

# Redes Neurais Recorrentes

Eduardo Adame

**Redes Neurais**

12 de novembro de 2025



# O Desafio: Sequências de Comprimento Variável



- Com imagens, forçamos uma dimensão de entrada específica
- Não é óbvio como fazer isso com texto ou outras sequências
- **Exemplo:** Classificar tweets como positivos, negativos ou neutros
- Tweets podem ter um número variável de palavras
- **Outros exemplos:**
  - ▶ Análise de séries temporais (preços de ações)
  - ▶ Reconhecimento de fala
  - ▶ Tradução automática
  - ▶ Análise de sequências de DNA

# O Problema da Ordem das Palavras



- Queremos fazer melhor que implementações de “bag of words” (saco de palavras)
- Idealmente, cada palavra é processada ou compreendida no contexto apropriado
- Precisamos ter alguma noção de “**contexto**”
- As palavras devem ser tratadas de forma diferente dependendo do contexto
- Além disso, cada palavra deve atualizar o contexto

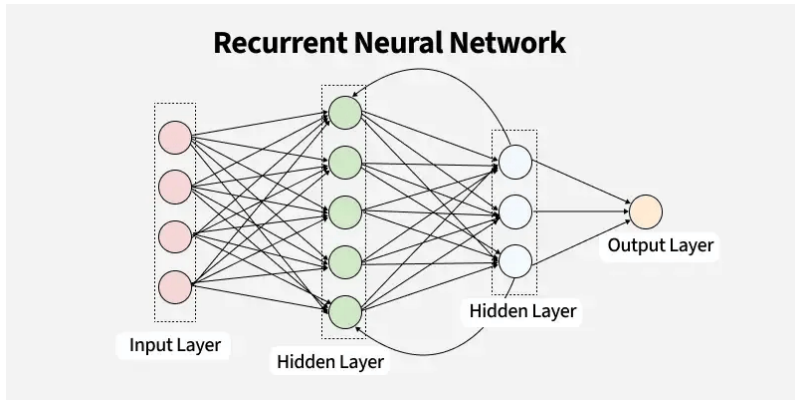
## Exemplo

“O banco estava cheio” vs “O banco do parque estava vazio”  
A palavra “banco” tem significados diferentes dependendo do contexto!

## Ideia: Usar o Conceito de “Recorrência”

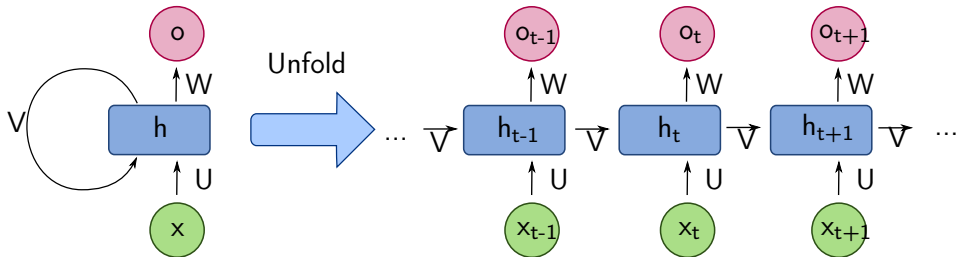


- Processar palavras uma por uma
- A rede produz duas saídas:
  1. **Predição:** Qual seria a predição se a sequência terminasse naquela palavra
  2. **Estado:** Resumo de tudo que aconteceu no passado
- Dessa forma, podemos lidar com comprimentos variáveis de texto
- A resposta a uma palavra depende das palavras que a precederam



O estado é **recorrente** - ele se alimenta de volta para a próxima etapa temporal!

## “Desenrolando” a RNN



Visualização “desenrolada” no tempo ajuda a entender o fluxo de informação



## Elementos:

- $w_i$ : palavra na posição  $i$
- $s_i$ : estado na posição  $i$
- $o_i$ : saída na posição  $i$
- $U$ : pesos de entrada (kernel)
- $W$ : pesos recorrentes
- $V$ : pesos de saída

## Importante:

### Compartilhamento

As matrizes  $U$ ,  $W$ ,  $V$  são as **mesmas** em todas as posições temporais!

Isso permite generalização através do tempo e reduz drasticamente o número de parâmetros.



Considere a frase: “Isso é ótimo!”

Posição	Palavra	Estado	Predição
1	“Isso”	$s_1$	?
2	“é”	$s_2$	?
3	“ótimo”	$s_3$	+
4	“!”	$s_4$	++

- A predição fica mais confiante à medida que mais palavras são processadas
- O estado  $s_i$  carrega informação de todas as palavras anteriores





## Equações Fundamentais

$$s_i = f(Uw_i + Ws_{i-1}) \quad (\text{RNN principal})$$

$$o_i = \text{softmax}(Vs_i) \quad (\text{camada densa subsequente})$$

### Em outras palavras:

- estado atual = função(estado anterior, entrada atual)
- saída atual = função(estado atual)
- Aprendemos as funções através do treinamento da rede!

$f$  é tipicamente uma função de ativação não-linear (tanh ou ReLU)

# Dimensões das Matrizes



- $r$  = dimensão do vetor de entrada
- $s$  = dimensão do estado oculto
- $t$  = dimensão do vetor de saída (após camada densa)

## Dimensões das Matrizes de Peso

- $U$  é uma matriz  $s \times r$
- $W$  é uma matriz  $s \times s$
- $V$  é uma matriz  $t \times s$

**Nota importante:** As matrizes  $U$ ,  $V$ ,  $W$  são as mesmas em todas as posições temporais.



- Matriz  $U$  (kernel): Controlada por `kernel_initializer`
- Matriz  $W$  (recorrente): Controlada por `recurrent_initializer`
- Matriz  $V$  (saída): Implementada como uma camada Dense subsequente

## Listing 1: Exemplo de Código

```
1 model.add(SimpleRNN(units=128,  
2                 kernel_initializer='glorot_uniform',  
3                 recurrent_initializer='orthogonal'))  
4 model.add(Dense(num_classes, activation='softmax'))
```



- Frequentemente, treinamos apenas na saída “final” e ignoramos as saídas intermediárias
- Uma variação chamada **Backpropagation Through Time (BPTT)** é usada para treinar RNNs
- Sensível ao comprimento da sequência (devido ao problema de “gradiente desaparecendo/explodindo”)
- Na prática, ainda definimos um comprimento máximo para nossas sequências:
  - ▶ Se a entrada é mais curta, fazemos **padding** (preenchimento)
  - ▶ Se a entrada é mais longa, truncamos



### Listing 2: Pré-processamento de Sequências

```
1 from tensorflow.keras.preprocessing import sequence
2
3 # Definir comprimento máximo
4 max_length = 100
5
6 # Padding das sequências
7 X_train = sequence.pad_sequences(X_train,
8                                   maxlen=max_length,
9                                   padding='post',
10                                  truncating='post')
```

padding='post': Adiciona zeros no final

truncating='post': Corta no final se exceder max\_length



### Listing 3: Modelo RNN Simples para Classificação de Sentimento

```
1 from tensorflow.keras.models import Sequential
2 from tensorflow.keras.layers import Embedding, SimpleRNN, Dense
3
4 model = Sequential([
5     Embedding(vocab_size, embedding_dim,
6               input_length=max_length),
7     SimpleRNN(64, return_sequences=False),
8     Dense(1, activation='sigmoid')
9 ])
10
11 model.compile(optimizer='adam',
12               loss='binary_crossentropy',
13               metrics=['accuracy'])
```

# Embedding Layer



- Antes de processar palavras, precisamos convertê-las em vetores numéricos
- **Embedding** aprende uma representação densa de palavras
- Cada palavra é mapeada para um vetor de tamanho fixo
- Palavras semanticamente similares terão embeddings próximos

## Exemplo

Palavra	Embedding (simplificado)
"bom"	[0.8, 0.3, -0.1, ...]
"ótimo"	[0.85, 0.35, -0.05, ...]
"ruim"	[-0.7, -0.4, 0.2, ...]



## Processamento de Linguagem Natural:

- Análise de sentimento
- Classificação de textos
- Geração de texto (chatbots básicos)
- Modelagem de linguagem

## Séries Temporais:

- Previsão de preços de ações
- Previsão de demanda de energia
- Análise de dados climáticos
- Monitoramento de sensores IoT



# Mais Aplicações de RNNs



## Reconhecimento de Fala:

- Transcrição de áudio para texto
- Comandos de voz

## Bioinformática:

- Análise de sequências de DNA
- Predição de estrutura de proteínas
- Classificação de sequências genômicas

## Outras Aplicações:

- Reconhecimento de escrita manuscrita
- Análise de vídeo (frame por frame)
- Detecção de anomalias em séries temporais



## TensorFlow Playground - RNN Visualizer

**URL:** <https://distill.pub/2019/memorization-in-rnns/>

**Este site interativo permite:**

- Visualizar como RNNs processam sequências
- Ver como o estado oculto evolui ao longo do tempo
- Experimentar com diferentes arquiteturas
- Entender o problema de memória de longo prazo

# Problema do Gradiente Desaparecendo/Explodindo



## Gradiente Desaparecendo:

- Gradientes ficam muito pequenos
- Informação do passado distante é “esquecida”
- Dificuldade em aprender dependências de longo prazo

## Gradiente Explodindo:

- Gradientes ficam muito grandes
- Instabilidade no treinamento
- Pode ser mitigado com gradient clipping

### Exemplo

Tentando lembrar o sujeito de uma frase: *“O homem que ... muitas palavras ... estava cansado”*

RNNs simples têm dificuldade em manter “homem” na memória.

# Limitações das RNNs Simples



- **Memória de curto prazo:** Dificuldade em manter informação do passado distante sem reforço
- **Processamento sequencial:** Não pode ser facilmente paralelizado
- **Sensibilidade ao comprimento:** Performance degrada com sequências muito longas
- **Treinamento lento:** Backpropagation through time é computacionalmente custoso

## Solução

Na próxima aula, veremos **LSTMs (Long Short-Term Memory)**, que foram projetadas especificamente para resolver esses problemas!

# Comparação: RNN vs Outras Arquiteturas (2025)



Arquitetura	Paralelização	Memória	Uso em 2025
RNN Simples	×	Curto prazo	Limitado
LSTM/GRU	×	Longo prazo	Moderado
Transformers	✓	Atenção total	Dominante
CNN 1D	✓	Local	Específico

**Nota:** Embora Transformers dominem em 2025, RNNs ainda são:

- Fundamentais para entender arquiteturas modernas
- Úteis em cenários com recursos limitados
- Base para variantes mais eficientes



# Long Short-Term Memory (LSTM)

### O que veremos:

- Arquitetura LSTM: gates de entrada, saída e esquecimento
- Como LSTMs resolvem o problema do gradiente desaparecendo
- Variante GRU (Gated Recurrent Unit)
- Aplicações práticas: tradução automática, chatbots, geração de texto
- Implementação em TensorFlow/Keras
- Comparação RNN vs LSTM vs GRU



## Leitura Recomendada:

- Goodfellow et al. - Deep Learning, Capítulo 10 (RNNs)
- <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>

## Tutoriais Interativos:

- TensorFlow RNN Tutorial:  
[https://www.tensorflow.org/text/tutorials/text\\_classification\\_rnn](https://www.tensorflow.org/text/tutorials/text_classification_rnn)
- Distill.pub - Visualizações de RNNs

## Vídeos Complementares:

- Stanford CS231n - Lecture 10 (RNNs)
- 3Blue1Brown - Neural Networks series

Obrigado!

Dúvidas?