Inteligência Artificial

Redes Neurais Artificiais

Prof. Fabio Augusto Faria

1º semestre 2021

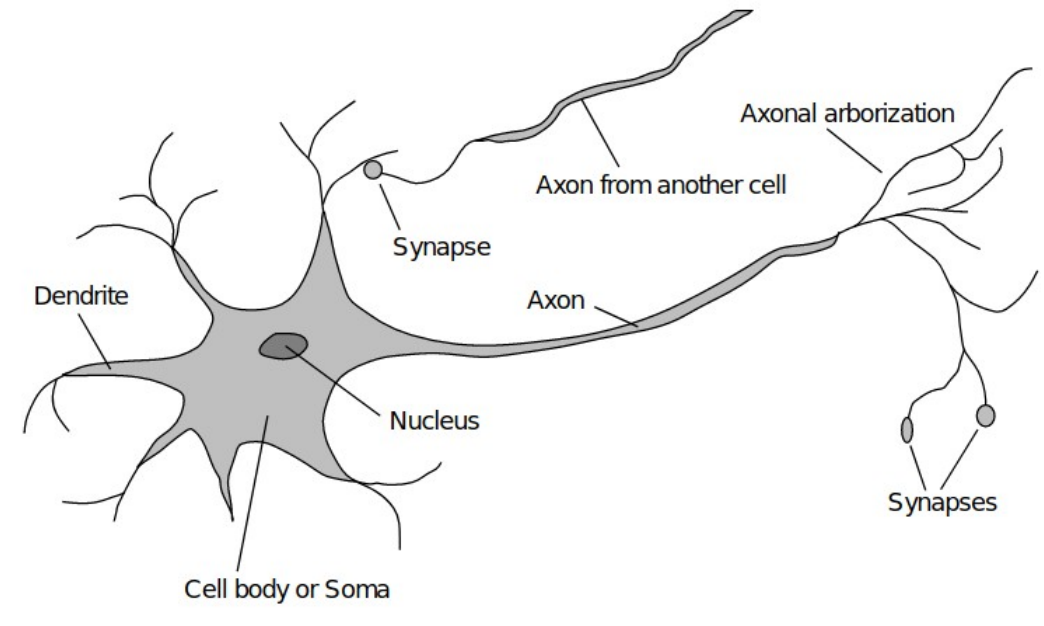
Tópicos

- Redes Neurais Artificiais (RNA)
- Tipos de Redes Neurais Artificiais
- Primeira RNA (Perceptron)
- Redes Neurais Multi-camadas (Multiple Layer Perceptron – MLP)

Histórico

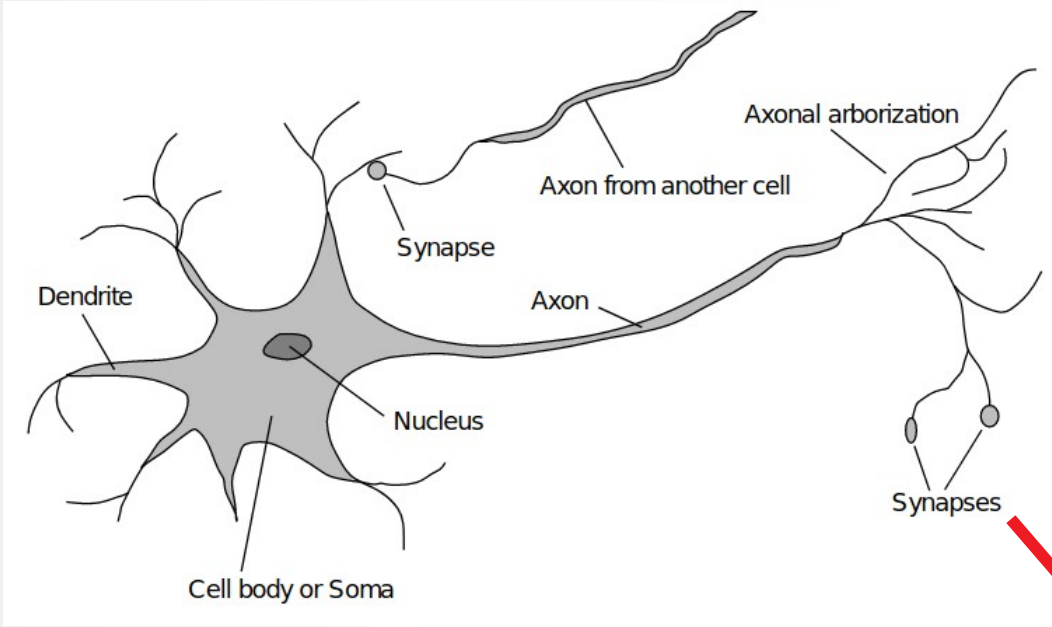
- Modelo computacional criado por Warren McCulloch and Walter Pitts (1943);
- Rosenblatt criou a perceptron (1958);
- Minsky and Papert descobriram as limitações das perceptrons (1969);
- Werbos realizou um treinamento prático das MLP com retro-propagação de erros (1975);
- Rumelhart, Hinton, and Williams mostrou que era possível predizer uma próxima palavra de uma sequencia de palavras utilizando da representação interna da retro-propagação aprendida (1986);
- …
- …
- …
- Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton criam a AlexNet (2012).

Neurônio

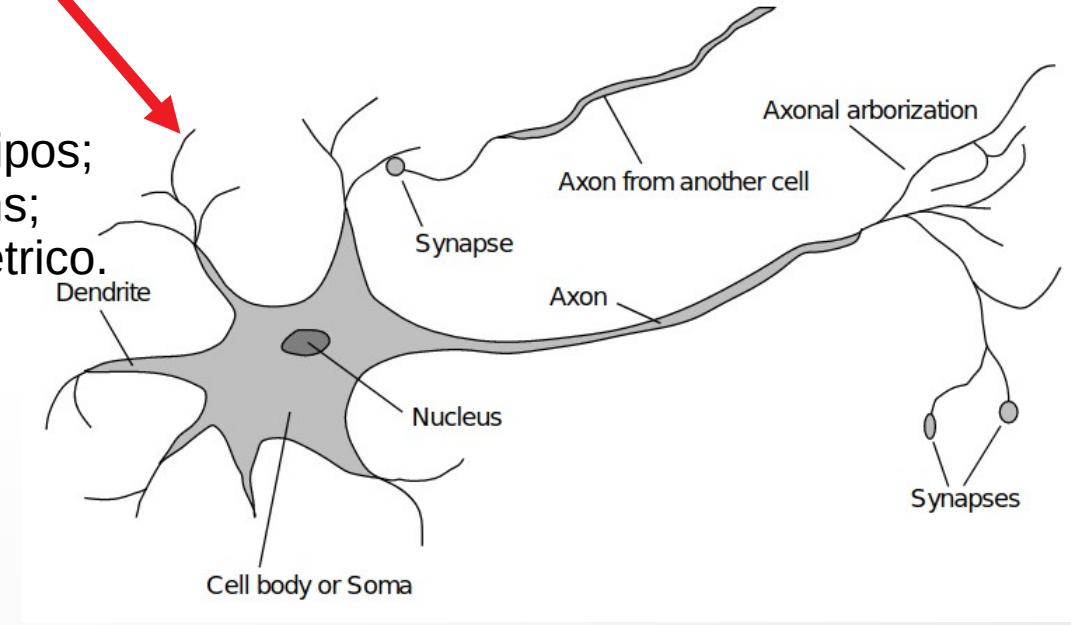


- Cérebro tem 10^11 neurônios de mais de 20 tipos;
- 10^14 sinapses com tempo de ciclo de 1–10ms;
- Sinal é um ruído “spike trains” de potencial elétrico.

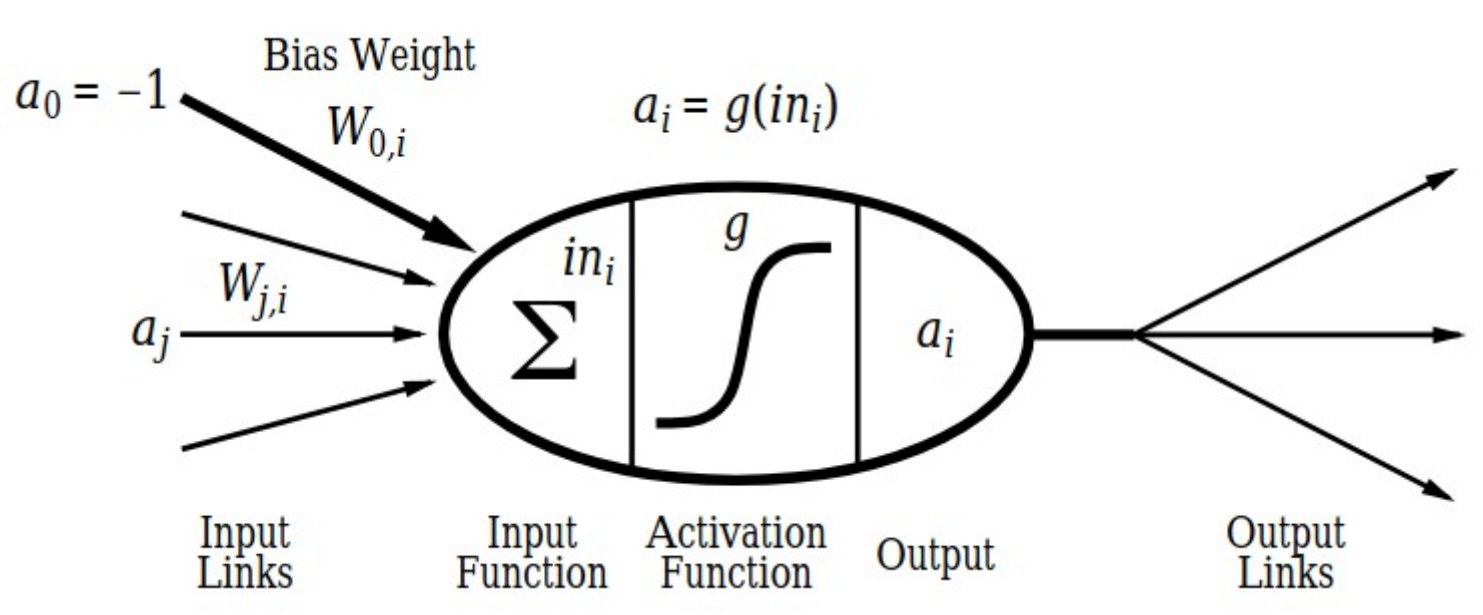
Neurônio



- Cérebro tem 10^11 neurônios de mais de 20 tipos;
- 10^14 sinapses com tempo de ciclo de 1–10ms;
- Sinal é um ruído “spike trains” de potencial elétrico.



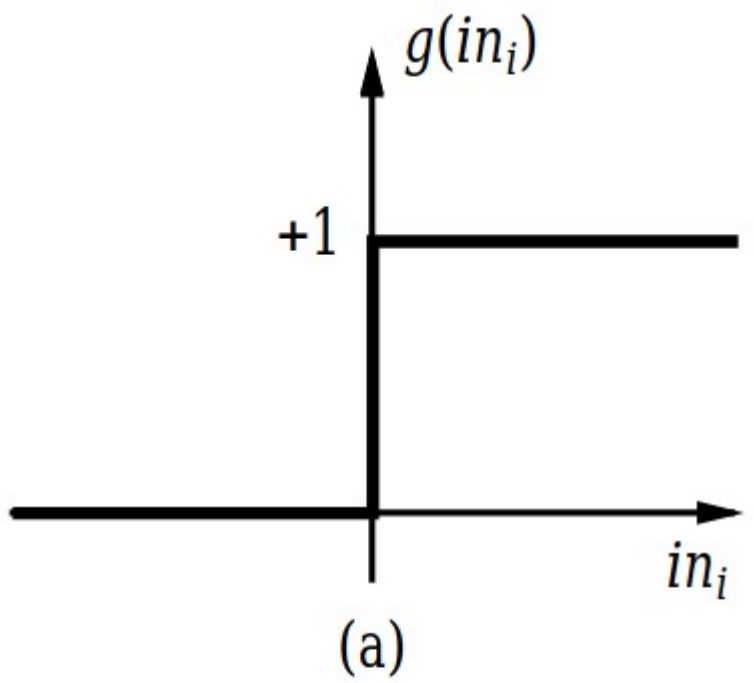
Neurônio Artificial (perceptron)



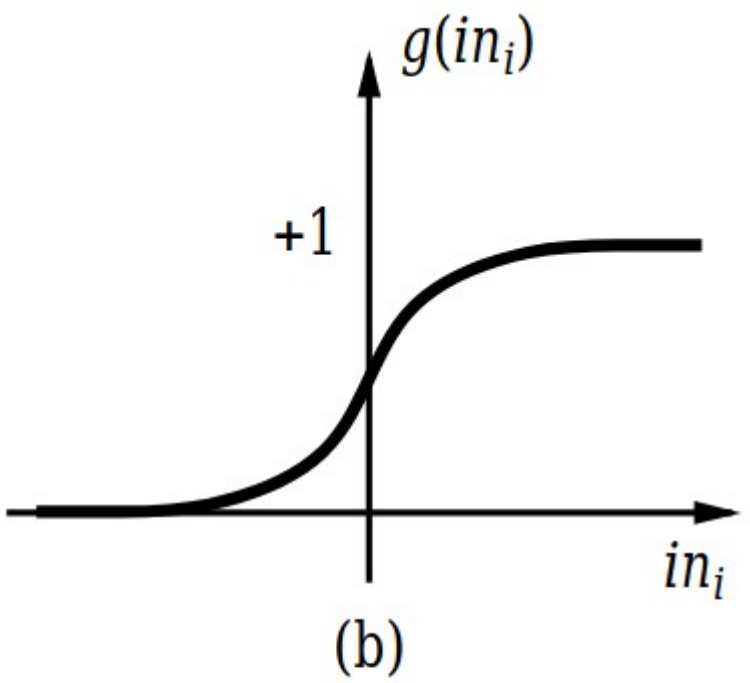
$$in_j = \sum_{i=0}^{n} w_{i,j} a_i \ .$$

$$a_j = g(in_j) = g\left(\sum_{i=0}^{n} w_{i,j} a_i\right) \ .$$

Função de Ativação (g)



(a)



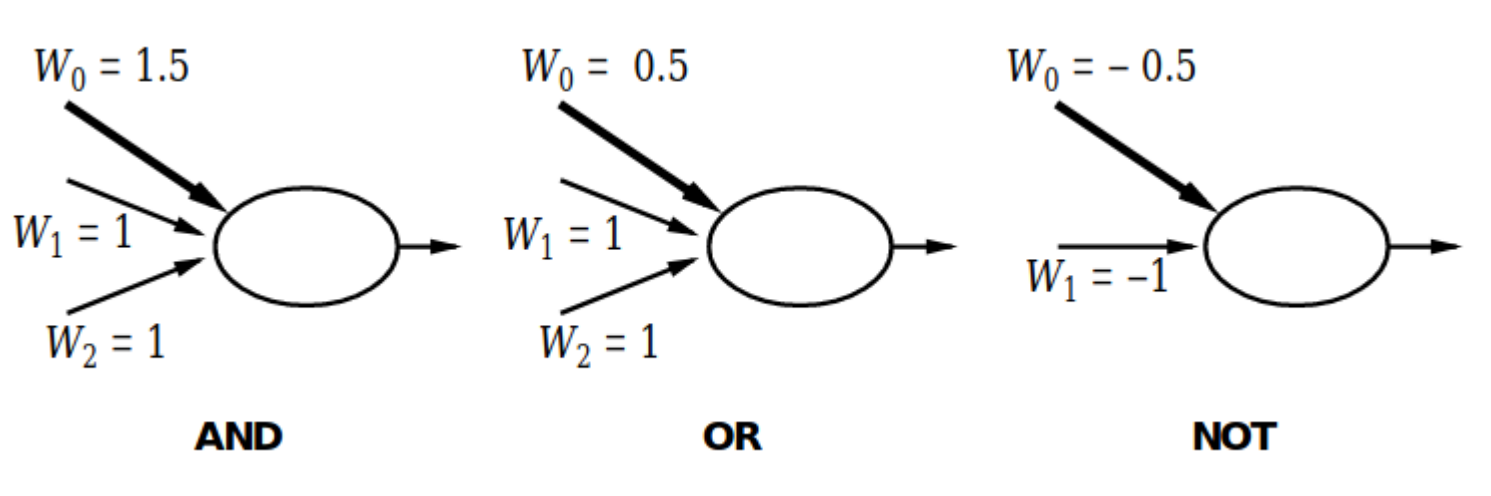
(b)

(a) is a step function or threshold function

(b) is a sigmoid function $1/(1+e^{-x})$

Changing the bias weight $W_{0,i}$ moves the threshold location

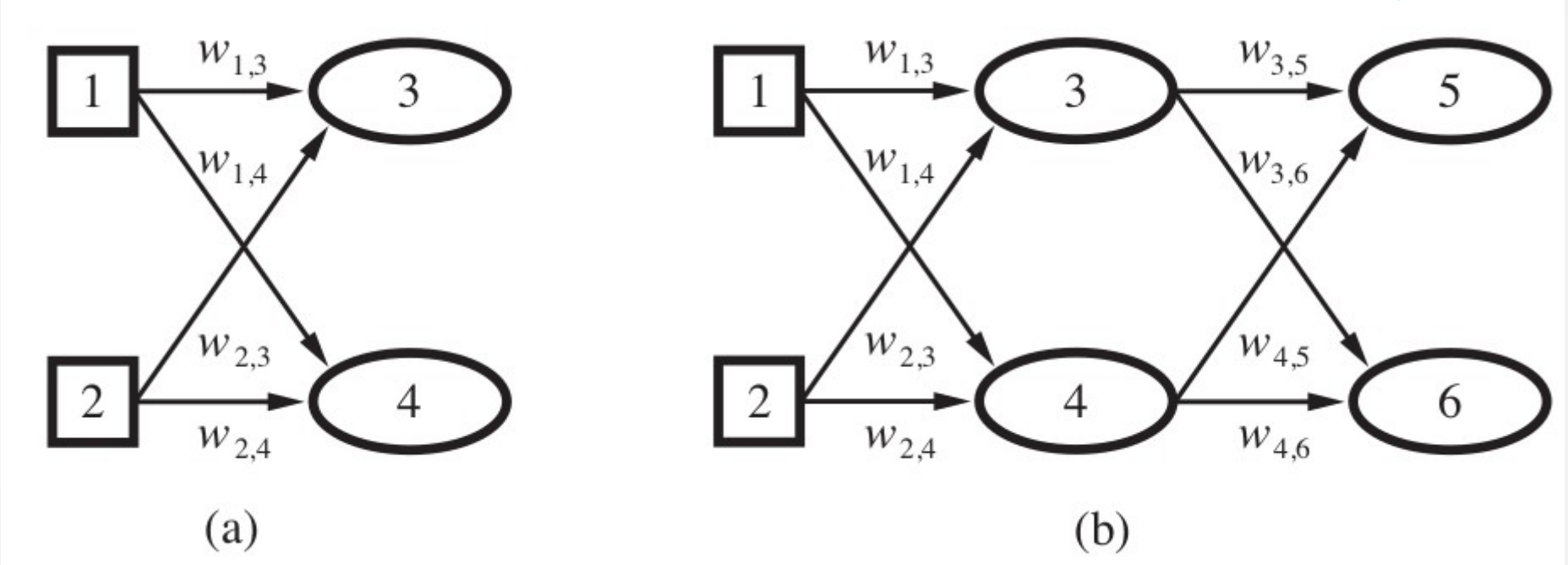
Neurônio (perceptron)



Toda função booleana pode ser implementada com perceptron.

Tipos de Redes Neurais

Redes com alimentação para frente (feed-forward)

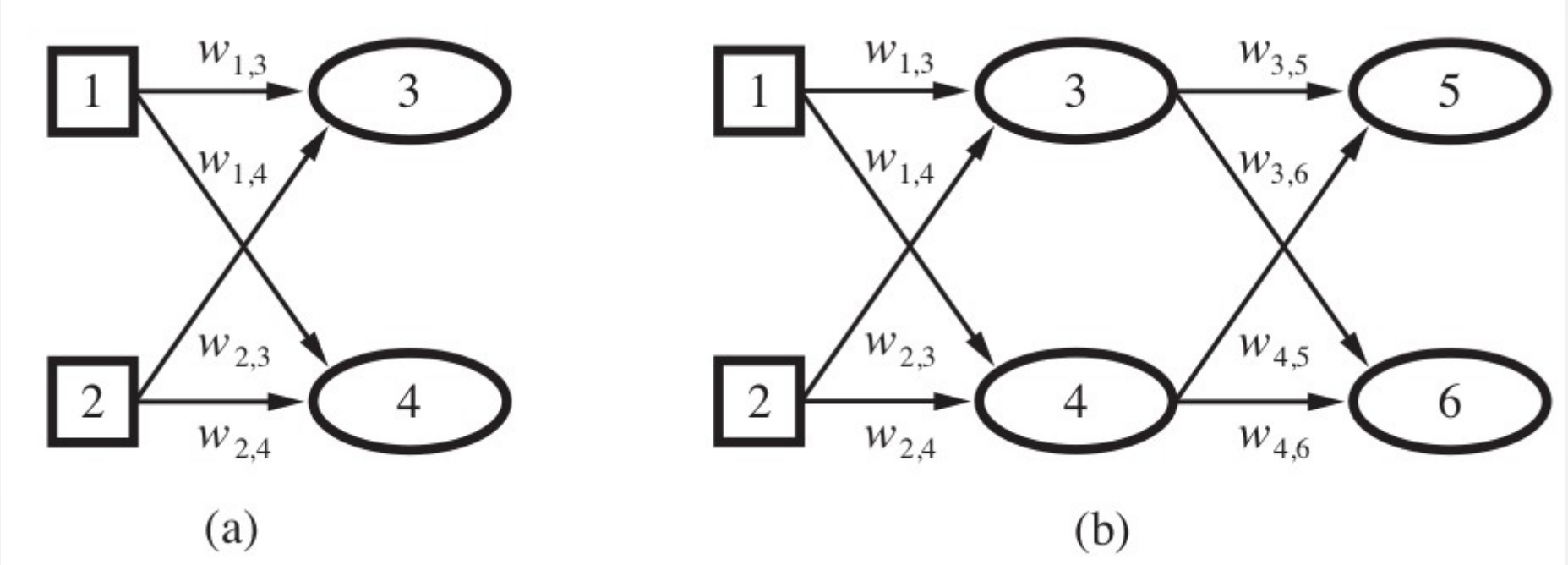


Única Camada

Múltiplas Camadas

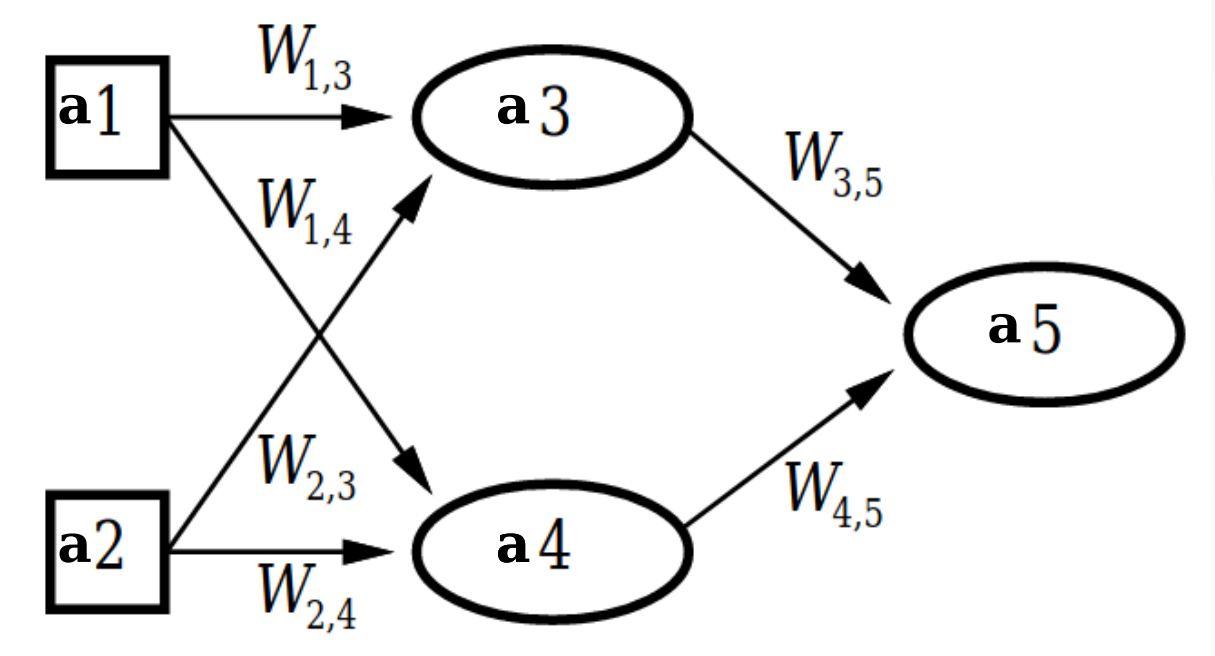
Tipos de Redes Neurais

Redes com alimentação para frente (feed-forward)



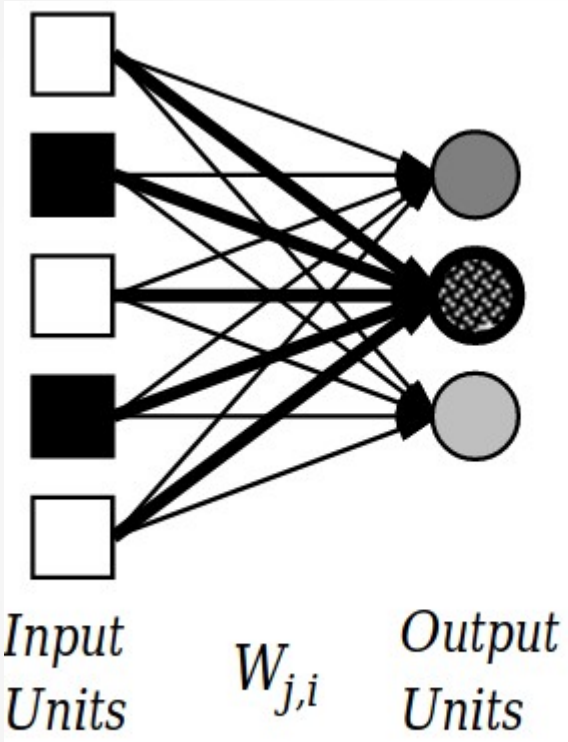
Retropropagação do erro (back-propagation)

Estrutura de Rede

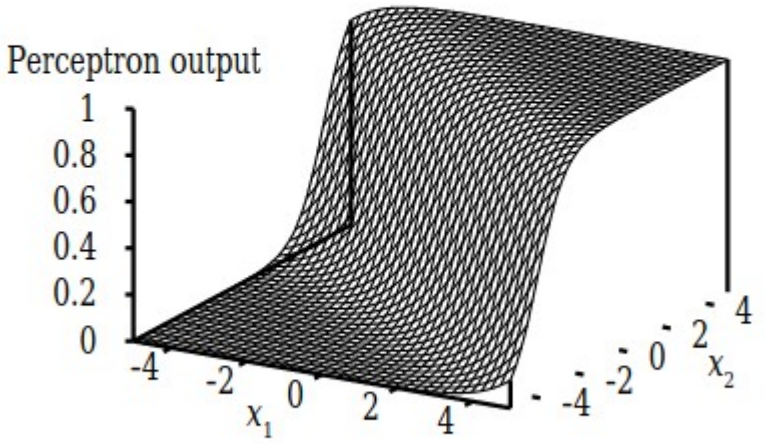


$$
\begin{aligned}
a_5 &= g(W_{3,5} \cdot a_3 + W_{4,5} \cdot a_4) \\
&= g(W_{3,5} \cdot g(W_{1,3} \cdot a_1 + W_{2,3} \cdot a_2) + W_{4,5} \cdot g(W_{1,4} \cdot a_1 + W_{2,4} \cdot a_2))
\end{aligned}
$$

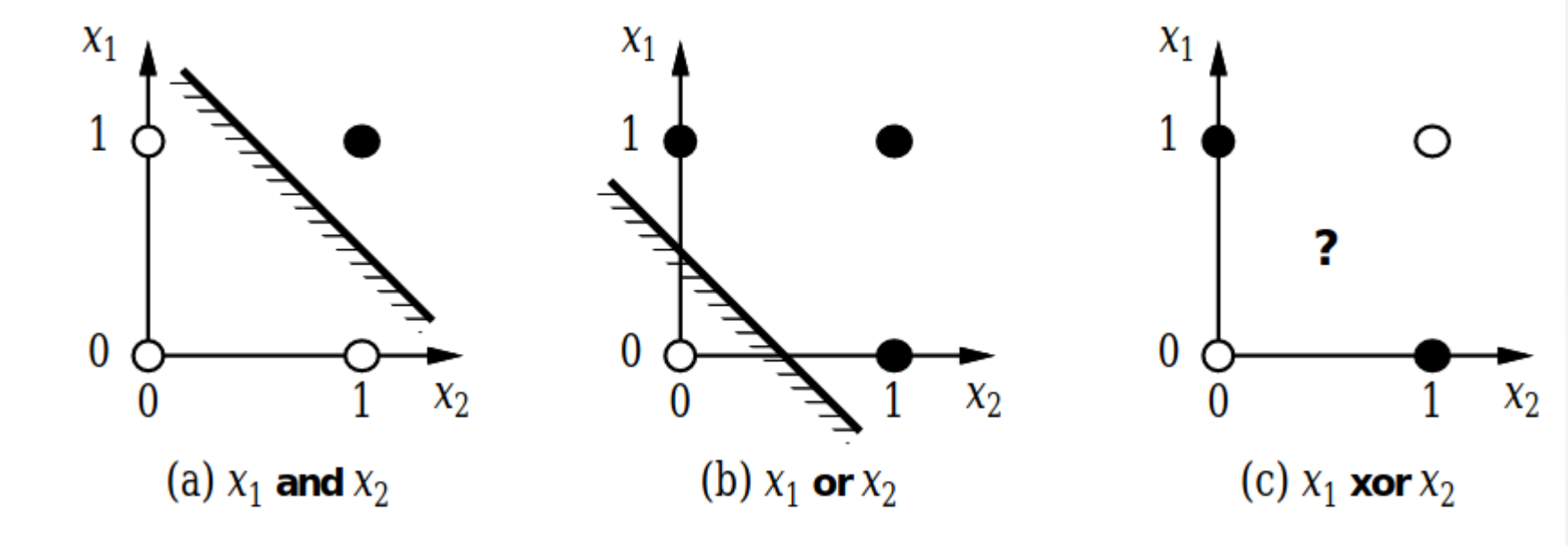
Rede Única Camada (Perceptron)



Função Sigmóide



Rede Única Camada (Perceptron)



- Problema **linearmente separável**?
- E os **NÃO** linearmente separável?

Aprendizagem em Perceptron

Ajuste dos pesos no **conjunto de treinamento**

Erro Quadrado para um exemplo de entrada x e saída y é:

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

Busca por otimização dos pesos por gradiente decendente:

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

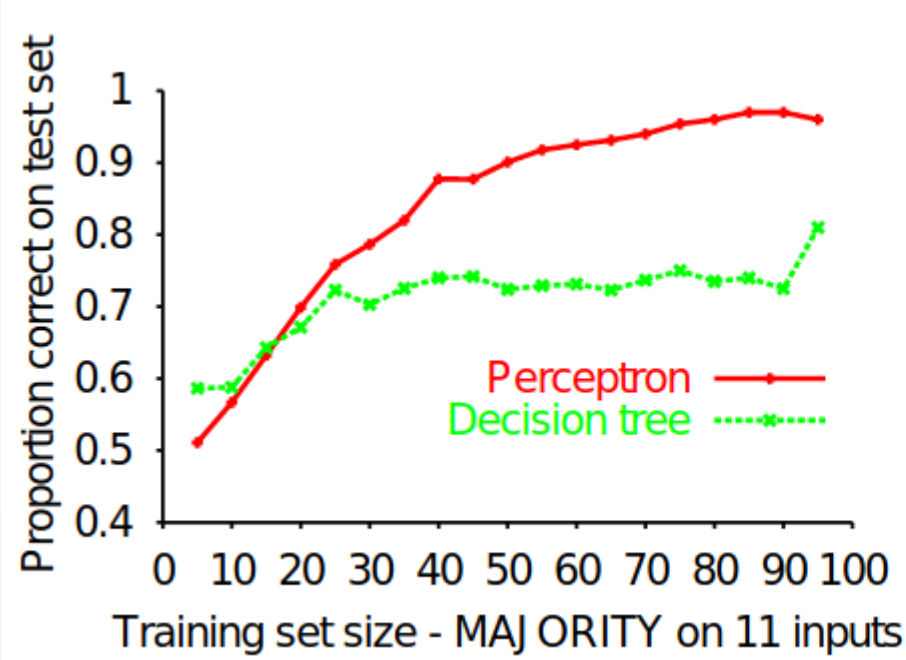
Regra de atualização de peso:

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

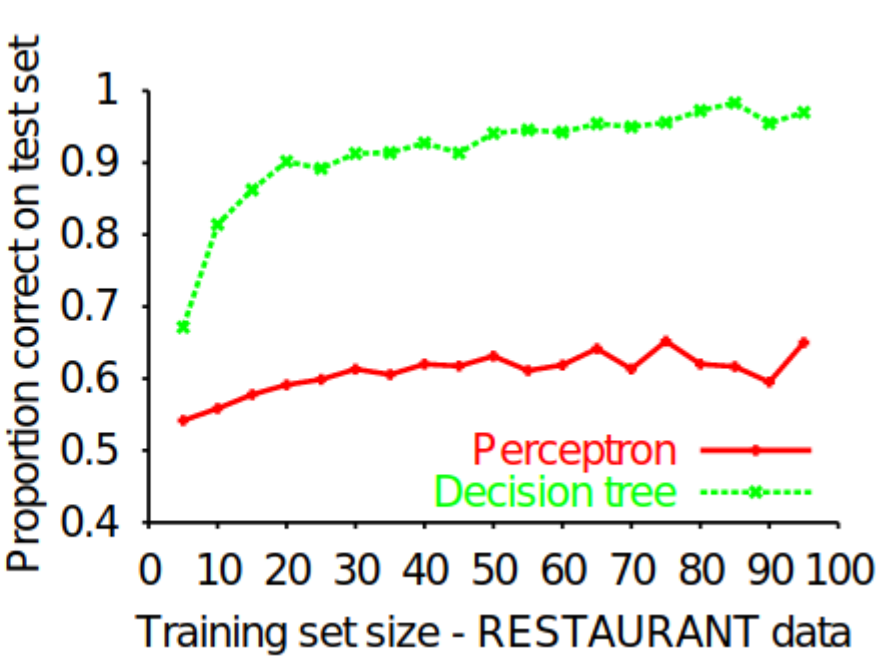
Erro positivo → aumenta saída da rede
Assim, **<u>aumenta</u>** pesos para entradas positivas **<u>ou</u>**
<u>diminui</u> para entradas negativas

Rede Única Camada (Perceptron)

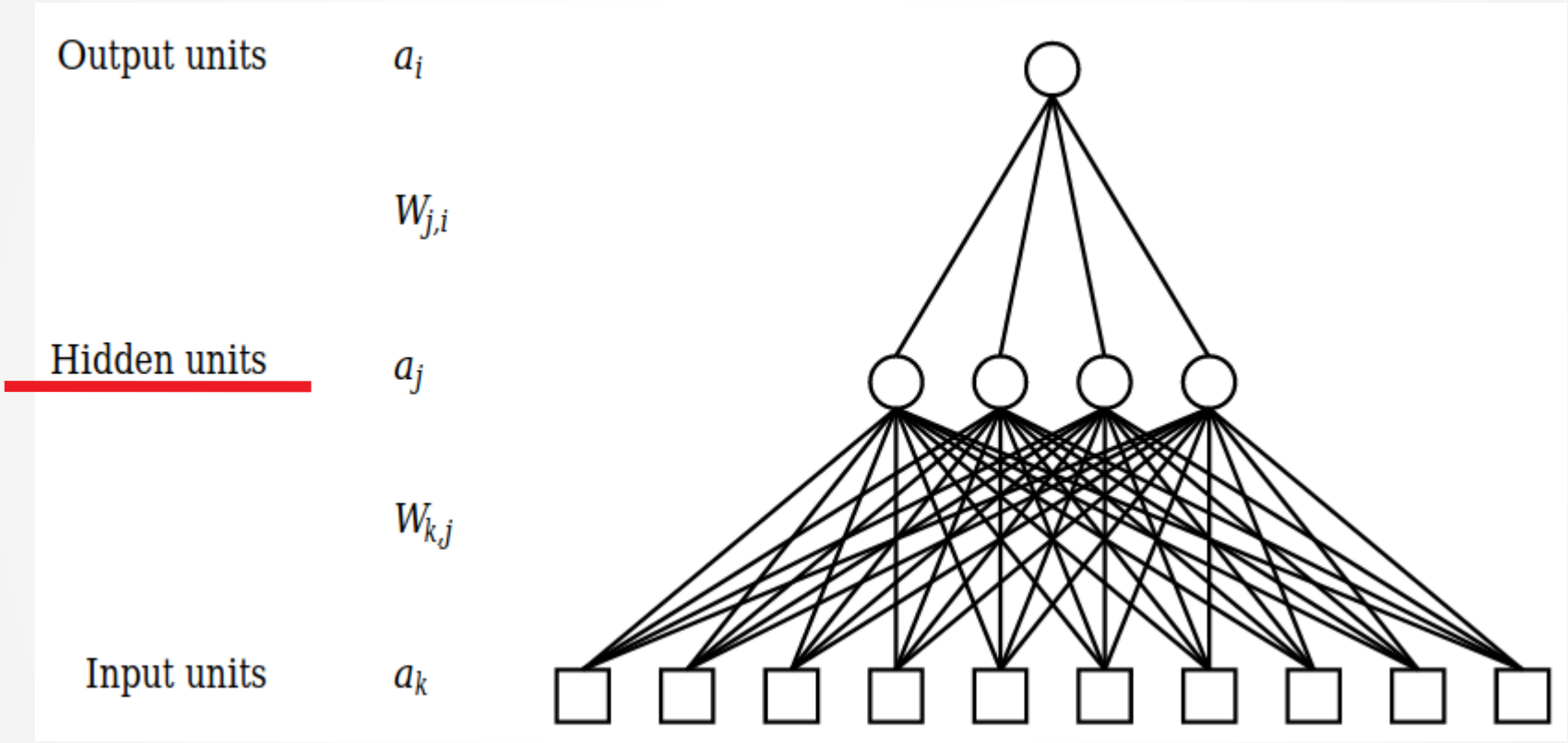
linearmente separável



NÃO linearmente separável

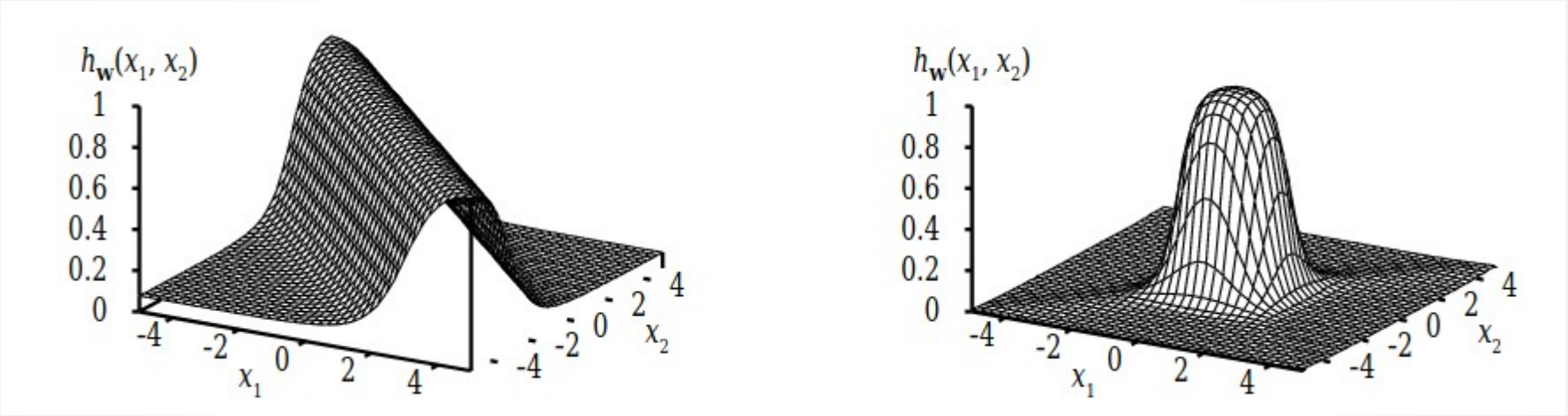


Múltiplas Camadas Perceptron (MLP)



- Adição de camadas **ocultas**
- Capacidade de modelar qualquer função matemática
- Aumento do número de parâmetro → **overfitting**

Múltiplas Camadas Perceptron (MLP)



Funções contínuas com 2 camadas ocultas e 3 camadas ocultas.

Retropropagação de Erros

Camada saída: mesmo para perceptron (única camada)

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

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

Camada oculta: retro-propaga o erro da camada de saída:

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

Regra de **atualização dos pesos** da camada oculta:

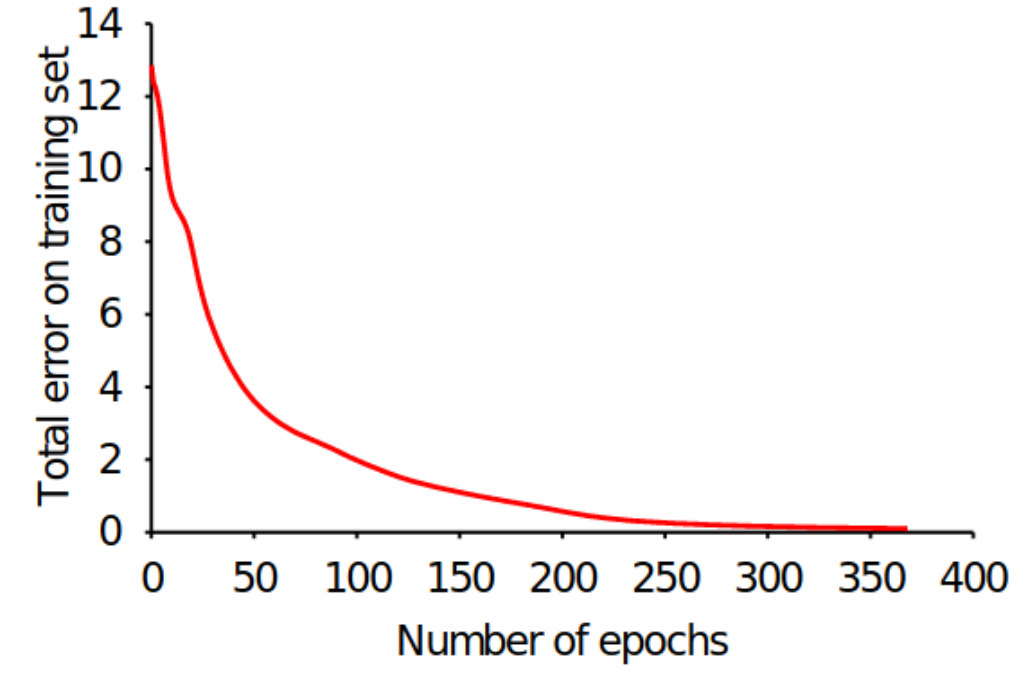
$$W_{k,j} \leftarrow W_{k,j} + \alpha \times a_k \times \Delta_j \ .$$

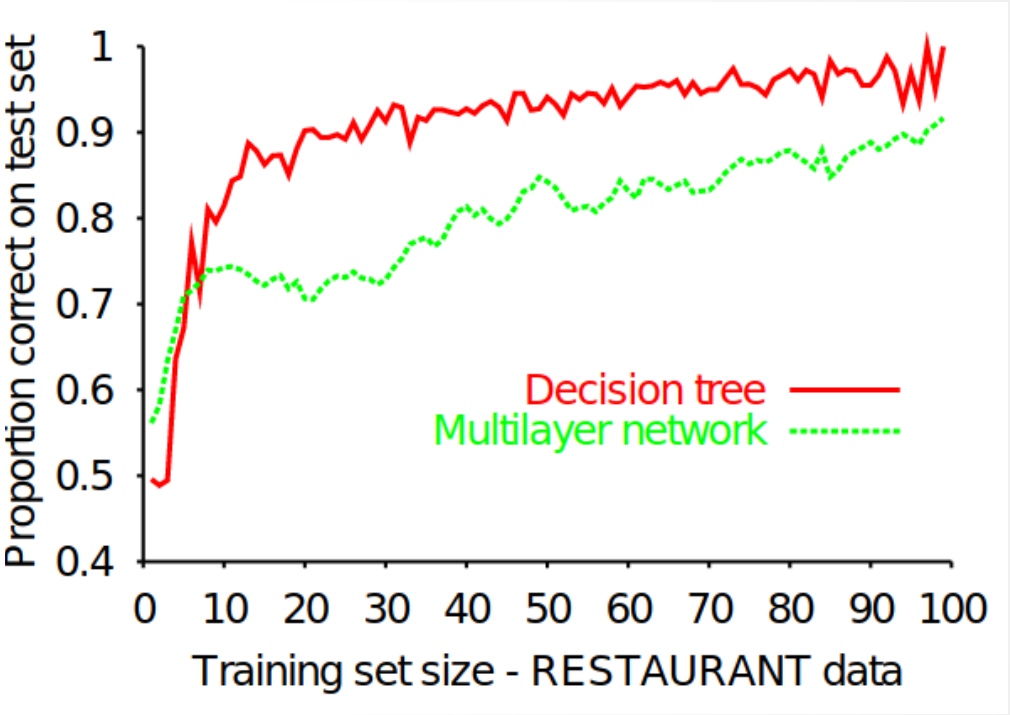
Retropropagação de Erros

function BACK-PROP-LEARNING($examples$, $network$) **returns** a neural network
  inputs: $examples$, a set of examples, each with input vector **x** and output vector **y**
    $network$, a multilayer network with L layers, weights $w_{i,j}$, activation function g
  local variables: Δ, a vector of errors, indexed by network node

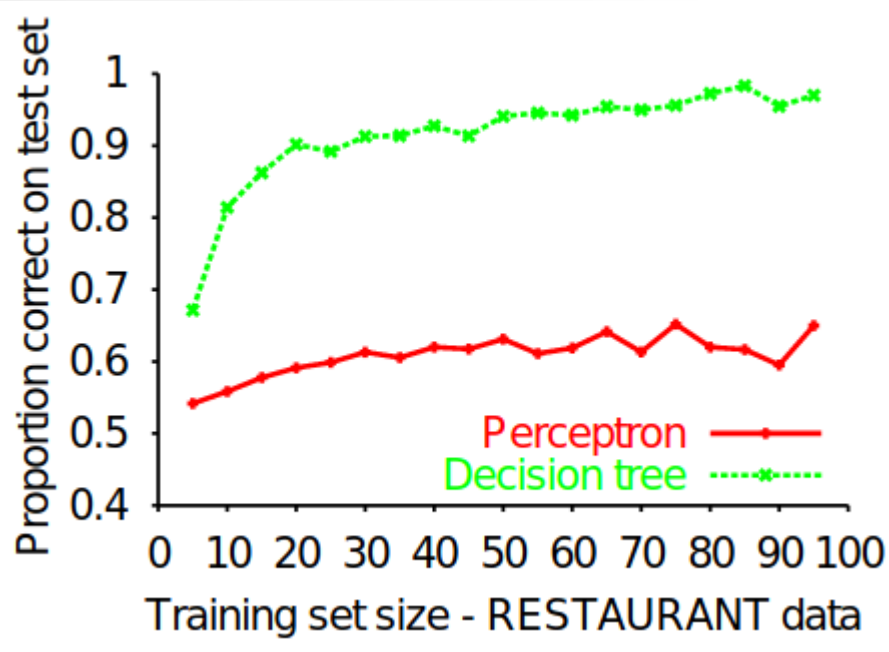
  repeat
    for each weight $w_{i,j}$ in $network$ **do**
      $w_{i,j} \leftarrow$ a small random number
    for each example $(\mathbf{x}, \mathbf{y})$ **in** $examples$ **do**
      /* *Propagate the inputs forward to compute the outputs* */
      for each node i in the input layer **do**
        $a_i \leftarrow x_i$
      for $\ell = 2$ **to** L **do**
        for each node j in layer ℓ **do**
          $in_j \leftarrow \sum_i w_{i,j} \; a_i$
          $a_j \leftarrow g(in_j)$
      /* *Propagate deltas backward from output layer to input layer* */
      for each node j in the output layer **do**
        $\Delta[j] \leftarrow g'(in_j) \times (y_j - a_j)$
      for $\ell = L - 1$ **to** 1 **do**
        for each node i in layer ℓ **do**
          $\Delta[i] \leftarrow g'(in_i) \sum_j w_{i,j} \; \Delta[j]$
      /* *Update every weight in network using deltas* */
      for each weight $w_{i,j}$ in $network$ **do**
        $w_{i,j} \leftarrow w_{i,j} + \alpha \times a_i \times \Delta[j]$
  until some stopping criterion is satisfied
  return $network$

Multiplas Camadas Perceptron (MLP)

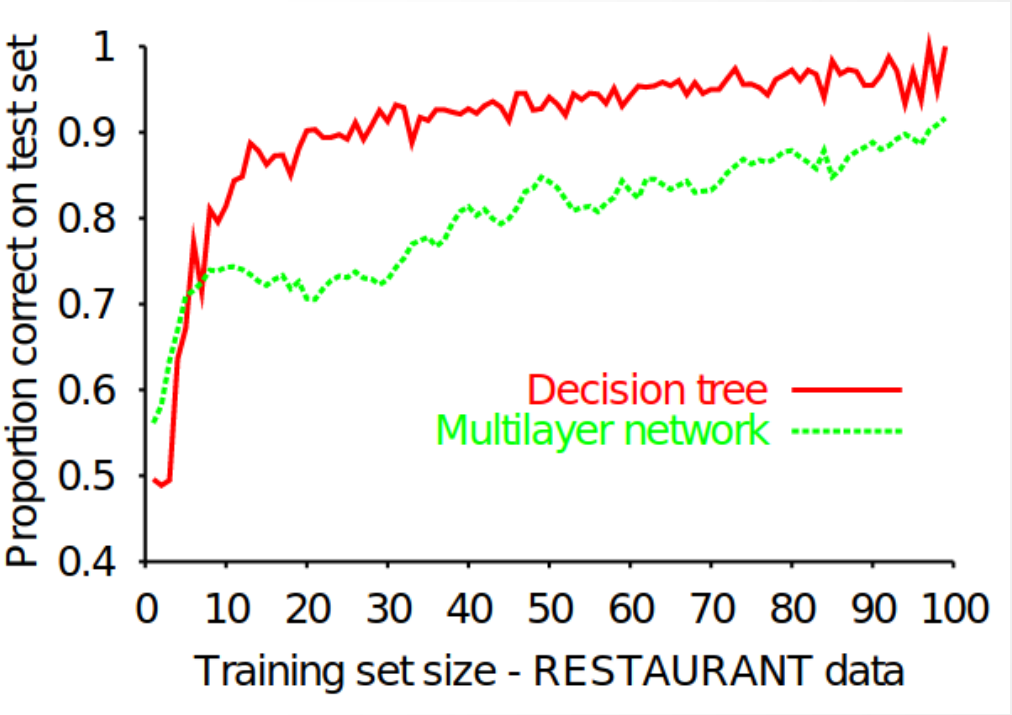




Multiplas Camadas Perceptron (MLP)



ANTES



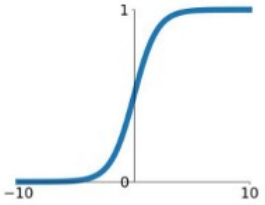
DEPOIS

Outras Funções de Ativação

Activation Functions

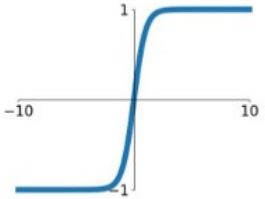
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



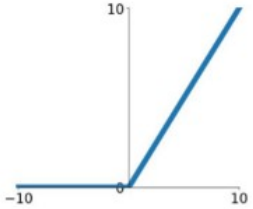
tanh

$$\tanh(x)$$



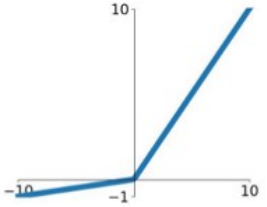
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

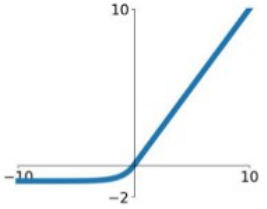


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Obs: Vide notebook no drive.

Evitar Superajustes (*Overfitting*)

- **Regularização Ativa** *(Activity Regularization)*: penalizar o modelo durante treinamento baseado na magnetude das ativações;
- **Restrição de Pesos** *(Weight Constraint): restringir a magnetude dos pesos em um intervalo pré-determinado;*
- **Poda** *(dropout)*: remover nós ou arestas aleatoriamente. Esta ideia também utilizada nas árvores de decisão, mas não igual;
- **Ruído** *(noise)*: adicionar ruído estatísticopara as entradas durante treinamento;
- **Parada precose** *(early stopping)*: monitorar o desempenho da rede no conjunto de validação e parar o treinamento quando iniciar a degradação.

Também funcionam para Aprendizado Profundo (Deep Learning)!

Trabalho dia 23/05/2021

1) Utilizar dois banco de dados de vacinação do estado de SP disponíveis no drive da disciplina (geral_vacinaja_classificacao.csv e geral_vacinaja_regressao.csv);

2) Escalar por coluna todos os atributos existentes nos bancos de dados. O intervalo de valores deve ficar entre [0,1] e utilize a média/desvio para isso;

3) Adotar protocolo de validação k-fold crossvalidation com **k=5**;

4) Rodar duas tarefas de mineração de dados (classificação e regressão):

    1) Na Classificação: criar dois modelos (Classificador Linear Rígido e MLP) que **classificam quais cidades de SP estão na classe "BOA" ou "RUIM" no processo de vacinação**? A porcentagem de acerto será a medida de avaliação utilizada.

    2) Na Regressão: criar dois modelos (Regressão Linear e MLP) que dado população, área, dose 1 e doses recebidas, **estimam qual é o valor estimado da dose 2**? O erro quadrático mínimo será a medida de avaliação utilizada.

5) Criar um gráfico para cada tarefa, mostrando o desempenho de cada modelo em cada uma das tarefas e responder qual o melhor modelo?

6) Caso tenham interesse de saber o nome das cidades de SP, basta abrir o arquivo "geral_vacinaja.csv".

DIVIRTA-SE!!!!

Referências

- Peter Norvig e Stuart Russel. **Inteligência Artificial**. Cap. 18. Seção 8.
- **Machine Learning Mastery**
  https://machinelearningmastery.com/introduction-to-regularization-to-reduce-overfitting-and-improve-generalization-error/
- **Scikit-learn:**
  https://scikit-learn.org/stable/modules/neural_networks_supervised.html#neural-networks-supervised