Introduction to Transformer (Part I)

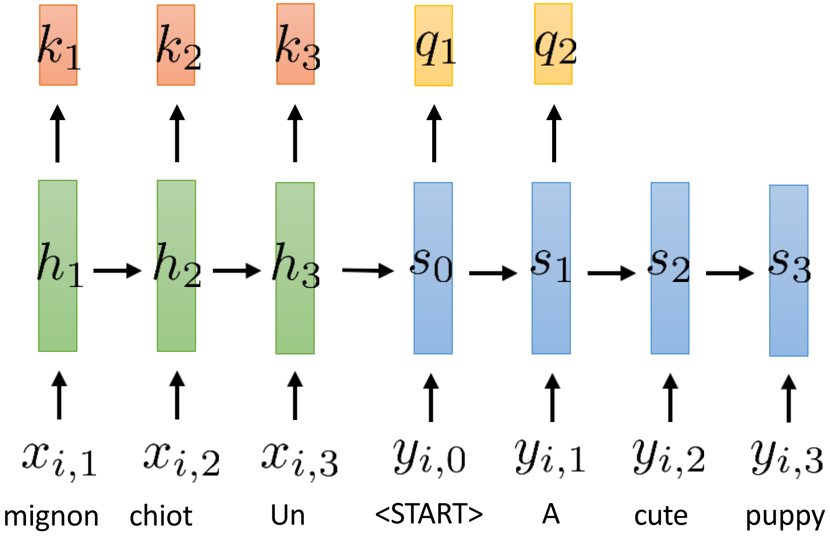
Tanmoy Chakraborty
Associate Professor, IIT Delhi
https://tanmoychak.com/

Slides are adopted from Sergey Levine

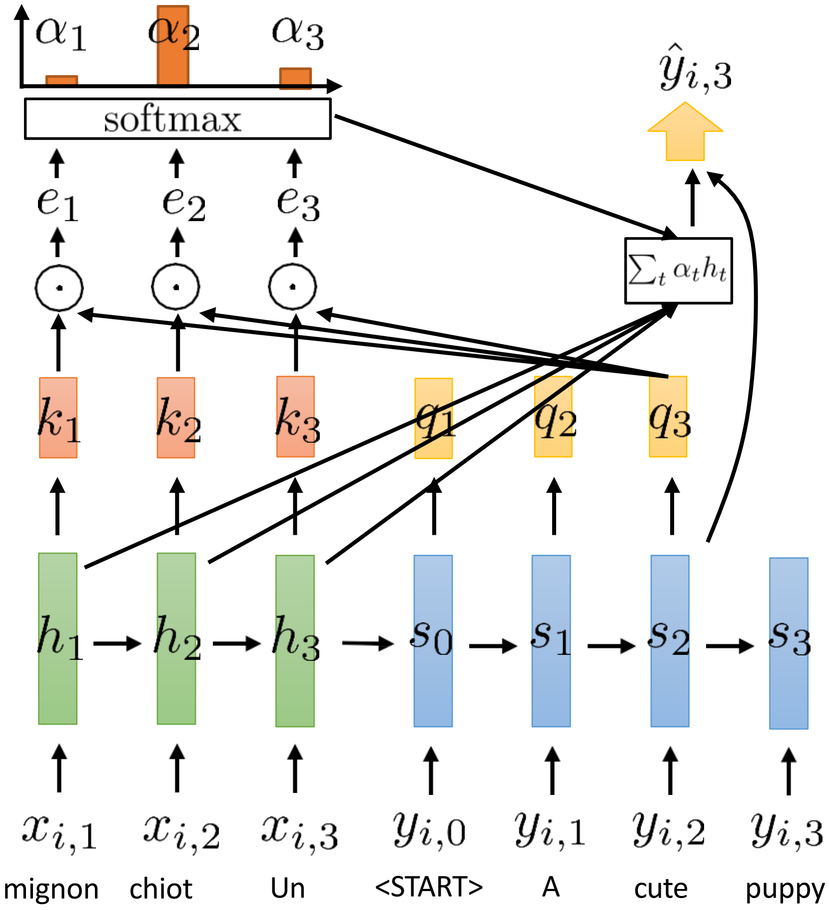
Is Attention All We Need?

Slides by Sergey Levine

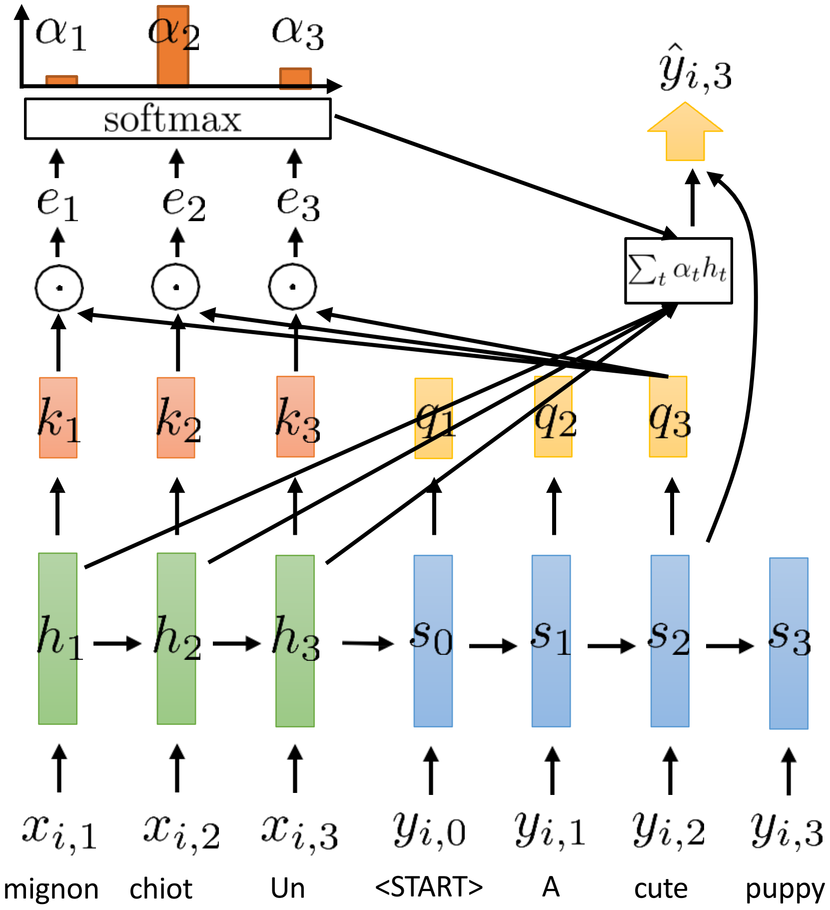
Recap: Attention



Recap: Attention



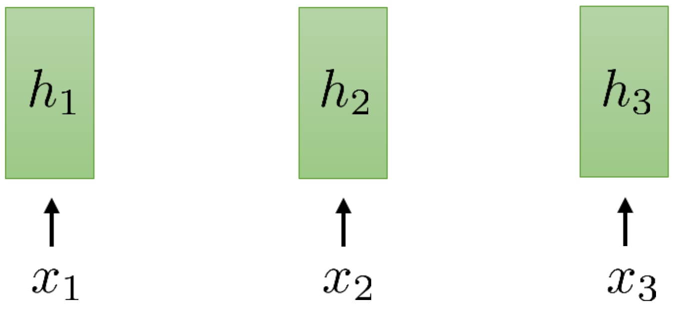
Recap: Attention



- If we have **attention**, do we even need recurrent connections?
- Can we transform our RNN into a **purely attention-based model**?
- Attention can access all time steps simultaneously, potentially doing everything that recurrence can, and even more. However, this approach presents some challenges:

The encoder lacks temporal dependencies at all!

Self-Attention



this is *not* a recurrent model!

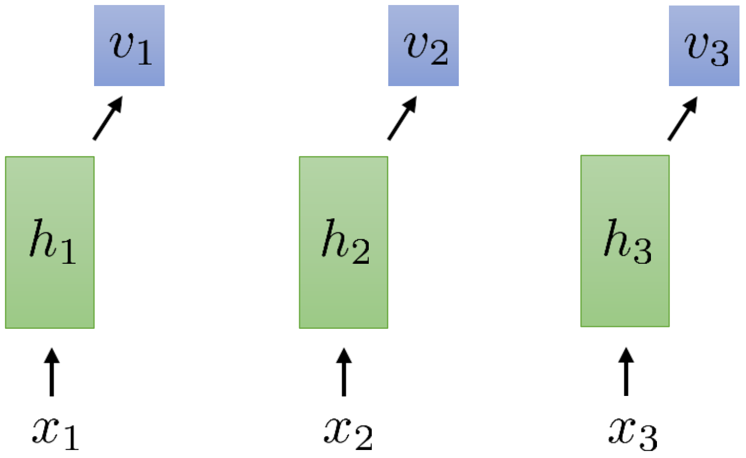
but still weight sharing:

$$h_t = \sigma(W x_t + b)$$

shared weights at all time steps

(or any other nonlinear function)

Self-Attention



this is *not* a recurrent model!
but still weight sharing:

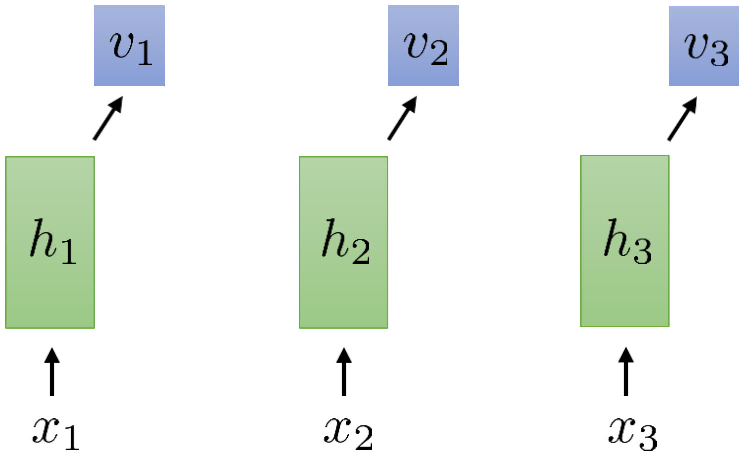
$$h_t = \sigma(W x_t + b)$$

shared weights at all time steps

(or any other nonlinear function)

Self-Attention

$v_t = v(h_t)$ before just had $v(h_t) = h_t$, now e.g. $v(h_t) = W_v h_t$



this is *not* a recurrent model!

but still weight sharing:

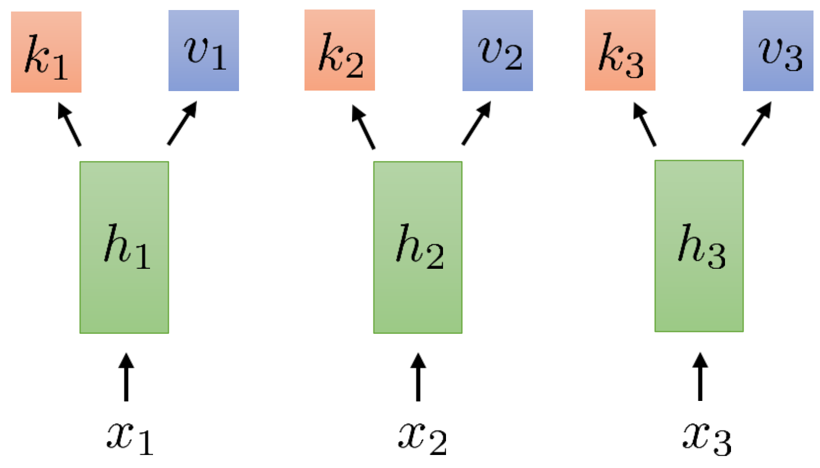
$$h_t = \sigma(W x_t + b)$$

shared weights at all time steps

(or any other nonlinear function)

Self-Attention

$v_t = v(h_t)$ before just had $v(h_t) = h_t$, now e.g. $v(h_t) = W_v h_t$



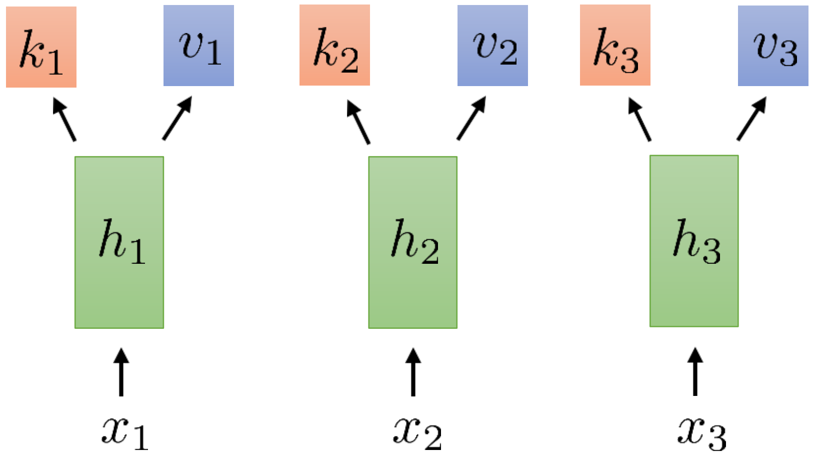
this is *not* a recurrent model!
but still weight sharing:

$$h_t = \sigma(W x_t + b)$$

shared weights at all time steps

(or any other nonlinear function)

Self-Attention



$v_t = v(h_t)$ before just had $v(h_t) = h_t$, now e.g. $v(h_t) = W_v h_t$

$k_t = k(h_t)$ (just like before) e.g., $k_t = W_k h_t$

this is *not* a recurrent model!

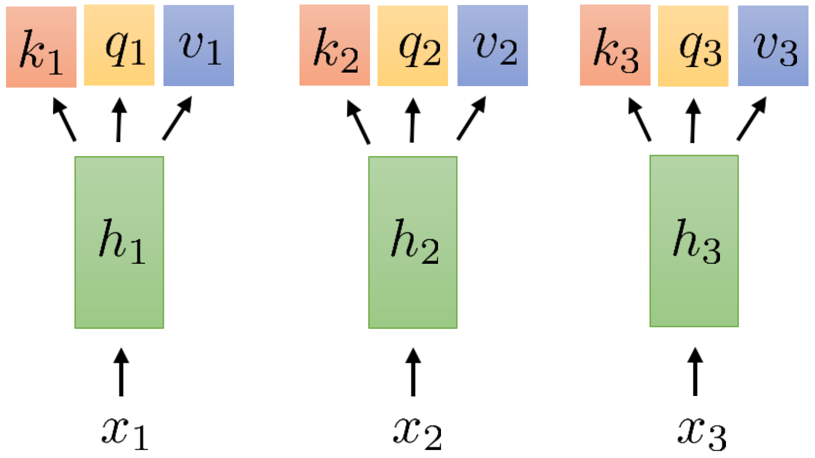
but still weight sharing:

$$h_t = \sigma(Wx_t + b)$$

shared weights at all time steps

(or any other nonlinear function)

Self-Attention



$v_t = v(h_t)$ before just had $v(h_t) = h_t$, now e.g. $v(h_t) = W_v h_t$

$k_t = k(h_t)$ (just like before) e.g., $k_t = W_k h_t$

this is *not* a recurrent model!

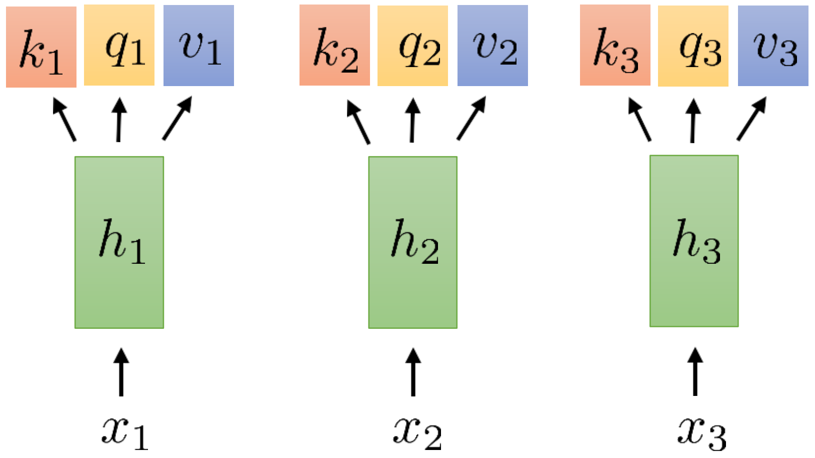
but still weight sharing:

$$h_t = \sigma(W x_t + b)$$

shared weights at all time steps

(or any other nonlinear function)

Self-Attention



$v_t = v(h_t)$ before just had $v(h_t) = h_t$, now e.g. $v(h_t) = W_v h_t$

$k_t = k(h_t)$ (just like before) e.g., $k_t = W_k h_t$

$q_t = q(h_t)$ e.g., $q_t = W_q h_t$

this is *not* a recurrent model!

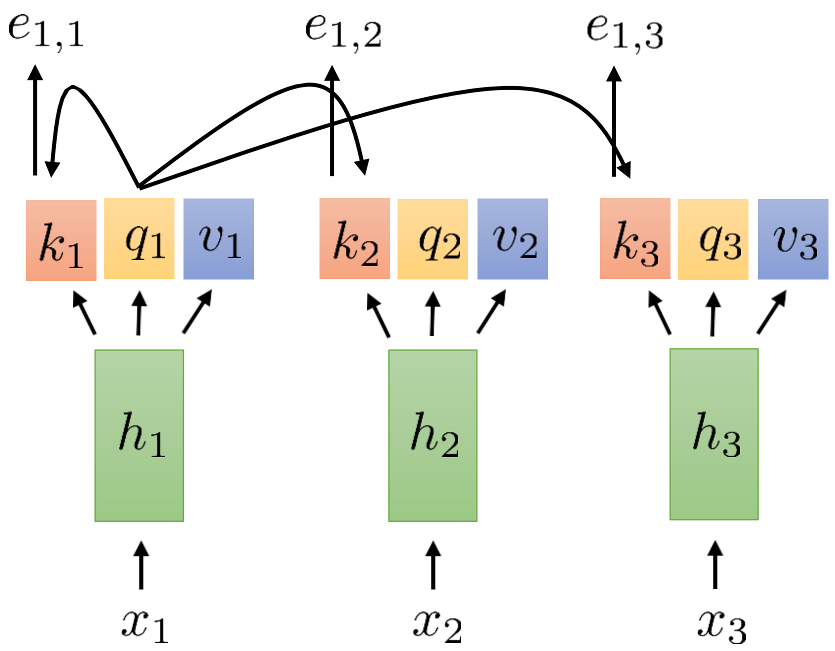
but still weight sharing:

$$h_t = \sigma(W x_t + b)$$

shared weights at all time steps

(or any other nonlinear function)

Self-Attention



$v_t = v(h_t)$ before just had $v(h_t) = h_t$, now e.g. $v(h_t) = W_v h_t$

$k_t = k(h_t)$ (just like before) e.g., $k_t = W_k h_t$

$q_t = q(h_t)$ e.g., $q_t = W_q h_t$

this is *not* a recurrent model!

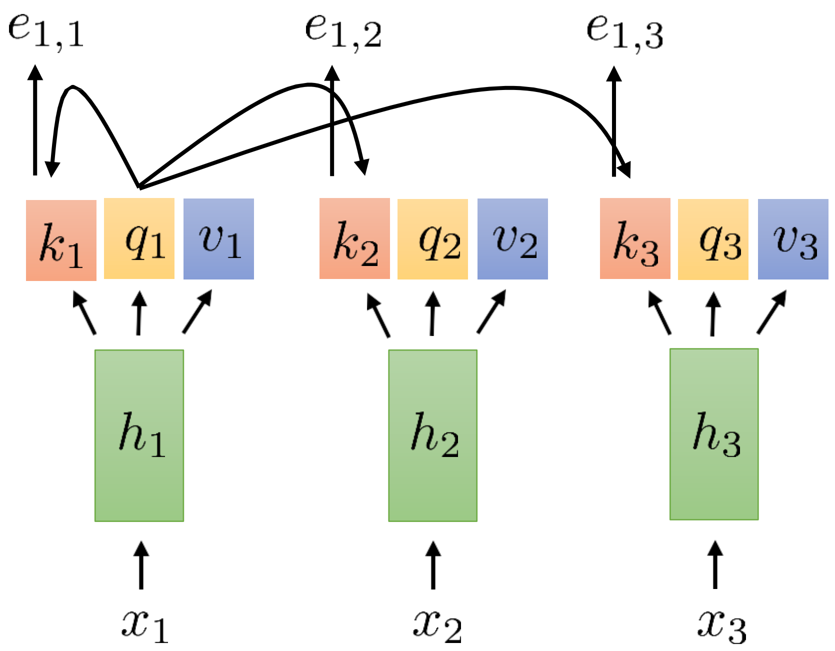
but still weight sharing:

$$h_t = \sigma(W x_t + b)$$

shared weights at all time steps

(or any other nonlinear function)

Self-Attention



$e_{l,t} = q_l \cdot k_t$

$v_t = v(h_t)$ before just had $v(h_t) = h_t$, now e.g. $v(h_t) = W_v h_t$

$k_t = k(h_t)$ (just like before) e.g., $k_t = W_k h_t$

$q_t = q(h_t)$ e.g., $q_t = W_q h_t$

this is *not* a recurrent model!

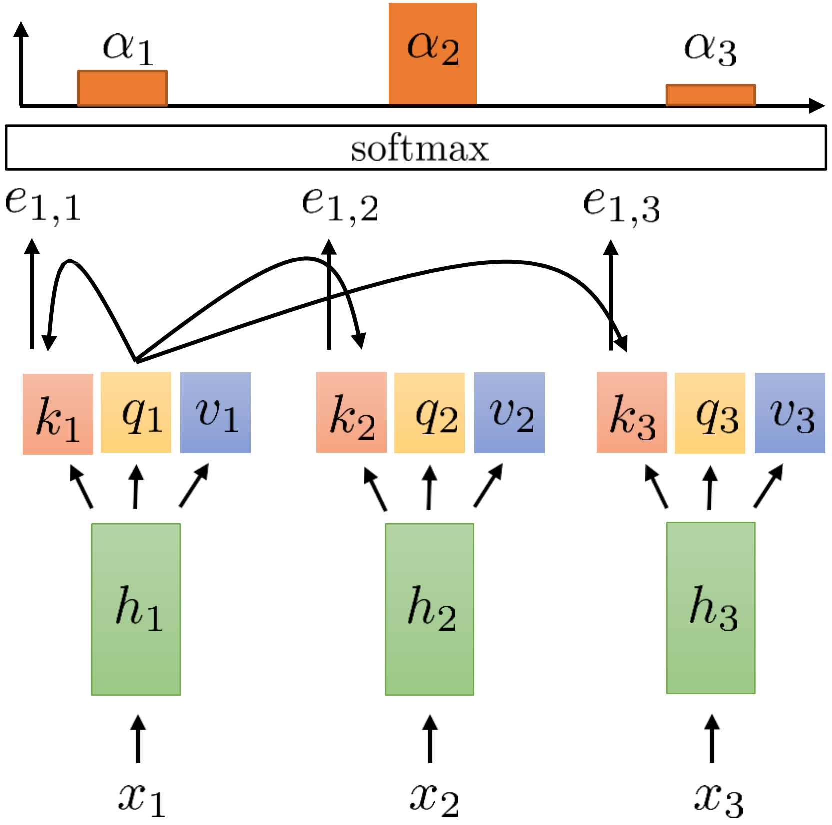
but still weight sharing:

$$h_t = \sigma(W x_t + b)$$

shared weights at all time steps

(or any other nonlinear function)

Self-Attention



$e_{l,t} = q_l \cdot k_t$

$v_t = v(h_t)$ before just had $v(h_t) = h_t$, now e.g. $v(h_t) = W_v h_t$

$k_t = k(h_t)$ (just like before) e.g., $k_t = W_k h_t$

$q_t = q(h_t)$ e.g., $q_t = W_q h_t$

this is *not* a recurrent model!

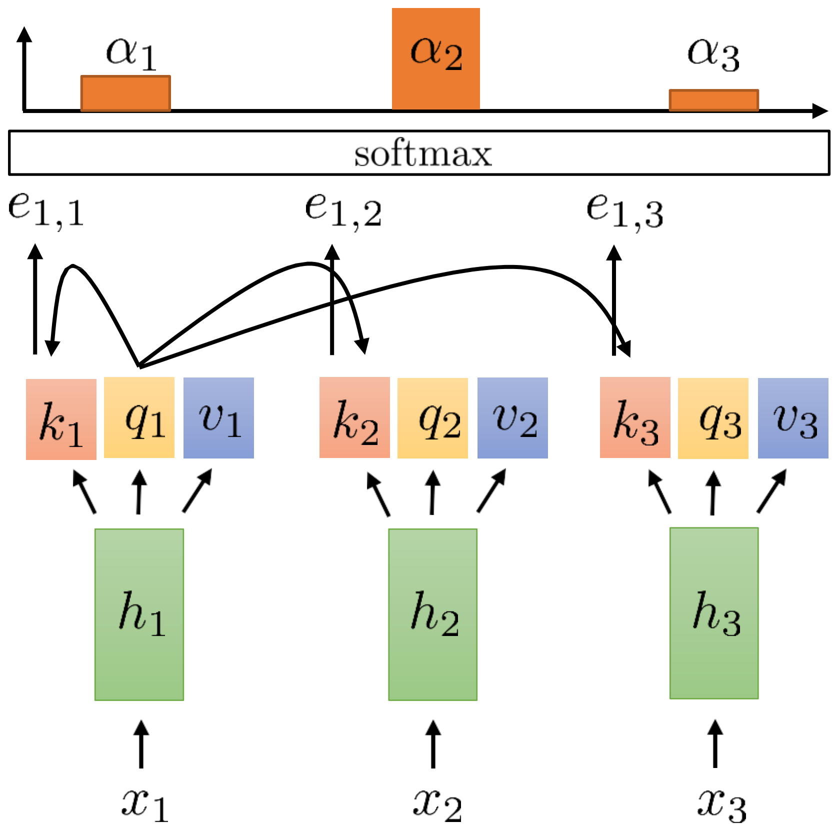
but still weight sharing:

$$h_t = \sigma(W x_t + b)$$

shared weights at all time steps

(or any other nonlinear function)

Self-Attention



$$\alpha_{l,t} = \exp(e_{l,t}) / \sum_{t'} \exp(e_{l,t'})$$

$$e_{l,t} = q_l \cdot k_t$$

$v_t = v(h_t)$ before just had $v(h_t) = h_t$, now e.g. $v(h_t) = W_v h_t$

$k_t = k(h_t)$ (just like before) e.g., $k_t = W_k h_t$

$q_t = q(h_t)$ e.g., $q_t = W_q h_t$

this is *not* a recurrent model!

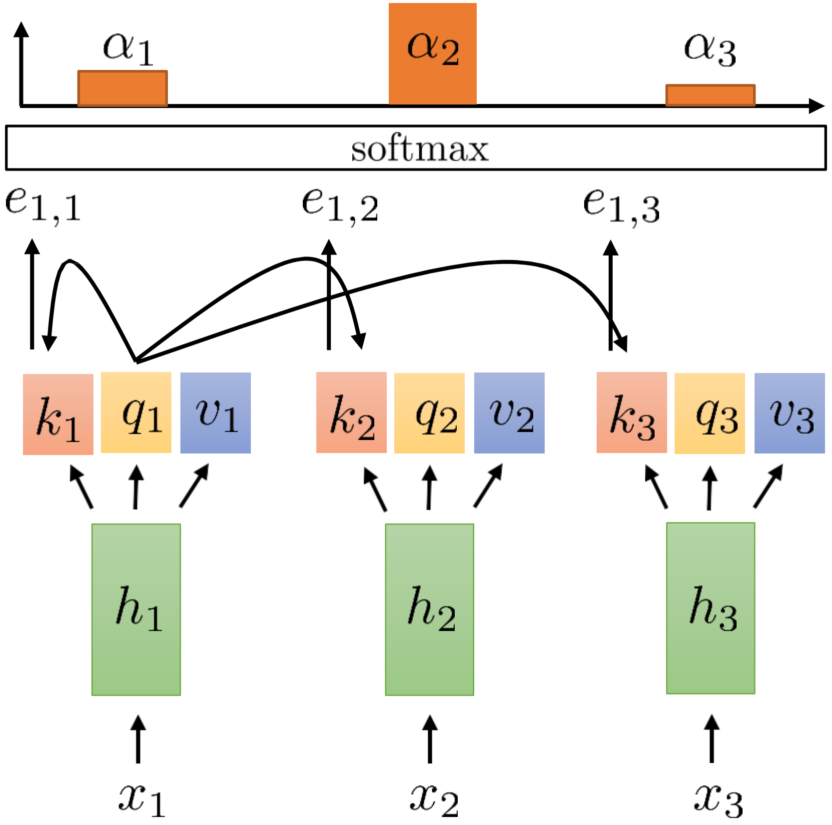
but still weight sharing:

$$h_t = \sigma(W x_t + b)$$

shared weights at all time steps

(or any other nonlinear function)

Self-Attention



$$\alpha_{l,t} = \exp(e_{l,t}) / \sum_{t'} \exp(e_{l,t'})$$

$$e_{l,t} = q_l \cdot k_t$$

$v_t = v(h_t)$ before just had $v(h_t) = h_t$, now e.g. $v(h_t) = W_v h_t$

$k_t = k(h_t)$ (just like before) e.g., $k_t = W_k h_t$

$q_t = q(h_t)$ e.g., $q_t = W_q h_t$

this is *not* a recurrent model!

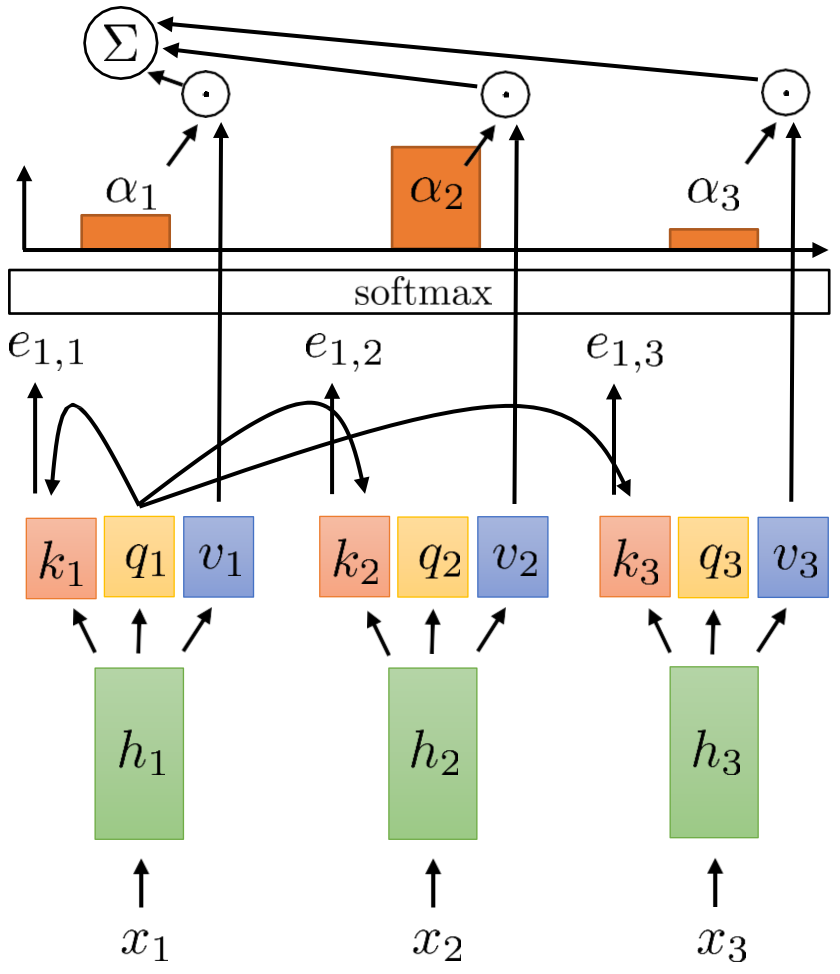
but still weight sharing:

$$h_t = \sigma(W x_t + b)$$

shared weights at all time steps

(or any other nonlinear function)

Self-Attention



$$\alpha_{l,t} = \exp(e_{l,t}) / \sum_{t'} \exp(e_{l,t'})$$

$$e_{l,t} = q_l \cdot k_t$$

$v_t = v(h_t)$ before just had $v(h_t) = h_t$, now e.g. $v(h_t) = W_v h_t$

$k_t = k(h_t)$ (just like before) e.g., $k_t = W_k h_t$

$q_t = q(h_t)$ e.g., $q_t = W_q h_t$

this is *not* a recurrent model!

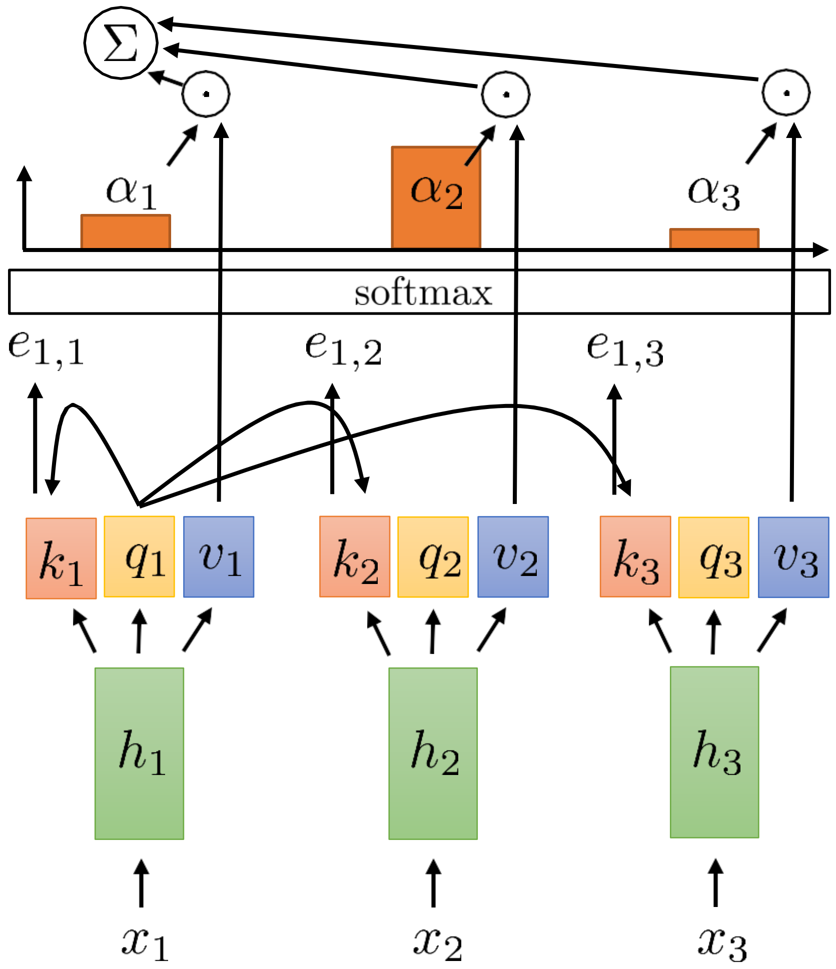
but still weight sharing:

$$h_t = \sigma(W x_t + b)$$

shared weights at all time steps

(or any other nonlinear function)

Self-Attention



$$a_l = \sum_t \alpha_{l,t} v_t$$

$$\alpha_{l,t} = \exp(e_{l,t}) / \sum_{t'} \exp(e_{l,t'})$$

$$e_{l,t} = q_l \cdot k_t$$

$v_t = v(h_t)$ before just had $v(h_t) = h_t$, now e.g. $v(h_t) = W_v h_t$

$k_t = k(h_t)$ (just like before) e.g., $k_t = W_k h_t$

$q_t = q(h_t)$ e.g., $q_t = W_q h_t$

this is *not* a recurrent model!

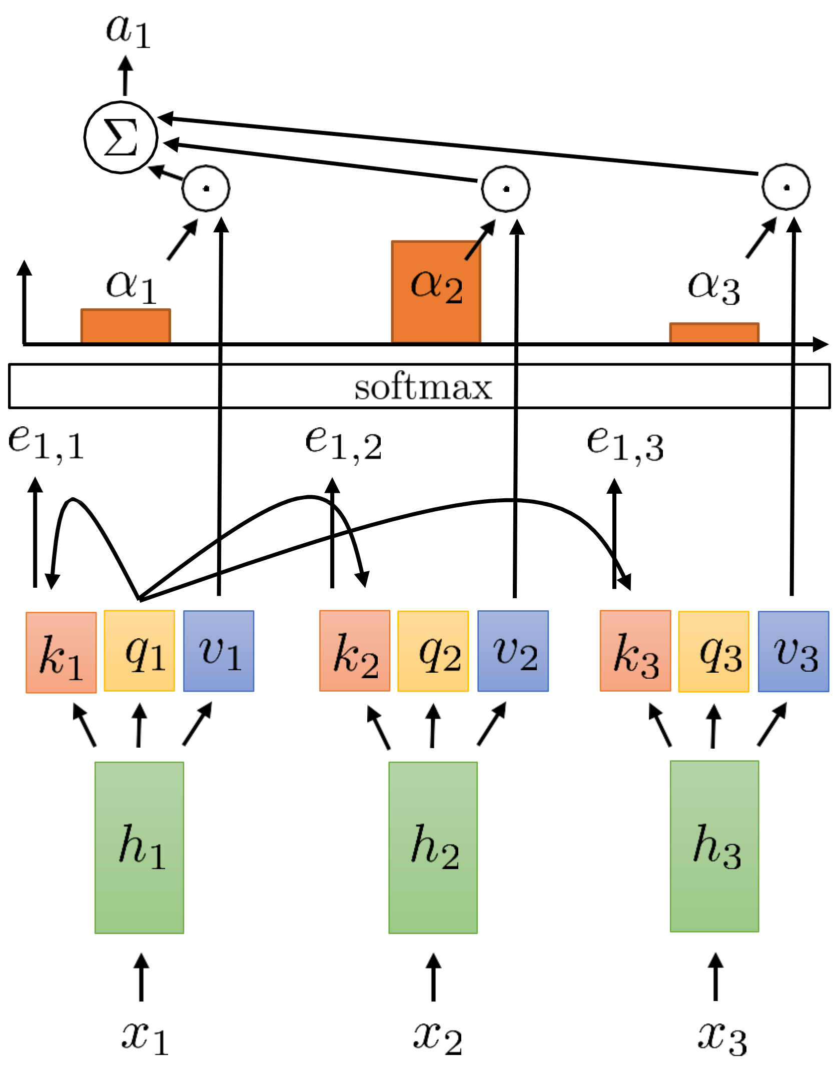
but still weight sharing:

$$h_t = \sigma(W x_t + b)$$

shared weights at all time steps

(or any other nonlinear function)

Self-Attention



$$a_l = \sum_t \alpha_{l,t} v_t$$

$$\alpha_{l,t} = \exp(e_{l,t}) / \sum_{t'} \exp(e_{l,t'})$$

$$e_{l,t} = q_l \cdot k_t$$

$v_t = v(h_t)$ before just had $v(h_t) = h_t$, now e.g. $v(h_t) = W_v h_t$

$k_t = k(h_t)$ (just like before) e.g., $k_t = W_k h_t$

$q_t = q(h_t)$ e.g., $q_t = W_q h_t$

this is *not* a recurrent model!

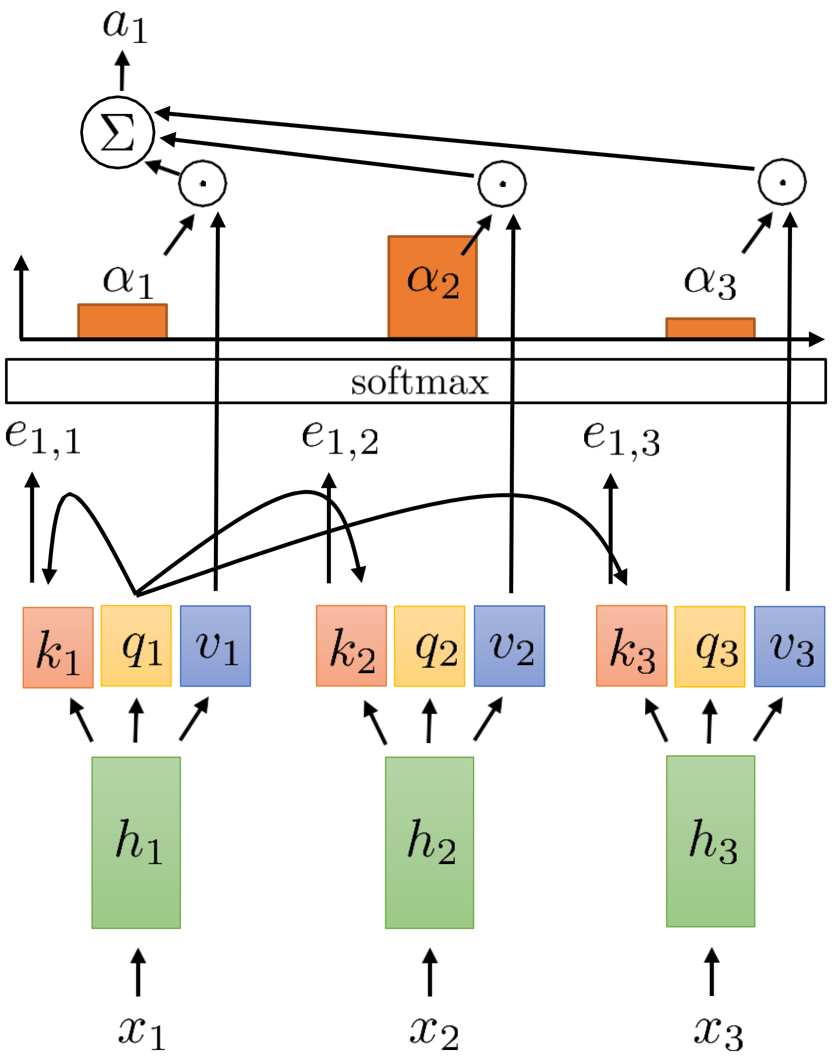
but still weight sharing:

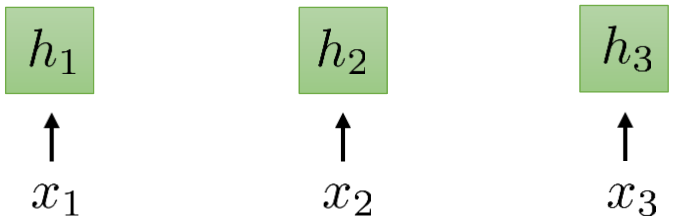
$$h_t = \sigma(W x_t + b)$$

shared weights at all time steps

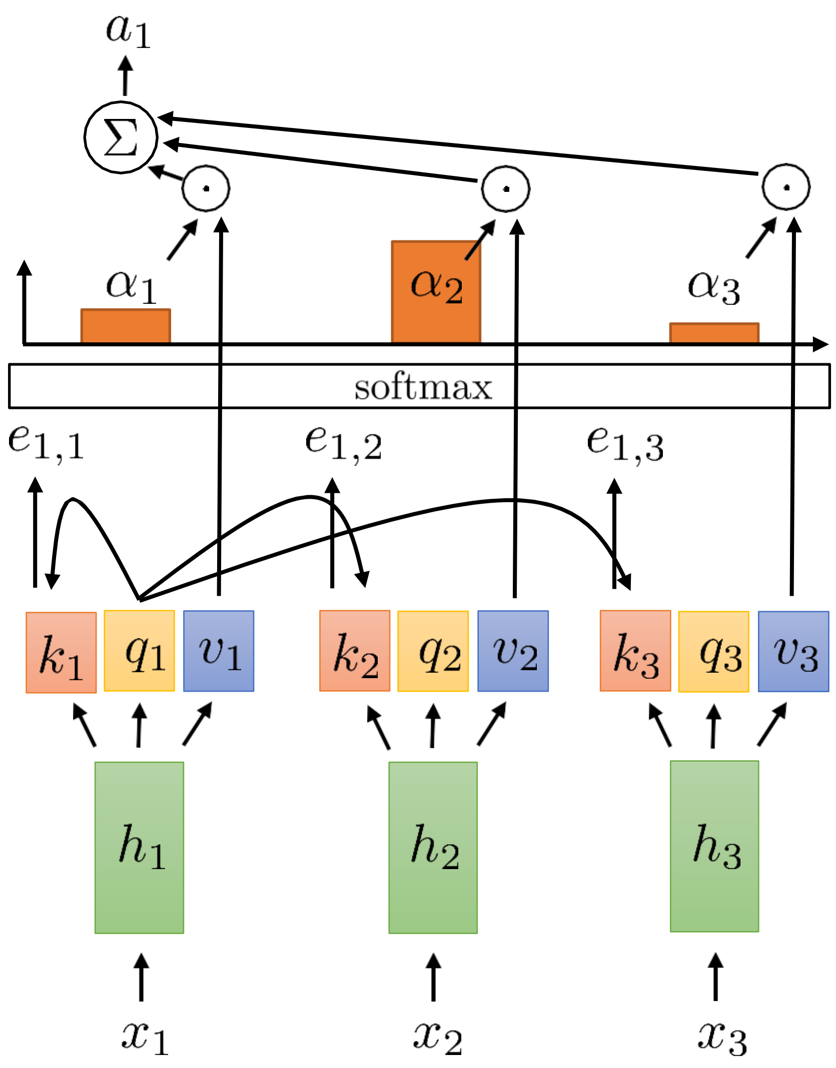
(or any other nonlinear function)

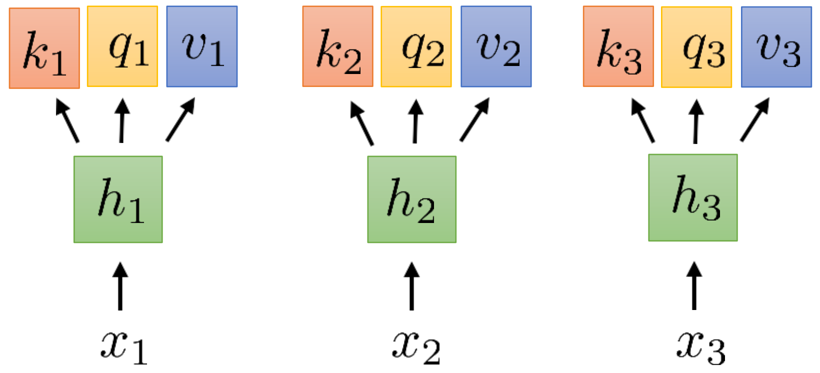
Self-Attention



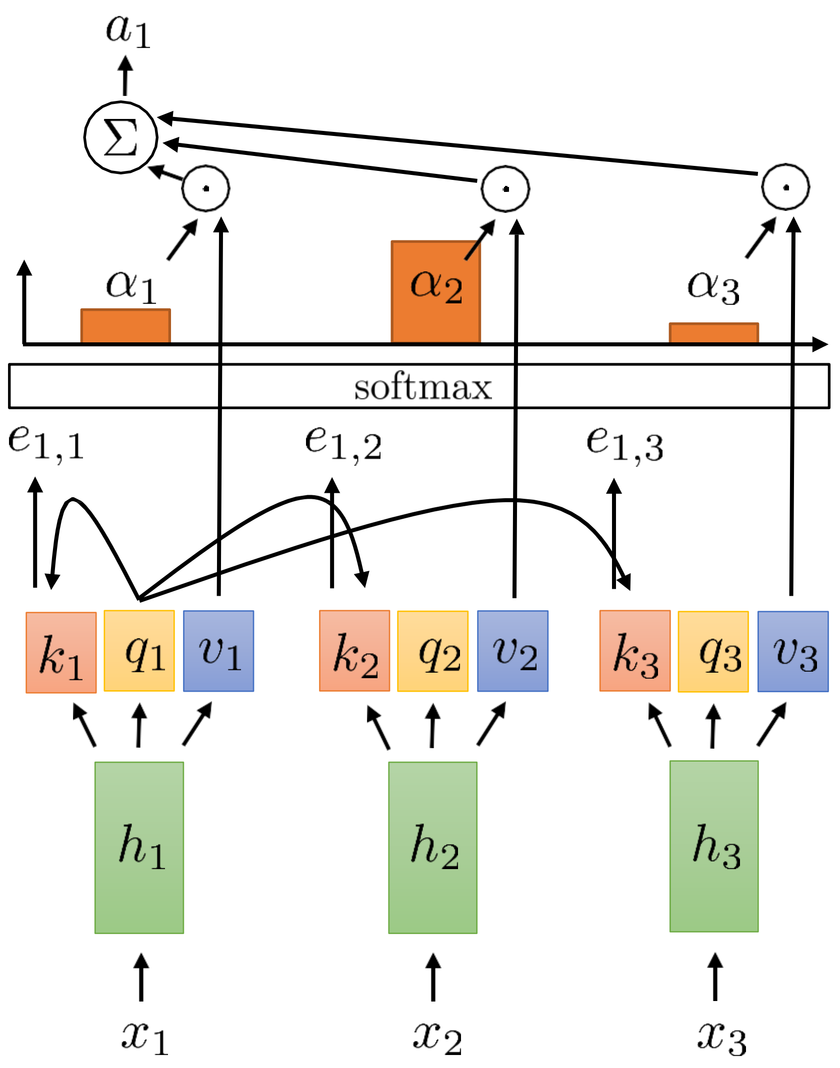


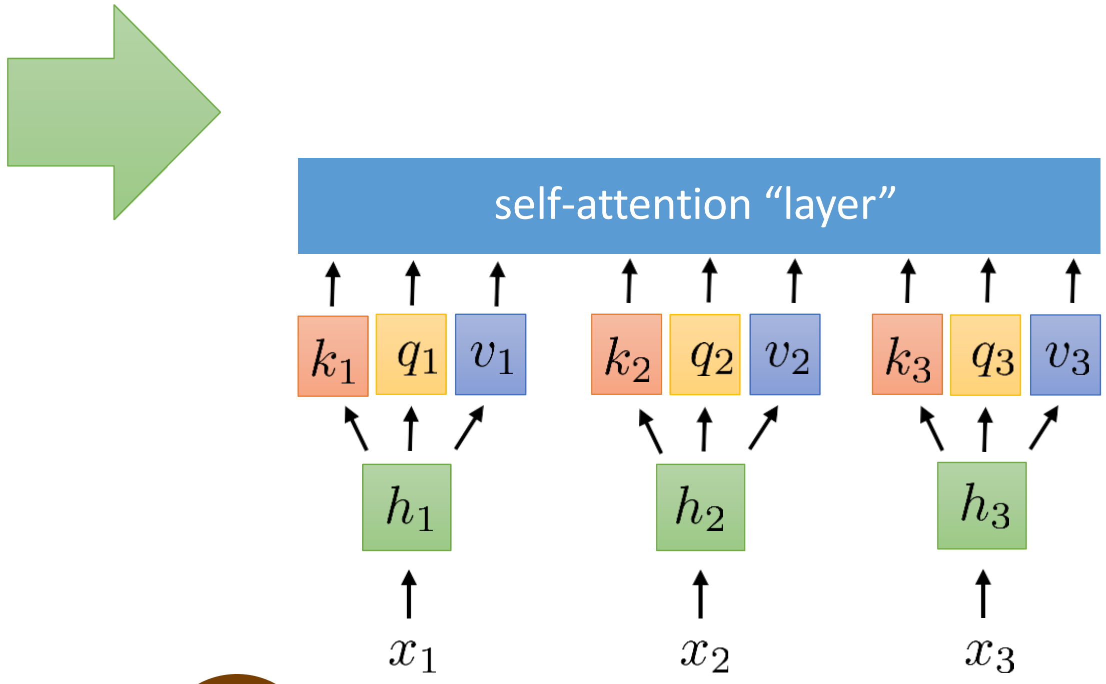
Self-Attention



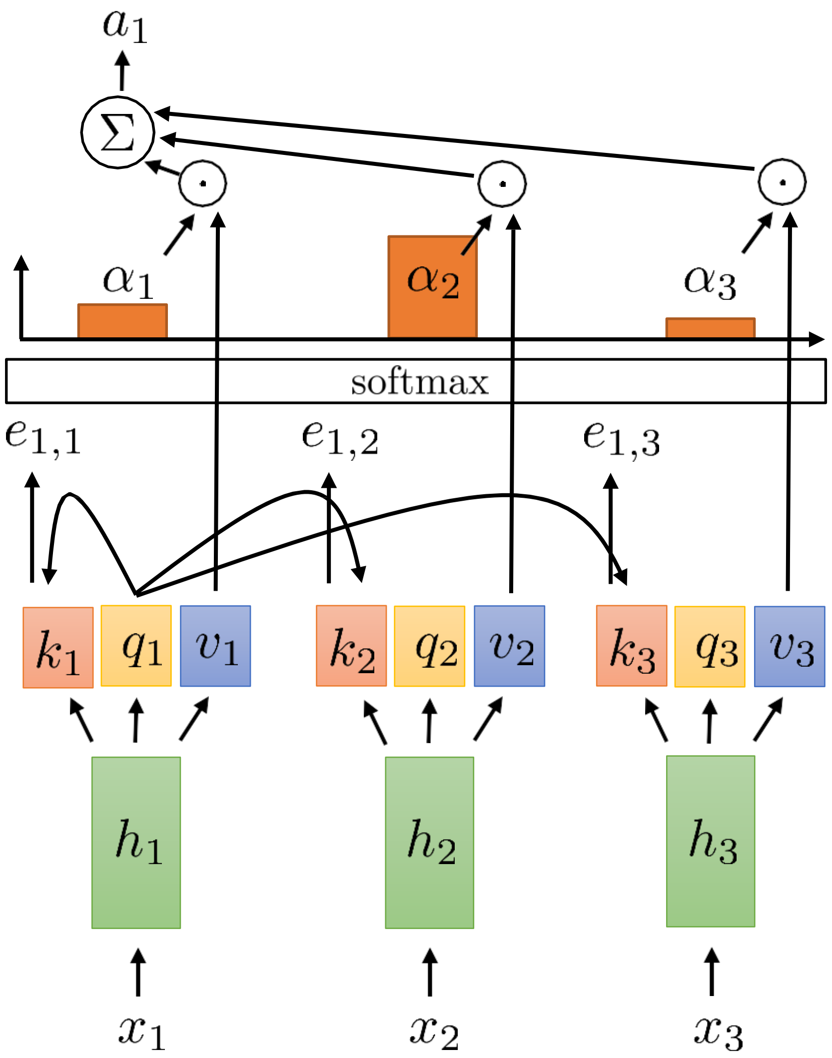


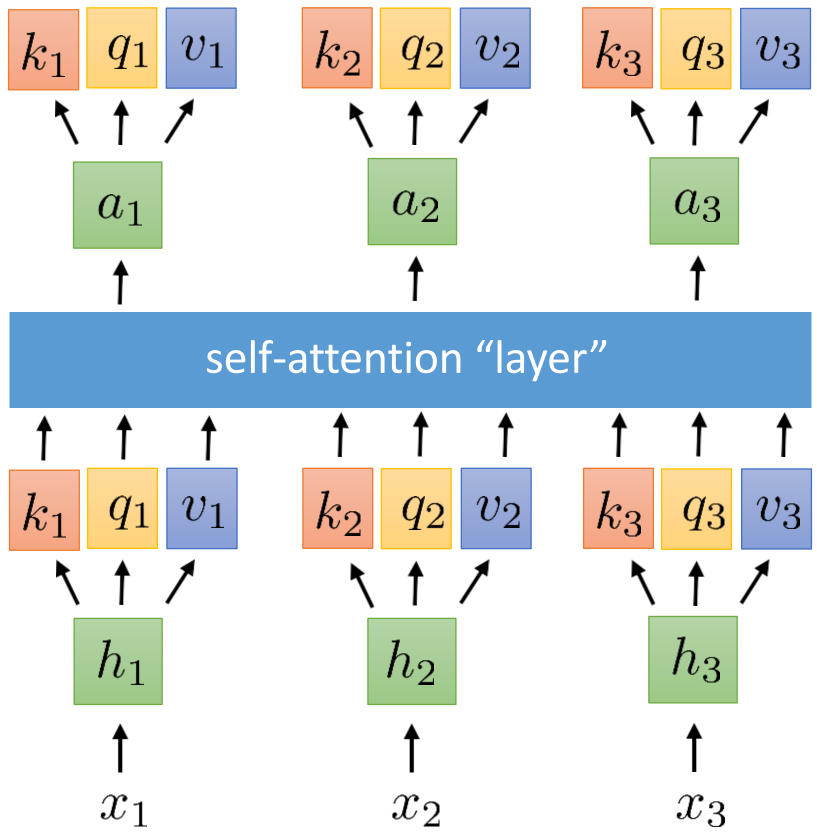
Self-Attention



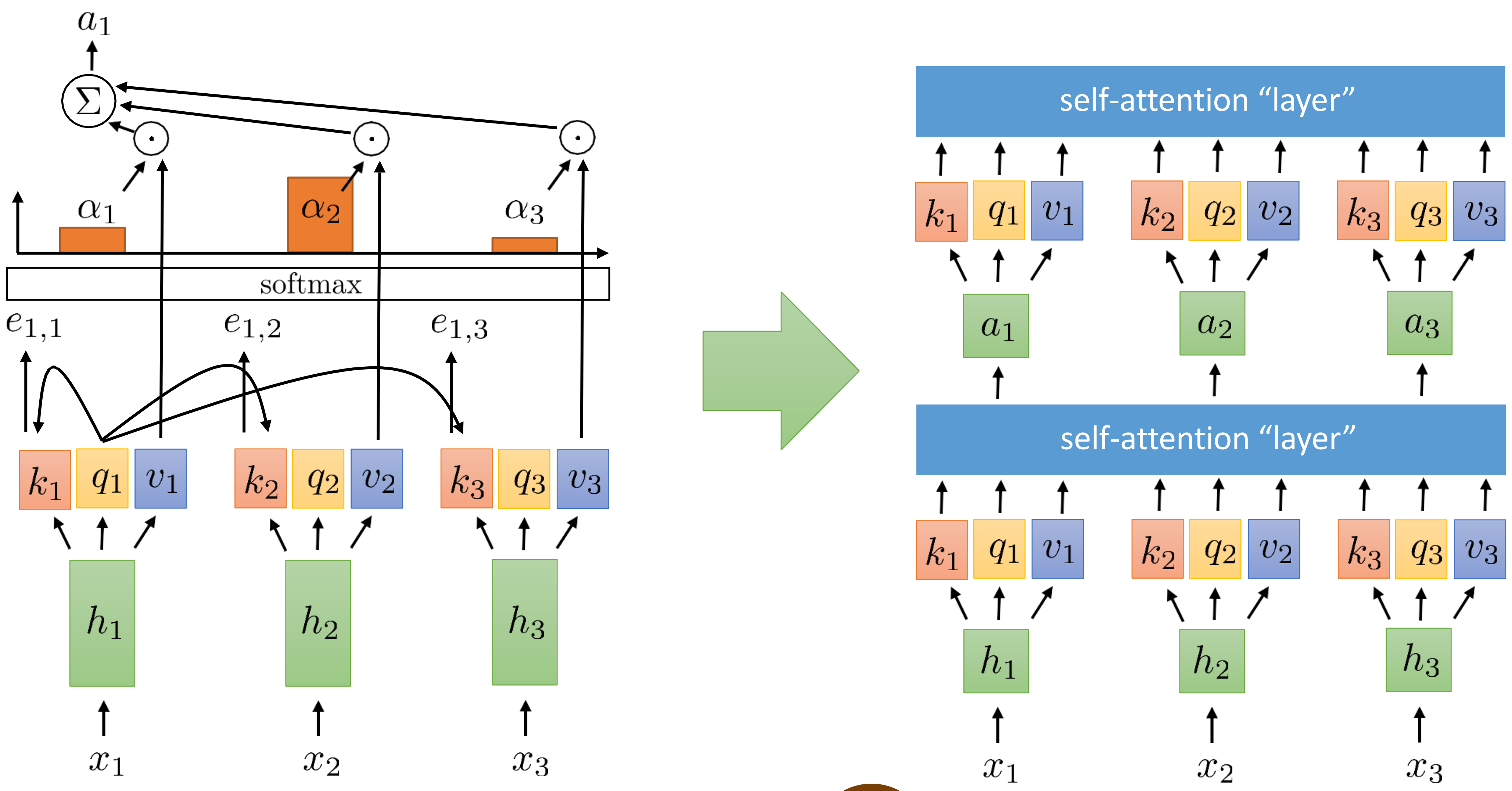


Self-Attention

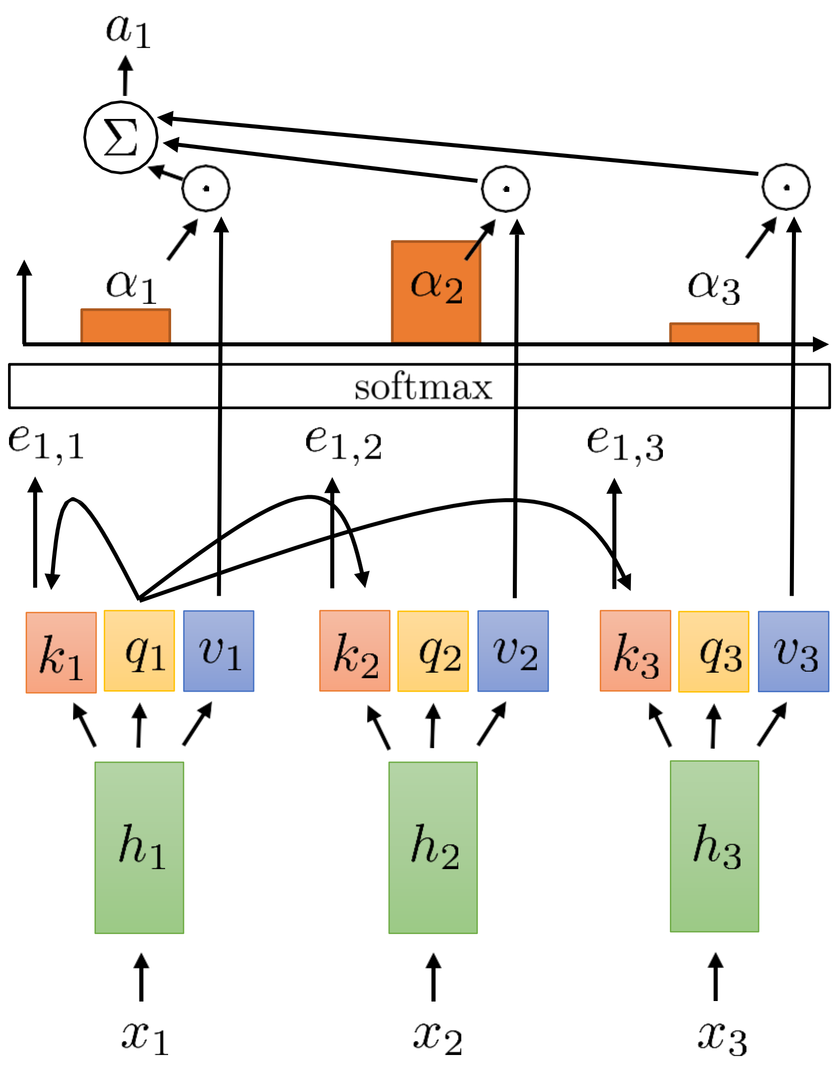


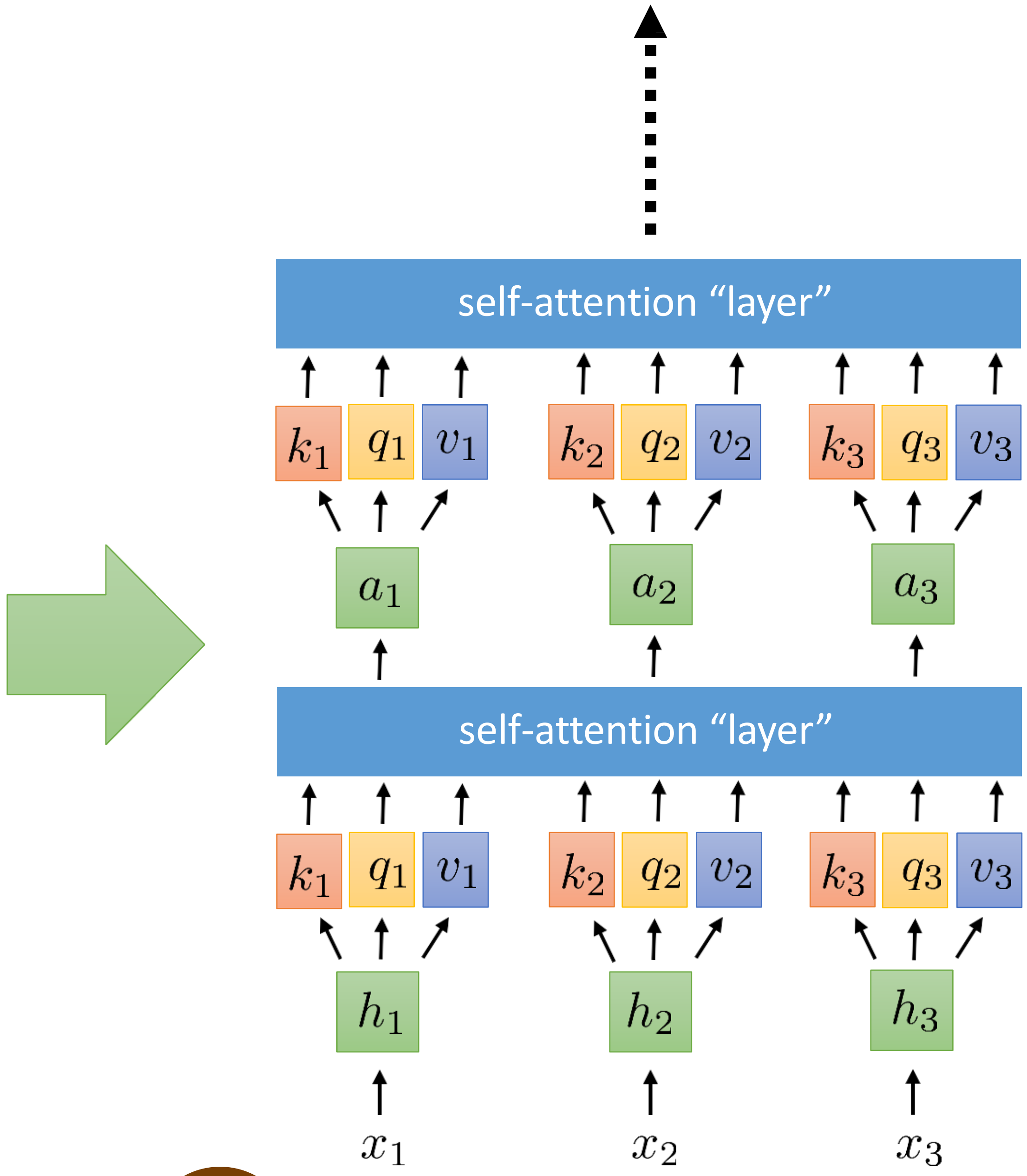


Self-Attention

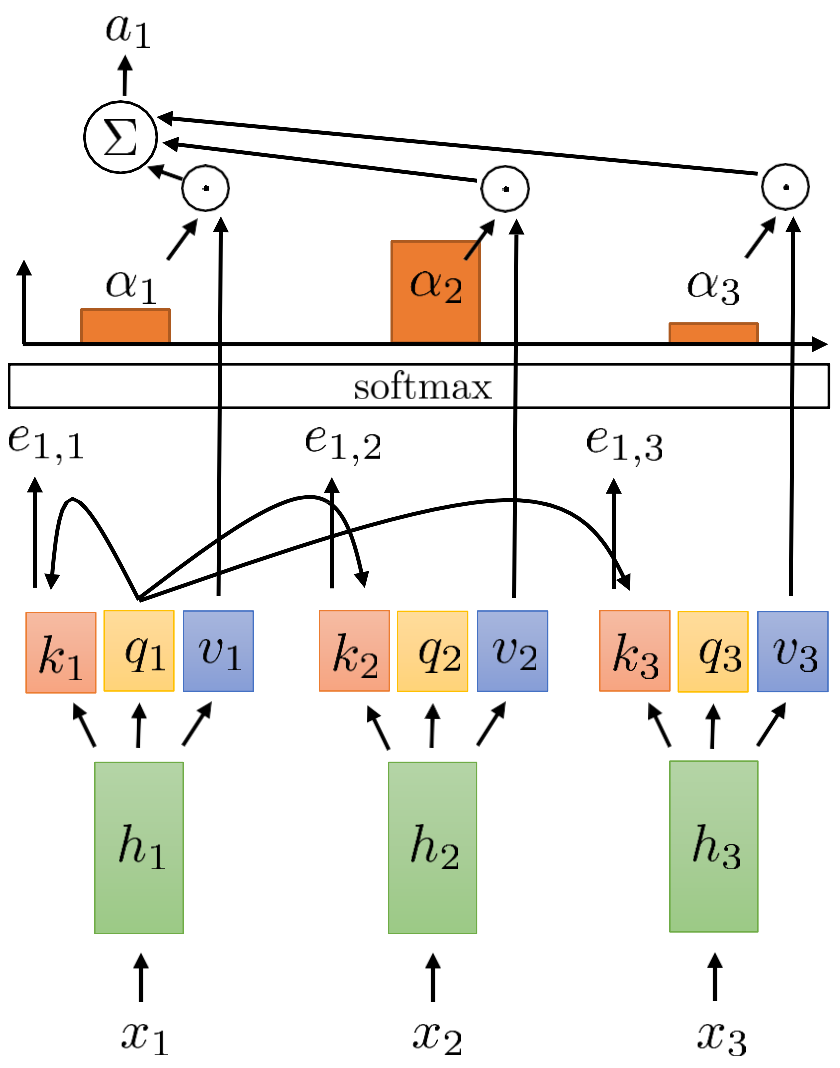


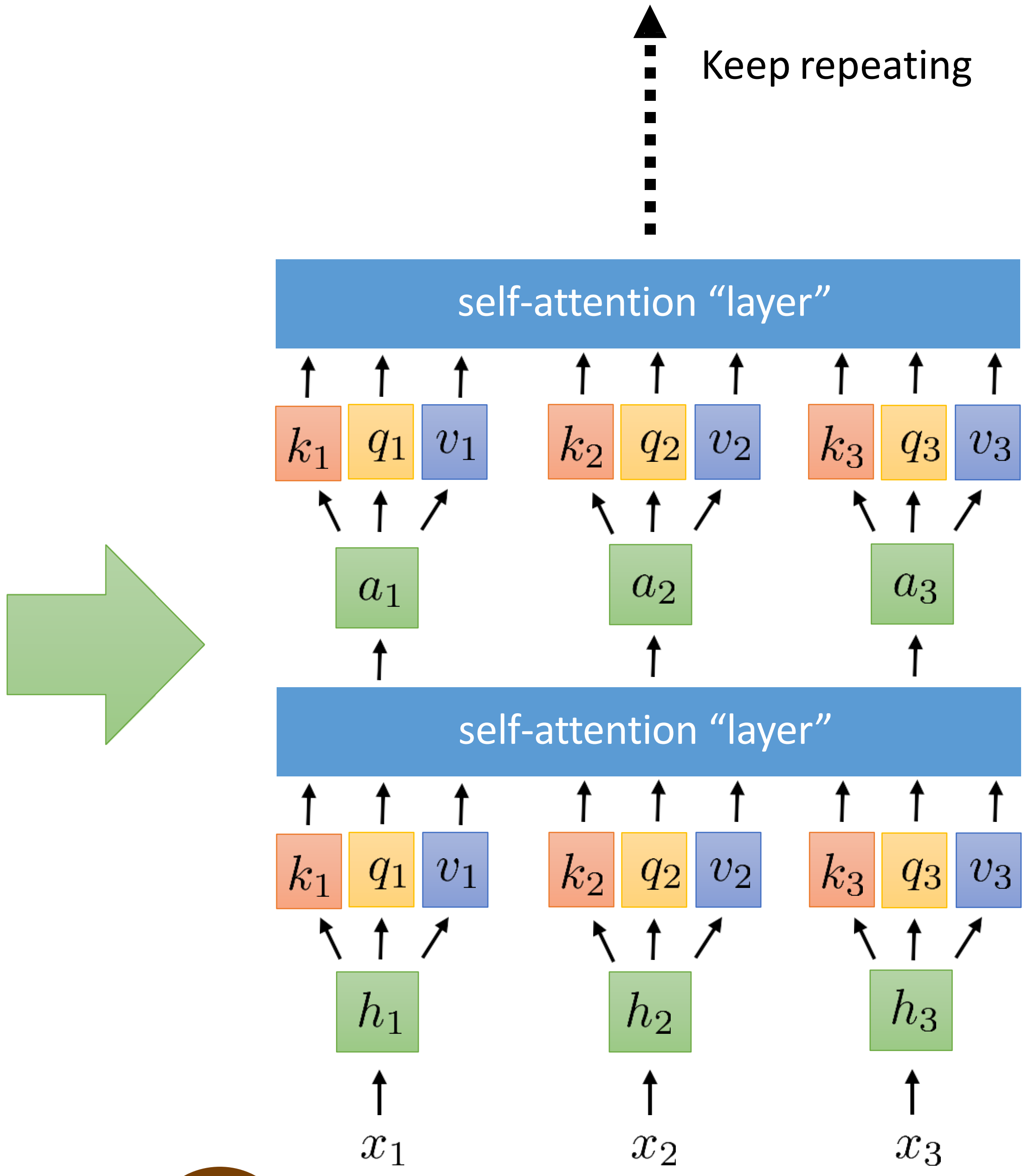
Self-Attention





Self-Attention





From Self-Attention to Transformers

- We will talk about a class of models for processing sequences that does not use recurrent connections but instead relies entirely on attention and will build up towards a class of models called **Transformers**.
- To address a few key limitations, we need to add certain elements:

1. Positional encoding	addresses lack of sequence information
2. Multi-headed attention	allows querying multiple positions at each layer
3. Adding nonlinearities	so far, each successive layer is *linear* in the previous one
4. Masked decoding	how to prevent attention lookups into the future?

From Self-Attention to Transformers

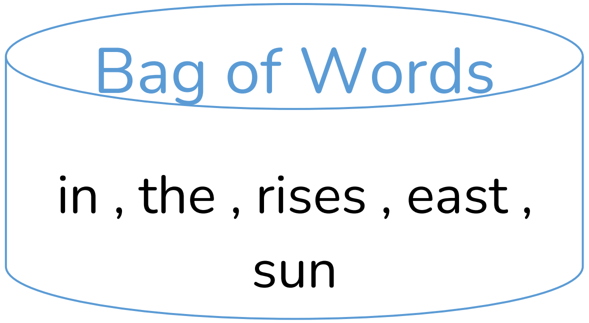
- We will talk about a class of models for processing sequences that does not use recurrent connections but instead relies entirely on attention and will build up towards a class of models called **Transformers**.
- To address a few key limitations, we need to add certain elements:

1. Positional encoding	addresses lack of sequence information
2. Multi-headed attention	allows querying multiple positions at each layer
3. Adding nonlinearities	so far, each successive layer is *linear* in the previous one
4. Masked decoding	how to prevent attention lookups into the future?

Positional Encoding - Motivation

- **Problem :** Self-attention processes all the elements of a sequence in parallel without any regard for their order.
  - Example : the sun rises in the east
  - Permuted version : rises in the sun the east

    the east rises in the sun



  - Self-attention is permutation invariant.
  - In natural language, it is important to take into account the order of words in a sentence.
- **Solution :** Explicitly add positional information to indicate where a word appears in a sequence

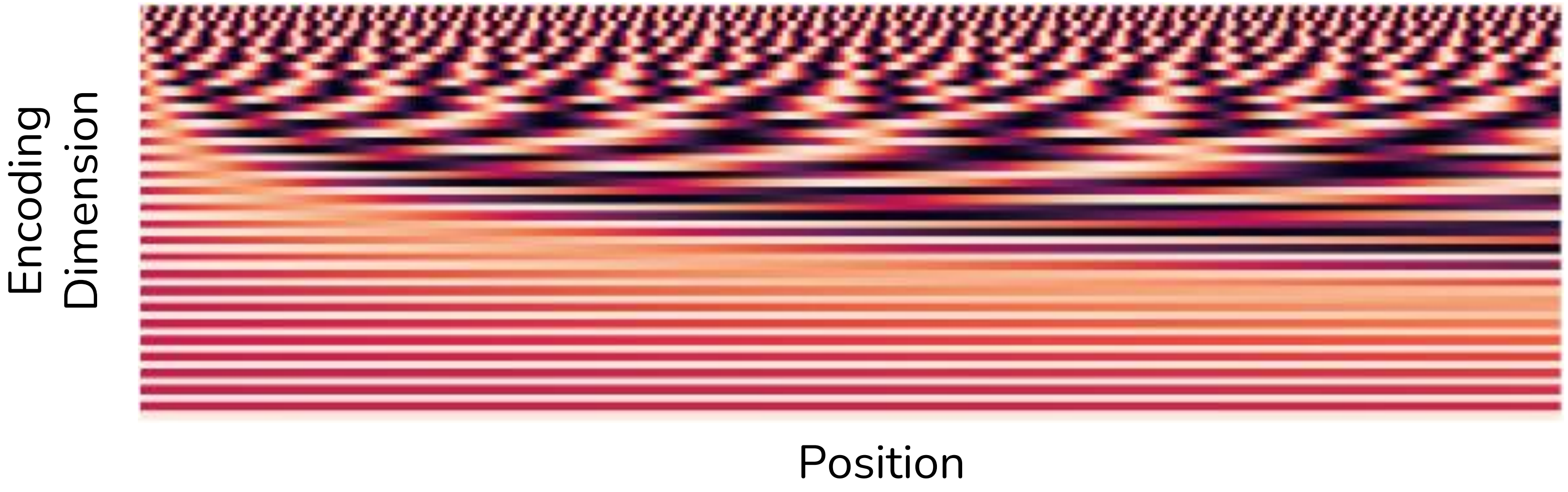
Sinusoidal Positional Encoding

- Helps it determine the position of each word (absolute positional information), or the distance between different words in the sequence(relative positional information)
- The frequency decreases along the encoding dimension.

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

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

Will be discussed in the next module!



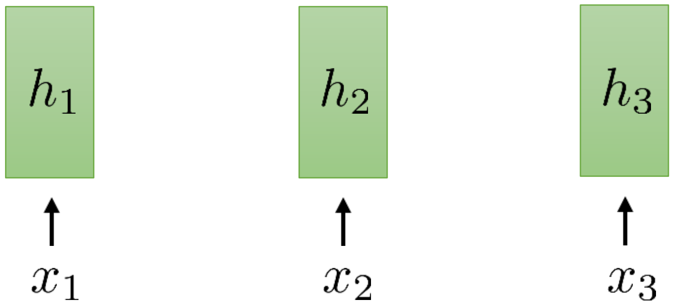
From Self-Attention to Transformers

- We will talk about a class of models for processing sequences that does not use recurrent connections but instead relies entirely on attention and will build up towards a class of models called transformers.
- To address a few key limitations, we need to add certain elements:

1. Positional encoding	addresses lack of sequence information
2. Multi-headed attention	allows querying multiple positions at each layer
3. Adding nonlinearities	so far, each successive layer is *linear* in the previous one
4. Masked decoding	how to prevent attention lookups into the future?

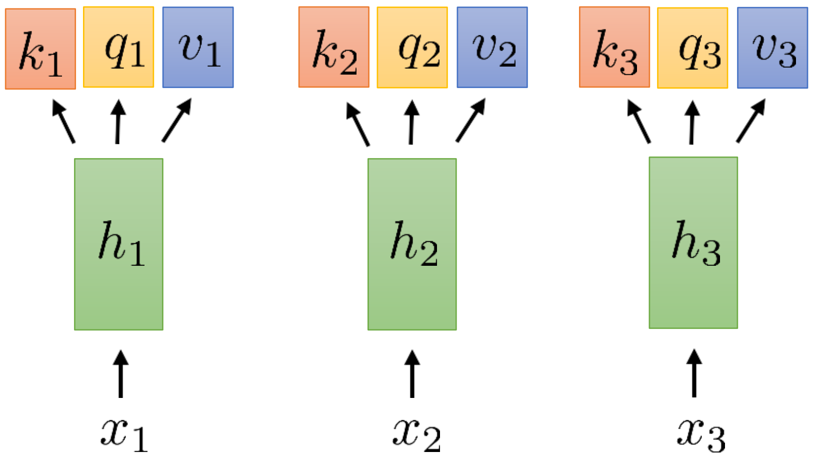
Multi-Head Attention

Given that we're fully depending on attention now, it could be beneficial to include more than one time step.



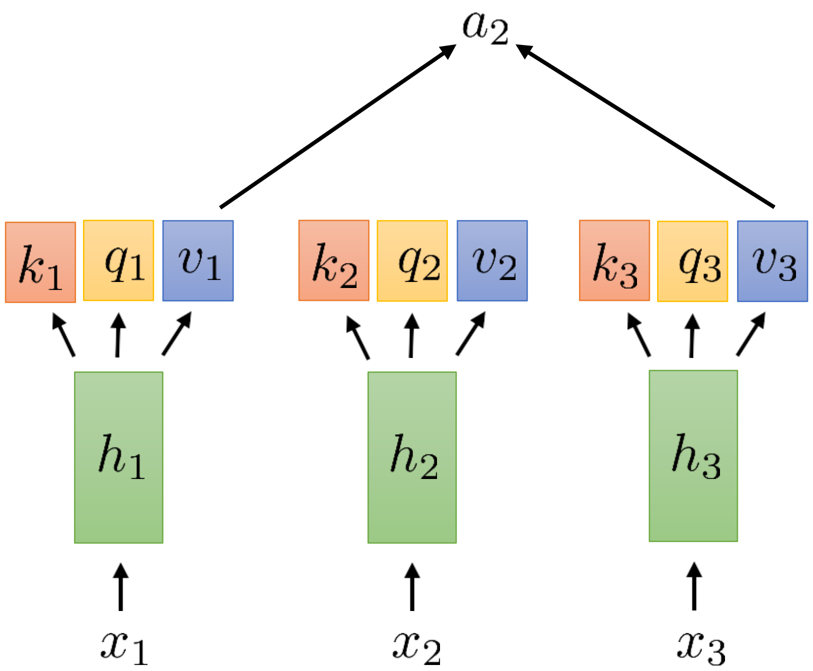
Multi-Head Attention

Given that we're fully depending on attention now, it could be beneficial to include more than one time step.



Multi-Head Attention

Given that we're fully depending on attention now, it could be beneficial to include more than one time step.

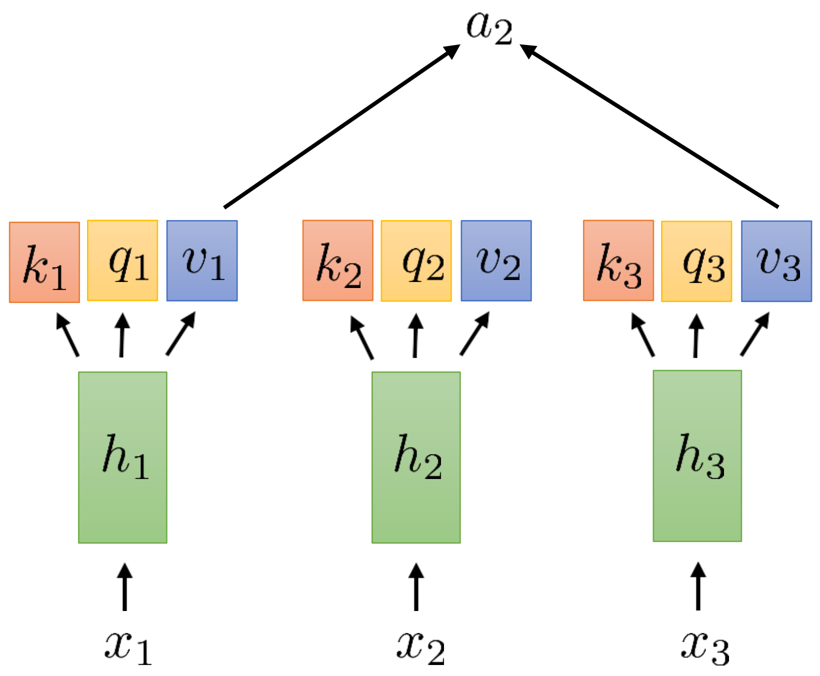


$$a_l = \sum_t \alpha_{l,t} v_t$$

Due to the softmax function, this will be heavily influenced by a single value.

Multi-Head Attention

Given that we're fully depending on attention now, it could be beneficial to include more than one time step.



$$a_l = \sum_t \alpha_{l,t} v_t$$

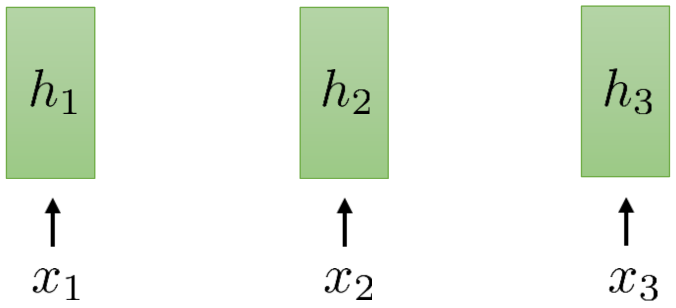
Due to the softmax function, this will be heavily influenced by a single value.

$$e_{l,t} = q_l \cdot k_t$$

It's challenging to clearly specify that you want two distinct elements, like the subject and object in a sentence.

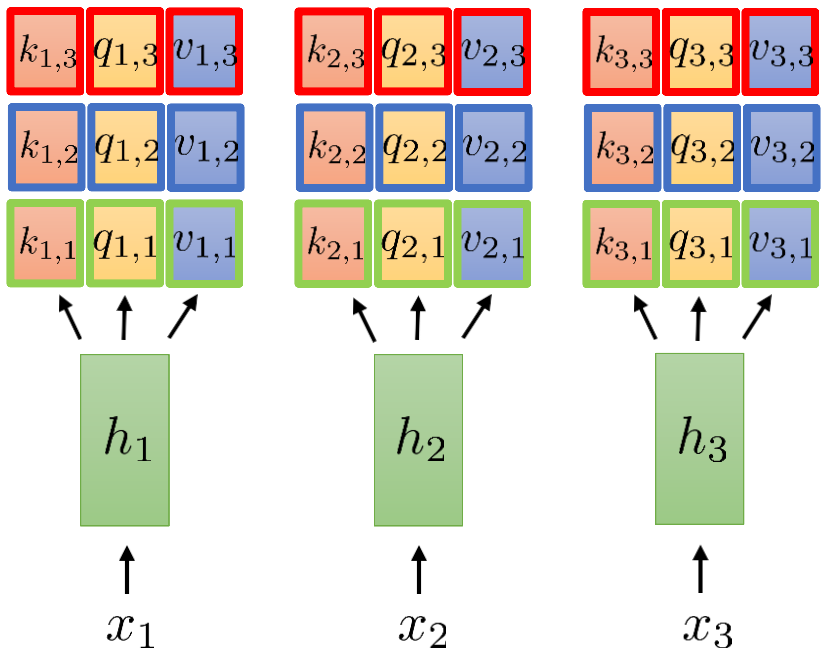
Multi-Head Attention

Solution: Use multiple keys, queries, and values for each time step



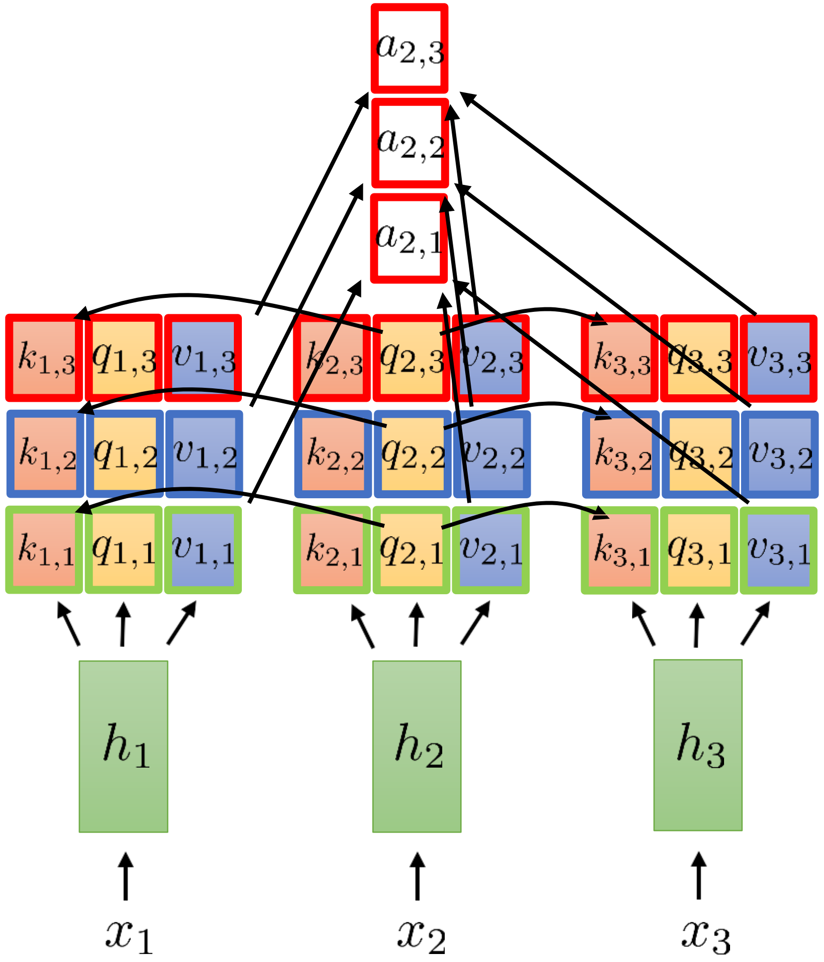
Multi-Head Attention

Solution: Use multiple keys, queries, and values for each time step



Multi-Head Attention

Solution: Use multiple keys, queries, and values for each time step



full attention vector formed by concatenation:

$$a_2 = \begin{bmatrix} a_{2,1} \\ a_{2,2} \\ a_{2,3} \end{bmatrix}$$

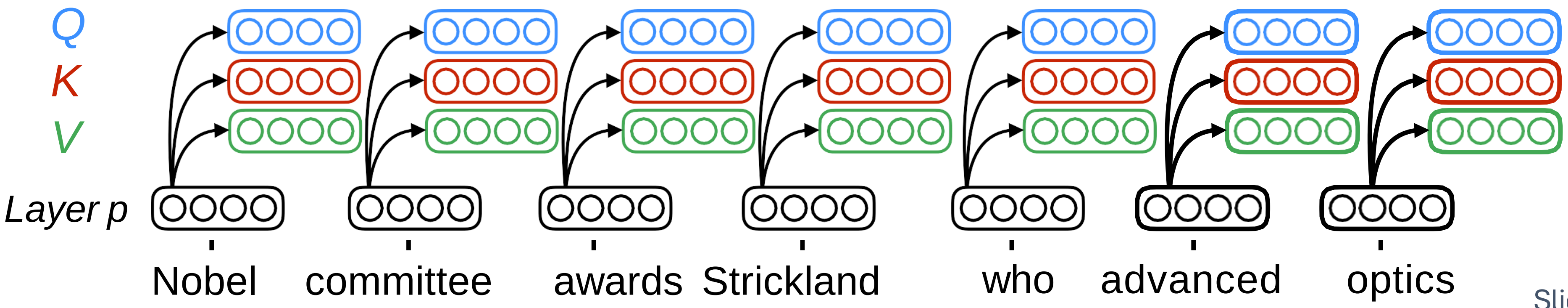
compute weights **independently** for each head

$$e_{l,t,i} = q_{l,i} \cdot k_{l,i}$$

$$\alpha_{l,t,i} = \exp(e_{l,t,i}) / \sum_{t'} \exp(e_{l,t',i})$$

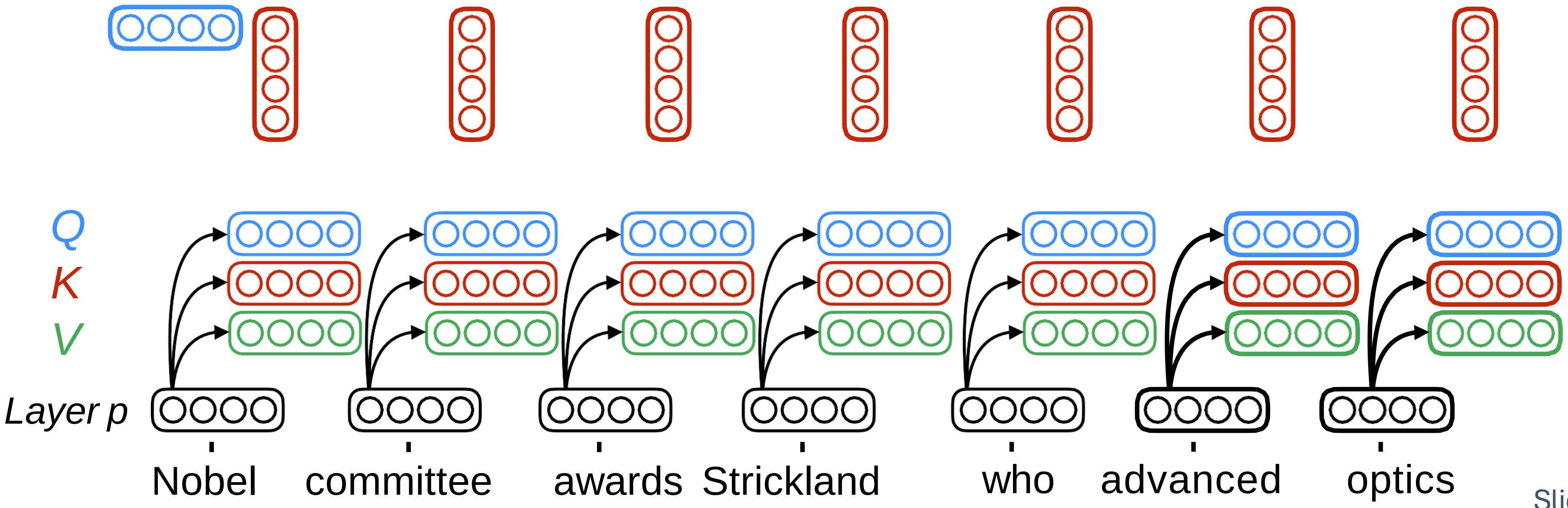
$$a_{l,i} = \sum_{t} \alpha_{l,t,i} v_{t,i}$$

Self-Attention (In Encoder)



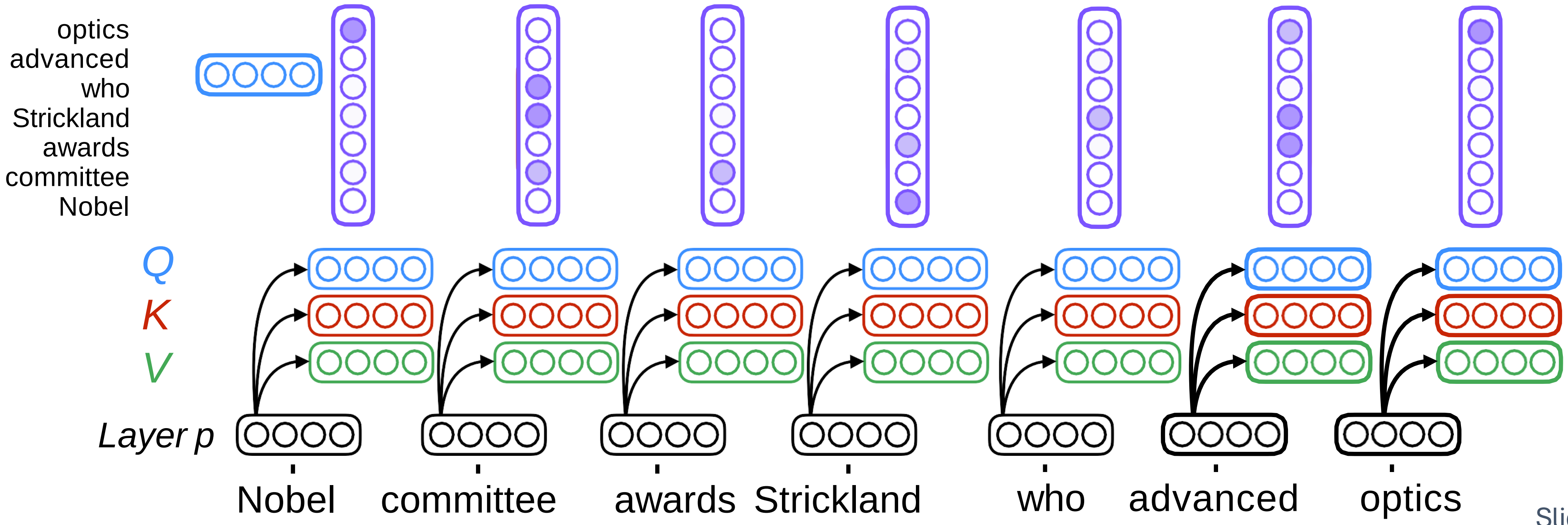
Slides by Emma Strubell

Self-Attention (In Encoder)



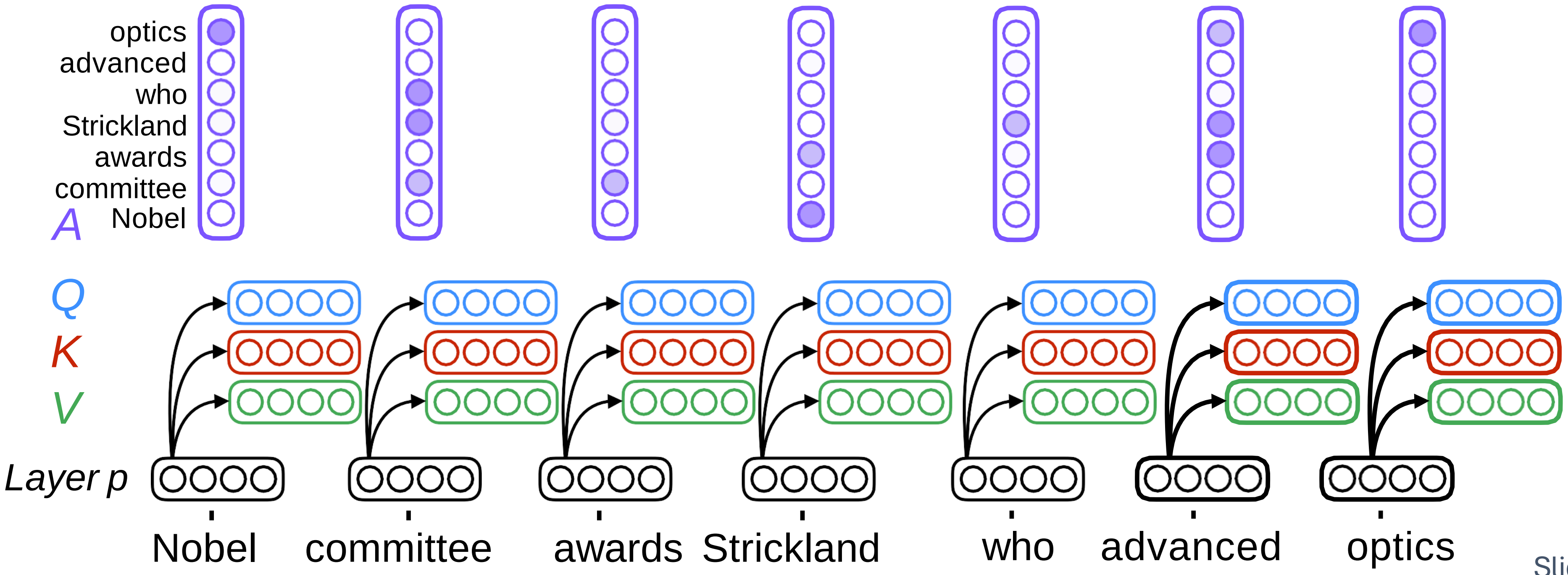
Slides by Emma Strubell

Self-Attention (In Encoder)



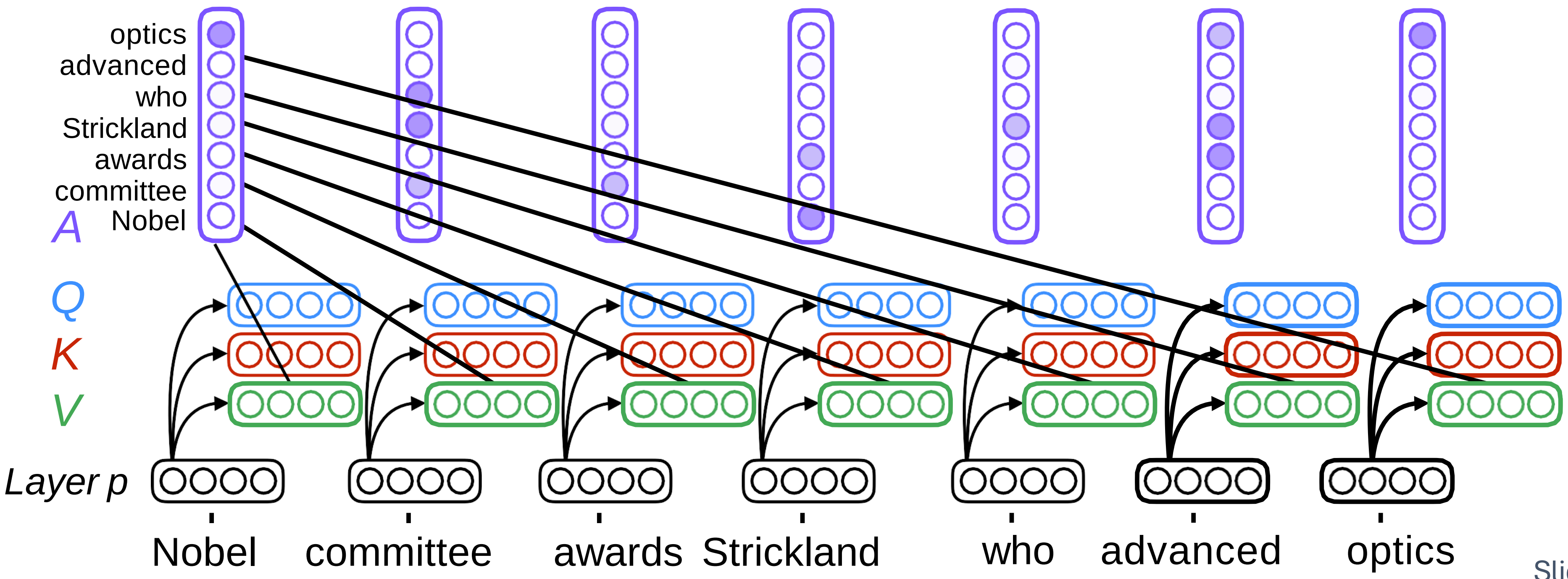
Slides by Emma Strubell

Self-Attention (In Encoder)



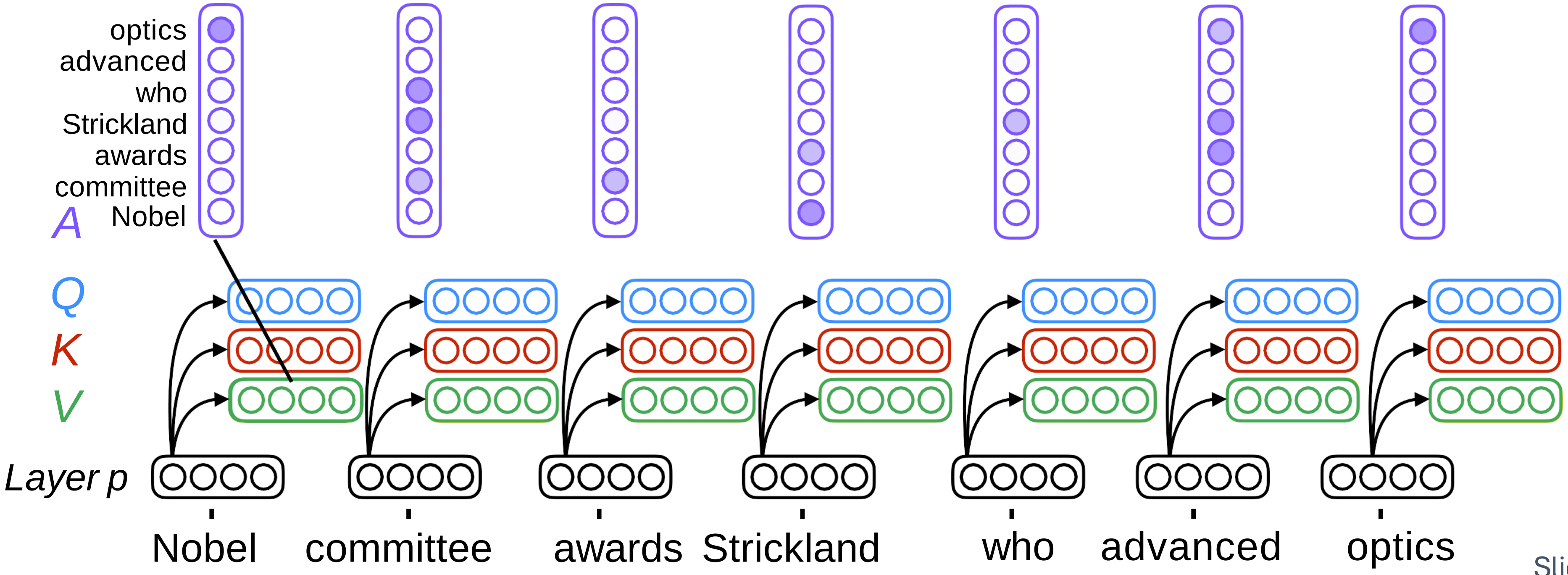
Slides by Emma Strubell

Self-Attention (In Encoder)



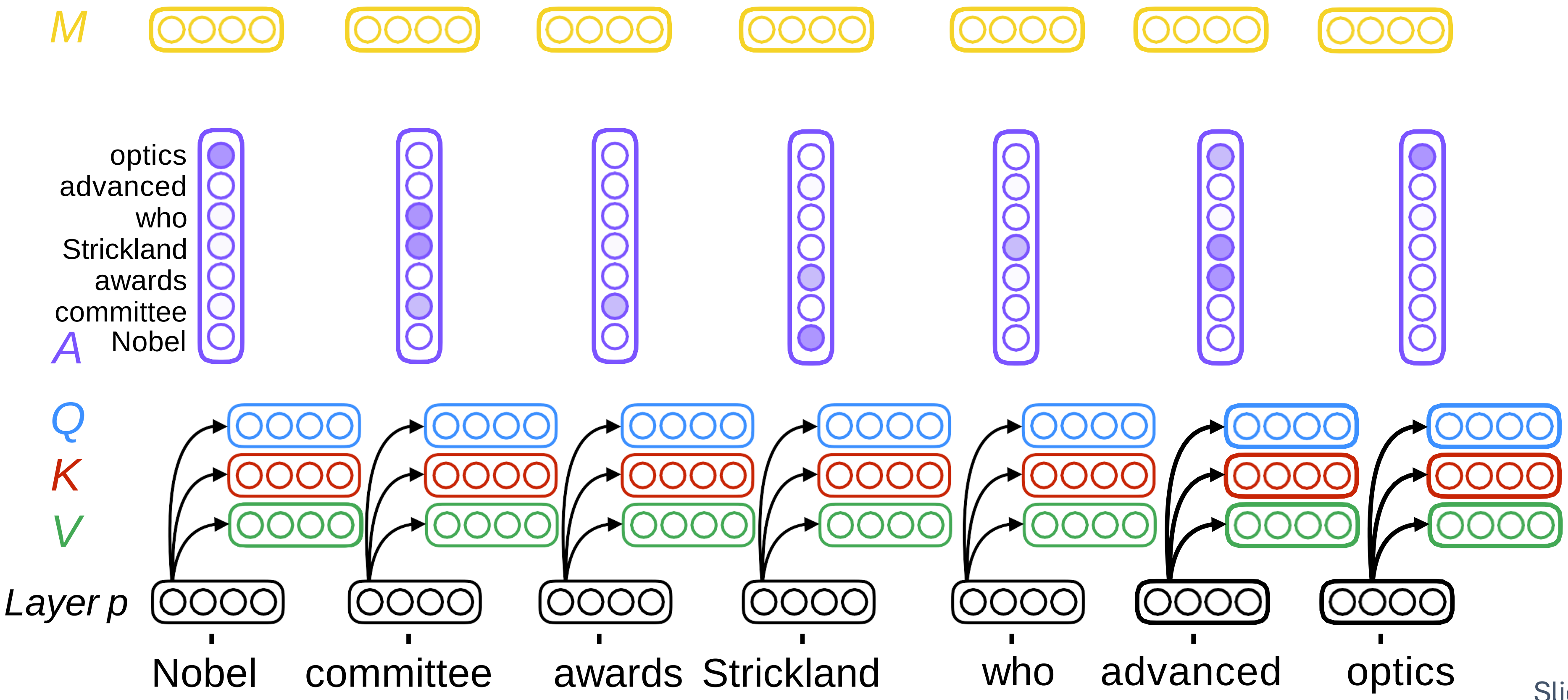
Slides by Emma Strubell

Self-Attention (In Encoder)



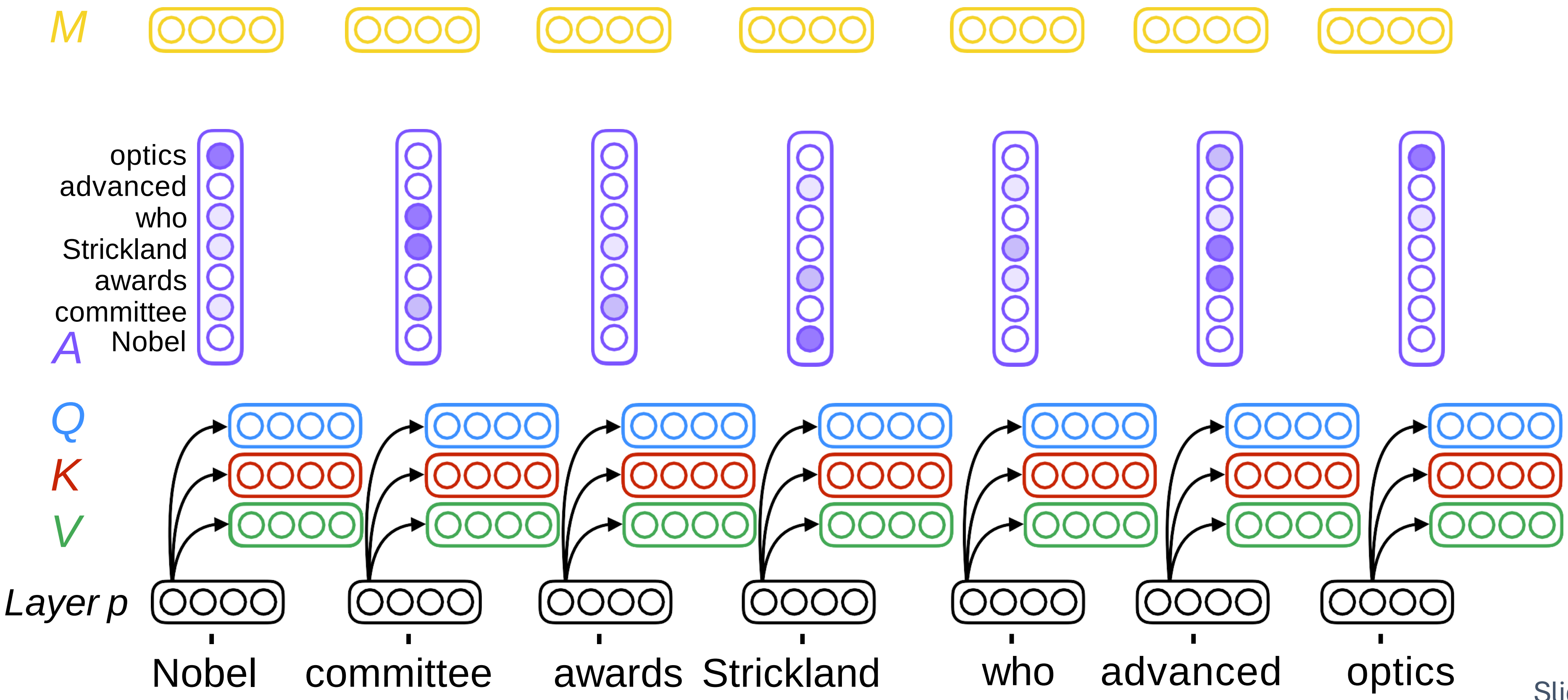
Slides by Emma Strubell

Self-Attention (In Encoder)



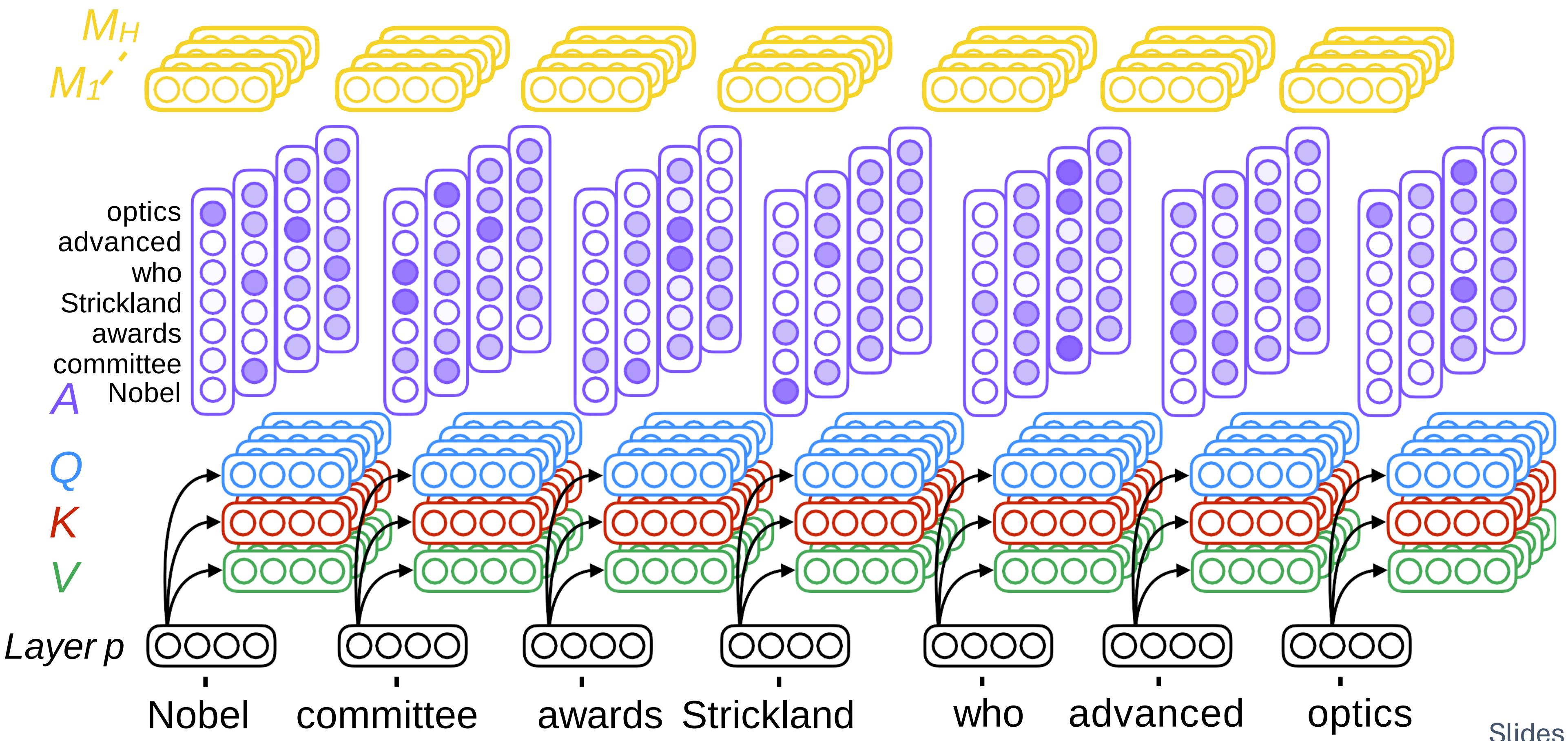
Slides by Emma Strubell

Self-Attention (In Encoder)



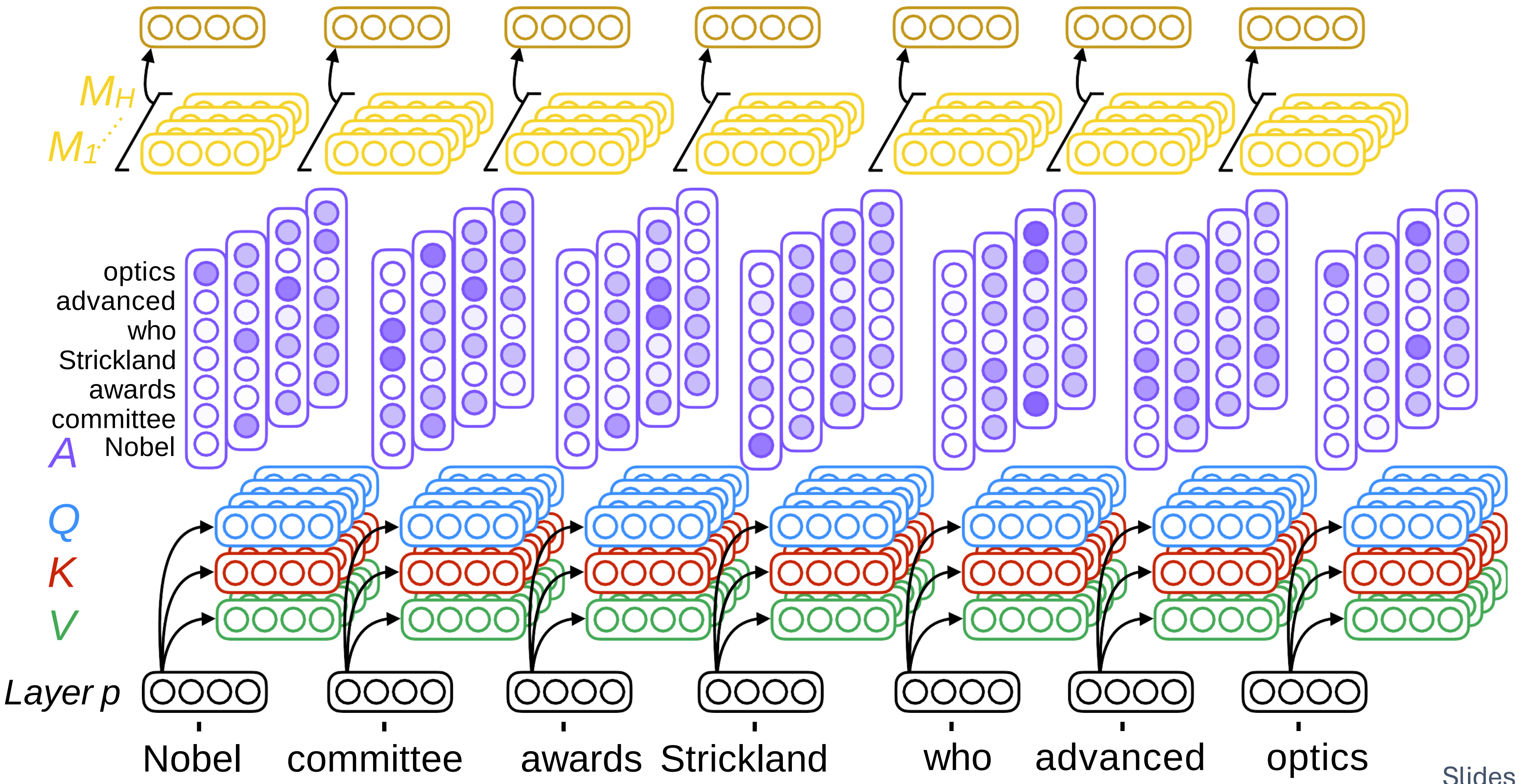
Slides by Emma Strubell

Multi-Head Self-Attention



Slides by Emma Strubell

Multi-Head Self-Attention



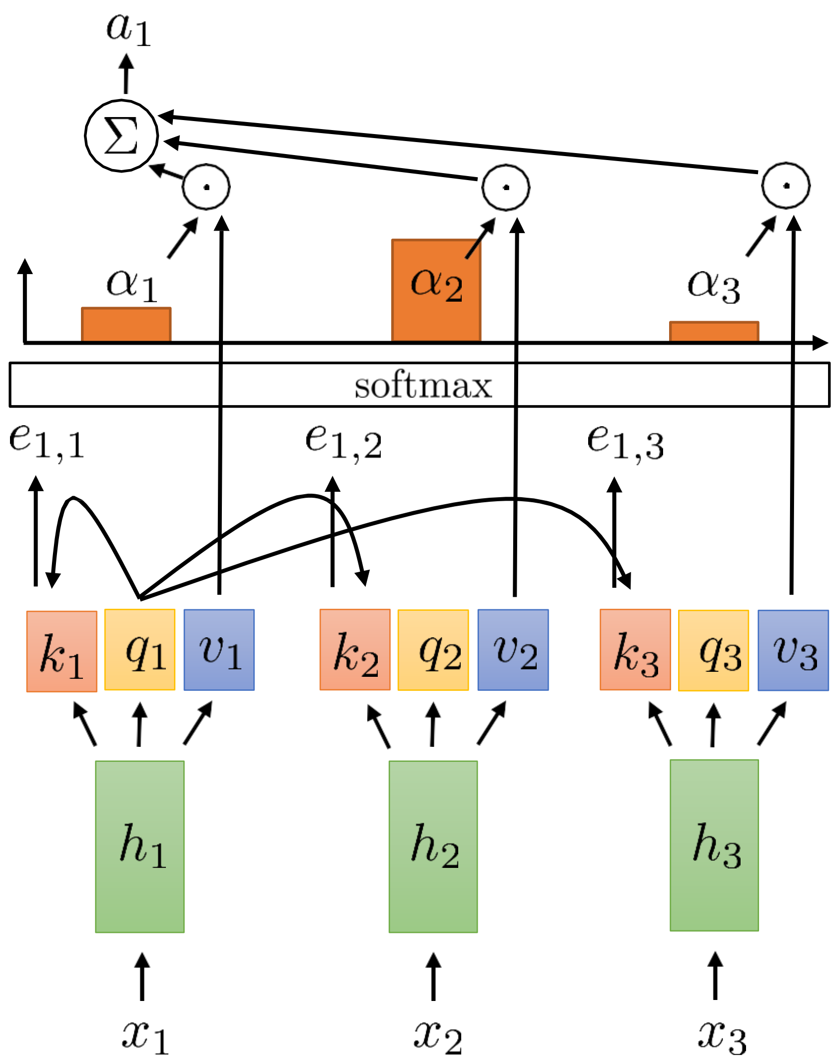
Slides by Emma Strubell

From Self-Attention to Transformers

- We will talk about a class of models for processing sequences that does not use recurrent connections but instead relies entirely on attention and will build up towards a class of models called transformers.
- To address a few key limitations, we need to add certain elements:

1. Positional encoding	addresses lack of sequence information
2. Multi-headed attention	allows querying multiple positions at each layer
3. Adding nonlinearities	so far, each successive layer is *linear* in the previous one
4. Masked decoding	how to prevent attention lookups into the future?

Self-Attention Is "Linear"



$$k_t = W_k h_t \qquad q_t = W_q h_t \qquad v_t = W_v h_t$$

$$\alpha_{l,t} = \exp(e_{l,t}) / \sum_{t'} \exp(e_{l,t'})$$

$$e_{l,t} = q_l \cdot k_t$$

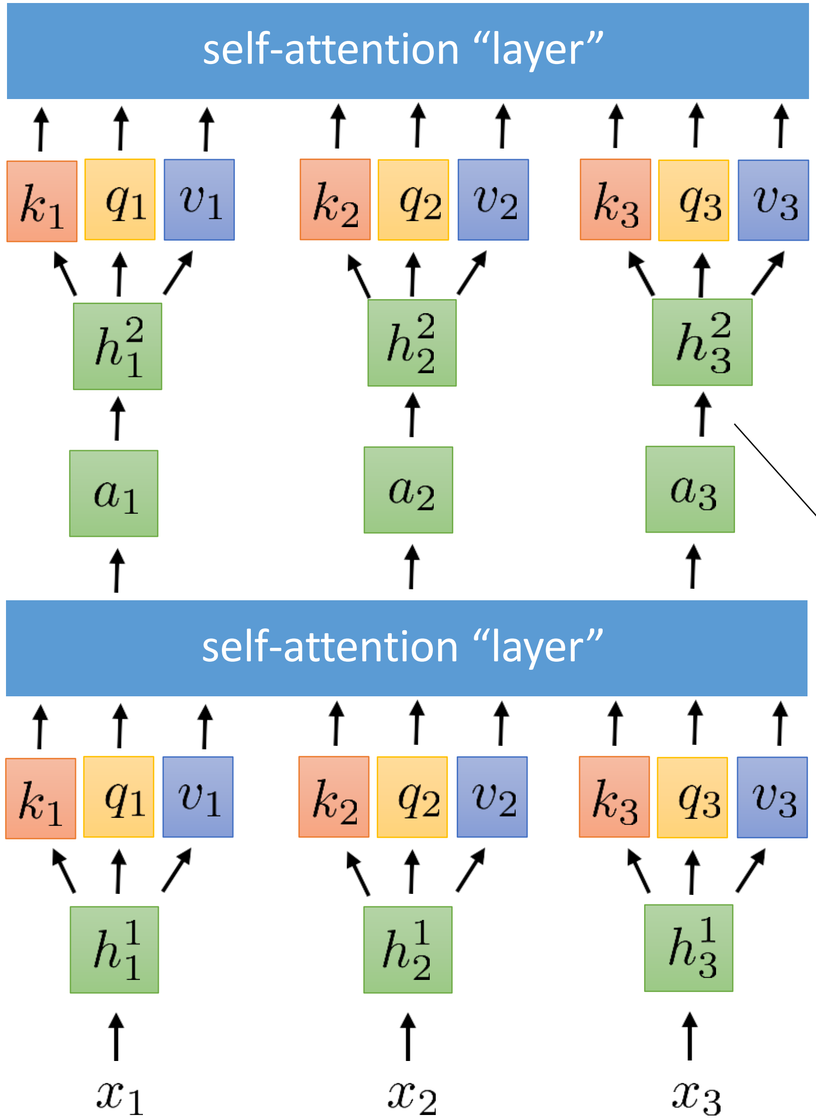
$$a_l = \sum_t \alpha_{l,t} v_t = \sum_t \alpha_{l,t} W_v h_t = W_v \sum_t \alpha_{l,t} h_t$$

linear transformation

non-linear weights

Problem: Every self-attention layer is a linear transformation of the previous layer with non-linear weights.

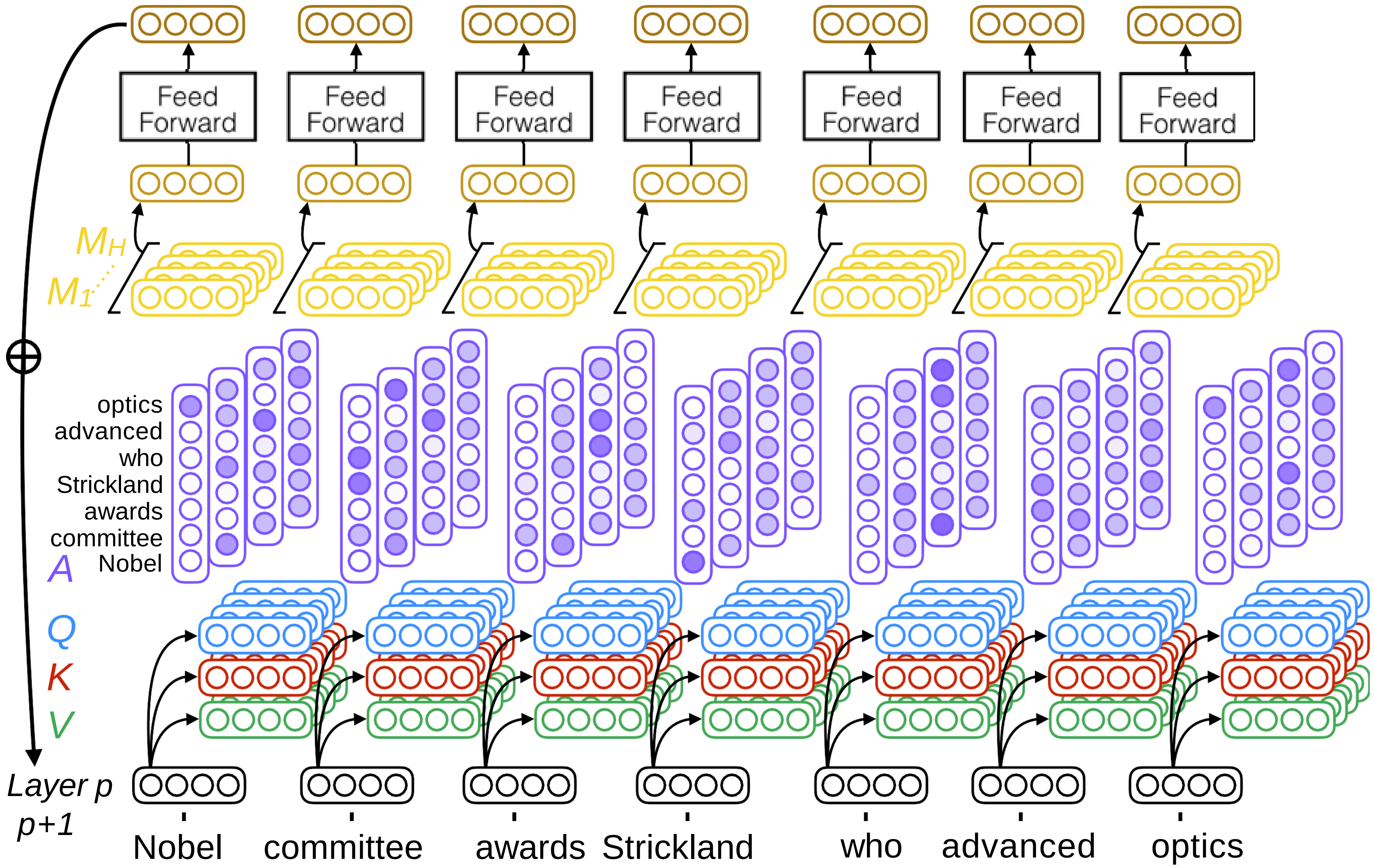
Position-wise Feed-Forward Networks



- **Solution :** Make the model more expressive is by alternating use of self-attention and non-linearity.
- Non-linearity is incorporated by means of a feed-forward network which consists of two linear transformations with a ReLU activation in between.

$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

- The same non-linearity is utilized across various positions but they differ from layer to layer.

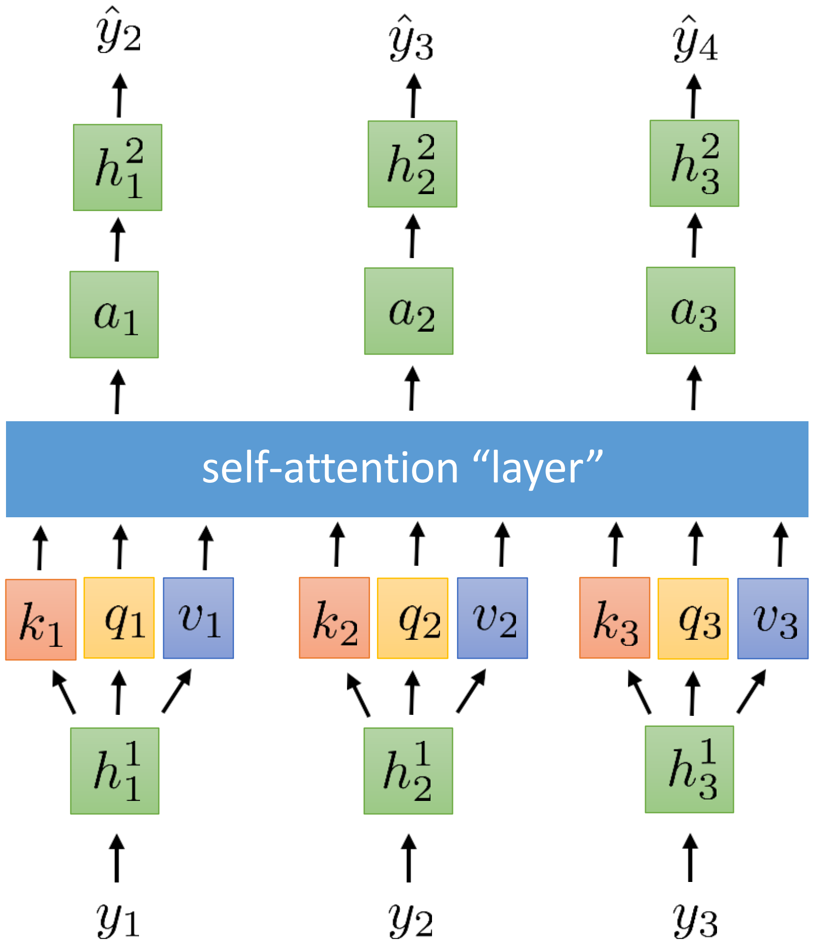

Feed Forward
M_H
M_1
optics
advanced
who
Strickland
awards
committee
Nobel
A
Q
K
V
Layer p
p+1
Nobel
committee
awards
Strickland
who
advanced
optics

From Self-Attention to Transformers

- We will talk about a class of models for processing sequences that does not use recurrent connections but instead relies entirely on attention and will build up towards a class of models called transformers.
- To address a few key limitations, we need to add certain elements:

1. Positional encoding	addresses lack of sequence information
2. Multi-headed attention	allows querying multiple positions at each layer
3. Adding nonlinearities	so far, each successive layer is *linear* in the previous one
4. Masked decoding	how to prevent attention lookups into the future?

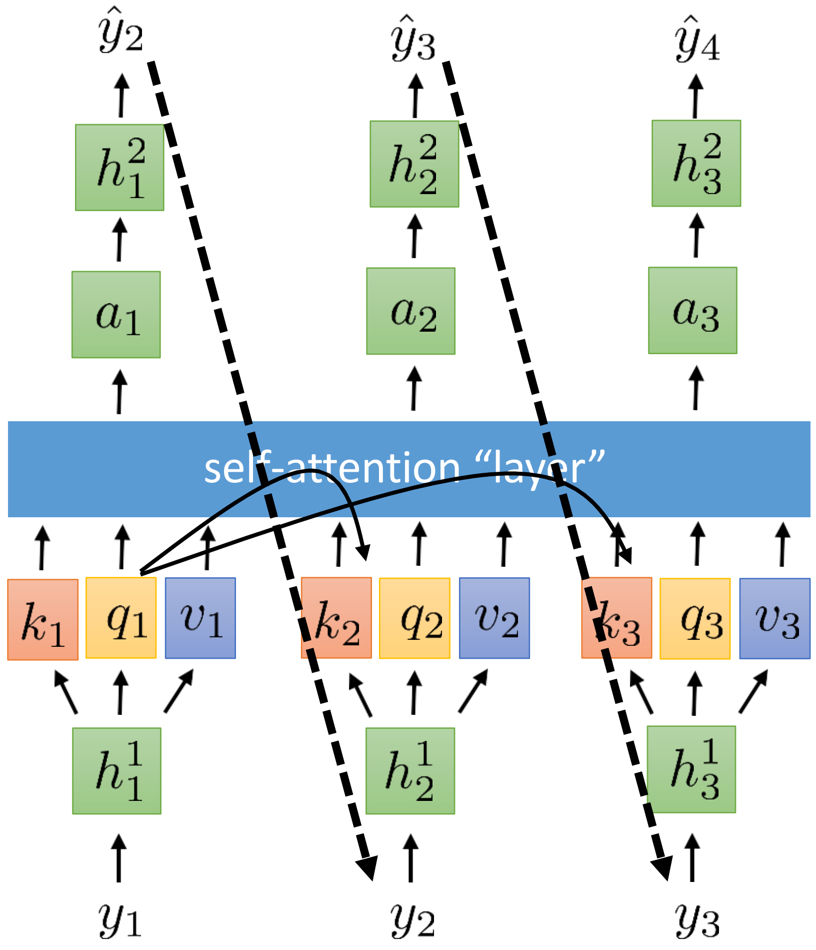
Self-attention can see the future!



A **crude** self-attention "language model":

In practice, there would be several alternating self-attention layers and position-wise feedforward networks

Self-attention can see the future!



A **crude** self-attention "language model":

In practice, there would be several alternating self-attention layers and position-wise feedforward networks

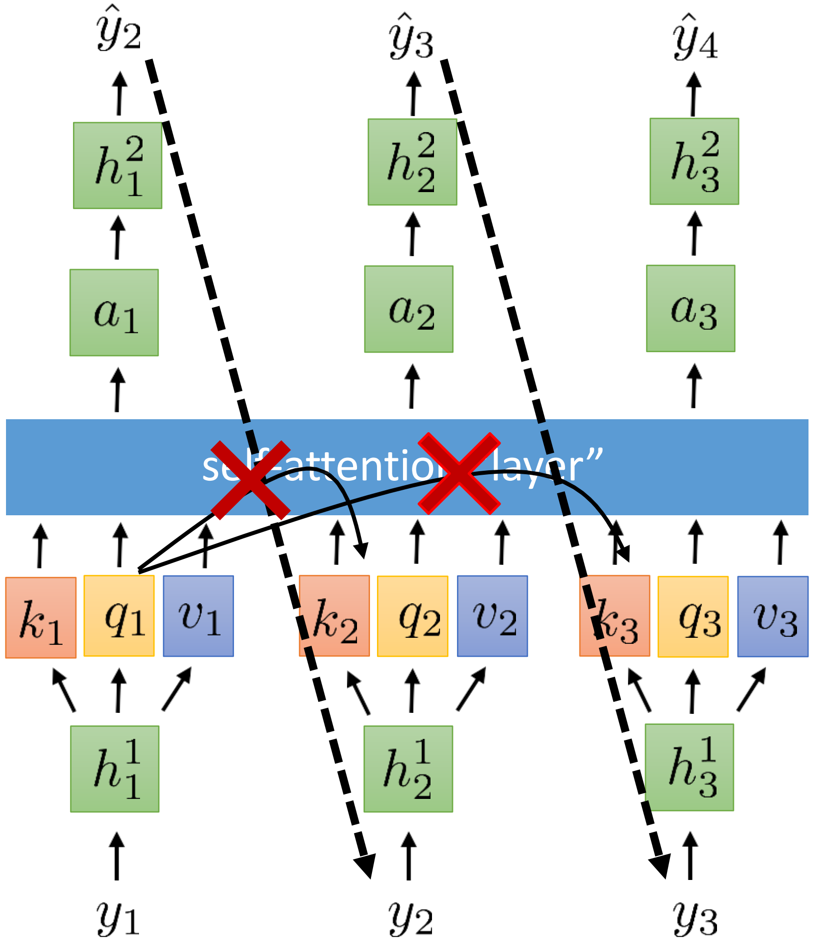
Big problem: self-attention at step 1 can look at the value at steps 2 & 3, which is based on the **inputs** at steps 2 & 3

At test time (when decoding), the **inputs** at steps 2 & 3 will be based on the output at step 1…

…which requires knowing the **input** at steps 2 & 3

Masked Attention

A **crude** self-attention "language model":



At test time (when decoding), the **inputs** at steps 2 & 3 will be based on the output at step 1…

…which requires knowing the **input** at steps 2 & 3

Must allow self-attention into the **past**…

…but not into the **future**

Easy solution:

$$\cancel{e_{l,t} = q_l \cdot k_t}$$

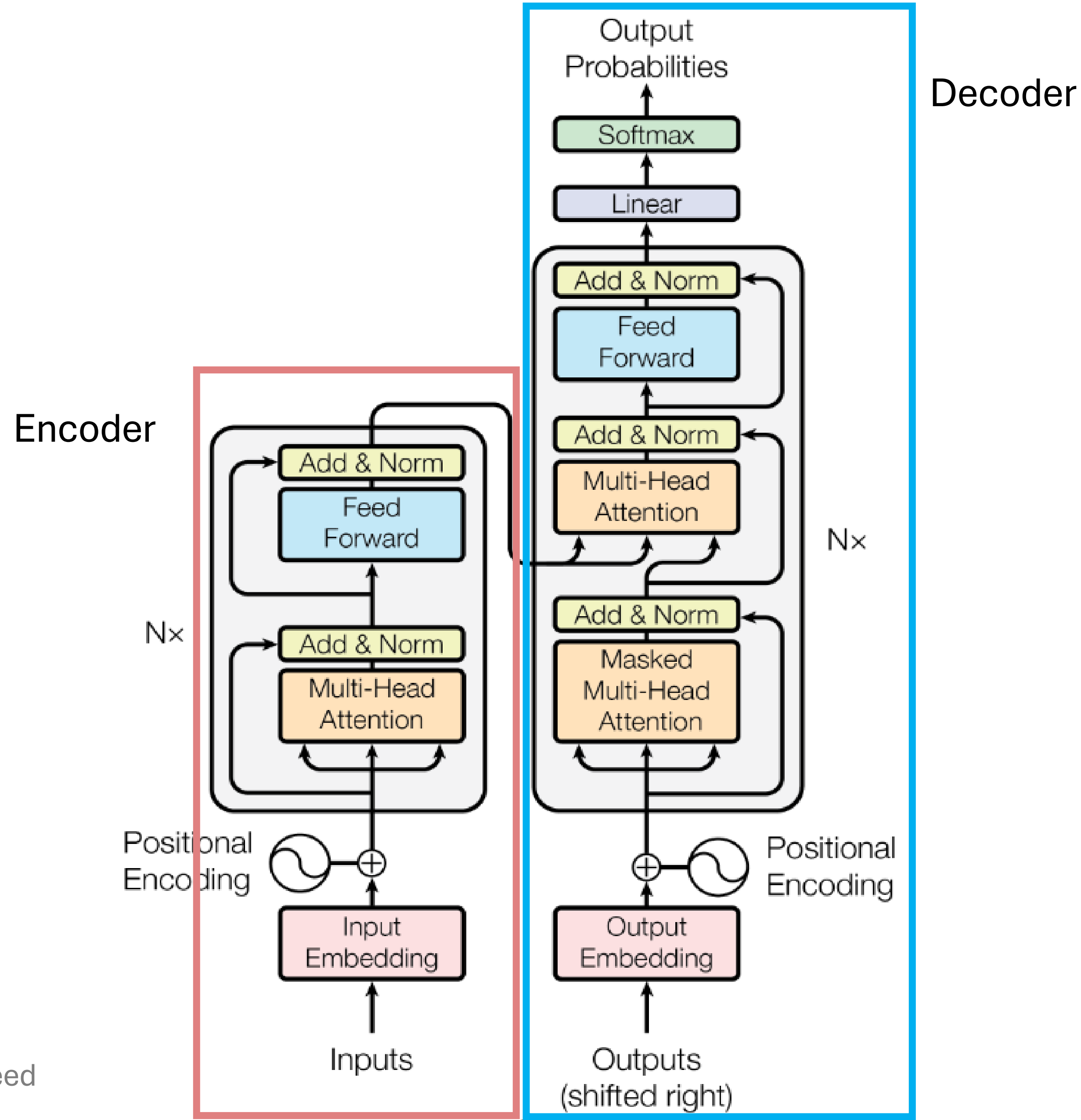
$$e_{l,t} = \begin{cases} q_l \cdot k_t & \text{if } l \geq t \\ -\infty & \text{otherwise} \end{cases}$$

in practice:

just replace $\exp(e_{l,t})$ with 0 if $l < t$

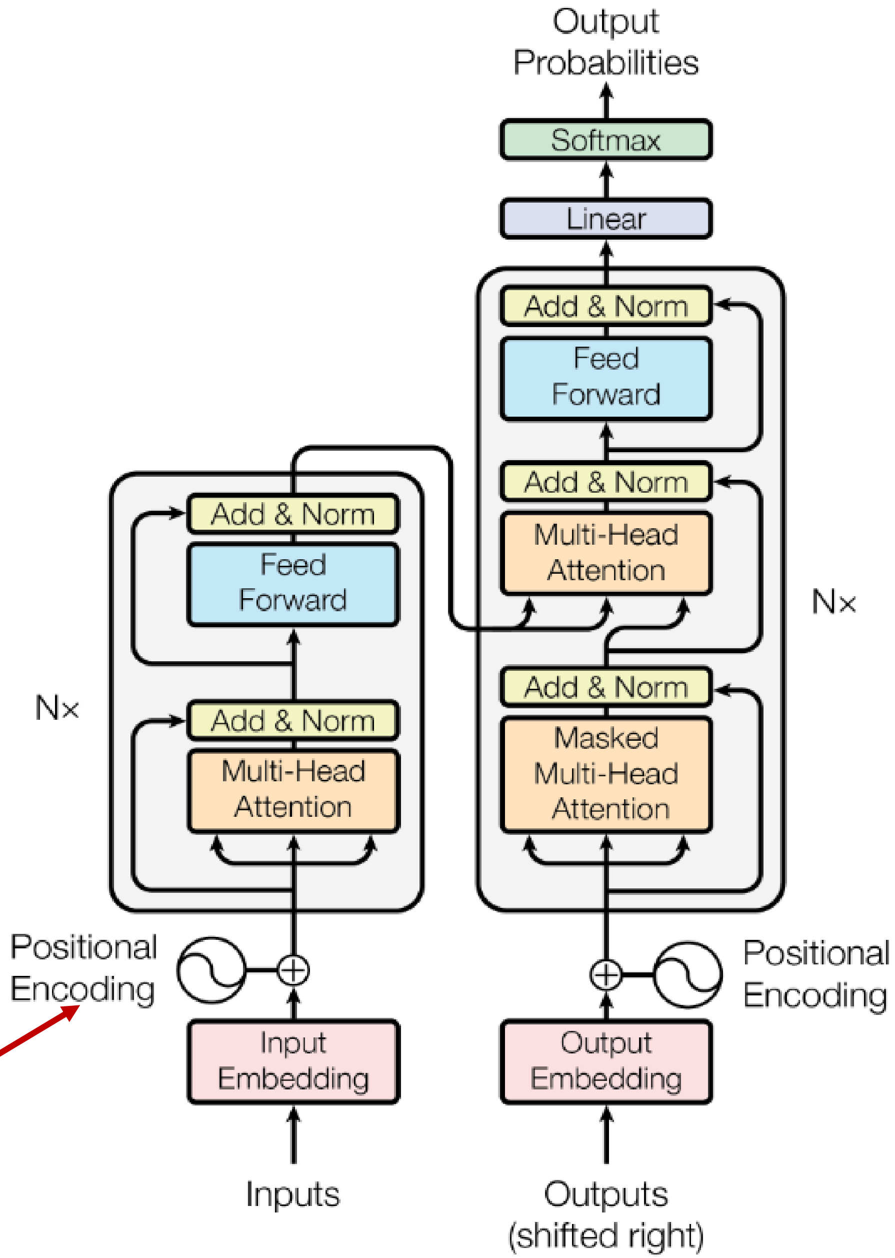
inside the softmax

Transformer Architecture



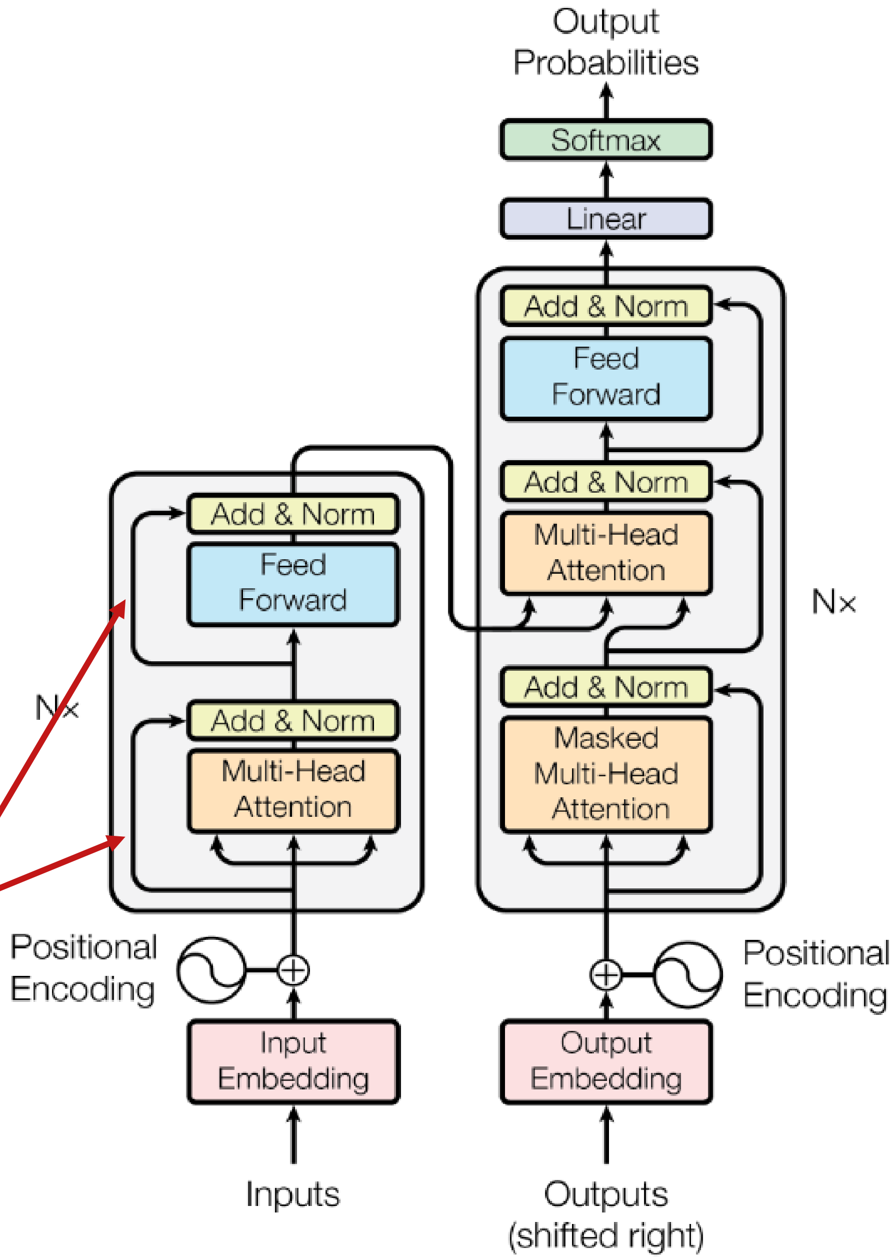
Source of Image : Attention is all you need (Vaswani t al., 2017)

Transformer Architecture



Position embeddings are *added* to each word embedding. Otherwise, since we have no recurrence, our model is unaware of the position of a word in the sequence!

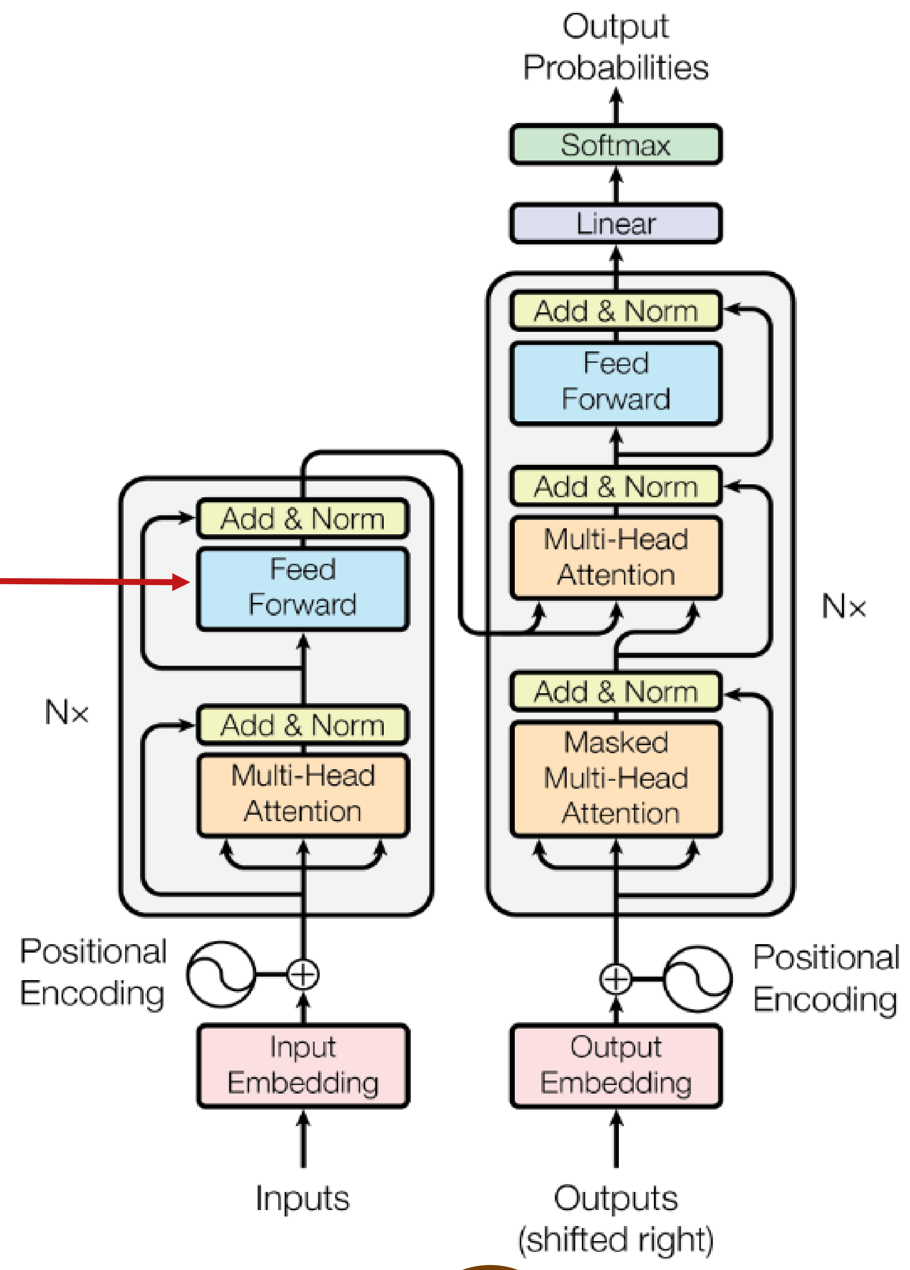
Transformer Architecture



Residual connections, which mean that we add the input to a particular block to its output, help improve gradient flow

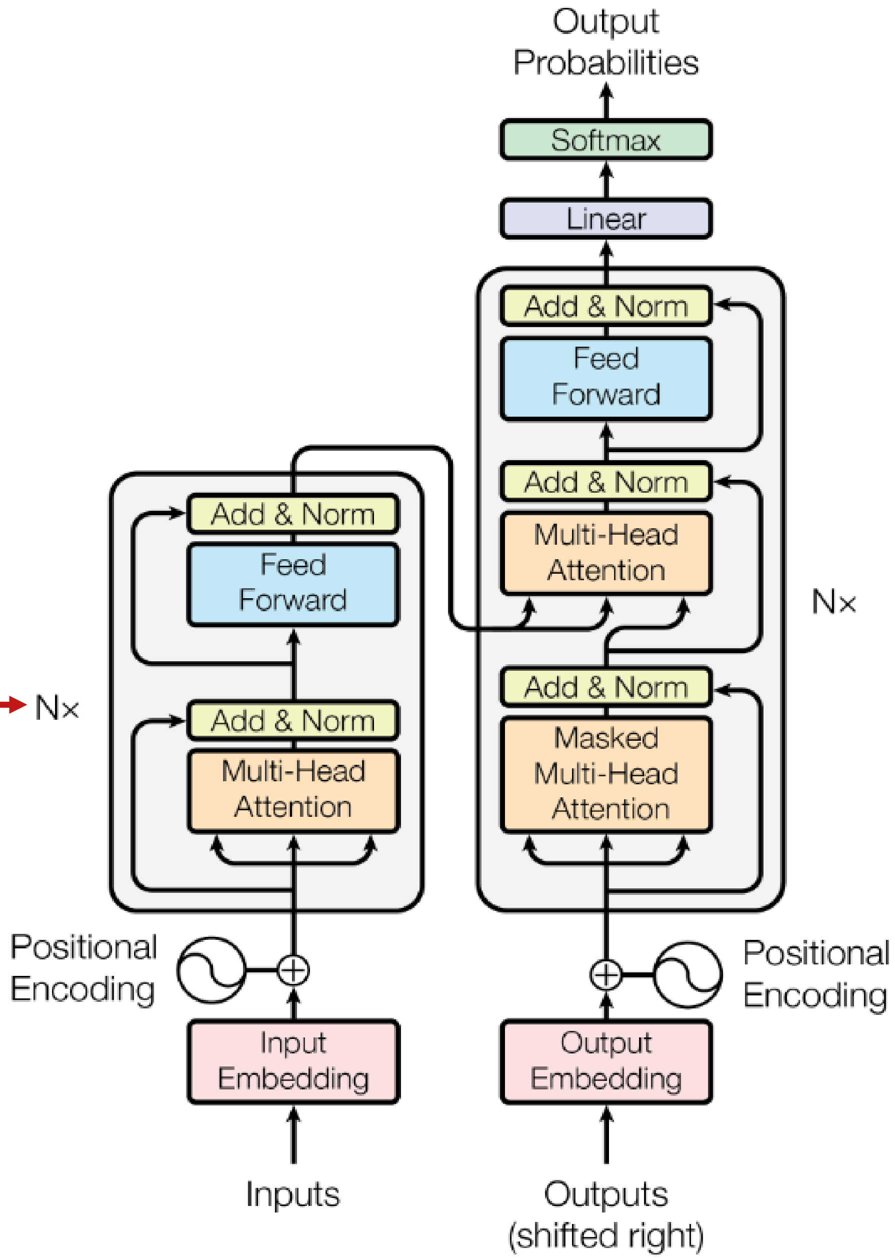
Transformer Architecture

A **feed-forward layer** on top of the attention- weighted averaged value vectors allows us to add more parameters / nonlinearity

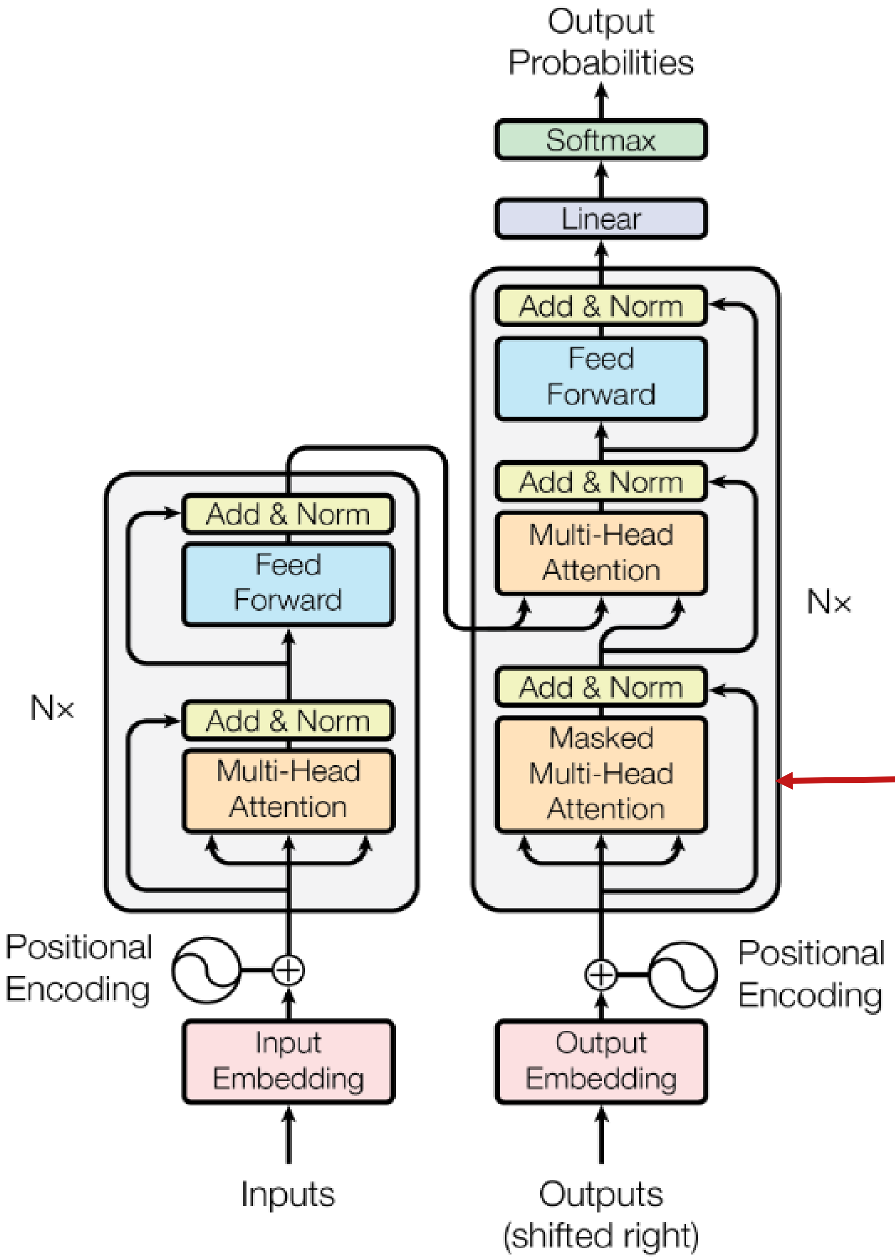


Transformer Architecture

We stack as many of these ***Transformer* blocks** on top of each other as we can (bigger models are generally better given enough data!)

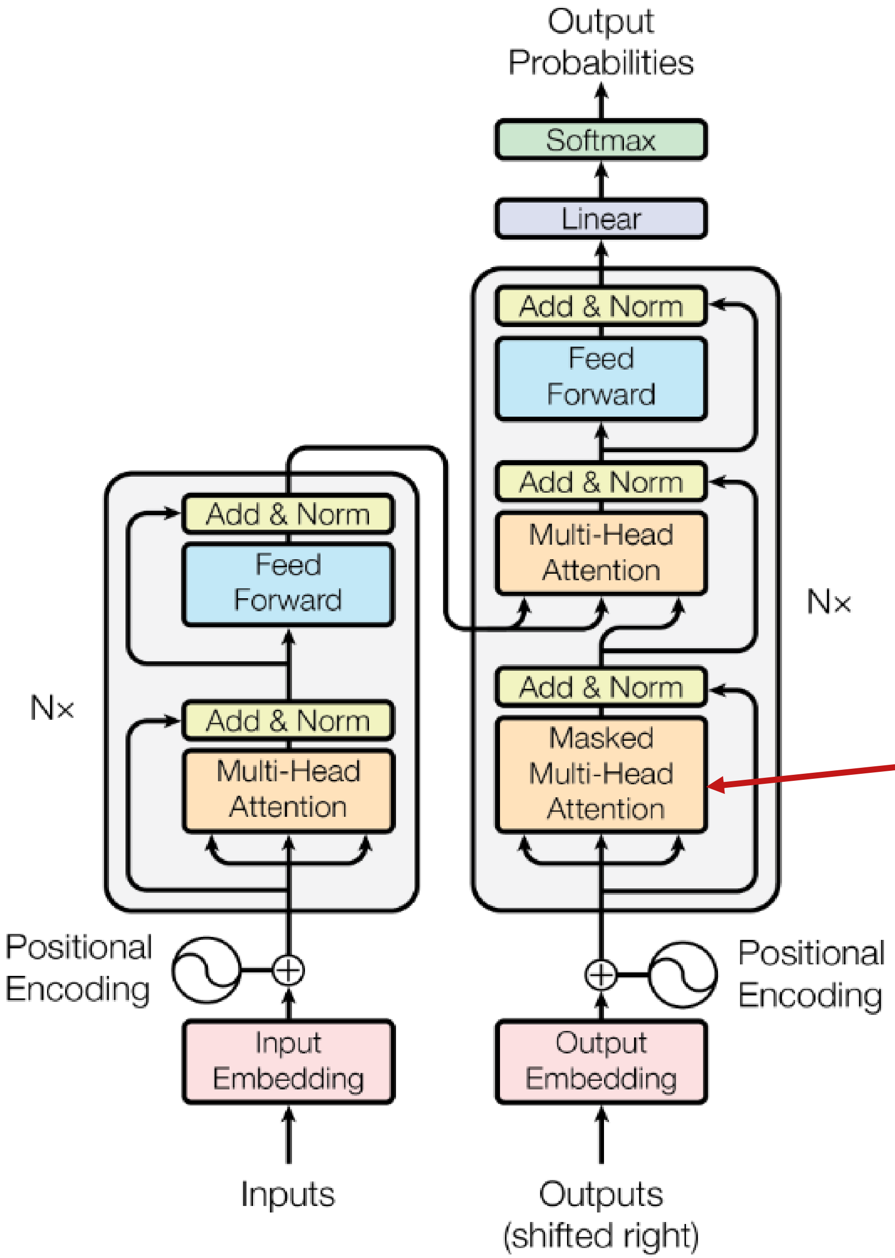


Transformer Architecture



Moving onto the decoder, which takes in English sequences that have been shifted to the right (e.g., *<START> schools opened their*)

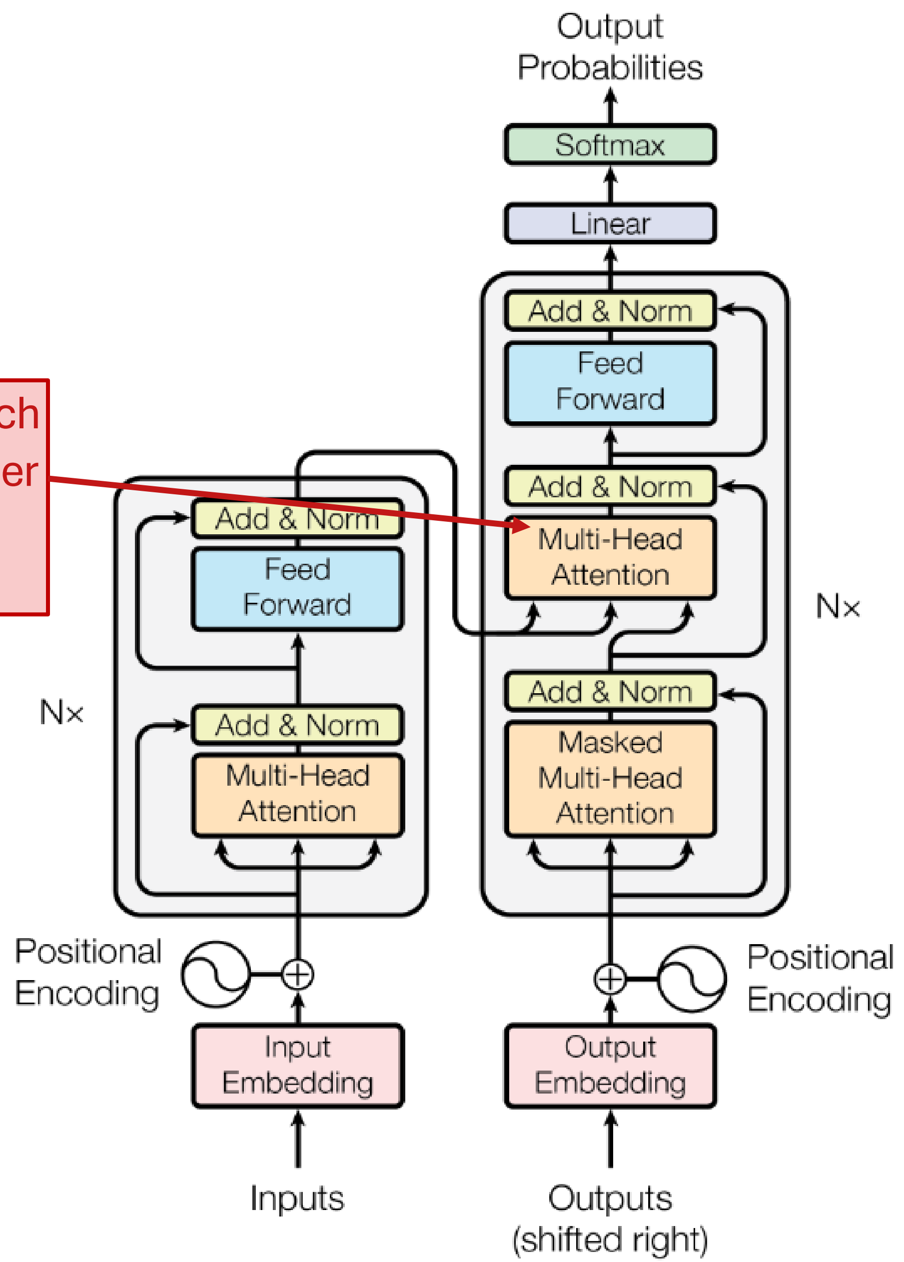
Transformer Architecture



We first have an instance of **masked self attention**. Since the decoder is responsible for predicting the English words, we need to apply masking as we saw before.

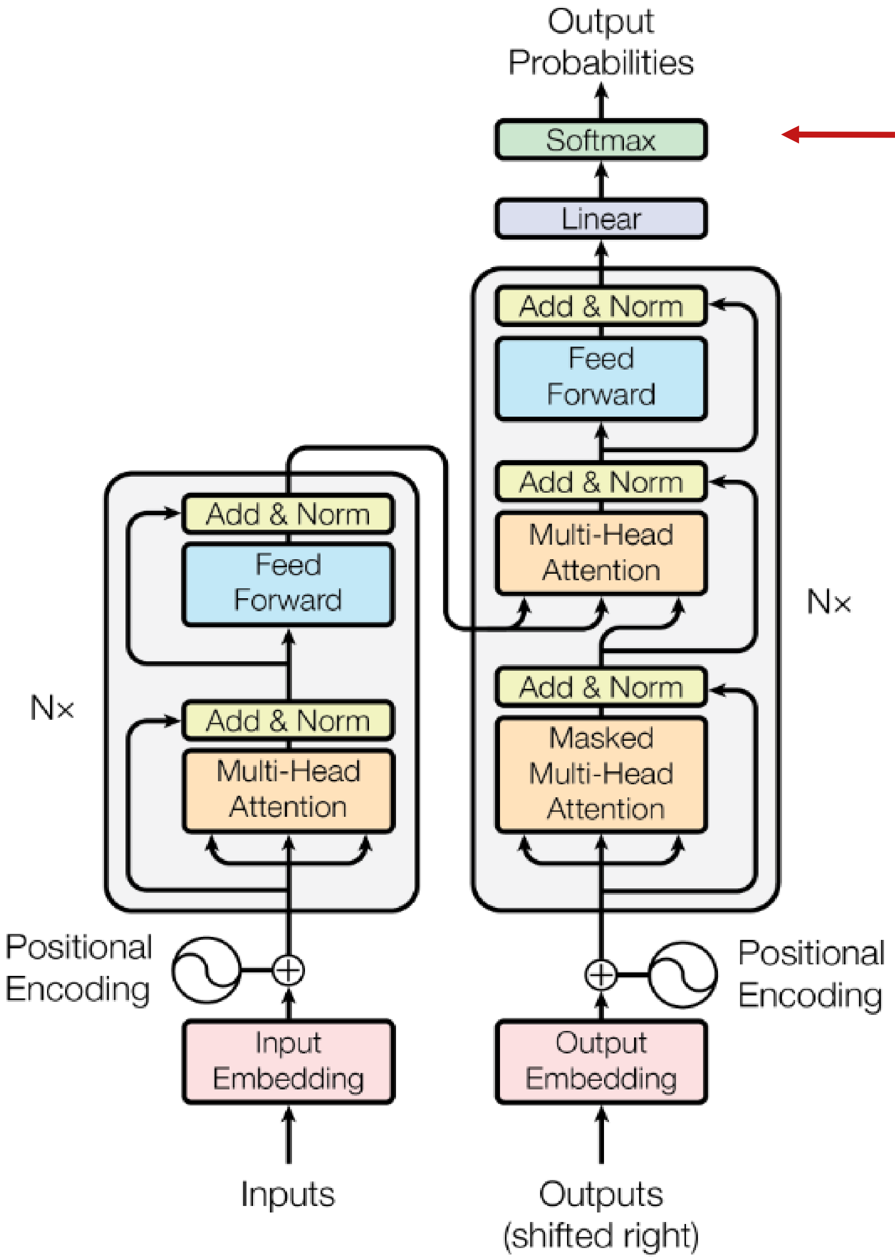
Transformer Architecture

Now, we have ***cross attention***, which connects the decoder to the encoder by enabling it to attend over the encoder's final hidden states.



Source of Image : Attention is all you need (Vaswani et al., 2017)

Transformer Architecture



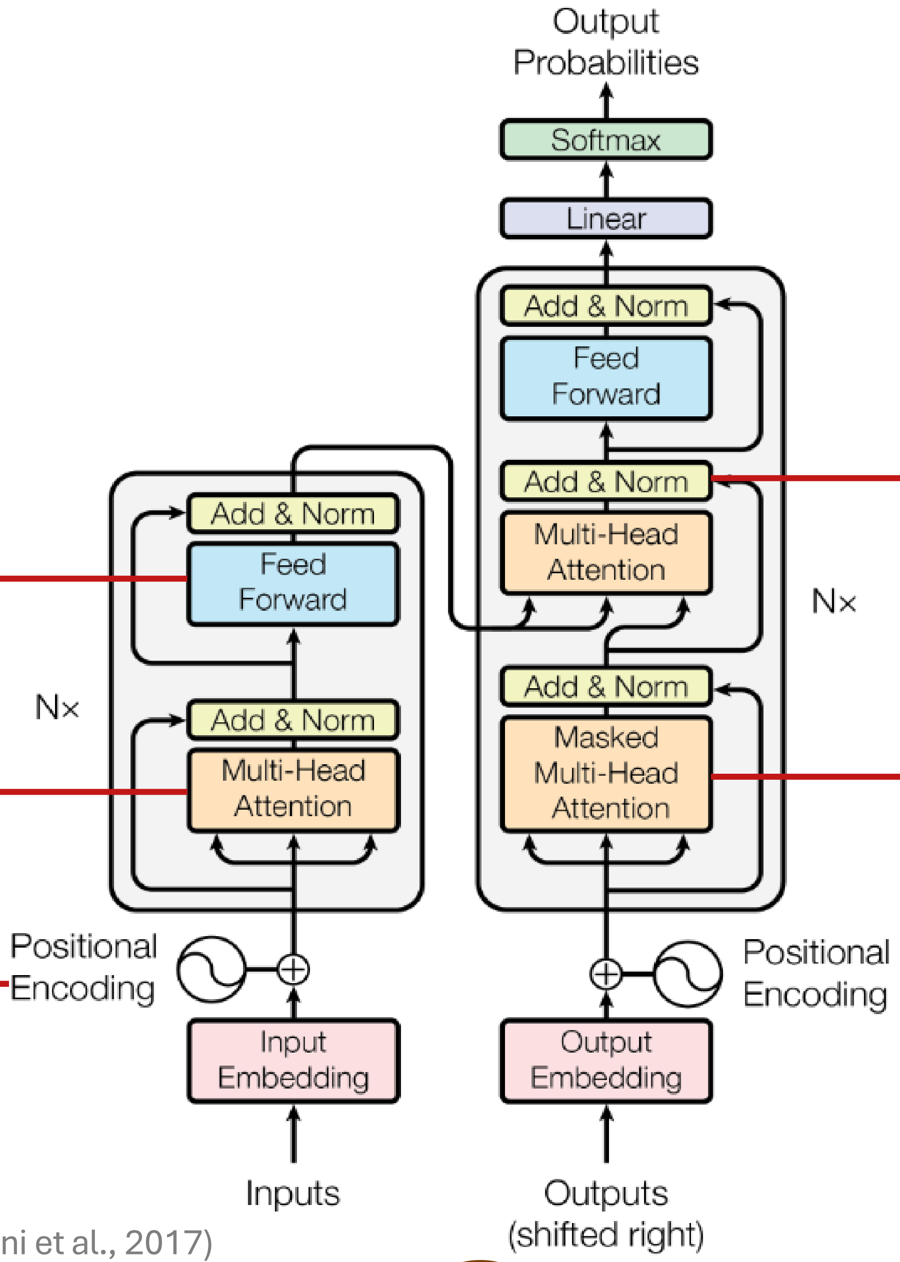
After stacking a bunch of these decoder blocks, we finally have our familiar **softmax** layer to predict the next English word.

Transformer Architecture

Adding non-linearities

Allows querying multiple positions at each layer

Adds positional information



Reduces covariance shift and makes the system stable

Prevents attention lookups into the future while decoding

Source of Image : Attention is all you need (Vaswani et al., 2017)

Layer normalization

- **Main idea:** Batch normalization is quite beneficial, but it's challenging to apply with sequence models. The varying lengths of sequences make it difficult to normalize across a batch. Sequences can be very long, which often results in smaller batch sizes.
- **Solution:** Layer normalization

$a_1, a_2, \ldots, a_B$ ← d-dimensional vectors for each sample in batch

d-dim → μ

$$\mu = \frac{1}{B}\sum_{i=1}^{B} a_i \quad \sigma = \sqrt{\frac{1}{B}\sum_{i=1}^{B}(a_i - \mu)^2}$$

$$\bar{a}_i = \frac{a_i - \mu}{\sigma}\gamma + \beta$$

Batch Norm

different *dimensions* of a → a_j

1-dim → μ

$$\mu = \frac{1}{d}\sum_{i=1}^{d} a_j \quad \sigma = \sqrt{\frac{1}{d}\sum_{i=1}^{d}(a_j - \mu)^2}$$

$$\bar{a} = \frac{a - \mu}{\sigma}\gamma + \beta$$

Layer Norm