Universidade Federal do Ceará

Departamento de Computação

Aprendizagem de Máquina

César Lincoln Cavalcante Mattos

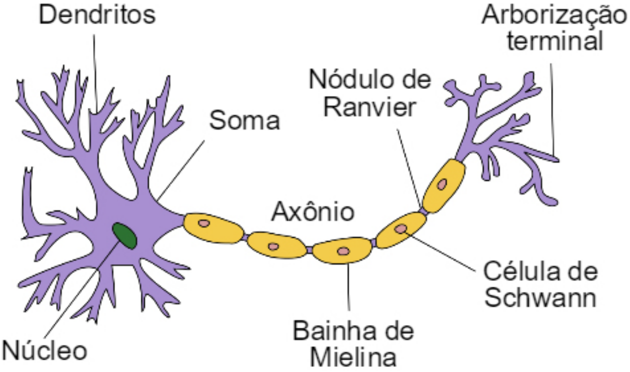
2020

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

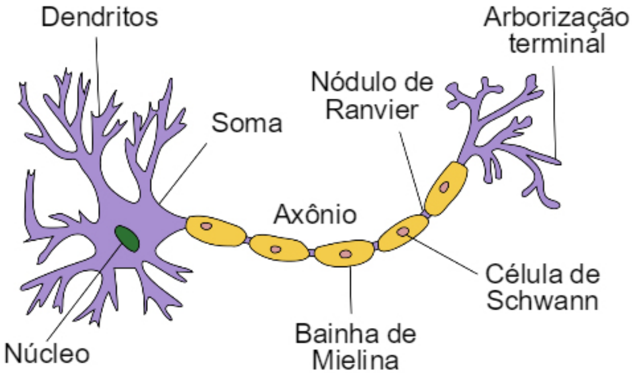
Redes Neurais Artificiais

- Modelos **"inspirados"** no neurônio biológico.



Redes Neurais Artificiais

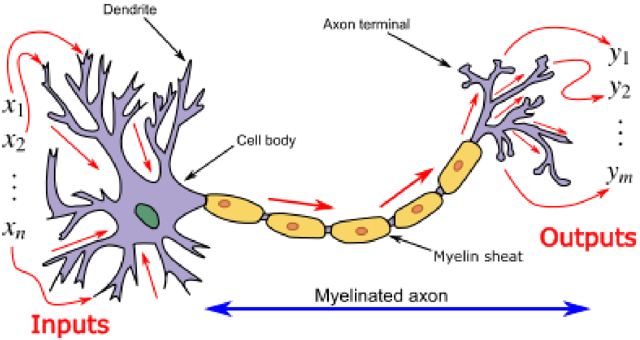
- Modelos **“inspirados”** no neurônio biológico.



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

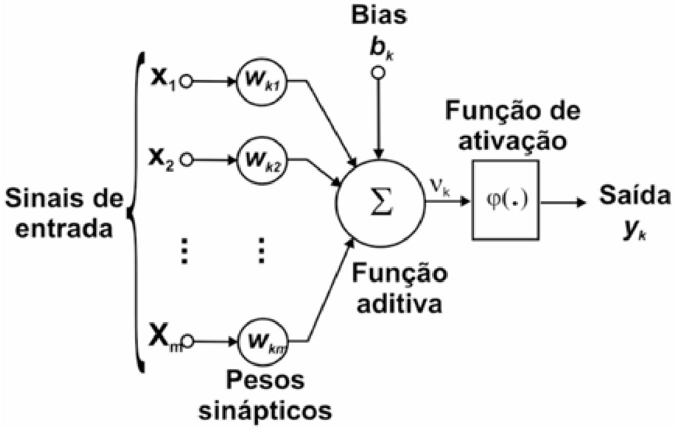
Redes Neurais Artificiais

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



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

Modelo neural de McCuloch-Pitts (1943)



Redes Neurais Artificiais

Perceptron de Rosenblatt (1957)

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

$$\hat{y}_i = \text{sign}(\boldsymbol{w}^\top \boldsymbol{x}_i),$$
$$\boldsymbol{x}_i = [1, x_1, x_2, \cdots, x_D]^\top,$$
$$\text{sign}(z) = \begin{cases} -1, & z < 0 \\ 1, & z \geq 0 \end{cases}$$

- **Função custo**:

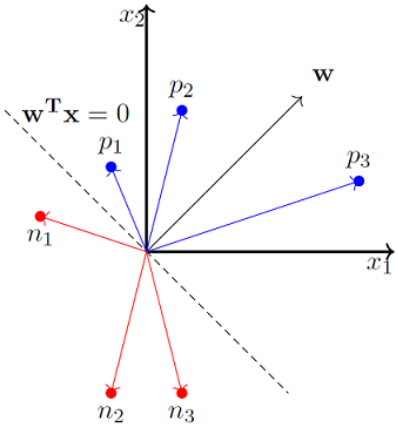
$$J(\boldsymbol{w}) = \sum_{i=1}^{N} \max(0, -y_i \hat{y}_i) = -\sum_{\forall i : y_i \neq \hat{y}_i} y_i \hat{y}_i.$$

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

Perceptron de Rosenblatt

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

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



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

Perceptron de Rosenblatt

Algoritmo de treinamento do Perceptron de Rosenblatt

1. Inicialize os pesos $\boldsymbol{w}(0)$;
2. Para cada par $(\boldsymbol{x}_i, y_i)|_{i=1}^{N}$, repita os passos abaixo:
   1. Calcule a saída do modelo: $\hat{y}_i = \text{sign}(\boldsymbol{w}(t)^\top \boldsymbol{x}_i)$;
   2. Calcule o erro obtido: $e_i = y_i - \hat{y}_i$;
   3. Atualize os pesos pela **regra de aprendizagem do Perceptron**:

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

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

Perceptron de Rosenblatt

Algoritmo de treinamento do Perceptron de Rosenblatt

1. Inicialize os pesos $\boldsymbol{w}(0)$;
2. Para cada par $(\boldsymbol{x}_i, y_i)|_{i=1}^{N}$, repita os passos abaixo:
   1. Calcule a saída do modelo: $\hat{y}_i = \text{sign}(\boldsymbol{w}(t)^\top \boldsymbol{x}_i)$;
   2. Calcule o erro obtido: $e_i = y_i - \hat{y}_i$;
   3. Atualize os pesos pela **regra de aprendizagem do Perceptron**:

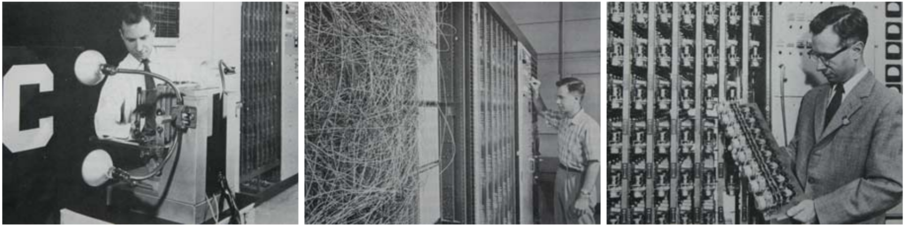
$$\boldsymbol{w}(t+1) = \boldsymbol{w}(t) + \alpha e_i \boldsymbol{x}_i,$$

   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 $\text{sign}(\cdot)$:

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

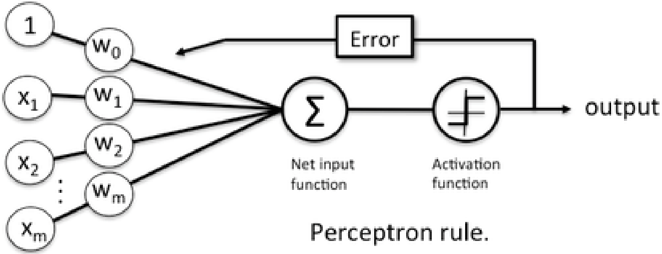
- **Função custo**:

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

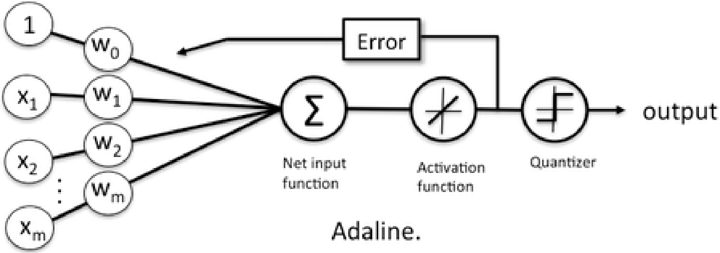
- Atualize os pesos pelo algoritmo LMS:

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

Perceptron × ADALINE



Perceptron rule.



Adaline.

Redes Neurais Artificiais

- Notação para K saídas: $\hat{\boldsymbol{y}}_i = \text{sign}(\boldsymbol{W}\boldsymbol{x}_i)$, $\boldsymbol{W} \in \mathbb{R}^{K\times(D+1)}$.

Multiple Perceptron

- **Função custo**:

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

MADALINE (*Multiple ADALINE*)

- **Função custo**:

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

- Atualização dos pesos:

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

Redes Neurais Artificiais

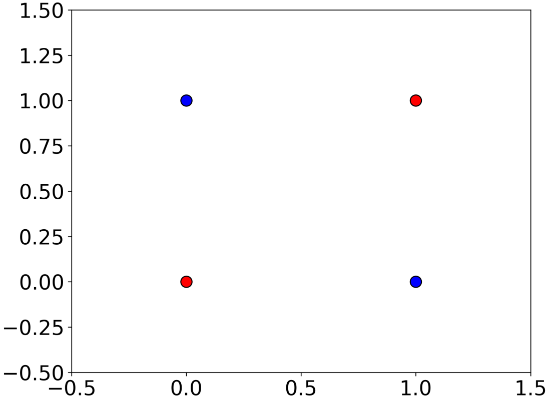
O perceptron mostrou-se digno de estudo, apesar (e até por causa!) de suas severas limitações. Ele possui muitos recursos que atraem a atenção: sua linearidade; seu teorema de aprendizado intrigante; sua clara simplicidade paradigmática como uma espécie de computação paralela. **Não há razão para supor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

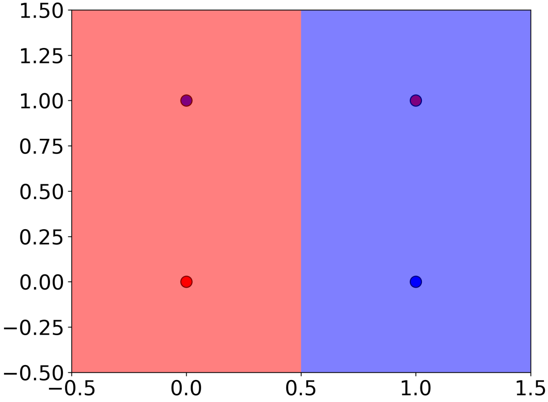
Redes Neurais Artificiais

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



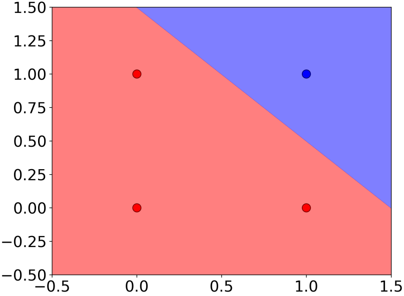
Redes Neurais Artificiais

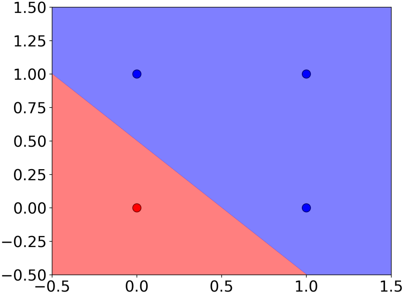
- **Problema**: O Perceptron é capaz de resolver o problema da porta lógica XOR? **Não**!



Redes Neurais Artificiais

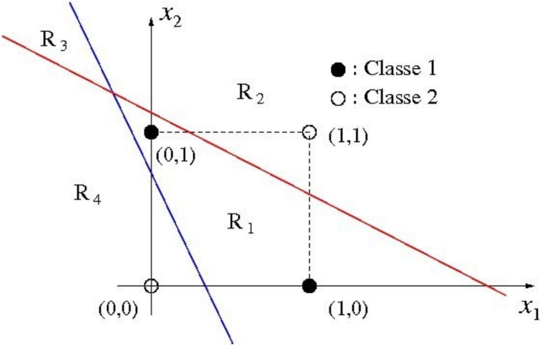
- **Ideia**: Treinar 2 perceptrons, um com a porta AND e outro com a porta OR.





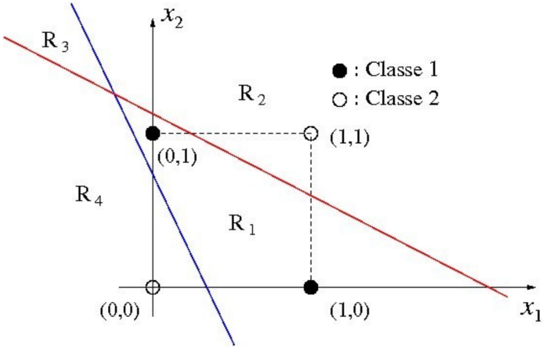
Redes Neurais Artificiais

- Reunimos as fronteiras de decisão de ambos os perceptrons:



Redes Neurais Artificiais

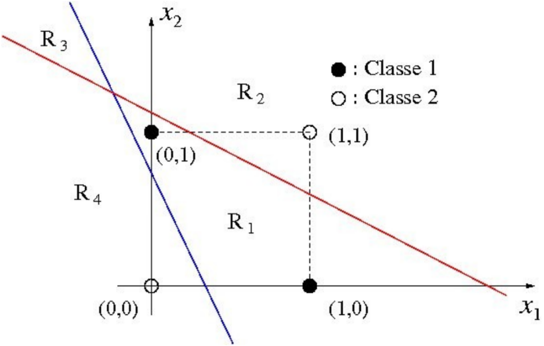
- Reunimos as fronteiras de decisão de ambos os perceptrons:



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

Redes Neurais Artificiais

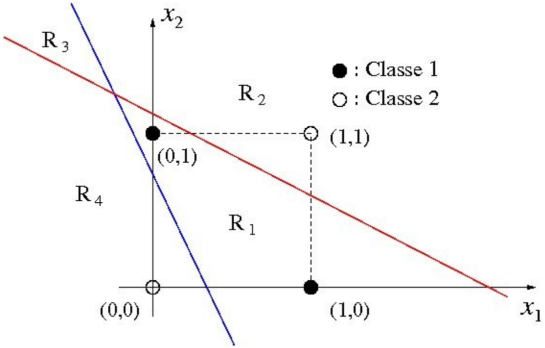
- Reunimos as fronteiras de decisão de ambos os perceptrons:



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

Redes Neurais Artificiais

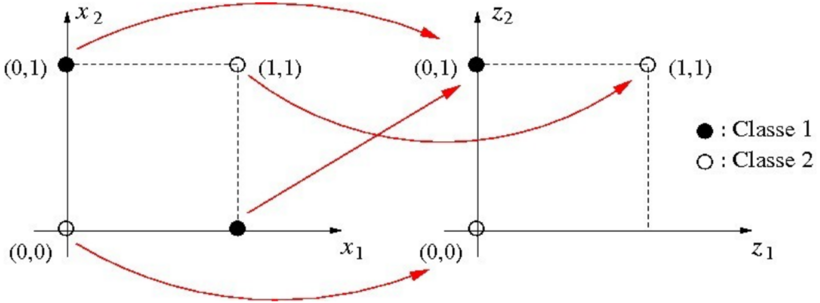
- Reunimos as fronteiras de decisão de ambos os perceptrons:



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

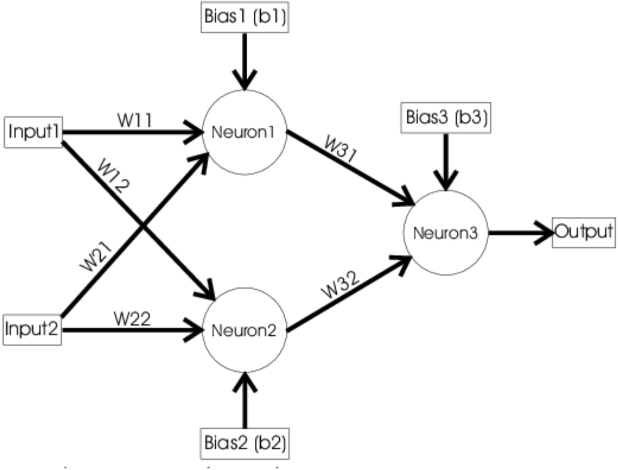
Redes Neurais Artificiais

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



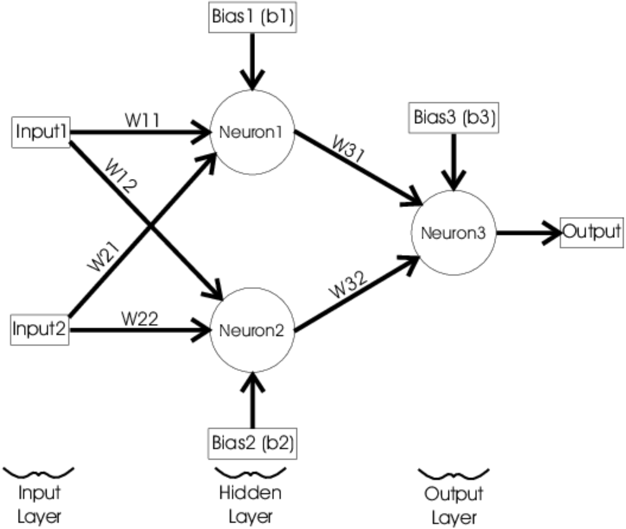
Redes Neurais Artificiais

- 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.

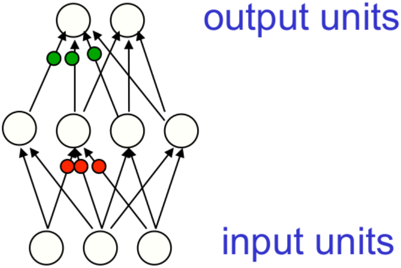


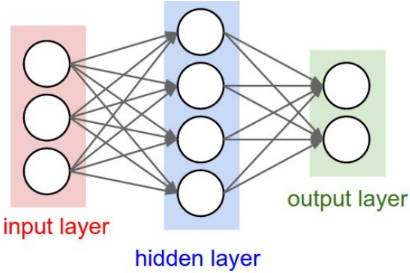
Redes Neurais Artificiais

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: $\phi(\boldsymbol{w}^\top \boldsymbol{x}_i)$

→ **Sigmóide**:

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

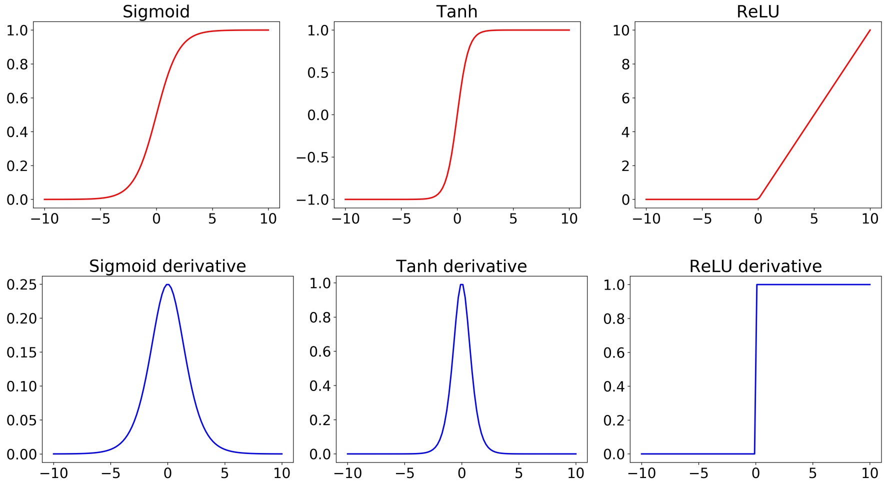
→ **Tangente hiperbólica**:

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

→ **Rectified Linear Unit (ReLU)**:

$$\mathsf{relu}(z) = \max(0, z), \quad \frac{\mathrm{d}\mathsf{relu}(z)}{\mathrm{d}z} = \begin{cases} 0, & z < 0 \\ 1, & z \geq 0 \end{cases}$$

Funções de ativação mais comuns



Multilayer Perceptron (MLP)

Perceptron Multicamadas

- Considere os dados $(\boldsymbol{x}_i, \boldsymbol{y}_i)|_{i=1}^{N}$, $\boldsymbol{x}_i \in \mathbb{R}^D$, $\boldsymbol{y}_i \in \mathbb{R}^K$. Além disso:
  - $\rightarrow$ seja uma rede MLP de 1 camada oculta com N_H neurônios;
  - $\rightarrow$ sejam K neurônios de saída;
  - $\rightarrow$ 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 $\boldsymbol{x}_i$:

$$
\begin{aligned}
z_j &= \phi_1(\boldsymbol{w}_j^\top \boldsymbol{x}_i), \quad 1 \leq j \leq N_H, \\
\boldsymbol{x_i} &= [1, x_{i1}, x_{i2}, \cdots, x_{iD}]^\top, \\
\boldsymbol{w}_j &\in \mathbb{R}^{D+1}.
\end{aligned}
$$

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

$$
\begin{aligned}
o_k &= \phi_2(\boldsymbol{m}_k^\top \boldsymbol{z}), \quad 1 \leq k \leq K, \\
\boldsymbol{z} &= [1, z_1, z_2, \cdots, z_{N_H}]^\top, \\
\boldsymbol{m}_k &\in \mathbb{R}^{N_H+1}.
\end{aligned}
$$

- Saída da rede:

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

Multilayer Perceptron (MLP)

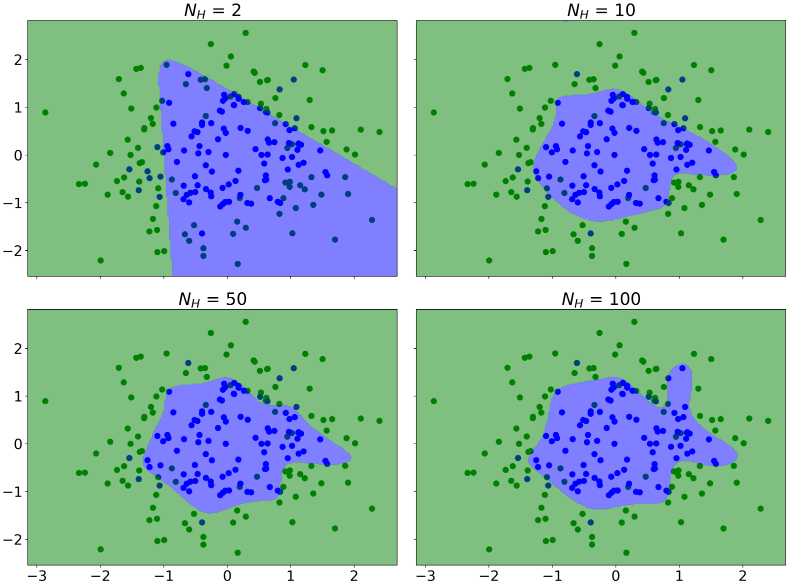
Sentido direto da MLP

- Notação matricial do modelo:

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

- **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:
  - $\rightarrow$ **Classificação binária**: 1;
  - $\rightarrow$ **Multiclasse com one-hot-encoding**: número de classes;
  - $\rightarrow$ **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**.
  - $\rightarrow$ 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**.
  - $\rightarrow$ 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 multiclasse**: 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.
  - $\rightarrow$ O número de neurônios ocultos pode ser arbitrariamente grande.
  - $\rightarrow$ 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.
  - $\rightarrow$ O número de neurônios ocultos pode ser arbitrariamente grande.
  - $\rightarrow$ 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.
  - $\rightarrow$ O número de neurônios ocultos pode ser arbitrariamente grande.
  - $\rightarrow$ Não indica como obter a melhor rede para um determinado conjunto de dados.

- **Questão:** Por que usar múltiplas camadas ocultas?
  - $\rightarrow$ Funções complexas podem requerer exponencialmente mais neurônios ocultos, o que é mitigado com a adição de camadas.
  - $\rightarrow$ Podem extrair padrões mais complexos dos dados automaticamente.
  - $\rightarrow$ 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:

$$
\begin{aligned}
\boldsymbol{z}^{(1)}_{1:N_1} &= \phi_1\left(\boldsymbol{W}^{(1)}\boldsymbol{x}_i\right), & z^{(1)}_0 &= 1, & \boldsymbol{W}^{(1)} &\in \mathbb{R}^{N_1\times(D+1)},\\
\boldsymbol{z}^{(h)}_{1:N_h} &= \phi_h\left(\boldsymbol{W}^{(h)}\boldsymbol{z}^{(h-1)}\right), & z^{(h)}_0 &= 1, & \boldsymbol{W}^{(h)} &\in \mathbb{R}^{N_h\times(N_{h-1}+1)},\\
\boldsymbol{o} &= \phi_o\left(\boldsymbol{M}\boldsymbol{z}^{(H)}\right), & & & \boldsymbol{M} &\in \mathbb{R}^{K\times(N_H+1)}.
\end{aligned}
$$

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:

$$
\begin{aligned}
\boldsymbol{z}^{(1)}_{1:N_1} &= \phi_1\left(\boldsymbol{W}^{(1)}\boldsymbol{x}_i\right), & z^{(1)}_0 &= 1, \quad \boldsymbol{W}^{(1)} \in \mathbb{R}^{N_1\times(D+1)},\\
\boldsymbol{z}^{(h)}_{1:N_h} &= \phi_h\left(\boldsymbol{W}^{(h)}\boldsymbol{z}^{(h-1)}\right), & z^{(h)}_0 &= 1, \quad \boldsymbol{W}^{(h)} \in \mathbb{R}^{N_h\times(N_{h-1}+1)},\\
\boldsymbol{o} &= \phi_o\left(\boldsymbol{M}\boldsymbol{z}^{(H)}\right), & & \quad \boldsymbol{M} \in \mathbb{R}^{K\times(N_H+1)}.
\end{aligned}
$$

- **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 LMS?

Redes Neurais Artificiais

- **Problema**: Como treinar os pesos de uma rede MLP? Podemos usar o algoritmo LMS?
- **Problema**: O algoritmo LMS não pode ser usado diretamente, pois não há uma saída desejada para os neurônios ocultos.

Redes Neurais Artificiais

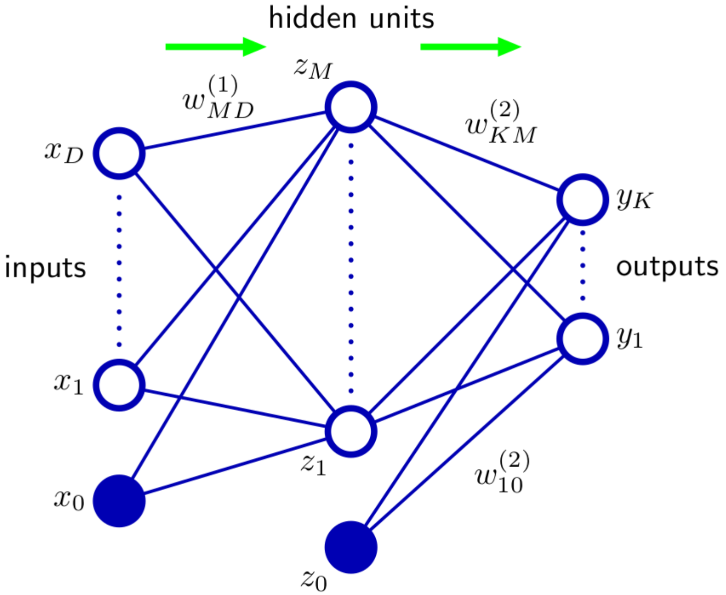
- **Problema**: Como treinar os pesos de uma rede MLP? Podemos usar o algoritmo LMS?
- **Problema**: O algoritmo LMS não pode ser usado diretamente, pois não há uma saída desejada para os neurônios ocultos.
- **Ideia**: Escrever uma função custo, computar seus gradientes e caminhar na direção contrária.
  - $\rightarrow$ **Algoritmo backpropation** (Rumelhart, Hinton, Williams, 1986).

Redes Neurais Artificiais

Algoritmo backpropagation

1. Defina uma função custo $\mathcal{J}$:
   - **Erro quadrático médio (regressão)**:
   $$\mathcal{J} = \frac{1}{2N}\sum_{i=1}^{N}\sum_{k=1}^{K}(y_{ik} - o_{ik})^2$$
   - **Crossentropy (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 $\boldsymbol{m}_k$ e $\boldsymbol{w}_j$.
3. Atualize os pesos via gradiente descendente (estocástico):
$$\boldsymbol{m}_k(t+1) = \boldsymbol{m}_k(t) - \alpha\frac{\partial\mathcal{J}}{\partial\boldsymbol{m}_k(t)},$$
$$\boldsymbol{w}_j(t+1) = \boldsymbol{w}_j(t) - \alpha\frac{\partial\mathcal{J}}{\partial\boldsymbol{w}_j(t)}.$$

Redes Neurais Artificiais



Algoritmo backpropagation

Sentido direto (*forward*) da MLP

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

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

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

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

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).

Algoritmo backpropagation

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

Algoritmo backpropagation

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

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

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

Algoritmo backpropagation

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

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

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

$$\frac{\partial \mathcal{J}}{\partial o_k} = -e_k, \quad (\mathcal{J} \rightarrow \text{erro quadrático ou crossentropia}),$$
$$\frac{\partial o_k}{\partial r_k} = \phi_2'(r_k), \quad (\text{derivada da função de ativação}),$$
$$\frac{\partial r_k}{\partial \boldsymbol{m}_k} = \boldsymbol{z}.$$

- Atualização dos pesos da camada de saída:

$$\boldsymbol{m}_k(t+1) = \boldsymbol{m}_k(t) + \alpha e_k \phi_2'(r_k)\boldsymbol{z}, \quad 1 \leq k \leq K.$$

Algoritmo backpropagation

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

Algoritmo backpropagation

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

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

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

Algoritmo backpropagation

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

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

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

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

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

$$\frac{\partial z_j}{\partial u_j} = \phi_1'(r_k), \quad \text{(derivada da função de ativação)},$$

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

- Atualização dos pesos da camada oculta:

$$\boldsymbol{w}_j(t+1) = \boldsymbol{w}_j(t) + \alpha \phi_1'(u_j) \boldsymbol{x} \sum_{k=1}^{K} e_k \phi_2'(r_k) m_{kj}, \quad 1 \leq j \leq N_H.$$

Algoritmo backpropagation

- Sentido direto:

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

- Sentido reverso:

$$
\begin{aligned}
e_k &= y_k - o_k, \quad 1 \leq k \leq K, \\
\delta_k &= e_k \phi_2'(r_k) \quad 1 \leq k \leq K, \\
\zeta_j &= \phi_1'(u_j) \sum_{k=1}^{K} \delta_k m_{kj}(t) \quad 1 \leq j \leq N_H, \\
\boldsymbol{m}_k(t+1) &= \boldsymbol{m}_k(t) + \alpha \delta_k \boldsymbol{z}, \quad 1 \leq k \leq K, \\
\boldsymbol{w}_j(t+1) &= \boldsymbol{w}_j(t) + \alpha \zeta_j \boldsymbol{x}, \quad 1 \leq j \leq N_H.
\end{aligned}
$$

Algoritmo backpropagation

- Sentido direto (forma matricial):

$$
\begin{aligned}
\boldsymbol{x} &= [1, x_1, x_2, \cdots, x_D]^\top, \\
z_0 = 1, \quad \boldsymbol{z}_{1:N_H} &= \phi_1(\boldsymbol{u}), \quad \boldsymbol{u} = \boldsymbol{W}\boldsymbol{x}, \quad \boldsymbol{W} \in \mathbb{R}^{N_H \times (D+1)}, \\
\boldsymbol{o} &= \phi_2(\boldsymbol{r}), \quad \boldsymbol{r} = \boldsymbol{M}\boldsymbol{z}, \quad \boldsymbol{M} \in \mathbb{R}^{K \times (N_H+1)}.
\end{aligned}
$$

- Sentido reverso (forma matricial):

$$
\begin{aligned}
\boldsymbol{e} &= \boldsymbol{y} - \boldsymbol{o}, \quad \boldsymbol{e} \in \mathbb{R}^K \\
\boldsymbol{\delta} &= \boldsymbol{e} \odot \phi_2'(\boldsymbol{r}), \quad \boldsymbol{\delta} \in \mathbb{R}^K, \\
\boldsymbol{\zeta} &= \phi_1'(\boldsymbol{u}) \odot \boldsymbol{M}_{1:N_H}^\top \boldsymbol{\delta}, \quad \boldsymbol{\zeta} \in \mathbb{R}^{N_H}, \\
\boldsymbol{M}(t+1) &= \boldsymbol{M}(t) + \alpha \boldsymbol{\delta} \boldsymbol{z}^\top, \\
\boldsymbol{W}(t+1) &= \boldsymbol{W}(t) + \alpha \boldsymbol{\zeta} \boldsymbol{x}^\top.
\end{aligned}
$$

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

Algoritmo backpropagation

- Sentido direto (forma matricial para H camadas ocultas):

$$\boldsymbol{x} = [1, x_1, x_2, \cdots, x_D]^\top,$$

$$z_0^{(1)} = 1, \quad \boldsymbol{z}_{1:N_1}^{(1)} = \phi_1(\boldsymbol{u}^{(1)}), \quad \boldsymbol{u}^{(1)} = \boldsymbol{W}^{(1)}\boldsymbol{x}, \quad \boldsymbol{W}^{(1)} \in \mathbb{R}^{N_1 \times (D+1)},$$

$$z_0^{(h)} = 1, \quad \boldsymbol{z}_{1:N_H}^{(h)} = \phi_1(\boldsymbol{u}^{(h)}), \quad \boldsymbol{u}^{(h)} = \boldsymbol{W}^{(h)}\boldsymbol{z}^{(h-1)}, \quad \boldsymbol{W}^{(h)} \in \mathbb{R}^{N_h \times (N_{h-1}+1)},$$

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

- Sentido reverso (forma matricial para H camadas ocultas):

$$\boldsymbol{e} = \boldsymbol{y} - \boldsymbol{o}, \quad \boldsymbol{e} \in \mathbb{R}^K$$

$$\boldsymbol{\delta} = \boldsymbol{e} \odot \phi_2'(\boldsymbol{r}), \quad \boldsymbol{\delta} \in \mathbb{R}^K,$$

$$\boldsymbol{\zeta}^{(H)} = \phi_1'(\boldsymbol{u}^{(H)}) \odot \boldsymbol{M}_{1:N_H}^\top \boldsymbol{\delta}, \quad \boldsymbol{\zeta} \in \mathbb{R}^{N_H},$$

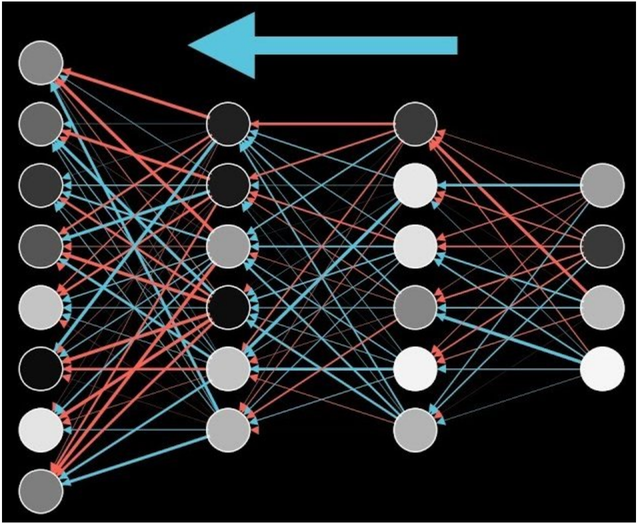
$$\boldsymbol{\zeta}^{(h)} = \phi_1'(\boldsymbol{u}^{(h)}) \odot \boldsymbol{W}_{1:N_H}^{(h+1)\top} \boldsymbol{\zeta}^{(h+1)}, \quad \boldsymbol{\zeta} \in \mathbb{R}^{N_h},$$

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

$$\boldsymbol{W}^{(h)}(t+1) = \boldsymbol{W}^{(h)}(t) + \alpha\boldsymbol{\zeta}^{(h)}\boldsymbol{z}^{(h-1)\top},$$

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

Redes Neurais Artificiais



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?
  - $\rightarrow$ **Gradiente descendente estocástico (SGD)**: Após cada padrão de treinamento.
  - $\rightarrow$ **Batch learning**: Após passar por todo o conjunto de treinamento.
  - $\rightarrow$ **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.

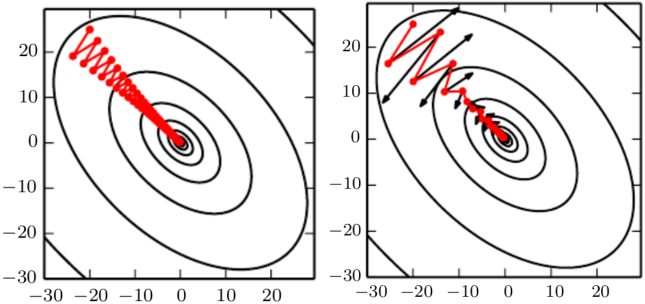
Técnicas para treinamento de redes neurais

Backpropagation com momentum

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

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

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



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:
  - $\rightarrow$ Decaimento linear:

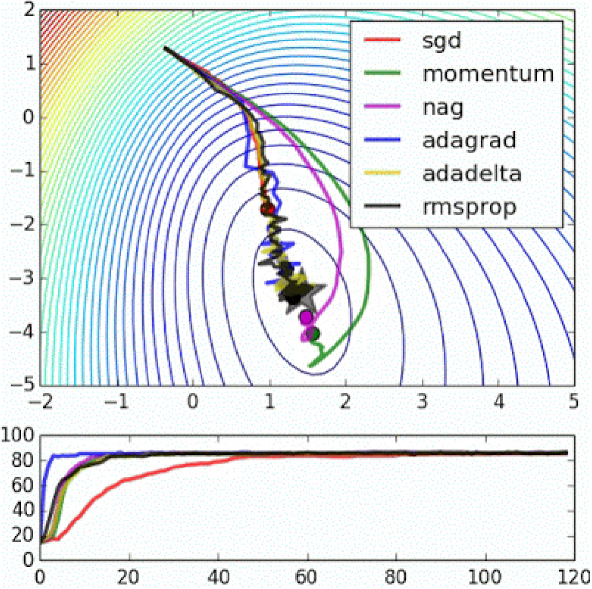
$$\alpha(t) = \alpha_0 \left(1 - \frac{t}{t_{max}}\right).$$

  - $\rightarrow$ Decaimento exponencial:

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

- Adaptar α para cada parâmetro ao longo das iterações.
  - $\rightarrow$ Adagrad, Adadelta, RMSprop, Adam...

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)**:

  $\rightarrow$ Função custo penaliza a norma quadrática dos pesos:

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

  $\rightarrow$ Atualização dos pesos regularizada:

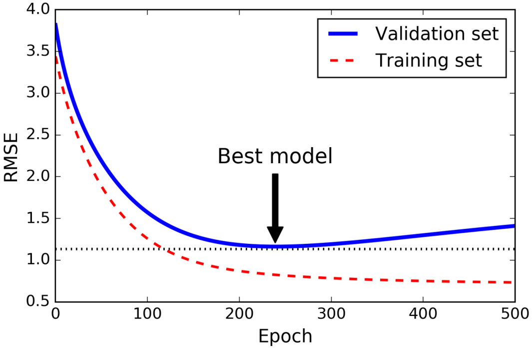
$$\boldsymbol{w}_j(t+1) = \boldsymbol{w}_j(t) - \alpha\frac{\partial \mathcal{J}}{\partial \boldsymbol{w}_j(t)} - \alpha\lambda\boldsymbol{w}_j(t), \ \lambda \geq 0 \text{ (termo de regularização).}$$

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

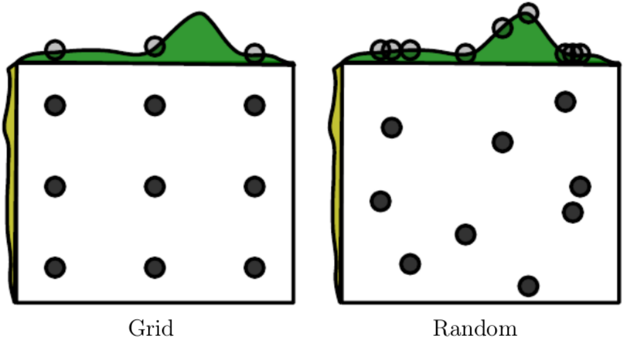
- **Problema**: Como inicializar os pesos de uma rede MLP?
- Inicialização aleatória baseada em heurísticas.
- Exemplos:
  - $\rightarrow$ Pesos amostrados de $\sqrt{\frac{1}{N_{in}}}\mathcal{N}(0,1)$, N_{in} é o número de entradas no neurônio.
  - $\rightarrow$ Bias iniciado com zero.
  - $\rightarrow$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

Técnicas para treinamento de redes neurais

- Redes neurais possuem vários hiperparâmetros:
  - $\rightarrow$ Número de camadas ocultas/neurônios, funções de ativação.
  - $\rightarrow$ Passo de aprendizagem, termo de momentum/regularização.
  - $\rightarrow$ 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 candidados;
   - Exemplo: taxa de aprendizagem $\alpha = 10^{\mathsf{U}(-5,-1)}$
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 neurais não-supervisionadas.
- Aprendizagem Bayesiana de redes neurais.
- Modelos generativos com redes neurais.

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

- **Caps. 8 e 16** - MURPHY, Kevin P. **Machine learning: a probabilistic perspective**, 2012.
- **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.