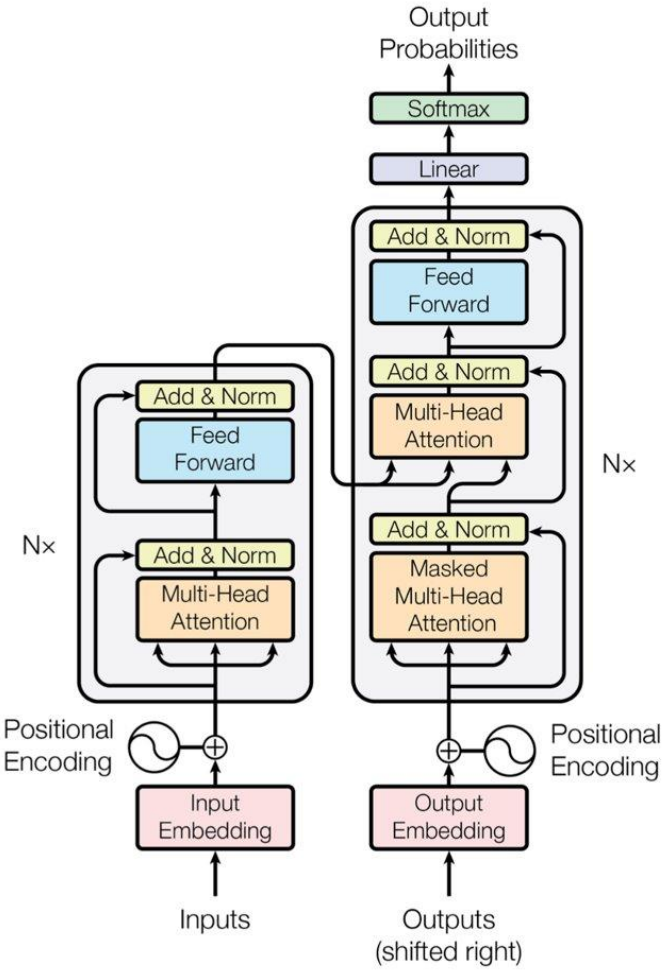
CSE 465
Lecture 14

Transformer

Attention is all you need

- It is an encoder-decoder architecture
- The input goes through encoder
  - Then through decoder
- On the encoder side
  - Each block has two sublayers
    - Multihead attention then feedforward
- On the decoder side
  - Each block has three sublayers
    - Two multihead attention followed by a feedforward layer
  - However, the first MHA layer is similar to the decoder MHA but the second one is different
- The attention mechanism is a “word to word” operation
  - Token to token operation, but we can think of it at the word level to simplify the explanations
- The attention mechanism will find how each word relates to all other words in a sequence, including the word being analyzed



Embedding

- Most real-world data (text, images, etc.) can’t be processed directly by machine learning models.
- They need to be turned into numbers.
  - But not just any numbers—we want numbers that preserve meaning and relationships.

Domain	Embedding Example	What It Represents
NLP	Word2Vec, GloVe, BERT embeddings	Words or sentences
Vision	CNN embeddings	Features of an image
Recommenders	User and item embeddings	Preferences and product attributes
Audio	Audio embeddings from spectrograms	Pitch, tone, speaker identity, etc.

Self Attention

- Attention is a mechanism that allows neural networks to assign a different amount of weight or "attention" to each element in a sequence
  - Instead of using a fixed embedding for each token, the current sequence computes a weighted average of each embedding
- Given a sequence of token embeddings $x_1, ..., x_n$, self-attention produces a sequence of new embeddings $x_1', ..., x_n'$ where each x'_i is a linear combination of all the x_j:

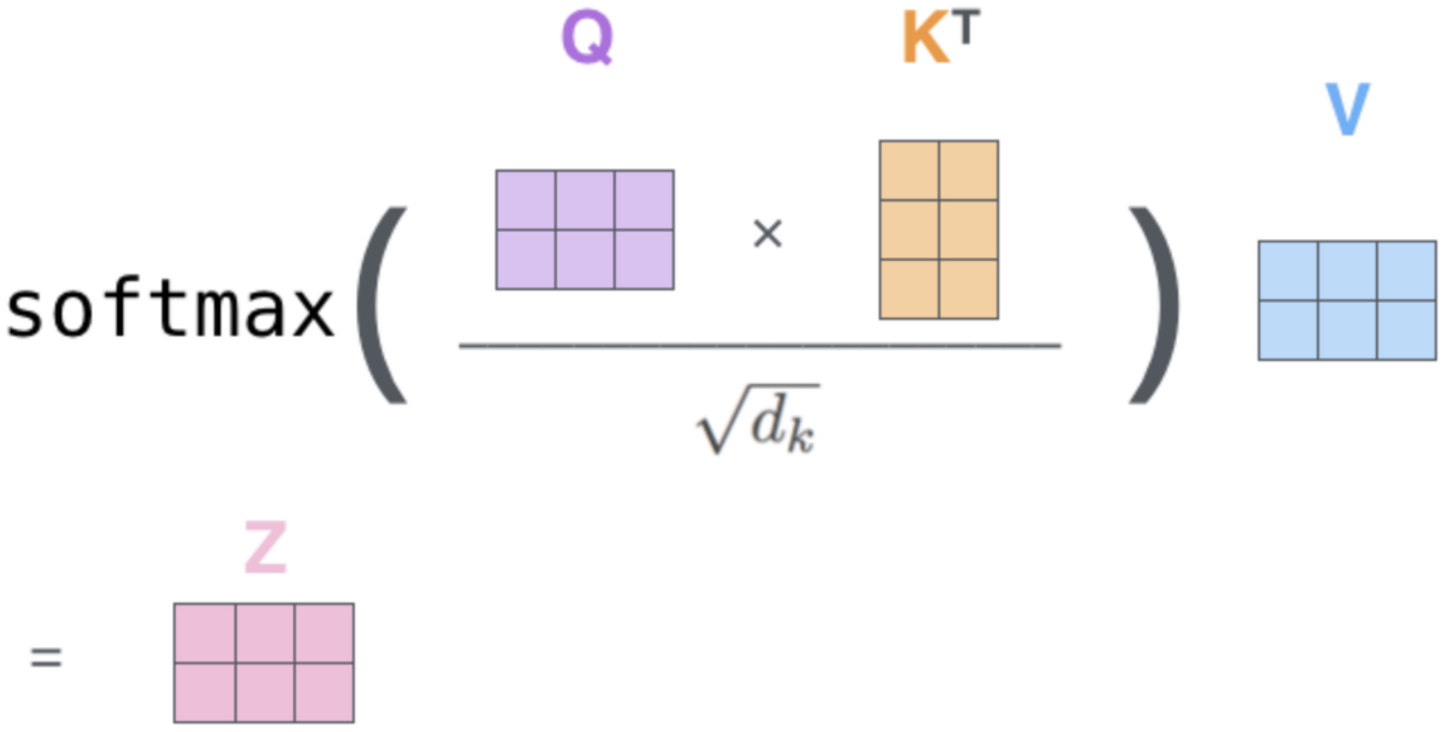
$$x'_i = \sum_{j=1}^{n} w_{ji} x_j$$

- The coefficients w_{ji} are called attention weights and are normalized so that $\sum_j w_{ji} = 1$.

Self-dot-product attention

- There are several ways to implement a self-attention layer, but the most common one is scaled dot-product attention
- There are four main steps
  - Project each token embedding into query, key, and value vectors
  - Compute attention scores: Using a similarity function, determine how much the query and key vectors relate to each other.
    - The similarity function for scaled dot-product attention is the dot product, computed efficiently using matrix multiplication of the embeddings.
    - Similar Queries and keys will have a large dot product, while those that don't share much in common will have little to no overlap.
    - The outputs from this step are called the attention scores, and for a sequence with n input tokens, there is a corresponding n × n matrix of attention scores.
  - Compute attention weights: Dot products generally produce arbitrarily large numbers, which can destabilize the training process.
    - To handle this, the attention scores are first multiplied by a scaling factor to normalize their variance and then normalized with a softmax to ensure all the column values sum to 1.
    - The resulting n × n matrix now contains all the attention weights, w_{ji}.
  - Update the token embeddings: Once the attention weights are computed, multiply them by the value vector $v_1, \ldots, v_n$ to obtain an updated representation for embedding $x'_i = \sum_j w_{ji} v_j$.

Self dot-product attention



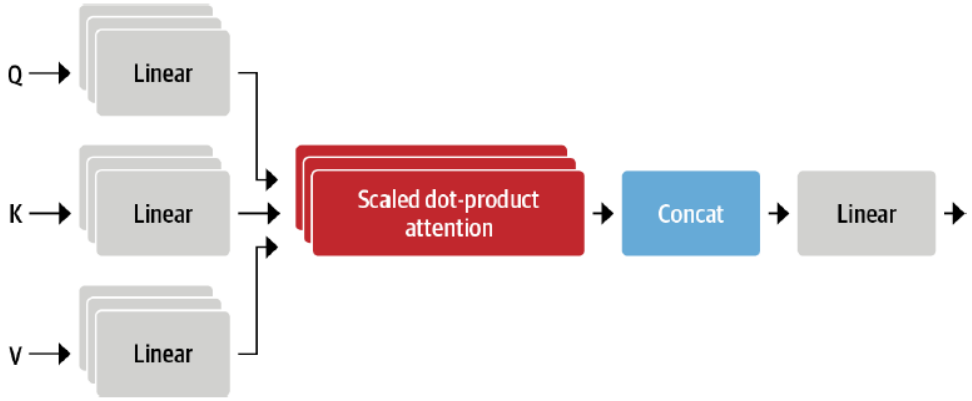
The self-attention calculation in matrix form

Key Query Value

- Let's say we have an input sequence of length N, represented as a matrix X of shape (N, d), where d is the dimension of the input embedding vector. We can calculate the query, key, and value vectors as follows:
  - Query: Multiply the input embedding matrix X (N,d) with a learned weight matrix Wq of shape (d, e), where e is the new embedding size. This produces a matrix Q of shape (N, e), where each column corresponds to the query vector for the corresponding token in the input sequence. $Q = XW_q$
  - Key: Multiply the input embedding matrix X with a learned weight matrix Wk of shape (d, e). This produces a matrix K of shape (N, e), where each column corresponds to the key vector for the corresponding token in the input sequence. $K = XW_k$
  - Value: Multiply the input embedding matrix X with a learned weight matrix Wv of shape (d, e). This produces a matrix V of shape (N, e), where each column corresponds to the value vector for the corresponding token in the input sequence. $V = XW_v$

```
# Stefania Christina (2022) The Attention Mechanism from
Scratch [Source code].
# https://machinelearningmastery.com/the-attention-
mechanism-from-scratch/

from numpy import random
from numpy import dot
from scipy.special import softmax

# encoder representations of four different words
word_1 = array([1, 0, 0])
word_2 = array([0, 1, 0])
word_3 = array([1, 1, 0])
word_4 = array([0, 0, 1])

# stacking the word embeddings into a single array
words = array([word_1, word_2, word_3, word_4])

# generating the weight matrices
random.seed(42)
W_Q = random.randint(3, size=(3, 3))
W_K = random.randint(3, size=(3, 3))
W_V = random.randint(3, size=(3, 3))

# generating the queries, keys and values
Q = words @ W_Q
K = words @ W_K
V = words @ W_V

# scoring the query vectors against all key vectors
scores = Q @ K.transpose()

# computing the weights by a softmax operation
weights = softmax(scores / K.shape[1] ** 0.5, axis=1)

# computing the attention by a weighted sum of the value
vectors
attention = weights @ V

print(attention)
```

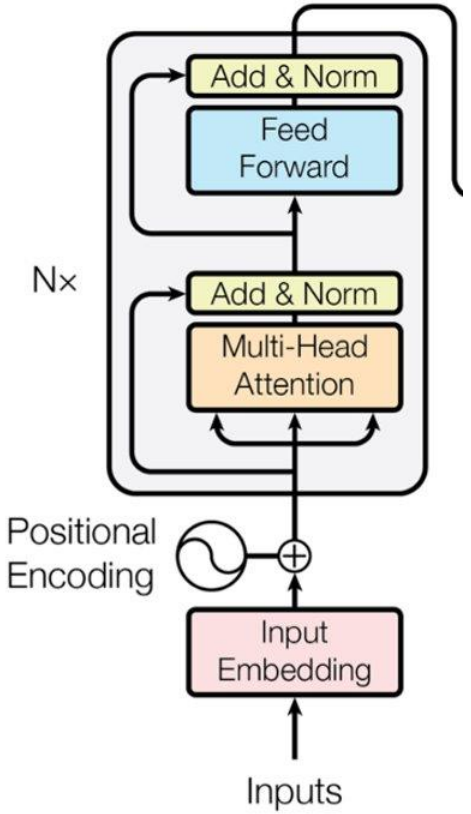
Multi-head Attention



- Transformer uses multiple sets of linear projections, each one representing a so-called attention head.
- Why do we need more than one attention head?
  - Because one head can focus on mostly one aspect of similarity
- Having several heads allows the model to focus on several aspects at once.
- For instance, one head can focus on subject-verb interaction, whereas another finds nearby adjectives.
- We don't handcraft these relations into the model, and they are fully learned from the data.

The Encoder Stack

- The original encoder layer structure remains the same for all N=6 layers of the Transformer model.
- Each layer contains two main sublayers: a multi-headed attention mechanism and a fully connected position-wise feedforward network.
- A residual connection surrounds each sublayer
  - These connections transport the raw input x of a sublayer to the output of the layer to the normalization function
  - The normalized output of each layer is thus:
    - *LayerNormalization (x + Sublayer(x))*
- Though the structure of each of the N=6 layers of the encoder is identical, the content of each layer is not identical to the previous layer.
  - Because the weights of the layers are different
  - Each layer augments the embedding with more context

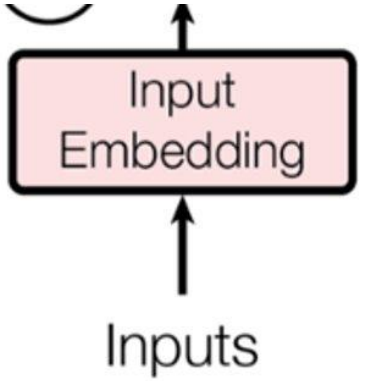


The Encoder Stack

- The multi-head attention mechanisms perform the same functions from layer 1 to 6.
- But these layers do not learn the same things.
- Each layer adds more contextual knowledge to the tokens/words it receives from the previous layer by exploring different ways of adding context to the tokens
  - It looks for various associations of words, just like we look for different associations of letters and words when we solve a crossword puzzle.
- The output of every sublayer of the model has a constant dimension, including the embedding layer and the residual connections
- This dimension is d_{model} and can be set to another value depending on the goals. In the original Transformer architecture, $d_{model} = 512$
  - d_{model} has a powerful consequence
    - It can improve the performance of the model
- We can add as many layers as we want or have computing resources available

Input Embedding

- The input embedding sublayer converts the input tokens to vectors of dimension d_{model} using learned embeddings from another model (transformer/word2vec/one-hot model)
- The embedding sublayer works on the tokens that the transformer will find embeddings
  - Tokenizer could be BPE, word piece, and sentence piece etc.
- A tokenizer provides an integer representation that will be used for the embedding process. For example:

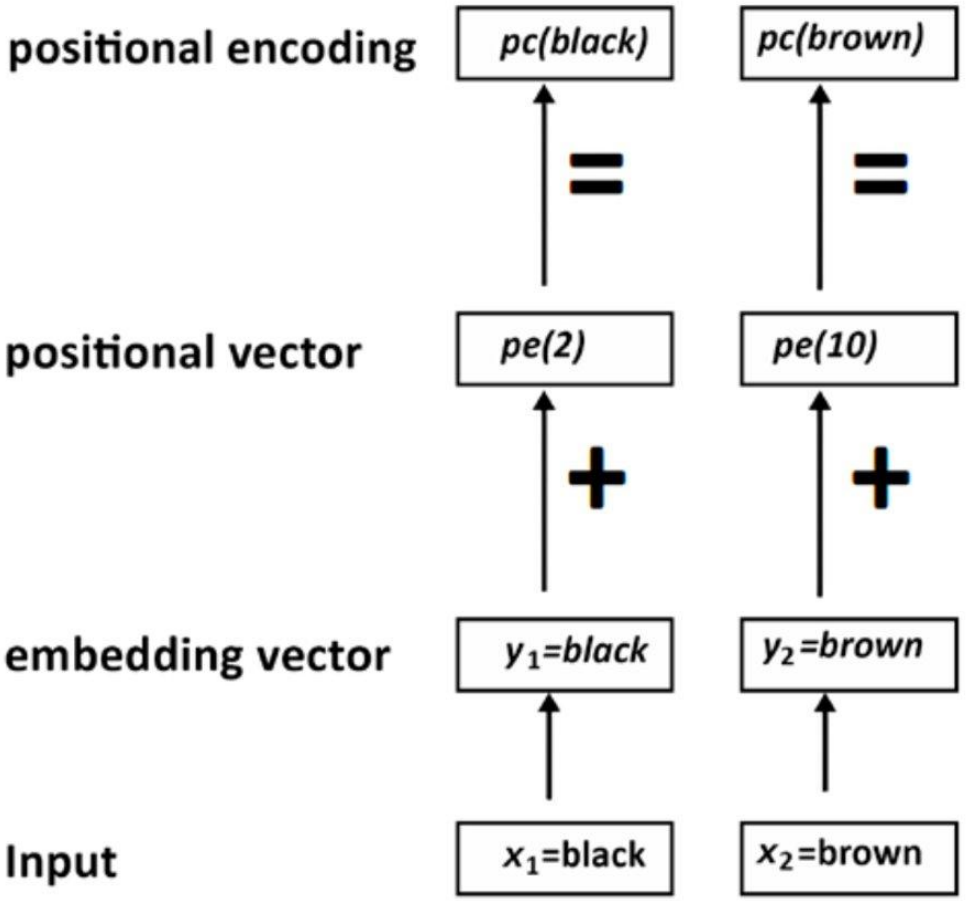



```
text = "The cat slept on the couch.It was too tired to get up."
tokenized text= [1996, 4937, 7771, 2006, 1996, 6411, 1012, 2009, 2001,
2205, 5458, 2000, 2131, 2039, 1012]
```

Positional Encoding

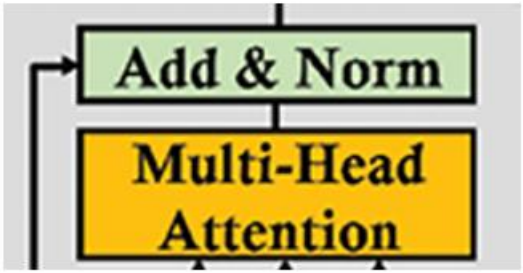
- The self-attention calculation method does not consider position of the tokens
  - Therefore it does not have any idea of the relative word/token orders
- Positional embeddings takes care of this shortcoming
  - The most basic embedding is simple: augment the token embeddings with a position-dependent pattern of values arranged in a vector
    - If the pattern is characteristic for each position, the attention heads and feed-forward layers in each stack can learn to incorporate positional information into their transformations
- It is possible to learn the pattern, especially when the pretraining dataset is sufficiently large
  - This works exactly the same way as the token embeddings, but using the position index instead of the token ID as input
- Absolute positional representations: Transformer models can use static patterns consisting of modulated sine and cosine signals to encode the positions of the tokens.
  - This works especially well if the dataset is relatively small.

Positional Encoding



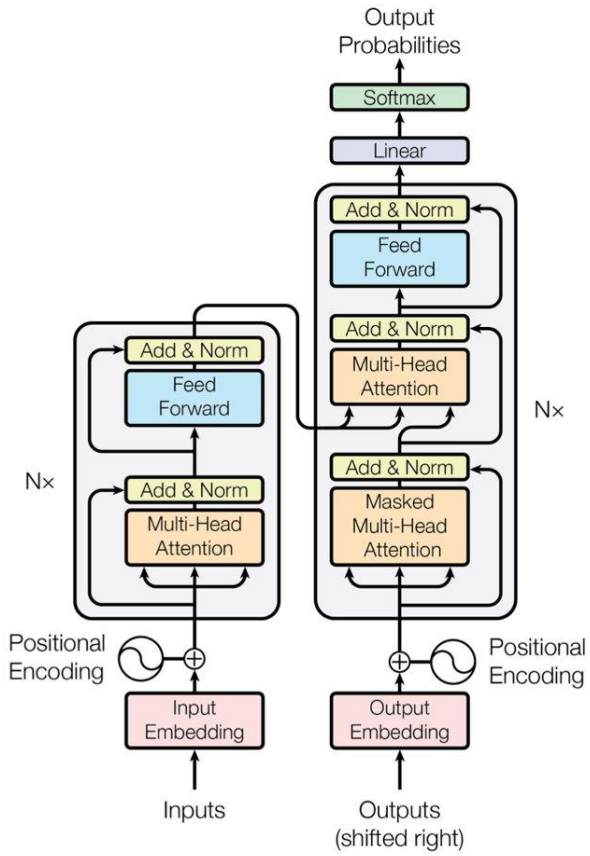
Layer Normalization

- All sublayers (attention and feedforward) of the Transformer is followed by Post-Layer Normalization
- The Post-LN contains an add function and a layer normalization process
- The add function processes the residual connections that come from the input of the sublayer and are normalized
  - The goal of the residual connections is to make sure critical information is not lost



Decoder Layer

- The structure of the decoder layer remains the same similar to the encoder for all the N = 6 layers of the original Transformer model
- Each layer contains three sublayers:
  - A multi-headed masked attention mechanism
  - A multi-headed attention mechanism
  - A fully connected position-wise feedforward network.
- The decoder has a different sublayer (than encoder), which is the masked multi-head attention mechanism
- In this sublayer; inputs are partially masked so that the Transformer can predict the next token on its own without looking into the rest of the sequence
- Similar to the encoder, residual connection, Sublayer(x), surrounds each of the three main sublayers to ensure proper flow of gradients



Decoder Layer

- The output embedding sublayer is only present at the bottom level of the stack, like for the encoder stack
- The output of every sublayer of the decoder stack has a constant dimension, d_{model}, like in the encoder stack, including the embedding layer and the output of the residual connections.
- We can see that the design is symmetrical: decoder decodes a sequence from the encoded information
- The structure of each sublayer and function of the decoder is similar to the encoder