ACH2016 - Inteligência Artificial
Aula 06 - Redes Neurais

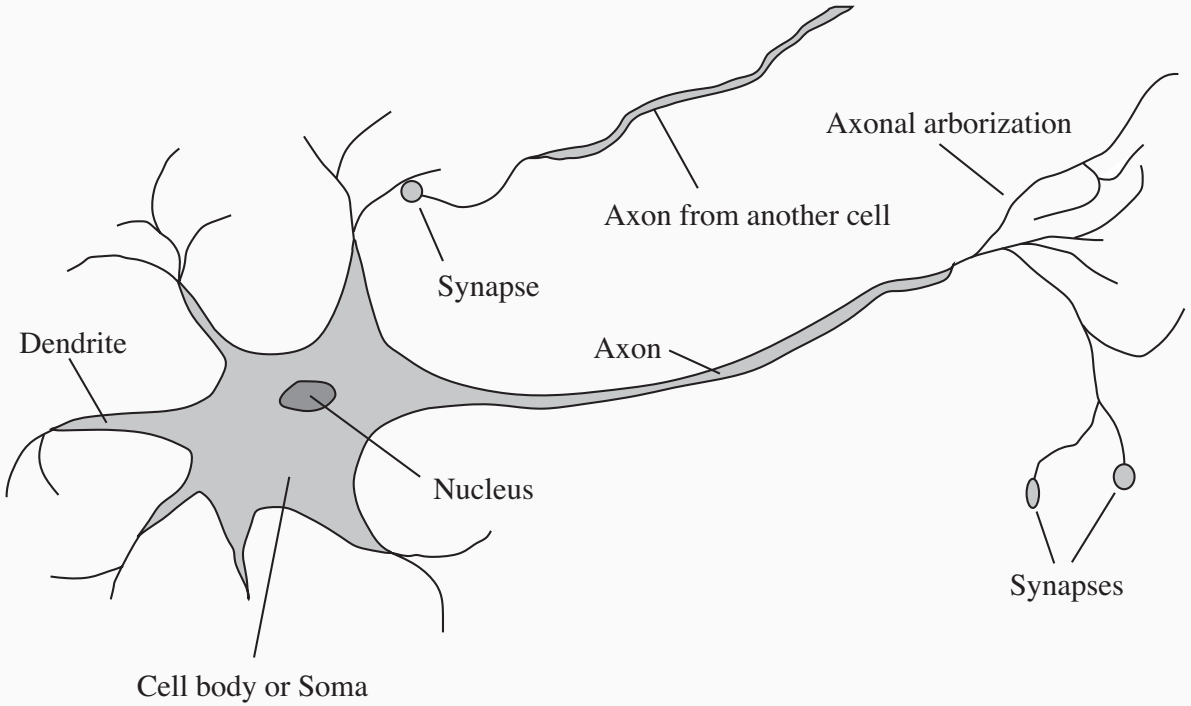
Valdinei Freire da Silva

valdinei.freire@usp.br - Bloco A1 100-O

Russell e Norvig, Capítulo 18

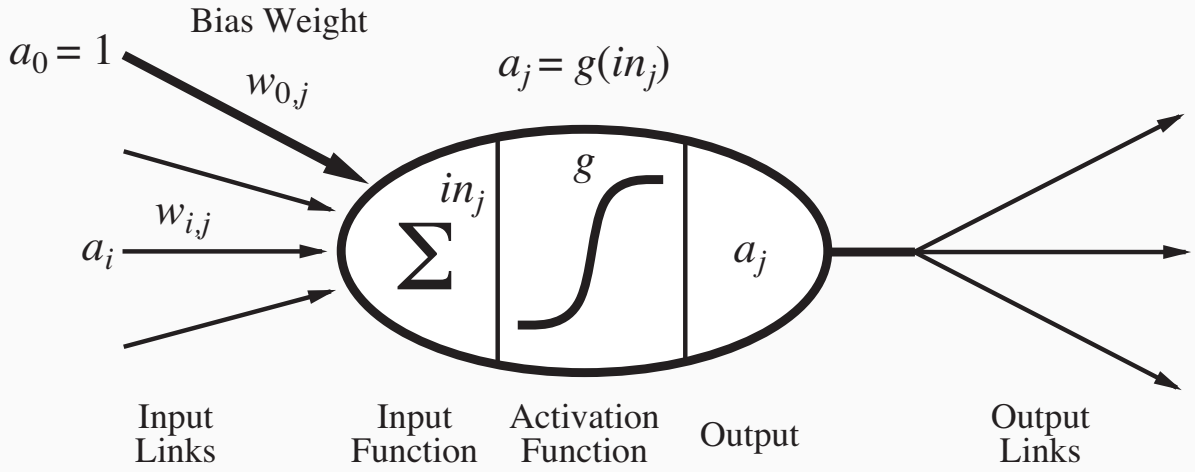
Neurônios

Hipótese: atividade mental depende de atividade eletroquímica de células do cérebro chamadas neurônios.



Perceptron: Modelo Computacional

1943: McCulloch e Pitts



Perceptron é ativado quando uma combinação linear de sua entrada atinge um *threshold*. $g(\cdot)$ é a função de ativação.

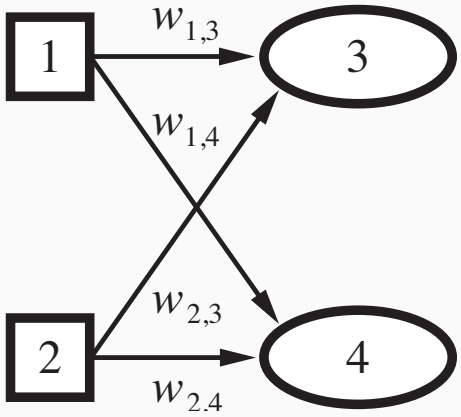
$$a_j = g\left(\sum_{i=0}^{n} w_{i,j} a_i\right)$$

Rede Neural

Rede Neural é uma coleção de perceptrons

Exemplo: operações lógicas

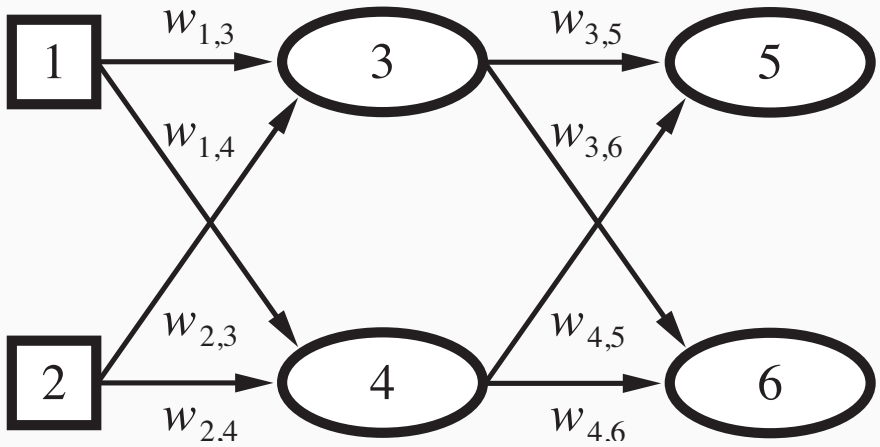
x_1	x_2	y_3 OR	y_4 AND
0	0	0	0
0	1	1	0
1	0	1	0
1	1	1	1



Quais pesos devem ser associados para cada perceptron (lembre-se do bias)?

Multilayer Feed-Forward Network

E se a entrada de um perceptron for a saída de outros perceptrons?



Pode-se pensar a rede organizada em várias camadas (*layers*): camadas de saída e camadas ocultas.

Quais pesos devem ser associados para cada perceptron para representar a função booleana XOR na saída do perceptron 5?

Multilayer Feed-Forward Network

Redes acíclicas ou de alimentação direta

- Representam uma função de sua entrada atual.
- Não têm nenhum estado interno além dos pesos.

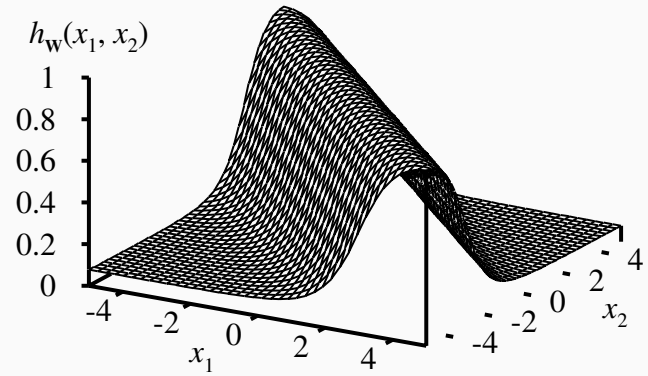
Redes cíclicas ou recorrentes.

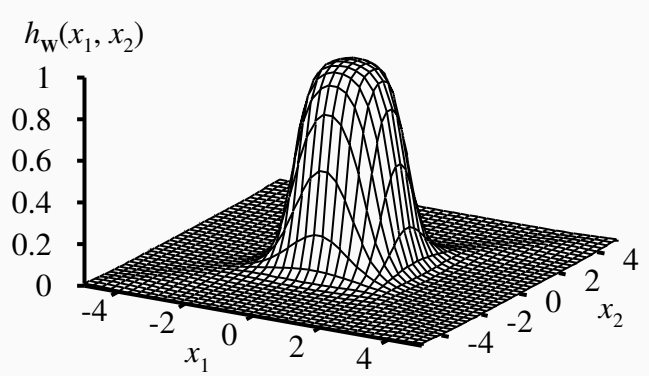
- Utilizam suas saídas para realimentar suas entradas.
- Níveis de ativação da rede formam um sistema dinâmico.
- Podem admitir memória de curto prazo.

Aproximador de função universal

Multilayer Feed-Forward Network com uma única camada oculta e a quantidade de perceptrons adequada é um aproximador de função universal.

Função Booleana: $\frac{2^n}{n}$ perceptrons ocultos são necessários.





Aproximador de função universal

Função Contínua: Seja I_m um subconjunto compacto de $\mathbb{R}^m$, e $C(I_m)$ a classe de funções contínuas no domínio I_m. Defina:

$$F(x) = \sum_{i=1}^{N} v_i g(\mathrm{w}_i^{\mathrm{T}} \mathrm{x} + b_i),$$

onde $v_i, b_i \in \mathbb{R}$, $\mathrm{w}_i \in \mathbb{R}^m$, e $g : \mathbb{R} \to \mathbb{R}$ é uma função limitada e estritamente monotônica.

Então, dada uma função $f \in C(I_m)$ e $\varepsilon > 0$, existe inteiro N e constantes $v_i, b_i \in \mathbb{R}$ e vetores reais $\mathrm{w}_i \in \mathbb{R}^m$ tais que:

$$|F(\mathrm{x}) - f(\mathrm{x})| < \varepsilon$$

para todo $x \in I_m$.

Aprendizado: Back-Propagation

Considere a perda para o conjunto de exemplos E:

$$Loss(h_{\mathrm{w}}) = -\sum_{i=1}^{n} y_i h_{\mathrm{w}}(x_i) + (1 - y_i)(1 - h_{\mathrm{w}}(x_i))$$

Busca na direção do Gradiente:

$$\mathrm{w}_{t+1} \leftarrow \mathrm{w}_t - \alpha_t \nabla_{\mathrm{w}_t} Loss(h_{\mathrm{w}_t})$$

No caso de uma rede neural, tem-se que $h_w = g(in_k)$.

Considere o perceptron k da camada de saída e o peso na entrada j deste perceptron, e seja $\Delta_k = \frac{\partial Loss(h_{\mathrm{w}_t})}{\partial in_k} = Loss'(h_{\mathrm{w}_t}) g'(in_k)$ temos que:

$$\frac{\partial Loss(h_{\mathrm{w}_t})}{\partial w_{j,k}} = \frac{\partial Loss(h_{\mathrm{w}_t})}{\partial h_{\mathrm{w}}} \frac{\partial g(in_k)}{\partial in_k} \frac{\partial in_k}{\partial w_{j,k}} = \frac{\partial Loss(h_{\mathrm{w}_t})}{\partial in_k} \frac{\partial in_k}{\partial w_{j,k}} = \Delta_k a_j$$

Aprendizado: Back-Propagation

Considere o perceptron j da camada oculta e o peso na entrada i deste perceptron, com relação ao erro da k-ésima saída temos que:

$$\begin{aligned}\frac{\partial Loss_k(h_{w_t})}{\partial w_{i,j}} &= \frac{\partial Loss(h_{w_t})}{\partial in_k}\frac{\partial in_k}{\partial a_j}\frac{\partial a_j}{\partial in_j}\frac{\partial in_j}{\partial w_{i,j}} \\ &= \Delta_k w_{j,k} g'(in_j) a_i\end{aligned}$$

Para o erro associado a todas saídas temos que:

$$\frac{\partial Loss(h_{w_t})}{\partial w_{i,j}} = \sum_{k=1}^{K} \Delta_k w_{j,k} g'(in_j) a_i = \Delta_j a_i$$

onde $\Delta_j = g'(in_j)\sum_{k=1}^{K} w_{j,k}\Delta_k$.

Algoritmo: Back-Propagation

1. Inicializa todos os pesos com valores aleatórios
2. Propaga *forward* a entrada na rede para computar a saída de cada perceptron

$$a_j \leftarrow g(in_j) = g\left(\sum_i w_{i,j} a_i\right)$$

3. Propaga *backward* os deltas da camada de saída até a camada de entrada

$$\text{camada de saída: } \Delta_k \leftarrow Loss'(h_{\mathbf{w}_t}) g'(in_k)$$
$$\text{camada oculta: } \Delta_j \leftarrow g'(in_j) \sum_{k=1}^{K} w_{j,k} \Delta_k$$

4. Atualiza todos os pesos $w_{i,j}$ na rede

$$w_{i,j} \leftarrow w_{i,j} - \alpha a_i \Delta_j$$

5. Repete até algum critério de convergência

Algoritmo: Back-Propagation

Critério de Convergência:

- gradiente baixo
- teste de validação
- quantidade de iterações

Inicialização dos Pesos:

- aleatória: baixo desempenho, mínimos locais ruins
- Nguyen e Widrow: distribui a ativação dos perceptrons da camada oculta no domínio de x

Modo de Treinamento:

- batch (lote ou batelada)
- estocástica

Sugestões para a Semana Santa

- Lista de exercícios do Russell e Norvig
- Implementar algoritmos estudados e comparar com os resultado das aulas
- Pesquisar sobre o algoritmo de inicialização de Nguyen-Widrow
- Pesquisar sobre algoritmos de gradiente descendente: Adam, Adagrad, RMSProp, etc.