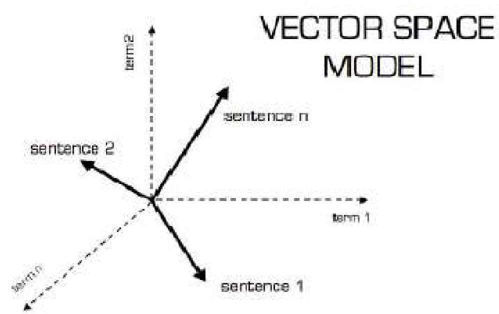
NLP

- Text representation



Document 1

The quick brown fox jumped over the lazy dog's back.

Document 2

Now is the time for all good men to come to the aid of their party.

Term	Document 1	Document 2
aid	0	1
all	0	1
back	1	0
brown	1	0
come	0	1
dog	1	0
fox	1	0
good	0	1
jump	1	0
lazy	1	0
men	0	1
now	0	1
over	1	0
party	0	1
quick	1	0
their	0	1
time	0	1

Text representation

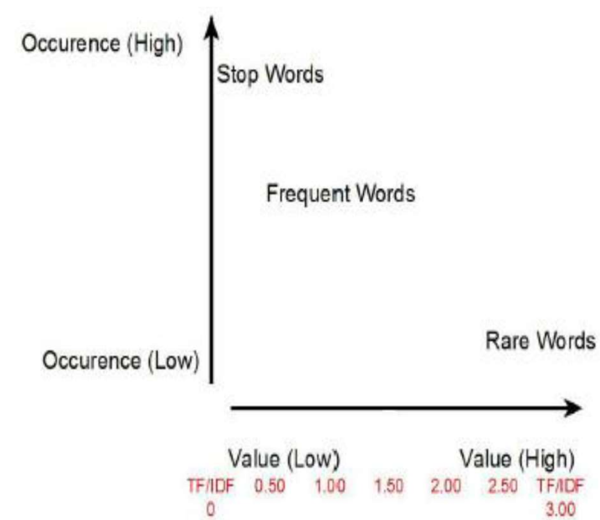
- TF-IDF - term frequency - inverse document freqency

$$\boldsymbol{TF-IDF} = TF_t \times \log\frac{N}{DF_t}$$ IDF

TF_t – **Term Frequency** of the term t. (How many times does the term occur in the document?)

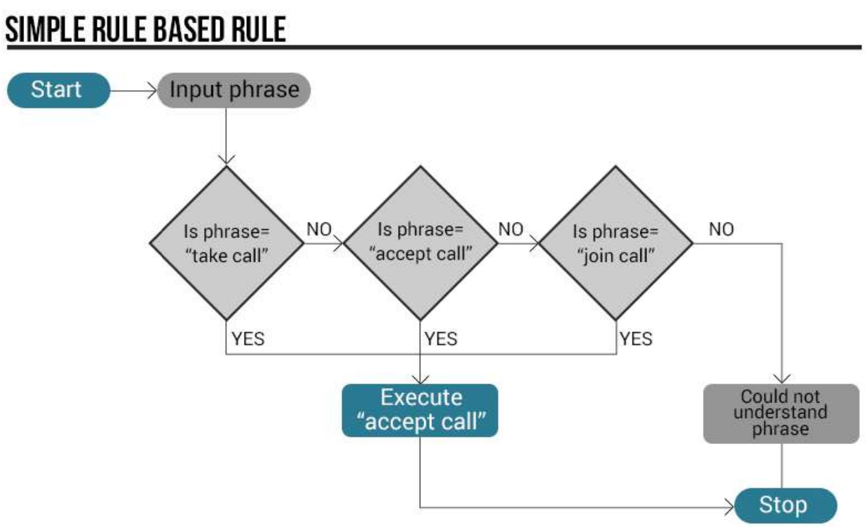
N – Total Number of Documents in the Corpus.

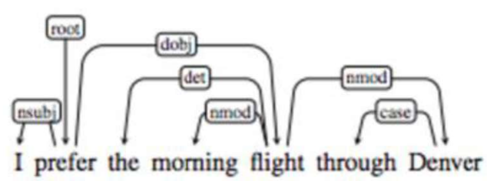
DF_t – **Document Frequency** of t. (How many documents have the term t?)

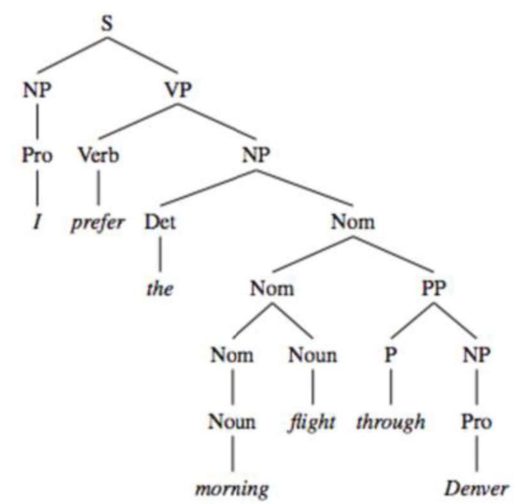


Linguistic Foundations

- Rule-based approaches
- Semantic parsing
- Analyzing linguistic structure and grammars of text

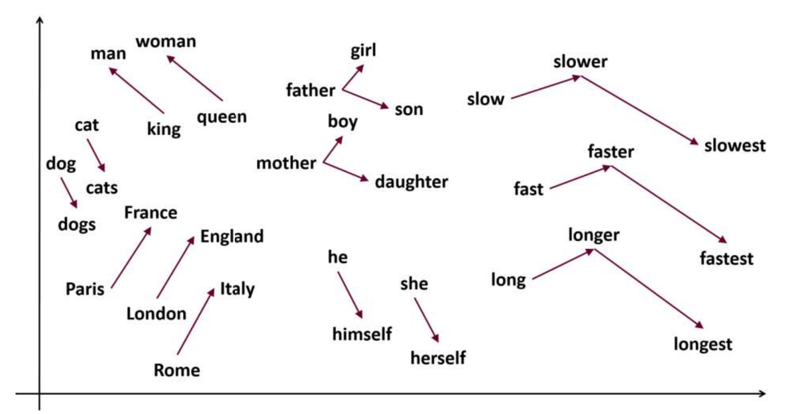






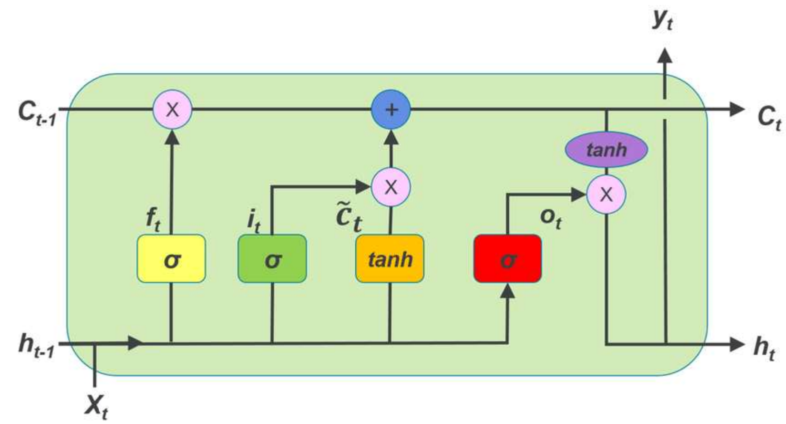
Word Embeddings

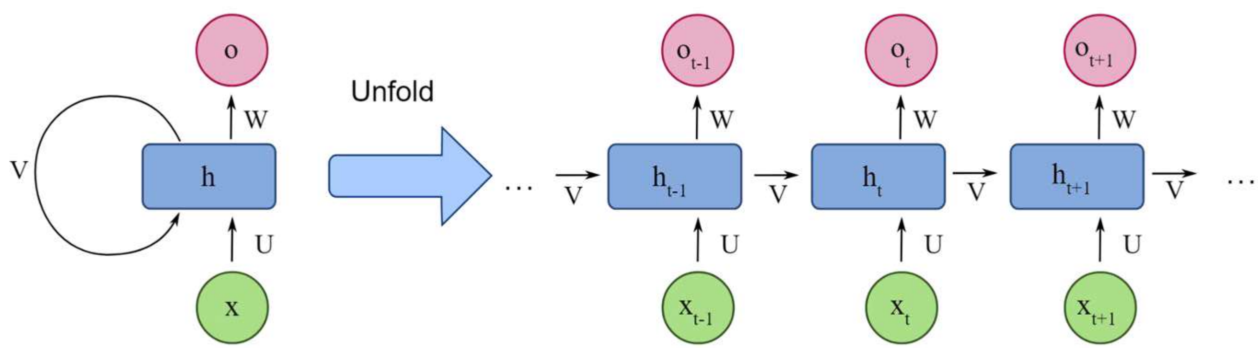
- Represent each word as a "vector" of numbers
- Converts a "discrete" representation to "continuous", allowing for:
  - More "fine-grained" representations of words
  - Useful computations such as cosine/eucl distance
  - Visualization and mapping of words onto a semantic space
- Examples:
  - Word2Vec (2013), GloVe, BERT, ELMo



Seq2seq Models

- Recurrent Neural Networks (RNNs)
- Long Short-Term Memory Networks (LSTMs)
- "Dependency" and info between tokens
- Gates to "control memory" and flow of information

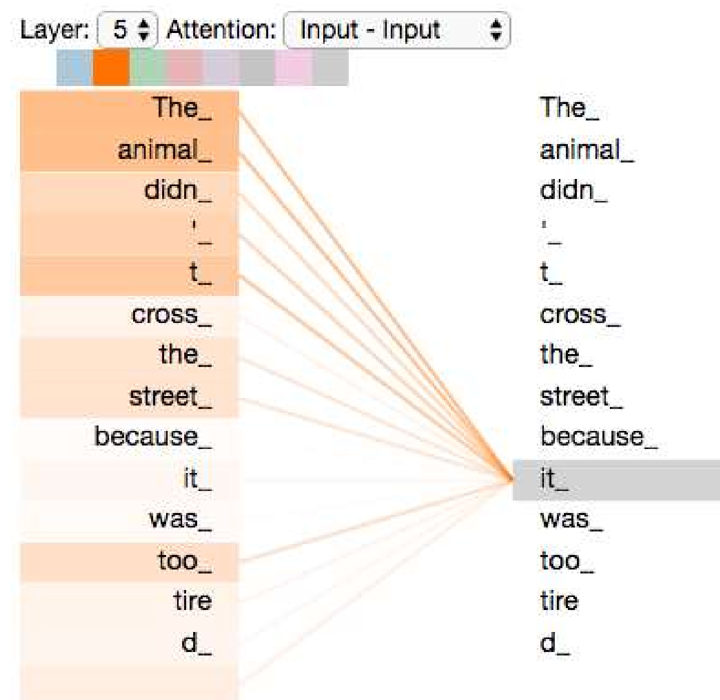




Attention and Transformers

- Allows to “focus attention” on particular aspects of the input text
- Done by using a set of parameters, called "weights," that determine how much attention should be paid to each input at each time step
- These weights are computed using a combination of the input and the current hidden state of the model
- Attention weights are computed (dot product of the query, key and value matrix), then a softmax function is applied to the dot product

$$\mathrm{attention}(Q,K,V) = \mathrm{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) V$$



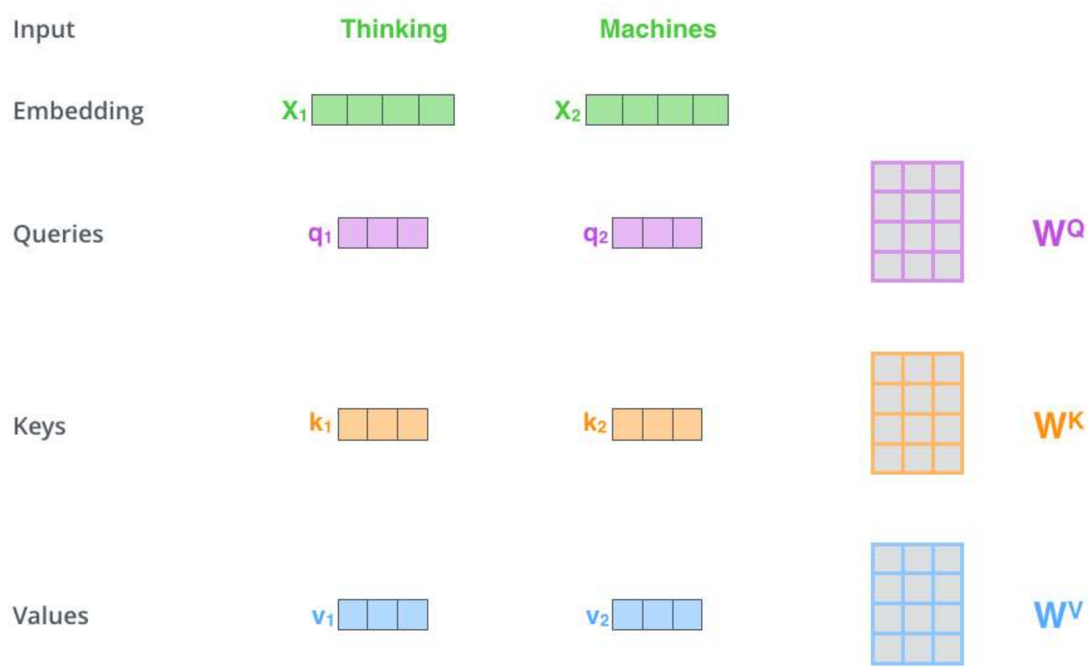
https://arxiv.org/abs/1706.03762
https://jalammar.github.io/illustrated-transformer/

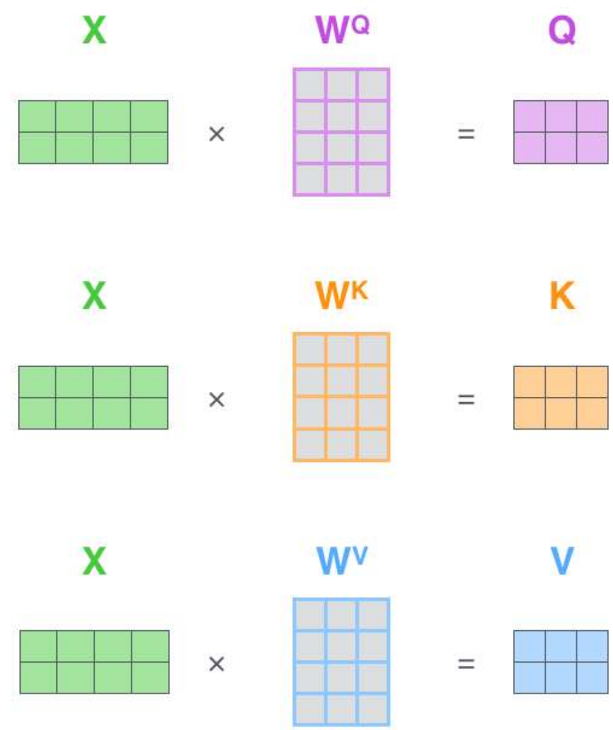
Analogy for Q, K, V

- Library system
- Imagine you're looking for information on a specific topic (query)
- Each book in the library has a summary (key) that helps identify if it contains the information you're looking for
- Once you find a match between your query and a summary, you access the book to get the detailed information (value) you need
- Here, in Attention, we do a "soft match" across multiple values, e.g. get info from multiple books ("book 1 is most relevant, then book 2, then book 3, etc.")

$$\mathrm{attention}(Q, K, V) = \mathrm{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) V$$

Self-Attention





https://jalammar.github.io/illustrated-transformer/

Transformer & Multi-Head Attention

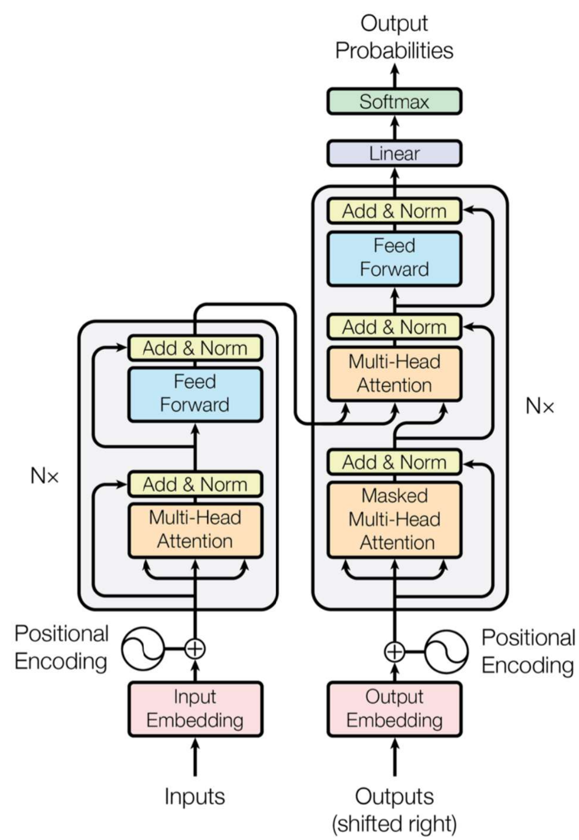
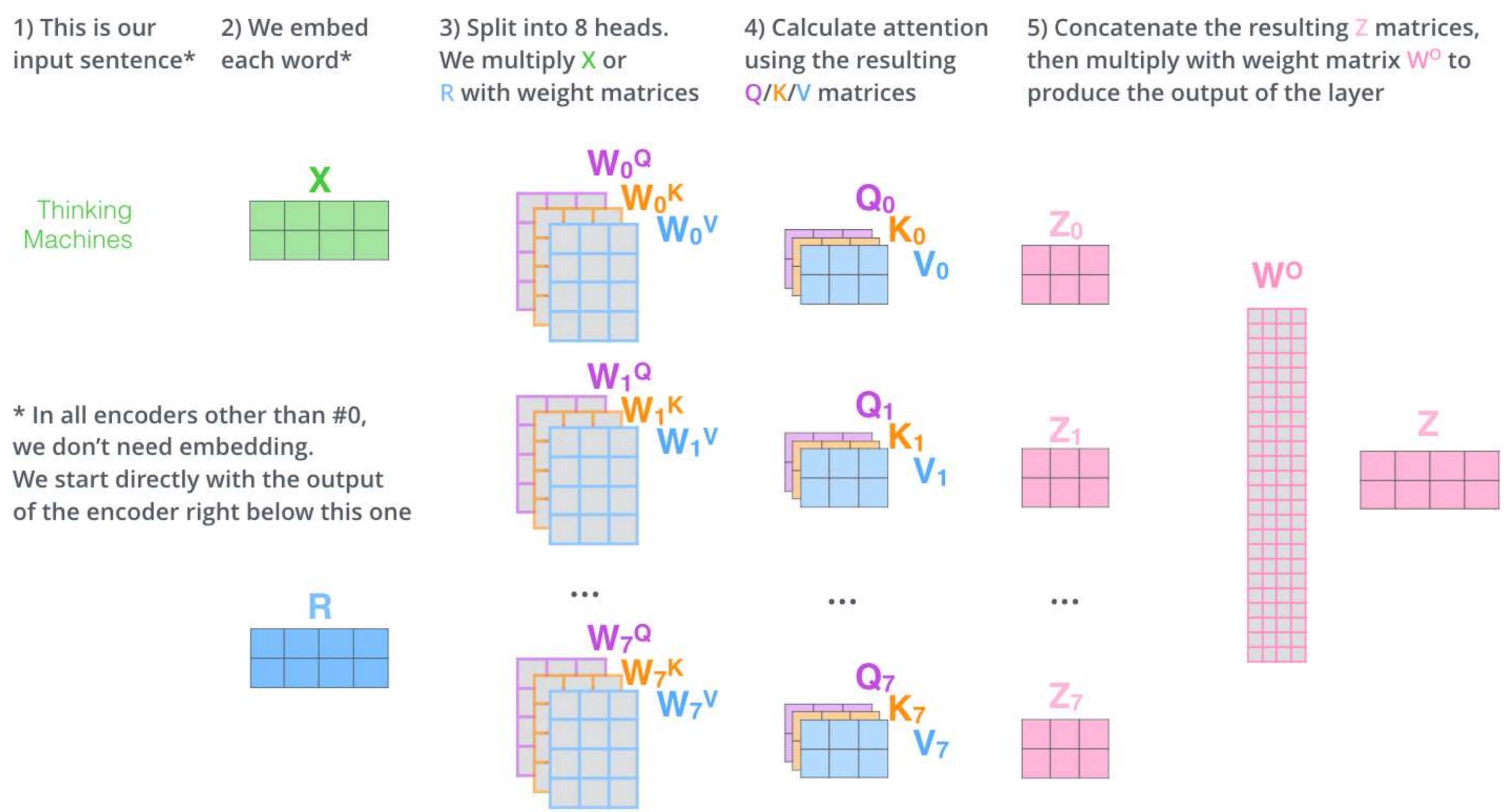


Figure 1: The Transformer - model architecture.

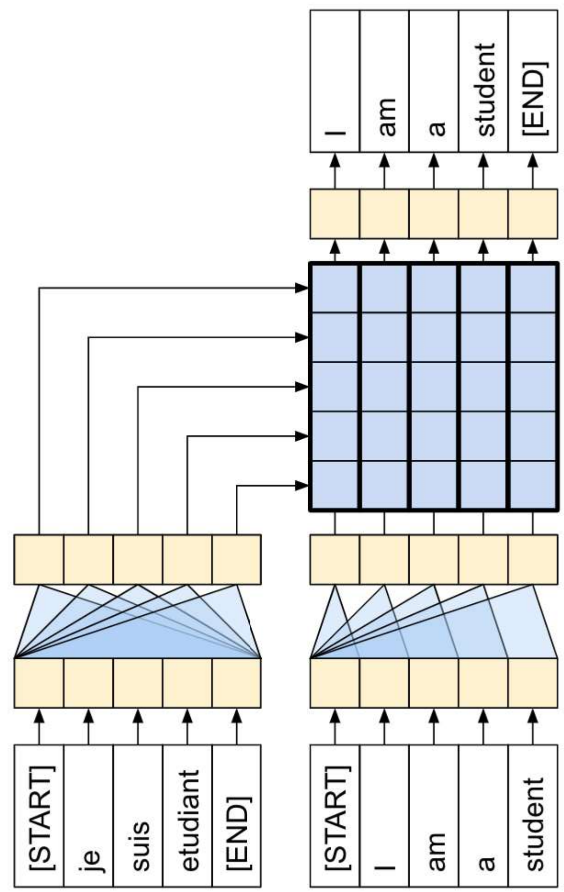
"Attention Is All You Need"
https://arxiv.org/abs/1706.03762

Multi-Head Attention



https://jalammar.github.io/illustrated-transformer/

Cross-Attention (e.g. Machine Translation)



https://jalammar.github.io/illustrated-transformer/

Transformers vs. RNNs

Challenges with RNNs	Transformers
• Long range dependencies • Gradient vanishing and explosion • Large # of training steps • Sequential/recurrence → can't parallelize • Complexity per layer: $O(n*d^2)$	• Can model long-range dependencies • No gradient vanishing and explosion • Fewer training steps • Can parallelize computation! • Complexity per layer: $O(n^2*d)$

Large Language Models

- Scaled up versions of Transformer architecture, e.g. millions/billions of parameters
- Typically trained on massive amounts of “general” textual data (e.g. web corpus)
- Training objective is typically “next token prediction”: $P(W_{t+1} | W_t, W_{t-1}, \ldots, W_1)$
- Emergent abilities as they scale up (e.g. chain-of-thought reasoning)
- Heavy computational cost (time, money, GPUs)
- Larger general ones: “plug-and-play” with few or zero-shot learning
  - Train once, then adapt to other tasks without needing to retrain
  - E.g. in-context learning and prompting

Emergent Abilities of Large Language Models

- Why do LLMs work so well? What happens as you scale up?
- Potential explanation: emergent abilities!
- An ability is emergent if it is present in larger but not smaller models
- Not have been directly predicted by extrapolating from smaller models
- Performance is near-random until a certain critical threshold, then improves heavily
  - Known as a “phase transition” and would not have been extrapolated

Wei et al., 2022. https://arxiv.org/abs/2206.07682