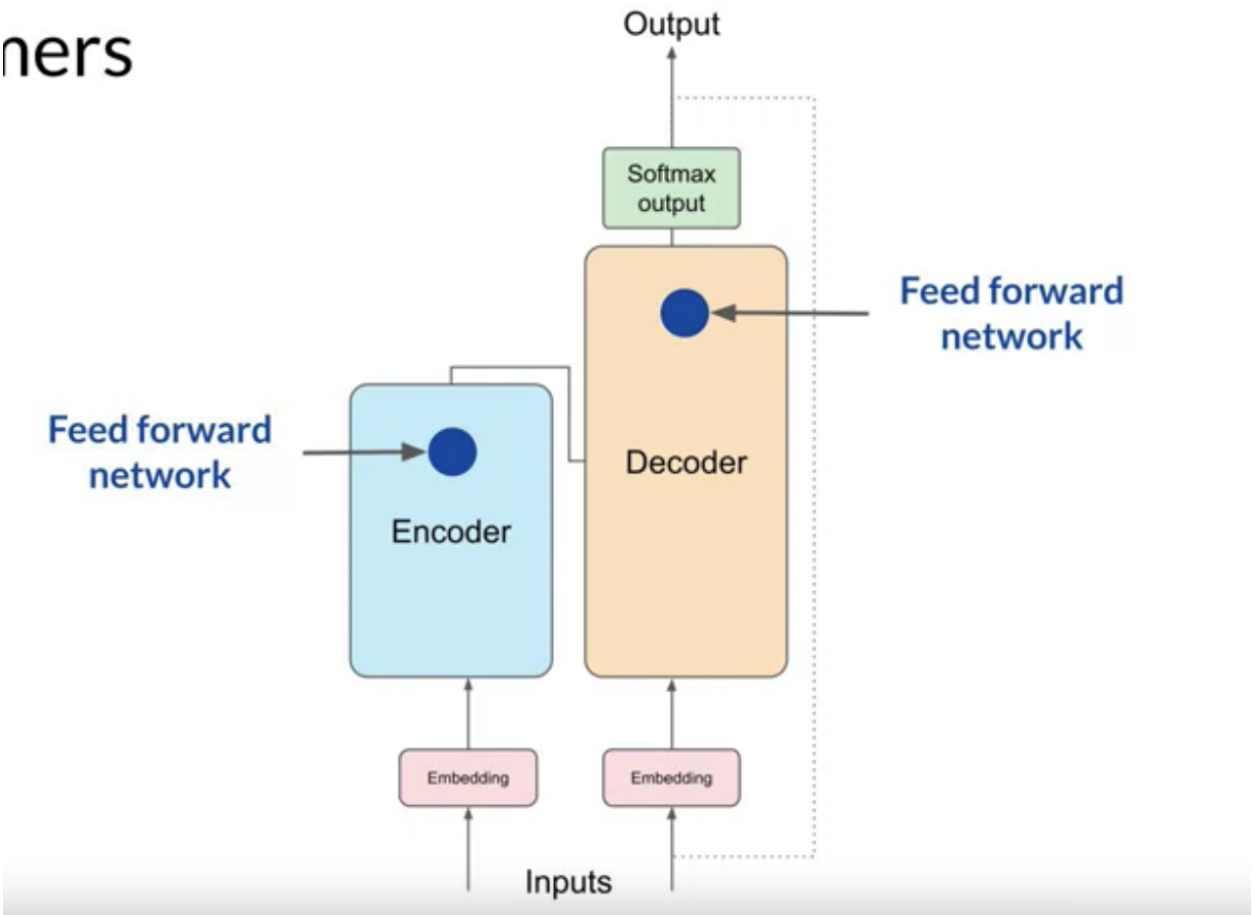
Transformer Architecture

Progress

RNN → attention → self-attention → multi-headed self-attention → transformer → BERT

Transformer Architecture



- The transformer architecture is split into two distinct parts, **the encoder and the decoder**.

▼ Encoder

- Before passing texts into the model to process, you must **first tokenize the words.**
    - This converts the words into numbers, with each number representing a position in a dictionary of all the possible words that the model can work with.
        - various tokenization methods. For example, **token IDs matching two complete words, or using token IDs to represent parts of words.**

> 💡 **You must use the same tokenizer when you generate text**.

- Input is represented as numbers, you can pass it to the **embedding layer. This layer is a trainable vector**

> 💡 **Embedding space**, **a high-dimensional space where each token is represented as a vector and occupies a unique location within that space.**

- **Each word has been matched to a token ID, and each token is mapped into a vecto**r.
- Each token ID in the vocabulary is matched to a multi-dimensional vector, and the intuition is that these vectors learn to encode the meaning and context of individual tokens in the input sequence.
    - Word2Vec approach.
- Adding the **positional encoding**, **you preserve the information about the word order and don't lose the relevance of the position of the word in the sentence.**
- The model analyzes the relationships between the tokens in your input sequence. As you saw earlier, this allows the model to attend to different

parts of the input sequence to better capture the **contextual dependencies** between the words.

> 💡 **Multi-headed self-attention: This means that multiple sets of self-attention weights or heads are learned in parallel independently of each other.**

- The intuition here is that each self-attention head will learn a different aspect of language.
- **The weights of each head are randomly initialized and given sufficient training data and time, each will learn different aspects of language.**
- **The output is processed through a fully-connected feed-forward network.**
- Pass logits to a final **softmax layer, where they are normalized into a probability score for each word.** This output includes a probability for every single word in the vocabulary, so there's likely to be thousands of scores here. **One single token will have a score higher than the rest.**

▼ Decoder

- This representation is inserted into the middle of the decoder to influence the **decoder's** self-attention mechanisms.
- **A start of sequence token** is added to the input of the decoder. This triggers the decoder to predict the next token, which it does based on the contextual understanding that it's being provided from the encoder.
- The output of the decoder's self-attention layers gets passed through the decoder feed-forward network and **through a final softmax output layer**. At this point, we have our first token. You'll continue this loop, passing the output token back to the input to trigger the generation of the next token, until the model predicts **an end-of-sequence token**.
- At this point, the final sequence of tokens can be detokenized into words, and you have your output.

▼ Output of Softmax Layer

The output of Softmax Layer: a list of word/token with its probabilities.

- Greedy: the work/token with the highest probability is selected.
- Random sampling: select a token using a random-weighted strategy across the probabilities of all tokens.