Redes Neurais

Prof. Danilo Silva

EEL7514/EEL7513 - Tópico Avançado em Processamento de Sinais:
Introdução ao Aprendizado de Máquina

EEL / CTC / UFSC

Notação: Regressão Linear/Logística Vetorial

- Suponha que $\mathbf{y} = (y_1, \ldots, y_K)^T \in \mathbb{R}^K$
  - Exemplo: classificação multi-classe com codificação 1-de-K
  - Não confundir a notação com o vetor de rótulos de treinamento
- Para cada y_k, temos parâmetros $\mathbf{w}_k = (\mathbf{w}_{k,1}, \ldots, \mathbf{w}_{k,n})^T$ e $b_k \in \mathbb{R}$
- Predição:

$$\hat{y}_k = \mathbf{w}_k^T \mathbf{x} + b_k \quad \text{(linear)} \quad \text{ou} \quad \hat{y}_k = \sigma(\mathbf{w}_k^T \mathbf{x} + b_k) \quad \text{(logística)}$$

- Em notação vetorial:

$$\hat{\mathbf{y}} = \mathbf{W}\mathbf{x} + \mathbf{b} \quad \text{(linear)} \quad \text{ou} \quad \hat{\mathbf{y}} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b}) \quad \text{(logística)}$$

onde

$$\mathbf{W} = \begin{bmatrix} — \ \mathbf{w}_1^T \ — \\ \vdots \\ — \ \mathbf{w}_K^T \ — \end{bmatrix} \in \mathbb{R}^{K \times n} \quad \text{e} \quad \mathbf{b} = \begin{bmatrix} b_1 \\ \vdots \\ b_K \end{bmatrix} \in \mathbb{R}^K$$

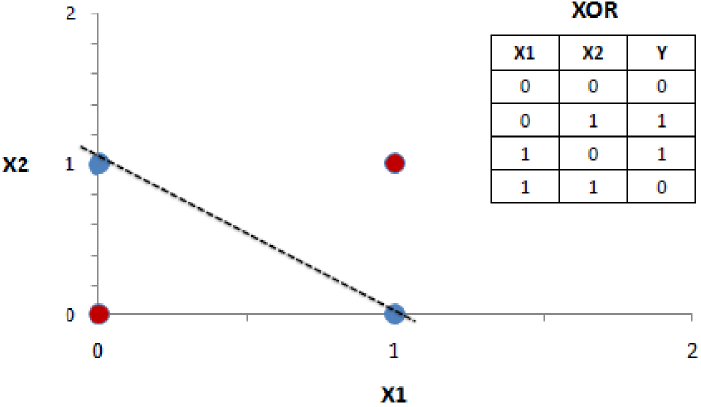
Motivação: Aumentando a Capacidade do Modelo

- Suponha por hora que não estamos preocupados com generalização, apenas em representar com mínimo erro o conjunto de treinamento
- Suponha que exista uma função $f^* : \mathbb{R}^n \to \mathbb{R}^K$ e que dispomos de m amostras (possivelmente ruidosas) de pares $(\mathbf{x}, \mathbf{y})$, onde $\mathbf{y} \approx f^*(\mathbf{x})$
- Nosso objetivo é encontrar uma função f que aproxima f^*
- Regressão logística (com os atributos originais $\mathbf{x}$), isto é,

  $$\hat{\mathbf{y}} = f(\mathbf{x}) = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$

  funciona bem quando as classes são separáveis por hiperplanos
- Caso contrário, precisamos criar novos atributos derivados dos originais para obter regiões de decisão mais complexas

Exemplo: XOR

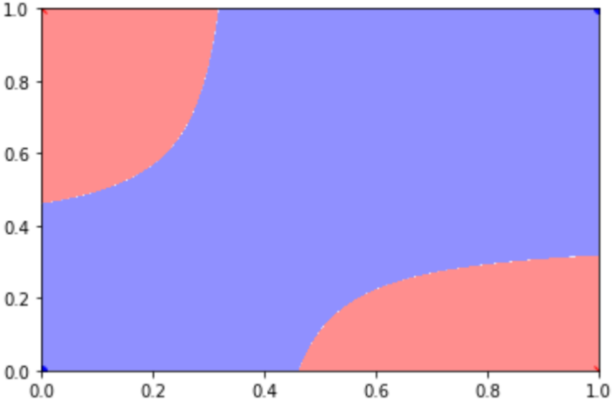


XOR

X1	X2	Y
0	0	0
0	1	1
1	0	1
1	1	0

É impossível representar os dados com os atributos originais x_1 e x_2, mas é possível adicionando o termo $x_1 x_2$

Exemplo: XOR



É impossível representar os dados com os atributos originais x_1 e x_2, mas é possível adicionando o termo $x_1 x_2$

Determinação de Atributos

- Como escolher os atributos derivados?
  - Escolha manual:
    - requer “criatividade” e conhecimento específico do problema
    - se adicionarmos atributos polinomiais até grau d, ocorrerá uma explosão de termos: total de $\binom{n+d}{d} \geq (n/d)^d$ atributos
    - cresce rapidamente com o aumento de n
  - Escolha automática:
    - Função genérica suficientemente flexível com parâmetros que podem ser encontrados via treinamento
    - Troca “engenharia de atributos” por “aprendizagem de atributos” (*feature learning / representation learning*)

Redes Neurais *Feedforward* (sem realimentação)

- A função $f(\mathbf{x})$ é construída através da composição de L funções vetoriais $f_\ell : \mathbb{R}^{n_{\ell-1}} \to \mathbb{R}^{n_\ell}$, com $n_0 = n$ e $n_L = K$:

$$f(\mathbf{x}) = f_L(f_{L-1}(\cdots f_2(f_1(\mathbf{x})))) = \hat{\mathbf{y}}$$

  onde

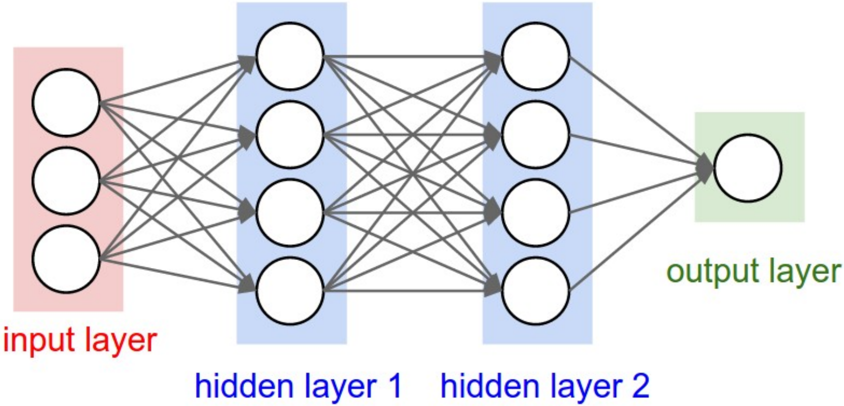
$$f_\ell(\mathbf{a}^{[\ell-1]}) = g_\ell(\mathbf{W}^{[\ell]}\mathbf{a}^{[\ell-1]} + \mathbf{b}^{[\ell]}) = \mathbf{a}^{[\ell]}$$

  - L é o número de camadas ou profundidade da rede
  - n_ℓ é o número de unidades ou largura da camada ℓ
  - $\mathbf{W}^{[\ell]} \in \mathbb{R}^{n_\ell \times n_{\ell-1}}$ é matriz de pesos da camada ℓ
  - $\mathbf{b}^{[\ell]} \in \mathbb{R}^{n_\ell}$ é o vetor de *bias* da camada ℓ
  - $\mathbf{a}^{[\ell]} \in \mathbb{R}^{n_\ell}$ é o vetor de ativações da camada ℓ (com $\mathbf{a}^{[0]} = \mathbf{x}$ e $\mathbf{a}^{[L]} = \hat{\mathbf{y}}$)
  - $g_\ell : \mathbb{R} \to \mathbb{R}$ é a função de ativação (não-linear) da camada ℓ, aplicada a cada elemento de um vetor em $\mathbb{R}^{n_\ell}$, i.e.,

$$\mathbf{z} = (z_1, \ldots, z_{n_\ell})^T \quad \Longrightarrow \quad g_\ell(\mathbf{z}) = (g_\ell(z_1), \ldots, g_\ell(z_{n_\ell}))^T$$

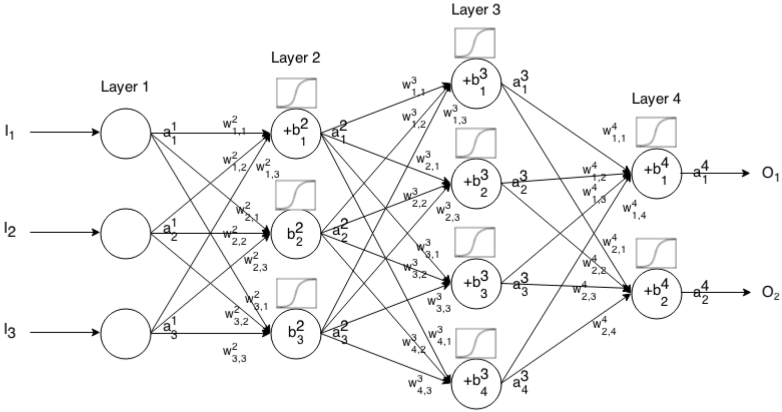
    Tipicamente escolhidas iguais, $g_\ell(z) = g(z)$, exceto possivelmente $g_L(z)$

Redes Neurais *Feedforward* (sem realimentação)



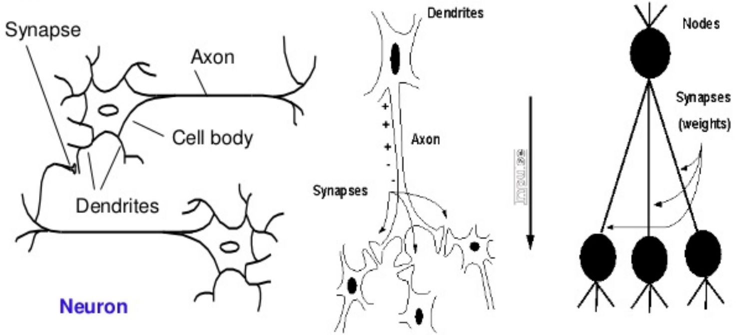
- Rede de $L = 3$ camadas (não contamos a camada de entrada)

Redes Neurais *Feedforward* (sem realimentação)



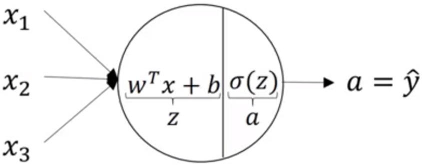
- Rede de $L = 3$ camadas (não contamos a camada de entrada)
- Obs: notação errada: a camada de entrada tem índice $\ell = 0$

Inspiração Biológica



- São chamadas de redes neurais pois seu funcionamento é inspirado pelo funcionamento dos neurônios no cérebro humano
- Porém, não sabemos como o cérebro realmente funciona e o nosso objetivo não é modelá-lo mas simplesmente aproximar uma função
- Em muitos casos as melhores soluções computacionais se afastam da solução biológica

Unidade (ou Neurônio)



$$z = w^T x + b$$

- Unidade computacional que realiza duas tarefas:
  - Ponderar linearmente as ativações da camada anterior (entradas)
  - Produzir uma ativação (saída) pela aplicação de uma função não-linear
- Exemplo de função de ativação: $g(z) = \sigma(z)$ (sigmóide logística)
- A não-linearidade é essencial para garantir a flexibilidade do modelo

Exemplo: AND

Exemplo: XOR

Aprendizagem via Redes Neurais

Dois aspectos ajudam a explicar o sucesso das redes neurais:

- São modelos extremamente flexíveis, capazes de representar qualquer função (desde que com largura e profundidade suficientemente grandes)
  - Prova: basta ser capaz de implementar portas lógicas
- Possuem um algoritmo eficiente de treinamento: *backpropagation*

Calculando a saída da rede: *Forward propagation*

- Dado o vetor de entrada $\mathbf{x}$, calcula-se:

$$
\begin{aligned}
\mathbf{a}^{[0]} &= \mathbf{x} \\
\mathbf{z}^{[1]} &= \mathbf{W}^{[1]}\mathbf{a}^{[0]} + \mathbf{b}^{[1]} \\
\mathbf{a}^{[1]} &= g(\mathbf{z}^{[1]}) \\
&\vdots \\
\mathbf{z}^{[\ell]} &= \mathbf{W}^{[\ell]}\mathbf{a}^{[\ell-1]} + \mathbf{b}^{[\ell]} \\
\mathbf{a}^{[\ell]} &= g(\mathbf{z}^{[\ell]}) \\
&\vdots \\
\mathbf{z}^{[L]} &= \mathbf{W}^{[L]}\mathbf{a}^{[L-1]} + \mathbf{b}^{[L]} \\
\mathbf{a}^{[L]} &= g(\mathbf{z}^{[L]}) \\
\hat{\mathbf{y}} &= \mathbf{a}^{[L]}
\end{aligned}
$$

Função Custo

- Assumindo como função perda a entropia cruzada

$$L(\hat{y}, y) = -y \log \hat{y} - (1 - y) \log(1 - \hat{y})$$

- Função custo:

$$J(\{\mathbf{W}^{[\ell]}, \mathbf{b}^{[\ell]}\}) = -\frac{1}{m} \sum_{i=1}^{m} \sum_{k=1}^{K} y_k^{(i)} \log \hat{y}_k^{(i)} + (1 - y_k^{(i)}) \log(1 - \hat{y}_k^{(i)})$$

onde $\hat{\mathbf{y}}^{(i)} = \mathbf{a}^{[L](i)}$ é a saída correspondente ao i-ésimo exemplo de treinamento

Treinamento: Método do gradiente

- Repita até a convergência (ou um número máximo de iterações):

$$
\begin{aligned}
\mathbf{W}^{[1]} &\leftarrow \mathbf{W}^{[1]} - \alpha \frac{\partial J}{\partial \mathbf{W}^{[1]}} \\
\mathbf{b}^{[1]} &\leftarrow \mathbf{b}^{[1]} - \alpha \frac{\partial J}{\partial \mathbf{b}^{[1]}} \\
&\vdots \\
\mathbf{W}^{[\ell]} &\leftarrow \mathbf{W}^{[\ell]} - \alpha \frac{\partial J}{\partial \mathbf{W}^{[\ell]}} \\
\mathbf{b}^{[\ell]} &\leftarrow \mathbf{b}^{[\ell]} - \alpha \frac{\partial J}{\partial \mathbf{b}^{[\ell]}} \\
&\vdots \\
\mathbf{W}^{[L]} &\leftarrow \mathbf{W}^{[L]} - \alpha \frac{\partial J}{\partial \mathbf{W}^{[L]}} \\
\mathbf{b}^{[L]} &\leftarrow \mathbf{b}^{[L]} - \alpha \frac{\partial J}{\partial \mathbf{b}^{[L]}}
\end{aligned}
$$

Cálculo do Gradiente: *Backward Propagation*

- Regra da cadeia
- Calcular $\frac{\partial J}{\partial z} = \frac{\partial J}{\partial a} \cdot \frac{\partial a}{\partial z}$ para uma unidade sigmóide $a = \sigma(z)$ com função custo de entropia cruzada $J(a, y) = -y \log a - (1 - y) \log(1 - a)$