DS4023（2025 Fall）
Machine Learning
Department of Computer Science
Beijing Normal-Hong Kong Baptist University

Chapter 5: Deep Learning

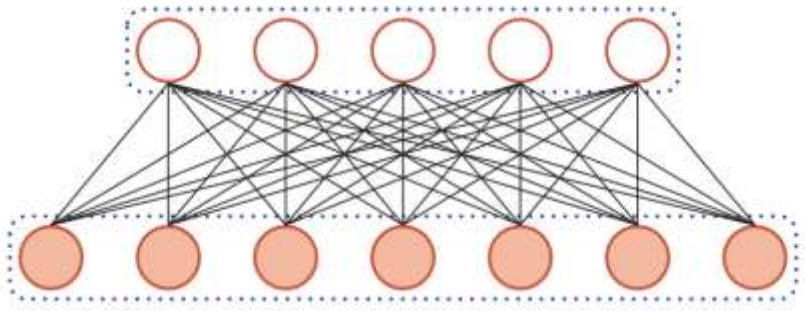
Outline

- Convolutional neural networks
- Recurrent neural networks

Chapter 5.1: Convolutional Neural Networks

Fully Connected Feedforward Neural Network

- Too many parameters in the weight matrix



- Local invariance
  - Objects in natural images all have local invariance characteristics
    - Scaling (缩放), translation (平移), rotation (旋转), and other operations do not affect its semantic information (语义信息)
  - Difficult for a fully connected FNN to extract these locally invariant features.

Convolutional Neural Networks

- Convolutional Neural Networks (CNN)
  - A kind of FNN
  - Put forward by the mechanism of Receptive Field in biology
    - In the visual nervous system, the receptive field of a neuron refers to a specific area on the retina, and only the stimulation in this area can activate the neuron.

- CNN has three structural characteristics:
  - Local connection (局部连接)
  - Parameter sharing (权重共享)
  - Sub-sampling in space or time (空间或时间上的次采样)

Content

- 5.1.1 Convolution
- 5.1.2 Convolutional Neural Networks
- 5.1.3 Other convolution types
- 5.1.4 Typical convolutional network
- 5.1.5 Application of Convolution Network
- 5.1.6 Apply to text data

5.1.1 Convolution

Convolution

- Convolution is often used in signal processing to calculate the delay accumulation of signals.
- Suppose a signal generator generates a signal x_t at every time t, and the attenuation rate of its information is w_k, that is, after k-1 time steps, the information is w_k times the original one
  - Suppose $w_1 = 1, w_2 = 1/2, w_3 = 1/4$
  - The signal y_t received at the time t is the superposition of the information generated at the current time and the delay information at the previous time.

Convolution

$$
\begin{aligned}
y_t &= 1 \times x_t + 1/2 \times x_{t-1} + 1/4 \times x_{t-2} \\
&= w_1 \times x_t + w_2 \times x_{t-1} + w_3 \times x_{t-2} \\
&= \sum_{k=1}^{3} w_k \cdot x_{t-k+1}.
\end{aligned}
$$

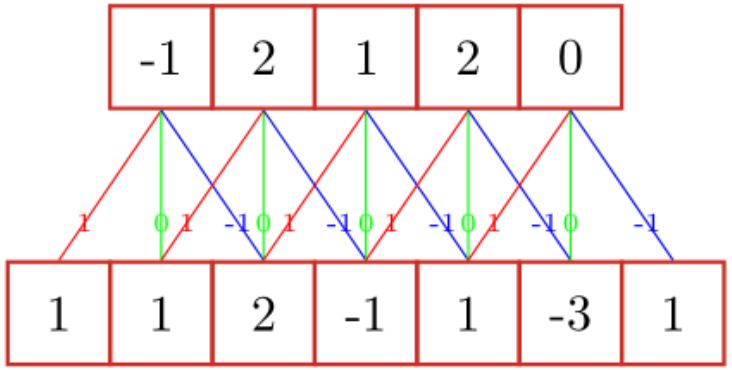
滤波器（filter）或卷积核（convolution kernel）

Convolution

- Given an input signal sequence x and a filter w, the output of convolution is:

$$y_t = \sum_{k=1}^{K} w_k x_{t-k+1} \qquad y = x * w$$

Filter: [-1,0,1]
Convolution kernel flip



Function of Convolution

- Approximate differentiation
  - When filer $w = \left[\frac{1}{2}, 0, -\frac{1}{2}\right]$, the first order differential of signal sequence can be approximated.

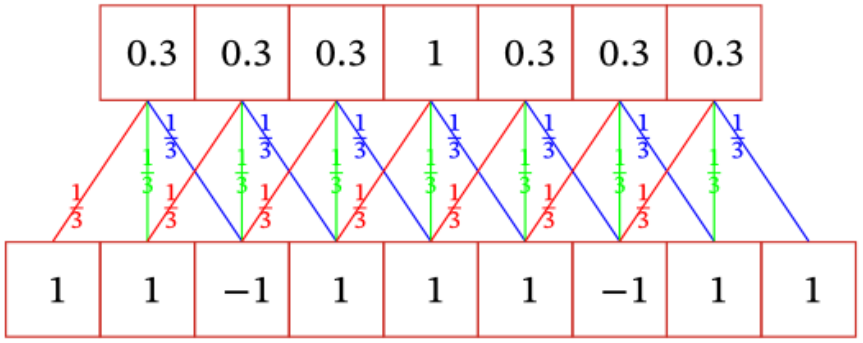
$$x'(t) = \frac{x(t+1) - x(t-1)}{2}$$

  - When filer $w = [1, -2, 1]$, the second order differential of signal sequence can be approximated.

$$x''(t) = x(t+1) + x(t-1) - 2x(t)$$

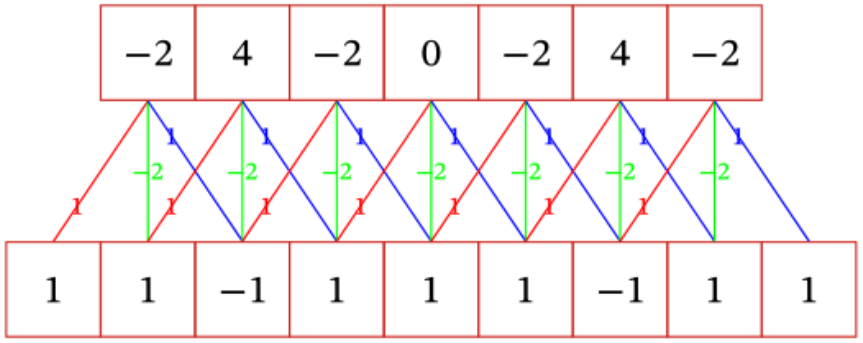
Convolution

- Different filters are used to extract different features in the signal sequence:
  - Filter $w = [1/3, 1/3, 1/3]$ can detect low frequency information
  - Filter $w = [1, -2, 1]$ can detect high frequency information



(a) 滤波器 [1/3, 1/3, 1/3]

Low frequency info

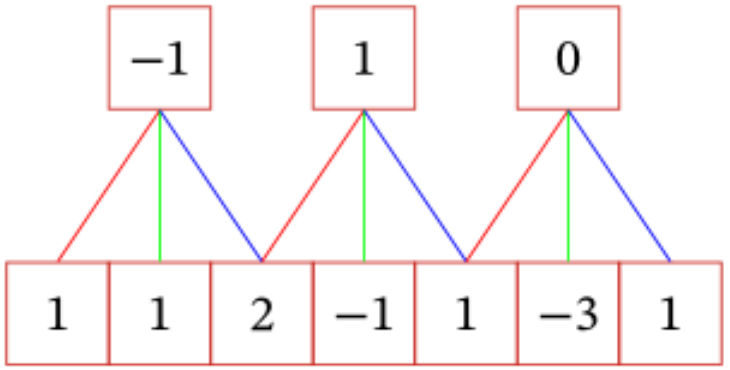


(b) 滤波器 [1, −2, 1]

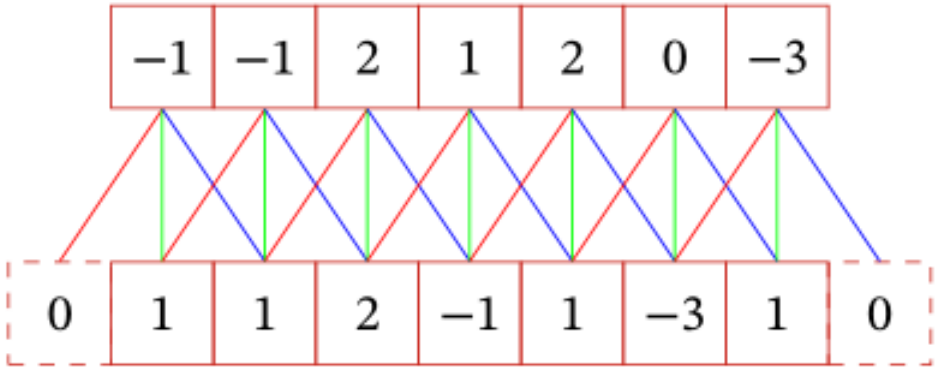
High frequency info

Convolution expansion

- Filter's stride (步长) S and zero padding (填充) P
- Kernel size K
- The number of neurons M



(a) 步长 $S = 2$



(b) 零填充 $P = 1$

Convolution Type

- Three types according to different output lengths:
  - Narrow convolution: $S = 1$, $P = 0$, output $M - K + 1$
  - Wide convolution: $S = 1$, $P = K - 1$, output $M + K - 1$
  - Equal-width convolution: $S = 1$, $P = (K - 1)/2$, output M

- In the early literature, narrow convolution by default
- In the current literature, equal-width convolution by default

Two-dimensional convolution

- In image processing, the image is input into NN in the form of 2-d matrix, so we need 2-d convolution.

一个输入信息 $\boldsymbol{X}$ 和滤波器 $\boldsymbol{W}$ 的二维卷积定义为

$$\boldsymbol{Y} = \boldsymbol{W} * \boldsymbol{X},$$

$$y_{ij} = \sum_{u=1}^{U} \sum_{v=1}^{V} w_{uv} x_{i-u+1, j-v+1}.$$

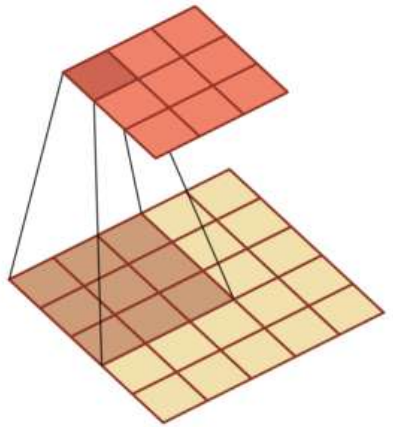
1	1	1 ×−1	1 ×0	1 ×0
−1	0	−3 ×0	0 ×0	1 ×0
2	1	1 ×0	−1 ×0	0 ×1
0	−1	1	2	1
1	2	1	1	1

$*$

1	0	0
0	0	0
0	0	−1

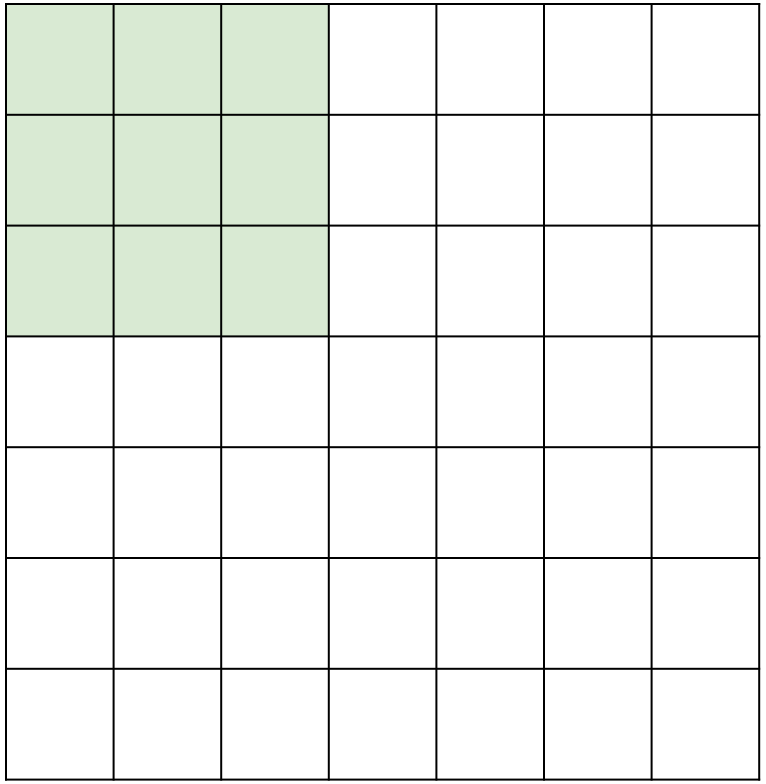
$=$

0	−2	−1
2	2	4
−1	0	0



A closer look at spatial dimensions:

7

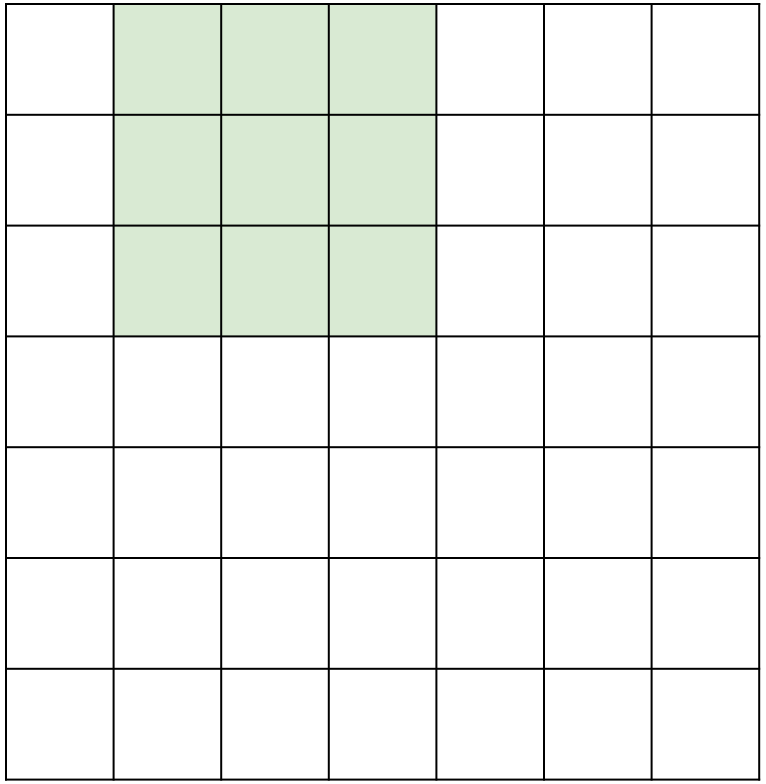


7

7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:

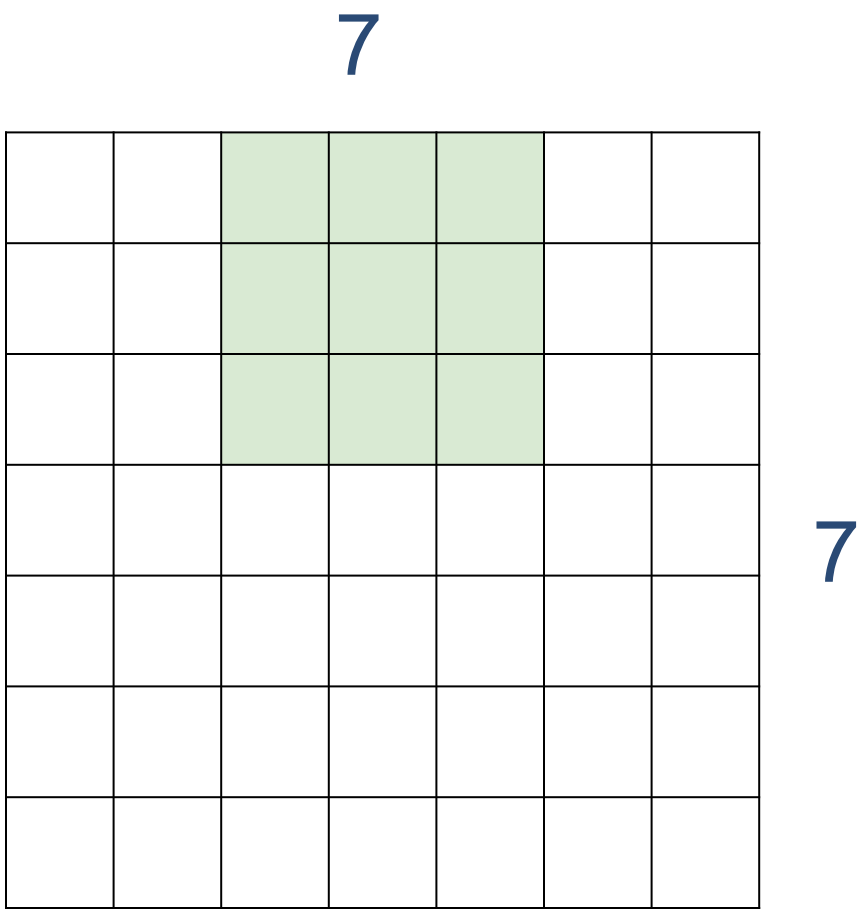
7



7

7x7 input (spatially)
assume 3x3 filter

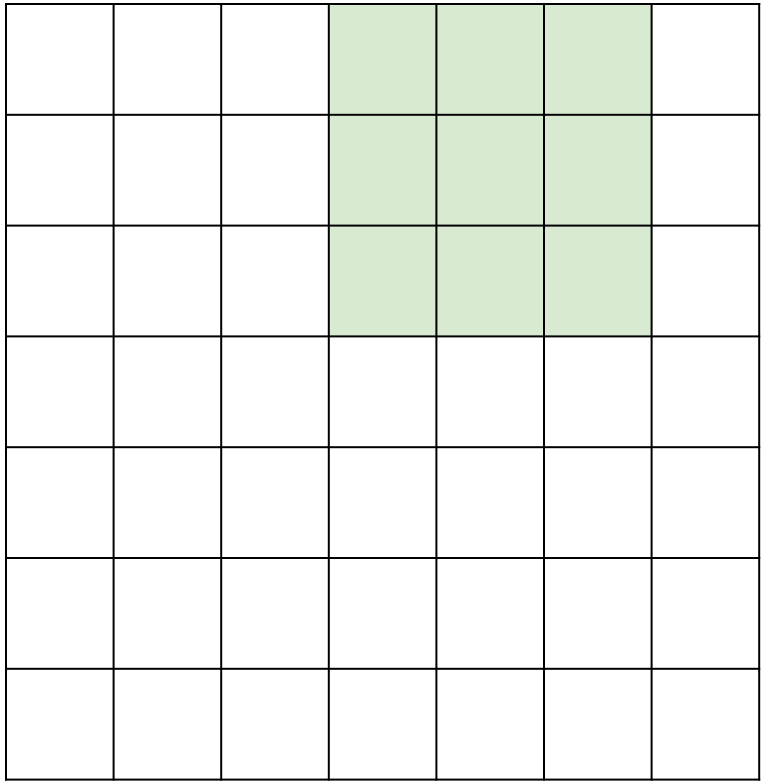
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:

7

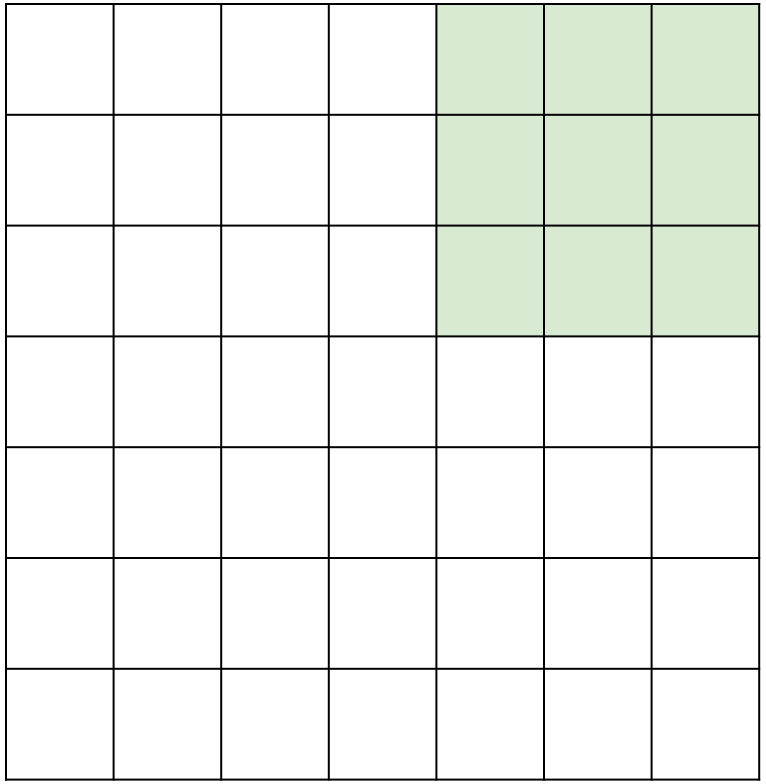


7

7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:

7

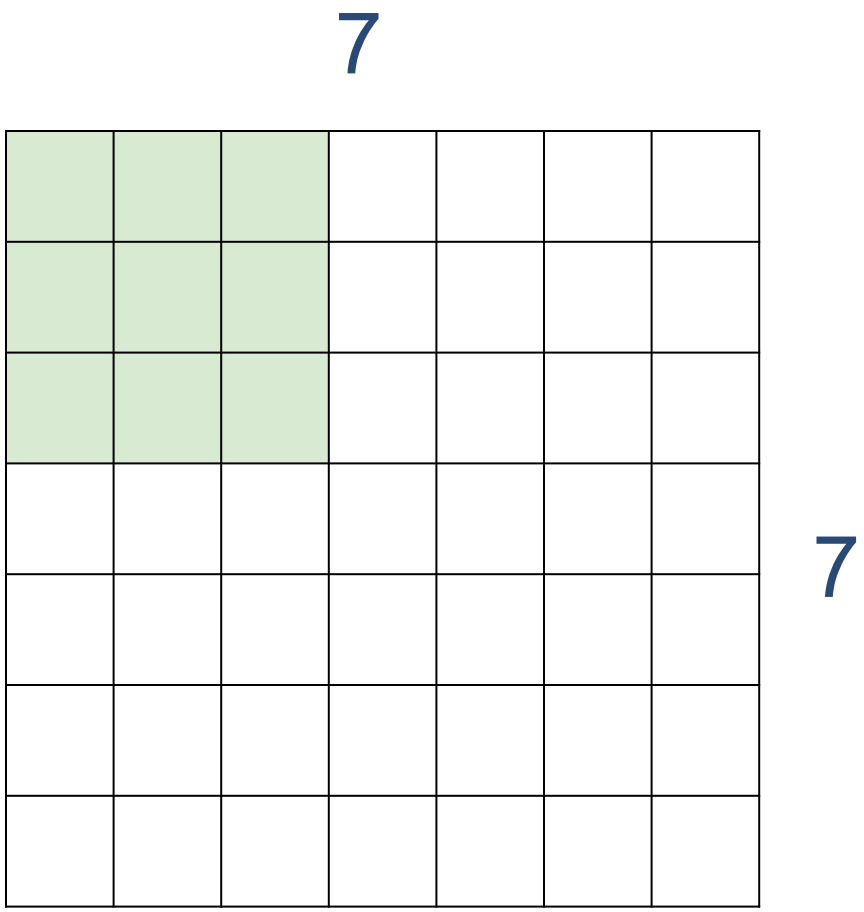


7

7x7 input (spatially)
assume 3x3 filter

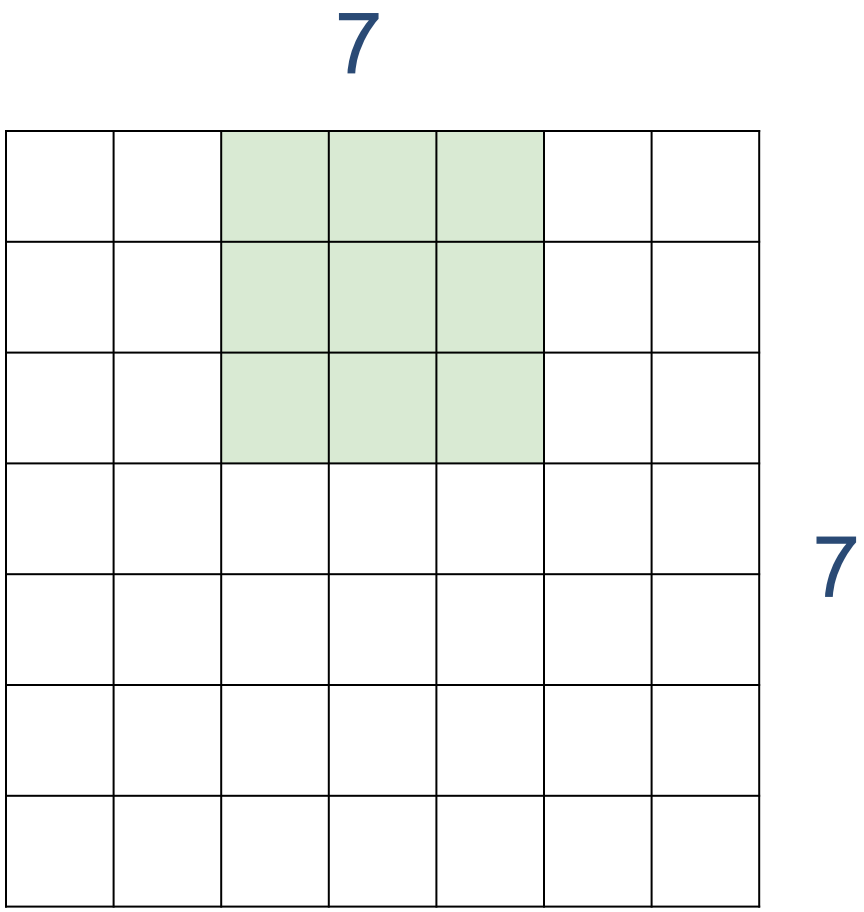
=> 5x5 output

A closer look at spatial dimensions:



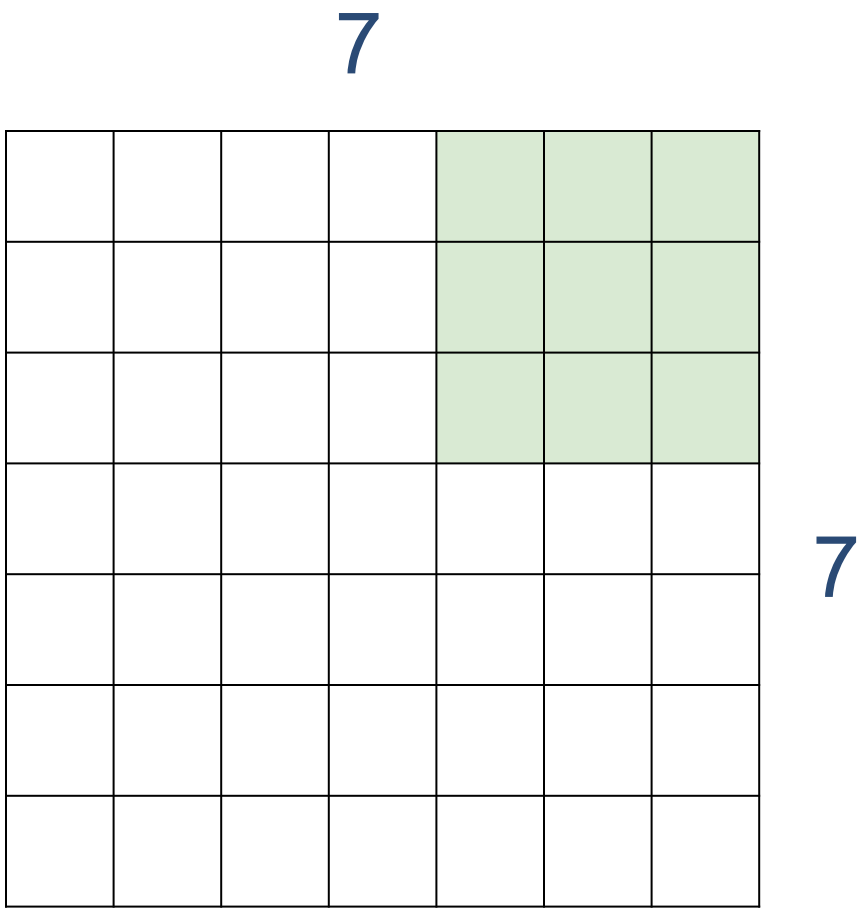
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimensions:



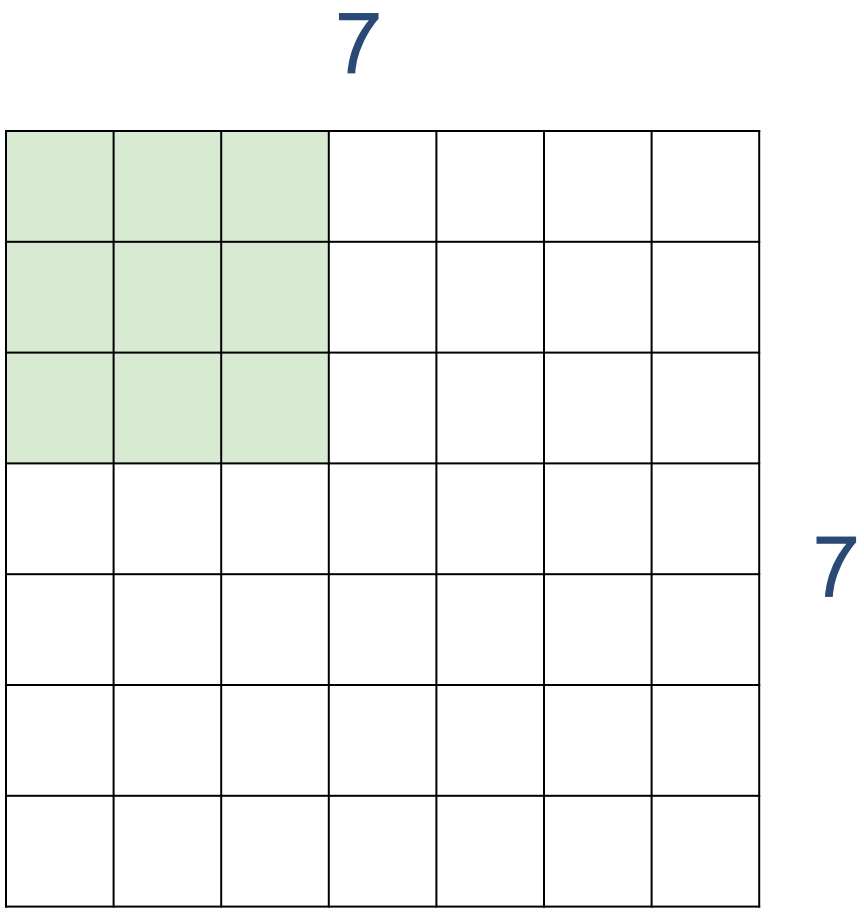
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimensions:



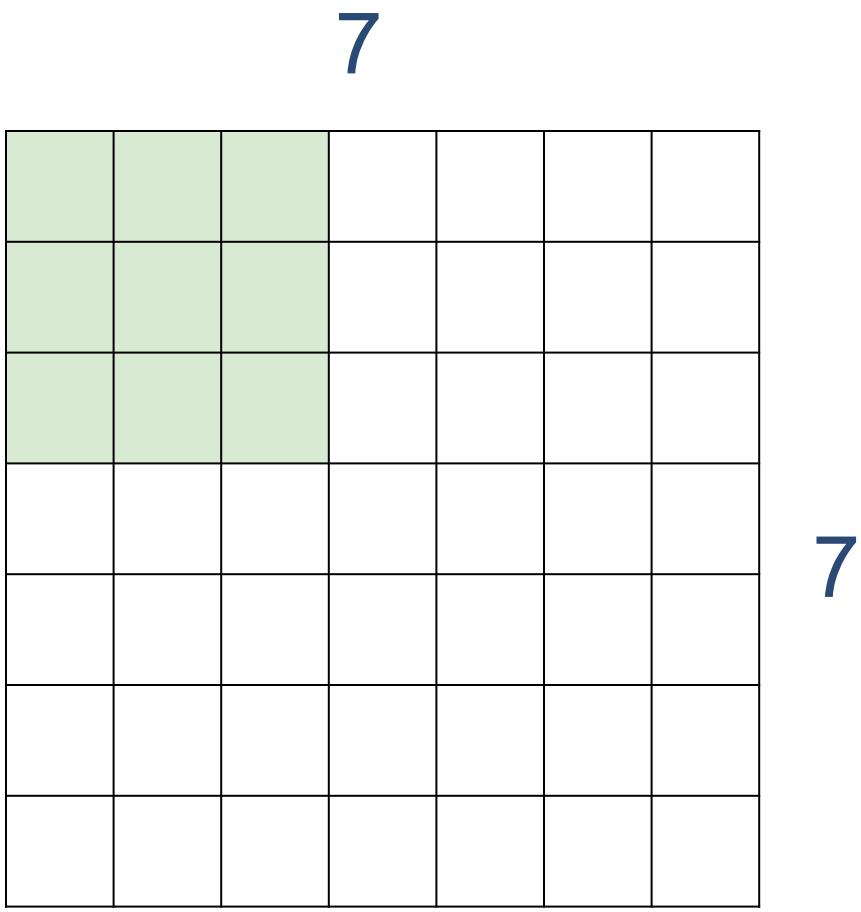
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output!

A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

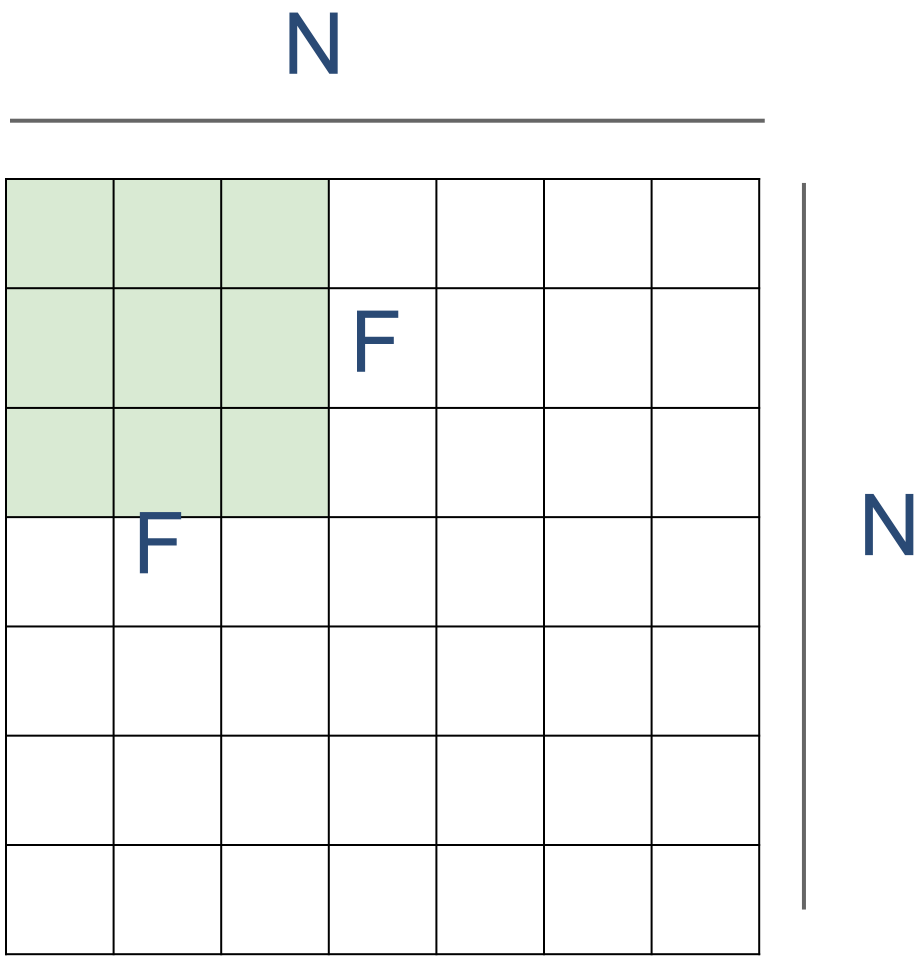
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

doesn't fit!
cannot apply 3x3 filter on
7x7 input with stride 3.

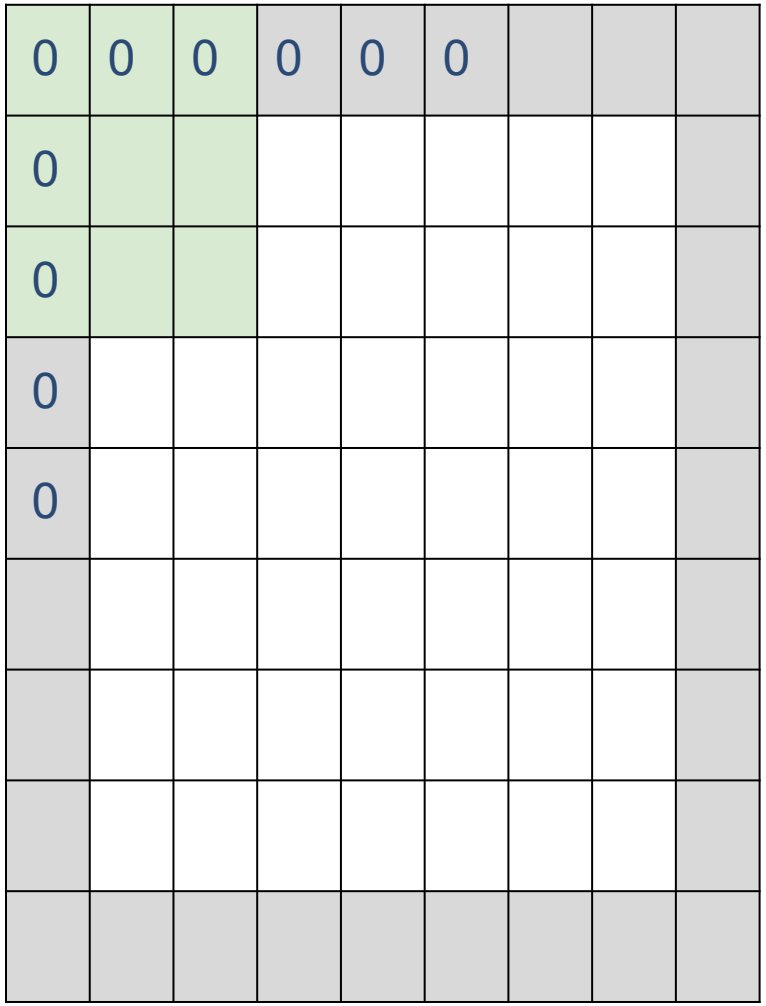
A closer look at spatial dimensions:



Output size:
(N - F) / stride + 1

e.g. N = 7, F = 3:
stride 1 => (7 - 3)/1 + 1 = 5
stride 2 => (7 - 3)/2 + 1 = 3
stride 3 => (7 - 3)/3 + 1 = 2.33 :\

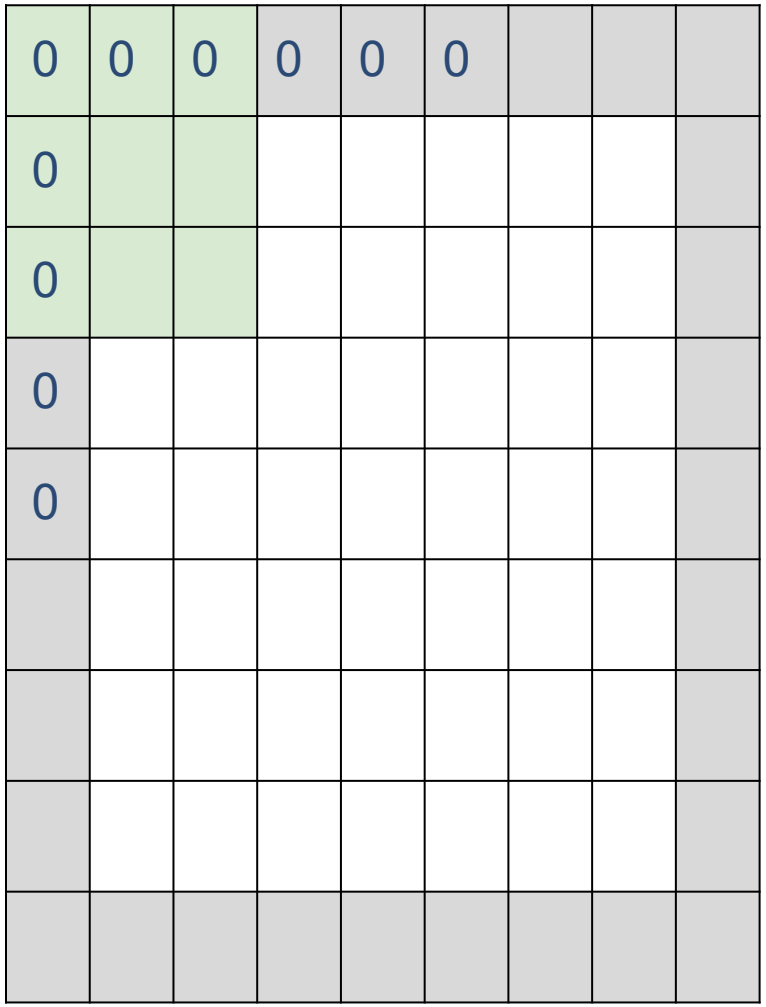
In practice: Common to zero pad the border



e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

(recall:)
(N - F) / stride + 1

In practice: Common to zero pad the border

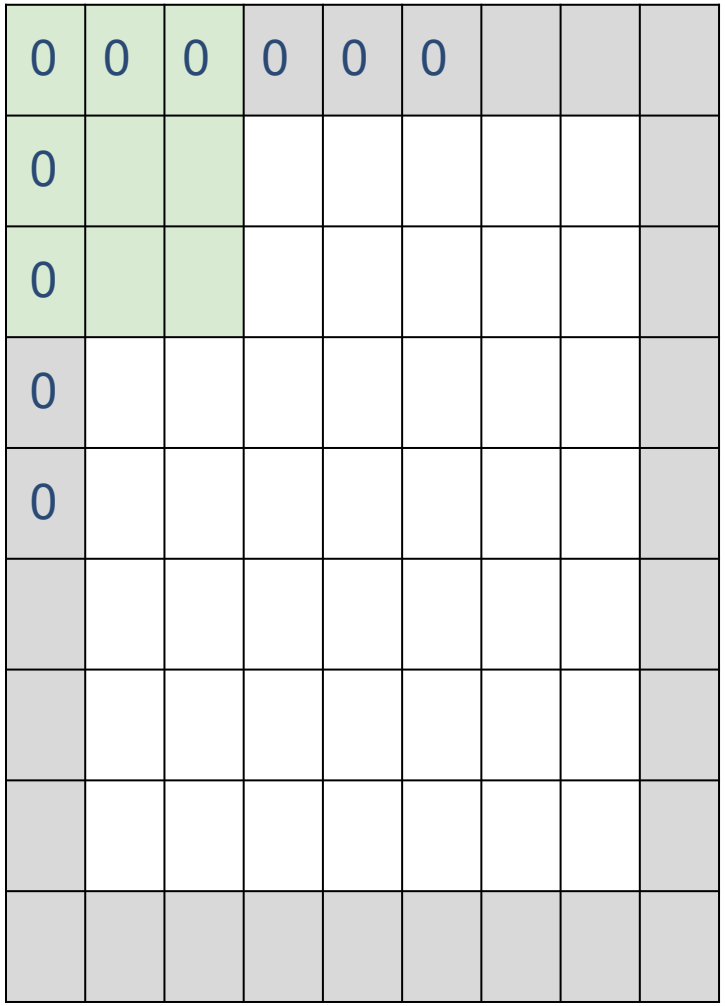


e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!

(recall:)
(N + 2P - F) / stride + 1

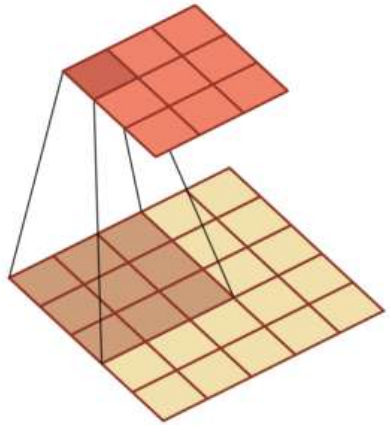
In practice: Common to zero pad the border



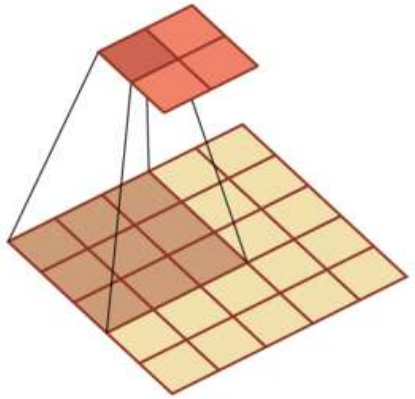
e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!
In general, common to see CONV layers with a stride of 1, filters of size FxF, and zero-padding with (F-1)/2. (will preserve size spatially)
e.g. F = 3 => zero pad with 1
F = 5 => zero pad with 2
F = 7 => zero pad with 3

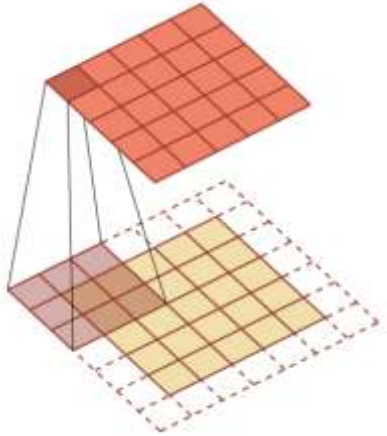
Two-dimensional convolution



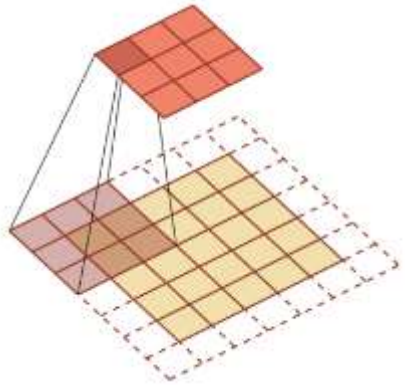
步长1，零填充0



步长2，零填充0



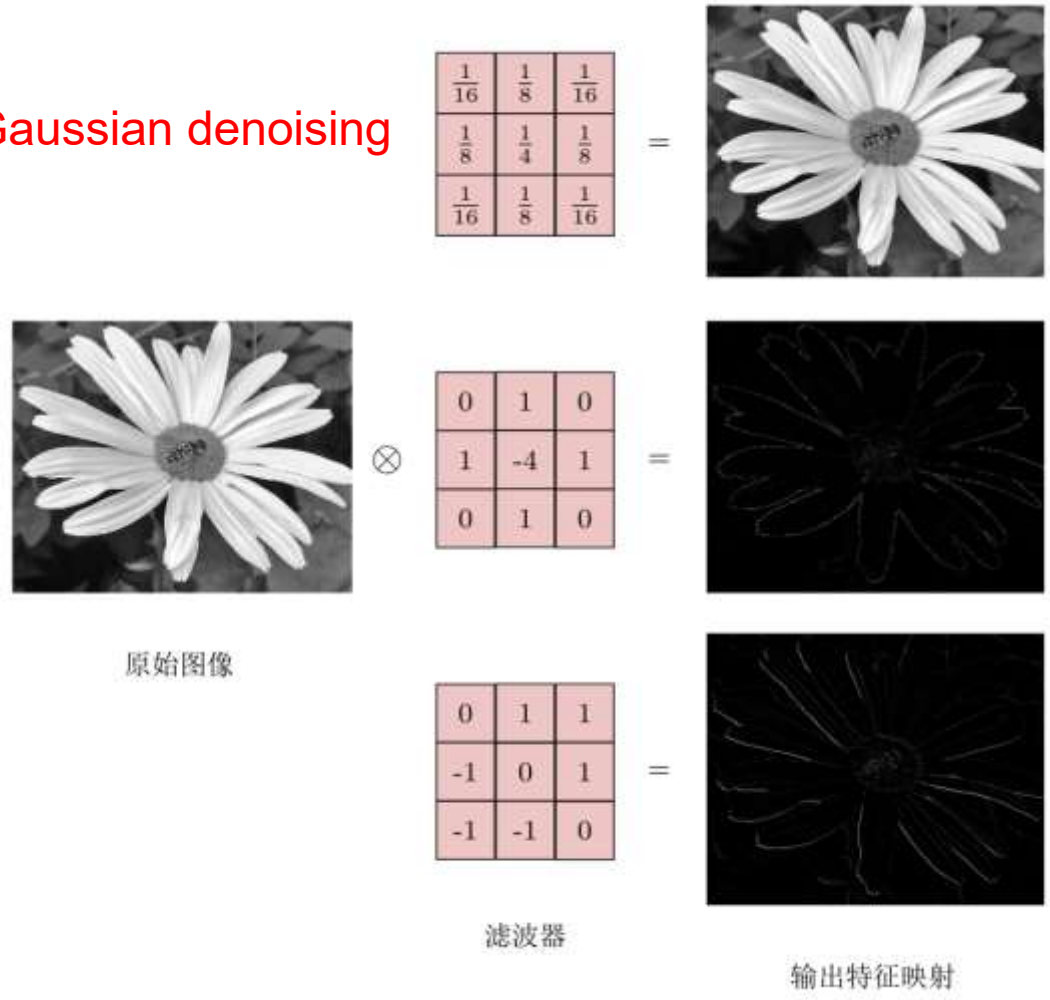
步长1，零填充1



步长2，零填充1

Convolution as Feature Extractor

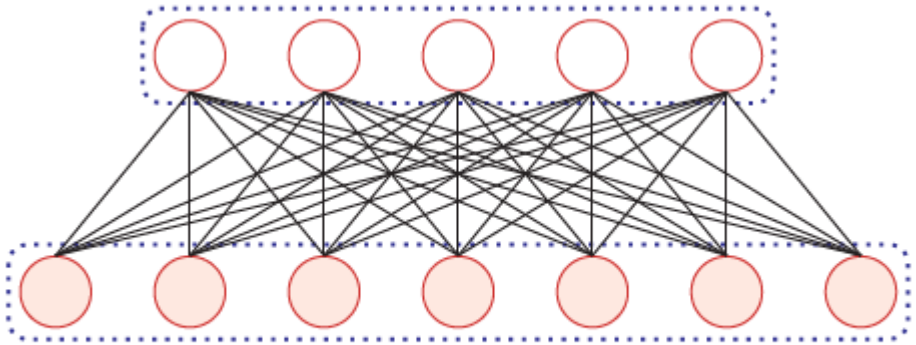
Gaussian denoising



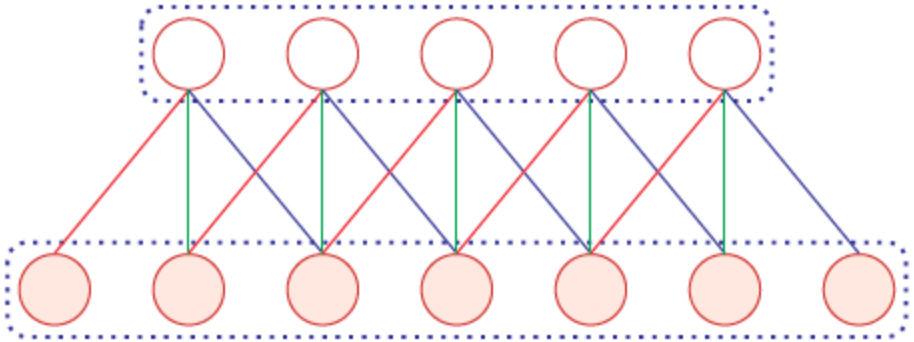
5.1.2 Convolutional Neural Networks

Convolutional Neural Networks (CNN)

- Convolution replace Fully connected: $z^{(l)} = w^{(l)} \otimes a^{(l-1)} + b^{(l)}$,
- Local connection:
  - Connected number $M_l \times M_{l-1}$ (FC) to $M_l \times K$.
- Weight sharing:
  - Convolution parameters has K-d weight $w^{(l)}$ and 1-d bias $b^{(l)}$
  - Totally K+1 parameters



(a) 全连接层



(b) 卷积层

Cross-correlation

- Computing conv requires kernel rotation
- The goal of conv operation is to extract features

Rotation is unnecessary!

- Cross-Correlation (互相关)

$$y_{ij} = \sum_{u=1}^{m}\sum_{v=1}^{n} w_{uv} \cdot x_{i+u-1,j+v-1}$$

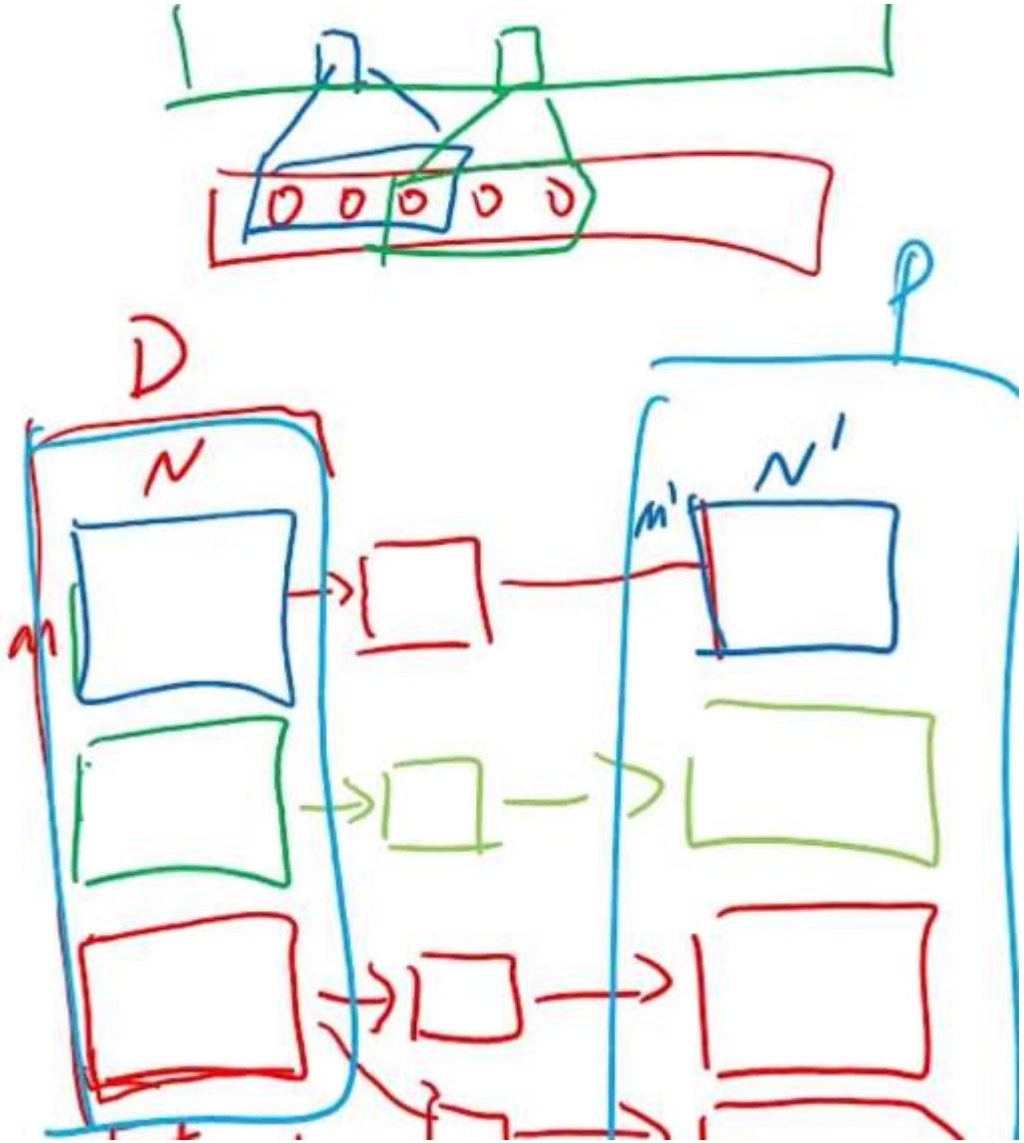
$$\begin{aligned} \boldsymbol{Y} &= \boldsymbol{W} \otimes \boldsymbol{X} \\ &= \mathrm{rot180}(\boldsymbol{W}) * \boldsymbol{X}, \end{aligned}$$

Unless otherwise stated, convolution generally refers to "cross-correlation".

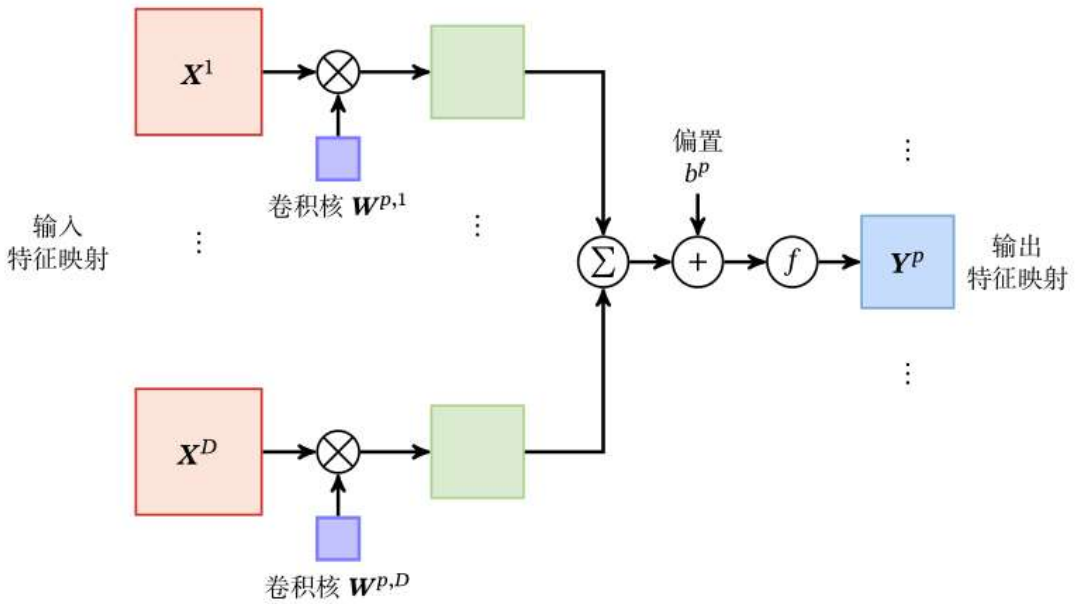
Multiple Kernel (卷积核)

- 卷积核看成一个特征提取器
  - 如何增强卷积层的能力？
    - 引入多个卷积核
- 以两维为例：
  - 特征映射（Feature Map）：图像经过卷积后得到的特征。
  - 卷积层
    - 输入：D个特征映射 $M \times N \times D$
    - 输出：P个特征映射 $M' \times N' \times P$

depth



Mapping Relation of Conv Layer

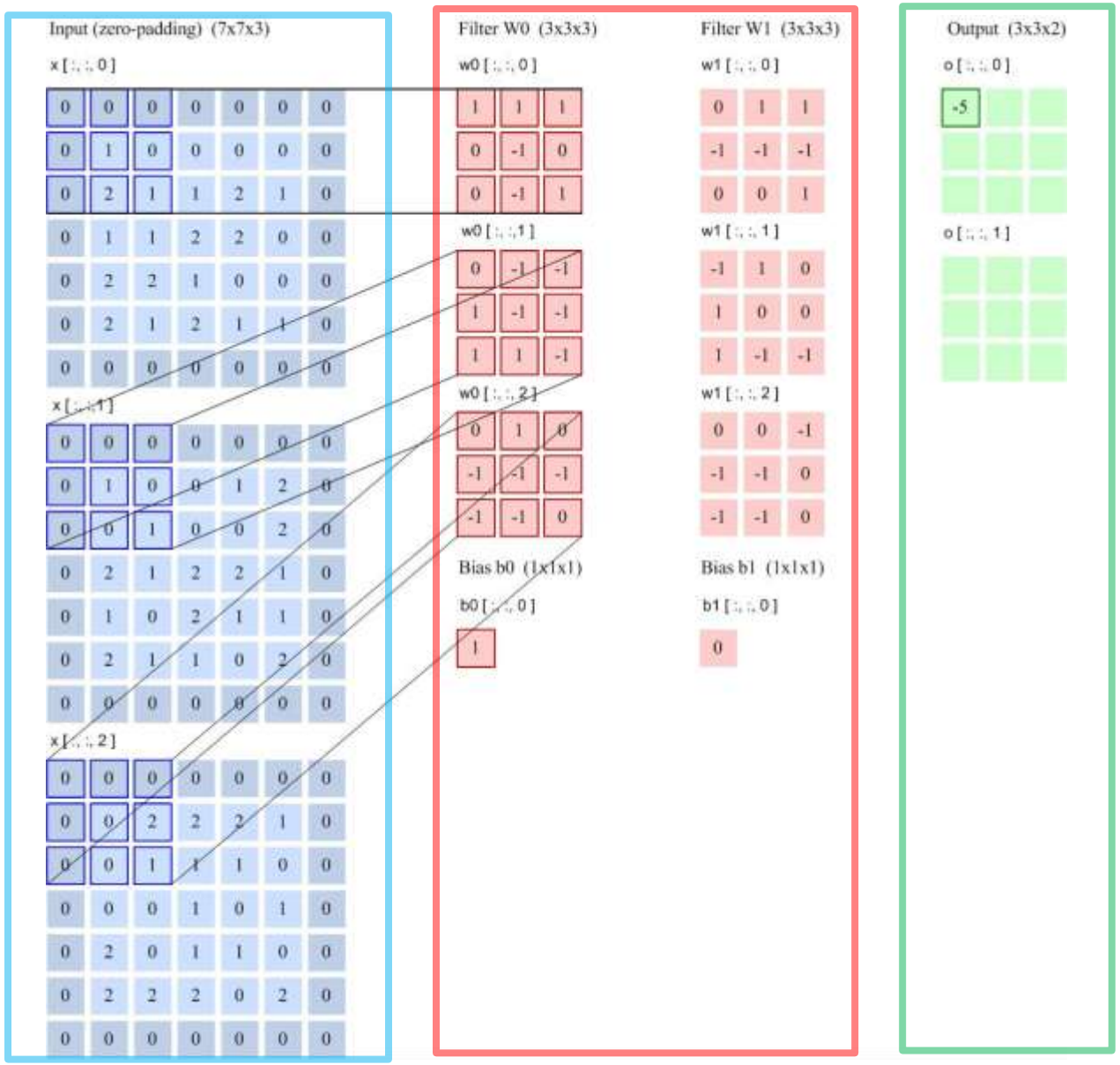


$$Z^p = W^p \otimes X + b^p = \sum_{d=1}^{D} W^{p,d} \otimes X^d + b^p,$$

$$Y^p = f(Z^p).$$

$W^p \in \mathbb{R}^{U \times V \times D}$

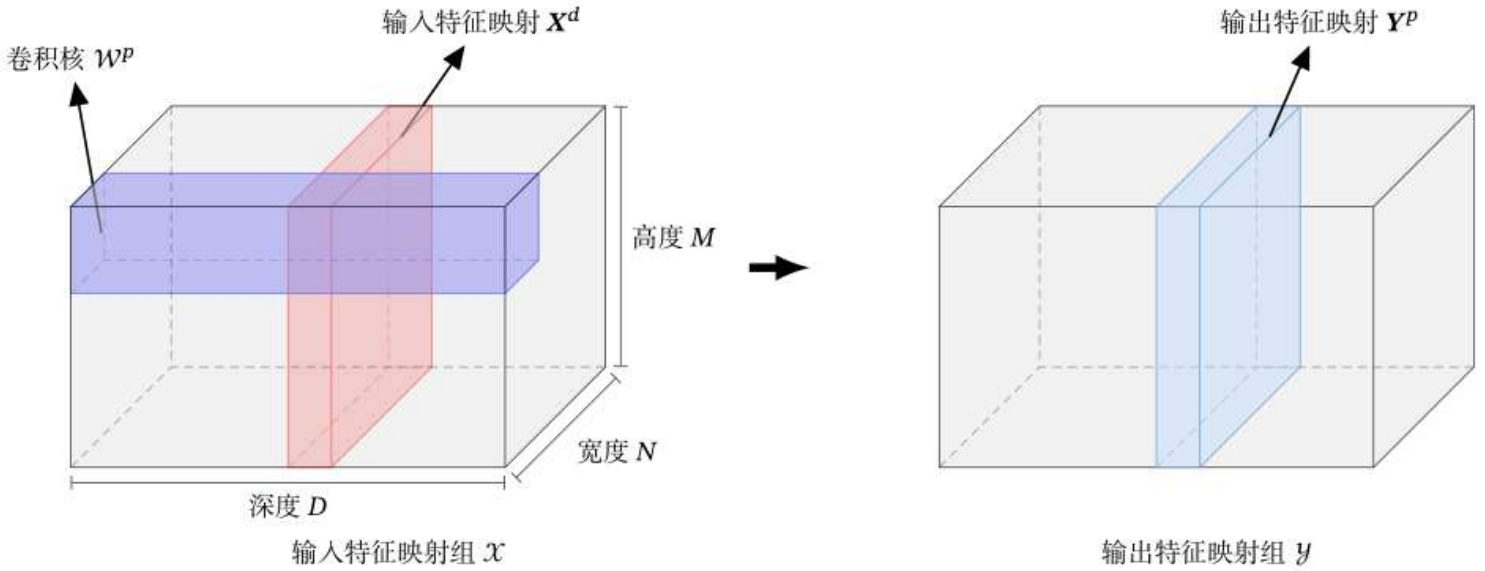
共需要P个这样的三维卷积核



步长2
filter 3*3
filter个数6
零填充 1

Convolution Layer

- Typical conv layer is a 3-d structure



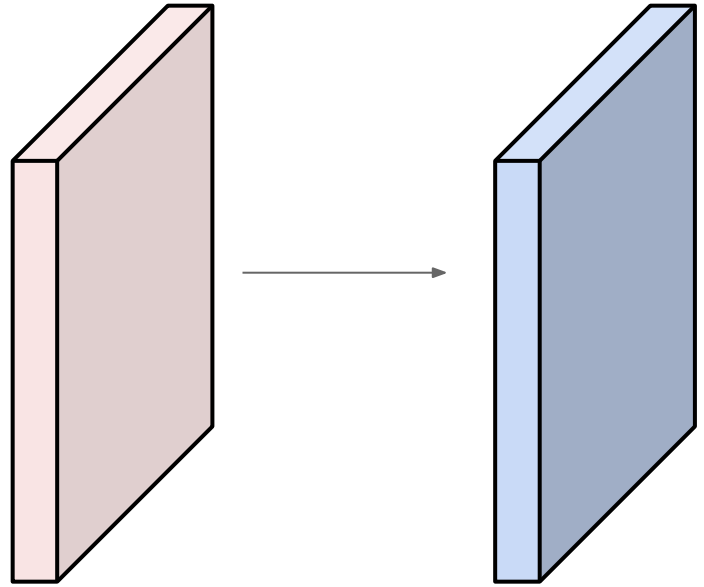
$$Z^p = W^p \otimes X + b^p = \sum_{d=1}^{D} W^{p,d} \otimes X^d + b^p,$$

$$Y^p = f(Z^p).$$

Required Parameters: $\big((U \times V) \times D + 1\big) \times P$

Examples time:

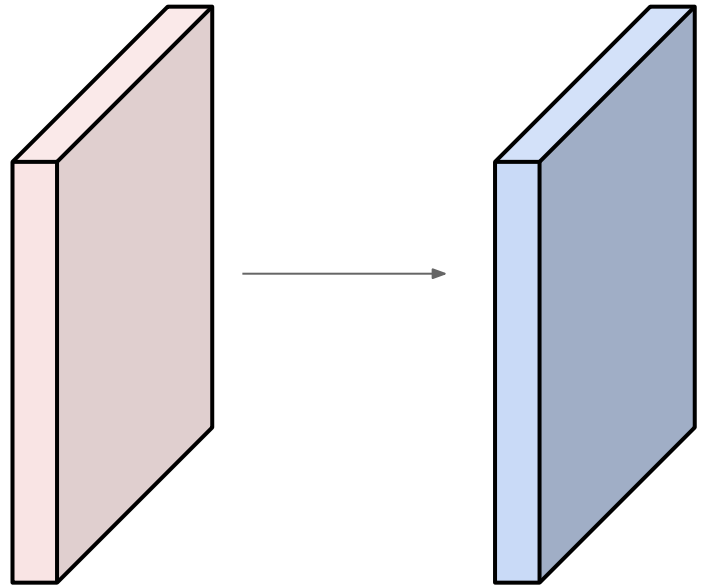
Input volume: **32x32x3**
10 5x5**x3** filters with stride 1, pad 2



Output volume size: ?
Number of parameters in this layer?

Examples time:

Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2



Output volume size:
(32+2*2-5)/1+1 = 32 spatially, so
32x32x10

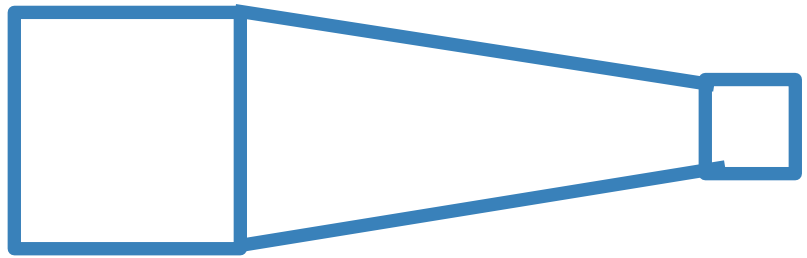
Examples time:

Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?
each filter has 5*5*3 + 1 = 76 params (+1 for bias)
=> 76*10 = **760**

Receptive Fields

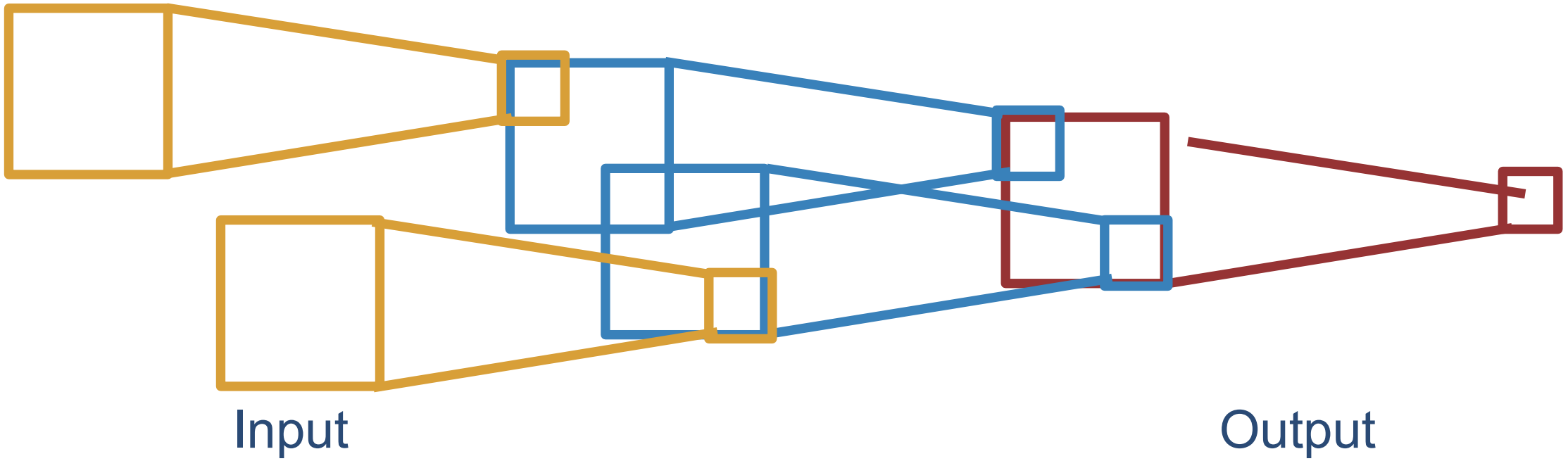
For convolution with kernel size K, each element in the output depends on a K x K **receptive field** in the input.



Input Output

Receptive Fields

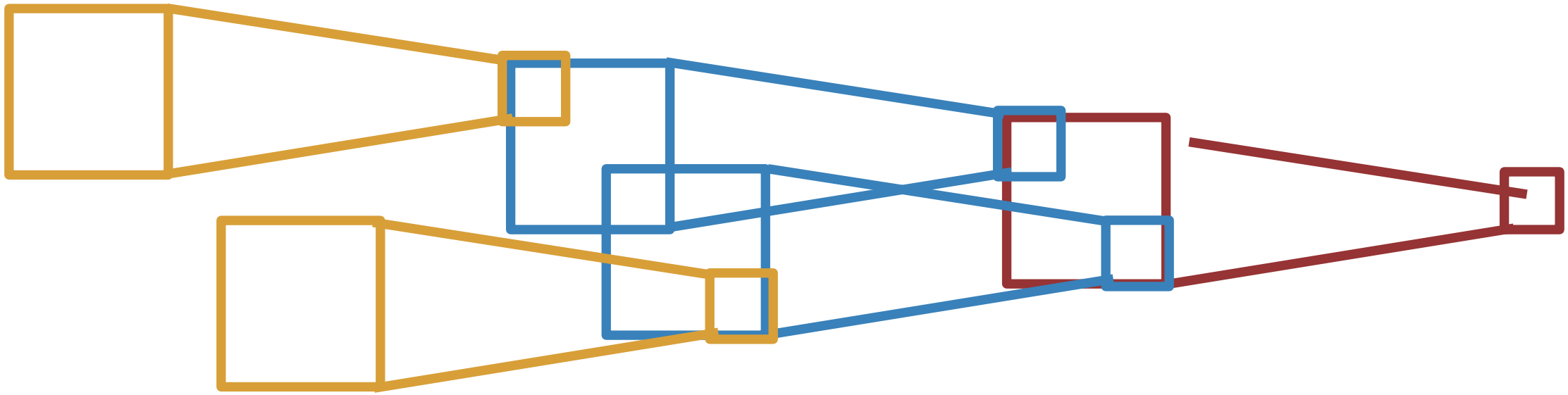
Each successive convolution adds K – 1 to the receptive field size
With L layers the receptive field size is 1 + L * (K – 1)



Be careful – “receptive field in the input” vs. “receptive field in the previous layer”

Receptive Fields

Each successive convolution adds K – 1 to the receptive field size
With L layers the receptive field size is 1 + L * (K – 1)



Input

Output

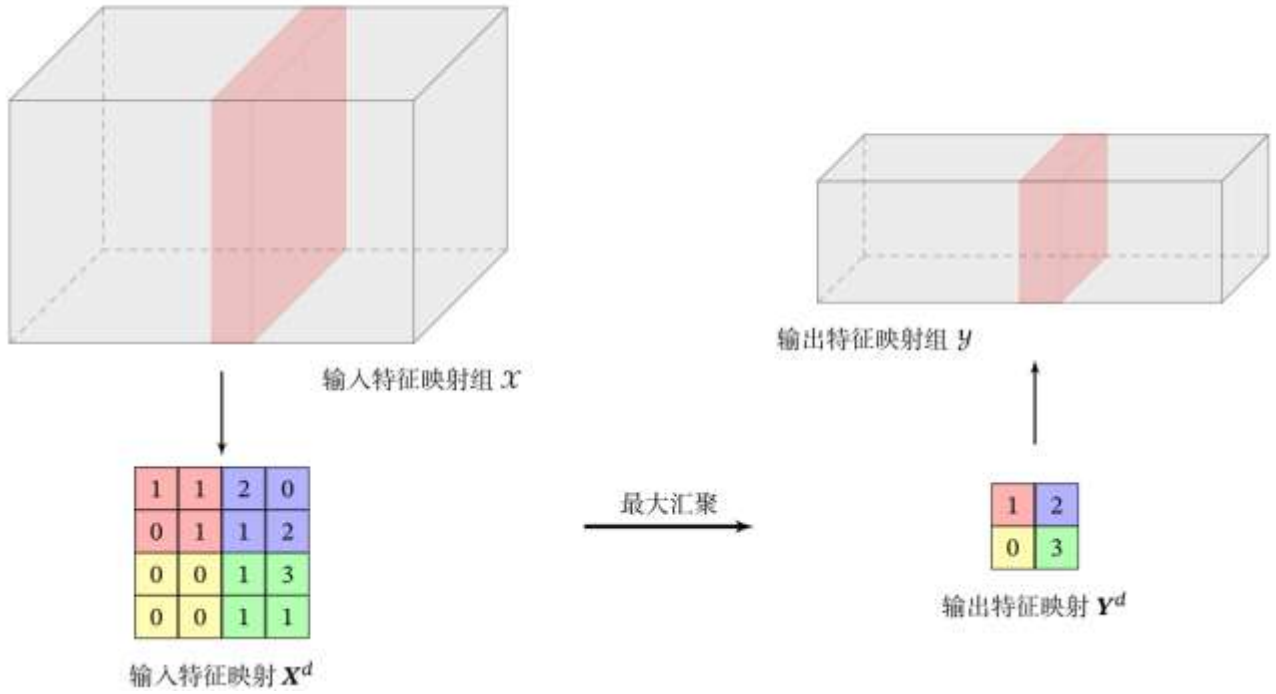
Problem: For large images we need many layers for each output to “see” the whole image image

Solution: Downsample inside the network

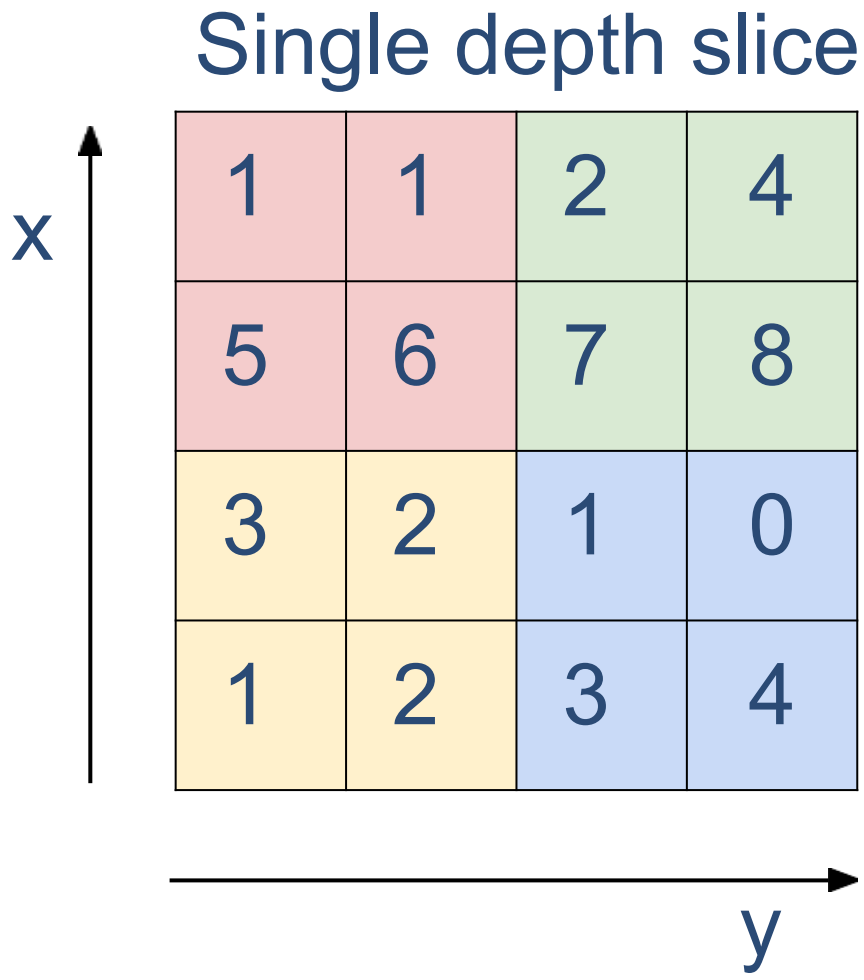
Slide inspiration: Justin Johnson

Pooling Layer (汇聚层)

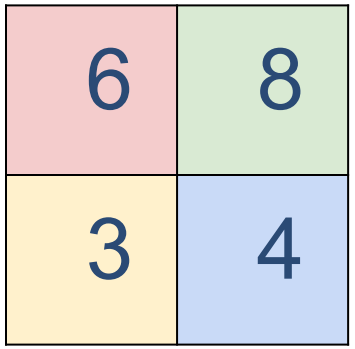
- Although conv layer can reduce the number of connections, the number of neurons in each feature map has not been reduced.



Max Pooling



max pool with 2x2 filters
and stride 2



- No learnable parameters
- Introduces spatial invariance

Pooling layer: summary

Let's assume input is W_1 x H_1 x C Conv layer needs 2 hyperparameters:

- The spatial extent **F**
- The stride **S**

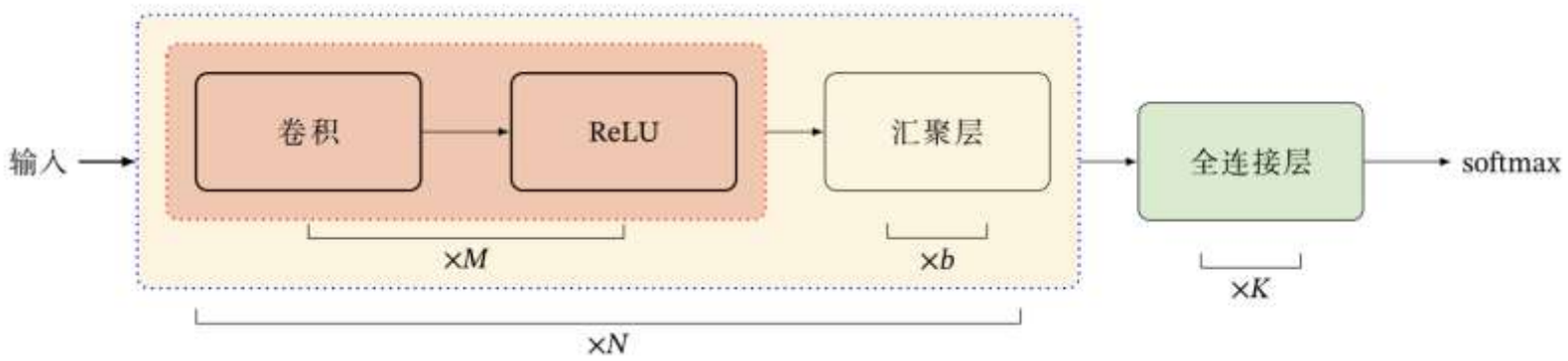
This will produce an output of W_2 x H_2 x C where:

- $W_2 = (W_1 - F)/S + 1$
- $H_2 = (H_1 - F)/S + 1$

Number of parameters: 0

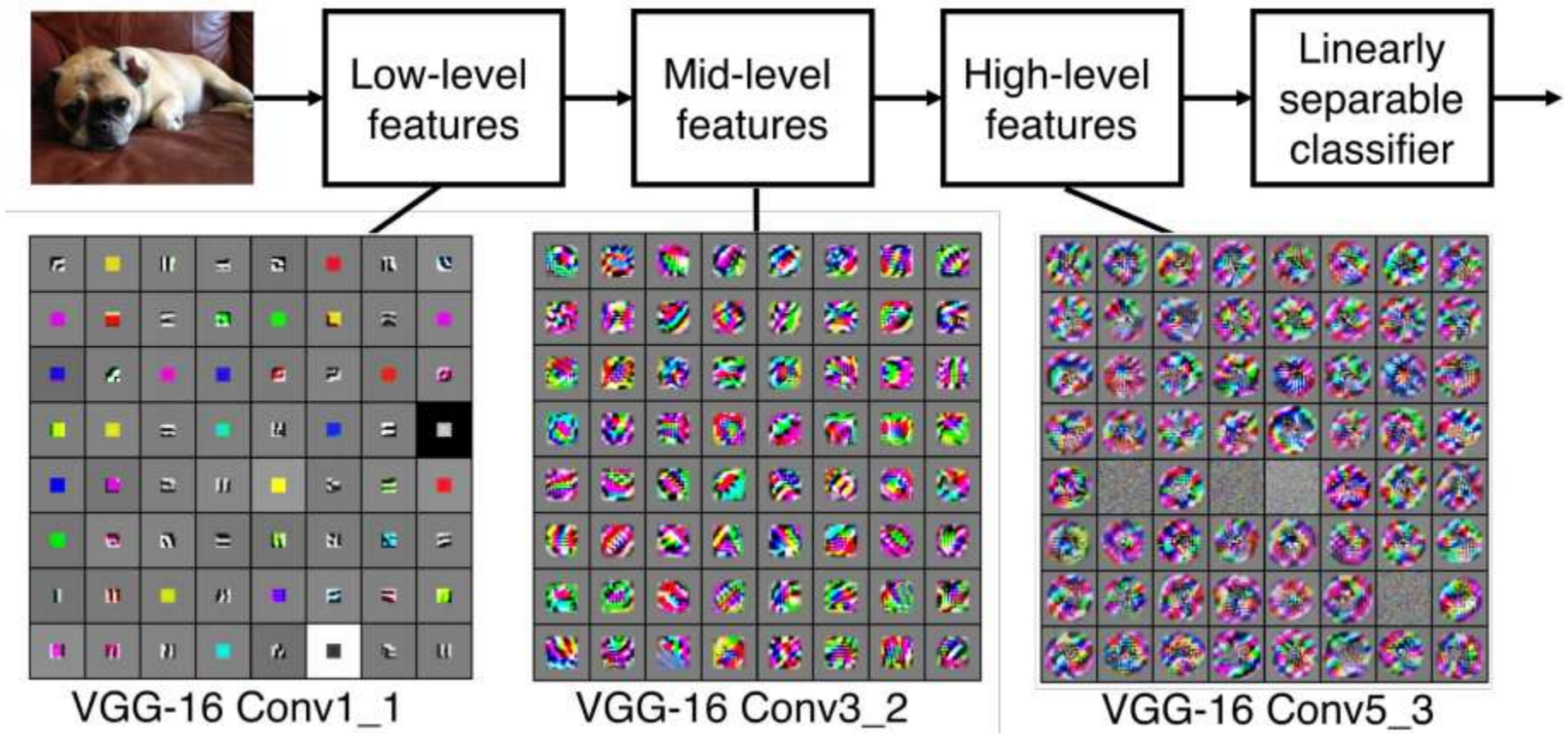
Structure of Conv Networks

- Conv networks: stacked by conv layer, pooling layer, and FC layer (交叉堆叠)
  - Tend to small convolution and large depth
  - Tend to total convolution (全卷积)
- Typical structure

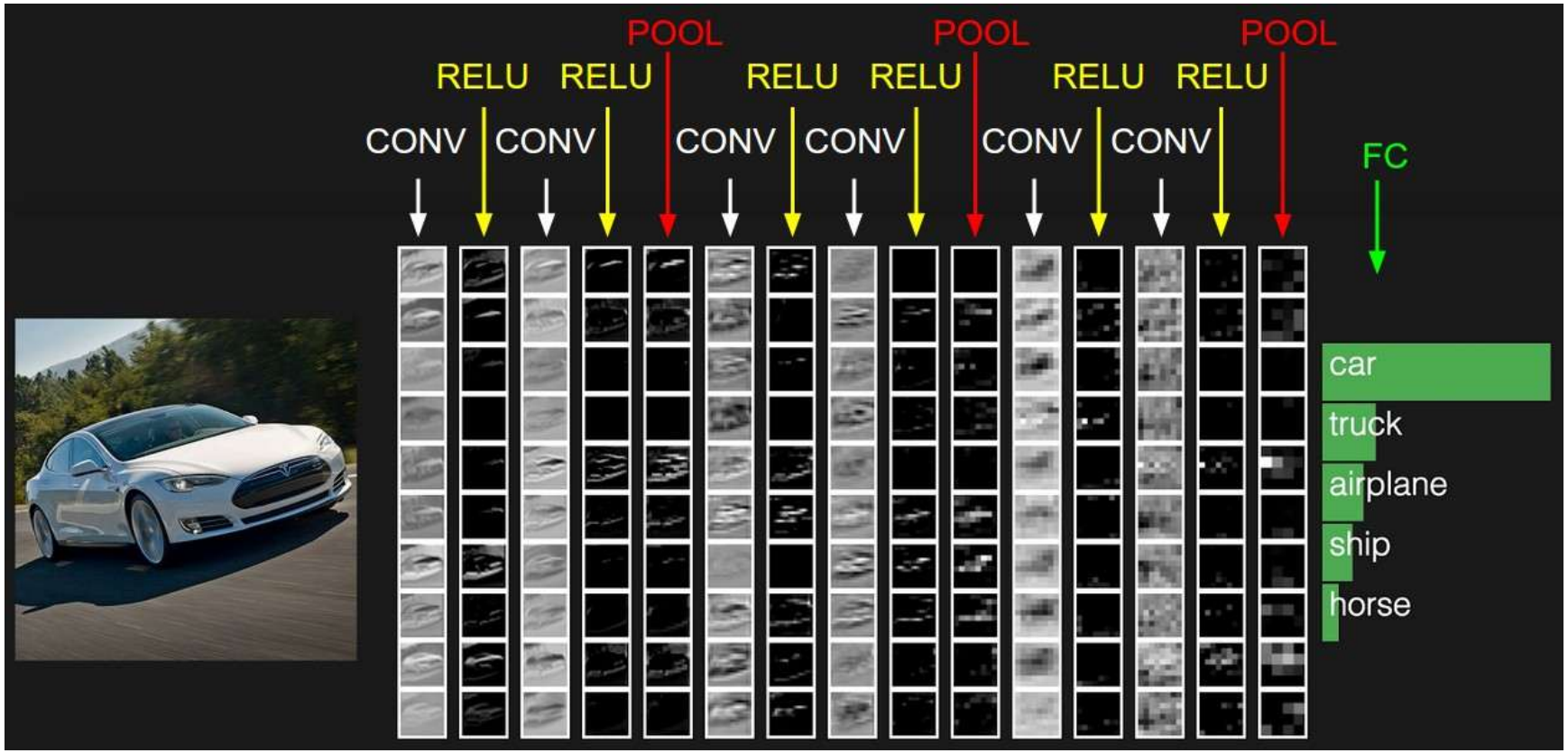


- 一个卷积块为连续M个卷积层和b个汇聚层(M通常设置为2～5，b为0或1)。一个卷积网络可以堆叠N 个连续的卷积块，然后在接着K个全连接层(N 的取值区间比较大，比如1～100或者更大；K一般为0～2)。

Representation Learning



Representation Learning



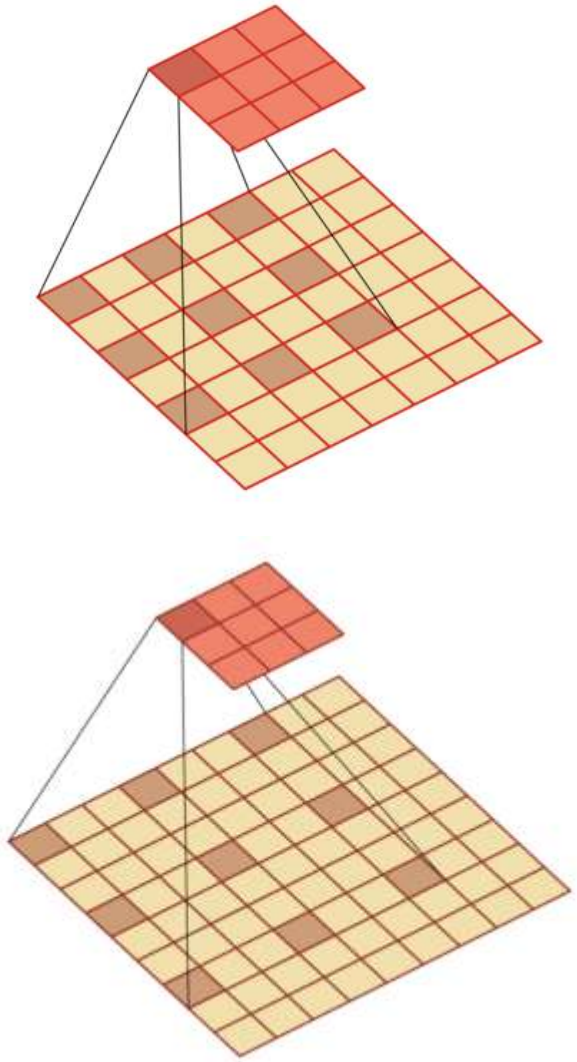
Thinking Questions

- Is CNN's Locality Hypothesis Reasonable?
  - How to improved?
- Extra Reading:
  - Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., ... & Houlsby, N. (2020). An image is worth 16x16 words: Transformers for image recognition at scale. arXiv preprint arXiv:2010.11929.
  - Tolstikhin, I. O., Houlsby, N., Kolesnikov, A., Beyer, L., Zhai, X., Unterthiner, T., ... & Dosovitskiy, A. (2021). Mlp-mixer: An all-mlp architecture for vision. Advances in neural information processing systems, 34, 24261-24272.

5.1.3 Other convolution types

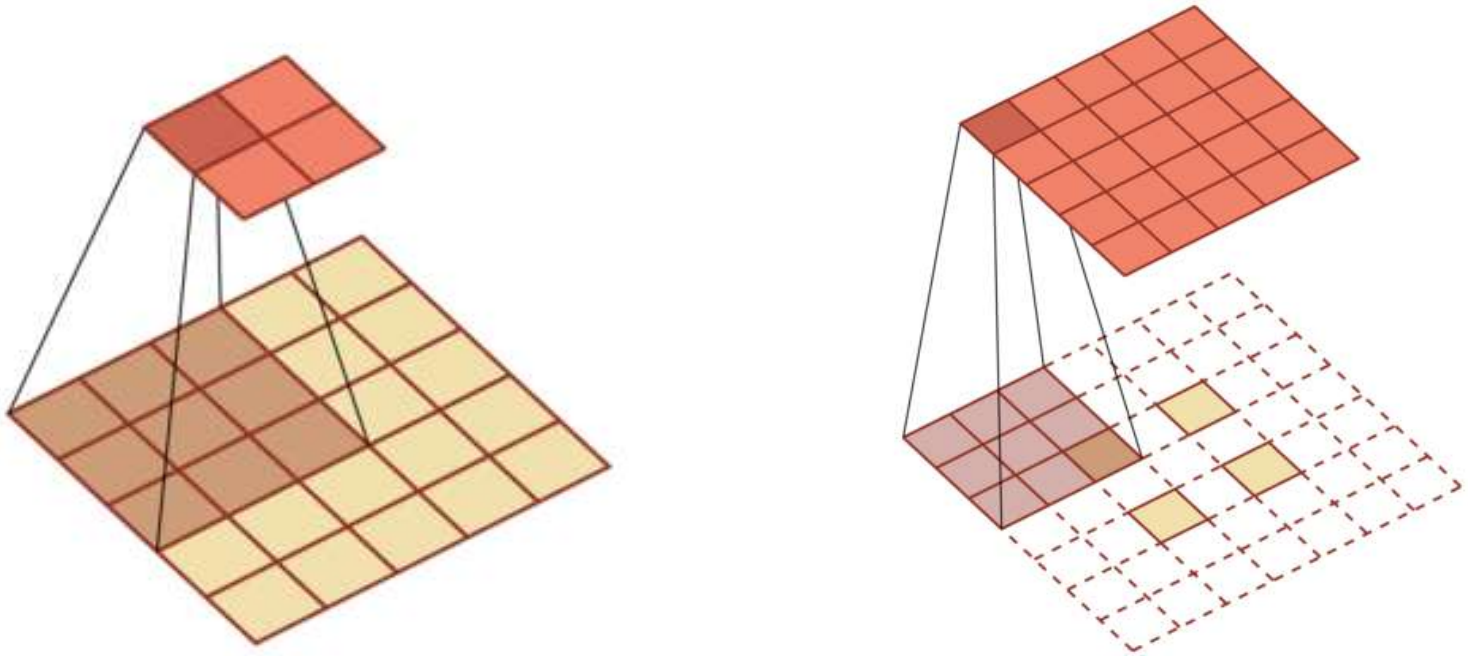
Atrous (空洞) / Dilated (膨胀) Convolution

- How to increase the receptive field of output unit?
  - Increase kernel size
  - Increase the number of layers
  - Pooling before conv
- Atrous Convolution
  - By inserting a "hole" into the convolution kernel, its size is increased in a disguised way.



Transposed Conv/Fractional-Stride Conv

- 转置卷积/微步卷积
- Mapping low-dimensional features to high-dimensional features

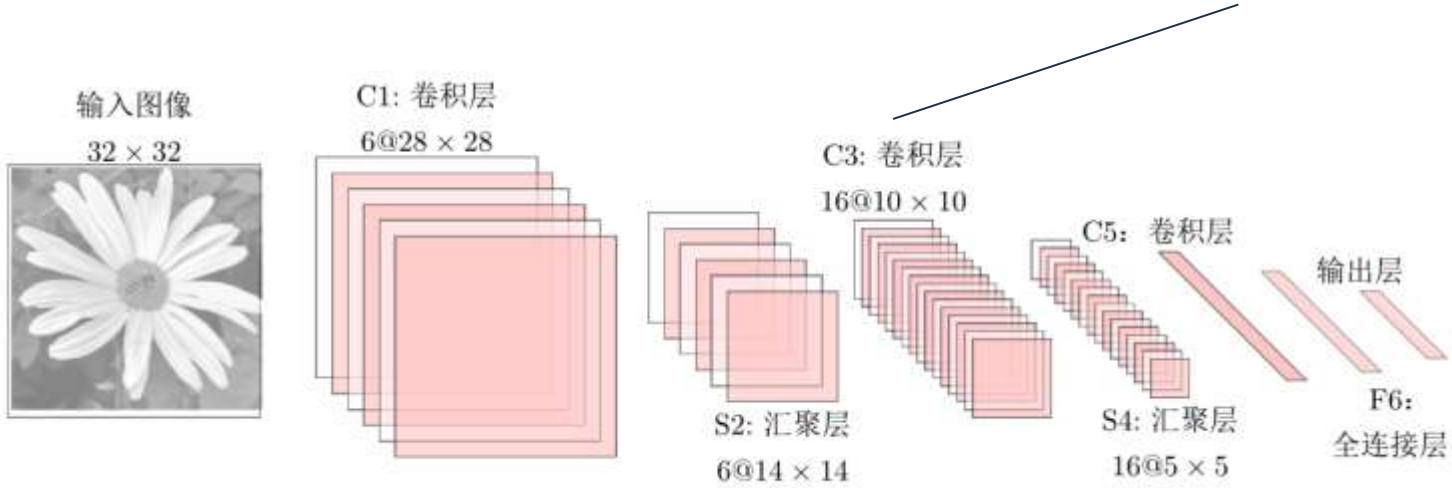


5.1.4 Typical convolutional network

LeNet-5

- LeNet-5 is a a very successful neural network model.
  - The handwritten recognition system based on LeNet-5 was used by many banks in US in the 1990s to recognize handwritten numerals on cheques.
  - LeNet-5 has 7 layers

How many kernels are needed?



Large Scale Visual Recognition Challenge

2012 Teams	%error
Supervision (Toronto)	15.3
ISI (Tokyo)	26.1
VGG (Oxford)	26.9
XRCE/INRIA	27.0
UvA (Amsterdam)	29.6
INRIA/LEAR	33.4

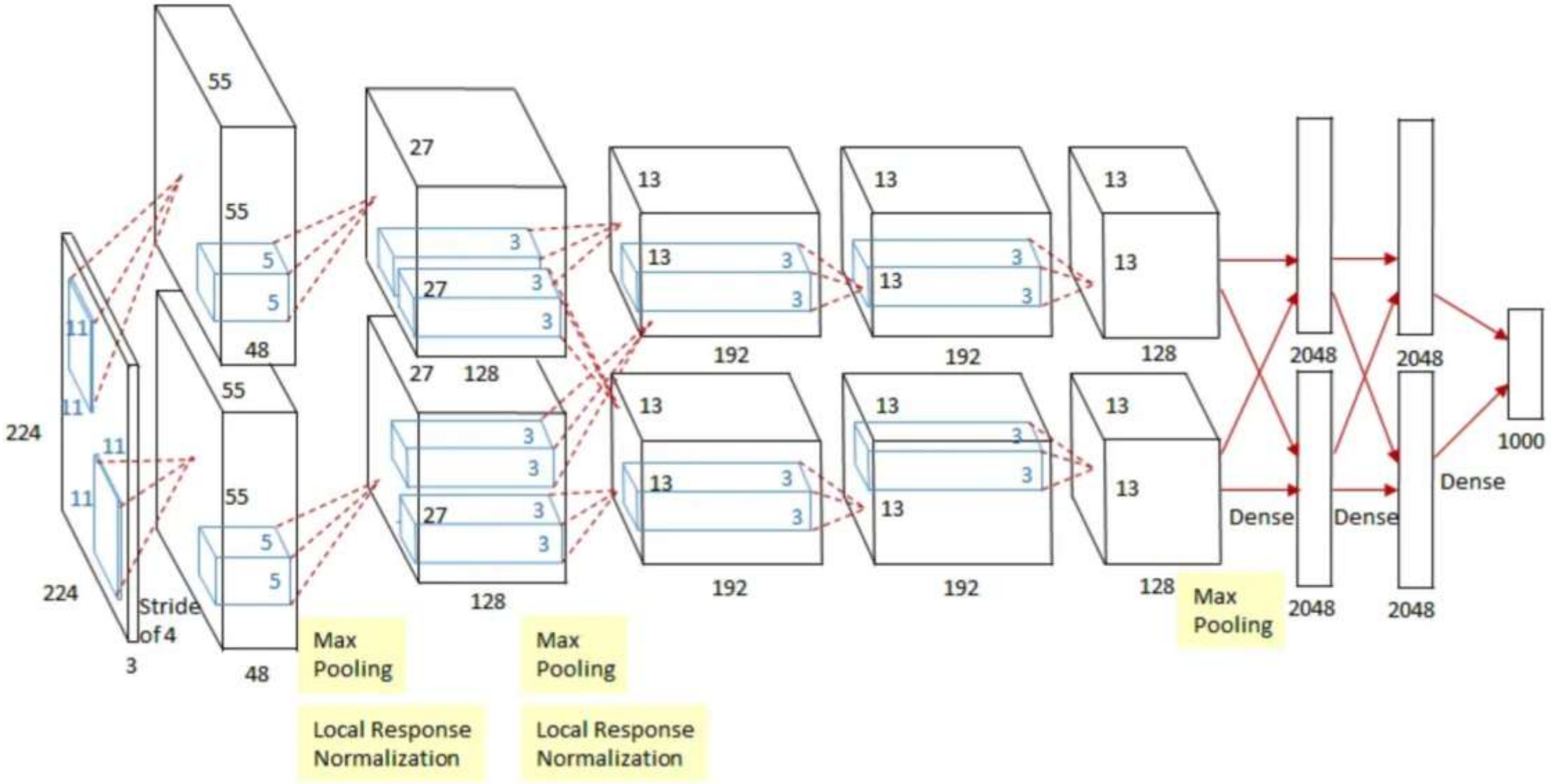
2013 Teams	%error
Clarifai (NYU spinoff)	11.7
NUS (singapore)	12.9
Zeiler-Fergus (NYU)	13.5
A. Howard	13.5
OverFeat (NYU)	14.1
UvA (Amsterdam)	14.2
Adobe	15.2
VGG (Oxford)	15.2
VGG (Oxford)	23.0

2014 Teams	%error
GoogLeNet	6.6
VGG (Oxford)	7.3
MSRA	8.0
A. Howard	8.1
DeeperVision	9.5
NUS-BST	9.7
TTIC-ECP	10.2
XYZ	11.2
UvA	12.1

AlexNet

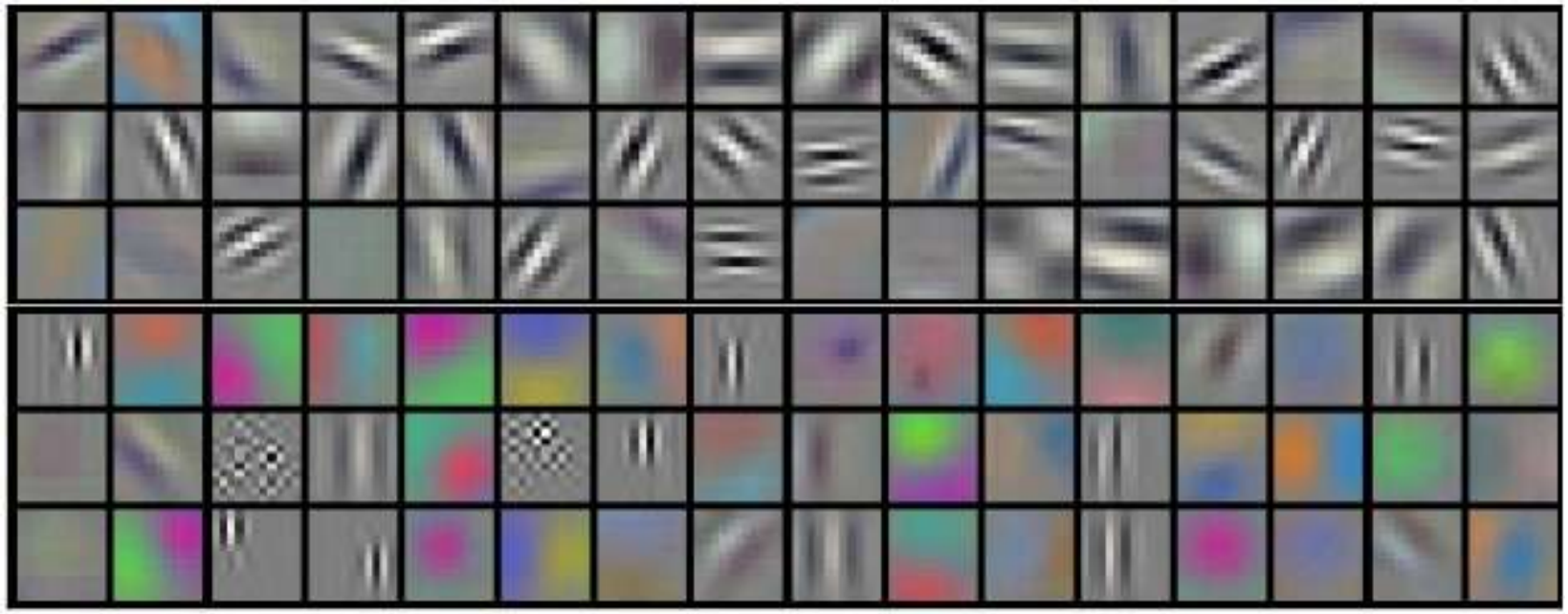
- 2012 ILSVRC winner
  - (top 5 error of 16% compared to runner-up with 26% error)
  - The first modern deep CNN model
  - GPU is used for parallel training, ReLU is used as nonlinear activation function, Dropout is used to prevent over-fitting, and data enhancement is used.
  - 5 conv layers, 3 pooling layers, and 3 FC layers

AlexNet


55
55
48
55
55
48
224
224
11
11
11
11
5
5
5
5
3
Stride of 4
27
27
27
128
128
3
3
3
3
3
3
13
13
13
13
13
13
13
13
13
192
192
192
192
128
128
2048
2048
2048
2048
1000
Dense
Dense
Dense
Max Pooling
Max Pooling
Local Response Normalization
Local Response Normalization

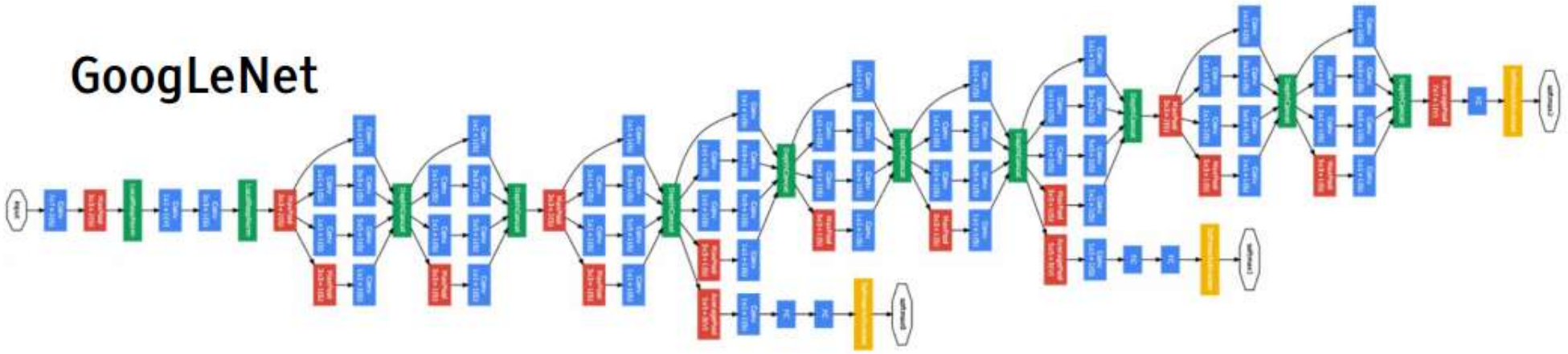
CNN Visualization: Filters

- Filers in AlexNet（96 filters [11x11x3]）



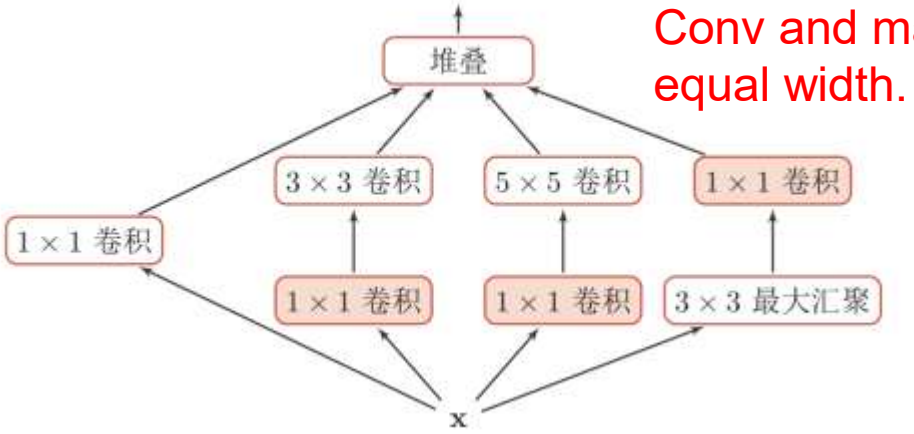
Inception Network

- 2014 ILSVRC winner (22 layers)
  - Parameter： GoogLeNet： 4M VS AlexNet： 60M
  - Error rate： 6.7%
  - Inception is stacked by many inception modules and a few pooling layers.



Inception Module v1

- In CNN, how to set the kernel size is a key problem
  - In Inception, a conv layer contains several conv operation with different sizes, called a Inception Module.
  - Inception module uses kernels with different sizes, such as 1 × 1、3 × 3、5 × 5, and the obtained feature maps are stacked in depth as output feature maps.



Conv and max pooling are of equal width.

Inception Module v3

- Multi-layer small kernels are used to replace the large kernel, to reduce computational overhead and parameters
  - 2 layer 3×3 kernel replaces 5×5 kernel in v
  - Successive $n \times 1$ and $1 \times n$ kernels replaces $n \times n$ kernel

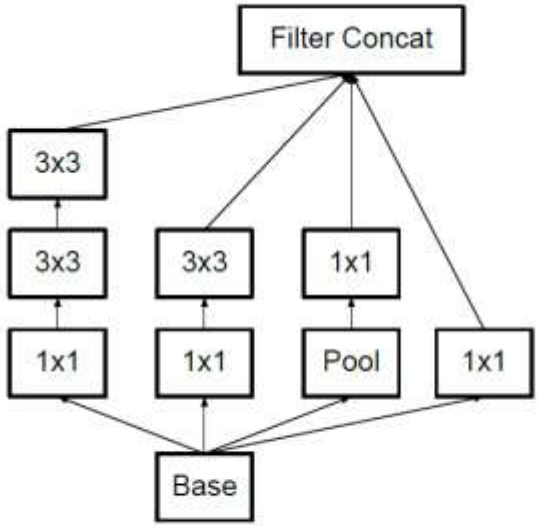


Figure 5. Inception modules where each 5×5 convolution is replaced by two 3×3 convolution, as suggested by principle 3 of Section 2.

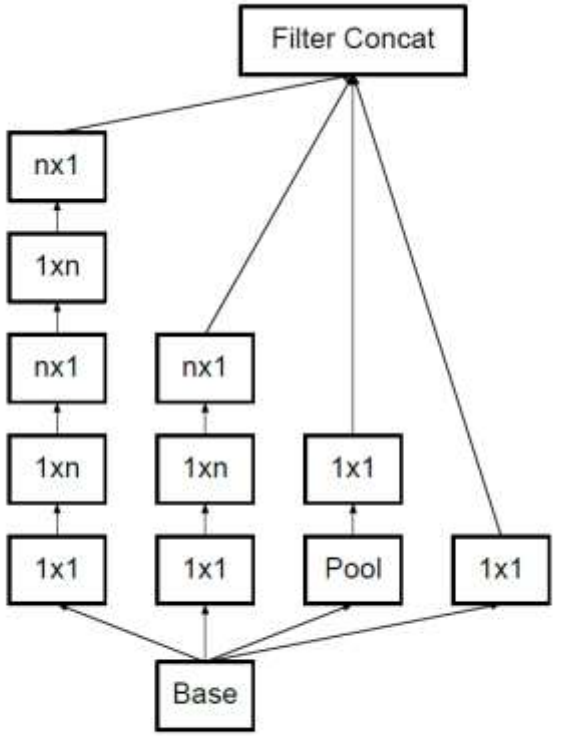
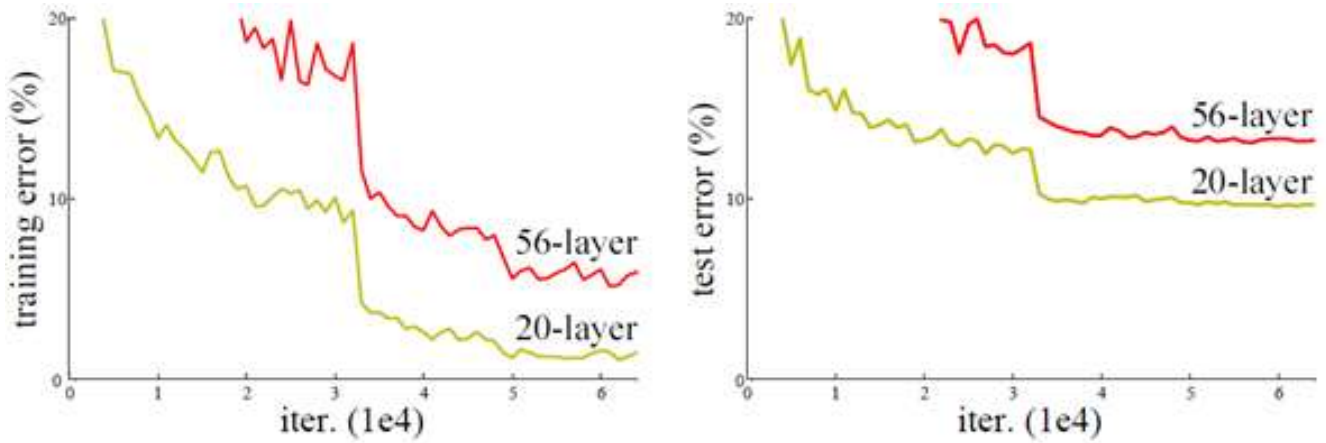


Figure 6. Inception modules after the factorization of the $n \times n$ convolutions. In our proposed architecture, we chose $n = 7$ for the 17×17 grid. (The filter sizes are picked using principle 3)

Residual Network (ResNet)

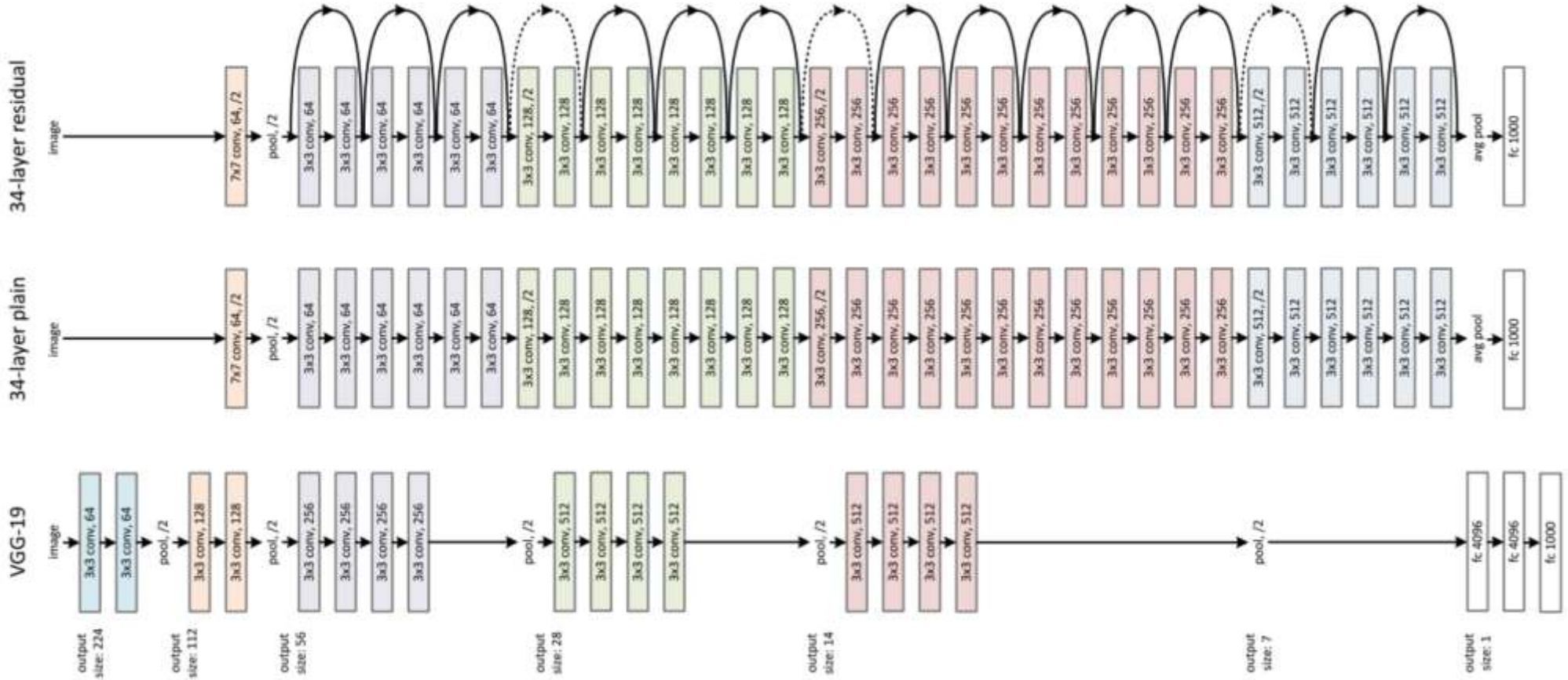
- Degradation problem in deep networks



- With increasing depth, accuracy saturated or even decreased/
  - In CIFAR-10, 56-layer is worse than 20-layer network. This is not an over-fitting. Gradient vanishing or explosion?
  - He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).

Residual Network (ResNet)

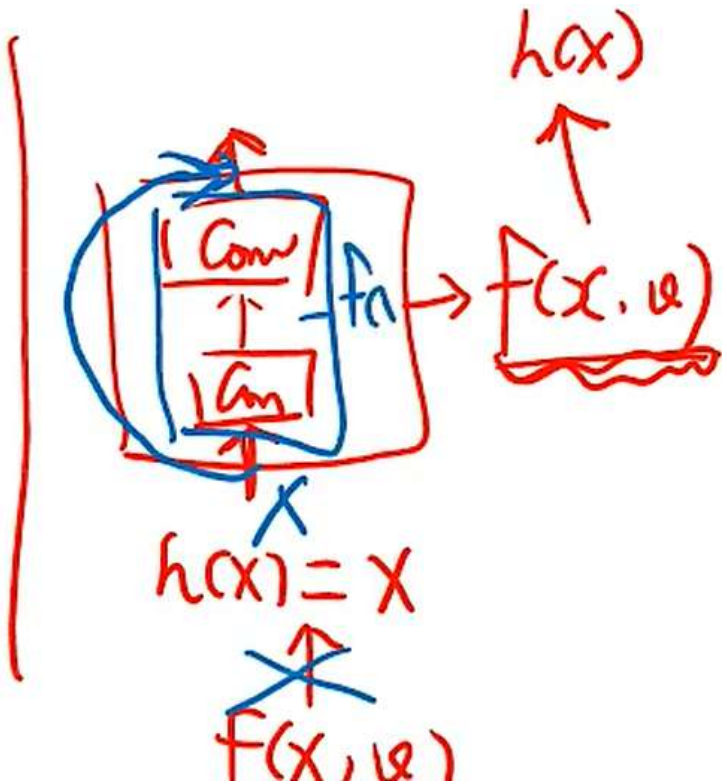
- 2015 ILSVRC winner （152 layers）
  - Error rate： 3.57%



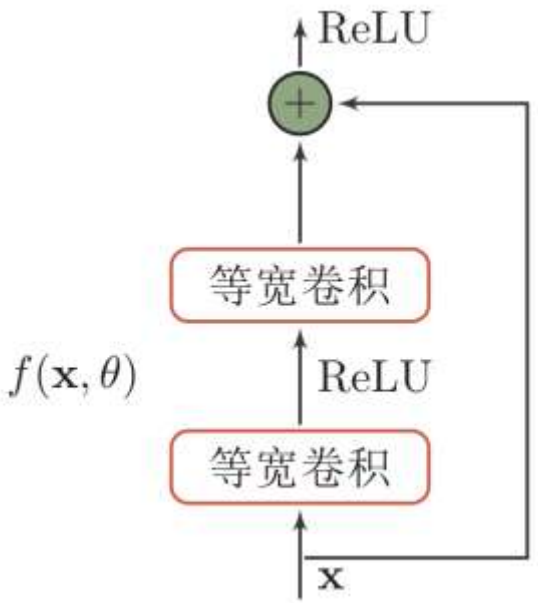
Residual Network (ResNet)

- ResNet adds a shortcut connection to the nonlinear conv layer to improve the efficiency of information propagation.
  - In a deep network, a nonlinear unit (one or more conv layers) f(x,θ) to approximate an objective function h(x).
  - Objective is decomposed to: identity function and residue function.

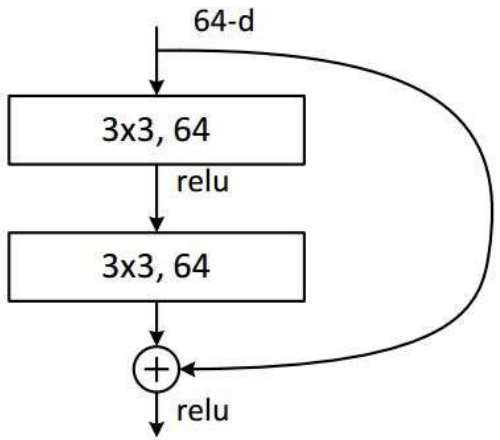
$$h(\mathbf{x}) = \underbrace{\mathbf{x}}_{\text{恒等函数}} + \underbrace{(h(\mathbf{x}) - \mathbf{x})}_{\text{残差函数}} \rightarrow f(\mathbf{x}, \theta)$$



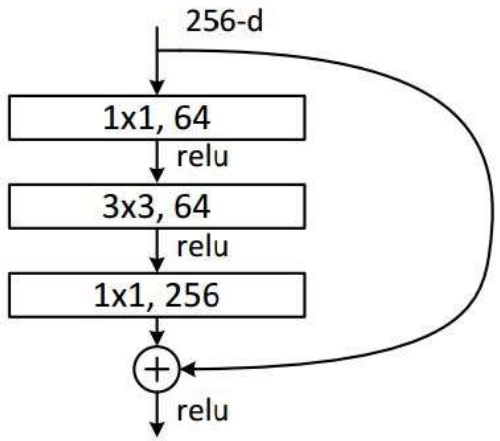
Residual Units



- Directly performing 3x3 convolutions with 256 feature maps at input and output:
  256 x 256 x 3 x 3 ~ 600K operations



- Using 1x1 convolutions to reduce 256 to 64 feature maps, followed by 3x3 convolutions, followed by 1x1 convolutions to expand back to 256 maps:
  256 x 64 x 1 x 1 ~ 16K
  64 x 64 x 3 x 3 ~ 36K
  64 x 256 x 1 x 1 ~ 16K
  Total: ~70K



Why ResNet is good?

为什么残差学习相对更容易，从直观上看残差学习需要学习的内容少，因为残差一般会比较小，学习难度小点。不过我们可以从数学的角度来分析这个问题，首先残差单元可以表示为：

$$y_l = h(x_l) + F(x_l, W_l)$$
$$x_{l+1} = f(y_l)$$

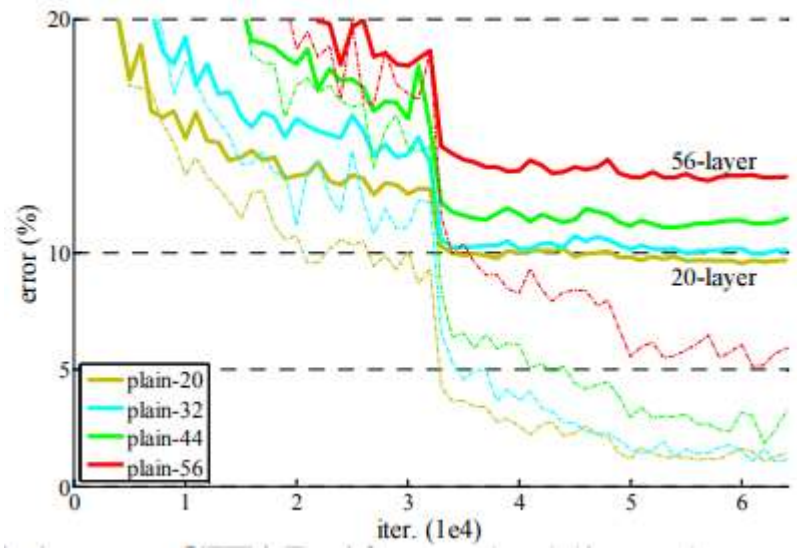
其中 x_l 和 x_{l+1} 分别表示的是第 l 个残差单元的输入和输出，注意每个残差单元一般包含多层结构。 F 是残差函数，表示学习到的残差，而 $h(x_l) = x_l$ 表示恒等映射， f 是ReLU激活函数。基于上式，我们求得从浅层 l 到深层 L 的学习特征为：

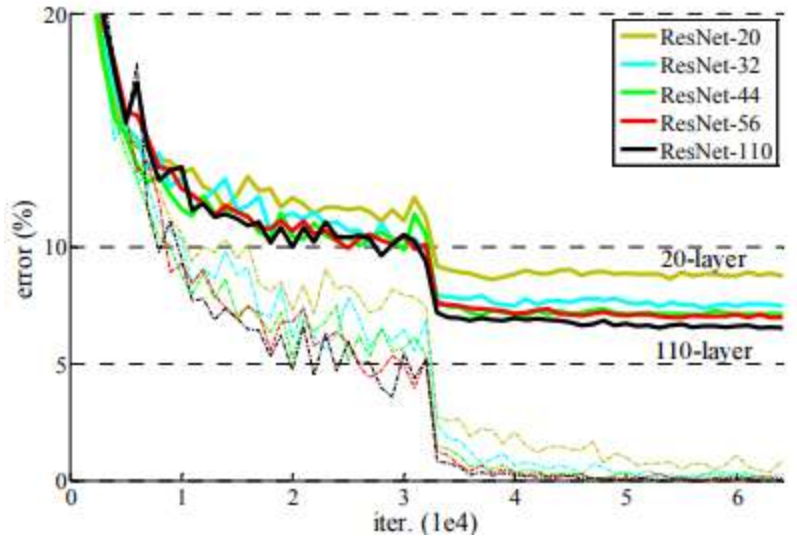
$$x_L = x_l + \sum_{i=l}^{L-1} F(x_i, W_i)$$

利用链式规则，可以求得反向过程的梯度：

$$\frac{\partial loss}{\partial x_l} = \frac{\partial loss}{\partial x_L} \cdot \frac{\partial x_L}{\partial x_l} = \frac{\partial loss}{\partial x_L} \cdot \left(1 + \frac{\partial}{\partial x_l} \sum_{i=l}^{L-1} F(x_i, W_i)\right)$$

式子的第一个因子 $\frac{\partial loss}{\partial x_L}$ 表示的损失函数到达 L 的梯度，小括号中的1表明短路机制可以无损地传播梯度，而另外一项残差梯度则需要经过带有weights的层，梯度不是直接传递过来的。残差梯度不会那么巧全为-1，而且就算其比较小，有1的存在也不会导致梯度消失。所以残差学习会更容易。要注意上面的推导并不是严格的证明。

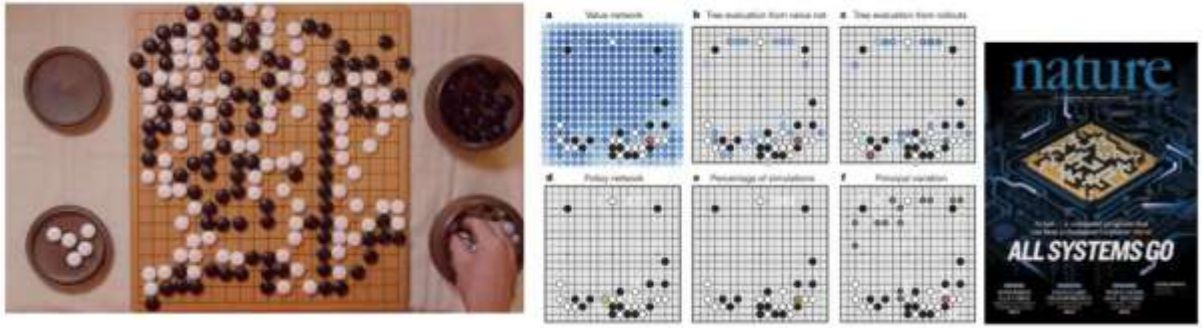




5.1.5 Application of Convolution Network

AlphaGo

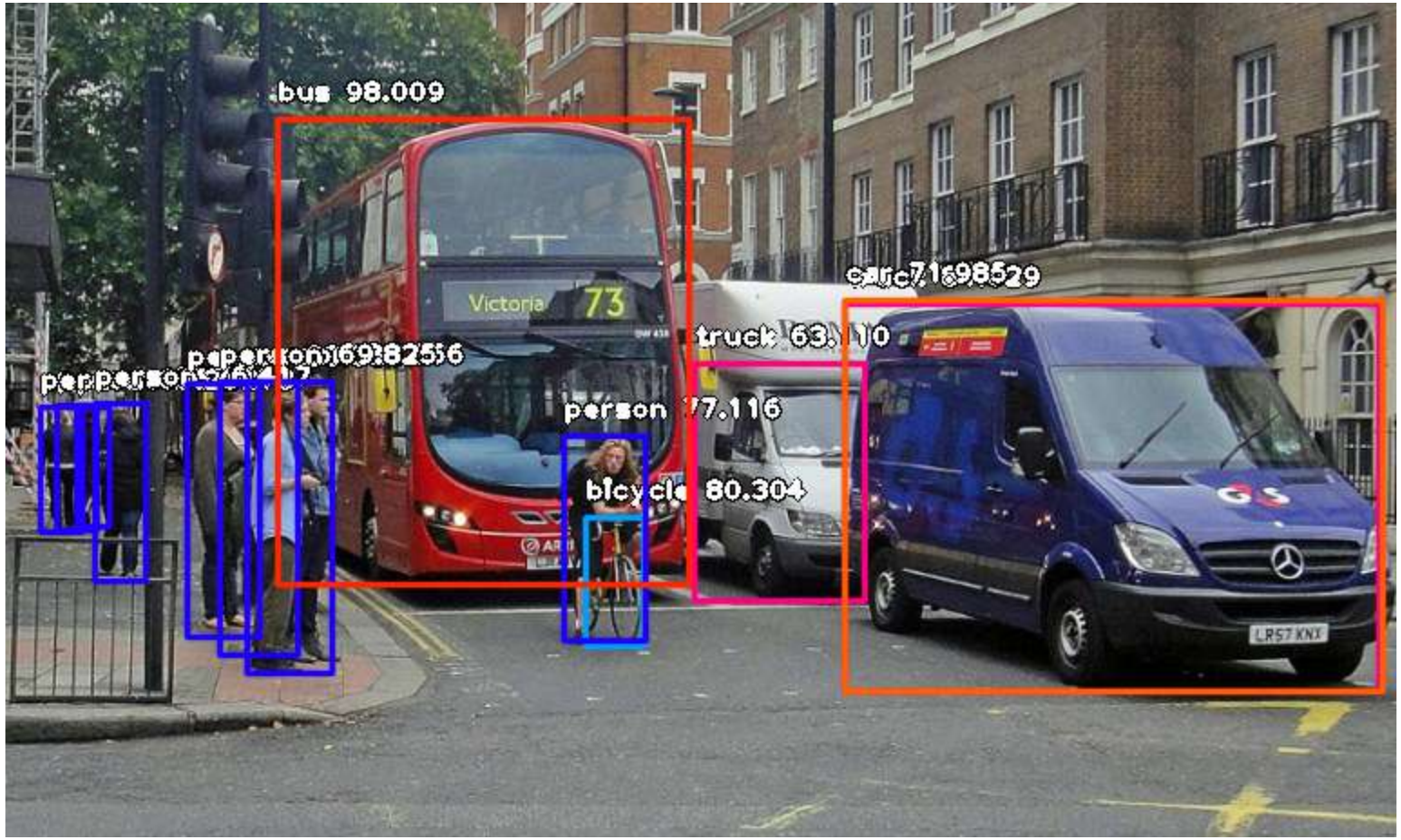
分布式系统：1202 个CPU 和176 块GPU
单机版：48 个CPU 和8 块GPU
走子速度：3 毫秒-2 微秒



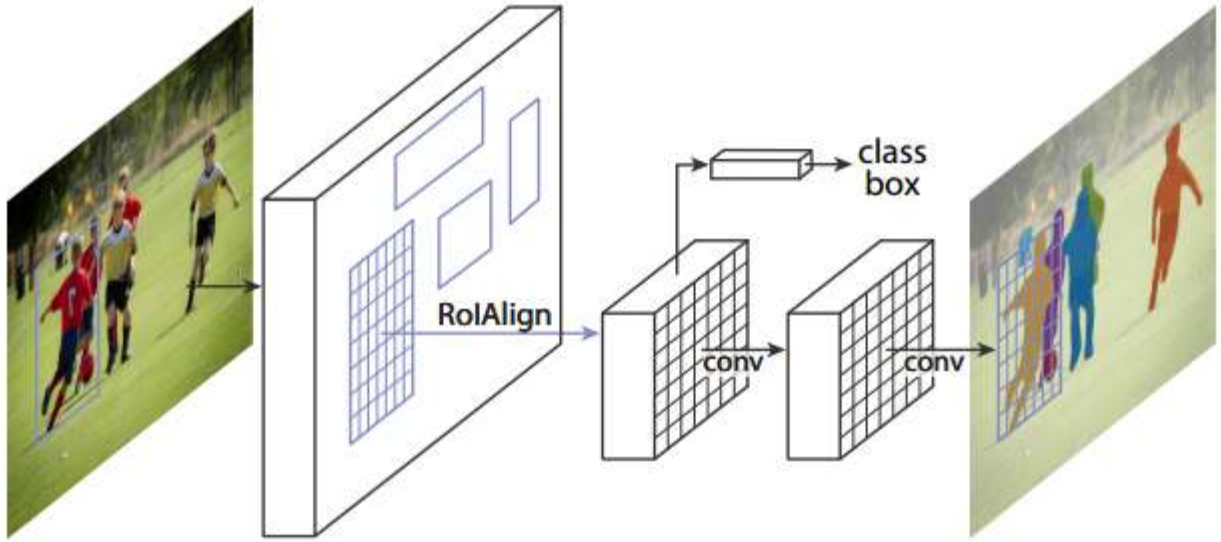
The input to the policy network is a 19 × 19 × 48 image stack consisting of 48 feature planes. The first hidden layer zero pads the input into a 23 × 23 image, then convolves *k* filters of kernel size 5 × 5 with stride 1 with the input image and applies a rectifier nonlinearity. Each of the subsequent hidden layers 2 to 12 zero pads the respective previous hidden layer into a 21 × 21 image, then convolves *k* filters of kernel size 3 × 3 with stride 1, again followed by a rectifier nonlinearity. The final layer convolves 1 filter of kernel size 1 × 1 with stride 1, with a different bias for each position, and applies a softmax function. The match version of AlphaGo used $k = 192$ filters; Fig. 2b and Extended Data Table 3 additionally show the results of training with $k = 128$, 256 and 384 filters.

policy network:
[19x19x48] Input
CONV1: 192 5x5 filters , stride 1, pad 2 => [19x19x192]
CONV2..12: 192 3x3 filters, stride 1, pad 1 => [19x19x192]
CONV: 1 1x1 filter, stride 1, pad 0 => [19x19] *(probability map of promising moves)*

Object Detection (目标检测)



Mask RCNN



He, K., Gkioxari, G., Dollár, P., & Girshick, R. (2017). Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision* (pp. 2961-2969).

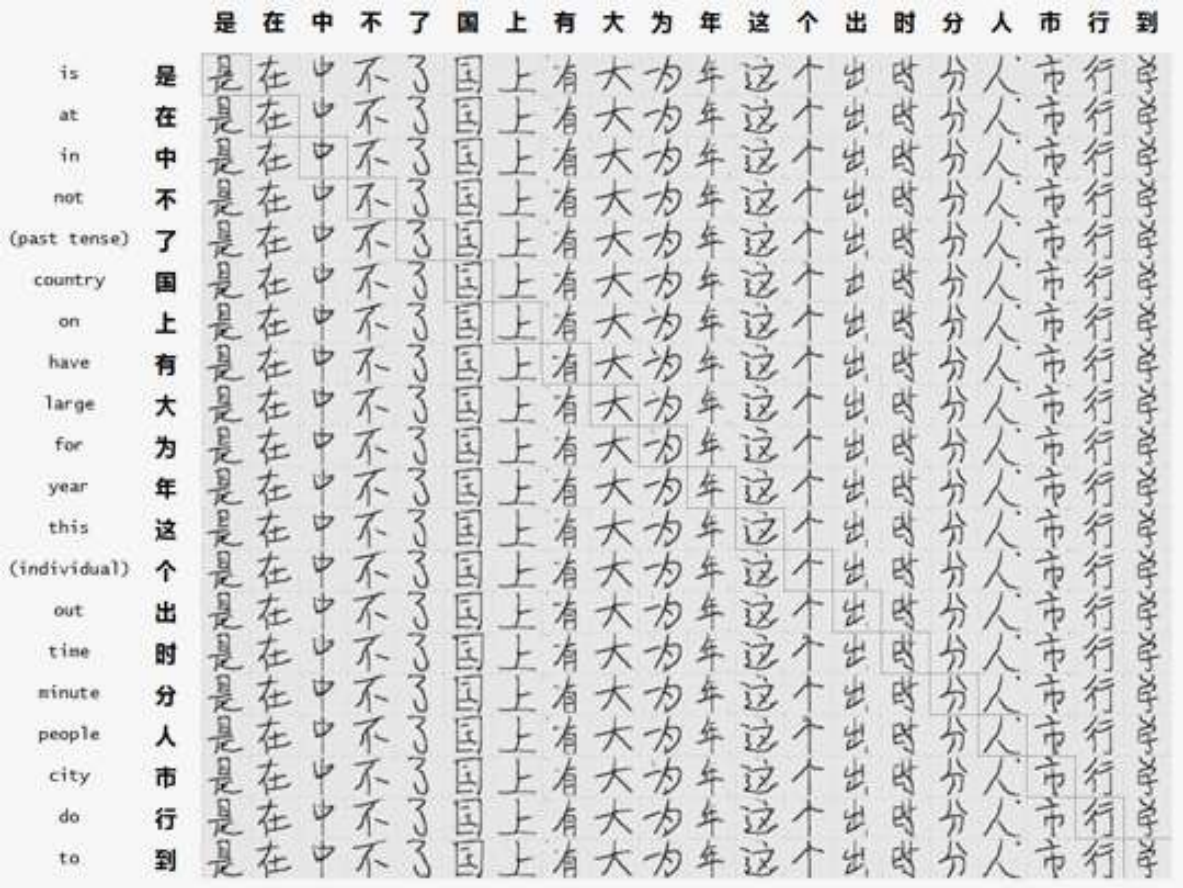
Mask RCNN

Figure 4. More results of **Mask R-CNN** on COCO test images, using ResNet-101-FPN and running at 5 fps, with 35.7 mask AP (Table 1).

OCR


THE WORLD FAMOUS
BRIXHAM
FISH MARKET
THE
ROYAL
SCOTSMAN
SWINGBRIDGE
WESTON
BREWING
1842
COMPANY
THIS WAY
AMERICAN
BUILT
ARMS COMPANY
GARDA
BMW
CLUB
CHARITY RUN
MIZEN HEAD
TO
MALIN HEAD
MAY 12TH 2007
HOTEL
GREEN
NANTUCKET

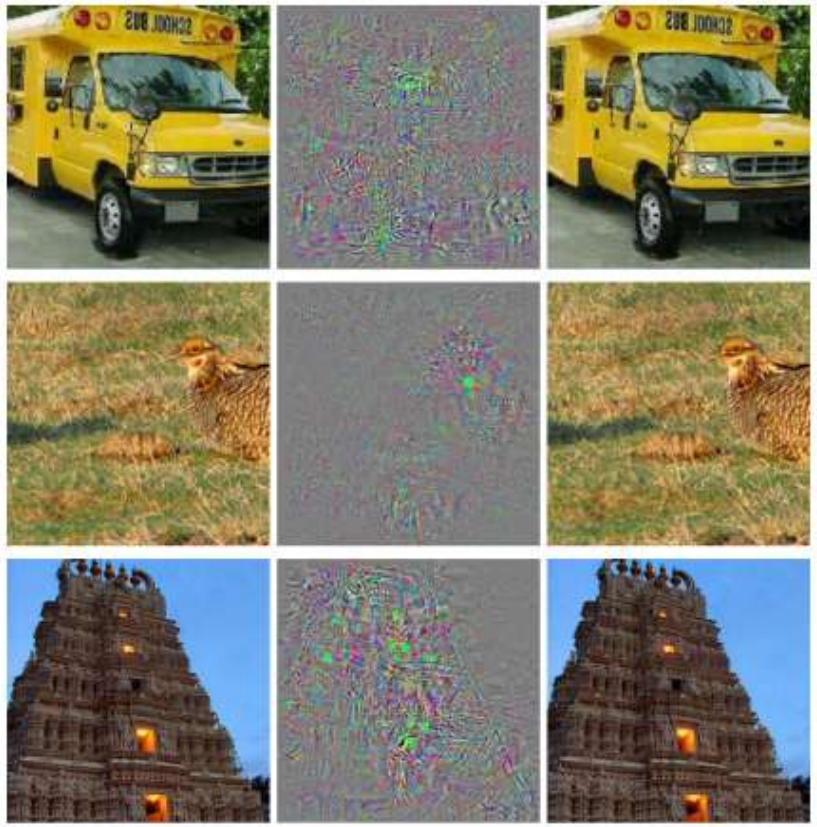
Image Generation (图像生成)



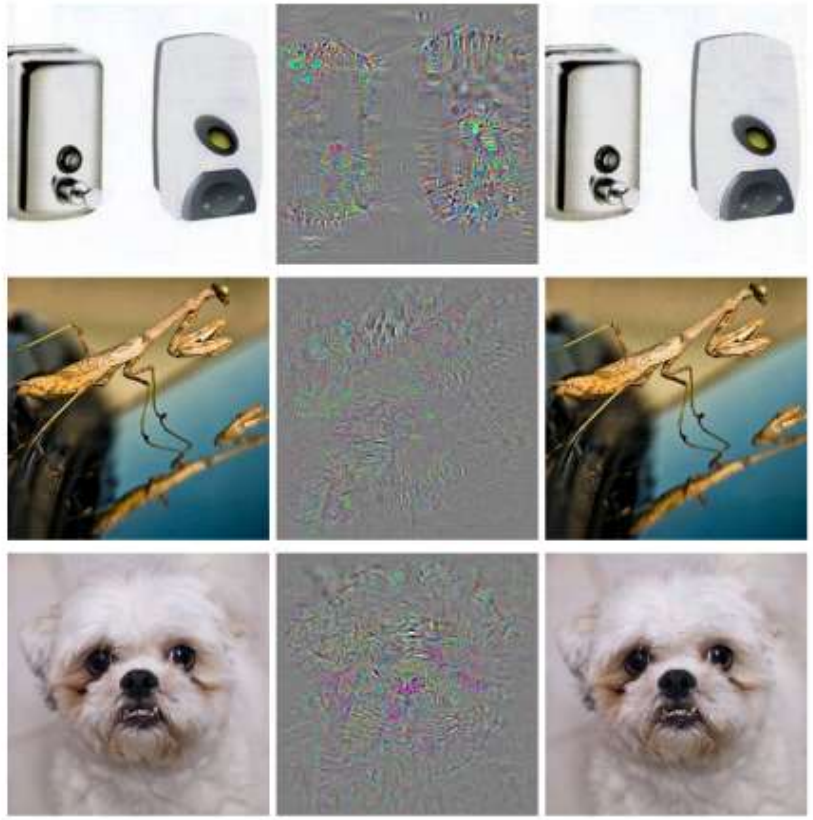
Deep Dream

Style Transfer (画风迁移)

Adversarial Samples



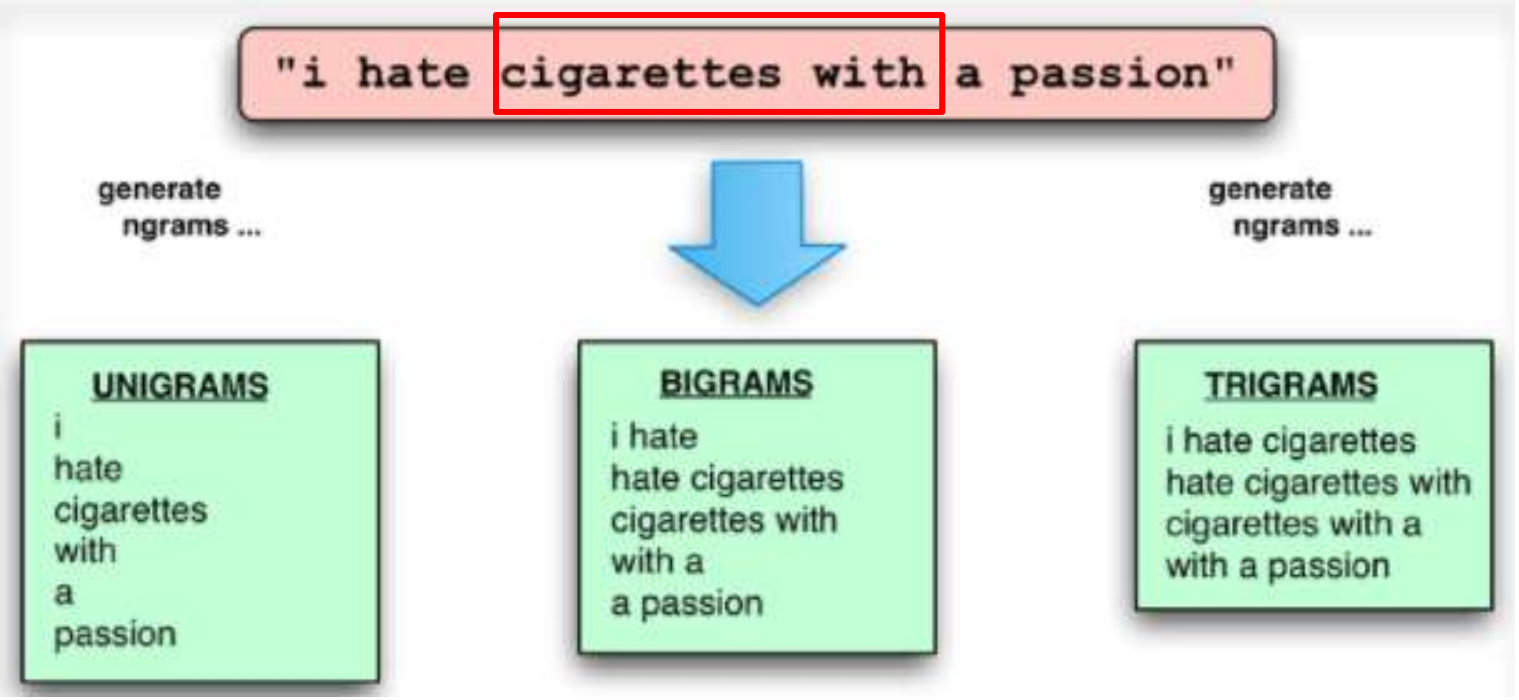
(a)



(b)

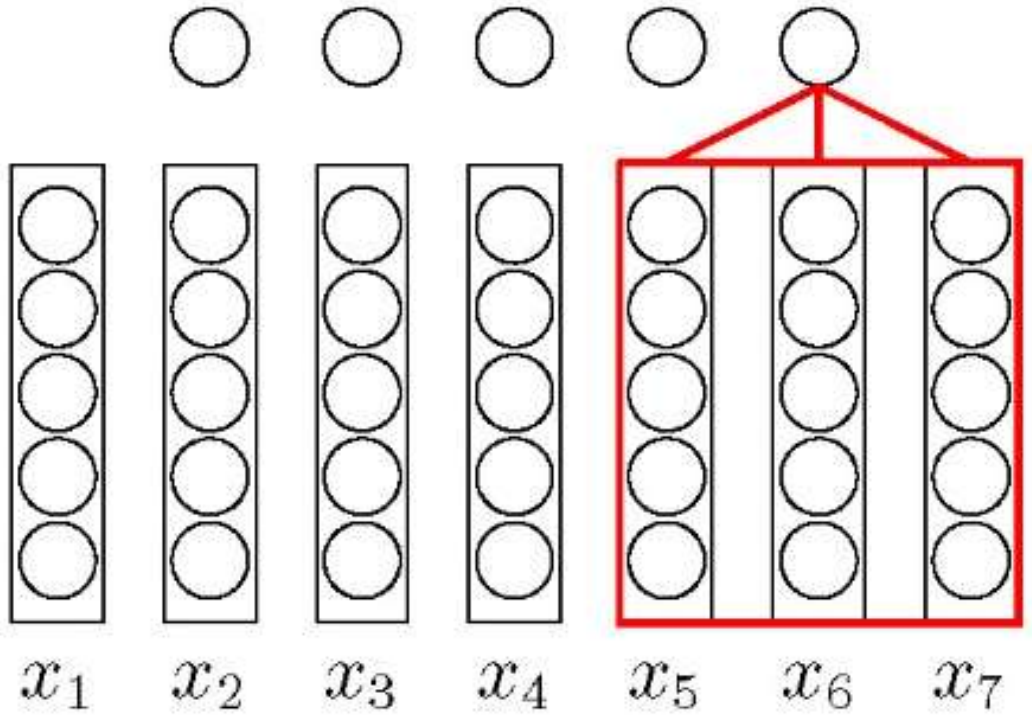
5.1.6 Apply to text data

Ngram Feature and Convolution

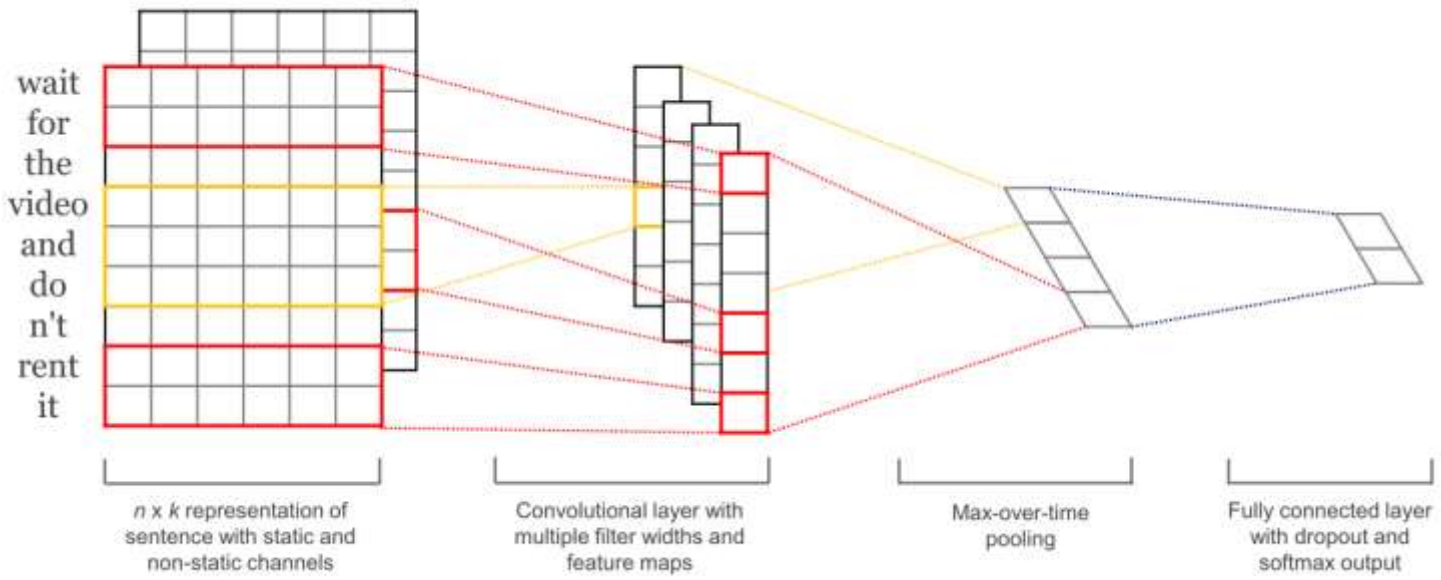


How to implement it with conv operation?

Convolution of Text Sequence

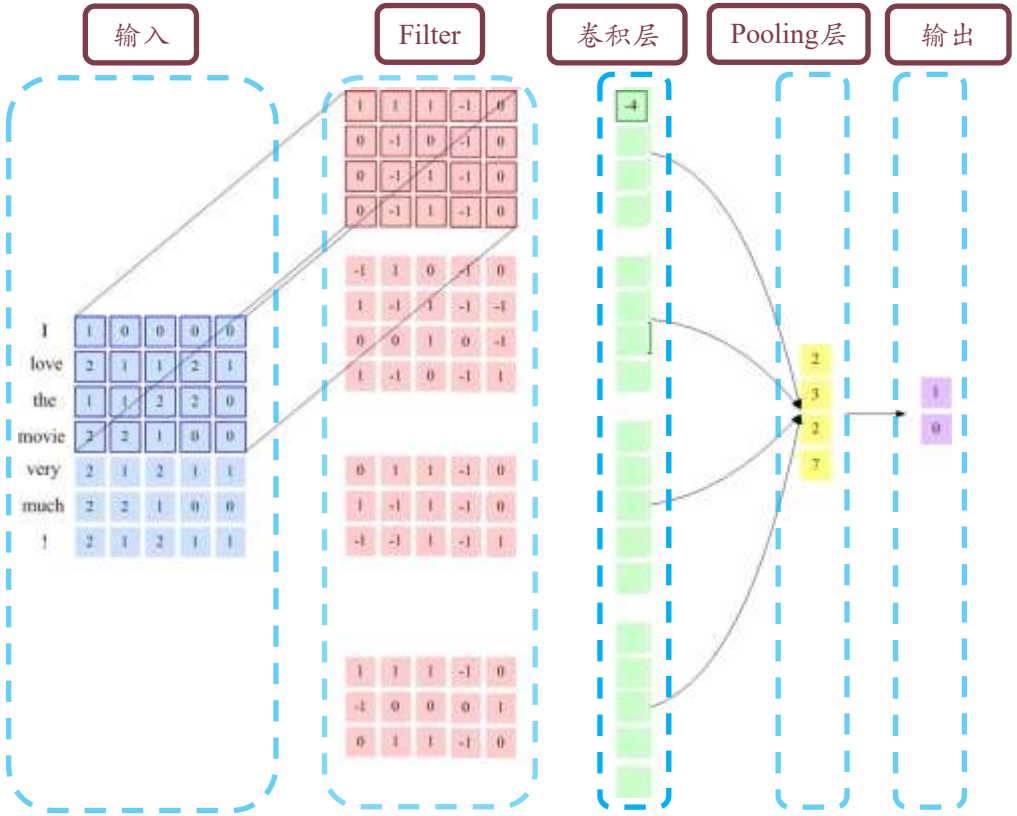


Sentence Representation Based on Convolution Model



Y. Kim. "Convolutional neural networks for sentence classification". In: *arXiv preprint arXiv:1408.5882* (2014).

Convolution Model of Text Sequence



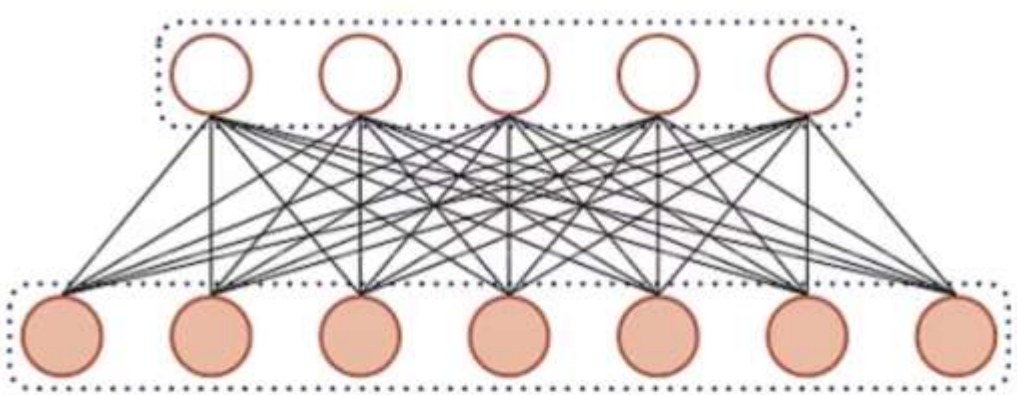
Chapter 5.2: Recurrent Neural Networks

Content

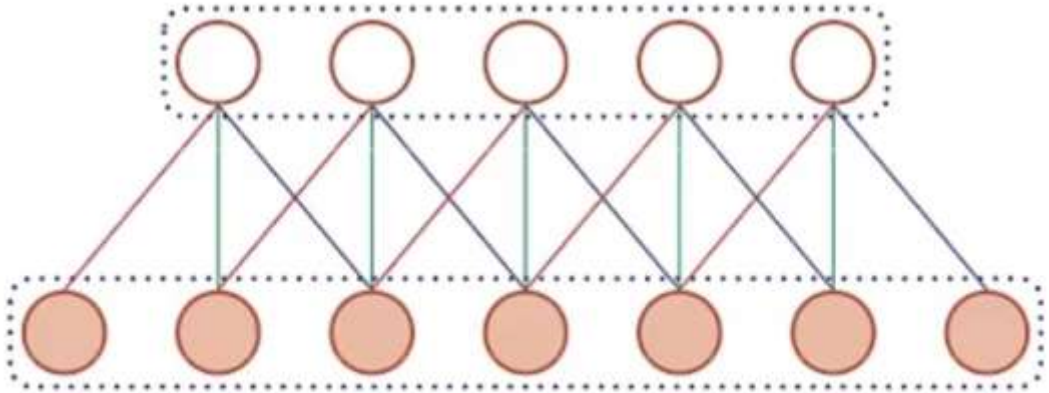
5.2.1 Adding Memory Ability to Neural Networks

Feedforward Neural Networks

- One-direction connection exists between two adjacent layers, and there is no connection in the layers.
  - Directed acyclic graph (DAG)
  - The dimensions of input and output are fixed and cannot be changed.
  - (Fully connected feedforward network) cannot handle variable-length sequence data.



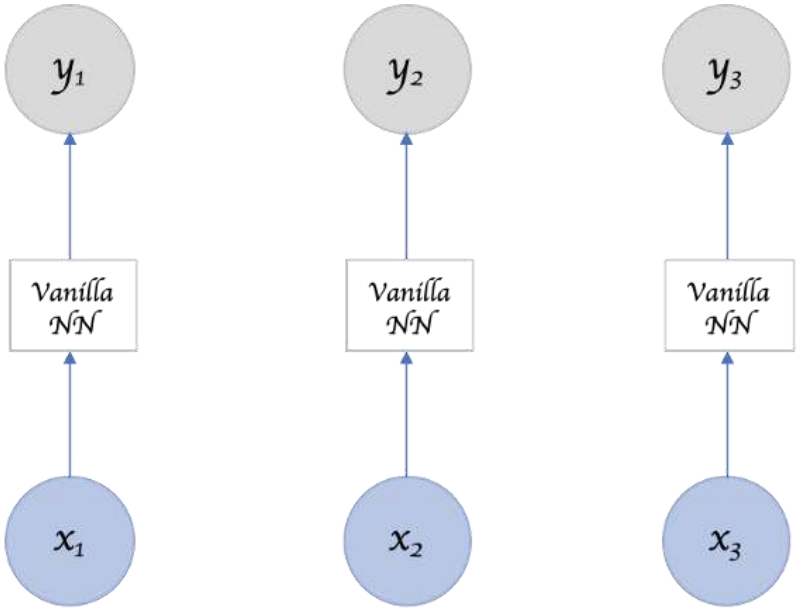
(a) 全连接层



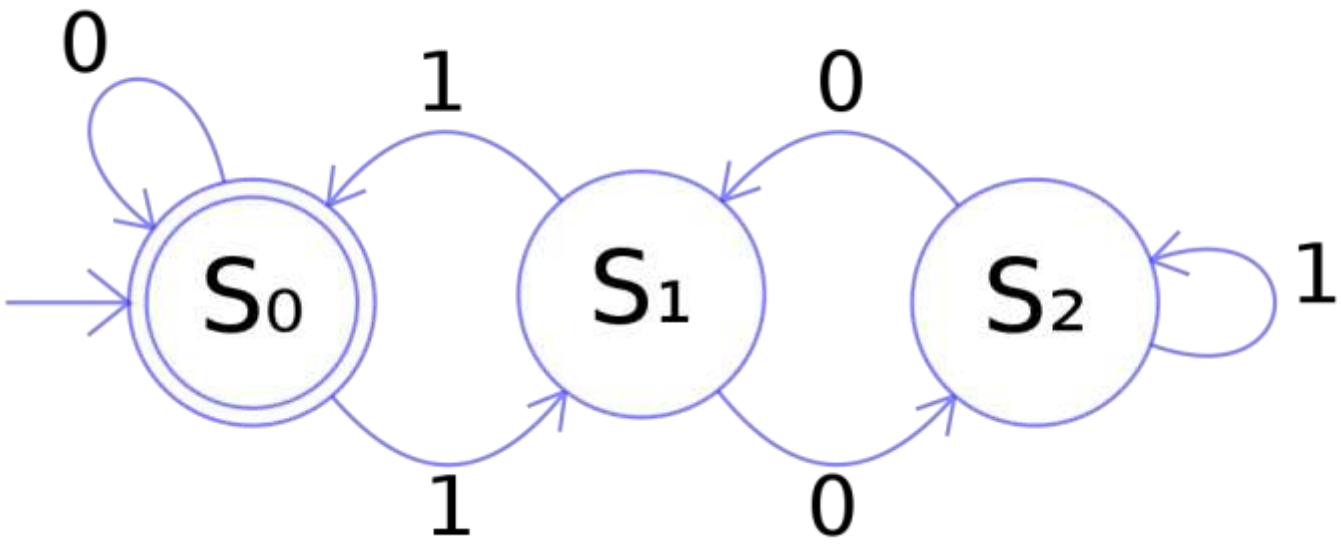
(b) 卷积层

Feedforward Neural Networks

- It is assumed that each input is independent, that is to say, each network output only depends on the current input.



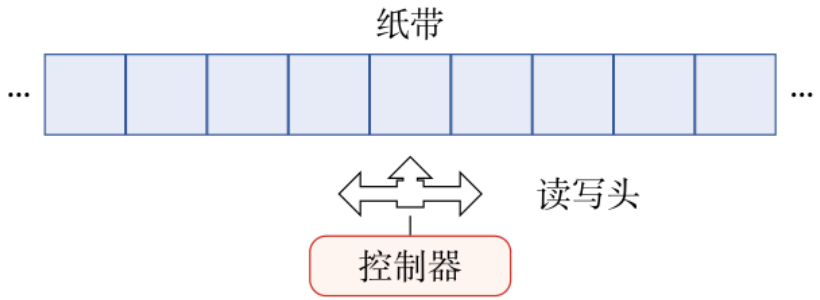
Finite Automata



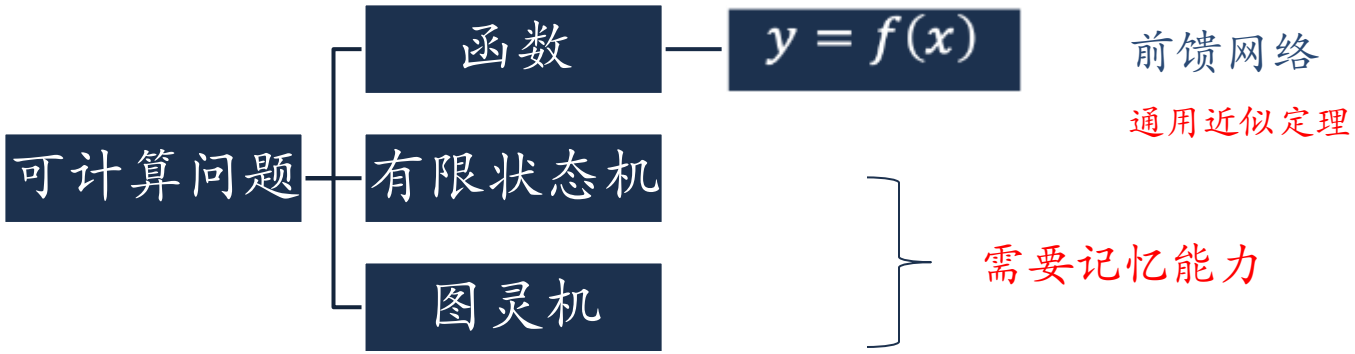
How to simulate a finite state automata with FNN?

Turing Machine

- An abstract mathematical model that can be used to simulate any computable problem.



Computable problem



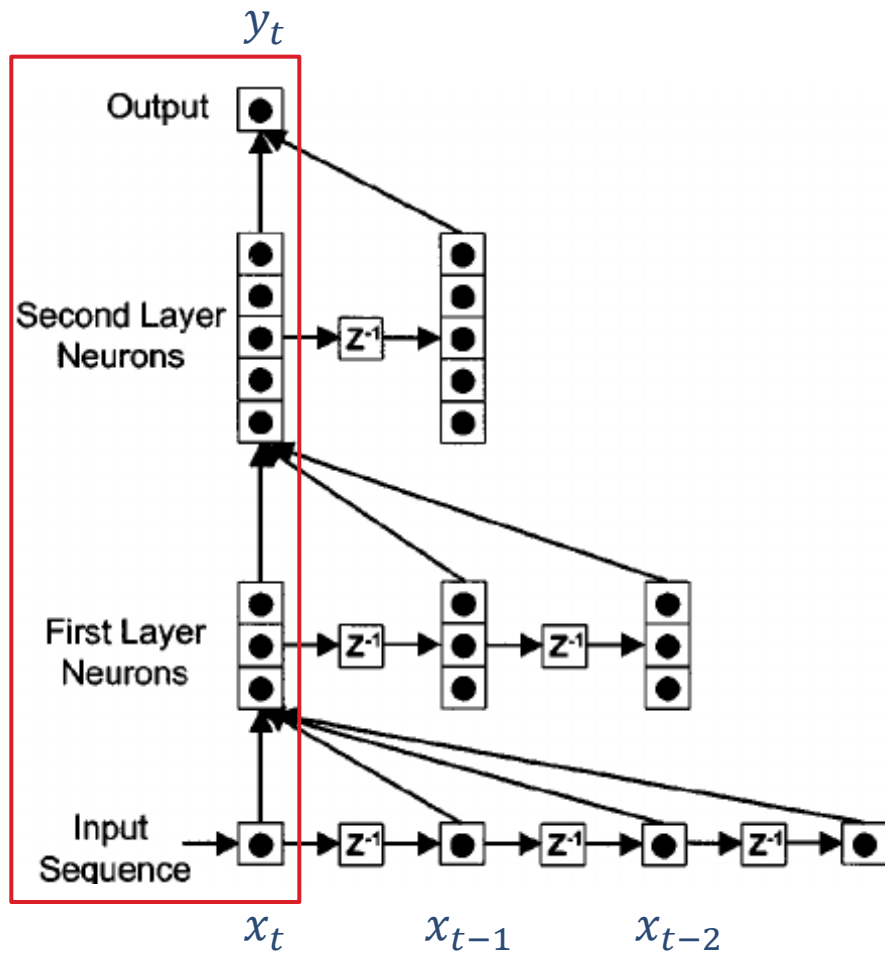
如何给网络增加记忆能力？

How to add memory ability to the network?

- Time Delay Neural Network (TDNN)
  - Establish an additional delay unit to store the historical information of the network (including input, output, hidden state, etc.)

$$\boldsymbol{h}_t^{(l)} = f\left(\boldsymbol{h}_t^{(l-1)}, \boldsymbol{h}_{t-1}^{(l-1)}, \cdots, \boldsymbol{h}_{t-K}^{(l-1)}\right)$$

  - In this way, FNN has the ability of short-term memory.



https://www.researchgate.net/publication/12314435_Neural_system_identification_model_of_human_sound_localization

How to add memory ability to the network?

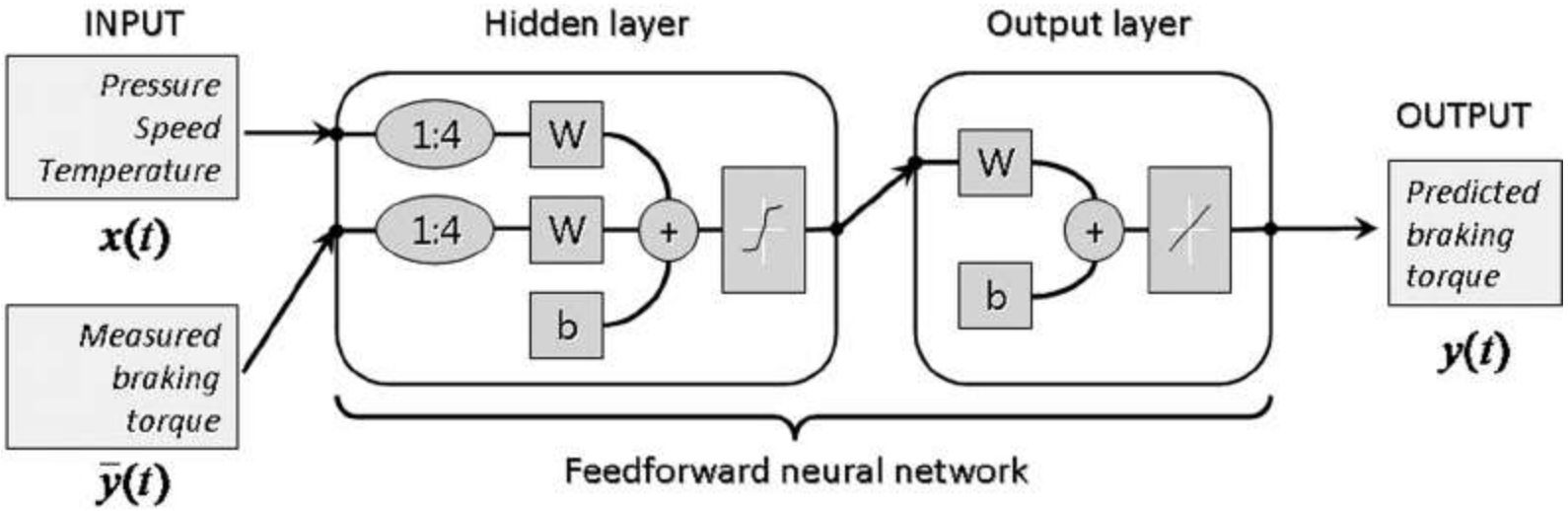
- Autoregressive Model (AR)
  - A kind of time series model that uses the historical information of variable y_t to predict itself.

$$\boldsymbol{y}_t = w_0 + \sum_{k=1}^{K} w_k \boldsymbol{y}_{t-k} + \epsilon_t$$

  - $\epsilon_t \sim N(0, \sigma^2)$ is the noise at time t
- Nonlinear Autoregressive with Exogenous Inputs Model (NARX)
  - $f(\cdot)$ is a nonlinear function, maybe it is a FNN
  - K_x and K_y are hyperparameters

$$\boldsymbol{y}_t = f(\boldsymbol{x}_t, \boldsymbol{x}_{t-1}, \cdots, \boldsymbol{x}_{t-K_x}, \boldsymbol{y}_{t-1}, \boldsymbol{y}_{t-2}, \cdots, \boldsymbol{y}_{t-K_y})$$

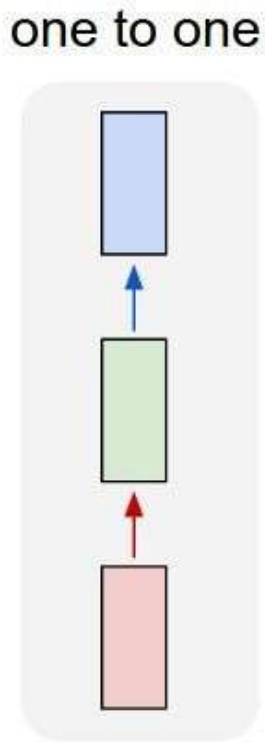
Nonlinear Autoregressive Model



https://www.researchgate.net/publication/234052442_Braking_torque_control_using_reccurent_neural_networks

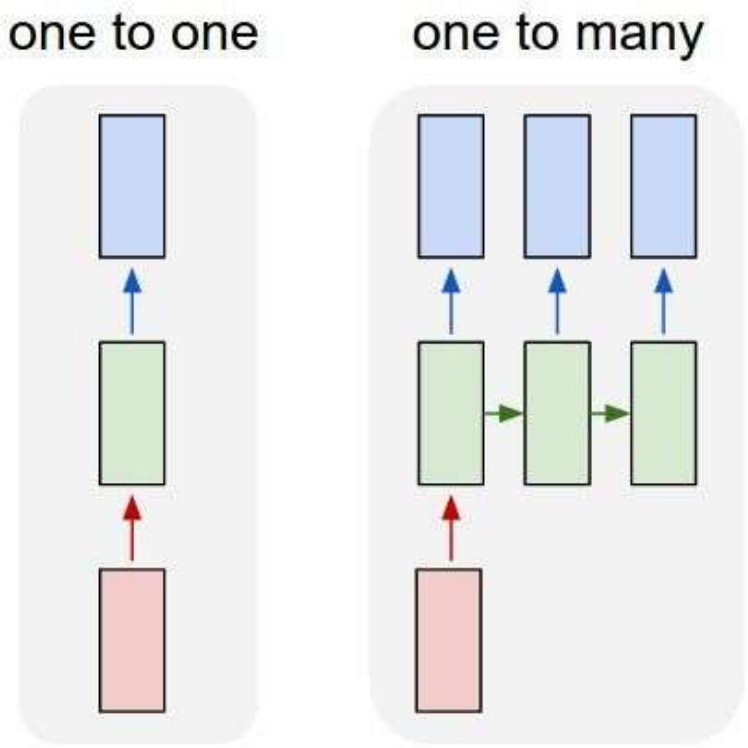
5.2.2 Recurrent Neural Networks

“Vanilla” Neural Network



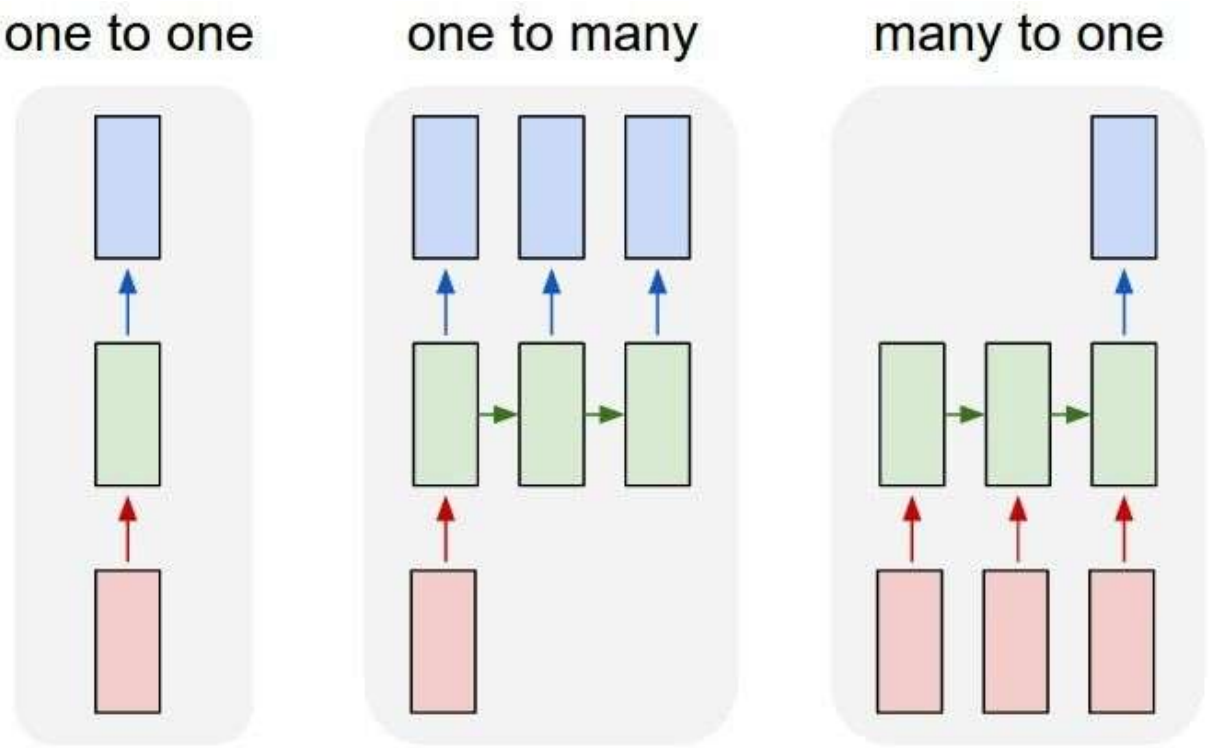
Vanilla Neural Networks

Recurrent Neural Networks: Process Sequences



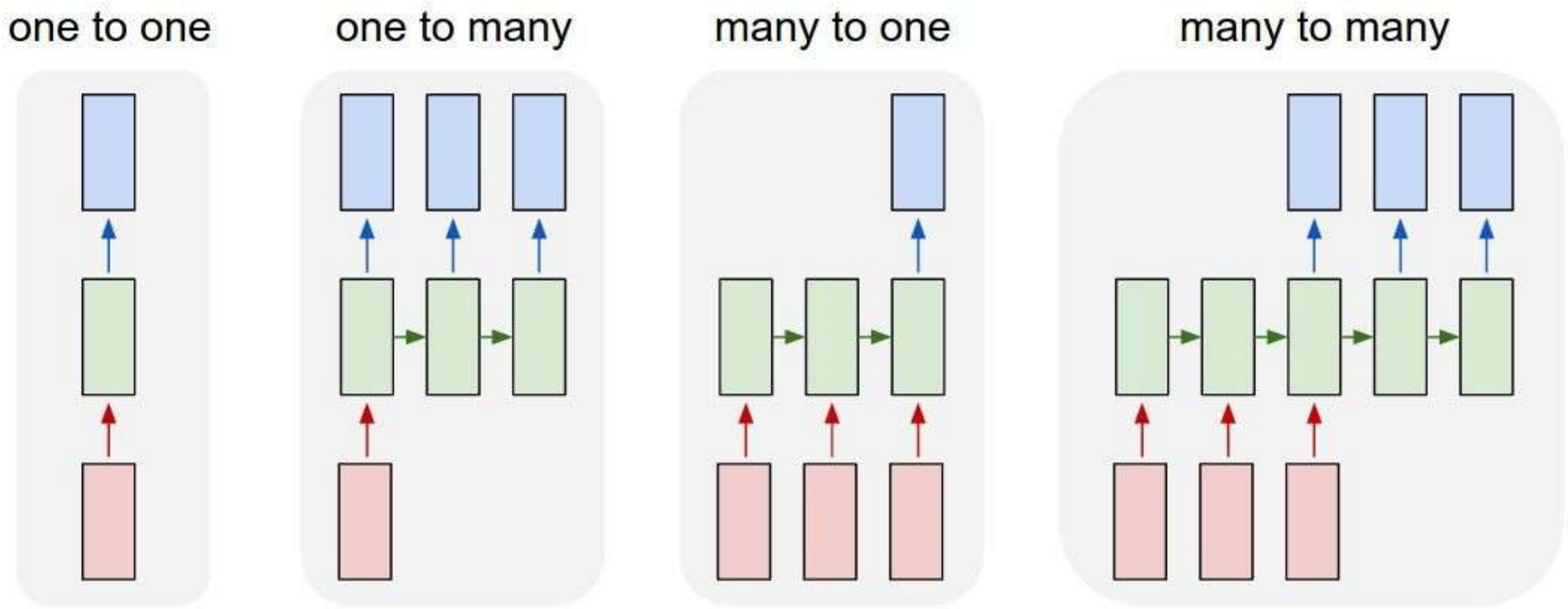
e.g. **Image Captioning**
image -> sequence of words

Recurrent Neural Networks: Process Sequences



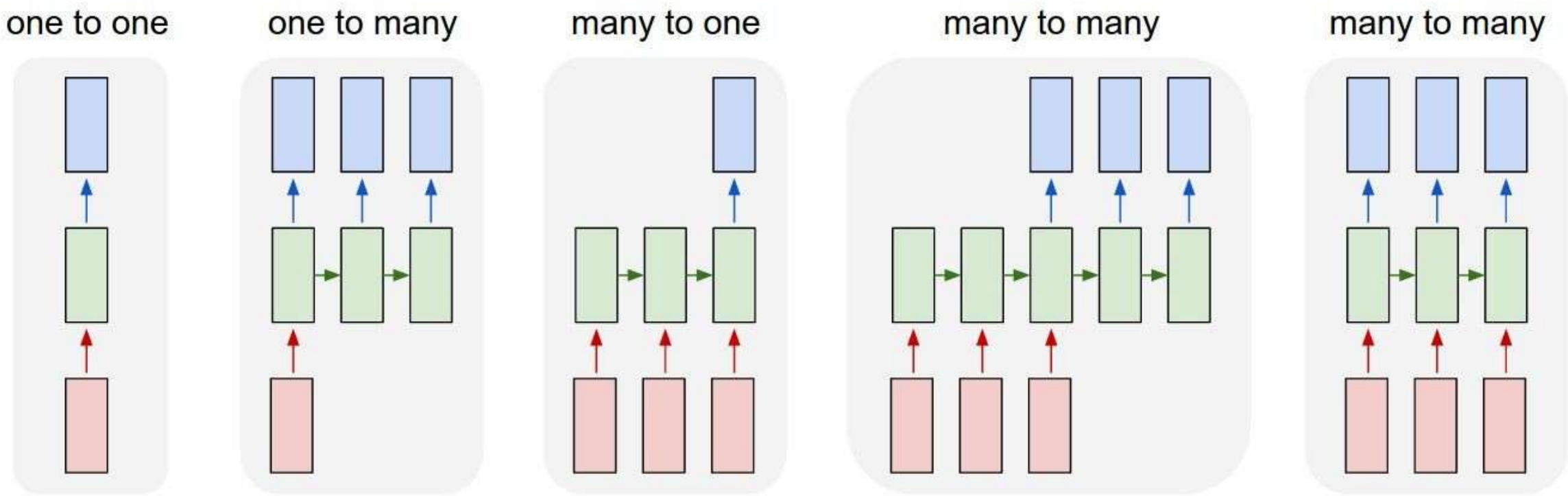
e.g. **action prediction**
sequence of video frames -> action class

Recurrent Neural Networks: Process Sequences



E.g. **Video Captioning**
Sequence of video frames -> caption

Recurrent Neural Networks: Process Sequences



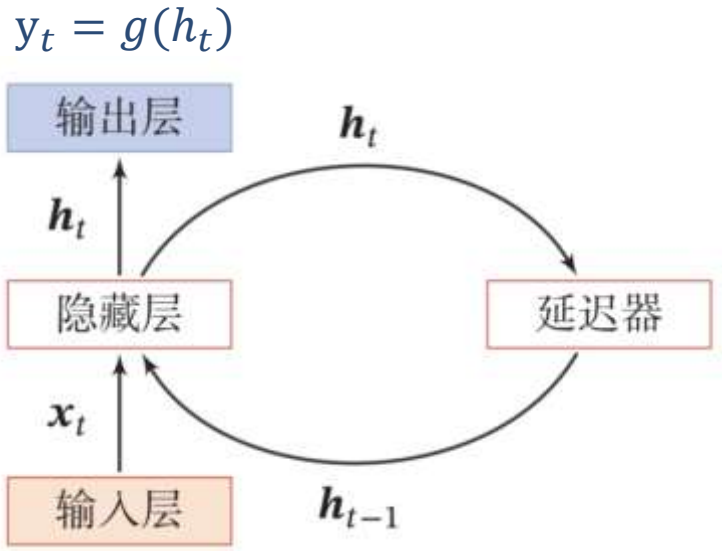
e.g. **Video classification on frame level**

Recurrent Neural Network (RNN)

- RNN can process time series data of any length by using neurons with self-feedback.

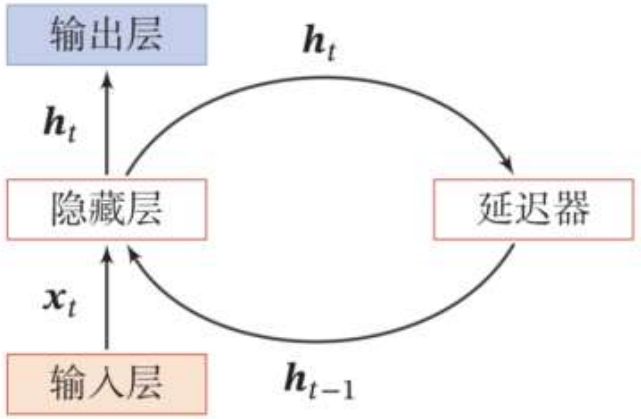
$$h_t = f(h_{t-1}, x_t)$$

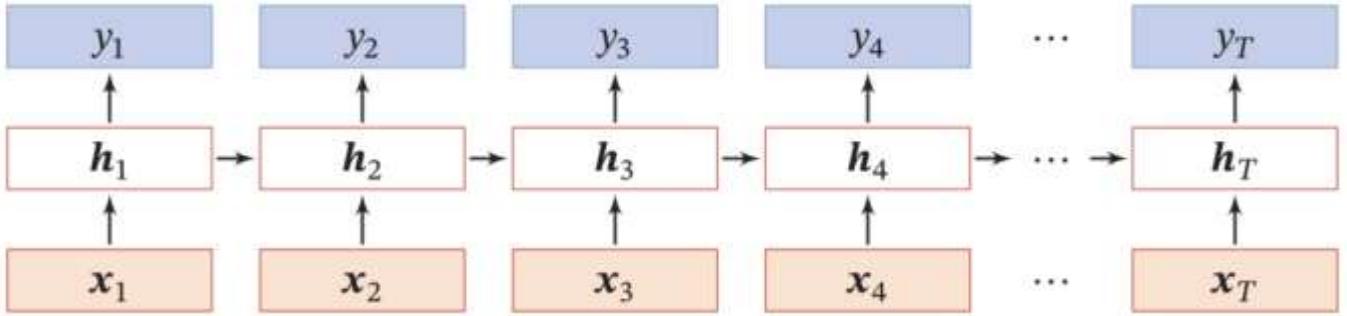
Activity value
State



- RNN is more in line with the biological structure than FNN.
- RNN has been widely used in speech recognition, language model and natural language generation.

Expand by Time





RNN hidden state update

We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

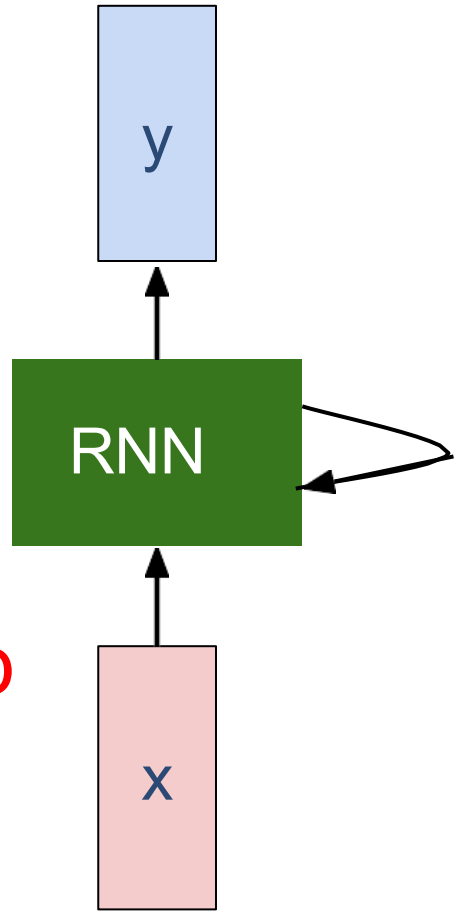
$$h_t = f_W(h_{t-1}, x_t)$$

new state

old state

input vector at some time step

some function with parameters W



RNN output generation

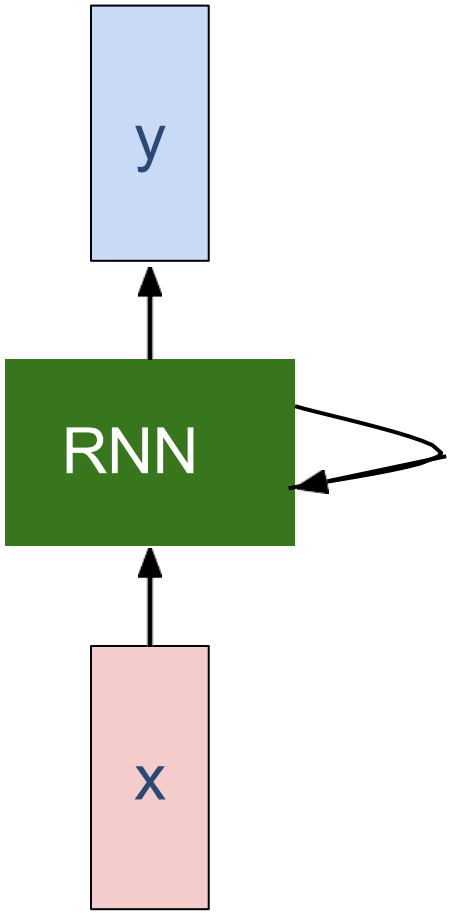
We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

$$y_t = f_{W_{hy}}(h_t)$$

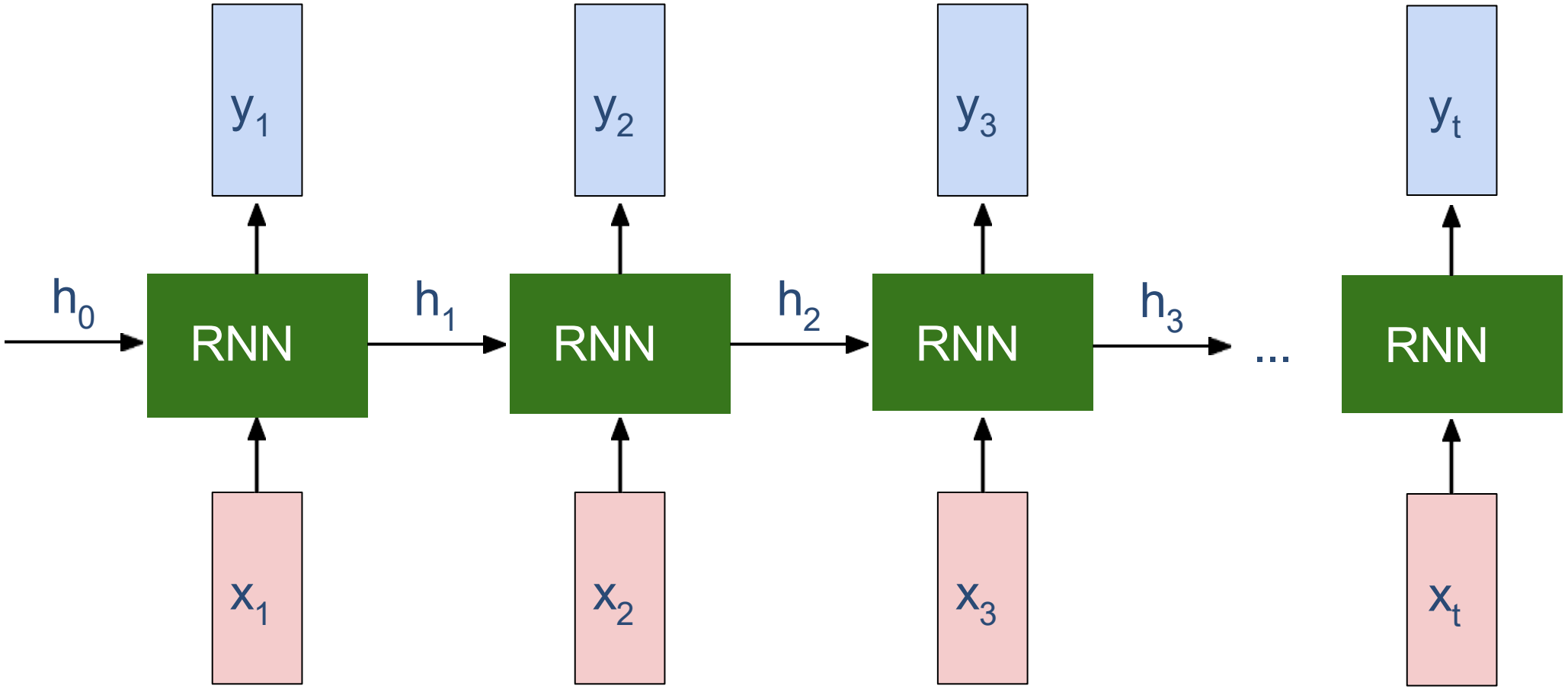
output

new state

another function with parameters W_o



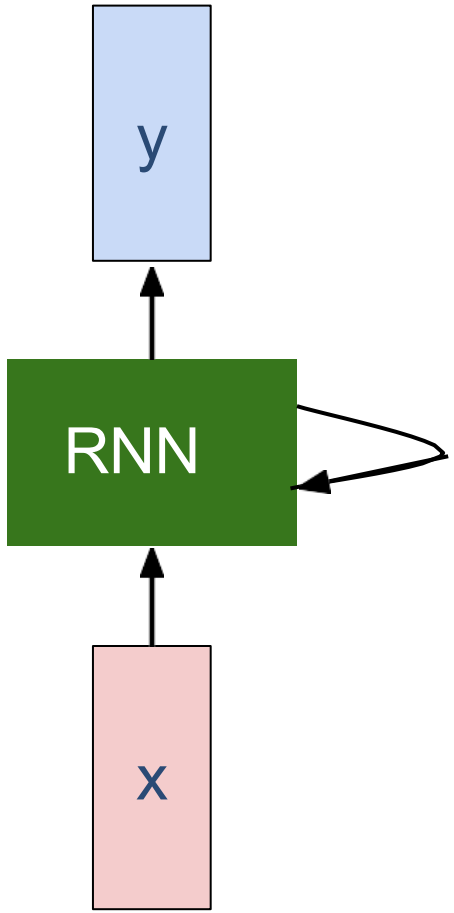
Recurrent Neural Network



Recurrent Neural Network

We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

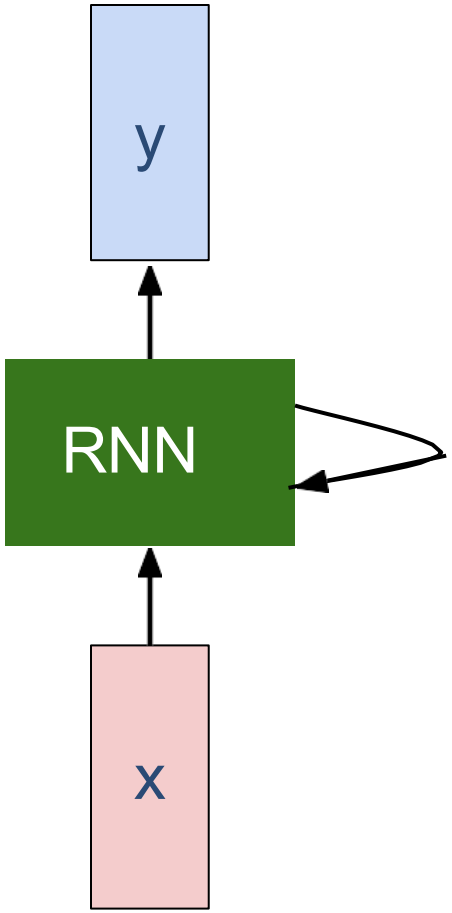
$$h_t = f_W(h_{t-1}, x_t)$$



Notice: the same function and the same set of parameters are used at every time step.

(Vanilla) Recurrent Neural Network

The state consists of a single *"hidden"* vector **h**:



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

$$h_t = f(\boldsymbol{U}\boldsymbol{h}_{t-1} + \boldsymbol{W}\boldsymbol{x}_t + \boldsymbol{b})$$

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

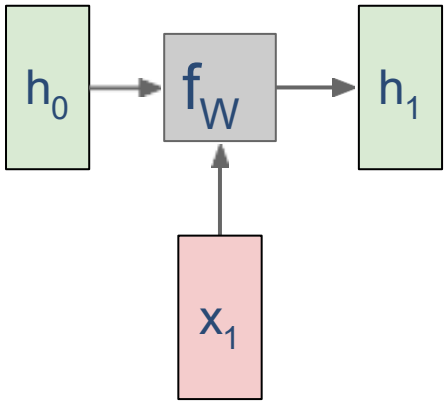
$$y_t = W_{hy}h_t$$

注: $x_t \in \mathbb{R}^M, h_t \in \mathbb{R}^D$; 状态-状态权重矩阵$W_{hh} \in \mathbb{R}^{D\times D}$，状态-输入权重矩阵$W_{xh} \in \mathbb{R}^{D\times M}$，偏置$b \in \mathbb{R}^D$

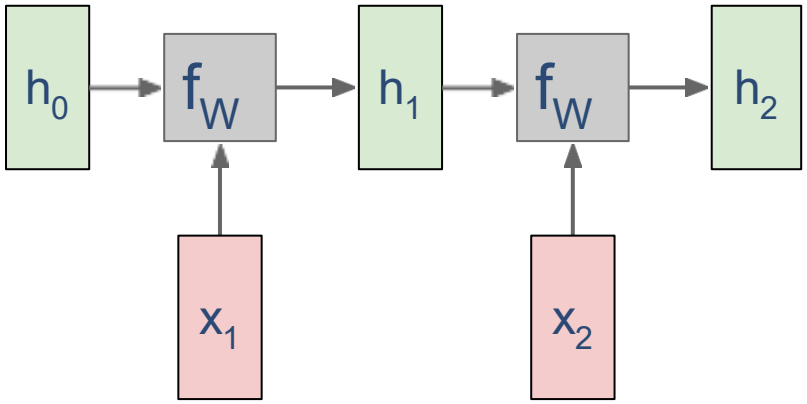
Sometimes called a "Vanilla RNN" or an "Elman RNN" after Prof. Jeffrey Elman

5.2.3 Applied to Machine Learning

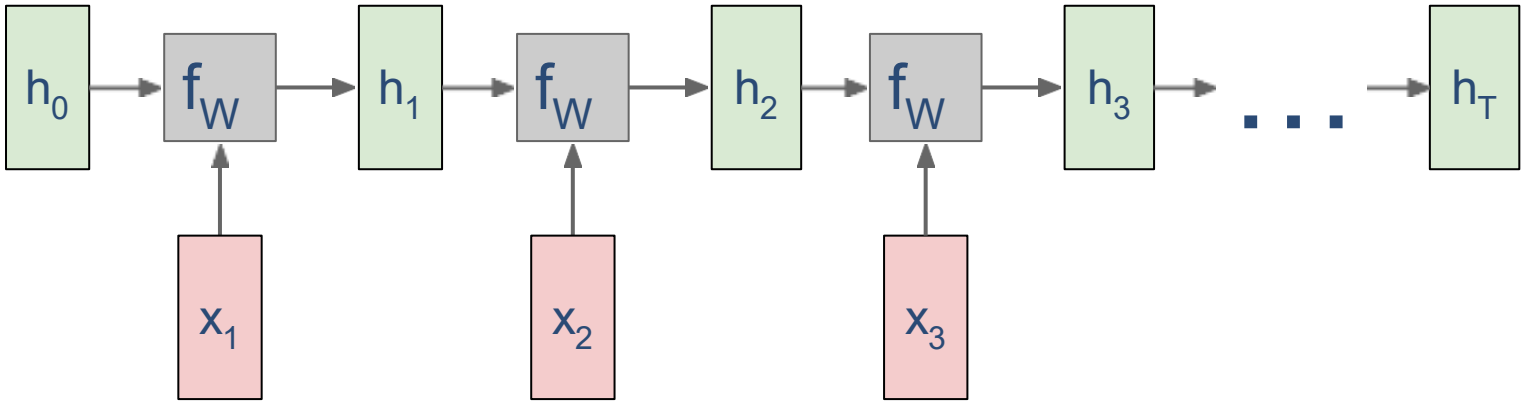
RNN: Computational Graph



RNN: Computational Graph

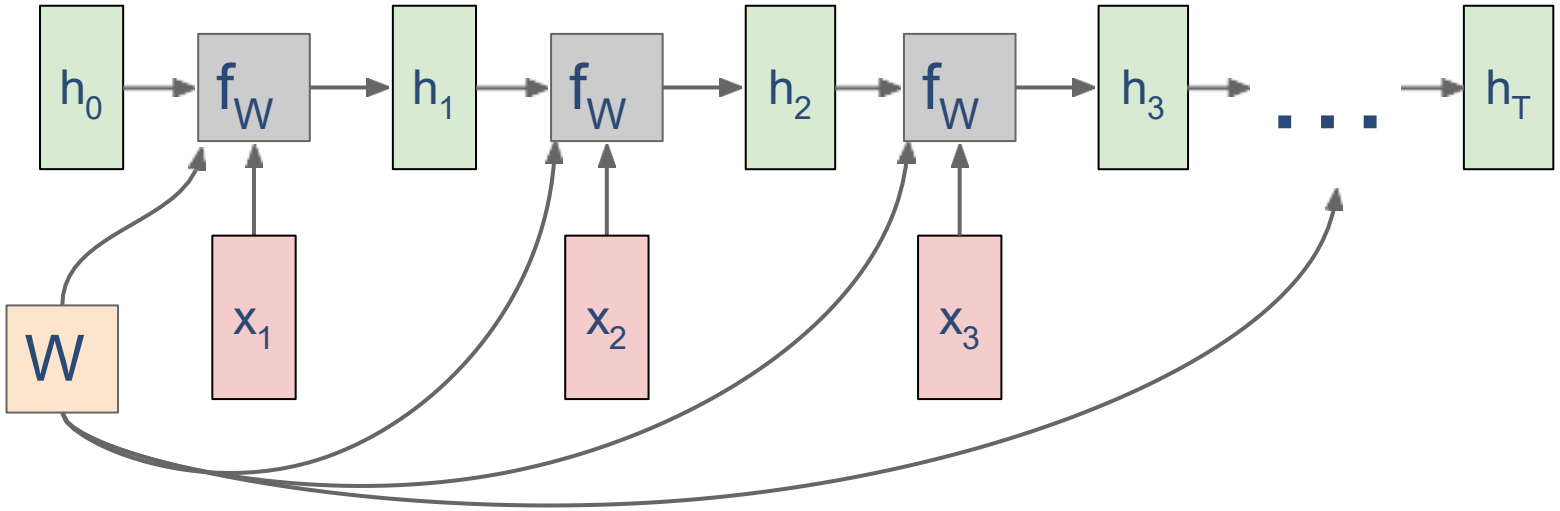


RNN: Computational Graph

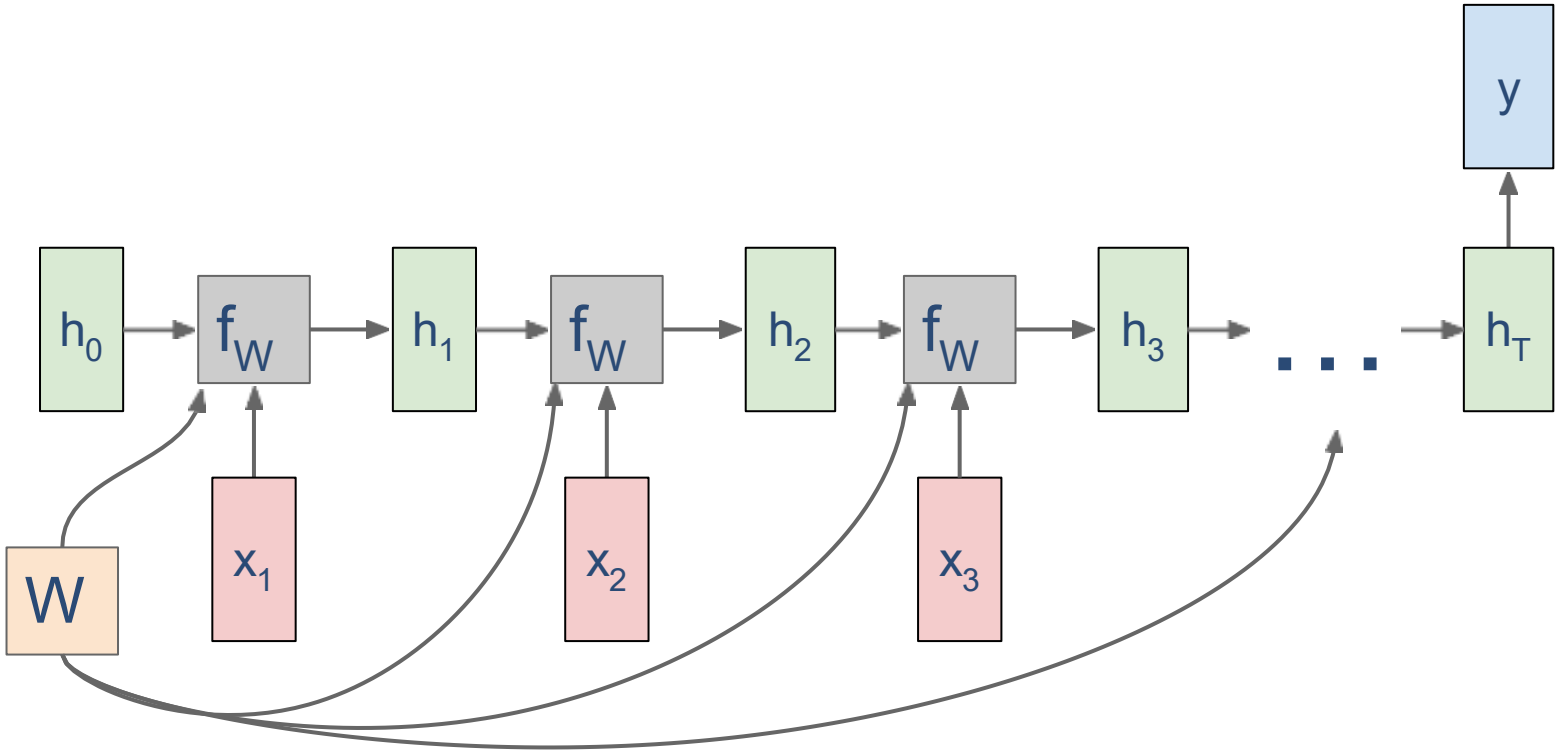


RNN: Computational Graph

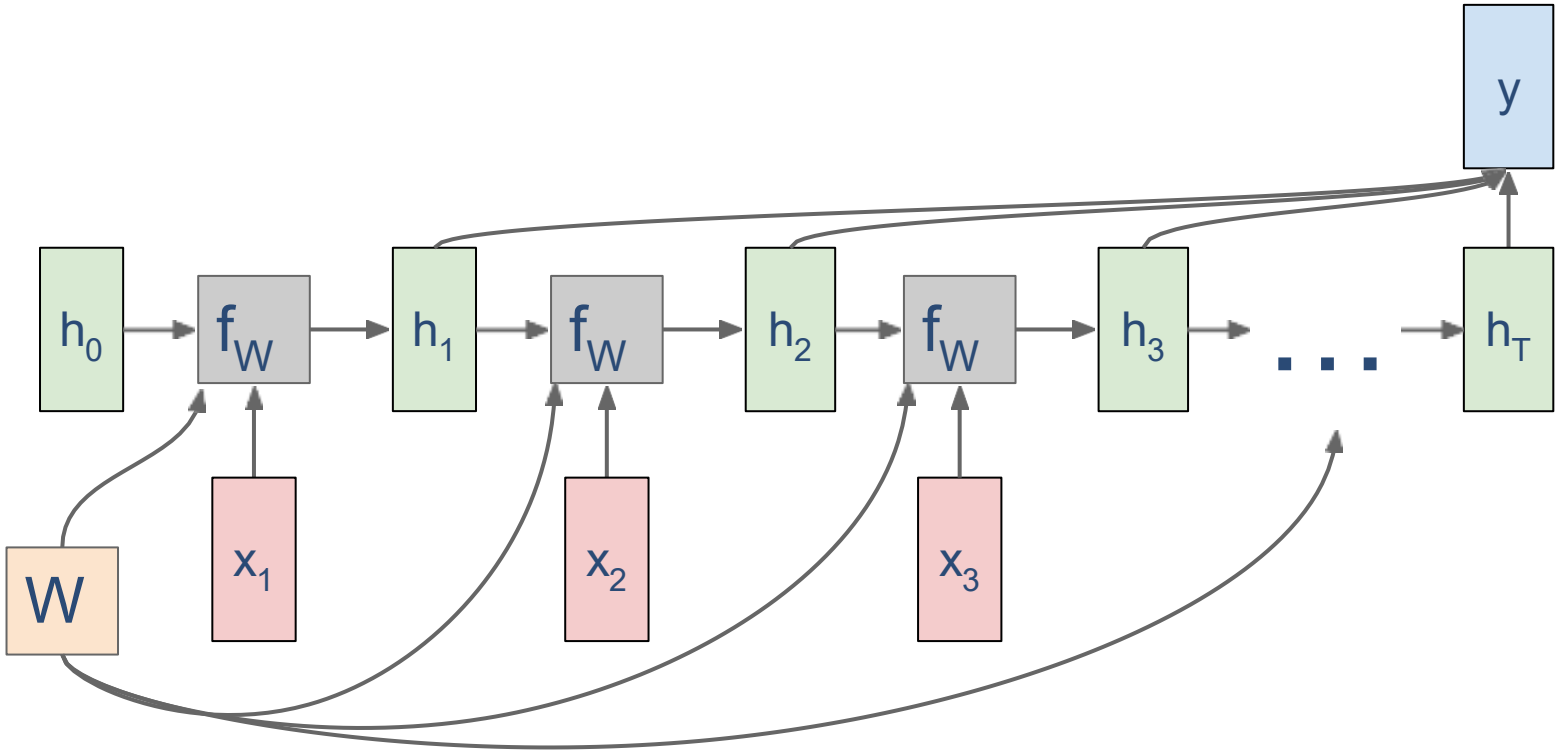
Re-use the same weight matrix at every time-step



RNN: Computational Graph: Many to One

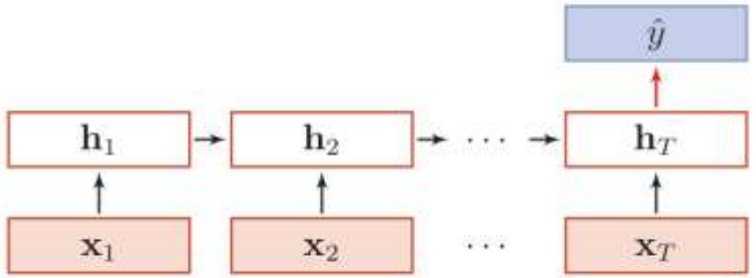


RNN: Computational Graph: Many to One

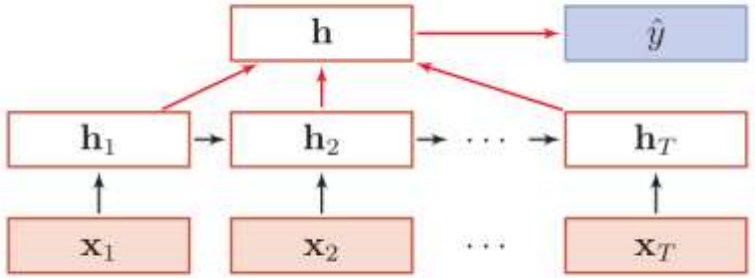


Applied to Machine Learning

- Sequence to class



(a) 正常模式



(b) 按时间进行平均采样模式

$$\hat{y} = g(\boldsymbol{h}_T),$$

$$\hat{y} = g\Big(\frac{1}{T}\sum_{t=1}^{T}\boldsymbol{h}_t\Big).$$

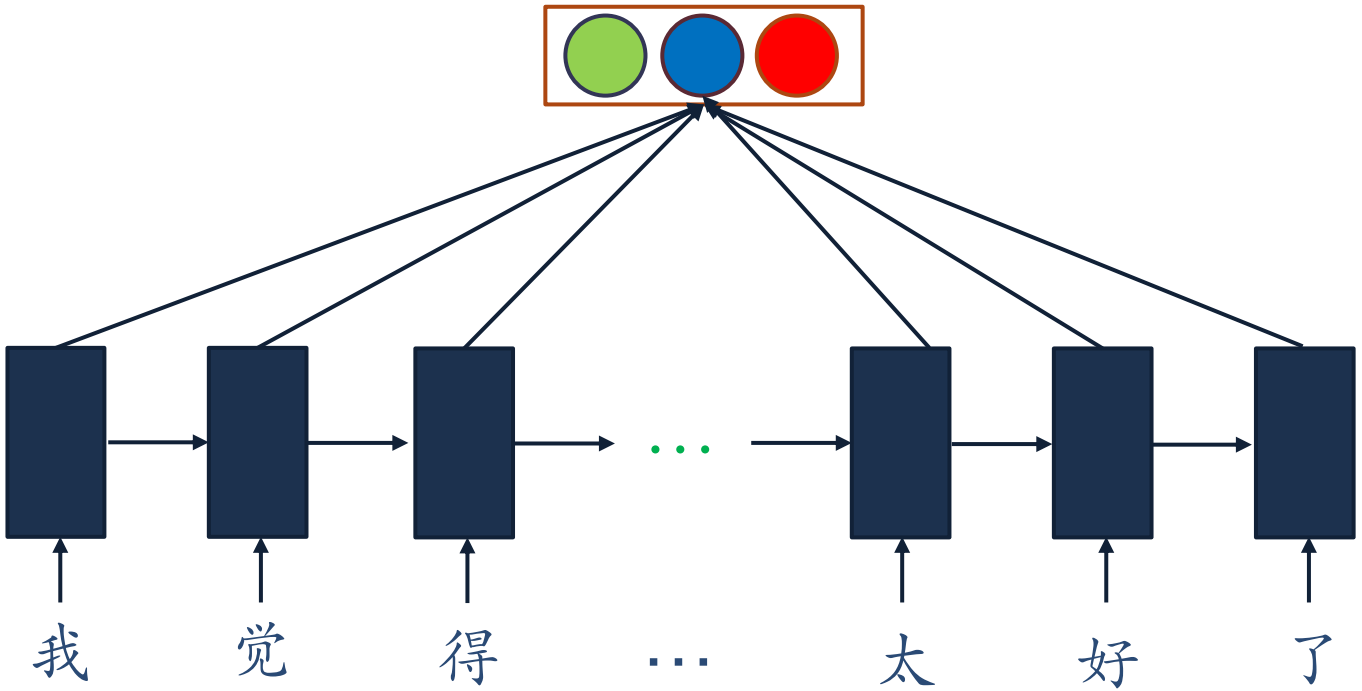
Sequence to Class

- Sentiment Analysis

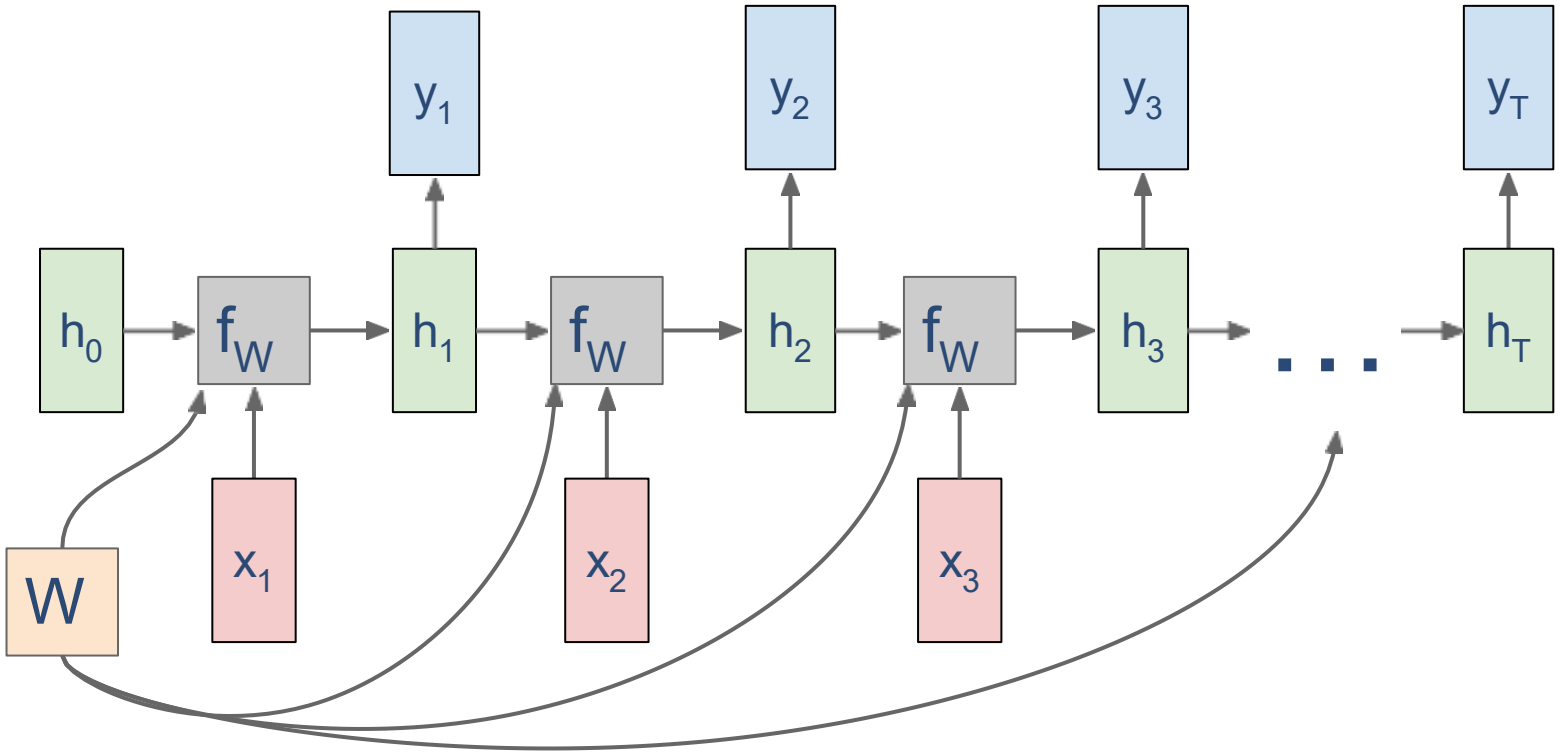
带着愉悦的心情看了这部电影 — Positive (正面)

这部电影太糟了 — Negative (负面)

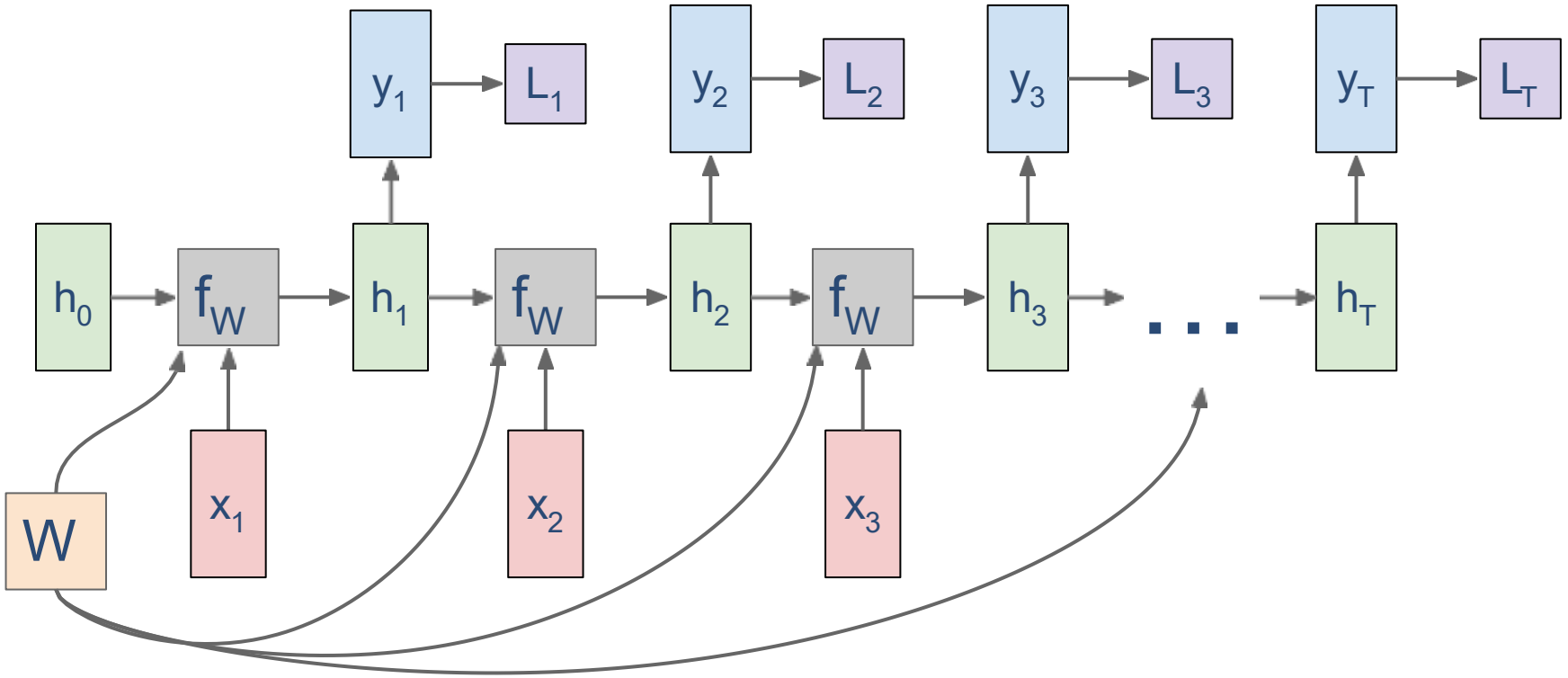
这部电影很棒 — Positive (正面)



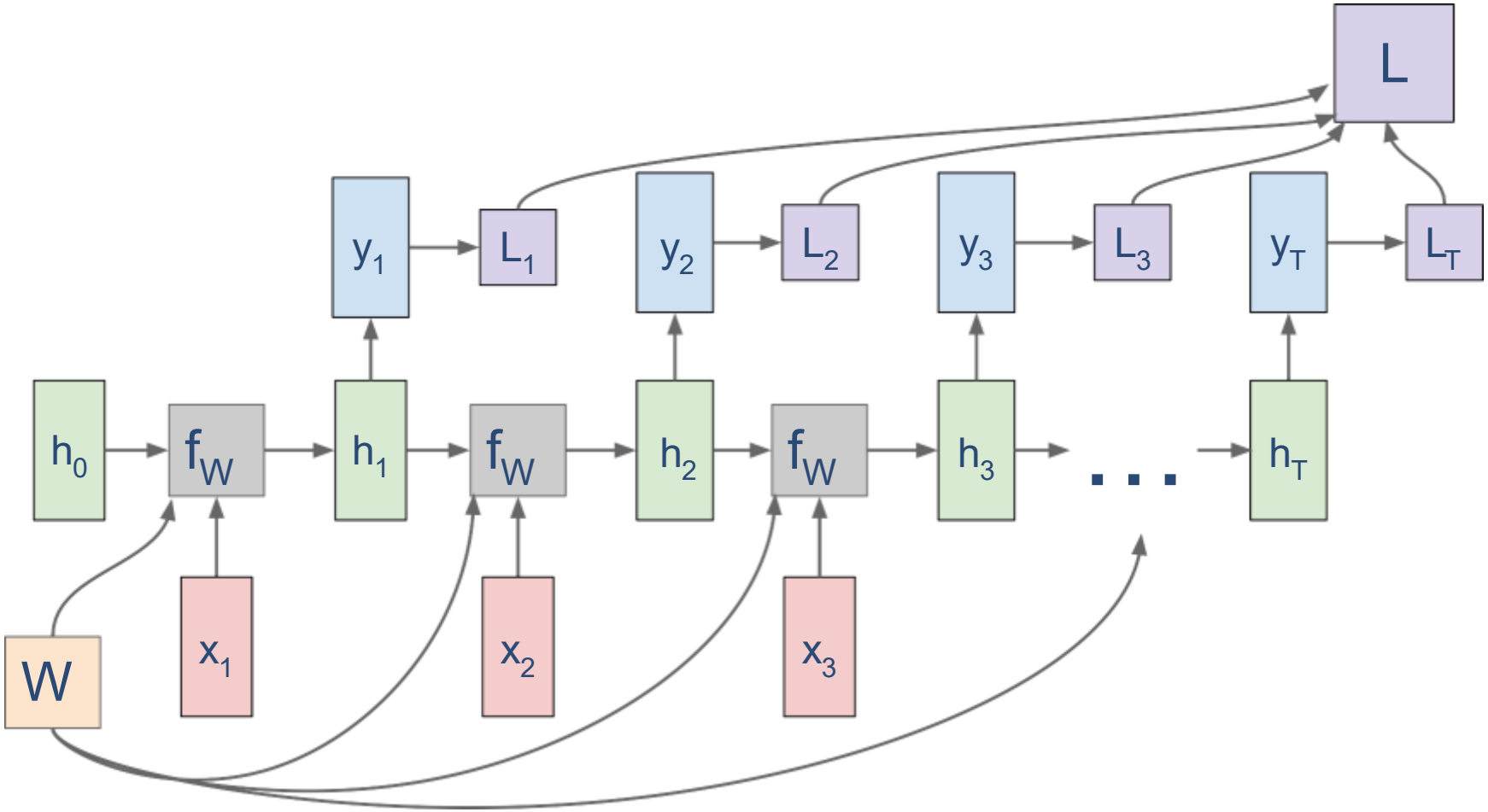
RNN: Computational Graph: Many to Many



RNN: Computational Graph: Many to Many

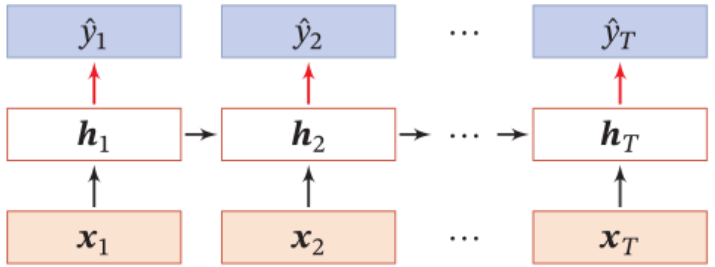


RNN: Computational Graph: Many to Many



Applied to Machine Learning

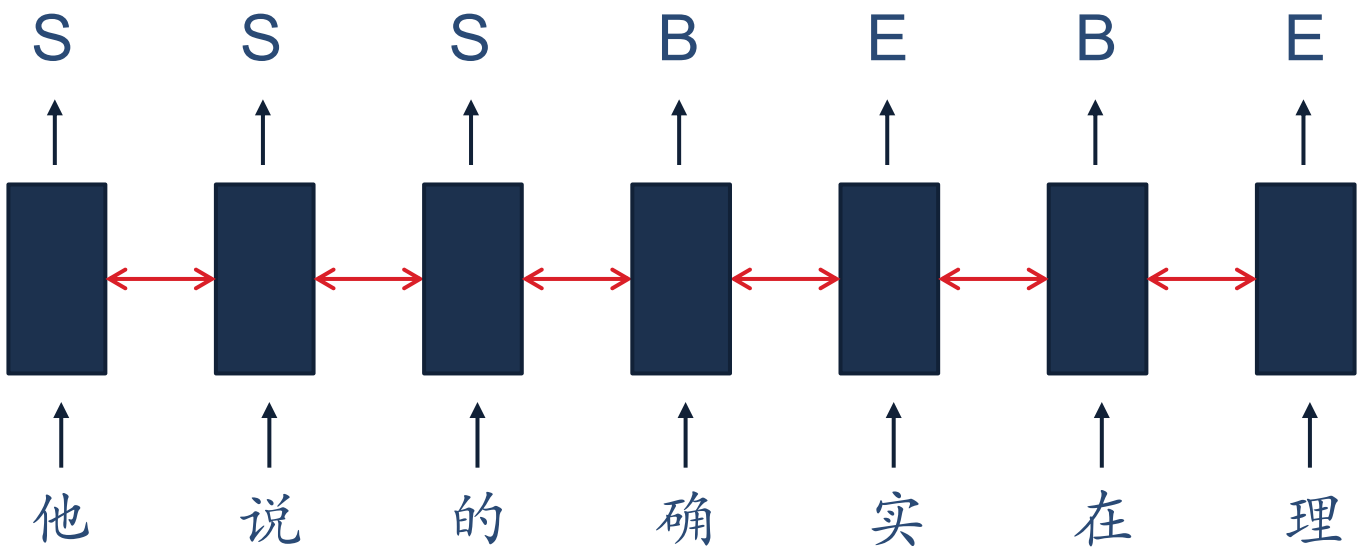
- Synchronous sequence-to-sequence



$$\hat{y}_t = g(\boldsymbol{h}_t), \qquad \forall t \in [1, T].$$

Synchronous sequence-to-sequence

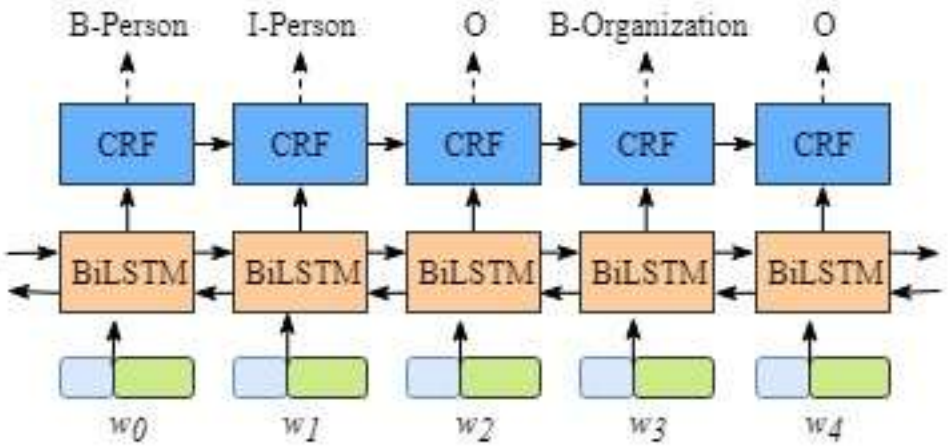
- Chinese word segmentation



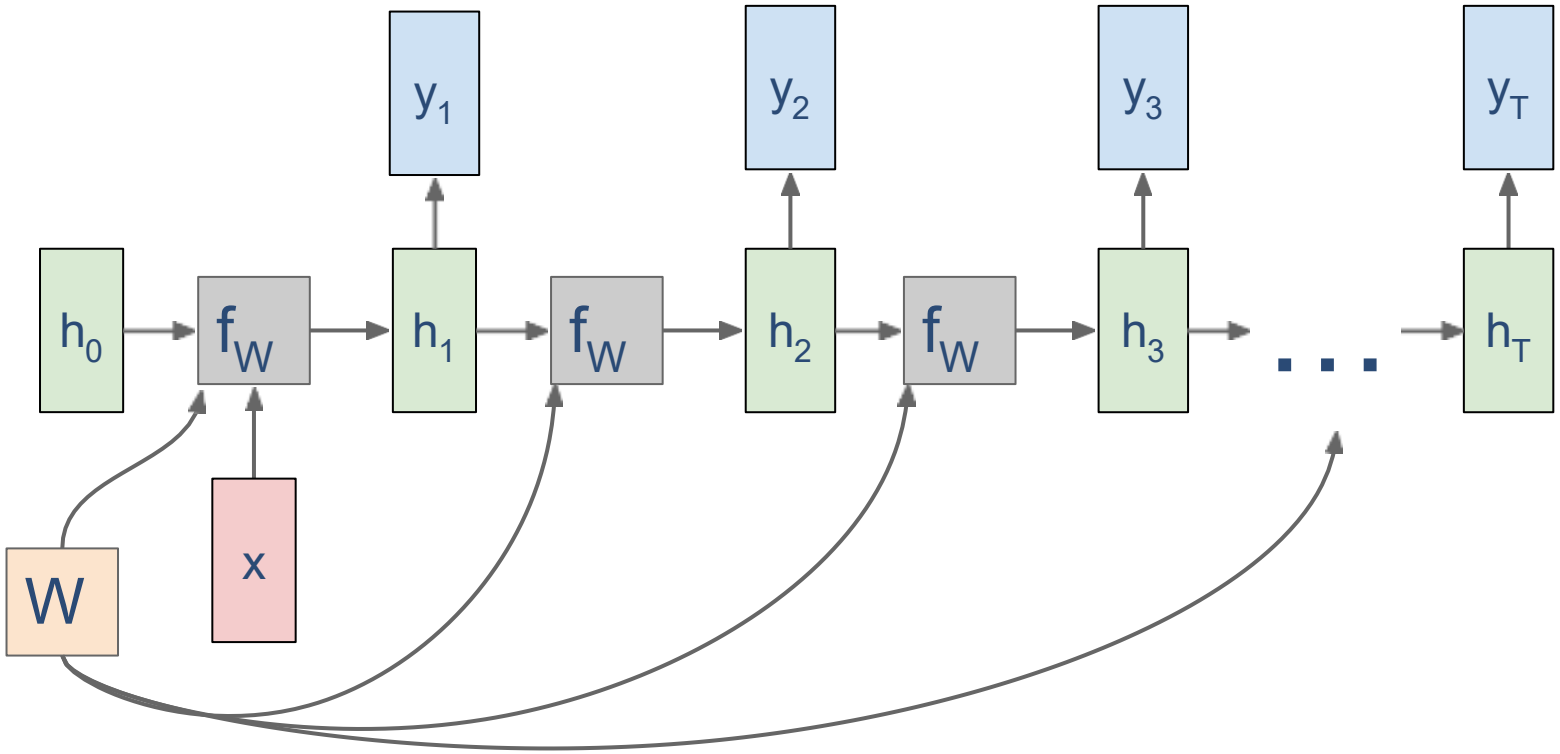
Synchronous sequence-to-sequence

- Information Extraction (IE)
  - Extracting structured information from unstructured texts to form knowledge.

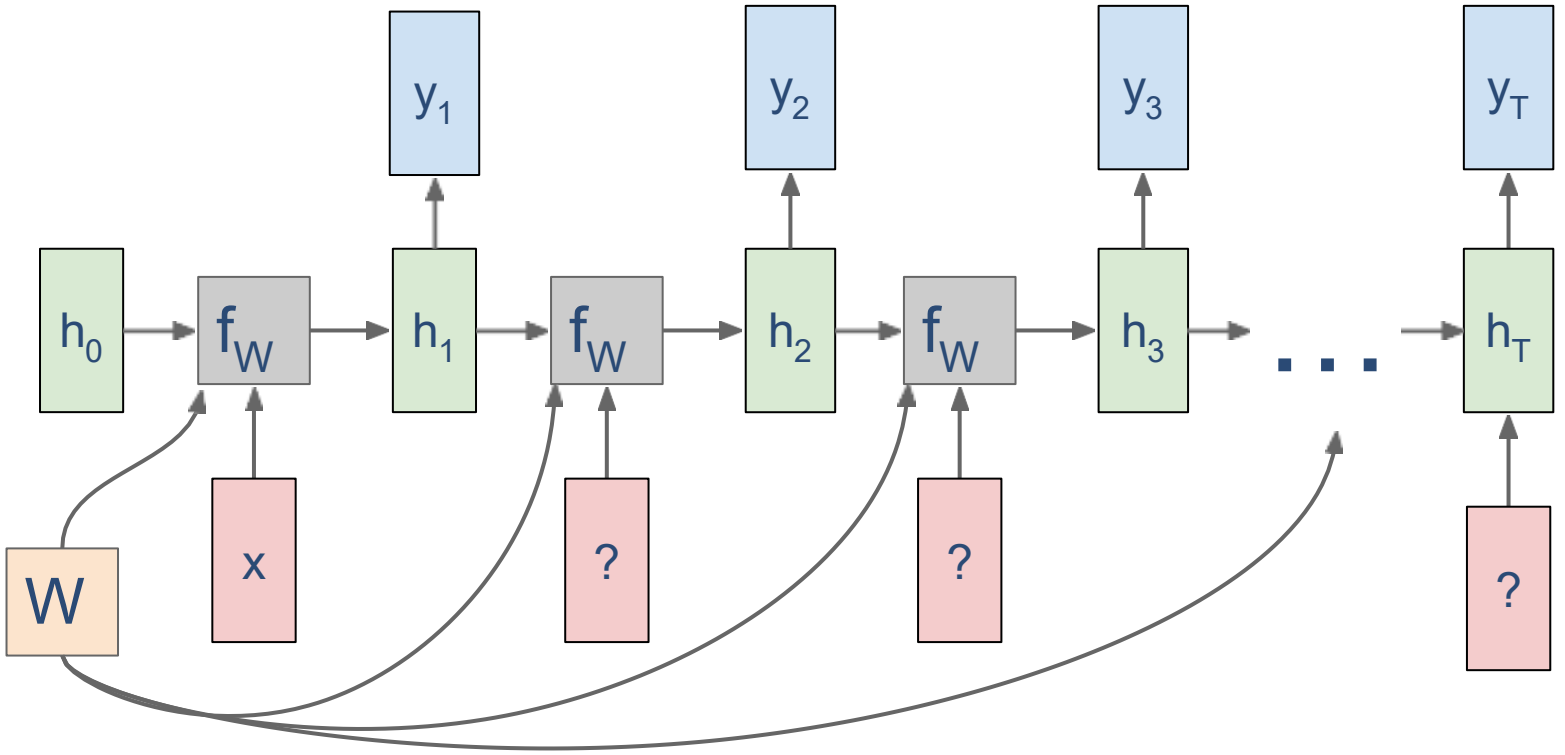
小米创始人雷军表示，该公司2015年营收达到780亿元人民币，较2014年的743亿元人民币增长了5%。



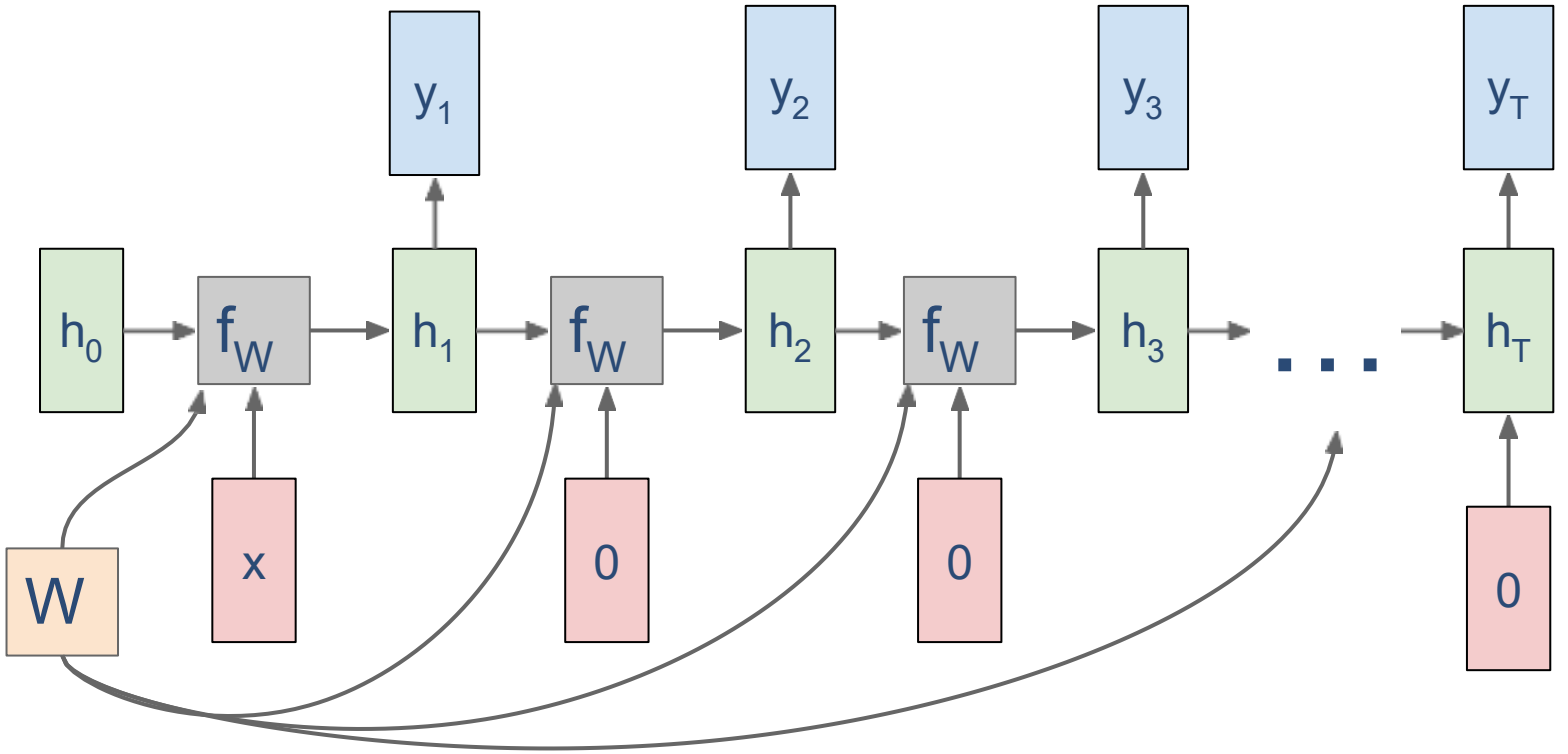
RNN: Computational Graph: One to Many



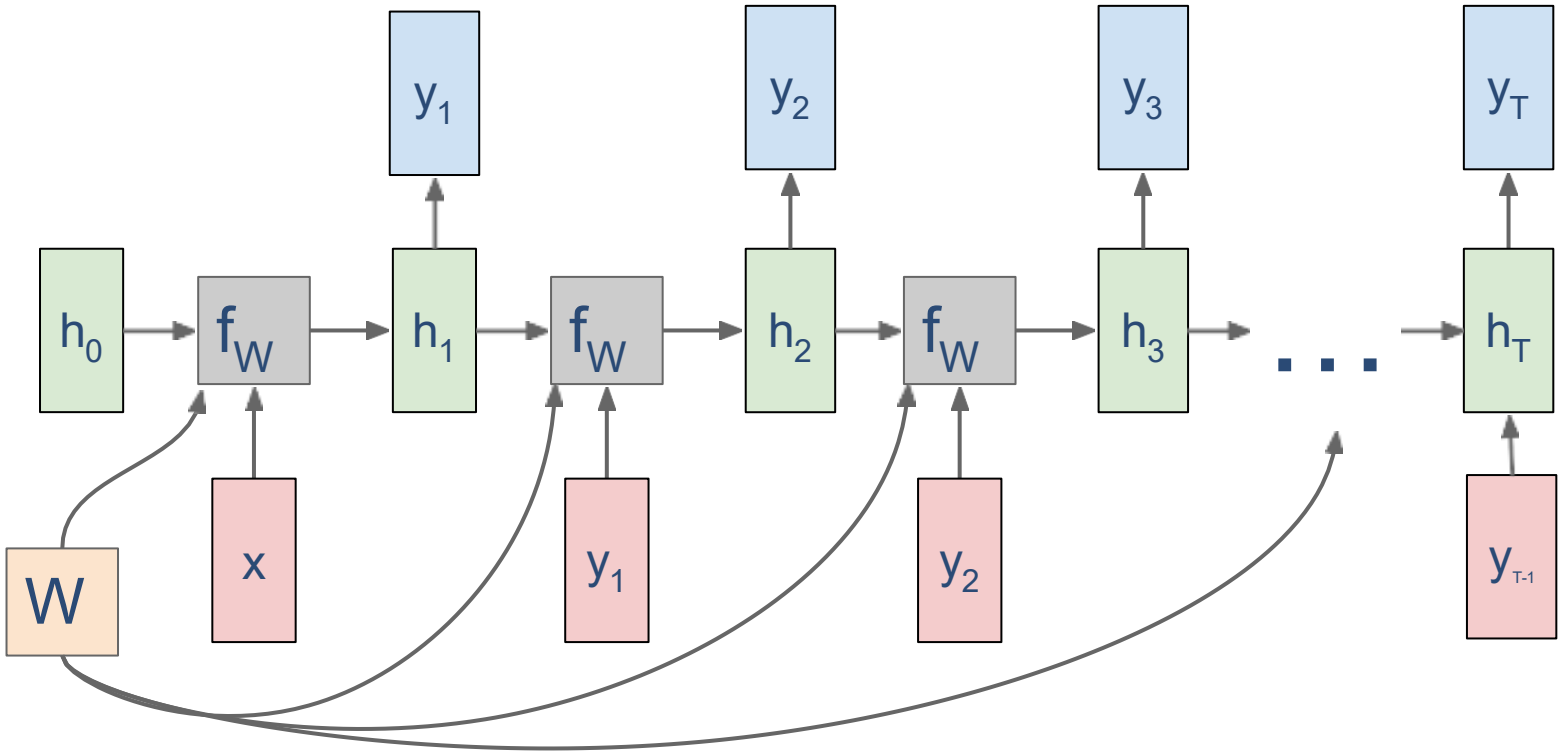
RNN: Computational Graph: One to Many



RNN: Computational Graph: One to Many

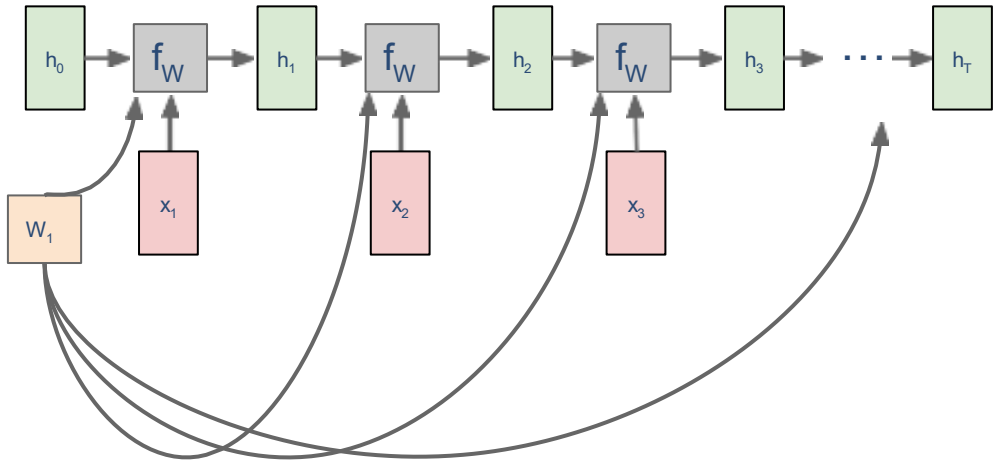


RNN: Computational Graph: One to Many



Sequence to Sequence: Many-to-one + one-to-many

Many to one: Encode input sequence in a single vector

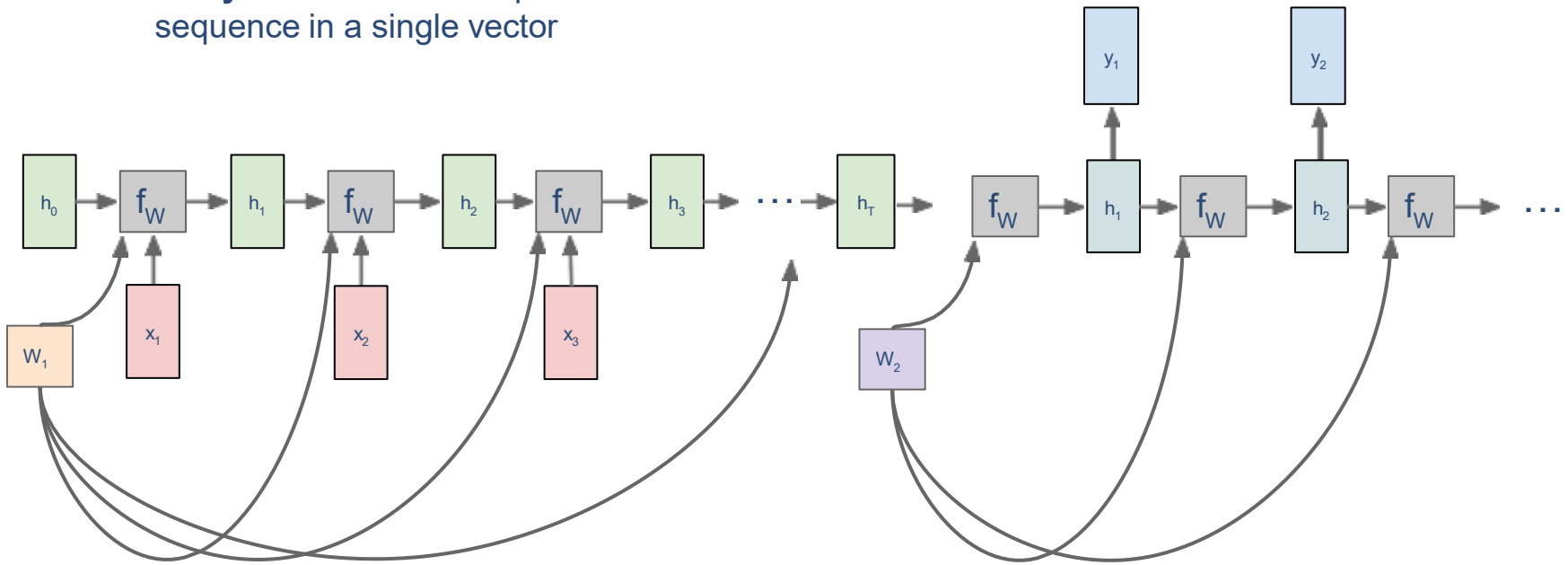


Sutskever et al, “Sequence to Sequence Learning with Neural Networks”, NIPS 2014

Sequence to Sequence: Many-to-one + one-to-many

Many to one: Encode input sequence in a single vector

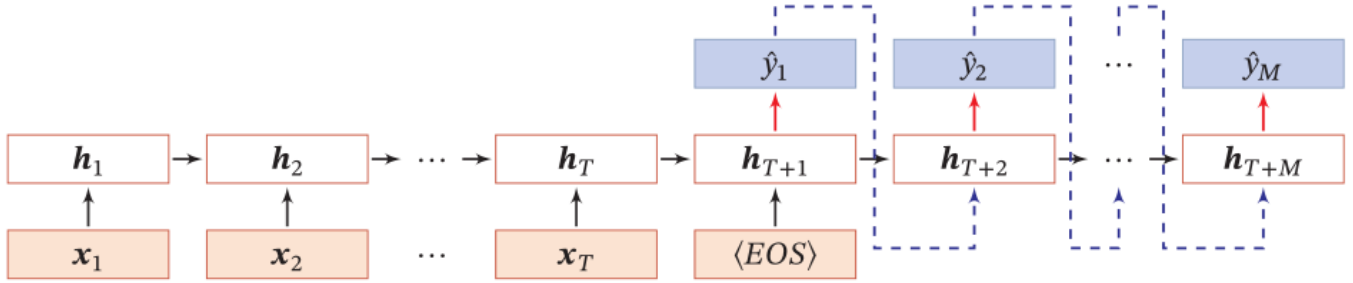
One to many: Produce output sequence from single input vector



Sutskever et al, “Sequence to Sequence Learning with Neural Networks”, NIPS 2014

Applied to Machine Learning

- Asynchronous sequence-to-sequence



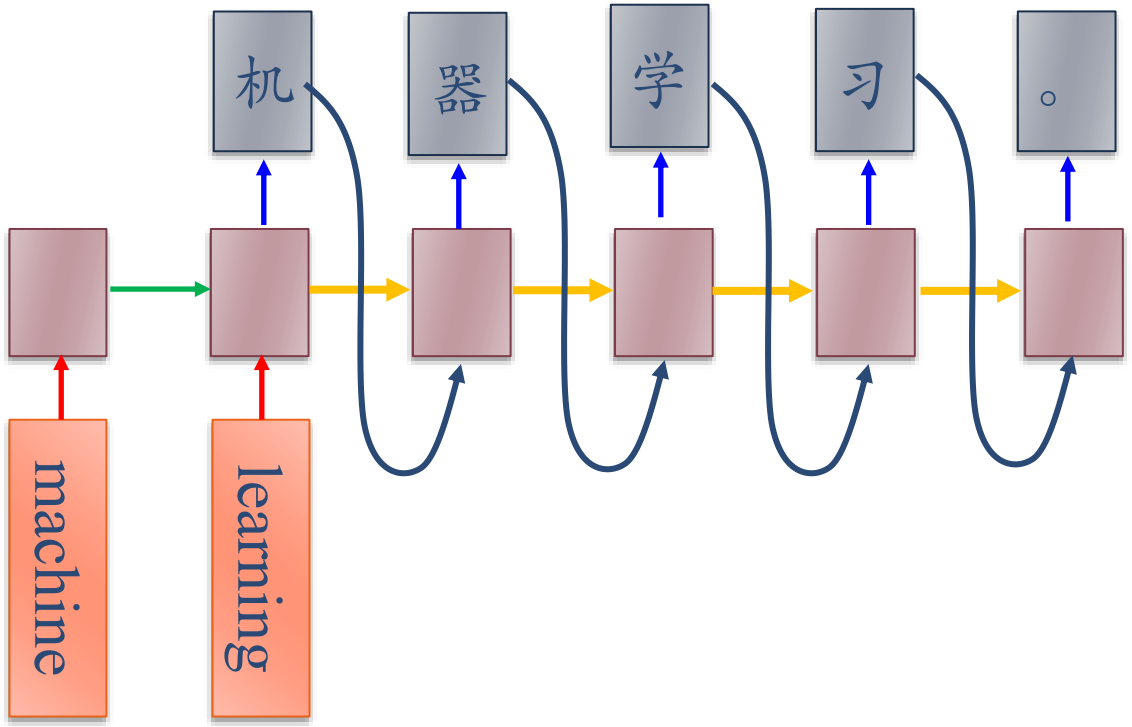
$$\boldsymbol{h}_t = f_1(\boldsymbol{h}_{t-1}, \boldsymbol{x}_t), \qquad \forall t \in [1, T]$$

$$\boldsymbol{h}_{T+t} = f_2(\boldsymbol{h}_{T+t-1}, \hat{\boldsymbol{y}}_{t-1}), \qquad \forall t \in [1, M]$$

$$\hat{y}_t = g(\boldsymbol{h}_{T+t}), \qquad \forall t \in [1, M]$$

Asynchronous sequence-to-sequence

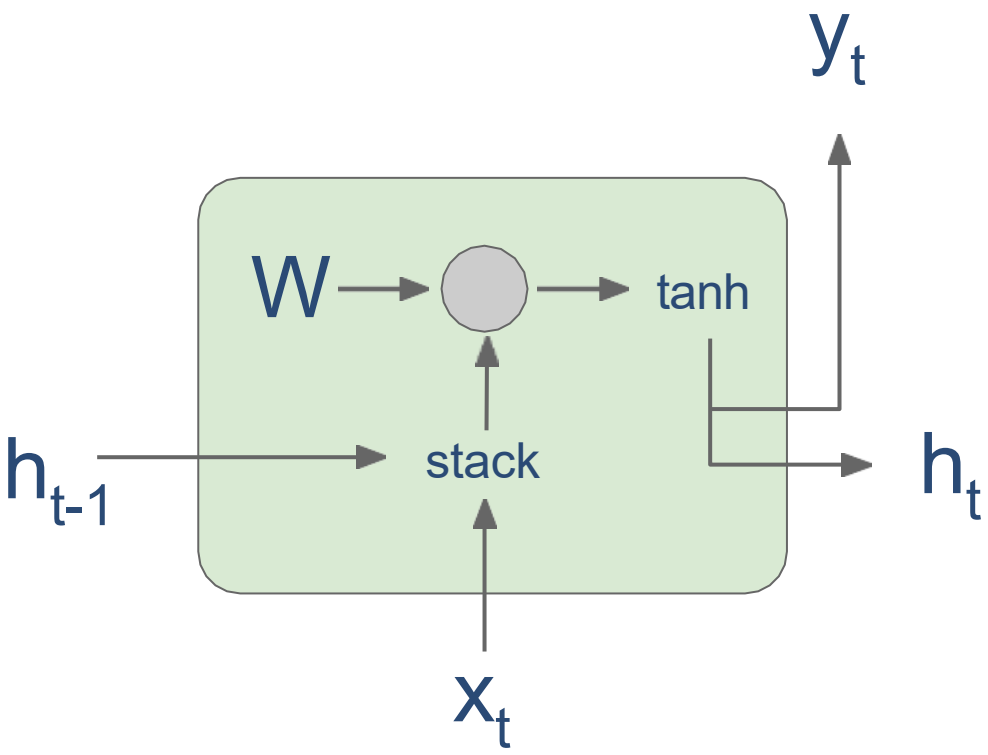
- Machine Translation



5.2.4 Parameter Learning and Long-Term Dependence

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



$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

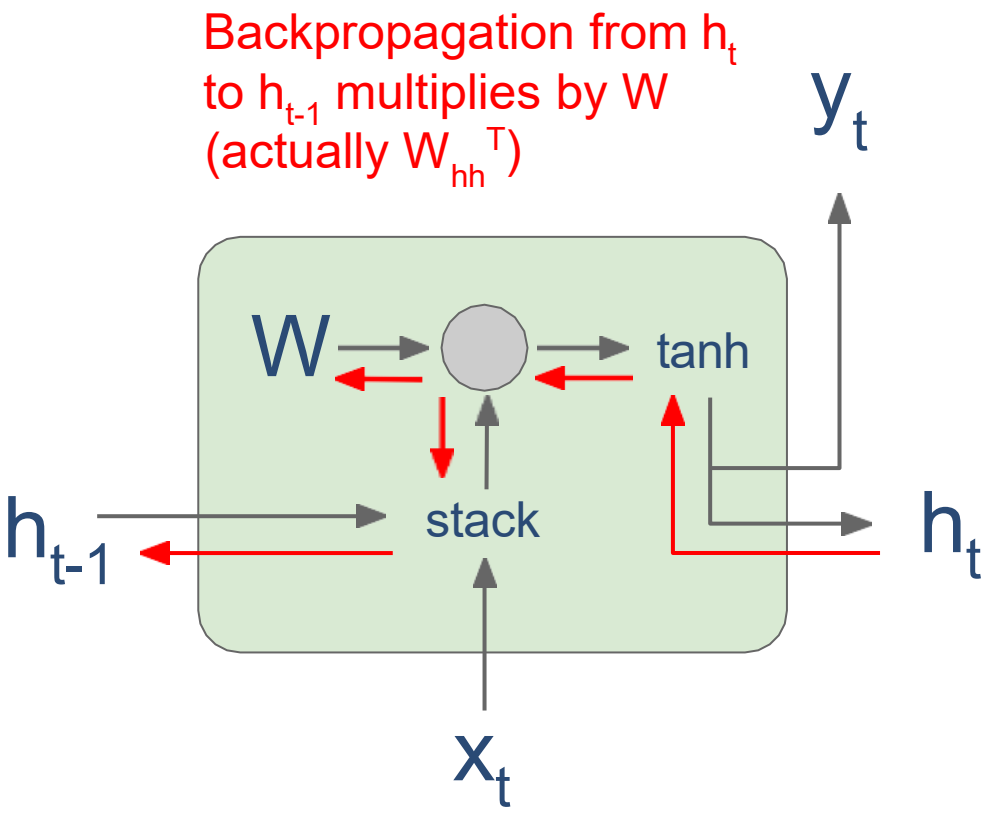
$$= \tanh\left((W_{hh} \quad W_{hx})\begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$

$$= \tanh\left(W\begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$

$$W \in \mathbb{R}^{D\times(M+D)}$$

Vanilla RNN Gradient Flow

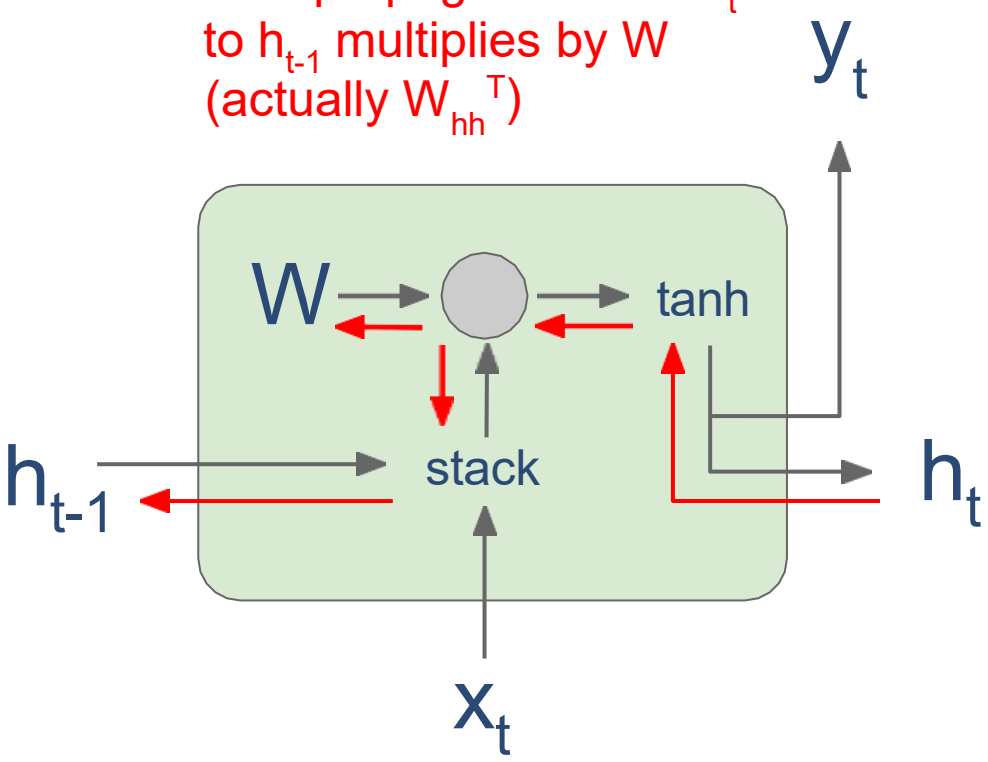
Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013



$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$
$$= \tanh\left((W_{hh} \quad W_{hx})\begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$
$$= \tanh\left(W\begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$

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

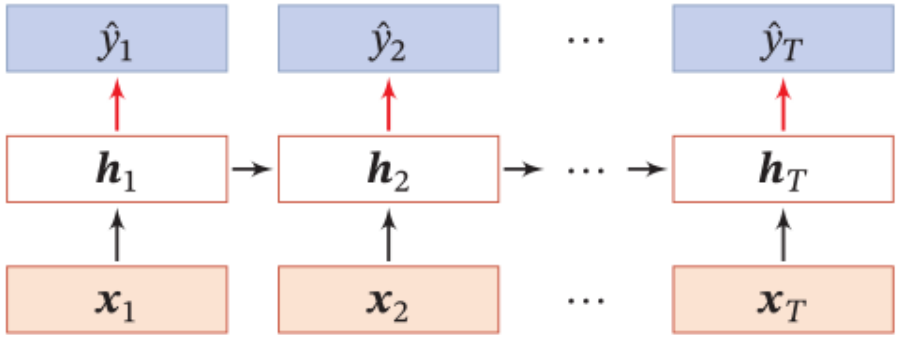


$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$
$$= \tanh\left((W_{hh} \quad W_{hx})\begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$
$$= \tanh\left(W\begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$

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

Parameter Learning

- Machine Learning
  - Given a training sample (x, y)
  - Input series $x = \{x_1, x_2, \cdots, x_T\}$
  - Output series $y = \{y_1, y_2, \cdots, y_T\}$



- Loss function at timestep t

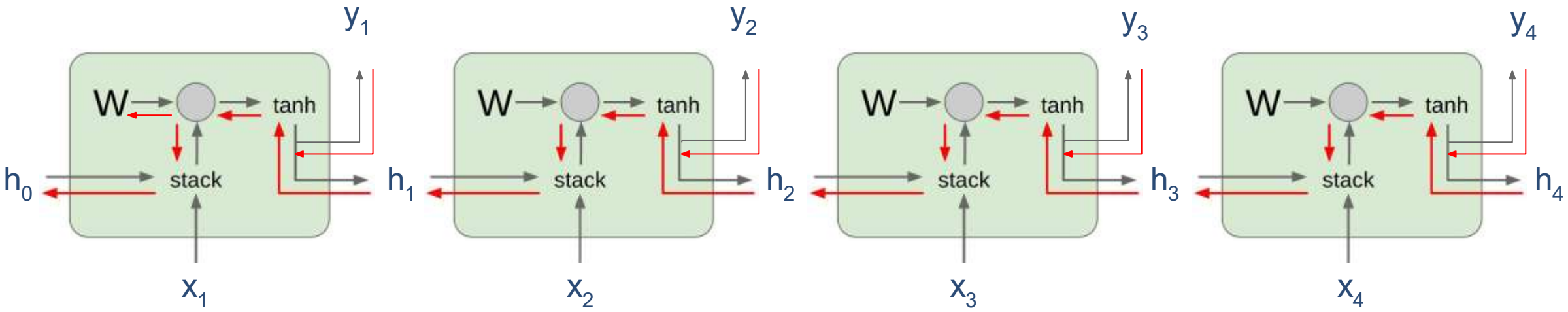
$$\mathcal{L}_t = \mathcal{L}(y_t, g(h_t))$$

- Total Loss

$$\mathcal{L} = \sum_{t=1}^{T} \mathcal{L}_t$$

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

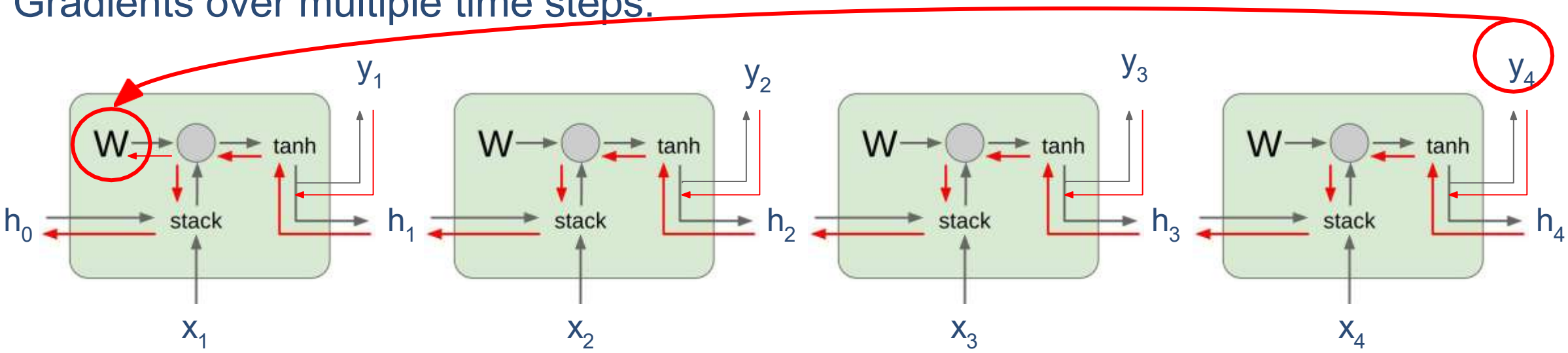


$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}$$

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:



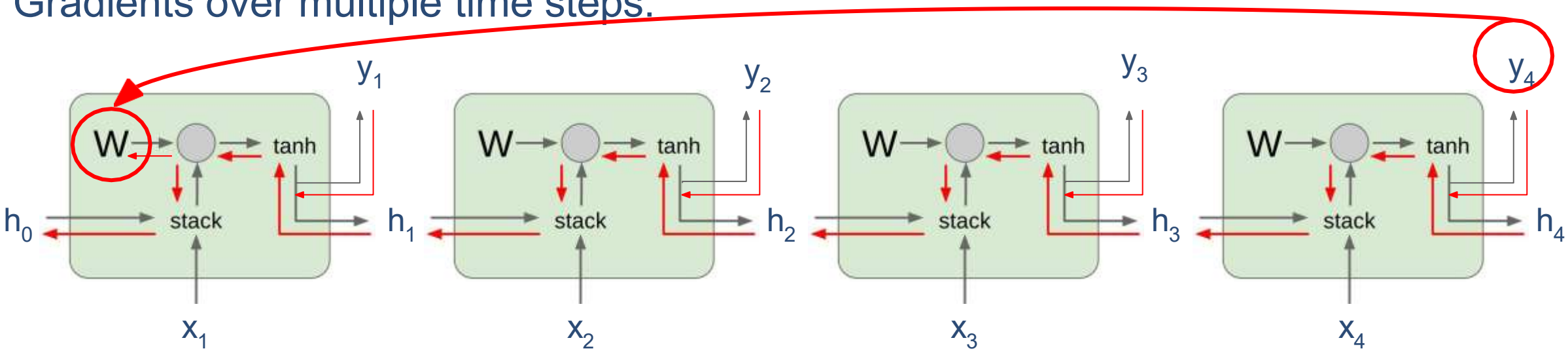
$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}$$

$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} \frac{\partial h_t}{\partial h_{t-1}} \cdots \frac{\partial h_1}{\partial W}$$

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:



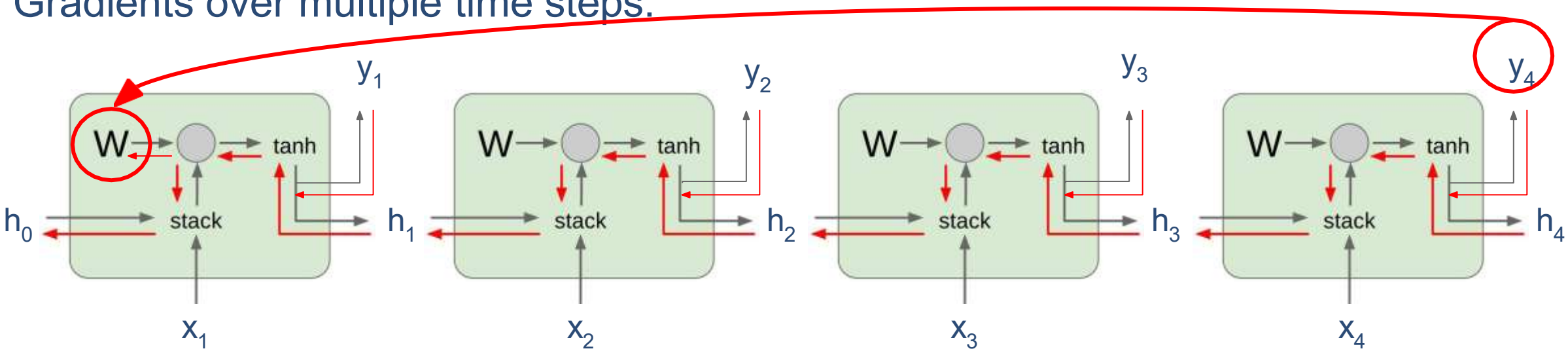
$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}$$

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

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:



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

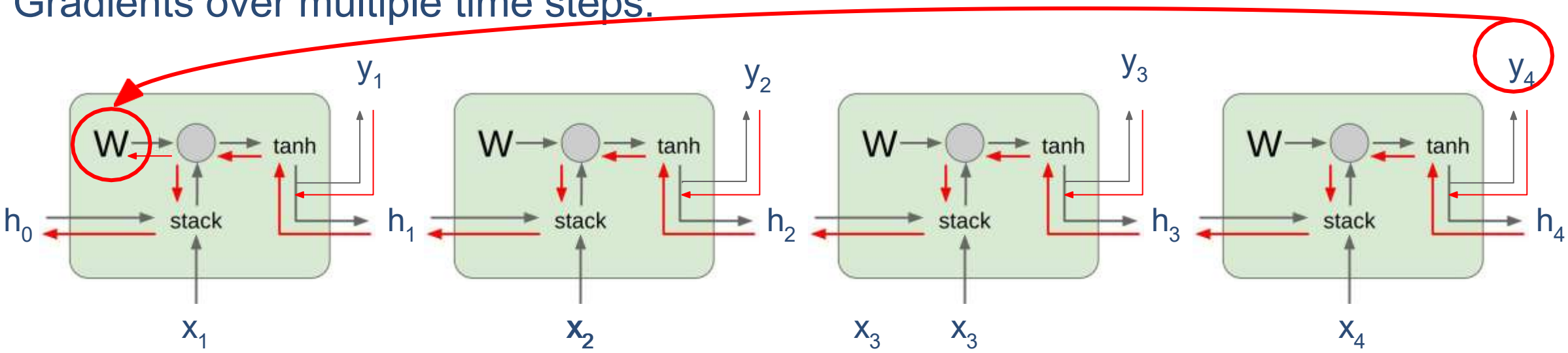
$$\frac{\partial h_t}{\partial h_{t-1}} = tanh'(W_{hh} h_{t-1} + W_{xh} x_t) W_{hh}$$

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

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:



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

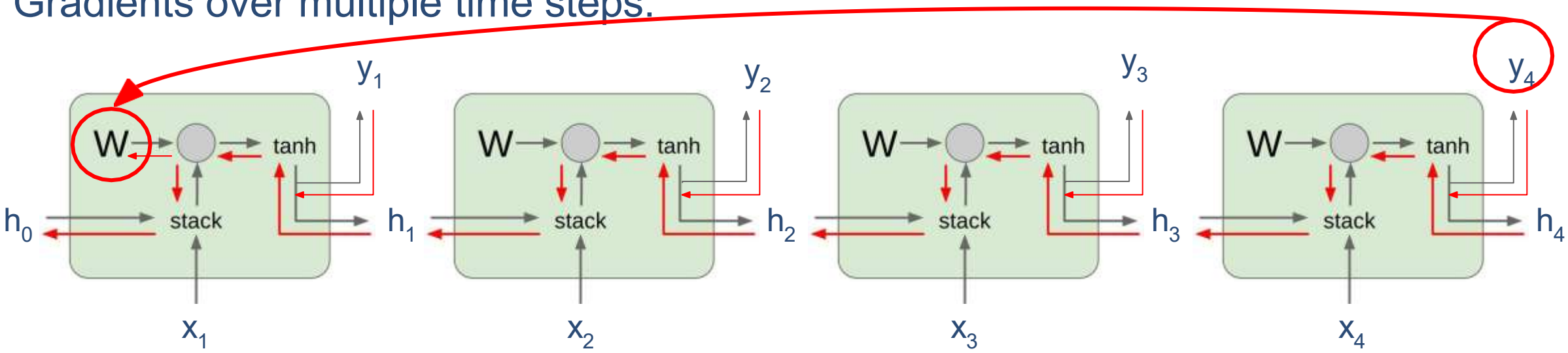
Almost always < 1
Vanishing gradients

$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} \left(\prod_{t=2}^{T} tanh'(W_{hh} h_{t-1} + W_{xh} x_t) \right) W_{hh}^{T-1} \frac{\partial h_1}{\partial W}$$

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:



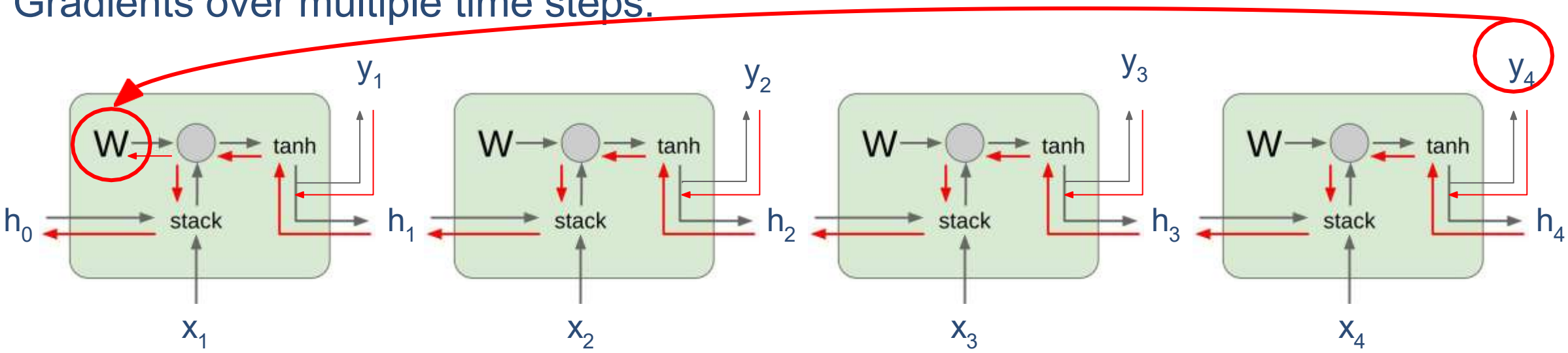
$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}$$

What if we assumed no non-linearity?

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:



What if we assumed no non-linearity?

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

$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} \boxed{W_{hh}^{T-1}} \frac{\partial h_1}{\partial W}$$

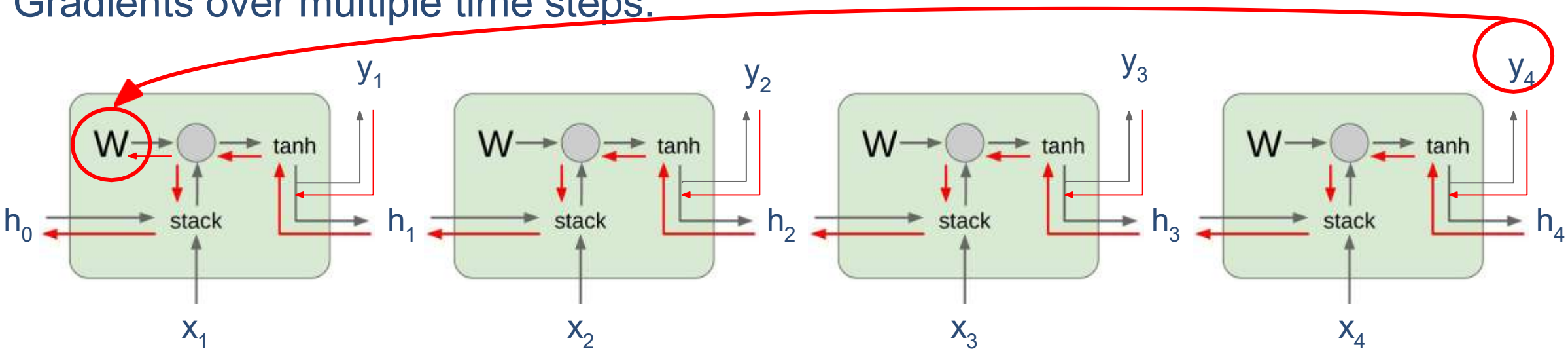
Largest singular value > 1:
Exploding gradients

Largest singular value < 1:
Vanishing gradients

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:



What if we assumed no non-linearity?

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

$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} W_{hh}^{T-1} \frac{\partial h_1}{\partial W}$$

Largest singular value > 1:
Exploding gradients

→ **Gradient clipping**: Scale gradient if its norm is too big

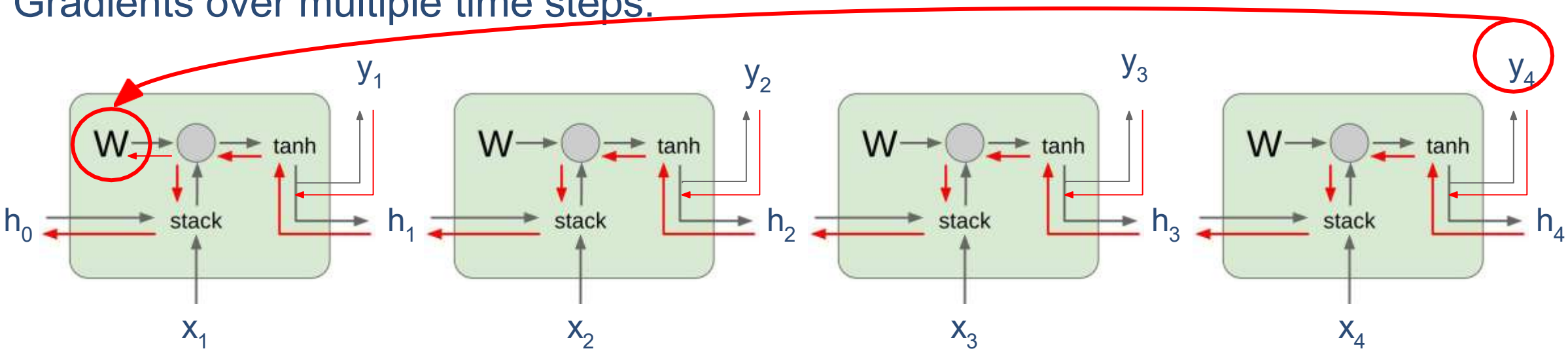
```
grad_norm = np.sum(grad * grad)
if grad_norm > threshold:
  grad *= (threshold / grad_norm)
```

Largest singular value < 1:
Vanishing gradients

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:



What if we assumed no non-linearity?

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

$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} \boxed{W_{hh}^{T-1}} \frac{\partial h_1}{\partial W}$$

Largest singular value > 1:
Exploding gradients

Largest singular value < 1:
Vanishing gradients

→ Change RNN architecture

8.2.5 Resolve Long-Term Dependency

Long-Term Dependency

- RNN is very deep in time dimension!
  - Gradient vanishing or exploding

- How to improve?
- Gradient exploding:
  - Weight decay
  - Gradient truncation
- Gradient vanishing:
  - Improved model

Long-Term Dependency

- Improving methods
  - Recurrent edges to linear dependency?

$$\mathbf{h}_t = \mathbf{h}_{t-1} + g(\mathbf{x}_t; \theta),$$

  - No gradient vanishing/exploding, but representation becomes weaker
  - Add nonlinear as:
  - Gradient exploding? Memory capacity?

$$\mathbf{h}_t = \mathbf{h}_{t-1} + g(\mathbf{x}_t, \mathbf{h}_{t-1}; \theta),$$ Residual Networks?

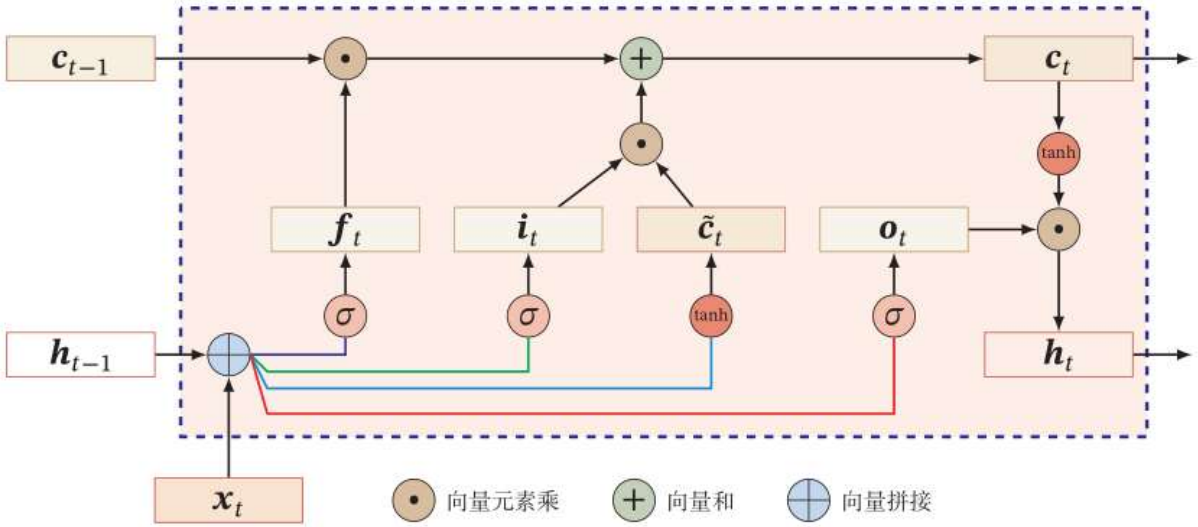
Chapter 5.2.6: LSTM and GRU

Gating Mechanism (门控机制)

- Long-term dependency problem

- Gating mechanism
  - Control the speed of information accumulation, including selectively adding new information and selectively forgetting previously accumulated information.
- Gated RNN
  - Gated Recurrent Unit (GRU)
  - Long Short-Term Memory Network (LSTM)

Long Short-Term Memory (LSTM)



内部状态$c_t \in \mathbb{R}^D$

$$\tilde{\boldsymbol{c}}_t = \tanh(\boldsymbol{W}_c \boldsymbol{x}_t + \boldsymbol{U}_c \boldsymbol{h}_{t-1} + \boldsymbol{b}_c)$$

$$\boldsymbol{c}_t = \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \tilde{\boldsymbol{c}}_t$$

$$\boldsymbol{i}_t = \sigma(\boldsymbol{W}_i \boldsymbol{x}_t + \boldsymbol{U}_i \boldsymbol{h}_{t-1} + \boldsymbol{b}_i)$$

$$\boldsymbol{f}_t = \sigma(\boldsymbol{W}_f \boldsymbol{x}_t + \boldsymbol{U}_f \boldsymbol{h}_{t-1} + \boldsymbol{b}_f)$$

外部状态$h_t \in \mathbb{R}^D$

$$\boldsymbol{h}_t = \boldsymbol{o}_t \odot \tanh(\boldsymbol{c}_t)$$

Long Short Term Memory (LSTM)

Vanilla RNN

$$h_t = \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$

LSTM

Four gates

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

Cell state

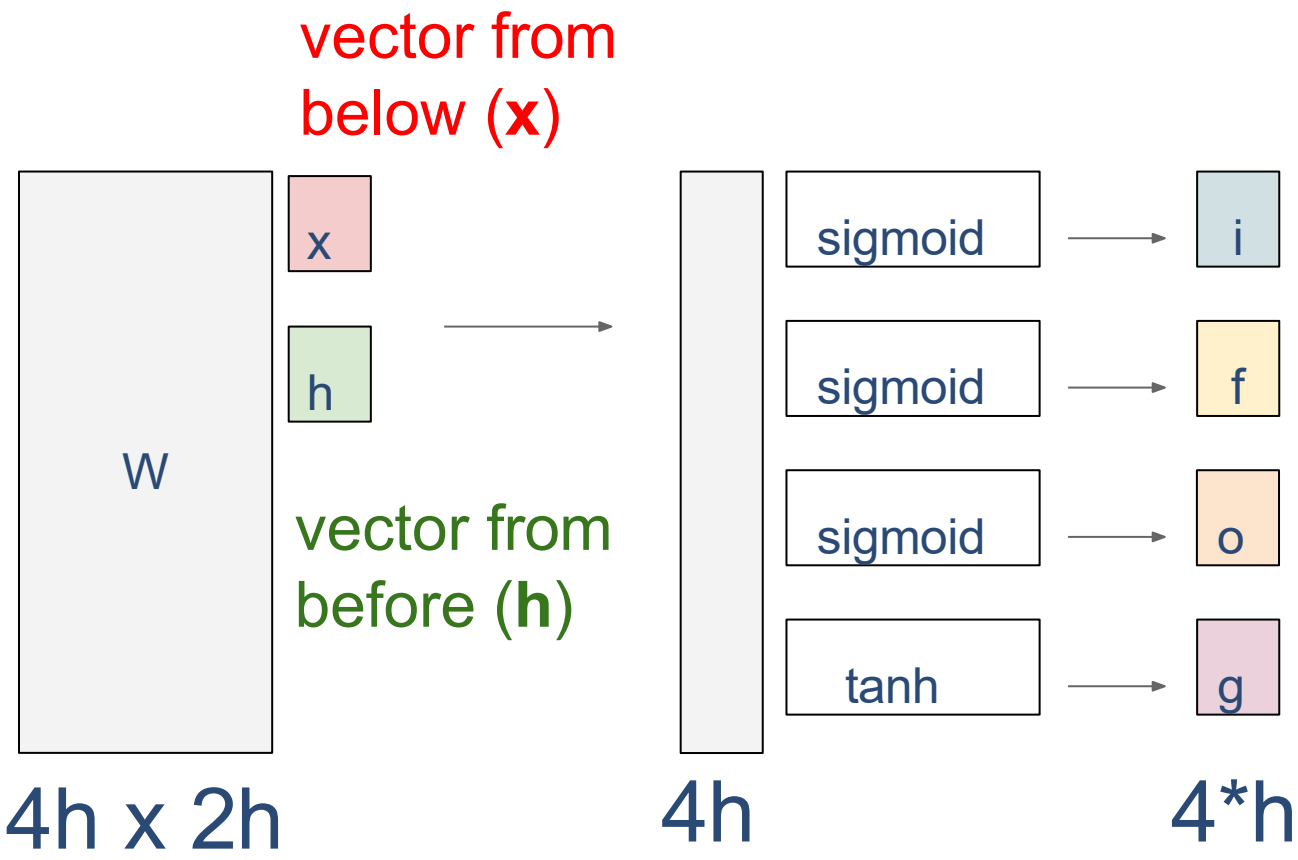
$$c_t = f \odot c_{t-1} + i \odot g$$

Hidden state

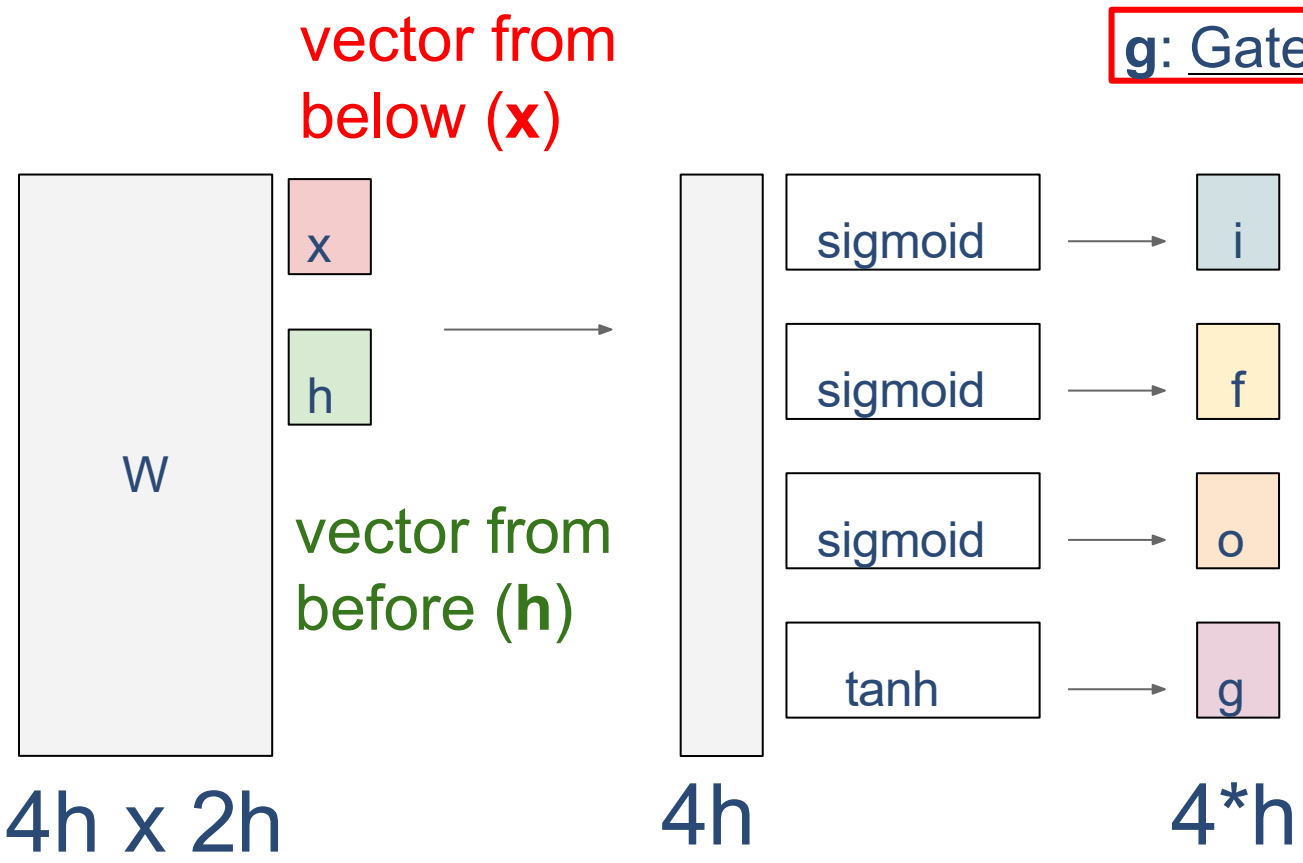
$$h_t = o \odot \tanh(c_t)$$

Hochreiter and Schmidhuber, "Long Short Term Memory", Neural Computation 1997

Long Short Term Memory (LSTM)



Long Short Term Memory (LSTM)



g: Gate gate (?), How much to write to cell

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

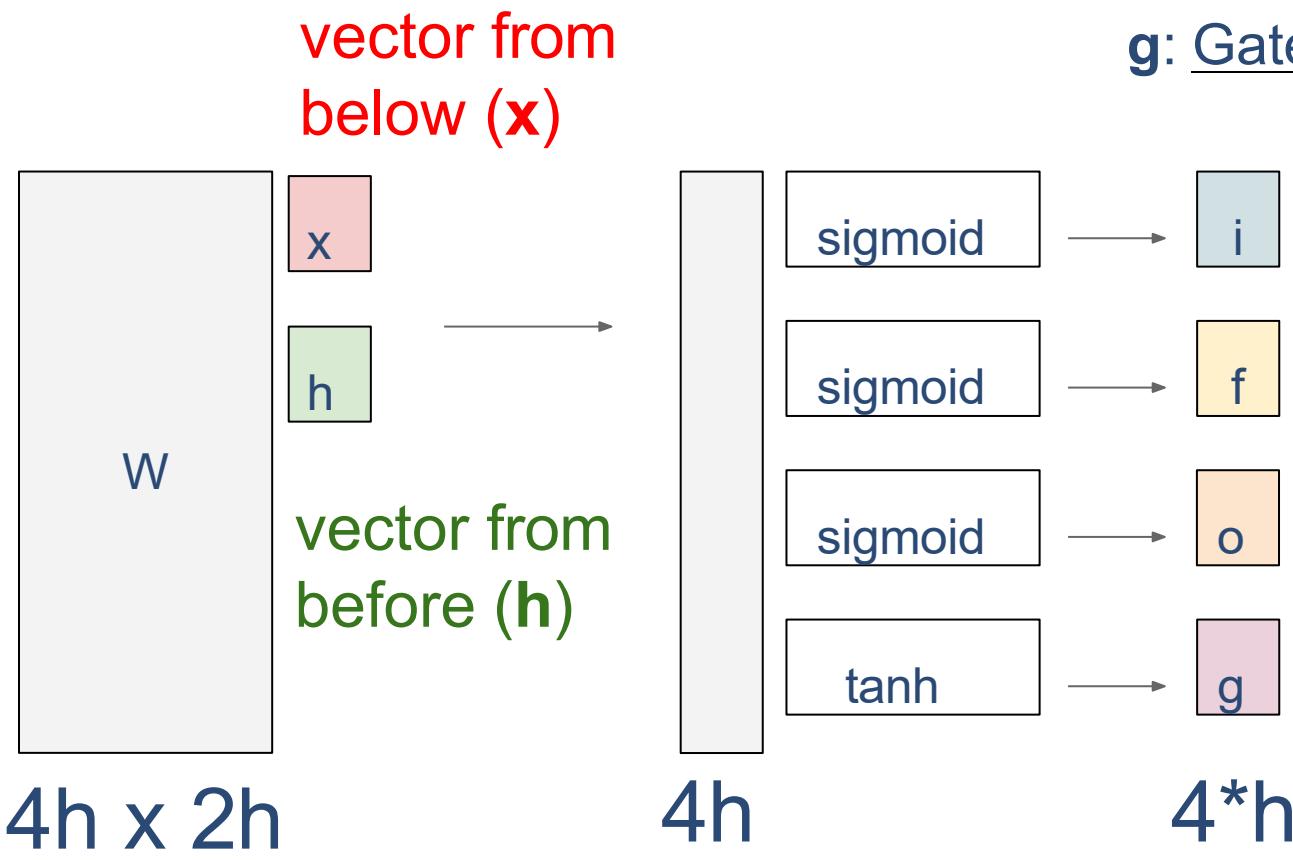
$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

LSTM

i: Input gate, whether to write to cell

g: Gate gate (?), How much to write to cell



$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

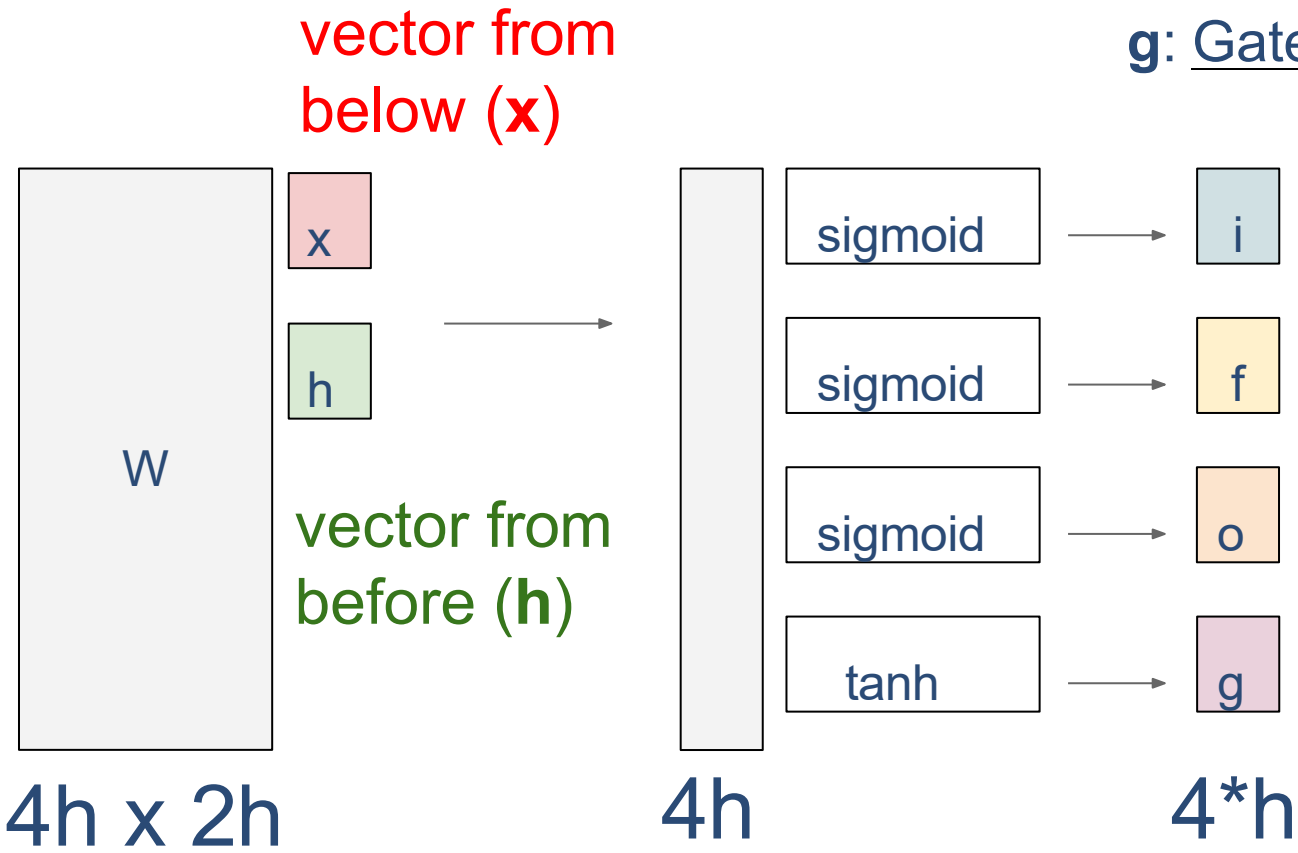
$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

LSTM

i: Input gate, whether to write to cell
f: Forget gate, Whether to erase cell

g: Gate gate (?), How much to write to cell



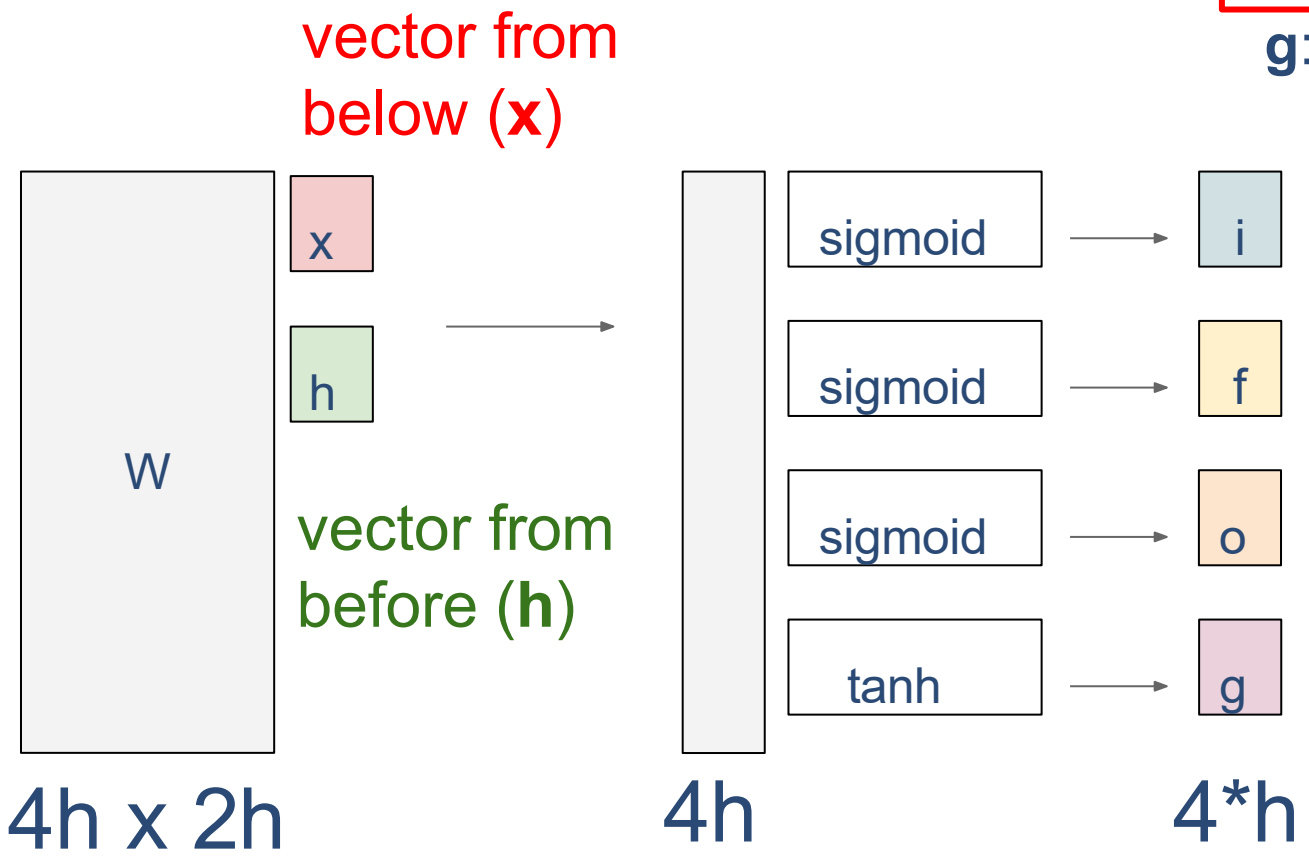
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

LSTM

i: Input gate, whether to write to cell
f: Forget gate, Whether to erase cell
o: Output gate, How much to reveal cell
g: Gate gate (?), How much to write to cell

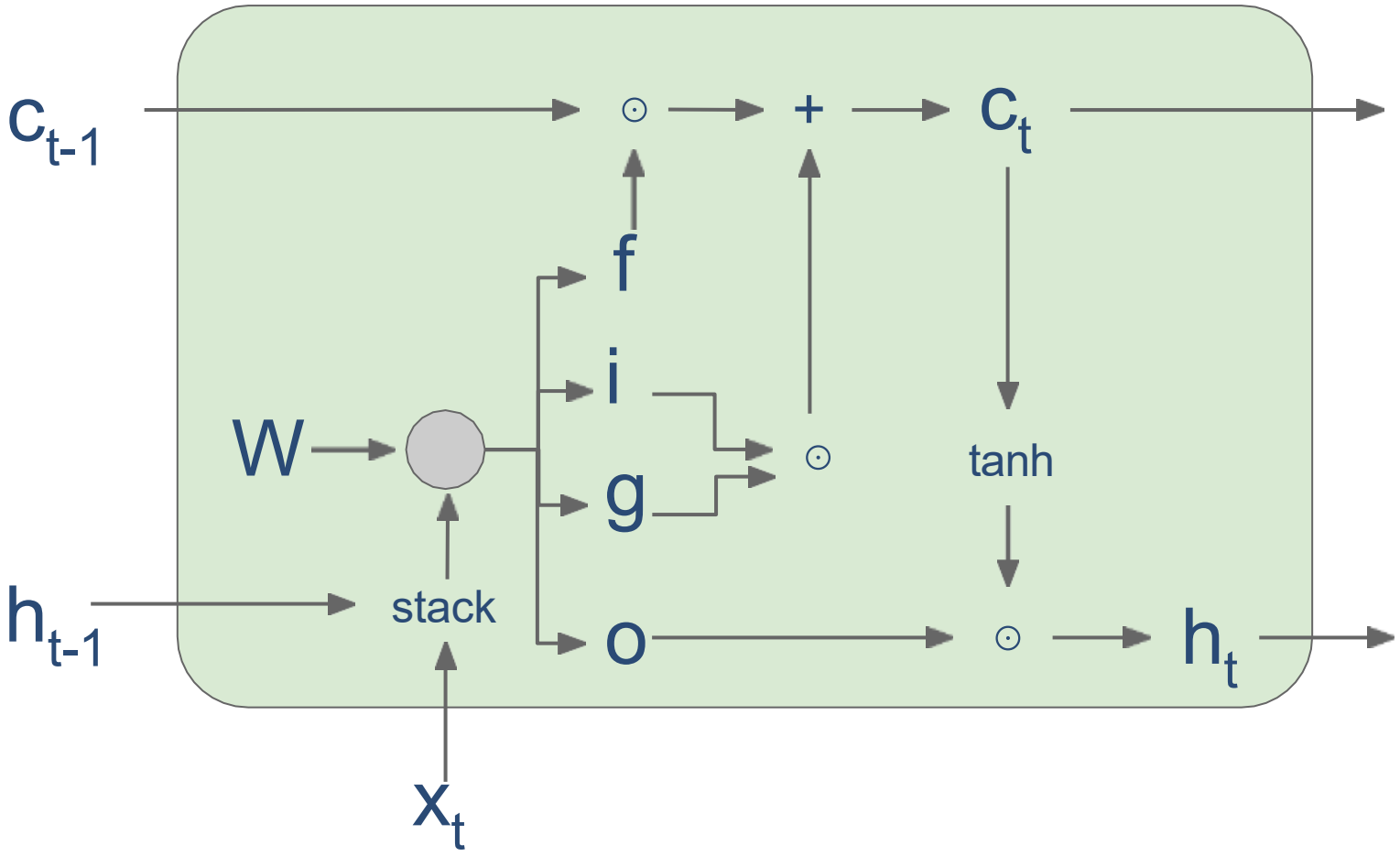


$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

Long Short Term Memory (LSTM)

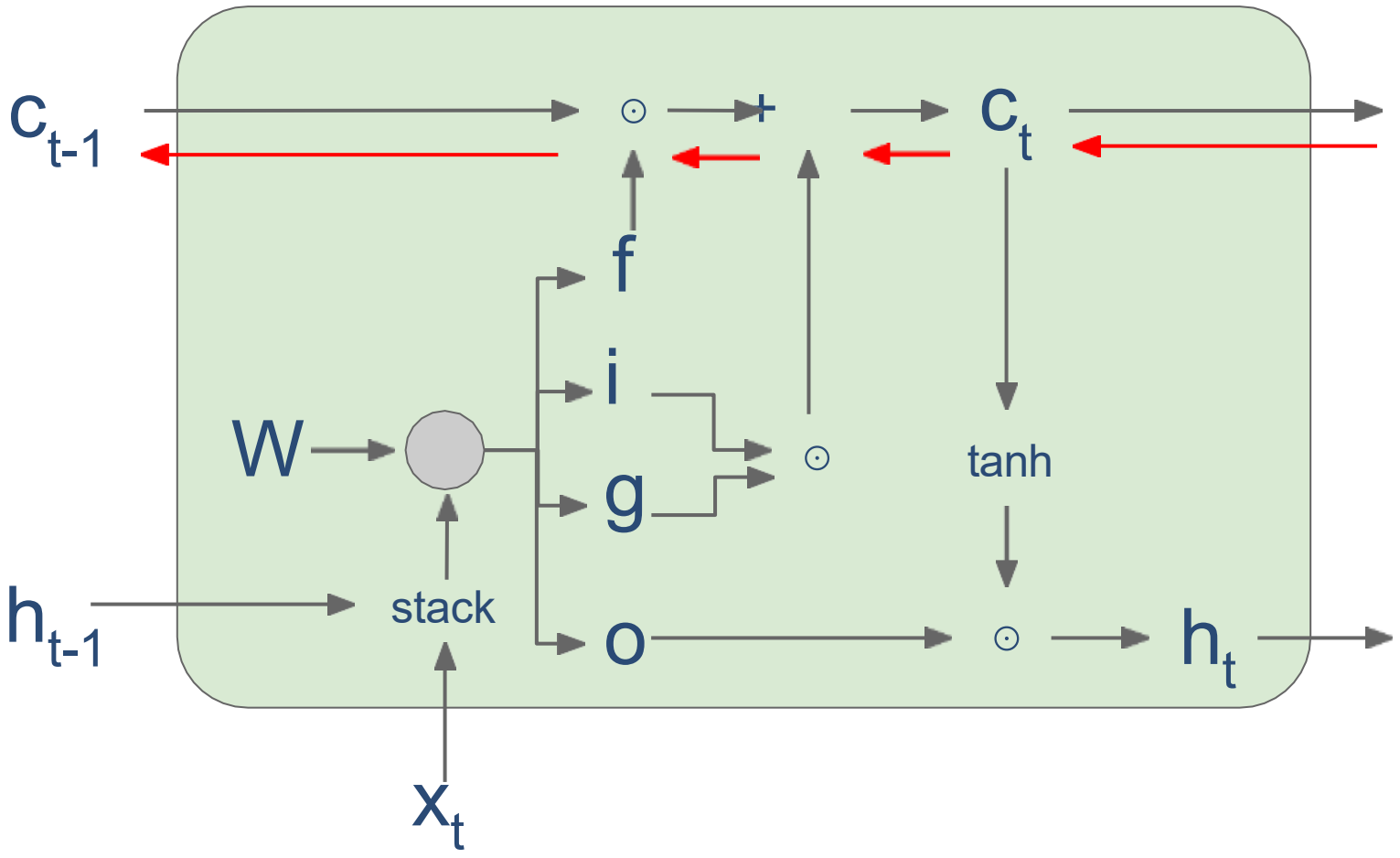


$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

Long Short Term Memory (LSTM): Gradient Flow



Backpropagation from c_t to c_{t-1} only elementwise multiplication by f, no matrix multiply by W

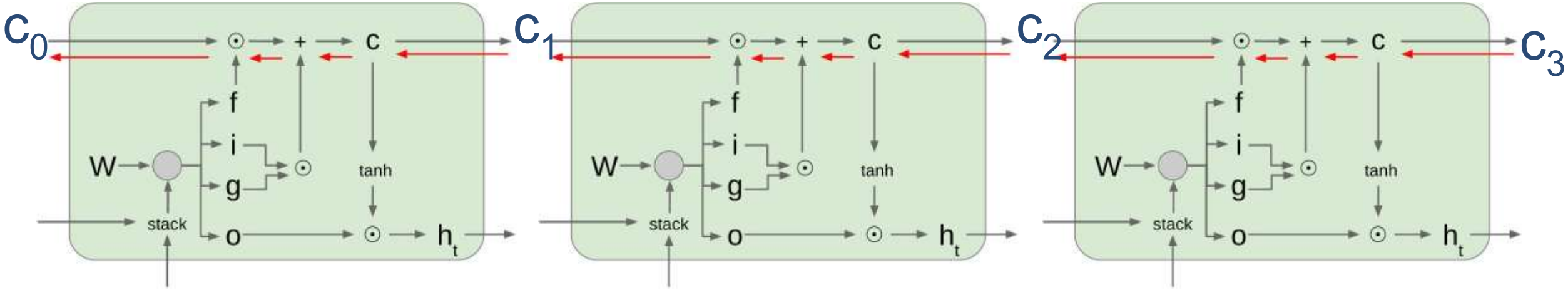
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

Long Short Term Memory (LSTM): Gradient Flow

Uninterrupted gradient flow!



Notice that the gradient contains the **f** gate's vector of activations

- allows better control of gradients values, using suitable parameter updates of the forget gate.

Also notice that are added through the **f, i, g,** and **o** gates

- better balancing of gradient values

Do LSTMs solve the vanishing gradient problem?

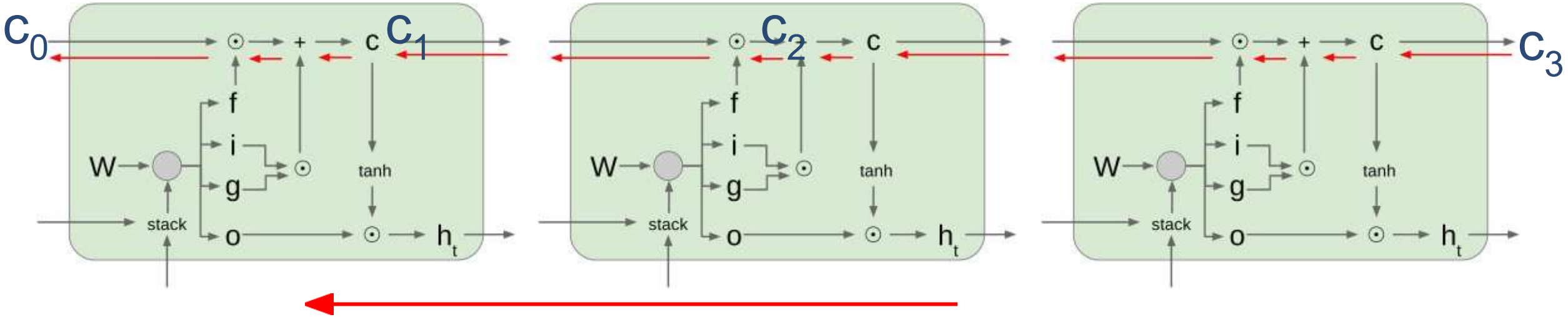
The LSTM architecture makes it easier for the RNN to preserve information over many timesteps

- e.g. **if the f = 1 and the i = 0**, then the information of that cell is preserved indefinitely.
- By contrast, it's harder for vanilla RNN to learn a recurrent weight matrix W_h that preserves info in hidden state

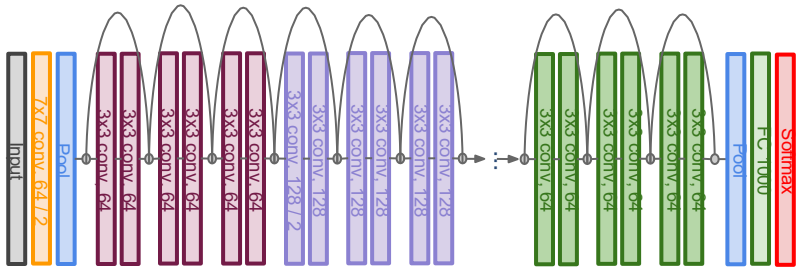
LSTM **doesn't guarantee** that there is no vanishing/exploding gradient, but it does provide an easier way for the model to learn long-distance dependencies

Long Short Term Memory (LSTM): Gradient Flow

Uninterrupted gradient flow!



Similar to ResNet!



In between:

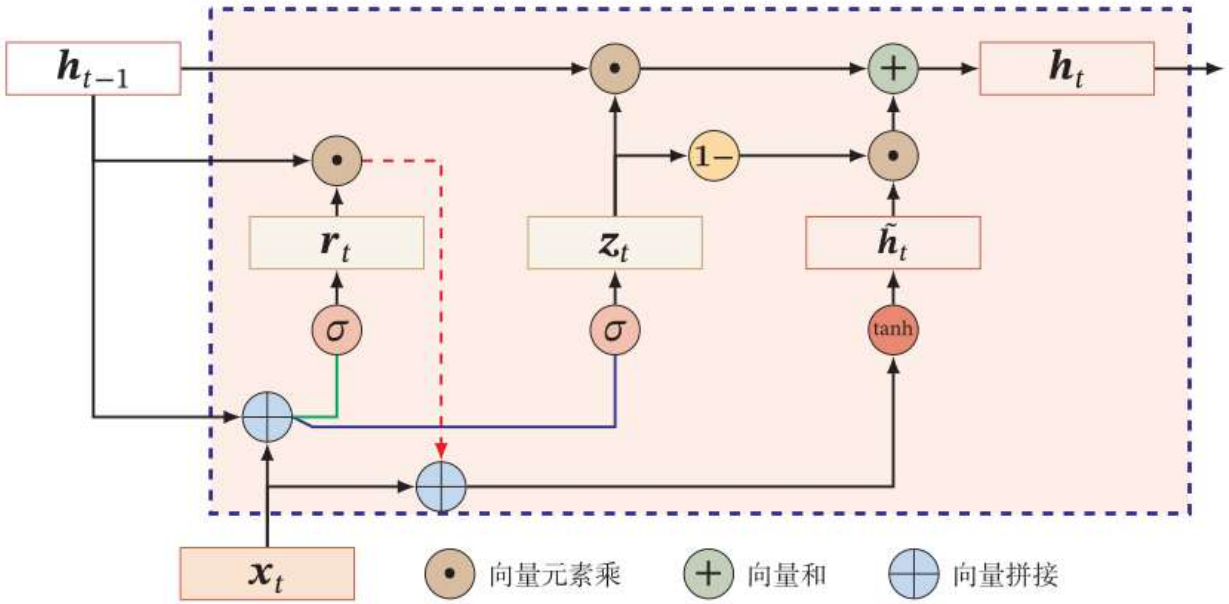
Highway Networks

$$g = T(x, W_T)$$

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

Srivastava et al, "Highway Networks", ICML DL Workshop 2015

Gated Recurrent Unit (GRU)



重置门

$$\mathbf{r}_t = \sigma(\mathbf{W}_r\mathbf{x}_t + \mathbf{U}_r\mathbf{h}_{t-1} + \mathbf{b}_r),$$

$$\mathbf{z}_t = \sigma(\mathbf{W}_z\mathbf{x}_t + \mathbf{U}_z\mathbf{h}_{t-1} + \mathbf{b}_z),$$

更新门

$$\tilde{\mathbf{h}}_t = \tanh(\mathbf{W}_c\mathbf{x}_t + \mathbf{U}(\mathbf{r}_t \odot \mathbf{h}_{t-1}))$$

$$\mathbf{h}_t = \mathbf{z}_t \odot \mathbf{h}_{t-1} + (1 - \mathbf{z}_t) \odot \tilde{\mathbf{h}}_t,$$

Other RNN Variants

GRU [*Learning phrase representations using rnn encoder-decoder for statistical machine translation*, Cho et al. 2014]

$$r_t = \sigma(W_{xr}x_t + W_{hr}h_{t-1} + b_r)$$
$$z_t = \sigma(W_{xz}x_t + W_{hz}h_{t-1} + b_z)$$
$$\tilde{h}_t = \tanh(W_{xh}x_t + W_{hh}(r_t \odot h_{t-1}) + b_h)$$
$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \tilde{h}_t$$

[*LSTM: A Search Space Odyssey*, Greff et al., 2015]

[*An Empirical Exploration of Recurrent Network Architectures,* Jozefowicz et al., 2015]

MUT1:

$$\begin{aligned} z &= \text{sigm}(W_{\text{xz}}x_t + b_{\text{z}}) \\ r &= \text{sigm}(W_{\text{xr}}x_t + W_{\text{hr}}h_t + b_{\text{r}}) \\ h_{t+1} &= \tanh(W_{\text{hh}}(r \odot h_t) + \tanh(x_t) + b_{\text{h}}) \odot z \\ &+ h_t \odot (1 - z) \end{aligned}$$

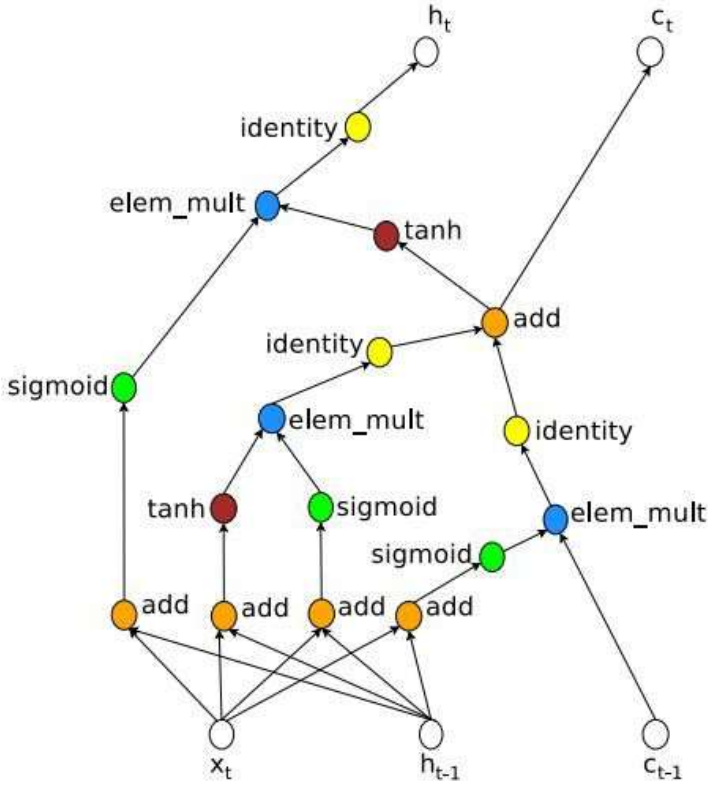
MUT2:

$$\begin{aligned} z &= \text{sigm}(W_{\text{xz}}x_t + W_{\text{hz}}h_t + b_{\text{z}}) \\ r &= \text{sigm}(x_t + W_{\text{hr}}h_t + b_{\text{r}}) \\ h_{t+1} &= \tanh(W_{\text{hh}}(r \odot h_t) + W_{xh}x_t + b_{\text{h}}) \odot z \\ &+ h_t \odot (1 - z) \end{aligned}$$

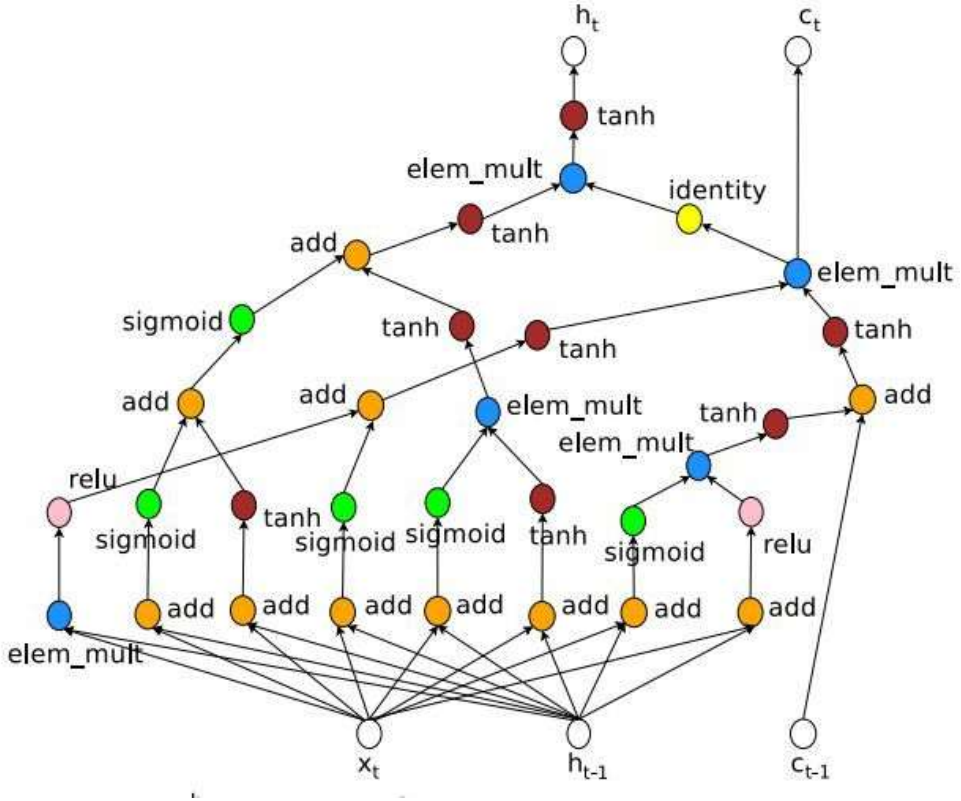
MUT3:

$$\begin{aligned} z &= \text{sigm}(W_{\text{xz}}x_t + W_{\text{hz}}\tanh(h_t) + b_{\text{z}}) \\ r &= \text{sigm}(W_{\text{xr}}x_t + W_{\text{hr}}h_t + b_{\text{r}}) \\ h_{t+1} &= \tanh(W_{\text{hh}}(r \odot h_t) + W_{xh}x_t + b_{\text{h}}) \odot z \\ &+ h_t \odot (1 - z) \end{aligned}$$

Neural Architecture Search for RNN architectures



LSTM cell



Cell they found

Zoph et Le, “Neural Architecture Search with Reinforcement Learning”, ICLR 2017

RNN tradeoffs

RNN Advantages:

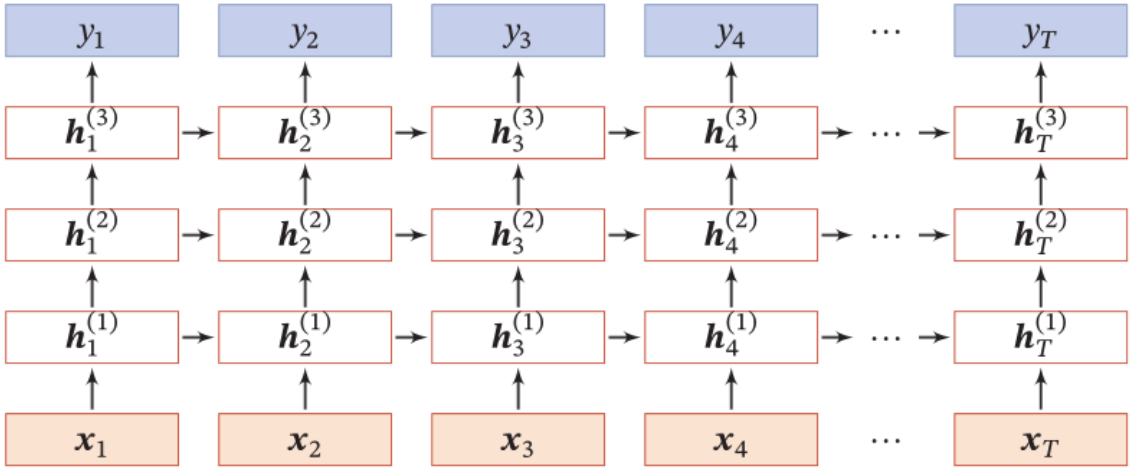
- Can process any length input
- Computation for step *t* can (in theory) use information from many steps back
- Model size doesn't increase for longer input
- Same weights applied on every timestep, so there is symmetry in how inputs are processed.

RNN Disadvantages:

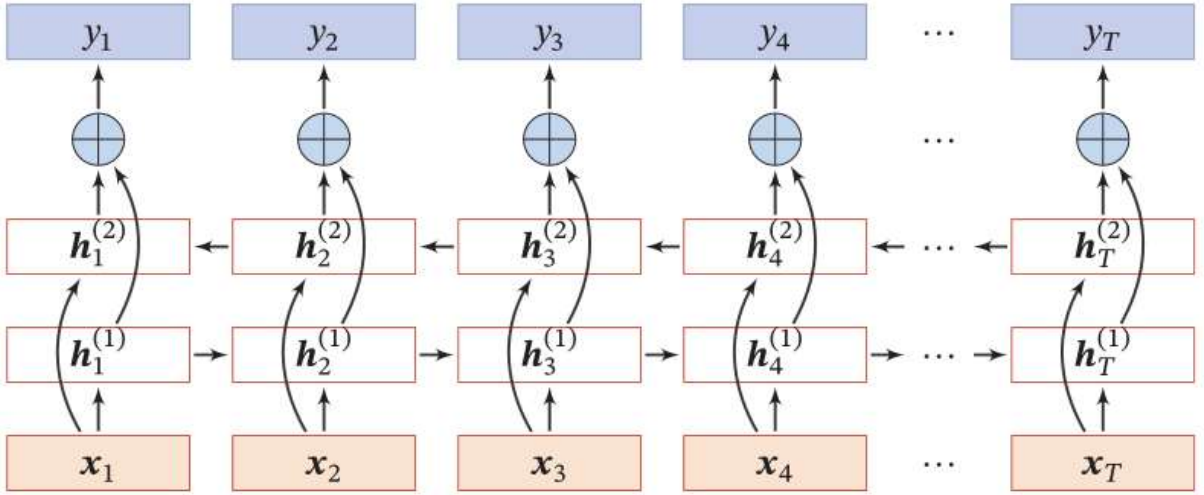
- Recurrent computation is slow
- In practice, difficult to access information from many steps back

5.2.7: Deep Recurrent Neural Networks

Stacked RNN (SRNN)



Bidirectional RNN (Bi-RNN)



5.2.8 Recurrent Network Application

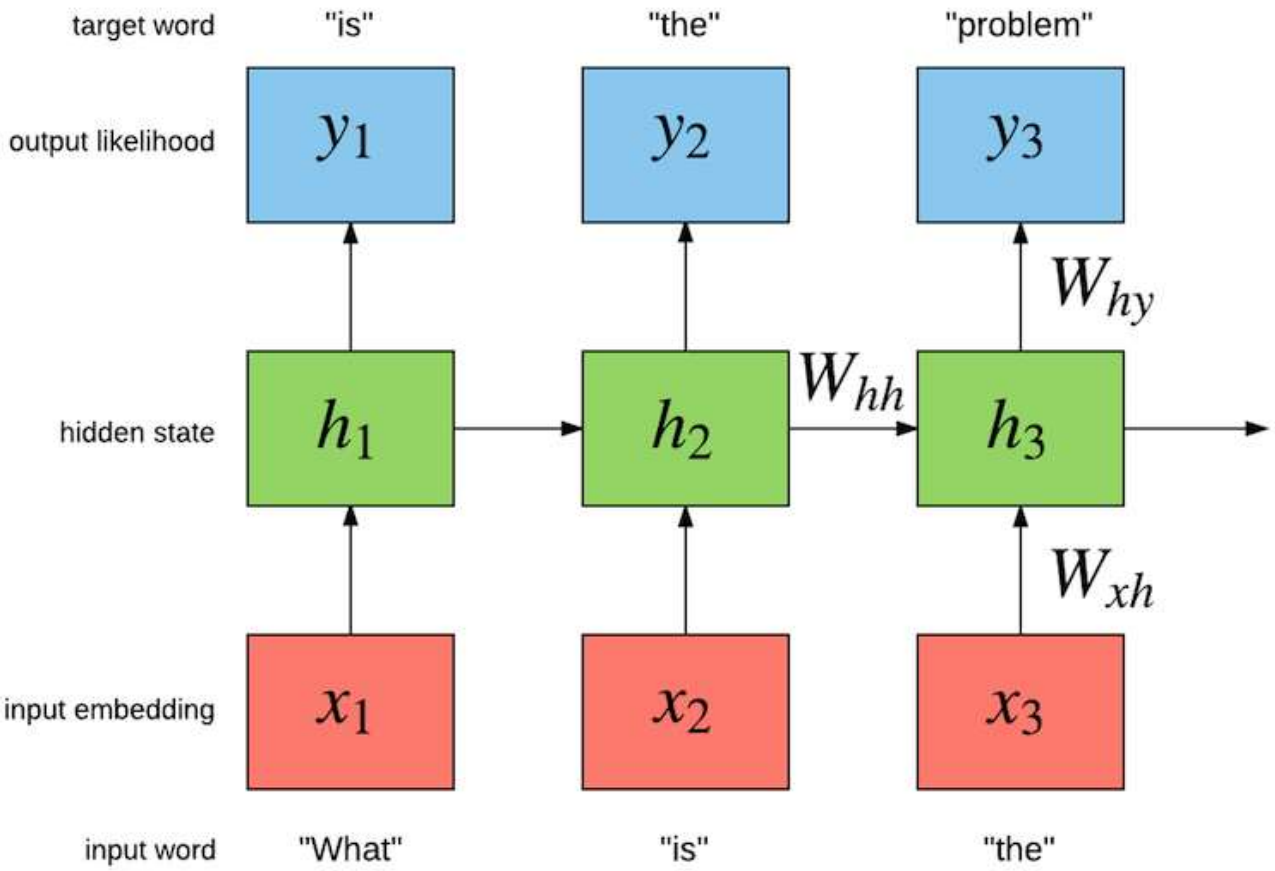
Language Model

- Language understanding → Possibility/rationality of a sentence
  - ！ 在报那猫告做只 😖
  - 那只猫在作报告！ 😓
  - 那个人在作报告！ 😀

- Everything is a probability!
  - $P(x_1, x_2, \cdots, x_T)$
  - $= \prod_i P(x_i | x_{i-1}, \cdots, x_1)$
  - $\approx \prod_i P(x_i | x_{i-1}, \cdots, x_{i-n+1})$

N-gram language model

Language Model



Generate LINUX Kernel Code

```
/*
 * If this error is set, we will need anything right after that BSD.
 */
static void action_new_function(struct s_stat_info *wb)
{
  unsigned long flags;
  int lel_idx_bit = e->edd, *sys & ~((unsigned long) *FIRST_COMPAT);
  buf[0] = 0xFFFFFFFF & (bit << 4);
  min(inc, slist->bytes);
  printk(KERN_WARNING "Memory allocated %02x/%02x, "
    "original MLL instead\n"),
    min(min(multi_run - s->len, max) * num_data_in),
    frame_pos, sz + first_seg);
  div_u64_w(val, inb_p);
  spin_unlock(&disk->queue_lock);
  mutex_unlock(&s->sock->mutex);
  mutex_unlock(&func->mutex);
  return disassemble(info->pending_bh);
}

static void num_serial_settings(struct tty_struct *tty)
{
  if (tty == tty)
    disable_single_st_p(dev);
  pci_disable_spool(port);
```

Word-Writing Machine

- The lyrics automatically generated by RNN after “learning” all 汪峰's works.
- https://github.com/phunterlau/wangfeng-rnn

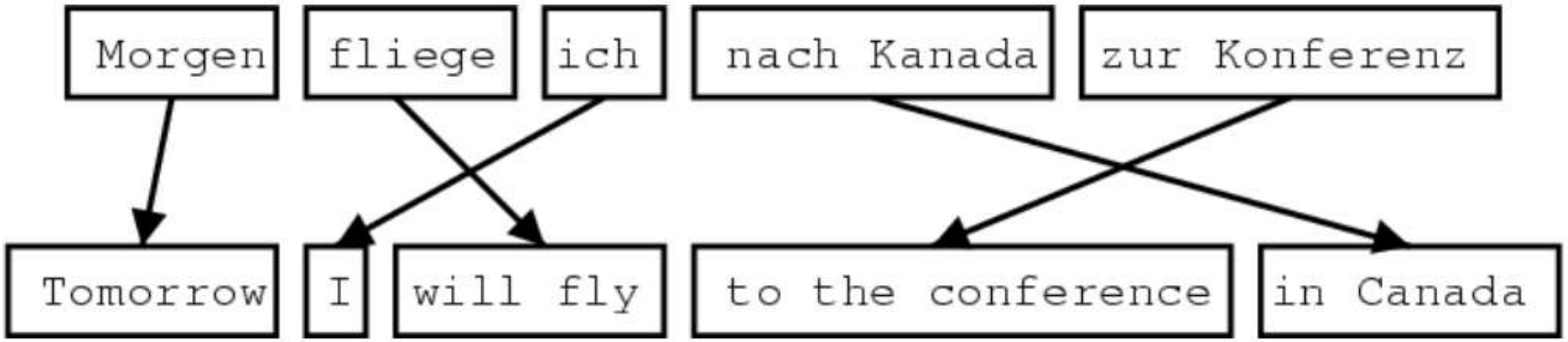
我在这里中的夜里
就像一场是一种生命的意叶
就像我的生活变得在我一样
可我们这是一个知道
我只是一天你会怎吗
可我们这是我们的是不要为你
我们想这有一种生活的时候

Poetizing

白鹭窥鱼立,	满怀风月一枝春,
Egrets stood, peeping fishes.	Budding branches are full of romance.
青山照水开.	未见梅花亦可人.
Water was still, reflecting mountains.	Plum blossoms are invisible but adorable.
夜来风不动,	不为东风无此客,
The wind went down by nightfall,	With the east wind comes Spring.
明月见楼台.	世间何处是前身.
as the moon came up by the tower.	Where on earth do I come from?

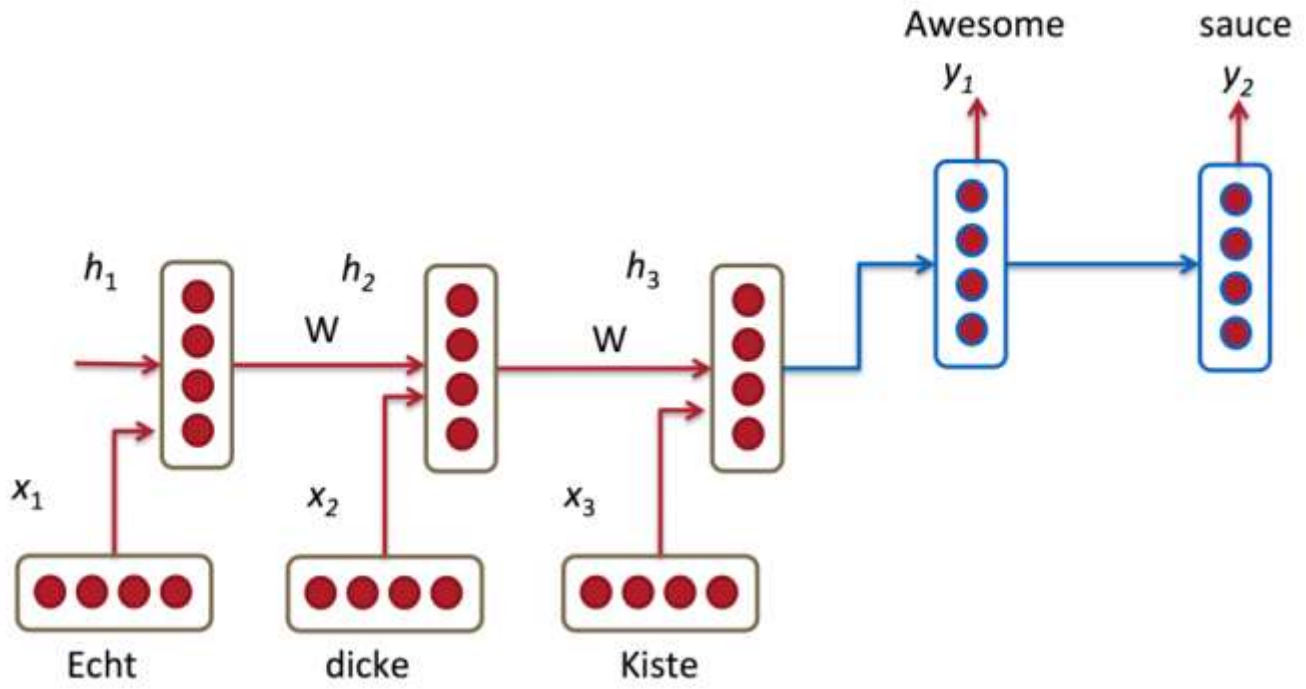
Traditional Statistical Machine Translation

- Source language： f
- Target language： e
  - Model： $\hat{e} = \text{argmax}_e\, p(e|f) = \text{argmax}_e\, p(f|e)p(e)$
  - p(f|e): Translation model
  - p(e) : Language model

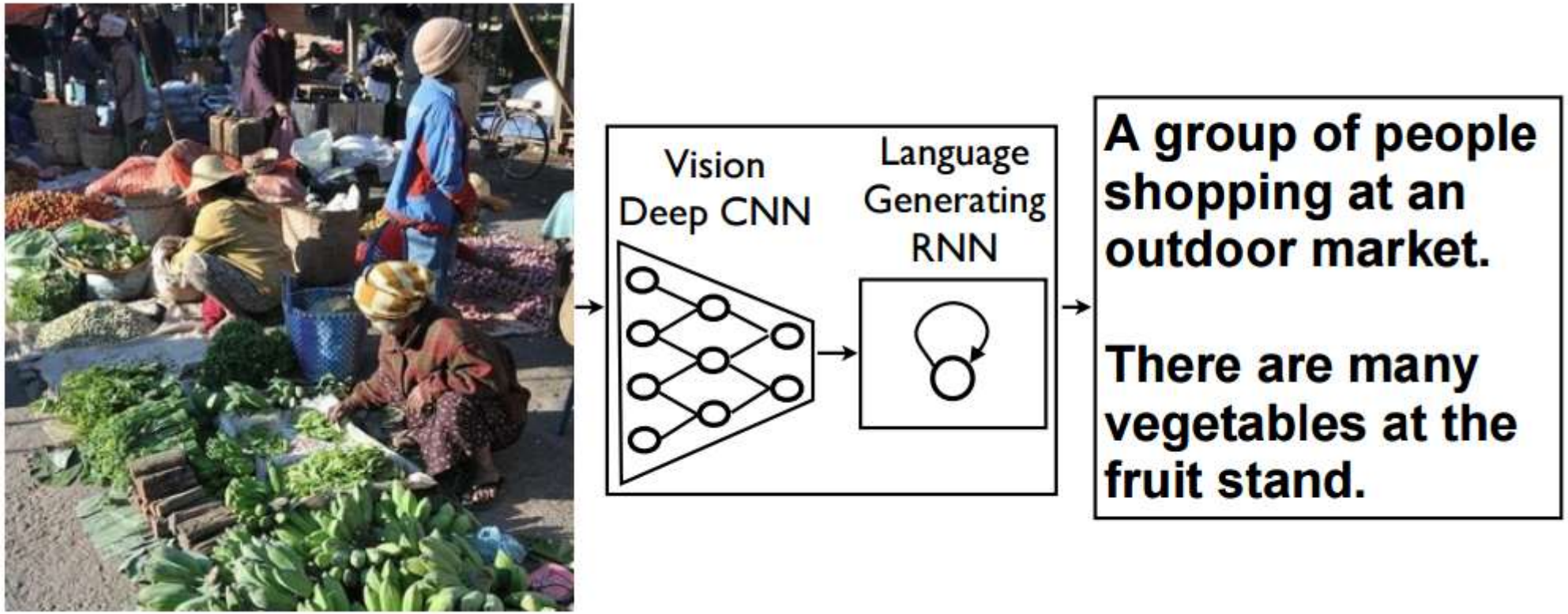


Sequence-to-sequence Based Machine Translation

- A RNN used for encoding
- Another RNN used for decoding



Picture Talk



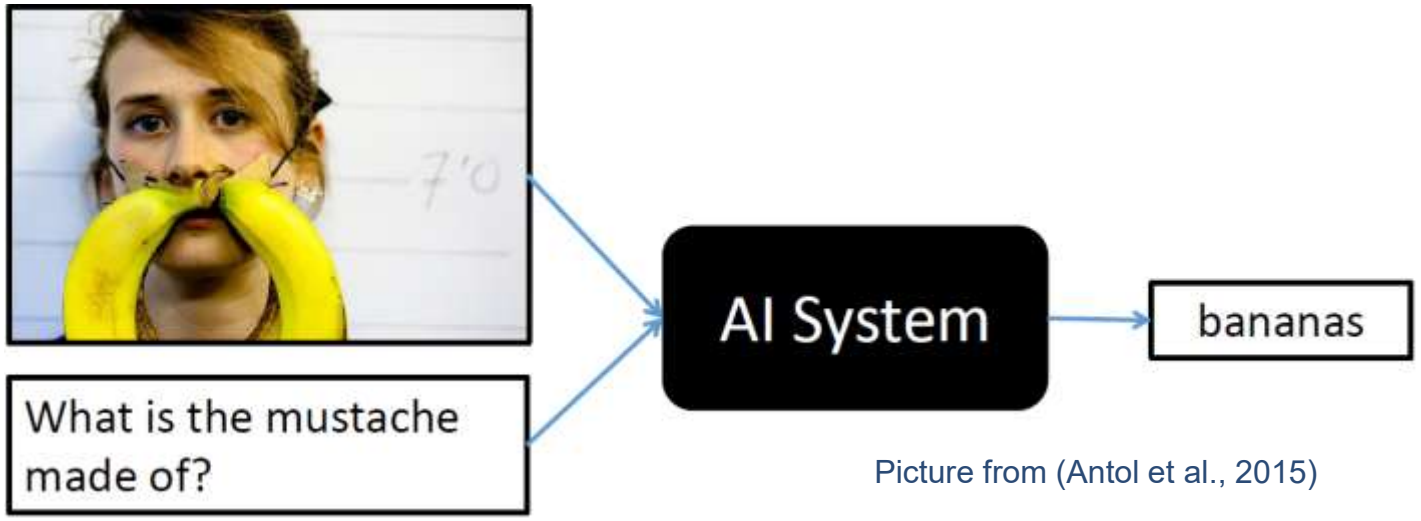
Picture Talk

Figure 5. A selection of evaluation results, grouped by human rating.

Visual Question Answering (VQA)

Demo Website

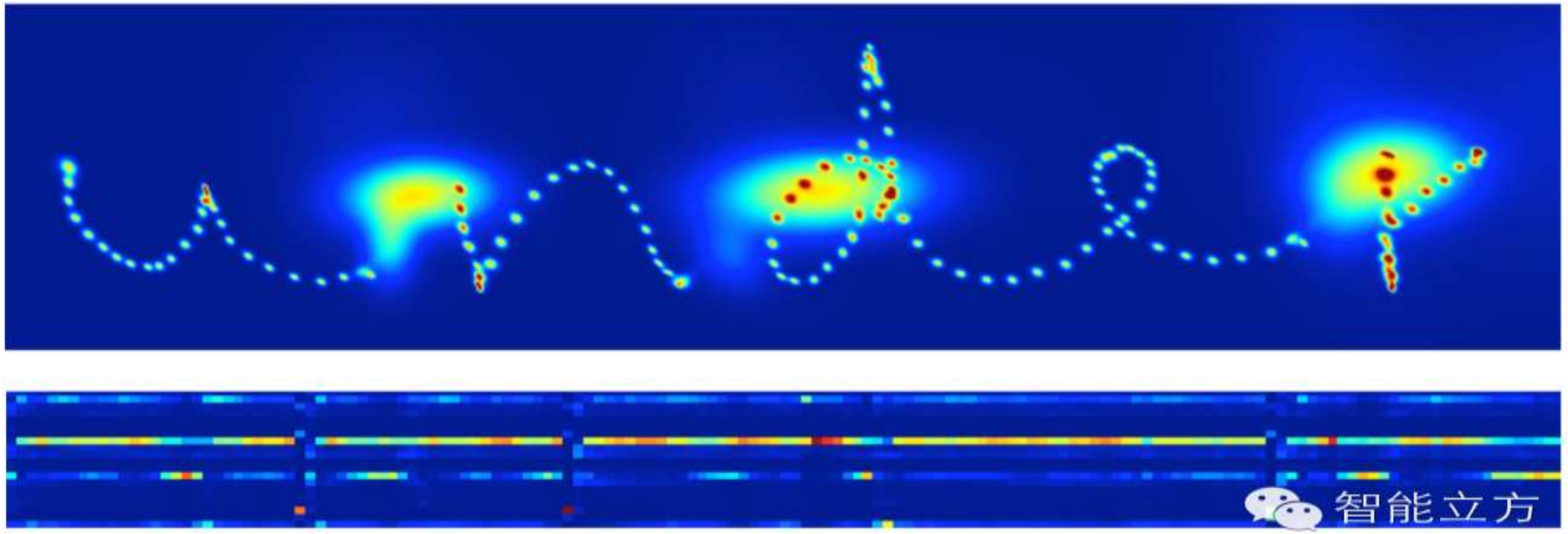
VQA: Given an image and a natural language question about the image, the task is to provide an accurate natural language answer



Picture from (Antol et al., 2015)

Writing

- 把一个字母的书写轨迹看作是一连串的点。一个字母的“写法”其实是每一个点相对于前一个点的偏移量，记为(offset x, offset y)。再增加一维取值为0或1来记录是否应该“提笔”。



Dialogue System

https://github.com/lukalabs/cakechat