Natural Language Processing (NLP) Extra Concepts

NLP

- Machine translation.
- Information retrieval (e.g., search engines).
- Sentiment analysis (e.g., positive, negative, happiness, sadness, etc.).
- Information extraction (e.g., summary, keywords, etc.).
- Text generation.

Concept: Tokens

- Individual pieces of text.
- E.g.,
  - Words.
  - Punctuation.
  - Names (ChatGPT…).
  - Abbreviations (LSTM, USA…).

The success of ChatGPT

1. Massive amounts of data.
2. Significant computing power.
3. Word embeddings.
4. Self-attention.
5. Transformers.
6. Reinforcement Learning from Human Feedback (RLHF).

The success of ChatGPT

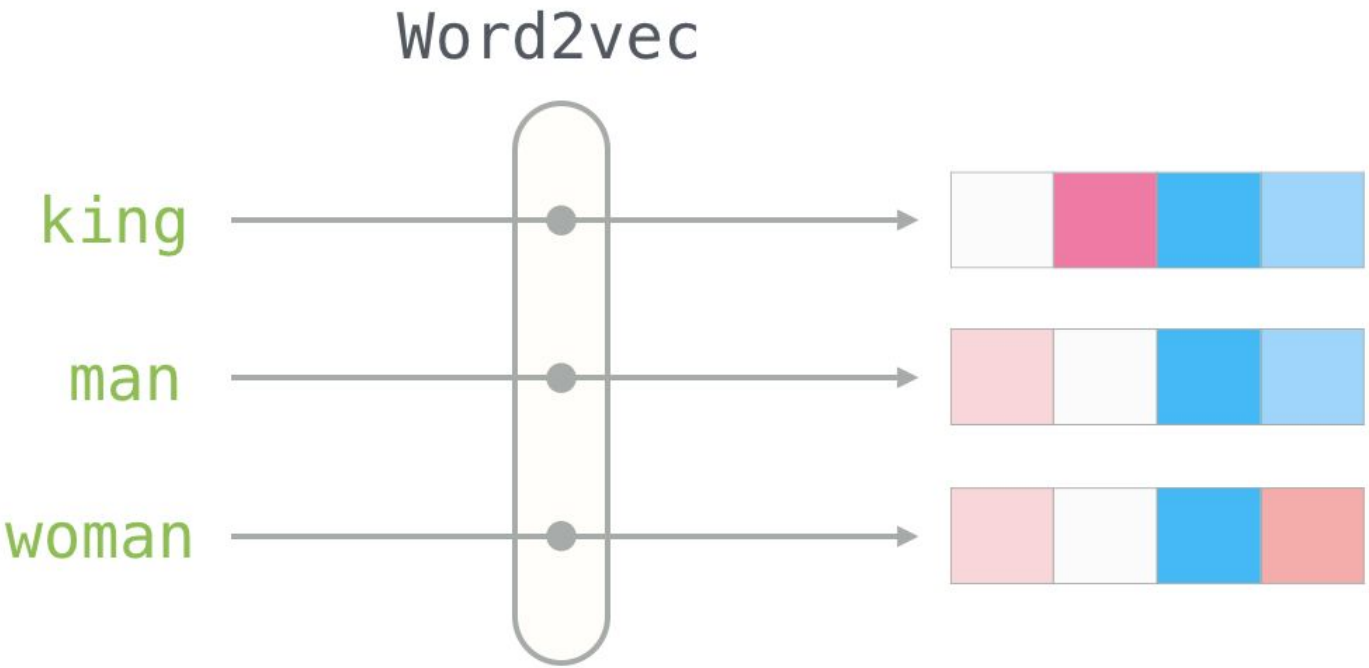
1. Massive amounts of data.
2. Significant computing power.
3. Word embeddings. → 1980; 2013 (word2vec)
4. Self-attention. → 2014; 2017 (transformers)
5. Transformers. → 2017
6. Reinforcement Learning from Human Feedback (RLHF).

The success of ChatGPT

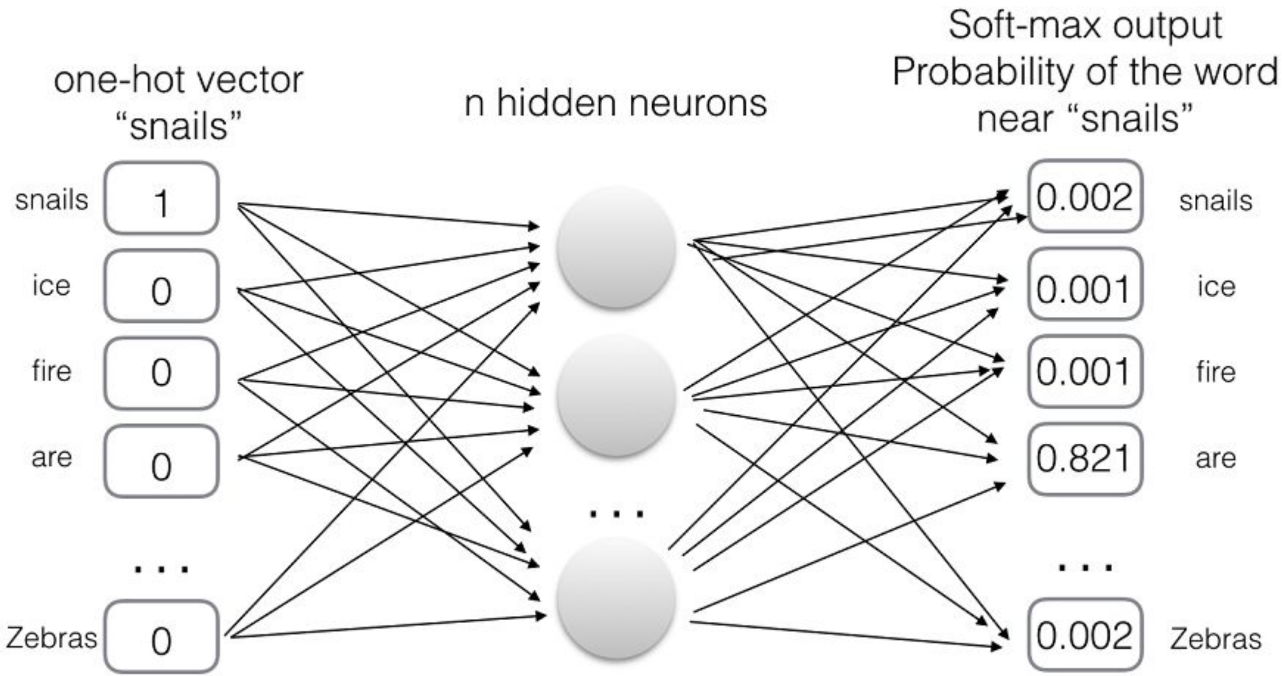
1. Massive amounts of data.
2. Significant computing power.
3. Word embeddings.
4. Self-attention.
5. Transformers.
6. Reinforcement Learning from Human Feedback (RLHF).

Word embedding

- Representation of words, typically vectors.
  - Words that share common contexts in the corpus should have similar vectors.
  - More dissimilar words are located further
- word2vec: Model that produces word embeddings.


Word2vec
king
man
woman

Word2vec



one-hot vector
"snails"
n hidden neurons
Soft-max output
Probability of the word
near "snails"
snails
1
ice
0
fire
0
are
0
...
Zebras
0
...
0.002
snails
0.001
ice
0.001
fire
0.821
are
...
0.002
Zebras

Word2vec

Model architectures:

- Continuous Bag-Of-Words (CBOW).
- Skip-gram.

Word2vec – CBOW

1. Identify all V unique words in the corpus.

Word2vec – CBOW

1. Identify all V unique words in the corpus.
2. Choose the dimensionality N for your word embeddings.

Word2vec – CBOW

1. Identify all *V* unique words in the corpus.
2. Choose the dimensionality *N* for your word embeddings.
3. Neural Network architecture:
   a. Input layer of size *V*.
   b. Hidden layer of size *N*.
   c. Output layer of size *V*.

Word2vec – CBOW

1. Identify all *V* unique words in the corpus.
2. Choose the dimensionality *N* for your word embeddings.
3. Neural Network architecture:
   a. Input layer of size *V*.
   b. Hidden layer of size *N*.
   c. Output layer of size *V*.
4. Task:
   a. Input: vector of size *V*, representing the context of a word.
   b. Output: Predict the desired word.

Word2vec – CBOW

- Assuming 2 words as context, the input to the NN is:
    - Compute the one-hot encoded vector of length *V* for each context word.
    - Compute the average of such vectors.

Word2vec – CBOW

- Assuming 2 words as context, the input to the NN is:
  - Compute the one-hot encoded vector of length *V* for each context word.
  - Compute the average of such vectors.
- Example: “the fierce gray wolf hunts”
  - Target: “gray”
  - Context of 2:
    - “fierce” → [0, 1, 0, 0, 0]
    - “wolf” → [0, 0, 0, 1, 0]
  - NN input = average of vectors “fierce” and “wolf”:
    - input → [0, ½, 0, ½, 0]

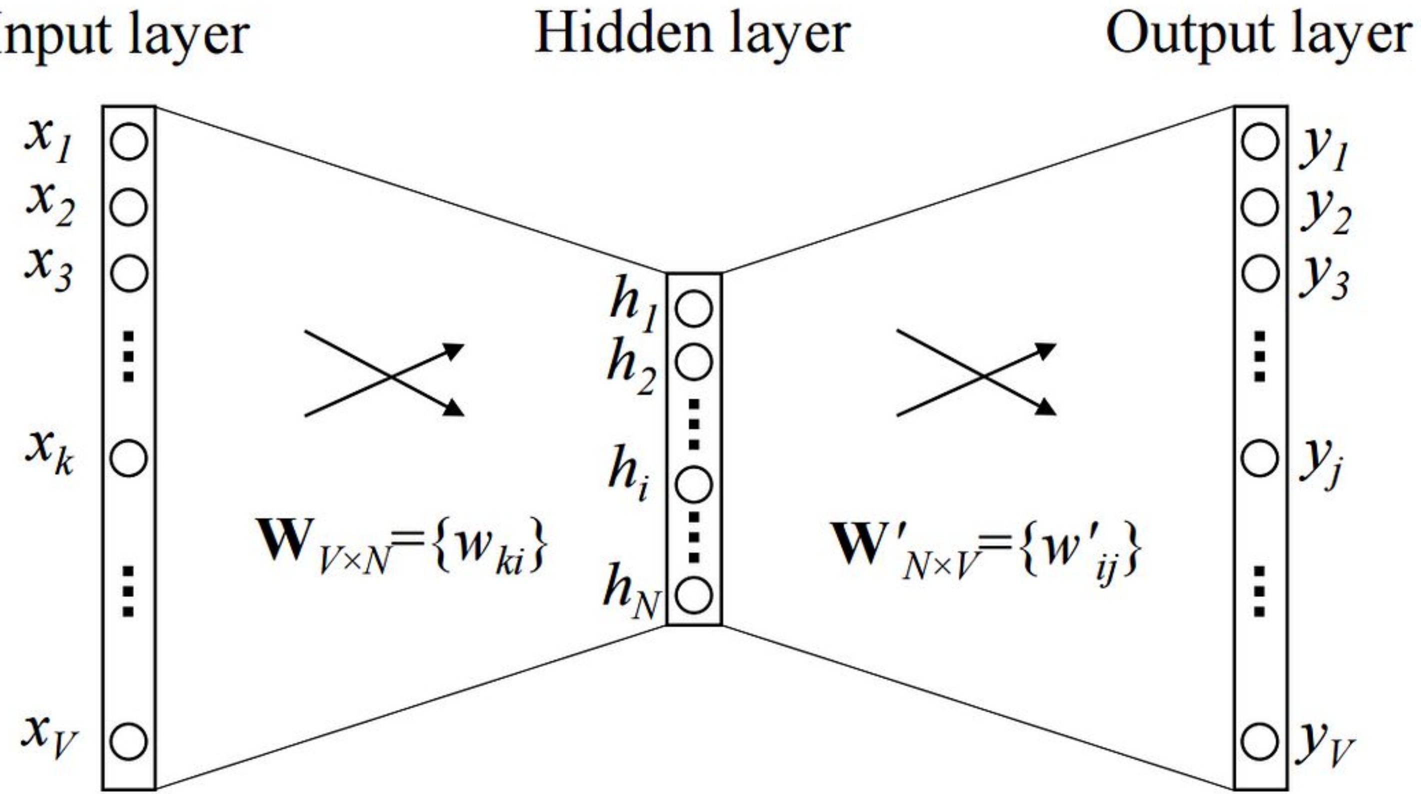
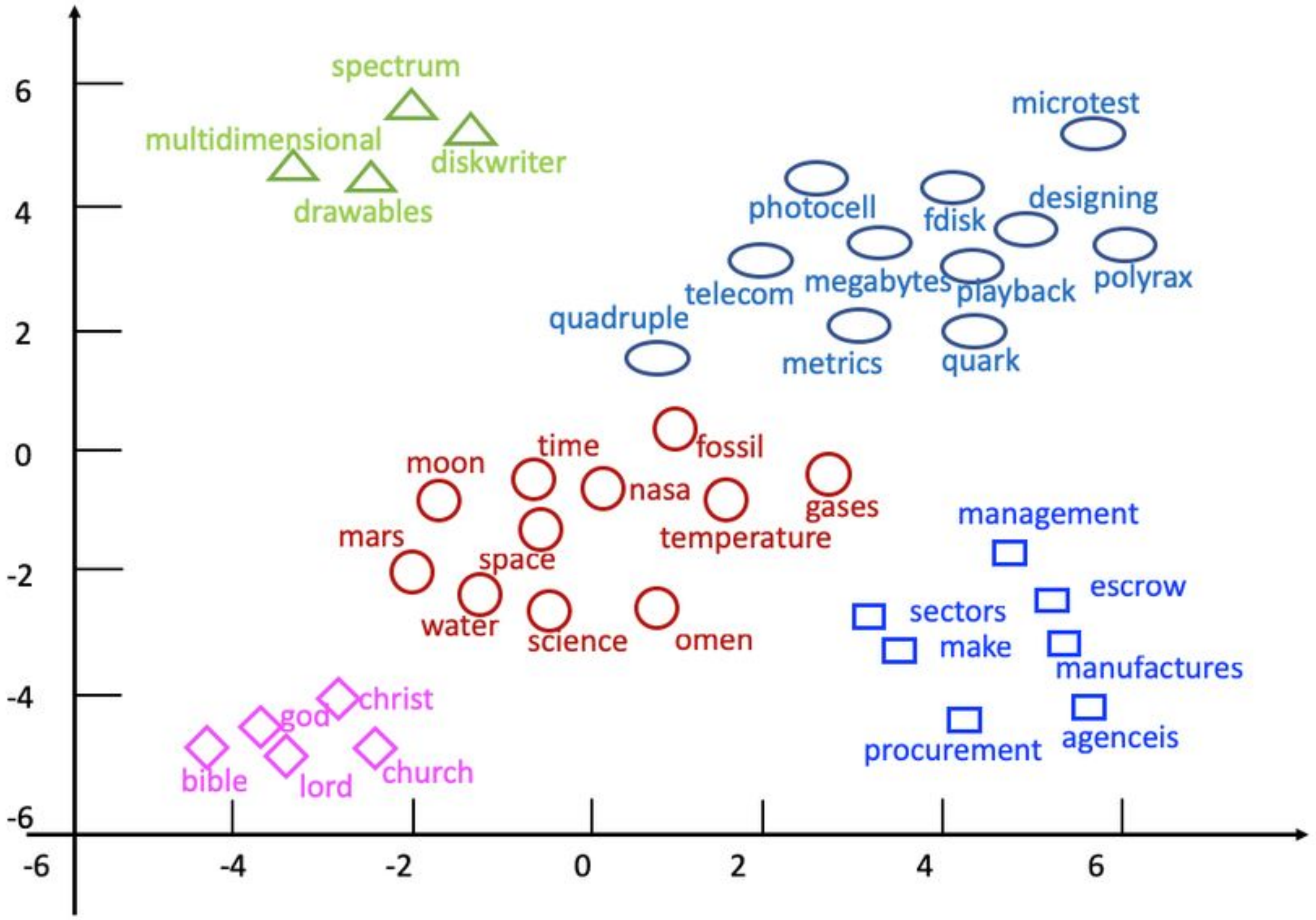


Figure 1: A simple CBOW model with only one word in the context

Word2vec – CBOW

- After training with a lot of text, the NN has learned the weights of abstract representations of words and context.
- For each word in the corpus, the embedded vector representation of length *N* is:
  - The weights associated to that word, from the input to the hidden layer.
  - E.g.,
    - The one-hot encoded vector of “wolf” is [0, 0, 0, 1, 0].
    - The embedded vector for “wolf” is the *N* weights associated with the 4th input neuron.



spectrum
multidimensional
diskwriter
drawables
microtest
photocell
fdisk
designing
telecom
megabytes
playback
polyrax
quadruple
metrics
quark
time
fossil
moon
nasa
gases
mars
temperature
space
water
science
omen
management
escrow
sectors
make
manufactures
christ
god
bible
lord
church
procurement
agenceis
6
4
2
0
-2
-4
-6
-6
-4
-2
0
2
4
6

Resources

- https://www.youtube.com/watch?v=UqRCEmrv1gQ

The success of ChatGPT

1. Massive amounts of data.
2. Significant computing power.
3. Word embeddings.
4. Self-attention.
5. Transformers.
6. Reinforcement Learning from Human Feedback (RLHF).

Self-attention

- Applied in a sequence of words.
- Allows to pay attention to similar/important words in the sequence.
- In contrast, RNNs process words sequentially, favoring more recent words.

Self-attention

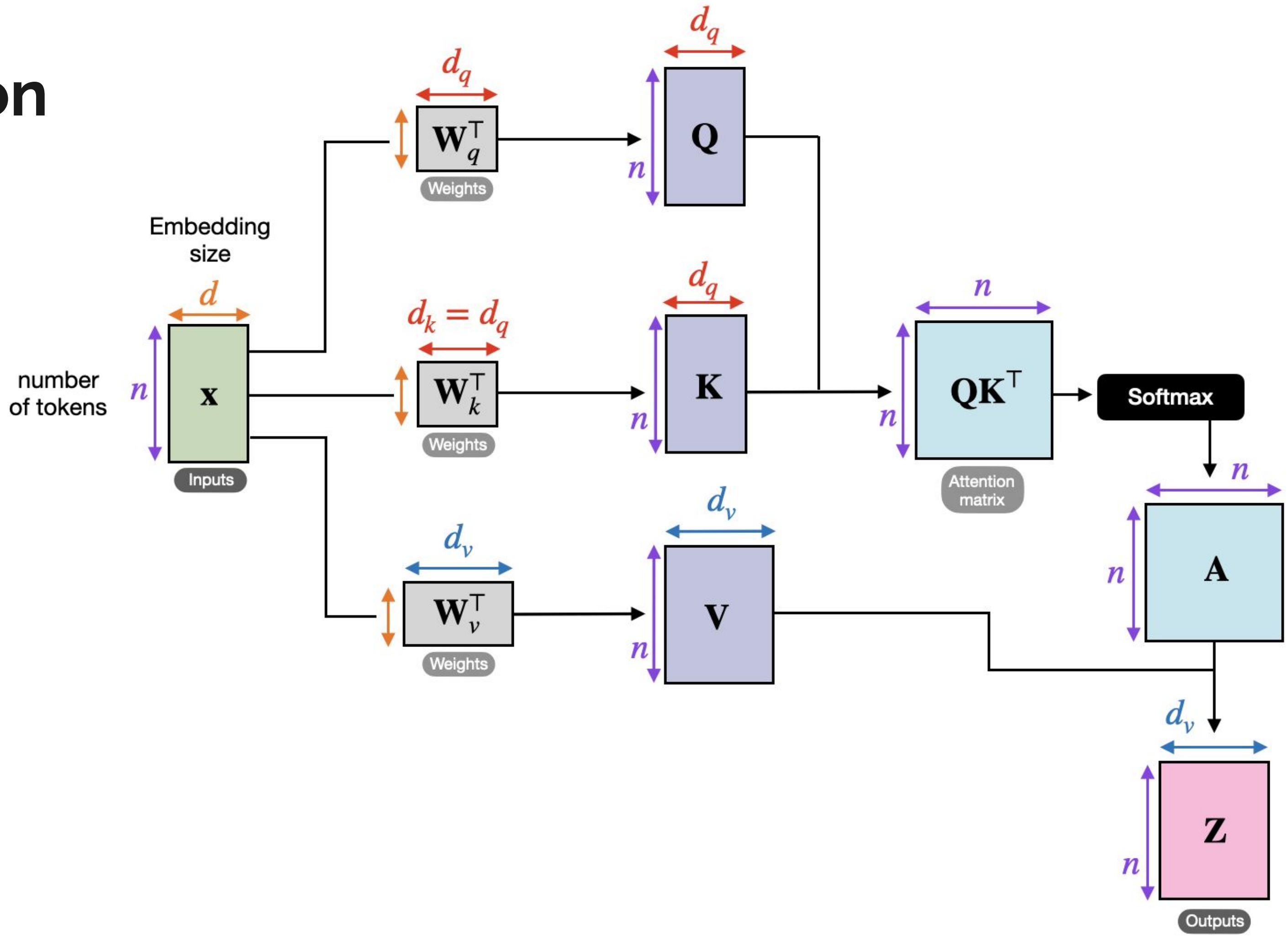
- Applied in a sequence of words.
- Allows to pay attention to similar/important words in the sequence.
- In contrast, RNNs process words sequentially, favoring more recent words.

The young <u>boy</u> always carries his toy car with $\boxed{?} \longrightarrow$ **him**

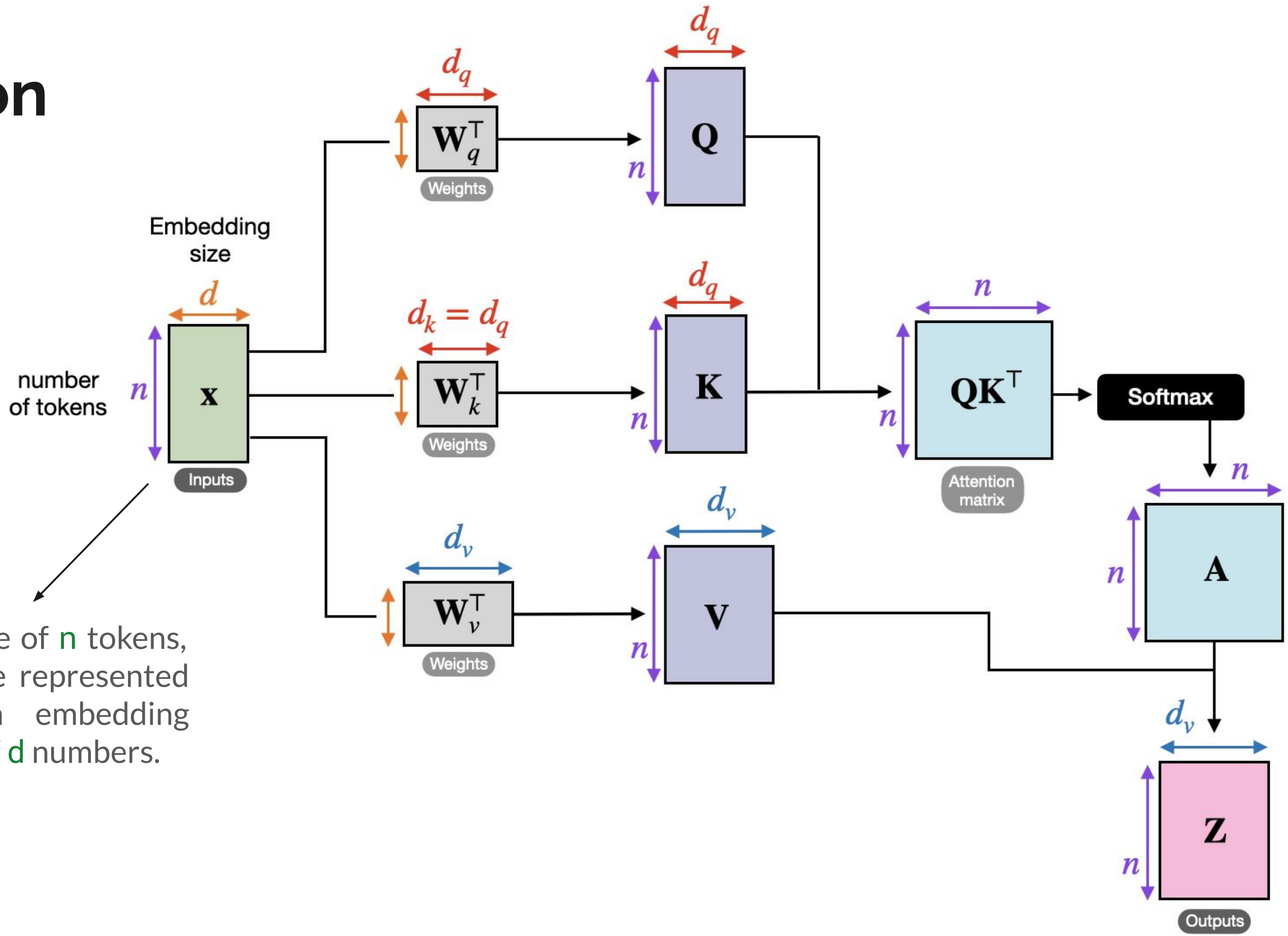
Self-attention

- Given a sequence of words,
- 3 weight matrices to optimize:
    - Query: used to compare words against other words.
    - Key: helps in measuring the relevance between words depending on context.
    - Value: the resulting vectors hold contextual information of each word.

Self-Attention

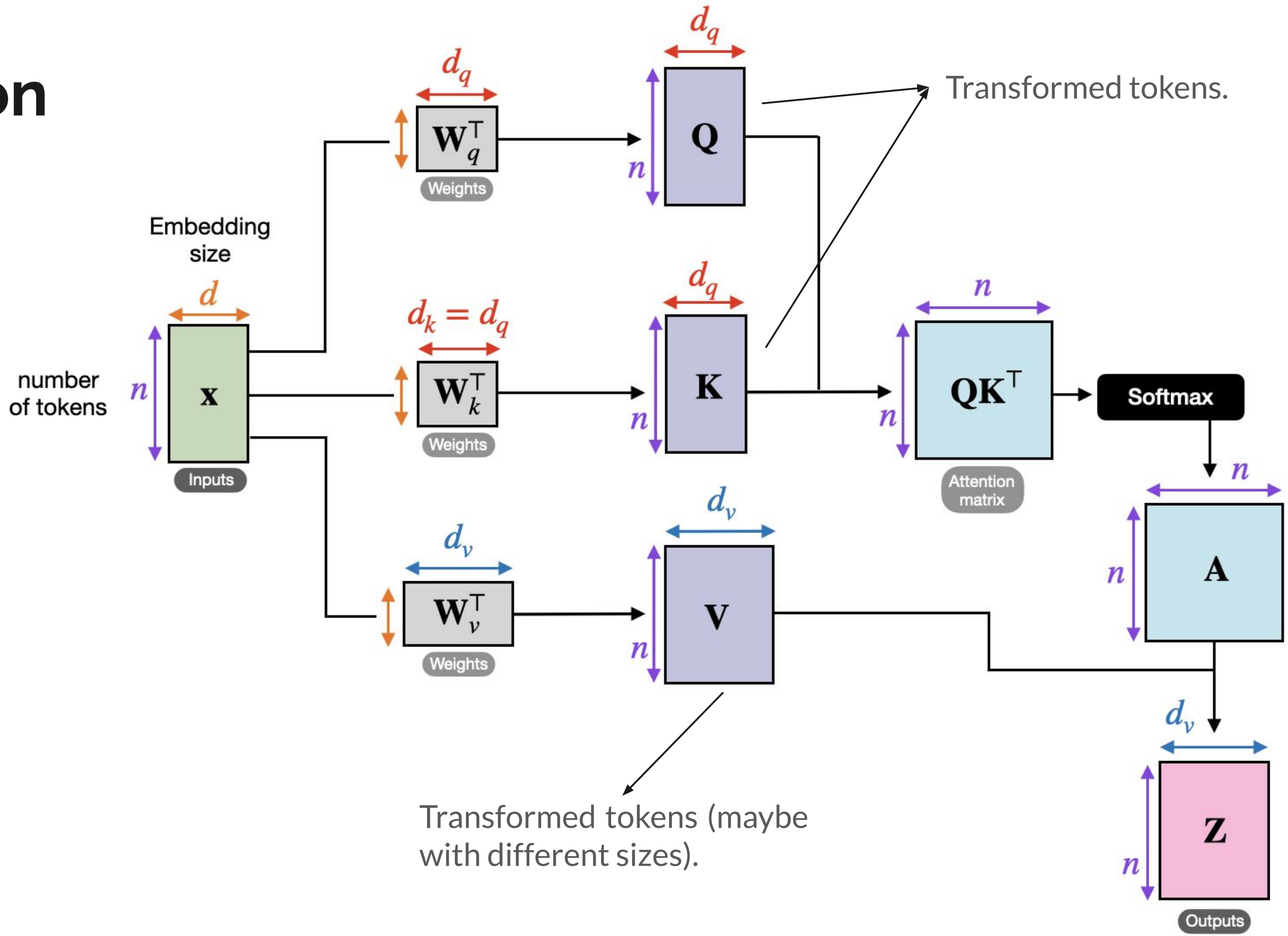


Self-Attention

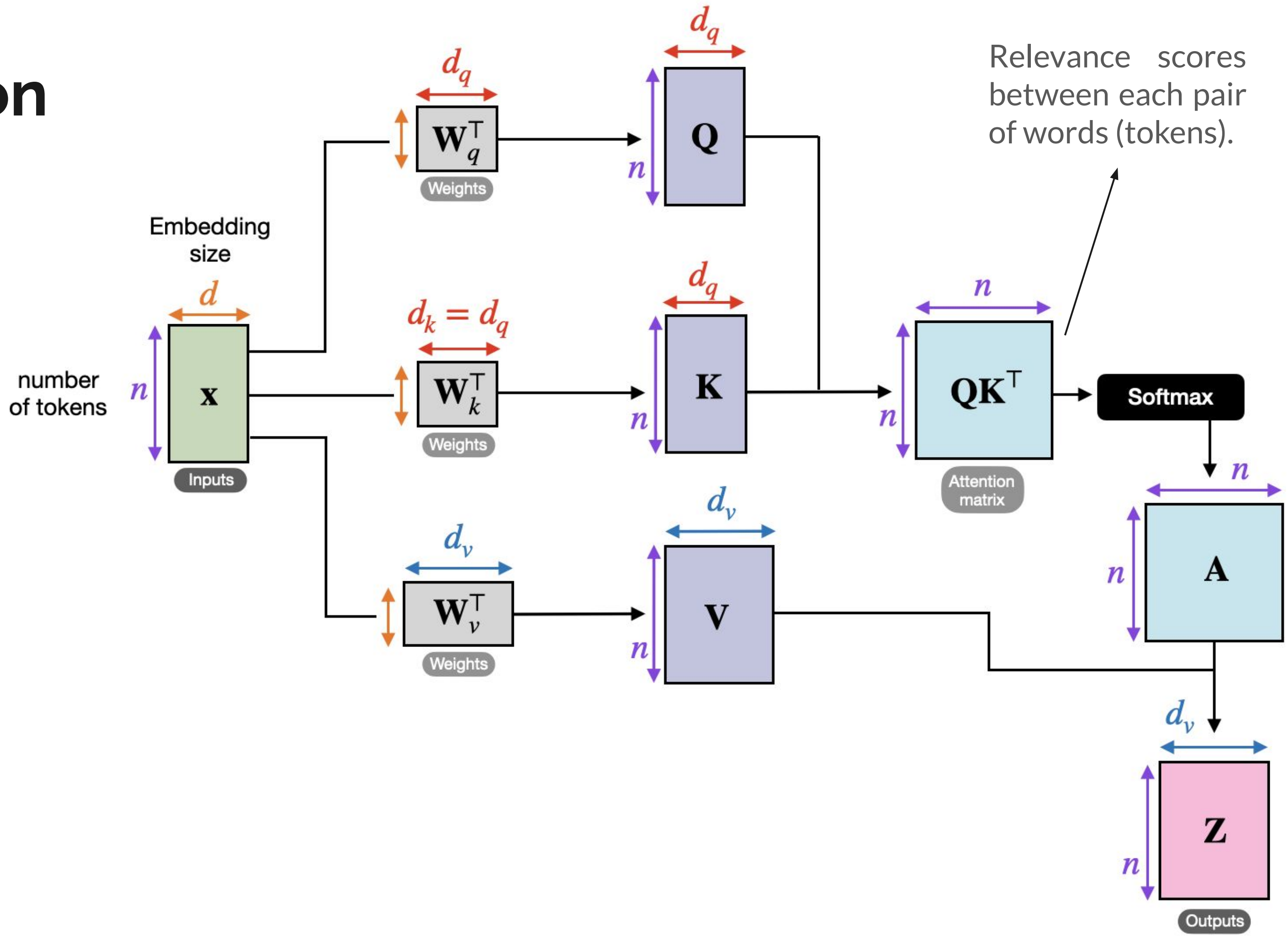


Sequence of n tokens, each one represented with an embedding vector of d numbers.

Self-Attention

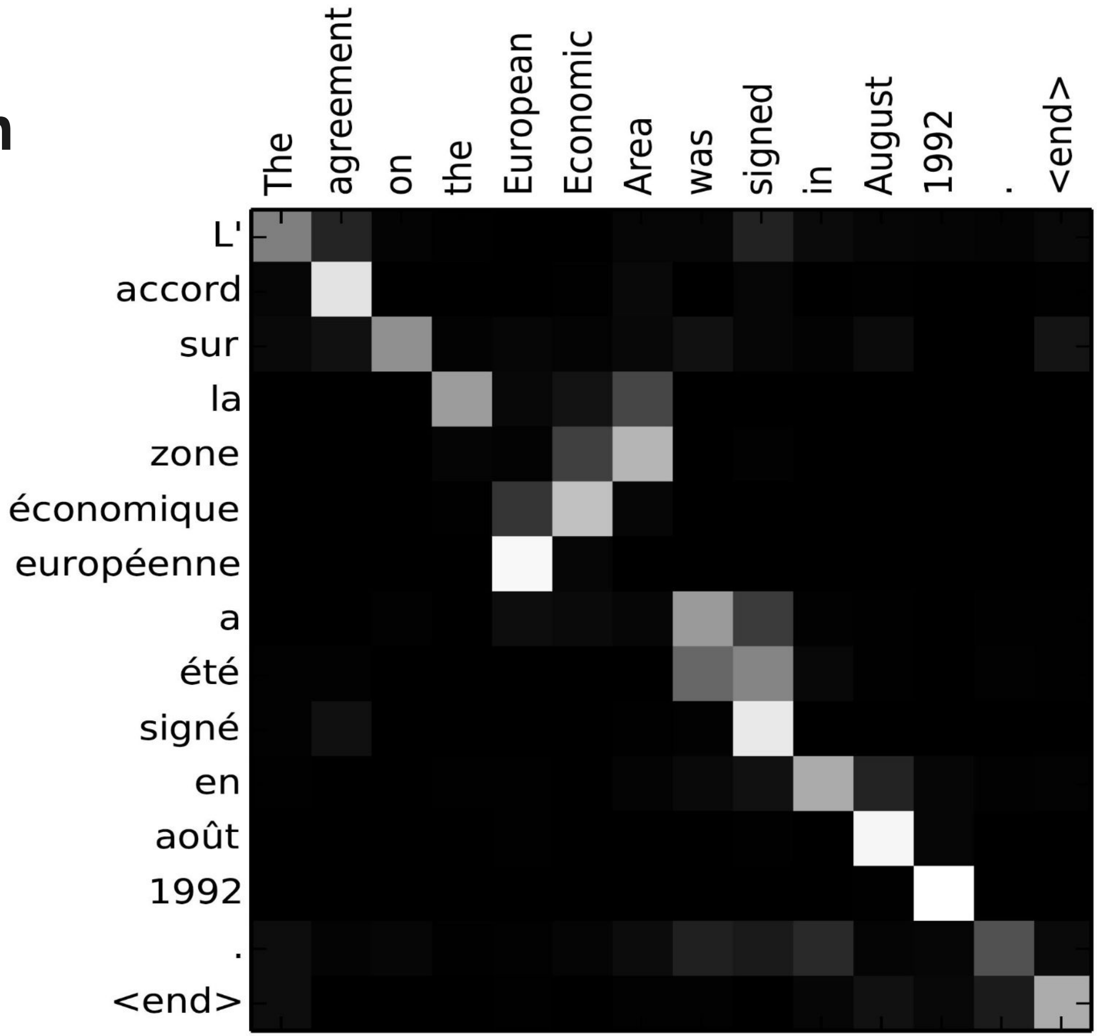


Self-Attention

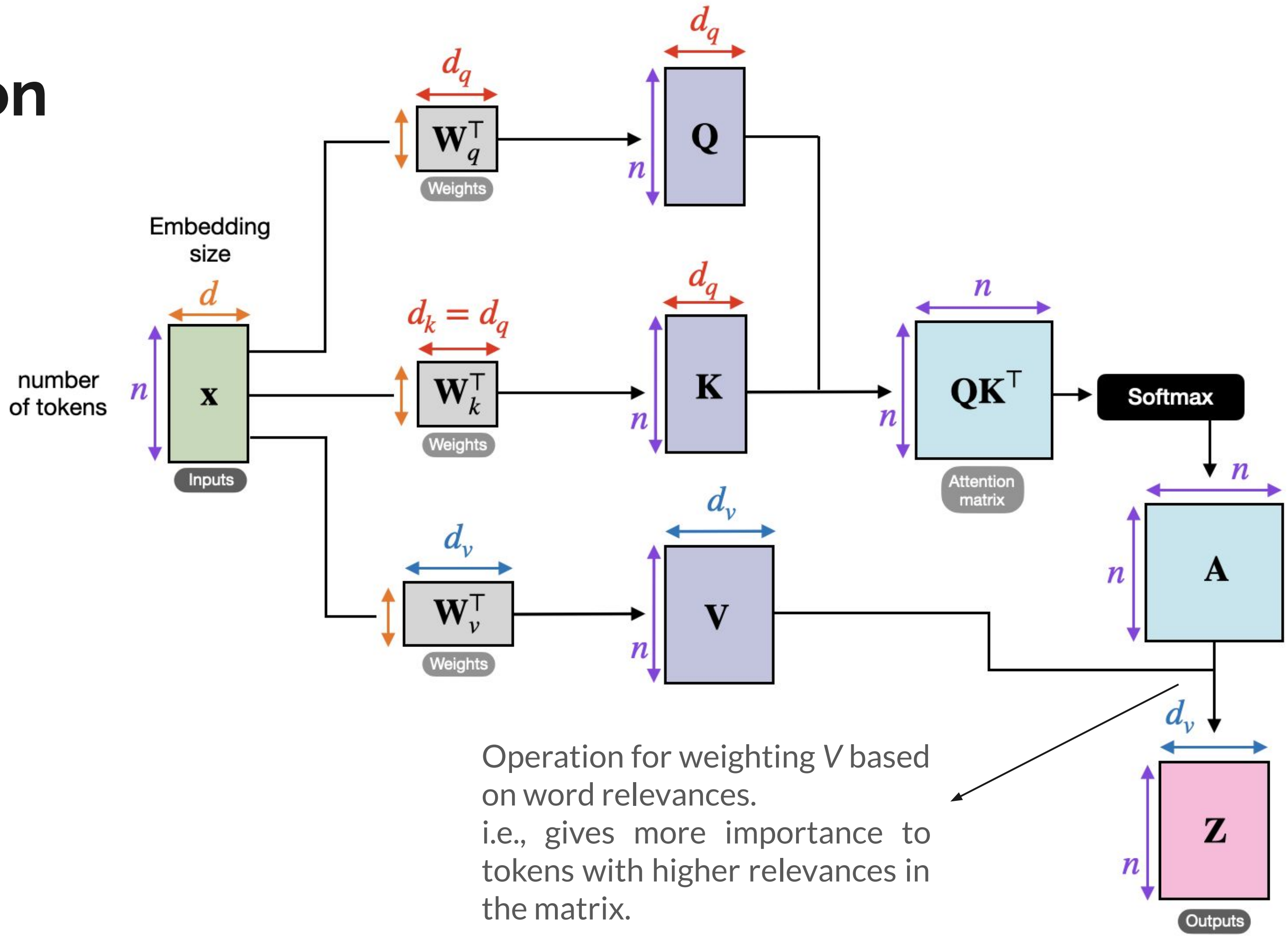


Relevance scores between each pair of words (tokens).

Attention Matrix

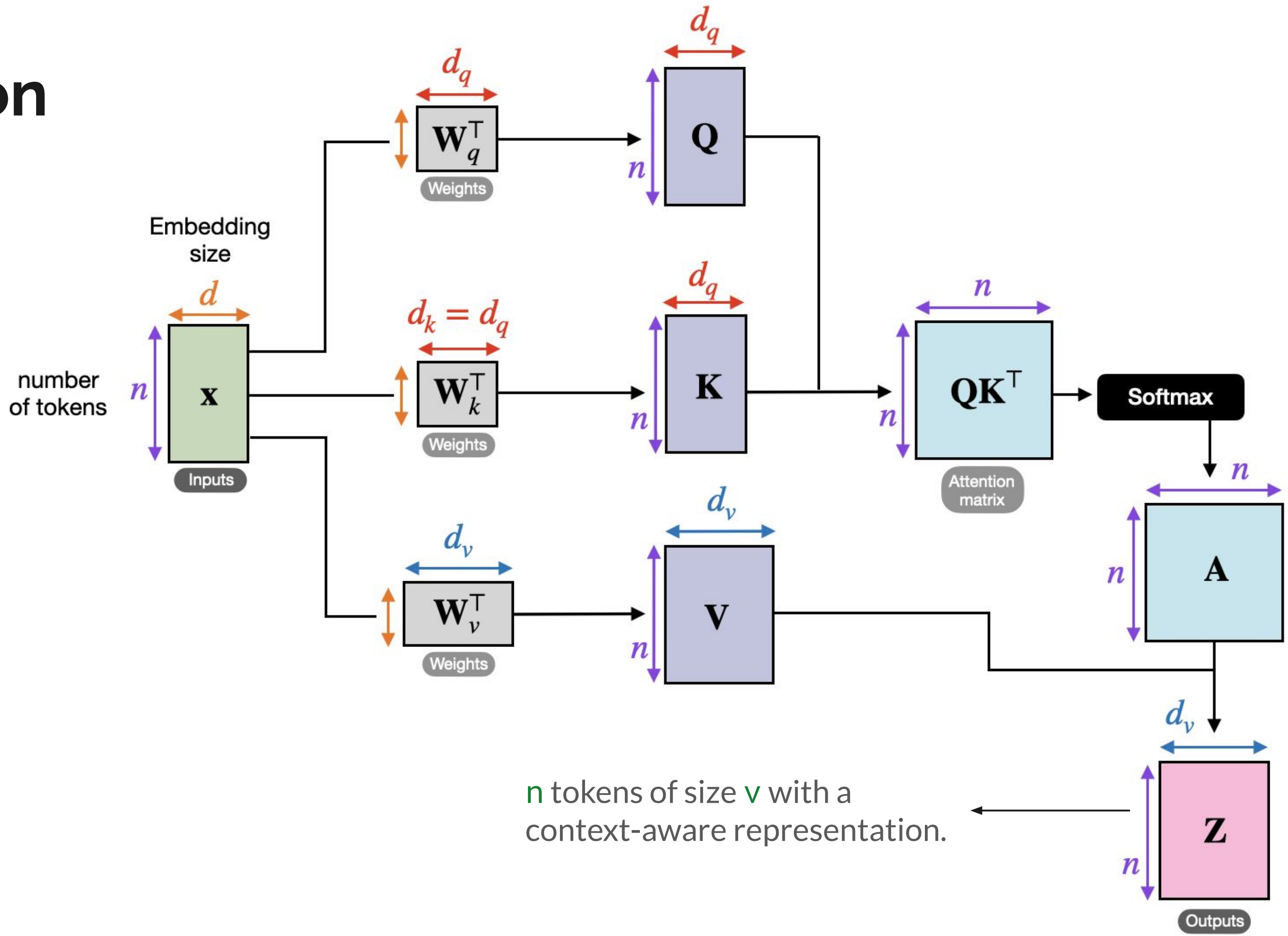


Self-Attention



Operation for weighting *V* based on word relevances.
i.e., gives more importance to tokens with higher relevances in the matrix.

Self-Attention



n tokens of size v with a context-aware representation.

Self-attention

- Pros:
  - Contextual understanding.
  - Operations between tokens can be applied in parallel.
- Cons:
  - Computationally and memory intensive for long sequences.
  - Does not consider sequential (positional) importance.

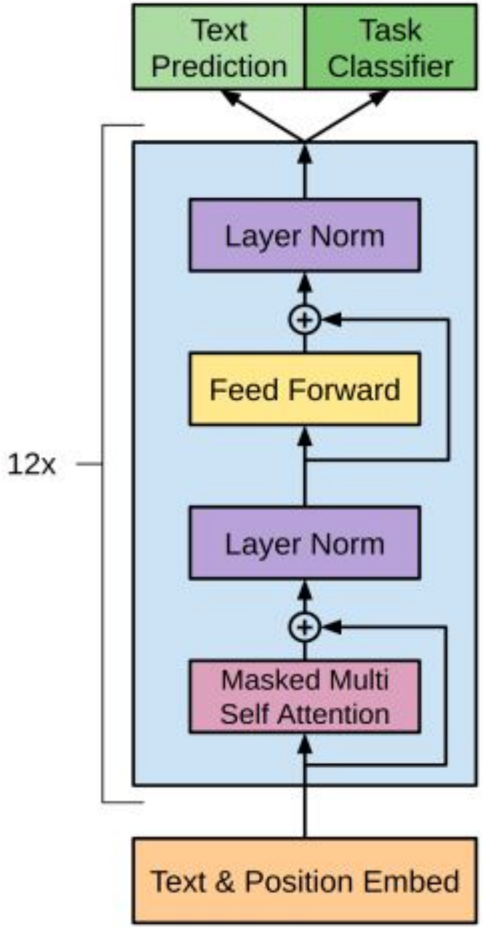
Resources

- https://www.linkedin.com/pulse/gpt-4-explaining-self-attention-mechanism-fatos-ismali/

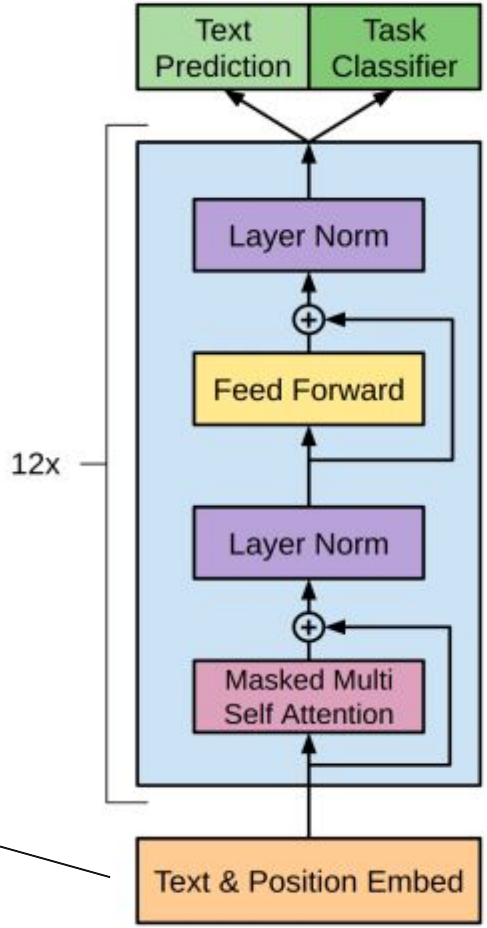
The success of ChatGPT

1. Massive amounts of data.
2. Significant computing power.
3. Word embeddings.
4. Self-attention.
5. Transformers.
6. Reinforcement Learning from Human Feedback (RLHF).

ChatGPT Transformers



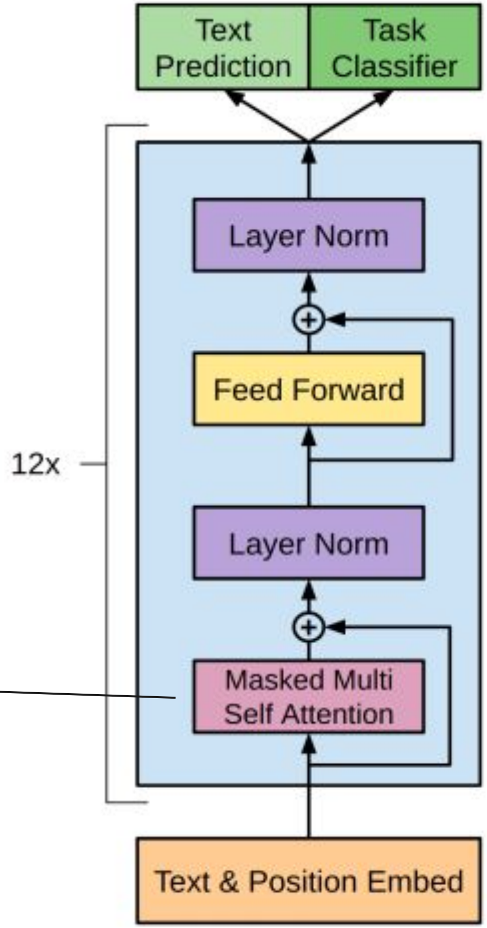
ChatGPT Transformers



Get embedded vectors as input and transform them based on word positioning in the context.

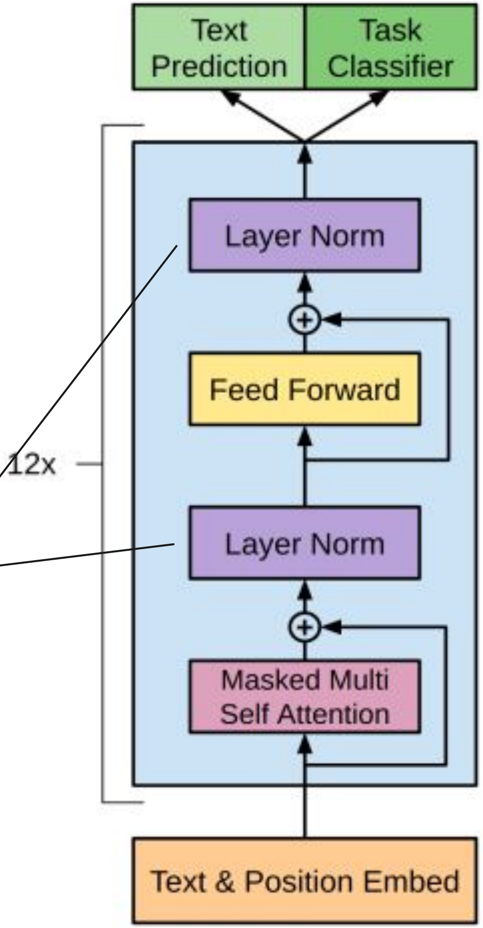
ChatGPT Transformers

Multiple attention layers.



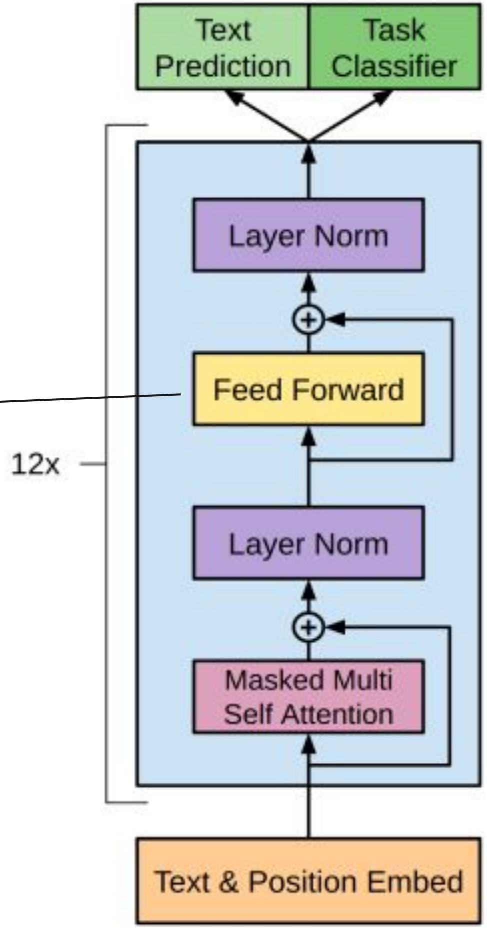
ChatGPT Transformers

Normalization.



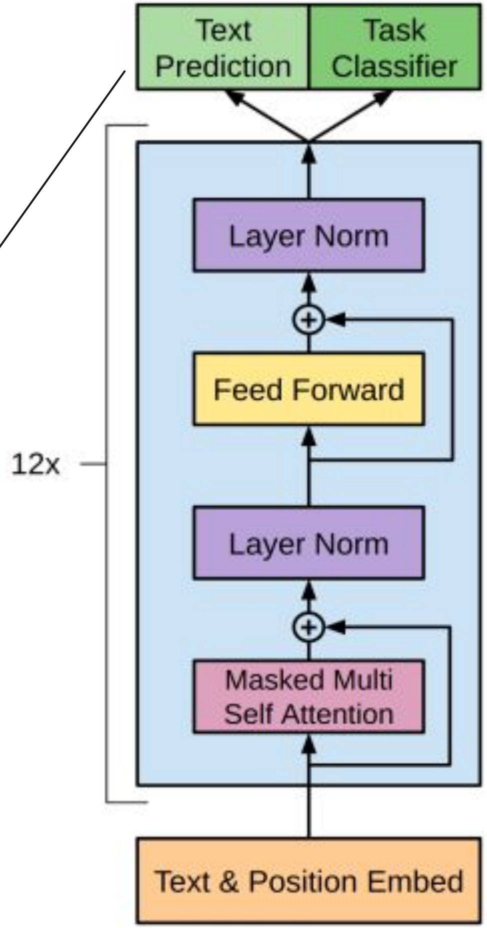
ChatGPT Transformers

Typical NN.



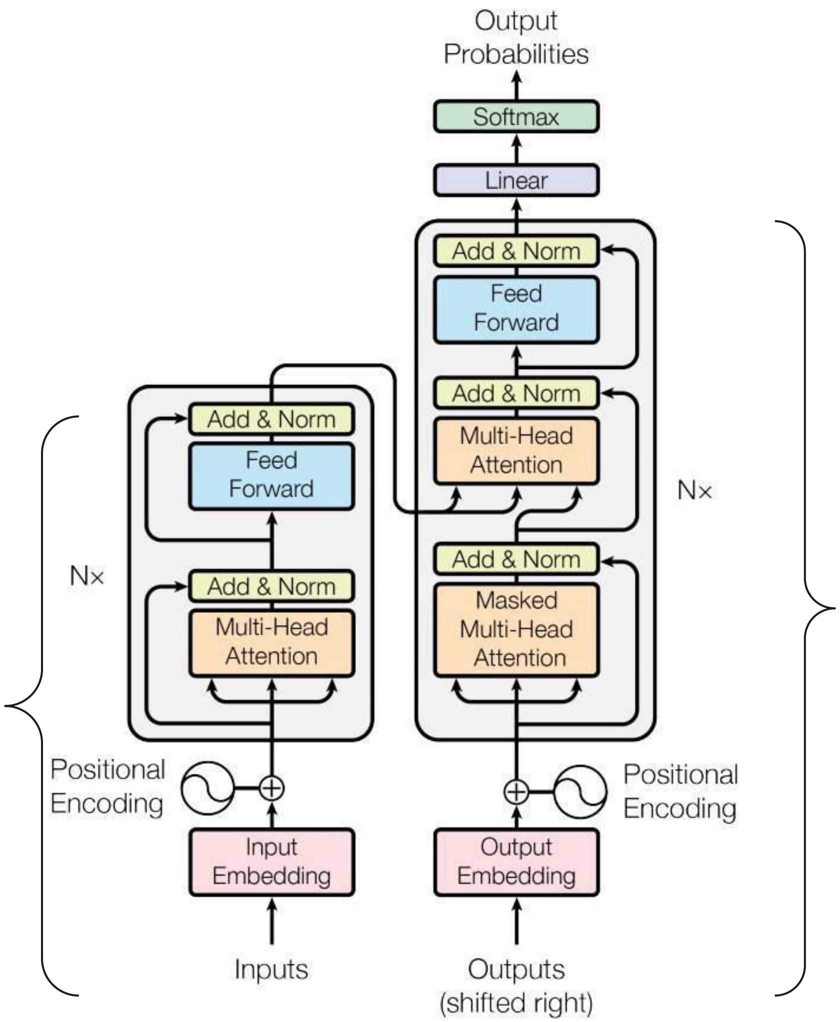
ChatGPT Transformers

Softmax of tokens, predict the most likely next word.



Common Transformer

Encoder:
- Processes input.
- Creates a representation of meaning and context.



Decoder:
- Takes encoder's output as input.
- Generates output.

GPT 3 – Architecture

Transformer layers	96
Token size (vector)	12,288
Context/Sequence length	2,048
Num. heads (total)	96
Parameters (weights)	$175 \cdot 10^9$

GPT 3 – Full Architecture

- https://dugas.ch/artificial_curiosity/img/GPT_architecture/fullarch.png

GPT – Training

	Pre-training	Post-training
Data quantity	~ the whole internet (trillions of words)	Small (milions?)
Data quality	Low	High
Goal	Most “learning” occurs	Makes model usable

*https://docs.google.com/presentation/d/1hQUd3pF8_2Gr2Obc89LKjmHL0DlH-uof9M0yFVd3FA4/mobilepresent?slide=id.g28e2e9aa709_0_121

GPT 3 – Tokens

GPT-3 training data[1]:9

Dataset	# tokens	Proportion within training
Common Crawl	410 billion	60%
WebText2	19 billion	22%
Books1	12 billion	8%
Books2	55 billion	8%
Wikipedia	3 billion	3%

Resources

- https://www.linkedin.com/pulse/gpt-4-explaining-self-attention-mechanism-fatos-ismali/
- https://towardsdatascience.com/all-you-need-to-know-about-attention-and-transformers-in-depth-understanding-part-1-552f0b41d021
- https://docs.google.com/presentation/d/1ZXFIhYczos679r70Yu8vV9uO6B1J0ztzeDxbnBxD1S0/mobilepresent?pli=1&slide=id.g31364026ad_3_2

The success of ChatGPT

1. Massive amounts of data.
2. Significant computing power.
3. Word embeddings.
4. Self-attention.
5. Transformers.
6. Reinforcement Learning from Human Feedback (RLHF).

RLHF

- Trains a model based on human feedback.
- Good human feedback → High reward (low error).
- Bad human feedback → Low reward (high error).

Well-known Architectures using Transformers

- GPT3
- BERT
- T5
- LLaMa
- ...
- https://huggingface.co/models?other=LLM&sort=trending

Huggingface

- Use trained models: https://huggingface.co/learn/nlp-course/chapter1/3
- Transfer learning: https://huggingface.co/docs/transformers/training