

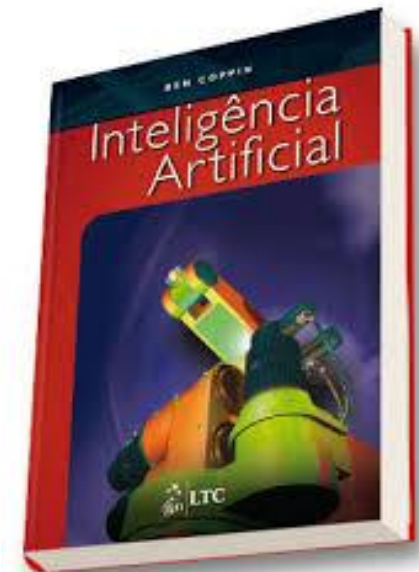
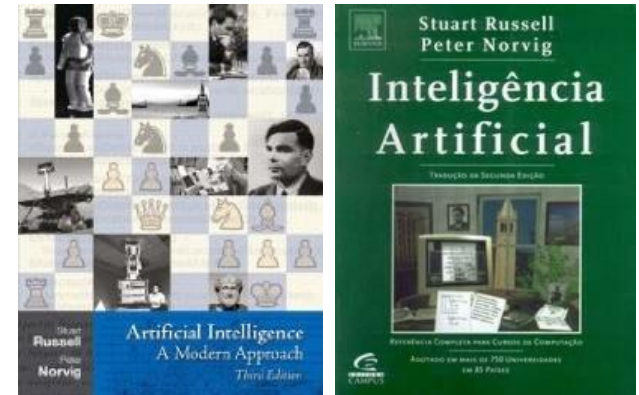
Tópicos Especiais: Inteligência Artificial

REDES NEURAIS

Material baseado e adaptado do Cap. 20 do Livro
Inteligência Artificial de Russell & Norvig

Bibliografia

- Inteligência Artificial
 - Russell & Norvig
 - Site: <http://aima.cs.berkeley.edu>
- Inteligência Artificial, Ben Coppin.



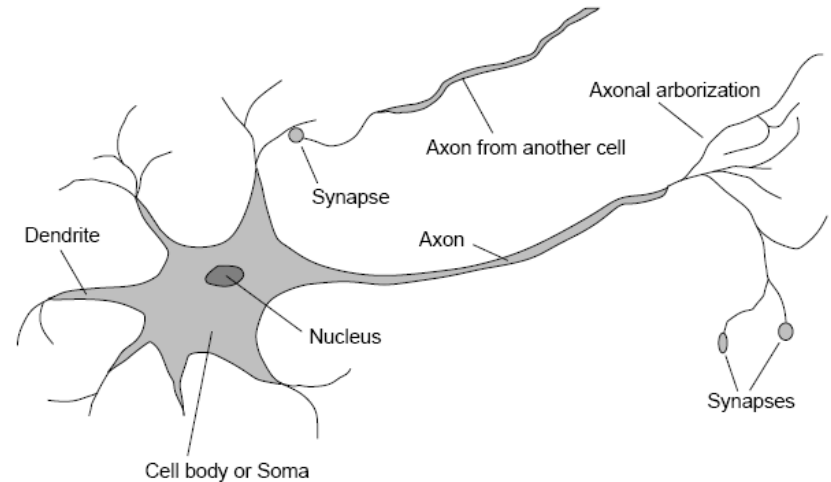
Métodos Estatísticos de Aprendizagem

Capítulo 20 – Russell & Norvig

Seção 20.5

Redes Neurais

- Um **neurônio** é uma célula no cérebro cuja principal função é coletar, processar e disseminar sinais elétricos.
 - A capacidade de processamento de informações no cérebro emerge de **redes** de neurônios.
- Parte do trabalho inicial em IA teve como objetivo tentar criar **redes neurais artificiais**.



Cérebro:

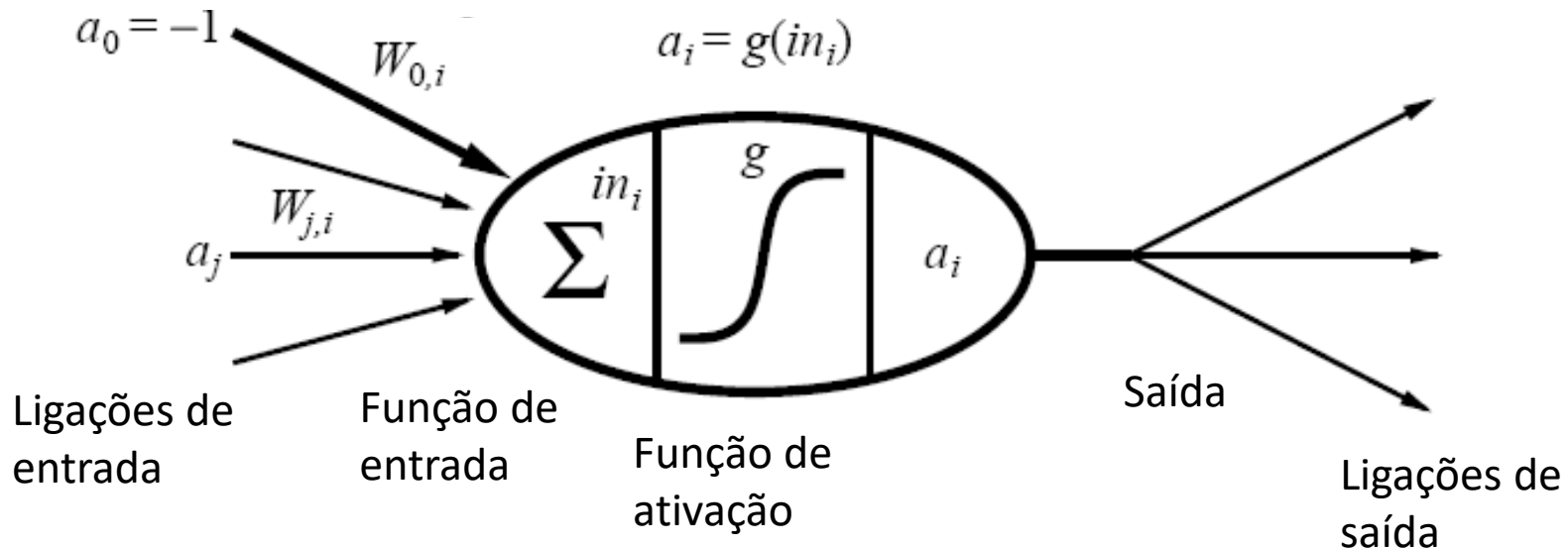
10^{11} neurônios de 20 tipos diferentes

10^{14} sinapses

Sinais são sequências de pulsos elétricos com ciclos de 1ms-10ms

“Neurônio” de McCulloch e Pitts (1943)

$$a_i \leftarrow g(in_i) = g(\sum_j W_{j,i} a_j)$$

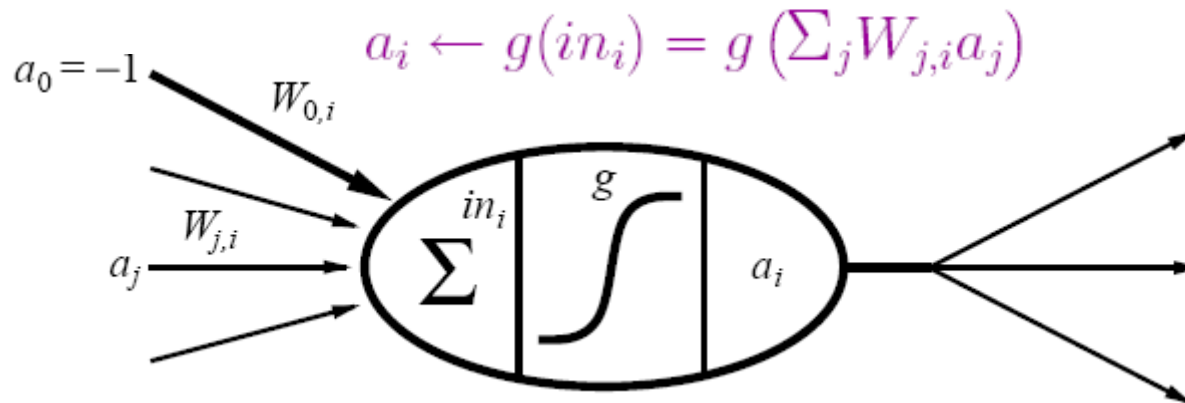


O neurônio “dispara” quando uma combinação linear de suas entradas excede algum limiar.

Modelo simplificado de um neurônio real, serve para estudar propriedades abstratas de redes neurais.

A área de neurociência computacional estuda modelos mais detalhados.

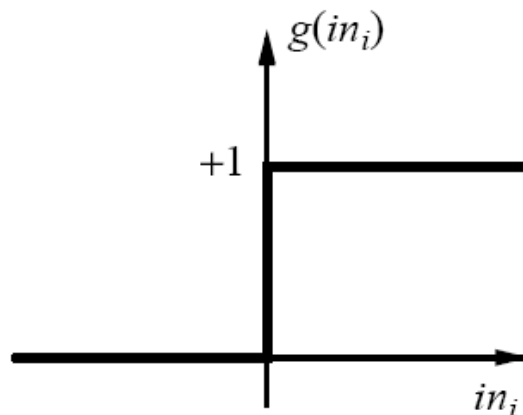
Unidades em redes neurais



- Uma ligação da unidade j para unidade i serve para propagar a ativação a_j desde j até i .
- Cada ligação tem um peso numérico $W_{j,i}$, que determina a sua intensidade.
- Cada unidade calcula uma soma ponderada de suas entradas.
 - Depois aplica uma função de ativação g a essa soma.
- O peso $W_{0,i}$ é conectado a uma entrada fixa $a_0 = -1$ para permitir a representação de funções que não passem pela origem.

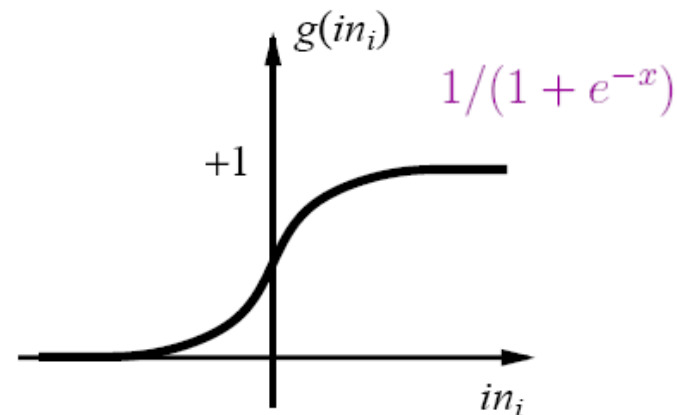
Funções de Ativação

- A função de ativação tem que atender a dois critérios:
 - Deve estar ativa (próxima de +1) para entradas “corretas” e inativa (próxima a 0) para entradas “erradas”.
 - Deve ser não linear para que a rede como um todo possa representar funções não-lineares.



(a)

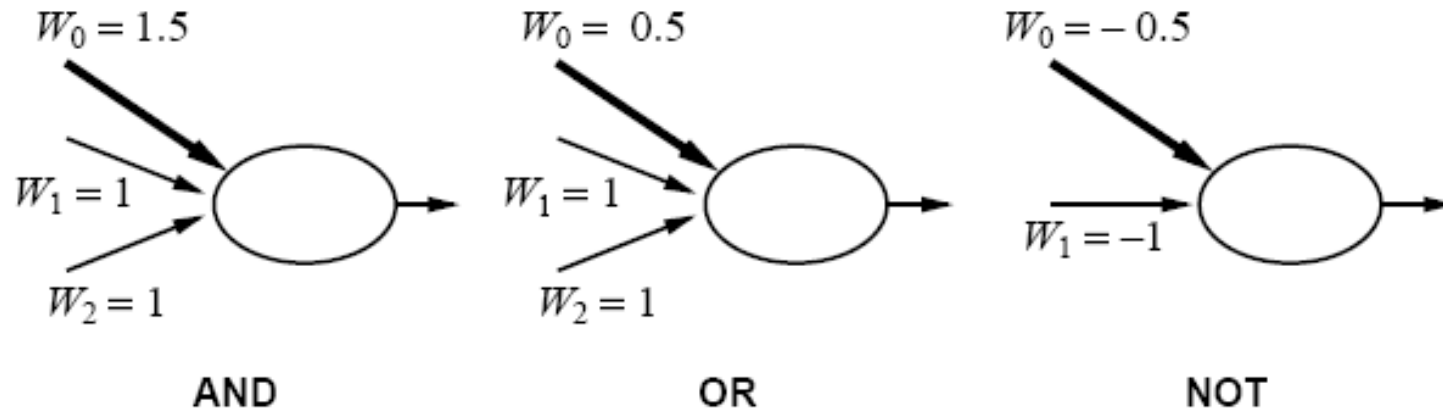
Função de limiar



(b)

Função sigmóide

Implementando funções lógicas

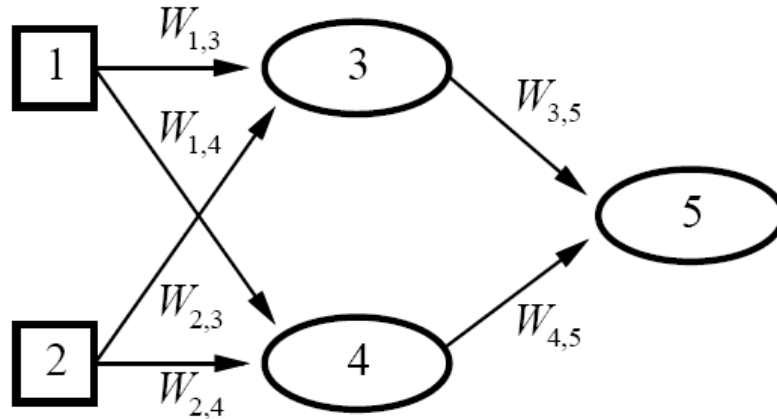


Podemos utilizar as unidades para construir uma rede que calcule qualquer função booleana das entradas.

Estruturas de Rede

- Existem duas categorias principais de redes neurais:
 - Redes acíclicas ou de alimentação direta.
 - ➔ • Representam uma função de sua entrada atual.
 - Não têm nenhum estado interno além dos pesos.
 - Redes cíclicas ou recorrentes.
 - Utilizam suas saídas para realimentar suas entradas.
 - Níveis de ativação da rede formam um sistema dinâmico.
 - Podem admitir memória de curto prazo, tornando-se mais interessantes como modelos do cérebro.

Exemplo: Rede de Alimentação Direta



Rede simples:

Duas entradas (1,2)

Duas unidades ocultas (3,4)

Uma saída (5)

- Uma rede de alimentação direta representa uma função não-linear de suas entradas:

$$\begin{aligned} a_5 &= g(W_{3,5} \cdot a_3 + W_{4,5} \cdot a_4) \\ &= g(W_{3,5} \cdot g(W_{1,3} \cdot a_1 + W_{2,3} \cdot a_2) + W_{4,5} \cdot g(W_{1,4} \cdot a_1 + W_{2,4} \cdot a_2)) \end{aligned}$$

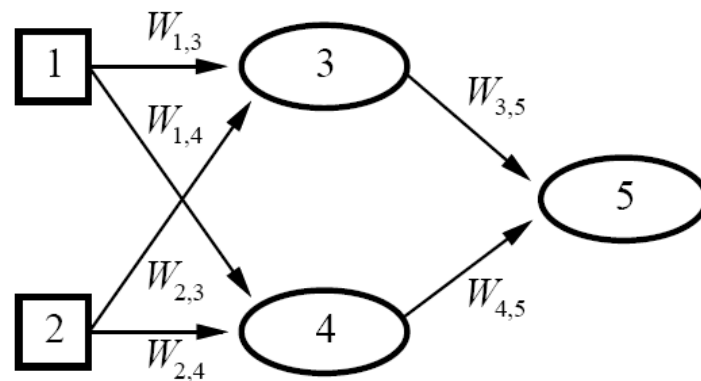
- O aprendizado ocorre através do ajuste dos pesos.

Unidade(s) de Saída

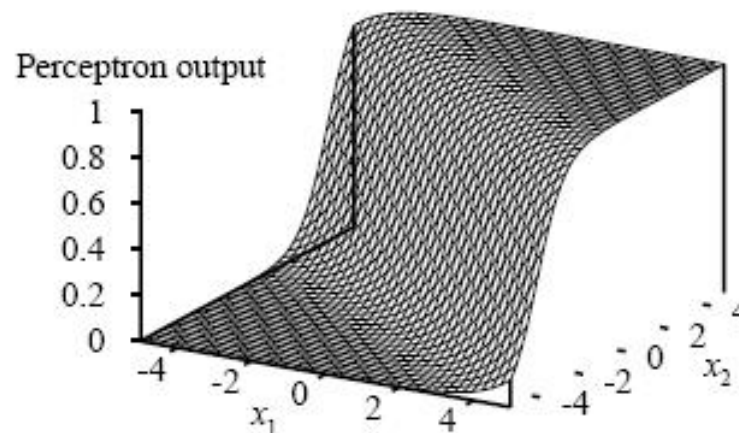
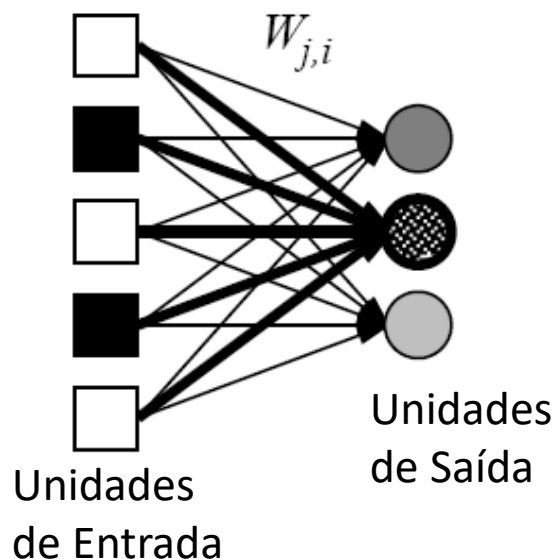
- Uma rede neural pode ser usada para classificação ou regressão.
- Para regressão:
 - Utiliza-se diretamente o valor na unidade de saída.
- Para classificação binária:
 - Um valor acima de 0,5 na unidade de saída é interpretado como uma classe e abaixo de 0,5 como a outra.
- Para classificação multi-classe (k classes):
 - Utiliza-se k unidades de saída, com o valor de cada uma representando a probabilidade relativa dessa classe, dada a entrada atual.

Camadas

- As redes de alimentação direta normalmente são organizadas em camadas.
- Cada unidade recebe apenas a entrada de unidades situadas na camada imediatamente precedente.
- A rede abaixo possui uma camada de entrada, uma camada oculta e uma camada de saída.



Redes de Alimentação Direta de Uma Camada (Perceptrons)



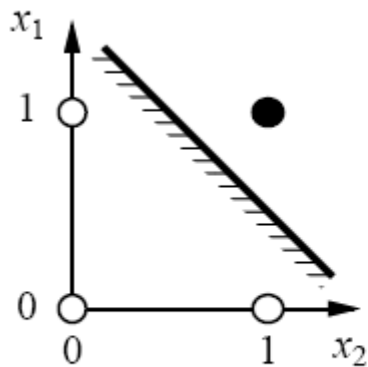
- As unidades de saída são independentes.
 - Cada peso afeta apenas uma das saídas.
 - Podemos estudar cada saída separadamente.
- Modificando os pesos podemos mudar a localização, orientação e a inclinação da “subida” na saída.

Espaço de Hipóteses de um Perceptron

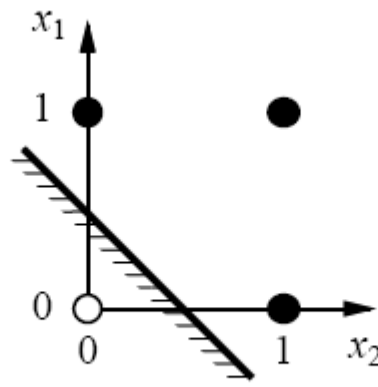
- Com uma função de ativação de limiar, a função de saída do perceptron é dada por:

$$\sum_j W_j x_j > 0 \quad \text{ou} \quad \mathbf{W} \cdot \mathbf{x} > 0$$

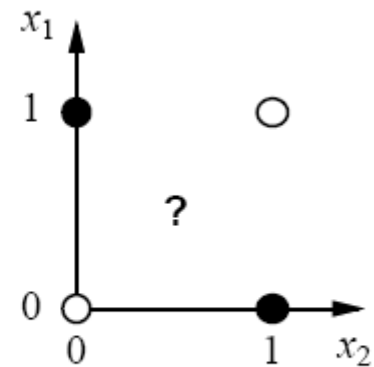
- Isso equivale a um separador linear:



(a) x_1 and x_2



(b) x_1 or x_2



(c) x_1 xor x_2

Crie sua NN em playground.tensorflow.org



Epoch
000,482

Learning rate

0.03

Activation

Tanh

Regularization

None

Regularization rate

0

Problem type

Classification

DATA

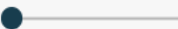
Which dataset do you want to use?



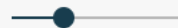
Ratio of training to test data: 50%



Noise: 0



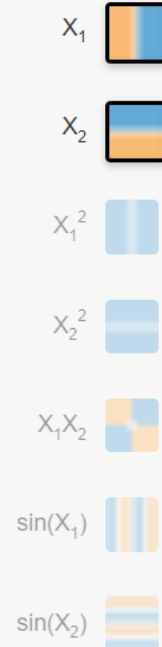
Batch size: 10



REGENERATE

FEATURES

Which properties do you want to feed in?



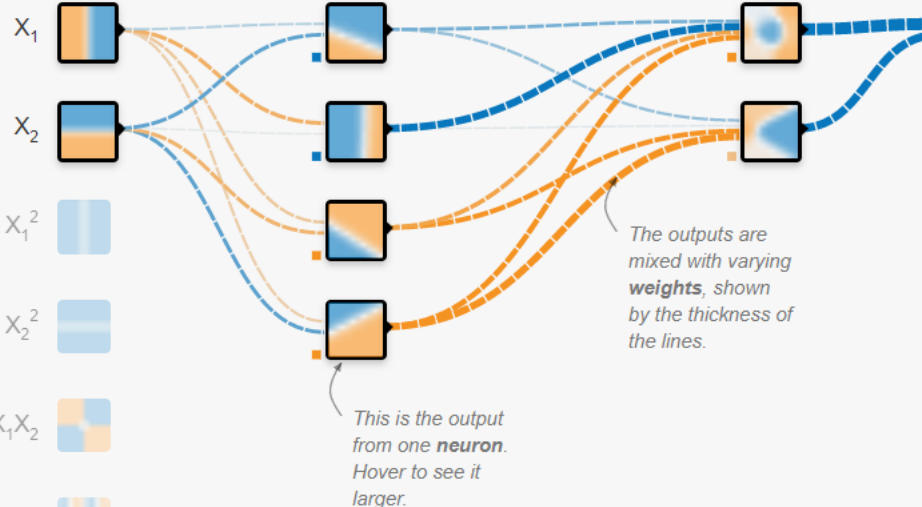
2 HIDDEN LAYERS

+ -

4 neurons

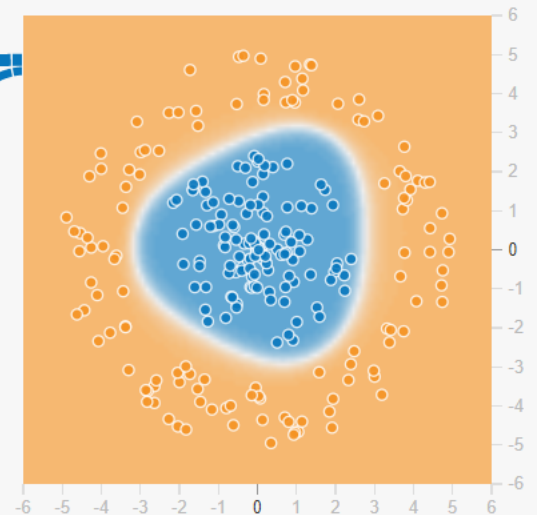
+ -

2 neurons

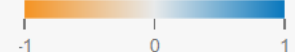


OUTPUT

Test loss 0.002
Training loss 0.001



Colors shows data, neuron and weight values.



☐ Show test data

☐ Discretize output

Aprendizado de Perceptrons

- O aprendizado é formulado como uma busca de otimização no espaço de pesos.
 - Ajusta-se os pesos da rede para minimizar alguma medida de erro no conjunto de treinamento.
- A medida “clássica” de erro é a **soma dos erros quadráticos**.
- O erro quadrático para um único exemplo com entrada $\mathbf{x}$ e saída verdadeira y é dado por:

$$E = \frac{1}{2}Err^2 \equiv \frac{1}{2}(y - h_{\mathbf{W}}(\mathbf{x}))^2$$

onde $h_{\mathbf{w}}(\mathbf{x})$ é a saída do perceptron.

Aprendizado de Perceptrons (cont.)

- Podemos usar o declínio de gradiente para reduzir o erro quadrático, calculando a derivada parcial de E em relação a cada peso:

$$\begin{aligned}\frac{\partial E}{\partial W_j} &= Err \times \frac{\partial Err}{\partial W_j} = Err \times \frac{\partial}{\partial W_j} (y - g(\sum_{j=0}^n W_j x_j)) \\ &= -Err \times g'(in) \times x_j\end{aligned}$$

- Logo podemos utilizar a seguinte regra de atualização de pesos:

$$W_j \leftarrow W_j + \alpha \times Err \times g'(in) \times x_j$$

onde α é a taxa de aprendizagem.

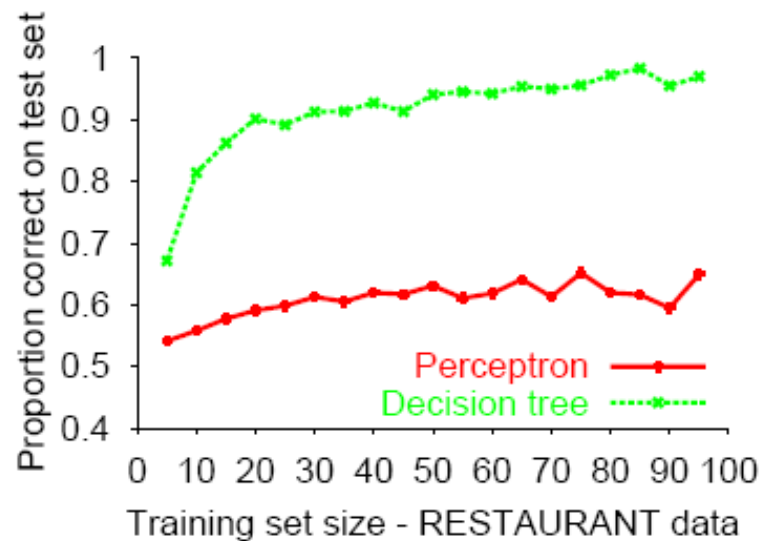
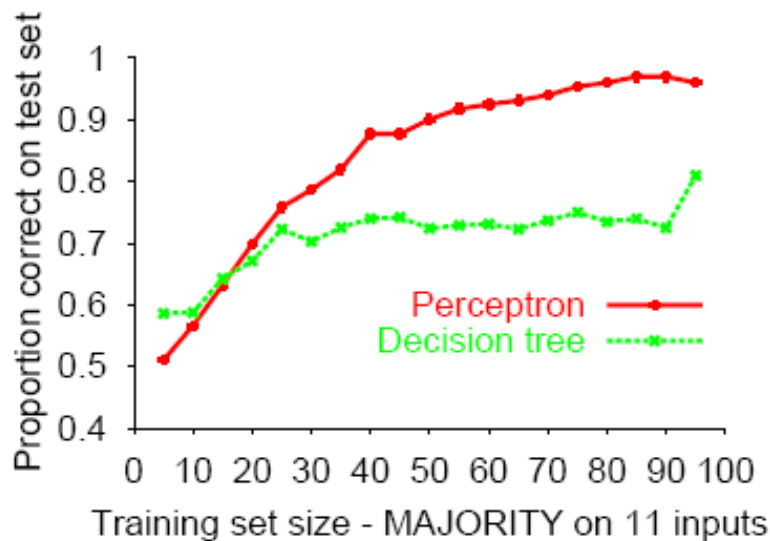
- Para a sigmóide, a derivada de g é $g(1-g)$.

Aprendizado de Perceptrons (cont.)

- O algoritmo executa os exemplos de treinamento através da rede, um de cada vez, ajustando os pesos depois de cada exemplo.
- Um ciclo através dos exemplos é chamado de **época**.
- As épocas são repetidas até ser alcançado algum critério de parada.
 - Em geral, o fato de as mudanças de pesos se tornarem muito pequenas.

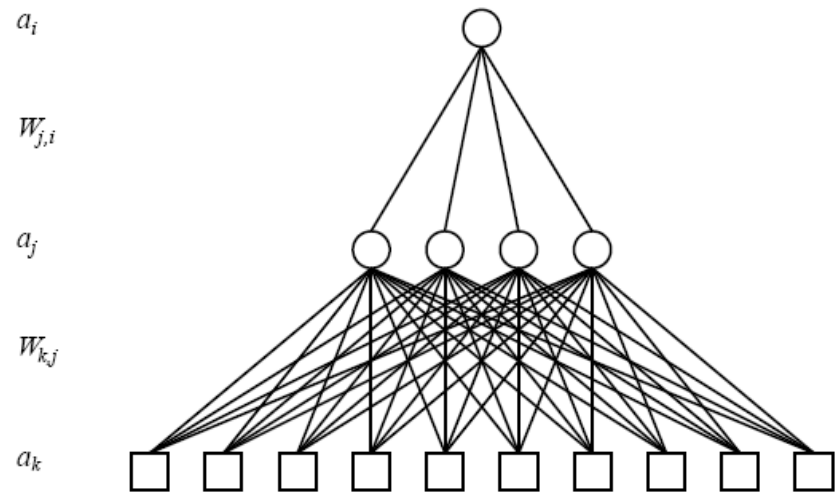
Curvas de Aprendizado

- O perceptron converge para uma função consistente se ela pode ser representada linearmente.

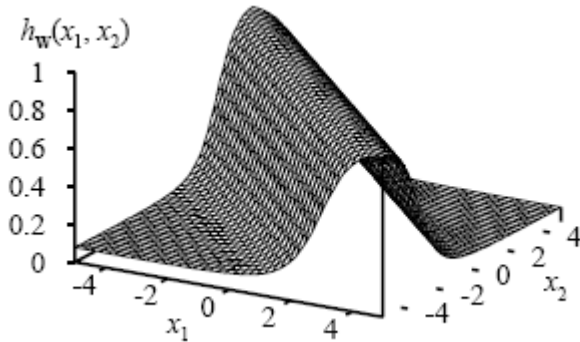


Redes Neurais de Alimentação Direta de Várias Camadas

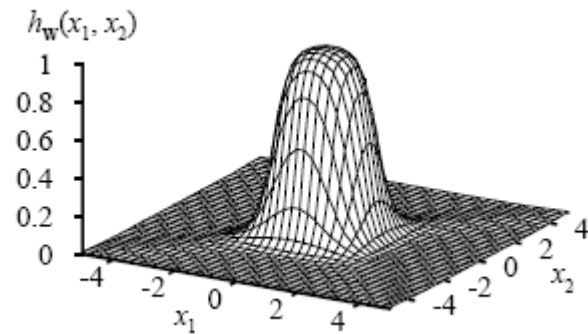
- O caso mais comum usa uma única camada oculta.
 - Com uma única camada podemos representar qualquer função contínua.
 - Com duas, até mesmo funções descontínuas podem ser representadas.
- O número de unidades ocultas tem que ser escolhido “manualmente”.
 - Normalmente utiliza-se validação cruzada.



Expressividade de Redes Neurais



Possível de
implementar com
duas unidades ocultas.



Possível de
implementar com
quatro unidades
ocultas.

Para representar qualquer função precisamos ir aumentando exponencialmente o número de unidades ocultas.

Aprendizado de redes neurais com uma camada oculta

- Camada de saída: regra de atualização idêntica a do perceptron

$$W_{j,i} \leftarrow W_{j,i} + \alpha \times a_j \times \Delta_i$$

onde $\Delta_i = Err_i \times g'(in_i)$

- Camada oculta: **propagação de retorno** do erro da camada de saída

$$\Delta_j = g'(in_j) \sum_i W_{j,i} \Delta_i$$

– A regra de atualização é dada por:

$$W_{k,j} \leftarrow W_{k,j} + \alpha \times a_k \times \Delta_j .$$

Derivação da regra de propagação de retorno

- O erro quadrático para um único exemplo é dado por (soma para todas as unidades de saída):

$$E = \frac{1}{2} \sum_i (y_i - a_i)^2$$

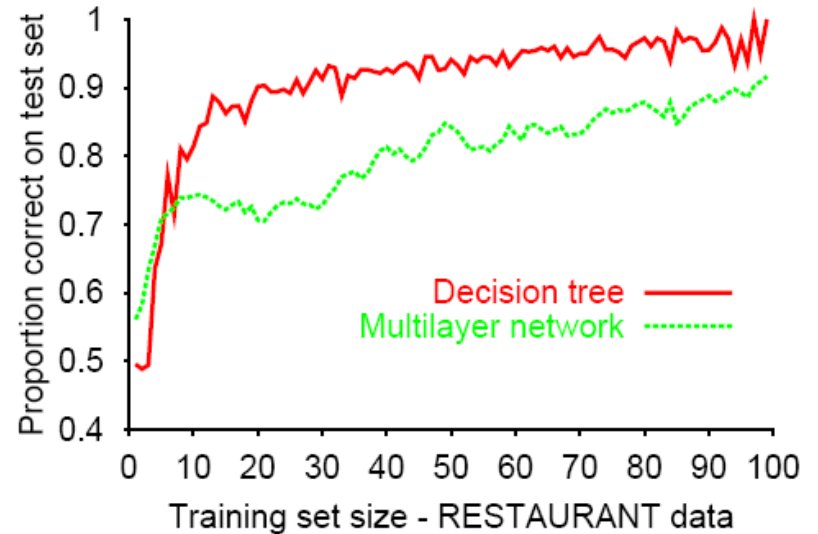
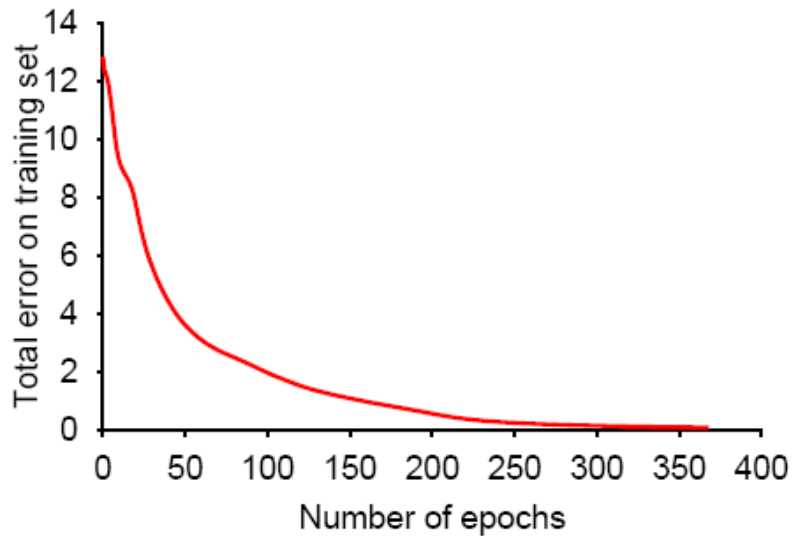
- A derivada parcial em relação a cada peso é dada por:

$$\begin{aligned} \frac{\partial E}{\partial W_{j,i}} &= -(y_i - a_i) \frac{\partial a_i}{\partial W_{j,i}} = -(y_i - a_i) \frac{\partial g(in_i)}{\partial W_{j,i}} \\ &= -(y_i - a_i) g'(in_i) \frac{\partial in_i}{\partial W_{j,i}} = -(y_i - a_i) g'(in_i) \frac{\partial}{\partial W_{j,i}} \left(\sum_j W_{j,i} a_j \right) \\ &= -(y_i - a_i) g'(in_i) a_j = -a_j \Delta_i \end{aligned}$$

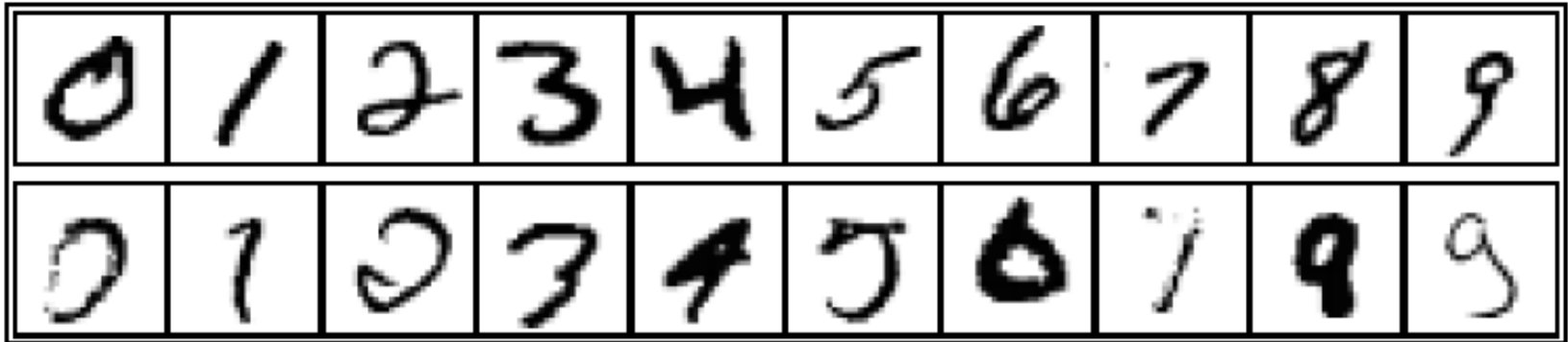
Derivação da regra de propagação de retorno (cont.)

$$\begin{aligned}\frac{\partial E}{\partial W_{k,j}} &= -\sum_i (y_i - a_i) \frac{\partial a_i}{\partial W_{k,j}} = -\sum_i (y_i - a_i) \frac{\partial g(in_i)}{\partial W_{k,j}} \\&= -\sum_i (y_i - a_i) g'(in_i) \frac{\partial in_i}{\partial W_{k,j}} = -\sum_i \Delta_i \frac{\partial}{\partial W_{k,j}} \left(\sum_j W_{j,i} a_j \right) \\&= -\sum_i \Delta_i W_{j,i} \frac{\partial a_j}{\partial W_{k,j}} = -\sum_i \Delta_i W_{j,i} \frac{\partial g(in_j)}{\partial W_{k,j}} \\&= -\sum_i \Delta_i W_{j,i} g'(in_j) \frac{\partial in_j}{\partial W_{k,j}} \\&= -\sum_i \Delta_i W_{j,i} g'(in_j) \frac{\partial}{\partial W_{k,j}} \left(\sum_k W_{k,j} a_k \right) \\&= -\sum_i \Delta_i W_{j,i} g'(in_j) a_k = -a_k \Delta_j\end{aligned}$$

Exemplo: Restaurante



Outro exemplo: Reconhecimento de Dígitos



3-nearest-neighbor = 2.4% error

400–300–10 unit MLP = 1.6% error

LeNet: 768–192–30–10 unit MLP = 0.9% error

Current best (kernel machines, vision algorithms) $\approx$ 0.6% error

XOR problem

- Analise (execute, adapte e teste) os 3 exemplos de Redes Neurais programadas em Python para resolver o problema XOR (Ou exclusivo).
- Os códigos estão disponíveis na página da disciplina.