344.063/163 KV Special Topic:
Natural Language Processing with Deep Learning
Transformers

Navid Rekab-Saz

navid.rekabsaz@jku.at

Institute of Computational Perception

JOHANNES KEPLER
UNIVERSITY LINZ

Institute of
Computational
Perception

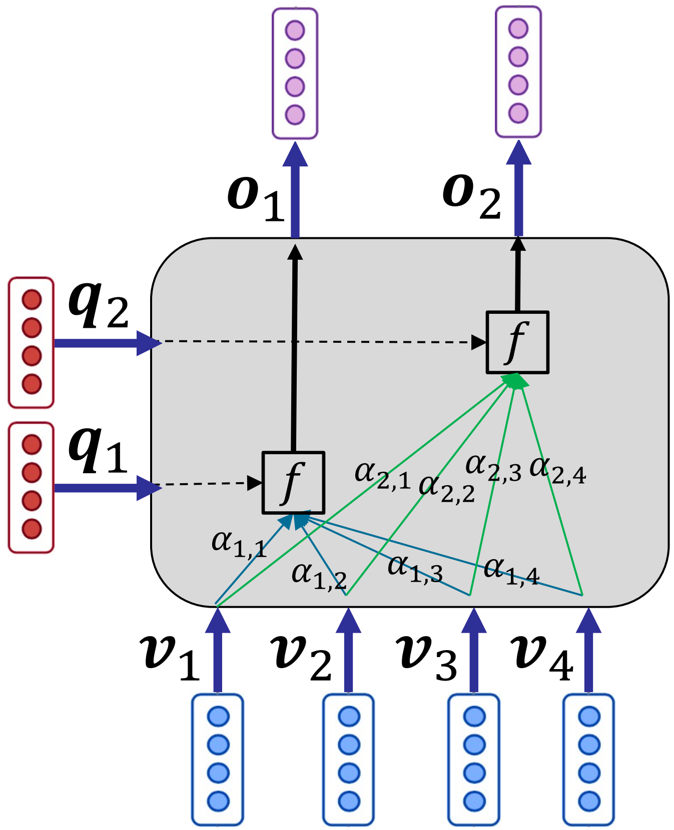
Agenda

- Transformers
  - Transformer encoder
  - Transformer decoder
- seq2seq with Transformers

Agenda

- **Transformers**
  - **Transformer encoder**
  - **Transformer decoder**
- seq2seq with Transformers

Attentions! – recap



$\alpha_{i,j}$ is the attention of query $\boldsymbol{q}_i$ on value $\boldsymbol{v}_j$

$\boldsymbol{\alpha}_i$ is the vector of attentions of query $\boldsymbol{q}_i$ on value vectors $\boldsymbol{V}$

$\boldsymbol{\alpha}_i$ <u>is a probability distribution</u>

f is the attention function

Attention Networks formulation – recap

- Given the query vector $\boldsymbol{q}_i$, an attention network assigns attention $\alpha_{i,j}$ to each value vector $\boldsymbol{v}_j$ using attention function f:

$$\alpha_{i,j} = f(\boldsymbol{q}_i, \boldsymbol{v}_j)$$

where $\boldsymbol{\alpha}_i$ forms a probability distribution over vector values:

$$\sum_{j=1}^{|\boldsymbol{V}|} \alpha_{i,j} = 1$$

- The output regarding each query is the weighted sum of the value vectors using attentions as weights:

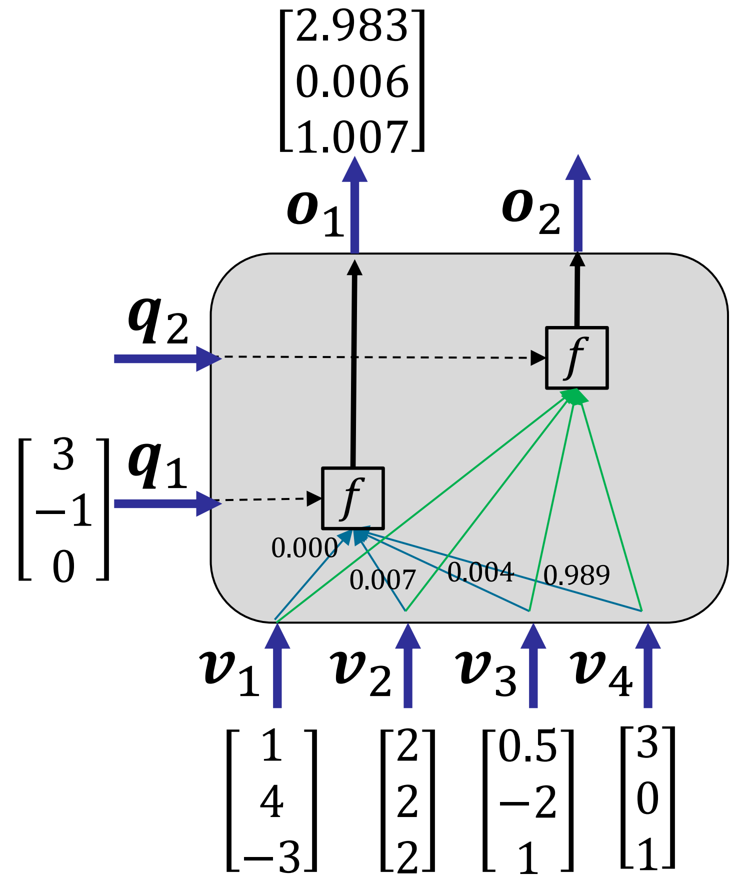
$$\boldsymbol{o}_i = \sum_{j=1}^{|\boldsymbol{V}|} \alpha_{i,j} \boldsymbol{v}_j$$

Example – recap

$$\tilde{\boldsymbol{\alpha}}_1 = \begin{bmatrix} \boldsymbol{q}_1\boldsymbol{v}_1^{\mathrm{T}} = -1 \\ \boldsymbol{q}_1\boldsymbol{v}_2^{\mathrm{T}} = 4 \\ \boldsymbol{q}_1\boldsymbol{v}_3^{\mathrm{T}} = 3.5 \\ \boldsymbol{q}_1\boldsymbol{v}_4^{\mathrm{T}} = 9 \end{bmatrix} \rightarrow \quad \boldsymbol{\alpha}_1 = \begin{bmatrix} 0.000 \\ 0.007 \\ 0.004 \\ 0.989 \end{bmatrix}$$

$$\boldsymbol{o}_1 = 0.000 \begin{bmatrix} 1 \\ 4 \\ -3 \end{bmatrix} + 0.007 \begin{bmatrix} 2 \\ 2 \\ 2 \end{bmatrix} + 0.004 \begin{bmatrix} 0.5 \\ -2 \\ 1 \end{bmatrix} + 0.989 \begin{bmatrix} 3 \\ 0 \\ 1 \end{bmatrix}$$

$$\boldsymbol{o}_1 = \begin{bmatrix} 2.983 \\ 0.006 \\ 1.007 \end{bmatrix}$$

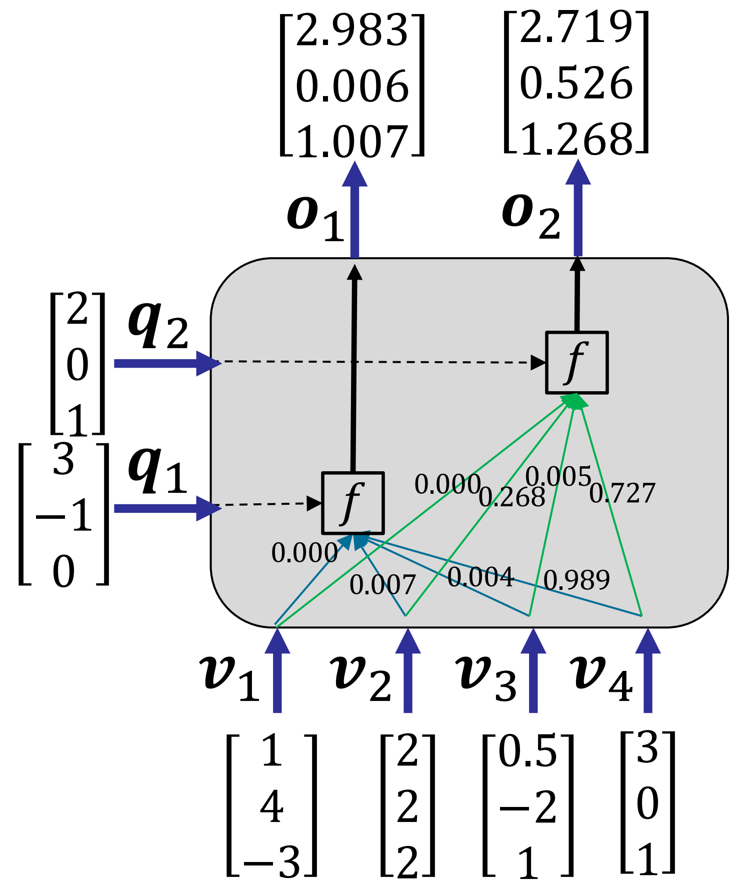


Example – recap

$$\tilde{\boldsymbol{\alpha}}_2 = \begin{bmatrix} \boldsymbol{q}_2\boldsymbol{v}_1^{\mathrm{T}} = -1 \\ \boldsymbol{q}_2\boldsymbol{v}_2^{\mathrm{T}} = 6 \\ \boldsymbol{q}_2\boldsymbol{v}_3^{\mathrm{T}} = 2 \\ \boldsymbol{q}_2\boldsymbol{v}_4^{\mathrm{T}} = 7 \end{bmatrix} \rightarrow \boldsymbol{\alpha}_2 = \begin{bmatrix} 0.000 \\ 0.268 \\ 0.005 \\ 0.727 \end{bmatrix}$$

$$\boldsymbol{o}_2 = 0.000 \begin{bmatrix} 1 \\ 4 \\ -3 \end{bmatrix} + 0.268 \begin{bmatrix} 2 \\ 2 \\ 2 \end{bmatrix} + 0.005 \begin{bmatrix} 0.5 \\ -2 \\ 1 \end{bmatrix} + 0.727 \begin{bmatrix} 3 \\ 0 \\ 1 \end{bmatrix}$$

$$\boldsymbol{o}_2 = \begin{bmatrix} 2.719 \\ 0.526 \\ 1.268 \end{bmatrix}$$

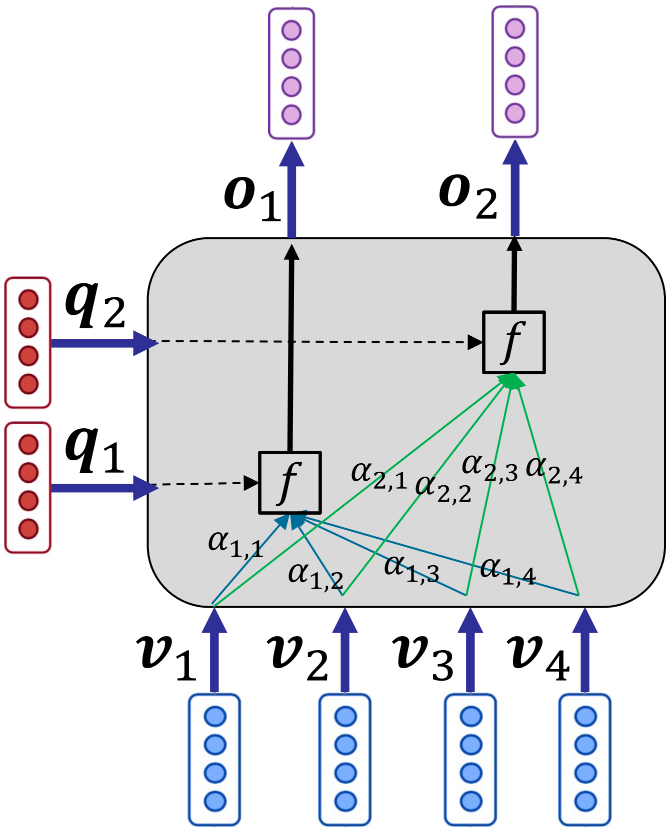


Attention table

	$\boldsymbol{v}_1$	$\boldsymbol{v}_2$	$\boldsymbol{v}_3$	$\boldsymbol{v}_4$
$\boldsymbol{q}_1$	$\alpha_{1,1}$	$\alpha_{1,2}$	$\alpha_{1,3}$	$\alpha_{1,4}$
$\boldsymbol{q}_2$	$\alpha_{2,1}$	$\alpha_{2,2}$	$\alpha_{2,3}$	$\alpha_{2,4}$

In the example:

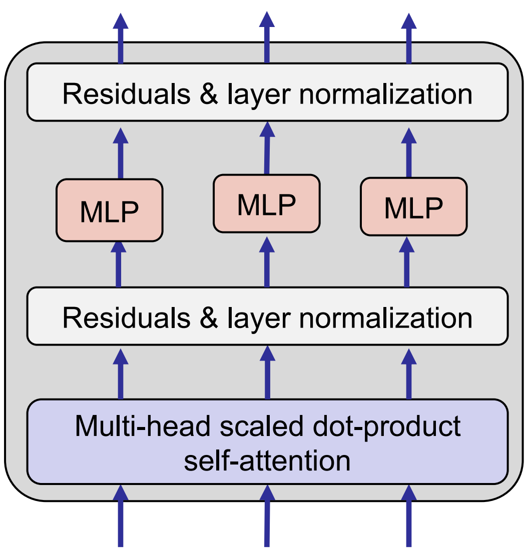
	$\boldsymbol{v}_1$	$\boldsymbol{v}_2$	$\boldsymbol{v}_3$	$\boldsymbol{v}_4$
$\boldsymbol{q}_1$	0.000	0.007	0.004	0.989
$\boldsymbol{q}_2$	0.000	0.268	0.005	0.727



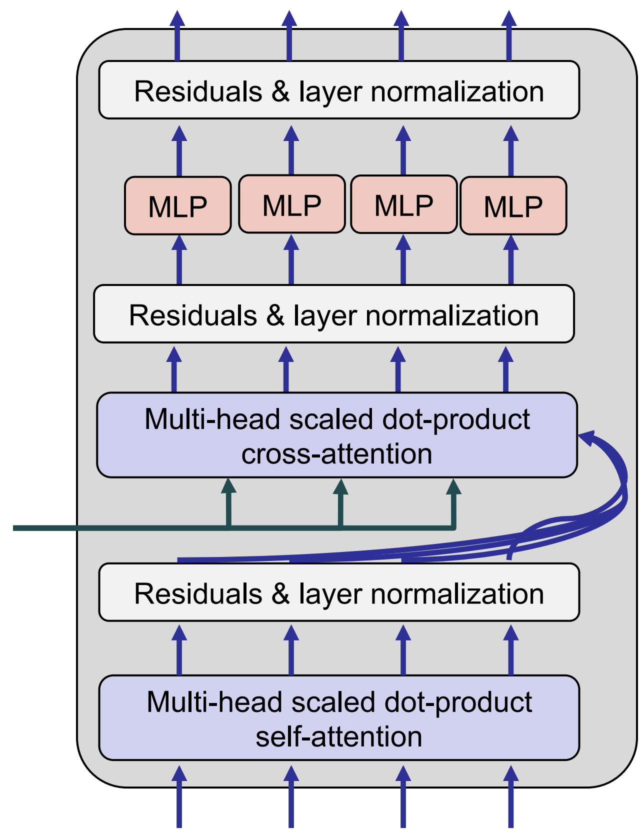
Transformers

- Attention network with DL best practices!
  - Originally introduced in the context of machine translation and is now widely adopted for sequence encoding and decoding

Transformer encoder



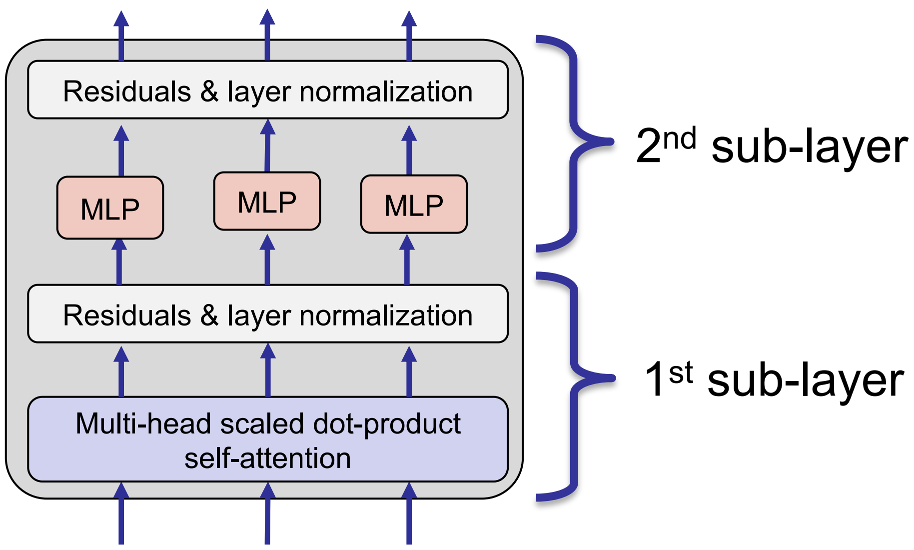
Transformer decoder



Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention is all you need." In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 2017.

Transformer encoder

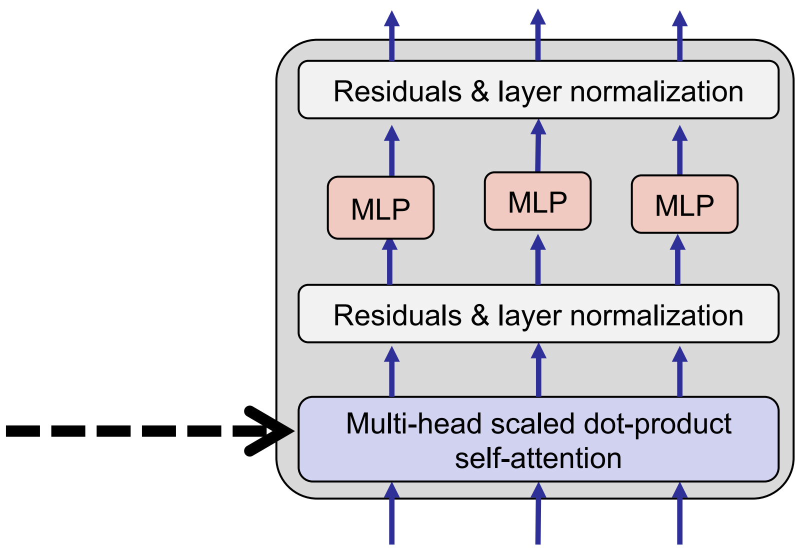
- Transformer encoder consists of two sub-layers:
  - 1st : Multi-head scaled dot-product self-attention
  - 2nd : Position-wise multi-layer perceptron (feed forward)
- Each sub-layer is followed by residual networks and layer normalization
  - drop-outs are applied after each computation



Transformer encoder

Let's start from <u>multi-head</u> <u>scaled dot-product</u> <u>self-attention</u>:

1. Scaled dot-product attention
2. Multi-head attention
3. self-attention



Basic dot-product attention – recap

- First, non-normalized attention scores:

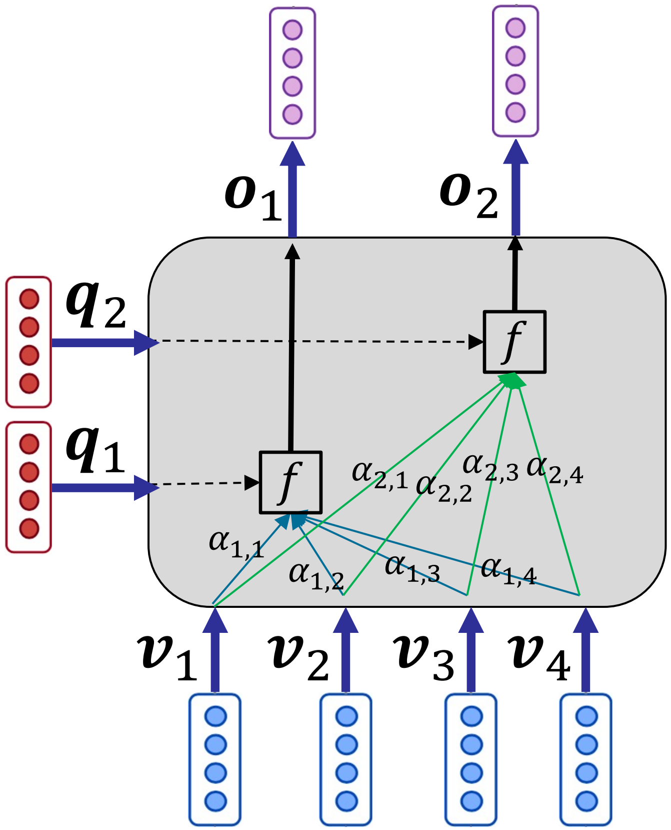
$$\tilde{\alpha}_{i,j} = \boldsymbol{q}_i \boldsymbol{v}_i^{\mathrm{T}}$$

  - In this variant $d_q = d_v$
  - There is no parameter to learn!

- Then, softmax over values:

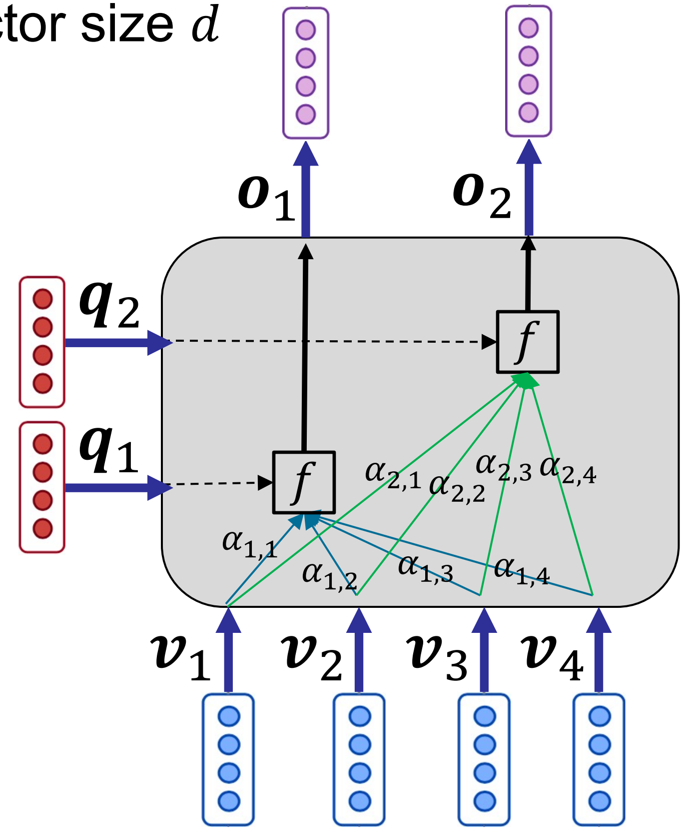
$$\boldsymbol{\alpha}_i = \mathrm{softmax}(\tilde{\boldsymbol{\alpha}}_i)$$

- Output: $\boldsymbol{o}_i = \sum_{j=1}^{|V|} \alpha_{i,j} \boldsymbol{v}_j$



Scaled dot-product attention

- Problem with basic dot-product attention:
  - As d gets large, the variance of $\tilde{\alpha}_{i,j}$ increases …
  - … this makes softmax very peaked for some values of $\tilde{\boldsymbol{\alpha}}_i$ …
  - … and hence its gradient gets smaller
- One approach: normalize/scale $\tilde{\alpha}_{i,j}$ by vector size d



Scaled dot-product attention

- Non-normalized attention scores:

$$\tilde{\alpha}_{i,j} = \frac{\boldsymbol{q}_i \boldsymbol{v}_j^{\mathrm{T}}}{\sqrt{d}}$$

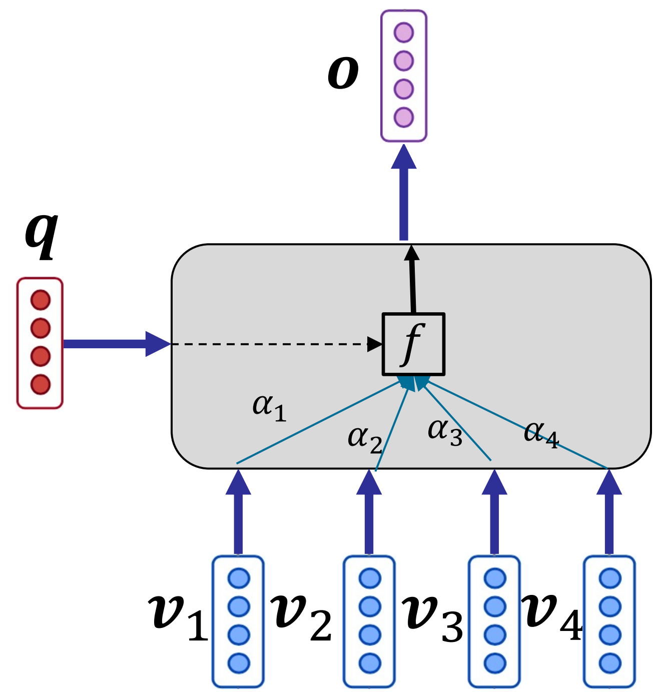
- Softmax over values: $\boldsymbol{\alpha}_i = \mathrm{softmax}(\tilde{\boldsymbol{\alpha}}_i)$
- Output: $\boldsymbol{o}_i = \sum_{j=1}^{|\boldsymbol{V}|} \alpha_{i,j} \boldsymbol{v}_j$

From single-head to multi-head attention

- softmax is applied to non-normalized attention vectors
  - Recall: softmax makes the maximum value much higher than the other

$$\boldsymbol{z} = [1 \quad 2 \quad 5 \quad 6] \rightarrow \text{softmax}(\boldsymbol{z}) = [0.004 \quad 0.013 \quad 0.264 \quad 0.717]$$

- Common in language, a word may be related to several other words in a sequence, each through a specific concept
  - Like the relations of a verb to its subject and to its object
- However, in a single-head attention network, all concepts are aggregated in one attention set
- Due to softmax, value vectors must compete for the attention of query vector → softmax bottleneck



Multi-head attention

- Multi-head attention approaches this issue by calculating multiple sets of attentions between a query and values

Multi-head attention:

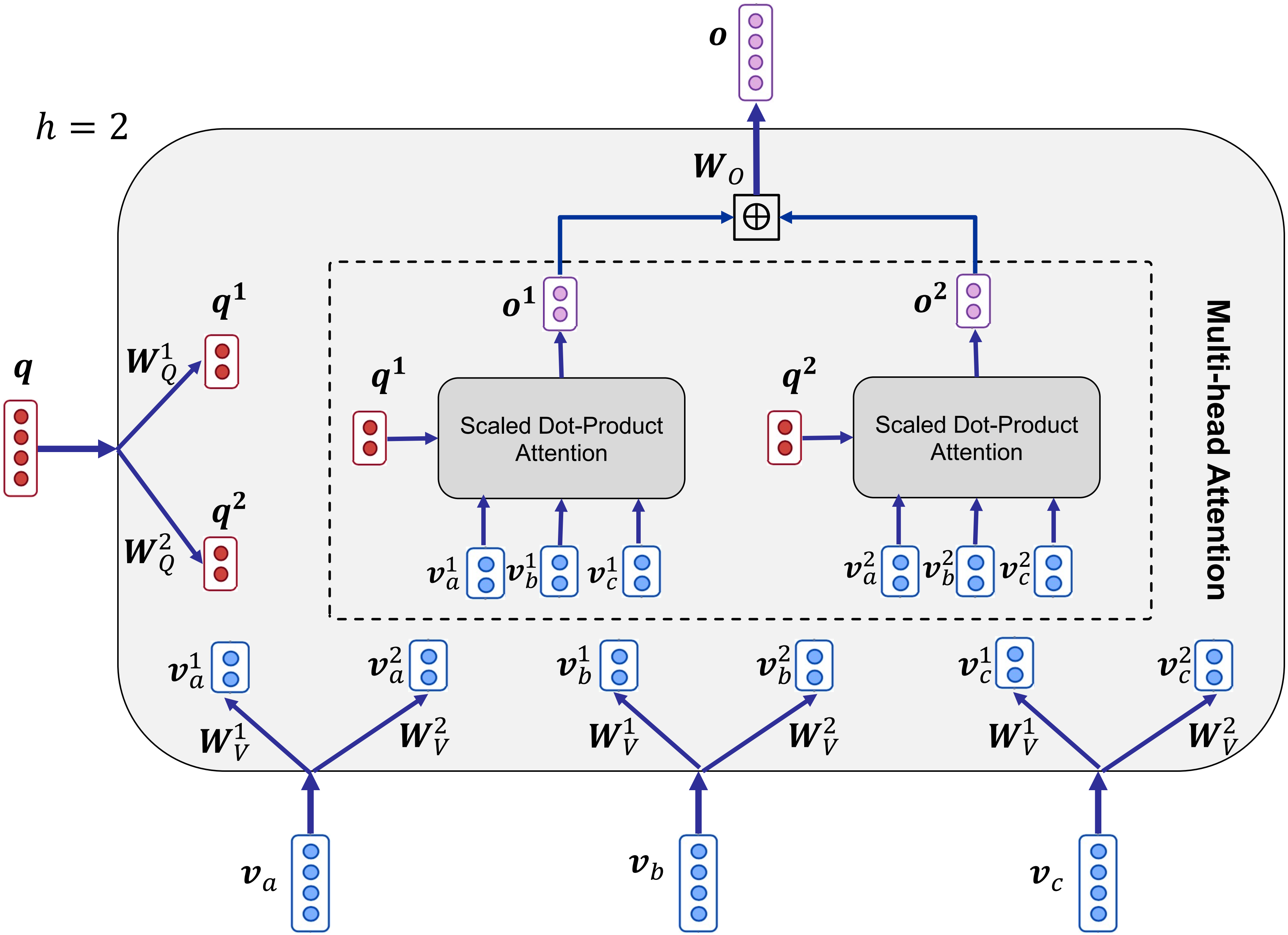
1. Transfer each query/value vector to h subspaces (heads)

2. In each subspace, apply a single-head attention network using the queries and values transferred to the subspace to achieve the output vectors of that subspace

3. Concatenate the output vectors of all subspaces in respect to a query to achieve the final output of the query

- In multi-head attention, each head (and each subspace) can specialize on capturing a specific kind of relations

Multi-head attention



Multi-head attention – formulation

- Transfer every query $\boldsymbol{q}_i$ to h vectors, each with size ${}^{d}/_{h}$:

size: ${}^{d}/_{h}$

$$\boldsymbol{q}_i^1 = \boldsymbol{q}_i \boldsymbol{W}_Q^1 \quad \ldots \quad \boldsymbol{q}_i^h = \boldsymbol{q}_i \boldsymbol{W}_Q^h$$

Matrix size: $d \times {}^{d}/_{h}$

- Transfer every value $\boldsymbol{v}_j$ to h vectors, each with size ${}^{d}/_{h}$:

size: ${}^{d}/_{h}$

$$\boldsymbol{v}_j^1 = \boldsymbol{v}_j \boldsymbol{W}_V^1 \quad \ldots \quad \boldsymbol{v}_j^h = \boldsymbol{v}_j \boldsymbol{W}_V^h$$

Matrix size: $d \times {}^{d}/_{h}$

- Calculate outputs of subspaces corresponding to $\boldsymbol{q}_i$:

size: ${}^{d}/_{h}$

$$\boldsymbol{o}_i^1 = \mathrm{ATT}\left(\boldsymbol{q}_i^1, \boldsymbol{V}^1\right) \quad \ldots \quad \boldsymbol{o}_i^h = \mathrm{ATT}\left(\boldsymbol{q}_i^h, \boldsymbol{V}^h\right)$$

- Concatenate outputs of subspaces for $\boldsymbol{q}_i$ as its final output:

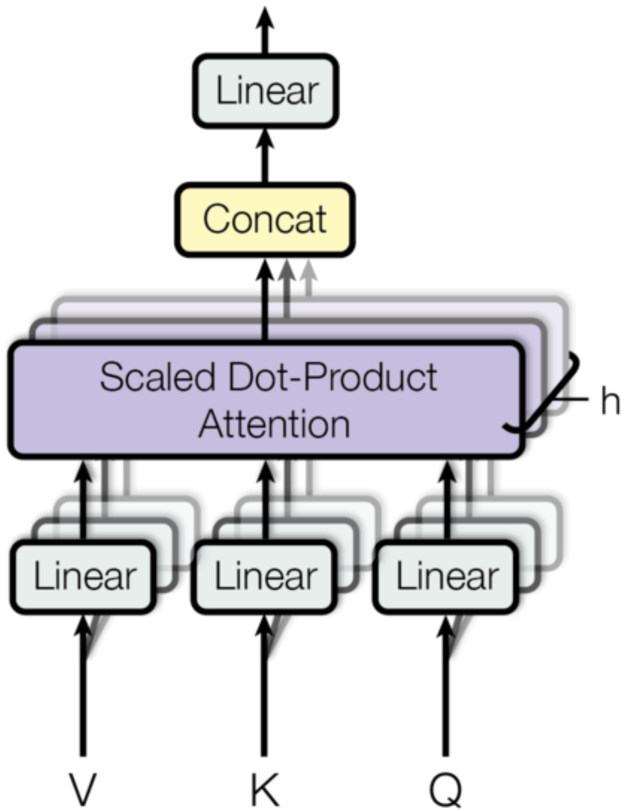
size: d

$$\boldsymbol{o}_i = \boldsymbol{W}_O \left[\boldsymbol{o}_i^1; \ldots; \boldsymbol{o}_i^h\right]$$

Size: $d \times d$
This matrix linearly combines the dimensions of the concatenated vectors

Parameters are shown in red

Multi-head attention – graphic in original paper

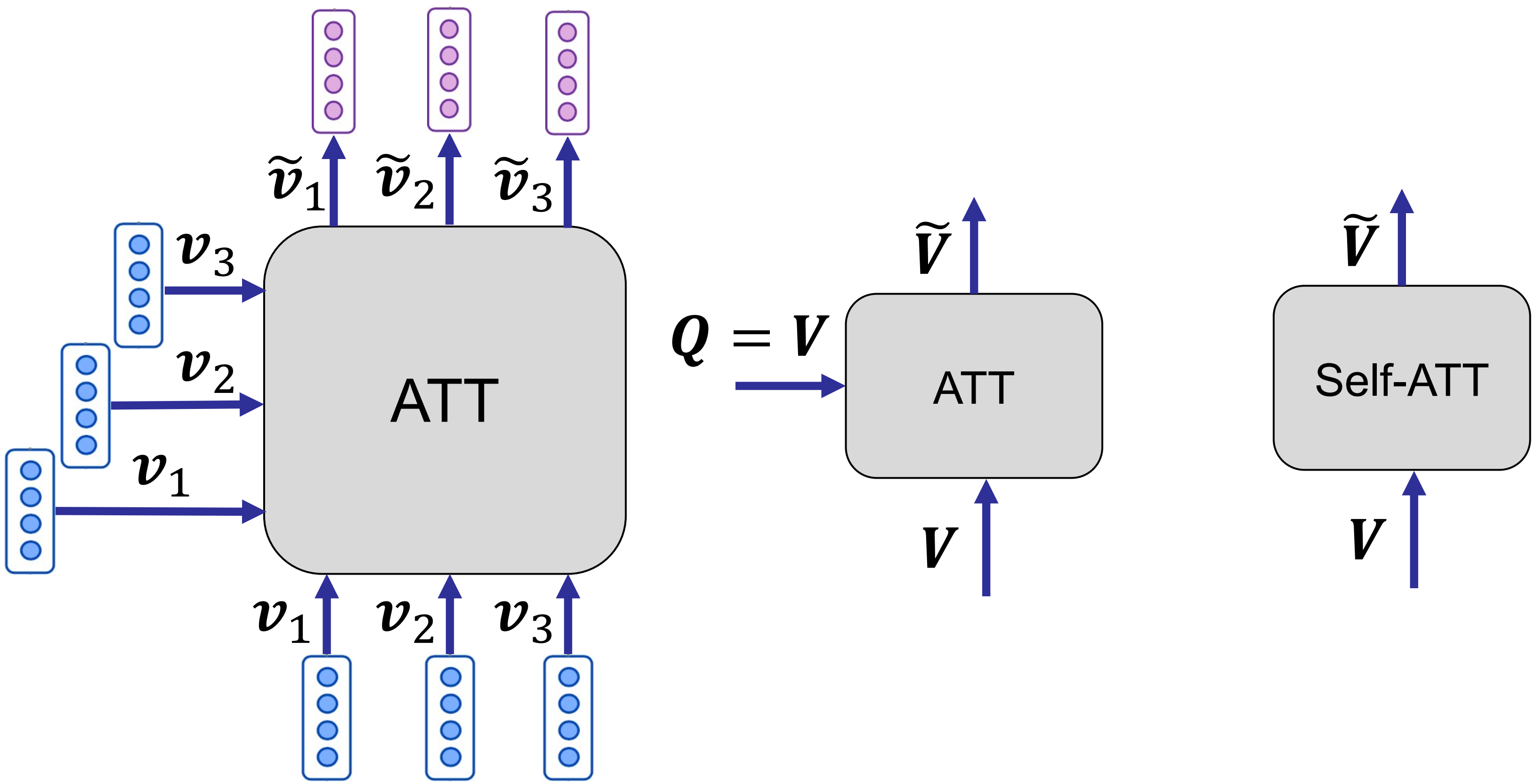


- Default number of heads in Transformers: $h = 8$
- Recall: Attentions (and Transformers) in fact have three inputs (not two), namely queries, keys, and values.
  - Keys are used to calculate attentions
  - Values are used to produce outputs

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention is all you need." In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 2017.

Self-attention – recap

- Values are the same as queries
- Each encoded vector is the contextual embedding of the corresponding input vector
  - $\tilde{\boldsymbol{v}}_i$ is the contextual embedding of $\boldsymbol{v}_i$

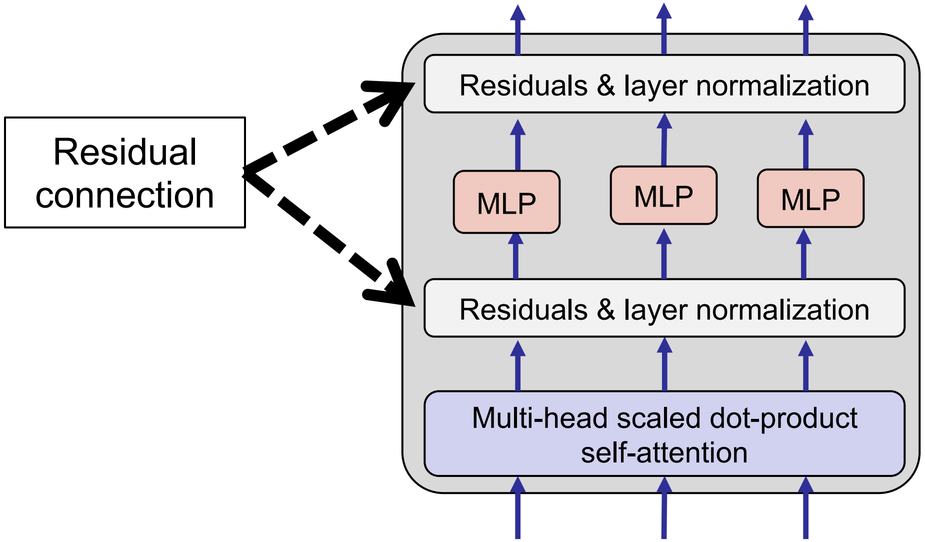


Residuals

- Residual (short-cut) connection:

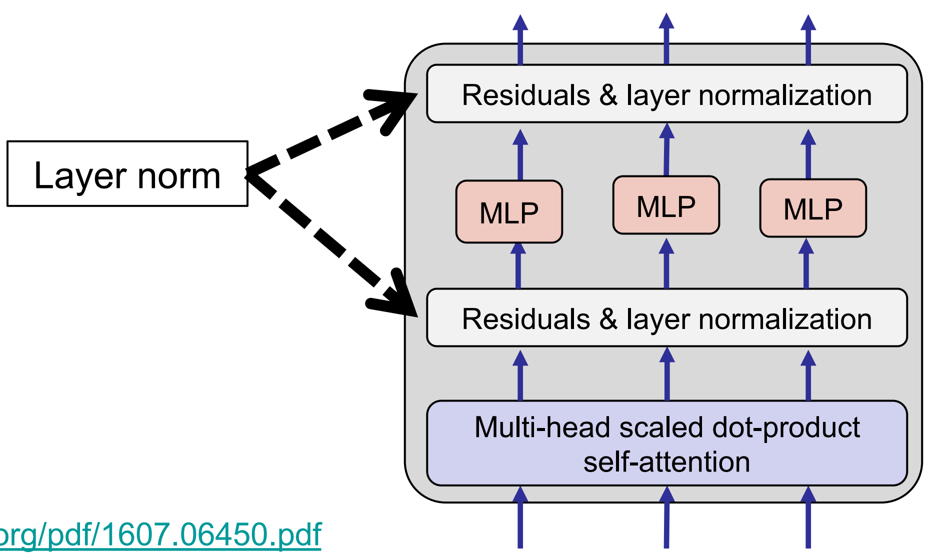
$$\text{output} = f(x) + x$$

- Learn in detail:
  - He, Kaiming; Zhang, Xiangyu; Ren, Shaoqing; Sun, Jian (2016). "Deep Residual Learning for Image Recognition" . In proc. of CVPR
  - Srivastava, Rupesh Kumar; Greff, Klaus; Schmidhuber, Jürgen (2015). "Highway Networks". https://arxiv.org/pdf/1505.00387.pdf



Layer normalization

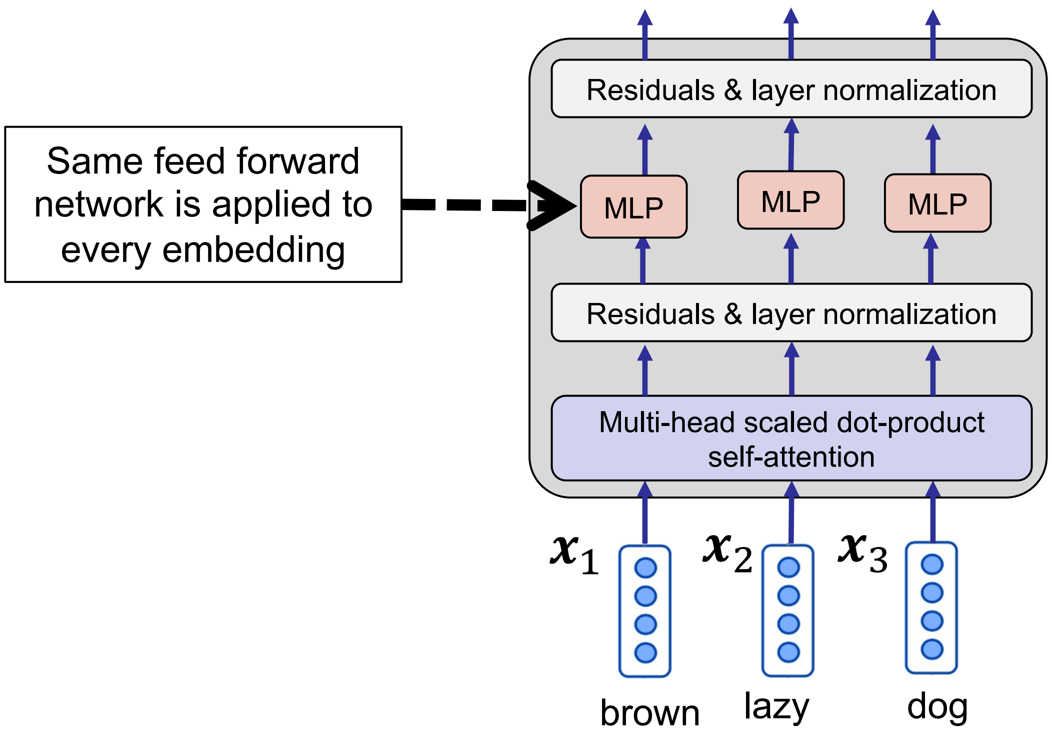
- Layer normalization changes the activations of each vector to have mean 0 and variance 1 …
  - … and learns two parameters per layer to shift the mean and variance



Paper: https://arxiv.org/pdf/1607.06450.pdf

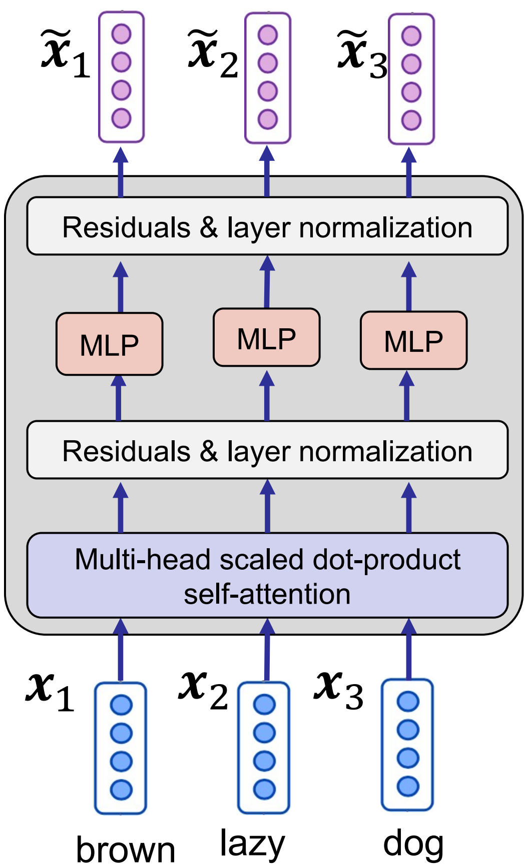
Multi-layer perceptron on embedding

- In Transformers, a two-layer multi-layer perceptron (feed forward) neural network with ReLU is applied to each output embedding
  - The feed forward layer provides capacity for non-linear transformations over each (contextualized) embedding



Transformer encoder – all together

- Transformer encoder receive input embeddings and outputs the corresponding contextualized embeddings
  - Processing all inputs happen ast the same time → non auto-regressive



Transformer encoder – summary

- A self-attention model using
  - multi-head scaled dot-product attention
  - followed by the same feed-forward layer applied to each embedding
  - all packed with residuals, layer norms, and dropouts

<u>Transformers as in attentions …</u>

- do not have locality (position) bias
  - A long-distance context has “equal opportunity”
- process all the input together with a single computation per each layer
  - Friendly with parallel computations in GPU

- Learn here more and study its PyTorch implementation: http://nlp.seas.harvard.edu/2018/04/03/attention.html

Position embeddings

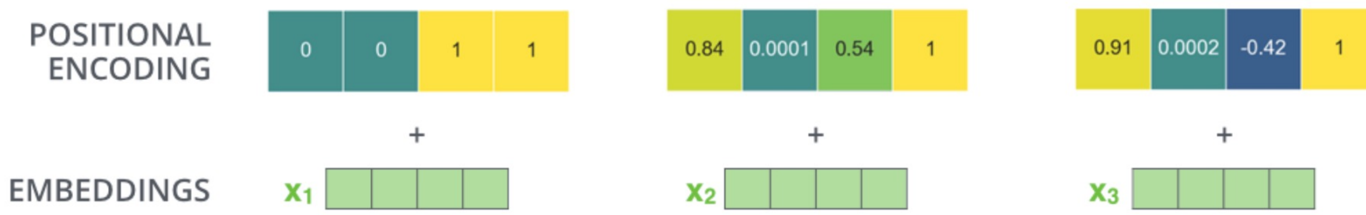
- Transformers are agnostic to the position of tokens
  - A context token in long-distance has the same effect as one in short-distance (no locality bias)
- However depending on the task, the positions of tokens in a sequence can be informative

Position embeddings – a common approach in Transformers:

- Create embeddings representing positions in a sequence, and add the corresponding position embedding to the token embedding
  - Position embedding is usually created using a sine/cosine function
    - It can also be learned end-to-end with the model parameters
  - Using position embeddings, the same word at different locations in a sequence will have different representations

Position embeddings – examples

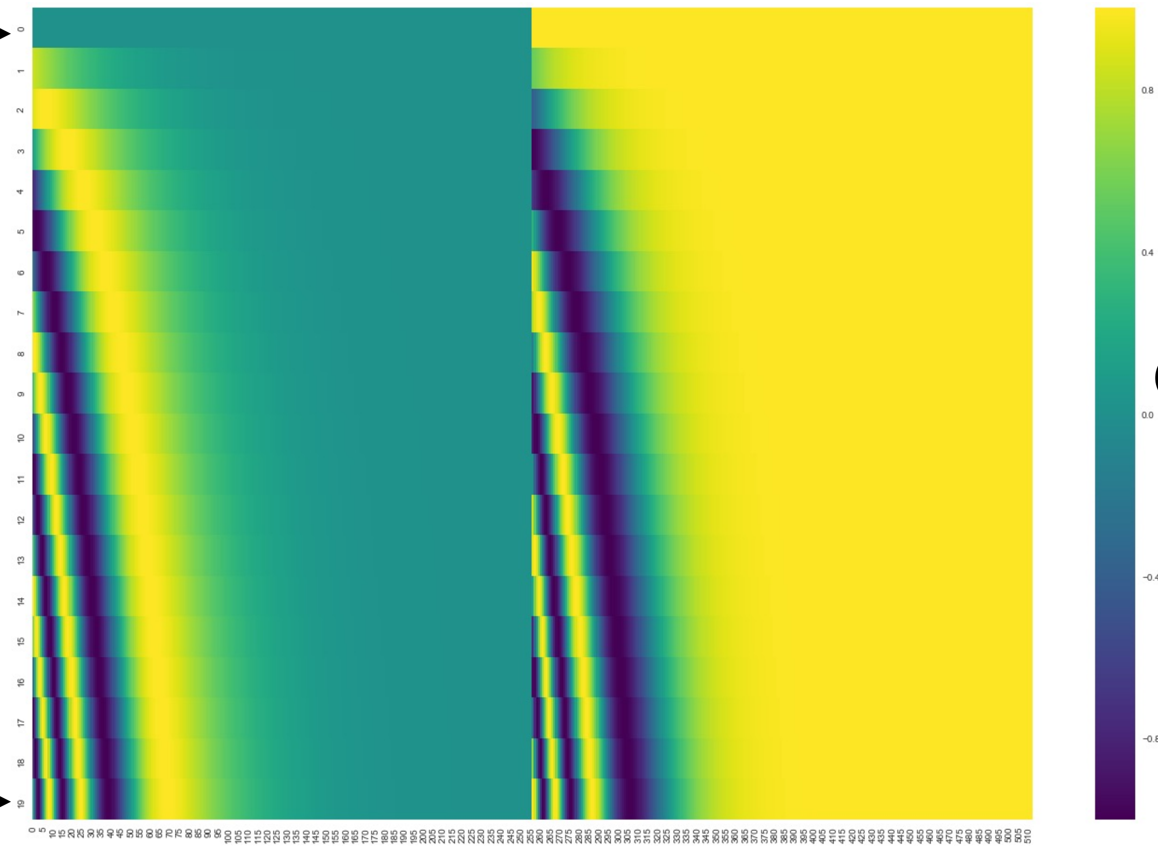
An example of embeddings with four dimensions:



Position embedding for location 0

Position embeddings

Position embedding for location 20

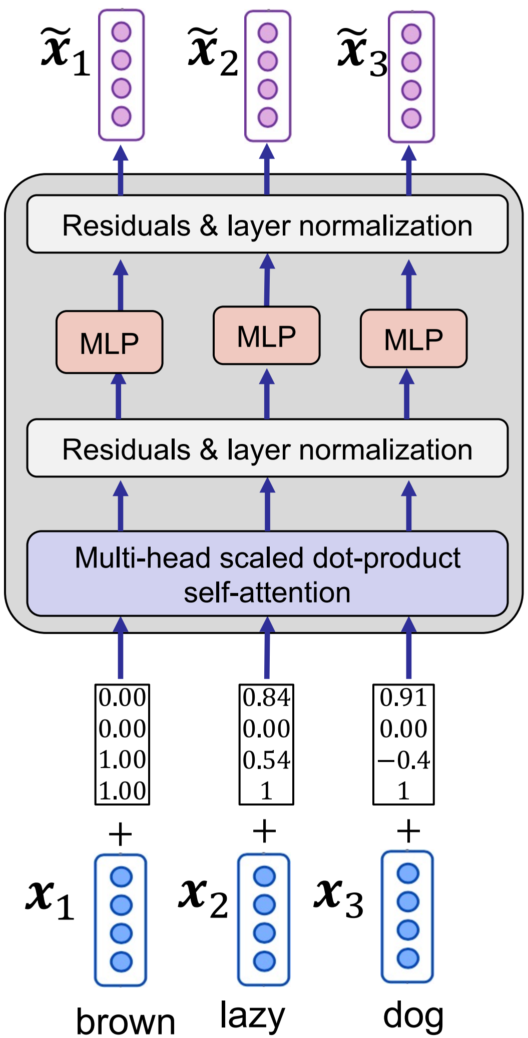


Values from -1 (dark) to +1 (light)

Dimensions (512)

Source: http://jalammar.github.io/illustrated-transformer/

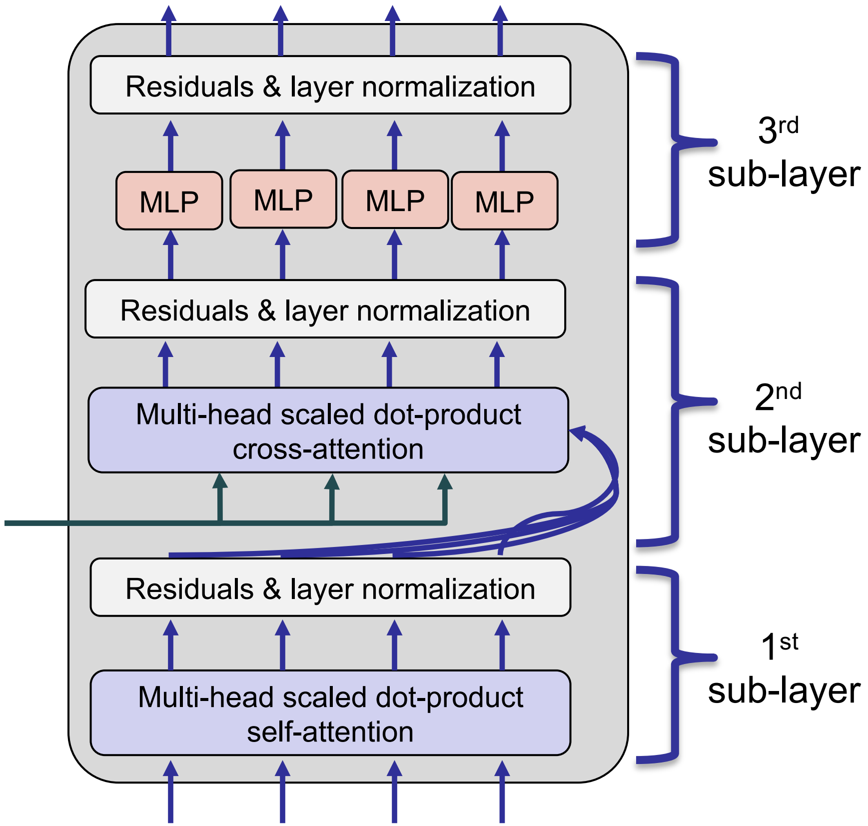
Transformers with position embedding



Transformer decoder

Transformer decoder consists of three sub-layers:

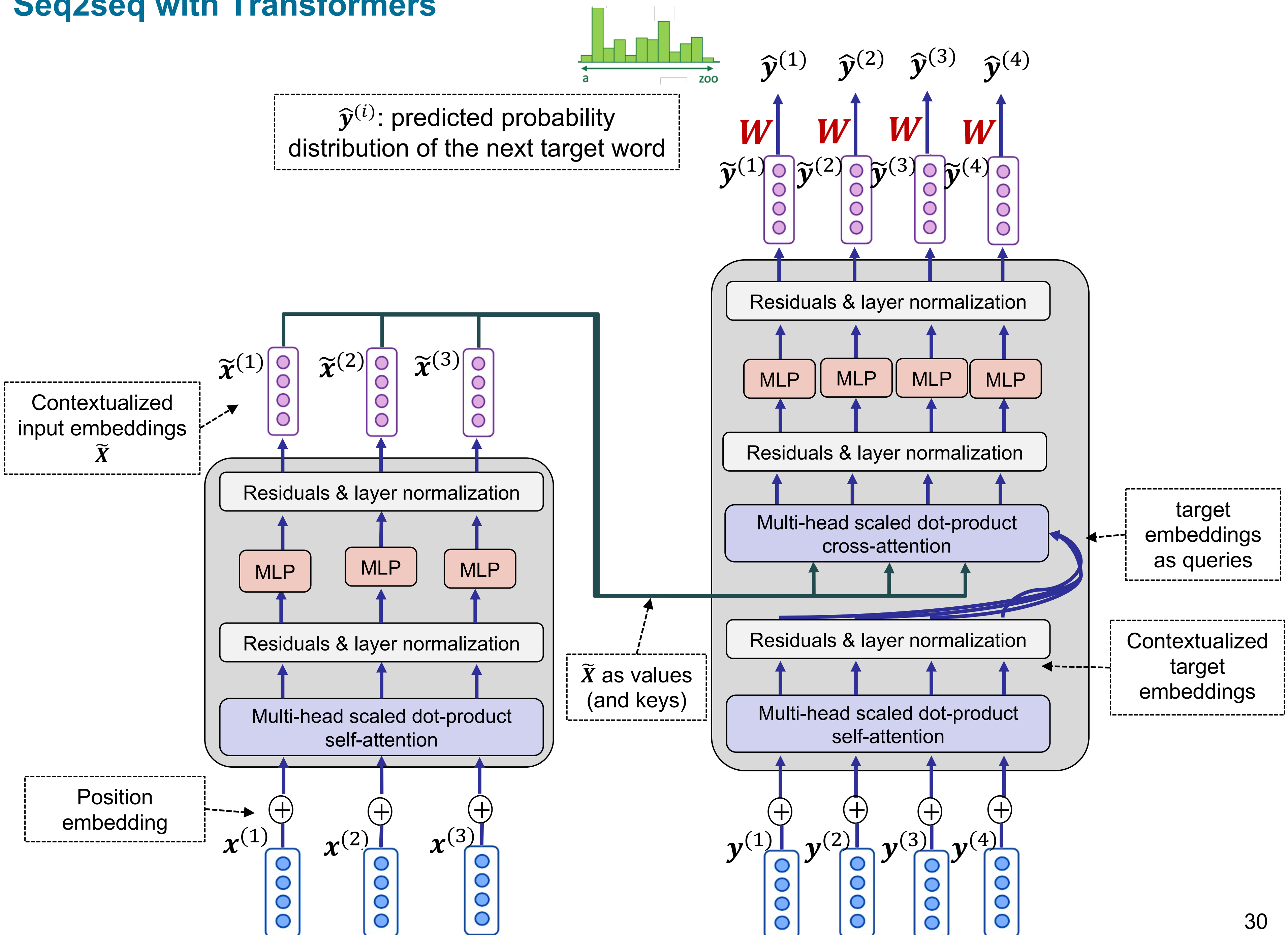
- 1st : Masked multi-head self-attention
- 2nd : Multi-head cross attention
  - Queries are the outputs of the previous sub-layer (contextualized embeddings)
  - Values come from outside (encoder)
  - Transformer decoder attends to the embeddings of an encoder (usually a Transformer encoder)
- 3rd : Position-wise multi-layer perceptron (feed forward)



Agenda

- Transformers
  - Transformer encoder
  - Transformer decoder
- **seq2seq with Transformers**

Seq2seq with Transformers



Seq2seq with Transformers – training

1. Data preparation: Source sequence X and target sequence Y are both started with $< \mathrm{bos} >$ and ended with $< \mathrm{eos} >$
2. Transformer encoder: outputs contextualized embeddings $\widetilde{\boldsymbol{X}}$
   - $\widetilde{\boldsymbol{X}}$ is in size $|X| \times d$, where d is the embedding vector dimensions
3. Decoder self-attention: create contextualized embeddings $\widetilde{\boldsymbol{Y}}$
   - $\widetilde{\boldsymbol{Y}}$ is in size $|Y| \times d$
4. Decoder cross-attention: $\widetilde{\boldsymbol{Y}}$ vectors attend to $\widetilde{\boldsymbol{X}}$ vectors ($\widetilde{\boldsymbol{Y}}$ queries $\widetilde{\boldsymbol{X}}$ values)
   - Resulting matrix is in size $|Y| \times d$
5. Prediction: Transformer decoder outputs are used to calculate $\widehat{\boldsymbol{Y}}$ – the probability distributions of the next tokens
   - $\widehat{\boldsymbol{Y}}$ is in size $|Y| \times |\mathbb{V}_d|$
   - E.g., vector $\widehat{\boldsymbol{y}}^{(2)}$ predicts the probability distribution of the token at position 3
6. Loss: is calculated using Negative Log Likelihood of the actual next words
   - E.g., based on the probability value of token $y^{(3)}$ in vector $\widehat{\boldsymbol{y}}^{(2)}$

Problem: in decoder self-attention, every token looks at all other tokens, namely the previous ones but also the next tokens!

- Every token has access to what it is supposed to predict!

Masking attentions

- In seq2seq with Transformers, we mask the attentions to every future token according to the self-attentions table of the decoder

Example

- Non-normalized self-attention table of decoder:

Vectors of output (target) sequence; attends to

	$\boldsymbol{y}^{(1)}$	$\boldsymbol{y}^{(2)}$	$\boldsymbol{y}^{(3)}$	$\boldsymbol{y}^{(4)}$
$\boldsymbol{y}^{(1)}$	5	3	1	-4
$\boldsymbol{y}^{(2)}$	1	4	-2	3
$\boldsymbol{y}^{(3)}$	0	2	2	-3
$\boldsymbol{y}^{(4)}$	3	-1	1	4

Non-normalized self-attention values

	$y^{(1)}$	$y^{(2)}$	$y^{(3)}$	$y^{(4)}$
$y^{(1)}$	5	3	1	-4
$y^{(2)}$	1	4	-2	3
$y^{(3)}$	0	-2	2	-3
$y^{(4)}$	3	-1	1	4

attentions masks

	$y^{(1)}$	$y^{(2)}$	$y^{(3)}$	$y^{(4)}$
$y^{(1)}$	1	0	0	0
$y^{(2)}$	1	1	0	0
$y^{(3)}$	1	1	1	0
$y^{(4)}$	1	1	1	1

Applying masks to attention scores

- adds $-\infty$ for every mask value 0
- adds 0 for every mask value 1

	$y^{(1)}$	$y^{(2)}$	$y^{(3)}$	$y^{(4)}$
$y^{(1)}$	5	$-\infty$	$-\infty$	$-\infty$
$y^{(2)}$	1	4	$-\infty$	$-\infty$
$y^{(3)}$	0	-2	2	$-\infty$
$y^{(4)}$	3	-1	1	4

softmax

Final sself-attention values

	$y^{(1)}$	$y^{(2)}$	$y^{(3)}$	$y^{(4)}$
$y^{(1)}$	1.00	0.00	0.00	0.00
$y^{(2)}$	0.04	0.96	0.00	0.00
$y^{(3)}$	0.11	0.01	0.86	0.00
$y^{(4)}$	0.25	0.01	0.34	0.70

☞ In Transformers, there are h times of such attention matrices. The same masking is applied to each of them.

Seq2seq with Transformers – training (completed!)

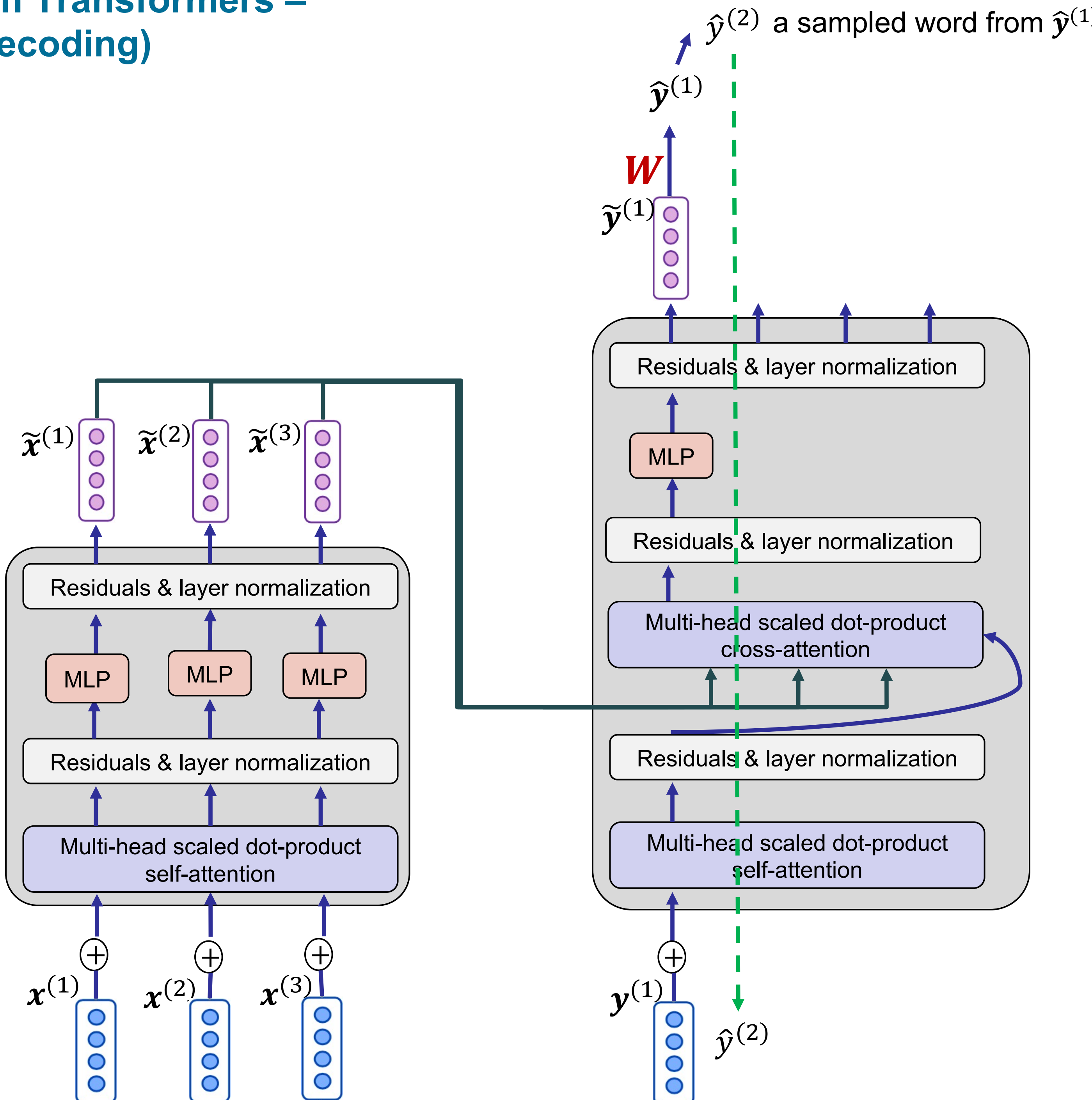
1. Data preparation: Source sequence X and target sequence Y are both started with $<\mathrm{bos}>$ and ended with $<\mathrm{eos}>$
2. Transformer encoder: outputs contextualized embeddings $\widetilde{\boldsymbol{X}}$
   - $\widetilde{\boldsymbol{X}}$ is in size $|X| \times d$, where d is the embedding vector dimensions
3. Decoder self-attention: create contextualized embeddings $\widetilde{\boldsymbol{Y}}$ **while masking future tokens**
   - $\widetilde{\boldsymbol{Y}}$ is in size $|Y| \times d$
4. Decoder cross-attention: $\widetilde{\boldsymbol{Y}}$ vectors attend to $\widetilde{\boldsymbol{X}}$ vectors ($\widetilde{\boldsymbol{Y}}$ queries $\widetilde{\boldsymbol{X}}$ values)
   - Resulting matrix is in size $|Y| \times d$
5. Prediction: Transformer decoder outputs are used to calculate $\widehat{\boldsymbol{Y}}$ – the probability distributions of the next tokens
   - $\widehat{\boldsymbol{Y}}$ is in size $|Y| \times |\mathbb{V}_d|$
   - E.g., vector $\widehat{\boldsymbol{y}}^{(2)}$ predicts the probability distribution of the token at position 3
6. Loss: is calculated using Negative Log Likelihood of the actual next words
   - E.g., based on the probability value of token $y^{(3)}$ in vector $\widehat{\boldsymbol{y}}^{(2)}$

Inference (decoding)

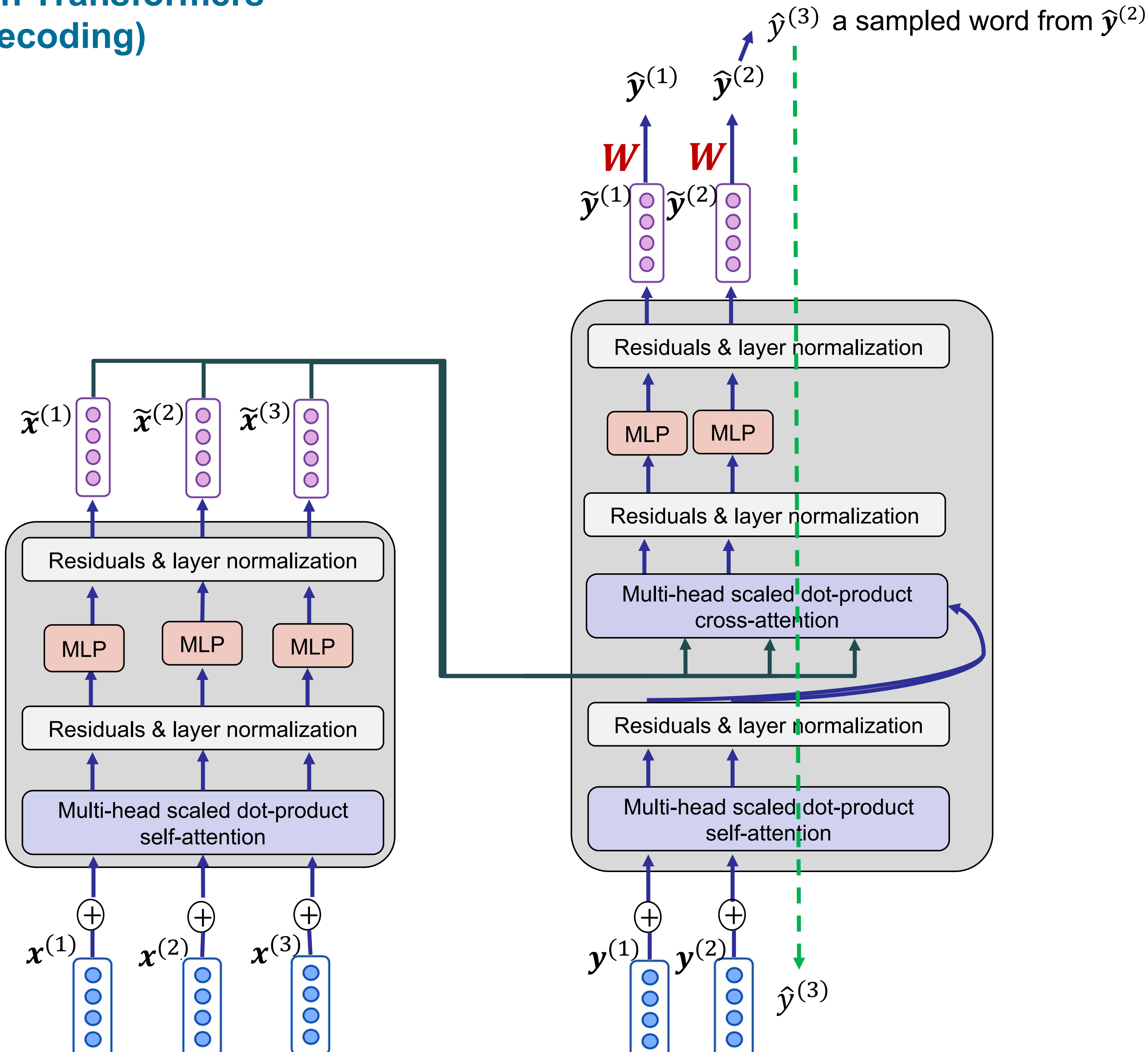
- In inference similar to training, the encoding of input sequence is done with a single computation (non-autoregressive)

- Decoding is however autoregressive (similar to seq2seq with RNNs):
  - Pass the 1st target token, generate the 2nd token
  - Pass the 1st and 2nd generated target tokens, generate the 3rd token
  - …

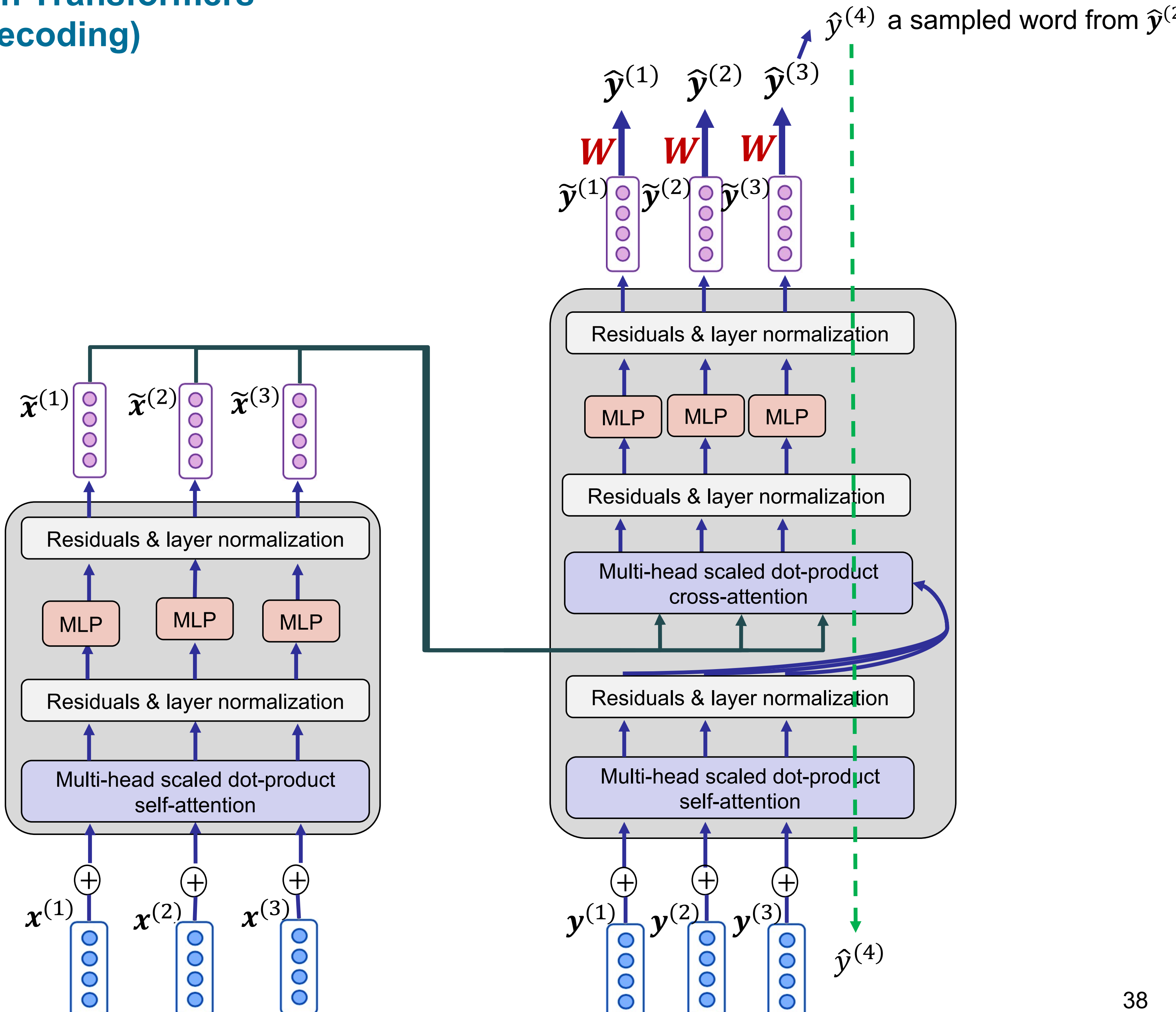
Seq2seq with Transformers – inference (decoding)



Seq2seq with Transformers – inference (decoding)



Seq2seq with Transformers – inference (decoding)



Seq2seq with Transformers – code

- Each Transformer encoder/decoder is a block. You can stack them several times and make the network deep!

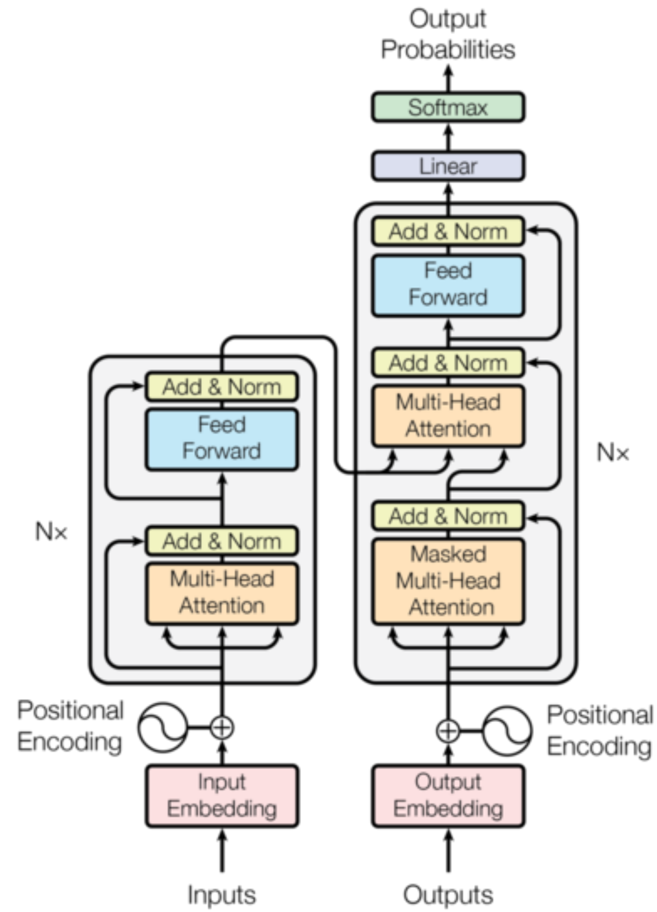
CLASS `torch.nn.TransformerEncoder(encoder_layer, num_layers, norm=None)` [SOURCE]

CLASS `torch.nn.TransformerEncoderLayer(d_model, nhead, dim_feedforward=2048, dropout=0.1, activation='relu')` [SOURCE]

CLASS `torch.nn.TransformerDecoder(decoder_layer, num_layers, norm=None)` [SOURCE]

CLASS `torch.nn.TransformerDecoderLayer(d_model, nhead, dim_feedforward=2048, dropout=0.1, activation='relu')` [SOURCE]

`forward(tgt, memory, tgt_mask=None, memory_mask=None, tgt_key_padding_mask=None, memory_key_padding_mask=None)` [SOURCE]



Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention is all you need." In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 2017.