

Pre-fill stage Transformers: In decoder-only transformers, refer to the phase where the model processes initial input sequence of tokens before it starts generating new tokens. This stage is crucial in many auto-regressive models.

Because it establishes the initial context that the model will use to generate future predictions.

1. Converting Input Prompt to word embeddings

- The main idea of word embeddings is to use a relatively simple neural network that has 1 input for every word and symbol in the vocabulary that we want to use.
- The inputs are then connected to something called an activation function.
- Reusing the exact same word embedding network for each word in the prompt allows the decoder-only transformers to handle different prompts.

2. Next step is adding positional encoding to word embedding

- Positional encoding is used to keep track of word order.
- One of the most commonly used methods uses a sequence of alternating sine and cosine functions.

3. Masked attention: In decoder-only transformers, the attention is typically causal (masked), meaning a token can only

attend to previous tokens in the sequence, not future ones. This ensures that the model generates tokens in an auto-regressive manner.

Detailed Computation Steps

1. Each token's embedded vector is transformed into 3 different vectors: Query (Q), Key (K), and Value (V). These values are used to determine the attention scores.

$$Q = XW^Q$$

$$K = XW^K$$

$$V = XW^V$$

Here W^Q , W^K , and W^V are parameter matrices that are learned during training.

Why Break Up into 3 different Vectors

- In the self-attention mechanism of a transformer, each input vector is transformed into the 3 vectors above, below is what they signify:

1. Query (Q): A query is a representation of a token used to score how well it matches every other token. Essentially when considering a specific token, its query vector is used to seek out which tokens (Keys) are most relevant to it.
2. Key (K): A key is associated with a token to be matched against queries. When a query is compared to this key, the resulting score determines the impact of the corresponding token's value on the output.
3. Value (V): A value is a representation of a token that is used to compute the final output. Once tokens are scored based on the query-key matches, the values are weighted by these scores and summed up to produce the output for the next layer.

Attention scores determine how much focus to place on other parts of the input sequence when processing a specific part of that sequence.

$$\text{Attention Score Calculation} = \frac{QK^T}{\sqrt{d_K}}$$

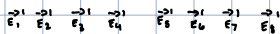
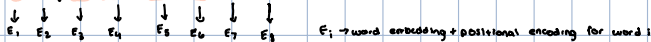
The scores are scaled down by the square root of the dimension of the key vectors (d_K) to stabilize the gradients during training.

4. Once the model has processed the input sequence during the pre-fill stage, it can begin generating new tokens. The pre-fill stage primes the model with an initial set of tokens that it will use

as context for future generation.

Example

a fluffy bat creature roamed the verdant forest



the goal is to create a new set of embeddings where each word considers the words around it to determine its meaning (Query Vectors)

to compute this take a weight vector W_Q and multiplying it by the embedding vectors

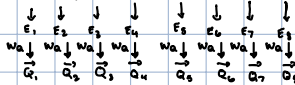
the entries of this matrix are parameters of the model (meaning true behavior is learned from data)

$$W_Q \cdot E_i = Q_i \text{ a query vector}$$

do this computation until you get a query vector for every token embedding

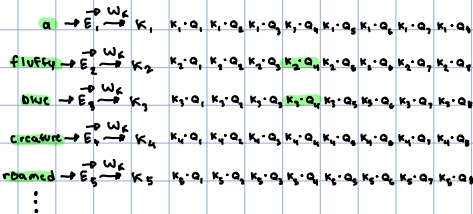
$$\begin{bmatrix} 1 \\ 512 \end{bmatrix} \times \begin{bmatrix} W_Q \\ 64 \end{bmatrix} = \begin{bmatrix} Q_i \\ 64 \end{bmatrix}$$

a fluffy bat creature roamed the verdant forest



conceptually this is checking if there are any words that change the words meaning

W_K = key matrix



= dot product values highlighted with reds are high values

all these dot products are being computed parallel

conceptually this is answering

if the dot product of a key and query vector is really high then the specific key embeddings attend to the query embeddings

to the query whether or

not a word is changing

its meaning

As at right now the grid of values ranges from $-\infty$ to ∞ giving us a score for how relevant each word is to updating the meaning of every other word

Next we will take a weighted sum along each column, weighted by relevance (so we want each column to add up to one, as if they were a probability distribution)