UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO
Departamento de Ciências de Computação

SCC-5809 - Capítulo 5
Perceptron Multicamadas

João Luís Garcia Rosa[1]

[1]SCC-ICMC-USP - joaoluis@icmc.usp.br

2011

Sumário

Sumário

O perceptron

- Como já visto, o primeiro modelo matemático do neurônio foi proposto por McCulloch & Pitts em 1943 [3].
- Mais tarde em 1957, Rosenblatt [5] criou o modelo do perceptron.
- Um perceptron modela um neurônio tomando uma soma ponderada de suas entradas e enviando a saída 1 (*spike*) se esta soma é maior que um determinado limiar de ativação.
- O perceptron, com função de ativação linear, pode ser modelado como um discriminador linear:
  - dados 2 pontos, uma reta é capaz de discriminar esses 2 pontos,
  - para algumas configurações de m pontos, uma reta é capaz de separar estes pontos em 2 classes.

Limitações do perceptron de camada única

- O perceptron é uma rede *feedforward* (não recorrente) de uma única camada.
- O perceptron só é capaz de aprender a solução de problemas linearmente separáveis.
- O algoritmo de aprendizagem do perceptron (regra delta) não funciona com redes de mais de uma camada.

Sumário

Mais camadas

- Redes Neurais alimentadas à frente com múltiplas camadas:
  - Um conjunto de unidades sensoriais (nós fonte): *camada de entrada*.
  - Uma ou mais *camadas escondidas* de nós computacionais.
  - Uma *camada de saída* de nós computacionais.
  - O sinal de entrada se propaga através da rede numa direção às frente, camada por camada.
  - Essas redes são conhecidas como perceptrons multicamadas (MLP), uma generalização do perceptron de camada única.

Perceptrons Multicamadas

- MLP têm sido aplicados com sucesso para resolver diversos e difíceis problemas.
- Treinados de forma supervisionada
- Algoritmo muito popular: *error back-propagation* [1, 6], baseado na regra de aprendizado de correção de erro:
  - Generalização do algoritmo do filtro adaptativo: LMS.
  - Duas fases: um passo à frente e um passo para trás.
  - No passo à frente, um padrão de atividade (vetor de entrada) é aplicado aos nós sensoriais da rede e seu efeito é propagado através da rede camada por camada.
  - Por fim, um conjunto de saídas é produzido como resposta real da rede.

Perceptrons Multicamadas

- BP:
  - Durante o passo à frente, os pesos sinápticos são *fixos*.
  - Durante o passo para trás, os pesos sinápticos são *ajustados* de acordo com uma regra de correção de erro.
  - Especificamente, a resposta real é subtraída da resposta desejada para produzir um *sinal de erro*.
  - Este sinal de erro é propagado de volta através da rede, em sentido contrário aos estímulos.
  - Os pesos são ajustados para fazer a resposta real se aproximar da resposta desejada, de uma forma estatística.

Características do MLP

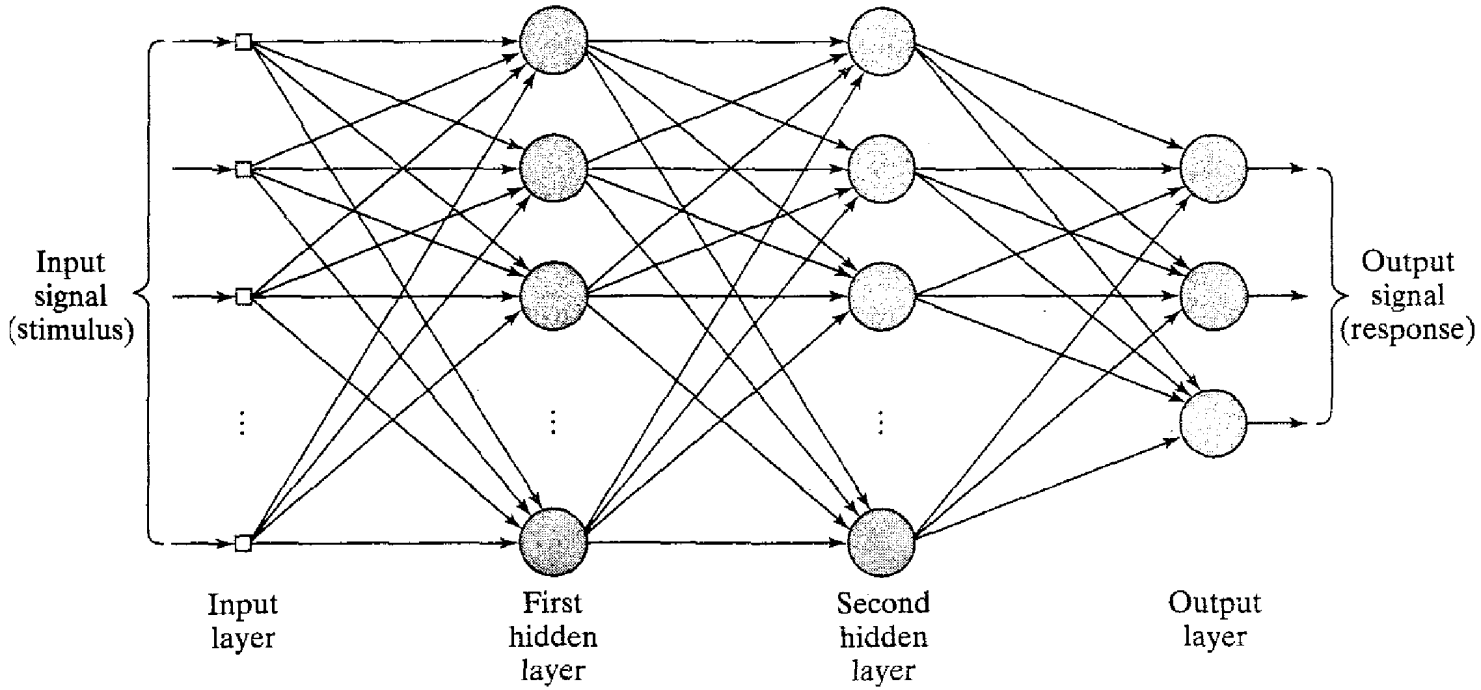
- Características do MLP:
  1. O modelo de cada neurônio da rede inclui uma *função de ativação não-linear*, a não-linearidade sigmoidal definida pela função logística:

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

     onde v_j é o campo local induzido (isto é, a soma ponderada das entradas mais o *bias*) do neurônio j e y_j é a saída.
  2. A rede contém uma ou mais camadas de *neurônios escondidos*, que não são parte da entrada nem da saída.
  3. A rede exibe um alto grau de *conectividade*, determinado pelas sinapses.

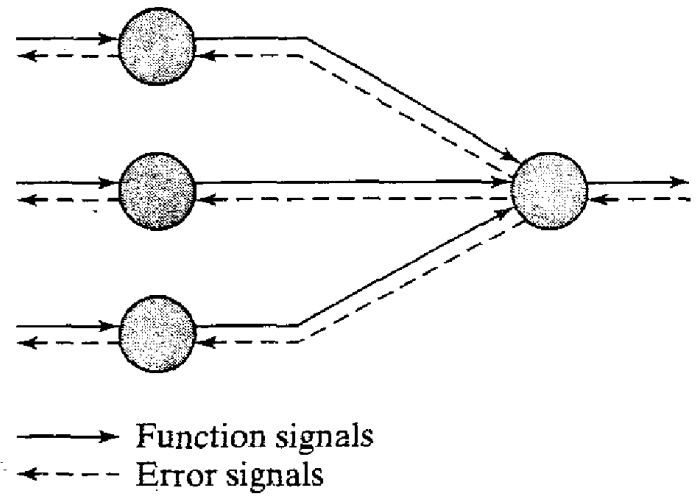
MLP

- A figura 6 abaixo [2] mostra um grafo arquitetural de um perceptron multi-camadas com duas camadas escondidas e uma camada de saída.
- Esta rede é *totalmente conectada*.



MLP

- A figura 7 abaixo [2] mostra uma porção de um MLP.
- Dois tipos de sinais são identificados:
  1. *Sinais de função*: sinal de entrada (estímulo) que chega na entrada, propaga-se para frente (neurônio por neurônio) através da rede e emerge na saída da rede como um sinal de saída. Também conhecido como *sinais de entrada*.
  2. *Sinais de erro*: origina em um neurônio de saída da rede e propaga-se de volta (camada por camada) através da rede.



MLP

- Cada neurônio escondido ou de saída de um MLP realiza duas computações:
  1. Sinal de função que aparece na saída de um neurônio, expresso como um função não-linear contínua do sinal de entrada e pesos associados.
  2. Estimativa do vetor gradiente (isto é, os gradientes da superfície de erro com respeito aos pesos conectados às entradas de um neurônio), necessária para o passo para trás através da rede.

Sumário

Algoritmo de aprendizagem LMS

- Algoritmo de aprendizagem usado por perceptrons de uma única camada.
- LMS = *Least-Mean-Square*.
- Seja um conjunto de treinamento composto de I padrões de entrada/saída desejada.
- Como a rede tem m unidades de entrada x e n unidades de saída desejada t, cada um dos I padrões é do tipo:

$$((x_1, ..., x_m), (t_1, ..., t_n))$$

- Uma vez que se apresenta à rede os I padrões de treinamento, pode-se obter uma medida do erro produzido pela rede.
- O erro é função:
  - de cada padrão, e
  - do erro produzido em cada unidade de saída, quando cada padrão é apresentado.

Treinamento Supervisionado: Regra delta

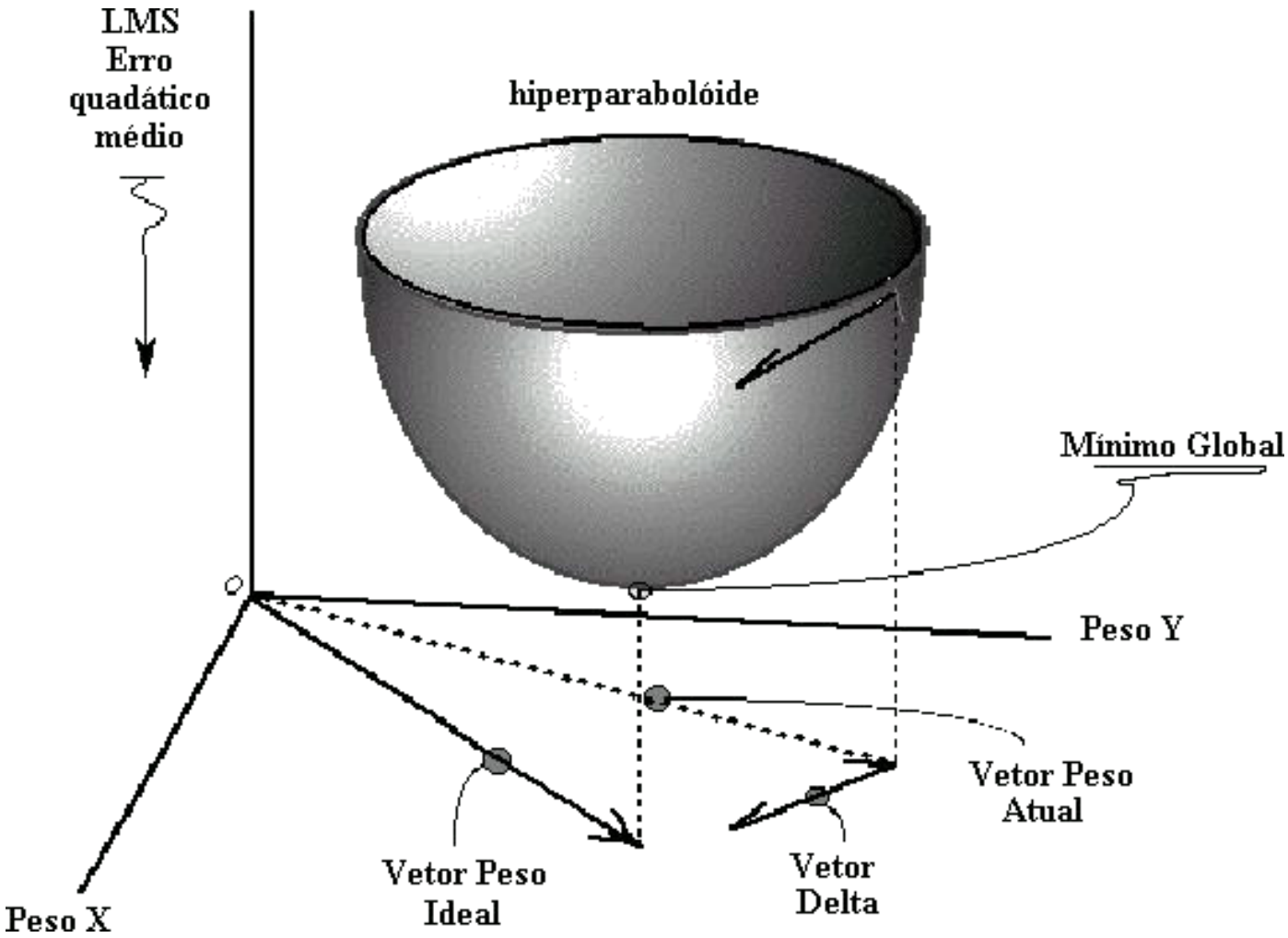


Figure: A direção do gradiente negativo é a de descida mais íngreme (*steepest descent*).

Algoritmo de aprendizagem LMS

- Se...
  - a rede aprende perfeitamente os padrões de treinamento, e
  - os padrões de treinamento refletem perfeitamente a tarefa que se quer aprender
- Então...
  - após o treinamento, o erro será zero.

- A principal causa do erro vem das diferenças entre saída real e saída desejada, que decorre da saída produzida por pesos (e *biases*) incorretos.
- Aprender significa achar os pesos que tornem mínimo o erro.

Algoritmo de aprendizagem LMS

- O erro total E após o treinamento é:

$$E(w) = \sum_{p=1}^{I} E_p$$

onde E_p é o erro produzido quando o p-ésimo padrão de treinamento é apresentado à rede.

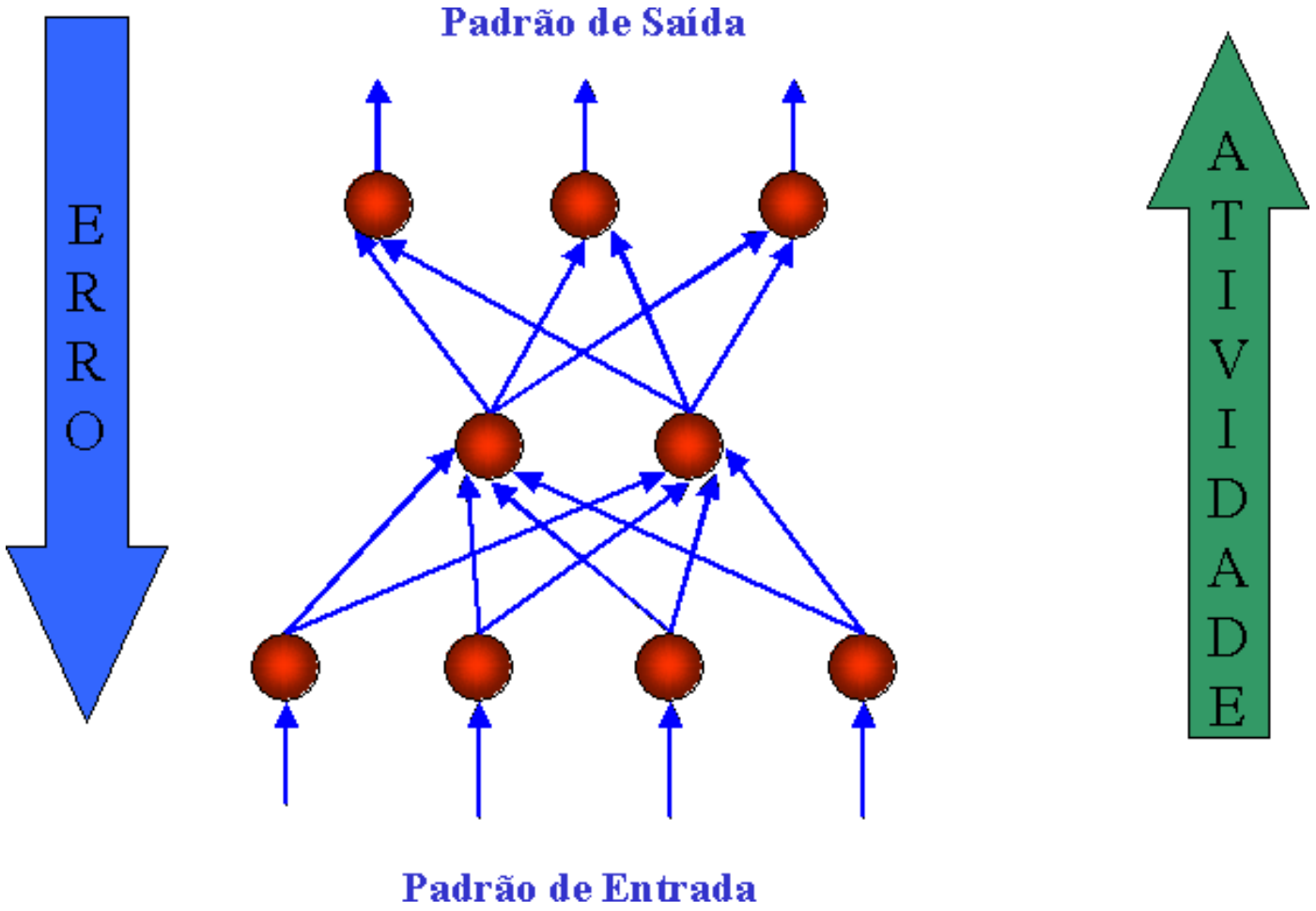
- O erro E_p pode ser medido de várias maneiras, mas a medida mais usada é o *erro quadrático médio*:

$$E_p = \frac{1}{2} \sum_{k=1}^{n} (t_k - y_k)^2$$

Back-propagation

- A dificuldade de aplicar a regra delta: como calcular o erro para uma unidade da camada escondida?
- Solução: (*Error*) *Back-propagation* $\Rightarrow$ Generalização da regra delta.
- *Back-propagation* é um algoritmo supervisionado, que requer duas fases:
  - propagação da ativação, e
  - retropropagação do erro.
- permite calcular o erro baseado só em informação local: disponível nos pesos e unidades próximas ao peso que está sendo modificado (como no SNC).

Error Back-propagation



Gradiente [4]

- Gradiente:

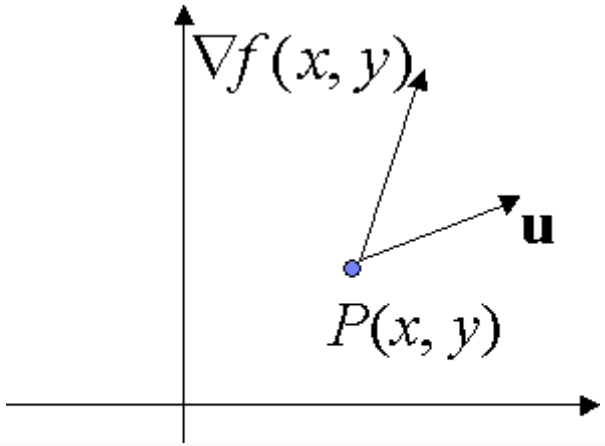
$$\nabla f(x,y) = (\frac{\partial}{\partial x} f(x,y), \frac{\partial}{\partial y} f(x,y)) \quad (2)$$

- Derivada direcional:

$$D_u f(x,y) = \nabla f(x,y).\mathbf{u} = \| \nabla f(x,y) \| \| \mathbf{u} \| \| cos\gamma \| \quad (3)$$

$$D_u f(x,y) = \| \nabla f(x,y) \| \| cos\gamma \| \quad (4)$$

- $D_u f(x,y)$ é a taxa de variação de $f(x,y)$ na direção definida por **u**.



Gradiente [4]

- **Teorema do gradiente**: Seja f uma função de duas variáveis diferenciável no ponto $P(x, y)$.
  - O máximo de $D_u f(x, y)$ em $P(x, y)$ é $\| \nabla f(x, y) \|$.
  - O máximo da taxa de crescimento de $f(x, y)$ em $P(x, y)$ ocorre na direção de $\nabla f(x, y)$.
- *Corolário*: Seja f uma função de duas variáveis, diferenciável no ponto $P(x, y)$.
  - O mínimo de $D_u f(x, y)$ em $P(x, y)$ é $- \| \nabla f(x, y) \|$.
  - O máximo da taxa de decrescimento de $f(x, y)$ em $P(x, y)$ ocorre na direção de $-\nabla f(x, y)$.

Gradiente [4]

- Método do Gradiente Descendente (GD):

$$\Delta w_{ij} = -\eta \frac{\partial E_j}{\partial w_{ij}} \tag{5}$$

- Cada peso sináptico i do elemento processador j é atualizado proporcionalmente ao negativo da derivada parcial do erro deste processador com relação ao peso.
- Logo

$$\Delta w_{ij} = -\eta \frac{\partial E_j}{\partial w_{ij}} = -\eta \frac{\partial E_j}{\partial y_j} \frac{\partial y_j}{\partial w_{ij}} \tag{6}$$

- Como $E_j = \frac{1}{2}(t_j - y_j)^2$, $\frac{\partial E_j}{\partial y_j} = 2.\frac{1}{2}(t_j - y_j).(-1)$.
- Como $y_j = \sum x_i w_{ij} + \theta_j$, $\frac{\partial y_j}{\partial w_{ij}} = x_i$
- Assim, $\Delta w_{ij} = -\eta x_i (t_j - y_j)$

Energia do erro

- O sinal de erro na saída do neurônio j na iteração n (isto é, apresentação do n-ésimo exemplo de treinamento) é definido por:

$$e_j(n) = d_j(n) - y_j(n) \tag{7}$$

- Define-se o valor instantâneo da energia do erro para o neurônio j como $\frac{1}{2}e_j^2(n)$.
- O valor instantâneo $\xi(n)$ da energia do erro total é obtida somando $\frac{1}{2}e_j^2(n)$ de *todos os neurônios da camada de saída*:

$$\xi(n) = \frac{1}{2}\sum_{j \in C} e_j^2(n) \tag{8}$$

onde o conjunto C inclui todos os neurônios da camada de saída.

Energia do erro

- A *energia do erro quadrático médio* é obtida somando $\xi(n)$ de todo n e normalizando em relação a N, o número total de padrões (exemplos) do conjunto de treinamento:

$$\xi_{av} = \frac{1}{N} \sum_{n=1}^{N} \xi(n) \tag{9}$$

- $\xi(n)$, e portanto ξ_{av}, é função de todos os parâmetros livres (pesos sinápticos e *biases*).
- ξ_{av} representa a *função custo* como uma medida da performance do aprendizado.
- O objetivo é minimizar ξ_{av}, sendo para isso, usada uma aproximação similar à adotada para a derivação do algoritmo LMS.

BP

- Considera-se um método onde os pesos são atualizados numa base *padrão a padrão* até que uma **época** tenha acontecido, isto é, até que todo o conjunto de treinamento tenha sido apresentado à rede.
- Os ajustes aos pesos são feitos de acordo com os respectivos erros computados para *cada* padrão apresentado à rede.
- A média aritmética destas mudanças de peso é uma *estimativa* da mudança real resultante da modificação dos pesos baseada na minimização da função custo ξ_{av} sobre todo o conjunto de treinamento.

Campo local induzido

- O campo local induzido $v_j(n)$ produzido na entrada da função de ativação associada com o neurônio j é:

$$v_j(n) = \sum_{i=0}^{m} w_{ji}(n) y_i(n) \tag{10}$$

onde m é o número total de entradas (excluindo o *bias*) aplicadas ao neurônio j.

- O peso sináptico w_{j0} (correspondendo à entrada fixa $y_0 = +1$) é igual ao *bias* b_j aplicado ao neurônio j.
- Portanto a função sinal $y_j(n)$ que aparece na saída do neurônio j na iteração n é

$$y_j(n) = \varphi_j(v_j(n)) \tag{11}$$

BP

- Similarmente ao algoritmo LMS, BP aplica uma correção $\Delta w_{ji}(n)$ ao peso sináptico $w_{ji}(n)$ proporcional à derivada parcial $\partial \xi(n)/\partial w_{ji}(n)$. De acordo com a regra da cadeia, este gradiente pode ser expresso:

$$\frac{\partial \xi(n)}{\partial w_{ji}(n)} = \frac{\partial \xi(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{ji}(n)} \tag{12}$$

- A derivada parcial $\partial \xi(n)/\partial w_{ji}(n)$ representa um *fator de sensibilidade*. que determina a direção da busca no espaço de pesos para o peso sináptico w_{ji}.
- Diferenciando ambos os lados da equação 8 com respeito a $e_j(n)$:

$$\frac{\partial \xi(n)}{\partial e_j(n)} = e_j(n) \tag{13}$$

BP

- Diferenciando ambos os lados da equação 7 com respeito a $y_j(n)$:

$$\frac{\partial e_j(n)}{\partial y_j(n)} = -1 \tag{14}$$

- Agora diferenciando a equação 11 com respeito a $v_j(n)$:

$$\frac{\partial y_j(n)}{\partial v_j(n)} = \varphi_j'(v_j(n)) \tag{15}$$

- Finalmente, diferenciando a equação 10 com respeito a $w_{ji}(n)$:

$$\frac{\partial v_j(n)}{\partial w_{ji}(n)} = y_i(n) \tag{16}$$

- O uso das equações 13 a 16 em 12:

$$\frac{\partial \xi(n)}{\partial w_{ji}(n)} = -e_j(n)\varphi_j'(v_j(n))y_i(n) \tag{17}$$

Sumário

Gradiente descendente

- A correção $\Delta w_{ji}(n)$ aplicada a $w_{ji}(n)$ é definida pela *regra delta*:

$$\Delta w_{ji}(n) = -\eta \frac{\partial \xi(n)}{\partial w_{ji}(n)} \tag{18}$$

- O uso de sinal de menos justifica o **gradiente descendente** no espaço de pesos (isto é, a busca de uma direção para a mudança do peso que reduza o valor de $\xi(n)$).
- Assim, o uso das equações 17 em 18:

$$\Delta w_{ji}(n) = \eta \delta_j(n) y_i(n) \tag{19}$$

onde o *gradiente local* $\delta_j(n)$ é definido por

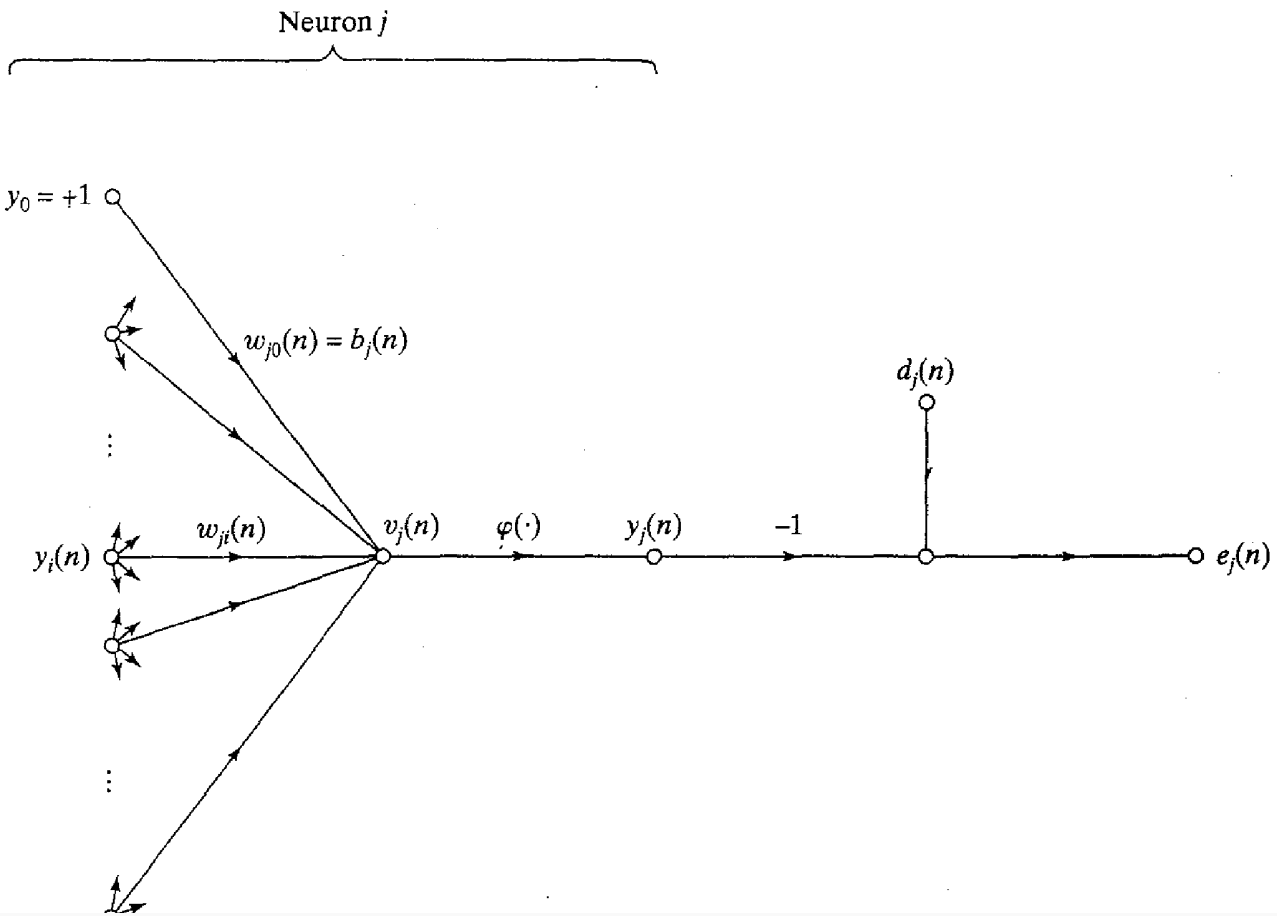
$$\delta_j(n) = -\frac{\partial \xi(n)}{\partial v_j(n)} = \frac{\partial \xi(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} = e_j(n) \varphi_j'(v_j(n)) \tag{20}$$

Gradiente local

- De acordo com a equação 20, o gradiente local $\delta_j(n)$ para o neurônio de saída j é igual ao produto do sinal de erro correspondente $e_j(n)$ para esse neurônio e a derivada $\varphi'_j(v_j(n))$ da função de ativação associada.
- Observe que, a partir das equações 19 e 20, o sinal de erro $e_j(n)$ é um fator chave no ajuste de pesos $\Delta w_{ji}(n)$.
- Dois casos distintos:
  1. neurônio j é saída: simples, pois há uma resposta desejada. Usa-se a equação 7 para computar o sinal de erro $e_j(n)$ associado a este neurônio; veja figura 2. Determinado $e_j(n)$, é fácil computar o gradiente local $\delta_j(n)$ usando a equação 20.
  2. neurônio j é escondido: como penalizar ou recompensar neurônios escondidos?

Neurônio de saída

Figure: Grafo do fluxo de sinal mostrando os detalhes do neurônio de saída *j* [2].



Neurônio escondido

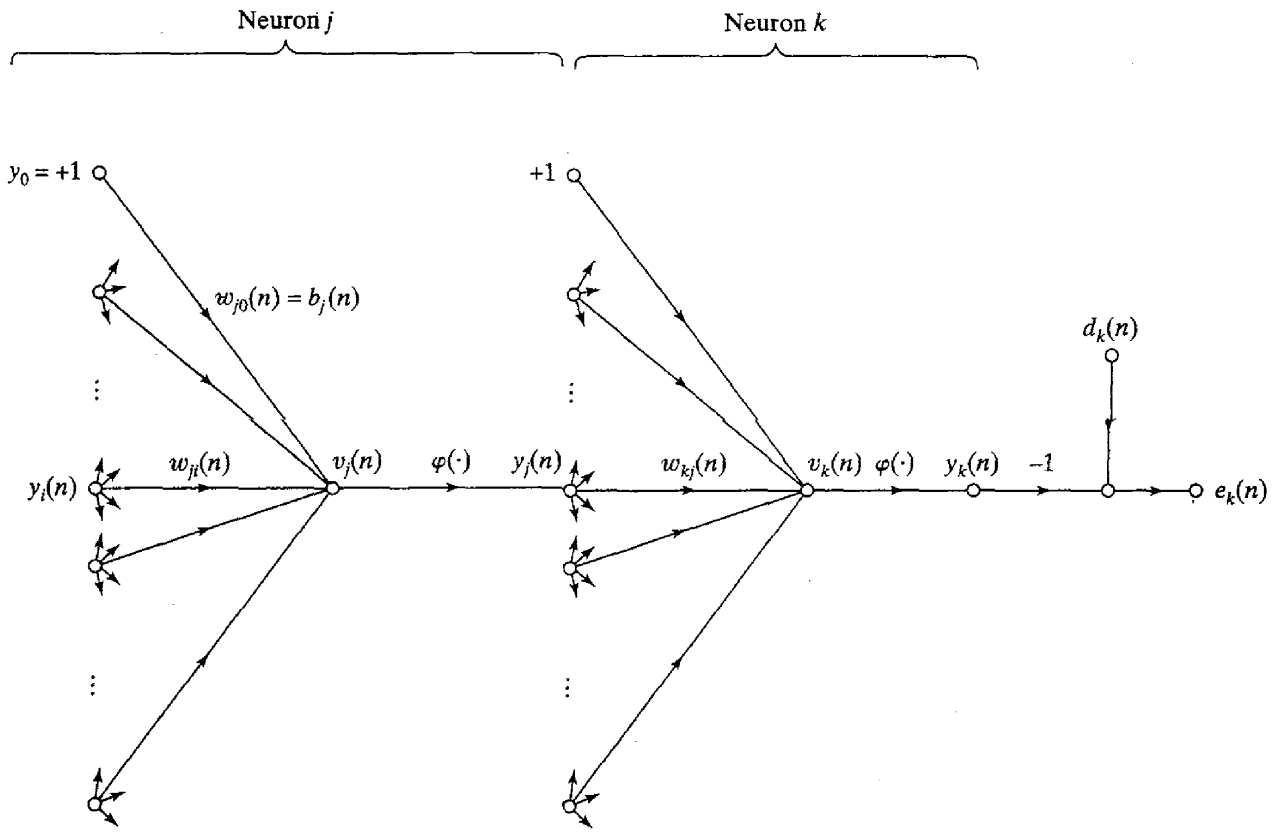
- Caso 2: Neurônio *j* é escondido:
  - Neste caso, não há resposta desejada para **este** neurônio.
  - O sinal de erro deve ser determinado recursivamente em termos dos sinais de erro de todos os neurônios aos quais o neurônio escondido está diretamente conectado: BP torna-se complicado!
  - Veja a situação retratada na figura 3, que mostra o neurônio *j* como um nó escondido da rede.
  - De acordo com a equação 20, pode-se redefinir o gradiente local $\delta_j(n)$ para o neurônio escondido *j* como

$$\delta_j(n) = -\frac{\partial \xi(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} = -\frac{\partial \xi(n)}{\partial y_j(n)} \varphi_j'(v_j(n)) \qquad (21)$$

  onde na segunda parte foi usada a equação 15.

Neurônio de saída conectado ao neurônio escondido

Figure: Grafo do fluxo de sinal mostrando os detalhes do neurônio de saída *k* conectado ao neurônio escondido *j* [2].



BP

- Para calcular a derivada parcial $\frac{\partial \xi(n)}{\partial y_j(n)}$, da figura 3, vê-se que:

$$\xi(n) = \frac{1}{2} \sum_{k \in C} e_k^2(n) \tag{22}$$

que é a equação 8 com o k no lugar do j.

- Diferenciando a equação 22 com respeito ao sinal de função $y_j(n)$ tem-se

$$\frac{\partial \xi(n)}{\partial y_j(n)} = \sum_k e_k \frac{\partial e_k(n)}{\partial y_j(n)} \tag{23}$$

- Depois, usa-se a regra da cadeia para a derivada parcial $\frac{\partial e_k(n)}{\partial y_j(n)}$, re-escrevendo a equação 23 na forma equivalente:

$$\frac{\partial \xi(n)}{\partial y_j(n)} = \sum_k e_k \frac{\partial e_k(n)}{\partial v_k(n)} \frac{\partial v_k(n)}{\partial y_j(n)} \tag{24}$$

BP

- Entretanto, da figura 3, nota-se que:

$$e_k(n) = d_k(n) - y_k(n) = d_k(n) - \varphi_k(v_k(n)) \tag{25}$$

- Portanto,

$$\frac{\partial e_k(n)}{\partial v_k(n)} = -\varphi_k'(v_k(n)) \tag{26}$$

- Nota-se da figura 3 que, para o neurônio de saída k, o campo local induzido é

$$v_k(n) = \sum_{j=0}^{m} w_{kj}(n) y_j(n) \tag{27}$$

onde m é o número total de entradas (excluindo o *bias*) aplicadas ao neurônio k.

- De novo, $w_{k0}(n) = b_k(n)$ e a entrada correspondente é fixa em +1.

Fórmula do BP

- Diferenciando a equação 27 com respeito a $y_j(n)$:

$$\frac{\partial v_k(n)}{\partial y_j(n)} = w_{kj}(n) \tag{28}$$

- Usando as equações 26 e 28 em 24:

$$\frac{\partial \xi(n)}{\partial y_j(n)} = -\sum_k e_k(n)\varphi'_k(v_k(n))w_{kj}(n) = -\sum_k \delta_k(n)w_{kj}(n) \tag{29}$$

  onde, na segunda parte, foi usada a definição de gradiente local $\delta_k(n)$ da equação 20.
- Finalmente, usando a equação 29 em 21, tem-se a fórmula do *back-propagation* para o gradiente local $\delta_j(n)$:

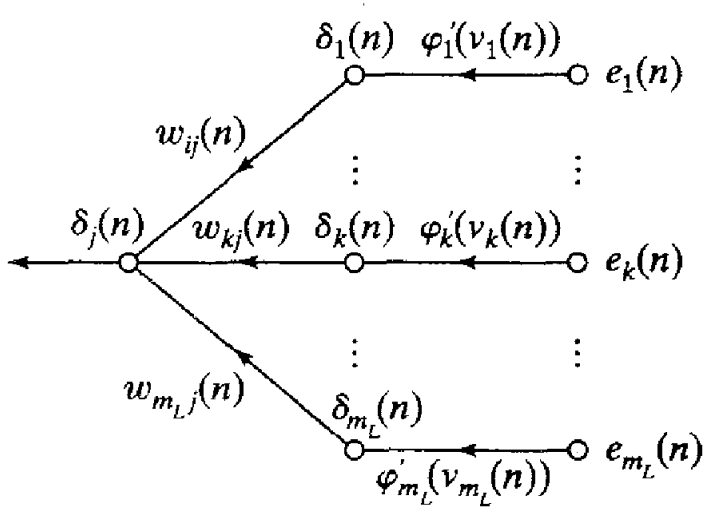
$$\delta_j(n) = \varphi'_j(v_j(n)) \sum_k \delta_k(n)w_{kj}(n) \tag{30}$$

BP

- A figura 10 abaixo [2] mostra a representação do grafo do fluxo de sinal da equação 30, assumindo que a camada de saída contém m_L neurônios.
- A correção $\Delta w_{ji}(n)$ aplicada ao peso sináptico entre os neurônios i e j é definida pela regra delta:

$$\Delta w_{ji}(n) = \eta.\delta_j(n).y_i(n) \tag{31}$$

onde $y_i(n)$ é o sinal de entrada do neurônio j.



BP

- O gradiente local $\delta_j(n)$ depende se o neurônio j é um neurônio de saída ou escondido:
  1. Se j é de saída, $\delta_j(n)$ é igual ao produto da derivada $\varphi'_j(v_j(n))$ e o sinal de erro $e_j(n)$, ambos associados ao neurônio j (vide equação 20).
  2. Se j é escondido, $\delta_j(n)$ é igual ao produto da derivada associada $\varphi'_j(v_j(n))$ e a soma ponderada dos δs computada para os neurônios da próxima camada escondida ou de saída que estão conectados ao neurônio j (vide equação 30).
- A aplicação do BP requer dois passos de computação:
  1. Passo *para frente*: os pesos mantêm-se fixos e os sinais de função são computados neurônio por neurônio.
  2. Passo *para trás*: os pesos são alterados, através do cálculo do gradiente local para cada neurônio, da camada de saída para a entrada.

BP

- Passo *para frente*:
  - Sinal de função:

$$y_j(n) = \varphi(v_j(n)) \tag{32}$$

onde

$$v_j(n) = \sum_{i=0}^{m} w_{ji}(n) y_i(n) \tag{33}$$

  - Se j está na primeira camada escondida, $m = m_0$ e o índice i refere-se ao i-ésimo terminal de entrada da rede:

$$y_i(n) = x_i(n) \tag{34}$$

onde $x_i(n)$ é o i-ésimo elemento do vetor (padrão) de entrada.

  - Se j está na camada de saída, $m = m_L$ e j refere-se ao j-ésimo terminal de saída:

$$y_j(n) = o_j(n) \tag{35}$$

onde $o_j(n)$ é o j-ésimo elemento do vetor (padrão) de saída.

BP

- Passo *para frente* (cont.):
  - $o_j(n)$ é comparada à resposta desejada $d_j(n)$, obtendo-se o sinal de erro $e_j(n)$ para o j-ésimo neurônio de saída.
  - Portanto, a fase *para frente* começa na primeira camada escondida através da apresentação do vetor de entrada e termina na camada de saída computando o sinal de erro para cada neurônio desta camada.
- Passo *para trás*:
  - Começa na camada de saída passando os sinais de erro "para trás" na rede, camada por camada, recursivamente computando o δ (gradiente local) para cada neurônio.
  - Esse processo recursivo permite mudanças nos pesos sinápticos de acordo com a regra delta da equação 31.
  - Para um neurônio localizado na camada de saída, δ é igual ao sinal de erro multiplicado pela primeira derivada de sua não-linearidade (figura 10).
  - Para as outras camadas, calcula-se o δ através da equação 30.

Sumário

Função de Ativação

- Para a computação do δ de cada neurônio do MLP há a necessidade de se conhecer a derivada da função de ativação $\varphi(\cdot)$ associada ao neurônio.
- Para que exista a derivada, $\varphi(\cdot)$ deve ser contínua (diferenciável).
- Não-linearidade sigmoidal:
  1. *Função logística*:

$$\varphi_j(v_j(n)) = \frac{1}{1 + exp(-av_j(n))}, \qquad a > 0, -\infty < v_j(n) < \infty \tag{36}$$

  2. *Função tangente hiperbólica*:

$$\varphi_j(v_j(n)) = a\, tanh(bv_j(n)), \qquad (a, b) > 0 \tag{37}$$

onde a e b são constantes.

Função de Ativação

1 *Função logística*:

- De acordo com esta não-linearidade, a amplitude da saída fica $0 \leq y_j \leq 1$.
- Diferenciando a equação 36 com respeito a $v_j(n)$:

$$\varphi'_j(v_j(n)) = \frac{a\, exp(-av_j(n))}{(1 + exp(-av_j(n)))^2} \tag{38}$$

- Com $y_j(n) = \varphi_j(v_j(n))$, elimina-se o termo exponencial:

$$\varphi'_j(v_j(n)) = a\, y_j(n)(1 - y_j(n)) \tag{39}$$

- Para um neurônio j da camada de saída, $y_j(n) = o_j(n)$:

$$\delta_j(n) = e_j(n)\varphi'_j(v_j(n)) = a(d_j(n) - o_j(n))o_j(n)(1 - o_j(n)) \tag{40}$$

- Por outro lado, para j escondido:

$$\delta_j(n) = \varphi'_j(v_j(n)) \sum_k \delta_k(n) w_{kj}(n) = a y_j(n)(1-y_j(n)) \sum_k \delta_k(n) w_{kj}(n) \tag{41}$$

Função de Ativação

1. *Função logística* (cont.):
   - Note, a partir da equação 39, que a derivada $\varphi_j'(v_j(n))$ tem seu valor máximo em $y_j(n) = 0.5$ e seu valor mínimo (zero) em $y_j(n) = 0$ ou $y_j(n) = 1$.
   - Como a quantidade de mudança em um peso sináptico é proporcional à derivada $\varphi_j'(v_j(n))$, segue que os pesos são mudados o máximo onde os sinais de função estão na metade: característica do aprendizado do BP que contribui para sua estabilidade [6].

2. *Função tangente hiperbólica*:
   - A derivada de 37 com respeito a $v_j(n)$:

$$\varphi_j'(v_j(n)) = ab\, sech^2(bv_j(n)) = ab(1 - tanh^2(bv_j(n))) \quad (42)$$

$$\varphi_j'(v_j(n)) = \frac{b}{a}(a - y_j(n))(a + y_j(n)) \quad (43)$$

Função de Ativação

2 *Função tangente hiperbólica* (cont.):

- Para j na saída:

$$\delta_j(n) = e_j(n)\varphi_j'(v_j(n)) = \frac{b}{a}(d_j(n) - o_j(n))(a - o_j(n))(a + o_j(n)) \tag{44}$$

- Para j na escondida:

$$\delta_j(n) = \varphi_j'(v_j(n)) \sum_k \delta_k(n) w_{kj}(n) \tag{45}$$

$$\delta_j(n) = \frac{b}{a}(a - y_j(n))(a + y_j(n)) \sum_k \delta_k(n) w_{kj}(n) \tag{46}$$

- Usando as equações 40 e 41 para a função logística e as equações 44, 45 e 46 para a *tanh*, pode-se calcular δ_j sem conhecimento explícito da função de ativação.

Taxa de aprendizado

- BP provê uma "aproximação" à trajetória no espaço de pesos computado pelo método do *steepest descent*.
- Quanto menor for η, menores as mudanças nos pesos sinápticos entre duas iterações e mais suave a trajetória no espaço de pesos.
- O custo é um aprendizado mais lento.
- Para η muito grande, a rede pode ficar instável.
- Uma forma de aumentar η sem o perigo da instabilidade é modificar a regra delta da equação 19 incluindo um termo *momentum* [6]:

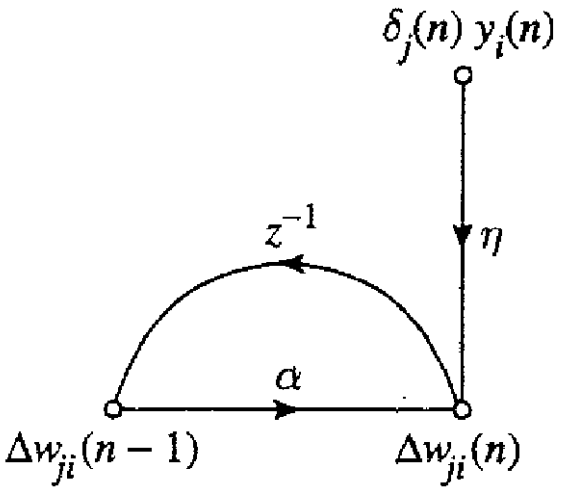
$$\Delta w_{ji}(n) = \alpha \Delta w_{ji}(n-1) + \eta \delta_j(n) y_i(n) \qquad (47)$$

onde α, número positivo, chamado de *constante momentum*, que controla o *loop* de retro-alimentação que age em volta de $\Delta w_{ji}(n)$ (figura 7 [2]).

Taxa de aprendizado

- A equação 47 é chamada de **regra delta generalizada**, que inclui a regra delta da equação 19 como um caso especial ($\alpha = 0$).
- Re-escrevendo a equação 47 como uma série temporal: o índice t vai de 0 a n.

$$\Delta w_{ji}(n) = \eta \sum_{t=0}^{n} \alpha^{n-t} \delta_j(t) y_i(t) \tag{48}$$



Taxa de aprendizado

- Das equações 17 e 20 nota-se que

$$\delta_j(n)y_i(n) = -\frac{\partial \xi(n)}{\partial w_{ji}(n)} \tag{49}$$

- Assim

$$\Delta w_{ji}(n) = -\eta \sum_{t=0}^{n} \alpha^{n-t} \frac{\partial \xi(t)}{\partial w_{ji}(t)} \tag{50}$$

- 3 observações:
  1. $\Delta w_{ji}(n)$ representa a soma de uma série temporal exponencialmente ponderada. Para que seja *convergente*: $0 \leq |\ \alpha\ | < 1$.
  2. Quando $\frac{\partial \xi(n)}{\partial w_{ji}(n)}$ tem o mesmo sinal algébrico em iterações consecutivas, $\Delta w_{ji}(n)$ cresce em magnitude.
  3. Quando $\frac{\partial \xi(n)}{\partial w_{ji}(n)}$ tem sinais opostos em iterações consecutivas, $\Delta w_{ji}(n)$ encolhe em magnitude.

Critérios de parada

- Não se pode provar que BP converge.
- Assim, não há critérios bem definidos de parada.
- A ideia é considerar as propriedades dos *mínimos locais ou globais* da superfície do erro.
- Seja o vetor de peso $\mathbf{w}^*$ denotando um mínimo, seja local ou global.
- Uma condição necessária para $\mathbf{w}^*$ ser mínimo é que o vetor gradiente $\mathbf{g(w)}$ (derivada parcial de primeira ordem) da superfície do erro com relação ao vetor de pesos $\mathbf{w}$ seja zero em $\mathbf{w} = \mathbf{w}^*$.

> **Critério 1***: O algoritmo BP converge quando a norma Euclidiana do vetor gradiente alcança um limiar de gradiente suficientemente pequeno.*

Critérios de parada

- A desvantagem do critério 1 é que, para tentativas bem sucedidas, os tempos de aprendizado podem ser longos.
- Além disso, requer a computação do vetor gradiente **g(w)**.
- Outro ponto a se considerar é que a função custo ou medida de erro $\xi_{av}(\mathbf{w})$ é estacionária no ponto $\mathbf{w} = \mathbf{w}^*$:

  Critério 2*: O algoritmo BP converge quando a taxa absoluta de mudança no erro quadrático médio por época é suficiente pequeno.*

- Essa taxa é considerada pequena na faixa de 0.1 a 1% por época (as vezes, 0.01%).
- Este critério pode resultar numa terminação prematura do processo de aprendizado.

Back-propagation

O algoritmo supervisionado *back-propagation* pode ser resumido:

1. Propague a ativação
   - da camada de entrada para a escondida,
   - da camada escondida para a de saída.
2. Calcule o erro.
   - para as unidades de saída,
   - Retropropague o erro para as unidades escondidas e para as de entrada.
     - Os passos 1 e 2 constituem um ciclo de ativação (época).

Back-propagation

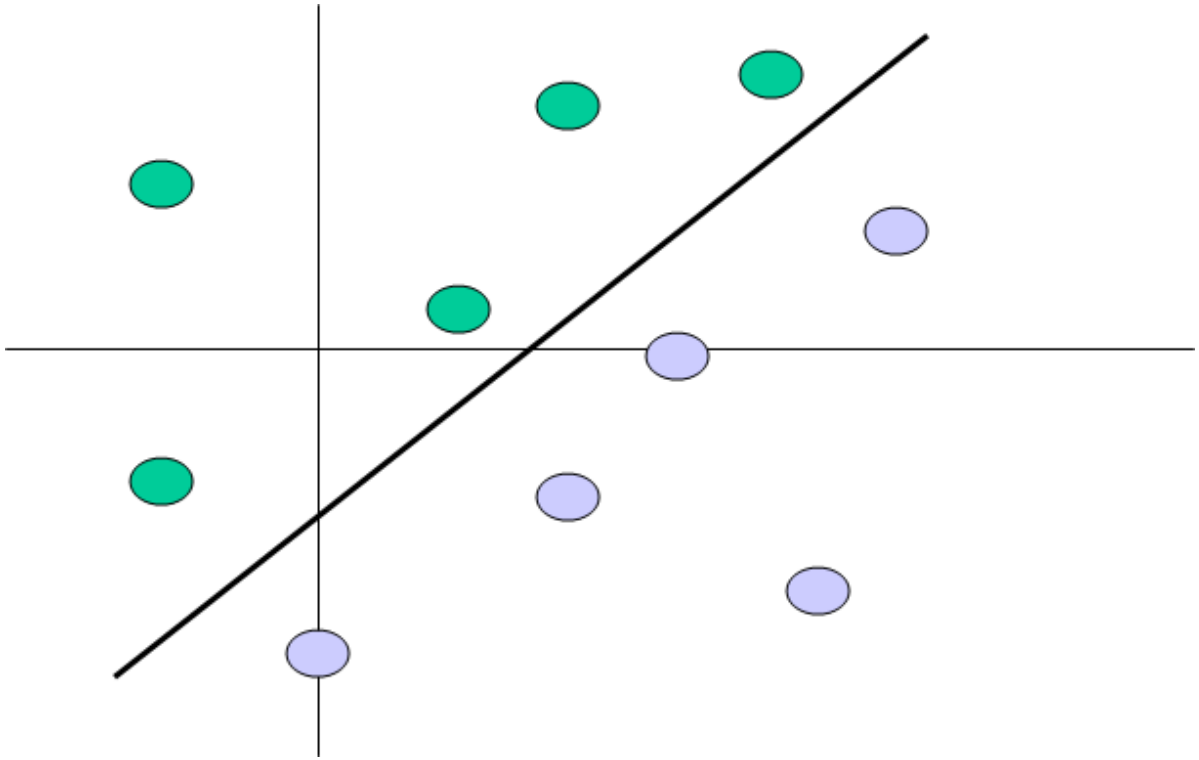
- Os problemas do *back-propagation*:
  - É bastante caro computacionalmente (lento),
  - Não resolve bem problemas de grande porte,
  - Às vezes, a solução encontrada é um mínimo local - um valor localmente mínimo para a função erro.
- Vantagens do *back-propagation*:
  - poder de aproximação universal:
    - dada uma função contínua, existe uma rede de duas camadas (uma escondida) que pode ser treinada por *back-propagation* de modo a aproximar o quanto se queira essa função.
  - algoritmo mais usado.

Sumário

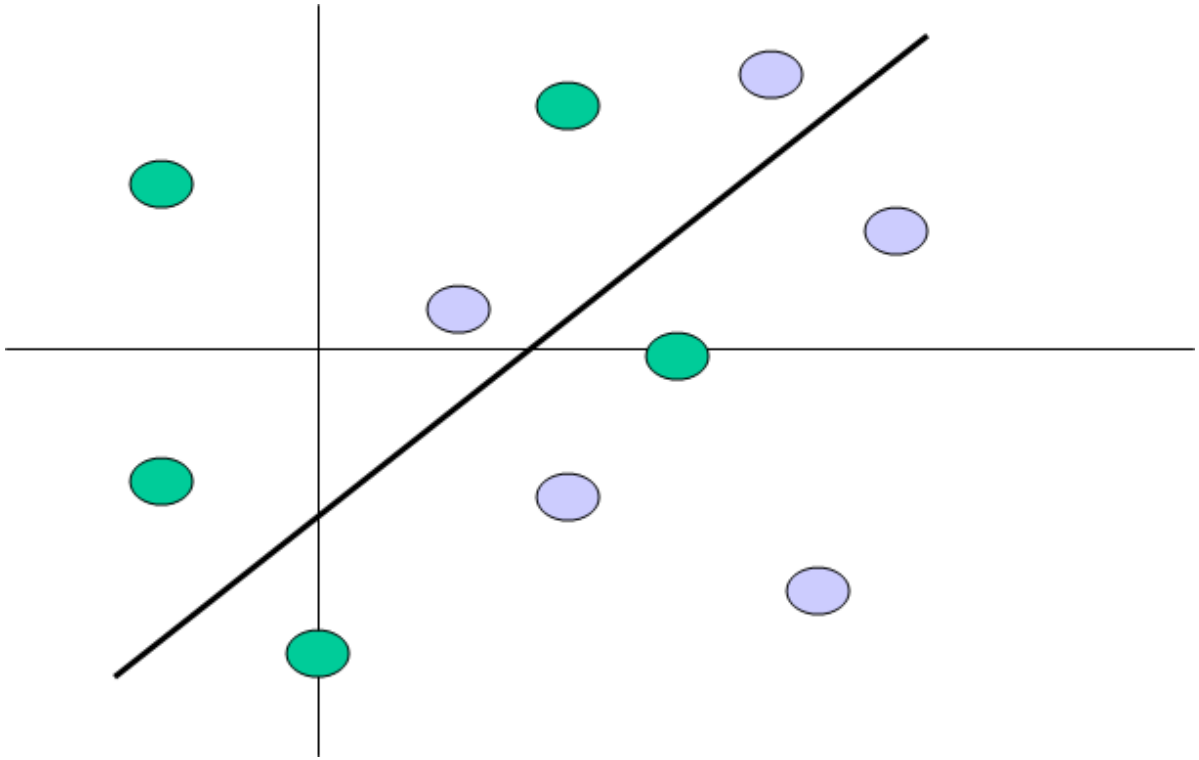
Problema XOR

- No perceptron de única camada não há neurônio escondido, porisso não é possível classificar padrões de entrada que não são linearmente separáveis.
- Entretanto, padrões não linearmente separáveis são comuns.
- Por exemplo, o problema do ou-exclusivo (XOR), que pode ser visto como um caso especial de um problema mais geral, a classificação de pontos no *hipercubo unitário*.
- Cada ponto no hipercubo ou é da classe 0 ou da classe 1.
- Porém, no caso do XOR, precisa-se considerar apenas os quatro cantos do *quadrado unitário*, que correspondem aos padrões de entrada (0,0), (0,1), (1,1) e (1,0).

Conjunto de pontos linearmente separáveis

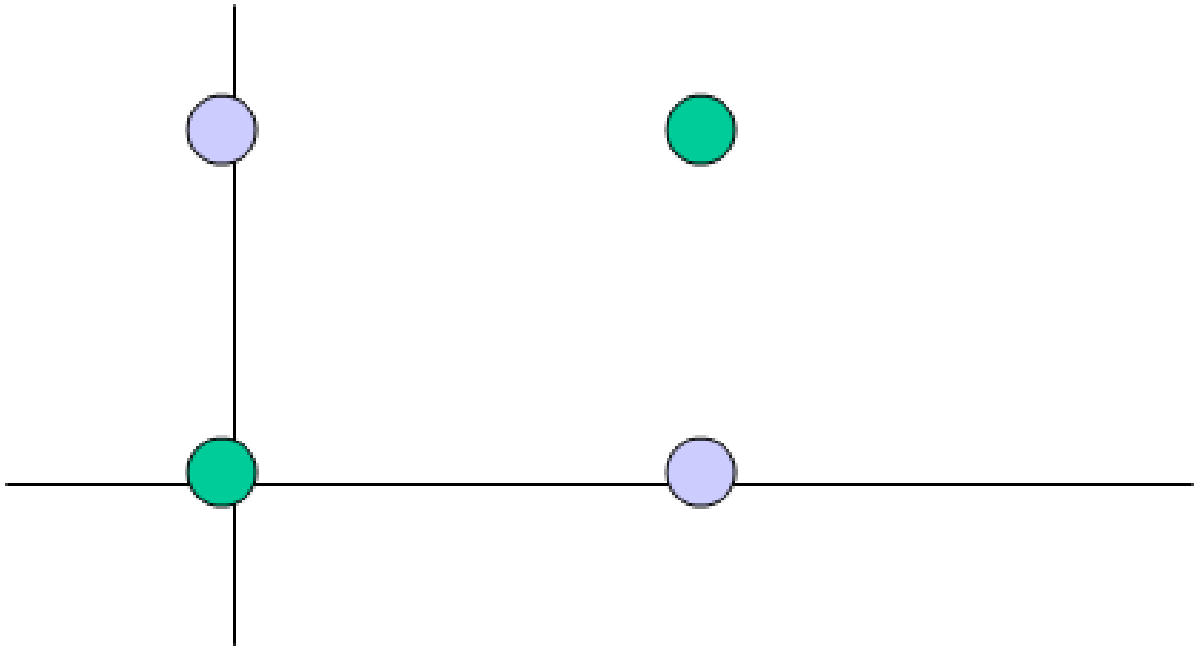


Conjunto de pontos *não*-linearmente separáveis (por 1 reta)



OU Exclusivo

- $\{(0,0),0;(0,1),1;(1,0),1;(1,1),0\}$



- E por *n* retas?

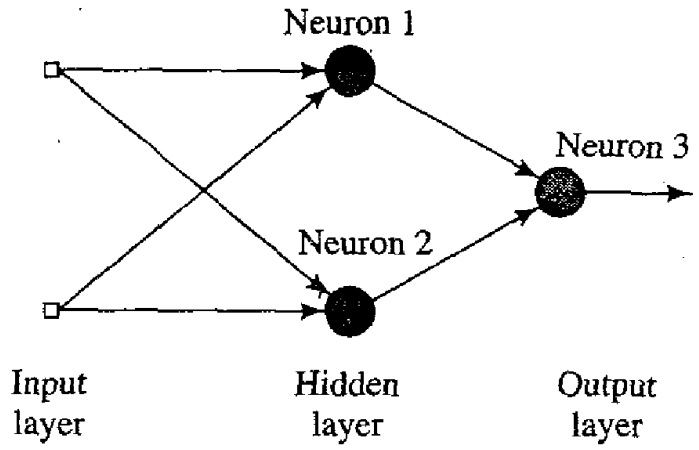
OU Exclusivo

- Pode-se resolver o problema XOR usando uma única camada escondida com dois neurônios (veja figura 7*a* [2]).
- O grafo do fluxo de sinal é mostrado na figura 7*b* [2]. Assume-se:
  - Cada neurônio é representado por um modelo de McCulloch-Pitts [3], que usa uma função limiar como função de ativação.
  - Os bits 0 e 1 são representados pelos níveis 0 e +1, respectivamente.
- O neurônio 1 da camada escondida é caracterizado como:

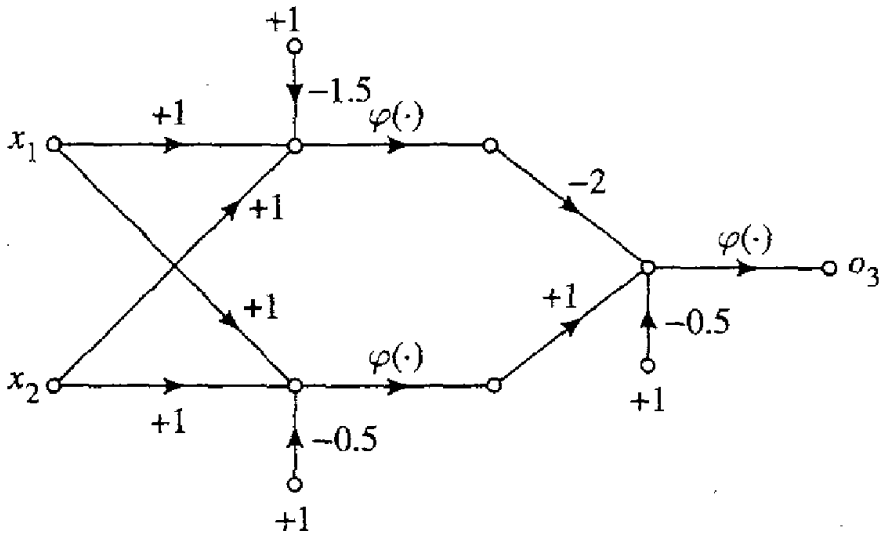
$$w_{11} = w_{12} = +1 \qquad b_1 = -\frac{3}{2} \tag{51}$$

- A inclinação do limite de decisão construído por esse neurônio escondido é igual a -1 (veja figura 8*a* [2]).

OU Exclusivo

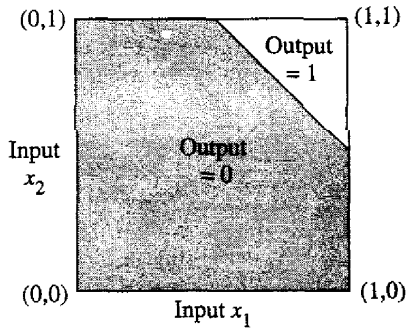


(a)

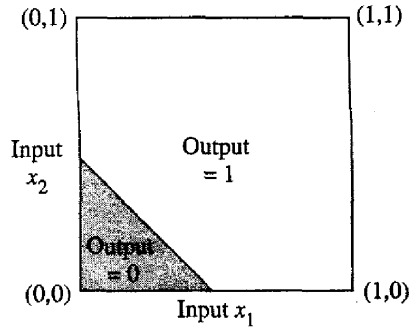


(b)

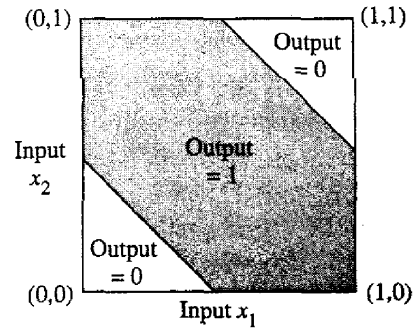
OU Exclusivo



(a)



(b)



(c)

OU Exclusivo

- O neurônio 2 da camada escondida é caracterizado por:

$$w_{21} = w_{22} = +1 \qquad b_2 = -\frac{1}{2} \tag{52}$$

- A orientação e posição do limite de decisão construído pelo segundo neurônio são mostrados na figura 8*b* [2].
- O neurônio de saída 3 é caracterizado como:

$$w_{31} = -2 \qquad w_{32} = +1 \qquad b_3 = -\frac{1}{2} \tag{53}$$

- A função desse neurônio de saída é construir uma combinação linear dos limites de decisão formados pelos dois neurônios escondidos (figura 8*c* [2]).

Sumário

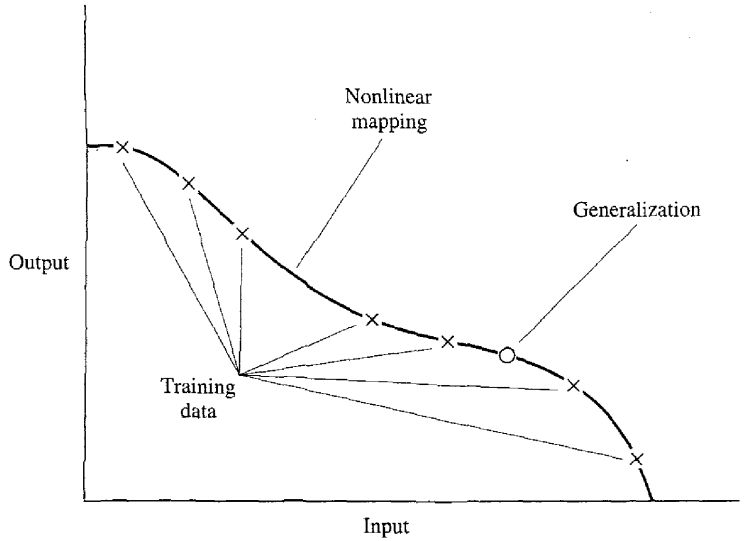
Generalização

- No aprendizado BP, começa-se com uma amostra de treinamento e usa-se o algoritmo BP para computar os pesos sinápticos de um MLP carregando (codificando) o máximo possível de exemplos de treinamento na rede.
- A esperança é que a RNA projetada *generalize*: quando os mapeamentos entrada-saída computados pela rede são corretos (ou próximos) para dados de teste **nunca** usados na criação ou no treinamento da rede.
- Assume-se que os dados de teste são retirados da mesma população usada para gerar o conjunto de treinamento.
- O processo de aprendizado (treinamento de uma rede neural) pode ser visto como um problema de "preenchimento de curva."

Generalização

- Veja na figura 4*a* [2] como a generalização pode ocorrer em uma rede hipotética.
- O mapeamento de entrada-saída não linear representado pela curva é computado pela rede como um resultado do aprendizado dos pontos “training data”.
- O ponto “generalization” é o resultado da interpolação feita pela rede.
- Problema: quando a rede aprende muitos exemplos de entrada-saída pode terminar memorizando os dados de treinamento.
- Isso pode fazer com que a RNA ache uma característica (devido ao ruído) que está presente nos dados de treinamento, mas que não é verdadeira para a função modelada: overfitting ou overtraining.
- Quando uma rede é “overtrained”, perde a habilidade de generalizar.

Generalização



Generalização

- Um exemplo de como a generalização pobre devido à memorização em uma RNA pode ocorrer é ilustrada na figura 4*b* [2], para os mesmos dados da figura 4*a*.
- "Memorização" é essencialmente uma "look-up table", que implica que o mapeamento de entrada-saída computado pela rede não é suave.
- A suavidade do mapeamento entrada-saída é relacionada aos critérios de seleção de modelo como a *navalha de Occam*[1]: selecionar a função "mais simples" na ausência de qualquer conhecimento prévio.
- Aqui, a função mais simples significa a função mais suave que aproxima o mapeamento para um dado critério de erro, porque isso acarreta menos recursos computacionais.

[1] Lei da Economia: a explicação mais simples é a correta.

Sumário

Aproximações de funções

- Um MLP treinado com BP pode realizar um *mapeamento de entrada-saída não-linear*.
- Seja m_0 o número de nós de entrada de um MLP e $M = m_L$ o número de neurônios na camada de saída.
- O relacionamento entrada-saída da rede define um mapeamento de um espaço de entrada Euclidiano m_0-dimensional para um espaço de saída Euclidiano M-dimensional, que é infinitamente continuamente diferenciável quando a função de ativação também é.
- Ao avaliar a capacidade de um MLP a partir desse ponto de vista de mapeamento entrada-saída, surge a seguinte questão fundamental:

> *Qual é o número mínimo de camadas escondidas em um MLP com um mapeamento entrada-saída que provê uma realização aproximada de qualquer mapeamento contínuo?*

Teorema da Aproximação Universal

- A resposta a essa pergunta (Teorema da Aproximação Universal):

  Seja $\varphi(\cdot)$ uma função não-constante, limitada, monotônica-crescente e contínua. Seja I_{m_0} o hipercubo unitário m_0-dimensional $[0,1]^{m_0}$. O espaço de funções contínuas em I_{m_0} é denotado por $C(I_{m_0})$. Então, dada qualquer função $f \ni C(I_{m_0})$ e $\epsilon > 0$, existe um inteiro M e conjuntos de constantes reais α_i, b_i e w_{ij}, onde $i = 1, ..., m_1$ e $j = 1, ..., m_0$ tais que pode-se definir

$$F(x_1, ..., x_{m_0}) = \sum_{i=1}^{m_1} \alpha_i \varphi(\sum_{j=1}^{m_0} w_{ij} x_j + b_i) \tag{54}$$

  como uma realização aproximada da função $f(\cdot)$, ou seja,

$$| F(x_1, ..., x_{m_0}) - f(x_1, ..., x_{m_0}) | < \epsilon \tag{55}$$

  para todos $x_1, x_2, ..., x_{m_0}$ que estão no espaço de entrada.

Teorema da Aproximação Universal

- O TAU é diretamente aplicável ao MLP.
- A função logística $\frac{1}{(1+exp(-v))}$ usada como a não-linearidade do MLP é não-constante, limitada e monotônica-crescente.
- A equação 54 representa a saída de um MLP descrito como:
  1. A rede tem m_0 nós de entrada $x_1, ..., x_{m_0}$ e uma única camada escondida com m_1 neurônios;
  2. O neurônio escondido i tem pesos sinápticos $w_{i_1}, ..., w_{m_0}$, e *bias* b_i;
  3. A saída da rede é uma combinação linear das saídas dos neurônios escondidos, com $\alpha_1, ..., \alpha_{m_1}$ definindo os pesos sinápticos da camada de saída.

Teorema da Aproximação Universal

- O TAU é um *teorema de existência*, já que provê a justificativa matemática para a aproximação de uma função contínua arbitrária.
- A equação 54 generaliza aproximações por séries finitas de Fourier.
- Estabelece que *uma única camada escondida é suficiente para um MLP computar uma aproximação* ϵ *uniforme para um dado conjunto de treinamento representado pelo conjunto de entradas* $x_1, ..., x_{m_0}$ *e uma saída desejada* $f(x_1, ..., x_{m_0})$.
- No entanto, o teorema não diz que uma única camada escondida é ótima no sentido de tempo de aprendizado, facilidade de implementação ou (mais importante) generalização.

Sumário

Validação cruzada

- A essência do aprendizado BP é codificar o mapeamento entrada-saída (representado por um conjunto de exemplos rotulados) nos pesos sinápticos e limiares de um MLP.
- A esperança é que a rede seja bem treinada, tal que aprenda o suficiente do passado para generalizar no futuro.
- Pode-se ver o problema de seleção da rede como a escolha, dentro de um conjunto de estruturas de modelos candidatos (parametrização), da "melhor" estrutura de acordo com certo critério.
- Nesse contexto, uma ferramenta da estatística conhecida como *validação cruzada*, pode ajudar.
- Primeiro, o conjunto de dados disponível é particionado aleatóriamente em um conjunto de treinamento e um conjunto de teste.

Validação cruzada

- O conjunto de treinamento é então dividido em:
  1. *Subconjunto de estimativa*: usado para selecionar o modelo.
  2. *Subconjunto de validação*: usado para testar ou validar o modelo.
- A motivação aqui é validar o modelo em um conjunto de dados **diferente** do usado para a estimativa de parâmetros.
- Dessa forma, utiliza-se o conjunto de treinamento para avaliar a performance de vários modelos candidatos e daí escolher o "melhor."
- É possível, no entanto, que o modelo com valores de parâmetros de melhor performance termine *overfitting* o subconjunto de validação.
- Nesse caso, a performance da generalização é medida no conjunto de teste, diferente do subconjunto de validação.

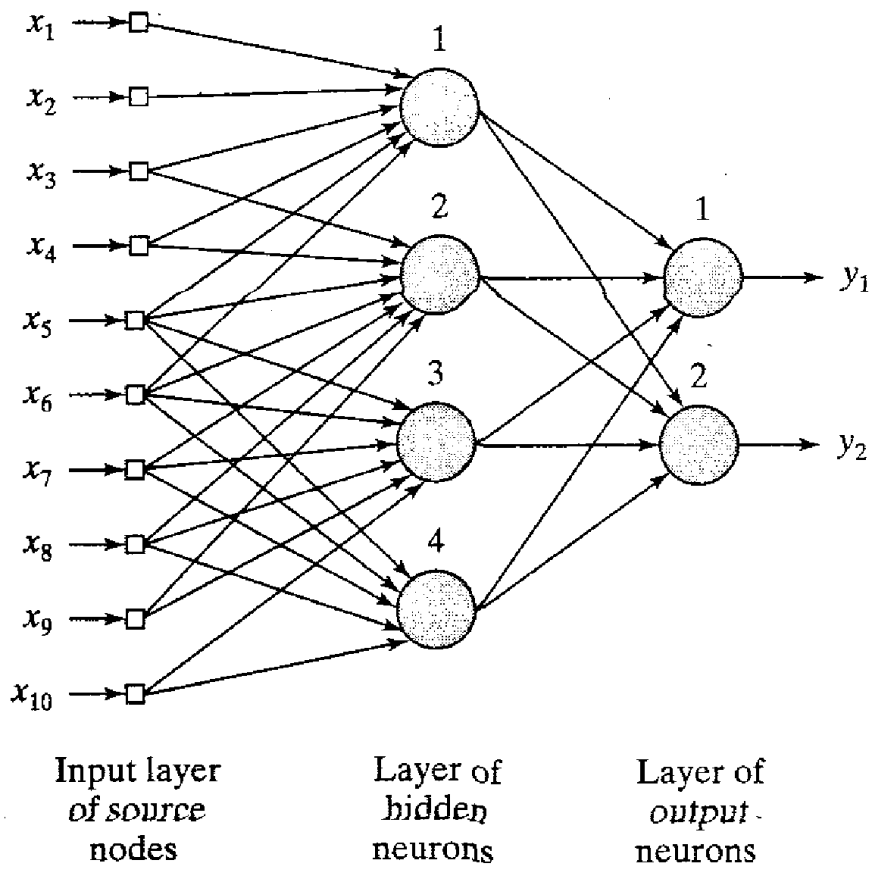
Sumário

Como incluir informação prévia no projeto de uma RNA

- Procedimento *ad-hoc* para construir informação prévia no *design* de uma RNA:
  1. *Restringir a arquitetura da rede* através do uso de conexões locais conhecidas como *campos receptivos*.
  2. *Limitar a escolha dos pesos sinápticos* através do uso do *compartilhamento de pesos*.
- Essas duas técnicas, especialmente a segunda, têm um efeito colateral benéfico: o número de parâmetros livres na rede é reduzido significativamente.
- Considere a rede *feedforward* parcialmente conectada da figura 4.
- Essa rede apresenta uma arquitetura restringida por construção.
- Os seis nós fonte superiores constituem o campo receptivo para o neurônio escondido 1.

Como incluir informação prévia no projeto de uma RNA

Figure: Ilustração do uso combinado do campo receptivo e compartilhamento de pesos [2].



Como incluir informação prévia no projeto de uma RNA

- Para satisfazer a restrição do compartilhamento de pesos, deve-se usar o mesmo conjunto de pesos sinápticos para cada um dos neurônios da camada escondida.
- Portanto, para seis conexões locais por neurônio escondido e um total de quatro neurônios escondidos (figura 4), pode-se expressar o campo local induzido do neurônio escondido j como (**soma de convolução**):

$$v_j = \sum_{i=1}^{6} w_i x_{i+j-1}, \qquad j = 1, 2, 3, 4 \tag{56}$$

onde $\{w_i\}_{i=1}^{6}$ constitui o mesmo conjunto de pesos compartilhados por todos os quatro neurônios escondidos e x_k é o sinal do nó fonte $k = i + j - 1$.

- É por essa razão que uma rede *feedforward* usando conexões locais e compartilhamento de pesos na forma descrita é conhecida como **rede convolucional**.

Convolução

- Foco no *layout* estrutural do MLP: classe especial chamada redes convolucionais.
- Uma rede convolucional é um MLP projetado para reconhecer formas bi-dimensionais com um alto grau de invariância para translação, mudança de escala, e outras formas de distorção.
- Esta tarefa difícil é aprendida de uma maneira supervisionada por uma rede cuja estrutura inclui as seguintes formas de *restrições*:
  1. *Extração de características*,
  2. *Mapeamento de características*,
  3. *Sub-amostragem*.

Convolução

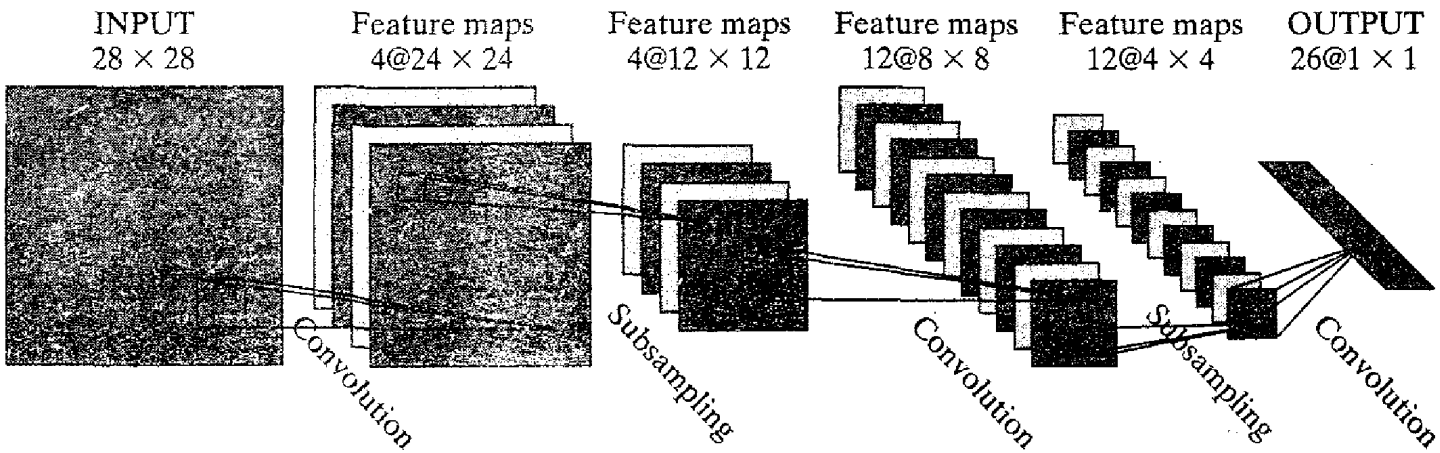
- Restrições:
  1. *Extração de características*: Cada neurônio obtém suas entradas sinápticas de um *campo receptivo* local da camada anterior, forçando-o a extrair características locais. A posição relativa de uma característica extraída em relação às outras é preservada.
  2. *Mapeamento de características*: Cada camada computacional da rede é composta de múltiplos *mapas de características*, onde cada mapa tem a forma de um plano no qual os neurônios individuais são *restringidos* para compartilhar o mesmo conjunto de pesos sinápticos. Efeitos benéficos:
     - *Invariância de deslocamento*: uso da *convolução* seguido de uma função sigmoide (achatamento).
     - *Redução no número de parâmetros livres*, acompanhado de *compartilhamento de pesos*.

Convolução

- Restrições (cont.):
  3. *Sub-amostragem*: cada camada convolucional é seguida por uma camada computacional que realiza *cálculo da média local* e *sub-amostragem*, onde a resolução do mapa de características é reduzida. Tem o efeito de reduzir a sensibilidade da saída do mapa a deslocamentos e outras formas de distorção.
- Todos os pesos em todas as camadas de uma rede convolucional são aprendidos através do treinamento.
- No entanto, a rede aprende a extrair suas próprias características automaticamente.

Convolução

- A figura 8 abaixo [2] mostra um *layout* arquitetural de uma rede convolucional com uma camada de entrada, quatro camadas escondidas e uma camada de saída.
- Esta rede foi projetada para realizar *processamento de imagem*: reconhecimento de caracteres manuscritos.



Convolução

- A camada de entrada, com 28×28 nós sensoriais, recebe as imagens de diferentes caracteres, centralizados e normalizados.
- Então, a computação alterna entre convolução e sub-amostragem:
  - 1^a escondida: convolução. 4 mapas de características com cada mapa consistindo de 24×24 neurônios. Cada neurônio tem um campo receptivo de tamanho 5×5.
  - 2^a escondida: sub-amostragem e média local. 4 mapas de características com cada mapa consistindo de 12×12 neurônios. Cada neurônio tem um campo receptivo de tamanho 2×2, um coeficiente treinável, um *bias* treinável e uma função de ativação sigmoide.
  - 3^a escondida: convolução. 12 mapas de características com cada mapa consistindo de 8×8 neurônios. Cada neurônio tem conexões sinápticas com vários mapas de características das camadas escondidas anteriores.

Convolução

- Computação (cont.):
  - 4^a escondida: sub-amostragem e média local. 12 mapas de características com cada mapa consistindo de 4×4 neurônios.
  - A camada de saída realiza um estágio final da convolução. Consiste de 26 neurônios, atribuídos a um dos 26 caracteres possíveis. Cada neurônio tem um campo receptivo de tamanho 4×4.
- Efeito "bipiramidal:" a cada camada convolucional ou de sub-amostragem, o número de mapas de características é aumentado enquanto que a resolução espacial é reduzida, comparada a camada anterior correspondente.
- O MLP da figura 8 contém 100.000 conexões sinápticas mas apenas 2.600 parâmetros livres.
- Esta redução dramática é conseguida através do uso de compartilhamento de pesos.

Convolução

- A capacidade da máquina de aprendizado é reduzida, que por sua vez, aumenta sua habilidade de generalização.
- Os ajustes dos parâmetros livres são feitos usando uma forma estocástica (sequencial) do aprendizado *back-propagation*.
- O uso de compartilhamento de pesos torna possível implementar a rede convolucional de forma paralela: outra vantagem sobre o MLP totalmente conectado.
- Duas lições (figura 8):
  1. Um MLP de tamanho gerenciável é capaz de aprender um mapeamento complexo, de alta dimensão e não-linear *restringindo* seu projeto através da incorporação de conhecimento prévio sobre a tarefa.
  2. Os níveis dos pesos sinápticos e *bias* podem ser aprendidos repetindo o algoritmo BP simples através do conjunto de treinamento.

Bibliografia I

[1] A. E. Bryson and Y.-C. Ho
Applied Optimal Control.
Blaisdell, New York, 1969.

[2] S. Haykin
Neural networks - a comprehensive foundation.
2nd. edition. Prentice Hall, 1999.

[3] W. S. McCulloch and W. Pitts
A logical calculus of the ideas immanent in nervous activity
Bulletin of Mathematical Biophysics, 5, pp. 115-133, 1943.

Bibliografia II

[4] R. A. F. Romero
SCC-5809 Redes Neurais.
Slides e listas de exercícios. Programa de Pós-Graduação em Ciência de Computação e Matemática Computacional. ICMC/USP, 2010.

[5] F. Rosenblatt
The perceptron: A perceiving and recognizing automaton.
Report 85-460-1, Project PARA, Cornell Aeronautical Lab., Ithaca, NY, 1957.

[6] D. E. Rumelhart, G. E. Hinton, and R. J. Williams
Learning representations of back-propagation errors.
Nature (London), vol. 323, pp. 533–536, 1986.