# Rede Neural MLP: Perceptron de Múltiplas Camadas[1]

## Conteúdo



[1]As figuras nas páginas 2, 40, 49 e 50 foram elaboradas pelo Prof. Leandro N. de Castro, enquanto que a figura na página 44 foi elaborada por Andrea Carolina Peres Kulaif.

# 1. Neurônio artificial

- Modelo matemático: Simplificações da realidade com o propósito de representar aspectos relevantes de um sistema em estudo, sendo que detalhes de menor significância são descartados para viabilizar a modelagem.

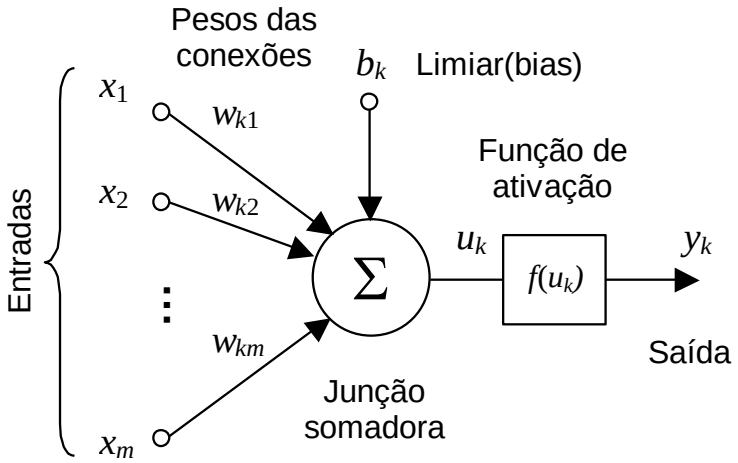



Figura 1 – Modelo matemático de um neurônio artificial

- A saída do neurônio *k* pode ser descrita por: $y_k = f(u_k) = f\left(\sum_{j=1}^{m} w_{kj} x_j + b_k\right)$
- É possível simplificar a notação acima de forma a incluir o bias simplesmente definindo um sinal de entrada de valor $x_0 = 1$ com peso associado $w_{k0} = b_k$:

$$y_k = f(u_k) = f\left(\sum_{j=0}^{m} w_{kj} x_j\right) = f\left(\mathbf{w}^T \mathbf{x}\right)$$

# 2. Exemplos mais usuais de funções de ativação

$$y = f(\mathbf{u}_k) = \frac{e^{p\mathbf{u}_k}}{e^{p\mathbf{u}_k} + 1} = \frac{1}{1 + e^{-p\mathbf{u}_k}} \qquad \frac{dy}{d\mathbf{u}_k} = py(1 - y) > 0$$

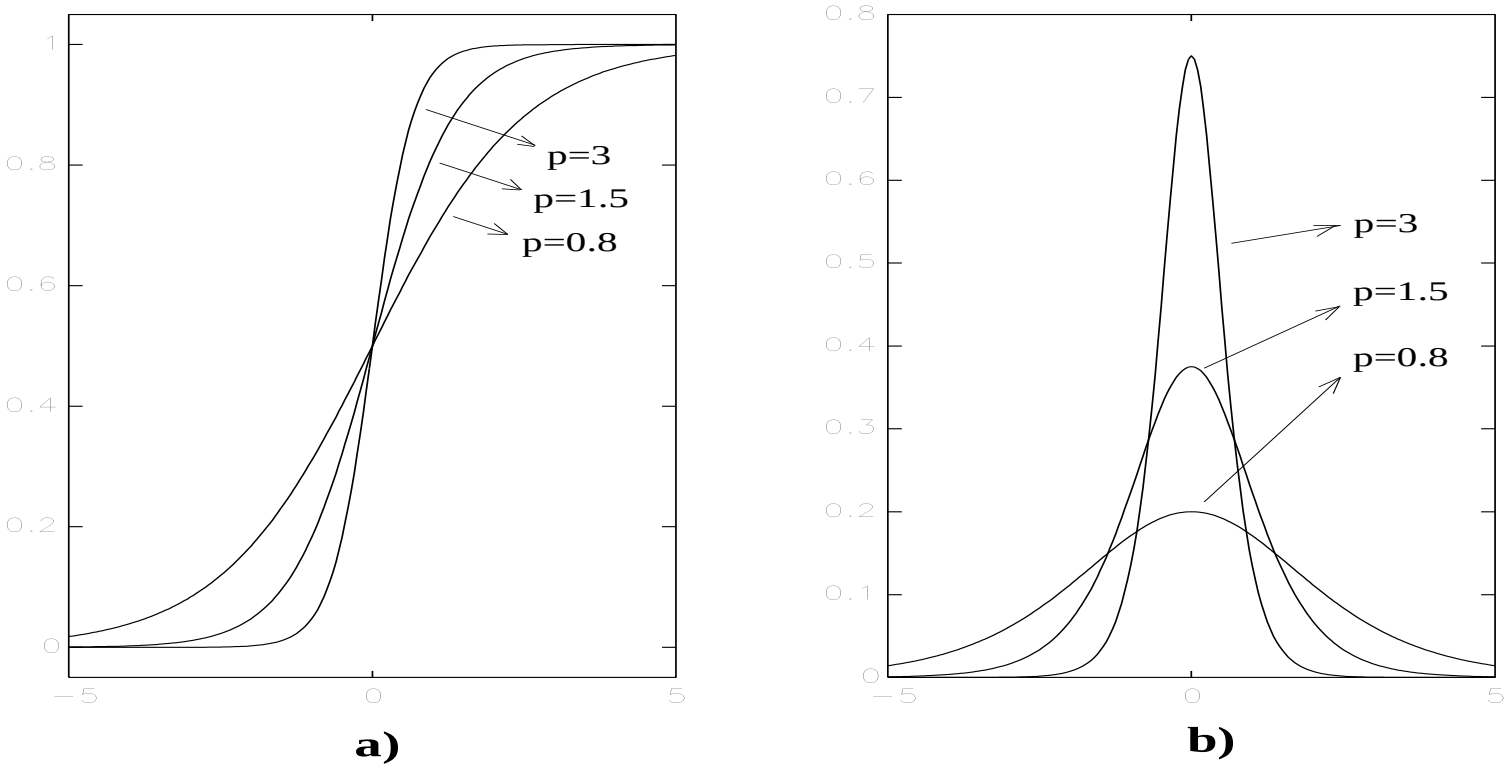



Figura 2 – Função logística (a) e sua derivada em relação à entrada interna (b)

$$y = f(\mathbf{u}_k) = \tanh(p\mathbf{u}_k) = \frac{e^{p\mathbf{u}_k} - e^{-p\mathbf{u}_k}}{e^{p\mathbf{u}_k} + e^{-p\mathbf{u}_k}} \qquad \frac{dy}{d\mathbf{u}_k} = p\left(1 - y^2\right) > 0$$

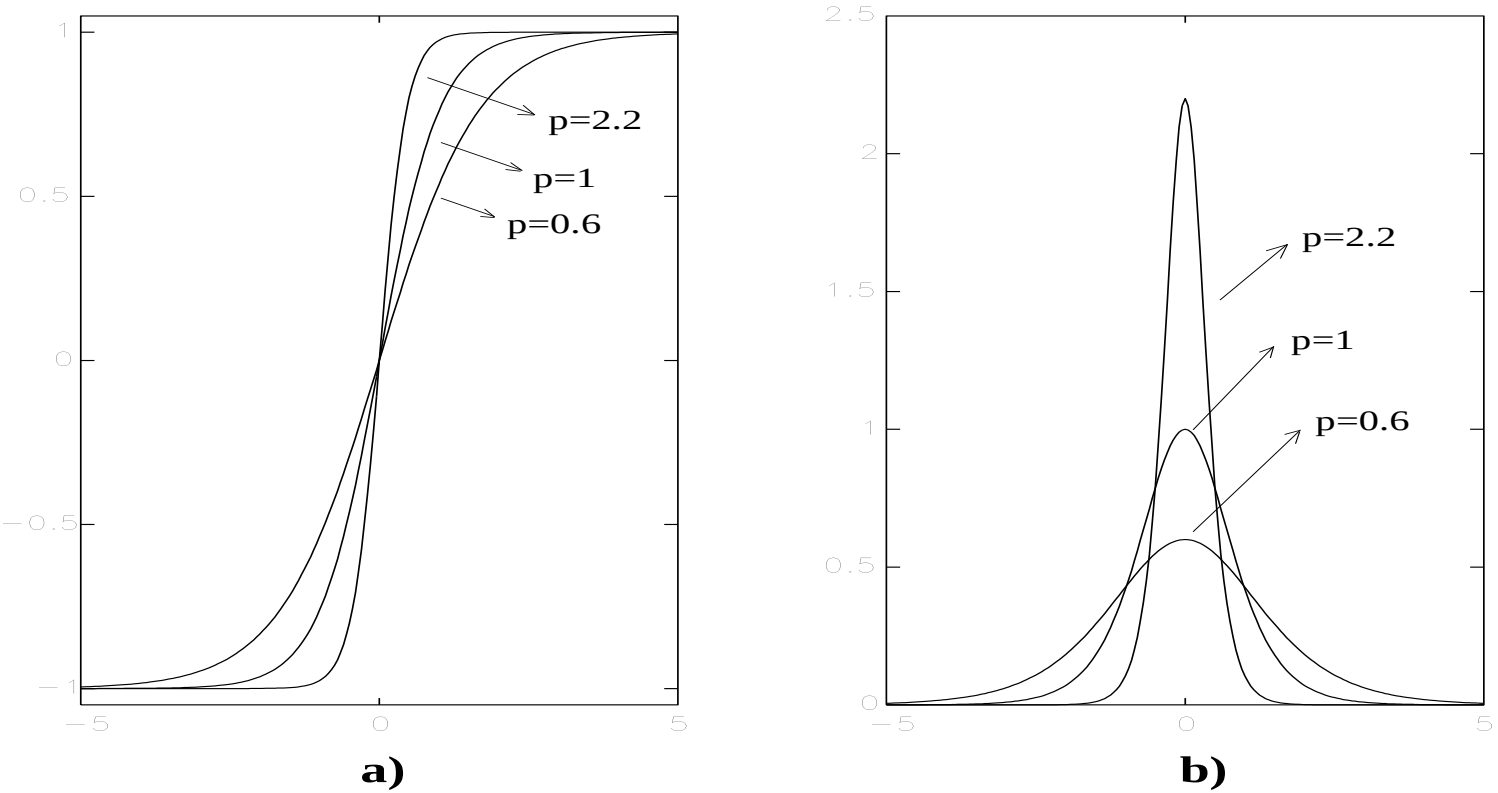


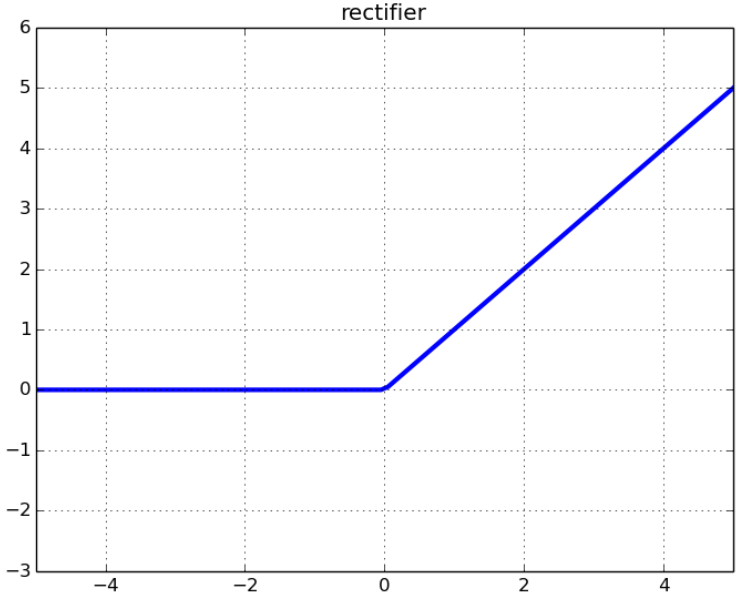

Figura 3 – Função tangente hiperbólica (a) e sua derivada em relação à entrada interna (b)

- Esta será a função de ativação a ser adotada em implementações práticas do curso que não envolvam *deep learning*, com parâmetro $p = 1$ fixo.

- Uma função de ativação que passou a ser largamente utilizada em *deep learning* é a Rectified Linear Unit (ReLU), apresentada a seguir e cuja equação é dada por:

$$y_k = f(u_k) = \max(0, u_k)$$

- Trata-se de uma função linear por partes, que facilita o cálculo do gradiente e que opera como uma "porta lógica" que deixa ou não passar adiante a ativação interna do neurônio.



# 3. Produto interno e projeção

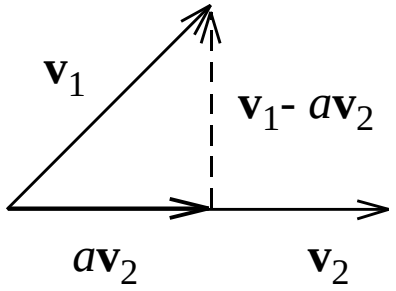



Figura 4 – Projeção realizada pelo produto interno no $\Re^2$

- Sejam $\mathbf{v}_1, \mathbf{v}_2 \in \Re^2$ elementos não-nulos. Considere um escalar $a$ tal que $a\mathbf{v}_2$ corresponda à projeção de $\mathbf{v}_1$ na direção de $\mathbf{v}_2$. Então, pode-se afirmar que

$$a\mathbf{v}_2 \perp \mathbf{v}_1 - a\mathbf{v}_2,$$

  conduzindo a

$$\langle a\mathbf{v}_2, \mathbf{v}_1 - a\mathbf{v}_2 \rangle = 0 .$$

- Logo,

$$a\langle \mathbf{v}_2, \mathbf{v}_1 \rangle - a^2 \langle \mathbf{v}_2, \mathbf{v}_2 \rangle = 0,$$

permitindo obter *a* na forma

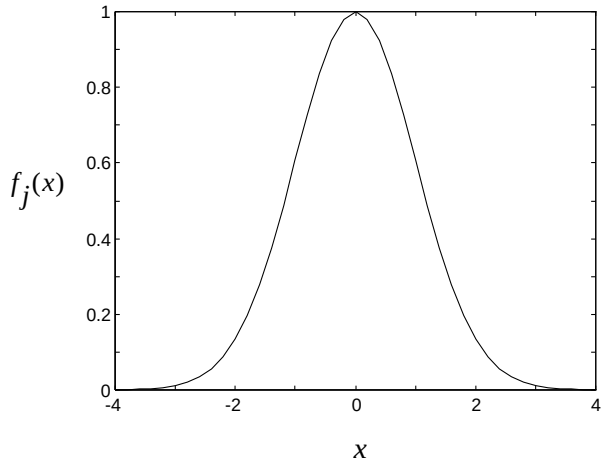
$$a = \frac{\langle \mathbf{v}_2, \mathbf{v}_1 \rangle}{\langle \mathbf{v}_2, \mathbf{v}_2 \rangle}.$$

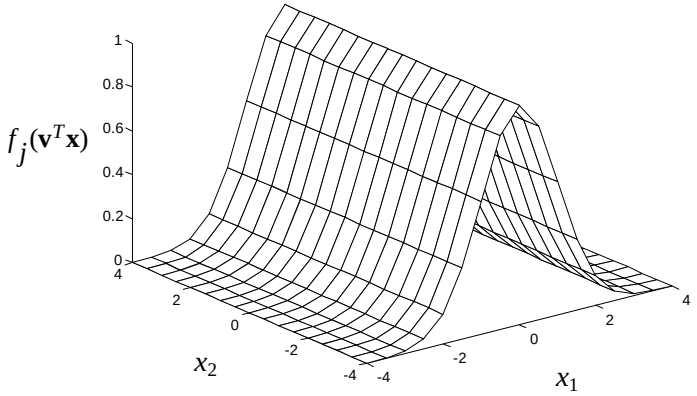
- Isto significa que a projeção de $\mathbf{v}_1$ na direção de $\mathbf{v}_2$ ($\mathbf{v}_2 \neq \mathbf{0}$) assume a forma:

$$\text{proj}_{\mathbf{v}_2}(\mathbf{v}_1) = \frac{\langle \mathbf{v}_2, \mathbf{v}_1 \rangle}{\langle \mathbf{v}_2, \mathbf{v}_2 \rangle} \mathbf{v}_2$$

- Mantendo constante o módulo de $\mathbf{v}_1$, a sua projeção na direção de $\mathbf{v}_2$ é tão maior quanto mais colineares forem esses dois vetores.
- O produto interno, portanto, permite o estabelecimento de uma associação bem definida entre o vetor de estímulos de entrada de um neurônio e o seu vetor de pesos sinápticos, de modo que o neurônio ativa mais quanto mais colineares forem esses dois vetores, supondo norma constante para esses vetores.

# 4. Função de expansão ortogonal




(a) $f_j(x) = e^{-0,5 \cdot x^2}$ (b) $f_j(\mathbf{v}^T\mathbf{x}) = e^{-0,5 \cdot \left( [1\ 0] \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \right)^2}$

Figura 5 – Função de expansão ortogonal em que $\mathbf{v} = [1\ 0]^T$ e $\mathbf{x} = [x_1\ \ x_2]^T$

- A função de expansão ortogonal é conhecida na literatura em língua inglesa como *ridge function*. Este conceito já foi evidenciado no Tópico 4 – Parte 3.

# 5. Redes neurais e perceptron com uma camada intermediária

- O processo de conexão entre neurônios artificiais leva à geração de sinapses e à construção de redes neurais artificiais.

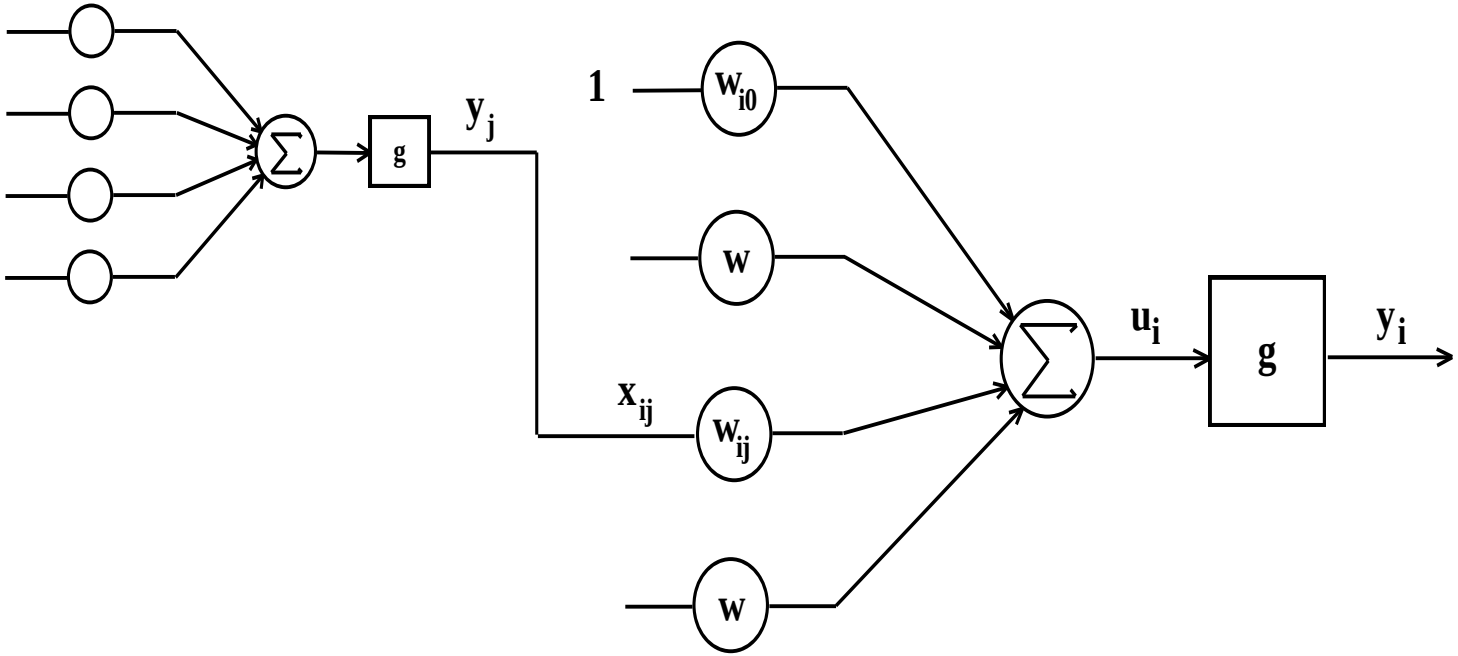



Figura 6 – Estabelecimento de conexão entre dois neurônios artificiais

- As estruturas mais conhecidas são em camadas, onde a saída de cada neurônio de uma camada precedente é entrada para todos os neurônios da camada seguinte.

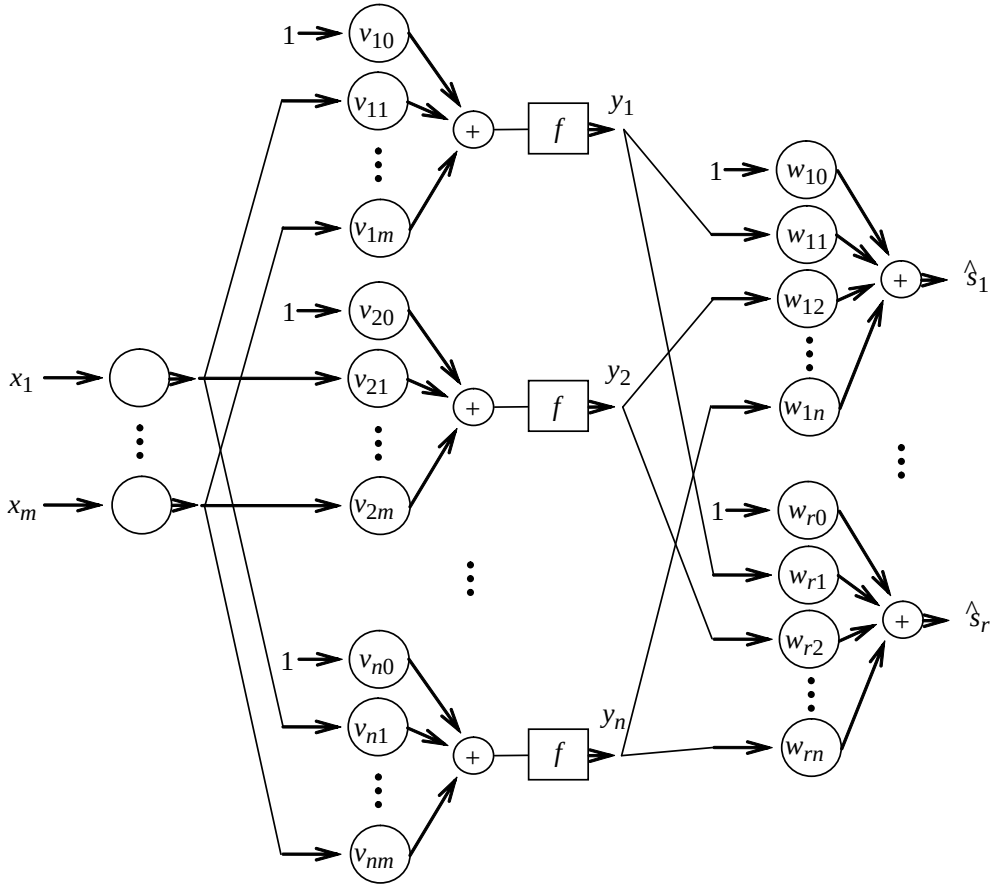



Figura 7 – Rede neural perceptron com uma camada intermediária
Do inglês *Multilayer Perceptron* (MLP)

$$\hat{s}_k = \sum_{j=0}^{n} w_{kj} f\left( \sum_{i=0}^{m} v_{ji} x_i \right) = \sum_{j=0}^{n} w_{kj} f\left(\mathbf{v}_j^T \mathbf{x}\right) = \hat{g}_k\left(\mathbf{x}, \theta\right), k = 1,...,r$$

# 6. Contribuição de cada neurônio em uma rede MLP

- O mapeamento não-linear realizado por uma rede neural do tipo perceptron de uma camada intermediária é uma **combinação linear de funções de expansão ortogonal**, ou seja, funções que têm a forma de tangente hiperbólica em uma direção e são constantes nas demais direções ortogonais a esta única direção em que a forma da função se manifesta.
- Como um exemplo, vamos tomar amostras de um mapeamento do $\Re^2$ para o $\Re^1$, e utilizar uma rede neural com 5 neurônios na camada intermediária para buscar aproximar este mapeamento, o qual pode ser visualizado no $\Re^3$.
- Os pesos sinápticos resultantes do processo de treinamento estão apresentados na sequência, sendo que a rede neural tem ao todo $3 \times 5 + 6 \times 1 = 21$ pesos ajustáveis. São 2 entradas, 5 neurônios na camada intermediária e 1 saída, mais as entradas constantes (entradas de polarização) de todos os 6 neurônios da rede neural.

- Pesos sinápticos da camada intermediária (cada coluna representa os pesos de um neurônio):

-0.20008939714462 -0.70051908010040 0.39699221844113 -0.10003863267278 0.69606262467282

0.70018168528932 0.10015860417667 0.19860028823484 -0.29996195303800 0.29869112235480

-0.30006398146599 0.80022209855791 0.49372400421686 0.50005427222963 0.89515012131364

- Pesos sinápticos da camada de saída:

0.99989340388393

0.79971888341317

0.90007841696146

0.38564988369799

0.79996881679466

0.71442550587375

- Obs: O peso de polarização é o primeiro peso de cada neurônio.

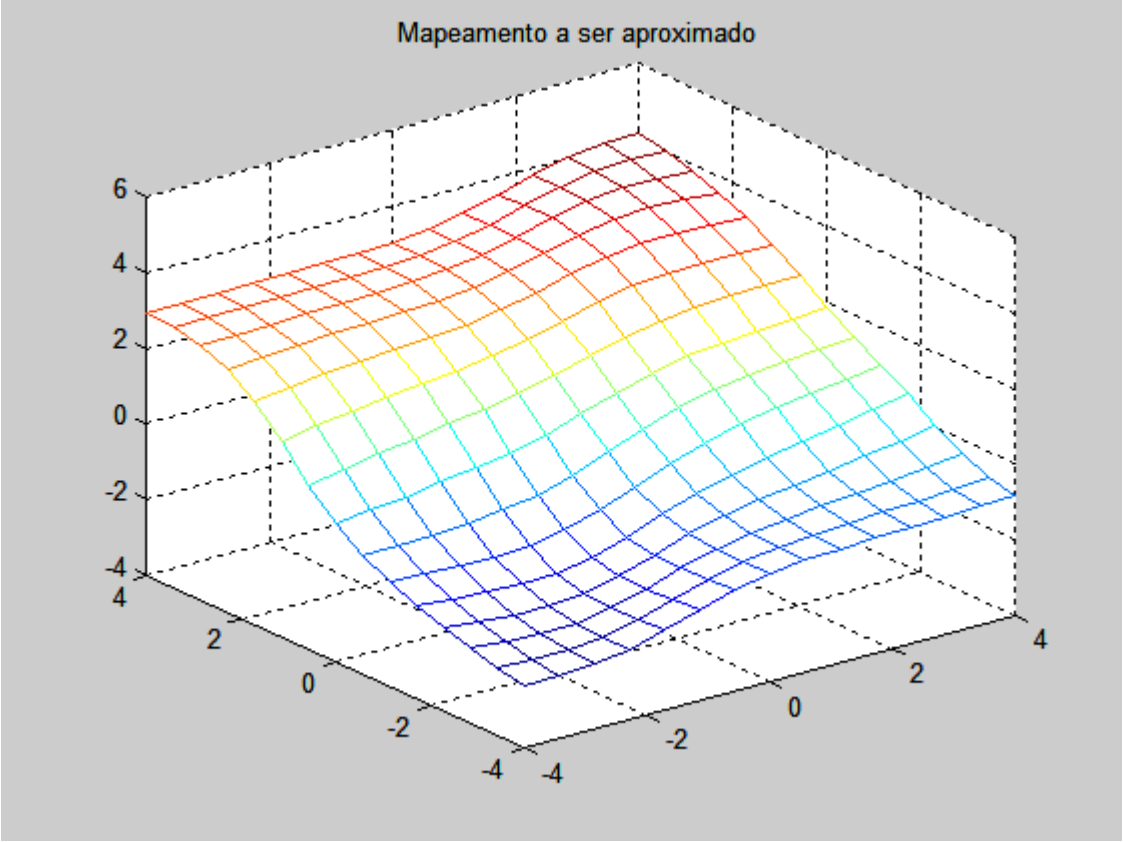



Figura 8 – Mapeamento a ser aproximado

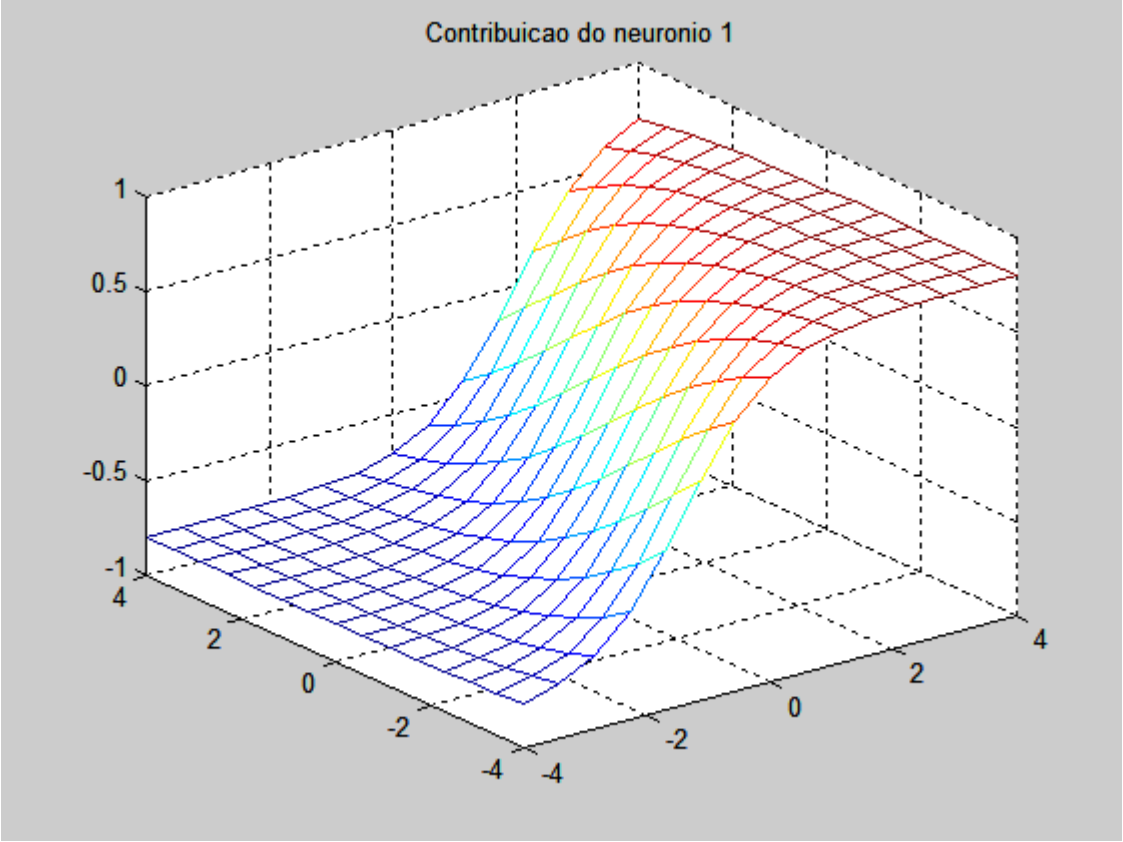



Figura 9 – Contribuição do neurônio 1

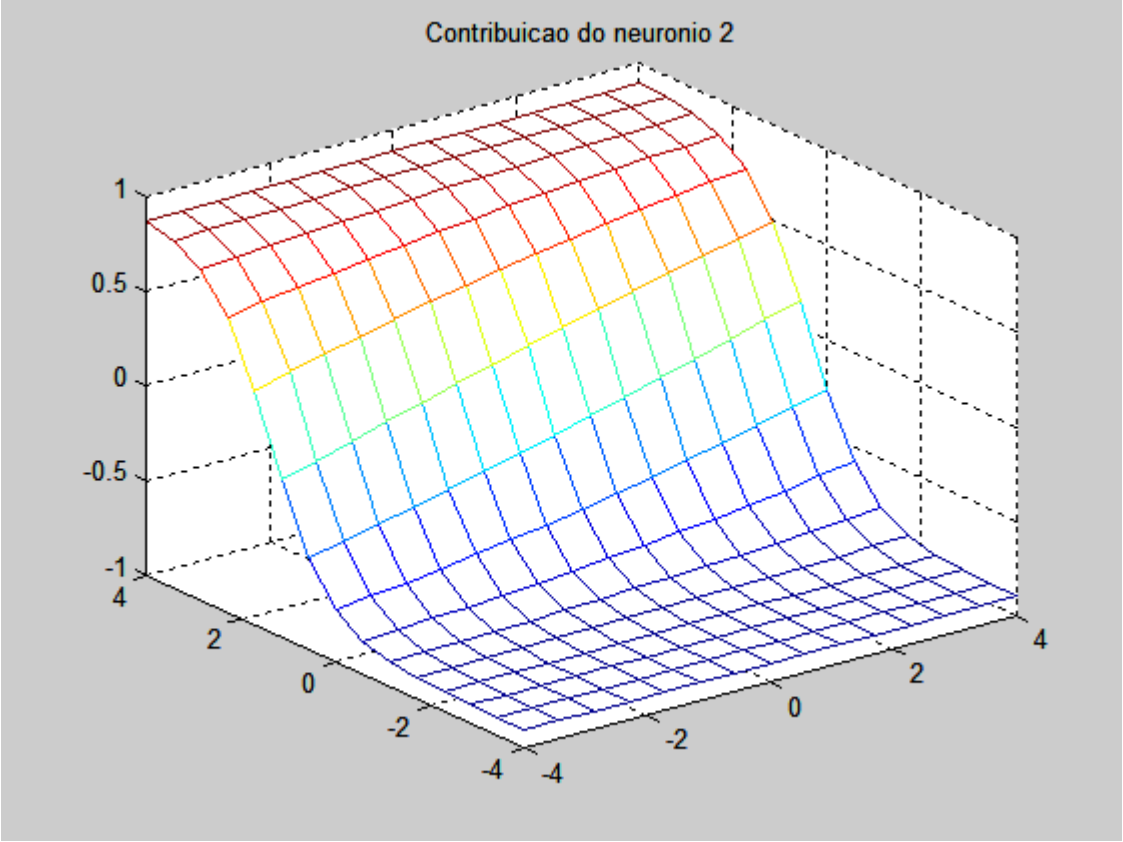



Figura 10 – Contribuição do neurônio 2

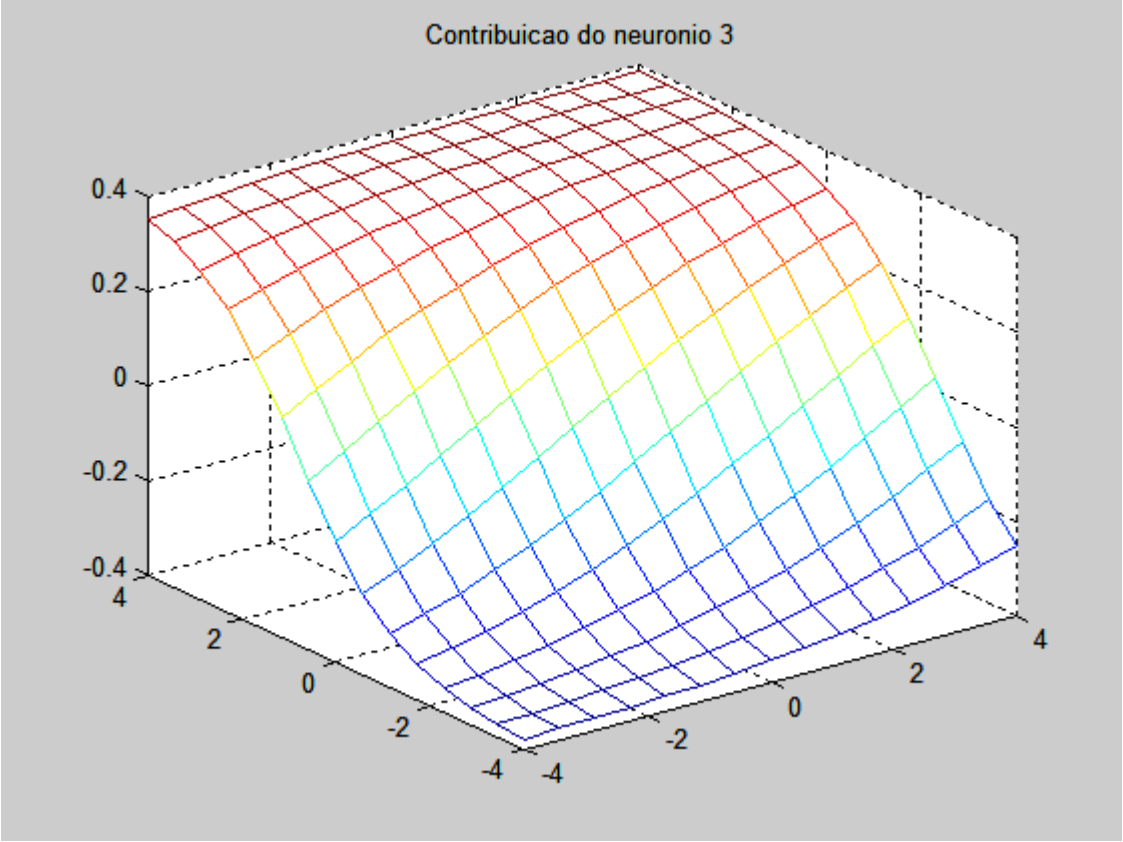



Figura 11 – Contribuição do neurônio 3

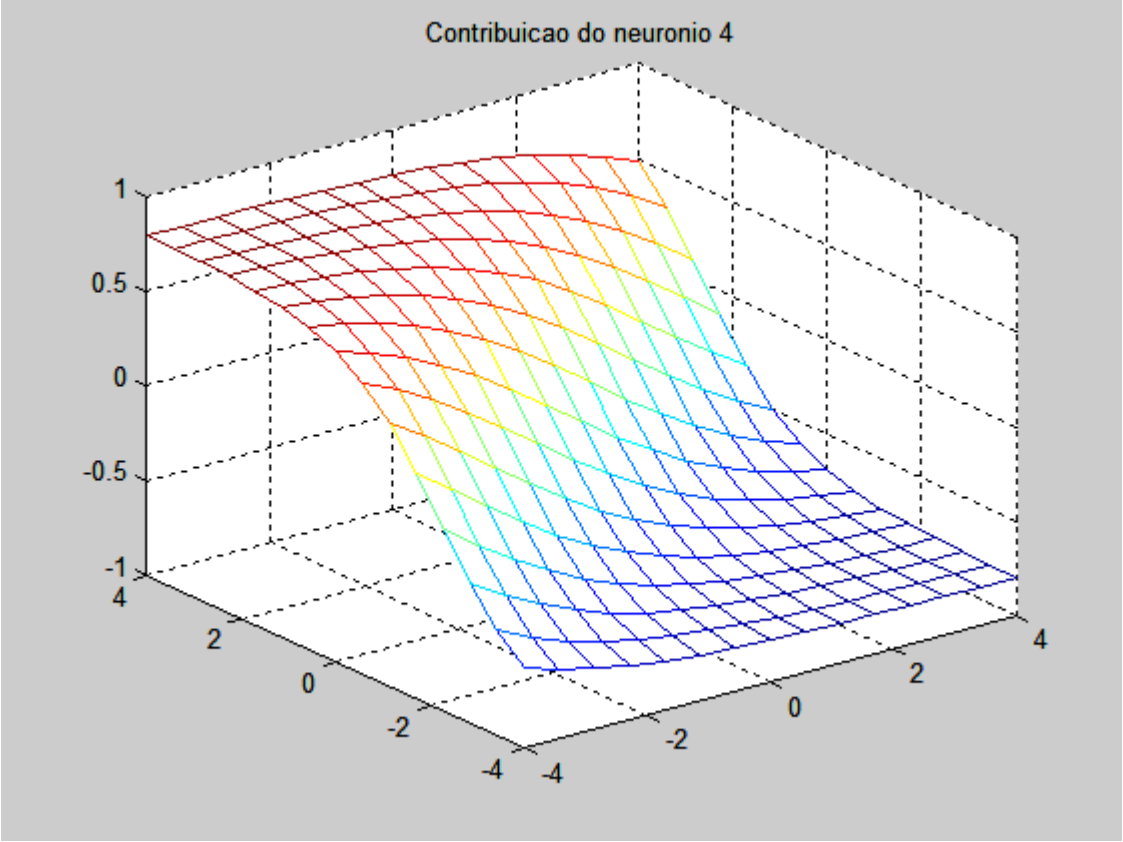



Figura 12 – Contribuição do neurônio 4

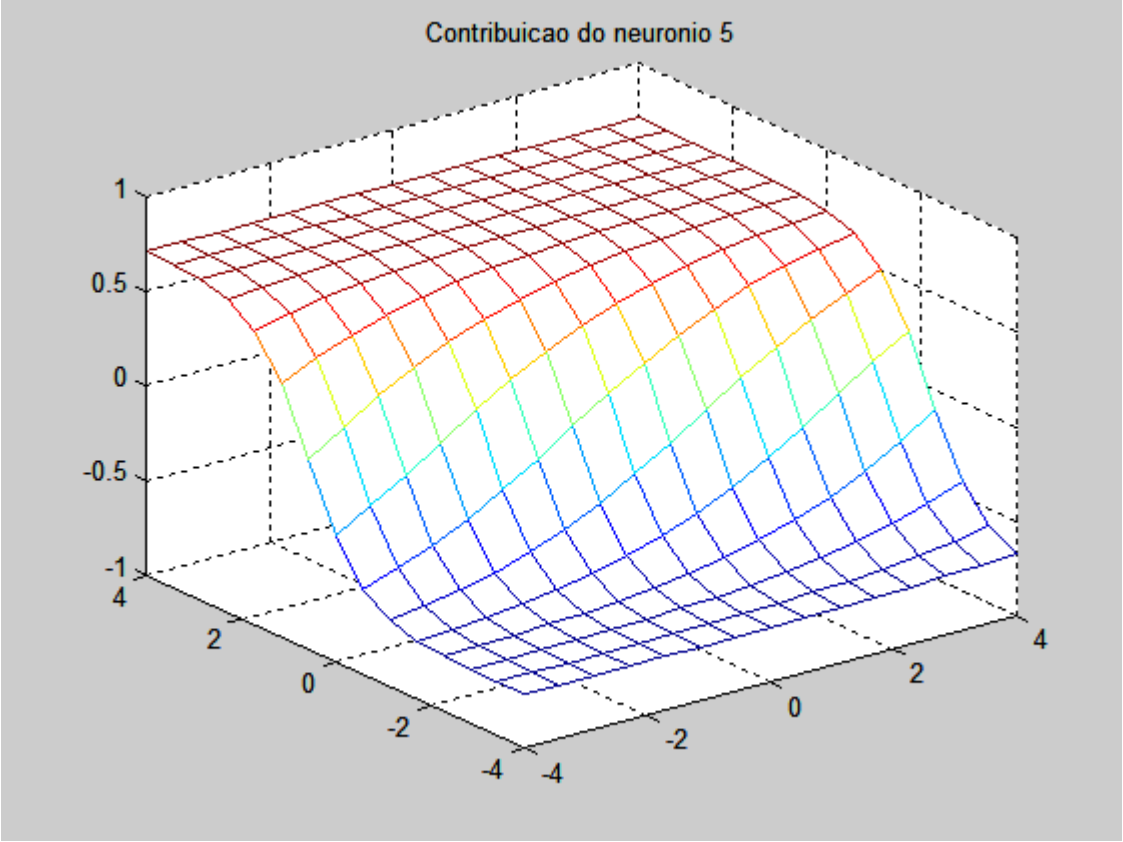
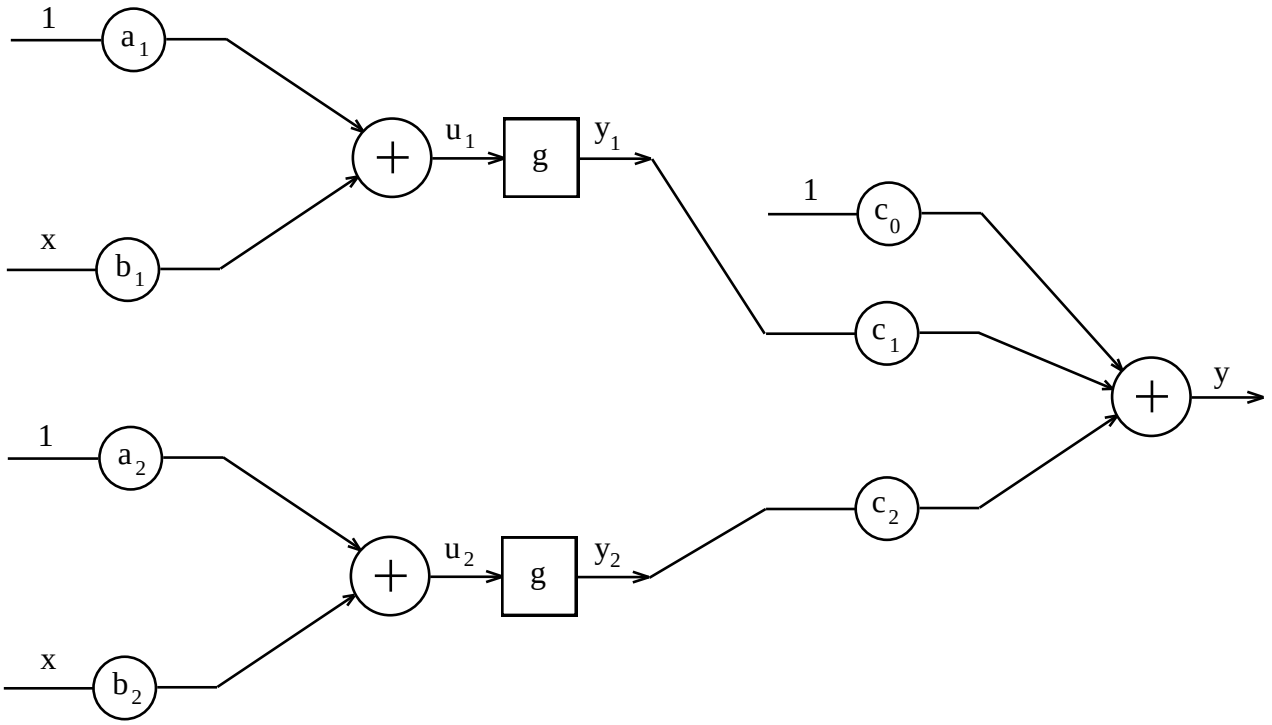



Figura 13 – Contribuição do neurônio 5

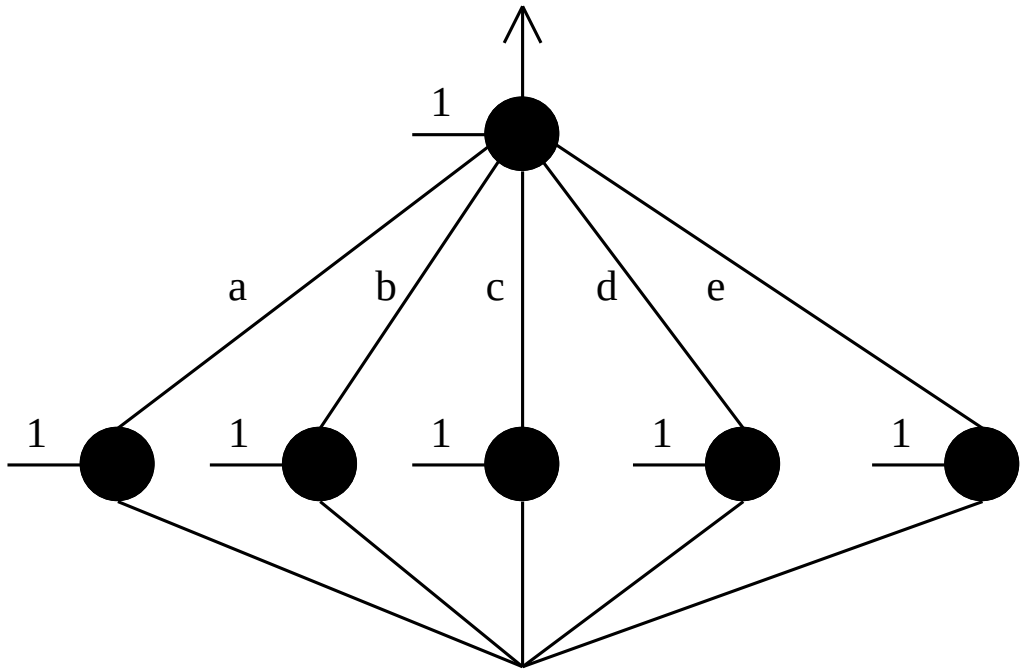
# 7. O papel dos pesos sinápticos

$$y = c_0 + \sum_{n=1}^{p} c_n g(b_n x + a_n)$$




$$y = c_0 + c_1 g(b_1 x + a_1) + c_2 g(b_2 x + a_2) \Rightarrow \begin{cases} \text{a : deslocamento no eixo x} \\ \text{b : inclinação da sigmóide} \\ \text{c : amplitude da sigmóide} \end{cases}$$

**Exemplo**: Forma "construtiva" de aproximação de um mapeamento não-linear empregando neurônios com função de ativação do tipo tangente hiperbólica. Exemplo considerando um único estímulo de entrada.




$$f(\mathbf{w}) = \underbrace{c_1 g(b_1 x + a_1)}_{a} + \underbrace{c_2 g(b_2 x + a_2)}_{b} + \underbrace{c_3 g(b_3 x + a_3)}_{c} + \underbrace{c_4 g(b_4 x + a_4)}_{d} + \underbrace{c_5 g(b_5 x + a_5)}_{e} + \underbrace{c_0}_{bias}$$

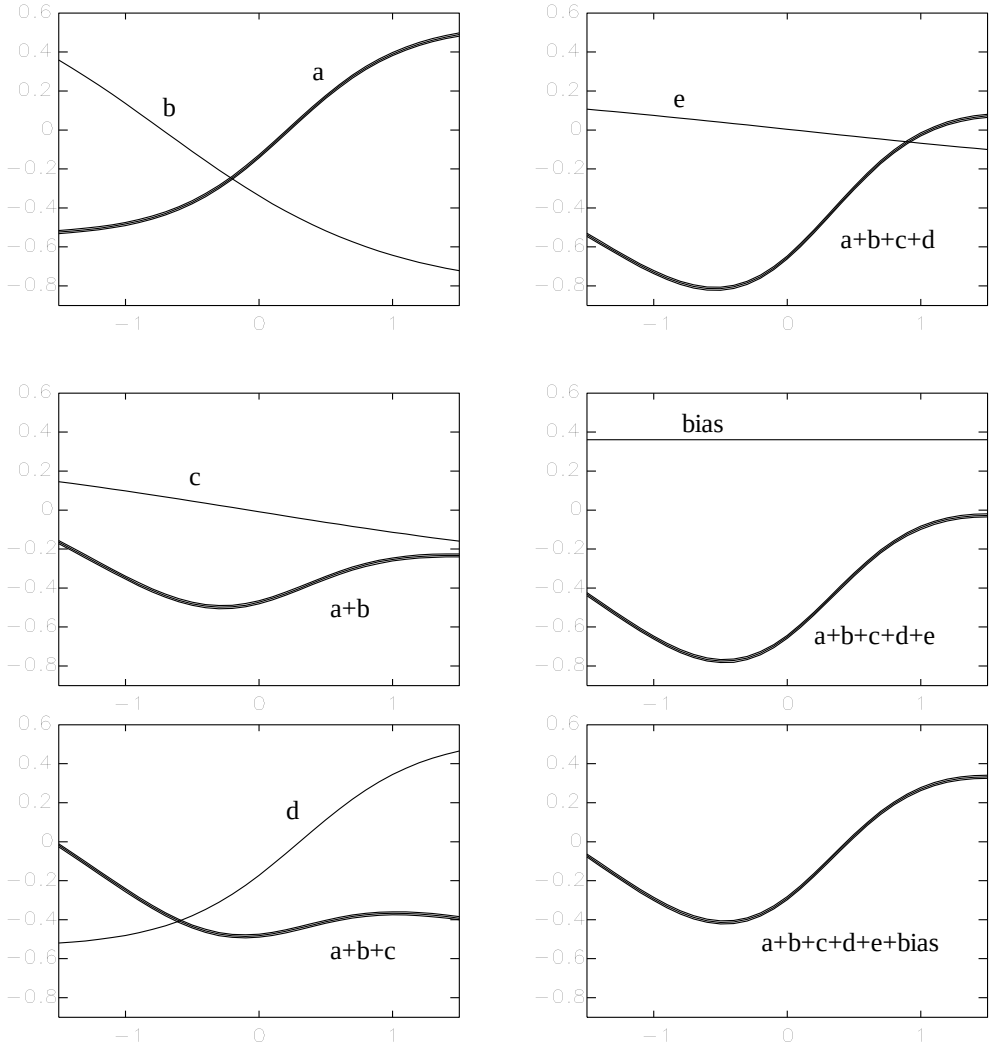



Figura 14 – Composição aditiva de ativações na reprodução de um mapeamento não-linear

O mapeamento a ser aproximado encontra-se na última figura à direita.

# 8. Superfície de erro

- Seja $X$ uma região compacta do $\Re^m$ e seja $g: X \subset \Re^m \rightarrow \Re$ a função a ser aproximada (formulação para uma única saída, $r = 1$);
- O conjunto de dados de aproximação $\{(\mathbf{x}_l, s_l) \in \Re^m \times \Re\}_{l=1}^N$ é gerado considerando-se que os vetores de entrada $\mathbf{x}_l$ estão distribuídos na região compacta $X \subset \Re^m$ de acordo com uma função densidade de probabilidade fixa $d_P$: $X \subset \Re^m \rightarrow [0,1]$ e que os vetores de saída $s_l$ são produzidos pelo mapeamento definido pela função $g$ na forma:

$$s_l = g(\mathbf{x}_l) + \varepsilon_l, \quad l = 1,\ldots,N,$$

  onde $\varepsilon_l \in \Re$ é uma variável aleatória de média zero e variância fixa.
- A função $g$ que associa a cada vetor de entrada $\mathbf{x} \in X$ uma saída escalar $s \in \Re$ pode ser aproximada com base no conjunto de dados de aproximação $\{(\mathbf{x}_l, s_l) \in \Re^m \times \Re\}_{l=1}^N$ por uma composição aditiva de funções de expansão ortogonal na forma:

$$\hat{s}_l = \hat{g}(\mathbf{x}_l, \theta) = \sum_{j=0}^{n} w_j \, f\left( \sum_{i=0}^{m} v_{ji} x_{li} \right) = \sum_{j=0}^{n} w_j \, f\left( \mathbf{v}_j^T \mathbf{x}_l \right)$$

onde $\theta$ é o vetor contendo todos os pesos da rede neural.

- Logo, o erro quadrático médio produzido na saída da rede neural, considerando as $N$ amostras, assume a forma:

$$J(\theta) = \frac{1}{N} \sum_{l=1}^{N} (\hat{s}_l - s_l)^2 = \frac{1}{N} \sum_{l=1}^{N} (\hat{g}(\mathbf{x}_l, \theta) - s_l)^2 =$$

$$= \frac{1}{N} \sum_{l=1}^{N} \left( \sum_{j=0}^{n} w_j \, f\left( \sum_{i=0}^{m} v_{ji} x_{li} \right) - s_l \right)^2 = \frac{1}{N} \sum_{l=1}^{N} \left( \sum_{j=0}^{n} w_j \, f\left( \mathbf{v}_j^T \mathbf{x}_l \right) - s_l \right)^2$$

- Sendo $P$ a dimensão do vetor $\theta$, então tem-se que: $J : \Re^P \rightarrow \Re^1$.
- A superfície de erro definida por $J(\theta)$ reside no espaço $\Re^{P+1}$, sendo que deve-se buscar em $\Re^P$ um ponto que minimiza $J(\theta)$, supondo que se queira minimizar o erro entre a saída produzida pelo rede neural e a saída desejada.

# 9. Aprendizado a partir de dados amostrados

- O aprendizado supervisionado visto como um problema de otimização não-linear sem restrições sobre os valores dos pesos sinápticos.
- A função objetivo (critério de desempenho a ser otimizado, no caso, erro quadrático médio a ser minimizado) e os parâmetros ajustáveis:

$$\min_{\theta \in \Re^P} J(\theta)$$

- Formalização matemática do que se quer otimizar + método de solução.
- Solução na forma fechada × Busca iterativa.
- Os dados de entrada/saída e a questão dos 3 mapeamentos envolvidos no processo:
  1. O mapeamento a ser aproximado (do qual se conhece apenas um conjunto finito de dados amostrados);
  2. O mapeamento resultante do processo de aproximação, associado a um único vetor de pesos sinápticos;
  3. O mapeamento entre cada vetor de pesos e o erro: superfície de erro.

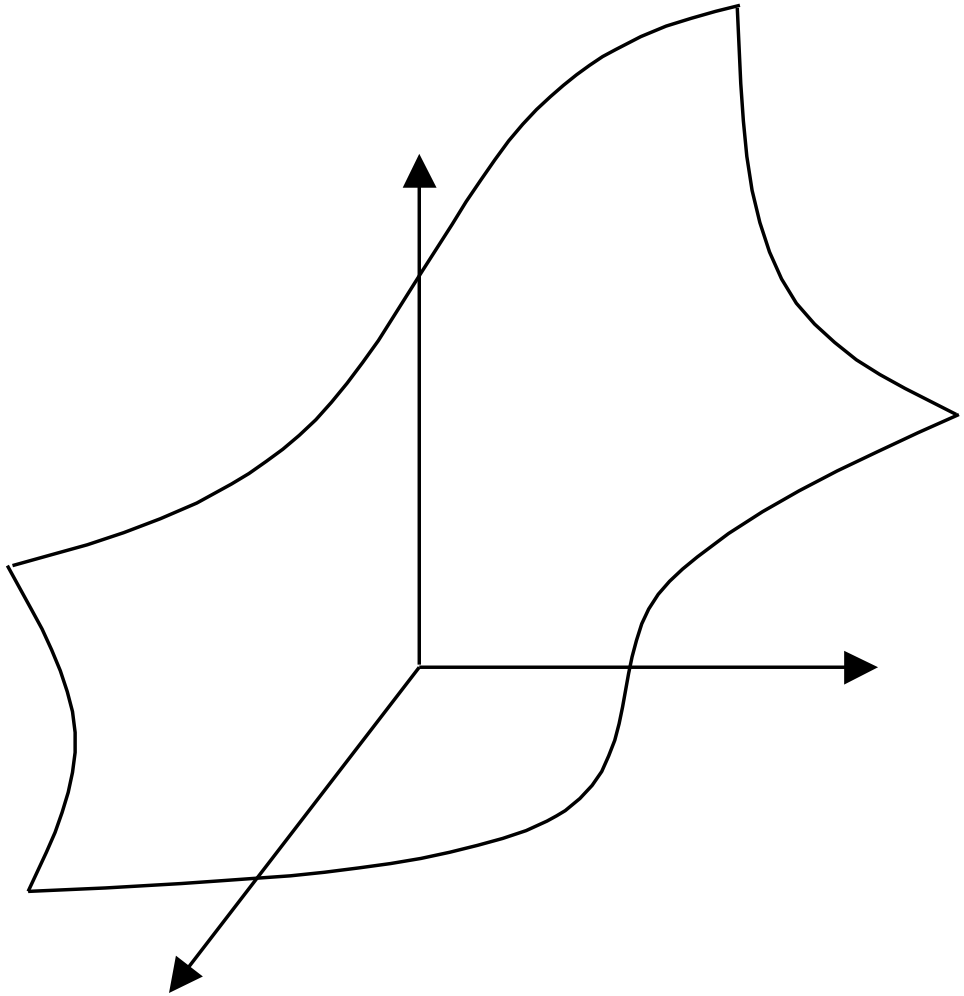


Figura 15– Mapeamento desconhecido a ser aproximado

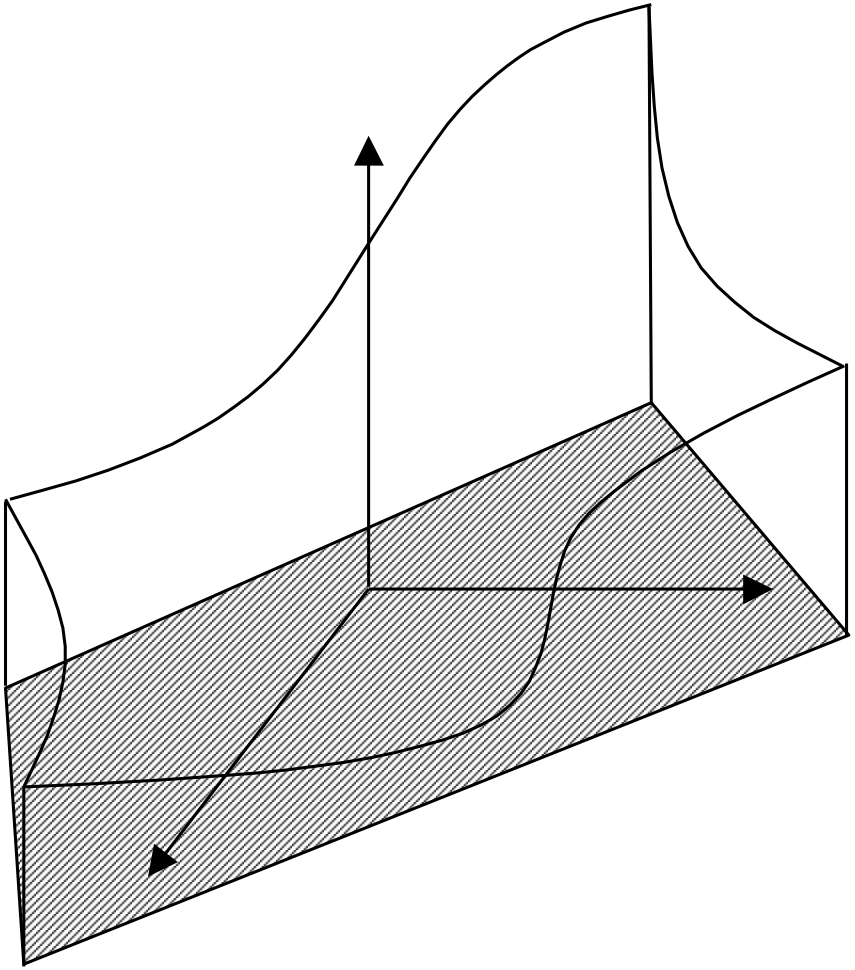


Figura 16 – Exemplo de região de operação. É uma região compacta (fechada e limitada).

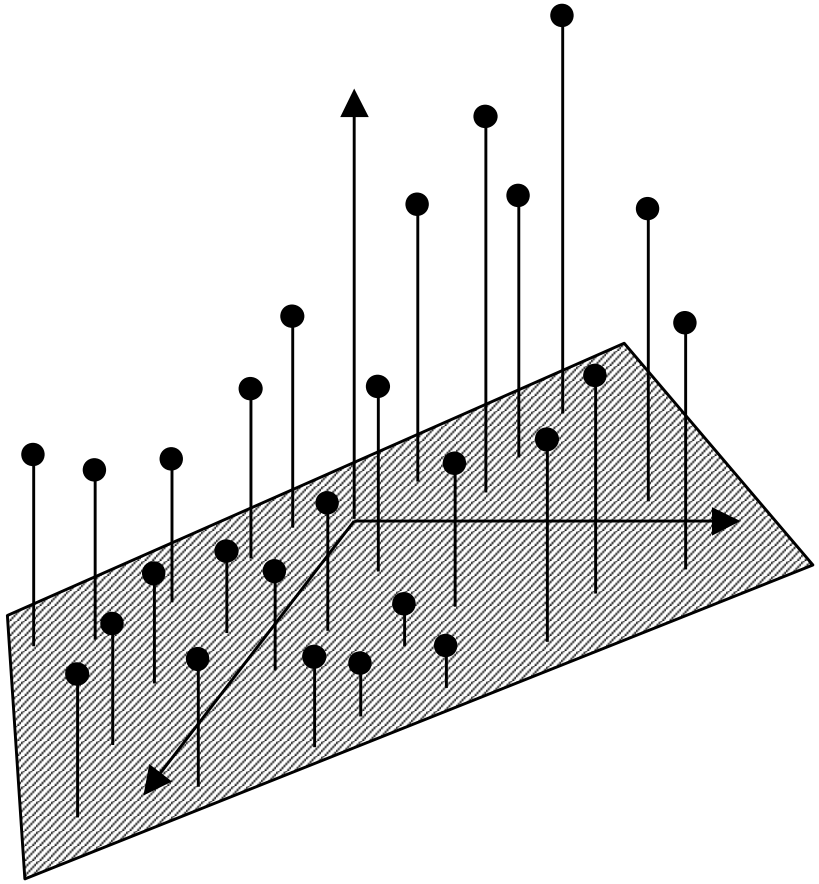


Figura 17 – Amostras expressando o comportamento da função para pontos específicos da região de operação. Essas amostras comporão os conjuntos de treinamento e validação (sendo que os dois conjuntos são independentes entre si)

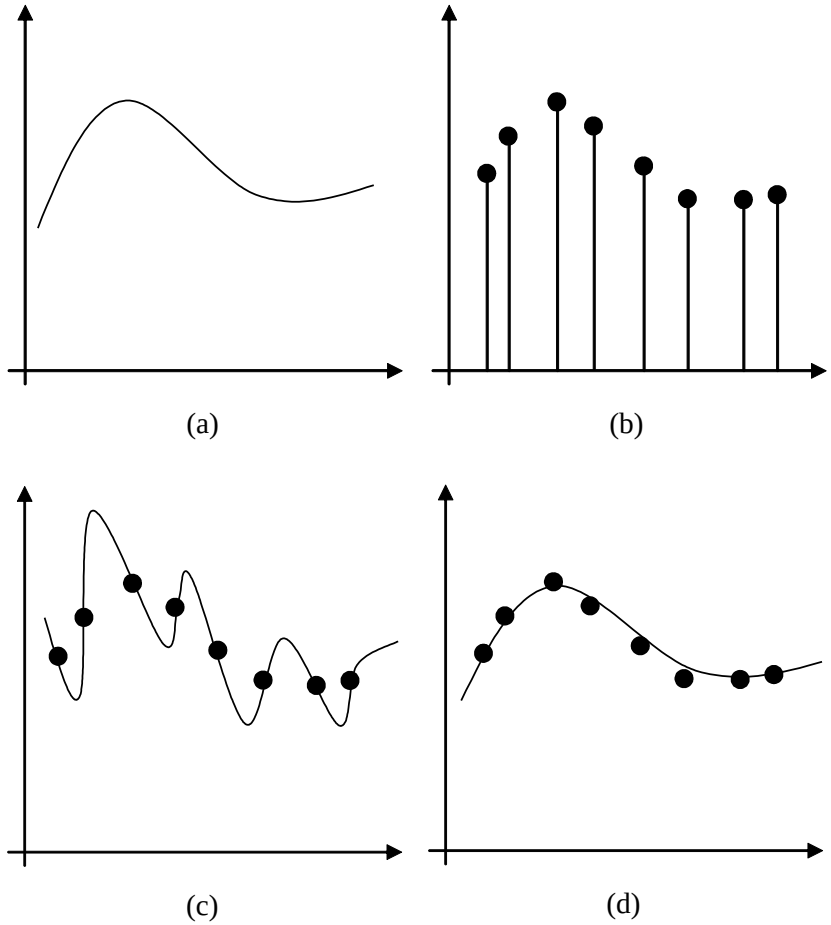



Figura 18 – (a) Função a ser aproximada (agora considerando apenas uma entrada); (b) Amostras disponíveis; (c) Resultado de um processo de aproximação com sobretreinamento; (d) Resultado de um processo de aproximação sem sobretreinamento.

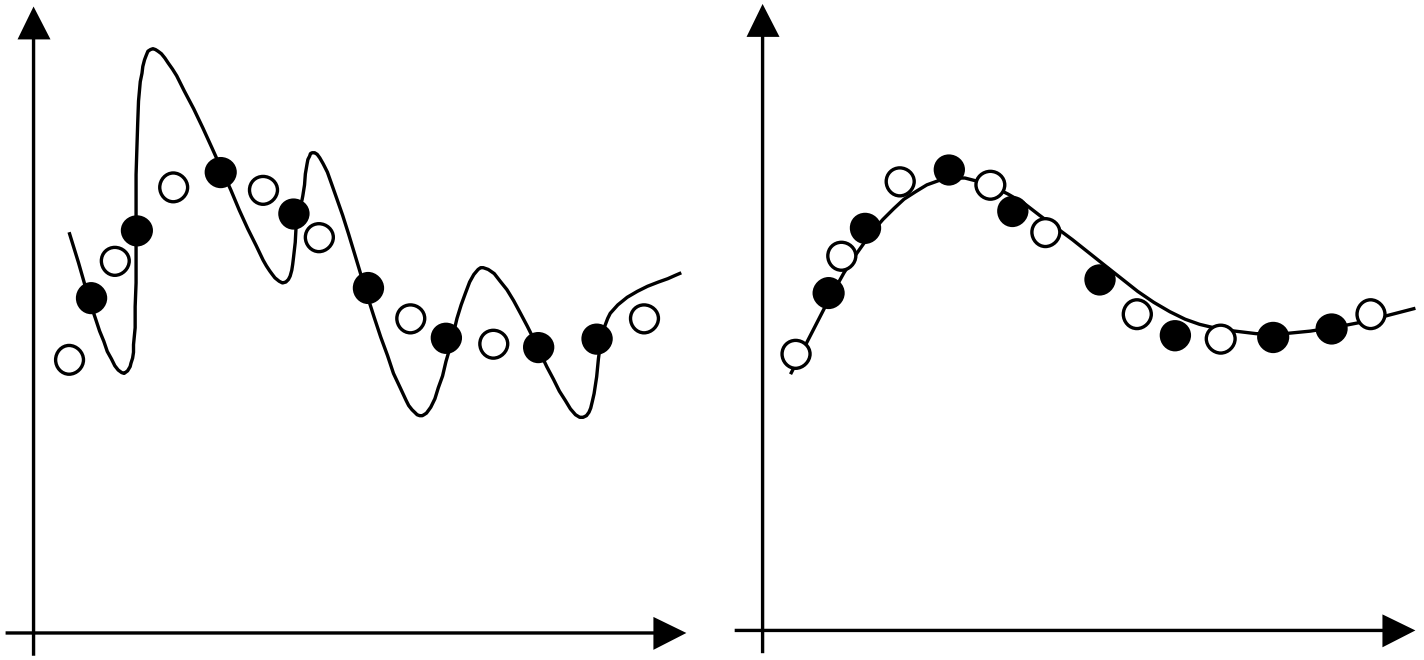



Figura 19 – Comparação de desempenho para dados de treinamento e validação, de modo a medir a capacidade de generalização dos mapeamentos produzidos.

- O mapeamento da esquerda apresenta um erro de treinamento muito baixo, mas um erro de validação bastante elevado, quando comparado ao mapeamento da direita.

# 10. O problema do OU-exclusivo em MLP

- Considere os pontos (0,0),(0,1),(1,0) e (1,1) no plano $\Re^2$, conforme apresentado na Figura 20. O objetivo é determinar uma rede com duas entradas $\mathbf{x}_i \in \{0,1\}$ ($i$=1,2), e uma saída $\mathbf{y} \in \{0,1\}$ de maneira que: $\begin{cases} (\mathbf{x_1}, \mathbf{x_2}) = (0,0) \text{ ou } (1,1) \Rightarrow \mathbf{y} = 0 \\ (\mathbf{x_1}, \mathbf{x_2}) = (1,0) \text{ ou } (0,1) \Rightarrow \mathbf{y} = 1 \end{cases}$

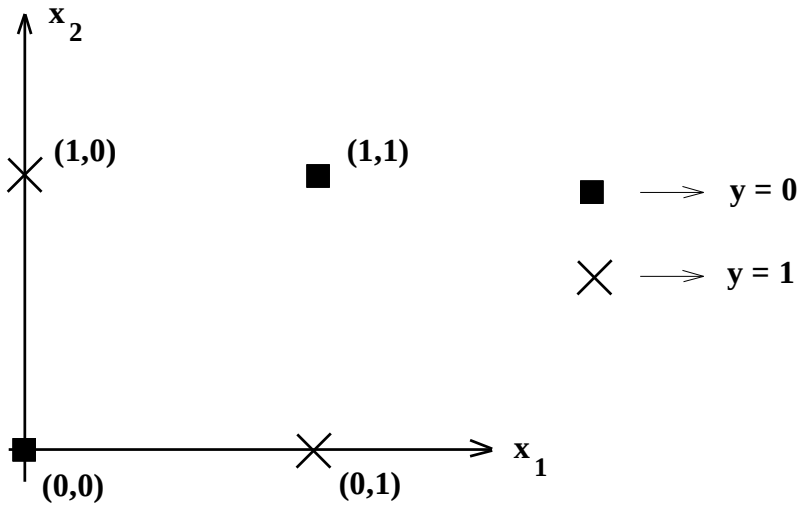



Figura 20 – O problema do OU-exclusivo

- Inicialmente será analisado o comportamento de um neurônio tipo perceptron (veja Figura 21) no processo de solução do problema exposto acima. A saída **y** pode ser representada na forma:

$$\mathbf{y} = \mathbf{g}(\mathbf{w_1 x_1} + \mathbf{w_2 x_2} + \mathbf{w_0}) \quad \text{onde} \quad \begin{cases} g(\mathbf{u}) = 1 \ \text{se} \ \mathbf{u} \geq 0 \\ g(\mathbf{u}) = 0 \ \text{se} \ \mathbf{u} < 0 \end{cases}$$

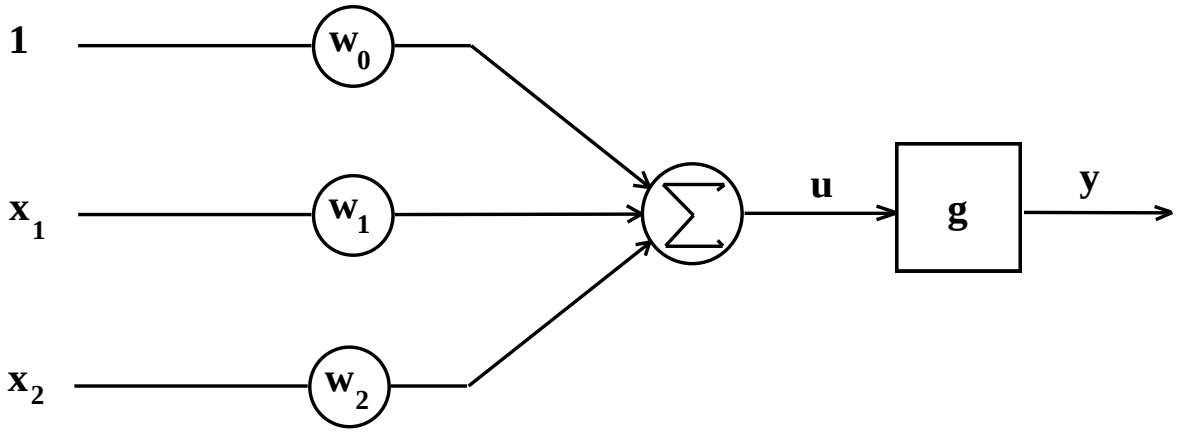



Figura 21 – Neurônio tipo perceptron, com duas entradas (mais a polarização)

- Para qualquer valor dos parâmetros $\mathbf{w_0}$ , $\mathbf{w_1}$ e $\mathbf{w_2}$, a função **g**(**u**) separa o espaço de entradas em duas regiões, sendo que a curva de separação é uma linha reta.

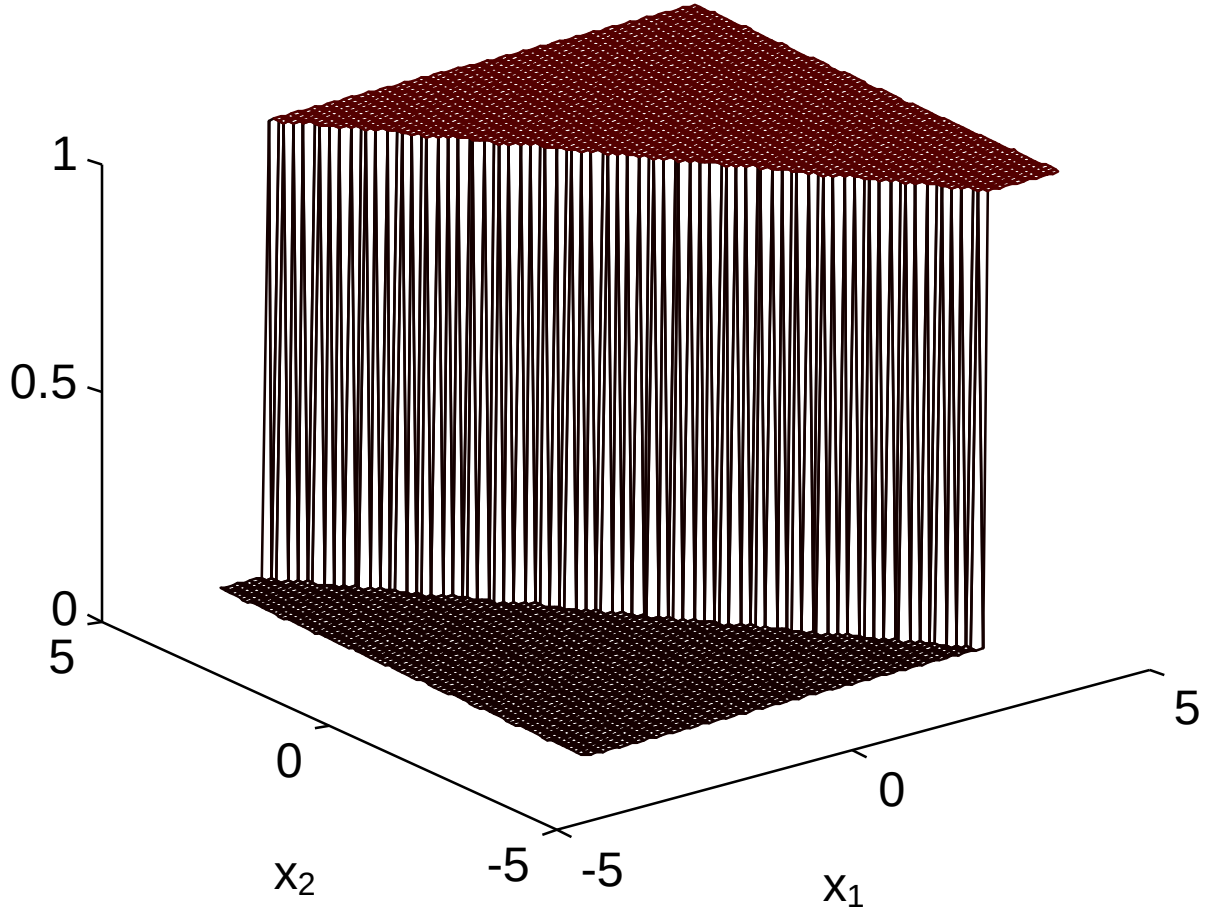



Figura 22 – Mapeamento de entrada-saída para o perceptron da Figura 21, com $\mathbf{w_0} = -6$, $\mathbf{w_1} = 4$ e $\mathbf{w_2} = 3$

- Aqui tomou-se a função g(·) como sendo a função sinal, pois as saídas são binárias.

- No problema do OU-exclusivo (Figura 20), pode-se constatar que não existe uma única linha reta divisória de forma que os pontos (0,0) e (1,1) se posicionem de um lado enquanto que (0,1) e (1,0) permaneçam do outro lado da linha.
- Logo, pode-se imediatamente concluir que um neurônio tipo perceptron não apresenta grau de liberdade suficiente para resolver o problema proposto, o que foi corretamente constatado por Minsky & Papert, em 1969.
- No entanto, esses autores também acreditavam que não havia razão para supor que redes multicamadas pudessem conduzir a uma solução para o problema proposto. Esta hipótese só foi definitivamente rejeitada com o desenvolvimento do algoritmo de retro-propagação (*back-propagation*), já nos anos 80, o qual permite o ajuste automático de pesos para redes neurais multicamadas, arquitetura necessária para a realização de mapeamentos não-lineares.
- Considere o problema de mapeamento de uma rede neural tipo perceptron, com uma camada intermediária (Figura 23), aplicada ao problema do OU-exclusivo.

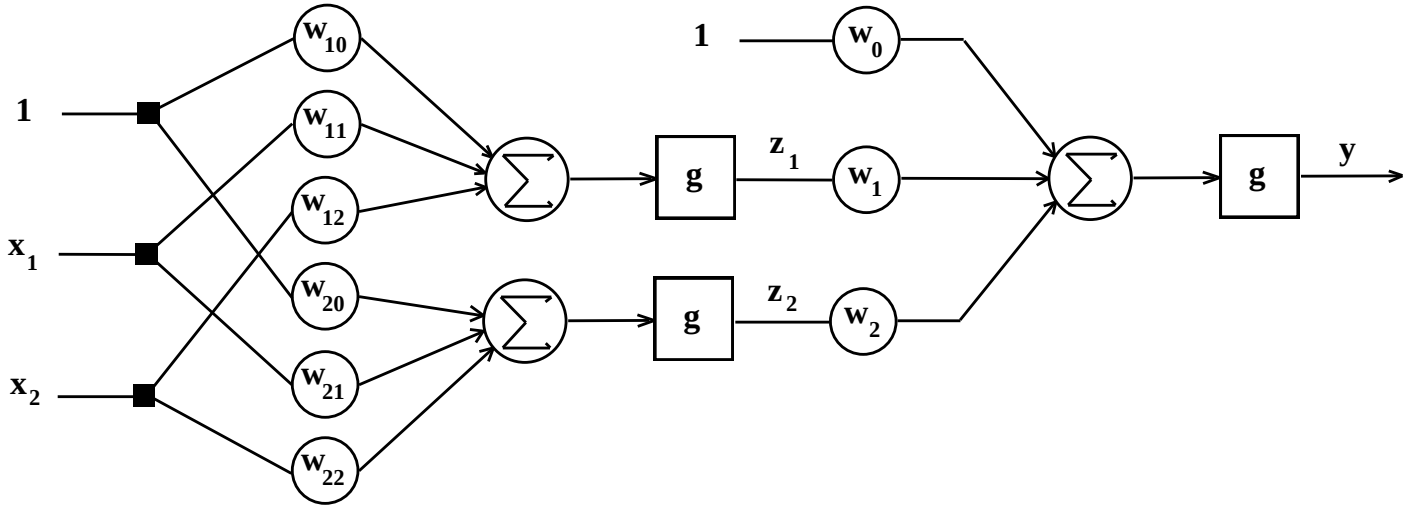



Figura 23 – Perceptron de três camadas (uma camada intermediária)

- A camada de entrada fornece um vetor de entrada ($\mathbf{x_1,x_2}$) para a camada intermediária, enquanto que a camada intermediária produz duas saídas $\mathbf{z_1}=\text{sgn}(\mathbf{w_{10}}+\mathbf{w_{11}x_1}+\mathbf{w_{12}x_2})$ e $\mathbf{z_2}=\text{sgn}(\mathbf{w_{20}}+\mathbf{w_{21}x_1}+\mathbf{w_{22}x_2})$. Na camada de saída, o sinal de saída da rede neural é dado por $\mathbf{y}=\text{sgn}(\mathbf{w_0}+\mathbf{w_1z_1}+\mathbf{w_2z_2})$.
- Surge uma questão: existem parâmetros $\mathbf{w_{ij}}$ (i=1,2; j=0,1,2) e $\mathbf{w_k}$ (k = 0,1,2) tais que $\mathbf{y}$ = 0 para as entradas (0,0) e (1,1) e $\mathbf{y}$ = 1 para as entradas (1,0) e (0,1)?

- As saídas da primeira camada ($\mathbf{z_1}$ e $\mathbf{z_2}$) podem ser consideradas como variáveis intermediárias utilizadas na geração da saída **y**. As variáveis $\mathbf{z_1}$ e $\mathbf{z_2}$ formam o que será denominado mais à frente no curso de espaço de características.
- Do que já foi visto a respeito de um neurônio tipo perceptron, sabe-se que existem pesos $\mathbf{w_{1j}}$ (j=0,1,2) tais que (veja curva de separação $\mathbf{L_1}$ na Figura 24(a)):

(0,1) produza $\mathbf{z_1} = 1$

(0,0),(1,0),(1,1) produza $\mathbf{z_1} = 0$.

- De forma similar, existem pesos $\mathbf{w_{2j}}$ (j=0,1,2) tais que (veja curva de separação $\mathbf{L_2}$ na Figura 24(a)):

(0,1),(0,0),(1,1) produza $\mathbf{z_2} = 1$

(1,0) produza $\mathbf{z_2} = 0$

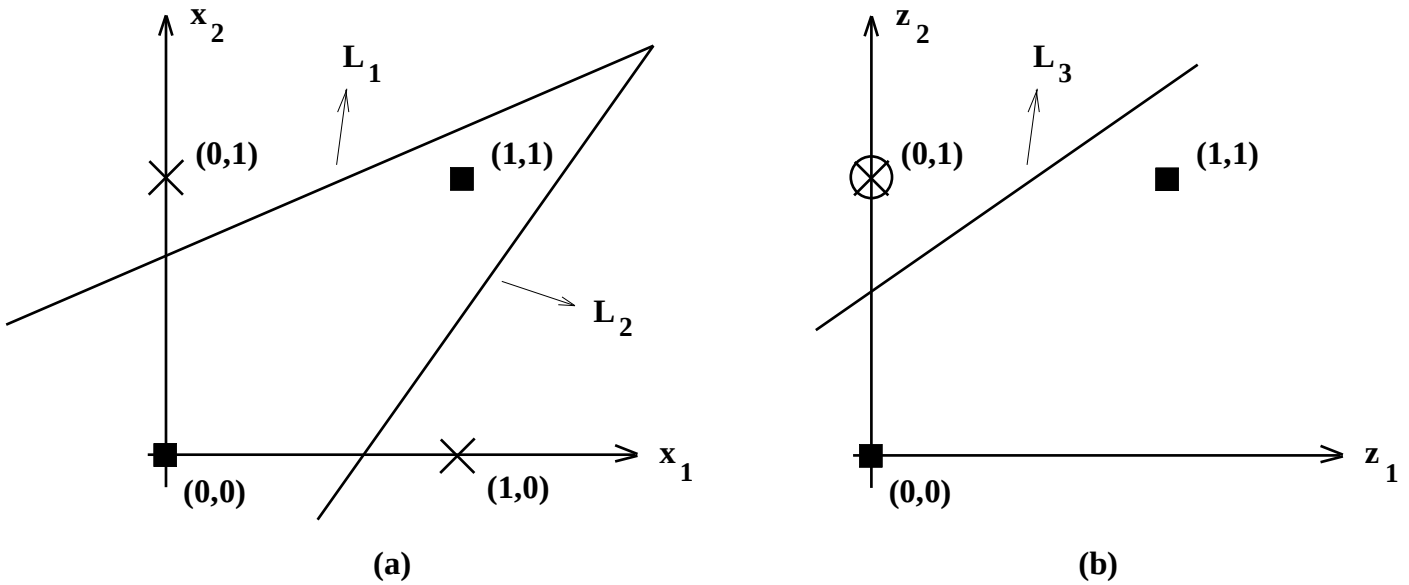



Figura 24 – Realização da função OU-exclusivo

- A discussão acima mostra que existem pesos $\mathbf{w_{ij}}$ (i=1,2; j=0,1,2) de maneira que a entrada (0,1) resulte em $\mathbf{z_1} = 1$, $\mathbf{z_2} = 1$, e a entrada (1,0) resulte em $\mathbf{z_1} = 0$, $\mathbf{z_2} = 0$, enquanto que (0,0) e (1,1) produzam $\mathbf{z_1} = 0$, $\mathbf{z_2} = 1$. Já que (0,0) e (1,1) podem ser separados linearmente de (0,1), como mostrado na Figura 24(b) pela curva de separação $\mathbf{L_3}$, pode-se concluir que a função booleana desejada pode ser obtida utilizando-se perceptrons em cascata, ou seja, 3 neurônios do tipo perceptron.

# 11. Otimização não-linear e capacidade de generalização

- Diversos tipos de parâmetros da rede neural poderiam ser submetidos a processos de ajuste durante o treinamento, como (*i*) pesos sinápticos; (*ii*) parâmetros da função de ativação de cada neurônio; (*iii*) número de neurônios na camada intermediária; (*iv*) número de camadas intermediárias.
- Iremos nos restringir aqui ao ajuste dos pesos sinápticos. Neste caso, o processo de treinamento supervisionado de redes neurais artificiais multicamadas é equivalente a um problema de otimização não-linear irrestrita, em que a superfície de erro reside em um espaço contínuo, que aponta o erro quadrático médio para cada vetor de pesos no $\Re^P$, e é minimizada a partir do ajuste dos pesos sinápticos.
- Iremos nos restringir também a redes MLP com uma única camada intermediária, visto que com apenas uma camada intermediária a rede neural já apresenta **capacidade de aproximação universal** (CYBENKO, 1989; HORNIK *et al.*, 1989; HORNIK *et al.*, 1990; HORNIK *et al.*, 1994).

- Um problema comum a todos os modelos de aproximação de funções que possuem capacidade de aproximação universal, não apenas redes neurais artificiais do tipo MLP, é a necessidade de controlar adequadamente o seu grau de flexibilidade.
- Como o conjunto de amostras disponível para treinamento supervisionado é finito, infinitos mapeamentos podem produzir o mesmo desempenho de aproximação, independente do critério de desempenho adotado. Esses mapeamentos alternativos vão diferir justamente onde não há amostras disponíveis para diferenciá-los.
- Visando maximizar a **capacidade de generalização** do modelo de aproximação (no caso, uma rede neural MLP), ou seja, buscando encontrar o grau de flexibilidade adequado para o modelo de aproximação (dada a demanda da aplicação), um procedimento recomendado é dividir o conjunto de amostras disponível para treinamento em dois: um conjunto que será efetivamente empregado no ajuste dos pesos (conjunto de treinamento) e um conjunto que será

empregado para definir o momento de interromper o treinamento (conjunto de validação).

- Deve-se assegurar que ambos os conjuntos sejam suficientemente representativos do mapeamento que se pretende aproximar. Assim, minimizar o erro junto ao conjunto de validação implica em maximizar a capacidade de generalização. Logo, espera-se que a rede neural que minimiza o erro junto ao conjunto de validação (não usado para o ajuste dos pesos) tenha o melhor desempenho possível junto a novas amostras.
- A figura alto/esquerda a seguir mostra um mapeamento unidimensional a ser aproximado (desconhecido pela rede neural) e amostras sujeitas a ruído de média zero (única informação disponível para o treinamento da rede neural). A figura alto/direita mostra o resultado da aproximação produzida por uma rede neural com muito poucos neurônios, a qual foi incapaz de realizar a aproximação (tem baixa flexibilidade).

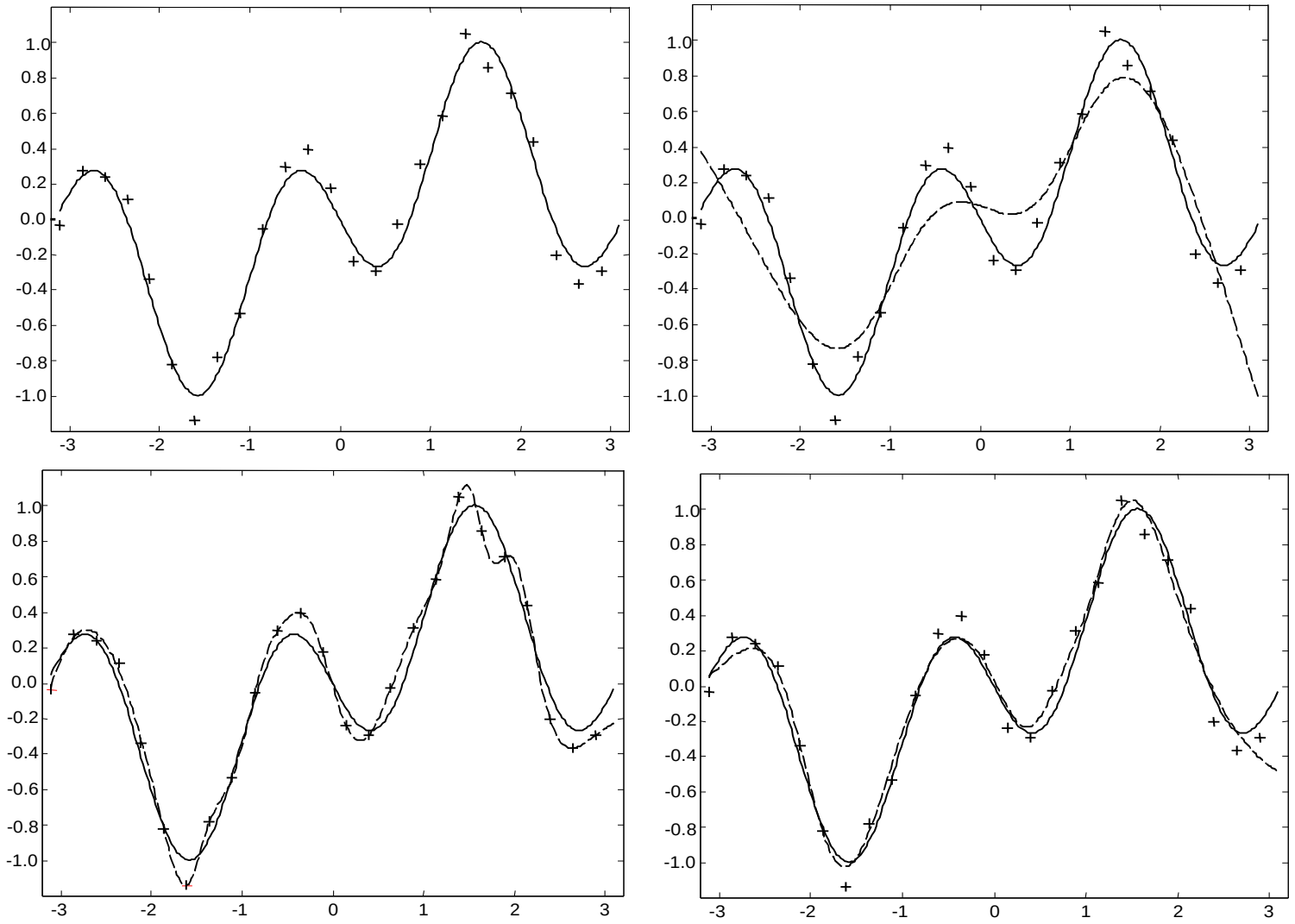



Figura 25 – Dados amostrados, função a ser aproximada e modelos de aproximação com diferentes capacidades de generalização

- Já as figuras baixo/esquerda e baixo/direita mostram o resultado de uma mesma rede neural (com número suficiente de neurônios), mas à esquerda ocorreu sobre-treinamento, enquanto que à direita o treinamento foi interrompido quando minimizou-se o erro junto a dados de validação (não apresentados).
- Curvas típicas de erro de treinamento e validação são apresentadas a seguir.

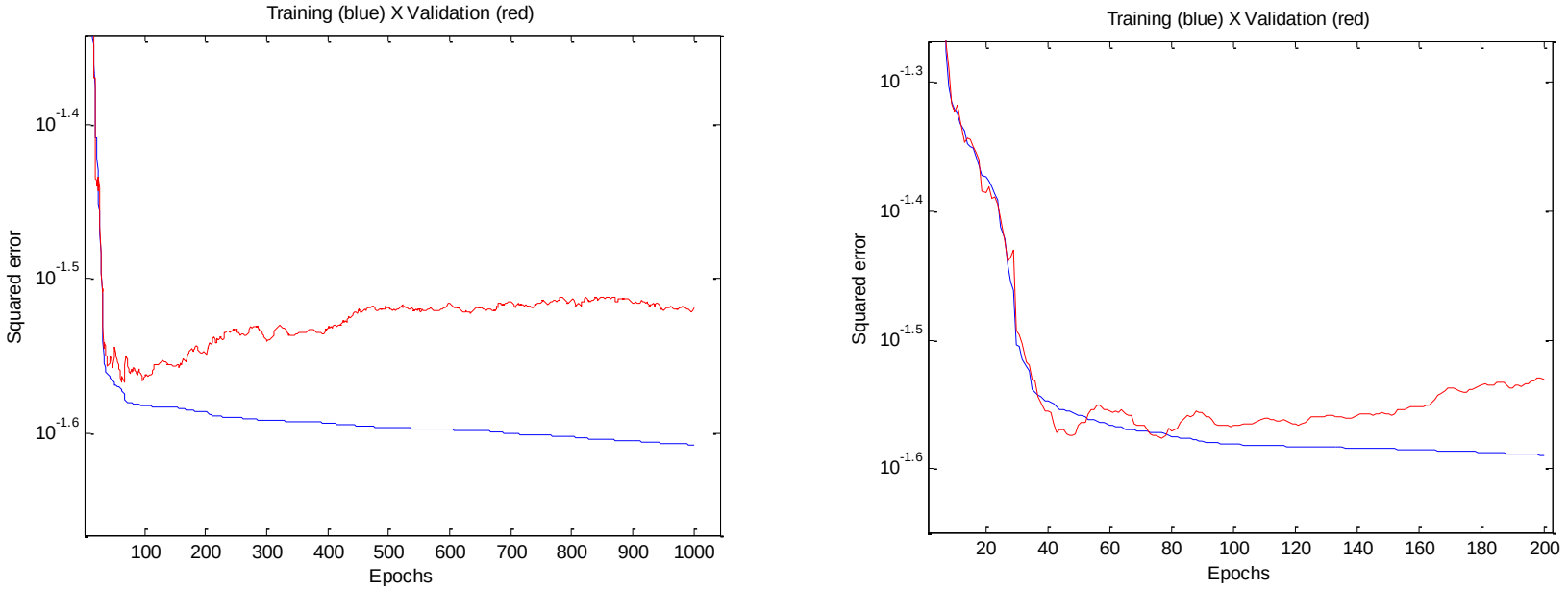



Figura 26 – Ilustrações típicas da evolução, ao longo das épocas de treinamento, dos erros de treinamento e de validação em treinamento supervisionado de redes MLP.

- No entanto, nem sempre o erro de validação apresenta este comportamento, e cada caso deve ser analisado isoladamente. Como a curva do erro de validação oscila bastante e esboça um comportamento pouco previsível, não é indicado desenvolver detectores automáticos de mínimos e encerrar o treinamento ali. O mais indicado é sobre-treinar a rede e armazenar os pesos associados ao mínimo do erro de validação.
- Não existe um consenso sobre como fazer o melhor particionamento do conjunto de dados, ou seja, sobre como dividi-lo de forma que possamos encontrar uma rede com a melhor capacidade de generalização em todos os casos. Uma sugestão de partida pode ser 80% das amostras para treinamento e 20% para validação.
- Quando as amostras correspondem a dados rotulados em problemas de classificação de padrões, procure respeitar a distribuição junto a cada classe.
- Este mecanismo de regularização do treinamento de redes neurais é denominado *holdout*, por manter de fora do treinamento um subconjunto das amostras.

## 11.1 Validação cruzada com *k* pastas

- A técnica *holdout* que acabou de ser descrita tem uma desvantagem importante: alguns dados disponíveis sempre serão usados no ajuste dos pesos e outros nunca serão usados para este fim, pois compõem o conjunto de validação.
- Uma técnica mais elaborada é dividir o conjunto de amostras disponíveis para treinamento em $k$ pastas e realizar $k$ treinamentos, cada um considerando $k$–1 pastas para o ajuste de pesos (equivale ao conjunto de treinamento da seção anterior) e 1 pasta para a validação (equivale ao conjunto de validação da seção anterior).
- Com este procedimento, toda amostra disponível vai aparecer $k$–1 vezes no conjunto de treinamento e 1 vez no conjunto de validação.
- Repare que os $k$ conjuntos de treinamento terão composição distinta, assim como os $k$ conjuntos de validação. O desempenho da rede neural é geralmente tomado como a média do desempenho das $k$ redes neurais obtidas.

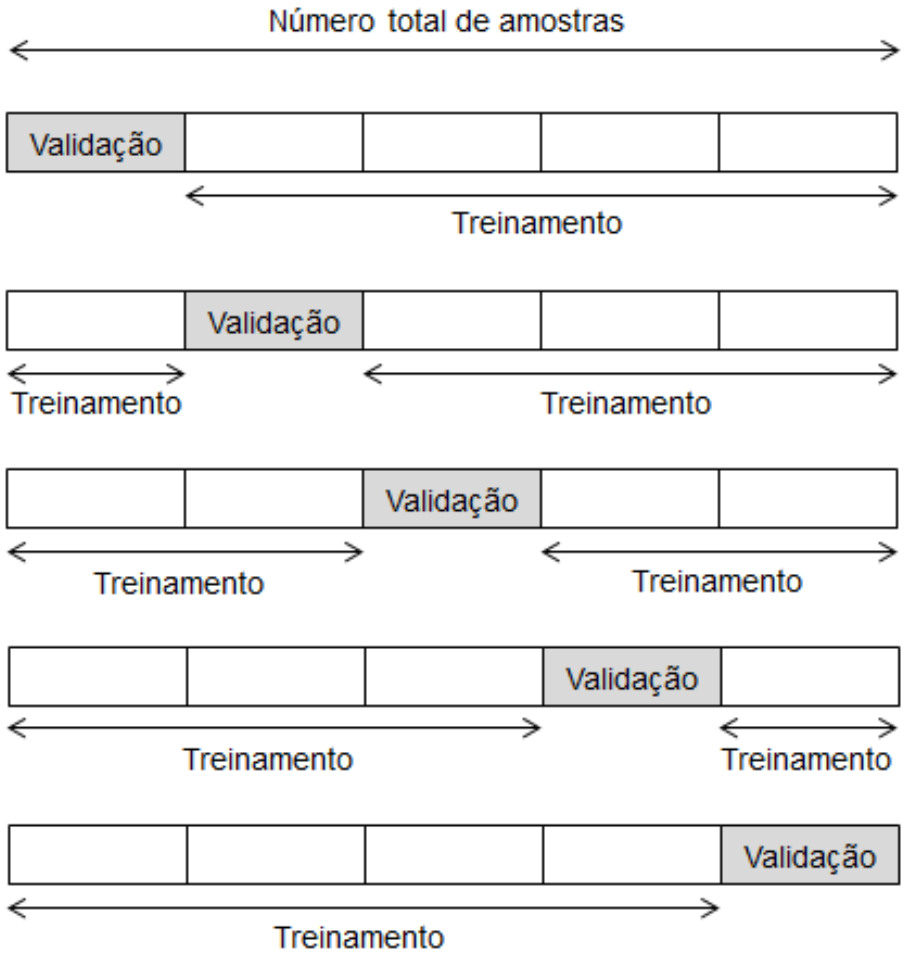



Figura 27 – Esquema de divisão das amostras disponíveis para treinamento na abordagem de validação cruzada com *k*-pastas (*k-fold crossvalidation*), com *k*=5.

## 11.2 Rede neural MLP como um modelo instável

- Como todos os modelos que apresentam capacidade de aproximação universal, a rede neural MLP é considerada um modelo instável, no sentido de que a proposta de mapeamento de entrada-saída pode sofrer mudanças não-esperadas com a simples inclusão de uma amostra adicional junto ao conjunto de dados de treinamento.
- Essa instabilidade simplesmente é o reflexo do grau de flexibilidade e do poder de aproximação e não necessariamente se apresenta como uma desvantagem em aplicações práticas. De fato, esta propriedade será fundamental para o sucesso da abordagem de comitê de máquinas denominada *ensemble*, a ser abordada mais adiante no curso.
- As técnicas de regularização *holdout* e *k-fold crossvalidation* podem ser interpretadas, então, como uma forma de "robustecer" o resultado do processo de aproximação do mapeamento de entrada-saída a partir de modelos instáveis.

## 11.3 Gradiente, hessiana e algoritmos de otimização

- Considere uma função contínua e diferenciável até 2a. ordem em todos os pontos do domínio de interesse, tal que: $f$: $\Re^n \rightarrow \Re$ e $\mathbf{x} \in \Re^n$
- Expansão em série de Taylor em torno do ponto $\mathbf{x}^* \in \Re^n$:

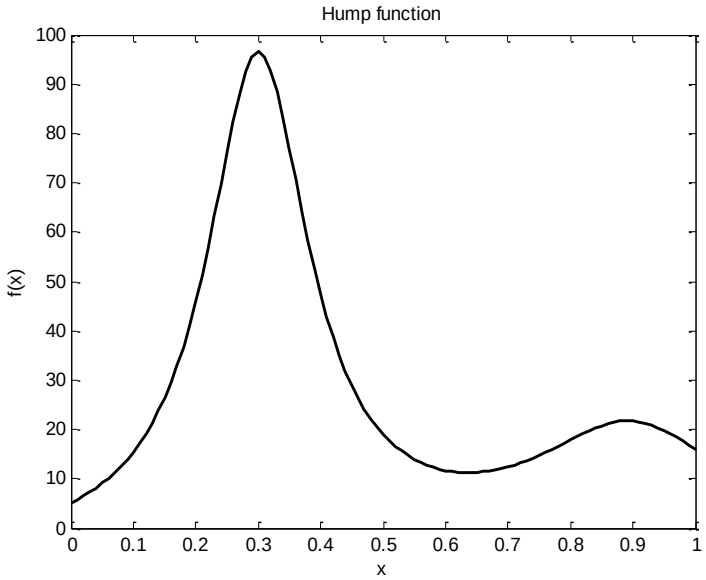
$$f(\mathbf{x}) = f(\mathbf{x}^*) + \nabla f(\mathbf{x}^*)^T (\mathbf{x} - \mathbf{x}^*) + \frac{1}{2}(\mathbf{x} - \mathbf{x}^*)^T \nabla^2 f(\mathbf{x}^*)(\mathbf{x} - \mathbf{x}^*) + O(3)$$

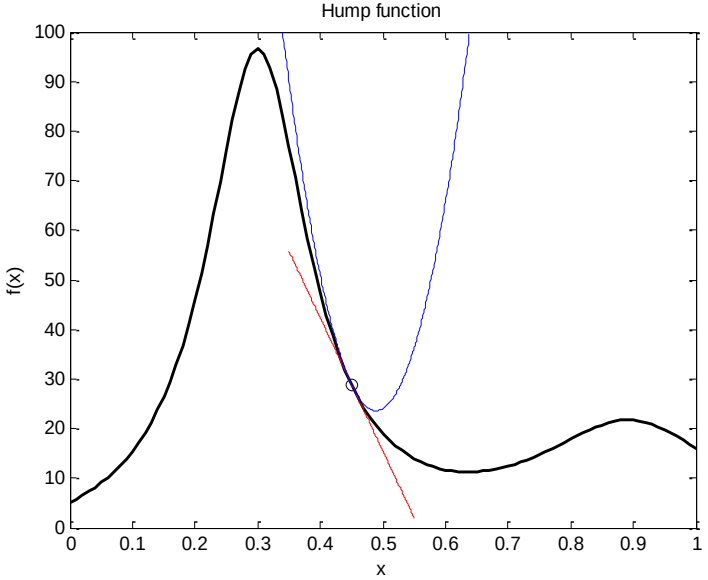
$$\nabla f(x^*) = \begin{bmatrix} \frac{\partial f(\mathbf{x}^*)}{\partial x_1} \\ \frac{\partial f(\mathbf{x}^*)}{\partial x_2} \\ \vdots \\ \frac{\partial f(\mathbf{x}^*)}{\partial x_n} \end{bmatrix} \qquad \nabla^2 f(\mathbf{x}^*) = \begin{bmatrix} \frac{\partial^2 f(\mathbf{x}^*)}{\partial x_1^2} & \frac{\partial^2 f(\mathbf{x}^*)}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f(\mathbf{x}^*)}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f(\mathbf{x}^*)}{\partial x_2 \partial x_1} & \frac{\partial^2 f(\mathbf{x}^*)}{\partial x_2^2} & & \vdots \\ & & \ddots & \\ \frac{\partial^2 f(\mathbf{x}^*)}{\partial x_n \partial x_1} & \cdots & & \frac{\partial^2 f(\mathbf{x}^*)}{\partial x_n^2} \end{bmatrix}$$

**Vetor gradiente** **Matriz hessiana**

- O algoritmo de retropopagação do erro (do inglês *backpropagation*) é empregado para obter o vetor gradiente, onde cada elemento do vetor gradiente está associado a um peso da rede neural e indica o quanto a saída é influenciada por uma variação incremental neste peso sináptico.

- Função $y = f(x) = \frac{1}{(x-0.3)^2 + 0.01} + \frac{1}{(x-0.9)^2 + 0.04} - 6$, conhecida como *hump function*, e suas aproximações de 1a. e 2a. ordem no ponto $x = 0.45$.



- Existem várias técnicas para obter exatamente ou aproximadamente a informação de 2a. ordem junto à superfície de erro produzida por redes neurais MLP (BATTITI, 1992; BISHOP, 1992).

- O processo de otimização não-linear envolvido no ajuste de pesos de uma rede neural vai realizar aproximações locais de primeira ordem ou de primeira e segunda ordem junto à superfície de erro e realizar ajustes incrementais e recursivos na forma:

$$\theta_{k+1} = \theta_k + \text{passo}_k * \text{direção}_k$$

- Parte-se de uma condição inicial $\theta_0$ e aplica-se iterativamente a fórmula acima, sendo que a direção depende da informação local de primeira e segunda ordem. Cada proposta de algoritmo de otimização vai diferir na forma de computar o *passo* e a *direção* de ajuste, a cada iteração *k*.
- A figura a seguir apresenta uma classificação dos principais algoritmos empregados para o treinamento supervisionado de redes neurais artificiais.

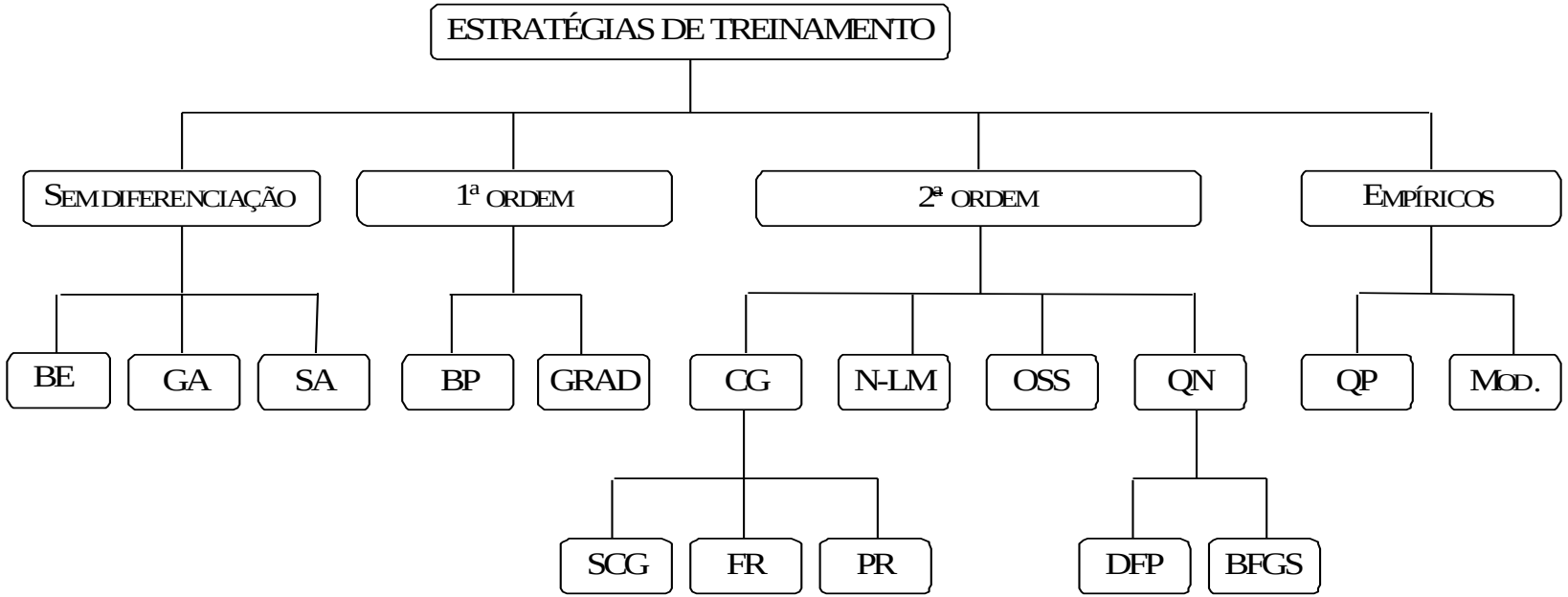



Figura 28 – Taxonomia de algoritmos de otimização para treinamento supervisionado de redes neurais MLP.

- Merece atenção o algoritmo do gradiente conjungado escalonado (do inglês *Scaled Conjugate Gradient* – SCG). Uma vantagem deste algoritmo é que ele apresenta um custo computacional (memória e processamento por iteração) linear com o número de pesos e não quadrático, como a maioria dos algoritmos de 2a. ordem.

## 11.4 Mínimos locais

- Como o processo de ajuste é iterativo e baseado apenas em informações locais, os algoritmos de otimização geralmente convergem para o mínimo local mais próximo de onde se encontra a busca, que pode representar uma solução inadequada (com nível de erro acima do aceitável).

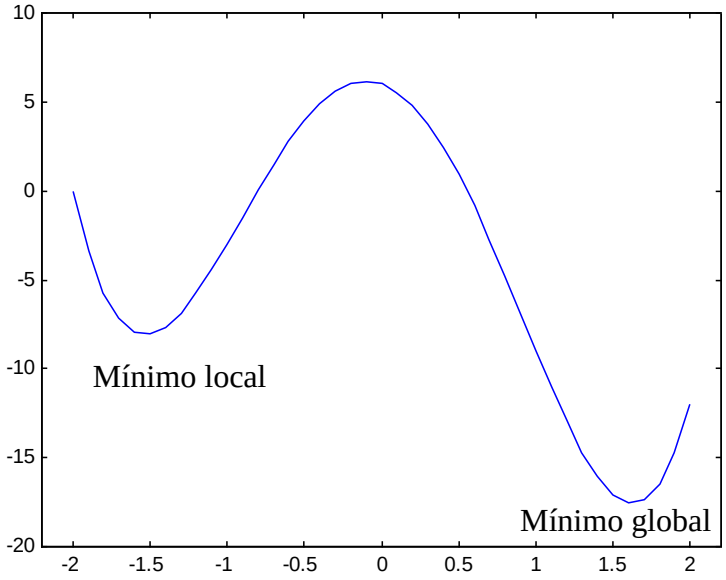



Figura 29 – Exemplo ilustrativo de mínimos local e global (considerando uma única variável).

- Note que algoritmos de 2a. ordem tendem a convergir mais rápido para os mínimos locais, mas não se pode afirmar que eles convergem para mínimos de melhor qualidade que aqueles produzidos pelos algoritmos de primeira ordem.
- Um conceito relevante aqui é o de **bacia de atração**. Todo mínimo local tem uma bacia de atração e o mínimo local a ser fornecido pelo algoritmo de otimização tende a ser aquele em cuja bacia de atração encontra-se a condição inicial (ponto de partida da busca iterativa).
- Isso é bem provável no caso de buscas iterativas que dão passos sempre minimizantes, mas podem ocorrer passos capazes de deslocar a busca para bacias de atração vizinhas, associadas a outros mínimos locais.
- A trajetória da condição inicial até o ótimo local resultante será certamente distinta para algoritmos de 1a. e 2a. ordem, pois eles diferem a cada iteração da busca, mas o resultado final tende a ser o mesmo, a menos que haja deslocamentos casuais para outras bacias de atração vizinhas.

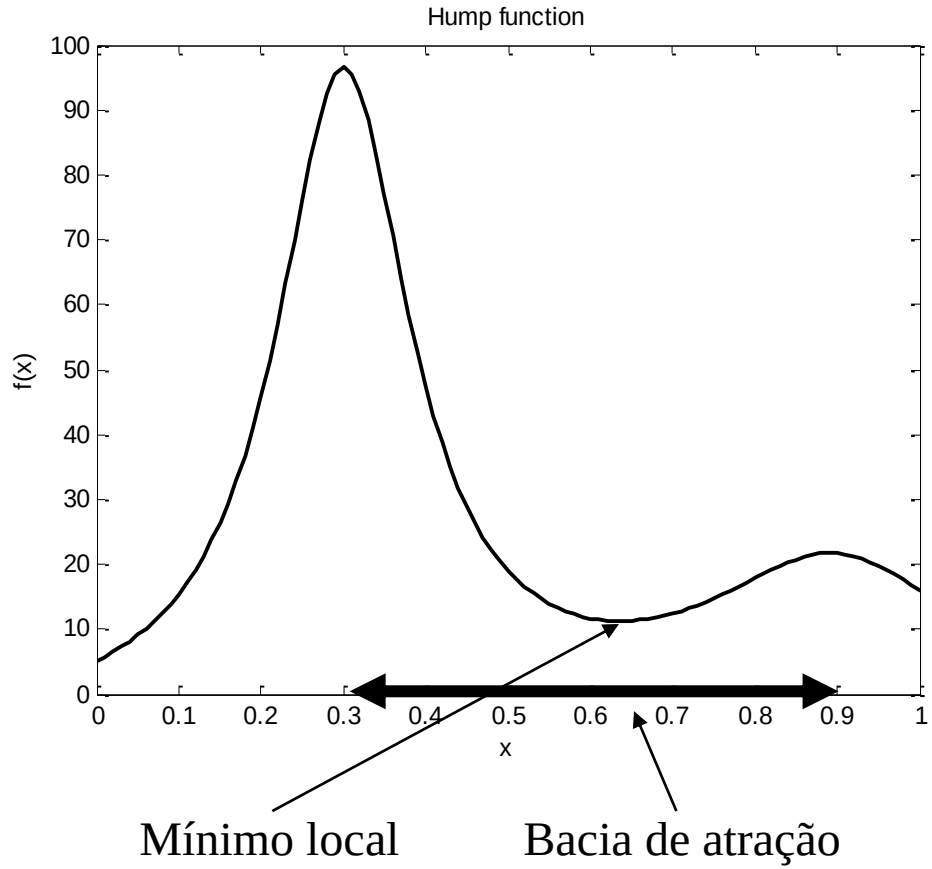



Figura 30 – Exemplo de bacia de atração de um mínimo local

## 11.5 Condição inicial para os pesos da rede neural

- Embora existam técnicas mais elaboradas, os pesos da rede neural podem ser inicializados com valores pequenos e aleatoriamente distribuídos em torno de zero.
- Esta inicialização promove as seguintes propriedades da rede neural inicial:
  - ✓ O mapeamento realizado pela MLP tende a se aproximar de um hiperplano, não apresentando, assim, nenhuma tendência definida, em termos de comportamento não-linear;
  - ✓ A ativação de todos os neurônios vai se encontrar fora da região de saturação, facilitando o processo de ajuste de pesos, a ser iniciado.
- Em termos práticos, pode-se afirmar que, com este procedimento de inicialização, o mapeamento produzido pela rede neural começa sem nenhuma contorção expressiva, mas com máxima capacidade de se contorcer de acordo com a demanda da aplicação.

## 11.6 Critério de parada

- O critério de parada mais empregado é o número máximo de épocas de treinamento, onde uma época é dada pela apresentação de todas as amostras do conjunto de treinamento à rede neural.
- Mas existem outros critérios que podem ser considerados conjuntamente ou em substituição ao número de épocas:
    - ✓ Módulo do vetor gradiente abaixo de um certo limiar;
    - ✓ Progresso do erro de treinamento abaixo de um certo limiar;
    - ✓ Grau de ajuste dos pesos sinápticos abaixo de um certo limiar.
- Esses limiares mencionados nos três itens acima devem ser definidos pelo usuário, havendo a necessidade de realização de alguns testes junto a cada aplicação pretendida.
- No toolbox fornecido pelo professor, empregam-se o número máximo de épocas conjuntamente com o módulo do vetor gradiente.

# 12. Processo Iterativo para MLP – Método Padrão-a-Padrão

Obs1: Está sendo considerada a aplicação de um método de 1a. ordem para ajuste dos pesos.

Obs2: Este método e variações dele são conhecidas como *stochastic gradient descent* (DUCHI et al. 2011).

- Defina uma condição inicial para o vetor de pesos **w** e um passo $\alpha$ pequeno;
- Faça $k = 0$ e calcule $J(\mathbf{w}(k))$;
- Enquanto o critério de parada não for atendido, faça:
    - ♦ Ordene aleatoriamente os padrões de entrada-saída;
    - ♦ Para $l$ variando de 1 até $N$, faça:

        Apresente o padrão $l$ de entrada à rede;

        Calcule $J_l(\mathbf{w}(k))$ e $\nabla J_l(\mathbf{w}(k))$;

        $\mathbf{w}(k+1) = \mathbf{w}(k) - \alpha \nabla J_l(\mathbf{w}(k))$;
    - ♦ $k = k + 1$;
    - ♦ Calcule $J(\mathbf{w}(k))$;

# 13. Processo Iterativo para MLP – Método em Lote ou Batelada

Obs: Está sendo considerada a aplicação de um método de 1a. ordem para ajuste dos pesos.

- Defina uma condição inicial para o vetor de pesos **w** e um passo $\alpha$ pequeno;
- Faça $k = 0$ e calcule $J(\mathbf{w}(k))$;
- Enquanto o critério de parada não for atendido, faça:
  - ♦ Para $l$ variando de 1 até $N$, faça:

    Apresente o padrão $l$ de entrada à rede;

    Calcule $J_l(\mathbf{w}(k))$ e $\nabla J_l(\mathbf{w}(k))$;
  - ♦ $\mathbf{w}(k+1) = \mathbf{w}(k) - \frac{\alpha}{N}\sum_{l=1}^{N}\nabla J_l(\mathbf{w}(k))$;
  - ♦ $k = k + 1$;
  - ♦ Calcule $J(\mathbf{w}(k))$;

# 14. Outros métodos de otimização não-linear empregados no treinamento de RNAs

- Os desafios associados ao treinamento de redes neurais profundas (*deep learning*) levaram à proposição de algoritmos supostamente mais competentes para o ajuste de pesos, no sentido de promoverem, em média, progressos mais expressivos na exploração da superfície de erro para a mesma carga de recursos computacionais empregada durante o treinamento da rede neural (RUDER, 2017).
- Alguns desses avanços são truques computacionais de natureza empírica, como os algoritmos adaptativos que não recorrem diretamente a informações de 2a. ordem, enquanto outros são melhor suportados por fundamentos conceituais, como a técnica de *dropout* (a ser abordada mais adiante neste curso) usada para conter o sobreajuste (*overfitting*) no processo de treinamento.
- São apresentados, a seguir, apenas noções básicas de parte desses progressos recentes em otimização não-linear voltada para o ajuste de pesos de RNAs.

- Evidentemente, existem iniciativas voltadas para outras estratégias de aprendizado de máquina, além do treinamento de RNAs (BOTTOU *et al*., 2018).

## *14.1 Mini-batch gradient descent*

- Trata-se de uma técnica intermediária entre os métodos padrão-a-padrão (*stochastic gradient descent* – SGD) e batelada (*batch*). Mais detalhes podem ser obtidos em: [https://machinelearningmastery.com/gentle-introduction-mini-batch-gradient-descent-configure-batch-size/].
- Definem-se partições ou lotes (subconjuntos) de dados de treinamento, sendo que o treinamento para cada lote é em batelada. A evolução do erro não oscila tanto quanto no caso padrão-a-padrão, mas possivelmente oscila o suficiente para escapar de mínimos locais ruins (LI *et al.*, 2014).
- Além disso, há um favorecimento das implementações em computação paralela e uma redução no uso de memória, já que não é necessário manter simultaneamente todos os dados de treinamento em memória.

## 14.2 Algoritmos adaptativos

- Os algoritmos adaptativos foram motivados basicamente pela limitação associada ao uso de uma mesma taxa de aprendizado para todos os padrões de entrada ou então pela definição prévia de uma política de adaptação dessa taxa de aprendizado. Em ambos os casos, é desafiador definir adequadamente e a priori essa única taxa de aprendizado ou então essa política de ajuste da taxa.
- Deve-se levar em conta que uma taxa de aprendizado muito pequena produz uma convergência muito lenta do treinamento, enquanto que uma taxa de aprendizado muito elevada pode impedir a convergência do processo de ajuste, seja por apresentar oscilações em torno de um ótimo local, seja por divergir.
- Além disso, usar uma mesma taxa de aprendizado para todos os pesos ajustáveis, seja ela fixa ou adaptativa, pode não ser a melhor solução em aplicações práticas caracterizadas, por exemplo, pela existência de dados esparsos, em que o ajuste dos pesos se dá em frequências distintas. Neste caso, pode não ser uma boa

estratégia ajustar os pesos com a mesma intensidade a cada iteração. Tende a ser mais produtivo, assim, promover ajustes mais intensos para pesos que sofrem ajustes mais raros ou menos intensos num histórico de ajustes recentes.

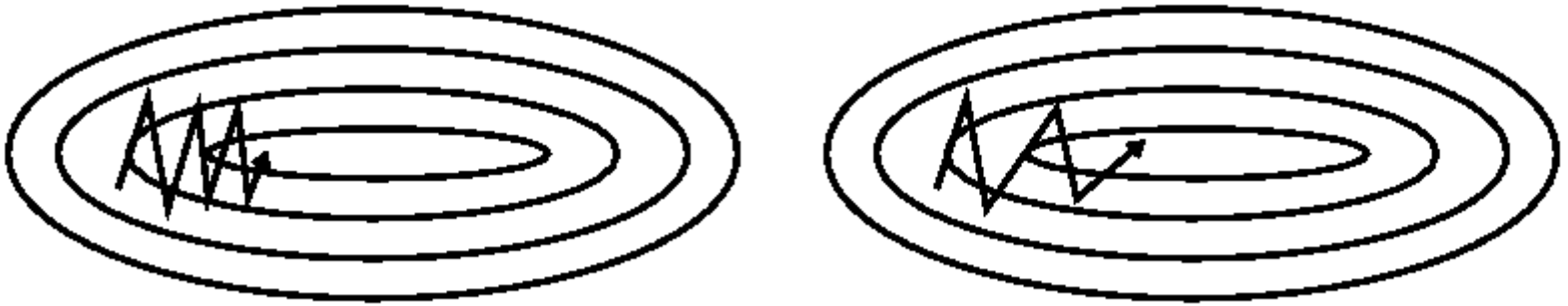
- Outro aspecto-chave em *deep learning* é a existência de uma quantidade muito grande de mínimos locais de baixa qualidade, de pontos de sela e de plateaus. Escapar dessas "armadilhas" geralmente requer passos de treinamento adaptativos.

## 14.3 Algoritmo adaptativo 1: SGD + momentum

- Sua principal missão é reduzir oscilações durante a exploração da superfície de erro, mas também opera como acelerador sempre que gradientes subsequentes preservarem direções de busca (QIAN, 1999).

$$\mathbf{w}(k+1) = \mathbf{w}(k) - \alpha \nabla J_l\left(\mathbf{w}(k)\right) + \gamma\left[\mathbf{w}(k) - \mathbf{w}(k-1)\right]$$

- Um valor típico geralmente adotado para $\gamma$ é 0,9. De qualquer modo, deve-se considerar o fato de que o processo de ajuste ganhou uma ordem na dinâmica e um hiperparâmetro a mais, a ser devidamente definido.



Sem termo de momentum Com termo de momentum

Figura 31 – Contribuição típica do termo de momentum na redução de oscilações. Fonte: [https://towardsdatascience.com/stochastic-gradient-descent-with-momentum-a84097641a5d]

## 14.4 Algoritmo adaptativo 2: *Nesterov accelerated gradient* (NAG)

- Numa tentativa de acelerar a busca, é possível calcular o vetor gradiente não em $\mathbf{w}(k)$, mas em $\mathbf{w}(k)+\gamma\left[\mathbf{w}(k)-\mathbf{w}(k-1)\right]$, que é uma aproximação do novo valor do vetor de pesos da rede neural. Com isso, a regra de ajuste assume a forma:

$$\mathbf{w}(k+1)=\mathbf{w}(k)-\alpha\nabla J_l\left(\mathbf{w}(k)+\gamma\left[\mathbf{w}(k)-\mathbf{w}(k-1)\right]\right)+\gamma\left[\mathbf{w}(k)-\mathbf{w}(k-1)\right]$$

- Trata-se de uma tentativa de antecipar o comportamento da superfície de erro.

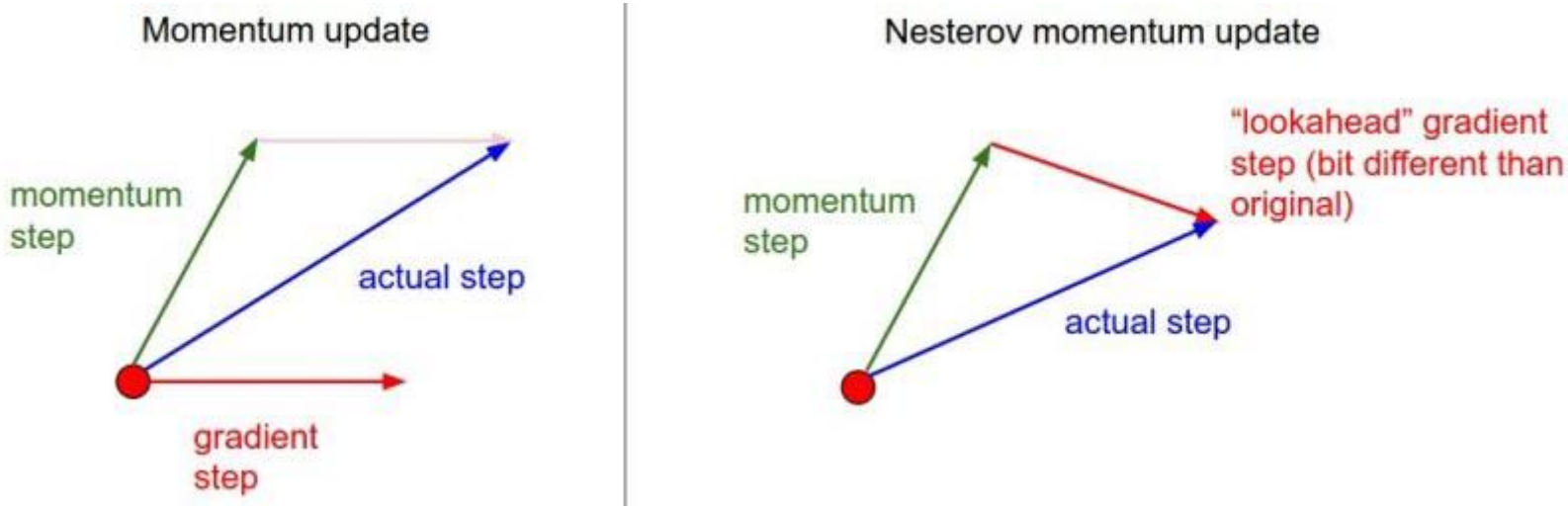



Figura 32 – Motivação geométrica da diferença entre SGD+momentum e NAG. Fonte: [https://towardsdatascience.com/stochastic-gradient-descent-with-momentum-a84097641a5d]

## 14.5 Estado-da-arte em algoritmos adaptativos

- Antes de listar as propostas mais avançadas em algoritmos adaptativos, é importante salientar que há dois aspectos a serem considerados ao medir a competência de um algoritmo de otimização não-linear no treinamento de redes neurais artificiais: (1) eficiência computacional; e (2) capacidade de produzir redes neurais que generalizam bem.

- De fato, os algoritmos adaptativos de uso mais frequente na literatura tendem a produzir ganhos de eficiência computacional, mas podem levar a redes neurais que generalizam pior que aquelas produzidas por *mini-batch* SGD (*stochastic gradient descent*), conforme investigado de forma empírica e também com evidências teóricas no trabalho de WILSON *et al*. (2018). A principal conclusão de WILSON *et al*. (2018) é que métodos adaptativos e não-adaptativos tendem a obter soluções de otimização diferentes e apresentar capacidades de generalização bem distintas.
- Como as propostas de algoritmos adaptativos são bem recentes, assim como essas críticas ao seu comportamento em aplicações práticas, associadas ao treinamento de redes neurais artificiais profundas ou não, cabem estudos mais detalhados visando chegar a regras de uso mais objetivas e efetivas. Havendo recursos computacionais disponíveis, por hora, a recomendação é treinar a rede neural com *mini-batch SCG* e também com ADAM, comparando o desempenho das redes neurais produzidas em cada caso, em termos de capacidade de generalização.

- O trabalho de RUDER (2017) apresenta uma formalização didática acerca das principais propostas em algoritmos adaptativos. Aqui, iremos apenas lista-las e citar referências relevantes em cada caso:
  - ✓ Adagrad (DUCHI *et al.*, 2011)
  - ✓ Adadelta (ZEILER, 2012)
  - ✓ ADAM (*Adaptive Moment Estimation*) (KINGMA & BA, 2015)
- O ADAM, em particular, calcula as taxas individuais de aprendizagem adaptativa para diferentes pesos da RNA a partir das estimativas de primeiro e segundo momentos dos gradientes consecutivos.
- Embora métodos de segunda ordem tradicionais, como o método de Newton, não sejam factíveis quando o número de parâmetros ajustáveis é elevado, existem iniciativas que viabilizam o emprego da informação de segunda ordem sem empregar a matriz hessiana, sendo técnicas bastante promissoras em *deep learning* (MARTENS, 2010), mas ainda pouco exploradas.

# 15. Referências

BATTITI, R. First- and Second-Order Methods for Learning: Between Steepest Descent and Newton's Method. *Neural Computation*, vol. 4, no. 2, pp. 141-166, 1992.

BISHOP, C. Exact Calculation of the Hessian Matrix for the Multilayer Perceptron. *Neural Computation*, vol. 4, no. 4, pp. 494-501, 1992.

BOTTOU, L., CURTIS, F.E., NOCEDAL, J. Optimization Methods for Large-Scale Machine Learning. arXiv:1606.04838v3, 2018.

CYBENKO, G. Approximation by superposition of sigmoidal functions. *Mathematics of Control, Signals and Systems*, vol. 2, no. 4, pp. 303-314, 1989.

DUCHI, J., HAZAN, E., SINGER, Y. Adaptive subgradient methods for online learning and stochastic optimization, *Journal of Machine Learning Research*, vol. 12, pp. 2121–2159, 2011.

HAYKIN, S. "Neural Networks: A Comprehensive Foundation", 2nd edition, Prentice-Hall, 1999.

HAYKIN, S. "Neural Networks and Learning Machines", 3rd edition, Prentice-Hall, 2008.

HEBB, D. O. "The Organization of Behavior", Wiley, 1949.

HORNIK, K., STINCHCOMBE, M., WHITE, H. Multi-layer feedforward networks are universal approximators. *Neural Networks*, vol. 2, no. 5, pp. 359-366, 1989.

HORNIK, K., STINCHCOMBE, M., WHITE, H. Universal approximation of an unknown function and its derivatives using multilayer feedforward networks. *Neural Networks*, vol. 3, no. 5, pp. 551-560, 1990.

HORNIK, K., STINCHCOMBE, M., WHITE, H., AUER, P. Degree of Approximation Results for Feedforward Networks Approximating Unknown Mappings and Their Derivatives. *Neural Computation*, vol. 6, pp. 1262-1275, 1994.

KINGMA, D.P., BA, J.L. Adam: A Method for Stochastic Optimization. International Conference on Learning Representations, pp. 1-13, 2015.

LI, M., ZHANG, T., CHEN, Y., SMOLA, A.J. Efficient Mini-batch Training for Stochastic Optimization. Proceedings of the 20th ACM SIGKDD international Conference on Knowledge Discovery and Data Mining (KDD'2014), pp. 661-670, 2014.

MARTENS, J. Deep learning via Hessian-free optimization. Proceedings of the 27th International Conference on Machine Learning (ICML'2010), pp. 735-742, 2010.

NERRAND, O., ROUSSEL-RAGOT, P., PERSONNAZ, L., DREYFUS, G. Neural Networks and Nonlinear Adaptive Filtering: Unifying Concepts and New Algorithms. *Neural Computation*, vol. 5, no. 2, pp. 165-199, 1993.

QIAN, N. On the momentum term in gradient descent learning algorithms. Neural Networks, vol. 12, pp. 145-151, 1999.

RUDER, S. An overview of gradient descent optimization algorithms. arXiv: 1609.04747v2, 14 pages, 2017.

WILSON, A.C., ROELOFS, R., STERN, M., SREBRO, N., RECHT, B. The Marginal Value of Adaptive Gradient Methods in Machine Learning. arXiv: 1705.08292v2, 2018.

ZEILER, M.D. ADADELTA: An Adaptive Learning Rate Method. arXiv: 1212.5701, 2012.