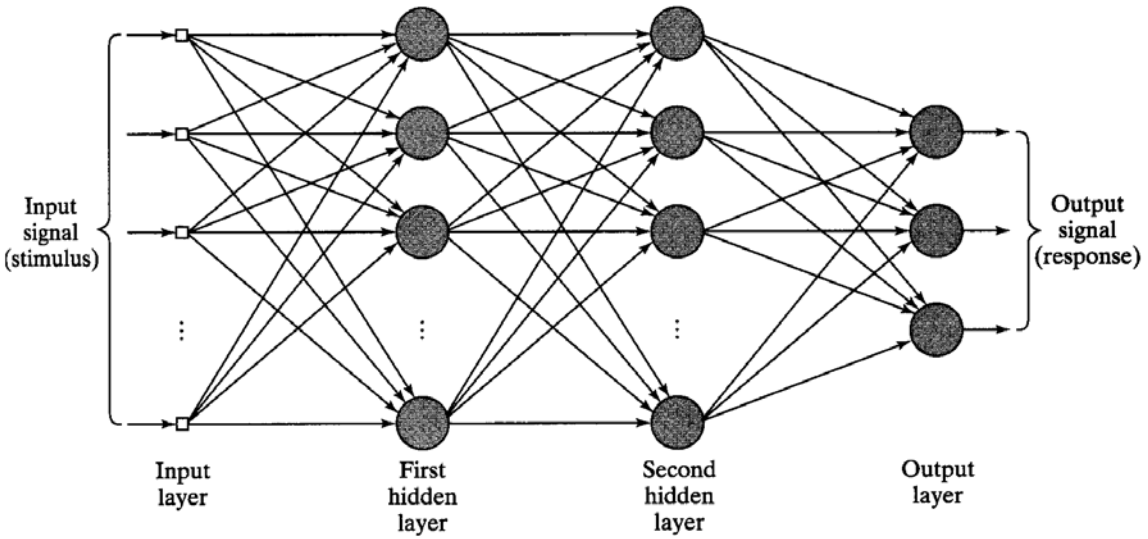
Multilayer Perceptron

Redes Neurais Artificiais Multilayer Perceptrons



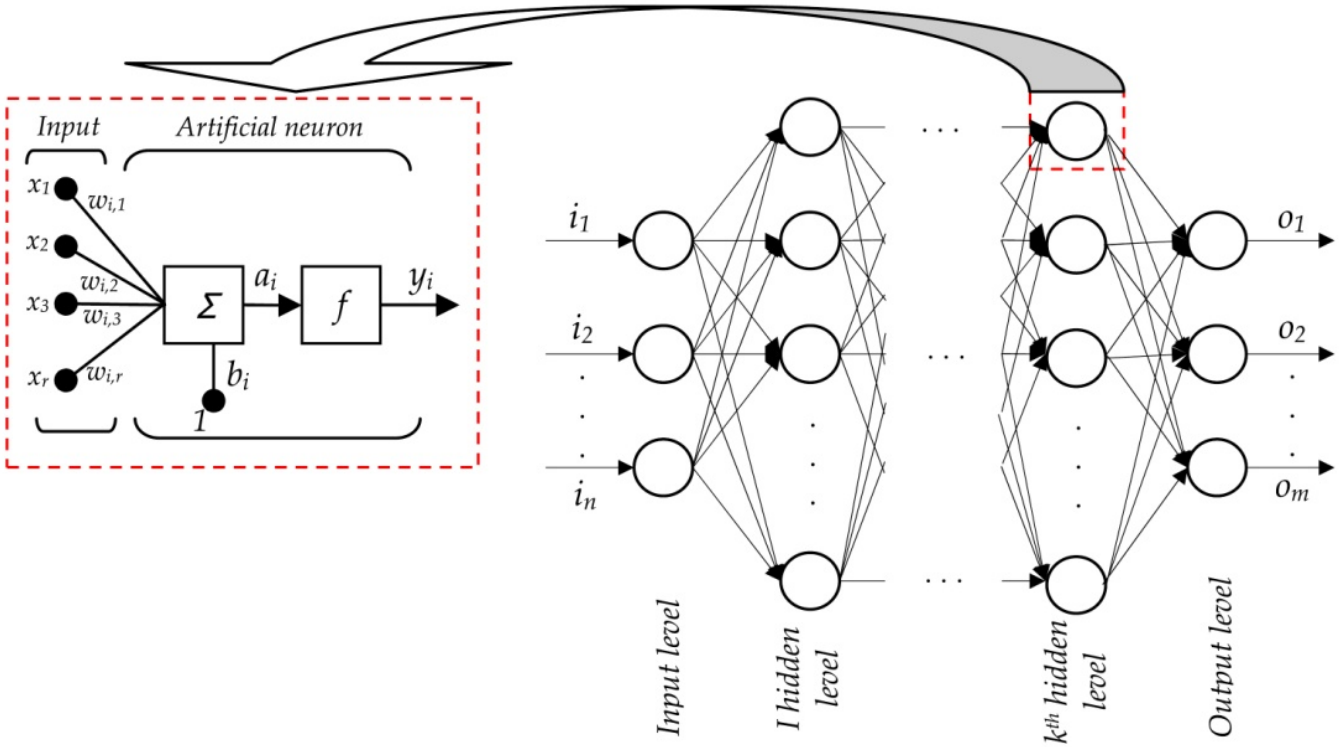
Arquitetura de uma rede neural *multilayer perceptron* com duas camadas escondidas.

Redes MLPs têm sido aplicadas com sucesso em uma variedade de áreas, desempenhando tarefas tais como:

classificação de padrões (reconhecimento), controle e processamento de sinais.

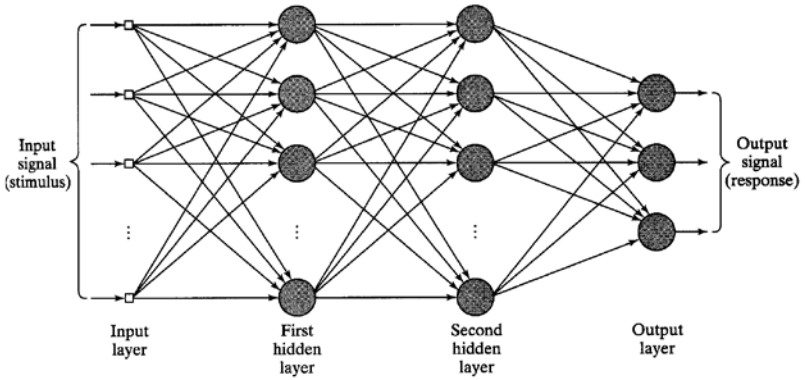
- Uma RNA MLP é constituída por um conjunto de nós fonte, os quais formam a camada de entrada da rede (*input layer*), uma ou mais camadas escondidas (*hidden layers*) e uma camada de saída (*output layer*).
- Com exceção da camada de entrada, todas as outras camadas são constituídas por neurônios e, portanto, apresentam capacidade computacional.

A RNA Multilayer Perceptron é uma generalização do Perceptron



Fonte: [2]

Duas características da arquitetura do MLP são imediatamente aparentes:



1. Uma RNA MLP é uma rede progressiva (*feedforward*), pois as saídas dos neurônios em qualquer particular camada se conectam unicamente às entradas dos neurônios da camada seguinte, sem a presença de laços de realimentação. Consequentemente, o sinal de entrada se propaga através da rede, camada a camada, em um sentido progressivo.

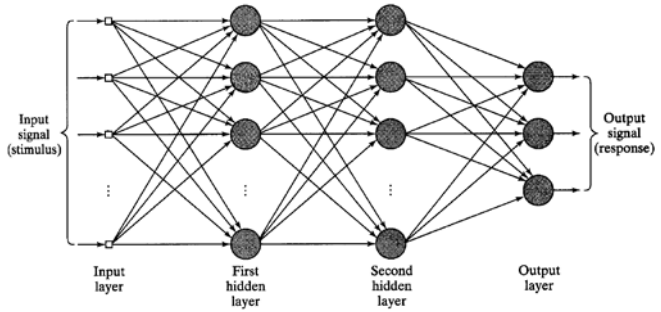
2. A rede pode ser:

Completamente conectada:	Cada nó (computacional ou não) em uma camada é conectado a todos os outros nós da camada adjacente.
Parcialmente conectada:	Sinapses podem ser propositalmente desconectadas, objetivando minimizar a especialização do MLP (maximizar a capacidade de generalização).
Localmente conectada:	Sinapses podem ser desconectadas para que a conectividade de um (ou mais) neurônio em uma camada da rede focalize um sub-conjunto de todas as possíveis entradas, objetivando modularizar a arquitetura do MLP, incentivando a cooperação entre as regiões modulares assim obtidas (uma região não afeta diretamente a outra e assim uma não "atrapalha" o processamento da outra).

- Na prática, a falta de uma determinada sinapse em um MLP é emulada fazendo-se a transmitância constante e igual a zero.
- Neste estudo, no entanto, consideraremos apenas MLPs completamente conectados.

O *design* e a configuração de uma rede MLP

requer a consideração dos seguintes aspectos:



• N°. de nós fonte na camada de entrada	• É função da dimensionalidade do espaço de observação, que é responsável pela geração dos sinais de entrada (por exemplo, pressão na tubulação , temperatura na tubulação, concentração do reagente denotam dimensionalidade 3 – 3 nós de entrada, portanto).
• N°. de neurônios na camada de saída	• É função da dimensionalidade requerida da resposta desejada (por exemplo, potência do compressor do gás, potência do refrigerador do gás denotam dimensionalidade 2 – 2 nós de saída, portanto).
• N°. de camadas escondidas • N°. de neurônios em cada uma das camadas escondidas	• Determinam a complexidade do modelo do MLP escolhido e não há regras determinadas para tal especificação. • A função das camadas escondidas em um MLP é a de influir na relação entrada-saída da rede de uma forma ampla. • Um MLP com uma ou mais camadas escondidas é apto a extrair as estatísticas de ordem superior de algum desconhecido processo aleatório subjacente, responsável pelo comportamento dos dados de entrada, processo sobre o qual a rede está tentando adquirir conhecimento. • O MLP adquire uma perspectiva global do processo aleatório, apesar de sua conectividade local, em virtude do conjunto adicional de pesos sinápticos e da dimensão adicional de interações neurais proporcionada pelas camadas escondidas.
• Especificação dos pesos sinápticos que interconectam os neurônios nas diferentes camadas da rede	• Requer a utilização de algoritmo de treino supervisionado. • O algoritmo de treino quase universalmente utilizado é o algoritmo de retro-propagação do erro, conhecido na literatura como ***Backpropagation Algorithm***

O algoritmo *backpropagation* (ou simplesmente *backprop*) baseia-se na heurística de aprendizagem por correção de erro (em que o erro é retro-propagado da camada de saída para as camadas intermediárias do MLP através do gradiente local de cada neurônio – a ser discutido adiante).

O *Backprop* é uma generalização do Algoritmo *Least Mean Square* (LMS) desenvolvido por Bernard Widrow, que já estudamos para o caso especial de um único neurônio linear.

O termo *backpropagation* surgiu após **1985**. No entanto, a idéia básica foi primeiramente descrita por Werbos em sua tese de doutorado em **1974**. Em **1986**, foi redescoberto por Rumelhart, Hinton e Williams e popularizado através da publicação do livro *Parallel Distributed Processing* de Rumelhart e McClelland em **1986**.

O desenvolvimento do *backpropagation* representa um marco fundamental em redes neurais, pois é um método computacionalmente eficiente para o treinamento de redes MLPs e por ter resolvido o problema de realizar a propagação reversa do erro em RNAs com múltiplas camadas, problema este que atrasou por muitos anos o desenvolvimento da área de redes neurais artificiais.

Algoritmo LMS aplica a correção $\Delta\underline{w}(n)$ aos pesos sinápticos $\underline{w}(n)$, tendo como base a direção contrária do gradiente local da superfície de erro $\mathrm{J}(\underline{w}(n))$ relativo aos pesos sinápticos.	**Algoritmo *Backpropagation*** aplica a correção $\Delta w_{ji}(n)$ ao peso sináptico $w_{ji}(n)$, sendo i o índice do respectivo nó de entrada do neurônio j, tendo como base a direção contrária do **gradiente local** da superfície de erro $\varepsilon\left(w_{ji}(n)\right)$ relativo aos pesos sinápticos, mesmo quando o neurônio j esteja em uma camada escondida, e, portanto, não haja erro local explícito no neurônio em questão.
LMS – Regra Delta $\rightarrow \underline{w}(n+1)=\underline{w}(n)-\eta\underline{\nabla}\,\mathrm{J}(\underline{w}(n))$, $\eta>0 \rightarrow$ razão de aprendizado do LMS	**Algoritmo Backpropagation – Regra Delta** $\rightarrow w_{ji}(n+1)=w_{ji}(n)-\eta\dfrac{\partial\varepsilon(n)}{\partial w_{ji}(n)}$, $\eta>0 \rightarrow$ razão de aprendizado do *backprop*

Uma rede MLP apresenta três características distintas, de cuja combinação com a habilidade de aprender através da experiência (através do treinamento), deriva sua capacidade computacional:

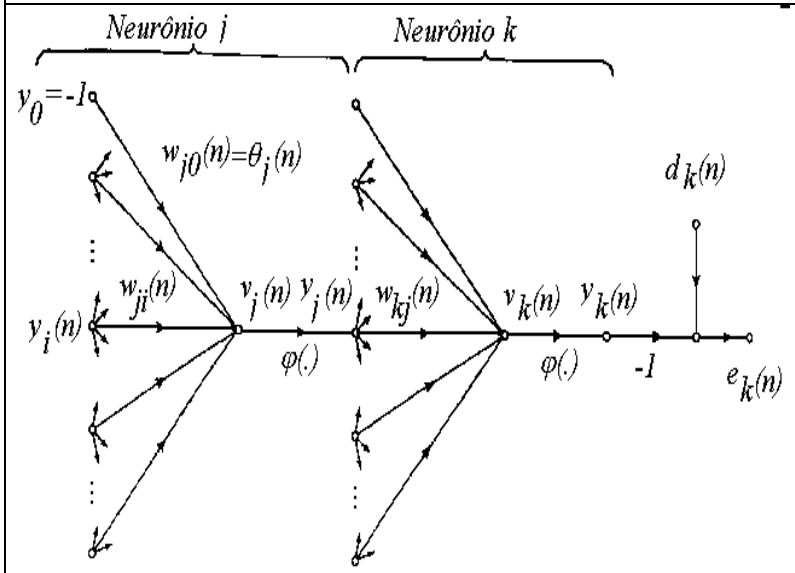
1. O modelo de cada neurônio de um MLP inclui uma função de ativação não-linear. Esta não-linearidade é suave (a função é diferenciável em qualquer ponto), ao contrário da função utilizada no modelo do Perceptron de Rosenblatt (função signum). Uma forma comumente utilizada de não-linearidade que satisfaz este requisito é a não-linearidade sigmoidal definida pela função logística:

$$y_j = \frac{1}{1+\exp(-v_j)}$$

onde v_j é o potencial de ativação (isto é, a soma ponderada de todas as entradas sinápticas mais a polarização) do neurônio j, e y_j é a saída do neurônio.

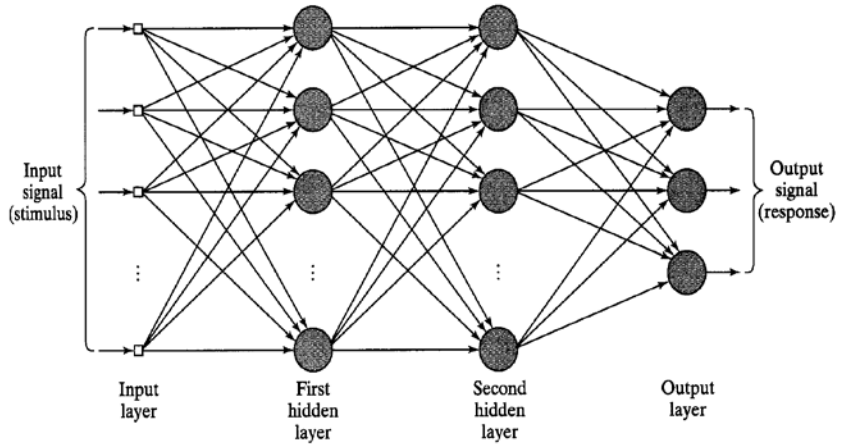
2. Um MLP contém uma ou mais camadas de neurônios escondidos que não são parte da camada de entrada ou da camada de saída da rede. Estes neurônios escondidos possibilitam que a rede aprenda tarefas complexas, extraindo progressivamente mais características significativas dos padrões de entrada (vetores de entrada).

3. A rede MLP exibe um alto grau de conectividade, determinado pelas sinapses da rede. Uma mudança na conectividade da rede requer uma mudança na população de conexões sinápticas, ou pesos sinápticos.

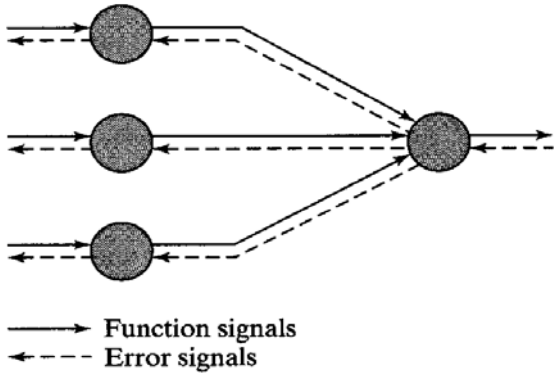


Estas características são também responsáveis pelas dificuldades encontradas na análise de tais redes:

- A presença das não-linearidades distribuídas e a alta conectividade tornam difícil a análise teórica das redes MLPs.
- O conhecimento aprendido sobre o ambiente é representado pelos valores assumidos pelos pesos sinápticos da rede. A natureza distribuída deste conhecimento ao longo da rede a torna de difícil interpretação.
- O uso de neurônios escondidos torna o processo de aprendizado mais difícil de ser "visualizado" na estrutura da rede.



MLP com duas camadas escondidas:
O sinal flui através da rede MLP no sentido direto, da esquerda para a direita, e de camada a camada.



Detalhe parcial de uma rede MLP, ilustrando as direções dos dois fluxos básicos de sinal:

- propagação direta dos sinais e
- retro-propagação dos sinais de erro.

Dois tipos de sinais são identificados nesta rede:

Sinais funcionais:

- São estímulos que chegam aos nós de entrada da rede, se propagam de forma direta (neurônio a neurônio) através da rede e emergem da camada de saída da rede como sinais de saída.
- Cada neurônio de uma MLP tem aplicado às suas entradas um conjunto de sinais funcionais que gera um sinal funcional na saída do respectivo neurônio.
- Na camada de entrada de uma MLP o conjunto de sinais funcionais aplicado a cada neurônio é o próprio conjunto de sinais de entrada (vetor de entrada).
- A denominação sinal funcional decorre do fato de que estes sinais são obtidos na saída de cada neurônio como uma função dos sinais de entrada do respectivo neurônio.

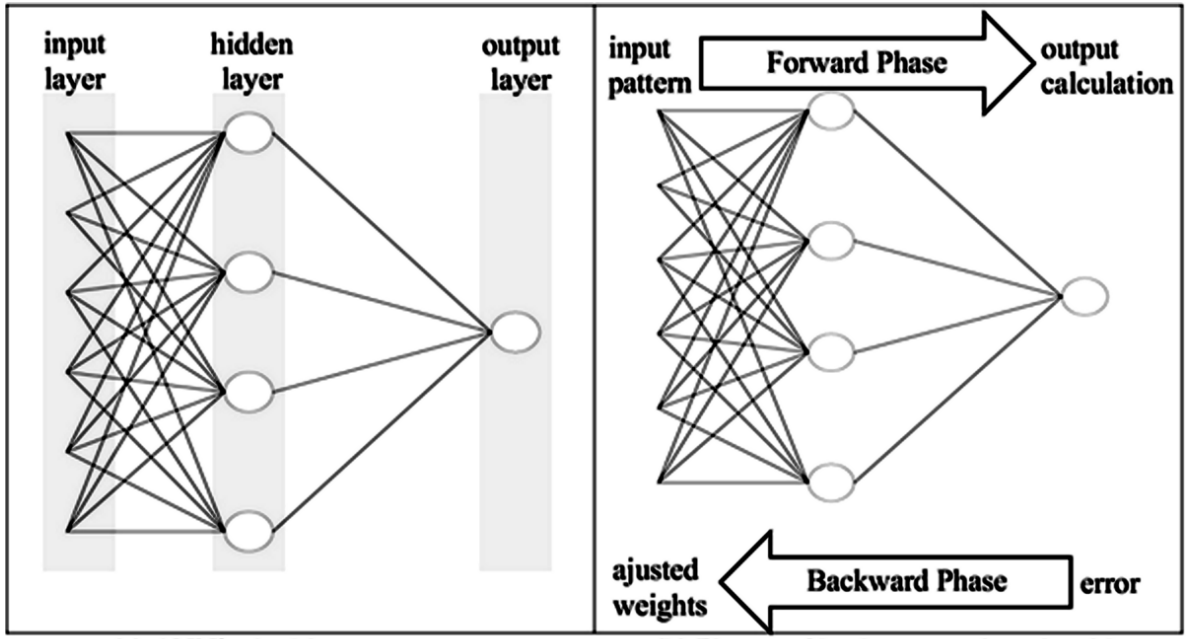
Sinais de Erro:

- Um sinal de erro se origina em um neurônio de saída da MLP e se propaga de volta (camada a camada) através da rede.
- Este sinal é referido como sinal de erro porque seu cálculo, a cada neurônio da rede, é um erro local implícito no **gradiente local** do neurônio..

Cada neurônio, seja de uma camada escondida ou da camada de saída de um MLP executa duas operações:

1. A computação do **sinal funcional** na saída de cada neurônio, o qual é expresso como uma função contínua não-linear do sinal funcional de entrada e dos pesos sinápticos associados com aquele neurônio.

2. A computação do **gradiente local** (que é uma estimativa "esmaecida" do gradiente global) de cada neurônio nas *hidden layers* é efetuada no sentido *backward* (da saída p/ a entrada - ver figura abaixo), começando com o gradiente dos neurônios da *output layer*. Para cada j-ésimo neurônio na iteração n, o gradiente $\frac{\partial \varepsilon(n)}{\partial w_{ji}(n)}$ da superfície de erro $\varepsilon\left(w_{ji}(n)\right)$ é determinado como se manualmente efetuássemos uma perturbação (variação) infinitesimal $\partial w_{ji}(n)$ nas transmitâncias das sinapses $w_{ji}(n)$ conectadas às entradas do neurônio e medíssemos a variação $\partial\varepsilon(n)$ resultante para o erro quadrático na camada de saída do MLP, de forma semelhante ao que é manualmente feito nos potenciômetros do ADALINE, já visto no capítulo de introdução. Mas as sinapses $w_{ji}(n)$ do neurônio j determinam o potencial de ativação $v_j(n)$ local, de modo que o gradiente local pode ser obtido através de $\delta_j(n) = -\frac{\partial \varepsilon(n)}{\partial v_j(n)}$, onde o sinal negativo decorre de $e_j(n) = d_j(n) - y_j(n)$ - Ver dedução no Apêndice B.

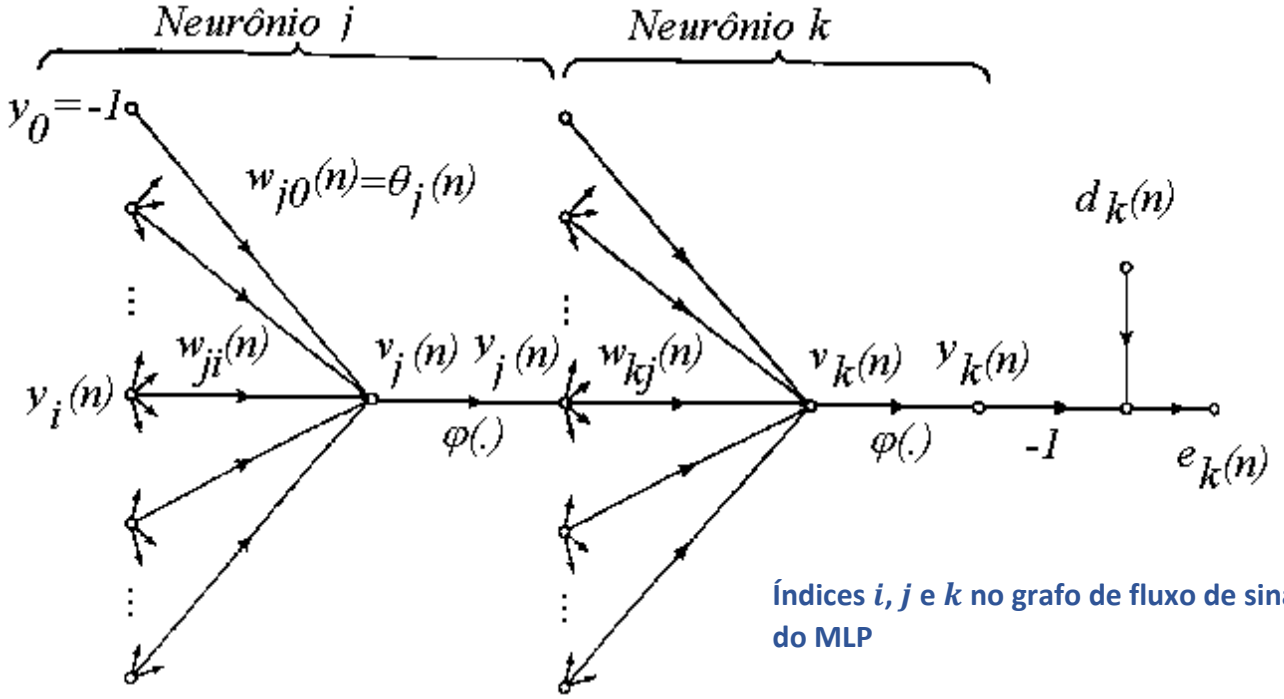


(a) ANN's Architecture (b) Phases of backpropagation algorithm

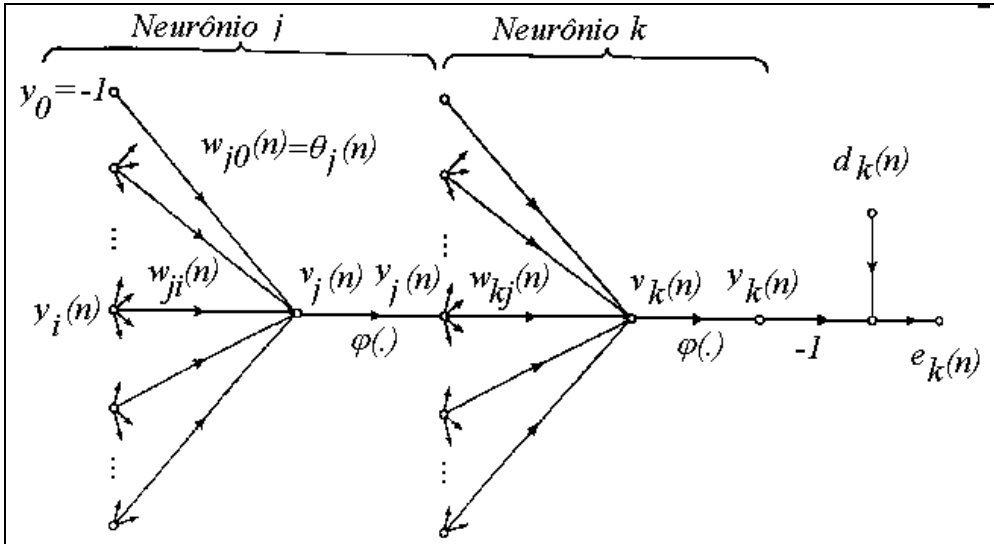
Fonte:[1]

Notação de índices e variáveis do algoritmo *backpropagation*

- Os índices i, j e k se referem a diferentes neurônios no MLP.
- Os sinais funcionais se propagam através da rede, da esquerda para a direita, sendo que o neurônio j está na camada à direita do neurônio i, e o neurônio k está na camada à direita do neurônio j, quando o neurônio j é uma unidade escondida $(i \rightarrow j \rightarrow k)$.
- Na iteração n, o n-ésimo padrão de treino (vetor-exemplo $\underline{x}(n)$) é apresentado aos nós de entrada do MLP.
- O símbolo $e_k(n)$ se refere ao sinal de erro na saída do neurônio de saída k para a iteração n.
- O símbolo $d_k(n)$ se refere à resposta desejada para o neurônio de saída k e é usado para computar $e_k(n)$.
- O símbolo $y_k(n)$ se refere ao sinal funcional encontrado na saída do neurônio k, na iteração n.
- O símbolo $w_{ji}(n)$ denota o peso sináptico que conecta a saída do neurônio i à entrada do neurônio j, na iteração n. A correção aplicada a este peso na iteração n é denotada por $\Delta w_{ji}(n)$.



Índices i, j e k no grafo de fluxo de sinal do MLP



➔ O potencial de ativação (isto é, a soma ponderada de todas as entradas sinápticas mais a polarização) do neurônio *j* na iteração *n* é denotado por $v_j(n)$ e constitui o sinal aplicado à função de ativação associada ao neurônio *j*.

➔ A função de ativação que descreve a relação funcional entrada-saída da não-linearidade associada ao neurônio *j* é denotada por $\varphi_j(\cdot)$.

➔ A polarização (*bias*) aplicada ao neurônio *j* é denotada por b_j. O efeito do *bias* é representado por uma sinapse de peso $w_{j0} = b_j$ conectada a uma entrada fixa igual a (+1).

➔ Alternativamente, a polarização pode ser gerada por uma sinapse de peso $w_{j0} = \theta_j$ conectada a uma entrada de valor fixo e igual a (–1), quando recebe o nome de *threshold*.

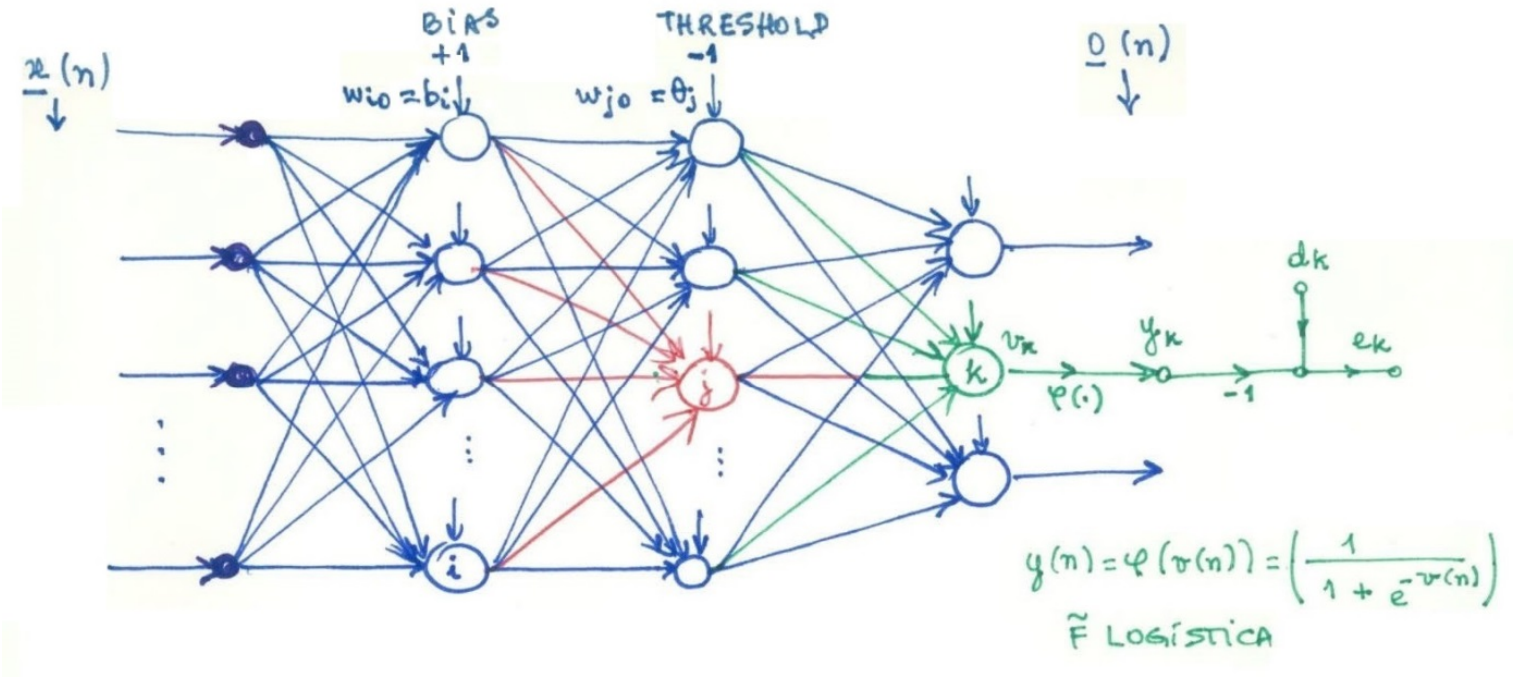
➔ Para todos os fins práticos, as duas alternativas apresentam os mesmos resultados. Neste estudo consideraremos apenas o nome genérico “polarização”, a qual pode ser originada de um valor fixo positivo (+1) ou negativo (-1).

➔ O *i*-ésimo componente do vetor de entrada do MLP é denotado por $x_i(n)$.

➔ O *k*-ésimo componente do vetor de saída do MLP é denotado por $o_k(n)$.

➔ O parâmetro razão de aprendizado é denotado por η.

Sinais envolvidos na operação do algoritmo *backpropagation*

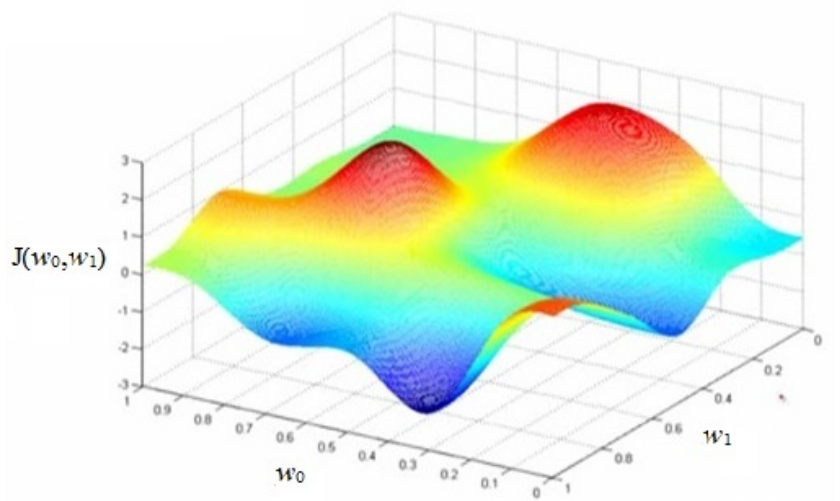


Sinal de Erro na saída do neurônio k da camada de saída, na iteração n (isto é, na apresentação do n-ésimo vetor de treinamento)	$e_k(n) = d_k(n) - y_k(n)$
Valor Instantâneo do Erro Quadrático para o neurônio de saída k, na iteração n.	$(1/2)e_k^{\,2}(n)$
Valor Instantâneo da Soma dos Erros Quadráticos ou Energia do Erro, na iteração n. ML é o número de neurônios na camada de saída da rede - **estes são os únicos neurônios visíveis para os quais os sinais de erro podem ser calculados de forma direta.**	$\varepsilon(n) = \frac{1}{2}\sum_{k=0}^{ML-1} e_k^{\,2}(n)$
Erro Médio Quadrático (MSE) ou Energia Média do Erro. N é o número total de padrões (vetores-exemplo $\underline{x}(n)$) contidos no conjunto de treino. ε_{av} é a média de $\varepsilon(n)$ resultante para todos os N vetores $\underline{x}(n)$ do conjunto de treino, isto é, ε_{av} é determinado por **época** de treino.	$\varepsilon_{av} = MSE = \frac{1}{N}\sum_{n=0}^{N-1} \varepsilon(n)$

O MSE ou ε_{av} é função dos parâmetros livres do MLP (pesos sinápticos w), porque é função de $\varepsilon(n)$ que, por sua vez, é função de $e(n)$.

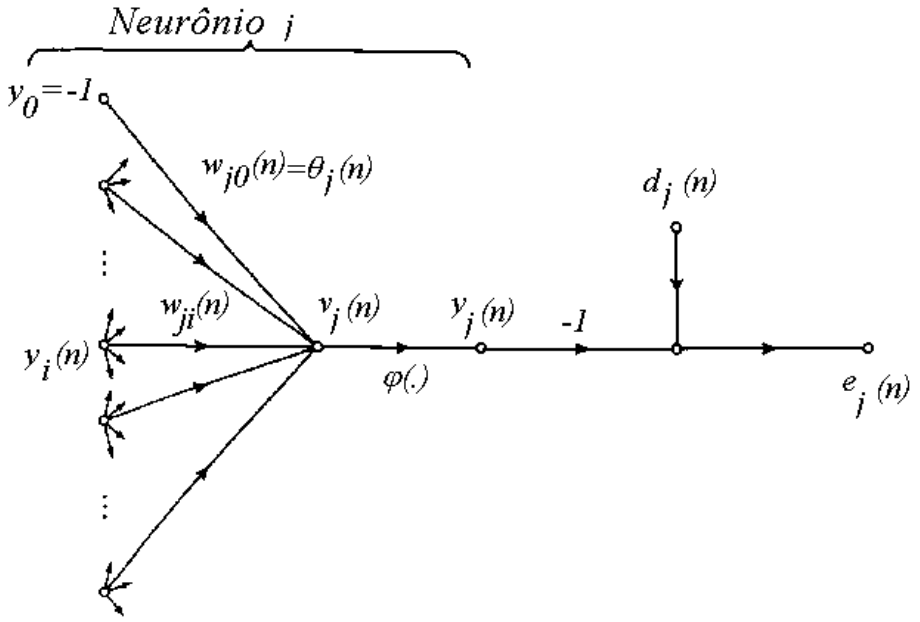
Valor Instantâneo do Erro → $e = f(w)$
Valor Instantâneo da Soma dos Erros Quadráticos → $\varepsilon = g(e)$
Erro Médio Quadrático → $\varepsilon_{av} = MSE = h(\varepsilon)$

- Para um dado conjunto de treino, ε_{av} representa a Função de Custo do processo de minimização do erro de aprendizado, constituindo uma medida inversa do desempenho do processo de aprendizado a partir do conjunto de treino.
- ε_{av} estima a função de custo $J(\boldsymbol{w})$ a partir da média aritmética de $\varepsilon(n)$ resultante para todos os N vetores $\underline{x}(n)$ do conjunto de treino. O domínio da função de custo J é o conjunto de pesos sinápticos $\boldsymbol{w}$.



- Para minimizar ε_{av} os pesos sinápticos são atualizados à cada apresentação n de um novo padrão ao MLP através do vetor de entrada $\underline{x}(n)$ até o término de uma **Época**.
- Uma **Época** consiste no intervalo correspondente à apresentação de todos os N vetores-exemplo $\underline{x}(n)$ do conjunto de treino à camada de entrada do MLP.
- O processo de aprendizagem, i.e., o ajuste do conjunto de pesos sinápticos $\boldsymbol{w}$ a cada iteração n, é feito de acordo a Regra Delta em função do erro $e_k(n) = d_k(n) - y_k(n)$ de cada neurônio k na camada de saída do MLP, ajuste que é efetuado para cada padrão $\underline{x}(n)$ apresentado aos nós de entrada do MLP.

Dado que o MSE ou ε_{av} é função dos parâmetros livres da RNA (conjunto de pesos sinápticos $\boldsymbol{w}$), a superfície da função de custo global $J(\boldsymbol{w})$ do problema é aproximada se variarmos todos os pesos sinápticos do conjunto de pesos $\boldsymbol{w}$ e plotarmos o ε_{av} resultante (lembrando que ε_{av} resulta da apresentação aos nós de entrada do MLP de todos os N vetores $\underline{x}(n)$ do conjunto de treino, isto é, ε_{av} é determinado por época de treino). Quanto mais representativo do problema a ser resolvido forem os N vetores $\underline{x}(n)$ do conjunto de treino, melhor será a aproximação da função de custo global $J(\boldsymbol{w})$.



Grafo de fluxo de sinal no neurônio j. O neurônio j é alimentado por um conjunto de sinais produzidos na saída dos neurônios da camada i à sua esquerda.

- O potencial de ativação $v_j(n)$ aplicado na entrada da não-linearidade associada ao neurônio *j* é, portanto

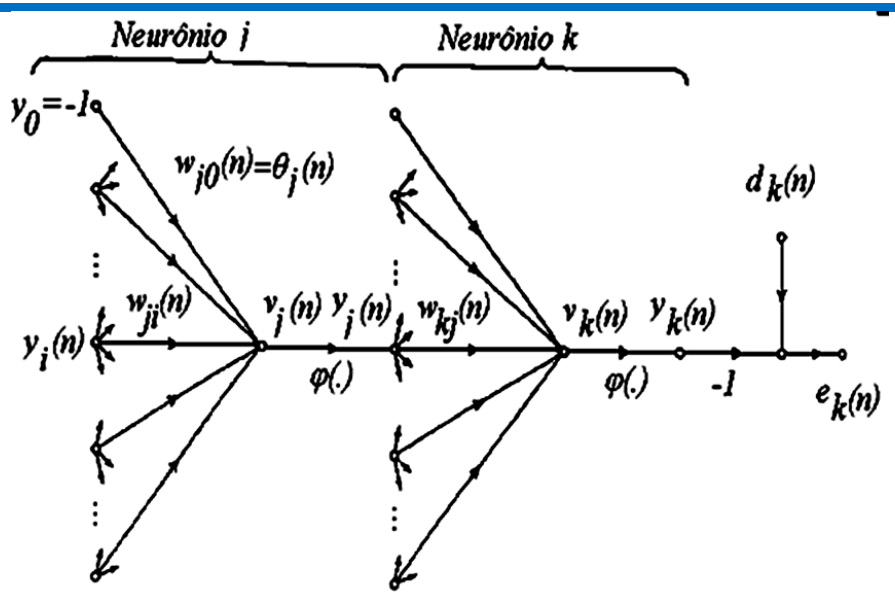
$$v_j(n) = \sum_{i=0}^{m} w_{ji}(n) y_i(n) \ ,$$

onde *m* é o número total de entradas aplicadas ao neurônio *j*.

- O peso sináptico w_{j0} (correspondente à entrada fixa y_0 = -1) define a polarização θ_j aplicada ao neurônio *j*.
- $w_{ji}(n)$ é o peso sináptico que conecta a saída do neurônio *i* ao neurônio *j*
- $y_i(n)$ é o sinal no *i*-ésimo nó de entrada do neurônio *j*, ou equivalentemente, o sinal na saída do neurônio *i*.
- Assim, o sinal $y_j(n)$ resultante na saída do neurônio *j* na iteração *n* é:

$$y_j(n) = \varphi_j\left(v_j(n)\right) \ .$$

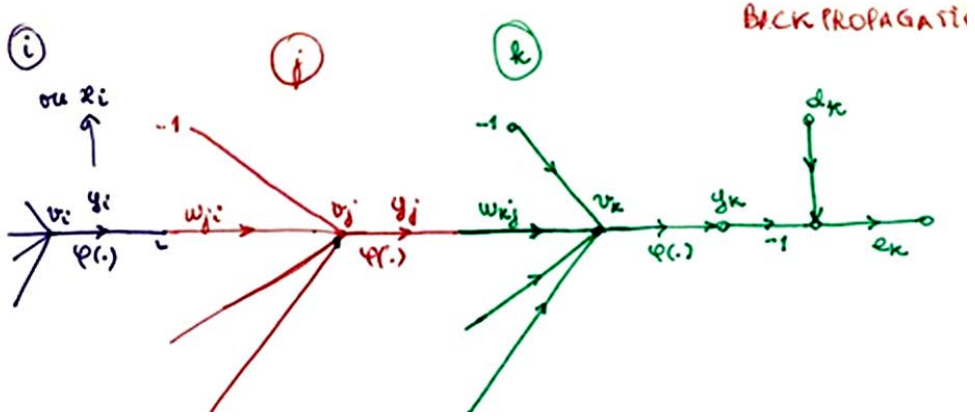
- O **algoritmo *Backpropagation*** estabelece o aprendizado de um MLP através da Regra Delta, com base no **gradiente local de cada neurônio**, com o ajuste efetuado na sinapse $w_{ji}(n)$ do neurônio j dado por $\Delta w_{ji}(n) = \eta \delta_j(n) y_i(n)$, onde:



- $\Delta w_{ji}(n)$ é o ajuste aplicado à *i*-ésima sinapse do neurônio *j*,
- $y_i(n)$ é o sinal de entrada no *i*-ésimo nó de entrada do neurônio *j*,
- $y_i(n)$ é também o sinal na saída do neurônio *i*, pertencente à camada à esquerda da que pertence o neurônio *j*, se este não estiver na primeira camada escondida. Se o neurônio *j* estiver na primeira camada escondida então $y_i(n)$ corresponde ao *i*-ésimo nó de entrada $x_i(n)$ do MLP.

- $\delta_j(n)$ é o gradiente local do neurônio *j*, definido por (vide dedução no Apêndice B):

$$\delta_j(n) = \begin{cases} \varphi'_j(v_j(n)) e_j(n) & \text{, quando neurônio } j \text{ é de saída} \\ \varphi'_j(v_j(n)) \sum_k \delta_k(n) w_{kj}(n) & \text{, quando neurônio } j \text{ é escondido} \end{cases}$$



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

$\varphi'_j(v_j(n)) \, e_j(n)$

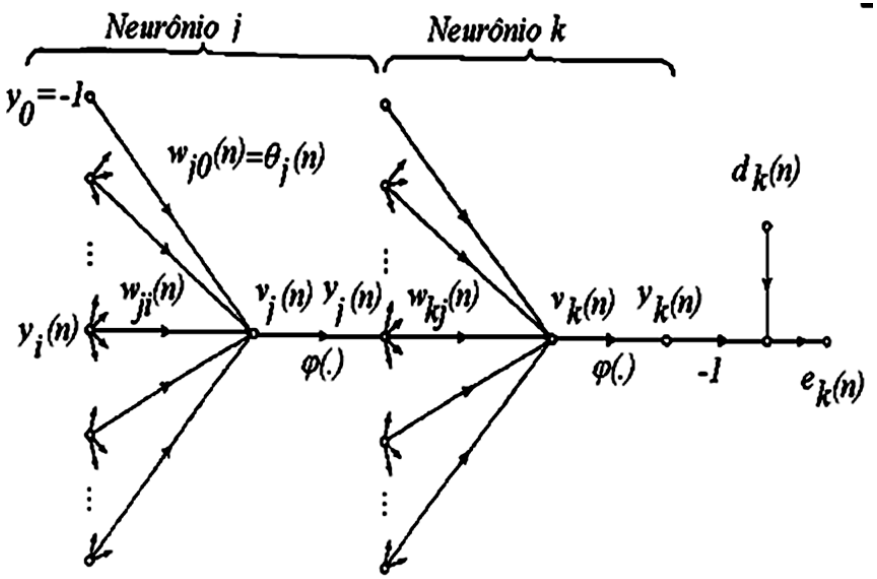
j É NEURÔNIO DE SAÍDA

$[\varphi'_k(v_k(n)) e_k(n)]$

$\varphi'_j(v_j(n)) \sum_k \delta_k(n) w_{kj}(n)$

j É NEURÔNIO ESCONDIDO

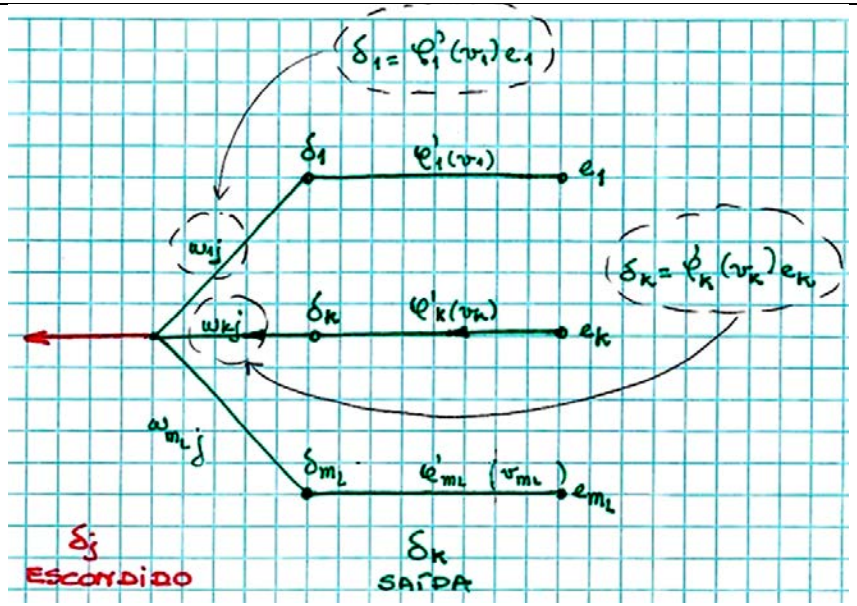
Gradiente local do neurônio *j* quando *j* é neurônio de saída



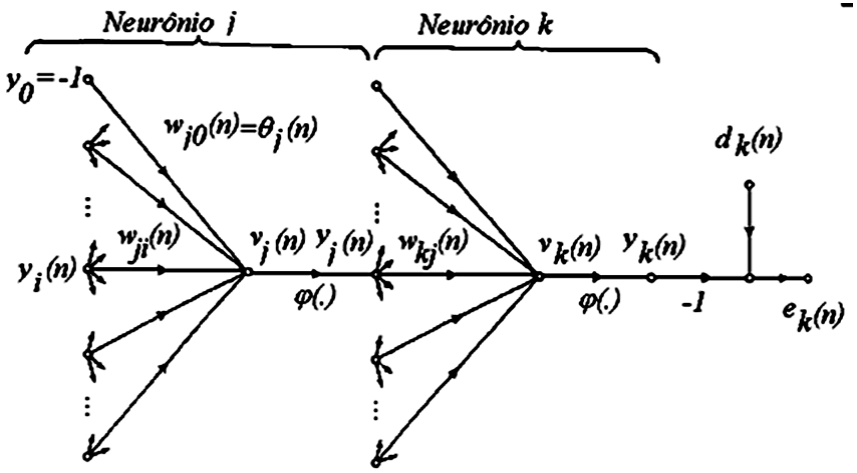
$$\delta_j(n)=\begin{cases}\varphi_j'(v_j(n))e_j(n) & \text{, neurônio } j \text{ é de saída}\\ \varphi_j'(v_j(n))\sum_k \delta_k(n)w_{kj}(n) & \text{, neurônio } j \text{ é escondido}\end{cases}$$

- O gradiente local $\delta_j(n)$ para o neurônio de saída j é igual ao produto do correspondente sinal de erro $e_j(n)$ pela derivada $\varphi_j'(v_j(n))$ da função de ativação associada (**ver slide 21 do capítulo de Introdução**).

- Neste caso o fator chave necessário envolvido no cálculo do ajuste dos pesos $\Delta w_{ji}(n)$ é o sinal de erro $e_j(n)$ na saída do neurônio *j*.

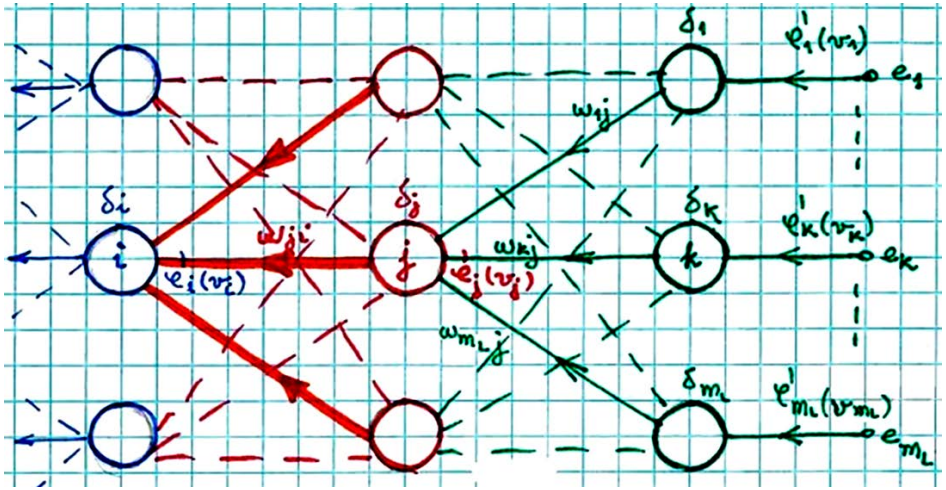


Gradiente local do neurônio *j* quando *j* é neurônio escondido

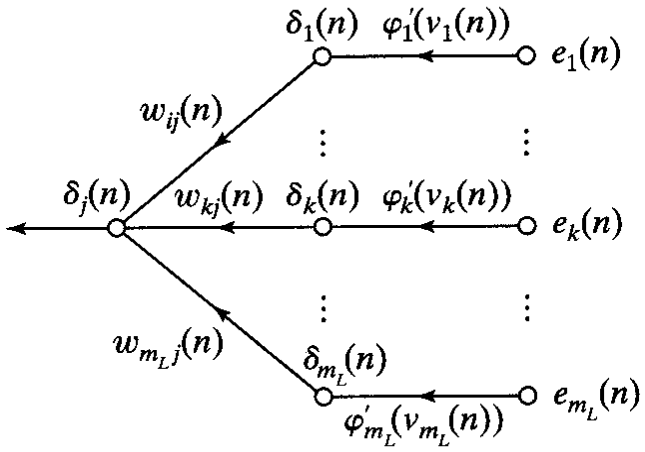


$$\delta_j(n)=\begin{cases}\varphi_j'\left(v_j(n)\right)e_j(n) & \text{, neurônio } j \text{ é de saída}\\ \varphi_j'\left(v_j(n)\right)\sum_k \delta_k(n)w_{kj}(n) & \text{, neurônio } j \text{ é escondido}\end{cases}$$

- Quando o neurônio *j* está localizado em uma camada escondida, conforme mostra a Figura, mesmo não sendo diretamente acessíveis, tais neurônios dividem a responsabilidade pelo erro resultante na camada de saída.
- A questão, no entanto, é saber como penalizar ou recompensar os pesos sinápticos de tais neurônios pela sua parcela de responsabilidade, já que não existe resposta desejada especificada neste local da RNA MLP e, portanto, não há como calcular o sinal de erro.



Grafo de fluxo de sinal mostrando o processo de retro-propagação dos sinais de erro na camada de saída para um neurônio j da camada escondida imediatamente à esquerda.



m_L é o número de neurônios da camada de saída.

$$\delta_j(n) = \begin{cases} \varphi_j'(v_j(n))e_j(n) & \text{, neurônio } j \text{ é de saída} \\ \varphi_j'(v_j(n))\sum_k \delta_k(n)w_{kj}(n) & \text{, neurônio } j \text{ é escondido} \end{cases}$$

A solução é computar o sinal de erro **recursivamente** para o neurônio escondido j, retro-propagando os sinais de erro de todos os neurônios à direita do neurônio j, aos quais a saída deste encontra-se conectada.

- O fator $\varphi_j'(v_j(n))$ envolvido na computação do gradiente local $\delta_j(n)$ na equação depende somente da função de ativação associada com o neurônio escondido j.

- Os demais fatores envolvidos no somatório sobre k dependem de dois conjuntos de termos:
  - O primeiro, $\delta_k(n)$, requer conhecimento dos sinais de erro $e_k(n)$ recursivamente retro-propagados, a partir de todos aqueles neurônios localizados na camada imediatamente à direita do neurônio escondido j e que estão diretamente conectados a ele.
  - O segundo conjunto de termos, $w_{kj}(n)$, consiste dos pesos sinápticos dos neurônios à direita do neurônio j e que com ele estabelecem conexão.

Os Dois Passos Computacionais do Algoritmo *Backpropagation*

No **passo direto (*forward pass*)** os pesos sinápticos permanecem inalterados em todo o MLP e os sinais são propagados da entrada para a saída do MLP, de neurônio a neurônio.

1. O sinal que resulta na saída do neurônio *j* é computado por $y_j(n) = \varphi(v_j(n))$, onde:

   - $v_j(n)$ é o potencial de ativação do neurônio *j*, definido por $v_j(n) = \sum_{i=0}^{m} w_{ji}(n) y_i(n)$,

     sendo m o número total de entradas aplicadas ao neurônio *j*,
   - $w_{ji}(n)$ é o peso sináptico conectando a saída do neurônio i ao neurônio j e
   - $y_i(n)$ é o sinal de entrada do neurônio *j*, ou equivalentemente, o sinal na saída do neurônio i.

Se o neurônio *j* está na primeira camada escondida do MLP, então o índice i refere-se ao i-ésimo nó de entrada do MLP, para o qual escreve-se $y_i(n) = x_i(n)$ onde $x_i(n)$ é o i-ésimo componente do vetor de entrada do neurônio *j*.	Se o neurônio *j* está na camada de saída, o índice j refere-se ao *j*-ésimo nó de saída do MLP, para o qual escreve-se $y_j(n) = o_j(n)$ sendo $o_j(n)$ o *j*-ésimo componente do vetor de saída.

2. A saída $y_j(n)$ é comparada com a resposta desejada $d_j(n)$ sendo obtido o sinal de erro $e_j(n)$ para o *j*-ésimo neurônio de saída.

Portanto, o passo direto começa na primeira camada escondida pela apresentação do vetor de entrada a ela e termina na camada de saída com a determinação do sinal de erro para cada neurônio desta camada.

O **passo reverso (*backward pass*)** começa na camada de saída, e os sinais de erro são propagados da saída para a entrada do MLP, de camada e camada (portanto, de volta para a entrada – retro-propagando), e **recursivamente** computando os gradientes locais para cada neurônio.

<table>
<tr><td colspan="2">Este processo recursivo de determinação dos gradientes locais efetua o ajuste nos pesos sinápticos do MLP de acordo com a Regra Delta $\Delta w_{ji}(n) = \eta \delta_j(n) y_i(n)$, conforme já discutido em slide anterior.</td></tr>
<tr><td>$\delta_j(n)$, quando neurônio j é de saída</td><td>$\delta_j(n)$, quando neurônio j é de camada escondida</td></tr>
<tr><td>1. O gradiente local é simplesmente o sinal de erro daquele neurônio multiplicado pela derivada de sua não-linearidade, conforme $\delta_j(n) = \varphi'_j(v_j(n)) e_j(n)$.</td><td>3. Obtidos os gradientes locais para os neurônios da camada de saída, computa-se o gradiente local de cada neurônio na camada à esquerda, conforme $\delta_j(n) = \varphi'_j(v_j(n)) \sum_k \delta_k(n) w_{kj}(n)$</td></tr>
<tr><td>2. A partir do gradiente local de cada neurônio da camada de saída, computa-se os ajustes em todas as sinapses que alimentam a camada de saída, conforme $\Delta w_{ji}(n) = \eta \delta_j(n) y_i(n)$.</td><td>4. A partir do gradiente local de cada neurônio da camada à esquerda, computa-se os ajustes em todas as sinapses que alimentam esta camada, conforme $\Delta w_{ji}(n) = \eta \delta_j(n) y_i(n)$.</td></tr>
<tr><td></td><td>5. Este procedimento é continuado recursivamente, propagando os gradientes locais e ajustando os pesos sinápticos camada por camada, até a camada de entrada.</td></tr>
</table>

Note que durante cada n-ésima iteração passo direto - passo reverso ao longo da apresentação do conjunto de treino à RNA MLP, o vetor de entrada $\underline{x}(n)$ é mantido fixo.

Razão de Aprendizado e Fator de Momento

O algoritmo *backpropagation* provê uma aproximação da trajetória de movimento sobre a superfície de erro no espaço de pesos sinápticos, trajetória esta que, a cada ponto da superfície, segue a direção de descida mais íngreme (Regra Delta).

Quanto menor a razão de aprendizado η, menores serão as correções aplicadas aos pesos sinápticos da RNA MLP de uma iteração p/ a próxima, e mais suave será a trajetória no espaço de pesos. Isto é obtido sob o custo de uma lenta convergência do algoritmo até um valor de erro suficientemente pequeno.

Se, por outro lado, quanto maior a razão de aprendizado η, de modo a acelerar a convergência do algoritmo, as correções feitas nos pesos sinápticos podem resultar demasiadamente grandes, de modo que o algoritmo se torna instável (oscilatório).

Um método simples utilizado para acelerar a convergência e manter a trajetória estável é o acréscimo do chamado **Fator de Momento à Regra Delta**.:

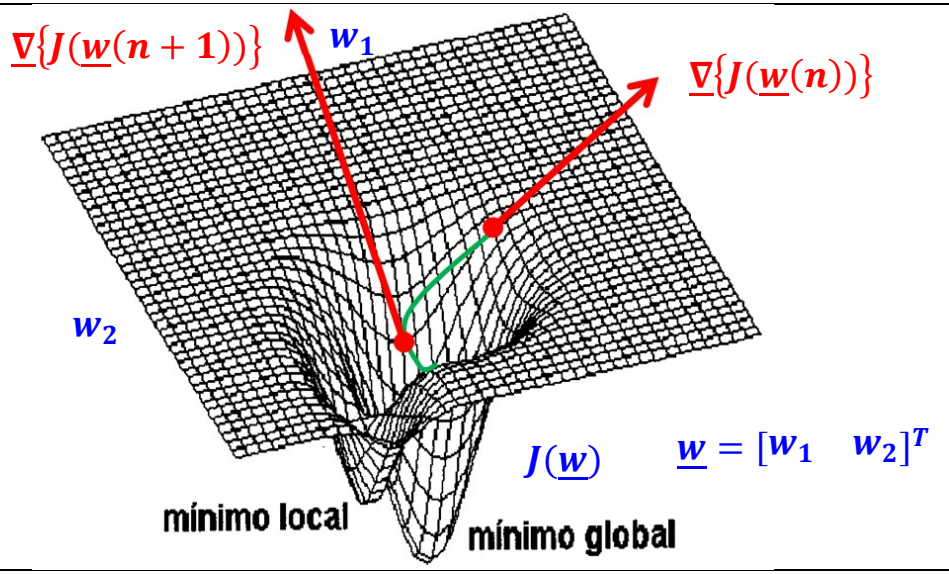
Regra Delta

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

Regra Delta com Fator de Momento

$$\Delta w_{ji}(n) = \alpha \Delta w_{ji}(n-1) + \eta \delta_j(n) y_i(n)$$

onde a constante α é denominada Constante de Momento, $0 < \alpha < 1$.



O efeito do Fator de Momento é aumentar a velocidade da trajetória no espaço de pesos, na direção da descida mais íngreme:

Regra Delta com Fator de Momento → $\Delta w_{ji}(n) = \alpha \Delta w_{ji}(n-1) + \eta \delta_j(n) y_i(n)$, $0 < \alpha < 1$

Se a correção aplicada em determinado peso sináptico mantém o mesmo sinal algébrico durante várias iterações consecutivas, situação que ocorre quando a trajetória na superfície de erro desenrola-se ao longo de um caminho em descida íngreme, **a correção do peso sináptico é acelerada pelo fator de momento**, já que, sendo o caminho uma descida íngreme, o mínimo deve estar longe ainda. Um eventual mínimo local encontrado ao longo desta descida acelerada pode, então, ser facilmente transpassado.

Isto ocorre porque, imaginando que a trajetória das coordenadas do vetor de pesos sinápticos $\underline{W}_j$ de um neurônio *j* qualquer seja a trajetória de um móvel de grande massa descendo uma ladeira irregular (i.e., com vários mínimos locais), em consequência do alto momento de inércia (energia cinética) do móvel devido à sua massa, as irregularidades (mínimos locais) não conseguem parar o movimento do móvel.

Se a correção aplicada em determinado peso sináptico troca o sinal algébrico durante várias iterações consecutivas, situação que se espera ocorrer quando a trajetória na superfície de erro desenrola-se ao longo de um caminho próximo ao mínimo global, **a correção do peso sináptico é freada pela redução do valor absoluto médio do fator de momento acrescentado**, já que um mínimo (provavelmente global) está próximo e uma alta velocidade poderia desestabilizar o algoritmo em torno do mínimo.

Sumário do Algoritmo Backprop e Recomendações/Interpretações Operacionais

I - Inicialização:

1. Define-se o número de camadas da MLP.

- Em geral, sob o ponto de vista de rapidez de redução do MSE, é preferível utilizar poucas camadas escondidas com muitos neurônios por camada do que muitas camadas escondidas com poucos neurônios por camada.
- Isto porque o uso de muitas camadas escondidas "dilui" o efeito corretivo da retro-propagação dos sinais de erro sobre as sinapses ao longo do *backward pass*.
- Em consequência, o MLP demorará mais Épocas para atingir um MSE suficientemente baixo.
- Por outro lado, um número maior de camadas escondidas habilita o MLP a captar melhor as estatísticas de ordem superior do processo a ser aprendido, melhorando, assim, a capacidade de generalização do MLP.
- Isto ocorre porque um maior número de camadas escondidas torna o mapeamento $\Re^{m_1} \rightarrow \Re^{m_L}$ realizado pela MLP, um mapeamento com "maior não-linearidade recursiva", sendo m_1 e m_L respectivamente o número de nós de entrada e saída da MLP.
- A informação sobre o processo a ser aprendido pela MLP fica armazenada nas sinapses dos neurônios de cada camada, e as saídas de cada camada recursivamente alimentam as entradas da camada seguinte durante a fase de treino.
- Cada camada executa uma operação não-linear devido à função de ativação, portanto, à medida que uma camada alimenta a seguinte, uma nova instância da operação não-linear é efetuada.

- A operação não-linear efetuada pela função de ativação é definida pela função exponencial e^x (ou por uma combinação de exponenciais no caso da Tangente Hiperbólica) , sendo e^x passível de ser expandida na série de potências

$$e^x = 1 + x + \frac{1}{2}\cdot x^2 + \frac{1}{6}\cdot x^3 + \frac{1}{24}\cdot x^4 + \frac{1}{120}\cdot x^5 + \ldots .$$

- Ora, como a informação é recursivamente acumulada nas sinapses do MLP, sendo processada através de várias instâncias recursivas de uma série de potências durante o treino, fica implícito que o MLP acumula informação na forma de "estruturas de correlação estatística de ordem superior".
- Isto é, após a fase de treino do MLP, a informação armazenada no conjunto de sinapses está associada à $E\{\underline{x}_i \otimes \underline{x}_j\} + E\{\underline{x}_i \otimes \underline{x}_j \otimes \underline{x}_k\} + E\{\underline{x}_i \otimes \underline{x}_j \otimes \underline{x}_k \otimes \underline{x}_l\} + \cdots$ onde:
  - ♦ $\underline{x}_i$, $\underline{x}_j$, ... representam individualmente todos os possíveis *N* vetores existentes no conjunto de treino;
  - ♦ $E\{\cdot\}$ é o operador média estatística;
  - ♦ $\underline{x}_i \otimes \underline{x}_j$ representa a matriz $m \times m$ formada pelos m^2 produtos entre os m componentes do vetor $\underline{x}_i$ pelos *m* componentes do vetor $\underline{x}_j$, isto é, $\underline{x}_i \otimes \underline{x}_j = \underline{x}_i \underline{x}_j^T$;
  - ♦ $\underline{x}_i \otimes \underline{x}_j \otimes \underline{x}_k$ representa a estrutura cúbica em $\Re^3$ formada pelos m^3 produtos entre os m^2 elementos da matriz $\underline{x}_i \otimes \underline{x}_j$ e os *m* componentes do vetor $\underline{x}_k$;
  - ♦ e assim sucessivamente.

2.	Subtrai-se o vetor média do conjunto de N vetores de treino.
3.	Normaliza-se a i-ésima componente de cada vetor de treino pelo desvio padrão do conjunto de N valores formado pela i-ésima componente de todos os N vetores de treino.
4.	Normaliza-se o conjunto de N saídas desejadas para o intervalo [-1,+1].
5.	Definem-se os parâmetros a e b da função de ativação. Em geral, $a = 1.7159$ e $b = 2/3$ são valores adequados para $\varphi(v) = a\tanh(bv)$, de modo que $\varphi'(0) = ab = 1.14 \approx 1$.
6.	Inicializam-se os pesos sinápticos com valores aleatórios de distribuição uniforme. • Uma possível heurística é adotar uma inicialização randômica com valores compreendidos no intervalo [-2.4/F_i , +2.4/F_i], onde F_i é o *fan-in* ou o número total de nós de entrada (sinapses) do neurônio. • Outra possível heurística é adotar uma inicialização randômica com conjunto de valores de média zero e variância definida por $1/F_i$.
7.	Definem-se o fator de momento ($0<\alpha<1$) e a razão de aprendizado ($0<\eta<1$) por camada do MLP. • Visto que os neurônios próximos da camada de saída tendem a ter maiores gradientes locais, atribui-se a eles usualmente razões de aprendizado menores. • Outro critério a ser considerado simultaneamente é que neurônios com muitas entradas devem ter η menores.

II - Treinamento:	
1.	Apresenta-se cada exemplo (vetor de entrada) do conjunto de treino aos nós de entrada do MLP. • Seja $\Gamma : \Re^{m_1} \rightarrow \Re^{m_L}$ o mapeamento ou processo a ser aprendido pelo MLP; onde m_1 e m_L representam respectivamente o número de nós de entrada e saída do MLP. • O conjunto de treino deve conter uma parcela suficientemente significativa do universo de vetores-exemplo que descrevem o processo Γ. Caso esta recomendação não seja atendida, após o treino, o MLP não terá condições de inferir um resultado correto quando a ele for apresentado um vetor de Γ que não pertencia ao conjunto de treino, o que denota **incapacidade de generalização do MLP** após o treino realizado.
2.	Para cada exemplo executa-se completamente um ciclo passo direto - passo reverso, mantendo-se o vetor de entrada aplicado à entrada do MLP.
3.	O final da apresentação de todos os exemplos do conjunto de treino define uma Época. A cada determinado número de Épocas em que for observada uma significativa queda no MSE, aumenta-se o momento α e/ou a razão de aprendizado η.
4.	Prossegue-se o treino do MLP de Época em Época, eventualmente ajustando α e η, até que se atinja o Critério de Parada.

III - Critério de Parada:

O critério de parada no treino de uma rede MLP é subjetivo, já que não existe prova de que o algoritmo *backpropagation* tenha convergido para o mínimo global da superfície de erro (se é que existe o mínimo global).

Sugere-se como critério de parada o seguinte procedimento:

As iterações de treino são terminadas se:

1- O valor do MSE atingiu um valor suficientemente baixo e/ou

2- A razão de variação do MSE atingiu um valor suficientemente baixo em valor absoluto e negativo.

- Quando qualquer uma das condições acima é atingida, considera-se que o MLP não necessita mais ser treinado.
- Note que o critério 2 pode significar que o *backpropagation* ficou preso em um mínimo local e não global.

- É importante observar que um MSE baixo ao final do treino não necessariamente implica em uma alta capacidade de generalização.
- Se o conjunto de treino escolhido para representar o processo Γ a ser aprendido pelo MLP constituir um sub-conjunto cujas propriedades estatísticas não são suficientemente representativas das propriedades estatísticas de Γ, então o MLP falhará em inferir o resultado correto quando um vetor de Γ que não pertença ao conjunto de treino for apresentado à RNA MLP.

Apêndice A - A Derivada da Função de Ativação

A determinação do gradiente local para cada neurônio da RNA MLP requer o conhecimento da derivada $\varphi'(\cdot)$ da função ativação $\varphi(\cdot)$ associada com o neurônio, conforme se infere da expressão para o gradiente local, (discutido em seção anterior, e cuja dedução encontra-se no Apêndice B deste capítulo):

$$\delta_j(n) = \begin{cases} \varphi'_j(v_j(n))e_j(n) & \text{, neurônio } j \text{ é de saída} \\ \varphi'_j(v_j(n))\sum_k \delta_k(n)w_{kj}(n) & \text{, neurônio } j \text{ é escondido} \end{cases}$$

- Para que esta derivada exista, é necessário que a função de ativação $\varphi(\cdot)$ seja contínua.
- Uma função de ativação não-linear continuamente diferenciável, comumente aplicada em redes MLP é a função sigmoidal, já descrita quando estudamos tipos de funções de ativação.

Duas formas da função sigmoidal são usuais: A Função Logística e a Função Tangente Hiperbólica.

a) Função Logística

Esta forma de não-linearidade sigmoidal é definida por:

$$\varphi_j(v_j(n)) = \frac{1}{1+\exp(-a v_j(n))}, \quad a > 0 \text{ e } -\infty < v_j(n) < \infty \qquad (1)$$

onde $v_j(n)$ é o potencial de ativação do neurônio j.

De acordo com esta não-linearidade, a amplitude da saída fica restrita ao intervalo $0 \le y_j \le 1$.

Omitindo os índices *n* e *j* por simplicidade, e diferenciando a função de ativação expressa em (1) com respeito a $v_j(n)$, temos

$$\varphi'(v) = \frac{d}{dv}\varphi(v) = \frac{d}{dv}\left\{\frac{1}{1+\exp(-av)}\right\} = \frac{a\ \exp(-av)}{[1+\exp(-av)]^2} = \qquad (2)$$

$$= a\varphi^2(v)\left(\frac{1}{\varphi(v)} - 1\right) = a\varphi(v)(1-\varphi(v))$$

E, como $y_j(n) = \varphi(v_j(n))$,

$$\rightarrow \varphi'(v_j(n)) = \frac{d}{dv}\varphi(v_j(n)) = a\, y_j(n)[1 - y_j(n)] \qquad (3)$$

Note na Eq. (3) que a derivada atinge valor máximo em $y_j(n) = 0.5$, e valor mínimo (=zero) em $y_j(n) = 0$, ou $y_j(n) = 1.0$.

Já que a mudança em um peso sináptico da RNA MLP é proporcional à derivada, segue que, para uma função de ativação sigmoidal, os pesos sinápticos sofrem a maior alteração para aqueles neurônios onde os sinais assumem valores no meio de seu intervalo de variação. Esta é uma característica que contribui para a estabilidade do algoritmo de aprendizagem.

b) Função Tangente HiperbólicaEsta forma de não-linearidade sigmoidal é definida por:

$$\varphi_j(v_j(n)) = a\tanh(bv_j(n)) = a\left\{\frac{1-\exp(-2bv_j(n))}{1+\exp(-2bv_j(n))}\right\}, \quad a,b>0 \qquad (2)$$

De acordo com esta não-linearidade, a amplitude da saída fica restrita ao intervalo $-a \le y_j \le a$.

Omitindo os índices *n* e *j* por simplicidade, a derivada da função ativação pode ser obtida através de

$$\varphi'(v) = \frac{d}{dv}\varphi(v) = \frac{d}{dv}a\tanh(bv) = ab\,\mathrm{sech}^2(bv) = ab\left(1-\tanh^2(bv)\right) = \qquad (4)$$

$$= ab\left(1-\left(\frac{a\tanh(bv)}{a}\right)^2\right) = ab\left(1-\frac{(a\tanh(bv))^2}{a^2}\right) = ab\left(1-\frac{\varphi^2(v)}{a^2}\right) =$$

$$= ab\left(1-\frac{y^2}{a^2}\right) = ab\left(\frac{a^2-y^2}{a^2}\right) = \frac{b}{a}\left(a^2-y^2\right) = \frac{b}{a}(a+y)(a-y)$$

Portanto,

$$\rightarrow \varphi'(v_j(n)) = \frac{d}{dv}\varphi(v_j(n)) = \frac{b}{a}(a+y_j(n))(a-y_j(n)). \qquad (5)$$

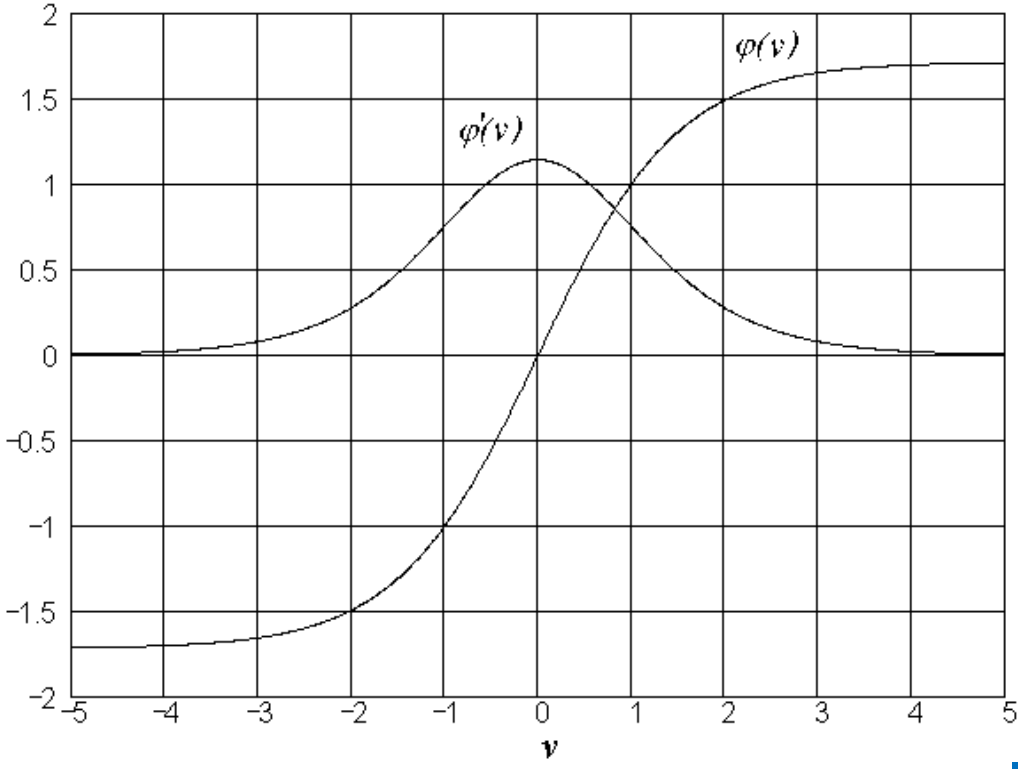


Gráfico de $\varphi(v) = a\tanh(bv)$ e $\varphi'(v) = ab\left(1 - \tanh^2(bv)\right)$

p/ $a = 1.7159$ e $b = 2/3$.

Observe que, ao utilizarmos a Equação (3) como derivada da função logística e a Equação (5) como derivada da função tangente hiperbólica, o gradiente local δ_j pode ser calculado sem o uso explícito da definição analítica da função de ativação:

$$\varphi'\left(v_j(n)\right) = \frac{d}{dv}\varphi\left(v_j(n)\right) = a\, y_j(n)\left[1 - y_j(n)\right] \qquad (3)$$

$$\varphi'\left(v_j(n)\right) = \frac{d}{dv}\varphi\left(v_j(n)\right) = \frac{b}{a}\left(a + y_j(n)\right)\left(a - y_j(n)\right) \qquad (4)$$

$$\delta_j(n) = \begin{cases} \varphi'_j\left(v_j(n)\right) e_j(n), & \textit{neurônio } j \textit{ é de saída} \\ \varphi'_j\left(v_j(n)\right) \sum_k \delta_k(n) w_{kj}(n), & \textit{neurônio } j \textit{ é escondido} \end{cases}$$

Apêndice B - Determinação da Expressão do Gradiente Local do Algoritmo *Backpropagation*

- O algoritmo *backpropagation* aplica a correção $\Delta w_{ji}(n)$ ao peso sináptico $w_{ji}(n)$, tendo como base a direção contrária do gradiente local $\frac{\partial \varepsilon(n)}{\partial w_{ji}(n)}$ da superfície de erro $\varepsilon(w)$ relativo ao peso sináptico.

- Em última análise, o gradiente local $\frac{\partial \varepsilon(n)}{\partial w_{ji}(n)}$ representa a variação resultante $\Delta\varepsilon(w)$ no erro quadrático instantâneo $\varepsilon(w)$ da RNA MLP quando é aplicada uma perturbação (variação) infinitesimal ao peso sináptico que liga a saída do neurônio i ao i-ésimo nó de entrada do neurônio j.

De acordo com a regra da cadeia do cálculo diferencial, o gradiente local $\frac{\partial \varepsilon(n)}{\partial w_{ji}(n)}$ pode ser expresso por

$$\frac{\partial \varepsilon(n)}{\partial w_{ji}(n)} = \frac{\partial \varepsilon(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{ji}(n)} \qquad (1)$$

Passemos a determinar, agora, cada uma das derivadas parciais em (1).

$$\frac{\partial \varepsilon(n)}{\partial w_{ji}(n)} = \frac{\partial \varepsilon(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{ji}(n)} \qquad (1)$$

Para determinar $\frac{\partial \varepsilon(n)}{\partial e_j(n)}$, partiremos de

$$\frac{1}{2}\left\{e_0{}^2(n) + e_1{}^2(n) + \cdots + e_j{}^2(n) + \cdots + e_{m_L-1}{}^2(n)\right\} \qquad (2)$$

onde m_L é o número de neurônios da camada de saída.

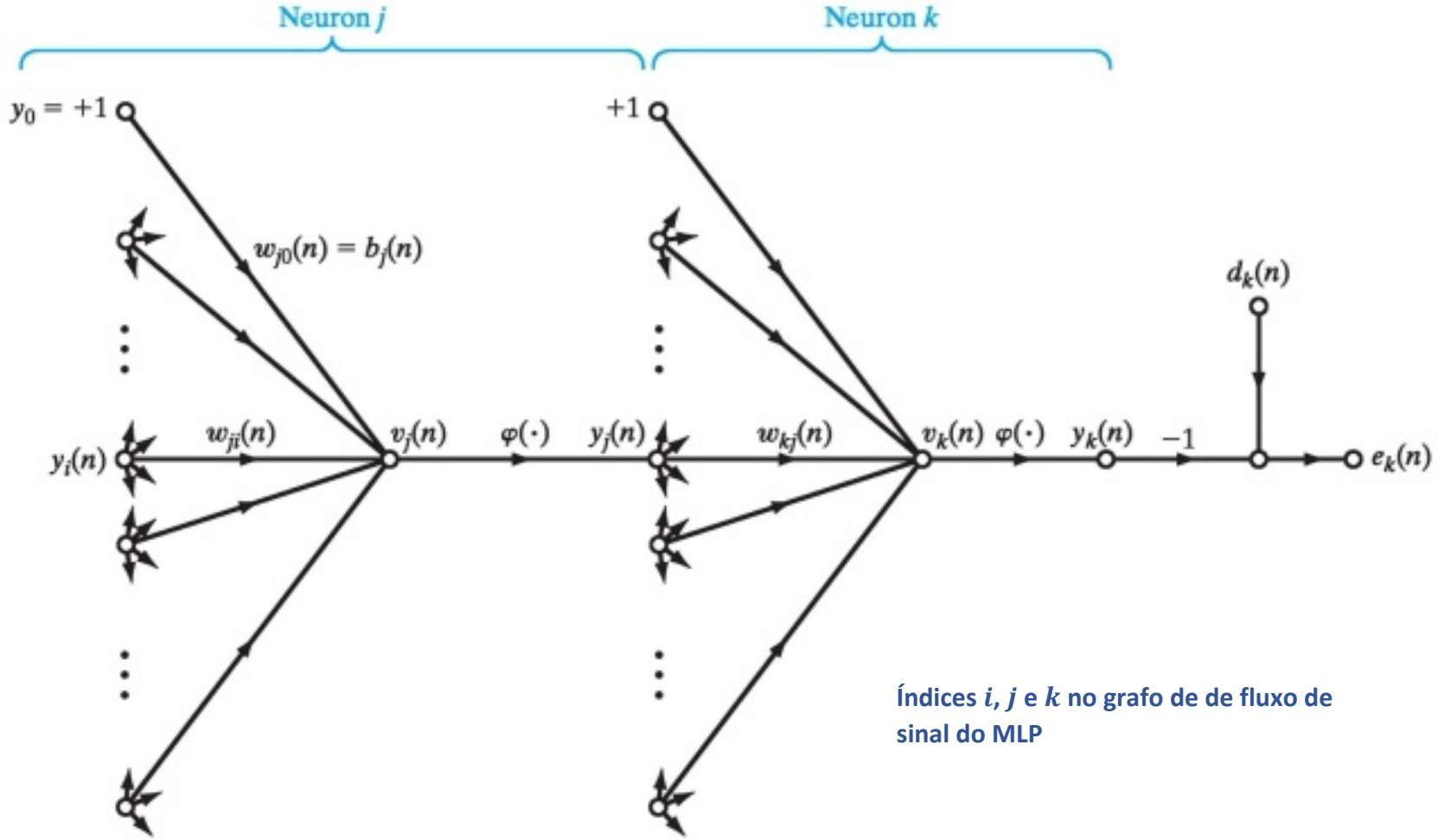
Diferenciando (2) em relação à $e_j(n)$ temos

$$\frac{\partial \varepsilon(n)}{\partial e_j(n)} = \frac{1}{2}\frac{\partial}{\partial e_j(n)}\left\{e_0{}^2(n) + e_1{}^2(n) + \cdots + e_j{}^2(n) + \cdots + e_{m_L-1}{}^2(n)\right\} = e_j(n). \qquad (3)$$

Determinando a próxima derivada parcial em (1):

$$\frac{\partial \varepsilon(n)}{\partial w_{ji}(n)} = \frac{\partial \varepsilon(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{ji}(n)} \qquad (1)$$

Para determinar $\frac{\partial e_j(n)}{\partial y_j(n)}$, derivemos $e_j(n) = d_j(n) - y_j(n)$ em relação à $y_j(n)$, obtendo $\frac{\partial e_j(n)}{\partial y_j(n)} = -1$ (4)



Índices *i*, *j* e *k* no grafo de de fluxo de sinal do MLP

Determinando a próxima derivada parcial em (1):

$$\frac{\partial \varepsilon(n)}{\partial w_{ji}(n)} = \frac{\partial \varepsilon(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{ji}(n)} \qquad (1)$$

Para determinar $\frac{\partial y_j(n)}{\partial v_j(n)}$, derivemos $y_j(n) = \varphi_j\left(v_j(n)\right)$ em relação à $v_j(n)$, obtendo

$$\frac{\partial y_j(n)}{\partial v_j(n)} = \varphi'_j\left(v_j(n)\right) \qquad (5)$$

resultado já determinado anteriormente, e que depende do tipo de função de ativação utilizada na RNA MLP, conforme:

Derivada da função logística → $\varphi'\left(v_j(n)\right) = \frac{d}{dv}\varphi\left(v_j(n)\right) = a\, y_j(n)\left[1 - y_j(n)\right]$

Derivada da função tangente hiperbólica → $\varphi'\left(v_j(n)\right) = \frac{d}{dv}\varphi\left(v_j(n)\right) = \frac{b}{a}\left(a + y_j(n)\right)\left(a - y_j(n)\right)$

Determinando a próxima derivada parcial em (1):

$$\frac{\partial \varepsilon(n)}{\partial w_{ji}(n)} = \frac{\partial \varepsilon(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{ji}(n)} \qquad (1)$$

Para determinar $\frac{\partial v_j(n)}{\partial w_{ji}(n)}$, partimos de $v_j(n) = \sum_{i=0}^{m} w_{ji}(n) y_i(n)$, onde temos

$$v_j(n) = \sum_{i=0}^{m} w_{ji}(n) y_i(n) = \left\{ \begin{matrix} w_{j0}(n) y_0(n) + w_{j1}(n) y_1(n) + \cdots \\ \cdots + w_{ji}(n) y_i(n) + \cdots + w_{jm}(n) y_m(n) \end{matrix} \right\} \qquad (6)$$

Diferenciando (6) em relação à $w_{ji}(n)$, obtemos

$$\frac{\partial v_j(n)}{\partial w_{ji}(n)} = \frac{\partial}{w_{ji}(n)} \left\{ \begin{matrix} w_{j0}(n) y_0(n) + w_{j1}(n) y_1(n) + \cdots \\ \cdots + w_{ji}(n) y_i(n) + \cdots + w_{jm}(n) y_m(n) \end{matrix} \right\} = y_i(n) \qquad (7)$$

Substituindo os resultados das derivadas parciais recém obtidos em

$$\frac{\partial \varepsilon(n)}{\partial w_{ji}(n)} = \frac{\partial \varepsilon(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{ji}(n)},$$ temos

$$\frac{\partial \varepsilon(n)}{\partial w_{ji}(n)} = \frac{\partial \varepsilon(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{ji}(n)} = e_j(n)(-1)\varphi_j'(v_j(n))y_i(n) \quad (8)$$

Substituindo o resultado de (8) em $\Delta w_{ji}(n) = w_{ji}(n+1) - w_{ji}(n) = -\eta \frac{\partial \varepsilon(n)}{\partial w_{ji}(n)}$, que expressa a correção $\Delta w_{ji}(n)$ aplicada a $w_{ji}(n)$ ditada pela Regra Delta para a RNA MLP, obtemos

$$\Delta w_{ji}(n) = w_{ji}(n+1) - w_{ji}(n) = \eta e_j(n)\varphi_j'(v_j(n))y_i(n) \quad (9)$$

Note de $\frac{\partial \varepsilon(n)}{\partial w_{ji}(n)} = \frac{\partial \varepsilon(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{ji}(n)} = e_j(n)(-1)\varphi_j'(v_j(n))y_i(n)$ (8)

que o termo $e_j(n)\varphi_j'(v_j(n))$ em $\Delta w_{ji}(n) = w_{ji}(n+1) - w_{ji}(n) = \eta e_j(n)\varphi_j'(v_j(n))y_i(n)$ (9)

origina-se da cadeia de operações

$$\frac{\partial \varepsilon(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} = e_j(n)(-1)\varphi_j'(v_j(n)) \quad \text{(10)}$$

, isto é,

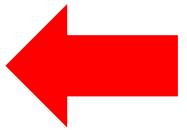
$$\frac{\partial \varepsilon(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} = -e_j(n)\varphi_j'(v_j(n)) \quad \text{(11)}$$

ou, ainda, simplificando os diferenciais intermediários em (11), obtemos o denominado **gradiente local *j***,

$$\delta_j(n) = -\frac{\partial \varepsilon(n)}{\partial v_j(n)} = e_j(n)\varphi_j'(v_j(n)) \quad \text{(12)}$$

A equação (12) é a equação para o Gradiente Local $\delta_j(n)$ quando *j* é neurônio de saída:

$$\delta_j(n) = \begin{cases} \varphi_j'(v_j(n))e_j(n) & \text{, neurônio } j \text{ é de saída} \\ \varphi_j'(v_j(n))\sum_k \delta_k(n)w_{kj}(n) & \text{, neurônio } j \text{ é escondido} \end{cases}$$



Substituindo $\delta_j(n) = -\frac{\partial \varepsilon(n)}{\partial v_j(n)} = e_j(n)\varphi'_j(v_j(n))$ (12)

em $\Delta w_{ji}(n) = w_{ji}(n+1) - w_{ji}(n) = \eta e_j(n)\varphi'_j(v_j(n))y_i(n)$ (9)

obtemos $\Delta w_{ji}(n) = w_{ji}(n+1) - w_{ji}(n) = \eta \delta_j(n) y_i(n)$ (13)

que é a Regra Delta $w_{ji}(n+1) = w_{ji}(n) - \eta \frac{\partial \varepsilon(n)}{\partial w_{ji}(n)}$ aplicada ao *backprop*, mas com o gradiente local

genérico $\frac{\partial \varepsilon(n)}{\partial w_{ji}(n)}$ agora substituído pela definição do gradiente local

$\delta_j(n) = -\frac{\partial \varepsilon(n)}{\partial v_j(n)} = e_j(n)\varphi'_j(v_j(n))$ do *j*-ésimo neurônio pertencente à camada de saída.

Passamos agora a deduzir a expressão do gradiente local $\delta_j(n) = \varphi'_j(v_j(n))\sum_k \delta_k(n) w_{kj}(n)$, válida quando o *j*-ésimo neurônio pertence a uma das camadas escondidas, situação em que não existe um erro $e_j(n)$ explícito associado ao neurônio.

Para obtermos a expressão do gradiente local $\delta_j(n) = \varphi'_j(v_j(n))\sum_k \delta_k(n) w_{kj}(n)$, válida quando o *j*-ésimo neurônio pertence a uma das camadas escondidas, situação em que não existe um erro $e_j(n)$ explícito associado ao neurônio, vamos inicialmente expandir a Equação (12), conforme

$$\delta_j(n) = -\frac{\partial \varepsilon(n)}{\partial v_j(n)} = e_j(n)\varphi'_j(v_j(n)) = -\frac{\partial \varepsilon(n)}{\partial y_j(n)}\frac{\partial y_j(n)}{\partial v_j(n)} = -\frac{\partial \varepsilon(n)}{\partial y_j(n)}\varphi'_j(v_j(n)) \qquad (14)$$

A seguir, reescrevendo a Equação (2) de forma que o índice do somatório seja substituído por *k*, para caracterizar que refere-se a erros quadráticos de neurônios da camada de saída (isto é feito para evitar confusão com neurônios da camada escondida imediatamente à esquerda da camada de saída, os quais, segundo a convenção aqui adotada, devem ser indexados por *j*), temos:

$$\varepsilon(n) = \frac{1}{2}\sum_{j=0}^{m_L-1} e_j^{\,2}(n) \qquad (2)$$

$$\varepsilon(n) = \frac{1}{2}\sum_{k=0}^{m_L-1} e_k^{\,2}(n) = \frac{1}{2}\sum_k e_k^{\,2}(n) \qquad (15)$$

$$\varepsilon(n) = \frac{1}{2}\sum_{k=0}^{m_L-1} e_k^{\,2}(n) = \frac{1}{2}\sum_{k} e_k^{\,2}(n) \tag{15}$$

Desejamos determinar o gradiente local $\delta_j(n)$ de neurônios pertencentes a camadas escondidas, onde não existe um erro explícito associado a cada neurônio. Diferenciando (15) em relação a y_j temos

$$\frac{\partial \varepsilon(n)}{\partial y_j(n)} = \frac{1}{2}\frac{\partial}{\partial y_j(n)}\sum_{k} e_k^{\,2}(n) = \sum_{k} e_k(n)\frac{\partial e_k(n)}{\partial y_j(n)} \tag{16}$$

que pode ser reescrita como

$$\frac{\partial \varepsilon(n)}{\partial y_j(n)} = \sum_{k} e_k(n)\frac{\partial e_k(n)}{\partial y_j(n)} = \sum_{k} e_k(n)\frac{\partial e_k(n)}{\partial v_k}\frac{\partial v_k}{\partial y_j(n)} \tag{17}$$

Para o neurônio *k* na camada de saída, $e_k(n)$ é dado por

$$e_k(n) = d_k(n) - y_k(n) = d_k(n) - \varphi_k(v_k(n)) \tag{18}$$

e, portanto, diferenciando (18) em relação à $v_k(n)$, temos

$$\frac{\partial e_k(n)}{\partial v_k(n)} = -\varphi_k'(v_k(n)) \tag{19}$$

O potencial de ativação v_k para um neurônio k situado na camada à direita da camada j onde estão situados os m neurônios de índice j a ele conectado é dado por

$$v_k(n) = \sum_{j=0}^{m} w_{kj}(n) y_j(n) = \begin{Bmatrix} w_{k0}(n) y_0(n) + w_{k1}(n) y_1(n) + \cdots \\ \cdots + w_{kj}(n) y_j(n) + \cdots + w_{km}(n) y_m(n) \end{Bmatrix} \tag{20}$$

onde m é o número total de entradas aplicadas ao neurônio k.

Diferenciando (20) em relação a $y_j(n)$ temos

$$\frac{\partial v_k(n)}{\partial y_j(n)} = \frac{\partial}{\partial y_j(n)} \begin{Bmatrix} w_{k0}(n) y_0(n) + w_{k1}(n) y_1(n) + \cdots \\ \cdots + w_{kj}(n) y_j(n) + \cdots + w_{km}(n) y_m(n) \end{Bmatrix} = w_{kj}(n) \tag{21}$$

Substituindo (19) e (21) em (17) obtemos

$$\frac{\partial \varepsilon(n)}{\partial y_j(n)} = \sum_k e_k(n) \frac{\partial e_k(n)}{\partial v_k} \frac{\partial v_k}{\partial y_j(n)} = -\sum_k e_k(n) \varphi_k'(v_k(n)) w_{kj}(n) \tag{22}$$

Consideremos o gradiente local p/ o neurônio j situado na camada j dado por (14), abaixo reescrita por comodidade de visualização:

$$\delta_j(n) = -\frac{\partial \varepsilon(n)}{\partial v_j(n)} = e_j(n)\varphi'_j\left(v_j(n)\right) = -\frac{\partial \varepsilon(n)}{\partial y_j(n)}\frac{\partial y_j(n)}{\partial v_j(n)} = -\frac{\partial \varepsilon(n)}{\partial y_j(n)}\varphi'_j\left(v_j(n)\right) \tag{14}$$

Podemos substituir j por k em (14) para que ela represente o gradiente local de um neurônio na camada k à direita da camada j:

$$\delta_k(n) = e_k(n)\varphi'_k\left(v_k(n)\right) \tag{23}$$

Substituindo (23) em (22)

$$\frac{\partial \varepsilon(n)}{\partial y_j(n)} = -\sum_k \delta_k(n) w_{kj}(n) \tag{24}$$

Substituindo (24) em (14) resulta no gradiente local $\delta_j(n)$ de neurônios pertencentes a camadas escondidas, onde não existe um erro explícito associado a cada neurônio:

$$\delta_j(n) = -\frac{\partial \varepsilon(n)}{\partial v_j(n)} = -\frac{\partial \varepsilon(n)}{\partial y_j(n)}\varphi'_j\left(v_j(n)\right) = \varphi'_j\left(v_j(n)\right)\sum_k \delta_k(n) w_{kj}(n) \tag{25}$$

Portanto, o gradiente local $\delta_j(n)$ é dado por

$$\delta_j(n) = \begin{cases} \varphi'_j\left(v_j(n)\right)e_j(n) & \text{, neurônio } \boldsymbol{j} \text{ é de saída} \\ \varphi'_j\left(v_j(n)\right)\sum_k \delta_k(n) w_{kj}(n) & \text{, neurônio } \boldsymbol{j} \text{ é escondido} \end{cases}$$

Referências:

[1] ARTIFICIAL NEURAL NETWORK AND WAVELET DECOMPOSITION IN THE FORECAST OF GLOBAL HORIZONTAL SOLAR RADIATION. Luiz Albino Teixeira Júnior, Rafael Morais de Souza, Moisés Lima de Menezes, Keila Mara Cassiano, José Francisco Moreira Pessanha, Reinaldo Castro Souza, Disponível em: http://dx.doi.org/10.1590/0101-7438.2015.035.01.0073

[2] Fundamentals of Deep Learning – Starting with Artificial Neural Network, AARSHAY JAIN , MARCH 16, 2016, disponível em https://www.analyticsvidhya.com/blog/2016/03/introduction-deep-learning-fundamentals-neural-networks/

[3] Sistema Informatizado para Diagnosticar Doenças Fúngicas na Cultura do Tomate. Felipe dos Santos Vieira, Rafael Paz. 2013.

[4] Neural networks and learning machines - Simon Haykin, 3rd - Prentice Hall - 2009

[5] Todas as demais figuras e conteúdos seguem as referências presentes na apostila de Redes Neurais Artificiais, disponível em http://www.fccdecastro.com.br/download.html.