



UNIVERSIDADE
FEDERAL DO CEARÁ



LogIA

Aprendizagem de Máquina

César Lincoln Cavalcante Mattos

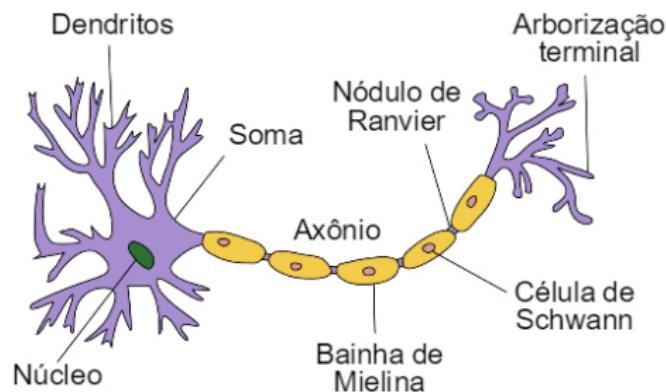
2025

Agenda

- ① Redes Neurais Artificiais Lineares
- ② Redes Neurais Artificiais Não-Lineares
- ③ Algoritmo backpropagation
- ④ Técnicas para treinamento de redes neurais
- ⑤ Tópicos adicionais
- ⑥ Referências

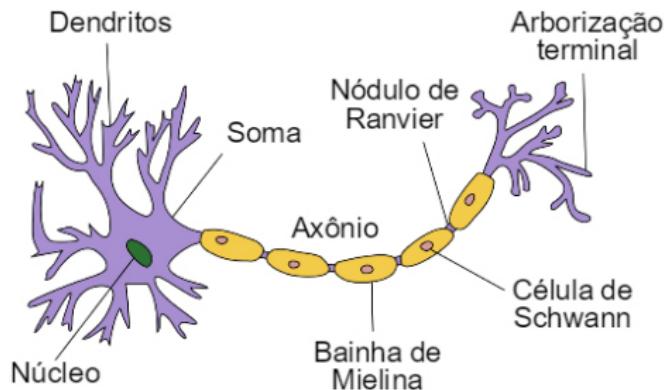
Redes Neurais Artificiais

- Modelos “**inspirados**” no neurônio biológico.



Redes Neurais Artificiais

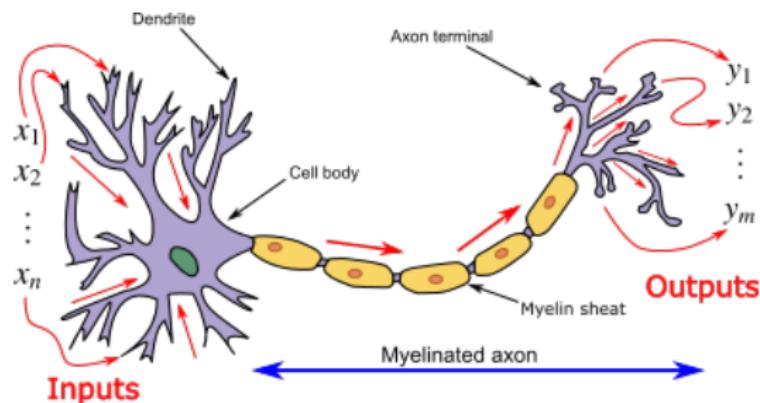
- Modelos “**inspirados**” no neurônio biológico.



- **Importante:** RNAs **não pretendem** ser biologicamente plausíveis.

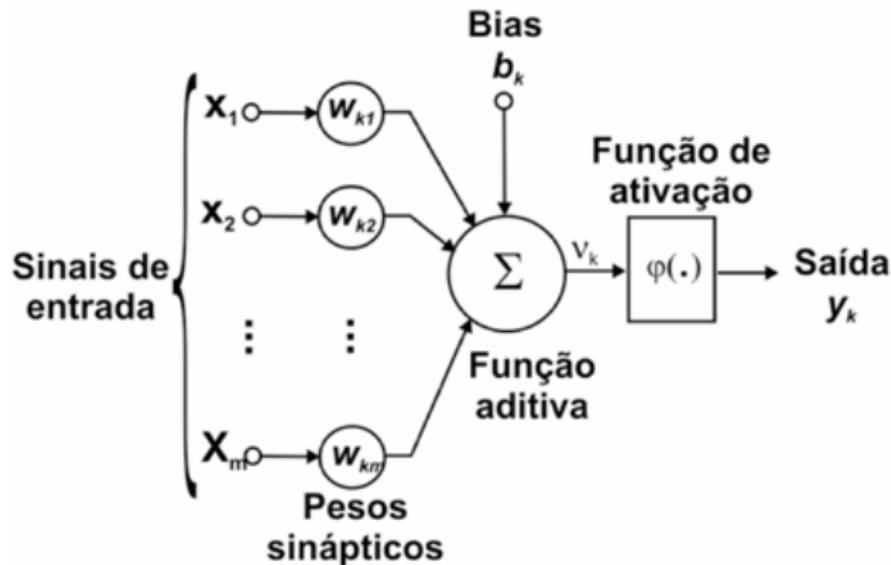
Redes Neurais Artificiais

- Modelos “**inspirados**” no neurônio biológico.



- **Importante:** RNAs **não pretendem** ser biologicamente plausíveis.

Modelo neural de McCulloch-Pitts (1943)



Redes Neurais Artificiais

Perceptron de Rosenblatt (1957)

- Modelo linear com ativação $\text{sign}(\cdot)$ (classificação binária):

$$\hat{y}_i = \text{sign}(\mathbf{w}^\top \mathbf{x}_i),$$

$$\mathbf{x}_i = [1, x_1, x_2, \dots, x_D]^\top,$$

$$\text{sign}(z) = \begin{cases} -1, & z < 0 \\ 1, & z \geq 0 \end{cases}$$

- **Função custo:**

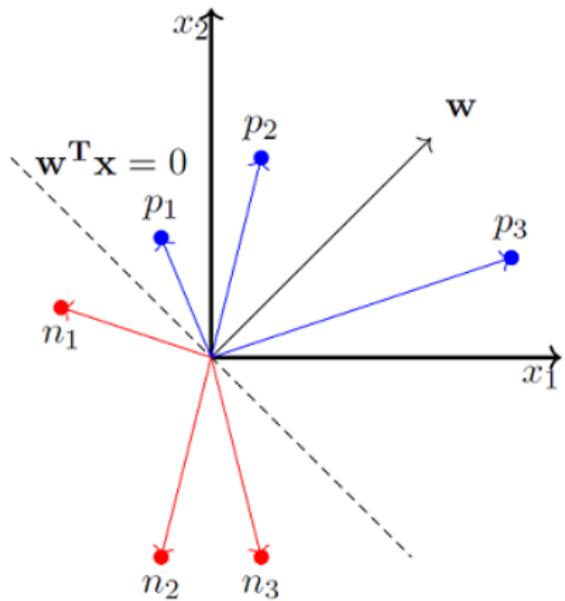
$$J(\mathbf{w}) = \sum_{i=1}^N \max(0, -y_i \hat{y}_i) = - \sum_{\forall i: y_i \neq \hat{y}_i} y_i \hat{y}_i.$$

- **Observação:** Padrões corretamente classificados não contribuem para a função custo.

Perceptron de Rosenblatt

- O produto interno $\mathbf{w}^\top \mathbf{x}_i$ pode ser analisado geometricamente:

$$\mathbf{w}^\top \mathbf{x}_i = \|\mathbf{w}\| \|\mathbf{x}_i\| \cos \beta \rightarrow \begin{cases} \cos \beta > 0, & 0^\circ \leq \beta < 90^\circ \\ \cos \beta < 0, & 90^\circ < \beta \leq 180^\circ \end{cases}$$



- Aproximamos o vetor \mathbf{w} dos padrões “positivos” e afastamos dos “negativos”.
- A superfície de decisão é definida por $\mathbf{w}^\top \mathbf{x} = 0$.

Perceptron de Rosenblatt

Algoritmo de treinamento do Perceptron de Rosenblatt

- ① Inicialize os pesos $\mathbf{w}(0)$;
- ② Repita por diversos ciclos (épocas):
 - ① Permute aleatoriamente a ordem dos dados.
 - ② Para cada padrão de entrada, $i = 1, \dots, N$, repita:
 - ① Faça $t \leftarrow t + 1$.
 - ② Calcule a saída do modelo: $\hat{y}_i = \text{sign}(\mathbf{w}(t)^\top \mathbf{x}_i)$;
 - ③ Calcule o erro obtido: $e_i = y_i - \hat{y}_i$;
 - ④ Atualize os pesos pela **regra de aprendizagem do Perceptron**:

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \alpha e_i \mathbf{x}_i,$$

em que $\alpha > 0$ é um passo de aprendizagem opcional.

Perceptron de Rosenblatt

Algoritmo de treinamento do Perceptron de Rosenblatt

- ① Inicialize os pesos $w(0)$;
- ② Repita por diversos ciclos (épocas):
 - ① Permute aleatoriamente a ordem dos dados.
 - ② Para cada padrão de entrada, $i = 1, \dots, N$, repita:
 - ① Faça $t \leftarrow t + 1$.
 - ② Calcule a saída do modelo: $\hat{y}_i = \text{sign}(w(t)^\top x_i)$;
 - ③ Calcule o erro obtido: $e_i = y_i - \hat{y}_i$;
 - ④ Atualize os pesos pela **regra de aprendizagem do Perceptron**:

$$w(t+1) = w(t) + \alpha e_i x_i,$$

em que $\alpha > 0$ é um passo de aprendizagem opcional.

- Note que no Perceptron de Rosenblatt o erro é quantizado:

$$e_i = y_i - \hat{y}_i = \begin{cases} 0 & , y_i = \hat{y}_i \\ -2 & , y_i = -1 \text{ e } \hat{y}_i = 1 \\ 2 & , y_i = 1 \text{ e } \hat{y}_i = -1 \end{cases}$$

Perceptron de Rosenblatt



Hardware do Mark 1 perceptron, capaz de classificar imagens de caracteres. À direita, potenciômetros para ajuste dos pesos. O ajuste podia ser automatizado via motores elétricos.

Redes Neurais Artificiais

ADALINE (*Adaptive Linear Element*) (Widrow & Hoff, 1960)

- Modelo linear semelhante ao Perceptron.
- Calcula o erro obtido antes da aplicação da função $\text{sign}(\cdot)$:

$$e_i = y_i - \mathbf{w}^\top \mathbf{x}_i.$$

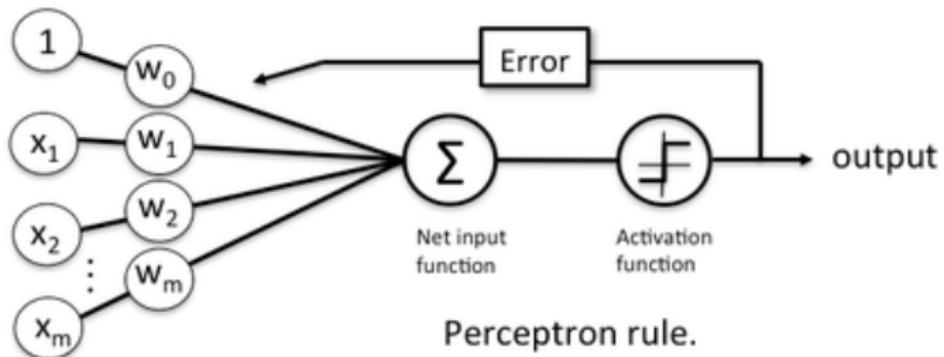
- **Função custo:**

$$J(\mathbf{w}) = \frac{1}{2N} \sum_{i=1}^N (y_i - \mathbf{w}^\top \mathbf{x}_i)^2.$$

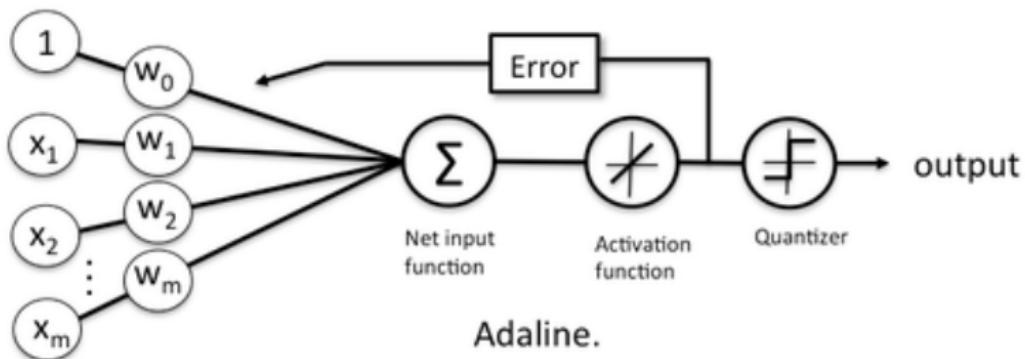
- Atualize os pesos pelo algoritmo LMS (*Least Mean Squares*):

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \alpha e_i \mathbf{x}_i.$$

Perceptron × ADALINE



Perceptron rule.



Adaline.

Redes Neurais Artificiais

- K saídas: $\hat{y}_{ik} = \text{sign}(\mathbf{w}_k^\top \mathbf{x}_i)$, $\mathbf{W} = [\mathbf{w}_1 \dots \mathbf{w}_k]^\top \in \mathbb{R}^{K \times (D+1)}$.

Multiple Perceptron

- **Função custo:**

$$J(\mathbf{W}) = \sum_{i=1}^N \sum_{k=1}^K \max(0, -y_{ik} \hat{y}_{ik}).$$

MADALINE (*Multiple ADALINE*)

- **Função custo:**

$$J(\mathbf{W}) = \frac{1}{2N} \sum_{i=1}^N \sum_{k=1}^K (y_{ik} - \mathbf{w}_k^\top \mathbf{x}_i)^2.$$

- Atualização dos pesos:

$$\mathbf{w}_k(t+1) = \mathbf{w}_k(t) + \alpha e_{ik} \mathbf{x}_i, \quad e_{ik} = \begin{cases} y_{ik} - \text{sign}(\mathbf{w}_k^\top(t) \mathbf{x}_i), & \text{Perceptron} \\ y_{ik} - \mathbf{w}_k^\top(t) \mathbf{x}_i, & \text{ADALINE} \end{cases}$$

Redes Neurais Artificiais

“O perceptron mostrou-se digno de estudo, apesar (e até por causa!) de suas severas limitações. Ele possui muitos recursos que atraem a atenção: sua linearidade; seu teorema de aprendizado intrigante; sua clara simplicidade paradigmática como uma espécie de computação paralela.

Redes Neurais Artificiais

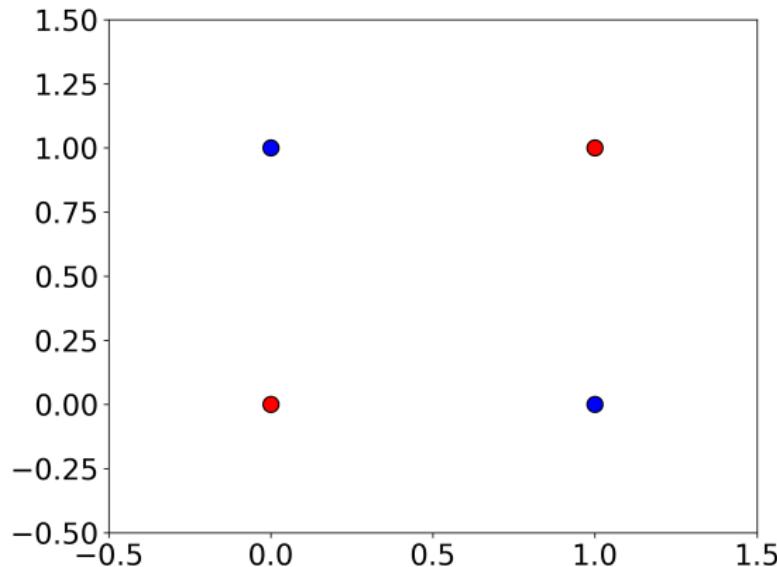
“O perceptron mostrou-se digno de estudo, apesar (e até por causa!) de suas severas limitações. Ele possui muitos recursos que atraem a atenção: sua linearidade; seu teorema de aprendizado intrigante; sua clara simplicidade paradigmática como uma espécie de computação paralela. **Não há razão para supor que qualquer uma dessas virtudes seja transferida para sua versão multicamadas.** Todavia, consideramos um importante problema de pesquisa elucidar (ou rejeitar) **nosso julgamento intuitivo de que sua extensão multicamadas é estéril.”**
(Minsky & Papert, Perceptrons, 1969)

Agenda

- ① Redes Neurais Artificiais Lineares
- ② Redes Neurais Artificiais Não-Lineares
- ③ Algoritmo backpropagation
- ④ Técnicas para treinamento de redes neurais
- ⑤ Tópicos adicionais
- ⑥ Referências

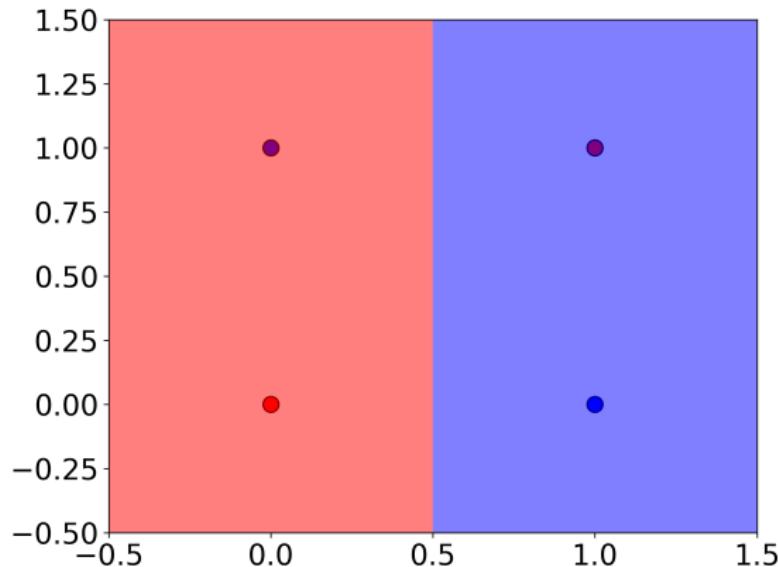
Redes Neurais Artificiais

- **Problema:** O Perceptron é capaz de resolver o problema da porta lógica XOR?



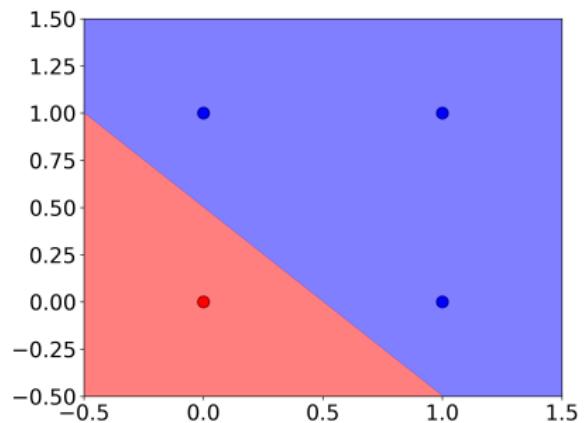
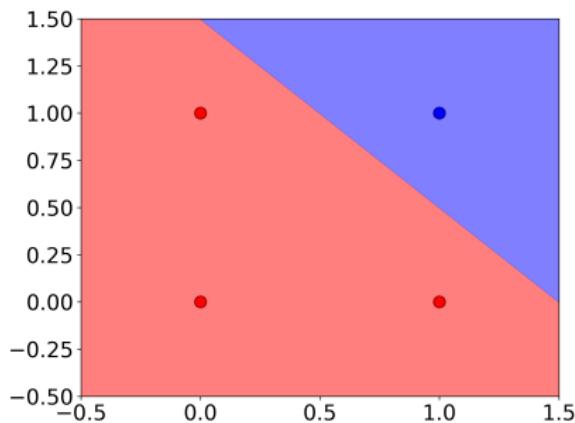
Redes Neurais Artificiais

- **Problema:** O Perceptron é capaz de resolver o problema da porta lógica XOR? **Não!**



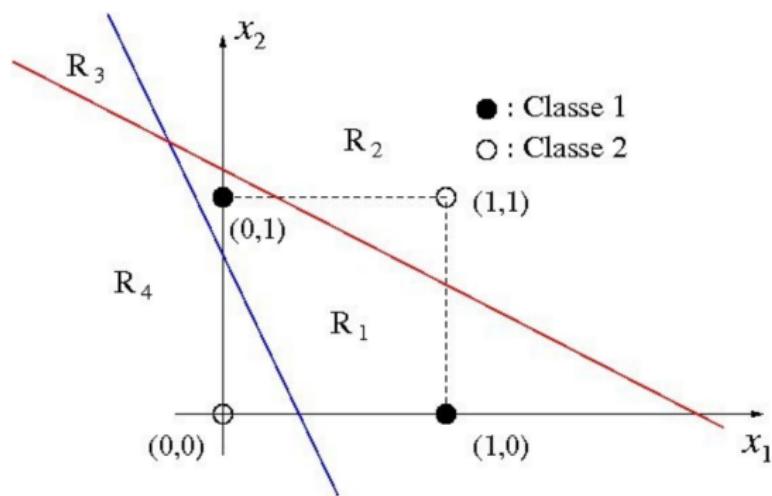
Redes Neurais Artificiais

- **Ideia:** Treinar 2 perceptrons, um com a porta AND e outro com a porta OR.



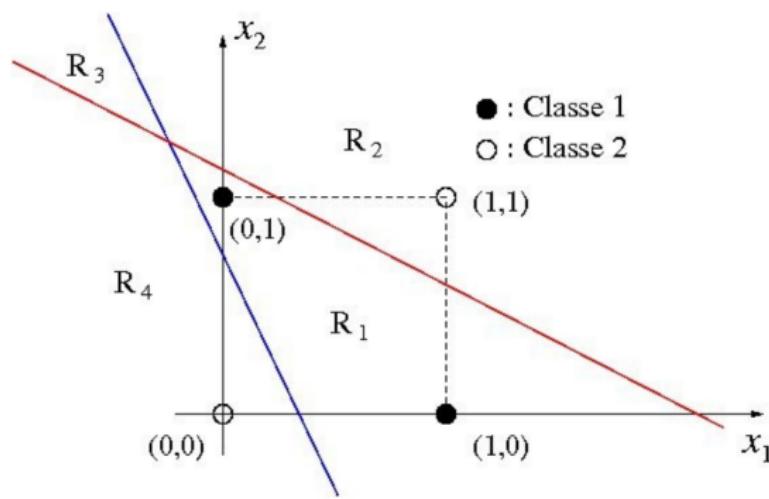
Redes Neurais Artificiais

- Reunimos as fronteiras de decisão de ambos os perceptrons:



Redes Neurais Artificiais

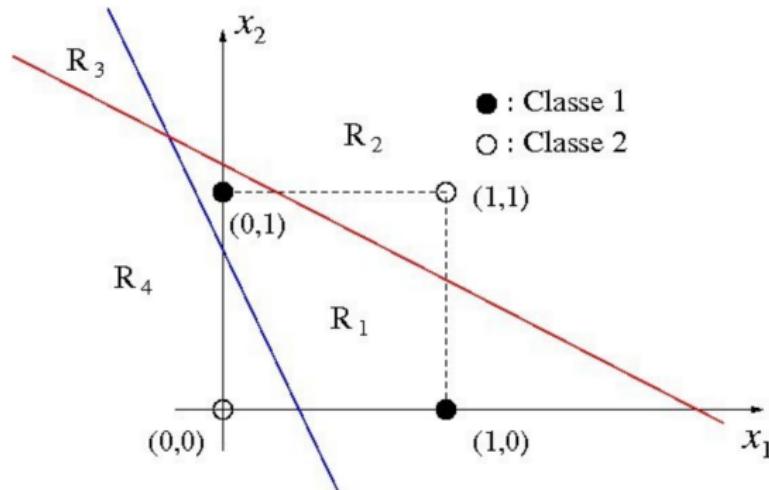
- Reunimos as fronteiras de decisão de ambos os perceptrons:



- **Problema:** Seria necessária uma tomada de decisão adicional para obter a porta XOR.

Redes Neurais Artificiais

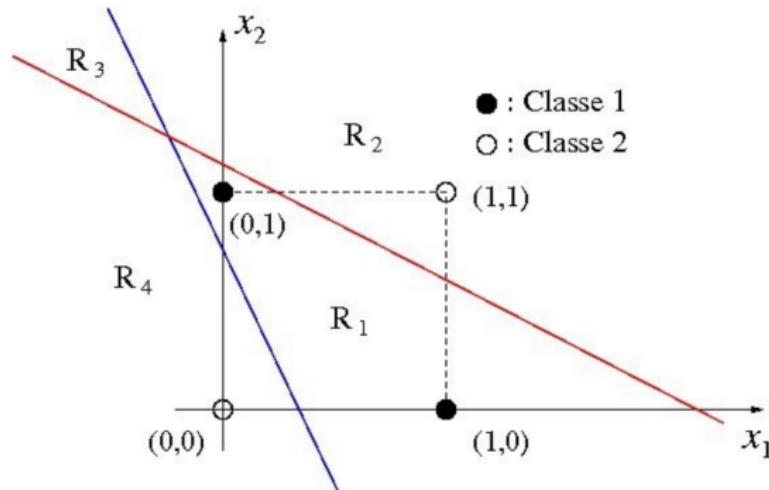
- Reunimos as fronteiras de decisão de ambos os perceptrons:



- **Ideia:** Seja $z^{(AND)}$ e $z^{(OR)}$ as saídas dos perceptrons.
 - Em R_1 : $z^{(AND)} = 0$, $z^{(OR)} = 1$
 - Em R_2 : $z^{(AND)} = 1$, $z^{(OR)} = 1$
 - Em R_3 : $z^{(AND)} = 1$, $z^{(OR)} = 0$
 - Em R_4 : $z^{(AND)} = 0$, $z^{(OR)} = 0$

Redes Neurais Artificiais

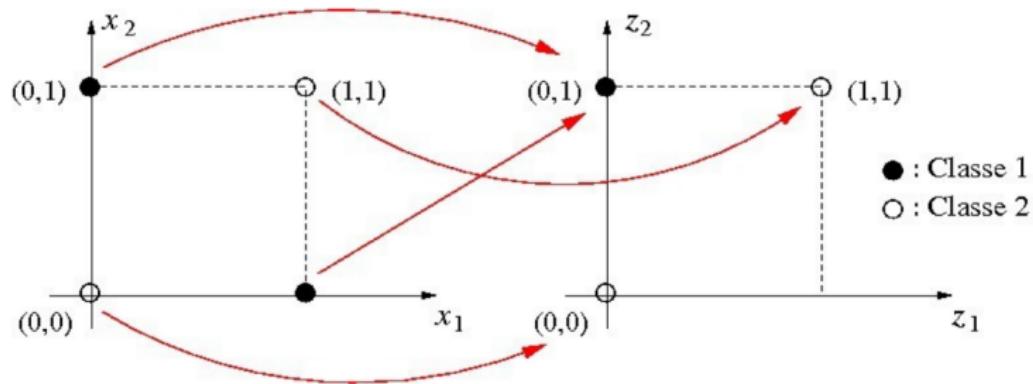
- Reunimos as fronteiras de decisão de ambos os perceptrons:



- **Ideia:** Seja $z^{(AND)}$ e $z^{(OR)}$ as saídas dos perceptrons.
 - Em R_1 : $z^{(AND)} = 0$, $z^{(OR)} = 1 \rightarrow \text{XOR} = 1$
 - Em R_2 : $z^{(AND)} = 1$, $z^{(OR)} = 1 \rightarrow \text{XOR} = 0$
 - Em R_3 : $z^{(AND)} = 1$, $z^{(OR)} = 0 \rightarrow \text{XOR} = 0$
 - Em R_4 : $z^{(AND)} = 0$, $z^{(OR)} = 0 \rightarrow \text{XOR} = 0$

Redes Neurais Artificiais

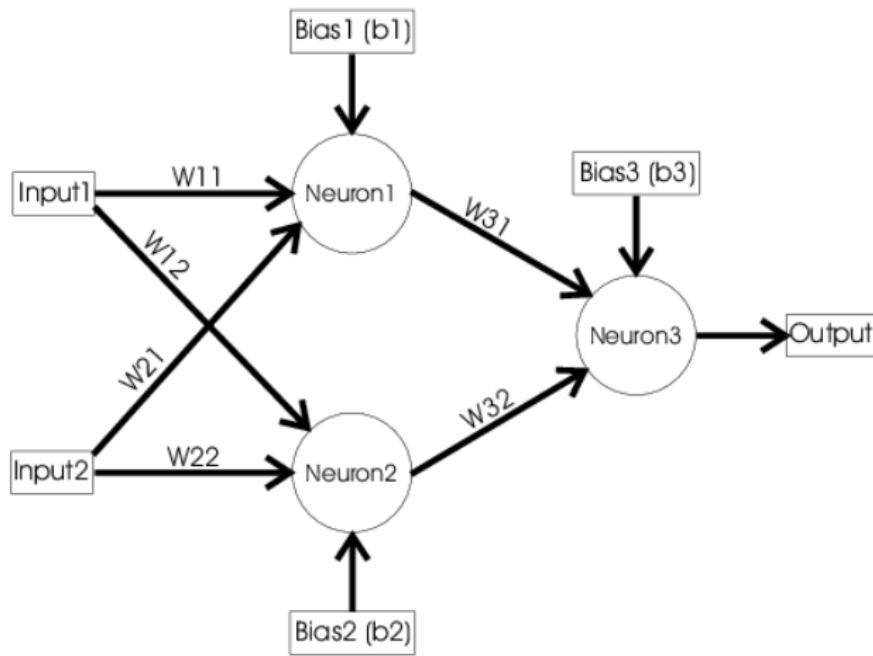
- O resultado equivale a uma nova representação dos dados, que tornam-se linearmente separáveis.



- Ideia:** Seja $z^{(AND)}$ e $z^{(OR)}$ as saídas dos perceptrons.
 - Em R_1 : $z^{(AND)} = 0$, $z^{(OR)} = 1 \rightarrow \text{XOR} = 1$
 - Em R_2 : $z^{(AND)} = 1$, $z^{(OR)} = 1 \rightarrow \text{XOR} = 0$
 - Em R_3 : $z^{(AND)} = 1$, $z^{(OR)} = 0 \rightarrow \text{XOR} = 0$
 - Em R_4 : $z^{(AND)} = 0$, $z^{(OR)} = 0 \rightarrow \text{XOR} = 0$

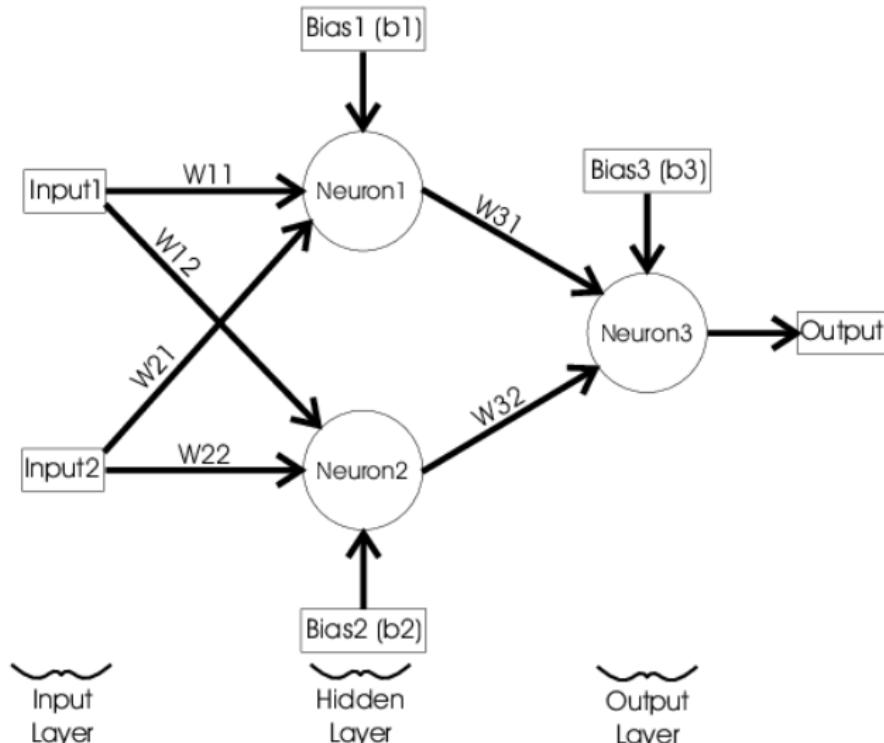
Redes Neurais Artificiais

- Arquitetura do novo modelo com 3 perceptrons, cada um contendo seus próprios parâmetros:



Redes Neurais Artificiais

- **Camada oculta (hidden layer)**: Neurônios que não possuem acesso direto à saída.

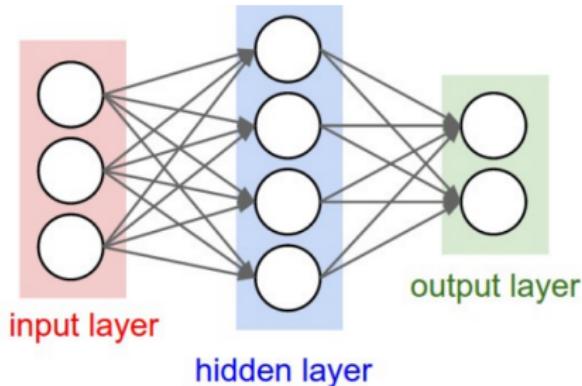
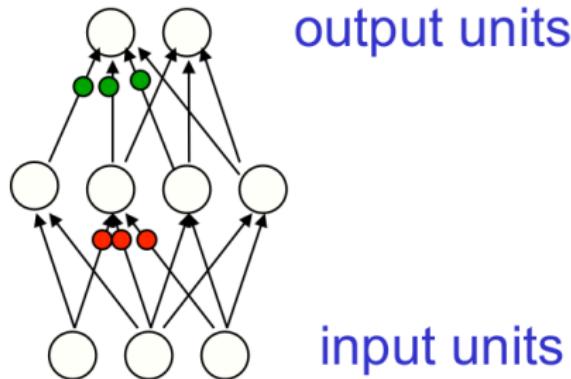


Redes Neurais Artificiais

Multilayer Perceptron (MLP)

Perceptron Multicamadas

- RNAs contendo uma ou mais camadas de **neurônios ocultos**.
- Possui **funções de ativação ocultas não-lineares**.
- Capaz de resolver **problemas não-lineares**.



Redes Neurais Artificiais

Função de ativação

Função aplicada na saída de um neurônio: $\phi(\mathbf{w}^\top \mathbf{x}_i)$

→ **Sigmóide:**

$$\sigma(z) = \frac{1}{1 + \exp(-z)}, \quad \frac{d\sigma(z)}{dz} = \sigma(z) - \sigma(z)^2$$

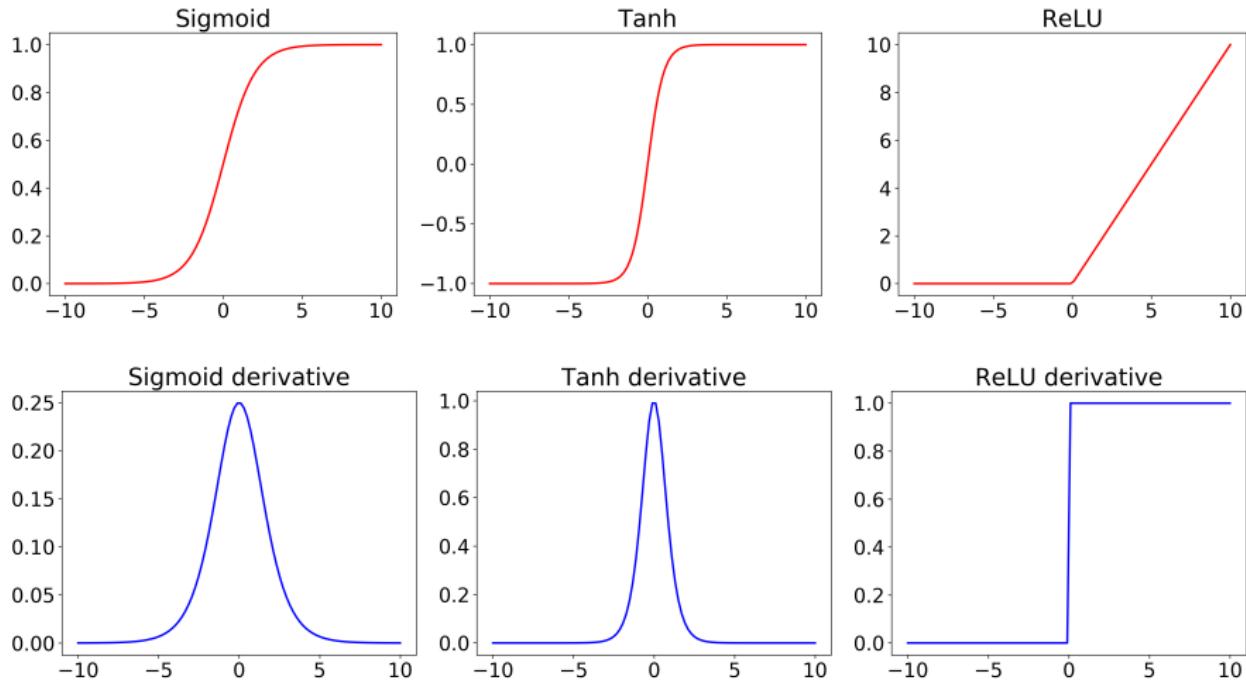
→ **Tangente hiperbólica:**

$$\tanh(z) = \frac{\exp(2z) - 1}{\exp(2z) + 1}, \quad \frac{d\tanh(z)}{dz} = 1 - \tanh(z)^2$$

→ **Rectified Linear Unit (ReLU):**

$$\text{relu}(z) = \max(0, z), \quad \frac{d\text{relu}(z)}{dz} = \begin{cases} 0, & z \leq 0 \\ 1, & z > 0 \end{cases}$$

Funções de ativação mais comuns



Multilayer Perceptron (MLP)

Perceptron Multicamadas

- Considere os dados $(\mathbf{x}_i, \mathbf{y}_i) |_{i=1}^N$, $\mathbf{x}_i \in \mathbb{R}^D$, $\mathbf{y}_i \in \mathbb{R}^K$. Além disso:
 - seja uma rede MLP de 1 camada oculta com N_H neurônios;
 - sejam K neurônios de saída;
 - sejam $\phi_1(\cdot)$ e $\phi_2(\cdot)$ as funções de ativação da camada oculta e da camada de saída, respectivamente.

Multilayer Perceptron (MLP)

Sentido direto da MLP

- Saída do j -ésimo neurônio da camada oculta para a entrada \mathbf{x}_i :

$$z_j = \phi_1(\mathbf{w}_j^\top \mathbf{x}_i), \quad 1 \leq j \leq N_H,$$

$$\mathbf{x}_i = [1, x_{i1}, x_{i2}, \dots, x_{iD}]^\top,$$

$$\mathbf{w}_j \in \mathbb{R}^{D+1}.$$

Multilayer Perceptron (MLP)

Sentido direto da MLP

- Saída do j -ésimo neurônio da camada oculta para a entrada \mathbf{x}_i :

$$z_j = \phi_1(\mathbf{w}_j^\top \mathbf{x}_i), \quad 1 \leq j \leq N_H,$$

$$\mathbf{x}_i = [1, x_{i1}, x_{i2}, \dots, x_{iD}]^\top,$$

$$\mathbf{w}_j \in \mathbb{R}^{D+1}.$$

- Saída do k -ésimo neurônio da camada de saída:

$$o_k = \phi_2(\mathbf{m}_k^\top \mathbf{z}), \quad 1 \leq k \leq K,$$

$$\mathbf{z} = [1, z_1, z_2, \dots, z_{N_H}]^\top,$$

$$\mathbf{m}_k \in \mathbb{R}^{N_H+1}.$$

Multilayer Perceptron (MLP)

Sentido direto da MLP

- Saída do j -ésimo neurônio da camada oculta para a entrada \mathbf{x}_i :

$$z_j = \phi_1(\mathbf{w}_j^\top \mathbf{x}_i), \quad 1 \leq j \leq N_H,$$

$$\mathbf{x}_i = [1, x_{i1}, x_{i2}, \dots, x_{iD}]^\top,$$

$$\mathbf{w}_j \in \mathbb{R}^{D+1}.$$

- Saída do k -ésimo neurônio da camada de saída:

$$o_k = \phi_2(\mathbf{m}_k^\top \mathbf{z}), \quad 1 \leq k \leq K,$$

$$\mathbf{z} = [1, z_1, z_2, \dots, z_{N_H}]^\top,$$

$$\mathbf{m}_k \in \mathbb{R}^{N_H+1}.$$

- Saída da rede:

$$\hat{\mathbf{y}}_i = [o_1, o_2, \dots, o_K]^\top.$$

Multilayer Perceptron (MLP)

Sentido direto da MLP

- Notação matricial do modelo (as funções $\phi_1(\cdot)$ e $\phi_2(\cdot)$ são aplicadas para cada elemento do argumento):

$$\begin{aligned} \mathbf{z}_{1:N_H} &= \phi_1(\mathbf{W}\mathbf{x}_i), & z_0 &= 1, & \mathbf{W} &\in \mathbb{R}^{N_H \times (D+1)}, \\ \mathbf{o} &= \phi_2(\mathbf{M}\mathbf{z}), & & & \mathbf{M} &\in \mathbb{R}^{K \times (N_H+1)}. \end{aligned}$$

Multilayer Perceptron (MLP)

Sentido direto da MLP

- Notação matricial do modelo (as funções $\phi_1(\cdot)$ e $\phi_2(\cdot)$ são aplicadas para cada elemento do argumento):

$$\begin{aligned} \mathbf{z}_{1:N_H} &= \phi_1(\mathbf{W}\mathbf{x}_i), \quad z_0 = 1, & \mathbf{W} &\in \mathbb{R}^{N_H \times (D+1)}, \\ \mathbf{o} &= \phi_2(\mathbf{M}\mathbf{z}), & \mathbf{M} &\in \mathbb{R}^{K \times (N_H+1)}. \end{aligned}$$

- **Modelo de bases adaptativas:** A camada oculta realiza uma transformação/mapeamento na entrada.
- Mais neurônios ocultos resultam em um mapeamento complexo.

Multilayer Perceptron (MLP)

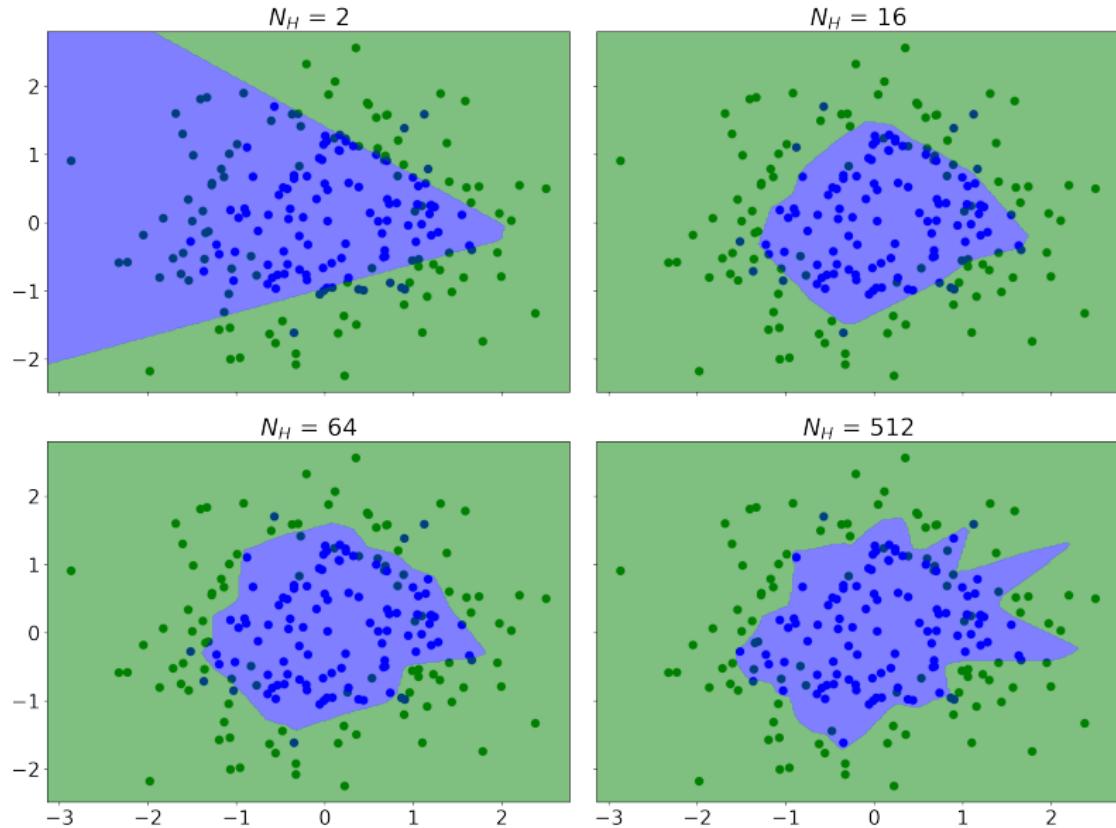
Sentido direto da MLP

- Notação matricial do modelo (as funções $\phi_1(\cdot)$ e $\phi_2(\cdot)$ são aplicadas para cada elemento do argumento):

$$\begin{aligned} \mathbf{z}_{1:N_H} &= \phi_1(\mathbf{W}\mathbf{x}_i), \quad z_0 = 1, & \mathbf{W} &\in \mathbb{R}^{N_H \times (D+1)}, \\ \mathbf{o} &= \phi_2(\mathbf{M}\mathbf{z}), & \mathbf{M} &\in \mathbb{R}^{K \times (N_H+1)}. \end{aligned}$$

- **Modelo de bases adaptativas:** A camada oculta realiza uma transformação/mapeamento na entrada.
- Mais neurônios ocultos resultam em um mapeamento complexo.
- O número de neurônios da camada de saída depende da tarefa:
 - **Classificação binária:** 1;
 - **Multiclasses com one-hot-encoding:** número de classes;
 - **Regressão:** dimensão da saída.

MLP com quantidade de neurônios ocultos variante



Redes Neurais Artificiais

- **Problema:** Como escolher as funções de ativação das camadas ocultas e de saída?

Redes Neurais Artificiais

- **Problema:** Como escolher as funções de ativação das camadas ocultas e de saída?
- Nas camadas ocultas a escolha não é óbvia, mas ela **deve ser não-linear**.
 - Caso contrário, a rede equivaleria a um modelo linear (sem camadas ocultas) e perderia sua capacidade não-linear.

Redes Neurais Artificiais

- **Problema:** Como escolher as funções de ativação das camadas ocultas e de saída?
- Nas camadas ocultas a escolha não é óbvia, mas ela **deve ser não-linear**.
 - Caso contrário, a rede equivaleria a um modelo linear (sem camadas ocultas) e perderia sua capacidade não-linear.
- Na saída, temos as seguintes opções para a função de ativação:
 - **Tarefa de regressão:** função identidade.
 - **Tarefa de classificação binária:** função sigmóide.
 - **Tarefa de classificação multclasse:** função softmax.

Redes Neurais Artificiais

Teorema da aproximação universal

- Uma rede MLP com 1 camada oculta não-linear é capaz de **representar qualquer função contínua**.
- **Importante:** Prova de existência somente.
 - O número de neurônios ocultos pode ser arbitrariamente grande.
 - Não indica como obter a melhor rede para um determinado conjunto de dados.

Redes Neurais Artificiais

Teorema da aproximação universal

- Uma rede MLP com 1 camada oculta não-linear é capaz de **representar qualquer função contínua**.
- **Importante:** Prova de existência somente.
 - O número de neurônios ocultos pode ser arbitrariamente grande.
 - Não indica como obter a melhor rede para um determinado conjunto de dados.
- **Questão:** Por que usar múltiplas camadas ocultas?

Redes Neurais Artificiais

Teorema da aproximação universal

- Uma rede MLP com 1 camada oculta não-linear é capaz de **representar qualquer função contínua**.
- **Importante:** Prova de existência somente.
 - O número de neurônios ocultos pode ser arbitrariamente grande.
 - Não indica como obter a melhor rede para um determinado conjunto de dados.
- **Questão:** Por que usar múltiplas camadas ocultas?
 - Funções complexas podem requerer exponencialmente mais neurônios ocultos, o que é mitigado com a adição de camadas.
 - Podem extrair padrões mais complexos dos dados automaticamente.
 - Organização hierárquica dos padrões aprendidos.

Redes Neurais Artificiais

Deep learning (redes de aprendizagem profunda)

- Modelos paramétricos com múltiplas camadas ocultas não-lineares com foco em aprendizagem automática de padrões complexos.
- Notação matricial da rede com H camadas ocultas, sendo a h -ésima camada com N_h neurônios ocultos:

$$\begin{aligned} \mathbf{z}_{1:N_1}^{(1)} &= \phi_1 \left(\mathbf{W}^{(1)} \mathbf{x}_i \right), & z_0^{(1)} &= 1, & \mathbf{W}^{(1)} &\in \mathbb{R}^{N_1 \times (D+1)}, \\ \mathbf{z}_{1:N_h}^{(h)} &= \phi_h \left(\mathbf{W}^{(h)} \mathbf{z}^{(h-1)} \right), & z_0^{(h)} &= 1, & \mathbf{W}^{(h)} &\in \mathbb{R}^{N_h \times (N_{h-1}+1)}, \\ \mathbf{o} &= \phi_o \left(\mathbf{M} \mathbf{z}^{(H)} \right), & \mathbf{M} &\in \mathbb{R}^{K \times (N_H+1)}. \end{aligned}$$

Redes Neurais Artificiais

Deep learning (redes de aprendizagem profunda)

- Modelos paramétricos com múltiplas camadas ocultas não-lineares com foco em aprendizagem automática de padrões complexos.
- Notação matricial da rede com H camadas ocultas, sendo a h -ésima camada com N_h neurônios ocultos:

$$\begin{aligned} \mathbf{z}_{1:N_1}^{(1)} &= \phi_1 \left(\mathbf{W}^{(1)} \mathbf{x}_i \right), & z_0^{(1)} &= 1, & \mathbf{W}^{(1)} &\in \mathbb{R}^{N_1 \times (D+1)}, \\ \mathbf{z}_{1:N_h}^{(h)} &= \phi_h \left(\mathbf{W}^{(h)} \mathbf{z}^{(h-1)} \right), & z_0^{(h)} &= 1, & \mathbf{W}^{(h)} &\in \mathbb{R}^{N_h \times (N_{h-1}+1)}, \\ \mathbf{o} &= \phi_o \left(\mathbf{M} \mathbf{z}^{(H)} \right), & \mathbf{M} &\in \mathbb{R}^{K \times (N_H+1)}. \end{aligned}$$

- **Observação:** Modelos de deep learning modernos podem apresentar camadas convolucionais, recorrentes, etc.

Agenda

- ① Redes Neurais Artificiais Lineares
- ② Redes Neurais Artificiais Não-Lineares
- ③ Algoritmo backpropagation
- ④ Técnicas para treinamento de redes neurais
- ⑤ Tópicos adicionais
- ⑥ Referências

Redes Neurais Artificiais

- **Problema:** Como treinar os pesos de uma rede MLP? Podemos usar o algoritmo SGD (*Stochastic Gradient Descent*)?

Redes Neurais Artificiais

- **Problema:** Como treinar os pesos de uma rede MLP? Podemos usar o algoritmo SGD (*Stochastic Gradient Descent*)?
- **Problema:** Não há uma saída desejada para os neurônios ocultos.

Redes Neurais Artificiais

- **Problema:** Como treinar os pesos de uma rede MLP? Podemos usar o algoritmo SGD (*Stochastic Gradient Descent*)?
- **Problema:** Não há uma saída desejada para os neurônios ocultos.
- **Ideia:** Escrever uma função custo, computar seus gradientes e caminhar na direção contrária.
 - **Algoritmo backpropagation** (popularizado em Rumelhart, Hinton, Williams (1986), com raízes na década de 1960).

Redes Neurais Artificiais

Algoritmo backpropagation

① Defina uma função custo \mathcal{J} :

- **Erro quadrático médio (regressão):**

$$\mathcal{J} = \frac{1}{2N} \sum_{i=1}^N \sum_{k=1}^K (y_{ik} - o_{ik})^2$$

- **Cross-entropy (classificação):**

$$\mathcal{J} = -\frac{1}{N} \sum_{i=1}^N \sum_{k=1}^K y_{ik} \log o_{ik}$$

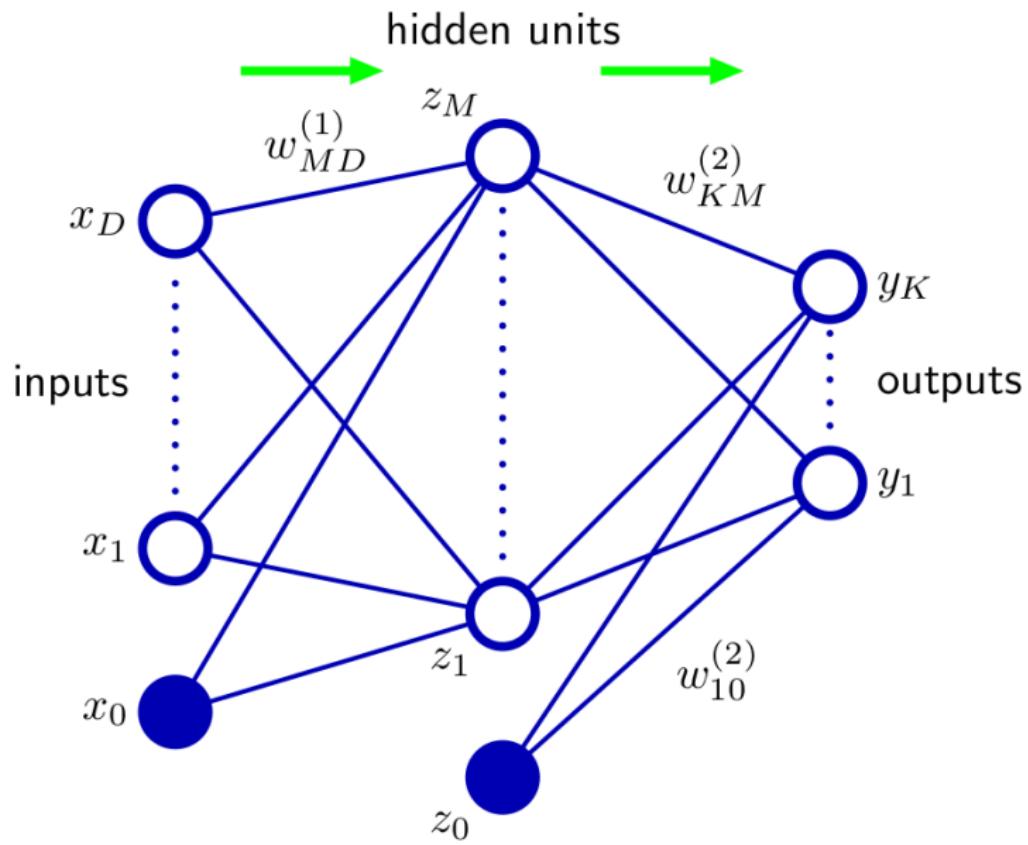
② Calcule os gradientes com relação a cada peso m_k e w_j .

③ Atualize os pesos via gradiente descendente (estocástico):

$$m_k(t+1) = m_k(t) - \alpha \frac{\partial \mathcal{J}}{\partial m_k(t)},$$

$$w_j(t+1) = w_j(t) - \alpha \frac{\partial \mathcal{J}}{\partial w_j(t)}.$$

Redes Neurais Artificiais



Algoritmo backpropagation

Sentido direto (*forward*) da MLP

- ① Saída do j -ésimo neurônio da camada oculta para a entrada \mathbf{x} :

$$z_j = \phi_1(u_j), \quad u_j = \mathbf{w}_j^\top \mathbf{x}, \quad 1 \leq j \leq N_H.$$

- ② Saída do k -ésimo neurônio da camada de saída:

$$o_k = \phi_2(r_k), \quad r_k = \mathbf{m}_k^\top \mathbf{z}, \quad 1 \leq k \leq K.$$

Algoritmo backpropagation

Sentido direto (*forward*) da MLP

- ① Saída do j -ésimo neurônio da camada oculta para a entrada \mathbf{x} :

$$z_j = \phi_1(u_j), \quad u_j = \mathbf{w}_j^\top \mathbf{x}, \quad 1 \leq j \leq N_H.$$

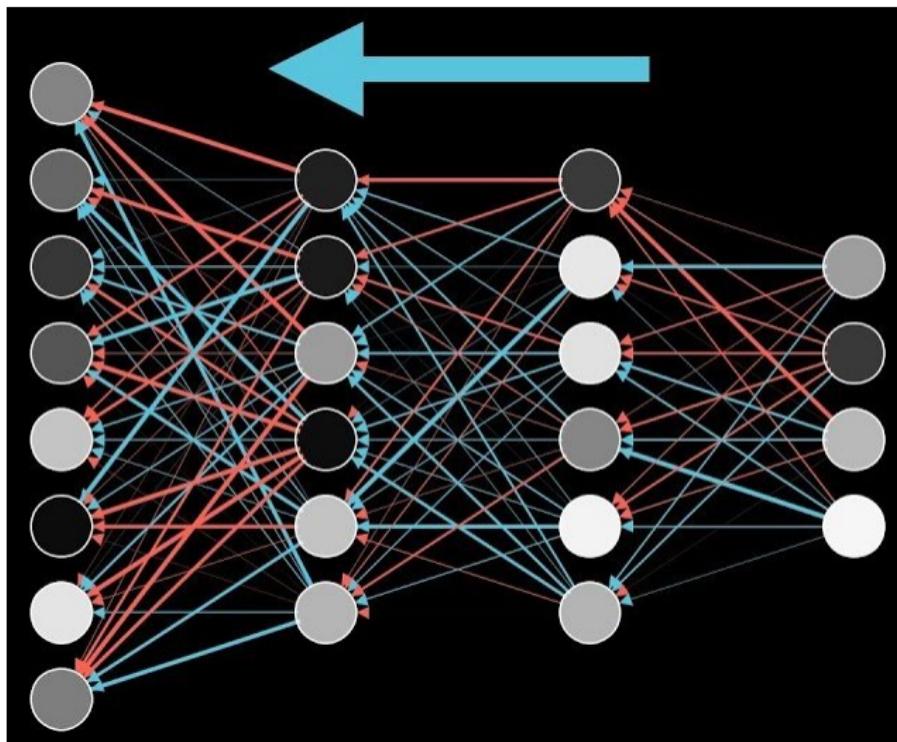
- ② Saída do k -ésimo neurônio da camada de saída:

$$o_k = \phi_2(r_k), \quad r_k = \mathbf{m}_k^\top \mathbf{z}, \quad 1 \leq k \leq K.$$

Sentido reverso (*backward*) da MLP

- ① Calcular gradientes da função custo na camada de saída;
- ② Retropropagar os erros para a camada oculta (regra da cadeia);
- ③ Atualizar todos os pesos via gradiente descendente (estocástico).

Redes Neurais Artificiais



Algoritmo backpropagation

$$\begin{aligned} z_j &= \phi_1(u_j), & u_j &= \mathbf{w}_j^\top \mathbf{x}, & 1 \leq j \leq N_H, \\ o_k &= \phi_2(r_k), & r_k &= \mathbf{m}_k^\top \mathbf{z}, & 1 \leq k \leq K. \end{aligned}$$

Algoritmo backpropagation

$$\begin{aligned}z_j &= \phi_1(u_j), & u_j &= \mathbf{w}_j^\top \mathbf{x}, & 1 \leq j \leq N_H, \\o_k &= \phi_2(r_k), & r_k &= \mathbf{m}_k^\top \mathbf{z}, & 1 \leq k \leq K.\end{aligned}$$

- Gradiente local do k -ésimo neurônio da camada de saída:

$$\frac{\partial \mathcal{J}}{\partial \mathbf{m}_k} = \frac{\partial \mathcal{J}}{\partial o_k} \frac{\partial o_k}{\partial r_k} \frac{\partial r_k}{\partial \mathbf{m}_k}, \quad 1 \leq k \leq K,$$

Algoritmo backpropagation

$$\begin{aligned} z_j &= \phi_1(u_j), & u_j &= \mathbf{w}_j^\top \mathbf{x}, & 1 \leq j \leq N_H, \\ o_k &= \phi_2(r_k), & r_k &= \mathbf{m}_k^\top \mathbf{z}, & 1 \leq k \leq K. \end{aligned}$$

- Gradiente local do k -ésimo neurônio da camada de saída:

$$\frac{\partial \mathcal{J}}{\partial \mathbf{m}_k} = \frac{\partial \mathcal{J}}{\partial o_k} \frac{\partial o_k}{\partial r_k} \frac{\partial r_k}{\partial \mathbf{m}_k}, \quad 1 \leq k \leq K,$$

$$\frac{\partial r_k}{\partial \mathbf{m}_k} = \mathbf{z},$$

$$\frac{\partial o_k}{\partial r_k} = \phi'_2(r_k), \quad (\text{derivada da função de ativação}),$$

$$\frac{\partial \mathcal{J}}{\partial o_k} \phi'_2(r_k) = \delta_k = -e_k \left\{ \begin{array}{l} \mathcal{J} \text{ erro quadrático e } \phi_2(r_k) = r_k, \\ \mathcal{J} \text{ entropia cruzada e } \phi_2(r_k) = \sigma(r_k) \text{ ou } \phi_2(r_k) = \text{softmax}(r_k). \end{array} \right.$$

Algoritmo backpropagation

$$\begin{aligned} z_j &= \phi_1(u_j), & u_j &= \mathbf{w}_j^\top \mathbf{x}, & 1 \leq j \leq N_H, \\ o_k &= \phi_2(r_k), & r_k &= \mathbf{m}_k^\top \mathbf{z}, & 1 \leq k \leq K. \end{aligned}$$

- Gradiente local do k -ésimo neurônio da camada de saída:

$$\frac{\partial \mathcal{J}}{\partial \mathbf{m}_k} = \frac{\partial \mathcal{J}}{\partial o_k} \frac{\partial o_k}{\partial r_k} \frac{\partial r_k}{\partial \mathbf{m}_k}, \quad 1 \leq k \leq K,$$

$$\frac{\partial r_k}{\partial \mathbf{m}_k} = \mathbf{z},$$

$$\frac{\partial o_k}{\partial r_k} = \phi'_2(r_k), \quad (\text{derivada da função de ativação}),$$

$$\frac{\partial \mathcal{J}}{\partial o_k} \phi'_2(r_k) = \delta_k = -e_k \begin{cases} \mathcal{J} \text{ erro quadrático e } \phi_2(r_k) = r_k, \\ \mathcal{J} \text{ entropia cruzada e } \phi_2(r_k) = \sigma(r_k) \text{ ou } \phi_2(r_k) = \text{softmax}(r_k). \end{cases}$$

- Atualização dos pesos da camada de saída:

$$\mathbf{m}_k(t+1) = \mathbf{m}_k(t) - \alpha \delta_k \mathbf{z}, \quad 1 \leq k \leq K.$$

Algoritmo backpropagation

$$\begin{aligned} z_j &= \phi_1(u_j), & u_j &= \mathbf{w}_j^\top \mathbf{x}, & 1 \leq j \leq N_H, \\ o_k &= \phi_2(r_k), & r_k &= \mathbf{m}_k^\top \mathbf{z}, & 1 \leq k \leq K. \end{aligned}$$

Algoritmo backpropagation

$$z_j = \phi_1(u_j), \quad u_j = \mathbf{w}_j^\top \mathbf{x}, \quad 1 \leq j \leq N_H,$$

$$o_k = \phi_2(r_k), \quad r_k = \mathbf{m}_k^\top \mathbf{z}, \quad 1 \leq k \leq K.$$

- Gradiente local do j -ésimo neurônio da camada oculta:

$$\frac{\partial \mathcal{J}}{\partial \mathbf{w}_j} = \sum_{k=1}^K \frac{\partial \mathcal{J}}{\partial o_k} \frac{\partial o_k}{\partial r_k} \frac{\partial r_k}{\partial z_j} \frac{\partial z_j}{\partial u_j} \frac{\partial u_j}{\partial \mathbf{w}_j} = \frac{\partial z_j}{\partial u_j} \frac{\partial u_j}{\partial \mathbf{w}_j} \sum_{k=1}^K \underbrace{\frac{\partial \mathcal{J}}{\partial o_k} \frac{\partial o_k}{\partial r_k}}_{\delta_k} \frac{\partial r_k}{\partial z_j}$$

Algoritmo backpropagation

$$z_j = \phi_1(u_j), \quad u_j = \mathbf{w}_j^\top \mathbf{x}, \quad 1 \leq j \leq N_H,$$

$$o_k = \phi_2(r_k), \quad r_k = \mathbf{m}_k^\top \mathbf{z}, \quad 1 \leq k \leq K.$$

- Gradiente local do j -ésimo neurônio da camada oculta:

$$\frac{\partial \mathcal{J}}{\partial \mathbf{w}_j} = \sum_{k=1}^K \frac{\partial \mathcal{J}}{\partial o_k} \frac{\partial o_k}{\partial r_k} \frac{\partial r_k}{\partial z_j} \frac{\partial z_j}{\partial u_j} \frac{\partial u_j}{\partial \mathbf{w}_j} = \frac{\partial z_j}{\partial u_j} \frac{\partial u_j}{\partial \mathbf{w}_j} \sum_{k=1}^K \underbrace{\frac{\partial \mathcal{J}}{\partial o_k} \frac{\partial o_k}{\partial r_k}}_{\delta_k} \frac{\partial r_k}{\partial z_j}$$

$$\frac{\partial r_k}{\partial z_j} = m_{kj},$$

$$\frac{\partial z_j}{\partial u_j} = \phi'_1(u_j), \quad (\text{derivada da função de ativação}),$$

$$\frac{\partial u_j}{\partial \mathbf{w}_j} = \mathbf{x}.$$

Algoritmo backpropagation

$$z_j = \phi_1(u_j), \quad u_j = \mathbf{w}_j^\top \mathbf{x}, \quad 1 \leq j \leq N_H,$$

$$o_k = \phi_2(r_k), \quad r_k = \mathbf{m}_k^\top \mathbf{z}, \quad 1 \leq k \leq K.$$

- Gradiente local do j -ésimo neurônio da camada oculta:

$$\frac{\partial \mathcal{J}}{\partial \mathbf{w}_j} = \sum_{k=1}^K \frac{\partial \mathcal{J}}{\partial o_k} \frac{\partial o_k}{\partial r_k} \frac{\partial r_k}{\partial z_j} \frac{\partial z_j}{\partial u_j} \frac{\partial u_j}{\partial \mathbf{w}_j} = \frac{\partial z_j}{\partial u_j} \frac{\partial u_j}{\partial \mathbf{w}_j} \sum_{k=1}^K \underbrace{\frac{\partial \mathcal{J}}{\partial o_k} \frac{\partial o_k}{\partial r_k}}_{\delta_k} \frac{\partial r_k}{\partial z_j}$$

$$\frac{\partial r_k}{\partial z_j} = m_{kj},$$

$$\frac{\partial z_j}{\partial u_j} = \phi'_1(u_j), \quad (\text{derivada da função de ativação}),$$

$$\frac{\partial u_j}{\partial \mathbf{w}_j} = \mathbf{x}.$$

- Atualização dos pesos da camada oculta:

$$\mathbf{w}_j(t+1) = \mathbf{w}_j(t) - \alpha \phi'_1(u_j) \mathbf{x} \sum_{k=1}^K \delta_k m_{kj}, \quad 1 \leq j \leq N_H.$$

Algoritmo backpropagation

- Sentido direto:

$$\begin{aligned}\mathbf{x} &= [1, x_1, x_2, \dots, x_D]^\top, \quad \mathbf{y} = [y_1, y_2, \dots, y_K]^\top, \\ z_0 &= 1, \quad z_j = \phi_1(u_j), \quad u_j = \mathbf{w}_j(t)^\top \mathbf{x}, \quad 1 \leq j \leq N_H, \\ o_k &= \phi_2(r_k), \quad r_k = \mathbf{m}_k(t)^\top \mathbf{z}, \quad 1 \leq k \leq K.\end{aligned}$$

- Sentido reverso:

$$\delta_k = \frac{\partial \mathcal{J}}{\partial o_k} \phi'_2(r_k), \quad 1 \leq k \leq K, \quad \overbrace{\text{e.g. } \delta_k = -(y_k - o_k)}^{\text{MSE ou entropia cruzada}},$$

$$\zeta_j = \phi'_1(u_j) \sum_{k=1}^K \delta_k m_{kj}(t), \quad 1 \leq j \leq N_H,$$

$$\mathbf{m}_k(t+1) = \mathbf{m}_k(t) - \alpha \delta_k \mathbf{z}, \quad 1 \leq k \leq K,$$

$$\mathbf{w}_j(t+1) = \mathbf{w}_j(t) - \alpha \zeta_j \mathbf{x}, \quad 1 \leq j \leq N_H.$$

Algoritmo backpropagation

- Sentido direto (forma matricial):

$$\mathbf{x} = [1, x_1, x_2, \dots, x_D]^\top, \quad \mathbf{y} = [y_1, y_2, \dots, y_K]^\top,$$

$$z_0 = 1, \quad \mathbf{z}_{1:N_H} = \phi_1(\mathbf{u}), \quad \mathbf{u} = \mathbf{W}\mathbf{x}, \quad \mathbf{W} \in \mathbb{R}^{N_H \times (D+1)},$$

$$\mathbf{o} = \phi_2(\mathbf{r}), \quad \mathbf{r} = \mathbf{M}\mathbf{z}, \quad \mathbf{M} \in \mathbb{R}^{K \times (N_H+1)}.$$

- Sentido reverso (forma matricial):

$$\boldsymbol{\delta} = \frac{\partial \mathcal{J}}{\partial \mathbf{o}} \odot \phi'_2(\mathbf{r}), \quad \boldsymbol{\delta} \in \mathbb{R}^K, \quad \overbrace{\text{e.g. } \boldsymbol{\delta} = -(\mathbf{y} - \mathbf{o})}^{\text{MSE ou entropia cruzada}},$$

$$\boldsymbol{\zeta} = \phi'_1(\mathbf{u}) \odot (\mathbf{M}_{1:N_H}^\top \boldsymbol{\delta}), \quad \boldsymbol{\zeta} \in \mathbb{R}^{N_H},$$

$$\mathbf{M}(t+1) = \mathbf{M}(t) - \alpha \boldsymbol{\delta} \mathbf{z}^\top,$$

$$\mathbf{W}(t+1) = \mathbf{W}(t) - \alpha \boldsymbol{\zeta} \mathbf{x}^\top.$$

- **Observação:** $\mathbf{a} \odot \mathbf{b}$ denota o produto de Hadamard.

Algoritmo backpropagation

- Sentido direto (forma matricial para N padrões):

$$\begin{aligned}\mathbf{X} &= [\mathbf{x}_1, \dots, \mathbf{x}_N]^\top \in \mathbb{R}^{N \times (D+1)}, & \mathbf{Y} &= [\mathbf{y}_1, \dots, \mathbf{y}_N]^\top \in \mathbb{R}^{N \times K}, \\ \mathbf{Z}_0 &= \mathbf{1}, & \mathbf{Z}_{1:N_H} &= \phi_1(\mathbf{U}), & \mathbf{U} &= \mathbf{X}\mathbf{W}^\top, & \mathbf{W} &\in \mathbb{R}^{N_H \times (D+1)}, \\ \mathbf{O} &= \phi_2(\mathbf{R}), & \mathbf{R} &= \mathbf{Z}\mathbf{M}^\top, & \mathbf{M} &\in \mathbb{R}^{K \times (N_H+1)}.\end{aligned}$$

- Sentido reverso (forma matricial para N padrões):

$$\begin{aligned}\boldsymbol{\delta} &= \frac{\partial \mathcal{J}}{\partial \mathbf{O}} \odot \phi'_2(\mathbf{R}), & \boldsymbol{\delta} &\in \mathbb{R}^{N \times K}, & \overbrace{\text{e.g. } \boldsymbol{\delta} = -(\mathbf{Y} - \mathbf{O})}^{\text{MSE ou entropia cruzada}}, \\ \boldsymbol{\zeta} &= \phi'_1(\mathbf{U}) \odot (\boldsymbol{\delta} \mathbf{M}_{1:N_H}), & \boldsymbol{\zeta} &\in \mathbb{R}^{N \times N_H}, \\ \mathbf{M}(t+1) &= \mathbf{M}(t) - \alpha \boldsymbol{\delta}^\top \mathbf{Z}, \\ \mathbf{W}(t+1) &= \mathbf{W}(t) - \alpha \boldsymbol{\zeta}^\top \mathbf{X}.\end{aligned}$$

- **Observação:** $\mathbf{a} \odot \mathbf{b}$ denota o produto de Hadamard.

Algoritmo backpropagation

- Sentido direto (H camadas ocultas):

$$\begin{aligned}\mathbf{X} &= [x_1, \dots, x_N]^\top \in \mathbb{R}^{N \times (D+1)}, \quad \mathbf{Y} = [y_1, \dots, y_N]^\top \in \mathbb{R}^{N \times K}, \\ \mathbf{Z}_0^{(1)} &= \mathbf{1}, \quad \mathbf{Z}_{1:N_1}^{(1)} = \phi_1(\mathbf{U}^{(1)}), \quad \mathbf{U}^{(1)} = \mathbf{X} \mathbf{W}^{(1)\top}, \quad \mathbf{W}^{(1)} \in \mathbb{R}^{N_1 \times (D+1)}, \\ \mathbf{Z}_0^{(h)} &= \mathbf{1}, \quad \mathbf{Z}_{1:N_h}^{(h)} = \phi_1(\mathbf{U}^{(h)}), \quad \mathbf{U}^{(h)} = \mathbf{Z}^{(h-1)} \mathbf{W}^{(h)\top}, \quad \mathbf{W}^{(h)} \in \mathbb{R}^{N_h \times (N_{h-1}+1)}, \\ \mathbf{O} &= \phi_2(\mathbf{R}), \quad \mathbf{R} = \mathbf{Z}^{(H)} \mathbf{M}^\top, \quad \mathbf{M} \in \mathbb{R}^{K \times (N_H+1)}.\end{aligned}$$

- Sentido reverso (H camadas ocultas):

$$\boldsymbol{\delta} = \frac{\partial \mathcal{J}}{\partial \mathbf{O}} \odot \phi'_2(\mathbf{R}), \quad \boldsymbol{\delta} \in \mathbb{R}^{N \times K}, \quad \overbrace{\text{e.g. } \boldsymbol{\delta} = -(\mathbf{Y} - \mathbf{O})}^{\text{MSE ou entropia cruzada}},$$

$$\boldsymbol{\zeta}^{(H)} = \phi'_1(\mathbf{U}^{(H)}) \odot (\boldsymbol{\delta} \mathbf{M}_{1:N_H}), \quad \boldsymbol{\zeta}^{(H)} \in \mathbb{R}^{N \times N_H},$$

$$\boldsymbol{\zeta}^{(h)} = \phi'_1(\mathbf{U}^{(h)}) \odot (\boldsymbol{\zeta}^{(h+1)} \mathbf{W}_{1:N_H}^{(h+1)}), \quad \boldsymbol{\zeta}^{(h)} \in \mathbb{R}^{N \times N_h}, \quad 1 \leq h < H,$$

$$\mathbf{M}(t+1) = \mathbf{M}(t) - \alpha \boldsymbol{\delta}^\top \mathbf{Z}^{(H)},$$

$$\mathbf{W}^{(h)}(t+1) = \mathbf{W}^{(h)}(t) - \alpha \boldsymbol{\zeta}^{(h)\top} \mathbf{Z}^{(h-1)}, \quad 1 < h \leq H,$$

$$\mathbf{W}^{(1)}(t+1) = \mathbf{W}^{(1)}(t) - \alpha \boldsymbol{\zeta}^{(1)\top} \mathbf{X}.$$

Agenda

- ① Redes Neurais Artificiais Lineares
- ② Redes Neurais Artificiais Não-Lineares
- ③ Algoritmo backpropagation
- ④ Técnicas para treinamento de redes neurais
- ⑤ Tópicos adicionais
- ⑥ Referências

Técnicas para treinamento de redes neurais

- **Problema:** Quando atualizar os pesos da rede?

Técnicas para treinamento de redes neurais

- **Problema:** Quando atualizar os pesos da rede?
 - **Gradiente descendente estocástico (SGD):** Após cada padrão de treinamento.
 - **Batch learning:** Após passar por todo o conjunto de treinamento.
 - **Mini-batch learning:** Após cada B padrões de treinamento.

Técnicas para treinamento de redes neurais

- **Problema:** Quando atualizar os pesos da rede?
 - **Gradiente descendente estocástico (SGD):** Após cada padrão de treinamento.
 - **Batch learning:** Após passar por todo o conjunto de treinamento.
 - **Mini-batch learning:** Após cada B padrões de treinamento.
- **Observações:**
 - *Batch learning* não é usualmente recomendado.
 - Custo computacional elevado.
 - Menor capacidade de generalização.
 - *Mini-batch learning* com B não muito grande (≤ 100) é o mais recomendado.
 - Gradientes mais estáveis quando comparado ao SGD ($B = 1$).
 - Custo computacional moderado.
 - Maior capacidade de generalização que batch learning ($B = N$).
 - O passo de aprendizagem deve ser dividido por B .

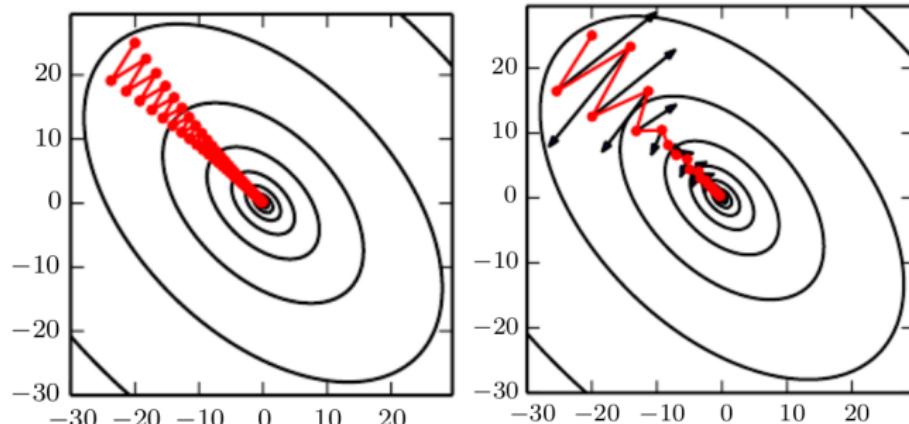
Técnicas para treinamento de redes neurais

Backpropagation com momentum

Parte da atualização anterior é repetida na iteração atual para estabilizar e acelerar o treinamento:

$$\Delta(t+1) = \mu\Delta(t) - \alpha \frac{\partial \mathcal{J}}{\partial \mathbf{w}_j(t)},$$

$$\mathbf{w}_j(t+1) = \mathbf{w}_j(t) + \Delta(t+1), \quad 0.5 \leq \mu < 1 \text{ (termo de momentum).}$$



Técnicas para treinamento de redes neurais

- **Problema:** Como controlar o passo de aprendizagem α ?

Técnicas para treinamento de redes neurais

- **Problema:** Como controlar o passo de aprendizagem α ?
- Valores iniciais maiores e reduzir ao longo das iterações/épocas:
 - Decaimento linear:

$$\alpha(t) = \alpha_0 \left(1 - \frac{t}{t_{max}}\right).$$

→ Decaimento exponencial:

$$\alpha(t) = \frac{\alpha_0}{1 + t}.$$

Técnicas para treinamento de redes neurais

- **Problema:** Como controlar o passo de aprendizagem α ?
- Adaptar α para cada parâmetro ao longo das iterações.
→ Adam (*adaptive moment estimation*):

$$\mathbf{g}(t+1) = \frac{\partial \mathcal{J}}{\partial \mathbf{w}_j(t)},$$

$$\mathbf{m}(t+1) = \beta_1 \mathbf{m}(t) + (1 - \beta_1) \mathbf{g}(t+1),$$

$$\mathbf{v}(t+1) = \beta_2 \mathbf{v}(t) + (1 - \beta_2) \mathbf{g}(t+1) \odot \mathbf{g}(t+1),$$

$$\hat{\mathbf{m}}(t+1) = \frac{\mathbf{m}(t+1)}{1 - \beta_1^{t+1}},$$

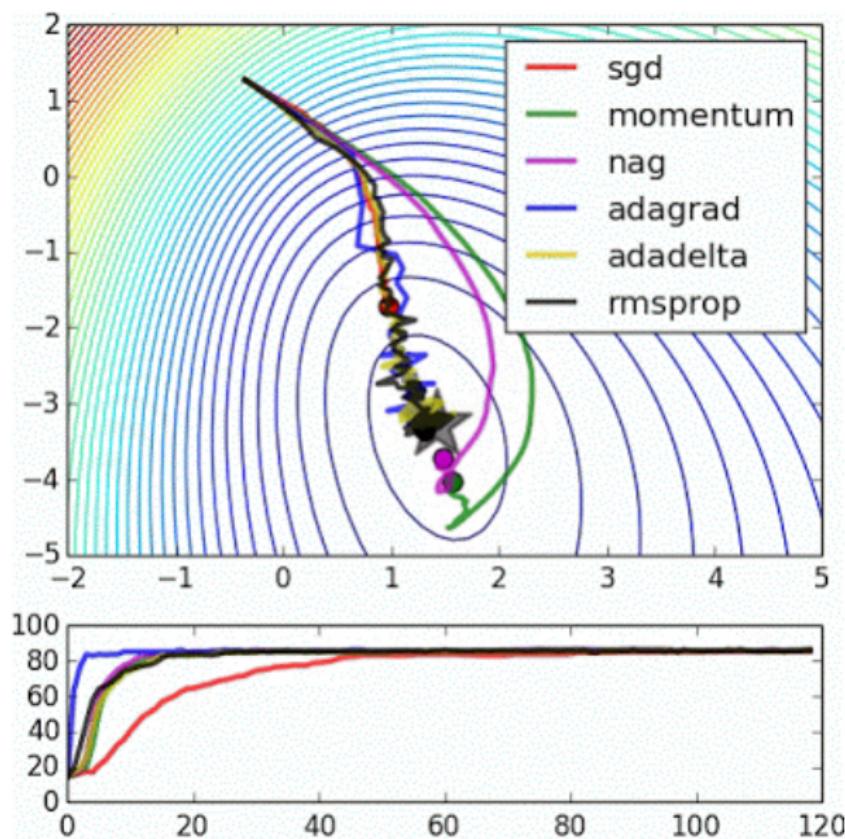
$$\hat{\mathbf{v}}(t+1) = \frac{\mathbf{v}(t+1)}{1 - \beta_2^{t+1}},$$

$$\mathbf{w}(t+1) = \mathbf{w}(t) - \alpha \hat{\mathbf{m}}(t+1) \oslash \left(\sqrt{\hat{\mathbf{v}}(t+1)} + \epsilon \right).$$

– Valores usuais: $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$.

→ Adagrad, Adadelta, AMSGrad, RMSprop, Nadam, Radam...

Técnicas para treinamento de redes neurais



Técnicas para treinamento de redes neurais

- **Problema:** Redes neurais são muito flexíveis e podem resultar em *overfitting*. Como evitá-lo?

Técnicas para treinamento de redes neurais

- **Problema:** Redes neurais são muito flexíveis e podem resultar em *overfitting*. Como evitá-lo?
- **Regularização L2 (weight decay):**
 - Função custo penaliza a norma quadrática dos pesos:

$$\mathcal{J}_{reg} = \mathcal{J} + \frac{\lambda}{2} \mathbf{w}^\top \mathbf{w}.$$

→ Atualização dos pesos regularizada:

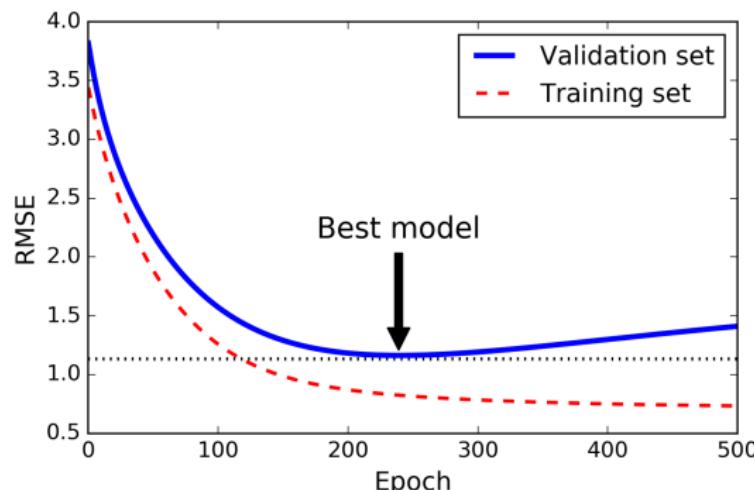
$$\mathbf{w}_j(t+1) = \mathbf{w}_j(t) - \alpha \left[\frac{\partial \mathcal{J}}{\partial \mathbf{w}_j(t)} + \lambda \mathbf{w}_j(t) \right], \quad \lambda \geq 0 \text{ (termo de regularização).}$$

Técnicas para treinamento de redes neurais

- **Problema:** Como perceber que o treinamento está resultando em *overfitting*?

Técnicas para treinamento de redes neurais

- **Problema:** Como perceber que o treinamento está resultando em *overfitting*?
- Monitorar o desempenho em um **conjunto de validação**.
- **Parada prematura (early stopping):**
 - Interrompe o treinamento quando o desempenho do modelo no conjunto de validação começar a piorar.
 - Forma simples e eficiente de regularização.



Técnicas para treinamento de redes neurais

- **Problema:** Como inicializar os pesos de uma rede MLP?

Técnicas para treinamento de redes neurais

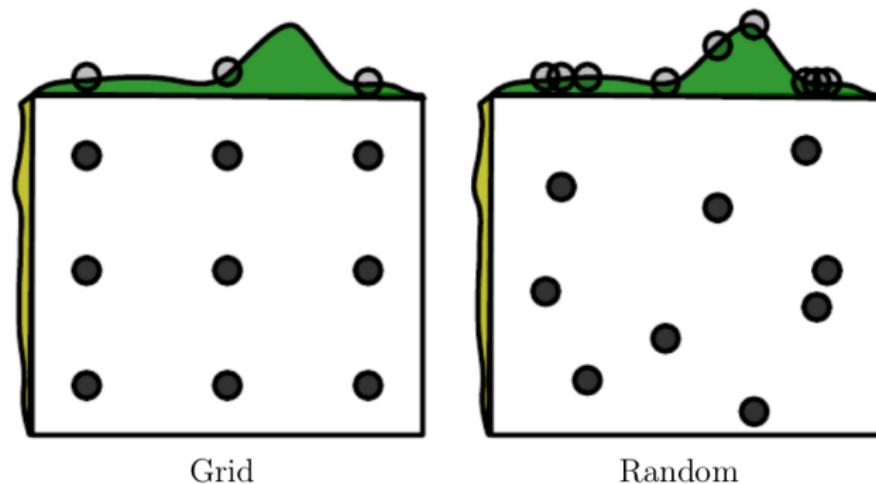
- **Problema:** Como inicializar os pesos de uma rede MLP?
- Inicialização aleatória baseada em heurísticas.
- Exemplos:
 - Pesos amostrados de $\sqrt{\frac{1}{N_{in}}}\mathcal{N}(0, 1)$, N_{in} é o número de entradas no neurônio.
 - Bias iniciado com zero.
 - Unidades ReLU costumam ser iniciadas com amostras de $\sqrt{\frac{2}{N_{in}}}\mathcal{N}(0, 1)$, e bias pequeno (~ 0.01).

Técnicas para treinamento de redes neurais

- Redes neurais possuem vários hiperparâmetros:
 - Número de camadas ocultas/neurônios, funções de ativação.
 - Passo de aprendizagem, termo de momentum/regularização.
 - Número de épocas, tamanho do minibatch, inicialização de parâmetros.
- **Problema:** Como selecionar bons hiperparâmetros?

Técnicas para treinamento de redes neurais

- Redes neurais possuem vários hiperparâmetros:
 - Número de camadas ocultas/neurônios, funções de ativação.
 - Passo de aprendizagem, termo de momentum/regularização.
 - Número de épocas, tamanho do minibatch, inicialização de parâmetros.
- **Problema:** Como selecionar bons hiperparâmetros?
- **Grid search × Random search.**



Técnicas para treinamento de redes neurais

Random search

- ① Separe 3 conjuntos de dados: **treinamento, validação e teste**;
- ② Amostre aleatoriamente uma lista de hiperparâmetros candidatos;
 - Exemplo: taxa de aprendizagem $\alpha = 10^{U(-5, -1)}$
- ③ Treine o modelo para os hiperparâmetros do primeiro candidato;
- ④ Verifique a qualidade do modelo obtido no conjunto de validação;
- ⑤ Repita os dois passos anteriores para os demais candidatos;
- ⑥ Escolha o hiperparâmetro com melhor avaliação no conjunto de validação;
- ⑦ Retreine o modelo com os conjuntos de treinamento e validação;
- ⑧ Verifique a generalização do modelo no conjunto de teste.

Técnicas para treinamento de redes neurais

Resumo de técnicas recomendadas (**não são gerais!**)

- Normalize os dados antes do treinamento.
- Use ativações ReLU (muitas camadas, classificação) ou tanh (poucas camadas, regressão).
- Use minibatches pequenos ($B = 32$).
- Embaralhe a ordem de apresentação dos dados de treinamento após cada época.
- Use termo de momentum ($\mu = 0.9$).
- Inicialize os pesos da rede cuidadosamente.
- Em caso de *overfitting*, use weight decay e/ou *early stopping*.
- Selecione hiperparâmetros via *random search*.

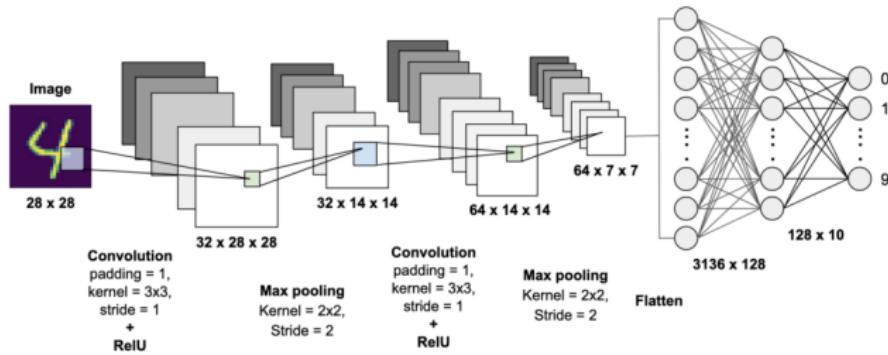
Agenda

- ① Redes Neurais Artificiais Lineares
- ② Redes Neurais Artificiais Não-Lineares
- ③ Algoritmo backpropagation
- ④ Técnicas para treinamento de redes neurais
- ⑤ Tópicos adicionais
- ⑥ Referências

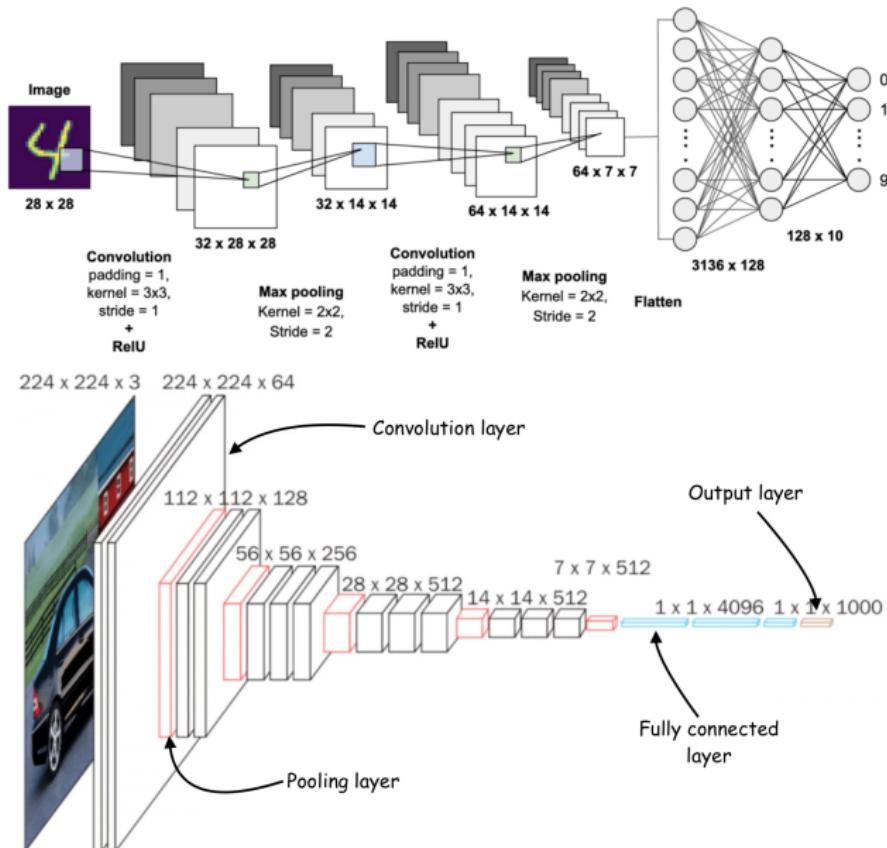
Tópicos adicionais

- Redes neurais convolucionais.
- Redes neurais recorrentes.
- Redes com módulos de atenção (Transformers).
- Redes neurais em grafos.
- Redes neurais não-supervisionadas.
- Aprendizagem Bayesiana de redes neurais.
- Modelos gerativos com redes neurais.
 - Variational Autoencoders (VAEs).
 - Generative Adversarial Networks (GANs).
 - Normalizing Flows (NFs).

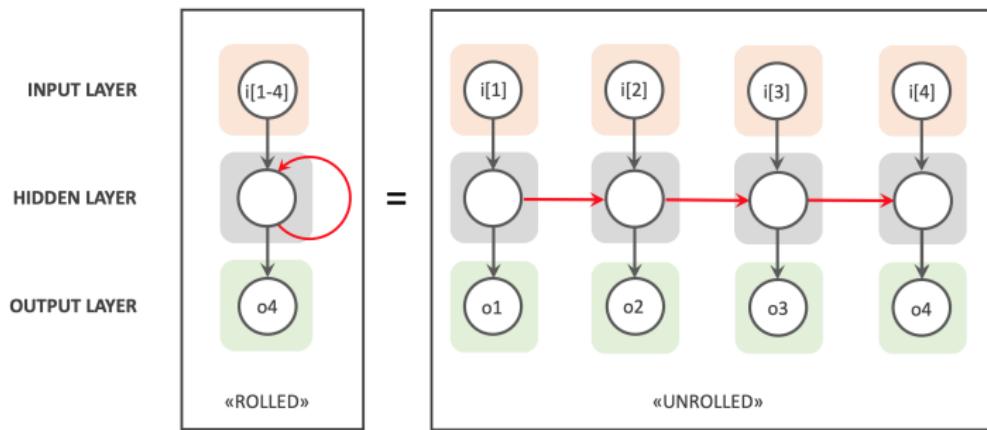
Convolutional Neural Networks (CNNs)



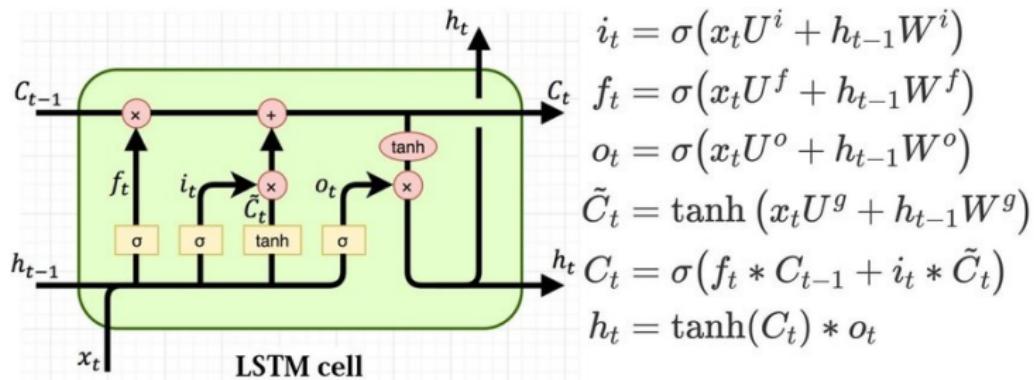
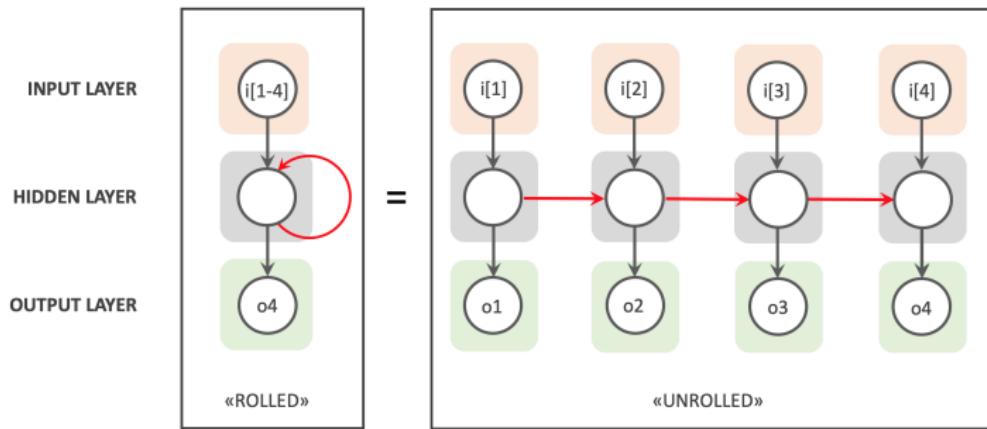
Convolutional Neural Networks (CNNs)



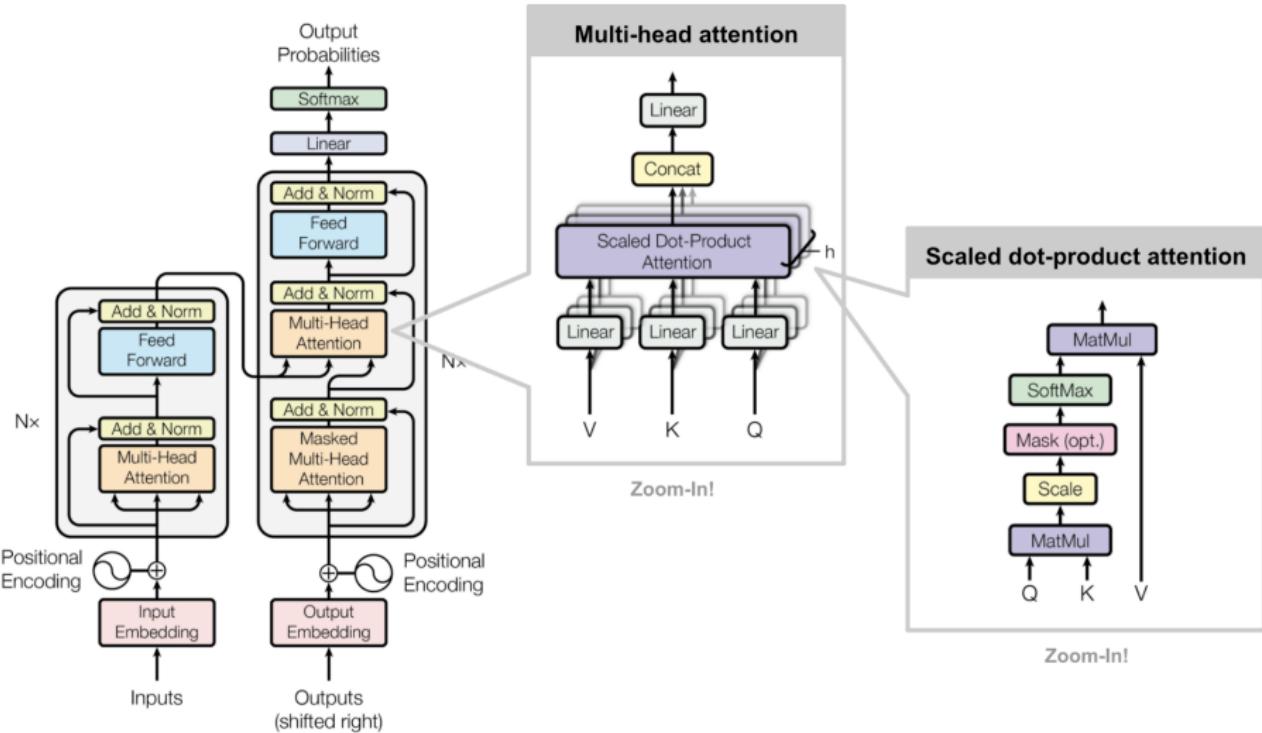
Recurrent Neural Networks (RNNs)



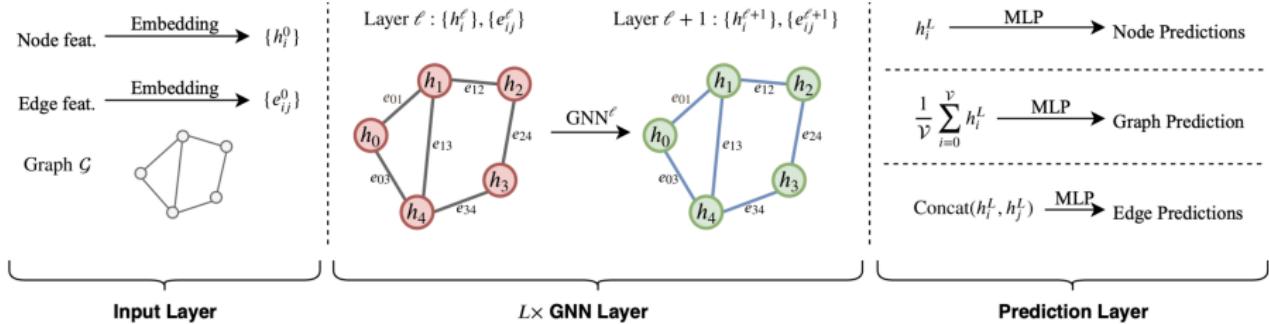
Recurrent Neural Networks (RNNs)



Redes com módulos de atenção (Transformers)

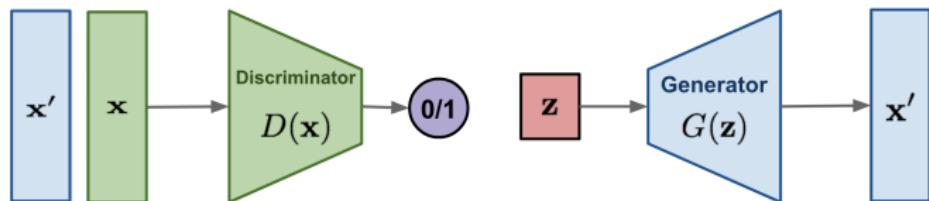


Graph Neural Networks (GNN)

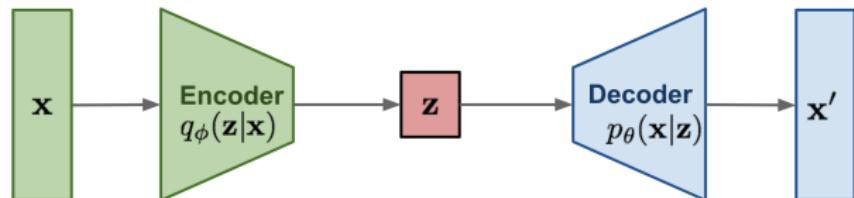


Deep Generative Models (DGMs)

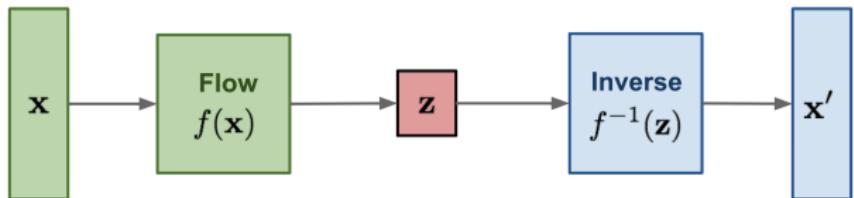
GAN: minimax the classification error loss.



VAE: maximize ELBO.



Flow-based generative models: minimize the negative log-likelihood



Agenda

- ① Redes Neurais Artificiais Lineares
- ② Redes Neurais Artificiais Não-Lineares
- ③ Algoritmo backpropagation
- ④ Técnicas para treinamento de redes neurais
- ⑤ Tópicos adicionais
- ⑥ Referências

Referências bibliográficas

- Cap. 5 - DEISENROTH, M. et al. **Mathematics for machine learning**. 2019.
- Caps. 8 e 16 - MURPHY, Kevin P. **Machine learning: a probabilistic perspective**, 2012.
- Caps. 10.2.5, 13 - MURPHY, Kevin P. **Probabilistic Machine Learning: An Introduction**, 2021.
- Caps. 4 e 5 - BISHOP, C. **Pattern recognition and machine learning**, 2006.
- Caps. 1, 3 e 4 - HAYKIN, S. **Neural networks and learning machines**, 2009.
- GOODFELLOW; BENGIO; COURVILLE. **Deep learning**, 2016.