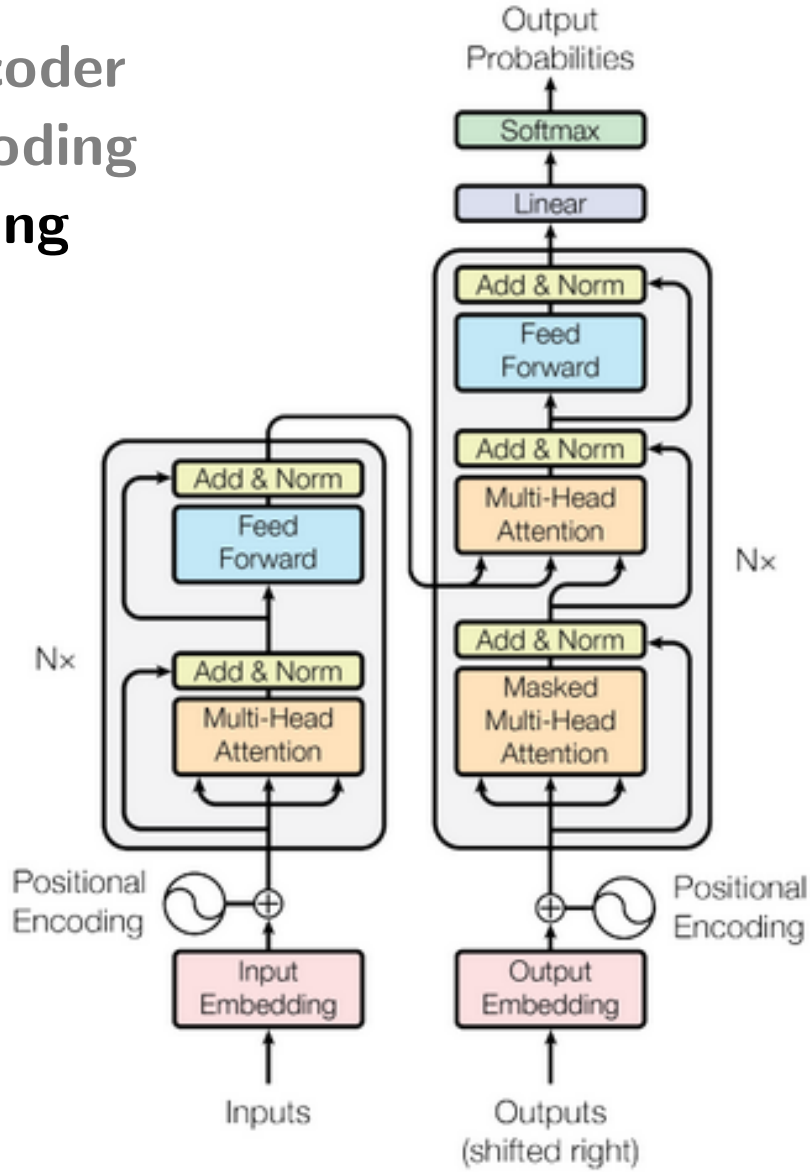
MAC5921 – Deep Learning

Aula 14 – 10/10/2023

Transformers – parte 3

Nina S. T. Hirata

Encoder / Decoder
Positional Encoding
Input Embedding



NLP: processamento de linguagem natural

Dados do tipo **texto** (sequência de caracteres, string)

Tipicamente alguns pré-processamentos são comumente aplicados para transformar o texto em uma **sequência de tokens** (unidades de informação que possam ser vistos como elementos discretos que compõem o texto)

Associa-se então uma representação (**embedding**) a cada *token*

Esta aula

1. Digressão (sobre NLP antes de DL)
2. *Embedding* – exemplo: `word2vec`
3. *Transformer* LLM (Large Language Models): BERT $\times$ GPT
4. *Tokenization*

Uma digressão, novamente

NLP antes de *machine learning*

- **Tokenizing** – quebrar o texto em pedaços (ex: palavras)
- **Stemming** – reduzir palavras ao seu radical (não necessariamente uma palavra)
- **Lemmatizing** – reduzir palavras a sua correspondente raiz (uma palavra válida); é uma alternativa à lematização

Isto gera um vocabulário (dicionário) de termos

Exemplo: Representação de texto (ou documento)

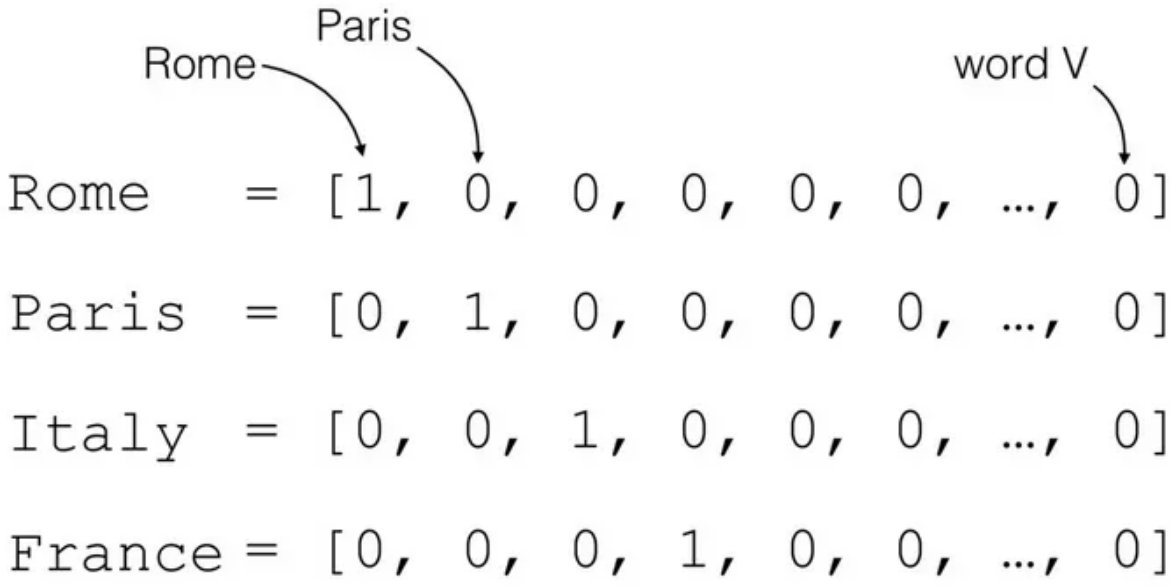
Vamos considerar um universo D de documentos e V possíveis termos (palavras) $\mathrm{t}_1, \mathrm{t}_2, \ldots, \mathrm{t}_{\mathrm{V}}$

Dado um texto (ou documento) $\mathrm{D}_\mathrm{j} \in \mathrm{D}$, podemos representá-lo numericamente por um array (vetor) booleano V-dimensional

$$\mathrm{b}_\mathrm{j}[\mathrm{i}] = 1 \Longleftrightarrow \mathrm{t}_\mathrm{i} \text{ ocorre em } \mathrm{D}_\mathrm{j}$$

b pode ser pensado como um *bag of words*

Em particular, se pensarmos que cada termo é um texto, temos o **one-hot encoding** para os termos



TF – Term frequency

Em vez de um vetor booleano, um vetor com a frequência

$$\mathrm{tf}(\mathrm{t}, \mathrm{D_j}) = \frac{\mathrm{f_{t,D_j}}}{\sum_{\mathrm{t'} \in \mathrm{D_j}} \mathrm{f_{t',D_j}}}$$

$\mathrm{f_{t,D_j}}$: quantas vezes o termo t ocorre no documento $\mathrm{D_j}$

tf pode ser pensado como um *bag of words* ponderado

Há variantes que consideram diferentes formas para ponderar ocorrência de termos (veja, por exemplo, wikipedia)

IDF (Inverse document frequency)

Termos que aparecem em todos os N documentos são pouco informativos; ao contrário, os raros é que podem revelar algo

$$\mathrm{idf}(t) = \log \frac{N}{n_t} \qquad \text{ou} \qquad \mathrm{idf}(t) = \log \frac{1+N}{1+n_t}$$

$n_t = \left|\{D_j \in D : t \in D_j\}\right|$: número de documentos que contém t

TF-IDF

Frequência de um termo t em $\mathrm{D_j}$ é relevante, mas se ele ocorre na maioria dos documentos, a relevância é menor

$$\mathrm{tfidf(t, D_j) = tf(t, D_j) * idf(t)}$$

Exemplo: Classificador spam/não-spam de emails

- Criar o dicionário (V termos)
- Para cada email, calcular a sua representação tfidf
- Treinar o classificador spam/não-spam usando as representações tfidf

Exemplo: Busca de documentos similares

Recuperar os k-NNs do documento *query*, usando por exemplo a similaridade coseno entre as representações tfidf

Alguns problemas do *one-hot encoding*

- representação muito esparsa
- tamanho do dicionário = tamanho da representação
- não considera contexto na representação (codificação é fixa, independentemente da semântica associada
  Ex: manga, para, casa

Fim da digressão

Como fazer embeddings ?

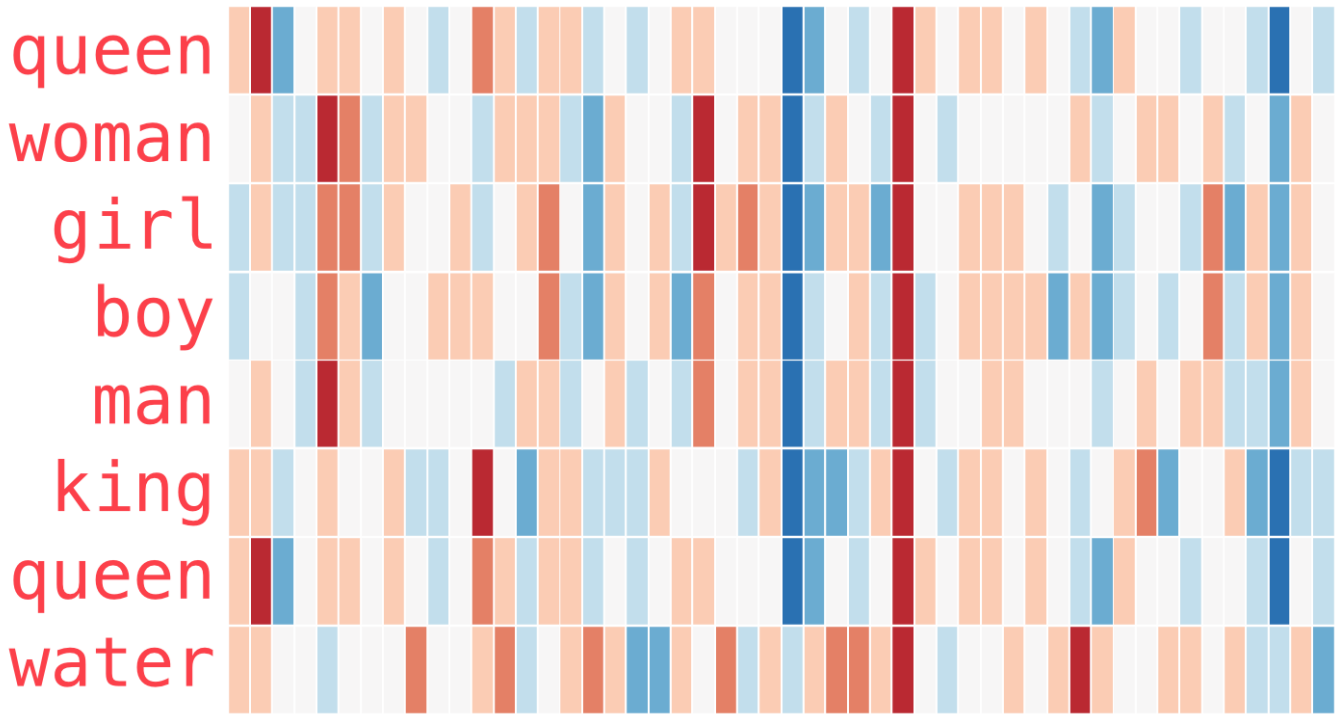
Como projetar vetores *one-hot encoded* de dimensão V para um espaço de dimensão d? $\mathrm{d} \ll V$

Palavras com significados similares deveriam ter representações similares

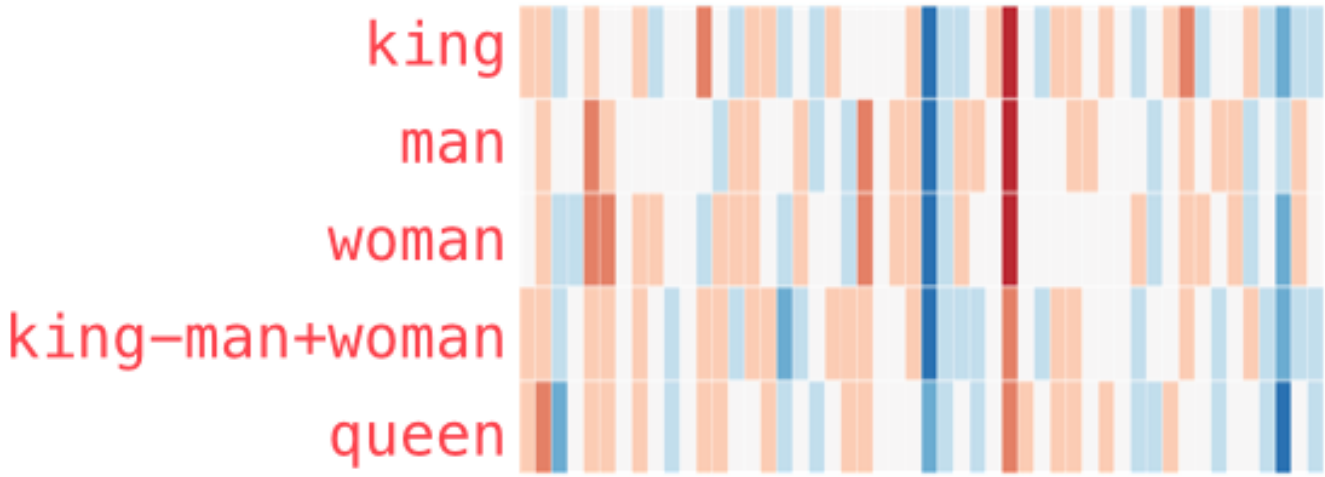
Usar *unsupervised ML*: ex. GloVe e word2vec

Visualização de *embedding* ($\mathrm{d} = 50$) aprendido pelo **GloVe**: *Global Vectors for Word Representation*

https://nlp.stanford.edu/projects/glove/



king − man + woman ~= queen



CBOW (*Continuous bag of words*)

Modelo simples: dada uma sequência de *tokens* e um *token* $\mathtt{x}$ específico nela, predizer o próximo *token* $\mathtt{y}$ da sequência

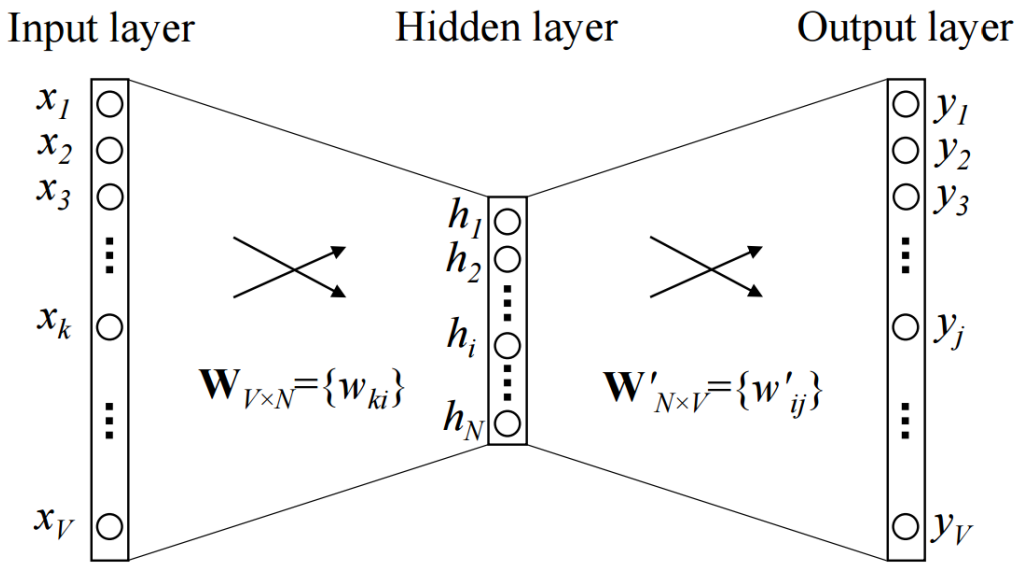
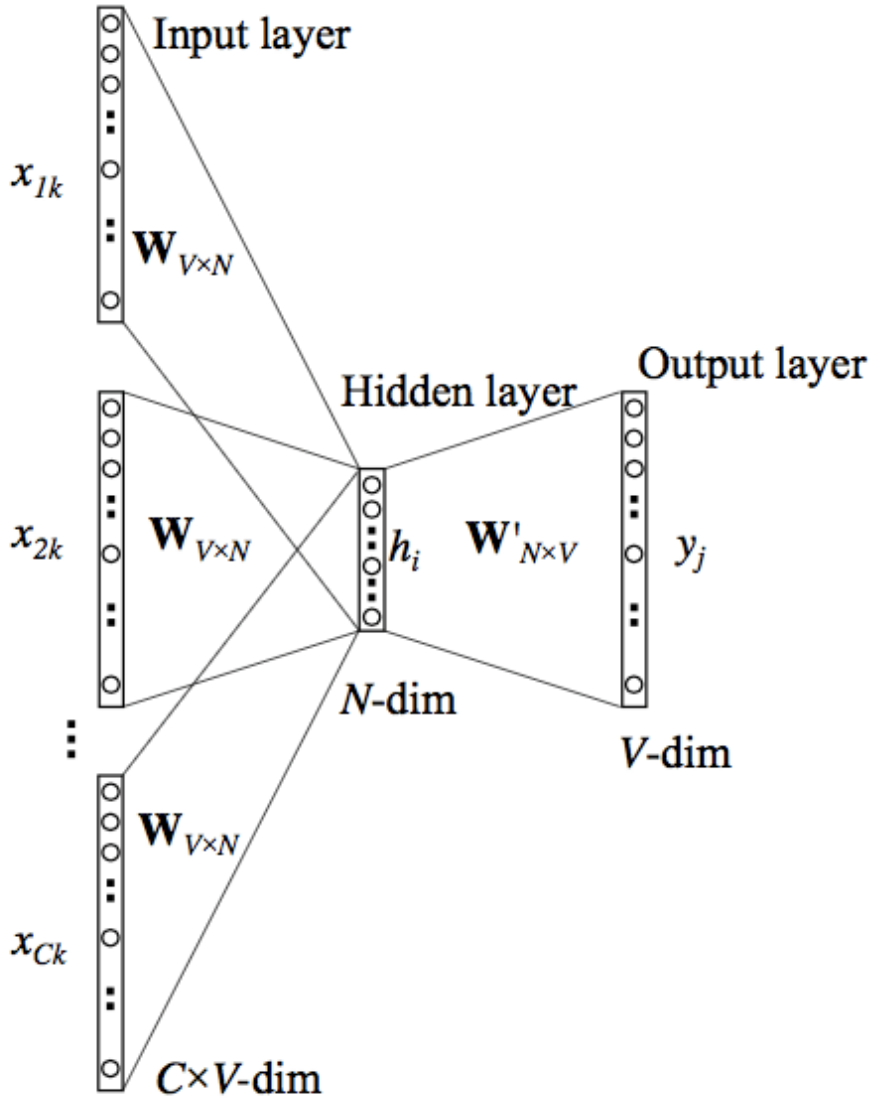


Figure 1: A simple CBOW model with only one word in the context

($\mathtt{x}$ funciona como **contexto** para ajudar o cálculo do *embedding* de $\mathtt{y}$)

($\mathtt{x}$ e $\mathtt{y}$ estão na forma *one-hot encoding* e $\mathtt{h} \in \mathbb{R}^{\mathtt{d}}$ é o *embedding* de $\mathtt{y}$)

Ampliar contexto: $\mathtt{C}$ *tokens* de contexto para predizer o *token* alvo $\mathtt{y}$ ($\mathtt{h} \in \mathbb{R}^{\mathtt{d}}$, *embedding* de $\mathtt{y}$, pode ser o vetor médio dos *embeddings* gerados pelas $\mathtt{C}$ entradas)



CBOW: podemos considerar os dois tokens precedentes como contexto (entrada x)

Thou shalt not make a machine in the likeness of a human mind

Sliding window across running text

thou	shalt	not	make	a	machine	in	the	…

Dataset

input 1	input 2	output
thou	shalt	not

Thou shalt not make a machine in the likeness of a human mind

Sliding window across running text

thou	shalt	not	make	a	machine	in	the	…
thou	shalt	not	make	a	machine	in	the	

Dataset

input 1	input 2	output
thou	shalt	not
shalt	not	make

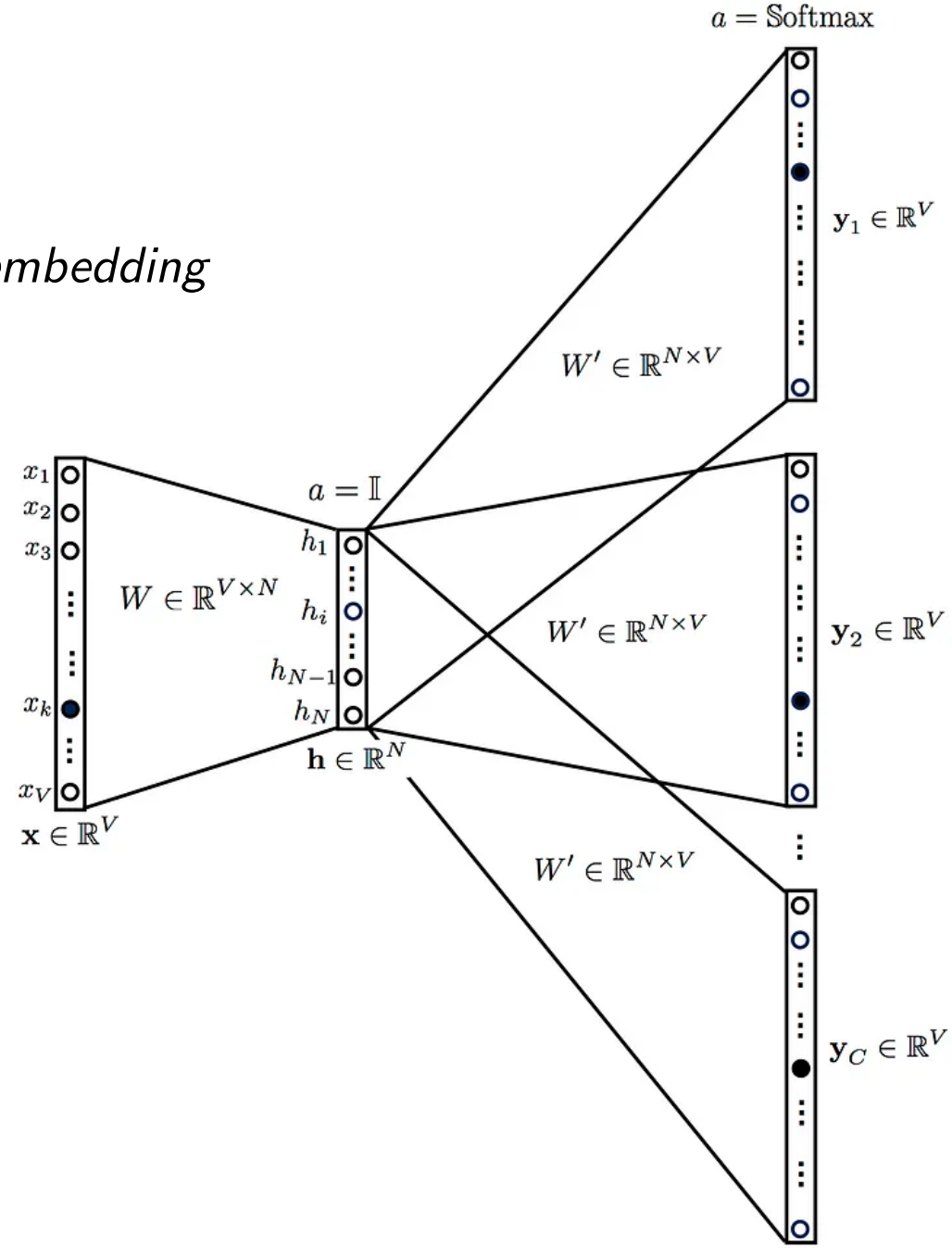
CBOW: podemos considerar *tokens* à esquerda e à direita

Jay was hit by a _____ bus in...

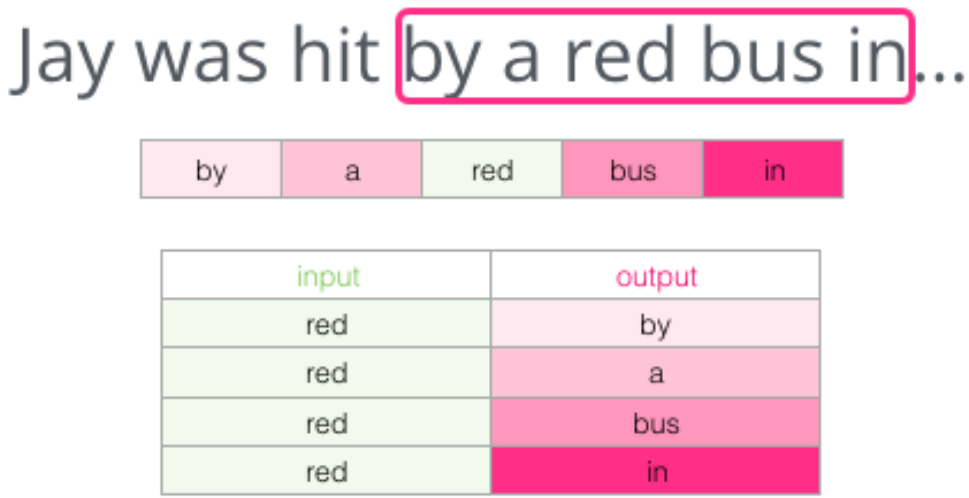
input 1	input 2	input 3	input 4	output
by	a	bus	in	red

skipgram

χ é alvo do *embedding*



No Skipgram, a entrada é o *token* para o qual queremos gerar um *embedding*
Os *tokens* a sua esquerda e direita são os contextos



Thou shalt not make a machine in the likeness of a human mind

thou	shalt	not	make	a	machine	in	the	…

input word	target word
not	thou
not	shalt
not	make
not	a

Thou shalt not make a machine in the likeness of a human mind

thou	shalt	not	make	a	machine	in	the	…
thou	shalt	not	make	a	machine	in	the	…
thou	shalt	not	make	a	machine	in	the	…
thou	shalt	not	make	a	machine	in	the	…
thou	shalt	not	make	a	machine	in	the	…

input word	target word
not	thou
not	shalt
not	make
not	a
make	shalt
make	not
make	a
make	machine
a	not
a	make
a	machine
a	in
machine	make
machine	a
machine	in
machine	the
in	a
in	machine
in	the
in	likeness

Comparação entre CBOW e Skipgram

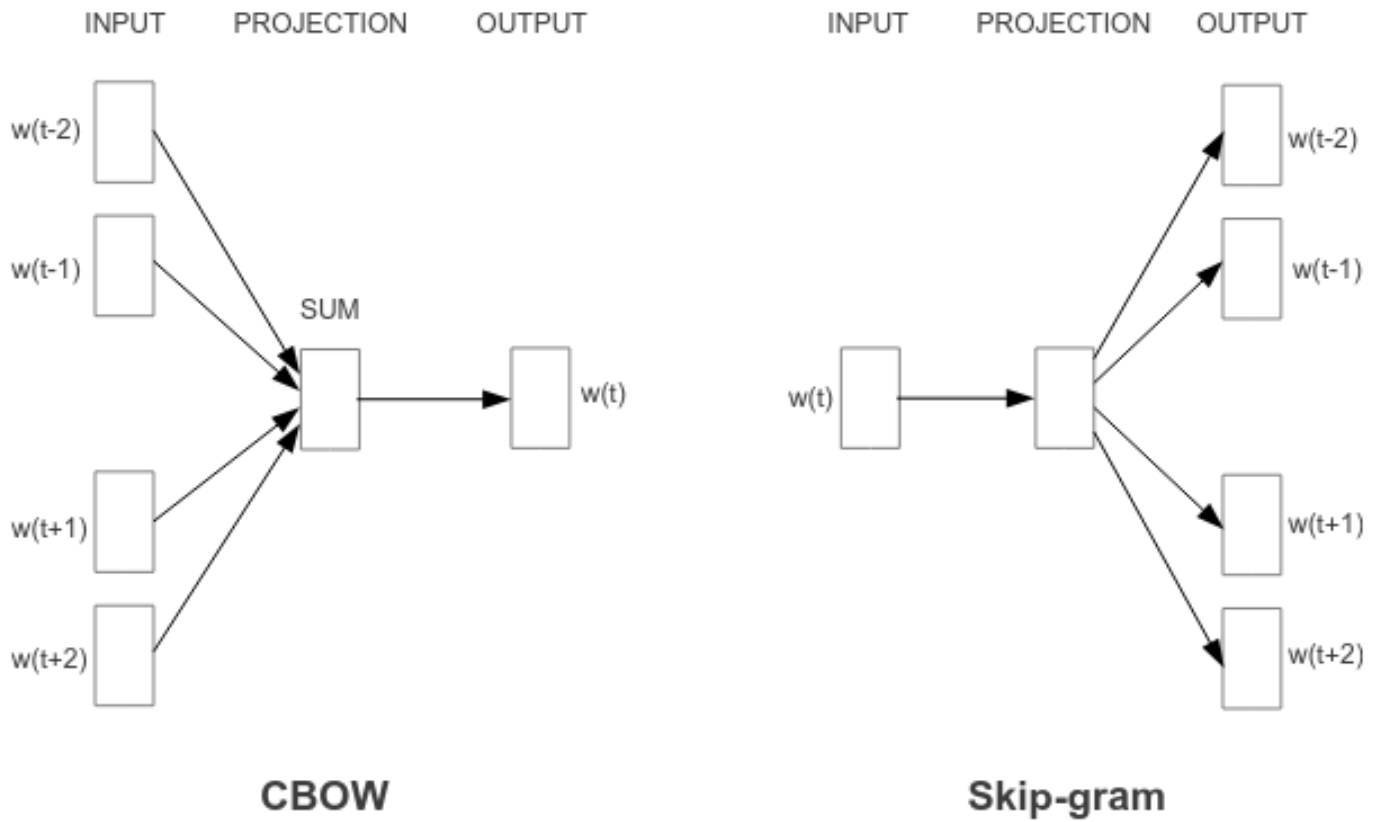


Figure 1: New model architectures. The CBOW architecture predicts the current word based on the context, and the Skip-gram predicts surrounding words given the current word.

Comparação entre CBOW e Skipgram

CBOW é computacionalmente mais eficiente

No caso de Skipgram, para cada saída há um *softmax* para V classes

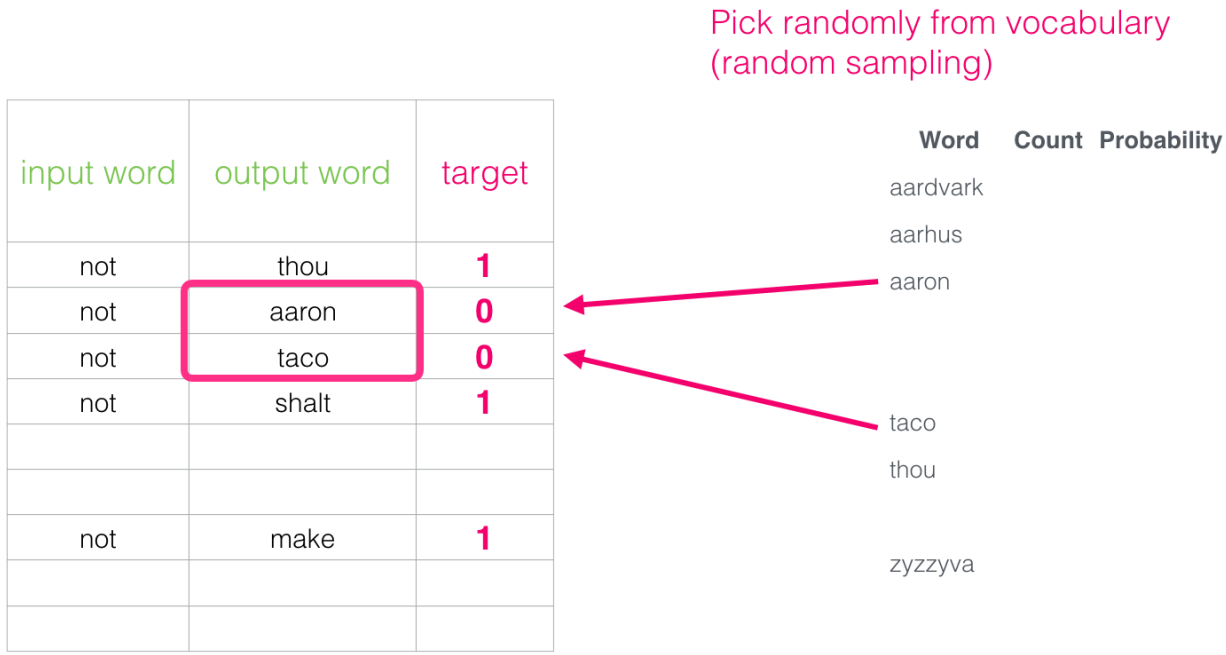
Segundo Mikolov (o autor), CBOW é melhor para palavras mais frequentes https://arxiv.org/pdf/1301.3781.pdf

Skipgram é melhor para poucos dados e palavras raras

Skipgram melhorado

Ideia: trocar o problema para, dadas duas palavras, dizer se a segunda é vizinha da primeira – regressão logística!

Mas, precisamos de exemplos negativos



skipgram with negative sampling (SGNS)

Skipgram

shalt	not	make	a	machine

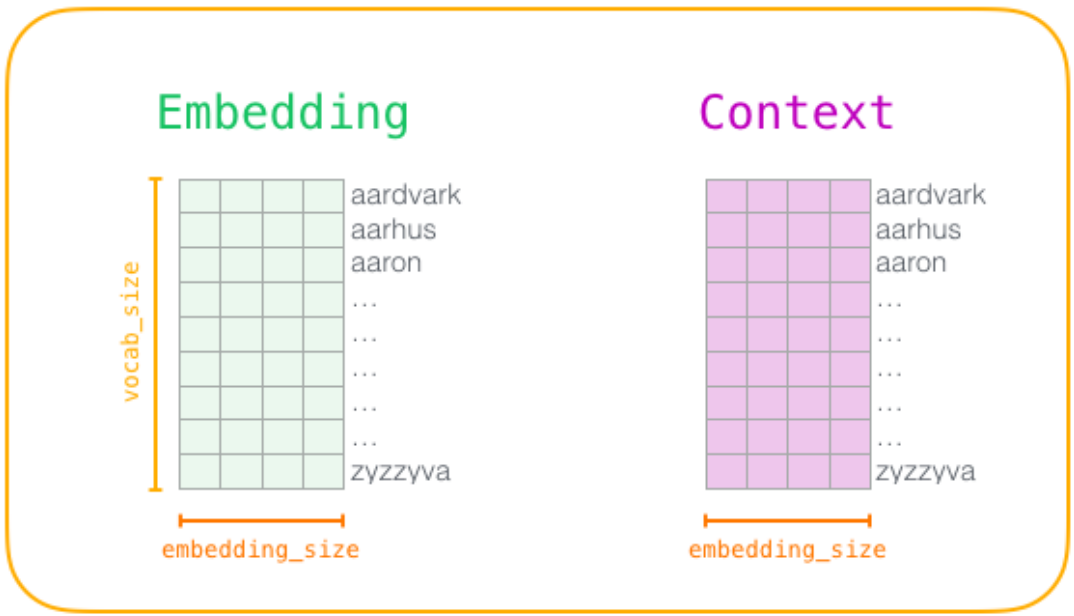
input	output
make	shalt
make	not
make	a
make	machine

Negative Sampling

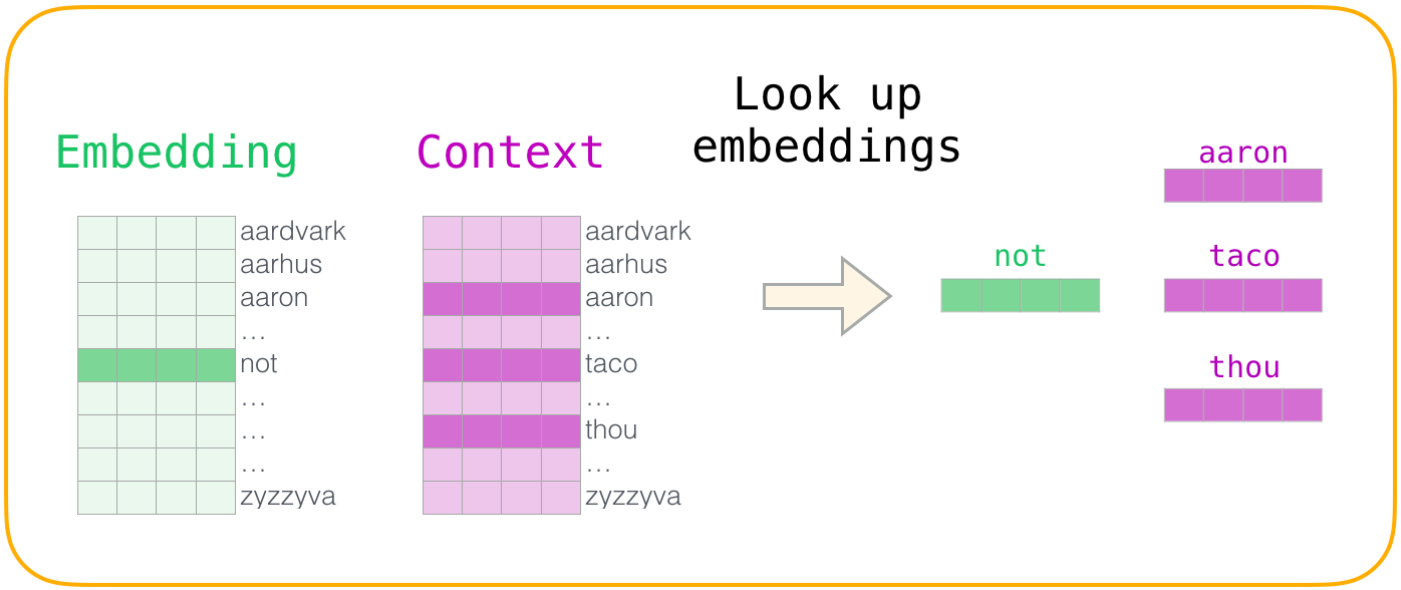
input word	output word	target
make	shalt	**1**
make	aaron	**0**
make	taco	**0**

Como o word2vec é treinado?

Definir V e o *embedding size* d
word2vec usa duas matrizes, de *embedding* e de *context*



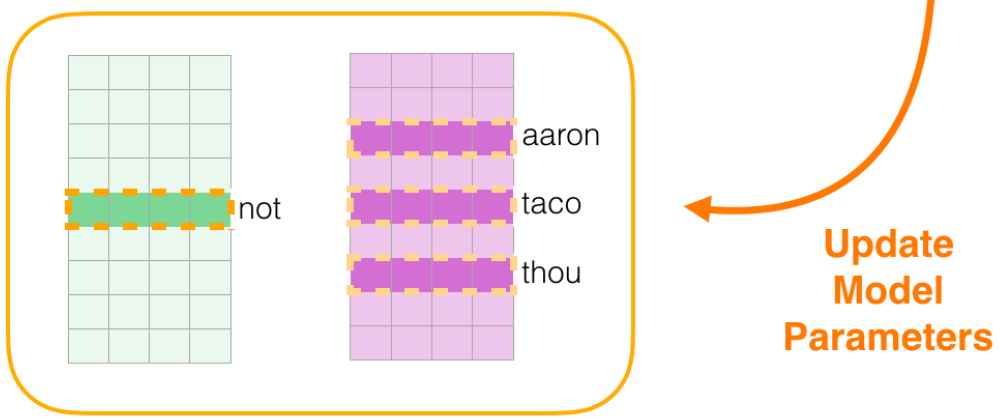
Para cada entrada e saídas positiva e negativas calcula-se *dot product* + softmax



input word	output word	target	input • output	sigmoid()	Error
not	thou	1	0.2	0.55	0.45
not	aaron	0	-1.11	0.25	-0.25
not	taco	0	0.74	0.68	-0.68

word2vec Training Process – weight update

input word	output word	target	input • output	sigmoid()	Error
not	thou	1	0.2	0.55	0.45
not	aaron	0	−1.11	0.25	−0.25
not	taco	0	0.74	0.68	−0.68



Após o treinamento, usa-se a matriz de *embedding* (e descarta-se a de *context*)

Parâmetros importantes

window size (quantas palavras antes e depois do current word considerar)

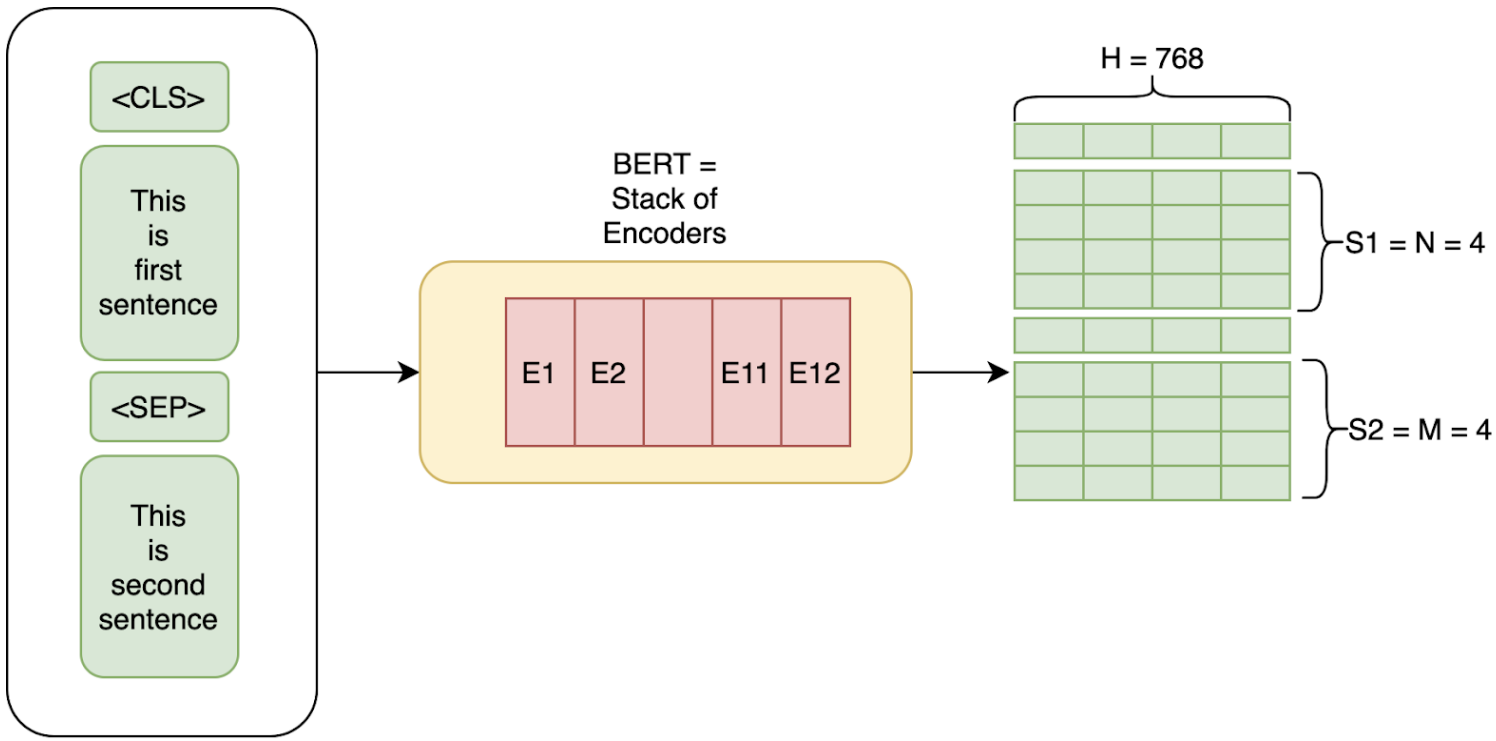
number of negative examples (quantos para cada positivo?)

BERT (*Bidirectional Encoder Representations from Transformers*)

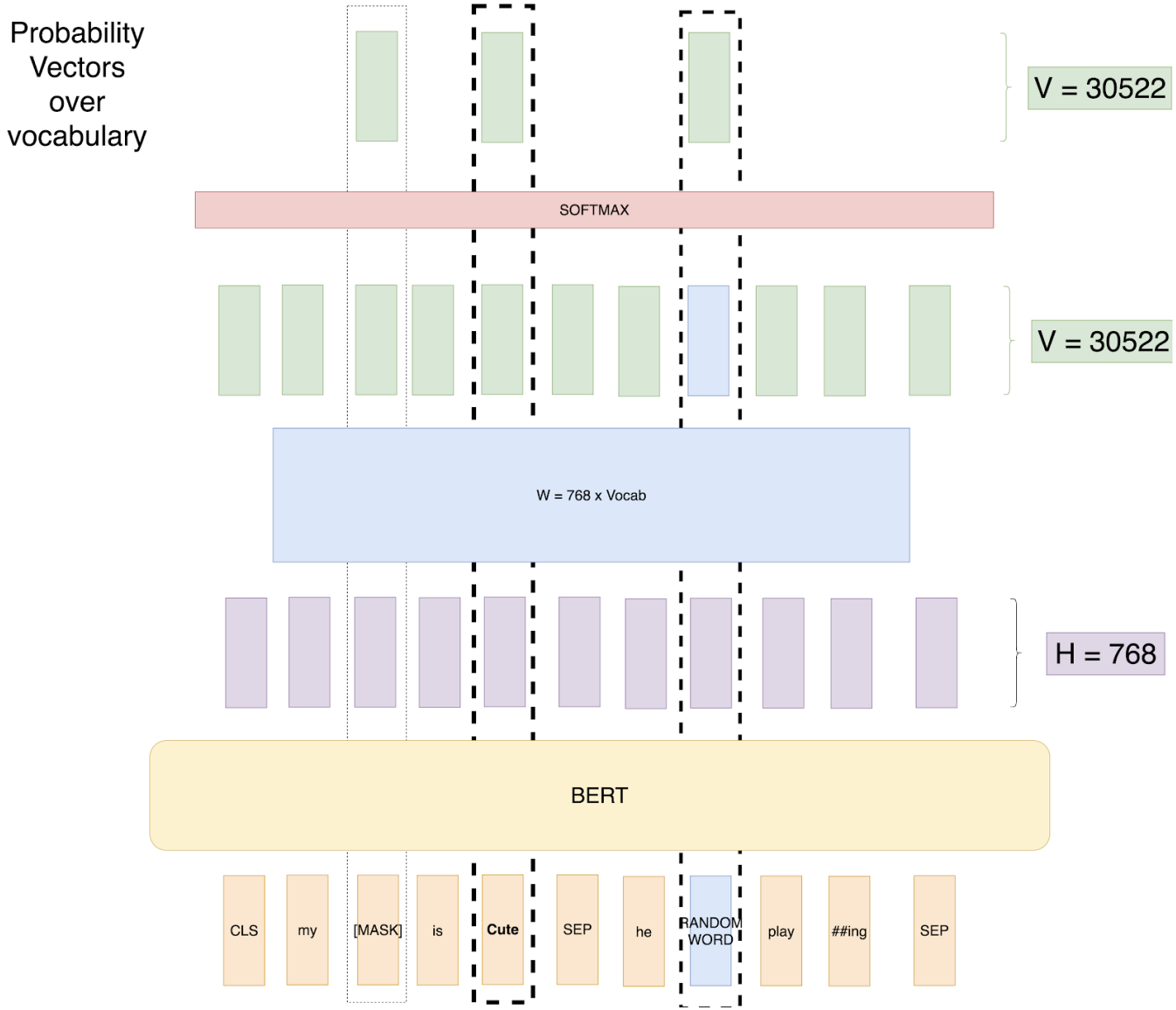
2018, https://arxiv.org/abs/1810.04805

- Usa apenas o lado *encoder* do *transformer*
- Em vez de bidirecional, seria mais preciso dizer adirecional
- BERT pode ser treinado usando as técnicas **Masked LM** (MLM) + **Next Sentence Prediction** (NSP)

Arquitetura geral do BERT



BERT training – Masked LM (MLM)



BERT training – Masked LM (MLM)

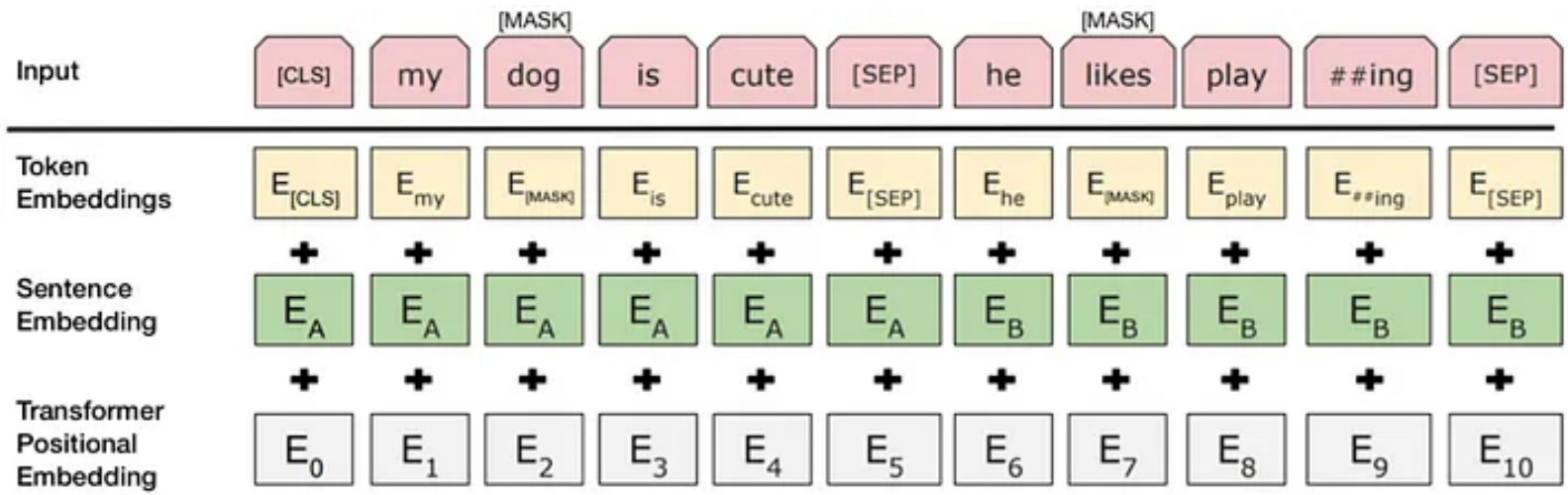
Treinamento é feito para predizer 15% dos *tokens* do *input*, escolhidos aleatoriamente.

Desses, 80% são trocados por "[MASK]", 10% por palavras aleatórias e 10% são as *tokens* originais.

Os demais *tokens* não são considerados na computação da função custo.

BERT training – Next Sentence Prediction (NSP)

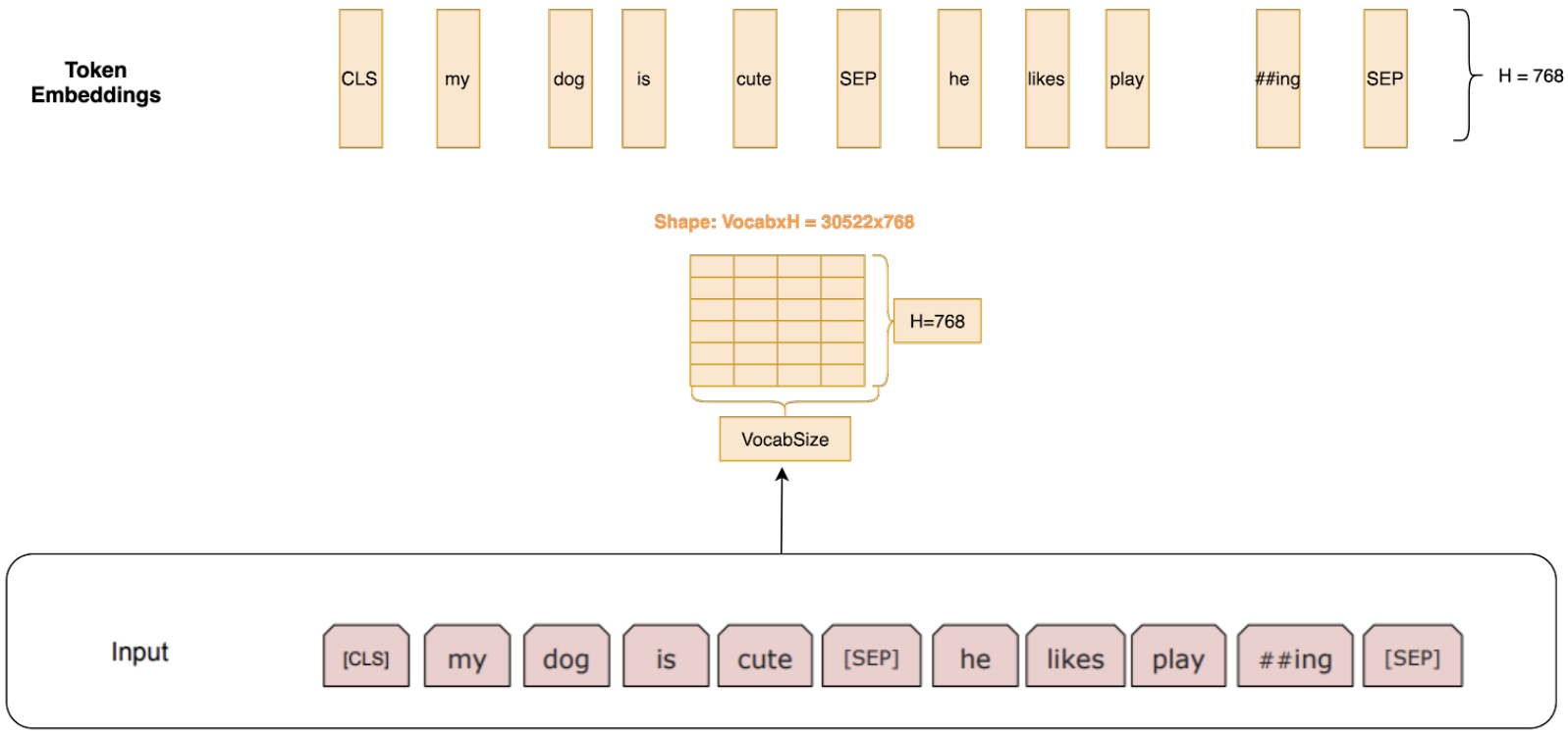
Predizer se a segunda sentença é subsequente à primeira ou não
Treina-se com 50% de casos sim e 50% de casos não
Pluga-se um softmax (binário) na saída [CLS]



Preparação da sequência para entrada nos módulos de atenção

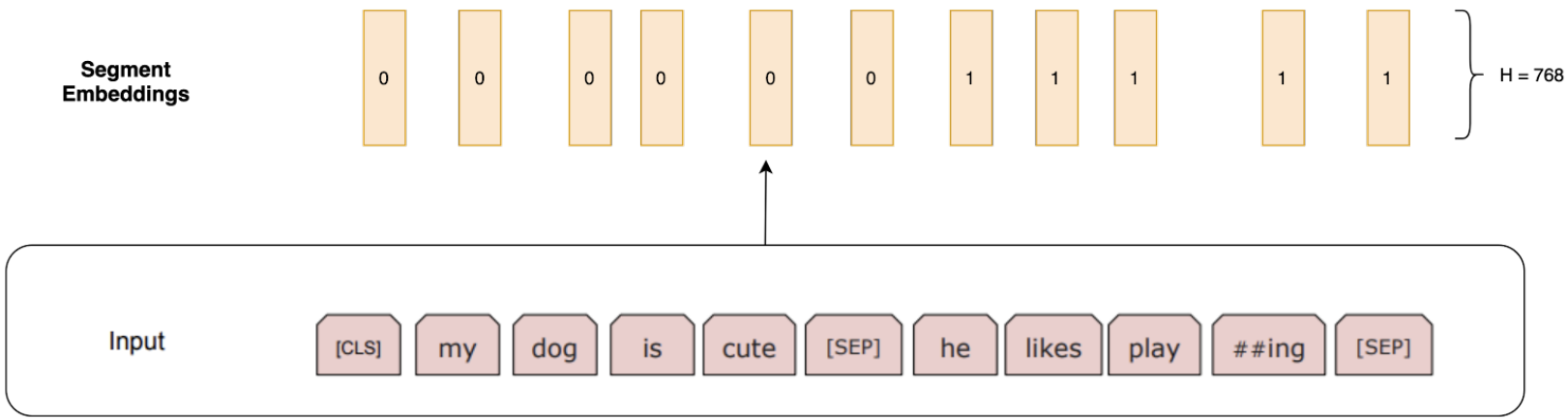
BERT – token embedding

Tokens são gerados pela técnica de tkenization `WordPiece`

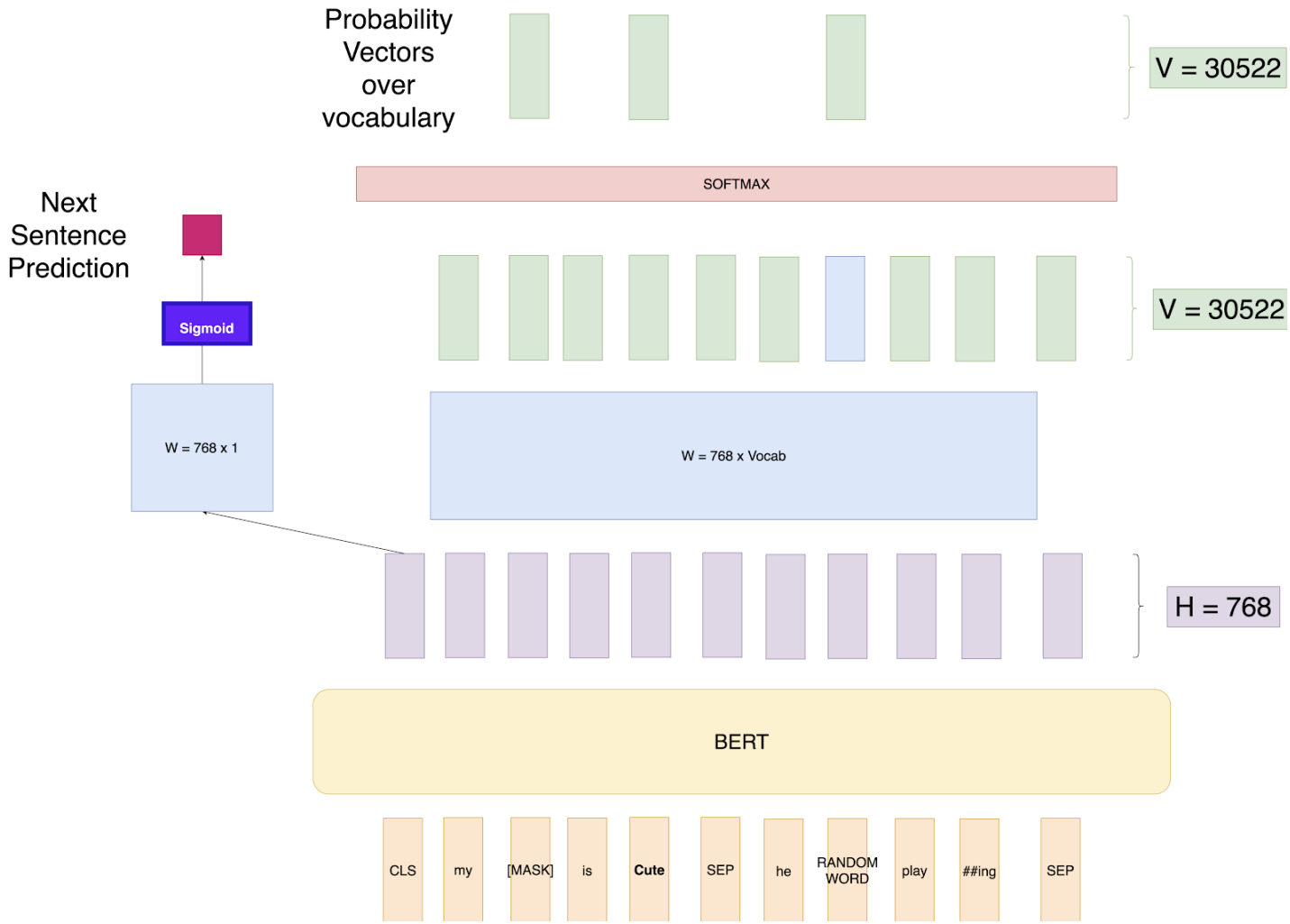


BERT – sentence embedding

Indicar se é primeira ou segunda sentença



BERT training – combinando MLM e NSP



Alguns métodos de embedding

Mesmo *embedding*, independentemente do contexto do *token*

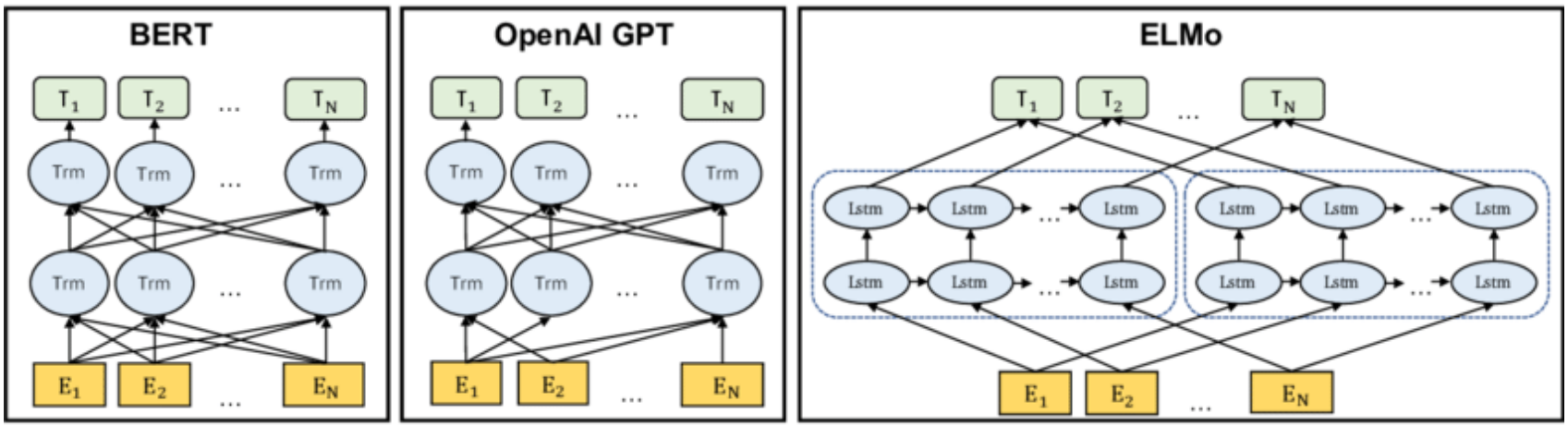
- GloVe
- word2vec

Embeddings distintos, dependendo do contexto do *token*

- ELMO (https://arxiv.org/abs/1802.05365)
- BERT (https://arxiv.org/abs/1810.04805)
- GPT

  (https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.

BERT usa o lado *encoder* do *transformer*
GPT usa o lado *decoder* do *transformer*
ELMo usa LSTM



Tokenização

Resumo de Tokenizers

https://huggingface.co/docs/transformers/tokenizer_summary

WordPiece

SentencePiece

Byte pair encoding (BPE)

https://huggingface.co/learn/nlp-course/chapter6/5?fw=pt

Byte pair encoding (BPE) – exemplo

"hug", "pug", "pun", "bun", "hugs"