Advanced Computer Vision

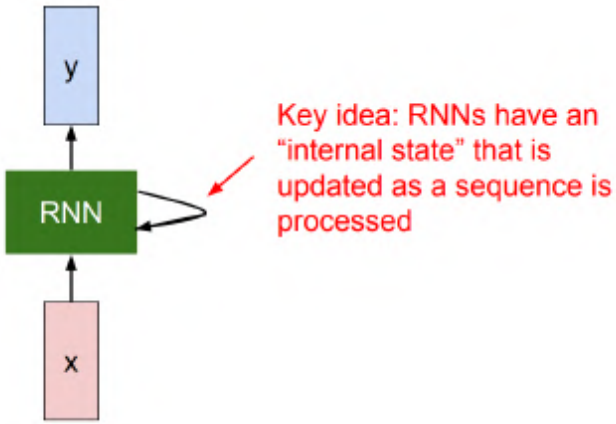
Naeemullah Khan
naeemullah.khan@kaust.edu.sa

جامعة الملك عبدالله
للعلوم والتقنية
King Abdullah University of
Science and Technology

KAUST Academy
King Abdullah University of Science and Technology

June 15, 2023

Recurrent Neural Networks (RNN)



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

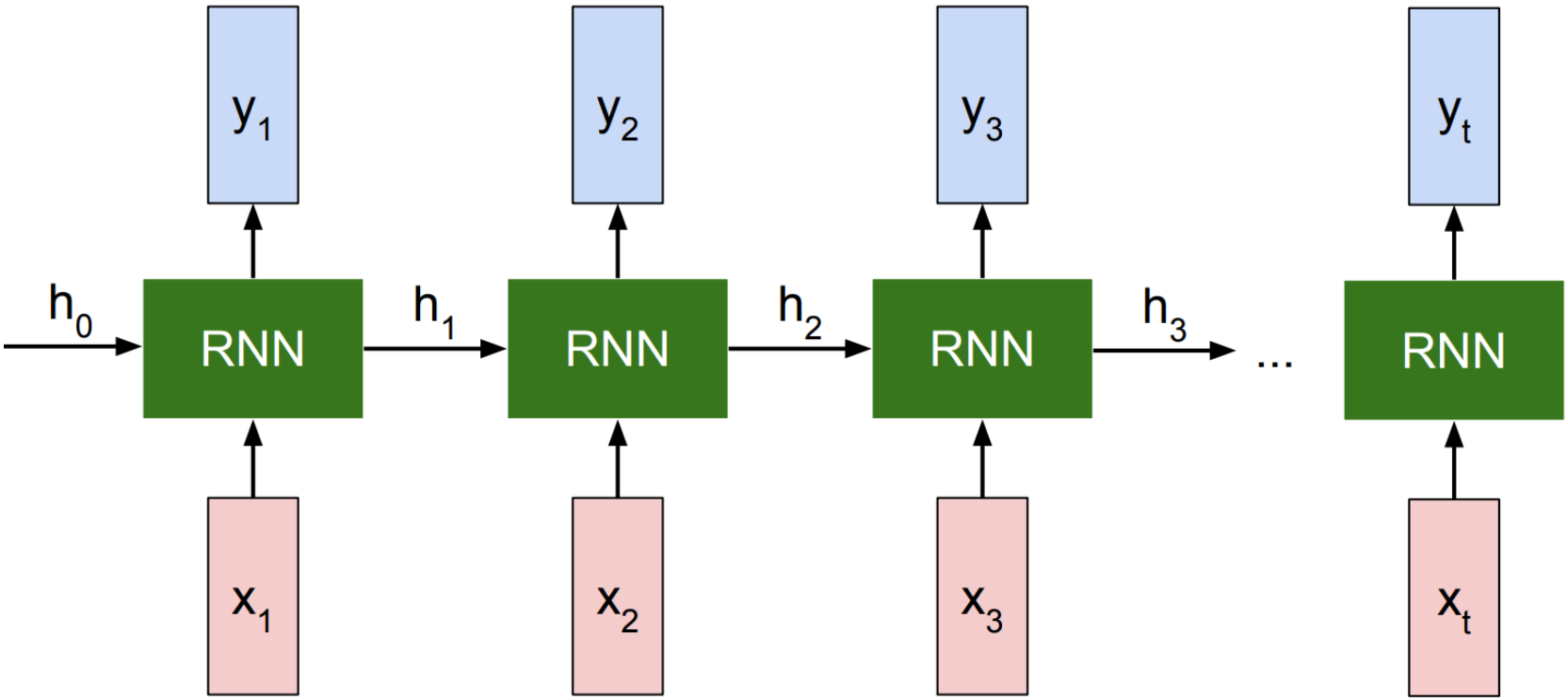
new state

some function with parameters W

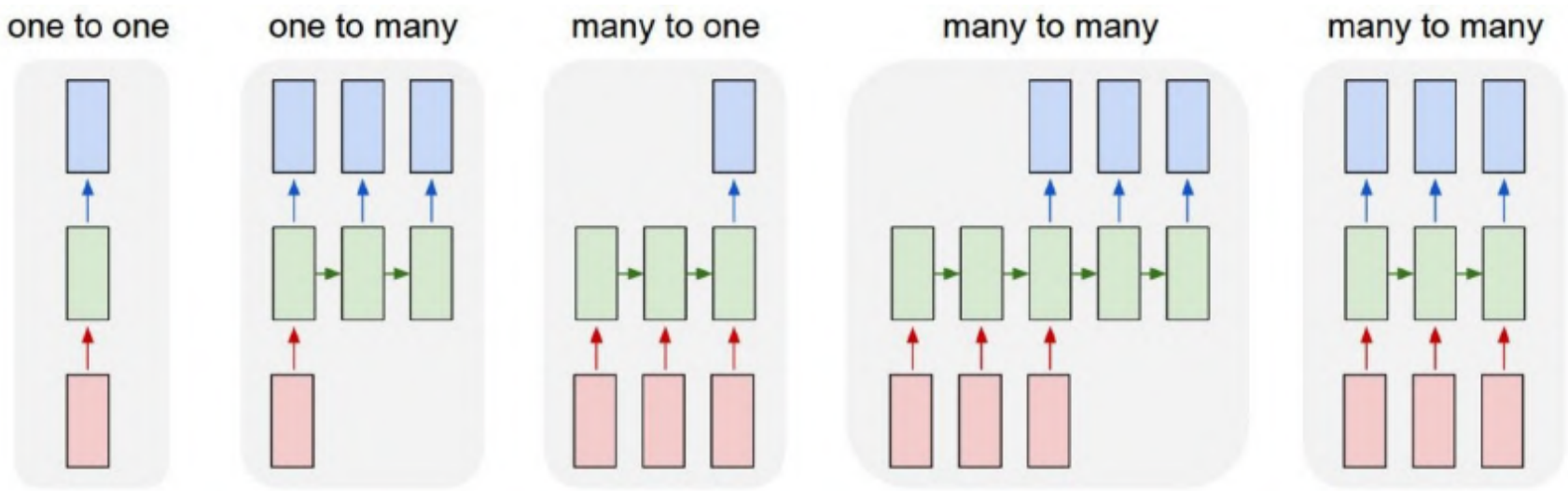
old state

input vector at some time step

Recurrent Neural Networks (RNN)



Recurrent Neural Networks (RNN)



Recurrent Neural Networks (RNN)

- RNN Advantages:
  - Can process any length input
  - Computation for step t can (in theory) use information from many steps back
  - Model size doesn't increase for longer input
  - Same weights applied on every timestep, so there is symmetry in how inputs are processed.

Recurrent Neural Networks (RNN)

- RNN Advantages:
  - Can process any length input
  - Computation for step t can (in theory) use information from many steps back
  - Model size doesn't increase for longer input
  - Same weights applied on every timestep, so there is symmetry in how inputs are processed.
- RNN Disadvantages:
  - Recurrent computation is slow
  - In practice, difficult to access information from many steps back

Image Captioning

- A computer Vision task in which we generate captions for images

A cat sitting on a suitcase on the floor

A cat is sitting on a tree branch

A dog is running in the grass with a frisbee

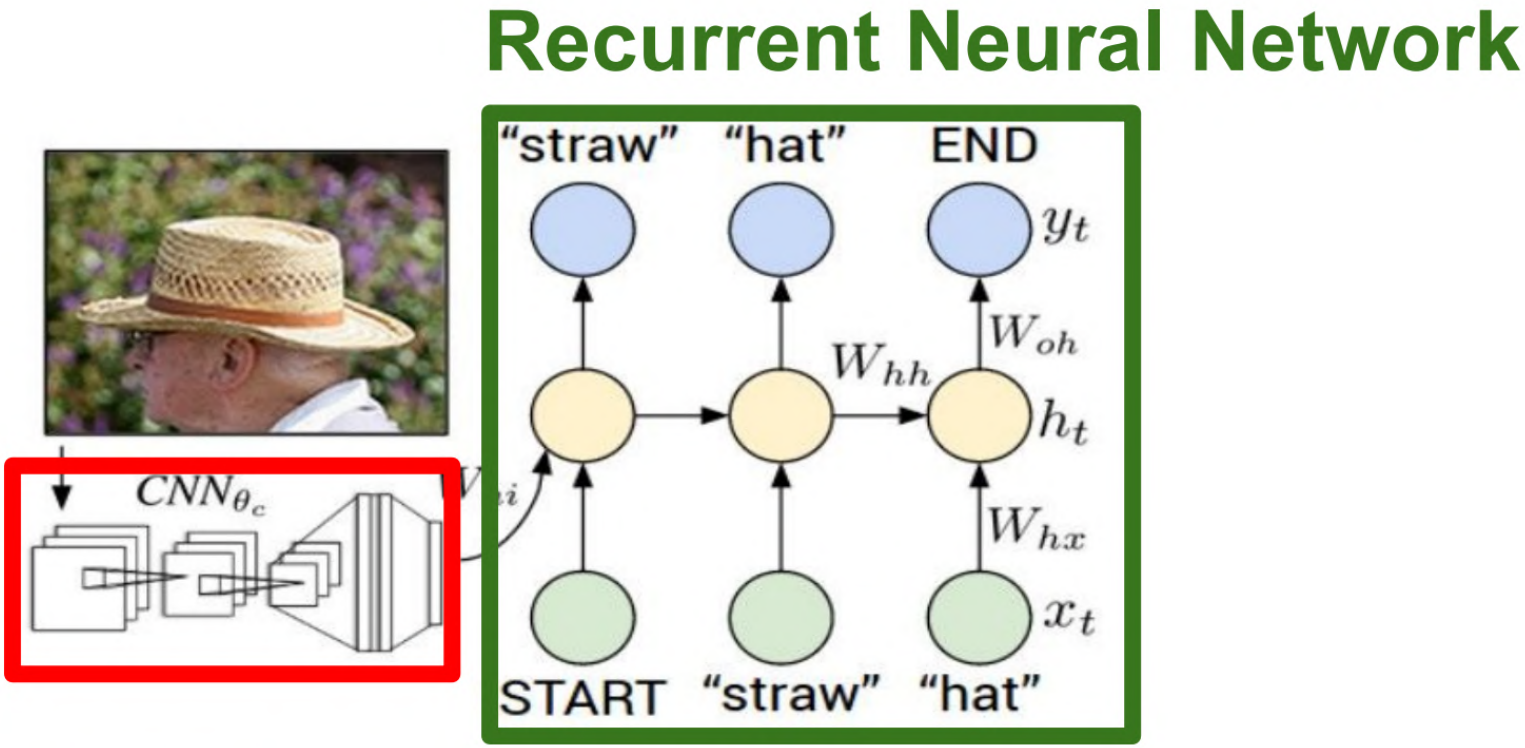
A white teddy bear sitting in the grass

Two people walking on the beach with surfboards

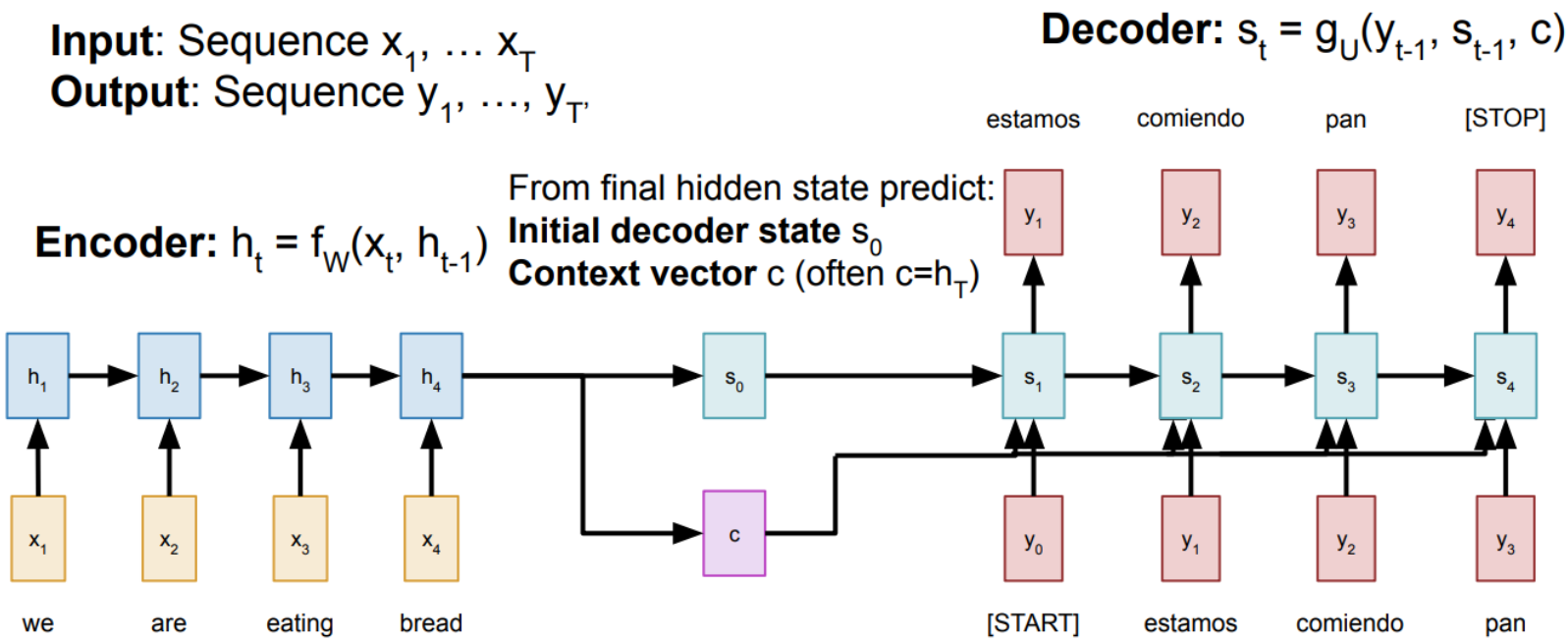
A tennis player in action on the court

Two giraffes standing in a grassy field

A man riding a dirt bike on a dirt track

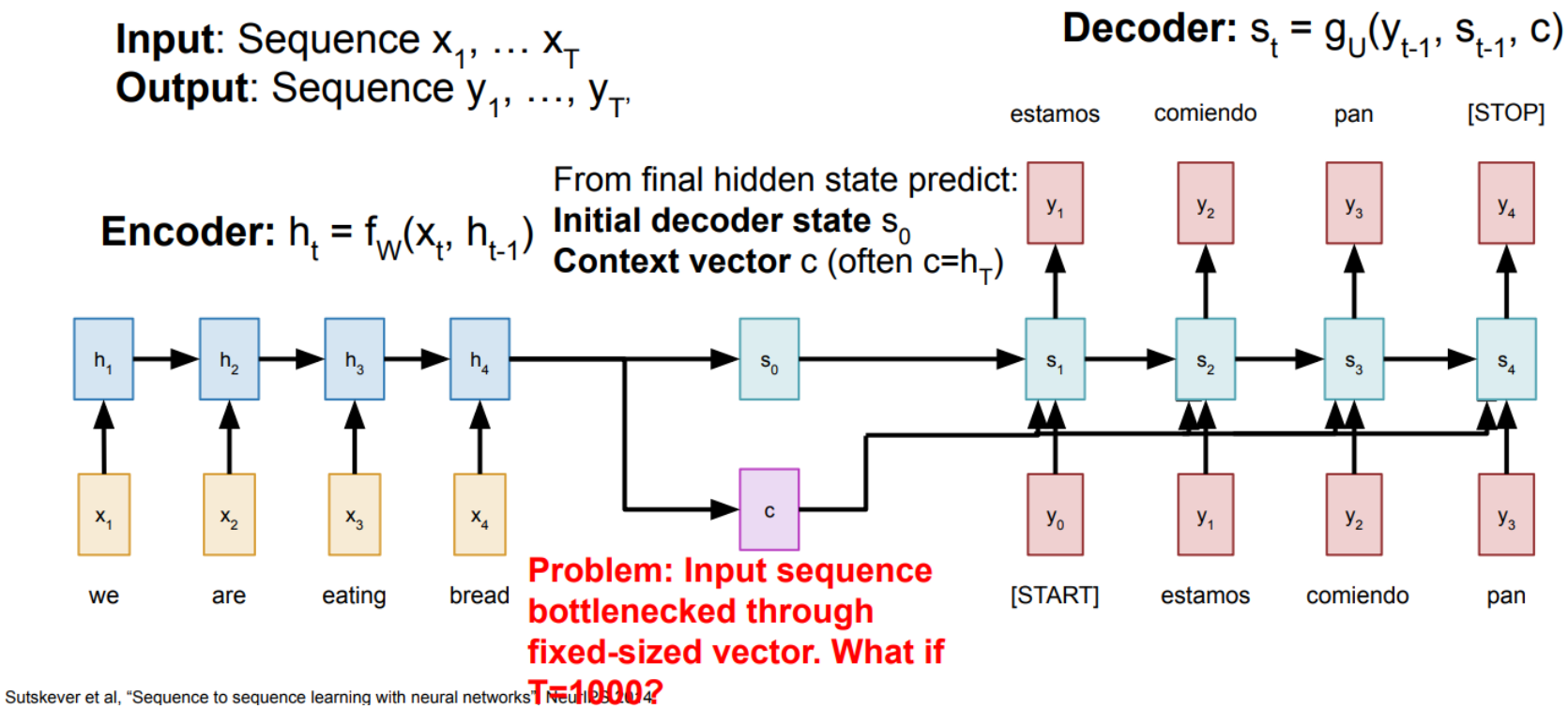

Recurrent Neural Network
"straw"
"hat"
END
y_t
W_{oh}
W_{hh}
h_t
W_{hx}
x_t
START
"straw"
"hat"
CNN_{θ_c}
Convolutional Neural Network

Sequence to Sequence with RNNs



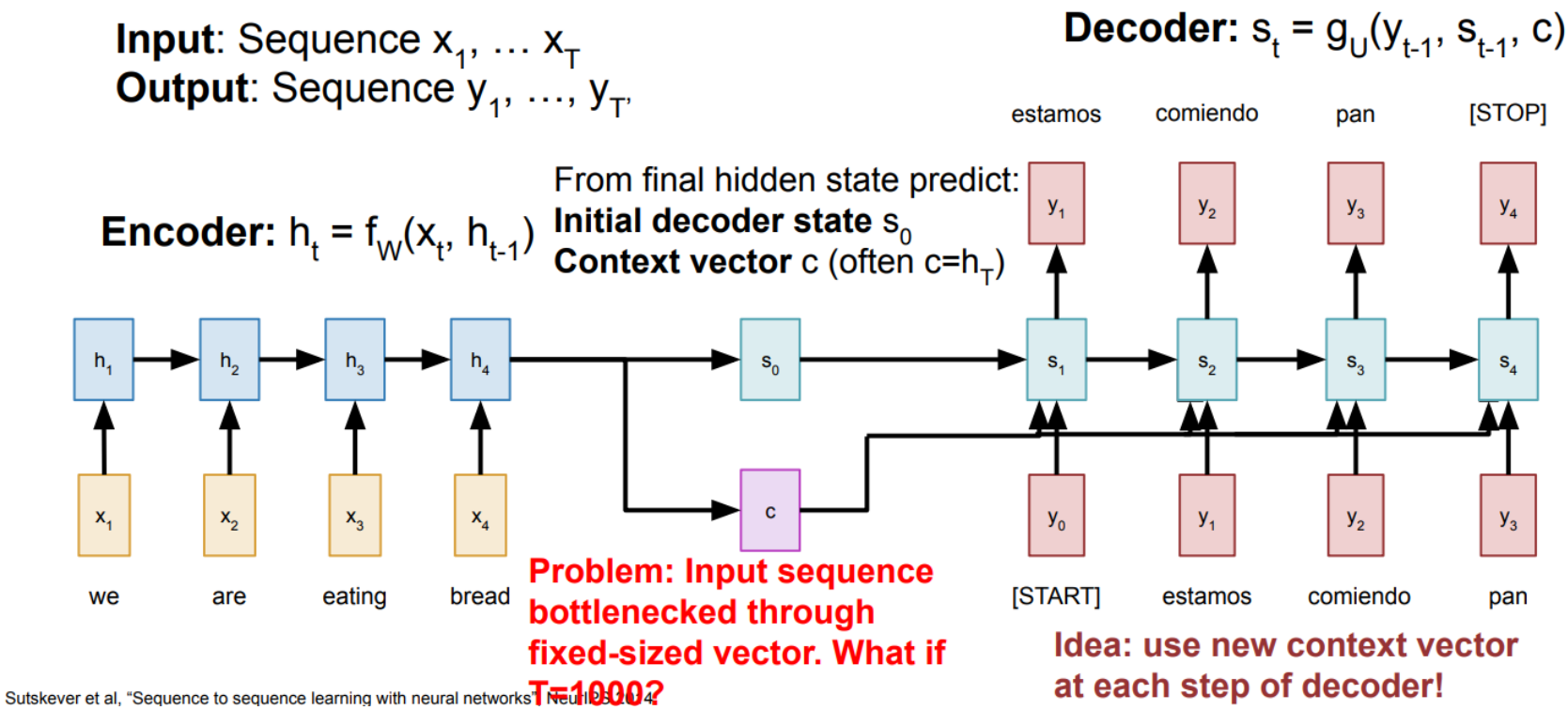
Sutskever et al, "Sequence to sequence learning with neural networks

Sequence to Sequence with RNNs



Sutskever et al, "Sequence to sequence learning with neural networks" NeurIPS 2014

Sequence to Sequence with RNNs

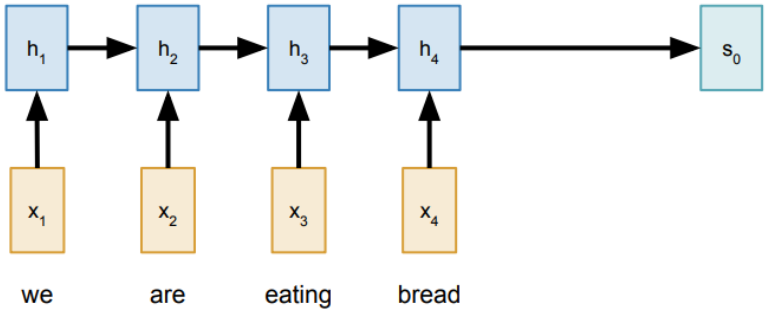


Sutskever et al, "Sequence to sequence learning with neural networks" NeurIPS 2014

Sequence to Sequence with RNNs and Attention

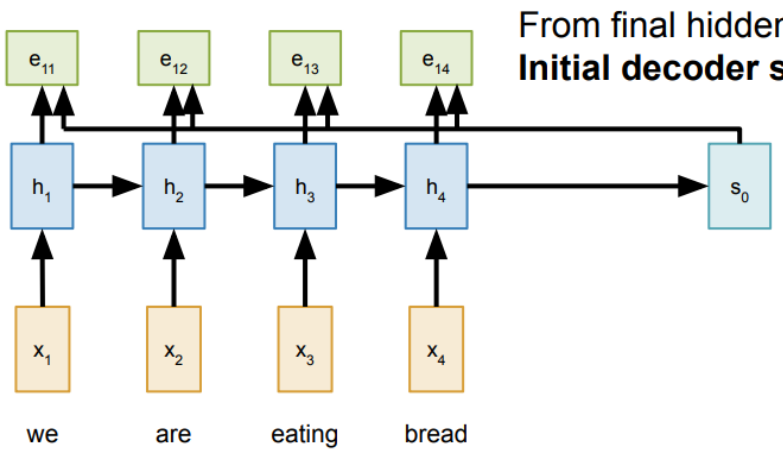
Input: Sequence $x_1, \ldots x_T$
Output: Sequence $y_1, \ldots, y_{T'}$

Encoder: $h_t = f_W(x_t, h_{t-1})$ From final hidden state: **Initial decoder state** s_0



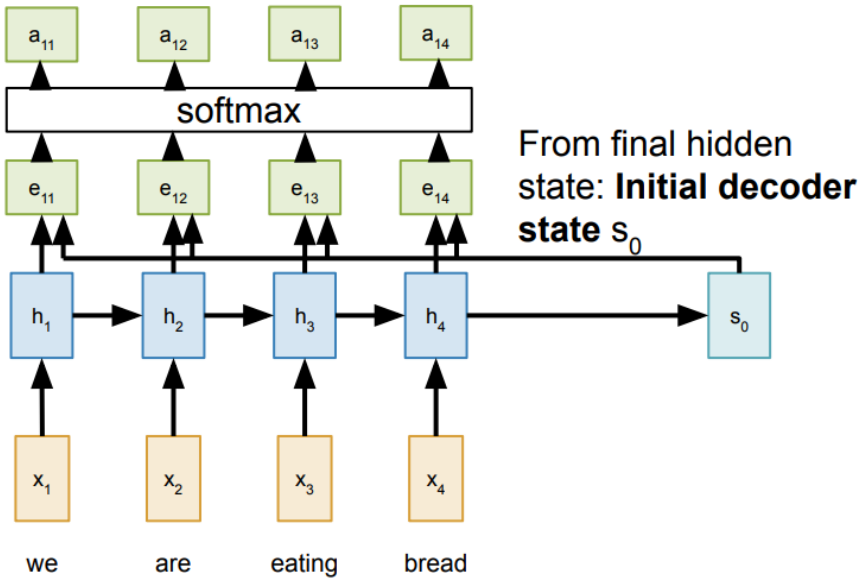
Sequence to Sequence with RNNs and Attention

Compute (scalar) **alignment scores**
$e_{t,i} = f_{att}(s_{t-1}, h_i)$ (f_{att} is an MLP)



From final hidden state:
Initial decoder state s_0

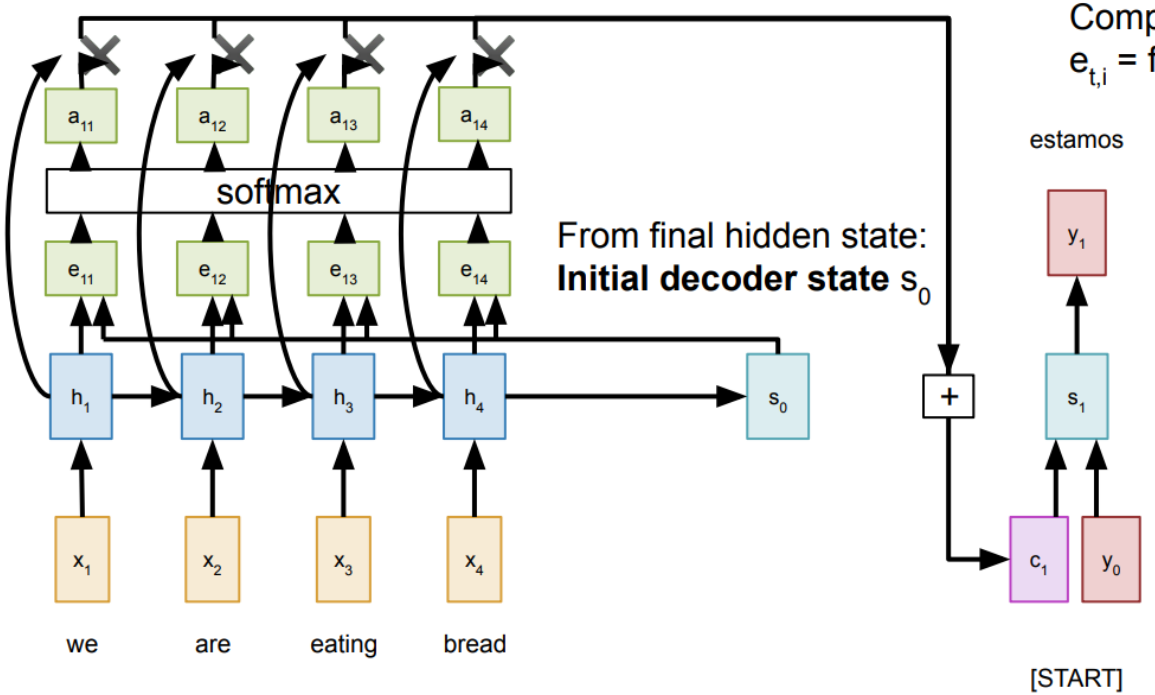
Sequence to Sequence with RNNs and Attention



Compute (scalar) **alignment scores**
$e_{t,i} = f_{att}(s_{t-1}, h_i)$ (f_{att} is an MLP)

Normalize alignment scores to get **attention weights**
$0 < a_{t,i} < 1 \quad \sum_i a_{t,i} = 1$

Sequence to Sequence with RNNs and Attention



Compute (scalar) **alignment scores**

$e_{t,i} = f_{att}(s_{t-1}, h_i)$ (f_{att} is an MLP)

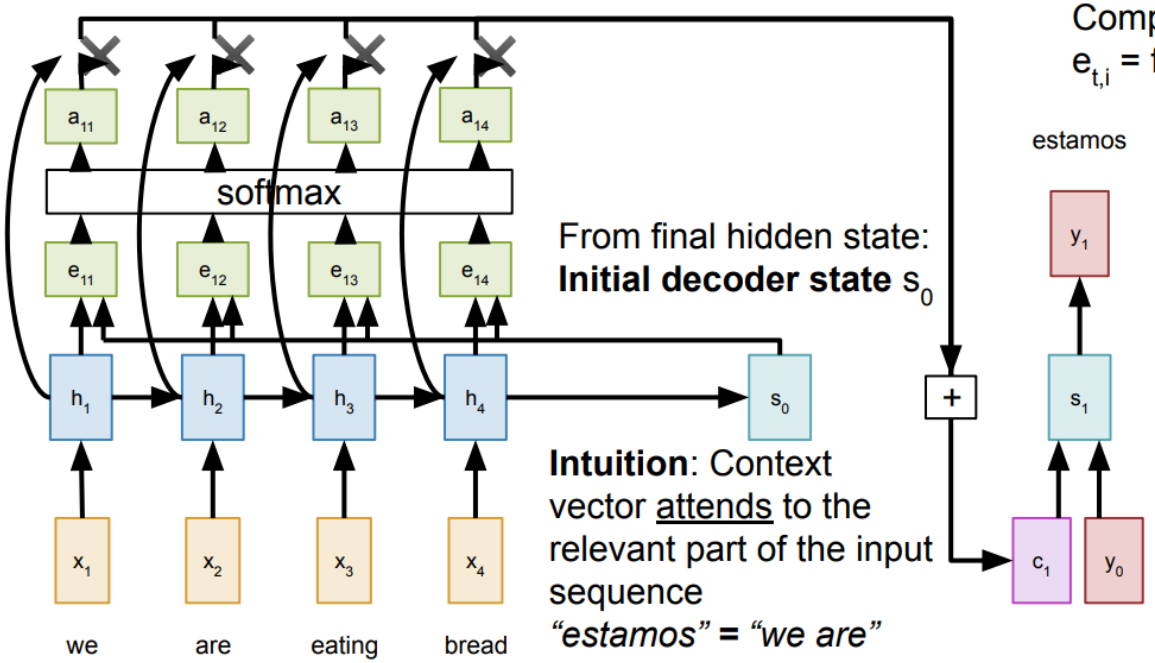
Normalize alignment scores to get **attention weights**

$0 < a_{t,i} < 1 \quad \sum_i a_{t,i} = 1$

Compute context vector as linear combination of hidden states

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

Sequence to Sequence with RNNs and Attention



Compute (scalar) **alignment scores**

$e_{t,i} = f_{att}(s_{t-1}, h_i)$ (f_{att} is an MLP)

Normalize alignment scores to get **attention weights**

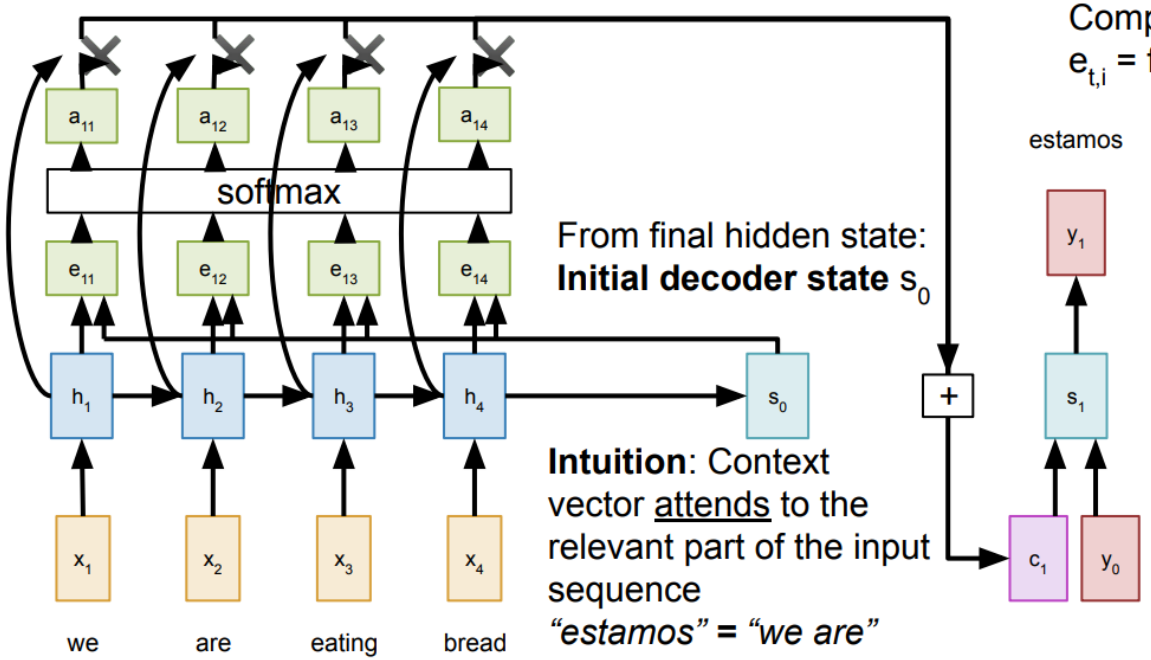
$0 < a_{t,i} < 1 \quad \sum_i a_{t,i} = 1$

Compute context vector as linear combination of hidden states

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

Use context vector in decoder: $s_t = g_U(y_{t-1}, s_{t-1}, c_t)$

Sequence to Sequence with RNNs and Attention



Compute (scalar) **alignment scores**
$e_{t,i} = f_{att}(s_{t-1}, h_i)$ (f_{att} is an MLP)

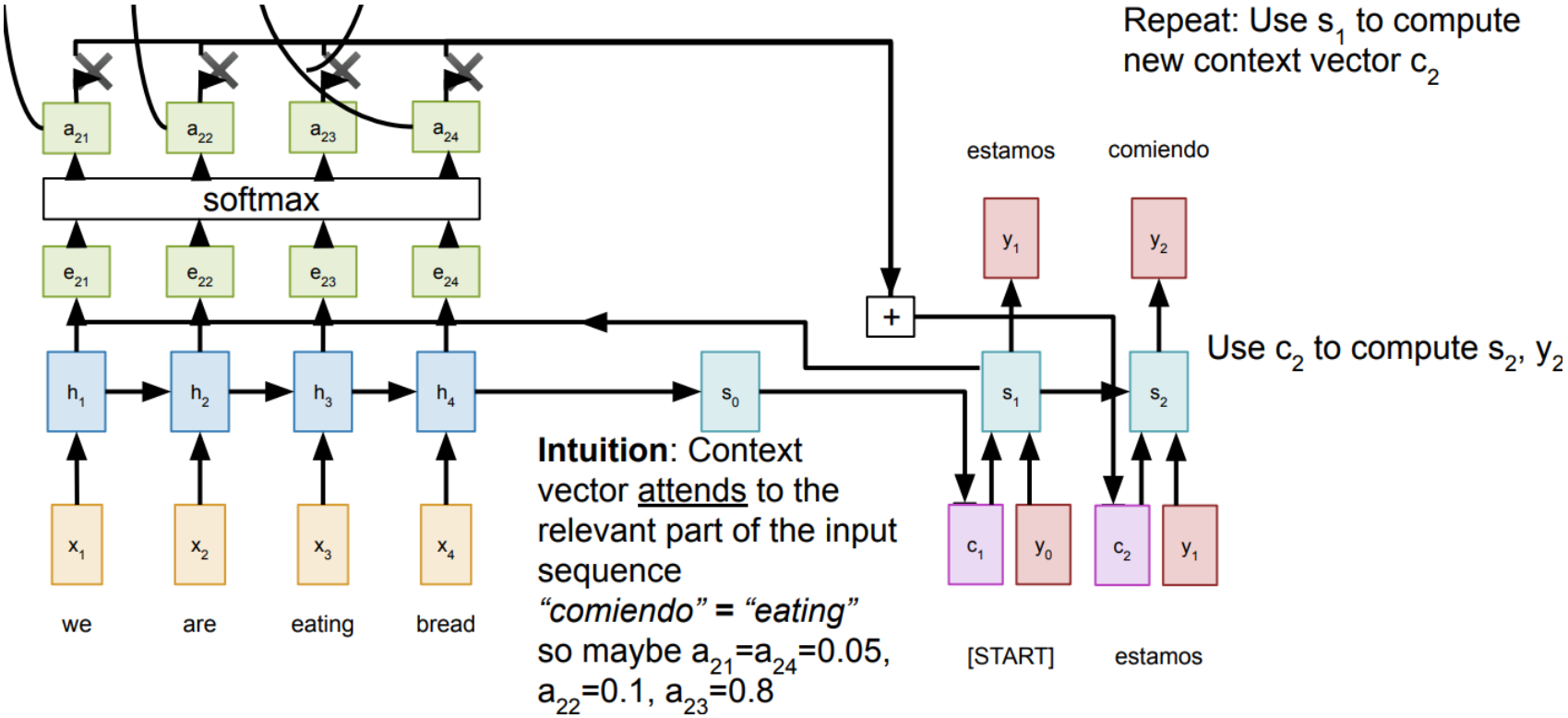
Normalize alignment scores to get **attention weights**
$0 < a_{t,i} < 1$ $\sum_i a_{t,i} = 1$

Compute context vector as linear combination of hidden states
$c_t = \sum_i a_{t,i} h_i$

Use context vector in decoder: $s_t = g_U(y_{t-1}, s_{t-1}, c_t)$

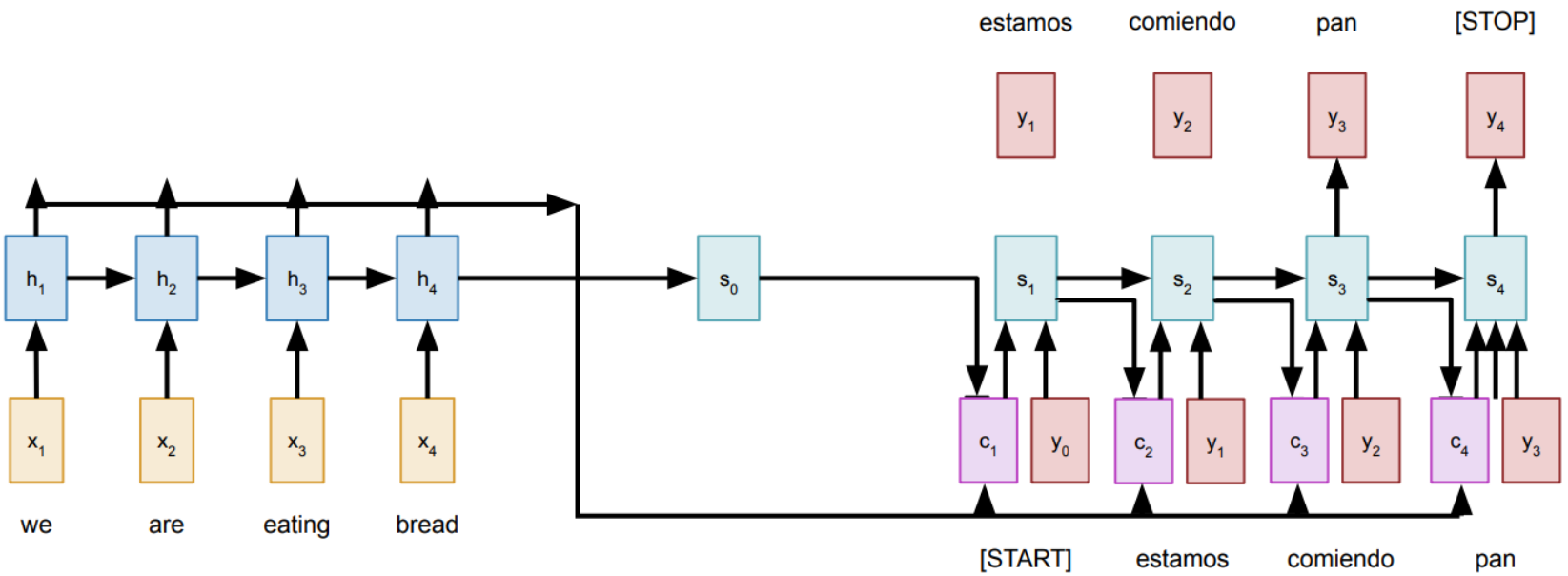
This is all differentiable! No supervision on attention weights – backprop through everything

Sequence to Sequence with RNNs and Attention



Sequence to Sequence with RNNs and Attention

- Use a different context vector in each timestep of decoder
- Input sequence not bottlenecked through single vector
- At each timestep of decoder, context vector "looks at" different parts of the input sequence



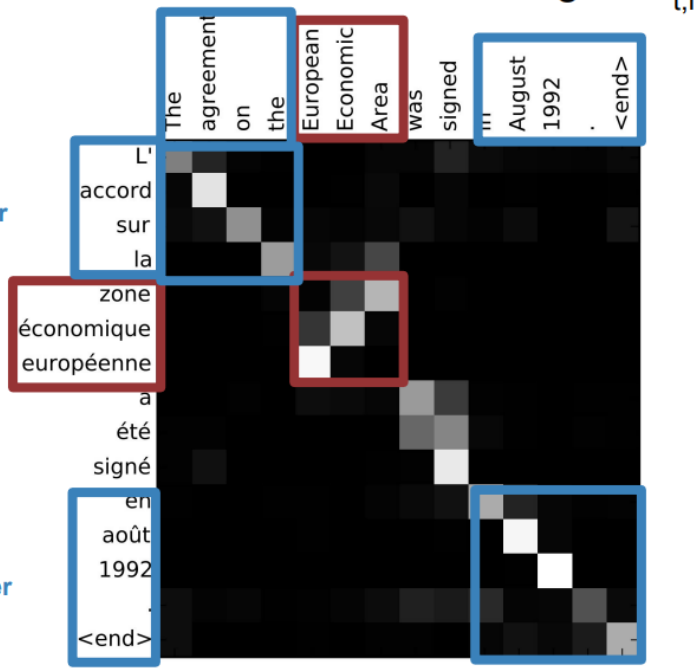
Sequence to Sequence with RNNs and Attention

Example: English to French translation

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

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

Visualize attention weights $a_{t,i}$



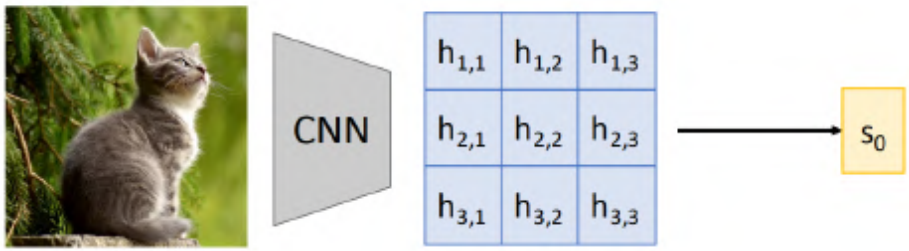
Diagonal attention means words correspond in order

Attention figures out different word orders

Diagonal attention means words correspond in order

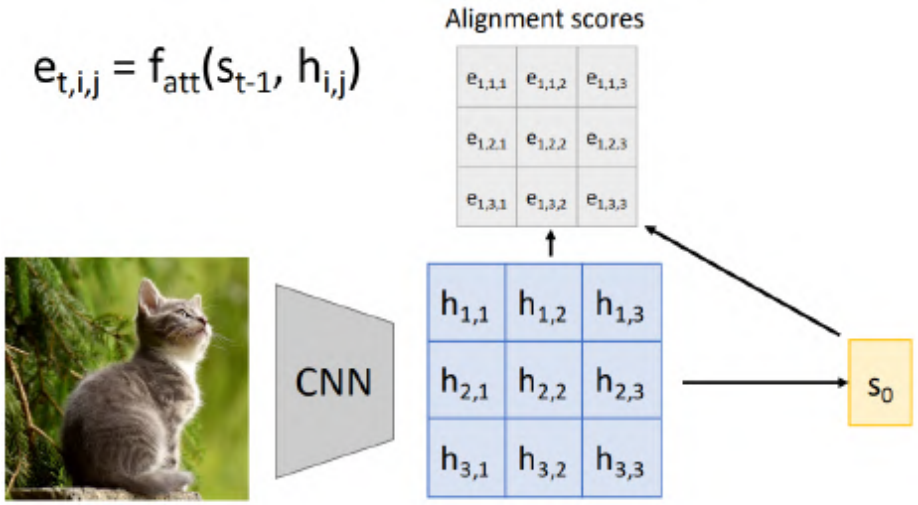
Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015

Image Captioning with RNNs and Attention



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

Image Captioning with RNNs and Attention



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

Image Captioning with RNNs and Attention

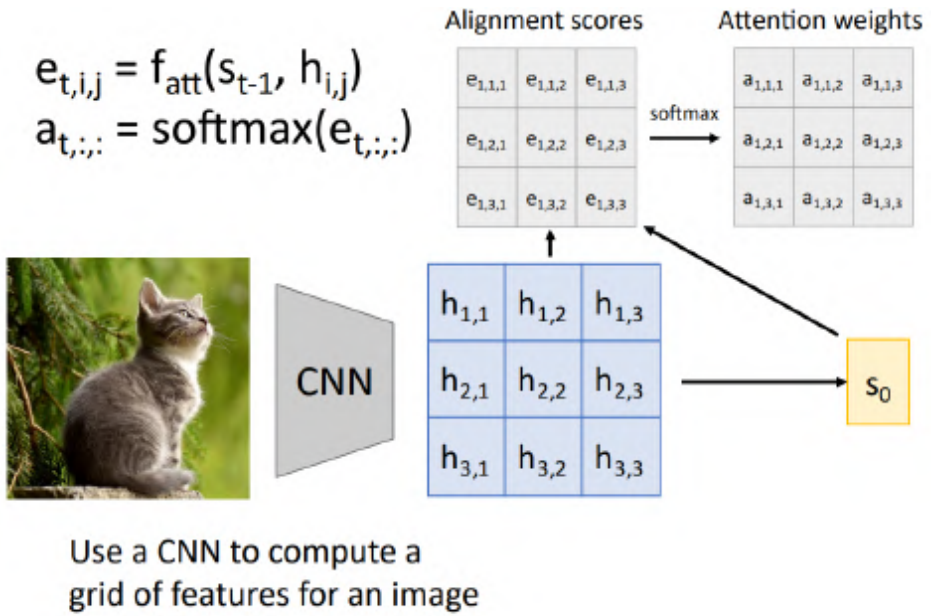


Image Captioning with RNNs and Attention

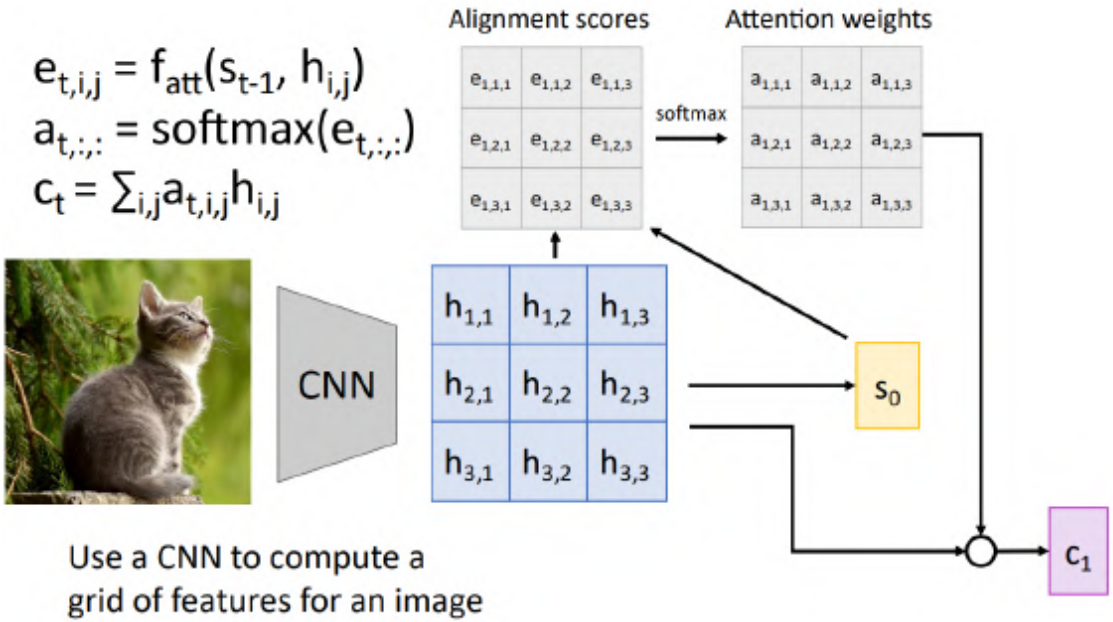


Image Captioning with RNNs and Attention

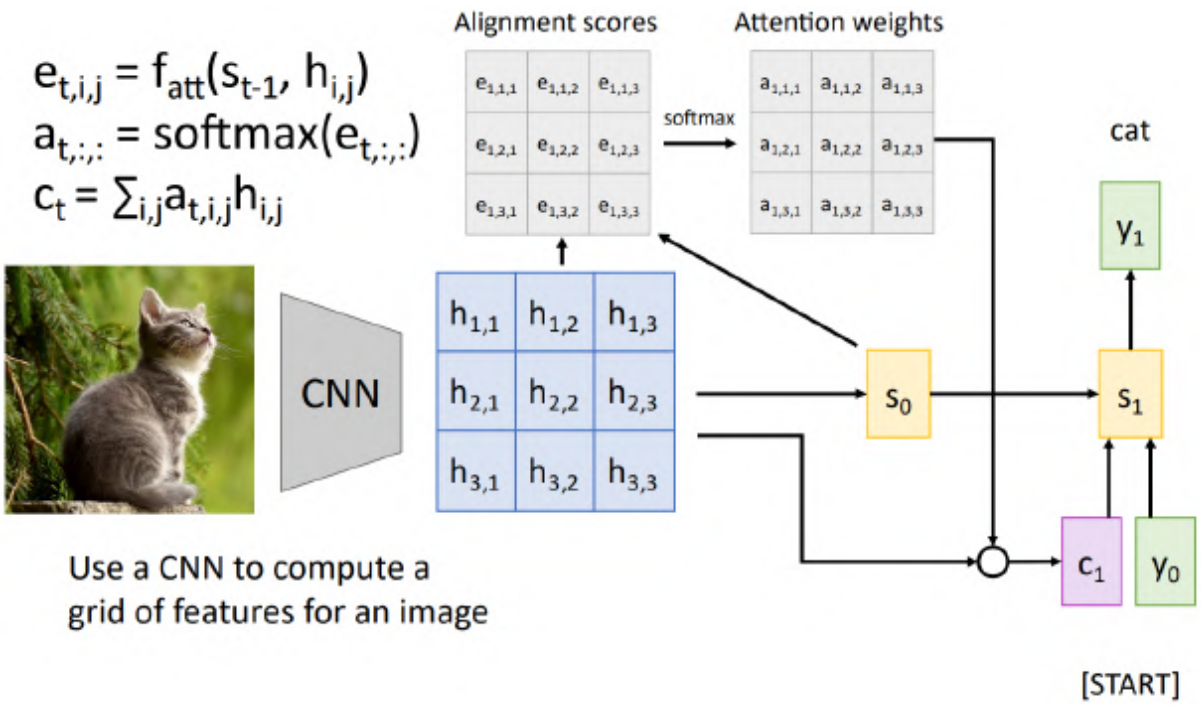


Image Captioning with RNNs and Attention

$e_{t,i,j} = f_{att}(s_{t-1}, h_{i,j})$
$a_{t,:,:} = \text{softmax}(e_{t,:,:})$
$c_t = \sum_{i,j} a_{t,i,j} h_{i,j}$

CNN

$h_{1,1}$ $h_{1,2}$ $h_{1,3}$
$h_{2,1}$ $h_{2,2}$ $h_{2,3}$
$h_{3,1}$ $h_{3,2}$ $h_{3,3}$

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

s_0 s_1 c_1 y_0 y_1

cat

[START]

Image Captioning with RNNs and Attention

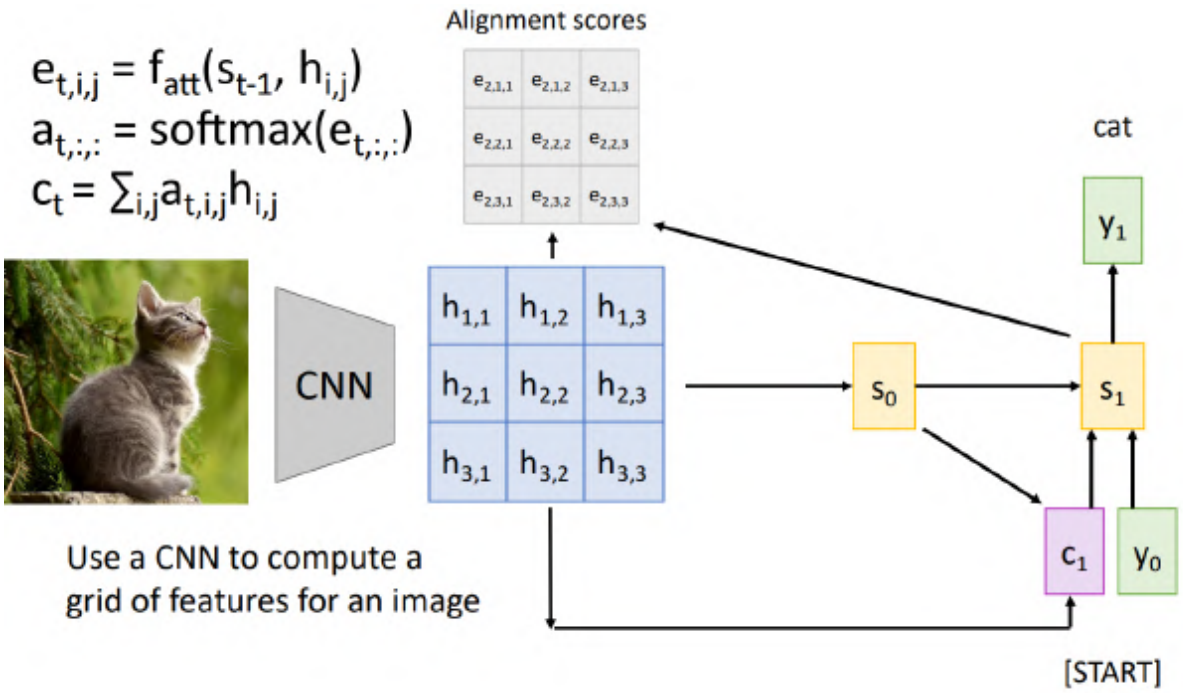


Image Captioning with RNNs and Attention

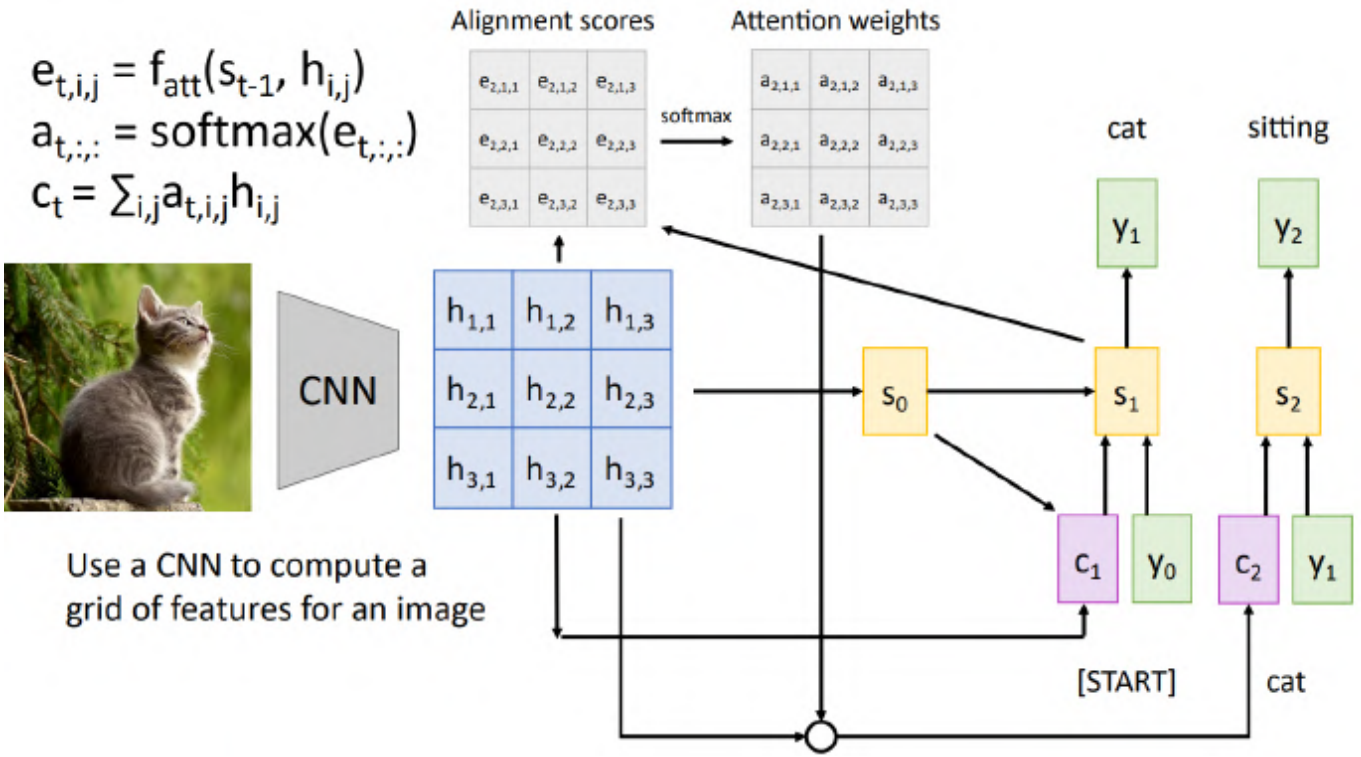


Image Captioning with RNNs and Attention

$e_{t,i,j} = f_{att}(s_{t-1}, h_{i,j})$
$a_{t,:,:} = \text{softmax}(e_{t,:,:})$
$c_t = \sum_{i,j} a_{t,i,j} h_{i,j}$

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

CNN

$h_{1,1}$ $h_{1,2}$ $h_{1,3}$
$h_{2,1}$ $h_{2,2}$ $h_{2,3}$
$h_{3,1}$ $h_{3,2}$ $h_{3,3}$

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

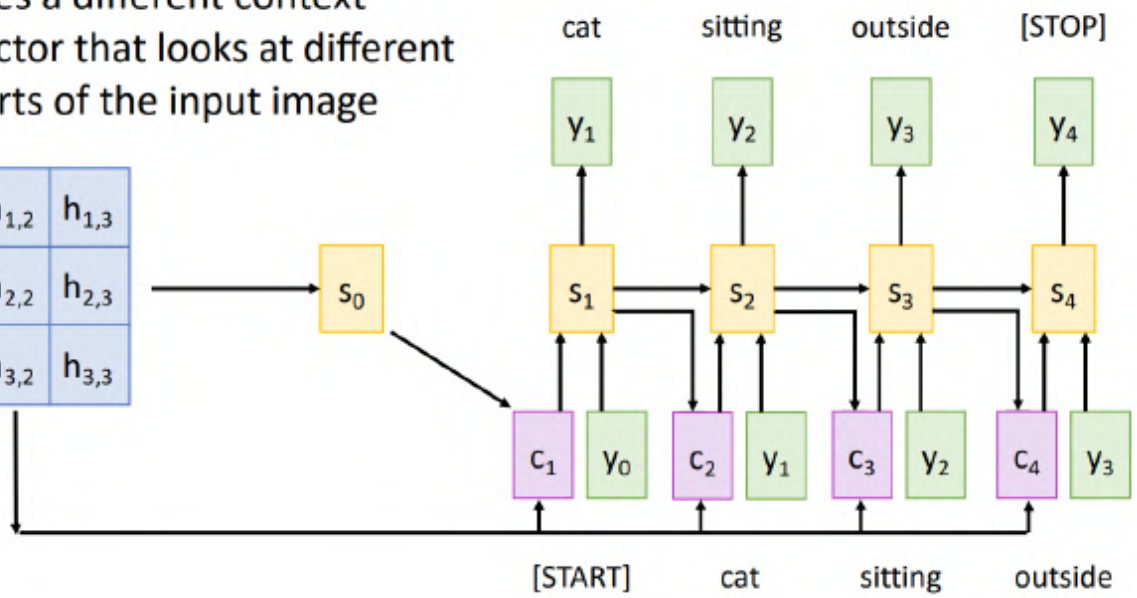


Image Captioning with RNNs and Attention

A dog is standing on a hardwood floor.

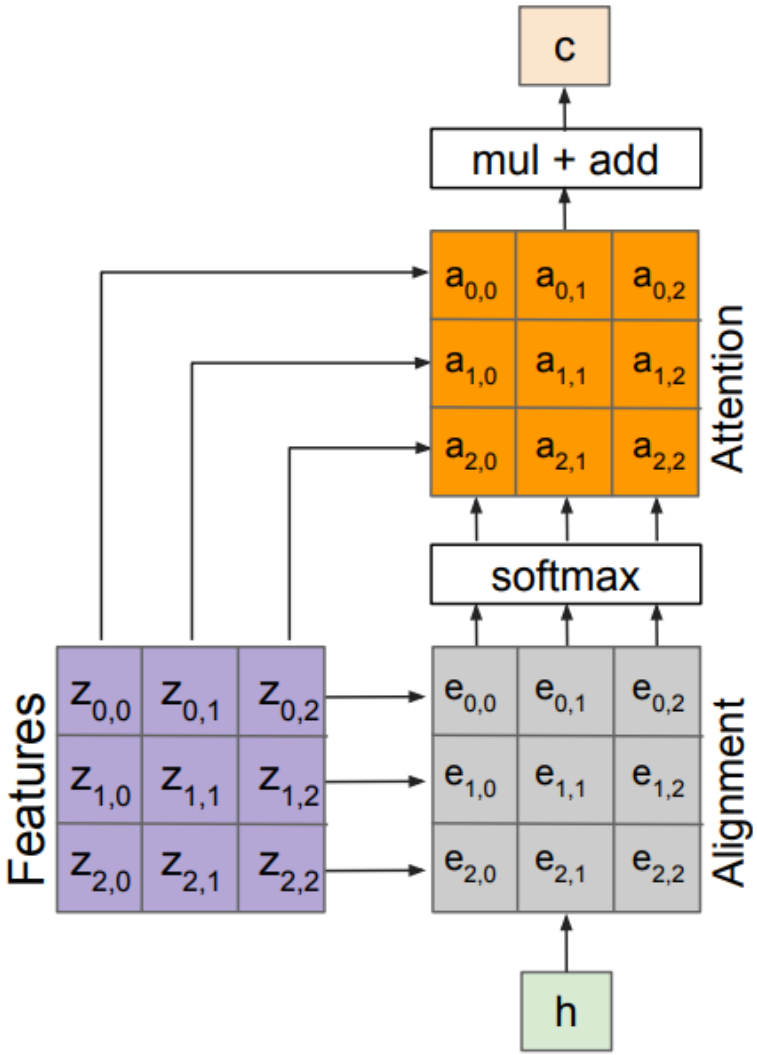
A stop sign is on a road with a mountain in the background.

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

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

[0]Xu et al, "Show, AKend, and Tell: Neural Image CapAon GeneraAon with Visual AKenAon", ICML 2015

Attention we just saw in image captioning

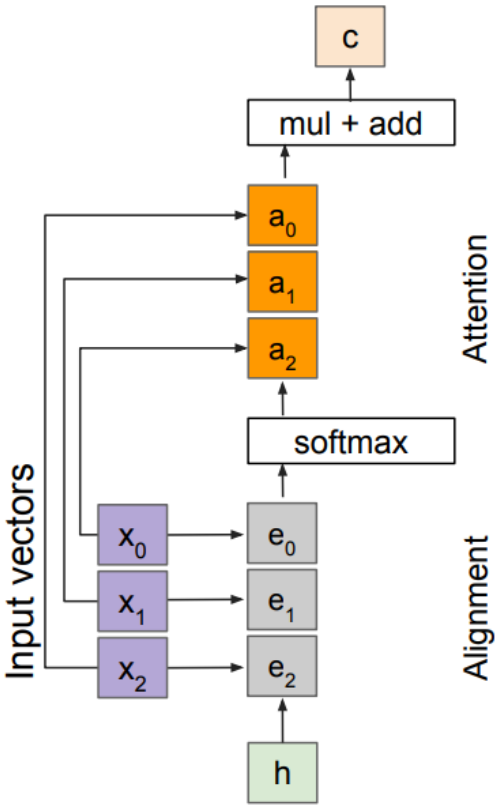


Outputs:
context vector: **c** (shape: D)

Operations:
Alignment: $e_{i,j} = f_{att}(h, z_{i,j})$
Attention: **a** = softmax(**e**)
Output: $\mathbf{c} = \sum_{i,j} a_{i,j} z_{i,j}$

Inputs:
Features: **z** (shape: H x W x D)
Query: **h** (shape: D)

General Attention Layer



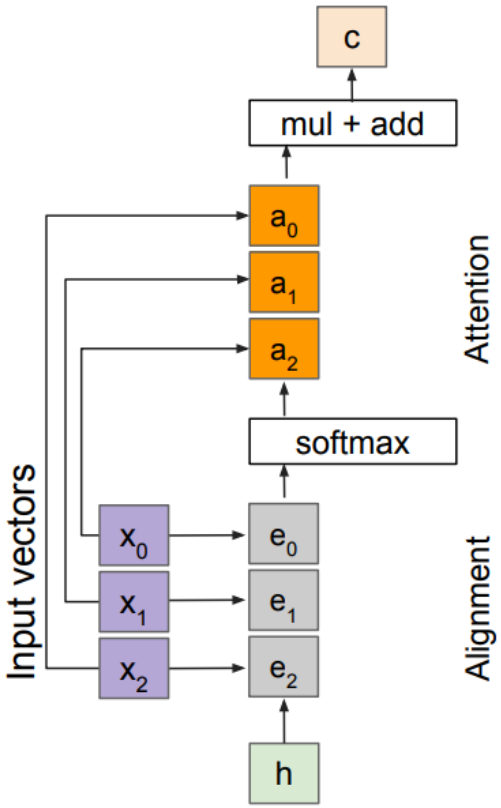
Outputs:
context vector: **c** (shape: D)

Operations:
Alignment: $e_i = f_{att}(h, x_i)$
Attention: **a** = softmax(**e**)
Output: $\mathbf{c} = \sum_i a_i x_i$

Inputs:
Input vectors: **x** (shape: N x D)
Query: **h** (shape: D)

- Attention operation is permutation invariant
- Doesn't care about ordering of the features
- Stretch H x W = N into N vectors

General Attention Layer



Outputs:
context vector: **c** (shape: D)

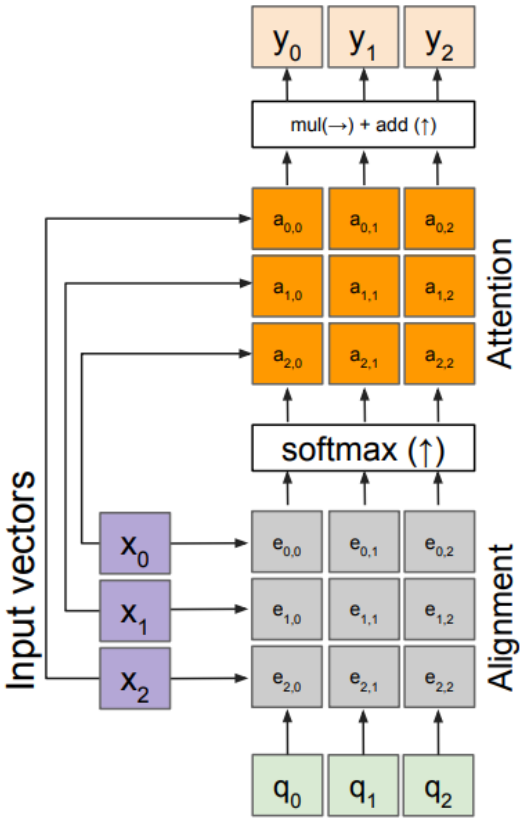
Operations:
Alignment: $e_i = h \cdot x_i / \sqrt{D}$
Attention: **a** = softmax(**e**)
Output: $\mathbf{c} = \sum_i a_i x_i$

Inputs:
Input vectors: **x** (shape: N x D)
Query: **h** (shape: D)

Change $f_{att}(.)$ to a scaled simple dot product

- Larger dimensions means more terms in the dot product sum.
- So, the variance of the logits is higher. Large magnitude vectors will produce much higher logits.
- So, the post-softmax distribution has lower-entropy, assuming logits are IID.
- Ultimately, these large magnitude vectors will cause softmax to peak and assign very little weight to all others
- Divide by $\sqrt{D}$ to reduce effect of large magnitude vectors

General Attention Layer



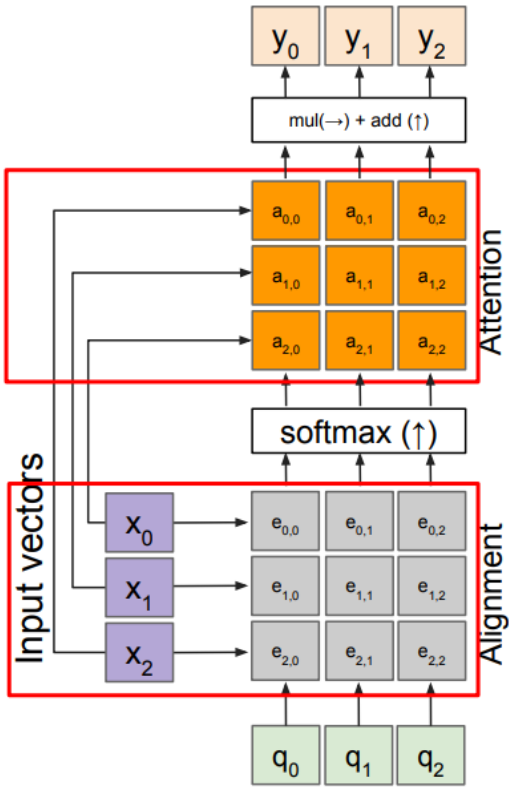
Outputs:
context vectors: **y** (shape: D)

Operations:
Alignment: $e_{i,j} = q_j \cdot x_i / \sqrt{D}$
Attention: $\mathbf{a} = \text{softmax}(\mathbf{e})$
Output: $y_j = \sum_i a_{i,j} x_i$

Inputs:
Input vectors: **x** (shape: N x D)
Queries: **q** (shape: M x D)

- We can have multiple Query Vectors.
- Each query creates a new output context vector

General Attention Layer



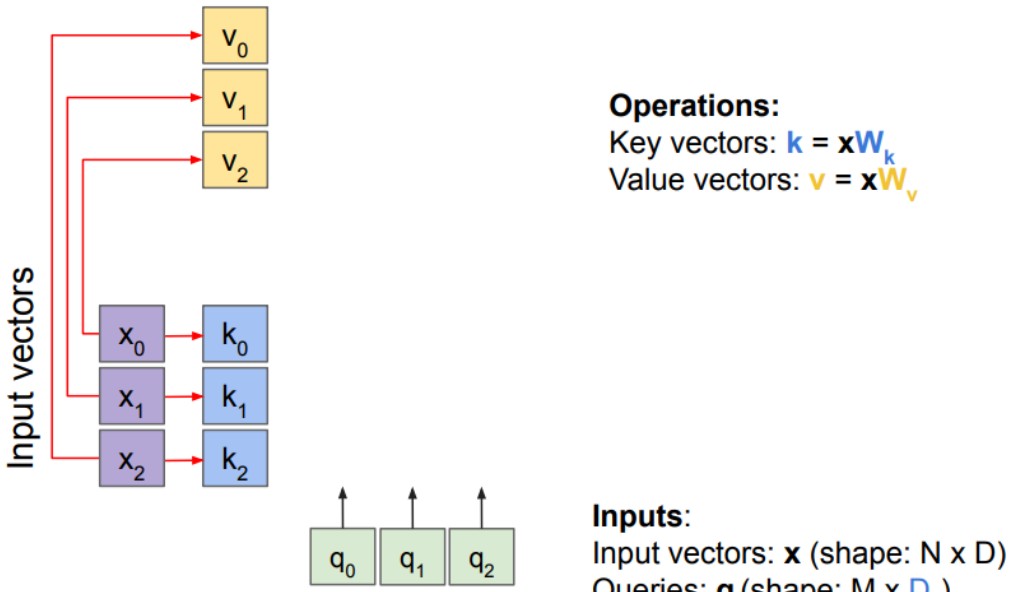
Outputs:
context vectors: **y** (shape: D)

Operations:
Alignment: $e_{i,j} = q_j \cdot x_i / \sqrt{D}$
Attention: **a** = softmax(**e**)
Output: $y_j = \sum_i a_{i,j} x_i$

Inputs:
Input vectors: **x** (shape: N x D)
Queries: **q** (shape: M x D)

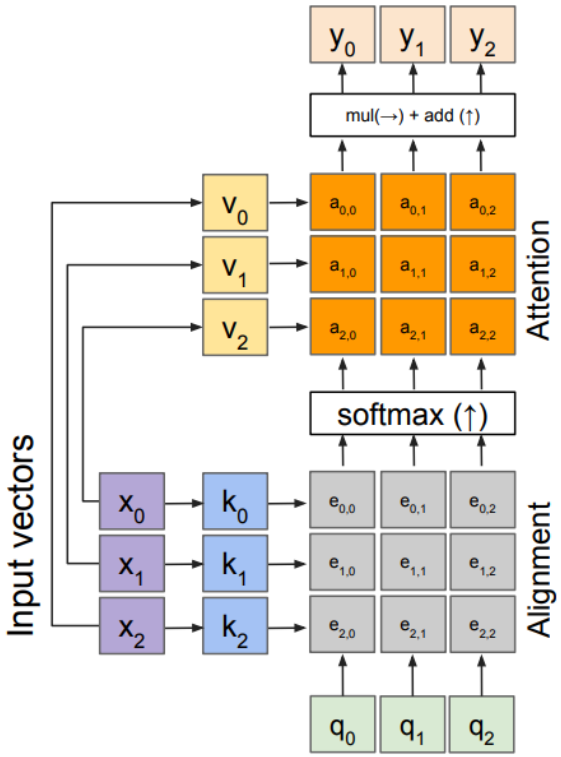
- Notice that the input vectors are used for both the alignment as well as the attention calculations.

General Attention Layer



- Notice that the input vectors are used for both the alignment as well as the attention calculations.
- We can add more expressivity to the layer by adding a different FC layer before each of the two steps.

General Attention Layer



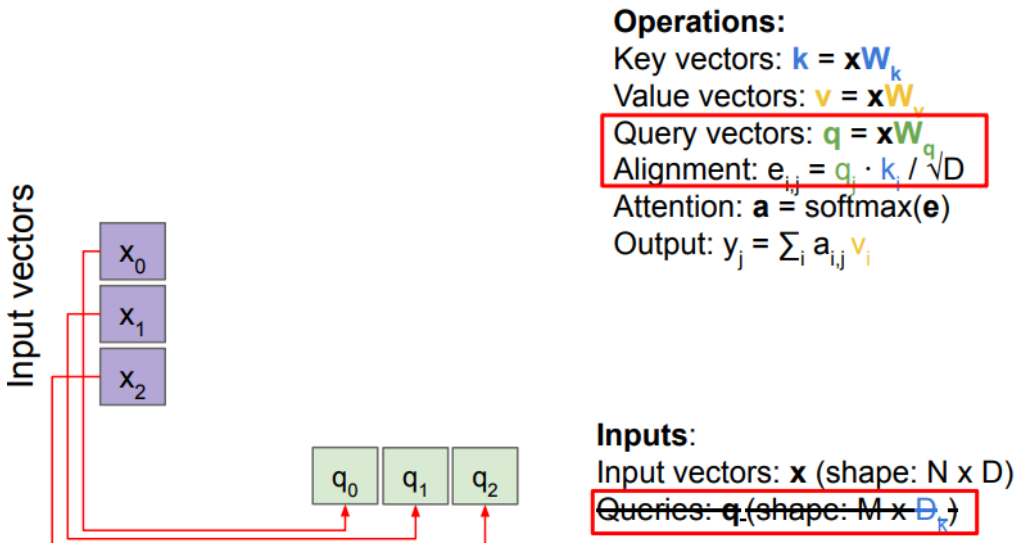
Outputs:
context vectors: **y** (shape: D_v)

Operations:
Key vectors: $k = xW_k$
Value vectors: $v = xW_v$
Alignment: $e_{i,j} = q_j \cdot k_i / \sqrt{D}$
Attention: $a = \text{softmax}(e)$
Output: $y_j = \sum_i a_{i,j} v_i$

Inputs:
Input vectors: **x** (shape: N x D)
Queries: **q** (shape: M x D_k)

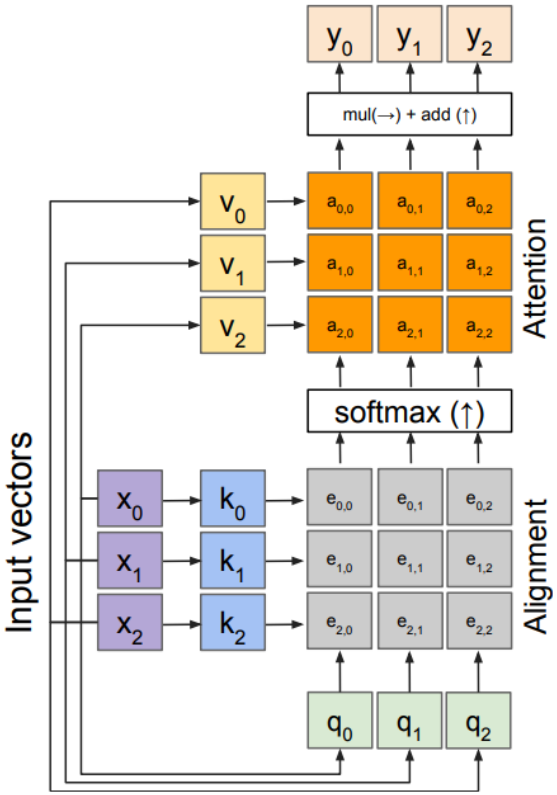
- Notice that the input vectors are used for both the alignment as well as the attention calculations.
- We can add more expressivity to the layer by adding a different FC layer before each of the two steps.
- The input and output dimensions can now change depending on the key and value FC layers

Self Attention Layer



- Recall that the query vector was a function of the input vectors
- We can calculate the query vectors from the input vectors, therefore, defining a "self-attention" layer.
- No input query vectors anymore
- Instead, query vectors are calculated using a FC layer.

Self Attention Layer



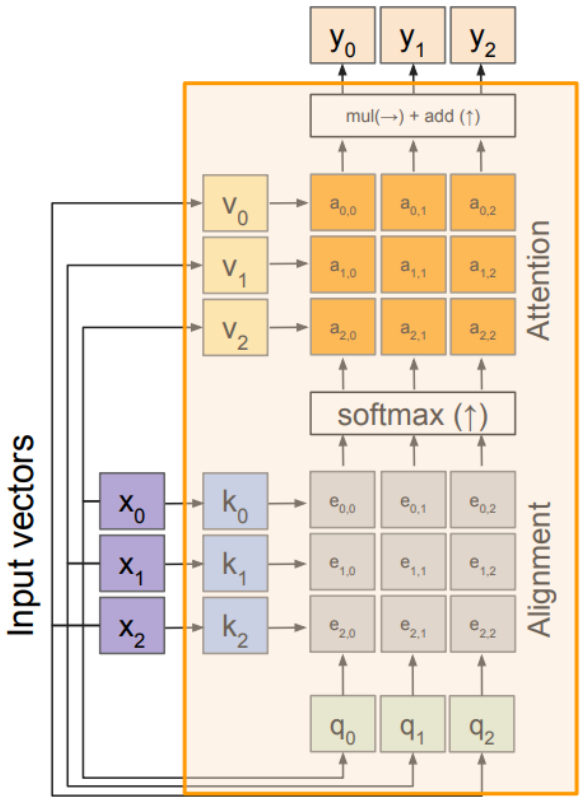
Outputs:
context vectors: **y** (shape: D_v)

Operations:
Key vectors: $\mathbf{k} = \mathbf{x}W_k$
Value vectors: $\mathbf{v} = \mathbf{x}W_v$
Query vectors: $\mathbf{q} = \mathbf{x}W_q$
Alignment: $e_{i,j} = q_j \cdot k_i / \sqrt{D}$
Attention: $\mathbf{a} = \text{softmax}(\mathbf{e})$
Output: $y_j = \sum_i a_{i,j} v_i$

Inputs:
Input vectors: **x** (shape: N x D)

- Recall that the query vector was a function of the input vectors
- We can calculate the query vectors from the input vectors, therefore, defining a "self-attention" layer.
- No input query vectors anymore
- Instead, query vectors are calculated using a FC layer.

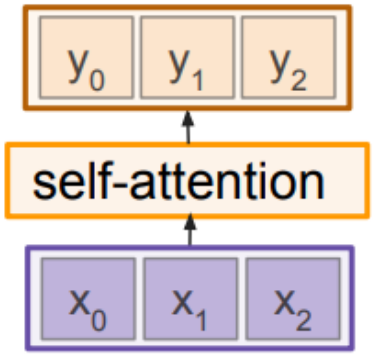
Self Attention Layer



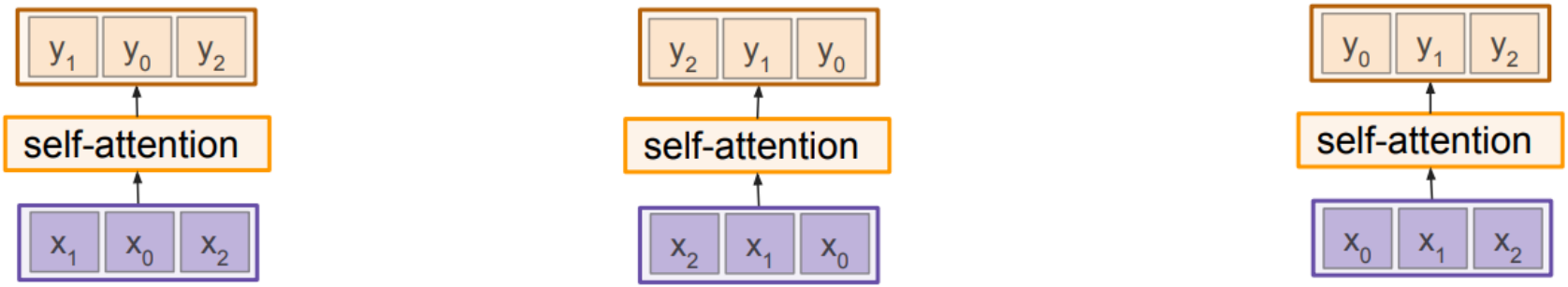
Outputs:
context vectors: **y** (shape: D_v)

Operations:
Key vectors: $\mathbf{k} = \mathbf{x}W_k$
Value vectors: $\mathbf{v} = \mathbf{x}W_v$
Query vectors: $\mathbf{q} = \mathbf{x}W_q$
Alignment: $e_{i,j} = q_j \cdot k_i / \sqrt{D}$
Attention: $\mathbf{a} = \text{softmax}(\mathbf{e})$
Output: $y_j = \sum_i a_{i,j} v_i$

Inputs:
Input vectors: **x** (shape: N x D)



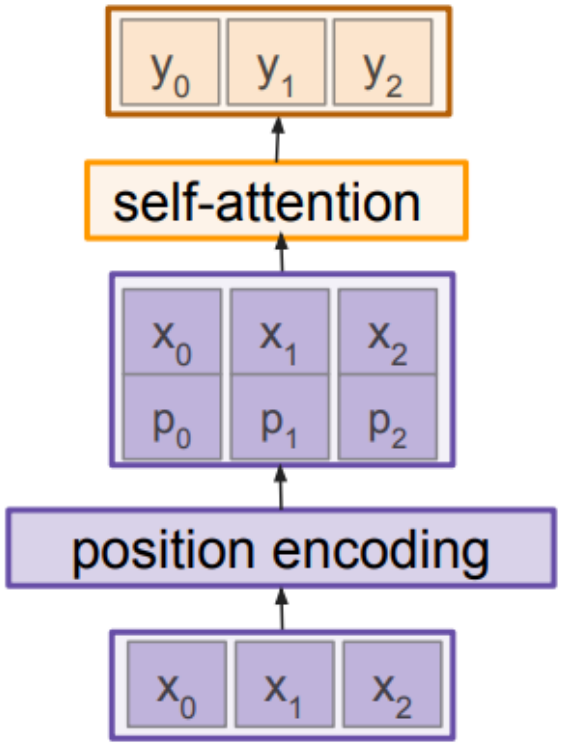
Permutation Invariance



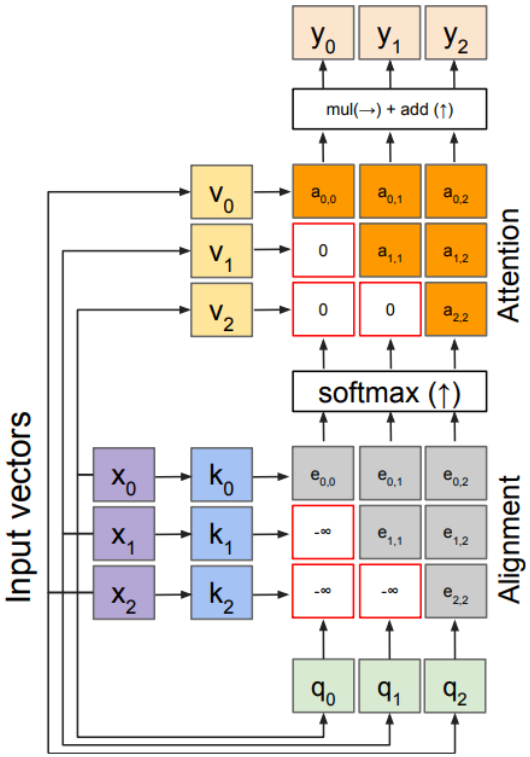
- Self-Attention Layer is Permutation equivariant
- It doesn't care about the orders of the inputs!
- **Problem:** How can we encode ordered sequences like language or spatially ordered image features?

Positional encoding

- Concatenate/add special positional encoding p_j to each input vector x_j
- We use a function pos: $N \rightarrow \mathbb{R}^D$ to process the position j of the vector into a d-dimensional vector
- So, $p_j = pos(j)$



Masked self-attention layer



Outputs:
context vectors: **y** (shape: D_v)

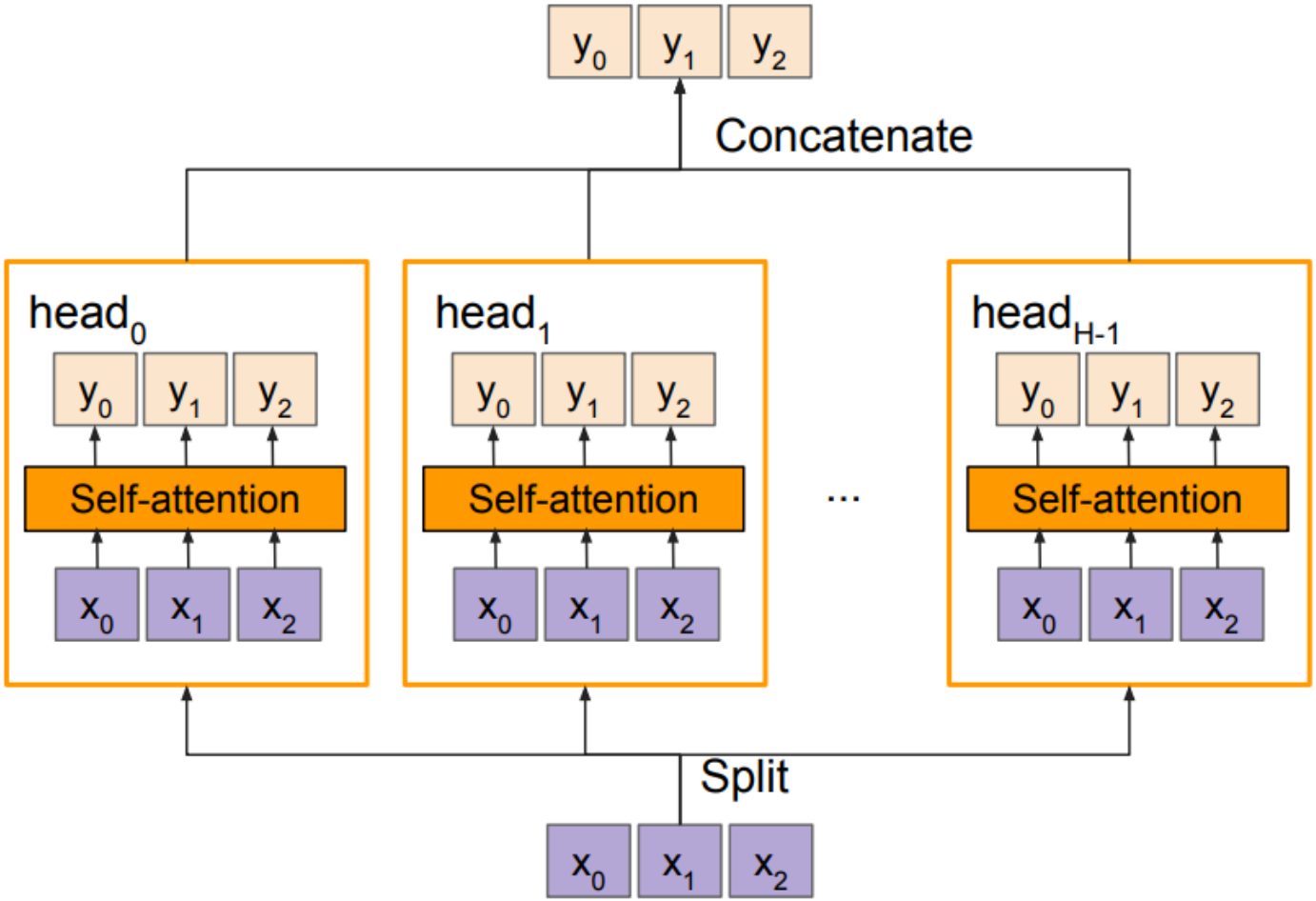
Operations:
Key vectors: $\mathbf{k} = \mathbf{x}W_k$
Value vectors: $\mathbf{v} = \mathbf{x}W_v$
Query vectors: $\mathbf{q} = \mathbf{x}W_q$
Alignment: $e_{i,j} = q_j \cdot k_i / \sqrt{D}$
Attention: $\mathbf{a} = \text{softmax}(\mathbf{e})$
Output: $y_j = \sum_i a_{i,j} v_i$

Inputs:
Input vectors: **x** (shape: N x D)

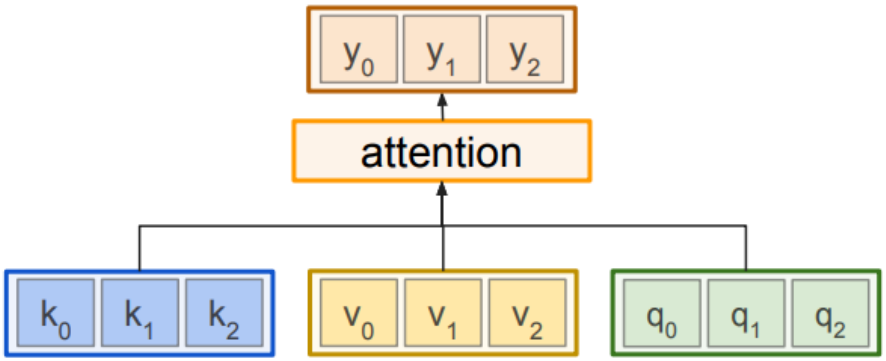
- Prevent vectors from looking at future vectors.
- Manually set alignment scores to $-\infty$

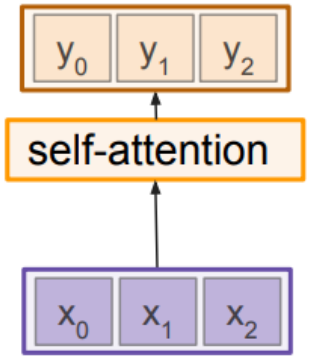
Multi-head self-attention layer

- Multiple self-attention heads in parallel

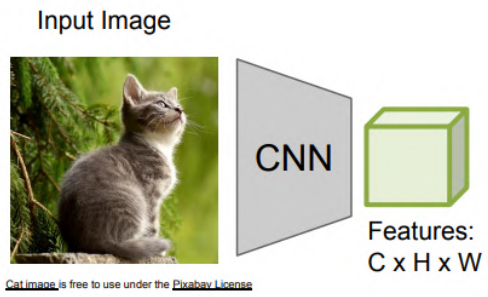


General attention versus self-attention

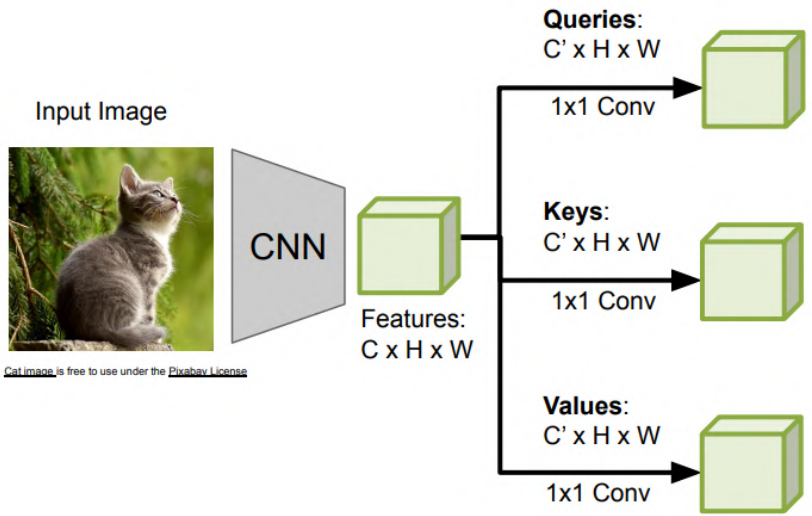




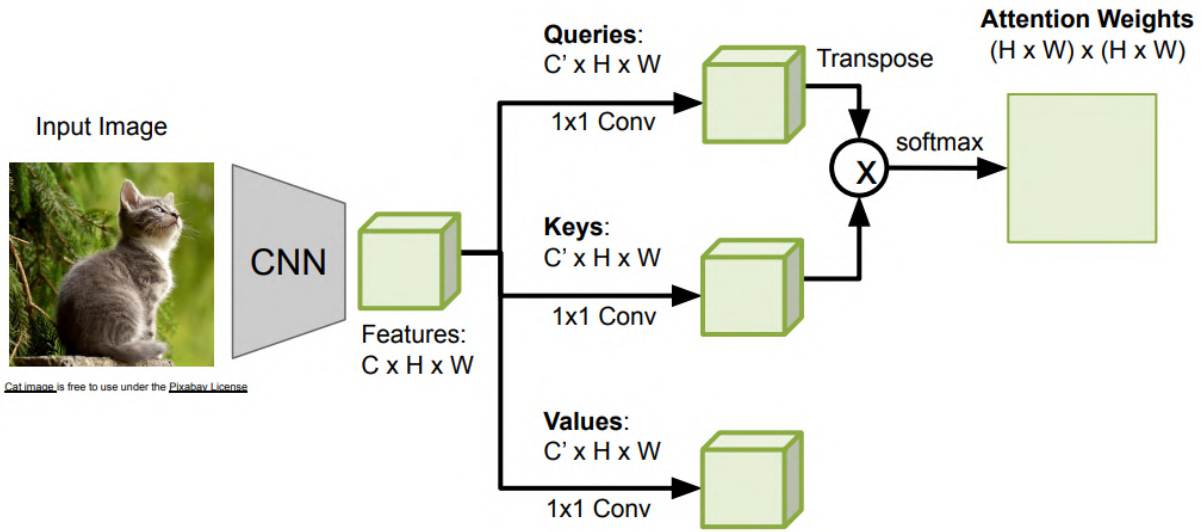
Example: CNN with Self-Attention



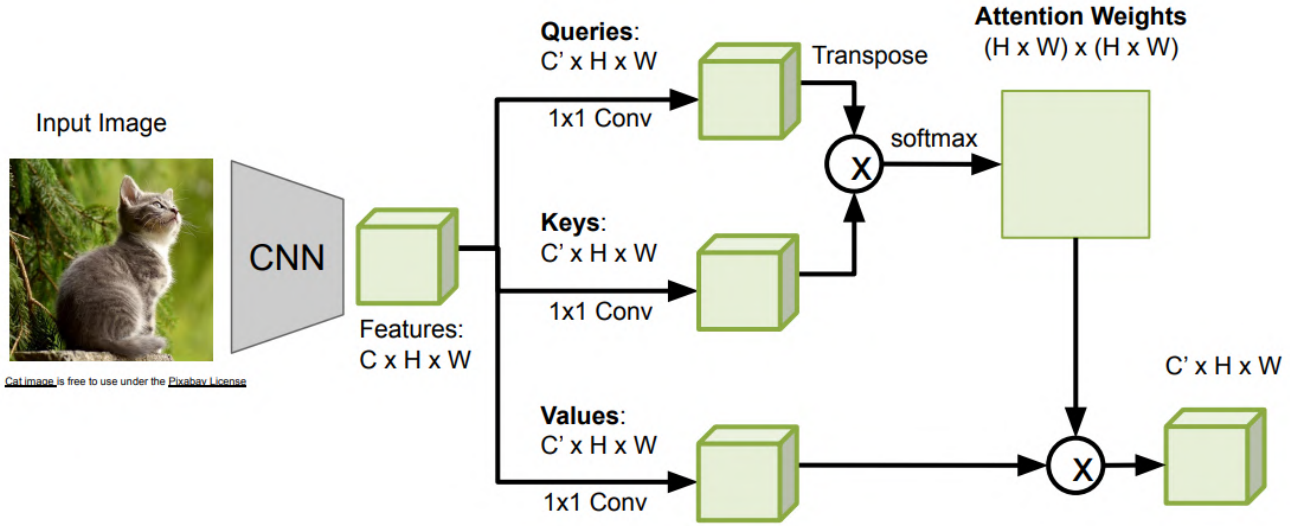
Example: CNN with Self-Attention



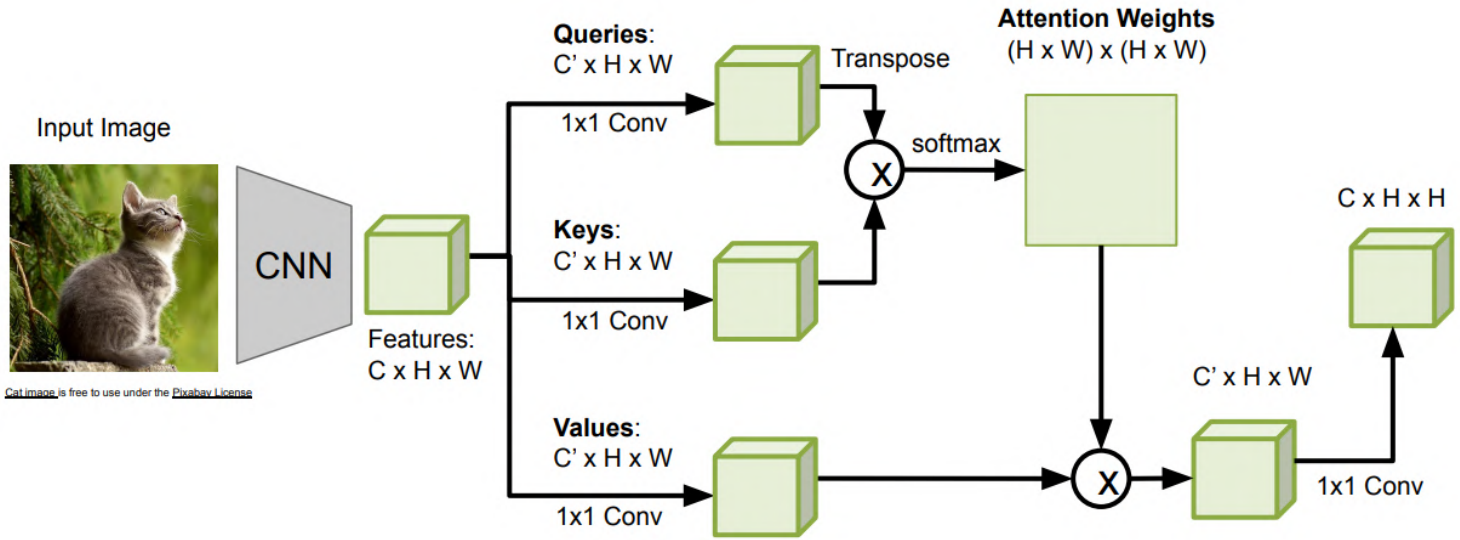
Example: CNN with Self-Attention



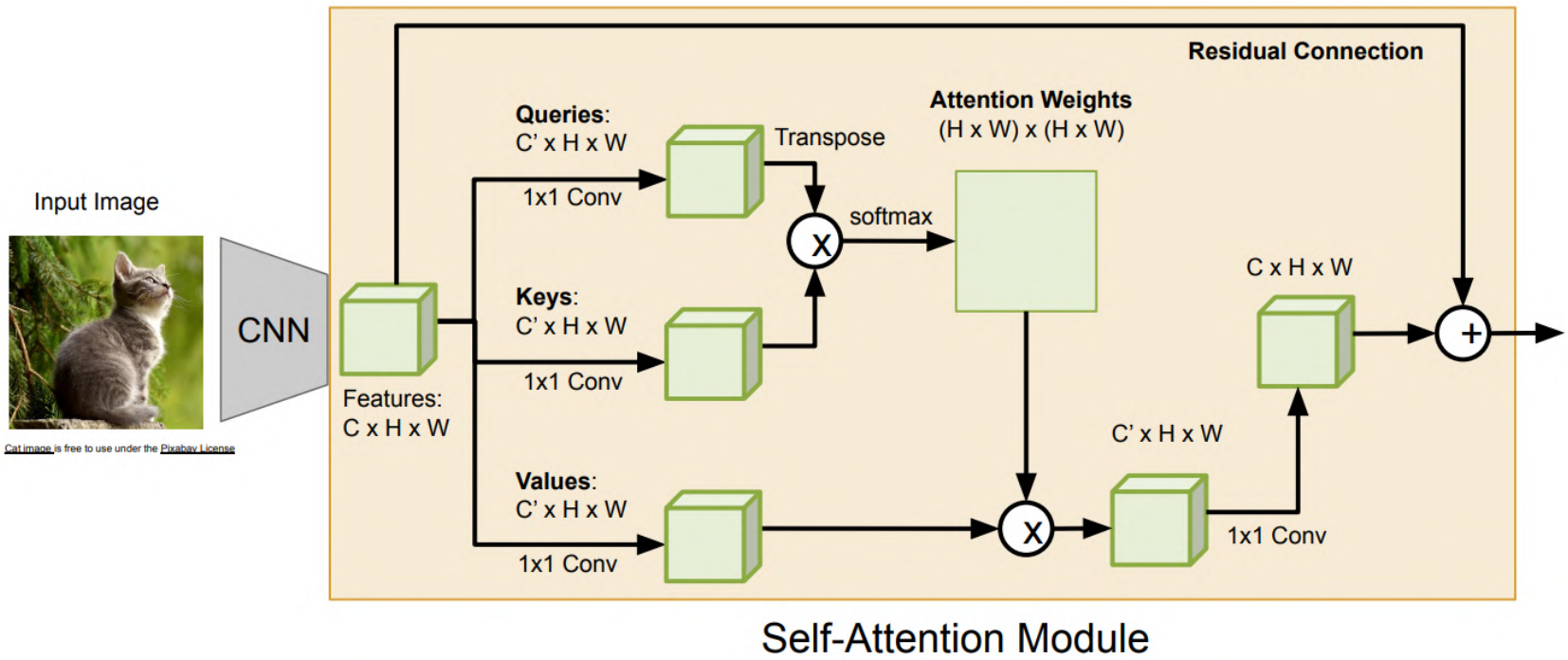
Example: CNN with Self-Attention



Example: CNN with Self-Attention

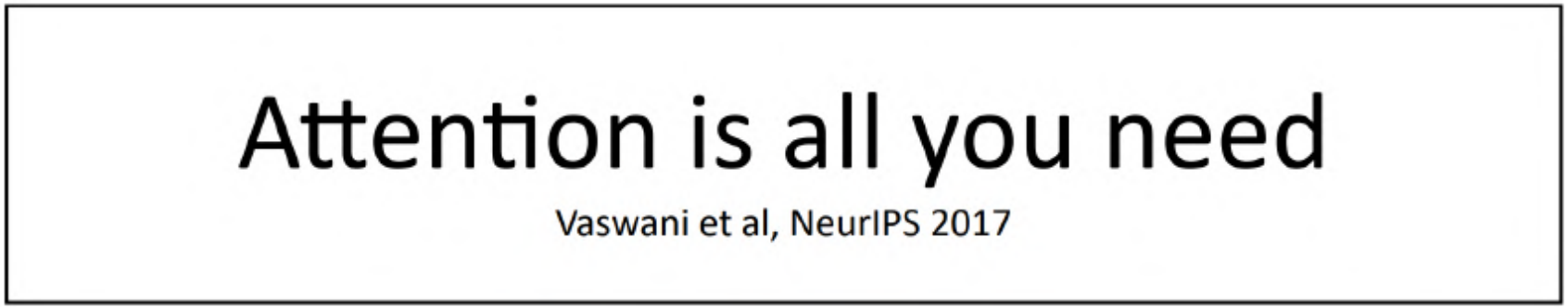


Example: CNN with Self-Attention



Cat image is free to use under the Pixabay License

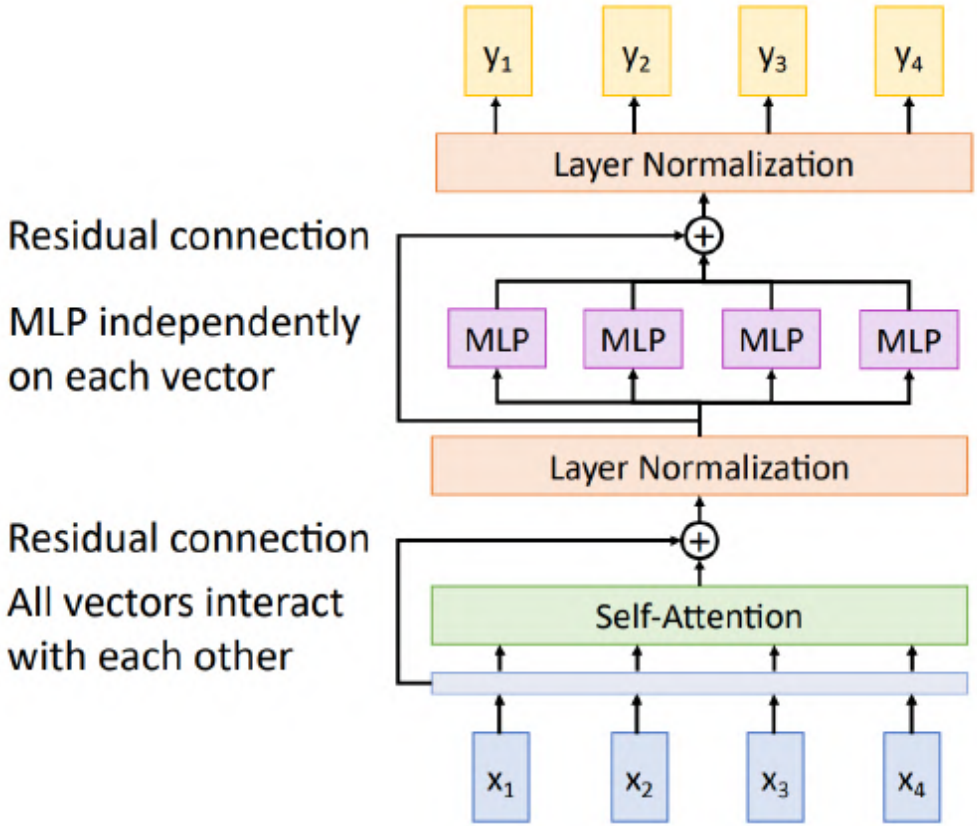
Self-Attention Module


Attention is all you need
Vaswani et al, NeurIPS 2017

Transformer Block

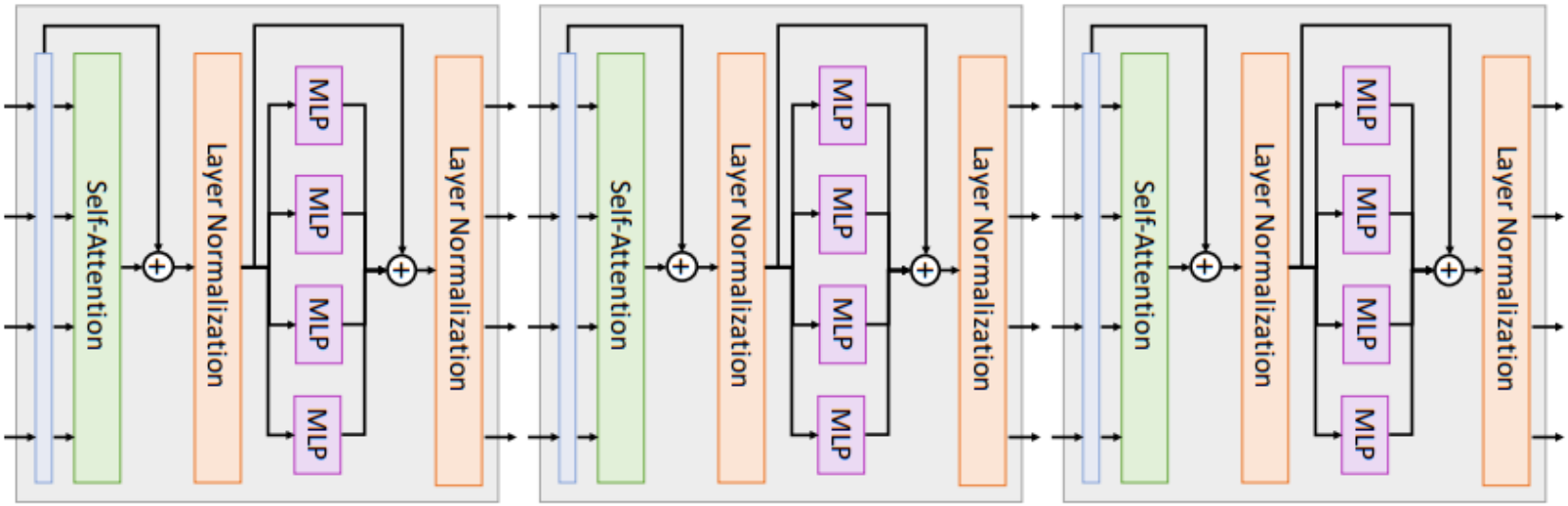
- **Input:** Set of vectors x
- **Output:** Set of vectors y

- Self-attention is the only interaction between vectors!
- Layer norm and MLP work independently per vector
- Highly scalable, highly parallelizable

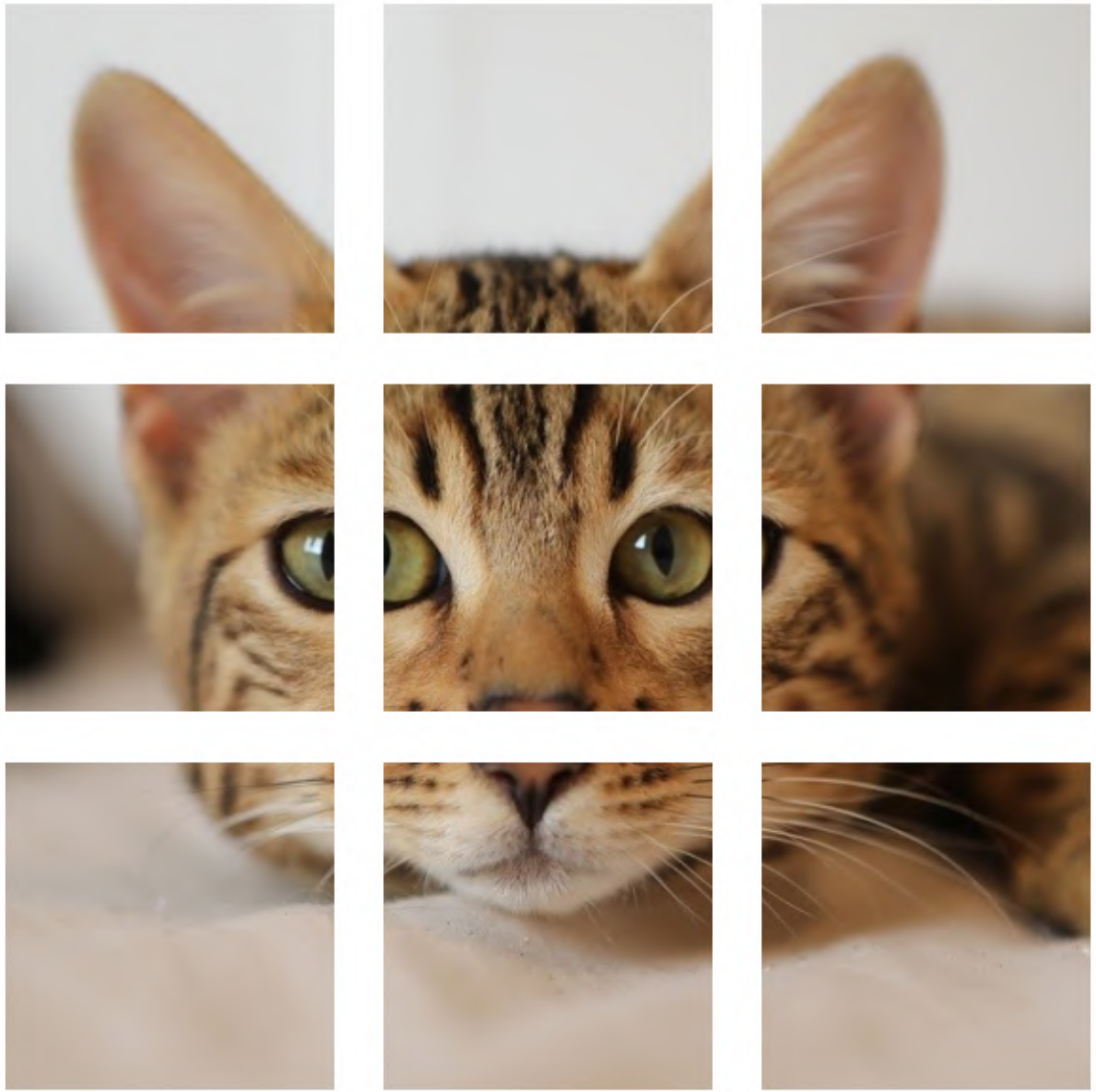


Transformers

- A Transformer is a sequence of transformer blocks
- Vaswani et al used 12 blocks, $D_Q = 512$ and 6 attention heads



Vision Transformers

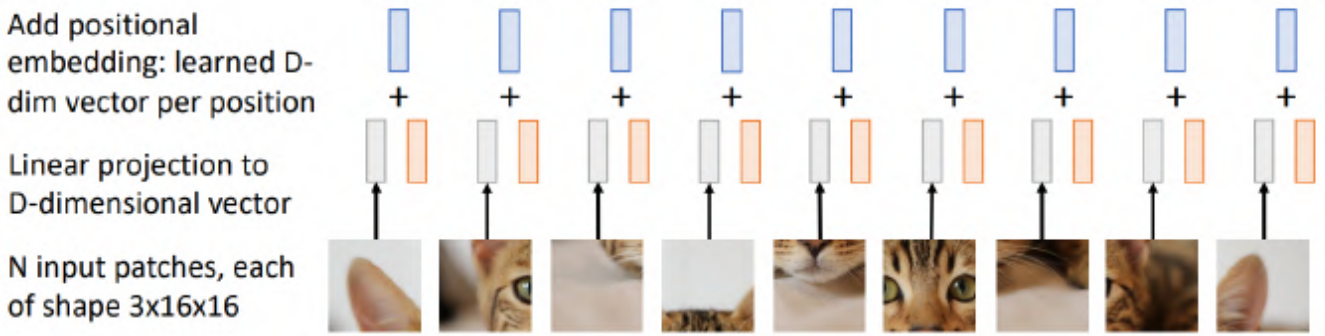


Vision Transformers

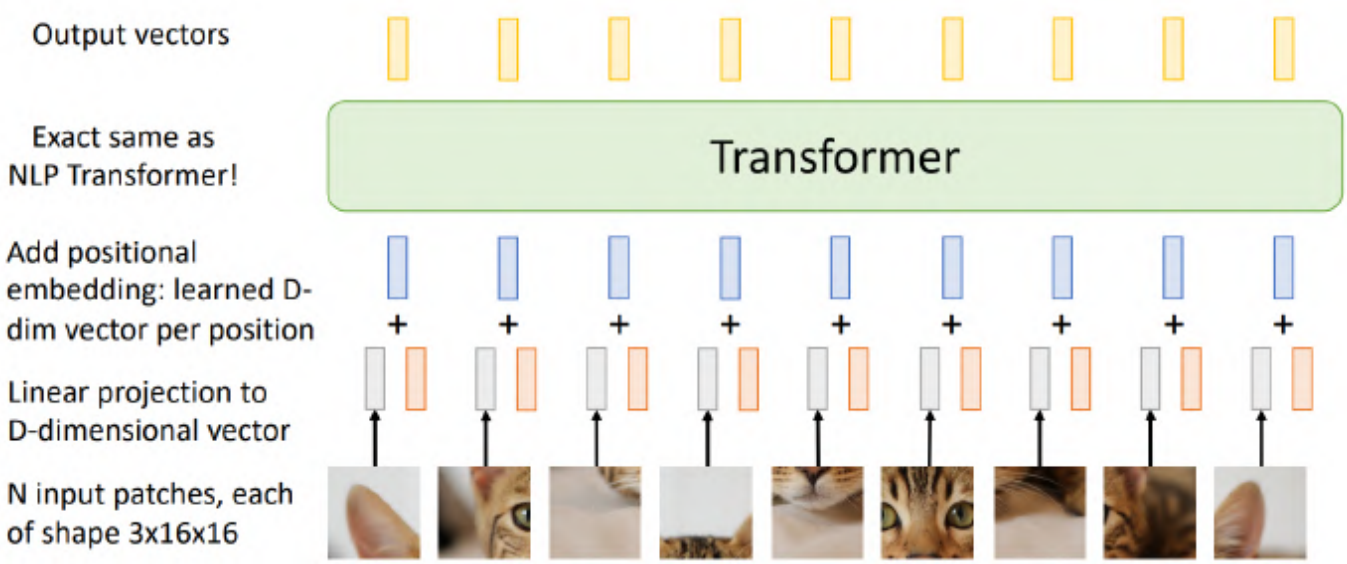
N input patches, each of shape 3x16x16

Vision Transformers

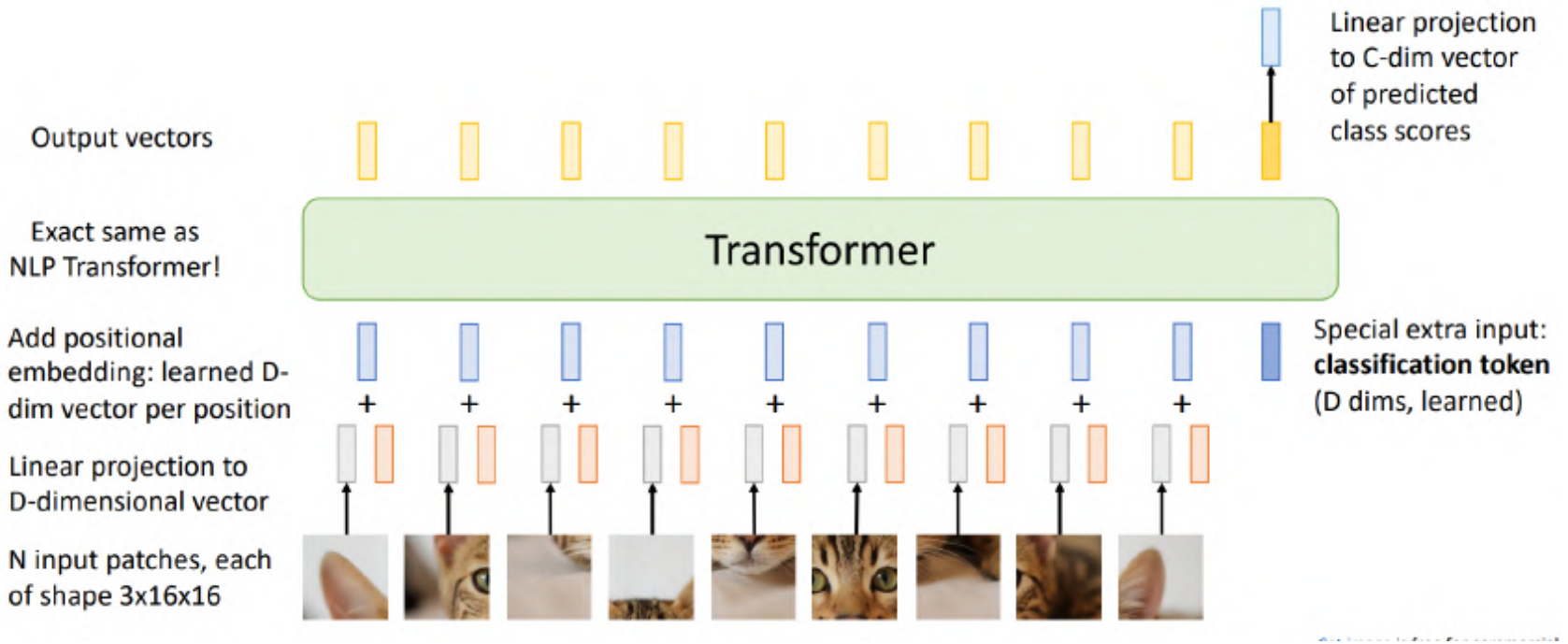
Vision Transformers



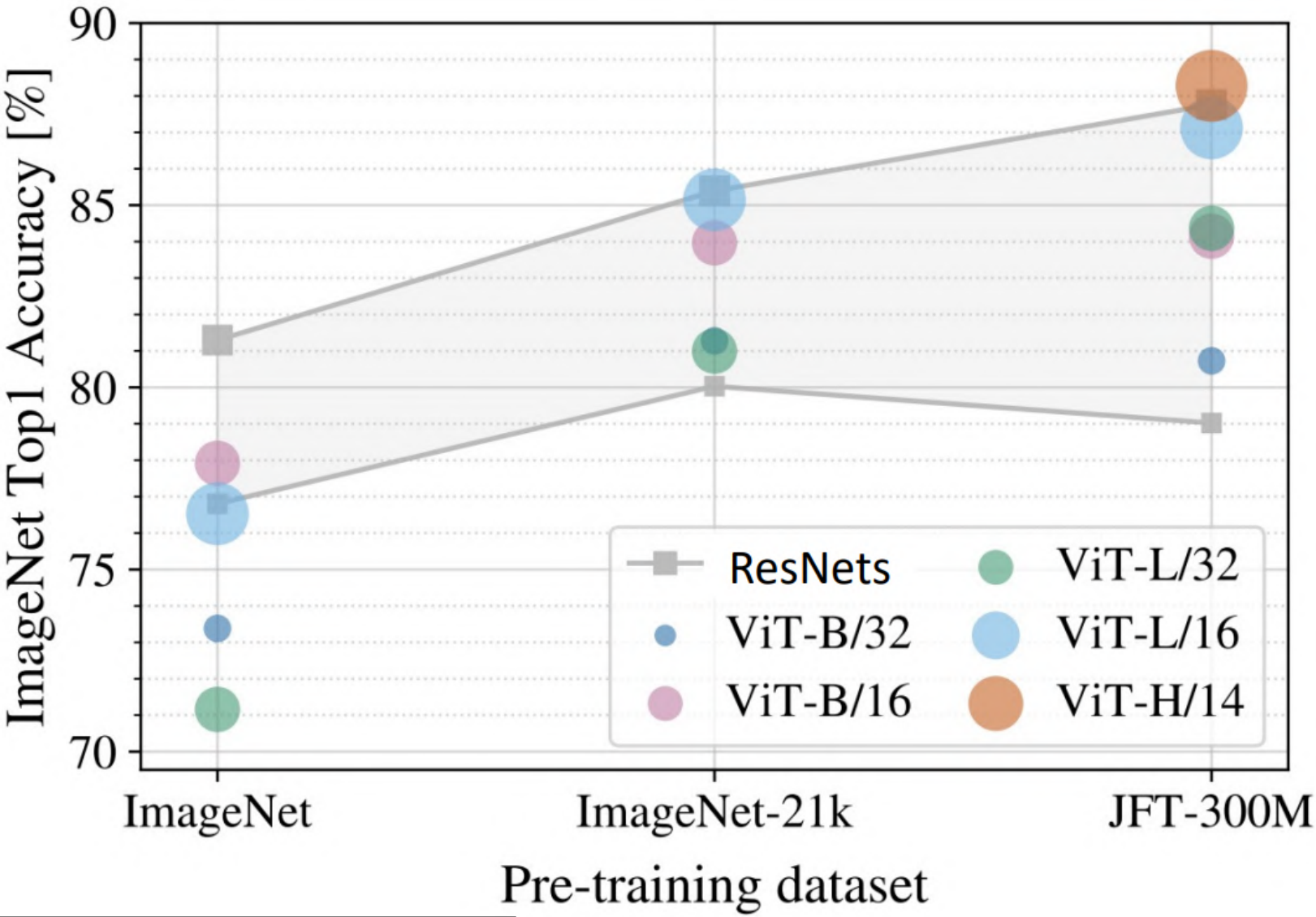
Vision Transformers



Vision Transformers



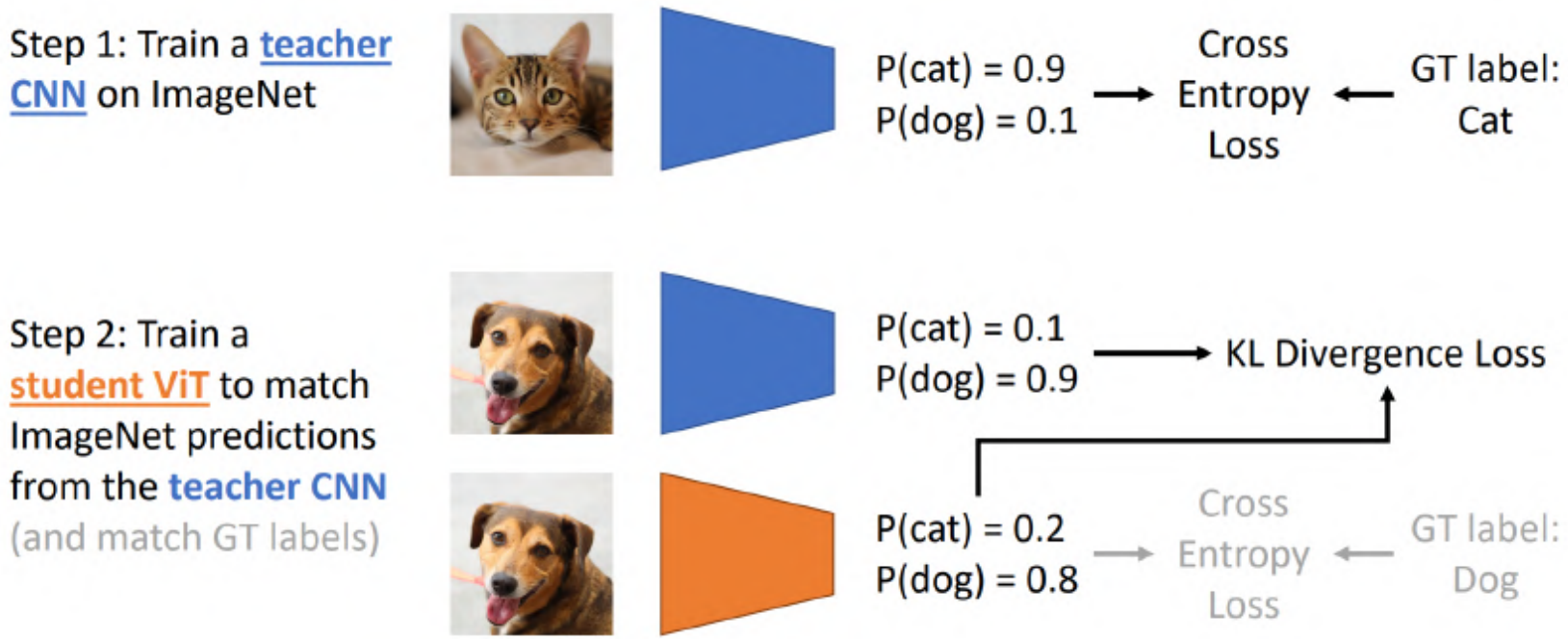
Vision Transformers



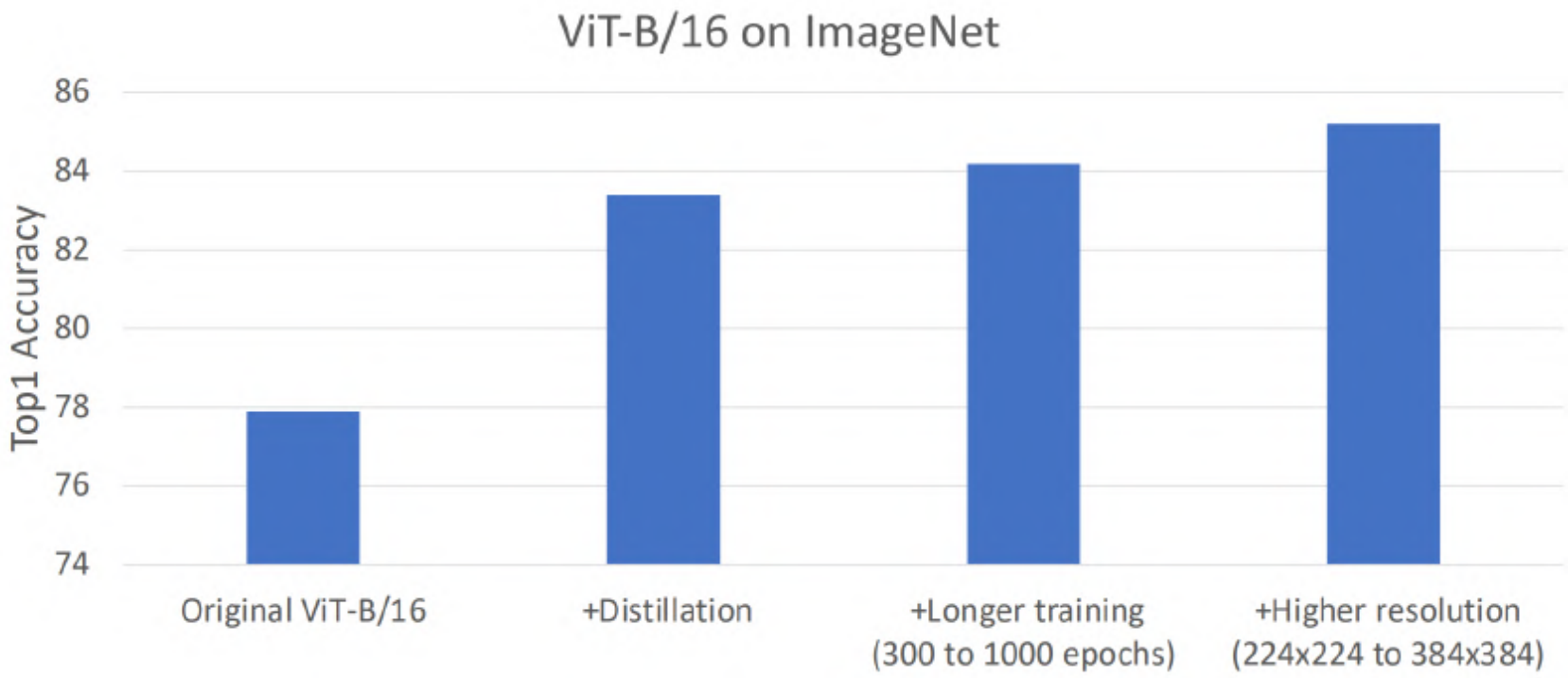
[0]Dosovitskiy et al, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR 2021

Frame Title

Improving ViT: Distillation



Improving ViT: Distillation

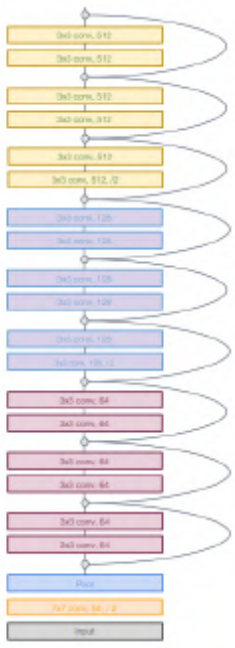


[0]Touvrom et al, "Training data-efficient image transformers distillation through attention", ICML 2021

CNN vs ViT

CNN		ViT
✓	1. Maintain 2D structure logic	✓
✓	2. Shift equivariant	✗
✓	3. Consider only local correlations	✗
✓	4. Hierarchically growing field of view	✗
✓	5. Hierarchically progressing complexity	✓
✓	6. Reasonable amount of params	✓
✗	7. Global representation	✓

CNN vs ViT

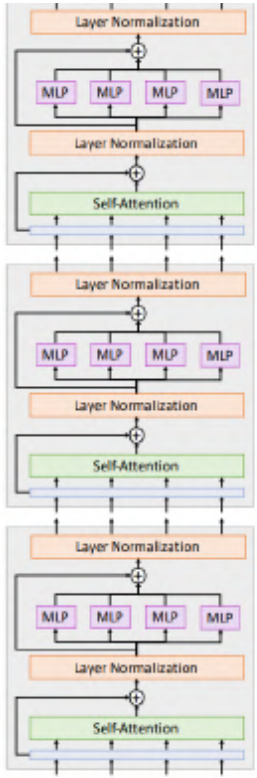


In most CNNs (including ResNets), **decrease** resolution and **increase** channels as you go deeper in the network (Hierarchical architecture)

Useful since objects in images can occur at various scales

In a ViT, all blocks have same resolution and number of channels (Isotropic architecture)

Can we build a **hierarchical** ViT model?



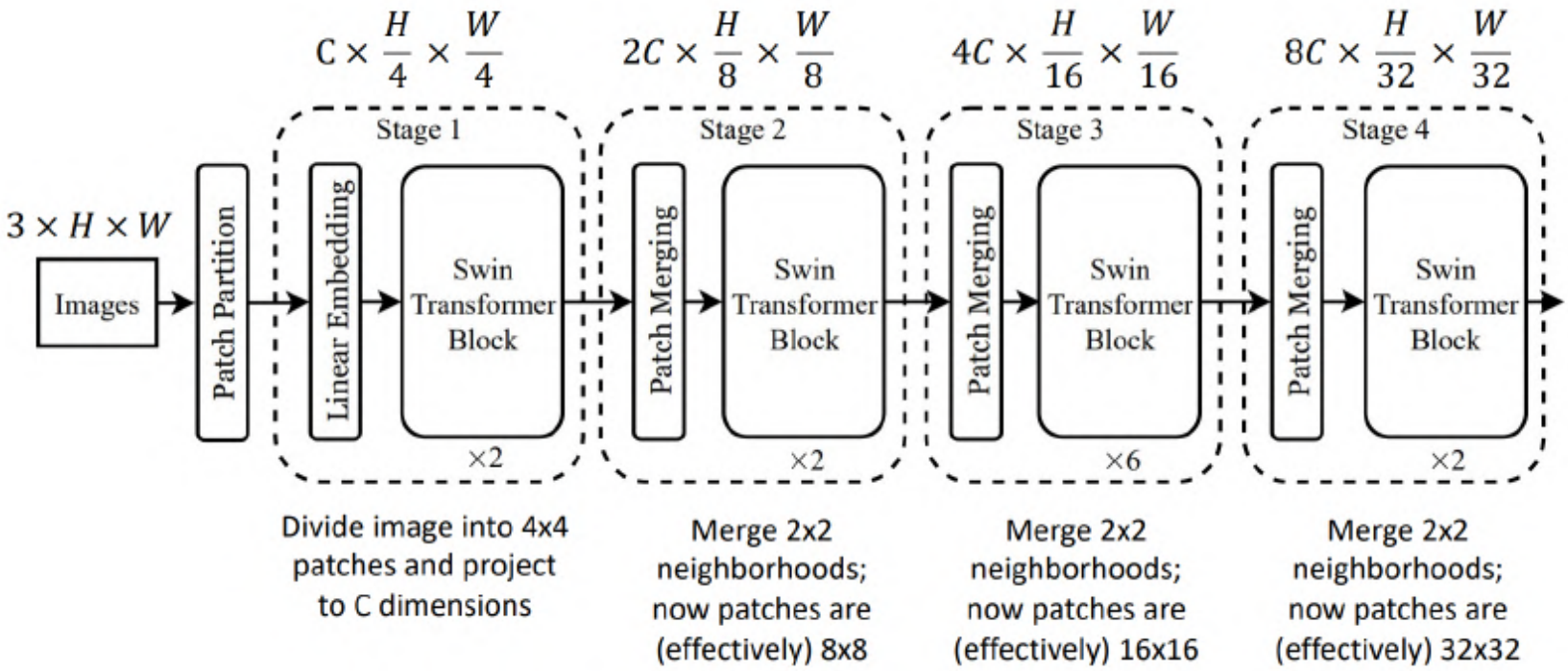
3rd block:
768 x 14 x 14

2nd block:
768 x 14 x 14

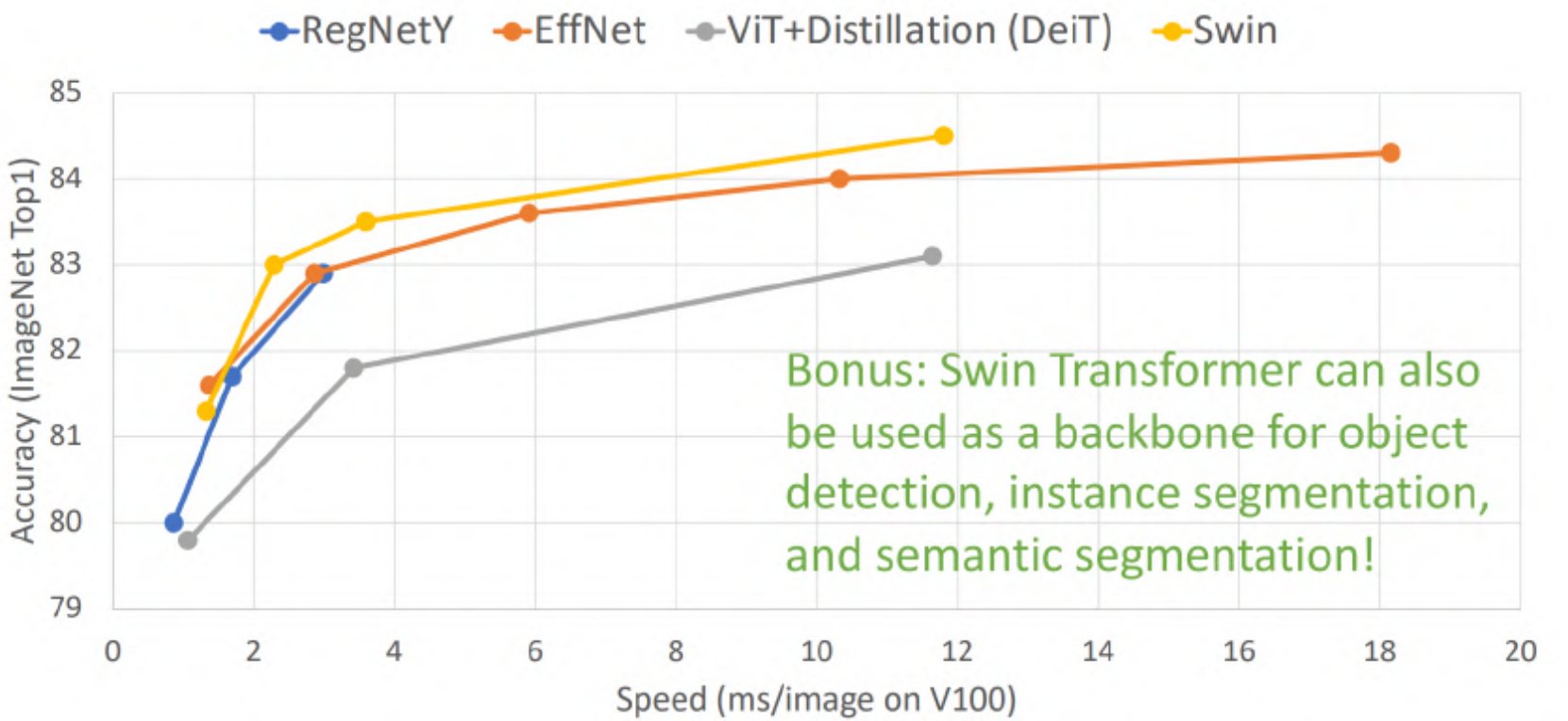
1st block:
768 x 14 x 14

Input:
3 x 224 x 224

Hierarchical ViT: Swin Transformer



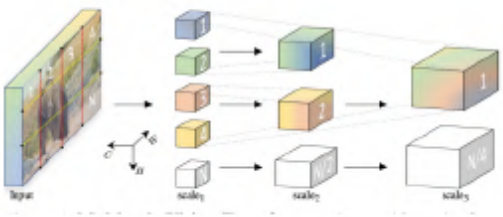
Hierarchical ViT: Swin Transformer



[0]Liu et al, "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows", CVPR 2021

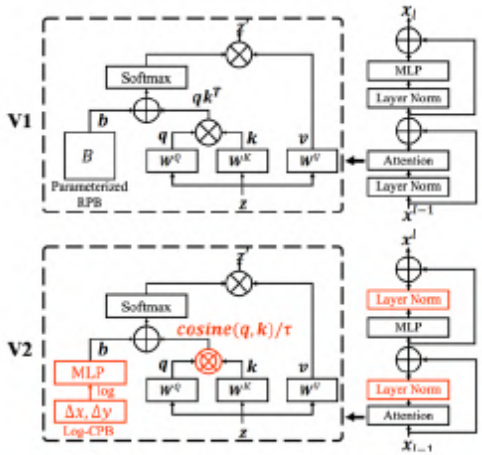
Other Hierarchical Vision Transformers

MViT



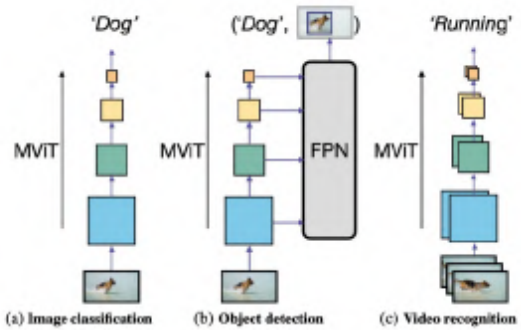
Fan et al, "Multiscale Vision Transformers", ICCV 2021

Swin-V2



Liu et al, "Swin Transformer V2: Scaling up Capacity and Resolution", CVPR 2022

Improved MViT



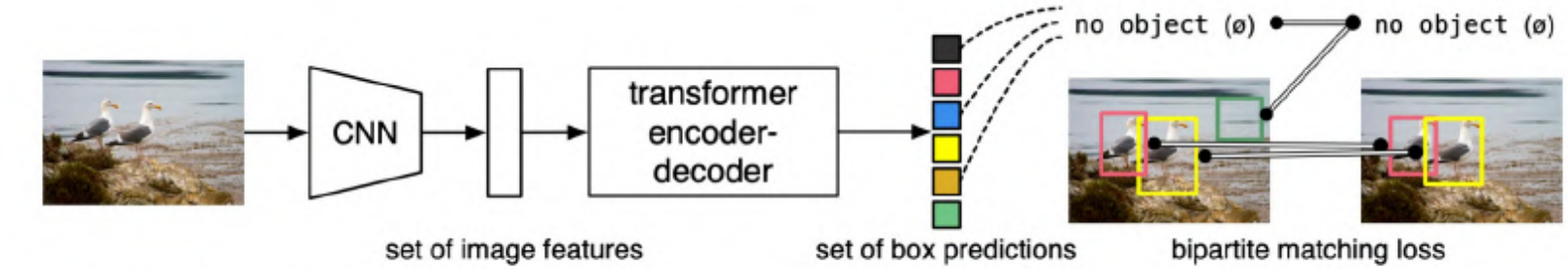
Li et al, "Improved Multiscale Vision Transformers for Classification and Detection", arXiv 2021

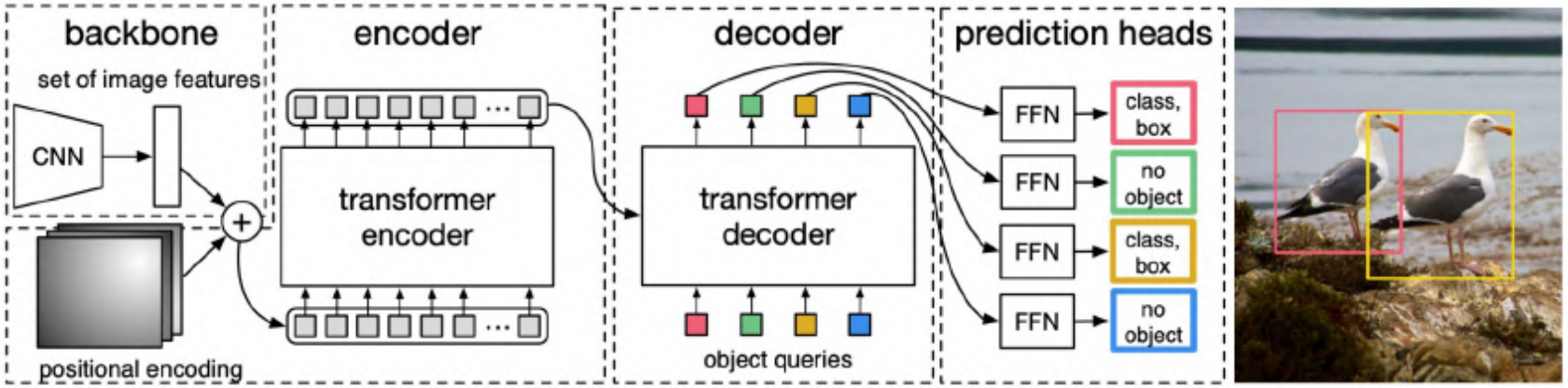
Object Detection with Transformers: DETR

- Simple object detection pipeline: directly output a set of boxes from a Transformer
- No anchors, no regression of box transforms
- Match predicted boxes to GT boxes with bipartite matching; train to regress box coordinates

[0] Carion et al, "End-to-End Object Detection with Transformers", ECCV 2020

Object Detection with Transformers: DETR





[0]Carion et al, "End-to-End Object Detection with Transformers", ECCV 2020

References

These slides have been adapted from

- Fei-Fei Li, Yunzhu Li & Ruohan Gao, Stanford CS231n: Deep Learning for Computer Vision
- Assaf Shocher, Shai Bagon, Meirav Galun & Tali Dekel, WAIC DL4CV Deep Learning for Computer Vision: Fundamentals and Applications
- Justin Johnson, UMich EECS 498.008/598.008: Deep Learning for Computer Vision