Deep Learning

Grafo Computacional

Lucas Silveira Kupssinskü

Machine Learning Theory and Applications Laboratory
PUCRS

agosto de 2026

Visão Geral

1. **Definição e exemplo simples**
2. **Forward e backward genéricos**
3. **Exemplo: regressão logística**
4. **O MLP como grafo computacional**

Visão Geral

1. **Definição e exemplo simples**
2. **Forward e backward genéricos**
3. **Exemplo: regressão logística**
4. **O MLP como grafo computacional**

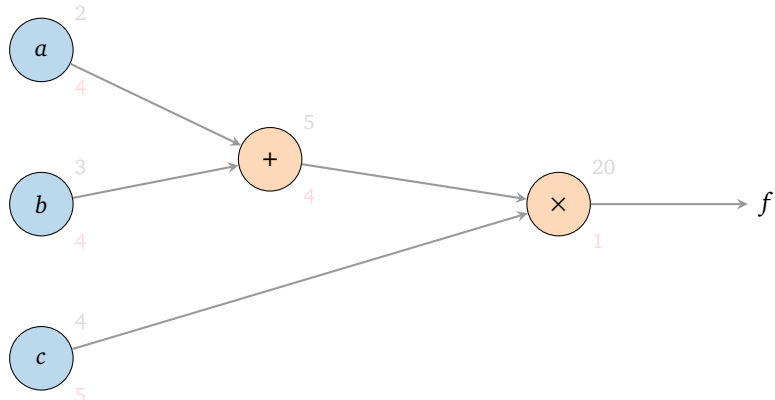
Por que pensar em grafo computacional?

- *Backpropagation* é a aplicação repetida da regra da cadeia em uma função composta.
- Funções complicadas (uma rede profunda inteira) podem ser decompostas em operações elementares.
- Representar essa decomposição como um grafo torna explícito o caminho que cada gradiente local precisa percorrer.
- Mesma estrutura permite ao computador fazer a derivada automaticamente, ideia central de *frameworks* como PyTorch e JAX.

Definição

- Um grafo computacional é um DAG (*directed acyclic graph*) em que:
  - nodos-folha representam **variáveis** (entradas, parâmetros);
  - nodos internos representam **operações** elementares (soma, produto, exp, ReLU, ...);
  - arestas indicam o fluxo de dados, das entradas em direção à saída final.
- Avaliar a função é fazer um forward pass: percorrer o grafo em ordem topológica.
- Calcular gradientes em relação a cada folha é fazer um backward pass: percorrer o grafo na ordem inversa, aplicando a regra da cadeia.

Exemplo simples[1]: $f(a, b, c) = (a + b) \cdot c$

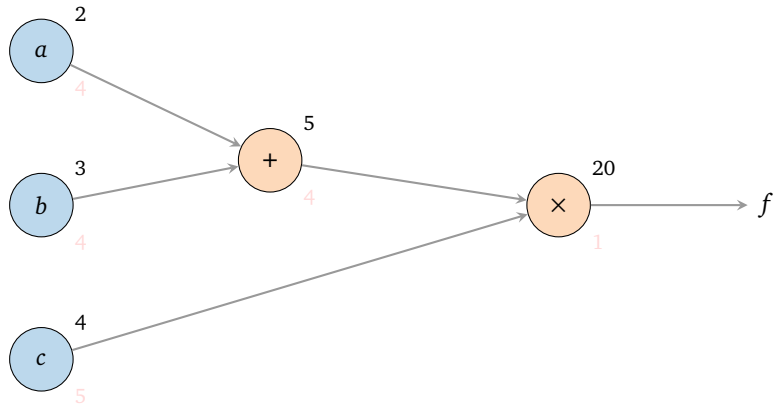


Valores em **preto** (forward), gradientes em **vermelho** (backward).

Nodo × comuta valores: gradiente recebe o valor da *outra* entrada. Nodo + distribui o gradiente igualmente.

[1] Andrej Karpathy, Fei-Fei Li e Justin Johnson. CS231n: Convolutional Neural Networks for Visual Recognition — Backpropagation, Intuitions. https://cs231n.github.io/optimization-2/. Stanford University course notes. 2016.

Exemplo simples[1]: $f(a, b, c) = (a + b) \cdot c$

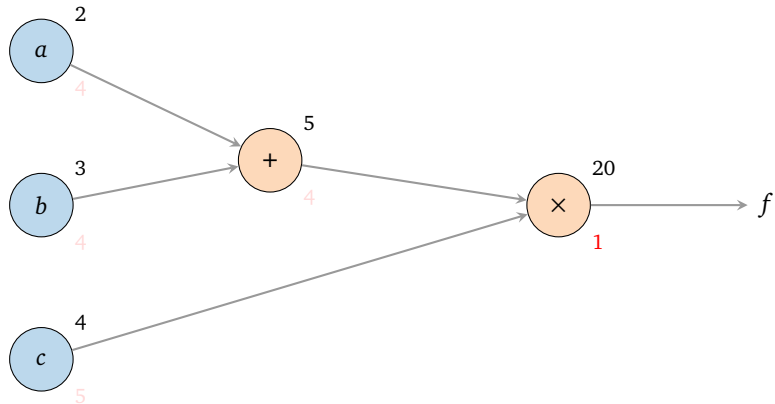


Valores em **preto** (forward), gradientes em **vermelho** (backward).

Nodo × comuta valores: gradiente recebe o valor da *outra* entrada. Nodo + distribui o gradiente igualmente.

[1] Andrej Karpathy, Fei-Fei Li e Justin Johnson. CS231n: Convolutional Neural Networks for Visual Recognition — Backpropagation, Intuitions. https://cs231n.github.io/optimization-2/. Stanford University course notes. 2016.

Exemplo simples[1]: $f(a, b, c) = (a + b) \cdot c$

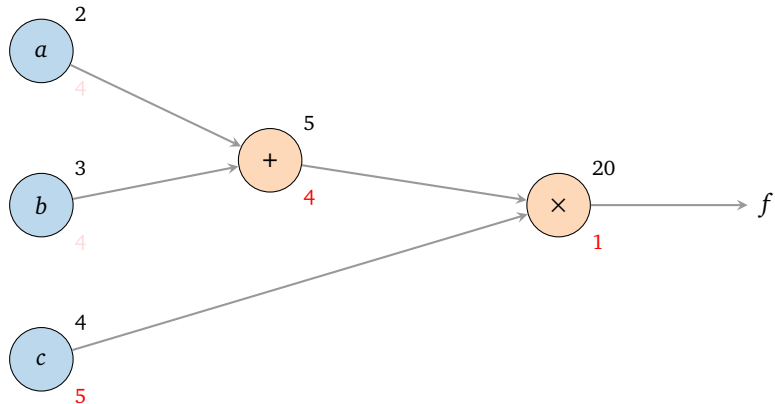


Valores em **preto** (forward), gradientes em **vermelho** (backward).

Nodo × comuta valores: gradiente recebe o valor da *outra* entrada. Nodo + distribui o gradiente igualmente.

[1] Andrej Karpathy, Fei-Fei Li e Justin Johnson. CS231n: Convolutional Neural Networks for Visual Recognition — Backpropagation, Intuitions. https://cs231n.github.io/optimization-2/. Stanford University course notes. 2016.

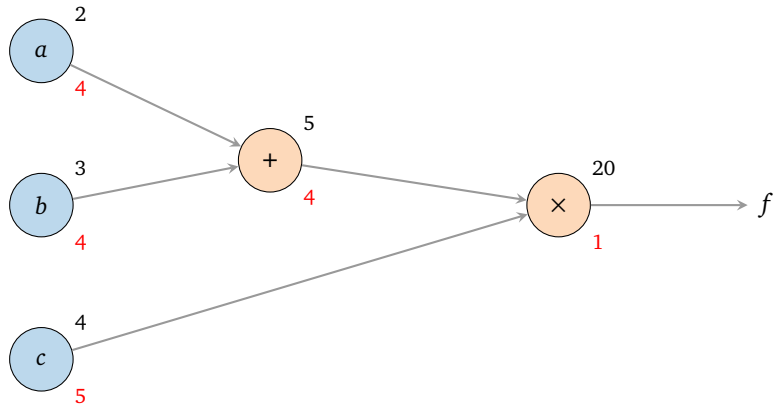
Exemplo simples[1]: $f(a, b, c) = (a + b) \cdot c$



Valores em **preto** (forward), gradientes em **vermelho** (backward).
Nodo × comuta valores: gradiente recebe o valor da *outra* entrada. Nodo + distribui o gradiente igualmente.

[1] Andrej Karpathy, Fei-Fei Li e Justin Johnson. CS231n: Convolutional Neural Networks for Visual Recognition — Backpropagation, Intuitions. https://cs231n.github.io/optimization-2/. Stanford University course notes. 2016.

Exemplo simples[1]: $f(a, b, c) = (a + b) \cdot c$



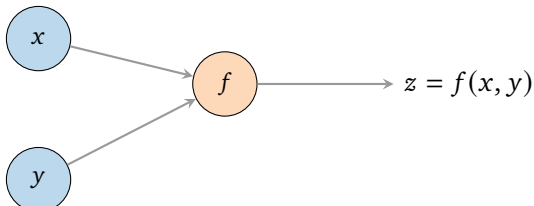
Valores em **preto** (forward), gradientes em **vermelho** (backward).
Nodo × comuta valores: gradiente recebe o valor da *outra* entrada. Nodo + distribui o gradiente igualmente.

[1] Andrej Karpathy, Fei-Fei Li e Justin Johnson. CS231n: Convolutional Neural Networks for Visual Recognition — Backpropagation, Intuitions. https://cs231n.github.io/optimization-2/. Stanford University course notes. 2016.

Visão Geral

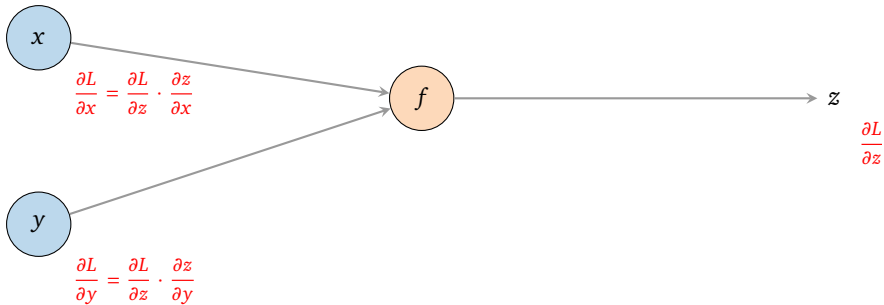
1. Definição e exemplo simples
2. **Forward e backward genéricos**
3. Exemplo: regressão logística
4. O MLP como grafo computacional

Um nodo genérico no *forward*



- No *forward*, cada nodo recebe valores de entrada e calcula sua saída local.
- As saídas alimentam os nodos seguintes na ordem topológica.
- No final do grafo temos o valor da função, tipicamente uma *loss* escalar L.

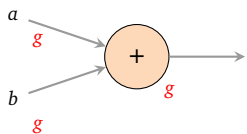
O mesmo nodo no *backward*



- Cada aresta no sentido oposto carrega o gradiente *upstream* $\partial L/\partial z$.
- O nodo multiplica esse gradiente pela derivada local $\partial z/\partial x$ (regra da cadeia) e envia para a aresta de entrada.
- Se uma variável aparece em múltiplos nodos, os gradientes que chegam nela são **somados**.

Padrões comuns de nodos

- **Soma** (+): *distribuidor*, copia o gradiente *upstream* para todas as entradas.
- **Produto** ($\times$): *comutador*, multiplica o gradiente pelo valor da *outra* entrada.
- **Máximo** (max): *roteador*, envia o gradiente inteiro para a entrada que venceu, zero para as demais.
- **Cópia** (uma variável usada duas vezes): *somador* no backward, soma os gradientes que chegam.



Visão Geral

1. Definição e exemplo simples
2. Forward e backward genéricos
3. **Exemplo: regressão logística**
4. O MLP como grafo computacional

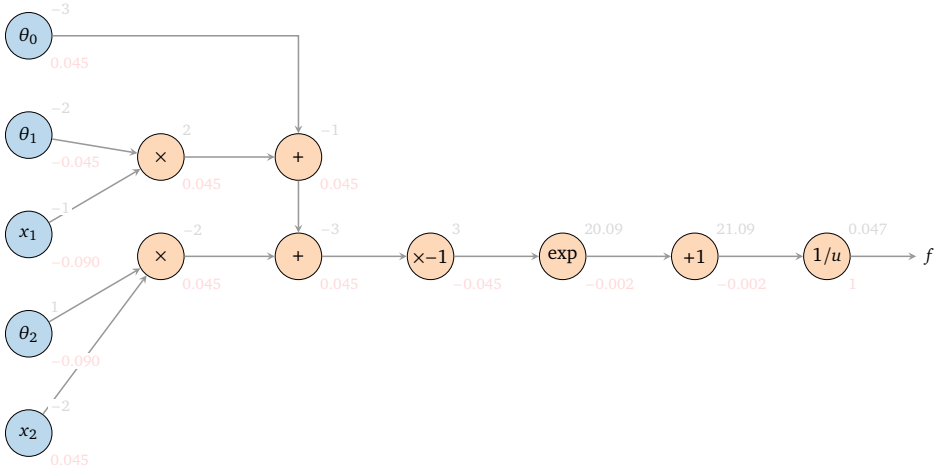
Outro exemplo: regressão logística

- Considere a função

$$f(\boldsymbol{\theta}, \boldsymbol{x}) = \frac{1}{1 + \exp\big(-(\theta_0 + \theta_1 x_1 + \theta_2 x_2)\big)}.$$

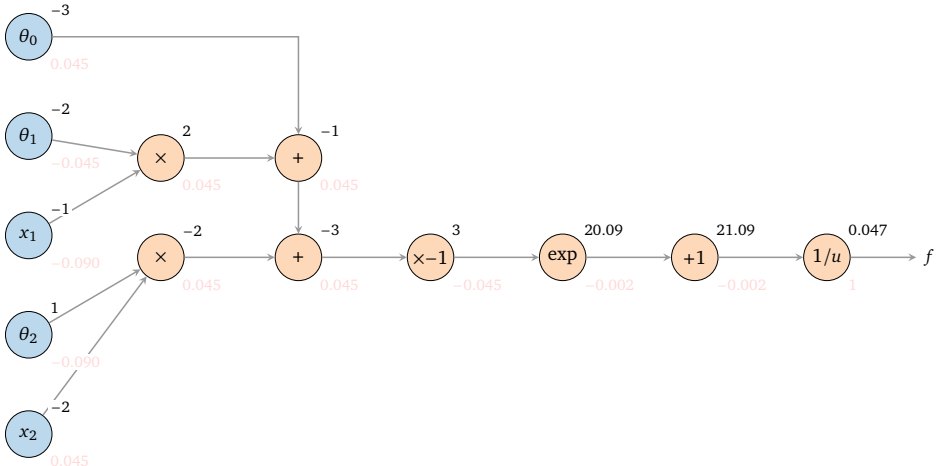
- Composição de nodos elementares: combinação afim, $\times(-1)$, exp, $+1$, $1/u$.
- Vamos avaliar para $\theta_0 = -3,\ \theta_1 = -2,\ \theta_2 = 1,\ x_1 = -1,\ x_2 = -2$, depois rodar o backward.
- No fim, observaremos que vários nodos podem ser agrupados em uma única operação *sigmoide*.

Grafo da regressão logística



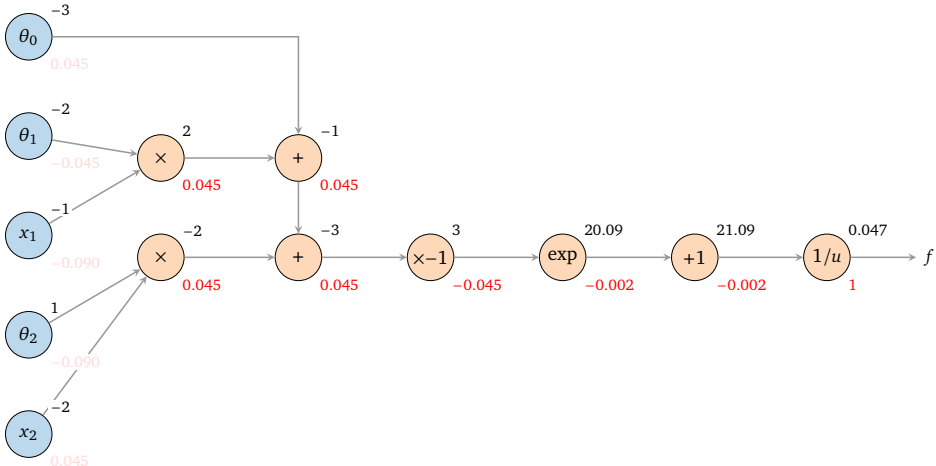
Valores em **preto** (forward), gradientes em **vermelho** (backward).

Grafo da regressão logística



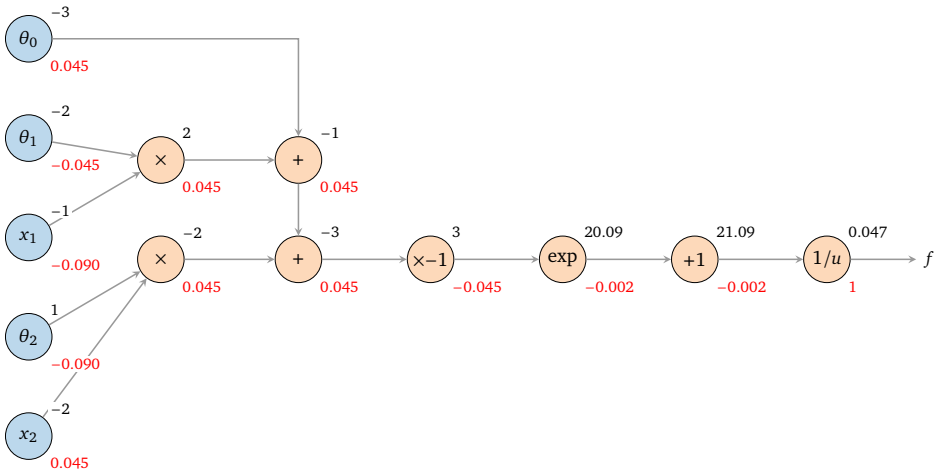
Valores em **preto** (forward), gradientes em **vermelho** (backward).

Grafo da regressão logística



Valores em **preto** (forward), gradientes em **vermelho** (backward).

Grafo da regressão logística



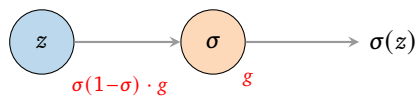
Valores em **preto** (forward), gradientes em **vermelho** (backward).

Agrupando nodos: a sigmoide

- A sequência $\times -1 \to \exp \to +1 \to 1/u$ implementa $\sigma(z) = 1/(1 + e^{-z})$.
- Sua derivada tem forma fechada:

$$\sigma'(z) = \sigma(z)\,(1 - \sigma(z)).$$

- Substituir os 4 nodos por um único nodo *sigmoide* simplifica o grafo, evita instabilidades numéricas e cobra apenas uma multiplicação no *backward*.
- Granularidade fina (flexível) versus grossa (eficiente) é uma decisão recorrente em *frameworks* de *deep learning*.



Forward entrega $\sigma(z)$; *backward* multiplica o gradiente *upstream* por $\sigma(1-\sigma)$.

Visão Geral

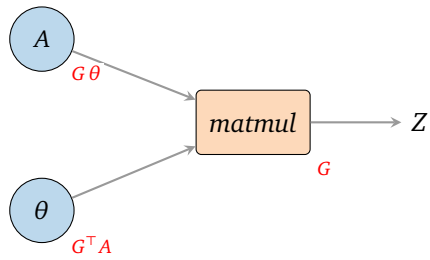
1. Definição e exemplo simples
2. Forward e backward genéricos
3. Exemplo: regressão logística
4. **O MLP como grafo computacional**

Nodos vetorizados: produto de matrizes

- No MLP vetorizado, a pré-ativação de cada camada é um produto de matrizes: $Z = A\,\theta^\top$, com $A \in \mathbb{R}^{N \times d_e}$ (entradas da camada), $\theta \in \mathbb{R}^{d_s \times d_e}$ (pesos) e $Z \in \mathbb{R}^{N \times d_s}$.
- No *backward*, o gradiente *upstream* é $G = \partial J / \partial Z$, com a mesma forma de Z:

$$\frac{\partial J}{\partial A} = G\,\theta \qquad \frac{\partial J}{\partial \theta} = G^\top A$$

- Mesma regra do nodo × escalar: cada entrada recebe o gradiente multiplicado pela *outra* entrada.
- Confira as dimensões: $G\,\theta$ é $N \times d_e$ (forma de A) e $G^\top A$ é $d_s \times d_e$ (forma de θ).

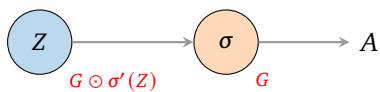


Nodos vetorizados: ativação elemento a elemento

- O nodo de ativação aplica σ em cada posição da matriz: $A = \sigma(Z)$, e A tem a mesma forma de Z.
- Como o nodo não mistura posições, o *backward* é a regra escalar aplicada elemento a elemento:

$$\frac{\partial J}{\partial Z} = G \odot \sigma'(Z) \qquad \sigma'(Z) = A \odot (1 - A)$$

- $\odot$ é o produto de Hadamard (elemento a elemento), o mesmo da aula de *backpropagation*.
- Como no nodo sigmoide escalar, σ' reaproveita a saída A já calculada no *forward*.

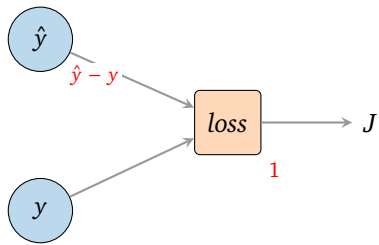


Nodos vetorizados: função de custo

- O último nodo compara a saída da rede com o alvo e reduz tudo a um escalar (*half SSE* da aula anterior):

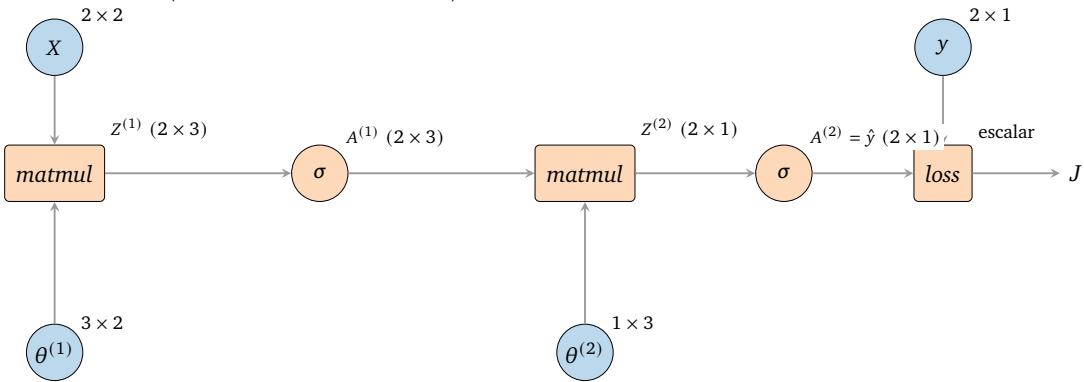
$$J = \frac{1}{2}\sum_i (y_i - \hat{y}_i)^2 \qquad \frac{\partial J}{\partial \hat{y}} = \hat{y} - y$$

- O *backward* começa nesse escalar, com $\partial J/\partial J = 1$; o gradiente devolvido tem a mesma forma de $\hat{y}$ (o fator $\frac{1}{2}$ cancela o 2 da derivada).
- y é dado do problema: não é treinável, então nenhum gradiente é propagado para ele.

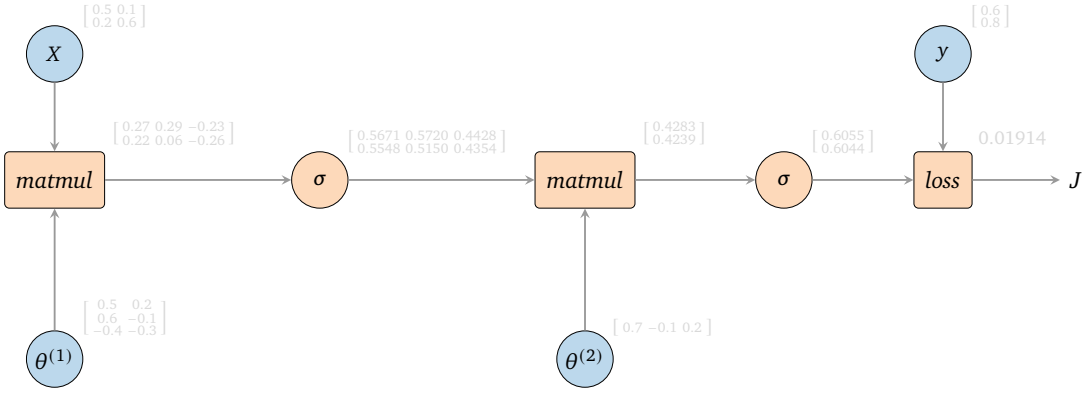


O MLP da aula anterior como grafo computacional

- Mesma rede do exemplo vetorizado: 2 entradas, camada oculta com 3 unidades, 1 saída, sigmoide nas duas camadas, sem *bias*.
- *Batch* com 2 exemplos: cinco nodos encadeados computam $J = \frac{1}{2} \sum \left(y - \sigma\big(\sigma(X\theta^{(1)\top})\theta^{(2)\top}\big) \right)^2$.

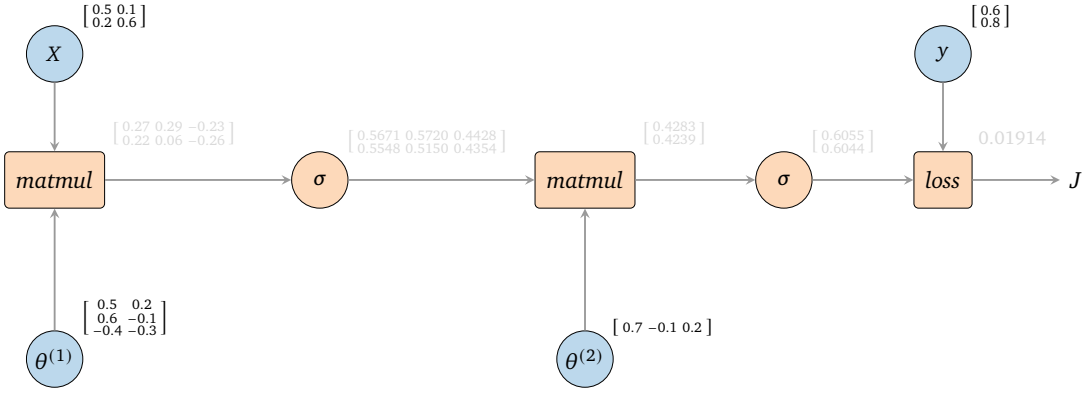


MLP no grafo: *forward pass*



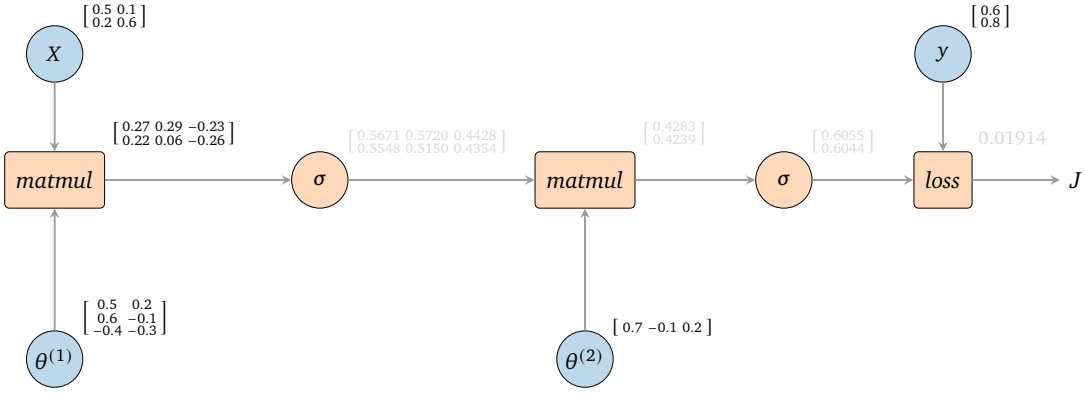
$$Z^{(1)} = X\,\theta^{(1)\top} \quad A^{(1)} = \sigma(Z^{(1)}) \quad Z^{(2)} = A^{(1)}\theta^{(2)\top} \quad \hat{y} = A^{(2)} = \sigma(Z^{(2)}) \quad J = \tfrac{1}{2}\sum (y - \hat{y})^2$$

MLP no grafo: *forward pass*



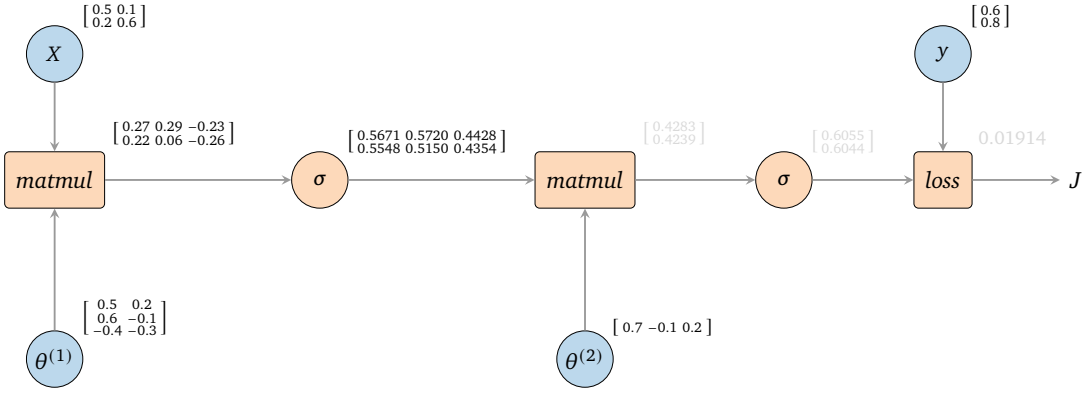
$$Z^{(1)} = X\,\theta^{(1)^\top} \quad A^{(1)} = \sigma(Z^{(1)}) \quad Z^{(2)} = A^{(1)}\theta^{(2)^\top} \quad \hat{y} = A^{(2)} = \sigma(Z^{(2)}) \quad J = \frac{1}{2}\sum (y - \hat{y})^2$$

MLP no grafo: *forward pass*



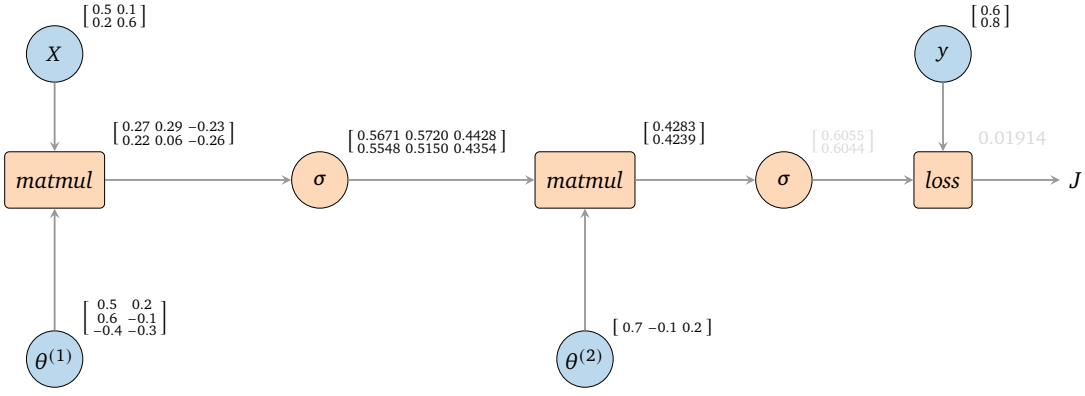
$$Z^{(1)} = X\,\theta^{(1)\top} \quad A^{(1)} = \sigma(Z^{(1)}) \quad Z^{(2)} = A^{(1)}\theta^{(2)\top} \quad \hat{y} = A^{(2)} = \sigma(Z^{(2)}) \quad J = \tfrac{1}{2}\sum (y - \hat{y})^2$$

MLP no grafo: *forward pass*



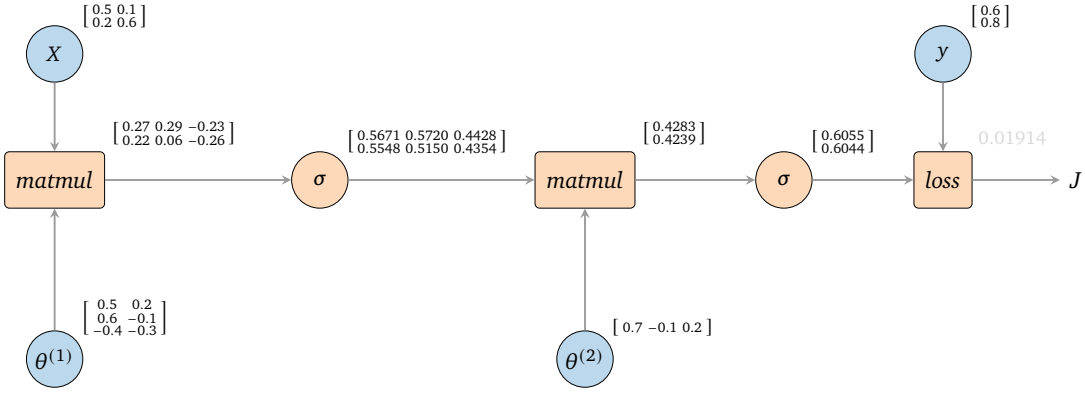
$$Z^{(1)} = X\,\theta^{(1)\top} \quad A^{(1)} = \sigma(Z^{(1)}) \quad Z^{(2)} = A^{(1)}\theta^{(2)\top} \quad \hat{y} = A^{(2)} = \sigma(Z^{(2)}) \quad J = \frac{1}{2}\sum (y - \hat{y})^2$$

MLP no grafo: *forward pass*



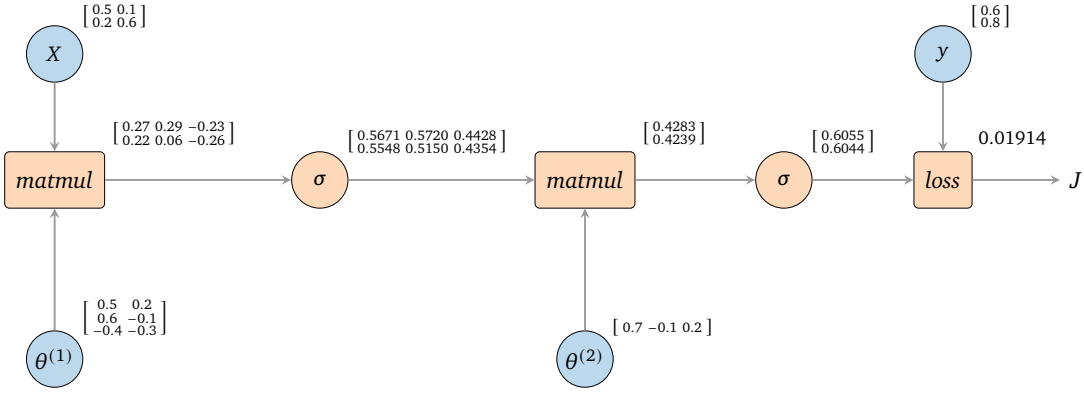
$$Z^{(1)} = X\,\theta^{(1)\top} \quad A^{(1)} = \sigma(Z^{(1)}) \quad Z^{(2)} = A^{(1)}\theta^{(2)\top} \quad \hat{y} = A^{(2)} = \sigma(Z^{(2)}) \quad J = \tfrac{1}{2}\sum (y - \hat{y})^2$$

MLP no grafo: *forward pass*



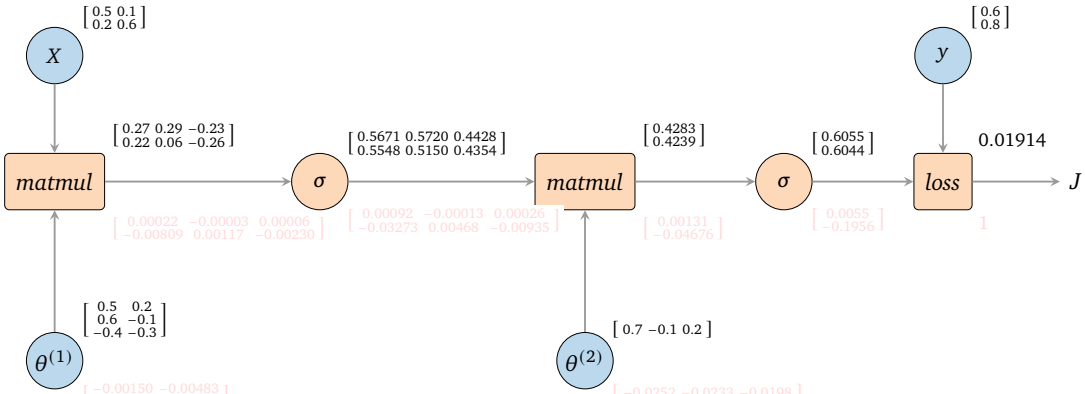
$$Z^{(1)} = X\,\theta^{(1)\top} \quad A^{(1)} = \sigma(Z^{(1)}) \quad Z^{(2)} = A^{(1)}\,\theta^{(2)\top} \quad \hat{y} = A^{(2)} = \sigma(Z^{(2)}) \quad J = \tfrac{1}{2}\sum (y - \hat{y})^2$$

MLP no grafo: *forward pass*



$$Z^{(1)} = X\,\theta^{(1)\top} \quad A^{(1)} = \sigma(Z^{(1)}) \quad Z^{(2)} = A^{(1)}\theta^{(2)\top} \quad \hat{y} = A^{(2)} = \sigma(Z^{(2)}) \quad J = \tfrac{1}{2}\sum (y - \hat{y})^2$$

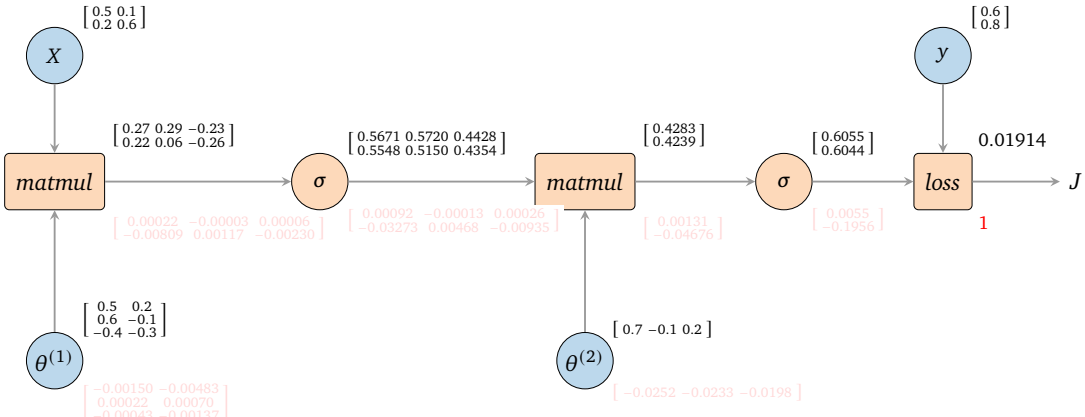
MLP no grafo: *backward pass*



$$\frac{\partial J}{\partial \hat{y}} = \hat{y} - y \quad \delta^{(2)} = \frac{\partial J}{\partial \hat{y}} \odot \sigma'(Z^{(2)}) \quad \frac{\partial J}{\partial \theta^{(2)}} = \delta^{(2)\top} A^{(1)} \quad \frac{\partial J}{\partial A^{(1)}} = \delta^{(2)} \theta^{(2)} \quad \delta^{(1)} = \frac{\partial J}{\partial A^{(1)}} \odot \sigma'(Z^{(1)}) \quad \frac{\partial J}{\partial \theta^{(1)}} = \delta^{(1)\top} X$$

X e y não são treináveis, então não precisam de gradiente. Os gradientes de $\theta^{(1)}$ e $\theta^{(2)}$ são os mesmos calculados à mão na aula de *backpropagation*.

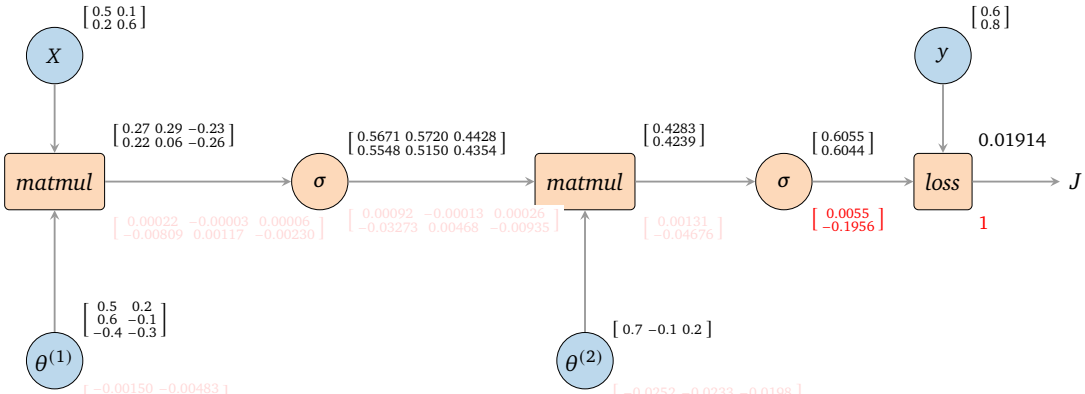
MLP no grafo: *backward pass*



$$\frac{\partial J}{\partial \hat{y}} = \hat{y} - y \quad \delta^{(2)} = \frac{\partial J}{\partial \hat{y}} \odot \sigma'(Z^{(2)}) \quad \frac{\partial J}{\partial \theta^{(2)}} = {\delta^{(2)}}^\top A^{(1)} \quad \frac{\partial J}{\partial A^{(1)}} = \delta^{(2)} \theta^{(2)} \quad \delta^{(1)} = \frac{\partial J}{\partial A^{(1)}} \odot \sigma'(Z^{(1)}) \quad \frac{\partial J}{\partial \theta^{(1)}} = {\delta^{(1)}}^\top X$$

X e y não são treináveis, então não precisam de gradiente. Os gradientes de $\theta^{(1)}$ e $\theta^{(2)}$ são os mesmos calculados à mão na aula de *backpropagation*.

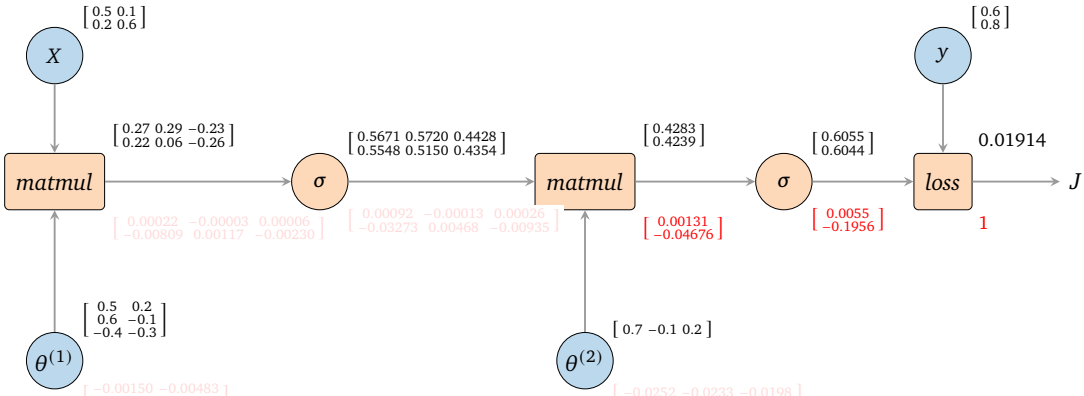
MLP no grafo: *backward pass*



$\frac{\partial J}{\partial \hat{y}} = \hat{y} - y \quad \delta^{(2)} = \frac{\partial J}{\partial \hat{y}} \odot \sigma'(Z^{(2)}) \quad \frac{\partial J}{\partial \theta^{(2)}} = \delta^{(2)\top} A^{(1)} \quad \frac{\partial J}{\partial A^{(1)}} = \delta^{(2)} \theta^{(2)} \quad \delta^{(1)} = \frac{\partial J}{\partial A^{(1)}} \odot \sigma'(Z^{(1)}) \quad \frac{\partial J}{\partial \theta^{(1)}} = \delta^{(1)\top} X$

X e y não são treináveis, então não precisam de gradiente. Os gradientes de $\theta^{(1)}$ e $\theta^{(2)}$ são os mesmos calculados à mão na aula de *backpropagation*.

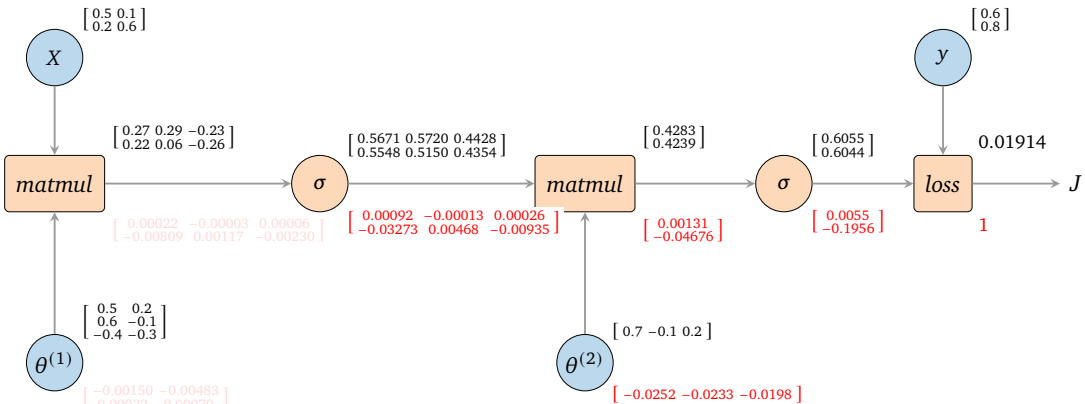
MLP no grafo: *backward pass*



$$\frac{\partial J}{\partial \hat{y}} = \hat{y} - y \quad \delta^{(2)} = \frac{\partial J}{\partial \hat{y}} \odot \sigma'(Z^{(2)}) \quad \frac{\partial J}{\partial \theta^{(2)}} = \delta^{(2)\top} A^{(1)} \quad \frac{\partial J}{\partial A^{(1)}} = \delta^{(2)} \theta^{(2)} \quad \delta^{(1)} = \frac{\partial J}{\partial A^{(1)}} \odot \sigma'(Z^{(1)}) \quad \frac{\partial J}{\partial \theta^{(1)}} = \delta^{(1)\top} X$$

X e y não são treináveis, então não precisam de gradiente. Os gradientes de $\theta^{(1)}$ e $\theta^{(2)}$ são os mesmos calculados à mão na aula de *backpropagation*.

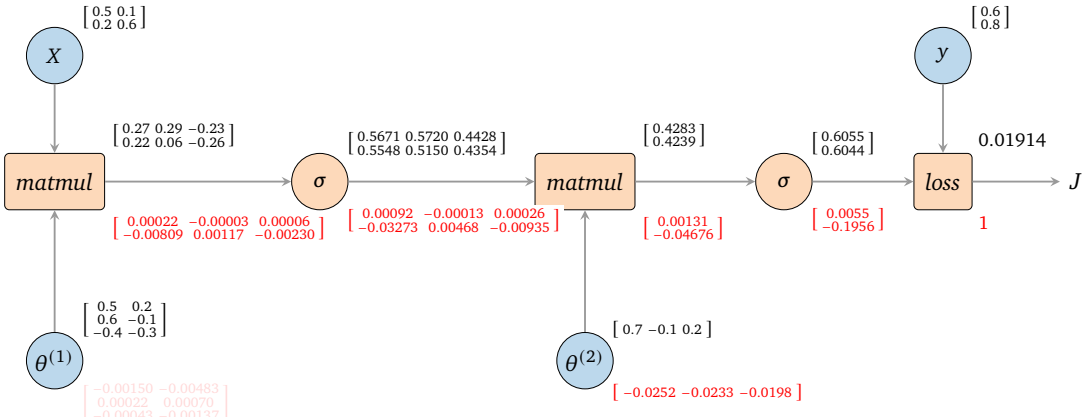
MLP no grafo: *backward pass*



$$\frac{\partial J}{\partial \hat{y}} = \hat{y} - y \quad \delta^{(2)} = \frac{\partial J}{\partial \hat{y}} \odot \sigma'(Z^{(2)}) \quad \frac{\partial J}{\partial \theta^{(2)}} = \delta^{(2)\top} A^{(1)} \quad \frac{\partial J}{\partial A^{(1)}} = \delta^{(2)} \theta^{(2)} \quad \delta^{(1)} = \frac{\partial J}{\partial A^{(1)}} \odot \sigma'(Z^{(1)}) \quad \frac{\partial J}{\partial \theta^{(1)}} = \delta^{(1)\top} X$$

X e y não são treináveis, então não precisam de gradiente. Os gradientes de $\theta^{(1)}$ e $\theta^{(2)}$ são os mesmos calculados à mão na aula de *backpropagation*.

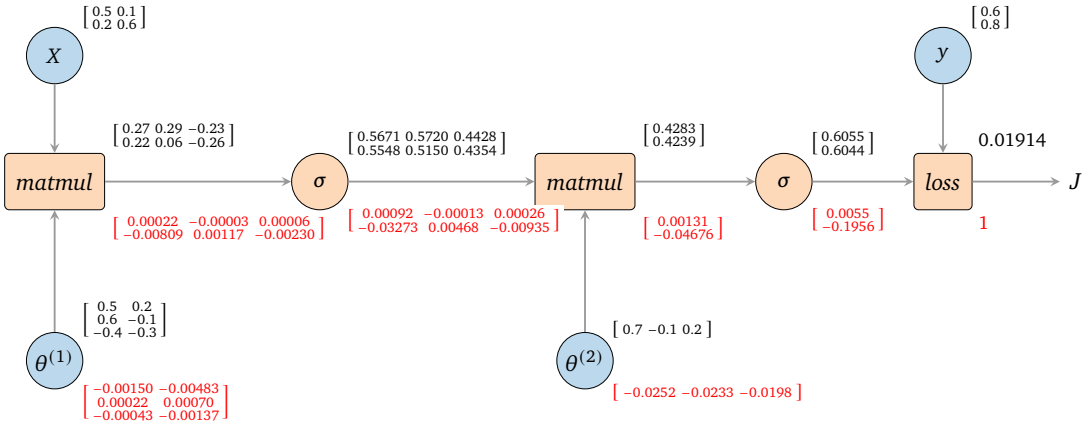
MLP no grafo: *backward pass*



$$\frac{\partial J}{\partial \hat{y}} = \hat{y} - y \quad \delta^{(2)} = \frac{\partial J}{\partial \hat{y}} \odot \sigma'(Z^{(2)}) \quad \frac{\partial J}{\partial \theta^{(2)}} = \delta^{(2)\top} A^{(1)} \quad \frac{\partial J}{\partial A^{(1)}} = \delta^{(2)} \theta^{(2)} \quad \delta^{(1)} = \frac{\partial J}{\partial A^{(1)}} \odot \sigma'(Z^{(1)}) \quad \frac{\partial J}{\partial \theta^{(1)}} = \delta^{(1)\top} X$$

X e y não são treináveis, então não precisam de gradiente. Os gradientes de $\theta^{(1)}$ e $\theta^{(2)}$ são os mesmos calculados à mão na aula de *backpropagation*.

MLP no grafo: *backward pass*



$$\frac{\partial J}{\partial \hat{y}} = \hat{y} - y \quad \delta^{(2)} = \frac{\partial J}{\partial \hat{y}} \odot \sigma'(Z^{(2)}) \quad \frac{\partial J}{\partial \theta^{(2)}} = \delta^{(2)\top} A^{(1)} \quad \frac{\partial J}{\partial A^{(1)}} = \delta^{(2)} \theta^{(2)} \quad \delta^{(1)} = \frac{\partial J}{\partial A^{(1)}} \odot \sigma'(Z^{(1)}) \quad \frac{\partial J}{\partial \theta^{(1)}} = \delta^{(1)\top} X$$

X e y não são treináveis, então não precisam de gradiente. Os gradientes de $\theta^{(1)}$ e $\theta^{(2)}$ são os mesmos calculados à mão na aula de *backpropagation*.

Redes neurais são implementadas como Grafos

- Podemos decompor funções em nodos com *forward* e derivada local: nodos escalares ($+$, $\times$, exp, $1/u$), o nodo sigmoide e os nodos vetorizados (*matmul*, ativação, *loss*).
- Todas as arquiteturas que veremos no curso são decompostas da mesma forma
- Se cada nodo conhece seu *forward* e sua derivada local, o *backpropagation* no grafo computacional treina qualquer arquitetura, por mais complexa que pareça.

Leituras Recomendadas

- Notas de aula CS231n, módulo *Backpropagation, Intuitions*: (Karpathy, Li e Johnson, 2016), *cs231n.github.io/optimization-2*.
- Survey de *automatic differentiation*: (Baydin et al., 2018).
- Capítulo 7 de Simon J. D. Prince. Understanding Deep Learning. MIT Press, 2023.
- PyTorch *autograd*: (Paszke et al., 2019) e documentação oficial em *pytorch.org/docs/stable/notes/autograd*.

Referências I

[1] Atilim Gunes Baydin et al. “Automatic differentiation in machine learning: a survey”. Em: JMLR 18.153 (2018), pp. 1–43.

[2] Andrej Karpathy, Fei-Fei Li e Justin Johnson. CS231n: Convolutional Neural Networks for Visual Recognition — Backpropagation, Intuitions. https://cs231n.github.io/optimization-2/. Stanford University course notes. 2016.

[3] Adam Paszke et al. “PyTorch: An imperative style, high-performance deep learning library”. Em: NeurIPS. Vol. 32. 2019.

[4] Simon J. D. Prince. Understanding Deep Learning. MIT Press, 2023.