Apresentação DPR

Juliana Resplande: julianarsg13@gmail.com

Agenda: “simular” a leitura do paper

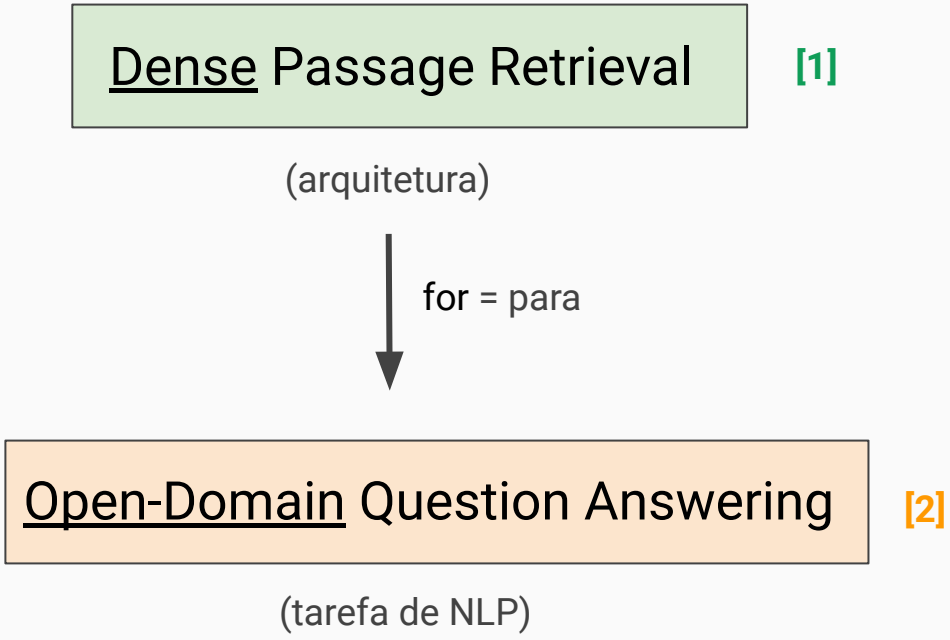
1. Título
2. *Abstract*/Conclusão
3. *Background*
4. Experimentos
5. Resultados

*sublinhei termos importantes em QA e NLP e em vermelho os essenciais em NLP

Obs: estamos em muitos momentos comparando o SBERT com o DPR

Título

Dense Passage Retrieval for Open-Domain Question Answering



O paper propõe uma arquitetura (DPR) [1] para uma tarefa de NLP (QA) [2]

Dense Passage Retrieval

(arquitetura)

- Passage retrieval
  - Busca de parágrafos
  - Primeira parte de um sistema de QA
  - Área comum de NLP com BD: Information Retrieval (IR)
- Dense
  - Denso/semântico
  - Usa embeddings de modelos de linguagem (de NLP) contextuais para fazer a busca
  - BM-25: esparso, usam algoritmos de *string matching*

DPR = busca de parágrafos, a qual usa embeddings/representações semânticas para fazer a busca.

Open-Domain Question Answering

(tarefa de NLP)

- Question Answering (QA)
  - Tarefa de NLP em que o modelo deve responder uma pergunta
- Open-Domain
  - Não fornece contextos específicos para a busca (parágrafos para encontrar a resposta)
  - Trata-se, geralmente, de *world knowledge*/*common sense*
  - Usa-se *Knowledge Bases* - KB como Wikipédia, Dbpedia, Freebase, etc.
  - SQuAD: *closed-domain* fornece contextos para as perguntas

Open-Domain QA = modelo deve responder uma pergunta sem ter contextos

Abstract/Conclusão

Contribuição

- “Nos mostramos que a busca pode ser realizada **usando somente as representações densas**”
  - Não necessita de modelos esparsos como BM25
- Supera o sistema robusto LuceneBM25 **de forma significativa em 9%-19%** em termos de **top-20** ***passage retrieval accuracy***

Observações

- Logo no começo da onda dos modelos Transformers, os pesquisadores já usavam e treinavam essa arquitetura State-Of-Art - SOTA em Information Retrieval e já testaram que era de LONGE superior do que o tradicional BM25.
- Mas, o paper do DPR foi o primeiro a sistematizar uma arquitetura com boas práticas:
  - Nomear as etapas (*retriever* e *reader*)
  - Mostrar como gerar e aproveitar exemplos negativos (ex. *in batch sampling*)
  - Indexação com FAISS
  - Pre-processamento

“In terms of top-20 passage retrieval accuracy”

- Em IR, usamos as métricas nos **top-K primeiros documentos**: *accuracy@k*, *recall@k*, *precision@k*, *f1@k* (média nos K primeiros resultados)
- São comuns: *accuracy@10*, *accuracy@20*, *accuracy@100*.
- *Accuracy*@20 (lê-se *accuracy at* 20)
- Além disso, existem métricas de ranqueamento como MRR@k e NDCG@k, que consideram a **ordem dos documentos retornados**.

Background

Descrição do problema

1. Configuração de *open-domain* QA com *extractive* QA (corta a resposta do trecho)
2. Inicialmente uma coleção C com D documents, $C = \{d_1, d_2, \ldots, d_D\}$
3. Pré-processamento: separamos cada documento d_i em trechos de tamanhos iguais*

   (Haystack: 300 palavras e não tokens)

   a. O corpus C' pode ser visto como uma coleção de M trechos: $C' = \{p_1, p_2, \ldots, p_M\}$

   b. Trecho p_i pode ser visto como uma sequência de tokens: $p_i = \{w_1, w_2, \ldots, w_{|pi|}\}$.

Problema depois do pré-processamento:
Dada questão/*query* q, a tarefa é encontrar um *span* $\{w_s, w_{s+1}, \ldots, w_{|e|}\}$ de uma dos trechos p_i que possa responder a questão.

*modelos de linguagem da família BERT usam necessariamente 512 tokens. Espaços vazios são preenchidos (*padded*) com [PAD]. Por padrão, os modelos do framework Transformers truncam (*truncation=True*) as sentenças.

Descrição da solução: sistema de QA

Dado um corpus $C = \{p_1, p_2, \ldots, p_M\}$ e uma query q, o sistema de QA possui o seguinte pipeline:

1. Retriever R eficiente: filtra um conjunto menor de trechos C_F de tamanho k
   - $R(q, C) = C_F$, onde $|C| >> |C_F| = k$
   - Avaliar o retriever R usando acurácia@k
2. Reader: dá um score para pedaço de cada trecho em C_F

Observações do capítulo seguinte - 3. Dense Passage Retriever (DPR):

- M pode ser gigante (e.g., 21 milhões de trechos)
- k geralmente é pequeno, entre 20–100

Observação: Estrutura de paper

- Quase todo bom paper define matematicamente o seu problema, bem como o algoritmo de solução
- Segue a estrutura:
  - Título: chamativo com o nome do modelo e o que ele resolve
  - *Abstract*: o problema, a ideia e os resultados
  - Introdução: descreve melhor o problema e os outros têm feito “revisão sistemática”
  - Metodologia
    - Arquitetura proposta com definição matemática
    - Experimentos realizados: modelos, *datasets*, detalhes
  - Resultados
  - Conclusão

Dense Passage Retriever (DPR)

Geral

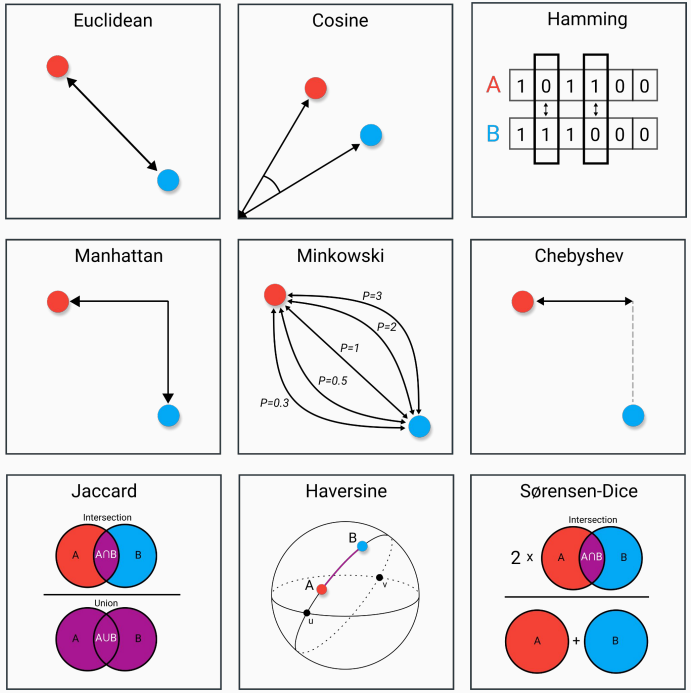
- Cria um índice para toda as M *encoded passages*/embeddings que vamos usar para a busca (FAISS).
- **Dois** modelos BERT treinado são utilizados para encode *query* e *passages*:
  - Query Encoder E_Q
  - Passage Encoder E_P
- Retriever $R(q, C)$ para a *query* q and contexto C:
  - Calcula a similaridade $sim(q, p)$ para cada trecho p em C
  - Filtra k trechos com maiores similaridades
    - Onde $sim(q, p) = E_Q(q) \cdot E_P(p)$
    - (*dot product* entre embeddings da *passage*/trecho e da *query*)

Observação: DPR e MSMARCO

- Reader do DPR usa dois *encoder* BERT para gerar *embeddings*:
  - Um voltado para perguntas e o outro em respostas.
  - É mais custoso que usar o mesmo *encoder* para perguntas e respostas
- Modelos recentes usam só um encoder e são treinados no MSMARCO superam de longe o DPR
  - MSMARCO possui cerca de **100 MILHÕES** de exemplos e atualmente usando ele “puro” têm batido muitos estados da arte mesmo em domínios específicos.
  - Além disso, o MSMARCO é mais rico em informações em cada exemplo como ordem de relevância trechos retornados, tornando-mais adequado para ranqueamento
  - Apesar disso muitas pessoas ainda mencionam usam o DPR pela análise exploratória da área

Observação: Distância de vetores

- Existem várias formas de calcular a distância (ou a sua recíproca, a similaridade) de embeddings/vetores:
  - As mais usadas na busca semântica são cosseno e *dot product**.
  - A distância cosseno é praticamente o dot product com uma normalização, em que os valores variam entre [-1, 1]. {-1: são a negação um do outro, 0: neutro, 1: idênticos}
  - O *dot product* é prefiro pelas bibliotecas de otimização, como o FAISS economizando a etapa de normalização, mas perde-se a noção de valores absolutos.



$$\text{cossine similarity}(v_1, v_2) = \frac{v_1 \cdot v_2}{|v_1| \cdot |v_2|}$$

*Já usei manhattan e euclidean como métricas de busca semântica, enquanto Levenstein [hamming em que A, B podem ter tamanho distintos] em algoritmo de correção ortográfica.

Observação: formas de similaridade semântica

Duas formas padrões de usar *Encoder* para gerar similaridade de perguntas e parágrafos:

1. **Cross-encoder**: Fornece as sentenças **concatenadas**: pergunta [SEP] parágrafo (classificação normal do BERT)
    a. Gera um embedding e faz a classificação padrão de um *Encoder* BERT
    b. **Melhores resultados**: melhor para **ranqueamento de candidatos** K, muito baixo
    c. Muito custoso: "Finding the most similar pair from **65 hours** with BERT / RoBERTa to about **5 seconds** with SBERT, while maintaining the accuracy from BERT"
2. **Bi-encoder**: Fornece cada pergunta e parágrafo **separadamente** para o modelo (DPR e SBERT)
    a. Gera dois embeddings separados pelos Encoders e calcula a similaridade entre eles
    b. **Eficiente**: melhor para **busca semântica** em temos **K candidatos altíssimo**
        i. Permite pré-computar os parágrafos da base e armazená-los em um índice como o FAISS.
        ii. Não precisa de usar um módulo de regressão em cima do *Encoder*

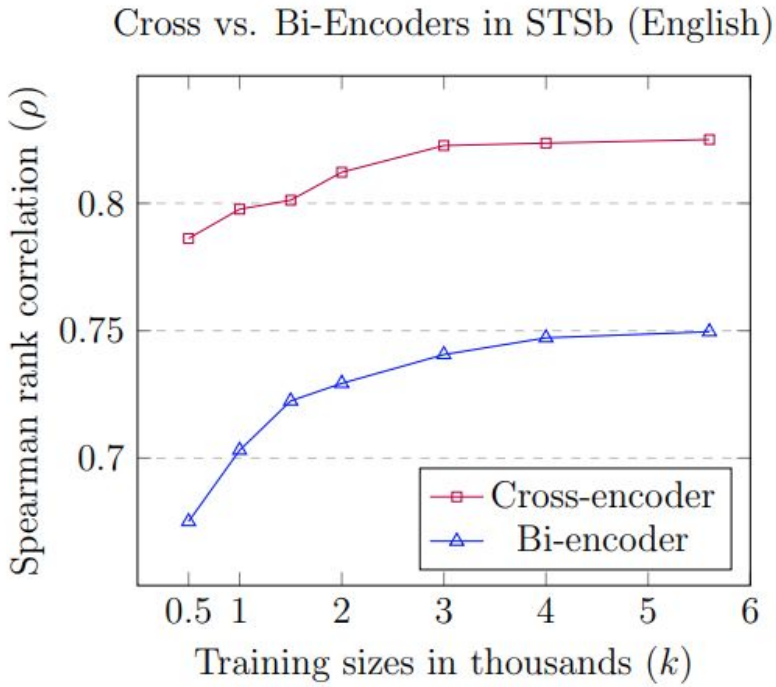


Figure 1: Spearman rank correlation (ρ) test scores for different STS Benchmark (English) training sizes.

Observação: Sentence Embeddings - DPR X SBERT

- Os *encoders* fornecem embeddings a nível de tokens e não para parágrafos, mas no cálculo da similaridade fazemos similaridade de embeddings de parágrafos.
- Como gerar embeddings de um parágrafo a partir de uma lista embeddings de tokens?
    - Pooling/redução dos embeddings
        - Max
        - Mean: SBERT
        - [CLS] -> padrão do BERT voltado para classificação/regressão: DPR

SBERT: “Average BERT embeddings or using the [CLS] token output from a BERT network achieved bad results for various STS tasks”.

Treinamento

Loss (Função objetivo, função de custo, indica a direção do gradiente dos pesos):

- Seja documento D com m instâncias,
- Cada instância D_i possui uma questão q_i e um trecho relevante/positivo p^+_i e n trechos irrelevantes/negativos $p^-_{i,j}$

$$D = \{q_i, p^+_i, p^-_{i,1}, \cdots, p^-_{i,n}\}_{i=1}^{m}$$

Negative log likelihood do trecho positivo:

$$L(q_i, p_i^+, p_{i,1}^-, \cdots, p_{i,n}^-) \quad (2)$$
$$= -\log \frac{e^{\text{sim}(q_i, p_i^+)}}{e^{\text{sim}(q_i, p_i^+)} + \sum_{j=1}^{n} e^{\text{sim}(q_i, p_{i,j}^-)}}.$$

- *Negative log likelihood* = *Cross-entropy loss*
- *Loss* de problema de classificação:
- p^+_i deve ser classificado como relevante e p^-_i deve ser classificado como irrelevante.

Observação: Treino em NLI

- Como a loss indica, os treinos de busca semântica são tratados como classificação
  - Trechos relevante ou não.
- A tarefa de “classificação de pares de sentenças” é conhecido em NLP como NLI/RTE
  - Problema: B implica A $(\mathrm{B} \rightarrow \mathrm{A})$?
  - NLI = Natural Language Inference
  - RTE = Recognizing Textual Entailment
  - Possuem duas ou três *labels*
    - entailment, not_entailment
    - entailment, neutral, contradiction

A: A soccer game with multiple males playing.

B: Some men are playing a sport.

Label: **entailment** SNLI

Observação: NLI ou RTE? (nem eu sei: mas a tendência é NLI, RTE é de velho)

Natural Language Inference or Recognizing Textual Entailment?

The terms Natural Language Inference (NLI) and RTE are often used interchangeably. Many papers begin by explicitly mentioning that these terms are synonymous (Liu et al., 2016; Gong et al., 2018; Camburu et al., 2018).[1] The broad phrase "natural language inference" is more appropriate for a class of problems that require making inferences from natural language. Tasks like sentiment analysis, event factuality, or even question-answering can be viewed as forms of natural language inference without having to convert them into the sentence pair classification format used in RTE. Earlier works used the term *natural language inference* in this way (Schwarcz et al., 1970; Wilks, 1975; Punyakanok et al., 2004).

The leading term *recognizing* in RTE is fitting as the task is to classify or predict whether the truth of one sentence likely follows the other. The second term *textual* is similarly appropriate since the domain is limited to textual data. Critics of the name RTE often argue that the term *entailment* is inappropriate since the definition of the NLP task strays too far from the technical definition from *entailment* in linguistics (Manning, 2006). Zaenen et al. (2005) prefer the term *textual inference* because examples in RTE datasets often require a system to not only identify entailments but also conventional implicatures, conversational implicatures, and world knowledge.

https://azpoliak.github.io/publications/rte-survey--eval4nlp.pdf

Observação: Treino em NLI - DPR x SBERT

- DPR
  - Escolhe um dataset de QA
  - Pega a pergunta, a resposta e gera contextos negativos

```
[
  {
        "question": "....",
        "answers": ["...", "...", "..."],
        "positive_ctxs": [{
                "title": "...",
                "text": "...."
        }],
        "negative_ctxs": ["..."],
        "hard_negative_ctxs": ["..."]
  },
  ...
]
```

- SBERT (antes do MSMARCO)
  - Treinava com dataset de NLI tradicionais como SNLI

A: A soccer game with multiple males playing.

B: Some men are playing a sport.

Label: **entailment**

SNLI

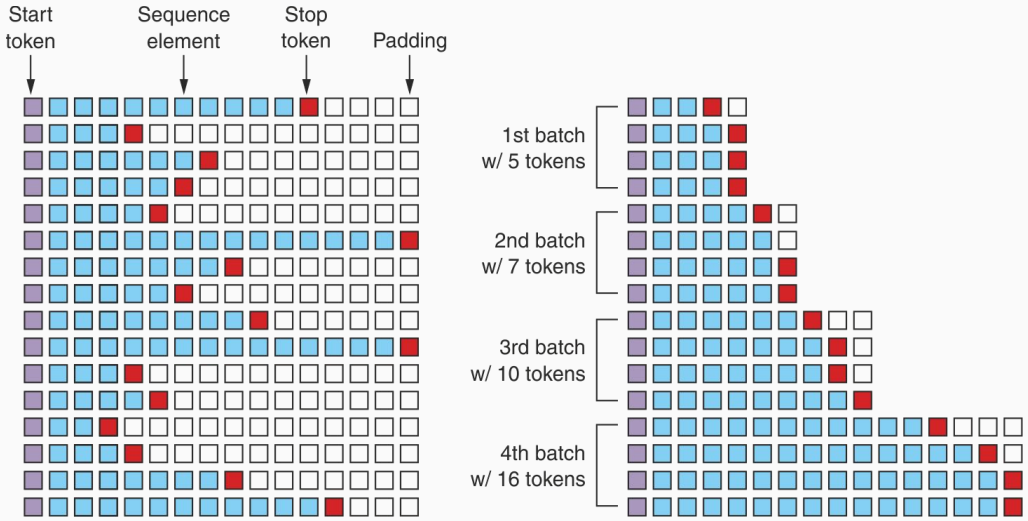
Otimizações do treino

Como obter trechos irrelevantes/negativos $p^-_{i,j}$ para instância instância D_i?

1. Random: qualquer trecho aleatório do *Corpus*
2. BM25: top trechos retornados pelo BM25 que não possuem a resposta mas têm match com os tokens da questão (hard negatives)
3. Gold: trechos positivos de outras questões do conjunto de treino (In-batch negatives)

- O nosso melhor modelo use trechos gold do mesmo mini-batch (In-batch negatives) e um trecho negativo do BM25.
    - Quanto mais trechos irrelevantes no cálculo da loss melhor, ou seja, quanto maior o batch melhor. **Vira uma questão de hardware**
- Adicionando somente um trecho negativo do BM25 melhora substancialmente os resultados, enquanto adicionando dois não melhora muito.

Observações: Técnica **faltante** - *Smart batching*



Model	CPU	GPU
Avg. GloVe embeddings	6469	-
InferSent	137	1876
Universal Sentence Encoder	67	1318
SBERT-base	44	1378
SBERT-base - smart batching	83	2042

Table 7: Computation speed (sentences per second) of sentence embedding methods. Higher is better.

- *Smart batching*: Ordenar os exemplos de treino em tamanho de sentença
  - Agrupa sentenças de tamanho parecido, fazendo com que o *batch* tenha pouco [PAD]
  - SBERT possui e o DPR não possui

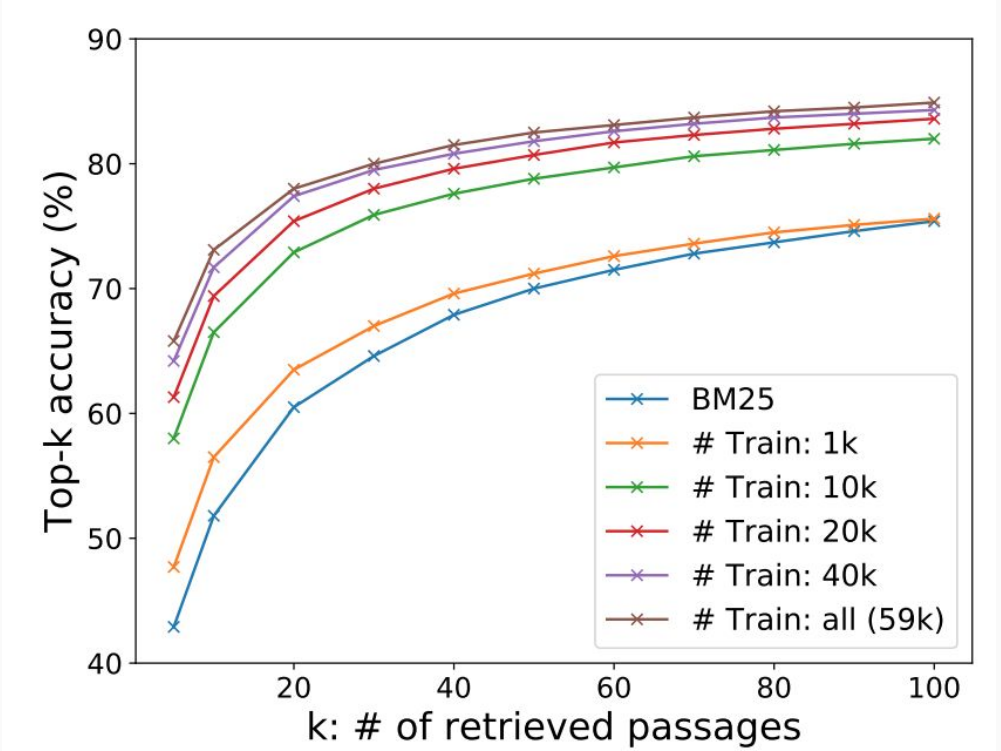
Resultados

Retriever

- Com exceção do SQuAD, DPR supera o BM25 em todos os datasets.
  - Já esperado, pois essa forma de busca semântica já era feita informalmente e era bem superior.
- Essa diferença de preformance é especialmente grande quando k is small (ex. 78.4% contra 59.1% na accurácia@20 do Natural Questions)
- As duas possíveis razões para menor perfomance no SQuAD.
  - Os anotadores escreveram as questões depois de verem os trechos. Nisso, existem muitas palavras em comum entre os trechos e as questões, favorecendo o BM25.
  - Os dados foram coletados em apenas 500+ artigos da Wikipedia e com isso a distribuição dos exemplos de treino possui *bias*, como argumentado anteriormente por Lee et al. (2019).

Observação: a escolha do top-K

- Top-K 20 e 40
- Maior K
  - Maior acurácia
  - Maior tempo inferência
  - Mais trechos para o reader
- Uso de reranqueador



Reader

Training	Model	NQ	TriviaQA	WQ	TREC	SQuAD
Single	BM25+BERT (Lee et al., 2019)	26.5	47.1	17.7	21.3	33.2
Single	ORQA (Lee et al., 2019)	33.3	45.0	36.4	30.1	20.2
Single	HardEM (Min et al., 2019a)	28.1	50.9	-	-	-
Single	GraphRetriever (Min et al., 2019b)	34.5	56.0	36.4	-	-
Single	PathRetriever (Asai et al., 2020)	32.6	-	-	-	**56.5**
Single	$\text{REALM}_{\text{Wiki}}$ (Guu et al., 2020)	39.2	-	40.2	46.8	-
Single	$\text{REALM}_{\text{News}}$ (Guu et al., 2020)	40.4	-	40.7	42.9	-
Single	BM25	32.6	52.4	29.9	24.9	38.1
	DPR	**41.5**	56.8	34.6	25.9	29.8
	BM25+DPR	39.0	57.0	35.2	28.0	36.7
Multi	DPR	**41.5**	56.8	**42.4**	49.4	24.1
	BM25+DPR	38.8	**57.9**	41.1	**50.6**	35.8

Table 4: End-to-end QA (Exact Match) Accuracy. The first block of results are copied from their cited papers. $\text{REALM}_{\text{Wiki}}$ and $\text{REALM}_{\text{News}}$ are the same model but pretrained on Wikipedia and CC-News, respectively. *Single* and *Multi* denote that our Dense Passage Retriever (DPR) is trained using individual or combined training datasets (all except SQuAD). For WQ and TREC in the *Multi* setting, we fine-tune the reader trained on NQ.

Observação: pouca atenção ao Reader

- Reader/Gerador de resposta possui dois tipos
  - Extrativo
    - Treinado no SQuAD
    - Extremamente consolidado da área
  - Abstrativo
    - Modelos seq2seq
    - Modelos pesados
    - Gera muitas alucinações
    - Pouquíssimo consolidado na área

Conclusão

Resuminho

- Sistema de QA
  - Retriever, busca semântica (Information Retrieval) + Reader, geração de resposta (NLP)
- Information Retrieval
  - Modelos densos, busca semântica e Encoder x Modelos esparsos, busca lexical e BM25
  - Indexação com FAISS
  - Geração de similaridade semântica: duas formas
  - Métricas de distância de embeddings
  - Treino em NLI
  - Otimizações: Uso de negativos na loss e *smart batching*
  - Custo benefício na escolha do top-K
  - Reader: extrativo e abstrativo

SBERT x DPR: resuminho

SBERT

- Agosto de 2019
- Framework: Sentence-Transformers
- Somente Retriever
- Treino em dataset de NLI genérico
- Retriever siamês adaptado para busca ✔️
- Usa distância cosseno
- Usa média como *pooling* ✔️
- *Loss* de *cross-entropy* "rápida"
- Otimizações
  - *Smart-batching* ✔️

DPR

- Abril de 2020
- Framework: Haystack
- QA: Retriever e Reader
- Treino em dataset de QA
- Retriever padrão com **DOIS** encoders ❌❌
- Usa *dot product* ✔️
- Usa [CLS] como *pooling*
- *Loss* de "ranqueamento" ✔️
- Otimizações
  - Usa *loss* com múltiplos negativos
  - Escolha de negativos✔️
  - Indexação ✔️
  - Quebra sentença em 300 ✔️

Adendo: Sentence-Transformers continuou melhorando (framework)

Sentence-Transformers

- Lançou mais papers
- Opção de treino voltado para busca semântica em QA:
  - Opção de dataset MSMARCO e outros voltados para QA
  - Incorporou
    - Loss de raqueamento
    - Uso de negativos gold em batch
    - Quebra em tamanho 300
- Outras opções de treino:
  - Cross-encoder (estilo DPR, melhor para ranqueamento)
  - *Knowledge Distillation* para treino multilíngue
  - Não supervisionado
  - Aumento de negativos usando BM25
- Lançou benchmark: BEIR

Obrigada!