

Self Attention

Self Attention is also an embedding technique just like other word embeddings like word2vec but with a twist.

In self attention embeddings are generated keeping word context in picture. But that was not the case with word2vec.

Suppose a custom word2vec model is trained on movie review dataset, in that dataset suppose word Apple was there at multiple places. 80 percent of the time apple was used for fruit and 20 percent of time apple was used as a technology.

So If custom word2vec is asked to generate embedding(vectors) for apple in two dimensional space i.e [fruit, tech]. It will generate an apple embedding something like [0.8,0.2].

Reason being, the word apple was used most of time as "fruit" i.e 80 percent.

But suppose with such vector embedding [0.8, 0.2] ,if we try to process sentence in neural network

"Apple Launched a new phone while I was eating **Apple**"

Here, if both Apple are processed taking the embedding [0.8, 0.2] both the times, it is not right.

Thus there is need to move from static embedding(word2vec) to contextual embedding(using self attention) i.e dynamic embedding.

Mutihead attention is an extended version of self attention.

How Self-Attention Works:

The goal of self-attention is to compute **contextualized embeddings** for each word. This is achieved by focusing on the relationships between each word and every other word in the sequence.

Consider the sentence:

"The animal didn't cross the street because it was too tired."

1. Inputs

Each word has: a **query**, **key**, and **value** vector are computed through learned linear transformations.

- **Query vector (Q):** Represents what this word is "asking" for.
- **Key vector (K):** Represents the "information" each word provides.
-

Understanding **Query (Q)** and **Key (K)** vectors is key to grasping how attention works. Let me explain them step by step with an example.

Query (Q) and Key (K): The Roles

1. **Query (Q):**
 - Represents what a word is **looking for** or **asking about**.
 - For example, if we're focusing on the word "**it**," its Query vector is like a spotlight saying:
"Tell me which words in this sentence are relevant to me."
2. **Key (K):**
 - Represents the **information provided by each word**.
 - Each word's Key vector tells the model how **important it is** in answering other words' queries.
3. **Value vector (V):** Represents the actual information to be combined for each word.

Example: Q and K in Action

Sentence:

"The animal didn't cross the street because it was too tired."

We'll focus on the word "**it**" and see how its Query (Q) interacts with Keys (K) of other words.

Step 1: Assign Query and Key Vectors

For simplicity, let's assume:

- Each word has a 3-dimensional **Query (Q)** and **Key (K):**

Word	Query (Q)	Key (K)
"The"	[0.1, 0.2, 0.3]	[0.2, 0.1, 0.0]
"animal"	[0.5, 0.4, 0.3]	[0.6, 0.5, 0.4]
"didn't"	[0.2, 0.1, 0.1]	[0.1, 0.2, 0.1]
"cross"	[0.0, 0.1, 0.2]	[0.2, 0.1, 0.3]
"street"	[0.4, 0.3, 0.2]	[0.5, 0.4, 0.3]
"it"	[0.6, 0.5, 0.4]	[0.7, 0.6, 0.5]
"was"	[0.1, 0.1, 0.0]	[0.1, 0.0, 0.2]
"tired"	[0.7, 0.6, 0.5]	[0.9, 0.8, 0.7]

Step 2: Compute Dot Products (Relevance Scores)

To find how much "it" (Q) relates to each word's Key (K):

Relevance Score = Dot Product of Q (it) and K (other word)

Dot Product Formula:

$$\text{Dot Product} = Q_x \cdot K_x + Q_y \cdot K_y + Q_z \cdot K_z$$

For "it" (Query = [0.6, 0.5, 0.4]):

1. With "The" (Key = [0.2, 0.1, 0.0]):

$$0.6 \times 0.2 + 0.5 \times 0.1 + 0.4 \times 0.0 = 0.12 + 0.05 + 0.00 = 0.17$$

2. With "animal" (Key = [0.6, 0.5, 0.4]):

$$0.6 \times 0.6 + 0.5 \times 0.5 + 0.4 \times 0.4 = 0.36 + 0.25 + 0.16 = 0.77$$

3. With "tired" (Key = [0.9, 0.8, 0.7]):

$$0.6 \times 0.9 + 0.5 \times 0.8 + 0.4 \times 0.7 = 0.54 + 0.40 + 0.28 = 1.22$$

Similarly, compute for all other words.

Step 3: Normalize the Scores (Softmax)

Apply **softmax** to convert these raw scores into probabilities (attention weights).

Example (normalized):

"The"	: 0.05
"animal"	: 0.60
"didn't"	: 0.02
"cross"	: 0.03
"street"	: 0.20
"it"	: 0.00
"was"	: 0.02
"tired"	: 0.08

- Self-attention computes scores to determine the importance of other words in the sentence for a specific word.
- Example: While processing "it", self-attention will compute how strongly "it" is related to "animal", "street", and all other words.
- This shows that "it" has the strongest relationship with "animal" (0.70), followed by "street" (0.20).

How Q and K Interact:

The **dot product** of Query (Q) and Key (K) vectors determines the **relevance score** between two words:

- **High relevance:** A large dot product means the Query is aligned with the Key (i.e., the words are related).
- **Low relevance:** A small or negative dot product means the Query and Key are not aligned (i.e., the words are less related).

These scores are normalized using **softmax** to compute the attention weights.

Step 4: Weighted Sum

The **Weighted Sum** is computed by taking each word's **Value (V)** vector and multiplying it by its attention score. Then, all these weighted vectors are summed up.

Assume the following **Value (V) vectors** for simplicity(it is diff from **Q and K vector**). **Q and K just used to calculate attention weight, which will be multiplying it with Value(V) vector:**

```
"The"      : [0.1, 0.2, 0.3]
"animal"   : [0.7, 0.6, 0.5]
"didn't"   : [0.1, 0.1, 0.2]
"cross"    : [0.0, 0.1, 0.1]
"the"      : [0.2, 0.3, 0.2]
"street"   : [0.5, 0.4, 0.3]
"because"  : [0.1, 0.2, 0.1]
"it"       : [0.6, 0.5, 0.4]
"was"      : [0.1, 0.1, 0.0]
"too"      : [0.0, 0.0, 0.1]
"tired"    : [0.8, 0.7, 0.6]
```

- Multiply each **Value vector (V)** by its corresponding attention score:

Calculation for "it":

"The" : $[0.1, 0.2, 0.3] \times 0.05 = [0.005, 0.01, 0.015]$
 "animal" : $[0.7, 0.6, 0.5] \times 0.70 = [0.49, 0.42, 0.35]$
 "didn't" : $[0.1, 0.1, 0.2] \times 0.02 = [0.002, 0.002, 0.004]$
 "cross" : $[0.0, 0.1, 0.1] \times 0.01 = [0.0, 0.001, 0.001]$
 "the" : $[0.2, 0.3, 0.2] \times 0.01 = [0.002, 0.003, 0.002]$
 "street" : $[0.5, 0.4, 0.3] \times 0.20 = [0.1, 0.08, 0.06]$
 "because" : $[0.1, 0.2, 0.1] \times 0.01 = [0.001, 0.002, 0.001]$
 "it" : $[0.6, 0.5, 0.4] \times 0.00 = [0.0, 0.0, 0.0]$
 "was" : $[0.1, 0.1, 0.0] \times 0.01 = [0.001, 0.001, 0.0]$
 "too" : $[0.0, 0.0, 0.1] \times 0.00 = [0.0, 0.0, 0.0]$
 "tired" : $[0.8, 0.7, 0.6] \times 0.00 = [0.0, 0.0, 0.0]$

Weighted Sum = $[0.005, 0.01, 0.015]$ (The)
 + $[0.49, 0.42, 0.35]$ (animal)
 + $[0.002, 0.002, 0.004]$ (didn't)
 + $[0.0, 0.001, 0.001]$ (cross)
 + $[0.002, 0.003, 0.002]$ (the)
 + $[0.1, 0.08, 0.06]$ (street)
 + $[0.001, 0.002, 0.001]$ (because)
 + $[0.0, 0.0, 0.0]$ (it)
 + $[0.001, 0.001, 0.0]$ (was)
 + $[0.0, 0.0, 0.0]$ (too)
 + $[0.0, 0.0, 0.0]$ (tired)

Final Weighted Sum = $[0.598, 0.518, 0.432]$

- The resulting vector **[0.598, 0.518, 0.432]** is the updated representation for the word "it". This vector incorporates contextual information, emphasizing the relationship between "it" and the most relevant words in the sequence (especially "animal" and "street").

- The weighted sum ensures that the word's new representation is a **contextualized embedding**, capturing the word's meaning within the sentence. For "it", the new vector heavily reflects "animal", allowing the model to understand that "it" refers to "the animal."
- The vector [0.598, 0.518, 0.432] is the contextualized embedding for the word "it."
- This embedding reflects that "it" is closely related to "animal" (and to a lesser extent, "street").
- Similarly, every word in the sentence undergoes the self-attention mechanism to produce its own contextualized vector.
- The new embeddings are no longer isolated word embeddings (like word2vec or GloVe).
- Instead, they are contextualized representations that capture the meaning of each word in the sentence context.

Key Takeaways:

1. **Every word gets a unique vector** after self-attention, capturing its relationship with other words in the sequence.
2. These contextualized embeddings are passed to subsequent layers (e.g., feedforward layers in Transformers) to refine their understanding further.
3. This mechanism allows Transformers to understand context-dependent meanings, such as resolving ambiguities (e.g., "it" referring to "animal").

Key Takeaways:

- **Query (Q):** What a word is asking for.
 - Example: For "it," Q is asking, *"Which words are relevant to understanding me?"*
- **Key (K):** The information provided by other words.
 - Example: "animal" has a high K alignment with "it" because it likely refers to the same entity.

The interaction between Q and K (via dot product and softmax) determines **how much attention each word gives to others**.

If the above attention mechanism is repeated multiple times, it is called multi headed self attention.

Why Self-Attention Works Well

1. Word Dependencies:

- Self-attention identifies dependencies like the relationship between "it" and "animal" across the sentence, regardless of distance.

2. Parallel Processing:

- Unlike RNNs, which process sequences sequentially, self-attention processes the entire sentence simultaneously, making it computationally efficient.

Conclusion

Self-attention allows the model to dynamically assign importance to words based on their relationships within the same sequence. This example illustrates how **"it"** gets contextualized to refer to **"animal"** and not **"street,"** showcasing self-attention's ability to resolve ambiguities in language.