LECTURE 4: ATTENTION IS ALL YOU NEED

CSC5991: Introduction to LLMs

Slides taken from Stanford CS231n, CMU 16-385, and UWaterloo CS 886.

OUTLINE

Recap

Seq2Seq
CNN

Image Captioning with Visual Attention

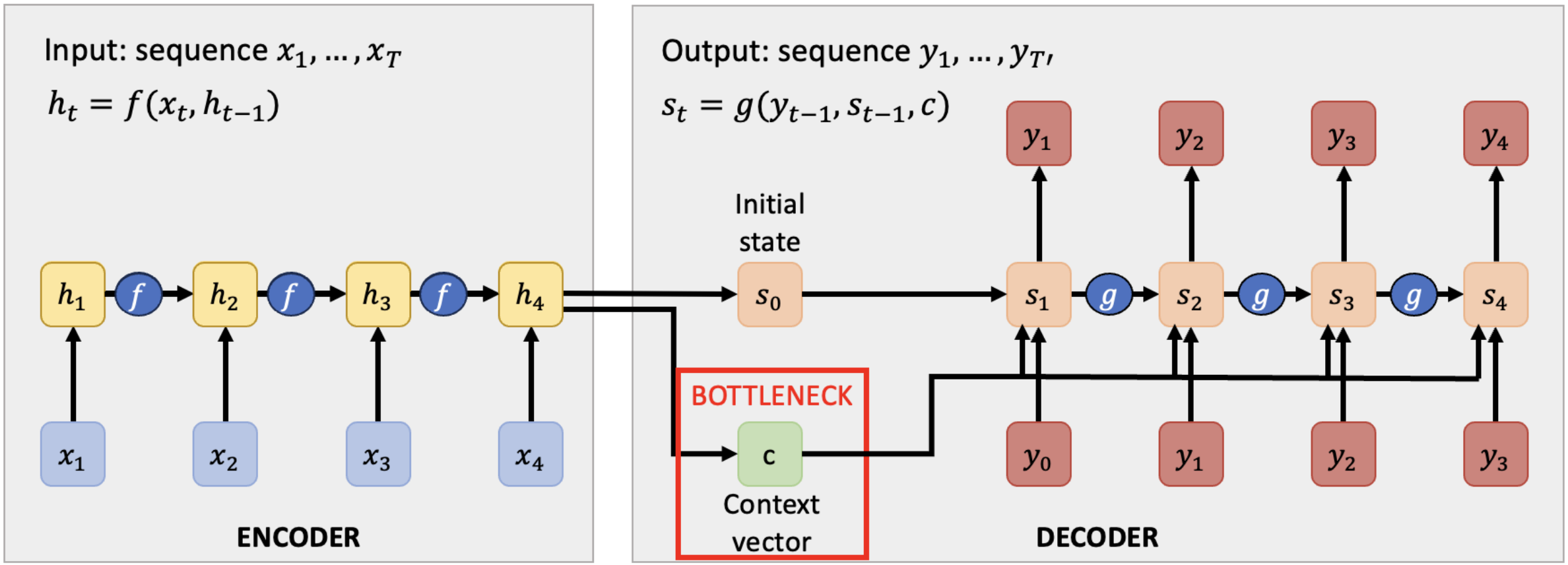
Attention Is All You Need:

Motivation
Attention decoupled from the RNN
Transformer architecture

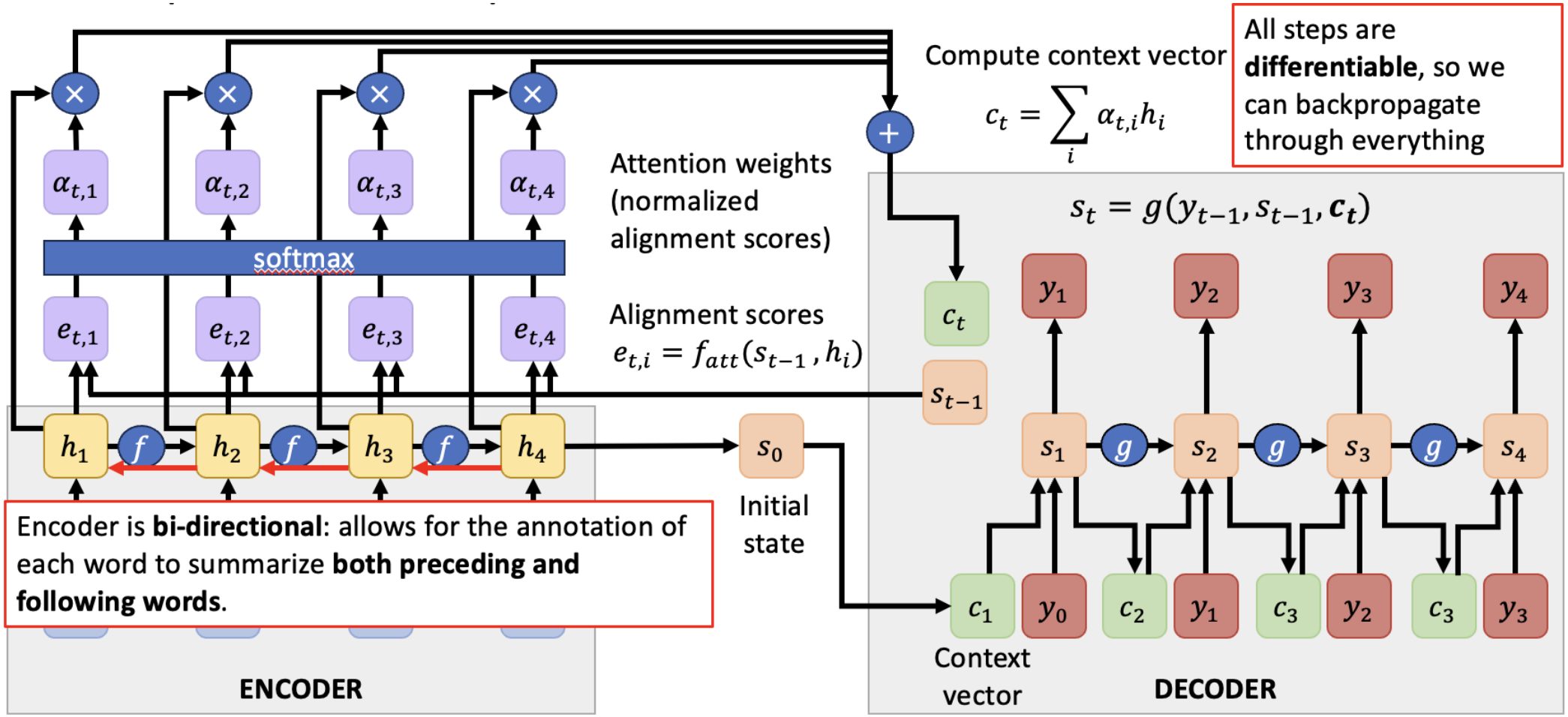
ATTENTION

- What we now call "attention" in DL
  - The idea of paying "**attention**" to the most relevant parts of I/P
  - Ideally, we'd like to **learn** this!
- There are many forms of attention mechanisms
  - **Additive**
  - **Dot-product**
- We have names to distinguish attention based on what is attended to
  - **Self-attention** (intra attention)
  - **Cross-attention** (encoder-decoder attention/inter attention)

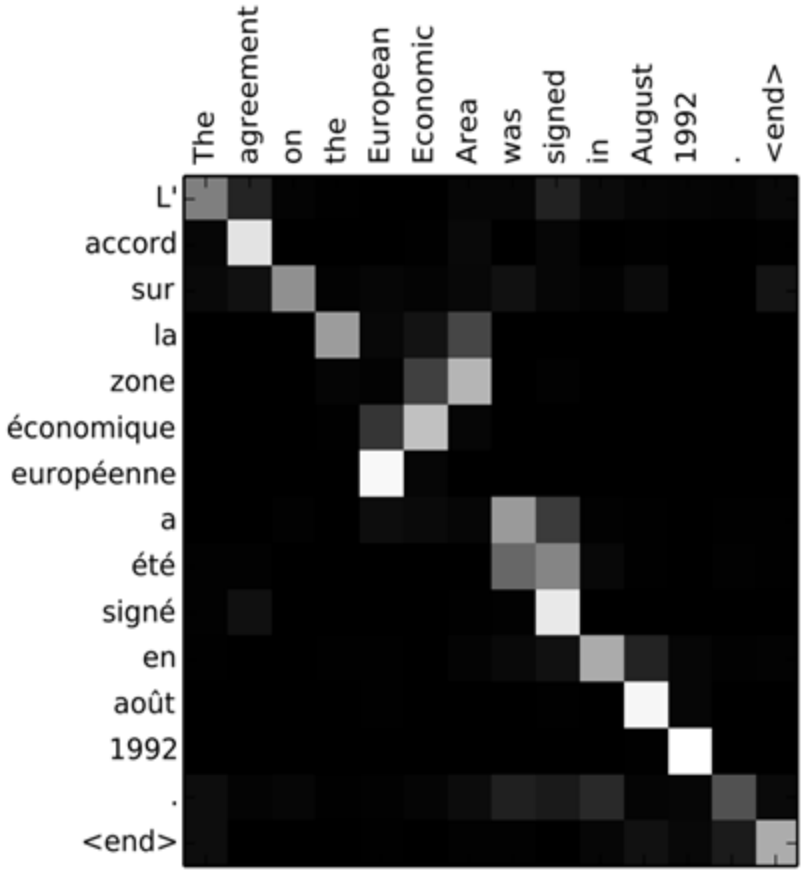
BOTTLENECK


Input: sequence $x_1, \dots, x_T$
$h_t = f(x_t, h_{t-1})$
h_1 f h_2 f h_3 f h_4
x_1 x_2 x_3 x_4
ENCODER
Output: sequence $y_1, \dots, y_{T'}$
$s_t = g(y_{t-1}, s_{t-1}, c)$
Initial state
s_0
BOTTLENECK
c
Context vector
y_1 y_2 y_3 y_4
s_1 g s_2 g s_3 g s_4
y_0 y_1 y_2 y_3
DECODER

SEQ2SEQ: RNNS + **ATTENTION**



ATTENTION MAP



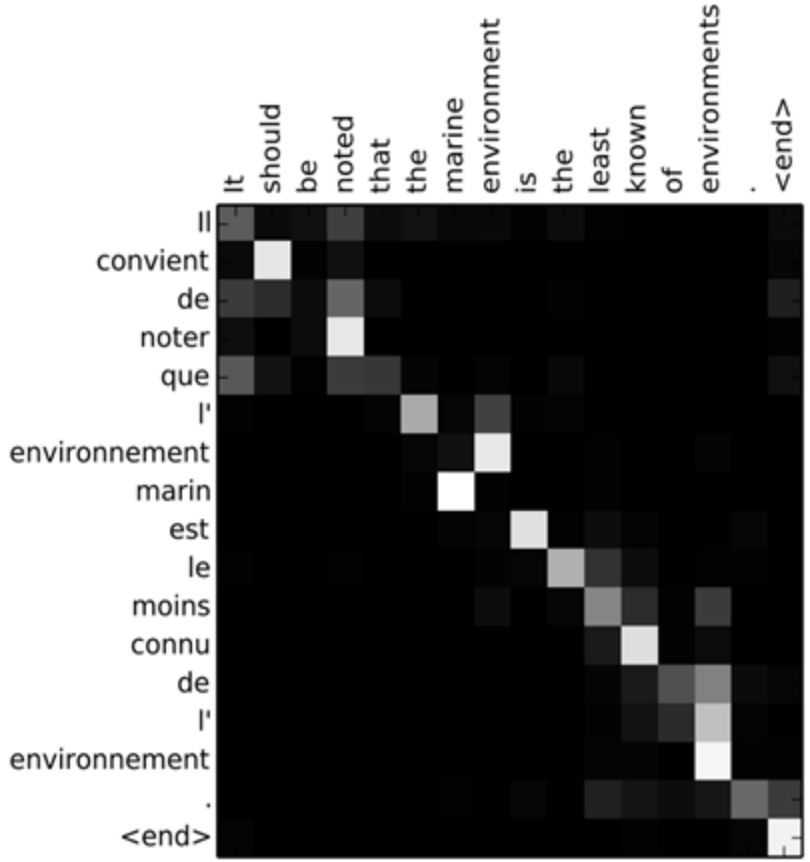


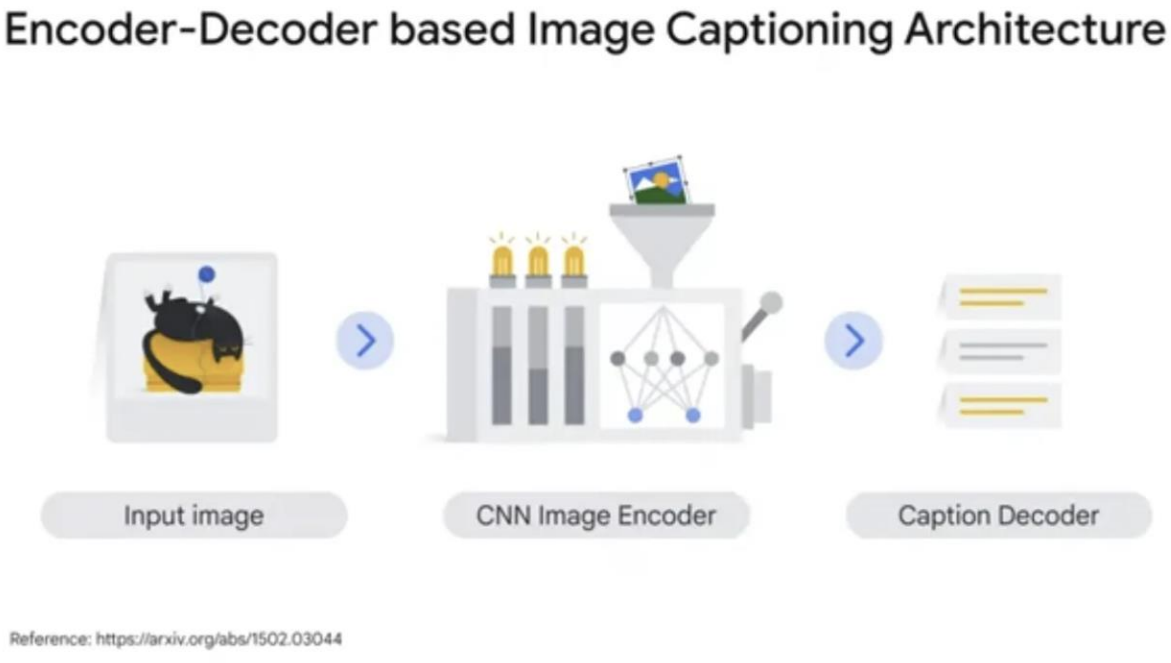
IMAGE CAPTIONING

- The task of the task of describing the content of an image in words.

ARCHITECTURE

Goal: Build and train a model that can generate text captions based on images.

- **Encoder-decoder model**
- **Encoder:** Images are passed to the encoder
  - Extracts information from the images and creates feature vectors.
  - The encoder can use any image backbone like CNNs, ResNets, ViTs.
- **Decoder:** Builds captions by generating words.



ENCODER: **CNN**-BASED ARCHITECTURE

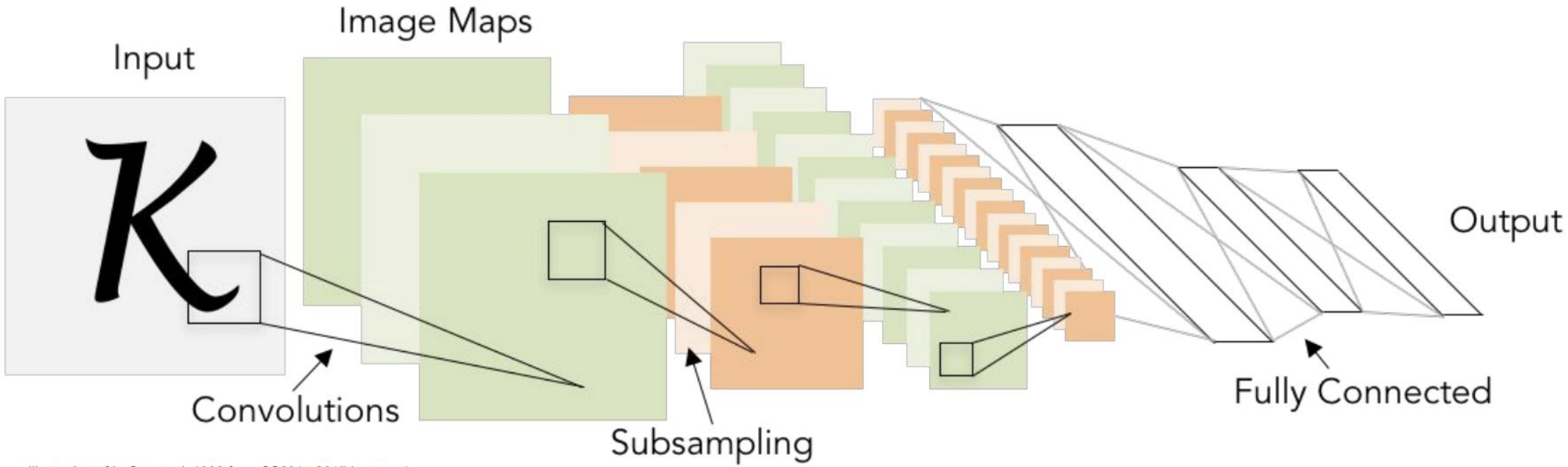
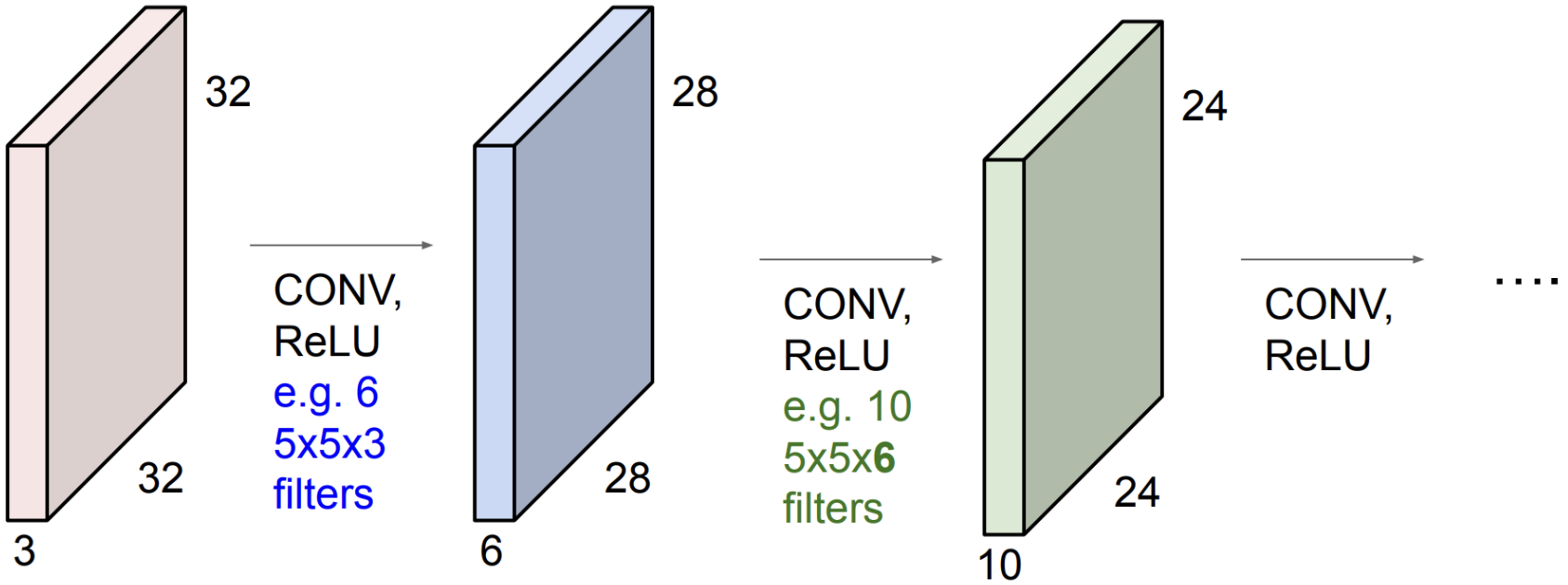


Illustration of LeCun et al. 1998 from CS231n 2017 Lecture 1

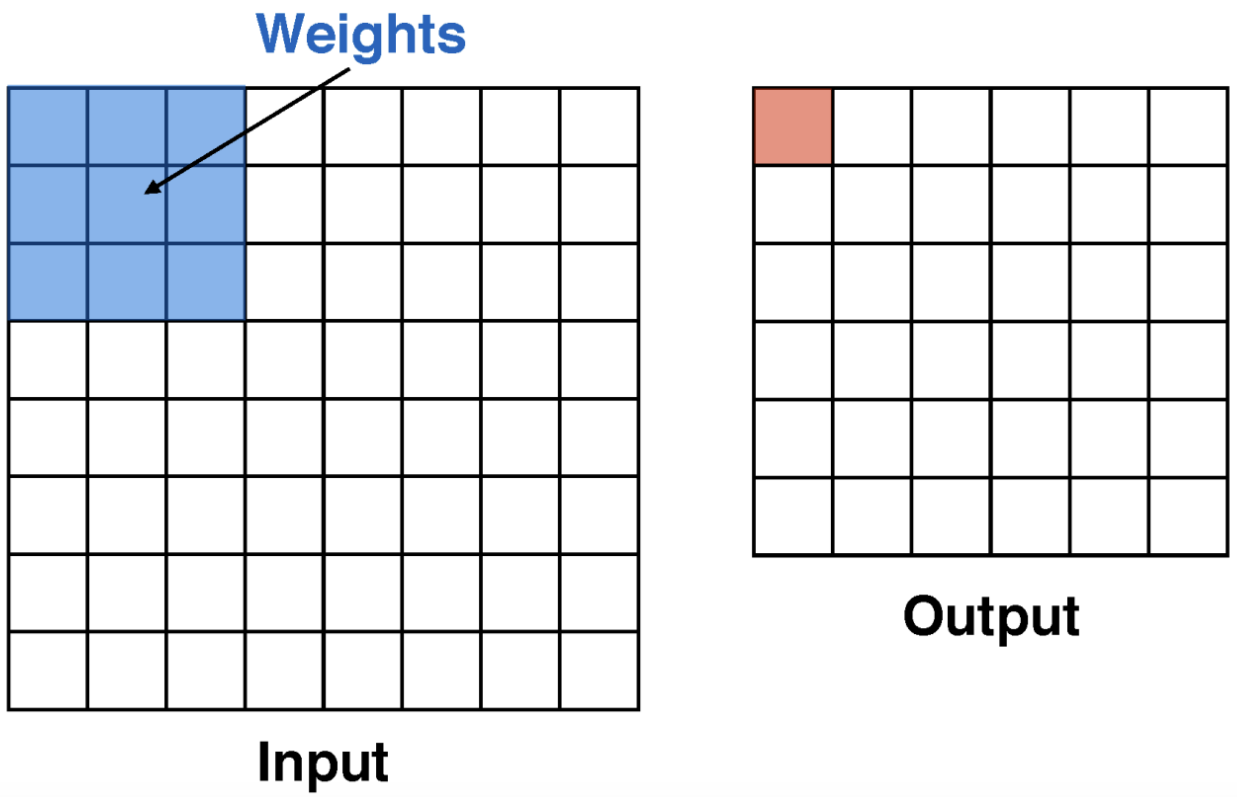
CNN: SEQUENCE OF CONVOLUTION LAYERS

Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions



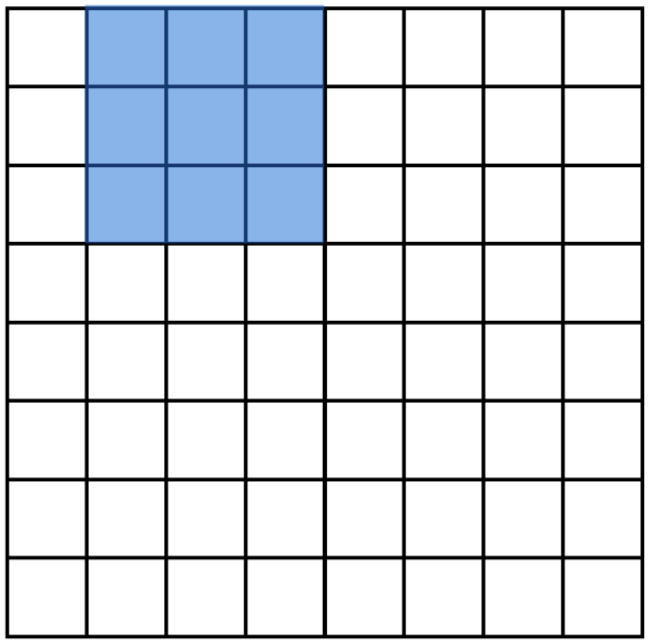
Convolution: Stride

During convolution, the weights “slide” along the input to generate each output

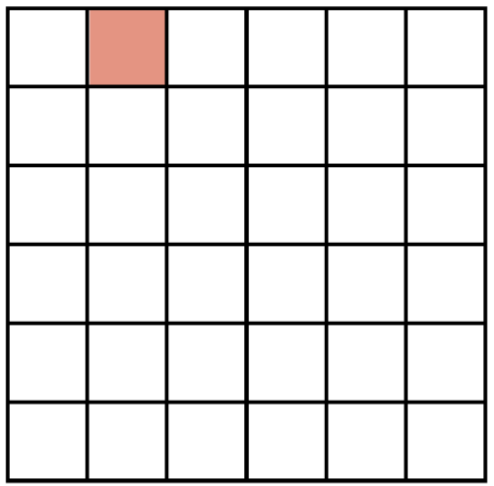


Convolution: Stride

During convolution, the weights “slide” along the input to generate each output



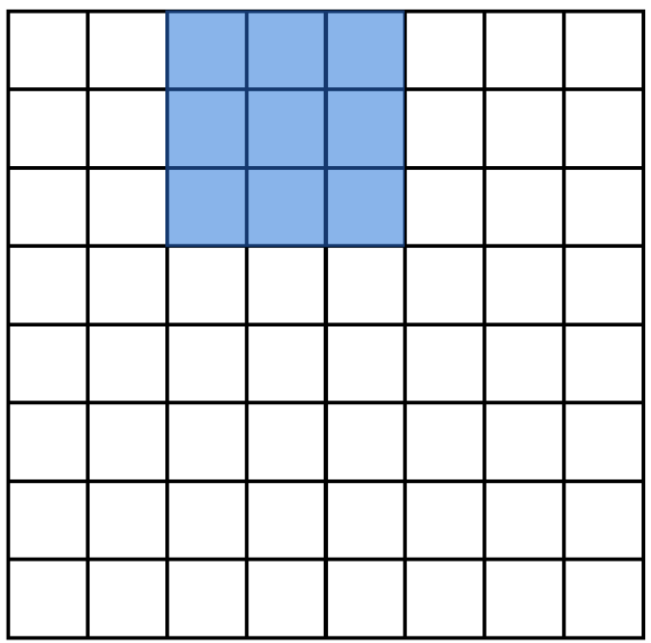
Input



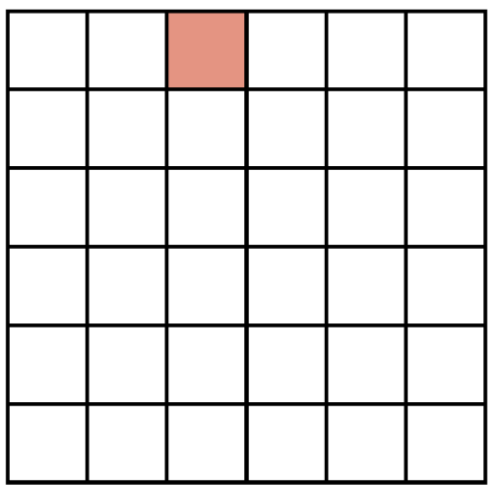
Output

Convolution: Stride

During convolution, the weights “slide” along the input to generate each output



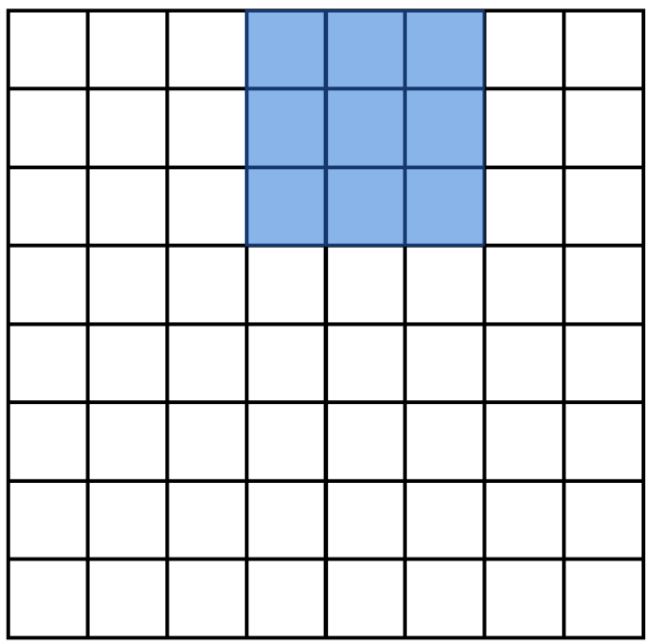
Input



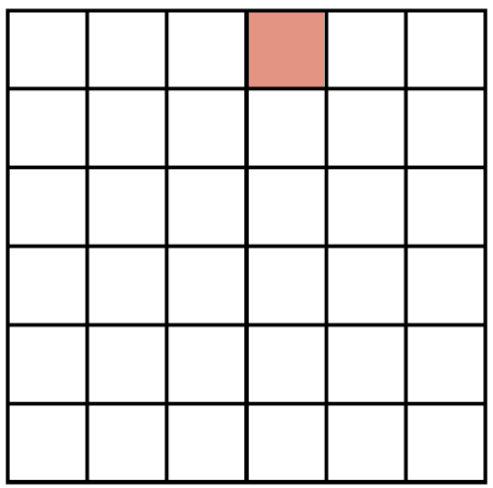
Output

Convolution: Stride

During convolution, the weights “slide” along the input to generate each output



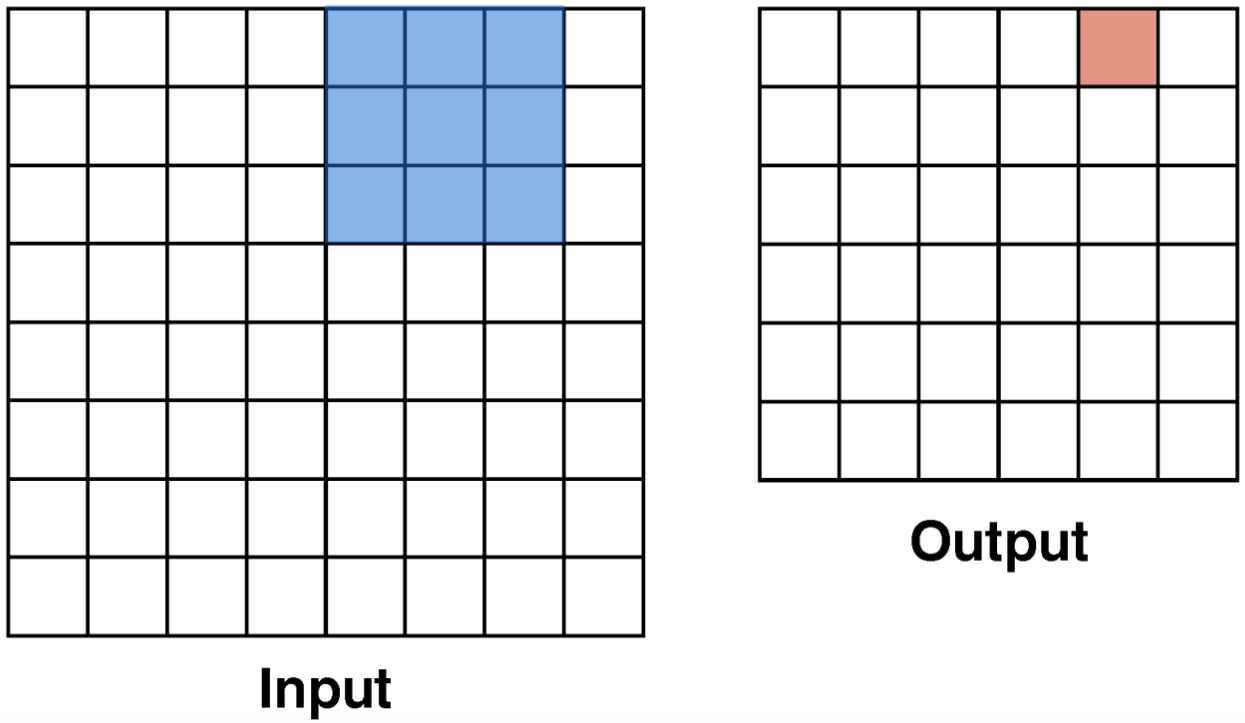
Input



Output

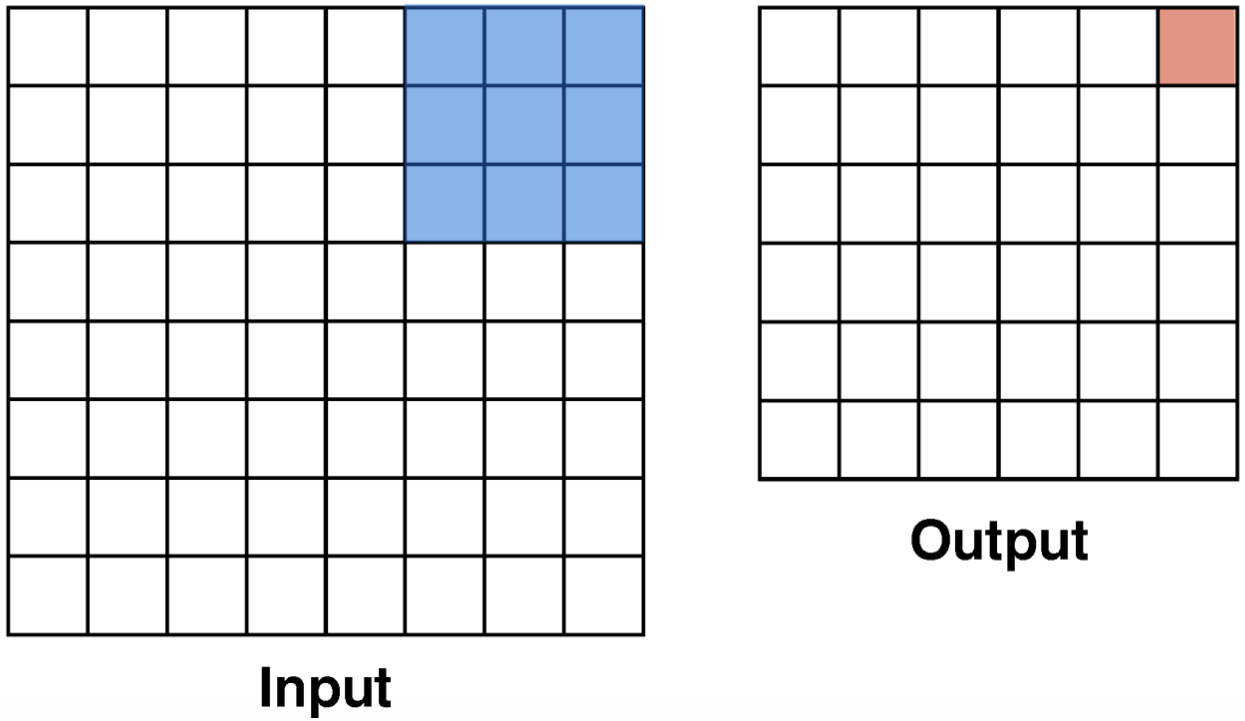
Convolution: Stride

During convolution, the weights “slide” along the input to generate each output



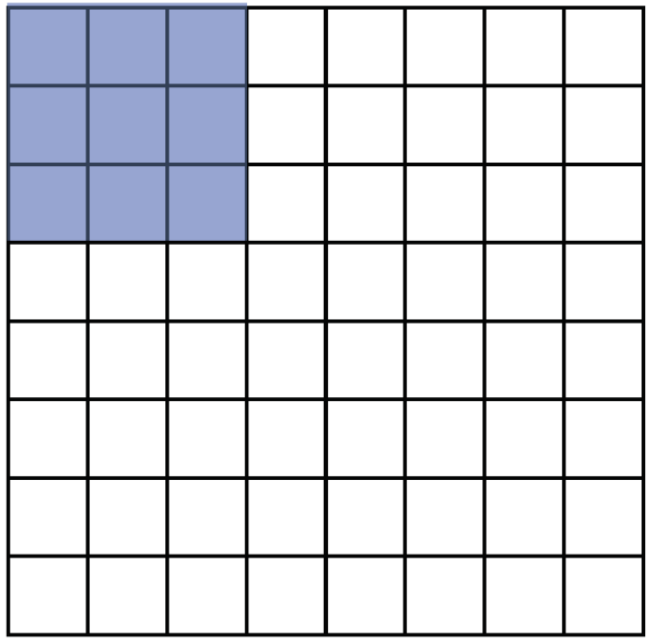
Convolution: Stride

During convolution, the weights “slide” along the input to generate each output



Convolution: Stride

During convolution, the weights “slide” along the input to generate each output



Input

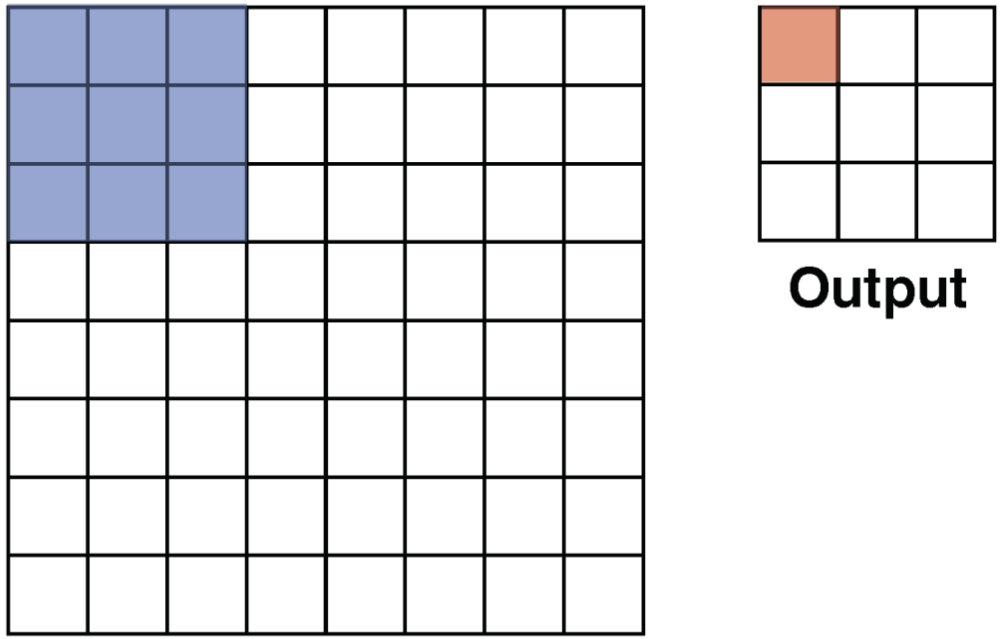
Recall that at each positior we are doing a **3D** sum:

$$h^r = \sum_{ijk} x^r{}_{ijk} W_{ijk} + b$$

(channel, row, column)

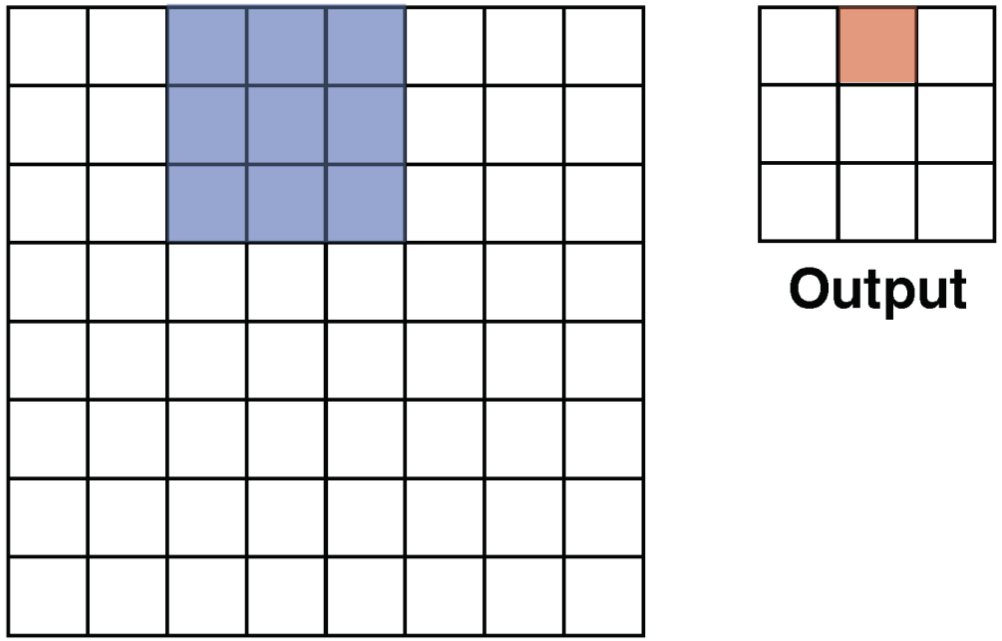
Convolution: Stride

But we can also convolve with a **stride**, e.g. stride = 2



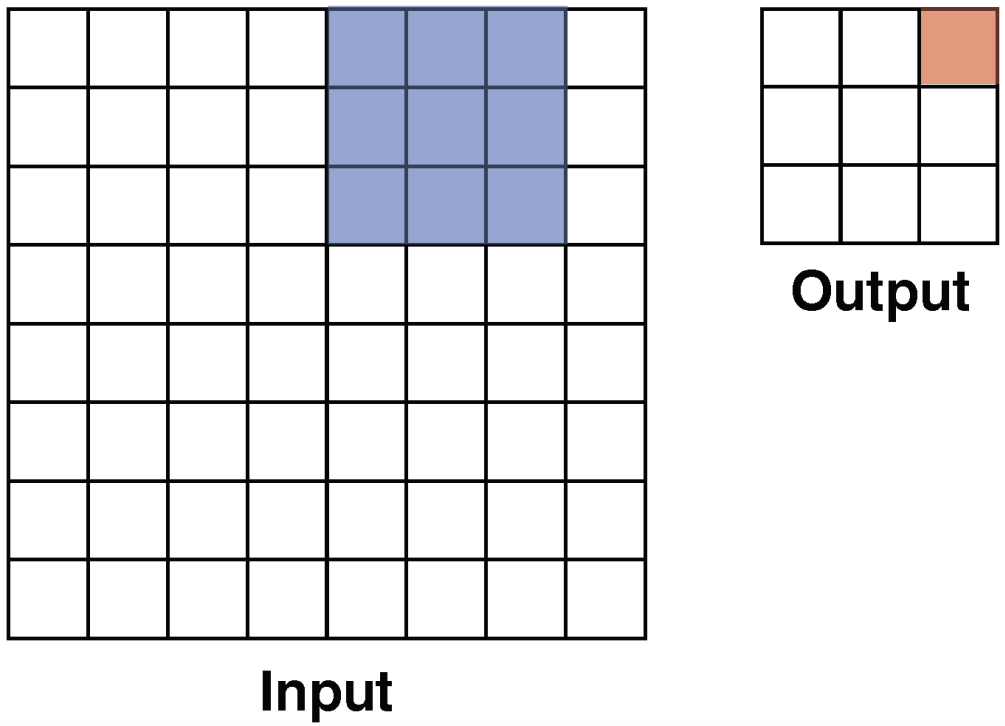
Convolution: Stride

But we can also convolve with a **stride**, e.g. stride = 2



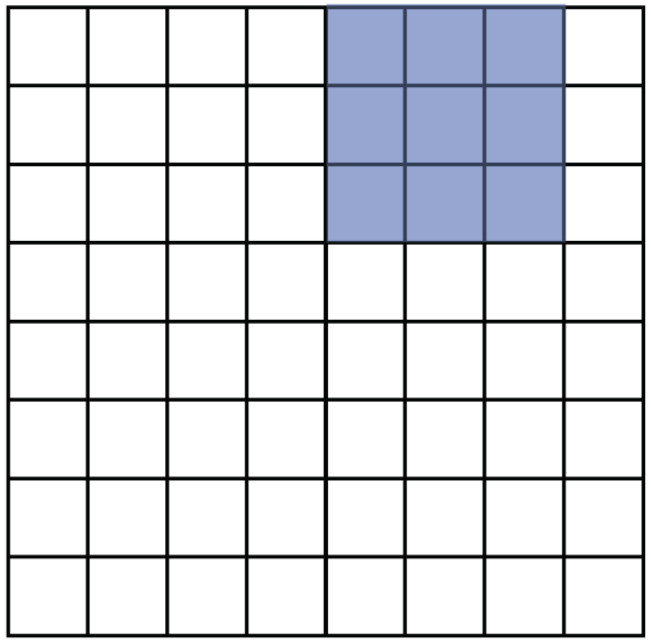
Convolution: Stride

But we can also convolve with a **stride**, e.g. stride = 2

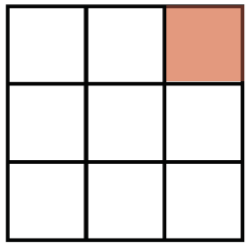


Convolution: Stride

But we can also convolve with a **stride**, e.g. stride = 2



Input



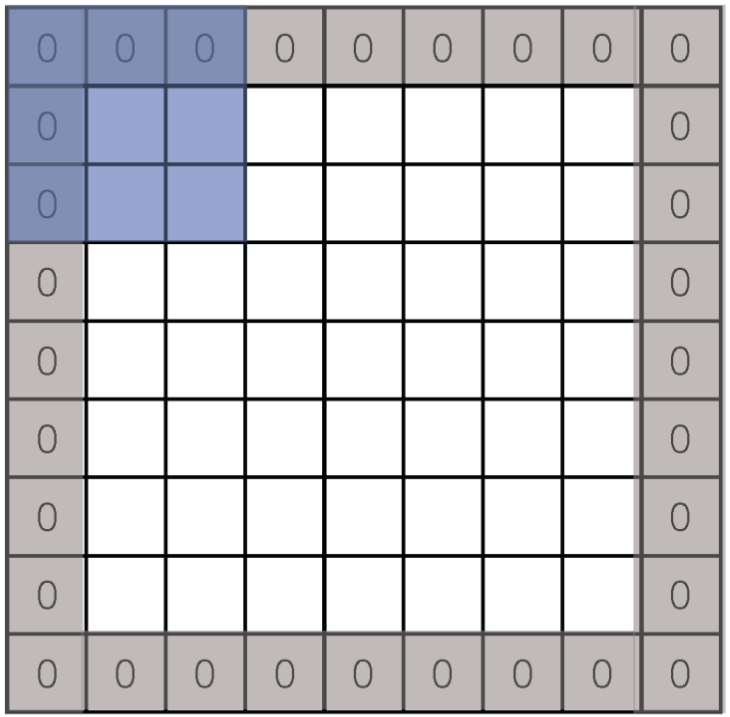
Output

- Notice that with certain strides, we may not be able t cover all of the input

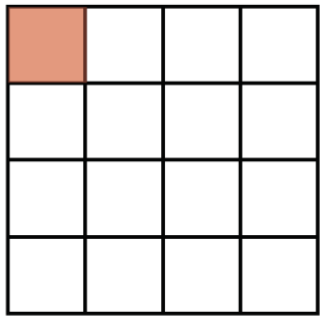
- The output is also half the size of the input

Convolution: Padding

We can also pad the input with zeros.
Here, **pad = 1, stride = 2**



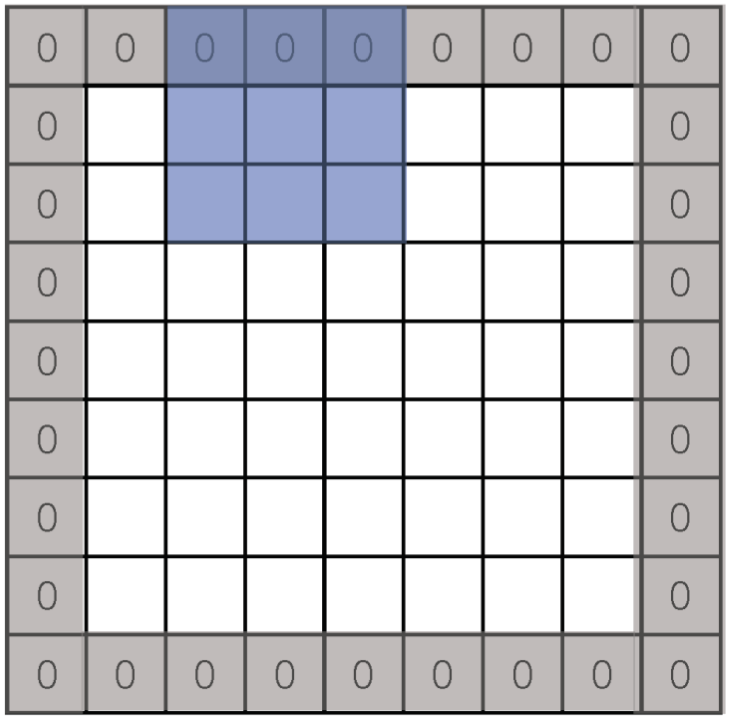
Input



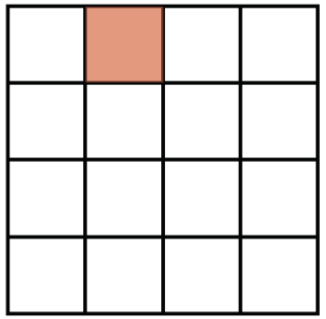
Output

Convolution: Padding

We can also pad the input with zeros.
Here, **pad = 1, stride = 2**



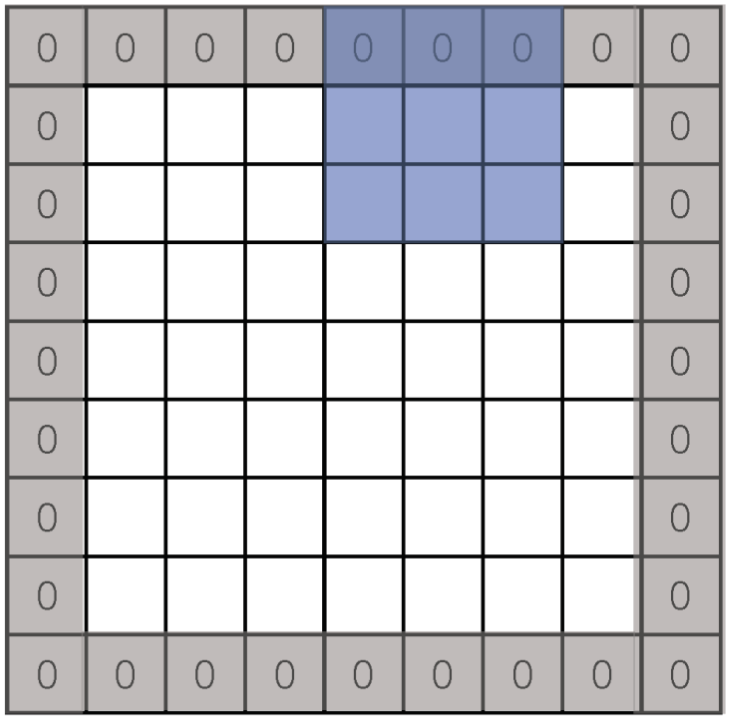
Input



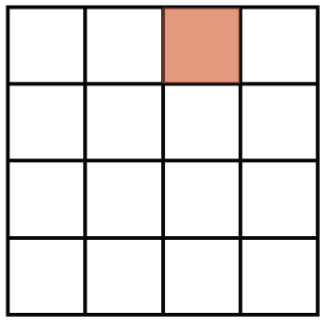
Output

Convolution: Padding

We can also pad the input with zeros.
Here, **pad = 1, stride = 2**



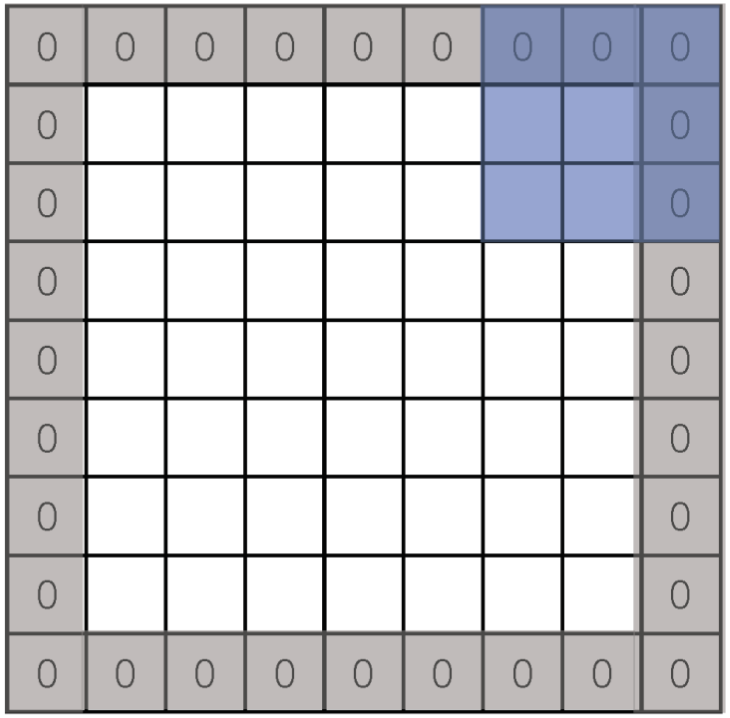
Input



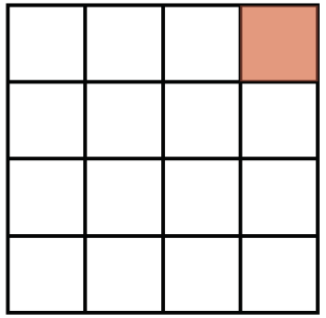
Output

Convolution: Padding

We can also pad the input with zeros.
Here, **pad = 1, stride = 2**



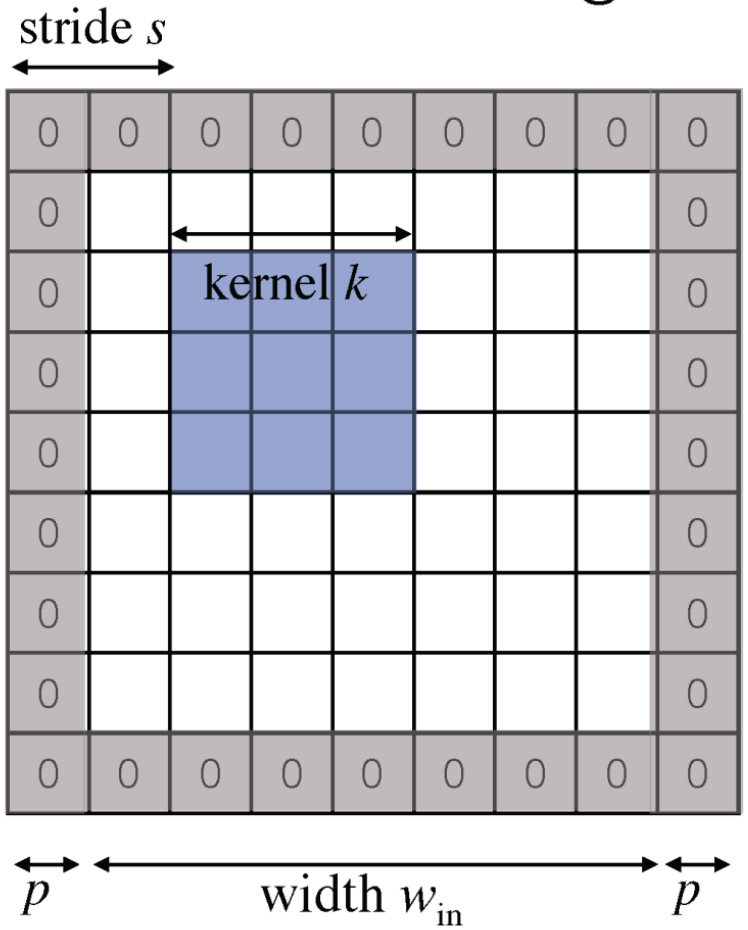
Input



Output

Convolution:

How big is the output?

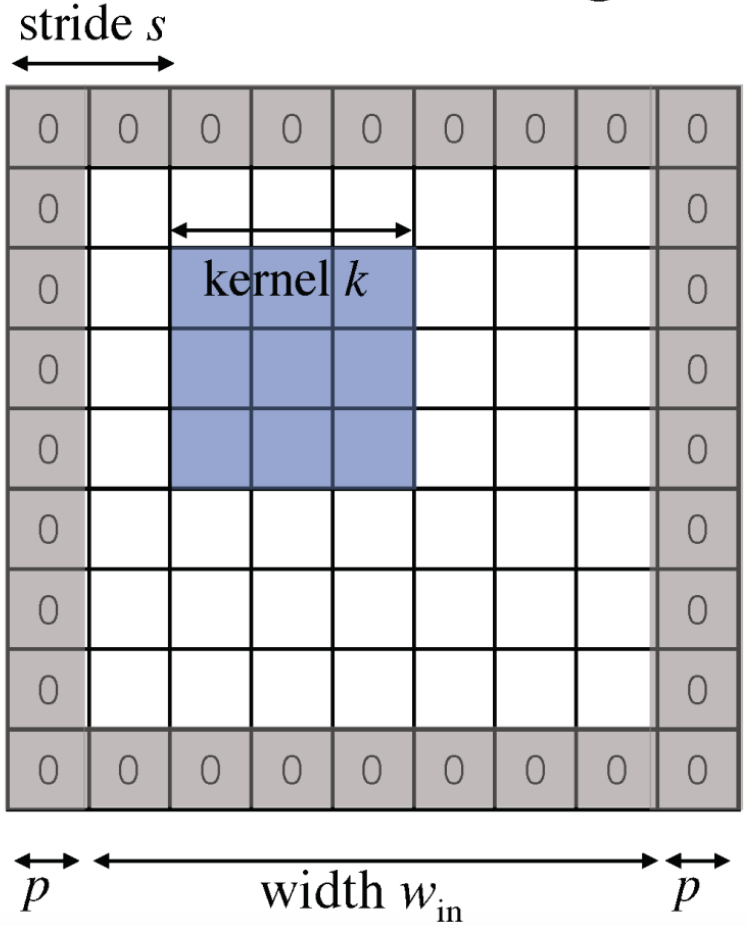


In general, the output has size:

$$w_{out} = \left\lfloor \frac{w_{in} + 2p - k}{s} \right\rfloor + 1$$

Convolution:

How big is the output?



Example: k=3, s=1, p=1

$$w_{out} = \left\lfloor \frac{w_{in} + 2p - k}{s} \right\rfloor + 1$$

$$= \left\lfloor \frac{w_{in} + 2 - 3}{1} \right\rfloor + 1$$

$$= w_{in}$$

VGGNet [Simonyan 2014] uses filters of this shape

Pooling

For most ConvNets, **convolution** is often followed by **pooling**:

- Creates a smaller representation while retaining the most important information
- The “max” operation is the most common
- Why might “avg” be a poor choice?

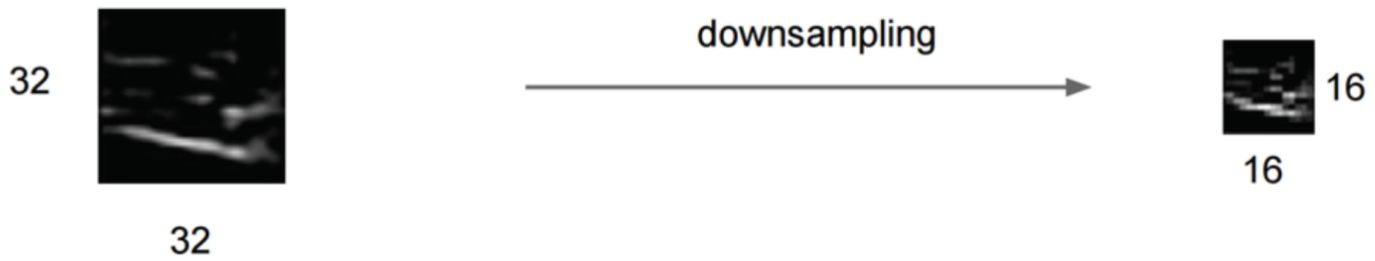
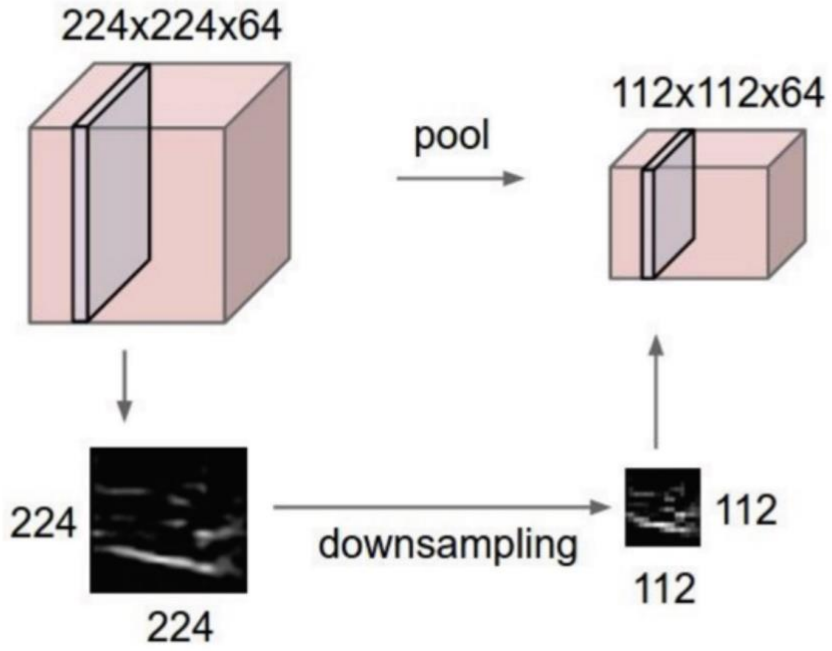


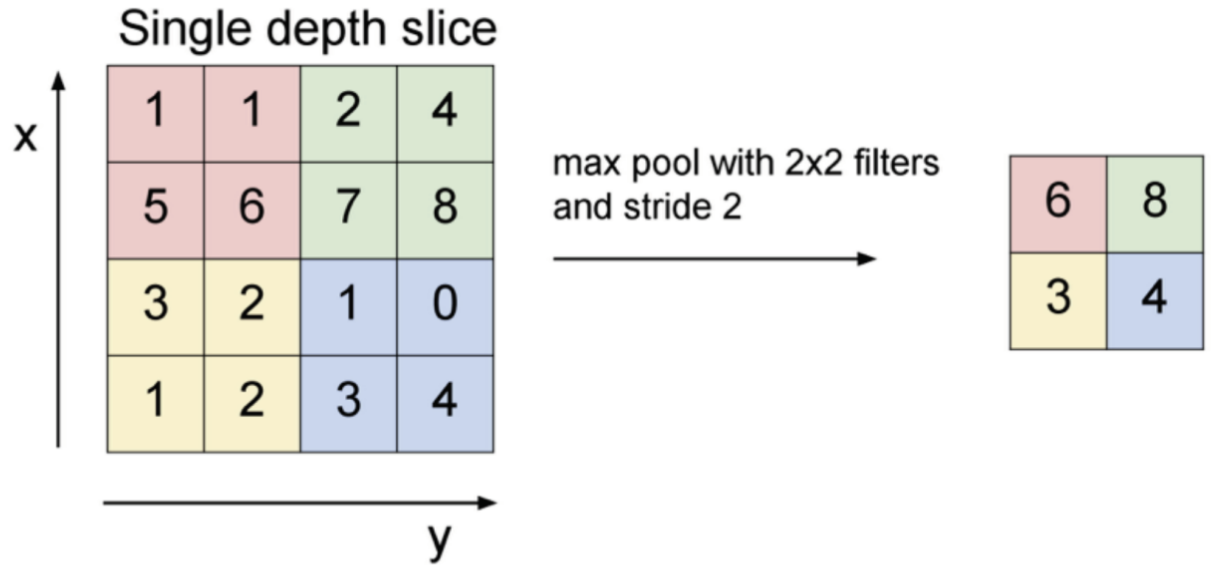
Figure: Andrej Karpathy

Pooling

- makes the representations smaller and more manageable
- operates over each activation map independently:



Max Pooling



What's the backprop rule for max pooling?

- In the forward pass, store the index that took the max
- The backprop gradient is the input gradient at that index

Figure: Andrej Karpathy

Example ConvNet

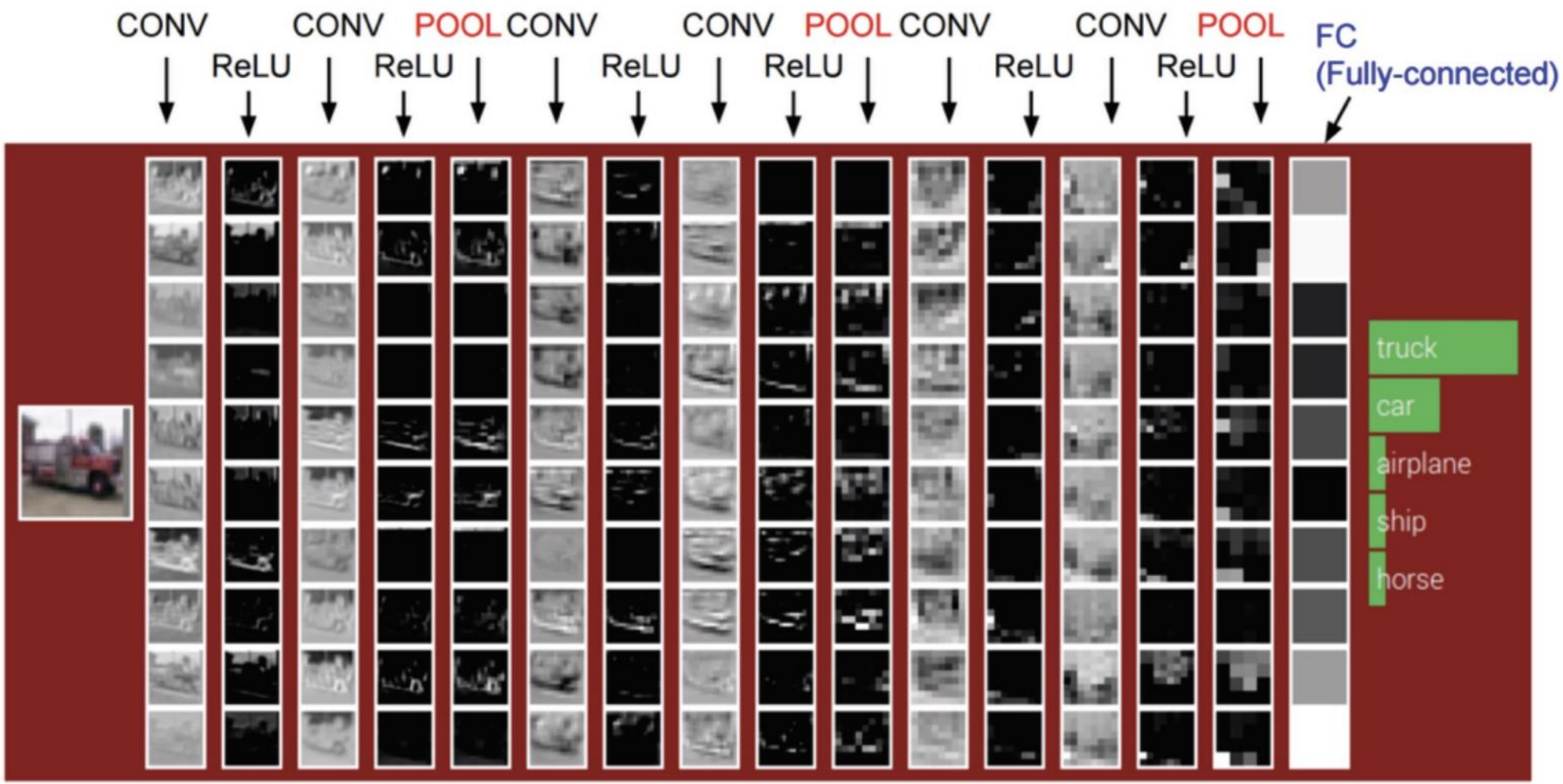


Figure: Andrej Karpathy

RESOURCES

- **CIFAR-10**
- https://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html

- **MNIST**
- https://cs.stanford.edu/people/karpathy/convnetjs/demo/mnist.html

IMAGE CAPTIONING WITH VISUAL ATTENTION

- Similarly to Seq2Seq we can use attention for image captioning (image → text)
- Builds directly on previous work

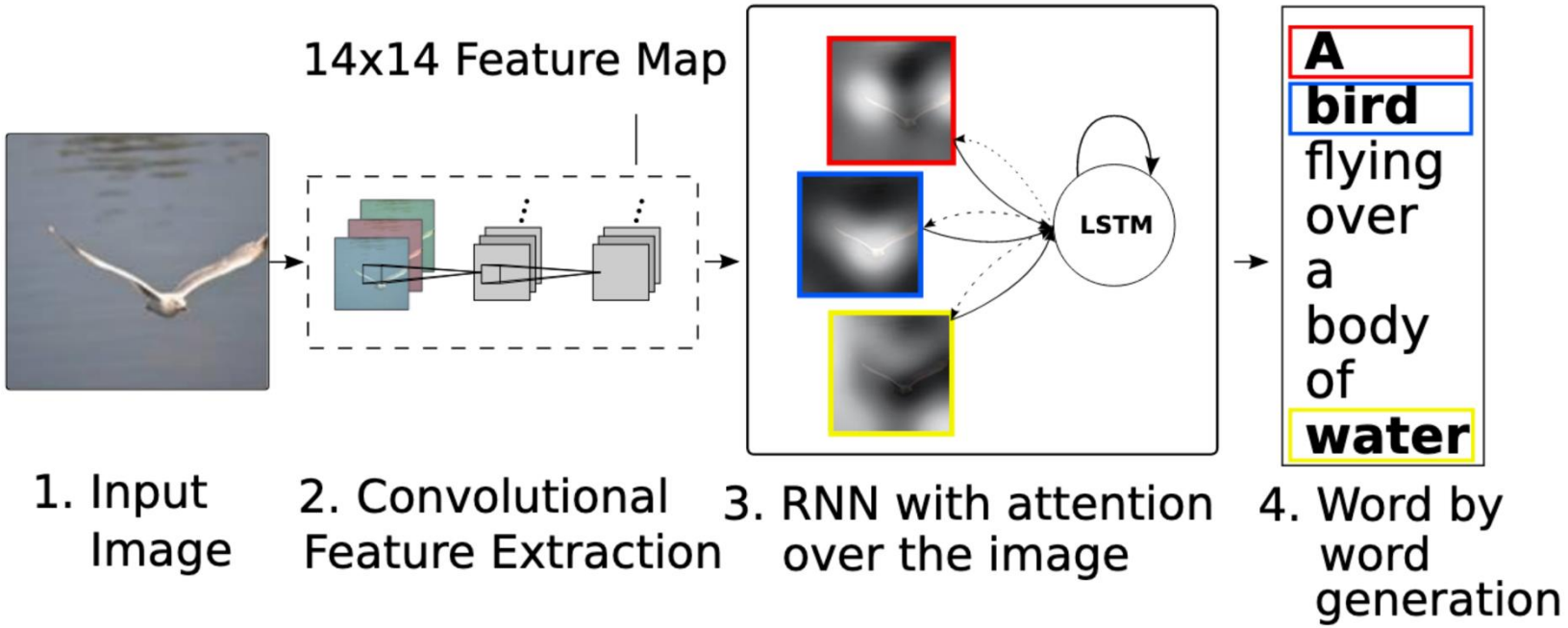
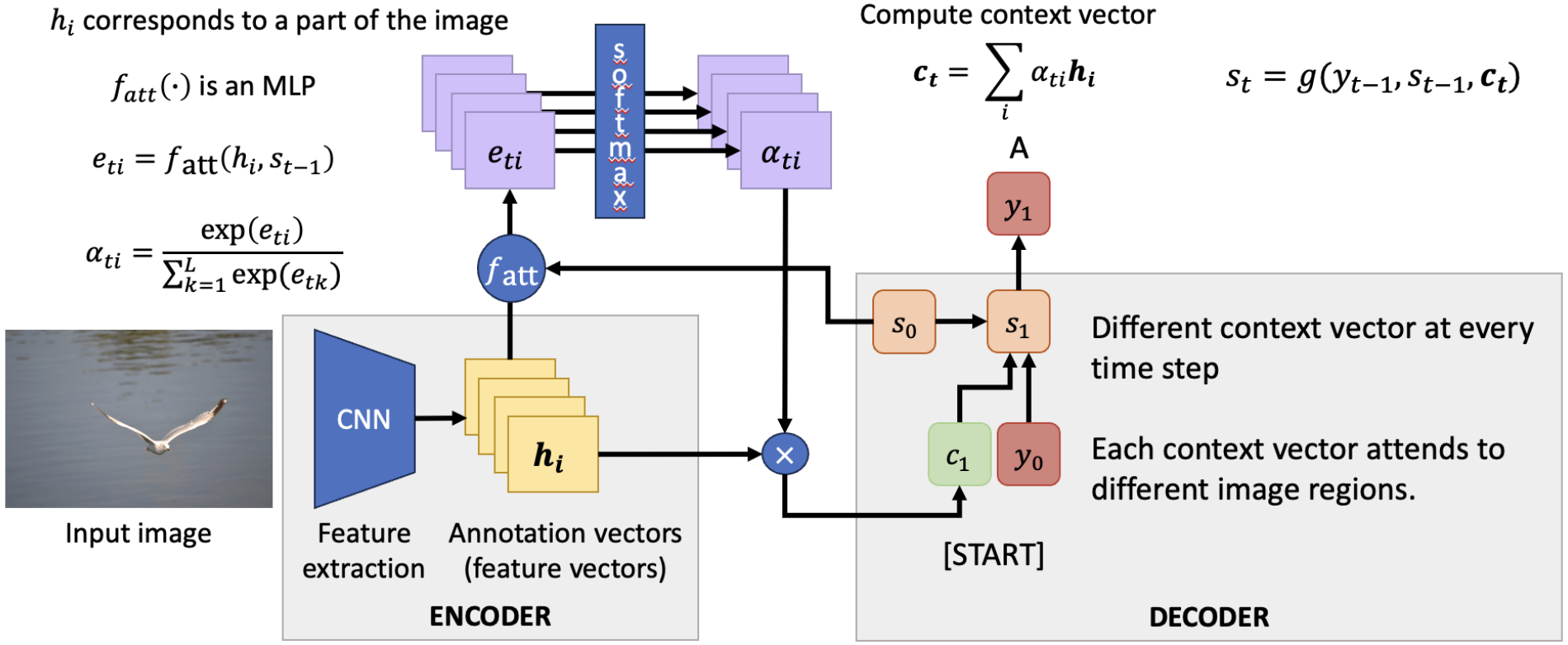
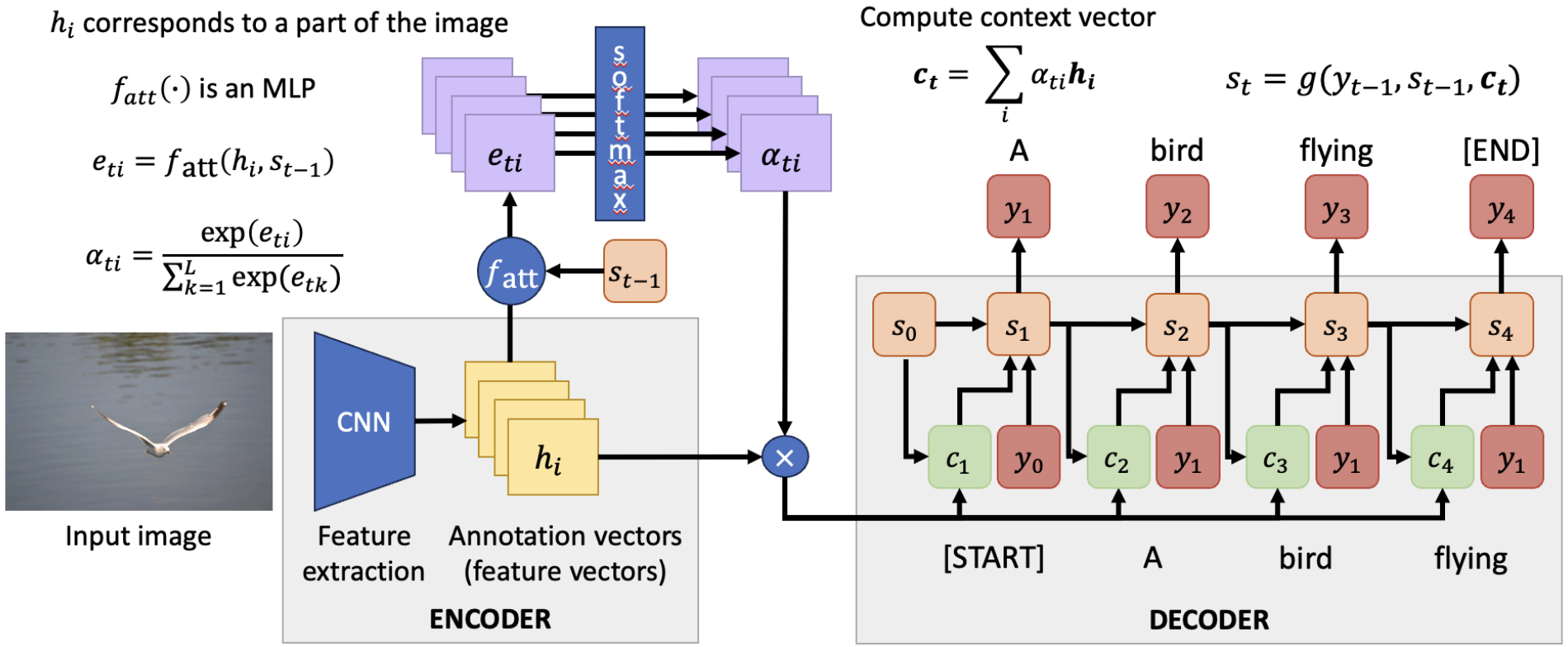


IMAGE CAPTIONING WITH VISUAL ATTENTION



K. Xu *et al.*, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention," in *PMLR*, 2015, pp. 2048–2057.

IMAGE CAPTIONING WITH VISUAL ATTENTION

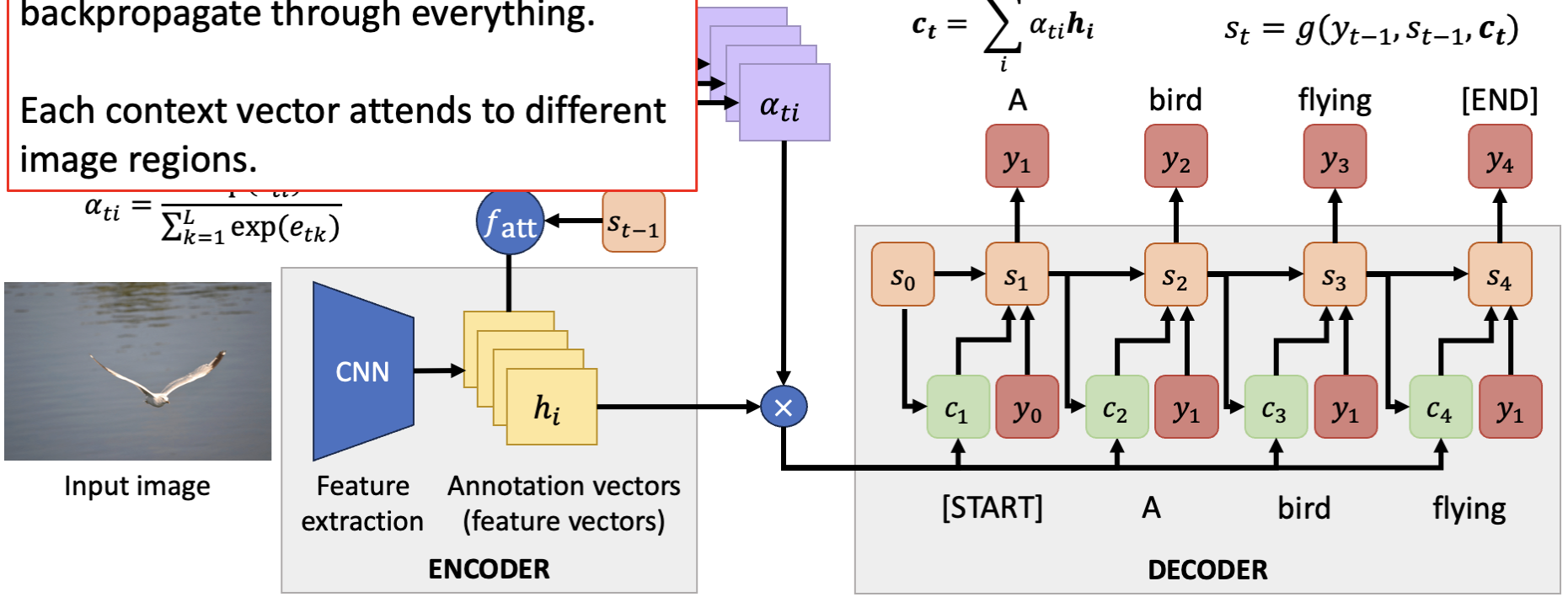


K. Xu *et al.*, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention," in *PMLR*, 2015, pp. 2048–2057.

IMAGE CAPTIONING WITH VISUAL ATTENTION

All steps are **differentiable**, so we can backpropagate through everything.

Each context vector attends to different image regions.



K. Xu *et al.*, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention," in *PMLR*, 2015, pp. 2048–2057.

IMAGE CAPTIONING WITH VISUAL ATTENTION

- Visualization of the attention for each generated word
  - Gives insight to “where” and “what” the attention focused on when generating each word

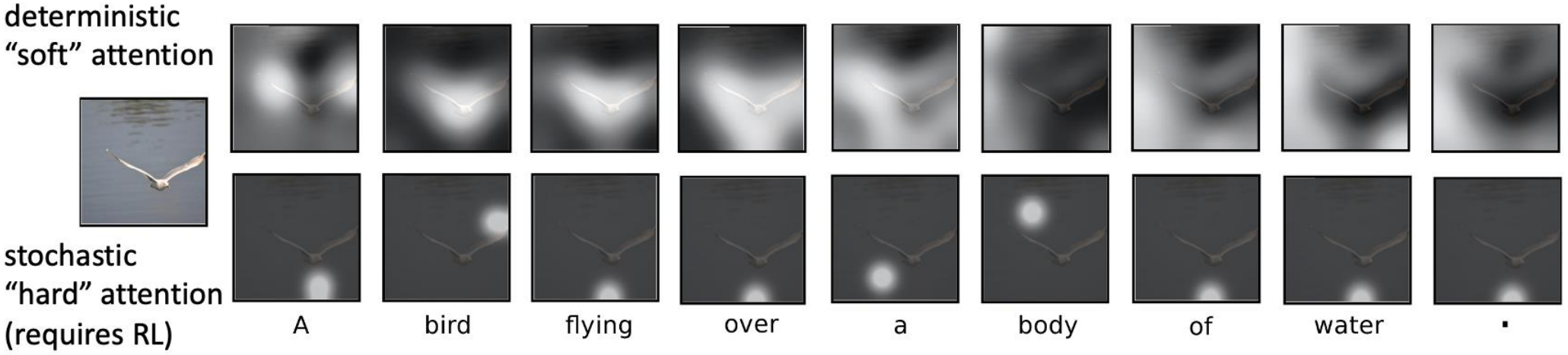


IMAGE CAPTIONING WITH VISUAL ATTENTION

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.

K. Xu *et al.*, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention," in *PMLR*, 2015, pp. 2048–2057.

RESOURCES

- https://medium.com/@deepeshrishu09/automatic-image-captioning-with-pytorch-cf576c98d319
- https://www.kaggle.com/code/mdteach/image-captioning-with-attention-pytorch
- Tensorflow: https://www.tensorflow.org/text/tutorials/image_captioning

ATTENTION IS ALL YOU NEED

- **Key Idea:**
  - Decouple attention from RNNs
  - Use self-attention to make this efficient
- **Contributions:**
  - Multi-head attention
  - Transformer architecture
- Highly impactful (as we'll touch on later)

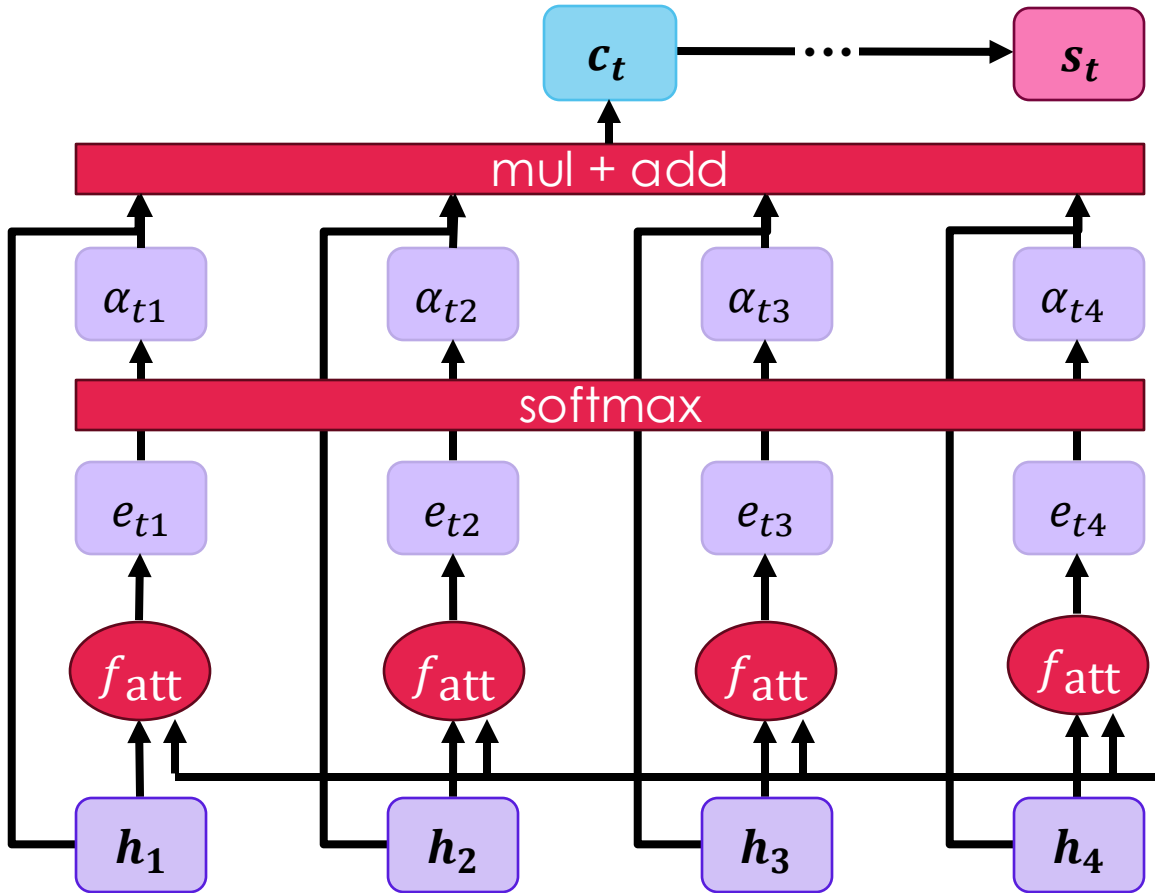
Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com
Noam Shazeer* Google Brain noam@google.com
Niki Parmar* Google Research nikip@google.com
Jakob Uszkoreit* Google Research usz@google.com
Llion Jones* Google Research llion@google.com
Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu
Łukasz Kaiser* Google Brain lukaszkaiser@google.com
Illia Polosukhin* ‡ illia.polosukhin@gmail.com

A. Vaswani *et al.*, "Attention is All you Need," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

ATTENTION WE'VE SEEN SO FAR

- Known as **"additive" recurrent attention** (type of encoder-decoder attention)



Context vector

$$\boldsymbol{c_t} = \sum_i \alpha_{ti} \boldsymbol{h_i}$$

Attention weights

$$\boldsymbol{\alpha_t} = \text{softmax}(\boldsymbol{e_t})$$

Alignment scores

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

f_{att}: simple feedforward network (e.g. MLP)

ISSUES WITH RECURRENT ATTENTION

- **Scalability issues:** Performance degrades as the distance between words increases
- **Parallelization limitations:** Recurrent processes lacks ability to be parallelized
- **Memory constraints**
  - RNNs have limited memory and struggle with long-range dependencies
  - Diluted impact of earlier elements on output as sequence progresses

- **Potential solution:** Decouple attention from RNNs
  - How? Separate the attention mechanism into smaller, self-contained components

DECOUPLING FROM RNNS

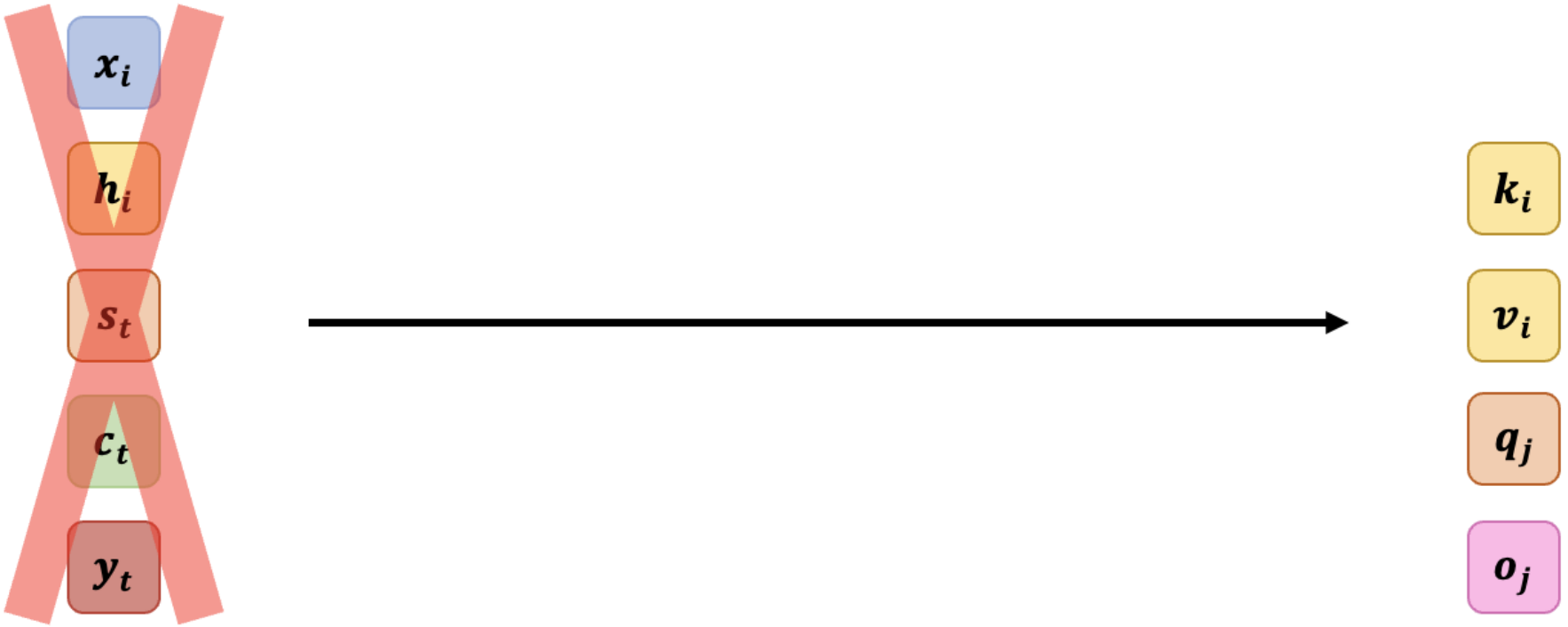
- **Recall:** Attention determines the **importance of elements** to be passed forward in the model.
  - These weights lets the model pay **attention** to the **most significant parts**

- **Objective**: A more general attention mechanism not confined to RNNs
- We need a modified procedure to:
  1. Determine weights based on context that indicate the elements to attend to
  2. Apply these weights to enhance attended features

RNN NOTATIONS

x_i	Input for position i in source sequence
h_i	Hidden states for position i in source sequence
s_t	Hidden states for position t in target sequence
c_t	Context vector for position t in target sequence
y_t	Output for position t in target sequence

NEW NOTATION

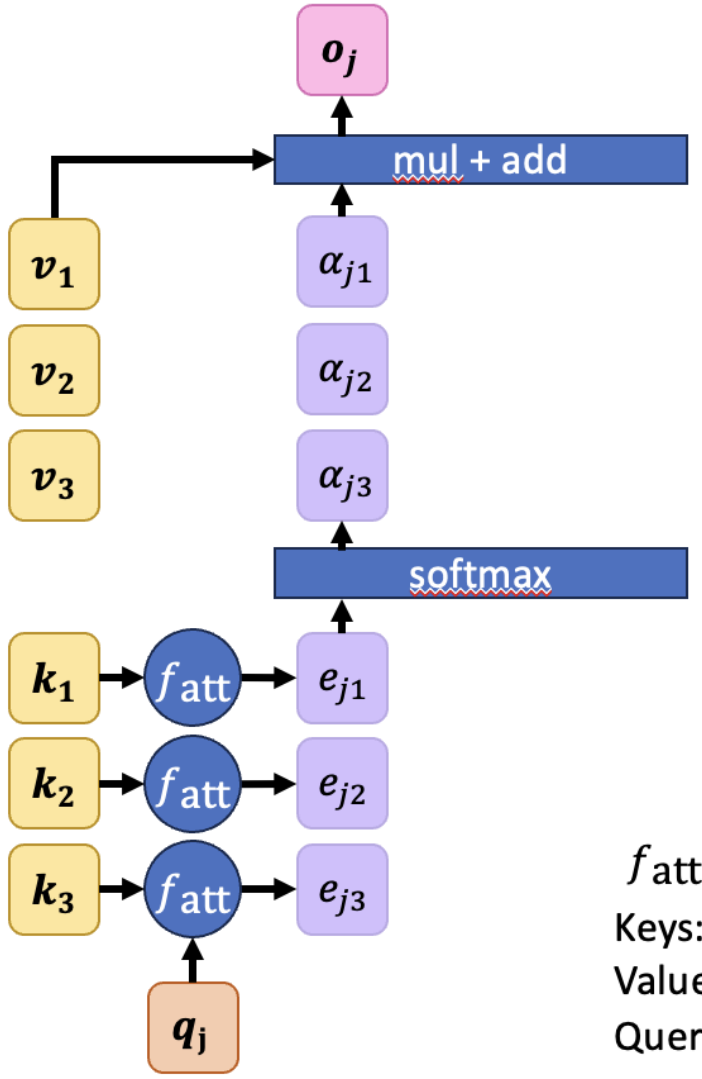


NEW NOTATION

k_i	**Key** vector for position i in an arbitrary sequence
v_i	**Value** vector for position i in an arbitrary sequence
q_j	**Query** vector for position j in a (same/different) arbitrary sequence
o_j	**Output** vector corresponding to position j

A GENERAL ATTENTION MECHANISM

$Attention(\boldsymbol{q}, \boldsymbol{k}, \boldsymbol{v})$



Attention

Alignment

Output vectors

$$\boldsymbol{o_j} = \sum_i \alpha_{ji} \boldsymbol{v_i} \quad \in \mathbb{R}^{d_k}$$

Attention weights

$$\boldsymbol{\alpha_j} = \text{softmax}(\boldsymbol{e_j})$$

Alignment scores

$$e_{ij} = f_{\text{att}}(\boldsymbol{q_j}, \boldsymbol{k_i})$$

Goal: find the "alignment" or "compatibility" of keys with a query to scale values

$f_{\text{att}}(\cdot)$

Keys: $\boldsymbol{k_i} \in \mathbb{R}^{d_k}$

Values: $\boldsymbol{v_i} \in \mathbb{R}^{d_k}$

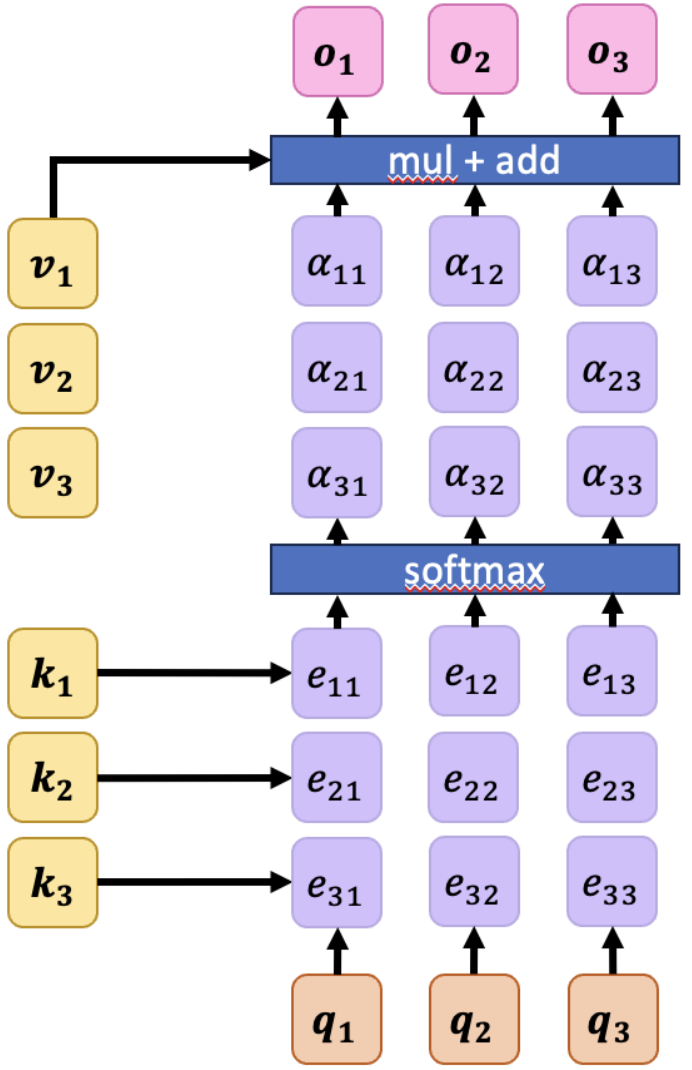
Query: $\boldsymbol{q_j} \in \mathbb{R}^{d_q}$

$i \in \{1, \ldots, N\}$

$j \in \{1, \ldots, T\}$

A GENERAL ATTENTION MECHANISM

$Attention(\boldsymbol{q}, \boldsymbol{k}, \boldsymbol{v})$



Attention

Alignment

Output vectors
$\boldsymbol{o_j} \in \mathbb{R}^{d_k}$,
for j in $\{1, \ldots, T\}$

Attention weights
$\boldsymbol{\alpha_j} = \text{softmax}(\boldsymbol{e_j})$

Alignment scores
$e_{ij} = f_{\text{att}}(\boldsymbol{q_j}, \boldsymbol{k_i})$

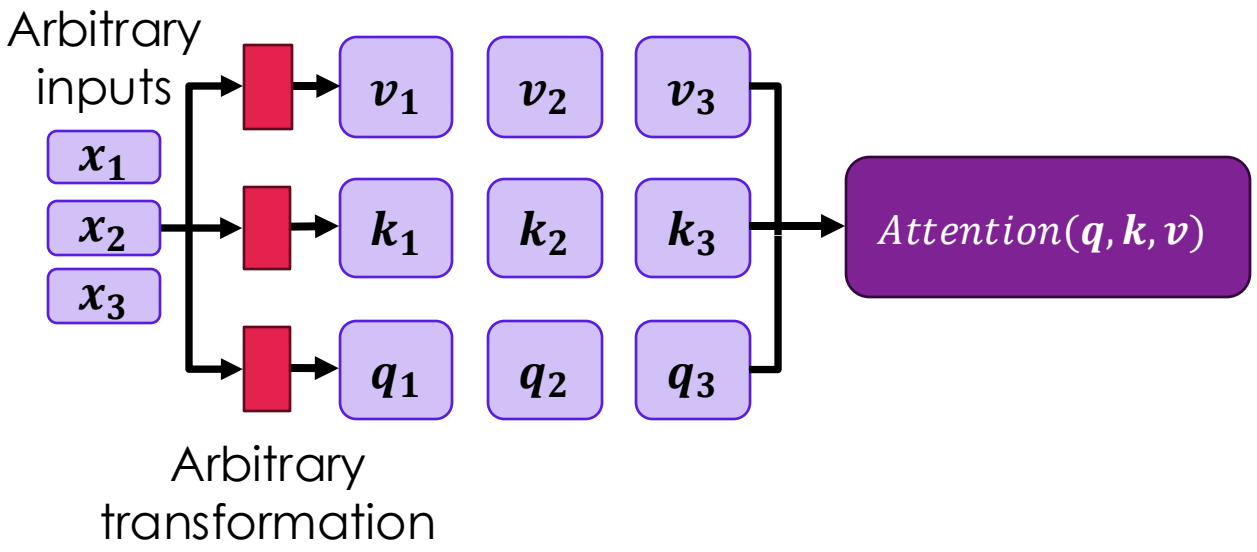
$f_{\text{att}}(\cdot)$

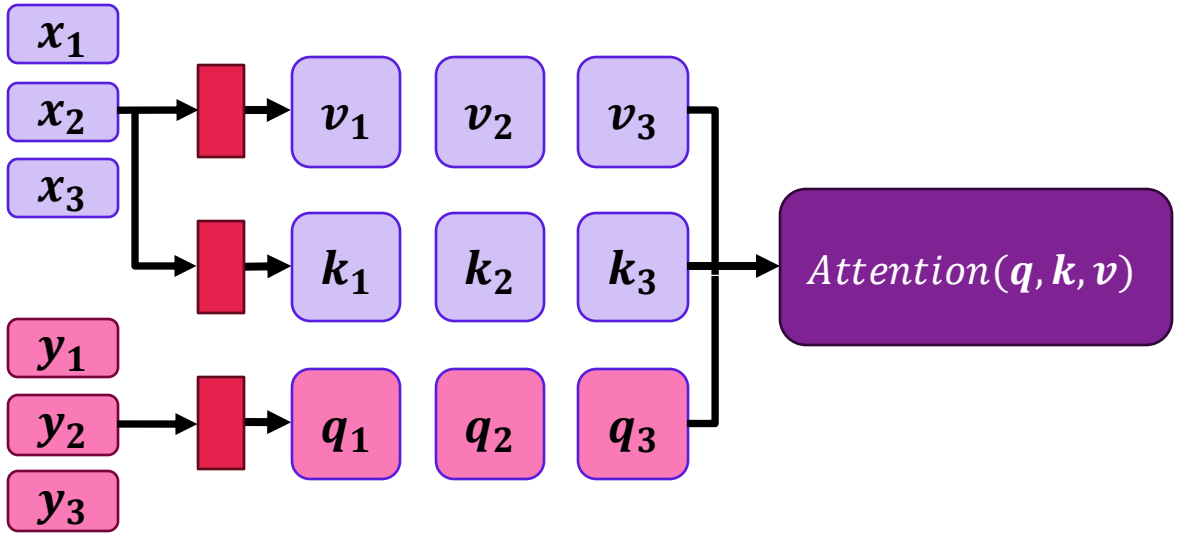
Keys: $\boldsymbol{k_i} \in \mathbb{R}^{d_k}$ $i \in \{1, \ldots, N\}$
Values: $\boldsymbol{v_i} \in \mathbb{R}^{d_k}$ j $\in \{1, \ldots, T\}$
Query: $\boldsymbol{q_j} \in \mathbb{R}^{d_q}$

Goal: find the "alignment" or "compatibility" between keys and queries to scale values

APPLYING THE ATTENTION MECHANISM

- **Self-Attention**
  - Keys, values, and queries are all derived from the same source.
- **Cross-Attention**
  - Keys-values and queries are derived from separate sources.





ATTENTION MECHANISM

To use a **decoupled attention mechanism**, it is implemented with properties:

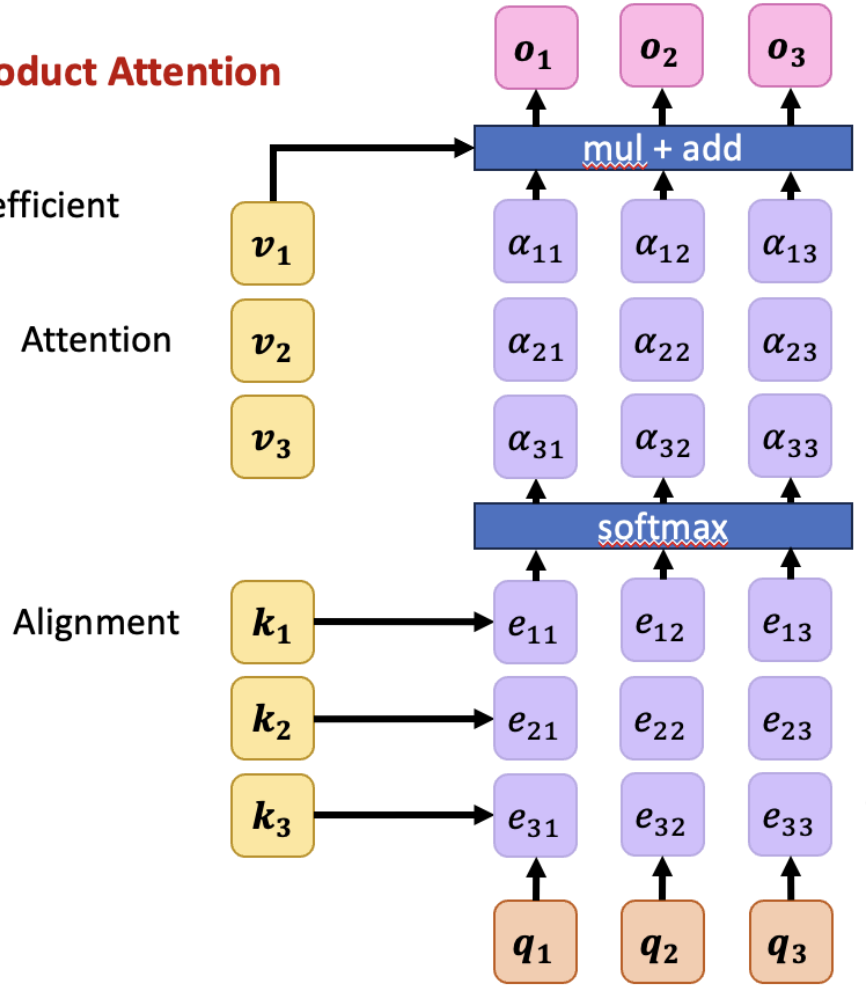
1. $f_{\text{att}}(\cdot) =$ **scaled dot-product attention**
   - Good representation of compatibility
   - Fast and interpretable computation
   - Parallelizable evaluation across all queries (can leverage GPUs)
   - Scaled dot-products for stable softmax gradients in high dimensions (prevents large magnitudes)

2. Imposed a **common dimension** for keys, values, and queries
   - Requirement for dot-product
   - Simplifies architecture with predictable attention output shape
   - Provides consistent hidden state dimensions for easier model analysis

ATTENTION MECHANISM

Scaled Dot-Product Attention

- Faster
- More space-efficient



Attention

Alignment

Output vectors

$o_j \in \mathbb{R}^{d_k}$

Attention weights

$\alpha_j = \text{softmax}(e_j)$

Alignment scores

$$e_{ij} = f_{\text{att}}(q_j, k_i) = \frac{q_j \cdot k_i}{\sqrt{d_k}}$$

$f_{\text{att}}(\cdot)$ = scaled dot-product

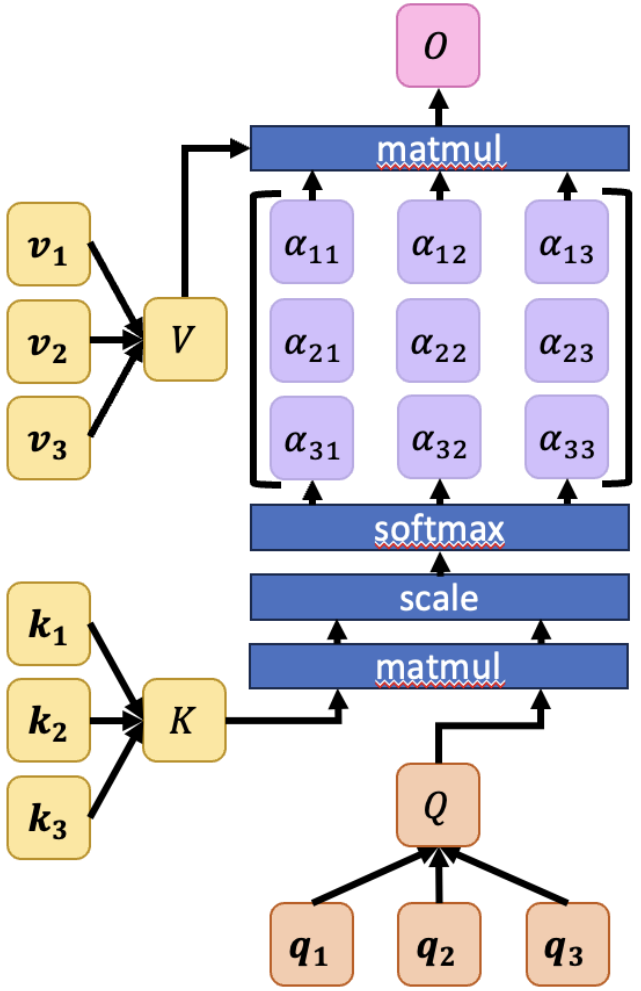
Keys: $k_i \in \mathbb{R}^{d_k}$
Values: $v_i \in \mathbb{R}^{d_k}$
Query: $q_j \in \mathbb{R}^{d_k}$

} Same dimension

ATTENTION MECHANISM

Calculate dot-products in **parallel** with **matrix multiplication**

- High concurrency on modern hardware (GPUs)
- Independently calculates each query



Matrix of outputs $O \in \mathbb{R}^{T \times d_k}$

Matrix containing attention weights

$$A[j,i] = \text{softmax}\left(\frac{Q \cdot K^T}{\sqrt{d_k}}\right)$$

Alignment scores

$e_{ji} = f_{\text{att}}(\boldsymbol{q_j}, \boldsymbol{k_i}) = \frac{Q \cdot K^T}{\sqrt{d_k}}$ [j, i]

Alignment

$f_{\text{att}}(\cdot)$ = scaled dot-product

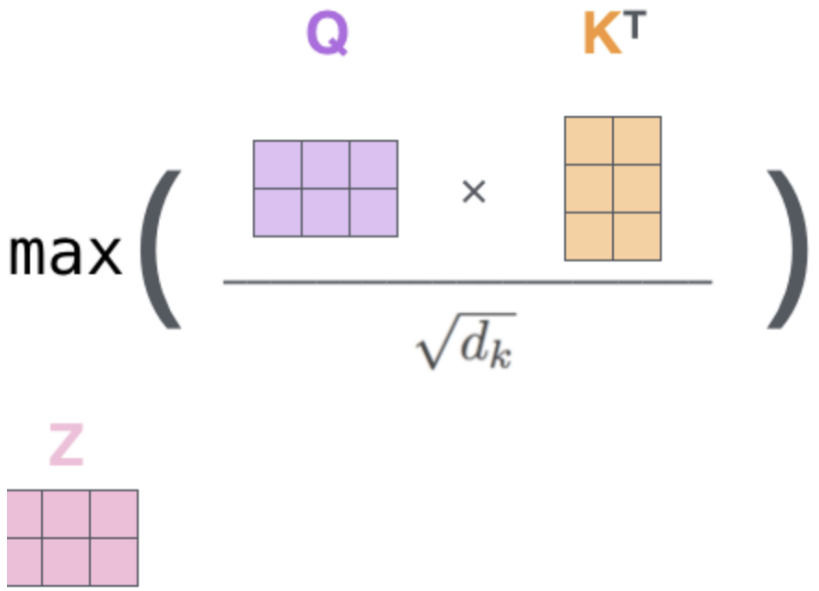
Keys: $K \in \mathbb{R}^{N \times d_k}$

Values: $V \in \mathbb{R}^{N \times d_k}$

Query: $Q \in \mathbb{R}^{T \times d_k}$

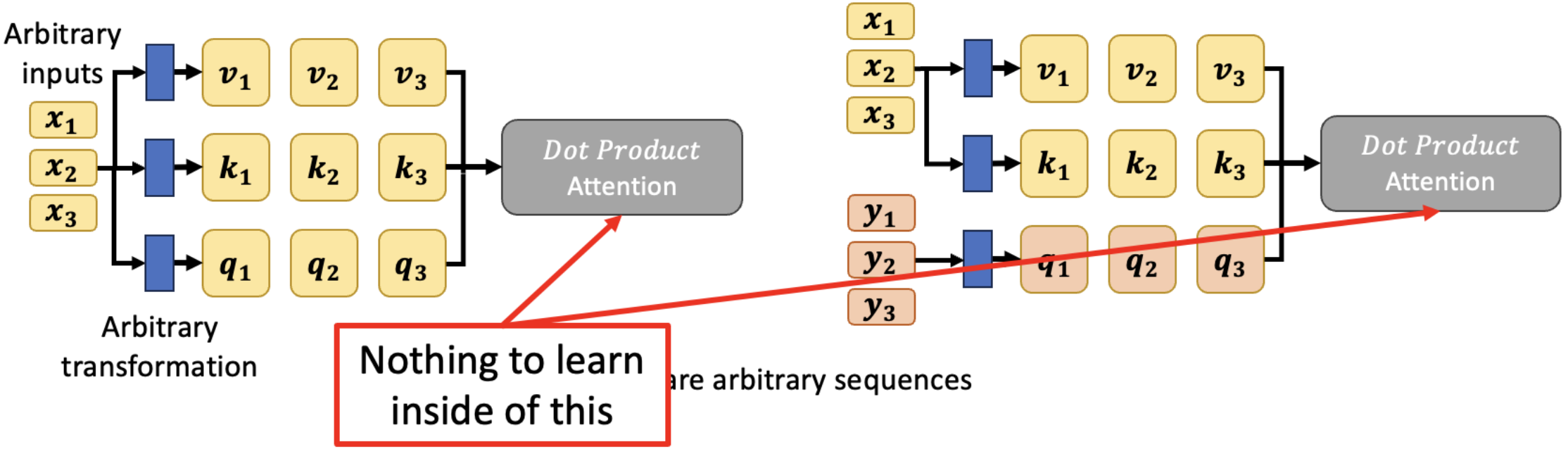
TRANSFORMERS

- How do we get Q, K, and V?
- What are we learning?
- Is this parametric or non-parametric?



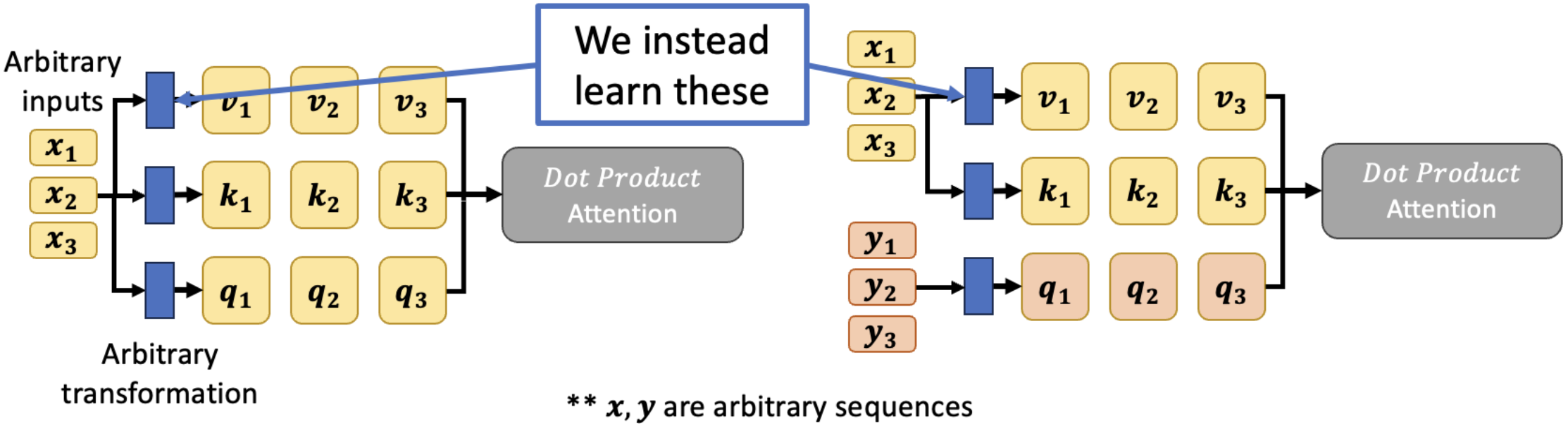
TRANSFORMER ATTENTION

- **Self-Attention**
  - Keys, values, and queries are all derived from the same source.
- **Cross-Attention**
  - Keys-values and queries are derived from separate sources.



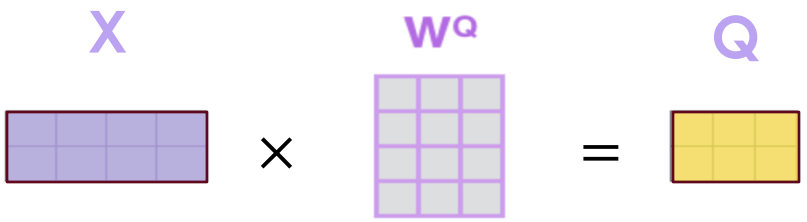
TRANSFORMER ATTENTION

- **Self-Attention**
  - Keys, values, and queries are all derived from the same source.
- **Cross-Attention**
  - Keys-values and queries are derived from separate sources.



LEARNING TRANSFORMER ATTENTION

Self-Attention



Cross-Attention

RESOURCES

- https://pytorch.org/docs/stable/generated/torch.nn.Transformer.html
- https://www.datacamp.com/tutorial/building-a-transformer-with-py-torch