



UNIVERSIDADE FEDERAL DO
RIO DE JANEIRO

SKIP-GRAM WITH NEGATIVE SAMPLES

Apresentado por **Rafael da Silva Fernandes**

Agenda

Falaremos sobre

- Breve background
- Skip-gram
 - ° Objetivo
 - ° Arquitetura
 - ° Negative-sampling
- Criação do modelo
- Utilização do modelo em redes neurais

Breve background



Breve background

Em contraste com as abordagens tradicionais de NLP que associam palavras com representações discretas, modelos de espaço vetoriais de significado incorporam cada palavra em um espaço de vetor contínuo no qual as palavras podem ser facilmente comparadas para similaridade. Eles são baseados na *hipótese distribuidora* informando que o significado de uma palavra pode ser inferido dos contextos que aparece. Após essa hipótese, as palavras são representadas por meio de seus vizinhos - cada palavra está associada a um vetor que codifica informações sobre sua co-ocorrência com outras palavras no vocabulário.

O método exato de construção de *word embeddings* difere entre os modelos, mas a maioria das abordagens pode ser categorizada como baseada em contagem ou baseada em previsão, com a última utilizando modelos neurais.



Skip-gram



Skip-gram

Objetivo

Skip-gram Objetivo

O objetivo do Skip-gram é prever os contextos de uma determinada palavra-alvo. Os contextos são vizinhos imediatos do alvo e são recuperados usando uma janela de tamanho arbitrário n – capturando n palavras à esquerda do alvo e n palavras à sua direita. Por exemplo, se $n=3$ no exemplo a seguir, Skip-gram seria treinado para prever todas as palavras destacadas em amarelo para a palavra *prickles*:

out of the gorse-bush, brushed the prickles from his nose, and began to
I have nothing against thorns and prickles so long as you can admire

skip-gram

Objetivo

Durante o treinamento o modelo é exposto a pares de dados (V_t, V_c) , onde V é o vocabulário e t, c são índices de uma palavra-alvo (*target*) e uma de suas palavras de contexto (*context*). Para o exemplo anterior, os dados de treinamento conteriam pares como $(prickles, nose)$ e $(prickles, thorns)$.

O objetivo do Skip-gram original é maximizar $P(V_c|V_t)$ — a probabilidade de V_c ser previsto como contexto de V_t para todos os pares de treinamento. Se definirmos o conjunto de todos os pares de treinamento como D , podemos formular esse objetivo maximizando a seguinte expressão:

$$\sum_{(V_t, V_c) \in D} \log P(V_c|V_t)$$

Skip-gram

Objetivo

Para calcular $P(V_c|V_t)$, precisaremos de um meio para quantificar a proximidade da palavra-alvo V_t e da palavra de contexto V_c . No Skip-gram essa proximidade é calculada usando o produto escalar entre a incorporação de entrada do destino e a incorporação de saída do contexto. A diferença entre embeddings de entrada e embeddings de saída está no fato de que os primeiros representam palavras quando servem como alvo, enquanto os últimos quando atuam como contextos de outra palavra.

A diferença entre *embeddings* de entrada e *embeddings* de saída está no fato de que os primeiros representam palavras quando servem como alvo, enquanto os últimos quando atuam como contextos de outra palavra. É importante perceber essa distinção e o fato de que no Skip-gram cada palavra está associada a duas representações separadas.

skip-gram

Objetivo

Agora, se definirmos u_c como a medida da proximidade das palavras, E como a matriz que contém os *embeddings* de entrada e O como a matriz de *embedding* de saída, obtemos:

$$u_c = E_t \cdot O_c$$

que podemos usar para calcular $P(V_c|V_t)$ usando a função softmax:

$$\sum_{(V_t, V_c) \in D} \log P(V_c|V_t) = \sum_{(V_t, V_c) \in D} \log \frac{e^{u_c}}{\sum_{k=1}^{|V|} e^{u_k}}$$

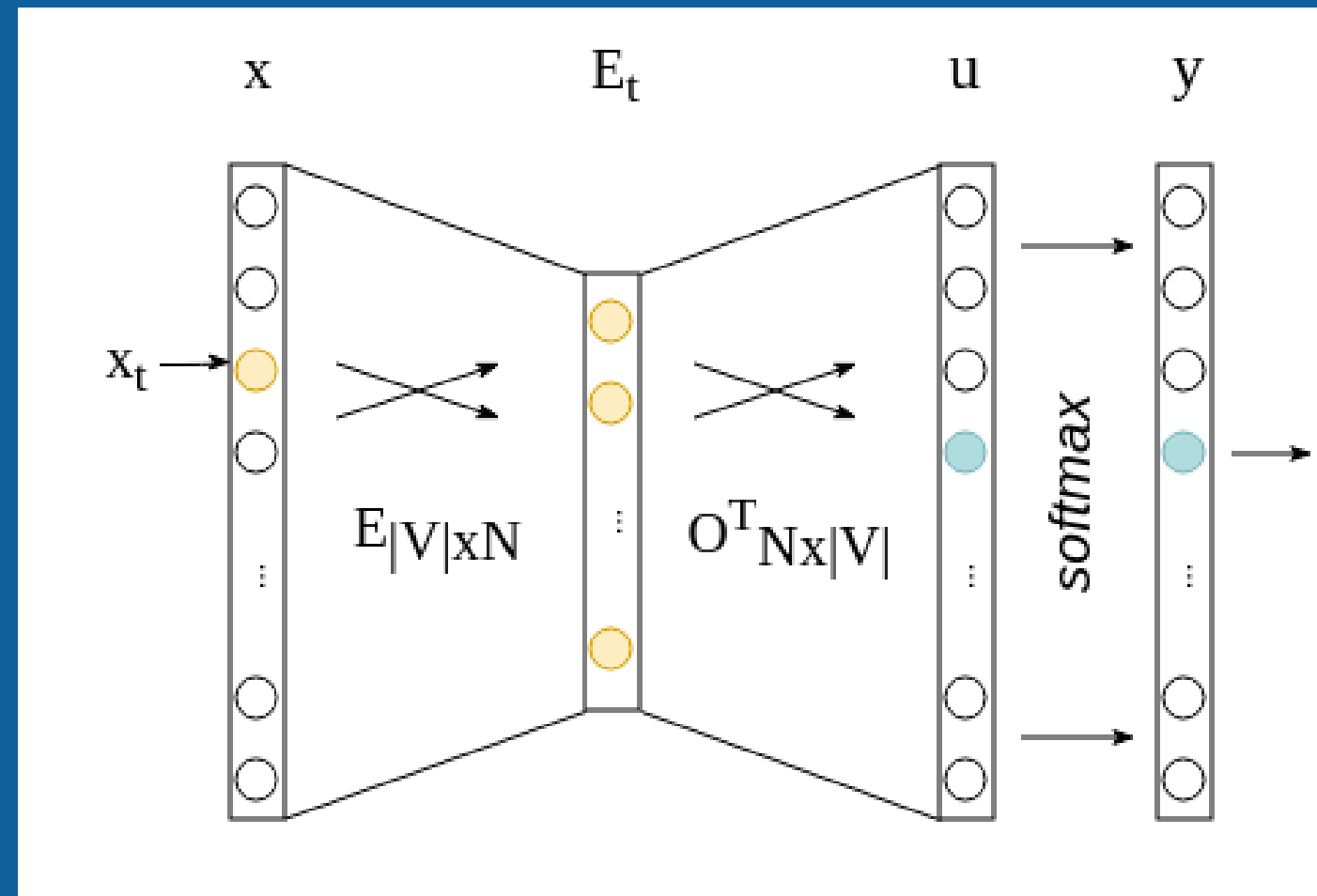


Skip-gram

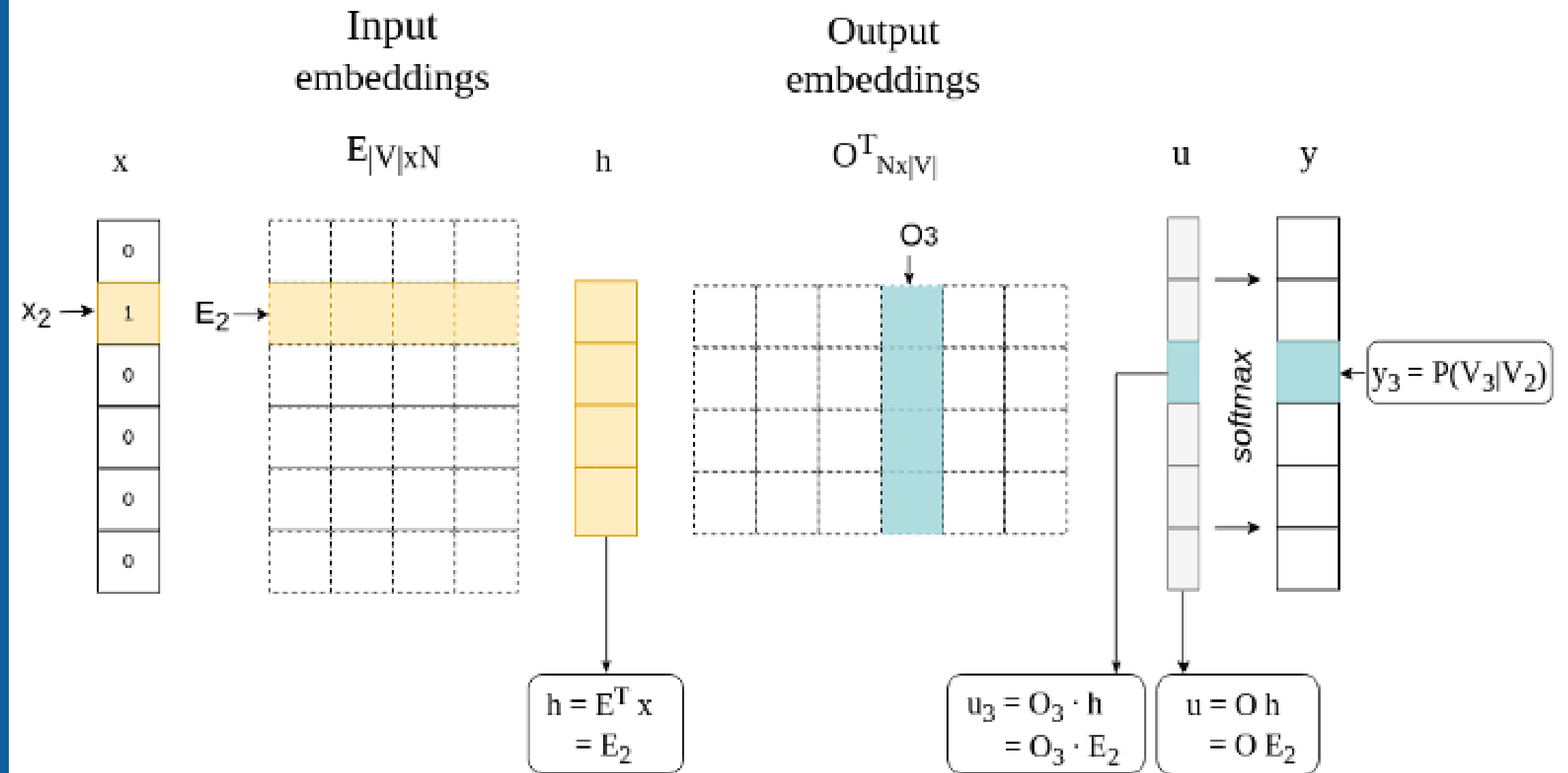
Arquitetura

Skip-gram Arquitetura

Em termos de arquitetura, Skip-gram é uma rede neural simples com apenas uma camada oculta. A entrada para a rede é uma representação vetorial *one-hot encode* de uma palavra-alvo — todas as suas dimensões são definidas como zero, exceto a dimensão correspondente à palavra-alvo. A saída é a distribuição de probabilidade sobre todas as palavras no vocabulário que define a probabilidade de uma palavra ser selecionada como o contexto da palavra de entrada



skip-gram Arquitetura





Skip-gram

Negative-sampling

Skip-gram Negative-sampling

Entretanto, há um problema com o objetivo original do softmax do Skip-gram - é altamente caro computacionalmente, pois requer a varredura através dos embeddings de saída de todas as palavras no vocabulário para calcular a soma do denominador. E normalmente esses vocabulários contêm centenas de milhares de palavras. Devido a essa ineficiência, a maioria das implementações usa um objetivo alternativo de amostragem negativa (*negative-sampling*), que reformula o problema como um conjunto de tarefas de classificação binária independentes. Relembrando nosso exemplo:

out of the gorse-bush, brushed the prickles from his nose, and began to
I have nothing against thorns and prickles so long as you can admire


skip-gram Negative-sampling

Em vez de definir a distribuição de probabilidade completa sobre as palavras, o modelo aprende a diferenciar entre os pares de treinamento corretos recuperados do corpus e os pares incorretos gerados aleatoriamente. Para cada par correto, o modelo desenha m negativos — com m sendo um hiperparâmetro. Todas as amostras negativas têm o mesmo V_t que o par de treinamento original, mas seu V_c é extraído de uma distribuição de ruído arbitrária. Com base no exemplo anterior, para o par de treinamento (*prickles*, *nose*) os incorretos podem ser (*prickles*, *worked*) ou (*prickles*, *truck*). O novo objetivo do modelo é maximizar a probabilidade das amostras corretas que vêm do corpus e minimizar a probabilidade do corpus para as amostras negativas, como (*prickles*, *truck*).

Skip-gram Negative-sampling

Vamos definir D para ser o conjunto de todos os pares corretos e D' para denotar um conjunto de todos os $|D| \times m$ pares de amostras negativas. Também definiremos $P(C = 1 | V_t, V_c)$ como a probabilidade de (V_t, V_c) ser um par correto, originário do corpus. Dada essa configuração, o objetivo de amostragem negativa é definido como maximizar:

$$\sum_{(V_t, V_c) \in D} \log P(V_c | V_t) = \sum_{(V_t, V_c) \in D} \log \frac{e^{u_c}}{\sum_{k=1}^{|V|} e^{u_k}}$$


$$\sum_{(V_t, V_c) \in D} \log P(C = 1 | V_t, V_c) - \sum_{(V_t, V_c) \in D'} \log (1 - P(C = 1 | V_t, V_c))$$

Skip-gram Negative-sampling

Como desta vez para cada amostra estamos tomando uma decisão binária, definimos $P(C = 1|V_t, V_c)$ usando a função sigmóide:

$$P(C = 1|V_t, V_c) = \sigma(u_c) = \frac{1}{1 + e^{-u_c}}$$

onde, como antes, $u_c = E_t \cdot O_c$. Agora, se colocarmos isso na equação de amostragem negativa anterior e simplificarmos um pouco, obtemos o seguinte objetivo:

$$\sum_{(V_t, V_c) \in D} \log \sigma(u_c) + \sum_{(V_t, V_c) \in D'} \log \sigma(-u_c)$$



Criação do modelo

Criação do modelo

Para treinar esse *embedding* fiz uso da implementação do Skip-gram do módulo Word2Vec da biblioteca gensim. Ele fornece os algoritmos para Skip-gram e um modelo estreitamente relacionado – *Continuous Bag-of-Words* (CBOW). Os modelos Word2Vec da Gensim são treinados em uma lista (ou algum outro iterável) de frases que foram pré-processadas e tokenizadas - divididas em palavras e pontuação separadas. Felizmente, a biblioteca NLTK fornece vários corpora tokenizados, como o CoNLL-2000 corpus, o que me permitiu ir direto para a definição do modelo.

Para fazer isso, bastou criar uma nova instância do Word2Vec. O construtor Word2Vec aceita uma ampla gama de hiperparâmetros, mas me concentrei apenas em alguns que são mais relevantes.

HIPERPARÂMETROS

- **sentences (Frases)**
O iterável sobre as sentenças tokenizadas nas quais treinaremos (as sentenças ConLL-2000).
- **window (Janela) = 5**
Determina quais palavras são consideradas contextos do alvo. Para a janela de tamanho n os contextos são definidos pela captura de n palavras à esquerda do alvo e n palavras à sua direita.
- **min_count (Contador mínimo) = 5**
Define um valor limite de frequência que precisa ser alcançado para que a palavra seja incluída no vocabulário.
- **negative (Negativo) = 15**
Define o número de amostras negativas (instâncias de pares de treinamento incorretos) que são extraídas para cada amostra boa.

Utilização do modelo em Redes Neurais





Utilização do modelo em Redes Neurais

Preparando os dados

Preparando os dados

Utilizei o *embedding* criado como recurso para o modelo de marcação de *part-of-speech* (POS) que será desenvolvido. Uma parte do discurso é uma categoria gramatical de uma palavra, como um substantivo, verbo ou adjetivo. Dada uma sequência de palavras, a tarefa é rotular cada uma delas com uma etiqueta POS adequada. Construí um modelo neural simples para classificação multiclasse. Por enquanto, ignorei o contexto da palavra que estamos marcando - a rede receberá apenas uma palavra como entrada e emitirá a distribuição de probabilidade sobre todas as tags POS possíveis.

Para lidar com as palavras sem representação no *embedding*, que aqui chamaremos de "palavras desconhecidas", criei e adicionei a palavra "UNK" (*unknown*).

Preparando os dados

```
[('Confidence', 'NN'),  
 ('in', 'IN'),  
 ('the', 'DT'),  
 ('pound', 'NN'),  
 ('is', 'VBZ'),  
 ('widely', 'RB'),  
 ('expected', 'VBN'),  
 ('to', 'TO'),  
 ('take', 'VB'),  
 ('another', 'DT')]
```

PORCENTAGEM DE PALAVRAS DESCONHECIDAS NOS DADOS

TREINO

14,3%

TESTE

14,9%



Utilização do modelo em Redes Neurais

Definindo e treinando o modelo

Definindo o modelo

O próximo passo foi definir o modelo de classificação do POS. Fiz isso usando a implementação da API Keras do TensorFlow. O modelo tomará como entrada um índice na matriz de *word embedding*, que será usada para procurar o apropriado. Ele terá uma camada oculta com a função de ativação tanh (tangente hiperbólica) e na camada final usará a ativação softmax — gerando uma distribuição de probabilidade sobre todas as tags possíveis.

Definindo o modelo

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
=====		
embedding (Embedding)	(None, 1, 100)	1517400
flatten (Flatten)	(None, 100)	0
dense (Dense)	(None, 50)	5050
activation (Activation)	(None, 50)	0
dense_1 (Dense)	(None, 44)	2244
activation_1 (Activation)	(None, 44)	0

```
=====
```

```
Total params: 1,524,694
```

```
Trainable params: 1,524,694
```

```
Non-trainable params: 0
```

Treinando o modelo

52s 31ms/ step

Tempo de execução

76,81%

Loss

79,32%

Acurácia

Avaliando o modelo com os dados de teste

2s 2ms/ step

Tempo de execução

50,61%

Loss

84,79%

Acurácia

Avaliando o modelo com os dados de teste

Erros mais comuns

Most common errors:

```
[('UNK', 5034), ('that', 136), ('have', 51), ('as', 37), ('more', 30), ('about', 18), ('executive', 18),
```




Utilização do modelo em Redes Neurais

**Criando um modelo
dependente de contexto**

Criando um modelo dependente de contexto

Agora altero o modelo construído nas etapas anteriores para receber mais de uma tag de palavras como entrada. Além do tag da palavra classificada, alimento as tags de duas palavras à esquerda e duas palavras à direita — tudo na ordem em que aparecem nos dados de treinamento.

Usei uma abordagem de janela deslizando para recuperar todos os intervalos de palavras de comprimento 5 — cada um consistindo na palavra marcada e suas palavras de contexto. Para cada intervalo, a tag correspondente será a tag da palavra do meio.

Treinando o modelo

54s 33ms/step

Tempo de execução

61,82%

Loss

83,20%

Acurácia

Avaliando o modelo com os dados de teste

3s 2ms/ step

Tempo de execução

31,62%

Loss

90,55%

Acurácia

Avaliando o modelo com os dados de teste

Erros mais comuns

Most common errors:

```
[('UNK', 2978), ('is', 67), ('out', 30), ('so', 19), ('old', 14), ('content', 10),
```

```
('boot', 9), ('It', 8), ('C.', 8), ('its', 8)]
```



Utilização do modelo em Redes Neurais

Comparando os modelos

Modelo sem contexto

Tempo de execução:
2s 2ms/ step

Loss:
50,61%

Acurácia:
84,79%

Modelo dependente de contexto

Tempo de execução:
3s 2ms/ step

Loss:
31,62%

Acurácia:
90,55%

Erros mais comuns

Modelo sem contexto

Most common errors:

```
[('UNK', 5034), ('that', 136), ('have', 51), ('as', 37), ('more', 30), ('about', 18), ('executive', 18),
```

Modelo dependente de contexto

Most common errors:

```
[('UNK', 2978), ('is', 67), ('out', 30), ('so', 19), ('old', 14), ('content', 10),
```


Obrigado!

CÓDIGO DISPONÍVEL EM:

<https://github.com/RafaelxFernandes/Word-Embeddings/blob/main/Skip-gram%20NS.ipynb>

CONTATO:

rafaelfernandes@ic.ufrj.br