MULTILAYER PERCEPTRONS

PROF. RICARDO CERRI

Universidade Federal de São Carlos
Departamento de Computação - DC/UFSCar

Multilayer Perceptron

- Supera as limitações práticas do Perceptron
  - O modelo de cada neurônio inclui uma função de ativação não linear e diferenciável
  - Contém uma ou mais camadas escondidas entre a camada de entrada e a camada de saída
  - A rede possui alto grau de conectividade

Multilayer Perceptron

- Deficiências

  - Análise teórica difícil, pois há muitas conexões e funções não lineares

  - Muitos neurônios escondidos tornam difícil a visualização do processo de aprendizado

  - O aprendizado é mais difícil, pois há um espaço muito maior de possíveis funções. Há mais representações dos padrões de entrada

Multilayer Perceptron

- Como aprender? Back-propagation

  - **Forward phase**: pesos fixos e o sinal é propagado através da rede, camada por camada, até a saída

  - Mudanças só ocorrem nos potenciais de ativação e nas saídas dos neurônios da rede

Multilayer Perceptron

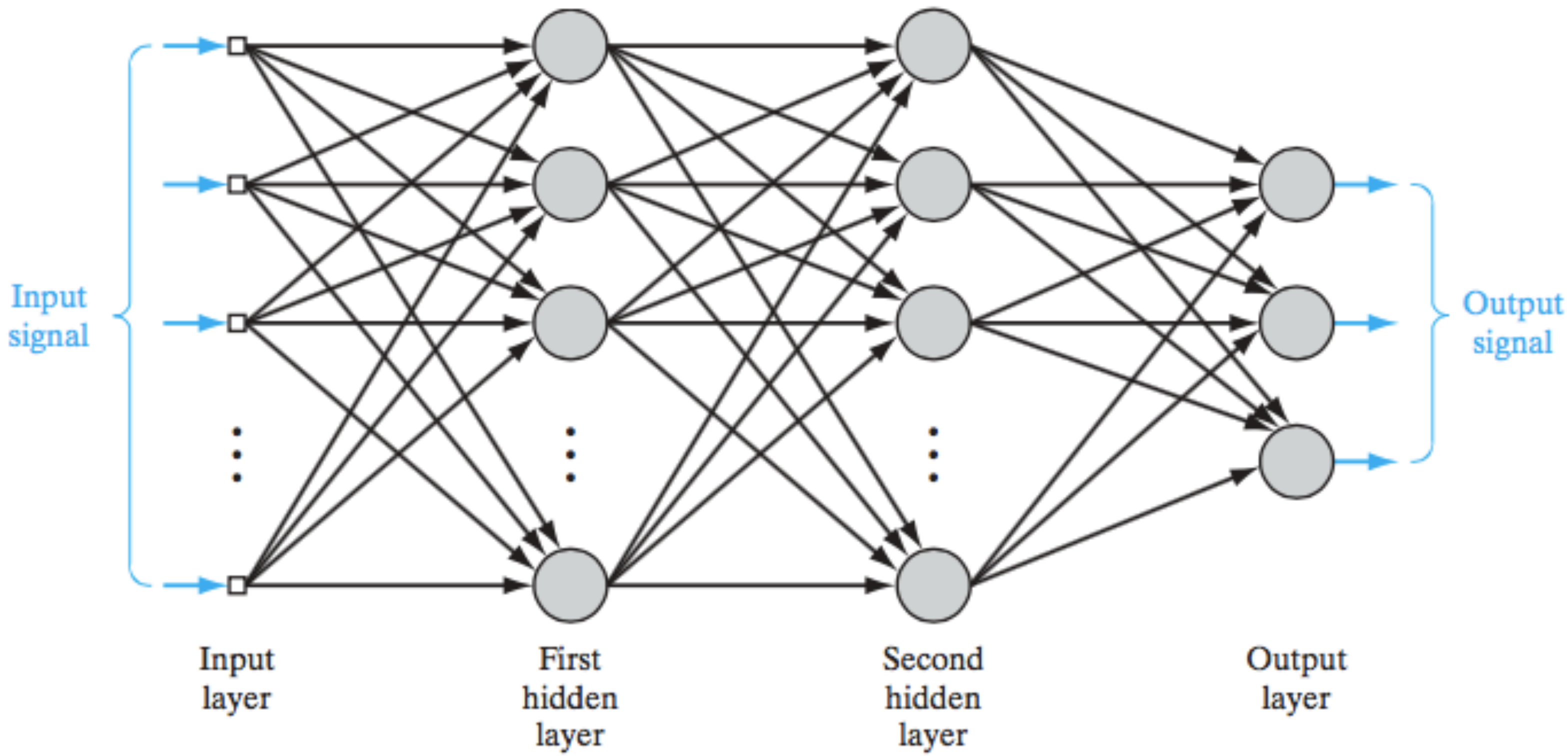
- Como aprender? Back-propagation
  - **Backward phase**: um sinal de erro é produzido comparando a saída desejada com a obtida
  - O erro é retropropagado através da rede, camada por camada
  - Ajustes são realizados nos pesos sinápticos da rede

Multilayer Perceptron

- O termo back-propagation se popularizou depois da publicação do livro *Parallel Distributed Processing (Rumelhart* e *McClelland, 1986)*

- O desenvolvimento do algoritmo Back-propagation representou um marco na área de redes neurais, pois forneceu um método computacionalmente eficiente para treinamento das multi-layer Perceptrons

Multilayer Perceptron



Multilayer Perceptron

- **Sinais de função:** sinal (estímulo) que vem da entrada da rede, é propagado neurônio a neurônio, e sai no fim da rede como uma sinal de saída
  - Presume-se que vá desempenhar uma função útil na saída da rede
  - Em cada neurônio, o sinal é calculado como uma função das entradas e pesos associados

Multilayer Perceptron

- **Sinais de erro:** sinal que tem origem na saída da rede (neurônio de saída) e é retropropagado camada à camada através da rede

  - Seu cálculo, para cada neurônio da rede, envolve uma função que depende do erro obtido na saída da rede

Multilayer Perceptron

- Sinais de função e sinais de erro

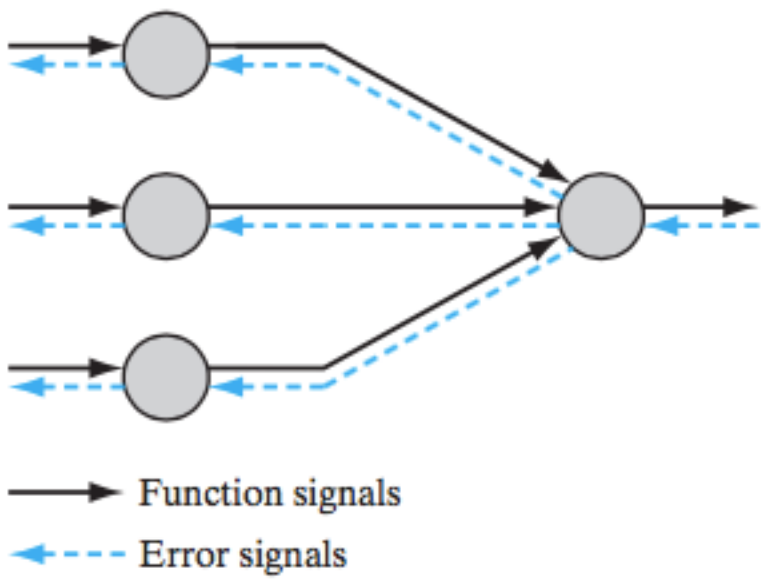


FIGURE 4.2 Illustration of the directions of two basic signal flows in a multilayer perceptron: forward propagation of function signals and back propagation of error signals.

Multilayer Perceptron

- Os neurônios de saída constituem a camada de saída da rede. Os neurônios restantes constituem as camadas escondidas

- Os neurônios escondidos não são parte nem da entrada, nem da saída da rede

  - A primeira camada escondida é alimentada com a saída da camada de entrada.

  - A saída resultante é então aplicada à segunda camada escondida, e assim em diante para toda a rede

Multilayer Perceptron

- Cada neurônio, de saída ou escondido, é desenvolvido para executar dois cálculos

  - Do sinal de função que aparece na saída de cada neurônio, expresso como uma função não linear contínua do sinal de entrada e pesos sinápticos associados

  - Da estimativa do vetor gradiente (gradientes da superfície do erro com respeito aos pesos conectados às entradas dos neurônios). Necessário para a fase de retropropagação

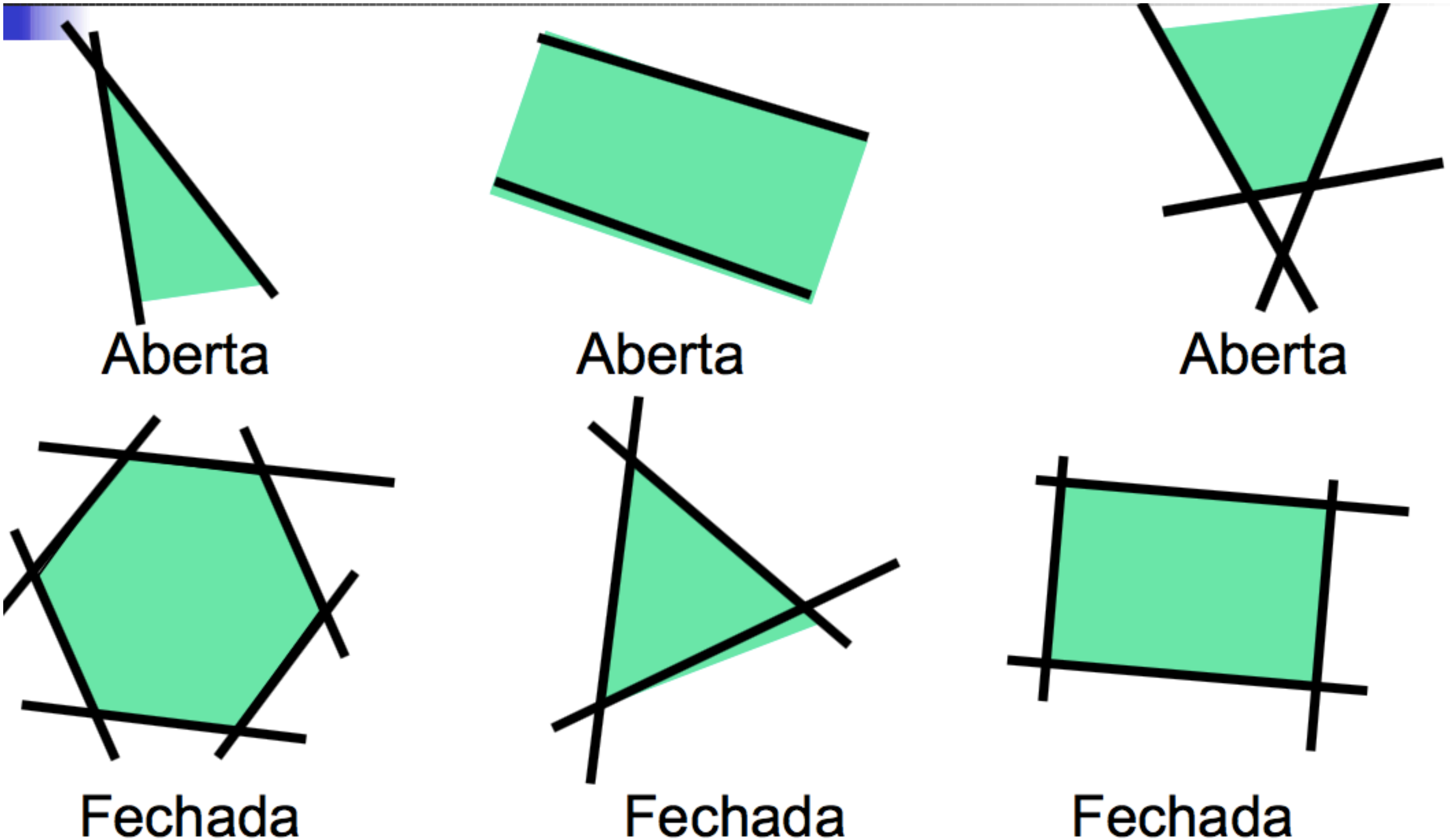
Multilayer Perceptron

- Função dos neurônios escondidos
  - Agem como detectores de atributos. Conforme o aprendizado progride, esses neurônios começam a descobrir os atributos que caracterizam os dados de treinamento
  - Isso é feito por meio da transformação não linear dos dados de entrada em um novo espaço chamado de espaço de características
  - Nesse novo espaço, classes (por exemplo em um problema de classificação) podem ser mais facilmente separadas umas das outras do que no espaço de entrada original

Multilayer Perceptron

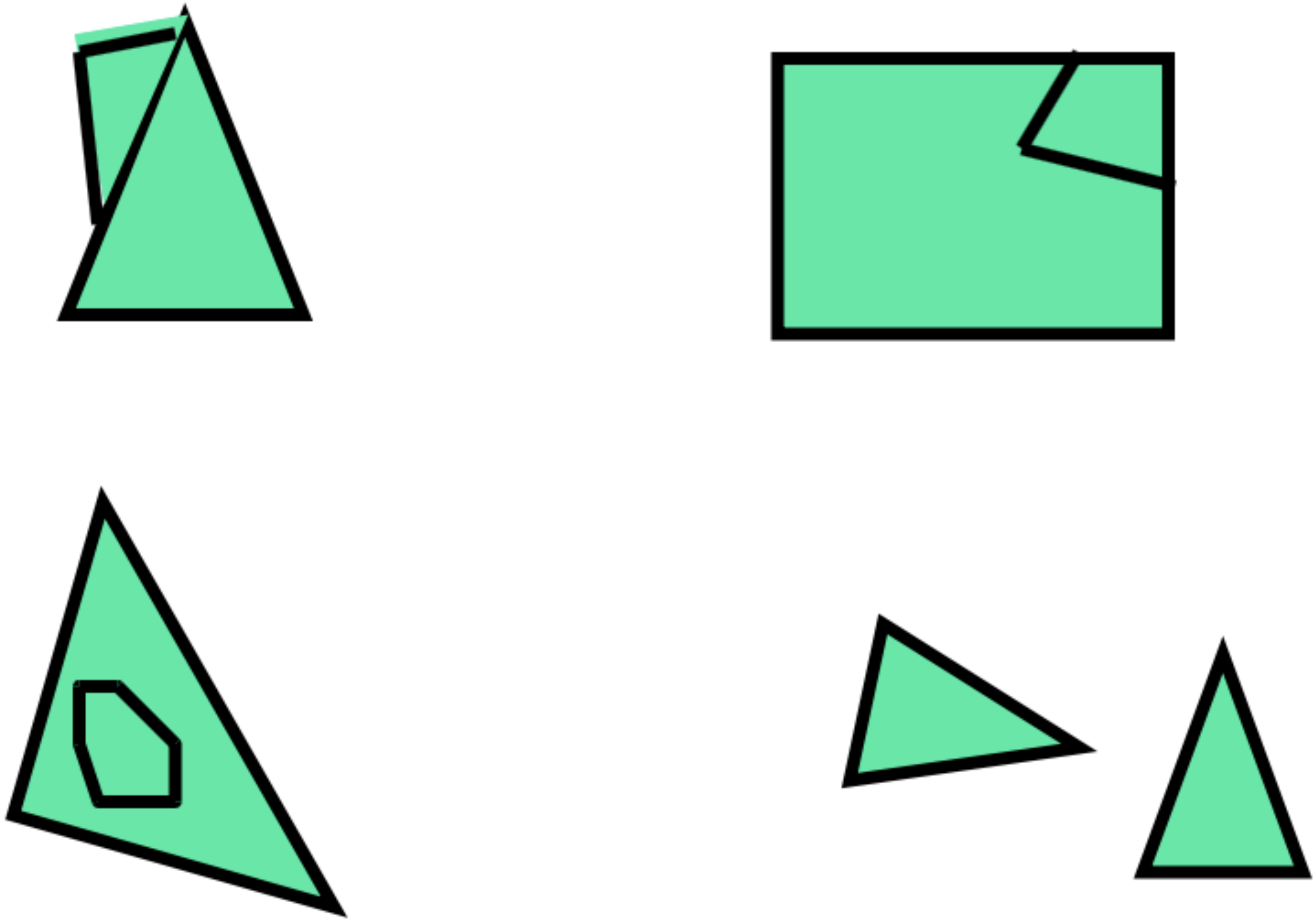
- Camadas intermediárias
  - Primeira camada: linhas retas no espaço de decisão
  - Segunda camada: combina as linhas da camada anterior para formar regiões convexas
  - Terceira camada: combina figuras convexas produzindo formatos abstratos

Multi-layer Perceptrons – Regiões Convexas

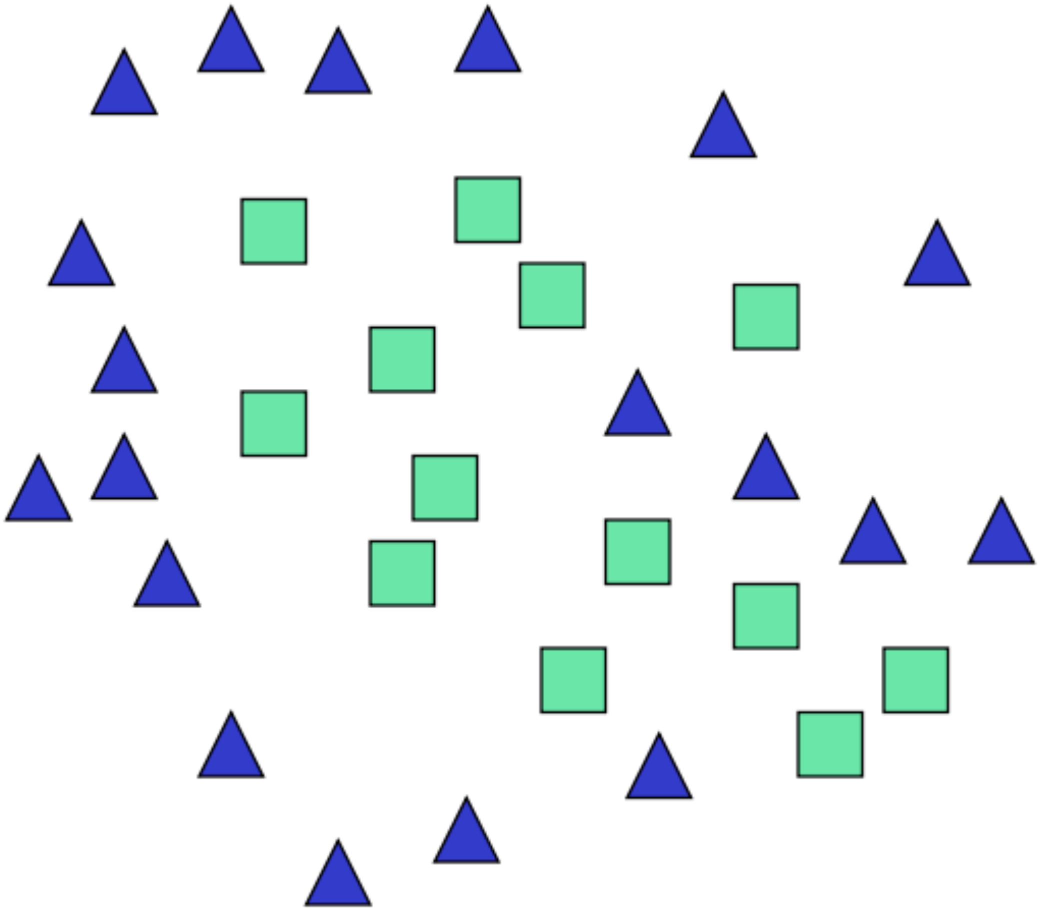


Multi-layer Perceptrons

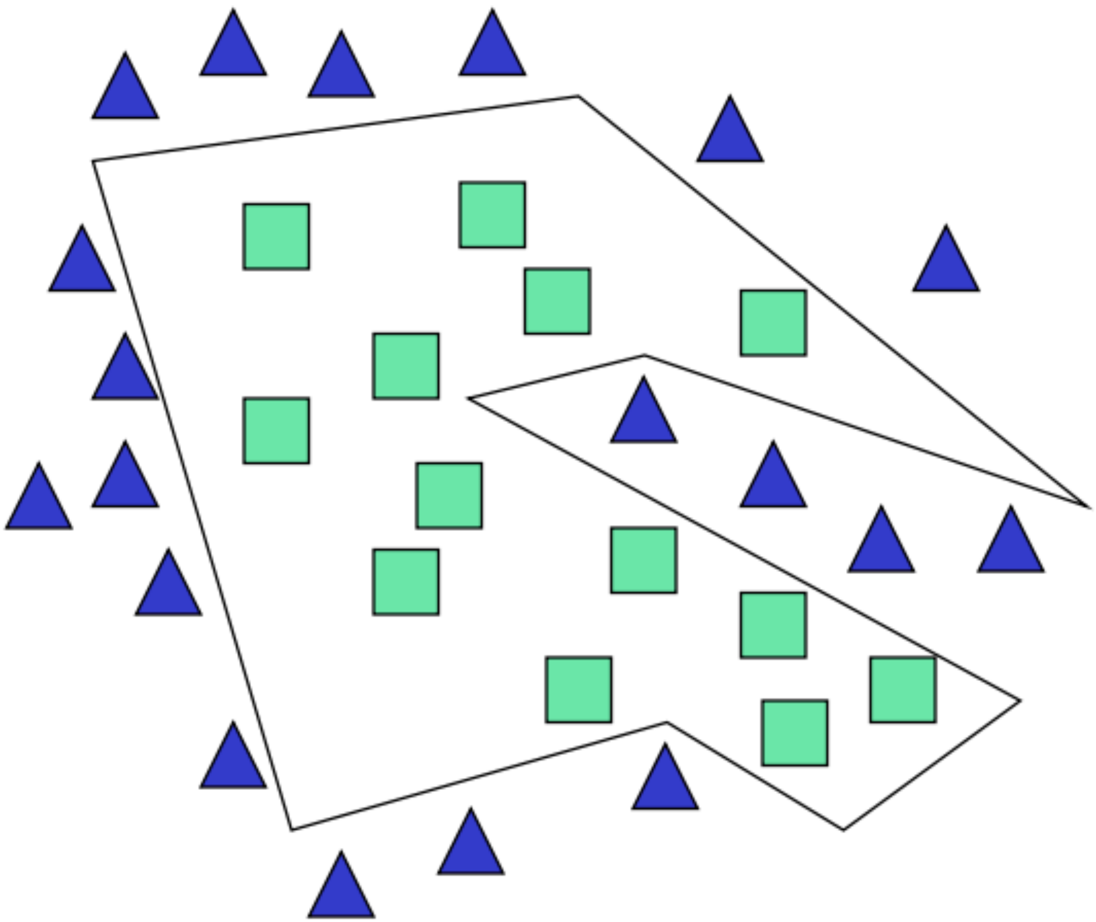
Combinações de Regiões Convexas



Multilayer Perceptron



Multilayer Perceptron



Multilayer Perceptron

- Considere um Multilayer Perceptron.

- Considere $\tau = \{\mathbf{x}(n), \mathbf{d}(n)\}_{n=1}^{N}$ um exemplo de treinamento. Seja $y_j(n)$ o sinal produzido na saída do neurônio j na camada de saída, estimulado por $\mathbf{x}(n)$ aplicado na camada de entrada

- O sinal de erro produzido na saída do neurônio j é dado por $e_j(n) = d_j(n) - y_j(n)$

Multilayer Perceptron

- O sinal de erro produzido na saída do neurônio j é dado por $e_j(n) = d_j(n) - y_j(n)$, em que $d_j(n)$ é o j-ésimo elemento do vetor de respostas desejadas $\mathbf{d}(n)$

- O erro instantâneo do neurônio j é dado por

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

Multilayer Perceptron

- Somando os erros de todos os neurônios da camada de saída, o erro total de toda a rede é dado por

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

- C é o conjunto de todos os neurônios de saída. Em um conjunto de treinamento com N exemplos, o erro médio sobre todos os exemplos (risco empírico) é dado por

$$\varepsilon_{av}(N) = \frac{1}{N} \sum_{n=1}^{N} \varepsilon(n) = \frac{1}{2N} \sum_{n=1}^{N} \sum_{j \in C} e_j^2(n)$$

Multilayer Perceptron

- **Modo de treinamento batch**: o ajuste dos pesos ocorre após a apresentação de todos os exemplos de treinamento a rede (uma época completa)

- Uma curva de aprendizado é obtida plotando $\mathcal{E}_{av}$ contra o número de épocas. Em cada época, os exemplos são embaralhados

- Realiza-se vários experimentos, cada um com valores diferentes para os parâmetros iniciais

Multilayer Perceptron

- Estimativa mais precisa do vetor de gradientes (derivada da função de custo $\mathcal{E}_{av}$ com relação ao vetor de pesos **w**), garantindo convergência para um mínimo local

- Paralelização do processo de aprendizado

Multilayer Perceptron

- De uma perspectiva prática, a busca deixa de ser estocástica por natureza, podendo ficar preso em mínimos locais

- Mais difícil detectar mudanças pequenas nos dados

- Quando há exemplos redundantes, não consegue tirar vantagem disso, pois ajusta pesos apenas após ver todos os exemplos

Multilayer Perceptron

- **Modo de treinamento online:** o ajuste dos pesos ocorre após a apresentação de cada exemplo. Pretende-se então minimizar o erro instantâneo de toda a rede $\varepsilon(n)$

- A busca no espaço de pesos multidimensional torna-se estocástica por natureza. Por isso também chamado método estocástico

- Menos susceptível a ficar preso em mínimos locais

Multilayer Perceptron

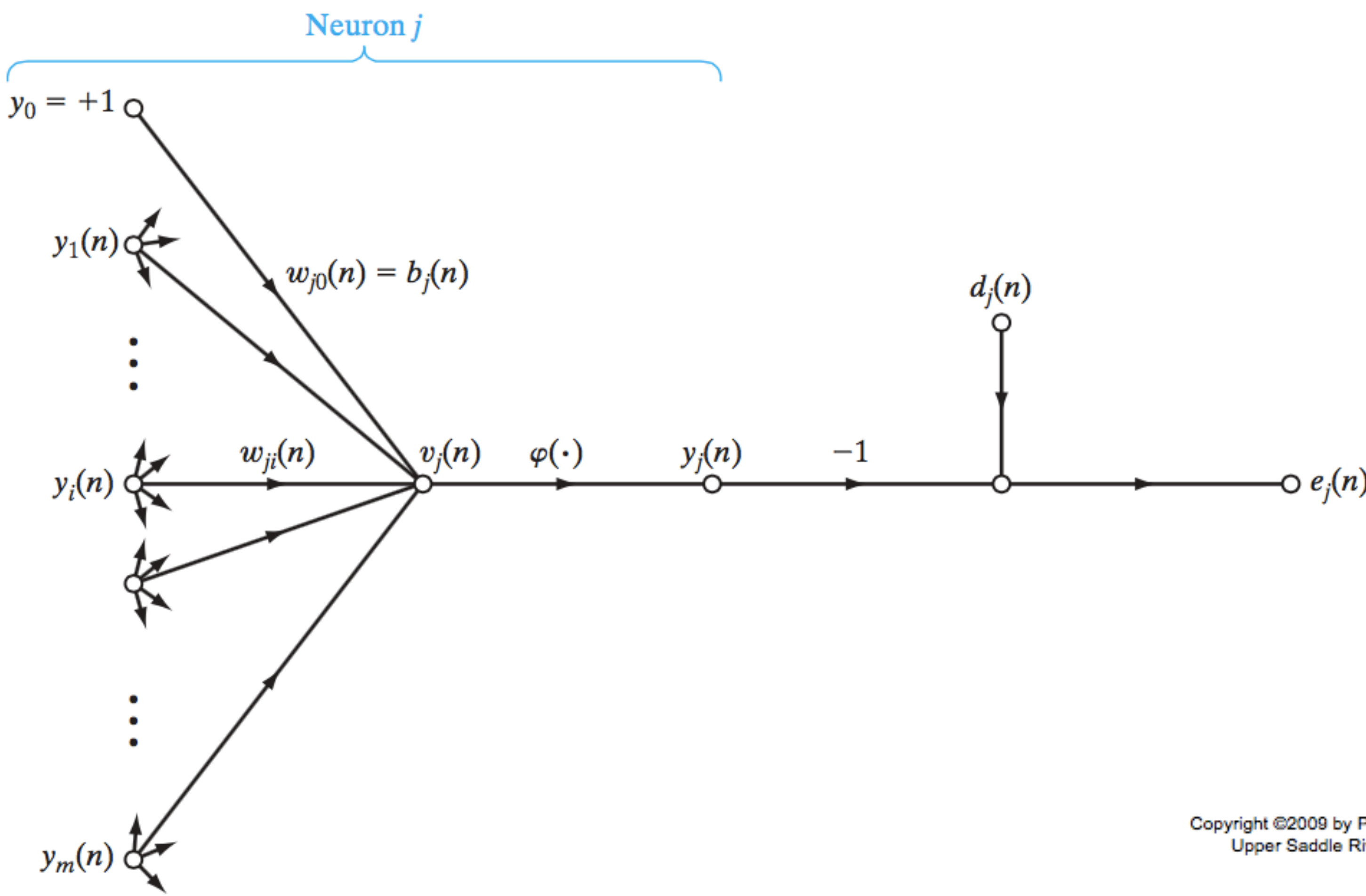
- Quando há redundância, tira melhor vantagem disso ao ajustar os pesos após a apresentação de cada exemplo

- Detecta melhor pequenas mudanças nos dados de treinamento

- Muito popular em problemas de classificação
  - Simples de implementar
  - Provê boas soluções para problemas difíceis

O Algoritmo Back-propagation

- Neurônio j sendo alimentado por um conjunto de sinais



O Algoritmo Back-propagation

- O potencial de ativação $v_j(n)$ produzido na entrada da função de ativação associada é dado por

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

  - m : número total de entradas (excluindo o bias)
  - w_{j0} : peso aplicado a entrada fixa $y_0=+1$ (bias)

O Algoritmo Back-propagation

- O sinal $y_j(n)$ na saída do neurônio j na iteração n é:

$$y_j(n) = \varphi_j(v_j(n))$$

- O algoritmo aplica um correção $\Delta w_{ji}(n)$ no peso sináptico $w_{ji}(n)$, proporcional a derivada parcial:

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

O Algoritmo Back-propagation

- A derivada parcial $\partial \varepsilon(n) / \partial w_{ji}(n)$ determina a direção da busca por w_{ji} no espaço de pesos

- Diferenciando a equação abaixo em ambos os lados com relação a $e_j(n)$:

$$\varepsilon(n) = \sum_{j \in C} e_j^2(n) \rightarrow \frac{\partial \varepsilon(n)}{\partial e_j(n)} = e_j(n)$$

O Algoritmo Back-propagation

- A derivada parcial $\partial \varepsilon(n) / \partial w_{ji}(n)$ determina a direção da busca por w_{ji} no espaço de pesos

- Diferenciando a equação abaixo em ambos os lados com relação a $y_j(n)$:

$$e_j(n) = d_j(n) - y_j(n) \rightarrow \frac{\partial e_j(n)}{\partial y_j(n)} = -1$$

O Algoritmo Back-propagation

- A derivada parcial $\partial \varepsilon(n) / \partial w_{ji}(n)$ determina a direção da busca por w_{ji} no espaço de pesos

- Diferenciando a equação abaixo em ambos os lados com relação a $v_j(n)$:

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

O Algoritmo Back-propagation

- A derivada parcial $\partial \varepsilon(n) / \partial w_{ji}(n)$ determina a direção da busca por w_{ji} no espaço de pesos

- Diferenciando a equação abaixo em ambos os lados com relação a $w_{ji}(n)$:

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

O Algoritmo Back-propagation

- A derivada parcial $\partial \varepsilon(n) / \partial w_{ji}(n)$ determina a direção da busca por w_{ji} no espaço de pesos

- Substituindo as equações obtidas na equação da regra da cadeia, obtemos

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

O Algoritmo Back-propagation

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

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

  - η : taxa de aprendizado do algoritmo
  - O sinal negativo refere-se ao gradiente descendente no espaço de pesos

- Usando a equação do slide 34 na equação acima:

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

O Algoritmo Back-propagation

- O gradiente local $\delta_j(n)$ é definido por

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

$$= -\frac{\partial \varepsilon(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)}$$

$$= e_j(n)\varphi_j'(v_j(n))$$

- O gradiente local define a mudança necessária nos pesos. Ele é dados pelo produto do erro e da derivada da função de ativação do neurônio

O Algoritmo Back-propagation

- O sinal de erro do neurônio de saída é o fator chave no cálculo do ajuste dos pesos

- Assim, dois casos para o cálculo do erro podem ser identificados

  - O neurônio está localizado na última camada (saída)

  - O neurônio está localizado em uma camada escondida

O Algoritmo Back-propagation

- Neurônio j é um neurônio de saída
  - Nesse caso, o neurônio está diretamente associado com a saída desejada. Assim, calcula-se o erro diretamente:

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

  - Tendo calculado o erro, o gradiente local é calculado de maneira direta:

$$\delta_j(n) = e_j(n)\varphi_j'(v_j(n))$$

O Algoritmo Back-propagation

- Neurônio j é um neurônio escondido
  - Nesse caso, não há saída desejada específica associada ao neurônio.
  - O sinal de erro deve ser calculado recursivamente, em termos dos sinais de erro de todos os neurônios os quais o neurônio j está diretamente conectado
  - Nesse ponto o desenvolvimento do back-propagation torna-se complicado

O Algoritmo Back-propagation

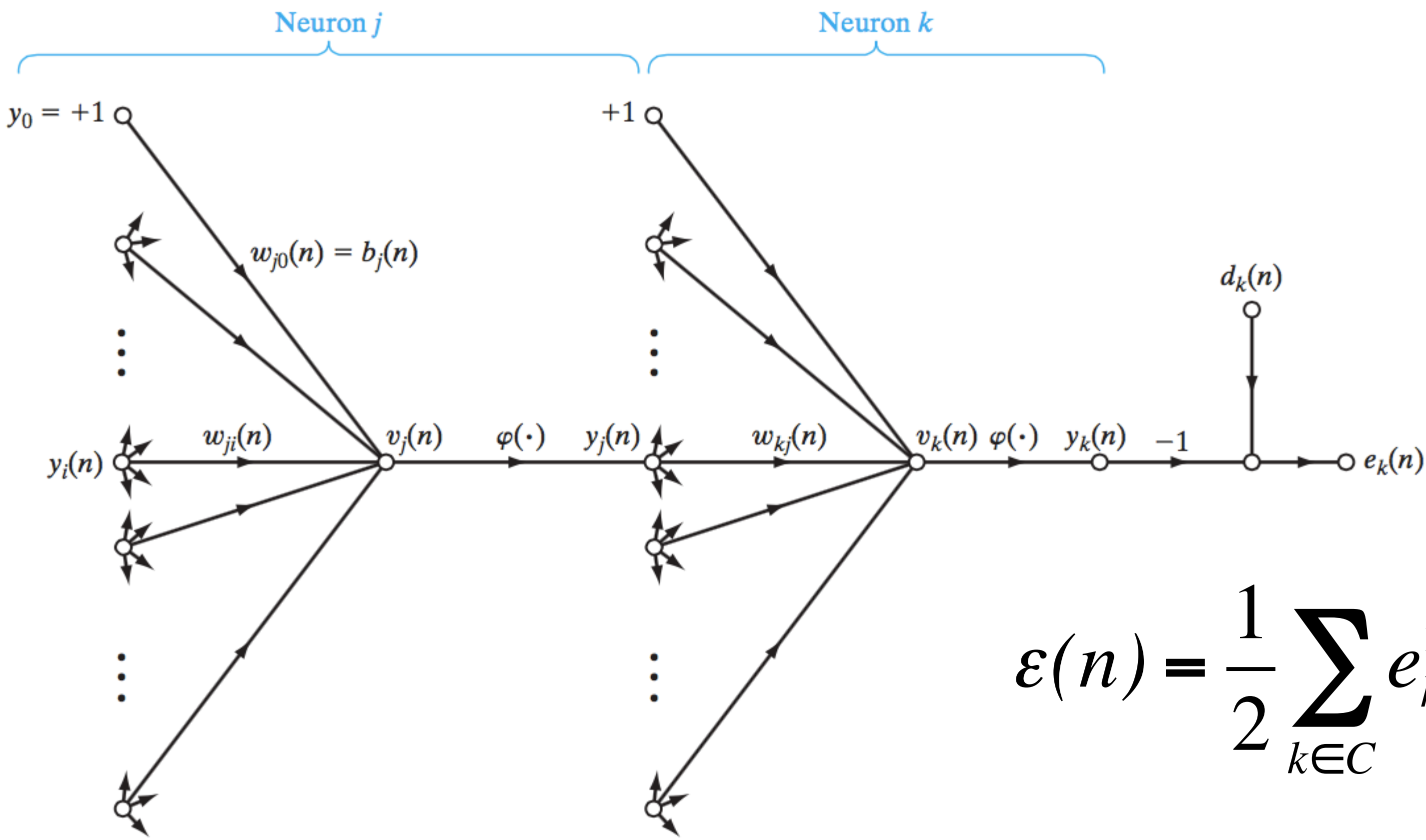
- Neurônio j é um neurônio escondido
  - Podemos redefinir o cálculo do gradiente (Slide 36), com o neurônio j sendo um neurônio escondido

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

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

O Algoritmo Back-propagation

- Neurônio j é um neurônio escondido



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

O Algoritmo Back-propagation

- Neurônio j é um neurônio escondido
  - Diferenciando $\varepsilon(n) = \frac{1}{2}\sum_{k \in C} e_k^2(n)$ com relação a $y_j(n)$:

$$\frac{\partial \varepsilon(n)}{\partial y_j(n)} = \sum_{k \in C} e_k \frac{\partial e_k(n)}{\partial y_j(n)}$$

O Algoritmo Back-propagation

- Neurônio j é um neurônio escondido

  - Aplicando a regra da cadeia em $\partial e_k(n)/\partial y_j(n)$:

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

  - Da figura do slide 41, temos que (neurônio k é de saída):

$$\begin{aligned} e_k(n) &= d_k(n) - y_k(n) \\ &= d_k(n) - \varphi_k(v_k(n)) \end{aligned}$$

O Algoritmo Back-propagation

- Neurônio j é um neurônio escondido
  - Assim, podemos escrever a derivada da função de ativação para o neurônio k

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

O Algoritmo Back-propagation

- Neurônio j é um neurônio escondido
  - Da figura do slide 41, temos que o potencial de ativação do neurônio k é dado por:

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

  - m : número total de entradas (excluindo o bias)
  - $w_{k0}(n)$ é igual ao bias $b_k(n)$ aplicado ao neurônio k, com entrada fixa igual a +1

O Algoritmo Back-propagation

- Neurônio j é um neurônio escondido
  - Derivada da equação do slide 45 com relação a $y_j(n)$:

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

O Algoritmo Back-propagation

- Neurônio j é um neurônio escondido
  - Temos $$\frac{\partial \varepsilon(n)}{\partial y_j(n)} = \sum_{k \in C} e_k \frac{\partial e_k(n)}{\partial v_k(n)} \frac{\partial v_k(n)}{\partial y_j(n)}$$

$$\frac{\partial e_k(n)}{\partial v_k(n)} = -\varphi_k'(v_k(n)) \qquad \frac{\partial v_k(n)}{\partial y_j(n)} = w_{kj}(n)$$

  - O que resulta em

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

O Algoritmo Back-propagation

- Neurônio j é um neurônio escondido
  - Fazendo a substituição, temos então o gradiente local do neurônio j

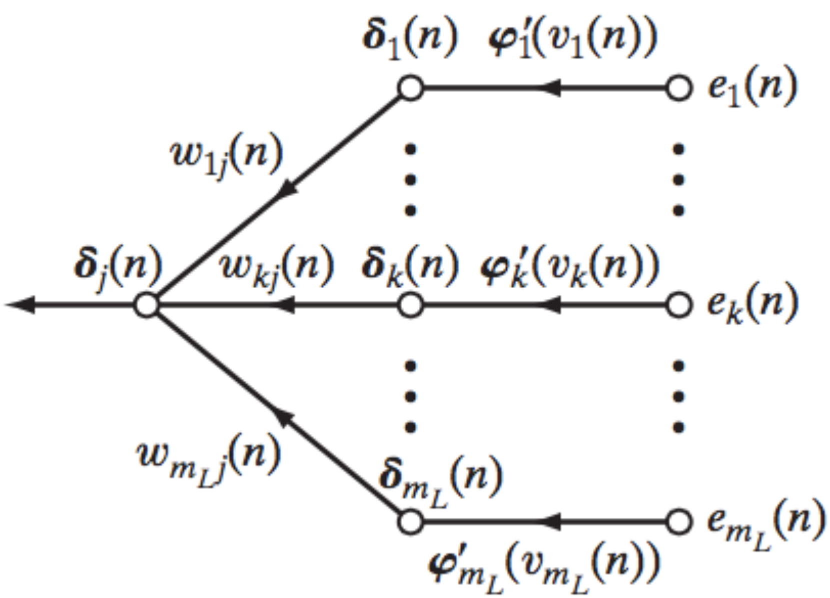
$$\delta_j(n) = -\frac{\partial \varepsilon(n)}{\partial y_j(n)} \varphi'_j(v_j(n)) \qquad \frac{\partial \varepsilon(n)}{\partial y_j(n)} = -\sum_k \delta_k(n) w_{kj}(n)$$

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

O Algoritmo Back-propagation

- Neurônio j é um neurônio escondido

  - Representação gráfica de $\delta_j(n) = \varphi'_j(v_j(n)) \sum_k \delta_k(n) w_{kj}(n)$



O Algoritmo Back-propagation

- **Resumindo:** a correção aplicada nos pesos conectando um neurônio i com um neurônio j é dada pela regra delta

$$\begin{pmatrix} Correção \\ pesos \\ \Delta w_{ji}(n) \end{pmatrix} = \begin{pmatrix} Taxa \\ aprendizado \\ \eta \end{pmatrix} \times \begin{pmatrix} Gradiente \\ local \\ \partial_j(n) \end{pmatrix} \times \begin{pmatrix} Sinal\ entrada \\ neurônio\ j \\ y_i(n) \end{pmatrix}$$

- **Saída:** $\delta_j(n)$ é igual ao produto da derivada $\varphi'_j(v_j(n))$ e do sinal de erro $e_j(n)$, ambos associados ao neurônio j

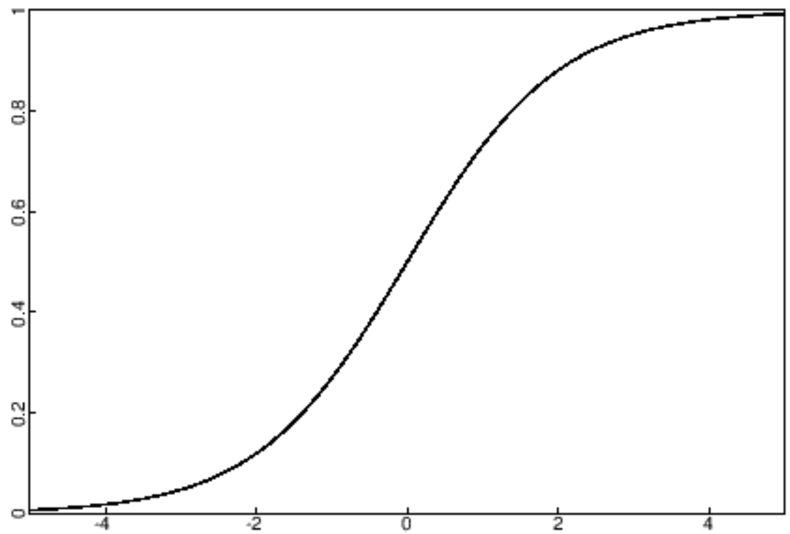
- **Escondido:** $\delta_j(n)$ é igual ao produto da derivada associada $\varphi'_j(v_j(n))$ e da soma ponderada dos δs calculados para os neurônios da próxima camada, ou da camada de saída, conectados ao neurônio j

Funções de Ativação

- Para calcular o δ para cada neurônio, precisamos conhecer a derivada da função de ativação $\varphi(\cdot)$ associada ao neurônio
- Para existir a derivada, $\varphi(\cdot)$ deve ser contínua
- Assim, ser **diferenciável** é o único requisito para a função de ativação
- Função contínua diferenciável comumente utilizada: **sigmoidal**

Funções de Ativação

- **Função logística:** $\varphi_j(v_j(n)) = \dfrac{1}{1 + exp(-av_j(n))}, \qquad a > 0$
  - Amplitude do sinal de saída: $0 \leq y_j \leq 1$



Funções de Ativação

- **Função logística:** $\varphi_j(v_j(n)) = \dfrac{1}{1+exp(-av_j(n))}, \qquad a > 0$
  - Amplitude do sinal de saída: $0 \leq y_j \leq 1$
  - A derivada da função com relação a $v_j(n)$ fornece:

$$\varphi_j'(v_j(n)) = \frac{a \;\; exp(-av_j(n))}{\left[1+exp(-av_j(n))\right]^2}$$

  - Com $y_j(n) = \varphi_j(v_j(n))$, podemos reescrever a derivada:

$$\varphi_j'(v_j(n)) = ay_j(n)\left[1 - y_j(n)\right]$$

Funções de Ativação

$$\varphi'_j(v_j(n)) = ay_j(n)\left[1 - y_j(n)\right]$$

- Para um neurônio j da camada de saída, $y_j(n) = o_j(n)$. O gradiente local do neurônio j é dado por:

$$\delta_j(n) = e_j(n)\varphi'_j(v_j(n))$$

$$= a\left[d_j(n) - o_j(n)\right]o_j(n)\left[1 - o_j(n)\right]$$

Funções de Ativação

$$\varphi_j'(v_j(n)) = ay_j(n)\left[1 - y_j(n)\right]$$

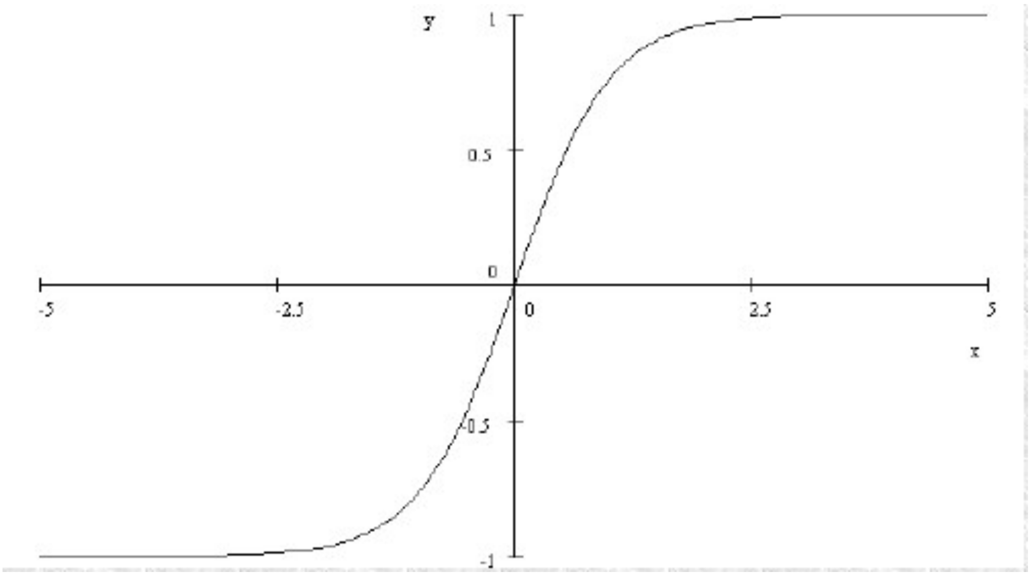
- Para um neurônio j de uma camada escondida, o gradiente local do neurônio j é dado por:

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

$$= ay_j(n)\left[1 - y_j(n)\right]\sum_k \delta_k(n)w_{kj}(n)$$

Funções de Ativação

- **Função tangente hiperbólica:** $\varphi_j(v_j(n)) = a\,tanh(bv_j(n))$
  - *a* e *b* são constantes positivas
  - Amplitude do sinal de saída: $-a \leq y_j \leq a$



Funções de Ativação

- **Função tangente hiperbólica:** $\varphi_j(v_j(n)) = a\, tanh(bv_j(n))$
  - *a* e *b* são constantes positivas
  - Amplitude do sinal de saída: $-1 \leq y_j \leq 1$
  - A derivada da função com relação a $v_j(n)$ fornece:

$$\varphi'_j(v_j(n)) = ab\ sech^2(bv_j(n))$$

$$= ab\left(1 - tanh^2(bv_j(n))\right)$$

$$= \frac{b}{a}\left[a - y_j(n)\right]\left[a + y_j(n)\right]$$

Funções de Ativação

$$\varphi_j'(v_j(n)) = \frac{b}{a}\left[a - y_j(n)\right]\left[a + y_j(n)\right]$$

- Para um neurônio j da camada de saída, o gradiente local do neurônio j é dado por:

$$\delta_j(n) = e_j(n)\varphi_j'(v_j(n))$$

$$= \frac{b}{a}\left[d_j(n) - o_j(n)\right]\left[a - o_j(n)\right]\left[a + o_j(n)\right]$$

Funções de Ativação

$$\varphi_j'(v_j(n)) = \frac{b}{a}\left[a - y_j(n)\right]\left[a + y_j(n)\right]$$

- Para um neurônio j de uma camada escondida, o gradiente local do neurônio j é dado por:

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

$$= \frac{b}{a}\left[a - y_j(n)\right]\left[a + y_j(n)\right]\sum_k \delta_k(n) w_{kj}(n)$$

Taxa de Aprendizado

- Quanto menor a taxa de aprendizado, menor a mudança nos pesos sinápticos da rede e mais suave será a trajetória no espaço de busca
  - O aprendizado será mais lento

- Aumentando a taxa de aprendizado, temos um aprendizado mais rápido, com grandes mudanças nos pesos sinápticos
  - A rede pode se tornar instável

Taxa de Aprendizado

- Como aumentar a taxa de aprendizado sem perder estabilidade?
  - Constante de momentum α : número positivo

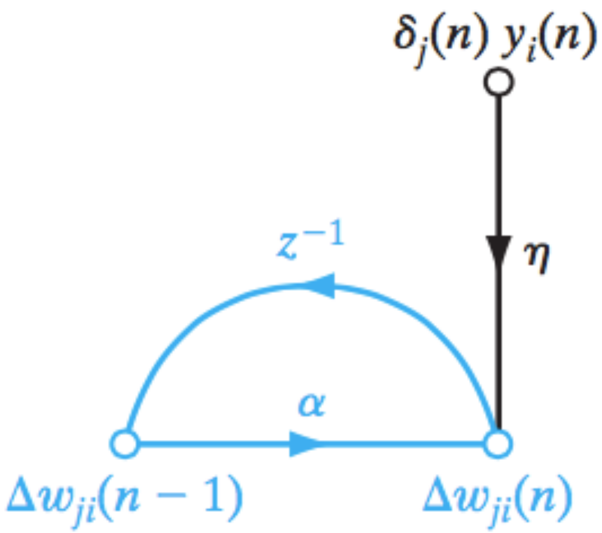
$$\Delta w_{ji}(n) = \alpha \Delta w_{ji}(n-1) + \eta \delta_j(n) y_i(n)$$

  - Controla o ajuste $\Delta w_{ji}(n)$

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

$$\delta_j(n) = e_j(n)\varphi_j'(v_j(n))$$

$$-\frac{\partial \varepsilon(n)}{\partial w_{ji}(n)} = \delta_j(n)y_i(n)$$



Taxa de Aprendizado

- Quando a derivada parcial $\partial \varepsilon(n) / \partial w_{ji}(n)$ tem o mesmo sinal em iterações consecutivas, $\Delta w_{ji}(n)$ cresce, e $w_{ji}(n)$ é ajustado de um quantidade grande. Assim a constante de momentum acelera o algoritmo em regiões de descida constante na superfície de erro

- Se a derivada parcial $\partial \varepsilon(n) / \partial w_{ji}(n)$ tem sinais opostos em iterações consecutivas, $\Delta w_{ji}(n)$ encolhe, e $w_{ji}(n)$ é ajustado em quantidade pequena. Assim a constante de momentum tem defeito estabilizador em direções nas quais o sinal oscila.

Critérios de Parada

- Em geral não se pode mostrar que o Back-propagation convergiu, e não há critérios de parada bem definidos

- 1 – Seja o vetor de pesos $\mathbf{w}^*$ um mínimo. Condição: vetor gradiente $\mathbf{g}(\mathbf{w})$ deve ser zero quando $\mathbf{w} = \mathbf{w}^*$
  - É considerada convergência quando a norma Euclidiana do vetor gradiente alcançar um valor suficientemente pequeno

- 2 – Usar a função de custo ε_{AV}
  - É considerada convergência quando a diminuição no erro quadrático médio, for suficientemente pequena

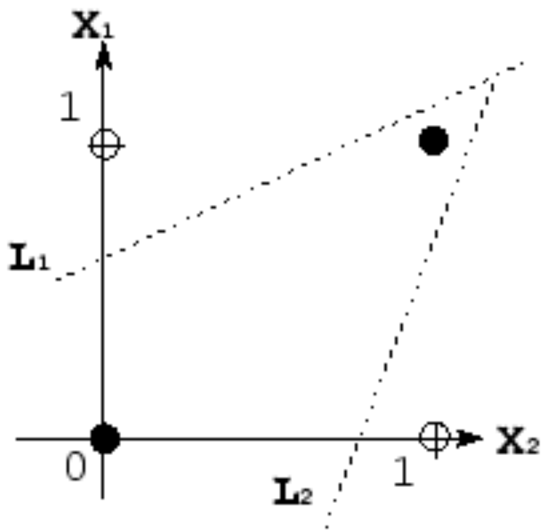
- 3 – A cada iteração, testa a generalização da rede

O Problema XOR

- O Perceptron de Rosenblatt não pode classificar padrões não linearmente separáveis

X_1	X_2	Y
0	0	0
0	1	1
1	0	1
1	1	0

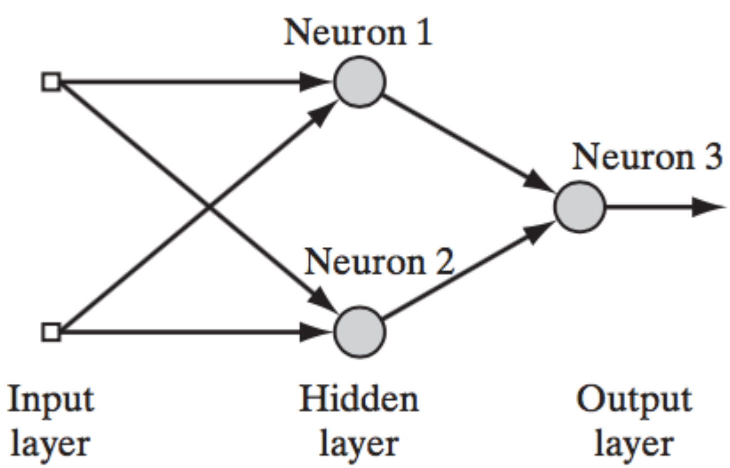
$$Y = X_1 \oplus X_2$$

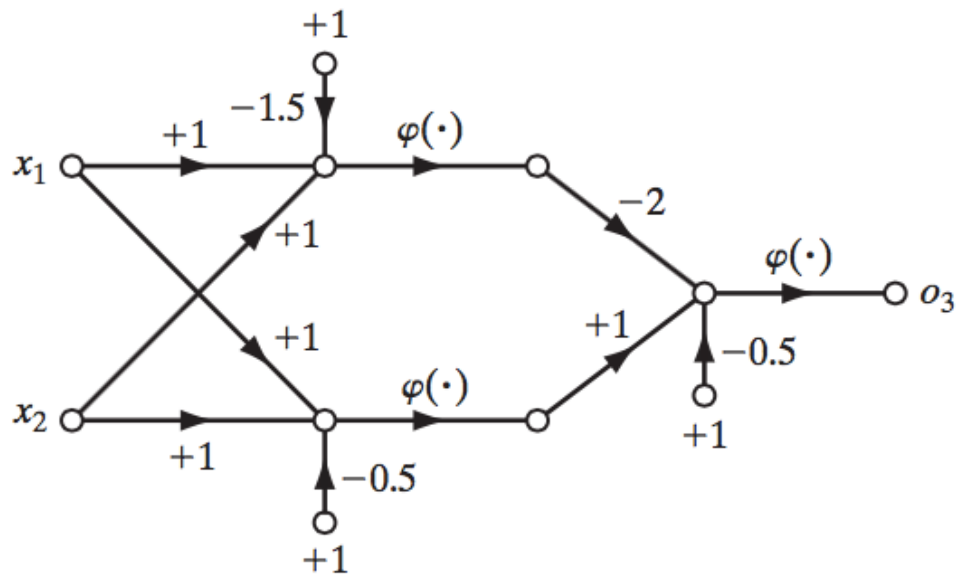


  - Um único Perceptron consegue traçar apenas um hiperplano

O Problema XOR

- Pode-se resolver o problema usando uma camada escondida com dois neurônios





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

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

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

O Problema XOR

- Pode-se resolver o problema usando uma camada escondida com dois neurônios

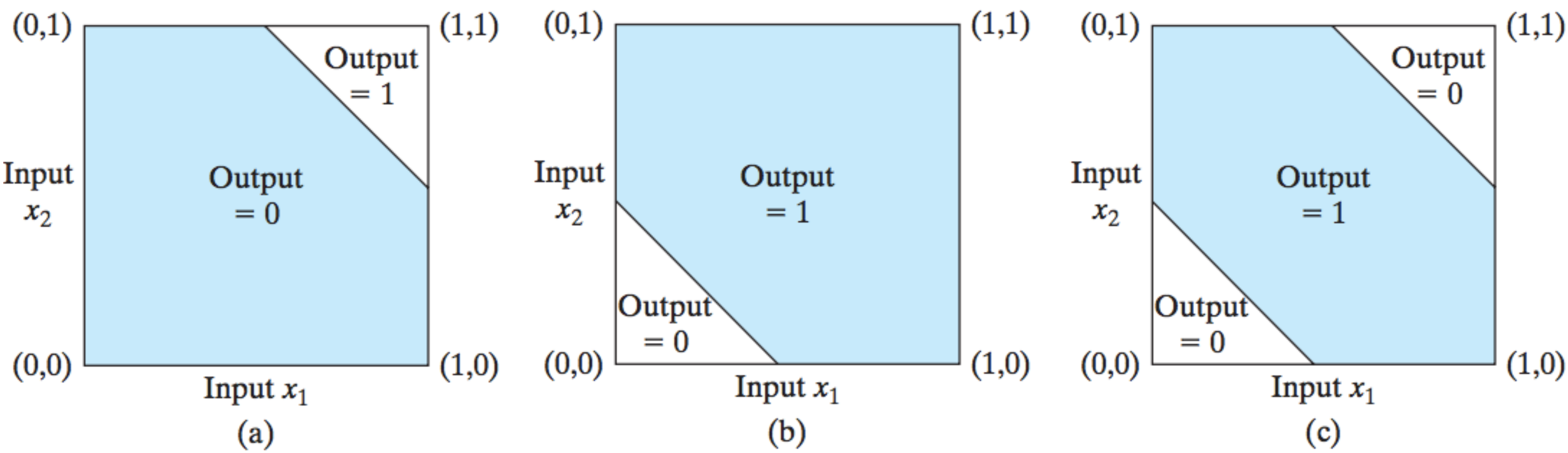


FIGURE 4.9 (a) Decision boundary constructed by hidden neuron 1 of the network in Fig. 4.8. (b) Decision boundary constructed by hidden neuron 2 of the network. (c) Decision boundaries constructed by the complete network.

Heurísticas para Melhor Desempenho

- O **modo online** é computacionalmente mais rápido que o modo batch. Verdade quando há muitos dados e redundância

- **Maximização da informação**: apresentar à rede, consecutivamente, exemplos bem diferentes (randomização dos exemplos)

- **Saída desejada**: os valores devem ser escolhidos para ficarem entre os limiares da função de ativação

Heurísticas para Melhor Desempenho

- **Normalização**: os atributos podem ser pré-processados para possuírem média próxima de 0
  - Melhora a convergência
  - Importante considerando os parâmetros (bias e pesos)
  - Bias: distância do hiperplano à origem
  - Pesos: orientação do hiperplano