Neural Networks

DCS310

Sun Yat-sen University

Outline

- Neural Networks
- Backpropagation
- Typical Neural Networks

Motivation

- Expressive nonlinear models are required in many applications
- Existing non-linearization methods

a) Feature transformation with basis function $\boldsymbol{\phi}(\cdot)$

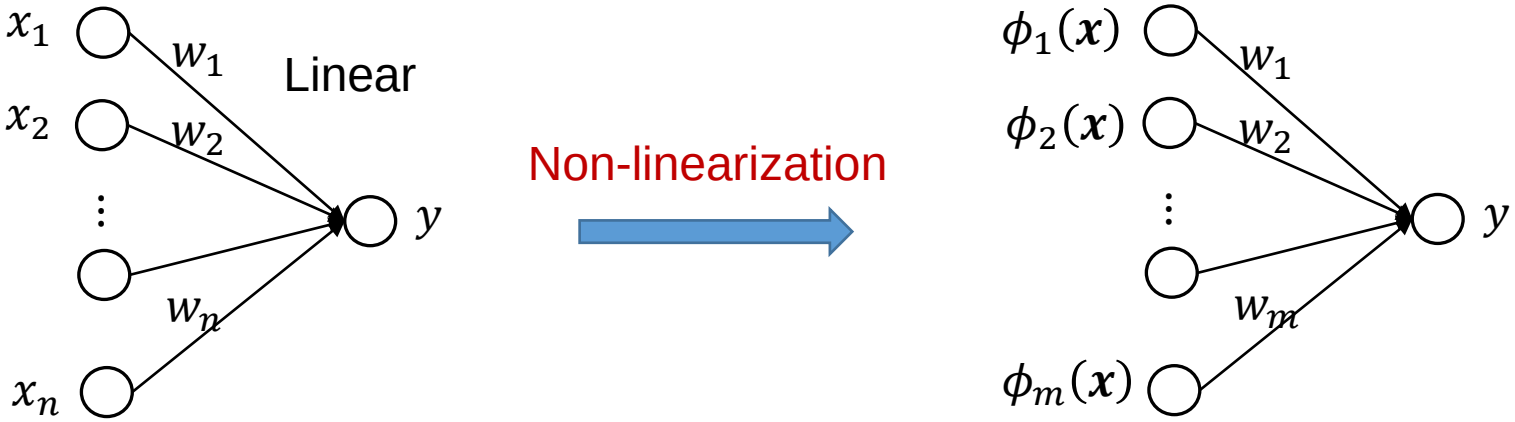
$$\boldsymbol{\phi}: \boldsymbol{x} \longrightarrow \boldsymbol{\phi}(\boldsymbol{x})$$

b) Kernel method, which can be understood as transformation with an infinite-dimensional basis function $\boldsymbol{\phi}(\cdot)$

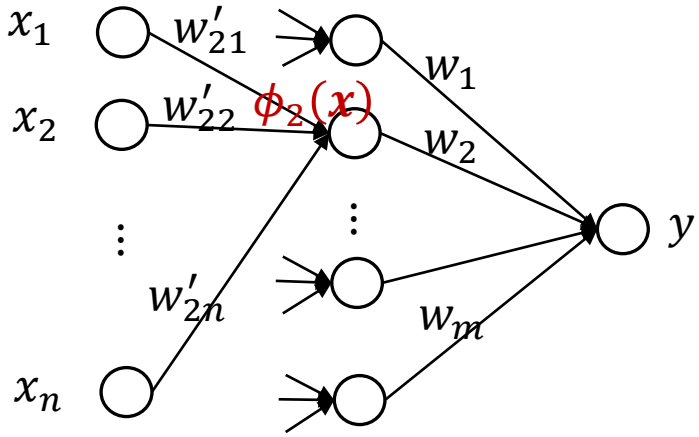
In both methods, the basis functions are fixed and *don't adaptively change according to the data*

Neural Networks

- The previous non-linearization process can be illustrated as



- To increase the flexibility, we let the function $\phi_i(\boldsymbol{x})$ *be learnable*



- The function $\phi_i(\boldsymbol{x})$ can be set as

$$\phi_i(\boldsymbol{x}) = a(\sum_{\ell=1}^{n} w'_{i\ell} x_\ell)$$

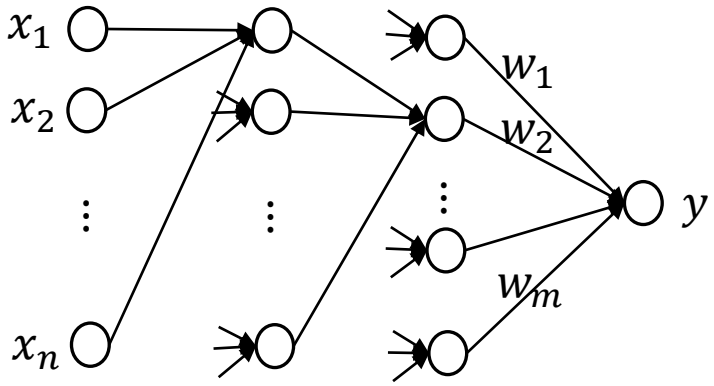
where $a(\cdot)$ is the activation function

> $a(\cdot)$ cannot be omitted, otherwise the output y is linearly related to the input $\boldsymbol{x}$

- The output y can be concisely written as

$$\hat{y}(\boldsymbol{x}) = \boldsymbol{w}_2^T a(\boldsymbol{W}_1 \boldsymbol{x})$$

- Further increase the number of layers

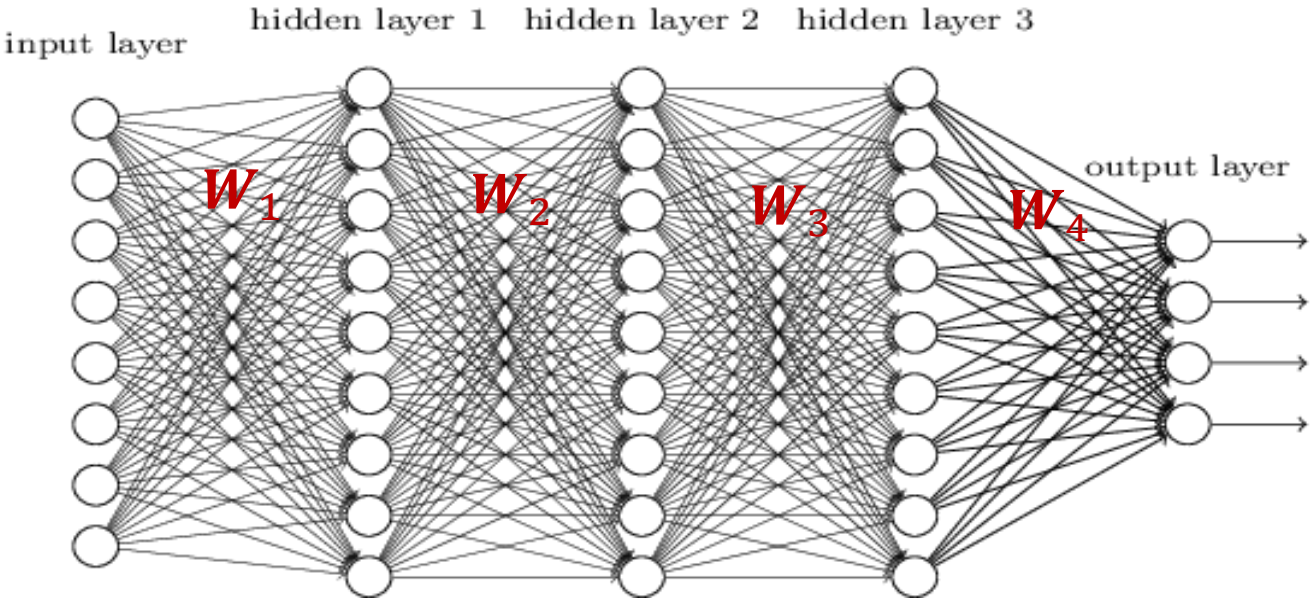


- In general, the output of a L-layer NN can be represented as

Regression: $\hat{y}(\boldsymbol{x}) = \boldsymbol{W}_L a\left(\cdots a\left(\boldsymbol{W}_2 a(\boldsymbol{W}_1 \boldsymbol{x})\right)\right)$

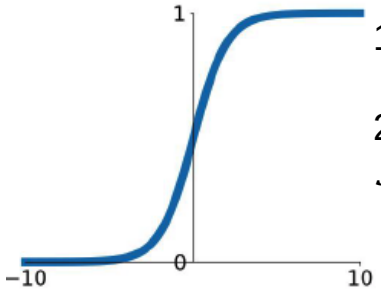
Classification: $\hat{y}(\boldsymbol{x}) = softmax\left(\boldsymbol{W}_L a\left(\cdots a\left(\boldsymbol{W}_2 a(\boldsymbol{W}_1 \boldsymbol{x})\right)\right)\right)$

where $\boldsymbol{W}_\ell$ is the parameter of the ℓ-th layer



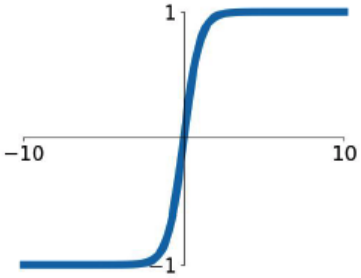
- Activation functions

Sigmoid: $a(x) = \frac{1}{1+e^{-x}}$



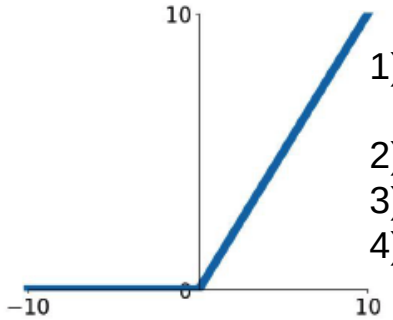
1) Vanishing gradient due to saturate
2) Output only positive values
3) $\exp(\cdot)$ is computationally expensive

Tanh: $a(x) = \frac{1-e^{-2x}}{1+e^{-2x}}$



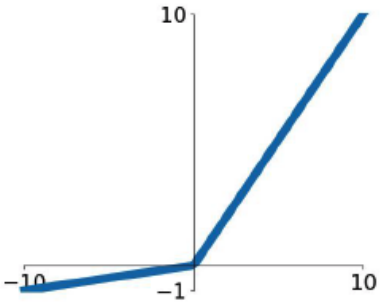
1) Vanishing gradient due to saturate
2) $\exp(\cdot)$ is computationally expensive
3) Output both positive and negative values

ReLU: $a(x) = \max(0, x)$



1) No vanishing gradient problem in x>0
2) Gradient=0 in x<0
3) Computationally efficient
4) Output only positive values

Leaky ReLU: $a(x) = \max(0.1x, x)$



1) No vanishing gradient problem for all x
2) Computationally efficient

Loss Functions

- Given a training dataset $\mathcal{D} = \{\boldsymbol{x}^{(i)}, y^{(i)}\}_{i=1}^{N}$, the training objectives for the regression and classification are as follows

1) Regression loss

$$\mathcal{L}_r(\boldsymbol{\theta}) = \frac{1}{N}\sum_{(\boldsymbol{x},y)\in\mathcal{D}} |y - \hat{y}(\boldsymbol{x})|^2$$

2) Multi-class classification loss (cross-entropy)

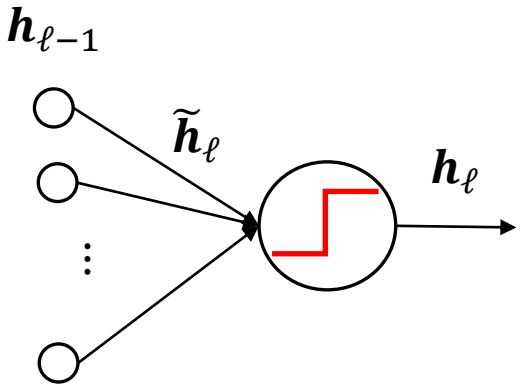
$$\mathcal{L}_c(\boldsymbol{\theta}) = \frac{1}{N}\sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{D}} \sum_{k=1}^{K} -y_k \log \hat{y}_k(\boldsymbol{x})$$

Outline

- Neural Networks
- Backpropagation
- Typical Neural Networks

Gradient

- To train NNs, we need the gradient of loss $\mathcal{L}$ *w.r.t.* $\boldsymbol{W}_\ell$, *i.e.*, $\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}_\ell}$
- To this end, we express the outputs of NNs in a recursive way as

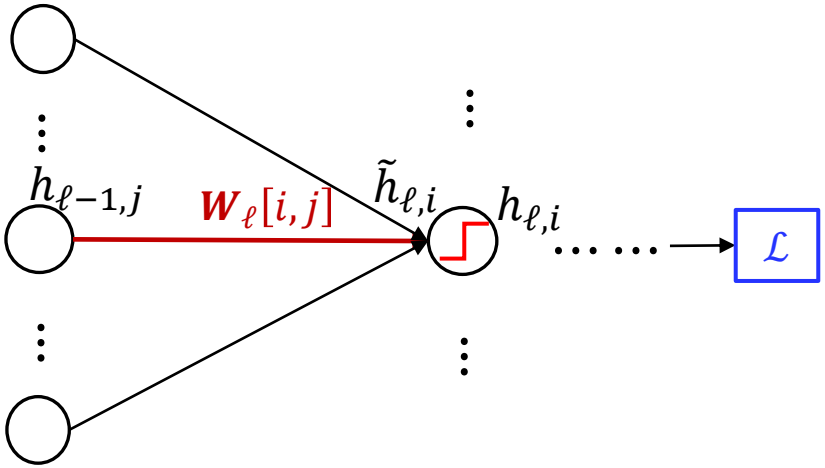


$$\tilde{\boldsymbol{h}}_\ell = \boldsymbol{W}_\ell \boldsymbol{h}_{\ell-1}, \qquad \boldsymbol{h}_\ell = a(\tilde{\boldsymbol{h}}_\ell) \qquad \boldsymbol{h}_\ell = a(\boldsymbol{W}_\ell \boldsymbol{h}_{\ell-1})$$

- The output layer

$$\hat{y} = h_L = \tilde{h}_L \text{ or } softmax(\tilde{h}_L)$$

- The derivative of the loss w.r.t. $\boldsymbol{W}_\ell[i,j]$, that is, $\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}_\ell[i,j]}$



$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}_\ell[i,j]} = \frac{\partial \mathcal{L}}{\partial h_{\ell,i}} \cdot \frac{\partial h_{\ell,i}}{\partial \boldsymbol{W}_\ell[i,j]}$$

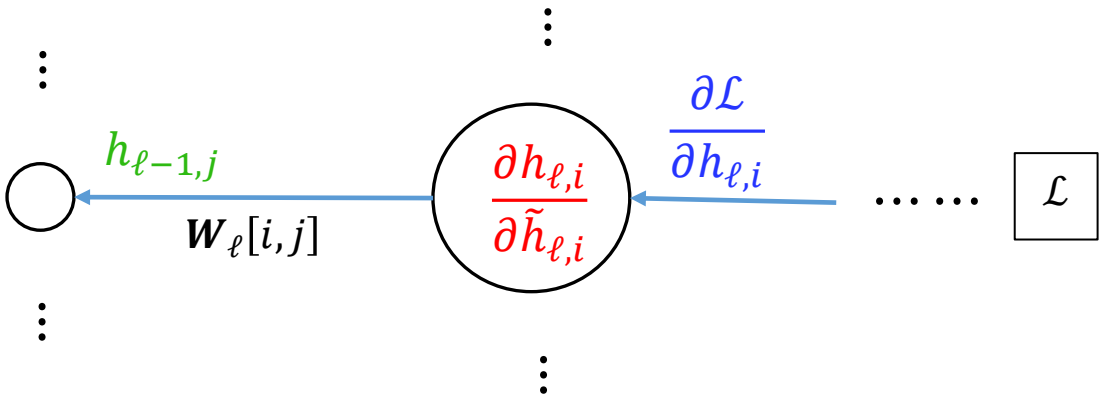
$$= \frac{\partial \mathcal{L}}{\partial h_{\ell,i}} \cdot \frac{\partial h_{\ell,i}}{\partial \tilde{h}_{\ell,i}} \cdot \frac{\partial \tilde{h}_{\ell,i}}{\partial \boldsymbol{W}_\ell[i,j]}$$

$$= \boxed{\frac{\partial \mathcal{L}}{\partial h_{\ell,i}}} \cdot \frac{\partial h_{\ell,i}}{\partial \tilde{h}_{\ell,i}} \cdot h_{\ell-1,j}$$

Activation function derevative

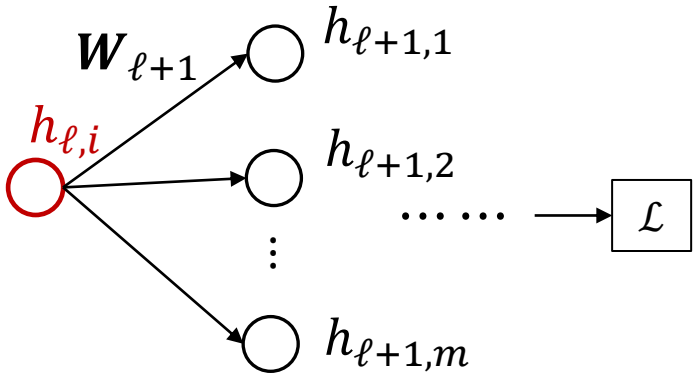
- Schematic showing the structure of the derivative

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}_\ell[i,j]} = \frac{\partial \mathcal{L}}{\partial h_{\ell,i}} \cdot \frac{\partial h_{\ell,i}}{\partial \tilde{h}_{\ell,i}} \cdot h_{\ell-1,j}$$



To compute $\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}_\ell[i,j]}$, *the key is to compute the derivative* $\frac{\partial \mathcal{L}}{\partial h_{\ell,i}}$

- The derivative of the loss *w.r.t.* $h_{\ell,i}$, that is, $\frac{\partial \mathcal{L}}{\partial h_{\ell,i}}$



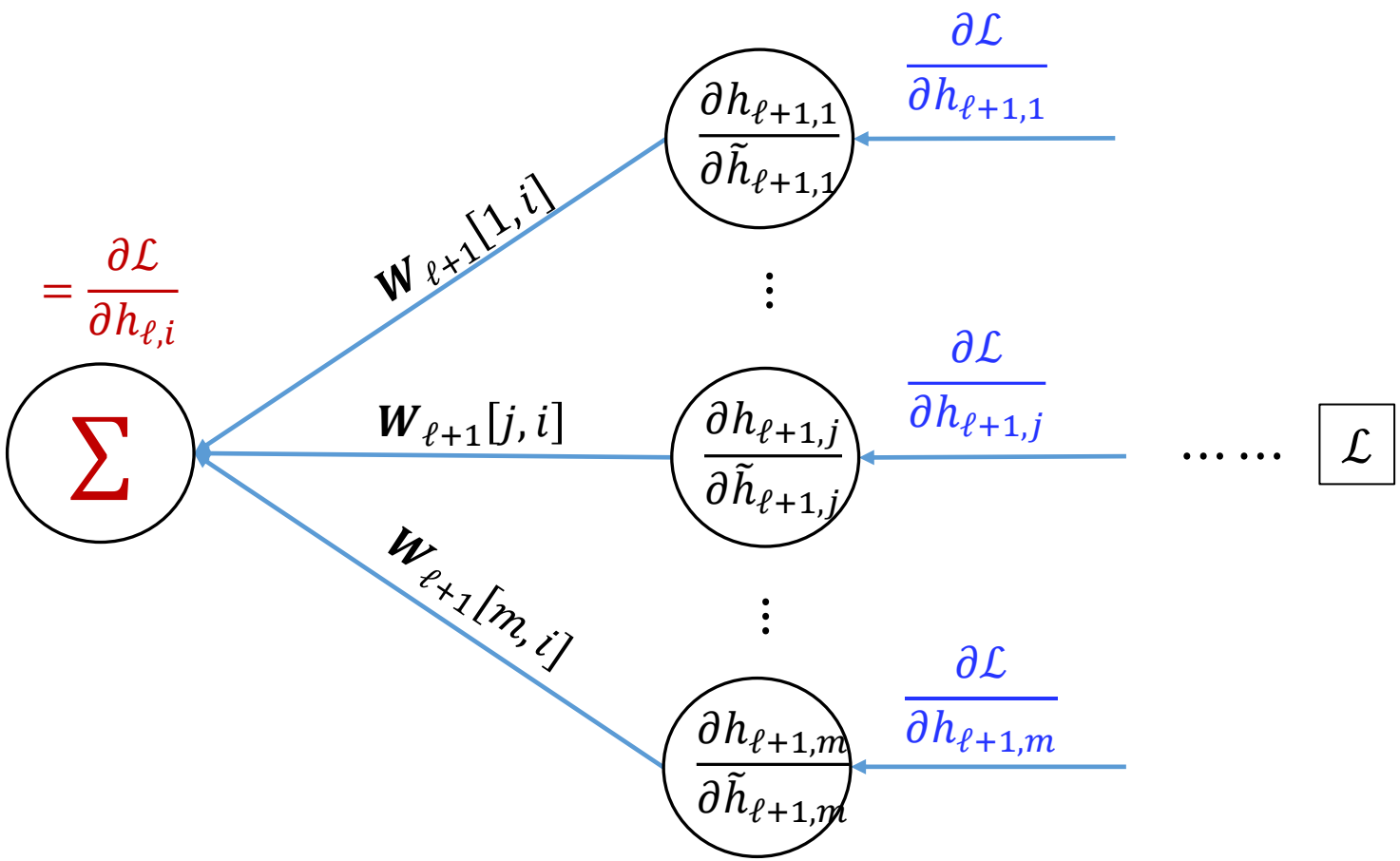
$$\frac{\partial \mathcal{L}}{\partial h_{\ell,i}} = \sum_{j=1}^{m} \left[\frac{\partial \mathcal{L}}{\partial h_{\ell+1,j}} \cdot \frac{\partial h_{\ell+1,j}}{\partial h_{\ell,i}} \right]$$

$$= \sum_{j=1}^{m} \left[\frac{\partial \mathcal{L}}{\partial h_{\ell+1,j}} \cdot \frac{\partial h_{\ell+1,j}}{\partial \tilde{h}_{\ell+1,j}} \cdot \frac{\partial \tilde{h}_{\ell+1,j}}{\partial h_{\ell,i}} \right]$$

$$= \sum_{j=1}^{m} \left[\boxed{\frac{\partial \mathcal{L}}{\partial h_{\ell+1,j}}} \cdot \frac{\partial h_{\ell+1,j}}{\partial \tilde{h}_{\ell+1,j}} \cdot \boldsymbol{W}_{\ell+1}[j,i] \right]$$

- The recursive structure in the expression

$$\frac{\partial \mathcal{L}}{\partial h_{\ell,i}} = \sum_{j=1}^{m} \left[\frac{\partial \mathcal{L}}{\partial h_{\ell+1,j}} \cdot \frac{\partial h_{\ell+1,j}}{\partial \tilde{h}_{\ell+1,j}} \cdot \boldsymbol{W}_{\ell+1}[j,i] \right]$$



- The initial derivative of the loss w.r.t. the last layer output h_L

1) Regression loss: $\mathcal{L} = \frac{1}{2}|y - h_L|^2$

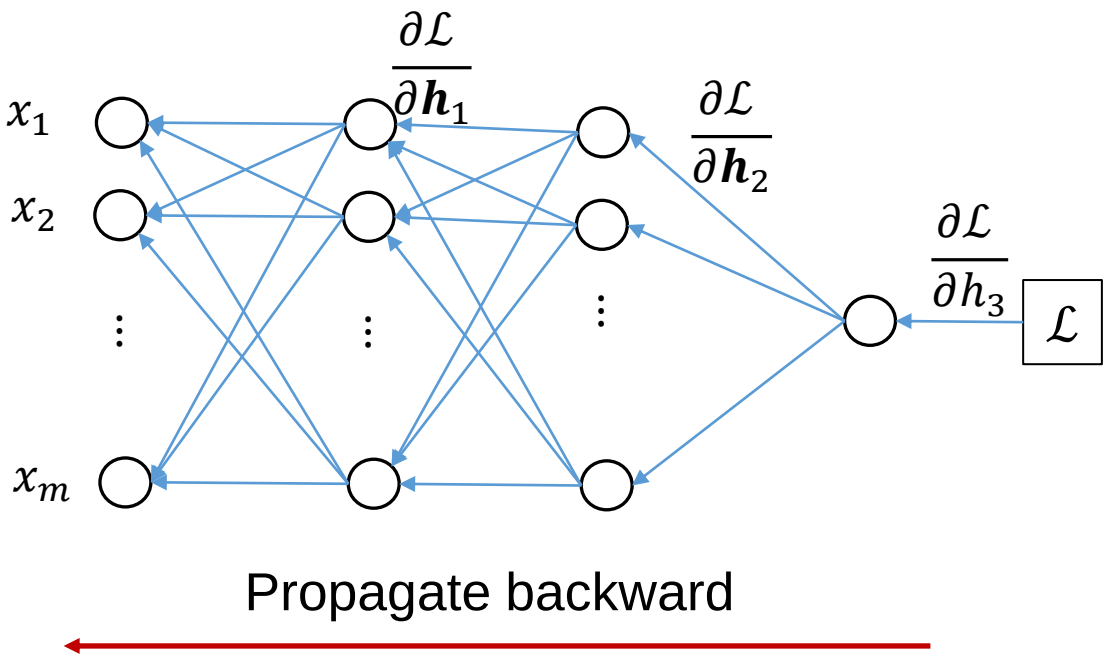
$$\frac{\partial \mathcal{L}}{\partial h_L} = (h_L - y)$$

2) Cross-entropy loss: $\mathcal{L} = -y \log h_L - (1 - y) \log(1 - h_L)$

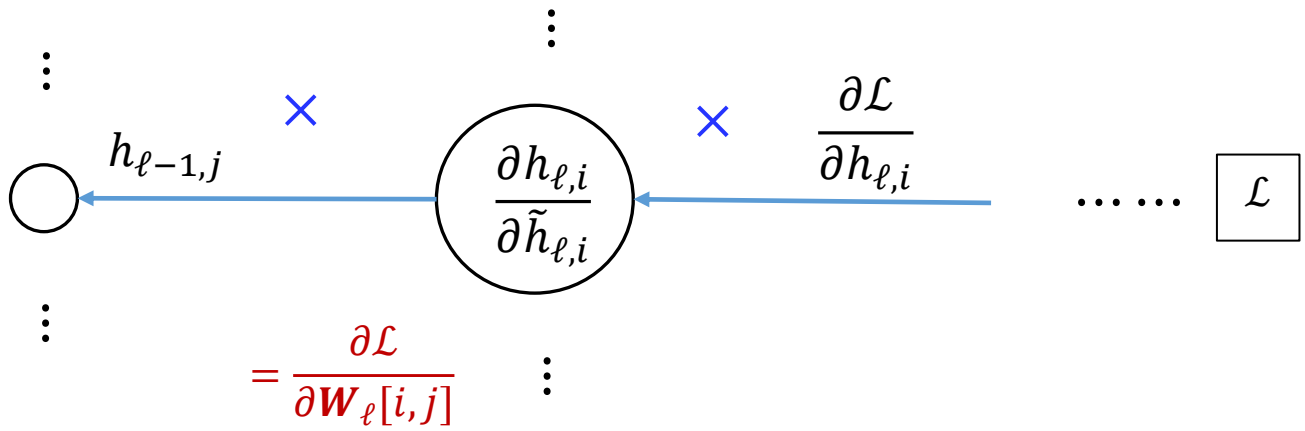
$$\frac{\partial \mathcal{L}}{\partial h_L} = \frac{h_L - y}{h_L(1 - h_L)}$$

The Whole Computation Graph of Gradient

- Two steps
  - Step 1: Computation of $\frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_\ell}$



➢ Step 2: Computation of $\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}_\ell}$



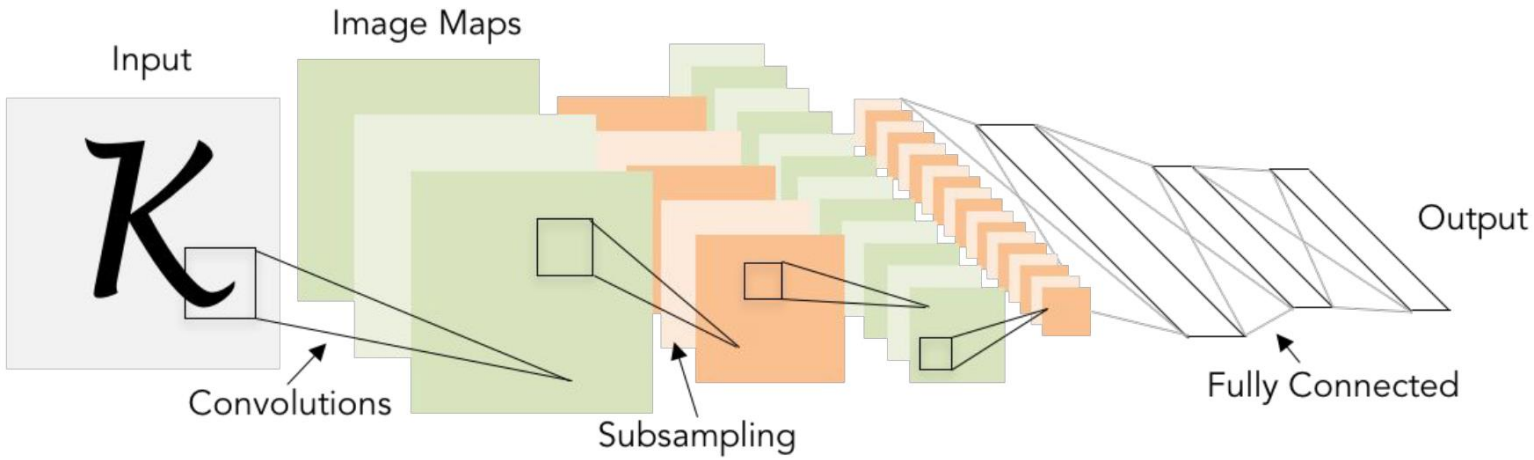
$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}_\ell[i,j]} = \frac{\partial \mathcal{L}}{\partial h_{\ell,i}} \times \frac{\partial h_{\ell,i}}{\partial \tilde{h}_{\ell,i}} \times h_{\ell-1,j}$$

Outline

- Neural Networks
- Backpropagation
- Typical Neural Networks

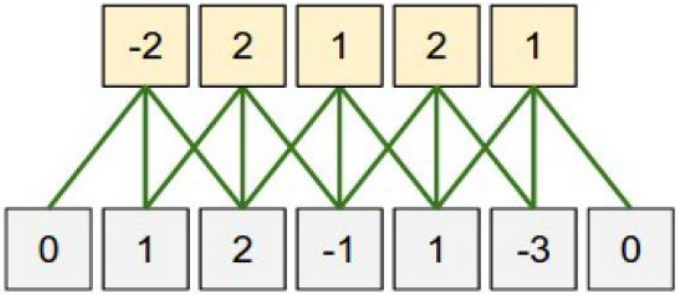
Convolutional Neural Networks (CNN)

- Weakness of fully connected NNs (also called multilayer perception (MLP))
  1) The number parameters are huge, prone to result in overfitting
  2) They didn't consider/exploit any structural characteristics of the data
- CNNs are proposed to leverage the spatial structures (*e.g.* translational invariance) in the image/visual data

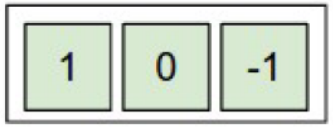


Convolution Operations

- One-dimensional convolution operation illustration



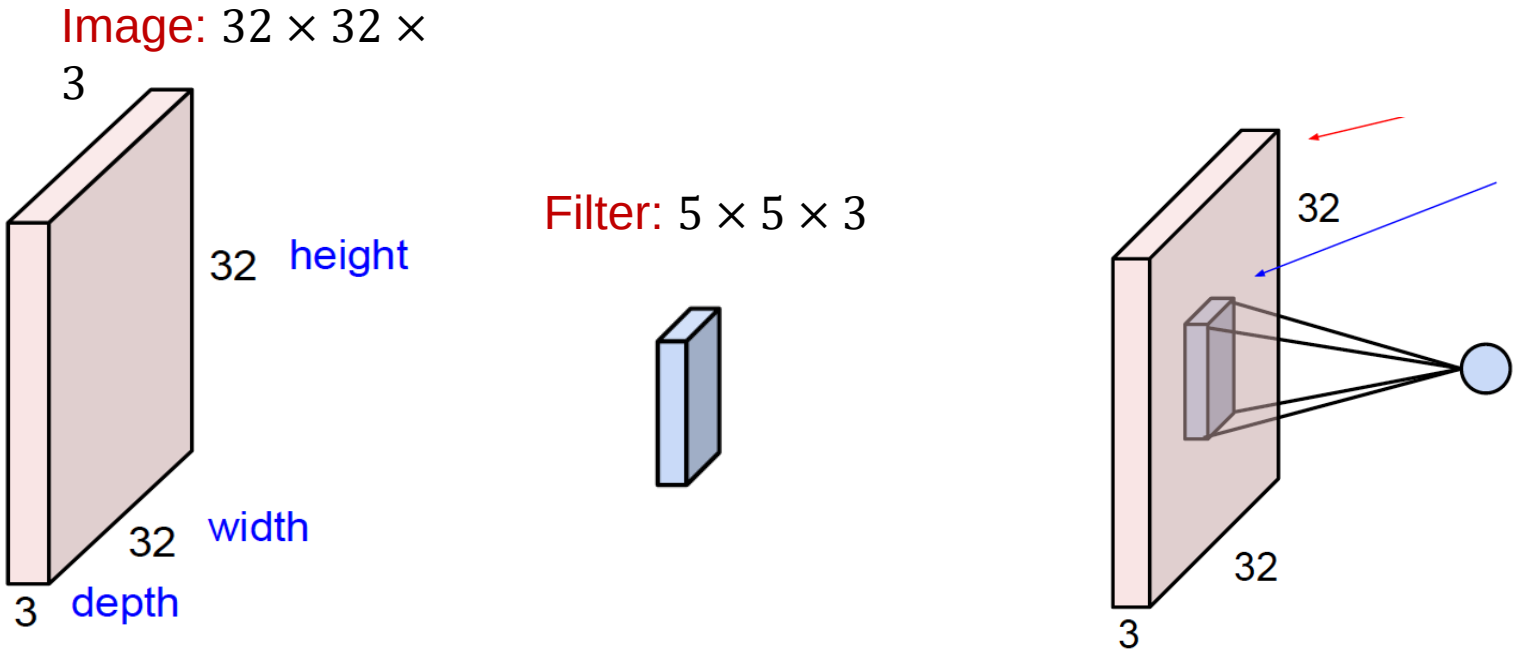
Neural Network



The Filter

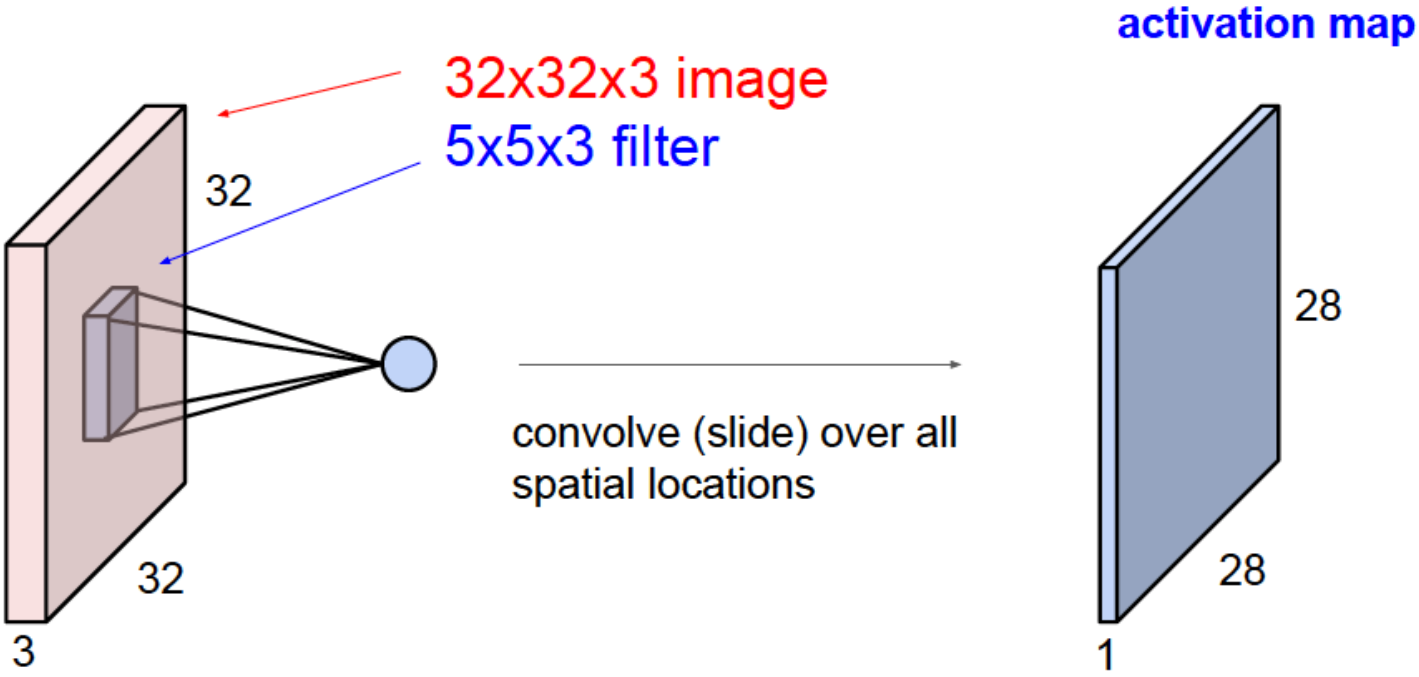
- Distinctions from the MLPs
  - Local connectivity: each neuron is only connected to a fraction of neurons in previous layer
  - Parameter sharing: the parameters for different neurons are tied

- Two-dimensional convolution operation illustration

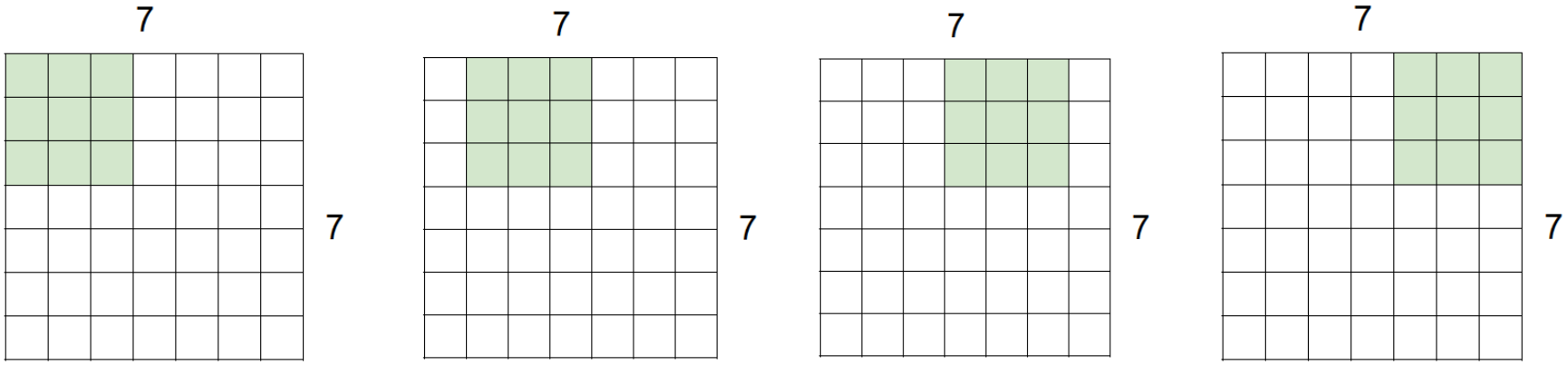


Feature Maps

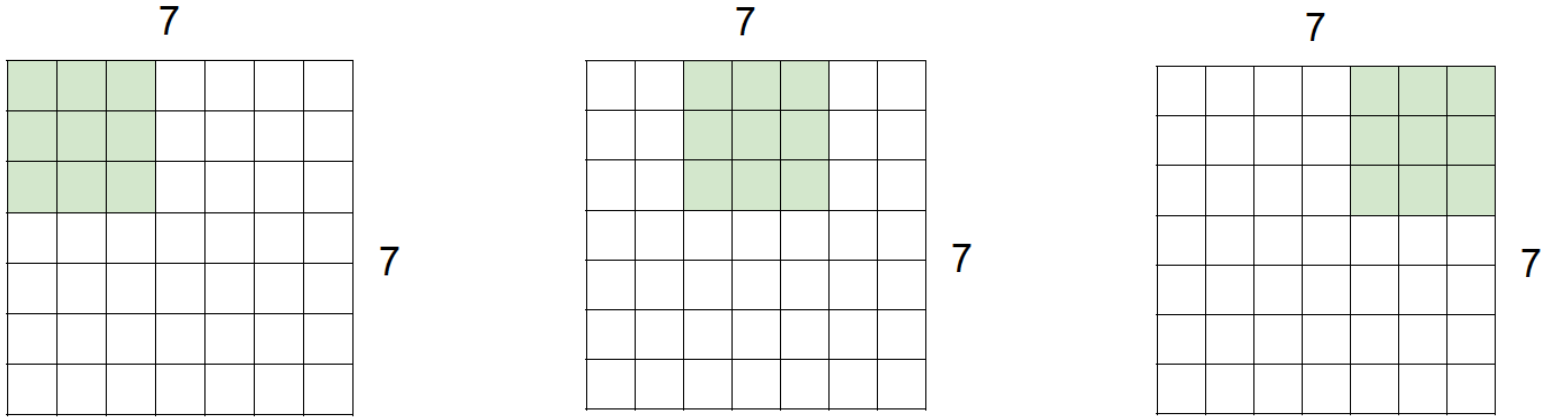
- Sliding the filter along the two dimensions of images, yielding an activation map with dimension $28 \times 28 \times 1$



A closer look at the convolution operation

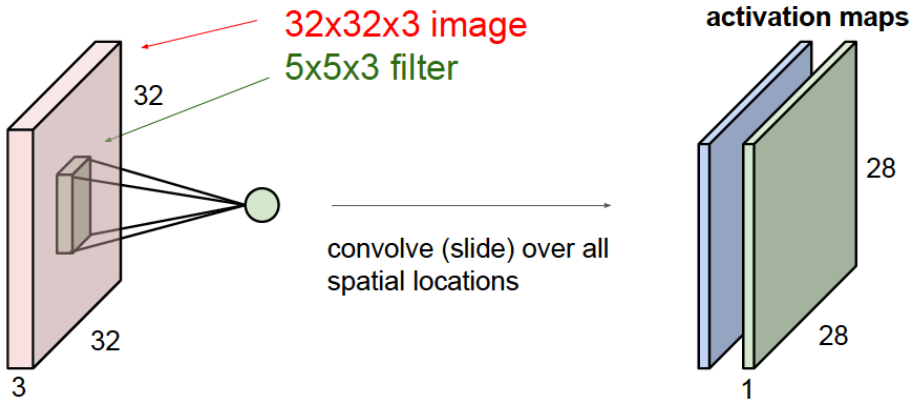


Stride=1

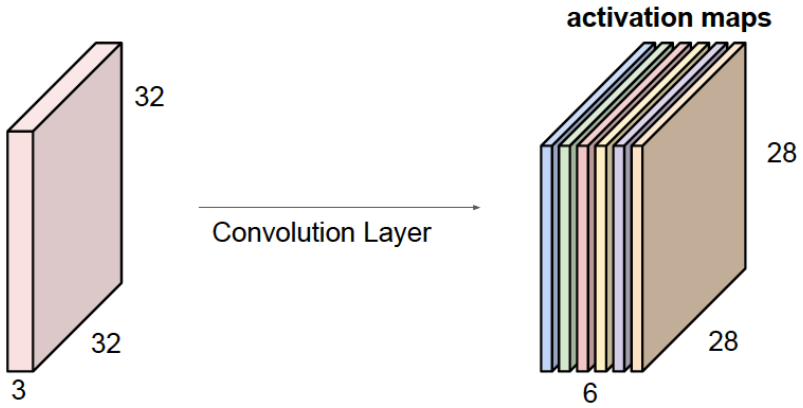


Stride=2

- Convolving the image with another filter, giving rise to another activation map of the same size



- Convolving with multiple filters



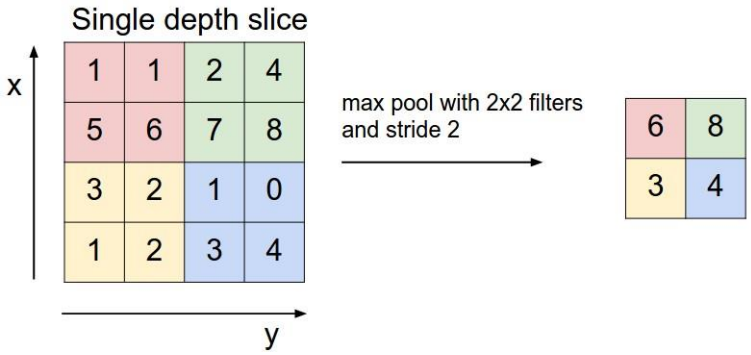
- Why are multiple filters needed?

Different filters account for different patterns in the image

Original | Sharpen | Edge Detect | Strong Edge Detect

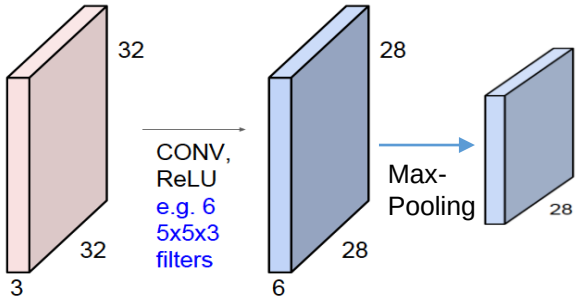
Pooling

- Illustration



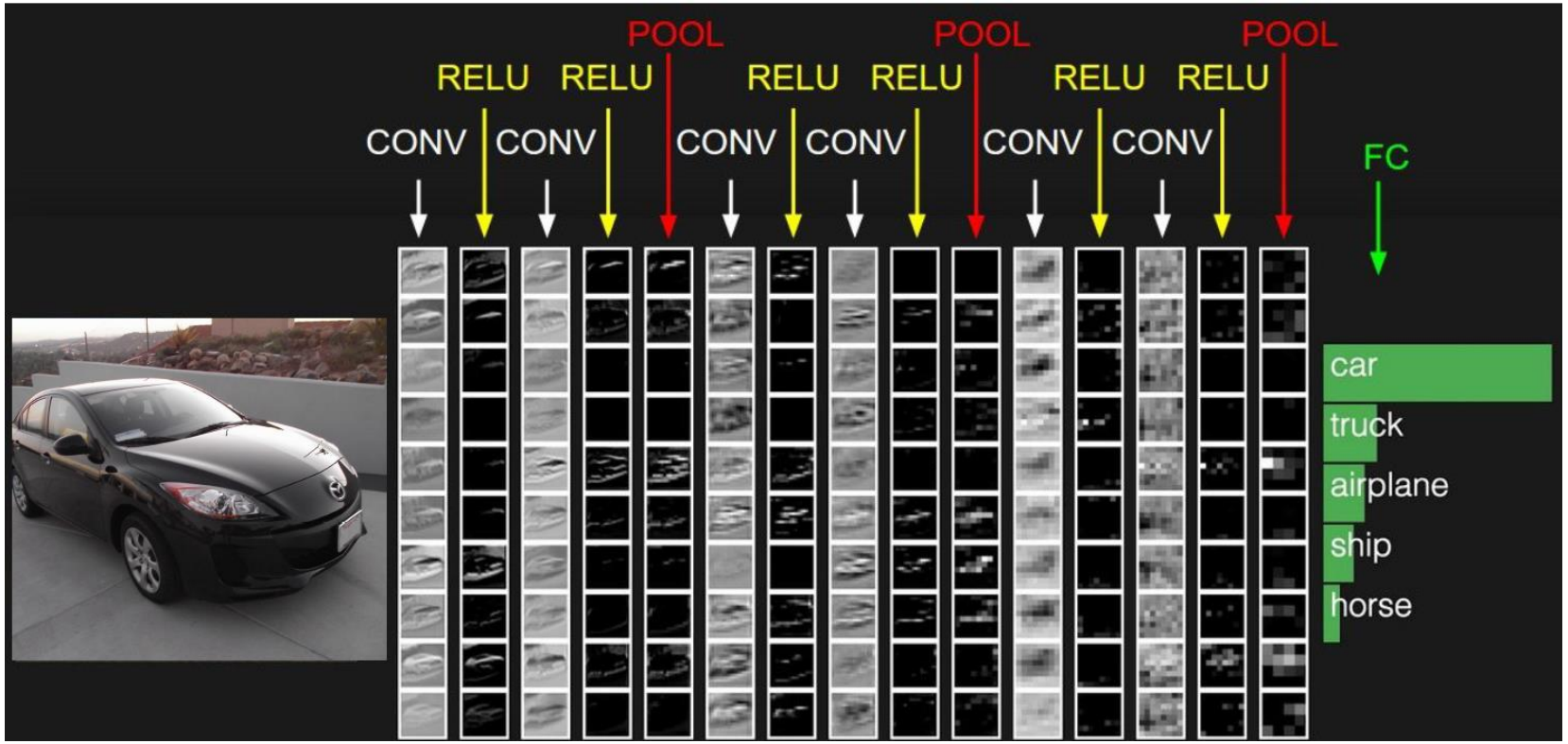
1) Reducing the dimensionality
2) Introducing nonlinearities
3) Maintaining some extent of spatial invariance
4) The pooling is not necessary at all layer

- The basic element consisting of Convolution + ReLU + Max-pooling



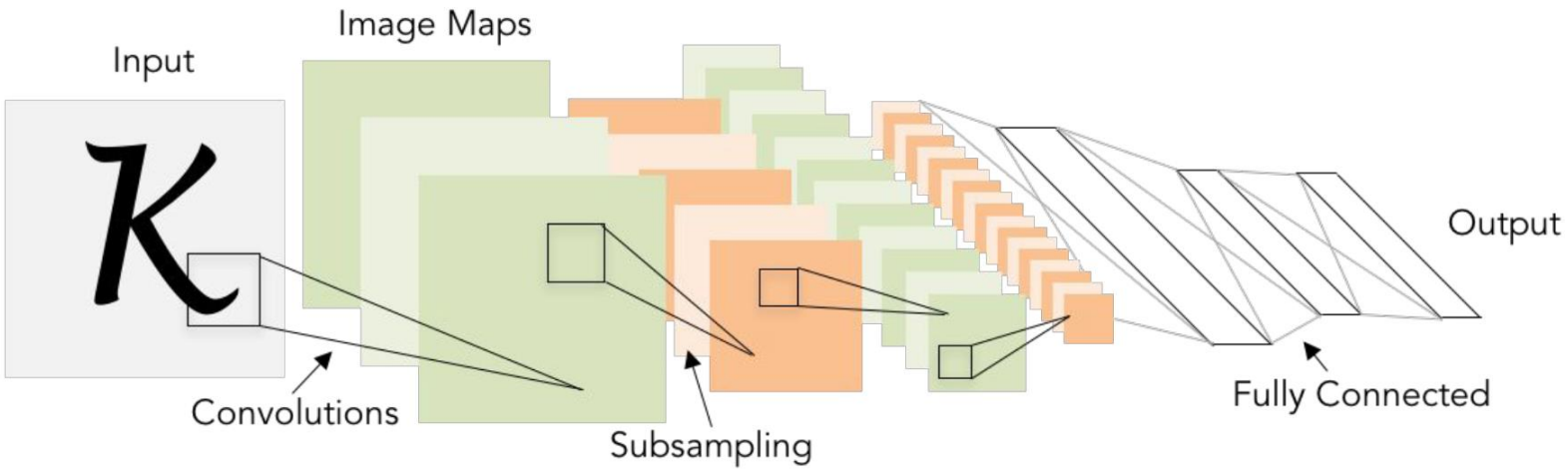
Illustration

- An illustration of image classification with CNN



Several Typical CNNs

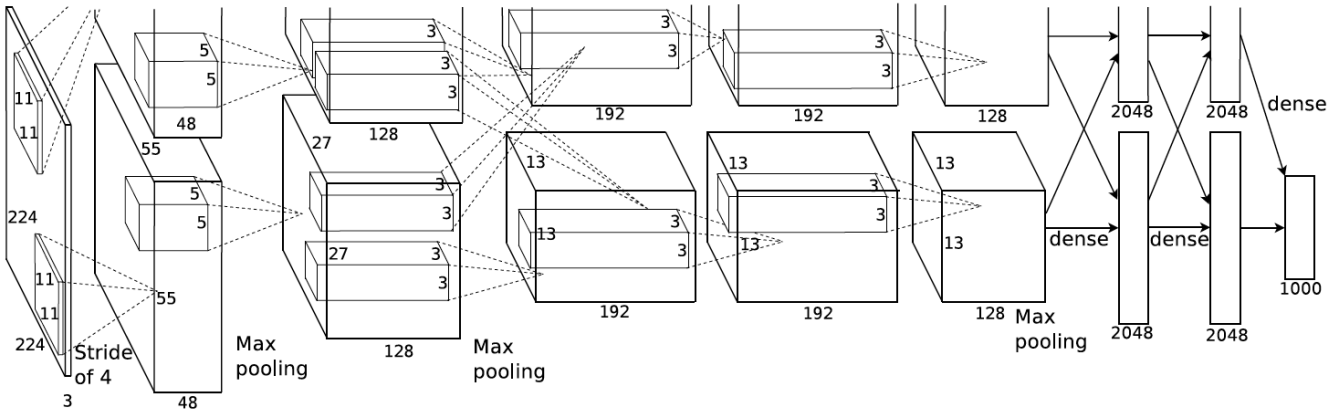
- LeNet



Structure: CONV-POOL-CONV-POOL-FC-FC

Yann LeCun, Bernhard E. Boser, John S. Denker, Donnie Henderson, Richard E. Howard, Wayne E. Hubbard, and Lawrence D. Jackel. Backpropagation applied to handwritten zip code recognition. Neural Computation, 1989.

- AlexNet

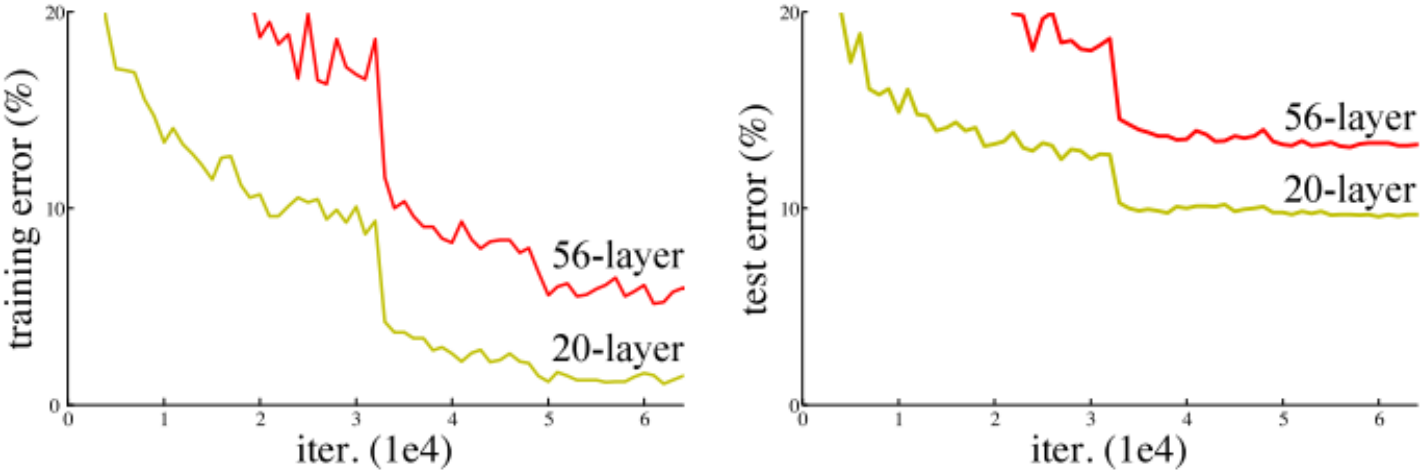


Input	Conv1	MaxPool1	Conv2	MaxPool2	Conv3
$227 \times 227 \times 3$	$55 \times 55 \times 96$	$27 \times 27 \times 96$	$27 \times 27 \times 256$	$13 \times 13 \times 256$	$13 \times 13 \times 384$

Conv4	Conv5	MaxPool3	FC6	FC7	FC8
$13 \times 13 \times 384$	$13 \times 13 \times 256$	$6 \times 6 \times 256$	4096	4096	1000

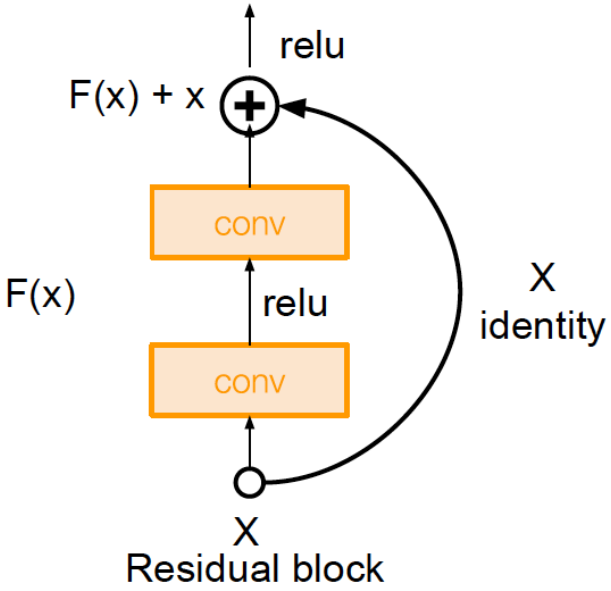
Alex Krizhevsky, Ilya Sutskever and Geoffrey E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. NIPS 2012.

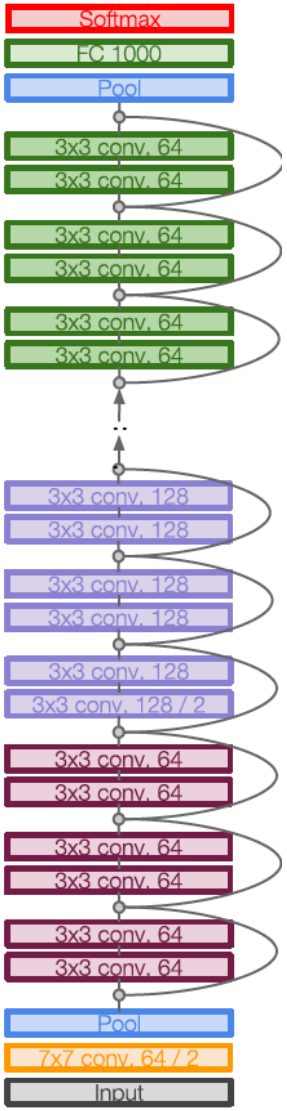
- ResNet
  - Simply stacking layers does not promise performance gains. On the contrary, it may deteriorate the performance



  - One of reasons is that deeper models are more difficult to train
  - ResNet alleviates the training difficulties by introducing a ‘shortcut’ connection between layers

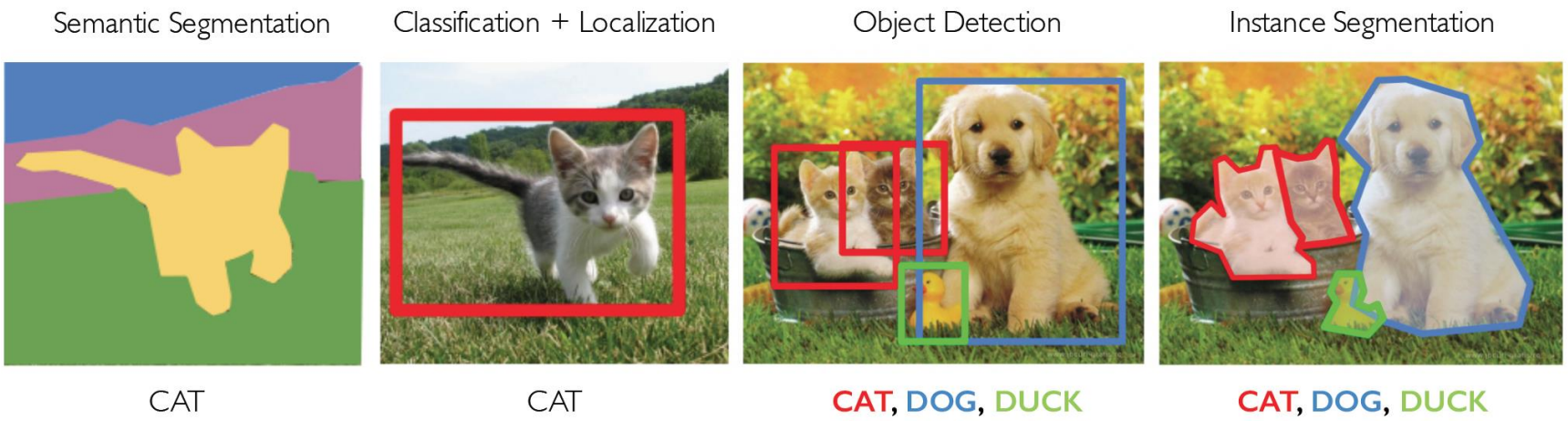
Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun. Deep Residual Learning for Image Recognition. CVPR 2016





The ResNet wining the ILSVRC 2015 classification champion contains as many as 152 layers!

Typical Applications beyond Classification



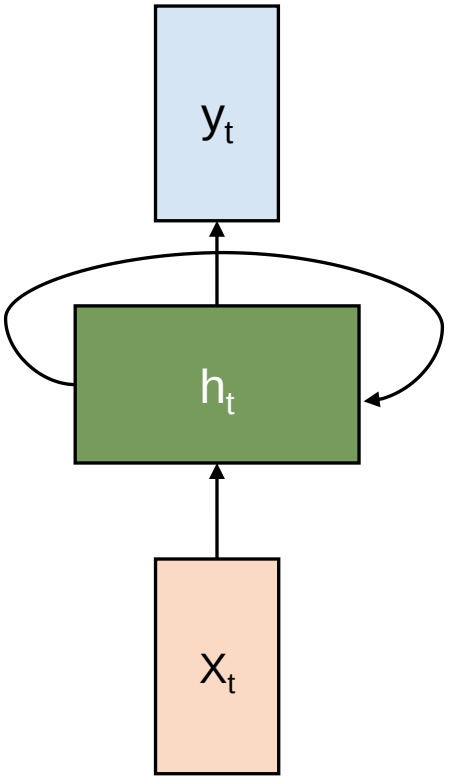
Vanilla RNNs

- The vanilla RNN can be described by two equations:
  1) Hidden state updating equation

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

  2) Output

$$y_t = f_o(h_t)$$

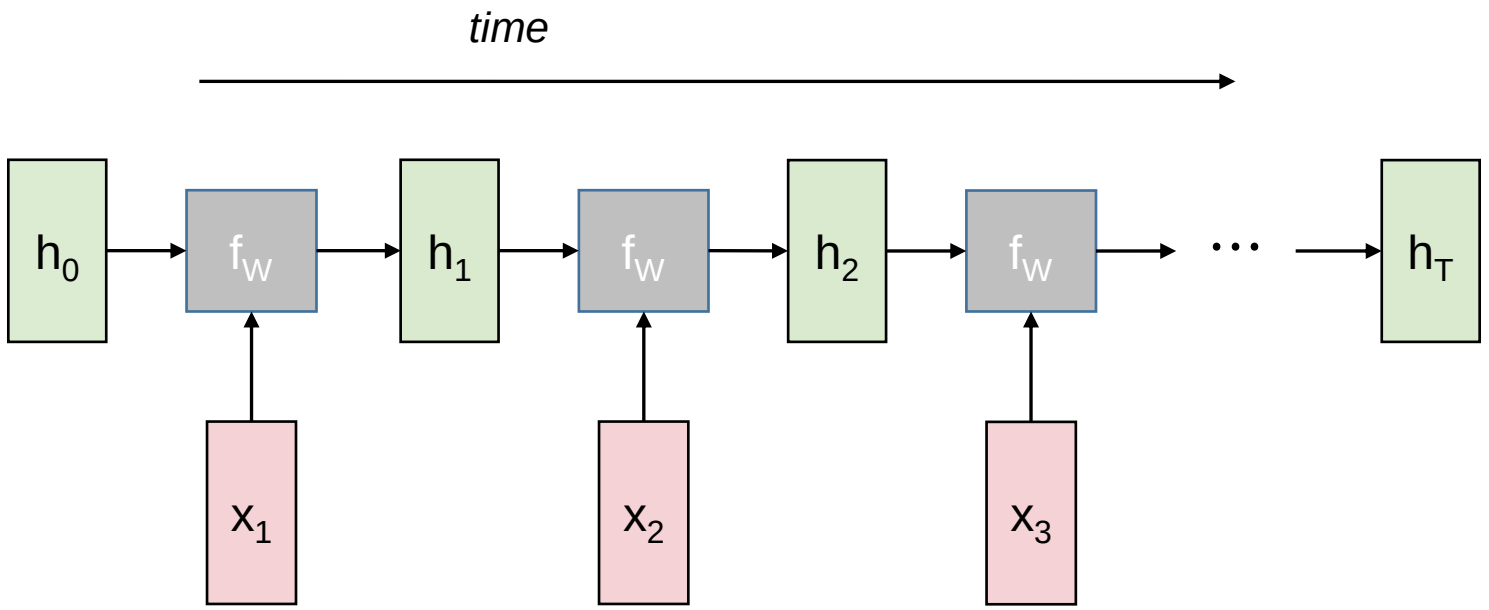


- For instance, $f_h(\cdot)$ and $f_o(\cdot)$ could be

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

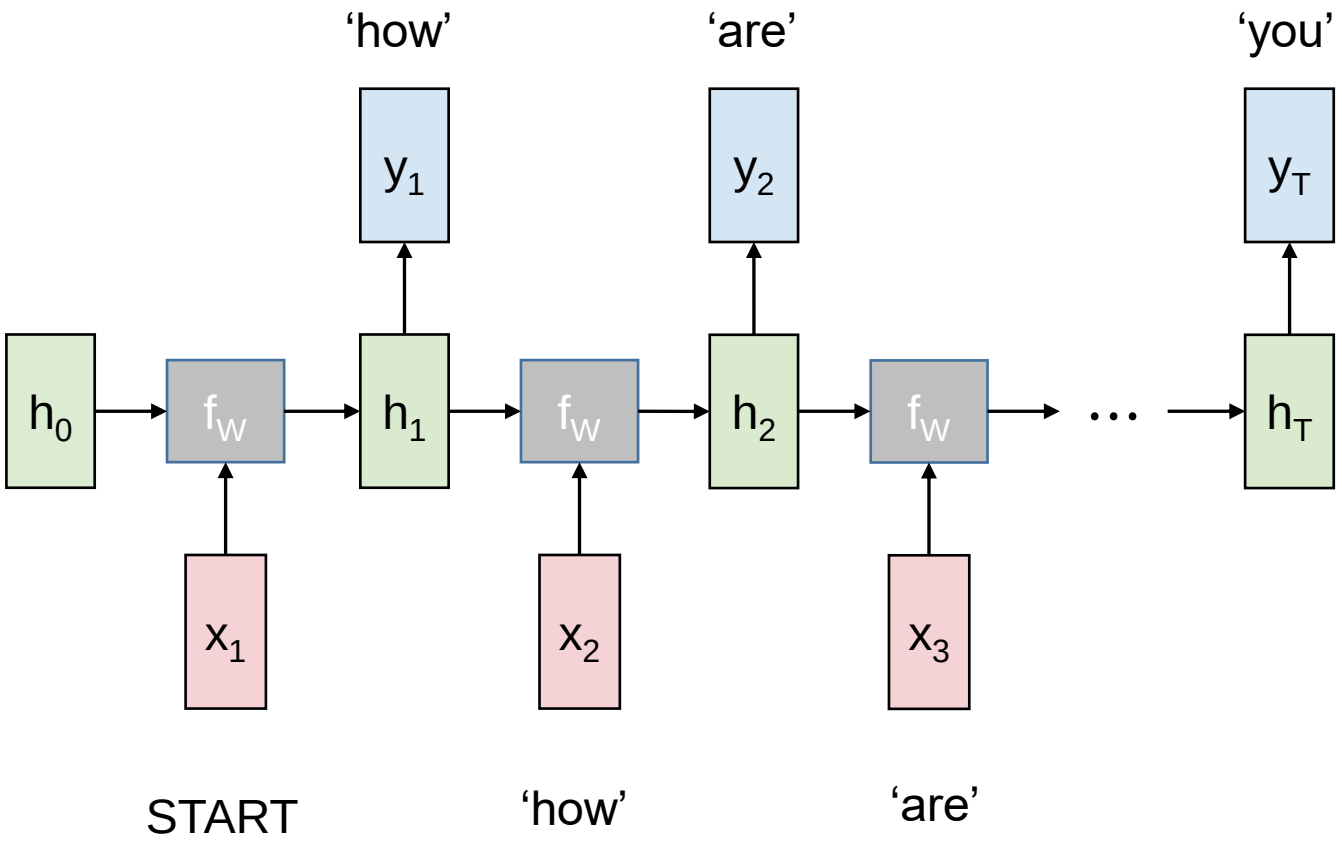
$$y_t = W_{hy}h_t \quad \text{or} \quad y_t = \text{softmax}(W_{hy}h_t)$$

- Unfolding the RNN over time



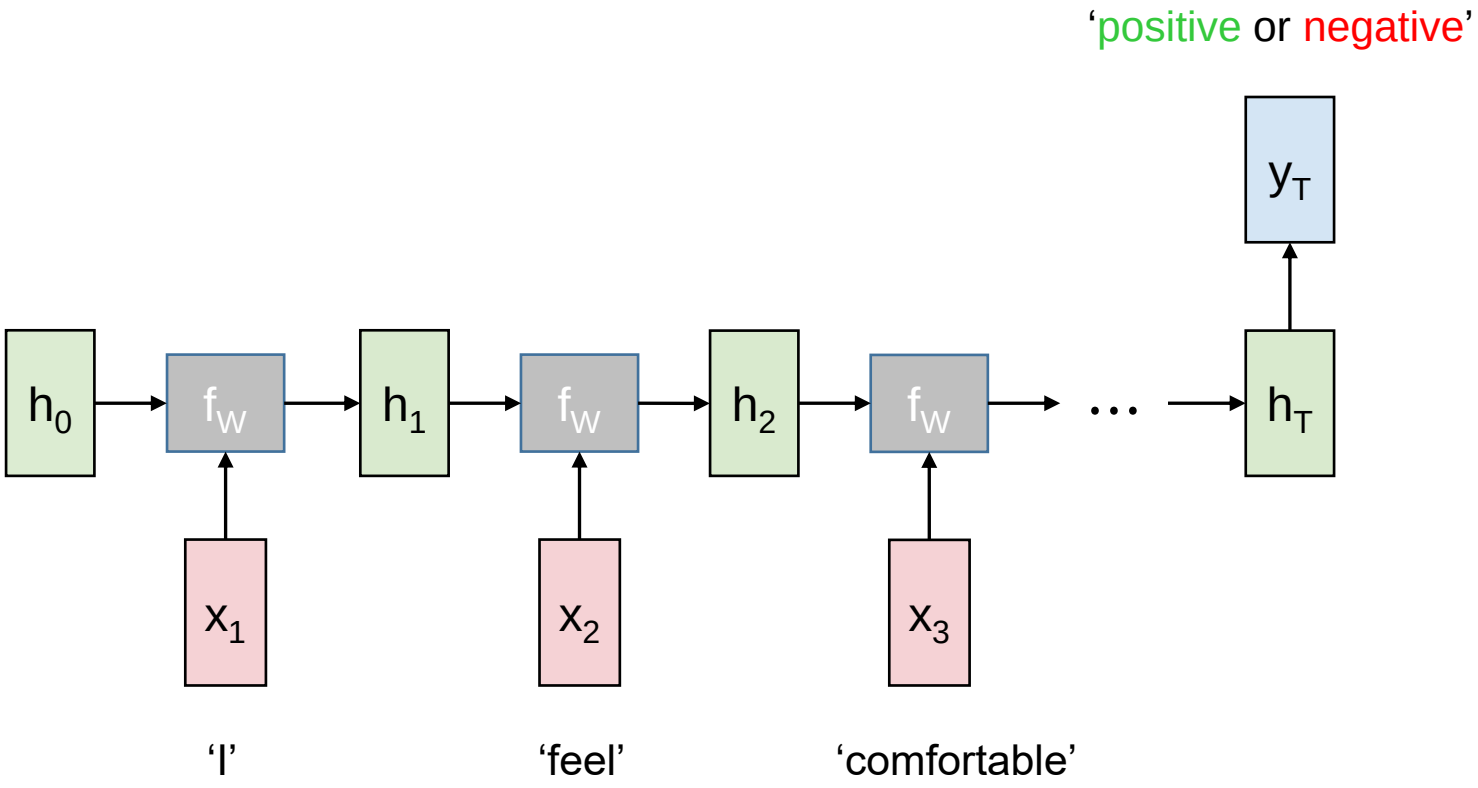
Output Types

- Many to Many



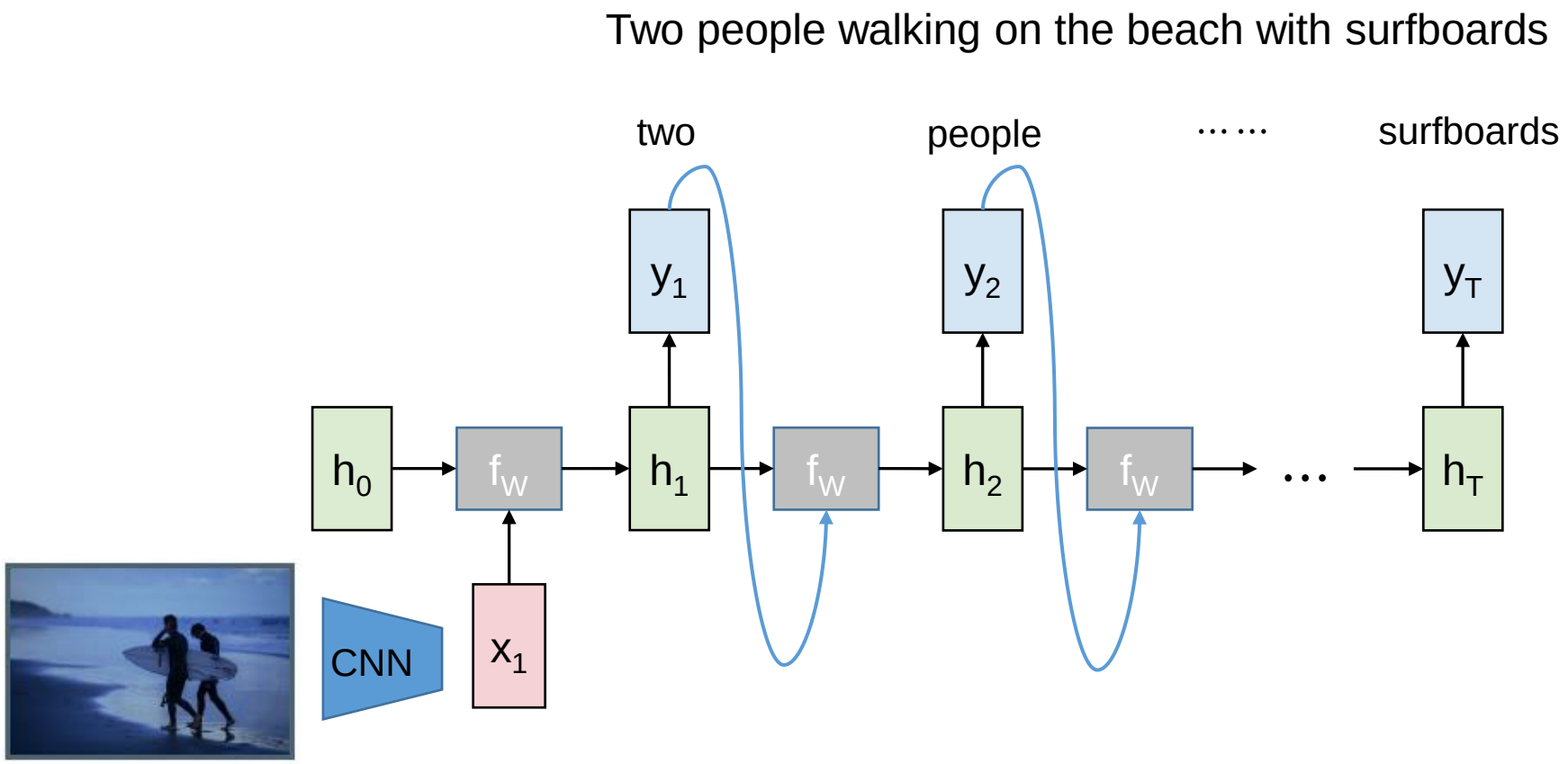
Applications: Language model, frame classification in video, *etc.*

- Many to one



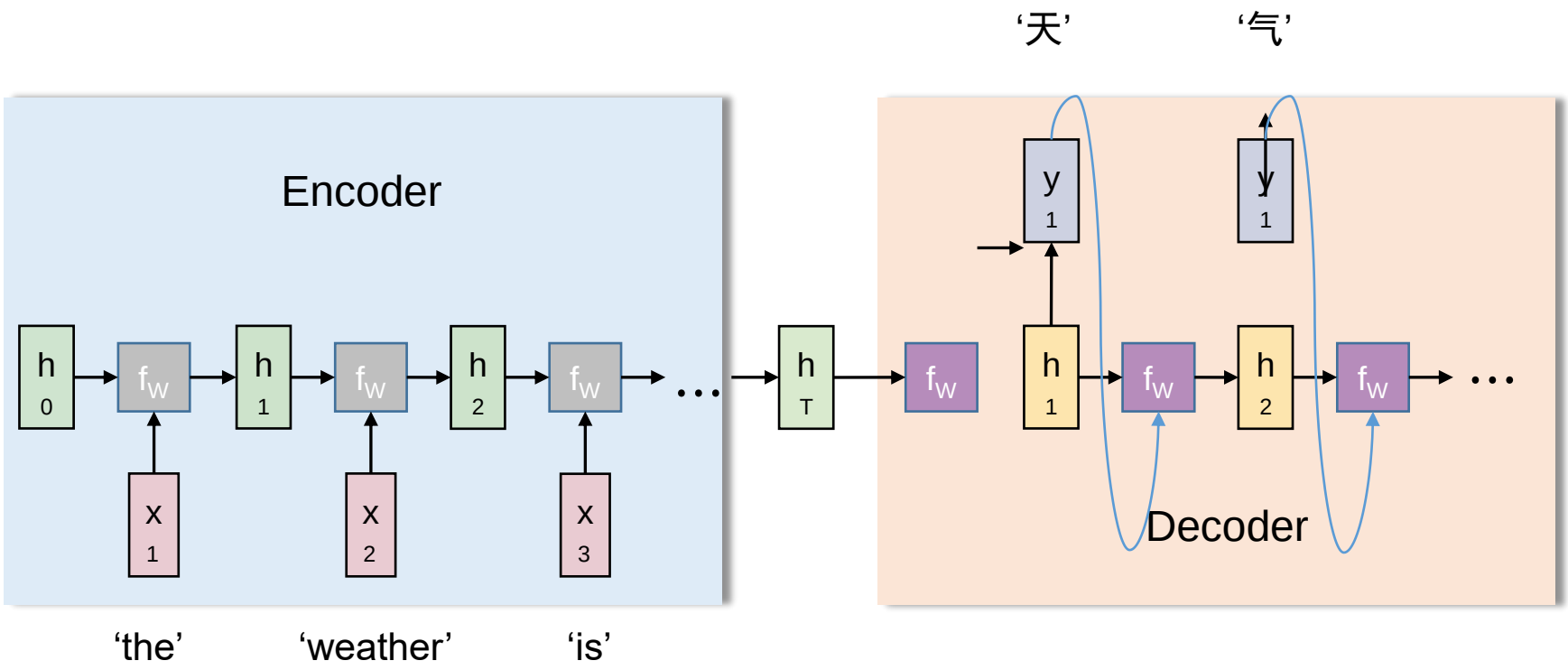
Applications: Sequence classification, sentiment analysis *etc.*

- One to many



Applications: Image captioning *etc.*

- Many-to-one + one-to-many



Applications: machine translation, sentence feature extraction *etc.*