Deep Learning

Redes Recorrentes

Lucas Silveira Kupssinskü

Machine Learning Theory and Applications Laboratory
PUCRS

agosto de 2026

Overview

1. Sequências
2. Tentando Modelar com MLP e CNN
3. Recorrência em Redes Neurais
4. Backpropagation Through Time
5. Considerações Práticas

Visão Geral

Sequências em Aprendizado de Máquina

- Muitos problemas relevantes envolvem dados **sequenciais**, em que a ordem dos elementos importa.
- Exemplos:
  - Análise de sentimento de uma frase ou resenha.
  - *Image captioning*: gerar uma descrição textual a partir de uma imagem.
  - Tradução automática entre idiomas.
  - Previsão de séries temporais (preço de ações, clima, energia).
  - Reconhecimento de fala e síntese de áudio.
- Em todos esses casos, mudar a ordem dos elementos altera o significado da entrada ou da saída.

Análise de Sentimento

- A tarefa: dada uma frase, prever se o sentimento é positivo ou negativo.

"esse filme é uma bela porcaria"

sentimento **negativo**

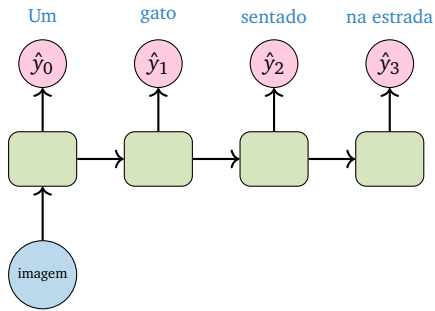
"eu esperava que fosse uma porcaria, mas é uma bela forma de passar o tempo"

sentimento **positivo**

As mesmas palavras ("bela", "porcaria") aparecem nas duas frases: a **ordem** faz toda a diferença!

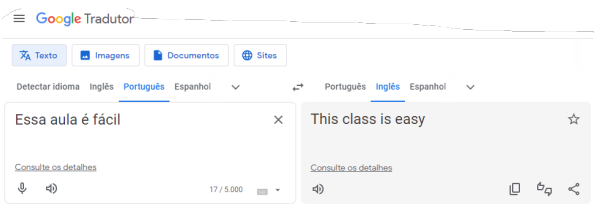
Exemplo: *Image Captioning* (*one-to-many*)

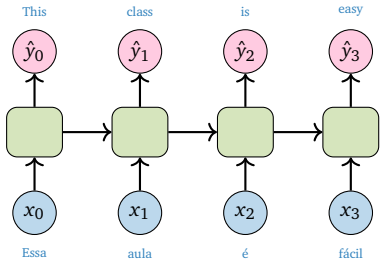
"Um gato sentado na estrada"



Uma única entrada (a imagem) gera uma **sequência** de palavras.

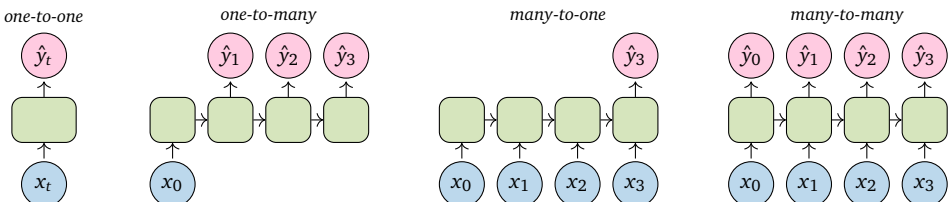
Exemplo: Tradução Automática (*many-to-many*)





Uma sequência de entrada gera uma sequência de saída.

Tipos de Problemas Sequenciais



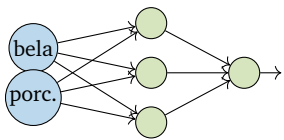
- *one-to-many*: *image captioning* (uma imagem gera uma sequência de palavras).
- *many-to-one*: análise de sentimento (uma sequência gera um rótulo).
- *many-to-many*: tradução automática, geração de texto.

Visão Geral

Uma Ideia: Usar um MLP

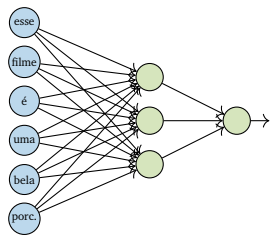
- Vamos usar um MLP para classificar o sentimento. Cada palavra vira uma *embedding* e a camada de entrada precisa de pesos para cada palavra.

"bela porcaria"



2 palavras: $2 \times 3 = \mathbf{6}$ pesos de entrada

"esse filme é uma bela porcaria"



6 palavras: $6 \times 3 = \mathbf{18}$ pesos de entrada

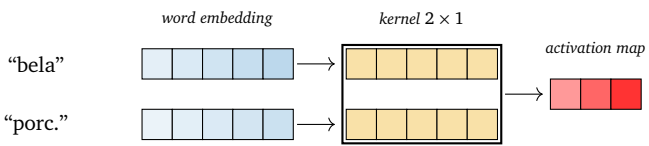
Quanto maior a frase, mais parâmetros treináveis na **camada de entrada**, e o tamanho da entrada muda de frase para frase.

Problemas com MLP em Sequências

- **Frases de tamanhos diferentes** demandariam entradas de tamanhos diferentes, com pesos diferentes.
- Entradas maiores demandam mais pesos.
- A mesma palavra em posições diferentes é multiplicada por pesos diferentes.
  - Cada posição tem que aprender separadamente o significado da palavra.
- Não há mecanismo natural para tratar entradas de comprimento variável.

Outra Ideia: Usar uma CNN 1D

- Vamos tentar usar uma CNN para classificar o sentimento da frase.
- Cada palavra é convertida em uma *word embedding*, e um *kernel* desliza ao longo da sequência.



Limitações da CNN 1D

- Para frases mais longas, o problema fica mais nítido:
  - Contextos “distantes” são difíceis de modelar.
  - Precisamos empilhar várias camadas convolucionais para aumentar o campo receptivo.
- Mesmo com várias camadas, o campo receptivo é finito e o tamanho da saída depende do tamanho da entrada.

Não são Boas Soluções

- MLPs exigem entradas de tamanho fixo. Frases têm tamanho variável.
- CNNs podem trabalhar com entradas de tamanho variável, mas se não tomarmos cuidado com o campo receptivo corremos o risco de não levar em conta a frase toda.

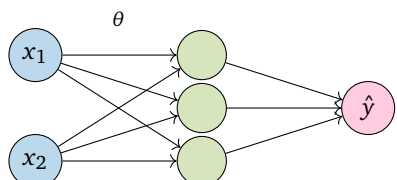
Precisamos de uma rede que:

- Trabalhe sequencialmente com as palavras, uma por vez.
- Mantenha um **estado** (memória) que carregue informações obtidas anteriormente.

Visão Geral

Revisitando o MLP

- Em uma camada densa, temos:
  - entradas x,
  - pesos θ,
  - função de ativação,
  - saídas $\hat{y}$.

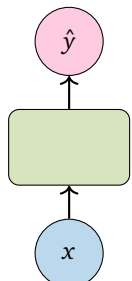


Revisitando o MLP

- Podemos **colapsar** o vetor de entrada em um único nó e a camada inteira em um único bloco.
- O desenho fica mais compacto, mas representa exatamente a mesma rede.

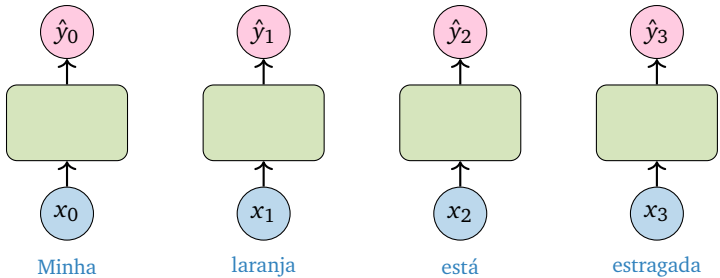
Revisitando o MLP

- Agora **rotacionamos** o desenho: a entrada fica embaixo e a saída em cima.
- É o mesmo MLP, apenas com a representação na vertical.
- O vetor de entrada foi colapsado, mas continua tendo um **tamanho fixo**.



Aplicando o MLP a uma Sequência

- Vamos rodar esse MLP na frase *"Minha laranja está estragada"*, palavra a palavra.



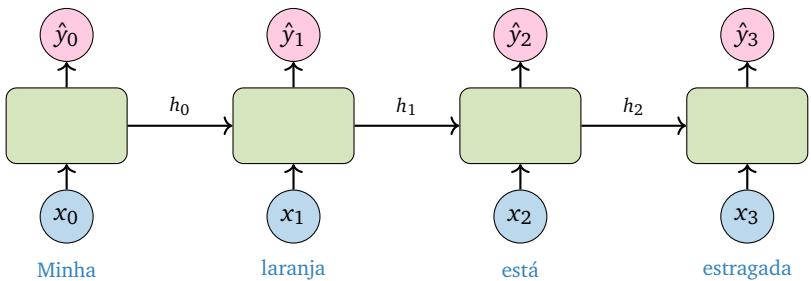
- Cada palavra é processada de forma independente.
- O modelo **não sabe** que "laranja" veio depois de "Minha".

O Problema Continua

- Qual é o problema dessa abordagem?
  - **O processamento de cada entrada é individual** e não leva em conta o restante da sequência.
  - Não há **memória** entre os instantes.
- Para um problema sequencial, precisamos que a saída em t dependa também do que aconteceu em $t-1, t-2, \ldots$

Solução: Recorrência

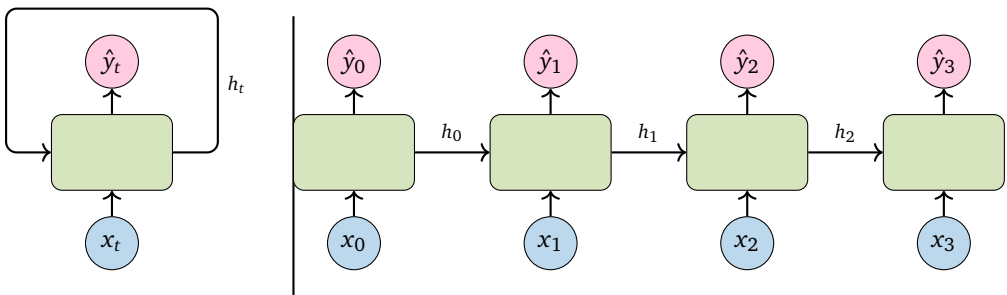
- Além da entrada x_t no instante atual, vamos passar adiante alguma informação do instante anterior.
- Chamamos essa informação de **estado interno** h_t.



- Os mesmos pesos são reutilizados em todos os passos de tempo.

Versão Dobrada vs Desdobrada

- A representação **desdobrada no tempo** pode ser compactada em uma versão **dobrada**, com uma única célula que se conecta a si mesma.



- Os dois desenhos representam a **mesma rede**: a versão desdobrada apenas explicita os passos de tempo.

Camadas Densas com Recorrência

- A saída de uma célula recorrente depende da **entrada** e do **estado interno**.
- Os pesos são reutilizados em todos os passos de tempo.

$$\underbrace{\left(\hat{y}_t\,,\; h_t\right)}_{\text{saída}} = f\Big(\underbrace{x_t}_{\text{entrada}}\,,\quad \underbrace{h_{t-1}}_{\substack{\text{memória ou}\\ \text{estado interno}}}\Big)$$

- O estado interno h_{t-1} resume tudo o que a rede “viu” até o passo anterior.

Pseudocódigo de uma RNN

```
rnn = RNN()
hidden_state = [0, 0, 0, 0]

sentence = ["Minha", "laranja", "está"]

for word in sentence:
    prediction, hidden_state = rnn(word, hidden_state)

# prediction -> palavra mais provavel (argmax sobre o vocabulario)
next_word = decode(prediction)
print(next_word)
# >>> "estragada"
```

- A cada iteração, a rede recebe a entrada atual e o estado interno, e devolve a predição e o novo estado.

Detalhando as Atualizações

- A célula recorrente mais simples (*vanilla RNN*) é dada por:

$$z_t = x_t\,\theta_{ih}^T + h_{t-1}\,\theta_{hh}^T + b_h, \qquad h_t = \tanh(z_t)$$

$$\hat{y}_t = h_t\,\theta_{ho}^T + b_o$$

  - Chamamos de z_t a **pré-ativação** (a soma antes do tanh); ela reaparece no *backward pass*.

- θ_{ih}: pesos da entrada para o estado interno (*input-to-hidden*, `weight_ih`).
- θ_{hh}: pesos do estado interno anterior para o novo estado interno (*hidden-to-hidden*, `weight_hh`).
- θ_{ho}: pesos do estado interno para a saída (camada `Linear` adicional, fora do `nn.RNN`).
- b_h: *bias* do estado interno; b_o: *bias* da saída.

  - Mesma notação do `nn.RNN` do PyTorch (que usa W no lugar de θ). O PyTorch mantém **dois** *bias* no estado interno, b_{ih} e b_{hh}; aqui os combinamos em um único b_h, pois só a soma importa.

Visão Geral

Backpropagation Through Time (BPTT)

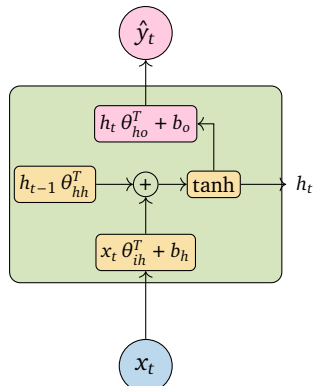
- O algoritmo de treinamento é o mesmo já conhecido: *backpropagation*.
- Porém agora precisamos considerar múltiplos passos de tempo (*timesteps*).
  - O mesmo peso influencia a saída de várias formas diferentes, conforme o *timestep*.
  - As contribuições do gradiente são **somadas** ao longo da sequência.
- Esse algoritmo é chamado de *Backpropagation Through Time*.

Setup do Exemplo

Vamos usar a rede recorrente mais simples possível para um exemplo numérico:

- $x = [1, 2]$
- $y = 3$
- $h_0 = 0$
- $\theta_{ih} = 0.5, \quad \theta_{hh} = 0.5$
- $\theta_{ho} = 1$
- $b_h = 0, \quad b_o = 0$

Para simplificar a aritmética, usamos dimensão 1: cada x_t e h_t é um escalar, e $x = [1, 2]$ são dois passos de tempo.

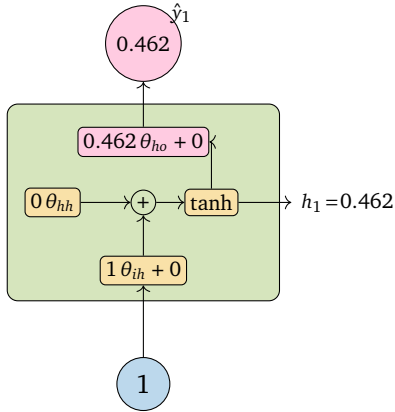


Forward Pass, Passo 1

- No passo 1, $x_1 = 1$ e $h_0 = 0$:

$$h_1 = \tanh(1 \cdot \theta_{ih} + 0 \cdot \theta_{hh} + b_h) = \tanh(0.5) \approx 0.462$$

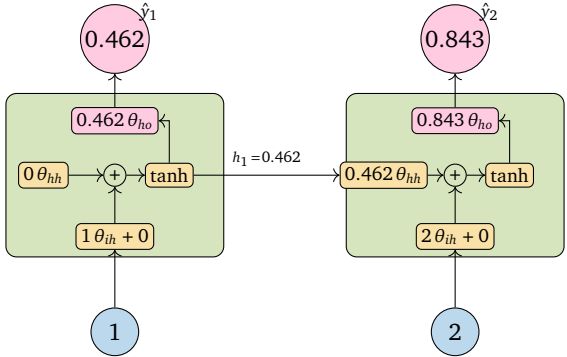
$$\hat{y}_1 = h_1 \cdot \theta_{ho} + b_o = 0.462 \cdot 1 + 0 = 0.462$$



Forward Pass, Passo 2

- No passo 2, $x_2 = 2$ e $h_1 \approx 0.462$:

$$h_2 = \tanh(2 \cdot \theta_{ih} + 0.462 \cdot \theta_{hh} + b_h) = \tanh(1.231) \approx 0.843$$
$$\hat{y}_2 = h_2 \cdot \theta_{ho} + b_o = 0.843 \cdot 1 + 0 = 0.843$$



Cálculo da *Loss*

- Após o *forward*, comparamos $\hat{y}_2$ com a meta $y = 3$ usando metade do erro quadrático:

$$J(\theta) = \frac{1}{2}\left(\hat{y}_2 - y\right)^2 = \frac{1}{2}(0.843 - 3)^2 \approx 2.326$$

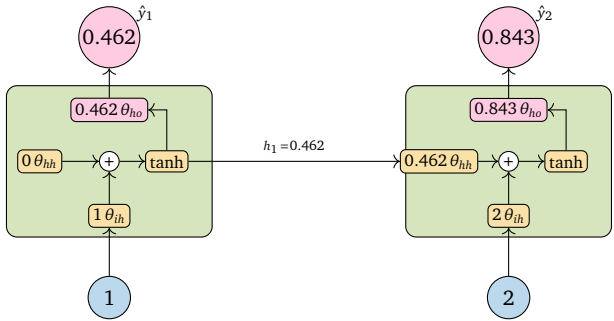
- O fator $\frac{1}{2}$ apenas cancela o expoente ao derivar, semeando o *backward pass*:

$$\frac{\partial J}{\partial \hat{y}_2} = (\hat{y}_2 - y)$$

- A *loss* no instante final pode ser usada sozinha (*many-to-one*) ou somada ao longo de todos os instantes (*many-to-many*).

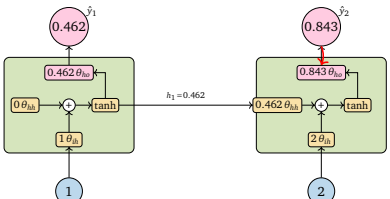
Backward Pass: O Grafo Computacional

- O *forward* desdobrado no tempo forma um **grafo computacional**: a *loss* $J = \frac{1}{2}(\hat{y}_2 - y)^2$ depende de $\hat{y}_2$, que depende de h_2, que depende de h_1, e assim por diante.



- Para treinar, vamos percorrer esse grafo **de trás para frente**, derivando o gradiente de J em relação a cada parâmetro, um de cada vez.

Backward Pass: Pesos da Saída

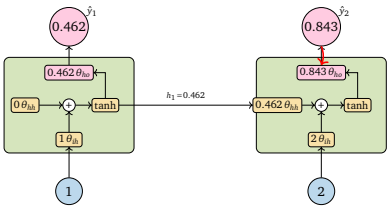


- Começamos pelos pesos da saída: o gradiente chega em $\hat{y}_2$ e atinge a cabeça de saída (em vermelho).

$$\frac{\partial J}{\partial \theta_{ho}} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \theta_{ho}} = (\hat{y} - y)\, h_2$$
$$= (0.843 - 3) \cdot 0.843 \approx -1.818$$

$$\frac{\partial J}{\partial b_o} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial b_o} = (\hat{y} - y) \cdot 1$$
$$= (0.843 - 3) \cdot 1 \approx -2.157$$

Backward Pass: Pesos da Saída

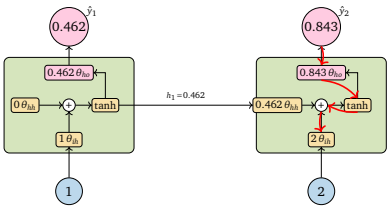


- Começamos pelos pesos da saída: o gradiente chega em $\hat{y}_2$ e atinge a cabeça de saída (em vermelho).

$$\frac{\partial J}{\partial \theta_{ho}} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \theta_{ho}} = (\hat{y} - y)\, h_2$$
$$= (0.843 - 3) \cdot 0.843 \approx -1.818$$

$$\frac{\partial J}{\partial b_o} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial b_o} = (\hat{y} - y) \cdot 1$$
$$= (0.843 - 3) \cdot 1 \approx -2.157$$

Backward Pass: Pesos da Entrada



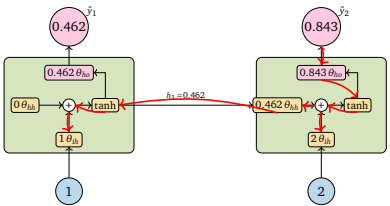
- Para os pesos θ_{ih}, o gradiente **atravessa todos os passos de tempo**.

$$\frac{\partial J}{\partial \theta_{ih}} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial h_2} \frac{\partial h_2}{\partial z_2} \left(\frac{\partial z_2}{\partial \theta_{ih}} + \underbrace{\frac{\partial z_2}{\partial h_1} \frac{\partial h_1}{\partial z_1} \frac{\partial z_1}{\partial \theta_{ih}}}_{\text{atravessa o tempo}} \right)$$

$$\frac{\partial J}{\partial \theta_{ih}} = (\hat{y} - y)\, \theta_{ho}\, (1 - \tanh^2(z_2))(x_2 + \theta_{hh}(1 - \tanh^2(z_1))\, x_1)$$

$$\frac{\partial J}{\partial \theta_{ih}} = (0.843 - 3) \cdot 1 \cdot (1 - 0.843^2)(2 + 0.5(1 - 0.462^2) \cdot 1) \approx -1.493$$

Backward Pass: Pesos da Entrada



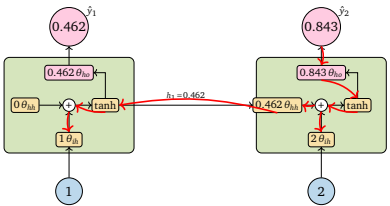
- Para os pesos θ_{ih}, o gradiente **atravessa todos os passos de tempo**.

$$\frac{\partial J}{\partial \theta_{ih}} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial h_2} \frac{\partial h_2}{\partial z_2} \left(\frac{\partial z_2}{\partial \theta_{ih}} + \underbrace{\frac{\partial z_2}{\partial h_1} \frac{\partial h_1}{\partial z_1} \frac{\partial z_1}{\partial \theta_{ih}}}_{\text{atravessa o tempo}} \right)$$

$$\frac{\partial J}{\partial \theta_{ih}} = (\hat{y} - y)\, \theta_{ho}\, (1 - \tanh^2(z_2))\, (x_2 + \theta_{hh} (1 - \tanh^2(z_1))\, x_1)$$

$$\frac{\partial J}{\partial \theta_{ih}} = (0.843 - 3) \cdot 1 \cdot (1 - 0.843^2)(2 + 0.5(1 - 0.462^2) \cdot 1) \approx -1.493$$

Backward Pass: Pesos da Entrada



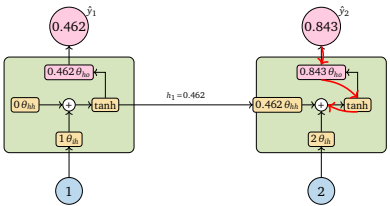
- Para os pesos θ_{ih}, o gradiente **atravessa todos os passos de tempo**.

$$\frac{\partial J}{\partial \theta_{ih}} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial h_2} \frac{\partial h_2}{\partial z_2} \left(\frac{\partial z_2}{\partial \theta_{ih}} + \underbrace{\frac{\partial z_2}{\partial h_1} \frac{\partial h_1}{\partial z_1} \frac{\partial z_1}{\partial \theta_{ih}}}_{\text{atravessa o tempo}} \right)$$

$$\frac{\partial J}{\partial \theta_{ih}} = (\hat{y} - y)\, \theta_{ho}\, (1 - \tanh^2(z_2))\, (x_2 + \theta_{hh}(1 - \tanh^2(z_1))\, x_1)$$

$$\frac{\partial J}{\partial \theta_{ih}} = (0.843 - 3) \cdot 1 \cdot (1 - 0.843^2)(2 + 0.5(1 - 0.462^2) \cdot 1) \approx -1.493$$

Backward Pass: *bias* e Pesos Recorrentes



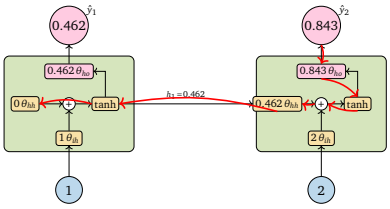
- A mesma ideia vale para o *bias* b_h e para os pesos recorrentes θ_{hh}.

$$\begin{aligned}\frac{\partial J}{\partial b_h} &= (\hat{y} - y)\,\theta_{ho}\,(1 - \tanh^2(z_2))\big(1 + \theta_{hh}(1 - \tanh^2(z_1))\big) \\ &\approx (0.843 - 3) \cdot 1 \cdot (1 - 0.843^2)\big(1 + 0.5(1 - 0.462^2)\big) \\ &\approx -0.870\end{aligned}$$

$$\begin{aligned}\frac{\partial J}{\partial \theta_{hh}} &= (\hat{y} - y)\,\theta_{ho}\,(1 - \tanh^2(z_2))\big(h_1 + \theta_{hh}(1 - \tanh^2(z_1))\,h_0\big) \\ &\approx (0.843 - 3) \cdot 1 \cdot (1 - 0.843^2)\big(0.462 + 0.5(1 - 0.462^2) \cdot 0\big) \\ &\approx -0.288\end{aligned}$$

- Como $h_0 = 0$, o segundo termo de $\partial J / \partial \theta_{hh}$ se anula.

Backward Pass: *bias* e Pesos Recorrentes



- A mesma ideia vale para o *bias* b_h e para os pesos recorrentes θ_{hh}.

$$\begin{aligned}\frac{\partial J}{\partial b_h} &= (\hat{y} - y)\,\theta_{ho}\,(1 - \tanh^2(z_2))\big(1 + \theta_{hh}(1 - \tanh^2(z_1))\big)\\ &\approx (0.843 - 3)\cdot 1 \cdot (1 - 0.843^2)\big(1 + 0.5(1 - 0.462^2)\big)\\ &\approx -0.870\end{aligned}$$

$$\begin{aligned}\frac{\partial J}{\partial \theta_{hh}} &= (\hat{y} - y)\,\theta_{ho}\,(1 - \tanh^2(z_2))\big(h_1 + \theta_{hh}(1 - \tanh^2(z_1))\,h_0\big)\\ &\approx (0.843 - 3)\cdot 1 \cdot (1 - 0.843^2)\big(0.462 + 0.5(1 - 0.462^2)\cdot 0\big)\\ &\approx -0.288\end{aligned}$$

- Como $h_0 = 0$, o segundo termo de $\partial J/\partial \theta_{hh}$ se anula.

Resumo dos Gradientes

Parâmetro	$\partial J/\partial \cdot$
θ_{ho}	-1.818
b_o	-2.157
θ_{ih}	-1.493
b_h	-0.870
θ_{hh}	-0.288

- Repare que o gradiente do peso recorrente θ_{hh} depende dos estados internos passados.
- Em sequências longas, esse caminho de derivadas multiplicadas é a origem dos problemas de *vanishing* e *exploding gradient*.

Passo do *Optimizer*

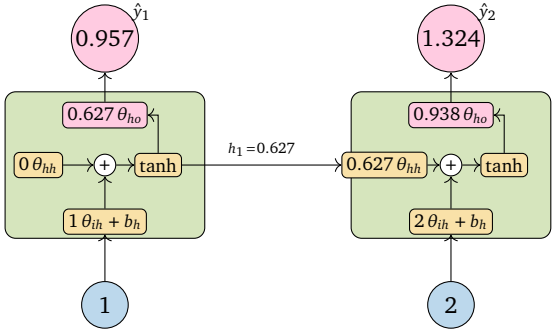
- Com taxa de aprendizado $\eta = 0.1$ e os gradientes calculados:

$$\theta_{ho} \leftarrow \theta_{ho} - \eta \frac{\partial J}{\partial \theta_{ho}} = 1 - 0.1 \cdot (-1.818) = 1.1818$$
$$b_o \leftarrow 0 - 0.1 \cdot (-2.157) = 0.2157$$
$$\theta_{ih} \leftarrow 0.5 - 0.1 \cdot (-1.493) = 0.6493$$
$$b_h \leftarrow 0 - 0.1 \cdot (-0.870) = 0.087$$
$$\theta_{hh} \leftarrow 0.5 - 0.1 \cdot (-0.288) = 0.5288$$

- Com os pesos atualizados, refazemos o *forward* para verificar o efeito na *loss*.

Resultado: a *Loss* Diminuiu

- Refazendo o *forward* com os pesos atualizados ($\theta_{ih} = 0.6493$, $\theta_{hh} = 0.5288$, $b_h = 0.087$, $\theta_{ho} = 1.1818$, $b_o = 0.2157$):



$$J_{\text{antes}} = \frac{1}{2}(0.843 - 3)^2 \approx 2.326 \quad \longrightarrow \quad J_{\text{depois}} = \frac{1}{2}(1.324 - 3)^2 \approx 1.404$$

- Uma única iteração de *BPTT* mais *optimizer* já reduziu a *loss*.

Implementação em PyTorch

```
1  class RNNSimples(torch.nn.Module):
2      def __init__(self) -> None:
3          super().__init__()
4          self.weight_ih = torch.nn.Parameter(torch.tensor(0.5))
5          self.weight_hh = torch.nn.Parameter(torch.tensor(0.5))
6          self.bias_h    = torch.nn.Parameter(torch.tensor(0.0))
7          self.weight_ho = torch.nn.Parameter(torch.tensor(1.0))
8          self.bias_o    = torch.nn.Parameter(torch.tensor(0.0))
9
10     def forward(self, x, h=torch.tensor(0.0)):
11         h = torch.tanh(x * self.weight_ih + h * self.weight_hh + self.bias_h)
12         y = h * self.weight_ho + self.bias_o
13         return y, h
```

Loop de Treinamento

```
model = RNNSimples()
optimizer = SGD(model.parameters(), lr=0.1)
x = [torch.tensor(1.0), torch.tensor(2.0)]
y = torch.tensor(3.0)

model.train()
optimizer.zero_grad()
h = torch.tensor(0.0)
for x_i in x:
    y_hat, h = model(x_i, h)
loss = (y - y_hat)**2 / 2
loss.backward()
optimizer.step()
```

- O `loss.backward()` acumula o gradiente de cada peso ao longo de todos os passos de tempo automaticamente.

Visão Geral

Problemas: *Vanishing* e *Exploding Gradient*

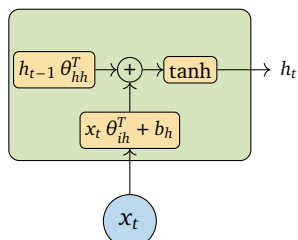
- Os velhos conhecidos: *vanishing* e *exploding gradient*.
- Com RNNs ficam piores, pois a rede também é **profunda no tempo**.
  - Em sequências longas, o gradiente se multiplica por $\theta_{hh}\,(1 - \tanh^2)$ a cada passo.
  - Após T passos, o produto de Jacobianos toma a forma:

$$\prod_{k=1}^{T} \theta_{hh}\left(1 - \tanh^2(z_k)\right)$$

- Se cada fator é menor que 1, o produto decai exponencialmente em T (*vanishing*); se é maior que 1, cresce exponencialmente (*exploding*).
- Soluções comuns:
  - *Gradient clipping by norm*, com limiar c: se $\|g\| > c$, então $g \leftarrow c \cdot g/\|g\|$. Preserva a direção do gradiente. Variantes *clipping by value* distorcem a direção e são menos usadas.
  - Arquiteturas com portas: LSTM e GRU.

Uma Observação sobre PyTorch

- É comum que *frameworks* como o PyTorch **não** incluam a camada de saída $\hat{y}$ diretamente no bloco RNN.
- Cabe ao desenvolvedor adicionar uma camada densa (`nn.Linear`) sobre o estado interno h_t.
- Isso dá flexibilidade para combinar a célula recorrente com diferentes camadas de saída (regressão, classificação, sequência).



Leituras Recomendadas

- Capítulo 10 de Ian Goodfellow, Yoshua Bengio e Aaron Courville. Deep Learning. MIT Press, 2016. URL: https://www.deeplearningbook.org/.
- *MIT Introduction to Deep Learning*, Ava Amini, 2023.

Referências I

[1] Ian Goodfellow, Yoshua Bengio e Aaron Courville. Deep Learning. MIT Press, 2016. URL: https://www.deeplearningbook.org/.