Deep Learning

Modelos Encoder-Only e Sentence Embeddings

Lucas Silveira Kupssinskü

Machine Learning Theory and Applications Laboratory
PUCRS

agosto de 2026

Visão Geral

1. Paradigmas Arquiteturais do Transformer
2. De Embeddings Estáticos a Contextuais
3. Tarefas em Nível de *Token*
4. BERT
5. De *Tokens* a Sentenças: *Sentence Embeddings*
6. Aprendizado Contrastivo
7. SBERT: *Sentence-BERT*
8. Tarefas e Avaliação de *Sentence Embeddings*
9. Encerramento

Visão Geral

As Três Famílias de Transformers

Desde o início da arquitetura *Transformer* deu origem a três famílias:

Encoder-Decoder

- T5, BART, mT5
- *Seq2seq*: entrada → saída
- *Cross-attention* liga os dois lados
- Tradução, sumarização

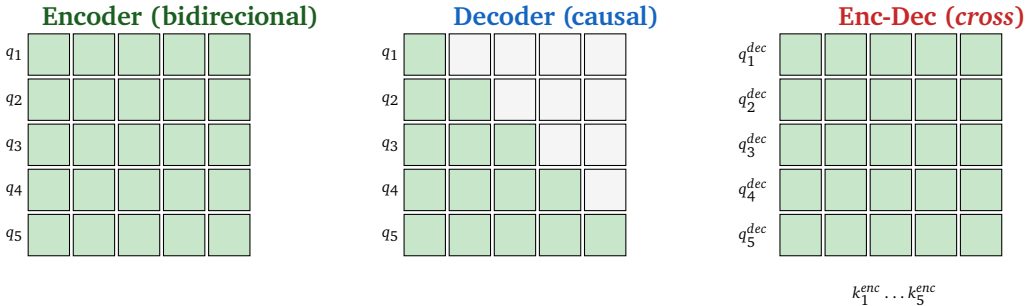
Decoder-Only

- GPT, LLaMA, Mistral
- Atenção **causal** (mascarada)
- Geração autorregressiva
- *Chatbots*, *code completion*

Encoder-Only

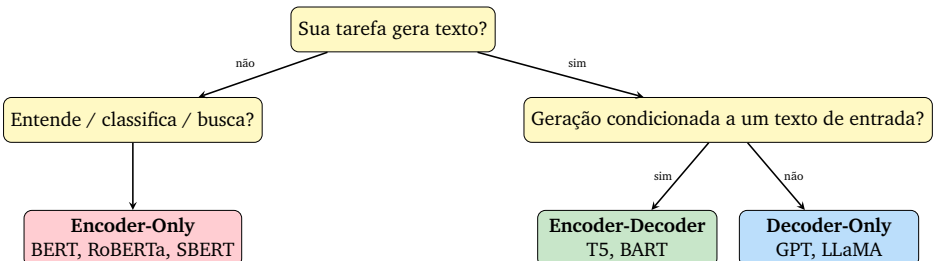
- BERT, RoBERTa, DeBERTa
- Atenção **bidirecional**
- Foco em *compreensão*
- Classificação, busca, NER

Padrões de Atenção: Visualizando a Diferença



- **Encoder**: cada *token* vê **todos** os outros → representação rica em contexto
- **Decoder**: cada *token* vê apenas o passado → geração autorregressiva
- **Enc-Dec**: *decoder* consulta *encoder* via *cross-attention*

Qual Família Usar? Uma Árvore de Decisão



- Tarefas de *compreensão* (classificação, busca, NER, STS) → **encoder-only**
- Geração aberta (*chat*, *code*, criação) → **decoder-only**
- Tradução / sumarização em que entrada e saída são muito diferentes → **encoder-decoder**

O Ramo da Compreensão

Enquanto modelos *decoder-only* (GPT, LLaMA) trabalham com geração de texto, modelos *encoder-only* trabalham com:

- Como transformar texto em vetores *densos* e *contextuais*
- Como usar esses vetores para tarefas discriminativas
- Como treinar embeddings de **frases inteiras** que alimentam sistemas de busca semântica e RAG

Onde usamos *encoder-only*?

- *Retrieval* para RAG
- Busca semântica
- Classificação de palavras/sentenças
- *Reranking* barato
- *Clustering* e deduplicação
- Detecção de conteúdo

Visão Geral

Por que Precisamos de *Word Embeddings*?

Representação *one-hot*:

- Vocabulário de $|V| = 50\text{k} \rightarrow$ vetores de 50k dimensões
- **Esparsa** (um único 1, resto zeros)
- Ortogonal: $\cos(\text{rei}, \text{rainha}) = 0$
- Nenhuma semântica capturada

Word embedding:

- Vetor de baixa dimensão ($d = 300$, 768, ...)
- **Densa** (todas as entradas ativas)
- Palavras semanticamente próximas $\rightarrow$ vetores próximos
- Aprendido a partir de grandes *corpora*

Hipótese distribucional (Harris, 1954)

"*You shall know a word by the company it keeps.*", palavras que aparecem em contextos similares tendem a ter significado similar.

Embeddings Estáticos: Word2Vec e GloVe

Word2Vec[1]

- **Skip-gram**: prever palavras do contexto a partir da palavra central
- **CBOW**: prever palavra central a partir do contexto
- Resultado: uma matriz $E \in \mathbb{R}^{|V| \times d}$

GloVe[2]

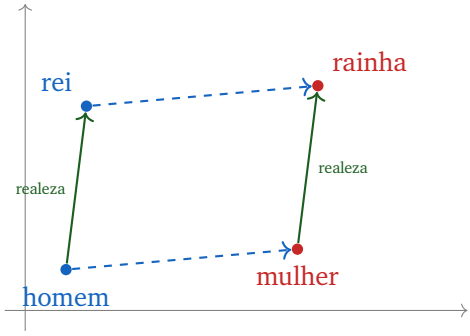
- Fatoração de matriz de co-ocorrência global
- Combina estatística global (como LSA) com janelas locais (como W2V)
- Resultado: uma matriz $E \in \mathbb{R}^{|V| \times d}$

Cada palavra é representada por *UM* único vetor, independente do contexto.

[1] Tomas Mikolov et al. "Distributed Representations of Words and Phrases and their Compositionality". Em: NeurIPS. vol. 26. 2013.

[2] Jeffrey Pennington, Richard Socher e Christopher D. Manning. "GloVe: Global Vectors for Word Representation". Em: EMNLP. 2014, pp. 1532–1543.

A Visualização Clássica



$$\mathbf{v}_{\text{rei}} - \mathbf{v}_{\text{homem}} + \mathbf{v}_{\text{mulher}} \approx \mathbf{v}_{\text{rainha}}$$

- Direções no espaço codificam **relações semânticas**
- Mas: **um único vetor por palavra**, independente do contexto

O Problema da Polissemia

Duas frases em português

1. “Ele sacou dinheiro no **banco** da esquina.”
2. “Ela sentou no **banco** do parque.”

Embeddings estáticos:

- $\mathbf{v}^{(1)}_{\text{banco}} = \mathbf{v}^{(2)}_{\text{banco}}$
- O vetor é uma *média* dos dois sentidos
- O modelo “confunde” instituição financeira com assento

Embeddings contextuais:

- $\mathbf{v}^{(1)}_{\text{banco}} \neq \mathbf{v}^{(2)}_{\text{banco}}$
- Cada ocorrência recebe um vetor diferente
- O contexto *molda* o vetor final

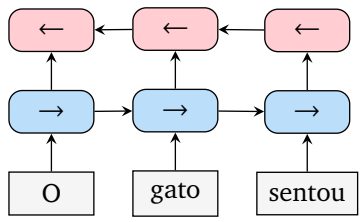
A função que gera o *embedding* deve depender do contexto $f(\text{palavra}, \text{contexto}) \to \mathbb{R}^d$.

A Ponte: ELMo (2018)[3]

Ideia: em vez de uma tabela *lookup*, use um **modelo de linguagem bidirecional** pré-treinado.
Duas LSTMs profundas:

- Uma da esquerda para direita $p(w_t|w_{<t})$
- A outra da direita para a esquerda $p(w_t|w_{>t})$
- *Embedding* final = combinação linear das camadas das duas LSTMs

Primeiros embeddings verdadeiramente contextuais a ganhar tração em NLP.



Duas LSTMs empilhadas, direções opostas.

[3] Matthew E. Peters et al. "Deep Contextualized Word Representations". Em: NAACL-HLT. 2018, pp. 2227–2237.

Embeddings Contextuais via *Transformers*

Com a chegada do *Transformer*, as LSTMs do ELMo foram substituídas por **camadas de *self-attention***:

- Cada *token* atende a **todos** os outros da sequência
- A representação final de cada *token* é uma mistura de todos os outros, ponderada pelos *scores* de atenção
- Paralelizável (ao contrário de LSTMs)
- Captura dependências de longo alcance sem o *vanishing gradient*

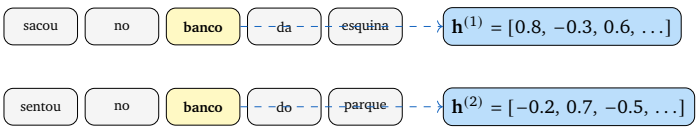
Formalizando

Dado um *token* w_i na posição i, o *embedding* contextual na camada ℓ é:

$$\mathbf{h}_i^{(\ell)} = f_\ell\left(\mathbf{h}_0^{(\ell-1)}, \mathbf{h}_1^{(\ell-1)}, \ldots, \mathbf{h}_{L-1}^{(\ell-1)},\ i\right)$$

onde f_ℓ é uma camada *Transformer encoder* (self-attention + FFN).

“Banco” em dois contextos



$\cos(\mathbf{h}^{(1)}, \mathbf{h}^{(2)}) \approx 0.15$ $\cos(\mathbf{h}^{(1)}, \mathbf{h}_{\text{agência}}) \approx 0.82$

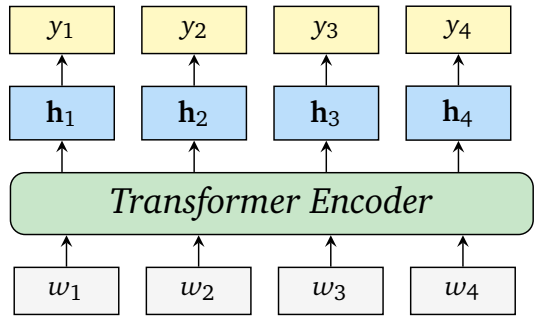
- A **mesma palavra** produz **vetores diferentes** em contextos diferentes
- No primeiro contexto, “banco” fica próximo de “agência”, “caixa”, “conta”
- No segundo, fica próximo de “cadeira”, “praça”, “sentar”

Visão Geral

A Receita Geral

Para qualquer tarefa em nível de *token*:

1. Passe a frase por um *encoder* pré-treinado
2. Extraia os *embeddings* contextuais $\mathbf{h}_1, \ldots, \mathbf{h}_L$
3. Adicione uma pequena cabeça linear $W \in \mathbb{R}^{d \times C}$
4. $\text{logits}_i = W^\top \mathbf{h}_i$
5. *Softmax* sobre C classes



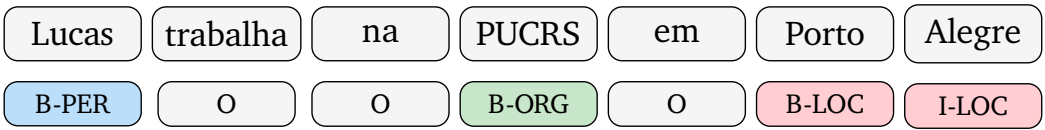
Fine-tuning da Cabeça de Classificação

A cabeça tem tipicamente $< 1\%$ dos parâmetros totais. Quase todo o conhecimento vem do *pré-treino*.

Tarefa: *Named Entity Recognition* (NER)

Classificar cada *token* em uma categoria de entidade (Pessoa, Local, Organização, ...).

Esquema BIO: **B**egin, **I**nside, **O**utside de uma entidade.

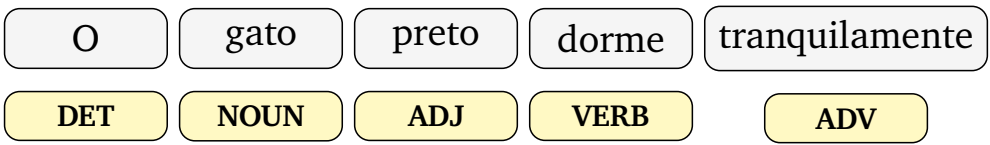


- **Entidades multi-*token*** ("Porto Alegre") são capturadas por **B-** seguido de **I-**
- Cabeça linear $W \in \mathbb{R}^{d \times C}$ com $C = 2k + 1$ (um B- e um I- por tipo, mais O), onde k é o número de categorias presentes no NER
- *Benchmark* clássico: CoNLL-2003[4]

[4] Erik F. Tjong Kim Sang e Fien De Meulder. "Introduction to the CoNLL-2003 Shared Task: Language-Independent Named Entity Recognition". Em: CoNLL. 2003, pp. 142–147.

Tarefa: *Part-of-Speech Tagging* (POS)

Classificar cada *token* em uma categoria gramatical (substantivo, verbo, adjetivo, …).



- Problema clássico de NLP, resolvido com **altíssima precisão** por encoders modernos (> 97%)
- *Embeddings contextuais* são ideais: a mesma palavra pode ter *tag* diferente em contextos diferentes
  - O **jogo** de ontem foi muito disputado.
  - Eu **jogo** playstation todas as terças-feiras
- *Universal Dependencies*: esquema de *tags* compartilhado entre > 100 línguas

Tarefa: *Extractive Question Answering* (SQuAD)[5]

Dado uma frase no seguinte formato: **Contexto** `[SEP]` **Pergunta** `[SEP]`, identificar o **trecho** do parágrafo que responde a pergunta.

Exemplo

"`[CLS]A PUCRS foi fundada em 1948 na cidade de Porto Alegre.[SEP]Quando a PUCRS foi fundada?[SEP]`"

Resposta: **1948** → *span* com início no *token* 5, fim no *token* 5

Duas saídas: posição de início e posição do final da resposta.

- $\mathbf{w}_{\text{start}}, \mathbf{w}_{\text{end}} \in \mathbb{R}^d$
- $p_{\text{start}}(i) = \text{softmax}(\mathbf{w}_{\text{start}}^\top \mathbf{h}_i)$
- $p_{\text{end}}(i) = \text{softmax}(\mathbf{w}_{\text{end}}^\top \mathbf{h}_i)$
- Treino: cross-entropy nos índices de início e fim

[5] Pranav Rajpurkar et al. "SQuAD: 100,000+ Questions for Machine Comprehension of Text". Em: EMNLP. 2016, pp. 2383–2392.

Por que Embeddings Contextuais Dominam Essas Tarefas?

Com embeddings estáticos (pré-2018):

- Necessário *feature engineering*, regras feitas a mão
- Desempenho limitado pela ambiguidade lexical
- NER no CoNLL-2003: $\sim$ 90% F1 com modelos específicos

Com embeddings contextuais:

- Uma camada linear acima do *encoder*
- Resolução de ambiguidade é "gratuita" (já contextual)
- NER no CoNLL-2003: $>$ 94% F1 com BERT
- SQuAD: humano $\approx$ 91%, BERT-Large $\approx$ 93%

A lição

*Pré-treinar um **encoder** em grande escala + cabeça linear pequena na tarefa alvo tornou-se a **receita padrão** de NLP entre 2018 e 2022.*

Visão Geral

BERT: *Bidirectional Encoder Representations*[6]

Contribuições principais:

- Usa apenas a **pilha encoder** do *Transformer*
- Pré-treino **bidirecional** com dois objetivos novos:
  - *Masked Language Modeling* (MLM)
  - *Next Sentence Prediction* (NSP)
- "*Fine-tuning*": mesmo modelo + cabeça pequena resolve > 11 tarefas de NLP
- *State-of-the-art* em quase todos os *benchmarks* da época

Por que "bidirecional"?

GPT-1 (2018) era unidirecional (causal). BERT mostrou que, para *compreensão*, ver **o futuro** da frase melhora drasticamente. Mas: **não pode gerar texto** diretamente.

[6] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". Em: NAACL-HLT. 2019, pp. 4171–4186.

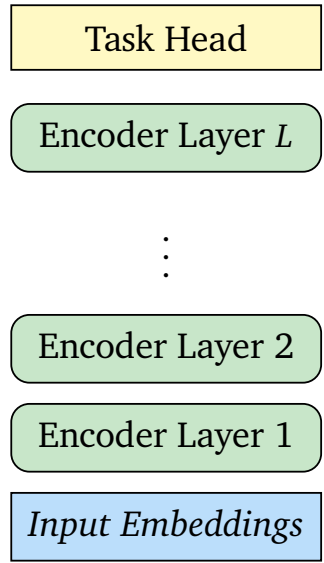
Arquitetura: Pilha de Encoders do *Transformer*

BERT = N camadas de *encoder*, cada uma com:

- *Multi-head self-attention* **bidirecional** (sem máscara causal)
- *Feed-forward network* (GELU)
- *Layer Norm* + conexões residuais

Duas versões oficiais:

	Base	**Large**
Camadas (L)	12	24
Hidden (d)	768	1024
Heads (h)	12	16
Parâmetros	110M	340M



Dados de Pré-Treino

BERT foi pré-treinado em:

- **BooksCorpus**: ~ 800M palavras de livros não publicados
- **Wikipedia inglesa**: ~ 2.5B palavras (só texto, sem tabelas/listas)

Total: ~ 3.3B palavras, ~ 16 GB de texto.

Por que esses dados?

- **Gratuitos** e de **alta qualidade**
- Contêm estruturas de longa distância (sentenças contíguas) → útil para NSP
- Linguagem formal e variada

Tokenização: WordPiece

BERT usa **WordPiece**.

Recapitulação rápida

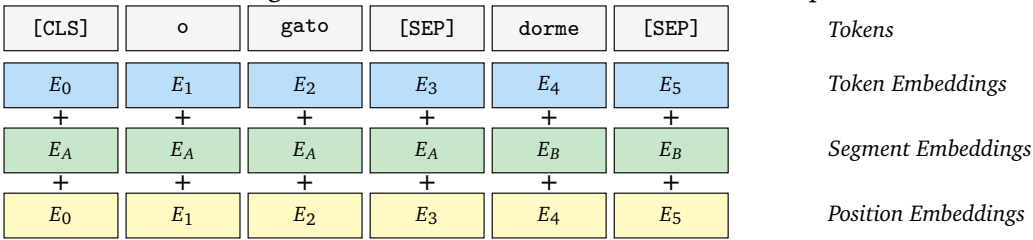
- Vocabulário de ~ 30k *subwords*
- Palavras raras quebradas em prefixos conhecidos + sufixos marcados com `##`
- Exemplo: `tokenização` → `token ##iza ##ção`
- Resolve o problema de OOV (*out-of-vocabulary*) sem explodir o vocabulário

Tokens especiais adicionados pelo tokenizer do BERT:

- `[CLS]`: marca o início da sequência
- `[SEP]`: separador de segmentos
- `[PAD]`: preenchimento para lotes
- `[MASK]`: usado apenas no pré-treino (MLM)
- `[UNK]`: *token* desconhecido (raro com WordPiece)

Representação de Entrada do BERT

O *embedding* final de cada *token* é a **soma** de três componentes:



$$\mathbf{x}_i = E_{\text{token}}(w_i) \; + \; E_{\text{segment}}(s_i) \; + \; E_{\text{position}}(i)$$

Tudo é aprendido (inclusive a posição, ao contrário do PE sinusoidal).

O *Token* [CLS]

[CLS] é um *token* **sempre** prefixado à sequência.

Papel:

- Não é uma palavra real, começa como um *embedding* aleatório aprendido
- Ao longo das L camadas, acumula informação de **toda a frase** via *self-attention*
- Seu estado final $\mathbf{h}_{[\text{CLS}]}^{(L)}$ é usado como **representação agregada da sequência**
- Cabeças de **classificação** de sentença ligam-se a ele

Observação

$\mathbf{h}_{[\text{CLS}]}$ do BERT não é um bom *sentence embedding*.

[SEP] e *Segment Embeddings*

BERT pode receber uma ou **duas** sentenças:

Formato para uma sentença

```
[CLS] o gato dorme [SEP]
```

Formato para duas sentenças (par)

```
[CLS] o gato dorme [SEP] no tapete vermelho [SEP]
```

- `[SEP]` marca **fronteiras** entre segmentos
- **Segment embeddings**: E_A adicionado aos *tokens* da primeira sentença, E_B aos da segunda
- Permite tarefas de **pares de sentença**: NLI, similaridade, paráfrase, QA

Insight: sem *segment embeddings*, o modelo não saberia se um *token* pertence à pergunta ou ao contexto em uma tarefa de QA.

Comprimento Fixo e *Padding*

BERT tem um comprimento máximo fixo de 512 *tokens*.

Por quê fixo?

- *Position embeddings* aprendidos: só existem para posições $0, \ldots, 511$
- Atenção $O(L^2)$ → custo proibitivo para L grande
- *Batching* eficiente requer forma uniforme

Como uniformizar um *batch*?

- Frases **curtas** → ***padding*** com `[PAD]` até o comprimento desejado
- Frases **longas** → **truncagem** (ou *sliding window*)
- *Attention mask* indica quais posições são reais

Máscara de atenção

`[CLS]`	`o`	`gato`	`[SEP]`	`[PAD]`	`[PAD]`	`[PAD]`	`[PAD]`
1	1	1	1	0	0	0	0

Posições com `0` recebem $-\infty$ nos *logits* de atenção e são efetivamente ignoradas.

Objetivo #1: *Masked Language Modeling* (MLM)

Ideia: esconder aleatoriamente 15% dos *tokens* e pedir ao modelo que os recupere.

modelo deve prever: “sentou”

- Perda: *cross-entropy* sobre o vocabulário para **cada posição mascarada**
- O modelo aprende **atenção bidirecional**: pode olhar para a esquerda E direita
- Contraste com GPT: que só vê a esquerda → **sinal de treino mais fraco por *token***

$$\mathcal{L}_{\text{MLM}} = -\frac{1}{|M|} \sum_{i \in M} \log p(w_i \mid \mathbf{w}_{\setminus M})$$

onde M é o conjunto de posições mascaradas.

A Regra 80/10/10 do MLM

Dos 15% *tokens* selecionados para máscara, **nem todos** viram `[MASK]`:

Distribuição

- 80%: substituído por `[MASK]`
- 10%: substituído por **palavra aleatória**
- 10%: **mantido** como está

Em todos os casos, o modelo deve prever a palavra original.

Por que essa mistura?

- `[MASK]` só existe no pré-treino, nunca no *fine-tuning* → **desalinhamento**
- 10% aleatórios forçam o modelo a ser "cético" sobre *tokens* visíveis
- 10% intactos garantem representações úteis para *tokens* não-mascarados

Exemplo

- "*O gato* [MASK] *no tapete*" 80%
- "*O gato* ***banana*** *no tapete*" 10%
- "*O gato sentou no tapete*" (inalterado) 10%

MLM: Exemplo Numérico

Frase: "`A capital da França é Paris`" (6 *tokens*)

1. $15\% \times 6 \approx 1$ *token* selecionado. Suponha que seja "*Paris*".
2. Com probabilidade 80% → "`A capital da França é [MASK]`"
3. *Forward pass* pelo BERT → estado oculto $\mathbf{h}_{[\text{MASK}]} \in \mathbb{R}^{768}$
4. Cabeça linear → *logits* sobre o vocabulário ($|V| \approx 30$k)
5. *Softmax* → distribuição de probabilidade

Topo das predições (hipotético):

Token	Probabilidade
Paris	0.71
Lyon	0.08
Marselha	0.04
Berlim	0.02
…	…

Perda: $-\log 0.71 \approx 0.34$ na posição mascarada.

Objetivo #2: *Next Sentence Prediction* (NSP)

Ideia: dado um par de sentenças (A, B), prever se B é a sentença que **segue** A no texto original, ou se é uma sentença aleatória.

Exemplo positivo (`IsNext`)

A: "Ele abriu a porta do carro."
B: "Em seguida, entrou e ligou o motor."

Exemplo negativo (`NotNext`)

A: "Ele abriu a porta do carro."
B: "Mercúrio é o menor planeta do sistema solar."

Arquitetura:

- Entrada: `[CLS] A [SEP] B [SEP]`
- Cabeça binária sobre $\mathbf{h}^{(L)}_{[\text{CLS}]}$
- *Cross-entropy* binária

Motivação: treinar o modelo a entender **relações entre sentenças** (útil para NLI, QA, …).

Spoiler: RoBERTa[7] mostrou que **NSP não ajuda**, modelos de *word embedding* e *sentence embedding* mais modernos não usam essa tarefa.

[7] Yinhan Liu et al. "RoBERTa: A Robustly Optimized BERT Pretraining Approach". Em: arXiv preprint arXiv:1907.11692 (2019).

Perda Total de Pré-Treino

As duas perdas são combinadas em um único escalar:

$$\mathcal{L}_{\text{BERT}} = \mathcal{L}_{\text{MLM}} + \mathcal{L}_{\text{NSP}}$$

- O **mesmo** *forward pass* gera ambas as perdas
- $\mathcal{L}_{\text{MLM}}$ usa as posições mascaradas
- $\mathcal{L}_{\text{NSP}}$ usa a posição `[CLS]`
- *Backprop* ocorre conjuntamente, com os gradientes somados

Treinamento original

- 1M passos de 256 sequências, $\sim$ 40 épocas
- 4 TPU *pods*, $\sim$ 4 dias para BERT-Base
- *Adam* com *warmup* linear

Fine-Tuning: Uma Cabeça, Muitas Tarefas

Depois do pré-treino, o BERT pode ser **adaptado** a uma tarefa específica treinando por poucas épocas com uma **cabeça nova**.

Receita genérica:

1. Carregar pesos pré-treinados
2. Substituir a cabeça MLM / NSP por uma cabeça específica da tarefa (inicializada aleatoriamente)
3. Treinar **tudo** de ponta a ponta por 2–5 épocas
4. *Learning rate* baixa ($\sim 2 \times 10^{-5}$)

Surpreendentemente geral

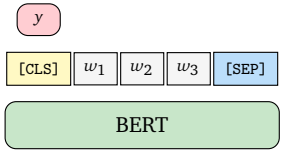
O **mesmo modelo base** resolve:

- Classificação de frase (SST-2, CoLA)
- Pares de frase (NLI, paráfrase)
- Marcação de *token* (NER, POS)

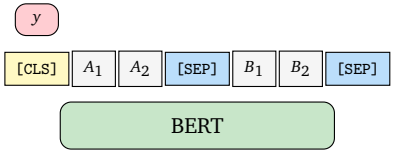
Só muda a cabeça e o formato da entrada.

Quatro Formatos de *Fine-Tuning*

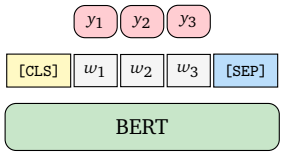
(a) Classificação de sentença



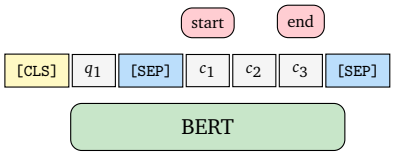
(b) Classificação de par



(c) Marcação de *token*



(d) SQuAD



- (a) e (b) usam $\mathbf{h}_{[\text{CLS}]}$ como representação da (pair de) frase
- (c) usa os vetores por *token* $\mathbf{h}_i$
- (d) usa dois vetores globais $\mathbf{w}_{\text{start}}, \mathbf{w}_{\text{end}}$ para classificar cada posição

Feature Extraction vs *Fine-Tuning*

Duas estratégias para usar um modelo pré-treinado:

Feature Extraction (pesos congelados)

- Congela BERT, treina só a cabeça
- Muito mais rápido e barato
- Pode **cachear** os *embeddings*
- Desempenho geralmente menor
- Útil quando temos poucos dados ou pouca GPU

Fine-Tuning (pesos descongelados)

- Treina BERT **inteiro** + cabeça
- *Learning rate* pequena (2×10^{-5})
- Melhor desempenho
- Custo: $\sim 10\times$ mais GPU
- Risco de *overfitting* com poucos dados

Ainda há possibilidade de usar adaptadores LoRA para treinar apenas um subconjunto pequeno de parâmetros.

O Ecossistema BERT: Principais Variantes

- **RoBERTa** [8]: Remove NSP, mais dados (160 GB), mascaramento dinâmico, *batch* maior. **Resultado**: +2 pts no GLUE.
- **ALBERT** [7]: Compartilhamento de parâmetros entre camadas + fatoração do *embedding*. **Resultado**: −90% parâmetros.
- **DeBERTa** [5]: *Disentangled attention* (conteúdo e posição separados) + *enhanced mask decoder*. **Resultado**: SOTA em SuperGLUE.
- **ELECTRA** [3]: Substitui MLM por *Replaced Token Detection*: discrimina *tokens* reais de gerados. **Resultado**: mesmo desempenho com ∼25% do *compute*.

- Todos são **encoder-only** e mantêm a receita pré-treino → *fine-tuning*
- ELECTRA: treina **todas** as posições, não só os 15% mascarados → sinal denso
- DeBERTa-v3 ainda é competitivo em classificação e NLI

Visão Geral

Motivação: Um Vetor por Sentença

Muitas aplicações precisam de **um único vetor** que represente uma sentença ou documento inteiro:

Aplicações

- **Busca semântica**: encontrar documentos similares a uma consulta
- **RAG**: recuperar contexto relevante para um LLM
- **Clustering**: agrupar documentos por tópico
- **Deduplicação**: detectar paráfrases
- **Classificação**: *zero-shot* com *embeddings* comparados a rótulos
- **Recomendação**: *item-to-item*

Requisitos:

- **Rápido**: milhões de vetores pré-computados, consulta em milissegundos
- **Compacto**: 384–1024 dimensões
- **Semanticamente** estruturado: similaridade cosseno deve refletir similaridade de significado

BERT *out-of-the-box* resolve isso?

Não muito bem, como veremos.

Abordagens Ingênuas

Como extrair **um** vetor de BERT para uma sentença?

`[CLS]` *pooling*

- Usar $\mathbf{h}_{[\text{CLS}]}^{(L)}$
- "Parece o certo": ele foi projetado para classificação
- **Ruim** para similaridade

Mean pooling

- $\frac{1}{L}\sum_i \mathbf{h}_i^{(L)}$
- Simples, robusto
- Usa a máscara para ignorar `[PAD]`
- Empiricamente **melhor** que `[CLS]`

Max pooling

- $\max_i \mathbf{h}_i^{(L)}$ por dimensão
- Captura "*features* mais salientes"
- Menos usada na prática

Descoberta de [15]

BERT *vanilla* + `[CLS]` performa *pior* que GloVe em *Semantic Textual Similarity* (STS) usando similaridade cosseno!

Por Que [CLS] *Vanilla* Falha em Similaridade?

O que o [CLS] aprende?

- No pré-treino: **NSP** → "esta é a próxima frase?"
- Esse é um sinal *binário* e ruidoso
- Não há incentivo para **estruturar o espaço** de forma que frases similares fiquem próximas

Consequência:

- O espaço de [CLS] é **anisotrópico**: todos os vetores ficam em um cone estreito
- Distâncias cosseno são pouco informativas
- Frases completamente diferentes podem ter cosseno > 0.9

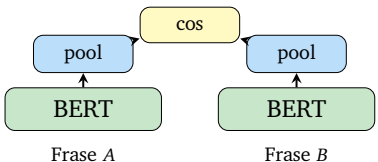
Logo...

BERT foi treinado para *token-level* MLM e classificação NSP, **não** para similaridade semântica.
Para obter bons *sentence embeddings* precisamos de um **novo objetivo de treino**.

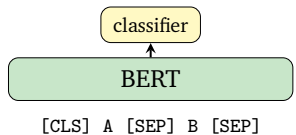
Bi-Encoder vs *Cross-Encoder*

Duas formas de comparar duas frases com um *encoder*:

Bi-Encoder



Cross-Encoder



Bi-Encoder:

- Codifica A e B **independentemente**
- A pode ser pré-computado e armazenado
- Consulta: 1 *forward pass* por B + busca em vetor
- **Rápido**, escala para bilhões

Cross-Encoder:

- Vê os dois **juntos**, atenção cruzada *full*
- Melhor desempenho (mais informação)
- 1 *forward pass* **por par** $\rightarrow O(n^2)$
- **Inviável** para busca em escala

A Diferença de Velocidade: 65 horas → 5 segundos

Experimento de [15]:

- Corpus: 10 000 frases
- Tarefa: encontrar os pares mais similares
- Pares possíveis: $\binom{10\,000}{2} = 49\,995\,000$

Cross-encoder BERT

~ 65 horas para computar todas as similaridades par-a-par.

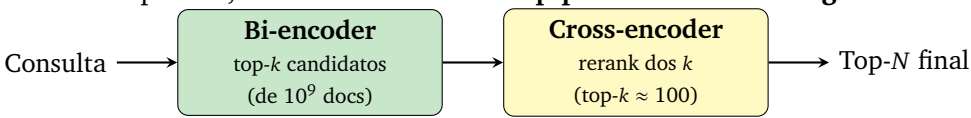
Bi-encoder SBERT

- Codificar as 10k frases: ~ 5 segundos
- Comparar todos os pares: ~ 0.01 segundos (matriz dot-product)

Mais de 40 000× mais rápido. Esse é o problema que SBERT resolve.

Arquitetura Híbrida: *Retrieve + Rerank*

Na prática, usa-se os dois em uma **pipeline em dois estágios**:



- **Bi-encoder** filtra bilhões → centenas em milissegundos
- **Cross-encoder** reordena as centenas com alta precisão
- Usa o melhor dos dois mundos: escala e qualidade
- Padrão em sistemas modernos de *retrieval* (Google, *semantic search* empresarial, RAG)

Visão Geral

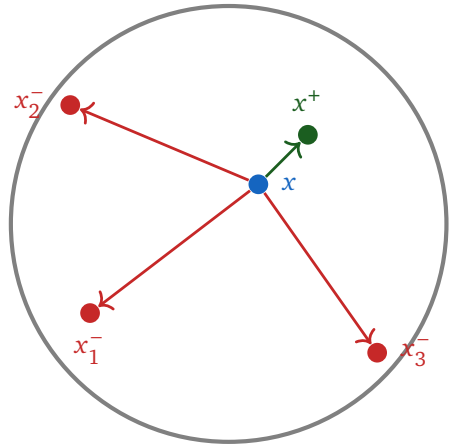
Intuição: Puxar os Similares, Empurrar os Diferentes

O **aprendizado contrastivo** treina um *encoder* f_θ tal que:

$$\cos(f_\theta(x), f_\theta(x^+)) \gg \cos(f_\theta(x), f_\theta(x^-))$$

- x: *anchor*
- x^+: positivo (semanticamente similar)
- x^-: negativo (diferente)

Não precisa de rótulos explícitos, basta **pares positivos**.



Direção do gradiente: puxa x^+ para perto, empurra x^- para longe.

De Onde Vêm os Pares Positivos?

Pares *naturais* (supervisão fraca)

- Pergunta ↔ resposta (StackExchange, Quora)
- Título ↔ corpo (notícias, artigos)
- Parágrafos adjacentes
- Pares *query*↔*click* de *logs*
- Traduções (mesma frase em línguas diferentes)

Pares *anotados* (supervisão forte)

- NLI (SNLI, MultiNLI): *entailment* → positivo, *contradiction* → negativo
- Duplicatas do Quora
- MS MARCO: *query*→passagem relevante (relevância humana)
- Paráfrases geradas por humanos

Escala importa

Modelos modernos (e5, bge) treinam com **centenas de milhões** de pares positivos coletados da web com supervisão fraca, depois fazem *fine-tuning* em dados anotados.

Triplet Loss

Originário do **FaceNet**[8] para reconhecimento facial.

Formulação

$$\mathcal{L}_{\text{triplet}} = \max\Big(0,\ \|f(x) - f(x^+)\|_2^2 - \|f(x) - f(x^-)\|_2^2 + \alpha\Big)$$

onde $\alpha > 0$ é a **margem**.

- A perda é zero quando o positivo está pelo menos α mais próximo que o negativo
- **Margem** evita colapso trivial (todas as representações iguais)
- Funciona, mas precisa de **triplos** (x, x^+, x^-) construídos explicitamente
- Limitação: **um único negativo por *anchor*** → pouco sinal por *step*

A continuação natural é usar **muitos** negativos de uma vez...

[8] Florian Schroff, Dmitry Kalenichenko e James Philbin. "FaceNet: A Unified Embedding for Face Recognition and Clustering". Em: CVPR. 2015, pp. 815–823.

InfoNCE / Multiple Negatives Ranking Loss[9]

Formulação

Para um *batch* de N pares (x_i, x_i^+) e todos os outros exemplos como negativos:

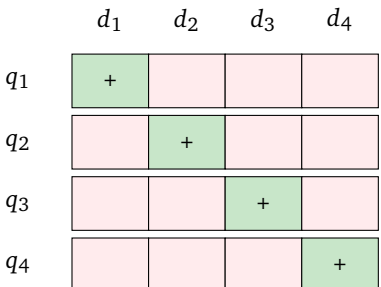
$$\mathcal{L}_{\text{NCE}} = -\frac{1}{N}\sum_{i=1}^{N} \log \frac{\exp(\text{sim}(f(x_i), f(x_i^+))/\tau)}{\sum_{j=1}^{N} \exp(\text{sim}(f(x_i), f(x_j^+))/\tau)}$$

- sim: produto escalar ou cosseno
- τ: **temperatura** (tipicamente $\tau = 0.05$ a 0.1)
- **In-batch negatives**: cada x_j^+ com $j \neq i$ atua como negativo para x_i
- Equivale a um problema de **classificação** de N vias: "qual dos N candidatos é o verdadeiro par?"

[9] Aaron van den Oord, Yazhe Li e Oriol Vinyals. "Representation Learning with Contrastive Predictive Coding". Em: arXiv preprint arXiv:1807.03748 (2018).

In-Batch Negatives: Exemplo Visual

Batch de 4 pares: $(q_1, d_1), (q_2, d_2), (q_3, d_3), (q_4, d_4)$.



- Matriz $S_{ij} = \text{sim}(f(q_i), f(d_j))/\tau$
- Aplica-se *softmax* em cada **linha**
- Alvo: classe i (a diagonal) $\rightarrow$ *cross-entropy*
- Os 3 elementos fora da diagonal de cada linha são **negativos** "grátis"
- *Batches* grandes ($N = 256, 1024, \ldots$) $\rightarrow$ mais negativos $\rightarrow$ gradiente mais informativo

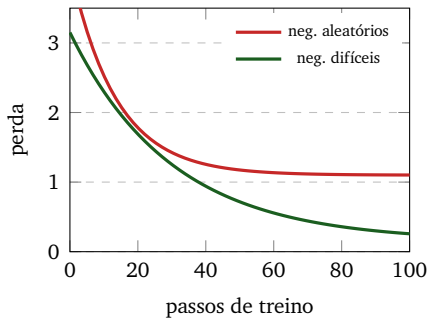
Negativos Aleatórios Saturam

No início do treino, negativos aleatórios são informativos: quase todos dão gradiente.

Depois de algumas épocas:

- A maioria dos negativos aleatórios é **trivialmente diferente** do *anchor*
- Já estão longe no espaço
- Gradiente próximo de zero → **sem sinal de aprendizado**
- O modelo "atinge um platô"

Precisamos de negativos que o modelo *ainda confunda com positivos*.



Ilustrativo: negativos aleatórios saturam, negativos duros continuam informativos.

Hard Negative Mining

Ideia: buscar ativamente negativos que estão **próximos** do *anchor* no espaço atual.

Estratégia 1: BM25 / léxica

- Para cada *query*, pegue documentos **não relevantes** mas com **alta sobreposição lexical**
- Rápido e sem modelo
- Usado em DPR[a]

[a] Vladimir Karpukhin et al. "Dense Passage Retrieval for Open-Domain Question Answering". Em: EMNLP. 2020, pp. 6769–6781.

Estratégia 2: auto-mineração

- Usa o ***checkpoint atual*** para codificar o corpus
- Seleciona os *top-k* vizinhos mais próximos (excluindo o positivo)
- Atualiza os negativos **periodicamente**
- Usado em ANCE[a]

[a] Lee Xiong et al. "Approximate Nearest Neighbor Negative Contrastive Learning for Dense Text Retrieval". Em: ICLR. 2021.

Ordenado por dificuldade: léxica < auto-minerada < adversarial.

A Armadilha dos *False Negatives*

Nem todo “negativo próximo” é realmente negativo.

Exemplo

Query: “Qual a capital do Brasil?”
Positivo rotulado: “Brasília é a capital.”
“Negativo” minerado: “A cidade de Brasília foi inaugurada em 1960 como capital.”

O segundo parágrafo é **tão relevante** quanto o positivo, mas não foi anotado. Treiná-lo como negativo ensina o modelo errado.

Mitigações

- Filtrar negativos com *score* muito alto (pode ser falso negativo)
- Usar um **cross-encoder** para re-verificar
- *Consistency filtering* entre modelos
- ***In-batch*** com lotes grandes é menos suscetível (muitos negativos, baixa probabilidade de colisão)

Notas Práticas: Temperatura e Tamanho de *Batch*

Temperatura τ

- τ pequeno (*e.g.* 0.05) $\rightarrow$ *softmax* "afiada", foca nos mais difíceis
- τ grande (*e.g.* 0.2) $\rightarrow$ distribuição mais plana, gradiente mais suave
- **Hiperparâmetro crítico**: valores errados podem impedir convergência
- Valor típico: 0.05–0.1

Tamanho de *batch* N

- Mais *in-batch* negatives $\rightarrow$ melhor treino
- Modelos SOTA usam $N = 1024$–$16\,384$
- Técnicas: *gradient cache*, *GradCache*, *cross-batch* negatives
- **Lei empírica**: desempenho escala com $\log N$

Receita moderna

Temperatura aprendida (τ como parâmetro treinável) + *batchs* grandes + mistura de positivos naturais e minerados difíceis. Similar ao *CLIP* em visão computacional.

Visão Geral

SBERT: O Primeiro *Sentence Encoder* Prático[10]

Problema

BERT *vanilla* é lento demais para busca semântica em escala (cross-encoder $O(n^2)$) e seus *embeddings* ingênuos são **piores que GloVe** em STS.

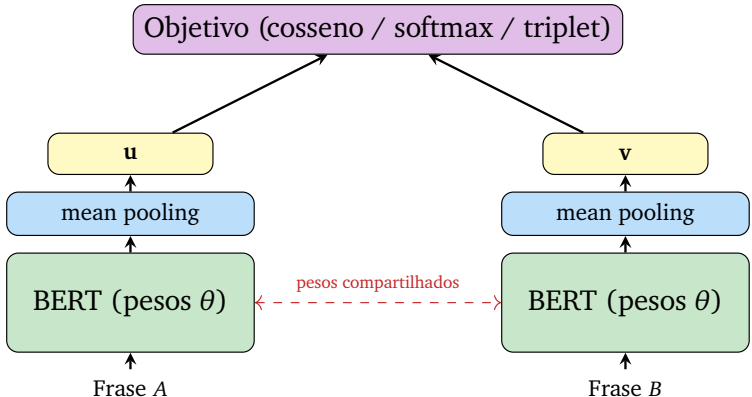
Solução

Fazer **fine-tuning supervisionado** no BERT usando uma arquitetura **siamesa** que produz embeddings densos, rápidos e semanticamente estruturados.

Impacto: a *library* `sentence-transformers` popularizou a técnica e hoje é a base de quase todos os sistemas de busca semântica em produção.

[10] Nils Reimers e Iryna Gurevych. "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks". Em: EMNLP. 2019, pp. 3982–3992.

Arquitetura Siamesa do SBERT



- **Dois** *forward passes* com o **mesmo** BERT
- *Mean pooling* sobre os *tokens* (ignorando *padding*)
- A função-objetivo depende da tarefa de treinamento

Três Objetivos de Treinamento

(1) Classificação (NLI)

Concatenar $(\mathbf{u}, \mathbf{v}, |\mathbf{u} - \mathbf{v}|)$ e alimentar um classificador linear de 3 classes:

$$y = W^\top [\mathbf{u}; \mathbf{v}; |\mathbf{u} - \mathbf{v}|]$$

Usado em SNLI + MultiNLI ($\sim$ 1M pares).

(2) Regressão (STS)

Perda MSE entre o cosseno e o *score* humano:

$$\mathcal{L} = (\cos(\mathbf{u}, \mathbf{v}) - s_{\text{humano}})^2$$

Usado no STS-Benchmark.

(3) Triplet

$\mathcal{L} = \max(0, \|\mathbf{u}_a - \mathbf{u}_p\| - \|\mathbf{u}_a - \mathbf{u}_n\| + \alpha)$

Usado quando se tem triplas (*anchor*, positivo, negativo).

Receita no paper: treinar primeiro em NLI (classificação), depois fazer *fine-tuning* em STS (regressão) $\rightarrow$ melhor *transfer*.

Resultados do SBERT

Modelo	Spearman em STS-b
GloVe (média)	58.02
BERT `[CLS]` *vanilla*	20.29
BERT *mean pooling*	46.35
SBERT-base	**77.03**
SBERT-large	**79.23**

- **BERT *vanilla* é pior que GloVe** → motivação original
- **SBERT** melhora ~ 30+ pontos sobre BERT, ~ 20 sobre GloVe
- Mesma arquitetura base, apenas **mudou o objetivo de treino**

Lição fundamental: o **objetivo de treino** importa tanto quanto a arquitetura.

Aplicações do SBERT (e Descendentes)

Busca semântica

- Substitui BM25 ou o complementa
- Base de motores de busca empresariais
- Indexação com FAISS / HNSW

RAG (*Retrieval-Augmented Generation*)

- Recupera contexto para LLMs
- Reduz alucinações
- Permite atualizações sem re-treino

Deduplicação

- Detecção de quase-duplicatas em grandes corpora
- Limpeza de dados de treino para LLMs

Recomendação item-a-item

- Produtos, artigos, *posts*
- “Mais como este” baseado em texto

Análise de tópicos

- Clustering + visualização
- *BERTopic*, *Top2Vec*

O Legado do SBERT: Modelos Modernos

Os melhores modelos de *embedding* de hoje seguem **a mesma receita** do SBERT.

Família	Organização	Característica distintiva
e5 [18]	Microsoft	Pré-treino contrastivo fraco + *fine-tuning* supervisionado
bge [19]	BAAI	Multilíngue, forte *retrieval*, variante *reranker*
GTE	Alibaba	Foco em generalização entre domínios
Nomic	Nomic AI	Código aberto, dados públicos, alto *context length*
Jina	Jina AI	*Long context* (8k+), multilíngue

- Todos usam **aprendizado contrastivo** em **centenas de milhões** de pares
- Maioria usa **hard negative mining**
- **Top-10 do MTEB** inclui modelos de $\sim$ 7B parâmetros, treinados com *instruction tuning*
- A receita SBERT (2019) continua fundamentalmente válida

O Legado do SBERT: A Receita

A receita que dominou o campo

1. Comece de um *encoder* pré-treinado (BERT, RoBERTa, ...)
2. Arquitetura siamesa com *mean pooling*
3. Aprendizado contrastivo com **in-batch negatives**
4. *Hard negative mining* para a fase final
5. Treinar em dados diversos e supervisionados

- Funciona do tamanho de 110M até 7B parâmetros
- Funciona em inglês, português, multilingue
- Base de todos os sistemas de RAG em produção
- Ainda é a forma **mais eficiente** de comparar textos em escala

Visão Geral

Tarefa: *Retrieval* (Busca Semântica)

Objetivo: dado uma *query*, encontrar os documentos mais relevantes em um corpus de milhões/bilhões. **Pipeline offline**:

1. Codifica todo o corpus em vetores
2. Armazena em um índice vetorial (FAISS, HNSW, ScaNN)

Pipeline online:

1. Codifica a *query*
2. Busca *top-k* por similaridade cosseno
3. Retorna os documentos (ou passa para um *reranker*)

Fundamental para RAG

RAG (*Retrieval-Augmented Generation*):

1. Recupera documentos relevantes (**bi-encoder**)
2. Alimenta um LLM com eles
3. LLM gera resposta *grounded*

Tarefa: *Clustering*

Objetivo: agrupar documentos semanticamente similares **sem rótulos**. **Pipeline**:

1. Codifica os documentos
2. Aplica k-means, *agglomerative clustering* ou HDBSCAN
3. Extrai tópicos / rótulos por *cluster*

Aplicações:

- Descoberta de tópicos em *feedback* de usuários
- Organização de *base* de conhecimento
- Detecção de spam / clones
- Visualização exploratória (com UMAP/t-SNE)

Tarefa: Classificação via *Linear Probe*

Ideia: congelar o *encoder*, treinar apenas um classificador linear sobre os *embeddings*.
Receita:

1. Codifica n exemplos rotulados em vetores
2. Treina *logistic regression* ou SVM
3. Avalia em *test set*

Benchmark de qualidade de embeddings

O *linear probe accuracy* é uma medida padrão de **quão útil** um *embedding* é para tarefas *downstream*:

- Alta acurácia → informação linear-separável presente
- Baixa acurácia → *embedding* ruim ou precisa de cabeça não-linear

Tarefa: *Semantic Textual Similarity* (STS)[11]

Prever um *score* de similaridade em $[0, 5]$ para um par de frases.

Exemplos do *benchmark* STS

Frase A	**Frase B**	**Score**
Um homem está tocando piano.	Um cara toca teclado.	4.2
Um gato preto dorme.	Um cão branco corre.	0.8
A menina está comendo.	A garota está jantando.	3.9

- **Avaliação**: **correlação de Spearman** entre cosseno predito e *score* humano
- Mede quão bem o espaço de *embeddings* reflete similaridade semântica
- Não requer *fine-tuning*: avalia diretamente a similaridade cosseno

[11] Daniel Cer et al. "SemEval-2017 Task 1: Semantic Textual Similarity Multilingual and Crosslingual Focused Evaluation". Em: SemEval. 2017, pp. 1–14.

Spearman vs Pearson: Por quê?

Pearson: correlação **linear**

$$r = \frac{\text{cov}(x, y)}{\sigma_x \sigma_y}$$

Exige que a relação entre x e y seja **linear**.

Spearman: correlação **monotônica**

$$\rho = \text{Pearson}(\text{rank}(x), \text{rank}(y))$$

Exige apenas que a **ordem** seja preservada.

Por que Spearman em STS?

- *Scores* humanos em $[0, 5]$ são subjetivos; a escala **não é linear**
- Cosseno $\in [-1, 1]$ vive numa escala diferente, só o **ranking** importa
- Queremos que pares mais similares fiquem **acima** de pares menos similares

Spearman vs Pearson: Exemplo

- Suponha que o modelo preveja **o quadrado** do *score* humano ($\hat{y} = y^2$)
- **Pearson** ≈ 0.98 (penaliza a não-linearidade).
- **Spearman** $= 1.00$ (ordem idêntica)
- Com curvatura maior, Pearson degrada ainda mais, apesar do ranking **continuar perfeito**.

Humano	y	1	2	3	4	5
Modelo	$\hat{y}$	1	4	9	16	25

Tarefa: *Natural Language Inference* (NLI)[12]

Tarefa: dada uma *premissa* P e uma *hipótese* H, classificar em:

- ***Entailment***: P implica H
- ***Contradiction***: P e H se contradizem
- ***Neutral***: nem uma coisa nem outra

Premise	**Hypothesis**	**Rótulo**
Um homem joga futebol.	Uma pessoa pratica esporte.	entailment
Um homem joga futebol.	Ele está dormindo.	contradiction
Um homem joga futebol.	Ele é profissional.	neutral

Duplo papel: NLI é (1) tarefa de avaliação **e** (2) fonte de sinal de treino para SBERT, através do mapeamento *entailment* → positivo, *contradiction* → negativo duro.

[12] Samuel R. Bowman et al. "A Large Annotated Corpus for Learning Natural Language Inference". Em: EMNLP. 2015, pp. 632–642.

MTEB: *Massive Text Embedding Benchmark*

[13] O **MTEB** é o *benchmark* padrão moderno para *sentence embeddings*.

- 8 tarefas de *embedding*
  - Classification
  - Clustering
  - Pair Classification
  - Reranking
  - Retrieval
  - STS
  - Summarization
  - *Bitext Mining* (multilíngue)
- > 60 *datasets*
- > 100 línguas
- *Leaderboard* público no Hugging Face

Por que ele mudou o jogo

Antes do MTEB, cada *paper* escolhia seu próprio *benchmark* favorito.

O MTEB forçou avaliação **uniforme** e revelou que modelos "*SOTA em STS*" eram muito mais fracos em *retrieval*.

Um **embedding generalista** é aquele que vai bem em **todas** as categorias.

[13] Niklas Muennighoff et al. "MTEB: Massive Text Embedding Benchmark". Em: EACL. 2023, pp. 2014–2037.

Visão Geral

Recap: Quando Usar o Quê

Encoder-only

- Classificação de texto
- NER, POS, QA extrativa
- **Busca semântica / RAG retrieval**
- *Clustering*, deduplicação
- Quando latência e custo importam
- Modelos pequenos (30M–7B) são suficientes

Decoder-only

- Geração aberta
- *Chat*, criação de conteúdo
- *Reasoning*, código
- *In-context learning* e *few-shot*
- Quando qualidade de geração importa mais que latência
- Modelos grandes (7B+) dominam

Sistemas reais usam os dois: encoder-only para indexar e buscar, decoder-only para gerar a resposta final (RAG).

Takeaways

1. **Contextual embeddings** (BERT, RoBERTa) resolvem a polissemia que atormentava Word2Vec/GloVe, e são a base de quase todas as tarefas de compreensão de texto.
2. **BERT** popularizou a receita *pré-treino MLM* → *fine-tuning*. A contribuição duradoura não é o modelo em si, mas o **paradigma**.
3. **BERT *vanilla* é ruim para similaridade de frases**. Precisamos treinar explicitamente para isso → **aprendizado contrastivo**.
4. **SBERT** mostrou como: arquitetura siamesa, *mean pooling*, perda contrastiva com *in-batch negatives* e *hard negative mining*.
5. Modelos de *embedding* modernos (**e5, bge, GTE, Nomic**) seguem a mesma receita em escala.

Referências I

[1] Samuel R. Bowman et al. “A Large Annotated Corpus for Learning Natural Language Inference”. Em: EMNLP. 2015, pp. 632–642.

[2] Daniel Cer et al. “SemEval-2017 Task 1: Semantic Textual Similarity Multilingual and Crosslingual Focused Evaluation”. Em: SemEval. 2017, pp. 1–14.

[3] Kevin Clark et al. “ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators”. Em: ICLR. 2020.

[4] Jacob Devlin et al. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. Em: NAACL-HLT. 2019, pp. 4171–4186.

[5] Pengcheng He et al. “DeBERTa: Decoding-enhanced BERT with Disentangled Attention”. Em: ICLR. 2021.

[6] Vladimir Karpukhin et al. “Dense Passage Retrieval for Open-Domain Question Answering”. Em: EMNLP. 2020, pp. 6769–6781.

[7] Zhenzhong Lan et al. “ALBERT: A Lite BERT for Self-supervised Learning of Language Representations”. Em: ICLR. 2020.

[8] Yinhan Liu et al. “RoBERTa: A Robustly Optimized BERT Pretraining Approach”. Em: arXiv preprint arXiv:1907.11692 (2019).

[9] Tomas Mikolov et al. “Distributed Representations of Words and Phrases and their Compositionality”. Em: NeurIPS. Vol. 26. 2013.

[10] Niklas Muennighoff et al. “MTEB: Massive Text Embedding Benchmark”. Em: EACL. 2023, pp. 2014–2037.

[11] Aaron van den Oord, Yazhe Li e Oriol Vinyals. “Representation Learning with Contrastive Predictive Coding”. Em: arXiv preprint arXiv:1807.03748 (2018).

[12] Jeffrey Pennington, Richard Socher e Christopher D. Manning. “GloVe: Global Vectors for Word Representation”. Em: EMNLP. 2014, pp. 1532–1543.

[13] Matthew E. Peters et al. “Deep Contextualized Word Representations”. Em: NAACL-HLT. 2018, pp. 2227–2237.

[14] Pranav Rajpurkar et al. “SQuAD: 100,000+ Questions for Machine Comprehension of Text”. Em: EMNLP. 2016, pp. 2383–2392.

[15] Nils Reimers e Iryna Gurevych. “Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks”. Em: EMNLP. 2019, pp. 3982–3992.

Referências II

[16] Florian Schroff, Dmitry Kalenichenko e James Philbin. “FaceNet: A Unified Embedding for Face Recognition and Clustering”. Em: CVPR. 2015, pp. 815–823.

[17] Erik F. Tjong Kim Sang e Fien De Meulder. “Introduction to the CoNLL-2003 Shared Task: Language-Independent Named Entity Recognition”. Em: CoNLL. 2003, pp. 142–147.

[18] Liang Wang et al. “Text Embeddings by Weakly-Supervised Contrastive Pre-training”. Em: arXiv preprint arXiv:2212.03533 (2022).

[19] Shitao Xiao et al. “C-Pack: Packed Resources For General Chinese Embeddings”. Em: SIGIR. 2024.

[20] Lee Xiong et al. “Approximate Nearest Neighbor Negative Contrastive Learning for Dense Text Retrieval”. Em: ICLR. 2021.