Lecture 13: Transformer and Attention

Yen-Yi Ho

Department of Statistics

Outline

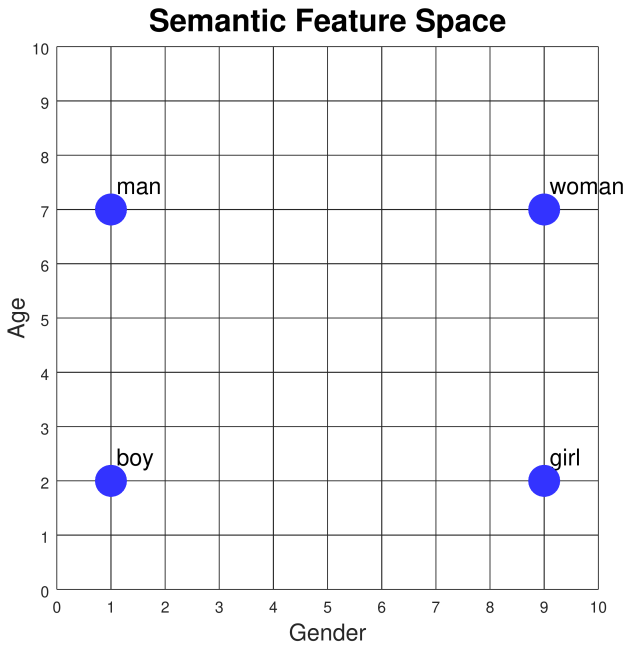
- Word Embedding
- Self-Attention
- Multi-Head Self-Attention
- Transformer Models
- Examples

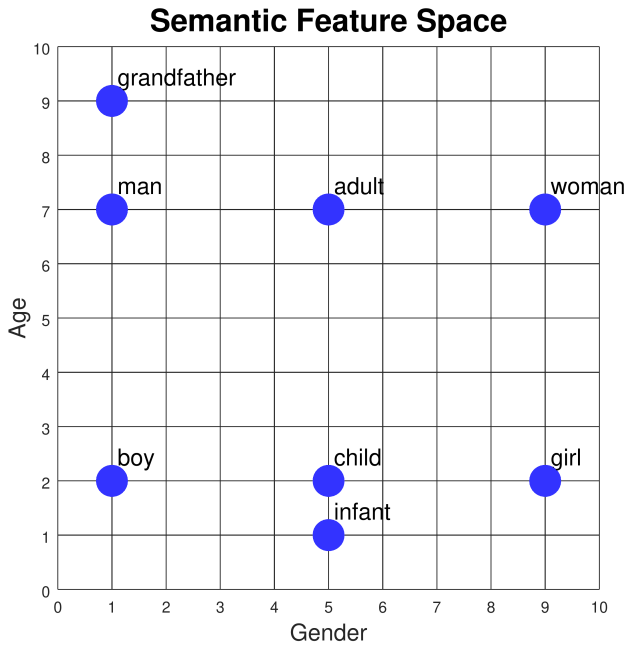
Word Embedding

A word embedding is a way to represent words as numerical vectors (lists of numbers) so that computers can understand and process language more effectively.
Instead of treating words as simple labels (like IDs), embeddings map words into a continuous vector space where:

- Similar words have similar vectors
- Relationships between words are captured mathematically
- Meaning is encoded in patterns of numbers

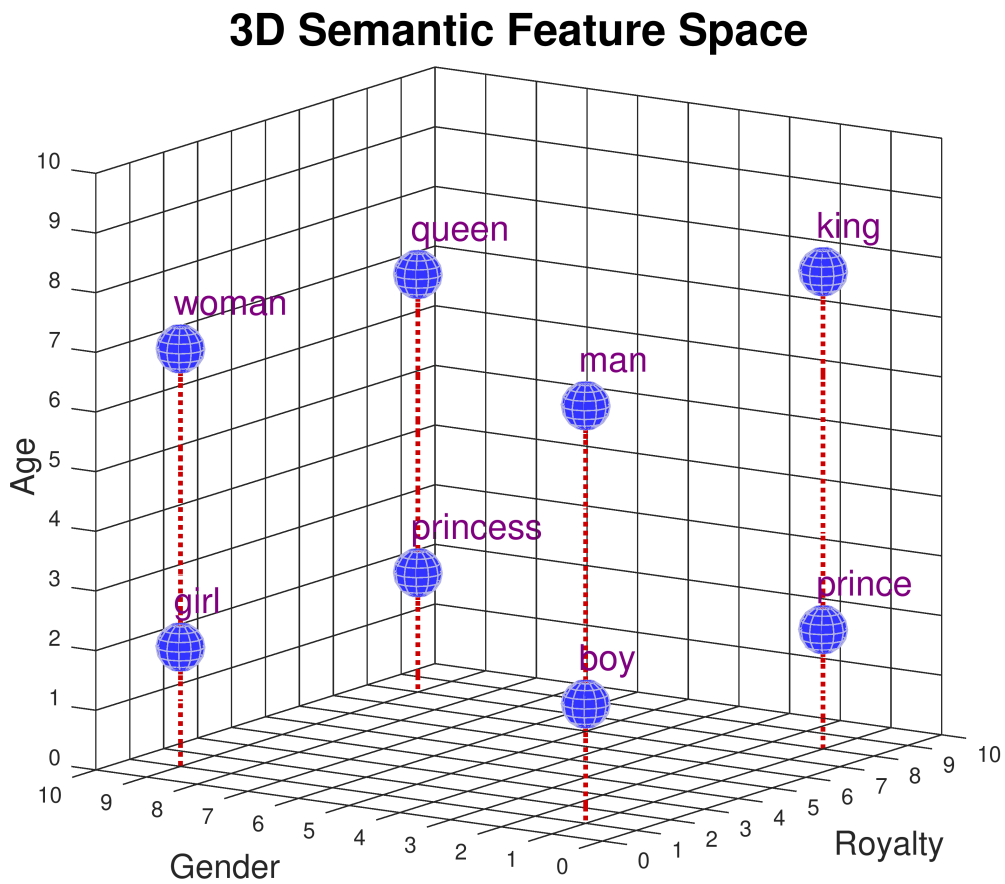
Word Embedding





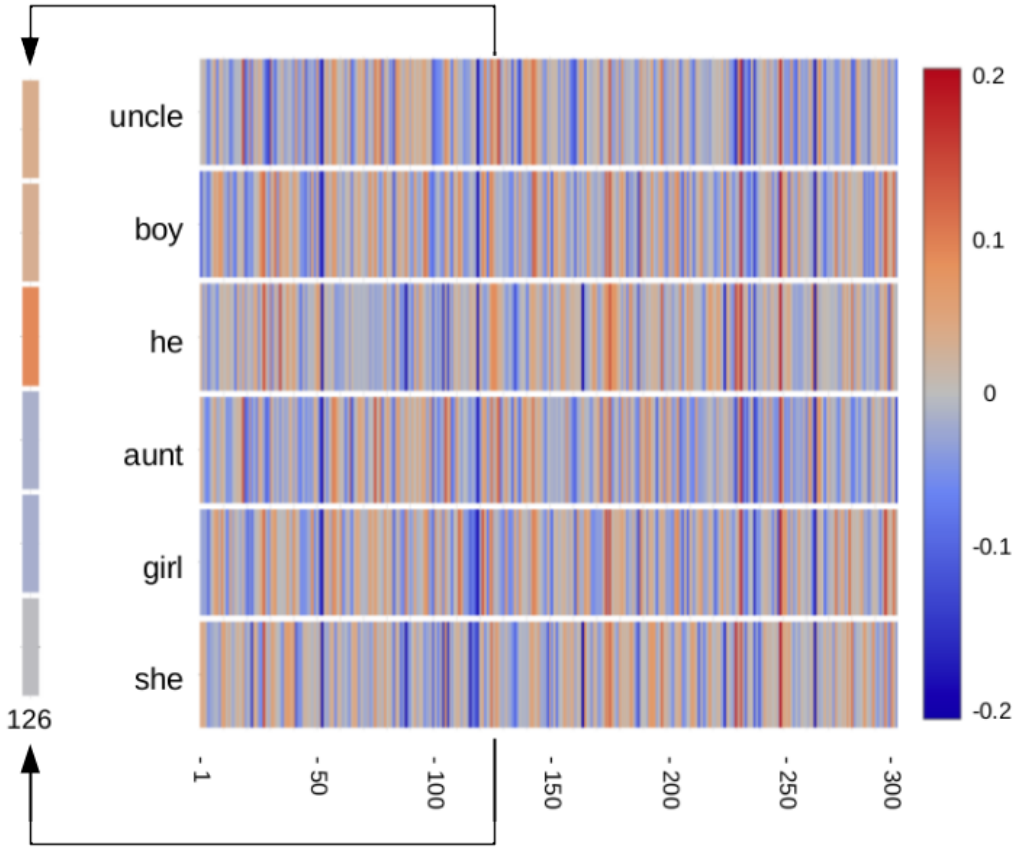
figures adapted from `https://www.cs.cmu.edu/~dst/WordEmbeddingDemo/tutorial.html`

Word Embedding

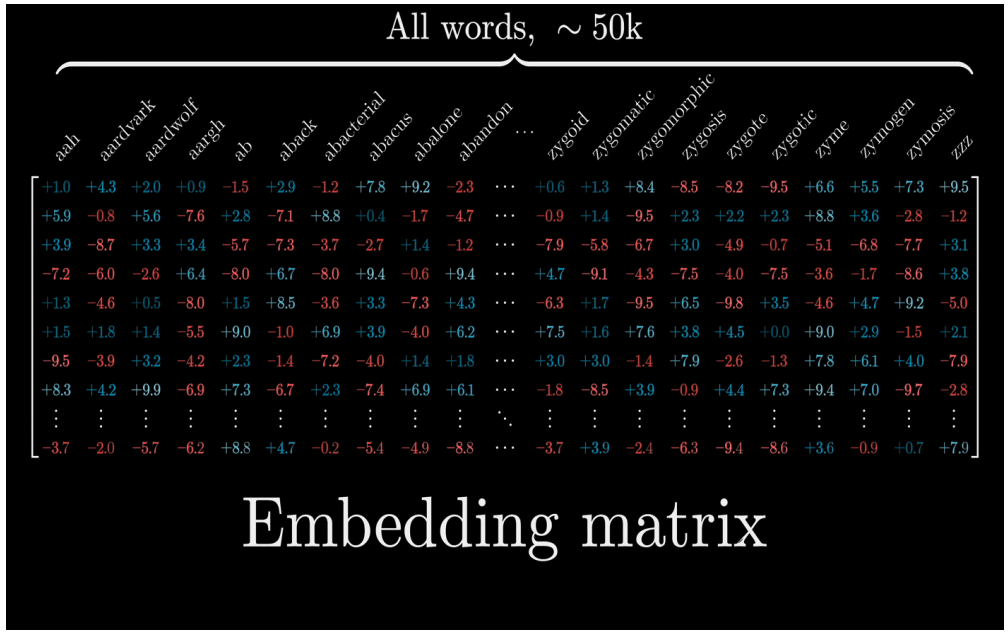


Word Embedding

A 300-dimensional embedding is constructed from word co-occurrence statistics derived from extensive text corpora, such as the entire Wikipedia or large collections of news articles.

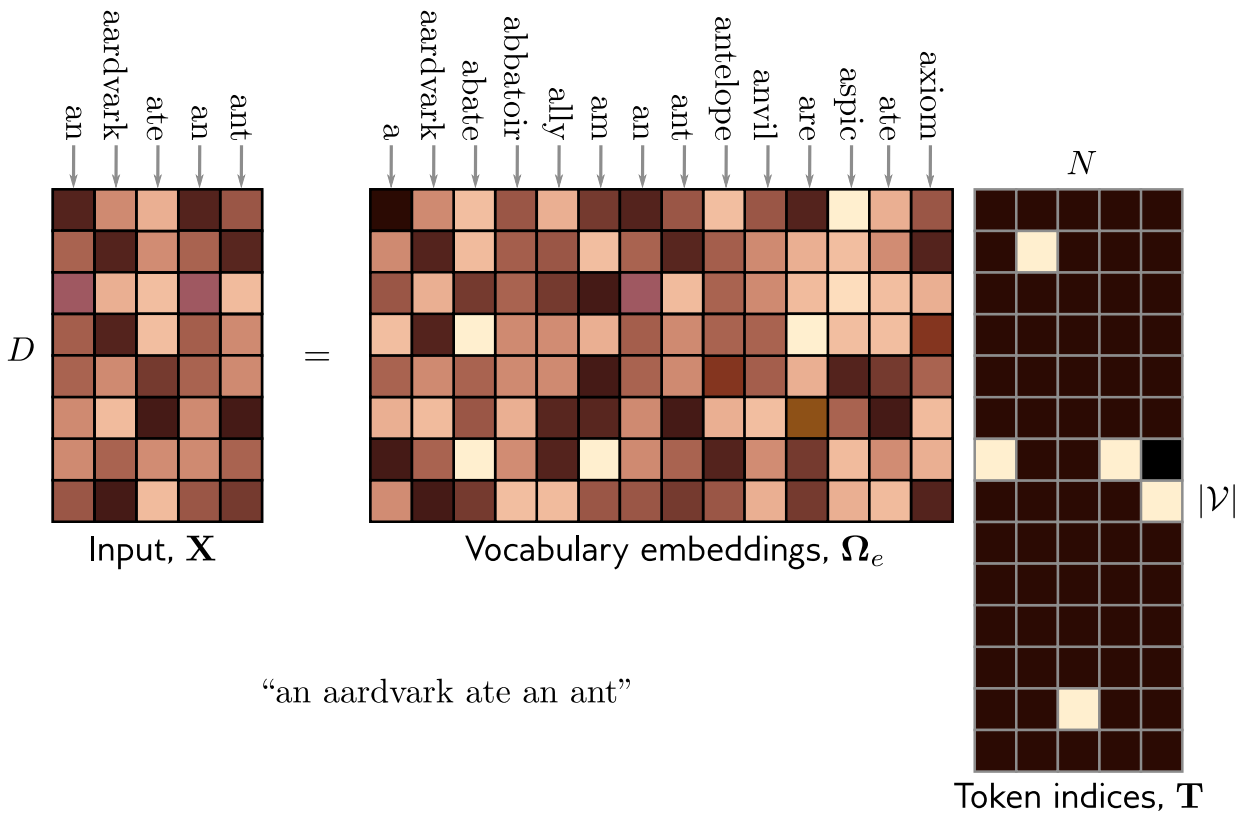


Word Embedding



Word Embedding

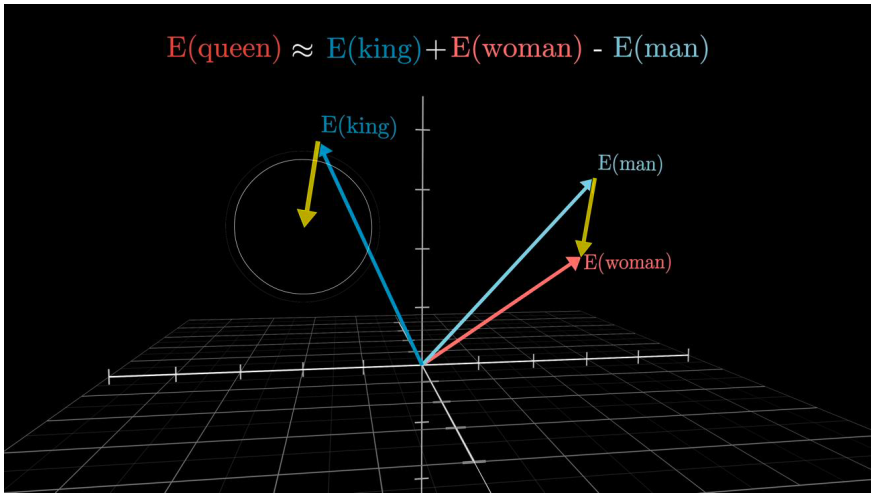
An aardvark ate an ant

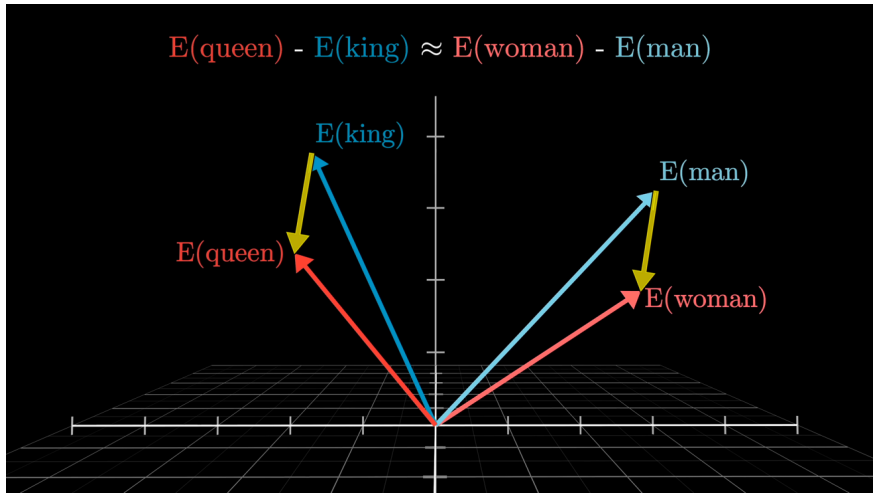


"an aardvark ate an ant"

Word Embedding

- In this embedding space, the directions of word vectors correspond to specific semantic relationships.
- Finding a female monarch by taking "king", adding the difference "woman"-"man" and search for such an embedding.





https://www.youtube.com/watch?v=wjZofJX0v4M

Tokenization

- Names
- Punctuation
- walk, walks, walked, walking

Breaks text down into smaller units—such as words or subwords (morphemes)—to help large language models operate more efficiently. Common methods include Byte Pair Encoding and WordPiece tokenization.

Self-Attention

"The animal didn't cross the street because it was too tired."

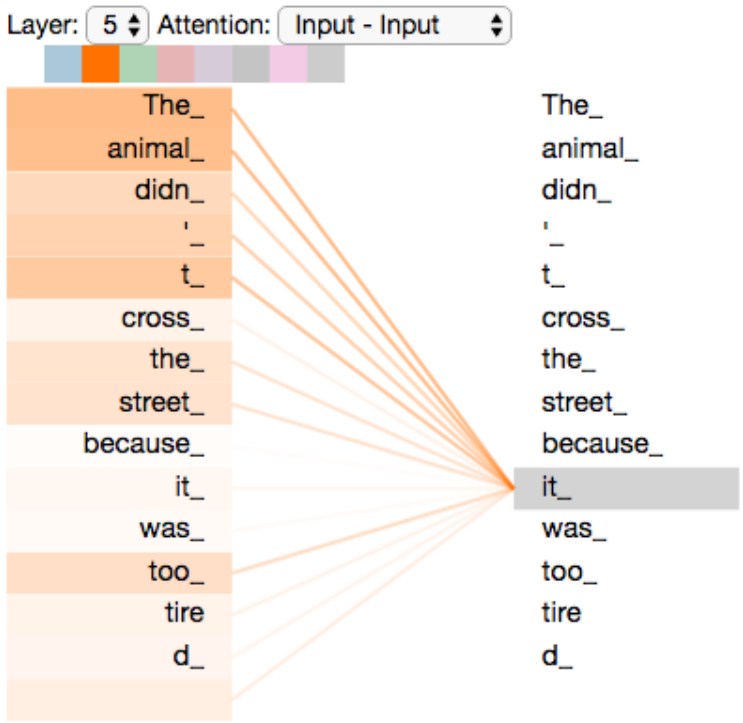
What does "it" in this sentence refer to?

Is it referring to the street or to the animal? It's a simple question to a human, but not as simple to an algorithm.

The self-attention mechanism enables the model to associate "it" with "animal".

Self-Attenton

As the model processes each word (each position in the input sequence), self attention allows it to look at other positions in the input sequence for clues that can help lead to a better encoding for this word.



Query, Key and Value

In self-attention, each token is projected into three different representations:

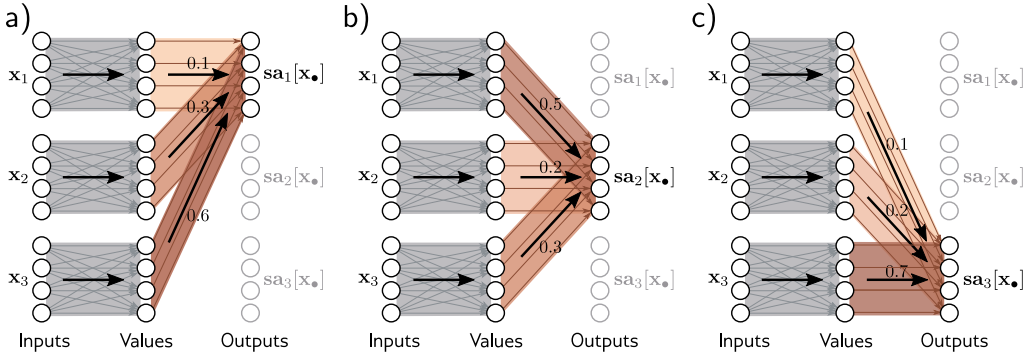
- a query (what information am I looking for?),
- a key (what information do I contain?), and
- a value (what information should I contribute?).

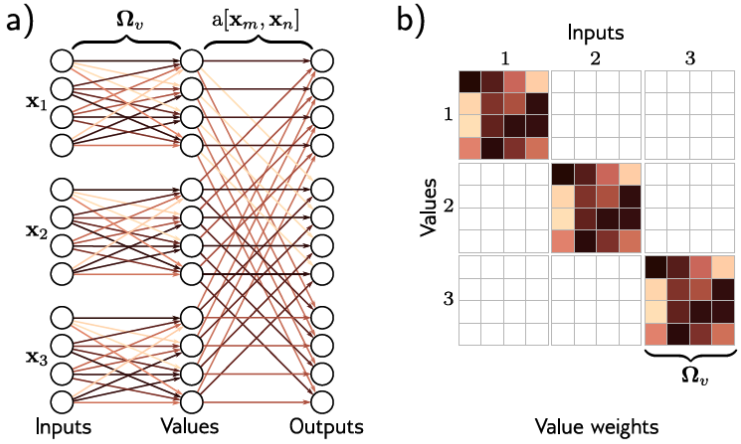
Attention weights are computed by matching queries to keys, then used to aggregate values.

Self Attention

- Self-attention: Every token attends to every other token, routing their values in varying proportions to produce each output.
- A self-attention block takes n inputs, each of dimension $d \times 1$, and returns n output vectors of the same size.
- Given a sequence of input tokens $x_1, ..., x_n$ where any $x_i \in \mathbb{R}^d$, its self attention outputs a sequence of same length $y_1, .., y_n \in \mathbb{R}^d$ (d=512).

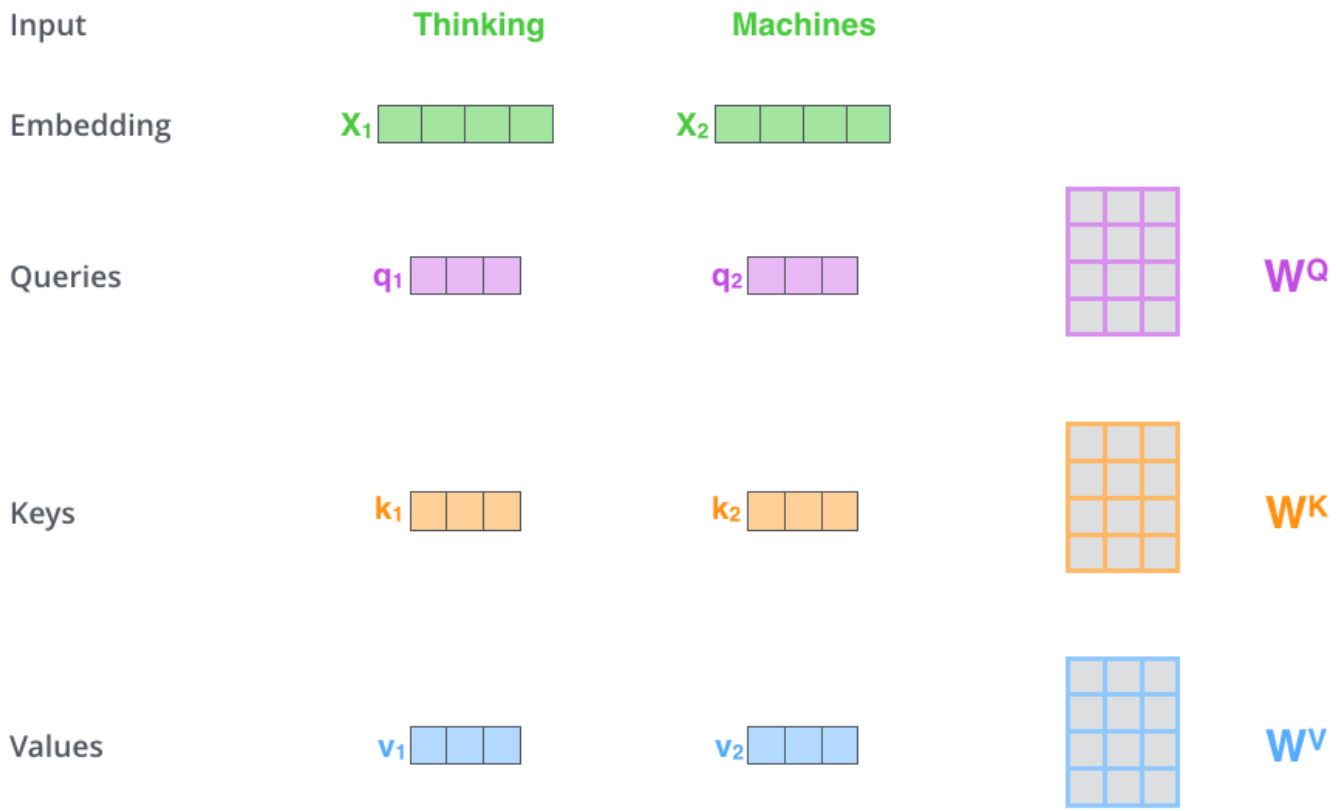
$$
\begin{aligned}
v_i &= \beta_v + \omega_v x_i \\
y_i &= \Sigma_j a(q_i, k_j) v_i
\end{aligned}
$$





Self-Attention: Details

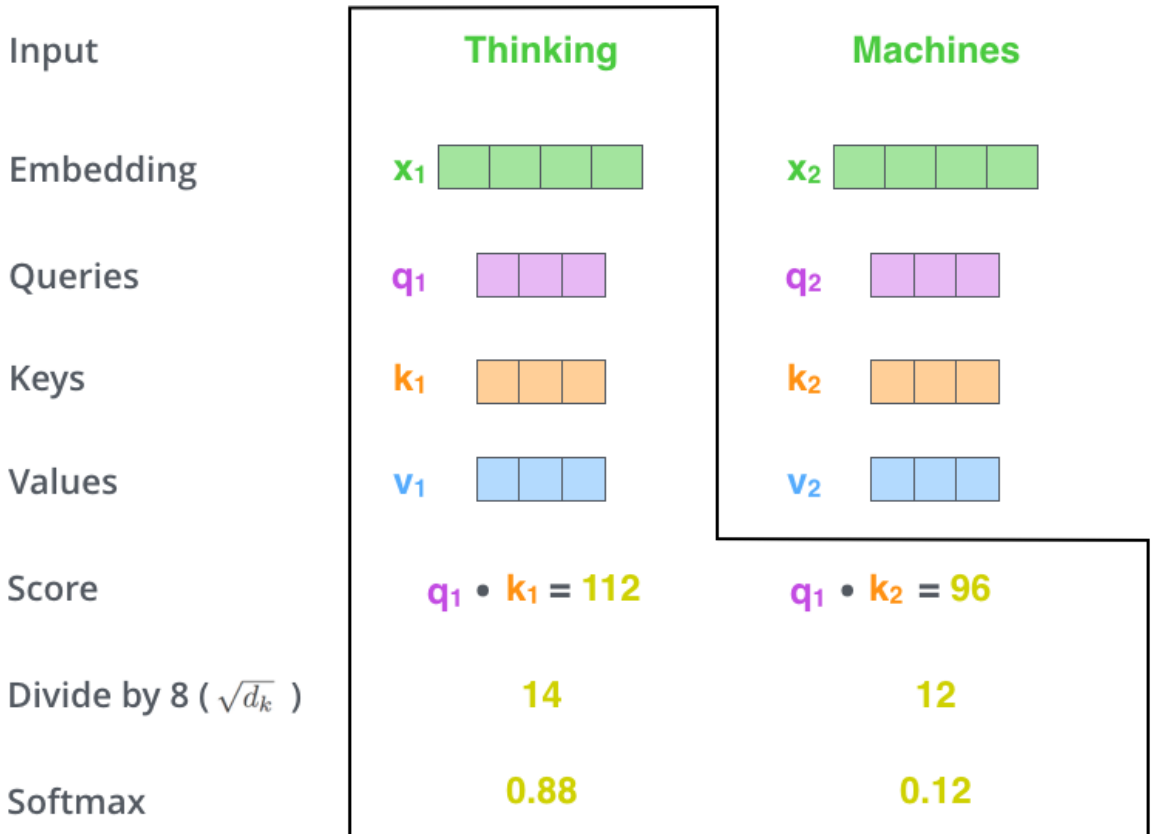
- The first step is to create three vectors from each of encoder's input vectors. These vectors are created by multiplying the embedding by three matrices that we trained during the training process (usually smaller in dimension than the embedding vector).



Figures adapted from `https://jalammar.github.io/illustrated-transformer/`

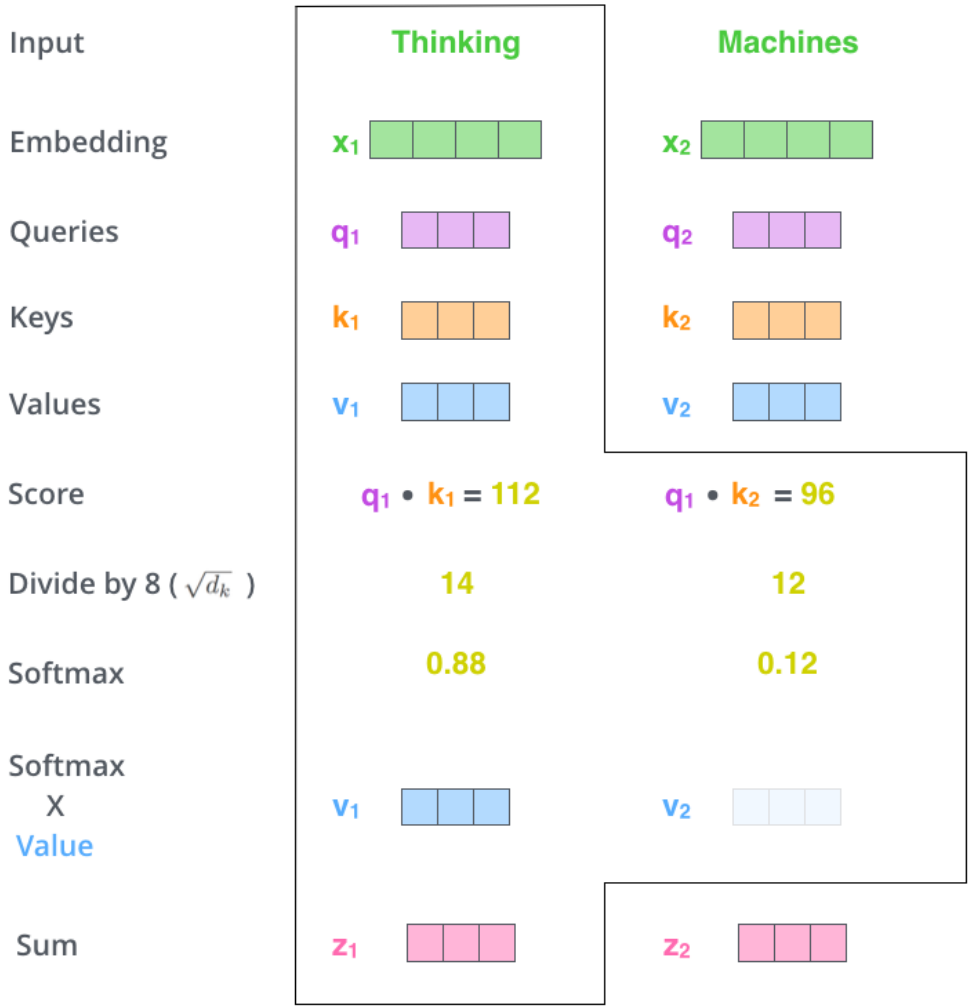
Self-Attention: Details

- The second step is to calculate the scoring function and then divide it by the square root of the dimension of the key vectors.



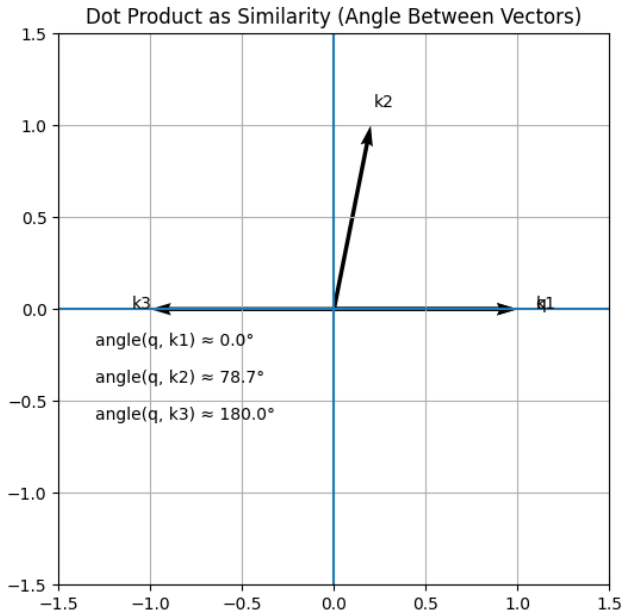
Self-Attention: Details

- The third step is to multiply each value vector by the softmax score and sum up the weighted value vectors.



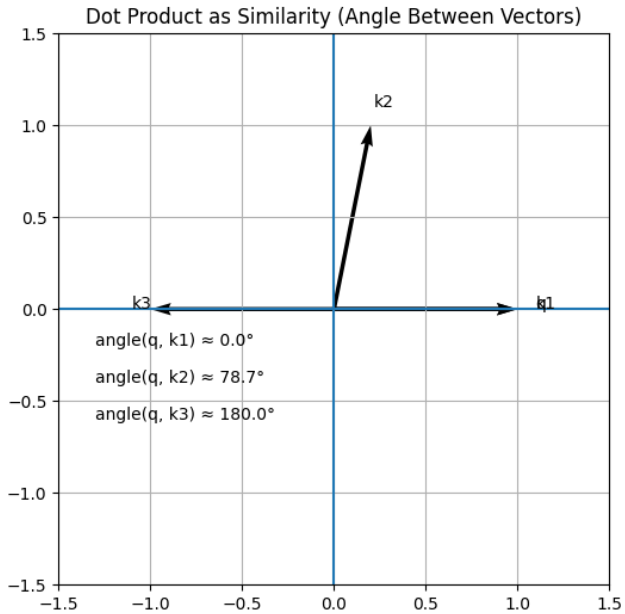
Dot Product:Geometric Interpretation

- q=query (what we are looking for)
- k_1, k_2, k_3= keys (possible matches)



Dot Product:Geometric Interpretation

- k_1 (0°): same direction, maximum similarity, high attention
- k_2 ($\approx 79^\circ$): partially aligned, moderate attention
- k_3 (180°): negative similarity, low attention

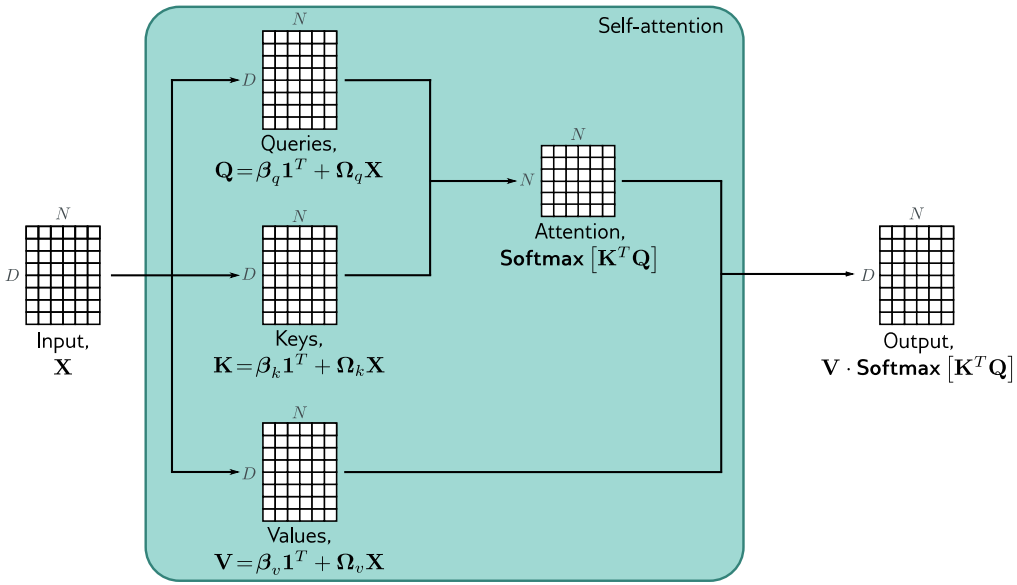


Attention is high when vectors point in the same direction — dot product measures this alignment.

Self-Attention: Matrix Calculation

- Query, key and value matrices are calculated through matrix multiplication.

$$
\begin{aligned}
V(X) &= \beta_v \mathbf{1}^T + \Omega_v X \\
Q(X) &= \beta_q \mathbf{1}^T + \Omega_q X \\
K(X) &= \beta_k \mathbf{1}^T + \Omega_k X \\
Y = Sa(X) &= V(X) \cdot \text{Softmax}[K(X)^T Q(x)]
\end{aligned}
$$



Self-Attention Summary

- Is the self-attention mechanism non-liner?
- A single shared set of parameters $\phi = \beta_v, \Omega_v, \beta_q, \Omega_q, \beta_k, \Omega_k$ that is **independent** of the number of input N.

For simplicity,

$$Sa(X) = V \cdot \text{Softmax}[K^T Q]$$

Self-Attention Implementation

$$Sa(X) = V \cdot \text{Softmax}[K^T Q]$$

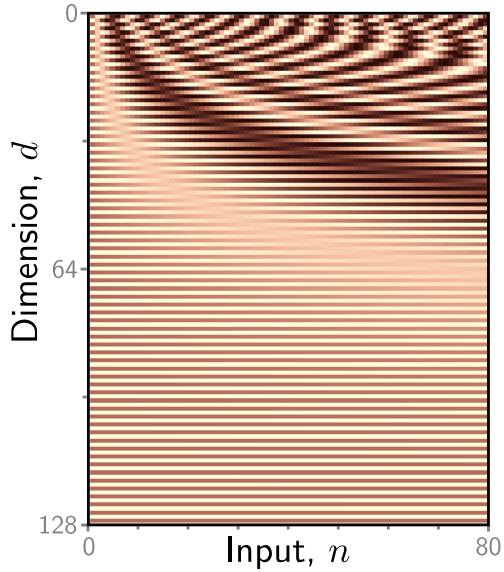
```python
class DotProductAttention(nn.Module):
    """Scaled dot product attention."""
    def __init__(self, dropout):
        super().__init__()
        self.dropout = nn.Dropout(dropout)

    # Shape of queries: (batch_size, no. of queries, d)
    # Shape of keys: (batch_size, no. of key-value pairs, d)
    # Shape of values: (batch_size, no. of key-value pairs, value dimension)
    # Shape of valid_lens: (batch_size,) or (batch_size, no. of queries)
    def forward(self, queries, keys, values, valid_lens=None):
        d = queries.shape[-1]
        # Swap the last two dimensions of keys with keys.transpose(1, 2)
        scores = torch.bmm(queries, keys.transpose(1, 2)) / math.sqrt(d)
        self.attention_weights = masked_softmax(scores, valid_lens)
        return torch.bmm(self.dropout(self.attention_weights), values)
```

Self-Attention: Extensions

- Positional encoding

  The woman ate the raccoon.
  The raccoon ate the woman.

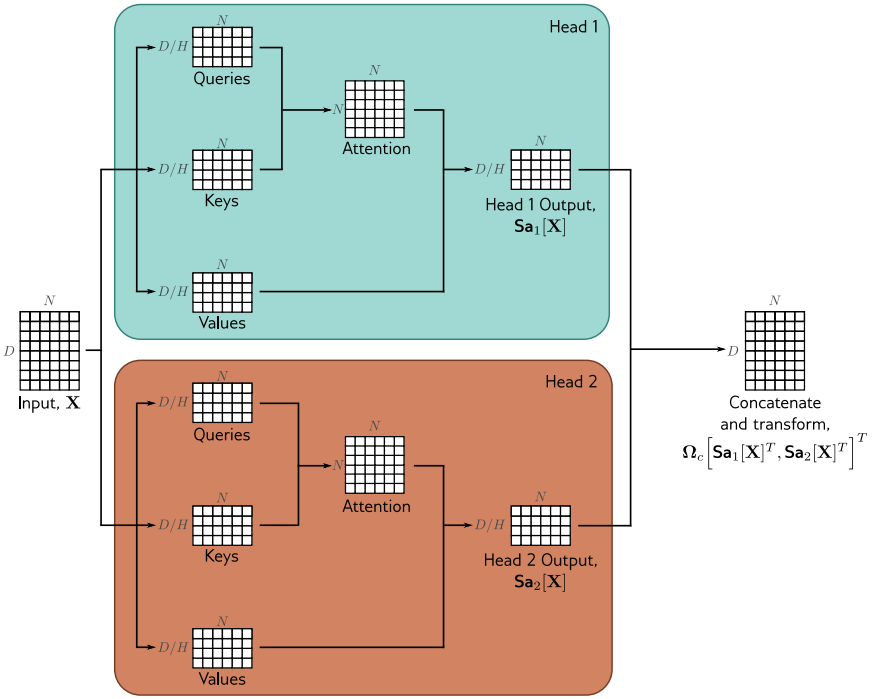
- Multi-head self-attention



Multi-Head Self-Attention

- H different sets of values, keys, and queries are computed.

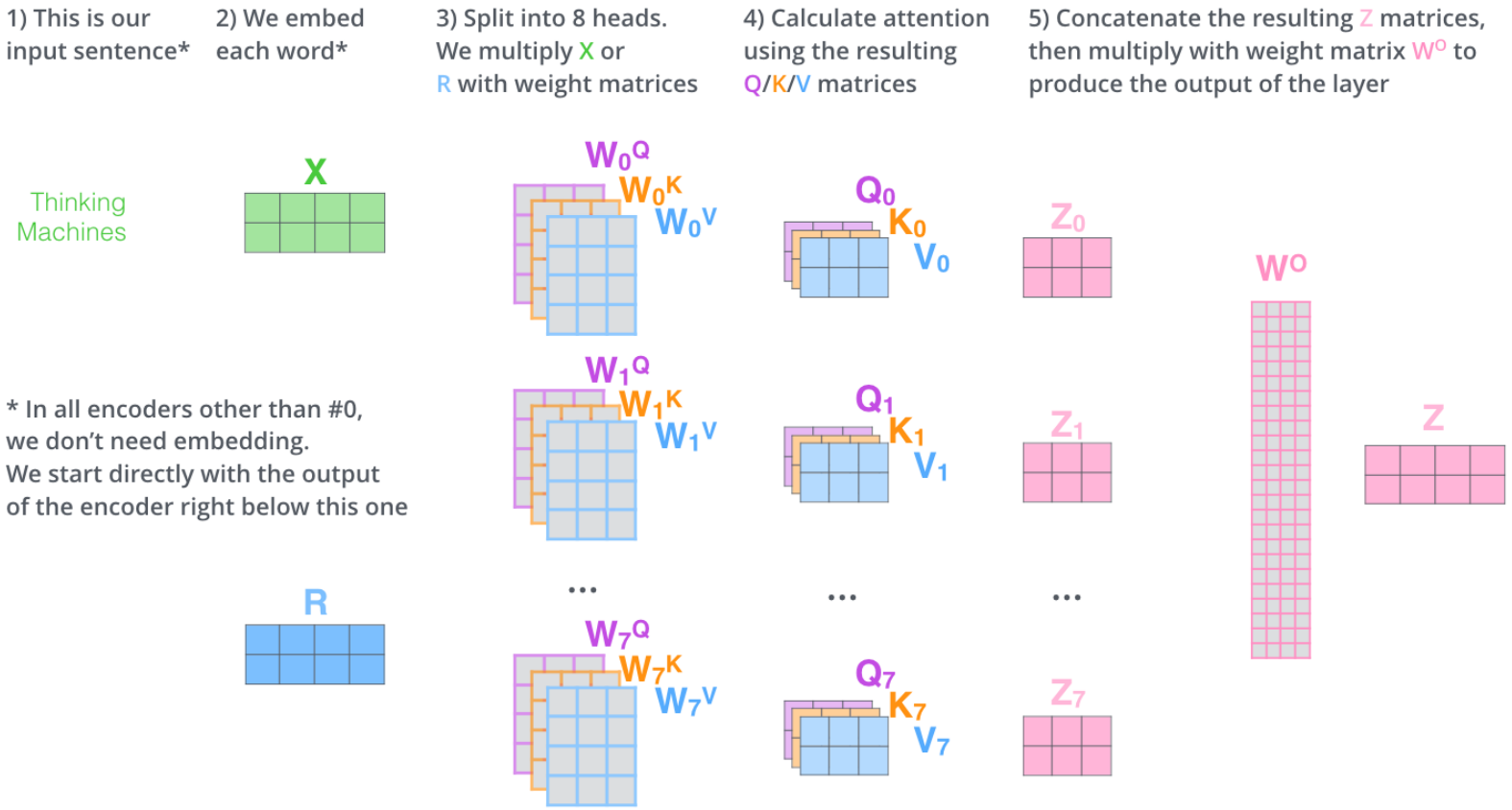
$$
\begin{aligned}
V_h &= \beta_{vh}\mathbf{1}^T + \Omega_{vh}X \\
Q_h &= \beta_{qh}\mathbf{1}^T + \Omega_{qh}X \\
K_h &= \beta_{kh}\mathbf{1}^T + \Omega_{kh}X \\
Sa_h(X) &= V_h \cdot \text{Softmax}[\frac{K_h^T Q_h}{\sqrt{Dq}}]
\end{aligned}
$$



$$MhSa(x) = \Omega_c[Sa_1(X)^T, Sa_2(X)^T, ..., Sa_H(X)^T]^T$$

Multi-Head Self-Attention

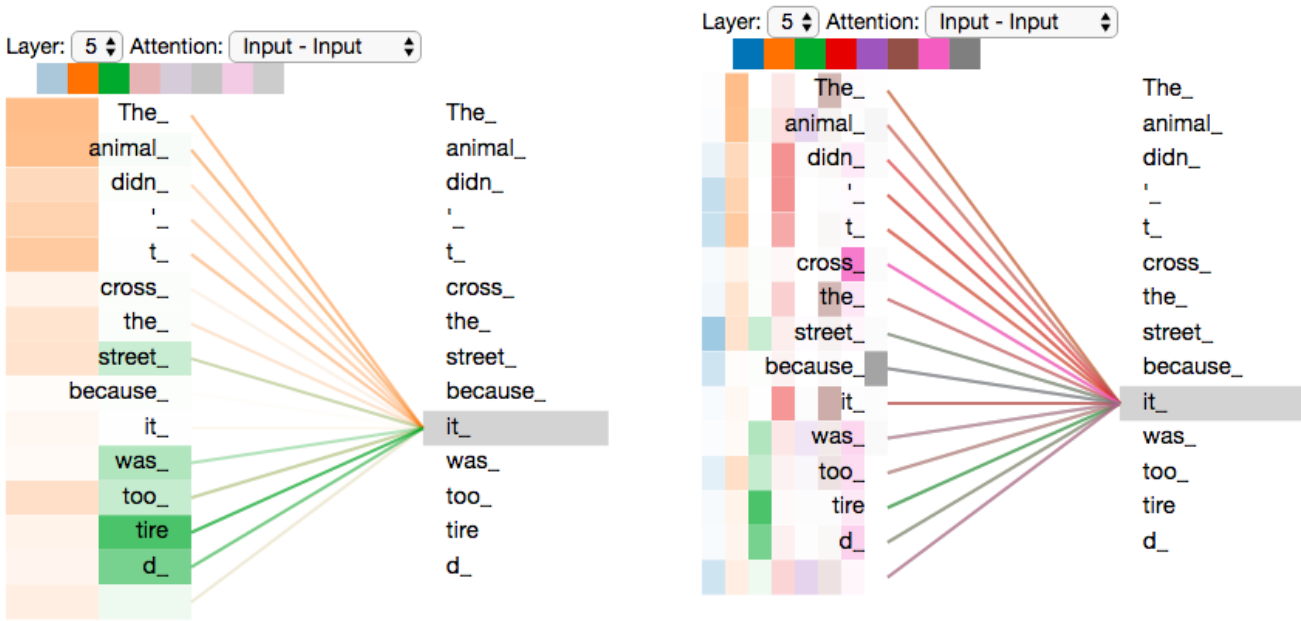
- Multi-head attention expands the model's ability to focus on different positions, as well as gives the attention layer multiple "representation subspaces", thus improving the expressivity of the model



Figures adapted from `https://jalammar.github.io/illustrated-transformer/`

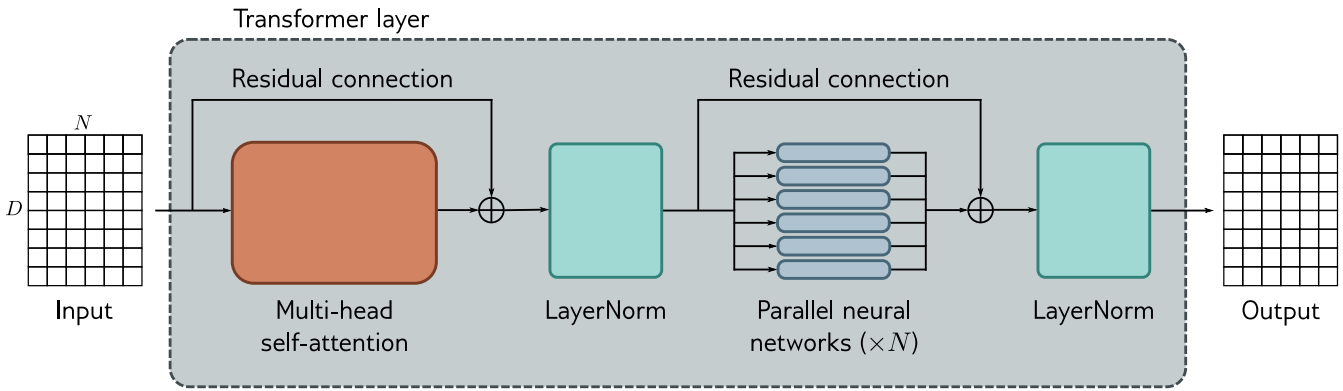
Multi-Head Self-Attention

- As we encode the word "it", one attention head is focusing most on "the animal", while another is focusing on "tired" – in a sense, the model's representation of the word "it" bakes in some of the representation of both "animal" and "tired".
- However, once all attention heads are to be considered, things can get started to be harder for interpretation.



Figures adapted from `https://jalammar.github.io/illustrated-transformer/`

Transformer Layers

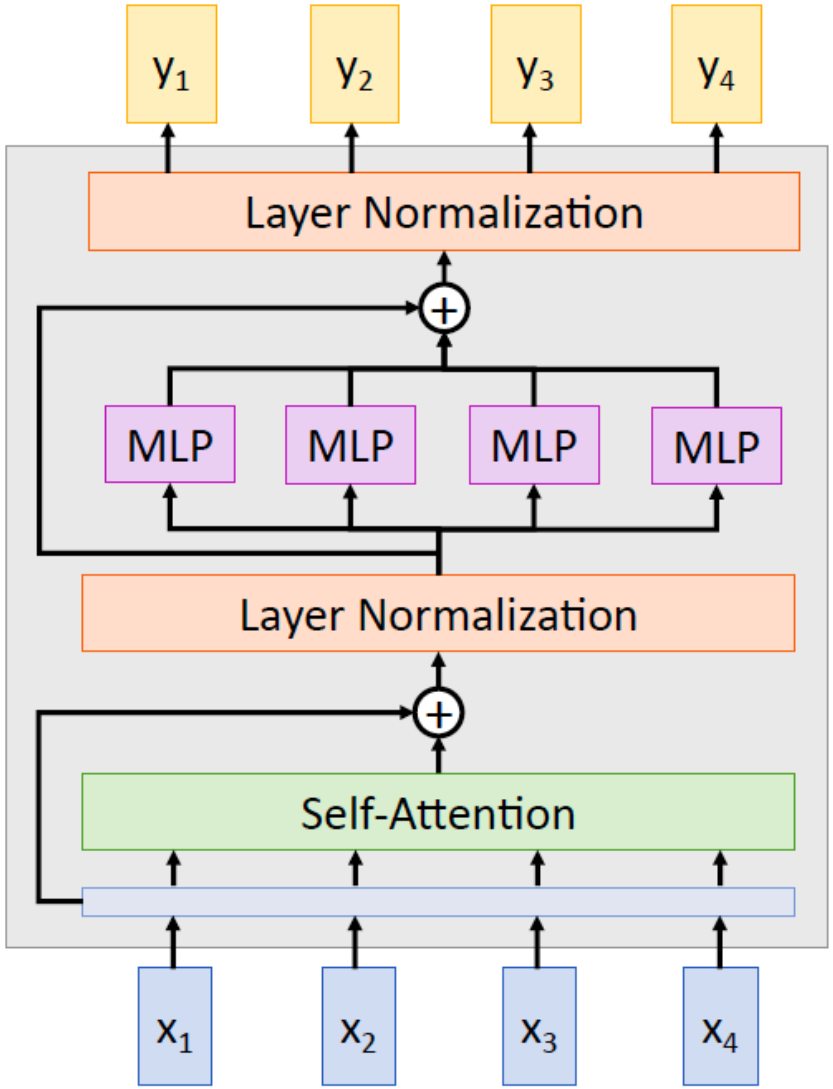


$$
\begin{aligned}
\mathbf{X} &\leftarrow \mathbf{X} + \mathbf{MhSa}[\mathbf{X}] \\
\mathbf{X} &\leftarrow \mathbf{LayerNorm}[\mathbf{X}] \\
\mathbf{x}_n &\leftarrow \mathbf{x}_n + \mathbf{mlp}[\mathbf{x}_n] \\
\mathbf{X} &\leftarrow \mathbf{LayerNorm}[\mathbf{X}], \qquad \forall\, n \in \{1, \ldots, N\}
\end{aligned}
$$

- Layer Norm: apply to each column (token) independently.
- MLP: Same MLP applied to each column.
- Residual connection: prevent network from "forgetting" or distorting important information.

$$x_l = f(x_{l-1}) + x_{l-1}$$

Putting it All Together



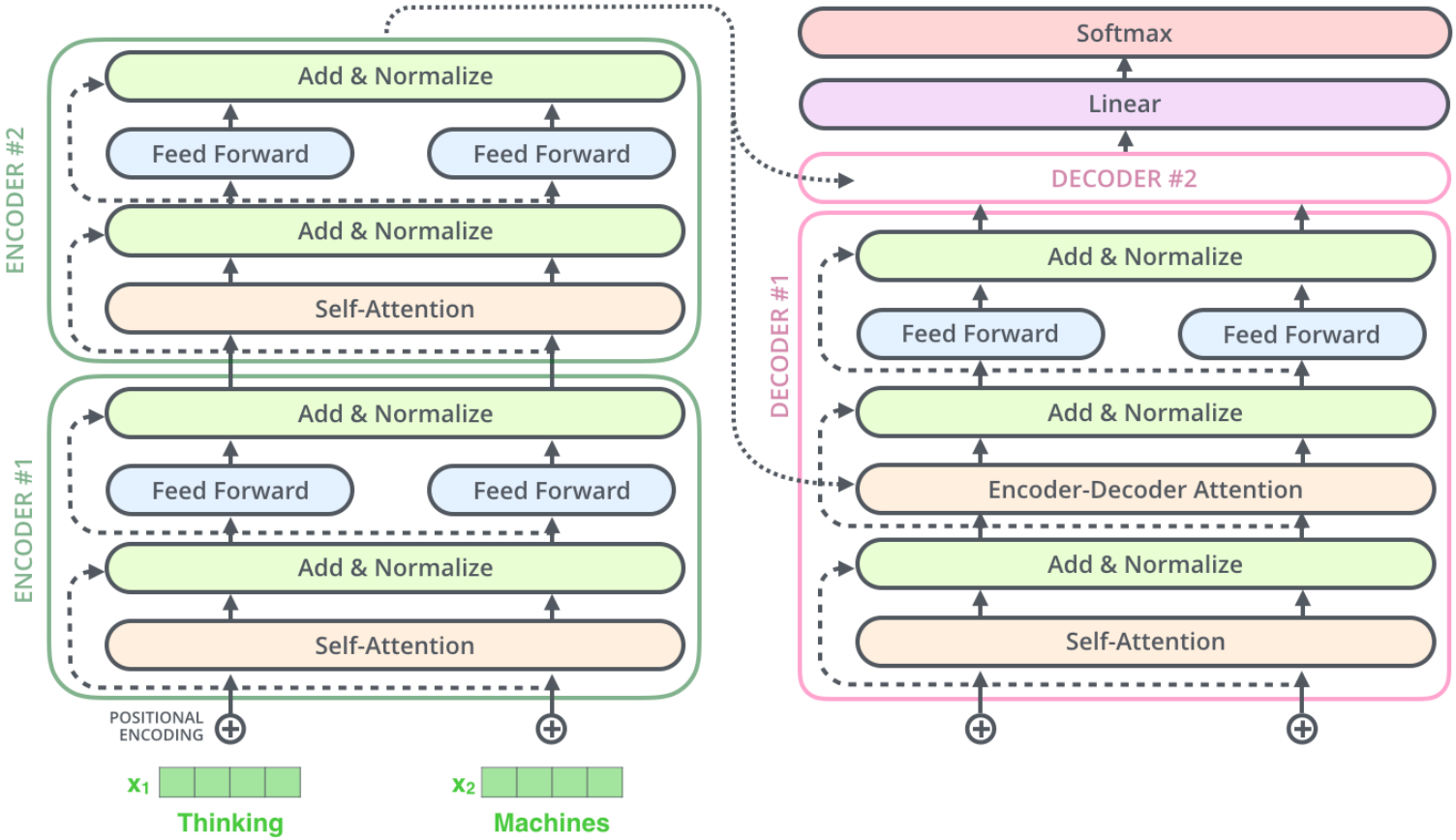
Transformer Models

A Transformer is a sequence of transformer blocks

- Encoder
- Decoder
- Encoder-Decoder

Transformer Models

- The encoder outputs a contextualized representation for each token, which serves as input to the decoder

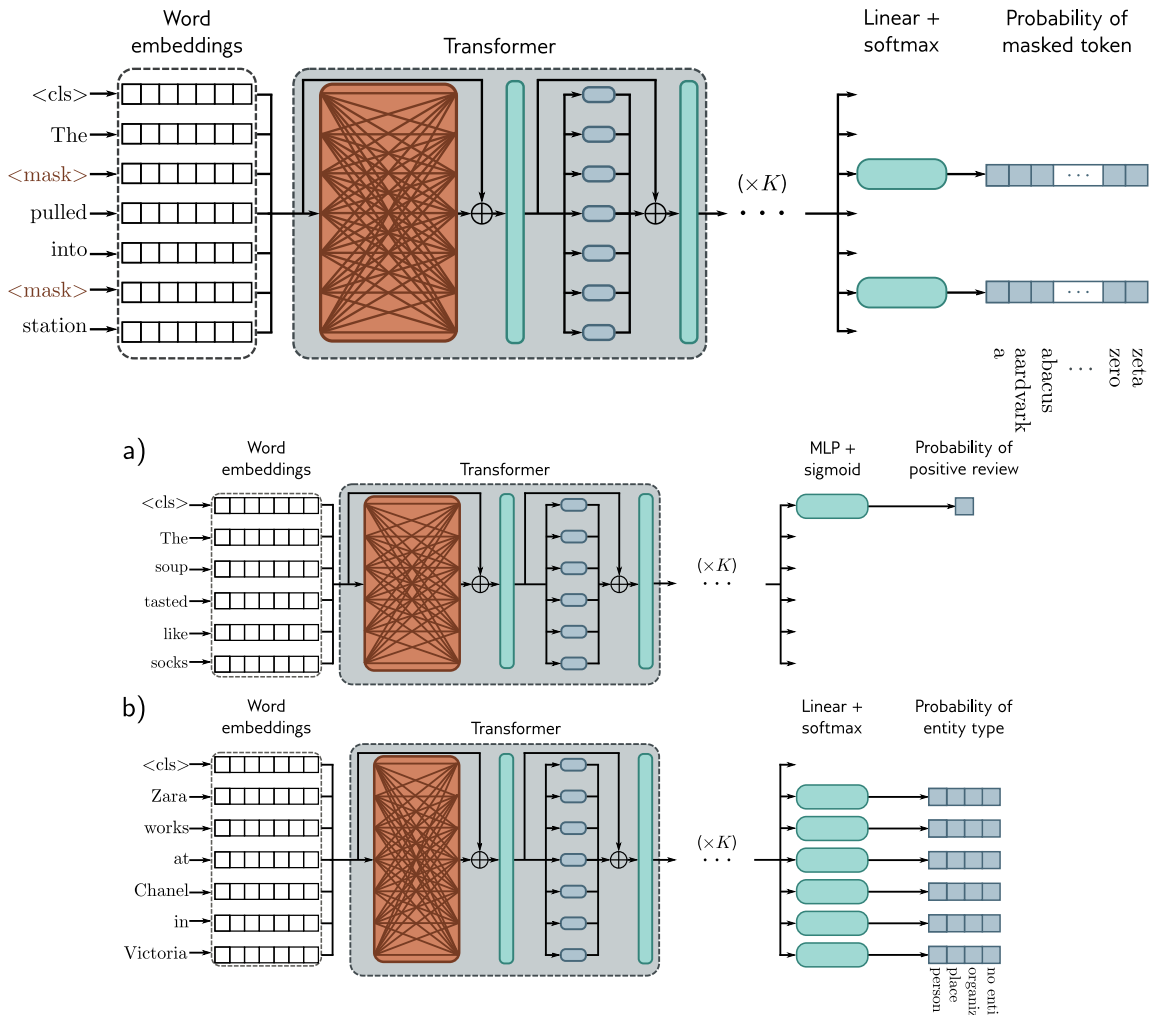


Examples of Encoder Transformer Model: BERT

- It started with a 30,000-tokens vocabulary
- Input tokens are converted to 1024-dimensional word embeddings
- The model consists of 24 transformer layers (each layer has 16 multi-head self attention)
- Total parameters are $\approx$ 340 million.

The Transformer: Transfer Learning

- Pre-training
- Fine-tuning



Decoder: An autoregressive model

Decoder: can be used to generate the next token in a sequence. It can generate a coherent text passage by feeding the extended sequence back into the model.

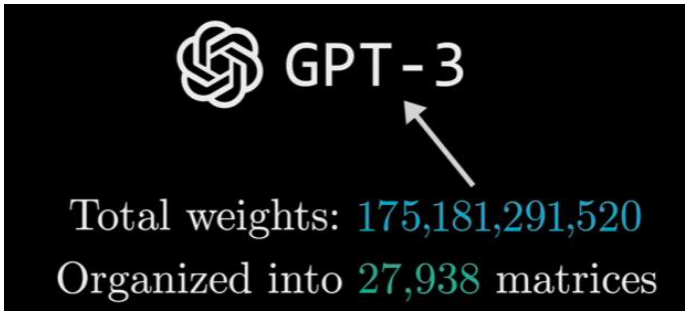
$Pr(\text{It takes great courage to let yourself appear weak})) = Pr(\text{It}) \times Pr(\text{takes}|\text{It}) \times Pr(\text{great}|\text{It takes}) \times ....$

$Pr(\text{weak}|\text{It takes great courage to let yourself appear}).$

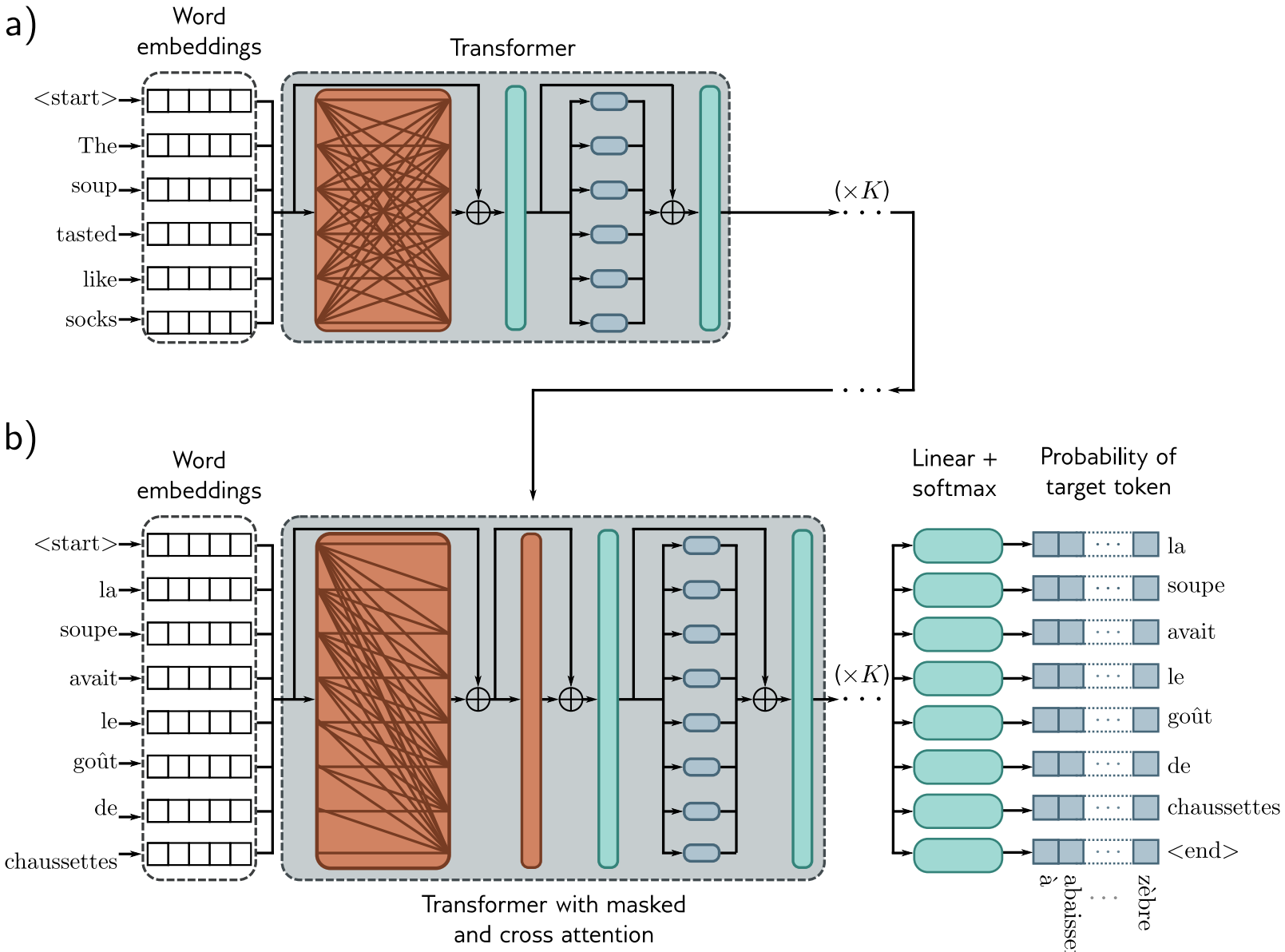
$$Pr(t_1, t_2, ..., t_N) = Pr(t_1) \prod_{n=2}^{N} Pr(t_n|t_1, t_2, ...t_{n-1})$$

Decoder Example: GPT-3

- It trained with 3.2 million tokens.
- Input tokens are converted to 12,288-dimensional word embeddings
- The model consists of 96 transformer layers (each layer has 96 multi-head self attention)
- Total parameters are $\approx$ 175 billion.

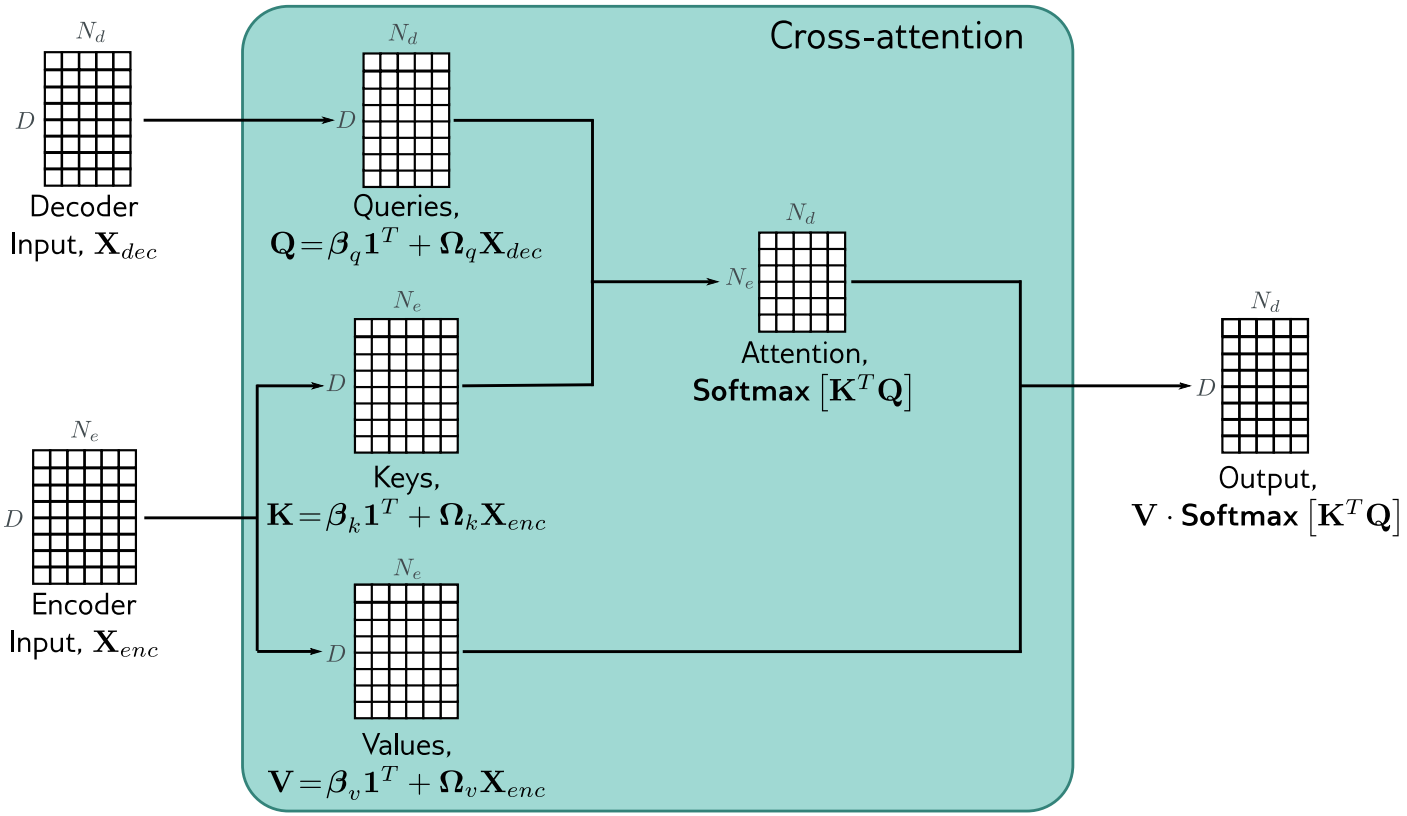


Encoder-Decoder Example: Machine Translation



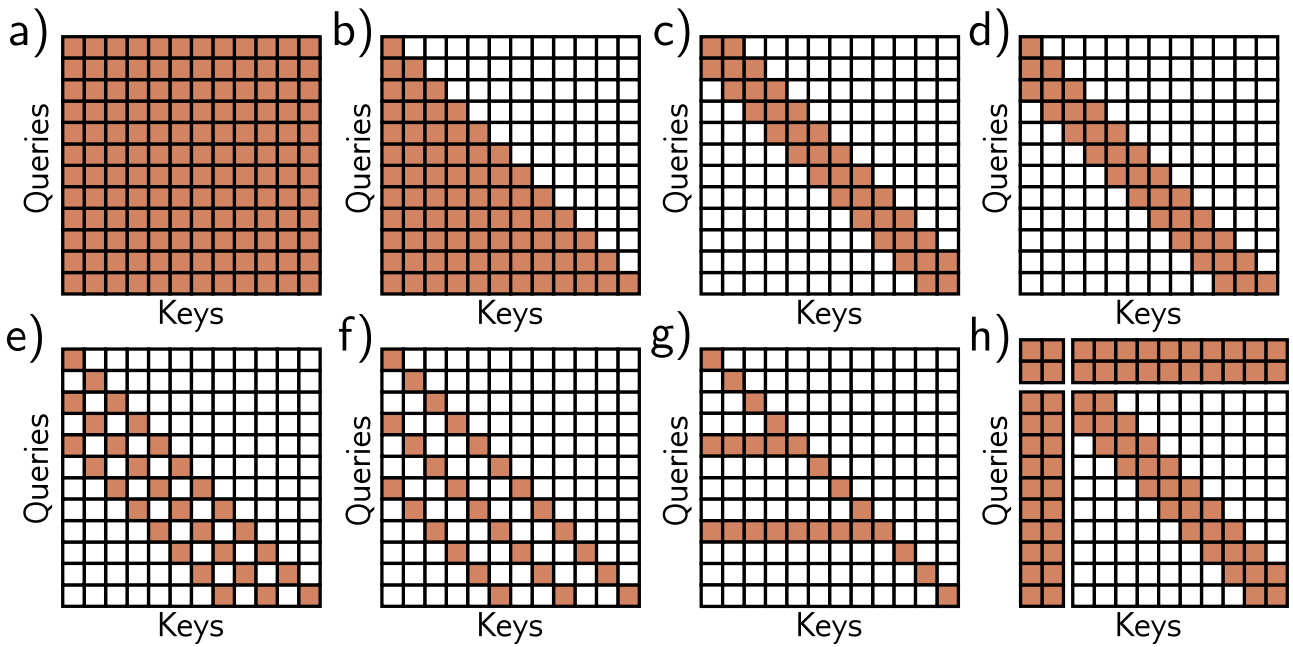
Encoder-Decoder Example: Machine Translation

The decoder embeddings attend to the encoder embeddings.



Transformer for long sequences

The computation increases quadratically with sequence length (N).
Sparsify attention (interaction) matrix



An example of a simple transformer encoder model using single-cell RNA-seq can be found on the course website.