LSTM + Extras

Eduardo Adame

Redes Neurais

18 de novembro de 2025

Agenda da Aula

- Revisão: Limitações das RNNs Padrão
- Long Short-Term Memory (LSTM)
  - Motivação e Arquitetura
  - Gates: Forget, Input, Output
  - Fluxo de Informação
- Variantes: GRU e outras
- **Extras: Tendências Modernas**
  - Vision Transformers (ViT)
  - Diffusion Models
- Encerramento do Curso

O Problema das RNNs Padrão

Limitação Fundamental

As RNNs tradicionais sofrem do problema de **memória de curto prazo**

- A matriz de transição necessariamente **enfraquece o sinal** ao longo do tempo
- **Vanishing gradients**: informação de passos distantes "desaparece"
- **Exploding gradients**: instabilidade numérica
- Dificuldade em capturar dependências de longo prazo

Precisamos de uma estrutura que possa manter algumas dimensões inalteradas por muitos passos!

LSTM: Tornando "Lembrar" Mais Fácil

Ideia Central

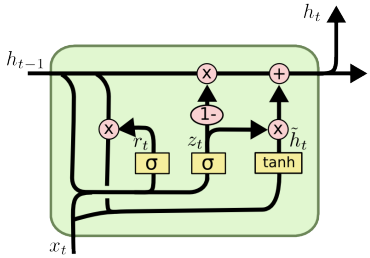
Criar um mecanismo mais sofisticado de atualização do estado interno

Princípio de Design:

- Por padrão, LSTMs **lembram** a informação do passo anterior
- Informações são **esquecidas** ou **adicionadas** como uma escolha ativa
- Usa "gates" (portões) para controlar o fluxo de informação

Publicado por Hochreiter & Schmidhuber (1997)
Revolucionou o processamento de sequências

Arquitetura LSTM: Visão Geral



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Componentes principais:

- h_t: hidden state (estado oculto) - saída da célula
- C_t: cell state (estado da célula) - "memória" de longo prazo
- x_t: entrada no tempo t
- Três gates (portões) controlam o fluxo

Referência: `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`

Gate 1: Forget Gate (Portão de Esquecimento)

Função

Decide quais informações do estado anterior devem ser descartadas

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

- σ: função sigmoid (saída entre 0 e 1)
- $f_t = 0$: "esqueça tudo"
- $f_t = 1$: "mantenha tudo"
- Aprende baseado em h_{t-1} (contexto anterior) e x_t (entrada atual)

Exemplo: Em processamento de texto, ao encontrar um novo sujeito, pode esquecer o gênero do sujeito anterior.

Gate 2: Input Gate (Portão de Entrada)

Função

Decide quais novas informações serão armazenadas no estado da célula

Duas operações:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \quad \text{(quanto adicionar)}$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \quad \text{(o que adicionar)}$$

- i_t: sigmoid determina quais valores atualizar (0 a 1)
- $\tilde{C}_t$: tanh cria vetor de candidatos (-1 a 1)
- Multiplicação elemento-a-elemento combina ambos

Atualização do Cell State

Combinação: Esquecer + Adicionar

O novo estado da célula combina informação antiga filtrada e nova informação

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$

Notação: $\odot$ representa multiplicação elemento-a-elemento (Hadamard)

- $f_t \odot C_{t-1}$: mantém partes do estado antigo
- $i_t \odot \tilde{C}_t$: adiciona novas informações
- Operação linear! Facilita propagação de gradientes

Esta é a "rodovia de informação" que resolve o problema de vanishing gradients

Gate 3: Output Gate (Portão de Saída)

Função

Decide qual parte do estado da célula será exposta como saída

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$
$$h_t = o_t \odot \tanh(C_t)$$

- o_t: controla quanto do cell state mostrar
- $\tanh(C_t)$: normaliza valores entre -1 e 1
- h_t: hidden state, usado como saída e feed para próximo passo

Nota: Não há pesos na aplicação do $\tanh(C_t)$, apenas no gate o_t

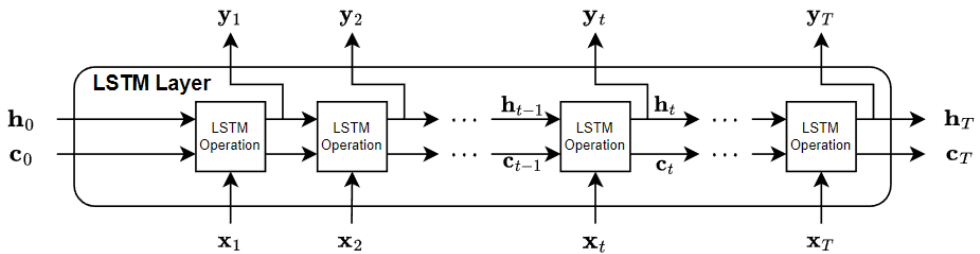
LSTM: Todas as Equações

Sistema Completo

$$
\begin{aligned}
f_t &= \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \quad \text{(forget gate)} \\
i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \quad \text{(input gate)} \\
\tilde{C}_t &= \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \quad \text{(candidate values)} \\
C_t &= f_t \odot C_{t-1} + i_t \odot \tilde{C}_t \quad \text{(cell state)} \\
o_t &= \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \quad \text{(output gate)} \\
h_t &= o_t \odot \tanh(C_t) \quad \text{(hidden state)}
\end{aligned}
$$

Parâmetros treináveis: $W_f, W_i, W_C, W_o, b_f, b_i, b_C, b_o$

LSTM Desenrolado no Tempo



- Mesmos pesos compartilhados em todos os passos de tempo
- Cell state C_t flui horizontalmente com modificações mínimas
- Hidden state h_t conecta verticalmente às saídas
- Backpropagation through time (BPTT) para treinamento

Variantes de LSTM

LSTM não é único! Várias arquiteturas foram propostas:

- **GRU (Gated Recurrent Unit)** - Cho et al., 2014
  - Simplificação do LSTM: combina forget e input gates
  - Menos parâmetros, mais rápido
  - Performance comparável em muitas tarefas
- **Peephole Connections** - Gers & Schmidhuber, 2000
  - Gates podem "espiar" o cell state
- **Coupled Gates**
  - Forget e input gates acoplados: $i_t = 1 - f_t$

GRU tornou-se muito popular pela eficiência!

GRU: Gated Recurrent Unit

Arquitetura simplificada com apenas 2 gates:

$$
\begin{aligned}
z_t &= \sigma(W_z \cdot [h_{t-1}, x_t]) \quad \text{(update gate)} \\
r_t &= \sigma(W_r \cdot [h_{t-1}, x_t]) \quad \text{(reset gate)} \\
\tilde{h}_t &= \tanh(W \cdot [r_t \odot h_{t-1}, x_t]) \\
h_t &= (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t
\end{aligned}
$$

Diferenças do LSTM:

- Não há cell state separado
- Menos parâmetros ($\sim 25\%$ menos)
- Computação mais rápida
- Performance similar em muitas aplicações

LSTM vs GRU: Quando Usar?

Aspecto	LSTM	GRU
Parâmetros	Mais	Menos
Velocidade	Mais lento	Mais rápido
Memória	Maior	Menor
Performance	Ligeiramente melhor*	Comparável
Sequências longas	Melhor	Bom

Recomendação prática:

- Comece com GRU (mais rápido para experimentar)
- Use LSTM se precisar de controle mais fino
- Teste ambos no seu problema específico
- *Depende muito da tarefa!

Aplicações de LSTMs (2015-2020)

LSTMs dominaram processamento sequencial até recentemente:

- **Processamento de Linguagem Natural**
  - Tradução automática (seq2seq)
  - Análise de sentimento
  - Reconhecimento de entidades nomeadas
- **Séries Temporais**
  - Previsão de demanda
  - Análise financeira
  - Previsão climática
- **Outros Domínios**
  - Reconhecimento de fala
  - Geração de texto
  - Análise de vídeo
  - Geração de música

Extra 1: Vision Transformers (ViT)

A Revolução dos Transformers

"Attention is All You Need" - Vaswani et al., 2017

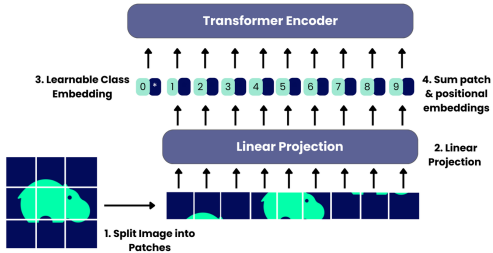
Transformers vs RNN/LSTM:

- Não processam sequencialmente - **paralelizável**!
- Usam mecanismo de **self-attention**
- Eliminaram domínio das RNNs em NLP (2017-2019)

Vision Transformers (Dosovitskiy et al., 2020):

- Aplicam Transformers a imagens
- Dividem imagem em patches (pedaços)
- Tratam patches como "tokens" de uma sequência
- Superaram CNNs em muitas tarefas (com dados suficientes)

Vision Transformer: Arquitetura Básica



Pipeline:

1. Divide imagem em patches fixos (ex: 16×16 pixels)
2. Lineariza cada patch em vetor
3. Adiciona position embeddings
4. Passa pela arquitetura Transformer
5. Classification head na saída

Vantagem chave: Atenção global desde a primeira camada (vs. campo receptivo local de CNNs)

Vision Transformers em 2025

Estado atual (tendências):

- **Modelos Híbridos:** Combinam CNNs e Transformers
  - ConvNeXt, CoAtNet, etc.
- **Eficiência:** ViTs leves para dispositivos móveis
  - MobileViT, EfficientViT
- **Foundation Models:**
  - CLIP (OpenAI): visão + linguagem
  - DINOv2 (Meta): self-supervised learning
  - SAM (Segment Anything Model)
- **Aplicações:** Detecção de objetos, segmentação, geração de imagens

ViTs são componente fundamental dos modelos multimodais modernos!

Extra 2: Diffusion Models

Geração de Imagens de Alta Qualidade

Modelos de difusão revolucionaram geração de imagens (2020-2025)

Exemplos famosos:

- DALL-E 2, DALL-E 3 (OpenAI)
- Stable Diffusion (Stability AI)
- Midjourney
- Imagen (Google)

Por que são importantes?

- Qualidade superior a GANs em muitos casos
- Mais estáveis de treinar
- Permitem controle detalhado (text-to-image)

Diffusion Models: Conceito Básico

Processo em duas fases:

1. **Forward process (difusão):**
   - Adiciona ruído gaussiano gradualmente à imagem
   - Em T passos: imagem $\rightarrow$ ruído puro
   - Processo simples e fixo (não aprendido)
2. **Reverse process (denoising):**
   - Aprende a remover ruído passo a passo
   - Ruído puro $\rightarrow$ imagem
   - Usa rede neural (geralmente U-Net)

Treinamento: aprende a "prever o ruído" em cada passo

Diffusion: Matemática Simplificada

Forward process (adicionando ruído):

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I)$$

Reverse process (removendo ruído):

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t))$$

Objetivo de treinamento (simplificado):

$$\mathcal{L} = \mathbb{E}_{t,x_0,\epsilon}\left[\|\epsilon - \epsilon_\theta(x_t, t)\|^2\right]$$

Aprende a prever o ruído ϵ que foi adicionado!

Diffusion Models: Arquitetura

Componentes principais:

- **U-Net com attention:**
  - Encoder-decoder com skip connections
  - Self-attention em múltiplas escalas
  - Recebe timestep t como input adicional
- **Conditioning (para text-to-image):**
  - Text embeddings via CLIP ou T5
  - Cross-attention entre imagem e texto
  - Permite controle semântico da geração
- **Latent Diffusion (Stable Diffusion):**
  - Trabalha em espaço latente (comprimido)
  - Muito mais eficiente computacionalmente
  - VAE para encoder/decoder

Diffusion Models em 2025

Avanços recentes:

- **Velocidade:** Modelos de poucos passos (vs. 50-1000 originais)
  - LCM (Latent Consistency Models)
  - SDXL Turbo
- **Controle:** ControlNet, IP-Adapter
  - Controle fino de pose, edges, depth
- **Vídeo:** Runway Gen-2, Pika, Stable Video Diffusion
- **3D:** Geração de objetos 3D
- **Outras modalidades:** Áudio, música, moléculas

Diffusion tornou-se framework dominante para geração!

Evolução: RNNs → Transformers → Diffusion

Aspecto	RNN/LSTM	Transformers	Diffusion
Ano	1997-2015	2017+	2020+
Paralelização	Não	Sim	Parcial
Domínio	Sequências	Múltiplo	Geração
Uso hoje	Menor	Dominante	Crescente

Mensagem principal:

- LSTMs foram fundamentais, mas Transformers dominam hoje
- Cada arquitetura resolveu problemas específicos
- Evolução continua: novos modelos surgem constantemente
- Fundamentos (gradientes, otimização) permanecem

Exemplo Interativo: LSTM Playground

Recomendação de demonstração online:

TensorFlow Playground - RNN Extension

`https://distill.pub/2019/memorization-in-rnns/`

Outras opções:

- **LSTMVis:** `http://lstm.seas.harvard.edu/`
  - Visualização de estados internos de LSTM
  - Análise de texto
- **Hugging Face Spaces:**
  - Text generation com GPT (Transformer)
  - Stable Diffusion demos

Experimente você mesmo para entender o comportamento!

Implementação: LSTM em PyTorch

Listing 1: Exemplo básico

```python
import torch.nn as nn
class LSTMModel(nn.Module):
    def __init__(self, input_size, hidden_size,
                 num_layers, output_size):
        super().__init__()
        self.lstm = nn.LSTM(input_size, hidden_size,
                            num_layers, batch_first=True)
        self.fc = nn.Linear(hidden_size, output_size)
    def forward(self, x):
        # x: (batch, seq_len, input_size)
        lstm_out, (h_n, c_n) = self.lstm(x)
        # Última saída
        out = self.fc(lstm_out[:, -1, :])
        return out
```

Implementação: GRU em PyTorch

Listing 2: Ainda mais simples

```
class GRUModel(nn.Module):
    def __init__(self, input_size, hidden_size,
                 num_layers, output_size):
        super().__init__()
        self.gru = nn.GRU(input_size, hidden_size,
                          num_layers, batch_first=True)
        self.fc = nn.Linear(hidden_size, output_size)
    def forward(self, x):
        gru_out, h_n = self.gru(x)
        out = self.fc(gru_out[:, -1, :])
        return out
```

PyTorch e TensorFlow implementam todas as otimizações!

Dicas Práticas para Usar LSTMs/GRUs

Ao trabalhar com RNNs:

1. **Normalização:** Use BatchNorm ou LayerNorm
2. **Regularização:** Dropout entre camadas
3. **Gradient clipping:** Essencial! (ex: clip à norm 5.0)
4. **Inicialização:** Xavier/He, bias do forget gate = 1
5. **Bidirectional:** Quando não há restrição temporal
6. **Tamanho:** 128-512 hidden units comum
7. **Camadas:** 1-4 layers (mais nem sempre é melhor)

Para produção hoje (2025):

- Considere Transformers primeiro (mais suporte)
- Use LSTMs quando sequências são muito longas
- GRUs quando recursos computacionais são limitados

Panorama de Deep Learning em 2025

Arquiteturas dominantes por domínio:

- **Linguagem:** Transformers (GPT, BERT, Claude, etc.)
- **Visão:** ViT, híbridos CNN-Transformer
- **Geração de Imagens:** Diffusion Models
- **Séries Temporais:** Ainda mix (LSTM, Transformers, N-BEATS)
- **Recomendação:** Deep Learning + métodos clássicos
- **Multimodal:** Transformers com múltiplos encoders

Tendências:

- Foundation models cada vez maiores
- Fine-tuning e transfer learning
- Eficiência: quantização, pruning, distillation

Reflexões sobre o Curso

O que vimos ao longo do semestre:

- **Fundamentos:** Perceptrons, MLPs, backpropagation
- **Otimização:** SGD, Adam, learning rate, regularização
- **CNNs:** Convolução, pooling, arquiteturas clássicas
- **Transfer Learning:** Fine-tuning, feature extraction
- **RNNs e LSTMs:** Processamento sequencial
- **Tópicos avançados:** Attention, Transformers, Diffusion

"A jornada de mil milhas começa com um único passo"

Vocês agora têm a base para explorar qualquer arquitetura moderna!

Recursos para Continuar Aprendendo

Cursos online:

- Fast.ai - Practical Deep Learning
- CS231n (Stanford) - Computer Vision
- CS224n (Stanford) - NLP
- Deep Learning Specialization (Coursera)

Leitura:

- Deep Learning (Goodfellow, Bengio, Courville)
- Papers with Code: `https://paperswithcode.com/`
- Distill.pub - Visualizações interativas

Prática:

- Kaggle competitions
- Projetos pessoais
- Contribuir para open source

Próximos Passos Sugeridos

Após este curso, você pode:

1. **Especializar-se:**
   - Computer Vision (detecção, segmentação, tracking)
   - NLP (LLMs, RAG, agents)
   - Generative AI (GANs, Diffusion, VAEs)
   - Reinforcement Learning
2. **Aplicar:**
   - Projetos de pesquisa
   - Competições (Kaggle, etc.)
   - Startup/empresa
3. **Aprofundar teoria:**
   - Otimização matemática
   - Teoria da informação
   - Probabilidade e estatística bayesiana

O campo de Deep Learning está em constante evolução

O que vocês aprenderam aqui são os fundamentos que permanecem, mesmo quando surgem novas arquiteturas

Backpropagation, gradientes, otimização, regularização, avaliação...

Esses conceitos são eternos!

Continue aprendendo, experimentando, e questionando. O melhor ainda está por vir!

Obrigado!

Dúvidas?

Espero que tenham gostado do curso, foi um prazer!
Contem comigo sempre que precisarem, estarei na torcida.

Eduardo Adame, 2025

Referências Principais

Papers Fundamentais:

- Hochreiter & Schmidhuber (1997). Long Short-Term Memory. Neural Computation.
- Cho et al. (2014). Learning Phrase Representations using RNN Encoder-Decoder.
- Vaswani et al. (2017). Attention is All You Need. NeurIPS.
- Dosovitskiy et al. (2020). An Image is Worth 16x16 Words: Transformers for Image Recognition.
- Ho et al. (2020). Denoising Diffusion Probabilistic Models. NeurIPS.
- Rombach et al. (2022). High-Resolution Image Synthesis with Latent Diffusion Models. CVPR.