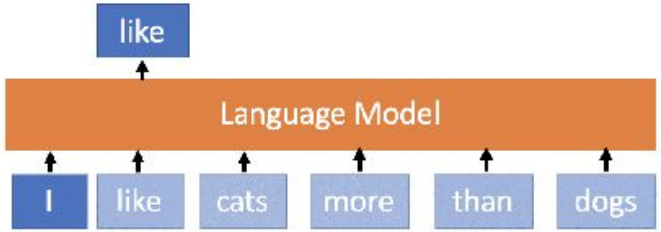
BERT

Bidirectional Encoder Representations from Transformers

Modelos de língua

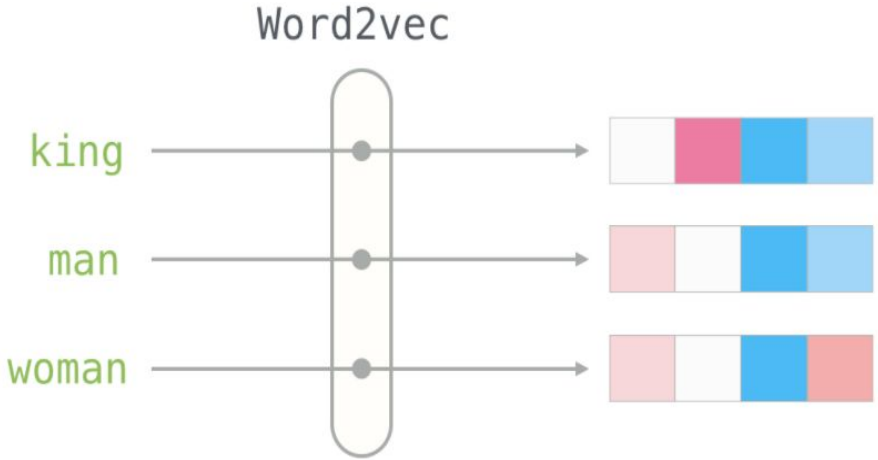
O que é um modelo de língua?

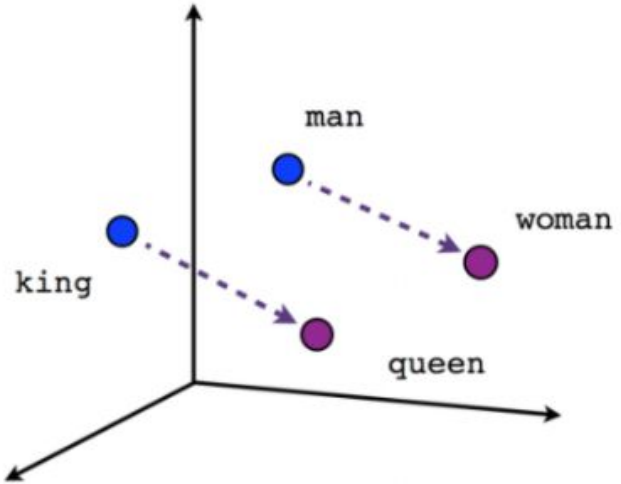
Um modelo de língua procura prever a próxima palavra em uma sentença:



Ou seja, dado uma sequência de palavras x(1), … x(n), queremos a distribuição de probabilidades para x(n+1)

Word embeddings





Male-Female

Word embeddings

Apesar desse tipo de representação ser muito útil, temos algumas limitações, como:

```
import spacy
```

```
nlp = spacy.load('pt_core_news_lg')
```

```
w1 = 'A manga de sua camiseta estava suja'
w2 = 'A manga de sua sobremesa estava estragada'
```

```
doc1 = nlp(w1)
doc2 = nlp(w2)
```

```
doc1.similarity(doc2)
```

```
0.9686704295888113
```

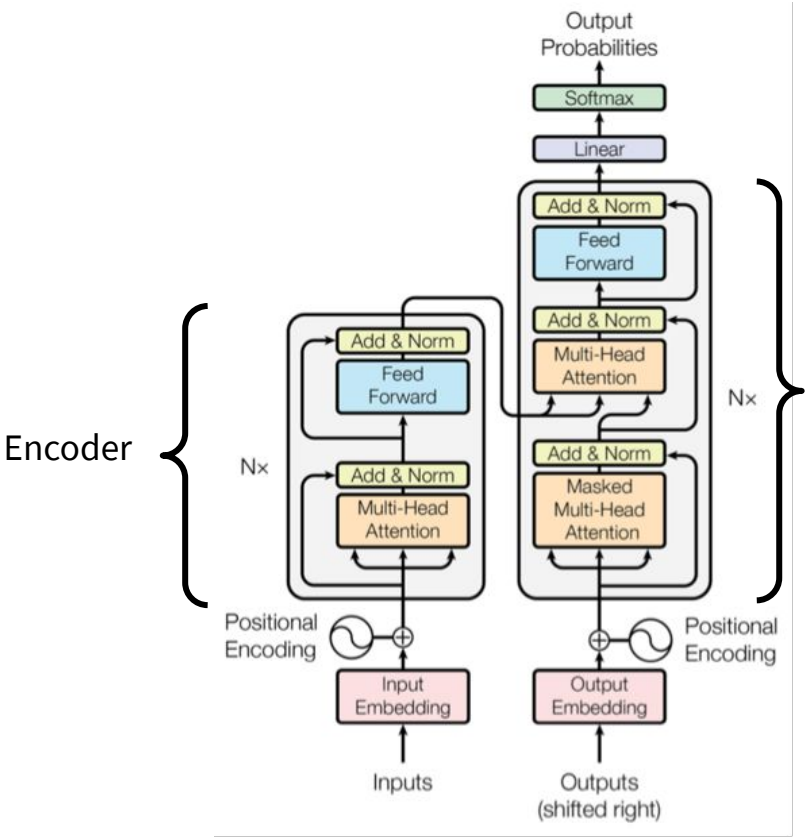
Não estamos capturando contexto com essa representação!

ELMo: Embeddings from language models

Ao invés de utilizar um embedding fixo para cada palavra, ELMo utiliza uma LSTM bi-direcional para calcular os seus vetores, ou seja, suas representações vão ser calculadas através de observações da sentença **inteira**

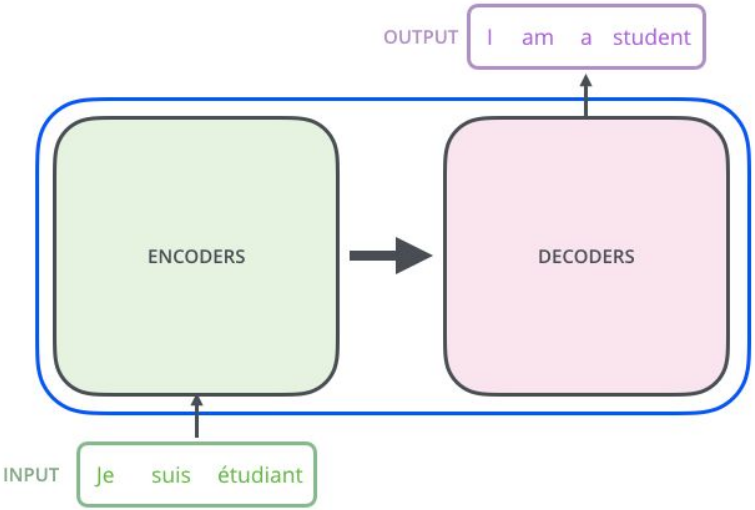
O Transformer original

Transformer

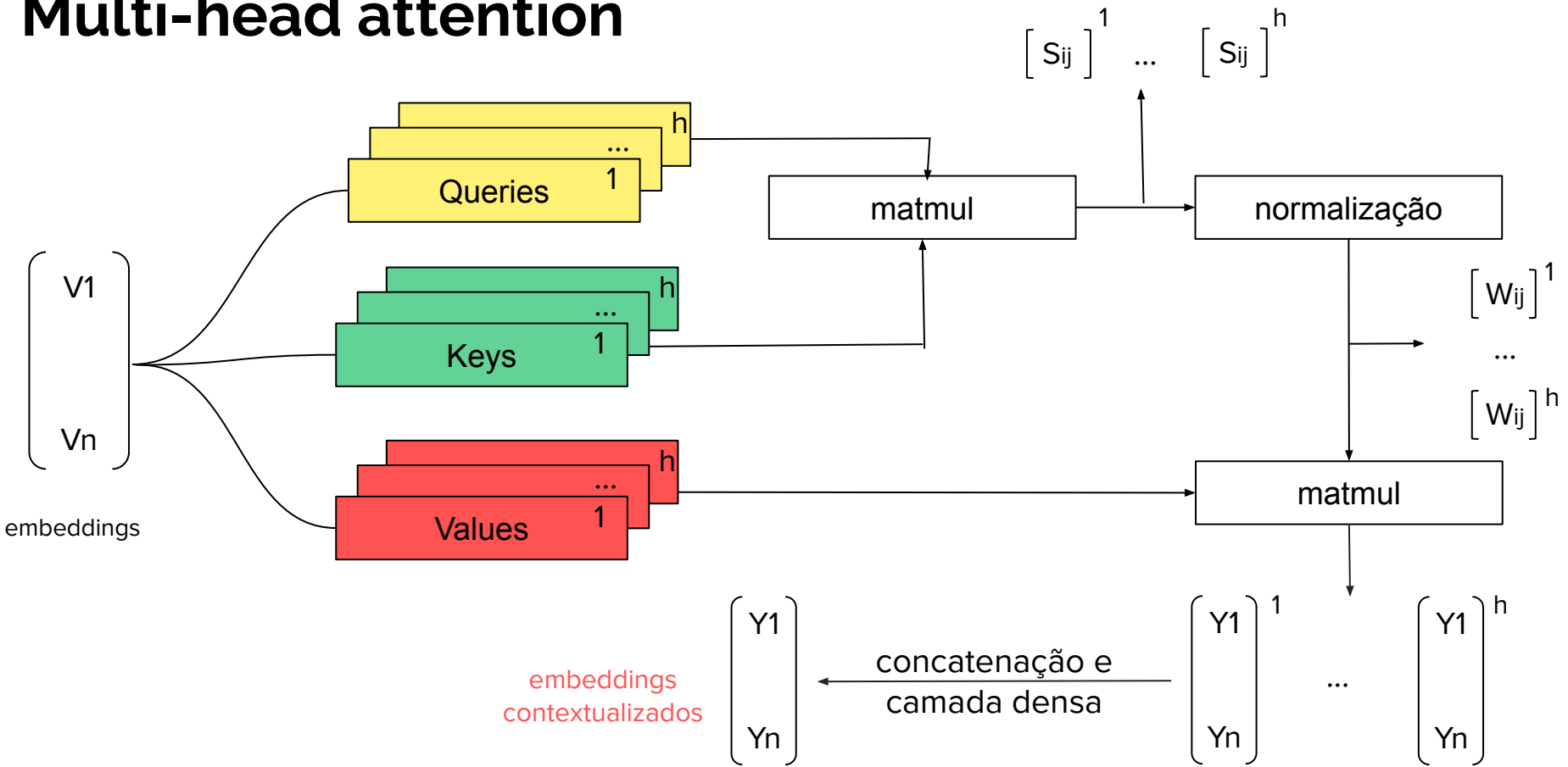


Encoder

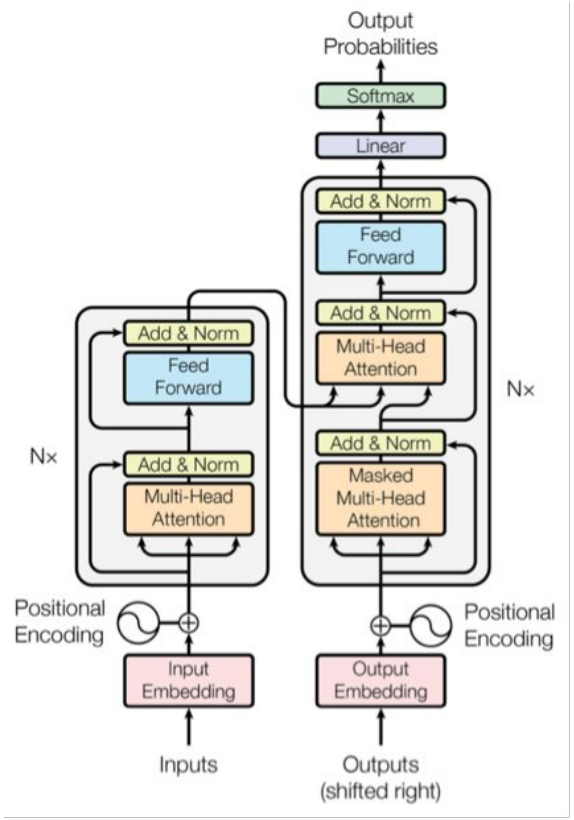
Decoder



Multi-head attention



Transformer



Com a arquitetura Transformer, passamos a não utilizar mais a estrutura de redes recorrentes, apenas as camadas de atenção - como sugerido pelo paper que introduz essa estrutura: "Attention is all you need".

Como não precisamos processar o texto de forma sequencial, agora podemos paralelizar as nossas operações. Com isso, modelos de arquiteturas muito mais profundas puderam ser introduzidos.

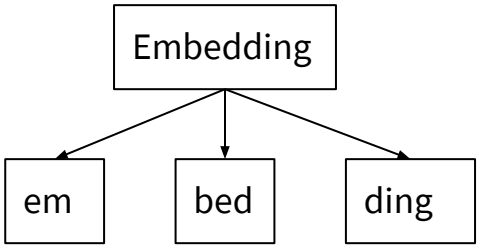
https://github.com/turing-usp/Aulas-NLP/tree/main/Transformers

Vamos para o BERT

Sentence Piece

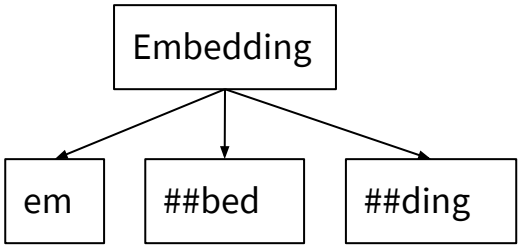
Vocabulário do BERT

- O BERT original tem um vocabulário de 30K tokens fixo
- Para palavras OOV (que não estão no vocabulário) os embeddings são obtidos da seguinte forma:
  - A palavra é quebrada em subwords, para os quais o BERT possui embeddings
    - Fasttext vibes -> mas é diferente, pois não é feita uma média entre os tokens de subwords e os tokens inteiros
  - No limite, o BERT possui embeddings para todos os caracteres individuais, eles podem ser usados para calcular o embedding de qualquer palavra



Tipos de pedaço (subwords)

- Todas as subwords começam com ##
  - Com exceção da primera subword
- Isso não tem uma explicação clara, mas poderia ser para coincidir os embeddings de palavras e subwords que sejam “raízes”/ “radicais”



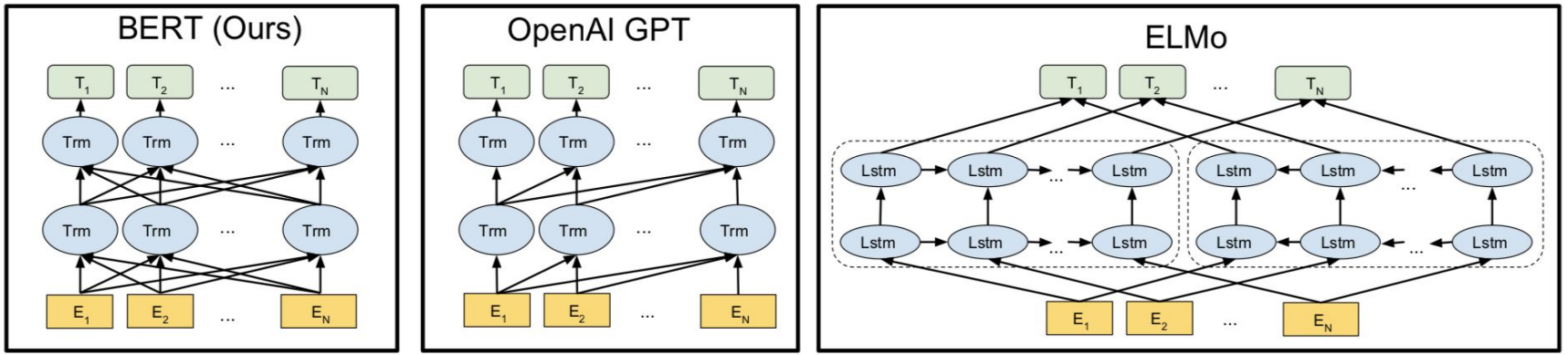
Sentencepiece

```
import sentencepiece as spm

params = ('--input=input.txt ' 
          '--model_prefix=spm ' 
          '--vocab_size=1000')
spm.SentencePieceTrainer.Train(params)

sp = spm.SentencePieceProcessor()
sp.Load('spm.model')

print(sp.EncodeAsPieces('Hello world.'))
print(sp.EncodeAsIds('Hello world.'))
print(sp.DecodeIds([151, 88, 21, 887, 6]))
```

Normalizer	Trainer	Encoder	Decoder
Lower case, utf-8 e etc	Aprender embeddings	tokenization	detokenization

Arquitetura BERT

Diferença na Bidirecionalidade



Pré-treinamento do BERT

O pré-treinamento do BERT é feito com dois modelos:

1. Modelagem de Linguagem Mascarada (MLM - Masked Language Model)

2. Predição da próxima sentença (NSP - Next Sentence Prediction)

1. Modelo de Linguagem Mascarada

A ideia é fazer com que o modelo não se vicie já que por ser um modelo bidirecional, a palavra pode "se enxergar" indiretamente, o que pode enviesar o modelo como um todo.

Para isso, mascara uma parte das palavras (15 % no caso), de 3 formas diferentes:

Exs: my dog is hairy

1. my dog is [MASK] - 80%

2. my dog is apple - 10%

3. my dog is hairy - 10%

2. Predição da próxima sentença

Predição binária (classificação) se é ou não a próxima sentença

Escolha das sentenças A e B para o pré-treino:

- 50% é sequência da sentença (B é sequência de A)

- 50% é aleatório (B é uma sentença aleatória do corpus)

Fine-tuning

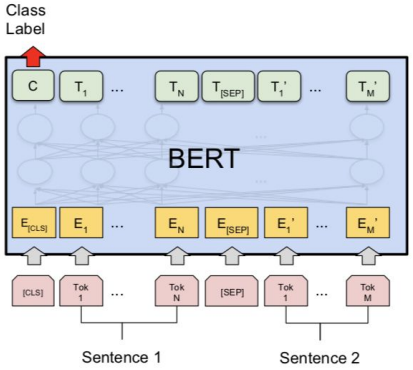
A ideia do BERT é que você vai precisar fazer o mínimo de alteração possível. Uma das grandes vantagens desse modelo é que como o modelo é muito poderoso (e complexo) então acaba tendo um Fine-tuning mais simples e rápido.

- Batch: 16,32

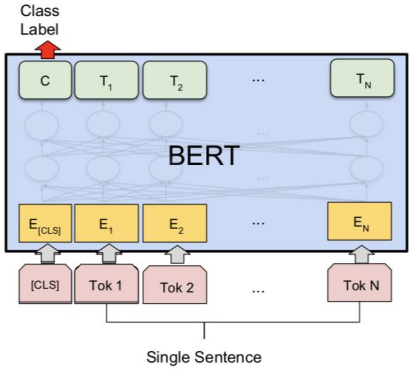
- Taxa ADAM: 5 x 10^(-5), 3 x 10^(-5) e 2 x 10^(-5)

- Épocas: 3 e 4

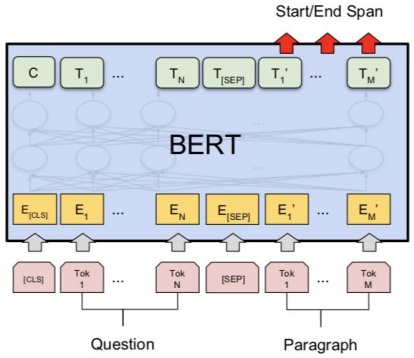
Fine-tuning



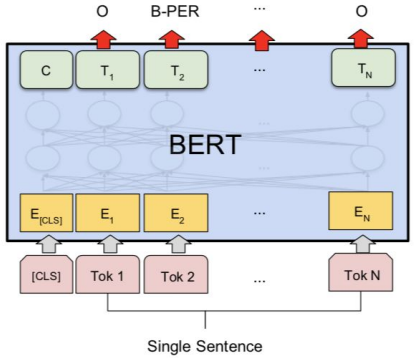
(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG



(b) Single Sentence Classification Tasks: SST-2, CoLA



(c) Question Answering Tasks: SQuAD v1.1



(d) Single Sentence Tagging Tasks: CoNLL-2003 NER