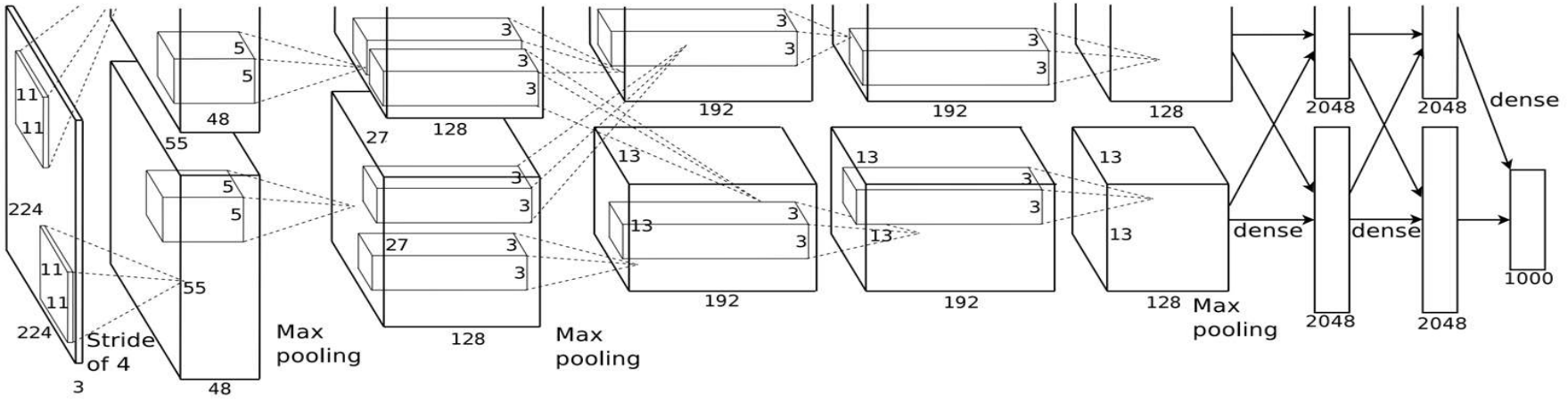
Deep NN


11
11
224
11
11
224
3
Stride
of 4
5
5
48
55
5
5
48
55
3
3
3
128
27
3
3
3
27
3
3
128
Max
pooling
Max
pooling
3
3
192
3
3
192
128
13
13
3
3
13
192
13
3
3
13
192
13
13
128
Max
pooling
dense
dense
2048
2048
2048
2048
dense
1000

2018 Turing Award — by ACM @ 2019/3/27

AWARDS & RECOGNITION

ACM Announces 2018 Turing Award Recipients

ACM has named Yoshua Bengio of the University of Montreal, Geoffrey Hinton of Google, and Yann LeCun of New York University recipients of the 2018 ACM A.M. Turing Award for conceptual and engineering breakthroughs that have made deep neural networks a critical component of computing. Working independently and together, Hinton, LeCun and Bengio developed conceptual foundations for the field, identified surprising phenomena through experiments, and contributed engineering advances that demonstrated the practical advantages of deep neural networks.

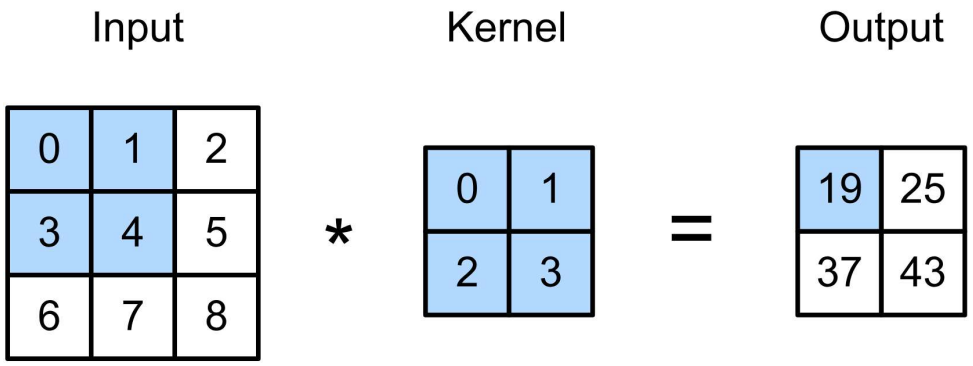
Outline

- Convolutional Neural Network (CNN)
- Recurrent Neural Network (RNN)
- Transformer
- Deep Learning Frameworks

Outline

- Convolutional Neural Network (CNN)
  - Convolution, Padding, Stride, Pooling
  - CNN Progress
- Recurrent Neural Network (RNN)
- Transformer
- Deep Learning Frameworks

2-D Cross Correlation

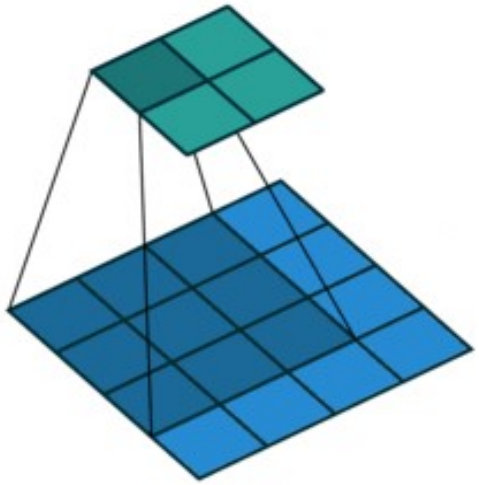


$$0 \times 0 + 1 \times 1 + 3 \times 2 + 4 \times 3 = 19,$$
$$1 \times 0 + 2 \times 1 + 4 \times 2 + 5 \times 3 = 25,$$
$$3 \times 0 + 4 \times 1 + 6 \times 2 + 7 \times 3 = 37,$$
$$4 \times 0 + 5 \times 1 + 7 \times 2 + 8 \times 3 = 43.$$

2-D Convolution Layer

$$\mathbf{Y} = \mathbf{X} \star \mathbf{W} + b$$

W and *b* are learnable parameters, b: scalar bias



(vdumoulin@ Github)

$\mathbf{X}$: $n_h \times n_w$ input matrix

$\mathbf{W}$: $k_h \times k_w$ kernel matrix

$\mathbf{Y}$: $(n_h - k_h + 1) \times (n_w - k_w + 1)$ output matrix feature map

0	1	2
3	4	5
6	7	8

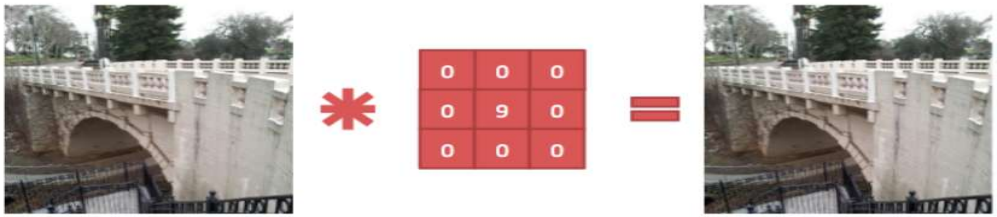
$*$

0	1
2	3

$=$

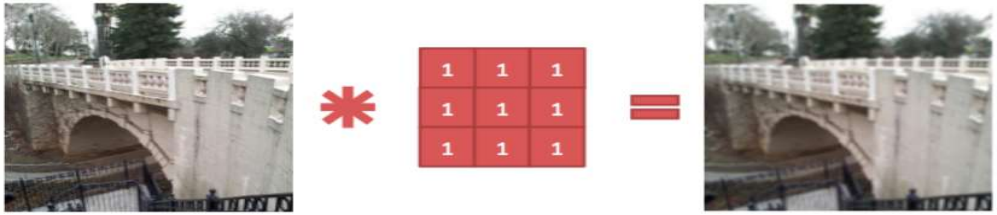
19	25
37	43

$$0 \times 0 + 1 \times 1 + 3 \times 2 + 4 \times 3 = 19,$$
$$1 \times 0 + 2 \times 1 + 4 \times 2 + 5 \times 3 = 25,$$
$$3 \times 0 + 4 \times 1 + 6 \times 2 + 7 \times 3 = 37,$$
$$4 \times 0 + 5 \times 1 + 7 \times 2 + 8 \times 3 = 43.$$

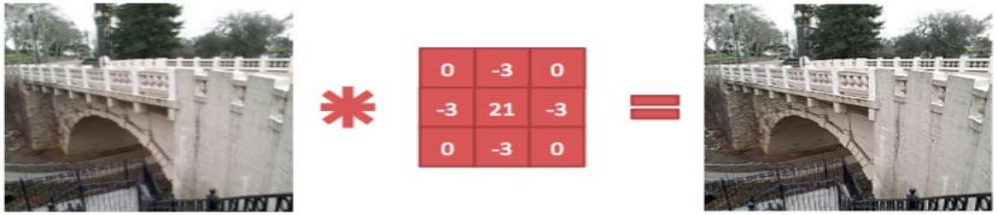


(a) Identity kernel

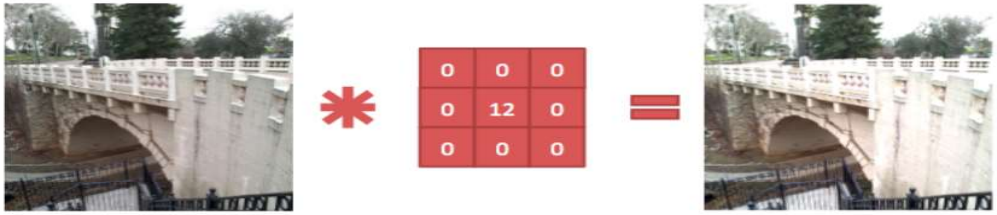
(b) Edge detection kernel



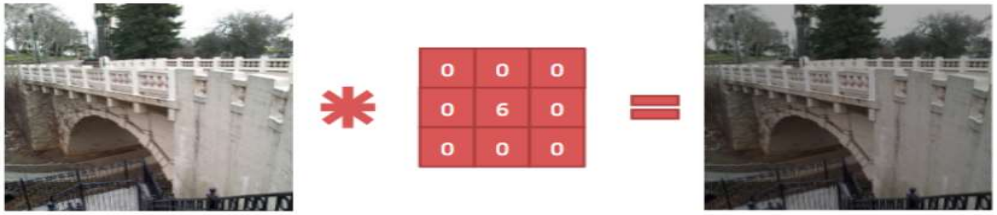
(c) Blur kernel



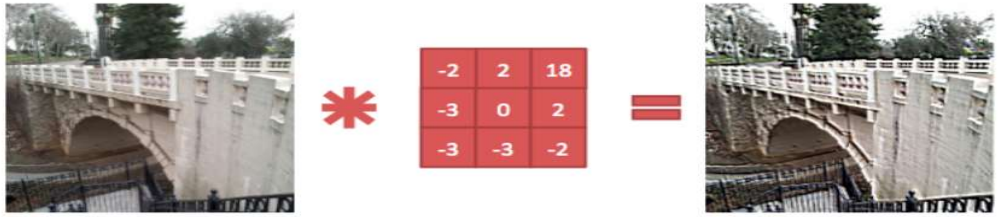
(d) Sharpen kernel



(e) Lighten kernel



(f) Darken kernel

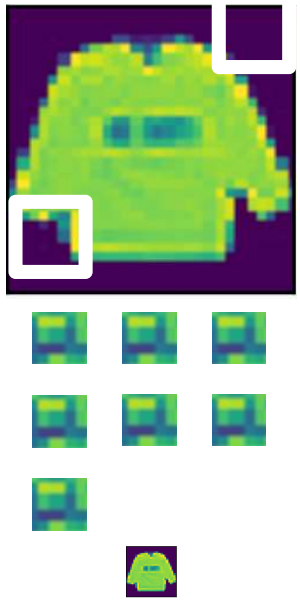


(g) Random kernel 1

(h) Random kernel 2

- Given a 32 x 32 input image
- Apply convolutional layer with 5 x 5 kernel
  - 28 x 28 output with 1 layer
  - 4 x 4 output with 7 layers
- **Shape decreases** faster with larger kernels
  - Shape reduces from $n_h \times n_w$ to

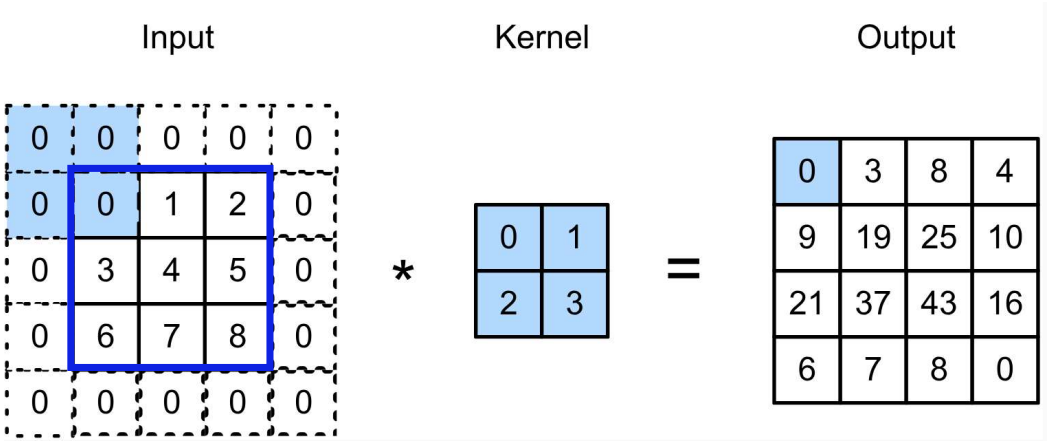
$$(n_h - k_h + 1) \times (n_w - k_w + 1)$$

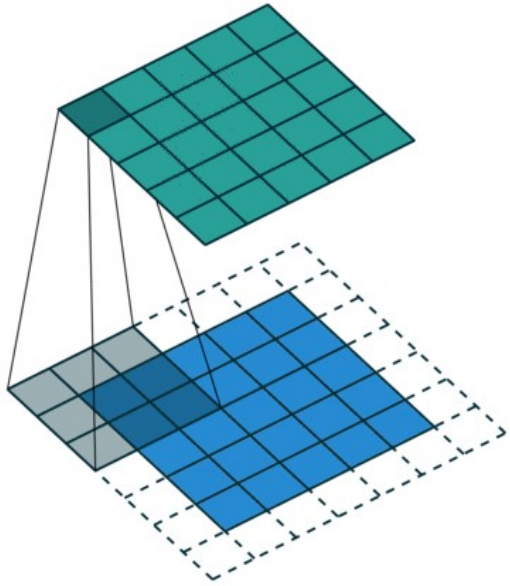


Padding

- Padding adds rows/columns around input
- Padding p_h rows and p_w columns, output shape will be

$$(n_h - k_h + p_h + 1) \times (n_w - k_w + p_w + 1)$$

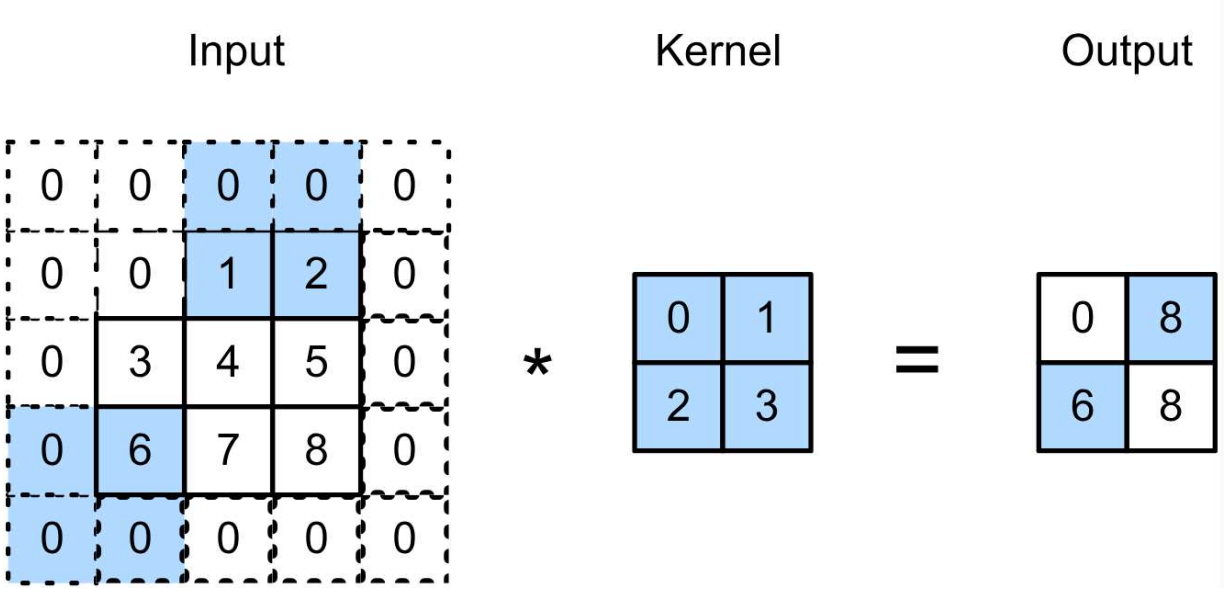




$$0 \times 0 + 0 \times 1 + 0 \times 2 + 0 \times 3 = 0$$

Stride

- Stride is the #rows/#columns per slide.
- Given stride *Sh* for the height and stride *Sw* for the width, the output shape is $\lfloor (n_h - k_h + p_h + s_h)/s_h \rfloor \times \lfloor (n_w - k_w + p_w + s_w)/s_w \rfloor$



Strides of 3 and 2
for height and width

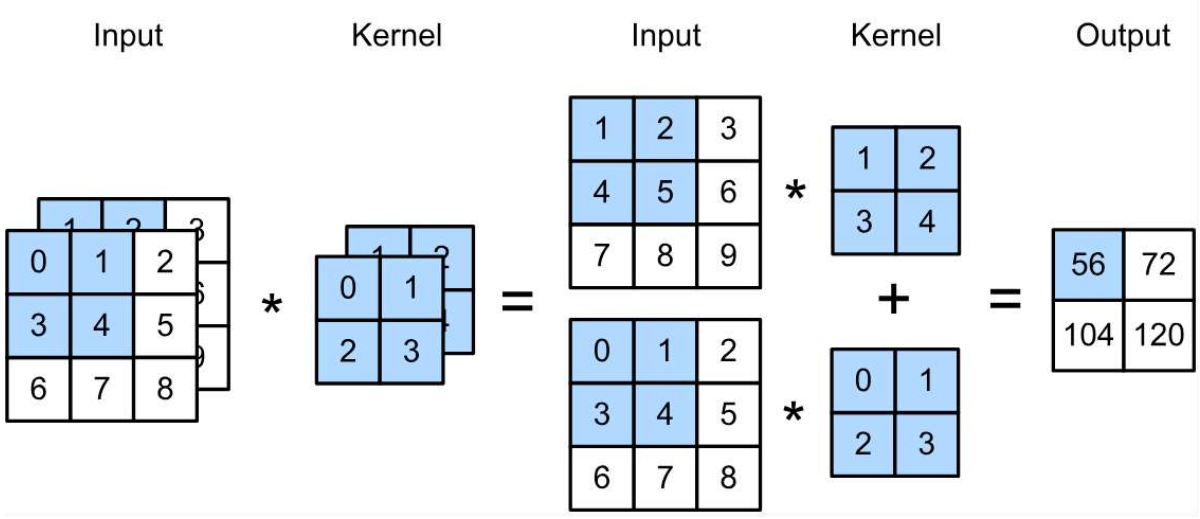
$0 \times 0 + 0 \times 1 + 1 \times 2 + 2 \times 3 = 8$
$0 \times 0 + 6 \times 1 + 0 \times 2 + 0 \times 3 = 6$

Multiple Input Channels

- Color image may have three RGB channels
- Converting to grayscale loses information

Multiple Input Channels

- Have a kernel for each channel, and then sum results over channels.



$$(1 \times 1 + 2 \times 2 + 4 \times 3 + 5 \times 4) + (0 \times 0 + 1 \times 1 + 3 \times 2 + 4 \times 3) = 56$$

* No matter how many inputs channels, we always get single output channel.

Input $\mathbf{X}$: $c_i \times n_h \times n_w$

Kernel $\mathbf{W}$: $c_i \times k_h \times k_w$

Output $\mathbf{Y}$: $m_h \times m_w$

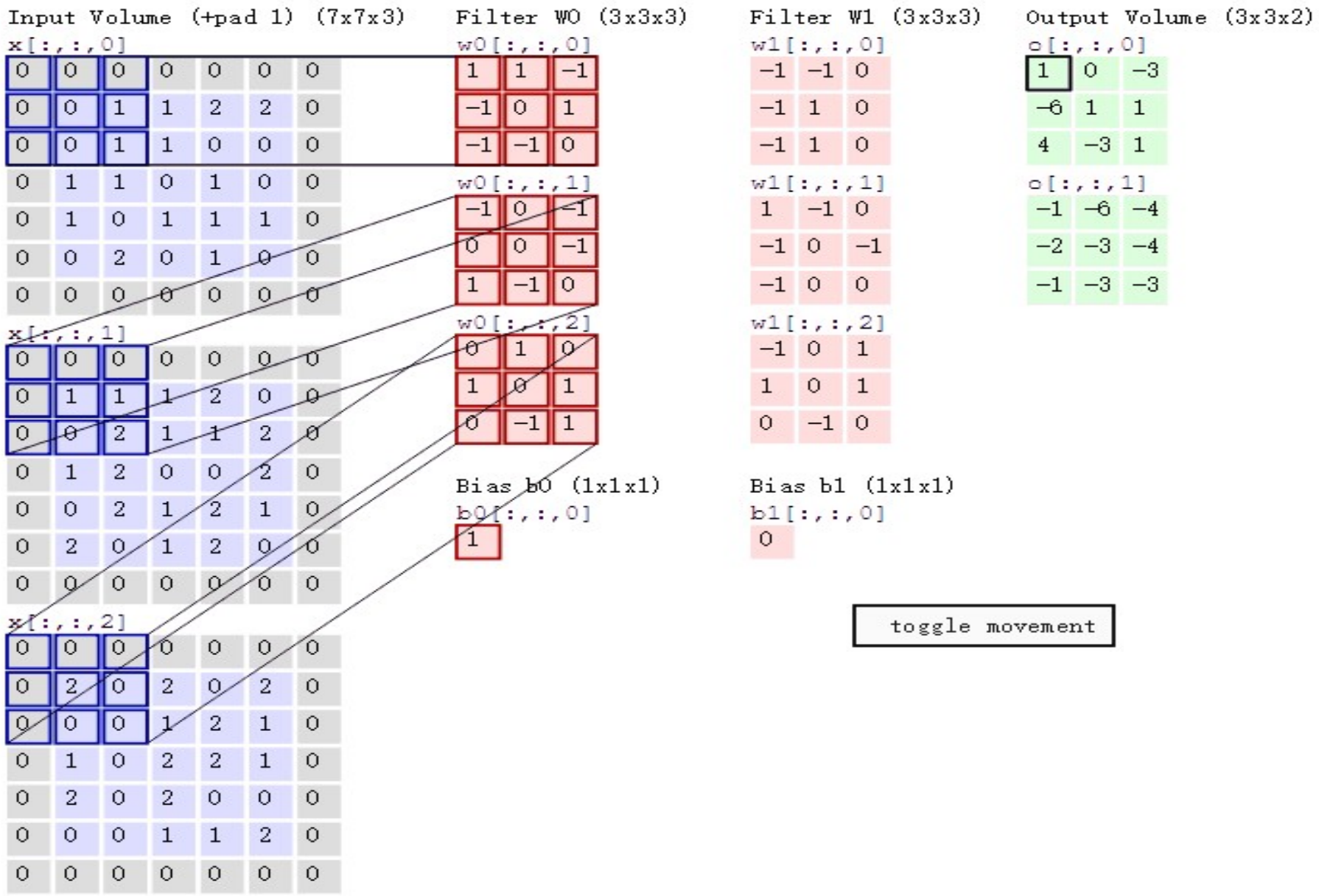
$$\mathbf{Y} = \sum_{i=0}^{c_i} \mathbf{X}_{i,:,:} \star \mathbf{W}_{i,:,:}$$

Multiple Output Channels

- We can have multiple 3-D kernels, each one generates a output channel.
  - Input $\mathbf{X}$: $c_i \times n_h \times n_w$
  - Kernel $\mathbf{W}$: $c_o \times c_i \times k_h \times k_w$
  - Output $\mathbf{Y}$: $c_o \times m_h \times m_w$

$$\mathbf{Y}_{i,:,:} = \mathbf{X} \star \mathbf{W}_{i,:,:,:}$$

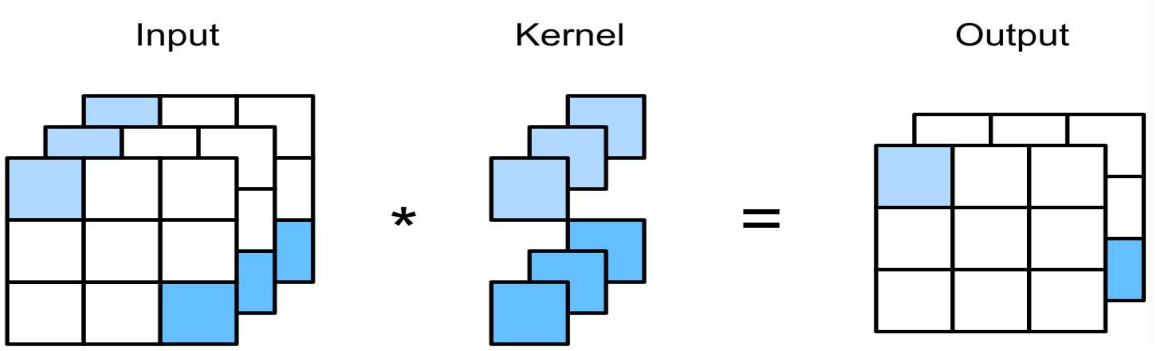
$$for\ i = 1, \ldots, c_o$$


Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 0 1 1 2 2 0
0 0 1 1 0 0 0
0 1 1 0 1 0 0
0 1 0 1 1 1 0
0 0 2 0 1 0 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 1 1 2 0 0
0 0 2 1 1 2 0
0 1 2 0 0 2 0
0 0 2 1 2 1 0
0 2 0 1 2 0 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 2 0 2 0 2 0
0 0 0 1 2 1 0
0 1 0 2 2 1 0
0 2 0 2 0 0 0
0 0 0 1 1 2 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
1 1 -1
-1 0 1
-1 -1 0
w0[:,:,1]
-1 0 -1
0 0 -1
1 -1 0
w0[:,:,2]
0 1 0
1 0 1
0 -1 1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 -1 0
-1 1 0
-1 1 0
w1[:,:,1]
1 -1 0
-1 0 -1
-1 0 0
w1[:,:,2]
-1 0 1
1 0 1
0 -1 0
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
1 0 -3
-6 1 1
4 -3 1
o[:,:,1]
-1 -6 -4
-2 -3 -4
-1 -3 -3
toggle movement

1 x 1 Convolutional Layer

$$k_h = k_w = 1$$

It doesn't recognize spatial patterns, but fuse channels.



Equal to a dense layer with $n_h n_w \times c_i$ input and $c_o \times c_i$ weight.

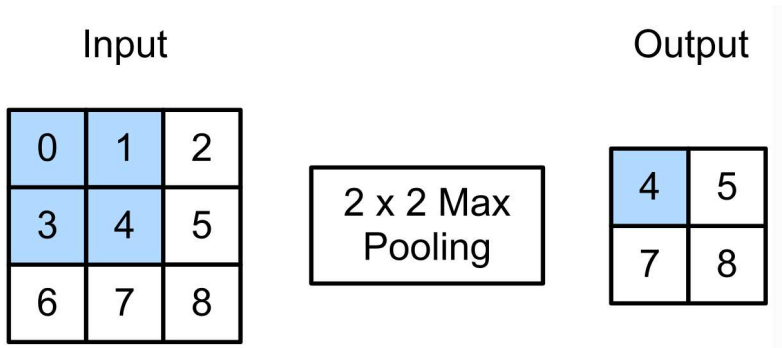
Pooling

- Why pooling:
  - reduce the dimensionality when processing data
  - Alleviate the excessive sensitivity of the convolutional layer to location
    - Lighting, object positions, scales, appearance vary among images
    - Convolution is sensitive to position
      - e.g. Detect vertical edges

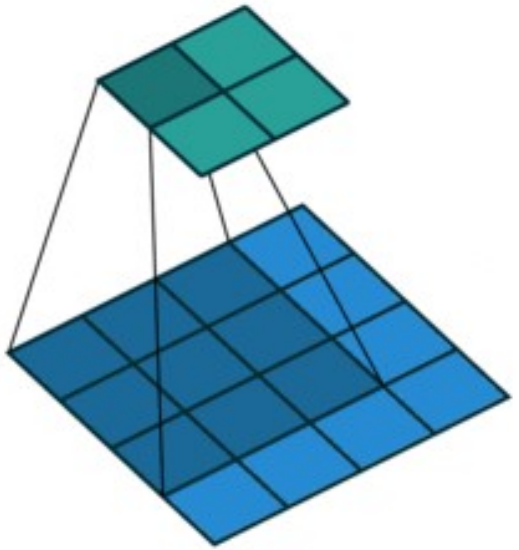
```
X [[1. 1. 0. 0. 0.
   [1. 1. 0. 0. 0.
   [1. 1. 0. 0. 0.
   [1. 1. 0. 0. 0.

Y [[ 0.  1.  0.  0.
   [ 0.  1.  0.  0.
   [ 0.  1.  0.  0.
   [ 0.  1.  0.  0.
```

2-D Max Pooling

- Returns the maximal value in the sliding window

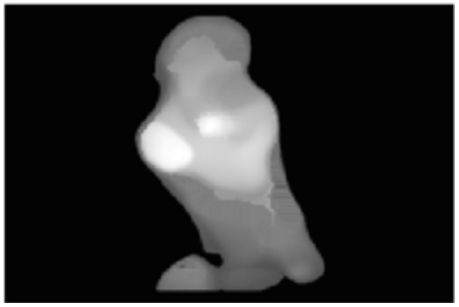


$$max(0,1,3,4) = 4$$



Average Pooling

- Max pooling: the strongest pattern signal in a window
- Average pooling: replace max with mean in max pooling
  - The average signal strength in a window



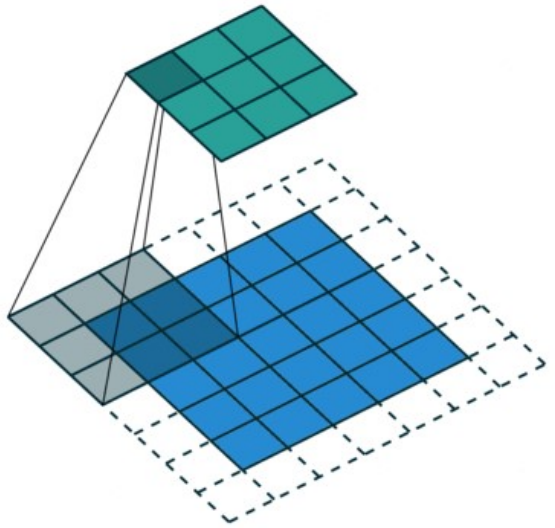
Max pooling

Average pooling

Padding, Stride, and Multiple Channels

- Pooling layers have similar padding and stride as convolutional layers
- No learnable parameters
- Apply pooling for each input channel to obtain the corresponding output channel

  #output channels = #input channels



Outline

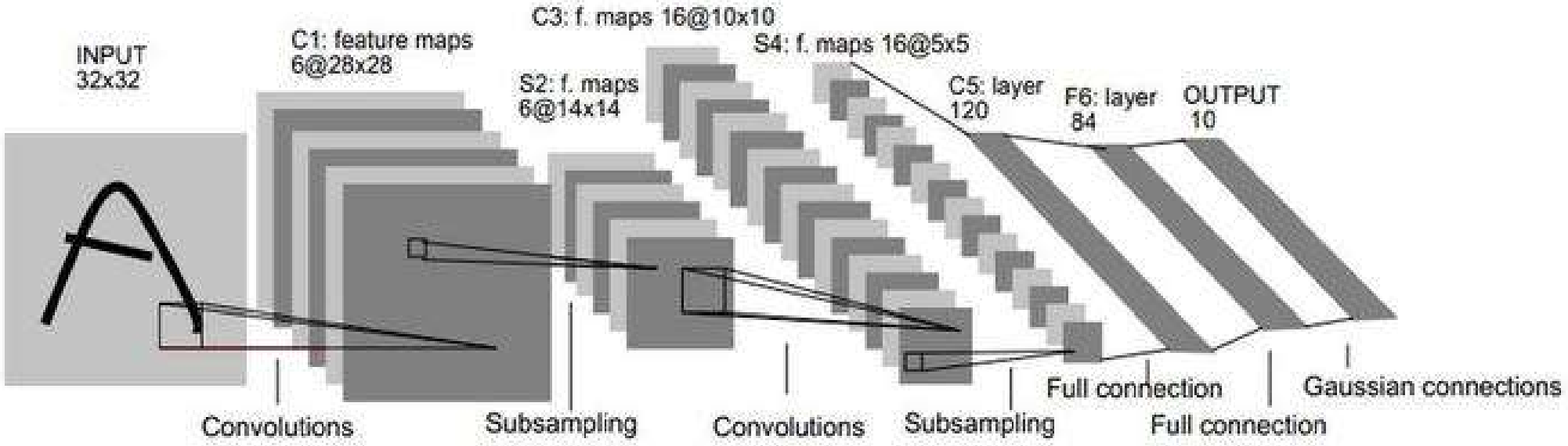
- Convolutional Neural Network (CNN)
  - Convolution, Padding, Stride, Pooling
  - CNN Progress
- Recurrent Neural Network (RNN)
- Transformer
- Deep Learning Frameworks

CNN Progress

- LeNet
  - 2 convolution + pooling layers ； 2 hidden dense hidden layers
- AlexNet(2012)
  - Bigger and deeper LeNet； ReLu, Dropout, preprocessing
- NiN (2013)
  - 1x1 convolutions + global pooling instead of dense
- VGG（2014）
- GoogLeNet(2014)
  - Incetption
- ResNet (2015)
  - Residual blocks

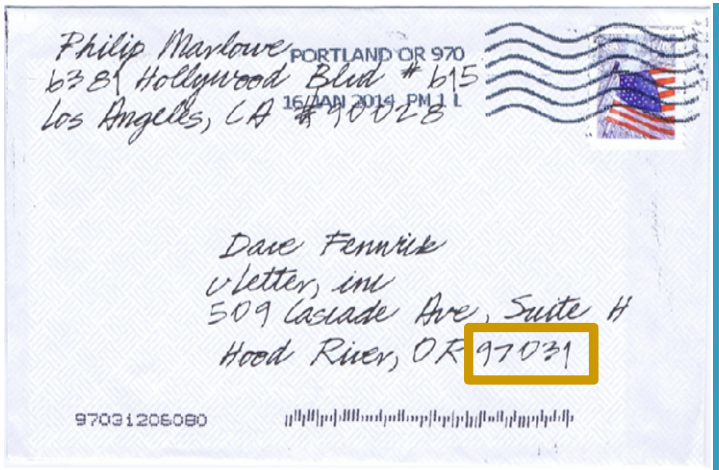
。 。 。 。 。 。

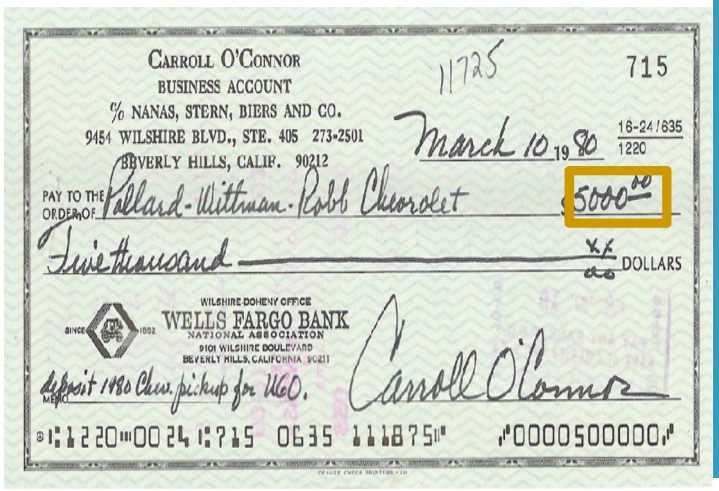
LeNet

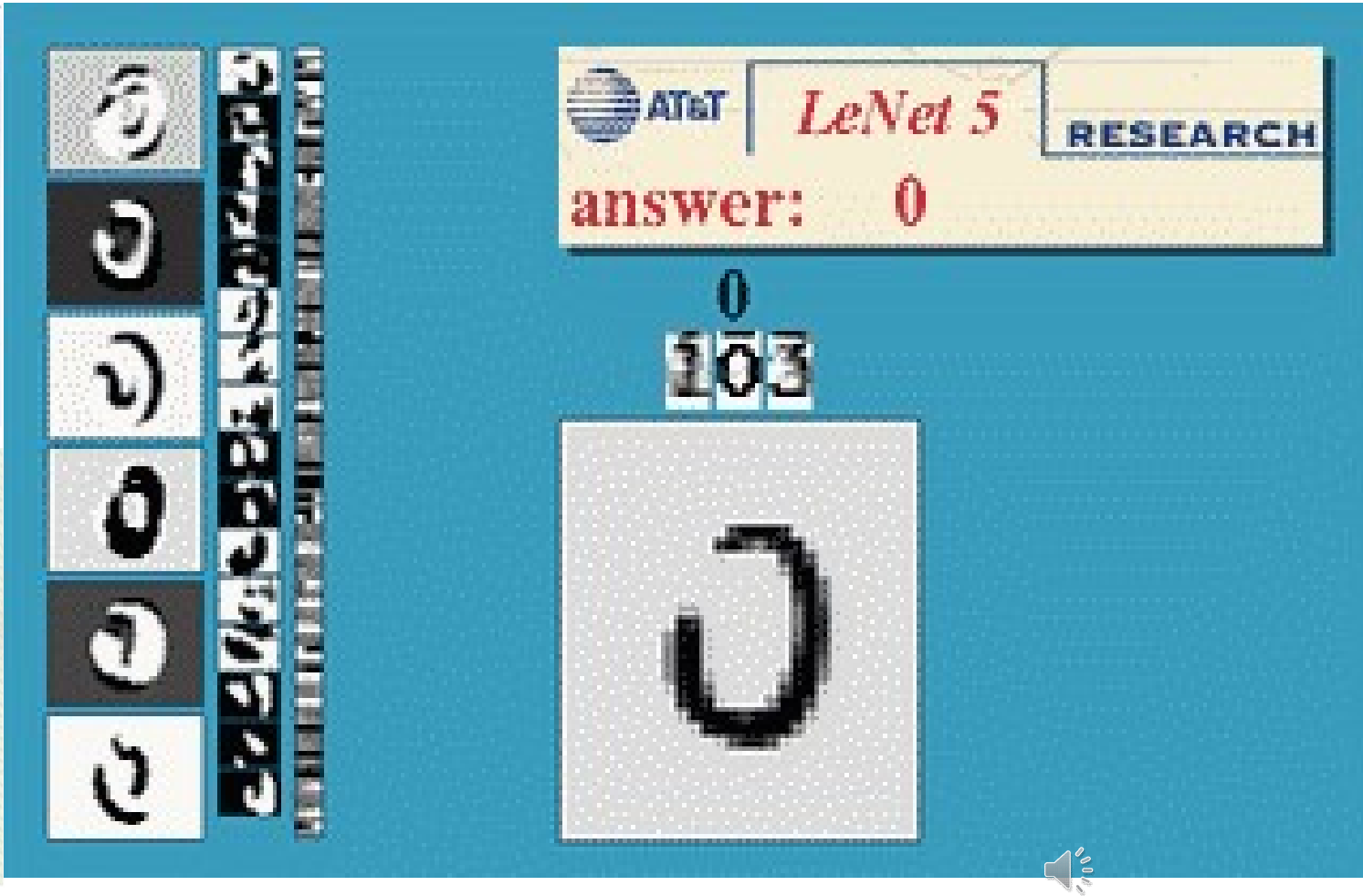


LeCun, Y., Bottou, L., **Bengio**, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324.

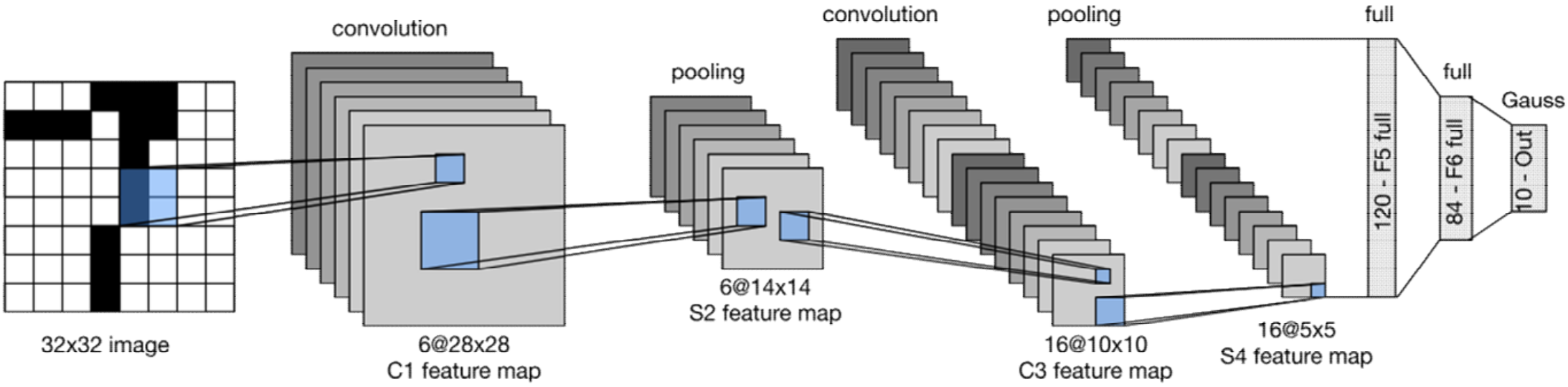
Handwritten Digit Recognition





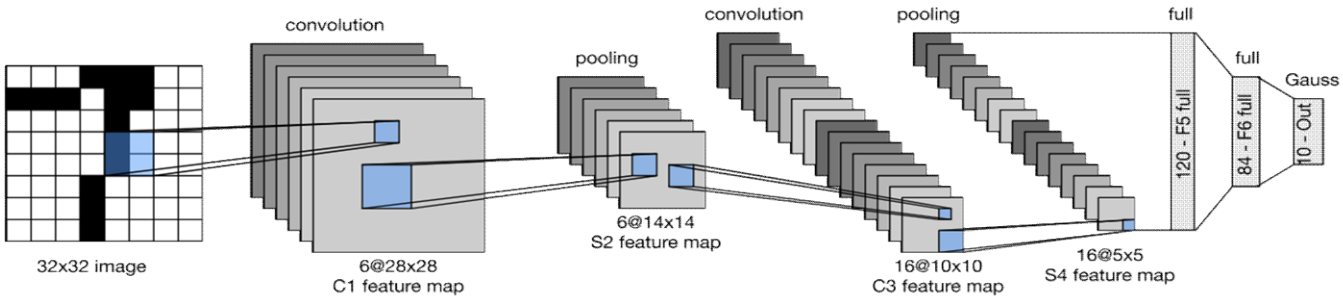


LeNet Architecture



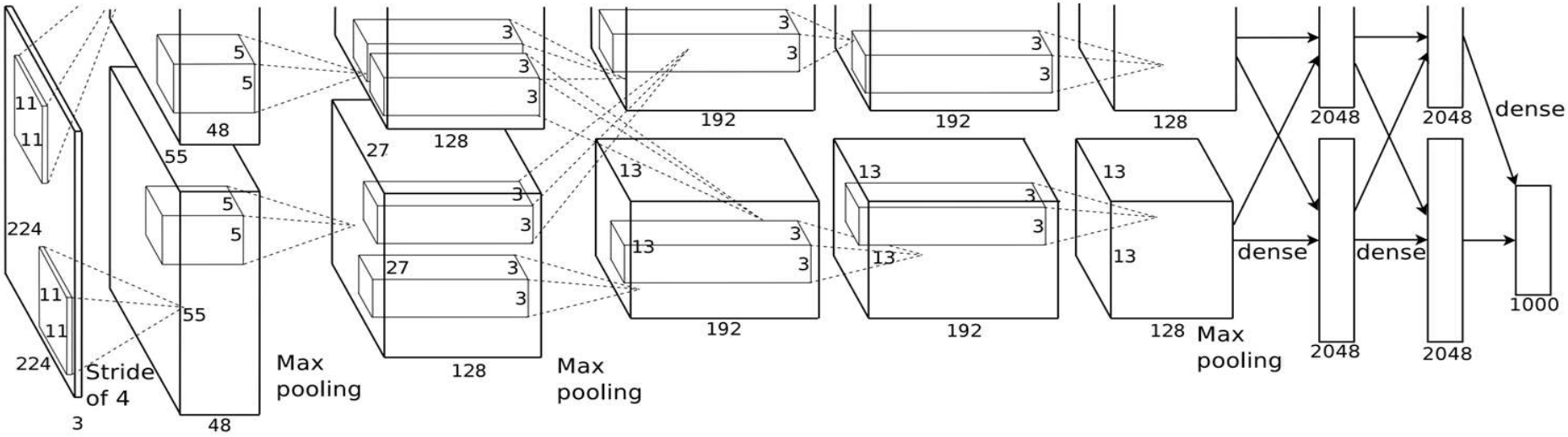
- convolutional layers labeled Cx,
- subsampling layers labeled Sx
- fully connected layers labeled Fx

LeNet



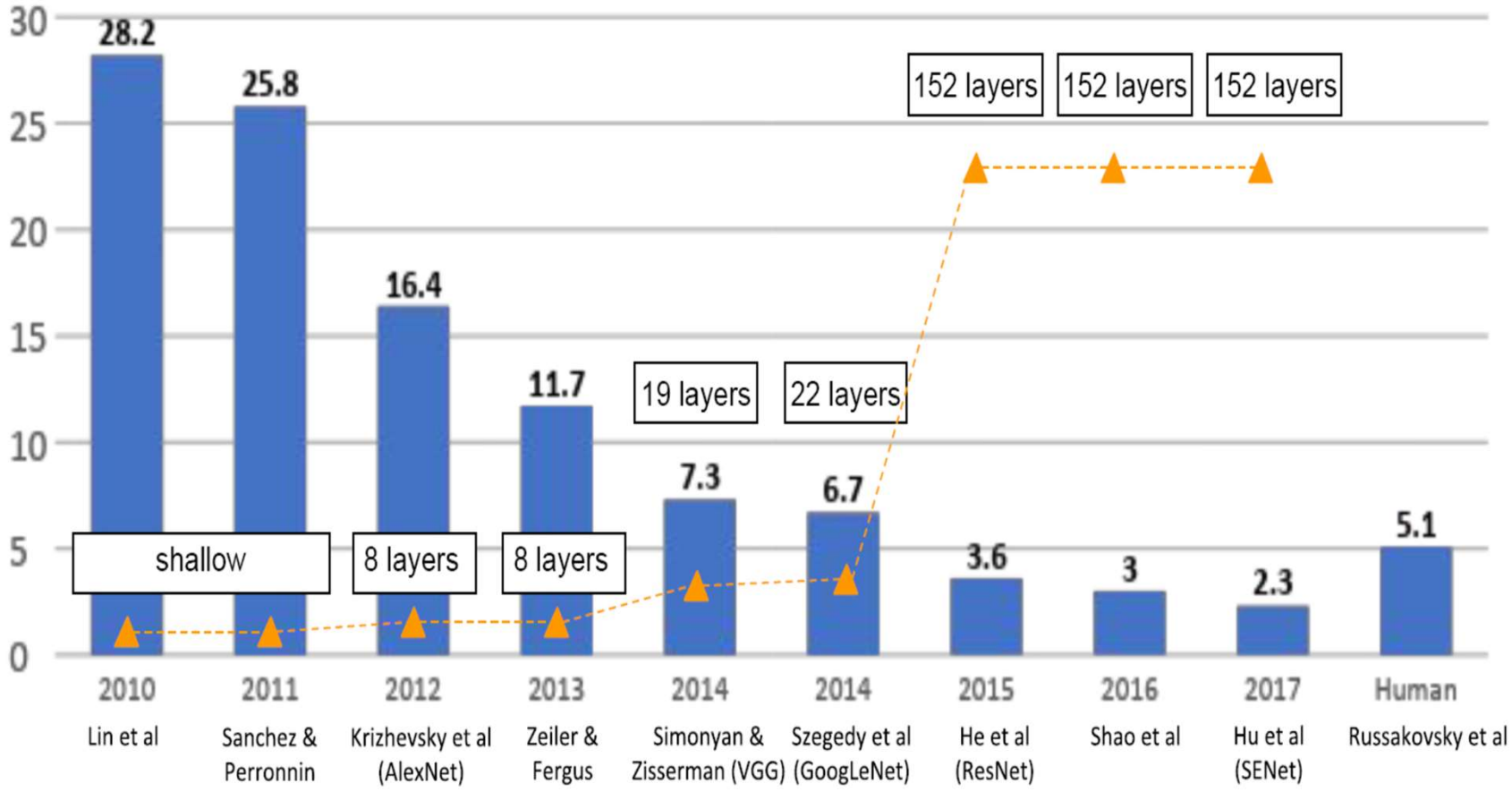
- convolutional block
  - convolutional layer
    - used to recognize the spatial patterns in the image, such as lines and the parts of objects
    - each convolutional layer uses a **5*5** window
  - pooling layer
    - **average pooling** layer is used to reduce the dimensionality
    - the window shape is 2 x 2 and the stride is 2

AlexNet



Krizhevsky, A., Sutskever, I., & **Hinton**, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems (pp. 1097-1105).

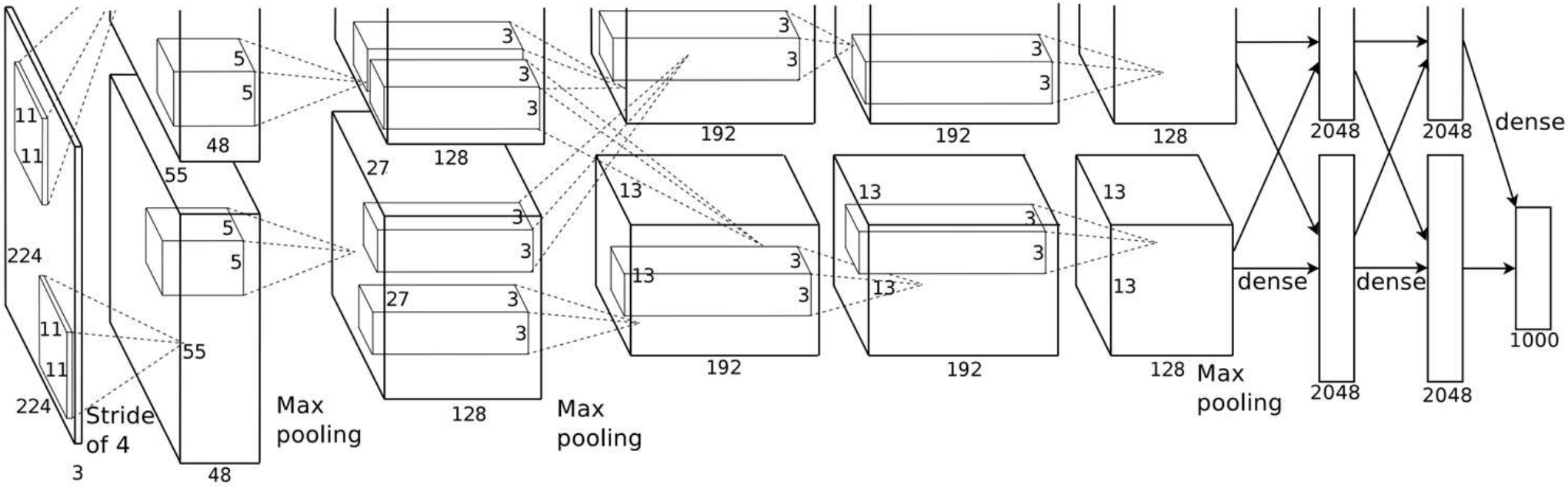
ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



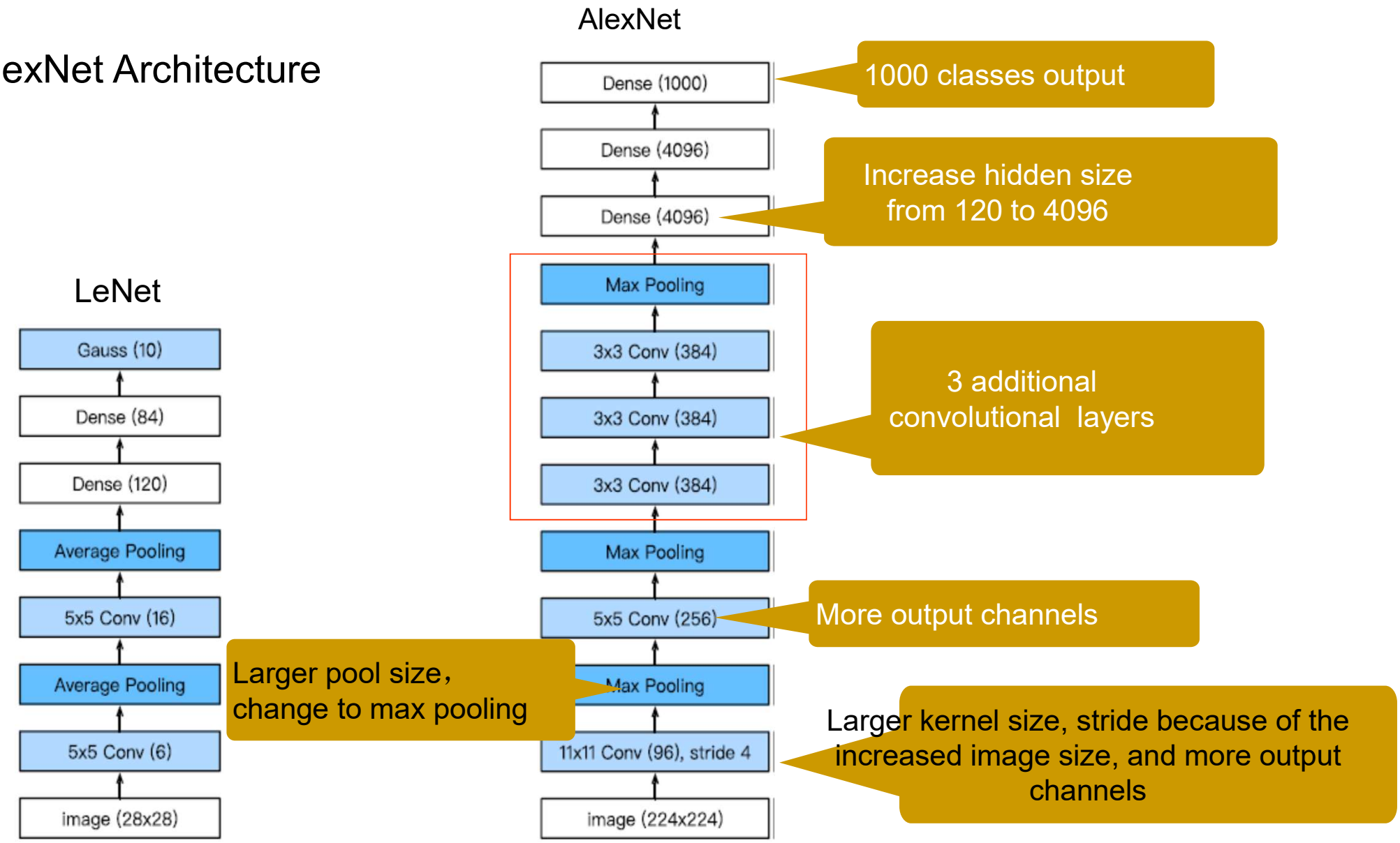
AlexNet

- A key step from shallow to deep networks！
- AlexNet won ImageNet competition in 2012
- Deeper and bigger LeNet
- Key modifications
  - Dropout (regularization)
  - ReLu (training)
  - Data augmentation、Max Pooling

AlexNet Architecture



AlexNet Architecture

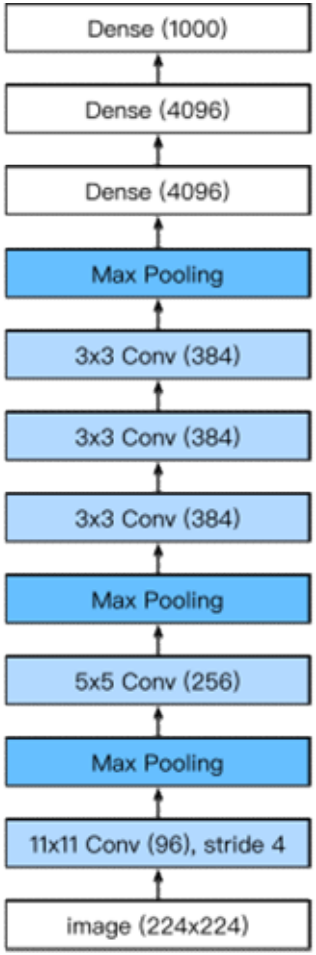


More Tricks

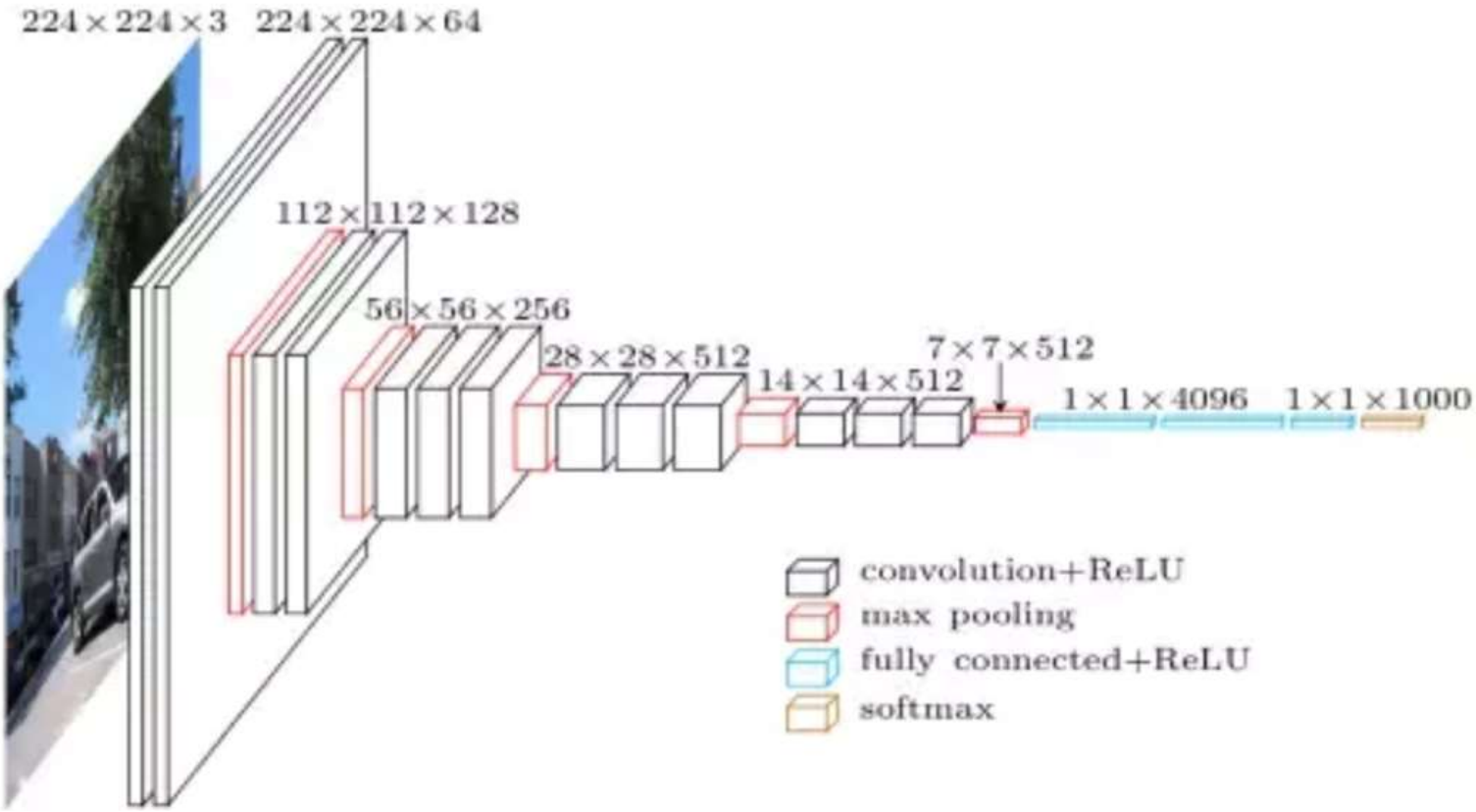
- Change activation function from sigmoid to ReLu (no more vanishing gradient)
- Add a dropout layer after two hidden dense layers (better robustness / regularization)
- Data augmentation

Complexity

	#parameters		FLOP	
	AlexNet	**LeNet**	**AlexNet**	**LeNet**
Conv1	35K	150	101M	1.2M
Conv2	614K	2.4K	415M	2.4M
Conv3-5	3M		445M	
Dense1	26M	0.48M	26M	0.48M
Dense2	16M	0.1M	16M	0.1M
Total	46M	0.6M	1G	4M
Increase	11x	1x	250x	1x

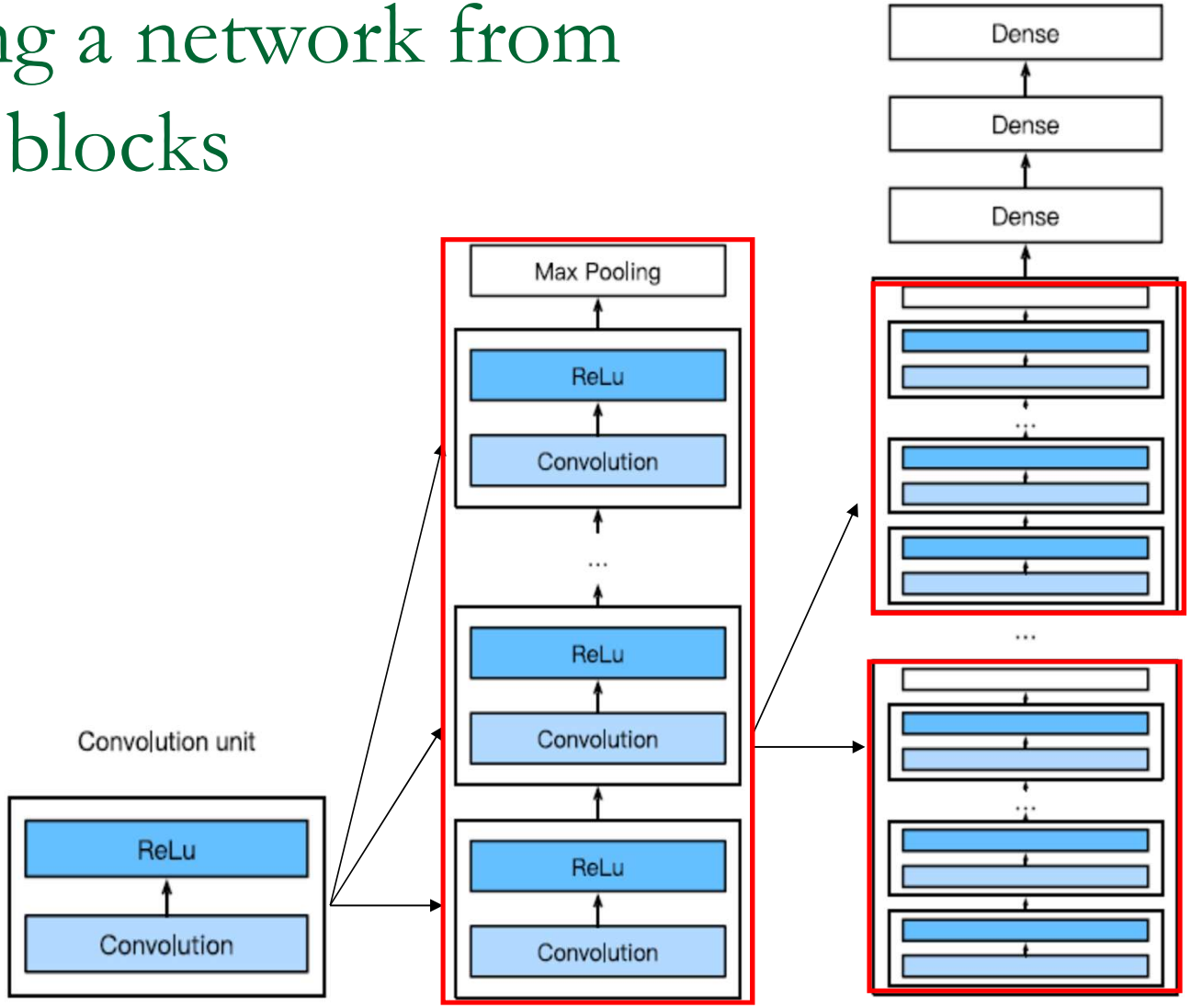


VGG



Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.

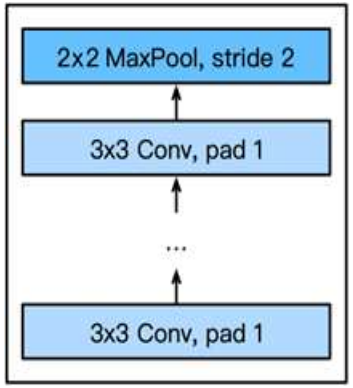
Designing a network from building blocks



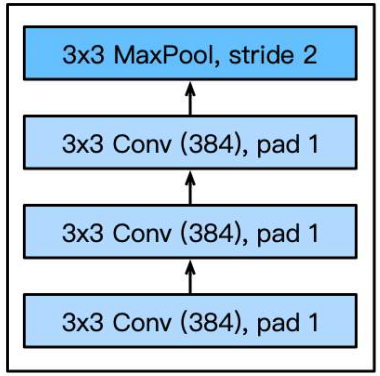
VGG Blocks

- Deeper vs. wider?
  - 5x5 convolutions
  - 3x3 convolutions (more)
  - **deep & narrow better**
- VGG block
  - *3x3* convolutions (pad 1)
    (*n* layers, *m* channels)
  - 2x2 max-pooling
    (stride 2)

VGG block



Part of AlexNet



VGG Architecture

- Multiple VGG blocks followed by dense layers

- Vary the repeating number to get different architectures, such as VGG-16, VGG-19, …

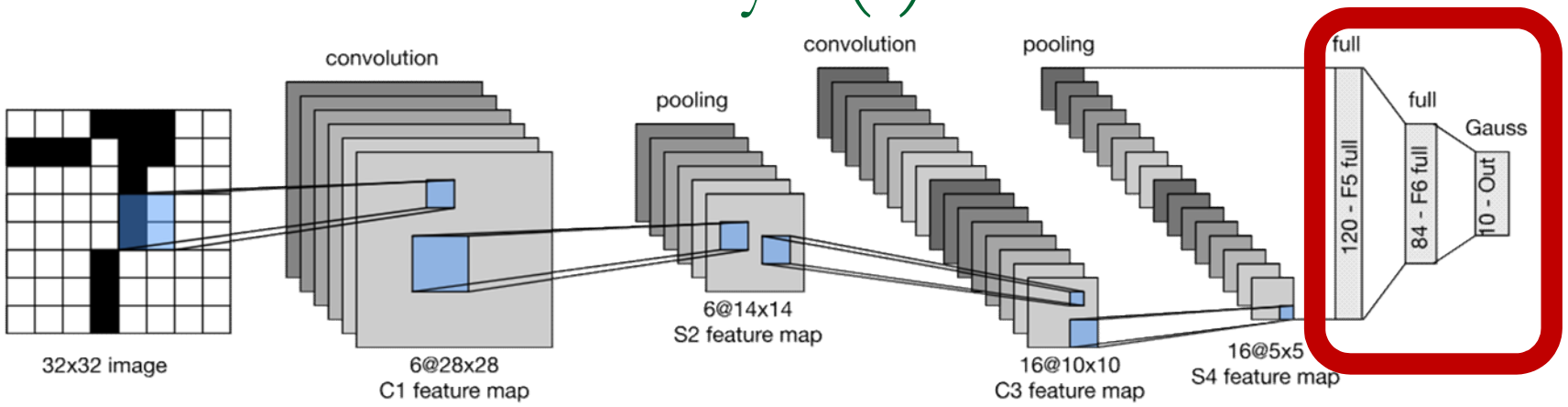
VGG

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Design pattern

- The design pattern of LeNet, AlexNet, and VGG：
  - extract the spatial features through a sequence of convolutions and pooling layers
  - post-process the representations via fully connected layers

The Curse of the Last Layer(s)



- Convolution layers need relatively few parameters

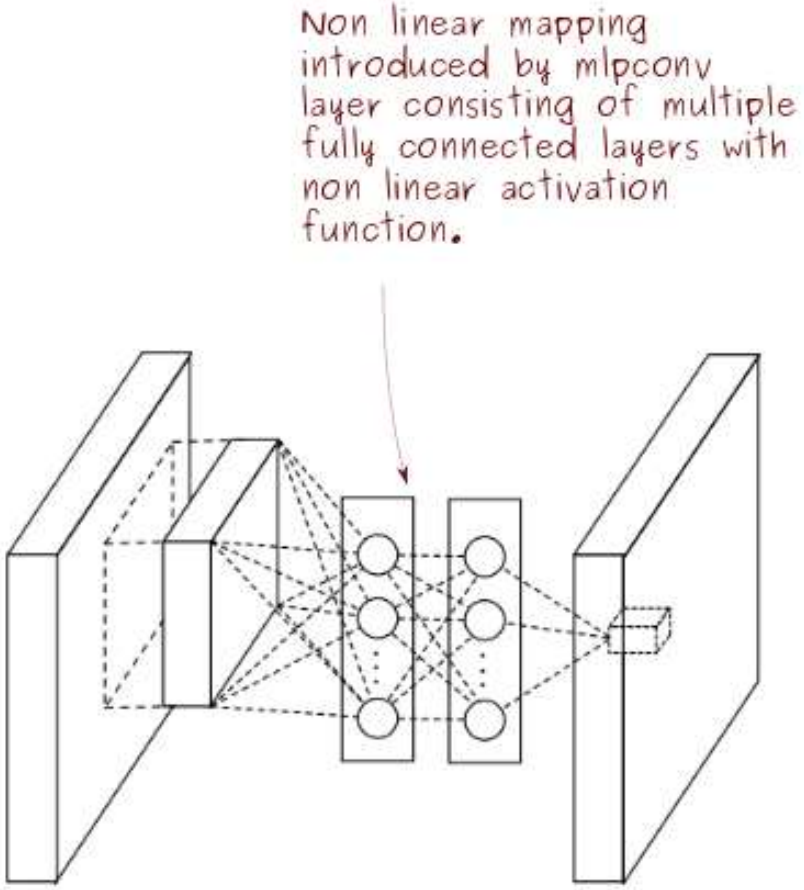
$$c_i \times c_o \times k^2$$

- Last layer needs many parameters for n classes

$$c \times m_w \times m_h \times n$$

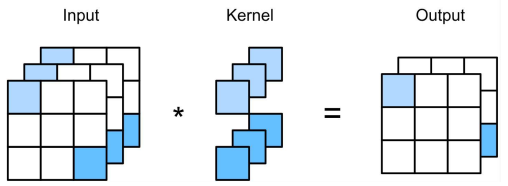
- LeNet: 16x5x5x120 = 48k
- VGG: 512x7x7x4096 = 102M

Network in Network



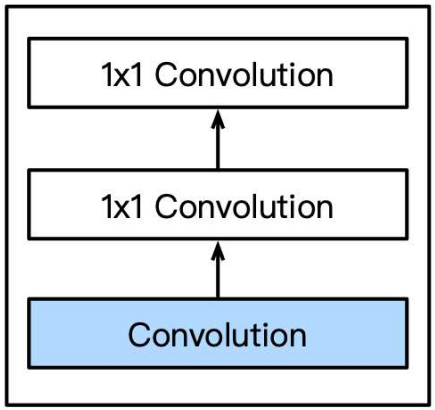
Breaking the Curse of the Last Layer

- Key Idea
  - **Get rid of the fully connected layer(s)**
  - Convolutions and pooling reduce resolution (e.g. stride of 2 reduces resolution 4x)
- Implementation details
  - Reduce resolution progressively
  - Increase number of channels
  - Use **1x1 convolutions** (they only act per pixel)
- **Global average pooling in the end**



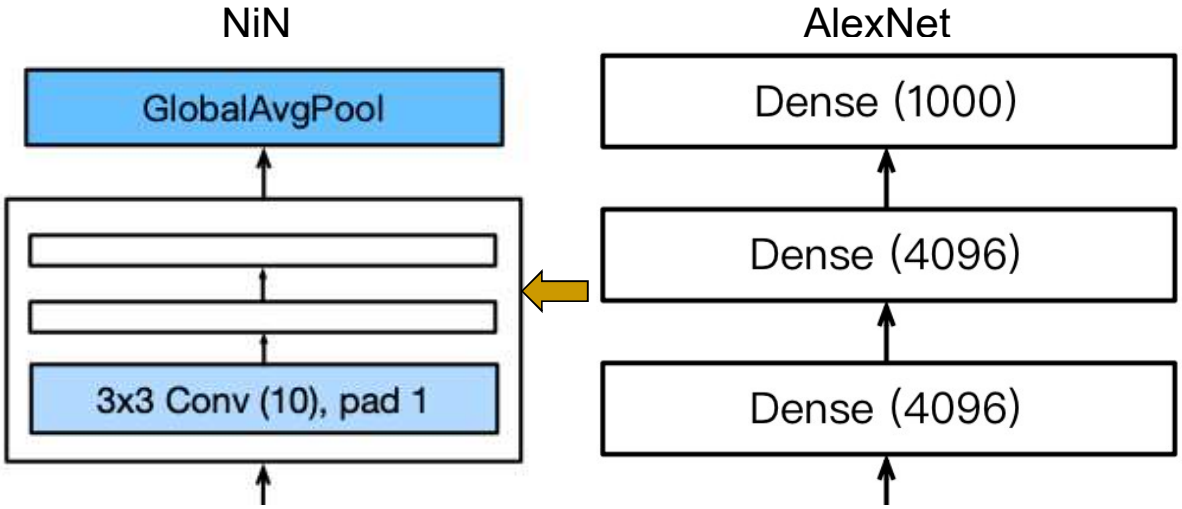
NiN Block

- A convolutional layer
  - kernel size, stride, and padding are hyper-parameters
- Followed by two 1x1 convolutions
  - 1 stride and no padding, share the same output channels as the first layer
  - Act as dense layers

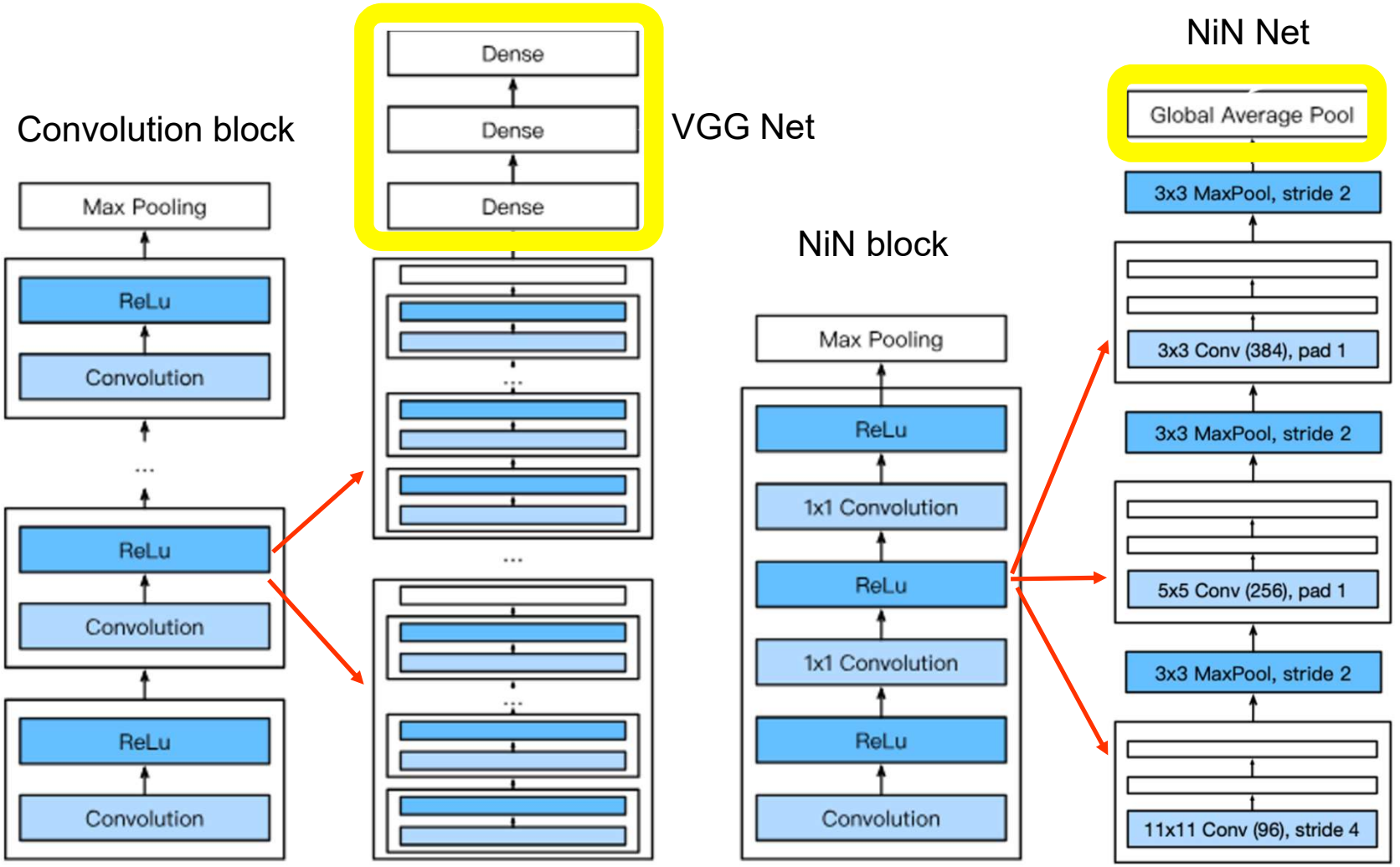


NiN Last Layers

- Replace AlexNet's dense layers with a NiN block
- Global average pooling layer to combine outputs

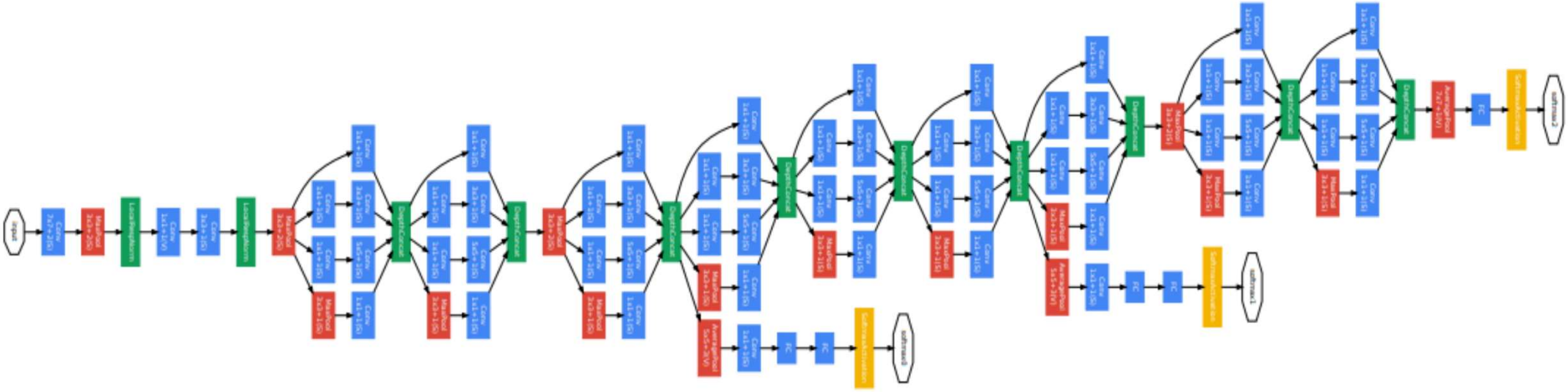


NiN Networks



GoogLeNet(2014)

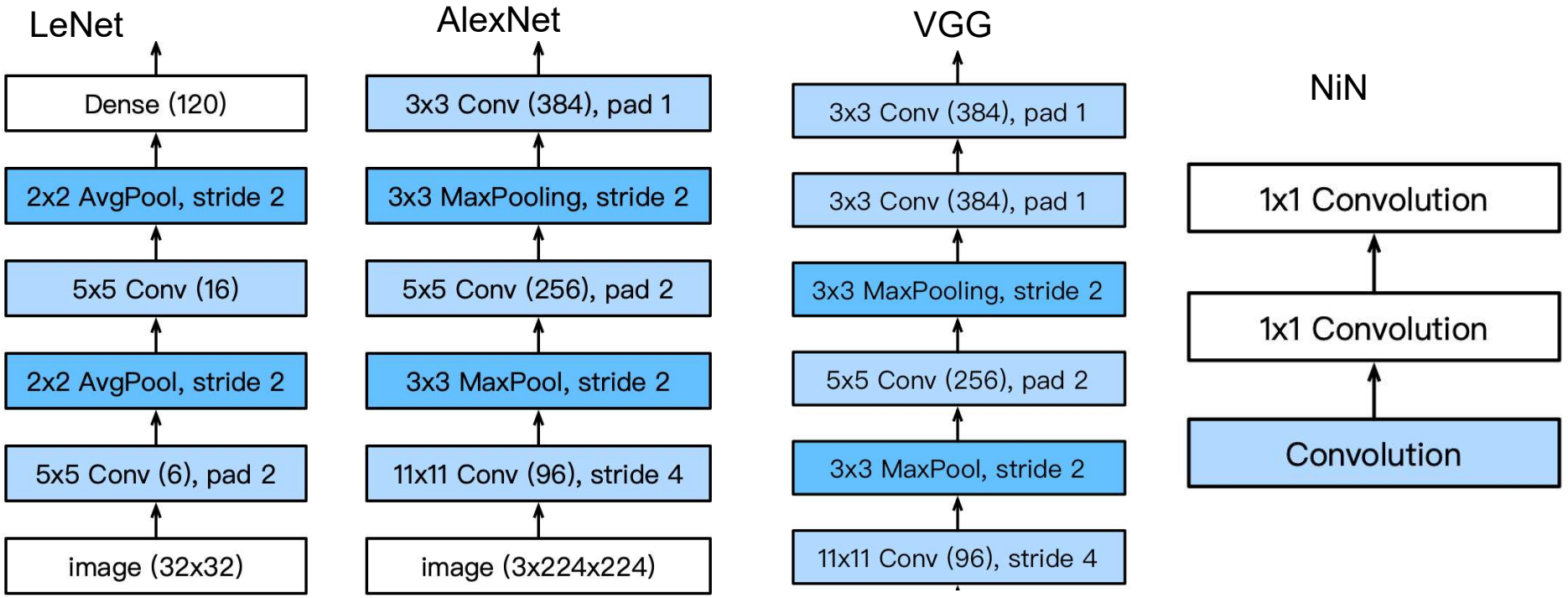
- Networks with Parallel Concatenations



Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., & Anguelov, D. & Rabinovich, A. (2015). Going deeper with convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1-9).

Picking the best convolution …

1x1 | 3x3 | 5x5 | 11x11 | Multiple 1x1

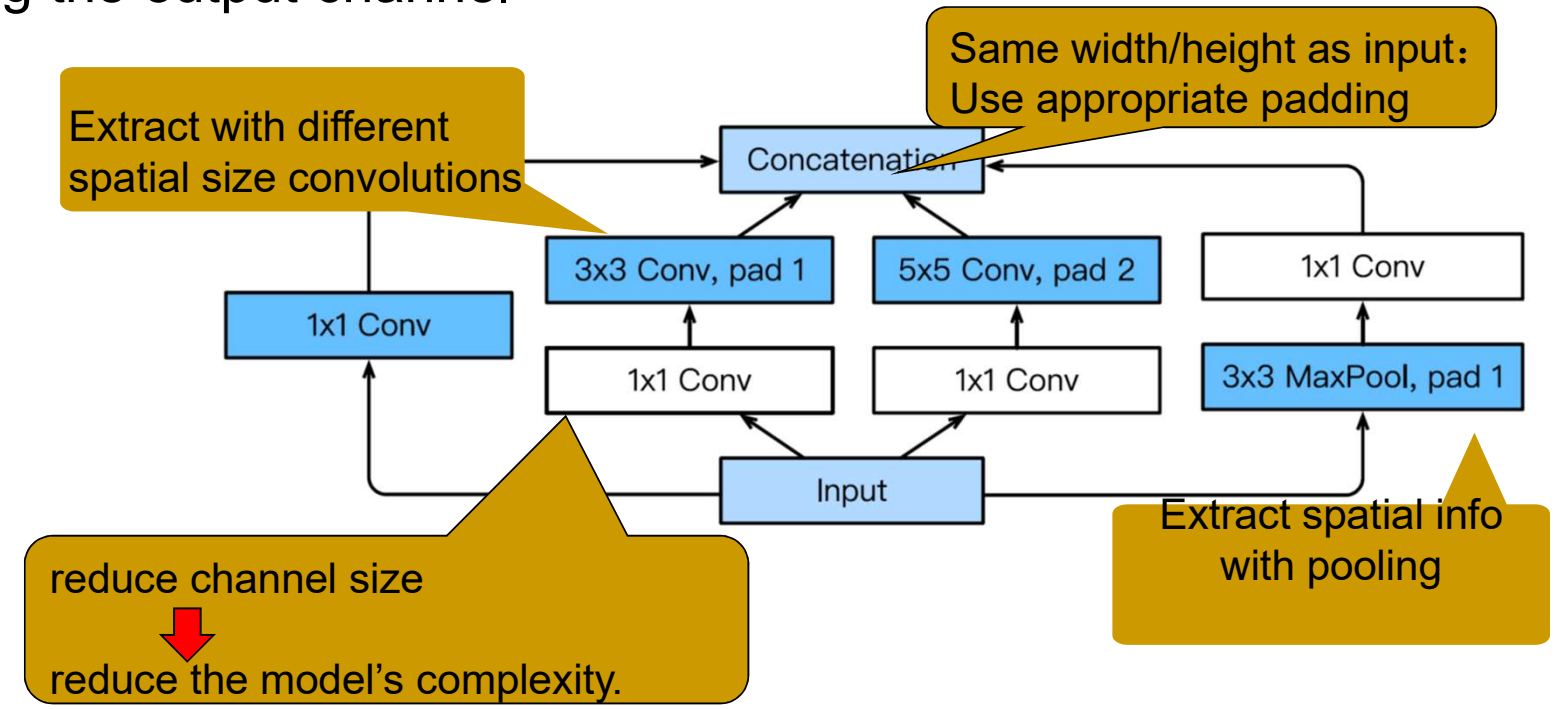


Why choose? Just pick them all !

Inception Blocks

- ✓ The basic convolutional block in GoogLeNet.

4 paths extract information from different aspects, then concatenate along the output channel

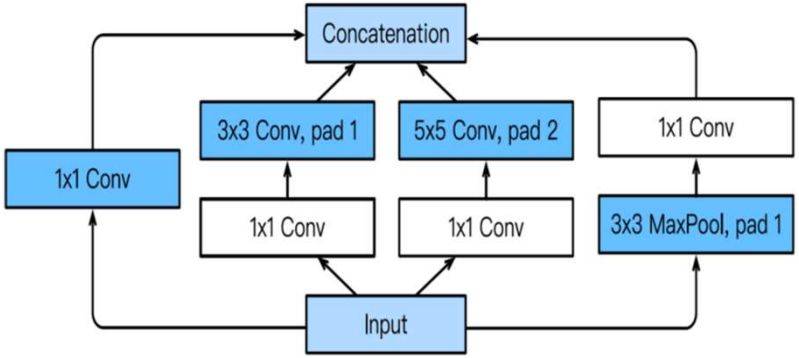


Inception Blocks

Inception blocks have fewer parameters and less computation complexity than a single 3x3 or 5x5 convolutional layer

- Mix of different functions (powerful function class)
- Memory and compute efficiency (good generalization)

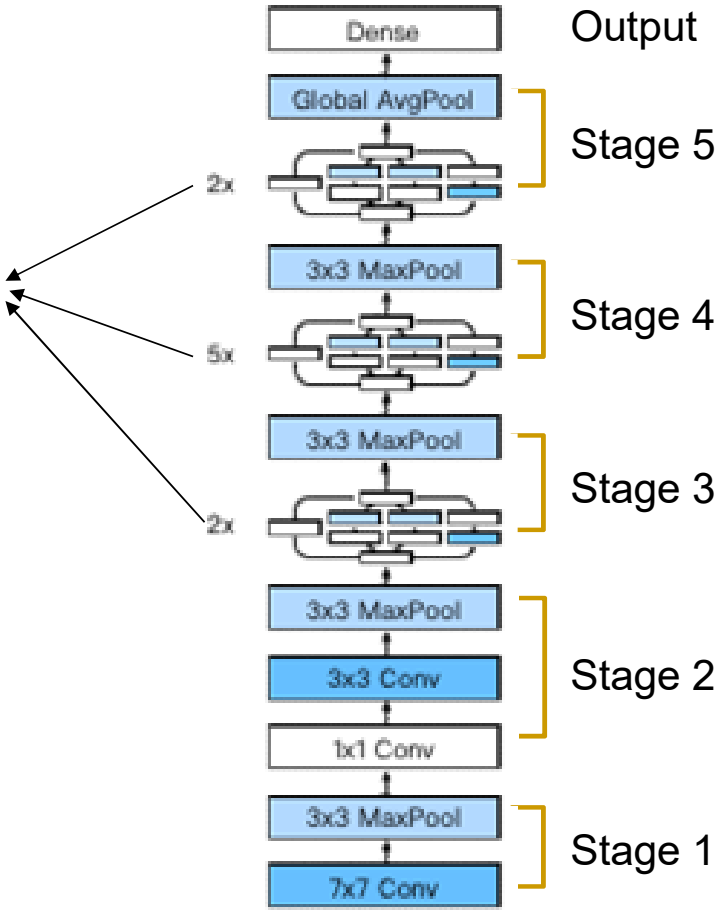
	#parameters	FLOPS
Inception	0.16 M	128 M
3x3 Conv	0.44 M	346 M
5x5 Conv	1.22 M	963 M



GoogLeNet

5 stages with

9 inception blocks



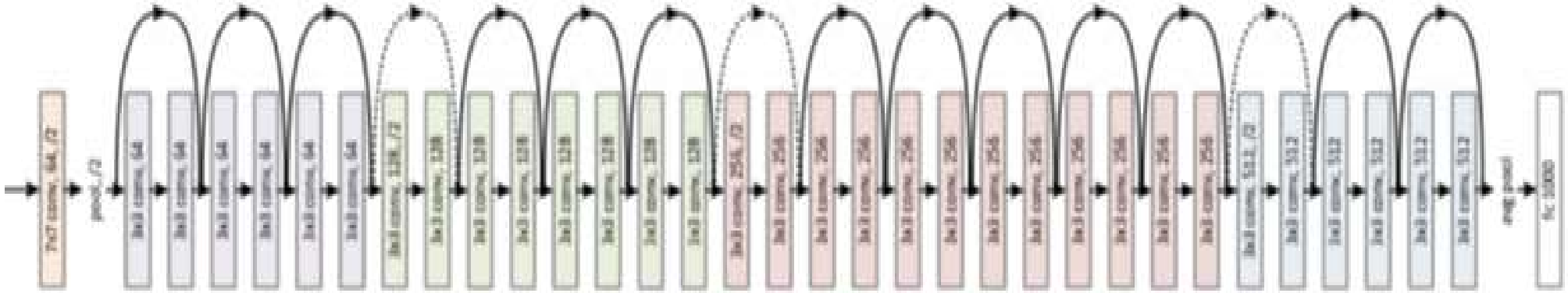
GoogLeNet Incarnation of the Inception Architecture

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

The many flavors of Inception Networks

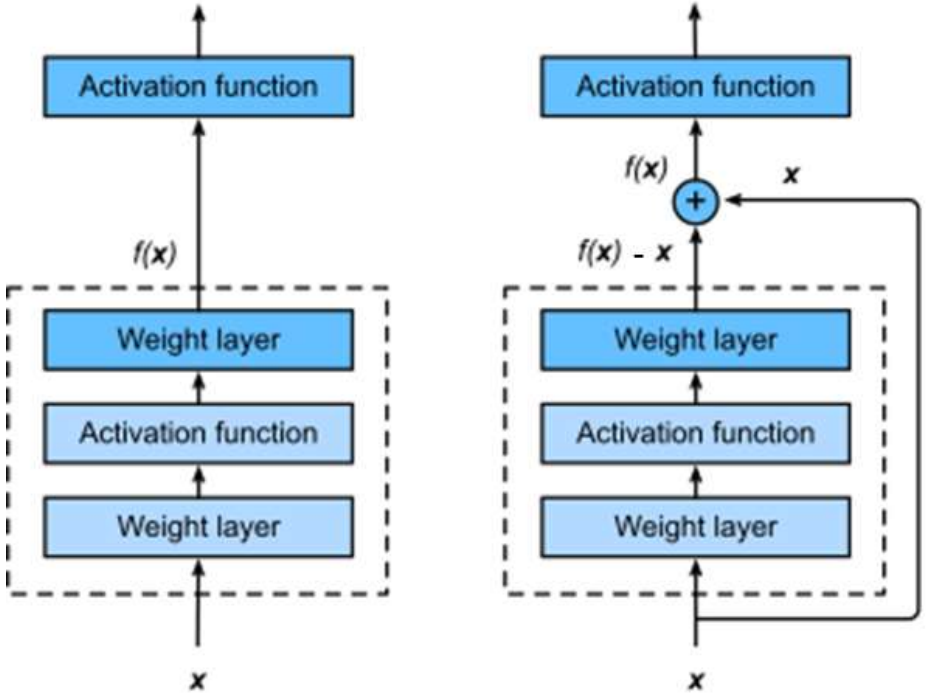
- Inception-BN (v2) - Add batch normalization
- Inception-V3 - Modified the inception block
  - Replace 5x5 by multiple 3x3 convolutions
  - Replace 5x5 by 1x7 and 7x1 convolutions
  - Replace 3x3 by 1x3 and 3x1 convolutions
  - Generally deeper stack
- Inception-V4 - Add residual connections (more later)

Residual Networks (ResNet)



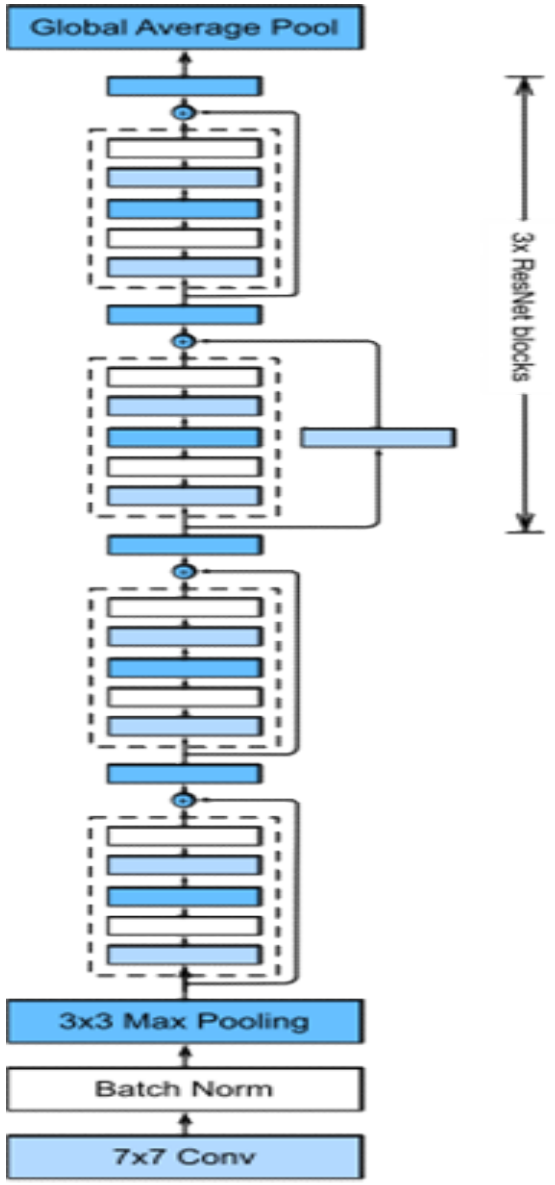
He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. CVPR (pp. 770-778).

Residual Networks (ResNet)

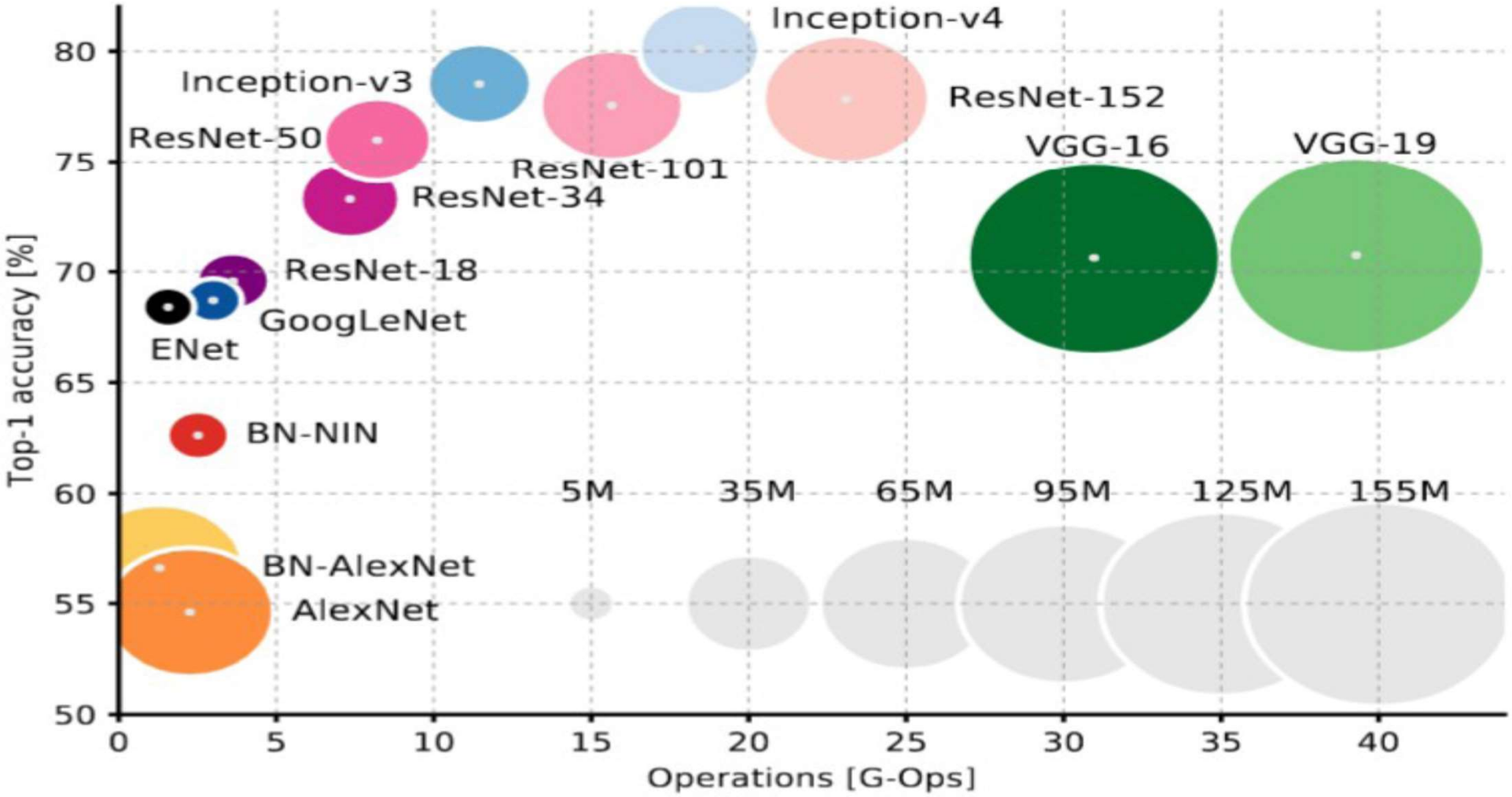


ResNet

- Same block structure as e.g. VGG or GoogleNet
- Residual connection to add to the expressiveness
- Pooling/stride for dimensionality reduction
  - Down sample per module (stride=2)
- Batch Normalization for capacity control



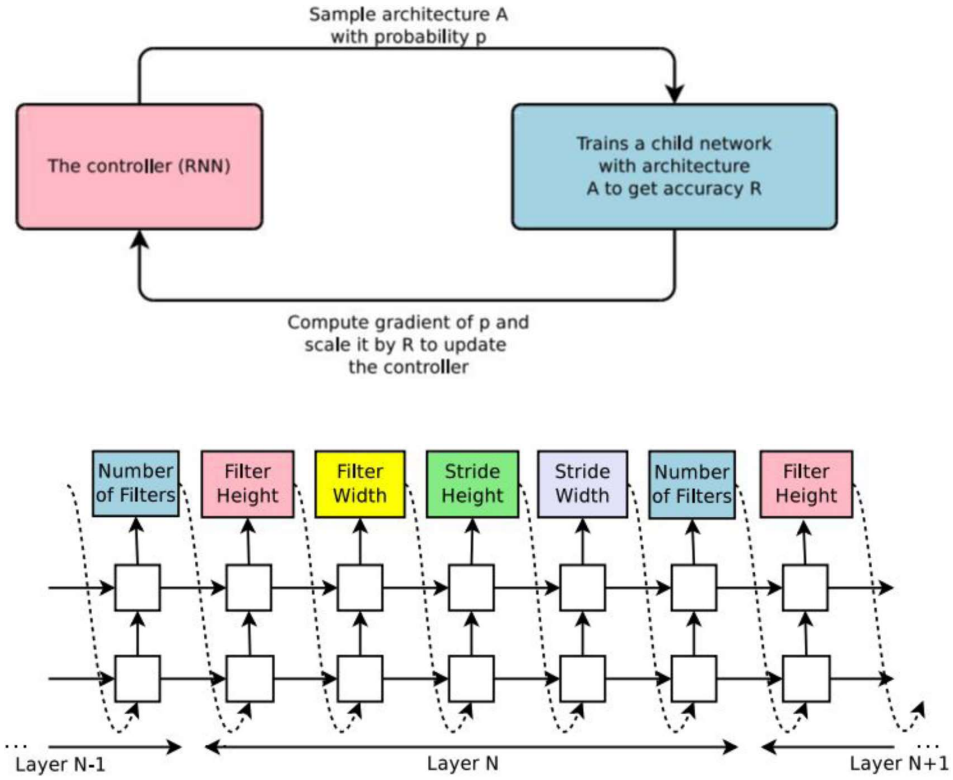
GOPS vs. Accuracy on ImageNet vs. #Parameters



Neural Architecture Search with Reinforcement Learning (NAS)

[Zoph et al. 2016]

- "Controller" network that learns to design a good network architecture (output a string corresponding to network design)
- Iterate:
  - 1）Sample an architecture from search space
  - 2) Train the architecture to get a "reward" R corresponding to accuracy
  - 3) Compute gradient of sample probability, and scale by R to perform controller parameter update (i.e. increase likelihood of good architecture being sampled, decrease likelihood of bad architecture)



Outline

- Convolutional Neural Network (CNN)
- Recurrent Neural Network (RNN)
- Transformer
- Deep Learning Frameworks

Another Scenario

- Data is not always generated i.i.d., all drawn from some distribution, but follows sequential order.

  e.g.

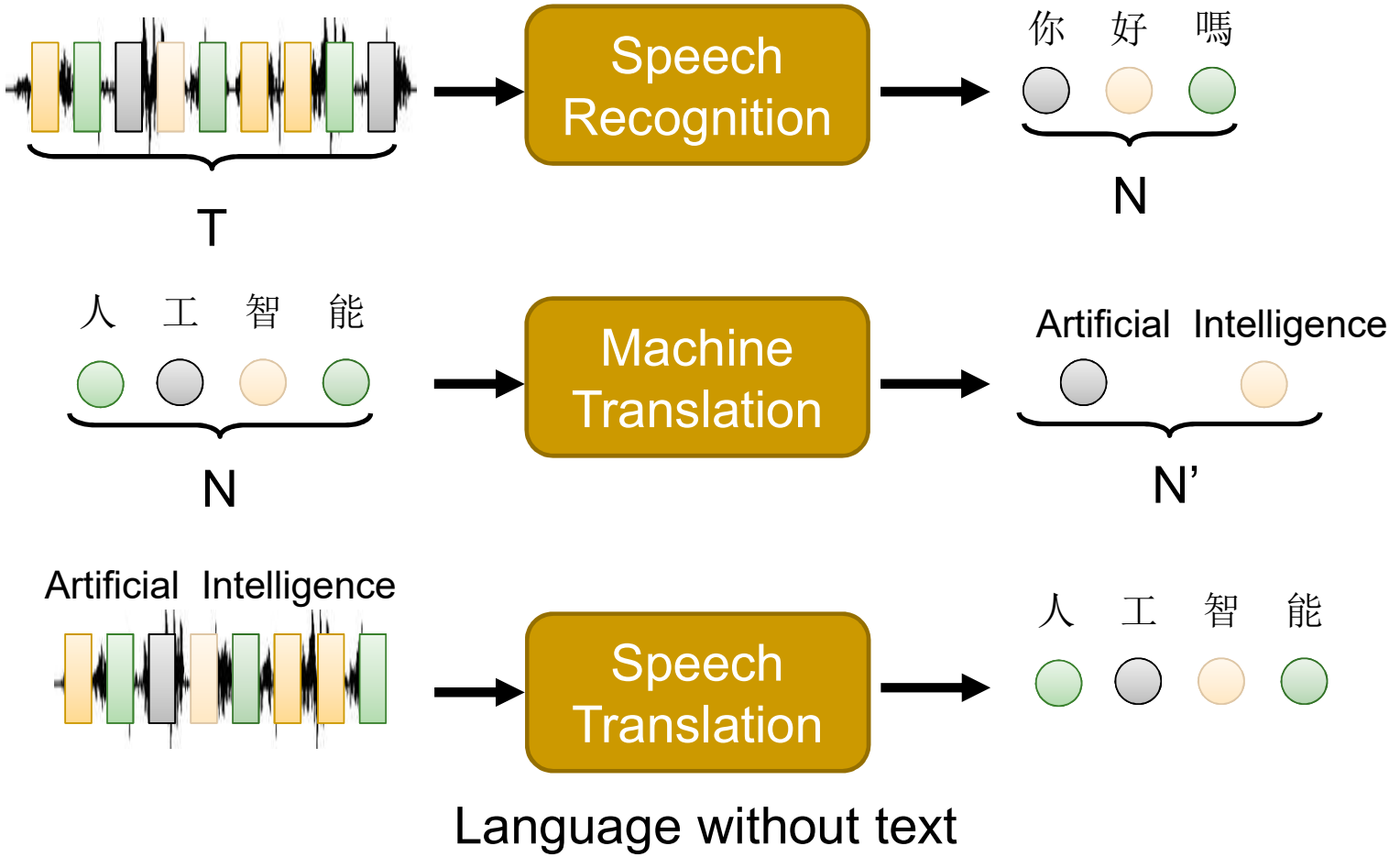
  - the words in a paragraph are written in sequence
  - image frames in a video
  - the audio signal in a conversation
  - the browsing behavior on a website

- Not only receive a sequence as an input, but rather might be expected to continue the sequence.

Sequence-to-sequence (Seq2seq)

Input a sequence, output a sequence

The output length is determined by model.

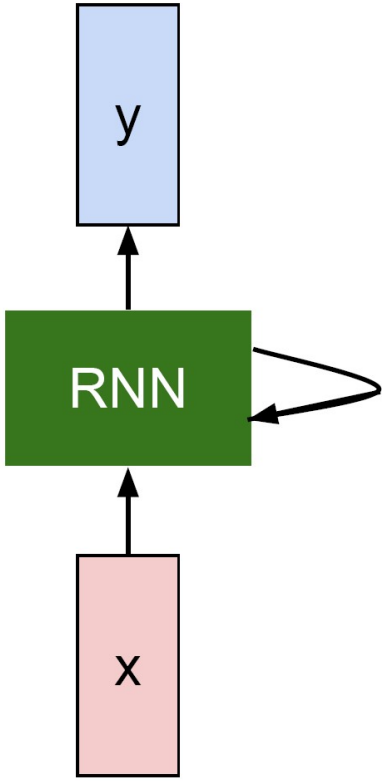


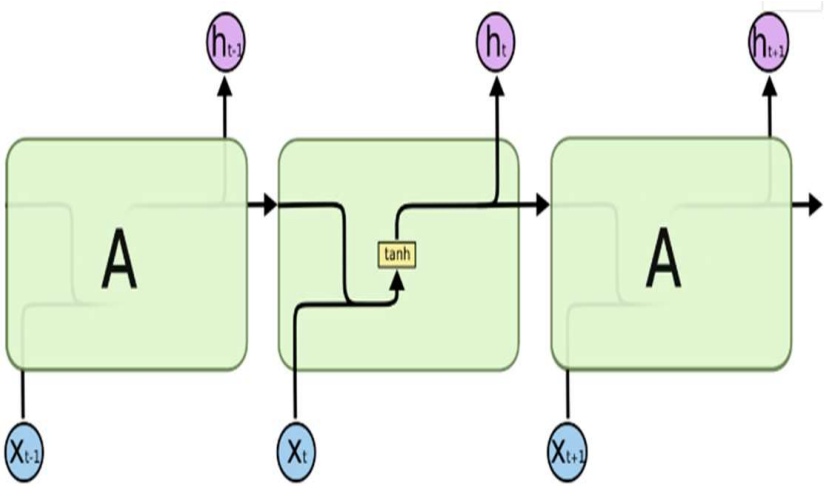
Language without text

When the order of data matters……

Sequence Models !

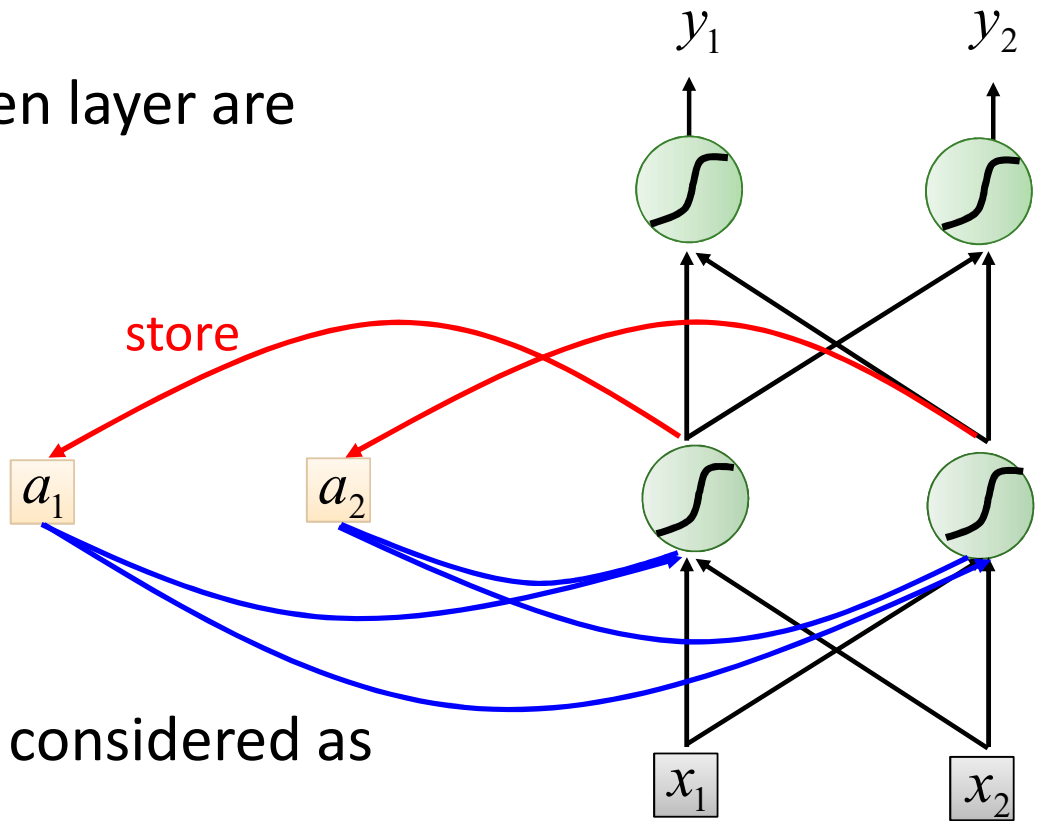
Recurrent Neural Networks





Recurrent Neural Network (RNN)

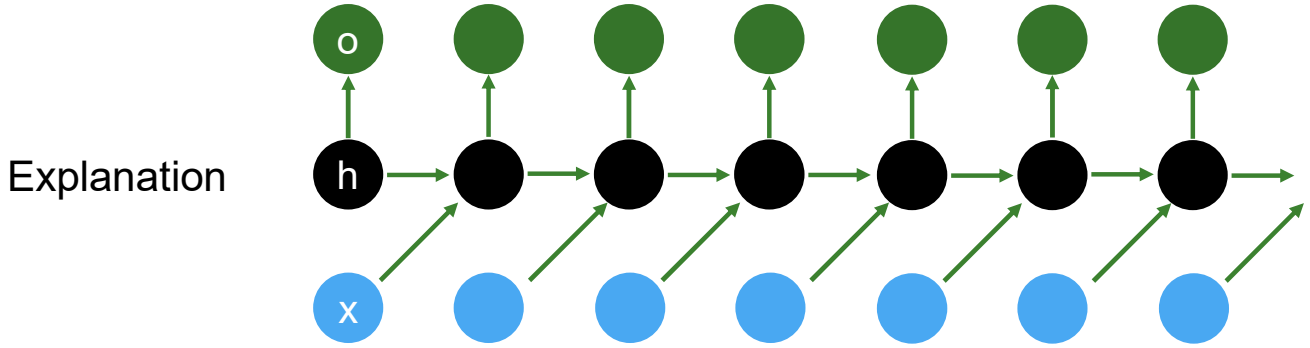
The outputs of the hidden layer are stored in the memory.



Memory can be considered as another input.

Recurrent Neural Networks

(with hidden state)



- Hidden State update

$$\mathbf{h}_t = \phi(\mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{W}_{hx}\mathbf{x}_t + \mathbf{b}_h)$$

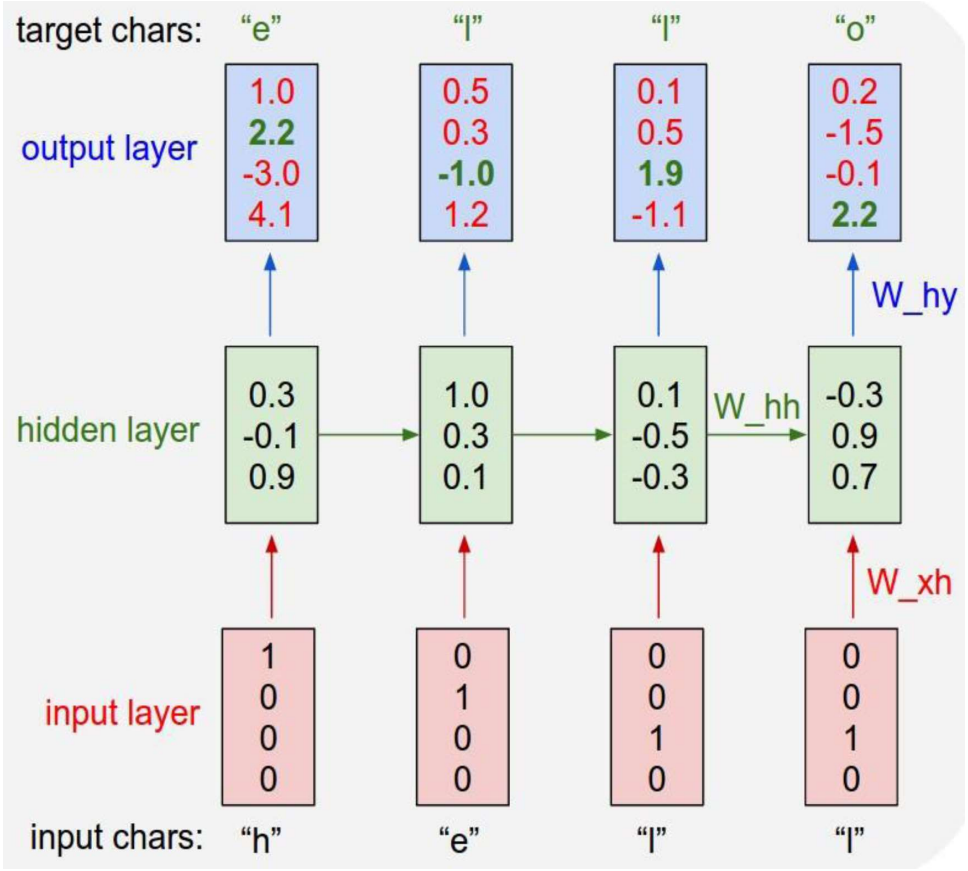
- Observation update

$$\mathbf{o}_t = \phi(\mathbf{W}_{ho}\mathbf{h}_t + \mathbf{b}_o)$$

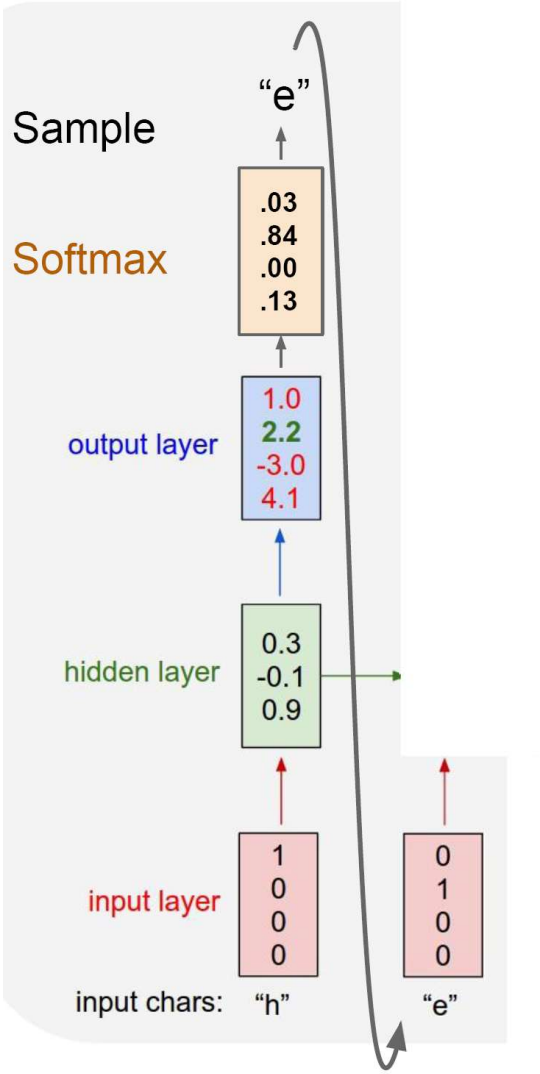
Example: Character-level Language Model

- Vocabulary:
  [h,e,l,o]

- Example **training** sequence:
  "hello"



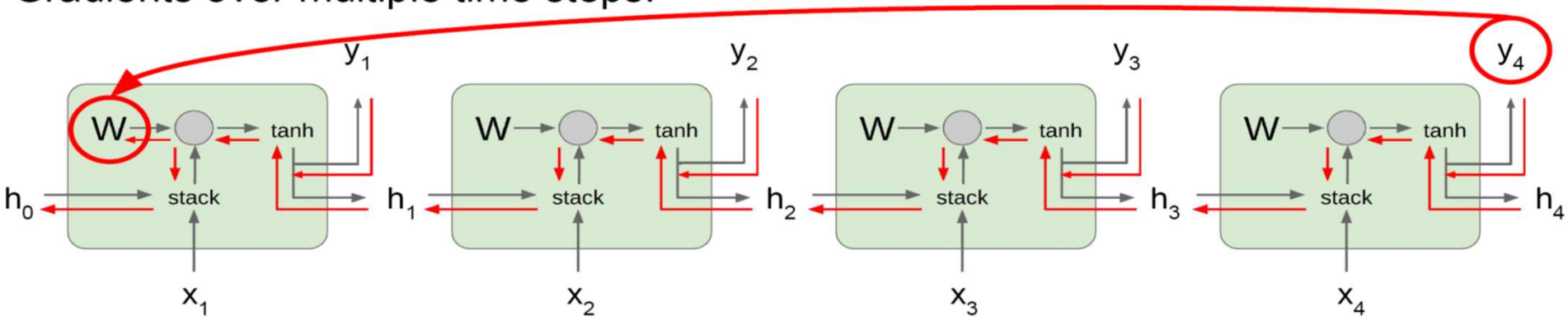
- At test-time sample characters one at a time, feed back to model



Vanilla RNN Gradient Flow

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013

Gradients over multiple time steps:



$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}$$

$$\frac{\partial h_t}{\partial h_{t-1}} = tanh'(W_{hh} h_{t-1} + W_{xh} x_t) W_{hh}$$

$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} \frac{\partial h_t}{\partial h_{t-1}} \cdots \frac{\partial h_1}{\partial W} = \frac{\partial L_T}{\partial h_T} \left(\prod_{t=2}^{T} \frac{\partial h_t}{\partial h_{t-1}} \right) \frac{\partial h_1}{\partial W}$$

Almost always < 1
Vanishing gradients

What if we assumed no non-linearity?
Largest singular value > 1: Exploding gradients
Largest singular value < 1: Vanishing gradients

Gradient Explode/ Vanish

$w = 1 \Rightarrow y^{1000} = 1$

$w = 1.01 \Rightarrow y^{1000} \approx 20000$

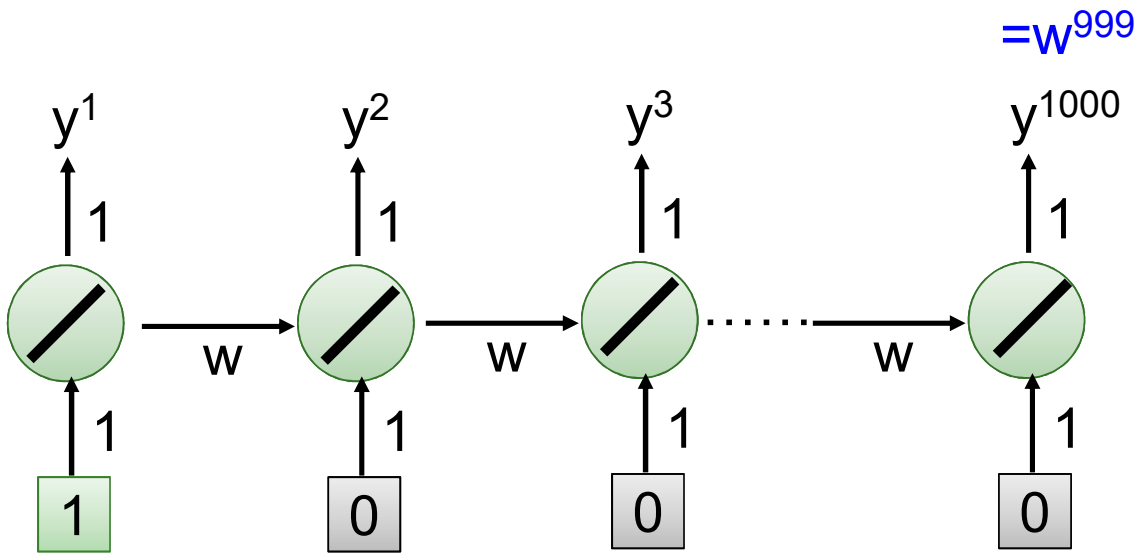
Large $\partial L/\partial w$ → Small Learning rate?

$w = 0.99 \Rightarrow y^{1000} \approx 0$

$w = 0.01 \Rightarrow y^{1000} \approx 0$

small $\partial L/\partial w$ → Large Learning rate?

Toy Example



Recurrent Neural Networks(RNN)

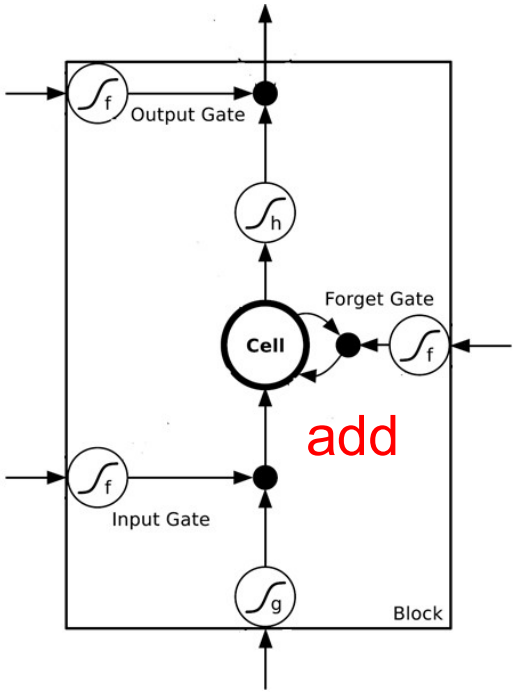
- ✓ Suitable for processing sequences, often applied to the processing of text.
- ❖ Has a problem about gradient vanishing
- ❖ Can not store long-term memory

Modify the structure

- Long Short-term Memory (LSTM)
  - Can deal with gradient vanishing (not gradient explode)
    - Memory and input are ***added***
    - The influence never disappears unless the forget gate is closed

      → No Gradient vanishing

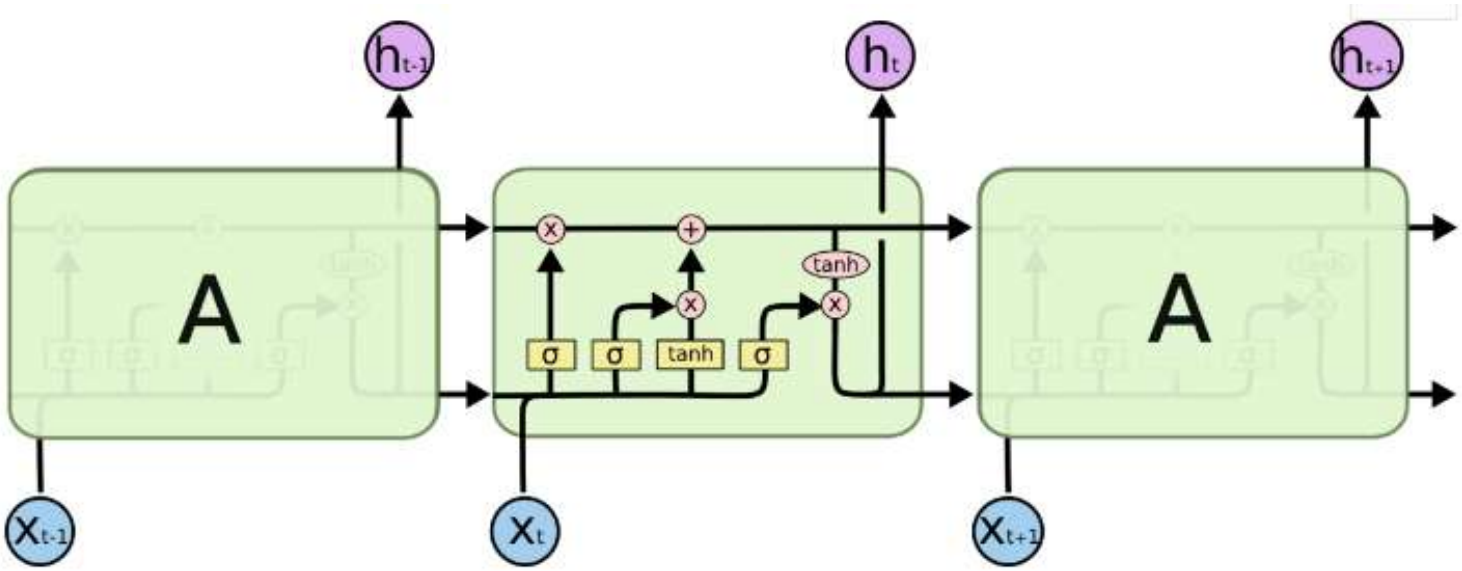
      (If the forget gate is opened.)



- Gated Recurrent Unit(GRU):

  simpler than LSTM

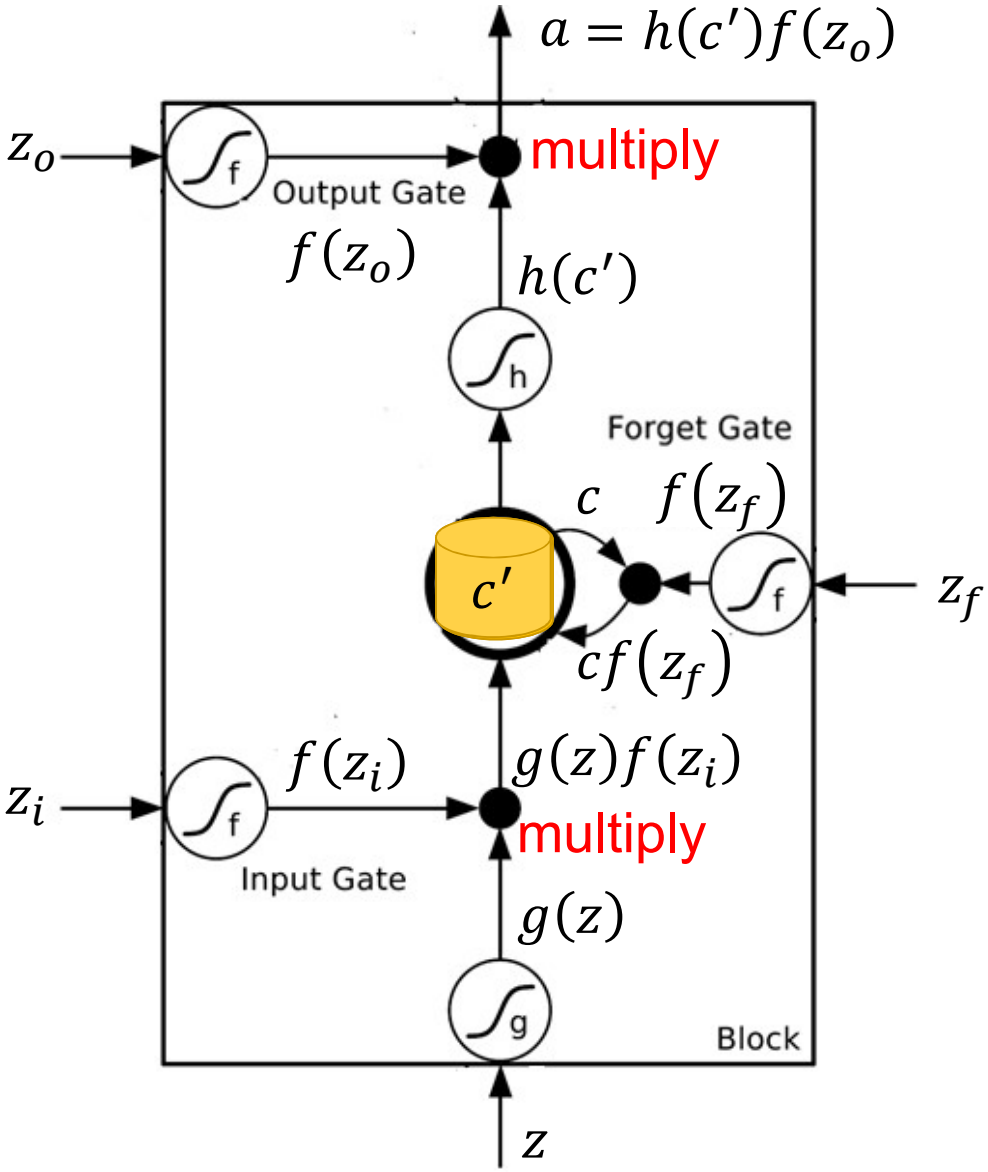
Long Short Term Memory



Hochreiter & Schmidhuber Long Short-term Memory[J]. Neural Computation, 1997,9(8):1735-1780.

Long Short Term Memory

- **Forget gate**
  - Shrink values towards zero**, whether to erase cell**
- **Input gate**
  - Decide whether we should ignore the input data
- **Output gate**
  - Decide whether the hidden state is used for the output generated by the LSTM
- **Hidden state** and **Memory cell**



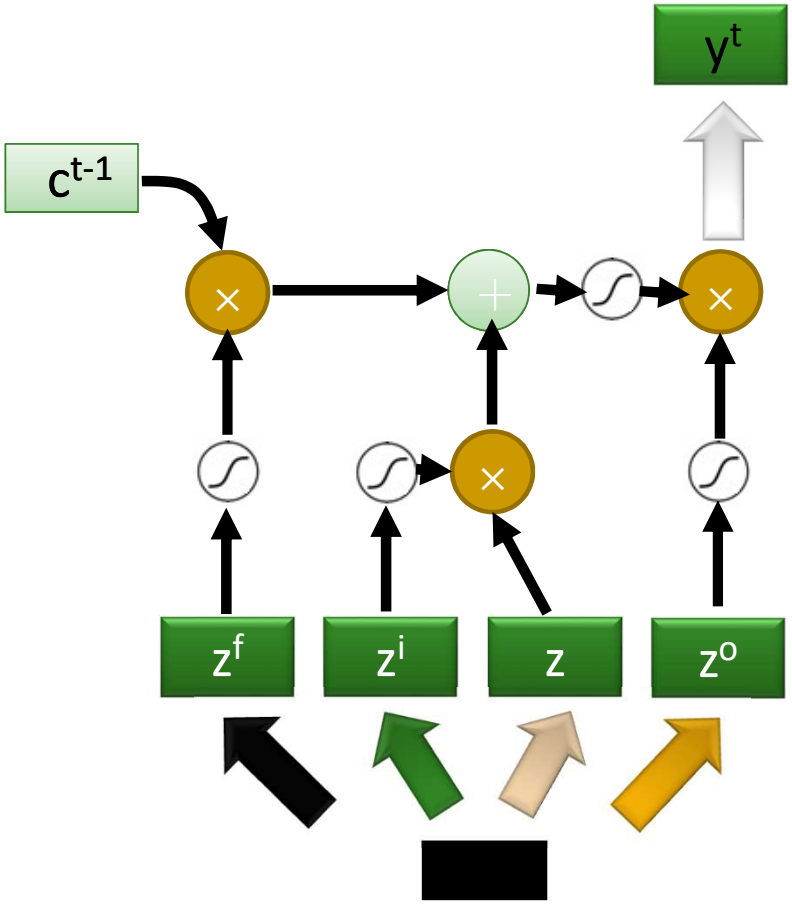
Activation function f is usually a sigmoid function

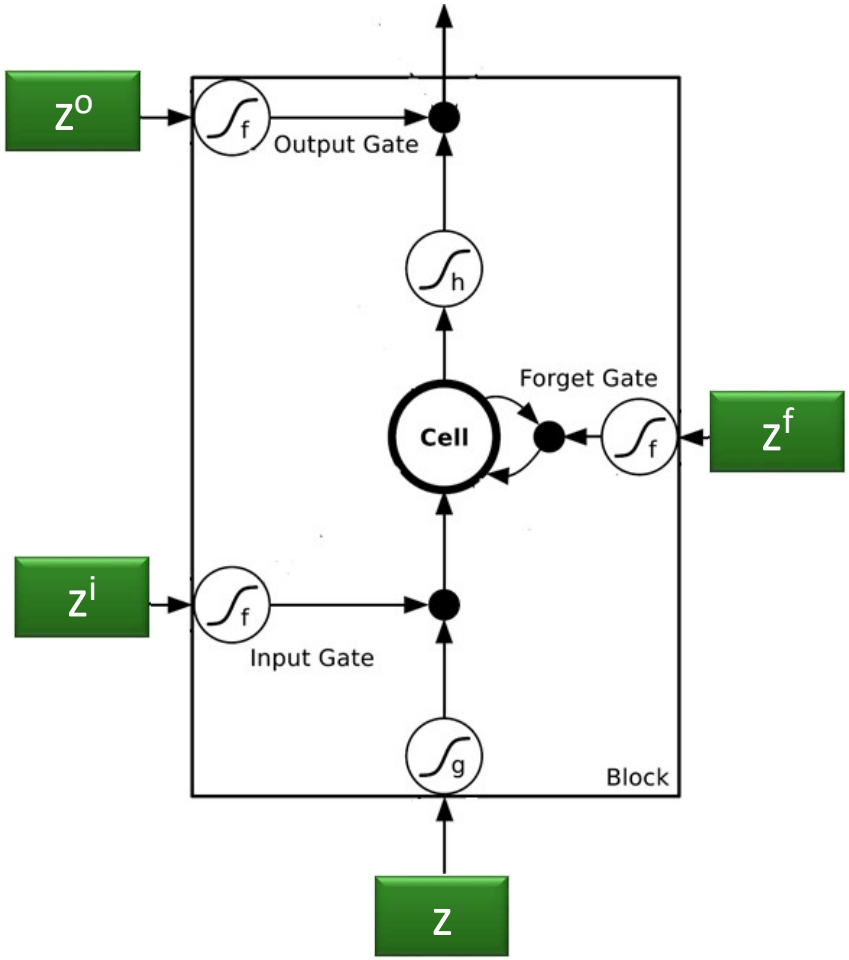
Between 0 and 1

Mimic open and close gate

$$c' = g(z)f(z_i) + cf(z_f)$$

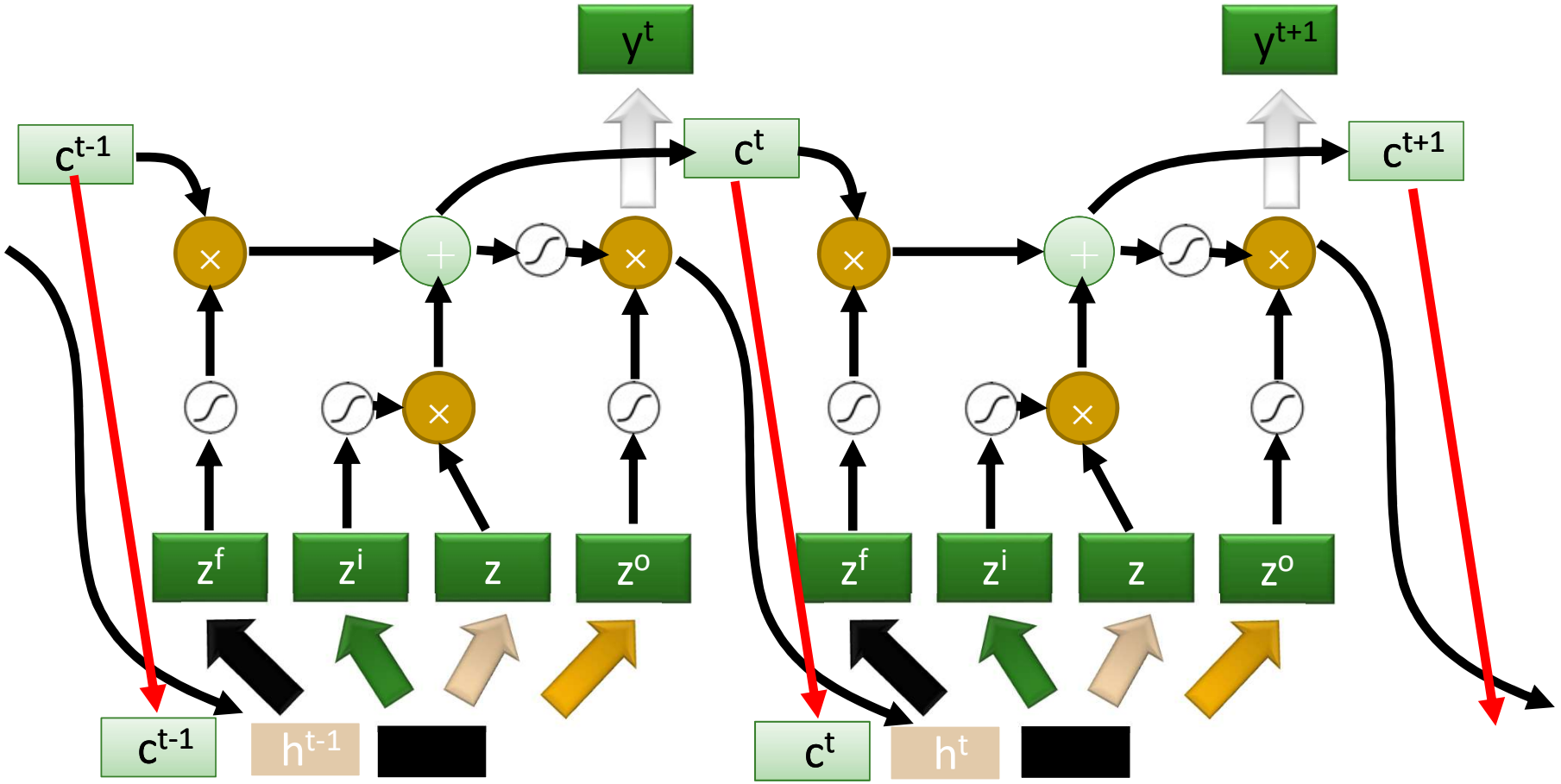
LSTM


y^t
c^{t-1}
z^f
z^i
z
z^o


z^o
Output Gate
Forget Gate
Cell
z^f
z^i
Input Gate
Block
z

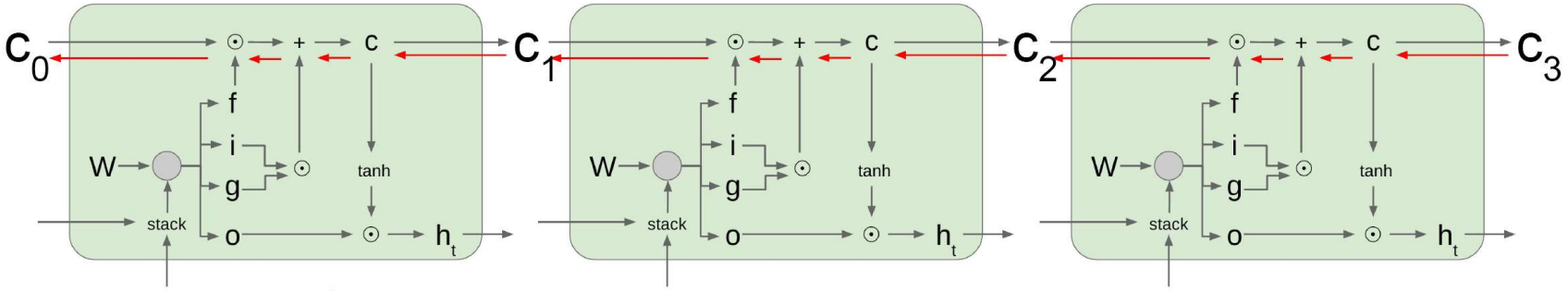
LSTM

Extension: “peephole”

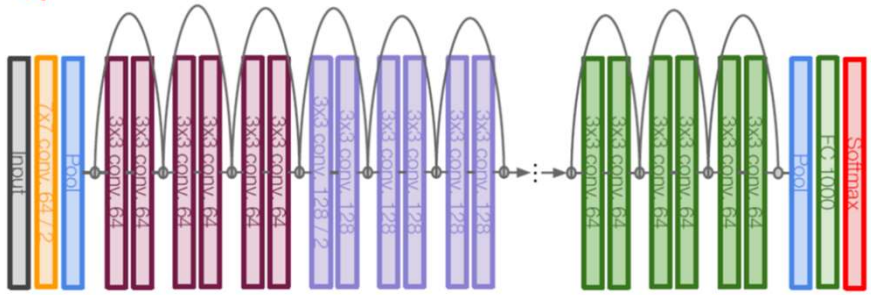


Long Short Term Memory (LSTM): Gradient Flow [Hochreiter et al., 1997]

Uninterrupted gradient flow!



Similar to ResNet!



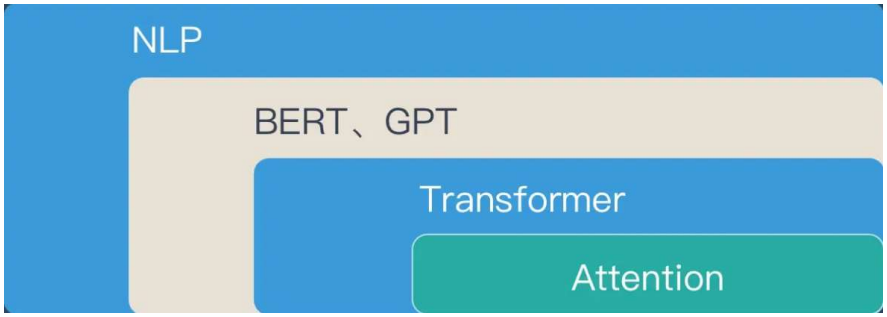
In between:
Highway Networks

$$g = T(x, W_T)$$
$$y = g \odot H(x, W_H) + (1 - g) \odot x$$

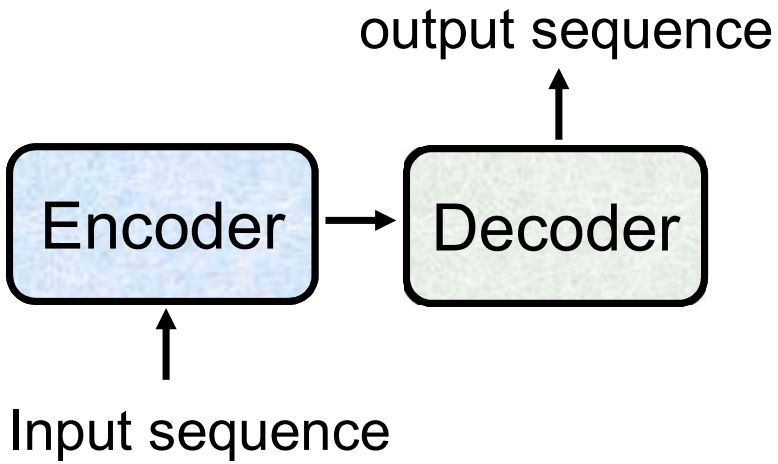
Srivastava et al, “Highway Networks”, ICML DL Workshop 2015

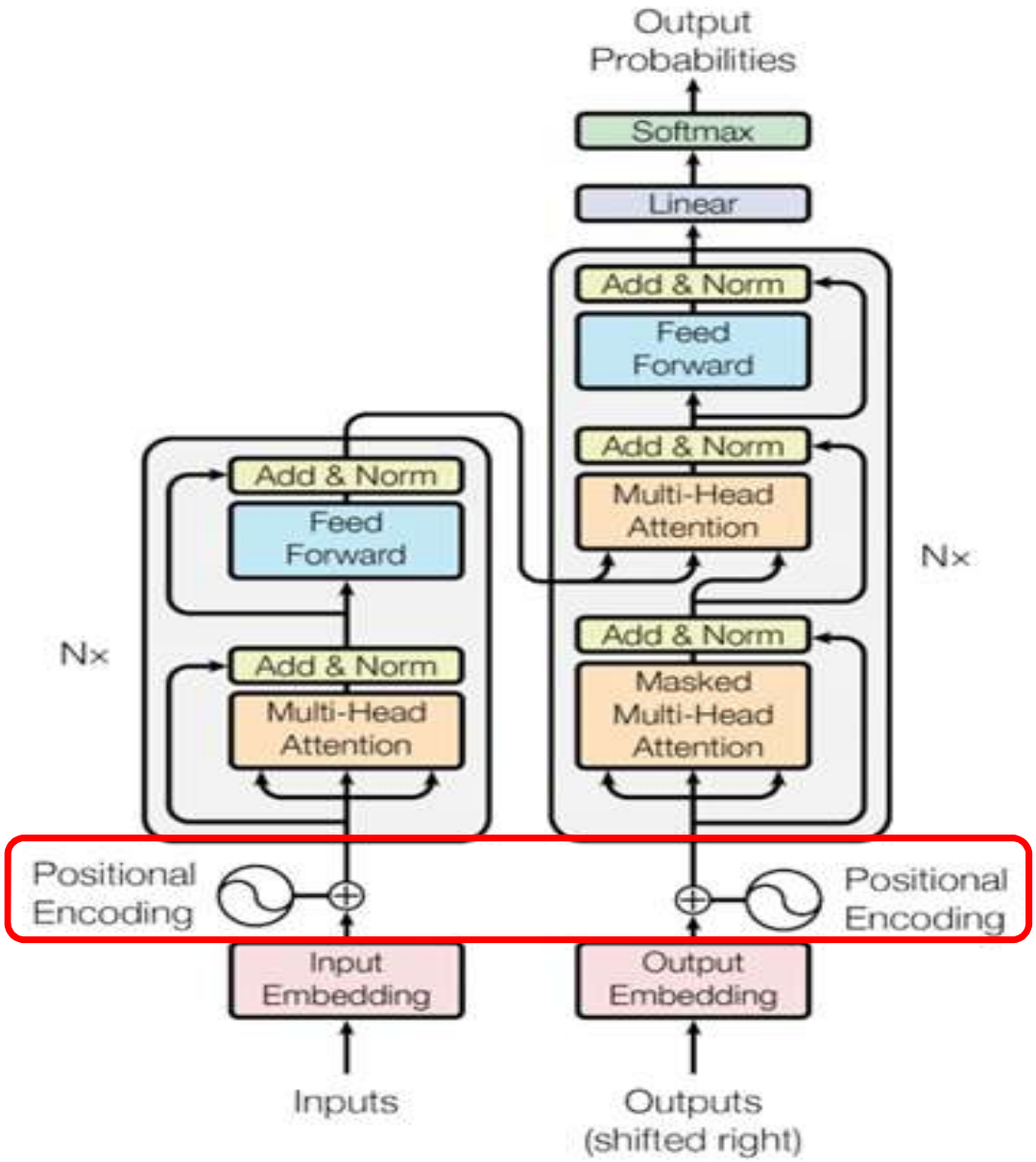
Outline

- Convolutional Neural Network (CNN)
- Recurrent Neural Network (RNN)
- Transformer
  - Self-attention
  - Encoder & Decoder
- Deep Learning Frameworks



Transformer

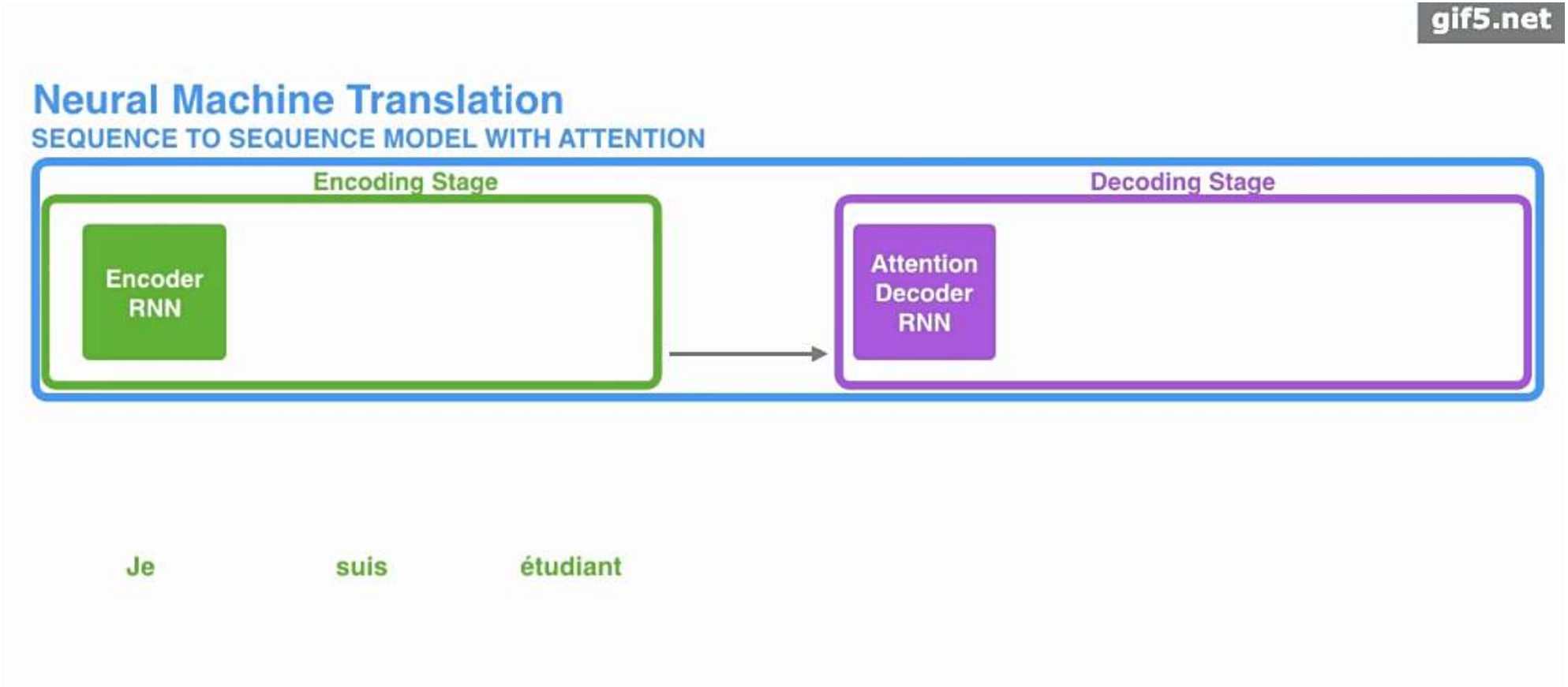




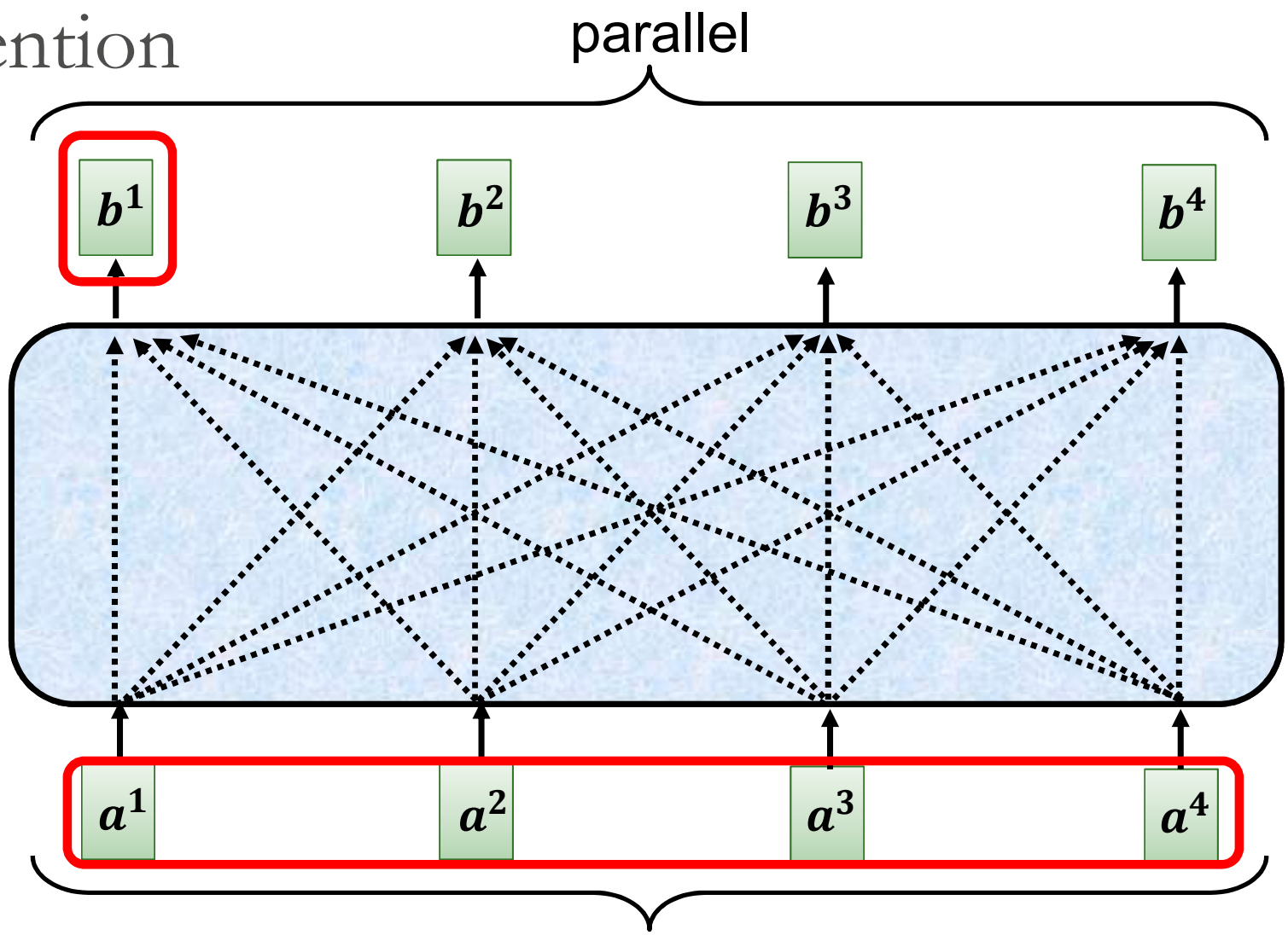
Attention Is All You Need
https://arxiv.org/abs/1706.03762

Attention

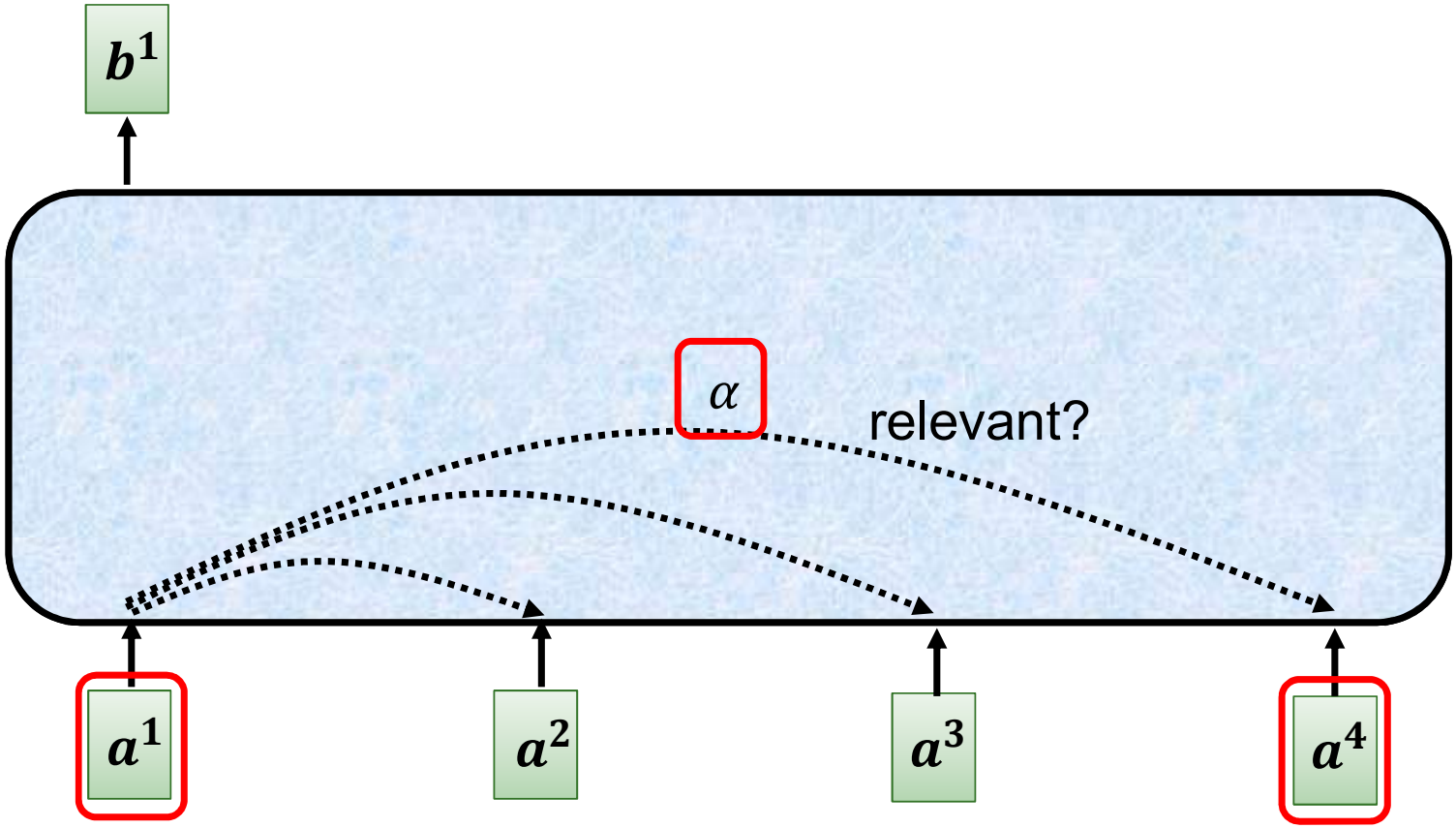
Sequence to sequence model with attention



Self-attention



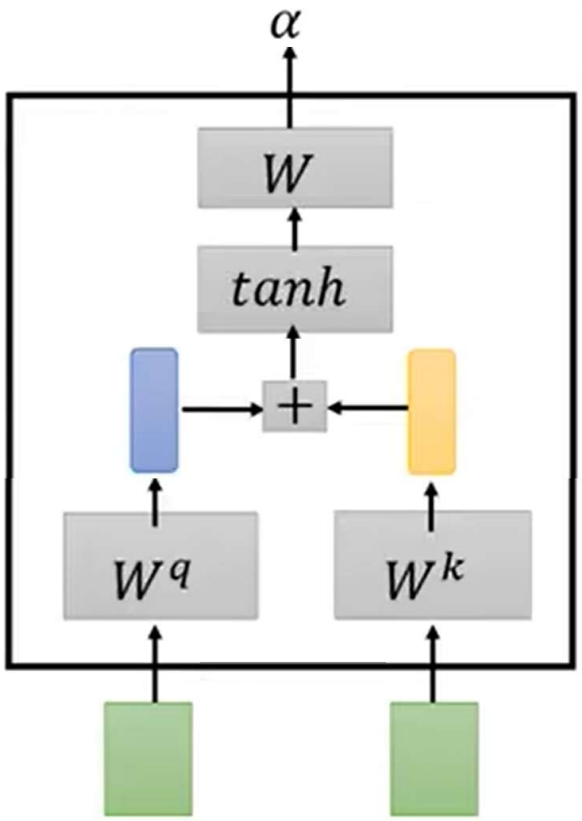
Self-attention



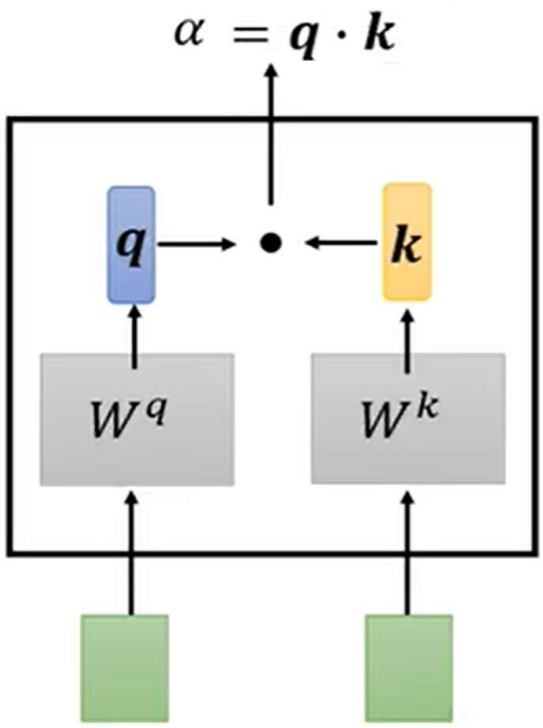
Find the relevant vectors in a sequence

Self-attention

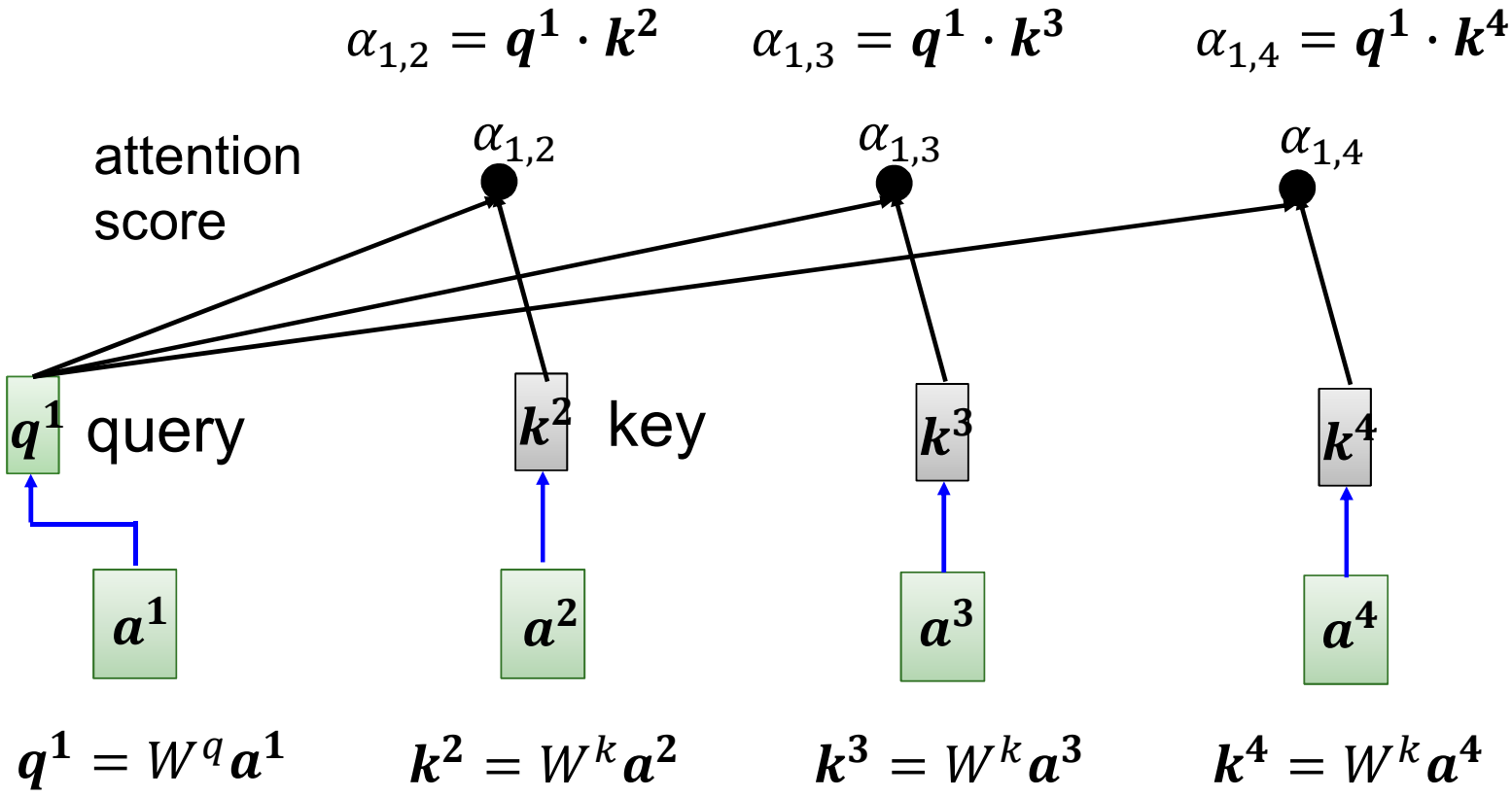
Additive



Dot-Product

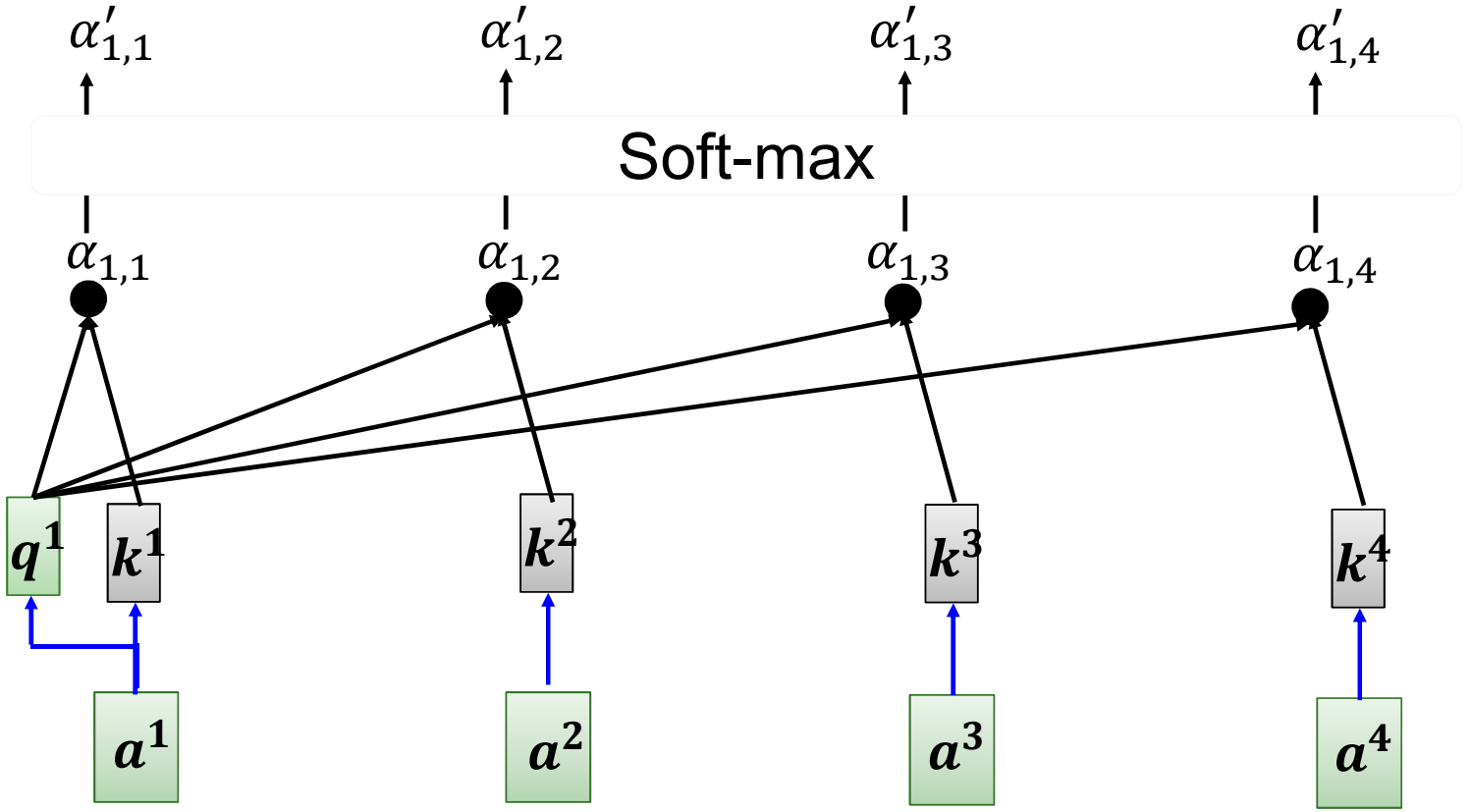


Self-attention



Self-attention

$$\alpha'_{1,i} = exp(\alpha_{1,i}) / \sum_j exp(\alpha_{1,j})$$

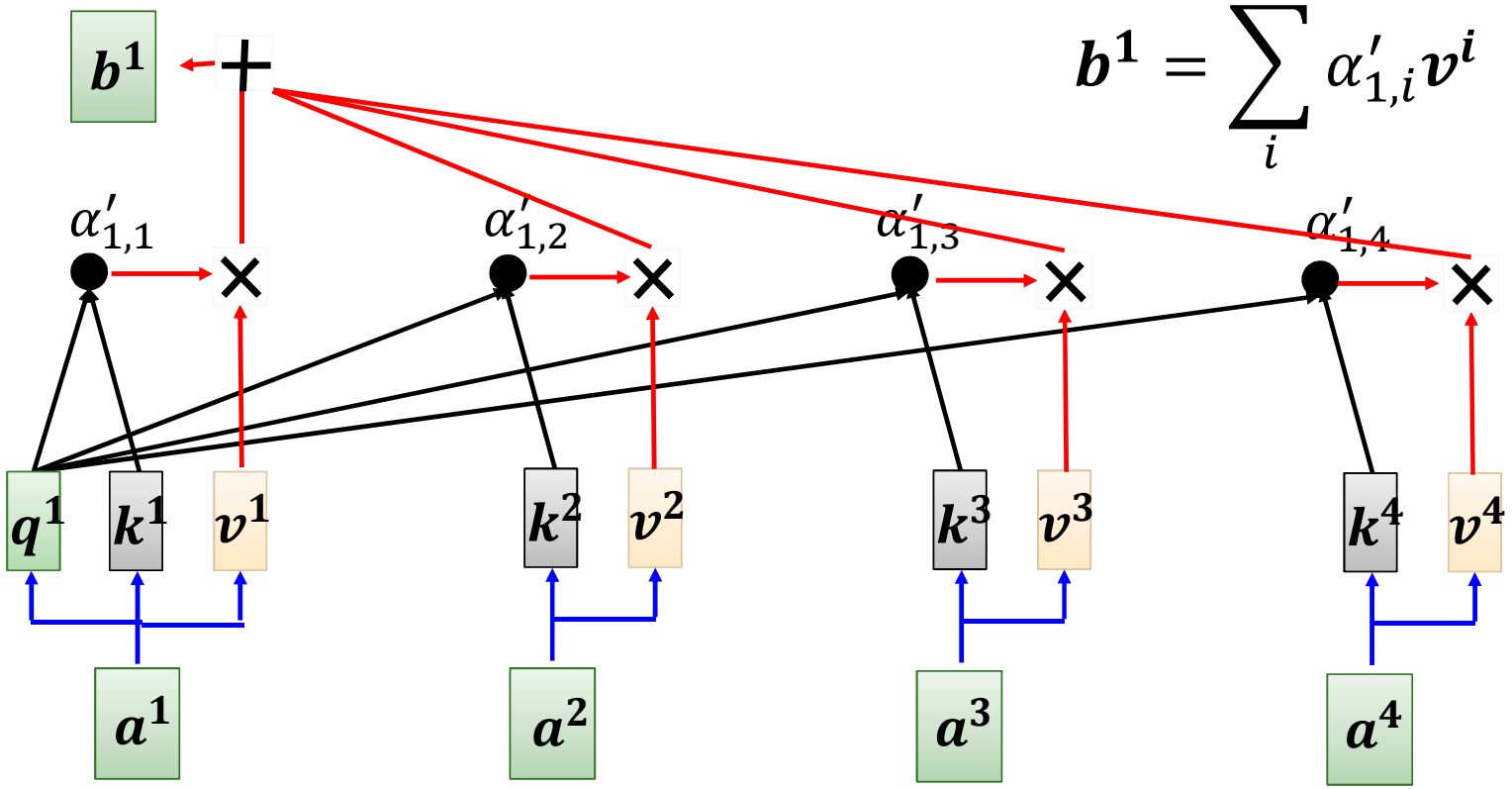


$q^1 = W^q a^1$ $\quad k^2 = W^k a^2$ $\quad k^3 = W^k a^3$ $\quad k^4 = W^k a^4$

$k^1 = W^k a^1$

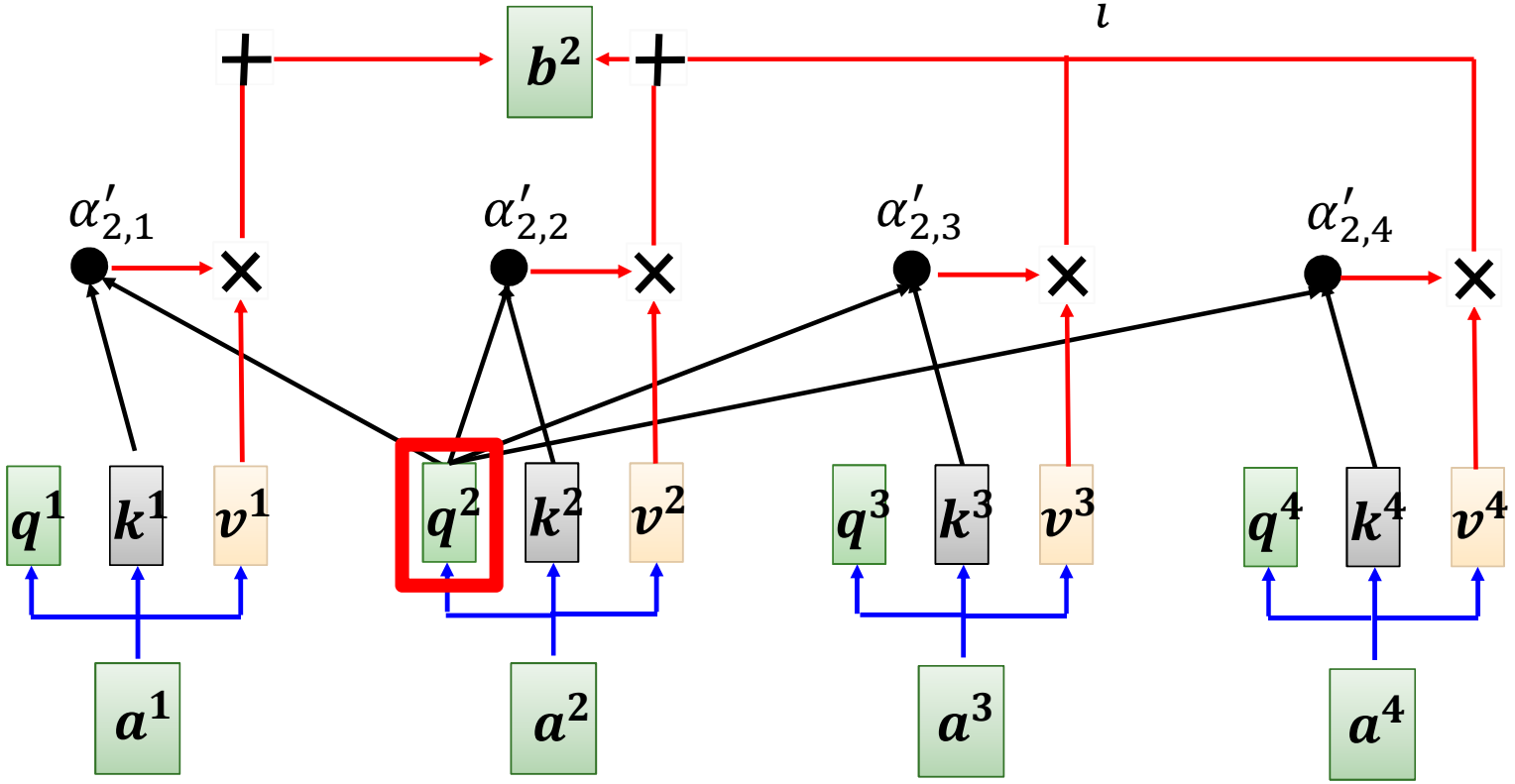
Self-attention

Extract information based on attention scores

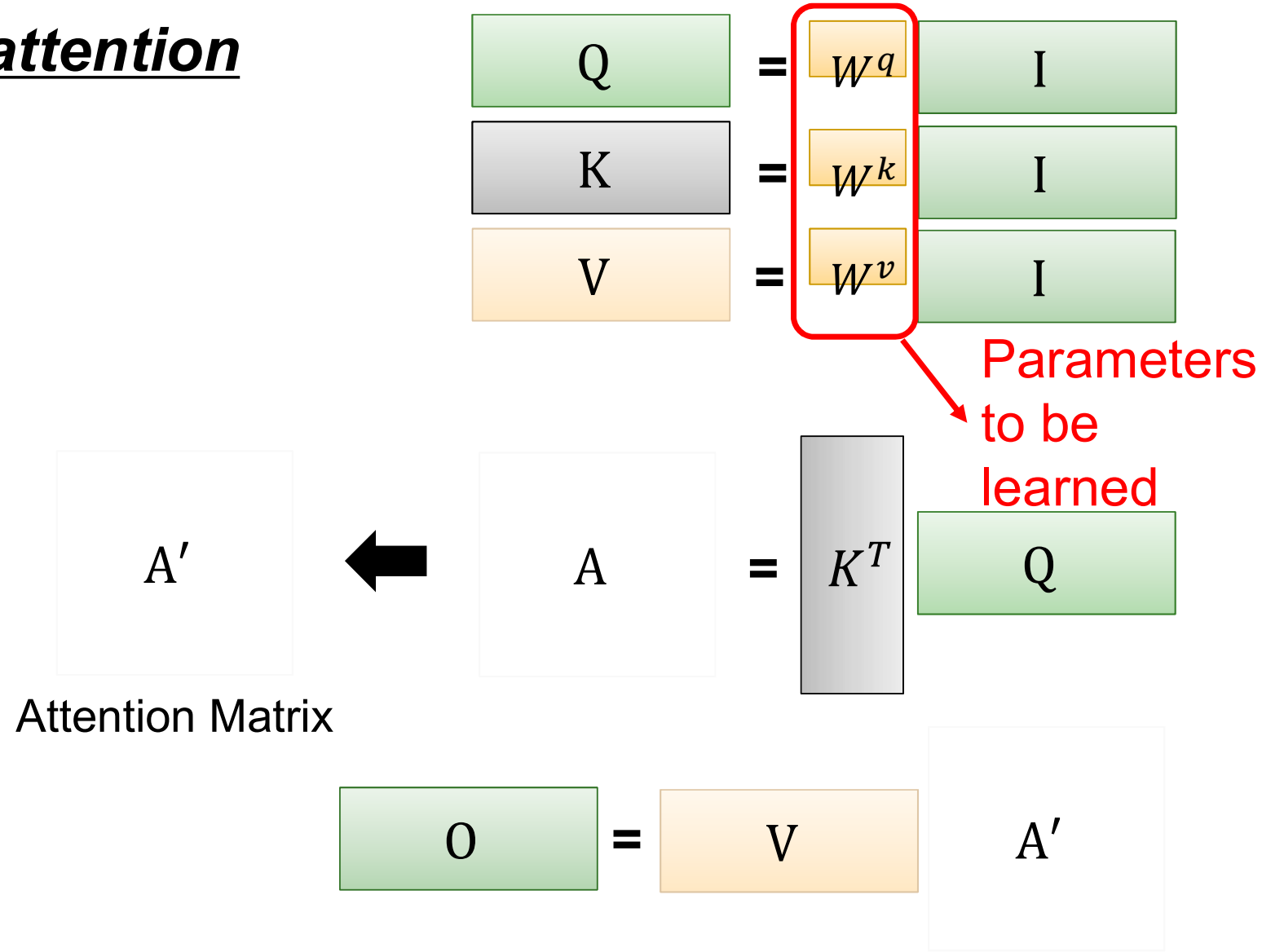


Self-attention

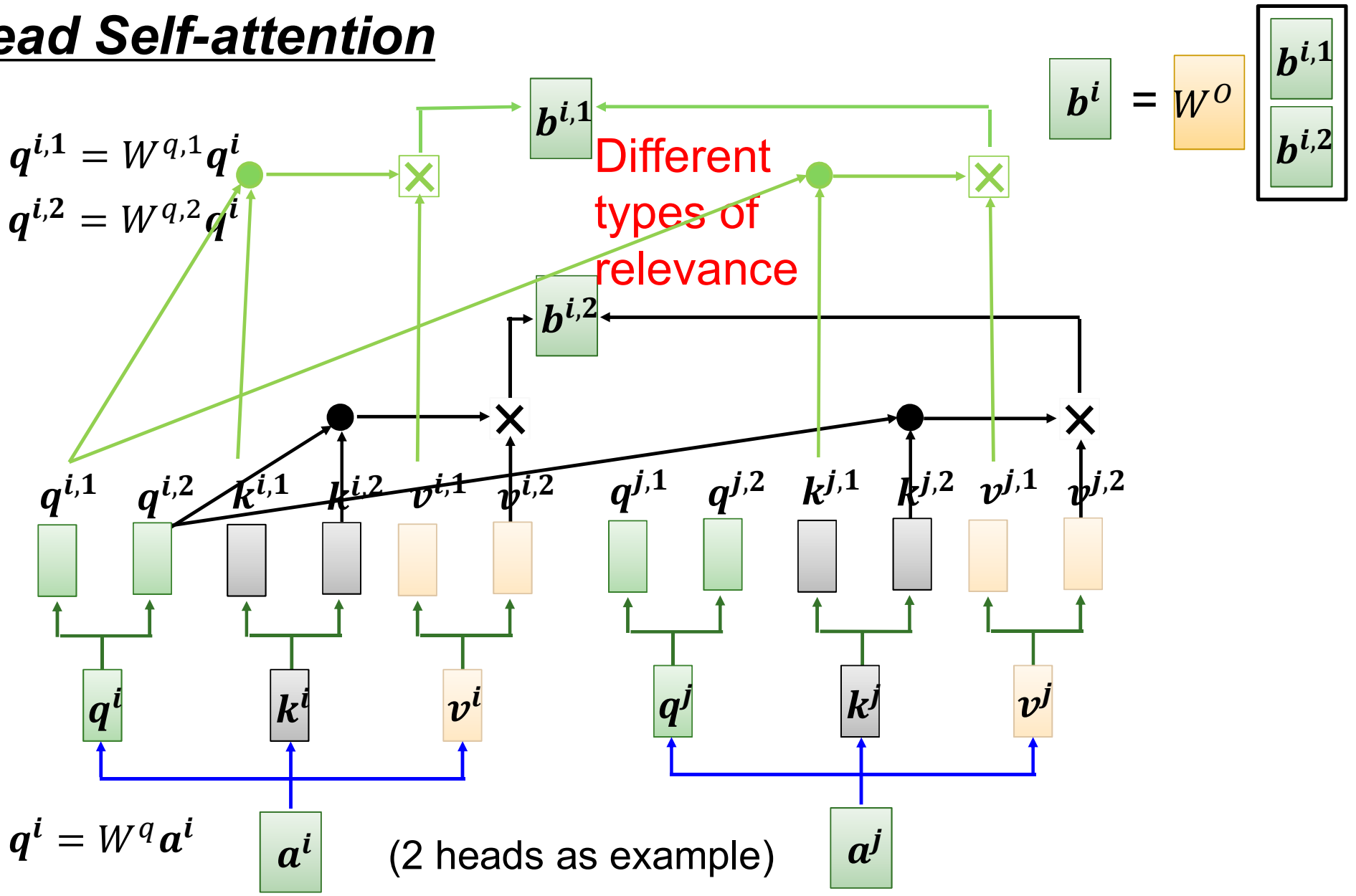
$$b^2 = \sum_i \alpha'_{2,i} v^i$$



Self-attention



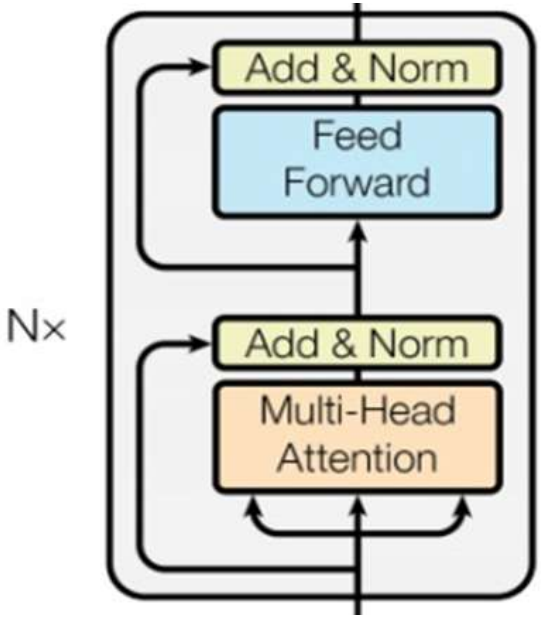
Multi-head Self-attention

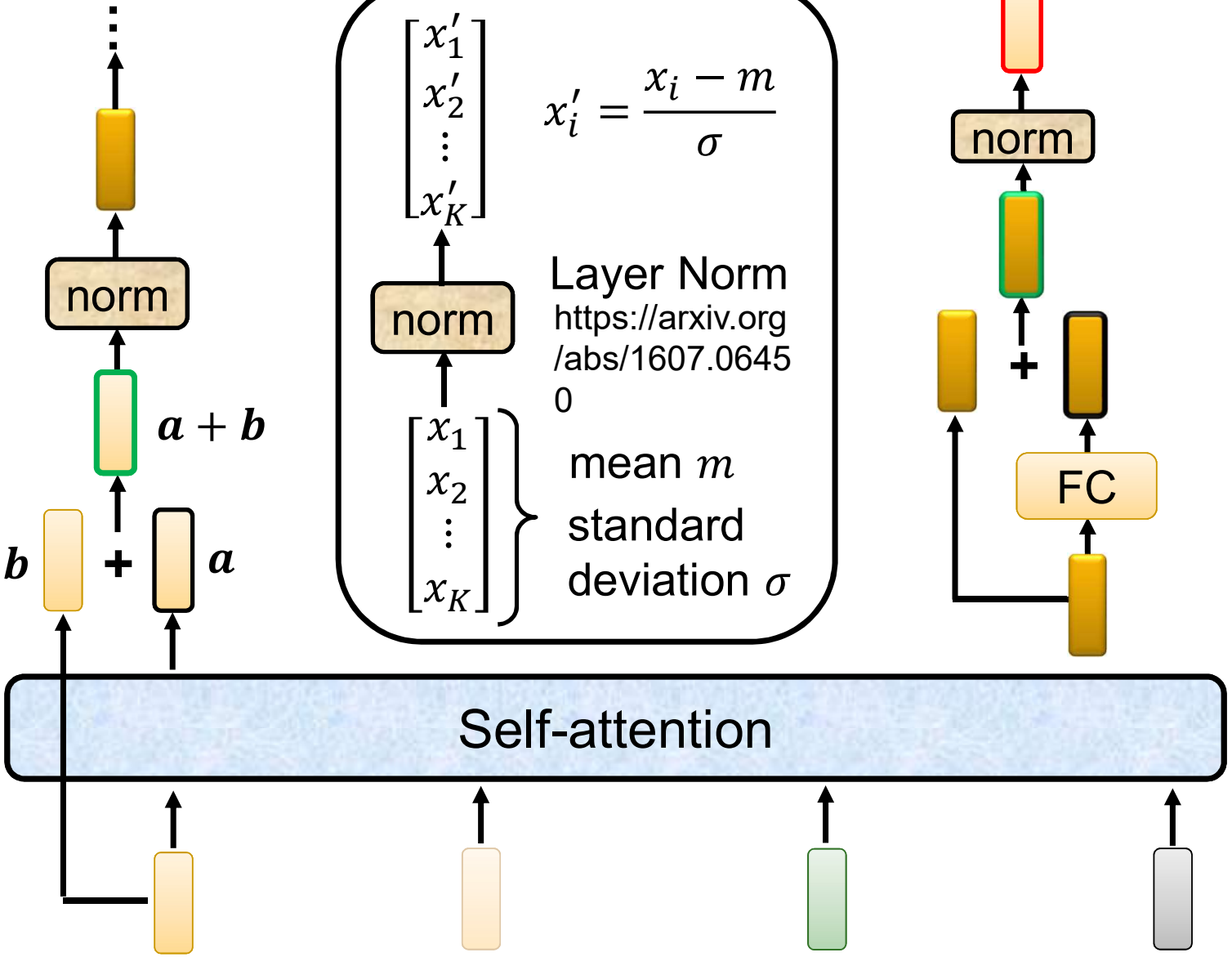


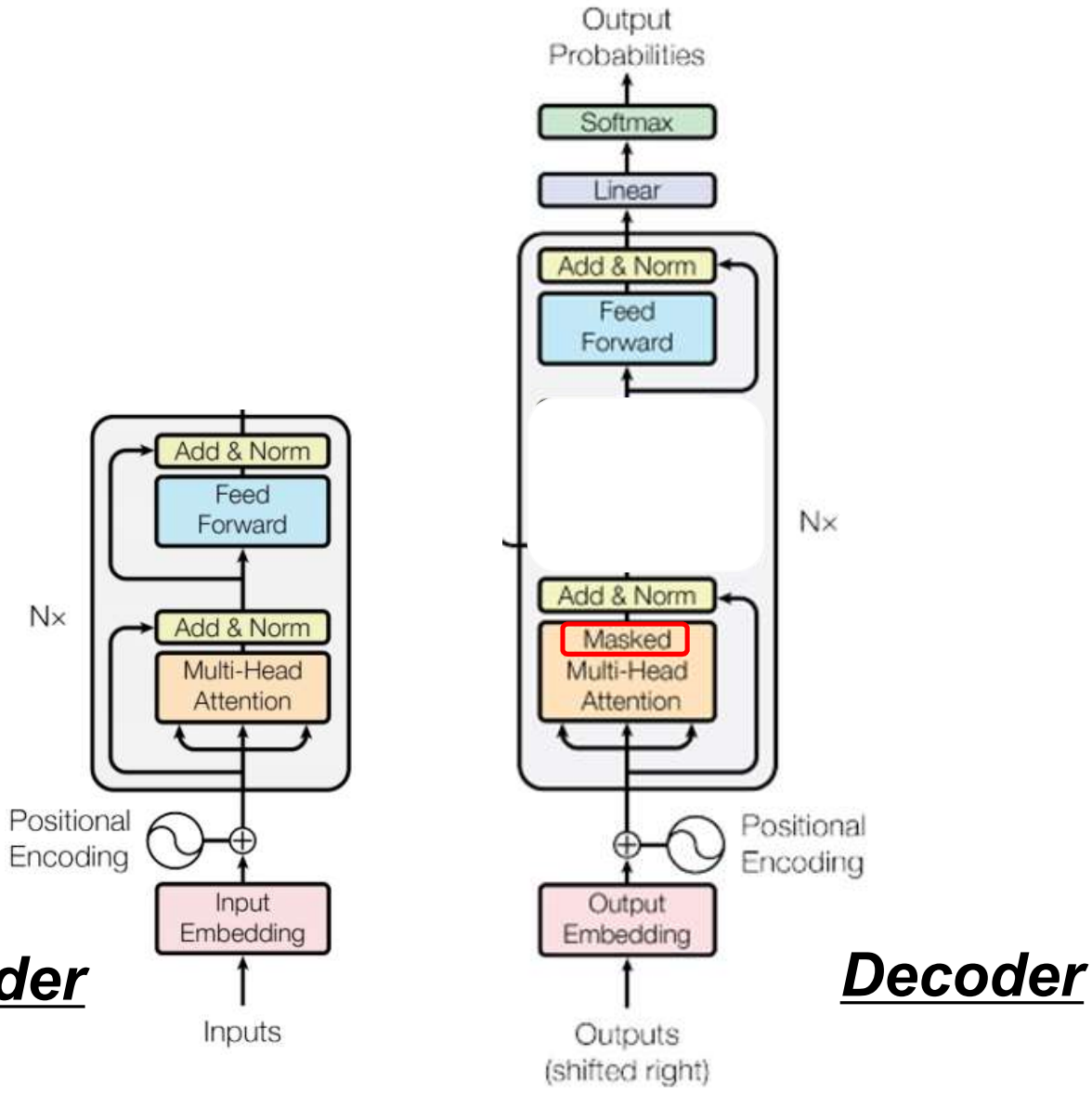
Outline

- Convolutional Neural Network (CNN)
- Recurrent Neural Network (RNN)
- Transformer
  - Self-attention
  - Encoder & Decoder
- Deep Learning Frameworks

Encoder

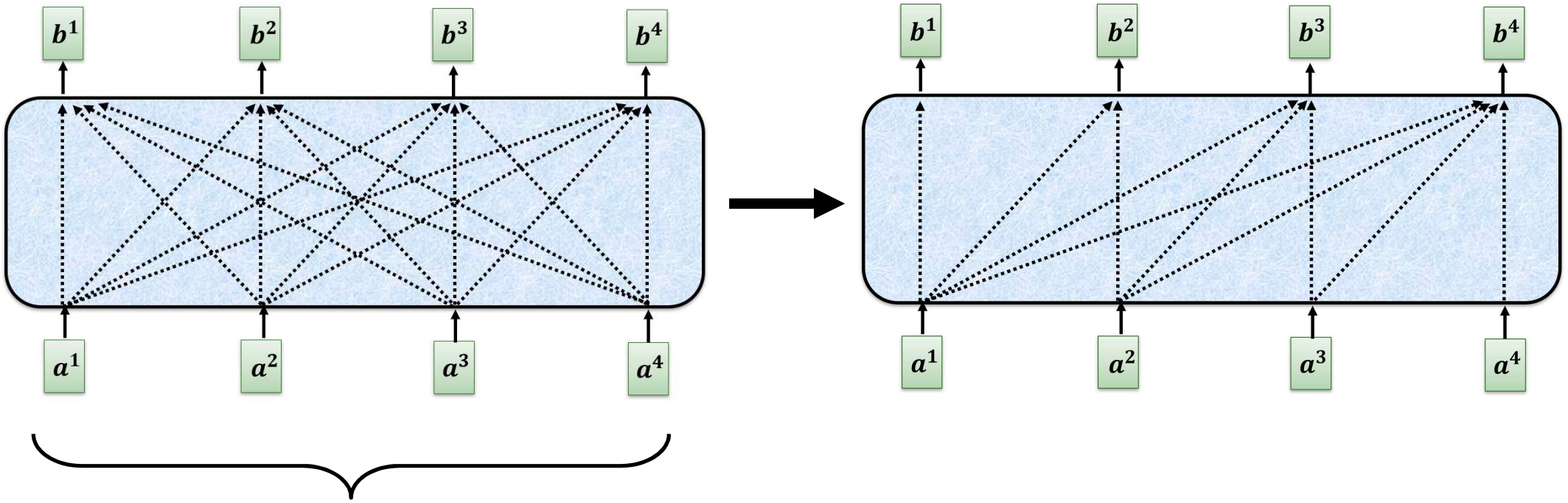

N×
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention


norm
$a+b$
b
+
a
Self-attention
$\begin{bmatrix} x_1' \\ x_2' \\ \vdots \\ x_K' \end{bmatrix}$
$x_i' = \frac{x_i - m}{\sigma}$
norm
Layer Norm
https://arxiv.org/abs/1607.06450
$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_K \end{bmatrix}$
mean m
standard deviation σ
norm
+
FC



Encoder ***Decoder***

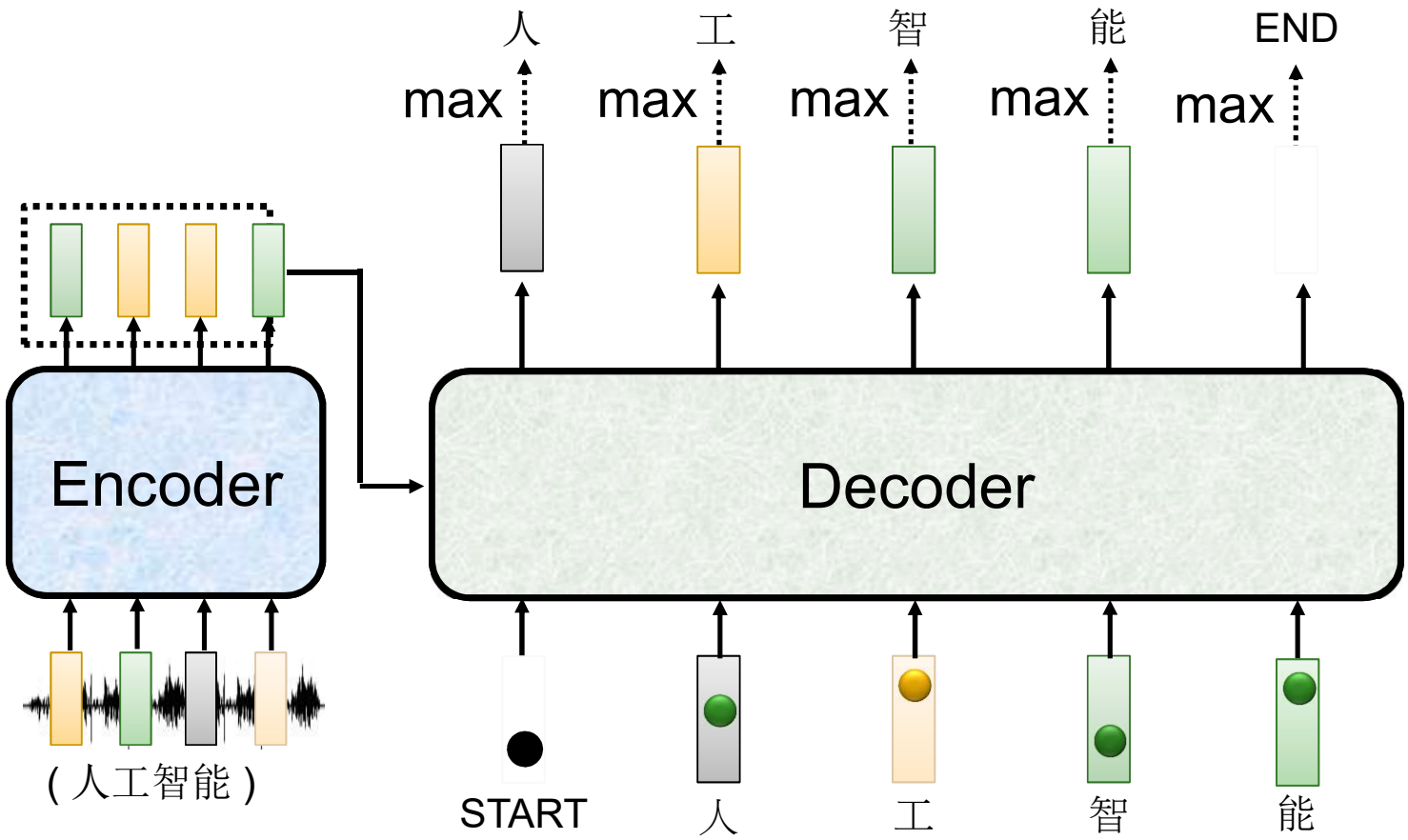
Self-attention → ***Masked Self-attention***



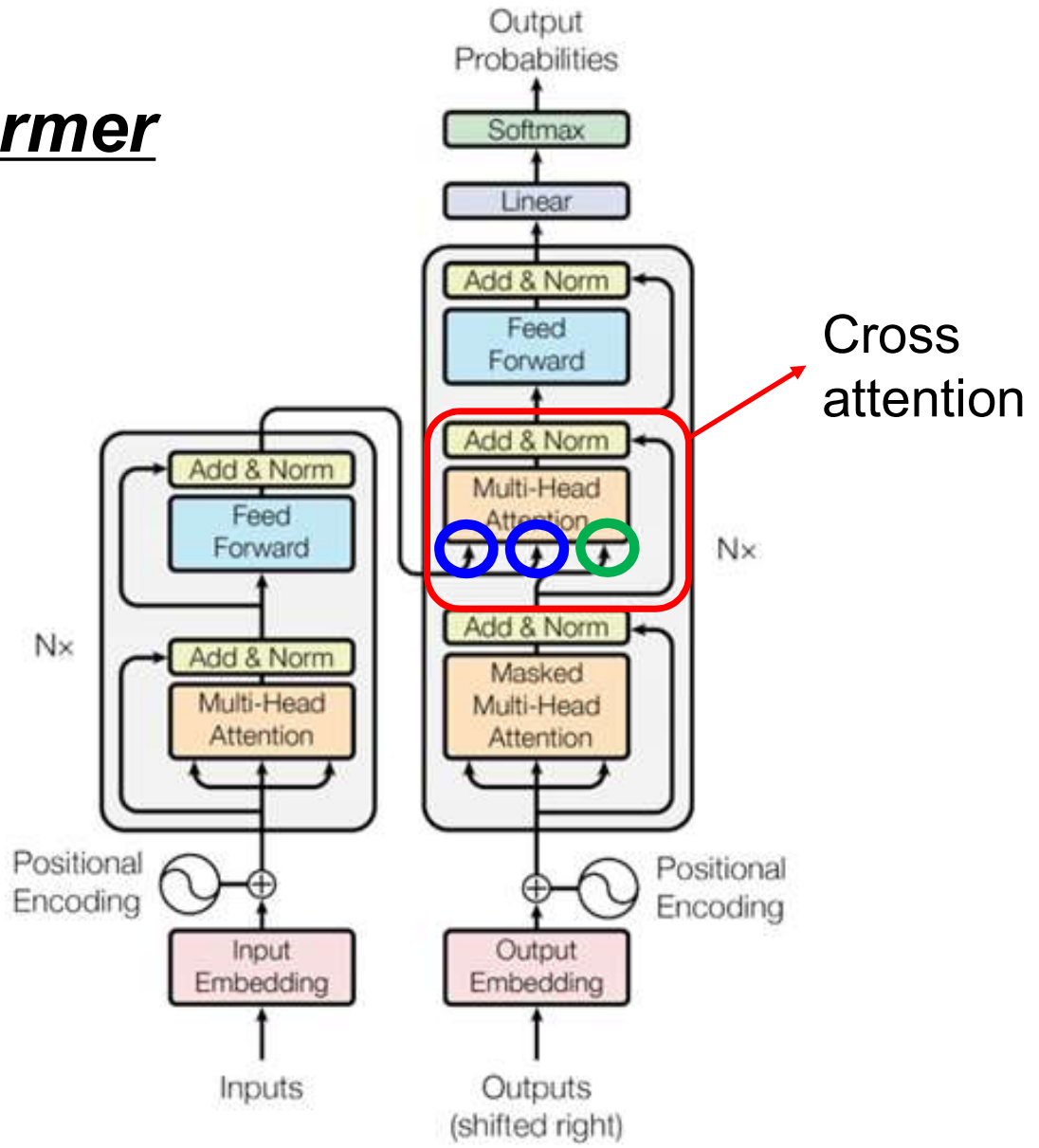
Can be either **input** or **a hidden layer**

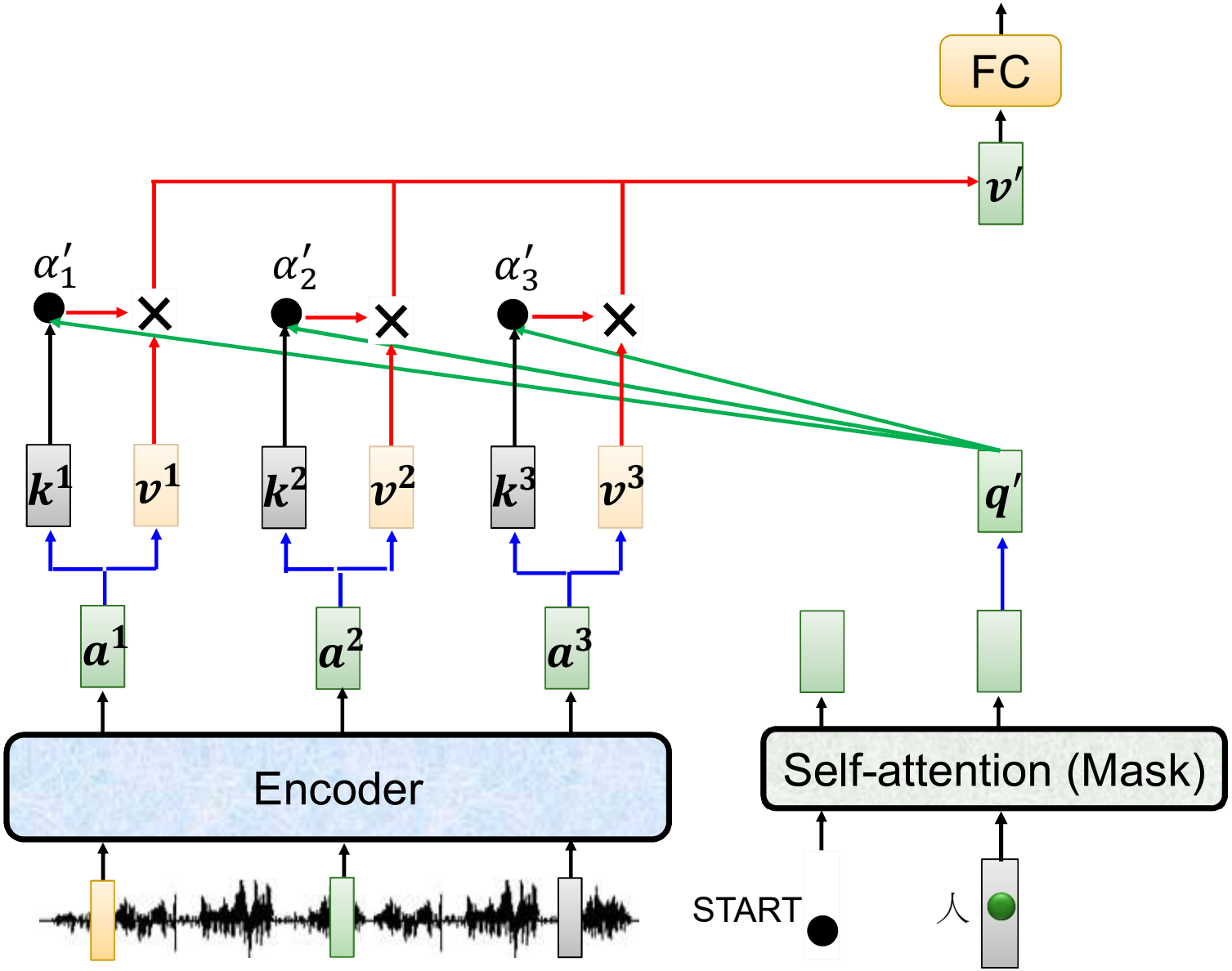
Autoregressive

Stop at here!



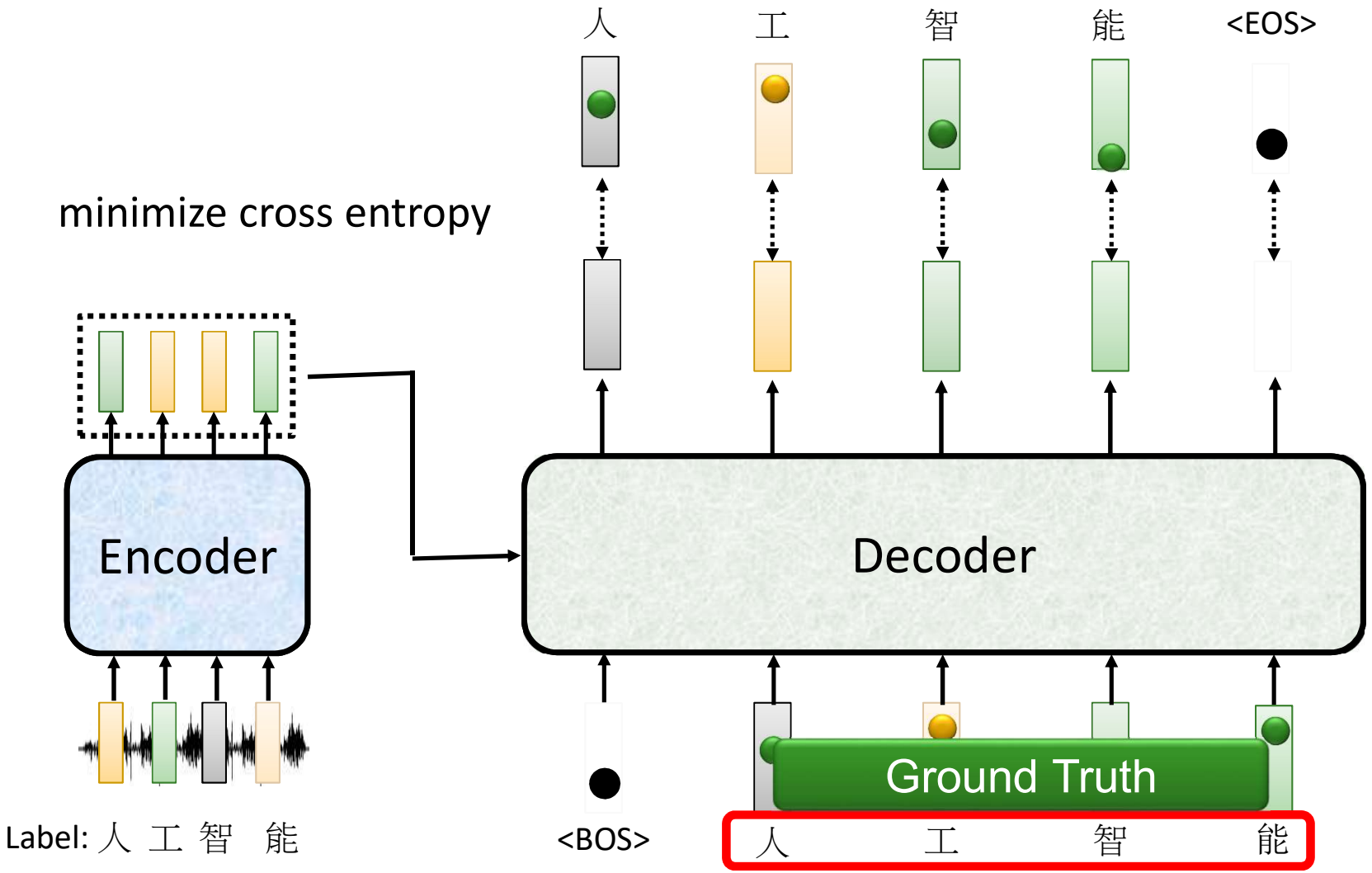
Transformer




FC
v'
α'_1
α'_2
α'_3
k^1
v^1
k^2
v^2
k^3
v^3
q'
a^1
a^2
a^3
Encoder
Self-attention (Mask)
START
人

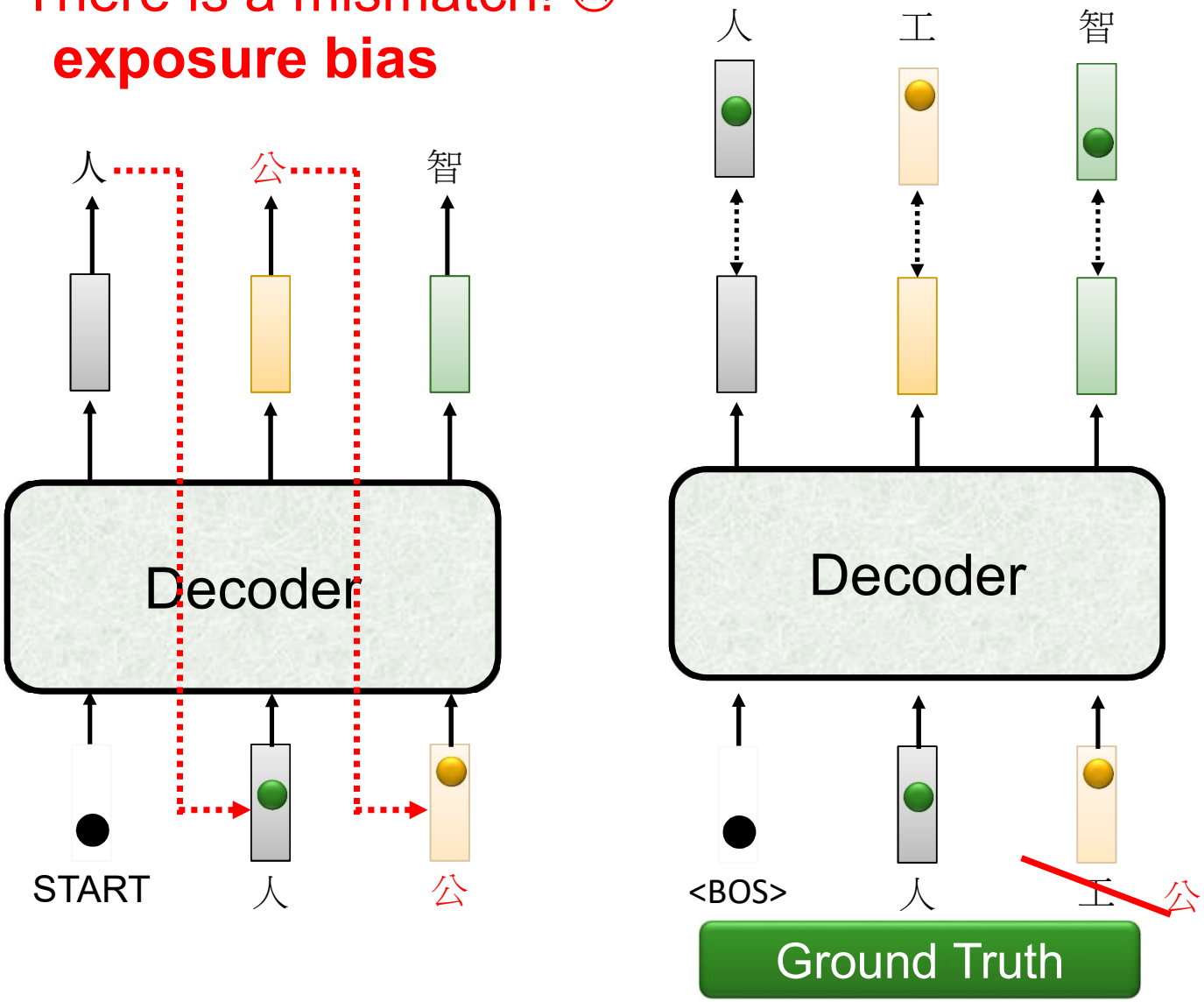
Teacher Forcing: using the ground truth as input.

Training



There is a mismatch! ☹

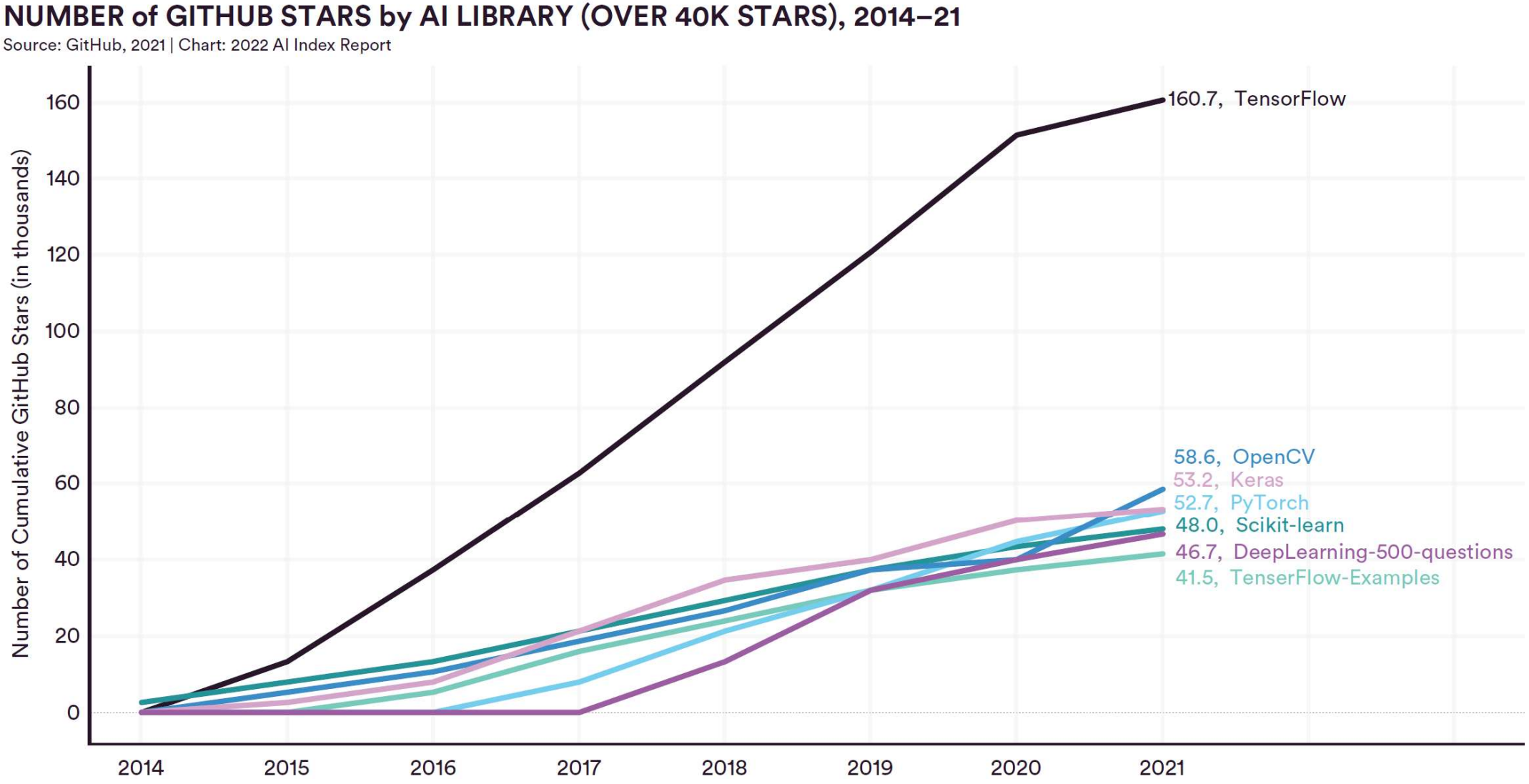
exposure bias



Outline

- Convolutional Neural Network (CNN)
- Recurrent Neural Network (RNN)
- Transformer
- Deep Learning Frameworks

Deep Learning Frameworks



from “The AI Index report 2022”

Q & A