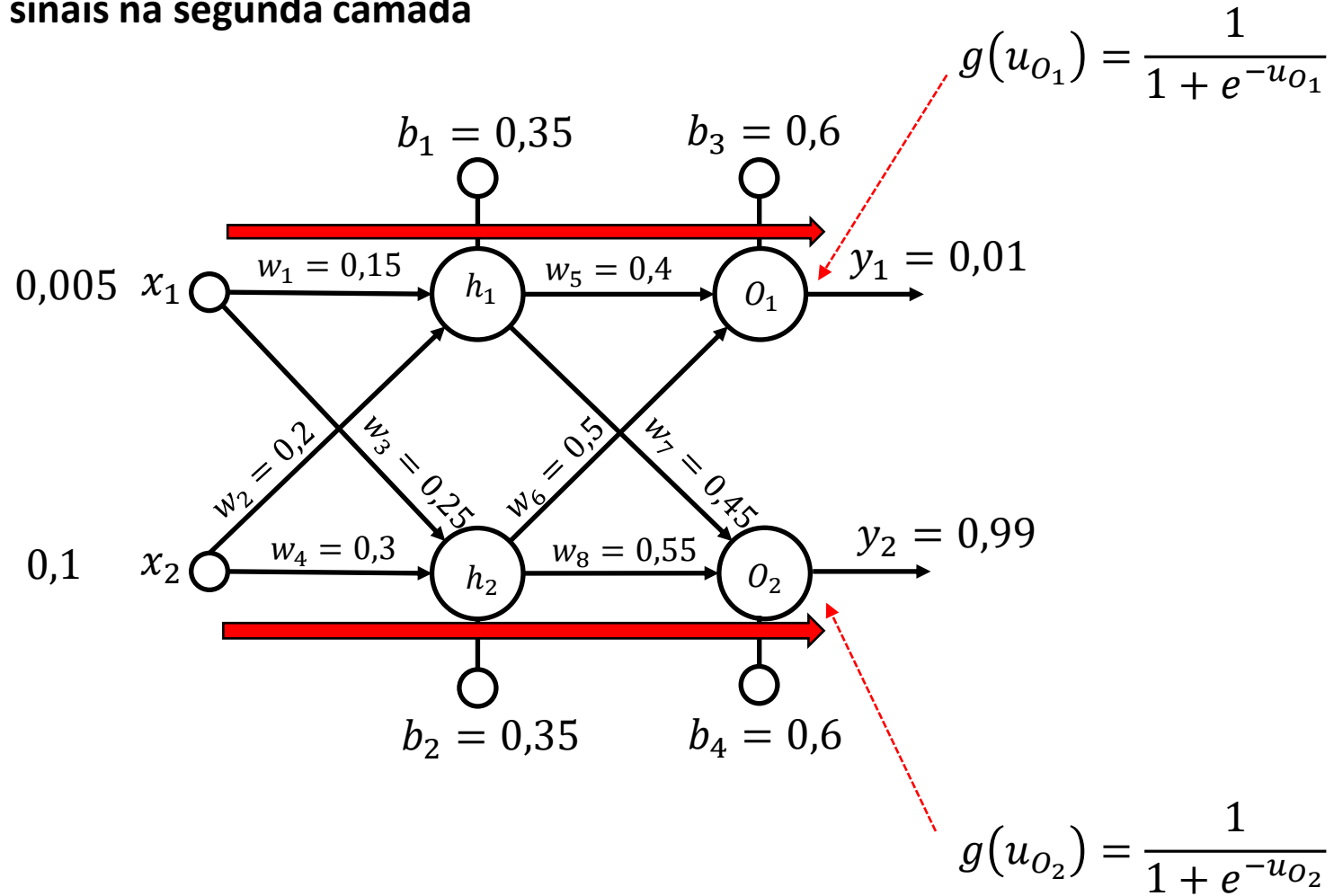
$$g(h_2) = g(u_{h_2}) = \frac{1}{1 + e^{-u_{h_2}}}$$

$\eta = 0,5$ (taxa de aprendizado)

Propagando os sinais na primeira camada

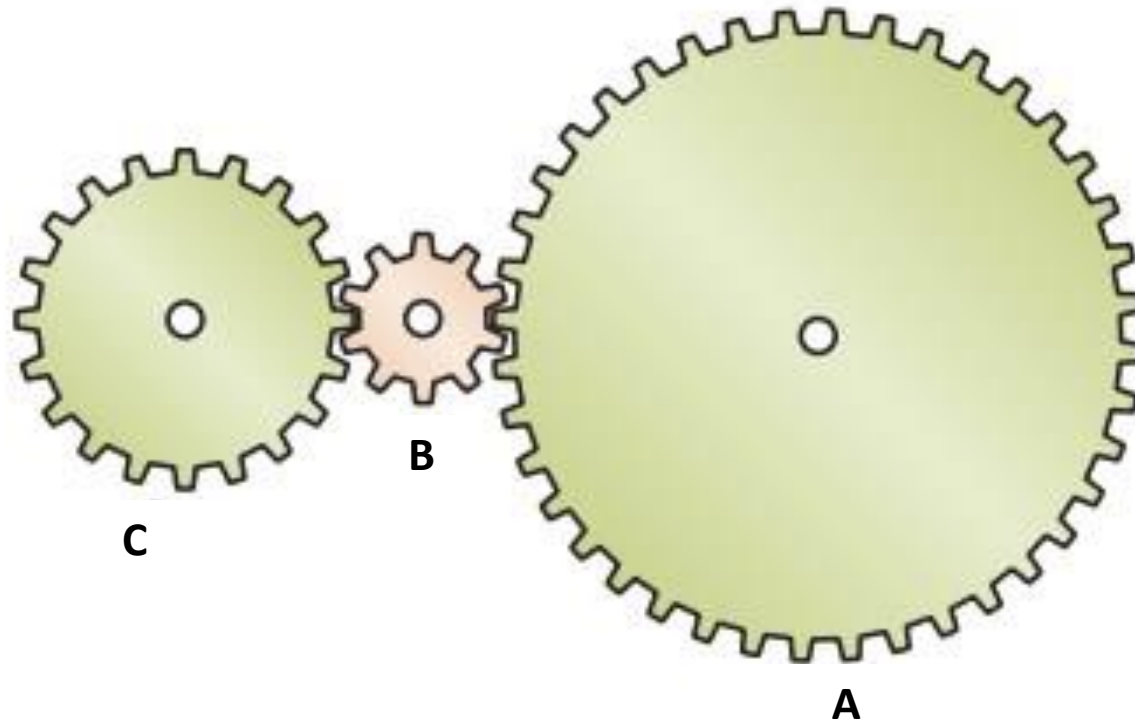


Propagando os sinais na segunda camada



Considere que a engrenagem A vira x voltas, a engrenagem B vira u voltas e a engrenagem C vira y voltas. Então:

$$\frac{dy}{dx} = \frac{dy}{du} \frac{du}{dx}$$



Determinar influência do erro na próxima correção dos pesos fica facilitada pela Regra da Cadeia. Se $y = f(u)$ e $u = g(x)$, então $y = f(g(x))$.

$$\frac{dy}{dx} = \frac{dy}{du} \frac{du}{dx}$$

$$\frac{dy}{dx} = \frac{dy}{dx} f(g(x)) = f'(g(x))g'(x)$$

Exemplo:

$$\begin{aligned} f(x) &= (3x - 5x^2)^3 \\ u &= 3x - 5x^2 \rightarrow 3 - 10x \\ y &= u^3 \rightarrow 3u^2 \\ \frac{dy}{dx} &= 3u^2(3 - 10x) \\ \frac{dy}{dx} &= 3(3x - 5x^2)^2(3 - 10x) \end{aligned}$$

Backpropagation Algorithm

O treinamento de uma rede MLP via Backpropagation envolve 3 estágios distintos:

- Fase 1 – Feed forward na qual os sinais de entrada são propagados através da rede seguindo os produtos internos e funções de ativação.
- Fase 2 – Determinação do erro nas saídas e posterior retropropagação do erro.
- Fase 3 – Atualização dos pesos e retorno à fase 1.

Na próxima página detalharemos as 3 fases acima.

Algoritmo de Treinamento Backpropagation para duas camadas

Step 1: Inicialize os pesos de maneira aleatória e com valores pequenos

Step 2: Faça até atingir o critério de parada

Step 2.a – Para cada amostra de treinamento execute o produto interno para cada neurônio da camada oculta.

$$u_{h_k} = \sum_{i=1}^n w_i x_i + b_k, \text{ onde } h_k \text{ é o } k\text{-ésimo neurônio da camada oculta}$$

Step 2.b – Para cada produto interno calculado no Step 2.a, calcule a função de ativação.

$$g(h_k) = \varphi(u_{h_k})$$

Step 2.c – Para cada neurônio de saída O_k compute:

$$g_{o_k} = \sum_{i=1} w_i g_{h_k} + b_k$$

Algoritmo de Treinamento Backpropagation para duas camadas

Step 1: Inicialize os pesos de maneira aleatória e com valores pequenos

Step 2: Faça até atingir o critério de parada

Step 2.a – Para cada amostra de treinamento execute o produto interno para cada neurônio da camada oculta.

$$u_{h_k} = \sum_{i=1}^n w_i x_i + b_k, \text{ onde } h_k \text{ é o } k\text{-ésimo neurônio da camada oculta}$$

Step 2.b – Para cada produto interno calculado no Step 2.a, calcule a função de ativação.

$$g(h_k) = \varphi(u_{h_k})$$

Step 2.c – Para cada neurônio de saída O_k compute:

$$g_{o_k} = \sum_{i=1} w_i g_{h_k} + b_k$$

Step 2.d – Para cada neurônio de saída O_k compute a função de ativação:

$$y_k = \varphi(g_{O_k})$$

Step 3 – Fase do Backpropagation. Para cada neurônio de saída compute o erro e a propagação:

$$\delta_k = (y_k - \varphi(g_{O_k})) \varphi'(g_{O_k})$$

Step 4 – Calcula o termo de correção do peso:

$$\Delta w_{j,k} = \eta \delta_k \varphi(h_k)$$

Step 5 – Calcula a correção do bias também:

$$\Delta b_k = \eta \delta_k$$

Step 6 – Propaga a variação δ_k para camadas anteriores

Step 7 – Para cada neurônio da camada oculta some suas entrada deltas vindas da camada seguinte:

$$\delta_{in_j} = \sum_{k=1}^m \delta_k w_{h_k}$$

Step 8 – Calcule a propagação do erro multiplicando o termo do step 7 pela derivada da função de ativação:

$$\delta_j = \delta_{in_j} \varphi'(g_{h_k})$$

Step 9 – Calcule a correção

$$\Delta v_{i,k} = \eta \delta_j x_i$$

Step 10 - Calcule a correção do bias:

$$\Delta b_{h_k} = \eta \delta_j$$

Step 11 – Para cada neurônio de saída atualize os pesos e bias:

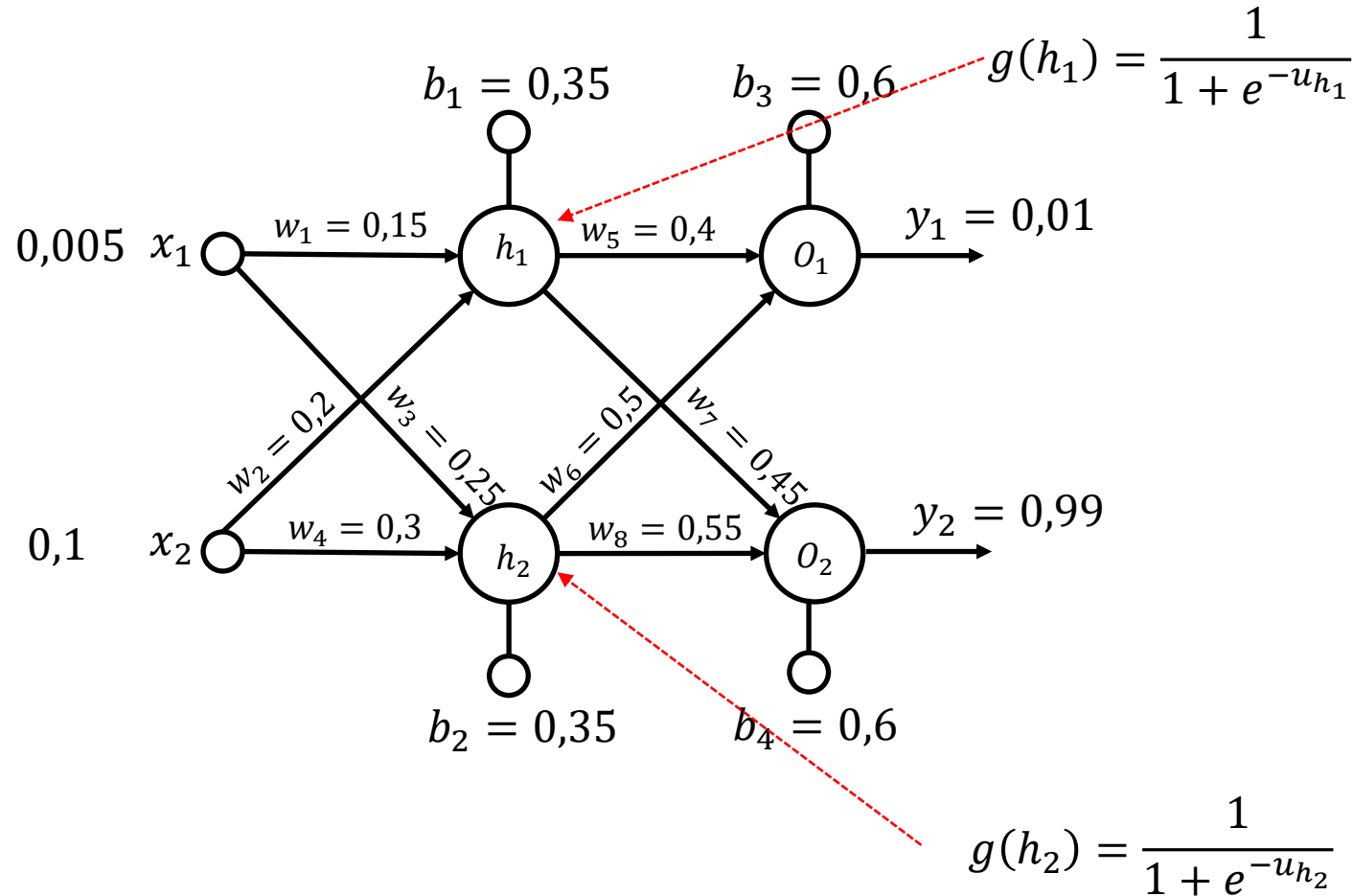
$$w_{j,k}^{t+1} = w_{j,k}^t + \Delta w_{j,k}$$

Step 12 – Para cada neurônio da camada oculta atualiza os pesos bias por:

$$w_{i,k}^{t+1} = w_{i,k}^t + \Delta v_{i,k}$$

Volte para o passo 2 até que o critério de parada definido seja atingido, seja por número de épocas a ser executado, pelo tamanho do erro, ou qualquer outra métrica desejada.

Exercício: Recalcule os pesos usando o algoritmo Backpropagation para 1 época.



MLP - Hiperparâmetros

Uma MLP têm vários hiper-parâmetros que afetam bastante o comportamento da rede. Antes de usar uma MLP é importante conhecer a influência deles nos resultados. Fundamentalmente, o erro produzido por uma MLP é função dos parâmetros livres.

Taxa de Aprendizagem η – É o tamanho do ajuste dos pesos a cada iteração:

- ✓ Se η é pequeno, produz variação muito lenta nos pesos e pode ficar preso em mínimos locais.
- ✓ Se η é grande, torna o aprendizado instável, oscilando entre dois extremos de um mínimo.

O problema é determinar qual a melhor taxa utilizar para evitar mínimos locais e oscilações. Um método é utilizar o termo de momento γ , modificando o a regra delta. A ideia é somar uma parcela dos ajustes dos termos de correção dos pesos* da iteração anterior nos termos de ajuste dos pesos atuais.

$$\Delta w_{j,k} = \eta \delta_k \varphi(h_k) + \gamma (\Delta w_{j,k}^{t-1})$$

Se os dois termos de ajustes tiverem o mesmo sinal, o $\Delta w_{j,k}$ aumentará, acelerando o aprendizado.

Se o dois termos de ajustes tiverem sinais opostos, $\Delta w_{j,k}$ diminuirá reduzindo a oscilação.

O problema é que ele é mais um parâmetro ($0 \leq \gamma < 1$)

O número de neurônios por *layers* (camada oculta) também é ajustável e o seu aumento implica no aumento da complexidade e graus de liberdade da rede, podendo gerar *overfitting*, bem como maior tempo de treinamento. Existem algumas heurísticas para determinar a quantidade de neurônios numa rede. As regras abaixo são para o número de neurônios N_h para uma camada oculta.

a) $N_h = 2n + 1$, onde n é a dimensionalidade de X (Kolmogorov).


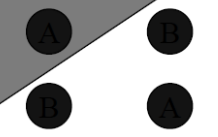

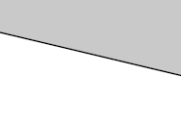
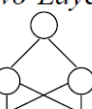
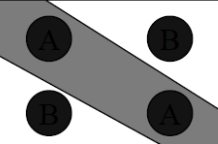
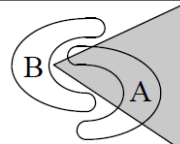
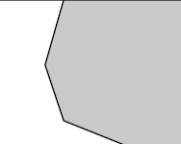
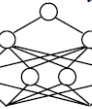
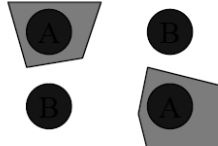
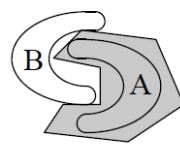
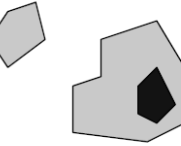
b) $N_h = n + D$, onde n é a dimensionalidade de X e D é o número de classes de X .

c) $N_h = \frac{n+D}{2}$, idem ao (b).

d) $N_h = D$, onde D é o número de classes de X .

Lembrando que o número de neurônios na camada de saída será $N_o = D$ para um MLP de classificação.

Número de camadas ocultas depende da complexidade do problema. Como cada neurônio cria um hiperplano de decisão, uma maior quantidade de neurônios aumenta fragmentação do espaço de entrada em regiões convexas. Quanto maior o número de *layers*, maior será a complexidade da rede e maior será a abstração do formato da segmentação do espaço de entrada.

Structure	Types of Decision Regions	Exclusive-OR Problem	Classes with Meshed regions	Most General Region Shapes
<i>Single-Layer</i> 	<i>Half Plane Bounded By Hyperplane</i>			
<i>Two-Layer</i> 	<i>Convex Open Or Closed Regions</i>			
<i>Three-Layer</i> 	<i>Arbitrary (Complexity Limited by No. of Nodes)</i>			

1 camada oculta gera hiperplanos.

2 camadas ocultas geram regiões convexas.

3 camadas ocultas geram combinações de convexidades.

Richard P. Lippmann – An introduction to neural networks by Andrew Hunter

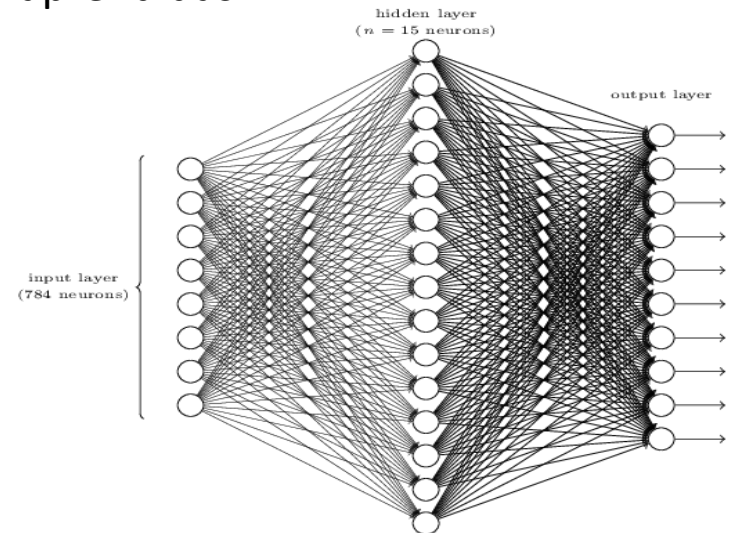
O número de neurônios por *layers* também é ajustável e o seu aumento implica no aumento da complexidade e graus de liberdade da rede, podendo gerar *overfitting*, bem como maior tempo de treinamento. Os neurônios da 1ª camada aprendem características locais de regiões específicas e as características globais são extraídas na 2ª camada.

A inicialização dos pesos talvez seja a tarefa mais difícil de determinar. Os principais problemas na seleção do pesos iniciais é evitar um longo tempo de convergência e, principalmente, para evitar mínimos locais e platôs.

- a) Inicializar todos os pesos com zero. Demora na convergência e pode ficar preso.
- b) Inicializar todos os pesos com o mesmo valor, produzirá uma mudança uniforme em todos os pesos.
- c) Inicialização estocástica. Nesse método, escolhem-se aleatoriamente os valores dos pesos. Também pode demorar e ficar presos e regiões de mínimo locais.
- d) Inicialização estocástica com distribuição uniforme entre os limites $(-\epsilon, +\epsilon)$. Para o caso cujas entradas variam entre $[-1, +1]$, pode-se adotar os limites entre $[-0,5, +0,5]$.

Existem vários outros métodos, mas que devem ser avaliadas caso a caso. Por exemplo, usar o PCA para encontrar as características mais significativas e distribuir uniformemente os pesos nas direções principais.

O número de entradas depende da quantidade de características a serem mapeadas e o número de saídas dependerá, no caso de classificadores, do número de classes a serem aprendidas.



Finalmente, a convergência da rede é algo que não pode ser determinado previamente. Assim, é necessário determinar critérios de parada que podem ser:

- a) Quando o vetor de erro dos pesos atingir um limite menor ou igual a um $E > 0$.
- b) Número de iterações limite foi atingido.

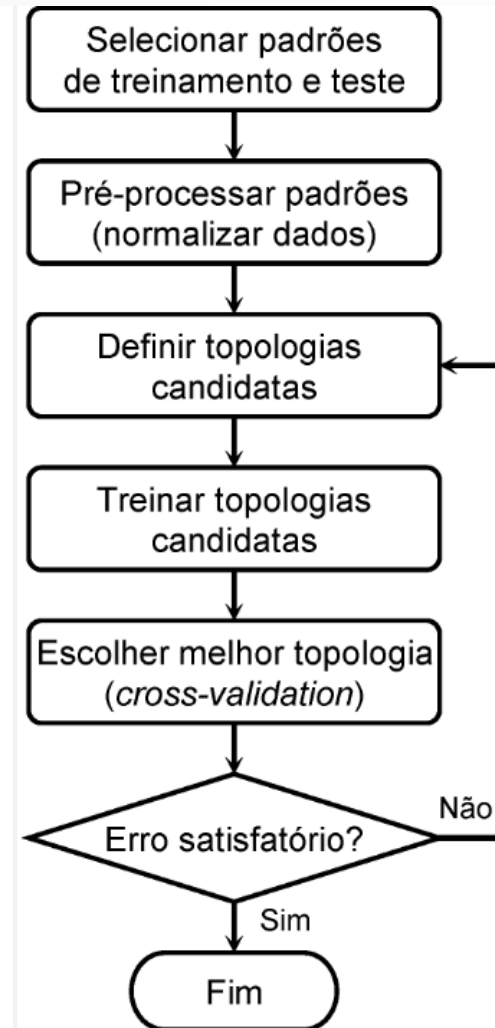
Como teste final para saber se a rede convergiu, faz-se o teste da rede com padrões não vistos e mede-se as taxas de erro de classificação (ROC), para diferentes configurações de parâmetros de rede.

Conclusões:

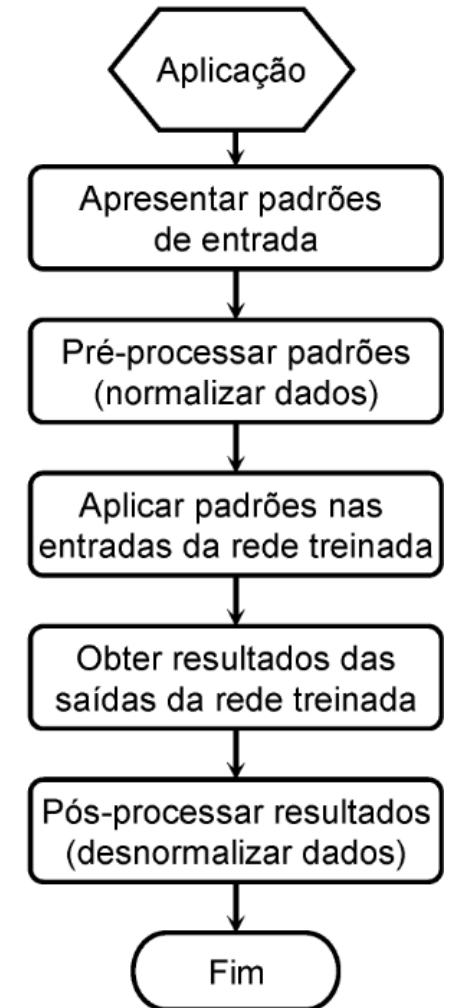
- a) O MLP resolve classificações não linearmente separáveis quando tem, pelo menos, 1 camada oculta
- b) Qualquer função contínua pode ser aproximada por uma MLP de 3 camadas com quantidades suficientes de neurônios na camada oculta.
- c) O MLP usa hiperplanos na forma $f(W^T X) = 0$ para criar as fronteiras de decisão. Portanto, há uma grande dependência da base de dados para criar os espaços decisórios.
- c) A capacidade de generalização e ajustes dos parâmetros exige uma sequência planejada de testes. Não é possível assumir o resultado na primeira vez que se roda a rede.
- d) Comparar os resultados e aplicar um a regra de custo para determinar a melhor configuração de parâmetros. Sempre que publicar um artigo ou texto deixar muito claro qual foi a configuração usada, valores dos parâmetros e etc. para permitir a reprodução dos ensaios e resultados.

Exemplo de Topologias Candidatas:

1. 5 neurônios na camada oculta
2. 10 neurônios na camada oculta
3. 15 neurônios na camada oculta
4. 20 neurônios na camada oculta



(a) Fase de Treinamento



(b) Fase de Operação

Teoria da Aproximação Universal

Teorema do Aproximador Universal

Os Aproximadores Universais de Funções buscam descrever o comportamento de funções complexas pela composição de funções mais simples. No caso das MLPs, o conceito da TAU foi formalizado por Cybenko (1989) a partir do Teorema da Superposição de Kolmogorov (1957). Cybenko provou que um perceptron com uma camada oculta é suficiente para aproximar qualquer função contínua contida num hipercubo de dimensão $\mathbb{I}^n = [0, 1]^n$, que define uma região compacta S do espaço de entrada.

O Teorema do Aproximador Universal estabelece que:

Seja uma função $\varphi(\cdot)$ contínua e não constante, limitada e monotonicamente crescente.

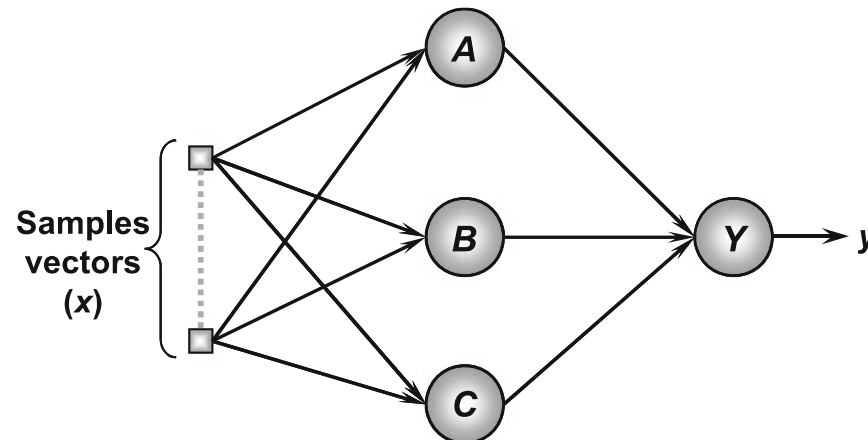
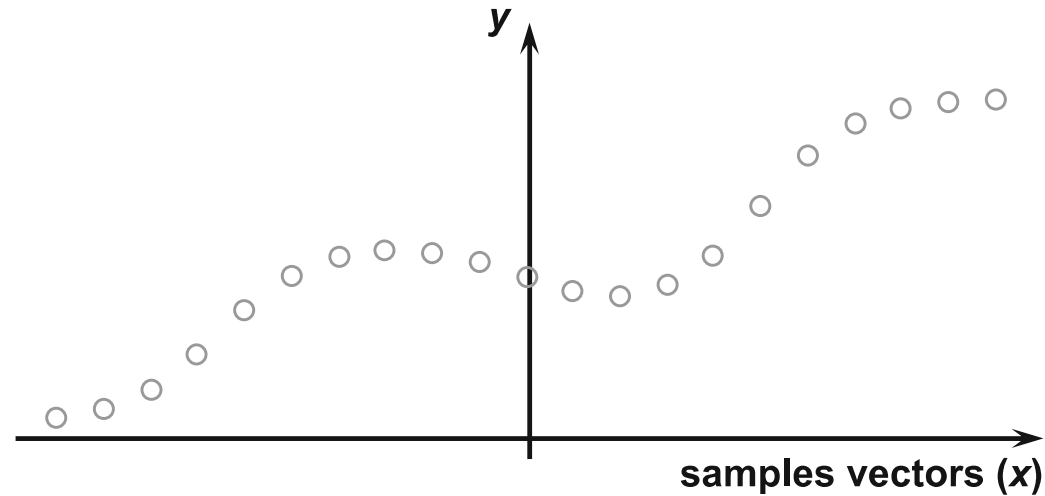
Suponha um hipercubo \mathbb{I}^n de dimensão $[0,1]^n$. Então, para qualquer função $F \subset \mathbb{I}^n$, existe um inteiro p e um conjunto de constantes reais λ_k, θ_k e $w_{i,k}$ tal que:

$$f(X, W) = \sum_{k=1}^p \lambda_k \cdot \varphi \left(\sum_{i=1}^n w_{i,k} x_i - \theta_k \right) \text{ tal que } X \in \mathbb{I}^n \text{ e } W \in \mathbb{R}^{n \times p}$$

onde λ_k é um ponderador da função definida por $\varphi(\cdot)$ para o k -ésimo neurônio. Note que o neurônio de saída neste caso terá uma função de ativação linear.

Teorema do Aproximador Universal

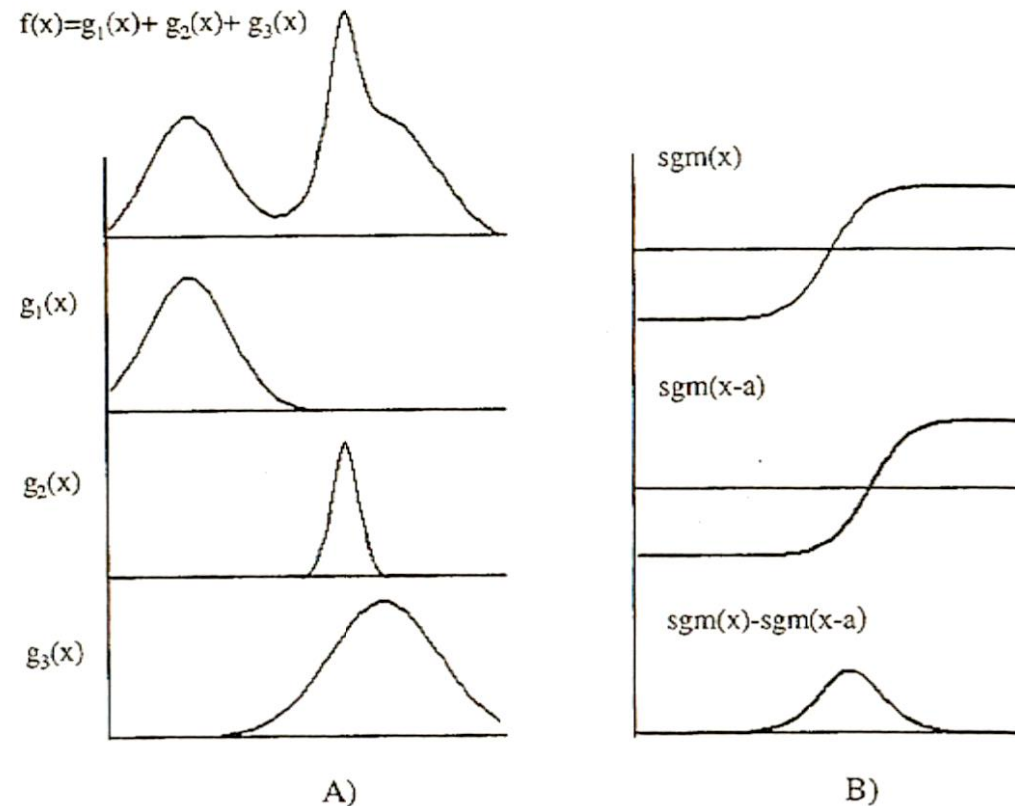
Considere um conjunto de pontos X representado abaixo. Um MLP com 3 neurônios na camada oculta pode representar essa função



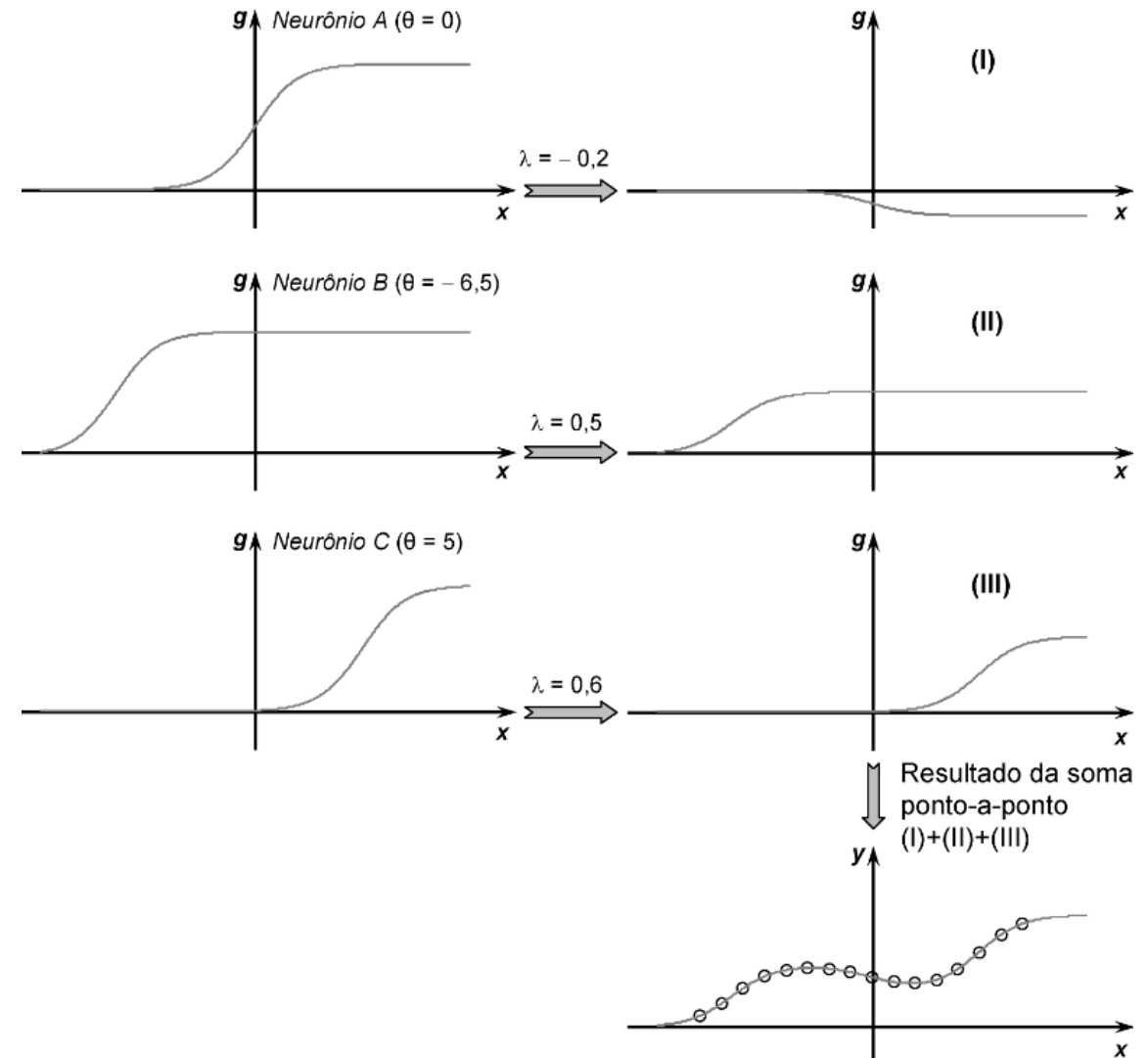
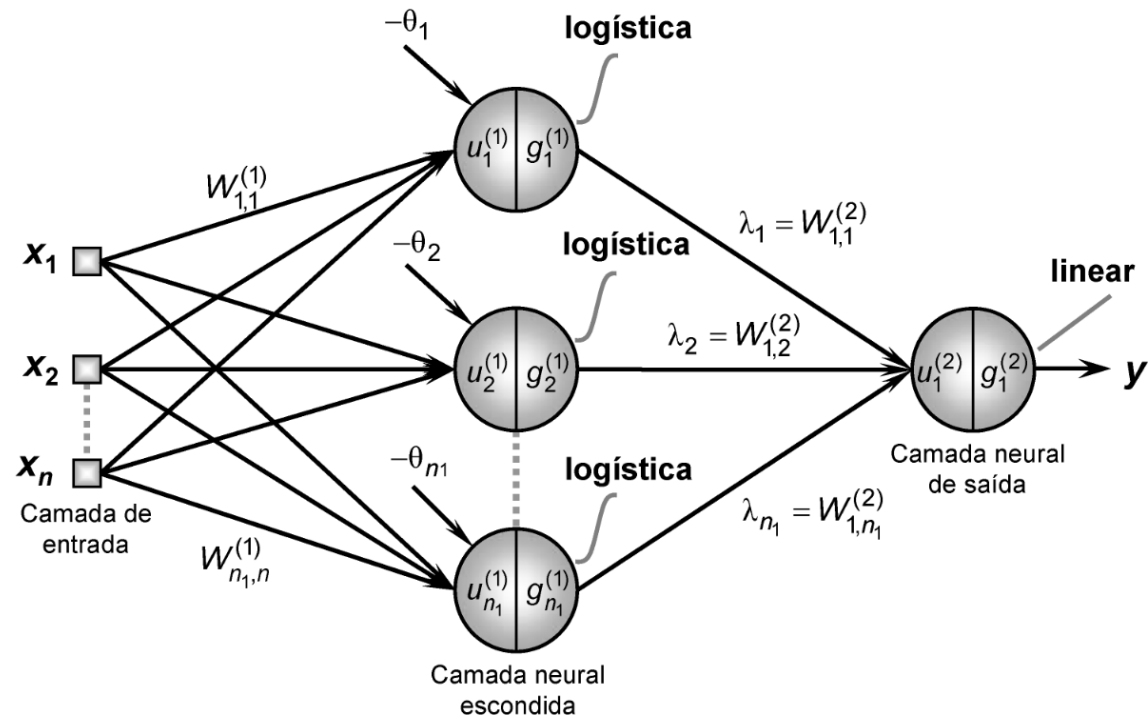
Artificial Neural Networks – A practical course

Teorema do Aproximador Universal

O teorema de Kolmogorov é um teorema de existência e não oferece nenhum procedimento para determinação das funções de ativação. Hornik e Cybenko, demonstraram a capacidade de aproximação através de funções sigmóides. A figura abaixo ilustra uma aproximação da função $f(x)$ através da superposição deslocado de funções gaussianas (a) e a composição de uma função gaussiana através da superposição de duas sigmóides deslocadas..



Redes Neurais Artificiais - Kovacs



Extraído e Adaptado de Ivan Nunes et al., 2 ed.