<https://colab.research.google.com/github/axel-sirota/nlp-and-transformers/blob/main/module3/NLPTransformers_Mod3Demo1_Attention_ipynb.ipynb#scrollTo=edRPa0-vYVcA&uniqifier=1>

Introducing Attention

© Data Trainers LLC. GPL v 3.0.

**Author:** Axel Sirota

Attention is one of the most groundbreaking ideas that revolutionized NLP and AI on the latest years. However, it is difficult to encounter a demo that is solely focused on attention... until now.

Prep

[1]

4s

1

2

3

4

5

6

7

8

9

import matplotlib.pyplot as plt  
import seaborn as sns  
import numpy as np  
import re  
import gensim  
from nltk.data import find  
import nltk  
  
nltk.download("word2vec\_sample")

account\_circle

[nltk\_data] Downloading package word2vec\_sample to /root/nltk\_data...

[nltk\_data] Unzipping models/word2vec\_sample.zip.

True

Let's define some helper functions we need:

* The softmax funciton definition for Numpy arrays
* An Embedder that transforms a list of words into its embedding representation according to word2vec\_sample from the package nltk.

If you are unfamiliar with these concepts you are welcome to come to my other course **Implement Natural Language Processing for Word Embedding**

[4]

0s

1

2

3

def softmax(x, axis=0):  
    """Compute softmax values for each sets of scores in x."""  
    return np.exp(x) / np.sum(np.exp(x))

[5]

0s

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

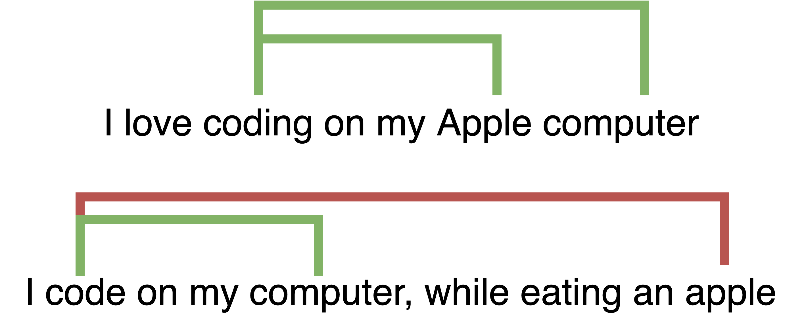
22

23

def get\_word2vec\_embedding(words):  
    """  
    Function that takes in a list of words and returns a list of their embeddings,  
    based on a pretrained word2vec encoder.  
    """  
    word2vec\_sample = str(find("models/word2vec\_sample/pruned.word2vec.txt"))  
    model = gensim.models.KeyedVectors.load\_word2vec\_format(  
        word2vec\_sample, binary=False  
    )  
  
    output = []  
    words\_pass = []  
    for word in words:  
        try:  
            output.append(np.array(model.word\_vec(word)))  
            words\_pass.append(word)  
        except:  
            pass  
  
    embeddings = np.array(output)  
    del model  # free up space again  
    return embeddings, words\_pass

Attention 101: Dot product Attention

The idea behind attention is simple, if you take any word, like Apple, its meaning will change with respect with the other words in the sentence. For example below, In the first sentence Apple refers to the company and has strong relationship with coding and computer; on the second one refers to the fruit and therefore at most it would have relationship with eating, but not coding.

Words relevance change with context

What I just spoke, is known as **Cross Attention**, because you will calculate the relationship of one word with respect to **all** the others in the sentence. In an image it would be:

A diagram of a software system

Description automatically generatedAttention

[ ]

1

In code it is even easier, don't worry about the image above it will make sense as we evolve through the course. The really important part is the following:

aij=f(hi,sj)

Where ai,j stands for the alignment of the word h\_i with the output word s\_j. The alignment may sound fancy, but it simply means how strongly connected those 2 words are in that sentence, like the Apple example!

The key is that the function f can be anything. In the original paper, and the one we are implementing now it is the dot product, which you have probably seen before, and if not check the course I referenced before, **Implement Natural Language Processing for Word Embedding**:

ai,j=dotproduct(hi,sj)=hTi∗sj

So this means that for a given initial word, which is a row in the matrix we created, we have a Tensor of how aligned it is with that output word; we call that Tensor c\_k or context vector.

And here comes the important stuff number 2, which is we take softmax to obtain weights, those wieghts will tell me for that input word how much weight (and importance) I should put into any output word. That is the attention matrix.

zj=softmaxk(cj,k)

If we multiply this with the context vector of an encoder we have an empowered context memory tensor that can be fed into the decoder, as it is done in Transformers. We will implement all of this alongside this module

[6]

0s

1

2

3

4

5

6

7

8

9

10

def dot\_product\_attention(hidden\_states, previous\_state):  
  
    # [T,d]\*[d,N] -> [T,N]  
    scores = np.matmul(previous\_state, hidden\_states.T)  
    w\_n = softmax(scores)  
  
    # [T,N]\*[N,d] -> [T,d]  
    c\_t = np.matmul(w\_n, hidden\_states)  
  
    return w\_n, c\_t

Now we will use a helper function that will plot those attention weights I told you about

[7]

0s

1

2

3

4

5

6

7

8

9

def plot\_attention\_weight\_matrix(weight\_matrix, x\_ticks, y\_ticks):  
    """Function that takes in a weight matrix and plots it with custom axis ticks"""  
    plt.figure(figsize=(15, 7))  
    ax = sns.heatmap(weight\_matrix, cmap="Blues")  
    plt.xticks(np.arange(weight\_matrix.shape[1]) + 0.5, x\_ticks)  
    plt.yticks(np.arange(weight\_matrix.shape[0]) + 0.5, y\_ticks)  
    plt.title("Attention matrix")  
    plt.xlabel("Attention score")  
    plt.show()

Testing it out

Let's try with some words related to royalty and some related to food:

[8]

10s

1

2

3

4

words = ["king", "queen", "royalty", "food", "apple", "pear", "computers"]  
word\_embeddings, words = get\_word2vec\_embedding(words)  
weights, \_ = dot\_product\_attention(word\_embeddings, word\_embeddings)  
plot\_attention\_weight\_matrix(weights, words, words)

account\_circle

As you can see, this was successful! We could detect the relationships between apple, pear and a little less food; aas one cluster. Then another cluster of the royalty, and finally commputers alone, so it detected what it is supposed to! In the next demo we will implement other forms of attention, ie: changing that function f