Pre-fill stage Transformers: in decoder only transformers refers 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 cosign squiggles

3. Masked attention: In decoder-only transformers, the attention is typically casual (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 (Query) $= XW^Q$
   - $K = XW^K$
   - $V = XW^V$

   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, it 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. Values (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.

$$\boxed{\text{• attention score calculation} = \frac{QK^T}{\sqrt{d_K}}}$$

the scores are scaled down by the square root of the dimension of the key vectors (d_K) to stabalize 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

$$\begin{array}{cccccccc}
\text{a} & \text{fluffy} & \text{blue} & \text{creature} & \text{roamed} & \text{the} & \text{verdant} & \text{forest} \\
\downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\
E_1 & E_2 & E_3 & E_4 & E_5 & E_6 & E_7 & E_8
\end{array}$$

$E_i \rightarrow$ word embedding + positional encoding for word i

$$\overrightarrow{E_1}\quad \overrightarrow{E_2}\quad \overrightarrow{E_3}\quad \overrightarrow{E_4}\quad \overrightarrow{E_5}\quad \overrightarrow{E_6}\quad \overrightarrow{E_7}\quad \overrightarrow{E_8}$$

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)

$W_Q \cdot E_i = \vec{Q}_1 \leftarrow$ query vector

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

$$1\ \underset{512}{\boxed{\quad \vec{E}_i \quad}} \times \underset{64}{\boxed{W_Q}}\ 512 = \underset{64}{\boxed{\quad \bar{Q}_1 \quad}}\ 1$$

$$\begin{array}{cccccccc}
\text{a} & \text{fluffy} & \text{blue} & \text{creature} & \text{roamed} & \text{the} & \text{verdant} & \text{forest} \\
\downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\
E_1 & E_2 & E_3 & E_4 & E_5 & E_6 & E_7 & E_8 \\
W_Q\downarrow & W_Q\downarrow & W_Q\downarrow & W_Q\downarrow & W_Q\downarrow & W_Q\downarrow & W_Q\downarrow & W_Q\downarrow \\
\vec{Q}_1 & \vec{Q}_2 & \vec{Q}_3 & \vec{Q}_4 & \vec{Q}_5 & \vec{Q}_6 & \vec{Q}_7 & \vec{Q}_8
\end{array}$$

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

- W_K = key matrix

$$\left.\begin{array}{lcccccccc}
\text{a} \rightarrow \vec{E}_1 \xrightarrow{W_K} K_1 & K_1\cdot Q_1 & K_1\cdot Q_2 & K_1\cdot Q_3 & K_1\cdot Q_4 & K_1\cdot Q_5 & K_1\cdot Q_6 & K_1\cdot Q_7 & K_1\cdot Q_8 \\
\text{fluffy} \rightarrow \vec{E}_2 \xrightarrow{W_K} K_2 & K_2\cdot Q_1 & K_2\cdot Q_2 & K_2\cdot Q_3 & K_2\cdot Q_4 & K_2\cdot Q_5 & K_2\cdot Q_6 & K_2\cdot Q_7 & K_2\cdot Q_8 \\
\text{blue} \rightarrow \vec{E}_3 \xrightarrow{W_K} K_3 & K_3\cdot Q_1 & K_3\cdot Q_2 & K_3\cdot Q_3 & K_3\cdot Q_4 & K_3\cdot Q_5 & K_3\cdot Q_6 & K_3\cdot Q_7 & K_3\cdot Q_8 \\
\text{creature} \rightarrow \vec{E}_4 \xrightarrow{W_K} K_4 & K_4\cdot Q_1 & K_4\cdot Q_2 & K_4\cdot Q_3 & K_4\cdot Q_4 & K_4\cdot Q_5 & K_4\cdot Q_6 & K_4\cdot Q_7 & K_4\cdot Q_8 \\
\text{roamed} \rightarrow \vec{E}_5 \xrightarrow{W_K} K_5 & K_5\cdot Q_1 & K_5\cdot Q_2 & K_5\cdot Q_3 & K_5\cdot Q_4 & K_5\cdot Q_5 & K_5\cdot Q_6 & K_5\cdot Q_7 & K_5\cdot Q_8 \\
\vdots & & & & & & & &
\end{array}\right\}$$

all these dot products are being computed parallel

🟩 = dot product values highlighted with this are high values

- conceptually this is answering to the query whether or not a word is changing its meaning

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

As of 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)