

Attention Mechanism

What is Attention?

- Attention is motivated by
 - how we pay visual attention to different regions of an image.
 - Human visual attention allows us to focus on a certain region with “high resolution” while perceiving the surrounding image in “low resolution”.

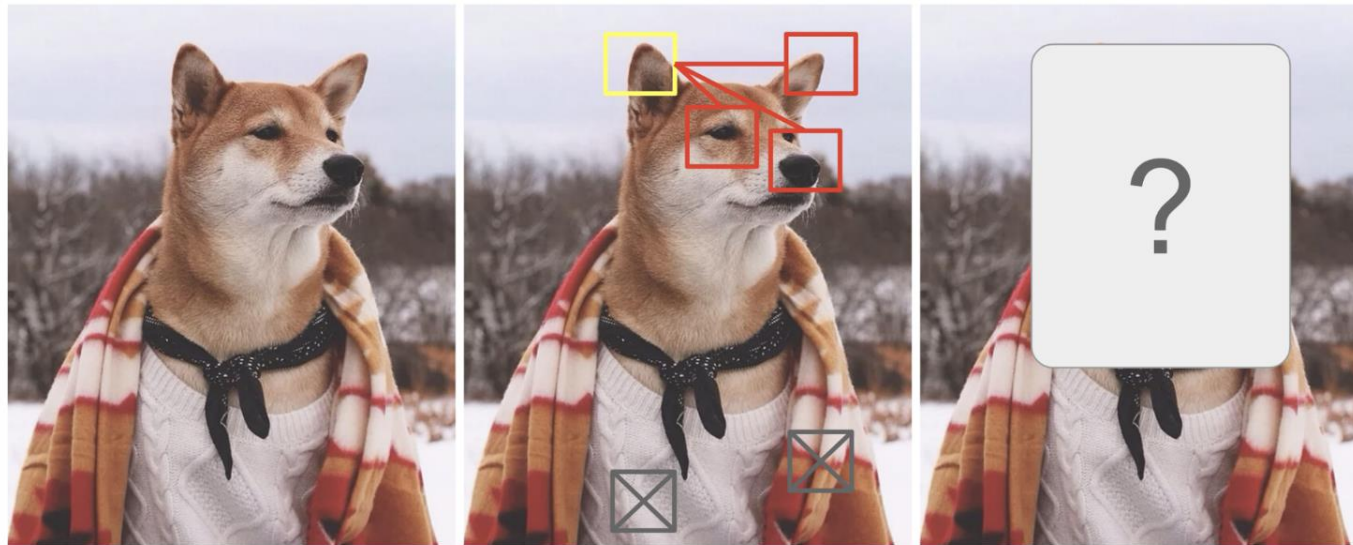
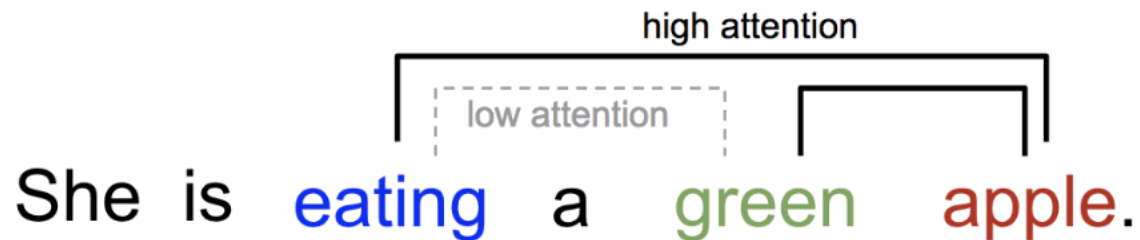


Fig. 1. A Shiba Inu in a men's outfit. The credit of the original photo goes to Instagram [@mensweardog](#).

What is Attention?

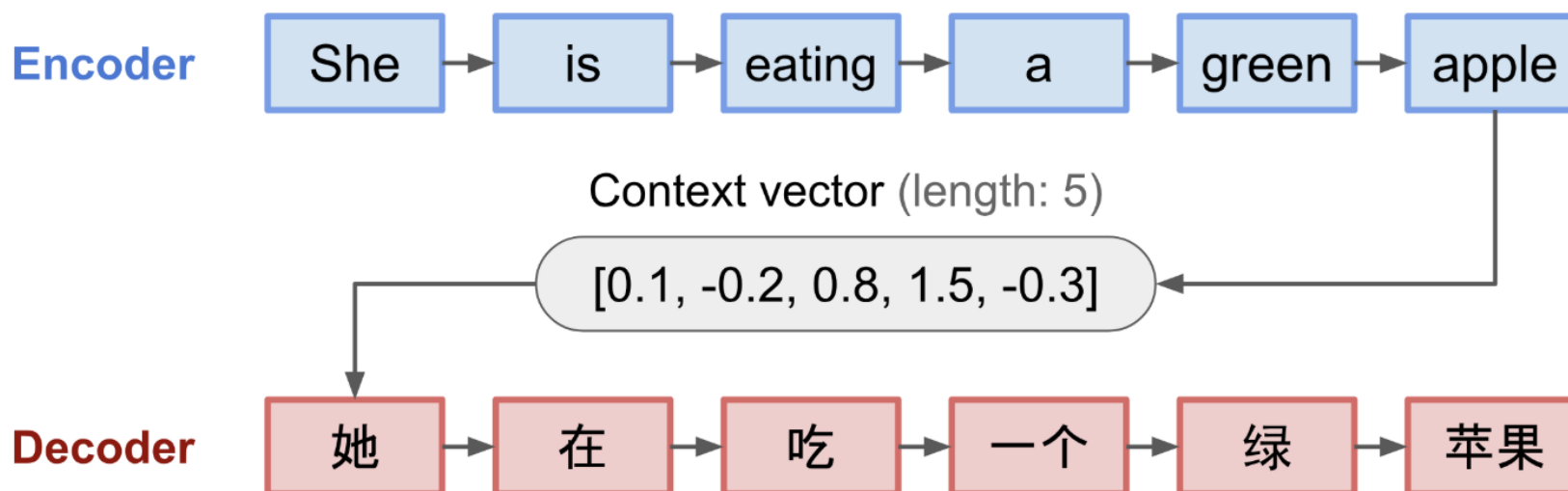
- Similar to image,
 - we can explain the relationship between words in one sentence.



- Attention in deep learning = a vector of importance weights
 - To predict one element (a pixel or a word), we estimate using the attention vector how strongly it is correlated with other elements.

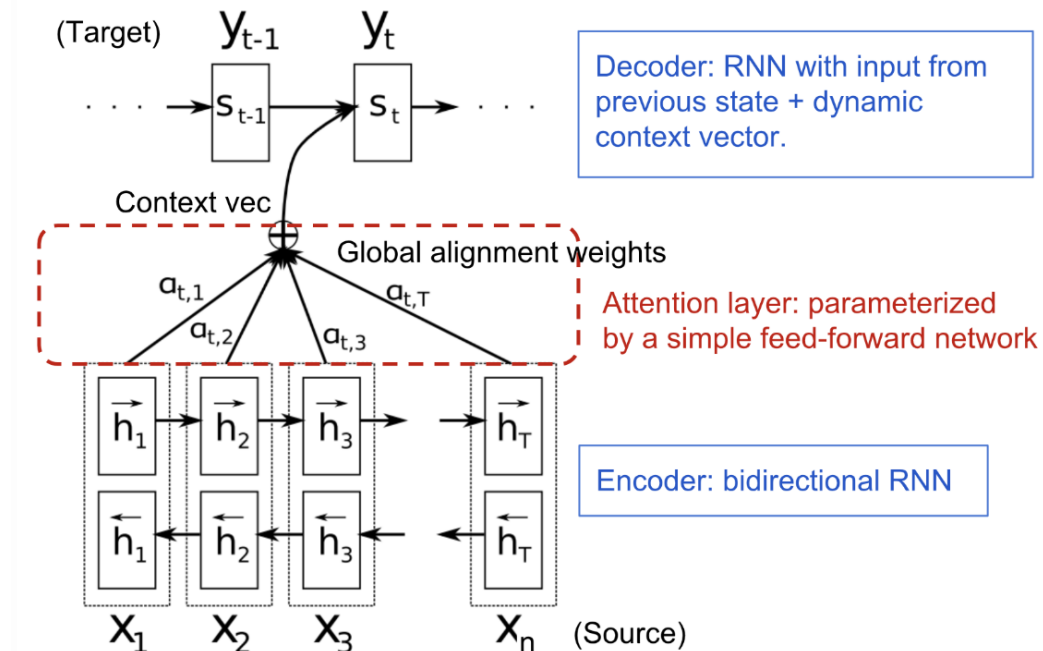
Motivation

- Limitations of Seq2Seq models
 - The fixed size embedding is incapable of remembering long sequences.

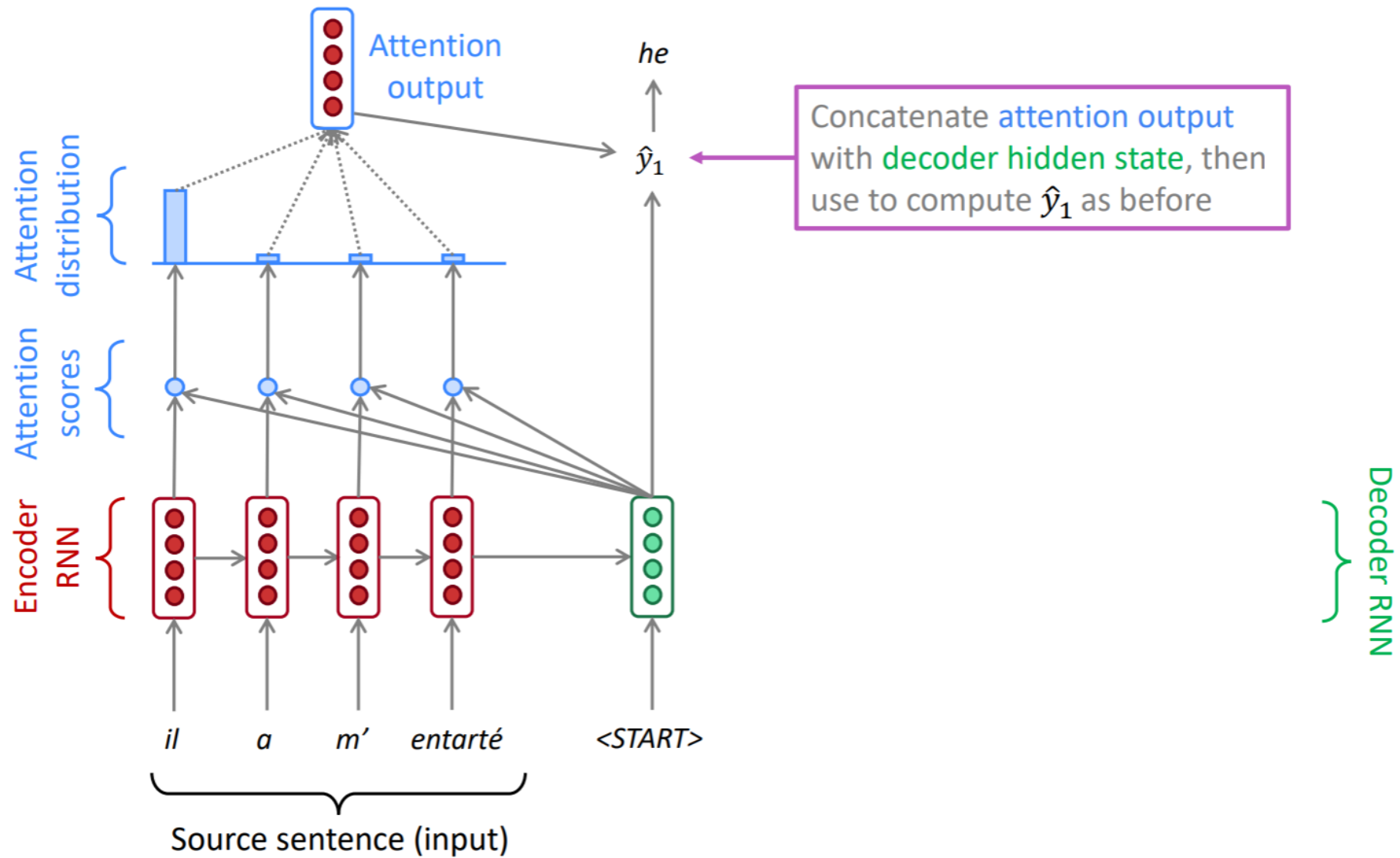


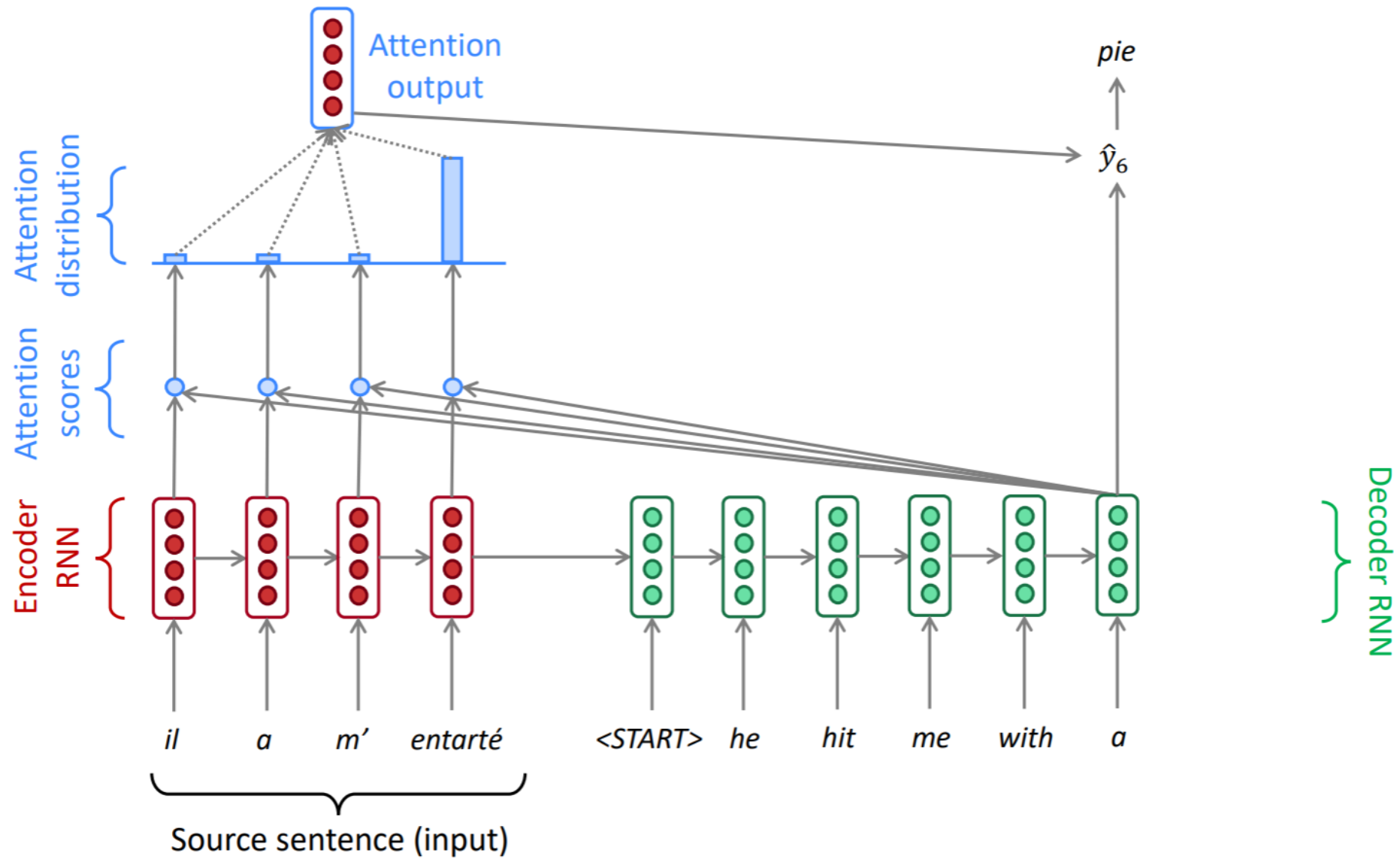
Attention Mechanism

- Instead of decoding from the last hidden state,
 - the context vector (decoder state) is obtained from the entire encoded vectors.
- Decoder state consumes
 - encoder hidden states
 - decoder hidden states
 - alignment between source and target



Additive Attention

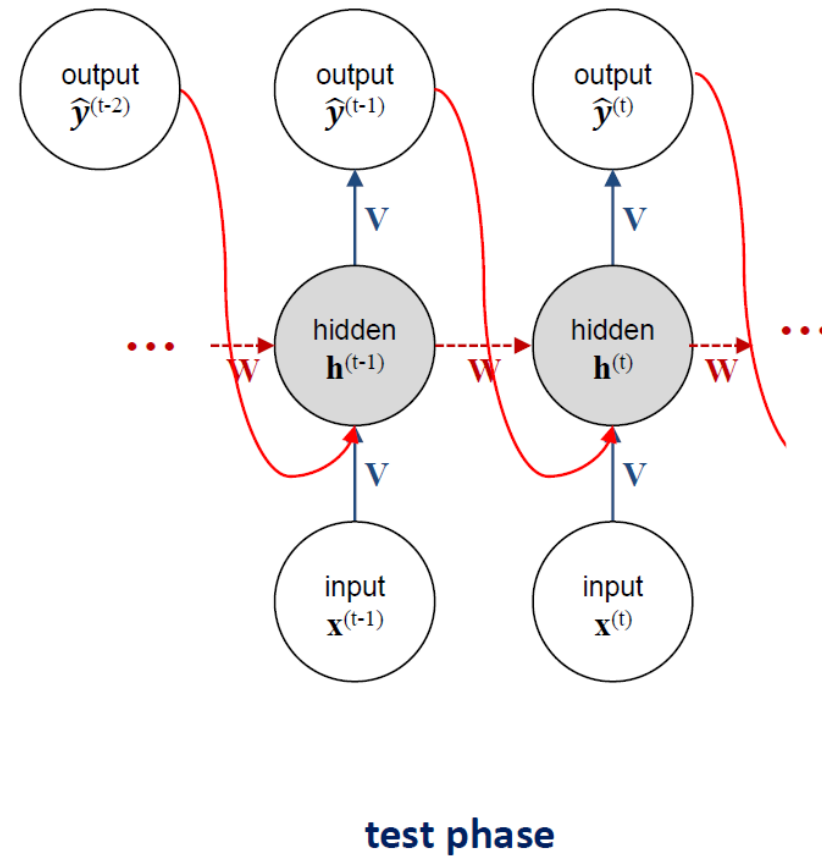
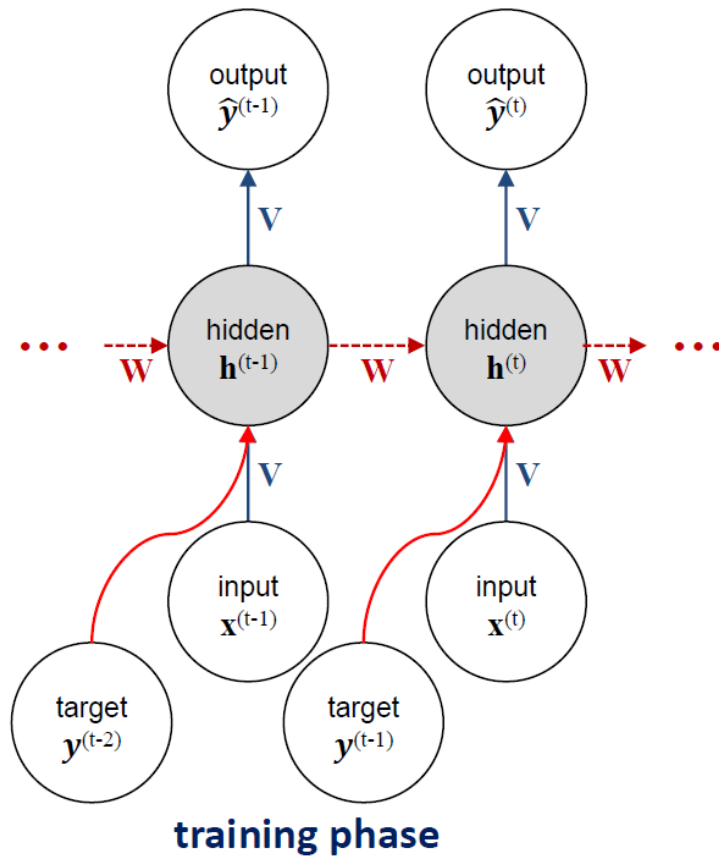




<http://web.stanford.edu/class/cs224n/slides/cs224n-2019-lecture07-fancy-rnn.pdf>

Training – Teacher Forcing

- Teacher forcing: feeding ground-truth tokens during training



- We have encoder hidden states $h_1, \dots, h_N \in \mathbb{R}^h$
- On timestep t , we have decoder hidden state $s_t \in \mathbb{R}^h$
- We get the attention scores e^t for this step:

$$e^t = [s_t^T h_1, \dots, s_t^T h_N] \in \mathbb{R}^N$$

- We take softmax to get the attention distribution α^t for this step (this is a probability distribution and sums to 1)

$$\alpha^t = \text{softmax}(e^t) \in \mathbb{R}^N$$

- We use α^t to take a weighted sum of the encoder hidden states to get the attention output a_t

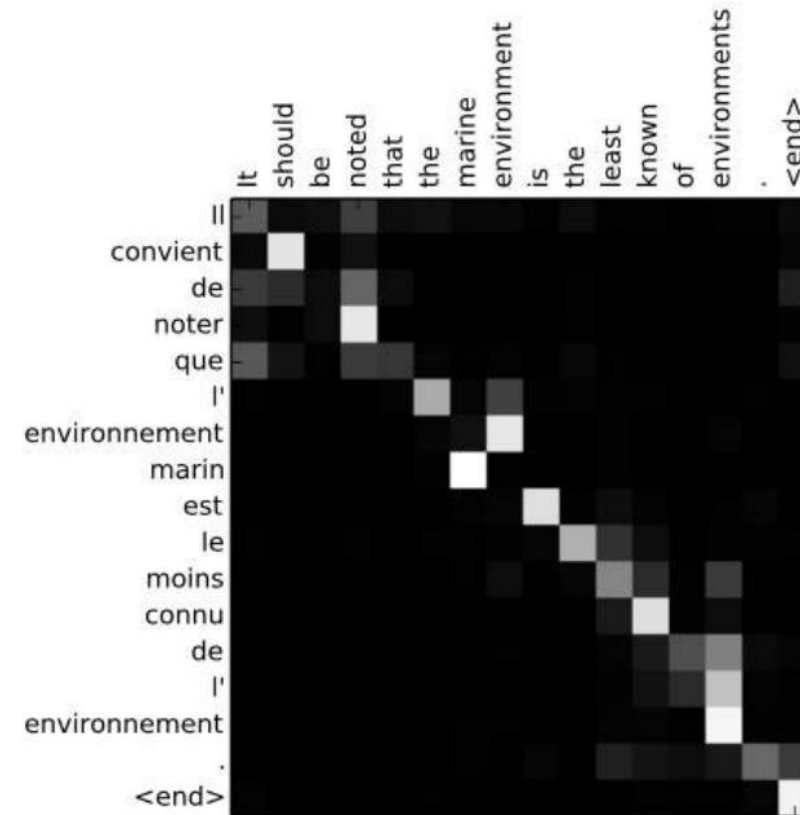
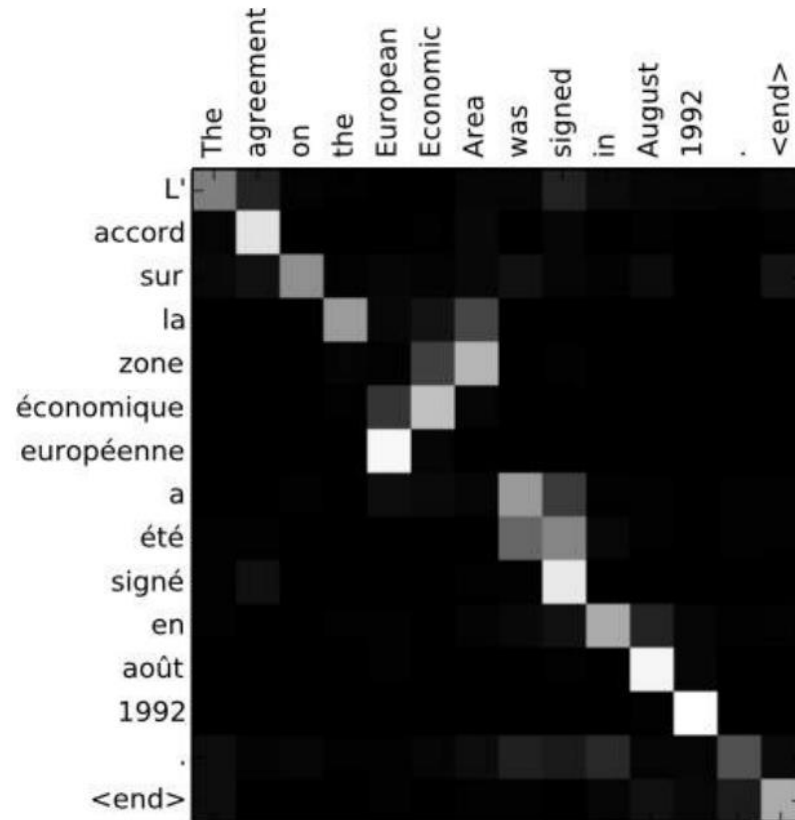
$$a_t = \sum_{i=1}^N \alpha_i^t h_i \in \mathbb{R}^h$$

- Finally we concatenate the attention output a_t with the decoder hidden state s_t and proceed as in the non-attention seq2seq model

$$[a_t; s_t] \in \mathbb{R}^{2h}$$

Attention Mechanism

- The attention matrix

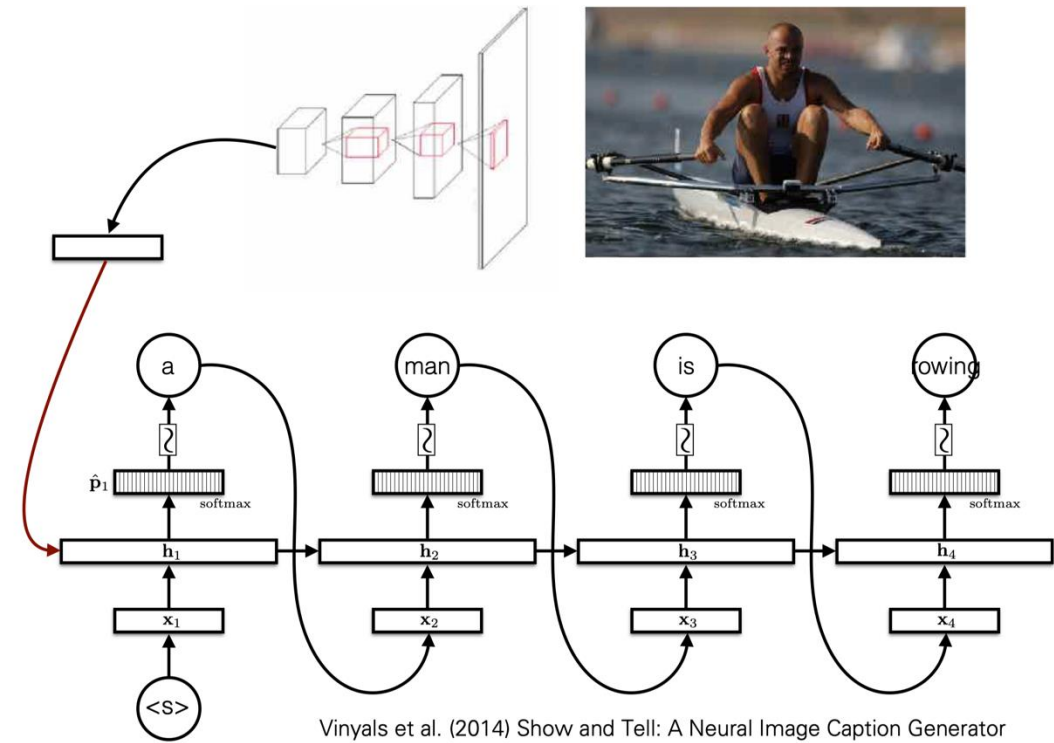
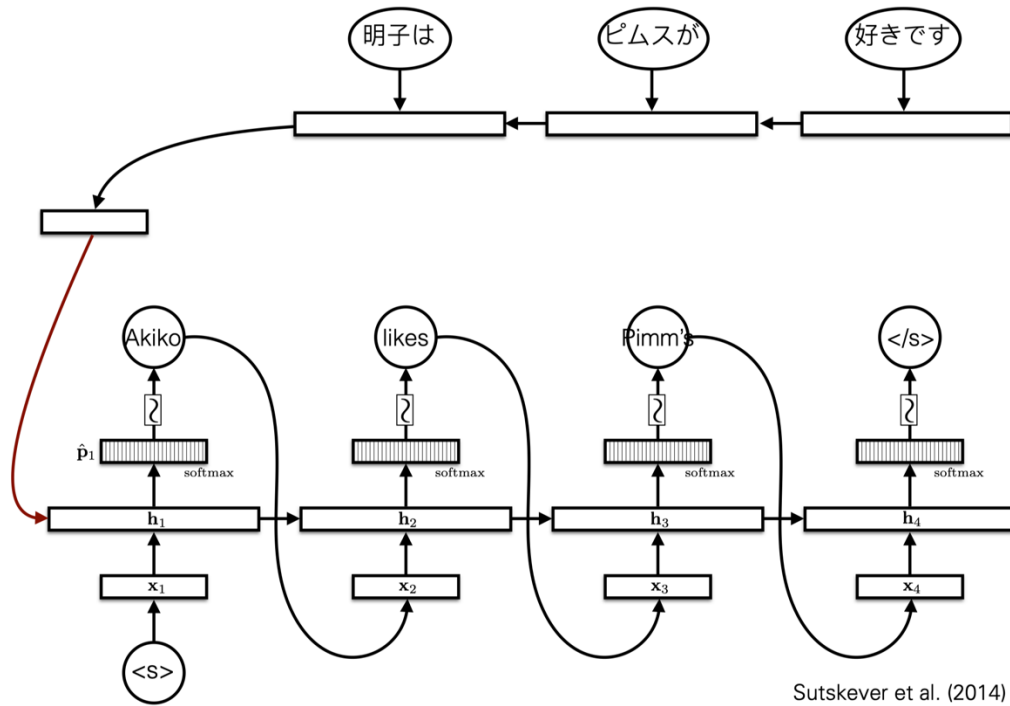


A Family of Attention Mechanisms

<https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html>

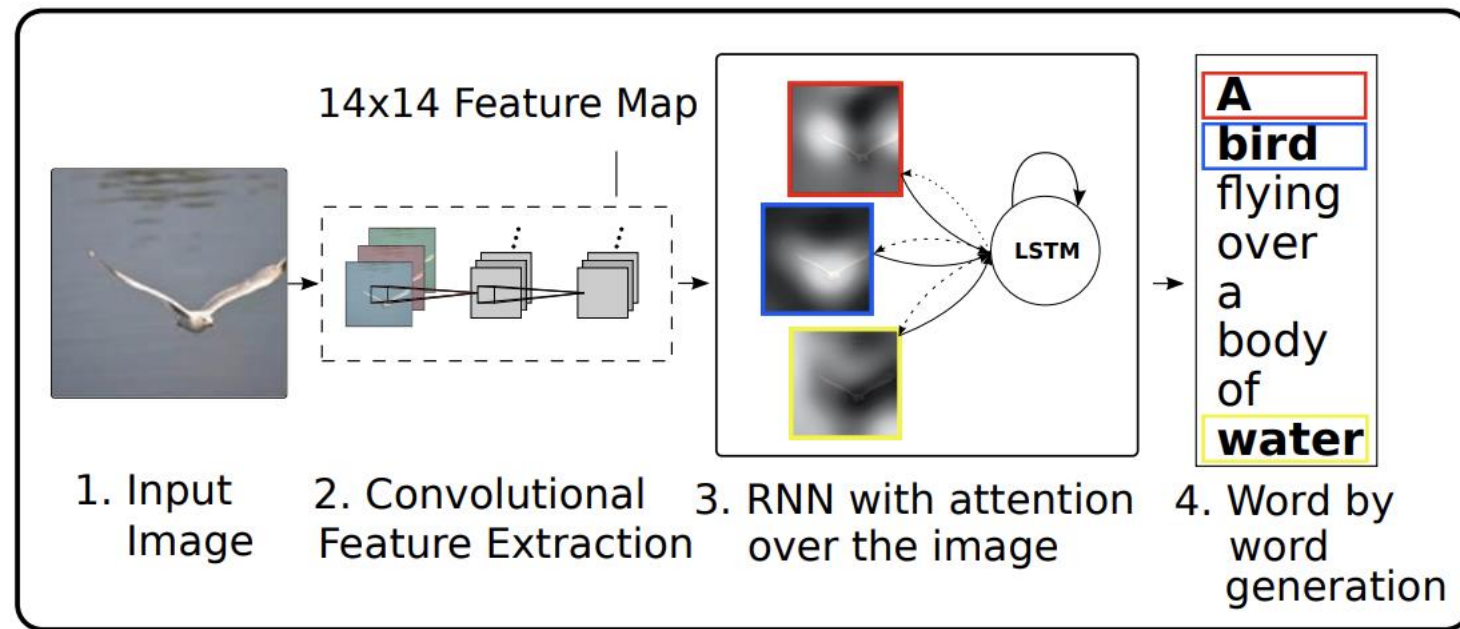
Name	Alignment score function	Citation
Content-base attention	$\text{score}(s_t, h_i) = \text{cosine}[s_t, h_i]$	Graves2014
Additive(*)	$\text{score}(s_t, h_i) = \mathbf{v}_a^\top \tanh(\mathbf{W}_a[s_t; h_i])$	Bahdanau2015
Location-Base	$\alpha_{t,i} = \text{softmax}(\mathbf{W}_a s_t)$ Note: This simplifies the softmax alignment to only depend on the target position.	Luong2015
General	$\text{score}(s_t, h_i) = s_t^\top \mathbf{W}_a h_i$ where \mathbf{W}_a is a trainable weight matrix in the attention layer.	Luong2015
Dot-Product	$\text{score}(s_t, h_i) = s_t^\top h_i$	Luong2015
Scaled Dot-Product(^)	$\text{score}(s_t, h_i) = \frac{s_t^\top h_i}{\sqrt{n}}$ Note: very similar to the dot-product attention except for a scaling factor; where n is the dimension of the source hidden state.	Vaswani2017
Self-Attention(&)	Relating different positions of the same input sequence. Theoretically the self-attention can adopt any score functions above, but just replace the target sequence with the same input sequence.	Cheng2016
Global/Soft	Attending to the entire input state space.	Xu2015
Local/Hard	Attending to the part of input state space; i.e. a patch of the input image.	Xu2015; Luong2015

Caption Generation with Attention (Xu et al. 2015)

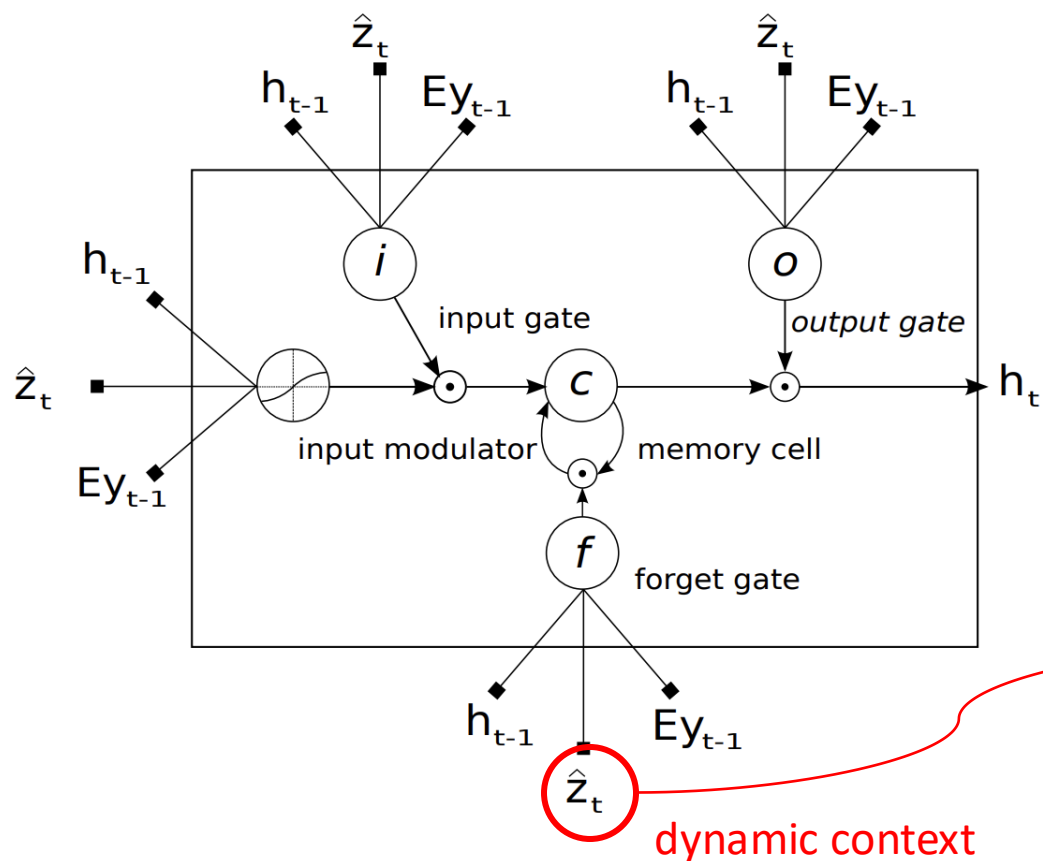


Caption Generation with Attention (Xu et al. 2015)

- Image caption generation with attention mechanism



Caption Generation with Attention (Xu et al. 2015)



$$\begin{pmatrix} \mathbf{i}_t \\ \mathbf{f}_t \\ \mathbf{o}_t \\ \mathbf{g}_t \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} T_{D+m+n,n} \begin{pmatrix} \mathbf{E}\mathbf{y}_{t-1} \\ \mathbf{h}_{t-1} \\ \hat{\mathbf{z}}_t \end{pmatrix} \quad (1)$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{g}_t \quad (2)$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t). \quad (3)$$

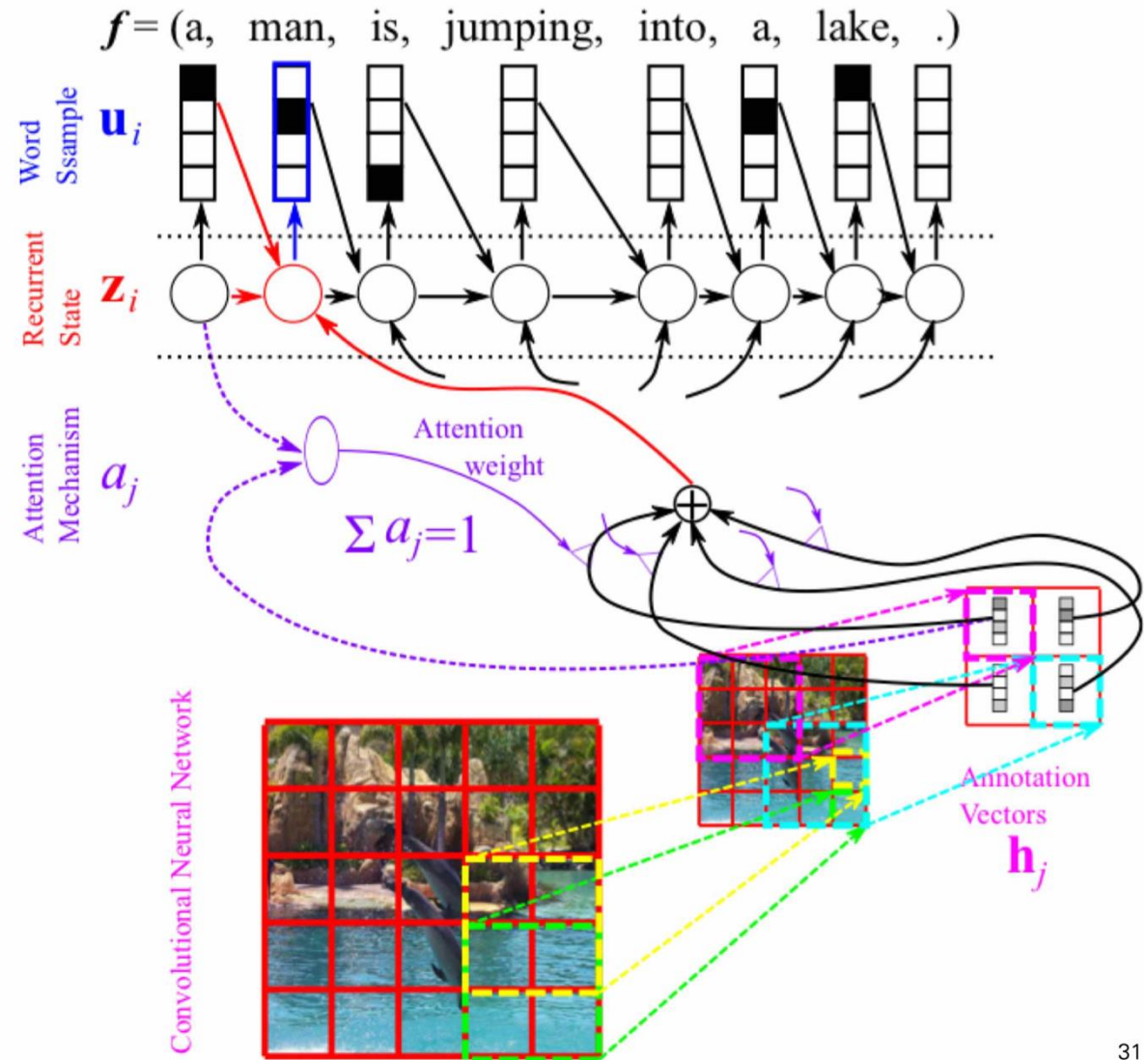
$$e_{ti} = f_{\text{att}}(\mathbf{a}_i, \mathbf{h}_{t-1})$$

$$\alpha_{ti} = \frac{\exp(e_{ti})}{\sum_{k=1}^L \exp(e_{tk})}.$$

$$\hat{\mathbf{z}}_t = \phi(\{\mathbf{a}_i\}, \{\alpha_i\}) \quad \leftarrow$$

Caption Generation with Attention (Xu et al. 2015)

- How the attention works



Caption Generation with Attention (Xu et al. 2015)

- Examples (good)



A woman is throwing a frisbee in a park.



A dog is standing on a hardwood floor.



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



A little girl sitting on a bed with a teddy bear.



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



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

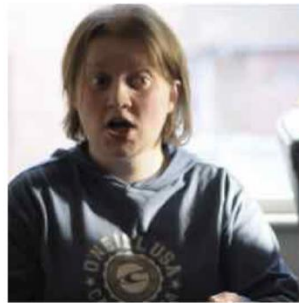


Caption Generation with Attention (Xu et al. 2015)

- Examples (bad)



A large white bird standing in a forest.



A woman holding a clock in her hand.



A man wearing a hat and a hat on a skateboard.



A person is standing on a beach with a surfboard.



A woman is sitting at a table with a large pizza.



A man is talking on his cell phone while another man watches.

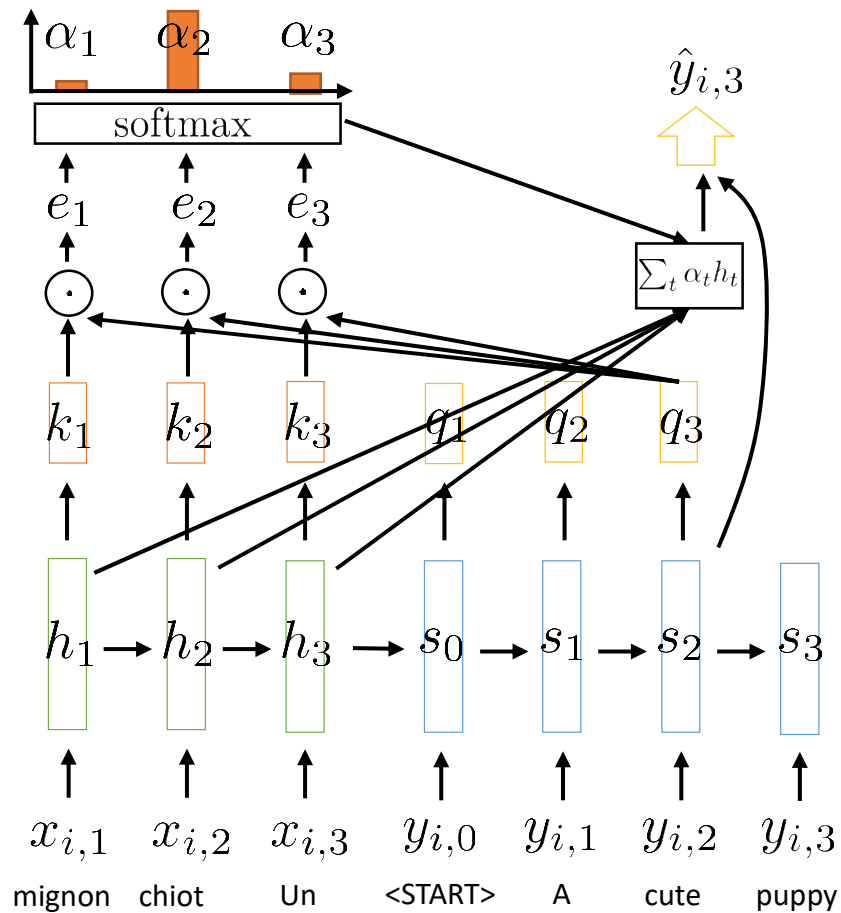


Transformers

Disclaimer: this material came from [CS W182 / 282A at UC Berkeley](#)

Is Attention All We Need?

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 **every** time step

Can in principle do **everything** that recurrence can, and more!

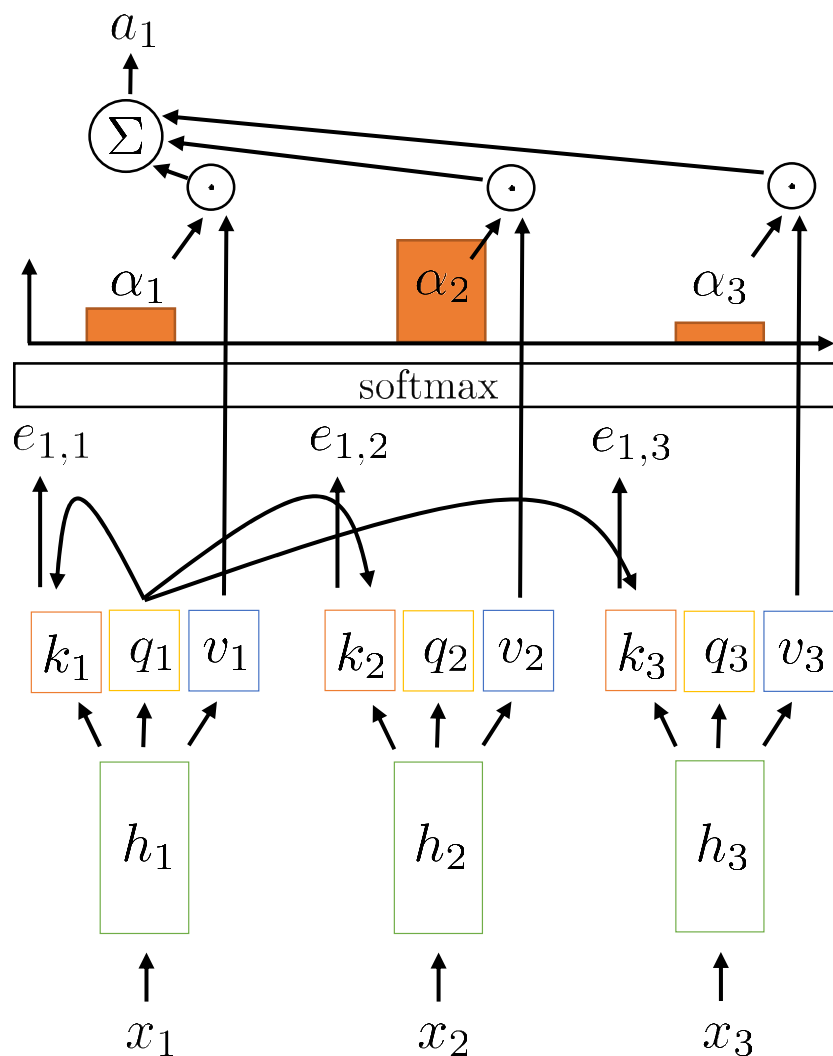
This has a few issues we must overcome:

Problem 1: now step $l = 2$ can't access s_1 or s_0

The encoder has no temporal dependencies at all!

We **must** fix this first

Self-Attention



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

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

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

we'll see why this is important soon

$v_t = v(h_t)$ 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!

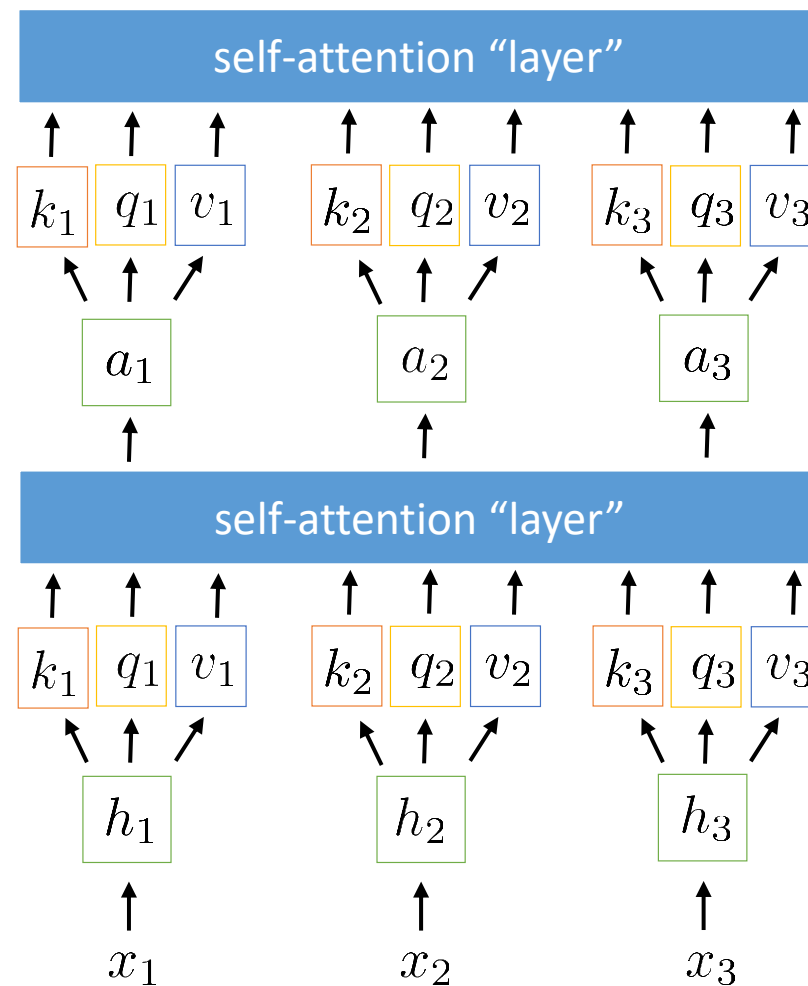
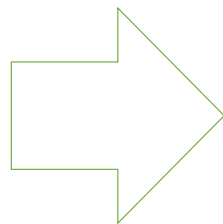
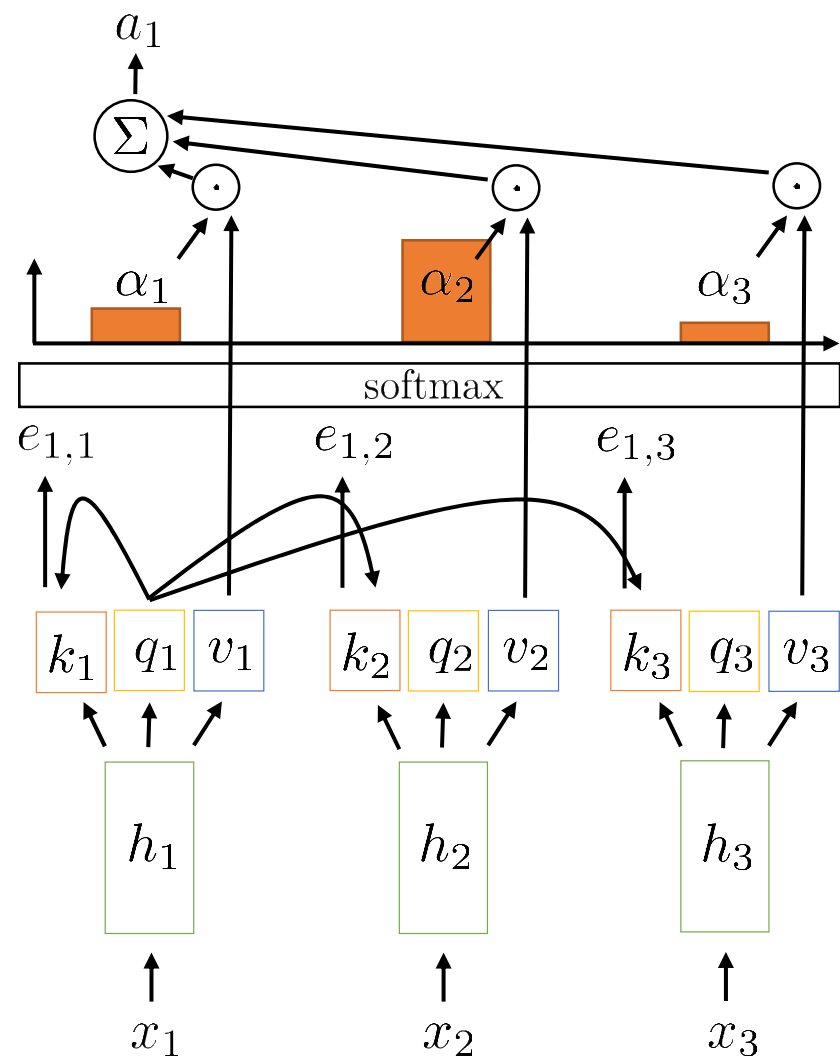
but still weight sharing:

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

shared weights at all time steps

(or any other nonlinear function)

Self-Attention



↑ keep repeating until we've processed this enough
at the end, somehow decode it into an answer (more on this later)

From Self-Attention to Transformers

The basic concept of **self-attention** can be used to develop a very powerful type of sequence model, called a **transformer**

But to make this actually work, we need to develop a few additional components to address some fundamental limitations

- | | |
|---------------------------|--|
| 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 <i>linear</i> in the previous one |
| 4. Masked decoding | how to prevent attention lookups into the future? |

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

Sequence Models with Self-Attention

From Self-Attention to Transformers

The basic concept of **self-attention** can be used to develop a very powerful type of sequence model, called a **transformer**

But to make this actually work, we need to develop a few additional components to address some fundamental limitations

1. Positional encoding

addresses lack of sequence information

2. Multi-headed attention

allows querying multiple positions at each layer

3. Adding nonlinearities

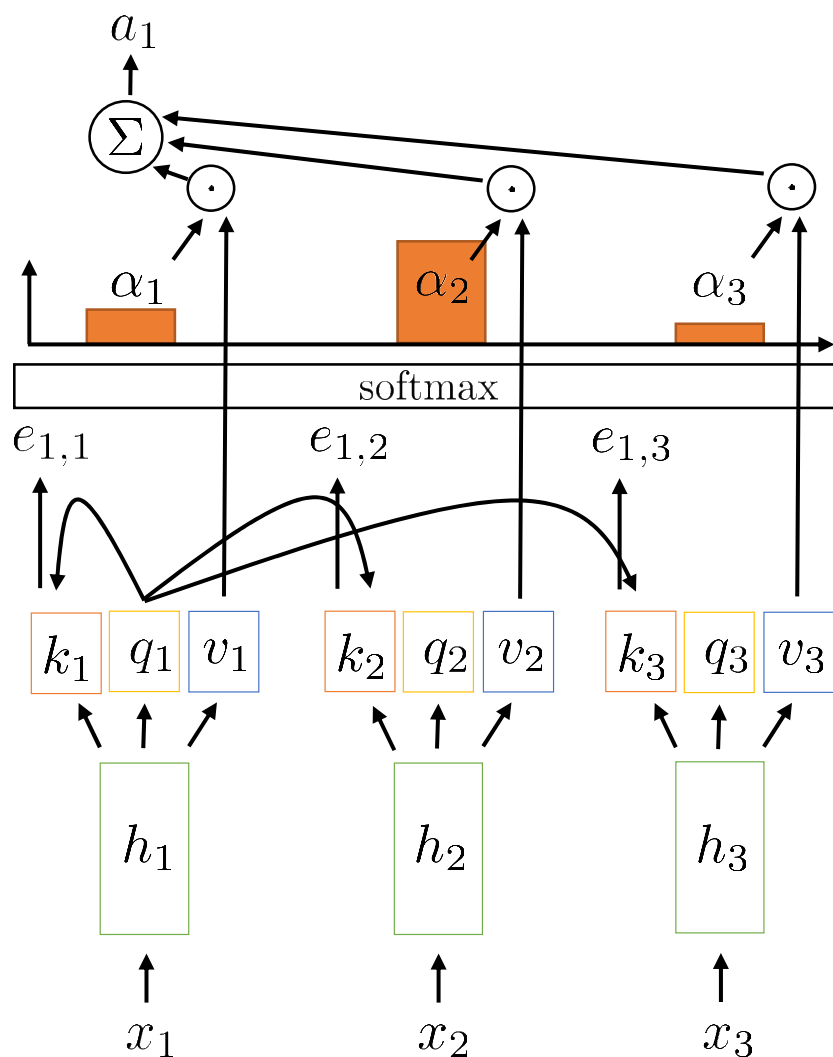
so far, each successive layer is *linear* in the previous one

4. Masked decoding

how to prevent attention lookups into the future?

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

Positional encoding: what is the order?



what we see:

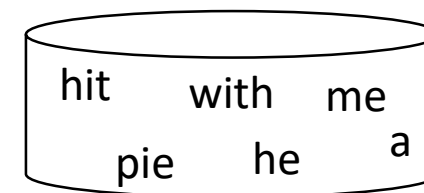
he hit me with a pie

what naïve self-attention sees:

a pie hit me with he

a hit with me he pie

he pie me with a hit



most alternative orderings are nonsense, but some change the meaning

in general the position of words in a sentence carries information!

Idea: add some information to the representation at the beginning that indicates where it is in the sequence!

$$h_t = f(x_t, t)$$

some function

Positional encoding: sin/cos

Naïve positional encoding: just append t to the input $\bar{x}_t = \begin{bmatrix} x_t \\ t \end{bmatrix}$

This is not a great idea, because **absolute** position is less important than **relative** position

I walk my dog every day



every single day I walk my dog



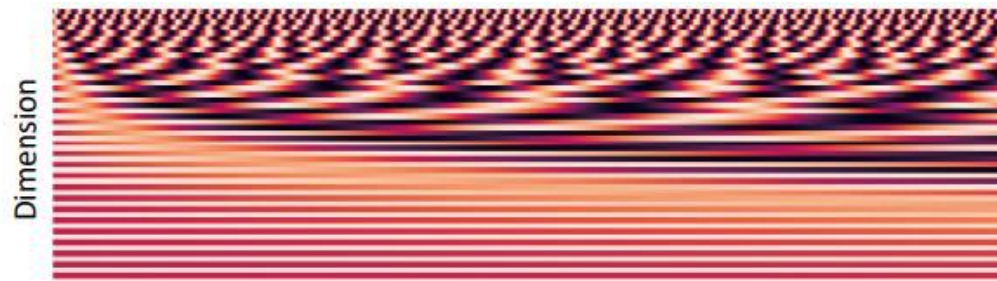
The fact that “my dog” is right after “I walk” is the important part, not its absolute position

we want to represent **position** in a way that tokens with similar **relative** position have similar **positional encoding**

Idea: what if we use **frequency-based** representations?

$$p_t = \begin{bmatrix} \sin(t/10000^{2*1/d}) \\ \cos(t/10000^{2*1/d}) \\ \sin(t/10000^{2*2/d}) \\ \cos(t/10000^{2*2/d}) \\ \dots \\ \sin(t/10000^{2*\frac{d}{2}/d}) \\ \cos(t/10000^{2*\frac{d}{2}/d}) \end{bmatrix}$$

dimensionality
of positional
encoding



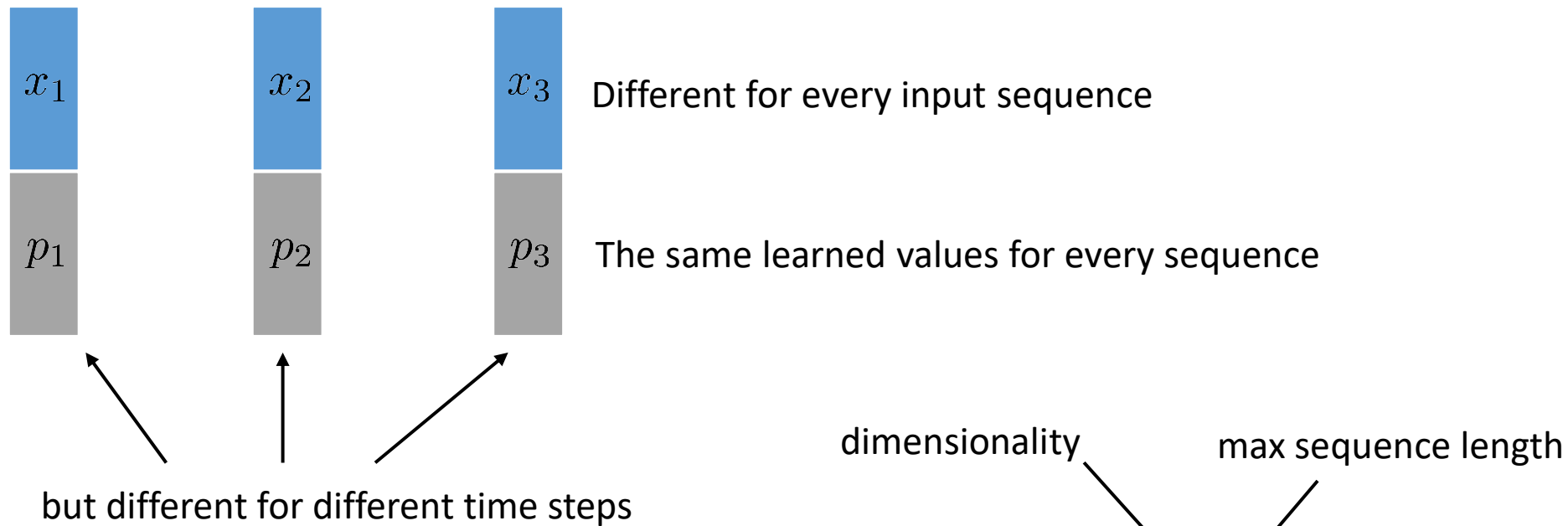
“even-odd” indicator

Index in the sequence

“first-half vs. second-half” indicator

Positional encoding: learned

Another idea: just learn a positional encoding



How many values do we need to learn?

$$P = [p_1, p_2, \dots, p_T] \in R^{d \times T}$$

+ more flexible (and perhaps more optimal) than sin/cos encoding

+ a bit more complex, need to pick a max sequence length (and can't generalize beyond it)

How to incorporate positional encoding?

At each step, we have x_t and p_t

Simple choice: just concatenate them $\bar{x}_t = \begin{bmatrix} x_t \\ p_t \end{bmatrix}$

More often: just add after **embedding** the input

input to self-attention is $\text{emb}(x_t) + p_t$



some learned function (e.g., some fully connected layers with linear layers + nonlinearities)

From Self-Attention to Transformers

The basic concept of **self-attention** can be used to develop a very powerful type of sequence model, called a **transformer**

But to make this actually work, we need to develop a few additional components to address some fundamental limitations

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

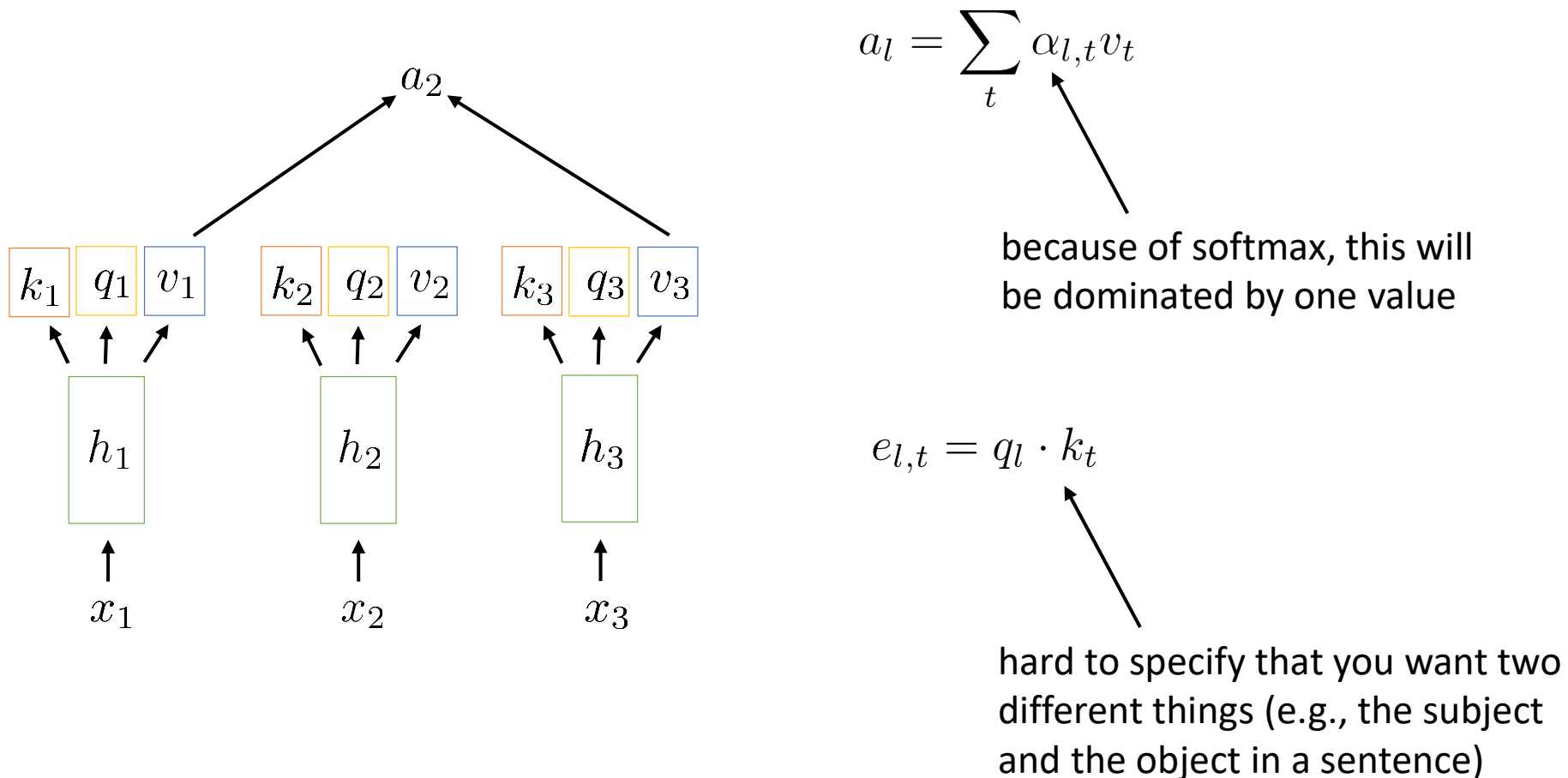
4. Masked decoding

how to prevent attention lookups into the future?

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

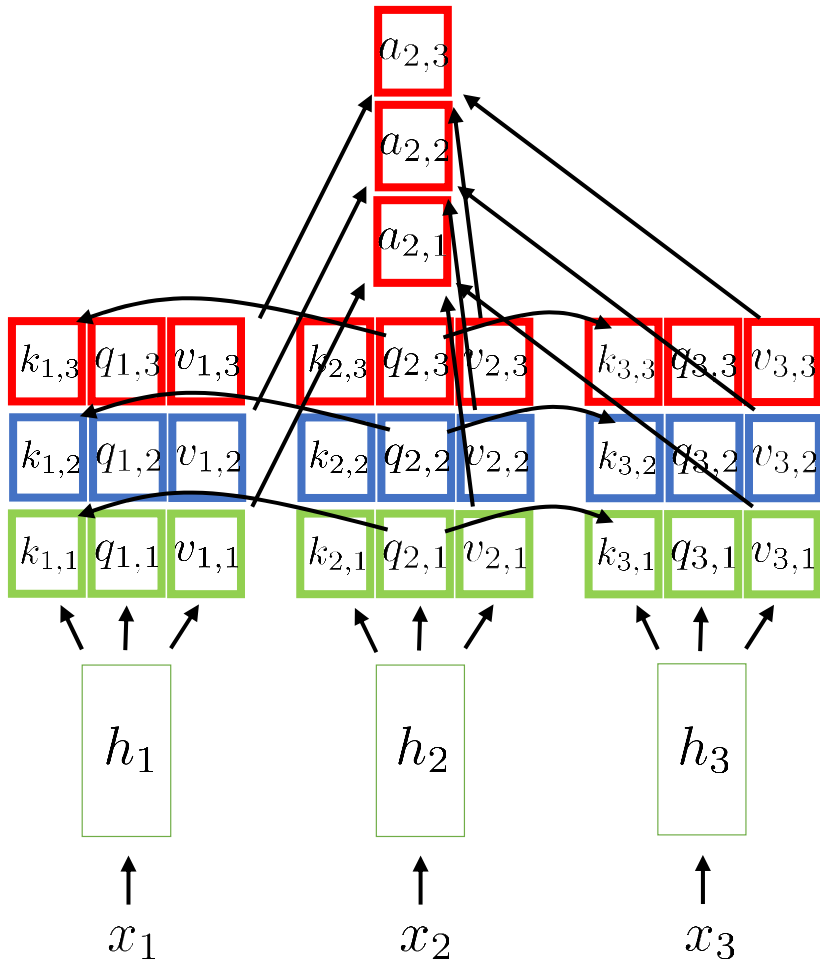
Multi-head attention

Since we are relying **entirely** on attention now, we might want to incorporate **more than one** time step



Multi-head attention

Idea: have multiple keys, queries, and values for every 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

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

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

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

around **8** heads seems to work pretty well for big models

From Self-Attention to Transformers

The basic concept of **self-attention** can be used to develop a very powerful type of sequence model, called a **transformer**

But to make this actually work, we need to develop a few additional components to address some fundamental limitations

1. Positional encoding

addresses lack of sequence information

2. Multi-headed attention

allows querying multiple positions at each layer

3. Adding nonlinearities

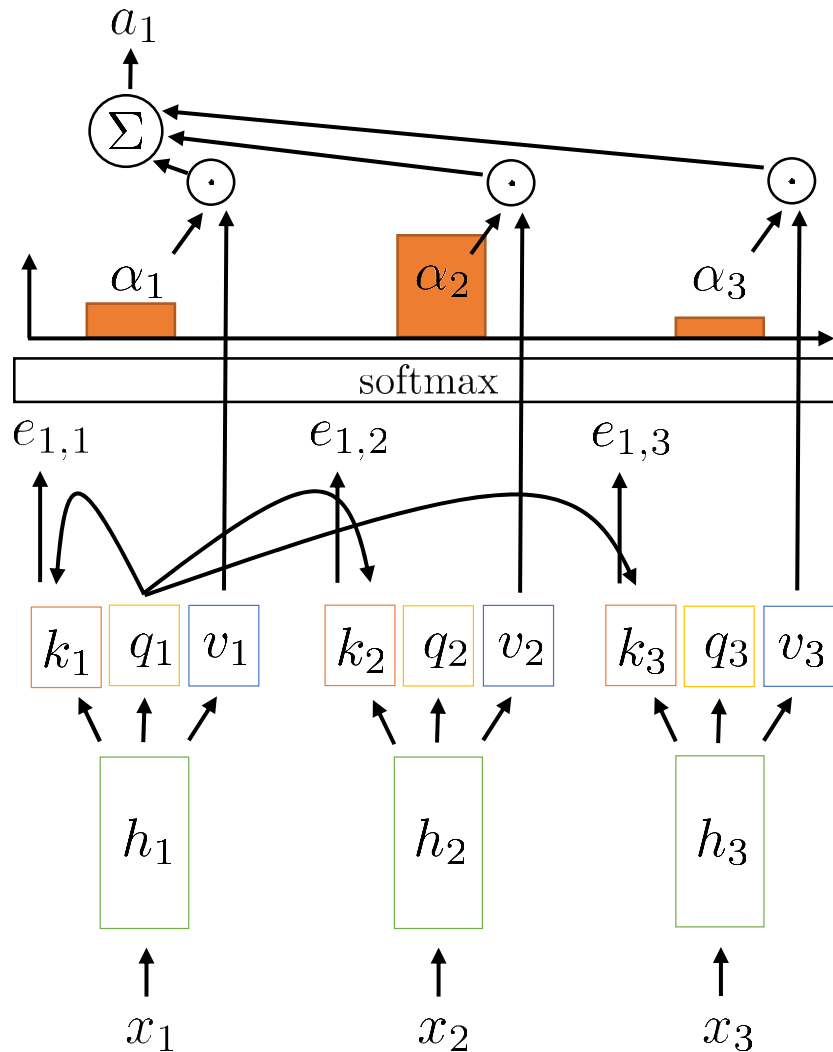
so far, each successive layer is *linear* in the previous one

4. Masked decoding

how to prevent attention lookups into the future?

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

Self-Attention is Linear



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

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

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

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

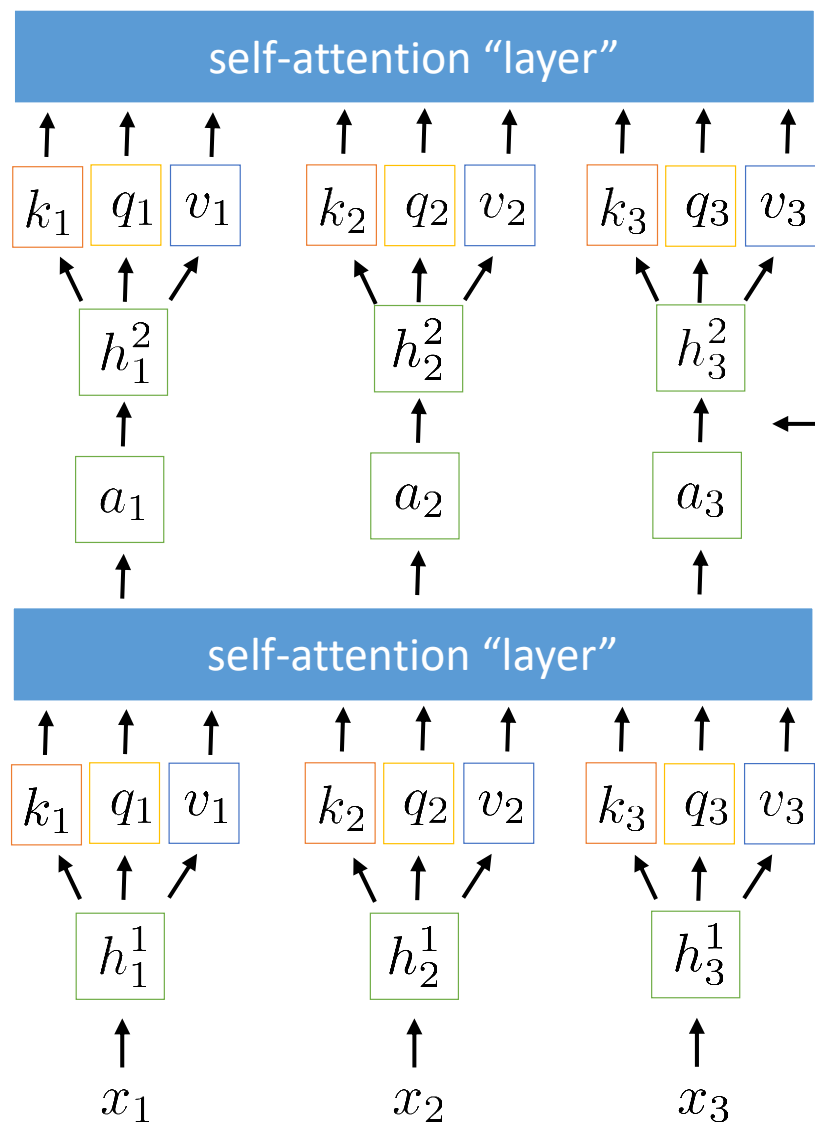
linear transformation

non-linear weights

Every self-attention “layer” is a linear transformation of the previous layer (with non-linear weights)

This is not very expressive

Alternating self-attention & nonlinearity



some non-linear (learned) function
e.g., $h_t^\ell = \sigma(W^\ell a_t^\ell + b^\ell)$

just a neural net applied at every position
after every self-attention layer!

Sometimes referred to as "position-
wise feedforward network"

We'll describe some specific
commonly used choices shortly

From Self-Attention to Transformers

The basic concept of **self-attention** can be used to develop a very powerful type of sequence model, called a **transformer**

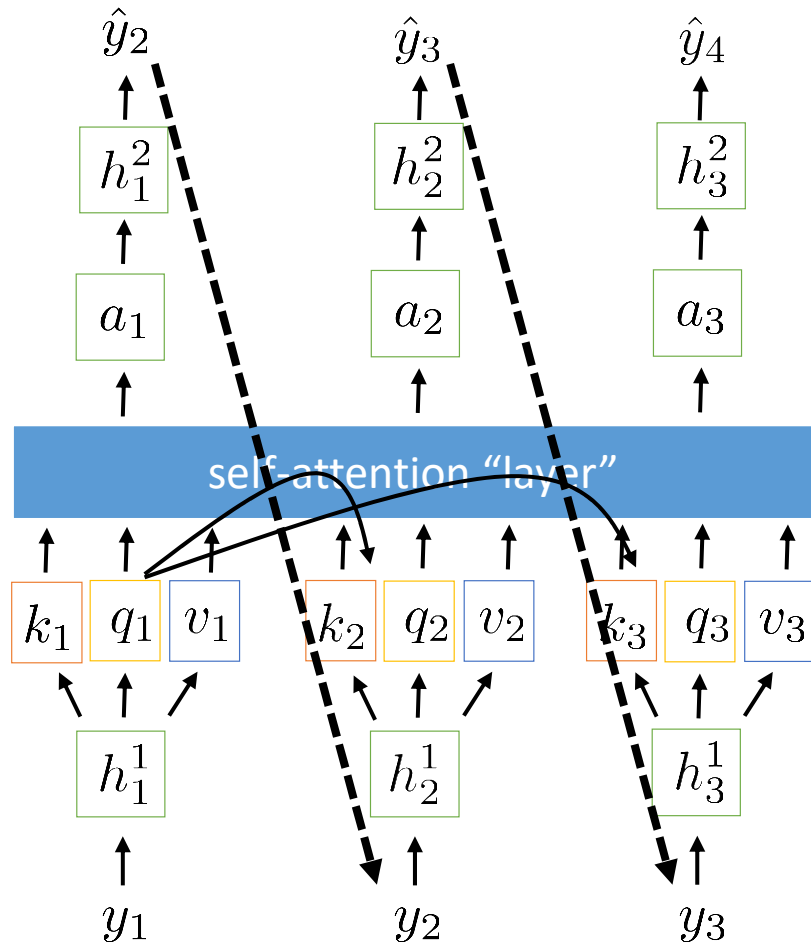
But to make this actually work, we need to develop a few additional components to address some fundamental limitations

- | | |
|---------------------------|--|
| 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 <i>linear</i> in the previous one |
| 4. Masked decoding | how to prevent attention lookups into the future? |

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

Self-attention can see the future!

A **crude** self-attention “language model”:



(in reality, we would have many alternating self-attention layers and position-wise feedforward networks, not just one)

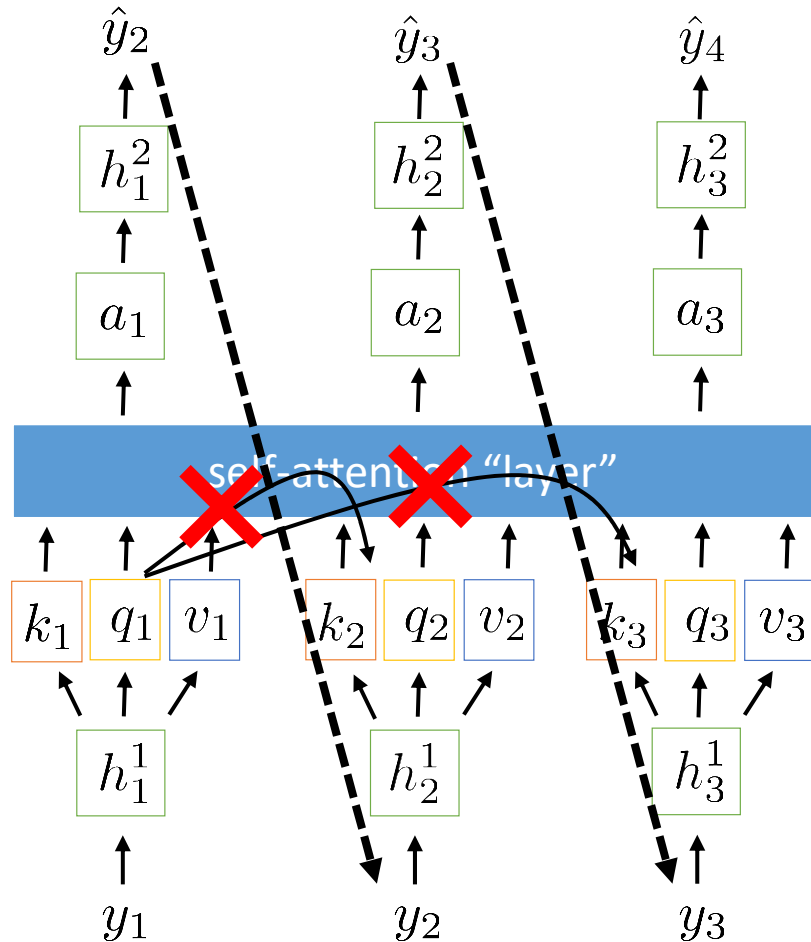
Big problem: self-attention at step 1 can look at the value at steps 2 & 3, which is based on the **inputs** at steps 2 & 3

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:

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

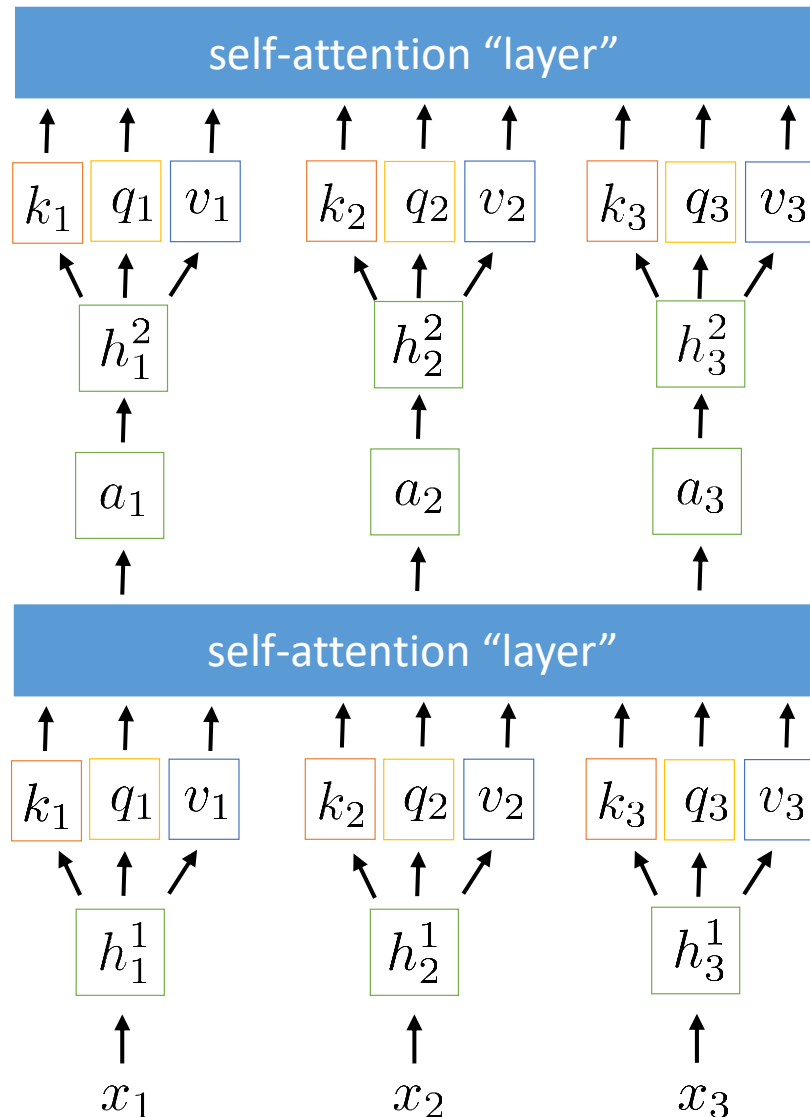
$$e_{l,t} = \begin{cases} q_l \cdot k_t & \text{if } l \geq t \\ -\infty & \text{otherwise} \end{cases}$$

in practice:

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

inside the softmax

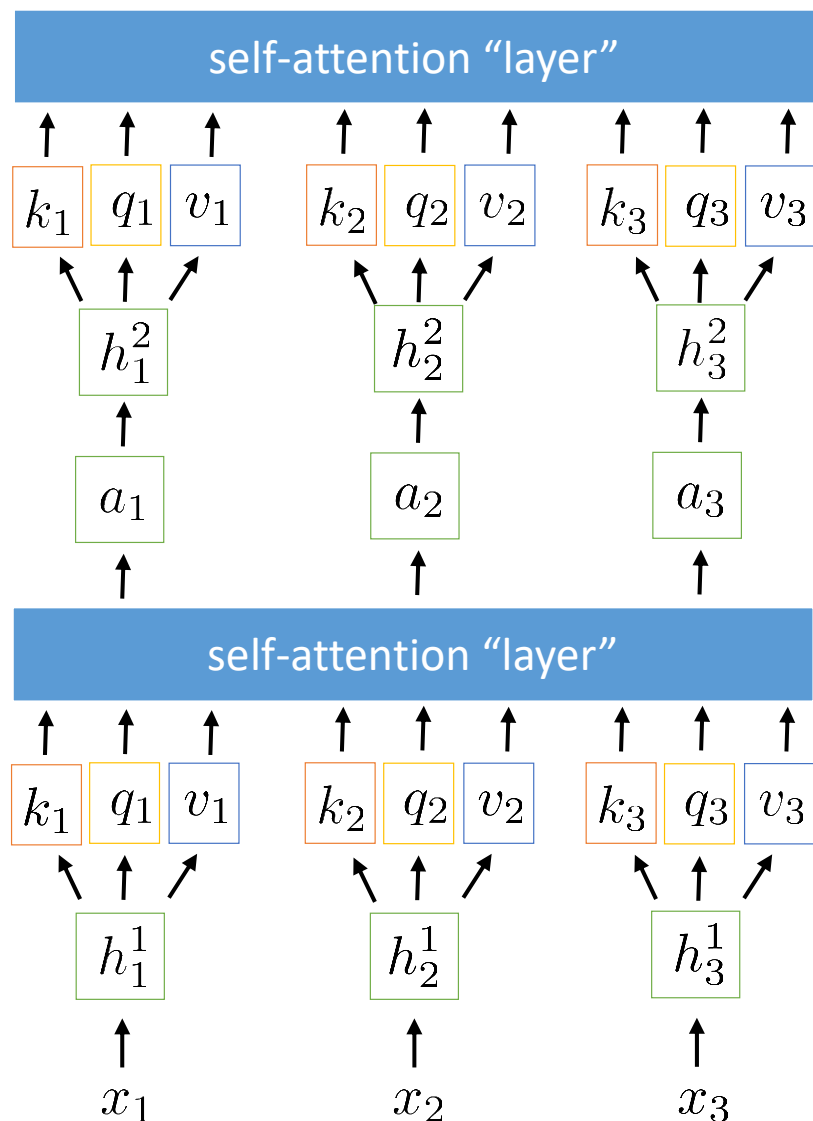
Implementation summary



- We can implement a **practical** sequence model based **entirely** on self-attention
- Alternate self-attention "layers" with nonlinear position-wise feedforward networks (to get nonlinear transformations)
- Use positional encoding (on the input or input embedding) to make the model aware of relative positions of tokens
- Use multi-head attention
- Use masked attention if you want to use the model for decoding

The Transformer

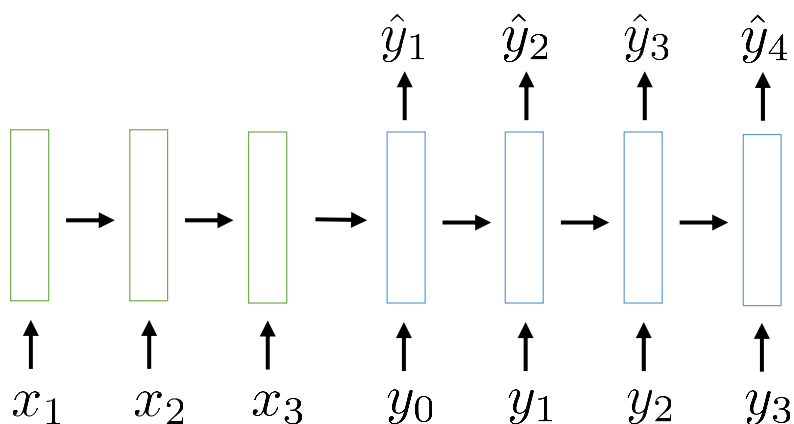
Sequence to sequence with self-attention



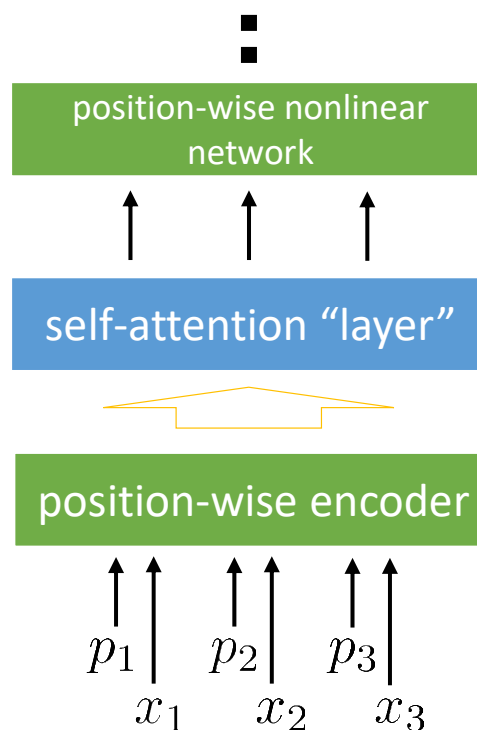
- There are a number of model designs that use successive self-attention and position-wise nonlinear layers to process sequences
- These are generally called “Transformers” because they transform one sequence into another at **each** layer
 - See Vaswani et al. **Attention Is All You Need**. 2017
- The “classic” transformer (Vaswani et al. 2017) is a **sequence to sequence** model
- A number of well-known follow works also use transformers for language modeling (BERT, GPT, etc.)

The “classic” transformer

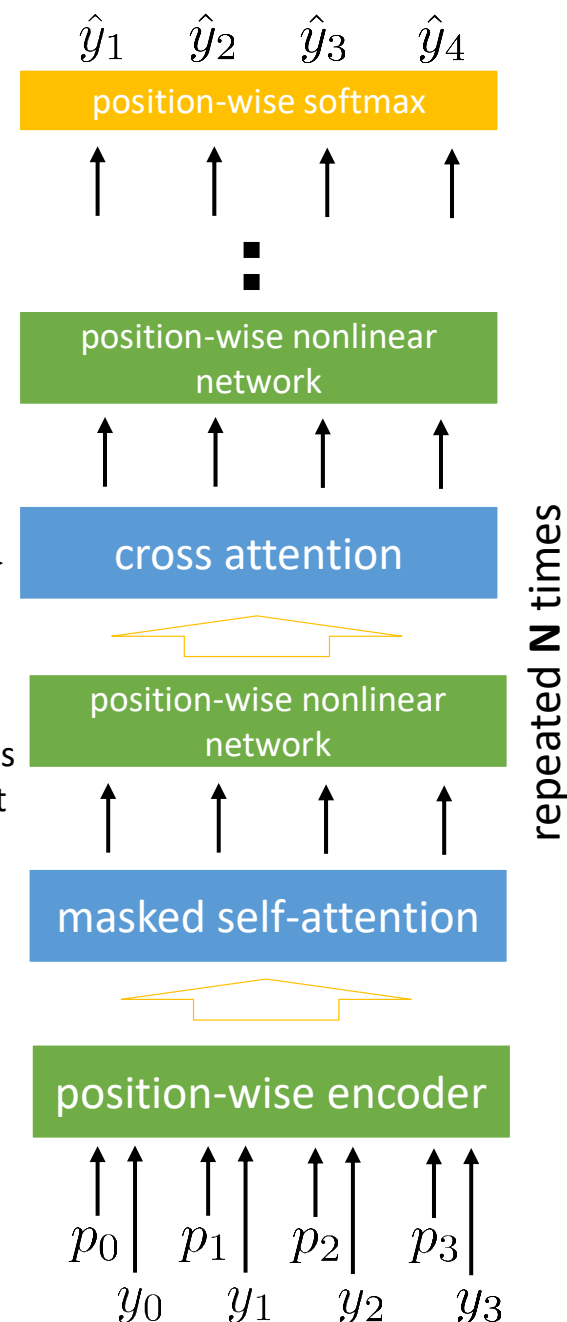
As compared to a sequence
to sequence RNN model



repeated N times



we’ll discuss
how this bit
works soon



Combining encoder and decoder values

“Cross-attention”

Much more like the **standard** attention from the previous lecture

query: $q_l^\ell = W_q^\ell s_l^\ell$ output of position-wise nonlinear network at (decoder) layer ℓ , step l

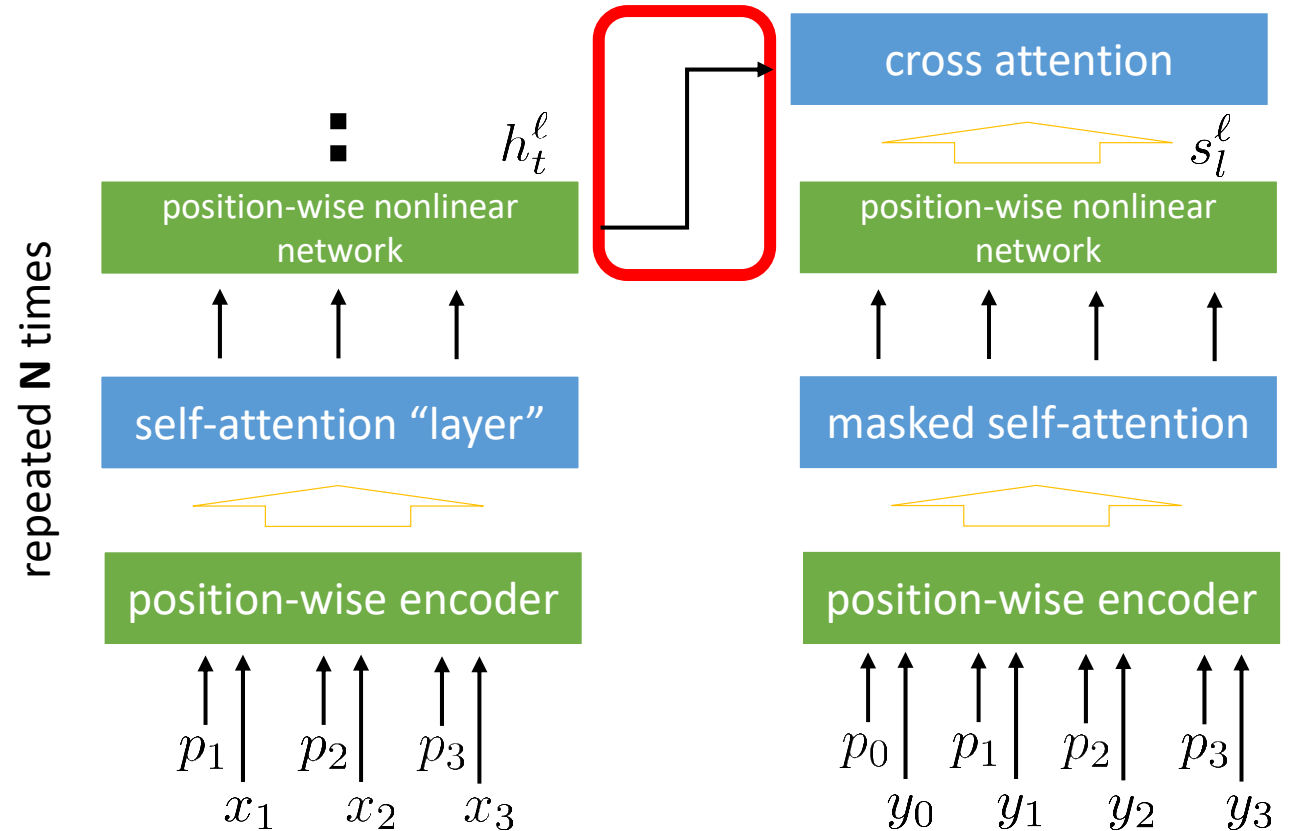
key: $k_t^\ell = W_k^\ell h_t^\ell$ output of position-wise nonlinear network at (encoder) layer ℓ , step t

value: $v_t^\ell = W_v^\ell h_t^\ell$

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

$$\alpha_{l,t}^\ell = \frac{\exp(e_{l,t}^\ell)}{\sum_{t'} \exp(e_{l,t'}^\ell)}$$

$$c_l^\ell = \sum_t \alpha_{l,t}^\ell v_t^\ell \quad \text{cross attention output}$$



in reality, cross-attention is **also** multi-headed!

One last detail: layer normalization

Main idea: batch normalization is very helpful, but hard to use with sequence models

Sequences are different lengths, makes normalizing across the batch hard

Sequences can be very long, so we sometimes have small batches

Simple solution: “layer normalization” – like batch norm, but not across the batch

Batch norm		Layer norm	
d -dim	a_1, a_2, \dots, a_B		a
	$\mu = \frac{1}{B} \sum_{i=1}^B a_i$		$\mu = \frac{1}{d} \sum_{j=1}^d a_j$
	$\sigma = \sqrt{\frac{1}{B} \sum_{i=1}^B (a_i - \mu)^2}$		$\sigma = \sqrt{\frac{1}{d} \sum_{j=1}^d (a_j - \mu)^2}$
	$\bar{a}_i = \frac{a_i - \mu}{\sigma} \gamma + \beta$		$\bar{a} = \frac{a - \mu}{\sigma} \gamma + \beta$

d-dimensional vectors for each sample in batch

different *dimensions* of *a*

1-dim

Putting it all together

Decoder decodes one position at a time with masked attention

The Transformer

6 layers, each with $d = 512$

$\bar{h}_t^\ell = \text{LayerNorm}(\bar{a}_t^\ell + h_t^\ell)$
passed to next layer $\ell + 1$

multi-head attention keys and values
 $k_{t,1}^\ell, \dots, k_{t,m}^\ell$ and $v_{t,1}^\ell, \dots, v_{t,m}^\ell$

$$h_t^\ell = W_2^\ell \text{ReLU}(W_1^\ell \bar{a}_t^\ell + b_1^\ell) + b_2^\ell$$

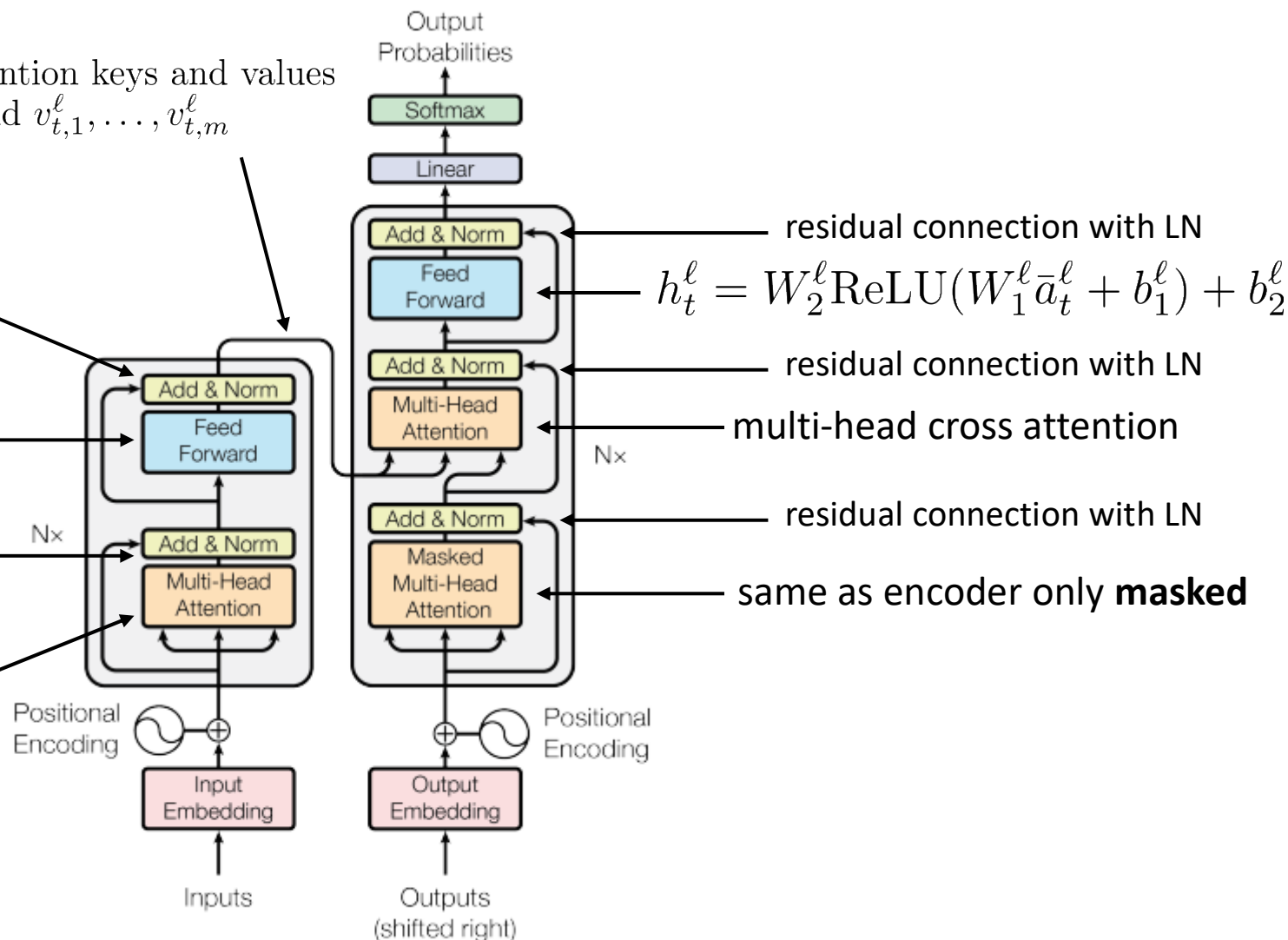
2-layer neural net at each position

$$\bar{a}_t^\ell = \text{LayerNorm}(\bar{h}_t^{\ell-1} + a_t^\ell)$$

essentially a residual connection with LN

input: $\bar{h}_t^{\ell-1}$
output: a_t^ℓ

concatenates attention from all heads



Why transformers?

Downsides:

- Attention computations are technically $O(n^2)$
- Somewhat more complex to implement (positional encodings, etc.)

Benefits:

- + Much better long-range connections
- + Much easier to parallelize
- + In practice, can make it much deeper (more layers) than RNN

The benefits seem to **vastly** outweigh the downsides, and transformers work **much** better than RNNs (and LSTMs) in many cases

Arguably one of the most important sequence modeling improvements of the past decade

Why transformers?

In practice, this means we can use
larger models for the same cost

larger model = better performance

much faster training

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	41.29	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	$3.3 \cdot 10^{18}$	
Transformer (big)	28.4	41.8	$2.3 \cdot 10^{19}$	

great translation results

Text summarization

previous state of the art seq2seq model

Model	Test perplexity	ROUGE-L
<i>seq2seq-attention, L = 500</i>	5.04952	12.7
<i>Transformer-ED, L = 500</i>	2.46645	34.2
<i>Transformer-D, L = 4000</i>	2.22216	33.6
<i>Transformer-DMCA, no MoE-layer, L = 11000</i>	2.05159	36.2
<i>Transformer-DMCA, MoE-128, L = 11000</i>	1.92871	37.9
<i>Transformer-DMCA, MoE-256, L = 7500</i>	1.90325	38.8

lower is better (this metric is similar to 1/likelihood)

We'll learn more about the power of transformers
as **language models** next time!

Implementation

- <https://nlp.seas.harvard.edu/annotated-transformer/>

Additional Resources

- <http://jalammar.github.io/illustrated-transformer/>
 - Must read!
- https://d2l.ai/chapter_attention-mechanisms-and-transformers/transformer.html