Variants of Recurrent Neural Network

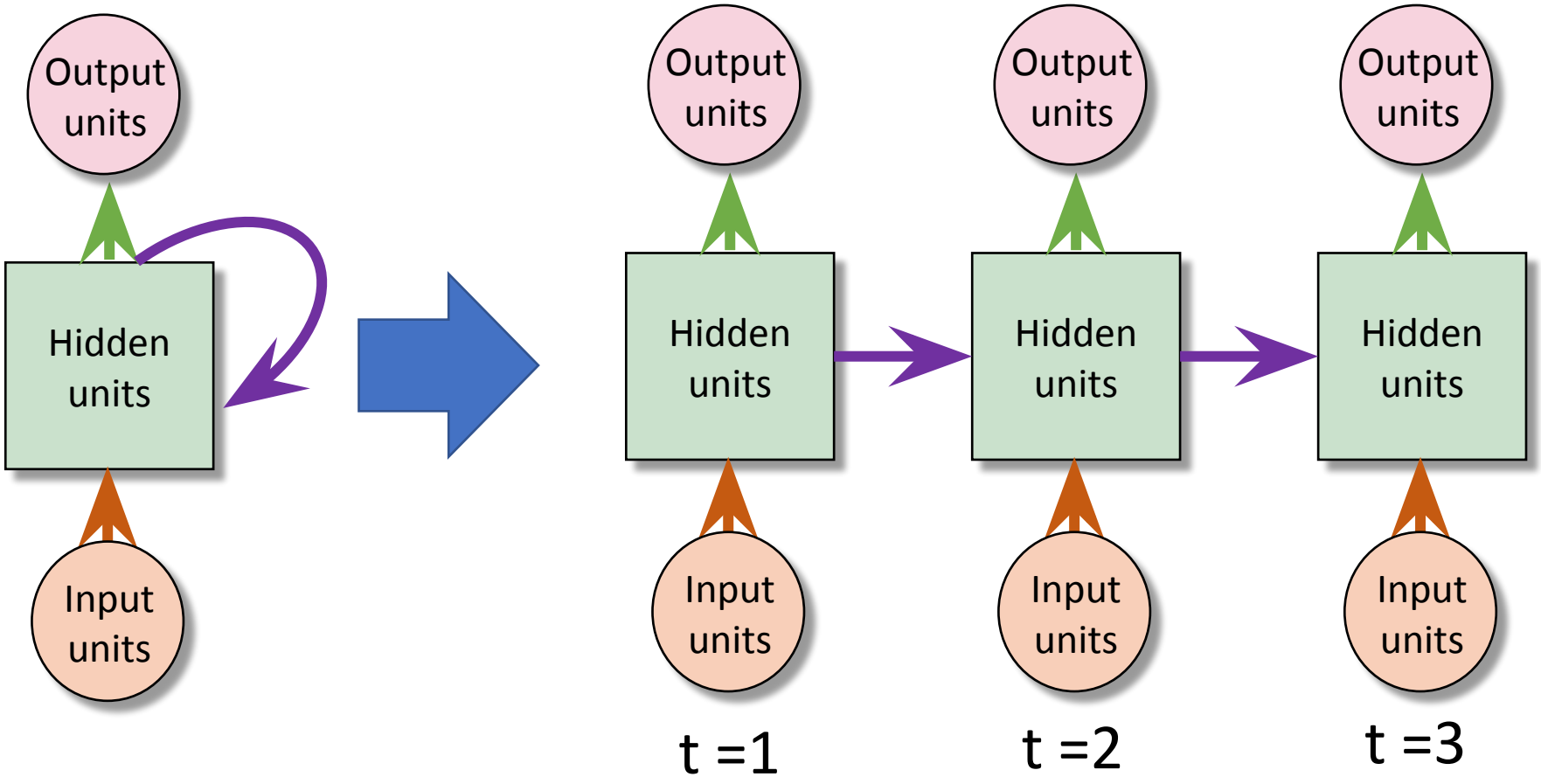
INNOPOLIS UNIVERSITY

Contents

- Recap! Recurrent Neural Network (RNN)
- Bidirectional RNN
- Long Short-Term Memories Cell (LSTM)
- Gate Recurrent Units (GRU)
- Attention Mechanism
- Self-Attention
- Transformer
- Summary

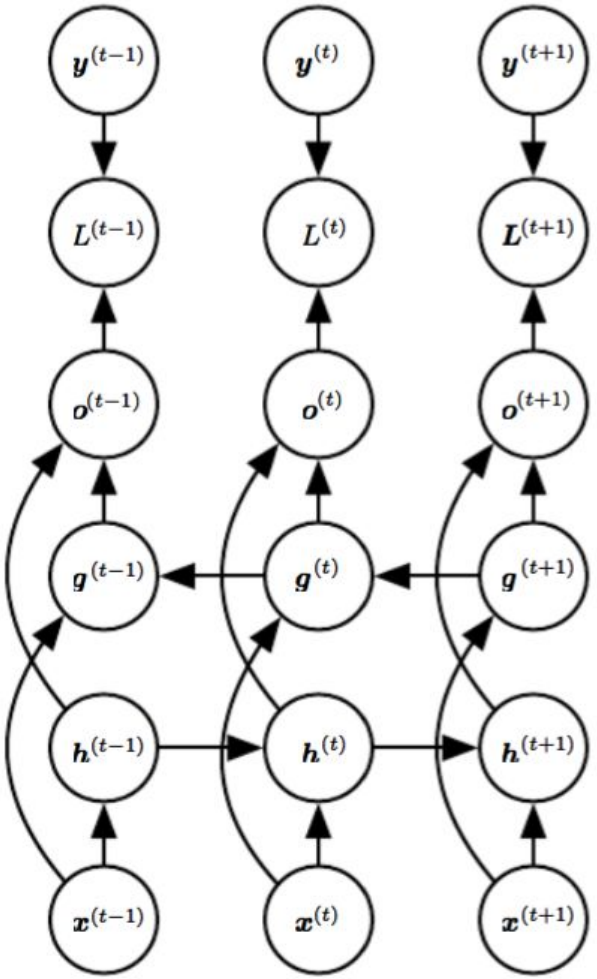
Recap! Recurrent Neural Network (RNN)

- Unfold the RNN (e.g., t=3)



Bidirectional RNNs

- We can think it combines two RNN
- One moves forward through time beginning from the start of the sequence while another RNN that moves backward through time beginning from the end of the sequence.



LSTM – Introduce a Memory Cell

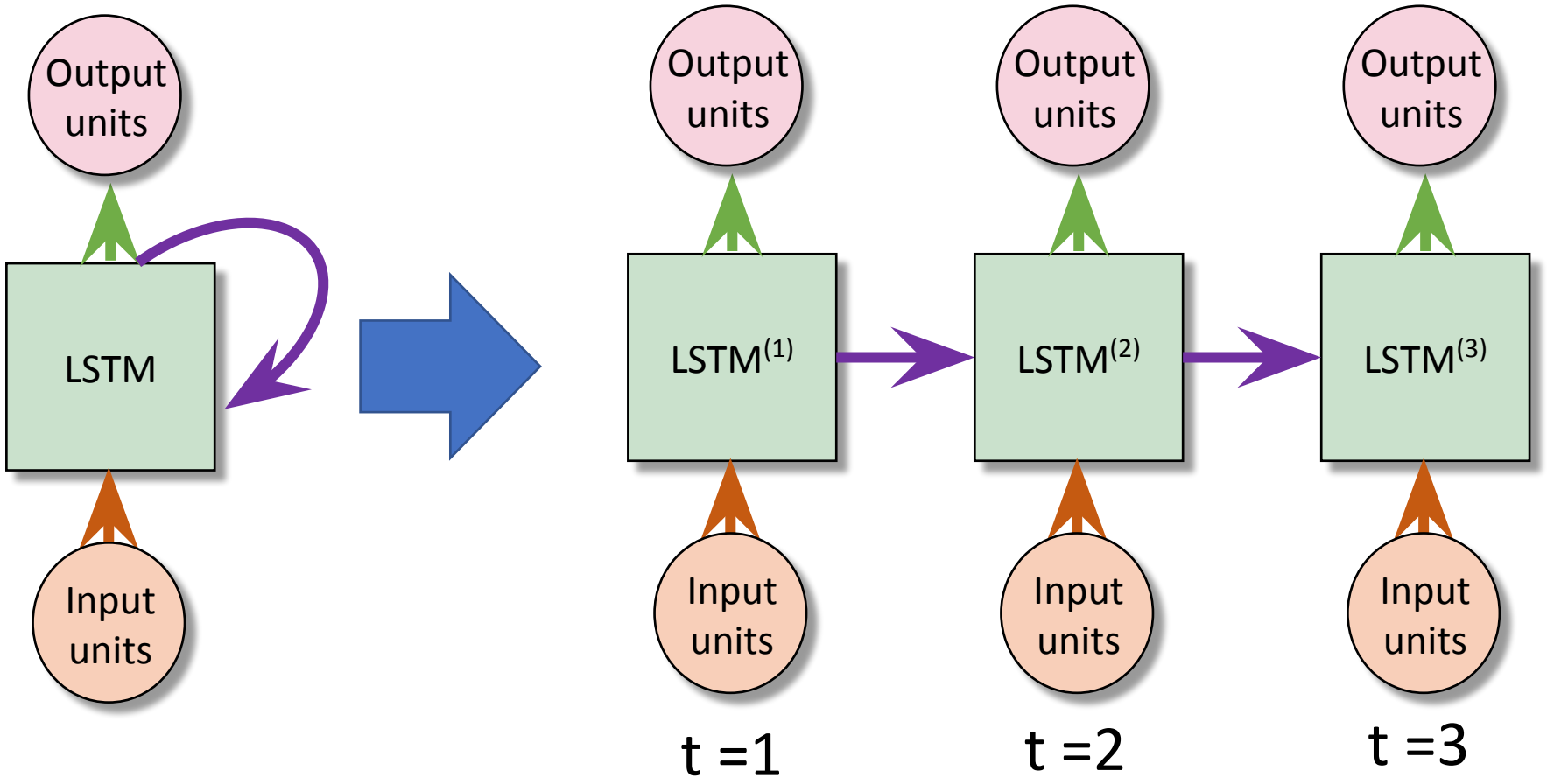
- LSTM was proposed as a solution to the vanishing gradients problem.

- On step t, there is a hidden state and a cell state
  - The cell stores long-term information
  - The LSTM can erase, write and read information from the cell

- The selection of which information is erased/written/read is controlled by three corresponding gates
  - On each timestep, each element of the gates can be open (1), closed (0), or somewhere in-between.
  - The gates are dynamic: their value is computed based on the current context

"Long short-term memory", Hochreiter and Schmidhuber, 1997.
https://www.bioinf.jku.at/publications/older/2604.pdf

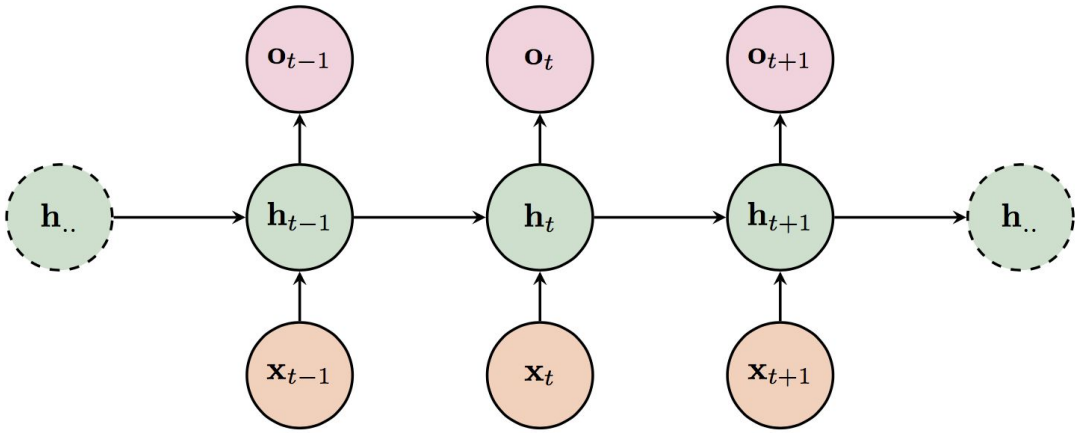
LSTM – Introduce a Memory Cell



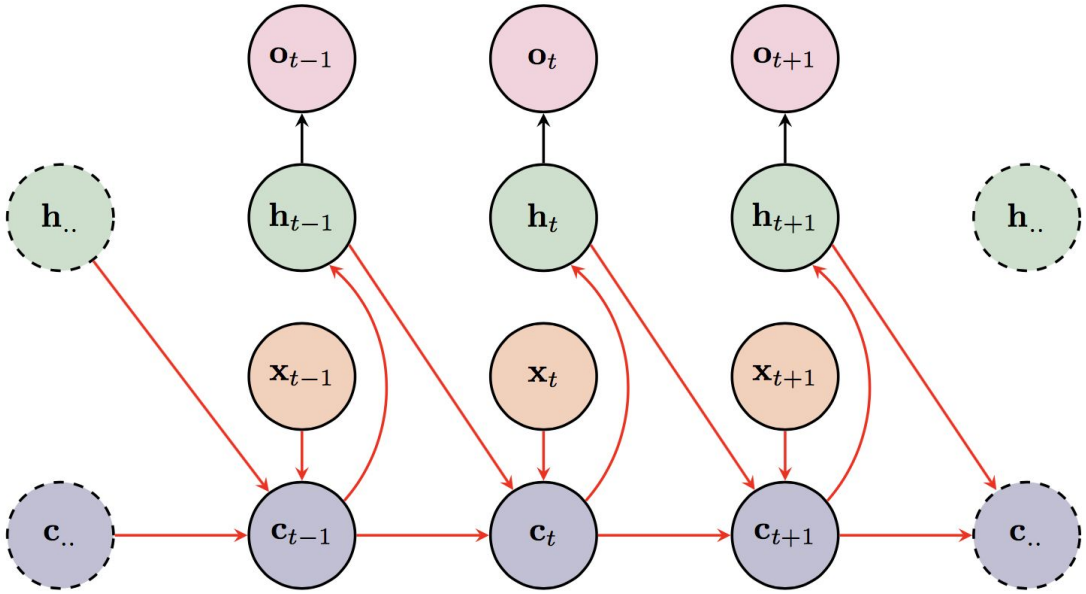
LSTM cell contains a hidden state and a cell state (Memory cell)

RNN vs LSTM

Graphical Structure

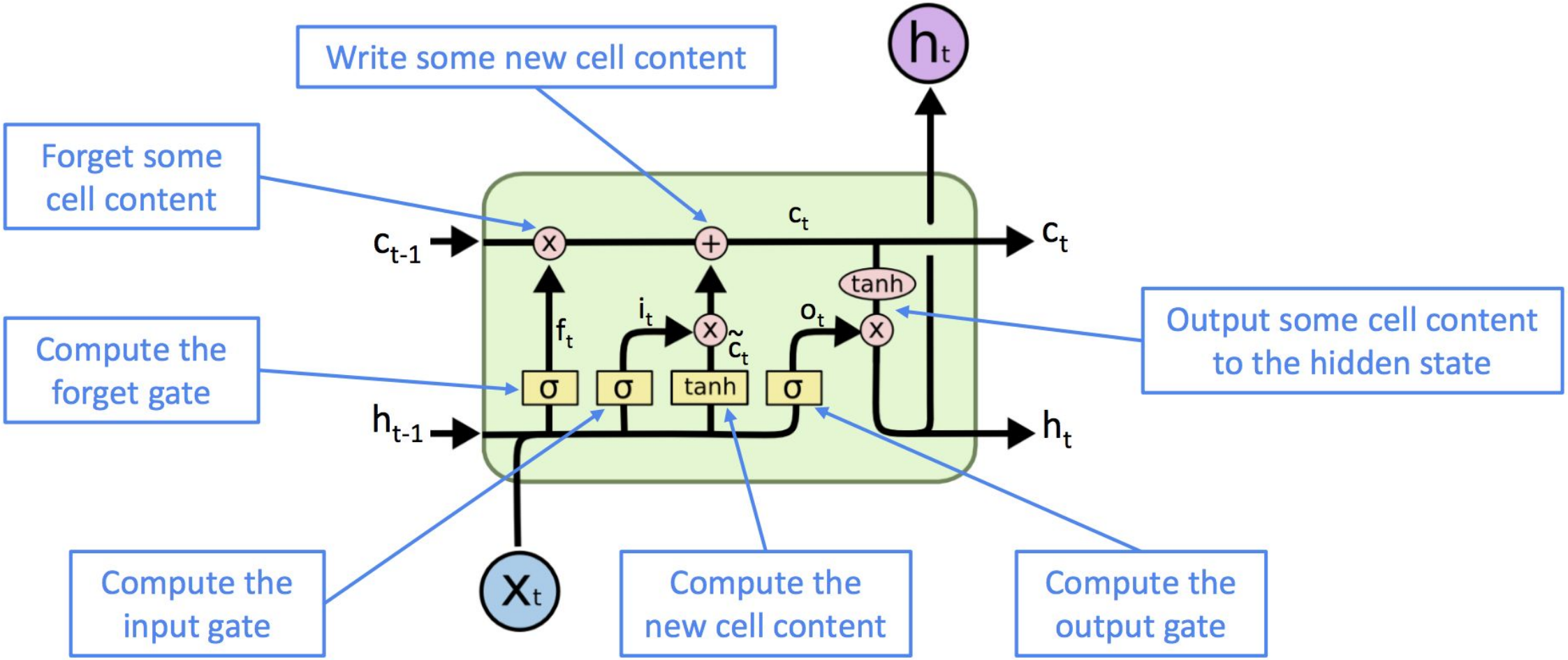


RNN



LSTM

LSTM – Memory Cell



Slide Credit: Abigail See

LSTM – Memory Cell

On time *t:* We have a sequence of inputs $x^{(t)}$, and we will compute a sequence of hidden states $h^{(t)}$ and cell states.

Forget gate: controls what is kept vs forgotten, from previous cell state

Input gate: controls what parts of the new cell content are written to cell

Output gate: controls what parts of cell are output to hidden state

New cell content: this is the new content to be written to the cell

Cell state: erase ("forget") some content from last cell state, and write ("input") some new cell content

Hidden state: read ("output") some content from the cell

Sigmoid function: all gate values are between 0 and 1

$$\boldsymbol{f}^{(t)} = \sigma\left(\boldsymbol{W}_f \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_f \boldsymbol{x}^{(t)} + \boldsymbol{b}_f\right)$$

$$\boldsymbol{i}^{(t)} = \sigma\left(\boldsymbol{W}_i \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_i \boldsymbol{x}^{(t)} + \boldsymbol{b}_i\right)$$

$$\boldsymbol{o}^{(t)} = \sigma\left(\boldsymbol{W}_o \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_o \boldsymbol{x}^{(t)} + \boldsymbol{b}_o\right)$$

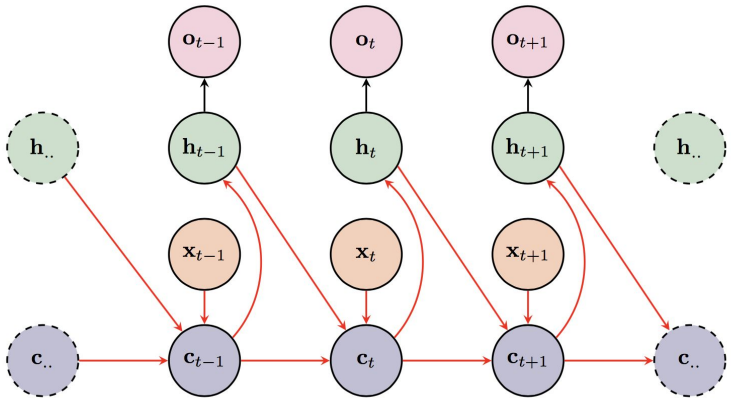
$$\tilde{\boldsymbol{c}}^{(t)} = \tanh\left(\boldsymbol{W}_c \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_c \boldsymbol{x}^{(t)} + \boldsymbol{b}_c\right)$$

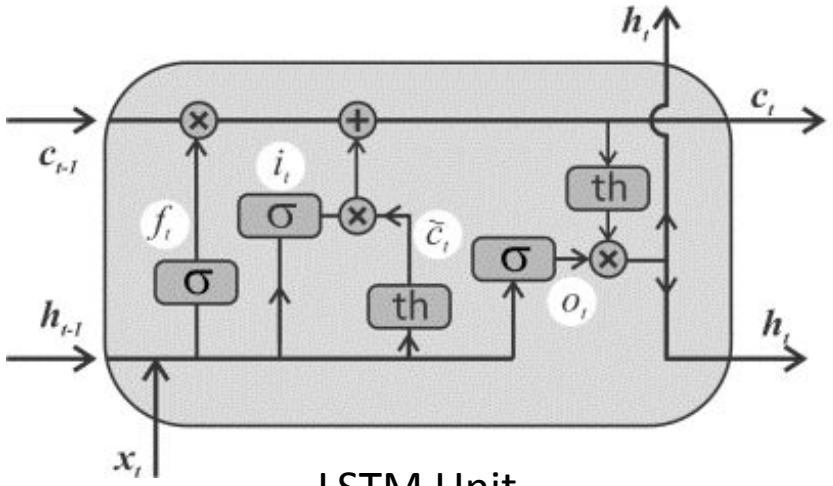
$$\boldsymbol{c}^{(t)} = \boldsymbol{f}^{(t)} \circ \boldsymbol{c}^{(t-1)} + \boldsymbol{i}^{(t)} \circ \tilde{\boldsymbol{c}}^{(t)}$$

$$\boldsymbol{h}^{(t)} = \boldsymbol{o}^{(t)} \circ \tanh \boldsymbol{c}^{(t)}$$

All these are vectors of same length n

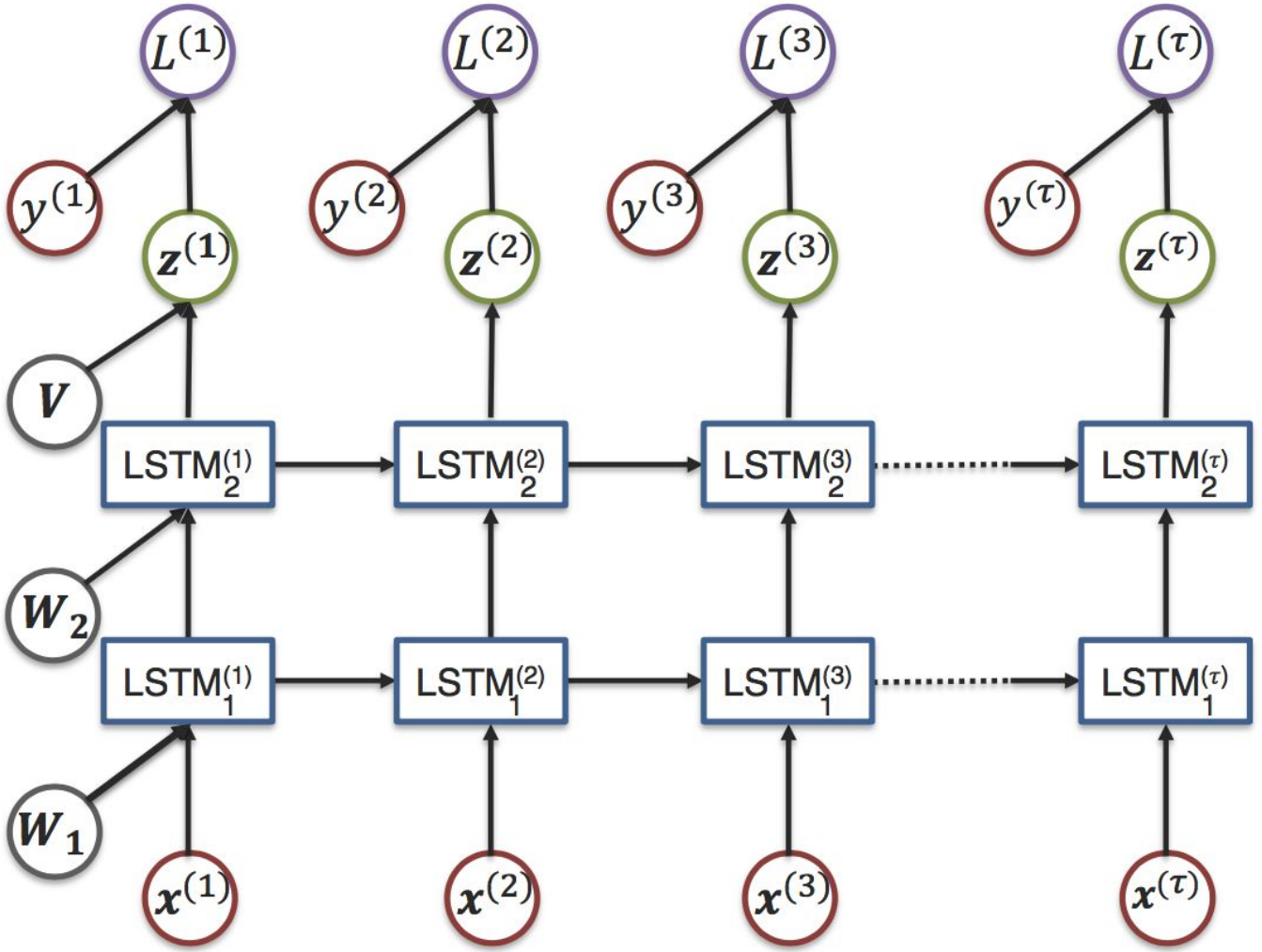
Gates are applied using element-wise product





LSTM Unit

Deep LSTM


$L^{(1)}$
$L^{(2)}$
$L^{(3)}$
$L^{(\tau)}$
$y^{(1)}$
$y^{(2)}$
$y^{(3)}$
$y^{(\tau)}$
$z^{(1)}$
$z^{(2)}$
$z^{(3)}$
$z^{(\tau)}$
V
W_2
W_1
$\text{LSTM}_2^{(1)}$
$\text{LSTM}_2^{(2)}$
$\text{LSTM}_2^{(3)}$
$\text{LSTM}_2^{(\tau)}$
$\text{LSTM}_1^{(1)}$
$\text{LSTM}_1^{(2)}$
$\text{LSTM}_1^{(3)}$
$\text{LSTM}_1^{(\tau)}$
$x^{(1)}$
$x^{(2)}$
$x^{(3)}$
$x^{(\tau)}$

Gated Recurrent Units (GRUs)

- Proposed by Cho et al. in 2014 as a simpler alternative to the LSTM.
- On each timestep t: we have input and hidden state (no cell state).

Update gate: controls what parts of hidden state are updated vs preserved

Reset gate: controls what parts of previous hidden state are used to compute new content

New hidden state content: reset gate selects useful parts of prev hidden state. Use this and current input to compute new hidden content.

Hidden state: update gate simultaneously controls what is kept from previous hidden state, and what is updated to new hidden state content

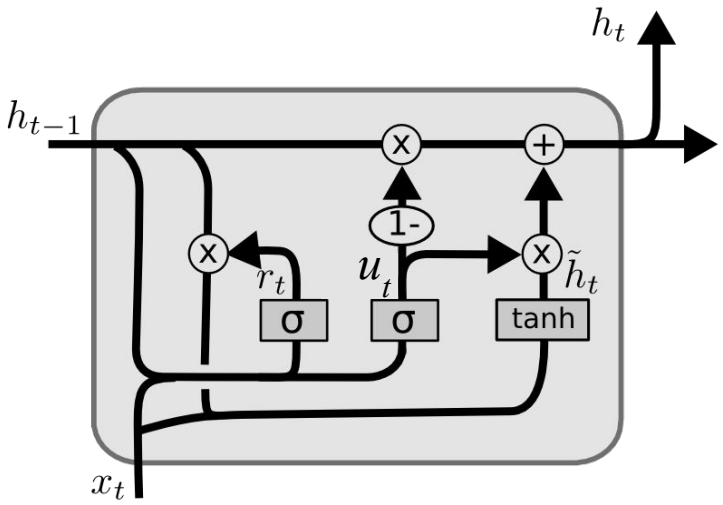
$$\boldsymbol{u}^{(t)} = \sigma\left(\boldsymbol{W}_u \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_u \boldsymbol{x}^{(t)} + \boldsymbol{b}_u\right)$$

$$\boldsymbol{r}^{(t)} = \sigma\left(\boldsymbol{W}_r \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_r \boldsymbol{x}^{(t)} + \boldsymbol{b}_r\right)$$

$$\tilde{\boldsymbol{h}}^{(t)} = \tanh\left(\boldsymbol{W}_h(\boldsymbol{r}^{(t)} \circ \boldsymbol{h}^{(t-1)}) + \boldsymbol{U}_h \boldsymbol{x}^{(t)} + \boldsymbol{b}_h\right)$$

$$\boldsymbol{h}^{(t)} = (1 - \boldsymbol{u}^{(t)}) \circ \boldsymbol{h}^{(t-1)} + \boldsymbol{u}^{(t)} \circ \tilde{\boldsymbol{h}}^{(t)}$$

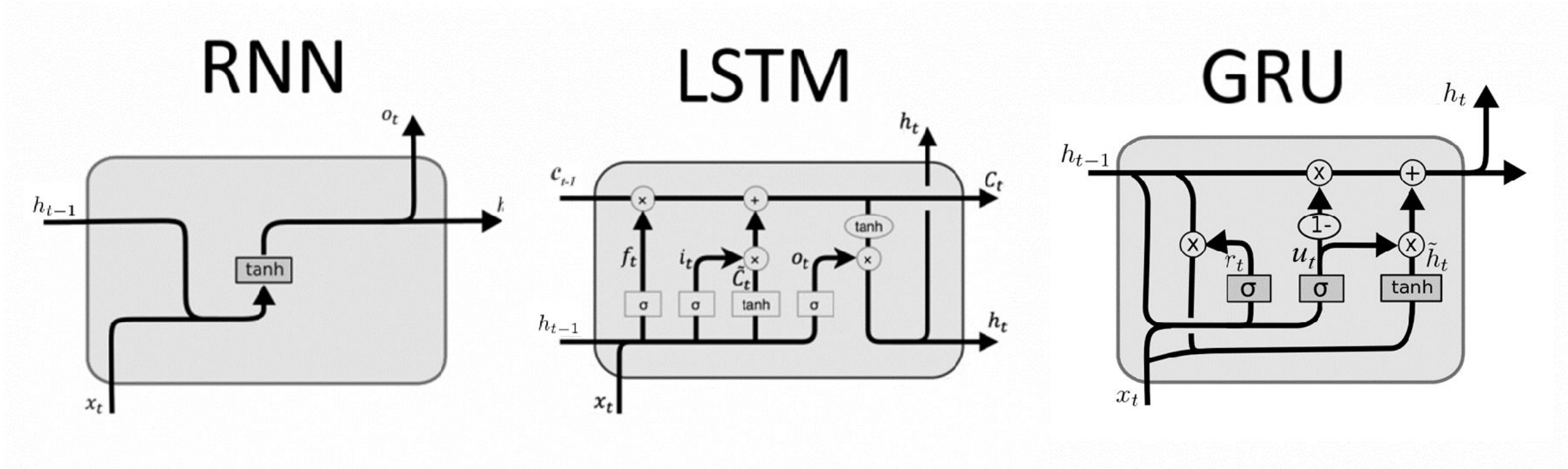
How does this solve vanishing gradient?
Like LSTM, GRU makes it easier to retain info long-term (e.g. by setting update gate to 0)



GRU Unit

"Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation", Cho et al. 2014, https://arxiv.org/pdf/1406.1078v3.pdf

RNN, LSTM and GRU



LSTM vs GRU

- Researchers have proposed many gated RNN variants, but LSTM and GRU are the most widely-used.
- The biggest difference is that GRU is quicker to compute and has fewer parameters
- There is no conclusive evidence that one consistently performs better than the other
- LSTM is a good default choice (especially if your data has particularly long dependencies, or you have lots of training data)
- Rule of thumb: start with LSTM, but switch to GRU if you want something more efficient

Transformers: Why new architecture?

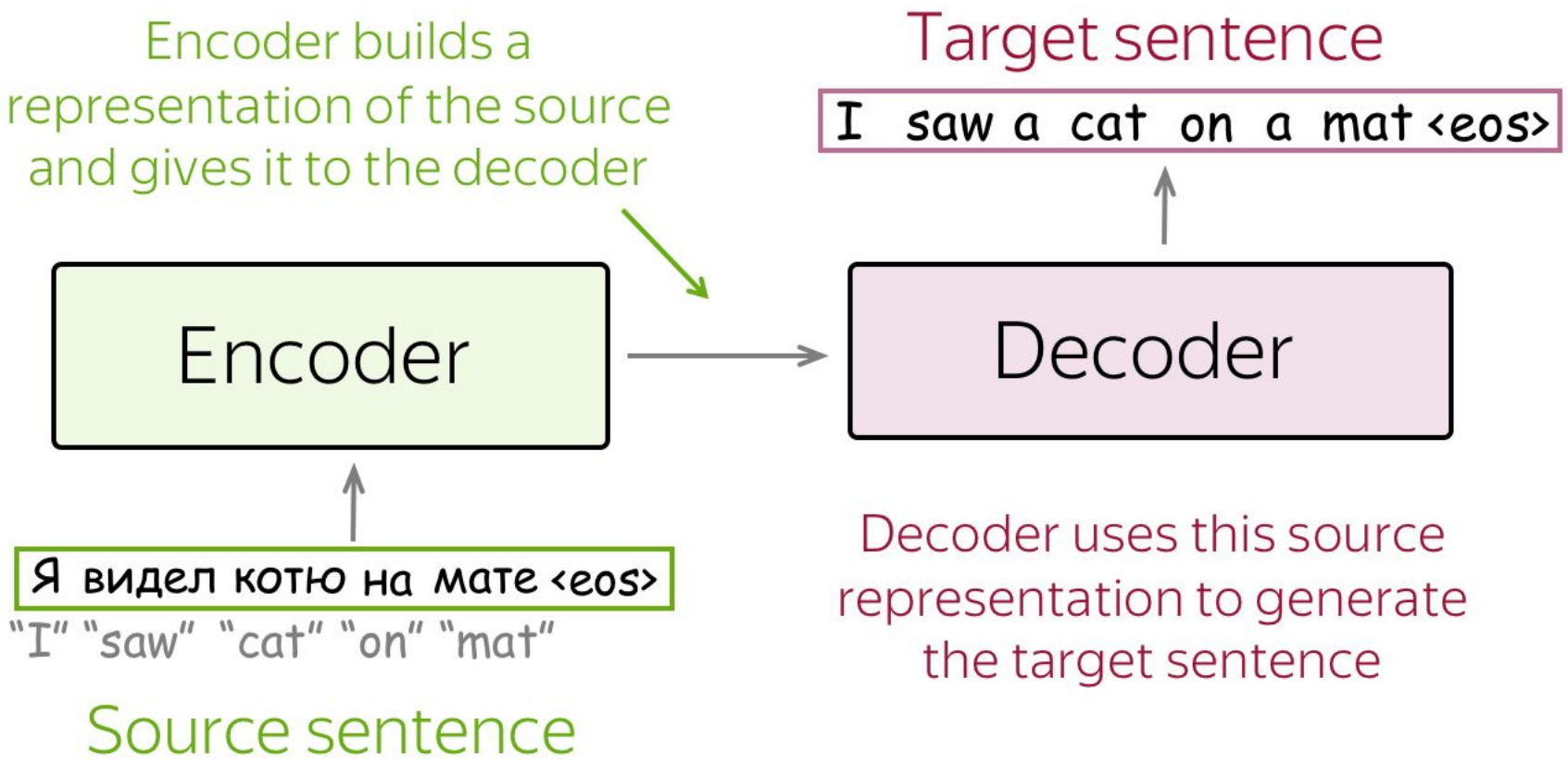
- **Challenges with RNN**
  - Long range dependencies
  - Gradient vanishing and explosion
  - Large number of training steps
  - Recurrence prevents parallel computation

- **Transformer Networks**
  - Facilitate long range dependencies
  - No gradient vanishing and explosion
  - Fewer training steps
  - No recurrence that facilitate parallel computation

Source: University of Waterloo (Pascal Poupart lecture slide)

Transformer Networks

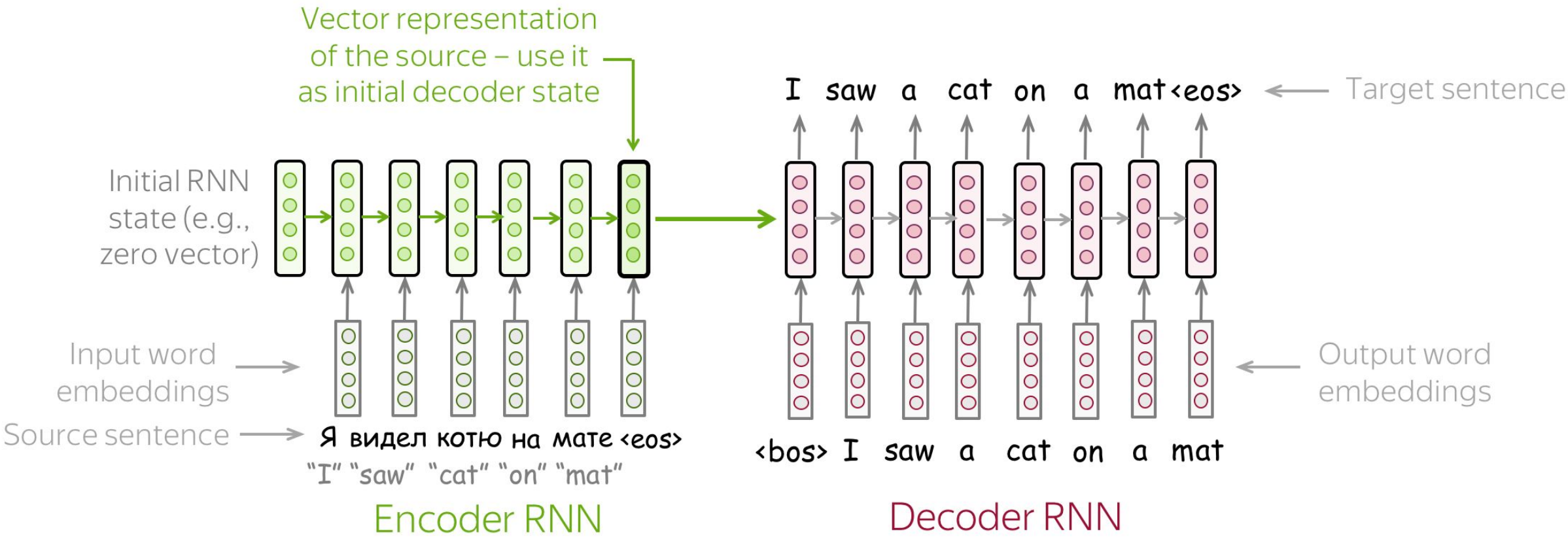
First understand Sequence to Sequence Model



Slide credit: Lena Voita

The Simplest Model

Two RNNs for Encoder and Decoder

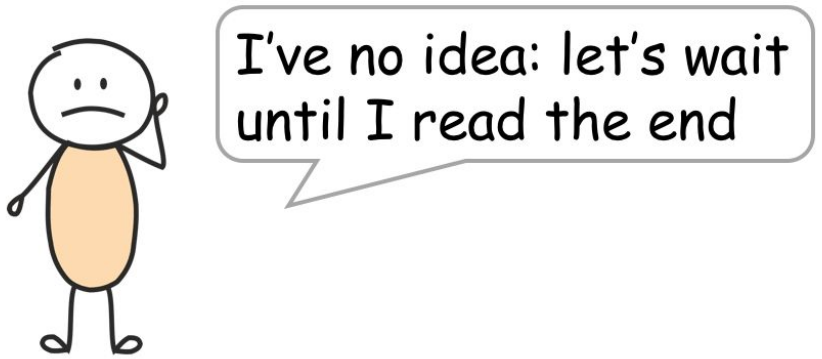


Slide credit: Lena Voita

Transformer Networks

I arrived at the bank after crossing the ... ...street? ...river?

What does **bank** mean in this sentence?



RNNs

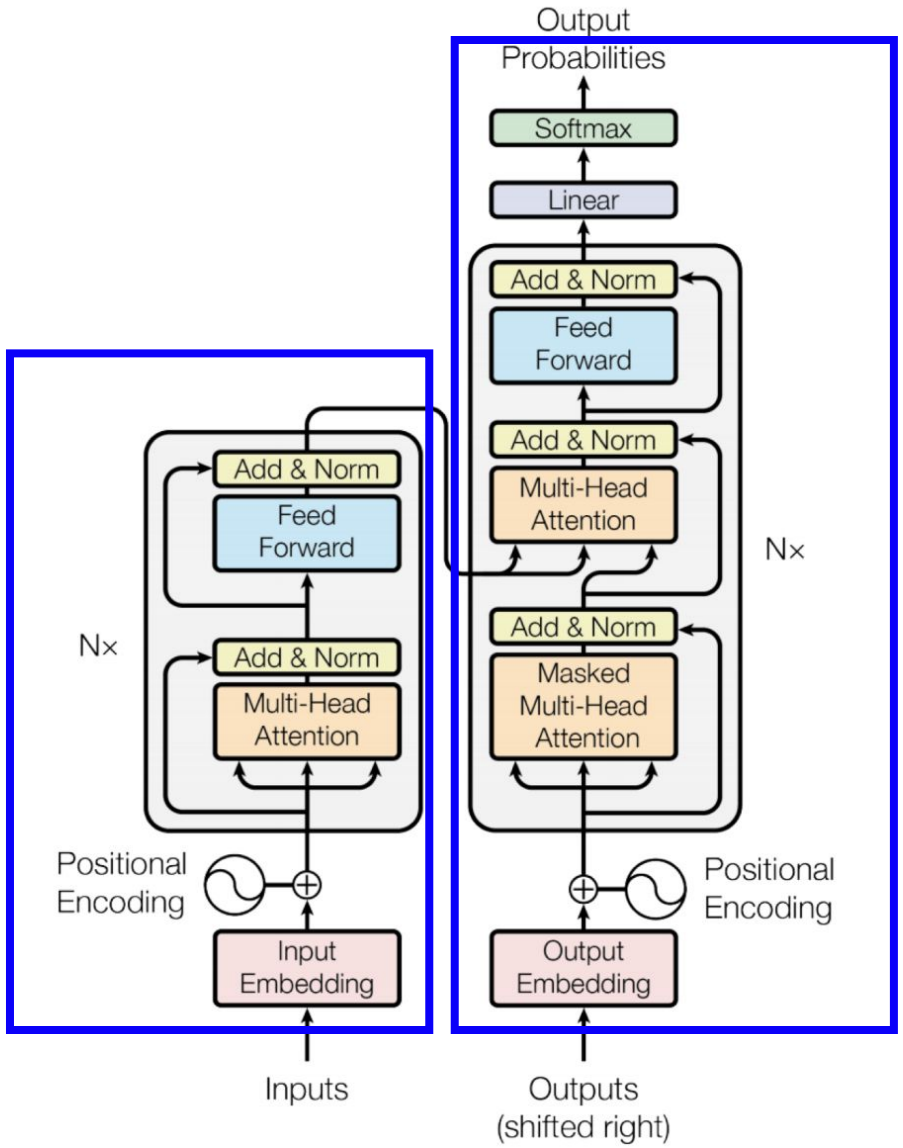
O(N) steps to process a sentence with length N

Transformer

Constant number of steps to process any sentence

Slide credit: Lena Voita

Transformer Basics



Encoder

Decoder

- The basic building blocks of transformer networks

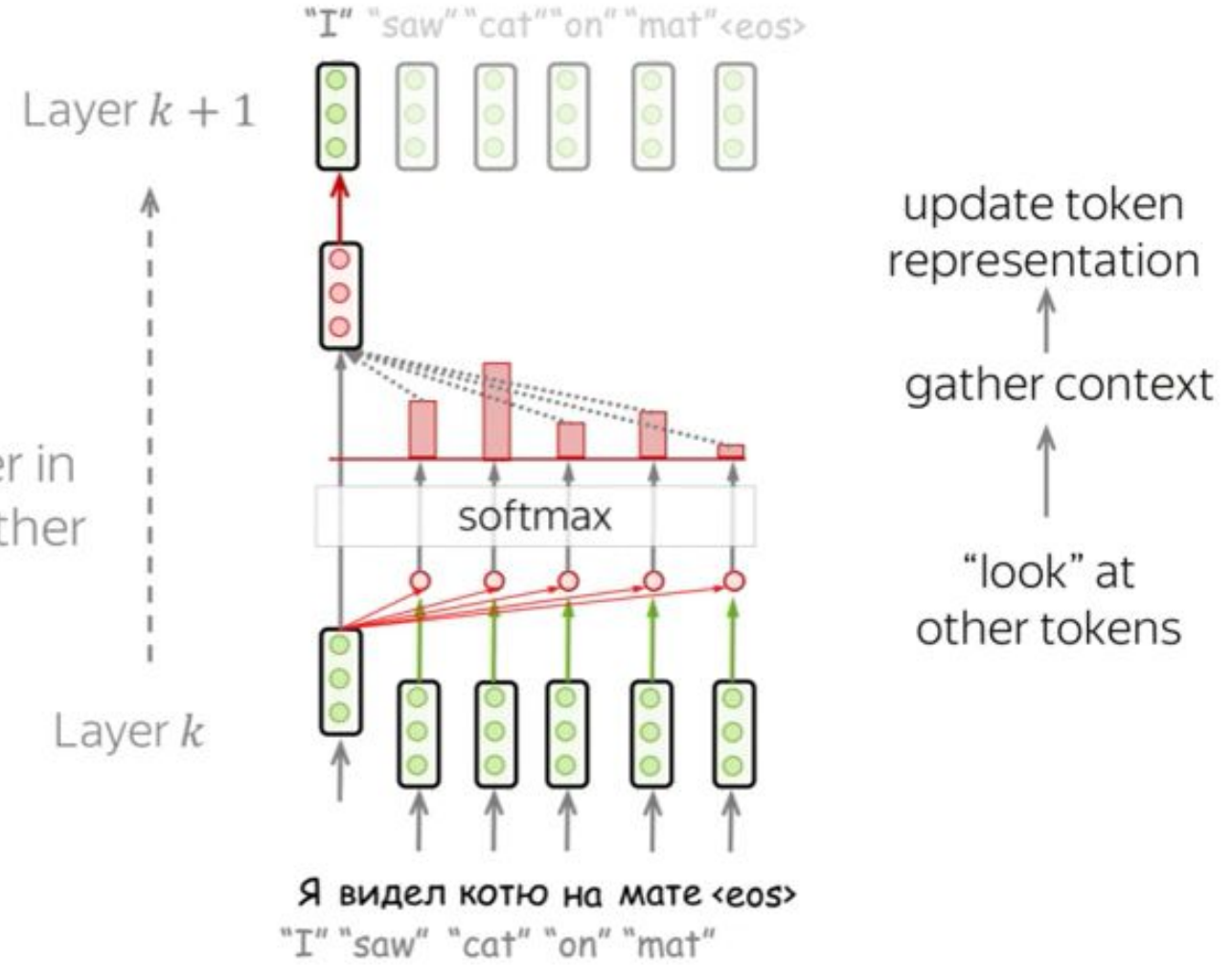
Self-Attention layers!

Self-Attention

$$Attention(q, k, v) = \overbrace{softmax\left(\frac{qk^T}{\sqrt{d_k}}\right)}^{\text{Attention weights}} v$$

from (q) · to (k, v) · vector dimensionality of K, V

Tokens try to understand themselves better in context of each other



Slide credit: Lena Voita

Self-Attention

$$Attention(q, k, v) = \overbrace{softmax\left(\frac{qk^T}{\sqrt{d_k}}\right)v}^{\text{Attention weights}}$$

from (q) · to (k) · vector dimensionality of K, V

Each vector receives three representations ("roles")

$W_Q \times = $ Query: vector **from** which the attention is looking

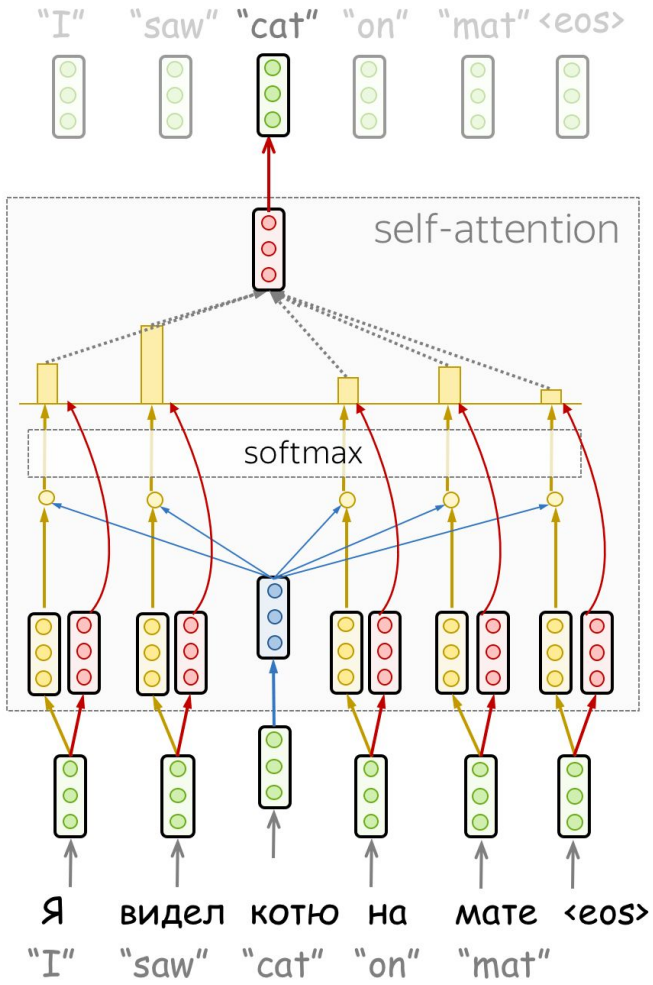
"Hey there, do you have this information?"

$W_K \times = $ Key: vector **at** which the query looks to compute weights

"Hi, I have this information – give me a large weight!"

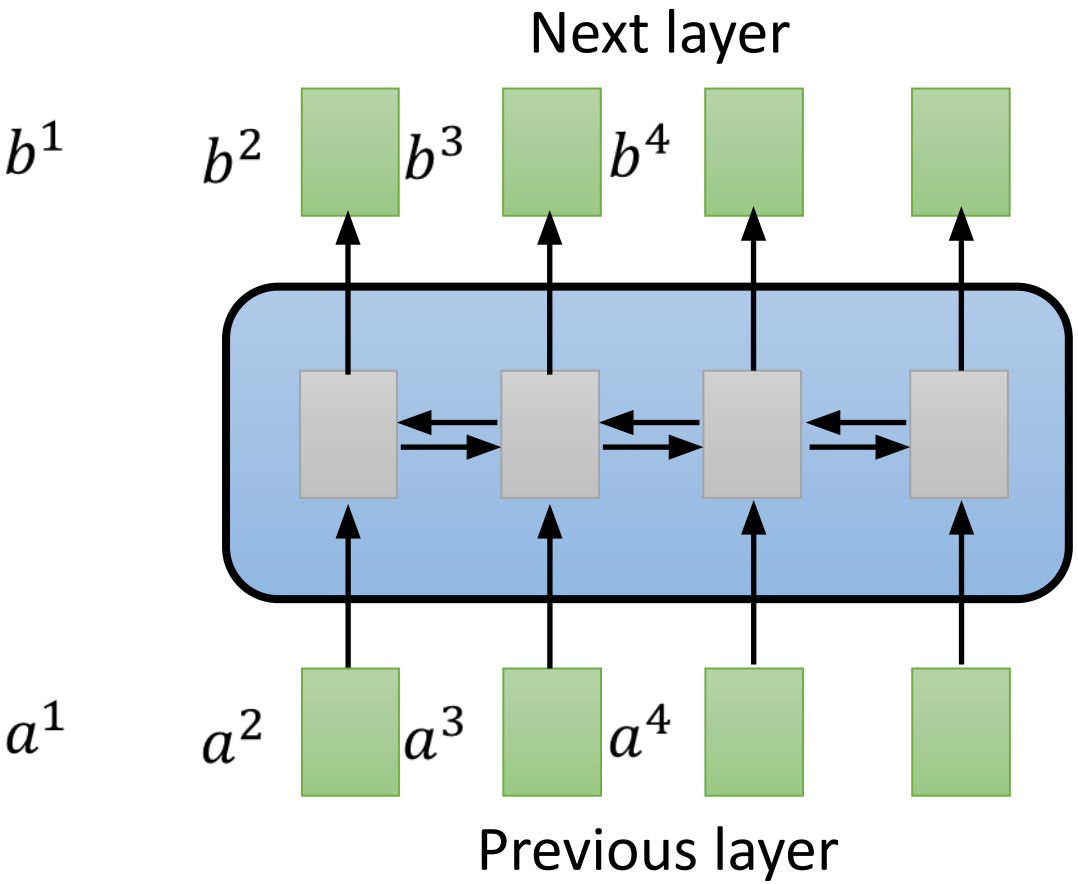
$W_V \times = $ Value: their weighted sum is attention output

"Here's the information I have!"



Slide credit: Lena Voita

Self-Attention



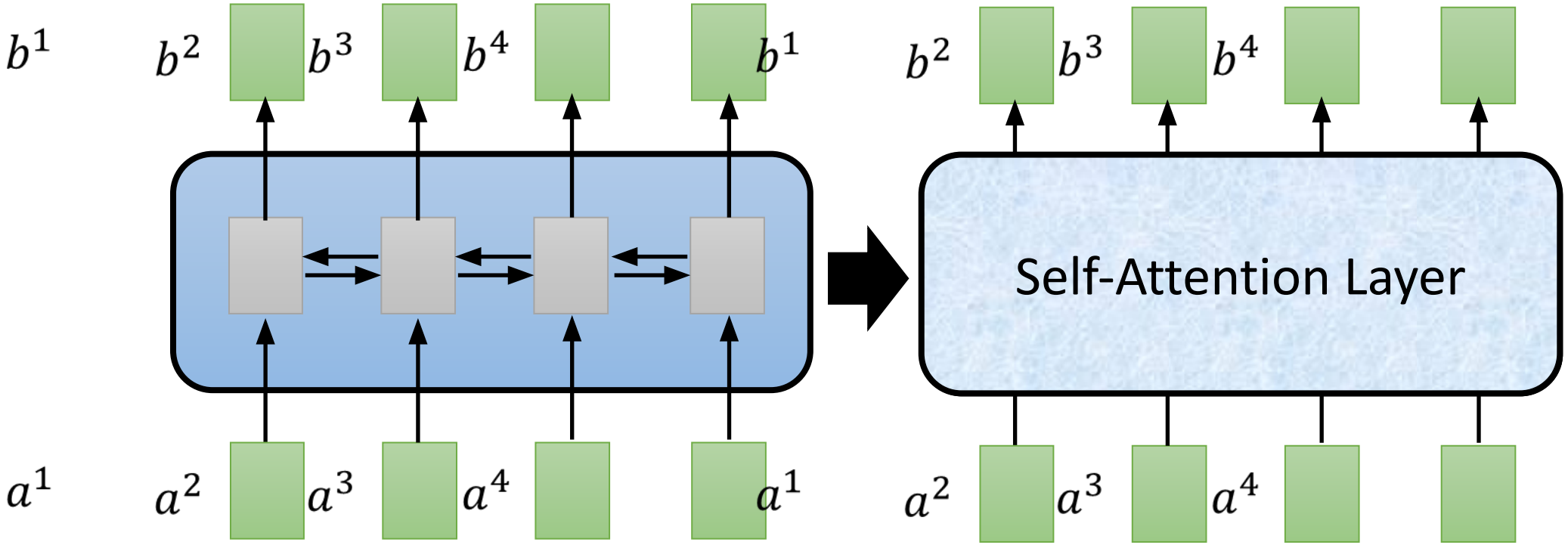
Hard to parallelize

Source: Hung-yi Lee Presentation

Self-Attention

b^i is obtained based on the whole input sequence.

b^1, b^2, b^3, b^4 can be parallelly computed.



You can try to replace anything that has been done by RNN with self-attention.

Source: Hung-yi Lee Presentation

Self-Attention

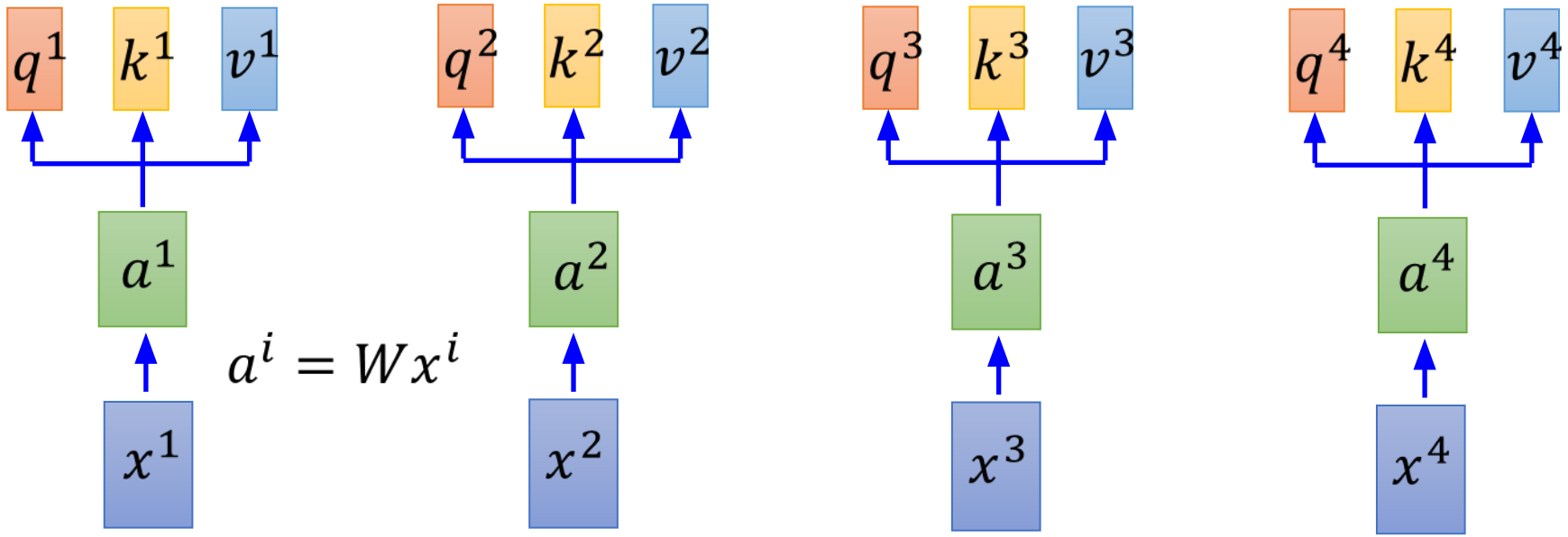
Each input token in self-attention receives three representations corresponding to the roles it can play:

- query (q) – asking for information
- key(k) – saying that it has some information
- value(v) – giving the information.

$$Attention(q, k, v) = \overbrace{softmax\left(\frac{qk^T}{\sqrt{d_k}}\right)}^{\text{Attention weights}} v$$

q: from, k: to, $\sqrt{d_k}$: vector dimensionality of K, V

$$q^i = W^q a^i$$
$$k^i = W^k a^i$$
$$v^i = W^v a^i$$

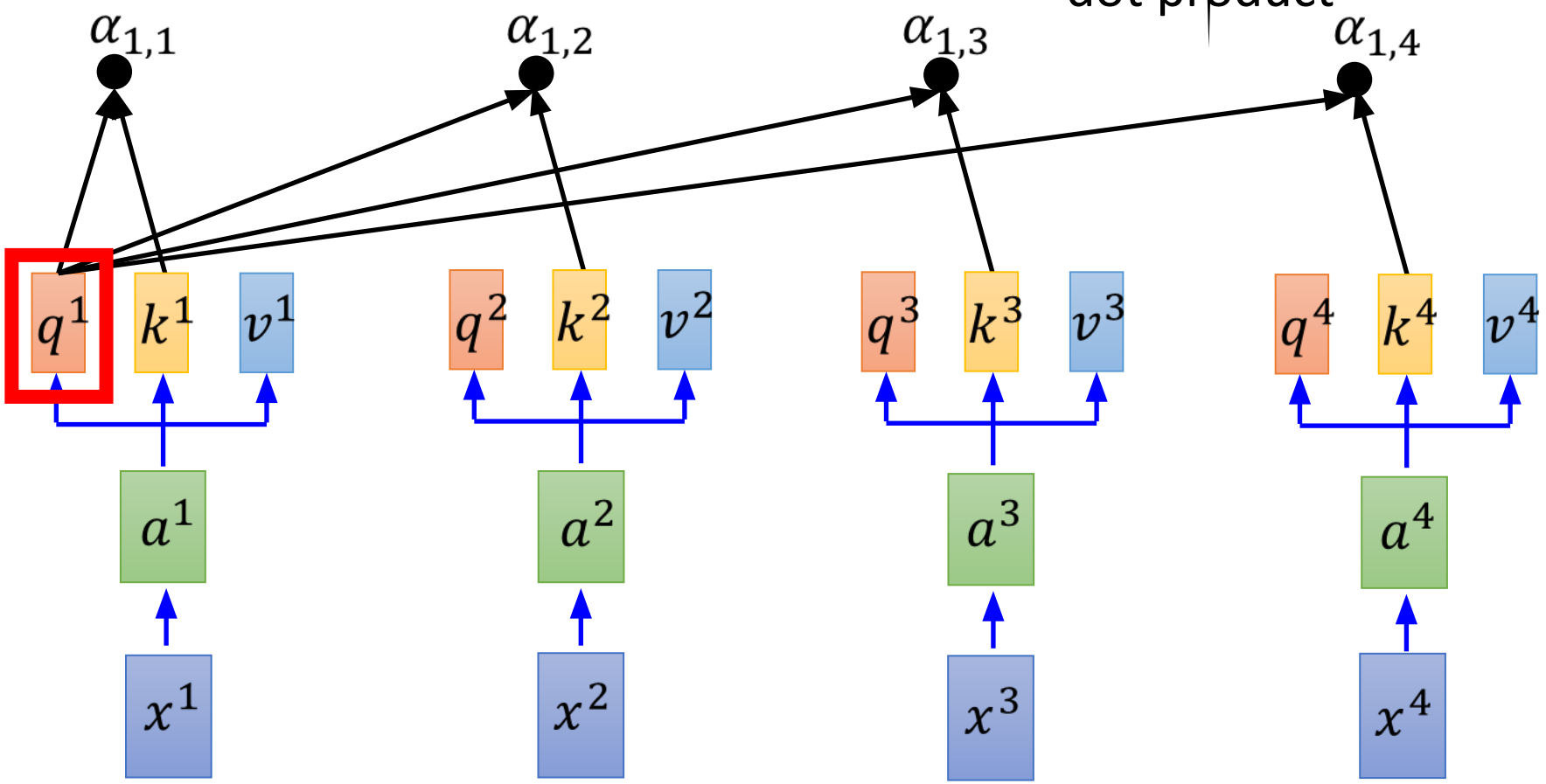


Source: Hung-yi Lee Presentation

Self-Attention

Scaled Dot-Product Attention: $\alpha_{1,i} = q^1 \cdot k^i / \sqrt{d}$

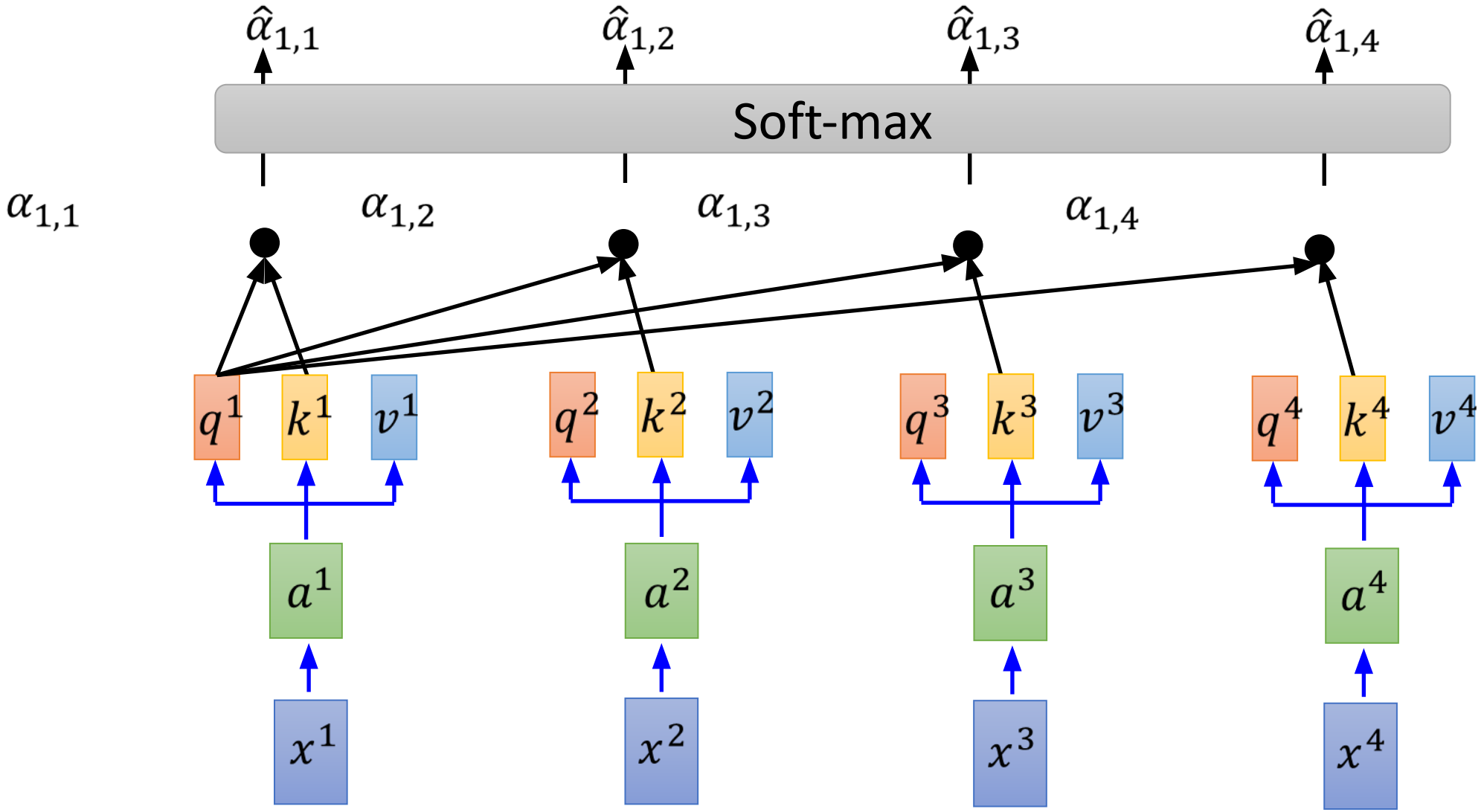
dot product



Source: Hung-yi Lee Presentation

Self-Attention

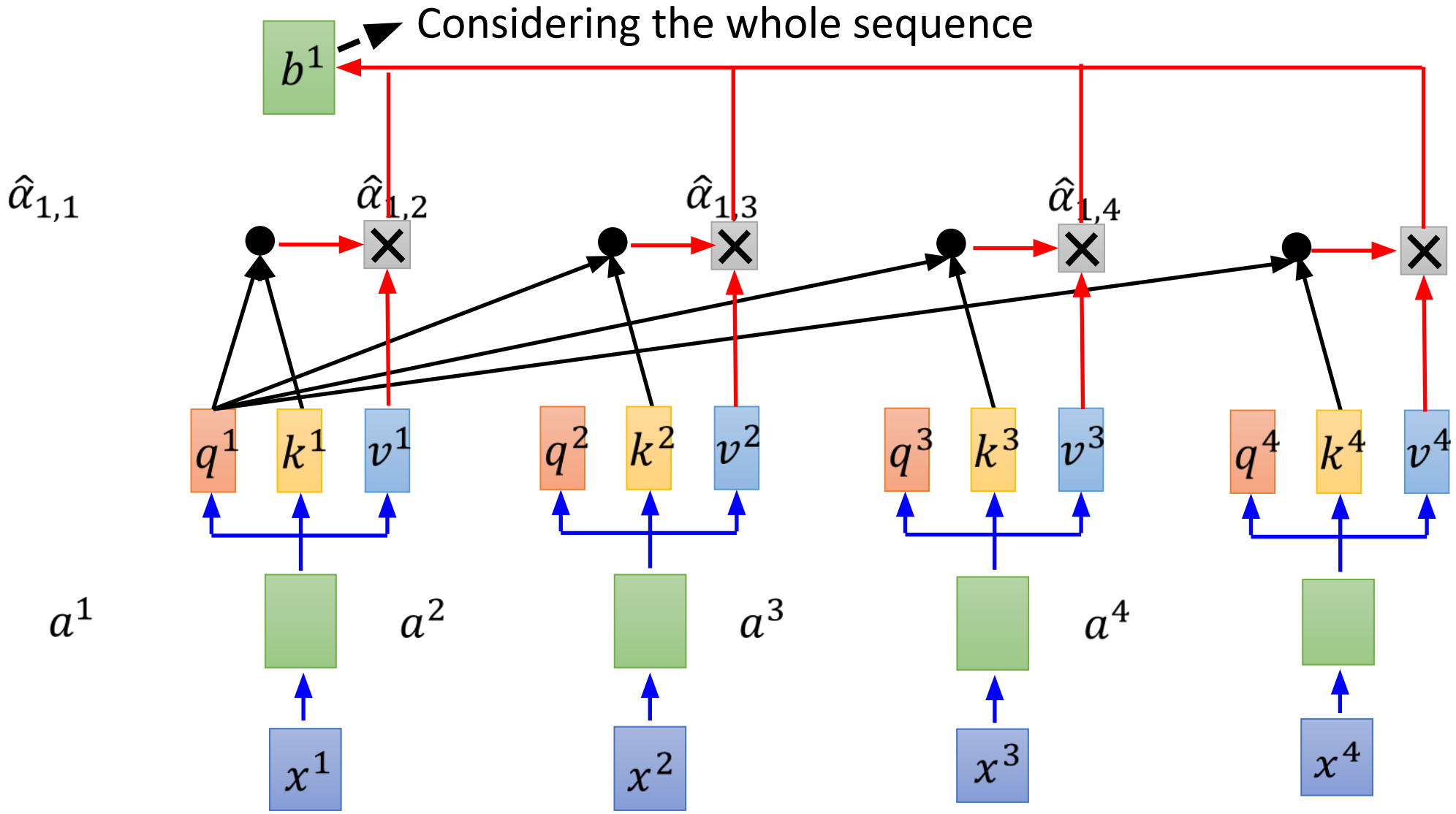
$$\hat{\alpha}_{1,i} = exp(\alpha_{1,i}) / \sum_j exp(\alpha_{1,j})$$



Source: Hung-yi Lee Presentation

Self-Attention

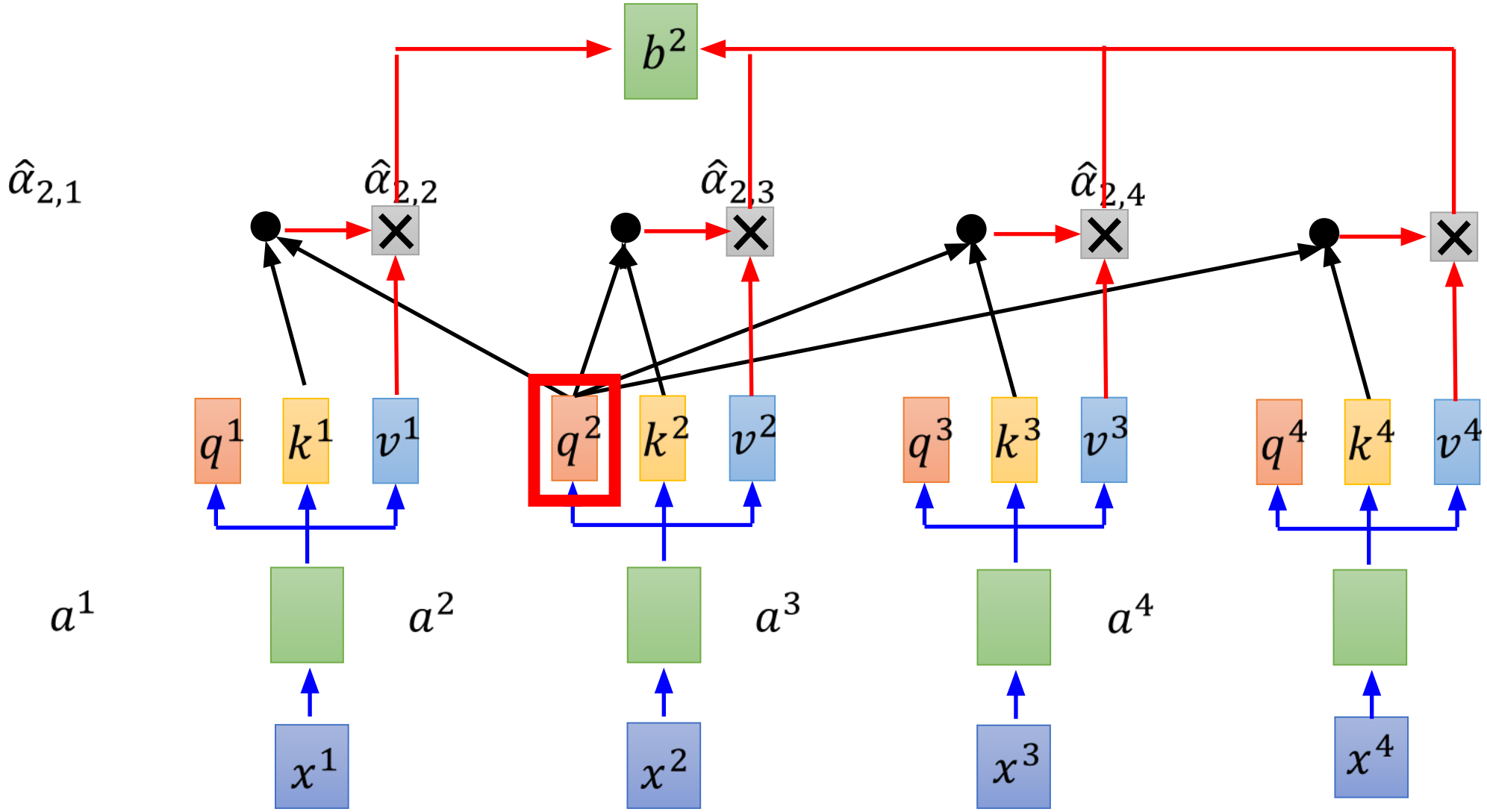
$$b^1 = \sum_i \hat{\alpha}_{1,i} v^i$$



Source: Hung-yi Lee Presentation

Self-Attention

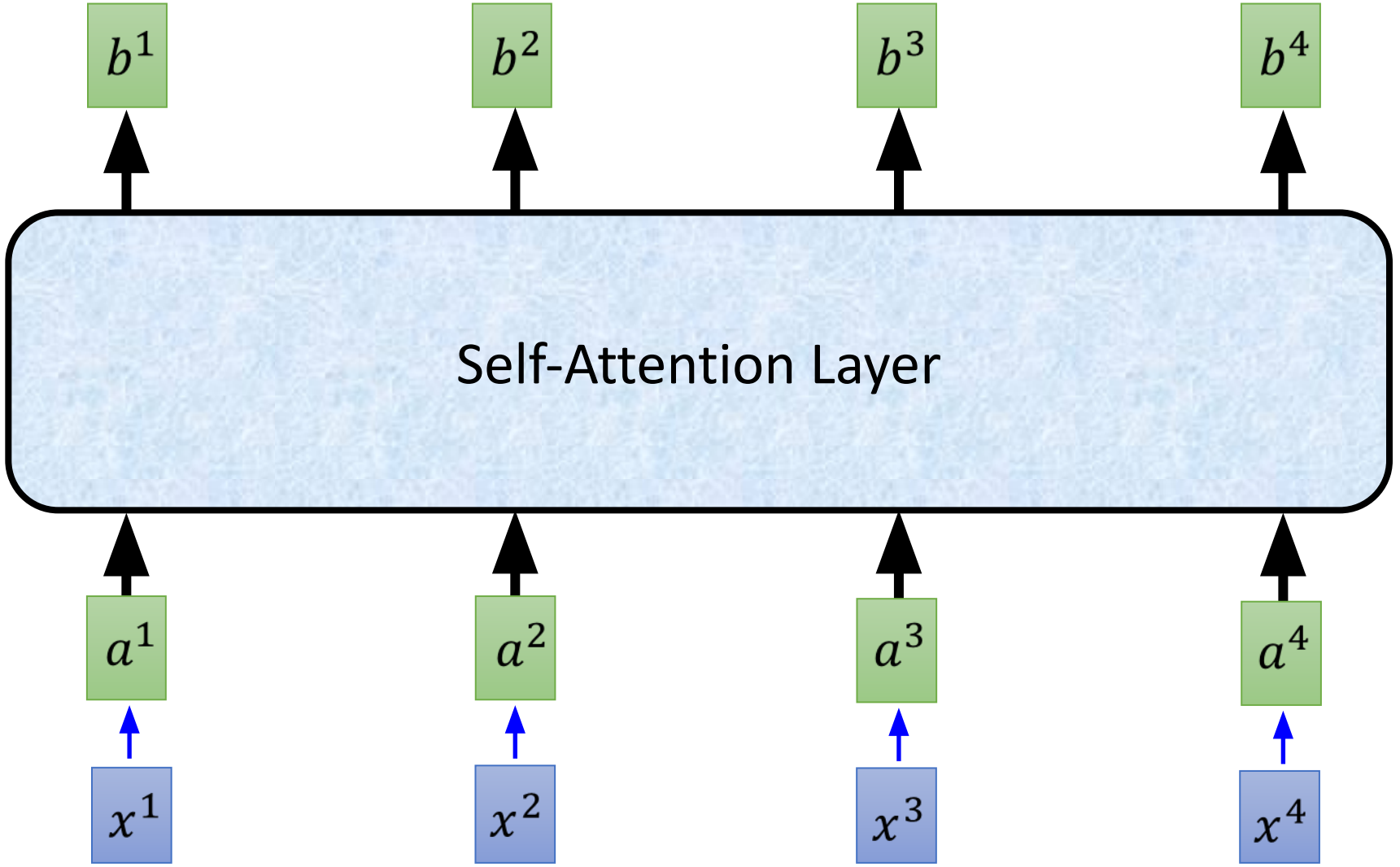
$$b^2 = \sum_i \hat{\alpha}_{2,i} v^i$$



Source: Hung-yi Lee Presentation

Self-Attention

b^1, b^2, b^3, b^4 can be parallelly computed.



Source: Hung-yi Lee Presentation

Self-Attention

$$q^i = W^q a^i$$

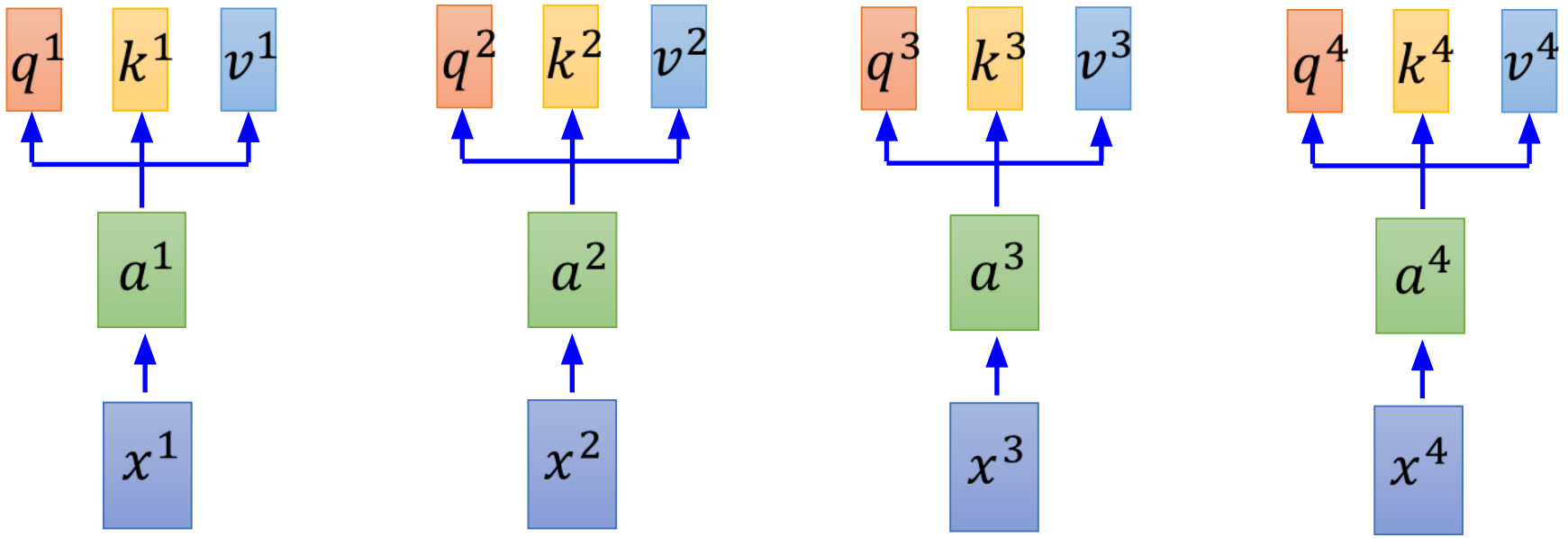
$$k^i = W^k a^i$$

$$v^i = W^v a^i$$

$$Q \quad q^1 q^2 q^3 q^4 = W^q \; a^1 a^2 a^3 a^4$$

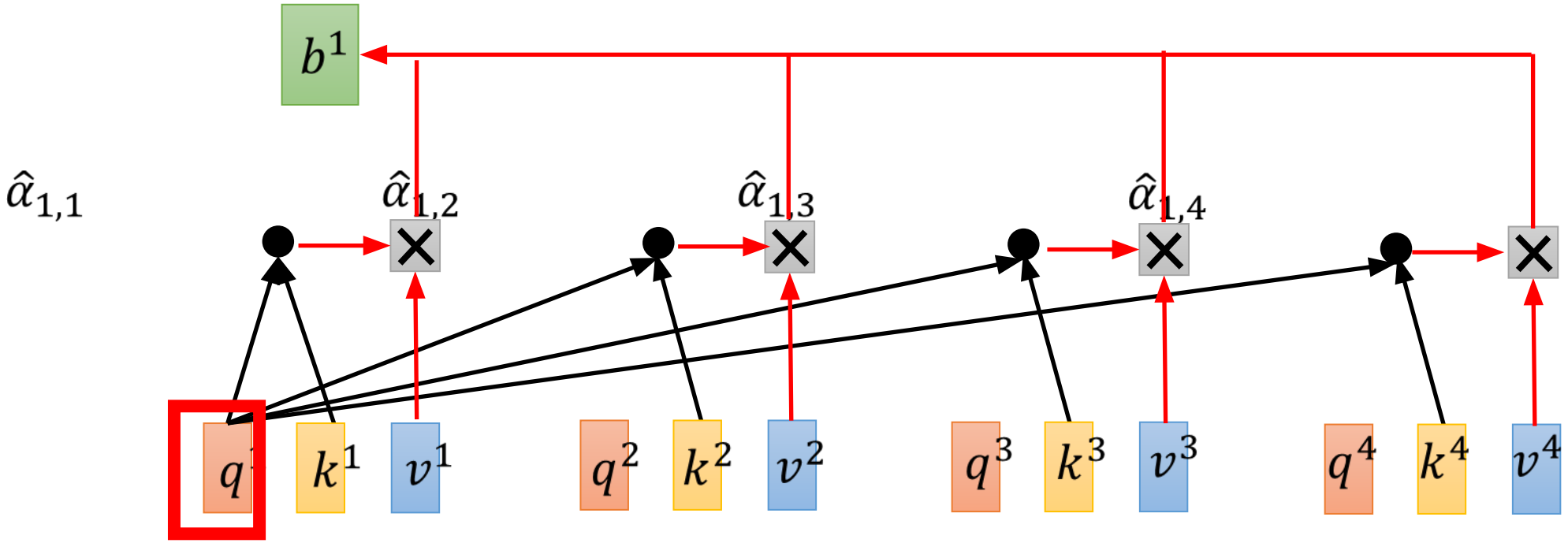
$$K \quad k^1 k^2 k^3 k^4 = W^k \; a^1 a^2 a^3 a^4$$

$$V \quad v^1 v^2 v^3 v^4 = W^v \; a^1 a^2 a^3 a^4$$



Source: Hung-yi Lee Presentation

Self-Attention



$$\alpha_{1,1} = k^1 q^1 \qquad \alpha_{1,2} = k^2 q^1$$

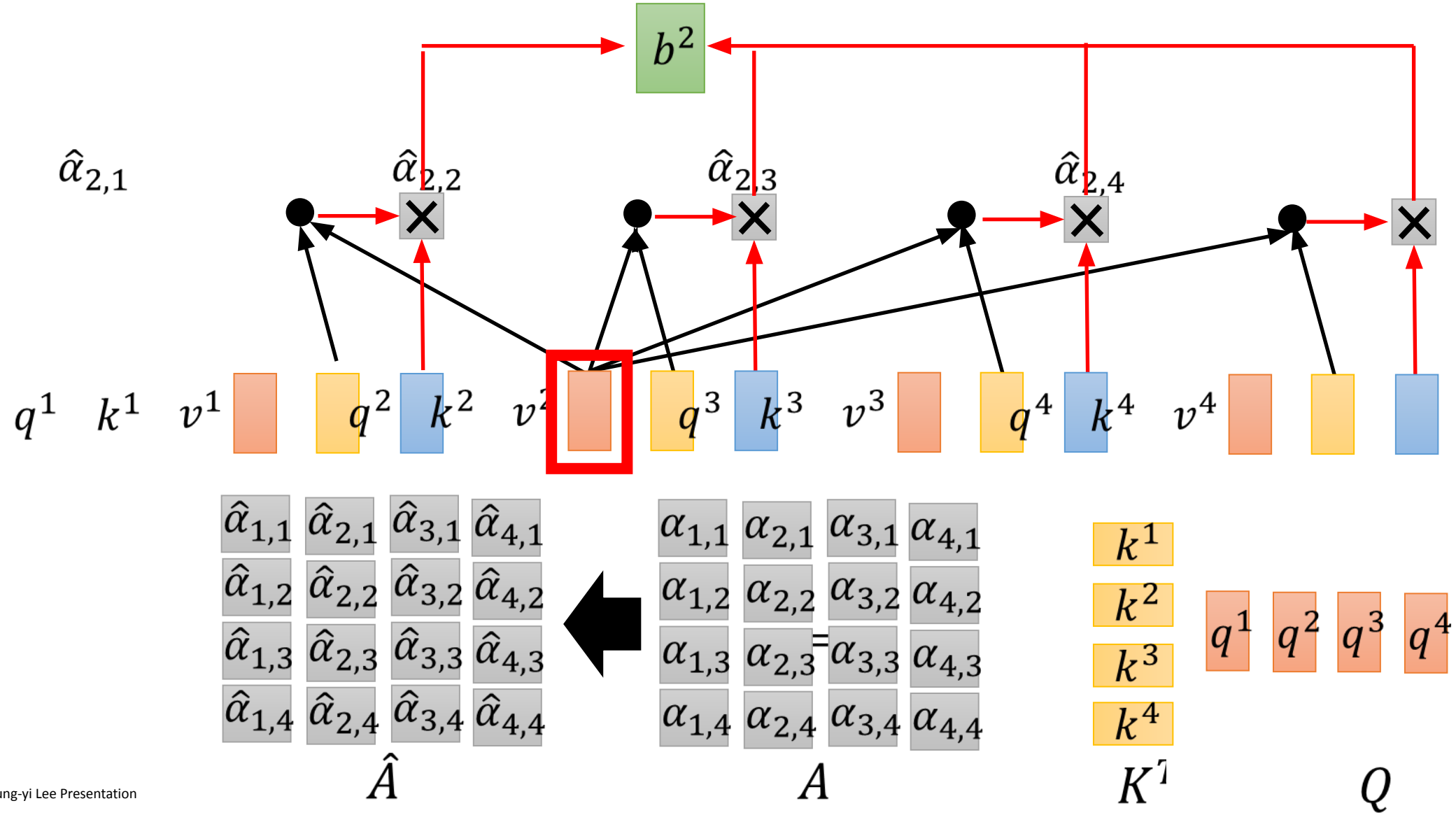
$$\alpha_{1,3} = k^3 q^1 \qquad \alpha_{1,4} = k^4 q^1 \qquad = \begin{matrix} \alpha_{1,1} \\ \alpha_{1,2} \\ \alpha_{1,3} \\ \alpha_{1,4} \end{matrix} \begin{matrix} k^1 \\ k^2 \\ k^3 \\ k^4 \end{matrix} q^1$$

(ignore $\sqrt{d}$ for simplicity)

Source: Hung-yi Lee Presentation

Self-Attention

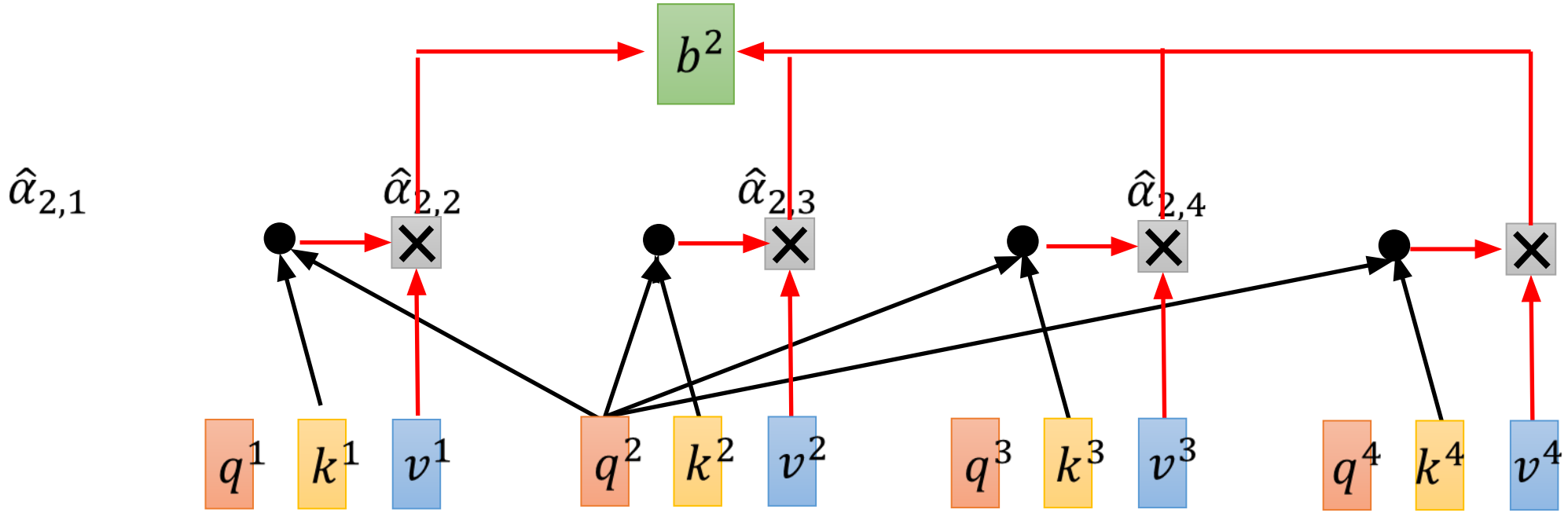
$$b^2 = \sum_i \hat{\alpha}_{2,i} v^i$$



Source: Hung-yi Lee Presentation

Self-Attention

$$b^2 = \sum_i \hat{\alpha}_{2,i} v^i$$

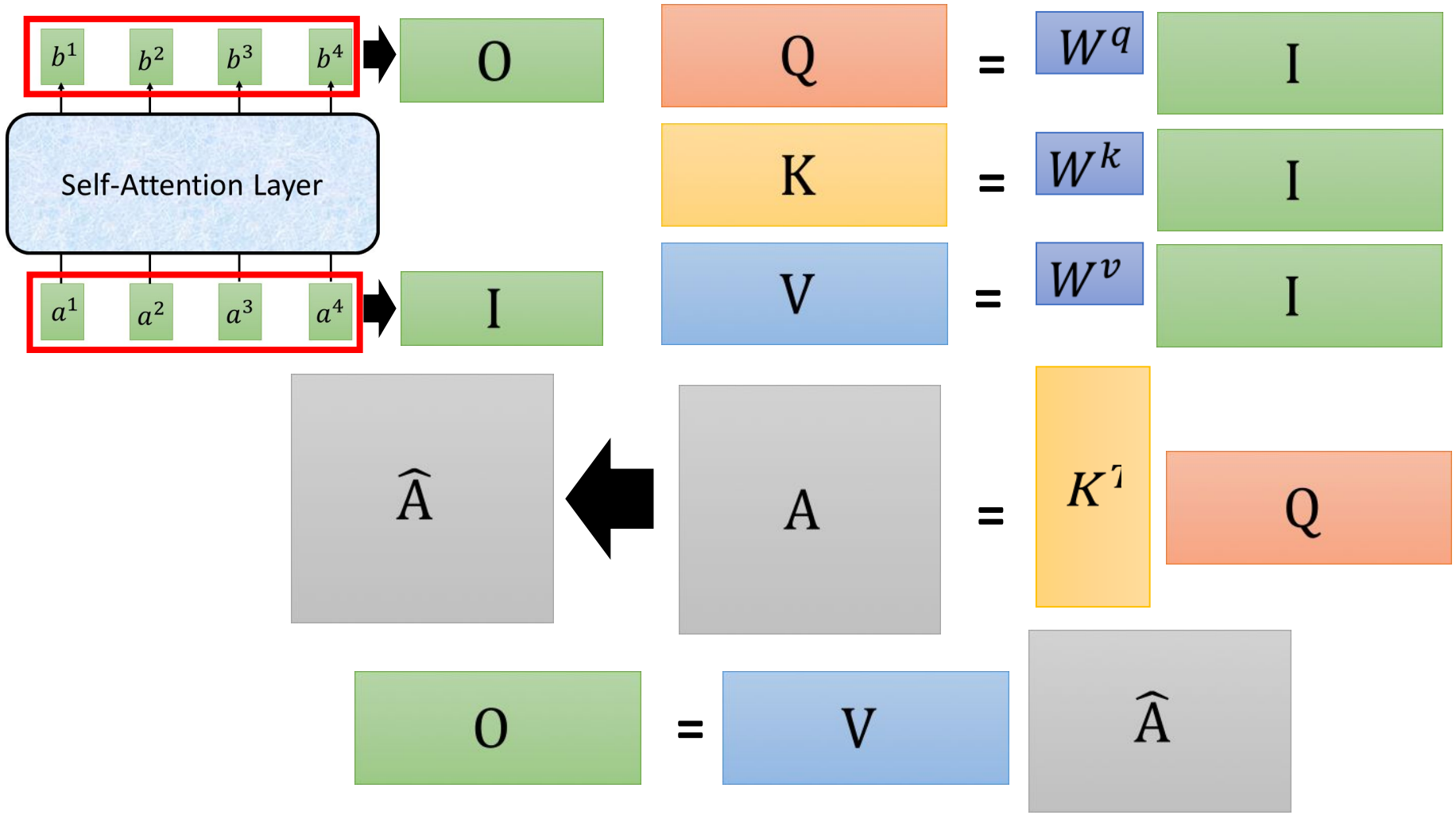


$$\begin{bmatrix} b^1 & b^2 & b^3 & b^4 \end{bmatrix} = \begin{bmatrix} v^1 & v^2 & v^3 & v^4 \end{bmatrix} \begin{bmatrix} \hat{\alpha}_{1,1} & \hat{\alpha}_{2,1} & \hat{\alpha}_{3,1} & \hat{\alpha}_{4,1} \\ \hat{\alpha}_{1,2} & \hat{\alpha}_{2,2} & \hat{\alpha}_{3,2} & \hat{\alpha}_{4,2} \\ \hat{\alpha}_{1,3} & \hat{\alpha}_{2,3} & \hat{\alpha}_{3,3} & \hat{\alpha}_{4,3} \\ \hat{\alpha}_{1,4} & \hat{\alpha}_{2,4} & \hat{\alpha}_{3,4} & \hat{\alpha}_{4,4} \end{bmatrix}$$

O V $\hat{A}$

Source: Hung-yi Lee Presentation

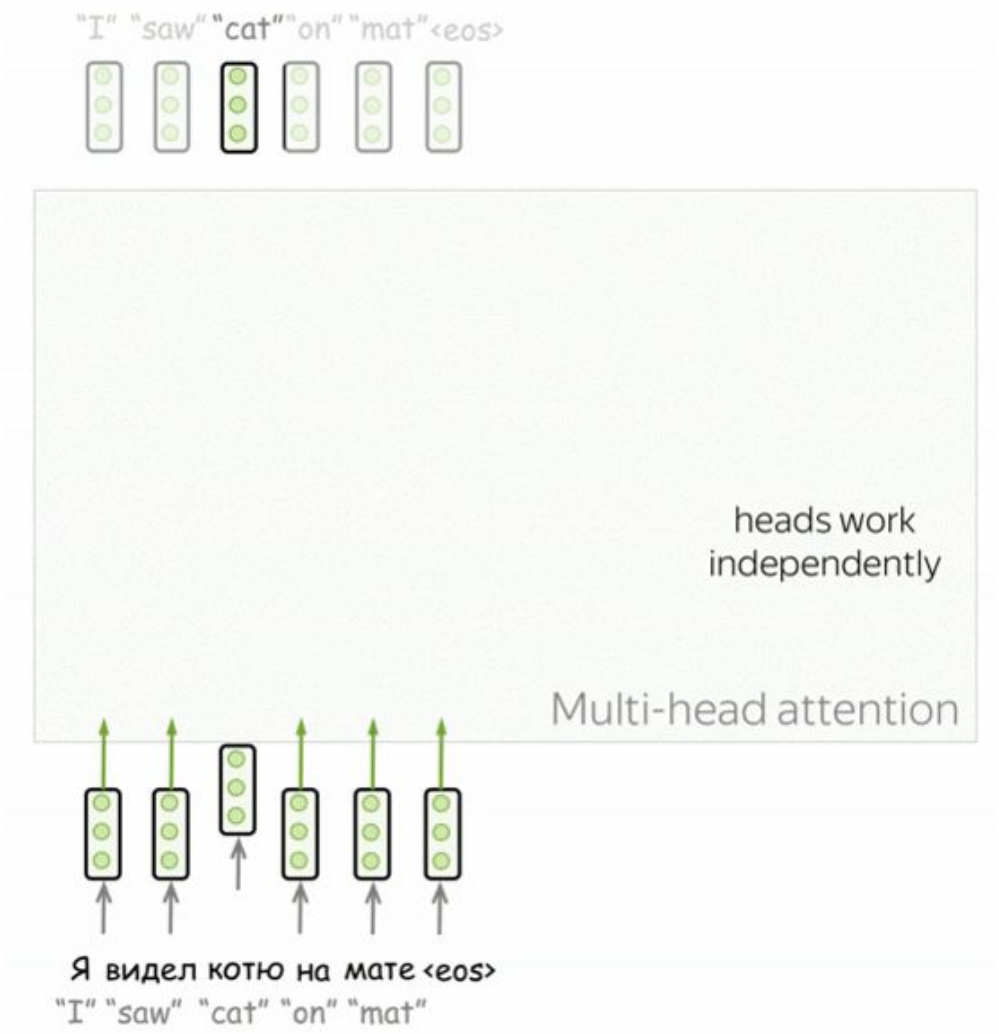
Self-Attention



It's a bunch of matrix multiplication, which can be accelerated by GPU

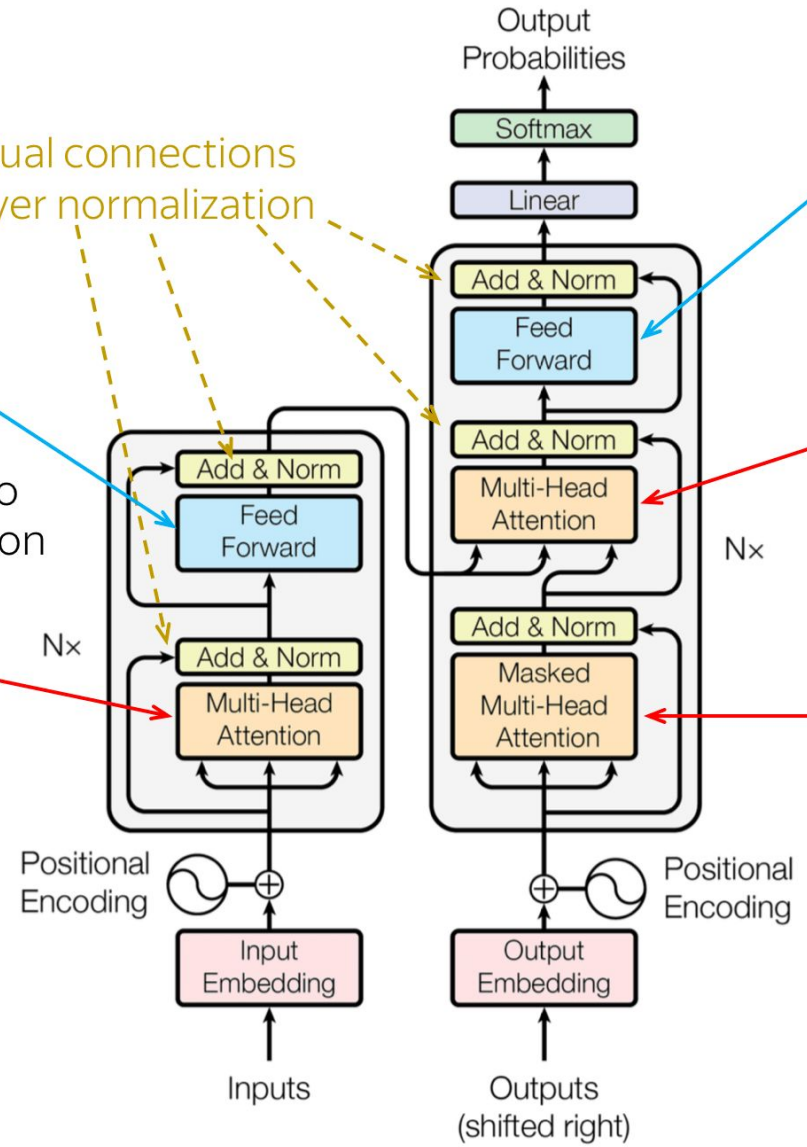
Source: Hung-yi Lee Presentation

Multi-head Attention



Slide credit: Lena Voita

Transformer



Residual connections and layer normalization

Feed-forward network: after taking information from other tokens, take a moment to think and process this information

Encoder self-attention: tokens look at each other

queries, keys, values are computed from encoder states

Feed-forward network: after taking information from other tokens, take a moment to think and process this information

Decoder-encoder attention: target token looks at the source

queries – from decoder states; keys and values from encoder states

Decoder self-attention (masked): tokens look at the previous tokens

queries, keys, values are computed from decoder states

Slide credit: Lena Voita

Flow of Information

Consider the following sentences and their French translations:

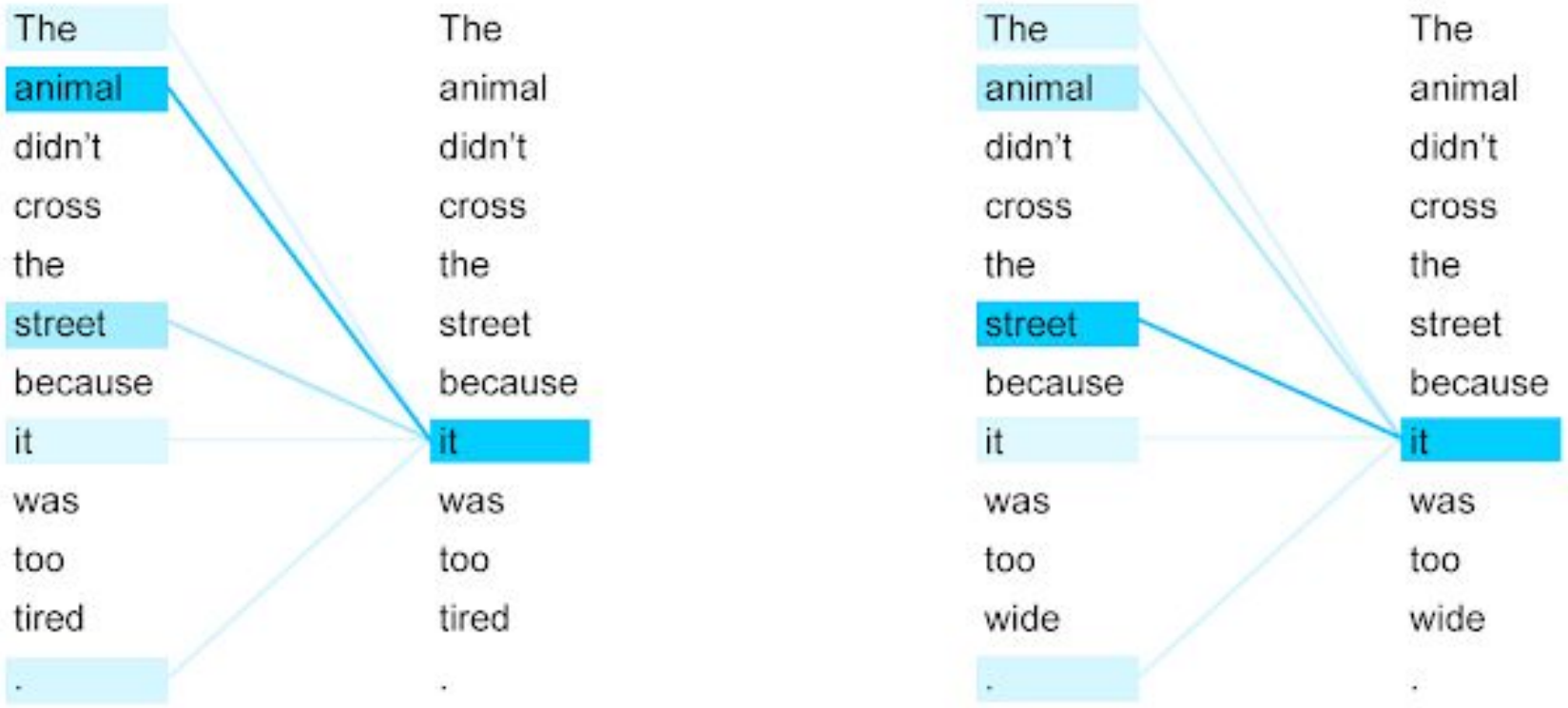
The animal didn't cross the street because it was too tired.
L'animal n'a pas traversé la rue parce qu'il était trop fatigué.

The animal didn't cross the street because it was too wide.
L'animal n'a pas traversé la rue parce qu'elle était trop large.

(Coreference resolution)

Transformer translates both of these sentences to French correctly

Attention Visualization

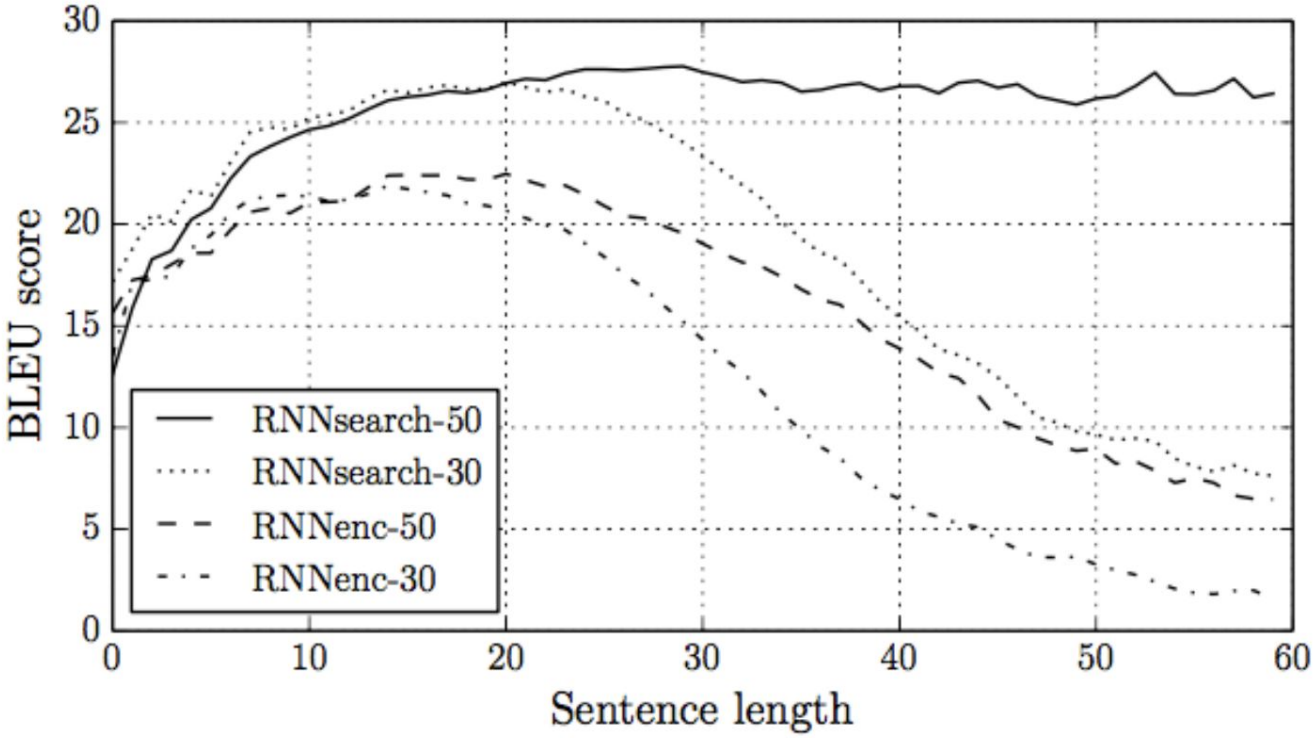


The encoder self-attention distribution for the word “it” from the 5th to the 6th layer of a Transformer trained on English to French translation.

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

Attention-Based Machine Translation

- The attention-based translation model does much better than the encoder/decoder model on long sentences.



Source: Neural Machine Translation by Jointly learning to align and translate, (Bahdanau et al), ICLR 2015

GPT

Google's BERT

- Bidirectional Encoder Representations from Transformers (BERT)

- The BERT framework, a new language representation model from Google AI, uses pre-training and fine-tuning to create state-of-the-art models for a wide range of tasks.

- These tasks include:
  - Question answering systems
  - Sentiment analysis
  - Language inference (http://nlpprogress.com/english/natural_language_inference.html)

BERT's Model Architecture

- BERT consists of a simple stack of transformer blocks
- This stack is *pre-trained* on a large general-domain corpus consisting of 800M words from English books (modern work, from unpublished authors), and 2.5B words of text from English Wikipedia articles (without markup).
- BERT uses a multi-layer bidirectional Transformer encoder.
- Its self-attention layer performs self-attention in both directions.

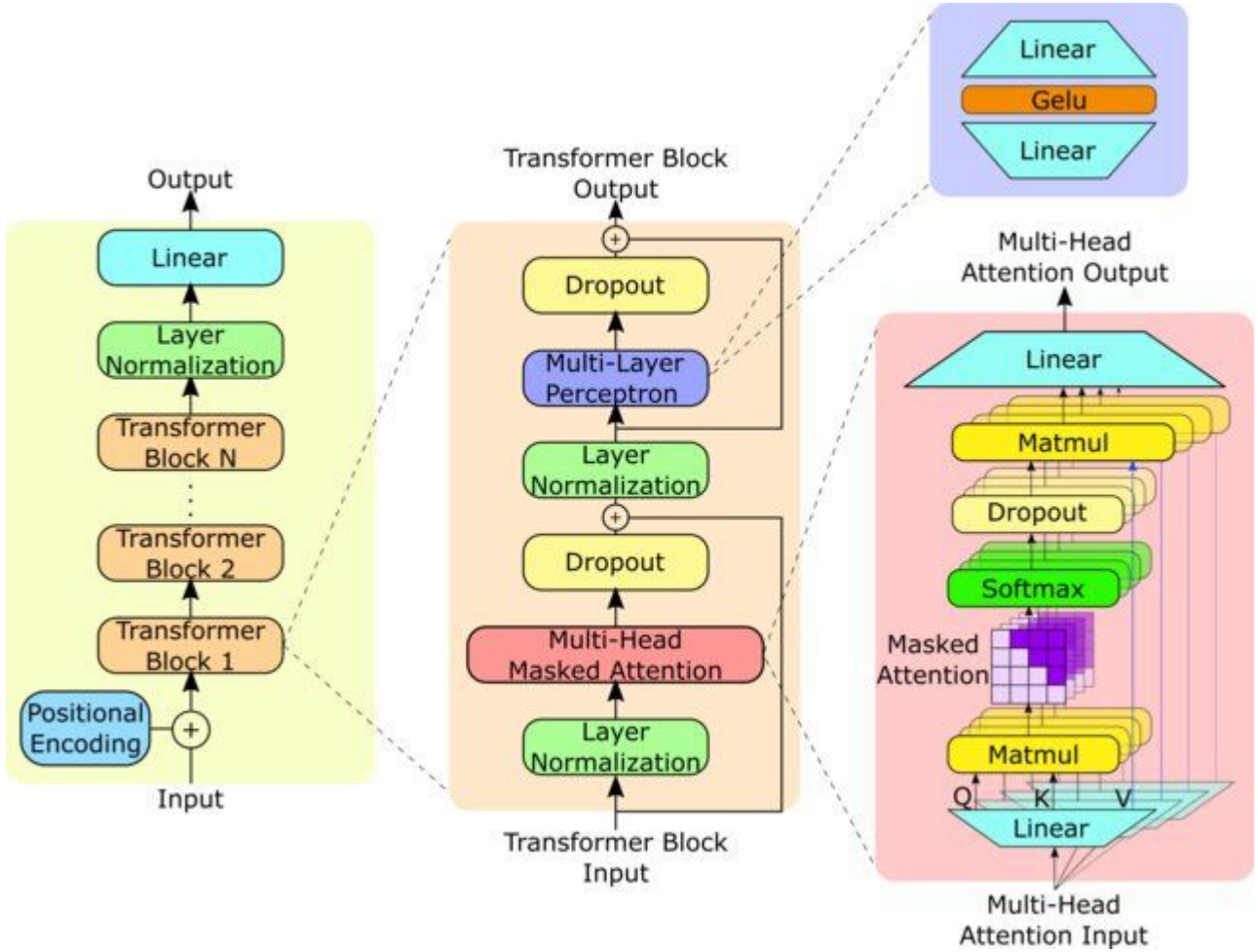
Google's BERT

- For pretraining, BERT uses the following hyper-parameters:
  - **Sequence length (single example):** 256
  - **Batch size:** 512
  - **Training steps:** 1,000,000 (Approximately 40 epochs)
  - **Optimizer:** Adam
  - **Learning rate:** 1e-4
  - **Learning rate schedule:** Warmup for 10,000 steps, then linear decay
  - **Dropout:** 0.1
  - **Activation function:** gelu (Gaussian Error Linear Unit)
  - **The training** took 4 days on 16 cloud TPUs (64 TPU chips).

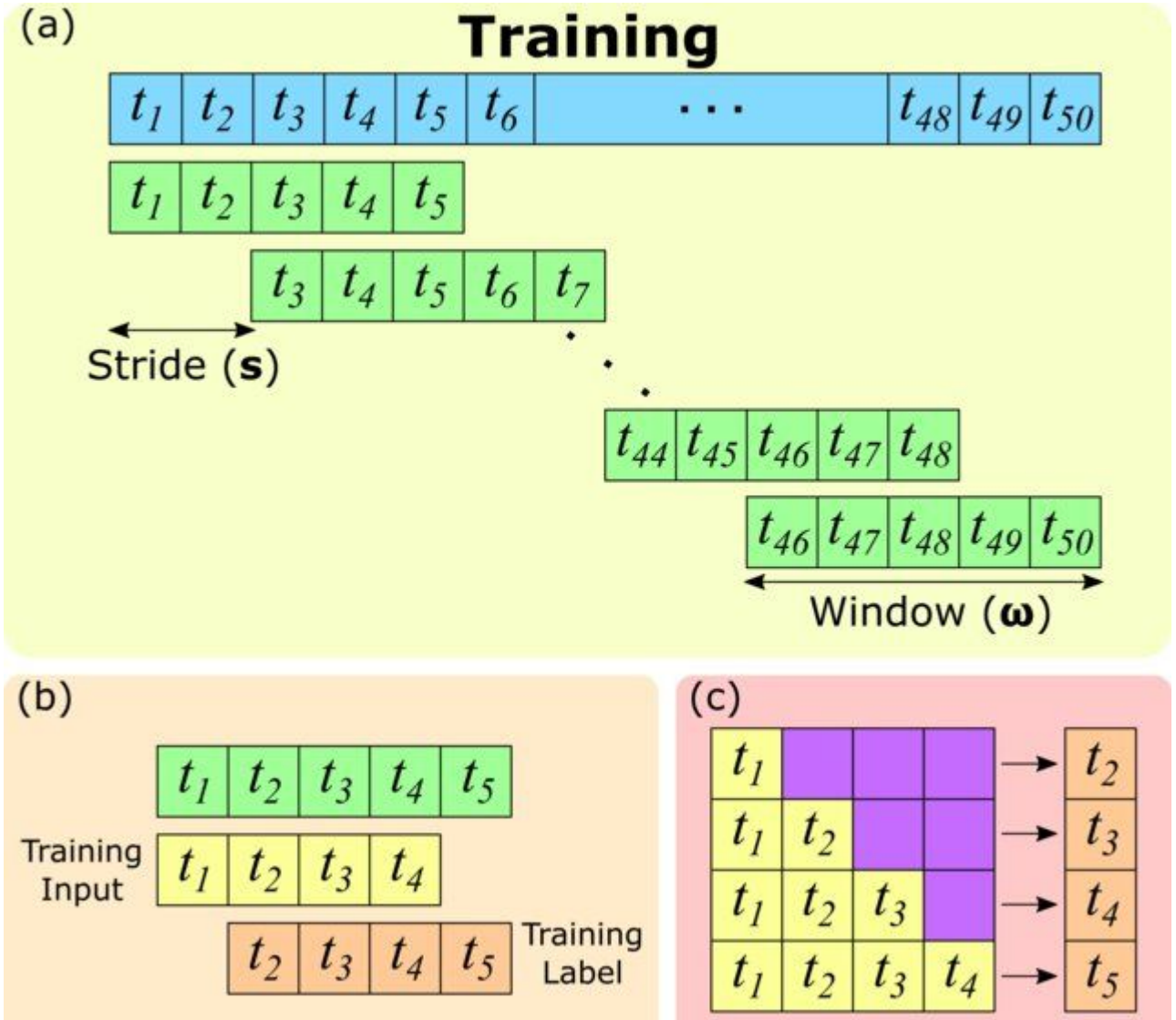
Reference

- This lecture is based on the following resources + other web resources.
  - Chapter 10: http://www.deeplearningbook.org/contents/rnn.html
  - https://arxiv.org/pdf/1703.03091.pdf
  - https://lena-voita.github.io/nlp_course/seq2seq_and_attention.html
  - http://karpathy.github.io/2015/05/21/rnn-effectiveness/
  - http://introtodeeplearning.com/materials/2018_6S191_Lecture2.pdf
  - RNN and LSTM slides: http://bit.ly/2sO00ZC
  - https://medium.com/@jianqiangma/all-about-recurrent-neural-networks-9e5ae2936f6e
  - https://www.bioinf.jku.at/publications/older/2604.pdf
  - CSC2535 2013: Advanced Machine Learning, Lecture 10 Recurrent neural networks by Geoffrey Hinton
  - http://colah.github.io/posts/2015-08-Understanding-LSTMs/
  - http://www.peterbloem.nl/blog/transformers
  - https://www.analyticsvidhya.com/blog/2019/06/understanding-transformers-nlp-state-of-the-art-models/
  - https://arxiv.org/pdf/1706.03762.pdf
  - https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1184/lectures/lecture12.pdf
  - http://nlp.seas.harvard.edu/2018/04/03/attention.html

GPT 2


Output
Linear
Layer Normalization
Transformer Block N
Transformer Block 2
Transformer Block 1
Positional Encoding
Input
Transformer Block Output
Dropout
Multi-Layer Perceptron
Layer Normalization
Dropout
Multi-Head Masked Attention
Layer Normalization
Transformer Block Input
Linear
Gelu
Linear
Multi-Head Attention Output
Linear
Matmul
Dropout
Softmax
Masked Attention
Matmul
Q
K
V
Linear
Multi-Head Attention Input

Striding


(a)
Training
t_1 t_2 t_3 t_4 t_5 t_6 . . . t_{48} t_{49} t_{50}
t_1 t_2 t_3 t_4 t_5
t_3 t_4 t_5 t_6 t_7
Stride (s)
t_{44} t_{45} t_{46} t_{47} t_{48}
t_{46} t_{47} t_{48} t_{49} t_{50}
Window (ω)
(b)
t_1 t_2 t_3 t_4 t_5
Training Input
t_1 t_2 t_3 t_4
t_2 t_3 t_4 t_5
Training Label
(c)
t_1 → t_2
t_1 t_2 → t_3
t_1 t_2 t_3 → t_4
t_1 t_2 t_3 t_4 → t_5

GPT 3

Bigger + fine-tuning (RL or supervised based on human feedback)

GPT 4

Bigger yet
8 submodels

Fine-tuning:
Reinforcement learning through human feedback RLHF -> reward agent
reward + “RL” -> fine-tuning
Also supervised learning to improve performance on rules and agents

Thank You

Appendix

Positional Encoding

- Transformer, all words in the input sentence are processed simultaneously – there is no inherent "word ordering" and thus no inherent position information.
- The authors of the Transformer propose adding a "positional encoding" to address this problem.
- The positional encoding makes it possible for the Transformer to use information about word order.
- The positional encoding uses a sequence of real-valued vectors that capture ordering information.
- Each word in a sentence is summed with a particular positional encoding vector based on its position within the sentence

https://glassboxmedicine.com/2019/08/15/the-transformer-attention-is-all-you-need/

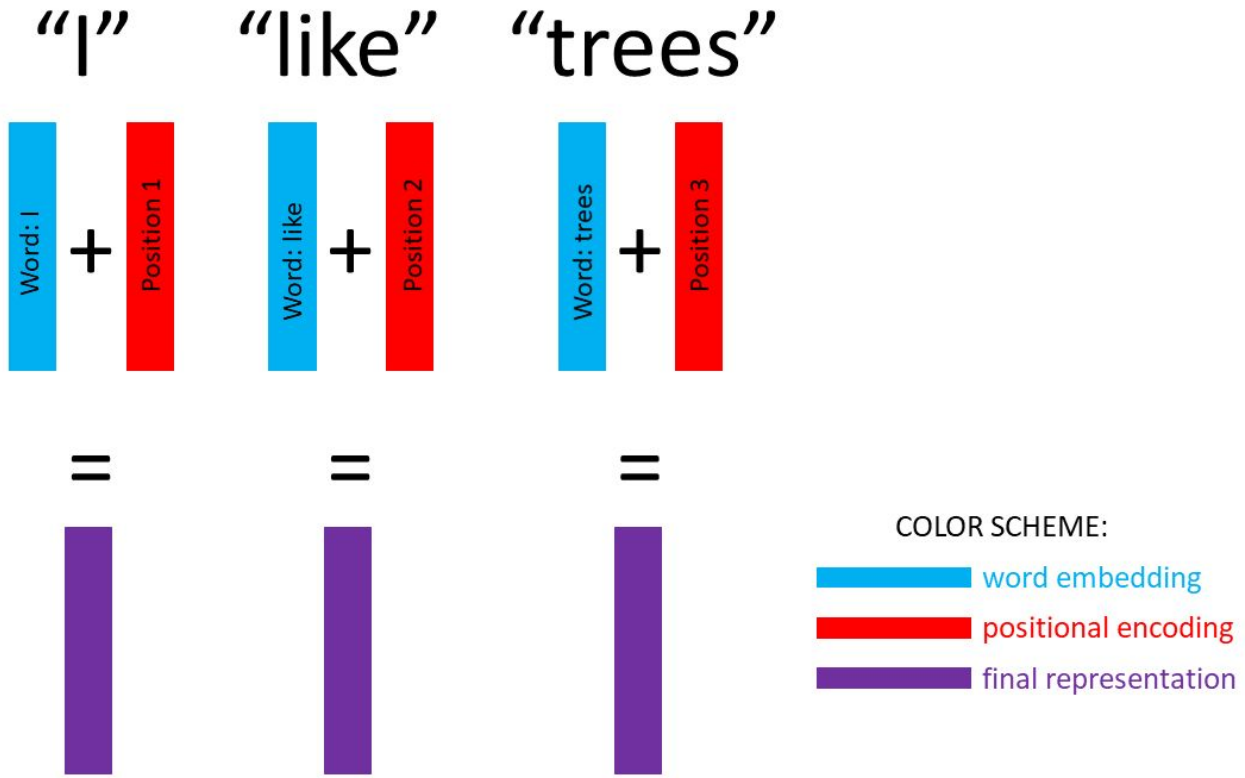
Positional Encoding Vectors

- **Option 1:** learning the positional encoding vectors (requires trainable parameters)
- **Option 2:** calculating the positional encoding vectors using an equation (requires no trainable parameters)

- They found the performance of both options was similar, so they went with option 2.

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

Positional Encoding



https://glassboxmedicine.com/2019/08/15/the-transformer-attention-is-all-you-need/