

IA368-DD

Deep Learning aplicado a Sistemas de Buscas

1o. semestre 2023

Professores: Roberto Lotufo e Rodrigo Nogueira

Assunto da primeira aula

- Questionário expectativa
- Introdução: Regras
- Dinâmica de grupo - Socialização
- Exercícios Próxima Aula (Aula 2):
 - Notebook/Apresentação:
 - Buscador Booleano/bag-of-words e buscador com TF-IDF
 - Leitura do artigo:
 - capítulo 1:
 - Pretrained Trans. for Text Ranking: BERT and Beyond

Roteiros das Aulas

3 horas no total, sendo:

- Colab Notebook: 4 apresentações *informais* de 15 minutos cada (1.5 horas no total, considerando discussões)
- Leitura do artigo: 2 apresentações de 15 minutos cada sobre os conceitos e contribuições mais importantes do artigo. (1 hora no total, considerando discussões)
- Discussão exercício da próxima semana (30 min)

Alunos serão escolhidos no dia pelos professores para a apresentação

Avaliação

15% - Leitura de artigos

- Apresentação

50% - Programas semanais feitos em PyTorch no Jupyter/Colab

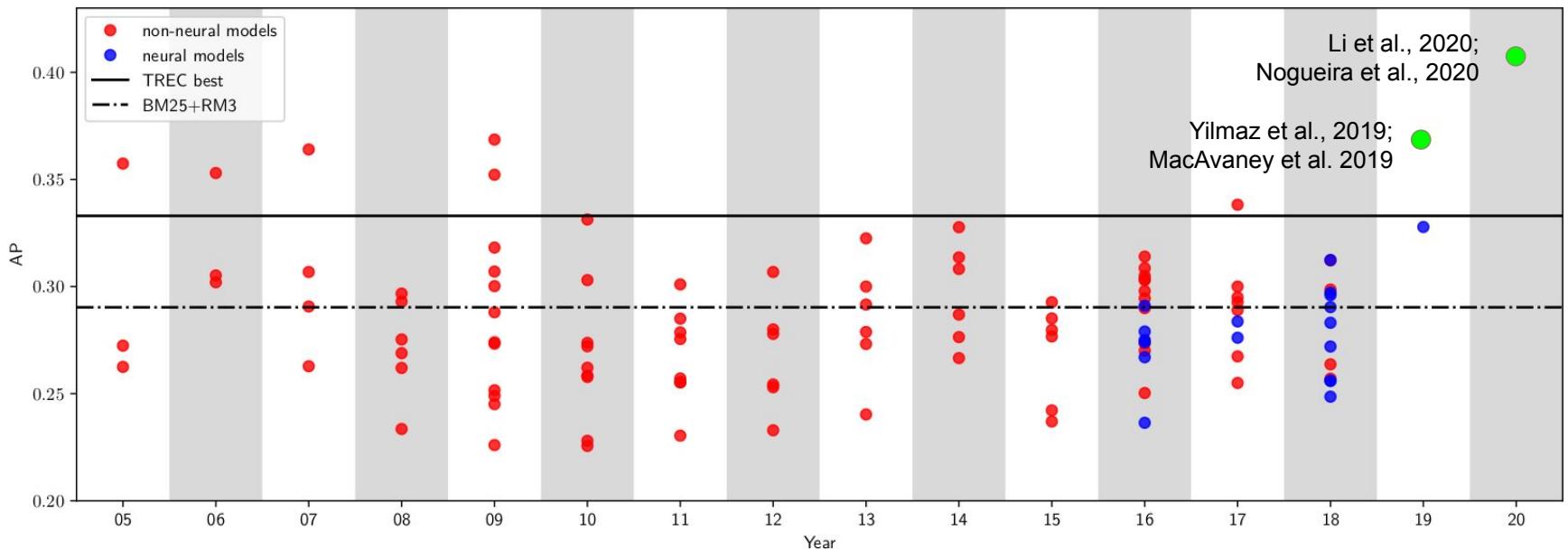
- Jupyter notebook
- Apresentação

35% - Projeto Final (4-5 semanas)

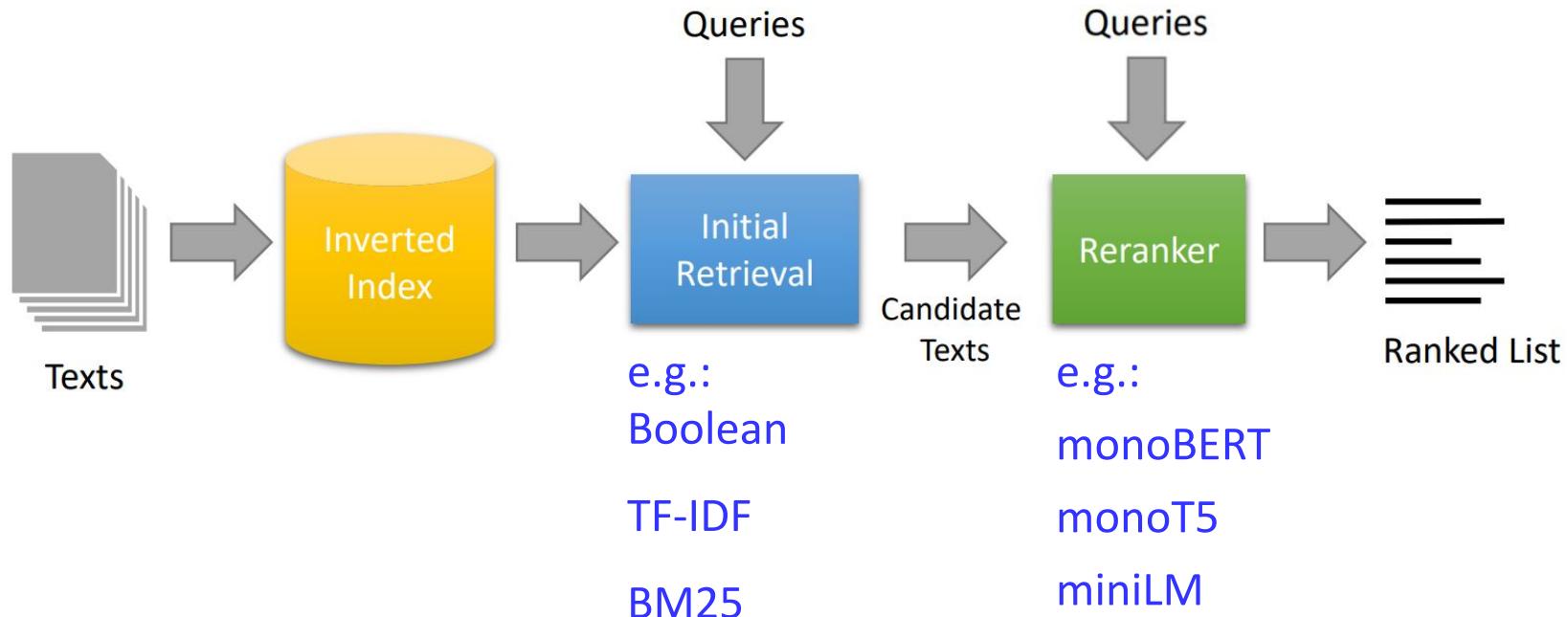
Código de conduta

Progress in Information Retrieval - Robust04

Some of them are
zero-shot!



A Simple Search Engine



Nossa experiência: nsx.ai



Login

Search

Web



for

Como plantar tulipas



Found 30 results (1.39 seconds)

Preencha metade do vaso com terra rica em nutrientes. Coloque uma fina camada de areia de construção. Depois destes passos importantes coloque o bulbo na tulipa no centro. A ponta do bulbo deve ficar voltada para cima. Cubra com terra sem encher completamente o vaso. Quando [Plantar Tulipas](#) A tulipa ...

<https://guiadasplantas.com/como-planter-tulipa/>

[Como Plantar Tulipa - Guia Das Plantas](#)

<https://www.vortexmag.net/jardim-como-e-quando-planter-tulipas/>

[Jardim: como e quando plantar tulipas? | VortexMag](#)

Cuidar do local onde serão plantadas é de extrema importância no cultivo de **tulipas**.

Prepare o solo da seguinte forma: Escolher um local ensolarado com boa drenagem Cavar a área e afrouxar o solo com cerca de 30 cm de profundidade Adicionar um pouco de adubo ou composto seco ao solo

<https://construindodecor.com.br/tulipa/>

[Tulipa: Significado, Como Plantar, Como Cultivar e Cores - ConstruindoDECOR](#)

Como **Plantar Tulipas**. Como **plantar tulipa**. A priori, use um vaso que tenha aproximadamente 20 centímetros de diâmetro e com furos. Também é necessário que o vaso tenha entre 15 e 40 centímetros de profundidade e podem ser de cerâmica ou plástico. Desse modo, um vasinho medindo 20 cm de diâmetro comp...

Resposta

Para plantar tulipas, primeiro é necessário preparar o solo, cavando a área e adicionando adubo ou composto seco ao solo. Em seguida, escolha um vaso com pelo menos 20 cm de diâmetro e furos embaixo, com entre 15 e 40 cm de profundidade. Adicione uma mistura de adubo para vasos de boa qualidade e encha-o de meio a dois terços. Coloque um grupo de bulbos no vaso, deixando apenas um pequeno espaço. Enterre os bulbos no substrato, com uma distância de mais ou menos 3 centímetros entre eles e aproximadamente 12 centímetros de profundidade. Quando chegar o inverno, o bulbo da tulipa deve ser plantado em um vaso com terra, mas sem que fique totalmente enterrado e com os brotos virados para cima. A terra deve ser regada e o vaso colocado em um saco plástico para seguir para a geladeira, onde permanecerá nos próximos seis meses.

Explicação

Os [Documento 1], [Documento 3], [Documento 4], [Documento 5], [Documento 8], [Documento 9], [Documento 11], [Documento 12], [Documento 14] e [Documento 15] fornecem informações sobre como plantar tulipas. Os [Documento 2], [Documento 6] e [Documento 13] fornecem informações sobre como preparar o solo para a plantação. O [Documento 7] e o [Documento 10] fornecem informações sobre quando plantar tulipas.

Conteúdo do curso

Aprendizado Profundo	PLN	RI
MLP	Text embeddings	BM25
Backpropagation	Modelos de linguagem	Indexação
Transformers	Modelos seq2seq	Buscadores Densos (e Approximate Nearest Neighbor)
Mecanismo de atenção	In-context learning (few-shot)	Buscadores Esparsos
Treinamento Supervisionado	Perguntas e respostas	Expansão de Documentos
Auto-supervisão	Tradução de máquina Classificadores de texto Teacher-forcing vs predição Greedy decoding, Beam Search TF-IDF/Bag-of-Words Tokenização Geração de Dados sintéticos	Expansão de Queries Rerankeadores Metricas (nDCG, MRR, MAP) Anotação de datasets Trade-offs qualidade vs velocidade Sumarização de múltiplos documentos baseado em perguntas

Artigos Relevantes

PLN	RI
<p><u>A Neural Probabilistic Language Model (Bengio et al, 2003)</u></p> <p>Word2Vec</p> <p>Attention is All you Need (2017)</p> <p>BERT (2018)</p> <p>T5 (2019)</p> <p>GPT-3 (2020)</p> <p>Scaling Laws (2020)</p> <p><u>Distilling the Knowledge in a NN (2015)</u></p> <p>PALM (2022)</p> <p><u>Chain of Thought</u></p>	<p><u>Lecture Notes on IR (Tonellootto)</u></p> <p><u>Pretrained Transformers for Text Ranking (2020)</u></p> <p><u>monoBERT (2019)</u></p> <p><u>doc2query (2019) e docT5query (2020)</u></p> <p><u>monoT5 (2020)</u></p> <p><u>DPR (2020)</u></p> <p><u>CoLBERT (2020)</u></p> <p><u>UniCOIL (2021)</u></p> <p><u>SPLADE (2021)</u></p> <p><u>NeuralSearchX (2022)</u></p> <p><u>InPars (2022)</u></p> <p><u>Visconde (2022)</u></p> <p><u>Transformer Memory as a Differentiable Search Index</u></p>

Programação dos Exercícios (sujeito à mudanças)

Aula	Exercício	Artigo	Tópicos
1	Buscador Simples: Booleano, TF-IDF, BM25	Pretrained Transformers for Text Ranking (até capítulo 1)	Indexação, Bag-of-Words, TF-IDF, BM25
2	Classificador binário: Análise de Sentimento e Ranqueamento	Pretrained Transformers for Text Ranking (desde seção 3 até o 3.2.2 (pags 44-61))	MLP, Treinamento Supervisionado, Classificadores de texto, Ranqueadores, Métricas (nDCG, MRR, MAP)
3	Aplicar LLM's Zero e Few-shot (aplicação escolhida pelo aluno)	GPT-3: Language Models are Few-Shot Learners	Tokenização, Modelos de linguagem, In-context learning (few-shot), Auto-supervisão
4	Treinamento de um modelagem de linguagem decoder-only em textos portugues	GPT-2 Language Models are Unsupervised Multitask Learners	Transformers, Mecanismo de atenção, Backpropagation
5	Modelo seq2seq: T5 para expansão de documentos (doc2query)	doc2query e docTTTTTquery	Modelos seq2seq, Teacher-forcing vs predição, Tradução de máquina, Expansão de Documentos
6	Buscadores Densos: DPR, encoder-only	DPR e ColBERT (opcional)	Buscadores Densos, Text Embeddings, Approximate Nearest Neighbor
7	Buscadores Esparsos: SPLADE	SPLADE e SPLADE-v2 (opcional)	Buscadores Esparsos, Expansão de Queries, Tokenização
8	InPars: Adaptação de modelos para novas tarefas	InPars e InPars-v2 (opcional)	Anotação de datasets, Geração de Dados sintéticos
9	Sistema multi-estágio + destilação		Trade-offs qualidade vs velocidade
10	Multi-document QA: Visconde		Perguntas e respostas, Sumarização de múltiplos documentos baseado em perguntas

Importante!

- É esperado que cada aluno assine o Colab Pro (~60 reais/mês) ou equivalente
- Cursos passados mostraram que é muito difícil realizar os exercícios usando a versão grátis do Colab.
- Caso precisem de ajuda financeira, contate os professores.

Dicas

- Usar ChatGPT para tudo
- Ser crítico nas respostas do ChatGPT
- Notebook com bastante documentação (texto explicando as células ou bloco de células)

Roteiro para Apresentação do Notebook

A apresentação deve cobrir *pelo menos* 3 dos 7 itens abaixo:

1. Explicação de conceitos importantes do exercício feito
2. Técnicas para garantir que a implementação está correta
3. Truques de código que funcionaram
4. Problemas e soluções no desenvolvimento
5. Resultados interessantes/inesperados
6. Uma dúvida "básica" que você ou os colegas possam ter
7. Um tópico "avançado" para discutirmos

Roteiro para Apresentação do Artigo

A apresentação deve cobrir *pelo menos* 2 dos 5 itens abaixo:

1. Explicação de conceitos importantes do artigo
2. A contribuição do artigo
3. Resultados interessantes/inesperados
4. Uma dúvida "básica" que você ou os colegas possam ter
5. Um tópico "avançado" para discutirmos

Exercício desta semana

1. Usar o BM25 implementado pelo pyserini para buscar queries no TREC-DL 2020
 - Documentação referência:
<https://github.com/castorini/pyserini/blob/master/docs/experiments-msmarco-passage.md>
2. Implementar um buscador booleano/bag-of-words.
3. Implementar um buscador com TF-IDF
4. Avaliar implementações 1, 2, e 3 no TREC-DL 2020 e calcular o nDCG@10

Nos itens 2 e 3:

- Fazer uma implementação que suporta buscar eficientemente *milhões* de documentos.
- Não se pode usar bibliotecas como sklearn, que já implementam o BoW e TF-IDF.

Perguntas a serem respondidas na atividade da aula 2 (formulario)

Quais os problemas que sistemas de busca resolvem?

Qual a diferença entre processamento de linguagem natural e recuperação de informações?

Qual o algoritmo de buscas mais usado e por quê?

Quais são as limitações do BM25?

Qual a estrutura de dados usada pelo BM25 que permite que busquemos eficientemente milhões de documentos?

Se aumentarmos em 10x o número de documentos na coleção, de quantas vezes vai aumentar o tempo para responder uma query?

Como avaliamos a qualidade de um sistema de buscas?

Quais metricas são comumente usadas?

Quais os problemas quando usamos um dataset de avaliação de sistemas de buscas

Sparse Retrieval with BM25

Widely used in Academia and Industry

Works in 2 phases:

1. **Indexing (offline)**: inverted index construction: a dictionary whose keys are words and values are documents that contain those words;
2. **Retrieval/ranking**: for each word q_i in the query Q , compute a score for each document D that contains the word:

$$\text{score}(D, Q) = \sum_{i=1}^n \text{IDF}(q_i) \cdot \frac{f(q_i, D) \cdot (k_1 + 1)}{f(q_i, D) + k_1 \cdot \left(1 - b + b \cdot \frac{|D|}{\text{avgdl}}\right)},$$

```
{  
  "apple": [doc_32, doc_5],  
  "house": [doc_85, doc_9],  
  ...  
}
```

Suffers from the "Vocabulary mismatch problem": *car* and *automobile* are completely different to BM25

However, a hard-to-beat algorithm

Index construction

doc1: "car banana apple"



```
inverted_index =  
    { "car": [doc1],  
      "banana": [doc1],  
      "apple": [doc1]  
    }
```

doc2: "car girl boy"



```
inverted_index =  
    { "car": [doc1, doc2],  
      "banana": [doc1],  
      "apple": [doc1],  
      "girl": [doc2],  
      "boy": [doc2]  
    }
```

doc3: "house"



```
inverted_index =  
    { "car": [doc1, doc2],  
      "banana": [doc1],  
      "apple": [doc1],  
      "girl": [doc2],  
      "boy": [doc2],  
      "house": [doc3]  
    }
```

Retrieval time

query: "car banana"

Returns:

- doc1: score = 2 -> rank 1
- doc2: score = 1 -> rank 2

```
inverted_index =  
    { "car": [doc1, doc2],  
      "banana": [doc1],  
      "apple": [doc1],  
      "girl": [doc2],  
      "boy": [doc2],  
      "house": [doc3]  
    }
```

query: "automobile"

Returns:

- None

Assunto da segunda aula

- Discussão do Exercício: Buscador Booleano e BM25
- Discussão do Artigo: Capítulo 1 do Pretrained Trans. for Text Ranking: BERT and Beyond
- Exercícios Próxima Aula
 - Notebook/Apresentação: Rerankeamento com classificador neural
 - Leitura do artigo

Aviso em relação aos projetos entregues

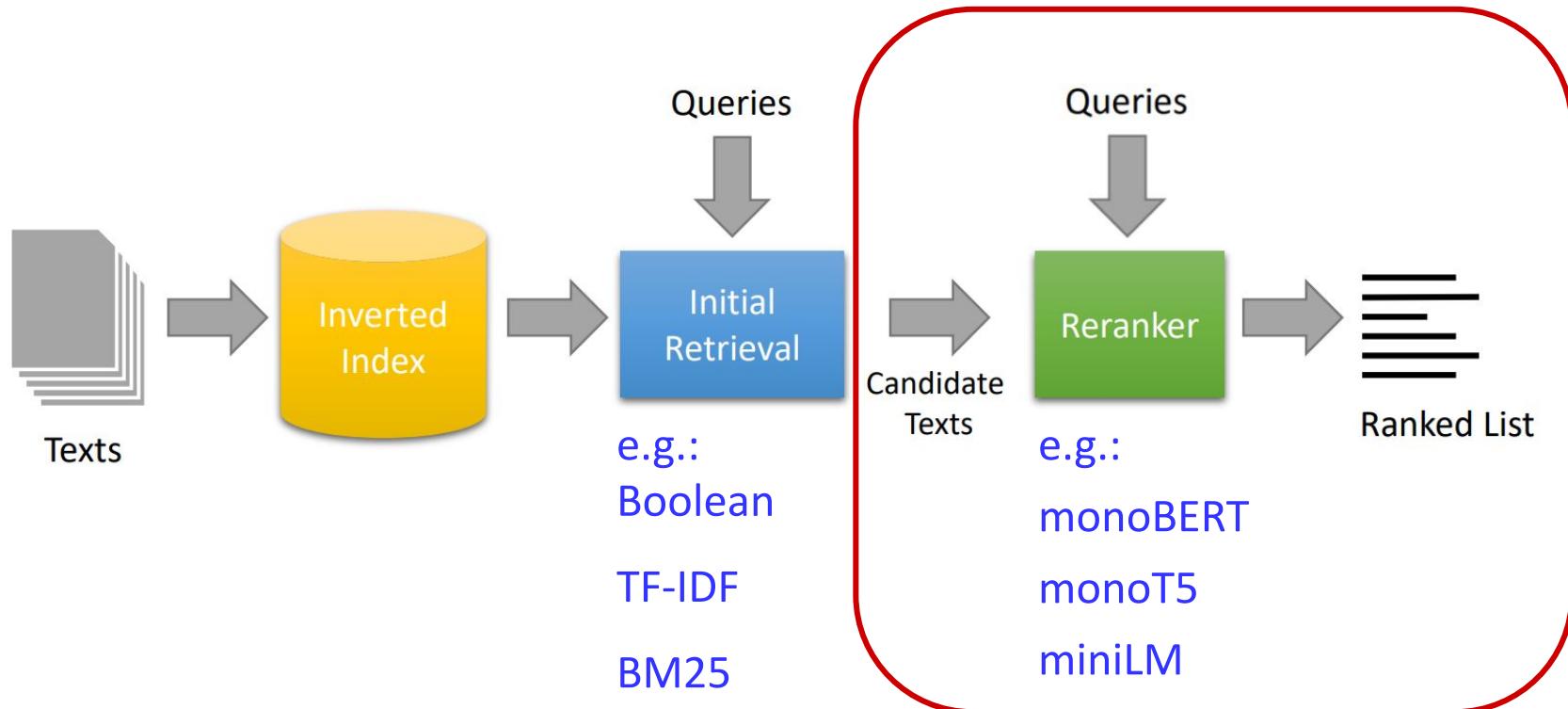
Na primeira aula, tivemos duas apresentações de projetos entregues no processo seletivo. Na aula, utilizamos este projeto como forma de motivação e introdução ao tema de buscas, porém os projetos não foram corrigidos e detectamos vários erros conceituais que ao longo do curso, vocês mesmos serão capazes de corrigir.

Assim, fica aqui uma tarefa. Criem um relatório incremental ao projeto que, ao longo do curso, vocês vão verificando os erros feitos e eventualmente corrigindo-os. Durante o curso vocês serão expostos a todos os problemas usuais de um sistema de buscas, com uma boa profundidade.

Conclusões sobre o exercício

- Tokenizer importa muito: pyserini.analyzer ainda é o melhor. Outras tecnicas como uso do NLTK podem deixar o processo de indexação bastante lento e não trazem melhorias. NLTK é muito para selecionar stopwords.
- Quem é melhor, BoW, Boolean, TF-IDF ou BM25?
 - BM25
- O que é o nDCG?
- O que é o MAP?
-
- Alguém testou com o ElasticSearch?
- Alguém implementou o BM25 e viu se bate com o do pyserini?
 - Leandro Carisio fez e quase bateu
- Alguém ajustou os hiperparametros k1 e b? Os resultados têm diferença em relação aos valores padrões?
 - Melhora, mas é preciso cautela: overfit pode ocorrer.

A Simple Search Engine



Exercício 2 desta semana: cross-encoder

Rerankeamento usando um modelo estilo-BERT com o treinamento no dataset do MS MARCO e avaliação no TREC-DL 2020

O treinamento é igual ao de um classificador binário, que será feito por vocês.

O que muda é a forma de avaliação: rerankeadores precisam ser alimentados com documentos candidatos (ex: trazidos pelo BM25 - exercício aula 1)

Sugestão: usar este dataset reduzido do MS MARCO como treinamento, com 10k triplas (query, passagem relevante, passagem não-relevante):

- https://storage.googleapis.com/unicamp-dl/ia368dd_2023s1/msmarco/msmarco_triples.train.tiny.tsv

Sugestão: usar miniLM (modelo BERT pequeno, 5x mais rápido) para começar o finetuning:

<https://huggingface.co/nreimers/MiniLM-L6-H384-uncased> pois oferece um bom compromisso entre qualidade e velocidade.

Sugestão: usar este notebook como base

- Análise de sentimentos (dataset IMDB) usando um modelo estilo BERT:
https://colab.research.google.com/drive/10etP7Lb915EC-uEuf1IKC8DYkyg_om6-?usp=sharing

Sugestão de debug: usar este minilm para ver se consegue ndcg ~0.70: <https://huggingface.co/cross-encoder/ms-marco-MiniLM-L-6-v2>

Sugestão: fazer overfit em um batch: treinar por 200 épocas um único batch, e ver se consegue loss=0, e accuracy=100%, ou ndcg=1

Run: saída do retriever

query 1, doc id #39, rank 1

query 1, doc id #192, rank 2

...

query 1, doc id #9181, rank 1000

query 2, doc id #1746, rank 1

...

query 2, doc id #83, rank 1000

...

Qrels: anotação humana

query 1, doc id #192, relevance: 1

query 1, doc id #28, relevance: 2

query 1, doc id #9181, relevance: 0

query 2, doc id #83, relevance: 0

query 2, doc id #1746, relevance: 1

...

Importante: nem todos os query-docs da run tem um correspondente no qrels.
Neste caso, assumimos relevance=0

Perguntas a serem respondidas na atividade da aula 3
(formulario)

Assunto da terceira aula: LLM - aprendizado contextual

- 8:10-8:40: Discussão do Artigo (30 minutos, 2 apresentações)
- 8:50-10:20: Discussão dos Exercícios (1.5 horas, 4 apresentações)
 - 10 min de apresentação com 5 minutos de discussão (possivelmente intercalados com a apresentação)
- 10:30-11:30: Exercícios Próxima Aula (1 hora)
 - Notebook/Apresentação
 - Leitura do artigo

Assuntos aprendidos na leitura do artigo

Para saber sobre tokenizadores: <https://huggingface.co/course/chapter2/4?fw=pt>

SciBERT usa tokenizadores especiais: <https://aclanthology.org/D19-1371.pdf>

Blog post discutindo que o vetor CLS ou a media dos vetores dos tokens não é util para tarefas de busca:

<https://bergum.medium.com/how-not-to-use-bert-for-search-ranking-4586716428d9>

Duvidas sobre BoW, TF-IDF e BM25

- 1) Na busca booleana, é necessário que todos os termos da query estejam presentes? Nas explicações que consegui com o chatgpt, e também em alguns slides, não ficava claro, havendo a ênfase na explicação do uso dos operadores lógicos; fiz a implementação considerando AND e o resultado ficou ainda pior do que com OR (entre os termos).
- 2) Com o BOW também se constrói vetores de termos para cada documento?
- 3) Outra dúvida que considero básica é sobre a aplicação do tf-idf e a construção de um vetor de termos para cada documento: o vetor de uma query vai ser comparado apenas com os vetores dos documentos que retornarem do índice invertido? Como calcular o tf-idf para os termos das queries que não estiverem no índice?

Exercício desta semana: zero e few-shot

1. O aluno irá escolher uma tarefa para resolver de maneira zero ou few-shot. Sugestões:
 - Classificação de textos (ex: análise de sentimos (IMDB))
 - Predizer se uma passagem/parágrafo é relevante para uma pergunta/query
 - Se uma resposta predita por um sistema de QA ou sumarizador é semanticamente igual à resposta ground-truth
2. É importante ter uma função de avaliação da qualidade das respostas do modelo few-shot. Por exemplo, acurácia.
3. É possível criar um pequeno dataset de teste manualmente (ex: com 10 à 100 exemplos)
4. Usar a API do LLAMA fornecida por nós (licença exclusiva para pesquisa). [Colab demo da API do LLAMA](#) (obrigado, Thales Rogério)
5. Opcionalmente, usar a API do **code-davinci-002**, que é de graça e trás resultados muito bons.
 - a. CUIDADO: NÃO USAR O **TEXT-DAVINCI-002/003**, que é pago
6. Opcionalmente, usar a API do ChatGPT (gpt-3.5-turbo) que é barata: ~1 centavo de real por 1000 tokens (uma página)
7. Opcionalmente, usar o Alpaca: <https://alpaca-ai.ngrok.io/>

Dicas:

- Teste com zero-shot E few-shot.
- No few-shot, faça testes com e sem instruções no cabeçalho (explicação da tarefa, ex: "Traduza de Ingles para Portugues"). Pode ser que sem a instrução o modelo até funcione melhor.
- Siga sempre um padrão ao criar os exemplos few-shot. Aqui tem uma pagina com dicas para prompt engineering:
<https://help.openai.com/en/articles/6654000-best-practices-for-prompt-engineering-with-openai-api>

Assunto da quarta aula: Treino de Modelo de Linguagem

- 8:10-8:40: Discussão do Artigo (30 minutos, 2 apresentações)
- 8:50-10:20: Discussão dos Exercícios (1.5 horas, 4 apresentações)
 - 10 min de apresentação com 5 minutos de discussão (possivelmente intercalados com a apresentação)
- 10:30-11:30: Exercícios Próxima Aula (1 hora)
 - Notebook/Apresentação
 - Leitura do artigo

Exercício desta semana

Treinar um modelo de linguagem em dados em portugues

- Avaliar o modelo usando a perplexidade, que é simplesmente a exponencial de todas as losses do dataset de validação
- Iremos treinar o modelo para prever o próximo token dado os anteriores (também conhecido como Causal Language Modeling). Não confundir com o Masked Language Modeling (MLM), que consiste em prever tokens mascarados em uma dada sequência (ex: BERT's MLM)

Dicas:

- Usar como ponto de partida o modelo [OPT-125M](#), que já foi treinado em 300B de tokens (maioria em Inglês)
- Usar este dataset reduzido do mc4 portugues, com ~300M de tokens:
`gs://unicamp-dl/ia025a_2022s1/aula9/sample-1gb.txt`

Perguntas para ajudar no entendimento do exercício

- No exercício de treinar o rerankeador, havia um ground-truth (true/false) para cada sequência de entrada (query+passagem). Neste exercício de treinar o modelo de linguagem, quantos ground-truths existem para cada sequência de entrada, assumindo que cada uma tem 512 tokens?
- Porque o Transformer é mais eficiente no treinamento da tarefa de modelagem de linguagem que o modelo do [Bengio et al. 2003](#) ou RNNs e LSTMs?
- Um modelo aleatorio (sem treino) avaliado em um problema de classificação com 20 classes terá quanto de perplexidade?

Entendo a perplexidade

Modelo aleatorio

$$p=1/20$$

$$\text{xent loss} = -\ln p_{\text{classe_correta}} = -\ln (1/20) = \ln 20$$

$$\text{PPL} = \exp(\text{xent}) = \exp(\ln(20)) = 20$$

Modelo Perfeito:

$$p=1$$

$$\text{xent loss} = -\ln p_{\text{classe_correta}} = -\ln (1) = 0$$

$$\text{PPL} = \exp(0) = 1$$

Modelo do contra:

$$p=0$$

$$\text{xent loss} = -\ln p_{\text{classe_correta}} = -\ln (0) = \text{inf}$$

$$\exp(\text{inf}) = \text{inf}$$

Código gerado pelo GPT-4 para o exercício desta semana

```
1 import torch
2 from datasets import load_dataset
3 from transformers import (
4     AutoTokenizer,
5     AutoModelForCausalLM,
6     DataCollatorForLanguageModeling,
7     Trainer,
8     TrainingArguments,
9 )
10
11 # Configurações do treinamento
12 model_name = "OPT-125M"
13 data_file = "./data.txt"
14 output_dir = "./trained_model"
15
16 # Carregar tokenizer e modelo
17 tokenizer = AutoTokenizer.from_pretrained(model_name)
18 model = AutoModelForCausalLM.from_pretrained(model_name)
19
20 # Carregar e tokenizar o dataset
21 dataset = load_dataset("text", data_files=data_file)
22 tokenized_dataset = dataset.map(
23     lambda x: tokenizer(x["text"], truncation=True, padding="max_length", max_length=128),
24     batched=True,
25     num_proc=4,
26     remove_columns=["text"],
27 )
28
29 # Preparar o Trainer
30 training_args = TrainingArguments(
31     output_dir=output_dir,
32     num_train_epochs=3,
33     per_device_train_batch_size=16,
34     per_device_eval_batch_size=16,
35     evaluation_strategy="epoch",
36     save_strategy="epoch",
37     logging_strategy="epoch",
38     learning_rate=2e-5,
39     weight_decay=0.01,
40     fp16=True,
```

```
41 )
42
43 data_collator = DataCollatorForLanguageModeling(tokenizer=tokenizer, mlm=False)
44
45 trainer = Trainer(
46     model=model,
47     args=training_args,
48     train_dataset=tokenized_dataset["train"],
49     eval_dataset=tokenized_dataset["train"],
50     data_collator=data_collator,
51 )
52
53 # Treinar e avaliar o modelo
54 trainer.train()
55 eval_results = trainer.evaluate()
56
57 # Calcular a perplexidade
58 perplexity = torch.exp(torch.tensor(eval_results["eval_loss"]))
59 print(f"Perplexidade: {perplexity.item()}")
60
61 # Salvar o modelo treinado e o tokenizer
62 model.save_pretrained(output_dir)
63 tokenizer.save_pretrained(output_dir)
```

Mais didático, por favor...



Voce poderia rescrever o codigo acima mas de maneira mais explicita, sem usar o `DataCollatorForLanguageModeling`. Mais especificamente, gostaria de entender o que acontece dentro dele



Claro! Vamos escrever uma função personalizada para lidar com a criação dos lotes e gerar os tensores necessários para a tarefa de next-token prediction.

Aqui está o código com a função personalizada `prepare_batch` em vez de usar o `DataCollatorForLanguageModeling`:

```
1 import torch
2 from torch.utils.data import DataLoader
3 from datasets import load_dataset
4 from transformers import (
5     AutoTokenizer,
6     AutoModelForCausalLM,
7     Trainer,
8     TrainingArguments,
9 )
10
11 def prepare_batch(batch, tokenizer):
12     input_ids = tokenizer(batch["text"], truncation=True, padding="max_length",
13     max_length=128, return_tensors="pt")["input_ids"]
14     labels = input_ids[:, 1:].clone()
15     input_ids = input_ids[:, :-1]
16
17     return {"input_ids": input_ids, "labels": labels}
18
19 # Configurações do treinamento
20 model_name = "OPT-125M"
21 data_file = "./data.txt"
22 output_dir = "./trained_model"
23
24 # Carregar tokenizer e modelo
25 tokenizer = AutoTokenizer.from_pretrained(model_name)
26 model = AutoModelForCausalLM.from_pretrained(model_name)
27
28 # Carregar e tokenizar o dataset
29 dataset = load_dataset("text", data_files=data_file)
30 tokenized_dataset = dataset.map(
31     lambda x: tokenizer(x["text"], truncation=True, padding="max_length", max_length=128),
32     batched=True,
33     num_proc=4,
34     remove_columns=["text"],
35 )
```

```
36 # Preparar DataLoader
37 train_dataloader = DataLoader(
38     tokenized_dataset["train"], batch_size=16, shuffle=True, collate_fn=lambda batch:
39     prepare_batch(batch, tokenizer)
40 )
41 eval_dataloader = DataLoader(
42     tokenized_dataset["train"], batch_size=16, shuffle=False, collate_fn=lambda batch:
43     prepare_batch(batch, tokenizer)
44 )
45 # Preparar o Trainer
46 training_args = TrainingArguments(
47     output_dir=output_dir,
48     num_train_epochs=3,
49     per_device_train_batch_size=16,
50     per_device_eval_batch_size=16,
51     evaluation_strategy="epoch",
52     save_strategy="epoch",
53     logging_strategy="epoch",
54     learning_rate=2e-5,
55     weight_decay=0.01,
56     fp16=True,
57 )
58 trainer = Trainer(
59     model=model,
60     args=training_args,
61     train_dataset=None,
62     eval_dataset=None,
63     train_dataloader=train_dataloader,
64     eval_dataloader=eval_dataloader,
65 )
66
67 # Treinar e avaliar o modelo
68 trainer.train()
69 eval_results = trainer.evaluate()
70
71 # Calcular a perplexidade
72 perplexity = torch.exp(torch.tensor(eval_results["eval_loss"]))
73 print(f"Perplexidade: {perplexity.item()}")
74
75 # Salvar o modelo treinado e o tokenizer
76 model.save_pretrained(output_dir)
77 tokenizer.save_pretrained(output_dir)
```

A função `prepare_batch` recebe um lote de exemplos e o `tokenizer`. Primeiro, ela tokeniza as sequências de texto, trunca e preenche-as para terem o mesmo comprimento (`max_length=128`). Em seguida, as sequências são convertidas em tensores do tipo `input_ids`.

Para a tarefa de next-token prediction, as entradas do modelo são os tokens até o penúltimo token de cada sequência, enquanto as saídas esperadas (labels) são os tokens do segundo até o último. Portanto, a função `prepare_batch` cria os tensores `input_ids` e `labels` com um deslocamento de 1 em relação um ao outro.

Avaliação de Modelos de Linguagem

- Acurácia?
 - 1 caso o modelo acertou a palavra, 0 caso contrário
 - "Nesta garagem cabem dois ____."
 - Predição do modelo: $P(\text{"carros"}) = 0.90$, $P(\text{"automoveis"}) = 0.08$, ...
 - Palavra original: "automóveis"
 - Acurácia = 0
- Erro Quadrático Médio (MSE)?
 - $\text{MSE} = \sum_i [P_{\text{truth}} - P_{\text{pred}}(w_i)]^2$
 - No exemplo: se $P(\text{automóveis}) = 0.9$, então, $\text{MSE} = 0.01$, assumindo $P_{\text{truth}} = 1$.
 - Funcionaria, mas seria bom usarmos uma métrica que penaliza mais quando o modelo colocou uma probabilidade muito baixa na palavra correta.

Avaliação de Modelos de Linguagem

Dan Jurafsky



Perplexity

The best language model is one that best predicts an unseen test set

- Gives the highest $P(\text{sentence})$

Perplexity is the inverse probability of the test set, normalized by the number of words:

Chain rule:

For bigrams:

Minimizing perplexity is the same as maximizing probability

$$PP(W) = P(w_1 w_2 \dots w_N)^{-\frac{1}{N}}$$

$$= \sqrt[N]{\frac{1}{P(w_1 w_2 \dots w_N)}}$$

$$PP(W) = \sqrt[N]{\prod_{i=1}^N \frac{1}{P(w_i | w_1 \dots w_{i-1})}}$$

$$PP(W) = \sqrt[N]{\prod_{i=1}^N \frac{1}{P(w_i | w_{i-1})}}$$

Avaliação de Modelos de Linguagem

- Perplexidade (PPL) e Entropia Cruzada (ENT):
 - $PPL = e^{XENT}$
 - $XENT = - \frac{1}{n} \sum_n \ln P(w_n)$
- Sentença S: "Eu gosto de pizza"
- $XENT(S) = \frac{1}{n} [-\ln P("Eu" | vazio) - \ln P("gosto" | "Eu") - \ln P("de" | "Eu gosto") - \ln P("pizza" | "Eu gosto de"))]$
- Métrica: por que usar PPL ao invés de entropia cruzada?
 - possível resposta:
<https://towardsdatascience.com/perplexity-intuition-and-derivation-105dd481c8f3>
- Usar PPL como função de custo?
- Sugestão de leitura: Meena (Google Chatbot) <https://arxiv.org/pdf/2001.09977.pdf>

Em diálogos, perplexidade e avaliação humana tem forte correlação

Propriedades interessantes da Perplexidade

Qual é a perplexidade inicial de um modelo de linguagem com 10k palavras no vocabulário?

$$\text{Prob média} = 1/10k = 10^{-4}$$

$$XENT = - \frac{1}{n} \sum_n \ln P(w_n) = - \frac{1}{n} \sum_n \ln 10^{-4} \sim -\ln 10^{-4}$$

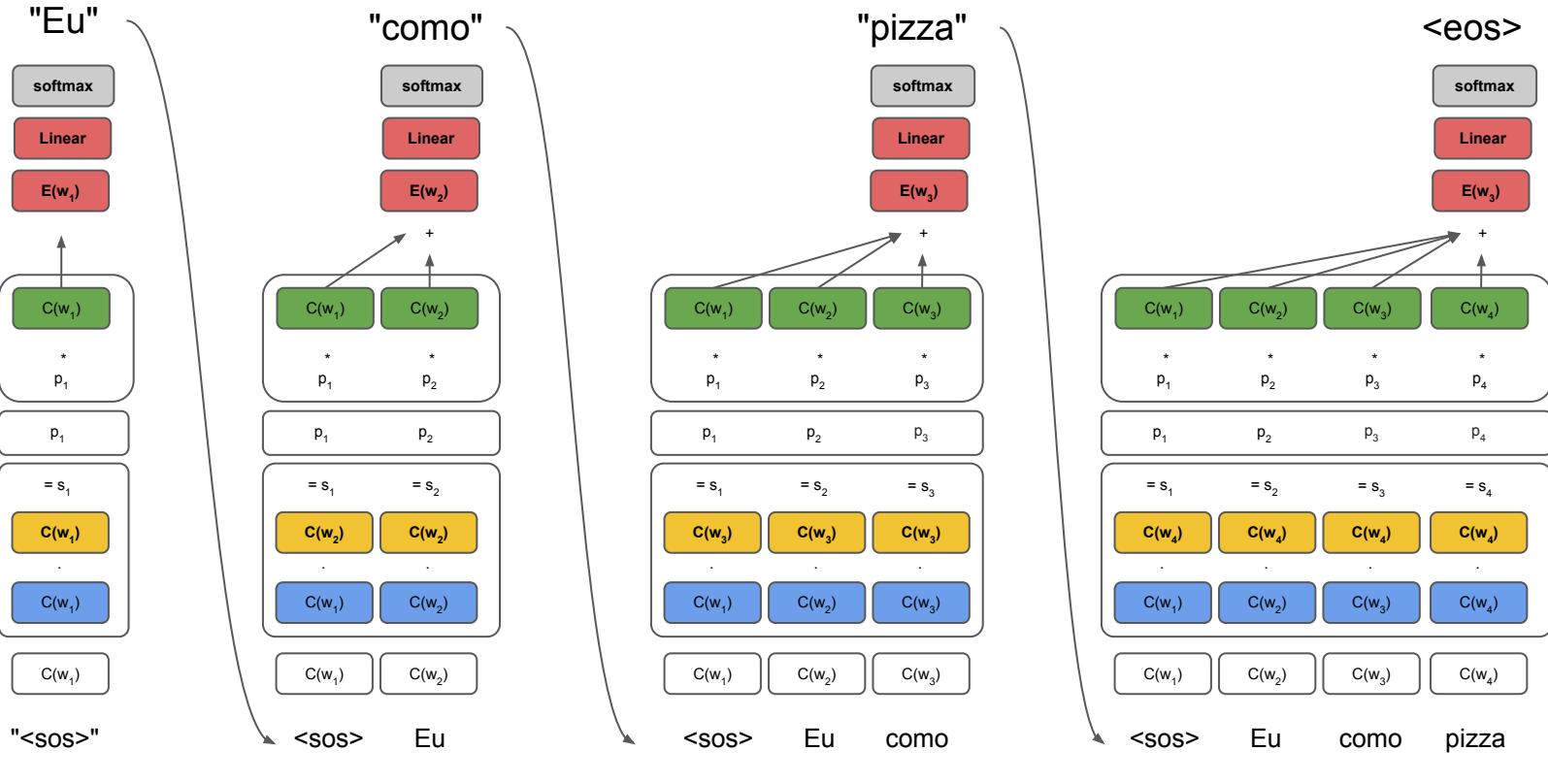
$$PPL = \exp^{XENT} = \exp(-\ln 10^{-4}) = \exp(\ln 10^4) = 10^4$$

A perplexidade é igual ao número de palavras no vocabulário!

Ou seja, se o seu modelo de linguagem tem perplexidade de N, significa que ele é igual a uma classificador aleatório de N classes

Diferença entre logits e probabilidades

Como um decoder-only (ex: GPT) gera texto?



Step 1

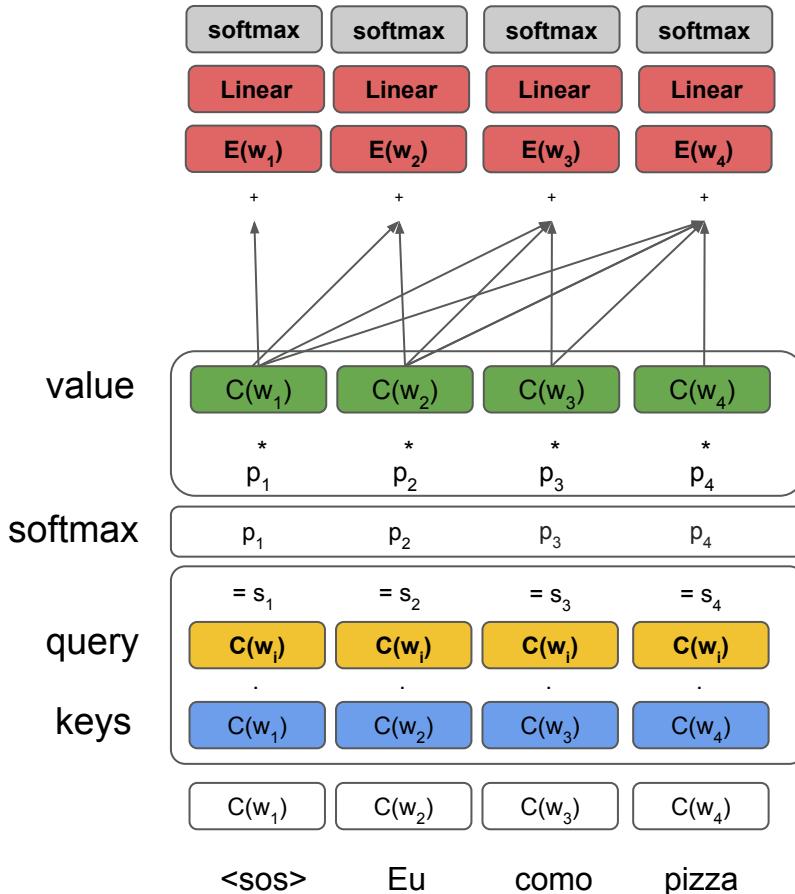
Step 2

Step 3

Step 4

Em treino

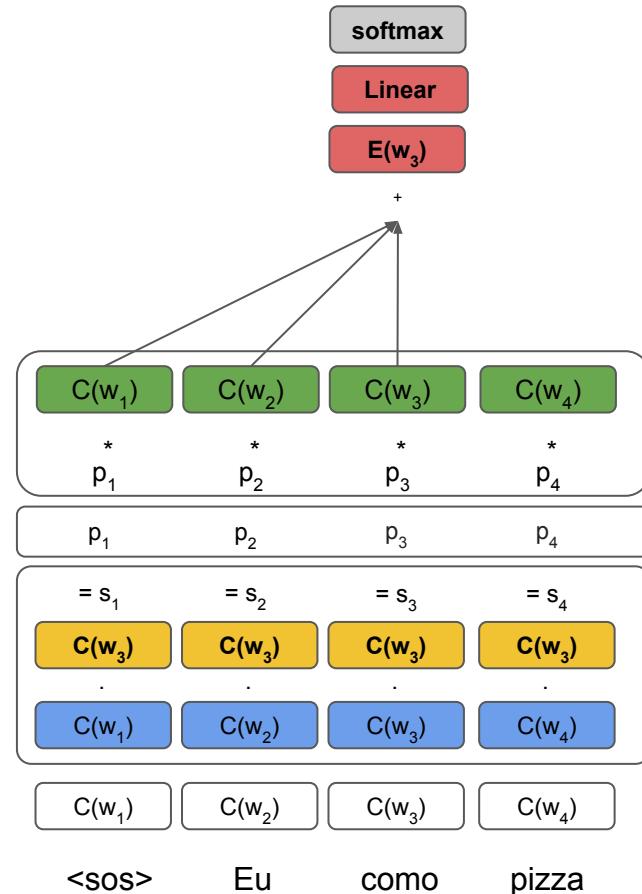
$-\log p("Eu") \quad -\log p("como") \quad -\log p("pizza") \quad -\log p("<\text{eos}>")$



Entrada: <sos> Eu como pizza

Saída: Eu como pizza <eos>

$-\log p("pizza")$



Número de operações de um modelo auto-regressivo

step	Query	keys/values					Custo
		<sos>	Eu	gosto	de	Pizza	
1	<sos>	■					1
2	Eu		■				2
3	gosto			■			3
4	de				■		4
5	pizza					■	5

Qual o custo computacional para gerar N tokens?

Nota 1: Em treino, a loss pode ser calculada em apenas um *forward pass* da rede, ou seja, não precisa de laço. Isso porque já sabemos os target tokens (y_1, \dots, y_N)

Exemplos de Tamanhos Variáveis: Padding

3 docs "originais":

Doc 1:	$w_{1,1}$	$w_{1,2}$	$w_{1,3}$			
Doc 2:	$w_{2,1}$	$w_{2,2}$	$w_{2,3}$	$w_{2,4}$	$w_{2,5}$	$w_{2,6}$
Doc 3:	$w_{3,1}$	$w_{3,2}$				

Input (assumindo context_size=4)

Exemplo 1:	<sos>	$w_{1,1}$	$w_{1,2}$	$w_{1,3}$
Exemplo 2:	<sos>	$w_{2,1}$	$w_{2,2}$	$w_{2,3}$
Exemplo 3:	<sos>	$w_{3,1}$	$w_{3,2}$	PAD
Exemplo 4:	<sos>	$w_{2,5}$	$w_{2,6}$	PAD

Target

$w_{1,1}$	$w_{1,2}$	$w_{1,3}$	<eos>
$w_{2,1}$	$w_{2,2}$	$w_{2,3}$	$w_{2,4}$
$w_{3,1}$	$w_{3,2}$	<eos>	PAD
$w_{2,5}$	$w_{2,6}$	<eos>	PAD

Exemplos de Tamanhos Variáveis: Com <UNK>

3 docs "originais":

Doc 1:	$w_{1,1}$	$w_{1,2}$	$w_{1,3}$			
Doc 2:	$w_{2,1}$	$w_{2,2}$	<unk>	$w_{2,4}$	$w_{2,5}$	$w_{2,6}$
Doc 3:	$w_{3,1}$	$w_{3,2}$				

Input (assumindo context_size=4)

Exemplo 1:	<sos>	$w_{1,1}$	$w_{1,2}$	$w_{1,3}$
Exemplo 2:	<sos>	$w_{2,1}$	<UNK>	$w_{2,3}$
Exemplo 3:	<sos>	$w_{3,1}$	$w_{3,2}$	PAD
Exemplo 4:	<sos>	$w_{2,5}$	$w_{2,6}$	PAD

Target

$w_{1,1}$	$w_{1,2}$	$w_{1,3}$	<eos>
$w_{2,1}$	PAD	$w_{2,3}$	$w_{2,4}$
$w_{3,1}$	$w_{3,2}$	<eos>	PAD
$w_{2,5}$	$w_{2,6}$	<eos>	PAD

Assunto da quinta aula

- Discussão do Artigo: GPT-2
- Discussão dos Exercícios: Treinamento de um modelo em corpus em português
 - 10 min de apresentação com 5 minutos de discussão (possivelmente intercalados com a apresentação)
- 10:30-11:30: Exercícios Próxima Aula: doc2query
 - Notebook/Apresentação
 - Leitura do artigo: doc2query

Perguntas interessantes do exercício da aula passada

Pedro:

- Quais as melhores formas de se retornar a PPL entre modelos com diferentes tokenizadores, uma vez que a métrica depende do vocabulário?

Thiago:

- Qual a correlação da perplexidade com a eficiência do modelo nas avaliações zero/few shot ?
Quanto menor a perplexidade melhor é o seu desempenho ?
- A perplexidade mede a confusão do modelo em escolher o próximo token, mas isso depende muito do dataset em que ele foi treinado, por exemplo: um modelo pré-treinado apenas na web, provavelmente apresenta uma perplexidade muito maior se avaliado em textos fora do contexto de pré-treino ?
-
-

Subwords

- Problema: vocabulário grande -> grande matriz de embeddings
- Palavras raras tem embeddings ruins
- Usar caracteres ao invés de palavras?
 - Funciona, mas modelo precisa combinar embeddings de caracteres para "formar" a palavra.
- Compromisso: substituir palavras por *sub-palavras*
- verde = verd e
- verdes = verd es
- verduras = verd uras
- maduras = mad uras

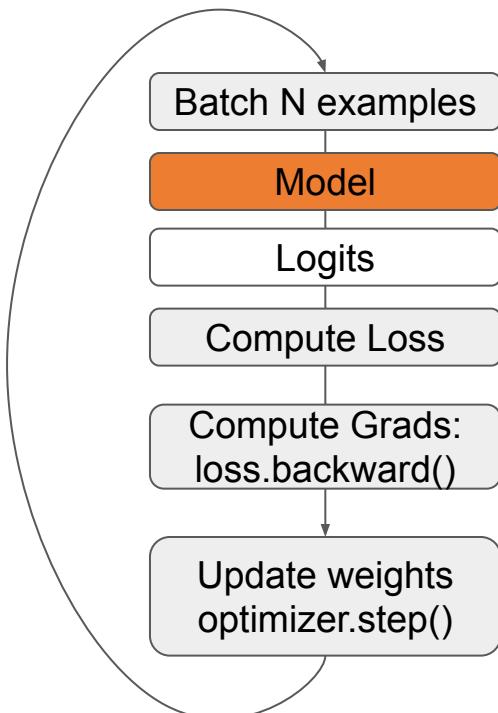
Como funciona:

[Neural Machine Translation of Rare Words with Subword Units](#)

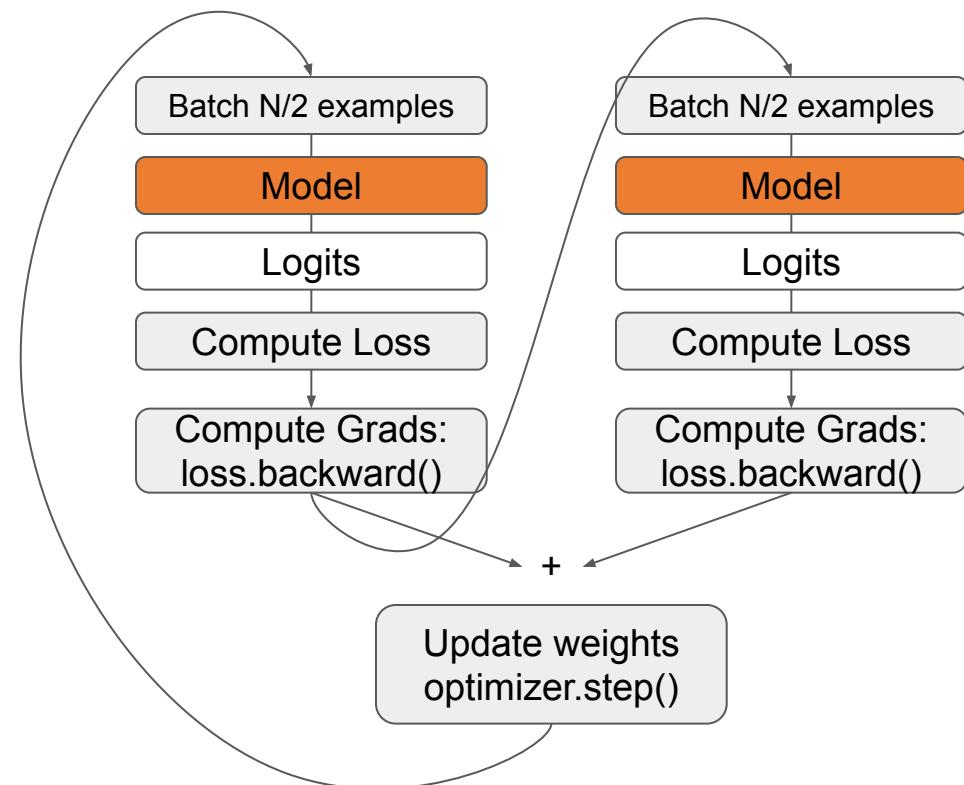
[Exemplo de uso no Colab](#)

[Tutorial Tokenizers - HuggingFace](#)

No Gradient Accumulation



Gradient Accumulation of 2



No Gradient Accumulation

```
for batch in dataloader:  
    x, y = batch  
    logits = model(x)  
    loss = criterion(logits, y)  
    loss.backward() # compute gradients  
  
    optimizer.step() # update weights  
    optimizer.zero_grad() # reset gradients
```

Gradient Accumulation of 2

```
accum_steps = 2  
  
for step_num, batch in enumerate(dataloader):  
    x, y = batch  
    logits = model(x)  
    loss = criterion(logits, y)  
    loss.backward() # compute gradients  
  
    if step_num % accum_steps == 0:  
        optimizer.step() # update weights  
        optimizer.zero_grad() # reset gradients
```

Exercício desta semana: T5 + doc2query

[Solução GPT-4](#)
(com erros!)

Treinar um modelo seq2seq (a partir do [T5-base](#)) na tarefa de expansão de documentos.

- Usar como treino o dataset "tiny" do MS MARCO na tarefa doc2query

https://storage.googleapis.com/unicamp-dl/ia368dd_2023s1/msmarco/msmarco_triples.train.tiny.tsv

- doc2query: A entrada é a passagem e o target é a query
- Note que apenas pares (query, passagem *relevante*) são usados como treino.
- O treino é relativamente rápido (<1 hora).
- Validar a cada X steps usando o [sacreBLEU](#)
- A parte lenta deste exercício é a pré-indexação: para cada documento da coleção, temos que gerar uma ou mais queries, que depois são concatenadas ao documento original, e esse documento "expandido" é indexado.
- Avaliar no TREC-COVID (171K docs), pois é menor que o MS MARCO/TREC-DL 2020 (8.8M passagens).
 - Índice invertido do Trec-covid no pyserini: beir-v1.0.0-trec-covid-flat
 - Corpus e queries na HF: <https://huggingface.co/datasets/BeIR/trec-covid>
 - qrels: <https://huggingface.co/datasets/BeIR/trec-covid-qrels>
 - Usar nDCG@10
 - Comparar com o BM25 com e sem os documentos expandidos pelo doc2query

Perguntas para motivar a entender os conceitos das diferentes arquiteturas do Transformer

- Quais são as diferenças entre encoder-only (BERT), decoder-only (GPT), encoder-decoder (T5)?
- Porque usar o T5 na tarefa do doc2query leva a melhores resultados que usar um decoder-only (ex: GPT-2) de mesmo tamanho?

Summary of Transformer Architectures

Encoder-only:

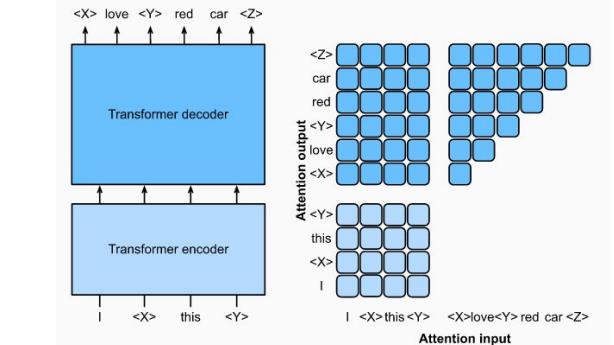
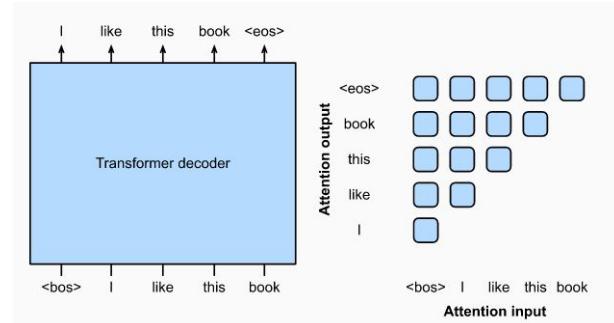
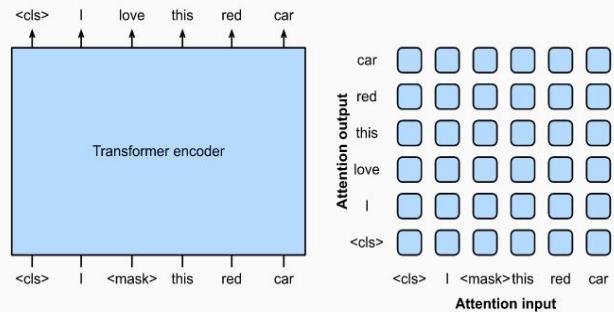
- Bidirectional: left and right tokens can be seen, that is, it has no causality, as we have access to the complete input sequence at inference time
- Good at classifying sequences or tokens (ex: NER). Needs finetuning.
- Ex: BERT, RoBERTa, XLM, DeBERTa

Decoder-only:

- Causal, i.e., right (future) tokens are "removed" from the attention calculation.
- Good text generator.
- With +3B params, it works as a few-shot learner.
- With finetuning, it's not as good as an encoder-only or encoder-decoder.
- Ex: GPT-2, GPT-3, BLOOM, OPT

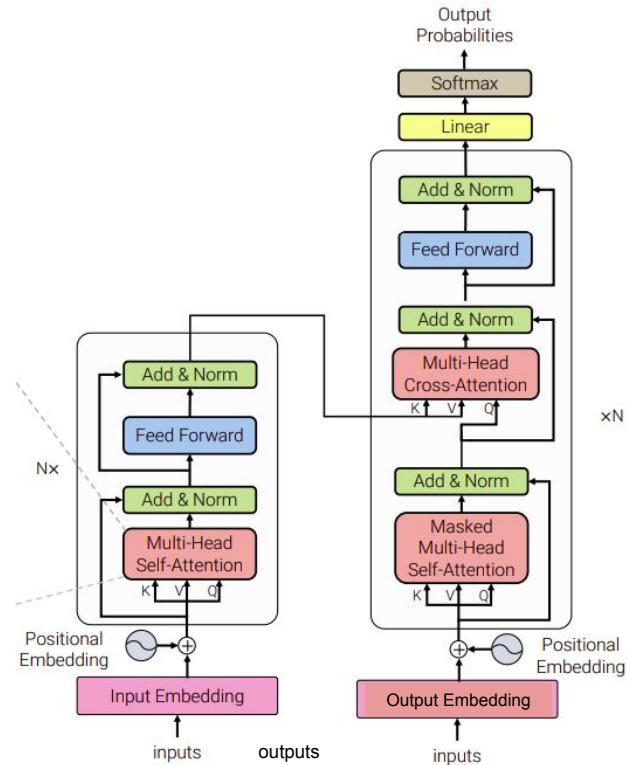
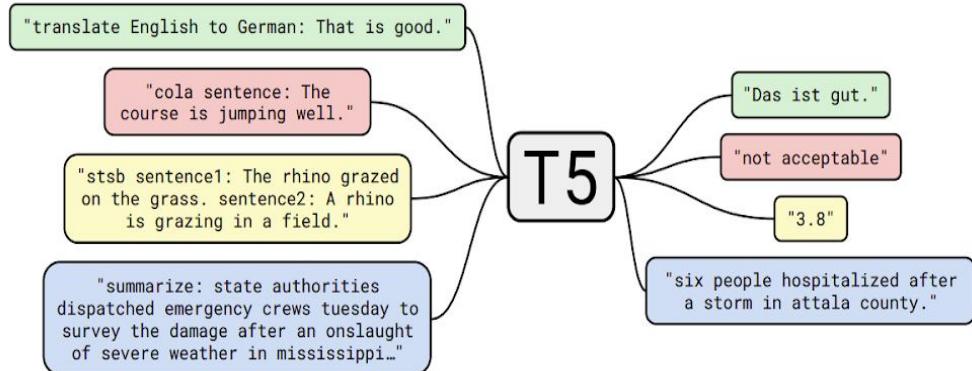
Encoder-Decoder:

- Encoder + Decoder with Causal Attention + Cross-attention (to communicate with the encoder)
- Separate input vs output.
- Good at everything as long as you have a supervised dataset to finetune it.
- BART, T5, FLAN-T5, FLAN-UL2



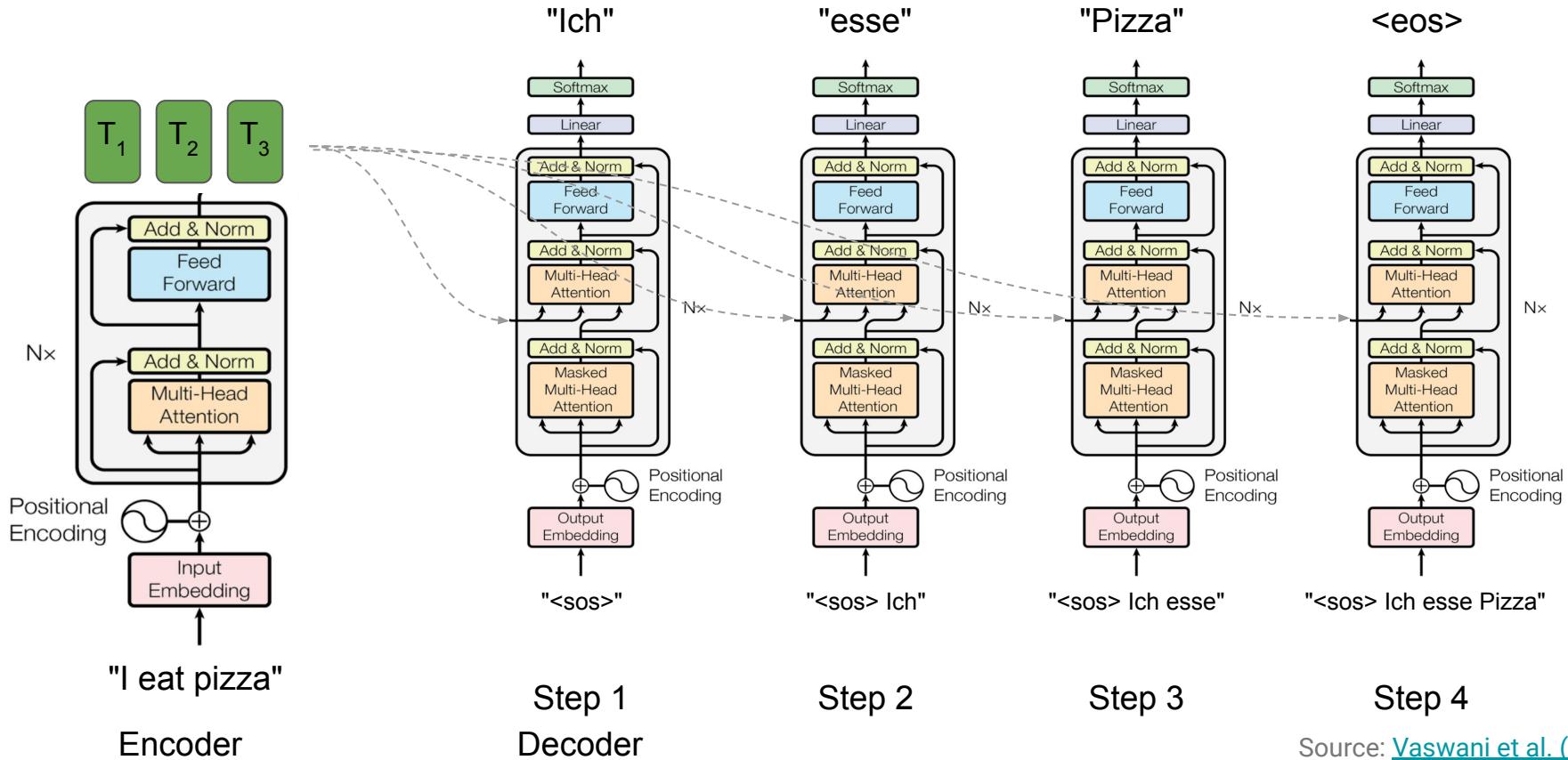
[Source](#)

T5: "Full" Transformer (encoder + decoder)



How does T5 generate text?

A English->German translation example



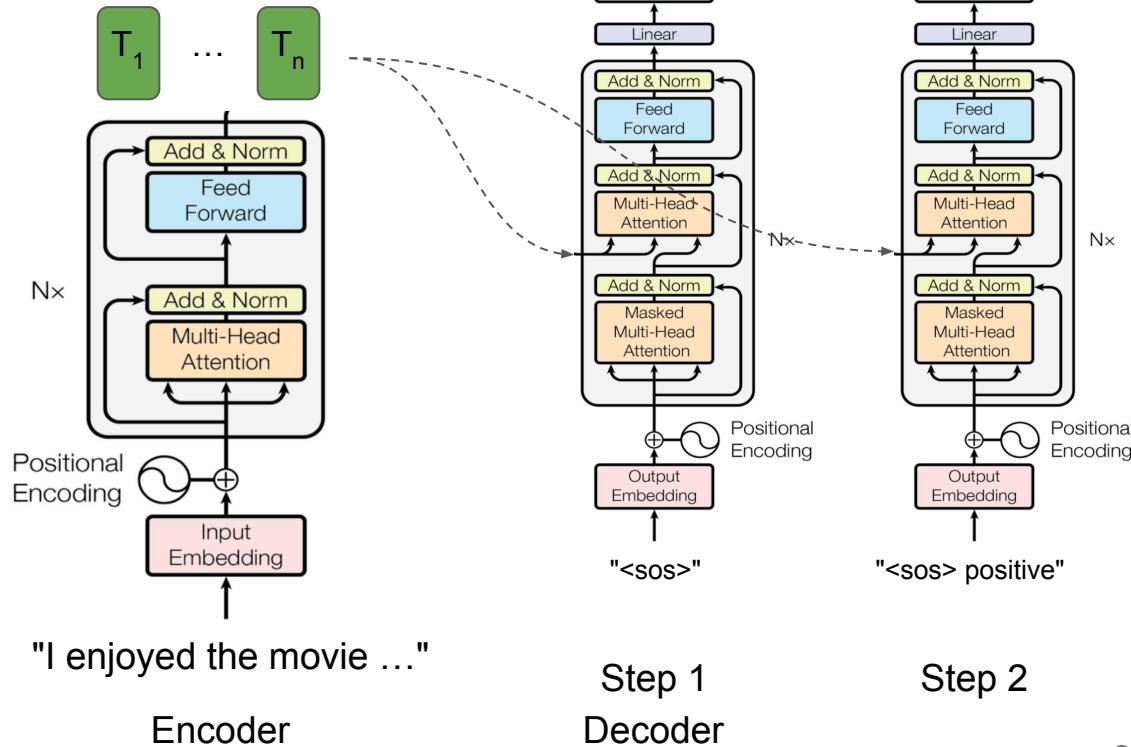
Source: [Vaswani et al. \(2017\)](#)

T5 as a text classifier

A Sentiment Analysis Example

Input: A movie review

Output: positive/negative



Why add the decoder?

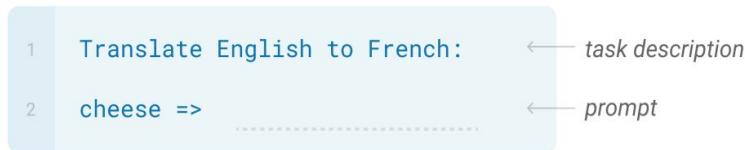
Output: text instead of labels

All weights are pretrained

Recap: Zero-shot and few-shot learning

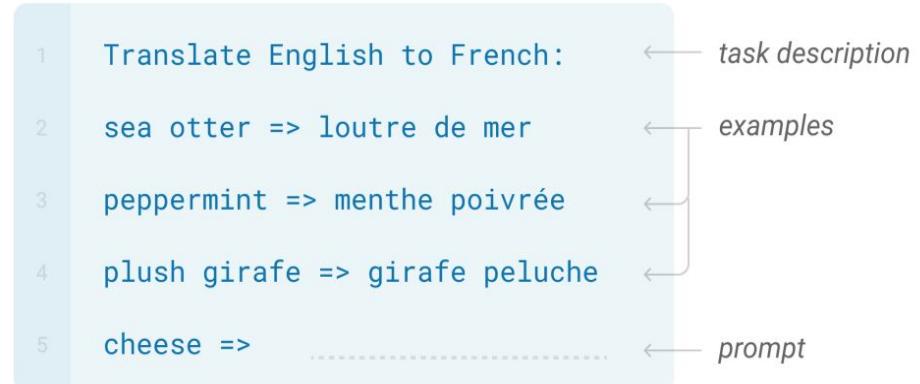
Zero-shot

The model predicts the answer given only a natural language description of the task. No gradient updates are performed.



Few-shot

In addition to the task description, the model sees a few examples of the task. No gradient updates are performed.



Training-Inference Paradigms

Self-supervised (aka unsupervised) Pretraining
(e.g., Web texts)



Supervised Finetuning
(e.g., MS MARCO)



Inference

In-domain
Test Input from the same dataset
(e.g. MS MARCO)



Out-of-domain (aka zero-shot)
Test Input from another dataset
(e.g. BEIR)

Test Output

Zero-shot:
Instruction +
Test Input



Few-shot:
Instruction +
N Input-Output pairs +
Test Input



Test Output

Zero-shot vs Few-shot vs Finetuning

	In-context Learning (Zero-shot or Few-shot)	Finetuning
Cost	I want a solution now, reduce costs later!	Expensive to develop, less expensive in production (smaller models)
Inference Speed	Slow (need at least 3B params)	High (ex: search)
Training data	I can create a small <i>dev</i> set in one day.	I have thousands of labeled examples
Instructions vs Examples	Easier to demonstrate a task with a instruction and few examples	Hard to explain with instructions (e.g., search " <i>is doc relevant to query?</i> ")
Diversity of inputs/outputs	Fast to adapt to new use cases; Few-shot offers more control on the outputs than zero-shot	Use cases will not change over time and inputs/outputs aren't too diverse

Computational Tradeoffs

No distillation (Splade v2, ColBERT v2)
No IR-specific pretraining (e.g. CoCondenser, Contriever)

Method	Indexing Time	Index Size	Retrieval Latency	nDCG@10	
				In-domain TREC-DL 20	Out-of-domain BEIR
BM25	Minutes	<10GB	<100ms	0.475	0.440
doc2query (sparse)	Days	<10GB	<100ms	0.619	0.453
SPLADE (sparse)	Hours	~10GB	~400ms	0.671	0.458
ANCE (dense)	Hours	~20GB	<100ms	0.646	0.413
ColBERT (dense multi vector)	Hours	~100GB	~400ms	0.676	0.453
COIL (dense multi vector)	Hours	~60GB	<100ms	0.688	0.483
BM25 + monoT5 (reranker)	Minutes	<10GB	>1000ms	0.701	0.490

Assunto da sexta aula

- Discussão do Artigo: doc2query
- Discussão dos Exercícios: Treinamento e avaliação do doc2query
- 10:30-11:30: Exercícios Próxima Aula: busca densa
 - Notebook/Apresentação
 - Leitura do artigo: DPR e ColBERT (opcional)

Dicas para realizar experimentos mais rapidamente, com menos bugs e com menor custo: **MVP do pipeline**

- Usar a GPU mais barata, ou até uma CPU, no desenvolvimento inicial;
- Usar modelo pequeno (ex: T5-tiny) inicialmente;
- Treinar com uma fração do dataset e rodar o pipeline todo para ver se a métrica sai do aleatório;
- Inferência em apenas uma fração do corpus (ex: indexar os qrels apenas) – o nDCG deveria ficar bem mais alto que indexar a coleção toda;
- Tentar reproduzir usando um modelo já treinado "bom" (ex: modelo doc2query disponível na HF).
- "Trapacear" na etapa de inferência. Ex: apenas expandir e indexar os documentos que tem uma query "relevante", e ver se o nDCG aumenta
- Tentar usar os hiperparametros sugeridos pelos autores.
- Sintonizar os hiperparametros apenas depois de ter uma versão inicial funcionando "ok" (possivelmente sem bugs)
- Variar LR em 10x para mais e para menos

Resumo do Exercicio do doc2query

Dificuldade de definir o numero de queries amostradas: + queries nem sempre melhora o resultado end2end de busca (i.e., nDCG@10)

- + queries amostradas melhora, mas talvez apenas no "in-domain"
- Já no zero-shot (aka out-of-domain/TREC-COVID) isso não foi verdade (vide grafico Marcos)
- Fp16 se mostrou instável em alguns casos. BF16 pode ajudar, mas tem algumas arquiteturas que não se dão bem com ele.
- Thiago: geração em batches aumenta bastante a velocidade
- Usar a loss de validação para fazer o early stop, e não o BLEU
- Tunar o k1 e b do BM25 traz grandes ganhos, mas tomar cuidado para avaliar em um dataset de teste a parte e ver se os ganhos se mantêm
- Ideias de projeto:
 - usar few-shot para deixar a expansão de queries mais "natural". Por exemplo usar modelo para predizer várias queries "diversas" que cobrem todo o documento de uma única vez
 - Usar o rerankeador a la "doc2query - -" para filtrar melhor as queries (isso pode ser útil para mitigar o problema da halucinação de modelos em geral)
 - Multilingual doc2query: gerar queries em várias línguas: facilitar busca multilingua e cross-lingual
 - Treinar doc2query em pt a partir do multilingue

Algoritmos de Decodificação

Questions from Last Session + Decoding Algorithms

Questions from last lab assignment

Question 1: When should we use top-p sampling and when should we use beam search?

Answer:

Top-p: **Non-deterministic** outputs; Good for paraphrasing, generating stories.

Beam search or greedy-decoding: **Deterministic** outputs; Good for information extraction, translation, question-answering.

Question 2: How does the temperature parameter of the top-p sampling algorithm influence the generated output?

Answer:

Higher temperature -> more diverse outputs but increased risk of "hallucinations". Lower temperature -> less diverse.

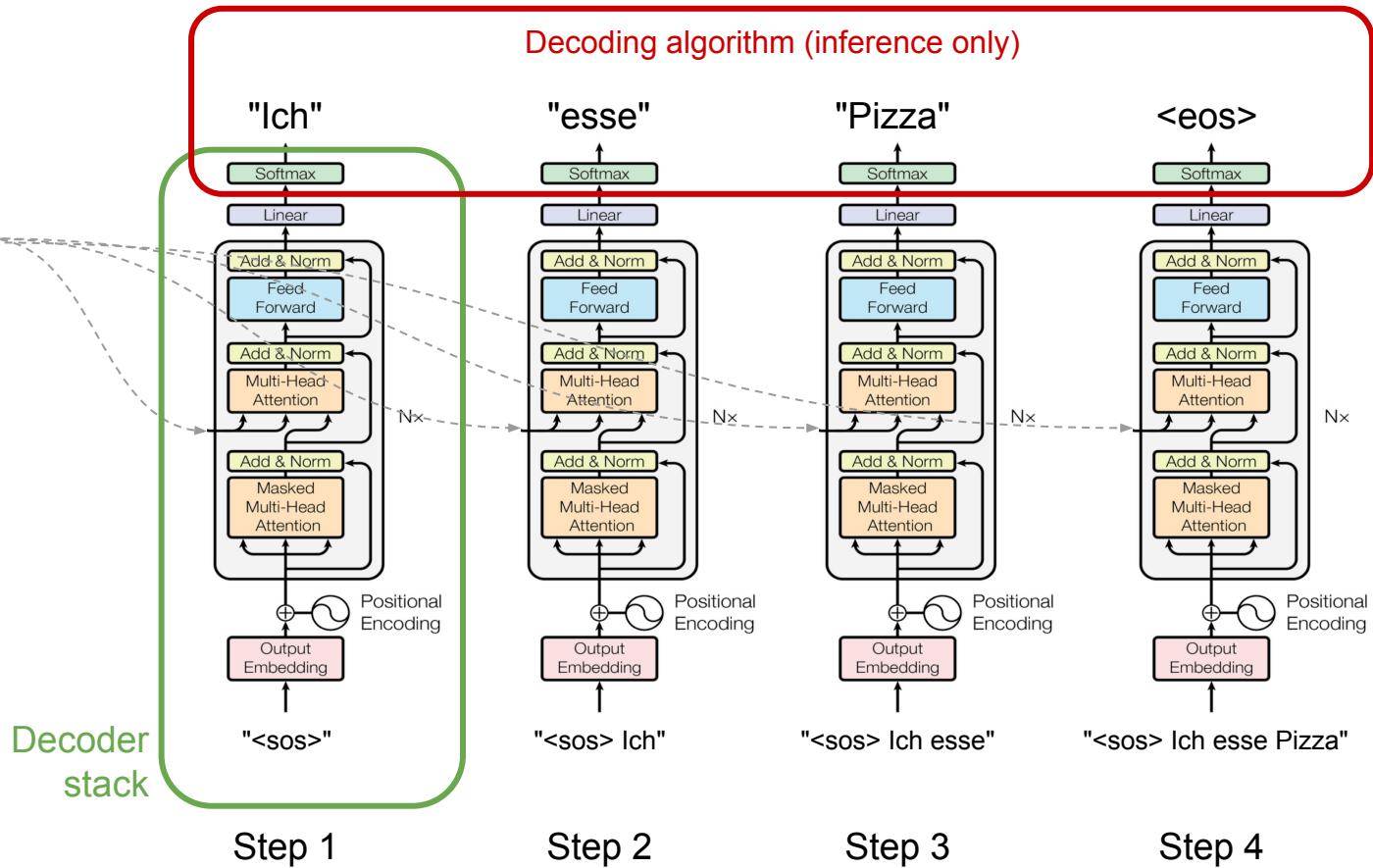
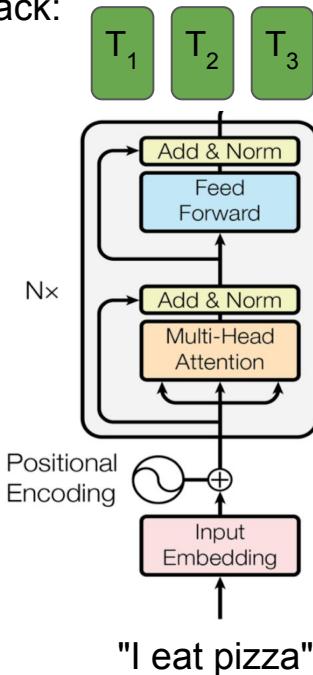
When temp=0, OpenAI (and many others) defaults to greedy decoding.

Decoder Stack vs Decoding algorithm

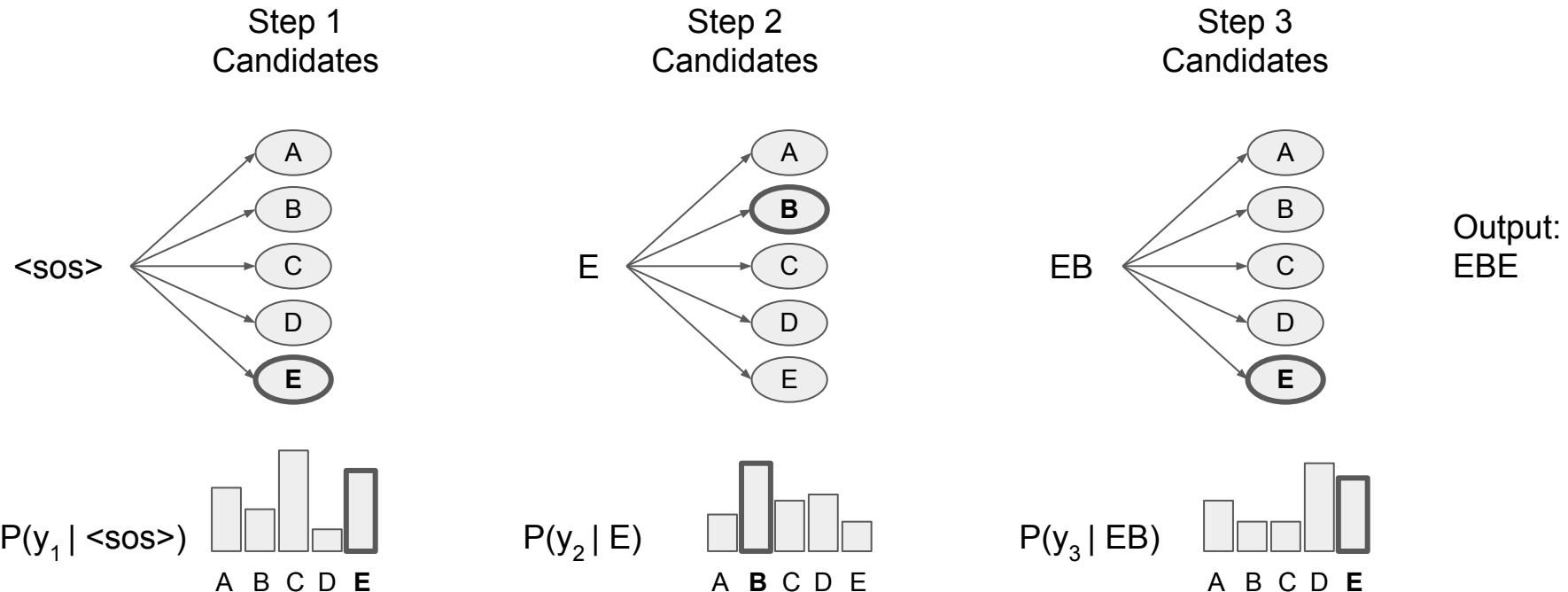
At training time: ground-truth tokens in the input (known as [teacher-forcing](#))

A English->German translation example

Encoder stack:



Sampling Decoder



How temperature affects sampling?

$$P(y^i) = \text{softmax}(z^i, T) = \exp(z^i / T) / \sum_j \exp(z^j / T)$$

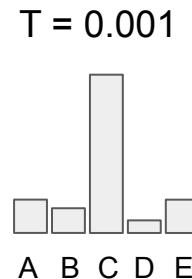
T = temperature (>0)

z^i = logit of the i -th token

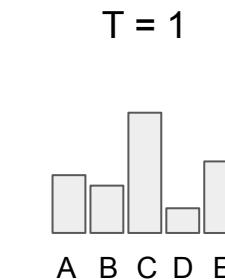
The lower the temperature, "softmax" becomes closer to "max", that is, more similar to greedy decoding.

The higher the temperature, the more "flattened" the distribution becomes, that is, the more random the predictions become.

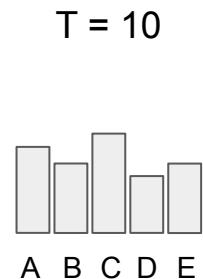
Input: Elle aime voyager
Target: She likes to travel



Prediction: She likes to travel



She likes Paris



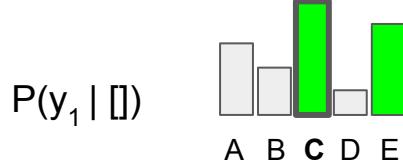
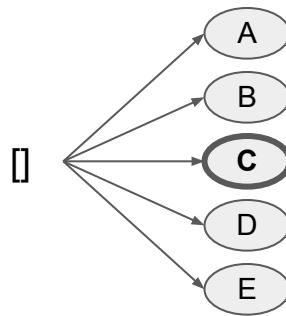
car fool ##ing

Top-k Sampling Decoding

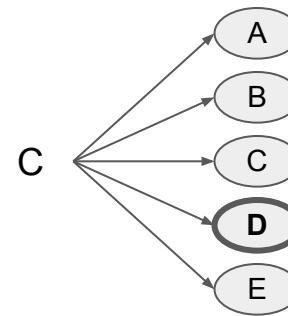
Sample a token among the top k ones with the highest probabilities

Example:  top-k probabilities, k=2

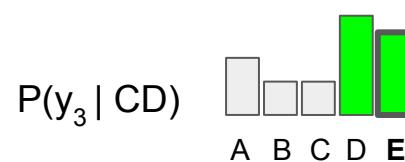
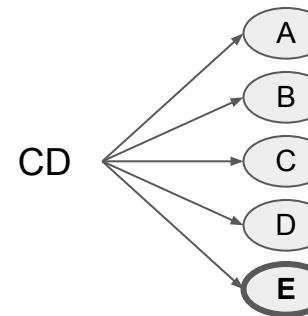
Step 1
Candidates



Step 2
Candidates



Step 3
Candidates



Saída:
CDE

Nucleus Sampling Decoding (aka top-p sampling)

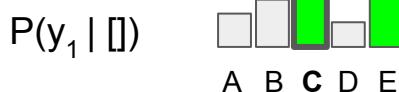
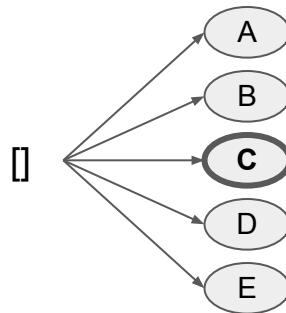
Sample a token from the largest ones that account
to p (%) of the probability distribution

Vocabulary $V^{(p)}$ is the smallest set such that:

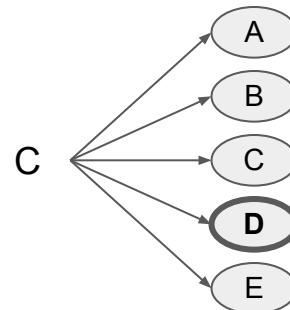
$$\sum_{x \in V^{(p)}} P(x|x_{1:i-1}) \geq p$$

Example:  top-p probabilities, p=80%

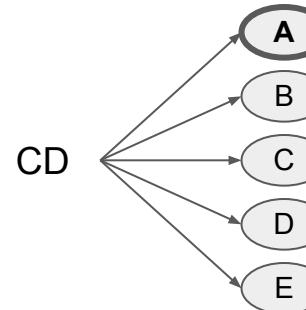
Step 1
Candidates



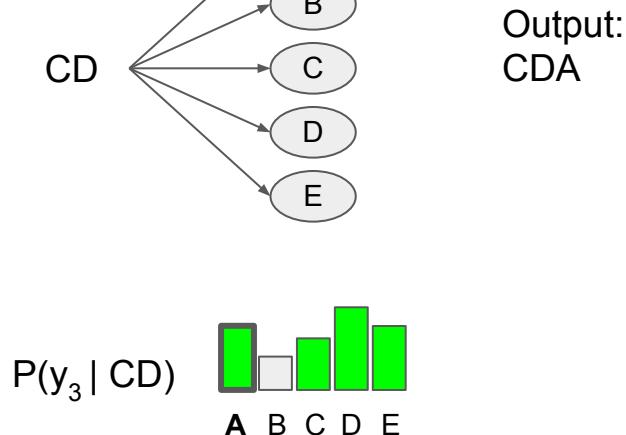
Step 2
Candidates



Step 3
Candidates



CD

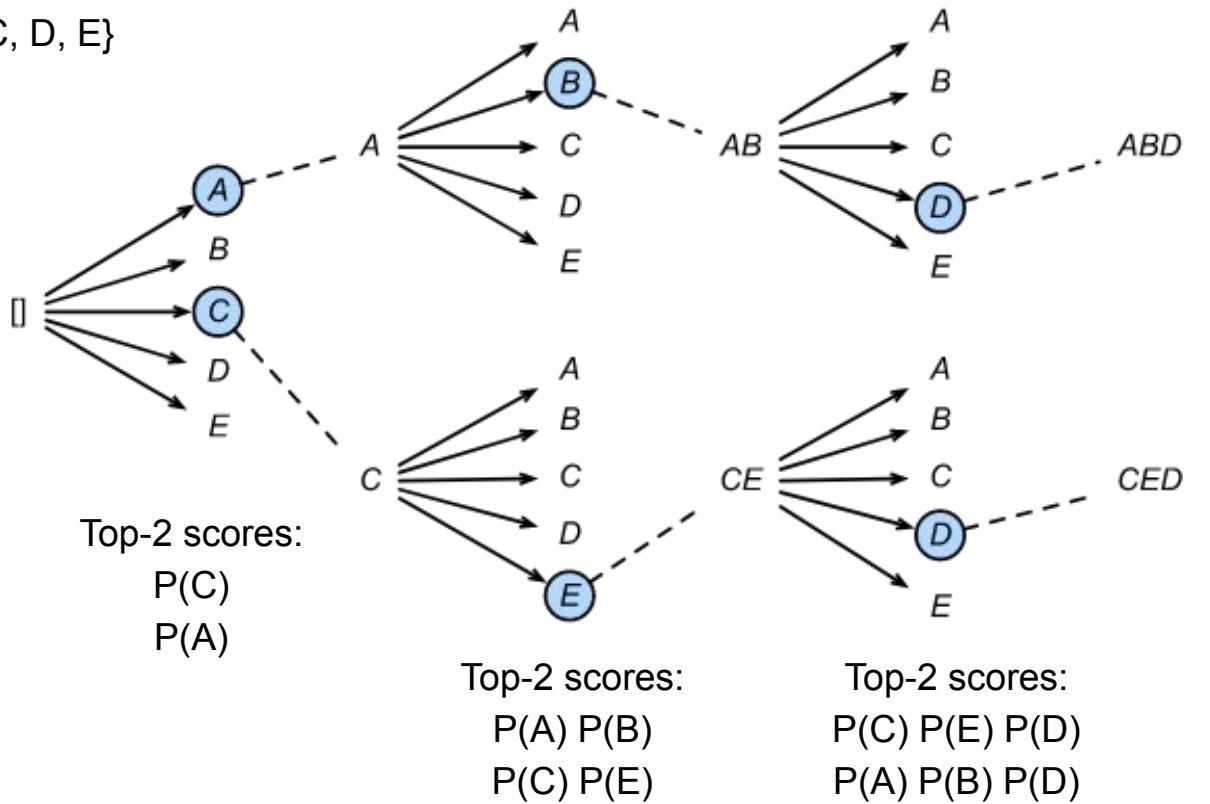


Beam Search

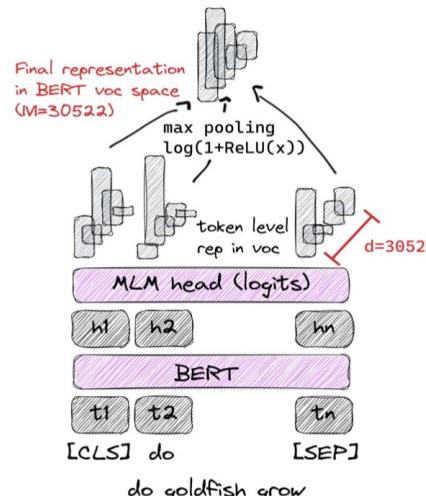
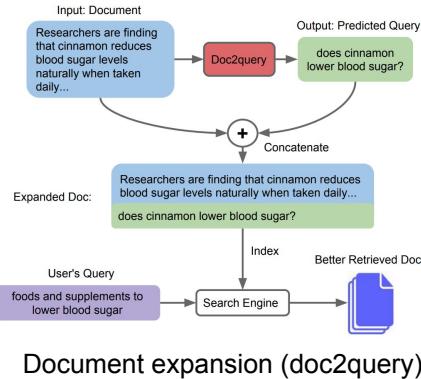
Vocabulary = {A, B, C, D, E}

Beam size = 2

Max time steps = 3

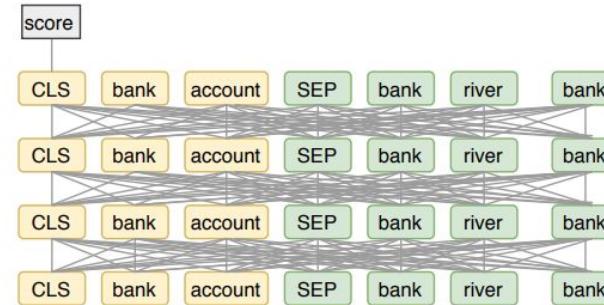


[Source](#)

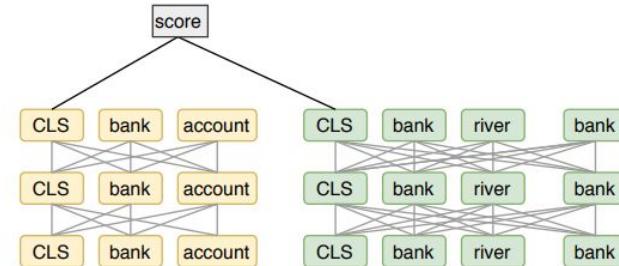


Learned Sparse (e.g., SPLADE)

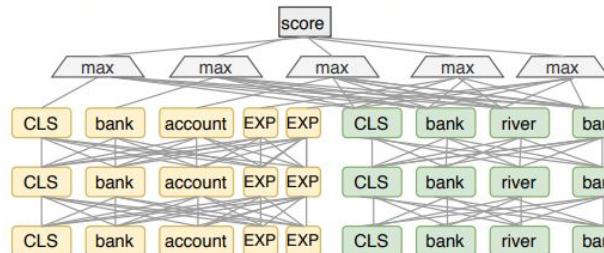
Some Transformer-based retrievers



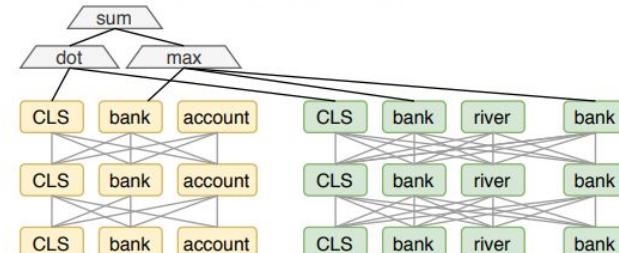
(a) Cross-Attention Model (e.g., BERT reranker)



(b) Dense Retrievers (e.g., DPR)



(c) ColBERT: All-to-All Match

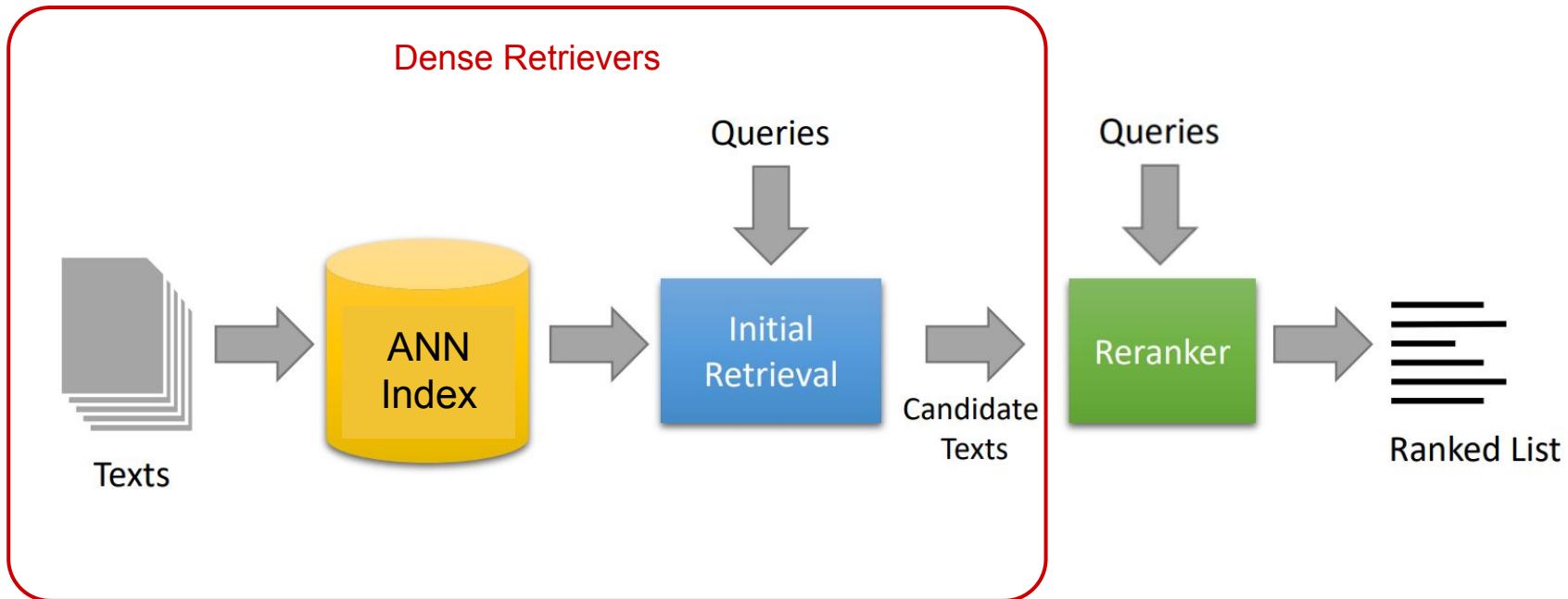


(d) COIL: Contextualized Exact Match

Sources: Gao et al., "COIL: Revisit Exact Lexical Match in Information Retrieval with Contextualized Inverted List", 2021
 Formal et al., "SPLADE: Sparse lexical and expansion model for first stage ranking", 2021
 Nogueira et al., "Document expansion by query prediction", 2019

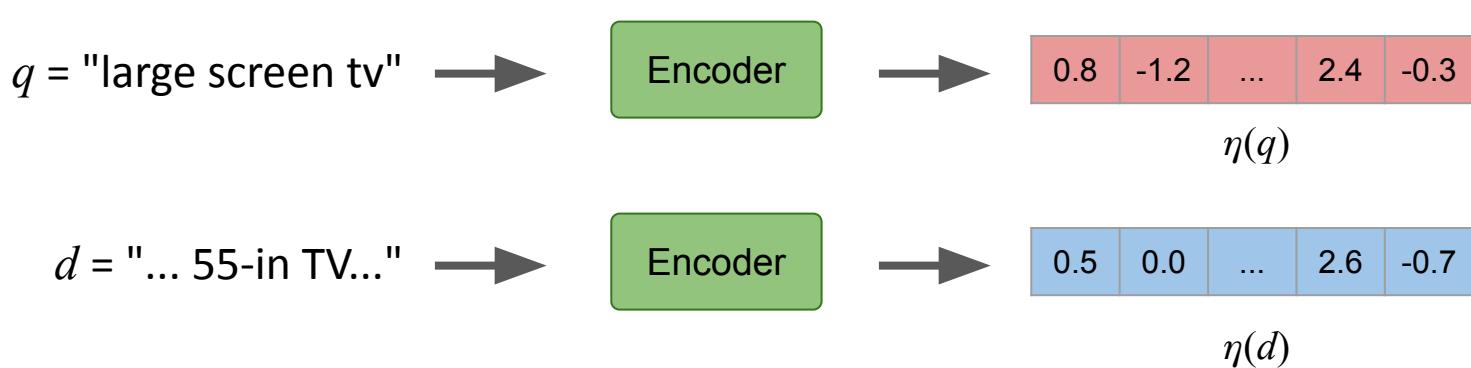
Dense Retrieval

A Simple Search Engine



Dense Representations

Continuous dense vectors \mathbb{R}^D

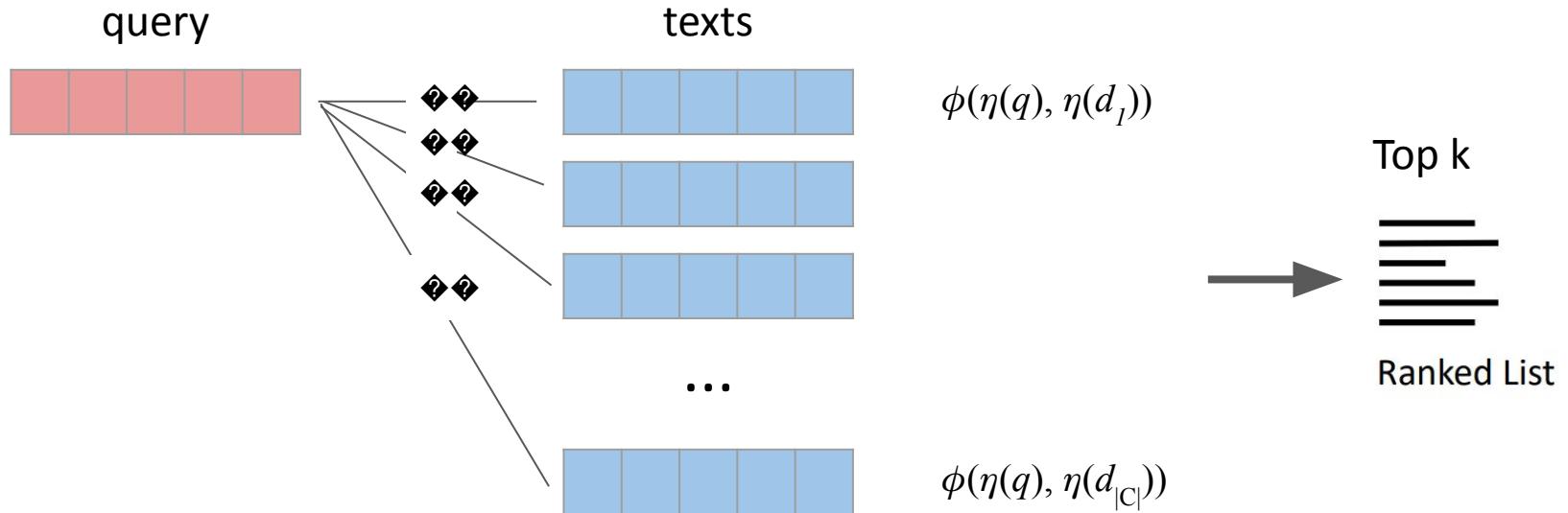


ϕ is a similarity function (e.g., inner product or cosine similarity)

$\phi(\eta(q), \eta(d)) \rightarrow$ ideally measures how relevant q and d are to each other

Task: find the top k most relevant texts to a query

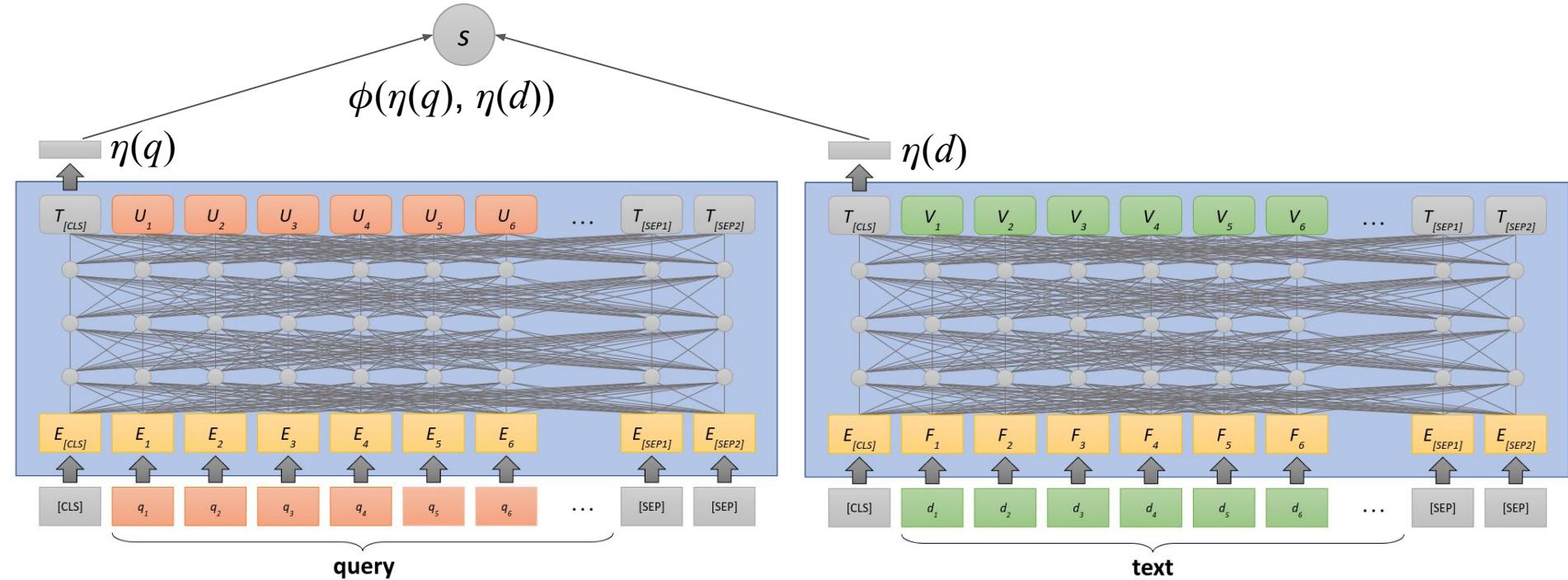
Brute-force search:



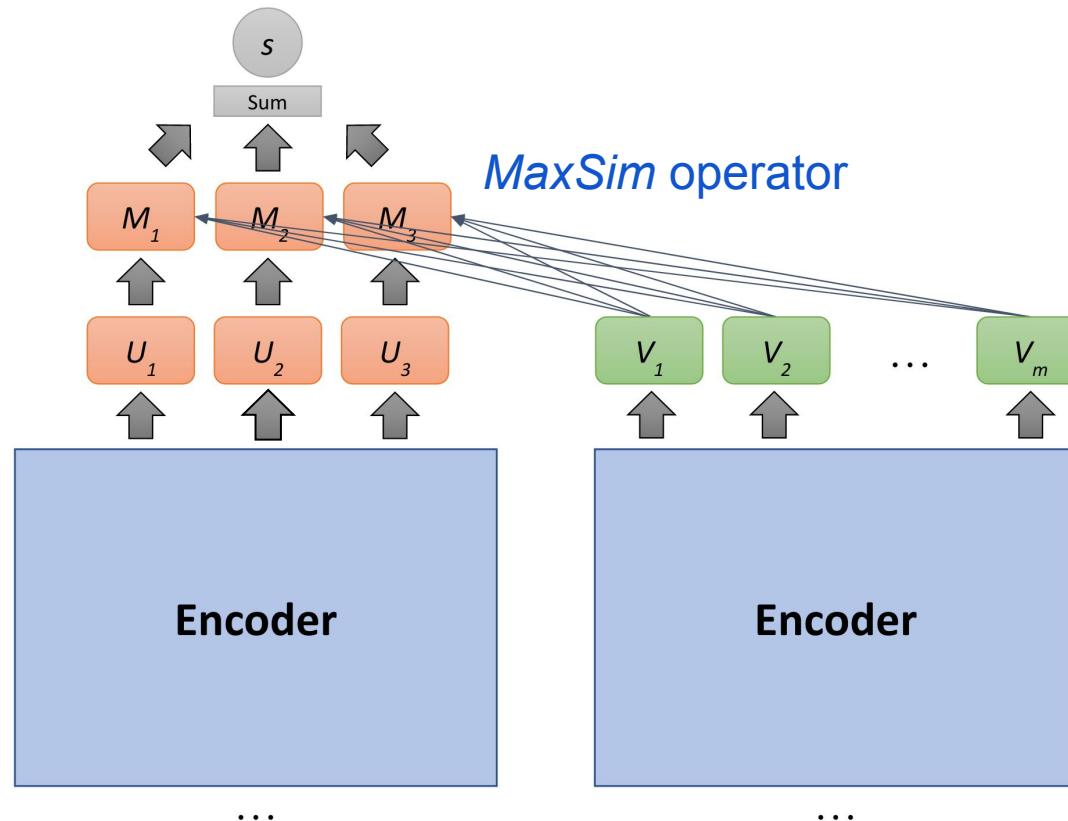
We often need to search many (e.g.: millions) of texts

- Brute-force won't scale

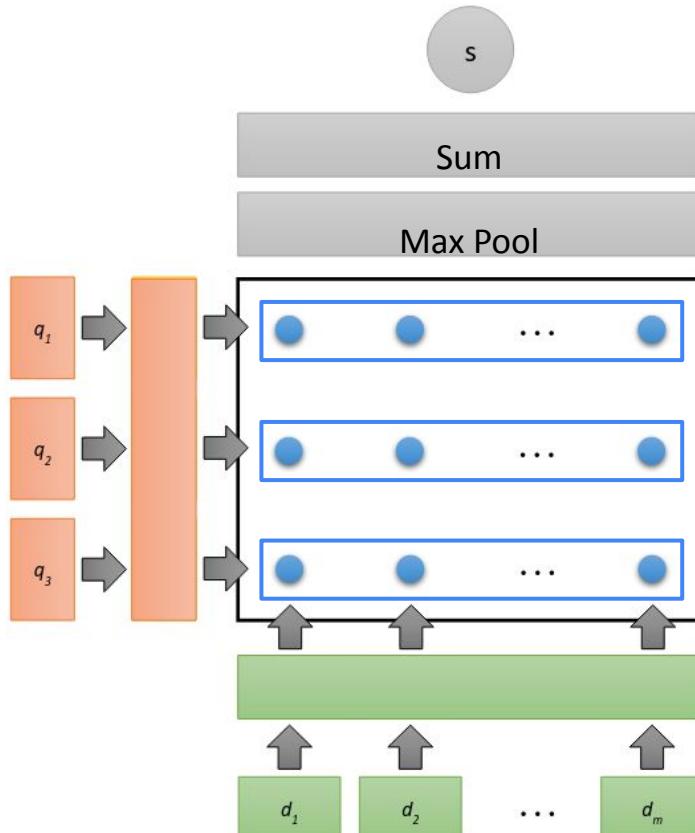
Types of Encoders: Bi-encoder



Comparison-based: ColBERT



Comparison-based: ColBERT



MaxSim:
Sim-mat max pooling
(along query dimension)

$$s_{q,d} = \sum_{i \in \eta(q)} \max_{j \in \eta(d)} \eta(q)_i \cdot \eta(d)_j^T$$

Approximate nearest neighbor (AAN) search

Given a high-dimensional vector space, we can retrieve the most similar neural vectors using k-NN. A brute force approach however is very slow (~10 queries/s)

With Approximate NN, this can be accelerated up to 10^5 times on GPU (hnsw, quantization, lsh)

- **FAISS - Billion-scale similarity search with GPUs**
- **Nmslib - Non-Metric Space Library**
- **SCANN - Scalable Nearest Neighbors**

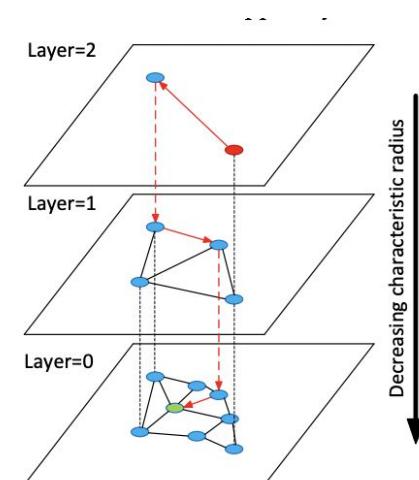
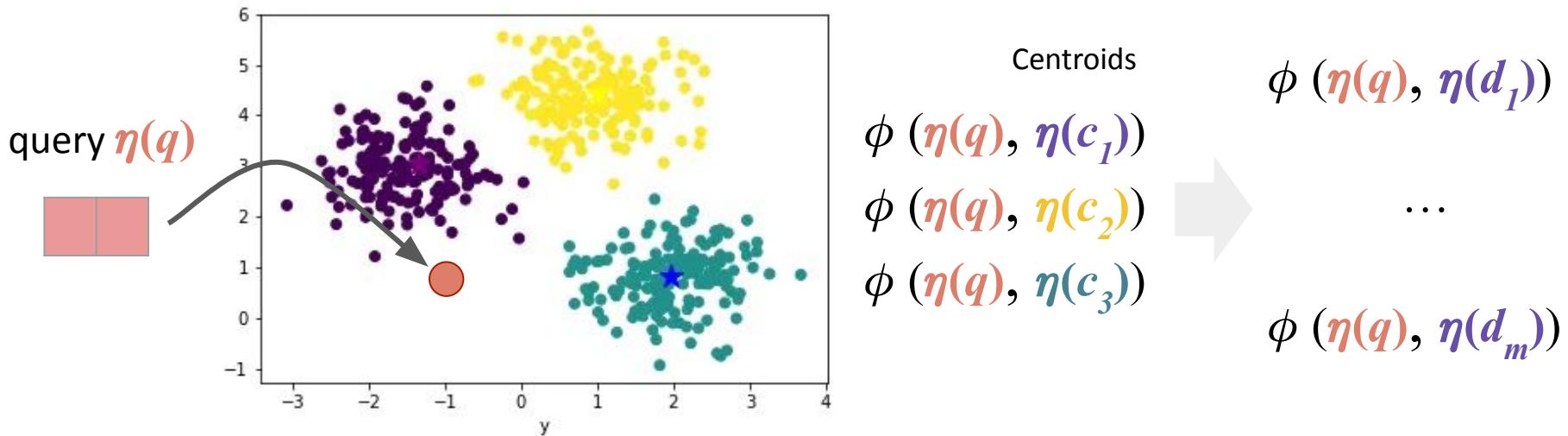


Fig. 1. Illustration of the Hierarchical NSW idea. The search starts from an element from the top layer (shown red). Red arrows show direction of the greedy algorithm from the entry point to the query (shown green).

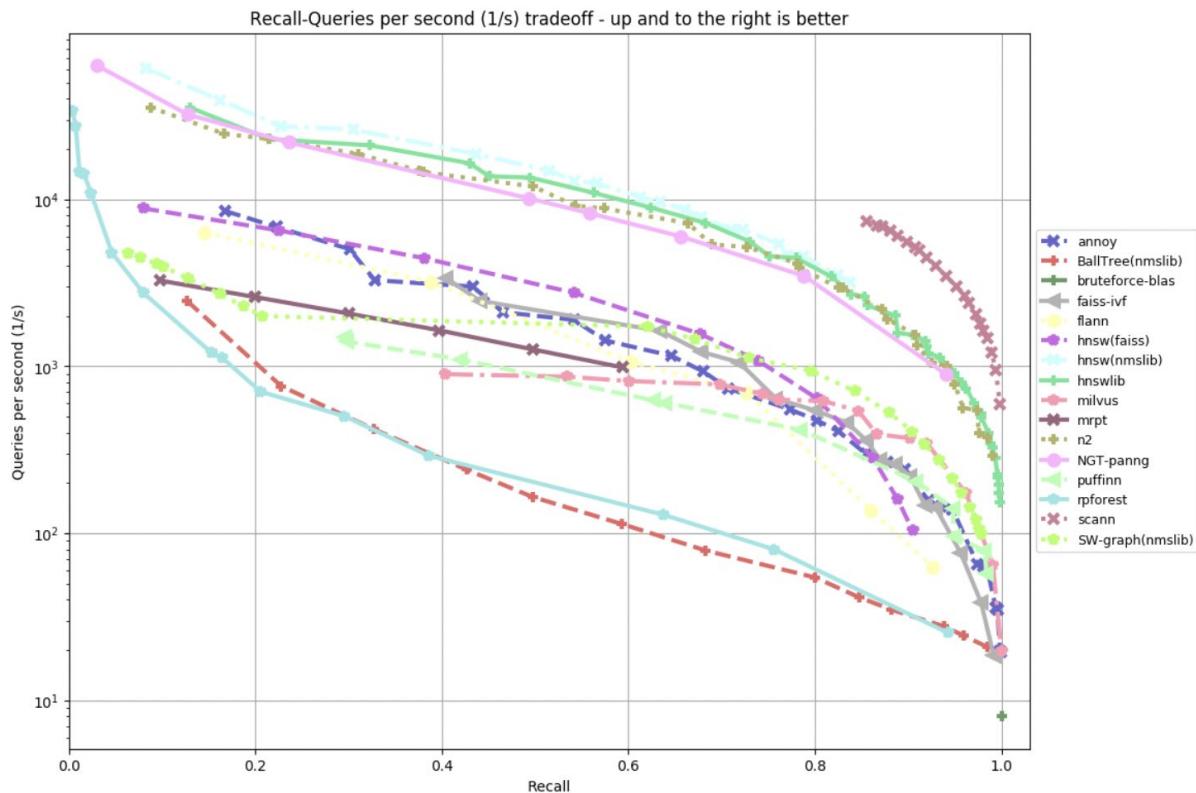
A simple ANN Algorithm using k-means



- In practice, ANN implementations are more complicated
- We assume a fast dense retrieval library is available (e.g.: Faiss, Annoy, ScaNN)

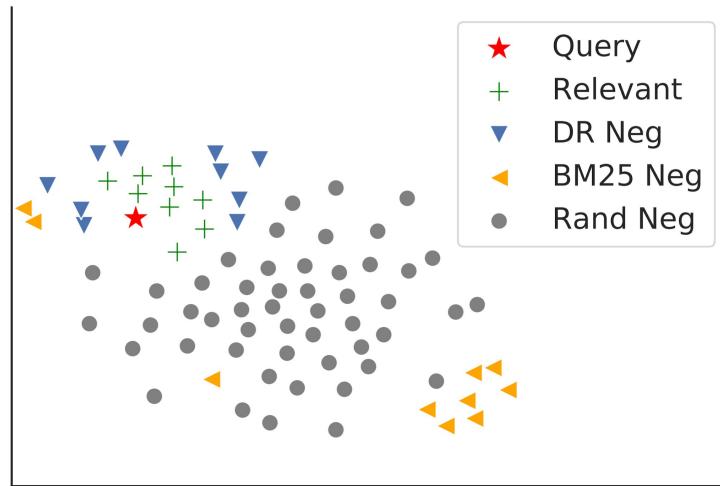
Approximate nearest neighbor search

glove-100-angular

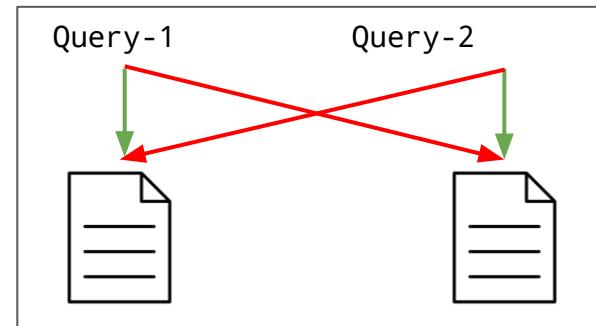


Techniques to train a better bi-encoder

Selecting negative examples: random, BM25, or **in batch**



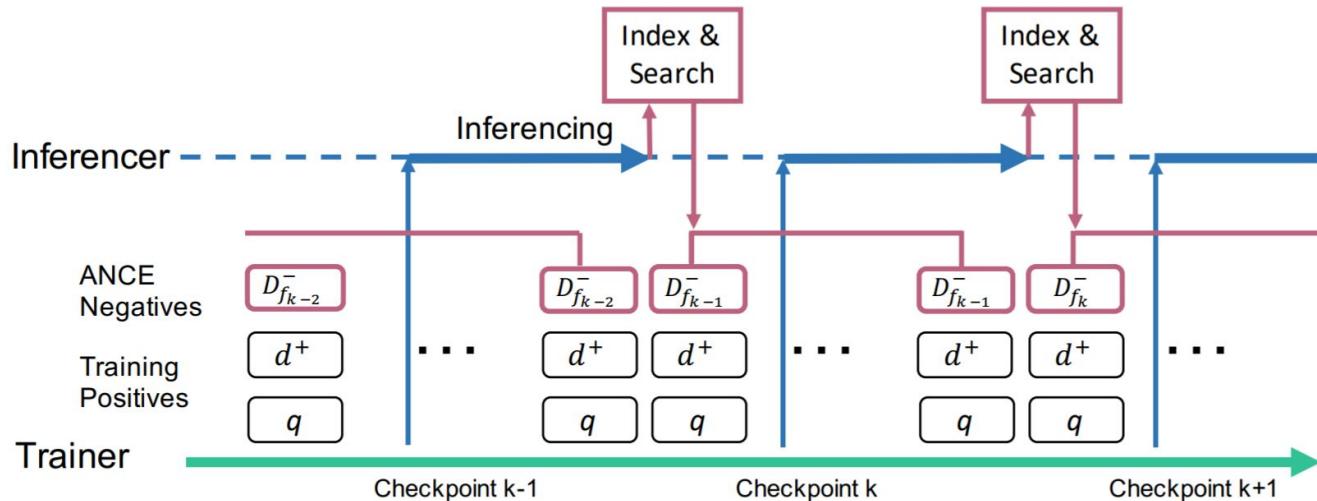
Batch



Techniques to train a better bi-encoder

ANCE:

- Reuse retriever to select negatives
- Checkpoint regularly



Exercício desta semana: Buscador Denso

Iremos fazer finetuning de um buscador denso

- Usar como treino o dataset "tiny" do MS MARCO
https://storage.googleapis.com/unicamp-dl/ia368dd_2023s1/msmarco/msmarco_triples.train.tiny.tsv
- Avaliar o modelo no TREC-COVID, e comparar os resultados com o BM25 e doc2query
- Comparar busca "exaustiva" (semelhança do vetor query com *todos* os vetores do corpus) com a busca aproximada (Approximate Nearest Neighbor - ANN)
- Para a busca aproximada, usar os algoritmos existentes na biblioteca sentence-transformers (ex: hnswlib) **OU implemente um você mesmo (Bonus!)**
- Dicas:
 - Usar a média dos vetores da última camada (conhecido como mean pooling) do transformer para representar queries e passagens; Alternativamente, usar apenas o vetor do [CLS] da última cada.
 - Tente inicialmente uma loss facil de implementar, como a entropia-cruzada
 - Começar o treino a partir do microsoft/MiniLM-L12-H384-uncased
 - Avaliar o pipeline usando um modelo já bem treinado: sentence-transformers/all-mpnet-base-v2
 - Comparar resultados usando semelhança de coseno e produto escalar como funções de similaridade
 - Para checar se seu código de avaliação está correto, comparar o seu desempenho com o do modelo já treinado no MS MARCO: <https://huggingface.co/sentence-transformers/all-MiniLM-L12-v2>; O nDCG@10 no TREC-COVID deve ser ~0.47
 - Usar a biblioteca do sentence-transformers para avaliar o modelo

Assunto da sétima aula

- Discussão do Artigo: DPR e ColBERT (opcional)
- Discussão dos Exercícios: Treinamento e avaliação do DPR
- 10:30-11:30: Exercícios Próxima Aula: Busca Esparsa Aprendida
 - Notebook/Apresentação
 - Leitura do artigo

Conclusões do Exercício de Busca Densa

- Pesos compartilhados vs dois codificadores:
 - Dois codificadores parece ser melhor
- Loss triplet vs CrossEntropy (original do artigo):
 - Inconclusivo
- Usar loss do artigo + produto escalar funciona
 - Se usar loss do artigo + semelhança de coseno, dai precisa de um fator de ajuste na exponencial da loss
- Negatives: In Batch vs BM25 vs Denso
 - In batch apenas parece ser suficiente.
 - BM25 não pareceu ajudar
- CLS vs Média
 - Ambos funcionam.
- Busca aproximada vs exaustiva
 - A aproximada tem uma queda notável de nDCG
- Busca aproximada com k-means: Mais clusters fica pior
- Quantidade de tokens query vs docs
 - Ajustar para o comprimento com base no dataset de treino, deixa o treino mais rápido.

Busca Esparsa Aprendida

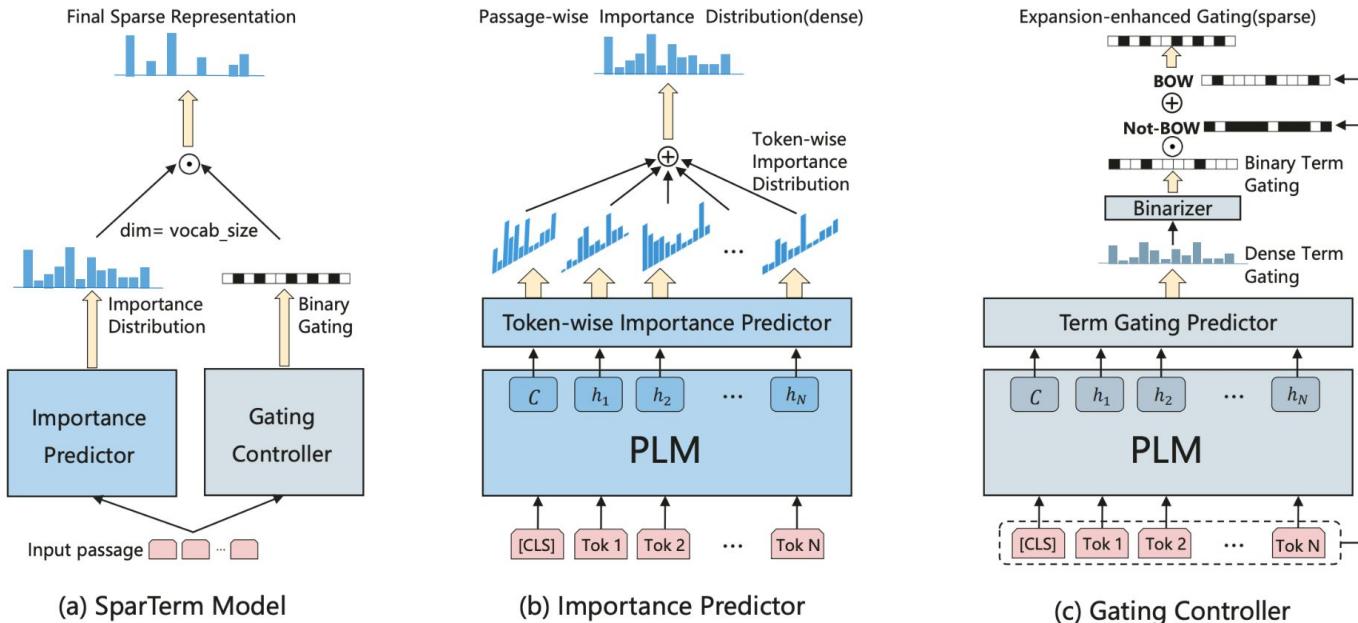


Figure 2: Model Architecture of SparTerm. Our overall architecture contains an importance predictor and a gating controller. The importance predictor generates a dense importance distribution with the dimension of vocabulary size, while the gating controller outputs a sparse and binary gating vector to control term activation for the final representation. These two modules cooperatively ensure the sparsity and flexibility of the final representation.

SPLADE: Learned Sparse Representations

State-of-art retriever on BEIR *without* a reranker

$|\text{embedding}| = \text{vocabulary size}$

$\log(1 + \text{ReLU}(\text{logits}))$ ensures sparsity (i.e., most elements are zeros)

Can be used with existing inverted index infrastructure (e.g., Lucene)

Training loss:

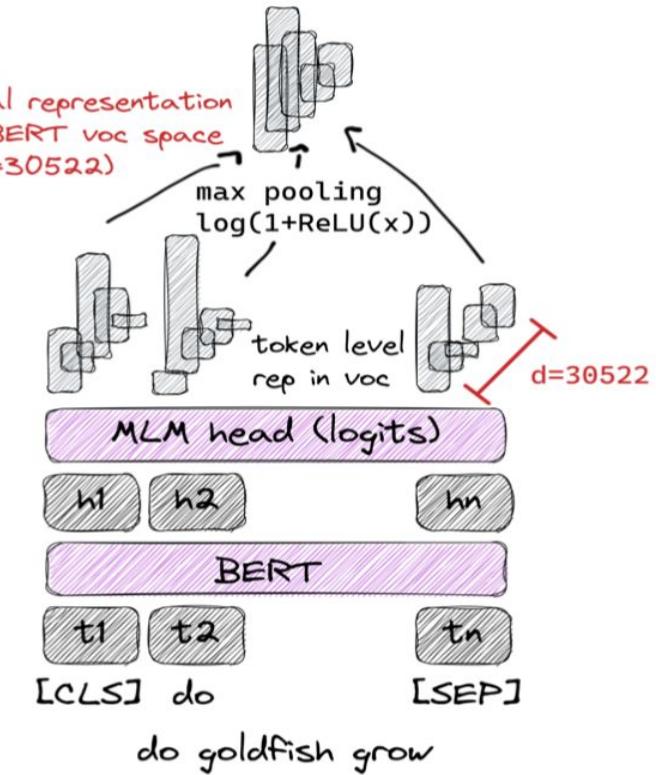
$$\mathcal{L} = \mathcal{L}_{\text{rank-IBN}} + \lambda_q \mathcal{L}_{\text{reg}}^q + \lambda_d \mathcal{L}_{\text{reg}}^d$$

$$\mathcal{L}_{\text{rank-IBN}} = -\log \frac{e^{s(q_i, d_i^+)}}{e^{s(q_i, d_i^+)} + e^{s(q_i, d_i^-)} + \sum_j e^{s(q_i, d_{i,j}^-)}}$$

high score for relevant query-doc pairs

$$\ell_{\text{FLOPS}} = \sum_{j \in V} \bar{a}_j^2 = \sum_{j \in V} \left(\frac{1}{N} \sum_{i=1}^N w_j^{(d_i)} \right)^2$$

ensures low term weights



do goldfish grow

[Formal et al. 2021](#)

Exercício desta semana: Busca Esparsa

Implementar a fase de indexação e buscas de um modelo sparso

- Usar este modelo SPLADE já treinado [naver/splade_v2_distil](#) (do distilbert) ou [splade-cocondenser-selfdistil](#) (do BERT-base 110M params). Mais informações sobre os modelos estão neste artigo: <https://arxiv.org/pdf/2205.04733.pdf>
- Não é necessário treinar o modelo
- Avaliar nDCG@10 no TREC-COVID e comparar resultados com o BM25 e buscador denso da semana passada
- A dificuldade do exercício está em implementar a função de busca e ranqueamento usada pelo SPLADE. A implementação do índice invertido é apenas um "dicionário python".
- Comparar seus resultados com a busca "original" do SPLADE.
- Medir latencia (s/query)

Exercicio desta semana: Busca Esparsa

```
query = where eat pizza
|
BERT
|
vetor_where = [0, ..., 29999] # (30000 escalares)
vetor_eat = [0, ..., 29999]
vetor_pizza = [0, ..., 29999]

vetor = log(1 + relu(vetor_where)) +
       log(1 + relu(vetor_where)) +
       log(1 + relu(vetor_where))

vetor.shape = (30000,) # Espenso

Apenas 6 tokens sao diferentes de 0
query = [0.5 - place,
          0.8 - restaur
          0.7 - ##ant
          1.5 - pizza
          0.3 - pizzas
          0.2 - parlor]

indice = {
    "place", (doc#23059, 0.5),
    "restaur", (doc#1159, 0.8),
    ...
}
```

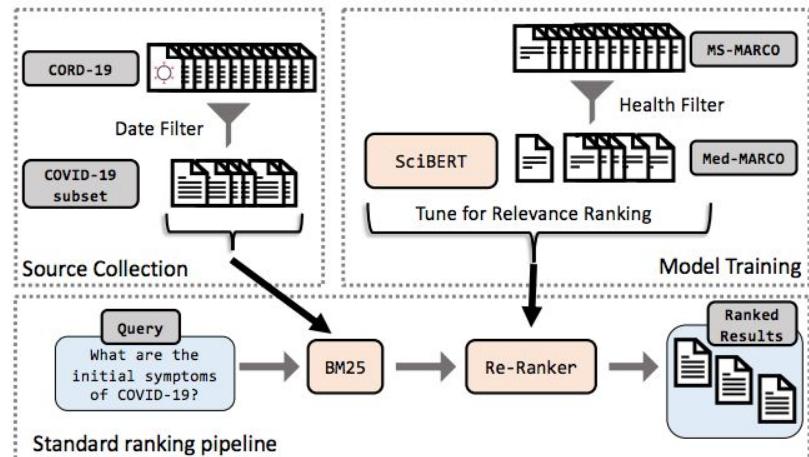
Assunto da oitava aula

- Discussão do Artigo: SPLADE e SPLADE-v2 (opcional)
- Discussão dos Exercícios: Implementação do pipeline de buscas do SPLADE
- 10:30-11:30: Exercícios Próxima Aula: Transfer Learning usando LLMs
 - Notebook/Apresentação: Geração de Datasets para Tarefas de Buscas com InPars
 - Leitura do artigo: [InPars](#)

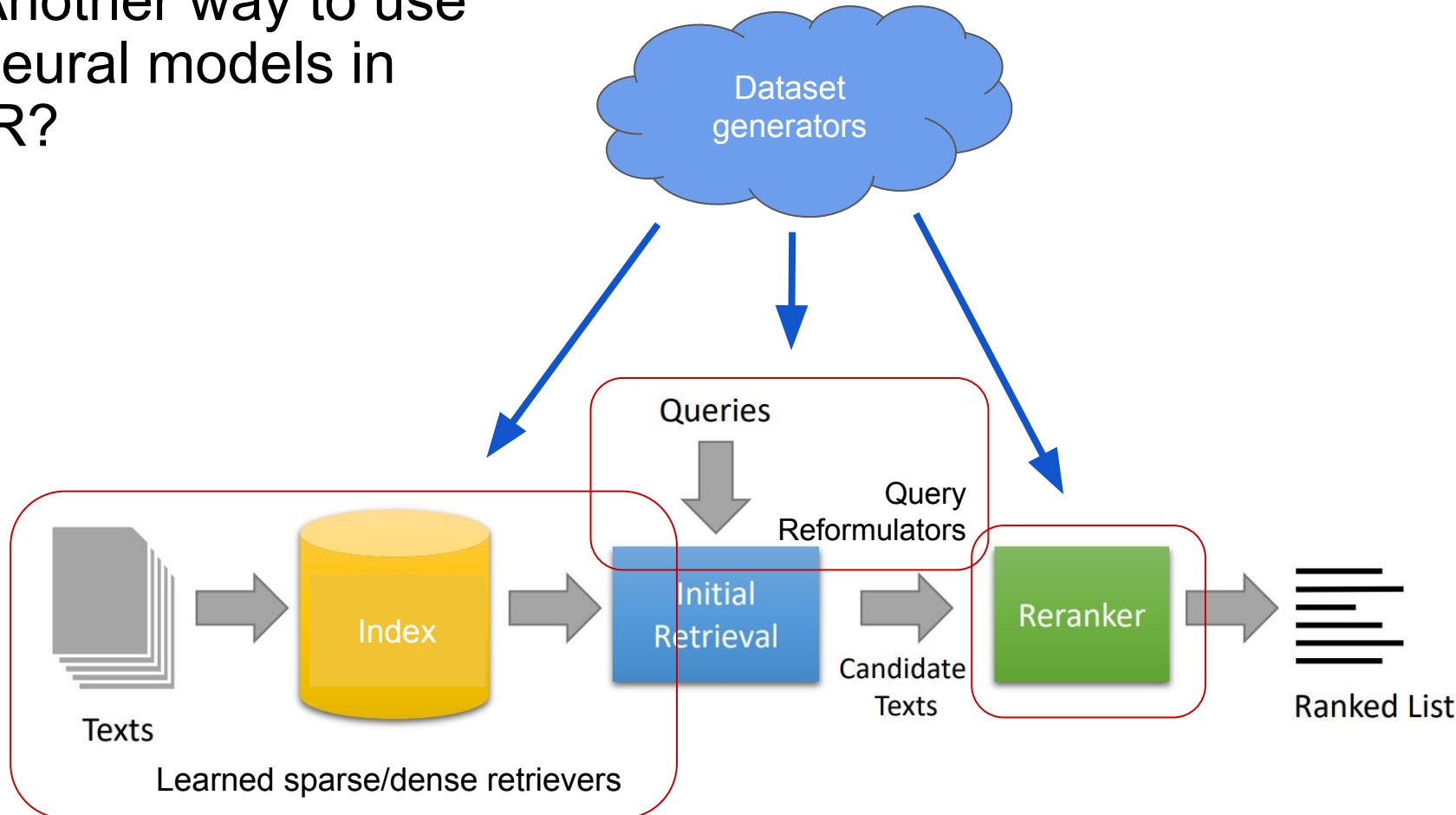
Domain Adaptation

Goal: Extract knowledge from *data or models* available only in one domain (ex: web texts, news) to adapt (ex: finetune, tune hyperparameters) a ML model to perform a task in the *target* domain (ex: legal or biomedical texts)

Example of a simple domain adaptation method: collect large amounts of data from the web and filter with simple rules text that likely belong to target domain.



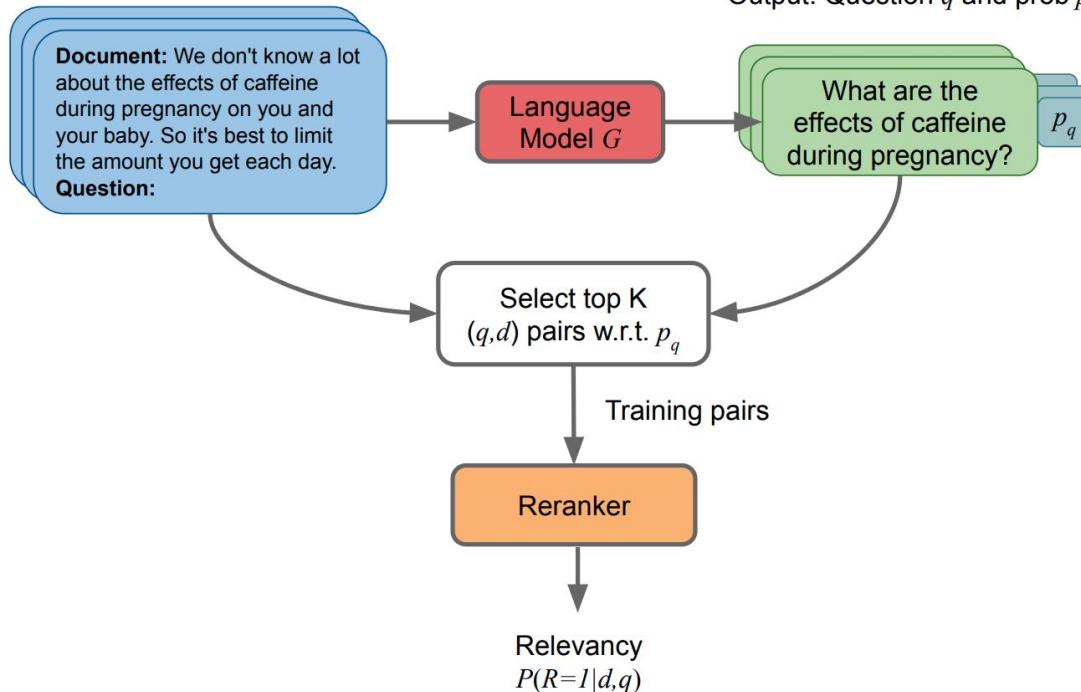
Another way to use neural models in IR?



First Method: *Inquisitive Parrots for Search* (InPars)



Few-shot input: 3 examples + Document d



few-shot prompt

Collection



"Vanilla" prompt

Example 1:

Document: We don't know a lot about the effects of caffeine during pregnancy on you and your baby. So it's best to limit the amount you get each day. If you are pregnant, limit caffeine to 200 milligrams each day. This is about the amount in 1½ 8-ounce cups of coffee or one 12-ounce cup of coffee.

Relevant Query: Is a little caffeine ok during pregnancy?

Example 2:

Document: Passiflora herbertiana. A rare passion fruit native to Australia. Fruits are green-skinned, white fleshed, with an unknown edible rating. Some sources list the fruit as edible, sweet and tasty, while others list the fruits as being bitter and inedible.

Relevant Query: What fruit is native to Australia?

Example 3:

Document: The Canadian Armed Forces. 1 The first large-scale Canadian peacekeeping mission started in Egypt on November 24, 1956. 2 There are approximately 65,000 Regular Force and 25,000 reservist members in the Canadian military. 3 In Canada, August 9 is designated as National Peacekeepers' Day.

Relevant Query: How large is the Canadian military?

Example 4:

Document: {document_text}

Relevant Query:

"Guided by bad questions" prompt

Example 1:

Document: We don't know a lot about the effects of caffeine during pregnancy on you and your baby. So it's best to limit the amount you get each day. If you are pregnant, limit caffeine to 200 milligrams each day. This is about the amount in 1½ 8-ounce cups of coffee or one 12-ounce cup of coffee.

Good Question: How much caffeine is ok for a pregnant woman to have?

Bad Question: Is a little caffeine ok during pregnancy?

Example 2:

Document: Passiflora herbertiana. A rare passion fruit native to Australia. Fruits are green-skinned, white fleshed, with an unknown edible rating. Some sources list the fruit as edible, sweet and tasty, while others list the fruits as being bitter and inedible.

Good Question: What is Passiflora herbertiana (a rare passion fruit) and how does it taste like?

Bad Question: What fruit is native to Australia?

Example 3:

Document: The Canadian Armed Forces. 1 The first large-scale Canadian peacekeeping mission started in Egypt on November 24, 1956. 2 There are approximately 65,000 Regular Force and 25,000 reservist members in the Canadian military. 3 In Canada, August 9 is designated as National Peacekeepers' Day.

Good Question: Information on the Canadian Armed Forces size and history.

Bad Question: How large is the Canadian military?

Example 4:

Document: {document_text}

Good Question:

Instruções Exercício - InPars

Objetivo: gerar dataset para treino de modelos de buscas usando a técnica do InPars e avaliar um modelo rerankeador treinado neste dataset no TREC-COVID:

Entrada: 3-5 exemplos few-shot + documento amostrado da coleção do TREC-COVID

Saída: query que seja relevante para o documento amostrado

É opcional fazer a etapa de filtragem usando as queries de maior prob descrita no Artigo.

- Como modelo gerador, use um dos seguintes modelos:
 - ChatGPT-3.5-turbo: ~1 USD para cada 1k exemplos
 - FLAN-T5 (base, large ou XL), LLAMA-(7,13B), Alpaca-(7/13B), que são possíveis de rodar no Colab Pro.
 - Também tem a inference-api da HF: <https://huggingface.co/inference-api>.
 - Com exceção do LLAMA, é possível usar zero-shot ao invés de few-shot.
- Dado 1k-10k pares <query sintética; documento>, treinar um modelo rerankeador miniLM igual ao da aula 2/3.
- Exemplos negativos (i.e., <query sintética; doc não relevant>) vêm do BM25: dado a query sintética, retornar **top 1000** com o BM25, e amostrar *aleatoriamente* alguns documentos como negativo
- Começar treino do miniLM já treinado no MS MARCO

Avaliar no TREC-COVID e comparar com o rerankeador apenas treinado no MSMARCO

Nota: Também usar o dataset dos colegas para obter diversidade de exemplos: Assim que tiver gerado o dataset sintético, favor colocar na planilha, assim outras pessoas podem usá-lo.

- Para aumentar a aleatoriedade, seed usada deve o seu numero na planilha.

Colocar dataset no formato jsonlines:

```
{"query": query, "positive_doc_id": doc_id, "negative_doc_ids": [opcional]}\n
```

Assunto da nona aula

- Discussão do Artigo: InPars
- Discussão dos Exercícios: Geração de dados sintéticos para busca
- 10:30-11:30: Exercícios Próxima Aula: Eficiência vs Qualidade
 - Notebook/Apresentação:
 - Leitura do artigo: ?

Computational Tradeoffs

No distillation (Splade v2, ColBERT v2)
No IR-specific pretraining (e.g. CoCondenser, Contriever)

Method	Indexing Time	Index Size	Retrieval Latency	nDCG@10	
				In-domain TREC-DL 20	Out-of-domain BEIR
BM25	Minutes	<10GB	<100ms	0.475	0.440
doc2query (sparse)	Days	<10GB	<100ms	0.619	0.453
SPLADE (sparse)	Hours	~10GB	~400ms	0.671	0.458
ANCE (dense)	Hours	~20GB	<100ms	0.646	0.413
ColBERT (dense multi vector)	Hours	~100GB	~400ms	0.676	0.453
COIL (dense multi vector)	Hours	~60GB	<100ms	0.688	0.483
BM25 + monoT5 (reranker)	Minutes	<10GB	>1000ms	0.701	0.490

Exercício desta semana: Trade-offs de eficiência e qualidade

O objetivo do exercício desta semana é construir alguns pipelines de busca e analisá-los em termos das seguintes métricas:

- Qualidade dos resultados: nDCG@10
- Latência (seg/query) considerando uma A100
- Custo de indexação (considerando 2 USD por A100 ou 0,1 USD por CPU core)
- Tamanho do índice em memória da CPU
- Custo por query assumindo utilização "perfeita": assim que terminou de processar uma query, já tem outra para ser processada
- Custo de deixar o sistema rodando para poucos usuários (ex: 100 queries/dia)

Iremos avaliar os pipelines usando TREC-COVID

Dicas:

- Utilizar modelos de busca "SOTA" já treinados no MS MARCO, como o [SPLADE distil](#) (esparsos), [contriever](#) (densos), [Colbert-v2](#) (densos), [minILM](#) (reranker), [monoT5-3B](#) (reranker), [doc2query minus-minus](#)