

Applied Machine Learning for Business Analytics

Lecture 8: Transformers

Logistic

1. Our TA Xiaohui will deliver two hands-on tutorials
 - a. **Huggingface** - April 5th
 - b. **Langchain** - April 12th
2. Before next Friday, you will receive the feedback on the group project proposal

Attention Is All You Need

Ashish Vaswani*
Google Brain
avaswani@google.com

Noam Shazeer*
Google Brain
noam@google.com

Niki Parmar*
Google Research
nikip@google.com

Jakob Uszkoreit*
Google Research
usz@google.com

Llion Jones*
Google Research
llion@google.com

Aidan N. Gomez* †
University of Toronto
aidan@cs.toronto.edu

Lukasz Kaiser*
Google Brain
lukaszkaizer@google.com

Illia Polosukhin* ‡
illia.polosukhin@gmail.com

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.

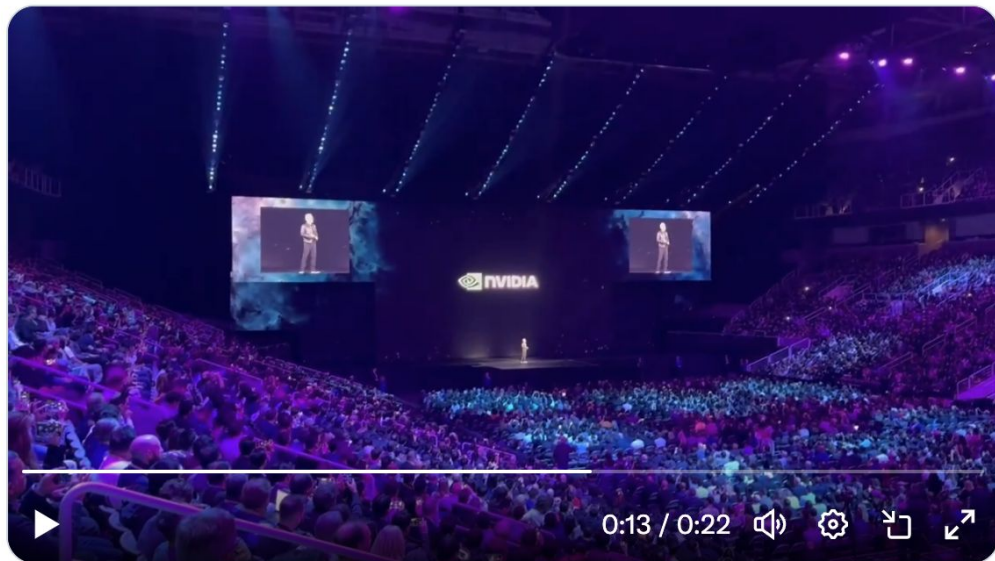


Jim Fan ✓

@DrJimFan

...

Jensen Huang is the new Taylor Swift



4:07 AM · Mar 19, 2024 from San Jose, CA · **511.2K** Views



Transforming AI Panel at GTC 2024

Jensen Huang will host a panel with the authors of "[Attention Is All You Need](#)", a seminal research paper that introduced the Transformer neural network architecture (NeurIPS, 2017)



Jensen Huang
Founder and CEO
NVIDIA



Ashish Vaswani
Co-Founder & CEO
Essential AI



Noam Shazeer
CEO and Co-Founder
Character.AI



Niki Parmar
Co-Founder
Essential AI



Jakob Uszkoreit
CEO
Inception



Llion Jones
Co-Founder and CTO
Sakana AI



Aidan Gomez
Co-Founder and CEO
Cohere



Lukasz Kaiser
Member of Technical Staff
OpenAI



Illia Polosukhina
Co-Founder
NEAR Protocol



Agenda

1. Transformers
2. **Attention** is all you need
 - a. Self-Attention
 - b. Positional Embeddings
 - c. Masked Self-Attention
 - d. Encoder-Decoder Attention
3. Summary

1. Transformers

What is Transformer



Transformer is a **sequence to sequence** model (Encoder and Decoder)

Attention Is All You Need

Ashish Vaswani*
Google Brain
avaswani@google.com

Noam Shazeer*
Google Brain
noam@google.com

Niki Parmar*
Google Research
nikip@google.com

Jakob Uszkoreit*
Google Research
usz@google.com

Llion Jones*
Google Research
llion@google.com

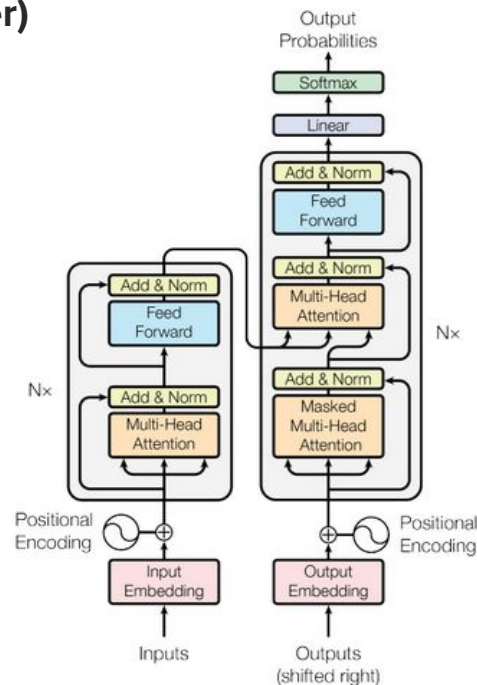
Aidan N. Gomez*[†]
University of Toronto
aidan@cs.toronto.edu

Lukasz Kaiser*
Google Brain
lukaszkaiser@google.com

Illia Polosukhin*[‡]
illia.polosukhin@gmail.com

Abstract

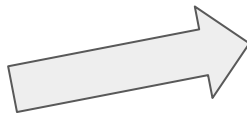
The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.



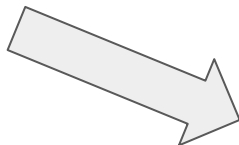
<https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf>

All LLMs so far use transformer architecture

- BERT and GPT are the most representative ones

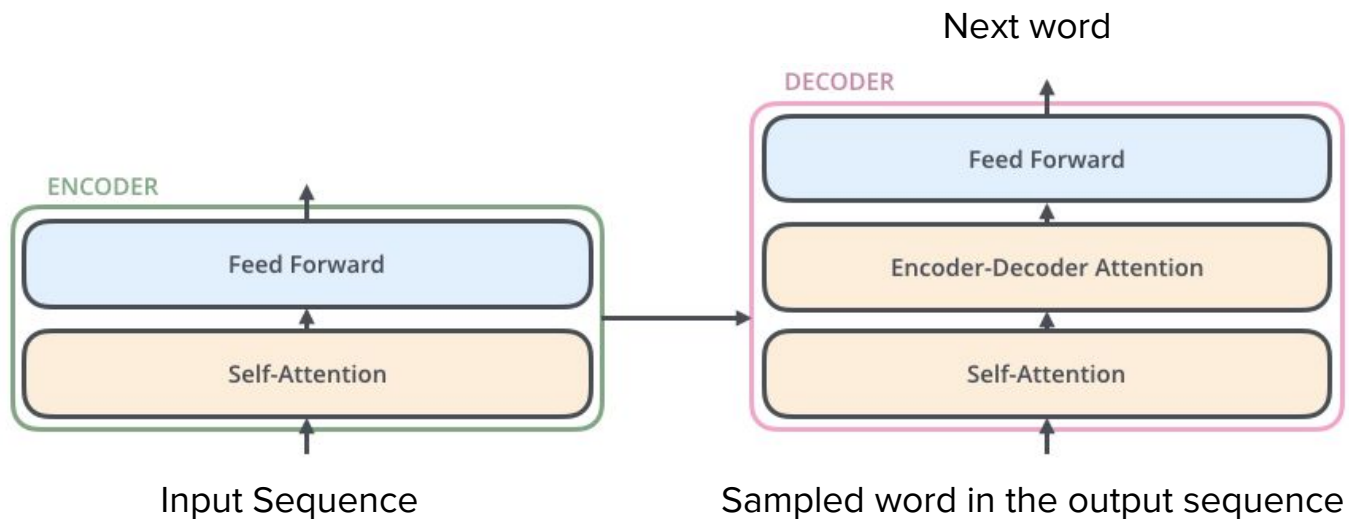


**Bi-directional Encoder
Representations from
Transformers**

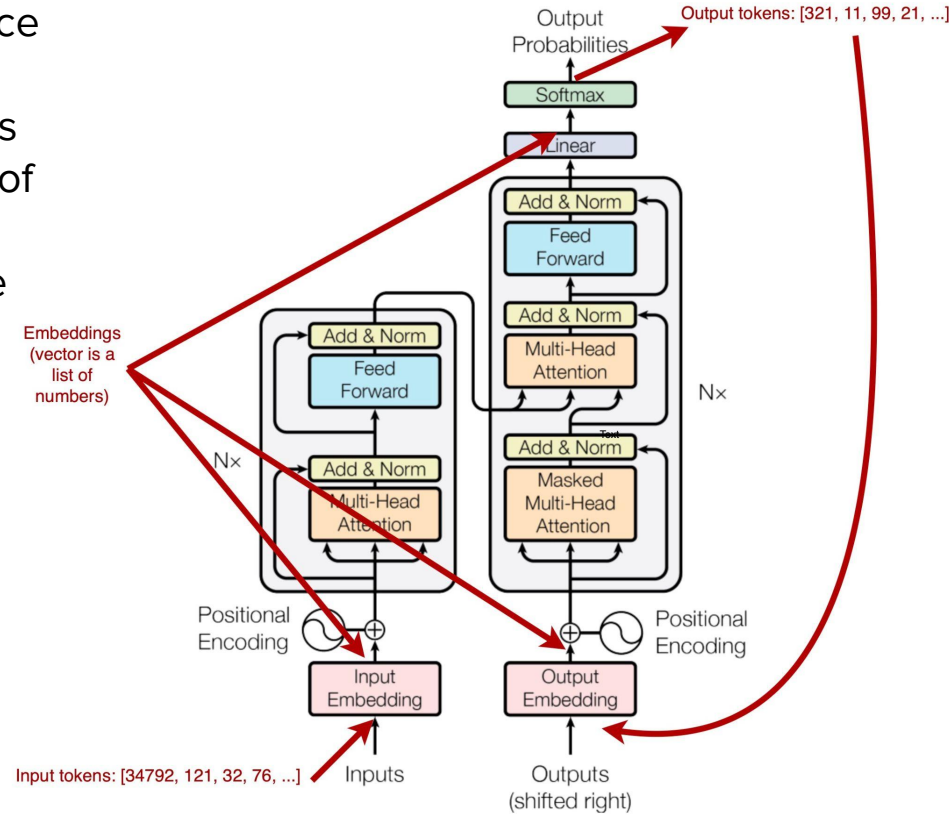


**Generative Pre-trained
Transformers**

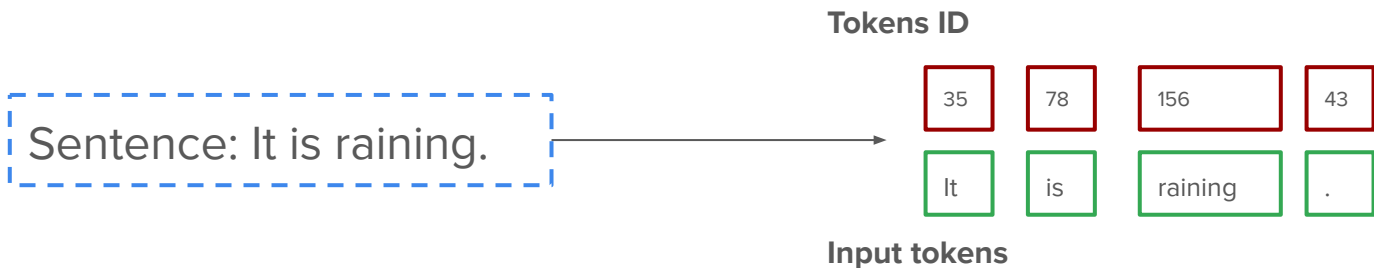
Transformer is solving Seq2Seq



- Input text is encoded with tokenizers to sequence of integers
- Input tokens are mapped to sequence of vectors
- Output vectors can be classified to a sequence of tokens
- Output tokens can then be decoded back to the text

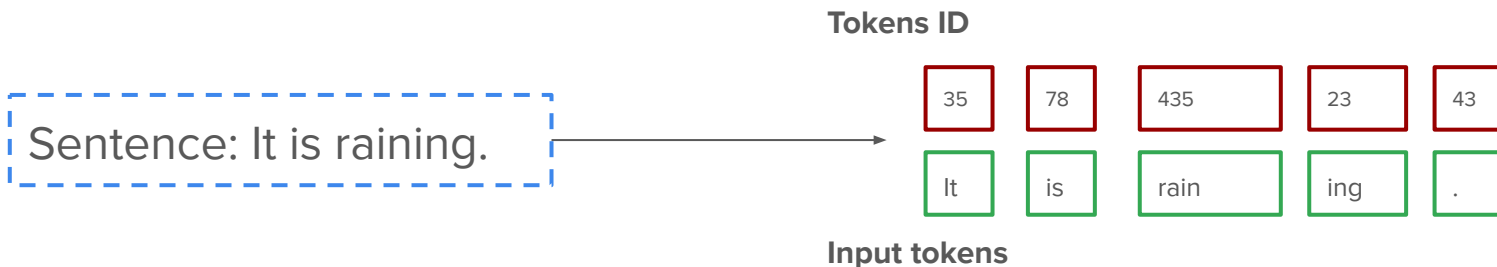


Tokenization: word level



Relies on a predefined vocabulary -> **Out-of-vocabulary issues**

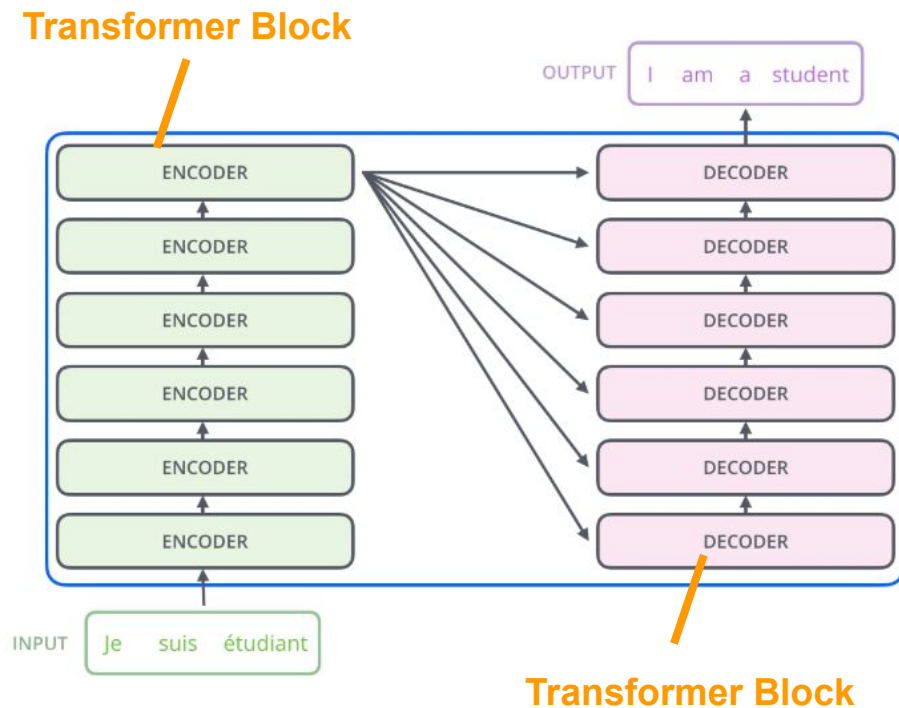
Tokenization: subword level



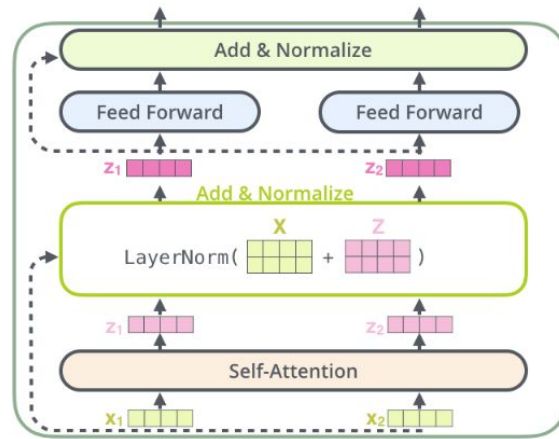
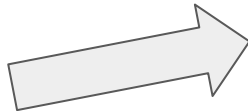
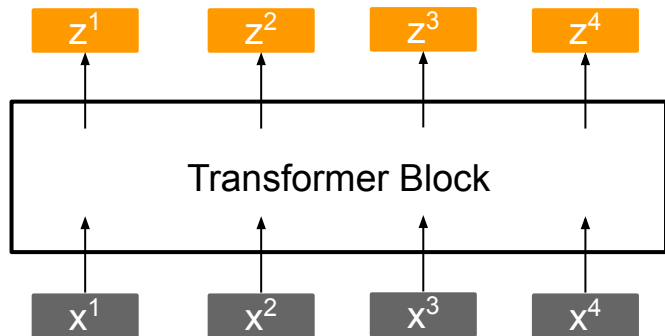
- Can represent out-of-vocabulary words by composing them from subword units
- The subword algorithms have two main modules:
 - A token learner: this takes a corpus as input and creates a vocabulary containing tokens
 - When should we decompose word into subwords and index those subwords
 - A token segmenter
 - Takes a piece of text and segments it into tokens

Transformer

1. Transformer block: operation unit
 - a. Consists of multiple computations
 - b. A sequence of embeddings in
 - c. A sequence of embeddings out
2. Encoder:
 - a. Stack 6 transformer blocks
 - b. Learn representations for the input sequence
3. Decoder:
 - a. Stack another 6 transformer blocks.
 - b. Generate output sequences conditioned on the learned representations from encoder.

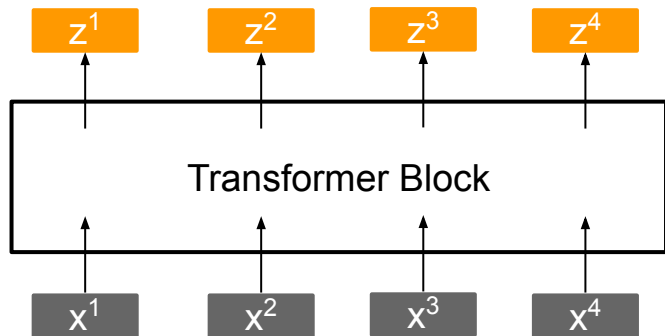


Transformer block in Encoder

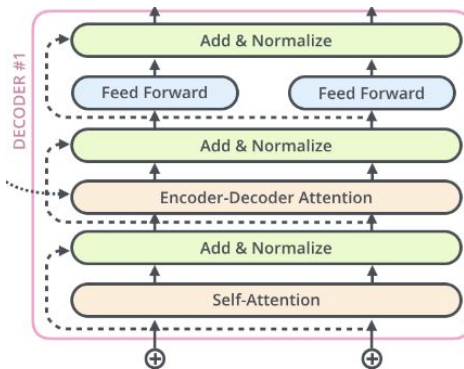
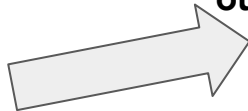


1. Input: A sequence of vectors
2. Output: A sequence of vectors
3. Key Components:
 - a. **Self-attention Layer**
 - b. **Positional Embeddings**
 - c. Residual and Normalization Layer
 - d. Fully-connected Layer

Transformer block in Decoder



Encoder
outputs



1. Input: A sequence of vectors
2. Output: A sequence of vectors
3. Key Components:
 - a. **Masked Self-attention Layer**
 - b. Positional Embeddings
 - c. **Encoder-Decoder Attention**
 - d. Residual and Normalization Layer
 - e. Fully-connected Layer

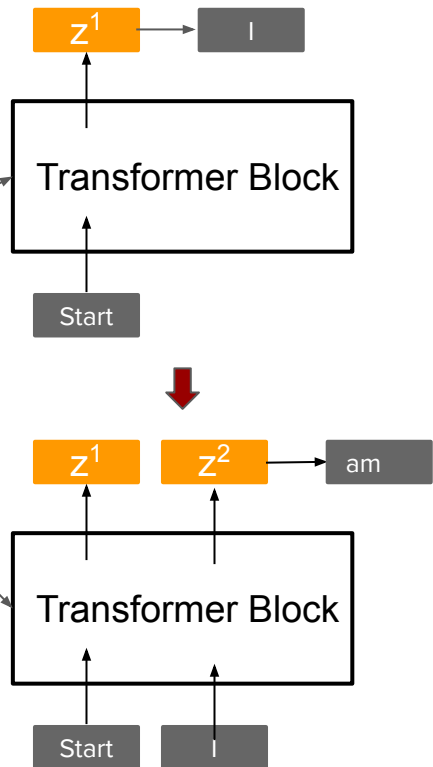
Decoding Process

- The decoder is **autoregressive**
 - Begins with a start token
 - Before the stop token is generated, repeat
 - Take the list of previous outputs with the encoder outputs that contain the attention information from the input
 - Generate the current output

Decoding Process

Encoder Outputs: **Je Suis etudiant**

Until stop symbol is generated



Decoding Process

- Linear Classifier with Final softmax for output probabilities
 - The output of the classifier would be the size of vocabulary
 - After softmax, probability scores between 0 and 1 will be generated
 - Decoding Strategies:
 - Greedy search: the index of the highest probability score would be taken to predict the current word
 - Beam search: takes into account the N most likely tokens
 - Other advanced sampling: <https://deci.ai/blog/from-top-k-to-beam-search-llm-decoding-strategies/>

Which word in our vocabulary
is associated with this index?

am

Get the index of the cell
with the highest value
(argmax)

5

log_probs



↑

Softmax

logits



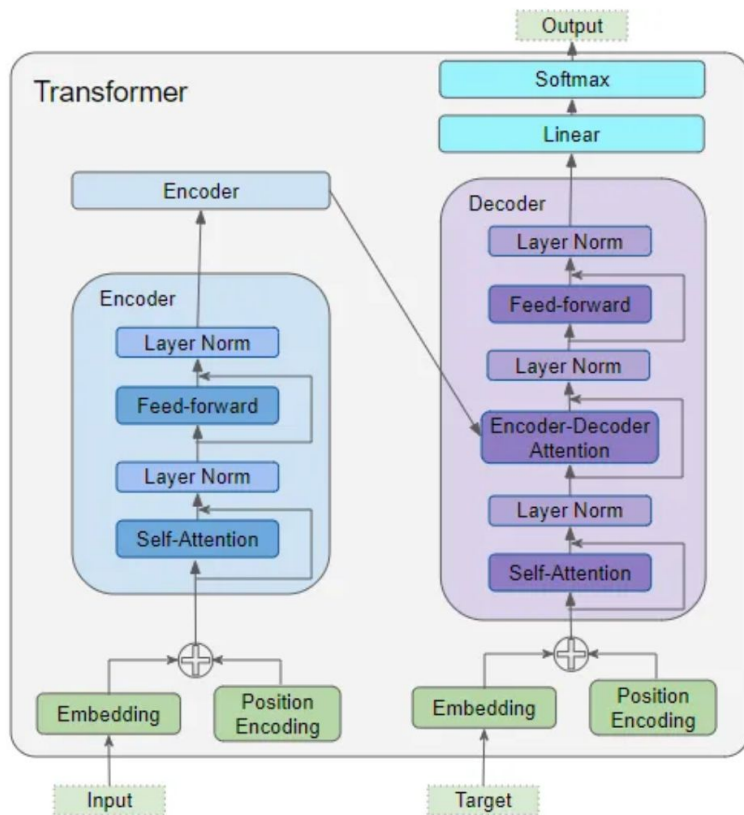
↑

Linear

Decoder stack output



Encoder-Decoder



Three kinds of attention in transformers:

- Self-attention
- Masked self-attention
- Encoder-Decoder attention

Source:

<https://towardsdatascience.com/transformers-explained-visually-part-2-how-it-works-step-by-step-b49fa4a64f34>

2. Attention

Word Embeddings

- **Apple** in two sentences:
 - Sentence 1: My favorite fruit is **apple**
 - Sentence 2: Solution: My favorite brand is **apple**



One embedding has multiple senses

Contextualized Word Embeddings

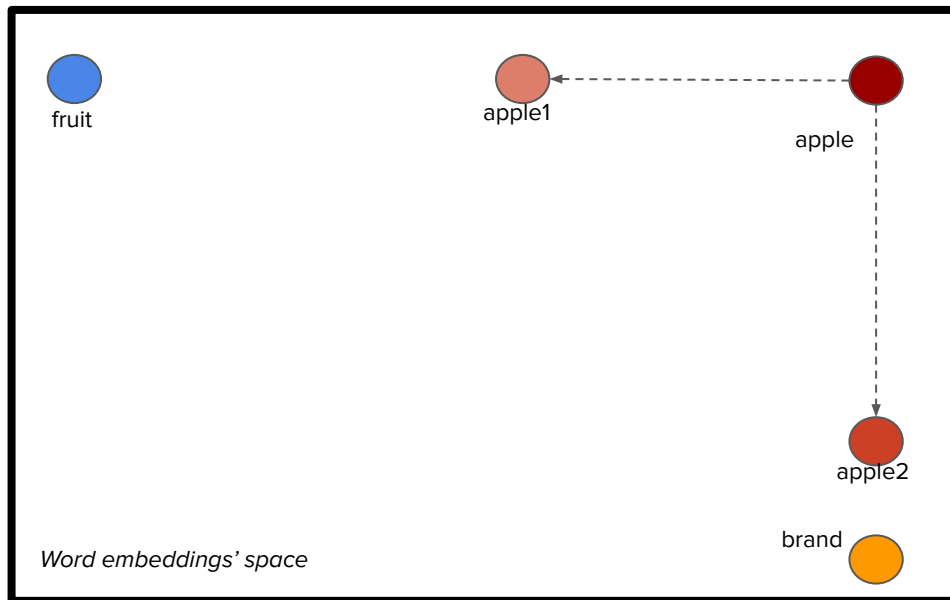
- Telling context in words
 - Sentence 1: My favorite fruit is apple1
 - Sentence 2: Solution: My favorite brand is apple2



From nearby words, we can guess two different meanings of this word (i.e., food and brand)

Contextualized Word Embeddings

- Telling context in words
 - Sentence 1: My favorite **fruit** is **apple1**
 - Sentence 2: Solution: My favorite **brand** is **apple2**



1. In sentence 1, move the word embedding of apple towards the word “fruit”
2. In sentence 2, move the word embedding of apple toward the word “brand”

This is how attention will work

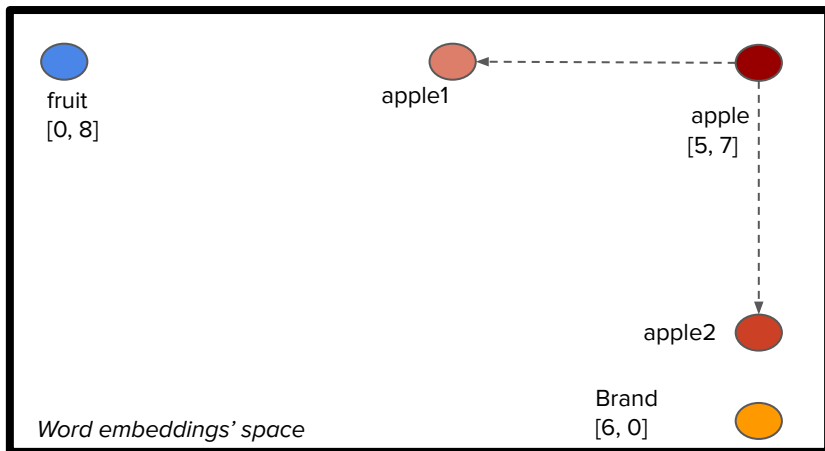
How to move one word closer to another one

- Average two words

- $\text{apple1} = 0.8 \cdot \text{apple} + 0.2 \cdot \text{fruit} = 0.8 \cdot [5, 7] + 0.2 \cdot [0, 8] = [4.0, 7.2]$
- $\text{apple2} = 0.9 \cdot \text{apple} + 0.1 \cdot \text{brand} = 0.9 \cdot [5, 7] + 0.1 \cdot [6, 0] = [5.1, 6.3]$

Similarity/Attention

Embeddings

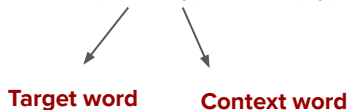


Attention mechanism is able to learn multiple embeddings for the same word in multiple sentences

How to derive similarity

- **Apple** in two sentences:
 - Sentence 1: My favorite fruit is **apple**
 - Sentence 2: My favorite brand is **apple**
- Why we move apple to fruit in sentence 1? Instead of other words as “my” and “is”
- It is based on the similarity!
- Assume every word has its own base vector (as word2vec), the contextualized word embedding of **apple** in the sentence: my favorite fruit is **apple**

$$= \text{Attention}(\text{apple}, \text{my}) * \text{base_vec}(\text{my}) + \text{Attention}(\text{apple}, \text{favorite}) * \text{base_vec}(\text{favorite}) + \text{Attention}(\text{apple}, \text{fruit}) * \text{base_vec}(\text{fruit}) + \text{Attention}(\text{apple}, \text{is}) * \text{base_vec}(\text{is}) + \text{Attention}(\text{apple}, \text{apple}) * \text{base_vec}(\text{apple})$$



How to derive similarity

- Via embeddings, the similarity between two irrelevant words would be zero, while the similarity between the related pair would be high

| | my | favourite | fruit | is | apple |
|-----------|----|-----------|-------|----|-------|
| my | 1 | 0 | 0 | 0 | 0 |
| favourite | 0 | 1 | 0 | 0 | 0 |
| fruit | 0 | 0 | 1 | 0 | 0.25 |
| is | 0 | 0 | 0 | 1 | 0 |
| apple | 0 | 0 | 0.25 | 0 | 1 |

| | my | favourite | brand | is | apple |
|-----------|----|-----------|-------|----|-------|
| my | 1 | 0 | 0 | 0 | 0 |
| favourite | 0 | 1 | 0 | 0 | 0 |
| brand | 0 | 0 | 1 | 0 | 0.11 |
| is | 0 | 0 | 0 | 1 | 0 |
| apple | 0 | 0 | 0.11 | 0 | 1 |

How to derive similarity

- The diagonal entries are all 1
- The similarity between any irrelevant words is 0 (for simplicity)
- The similarity between apple and fruit is 0.25 while the one between apple and brand is 0.11 considering apple is used more often in the same context as fruit

| | my | favourite | fruit | is | apple |
|-----------|----|-----------|-------|----|-------|
| my | 1 | 0 | 0 | 0 | 0 |
| favourite | 0 | 1 | 0 | 0 | 0 |
| fruit | 0 | 0 | 1 | 0 | 0.25 |
| is | 0 | 0 | 0 | 1 | 0 |
| apple | 0 | 0 | 0.25 | 0 | 1 |

| | my | favourite | brand | is | apple |
|-----------|----|-----------|-------|----|-------|
| my | 1 | 0 | 0 | 0 | 0 |
| favourite | 0 | 1 | 0 | 0 | 0 |
| brand | 0 | 0 | 1 | 0 | 0.11 |
| is | 0 | 0 | 0 | 1 | 0 |
| apple | 0 | 0 | 0.11 | 0 | 1 |

How to derive similarity

- Contextualized Target Word = The sum of a product between the similarity between **target** word and **context** word * **context word embeddings**
- We should also normalize the similarity along the sentence (**softmax**)
- Therefore
 - my (in the sentence 1) = **my**
 - apple (in the sentence 1) = $0.2 * \text{fruit} + 0.8 * \text{apple}$
 - apple (in the sentence 2) = ?

| | my | favourite | fruit | is | apple |
|-----------|----|-----------|-------|----|-------|
| my | 1 | 0 | 0 | 0 | 0 |
| favourite | 0 | 1 | 0 | 0 | 0 |
| fruit | 0 | 0 | 0.8 | 0 | 0.2 |
| is | 0 | 0 | 0 | 1 | 0 |
| apple | 0 | 0 | 0.2 | 0 | 0.8 |

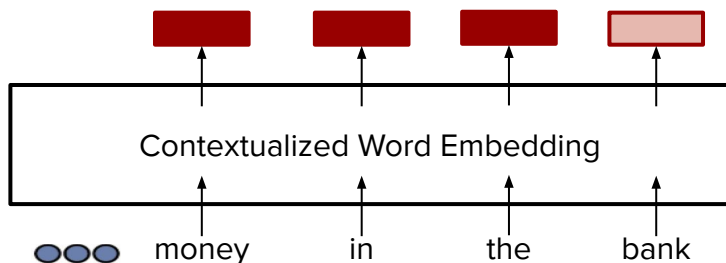
Normalized

| | my | favourite | brand | is | apple |
|-----------|----|-----------|-------|----|-------|
| my | 1 | 0 | 0 | 0 | 0 |
| favourite | 0 | 1 | 0 | 0 | 0 |
| brand | 0 | 0 | 0.9 | 0 | 0.9 |
| is | 0 | 0 | 0 | 1 | 0 |
| apple | 0 | 0 | 0.1 | 0 | 0.1 |

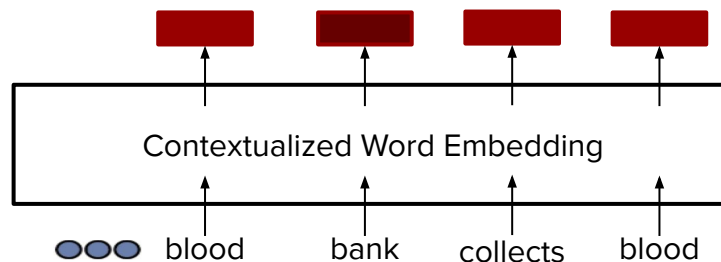
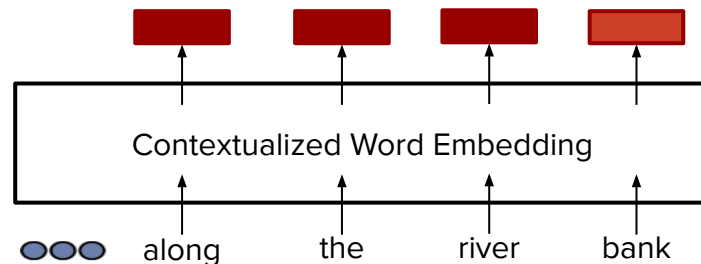
Contextualized Word Embeddings

- Transformers is proposed to learn better feature for NLP data
- The core layer is self-attention layer which can map a sequence of word embeddings to another sequence of word embeddings which is contextualized

Encoded Embeddings



Different Contexts,
Different Encoded Embeddings for bank.



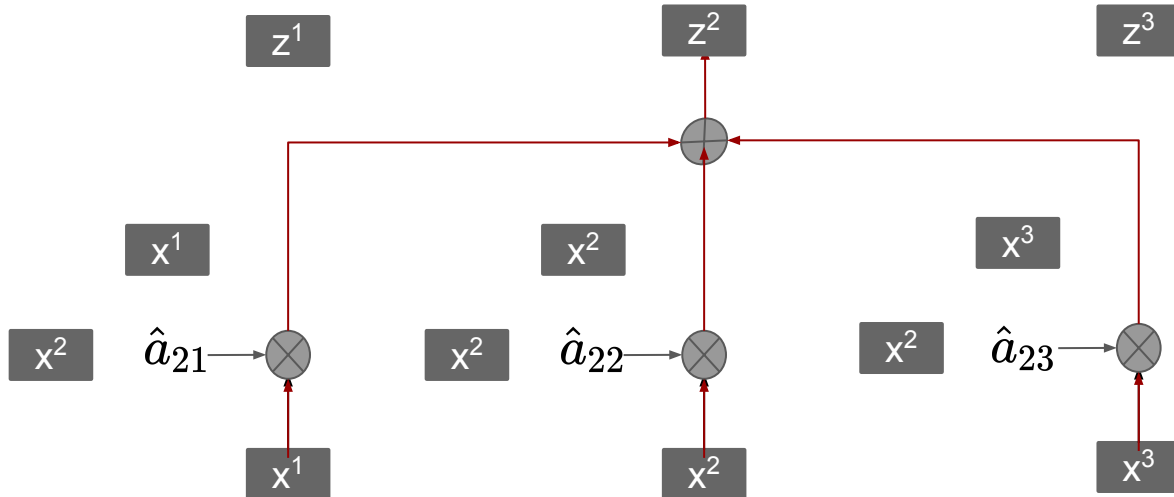
In the case of ChatGPT the generated numbers are probabilities. ChatGPT has a limited vocabulary, and the probabilities indicate how likely each vocabulary word is based on the input word sequence. ChatGPT has a **limited reading range**, and the input sequence has a maximum length of about **3000** words, broken into 4000 sub-word tokens. Once ChatGPT generates a word, it adds that word to the input sequence, and generates a new word. This process continues until it produces a special word called a “stop” token, or it hits a preset word limit.

Why the reading range is limited?

2.1 Self-Attention

Basic Self-Attention

- A sequence-to-sequence operation taking a sequence of vectors in and generate a sequence of vectors out
 - $[x_1, x_2, x_3] \rightarrow [z_1, z_2, z_3]$
- Relating different positions of the input sequence in order to compute the representation



$$z^i = \sum_j \hat{a}_{ij} x^j$$

$$a_{ij} = (x^i)^T (x^j)$$

$$\hat{a}_{ij} = \frac{e^{a_{ij}}}{\sum_j e^{a_{ij}}}$$

Basic Self-Attention

Sentence i: My favourite fruit is apple

| | my | favourite | fruit | is | apple |
|-----------|----|-----------|-------|----|-------|
| my | 1 | 0 | 0 | 0 | 0 |
| favourite | 0 | 1 | 0 | 0 | 0 |
| fruit | 0 | 0 | 1 | 0 | 0.25 |
| is | 0 | 0 | 0 | 1 | 0 |
| apple | 0 | 0 | 0.25 | 0 | 1 |

New Word Index

my_i

favourite_i

fruit_i

is_i

apple_i

Attention Step

my

favourite

$0.8 \cdot \text{fruit} + 0.2 \cdot \text{apple}$

is

$0.2 \cdot \text{fruit} + 0.8 \cdot \text{apple}$

Basic Self-Attention

- There are no model parameters. It is totally determined by the embedding layer
 - Solution: introduce model parameters -> using three sets of embeddings to get contextualized embeddings
- Self attention is permutation equivariant. It ignores the order information.
 - Solution: add positional embeddings

Self-Attention Layer

Step 1: Generate **query**, **key**, and **value** vector for the **input** vector at each time step.

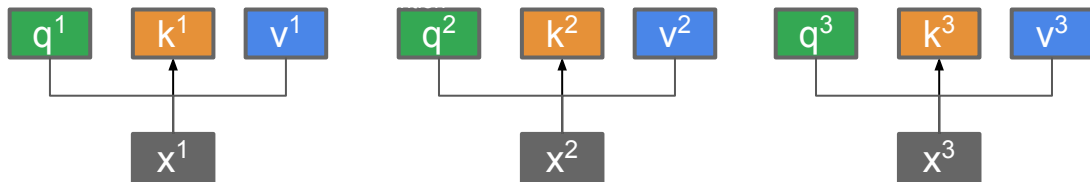
q Query (to match others): $q^i = W^q x^i$

k Key (to be matched): $k^i = W^k x^i$

v Value (representation): $v^i = W^v x^i$

Model parameters are introduced here.

In practice, bias vectors may be added to the product of matrix multiplication



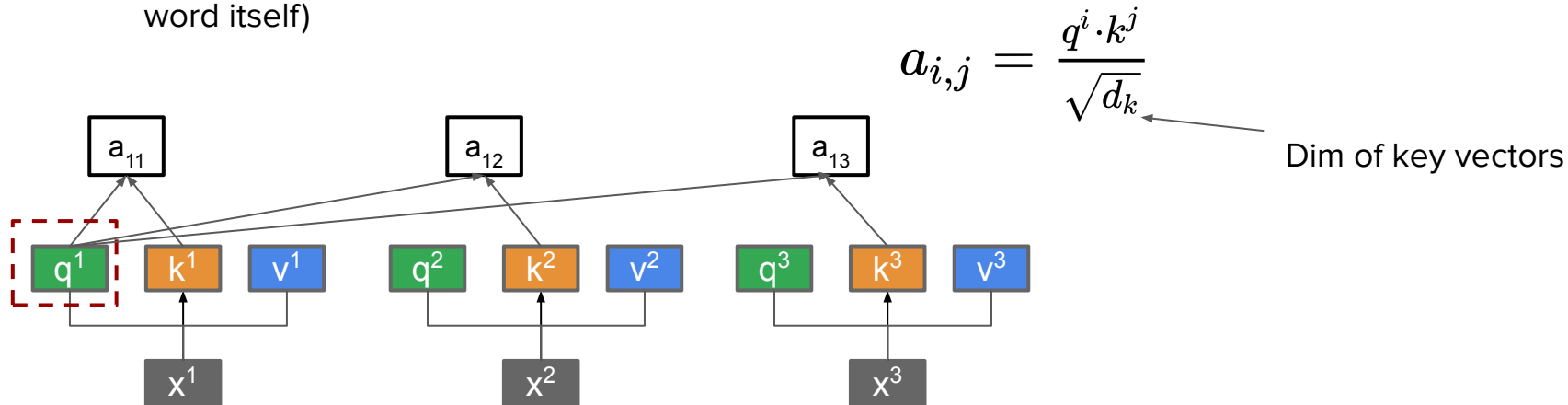
Word embeddings

Self-Attention Layer

Step 2: Compute attention scores using query vectors and key vectors

To encode the i -th word in the sequence, we need to compute the attention scores between this i -th word and all the words in the sequence.

1. Pick the query vector from the i -th word: q^i
2. Attention score computation between q^i and all key vectors of the nearby words (including the target word itself)

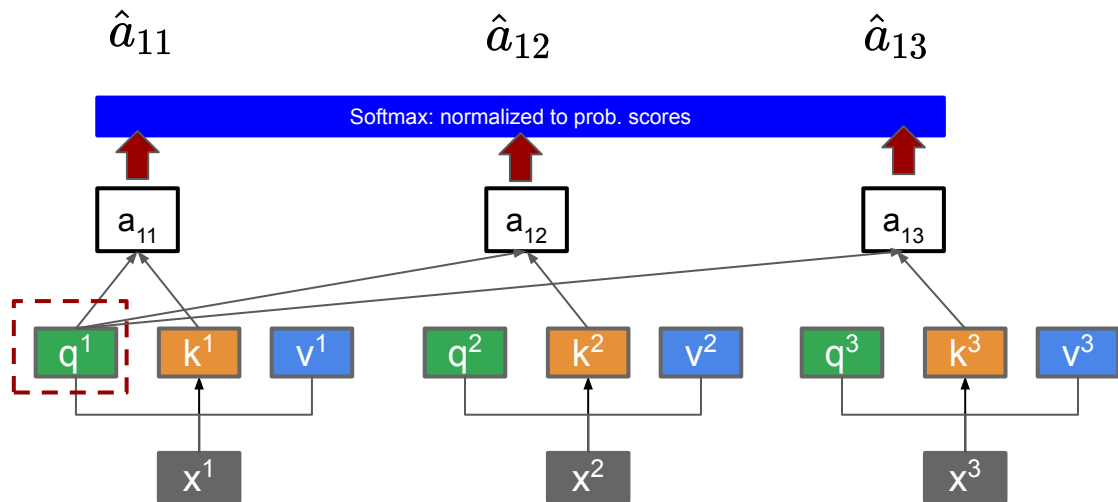


Word vectors

Self-Attention Layer

Step 3: Fed unscaled attention scores into softmax layers

$$\hat{a}_{1i} = \frac{e^{a_{1i}}}{\sum_j e^{a_{1j}}}$$



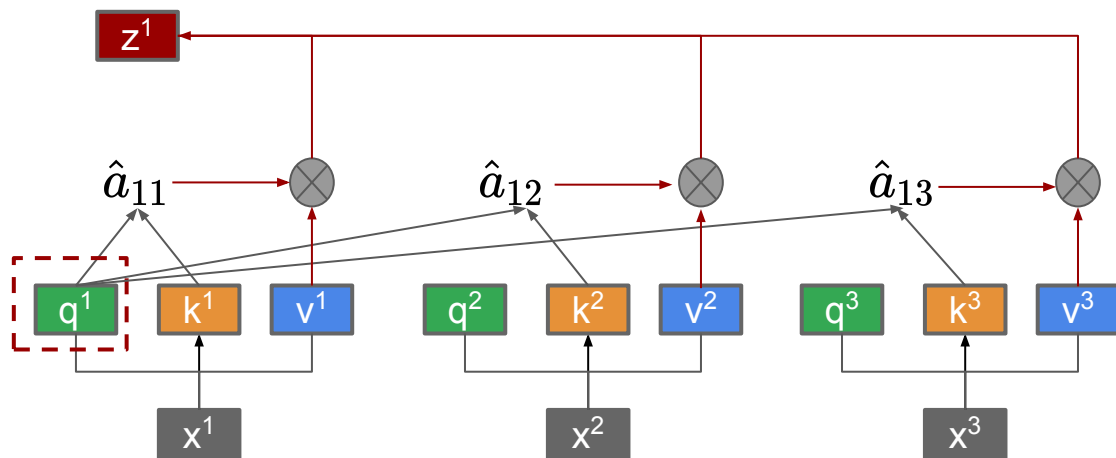
Word vectors

Self-Attention Layer

Step 4: Take the sum of all the value vectors weighted by the attention scores.

Encoded vector for
the first element

$$z^1 = \sum_i \hat{a}_{1i} v^i$$



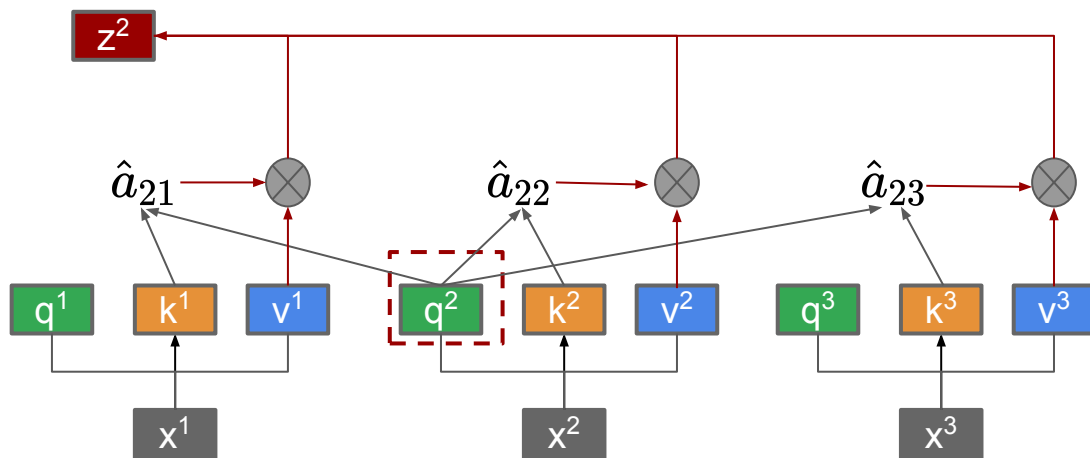
Word vectors

Self-Attention Layer

Step 5: All elements in input sequence x^i will be encoded into new vectors z^i

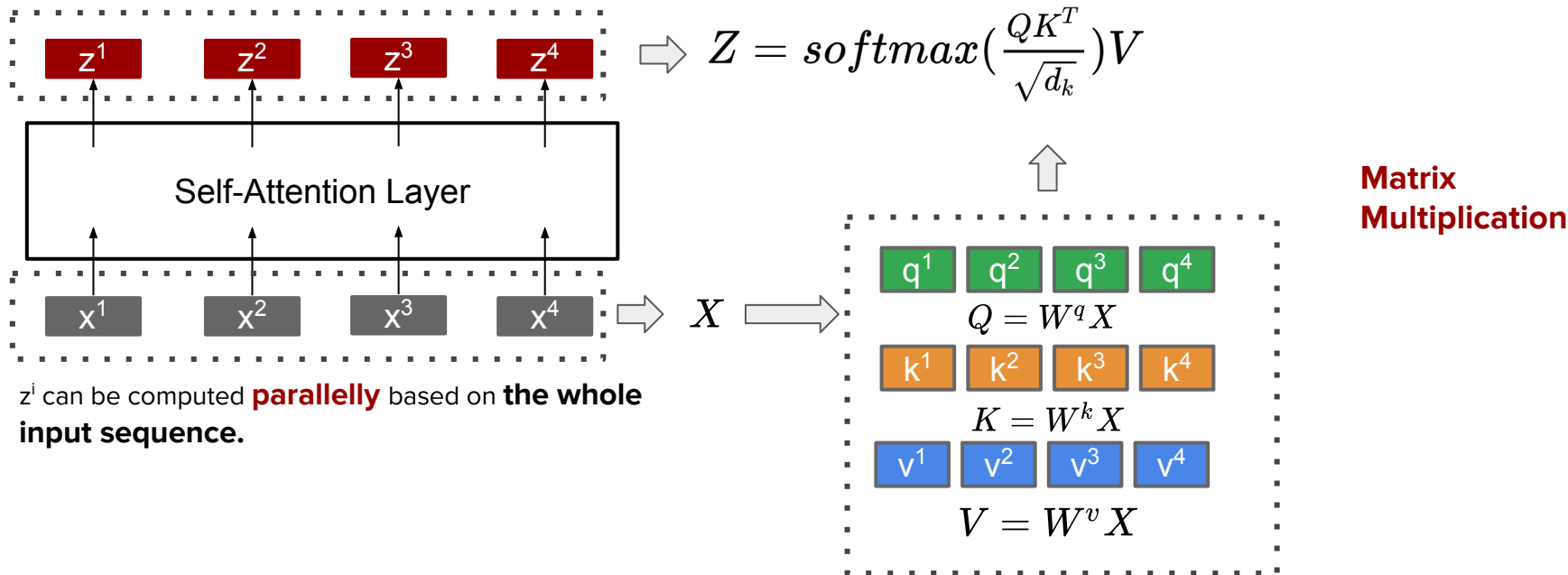
Encoded vector for
the second element

$$z^2 = \sum_i \hat{a}_{2i} v^i$$



Word vectors

Matrix Formulation

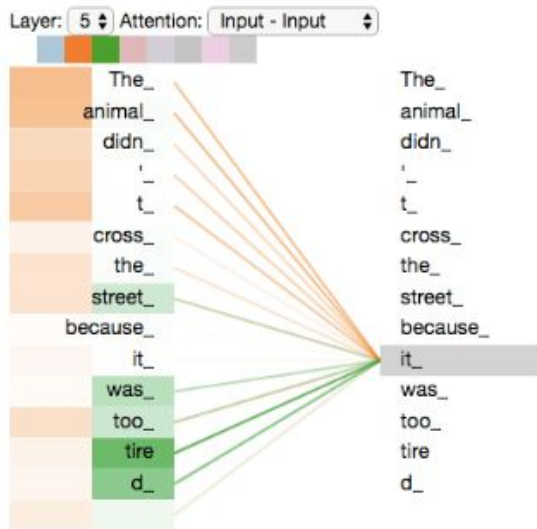


Multi-head Self-Attention

- Model parameters: W^k , W^q , W^v specific one kind of attention
- Multi-head means separate W^k , W^q , W^v matrices
 - Expands the model's ability to focus on different positions
 - Gives the attention layer multiple “representation subspaces”

High attention from one head

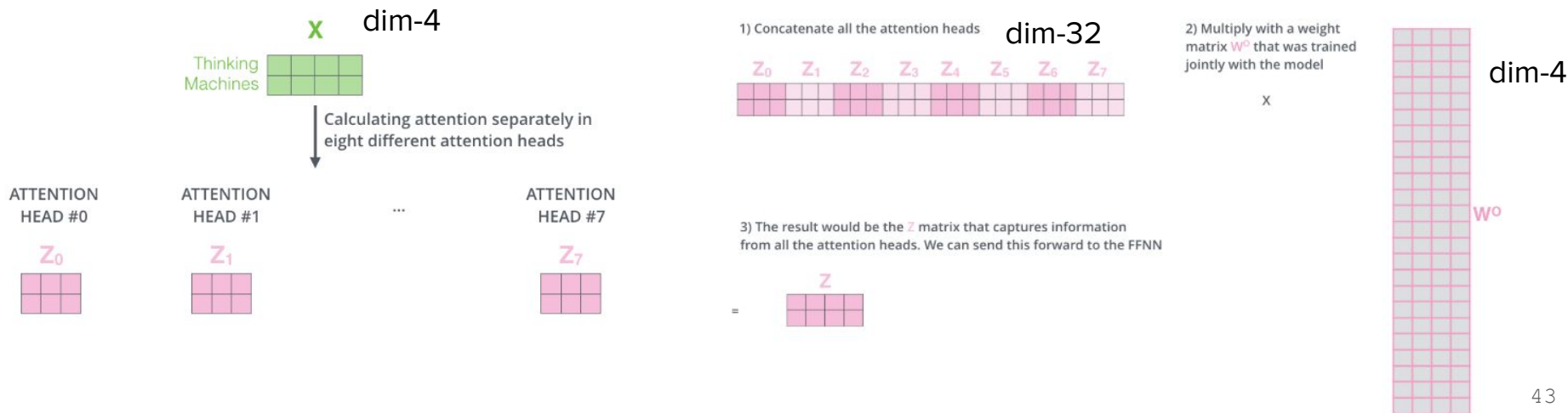
High attention from another head



Head here is similar to the filter in convolutional layer

Multi-head Self-Attention

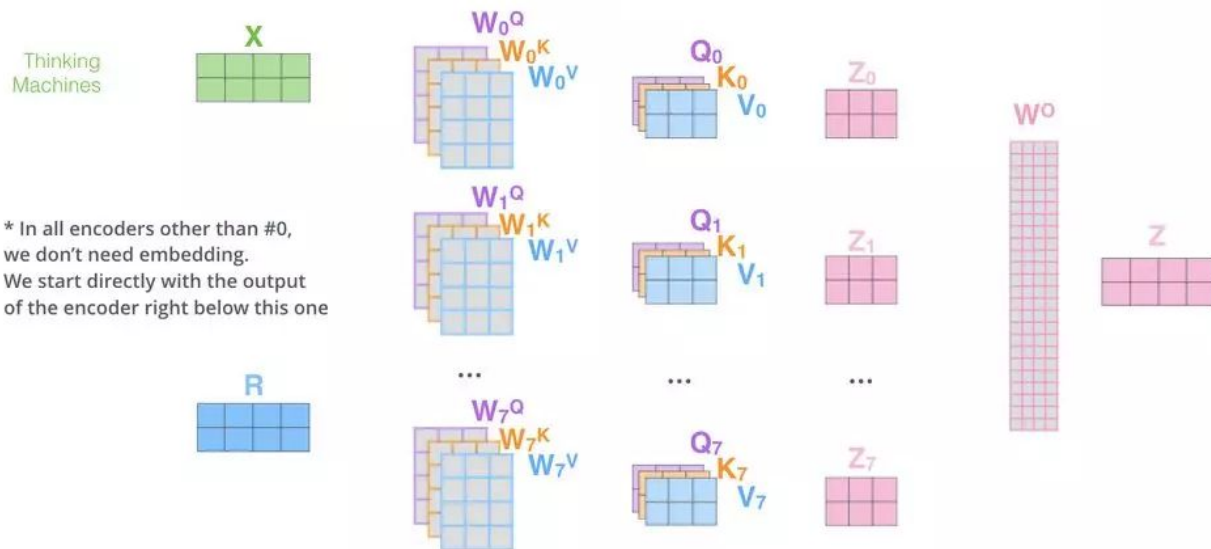
- If the layer has k heads, the output would be k sets of embeddings
- We need to reduce the dimensionality by concatenating and projection into the low dimensional
 - For example, as below: $4 \rightarrow 32 \rightarrow 4$



Self-Attention Layer

That's pretty much all there is to multi-headed self-attention. It's quite a handful of matrices, I realize. Let me try to put them all in one visual so we can look at them in one place

- 1) This is our input sentence*
- 2) We embed each word*
- 3) Split into 8 heads. We multiply X or R with weight matrices
- 4) Calculate attention using the resulting $Q/K/V$ matrices
- 5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer



MultiHeadAttention in Keras

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.layers import MultiHeadAttention
```

```
target = tf.keras.Input(shape=[6, 16])
```

assume it is a sentence of 6 words. Then, each word has a

```
layer = MultiHeadAttention(num_heads=1, key_dim=2)
output_tensor, attention_scores = layer(target, target, return_attention_scores=True)
print(output_tensor.shape)
print(attention_scores.shape)
```

```
(None, 6, 16)
```

```
(None, 1, 6, 6)
```

```
for matrix in layer.weights:
    print(matrix.shape)
```

```
(16, 1, 2)
```

```
(1, 2)
```

```
(16, 1, 2)
```

```
(1, 2)
```

```
(16, 1, 2)
```

```
(1, 2)
```

```
(1, 2, 16)
```

```
(16,)
```

MultiHeadAttention in Keras

```
layer = MultiHeadAttention(num_heads=3, key_dim=2)
target = tf.keras.Input(shape=[6, 16])
output_tensor, attention_scores = layer(target, target, return_attention_scores=True)
print(output_tensor.shape)
print(attention_scores.shape)
```

(None, 6, 16)

(None, 3, 6, 6)

```
for matrix in layer.weights:
    print(matrix.shape)
```

(16, 3, 2)

(3, 2)

(16, 3, 2)

(3, 2)

(16, 3, 2)

(3, 2)

(3, 2, 16)

(16,)

MultiHeadAttention in Keras

If we change the key_dim from 2 to 5?

```
layer = MultiHeadAttention(num_heads=3, key_dim=2)
target = tf.keras.Input(shape=[6, 16])
output_tensor, attention_scores = layer(target, target, return_attention_scores=True)
print(output_tensor.shape)
print(attention_scores.shape)
```

```
(None, 6, 16)
(None, 3, 6, 6)
```

```
for matrix in layer.weights:
    print(matrix.shape)
```

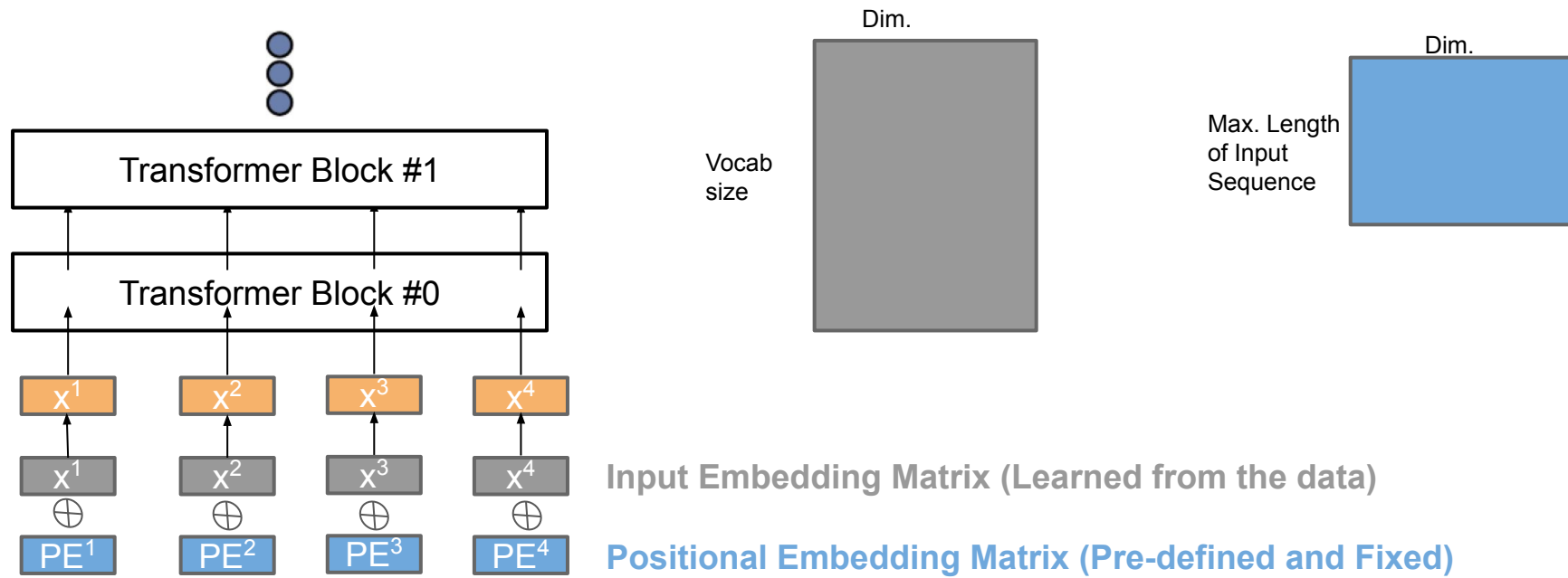
```
(16, 3, 2)
(3, 2)
(16, 3, 2)
(3, 2)
(16, 3, 2)
(3, 2)
(3, 2, 16)
(16,)
```

https://github.com/rz0718/BT5153_2024/blob/main/codes/lab_lecture08/Attention_Layer_in_Keras.ipynb

2.2 Position Embedding

Positional Embeddings

- No position information in self-attention
- Positional Embeddings: each position has a unique positional vector $PE(pos)$
 - Add this vector to each input embeddings
 - Expands the model's ability to focus on different positions.



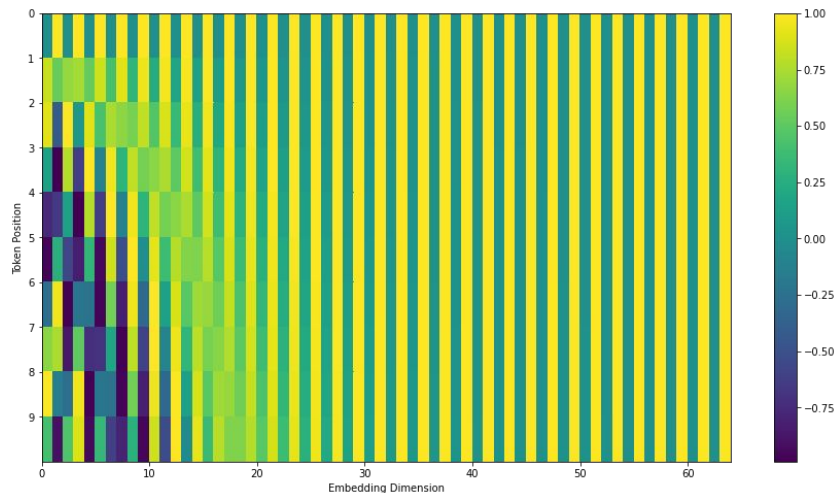
Positional Embeddings

- The equation in the original paper:

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{model}})$$

$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{model}})$$

Core idea: using fixed weights which encode information related to a specific position of a token in a sentence



More details: https://kazemnejad.com/blog/transformer_architecture_positional_encoding/

2.3 Masked Self-Attention

Masked Self-Attention

- This is the attention layer used to compute the dependency among the target words
- Since the sequence is generated word by word, we need to prevent it from conditioning to the future tokens

Target Self Attention Score

| | I | am | a | student |
|---------|-----|-------|-------|---------|
| I | 0.7 | 0.1 ✖ | 0.1 ✖ | 0.1 ✖ |
| am | 0.1 | 0.6 | 0.2 ✖ | 0.1 ✖ |
| a | 0.1 | 0.3 | 0.6 | 0.1 ✖ |
| student | 0.1 | 0.3 | 0.3 | 0.3 |



Look-ahead Bias

| | I | am | a | student |
|---------|---|------|------|---------|
| I | 0 | -inf | -inf | -inf |
| am | 0 | 0 | -inf | -inf |
| a | 0 | 0 | 0 | -inf |
| student | 0 | 0 | 0 | 0 |



Masked Self Attention Score

| | I | am | a | student |
|---------|-----|------|------|---------|
| I | 0.7 | -inf | -inf | -inf |
| am | 0.1 | 0.6 | -inf | -inf |
| a | 0.1 | 0.3 | 0.6 | -inf |
| student | 0.1 | 0.3 | 0.3 | 0.3 |

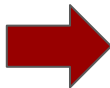
Masked Self-Attention

- Add look-ahead Mask matrix
- Apply softmax
 - The negative infinities would become zero after softmax
 - For example, the attention score for “a”
 - has values for itself and all words before it
 - Zero for the word “student”

Masked Self Attention Score

| | I | am | a | student |
|---------|-----|------|------|---------|
| I | 0.7 | -inf | -inf | -inf |
| am | 0.1 | 0.6 | -inf | -inf |
| a | 0.1 | 0.3 | 0.6 | -inf |
| student | 0.1 | 0.3 | 0.3 | 0.3 |

softmax



Masked Self-Attention Scores

| | I | am | a | student |
|---------|------|------|------|---------|
| I | 1 | 0 | 0 | 0 |
| am | 0.37 | 0.62 | 0 | 0 |
| a | 0.26 | 0.3 | 0.43 | 0 |
| student | 0.21 | 0.26 | 0.26 | 0.26 |

2.4 Encoder-decoder Attention

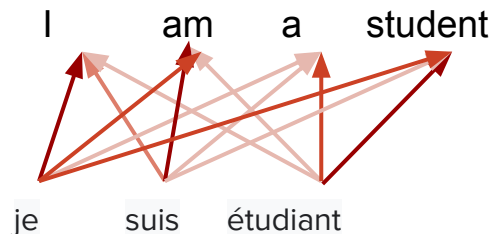
Encoder-decoder attention

Attention in decoder layer:

1. Attention vectors: a vector of importance weights (measure the interaction between each target word with each input word)

Output

Input



→ High Attention
→ Low Attention

2. The target is approximated by the sum of their input values weighted by the attention scores.

$$\text{Vec}_{\text{student}} = 0.15 * \text{Vec}_{\text{je}} + 0.05 * \text{Vec}_{\text{suis}} + 0.8 * \text{Vec}_{\text{etudiant}}$$

Encoder-Decoder attention Layer

Different from Self attention layer

1. Generate **query** vector for the generated output sequence (from itself: Decoder)
2. Generate **key** and **value** vector for the **input** sequence at each time step (from Encoder)

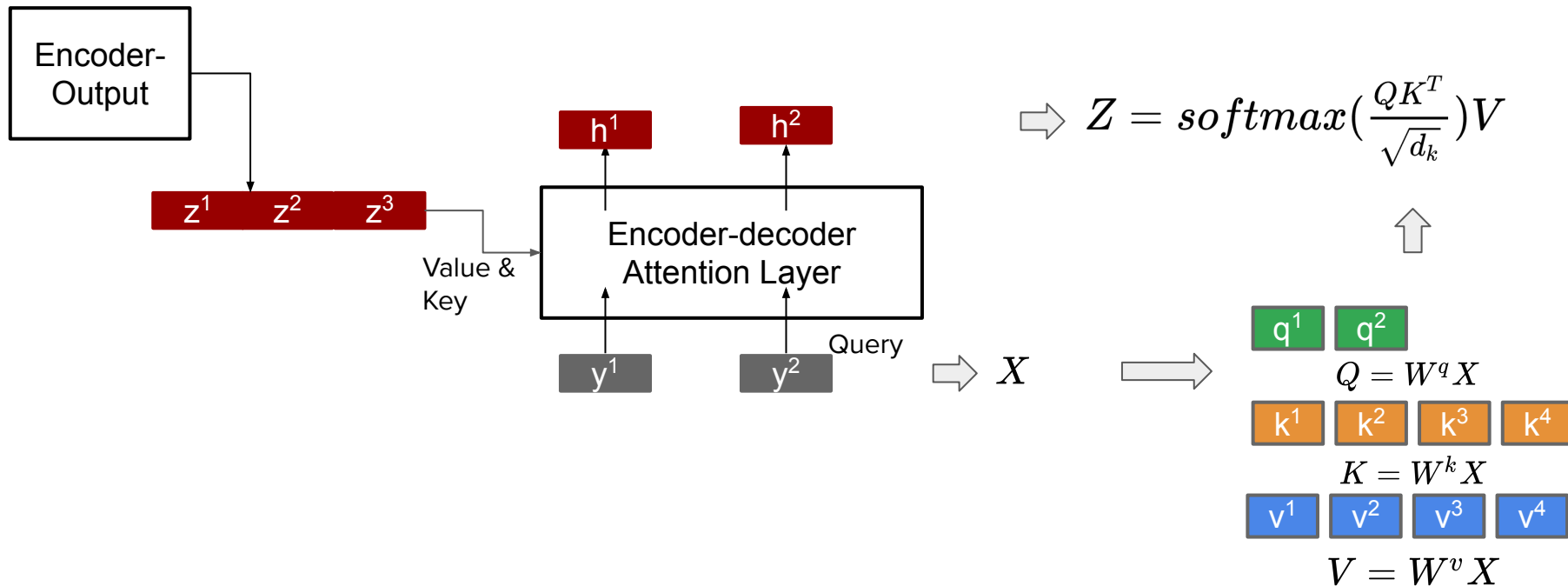
Self Attention Score n

| | je | suis | eludiant |
|----------|----|------|----------|
| je | | | |
| suis | | | |
| eludiant | | | |

Encoder-decoder Attention Score

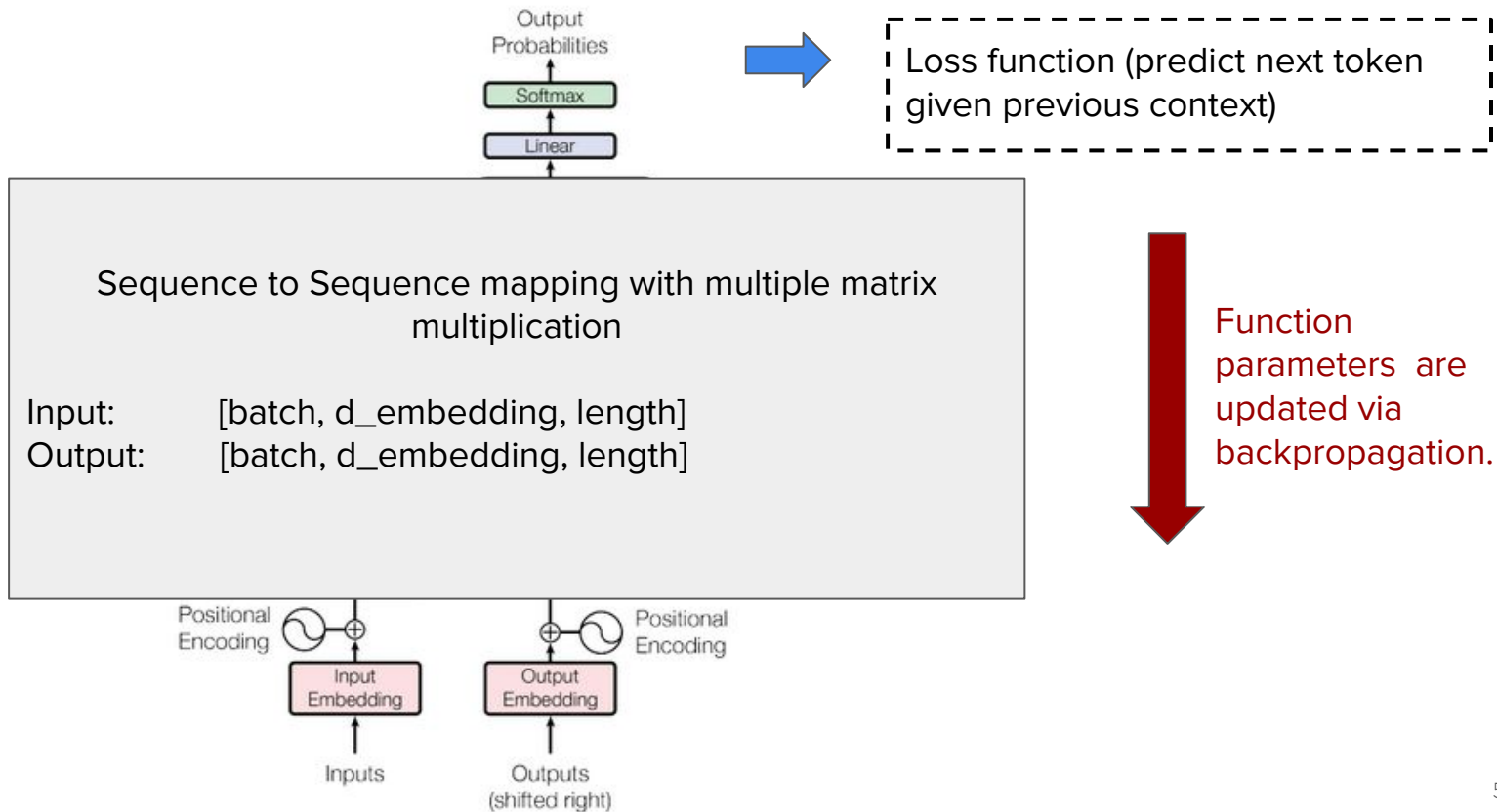
| | I | am | a | student |
|----------|---|----|---|---------|
| je | | | | |
| suis | | | | |
| eludiant | | | | |

Matrix Formulation



3. Summary

A “functional” viewpoint on Transformer



Transformers is replacing RNN and CNN

- Compared to Transformers, RNN
 - can not be trained in parallel
 - suffers from long dependency issues
- Compared to Transformers, CNN
 - is unable to capture all possible combinations of words (filter size is predefined)
- Compared to the previous NN, Transformers
 - **Non sequential:** the input sequence are processed as a whole
 - **Self Attention:** contextualized word embeddings
 - **Positional embeddings:** a better way than recurrence to capture order information

BLOG

Transformer: A Novel Neural Network Architecture for Language Understanding

THURSDAY, AUGUST 31, 2017

Posted by Jakob Uszkoreit, Software Engineer, Natural Language Understanding

Source:

<https://blog.research.google/2017/08/transformer-novel-neural-network.html>

Must Read !!

[The illustrated transformer](#) (the source of the awesome visualizations)



Implementations of Transformers

- [Build Transformer from Scratch](#)
- [Keras Implementation](#)

Next Class: LLM