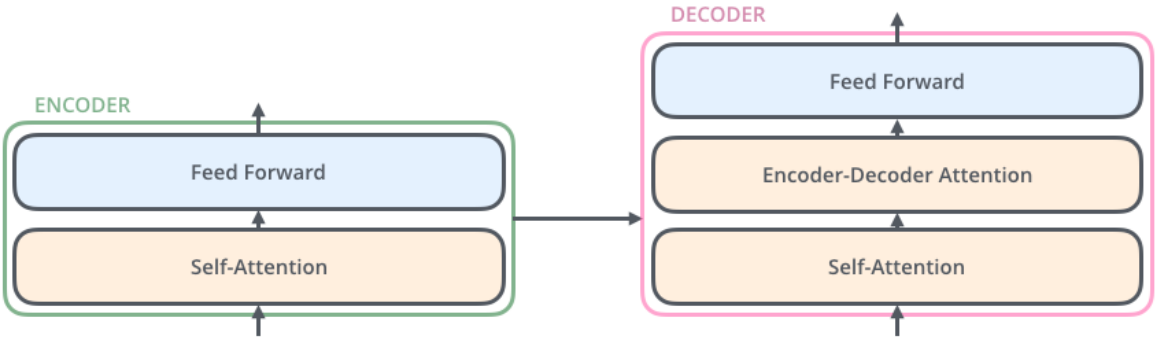
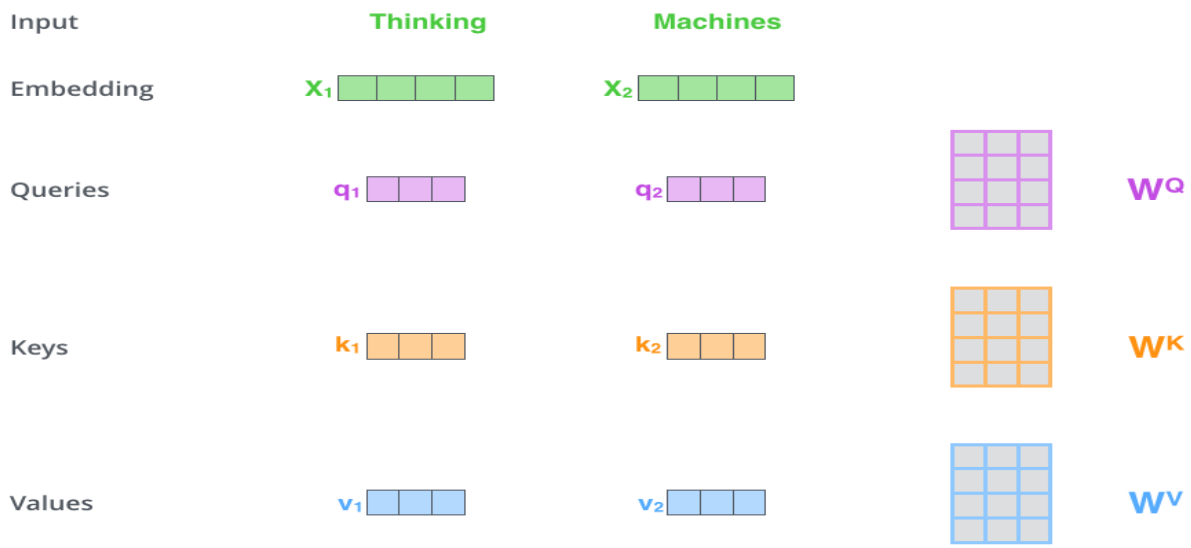
Q1: What is the purpose of self-attention, and how does it facilitate capturing dependencies in sequences?

Self Attention serves the purpose of capturing dependencies and relationships between different elements in a sequence, such as words in a sentence. Self-attention enables us to understand and process sequential data effectively.
And tell how much attention we need to give to each word in the given sequence.



Here we have query, key, values pair which helps us to find this relationhip

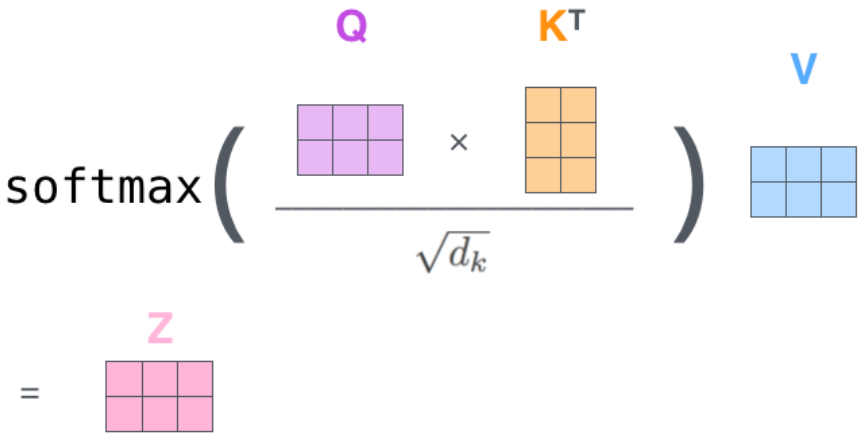


For each words query we multiply with other words keys to get the matrix of size seq_len, seq_len which we multiply with values vector to get the final attention values.

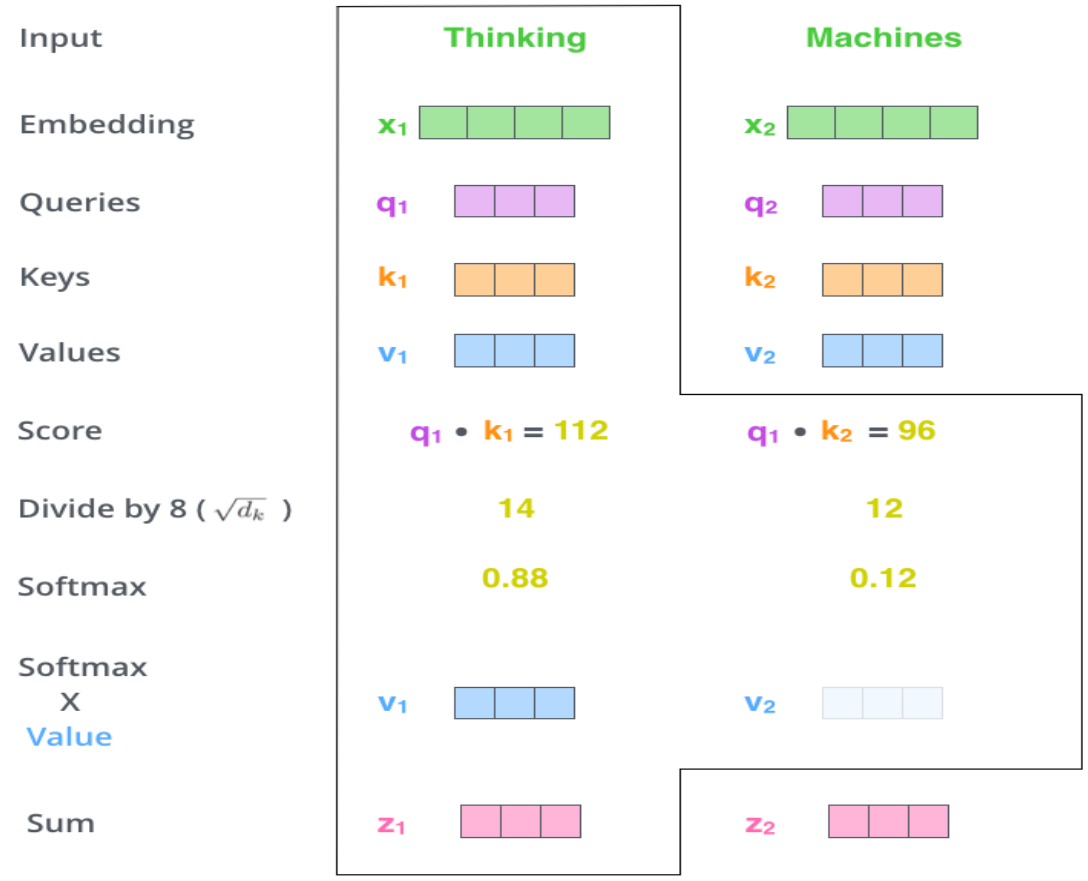
We have values vector to save the compute and copying cost of the real input matrix.

- Get relationship :

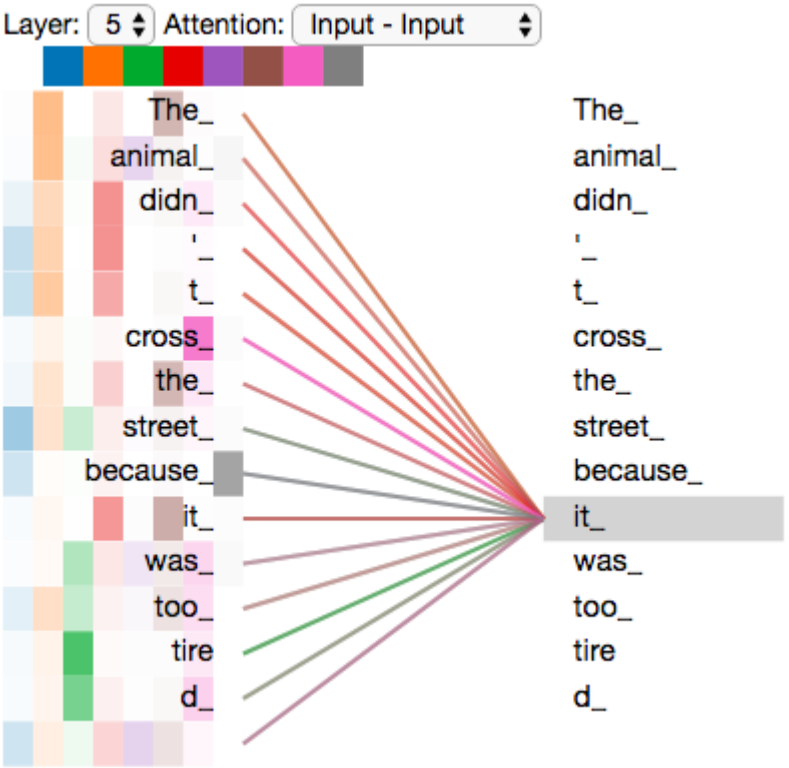
Self-attention allows a model to consider how each element in the sequence (e.g., word in a sentence) relates to every other element. This means it can capture both short-range and long-range dependencies within the sequence, giving the model the ability to understand the context and relationships between different words or tokens.

- 

-
- And the we find the softmax values to get each words best attentions and multiply with values matrix.
- Self-attention computes weighted combinations of all elements in the sequence for each element. These weights are learned during training and determine the importance of each element's contribution to the representation of the current element. This means that the model can assign higher weights to elements that are more relevant to the current element, capturing the strength of their relationships.
-



-
- Now for each word we get a corresponding attention vector to tell how much to give on each word.

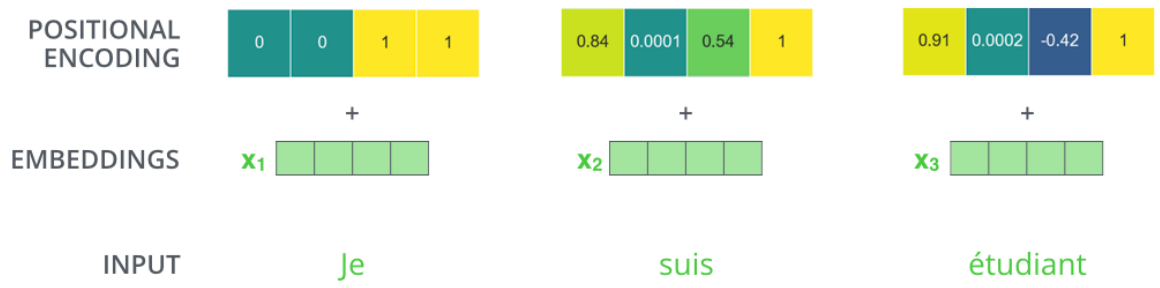


-

- In this dia we see how much important a word is to other word.
- Most of the time the matrix has large values on diagonals.
- Self-attention is often implemented with multiple attention "heads" that learn different aspects of the relationships in the data e.g. for a sentence it can learn diff patterns, nouns, verb relationship etc. hence it allows the model to capture a wide range of dependencies and relationships simultaneously, leading to improved performance.

Q2. Why do transformers use positional encodings in addition to word embeddings? Explain how positional encodings are incorporated into the transformer architecture.

Transformers use positional encodings in addition to word embeddings because word any embeddings alone do not inherently encode information about the positions they can at max store the context but to store order of words in a sequence they are not capable enough. The order of words is essential for meaning, context, and interpretation. Transformers use positional encodings in addition to word embeddings because word embeddings alone do not inherently encode information about the position or order of words in a sequence. This is a crucial distinction because in many natural language processing tasks, the order of words is essential for meaning, context, and interpretation.



Eg here we are storing positional encoding with the original embeddings itself to get the positional information among then words.

Other Properties :

- Preserving Sequence Order: Word embeddings itself cant store order of words in a sequence. Positional encodings are designed to fill this gap by giving positional information about word positions within the sequence.

- Retaining Temporal Information: Positional encodings enable the model to retain and understand this temporal context.
- Distinguishing Identical Words: In a given sentence, the same word may appear multiple times, If we dont consider the positional encodings, model would treat all instances of the same word as identical.

Positional encodings are an integral part of the transformer architecture, responsible for imparting information about the positions of elements within a sequence. Their inclusion is crucial for tasks that depend on understanding the order of elements, such as those encountered in natural language processing. Here's an alternative explanation of how positional encodings are integrated into the transformer architecture:

Positional Encoding Vectors: Within transformers, positional encodings are realized as vectors, possessing the same dimension as word embeddings. These vectors serve the purpose of providing a distinctive encoding for every position in the sequence.

Fixed Patterns: Transformers often rely on established mathematical patterns to define positional encodings. The most common patterns include:

- Sine and Cosine Functions: These functions create smooth, periodic patterns that are superimposed onto word embeddings. The exact nature of these patterns is determined by the positions of elements within the sequence.
- Learned Positional Embeddings: In certain transformer variants, positional encodings are learned during the model's training. This can be accomplished either as an integral component of the architecture or as a separate embedding layer, enabling the model to adapt to the unique characteristics of the input data.

Addition of Positional Information: Positional encoding vectors are added element-wise to the corresponding word embeddings for each position in the sequence. This addition operation fuses the information regarding a word's meaning (captured by word embeddings) with its positional context (captured by positional encodings).

Static Positional Information: It's important to emphasize that positional encodings remain static throughout the model's training and application. They do not change based on the input data or the specific task being performed. Instead, they encode a fixed, universally applicable pattern of positional information.

Handling Variable-Length Sequences: Positional encodings enable transformers to flexibly accommodate sequences of varying lengths.

Regardless of the input sequence's length, the model can effectively differentiate elements based on their positions, thanks to the information conveyed by positional encodings.

In summary, the incorporation of positional encodings into the transformer architecture ensures the model's ability to understand and leverage the positional order of elements within a sequence. By combining them with word embeddings, transformers construct a comprehensive representation that encompasses both the content and the position of each word or token. This capacity allows transformers to proficiently process and model the sequential attributes of the data, rendering them exceptionally adaptable for a diverse range of tasks, extending beyond natural language processing.

Report:

Hyperparameters : `embedding_dim = 100`

```
hidden_dim = 50
num_layers = 2
batch_size_lstm = 32
loss_function = nn.CrossEntropyLoss()
learning_rate = 0.01
model = Transformer().to(device)
optimizer = optim.Adam(model.parameters(), lr=0.01, weight_decay=1e-5)
```

This is theb best model is got