Deep Learning

Aplicações: Modelagem de Linguagem, Tradução Automática e Atenção

Lucas Silveira Kupssinskü

Machine Learning Theory and Applications Laboratory
PUCRS

agosto de 2026

Overview

1. Modelagem de Linguagem
2. LM com Redes Neurais
3. Decodificação
4. Avaliação
5. seq2seq
6. Atenção

Visão Geral

O Que é Modelar?

- Modelar um fenômeno significa ter alguma capacidade preditiva sobre ele.
- Em *Language Modeling* queremos modelar a linguagem natural, ou seja, ser capazes de **computar a probabilidade de uma frase**.
- Em notação compacta, dada uma sequência de *tokens* $y_1, y_2, \ldots, y_n$, queremos a probabilidade conjunta:

$$P(y_1, y_2, \ldots, y_n)$$

Probabilidade de uma Frase

- Aplicando a regra da cadeia da probabilidade:

$$P(y_1, y_2, \ldots, y_n) = P(y_1) \cdot P(y_2 \mid y_1) \cdots P(y_n \mid y_1, \ldots, y_{n-1})$$
$$= \prod_{t=1}^{n} P(y_t \mid y_{<t})$$

- Modelar a frase inteira se reduz a modelar, a cada passo, a distribuição do **próximo *token*** dado o **contexto** dos *tokens* anteriores.

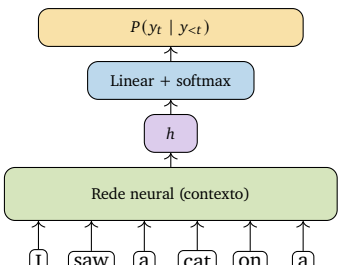
Como Estimar Essas Probabilidades?

- Não temos meios práticos de computar $P(y_t \mid y_{<t})$ por contagem direta em corpus, sequências longas raramente se repetem.
- Diversas abordagens baseadas em *n-grams* foram usadas ao longo dos anos.
  - Aproximam $P(y_t \mid y_{<t})$ por $P(y_t \mid y_{t-n+1}, \ldots, y_{t-1})$.
  - Sofrem com *sparsity* e contexto curto.
- O que tem funcionado melhor nos últimos anos é usar **redes neurais** para estimar essas probabilidades.

Visão Geral

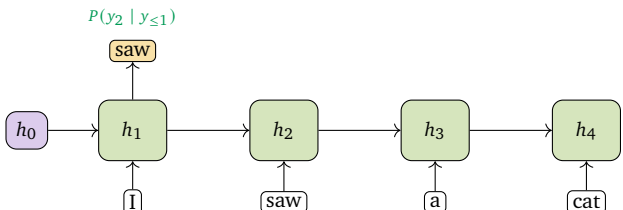
Estrutura Geral

- Modelos de linguagem com NNs costumam ter duas partes:
  - Uma rede que processa o **contexto** $y_{<t}$ produzindo um vetor h.
  - Uma camada de saída que aproxima a distribuição do **próximo *token*** a partir de h.
- Para processar contexto:
  - CNNs (em desuso),
  - RNNs (foco desta aula),
  - Transformers (próximas aulas).



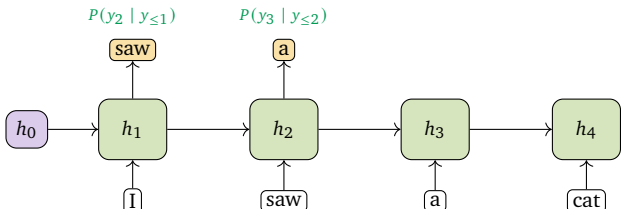
RNN como Modelo de Linguagem

- Numa RNN, o estado oculto h_t resume todo o contexto $y_{<t}$ e é atualizado a cada *token*.
- A cada passo a rede prevê o **próximo *token*** a partir de h_t.



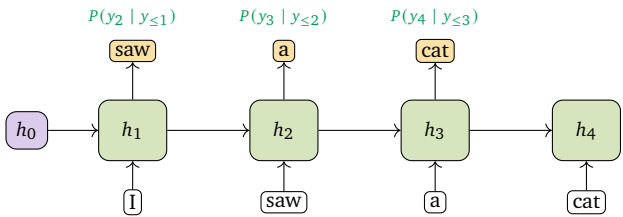
RNN como Modelo de Linguagem

- Numa RNN, o estado oculto h_t resume todo o contexto $y_{<t}$ e é atualizado a cada *token*.
- A cada passo a rede prevê o **próximo *token*** a partir de h_t.



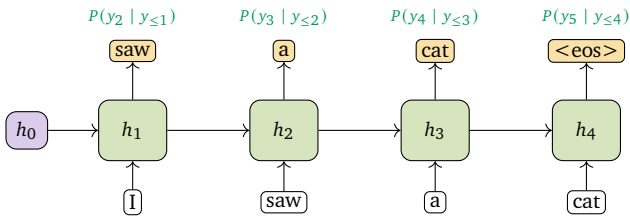
RNN como Modelo de Linguagem

- Numa RNN, o estado oculto h_t resume todo o contexto $y_{<t}$ e é atualizado a cada *token*.
- A cada passo a rede prevê o **próximo *token*** a partir de h_t.



RNN como Modelo de Linguagem

- Numa RNN, o estado oculto h_t resume todo o contexto $y_{<t}$ e é atualizado a cada *token*.
- A cada passo a rede prevê o **próximo *token*** a partir de h_t.



- O mesmo conjunto de pesos é reaplicado a cada passo (*weight sharing*), por isso a sequência pode ter qualquer comprimento.

Camada de Saída

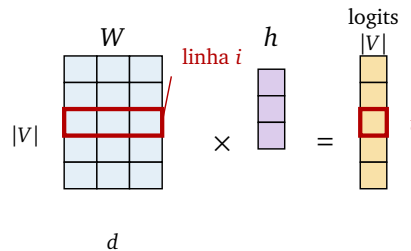
- A partir do vetor de contexto $h \in \mathbb{R}^d$ aplicamos uma transformação para a dimensão do vocabulário $|V|$.

$$\text{logits} = Wh + b, \qquad W \in \mathbb{R}^{|V| \times d}$$

- Em seguida, *softmax* converte os *logits* em probabilidades:

$$P(y_t = w_i \mid y_{<t}) = \frac{\exp(w_i^\top h)}{\sum_{w_j \in V} \exp(w_j^\top h)}$$

- Cada **linha** de W é um *output word embedding*, a representação do *token* de saída.



Treinando como Classificação

- A cada passo de tempo temos um problema de classificação multiclasse com $|V|$ classes.
- Usamos *cross-entropy* como função de custo:

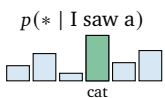
$$\mathcal{L}(y, \hat{y}) = -\sum_{i=1}^{|V|} y_i \ln\big(p(\hat{y}_i)\big) = -\ln\big(p(y_t \mid y_{<t})\big)$$

- O alvo y é um *one-hot* para o *token* correto, de modo que a soma colapsa para o termo do alvo.

Exemplo de Treinamento

Training example: `I saw a` **cat** `on a mat <eos>`

alvo: prever **cat**



- O *backprop* ajusta os pesos para aumentar a probabilidade do *token* correto e reduzir a dos demais.

Exemplo de Treinamento

Training example: `I saw a` **cat** `on a mat <eos>`

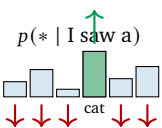
alvo: prever **cat**

- O *backprop* ajusta os pesos para aumentar a probabilidade do *token* correto e reduzir a dos demais.

Exemplo de Treinamento

Training example: `I saw a` **cat** `on a mat <eos>`

alvo: prever **cat**



target

0	0	0	1	0	0

$\mathcal{L} = -\log\big(p(\text{cat})\big)$

aumenta $p(\text{cat})$

diminui os demais

- O *backprop* ajusta os pesos para aumentar a probabilidade do *token* correto e reduzir a dos demais.

Visão Geral

Gerando Frases

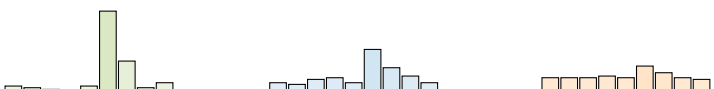
- Com um *LM* treinado podemos **gerar** frases (*decoding*).
- Algumas escolhas no processo de *decoding* são:
  - *Alterar a distribuição* de probabilidade sobre o vocabulário.
  - Usar *estratégias de busca* para escolher os *tokens* a cada passo.
  - Fazer *amostragem* para introduzir diversidade.
- Queremos tanto **coerência** quanto **diversidade** nas frases geradas.
  - Existe um *tradeoff* entre essas características dependendo da forma de gerar os textos.

Temperatura na Softmax

- Para controlar diversidade, dividimos os *logits* por uma temperatura τ:

$$\frac{\exp(w_i^\top h)}{\sum_{w_j \in V} \exp(w_j^\top h)} \quad \longrightarrow \quad \frac{\exp\left(\frac{w_i^\top h}{\tau}\right)}{\sum_{w_j \in V} \exp\left(\frac{w_j^\top h}{\tau}\right)}$$

$\tau = 0.5$ (mais nítida) $\tau = 1$ (*standard*) $\tau = 2$ (mais plana)



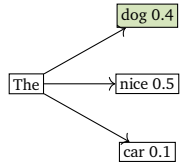
- $\tau \to 0$: tende ao *argmax* (coerência alta, diversidade baixa).
- $\tau \to \infty$: tende a uma distribuição uniforme (diversidade alta, coerência baixa).
- A temperatura só altera a geração quando **amostramos** da distribuição (ver *Sampling*): sob *argmax* ela não muda a escolha.

Greedy Search

- O método mais simples: a cada passo, pegar o *token* mais provável.

$$y_t = \underset{y}{\operatorname{argmax}}\, P(y \mid y_{<t})$$

- Decisão local pode ser ruim globalmente.
- No exemplo ao lado, escolheríamos “The nice” (0.5), perdendo “The dog has” (cuja sequência tem probabilidade conjunta maior).

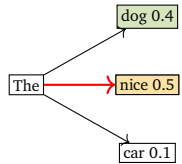


Greedy Search

- O método mais simples: a cada passo, pegar o *token* mais provável.

$$y_t = \underset{y}{\operatorname{argmax}}\, P(y \mid y_{<t})$$

- Decisão local pode ser ruim globalmente.
- No exemplo ao lado, escolheríamos “The nice” (0.5), perdendo “The dog has” (cuja sequência tem probabilidade conjunta maior).

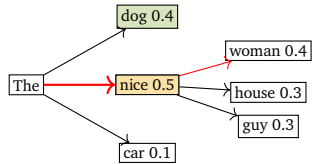


Greedy Search

- O método mais simples: a cada passo, pegar o *token* mais provável.

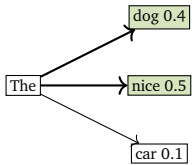
$$y_t = \underset{y}{\operatorname{argmax}}\, P(y \mid y_{<t})$$

- Decisão local pode ser ruim globalmente.
- No exemplo ao lado, escolheríamos “The nice” (0.5), perdendo “The dog has” (cuja sequência tem probabilidade conjunta maior).



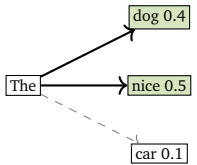
Beam Search

- Mantém os *b beams* mais prováveis a cada passo (*beam size b*). A pontuação de uma sequência é o **produto** das probabilidades condicionais ao longo do caminho (regra da cadeia).



Beam Search

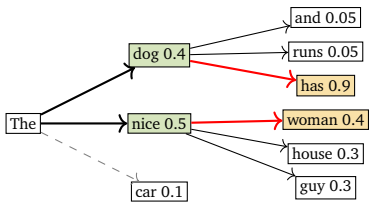
- Mantém os b *beams* mais prováveis a cada passo (*beam size* b). A pontuação de uma sequência é o **produto** das probabilidades condicionais ao longo do caminho (regra da cadeia).
- Com $b = 2$, a partir de "The" guardamos as duas continuações mais prováveis: "dog" (0.4) e "nice" (0.5).



car podado

Beam Search

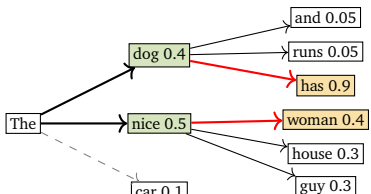
- Mantém os b *beams* mais prováveis a cada passo (*beam size* b). A pontuação de uma sequência é o **produto** das probabilidades condicionais ao longo do caminho (regra da cadeia).
- Com $b = 2$, a partir de "The" guardamos as duas continuações mais prováveis: "dog" (0.4) e "nice" (0.5).



car podado

Beam Search

- Mantém os b *beams* mais prováveis a cada passo (*beam size* b). A pontuação de uma sequência é o **produto** das probabilidades condicionais ao longo do caminho (regra da cadeia).
- Com $b = 2$, a partir de "The" guardamos as duas continuações mais prováveis: "dog" (0.4) e "nice" (0.5).



car podado

Candidatos $b = 2$:

The dog has $0.4 \cdot 0.9 = \mathbf{0.36}$

The nice woman $0.5 \cdot 0.4 = 0.20$

- Embora "The dog" fosse menos provável que "The nice", a expansão revela "The dog has" como melhor sequência.
- Continuamos até encontrar `<eos>` ou um limite de comprimento.

Beam Search: Formulação

- O *Beam Search* busca aproximar:

$$\underset{y}{\operatorname{argmax}} \prod_{t=1}^{n} P(y_t \mid y_{<t})$$

- Por questões de **estabilidade numérica** é usual trabalhar com a soma dos *logs* das probabilidades:

$$\underset{y}{\operatorname{argmax}} \sum_{t=1}^{n} \log\big(P(y_t \mid y_{<t})\big)$$

- Como log é monotonicamente crescente, maximizar p ou $\log p$ é equivalente.

Penalidade por Comprimento

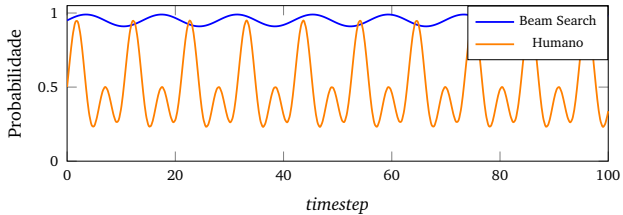
- Sequências mais longas acumulam mais $\log P$ negativos, somando *logs* a busca prefere sequências mais curtas.
- Para amenizar, dividimos pelo tamanho da sequência elevado a um hiperparâmetro $\alpha \in (0, 1)$:

$$\underset{y}{\operatorname{argmax}} \frac{1}{n^{\alpha}} \sum_{t=1}^{n} \log\big(P(y_t \mid y_{<t})\big)$$

- Existem outras normalizações que enviesam a busca para soluções mais longas, mais curtas, ou para maior diversidade.
- Referência prática: `https://opennmt.net/OpenNMT/translation/beam_search/`.

Beam Search vs Texto Humano

- *Beam Search* funciona bem em tarefas onde o tamanho da sequência é relativamente conhecido (tradução, sumarização).
- Em geração aberta, gera saídas **repetitivas** e pouco humanas, sempre escolhe *tokens* de probabilidade alta.



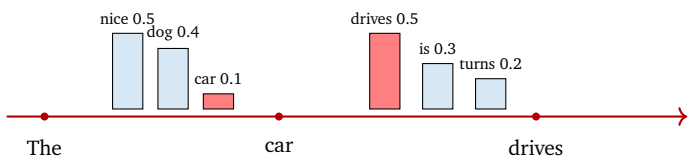
- Ilustração baseada em Holtzman et al., *The Curious Case of Neural Text Degeneration* (2019).

Sampling

- Em vez de pegar o *argmax*, podemos **amostrar** da distribuição de probabilidade:

$$y_t \sim P(y \mid y_{<t})$$

- Geração deixa de ser determinística, mas ganha em diversidade.
- Continuamos até encontrar o *token* especial `<eos>`.



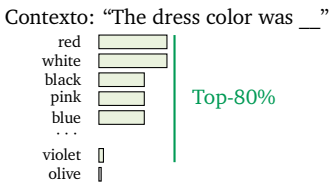
Top-K Sampling

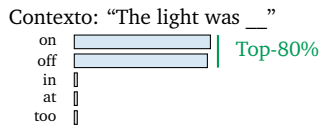
- Heurística aplicada à amostragem:
  - Sempre amostra apenas dos K *tokens* mais prováveis.
  - Evita *tokens* pouco prováveis (que costumam ser ruídos).

- Limitação: o K ideal depende da forma da distribuição.

Top-P Sampling (Nucleus)

- Como fixar um K é difícil, definimos uma massa de probabilidade p.
  - Sempre amostra apenas dos *tokens* mais prováveis cuja probabilidade acumulada chega a p.





- O conjunto sorteado se adapta à distribuição.

Qual o Melhor Decoding?

- Não há consenso.
- *Top-P* é popular para geração aberta (*chatbots*, *storytelling*).
- *Beam Search* continua bastante usado em tarefas com saída bem definida (tradução, sumarização).
- Combinar técnicas (*Beam Search* + *sampling* + temperatura) também é comum.

Visão Geral

Perplexidade

- Métrica mais usada para avaliar *Language Models*, baseada em *log-likelihood*.
- Um bom modelo atribui alta probabilidade a textos reais não vistos e baixa a textos sem sentido.
- *Log-likelihood* da sequência (em bits, com $\log_2$):

$$L(y_{1:m}) = \sum_{t=1}^{m} \log_2\big(p(y_t \mid y_{<t})\big)$$

- Perplexidade:

$$\text{Perplexidade}(y_{1:m}) = 2^{-\frac{1}{m}L(y_{1:m})}, \qquad \text{Perplexidade} \in [1, |V|]$$

- É 2 elevado à *cross-entropy* média por *token* (em bits), por isso varia entre 1 e $|V|$.

Interpretando a Perplexidade

- Bons modelos têm **log-verossimilhança alta** e **perplexidade baixa**.
- Intuição: a perplexidade pode ser lida como o "número médio efetivo de *tokens*" entre os quais o modelo está em dúvida em cada passo.
  - Perplexidade = 1: o modelo é determinístico e está sempre certo.
  - Perplexidade = $|V|$: o modelo é equivalente a uma distribuição uniforme sobre o vocabulário.
- A perplexidade só é **comparável** entre modelos com o **mesmo tamanho de vocabulário** (idealmente o mesmo tokenizador).

Exemplo Numérico: Perplexidade

- Sentença de $m = 4$ *tokens* $y_{1:4}$. Probabilidades atribuídas pelo modelo a cada *token* dado o contexto anterior:

t	1	2	3	4
$p(y_t \mid y_{<t})$	0.5	0.25	0.5	0.5
$\log_2 p(y_t \mid y_{<t})$	-1	-2	-1	-1

- Soma da *log-likelihood*:

$$L = -1 + (-2) + (-1) + (-1) = -5.$$

- Expoente da perplexidade:

$$-\frac{L}{m} = -\frac{-5}{4} = 1.25.$$

- Perplexidade:

$$2^{1.25} \approx 2.378.$$

- Leitura: em média, o modelo se comporta como se estivesse em dúvida entre cerca de 2.4 *tokens* a cada passo.

Visão Geral

Language Models Condicionados

- Em tarefas *seq2seq* (*sequence-to-sequence*), queremos gerar uma sequência y a partir de uma entrada x.
- Adicionamos x como condicionante:

$$P(y_1, y_2, \ldots, y_n \mid x) = \prod_{t=1}^{n} P(y_t \mid y_{<t}, x)$$

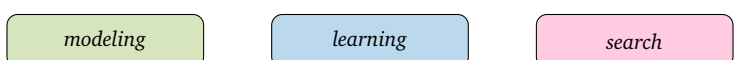
- Na prática, x entra no modelo por meio de um **vetor de contexto** produzido por um *encoder*, que inicializa e alimenta o *decoder* (próximos slides).
- Mesma formulação serve para diversas tarefas:
  - Tradução automática (x é a frase no idioma de origem).
  - *Image captioning* (x é uma imagem).
  - Sumarização (x é um documento longo).
  - Resposta a perguntas (x é a pergunta).

Tradução Automática

- Dada uma sequência em um idioma, queremos descobrir uma sequência em outro idioma.
- Em termos probabilísticos:

$$y' = \underset{y}{\text{argmax}}\ p(y \mid x, \theta)$$

Três perguntas a responder:



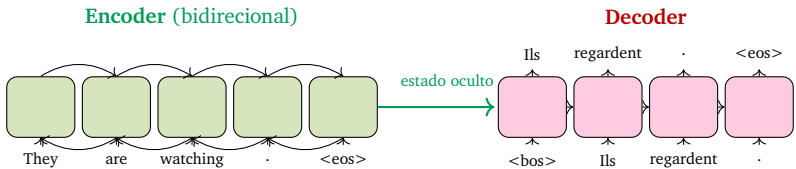
Como é o modelo $p(y \mid x, \theta)$?

Como achar θ?

Como achar o *argmax*?

Arquitetura Encoder-Decoder

- *Encoder-Decoder* é a arquitetura clássica de *seq2seq*.
  - O *encoder* transforma a entrada de tamanho variável em um **estado oculto**.
  - O *decoder* transforma esse estado oculto em uma sequência de saída.
  - O *encoder* costuma ser **bidirecional**: lê a sequência nos dois sentidos, de modo que o estado de cada posição combina contexto à esquerda e à direita.
  - *Tokens* especiais `<bos>` e `<eos>` marcam começo e fim.
- Notação: s_k são os estados do *encoder* e h_t o estado do *decoder* no passo t.

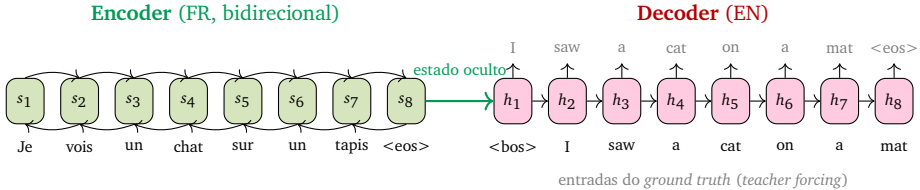


Decoder em Treino vs Teste

- Entrar com os dados no *encoder* é trivial (a sequência de origem).
- Para o *decoder*, há mais de uma forma de alimentar a entrada:
  - **Em treino** (*teacher forcing*): condiciona no estado oculto do *encoder* e nos *tokens* precedentes do *ground truth*.
  - **Em teste**: condiciona no estado oculto do *encoder* e nos *tokens* já preditos pelo próprio modelo.
- Essa diferença entre treino e teste gera o *exposure bias*: o modelo só vê *tokens* corretos no treino, mas no teste pode encontrar *tokens* errados gerados por ele mesmo.
- Uma mitigação possível é o *scheduled sampling*: durante o treino, alterna-se entre o *token* do *ground truth* e o *token* predito pelo próprio modelo.

Treinando uma MT

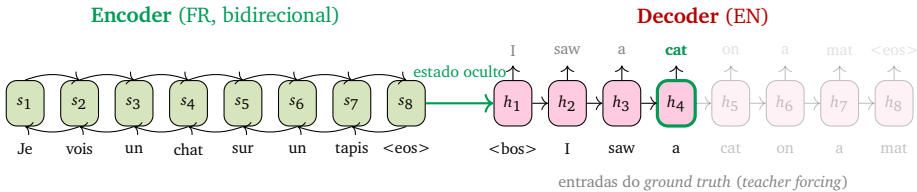
- O treino e a amostragem em tradução automática seguem os mesmos princípios da modelagem de linguagem[1]:
  - *Cross-Entropy Loss* a cada passo do *decoder*.
  - *Beam Search* para gerar a tradução final.
- Exemplo: traduzir `Je vois un chat sur un tapis` (FR) para `I saw a cat on a mat` (EN).
- O *encoder* bidirecional produz os estados ocultos $s_1, \ldots, s_8$ e o *decoder* parte do estado oculto final para gerar a tradução.



[1] Ilya Sutskever, Oriol Vinyals e Quoc V. Le. "Sequence to sequence learning with neural networks". Em: NeurIPS. vol. 27. 2014.

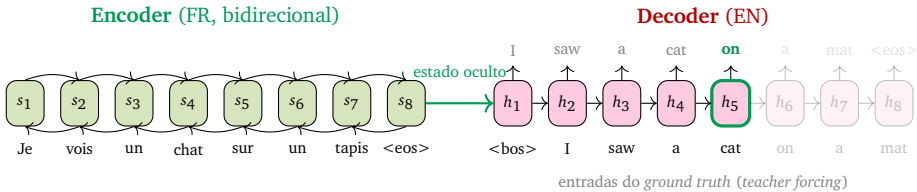
Treinando uma MT: Passo a Passo

- A cada passo, o *decoder* condiciona no estado oculto do *encoder* e no prefixo do *ground truth* para prever o próximo *token*.

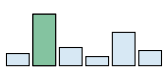


Treinando uma MT: Passo a Passo

- A cada passo, o *decoder* condiciona no estado oculto do *encoder* e no prefixo do *ground truth* para prever o próximo *token*.



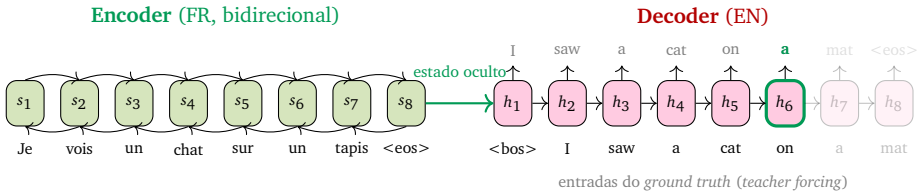
$p(* \mid \text{I saw a cat}, x)$



$\mathcal{L}_5 = -\log(p(\text{on}))$

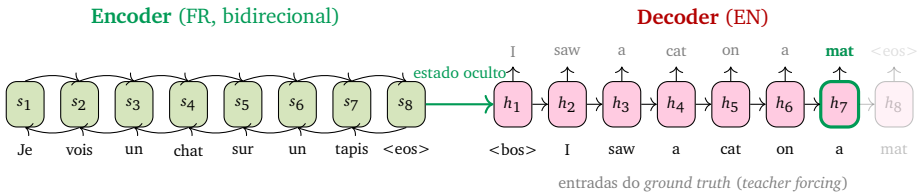
Treinando uma MT: Passo a Passo

- A cada passo, o *decoder* condiciona no estado oculto do *encoder* e no prefixo do *ground truth* para prever o próximo *token*.

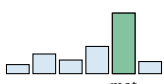


Treinando uma MT: Passo a Passo

- A cada passo, o *decoder* condiciona no estado oculto do *encoder* e no prefixo do *ground truth* para prever o próximo *token*.



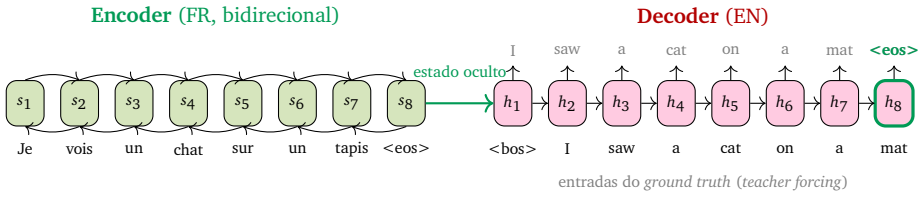
$p(* \mid \text{I saw a cat on a}, x)$



$\mathcal{L}_7 = -\log(p(\text{mat}))$

Treinando uma MT: Passo a Passo

- A cada passo, o *decoder* condiciona no estado oculto do *encoder* e no prefixo do *ground truth* para prever o próximo *token*.



A *loss* da frase soma as *losses* de todos os passos do *decoder*: $\mathcal{L} = \sum_{t=1}^{8} \mathcal{L}_t$.

Avaliação de Tradução: BLEU

- **BLEU** (*Bilingual Evaluation Understudy*) é uma métrica similar a *precisão*, que mede quantos *n-grams* da saída aparecem no texto de referência.
- A **precisão modificada** p_n usa contagens **clipadas**, cada *n-gram* conta no máximo o número de vezes em que aparece na referência:

$$p_n = \frac{\sum_{g \in G_n} \min\big(\mathrm{cont}_{\mathrm{cand}}(g),\ \mathrm{cont}_{\mathrm{ref}}(g)\big)}{\sum_{g \in G_n} \mathrm{cont}_{\mathrm{cand}}(g)}, \qquad G_n = \{\textit{n-grams do candidato}\}.$$

- A **penalidade por brevidade** evita inflar a nota com saídas curtas (c é o tamanho do candidato, r o da referência):

$$\mathrm{BP} = \begin{cases} 1, & c > r \\ e^{1-r/c}, & c \le r \end{cases} \qquad \mathrm{BLEU} = \mathrm{BP} \cdot \exp\Big(\textstyle\sum_{n=1}^{N} \frac{1}{N} \ln p_n\Big).$$

- Tipicamente $N = 4$, combinando unigramas a 4-gramas por média geométrica.

BLEU: Cálculo Passo a Passo

- Referência: `the small cat sat on the mat` ($r = 7$).
- Candidato: `the big cat sat on the mat` ($c = 7$, trocou `small` por `big`).

n	*n-grams* que casam (clipados)	total no candidato	p_n
1	`the`×2, `cat`, `sat`, `on`, `mat` (= 6)	7	$6/7 \approx 0.857$
2	`cat sat`, `sat on`, `on the`, `the mat` (= 4)	6	$4/6 \approx 0.667$
3	`cat sat on`, `sat on the`, `on the mat` (= 3)	5	$3/5 = 0.600$
4	`cat sat on the`, `sat on the mat` (= 2)	4	$2/4 = 0.500$

- Brevidade: como $c = r = 7$, temos $\mathrm{BP} = e^{1-7/7} = e^0 = 1$.
- Média geométrica das precisões (com $w_n = 1/4$):

$$\mathrm{BLEU} = 1 \cdot \exp\Big(\tfrac{1}{4}\big(\ln\tfrac{6}{7} + \ln\tfrac{4}{6} + \ln\tfrac{3}{5} + \ln\tfrac{2}{4}\big)\Big) \approx \exp(-0.441) \approx 0.64.$$

- Se o candidato fosse mais curto (ex: $c = 5$), teríamos $\mathrm{BP} = e^{1-7/5} \approx 0.67$.

Avaliação de Sumarização: ROUGE

- **ROUGE** (*Recall-Oriented Understudy for Gisting Evaluation*) é a métrica mais usada em **sumarização**. Mas também aplicada em tradução, resposta a perguntas, etc.
- Ao contrário do BLEU (precisão), ROUGE é orientada a **recall**: mede quanto da **referência** foi coberto pela saída.
- **ROUGE-N** usa a sobreposição de *n-grams* entre candidato e referência:

$$\text{Recall} = \frac{\textit{n-grams} \text{ em comum}}{\textit{n-grams} \text{ da referência}}, \qquad \text{Precisão} = \frac{\textit{n-grams} \text{ em comum}}{\textit{n-grams} \text{ do candidato}}.$$

- **ROUGE-L** usa a maior subsequência comum (*Longest Common Subsequence*, LCS), capturando a ordem sem exigir que os *tokens* casados sejam contíguos.
- Costuma-se reportar o F_1, combinando precisão e recall:

$$F_1 = \frac{2 \cdot \text{Precisão} \cdot \text{Recall}}{\text{Precisão} + \text{Recall}}.$$

ROUGE: Cálculo Passo a Passo

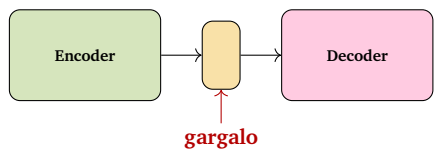
- Referência: `the cat was found under the bed` (7 *tokens*).
- Candidato: `the cat was under the bed` (6 *tokens*, omitiu `found`).

Métrica	em comum	Recall	Precisão	F_1
ROUGE-1	6 unigramas	$6/7 \approx 0.86$	$6/6 = 1$	$12/13 \approx 0.92$
ROUGE-2	4 bigramas	$4/6 \approx 0.67$	$4/5 = 0.8$	$8/11 \approx 0.73$
ROUGE-L	6 (comprimento da LCS)	$6/7 \approx 0.86$	$6/6 = 1$	$12/13 \approx 0.92$

- **ROUGE-1**: unigramas em comum `the`×2, `cat`, `was`, `under`, `bed` = 6; apenas `found` fica de fora.
- **ROUGE-2**: bigramas em comum `the cat`, `cat was`, `under the`, `the bed` = 4 (de 6 na referência e 5 no candidato).
- **ROUGE-L**: a LCS `the cat was under the bed` tem comprimento 6 (pula `found`), preservando a ordem.

O Problema do Gargalo

- O *encoder* comprime toda a frase de origem em **um único vetor**.
- Dois problemas:
  - É difícil para o *encoder* capturar todo o contexto da frase original.
  - É difícil para o *decoder* gerar todos os *tokens* condicionado em um contexto fixo do *encoder*.
- Esse vetor é um **gargalo** de informação.



Visão Geral

Mecanismo de Atenção

- Para resolver o gargalo entre *encoder* e *decoder* foi introduzido o mecanismo de **atenção**[2].
  - Um bloco que conecta *encoder* e *decoder* e atribui *scores* a cada estado do *encoder*.
  - Interpretação: a atenção decide **quais partes da entrada são mais importantes** para gerar o próximo *token*.
- Existem várias implementações, vamos ver a forma de *Bahdanau* aplicada a tradução com RNNs.

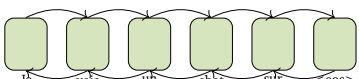
[2] Dzmitry Bahdanau, Kyunghyun Cho e Yoshua Bengio. "Neural machine translation by jointly learning to align and translate". Em: arXiv preprint arXiv:1409.0473 (2014).

Como Funciona a Atenção

- A cada passo t do *decoder*:
  1. Recebe o estado oculto do *decoder* h_t e todos os estados do *encoder* $s_1, \ldots, s_m$.
  2. Computa um *score* $\text{score}(h_t, s_k)$ para cada $k = 1, \ldots, m$.
  3. Aplica *softmax* sobre os *scores* para obter os **pesos de atenção** $a_k^{(t)}$.
  4. Calcula o *context vector* $c^{(t)}$ como soma ponderada dos s_k pelos pesos.
  5. O contexto é passado para o *decoder* para a previsão do próximo *token*.

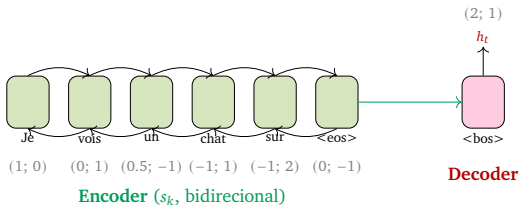
$$a_k^{(t)} = \frac{\exp(\text{score}(h_t, s_k))}{\sum_{i=1}^{m} \exp(\text{score}(h_t, s_i))}, \qquad c^{(t)} = \sum_{k=1}^{m} a_k^{(t)} s_k$$

Atenção: Diagrama

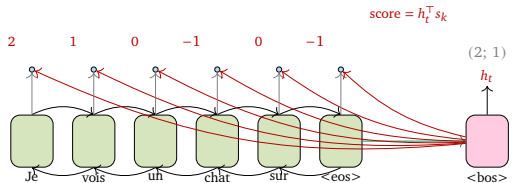


Encoder (s_k, bidirecional)

Atenção: Diagrama



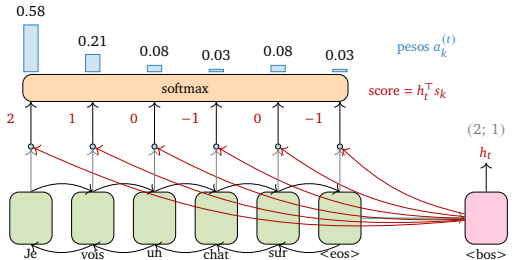
Atenção: Diagrama



Scores: empilhando os s_k como colunas de S, um único produto $h_t^\top S$:

$$(2\ \ 1)\begin{pmatrix}1 & 0 & 0.5 & -1 & -1 & 0\\ 0 & 1 & -1 & 1 & 2 & -1\end{pmatrix} = (2\ \ 1\ \ 0\ \ -1\ \ 0\ \ -1)$$

Atenção: Diagrama

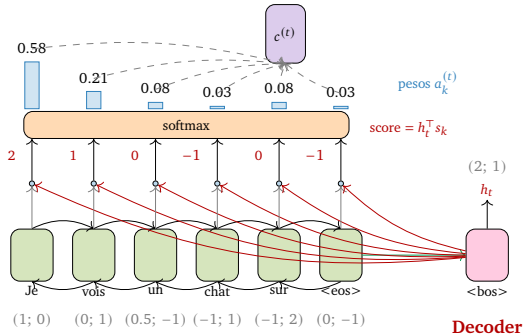


Scores: empilhando os s_k como colunas de S, um único produto $h_t^\top S$:

$$(2\ \ 1)\begin{pmatrix}1 & 0 & 0.5 & -1 & -1 & 0\\ 0 & 1 & -1 & 1 & 2 & -1\end{pmatrix} = (2\ \ 1\ \ 0\ \ -1\ \ 0\ \ -1)$$

Pesos de Atenção: $a^{(t)} = \text{softmax}(\text{scores}_t) = [\ 0.58;\ 0.21;\ 0.08;\ 0.03;\ 0.08;\ 0.03\ ]$

Atenção: Diagrama



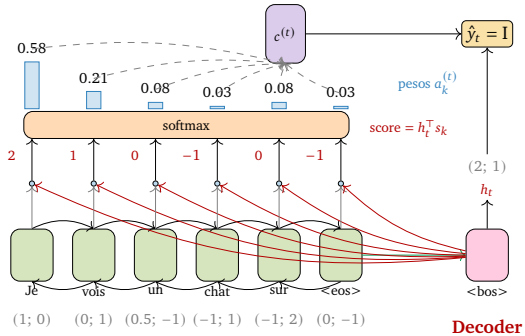
Scores: empilhando os s_k como colunas de S, um único produto $h_t^\top S$:

$$(2\ \ 1)\begin{pmatrix}1 & 0 & 0.5 & -1 & -1 & 0\\ 0 & 1 & -1 & 1 & 2 & -1\end{pmatrix} = (2\ \ 1\ \ 0\ \ -1\ \ 0\ \ -1)$$

Pesos de Atenção: $a^{(t)} = \text{softmax}(\text{scores}_t) = [\ 0.58;\ 0.21;\ 0.08;\ 0.03;\ 0.08;\ 0.03\ ]$

Contexto: $c^{(t)} = \sum_k a_k^{(t)}\, s_k \approx (0.51;\ 0.29)$

Atenção: Diagrama



Scores: empilhando os s_k como colunas de S, um único produto $h_t^\top S$:

$$(2\ \ 1)\begin{pmatrix} 1 & 0 & 0.5 & -1 & -1 & 0 \\ 0 & 1 & -1 & 1 & 2 & -1 \end{pmatrix} = (\,2\ \ 1\ \ 0\ \ -1\ \ 0\ \ -1\,)$$

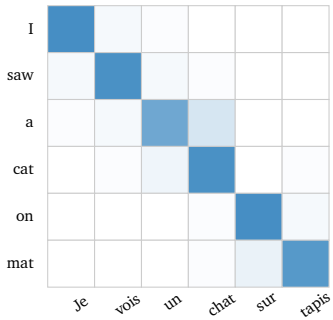
Pesos de Atenção: $a^{(t)} = \text{softmax}(\text{scores}_t) = [\ 0.58;\ 0.21;\ 0.08;\ 0.03;\ 0.08;\ 0.03\ ]$

Contexto: $c^{(t)} = \sum_k a_k^{(t)}\, s_k \approx (0.51;\ 0.29)$

- A cada *token* gerado o cálculo se repete e a distribuição de atenção se **redistribui** sobre a entrada (ver a matriz de alinhamento a seguir).

O Que a Atenção Aprende

- Empilhando os pesos $a_k^{(t)}$ de todos os passos formamos uma **matriz de alinhamento**.
- Cada célula mostra o quanto o *token* de saída (linha) prestou atenção ao *token* de entrada (coluna).
- O padrão quase diagonal indica que o modelo aprendeu o alinhamento entre os idiomas sem supervisão explícita.



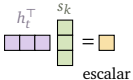
escuro = peso de atenção alto

Variantes do Score

- Existem várias formas de calcular score(h_t, s_k). As três mais clássicas[3], vistas como produtos de matrizes:

Dot-product

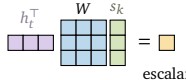
score $= h_t^\top s_k$



sem parâmetros

Bilinear (Luong)

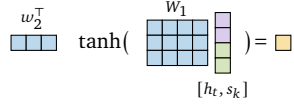
score $= h_t^\top W s_k$



matriz W aprendida

MLP (Bahdanau)

score $= w_2^\top \tanh(W_1 [h_t, s_k])$



camada não linear

- Em Transformers, usamos (*scaled dot-product*)[4].

[3] Minh-Thang Luong, Hieu Pham e Christopher D. Manning. "Effective approaches to attention-based neural machine translation". Em: arXiv preprint arXiv:1508.04025 (2015).

[4] Ashish Vaswani et al. "Attention is all you need". Em: NeurIPS. vol. 30. 2017.

Leituras Recomendadas

- Capítulo 10 de Aston Zhang et al. Dive into Deep Learning. Cambridge University Press, 2023. URL: https://d2l.ai/.
- *Excelente blog post* de Lena Voita: https://lena-voita.github.io/nlp_course/language_modeling.html.
- *HuggingFace blog*: https://huggingface.co/blog/how-to-generate.
- Holtzman et al., *The Curious Case of Neural Text Degeneration* (2019).

Referências I

[1] Dzmitry Bahdanau, Kyunghyun Cho e Yoshua Bengio. “Neural machine translation by jointly learning to align and translate”. Em: arXiv preprint arXiv:1409.0473 (2014).

[2] Minh-Thang Luong, Hieu Pham e Christopher D. Manning. “Effective approaches to attention-based neural machine translation”. Em: arXiv preprint arXiv:1508.04025 (2015).

[3] Ilya Sutskever, Oriol Vinyals e Quoc V. Le. “Sequence to sequence learning with neural networks”. Em: NeurIPS. Vol. 27. 2014.

[4] Ashish Vaswani et al. “Attention is all you need”. Em: NeurIPS. Vol. 30. 2017.

[5] Aston Zhang et al. Dive into Deep Learning. Cambridge University Press, 2023. URL: https://d2l.ai/.