Neural Networks

Qinliang Su （苏勤亮）

Sun Yat-sen University

suqliang@mail.sysu.edu.cn

Outline

- Neural Networks
- Backpropagation
- Typical Neural Networks
  - CNN
  - RNN
  - Transformer-based NN (BERT, GPT)
  - GNN

Motivation

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

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

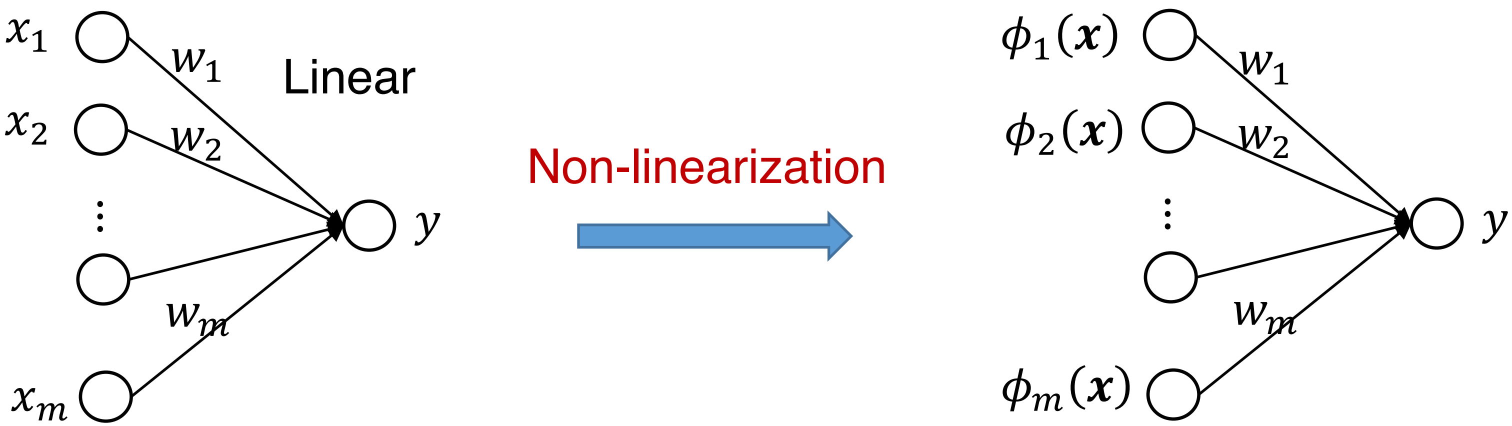
$$\boldsymbol{\phi}: \boldsymbol{x} \longrightarrow \boldsymbol{\phi}(\boldsymbol{x})$$

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

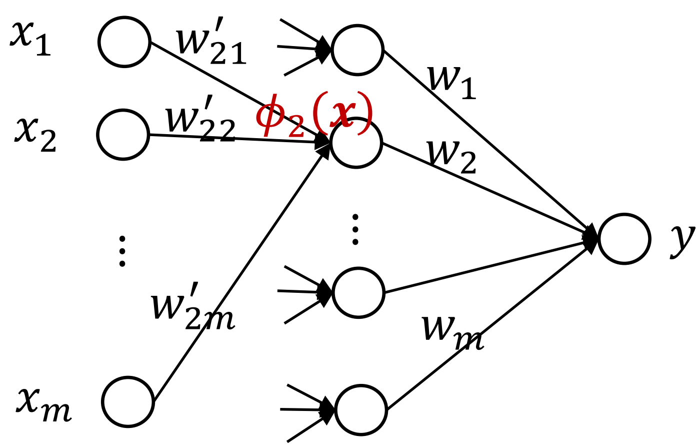
In both methods, the basis functions are fixed and *cannot adaptively adjust according to the characteristics of data*

Neural Networks

- The previous non-linearization process can be illustrated as



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



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

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

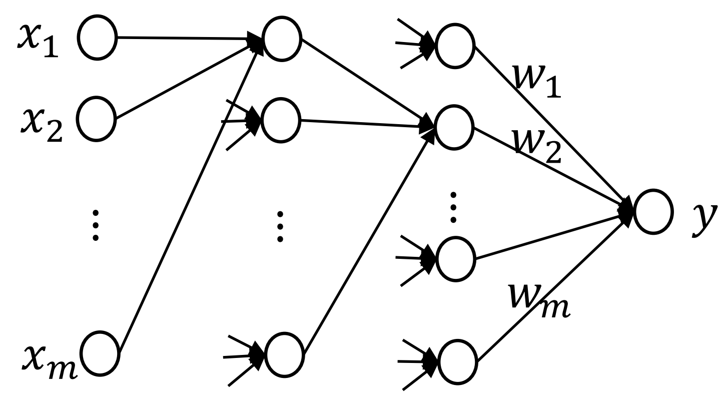
where $a(\cdot)$ is the activation function

$a(\cdot)$ cannot be dropped, otherwise the output y is in a linear relation with the input $\boldsymbol{x}$

- The output y can be concisely written as

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

- Further increase the number of layers

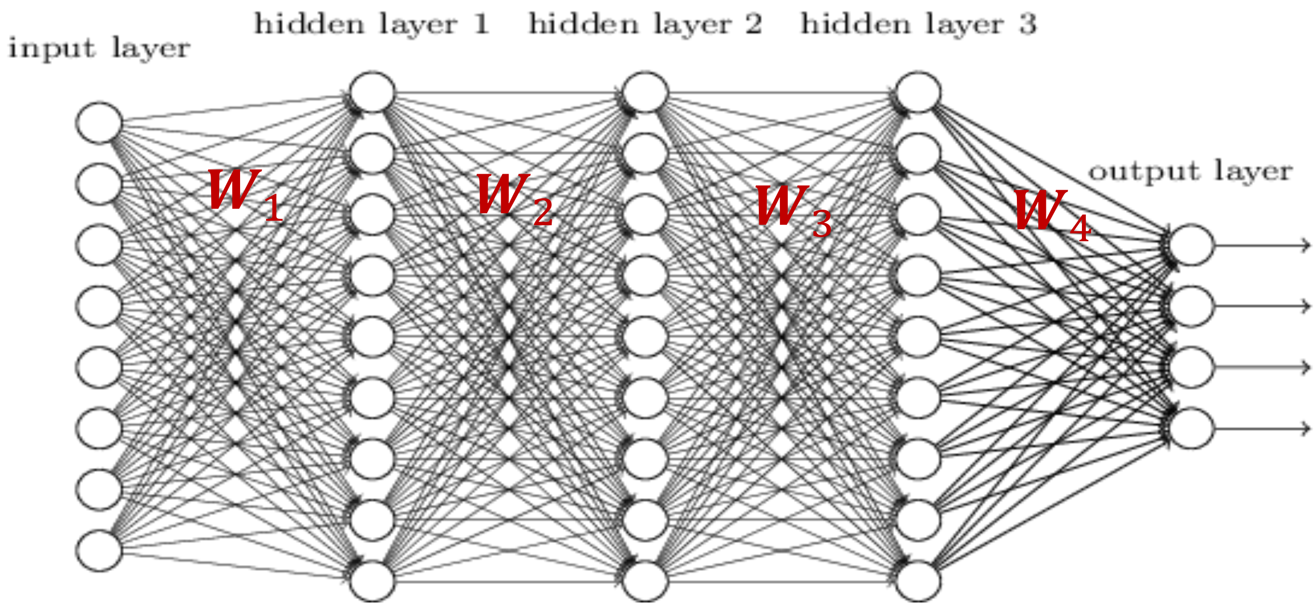


- In general, the output of a L-layer NN can be represented as

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

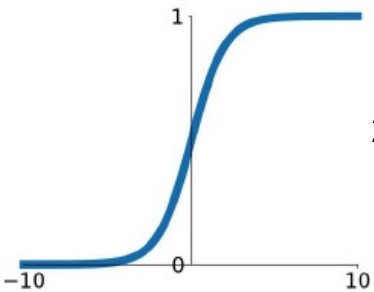
Classification: $\hat{y}(\boldsymbol{x}) = softmax\left(\boldsymbol{W}_L a\left(\cdots a\left(\boldsymbol{W}_2 a(\boldsymbol{W}_1 \boldsymbol{x})\right)\right)\right)$

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



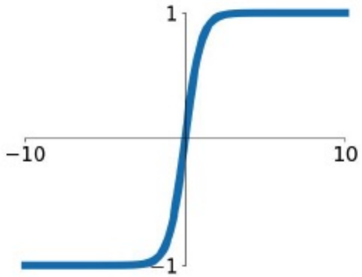
Activation functions

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



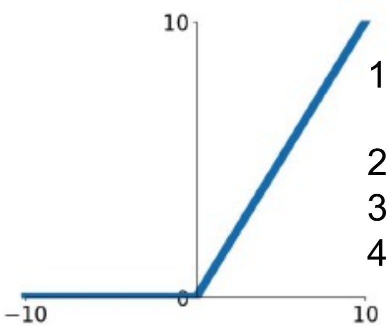
1) Vanishing gradient due to saturate
2) Output only positive values
3) $\exp(\cdot)$ is computationally expensive

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



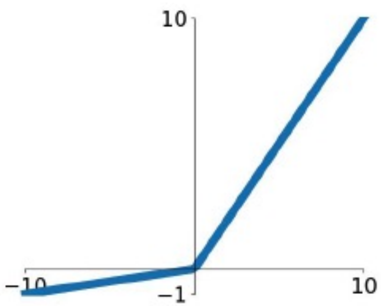
1) Vanishing gradient due to saturate
2) $\exp(\cdot)$ is computationally expensive
3) Output both positive and negative values

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:

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

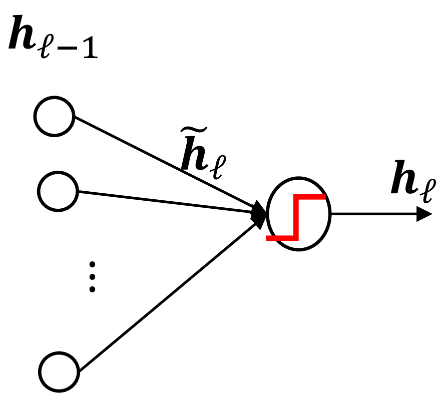
$$\mathcal{L}_c(\boldsymbol{\theta}) = -\frac{1}{N}\sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{D}}\sum_{k=1}^K y_k \log \hat{y}_k(\boldsymbol{x})$$

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

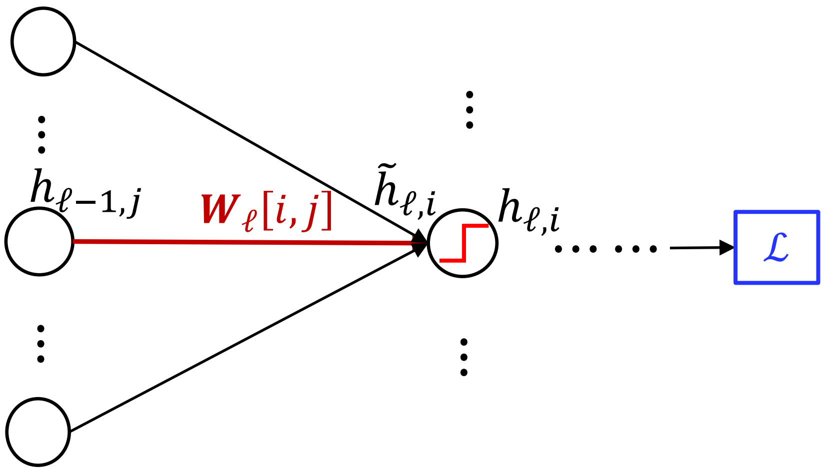


$$\tilde{\boldsymbol{h}}_\ell = \boldsymbol{W}_\ell \boldsymbol{h}_{\ell-1}, \qquad \boldsymbol{h}_\ell = a\left(\tilde{\boldsymbol{h}}_\ell\right), \qquad \boldsymbol{h}_\ell = a(\boldsymbol{W}_\ell \boldsymbol{h}_{\ell-1})$$

➢ The output layer

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

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



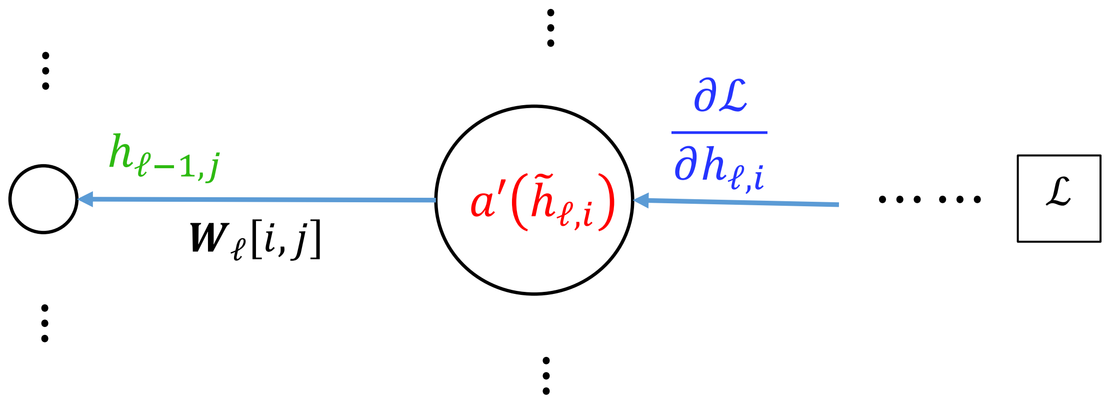
$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}_\ell[i,j]} = \frac{\partial \mathcal{L}}{\partial h_{\ell,i}} \cdot \boxed{\frac{\partial h_{\ell,i}}{\partial \boldsymbol{W}_\ell[i,j]}}$$

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

$$\boxed{= \boxed{\frac{\partial \mathcal{L}}{\partial h_{\ell,i}}} \cdot a'\left(\tilde{h}_{\ell,i}\right) \cdot h_{\ell-1,j}}$$

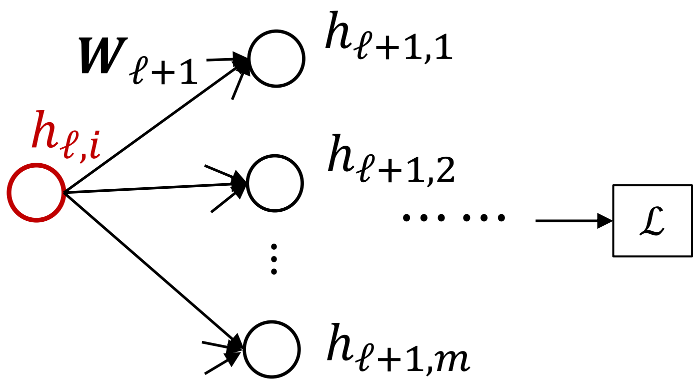
Schematic diagram showing the structure of the derivative

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



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, $\frac{\partial \mathcal{L}}{\partial h_{\ell,i}}$



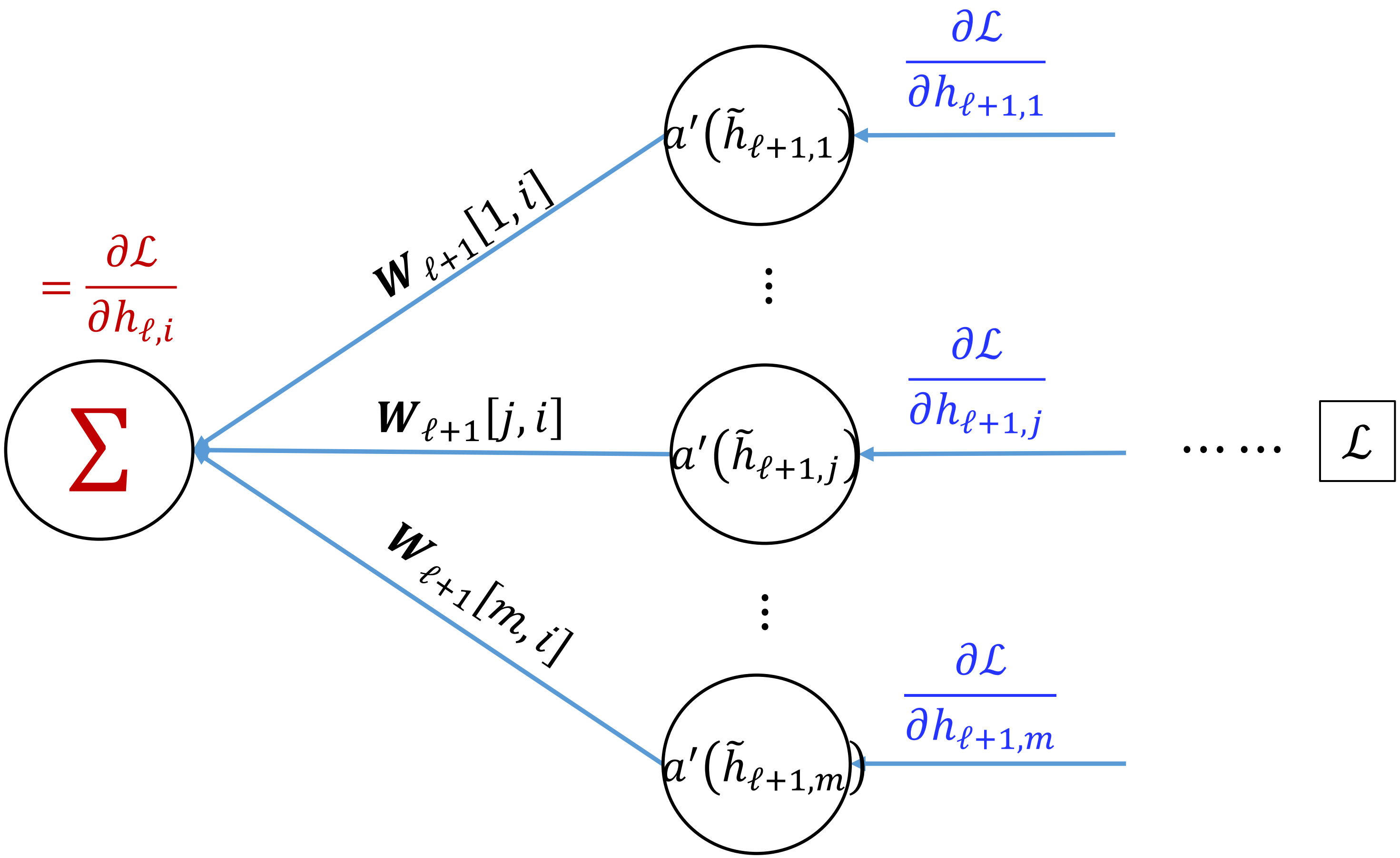
$$\frac{\partial \mathcal{L}}{\partial h_{\ell,i}} = \sum_{j=1}^{m} \left[\frac{\partial \mathcal{L}}{\partial h_{\ell+1,j}} \cdot \boxed{\frac{\partial h_{\ell+1,j}}{\partial h_{\ell,i}}} \right]$$

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

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

➢ The recursive structure in the expression

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



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

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

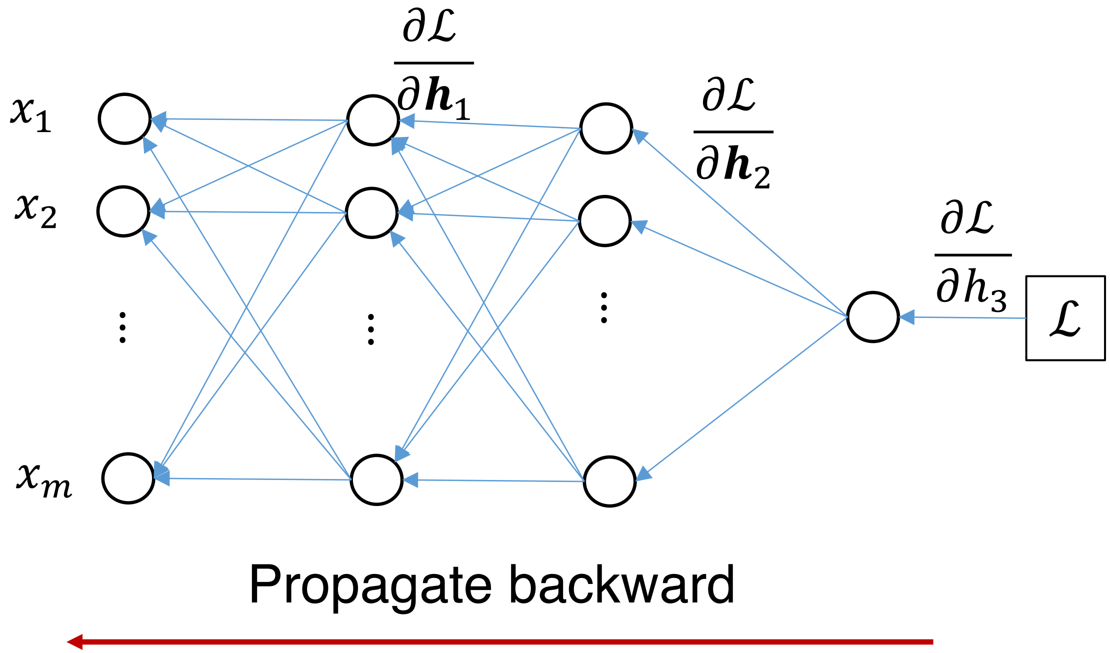
$$\frac{\partial \mathcal{L}}{\partial h_L} = (h_L - y)$$

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

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

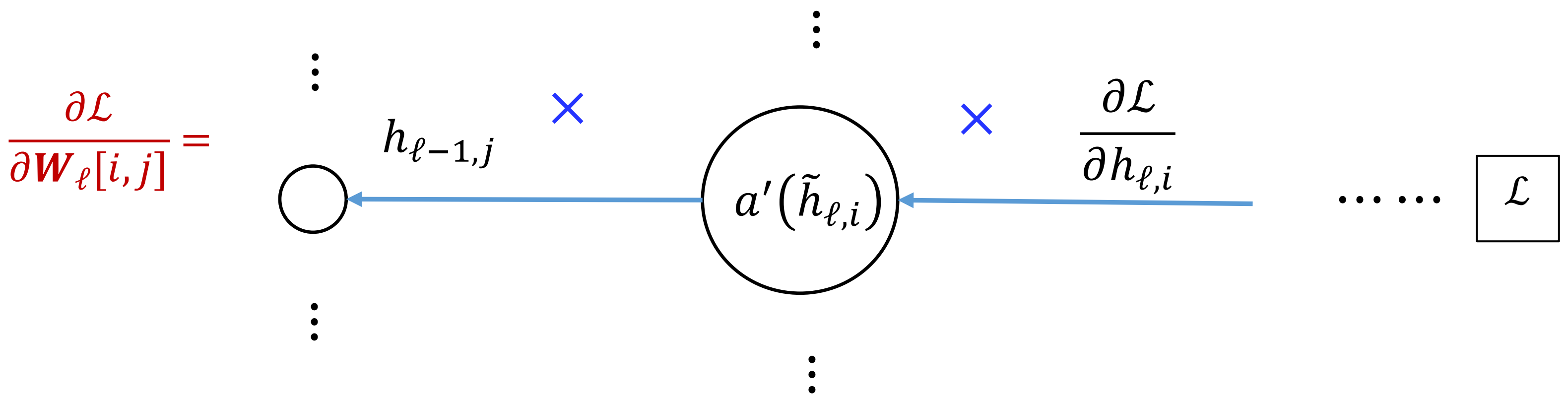
The Whole Computation Graph of Gradient

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



$$\boxed{\frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_\ell} = \boldsymbol{W}_{\ell+1}^T \left(\tilde{\boldsymbol{g}}_{\ell+1} \odot \frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_{\ell+1}} \right)}$$ where $$\tilde{\boldsymbol{g}}_{\ell+1} = \left[\frac{\partial h_{\ell+1,1}}{\partial \tilde{h}_{\ell+1,1}}, \cdots, \frac{\partial h_{\ell+1,m}}{\partial \tilde{h}_{\ell+1,m}} \right]^T$$

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



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

Written in matrix form as

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}_\ell} = \left(\tilde{\boldsymbol{g}}_\ell \odot \frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_\ell}\right) \boldsymbol{h}_{\ell-1}^T$$

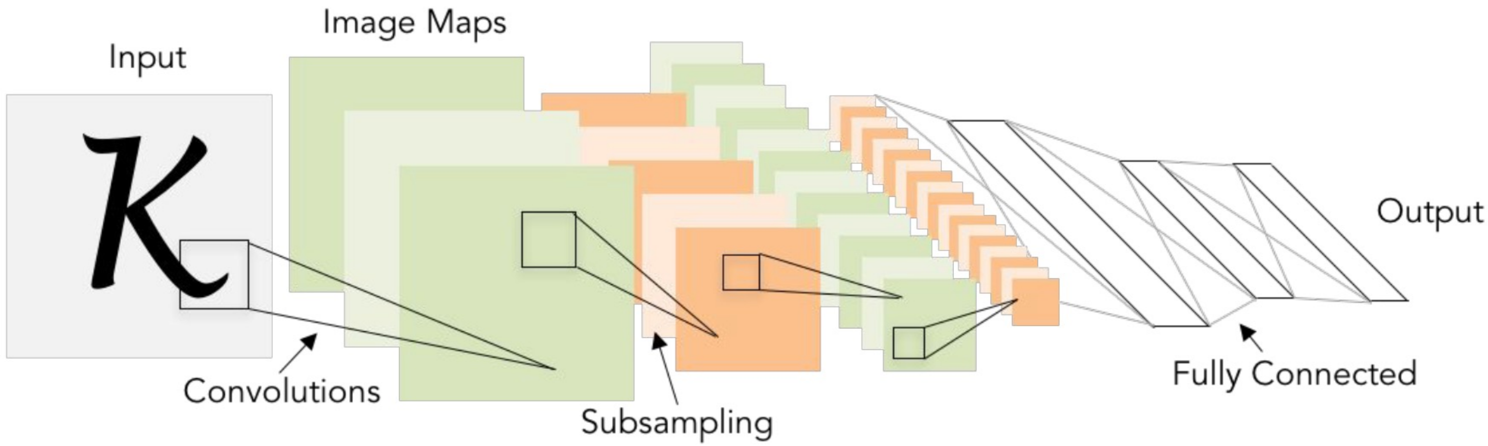
Outline

- Neural Networks
- Backpropagation
- Typical Neural Networks
  - CNN
  - RNN
  - Transformer-based NN (BERT, GPT)
  - GNN

Convolutional Neural Networks (CNN)

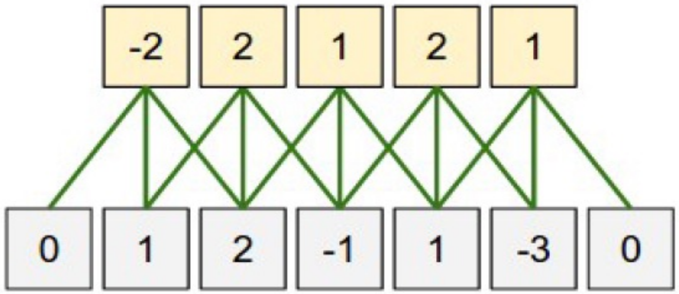
- Limitations of fully connected NNs (also called multilayer perception (MLP))
  1) The number of parameters are huge, prone to result in overfitting
  2) They never consider/exploit any structural information of data

- CNNs are proposed to leverage the spatial structures (*e.g.* translational invariance) in the image/visual data

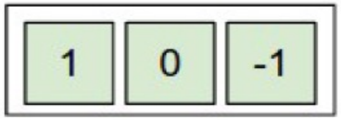


Convolution Operations

- One-dimensional convolution operation illustration



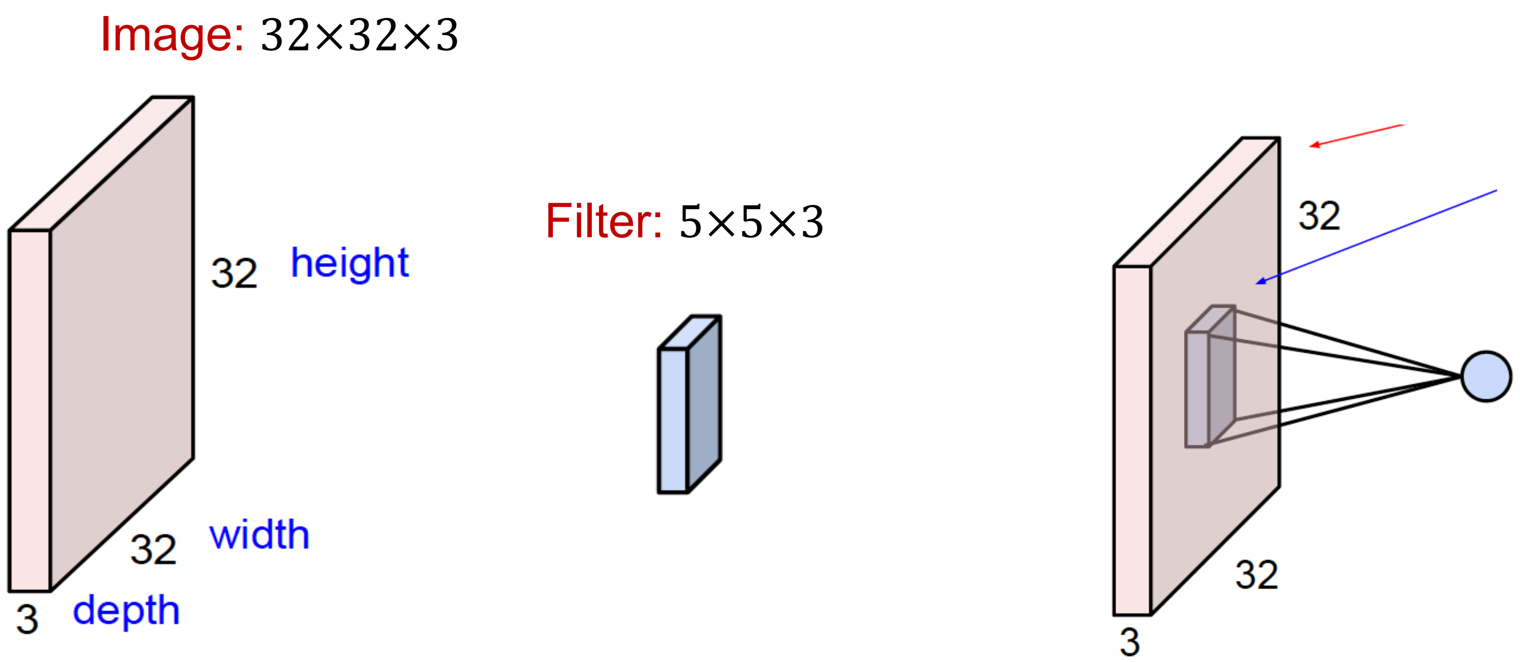
Neural Network



The Filter

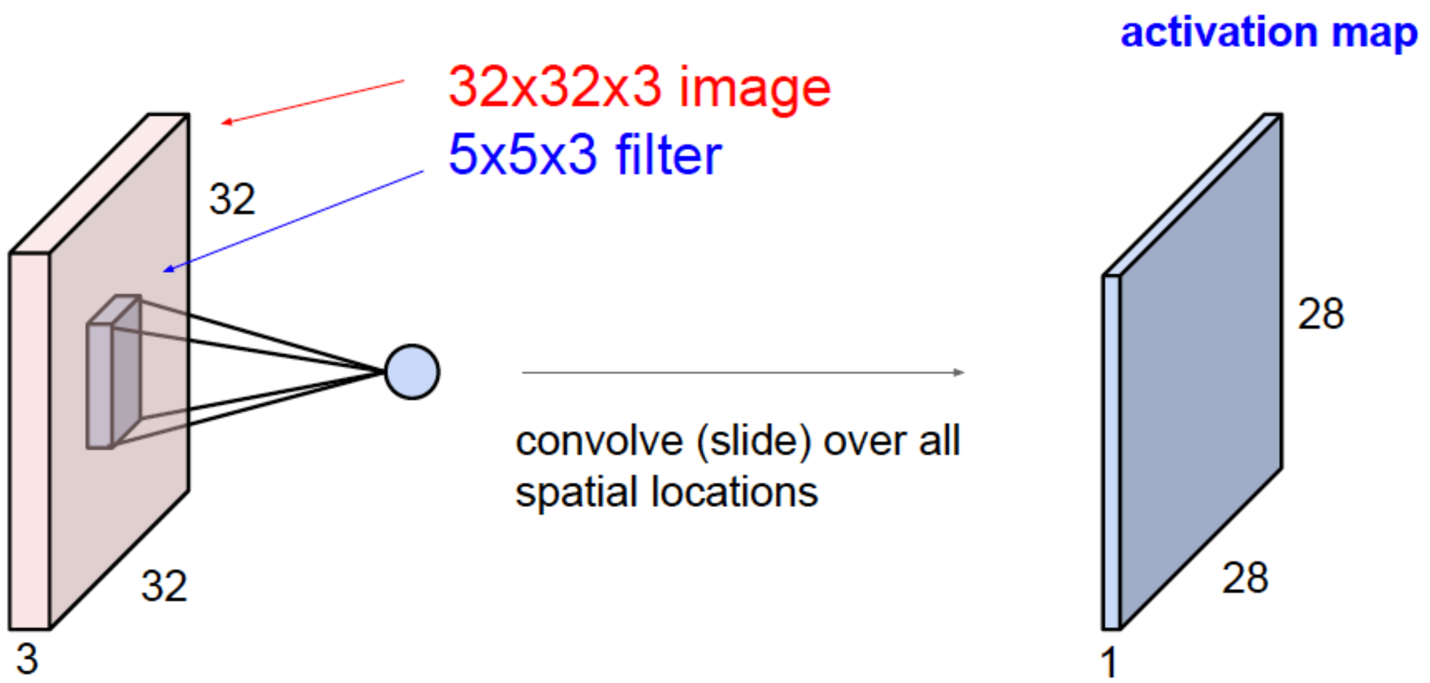
- Distinctions from the MLPs
  - Local connectivity: each neuron is only connected to a fraction of neurons in previous layer
  - Parameter sharing: the parameters for different neurons are tied

- Two-dimensional convolution operation illustration

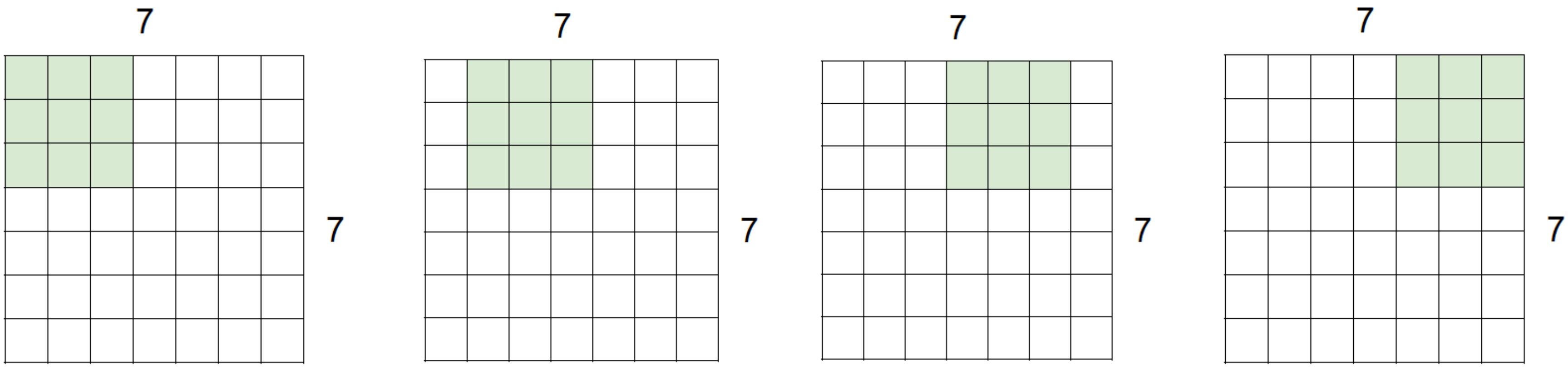


Feature Maps

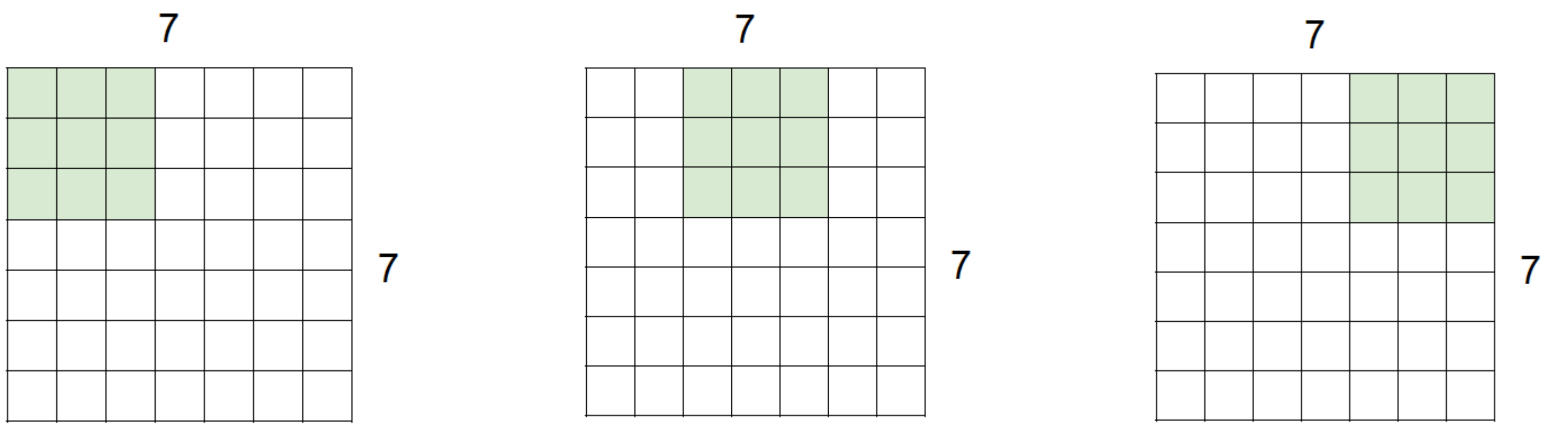
- Sliding the filter along the two dimensions of images, yielding an activation map with dimension $28 \times 28 \times 1$



A closer look at the convolution operation

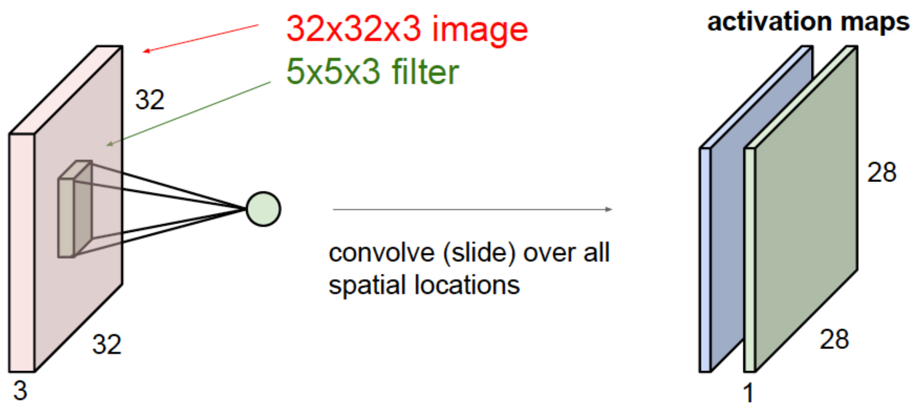


Stride=1

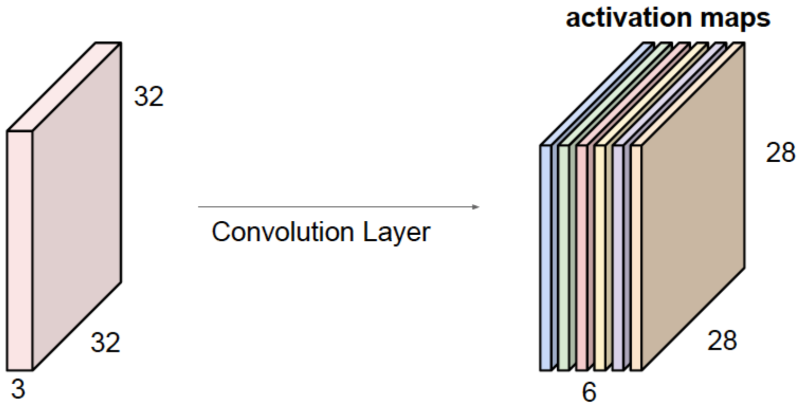


Stride=2

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



- Convolving with multiple filters



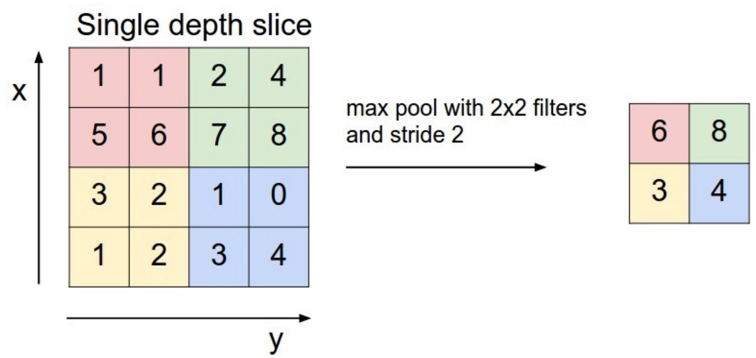
- Why are multiple filters needed?

Different filters are responsible to recognize different patterns of an image

Original | Sharpen | Edge Detect | Strong Edge Detect

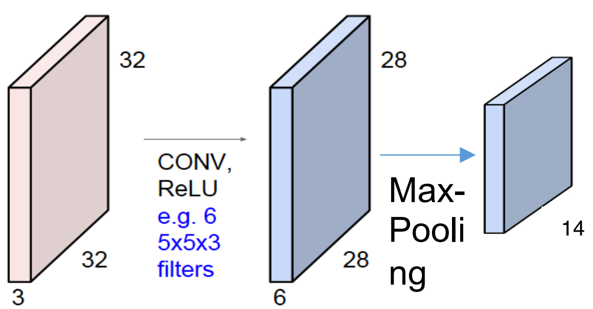
Pooling

- Illustration



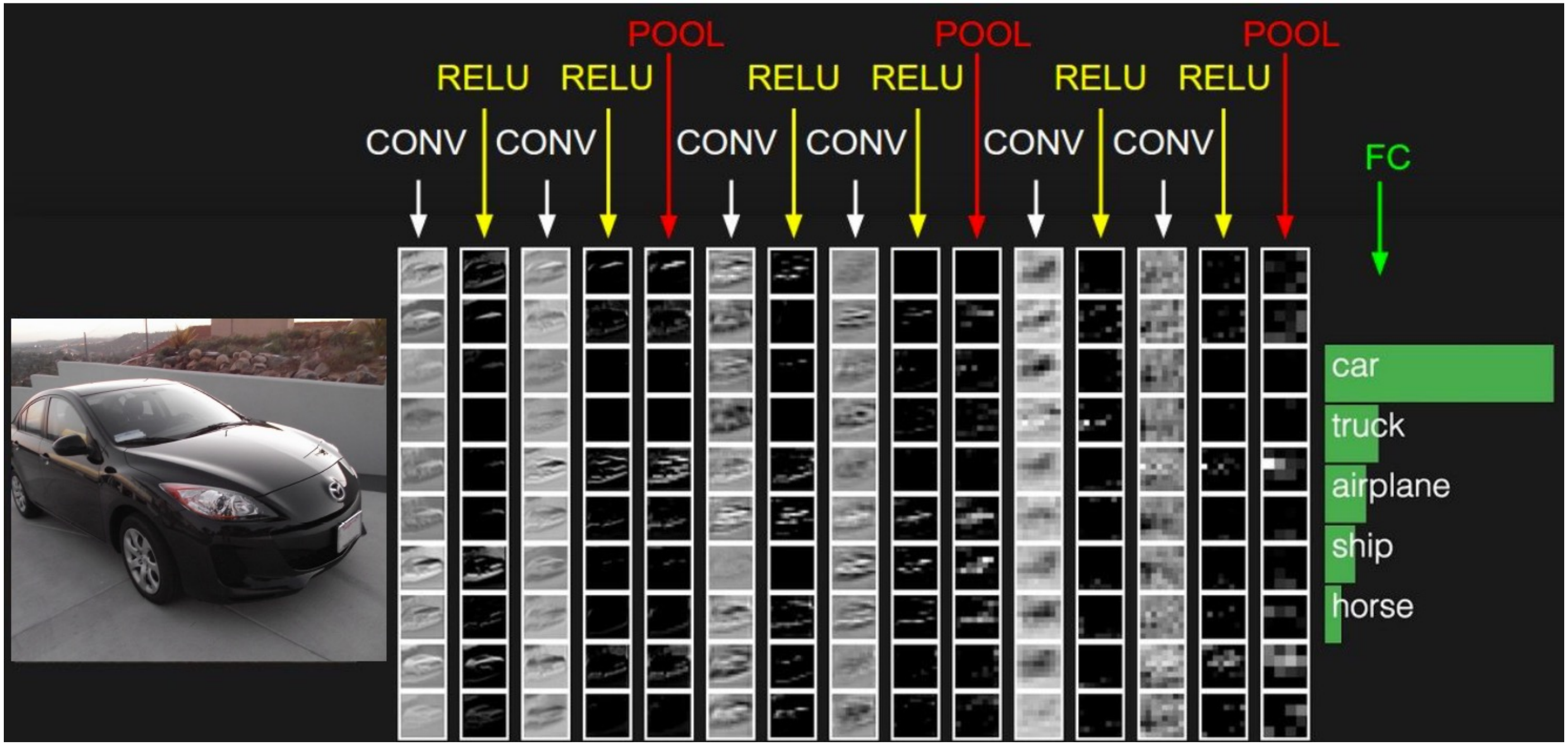
1) Reducing the dimensionality
2) Introducing nonlinearities
3) Maintaining some extent of spatial invariance
4) The pooling is not necessary at all layer

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



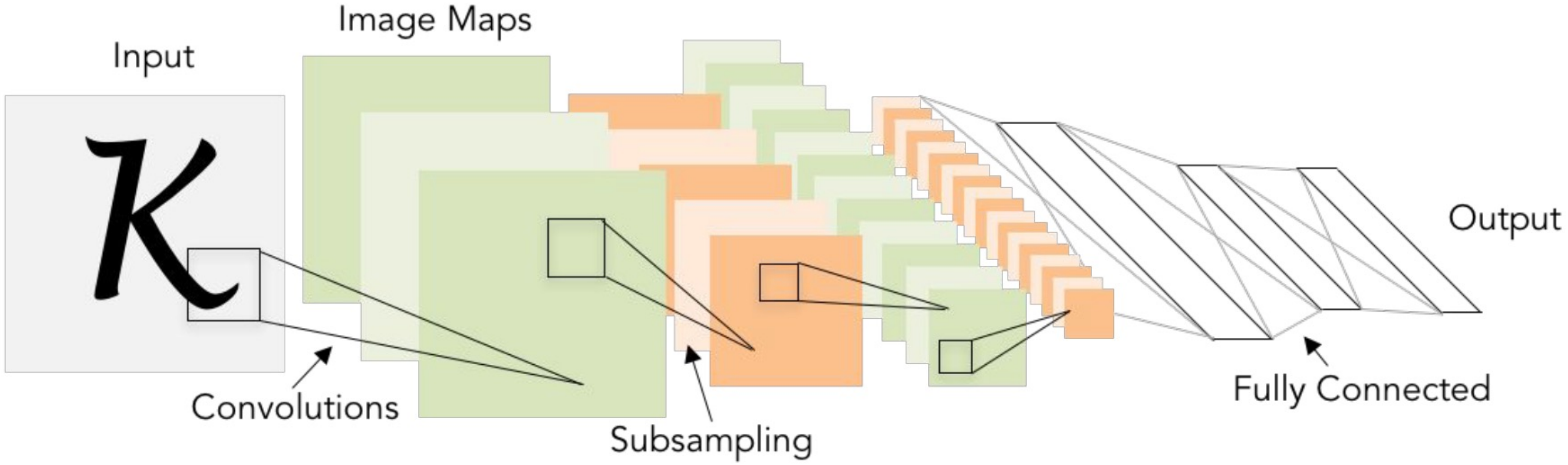
Illustration

- An illustration of image classification with CNN



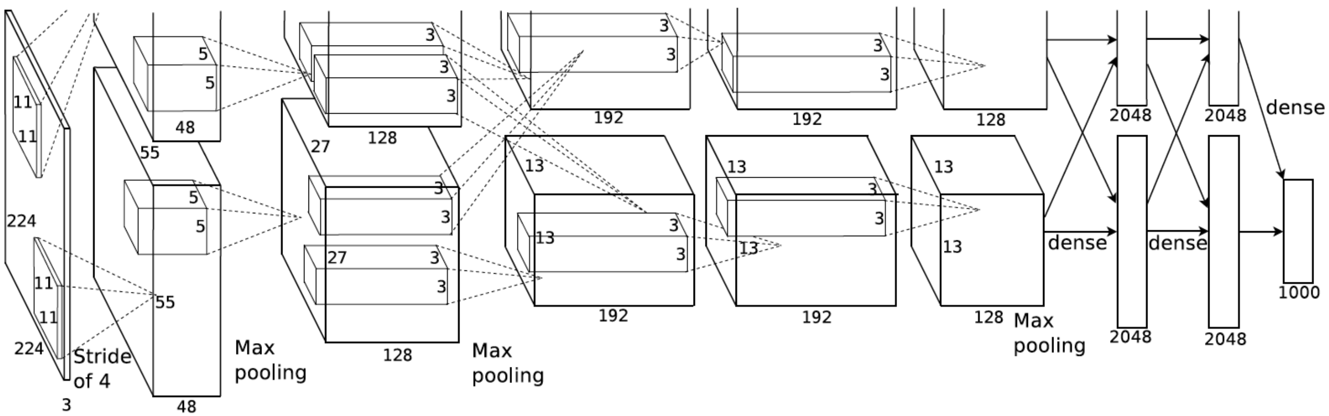
Several Typical CNNs

- LeNet



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

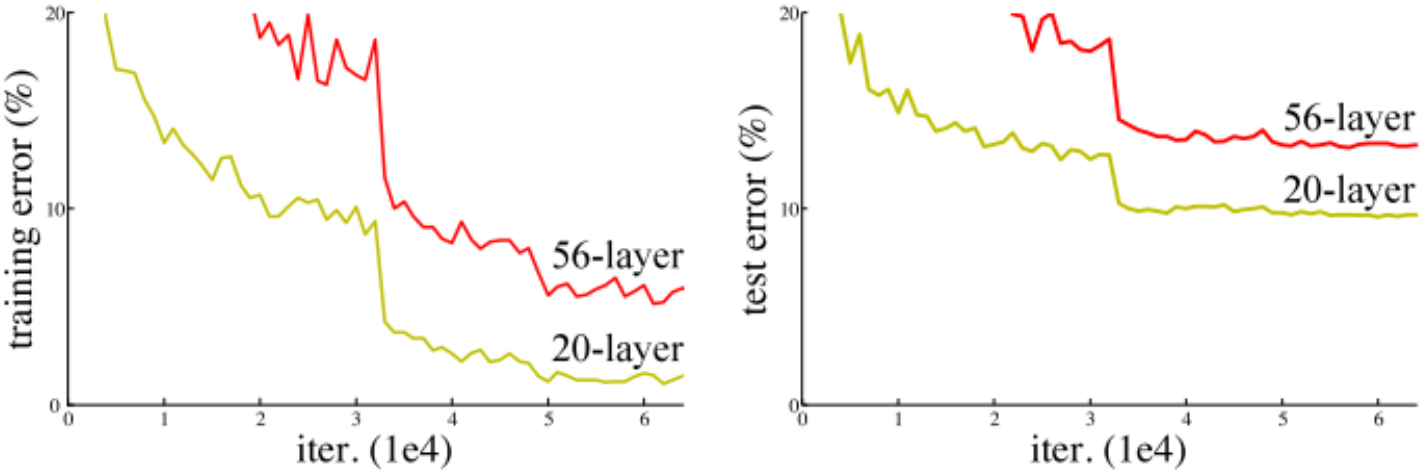
- AlexNet



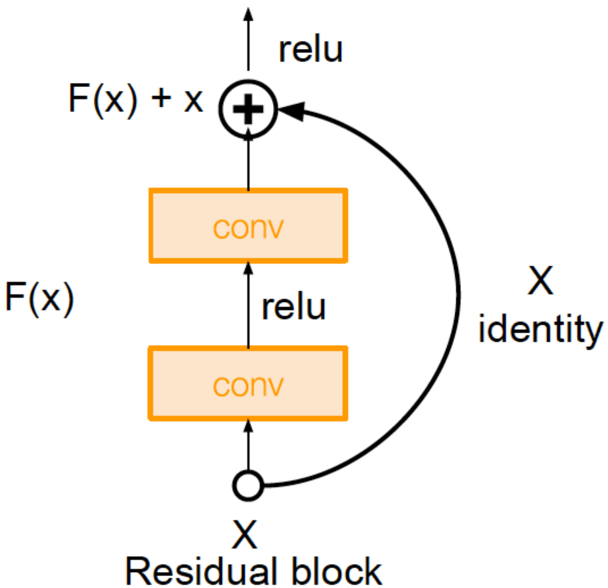
Input	Conv1	MaxPool1	Conv2	MaxPool2	Conv3
$227\times227\times3$	$55\times55\times96$	$27\times27\times96$	$27\times27\times256$	$13\times13\times256$	$13\times13\times384$

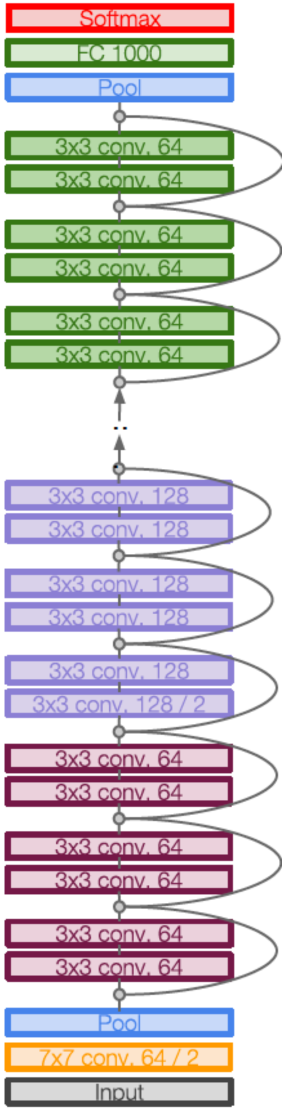
Conv4	Conv5	MaxPool3	FC6	FC7	FC8
$13\times13\times384$	$13\times13\times256$	$6\times6\times256$	4096	4096	1000

- ResNet
  - Simply stacking layers does not lead to performance gains. On the contrary, it may hurt the performance



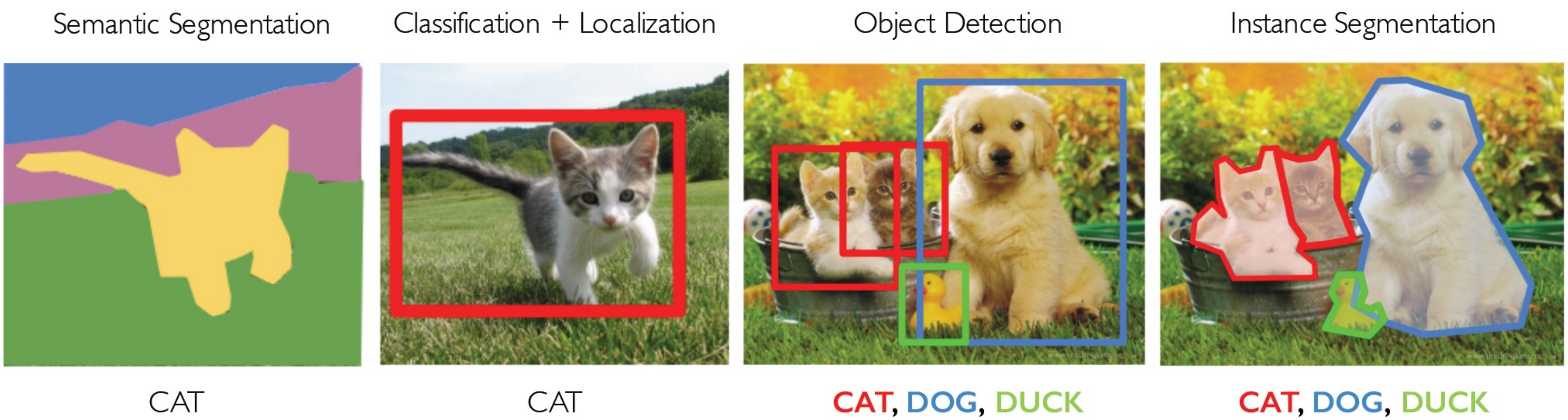
  - One of the reasons is that deeper models are more difficult to train
  - ResNet alleviates the training difficulties by introducing a ‘shortcut’ connection between layers





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

Typical Applications beyond Classification



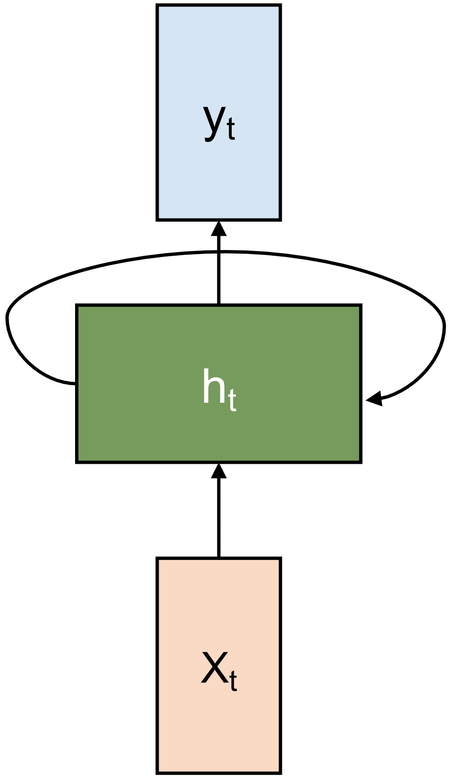
Vanilla RNNs

- 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

$$y_t = f_o(h_t)$$

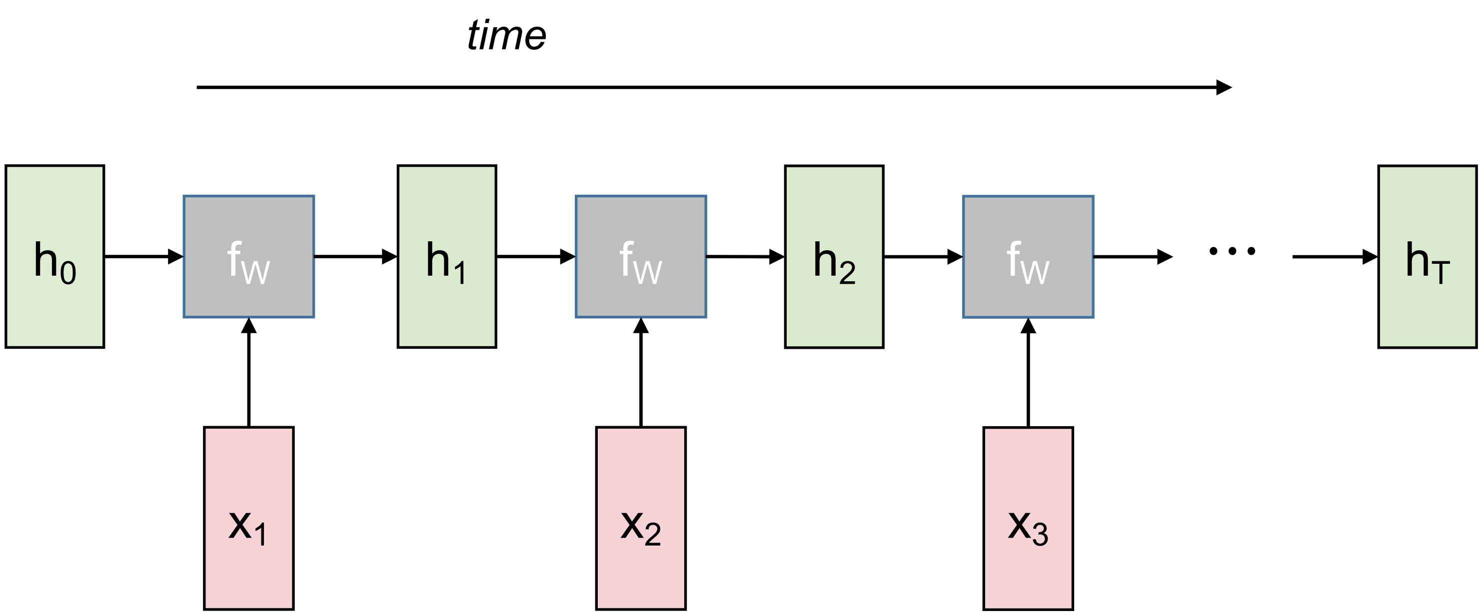


- For instance, $f_h(\cdot)$ and $f_o(\cdot)$ could be

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

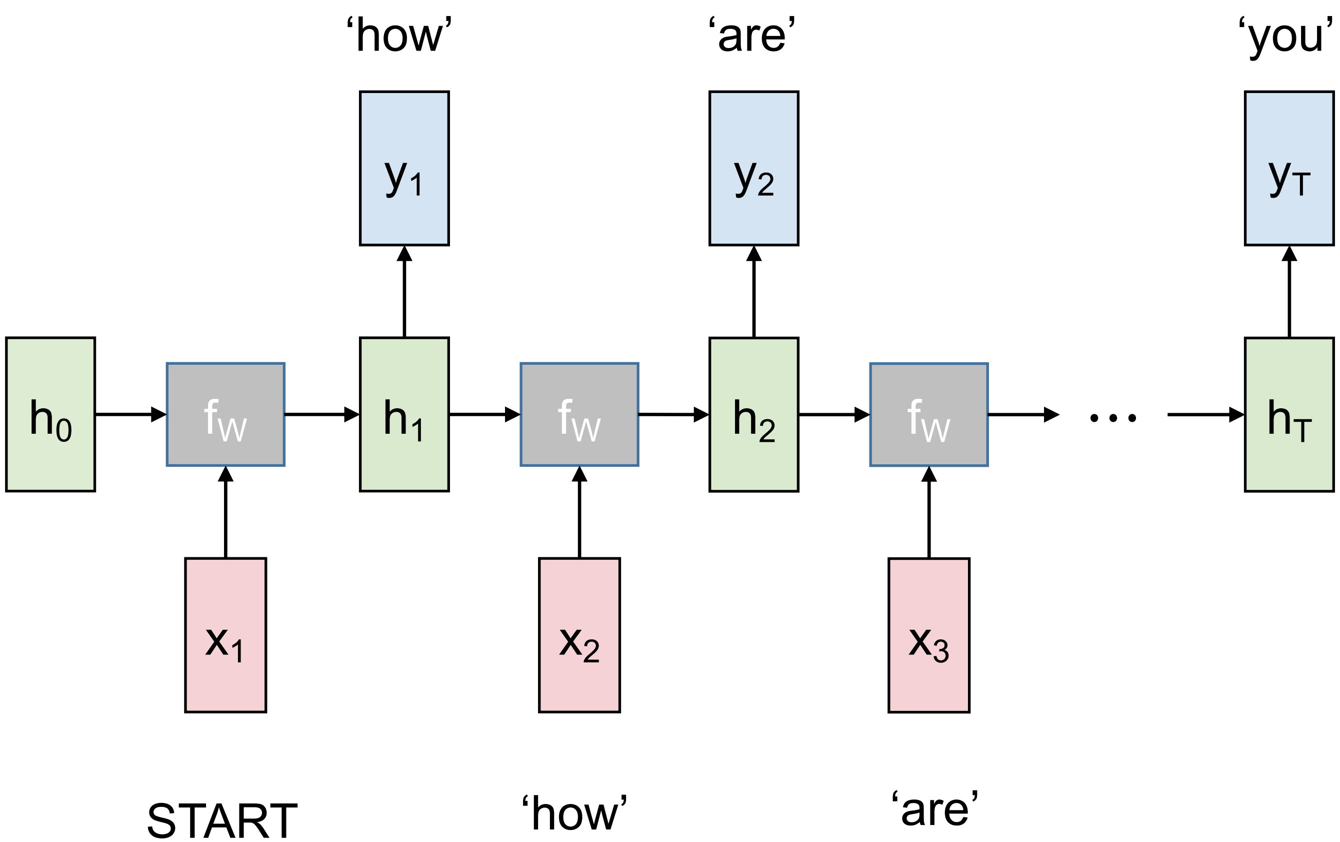
$$y_t = W_{hy}h_t \quad \text{or} \quad y_t = \mathrm{softmax}(W_{hy}h_t)$$

- Unfolding the RNN over time



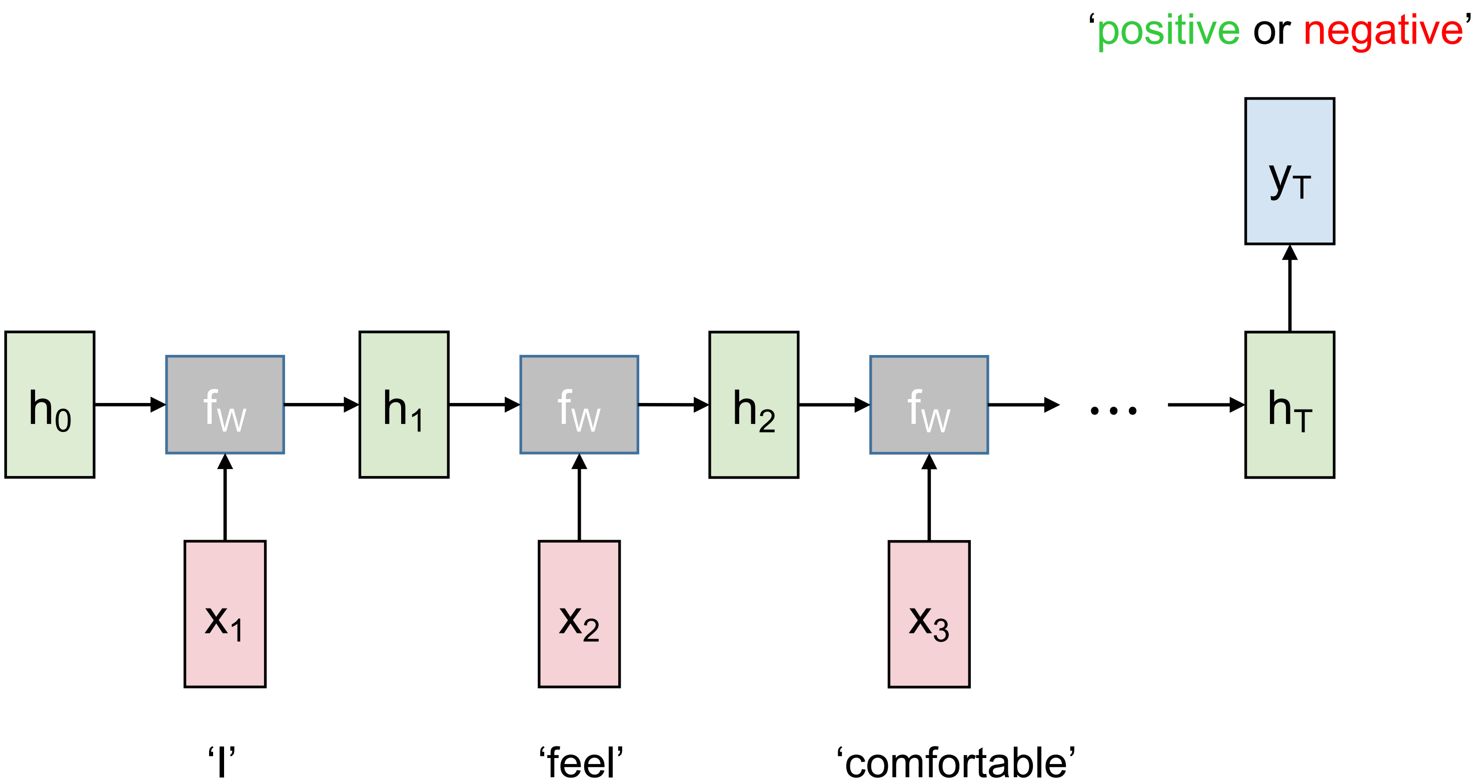
Output Types

- Many to Many



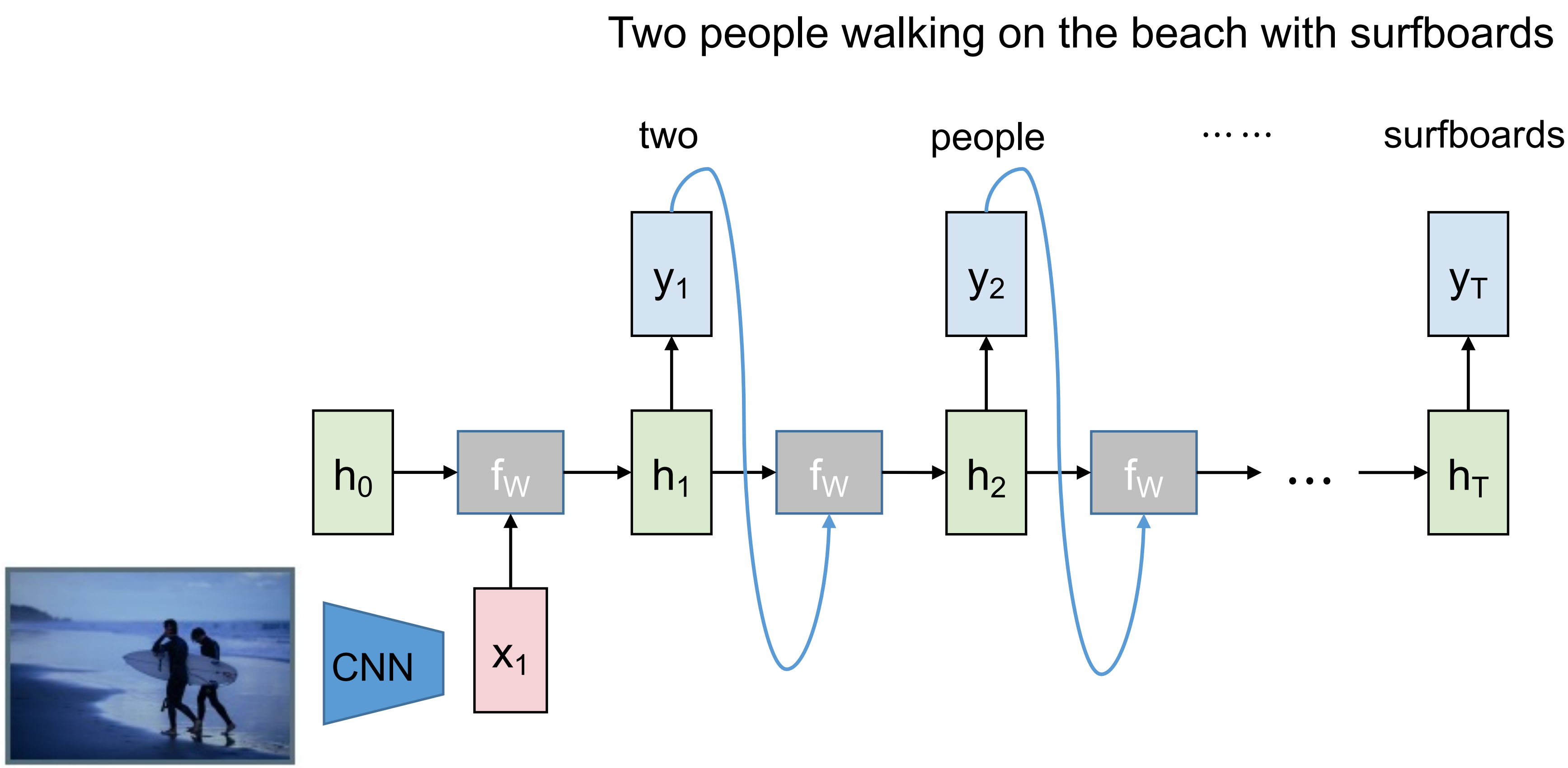
Applications: Language model, frame classification in video, *etc.*

- Many to one



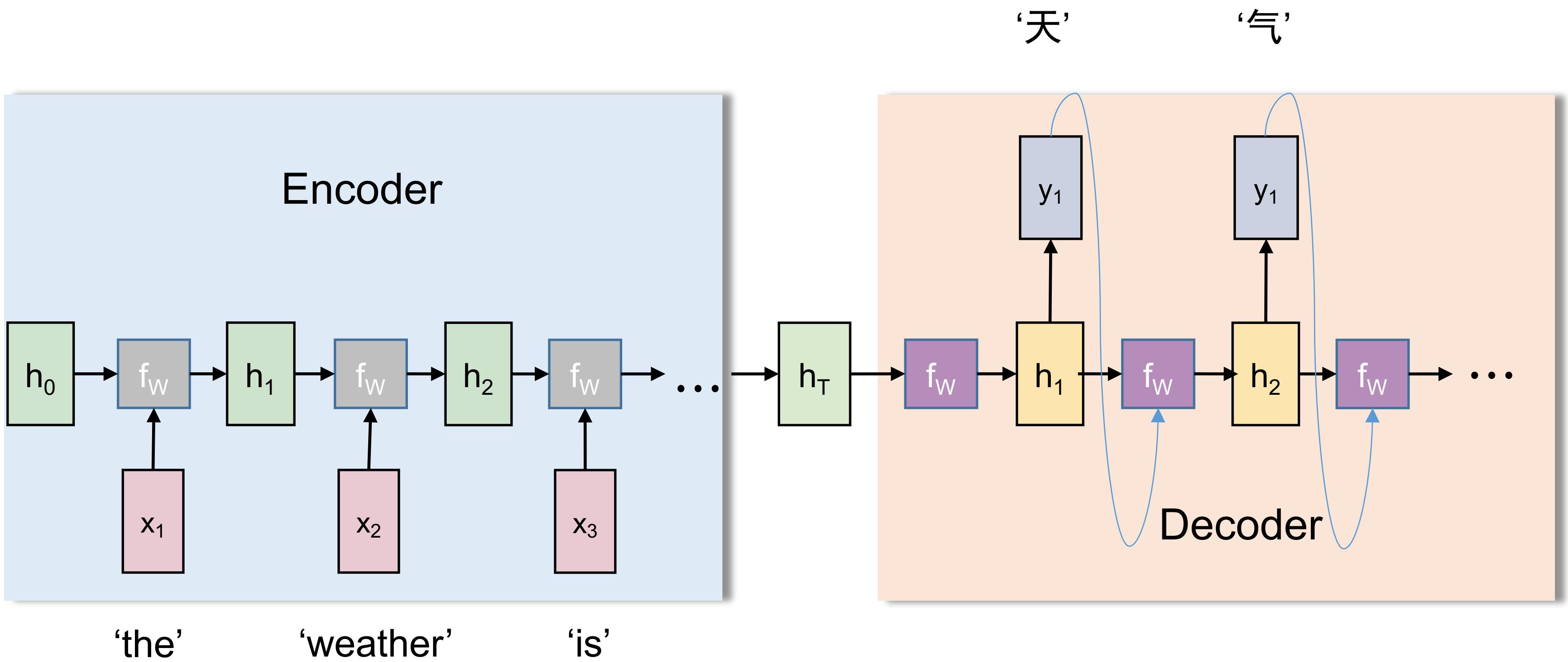
Applications: Sequence classification, sentiment analysis *etc.*

- One to many



Applications: Image captioning *etc.*

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



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

- LSTM

➢ To increase the capability of memorizing distant content, LSTM doesn't update the state as $h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$

➢ Instead, it *introduces an additional cell* C_t to remember the RNN state at time step t, and update the hidden states as

1) $i_t = \sigma(W_i[x_t, h_{t-1}])$

2) $f_t = \sigma(W_f[x_t, h_{t-1}])$

3) $o_t = \sigma(W_o[x_t, h_{t-1}])$

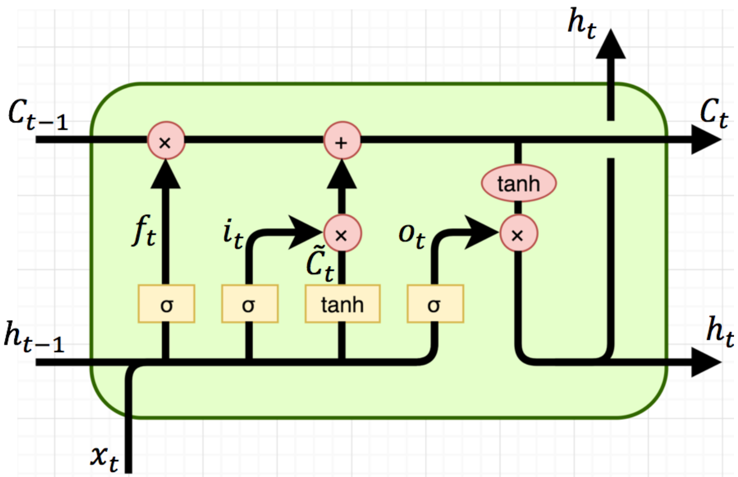
Gates

4) $\tilde{C}_t = \tanh(W_g[x_t, h_{t-1}])$

5) $C_t = \sigma(f_t * C_{t-1} + i_t * \tilde{C}_t)$

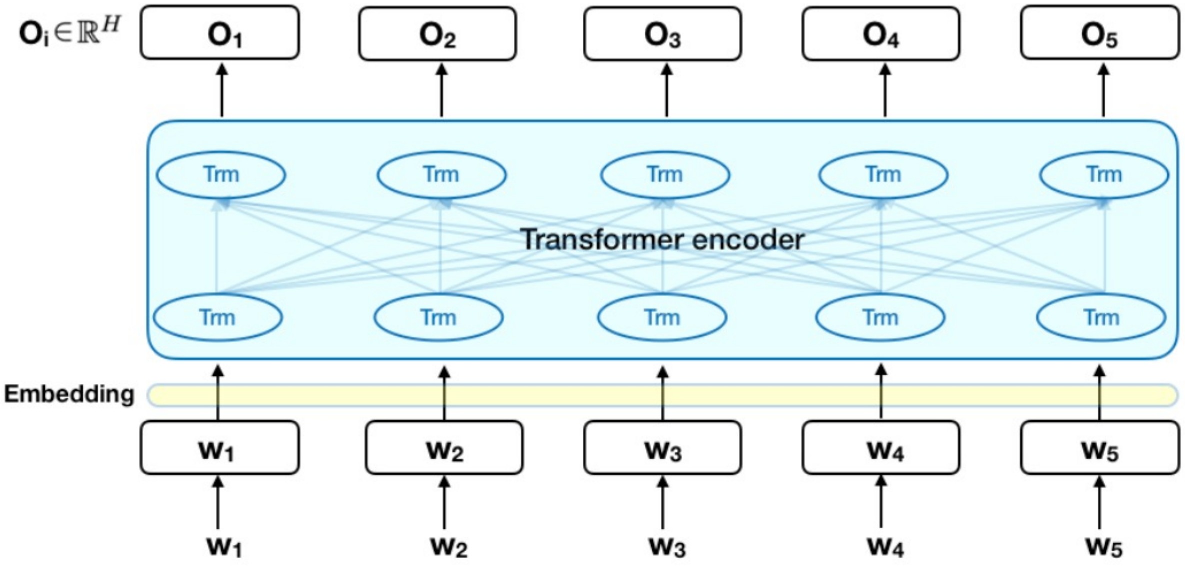
6) $h_t = \tanh(C_t) * o_t$

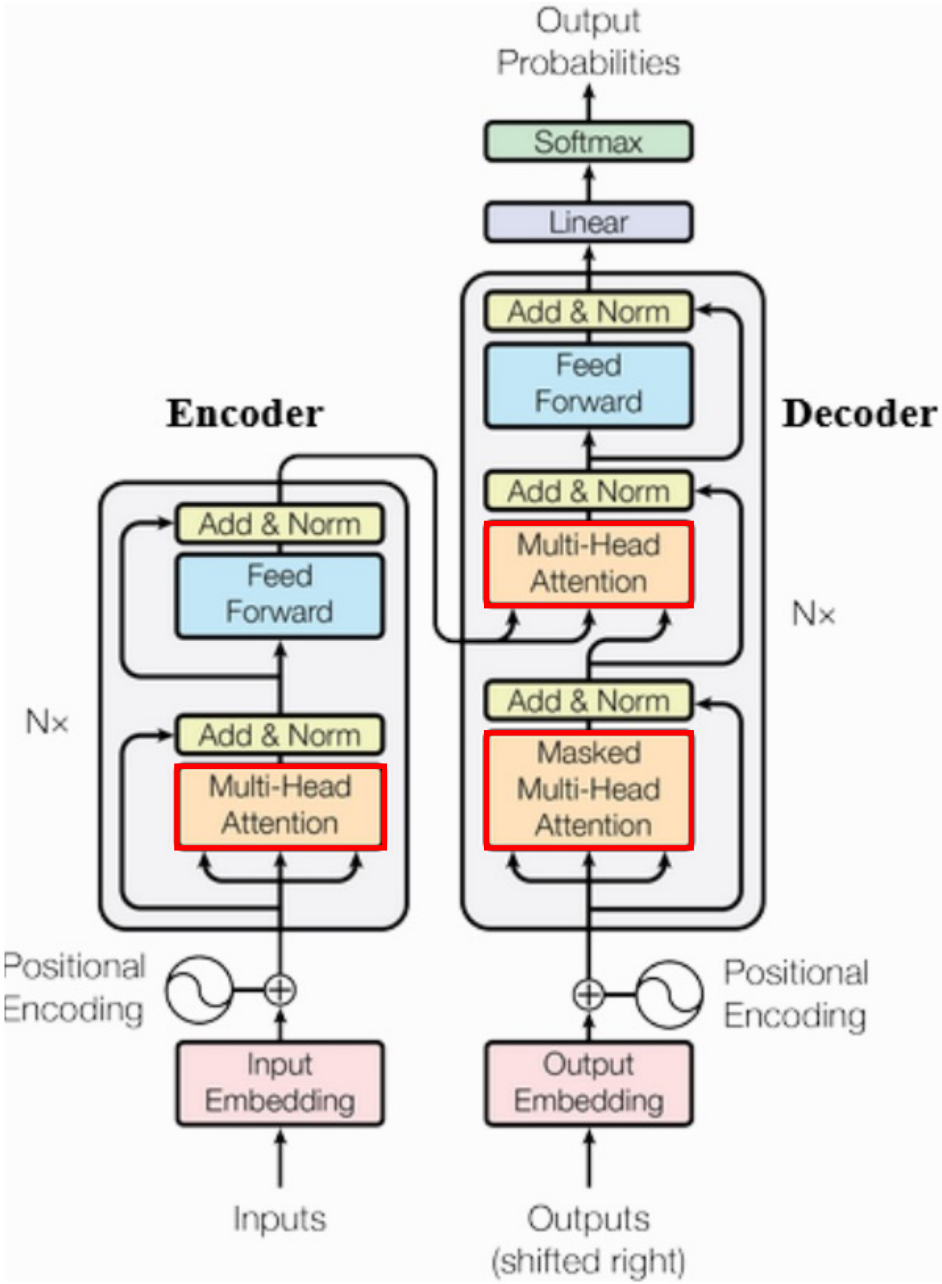
Outputs



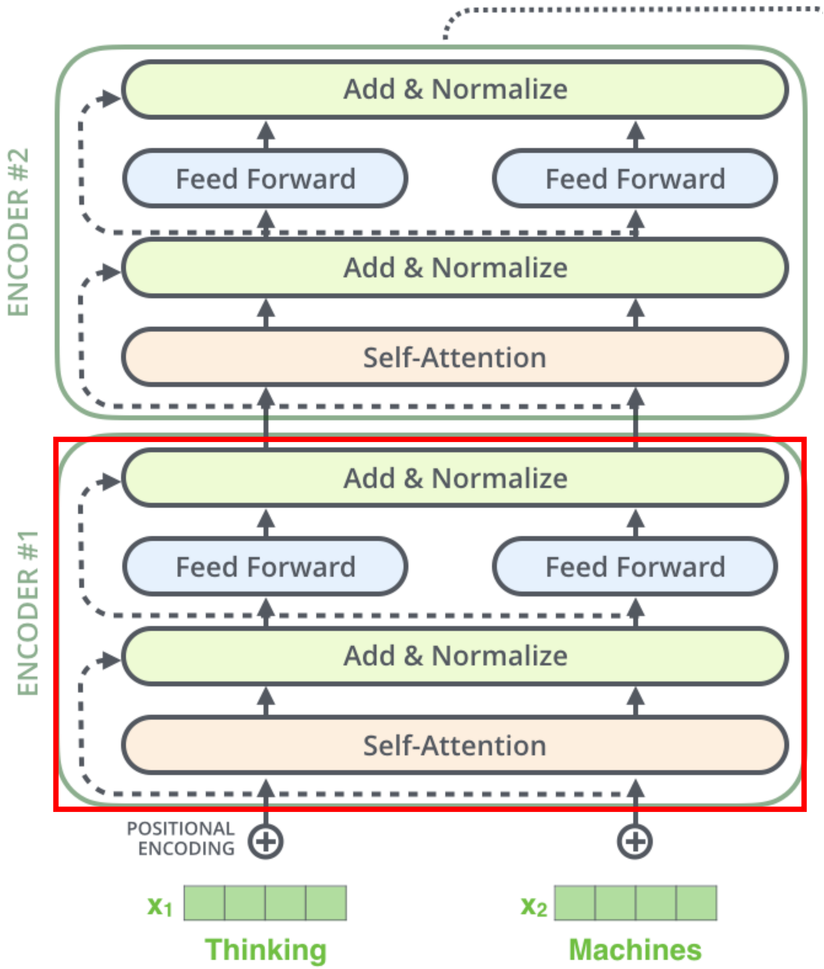
Transformer-based Neural Networks

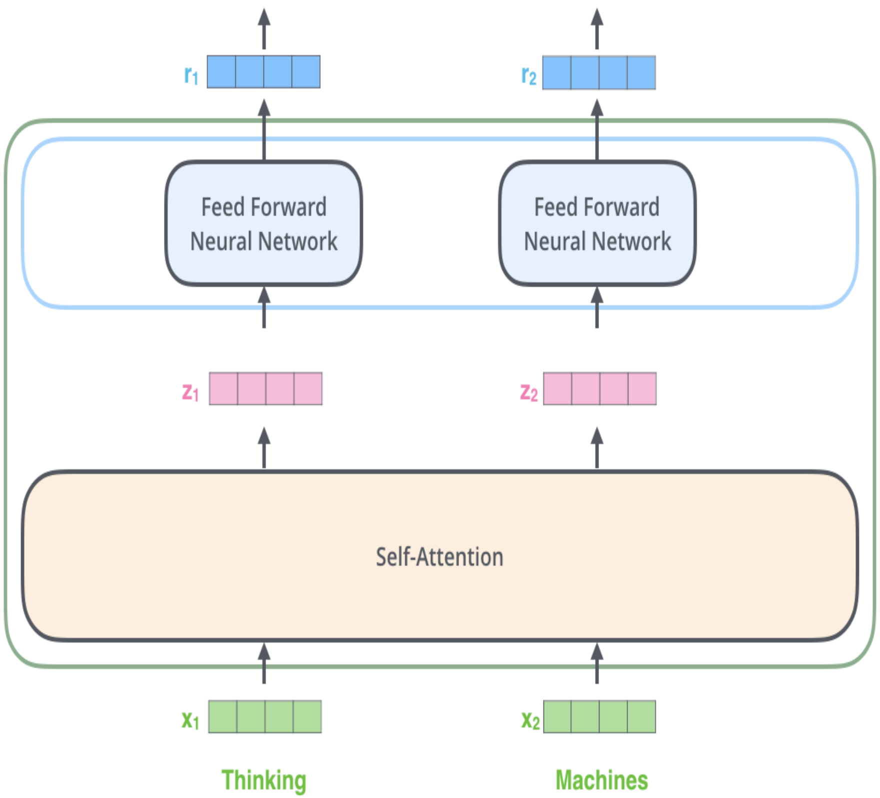
- Problems with RNN
  1) Difficult to capture long-term dependencies in documents
  2) Only allow sequential executions, unable to exploit the *parallel computation resources* in GPUs
- Bidirectional Encoder Representations from Transformers (BERT) and the GPT models adopts a transformer-based ‘fully-connected’ structure



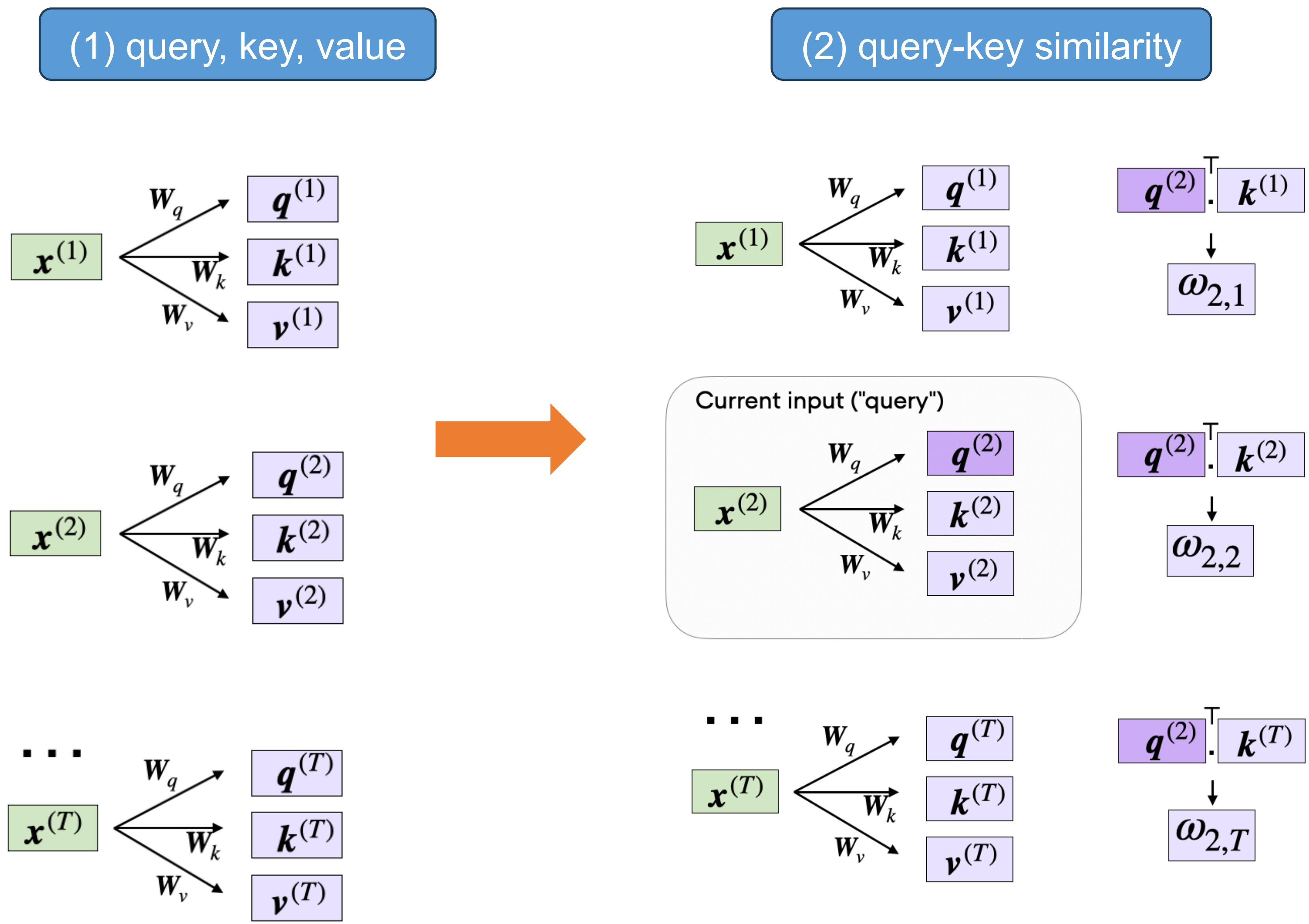


Output
Probabilities
Softmax
Linear
Add & Norm
Feed
Forward
Encoder
Decoder
Add & Norm
Add & Norm
Multi-Head
Attention
Feed
Forward
Nx
Add & Norm
Nx
Add & Norm
Multi-Head
Attention
Masked
Multi-Head
Attention
Positional
Encoding
Positional
Encoding
Input
Embedding
Output
Embedding
Inputs
Outputs
(shifted right)

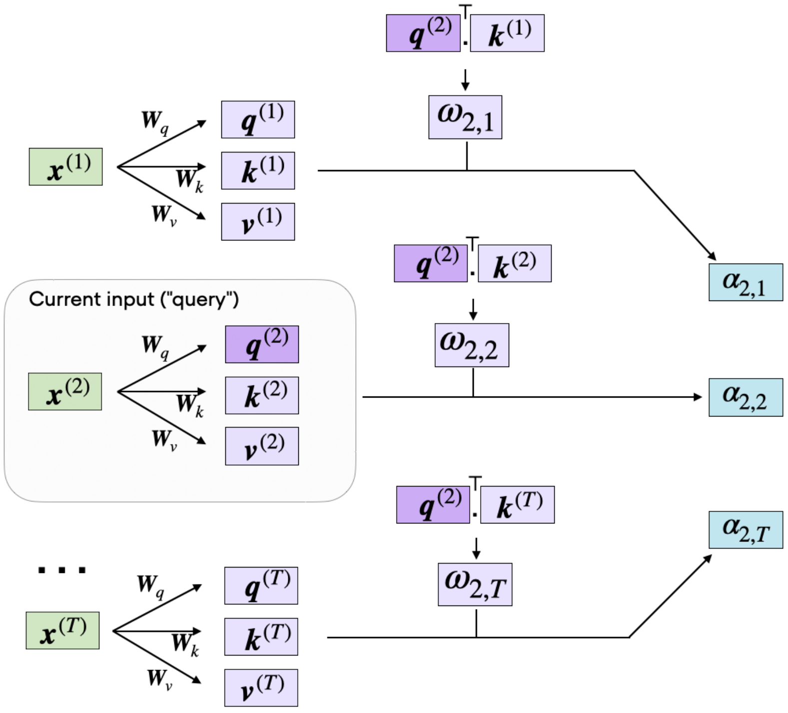

Add & Normalize
Feed Forward
Feed Forward
Add & Normalize
Self-Attention
ENCODER #2
Add & Normalize
Feed Forward
Feed Forward
Add & Normalize
Self-Attention
ENCODER #1
POSITIONAL ENCODING
x1
x2
Thinking
Machines


r1
r2
Feed Forward Neural Network
Feed Forward Neural Network
z1
z2
Self-Attention
x1
x2
Thinking
Machines

- Self-attention mechanism



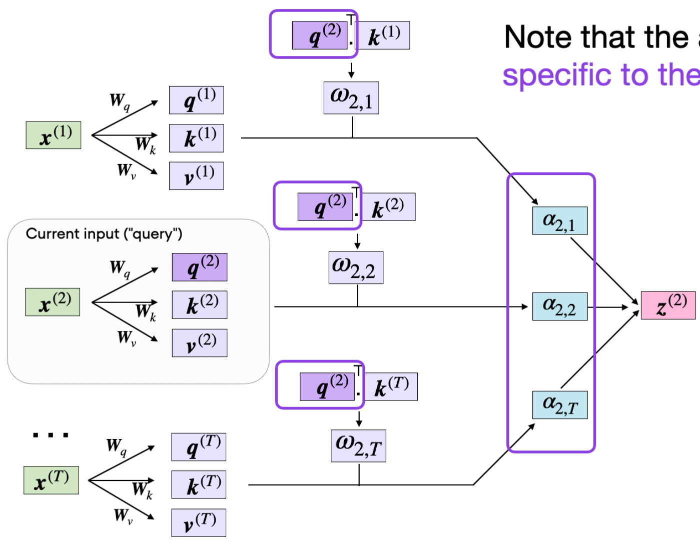
(3) attention weigh



where $\alpha_{2,i} = \text{softmax}\left(\frac{\omega_{2,i}}{\sqrt{d_k}}\right)$

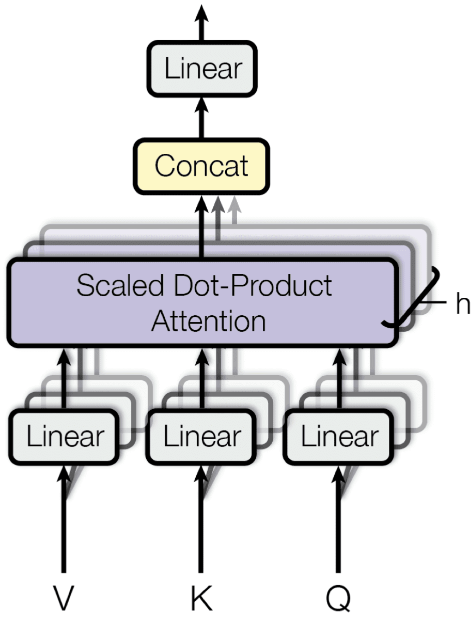
(4) output

Note that the attention scores are specific to the current input token

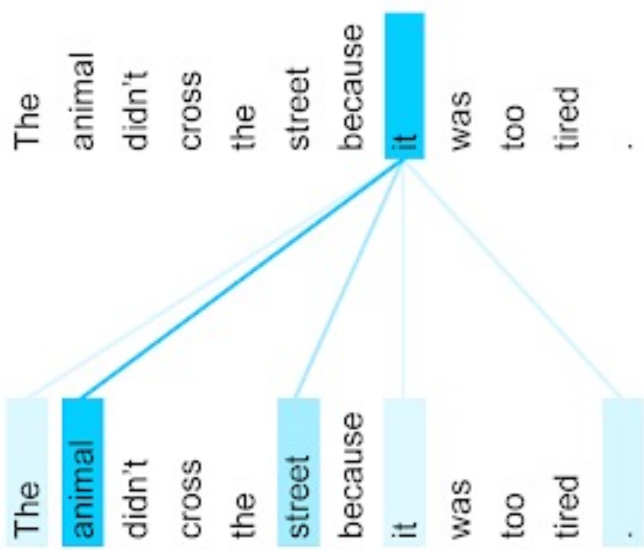


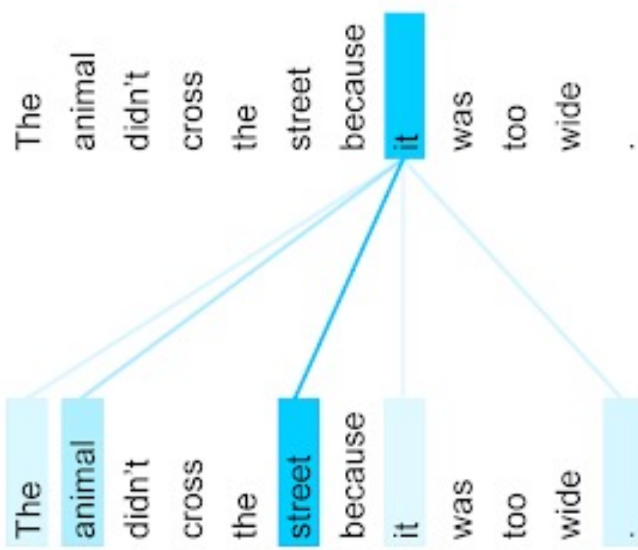
where $z^{(2)} = \sum_{j=1}^{T} \boxed{\alpha_{2,j}} v^{(j)}$

(5) multi-head attention

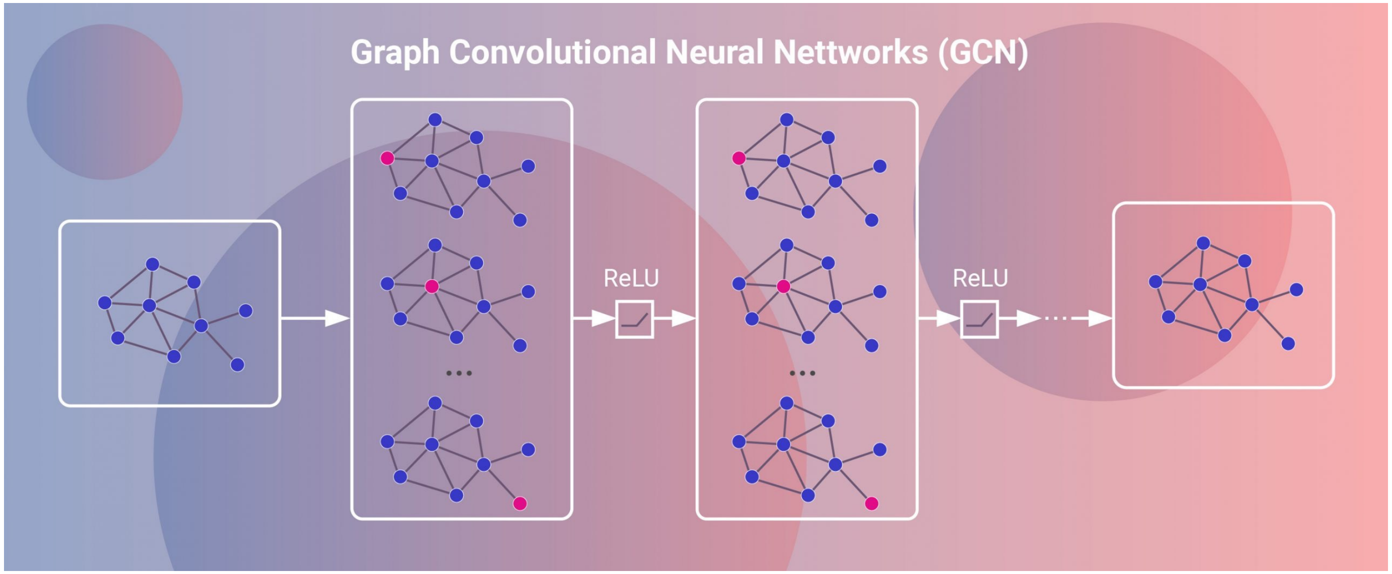


Each head is responsible for capturing different meanings





Graph Neural Networks



The hidden representation of i-th node at the $(\ell+1)$-th layer can be expressed as

$$\boldsymbol{h}_i^{\ell+1} = \sigma\left(\boldsymbol{W}_1^{\ell}\boldsymbol{h}_i^{\ell} + \sum_{j\in\mathcal{N}_i} \boldsymbol{W}_2^{\ell}\boldsymbol{h}_j^{\ell}\right)$$