EECS 545: Machine Learning

Lecture 14. Attention Mechanism and Transformer Networks

Honglak Lee

02/26/2025

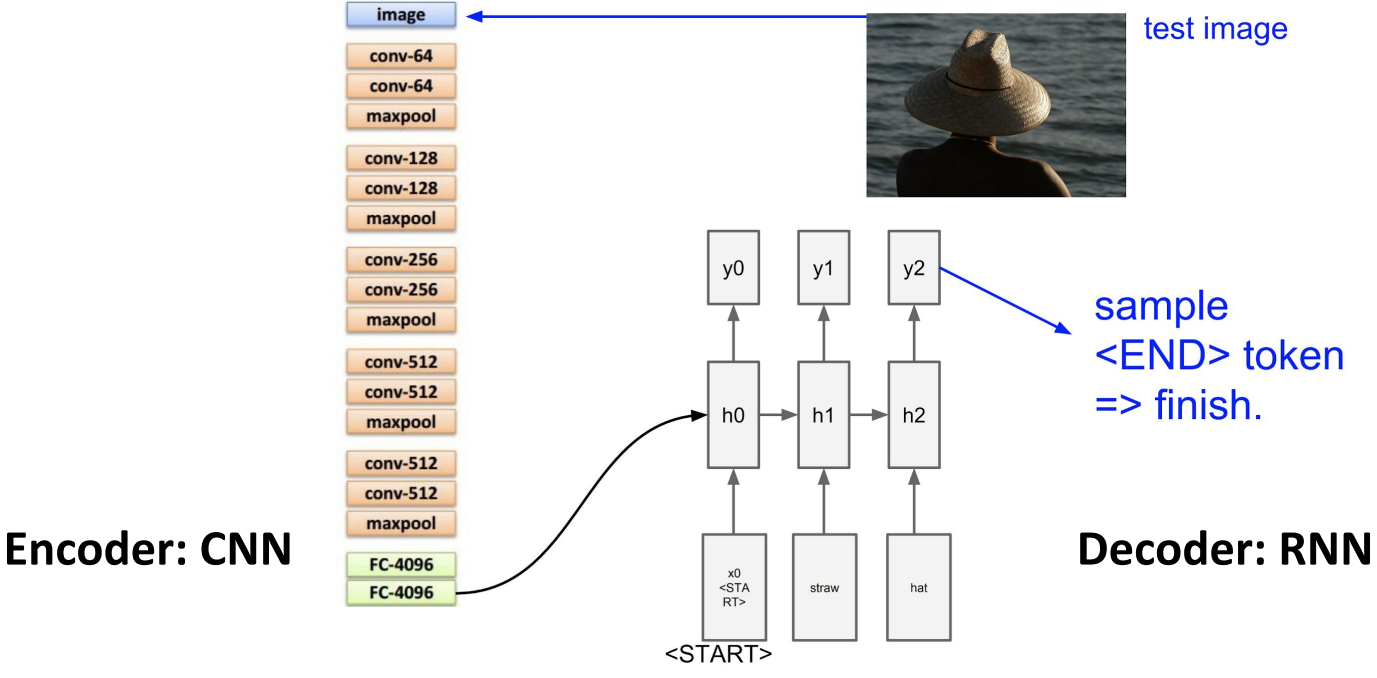
UNIVERSITY OF MICHIGAN

Outline

- Issues with RNN
- Attention Mechanism
  - Sequence-to-Sequence with attention
  - Attention Layer
  - Self- and cross- attention
- Transformer Architectures
- Some exemplar Transformer models

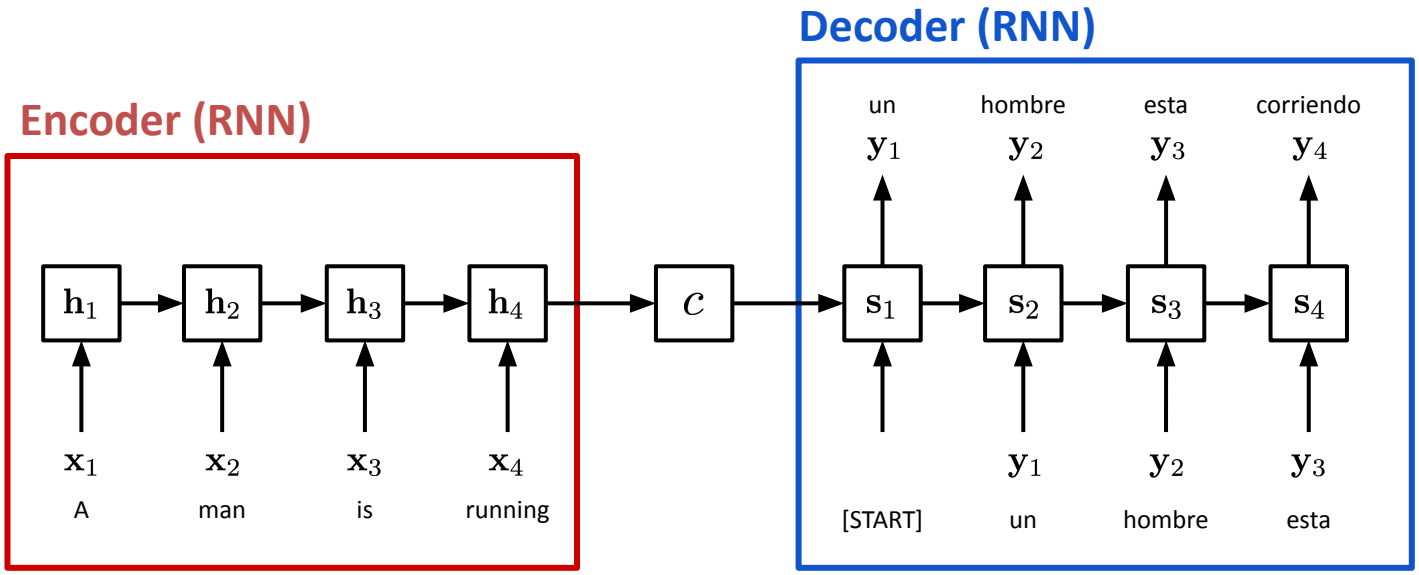
Recap: Image Captioning

- Input: an image
- Output: a sequence of words (generated by RNN)



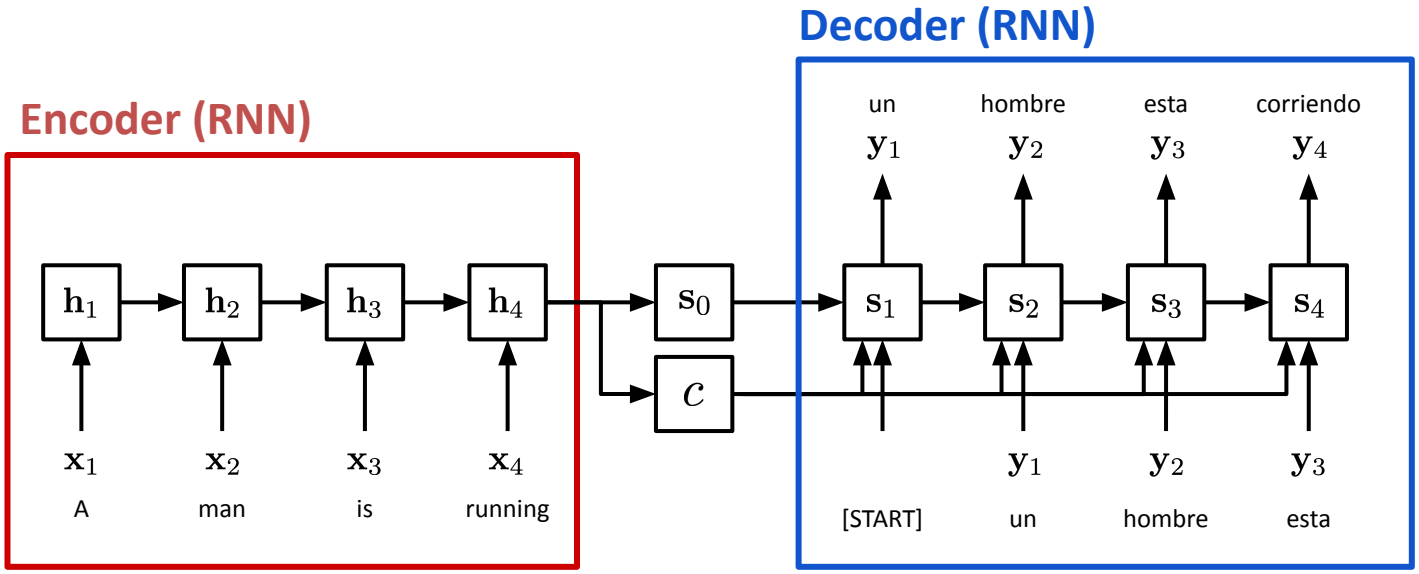
Sequence-to-Sequence with RNNs

- Example: Machine Translation (e.g. English to Spanish)
  - Input: a sequence of words $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_T$
  - Output: a sequence of words $\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_{T'}$



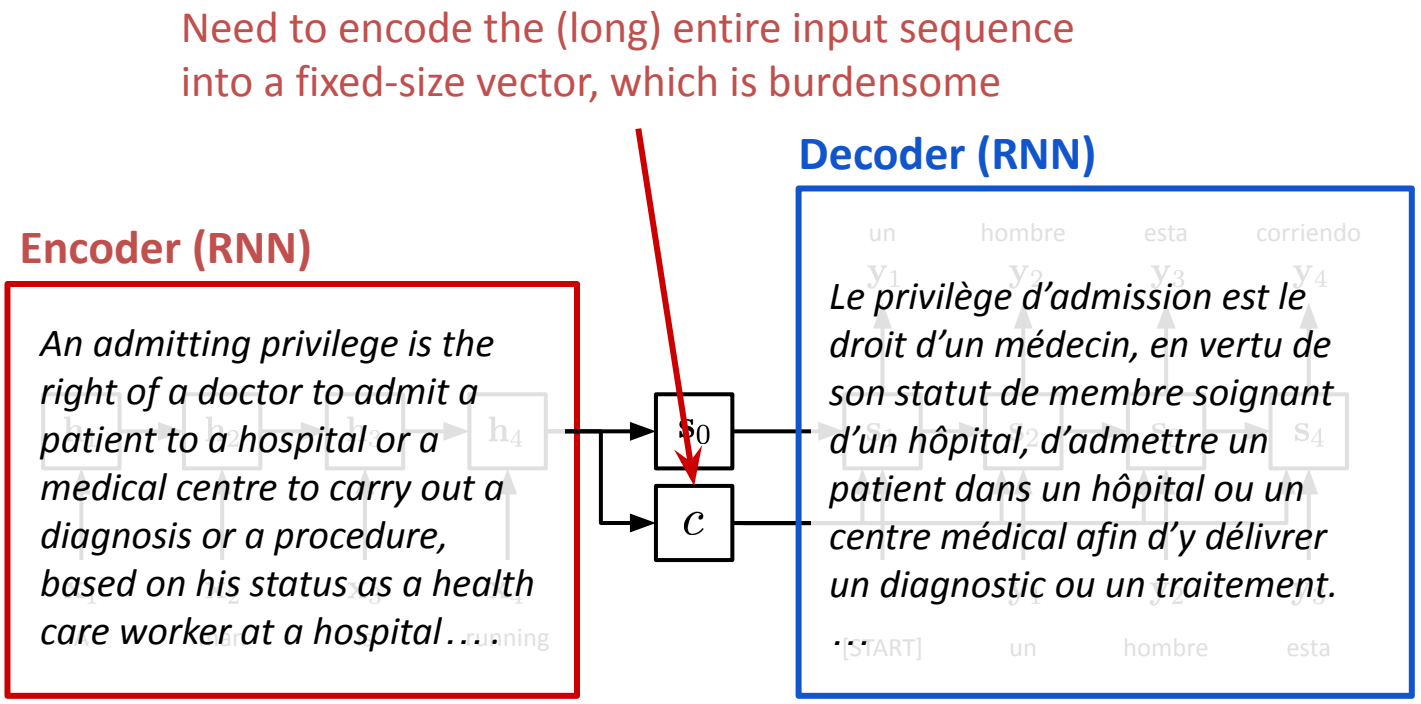
Sequence-to-Sequence with RNNs

- Example: Machine Translation (e.g. English to Spanish)
  - Input: a sequence of words $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_T$
  - Output: a sequence of words $\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_{T'}$



Sequence-to-Sequence with RNNs

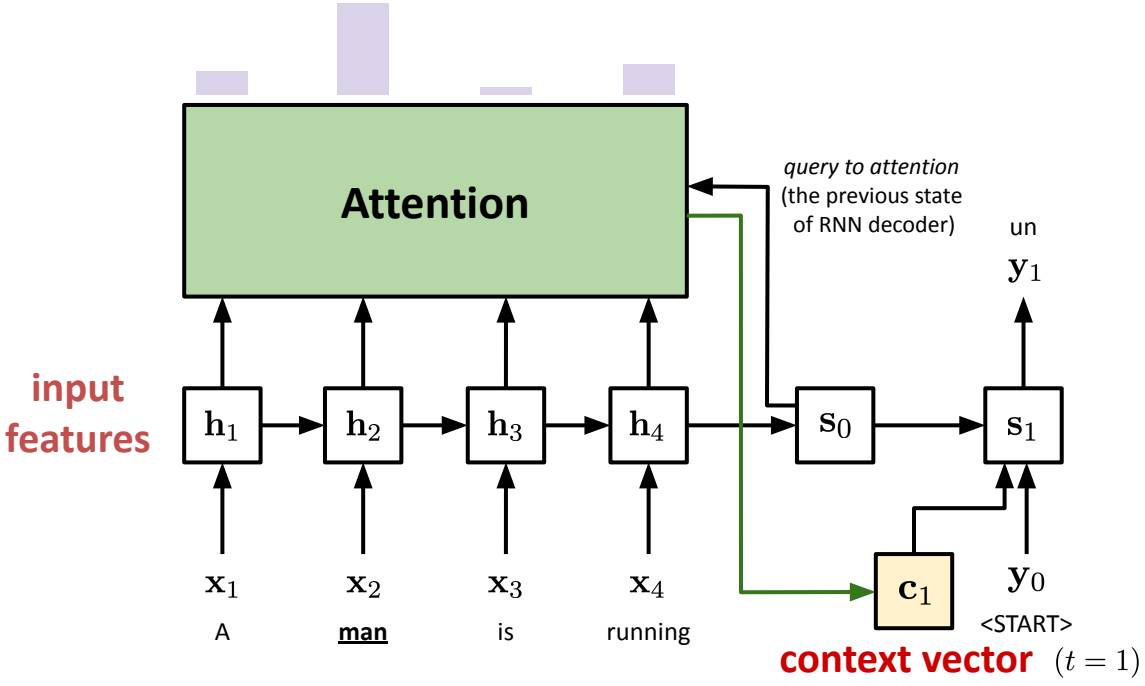
- Issues with RNN: the entire input needs to be squashed into a fixed-sized vector, encoder being the bottleneck
  - e.g. What if $|\mathbf{x}| = \mathbf{T} = 1000$?



(Bahdanau et al., 2015)

Attention Mechanism

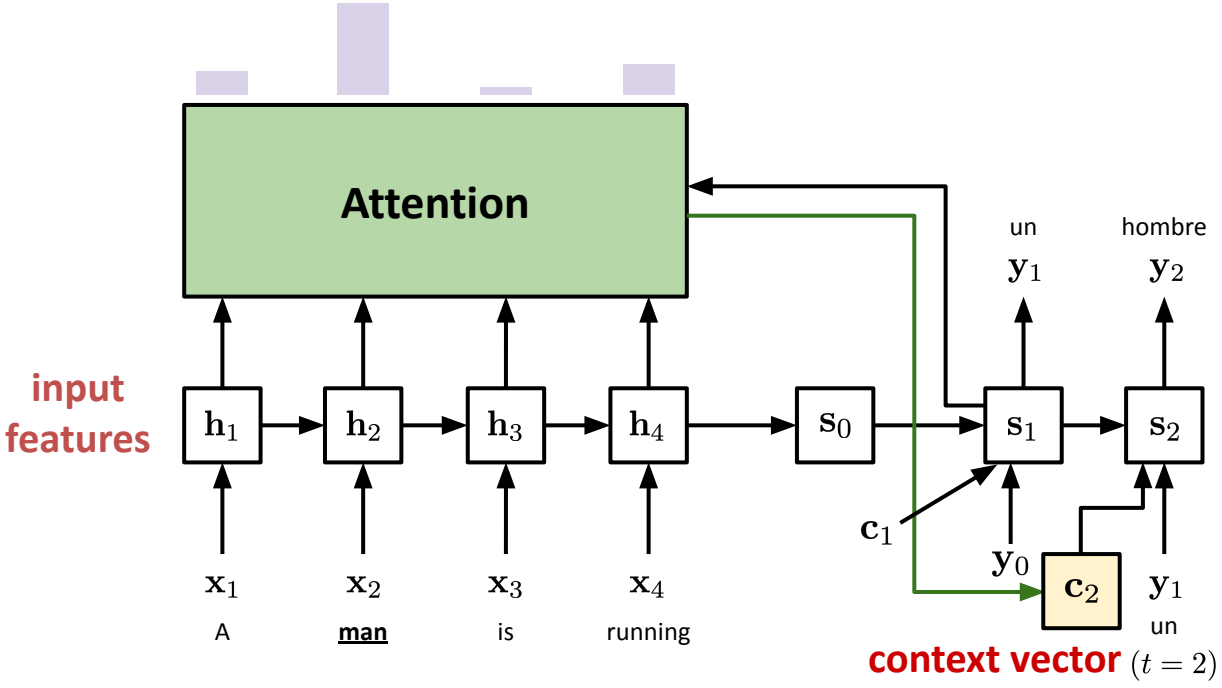
- Main idea: Use different context vector at *each* step of decoding, by **attending** to the *relevant* part of the input sequence



(Bahdanau et al., 2015)

Attention Mechanism

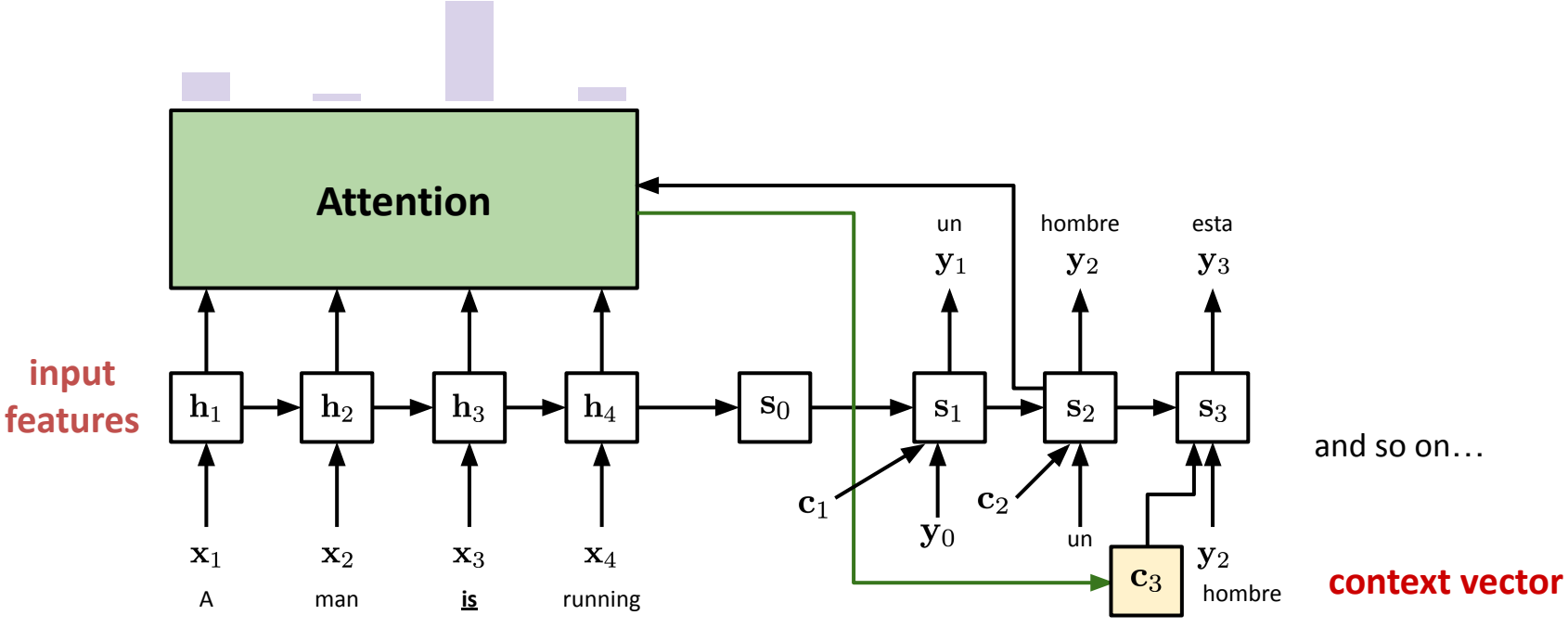
- Main idea: Use different context vector at *each* step of decoding, by **attending** to the *relevant* part of the input sequence



(Bahdanau et al., 2015)

Attention Mechanism

- Main idea: Use different context vector at *each* step of decoding, by **attending** to the *relevant* part of the input sequence

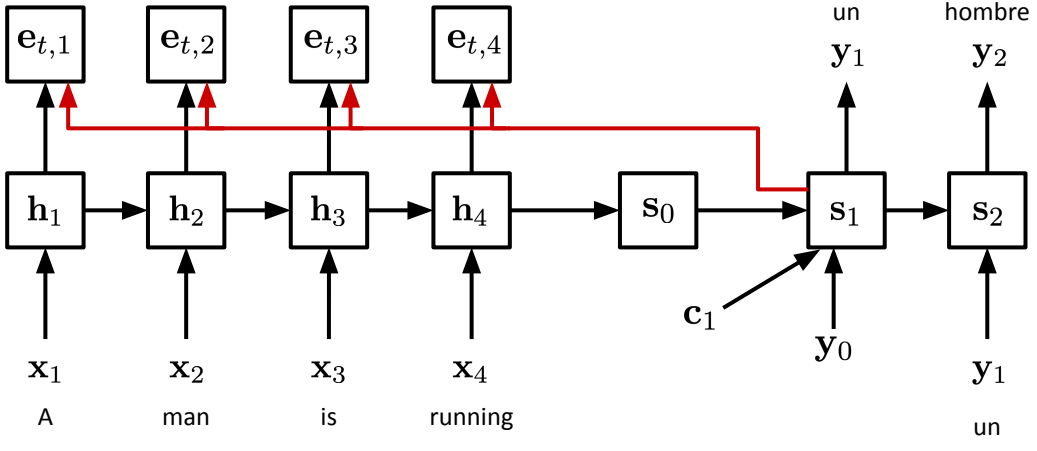


Attention Mechanism

- Compute alignment score at timestep ***t***: $\mathbf{e}_{t,j} = f_{\text{att}}(\mathbf{s}_{t-1}, \mathbf{h}_j)$
  - "similarity score" (e.g., dot product)

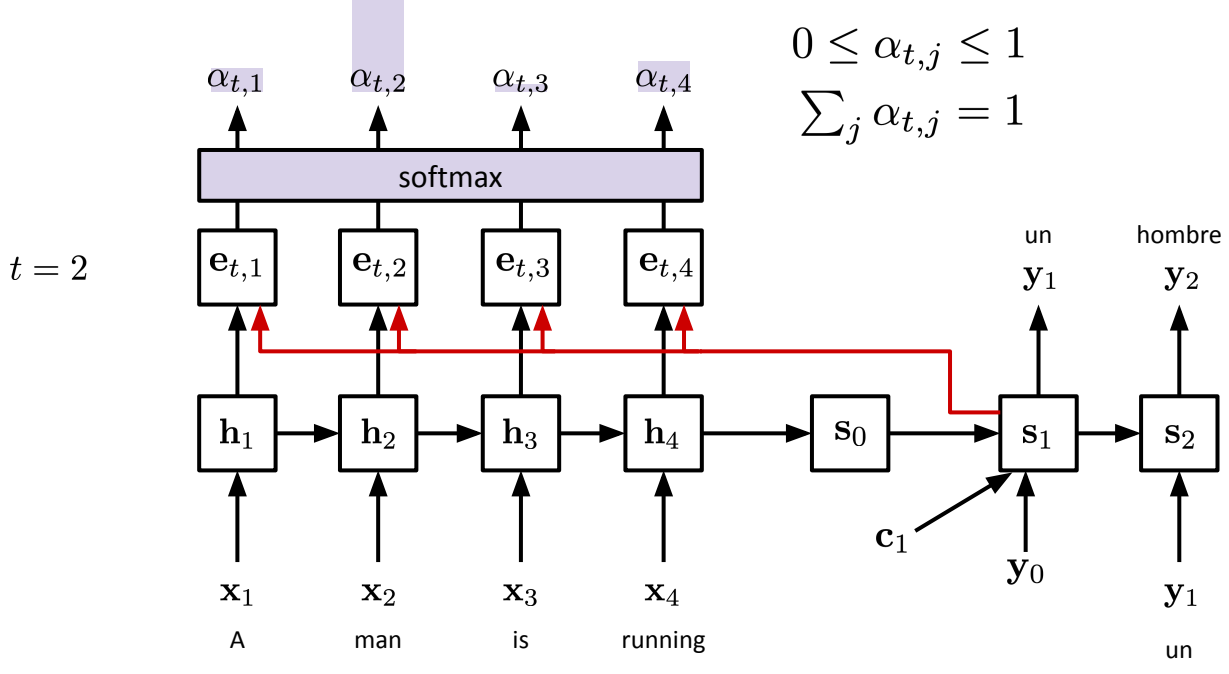
Consider a decoder step $t = 2$:

$\mathbf{e}_{t,j} = f_{\text{att}}(\mathbf{s}_{t-1}, \mathbf{h}_j)$ (e.g., a dot product, bilinear form, or any learnable MLP, etc.)



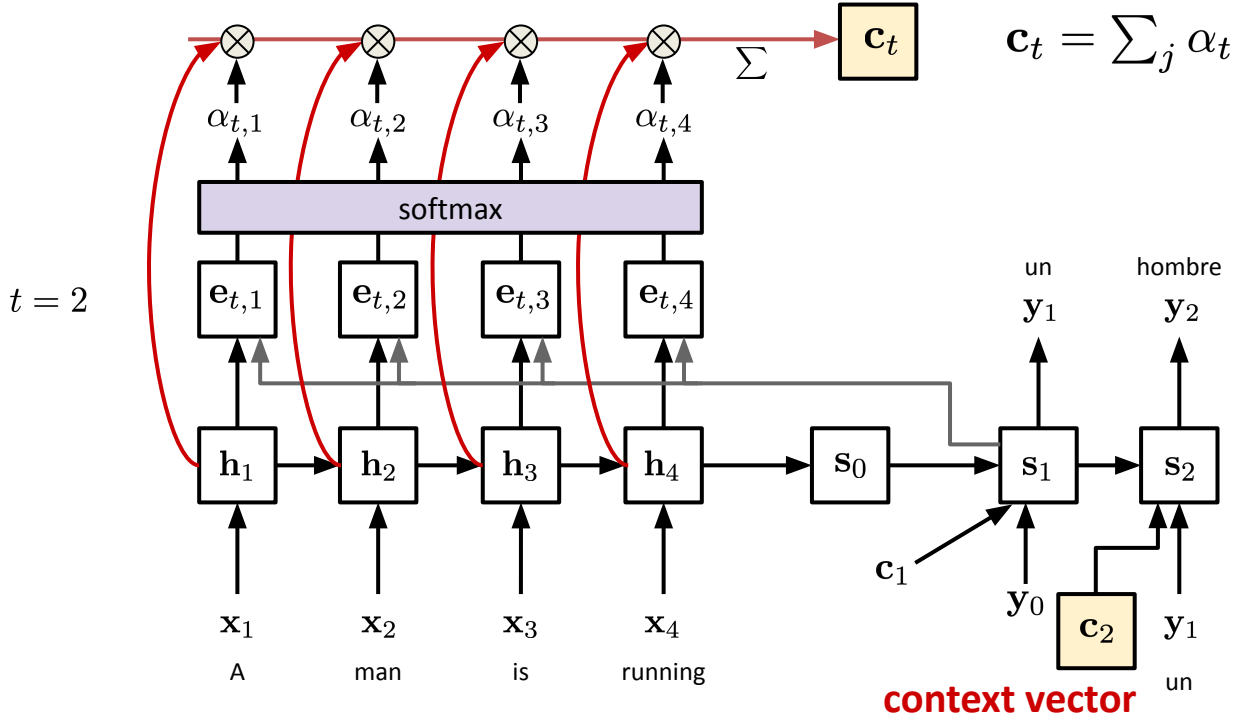
Attention Mechanism

- Compute alignment score at timestep ***t***: $\mathbf{e}_{t,j} = f_{\text{att}}(\mathbf{s}_{t-1}, \mathbf{h}_j)$
- Normalize alignment scores with softmax to get attention weights: $\boldsymbol{\alpha}_t = \text{softmax}(\mathbf{e}_t)$

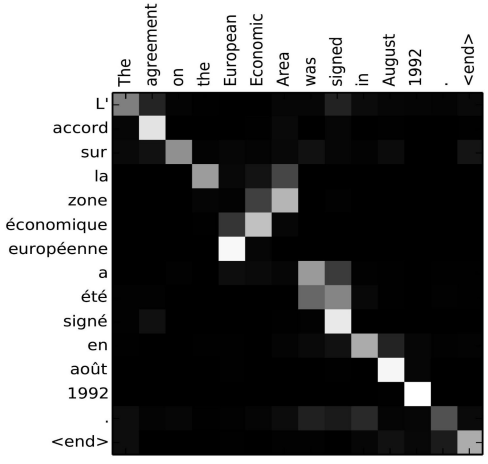


Attention Mechanism

- Compute attention weights via softmax: $\boldsymbol{\alpha}_t = \mathrm{softmax}(\mathbf{e}_t)$
- Compute context vector (attention readout) as the linear combination of input features



Maybe useful visualizations for attention x translation



Input (English):

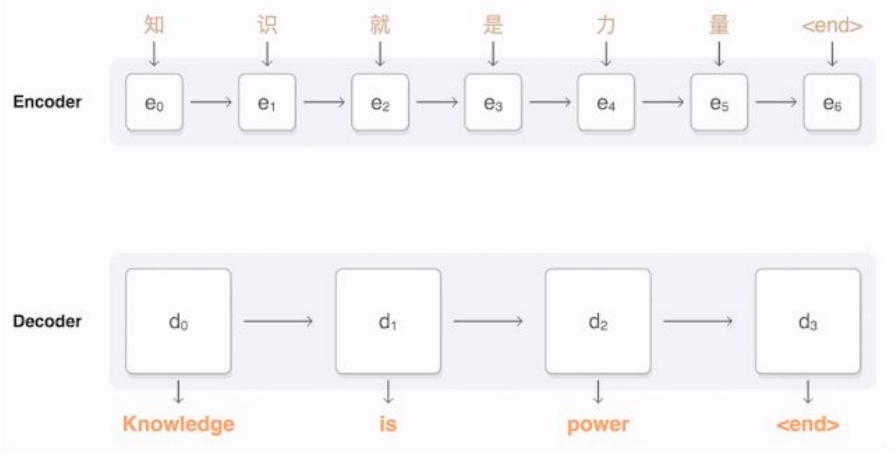
"The agreement on the European Economic Area was signed in August 1992."

Output (French):

"L'accord sur la zone économique européenne a été signé en août 1992."

Fig. 5. Alignment matrix of "The agreement on the European Economic Area was signed in August 1992" (English) and its French translation "L'accord sur l'Espace économique européen a été signé en août 1992".
(Image source: Fig 3 in Bahdanau et al., 2015)

The following visualization shows the progression of GNMT as it translates a Chinese sentence to English. First, the network encodes the Chinese words as a list of vectors, where each vector represents the meaning of all words read so far ("Encoder"). Once the entire sentence is read, the decoder begins, generating the English sentence one word at a time ("Decoder"). To generate the translated word at each step, the decoder pays attention to a weighted distribution over the encoded Chinese vectors most relevant to generate the English word ("Attention"; the blue link transparency represents how much the decoder pays attention to an encoded word).



https://ai.googleblog.com/2016/09/a-neural-network-for-machine.html

Image Captioning with Attention

A dog is standing on a hardwood floor.

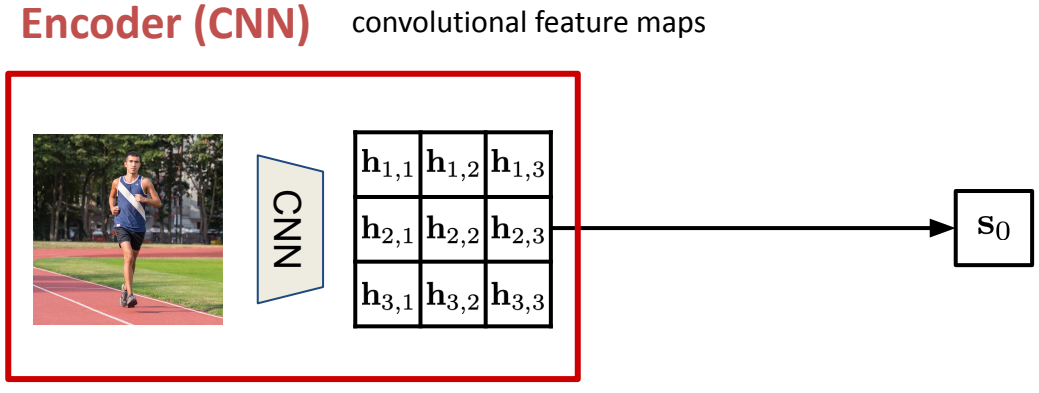
A stop sign is on a road with a mountain in the background.

A group of people sitting on a boat in the water.

A giraffe standing in a forest with trees in the background.

Source: “Show-Attend-Tell” (Xu et al., 2015)

Image Captioning with Attention



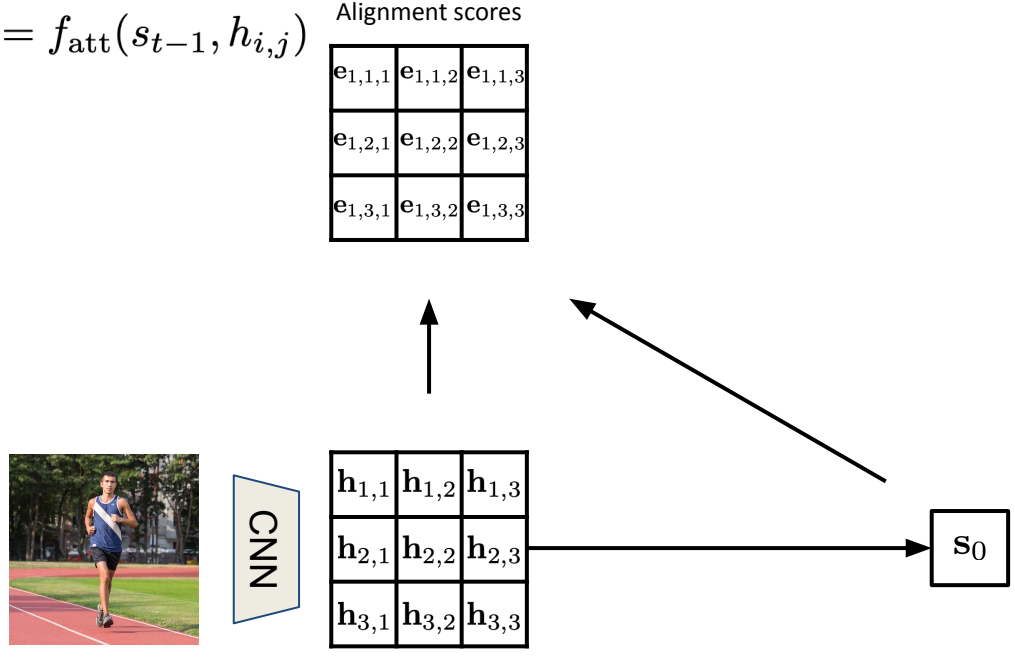
slide credit: Justin Johnson

Image Captioning with Attention

t=1

$$\mathbf{e}_{t,i,j} = f_{\text{att}}(s_{t-1}, h_{i,j})$$

$$\mathbf{e}_{t,i,j} = f_{\text{att}}(s_{t-1}, h_{i,j})$$



Use a CNN to get a
grid of spatial features

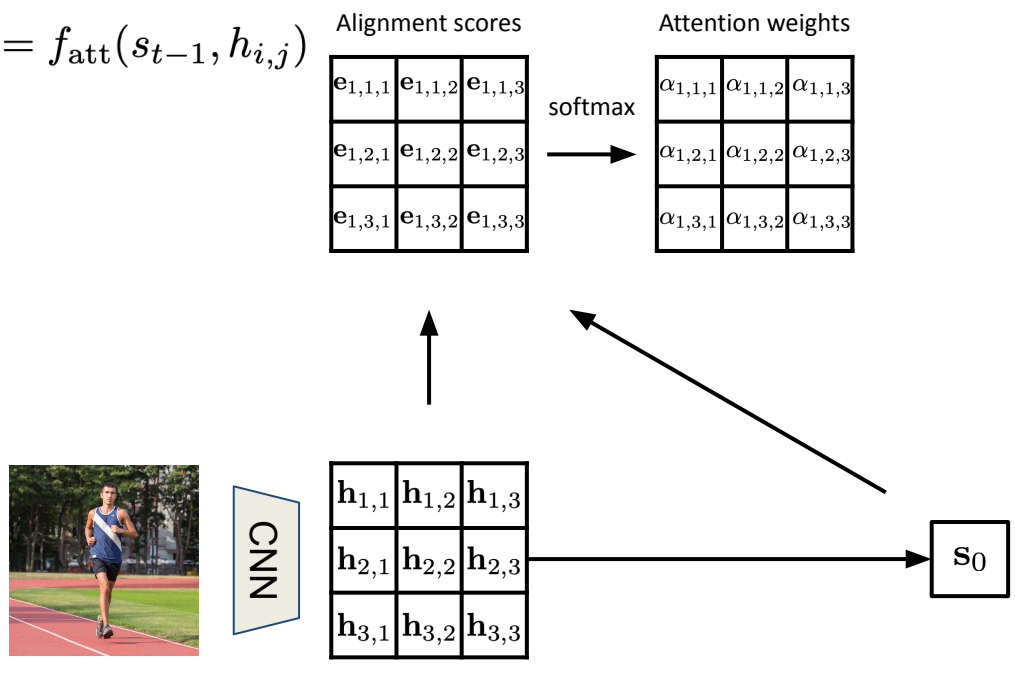
slide credit: Justin Johnson

Image Captioning with Attention

$$\alpha_{t,i,j} = \text{softmax}(e_{t,i,j})$$

t=1

$$\mathbf{e}_{t,i,j} = f_{\text{att}}(s_{t-1}, h_{i,j})$$



$$\mathbf{e}_{t,i,j} = f_{\text{att}}(s_{t-1}, h_{i,j})$$
$$\alpha_{t,i,j} = \text{softmax}(e_{t,i,j})$$

Use a CNN to get a
grid of spatial features

slide credit: Justin Johnson

Image Captioning with Attention

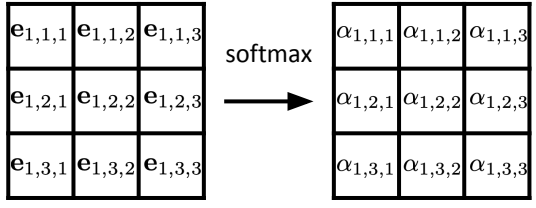
$$\alpha_{t,i,j} = \text{softmax}(e_{t,i,j})$$

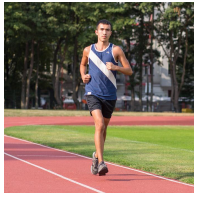
t=1

$$\mathbf{e}_{t,i,j} = f_{\text{att}}(s_{t-1}, h_{i,j})$$

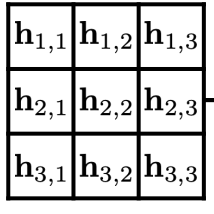
Alignment scores

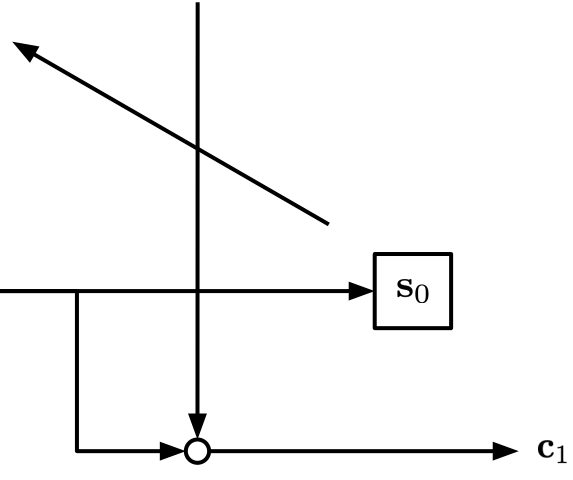
Attention weights





CNN





Use a CNN to get a grid of spatial features

$$\mathbf{c}_t = \sum_{i,j} \alpha_{t,i,j} \mathbf{h}_{i,j}$$

$$\mathbf{e}_{t,i,j} = f_{\text{att}}(s_{t-1}, h_{i,j})$$

$$\alpha_{t,i,j} = \text{softmax}(e_{t,i,j})$$

$$\mathbf{c}_t = \sum_{i,j} \alpha_{t,i,j} \mathbf{h}_{i,j}$$

slide credit: Justin Johnson

Image Captioning with Attention

$$\alpha_{t,i,j} = \text{softmax}(e_{t,i,j})$$

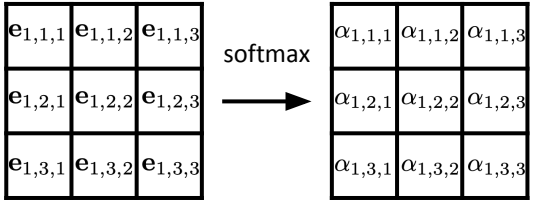
t=1

$$\mathbf{e}_{t,i,j} = f_{\text{att}}(s_{t-1}, h_{i,j})$$

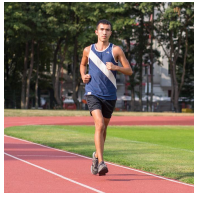
Alignment scores

Attention weights

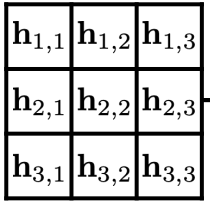
softmax

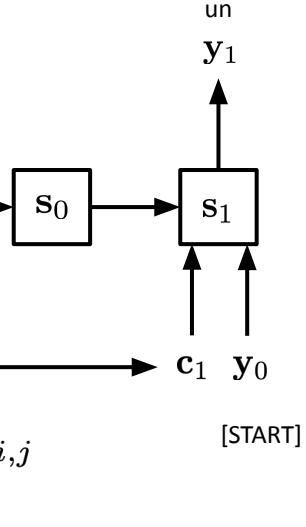


$$\mathbf{e}_{t,i,j} = f_{\text{att}}(s_{t-1}, h_{i,j})$$
$$\alpha_{t,i,j} = \text{softmax}(e_{t,i,j})$$
$$\mathbf{c}_t = \sum_{i,j} \alpha_{t,i,j} \mathbf{h}_{i,j}$$



CNN





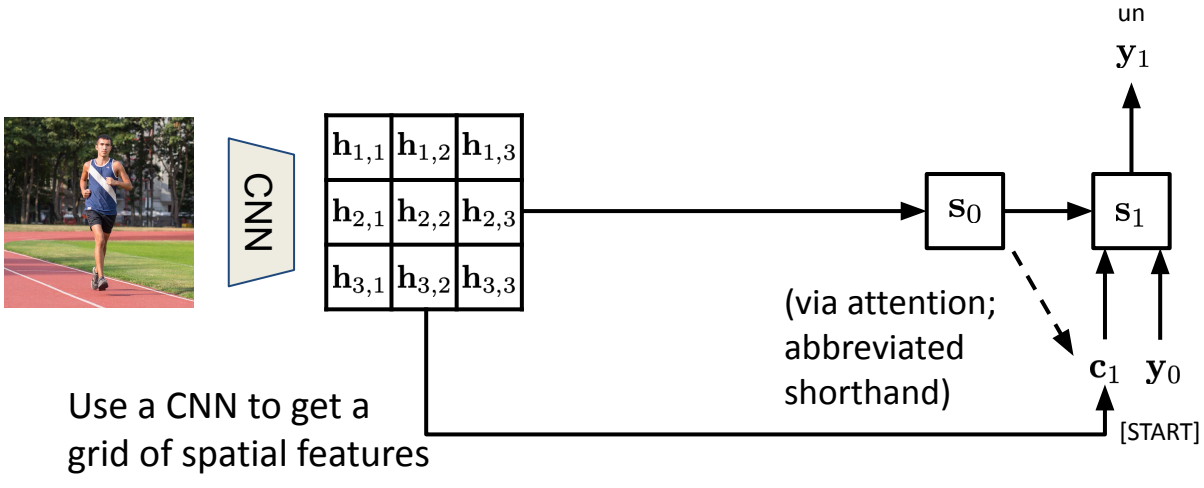
Use a CNN to get a grid of spatial features

$$\mathbf{c}_t = \sum_{i,j} \alpha_{t,i,j} \mathbf{h}_{i,j}$$

slide credit: Justin Johnson

Image Captioning with Attention

$$\mathbf{e}_{t,i,j} = f_{\text{att}}(s_{t-1}, h_{i,j})$$
$$\alpha_{t,i,j} = \text{softmax}(e_{t,i,j})$$
$$\mathbf{c}_t = \sum_{i,j} \alpha_{t,i,j} \mathbf{h}_{i,j}$$



slide credit: Justin Johnson

Image Captioning with Attention

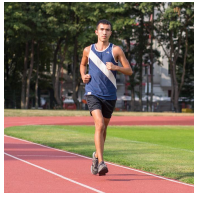
t=2

$$\mathbf{e}_{t,i,j} = f_{\text{att}}(s_{t-1}, h_{i,j})$$

Alignment scores

Repeat the process
for the next time step:

$$\mathbf{e}_{t,i,j} = f_{\text{att}}(s_{t-1}, h_{i,j})$$
$$\alpha_{t,i,j} = \text{softmax}(e_{t,i,j})$$
$$\mathbf{c}_t = \sum_{i,j} \alpha_{t,i,j} \mathbf{h}_{i,j}$$



CNN

$\mathbf{h}_{1,1}$ $\mathbf{h}_{1,2}$ $\mathbf{h}_{1,3}$
$\mathbf{h}_{2,1}$ $\mathbf{h}_{2,2}$ $\mathbf{h}_{2,3}$
$\mathbf{h}_{3,1}$ $\mathbf{h}_{3,2}$ $\mathbf{h}_{3,3}$

$\mathbf{s}_0$ $\mathbf{s}_1$

un
$\mathbf{y}_1$

$\mathbf{c}_1$ $\mathbf{y}_0$

[START]

Use a CNN to get a
grid of spatial features

slide credit: Justin Johnson

Image Captioning with Attention

t=2

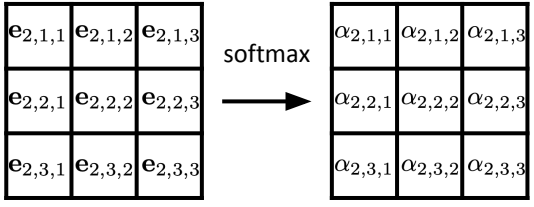
$$\mathbf{e}_{t,i,j} = f_{\text{att}}(s_{t-1}, h_{i,j})$$

$$\alpha_{t,i,j} = \text{softmax}(e_{t,i,j})$$

Alignment scores

Attention weights

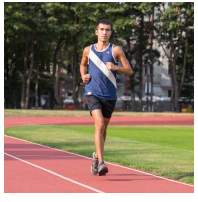
Repeat the process
for the next time step:



$$\mathbf{e}_{t,i,j} = f_{\text{att}}(s_{t-1}, h_{i,j})$$

$$\alpha_{t,i,j} = \text{softmax}(e_{t,i,j})$$

$$\mathbf{c}_t = \sum_{i,j} \alpha_{t,i,j} \mathbf{h}_{i,j}$$



CNN

$\mathbf{h}_{1,1}$ $\mathbf{h}_{1,2}$ $\mathbf{h}_{1,3}$
$\mathbf{h}_{2,1}$ $\mathbf{h}_{2,2}$ $\mathbf{h}_{2,3}$
$\mathbf{h}_{3,1}$ $\mathbf{h}_{3,2}$ $\mathbf{h}_{3,3}$

$\mathbf{s}_0$ $\mathbf{s}_1$

un

$\mathbf{y}_1$

$\mathbf{c}_1$ $\mathbf{y}_0$

[START]

Use a CNN to get a
grid of spatial features

slide credit: Justin Johnson

Image Captioning with Attention

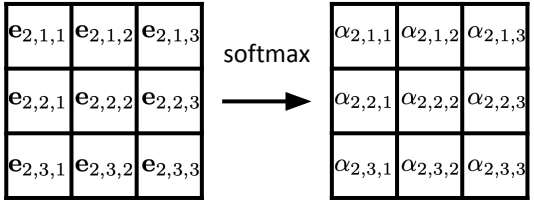
t=2

$$\mathbf{e}_{t,i,j} = f_{\text{att}}(s_{t-1}, h_{i,j})$$

$$\alpha_{t,i,j} = \text{softmax}(e_{t,i,j})$$

Alignment scores

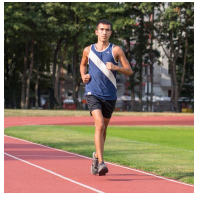
Attention weights



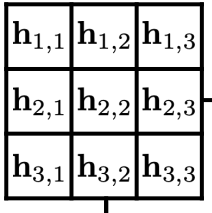
$$\mathbf{e}_{t,i,j} = f_{\text{att}}(s_{t-1}, h_{i,j})$$

$$\alpha_{t,i,j} = \text{softmax}(e_{t,i,j})$$

$$\mathbf{c}_t = \sum_{i,j} \alpha_{t,i,j} \mathbf{h}_{i,j}$$



CNN



Use a CNN to get a grid of spatial features

slide credit: Justin Johnson

Image Captioning with Attention

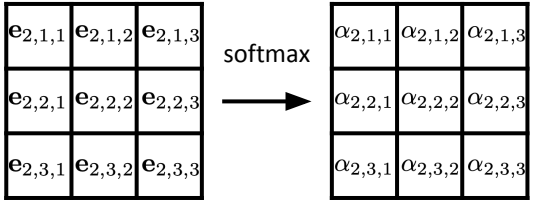
t=2

$$\mathbf{e}_{t,i,j} = f_{\text{att}}(s_{t-1}, h_{i,j})$$

$$\alpha_{t,i,j} = \text{softmax}(e_{t,i,j})$$

Alignment scores

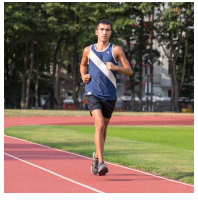
Attention weights



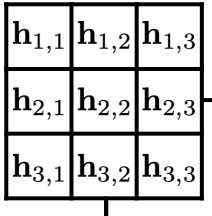
$$\mathbf{e}_{t,i,j} = f_{\text{att}}(s_{t-1}, h_{i,j})$$

$$\alpha_{t,i,j} = \text{softmax}(e_{t,i,j})$$

$$\mathbf{c}_t = \sum_{i,j} \alpha_{t,i,j} \mathbf{h}_{i,j}$$



CNN



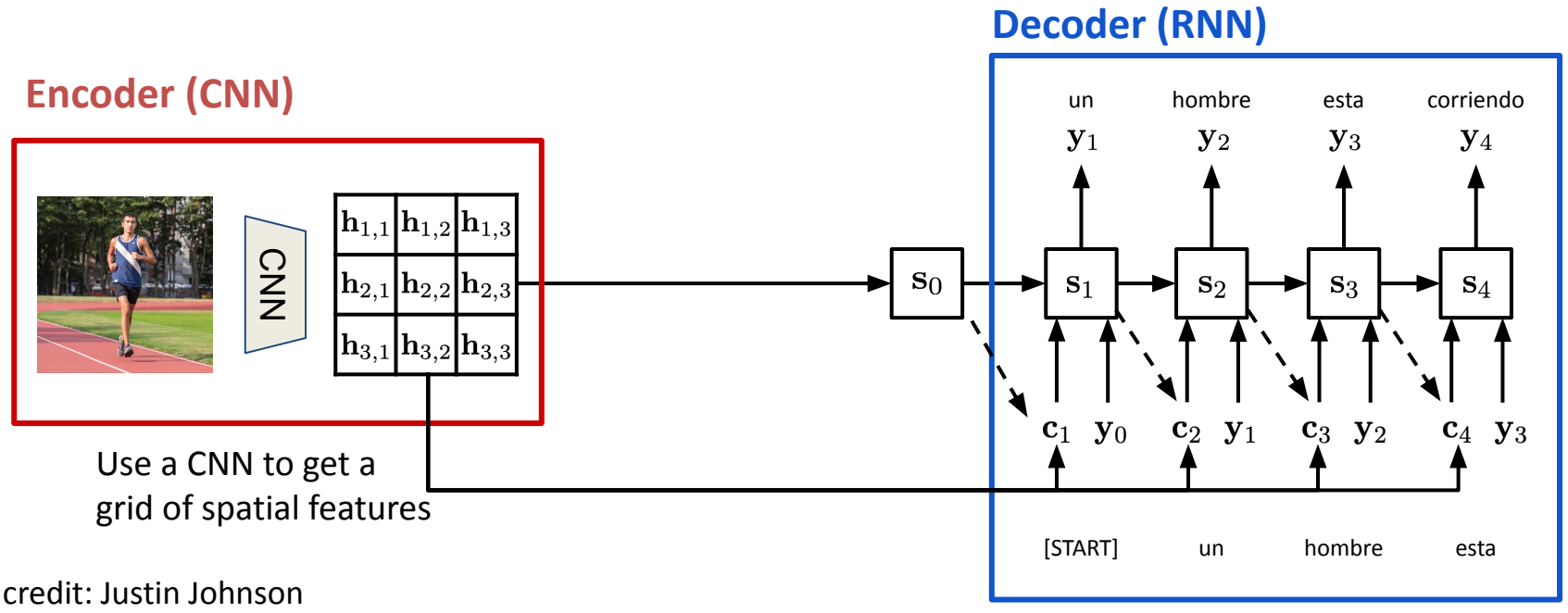
Use a CNN to get a grid of spatial features

slide credit: Justin Johnson

Image Captioning with Attention

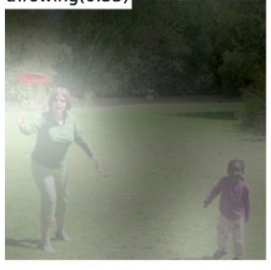
- Each timestep of decoder uses a different context vector that looks at different parts of the input image

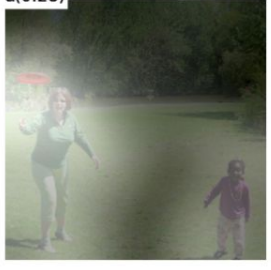
$$\mathbf{e}_{t,i,j} = f_{\text{att}}(s_{t-1}, h_{i,j})$$
$$\alpha_{t,i,j} = \text{softmax}(e_{t,i,j})$$
$$\mathbf{c}_t = \sum_{i,j} \alpha_{t,i,j} \mathbf{h}_{i,j}$$

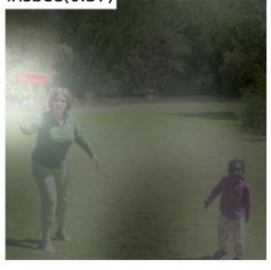


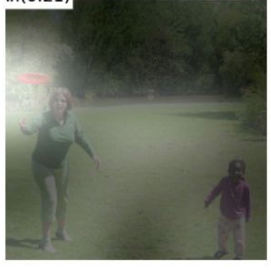
slide credit: Justin Johnson

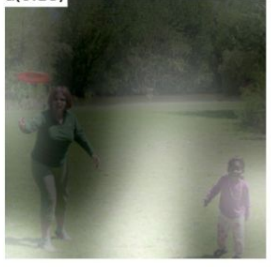
More examples

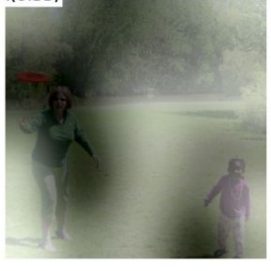






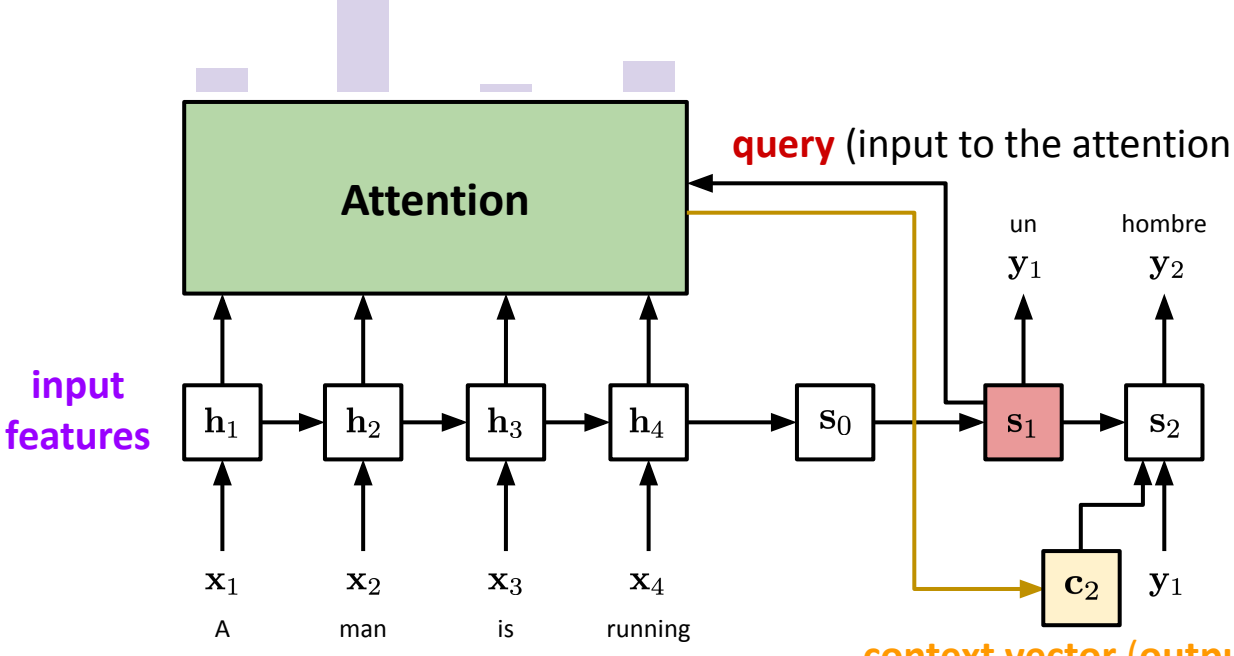






From Attention to Transformer

- Recall the attention layer in the seq2seq example:
  - **query**: the previous (decoder) hidden state
  - **input**: input features (e.g. sequence)
  - **output**: the context vector (the weighted sum w.r.t attention)

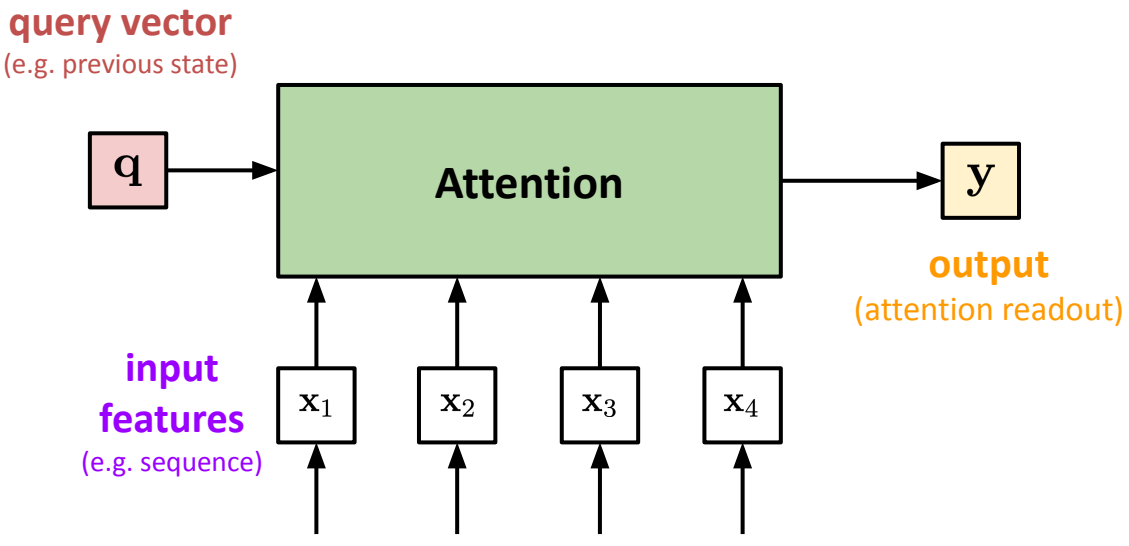


$$\mathbf{e}_{t,i} = f_{\text{att}}(\mathbf{s}_{t-1}, \mathbf{h}_i)$$
$$\boldsymbol{\alpha}_t = \text{softmax}(\mathbf{e}_t)$$
$$\mathbf{c}_t = \sum_i \alpha_{t,i} \mathbf{h}_i$$

From Attention to Transformer

- **input** vectors: $\mathbf{X}$ $[N \times d_X]$
- **query** vector: $\mathbf{q}$ $[d_Q]$
- **output**: $\mathbf{y}$ $[d_X]$

Note: For matrices, we use row-major notation, following the literature.



alignment (similarity) $e_j = f_{\text{att}}(\mathbf{q}, \mathbf{x}_j)$

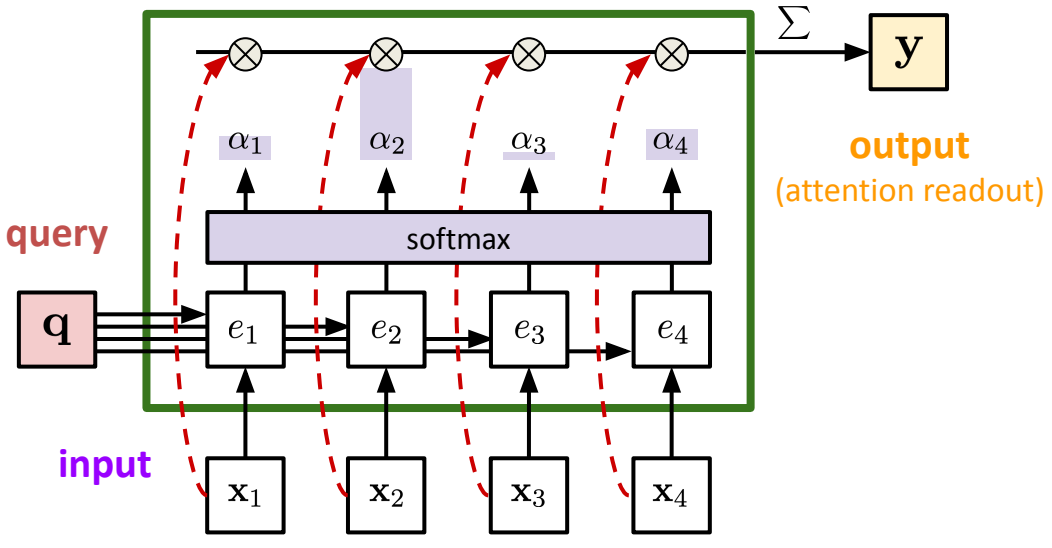
attention $\alpha_j = \frac{\exp(e_j)}{\sum_{j'} \exp(e_{j'})}$

output $\mathbf{y} = \sum_j \alpha_j \mathbf{x}_j$

From Attention to Transformer

- **input** vectors: $\mathbf{X}$ $[N \times d_X]$
- **query** vector: $\mathbf{q}$ $[d_Q]$
- **output**: $\mathbf{y}$ $[d_X]$

Inside of it: (recap of what we've already seen)



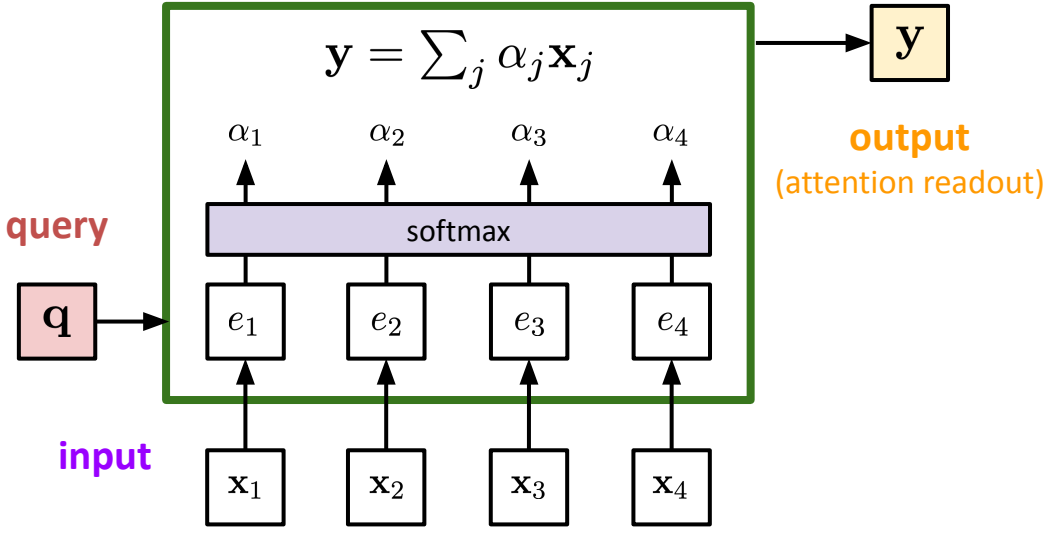
alignment (similarity) $e_j = f_{\text{att}}(\mathbf{q}, \mathbf{x}_j)$

attention $\alpha_j = \dfrac{\exp(e_j)}{\sum_{j'} \exp(e_{j'})}$

output $\mathbf{y} = \sum_j \alpha_j \mathbf{x}_j$

Scaled Dot Products

- Use **scaled dot product** to calculate the alignment: $e_j = \frac{\mathbf{q}^\top \mathbf{x}_j}{\sqrt{d}}$
  - Normalize the dot product by the factor $d^{0.5}$
  - The dot product $\mathbf{q}^\top \mathbf{x}$ becomes larger as the dimension d increases, where the gradient would become (too) small when softmax saturates.



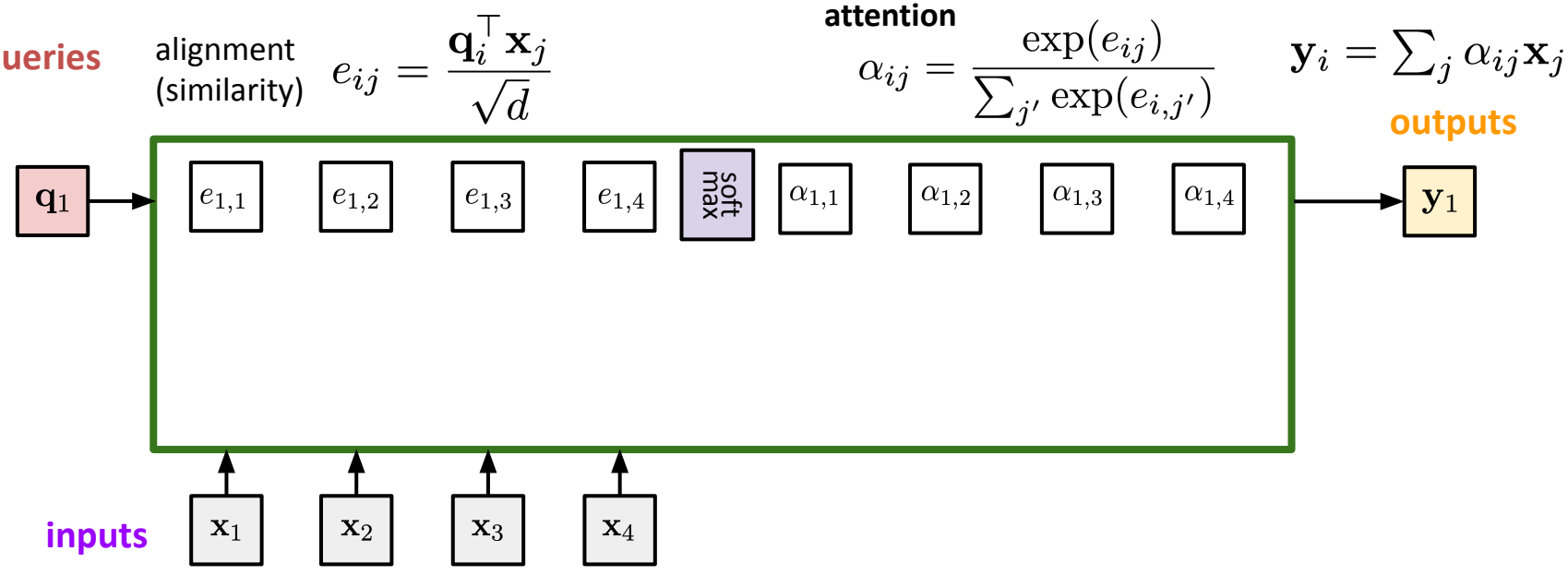
alignment (similarity) $e_j = \frac{\mathbf{q}^\top \mathbf{x}_j}{\sqrt{d}}$

attention $\alpha_j = \frac{\exp(e_j)}{\sum_{j'} \exp(e_{j'})}$

output $\mathbf{y} = \sum_j \alpha_j \mathbf{x}_j$

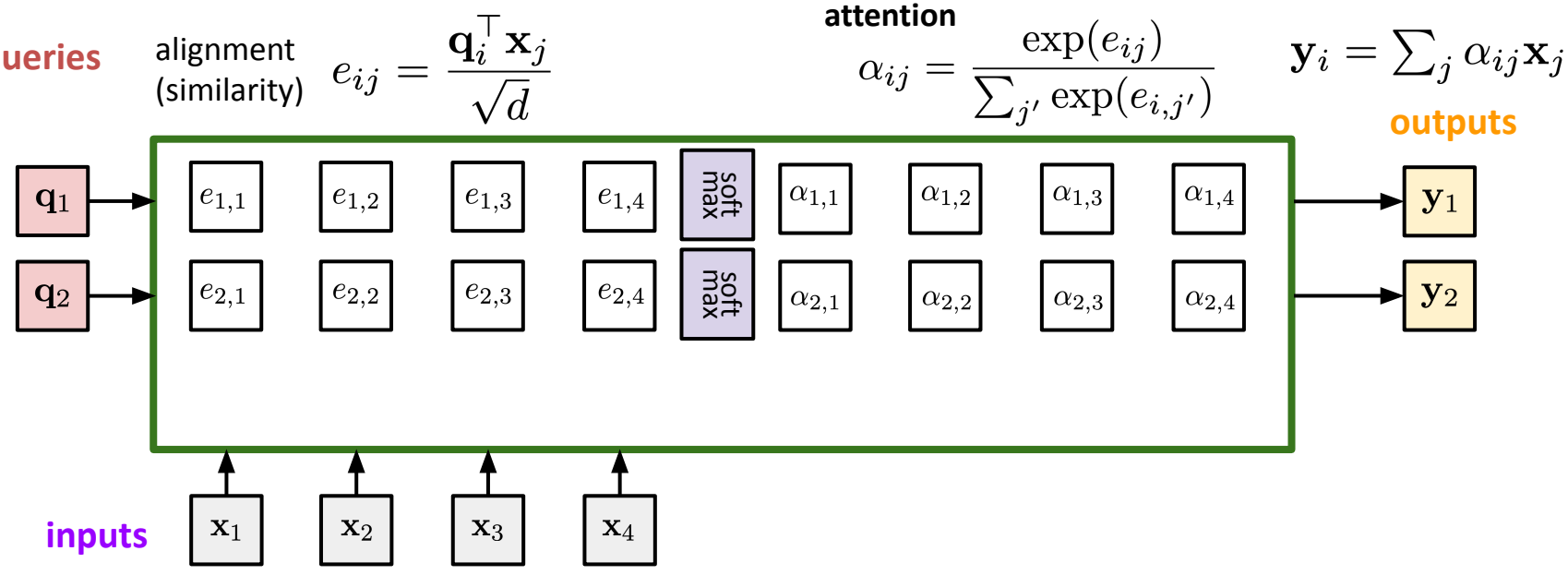
(General) Attention Layer

- Use multiple queries $\mathbf{Q} = (\mathbf{q}_1, \mathbf{q}_2, \cdots, \mathbf{q}_M) \quad [M \times d]$ to produce multiple context vectors as output



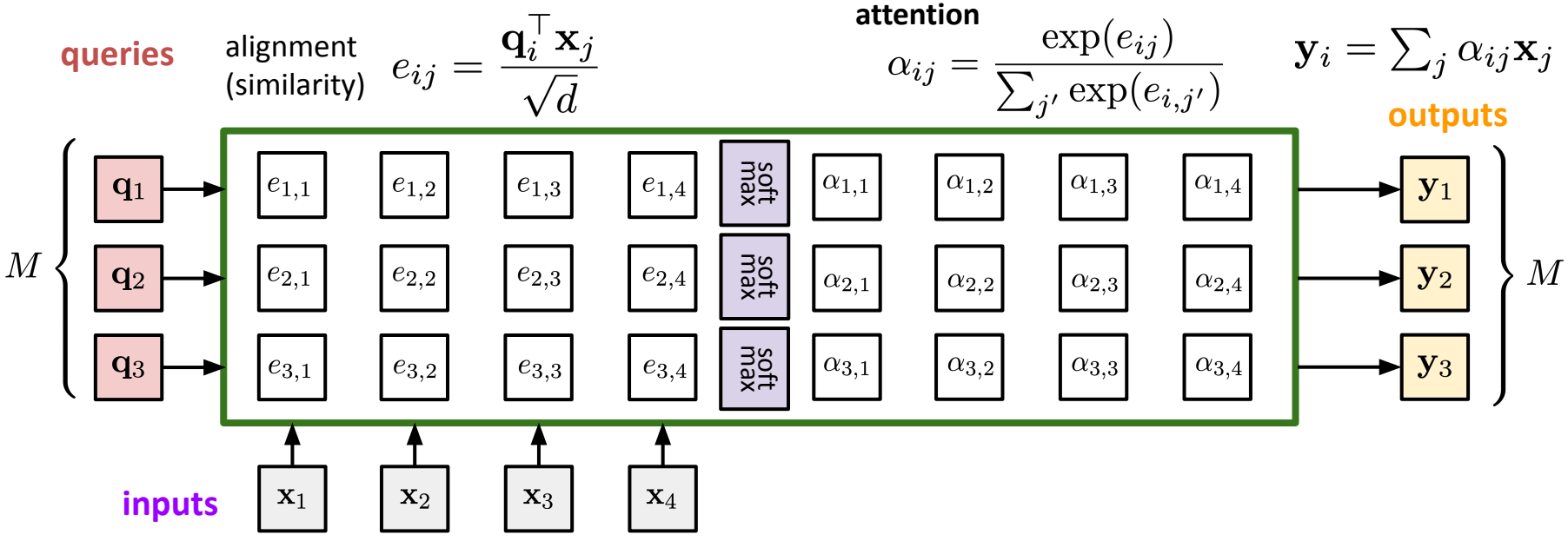
(General) Attention Layer

- Use multiple queries $\mathbf{Q} = (\mathbf{q}_1, \mathbf{q}_2, \cdots, \mathbf{q}_M) \quad [M \times d]$ to produce multiple context vectors as output



(General) Attention Layer

- Use multiple queries $\mathbf{Q} = (\mathbf{q}_1, \mathbf{q}_2, \cdots, \mathbf{q}_M) \quad [M \times d]$ to produce multiple context vectors as output



(General) Attention Layer

- Compute **keys** and **values** from the input features

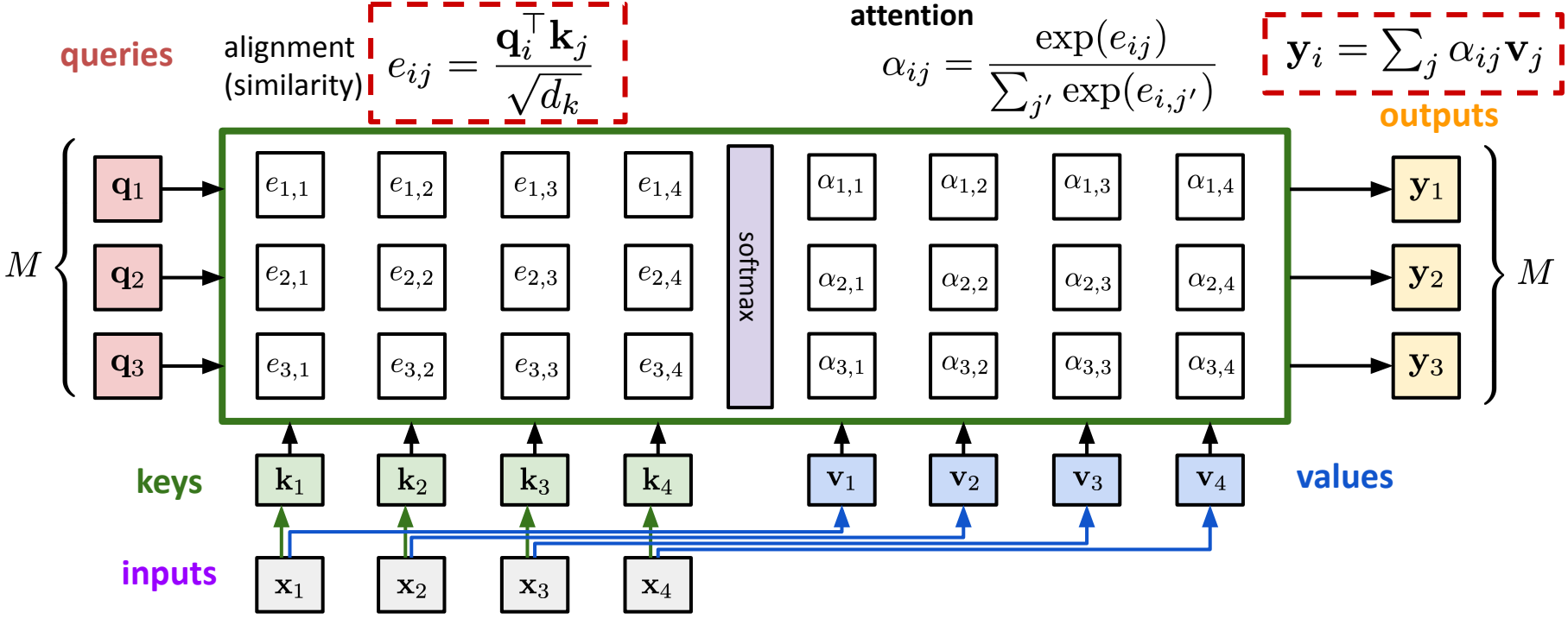
$$\mathbf{K} = \mathbf{X} \cdot \mathbf{W}_K$$

[N x d_K] [N x d_X] [d_X x d_K]

$$\mathbf{V} = \mathbf{X} \cdot \mathbf{W}_V$$

[N x d_V] [N x d_X] [d_X x d_V]

Note: For matrices, we use row-major notation, following the literature.



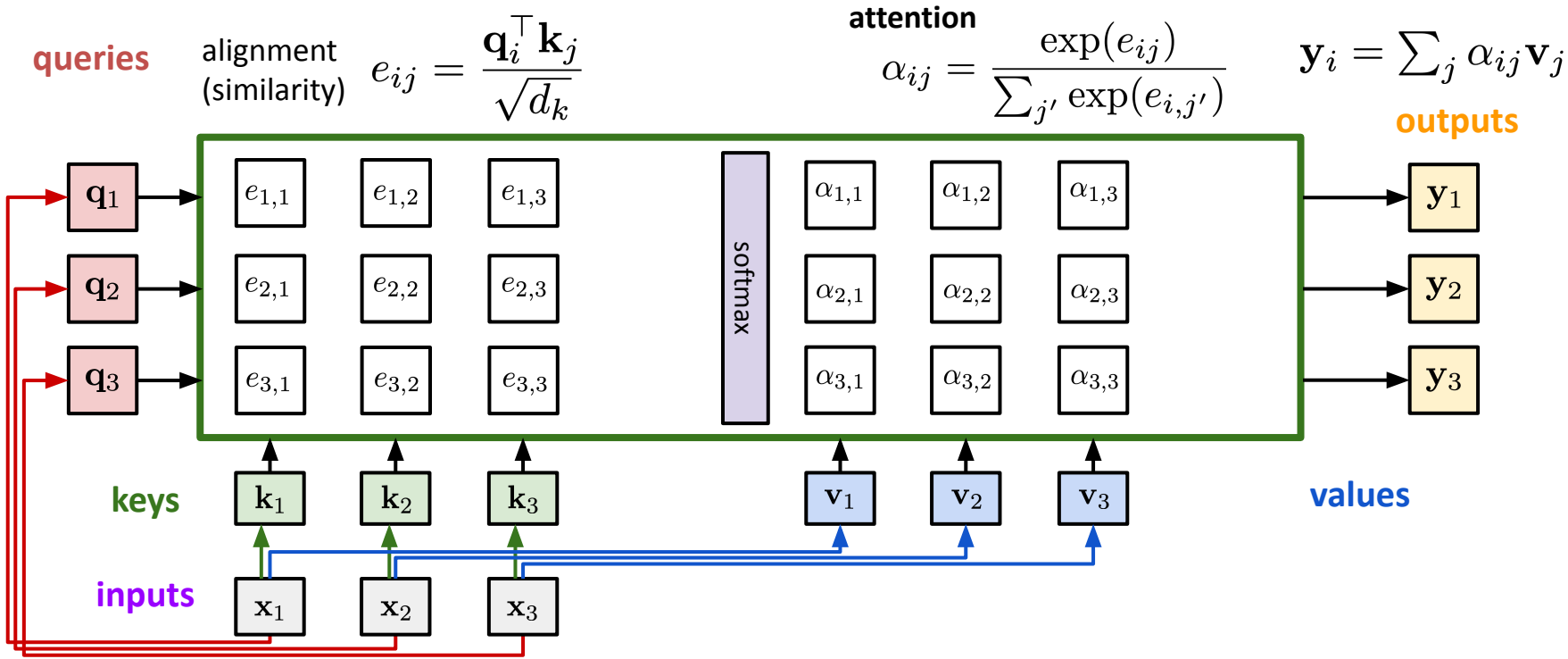
Self-Attention Layer

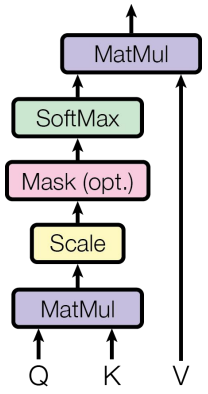
- Compute keys, values, and **queries** from the input features

$$\mathbf{K} = \mathbf{X} \cdot \mathbf{W}_\mathrm{K}$$
[N x d_K] [N x d_X] [d_X x d_K]

$$\mathbf{V} = \mathbf{X} \cdot \mathbf{W}_\mathrm{V}$$
[N x d_V] [N x d_X] [d_X x d_V]

$$\mathbf{Q} = \mathbf{X} \cdot \mathbf{W}_\mathrm{Q}$$
[N x d_K] [N x d_X] [d_X x d_K]





To summarize:

Self-Attention

Note: For matrices, we use row-major notation, following the literature.

(output: sequence of ***N*** vectors)

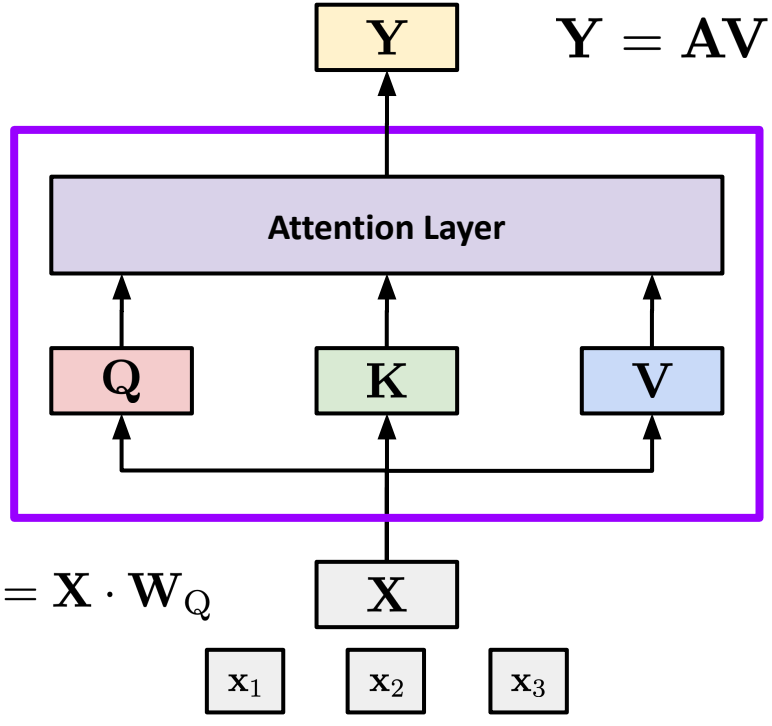
$$\mathbf{Y} = \mathbf{AV} \qquad \mathbf{y}_i = \sum_j \alpha_{ij} \mathbf{v}_j$$

$$\mathbf{A} = \text{softmax}\left(\frac{\mathbf{QK}^\top}{\sqrt{d_K}}\right)$$

$$e_{ij} = \frac{\mathbf{q}_i^\top \mathbf{k}_j}{\sqrt{d_k}}$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{j'} \exp(e_{i,j'})}$$

$$\mathbf{A} = (\alpha_{ij})$$



Self-attention block

$$\mathbf{K} = \mathbf{X} \cdot \mathbf{W}_\text{K}$$

$$\mathbf{V} = \mathbf{X} \cdot \mathbf{W}_\text{V}$$

$$\mathbf{Q} = \mathbf{X} \cdot \mathbf{W}_\text{Q}$$

(input: sequence of ***N*** vectors)

Self-Attention Layer: Visualizations

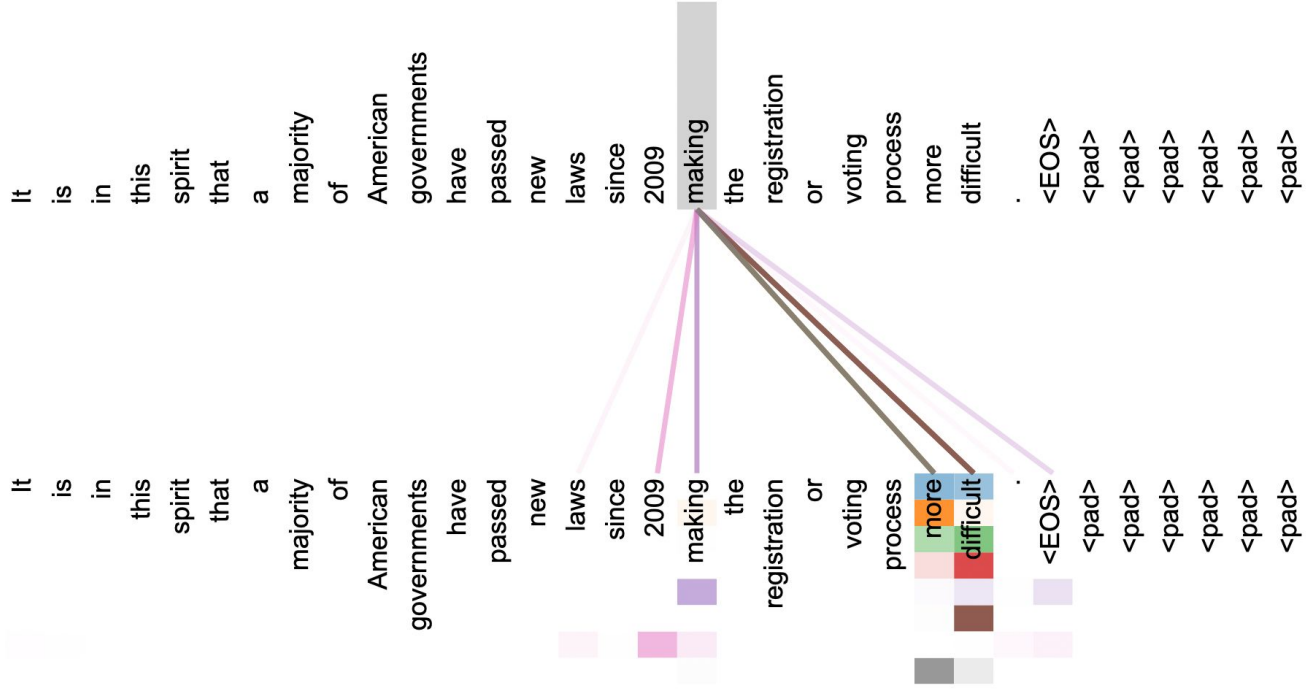
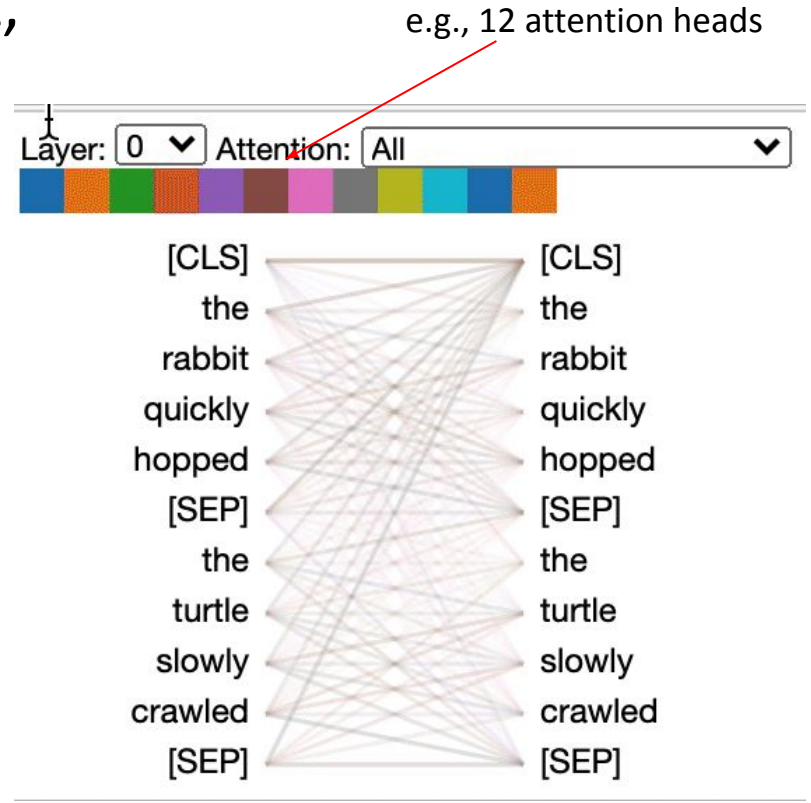


Figure 3: An example of the attention mechanism following long-distance dependencies in the encoder self-attention in layer 5 of 6. Many of the attention heads attend to a distant dependency of the verb 'making', completing the phrase 'making...more difficult'. Attentions here shown only for the word 'making'. Different colors represent different heads. Best viewed in color.

(Source: Vaswani et al., 2017)

Multi-head Attention: Motivation

What if we want to look at different, multiple places at once?



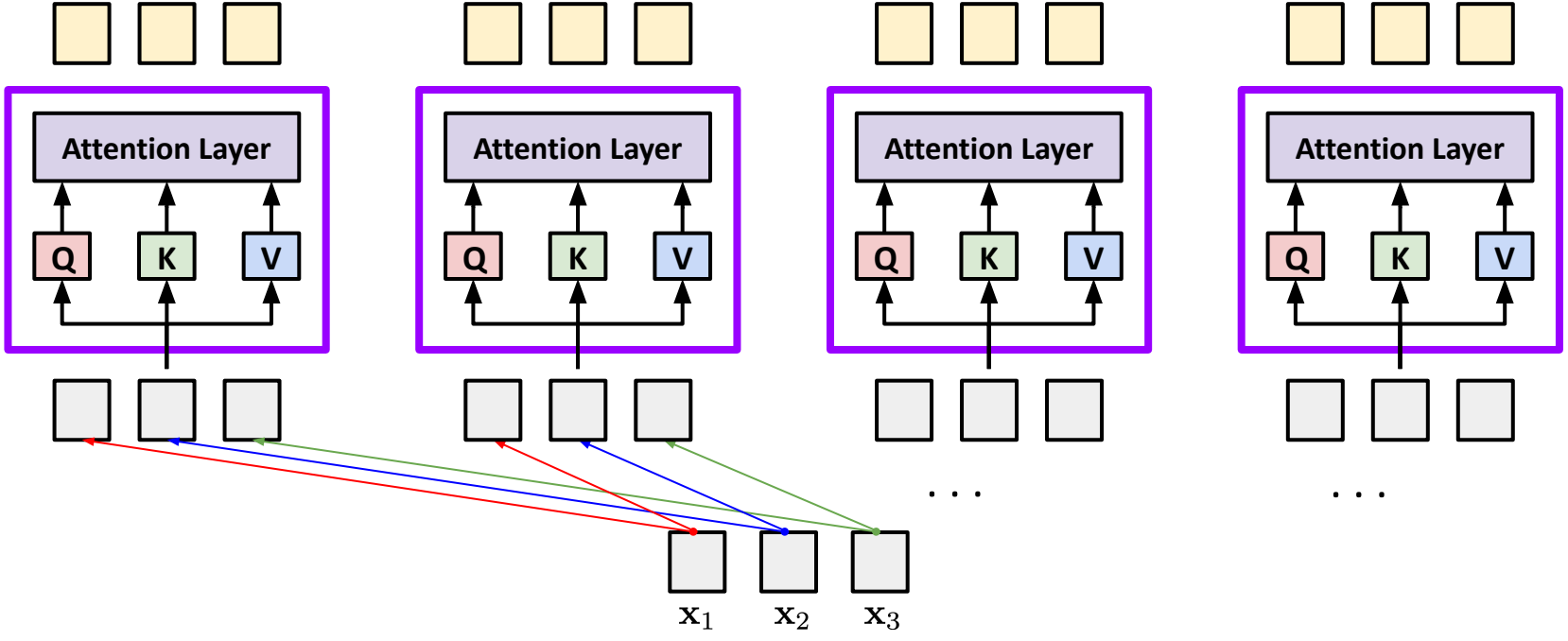
- With a (single) attention layer, we would attend to only one part of the sequence.
- But, for example, for a word "*it*":
  - What "it" refers to?
  - What happened to it?
  - What it is doing?
  - etc …

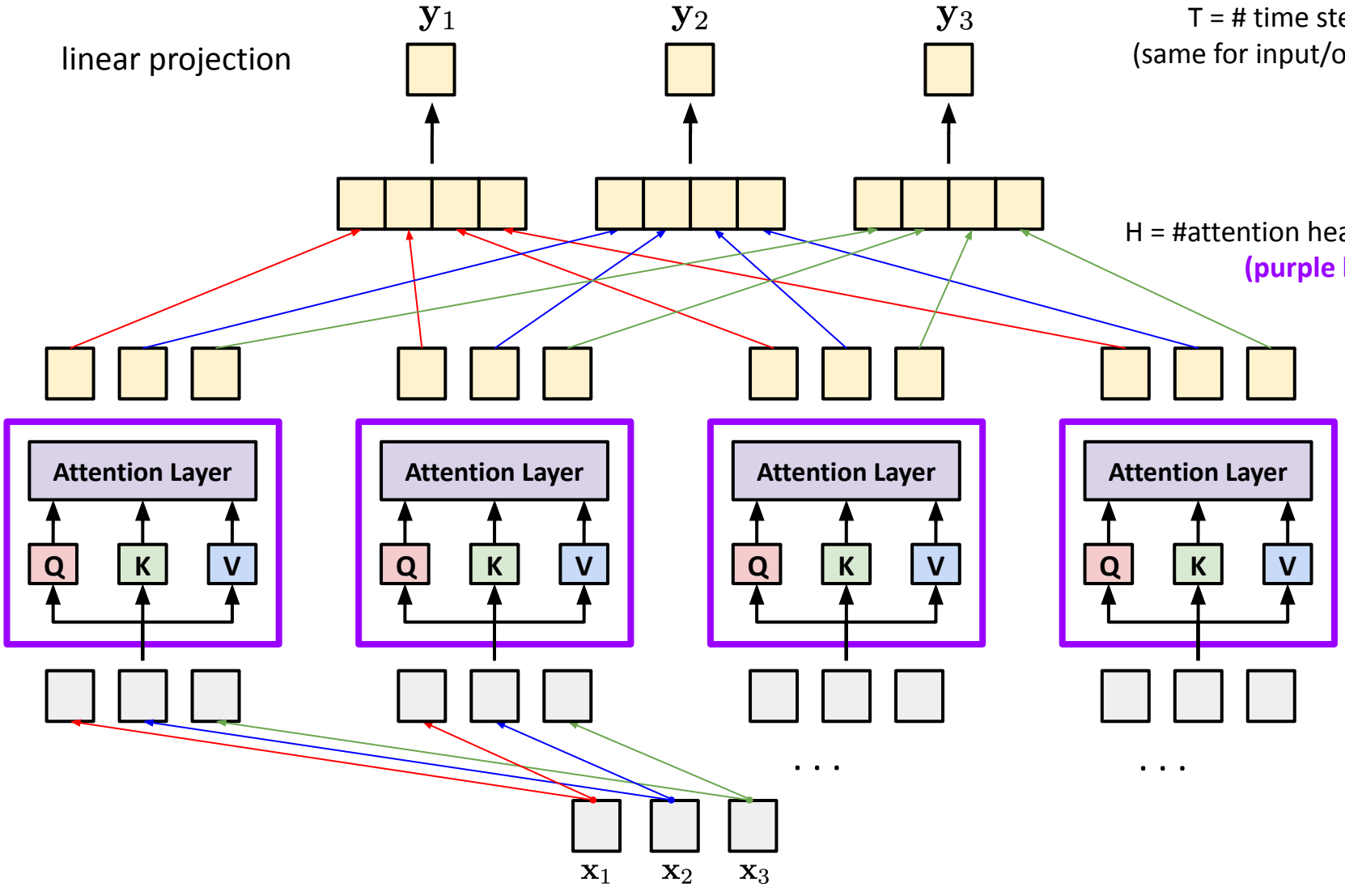
Visualization from https://github.com/jessevig/bertviz

Multi-head Attention

- What if we want to look at different, multiple places at once?
- Employ **multiple attention heads** to process input in parallel

Suppose we want to use **H** independent attention heads (e.g. H=4):

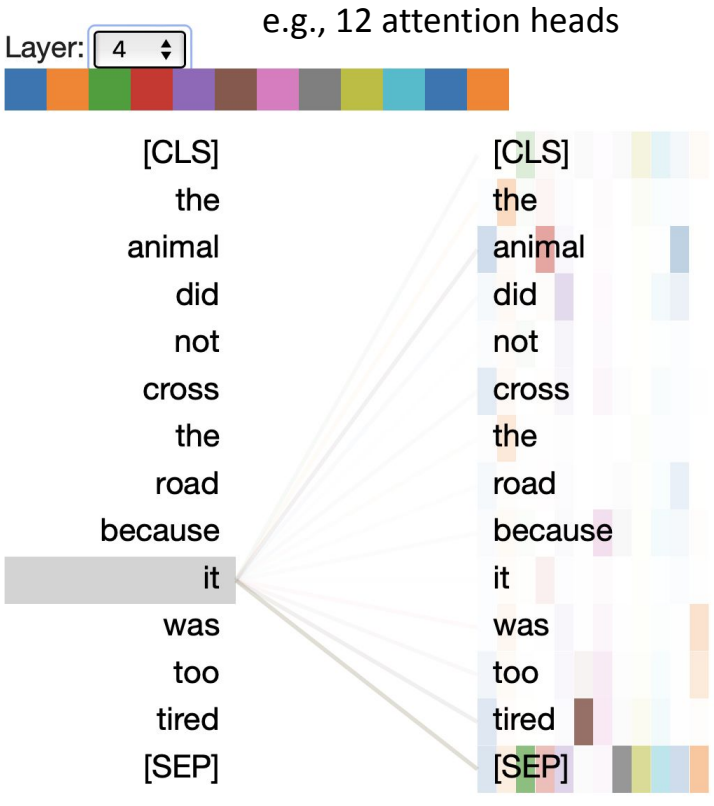



linear projection
$\mathbf{y}_1$
$\mathbf{y}_2$
$\mathbf{y}_3$
T = # time steps = 3
(same for input/output)
H = #attention heads = 4
(purple boxes)
Attention Layer
Q
K
V
Attention Layer
Q
K
V
Attention Layer
Q
K
V
Attention Layer
Q
K
V
$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_3$

Multi-head Attention Example

Multiple attention heads allows to make the same word attend to different parts independently:

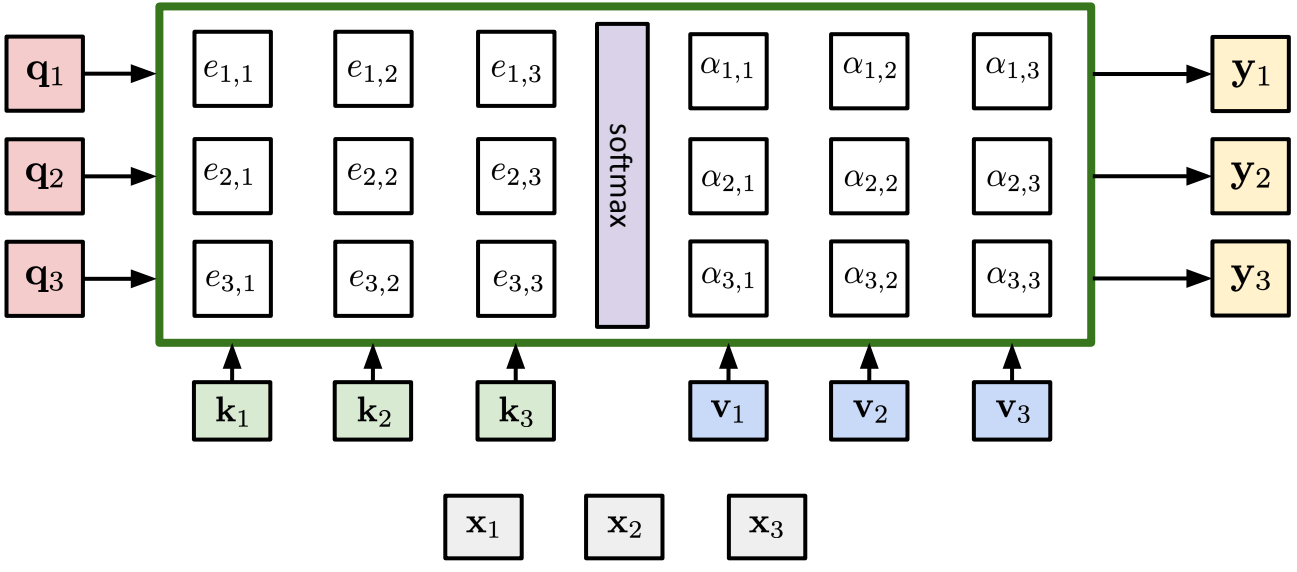
- Different weights per head!
  → Each head can learn a different thing
- For a word "*it*":
  - **one head** attends to "*animal*"
    (e.g., learned pronoun resolution)
  - **another head** attends to "*tired*"
    (e.g., learned some subject + complement semantics)
- Each head can focus on different parts of the input for **different reasons**.



Visualization from https://github.com/jessevig/bertviz

Self-Attention operates on “sets”

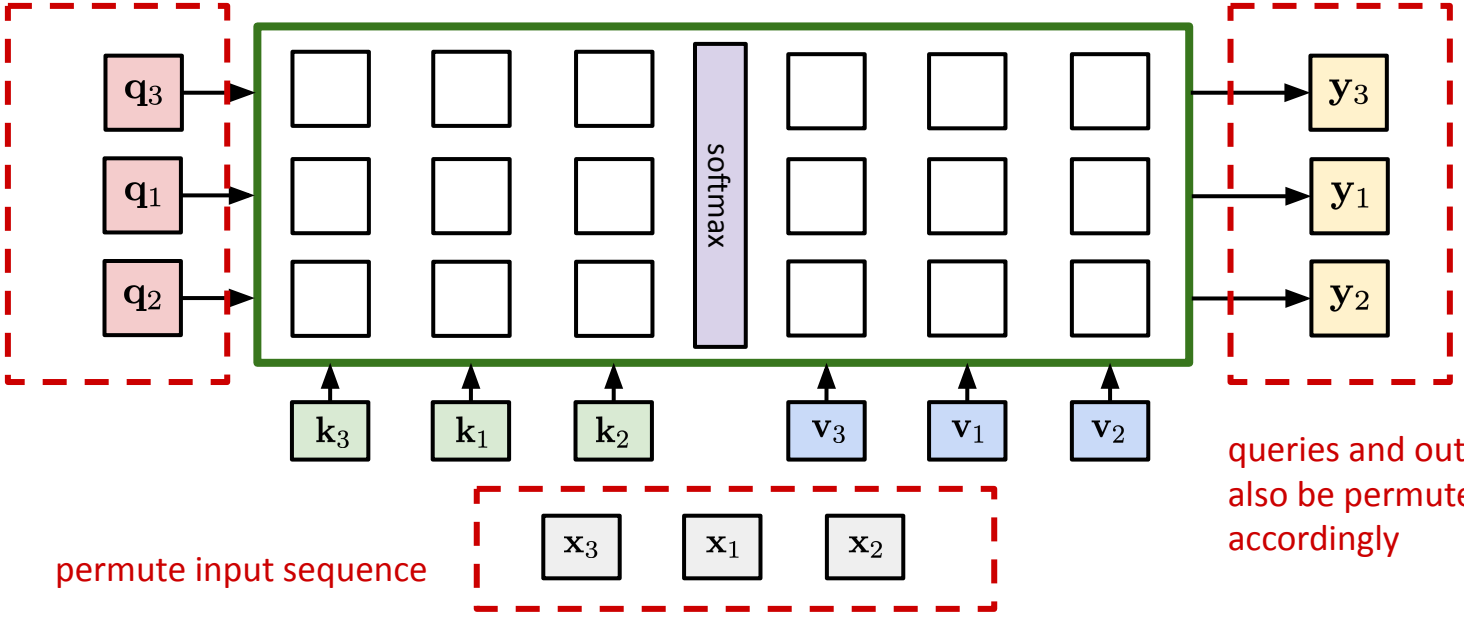
- Self-attention doesn’t have ordering information!



Self-Attention operates on “sets”

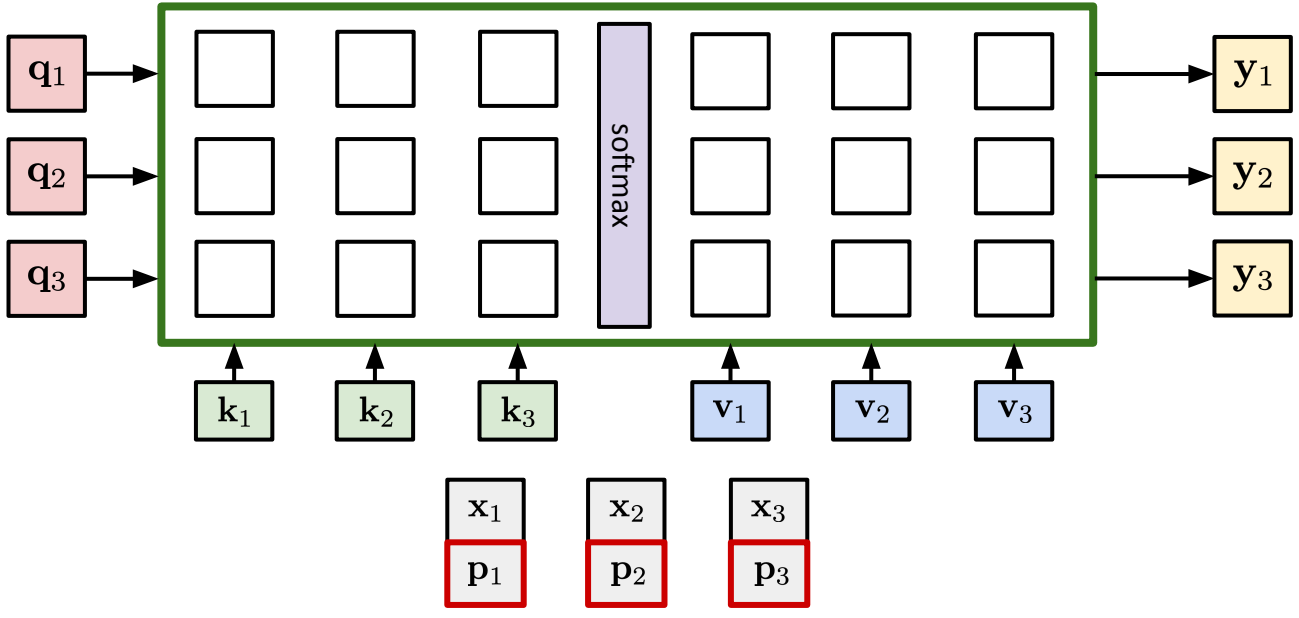
- Self-attention doesn’t have ordering information!
- When permuting the input vectors, outputs will also be permuted (permutation-equivariant)

f(perm(x)) = perm(f(x)), where f: S ⟶ S



Positional Encodings

- To make self attention aware of the order, concatenate or add a **positional encoding** to the input vector
  - e.g., $\widetilde{\mathbf{x}}_i = \mathbf{x}_i + \mathbf{p}_i$
  - The positional encoding can be a fixed or learnable vector

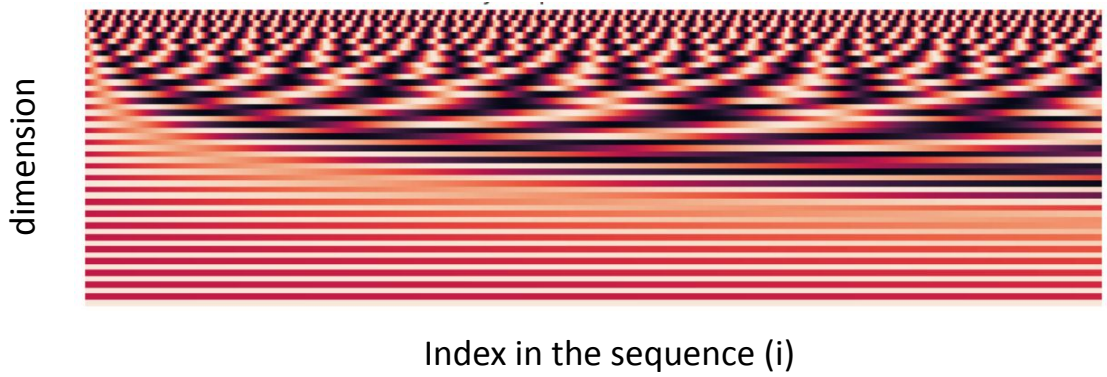


Positional Encodings

Example: Sinusoidal position representations

- a concatenation of sinusoidal functions of different periods

$$\mathrm{PE}(i,\delta) = \begin{cases} \sin\left(\frac{i}{10000^{2\delta'/d}}\right) & \text{if } \delta = 2\delta' \\ \cos\left(\frac{i}{10000^{2\delta'/d}}\right) & \text{if } \delta = 2\delta' + 1 \end{cases}$$



https://lilianweng.github.io/posts/2023-01-27-the-transformer-family-v2/#positional-encoding
https://timodenk.com/blog/linear-relationships-in-the-transformers-positional-encoding/

Positional Encodings

Example: Rotary positional encoding

- Rotary Positional Encoding (RoPE) integrates positional information directly into query/key of the attention mechanisms.
- Compared to sinusoidal encoding, RoPE preserves relative positions information more effectively

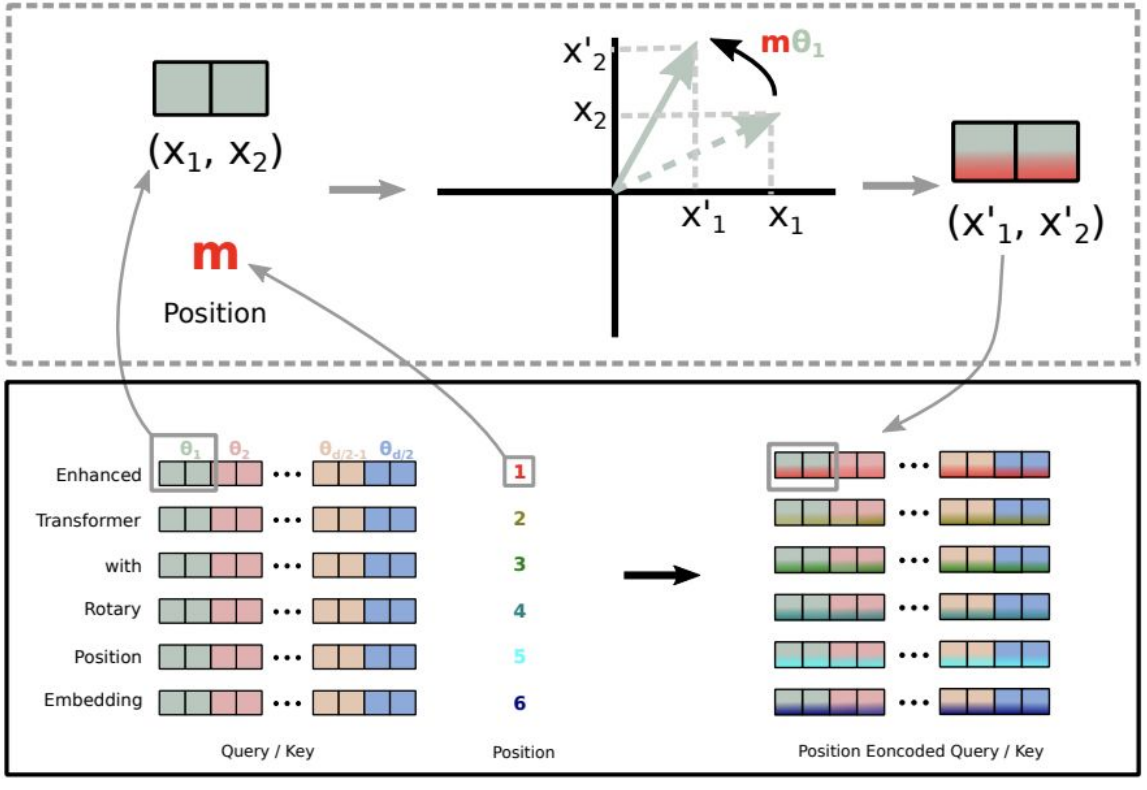
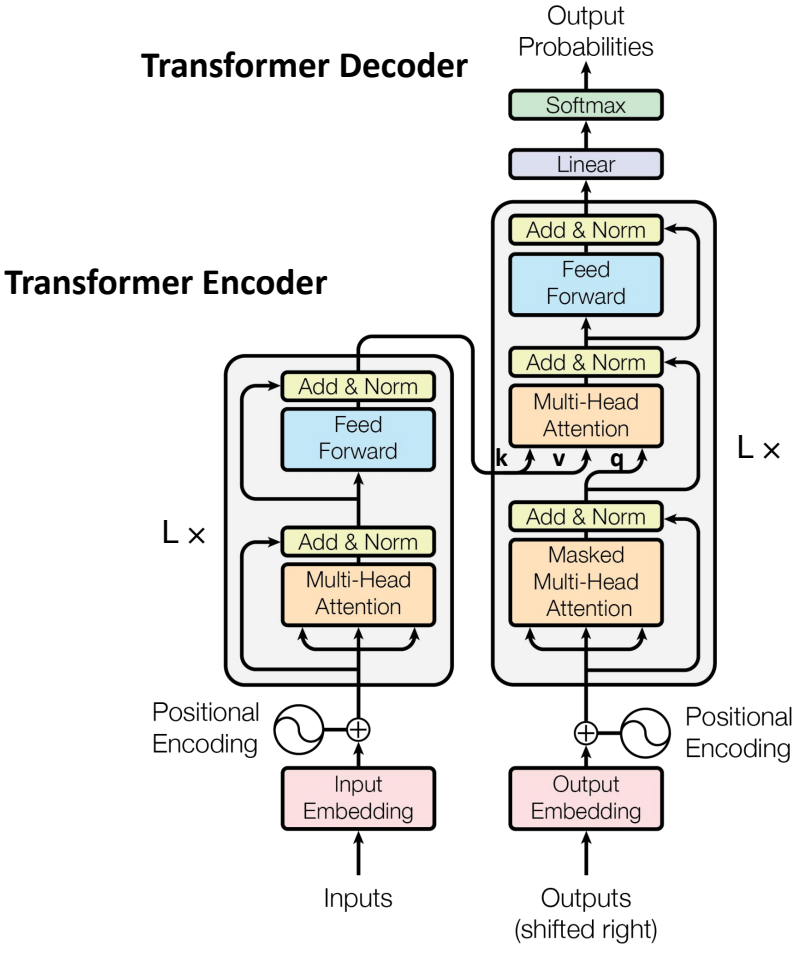


Image source: https://arxiv.org/pdf/2104.09864

The Transformer Network

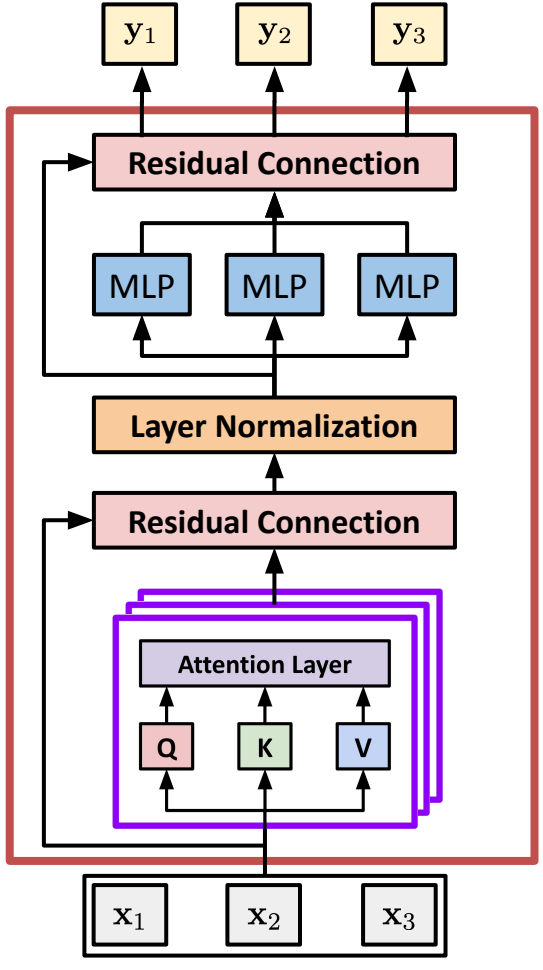
- "Attention is all you Need" (Vaswani et al., 2017)



The Transformer block

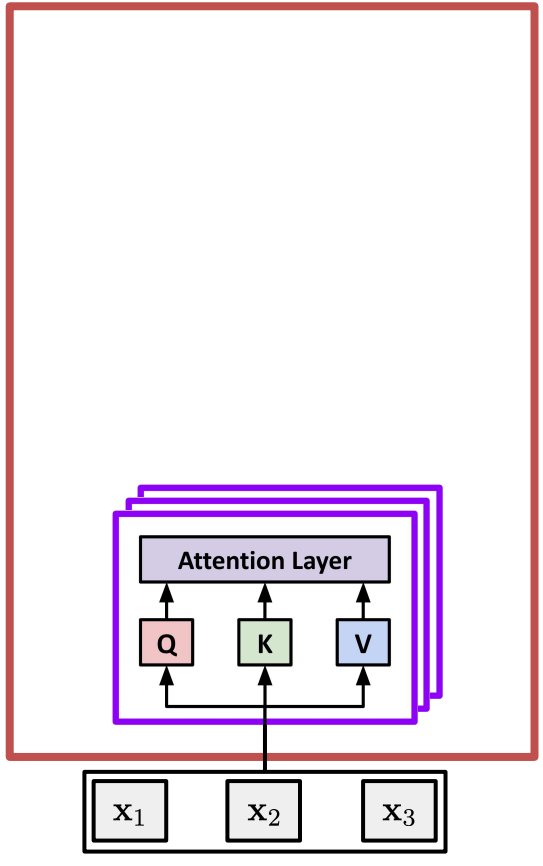
- The transformer block is a building block consisting of:
  - (multi-head) Self-attention
  - Add & Norm
  - Feed-forward
  - Add & Norm

- Input: a sequence, *N* items
- Output: a sequence, *N* items
  - each item being a *d*-dim vector



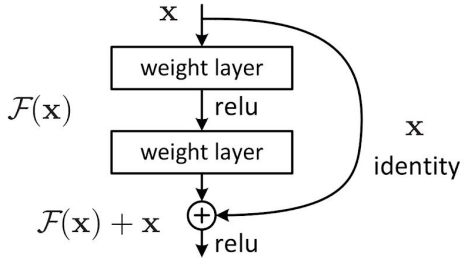
The Transformer block

- **Self-attention**
  - This is the multi-head self-attention block (layer) that we have seen so far



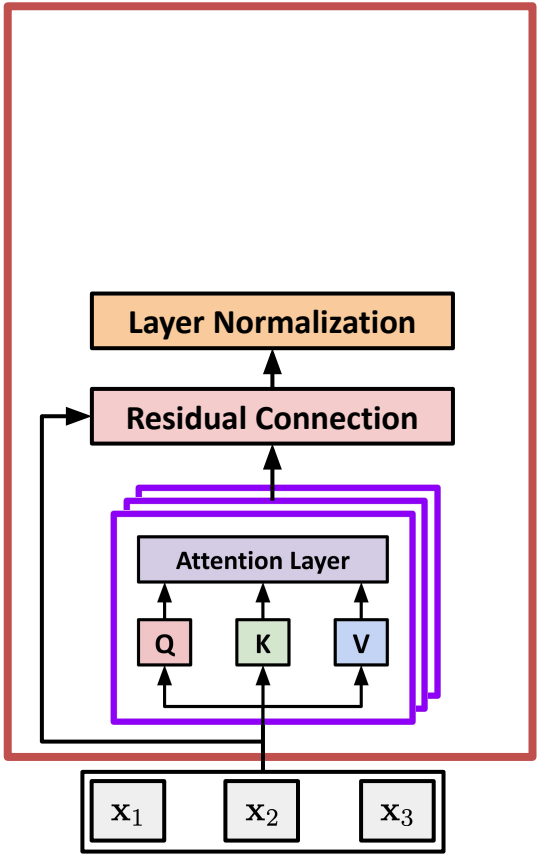
The Transformer block

- Self-attention
- **Add & Norm**
  - Residual Connection (Lec12, p.74)



  - Layer Normalization
    - Similar to Batch Normalization
    - Rescales the data

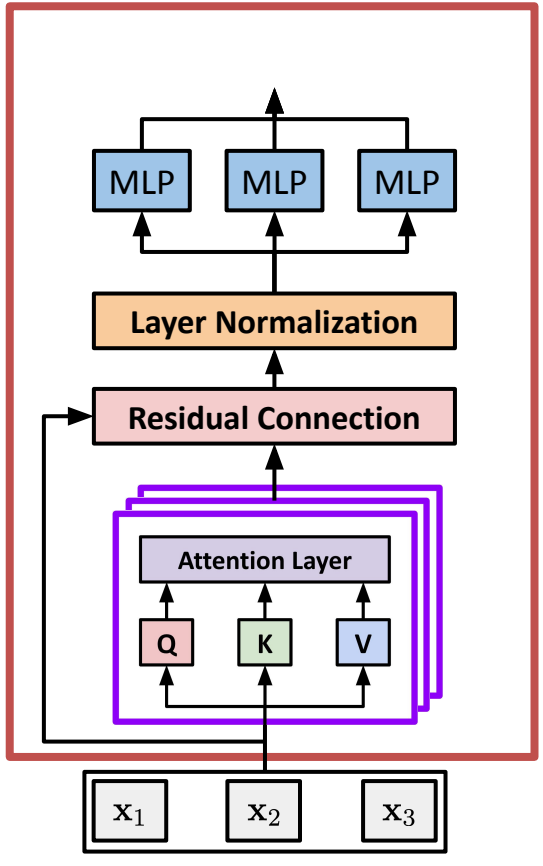
$$\mu^l = \frac{1}{H}\sum_{i=1}^{H} a_i^l \qquad \sigma^l = \sqrt{\frac{1}{H}\sum_{i=1}^{H}\left(a_i^l - \mu^l\right)^2}$$



See [He et al., 2016] (residual) and [Ba et al., 2016] (layer norm)

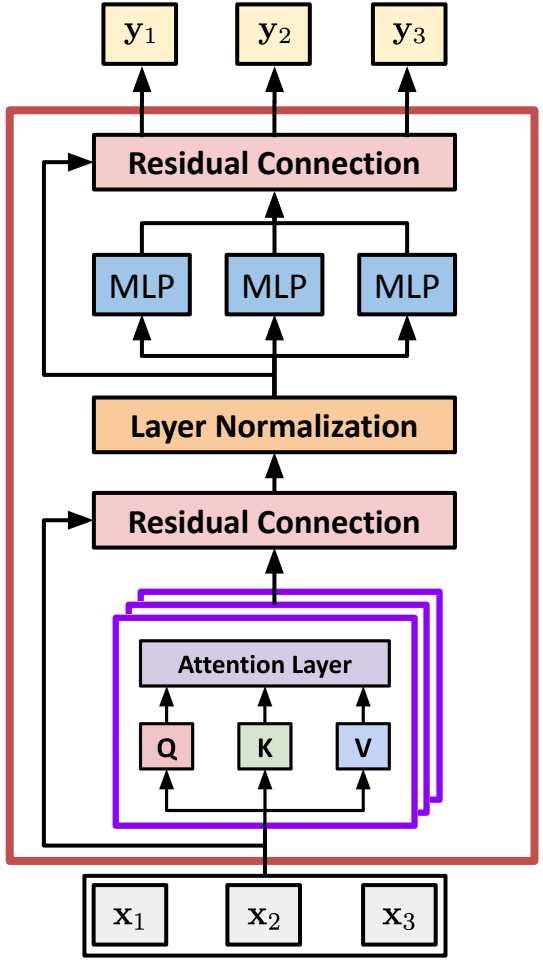
The Transformer block

- Self-attention
- Add & Norm
- **Feed-forward**
  - The same (shared) MLP independently on each vector in the sequence
  - non-linearity!



The Transformer block

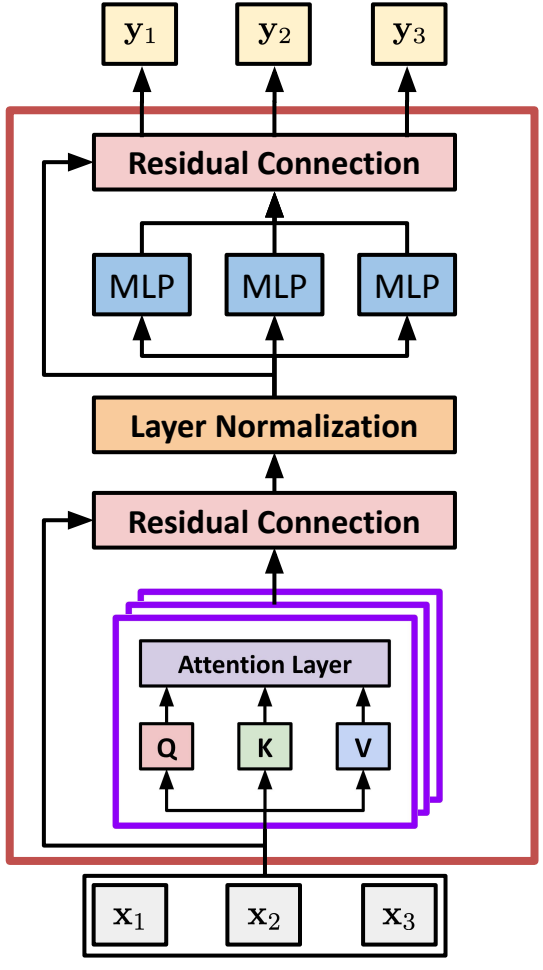
- Self-attention
- Add & Norm
- Feed-forward
- **Add & Norm (again)**
  - Residual Connection
  - Layer Normalization



The Transformer block

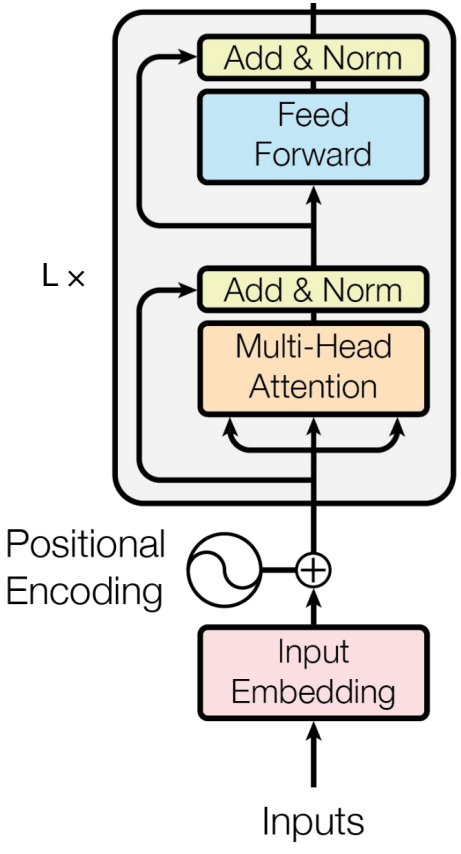
- The transformer block is a building block consisting of:
  - (multi-head) Self-attention
  - Add & Norm
  - Feed-forward
  - Add & Norm

- A Transformer is a sequence of consecutive transformer blocks
  - e.g. 12 blocks, 6 heads, D=512 (Vaswani et al., 2017)



The Transformer Encoder

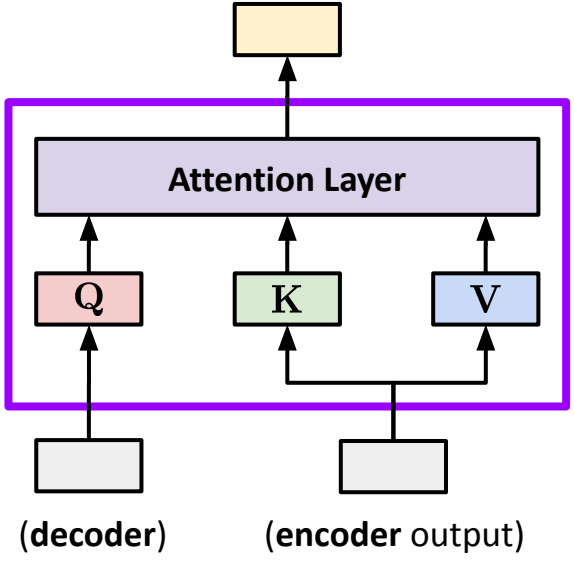
- Inputs: sequence (e.g. a list of words)
  - Use word embedding as usual
- Positional encoding
- Stack the transformer blocks *L* times
- Basically it's a *bidirectional* model;
  a word at time ***t*** can look at future steps!

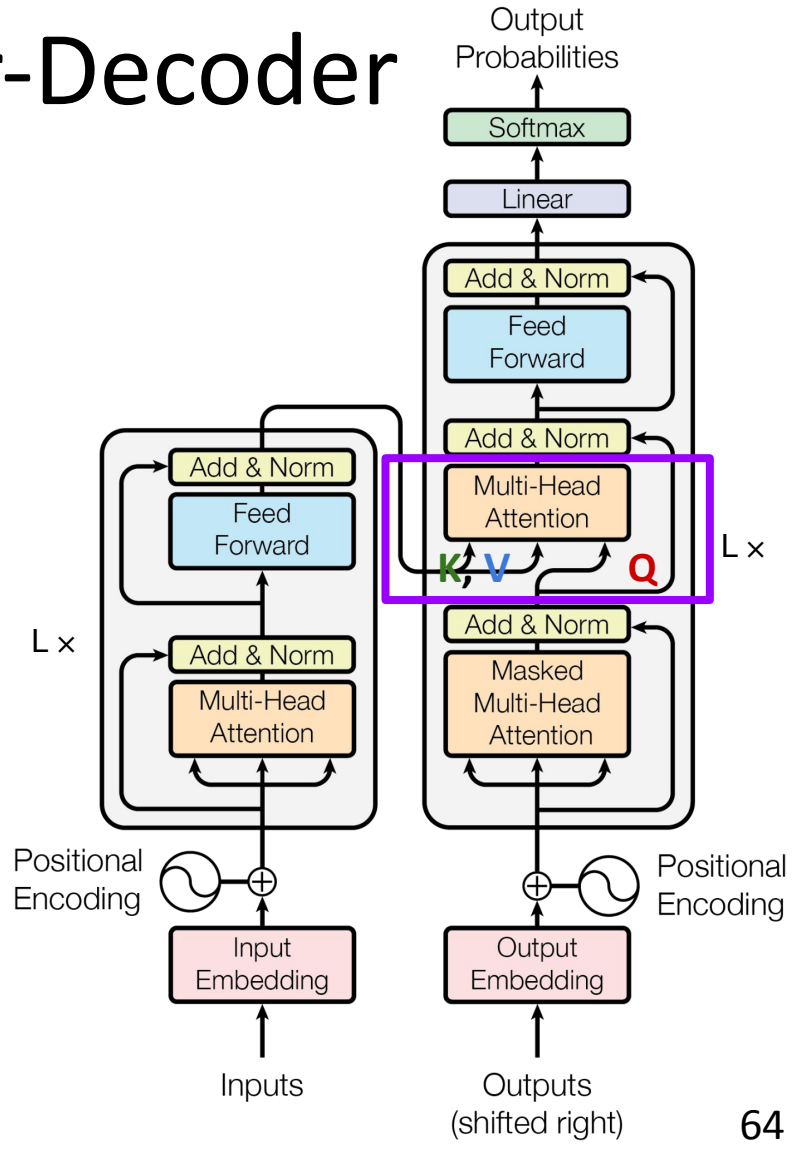


GIF: https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html

The Transformer Encoder-Decoder

- A sequence-to-sequence-like (encoder-decoder) architecture
- Perform **cross-attention** to feed the encoder output



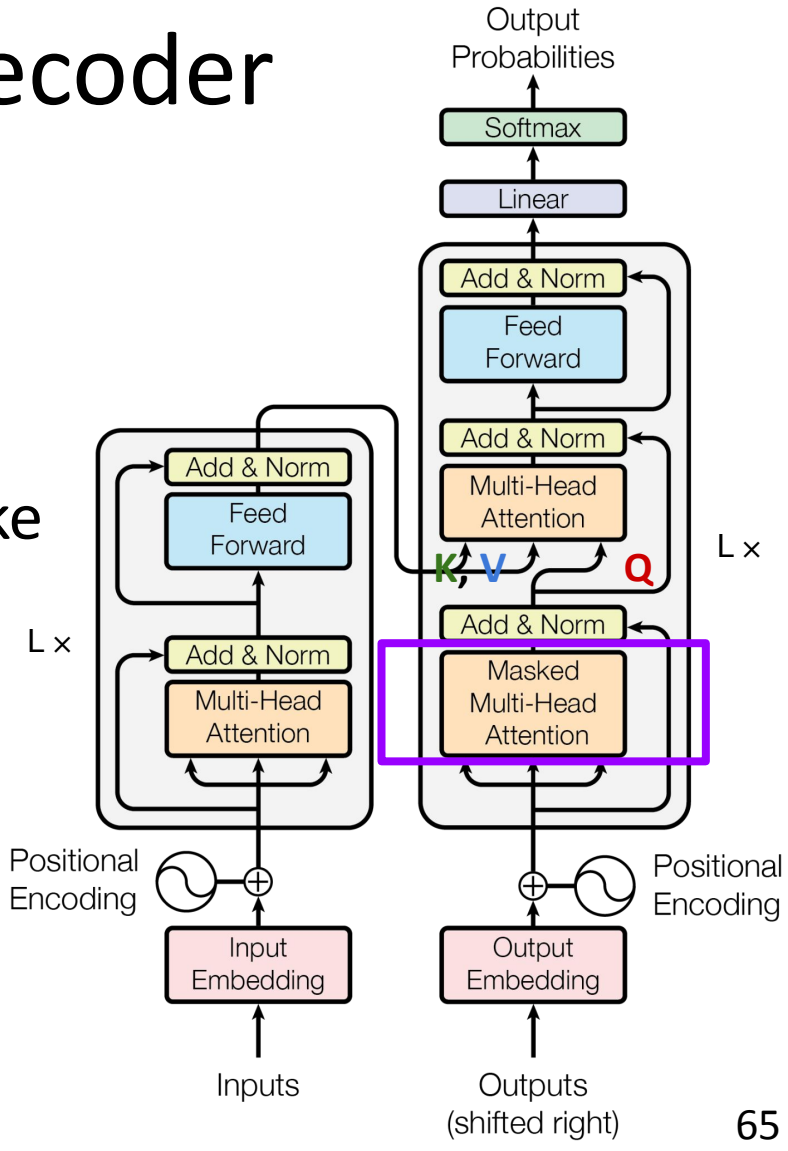


The Transformer Decoder

- Unlike the encoder, masked multi-head self-attention is used
- AKA a ***causal*** attention mask
- When decoding, we need to make sure we do not look ahead while predicting a sequence

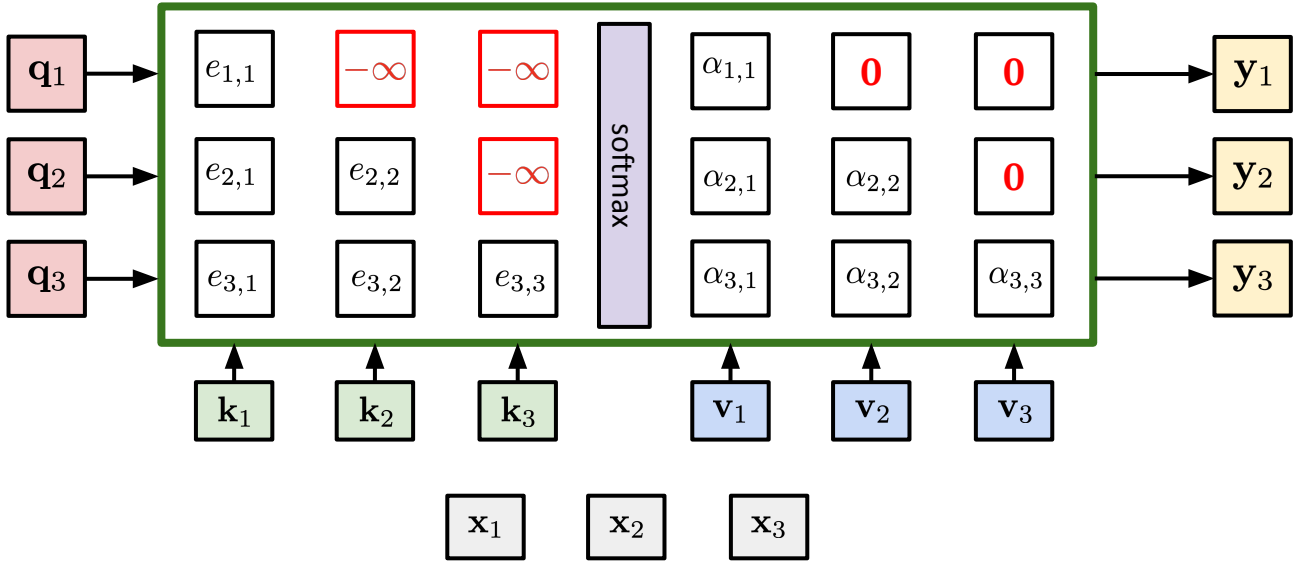
encoder output (sequence)

decoder output (decode one at a time)



Masked Self-Attention

- A simple solution: we mask out attention to future elements by setting the alignment score to –∞ (infinity).
- Example: when *t*=2, $\mathbf{y}_2$ cannot look at $\mathbf{x}_3$ $(\text{or } \mathbf{v}_3)$.

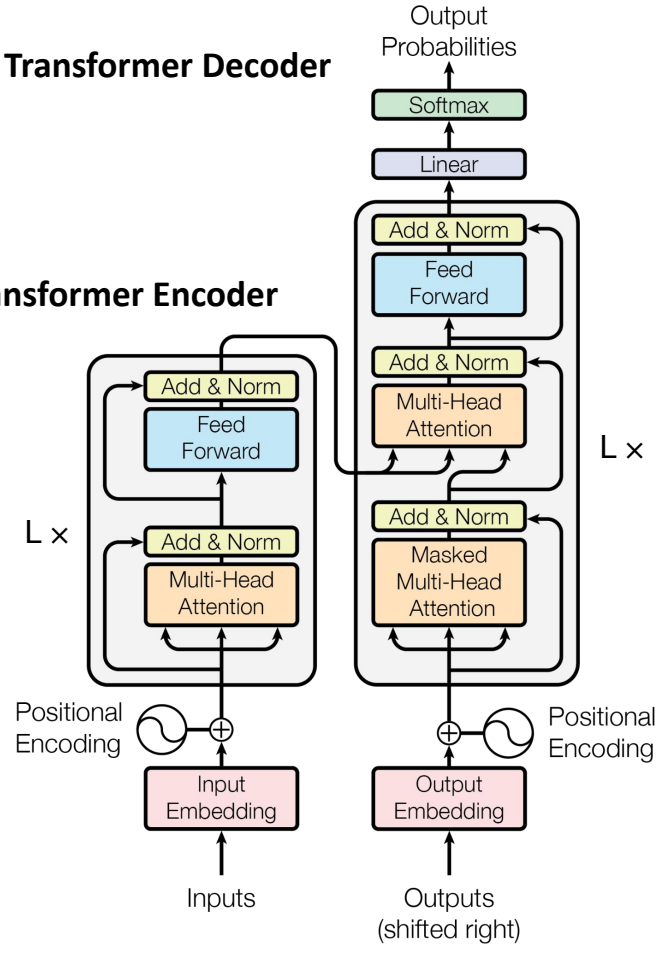


The Transformer Network

- "Attention is all you Need" (Vaswani et al., 2017)

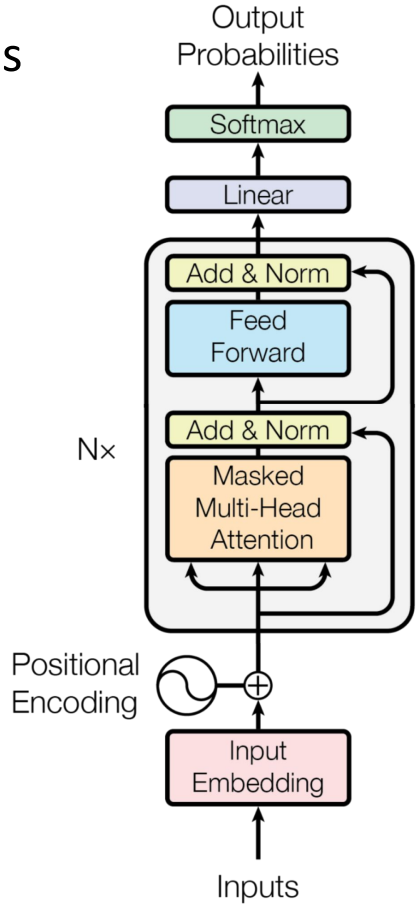
  Transforms a sequence into a sequence!

- An encoder-decoder structure made only of **attentions**
  - i.e., no convolutions and no recurrent units!
  - exhibits better ability to handle long sequences than RNNs



Decoder-only Transformers

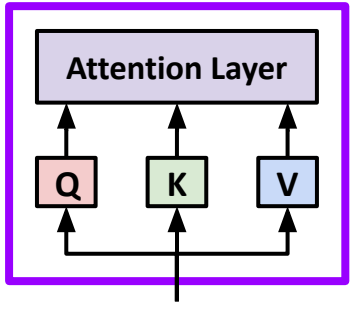
- for Language Models (e.g., GPT)

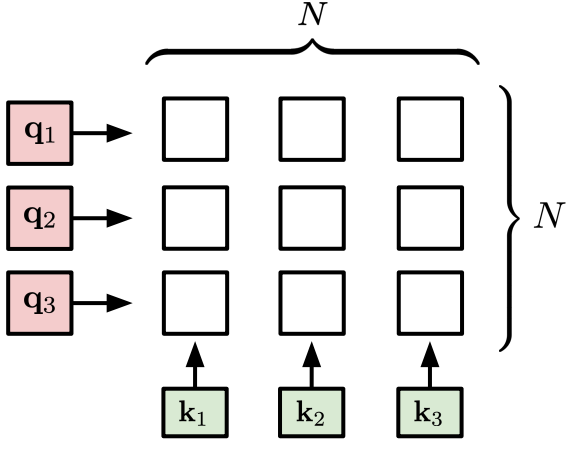


- No encoders; no cross-attention
- Just stack the transformer blocks!

Time Complexity of Self-Attention

- How expensive is a self-attention layer?
  - A quadratic time complexity: $\mathcal{O}(N^2 d)$
    - N is the number of tokens in the sequence
  - However, computations can be parallelized in GPUs
- Notably, there are some successor works that try to improve the time complexity, e.g. Linear Transformers, Perceiver I/O.

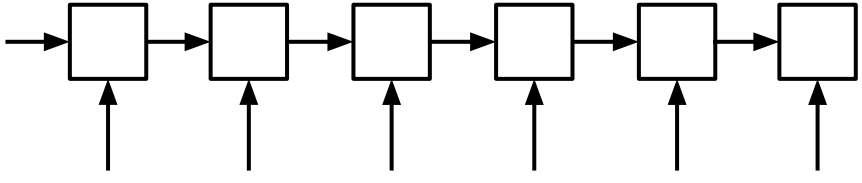


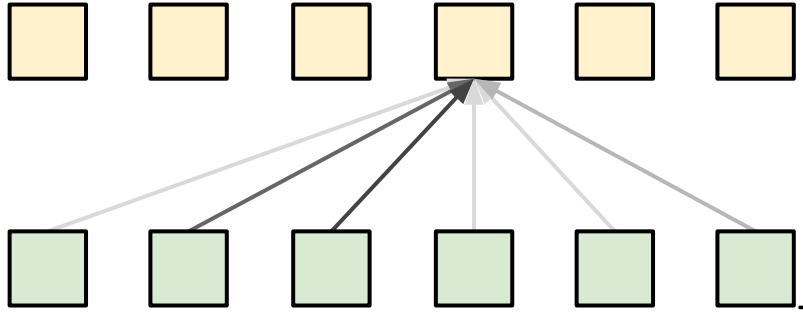


See (Tay et al., 2022)
"Efficient Transformers: A Survey"

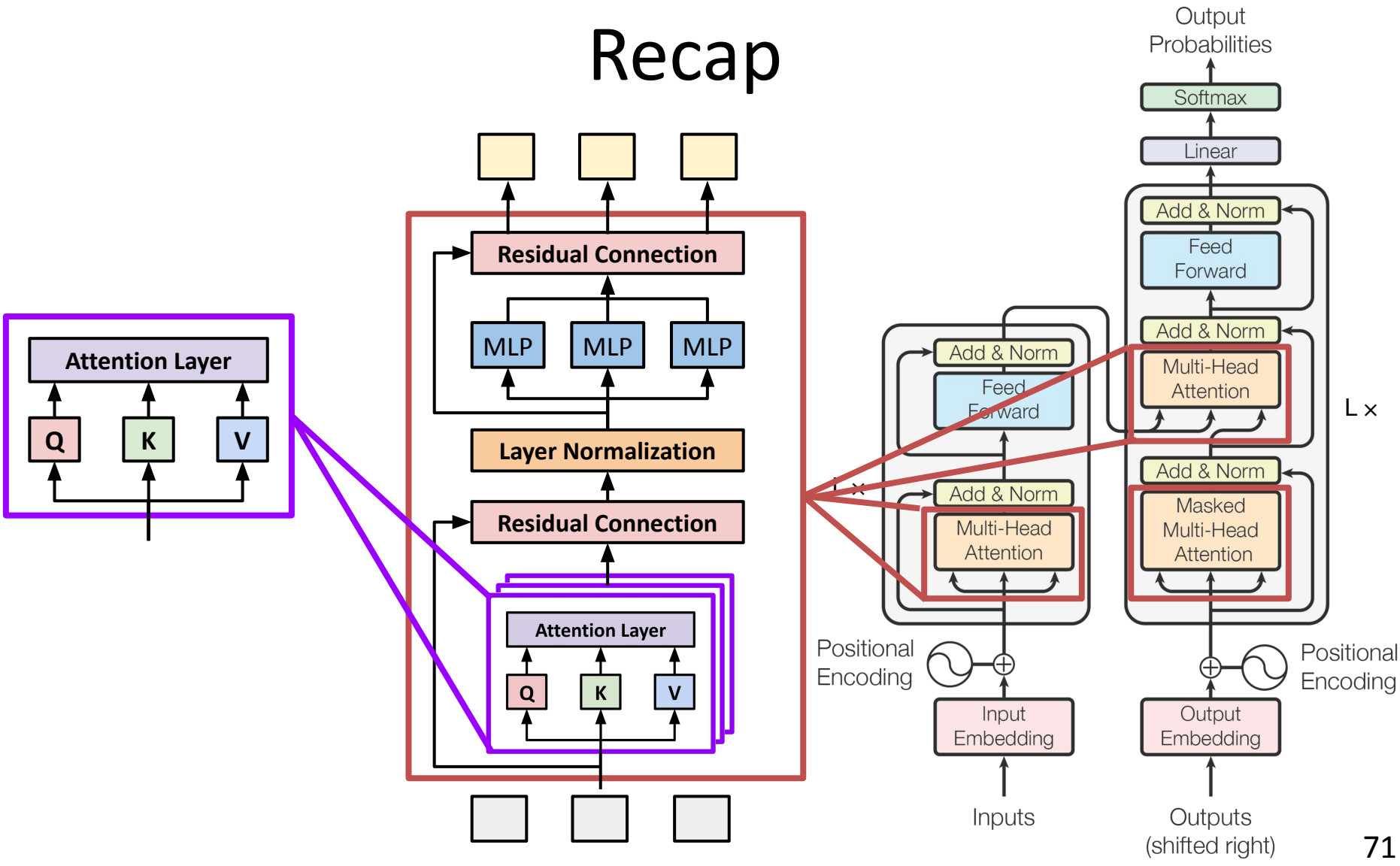
RNN v.s. Self-Attention

- RNN:
  - Serial dependency: O(sequence length)
- Transformers:
  - Maximum path length is O(1); with a single self-attention layer, any word can look at all other words in the sequence
  - O(1) sequential operations; many computations can be parallelized

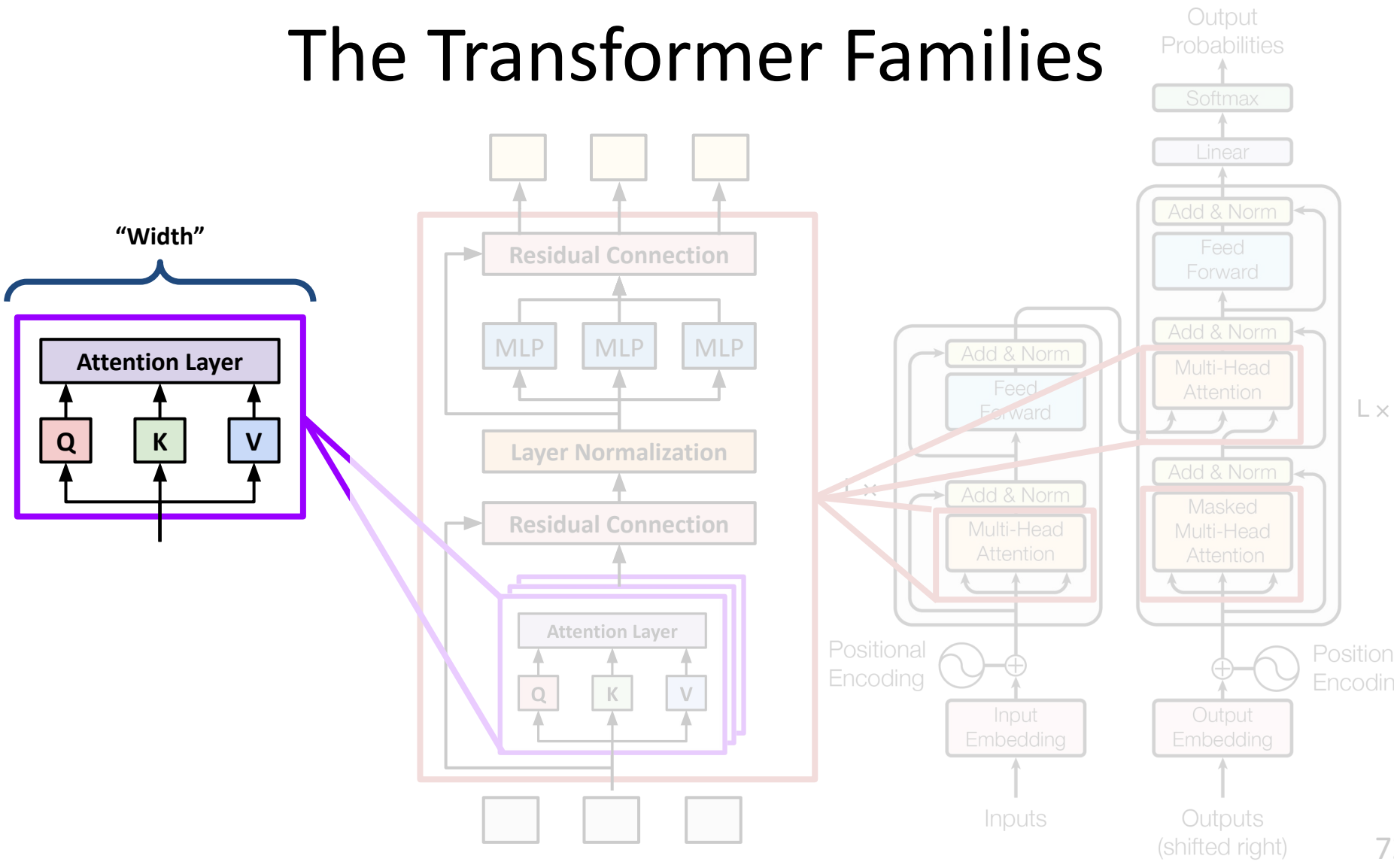




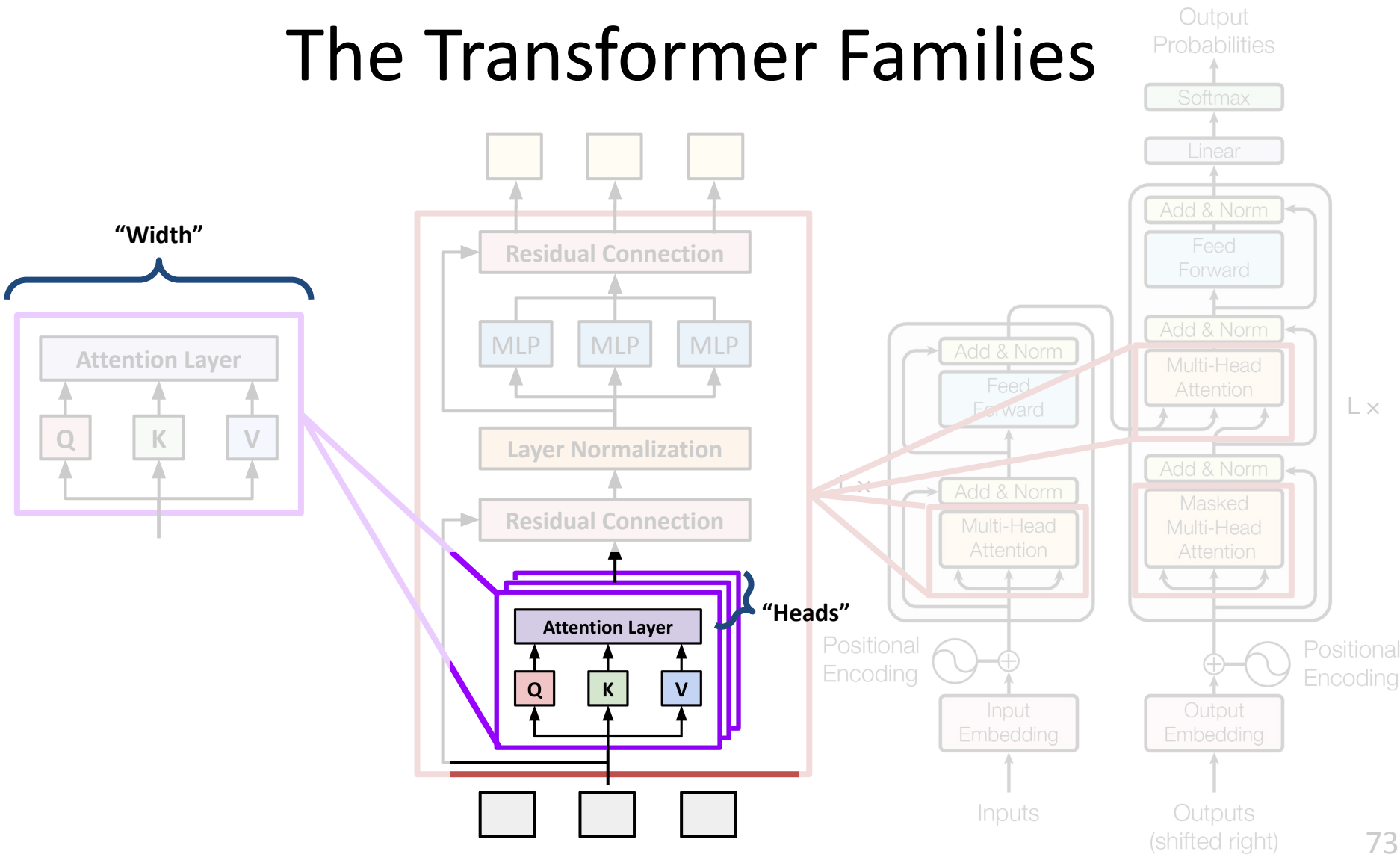
Recap


Attention Layer
Q
K
V
Residual Connection
MLP
MLP
MLP
Layer Normalization
Residual Connection
Attention Layer
Q
K
V
Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
L ×
Positional Encoding
Input Embedding
Inputs
Positional Encoding
Output Embedding
Outputs (shifted right)

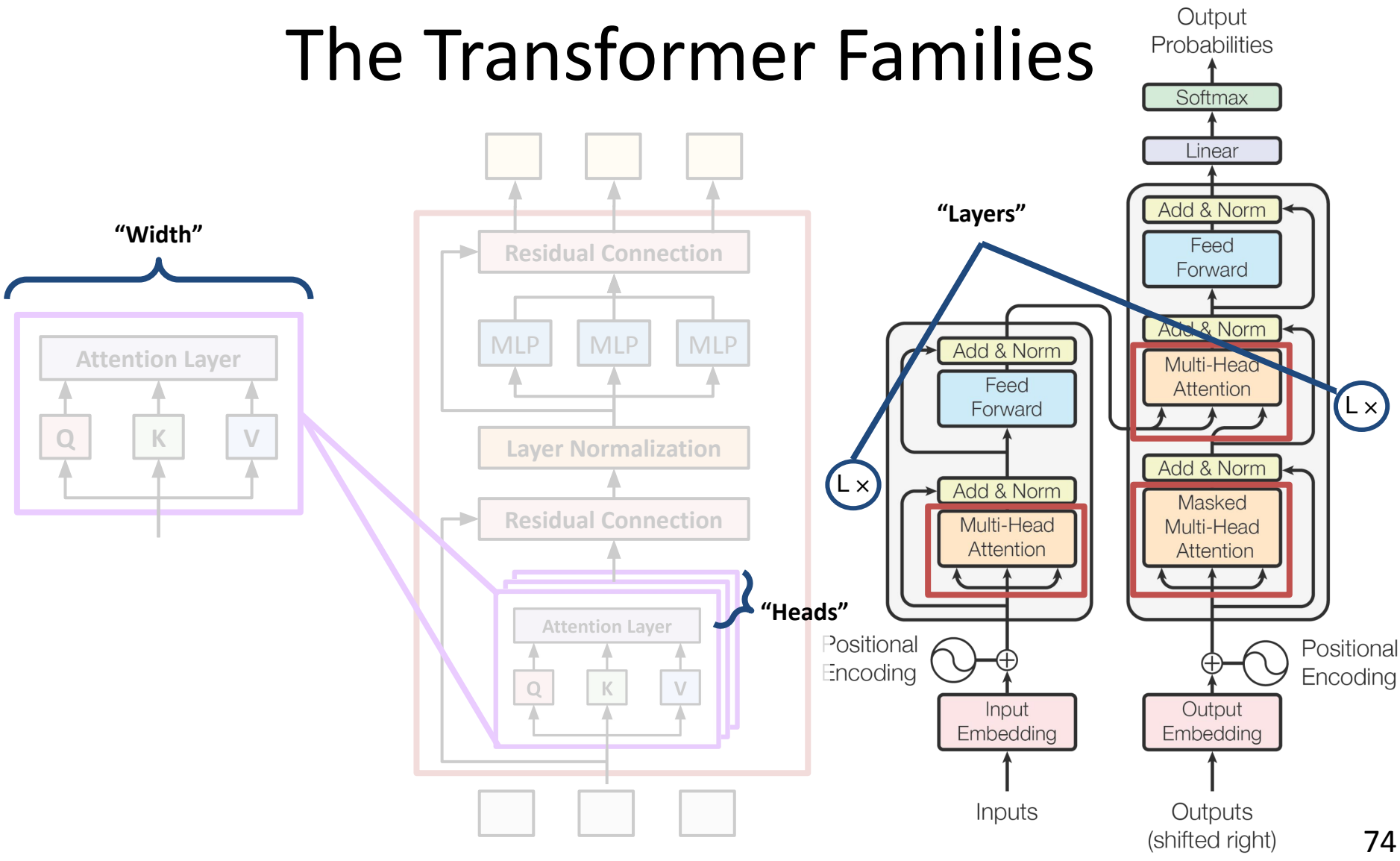
The Transformer Families



The Transformer Families



The Transformer Families



Large Language Models

Model	Layers	Width	Heads	Params	Data	Training
Transformer-Base	12	512	8	65M		8x P100 (12 hours)
Transformer-Large	12	1024	16	213M		8x P100 (3.5 days)
BERT-Base	12	768	12	110M	13 GB	
BERT-Large	24	1024	16	340M	13 GB	
XLNet-Large	24	1024	16	~340M	126 GB	512x TPU-v3 (2.5 days)
RoBERTa	24	1024	16	355M	160 GB	1024x V100 GPU (1 day)
GPT-2	48	1600	?	1.5B	40 GB	
Megatron-LM	72	3072	32	8.3B	174 GB	512x V100 GPU (9 days)
Turing-NLG	78	4256	28	17B	?	256x V100 GPU
GPT-3	96	12,288	96	175B	694GB	?
Gopher	80	16,384	128	280B	10.55 TB	4096x TPUv3 (38 days)

$3,768,320 on Google Cloud (estimated price)

Large Language Models

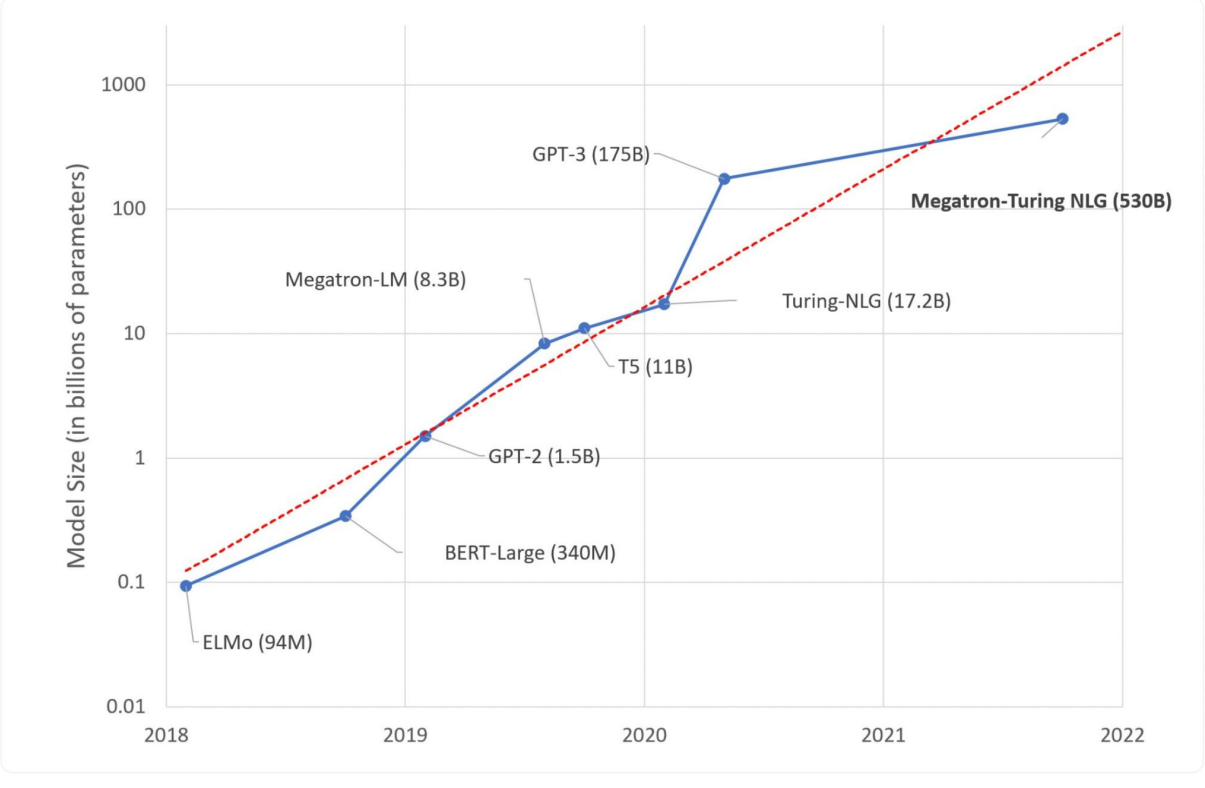
Model	Layers	Width	Heads	Params	Context Window	Data	Training
Transformer-Base	12	512	8	65M	-	-	8x P100 (12 hours)
Transformer-Large	12	1024	16	213M	-	-	8x P100 (3.5 days)
BERT-Base	12	768	12	110M	512 tokens	13 GB	-
BERT-Large	24	1024	16	340M	512 tokens	13 GB	-
XLNet-Large	24	1024	16	~340M	512 tokens	126 GB	512x TPU-v3 (2.5 days)
RoBERTa	24	1024	16	355M	512 tokens	160 GB	1024x V100 GPU (1 day)
GPT-2	48	1600	-	1.5B	1024 tokens	40 GB	-
Megatron-LM	72	3072	32	8.3B	-	174 GB	512x V100 GPU (9 days)
Turing-NLG	78	4256	28	17B	512 tokens	-	256x V100 GPU
GPT-3	96	12,288	96	175B	2048 tokens	694 GB	-
Gopher	80	16,384	128	280B	2048 tokens	10.55 TB	4096x TPUv3 (38 days)
GPT-4	120*	-	-	1.75T*	8K/32K tokens	-	-
LLaMA-2-70B	80	8192	64	70B	4096 tokens	2T tokens	1.7M A100 GPU Hours
LLaMA-3-405B	126	16,384	128	405B	128K tokens	15T tokens	30.84M A100 GPU Hours

* GPT-4 size is estimated. There is no information publically available for concrete number of GPT-4 model size.

$60M-$120M estimated training cost

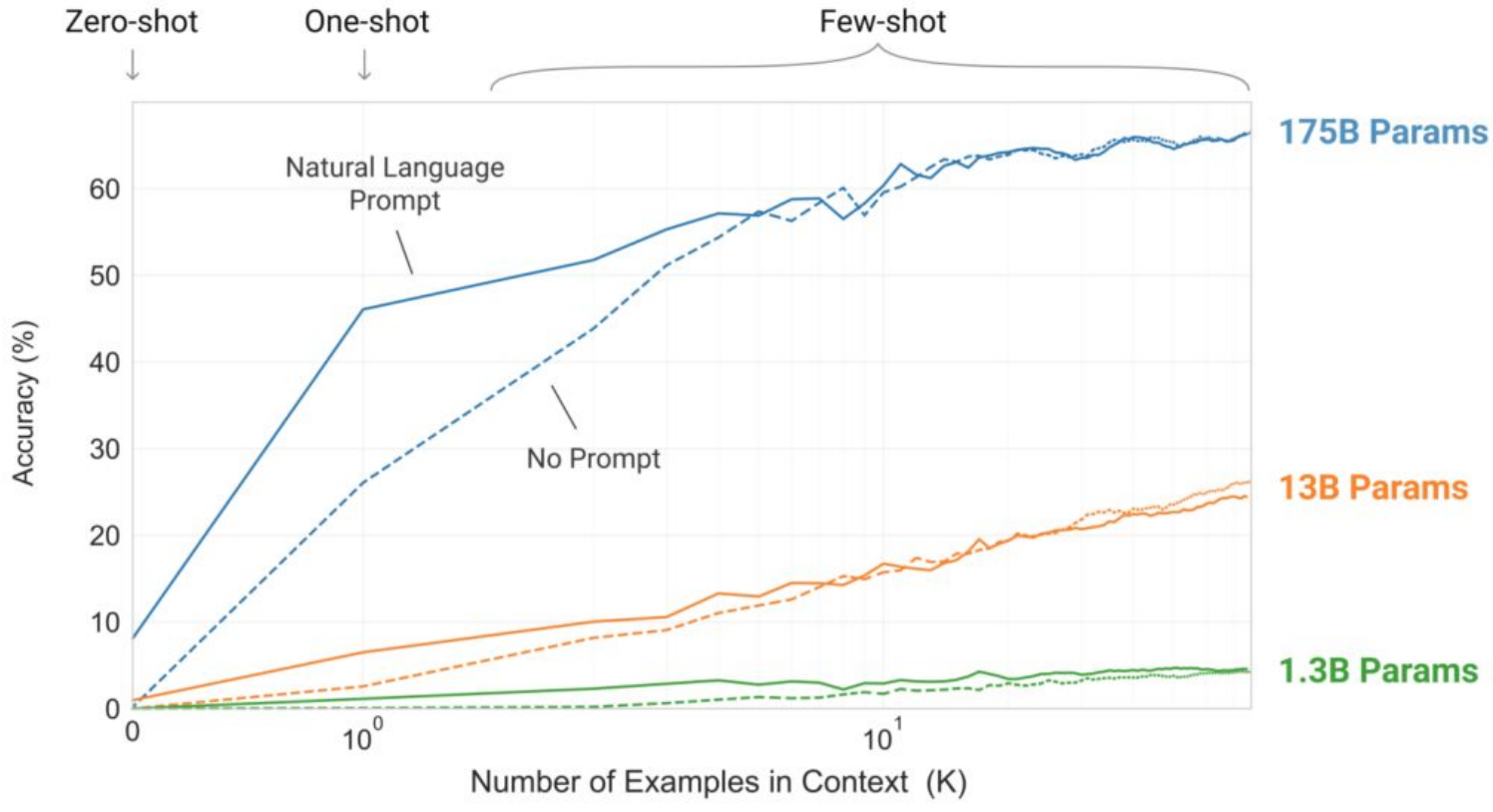
Large Language Models

Since 2017, Transformers have dominated many ML domains (NLP, vision)!

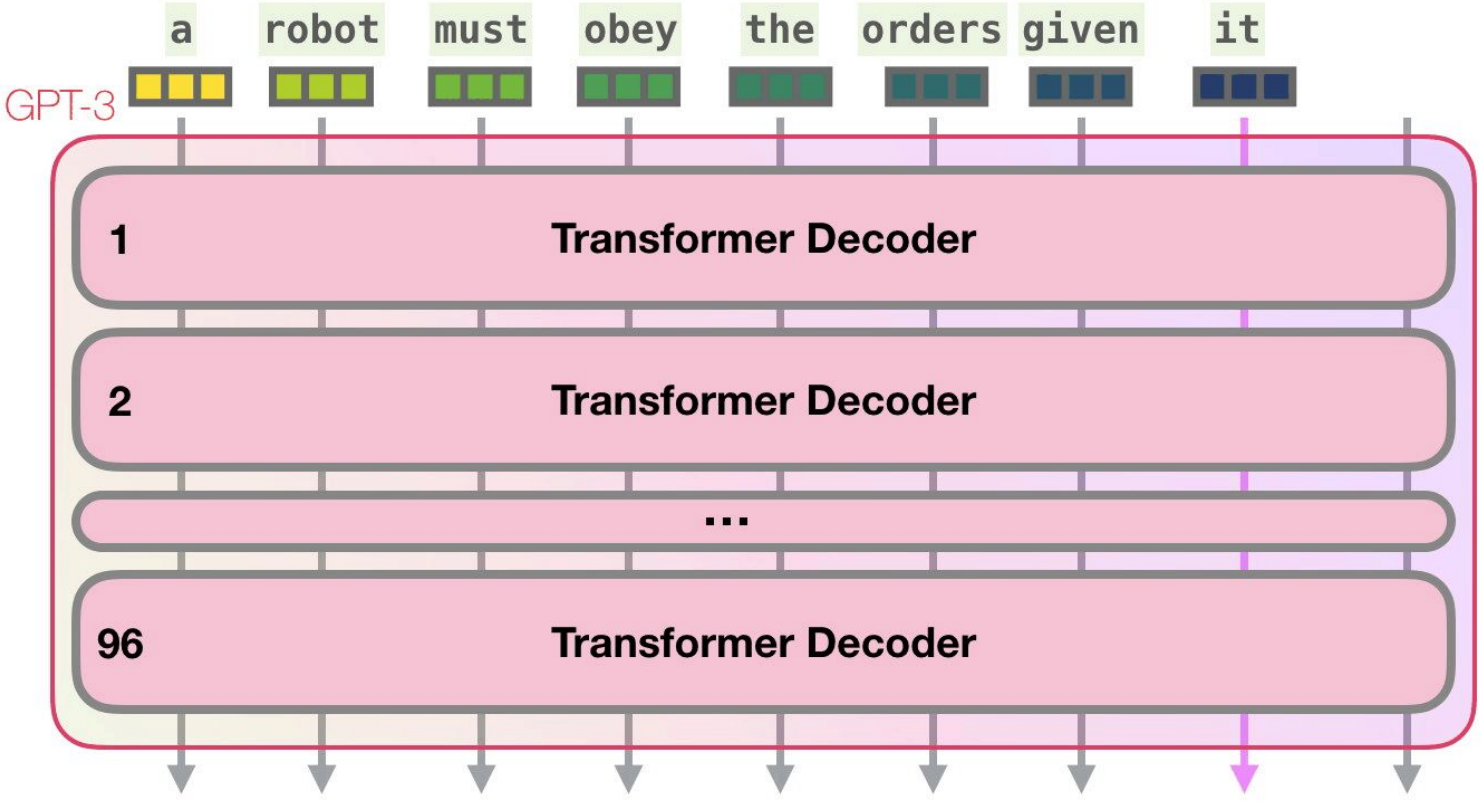


https://huggingface.co/blog/large-language-models (2021)

Large Language Models: GPT-3

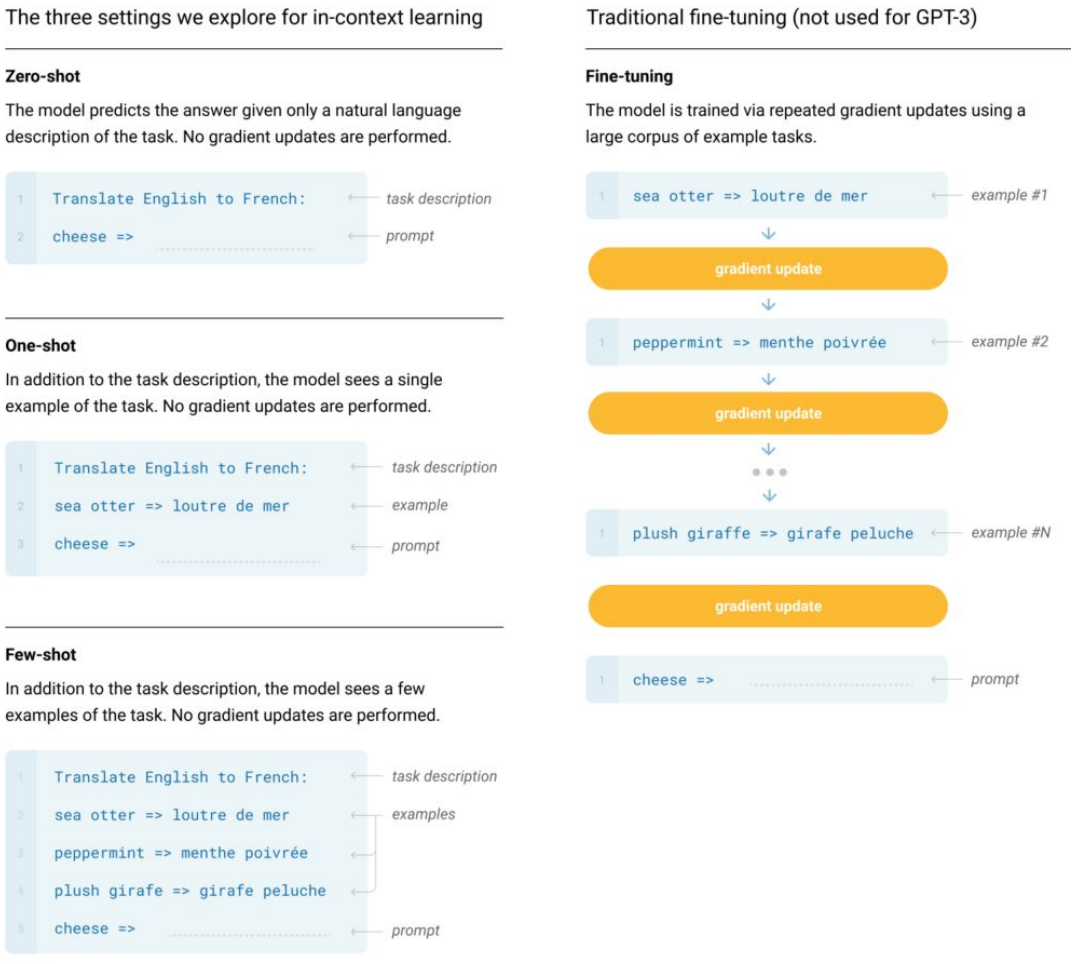


Large Language Models: GPT-3



https://jalammar.github.io/how-gpt3-works-visualizations-animations/

Large Language Models: GPT-3



ChatGPT (Built on GPT-3, GPT-4, etc.)

HO

I'm teaching machine learning, and I'm making a powerpoint lecture about attention and transformers. Can you make a powerpoint slide with a conclusion to the lecture? Also include why students should start homework 4 early.

Sure, here's a PowerPoint slide with a conclusion to the lecture and a note about starting homework early:

[Title: Conclusion]

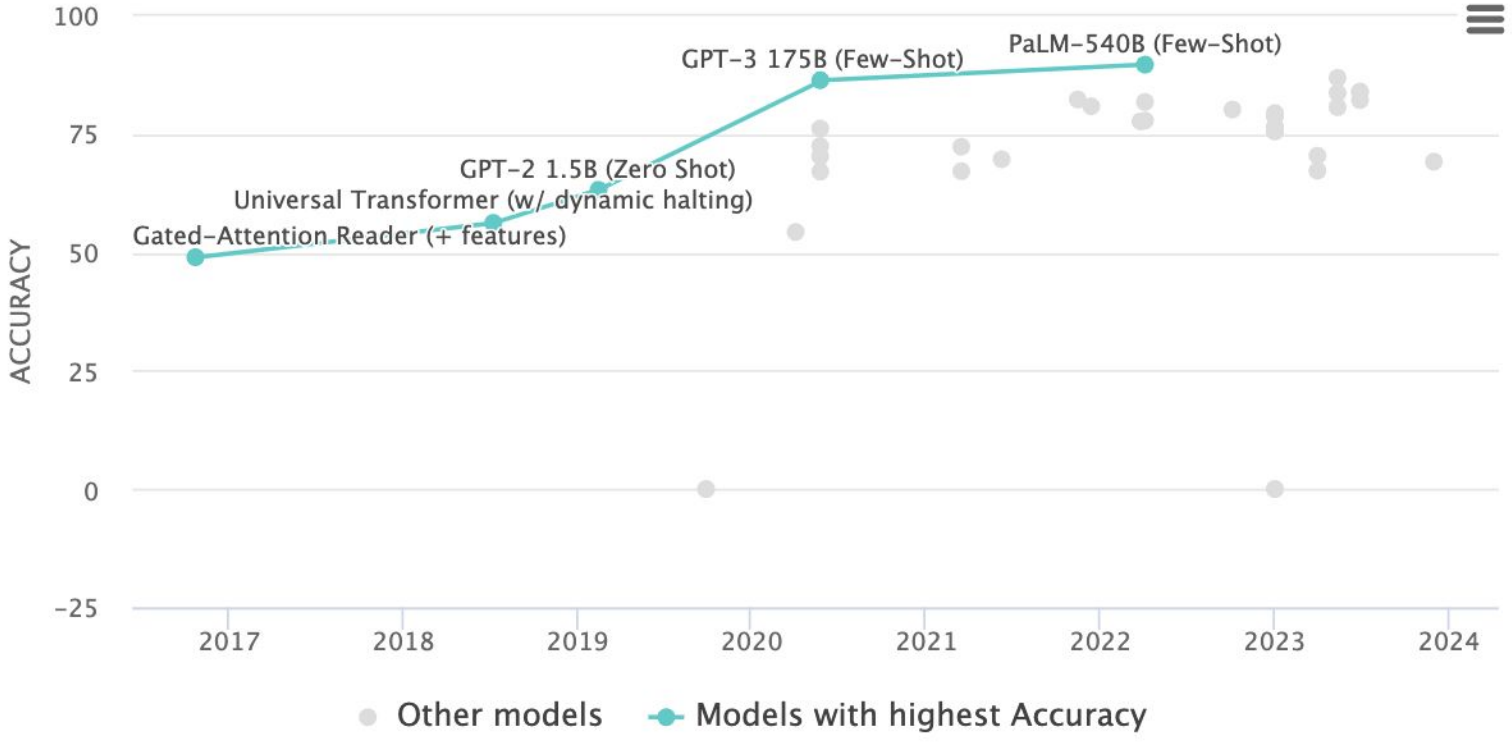
[Image: A graphic of a transformer with attention heads]

In conclusion, attention and transformers have revolutionized the field of natural language processing and have led to breakthroughs in other areas of machine learning as well. By

https://chat.openai.com/

Large Language Models

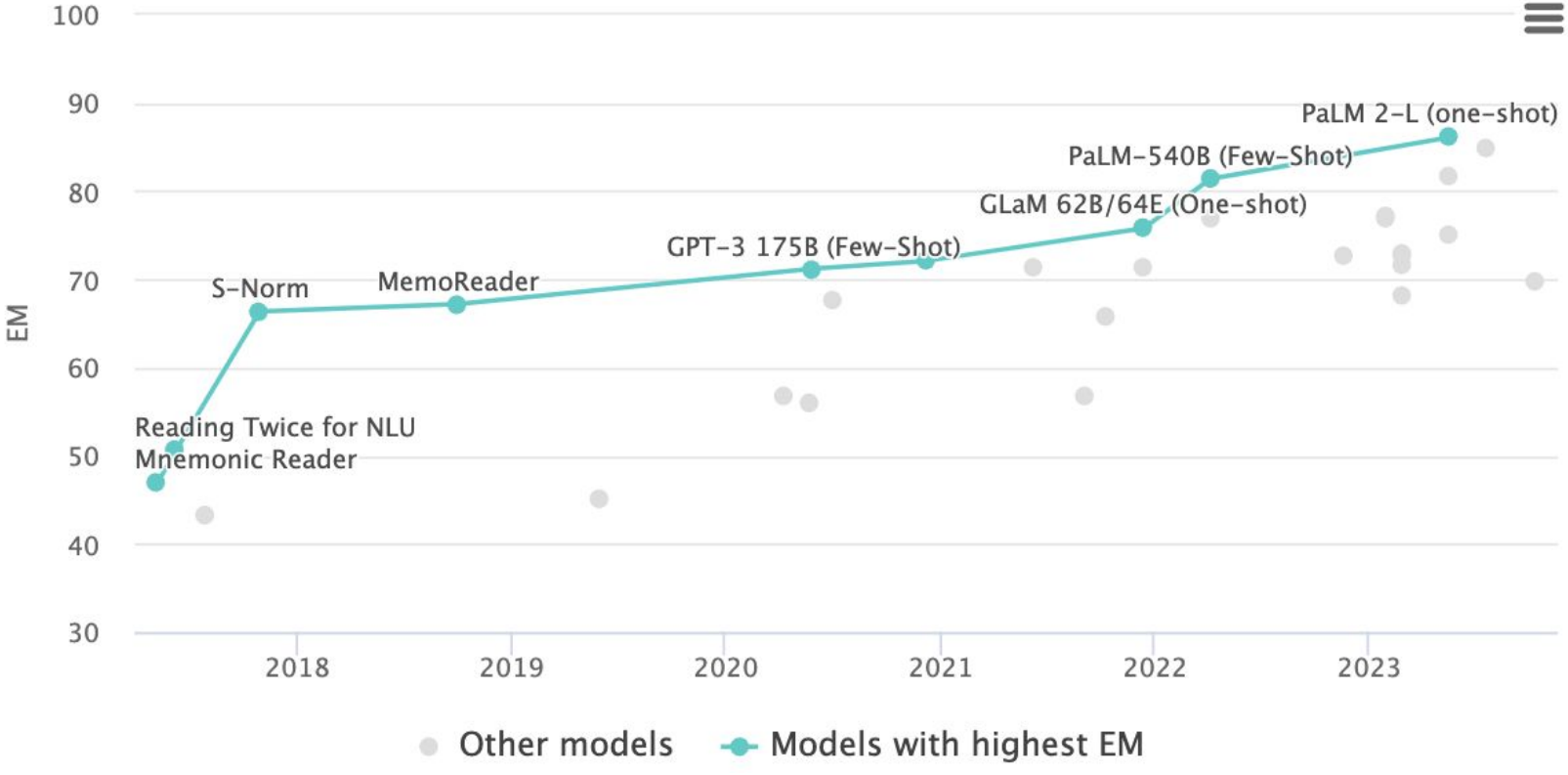
Language Modelling on LAMBADA (sentence prediction)



https://paperswithcode.com/sota/language-modelling-on-lambada

Large Language Models

Question Answering on TriviaQA (trivia questions)



https://paperswithcode.com/sota/question-answering-on-triviaqa

LoRA: Low Rank Adaptation[1]

- GPT-3 has a *175 Billion* parameters
  - Finetuning the entire model is expensive and challenging

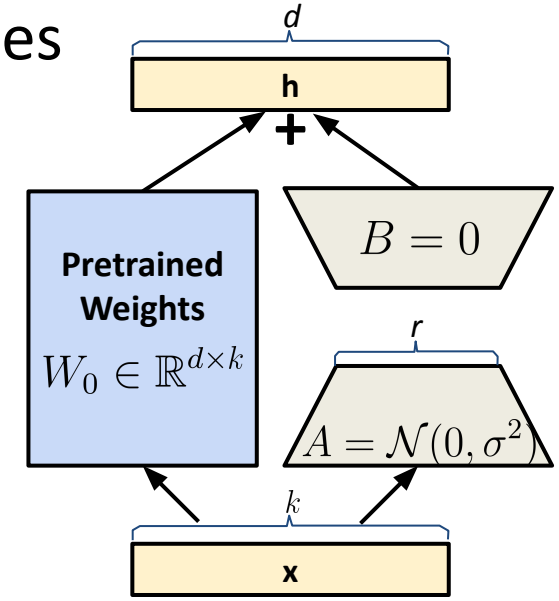
- Low-rank-parameterized update matrices
  - W_0 is a pretrained weight matrix

$$W_0 + \Delta W = W_0 + BA$$

$$\text{where } W_0 \in \mathbb{R}^{d \times k}, B \in \mathbb{R}^{d \times r},$$

$$A \in \mathbb{R}^{r \times k}, r \ll \min(d, k)$$

  - Only *A* and *B* are *trainable* parameters
  - Often LoRA is applied to the weight matrices in the self-attention module

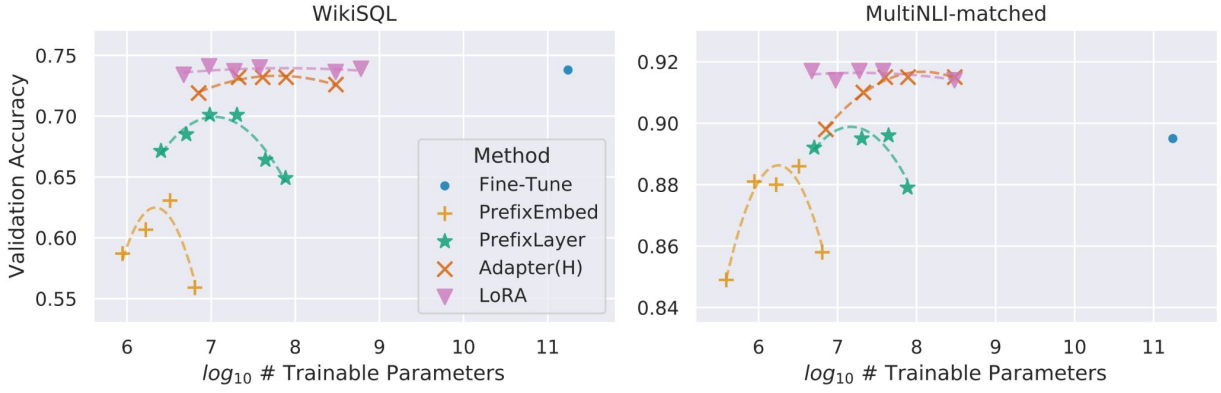


Note: k,d are arbitrary symbols for input/output dimensions

[1] Hu et al., LoRA: Low-Rank Adaptation of Large Language Models, in ICLR 2022
[2] Aghajanyan et al., Intrinsic Dimensionality Explains the Effectiveness of Language Model Fine-Tuning, in ACL 2021

Scaling up to GPT-3 175B

Model&Method	# Trainable Parameters	WikiSQL Acc. (%)	MNLI-m Acc. (%)	SAMSum R1/R2/RL
GPT-3 (FT)	175,255.8M	**73.8**	89.5	52.0/28.0/44.5
GPT-3 (BitFit)	14.2M	71.3	91.0	51.3/27.4/43.5
GPT-3 (PreEmbed)	3.2M	63.1	88.6	48.3/24.2/40.5
GPT-3 (PreLayer)	20.2M	70.1	89.5	50.8/27.3/43.5
GPT-3 ($\text{Adapter}^{\text{H}}$)	7.1M	71.9	89.8	53.0/28.9/44.8
GPT-3 ($\text{Adapter}^{\text{H}}$)	40.1M	73.2	**91.5**	53.2/29.0/45.1
GPT-3 (LoRA)	4.7M	73.4	**91.7**	**53.8/29.8/45.9**
GPT-3 (LoRA)	37.7M	**74.0**	**91.6**	53.4/29.2/45.1



LoRA matches or exceeds the finetuning baseline on all three datasets

LoRA exhibits better scalability and task performance.

[1] Hu et al., LoRA: Low-Rank Adaptation of Large Language Models, in ICLR 2022
[2] Aghajanyan et al., Intrinsic Dimensionality Explains the Effectiveness of Language Model Fine-Tuning, in ACL 2021

Slide Credit: Diyi Yang

Understanding Low Rank Adaptation

Q: Which weight matrices in Transformers should we apply LoRA to?

A: Adapting both W_q and W_v gives the biggest performance boost.

	# of Trainable Parameters = 18M						
Weight Type	W_q	W_k	W_v	W_o	W_q, W_k	W_q, W_v	W_q, W_k, W_v, W_o
Rank r	8	8	8	8	4	4	2
WikiSQL (±0.5%)	70.4	70.0	73.0	73.2	71.4	**73.7**	**73.7**
MultiNLI (±0.1%)	91.0	90.8	91.0	91.3	91.3	91.3	**91.7**

Q: What is the optimal rank *r* for LoRA?

A: LoRA already performs competitively with a very small *r*

	Weight Type	$r = 1$	$r = 2$	$r = 4$	$r = 8$	$r = 64$
WikiSQL(±0.5%)	W_q	68.8	69.6	70.5	70.4	70.0
	W_q, W_v	73.4	73.3	73.7	73.8	73.5
	W_q, W_k, W_v, W_o	74.1	73.7	74.0	74.0	73.9
MultiNLI (±0.1%)	W_q	90.7	90.9	91.1	90.7	90.7
	W_q, W_v	91.3	91.4	91.3	91.6	91.4
	W_q, W_k, W_v, W_o	91.2	91.7	91.7	91.5	91.4

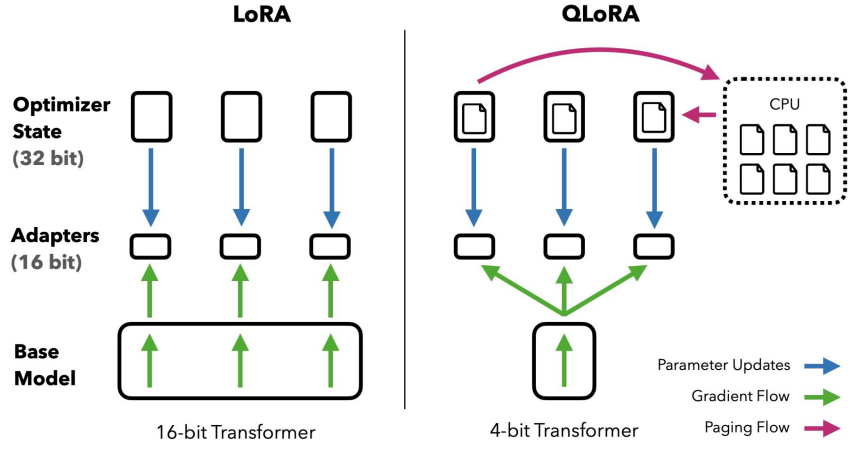
[1] Hu et al., LoRA: Low-Rank Adaptation of Large Language Models, in ICLR 2022
[2] Aghajanyan et al., Intrinsic Dimensionality Explains the Effectiveness of Language Model Fine-Tuning, in ACL 2021

Slide Credit: Diyi Yang

From LoRA[1] to QLoRA[2]

- QLoRA[2] improves over LoRA[1] by quantizing the transformer model to 4-bit precision and using paged optimizer to handle memory spikes

- cf) 4-bit NormalFloat (NF4) is a new data type that is information theoretically optimal for normally distributed weights

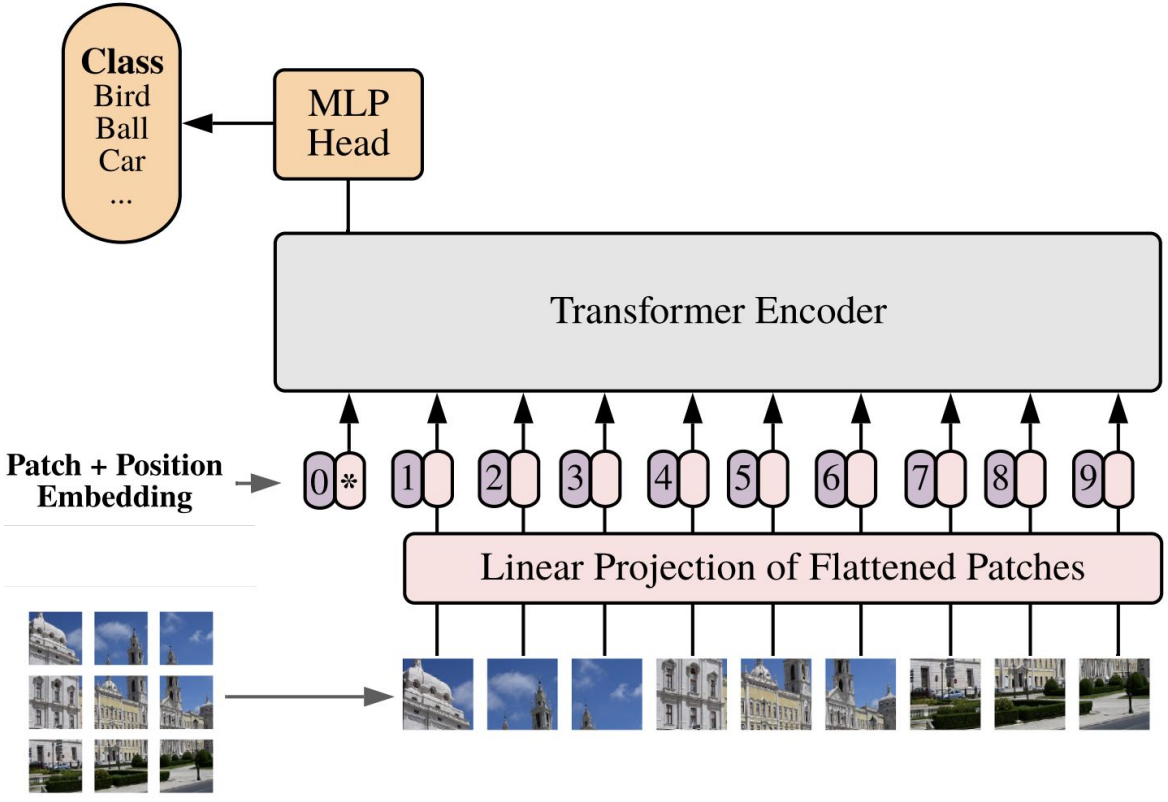


[1] Hu et al., LoRA: Low-Rank Adaptation of Large Language Models, in ICLR 2022
[2] Dettmers et al., QLoRA: Efficient Finetuning of Quantized LLMs, in NeurIPS 2023

Slide Credit: Diyi Yang

Vision Transformers (ViT)

Idea: Apply transformer directly to image patches (3x16x16)



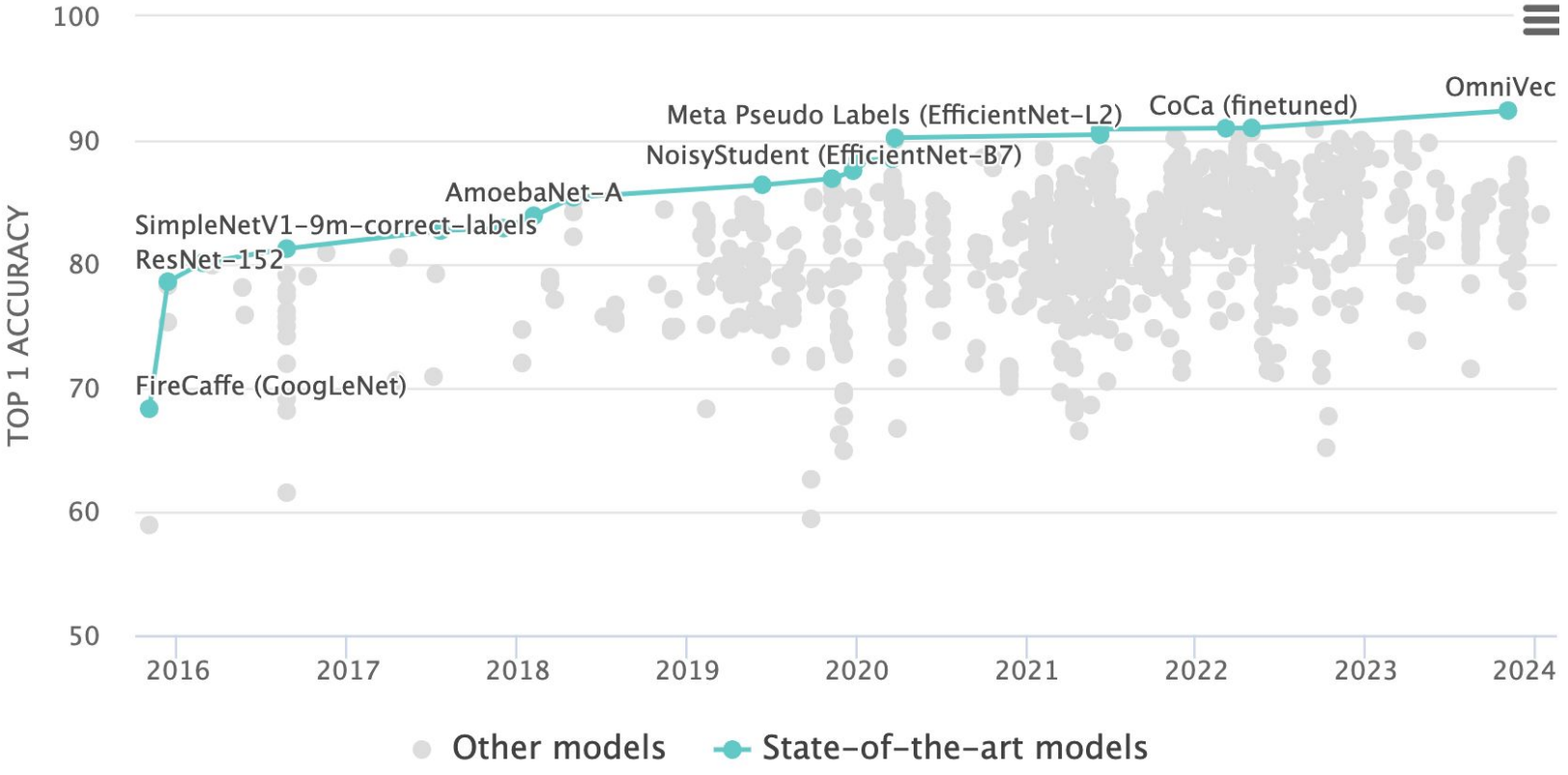
Exact same as NLP Transformer!

In practice: take 224x224 input image, divide into 14x14 grid of 16x16 pixel patches (or 16x16 grid of 14x14 patches)

[1] Cordonnier et al., On the Relationship between Self-Attention and Convolutional Layers, in ICLR 2020
[2] Dosovitskiy et al., An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale, in ICLR 2021

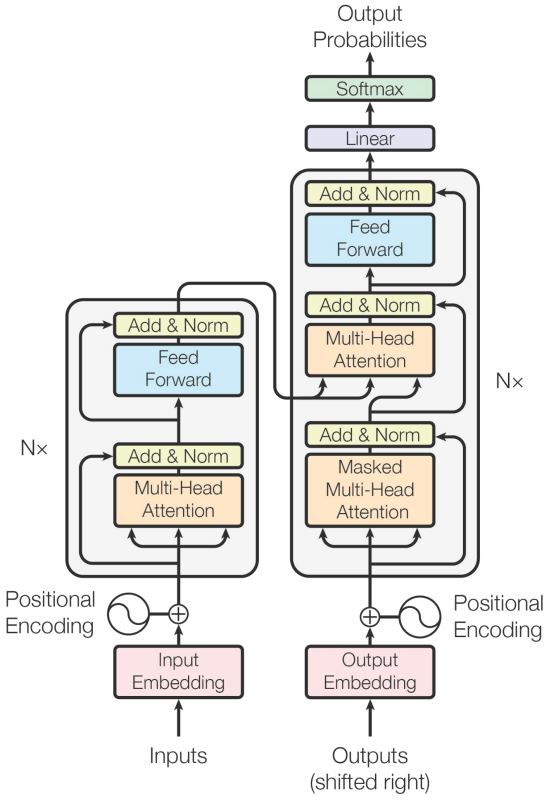
Vision Transformers

Image Classification on ImageNet



https://paperswithcode.com/sota/image-classification-on-imagenet

Summary



- Transformers, in particular, have become the state-of-the-art in many NLP tasks and have been shown to outperform traditional RNNs and CNNs.
- By allowing models to selectively focus on important parts of the input, attention has enabled more effective and efficient learning.
  - multi-head attention allows the model to attend to different parts of the input simultaneously.
- Attention and transformers have revolutionized the field of natural language processing and have led to breakthroughs in many areas of machine learning.