Relatório 6 - Prática: Embedding (II)

Vitor Eduardo de Lima Kenor

Descrição da atividade

Nesse card somos apresentados a dois vídeos, um que apresenta o que são word embeddings, e o outro ensinado a criar um LLM do zero.

Começando pelo primeiro vídeo, já no começo antes de ir para a explicação sobre embeddings é retomado o que é a Linguagem natural. E a linguagem natural nada mais é do que a linguagem que usamos no dia a dia para expressar opiniões e conversar com as pessoas. A ideia dos word embeddings é capturar os significados presentes nessa linguagem. Antigamente a representação das palavras se dava por um one hot vector, que basicamente se trata de um vetor de zeros e onde estiver posicionado o um irá representar uma palavra, o problema dessa abordagem é que não existia uma forma de achar semelhanças entre palavras. Já o word embeddings se trata de uma maneira de representar palavras em um espaço vetorial de N dimensões, e estas representações podem ser utilizadas para encontrar analogias entre palavras. No final é apresentado três frameworks famosos para gerar/aprender word embeddings, foram citados o Word2vec, Glove e Bert.

No segundo vídeo, iniciamos aprendendo uma técnica antiga porém didática de como representar as palavras de um jeito que o computador entenda.

```
[1] texto = 'hoje estou fazendo um card do fastcamp de LLM!'
    caracteres = sorted(list(set(texto)))
    print(caracteres)
    print('Tamanho do seu vocabulário >> ', len(caracteres))
```

```
[' ', '!', 'L', 'M', 'a', 'c', 'd', 'e', 'f', 'h', 'j', 'm', 'n', 'o', 'p', 'r', 's', 't', 'u', 'z']
Tamanho do seu vocabulário >>  20
```

Nesse primeiro trecho de código, estamos criando uma lista que terá todos os caracteres que foram utilizados no texto.

```
letraParaIndex = { lt:i for i,lt in enumerate(caracteres) }
letraParaIndex
```

```
{' ': 0,
 '!': 1,
 'L': 2,
 'M': 3,
 'a': 4,
 'c': 5,
 'd': 6,
 'e': 7,
 'f': 8,
 'h': 9,
 'j': 10,
 'm': 11,
 'n': 12,
 'o': 13,
 'p': 14,
 'r': 15,
 's': 16,
 't': 17,
 'u': 18,
 'z': 19}
```

Agora estamos criando um dicionário onde relacionamos cada caractere com um número, assim podemos representar os caracteres por esses números.

```
[3]  encode = lambda s: [letraParaIndex[c] for c in s] # encoder

     frase = 'fastcamp de LLM'
     frase = encode(frase)
     print(frase)
```

```
[8, 4, 16, 17, 5, 4, 11, 14, 0, 6, 7, 0, 2, 2, 3]
```

Depois de criar o dicionário com as relações entre letras e números, podemos criar uma função que pega cada caracter de um texto e troca por seu número correspondente no dicionário.

```
IndexParaLetra = { i:lt for i,lt in enumerate(caracteres) }
decode = lambda l: ''.join([IndexParaLetra[i] for i in l]) # decoder

print(decode(frase))
```

```
fastcamp de LLM
```

Também criamos o processo inverso, para pegarmos a representação numérica do texto e voltar a sua representação em caracteres. O problema de representar cada caractere como um número é que para textos gigantescos isso se torna inviável, pois geraria uma necessidade de processamento muito alta. Então uma maneira mais eficiente seria representar numericamente cada palavra ao invés de cada caractere.

```
[5]  import pandas as pd
     import numpy as np

     texto = {'texto':[
         'você está começando a aprender Transformer',
         'talvez você esteja começando a gostar',
     ]}
     texto = pd.DataFrame(texto)
     texto
```

	texto
0	você está começando a aprender Transformer
1	talvez você esteja começando a gostar

Próximas etapas: Gerar código com texto | Ver gráficos recomendados | New interactive sheet

```
[6]  texto["texto"].str.get_dummies(" ")
```

	Transformer	a	aprender	começando	esteja	está	gostar	talvez	você
0	1	1	1	1	0	1	0	0	1
1	0	1	0	1	1	0	1	1	1

Esse trecho de código que é apresentado no vídeo, usa a função get_dummies do pandas para pegar todas as palavras existentes no texto e mostrar a ocorrência de cada uma em cada trecho do texto. Porém ainda é um método inviável para textos grandes, já que teremos palavras com diversas ocorrências e outras com poucas ocorrências e isso geraria uma matriz gigante sem necessidade.

```
import tiktoken
enc = tiktoken.get_encoding("gpt2")
enc.encode('vamos aprender transformers')
```

```
[85, 321, 418, 2471, 13287, 6121, 364]
```

Para solucionar o problema que tínhamos basta usarmos o método de tokenização usado pelos modelos de LLM. Neste trecho de código acima, usamos a biblioteca tiktoken que irá fazer o processo de tokenização de nossos textos e nele usamos a mesma tabela de tokens do gpt2 da OpenAI.

```
[21] for i in range(len(texto)-1):
       x = texto[:i]
       y = texto[i]
       if x != []:
         print(f'Quando os dados forem: {x} o alvo é {y}')
```

```
Quando os dados forem: [85] o alvo é 321
Quando os dados forem: [85, 321] o alvo é 418
Quando os dados forem: [85, 321, 418] o alvo é 2471
Quando os dados forem: [85, 321, 418, 2471] o alvo é 13287
Quando os dados forem: [85, 321, 418, 2471, 13287] o alvo é 6121
```

Este trecho de código mostra a forma com que o GPT tenta prever o carácter e logo abaixo como o BERT tenta prever.

```
from random import randint

for i in range(len(texto)):
  x = texto
  y = texto.copy()
  idx_mask = randint(0,len(texto)-1)
  y[idx_mask] = '<mask>'
  print(f'Quando os dados forem: {x} o alvo é {y}')
```

```
Quando os dados forem: [85, 321, 418, 2471, 13287, 6121, 364] o alvo é [85, 321, '<mask>', 2471, 13287, 6121, 364]
Quando os dados forem: [85, 321, 418, 2471, 13287, 6121, 364] o alvo é [85, '<mask>', 418, 2471, 13287, 6121, 364]
Quando os dados forem: [85, 321, 418, 2471, 13287, 6121, 364] o alvo é [85, 321, 418, 2471, 13287, '<mask>', 364]
Quando os dados forem: [85, 321, 418, 2471, 13287, 6121, 364] o alvo é [85, 321, 418, 2471, '<mask>', 6121, 364]
Quando os dados forem: [85, 321, 418, 2471, 13287, 6121, 364] o alvo é [85, 321, 418, 2471, 13287, 6121, '<mask>']
Quando os dados forem: [85, 321, 418, 2471, 13287, 6121, 364] o alvo é [85, 321, '<mask>', 2471, 13287, 6121, 364]
Quando os dados forem: [85, 321, 418, 2471, 13287, 6121, 364] o alvo é [85, 321, 418, 2471, 13287, '<mask>', 364]
```

O GPT funciona fazendo uma previsão do próximo token se baseando sempre no token anterior. Já o BERT funciona escolhendo de forma aleatória um token o qual ele irá ocultar, aí ele terá que fazer a previsão para saber qual token deve ir naquela posição se baseando nos tokens adjacentes.

Conclusões

Os vídeos do card fornecem uma visão clara sobre a evolução das representações textuais no processamento de linguagem natural. O primeiro vídeo aborda os word embeddings, uma maneira de representar palavras em espaços vetoriais, permitindo encontrar relações semânticas entre elas. E o segundo vídeo explica desde métodos básicos de representação de caracteres até abordagens mais avançadas, como a tokenização usada em LLMs modernos.

Referências

Link do vídeo sobre Word Embeddings:
O que são Word Embeddings? | Processamento de Linguagem Natural | Leonardo Ri…

Link do vídeo ensinado a criar um LLM do zero:
Criando um LLM do Zero