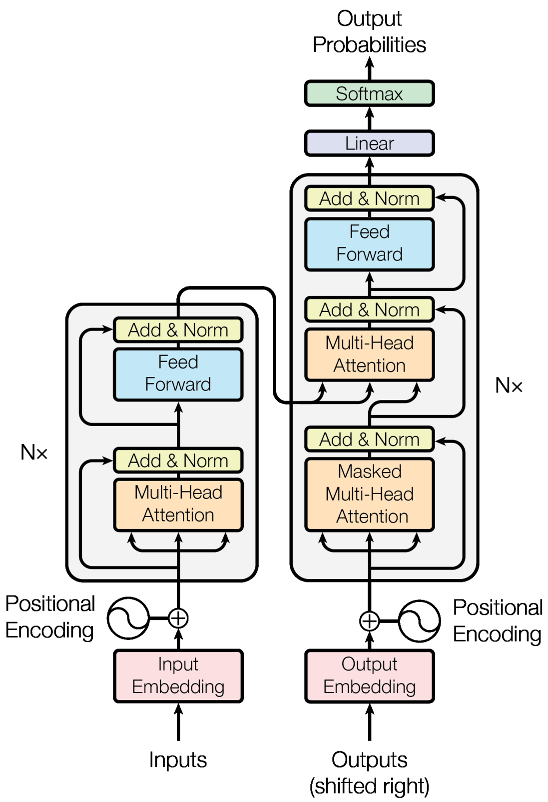
Transformer: Intuition

We try to briefly explain what each the "moving parts" of the Transformer is doing.

Transformer (Encoder/Decoder)



General

Each of the paths in the Transformer is a vector of length d_{model}

- sometimes just referred to as d

Having a common length simplifies the architecture

- can stack Transformer blocks (since input and output are same size)
- Self-Attention and Cross-Attention:
    - map a query of size d to an output of size d
- Needed for the Residual Connection (Add and Norm)
    - adding the input of Attention to the output of Attention
        - need to be same length

Residual connections

- Residual connections from Intro course (RNN_Residual_Networks.ipynb)

Embedding

Words (really: tokens) are *categorical* variables.

Categorical variables are usually encoded as long vectors via One Hot Encoding (OHE)

- very long: number of distinct elements in class
    - e.g., number of words in vocabulary
- *sparse*: only a single non-zero element in the vector
- the similarity (e.g., dot product) of two related words (e.g., "cat", "cats") is zero !
    - same as for two unrelated words (e.g., "cat", "car")

An *Embedding* is a a *short* and *dense* vector representation of words (tokens).

In addition to being shorter (and dense: many non-zero elements possible) their construction results in

- the similarity of embeddings for two related words being *non-zero*

This makes Embeddings much more valuable for NLP.

The *Embedding Layer* converts the OHE representation to an Embedding.

See the [module from the Intro course (NLP_Embeddings.ipynb)] for details.

Positional Encoding

The Transformer input is a *sequence*

- there is a total ordering between elements based on absolute position

The Transformer needs to be able to discern

- at least: the *relative* ordering of two elements in different positions in the sequence

The *Positional Encoding* layer

- adds a vector that encodes position
- to the Embedding
- such that the Transformer has a representation with both meaning and positions

This is much more involved than simply using an integer to encode the position.

The fundamental operation of a Neural Network is matrix multiplication

- the positional encoding needs to be preserved as it traverses the layers

The details are not trivial.

See the module on [Positional Embeddings (Transformer_PositionalEmbedding.ipynb)](Transformer_PositionalEmbedding.ipynb) if you are interested.

Encoder

The Encoder style Transformer can either be used

- stand-alone: create a fixed length, alternate length representation of the input
    - for further processing: e.g., Classification
- as part of an Encoder-Decoder architecture
    - transform the Input sequence $\mathbf{x}$ in a *processed sequence* $\bar{\mathbf{h}}_{(1)}, \ldots, \bar{\mathbf{h}}_{(\bar{T})}$
    - to be consumed by a Decoder

Encoder Self-Attention

The Self-Attention allows each $\bar{\mathbf{h}}_{(\bar{t})}$ to depend on the **complete** input sequence $\mathbf{x}$.

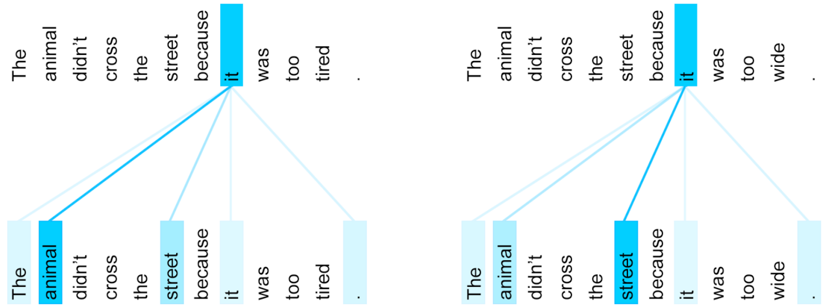
If we view $\bar{\mathbf{h}}_{(\bar{t})}$ as the "meaning" of $\mathbf{x}_{(t)}$

- it is a meaning based on the *full context*
    - not just the preceding elements $\mathbf{x}_{(1:t-1)}$

By making the meaning dependent on the full context, we can disambiguate the meaning of the world "it"

Attention weights

Thickness of the blue lines indicate the strength of attention to other tokens



Picture from: https://1.bp.blogspot.com/-AVGK0ApREtk/WaiAuzddKVI/AAAAAAAAB_A/WPV5ropBU-cxrcMpqJBFHg73K9NX4vywwCLcBGAs/s1600/image2.png

Decoder

The salient characteristic of a Decoder style Transformer is the *autoregressive* behavior

- generates an output one token at a time
- by appending each generated token to the sequence of already-generated tokens

The Decoder style Transformer can either be used

- stand-alone: for generative tasks
- as part of an Encoder-Decoder architecture

Decoder Self-Attention

The input needs to decide why of the previously-generated Decoder outputs (now Decoder inputs) to attend to.

There are multiple potential uses for this

- to help generate the "next" token (goal of the Decoder), by referencing the partially complete Decoder output
- to help in the Cross Attention step
    - decide which part of the Inputs $\mathbf{x}$ (Decoder Outputs) to attend to
    - "looking up" facts, e.g., our Question Answering example or Language Translation example

Note the use of Causal Masking

- we can only reference the Decoder output already generated

Decoder-Encoder Cross-Attention

The output of the Decoder Self-Attention is used as a "query"

- to reference the relevant part of the Input $\mathbf{x}$

Feed Forward Network (FFN)

Maps the output of the Decoder-Encoder Attention into the "next output token".

- actually: it is still an embedding of the next token, rather than the true next token
    - that way: it can be appended to the already-generated output to become the Decoder input for next position

This acts as a Classifier

- mapping the input
- to a vector of logits
    - one element per possible element of the Output Vocabulary

There is some evidence that

- the parameters of the FFN are where "world knowledge" is stored
    - every "fact" learned during training

Linear

This layer is append *only* to the final block in the stacked Transformer blocks.

It acts as a typical Classifier

- "classifies" the final block's output of length d
- returning a vector
    - whose length is equal to number of elements of the Vocabulary
    - each element is a logit
        - to be converted into probability distribution over elements of the Vocabulary

Softmax

Converts the logit for each possible element of the Vocabulary

- into Probability that the element is the next Decoder Output

In [2]:

```
print("Done")
```

```
Done
```