DHENY R. FERNANDES

DATA ANALYTICS

DADOS GERADOS POR HUMANOS

AULA 04

SUMÁRIO

O QUE VEM POR AÍ?

Nesta aula, discutiremos um tipo de representação vetorial que lida com os problemas das representações que vimos até agora. Este tipo de representação consegue capturar relações semânticas, bem como lidar melhor com o contexto de uma palavra em uma frase. Assim, uma mesma palavra, usada em contextos diferentes, consegue ser representada pelo mesmo vetor.

HANDS ON

Aprenderemos sobre embeddings, como criá-los e usá-los em algoritmos de Machine Learning. Acompanhe pelo Jupyter Notebook a execução dos códigos.

SAIBA MAIS

Introdução

Nas representações que vimos até aqui, descobrimos várias fraquezas que fazem com que esses modelos se tornem impraticáveis de serem usados em grandes conjuntos de dados. Podemos elencar os principais problemas que discutimos:

1. Elas são representações discretas, ou seja, tratam as unidades de linguagem como unidades atômicas, o que as impede de capturar relações entre palavras.
2. Os vetores de características são representações esparsas e de alta dimensionalidade. Além de prejudicar a eficiência computacional, conforme o tamanho do vocabulário aumenta, a dimensionalidade também aumenta, com a maioria dos valores sendo zero para qualquer vetor, o que prejudica o aprendizado.
3. Não podem lidar com palavras que estão fora do vocabulário.

Para resolver esse problema, métodos para aprender uma representação de baixa dimensão foram sugeridos.

Para entendermos o conceito dessas representações, também chamadas de Representações Distribuídas, precisamos entender alguns pontos antes.

O primeiro deles é o de **Similaridade Distribucional**, que representa a ideia de que o significado de uma palavra pode ser entendido a partir do contexto em que aparece. Isto é conhecido também como conotação, ou seja, o significado é definido pelo contexto, diferente de denotação, que é o significado literal de uma palavra.

O segundo ponto é a **Hipótese Distribucional**, segundo o qual palavras que aparecem em contextos similares possuem significados similares. Assim, se palavras podem ser representadas por vetores, então duas palavras que aparecem em contextos similares devem possuir vetores similares.

O terceiro refere-se à **Representação Distribucional** e são os vetores de representação que vimos até agora, caracterizados pela alta dimensão e grande esparsidade.

Por fim, todos esses conceitos nos levam à **Representação Distribuída**, que são vetores compactos (baixa dimensão) e densos (não-esparsos). Daqui, surgiu o conceito de Word Embeddings.

Motivação

A ideia por de trás de Word Embeddings é a possibilidade de representar uma palavra usando um vetor compacto e denso que preserve sua conotação, ou seja, seu significa inferido a partir de um contexto. Mas como eu represento o significado de uma palavra? Aliás, qual é a definição para significado?

Segundo o Dicionário Online de Português, essas são as definições da palavra “significado”:

1. Definição atribuída a um termo, palavra, frase ou texto; acepção.
2. Aquilo que alguma coisa quer dizer; sentido.
3. Uma ideia que é representada por uma palavra ou frase.

Logo, percebemos que as representações usadas até aqui perdiam exatamente esse tipo de informação, ou seja:

1. Havia perda de nuances. Exemplo: sinônimos - apto, bom, expert.
2. Novas palavras. Exemplo: ninja.
3. Como não se leva em consideração o contexto, é difícil calcular a similaridade entre palavras.

Dessa maneira, precisamos de uma representação que consiga capturar tais relações entre diferentes palavras, levando em consideração o contexto. Para resolver esse problema, usaremos representações baseadas em similaridade distribucional.

Mas como capturar a relação de uma palavra com seus vizinhos e representar isso através de um vetor denso? Vamos fazer uma analogia:

Nessa analogia, estamos contabilizando o quanto uma pessoa é tímida, numa escala de 0 a 100. Podemos representar alguém que tenha dado a nota 38 da seguinte maneira:

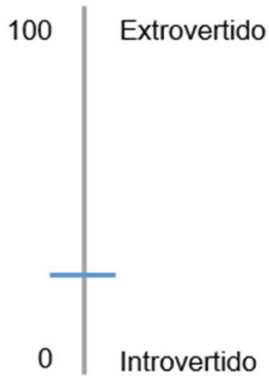

Figura 1 – Gráfico 1

Fonte: elaborado pelo autor (2023)

Normalizando o range para o intervalo (-1,1), temos o seguinte gráfico:

Figura 2 – Gráfico 2

Fonte: elaborado pelo autor (2023)

Podemos adicionar mais um traço de personalidade.

Figura 3 – Gráfico 3

Fonte: elaborado pelo autor (2023)

Com esses dois traços de personalidade, podemos traçar um vetor que represente uma determinada pessoa.

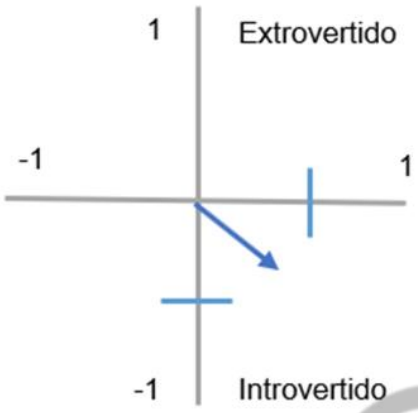

Figura 4 – Gráfico 4

Fonte: elaborado pelo autor (2023)

Podemos, assim, comparar diversas pessoas.

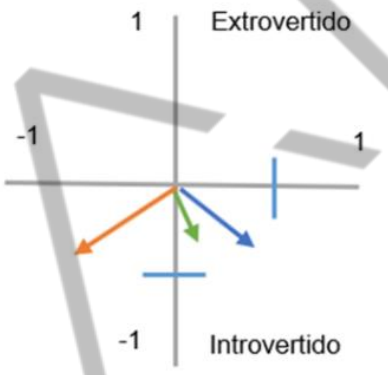

Figura 5 – Gráfico 5

Fonte: elaborado pelo autor (2023)

Esses vetores podem ser representados em números, cada qual com a informação associada ao traço de personalidade em questão. A “pessoa azul” pode ser representada por esse vetor: [-0.4, 0.6]. Já a “pessoa laranja” pode ser representada por esse vetor: [-0.3, -0.7].

Com esses vetores numéricos, é possível usar uma métrica de distância, distância do cosseno ou euclidiana, por exemplo, e verificar o quão similar dois vetores são. Essa é a ideia por trás dos Words Embeddings.

A técnica que deu início aos Words Embeddings foi divulgada em um paper de 2013, do Google, cuja referência está no fim do texto. Essa técnica recebeu o nome de “Word2Vec” e entenderemos seu funcionamento em seguida.

Word2Vec

O que significa dizer que uma representação textual deveria capturar a similaridade distribucional entre palavras? Vamos analisar alguns exemplos: se eu fornecer a palavra “Brasil”, outras palavras com similaridade distribucional a essa poderiam ser outros países (“Chile”, “Uruguai”, etc.). Se eu forneço a palavra “Bela”, poderia em pensar em sinônimos ou antônimos como palavras com similaridade distribucional. Ou seja, o que estamos tentando capturar são palavras que possuem alta probabilidade de aparecerem num mesmo contexto.

Ao aprender tais relações semânticas, o Word2Vec garante que a representação aprendida possui baixa dimensionalidade (palavras são representadas por vetores de 50-1000 dimensões) e são densas (a maioria dos valores dos vetores são diferente de zero). Tais representações tornam as tarefas de modelos de Machine Learning mais eficiente.

Antes de entrarmos nos detalhes de como o Word2Vec consegue capturar tais relações, vamos construir uma intuição de como ele funciona. Dado um corpus de texto, o objetivo é aprender embeddings de cada palavra no corpus, de modo que o vetor da palavra no espaço de embeddings melhor capture o significa da palavra. Para isso, Word2Vec usa similaridade distribucional e hipótese distribucional, ou seja, extrai o significado de uma palavra a partir do seu contexto. Assim, se duas palavras geralmente ocorrem em contextos similares, é altamente provável que seus significados também sejam similares.

Dessa maneira, o Word2Vec projeta o significado das palavras em um espaço vetorial, onde palavras com significados similares tendem a ser agrupadas juntas e palavras com significados muito diferentes estão longe umas das outras.

Conceitualmente, o que queremos saber é, dada uma palavra w e as palavras que aparecem em seu contexto C, como encontramos um vetor que melhor representa o significado da palavra? Bom, para cada palavra w no corpus, iniciamos um vetor v_w com valores aleatórios. O modelo Word2Vec refina os valores em v_w, predizendo v_w dados os vetores de palavras no contexto C. Isto é feito através de uma rede neural de duas camadas, que vamos explorar como construir futuramente. Antes, porém, veremos modelos pré-treinados.

Word Embeddings Pré-treinados

Treinar seu próprio embedding é uma tarefa muito custosa, tanto em termos de tempo quanto computacionais. Entretanto, para muitos cenários, não é necessário treiná-lo, pois muitos embeddings pré-treinados serão suficientes.

Um embedding pré-treinado nada mais é que o processo de treinar um embedding num grande corpus de dados e disponibilizar os vetores na internet. Assim, eles podem ser baixados e usados em várias aplicações. Tais embeddings podem ser entendidos como uma grande coleção de pares de chave-valor, em que as chaves representam as palavras no vocabulário e os valores seus vetores correspondentes.

Nas referências, é apresentado um site que compila vários embeddings pré-treinados para utilização. Agora, vamos entender como construímos nosso próprio embedding usando Word2Vec.

Construindo um Embedding usando Word2Vec

Baseado no artigo que originou o Word2Vec, temos duas variantes arquiteturais para treinar nosso próprio embedding: Continuous Bag of words (CBOW) e SkipGram. Ambas possuem similaridades em muitos aspectos. Vamos começar pelo CBOW.

No CBOW, o objetivo é construir um modelo que corretamente prediga a palavra central, dadas as palavras que estão num contexto dela. Em uma sentença de m palavras, o modelo atribui uma probabilidade $P(w_1, w_2, \ldots, w_n)$ para ela toda. Assim, o objetivo de um modelo de linguagem é atribuir probabilidades, de tal maneira que forneça alta probabilidade para “boas” sentenças e baixa probabilidade para “más” sentenças. Uma “boa” sentença seria a frase: “o gato pulou sobre o cachorro”. Já uma “má” sentença seria: “sobre pulou o gato o cachorro”.

Como dito, o CBOW tenta aprender um modelo de linguagem ao predizer a palavra central a partir das palavras em seu contexto. Para exemplificar esse conceito, usaremos um simples exemplo como se fosse nosso corpus: “the quick brown fox jumps over the lazy frog”.

Se considerarmos a palavra “jumps” como a palavra central, então seu contexto será formado pelas palavras em sua vizinhança. Tomando uma janela de tamanho 2

para definir o contexto de uma palavra, o contexto de “jumps” seriam as palavras “brown”, “fox”, “over” e “the”.

O CBOW faz isso para todas as palavras presente no corpus, ou seja, toma cada palavra no corpus como a palavra a ser predita e tenta predizê-la a partir das palavras em seu contexto. A imagem abaixo nos apresenta como seria o processo de treino considerando nosso corpus de exemplo:

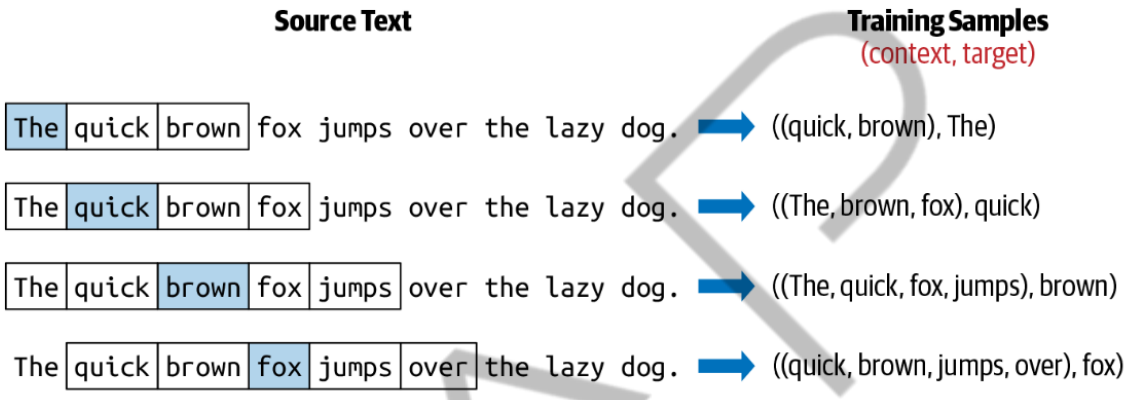

Figura 6 – CBOW

Fonte: Practical Natural Language Processing (2020)

Na prática, construímos uma rede neural que irá aprender a relação de uma palavra com as demais no corpus. Vejamos como construir esse modelo.

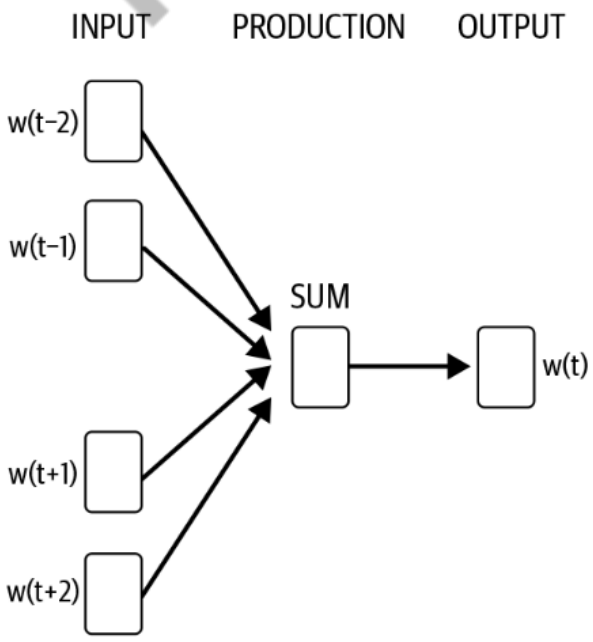

Figura 7 – Arquitetura CBOW

Fonte: Word2Vec (2016)

Já no SkipGram, entrando em mais detalhes, o objetivo é predizer as palavras em um contexto de uma palavra central. Usando o mesmo corpus do exemplo dado no CBOW, esse seria o processo de treino para o skipgram:

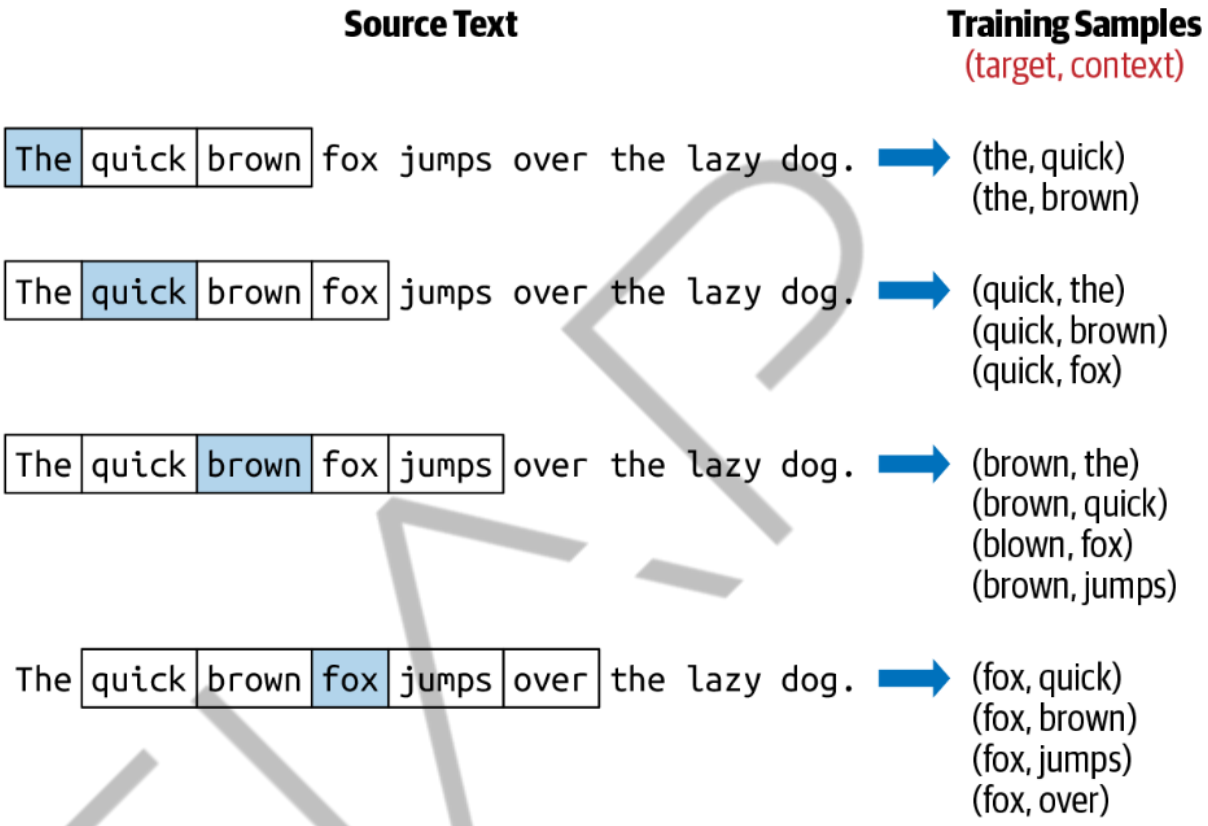

Figura 8 – SkipGram

Fonte: Practical Natural Language Processing (2020)

A rede neural usada pelo skipgram é representada pela figura 9.

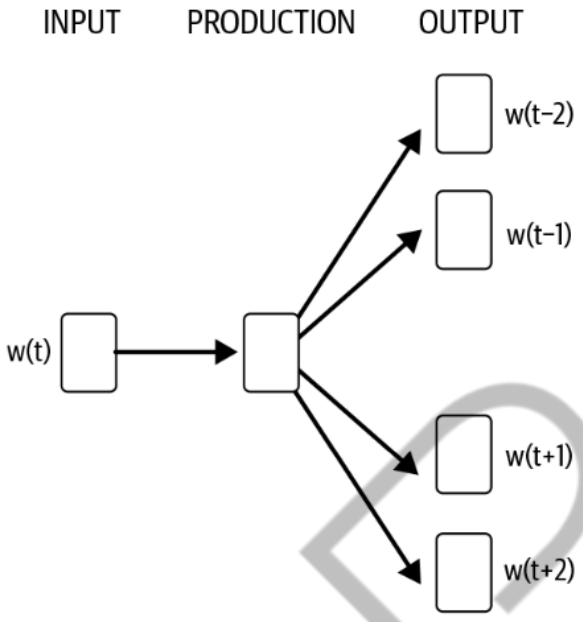

Figura 9 – Arquitetura SkipGram
Fonte: Word2Vec (2016)

Na prática, o que estamos construindo é uma rede neural com uma camada de input, uma camada intermediária e uma de saída. Isso vale tanto para o CBOW quanto para o SkipGram, mas para efeitos didáticos, tomaremos o SkipGram como padrão a partir de agora. Entretanto, o objetivo final não é usar essa rede treinada, mas usar apenas os pesos aprendidos na camada intermediária da rede. Vamos entender isso em detalhes.

Como exemplo, considere que temos um vocabulário com 10.000 palavras únicas. A camada de entrada da nossa rede será um vetor one-hot-encoding de 10.000 posições (uma para cada palavra do dicionário). Supondo a palavra “ants”, será colocado 1 na posição do vetor em que essa palavra aparecer e 0 para as demais posições.

A camada de saída da rede é também um vetor de 10.000 posições contendo, para cada palavra no vocabulário, a probabilidade de que uma palavra próxima selecionada aleatoriamente seja aquela palavra do vocabulário. Dessa forma, podemos representar a rede neural da seguinte maneira:

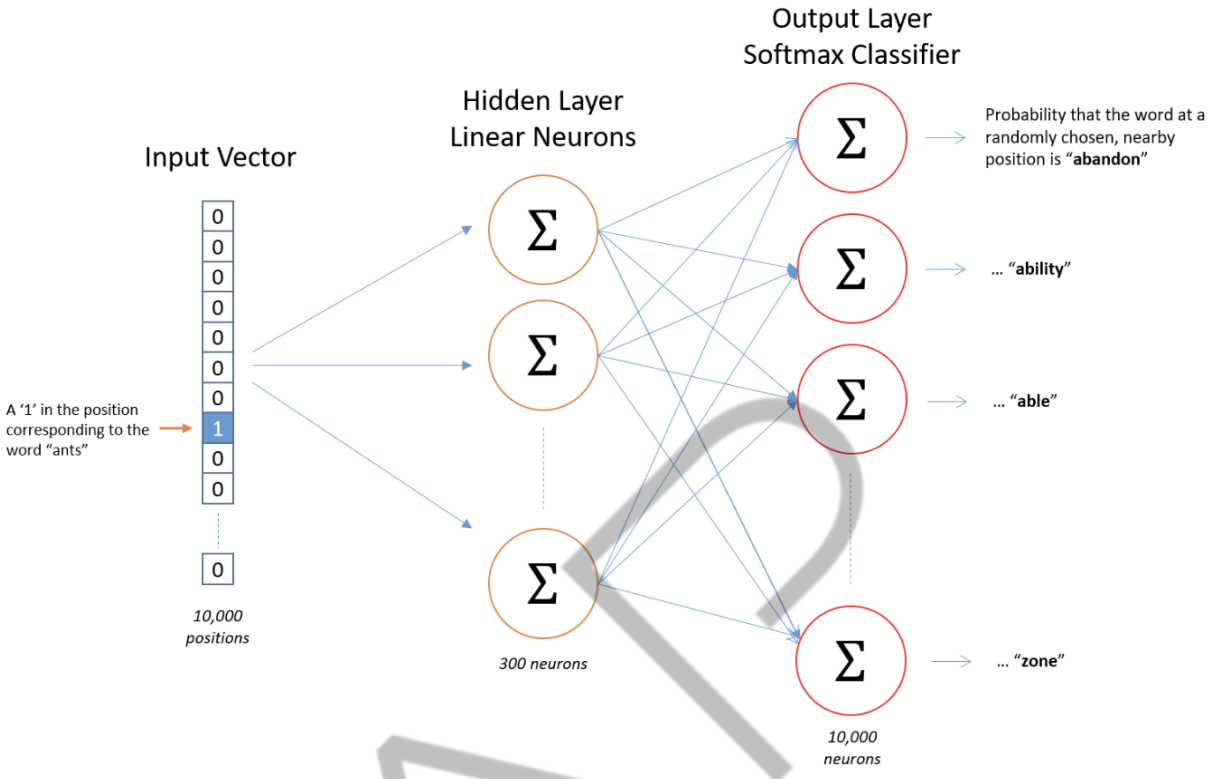

Figura 10 – Arquitetura da rede neural

Fonte: Word2Vec (2016)

Quando falamos de Word Embeddings, os vetores são densos e de baixa dimensionalidade. Na prática, ser de baixa dimensionalidade possui relação direta com a quantidade de neurônios na camada intermediária e esse valor é um hiperparâmetro que determinamos ao realizar o treinamento da rede neural. Para o nosso exemplo, usaremos um vetor de 300 posições. Dessa maneira, a camada oculta será representada por uma matriz de pesos com 10000 linhas (tamanho do dicionário) e 300 colunas (uma para cada neurônio oculto). O objetivo final, então, é aprender os pesos dessa camada oculta, que serão os vetores de embendings das palavras no vocabulário.

Você pode ter notado que nossa rede possui uma quantidade muito grande de pesos. No exemplo dado, há 3 milhões de pesos entre a camada de entrada e a camada oculta e mais 3 milhões de pesos entre a camada oculta e a camada de saída. Realizar um treino com essas especificações em um dataset grande é proibitivo. Pensando nisso, os autores propuseram uma maneira eficiente de realizar o treino minimizando o processamento. Essa técnica foi denominada “negative sampling”.

Negative sampling, então, endereça esse problema fazendo com que cada amostra de treino modifique apenas uma pequena porcentagem dos pesos. Vamos entender seu funcionamento. Quando na etapa de treinamento eu tenho um par (“fox”, “quick”) a saída correta é um one-hot vector, ou seja, para o neurônio de saída que corresponde a “quick” retornar 1 e todos os demais retornarem 0.

Com o negative sampling, vamos selecionar aleatoriamente um pequeno número (5) de palavras “negativas” para atualizar os pesos (“negativa” significa uma palavra para a qual queremos que a rede produza um 0). Ainda atualizaremos os pesos para a palavra “positiva” (a palavra “quick”, em nosso exemplo).

Assim, vamos ajustar os pesos da palavra positiva mais os pesos de 5 palavras negativas, o que corresponde a um total de 6 neurônios e 1800 pesos no total. Isso é apenas 0,06% dos 3M de pesos da camada de saída. Mas como fazemos para selecionar amostras negativas?

As amostras negativas (negative samples), ou seja, as 5 palavras que vou treinar para retornar um 0, são selecionadas usando uma distribuição unigrama, em que palavras mais frequentes são mais prováveis de serem selecionadas como amostras negativas.

Supondo um vocabulário de tamanho n, a probabilidade de selecionar aleatoriamente uma palavra é dada pela seguinte equação:

$$p(w_i) = \frac{f(w_i)}{\sum_{j=0}^{n} f(w_j)}$$

Os especialistas afirmaram que testaram diversas variações dessa equação e aquela que obteve o melhor resultado foi a que elevou a contagem de palavras à potência de $^3/_4$. Assim, a equação ficaria da seguinte maneira:

$$p(w_i) = \frac{f(w_i)^{3/4}}{\sum_{j=0}^{n} f(w_j)^{3/4}}$$

O motivo é que essa maneira possui a tendência de aumentar a probabilidade de palavras menos frequentes e diminuir a probabilidade de palavras mais frequentes.

O QUE VOCÊ VIU NESTA AULA?

Nesta aula, vimos como podemos usar representações textuais melhores, que conseguem capturar de maneira mais eficiente as relações semânticas entre as palavras em um texto. Para isso, usamos uma técnica chamada Word2Vec, que cria vetores densos e compactos usados para representar as palavras. Também vimos como criar nossos embeddings e usá-los para treinar um modelo de Machine Learning.

O que achou do conteúdo? Conte-nos no Discord! Estamos disponíveis na comunidade para fazer networking, tirar dúvidas, enviar avisos e muito mais. Participe!

REFERÊNCIAS

BIRD, S; et al. **Natural Language Processing with Python**. Sebastopol: O'Reilly Media, 2009.

McCormick, C. **Word2Vec Tutorial - The Skip-Gram Model**. 2016. Disponível em: <http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/>. Acesso em: 04 out. 2023.

MIKOLOV, T; et al. **Efficient Estimation of Word Representations in Vector Space**. 2013. Disponível em: <https://arxiv.org/pdf/1301.3781.pdf>. Acesso em: 04 out. 2023.

Pretrained Embeddings. **Wikipedia2Vec**. Disponível em: <https://wikipedia2vec.github.io/wikipedia2vec/pretrained/>. Acesso em: 04 out. 2023.

VAJJALA, S; et al. **Practical Natural Languagem Processing: A Comprehensive Guide to Building Real-World NLP Systems**. Sebastopol: O'Reilly Media, 2020.

PALAVRAS-CHAVE

Palavras-chave: Embbeding, Word2vec, Representação vetorial.