Transformer: Part I (Encoder)

CS7150, Spring 2025
Prof. Huaizu Jiang
Northeastern University

Announcement

- Fixing an error in Task 2.7 of PA2
  - Check the announcement on Canvas
- Grades and feedback of the project proposal will be released by the end of Friday

Recap

Recall: Batch Normalization for ConvNets

Batch Normalization for
fully-connected networks

$$x : N \times D$$

Normalize

$$\mu, \sigma : 1 \times D$$
$$\gamma, \beta : 1 \times D$$
$$y = \frac{(x - \mu)}{\sigma}\gamma + \beta$$

Batch Normalization for
convolutional networks
(Spatial Batchnorm, BatchNorm2D)

$$x : N \times C \times H \times W$$

Normalize

$$\mu, \sigma : 1 \times C \times 1 \times 1$$
$$\gamma, \beta : 1 \times C \times 1 \times 1$$
$$y = \frac{(x - \mu)}{\sigma}\gamma + \beta$$

How to use BatchNorm in an RNN?

Naïve way of applying BN in RNN

Why it is not a good idea?

1. Variable lengths in the input.
2. The recurrent nature of RNN.

Batch Normalization for **recurrent** networks

$$x : N \times L \times C$$

$$\mu, \sigma \ : 1 \times 1 \times C$$

$$\gamma, \beta \ : 1 \times 1 \times C$$

$$y = \frac{(x - \mu)}{\sigma}\gamma + \beta$$

Layer Normalization

Computing of μ, σ is independent of the batch and length dimension.

But the learnable parameters γ, β are still for each channel.

Largely adopted in Transformer.

Layer Normalization for **recurrent** networks

$$x : N \times L \times C$$

$$\downarrow$$

$$\mu, \sigma : N \times L \times 1$$

$$\gamma, \beta : 1 \times 1 \times C$$

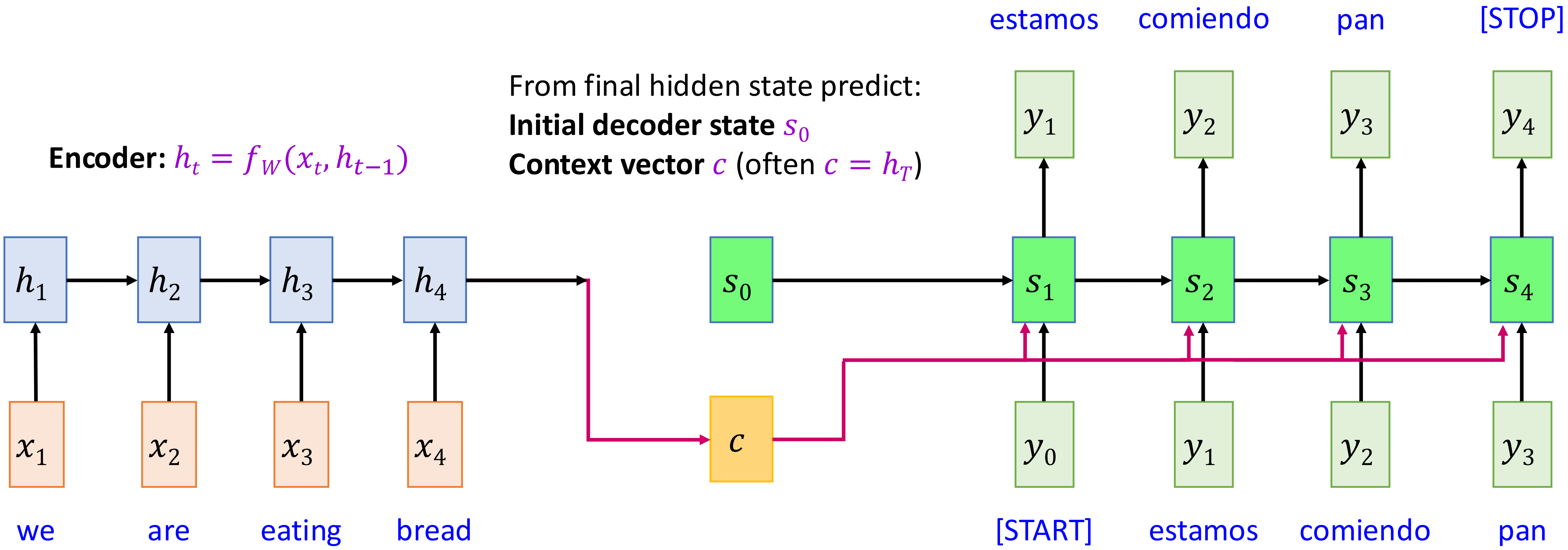
$$y = \frac{(x - \mu)}{\sigma}\gamma + \beta$$

Sequence-to-sequence with RNNs

Decoder: $s_t = g_U(y_{t-1}, s_{t-1}, c)$

Encoder: $h_t = f_W(x_t, h_{t-1})$

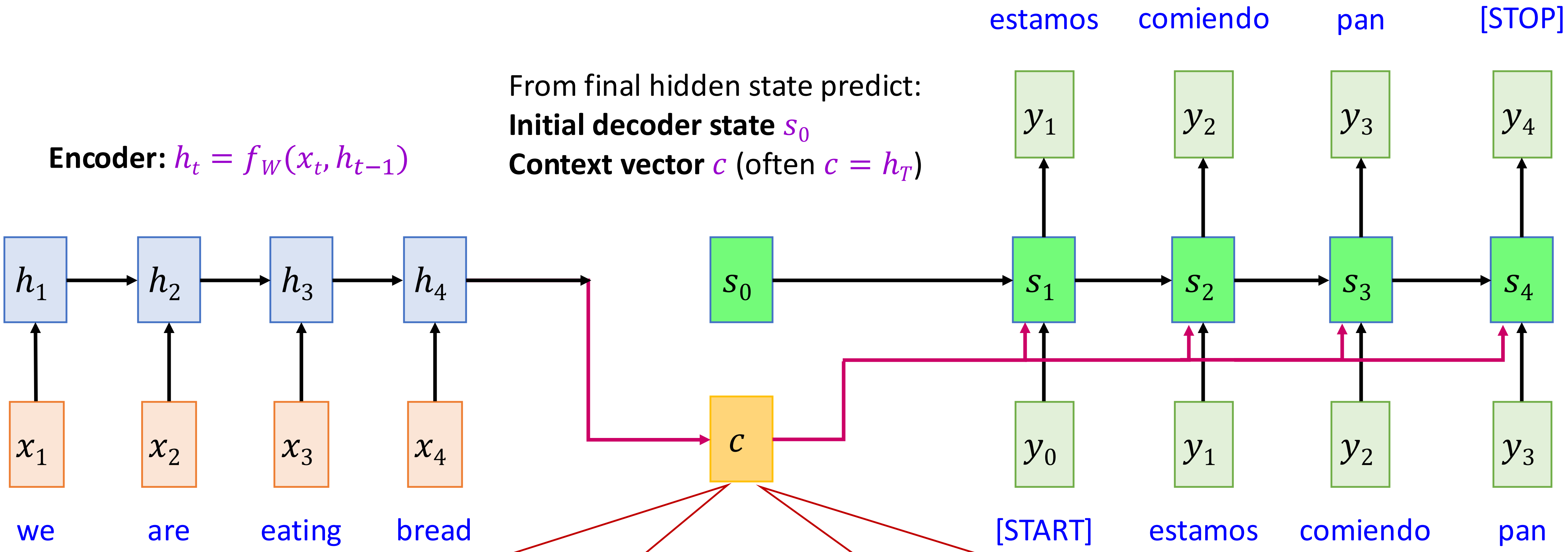
From final hidden state predict:
Initial decoder state s_0
Context vector c (often $c = h_T$)



Slide credit: S. Lazebnik

Sequence-to-sequence with RNNs

Decoder: $s_t = g_U(y_{t-1}, s_{t-1}, c)$



Problem: Input sequence bottlenecked through fixed-sized vector

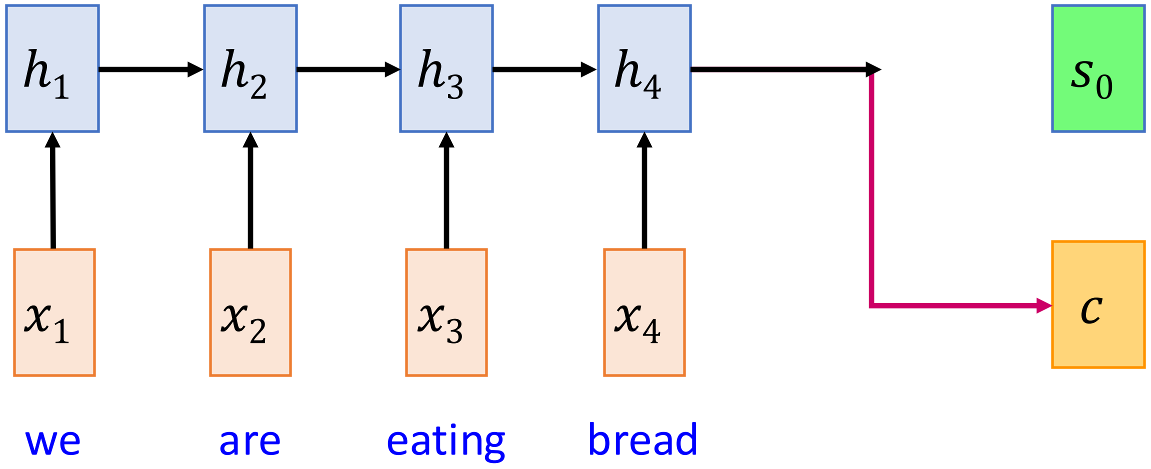
Idea: use *new* context vector at each step of decoder!

Slide credit: S. Lazebnik

A. Sutskever, O. Vinyals, Q. Le, Sequence to sequence learning with neural networks, NeurIPS 2014

Sequence-to-sequence with RNNs and attention

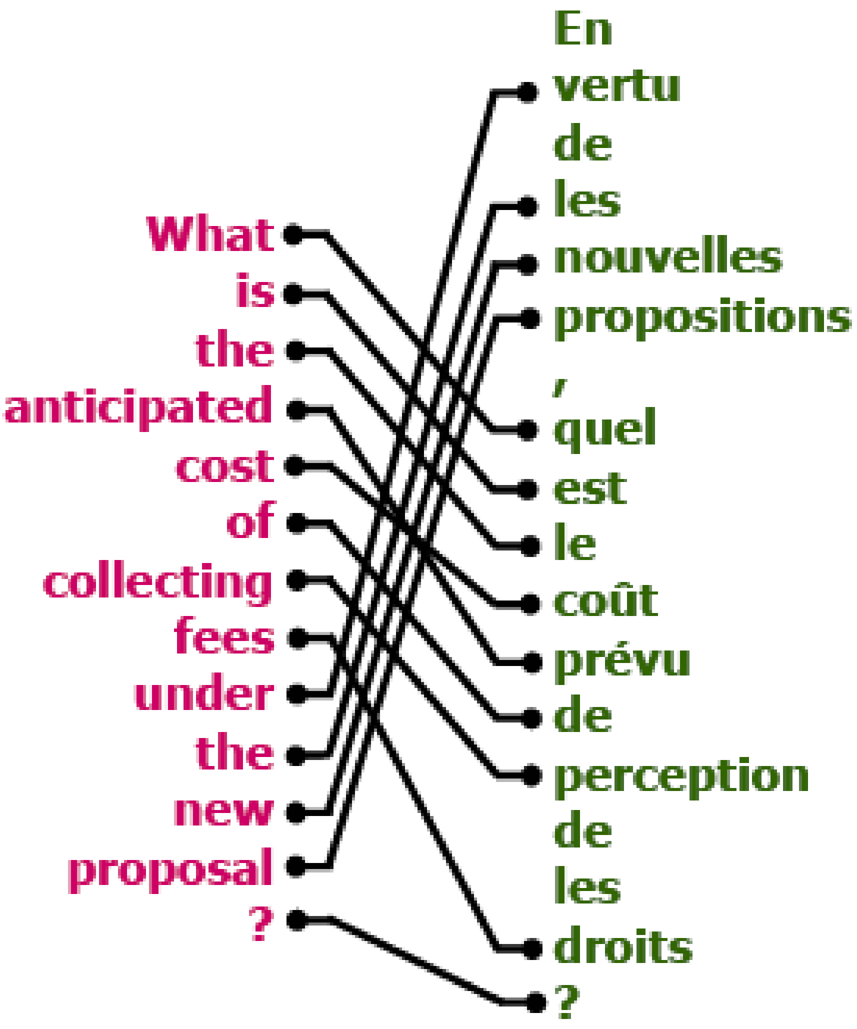
- At each timestep of decoder, context vector "looks at" different parts of the input sequence



Slide credit: S. Lazebnik

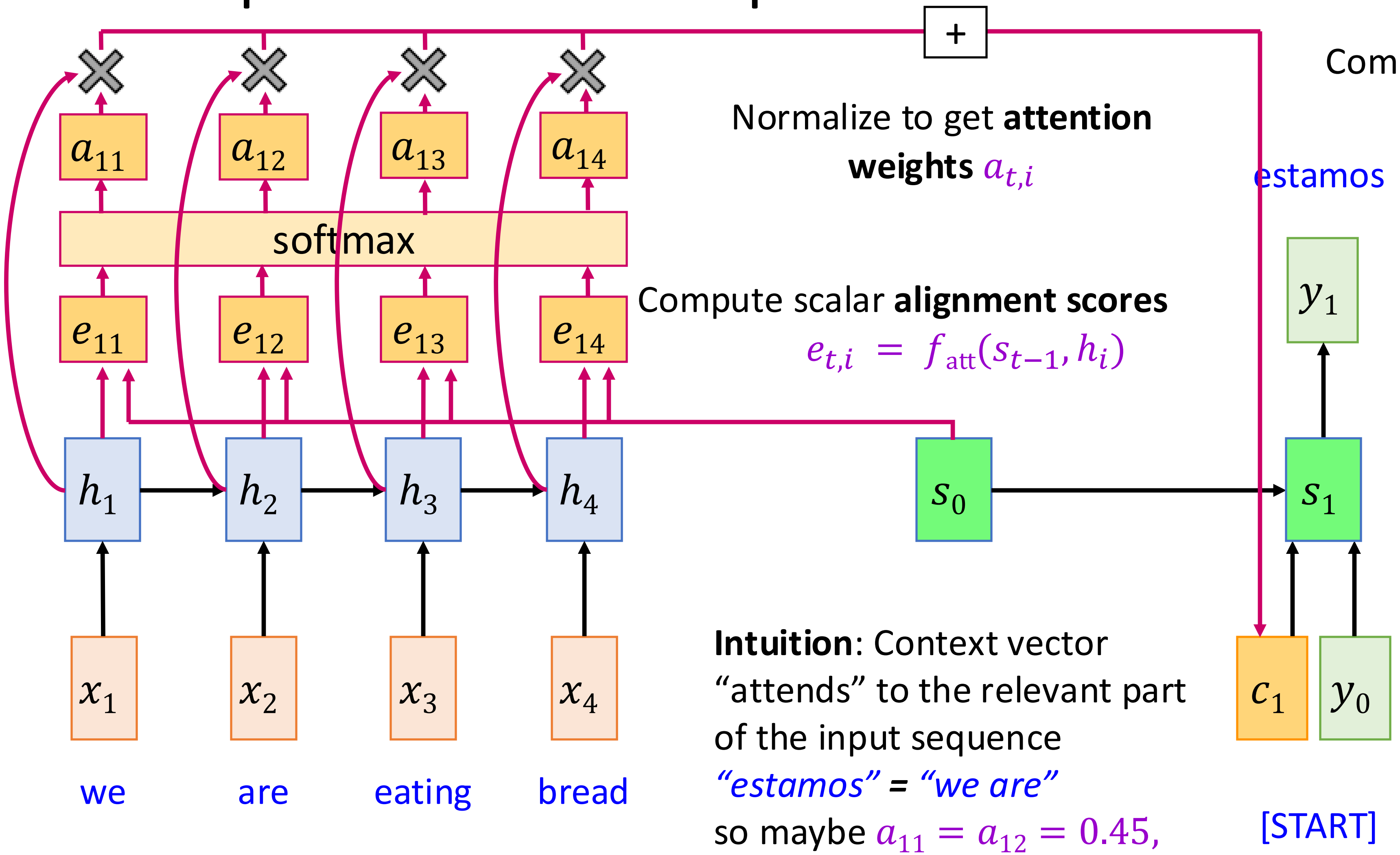
Sequence-to-sequence with RNNs and attention

- Intuition: translation requires *alignment*



Slide credit: S. Lazebnik

Sequence-to-sequence with RNNs and attention



Compute **context vector** as

$$c_t = \sum_i a_{t,i} h_i$$

Normalize to get **attention weights** $a_{t,i}$

Compute scalar **alignment scores**

$$e_{t,i} = f_{\text{att}}(s_{t-1}, h_i)$$

Use context vector in decoder:

$$s_t = g_U(y_{t-1}, s_{t-1}, c_t)$$

Intuition: Context vector "attends" to the relevant part of the input sequence

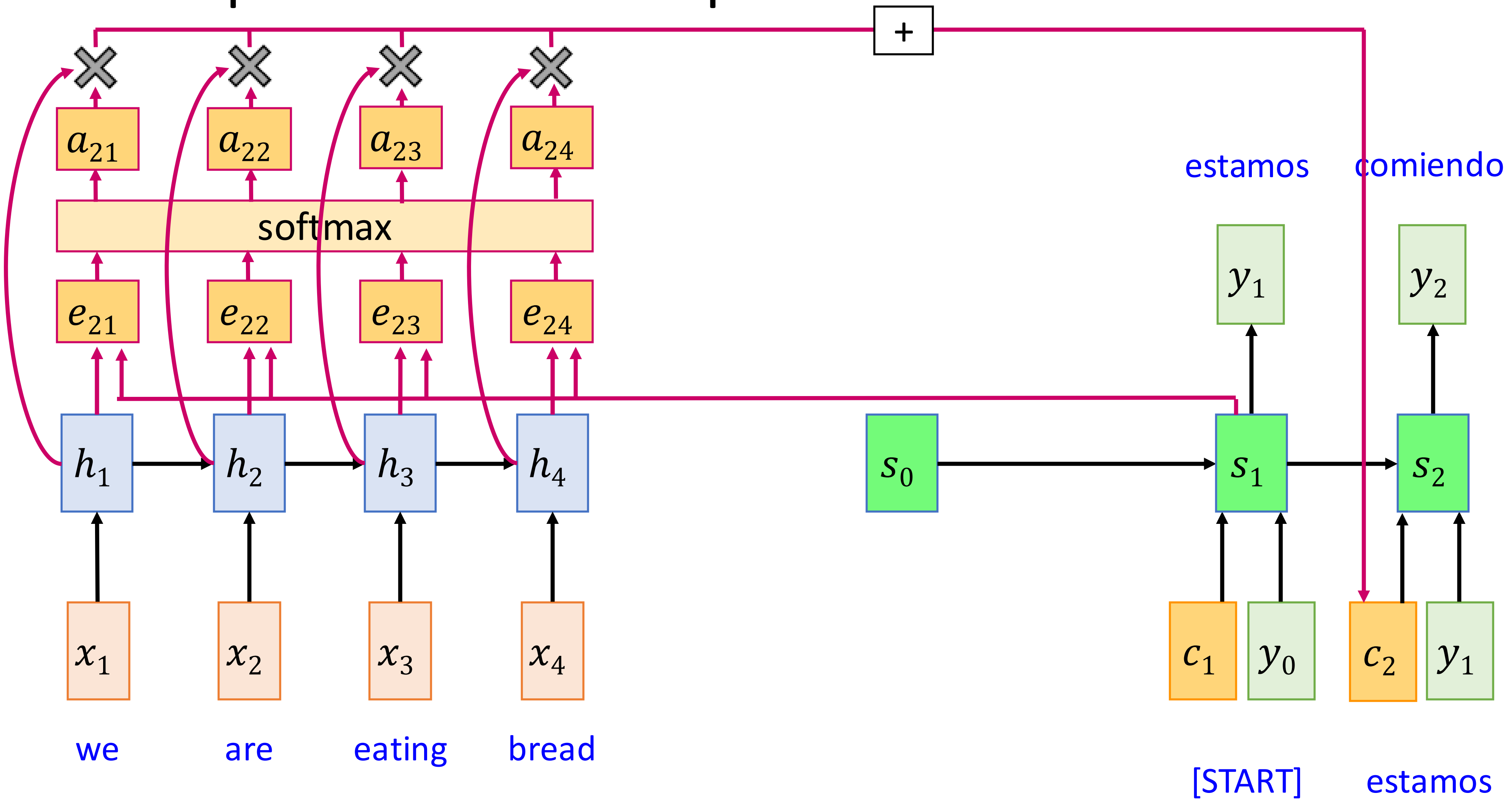
"estamos" = *"we are"*

so maybe $a_{11} = a_{12} = 0.45$,
$a_{13} = a_{14} = 0.05$

Slide credit: S. Lazebnik

D. Bahdanau, K. Cho, Y. Bengio, Neural Machine Translation by Jointly Learning to Align and Translate, ICLR 2015

Sequence-to-sequence with RNNs and attention

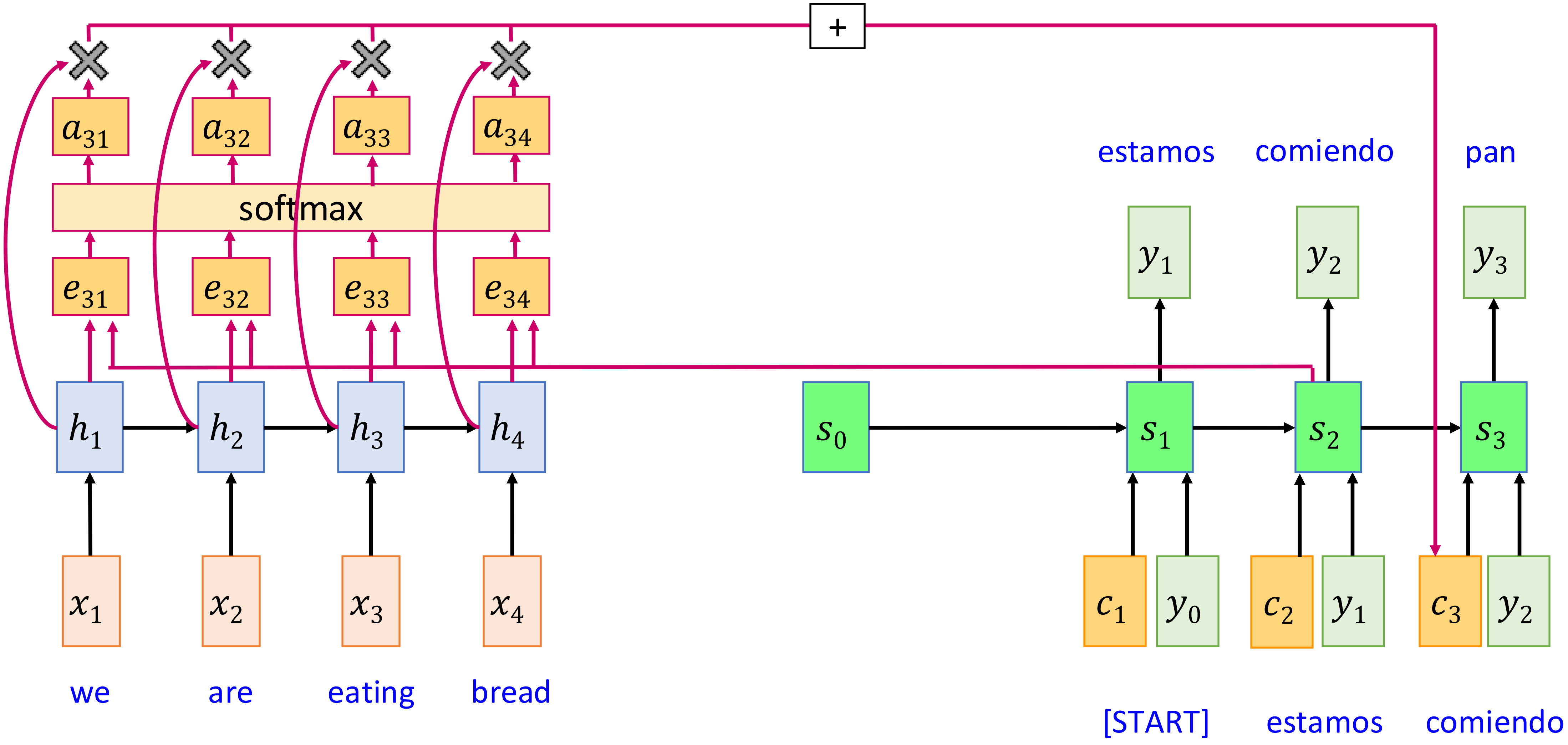


Use c_2 to compute s_2, y_2

Repeat: Use s_1 to compute new context vector c_2

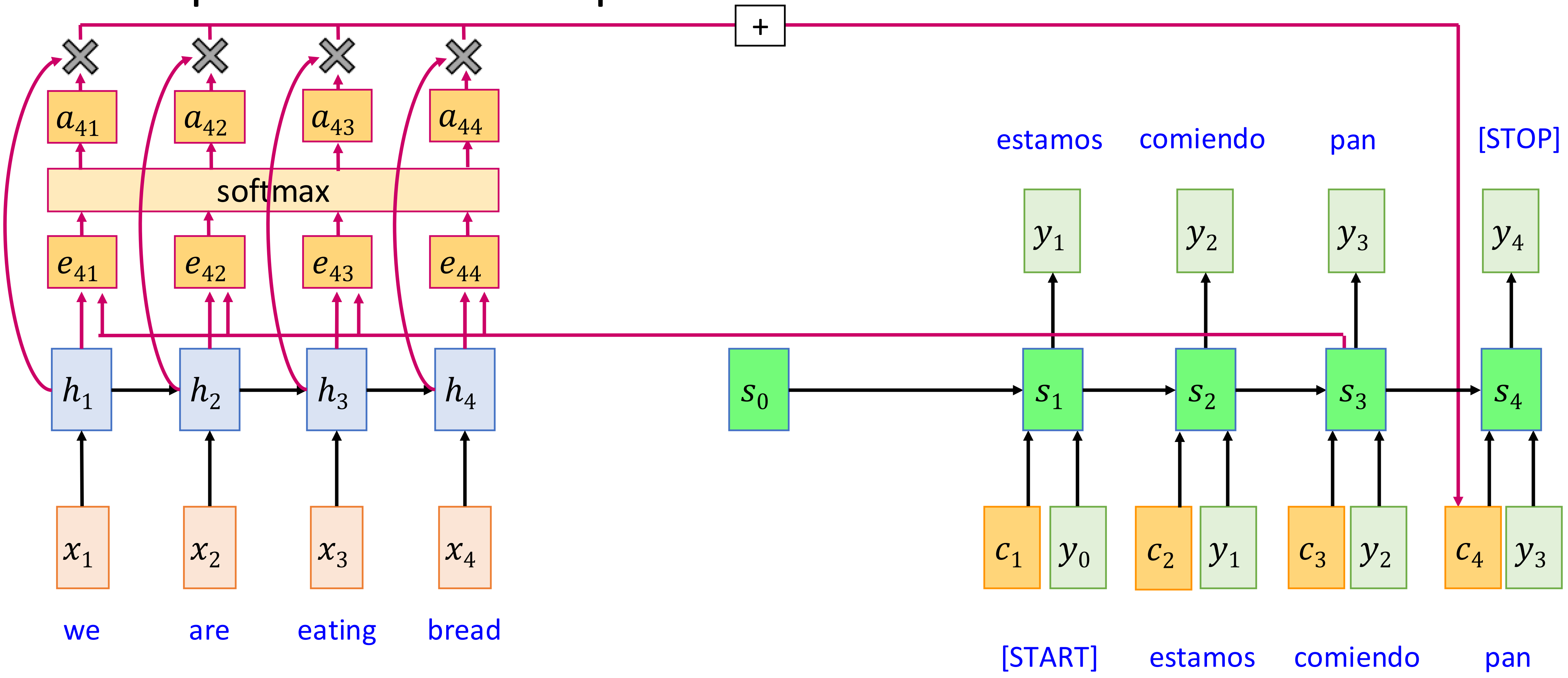
Slide credit: S. Lazebnik

Sequence-to-sequence with RNNs and attention



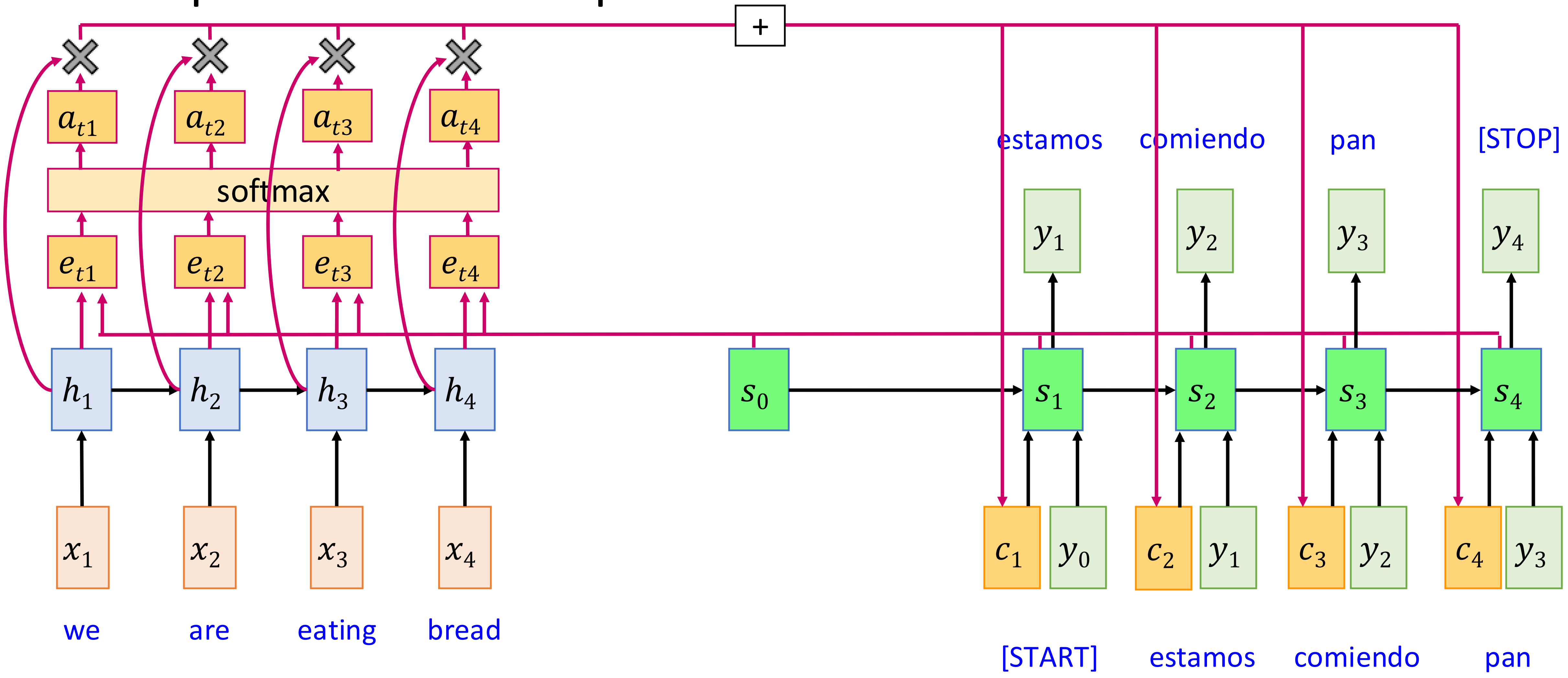
Slide credit: S. Lazebnik

Sequence-to-sequence with RNNs and attention



Slide credit: S. Lazebnik

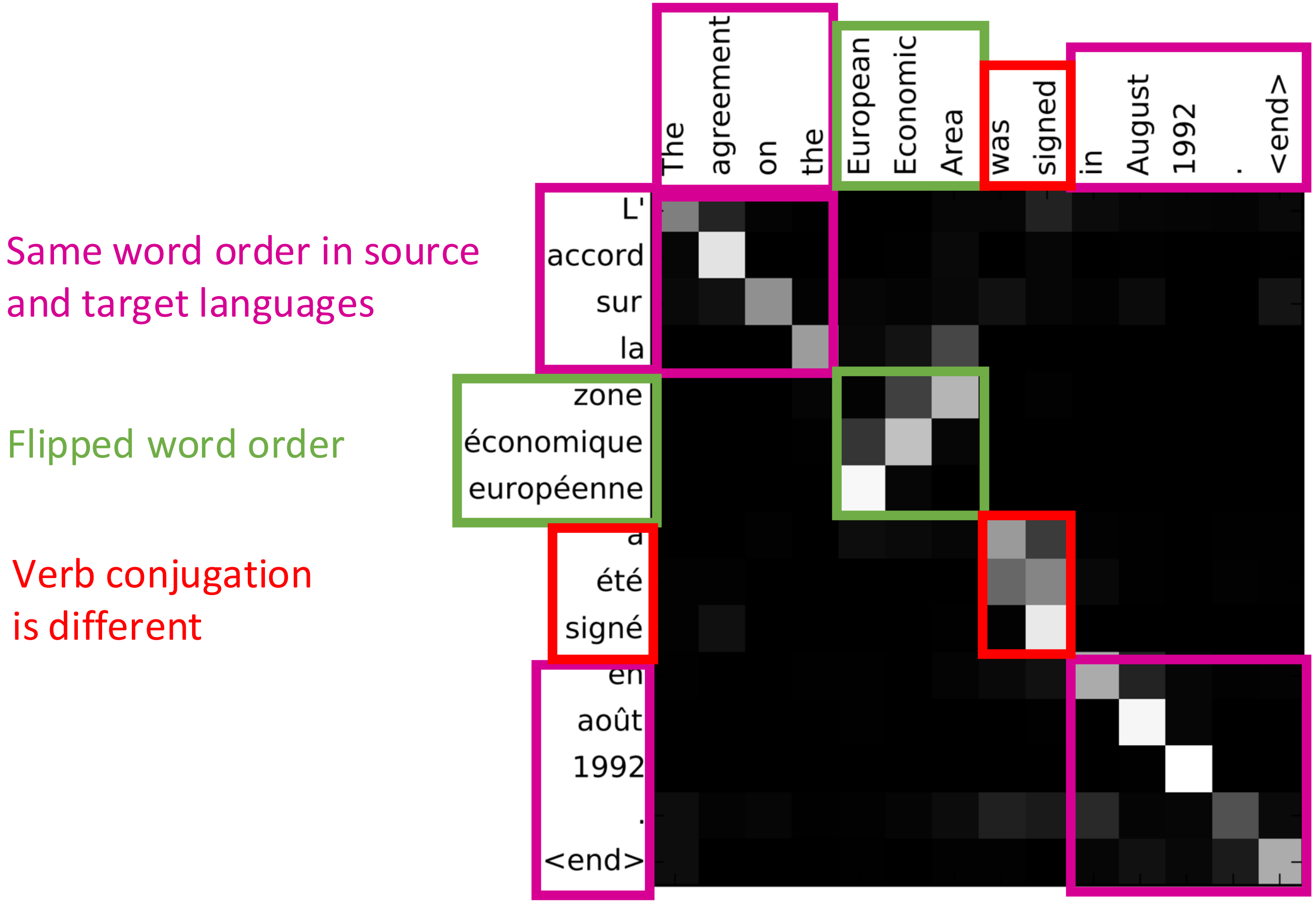
Sequence-to-sequence with RNNs and attention



Slide credit: S. Lazebnik

Sequence-to-sequence with RNNs and attention

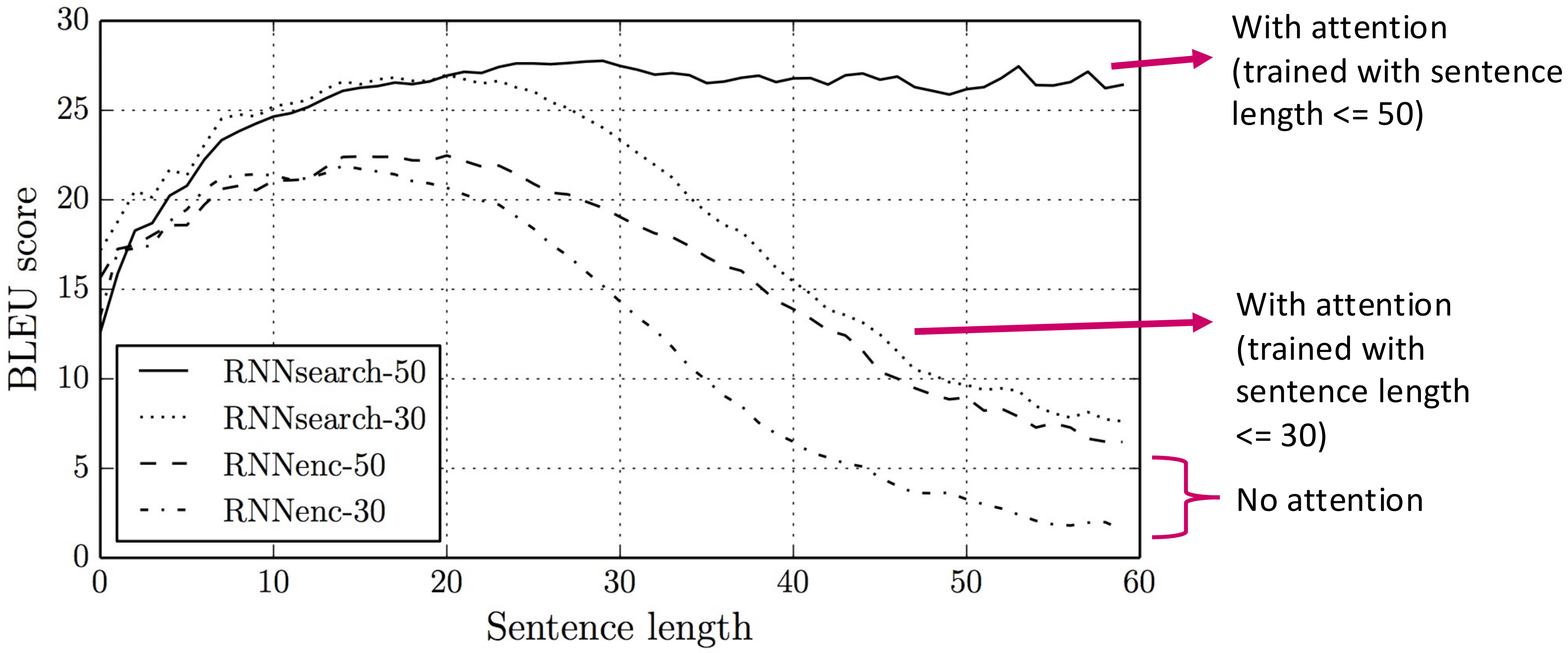
- Visualizing attention weights (English source, French target):



Slide credit: S. Lazebnik

D. Bahdanau, K. Cho, Y. Bengio, [Neural Machine Translation by Jointly Learning to Align and Translate](), ICLR 2015

Quantitative evaluation



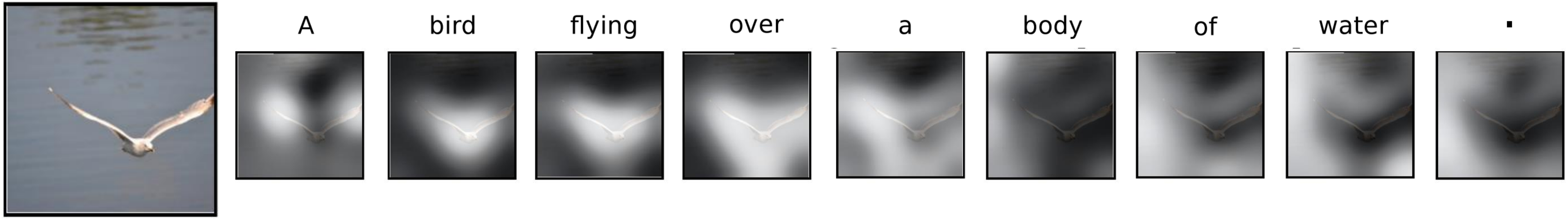
D. Bahdanau, K. Cho, Y. Bengio, [Neural Machine Translation by Jointly Learning to Align and Translate](), ICLR 2015

Slide credit: S. Lazebnik

Attention in a Nutshell

Image captioning with RNNs and attention

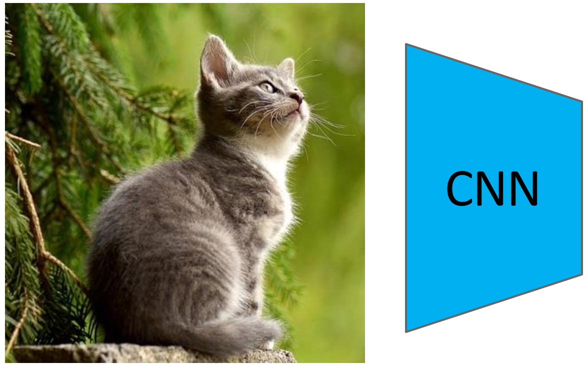
- Idea: pay attention to different parts of the image when generating different words
- Automatically learn this *grounding* of words to image regions without direct supervision

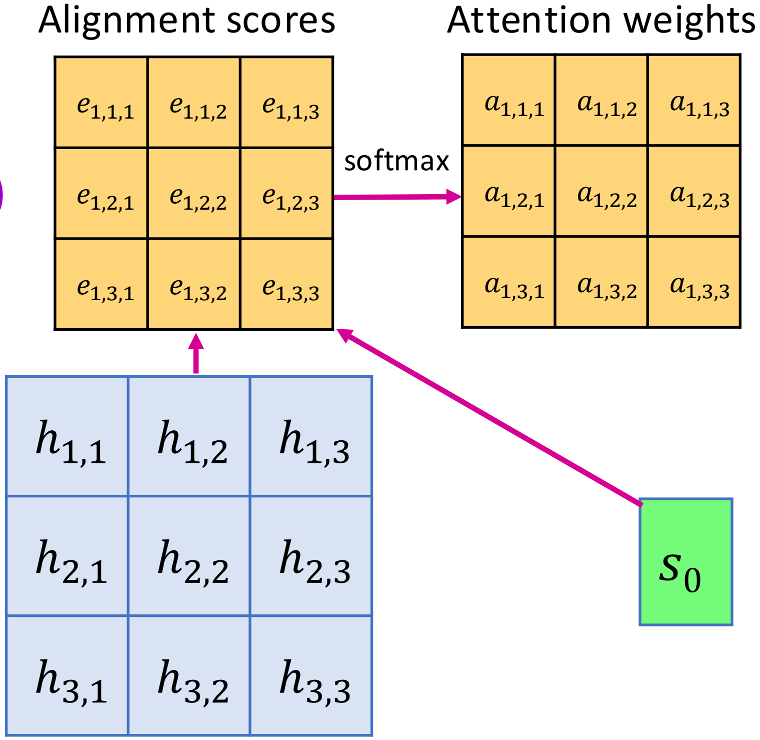


K. Xu et al., [Show, Attend and Tell: Neural Image Caption Generation with Visual Attention](), ICML 2015

Image captioning with RNNs and attention

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



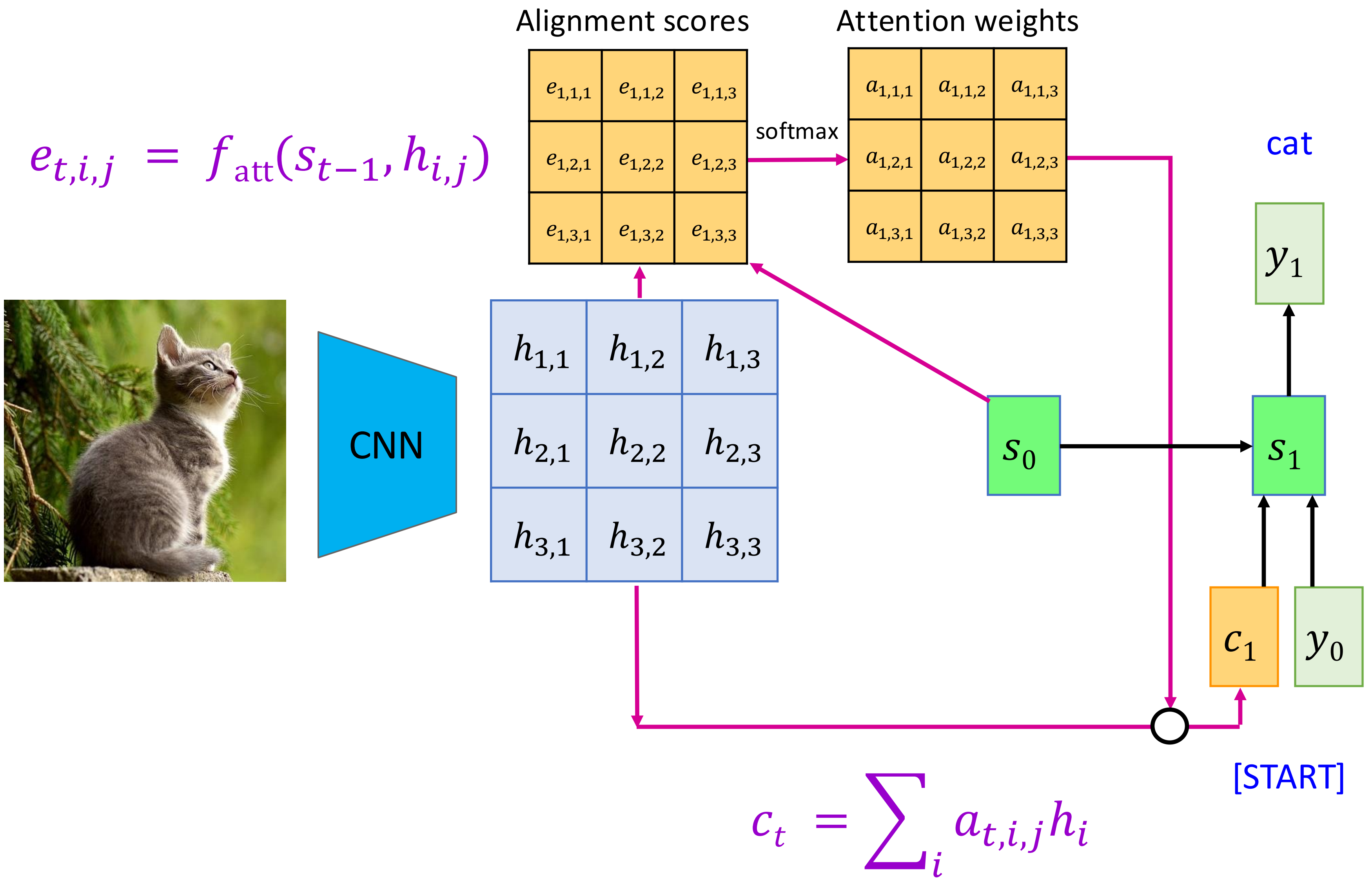


Use CNN to extract a grid of features

K. Xu et al., [Show, Attend and Tell: Neural Image Caption Generation with Visual Attention](), ICML 2015

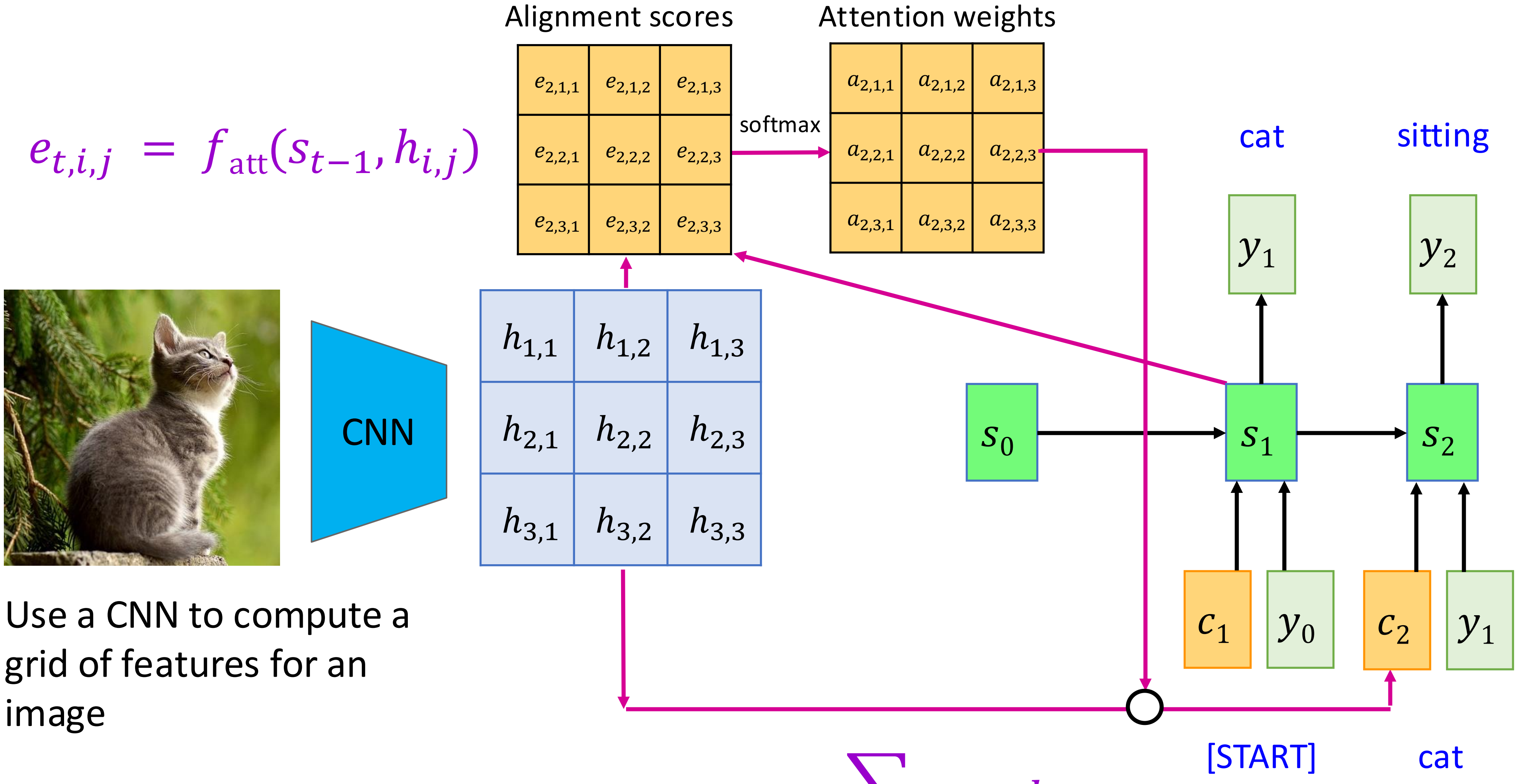
Slide credit: S. Lazebnik

Image captioning with RNNs and attention



Slide credit: S. Lazebnik

Image captioning with RNNs and attention



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

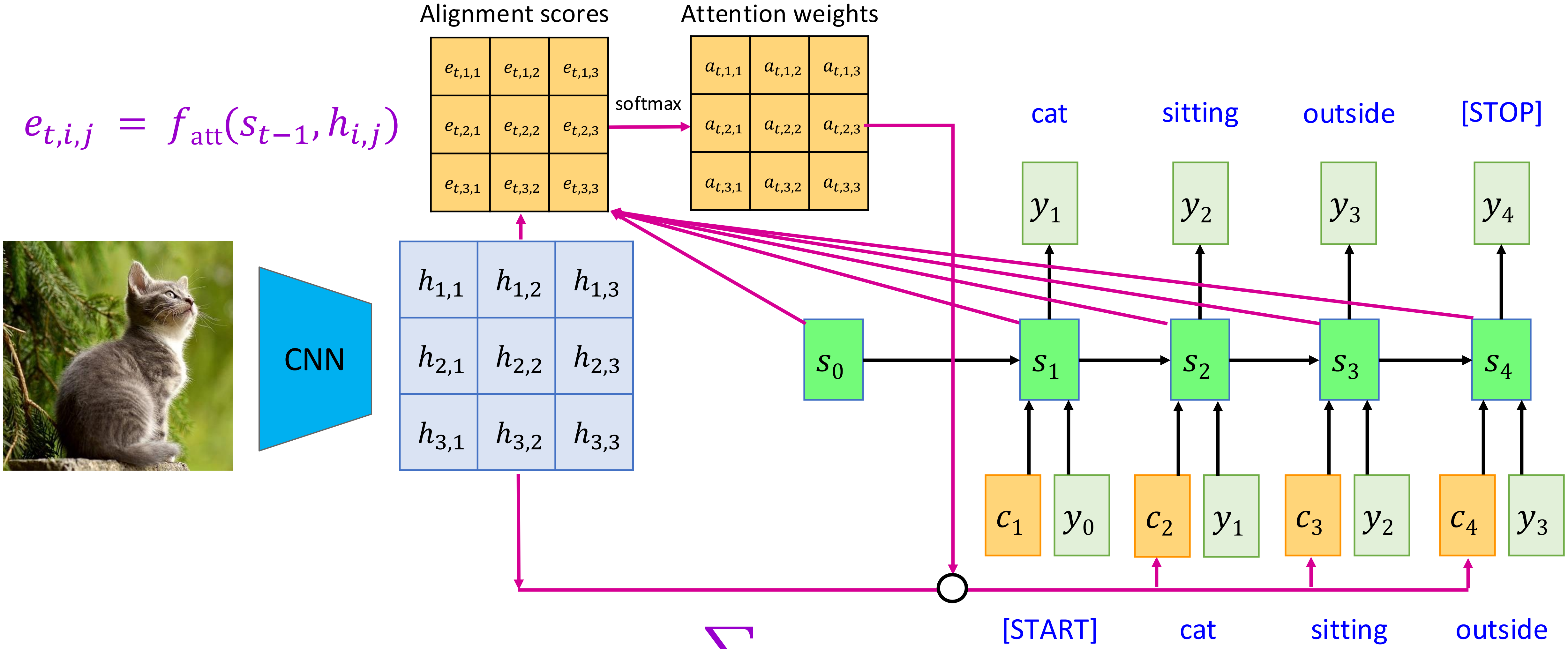
$$c_t = \sum_i a_{t,i,j} h_i$$

Use a CNN to compute a grid of features for an image

Slide credit: S. Lazebnik

Image captioning with RNNs and attention

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



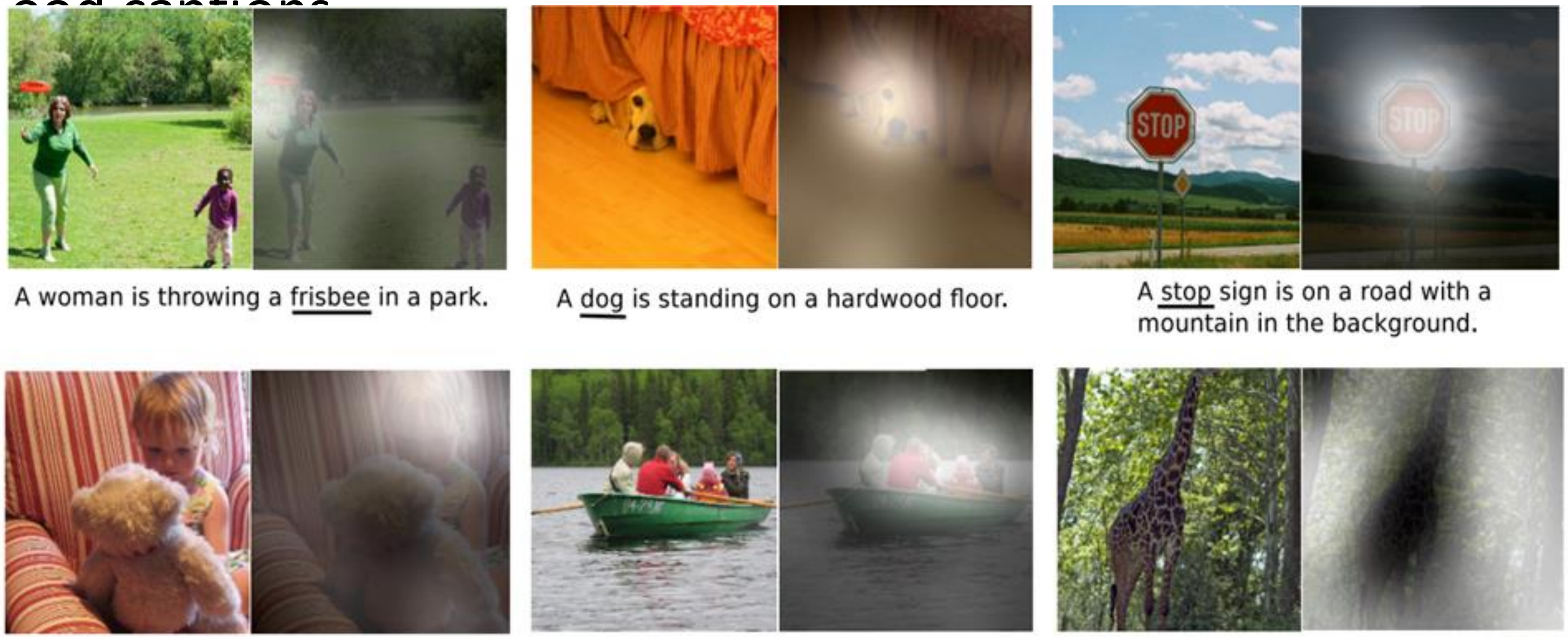
$$c_t = \sum_i a_{t,i,j} h_i$$

Each time step of decoder uses a different context vector that looks at different parts of the input image

Slide credit: S. Lazebnik

Example results

- Good captions



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.

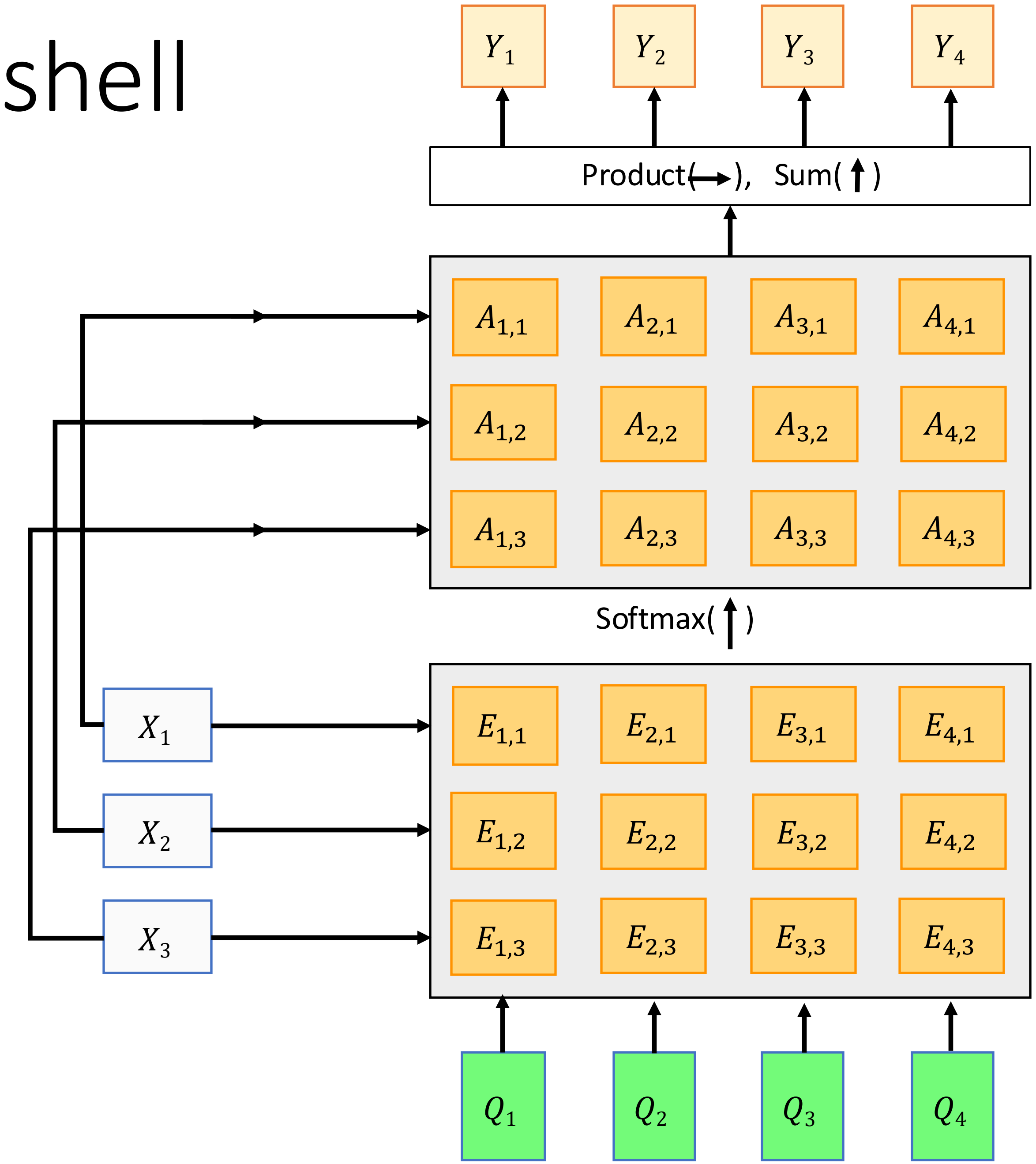
A little girl sitting on a bed with a teddy bear.

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

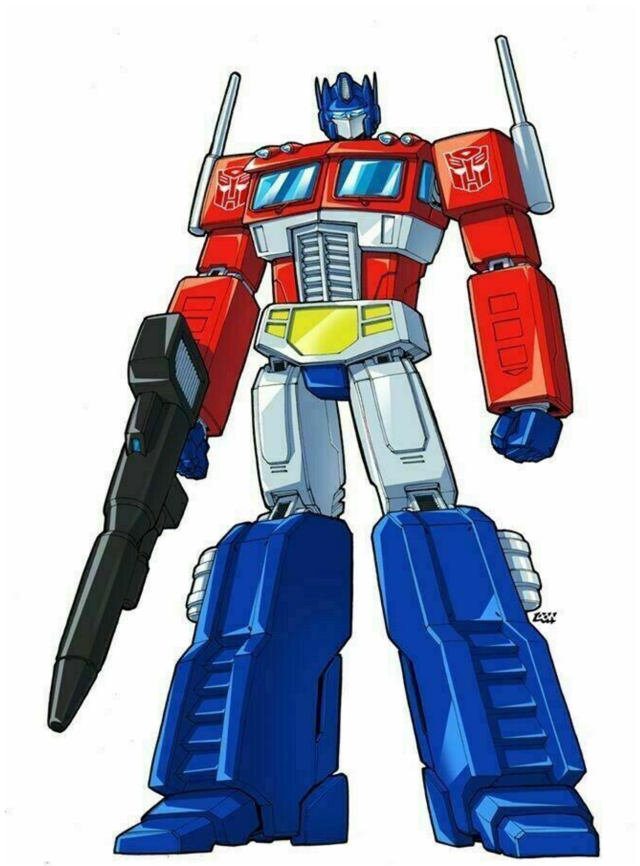
A giraffe standing in a forest with trees in the background.

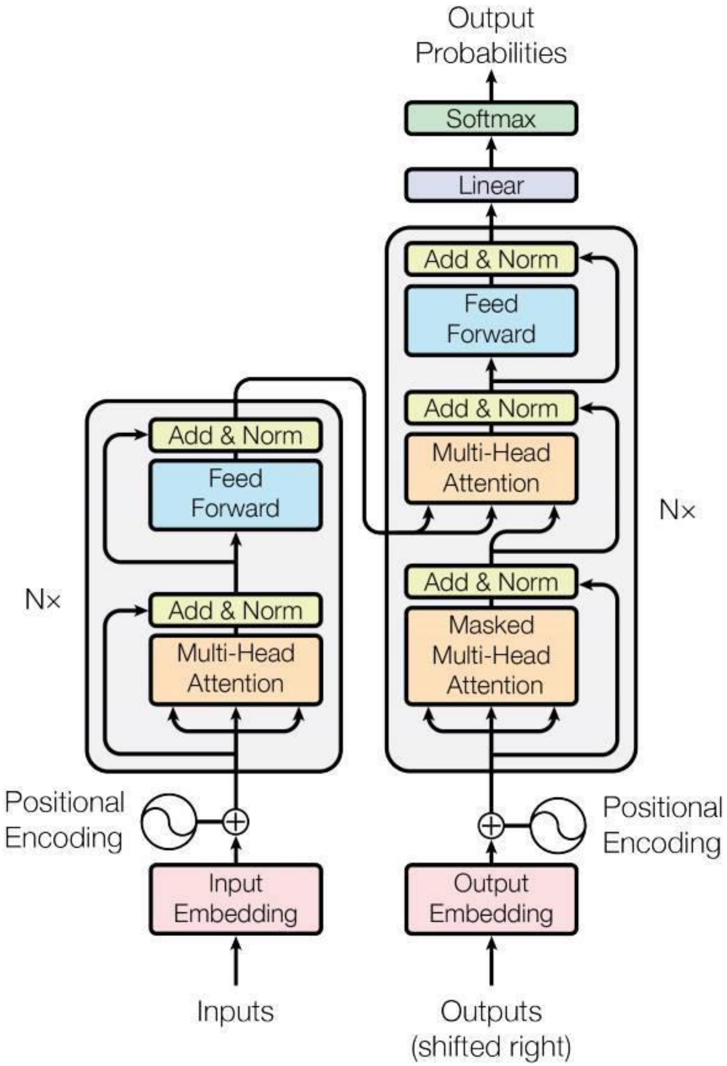
Slide credit: S. Lazebnik

Attention in a nutshell

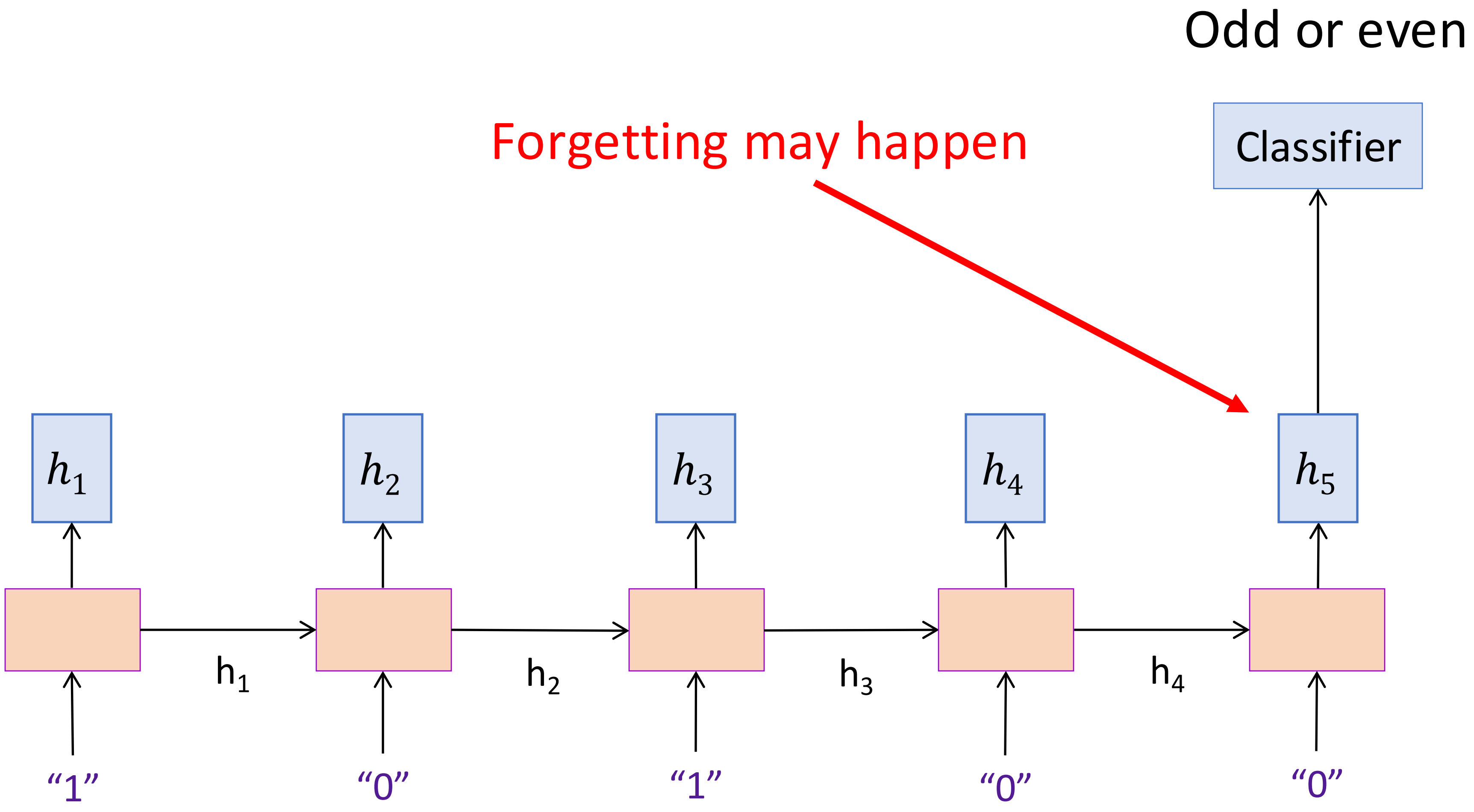


Transformer

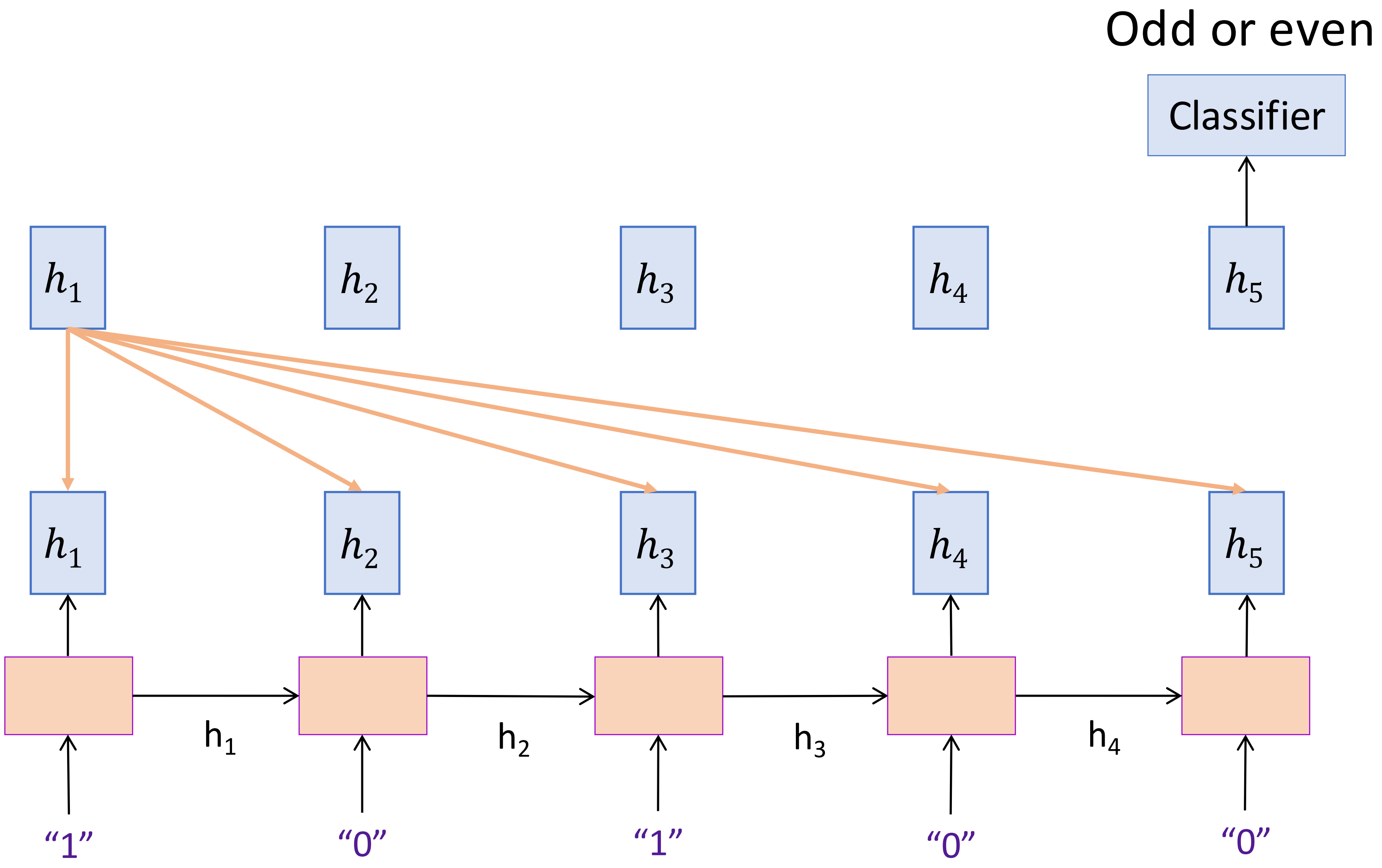




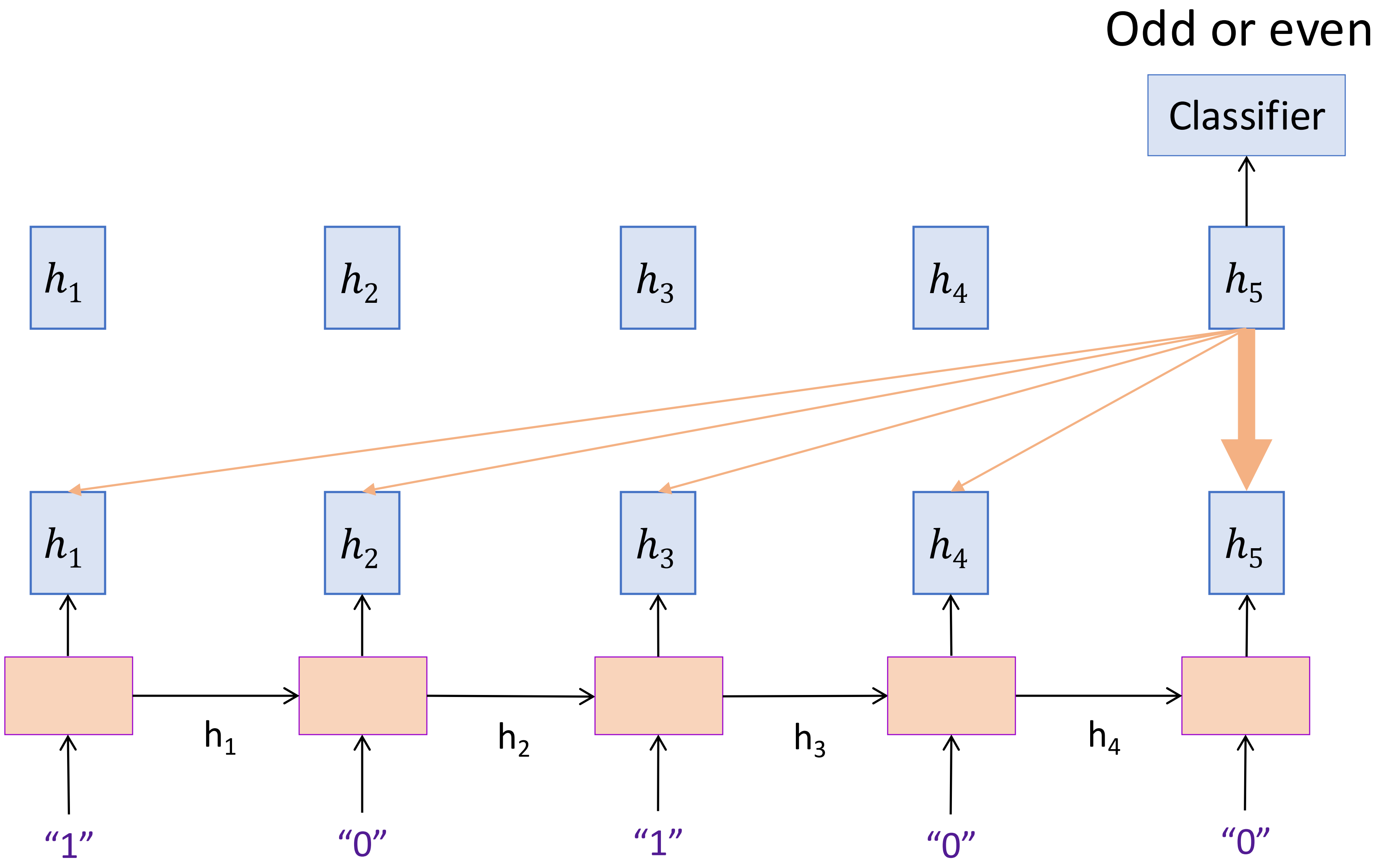
Intuition of Transformer: Self-attention



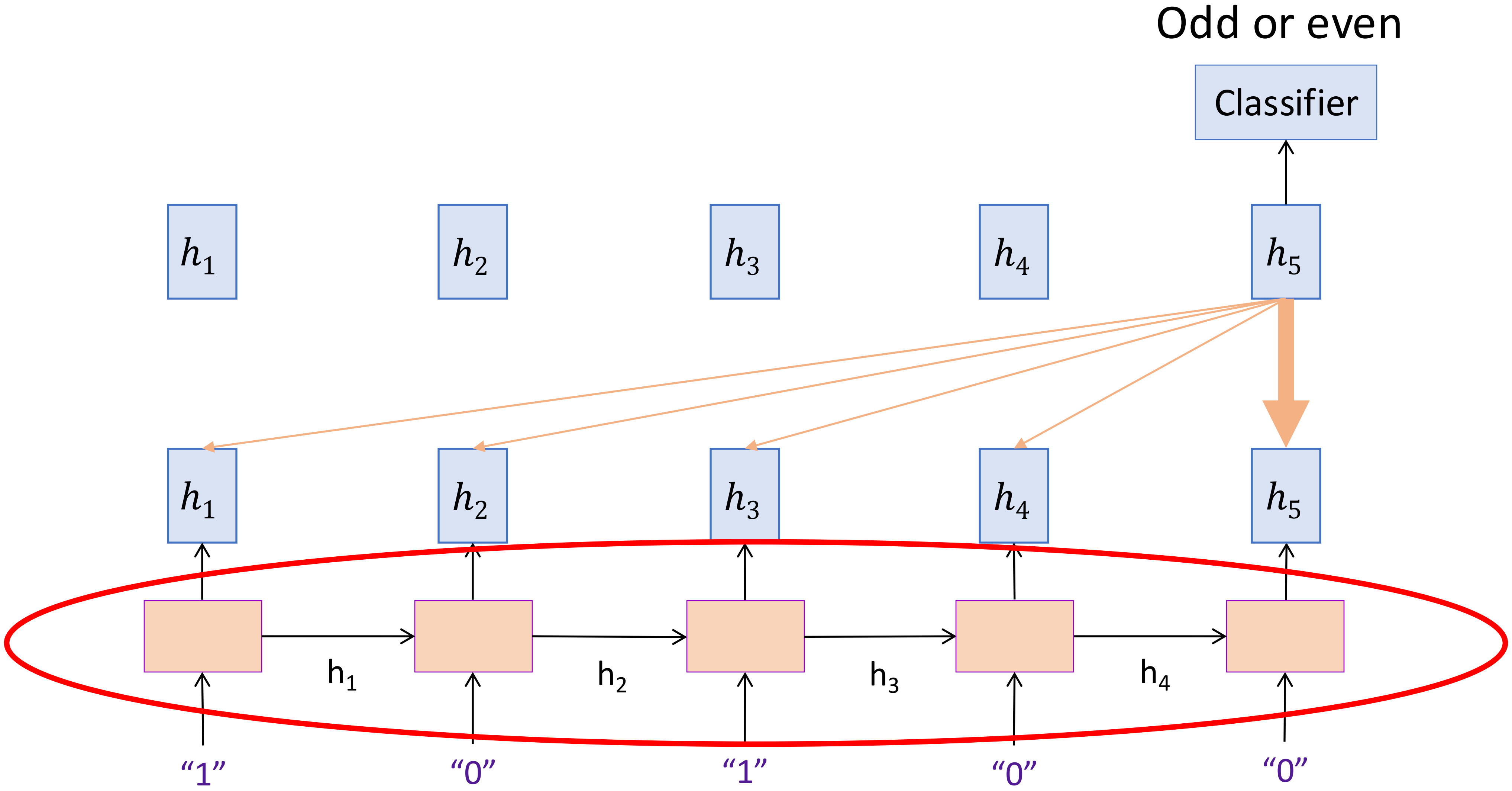
Intuition of Transformer: Self-attention



Intuition of Transformer: Self-attention

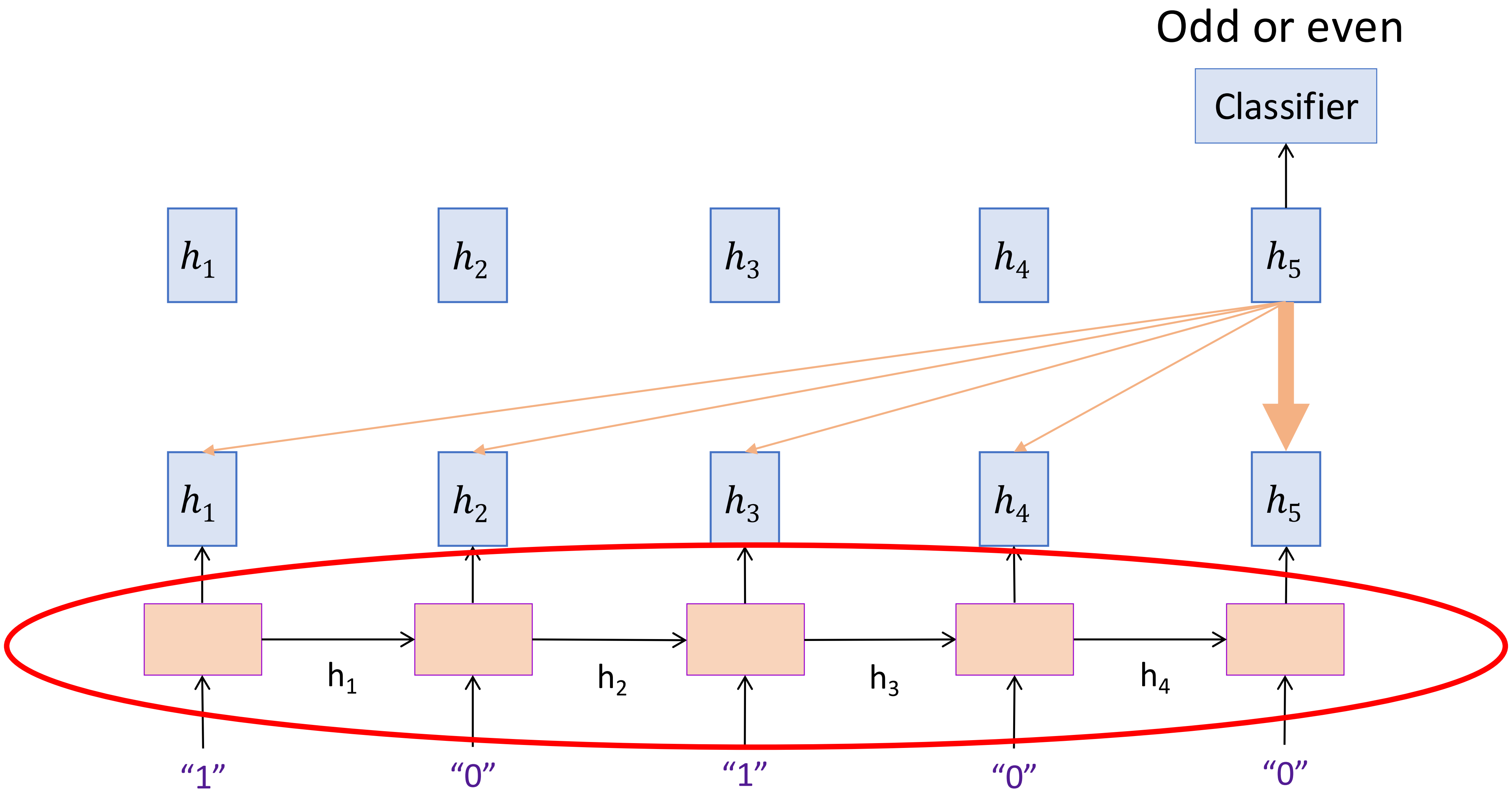


Intuition of Transformer: Self-attention



Do we still need the Recurrent Neural Network to encode the input?

Intuition of Transformer: Self-attention



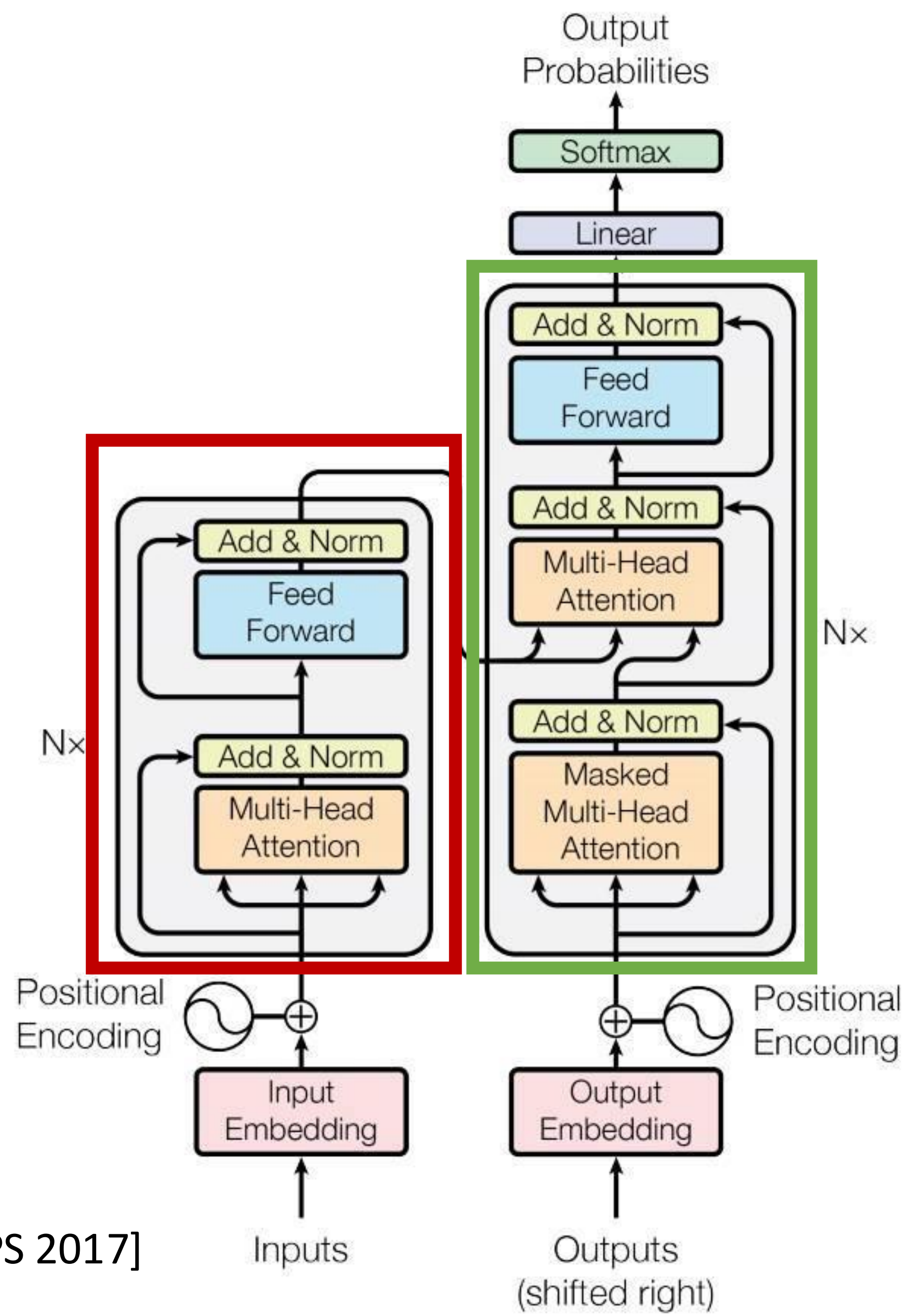
Do we still need the Recurrent Neural Network to encode the input?

Today’s Class

Transformer Encoder

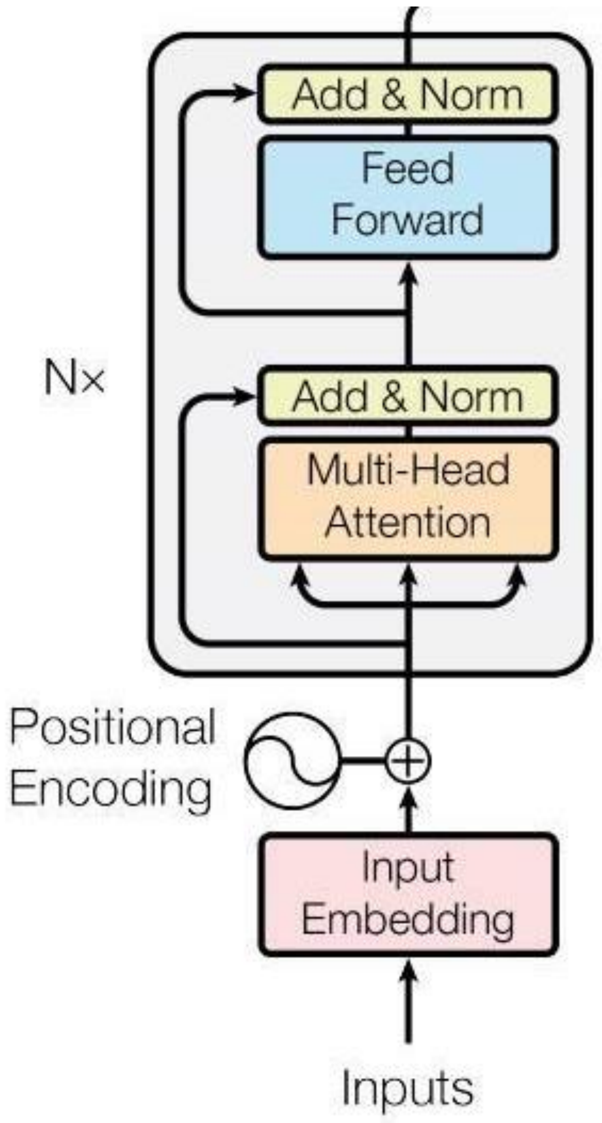
Overview of Transformer

- Encoder (left part)
  - multi-head self attention
  - Position embedding
  - Feedforward network
  - Non-linearity and normalization in-between
- Decoder (right part), optional
  - Multi-head cross attention
  - And others in the encoder



[Vaswani et al., [Attention is all you need](). NeurIPS 2017]

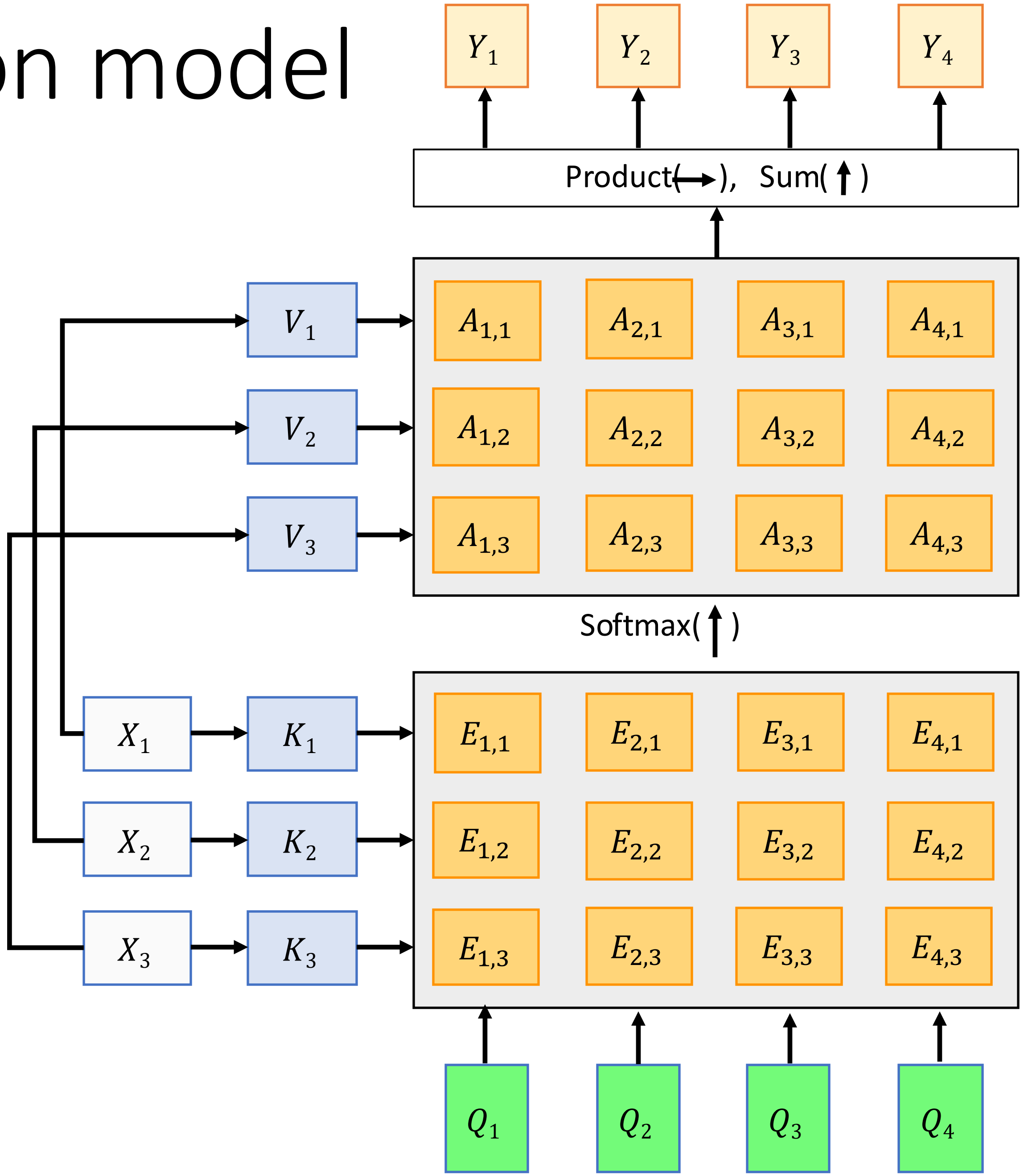
Overview of the Transformer Encoder (Cell)



- Multi-head (self-)attention
- Positional encoding
- Feedforward network
- Residual connections
- Regularization tricks

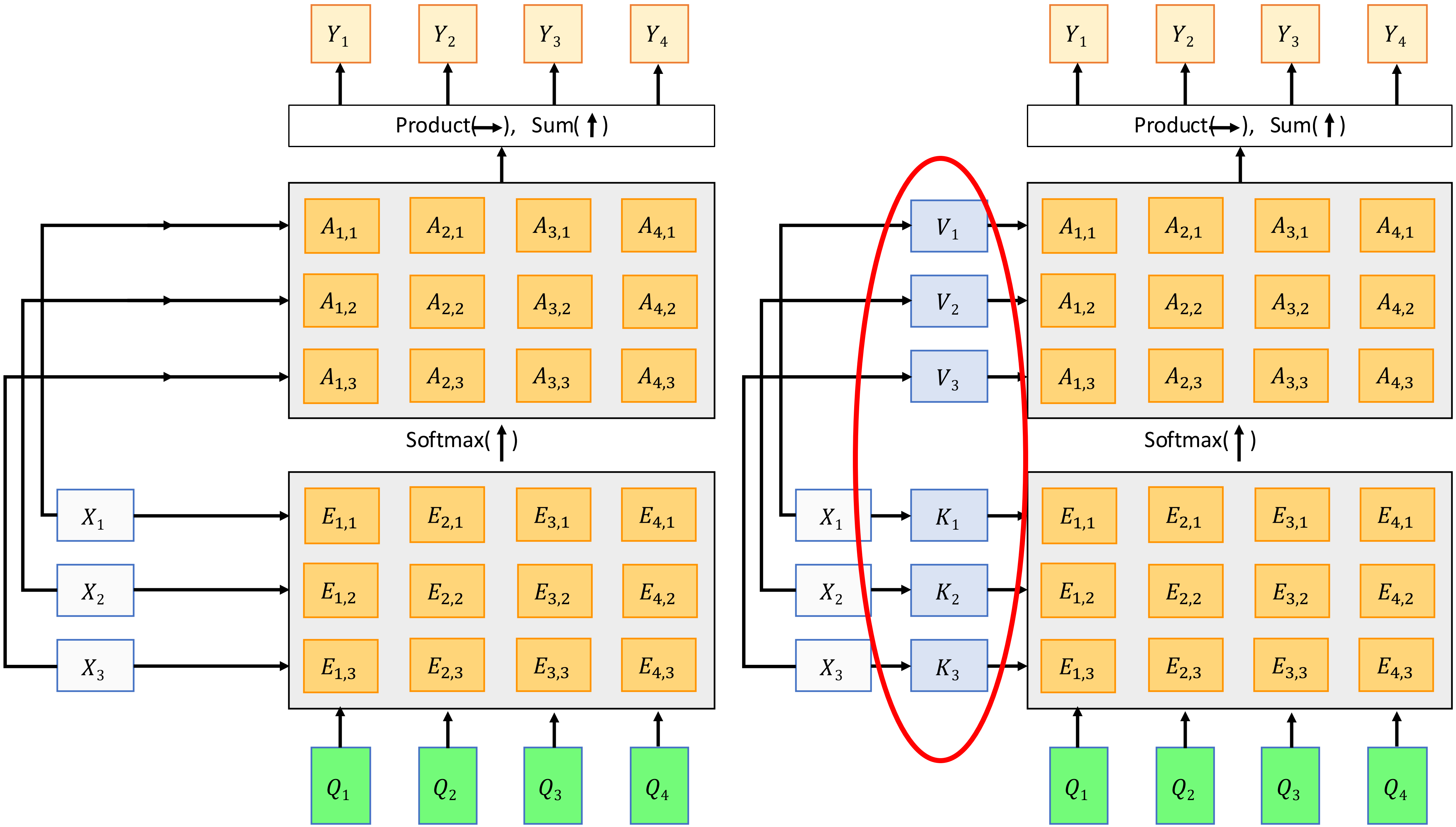
Key-Value-Query attention model

- Key vectors: $K = XW_K$
- Value Vectors: $V = XW_V$
- Query vectors
- Similarities: *scaled dot-product attention*
  - $E_{i,j} = \frac{(Q_i \cdot K_j)}{\sqrt{D}}$ or $E = QK^T / \sqrt{D}$
  - (D is the dimensionality of the keys)
- Attn. weights: $A = \mathrm{softmax}(E, \mathrm{dim} = 1)$
- Output vectors:
  - $Y_i = \sum_j A_{i,j} V_j$ or $Y = AV$



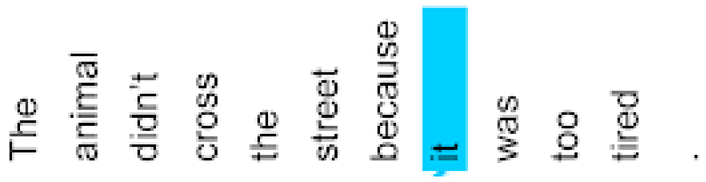
Slide credit: S. Lazebnik

Different Attention Mechanisms

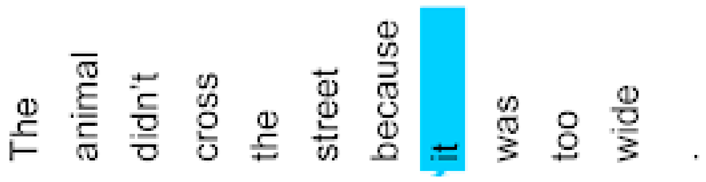


Self-attention

- Used to capture context *within the sequence*



As we are encoding “it”, we should focus on “the animal”



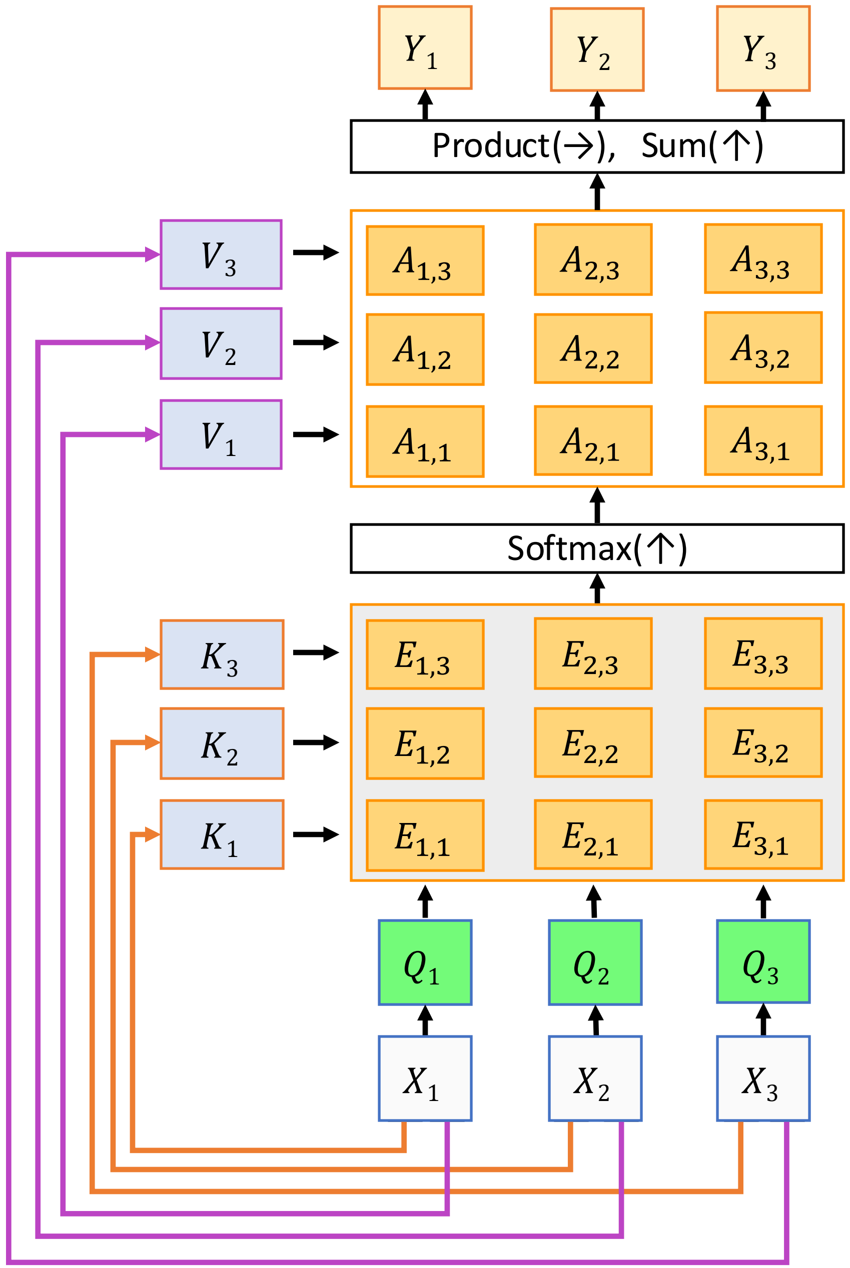
As we are encoding “it”, we should focus on “the street”

Slide credit: S. Lazebnik

Image source

Self-attention layer

- Query vectors: $Q = XW_Q$
- Key vectors: $K = XW_K$
- Value vectors: $V = XW_V$
- Similarities: *scaled dot-product attention*
  - $E_{i,j} = \frac{(Q_i \cdot K_j)}{\sqrt{D}}$ or $E = QK^T / \sqrt{D}$
  - (D is the dimensionality of the keys)
- Attn. weights: $A = \text{softmax}(E, \text{dim} = 1)$
- Output vectors:
  - $Y_i = \sum_j A_{i,j} V_j$ or $Y = AV$

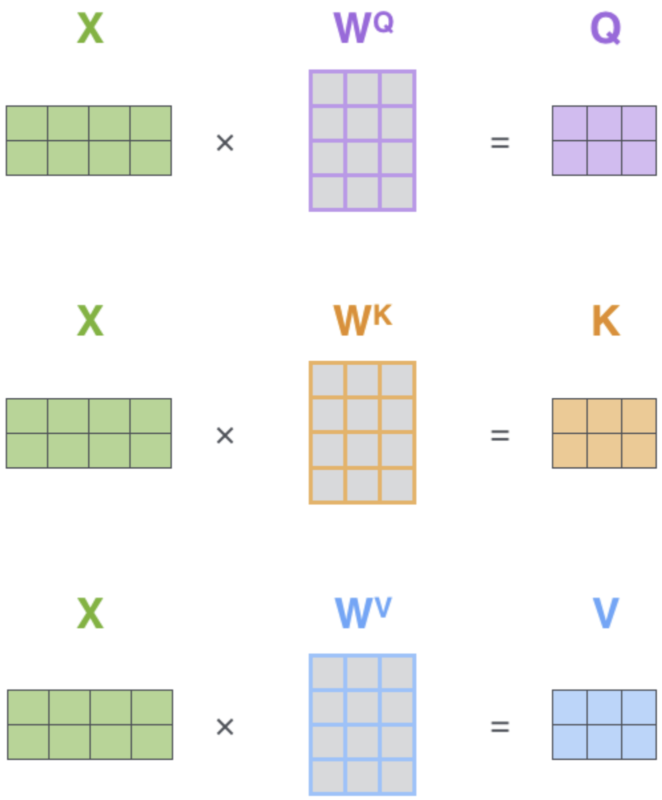


One query per input vector

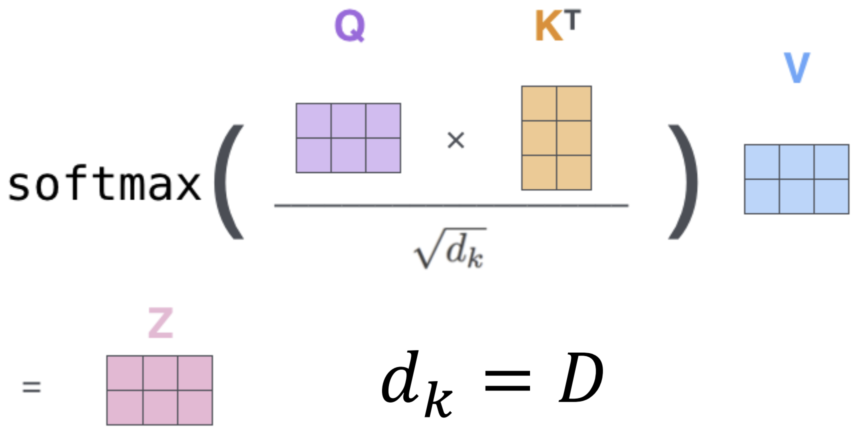
Slide credit: S. Lazebnik

Matrix Calculation of Self-Attention

$X \in R^{L \times C}, W^Q \in R^{C \times D}, \;\; Q \in R^{L \times D}$



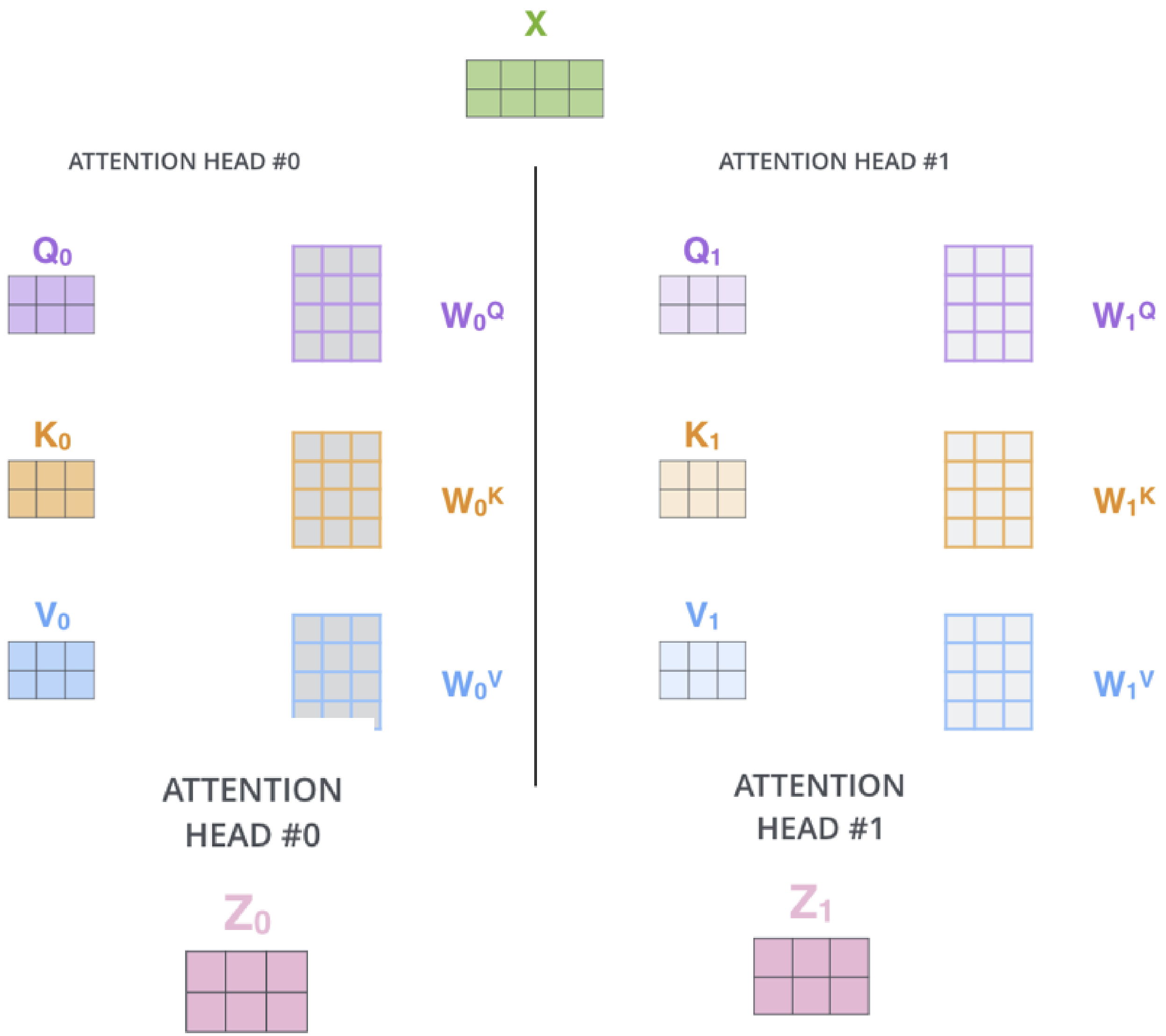
Dimension: $R^{L \times L}$



$d_k = D$

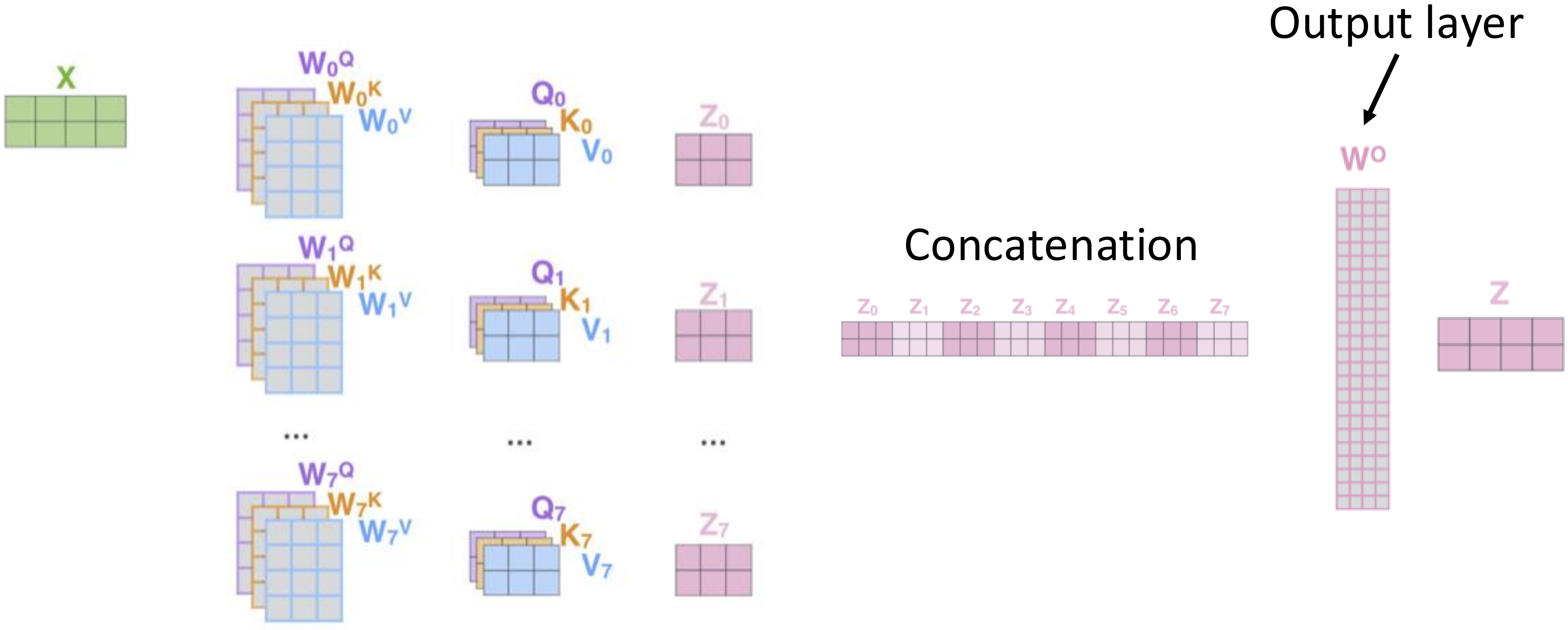
In the implementation, we need to process batched data, where $X \in R^{B \times L \times C}$, check torch.matmul for batched matrix multiplication.

Multi-Head Self-Attention



Intuition:
Multiple attention heads can capture more information in the input

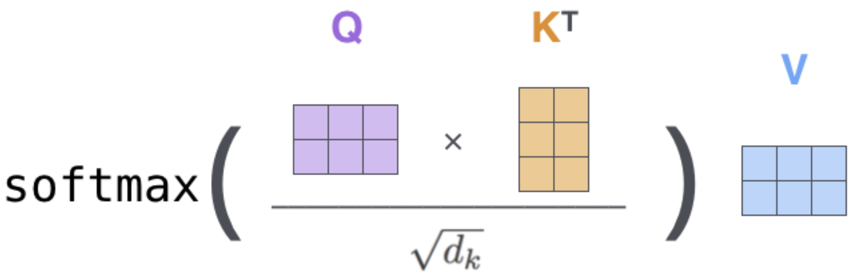
Multi-Head Self-Attention



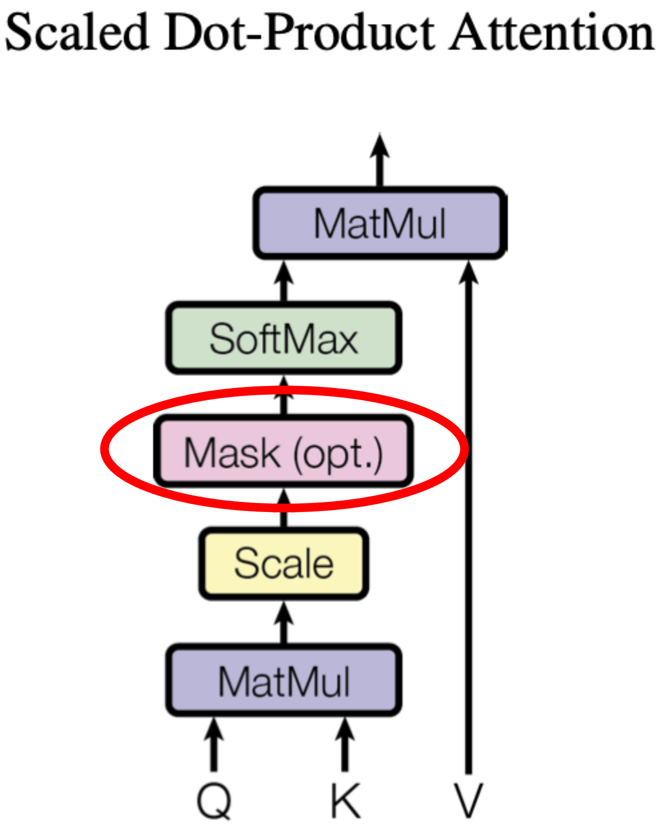
Can we finish the entire operation without any for loops?

Finish Multi-head Attention without For Loops in PA3

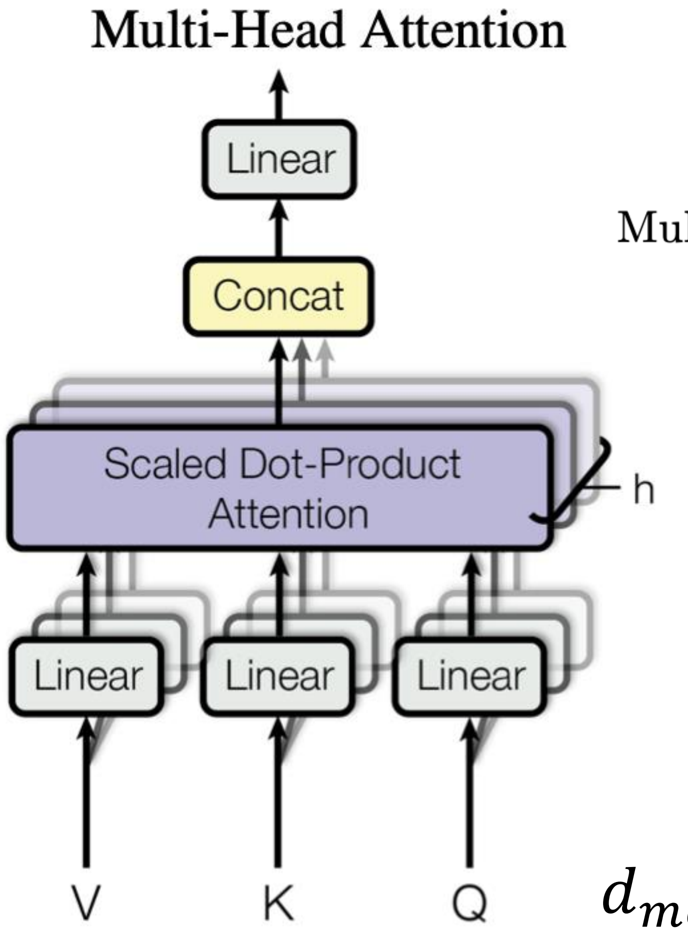
- Linear transformations of Query, Key, and Value
  - Use a big (concatenated) transformation matrix for each of them, respectively
- How about the attention?
  - What are the shapes of Q, K, and V?
    - With and without the multi-head attention
  - The matrix multiplication is defined over a single attention
  - Solution:
    - Reshape the tensors so that the channel dimension is only about a single head
    - Merge the number of heads to the batch dimension



Multi-Head Self-Attention Summary



Will be clear when we talk about the decoder



$$\mathrm{head_i} = \mathrm{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

$$\mathrm{MultiHead}(Q, K, V) = \mathrm{Concat}(\mathrm{head_1}, ..., \mathrm{head_h})W^O$$

$$W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}, W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$$

$$W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$$

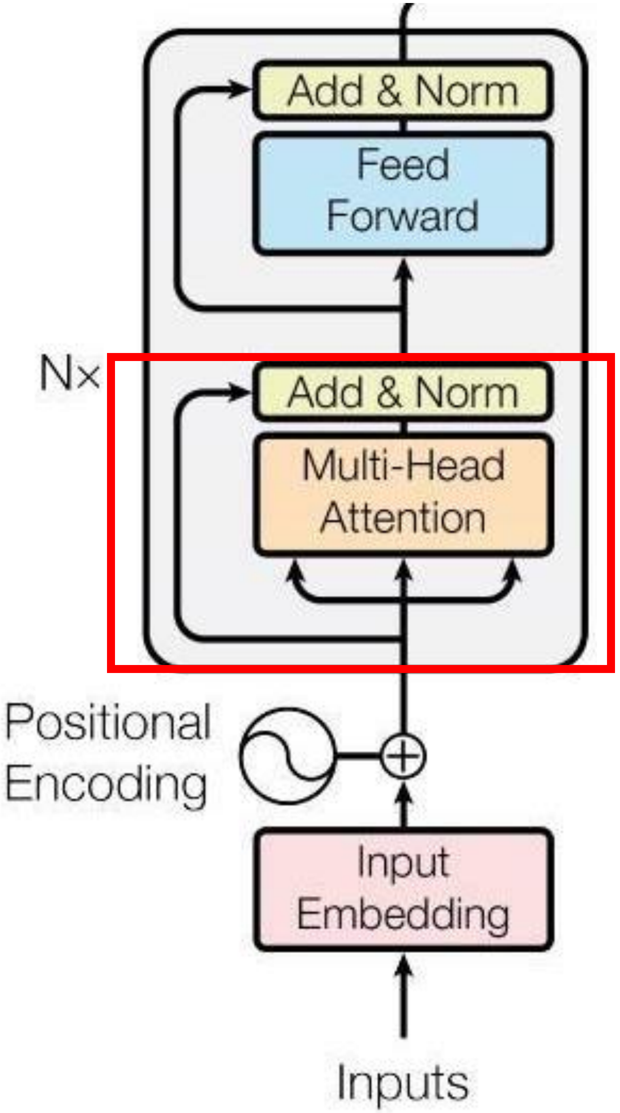
$$W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$$

$$d_k = d_v = d_{\text{model}}/h$$

d_{model}: dimension of input embeddings
h: number of attention heads

[Vaswani et al., Attention is all you need. NeurIPS 2017]

Residual Connections, Dropout, and Normalization



$$X_{att} = \mathrm{MHA}(X)$$
$$X_{att} = \mathrm{Dropout}(X_{att}, p = 0.1)$$
$$X = \mathrm{LayerNorm}(X + X_{att})$$

Recall: Layer Normalization

1. Variable lengths in the input.
2. The recurrent nature of RNN.

Computing of μ, σ is independent of the batch and length dimension.

But the learnable parameters γ, β are still for each channel.

Largely adopted in Transformer.

Layer Normalization for **recurrent** networks

$$x : N \times L \times C$$

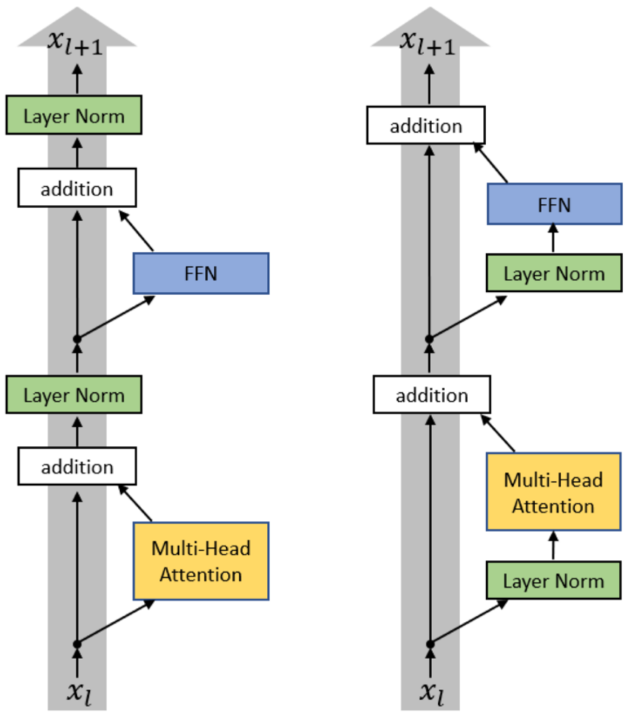
$$\mu, \sigma : N \times L \times 1$$

$$\gamma, \beta : 1 \times 1 \times C$$

$$y = \frac{(x - \mu)}{\sigma} \gamma + \beta$$

On the LayerNorm in Transformer

https://arxiv.org/pdf/2002.04745



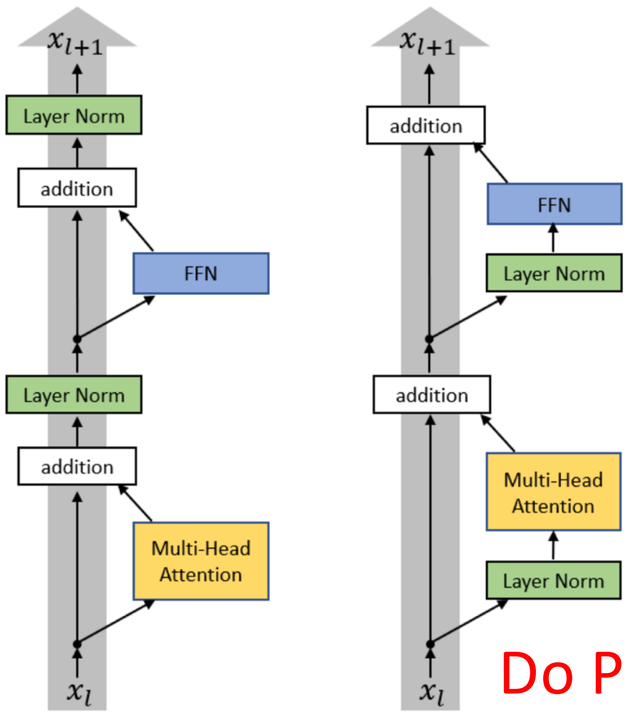
(a) (b)

Implement Post-Norm in PA3

Post-LN Transformer	Pre-LN Transformer
$x_{l,i}^{post,1} = \text{MultiHeadAtt}(x_{l,i}^{post}, [x_{l,1}^{post}, \cdots, x_{l,n}^{post}])$	$x_{l,i}^{pre,1} = \text{LayerNorm}(x_{l,i}^{pre})$
$x_{l,i}^{post,2} = x_{l,i}^{post} + x_{l,i}^{post,1}$	$x_{l,i}^{pre,2} = \text{MultiHeadAtt}(x_{l,i}^{pre,1}, [x_{l,1}^{pre,1}, \cdots, x_{l,n}^{pre,1}])$
$x_{l,i}^{post,3} = \text{LayerNorm}(x_{l,i}^{post,2})$	$x_{l,i}^{pre,3} = x_{l,i}^{pre} + x_{l,i}^{pre,2}$
$x_{l,i}^{post,4} = \text{ReLU}(x_{l,i}^{post,3} W^{1,l} + b^{1,l}) W^{2,l} + b^{2,l}$	$x_{l,i}^{pre,4} = \text{LayerNorm}(x_{l,i}^{pre,3})$
$x_{l,i}^{post,5} = x_{l,i}^{post,3} + x_{l,i}^{post,4}$	$x_{l,i}^{pre,5} = \text{ReLU}(x_{l,i}^{pre,4} W^{1,l} + b^{1,l}) W^{2,l} + b^{2,l}$
$x_{l+1,i}^{post} = \text{LayerNorm}(x_{l,i}^{post,5})$	$x_{l+1,i}^{pre} = x_{l,i}^{pre,5} + x_{l,i}^{pre,3}$
	Final LayerNorm: $x_{Final,i}^{pre} \leftarrow \text{LayerNorm}(x_{L+1,i}^{pre})$

On the LayerNorm in Transformer

https://arxiv.org/pdf/2002.04745



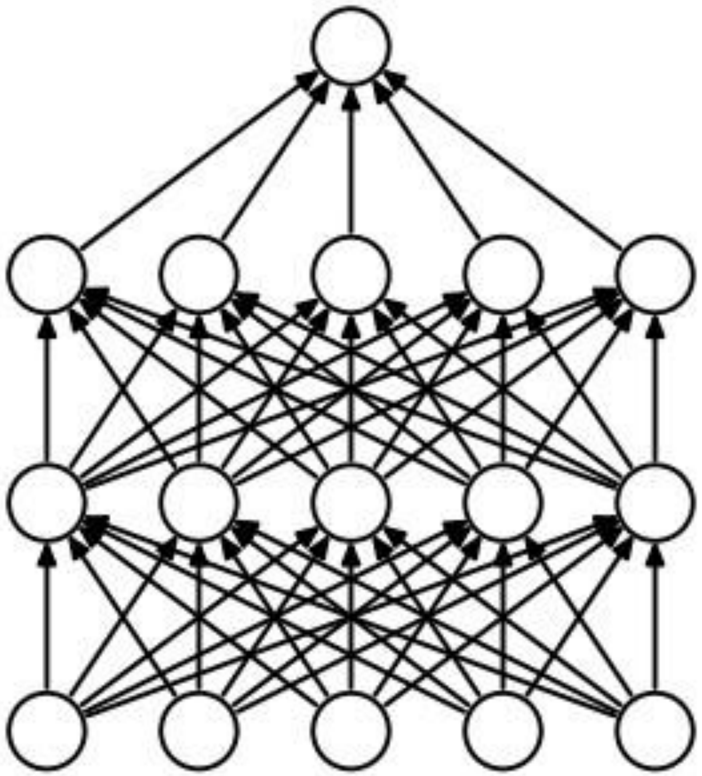
(a) (b)

Do Pre-Norm in practice

Post-LN Transformer	Pre-LN Transformer
$x_{l,i}^{post,1} = \text{MultiHeadAtt}(x_{l,i}^{post}, [x_{l,1}^{post}, \cdots, x_{l,n}^{post}])$	$x_{l,i}^{pre,1} = \text{LayerNorm}(x_{l,i}^{pre})$
$x_{l,i}^{post,2} = x_{l,i}^{post} + x_{l,i}^{post,1}$	$x_{l,i}^{pre,2} = \text{MultiHeadAtt}(x_{l,i}^{pre,1}, [x_{l,1}^{pre,1}, \cdots, x_{l,n}^{pre,1}])$
$x_{l,i}^{post,3} = \text{LayerNorm}(x_{l,i}^{post,2})$	$x_{l,i}^{pre,3} = x_{l,i}^{pre} + x_{l,i}^{pre,2}$
$x_{l,i}^{post,4} = \text{ReLU}(x_{l,i}^{post,3} W^{1,l} + b^{1,l}) W^{2,l} + b^{2,l}$	$x_{l,i}^{pre,4} = \text{LayerNorm}(x_{l,i}^{pre,3})$
$x_{l,i}^{post,5} = x_{l,i}^{post,3} + x_{l,i}^{post,4}$	$x_{l,i}^{pre,5} = \text{ReLU}(x_{l,i}^{pre,4} W^{1,l} + b^{1,l}) W^{2,l} + b^{2,l}$
$x_{l+1,i}^{post} = \text{LayerNorm}(x_{l,i}^{post,5})$	$x_{l+1,i}^{pre} = x_{l,i}^{pre,5} + x_{l,i}^{pre,3}$
	Final LayerNorm: $x_{Final,i}^{pre} \leftarrow \text{LayerNorm}(x_{L+1,i}^{pre})$

Regularization: **Dropout**

“randomly set some neurons to zero in the forward pass”



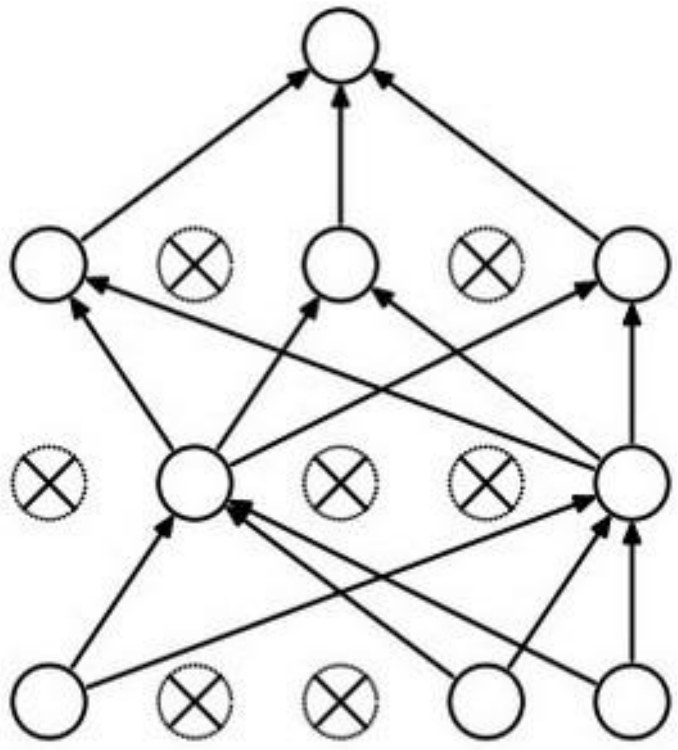
(a) Standard Neural Net

Randomly set the output value of some neurons to be 0

Slide credit: E. Learned-Miller, originally developed by Andrej Karpathy, Justin Johnson, Fei-Fei Li

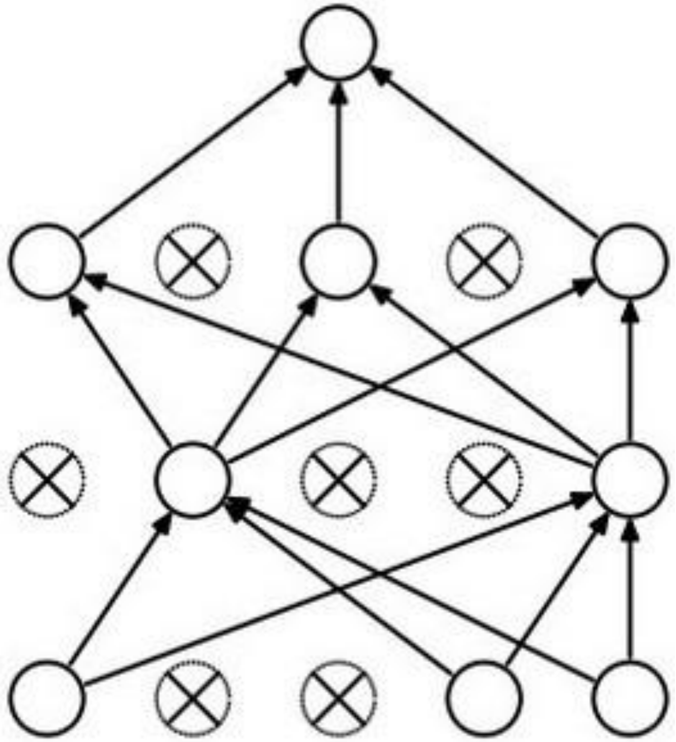
Waaaait a second…

How could this possibly be a good idea?

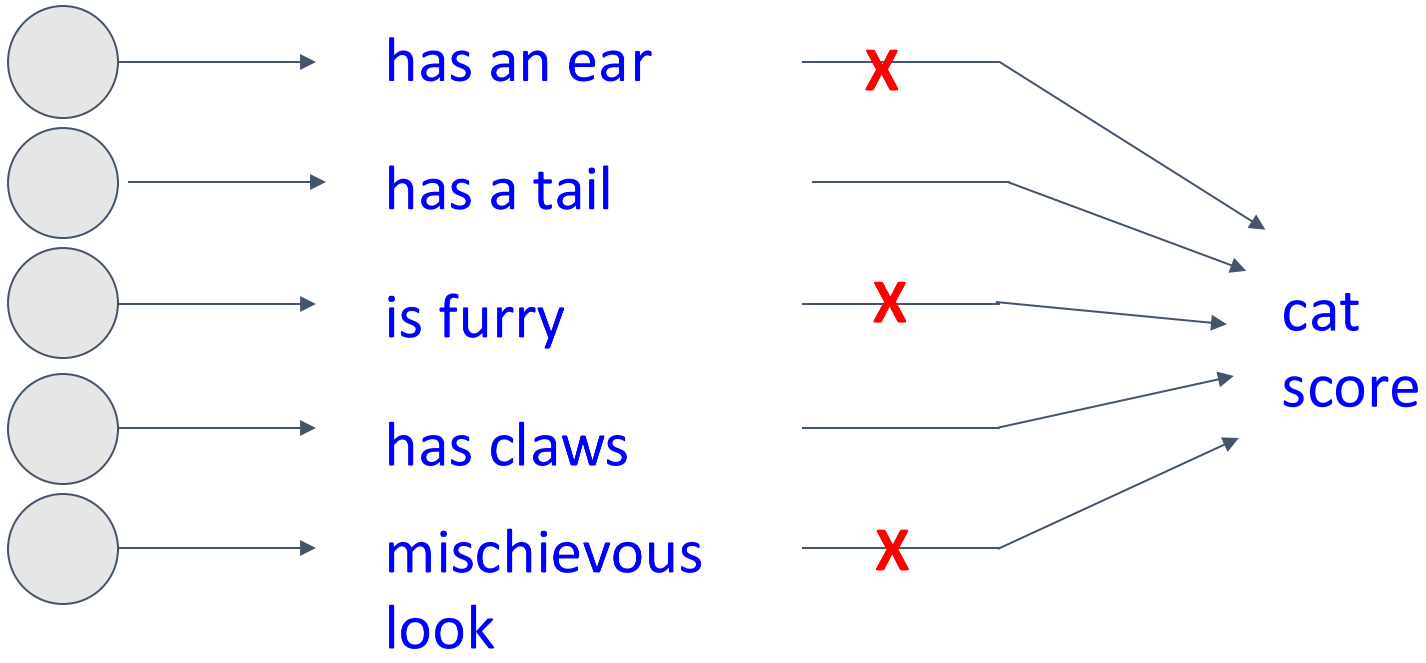


Slide credit: E. Learned-Miller, originally developed by Andrej Karpathy, Justin Johnson, Fei-Fei Li

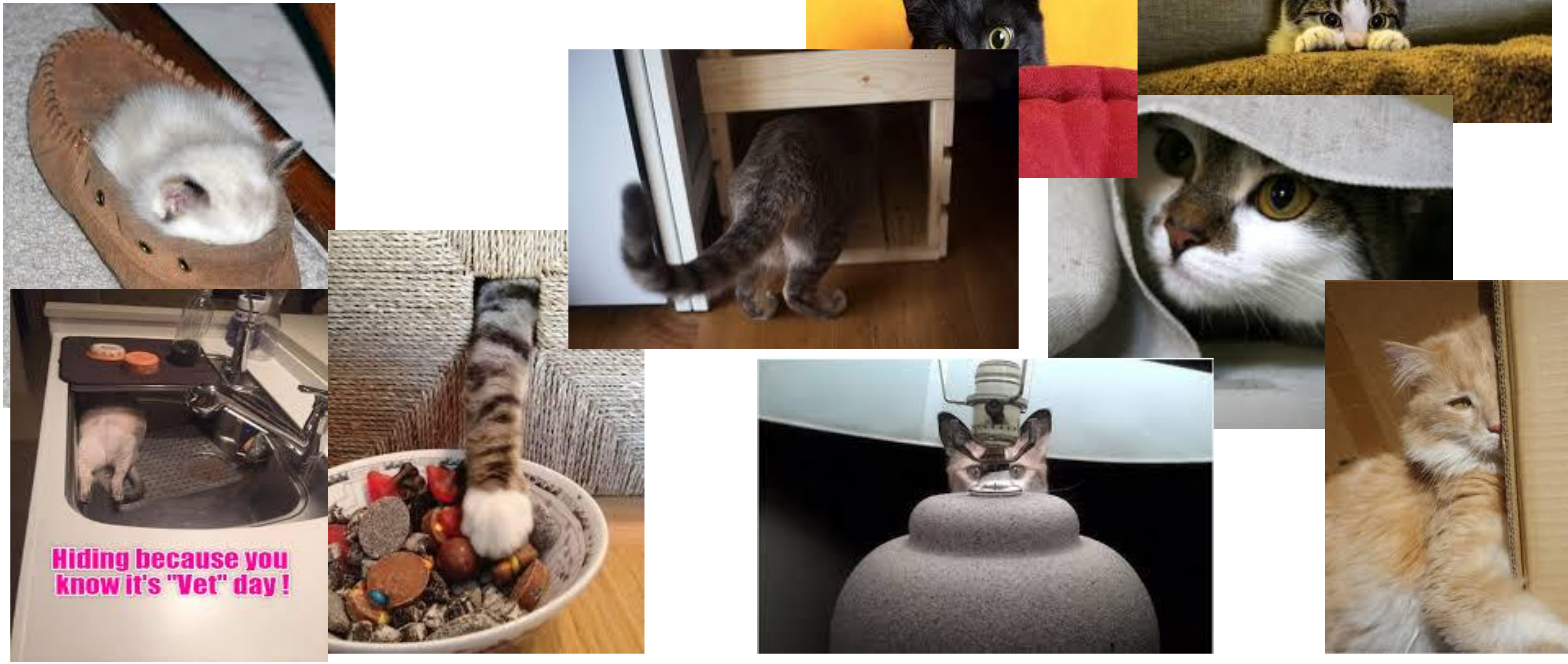
Waaaait a second…
How could this possibly be a good idea?



Forces the network to have a redundant representation.

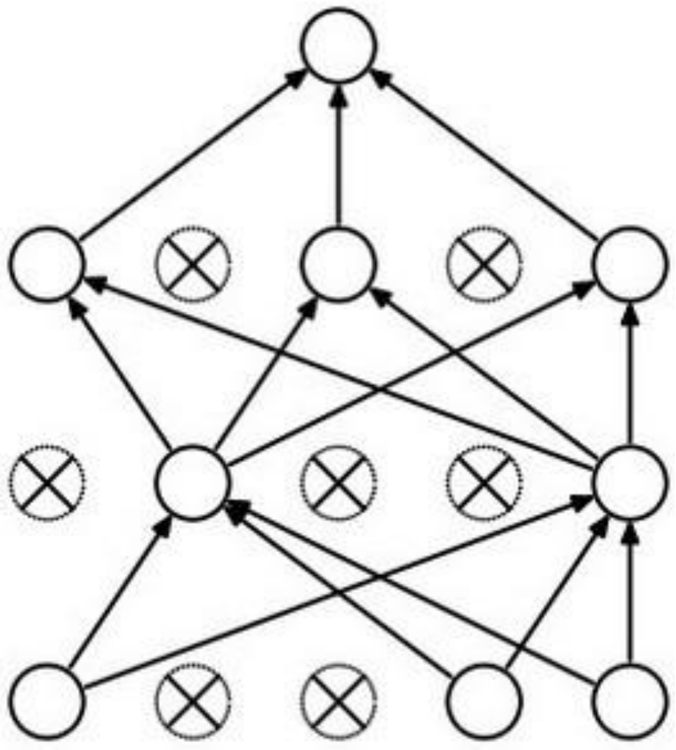


Training with occlusions?



Slide credit: E. Learned-Miller, originally developed by Andrej Karpathy, Justin Johnson, Fei-Fei Li

Waaaait a second…
How could this possibly be a good idea?



Another interpretation:

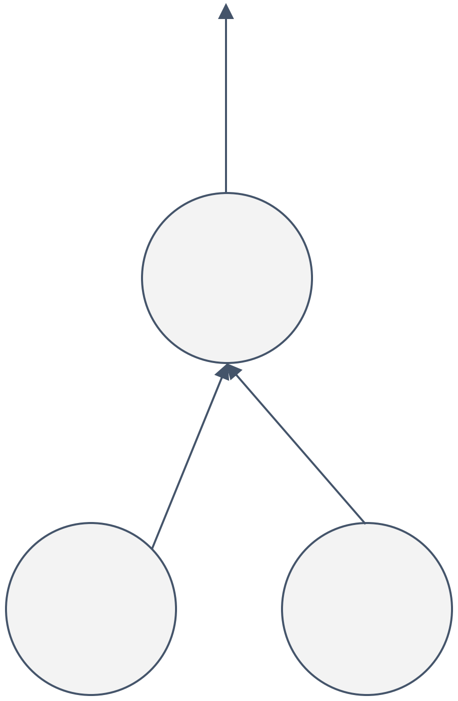
Dropout is training a large ensemble of models (that share parameters).

Each binary mask is one model, gets trained on only ~one datapoint.

Dropout: Test Time

Can in fact do this with a single forward pass! (approximately)

Leave all input neurons turned on (no dropout).



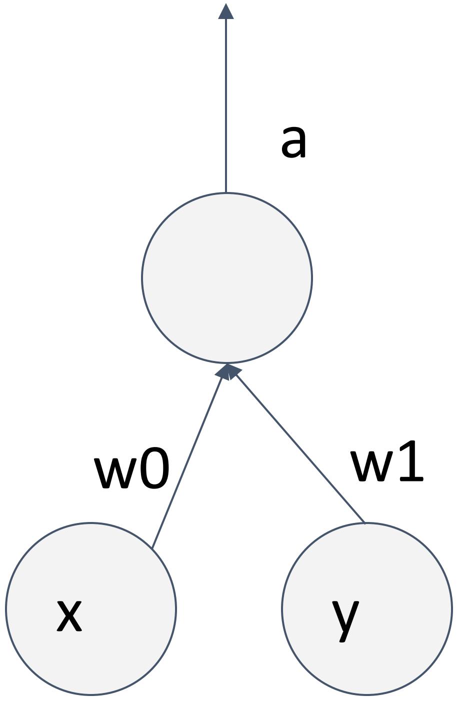
Q: Suppose that with all inputs present at test time the output of this neuron is *a*.

What would its output be during training time, in expectation? (e.g. if p = 0.5)

Dropout: Test Time

Can in fact do this with a single forward pass! (approximately)

Leave all input neurons turned on (no dropout).



during test: $\boldsymbol{a = w_0 * x + w_1 * y}$

during train:

$$E[a] = \tfrac{1}{4} * (w_0*0 + w_1*0 + w_0*0 + w_1*y + w_0*x + w_1*0 + w_0*x + w_1*y)$$

$$= \tfrac{1}{4} * (2\, w_0*x + 2\, w_1*y)$$

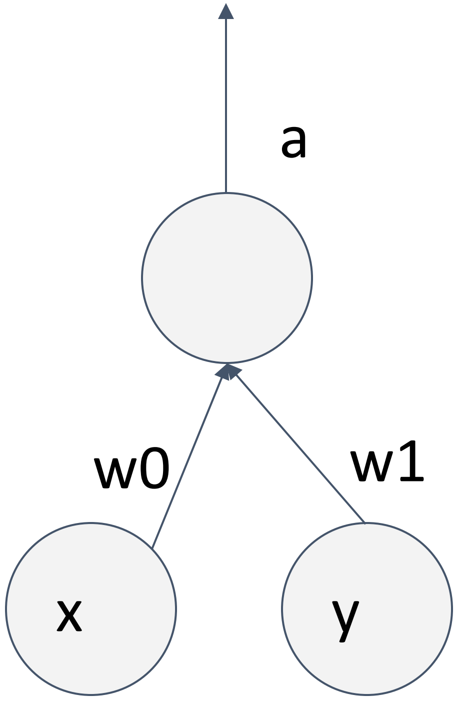
$$= \boldsymbol{\tfrac{1}{2} * (w_0*x + w_1*y)}$$

Slide credit: E. Learned-Miller, originally developed by Andrej Karpathy, Justin Johnson, Fei-Fei Li

At test time….

Can in fact do this with a single forward pass! (approximately)

Leave all input neurons turned on (no dropout).



during test: $\boldsymbol{a = w_0 * x + w_1 * y}$

during train:

$$E[a] = \tfrac{1}{4} * (w_0 * 0 + w_1 * 0 + w_0 * 0 + w_1 * y + w_0 * x + w_1 * 0 + w_0 * x + w_1 * y)$$

$$= \tfrac{1}{4} * (2\, w_0 * x + 2\, w_1 * y)$$

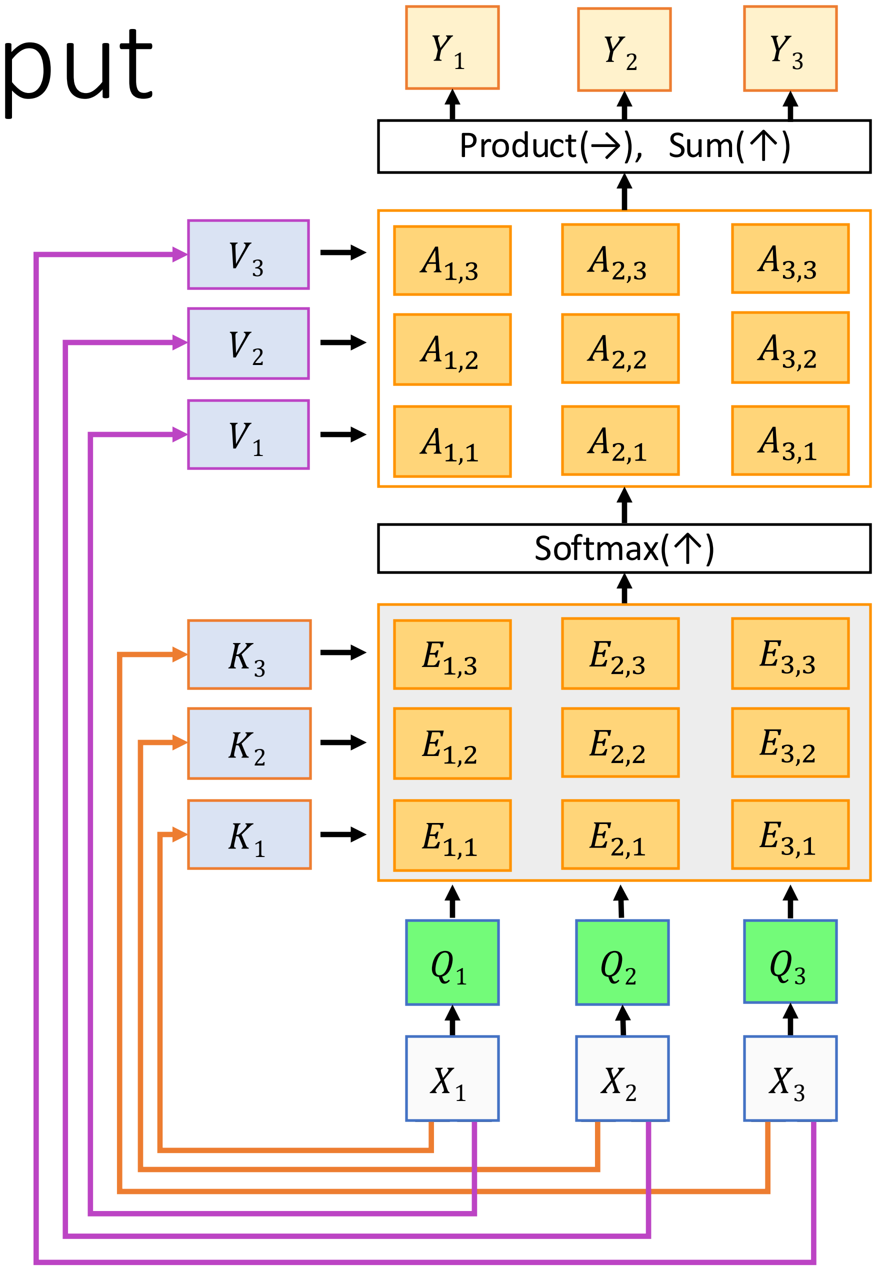
$$= \boldsymbol{\tfrac{1}{2} * (w_0 * x + w_1 * y)}$$

With p=0.5, using all inputs in the forward pass would inflate the activations by 2x from what the network was "used to" during training!
=> Have to compensate by scaling the activations back down by ½

Slide credit: E. Learned-Miller, originally developed by Andrej Karpathy, Justin Johnson, Fei-Fei Li

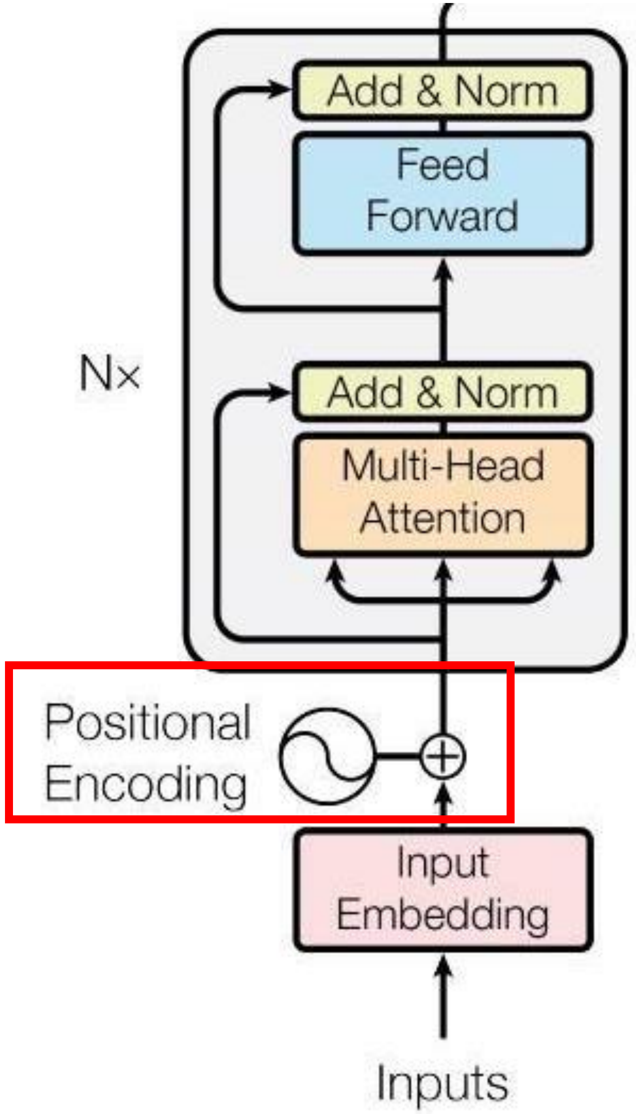
Capturing the Order of the Input

- Do the embeddings of each token/input change if we randomly permute the input?
- Is it good or bad?



One query per input vector

Augmenting the MSA with Positional Encoding



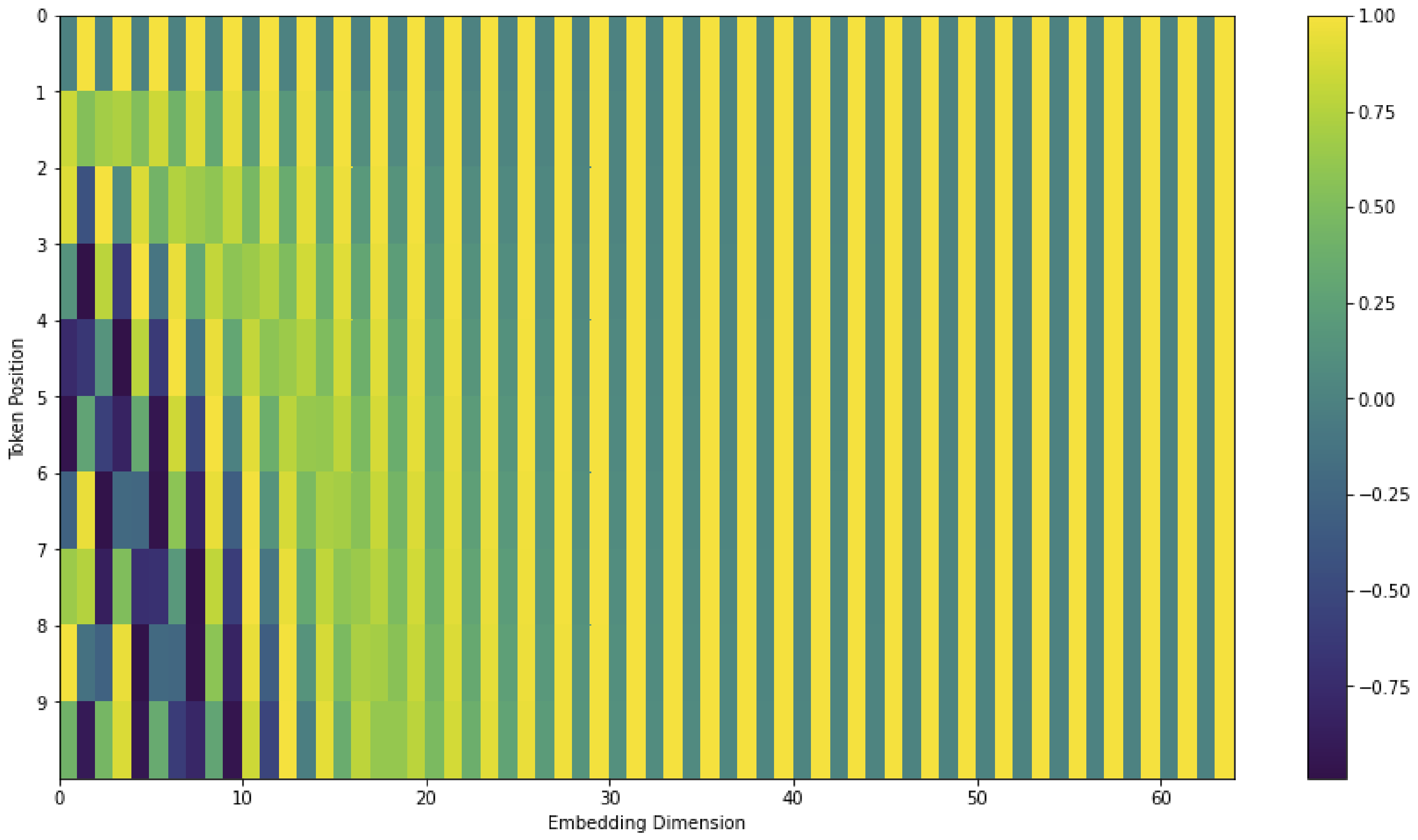
$$X = X + PE(X)$$

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

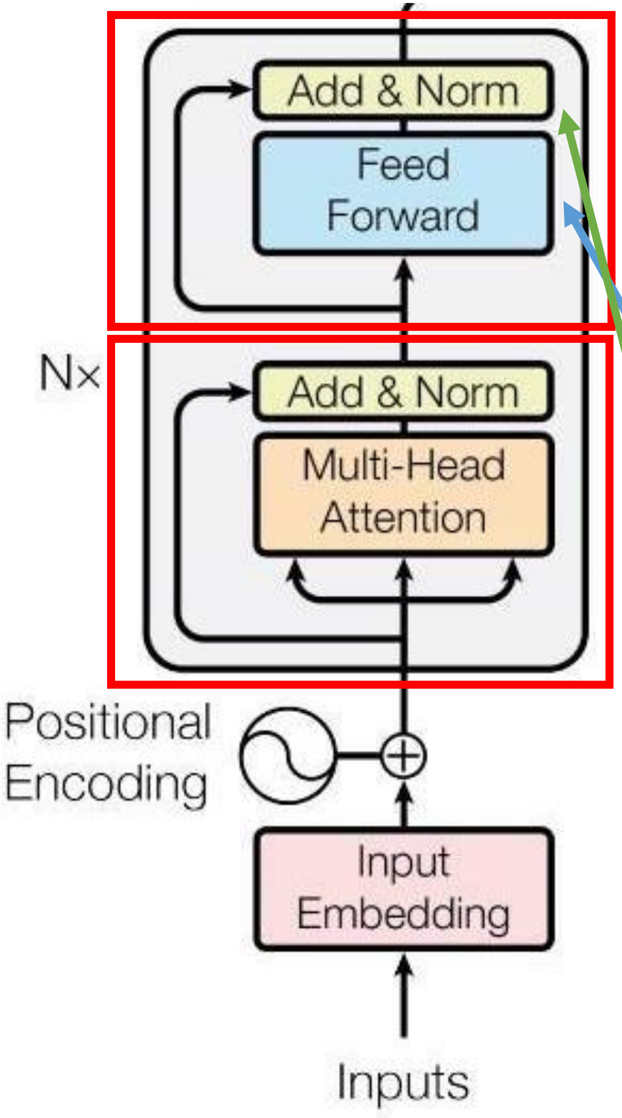
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

- *PE* has the same dimension with the input embeddings of each token.
- *pos*: Position of each token in the input, [*0, L-1*]
- *i*: index of the embeddings, [*0*, d_{model} *-1*]
- What does 10000 mean here?
  - The maximum input length

What Does Positional Encoding Look Like?



Feedforward Network

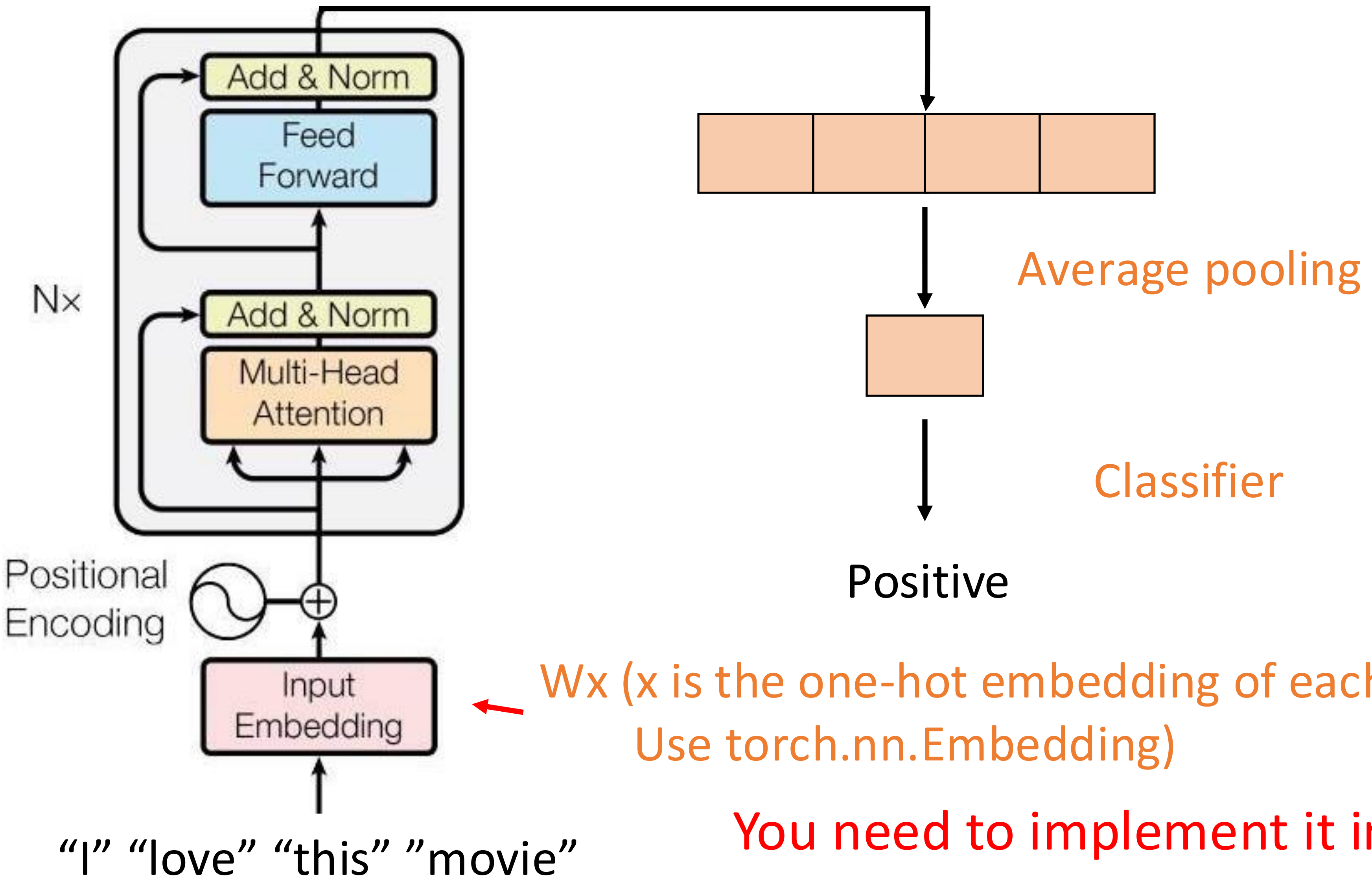


Are there any non-linearities in the lower part?

A two-layer fully-connected network (one hidden layer).

Residual connections, dropout, and normalization work in a similar way to the multi-head self-attention module

Transformer Encoder for Text Classification

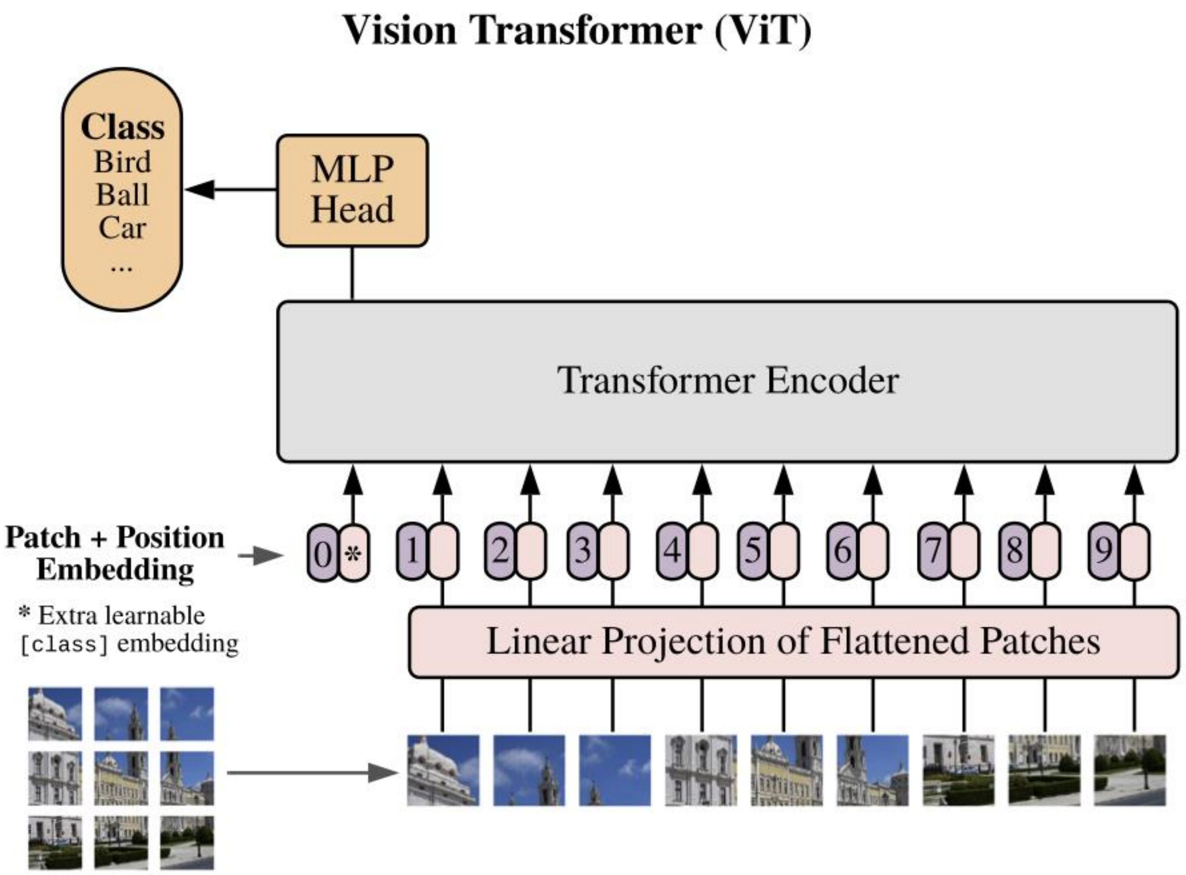


Stack of *N* Transformer Encoder Cell

Wx (x is the one-hot embedding of each token. Use torch.nn.Embedding)

You need to implement it in PA3

Transformer Encoder for Image Classification



How to represent each patch?

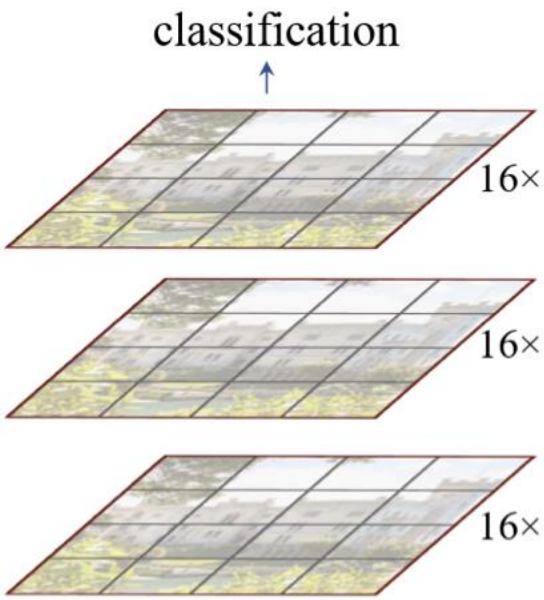
Flattened RGB pixels

Extra credits in PA3:

- Use the special [CLASS] token to aggregate the image's information
- Alternatively, we get embeddings of each patch and do average pooling as in the previous example
- The details of the Transformer encoder is slightly different
- Positional encoding: 1D is sufficient.

[Dosovitskiy et al., An image is worth 16x16 words. ICLR 2021]

SwinTransformer



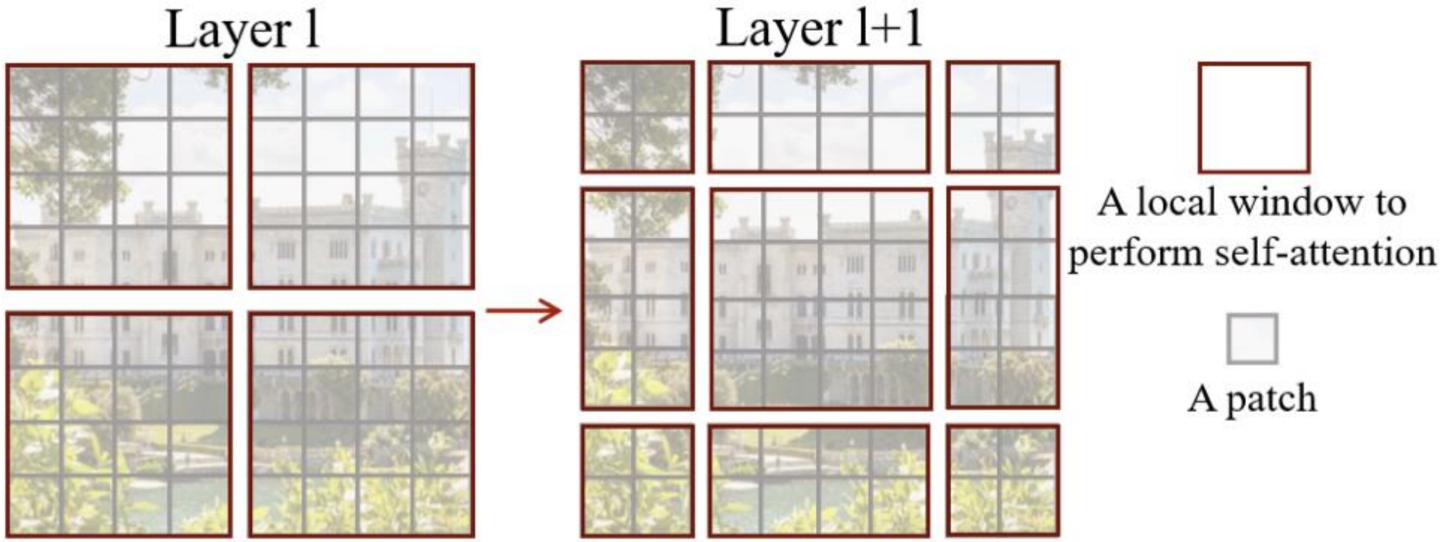
(b) ViT

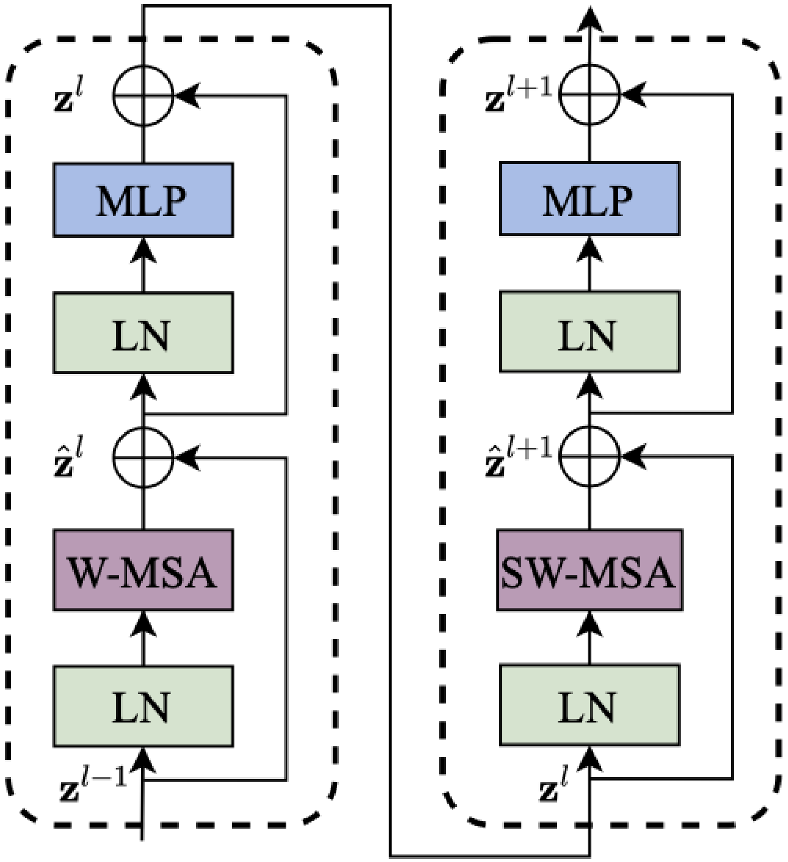
Lesson we learned from CNNs:

- We process the input at different hierarchies (resolutions) for efficiency purpose.
- We process the input at different hierarchies (resolutions) for different receptive field sizes.
- We need to fuse features from different hierarchies (resolutions) for dense prediction tasks

[Liu et al., Swin Transformer: Hierarchical Vision Transformer using Shifted Windows. ICCV 2021. Best paper award.]

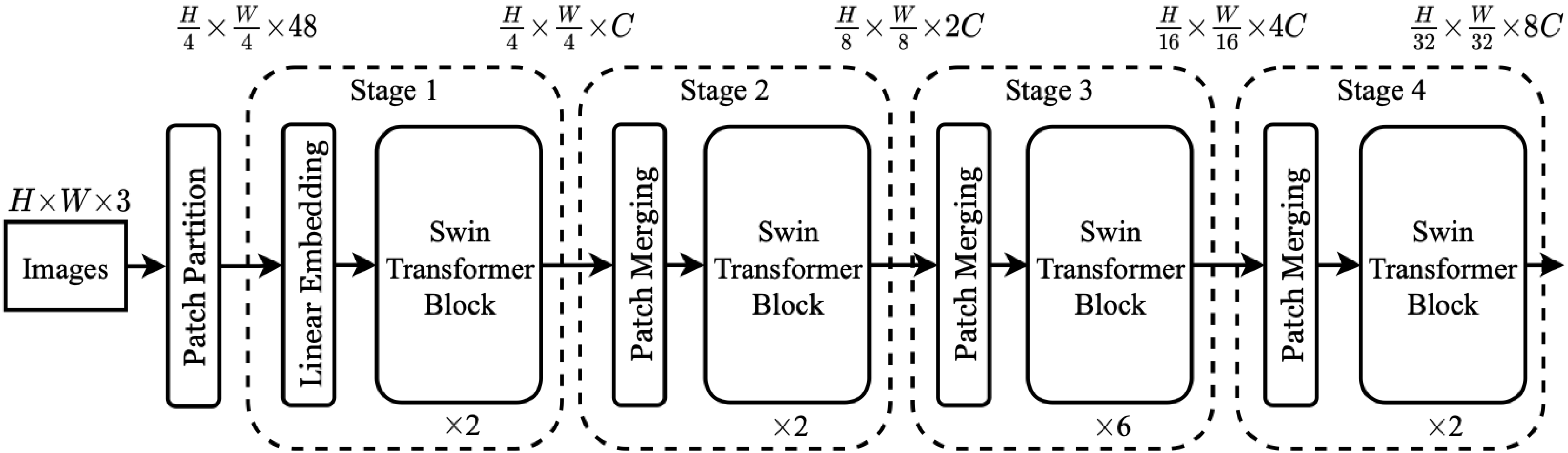
Shifted Window-based Efficient Attention





[Liu et al., Swin Transformer: Hierarchical Vision Transformer using Shifted Windows. ICCV 2021. Best paper award.]

Hierarchical Representations in SwinTransformer



[Liu et al., Swin Transformer: Hierarchical Vision Transformer using Shifted Windows. ICCV 2021. Best paper award.]

Transformer vs CNN

(a) Various frameworks

Method	Backbone	AP^{box}	AP^{box}_{50}	AP^{box}_{75}	#param.	FLOPs	FPS
Cascade Mask R-CNN	R-50	46.3	64.3	50.5	82M	739G	18.0
	Swin-T	**50.5**	**69.3**	**54.9**	86M	745G	15.3
ATSS	R-50	43.5	61.9	47.0	32M	205G	28.3
	Swin-T	**47.2**	**66.5**	**51.3**	36M	215G	22.3
RepPointsV2	R-50	46.5	64.6	50.3	42M	274G	13.6
	Swin-T	**50.0**	**68.5**	**54.2**	45M	283G	12.0
Sparse R-CNN	R-50	44.5	63.4	48.2	106M	166G	21.0
	Swin-T	**47.9**	**67.3**	**52.3**	110M	172G	18.4

(b) Various backbones w. Cascade Mask R-CNN

	AP^{box}	AP^{box}_{50}	AP^{box}_{75}	AP^{mask}	AP^{mask}_{50}	AP^{mask}_{75}	param	FLOPs	FPS
DeiT-S†	48.0	67.2	51.7	41.4	64.2	44.3	80M	889G	10.4
R50	46.3	64.3	50.5	40.1	61.7	43.4	82M	739G	18.0
Swin-T	**50.5**	**69.3**	**54.9**	**43.7**	**66.6**	**47.1**	86M	745G	15.3
X101-32	48.1	66.5	52.4	41.6	63.9	45.2	101M	819G	12.8
Swin-S	**51.8**	**70.4**	**56.3**	**44.7**	**67.9**	**48.5**	107M	838G	12.0
X101-64	48.3	66.4	52.3	41.7	64.0	45.1	140M	972G	10.4
Swin-B	**51.9**	**70.9**	**56.5**	**45.0**	**68.4**	**48.7**	145M	982G	11.6

Next Class

- Transformer Decoder