



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 $\phi(\cdot)$

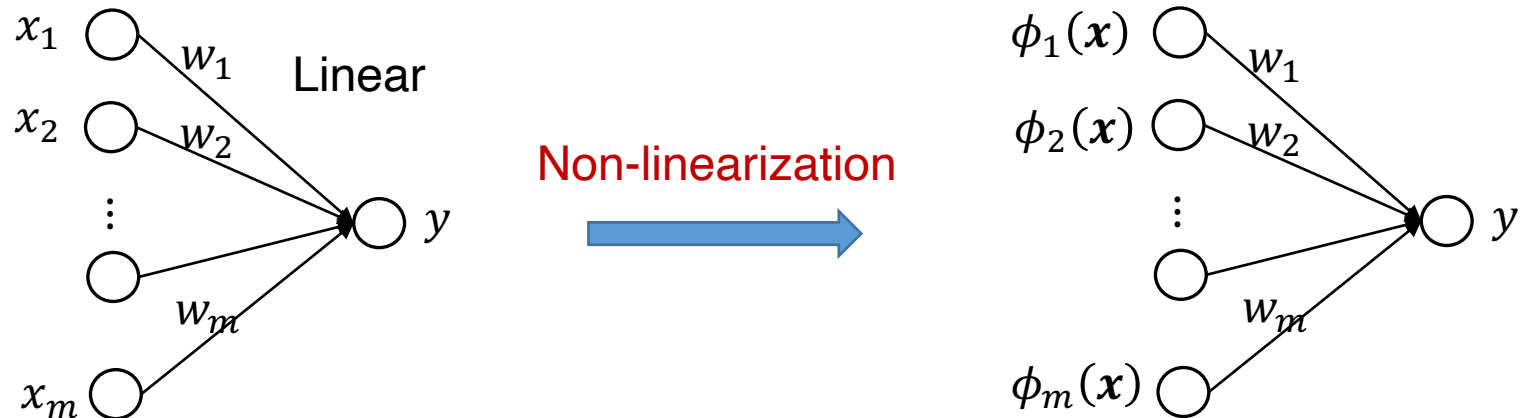
$$\phi: x \rightarrow \phi(x)$$

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

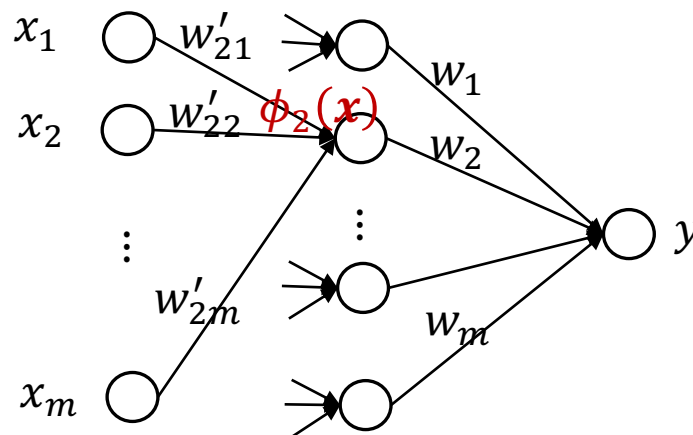
In both methods, the basis functions are fixed and *cannot adaptively adjust according to the characteristics of data*

Neural Networks

- The previous non-linearization process can be illustrated as



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



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

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

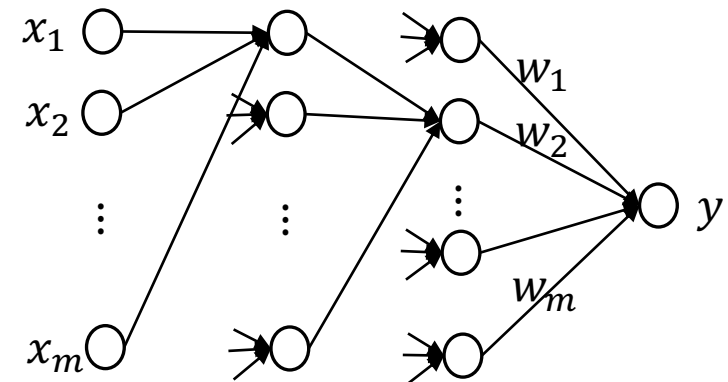
where $a(\cdot)$ is the activation function

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

- The output y can be concisely written as

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

- Further increase the number of layers

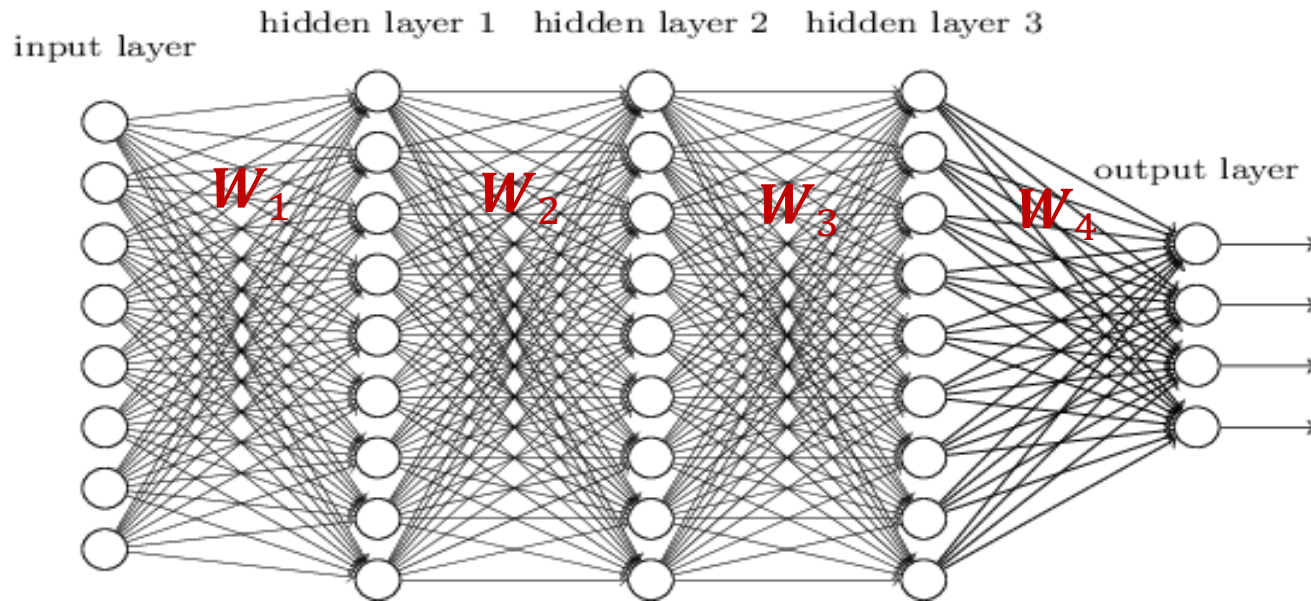


- In general, the output of a L -layer NN can be represented as

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

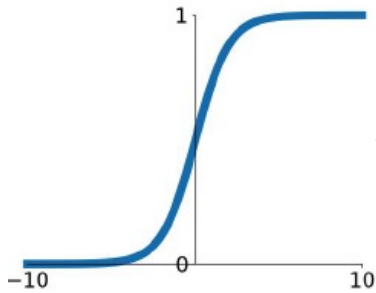
$$\text{Classification: } \hat{y}(\mathbf{x}) = \text{softmax} \left(\mathbf{W}_L a \left(\cdots a(\mathbf{W}_2 a(\mathbf{W}_1 \mathbf{x})) \right) \right)$$

where \mathbf{W}_ℓ is the parameter of the ℓ -th layer



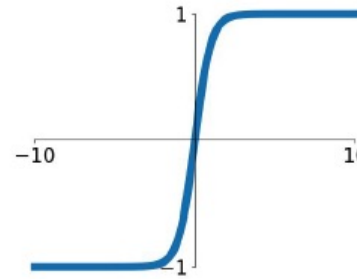
- Activation functions

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



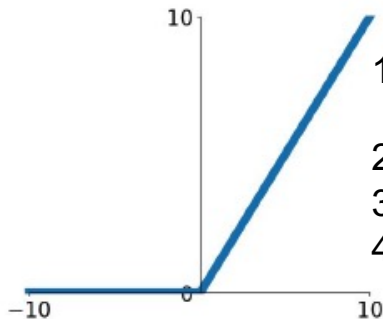
- 1) Vanishing gradient due to saturate
- 2) Output only positive values
- 3) $\exp(\cdot)$ is computationally expensive

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



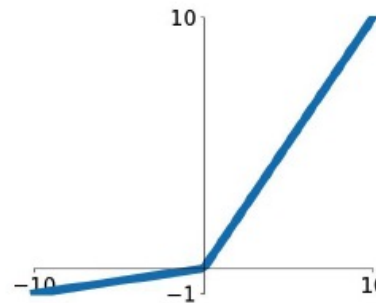
- 1) Vanishing gradient due to saturate
- 2) $\exp(\cdot)$ is computationally expensive
- 3) Output both positive and negative values

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 $\mathcal{D} = \{\mathbf{x}_i, y_i\}_{i=1}^N$, the training objectives for the regression and classification are:

1) Regression loss

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

2) Multi-class classification loss

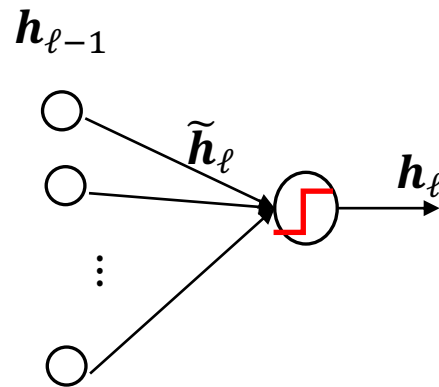
$$\mathcal{L}_c(\boldsymbol{\theta}) = -\frac{1}{N} \sum_{(\mathbf{x}, y) \in \mathcal{D}} \sum_{k=1}^K y_k \log \hat{y}_k(\mathbf{x})$$

Outline

- Neural Networks
- Backpropagation
- Typical Neural Networks

Gradient

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

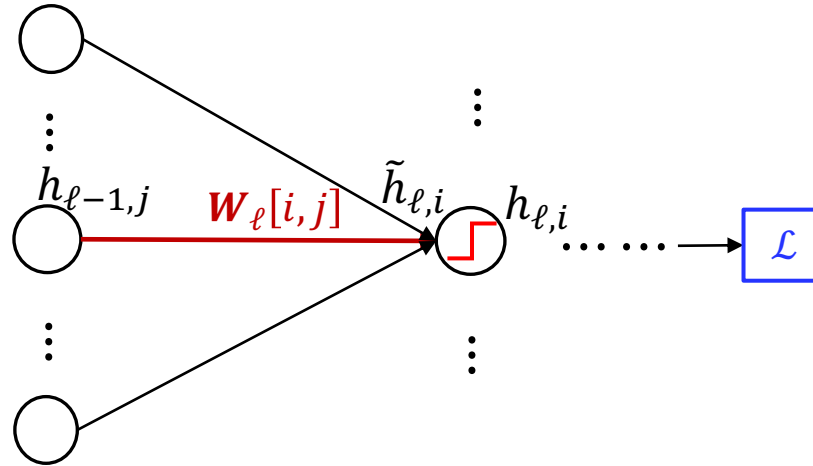


$$\tilde{\mathbf{h}}_\ell = \mathbf{W}_\ell \mathbf{h}_{\ell-1}, \quad \mathbf{h}_\ell = a(\tilde{\mathbf{h}}_\ell), \quad \mathbf{h}_\ell = a(\mathbf{W}_\ell \mathbf{h}_{\ell-1})$$

➤ The output layer

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

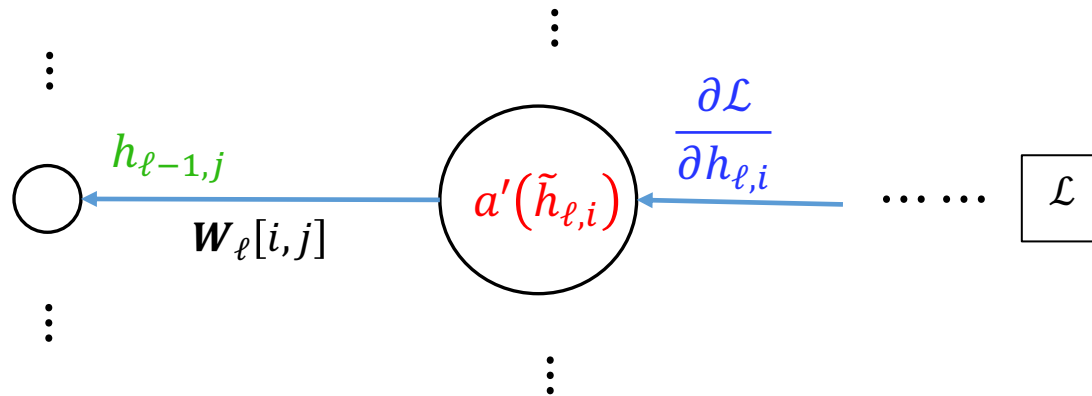
- The derivative of the loss w.r.t. $\mathbf{W}_\ell[i, j]$, that is, $\frac{\partial \mathcal{L}}{\partial \mathbf{W}_\ell[i, j]}$



$$\begin{aligned}
 \frac{\partial \mathcal{L}}{\partial \mathbf{W}_\ell[i, j]} &= \frac{\partial \mathcal{L}}{\partial h_{\ell, i}} \cdot \frac{\partial h_{\ell, i}}{\partial \mathbf{W}_\ell[i, j]} \\
 &= \frac{\partial \mathcal{L}}{\partial h_{\ell, i}} \cdot \frac{\partial h_{\ell, i}}{\partial \tilde{h}_{\ell, i}} \cdot \frac{\partial \tilde{h}_{\ell, i}}{\partial \mathbf{W}_\ell[i, j]} \\
 &= \frac{\partial \mathcal{L}}{\partial h_{\ell, i}} \cdot a'(\tilde{h}_{\ell, i}) \cdot h_{\ell-1, j}
 \end{aligned}$$

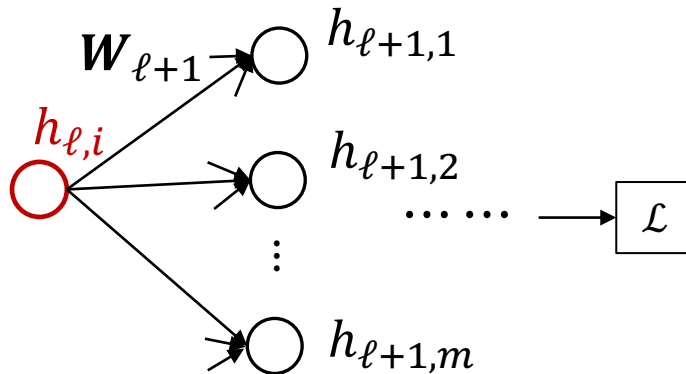
➤ Schematic diagram showing the structure of the derivative

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



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

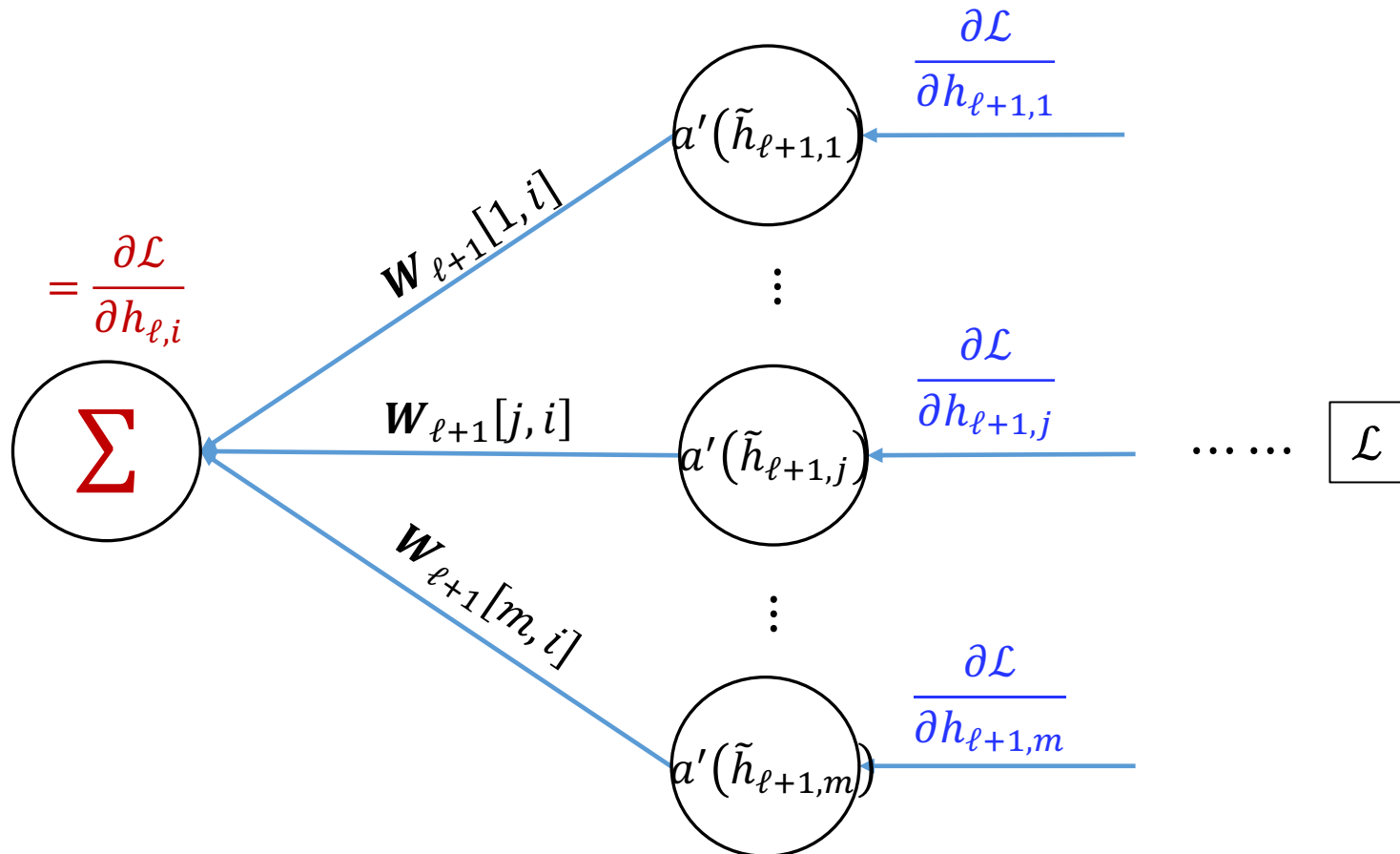
- The derivative of the loss w.r.t. $h_{\ell,i}$, that is, $\frac{\partial \mathcal{L}}{\partial h_{\ell,i}}$



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

➤ The recursive structure in the expression

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



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

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

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

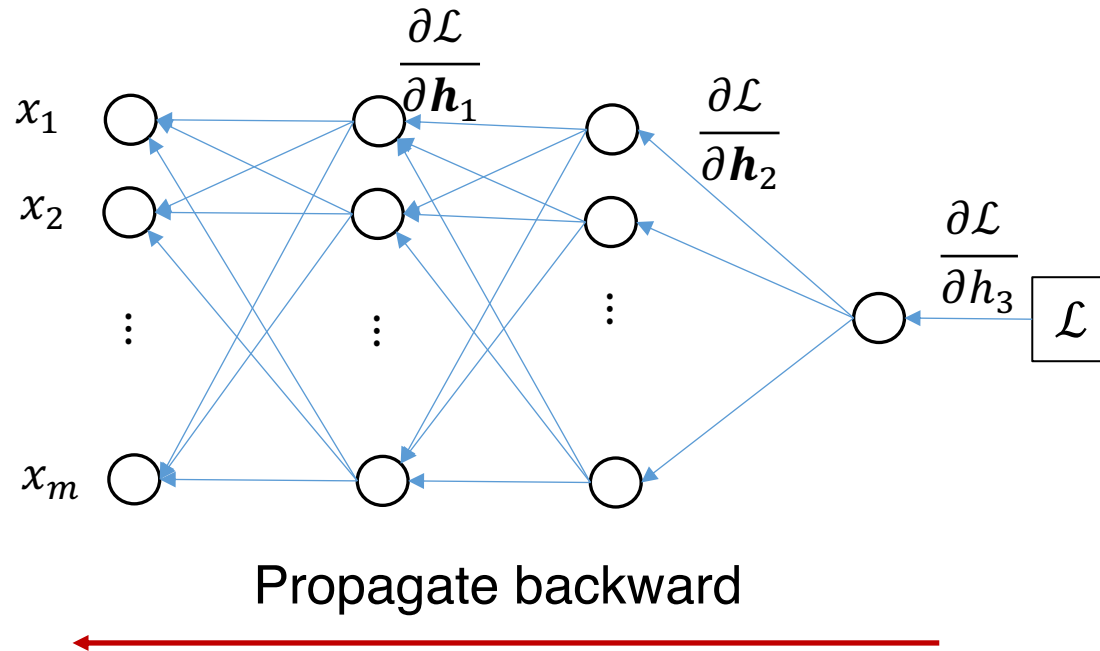
2) Cross-entropy loss: $\mathcal{L} = -y \log h_L - (1 - y) \log(1 - h_L)$

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

The Whole Computation Graph of Gradient

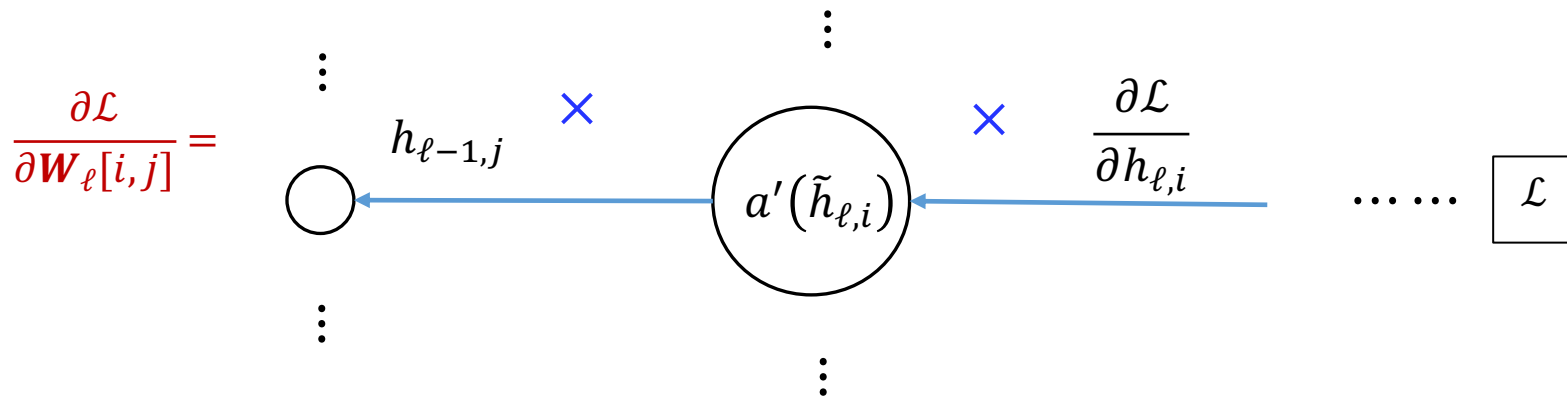
- Two steps

➤ **Step 1:** Computing $\frac{\partial \mathcal{L}}{\partial \mathbf{h}_\ell}$



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

➤ **Step 2:** Computation of $\frac{\partial \mathcal{L}}{\partial \mathbf{W}_\ell}$



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

Written in matrix form as

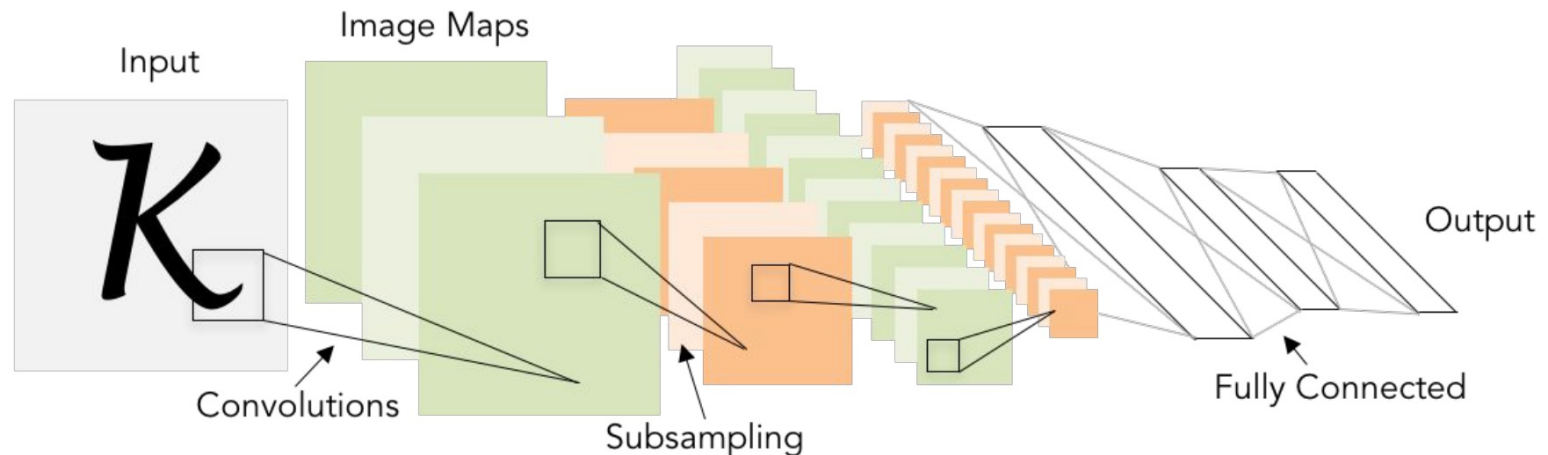
$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_\ell} = \left(\tilde{\mathbf{g}}_\ell \odot \frac{\partial \mathcal{L}}{\partial \mathbf{h}_\ell} \right) \mathbf{h}_{\ell-1}^T$$

Outline

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

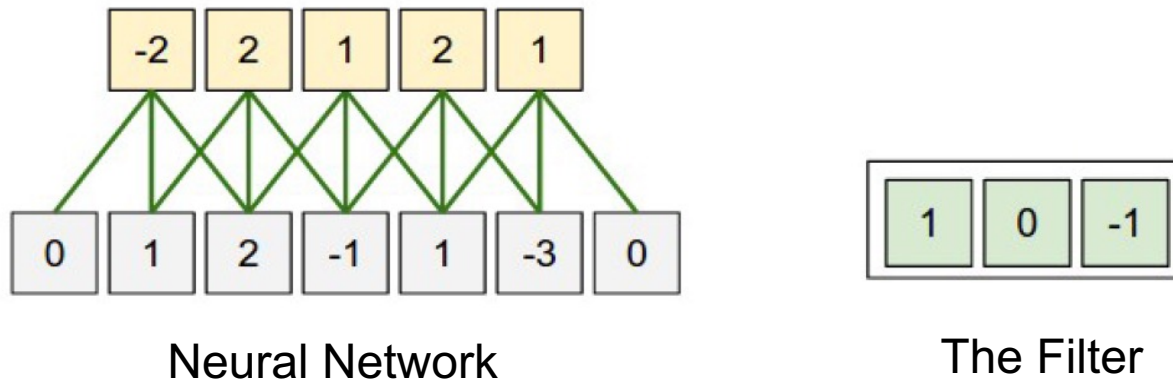
Convolutional Neural Networks (CNN)

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



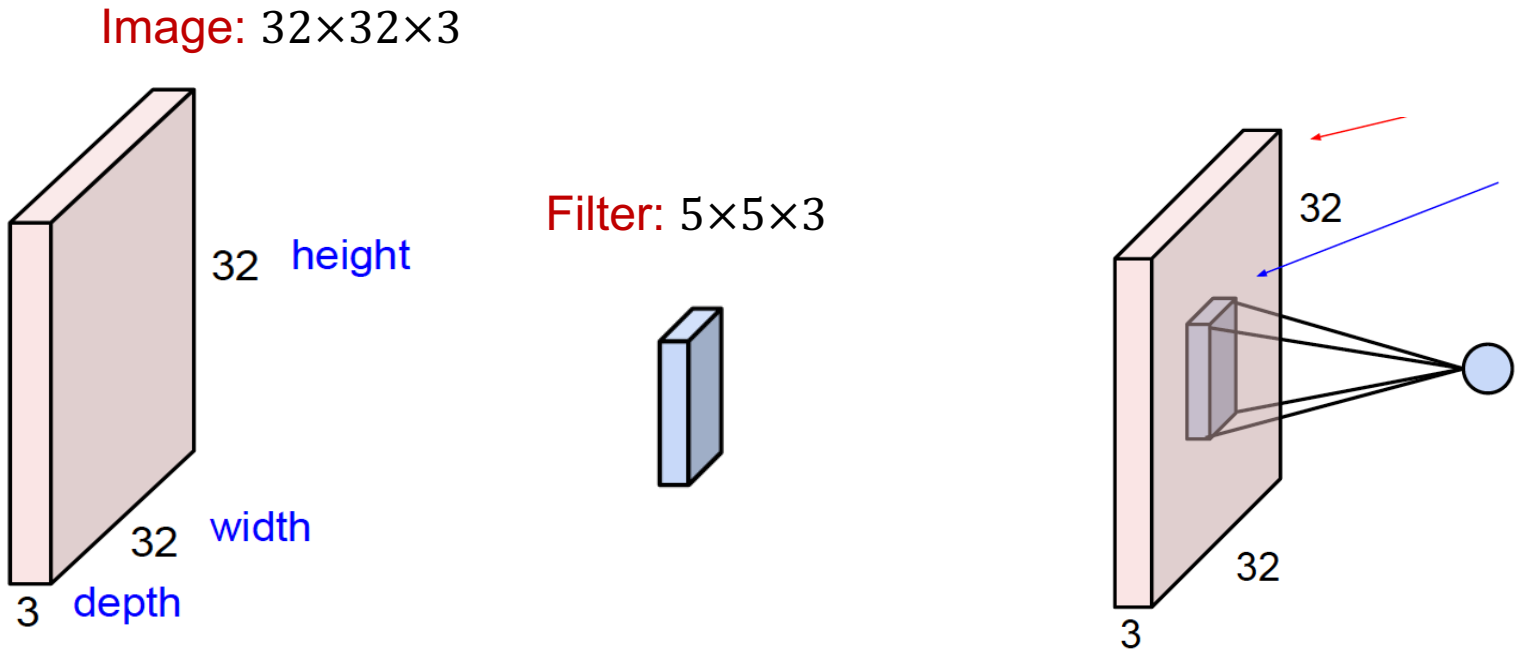
Convolution Operations

- One-dimensional convolution operation illustration



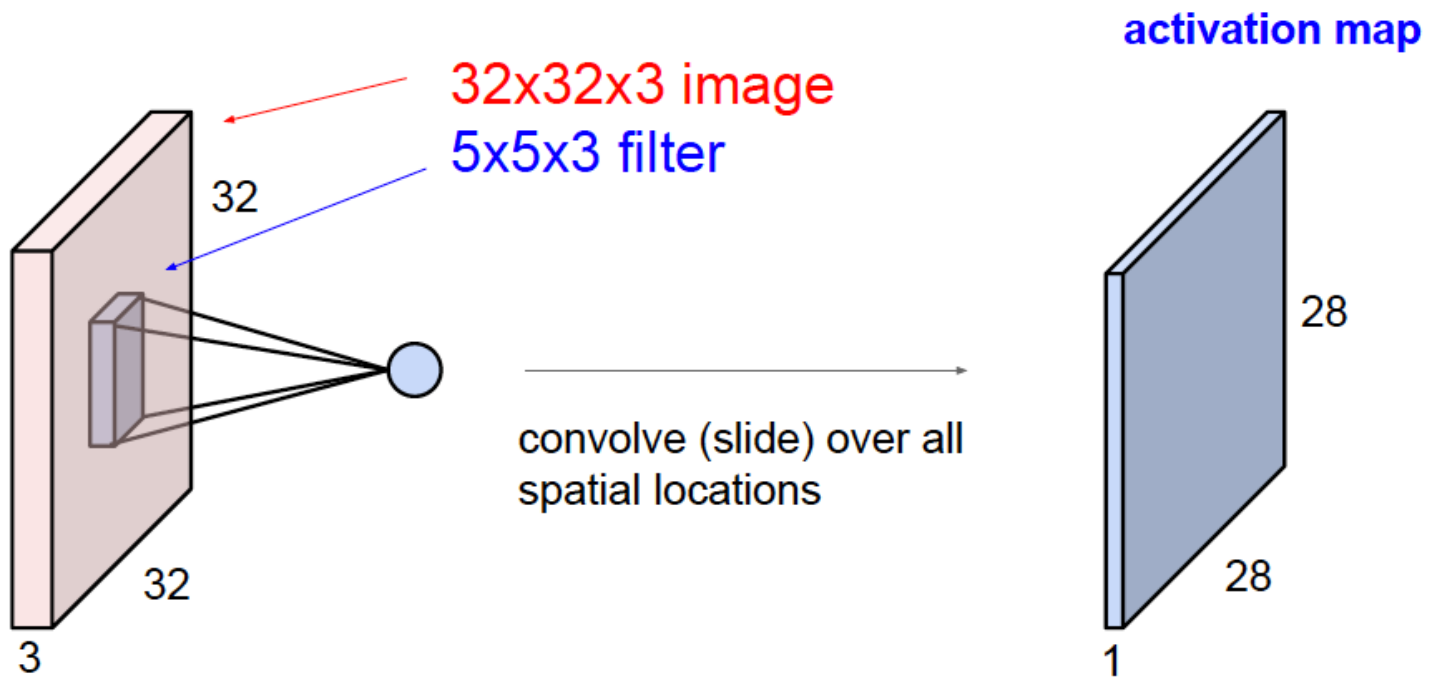
- Distinctions from the MLPs
 - **Local connectivity:** each neuron is only connected to a fraction of neurons in previous layer
 - **Parameter sharing:** the parameters for different neurons are tied

- Two-dimensional convolution operation illustration

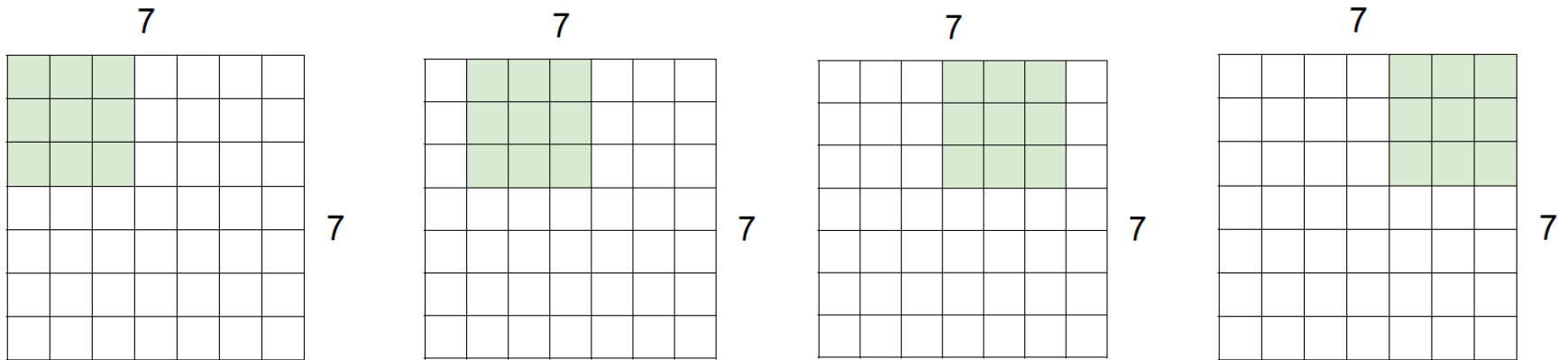


Feature Maps

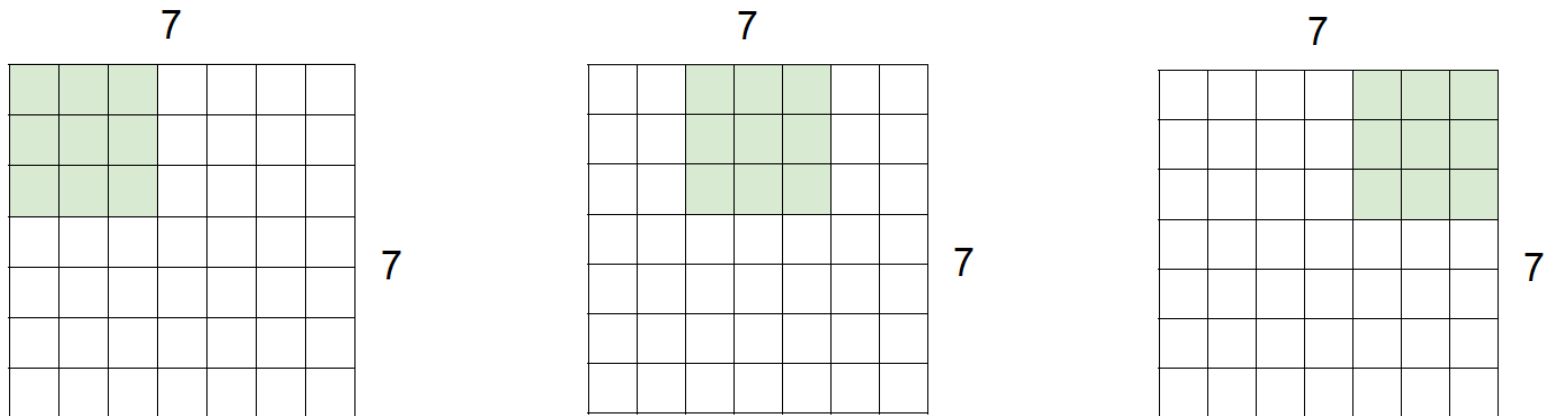
- Sliding the filter along the two dimensions of images, yielding an activation map with dimension $28 \times 28 \times 1$



A closer look at the convolution operation

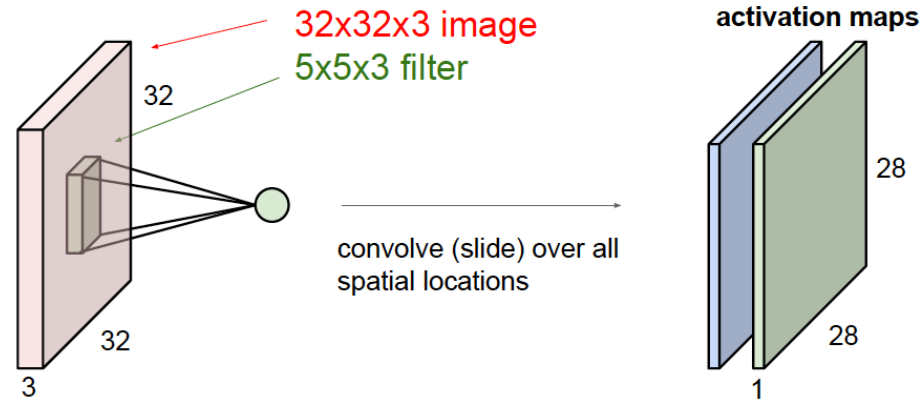


Stride=1

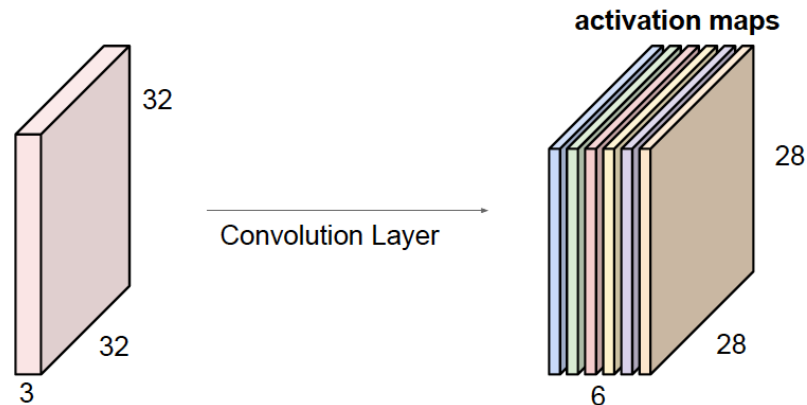


Stride=2

- 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



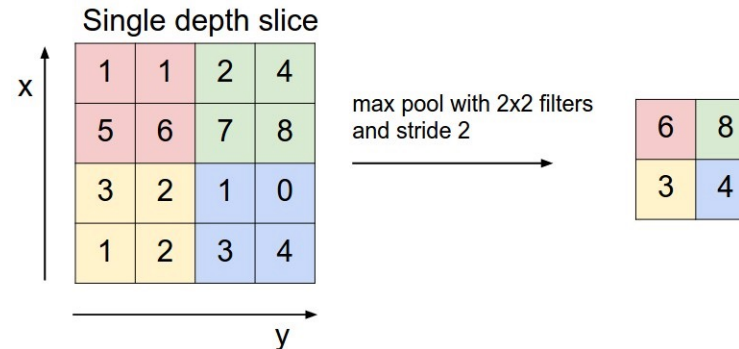
Edge Detect



Strong Edge Detect

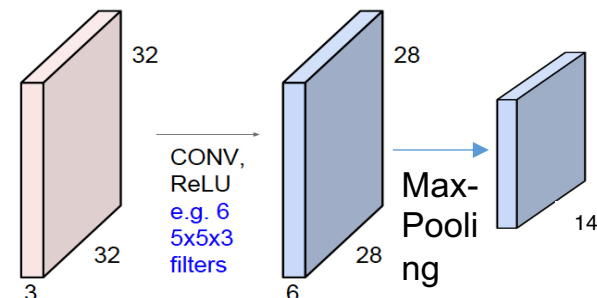
Pooling

- Illustration



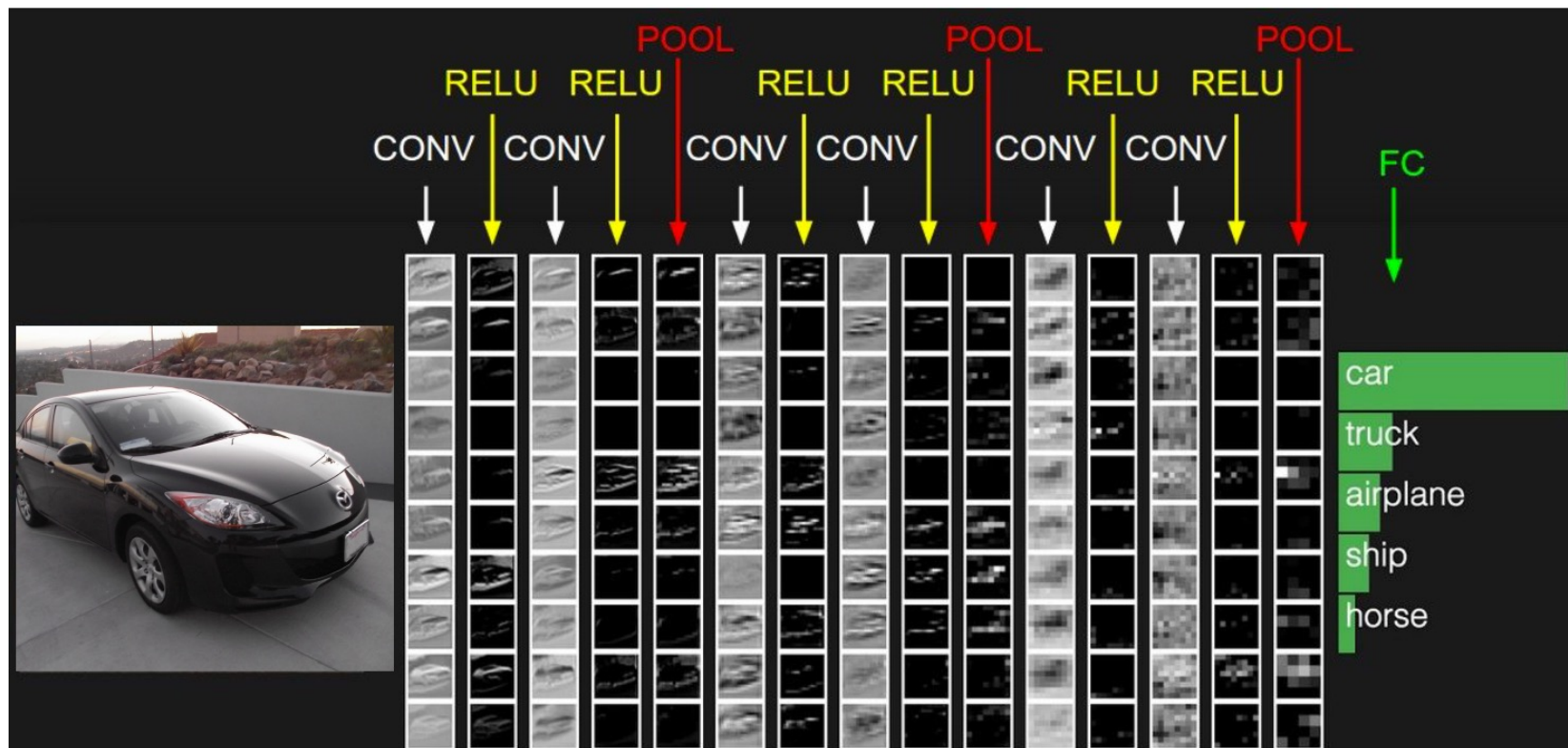
- 1) Reducing the dimensionality
- 2) Introducing nonlinearities
- 3) Maintaining some extent of spatial invariance
- 4) The pooling is not necessary at all layer

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



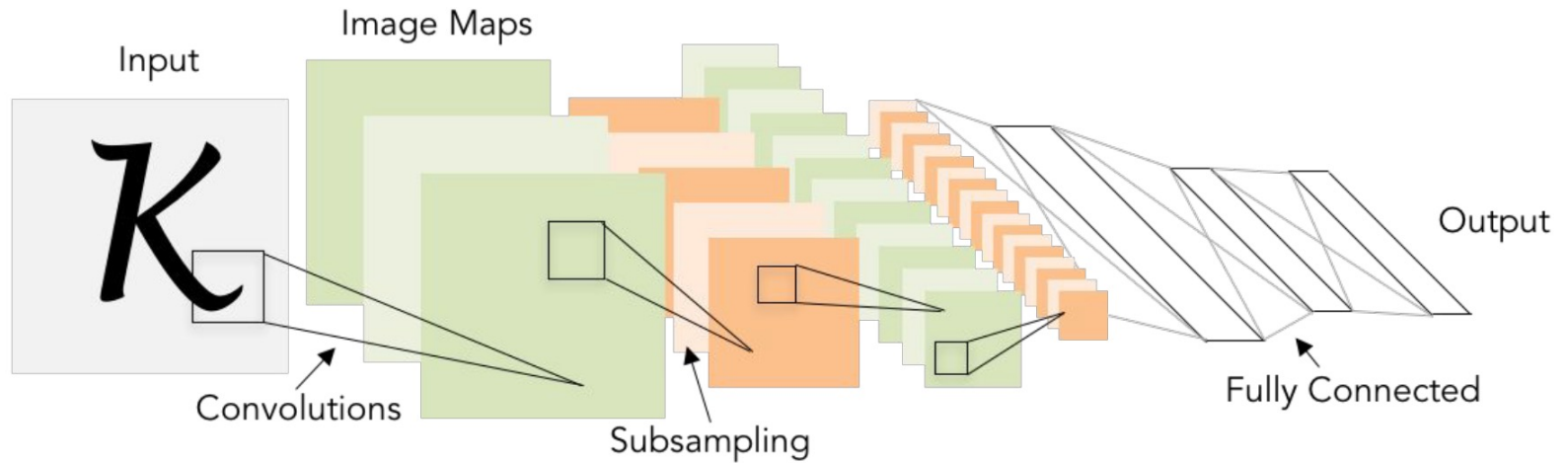
Illustration

- An illustration of image classification with CNN



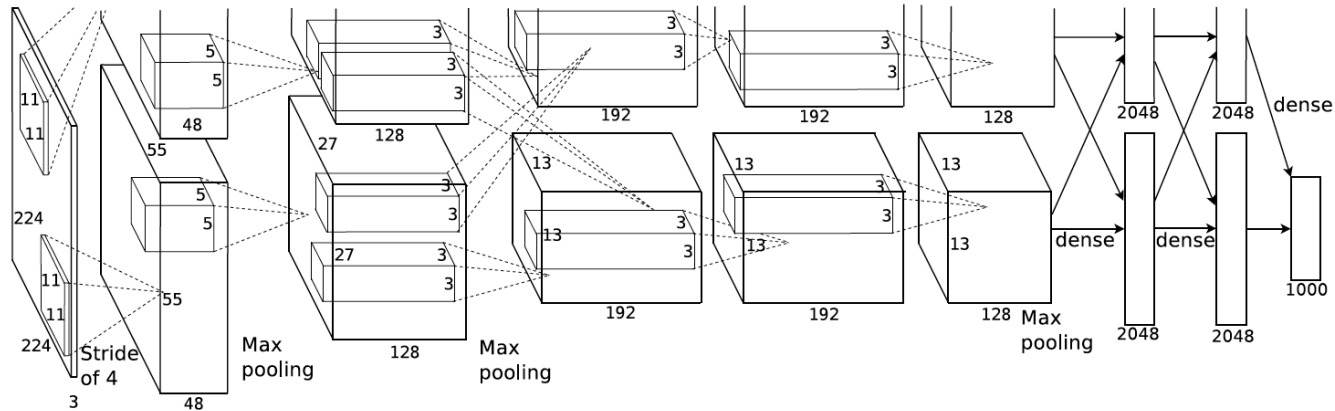
Several Typical CNNs

- LeNet



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

- AlexNet

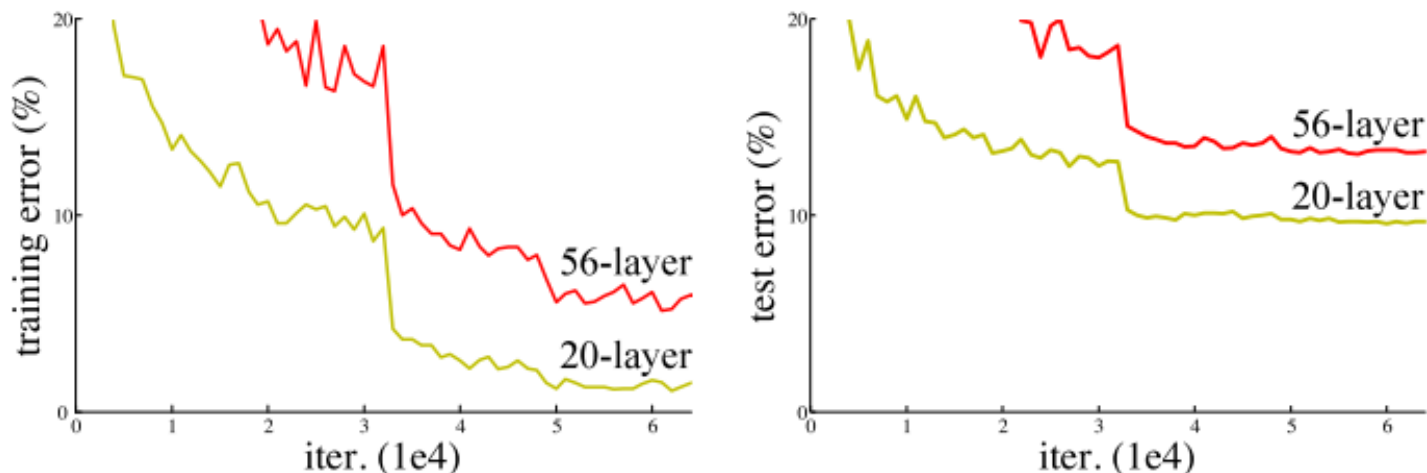


Input	Conv1	MaxPool1	Conv2	MaxPool2	Conv3
227×227×3	55×55×96	27×27×96	27×27×256	13×13×256	13×13×384

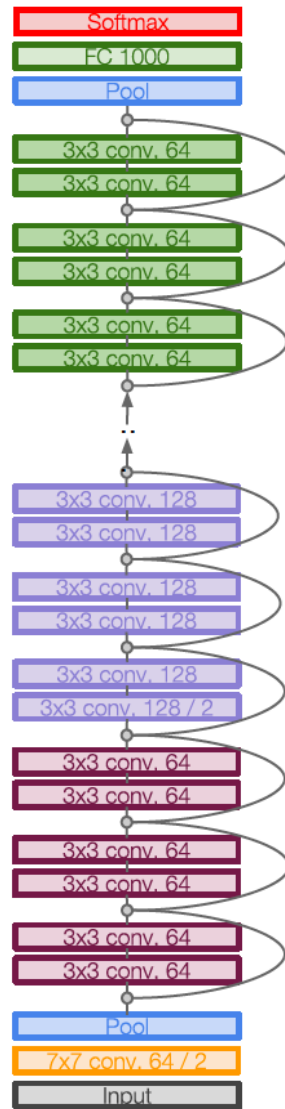
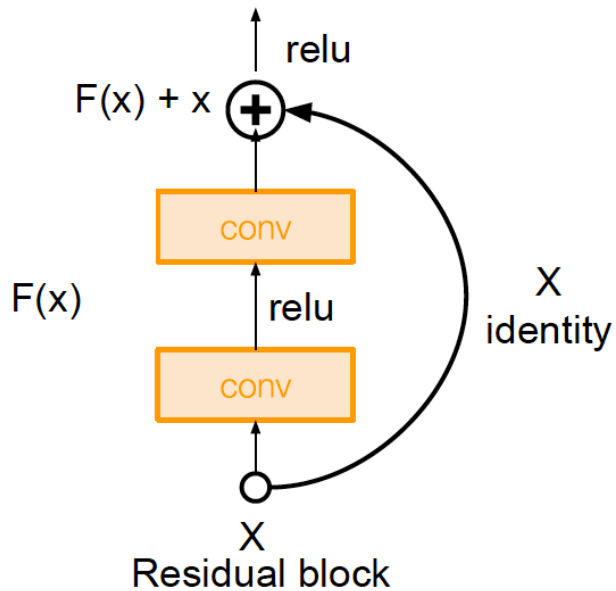
Conv4	Conv5	MaxPool3	FC6	FC7	FC8
13×13×384	13×13×256	6×6×256	4096	4096	1000

- ResNet

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



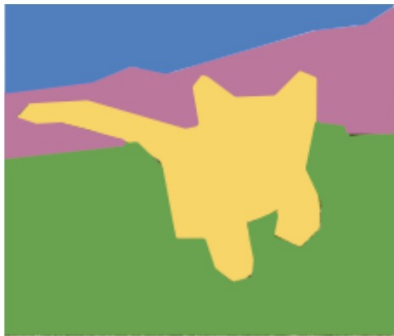
- One of the reasons is that deeper models are more difficult to train
- ResNet alleviates the training difficulties by introducing a 'shortcut' connection between layers



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

Typical Applications beyond Classification

Semantic Segmentation



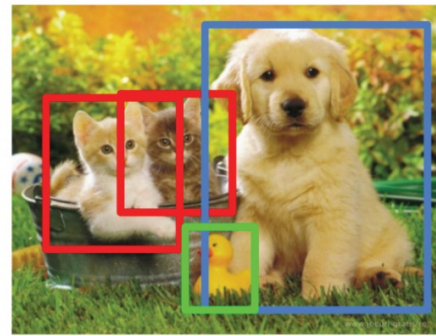
CAT

Classification + Localization



CAT

Object Detection



CAT, DOG, DUCK

Instance Segmentation



CAT, DOG, DUCK

Vanilla RNNs

- The vanilla RNN can be described by two equations:

- 1) Hidden state updating equation

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

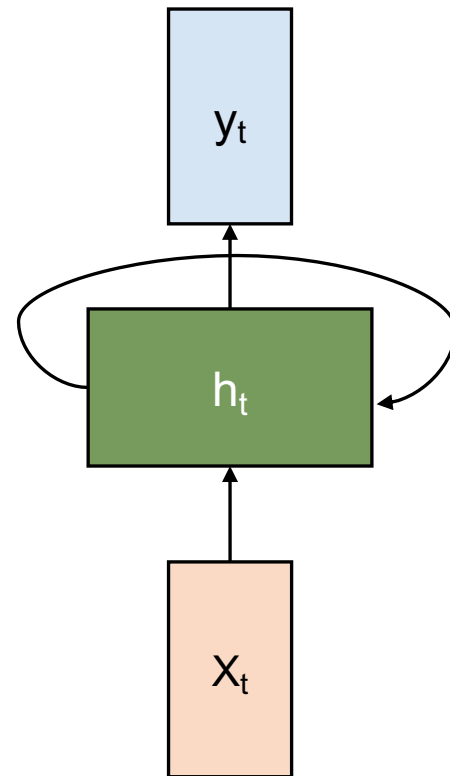
- 2) Output

$$y_t = f_o(h_t)$$

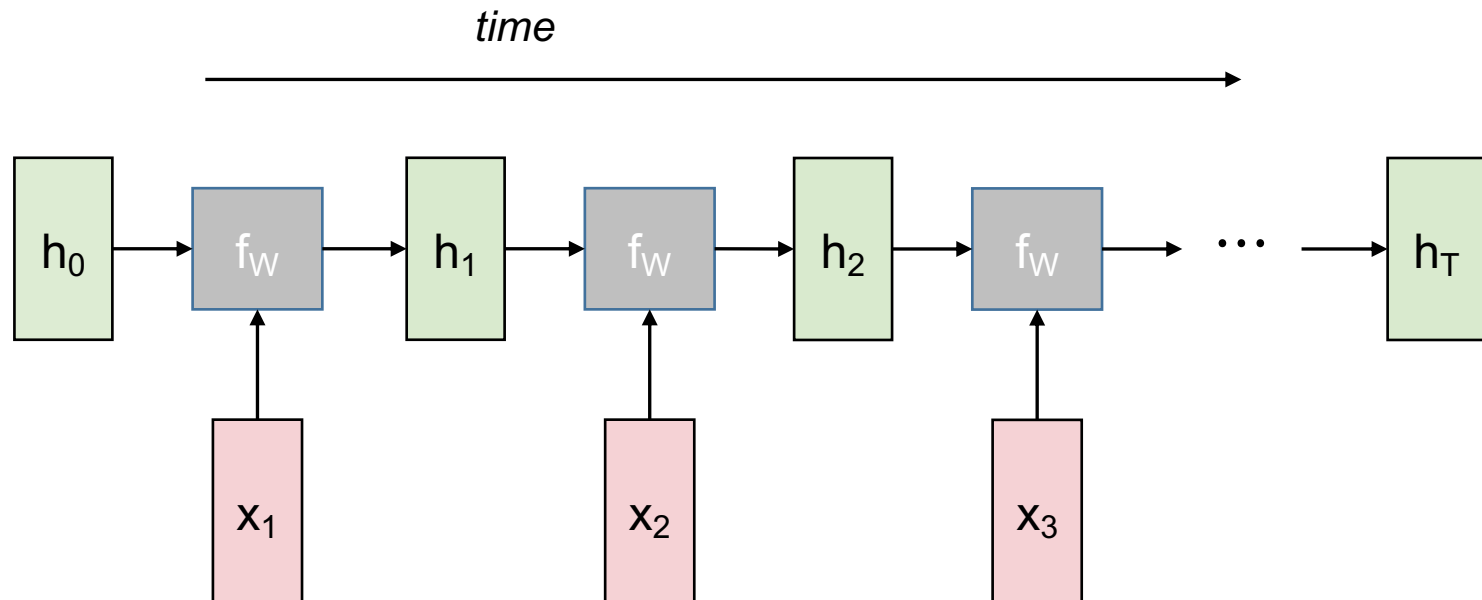
- For instance, $f_h(\cdot)$ and $f_o(\cdot)$ could be

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

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

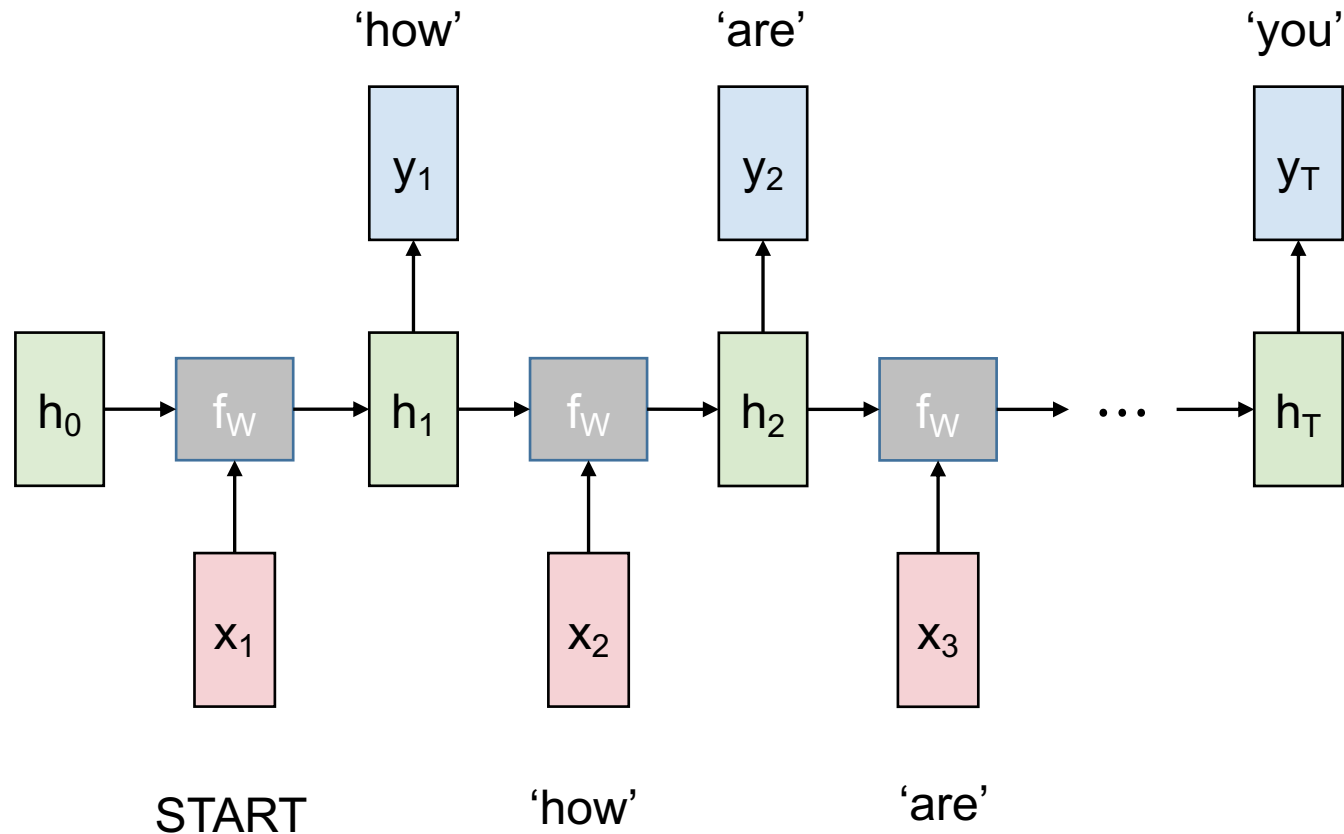


- Unfolding the RNN over time



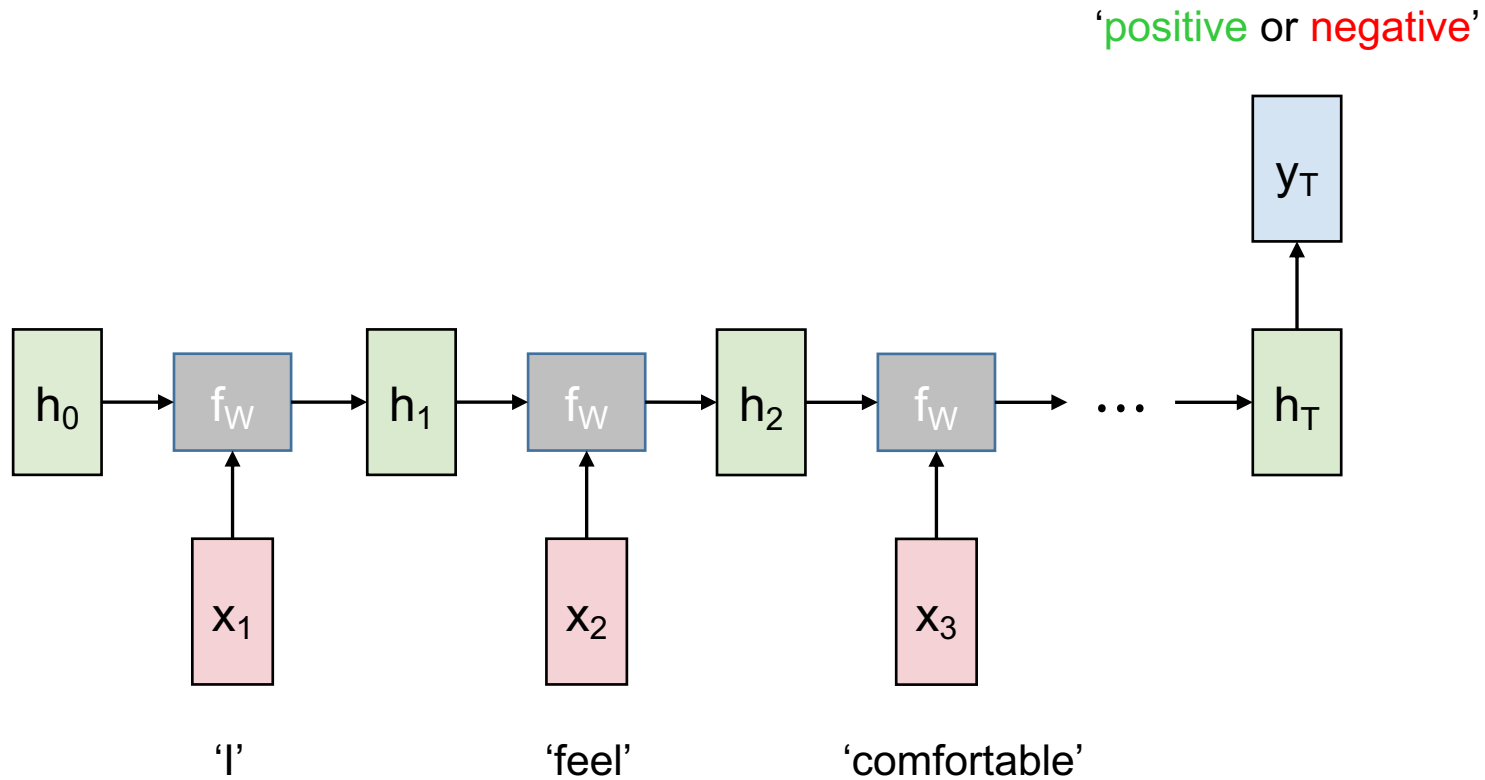
Output Types

- Many to Many



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

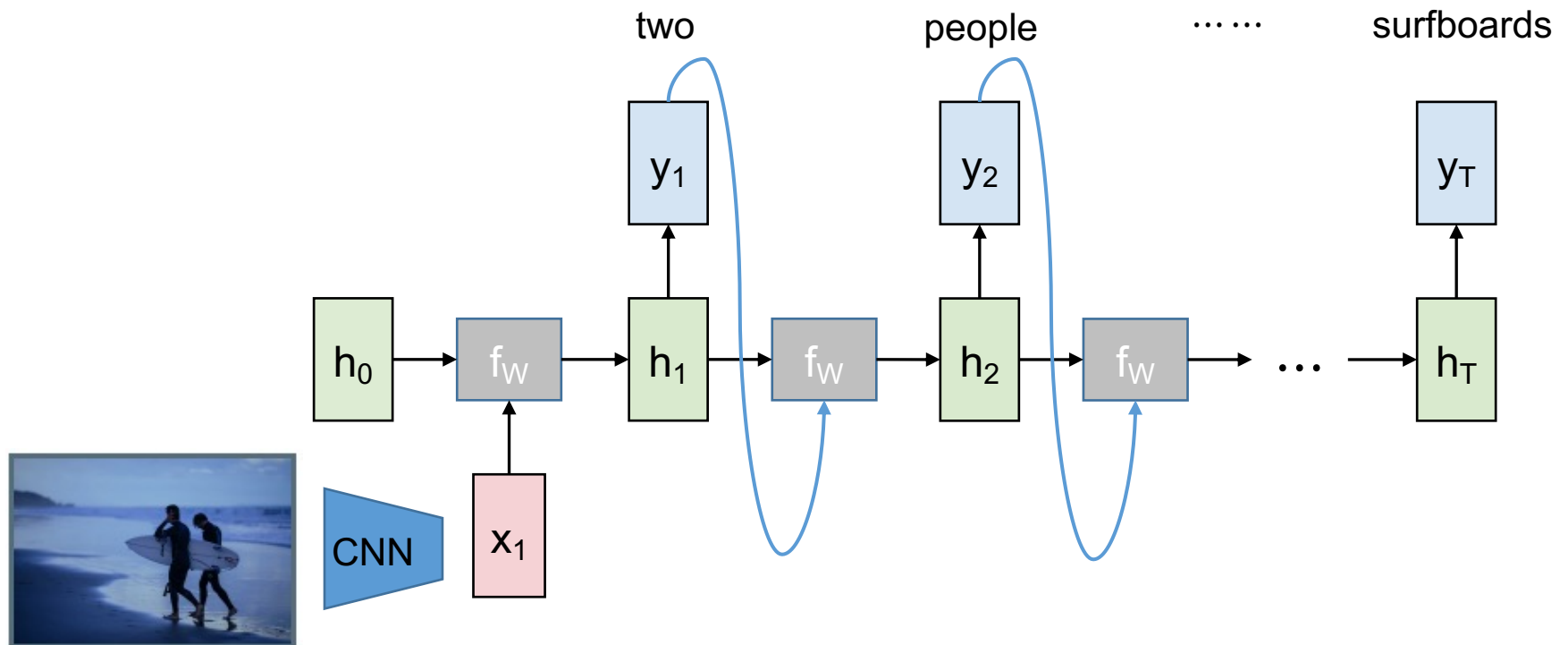
- Many to one



Applications: Sequence classification, sentiment analysis *etc.*

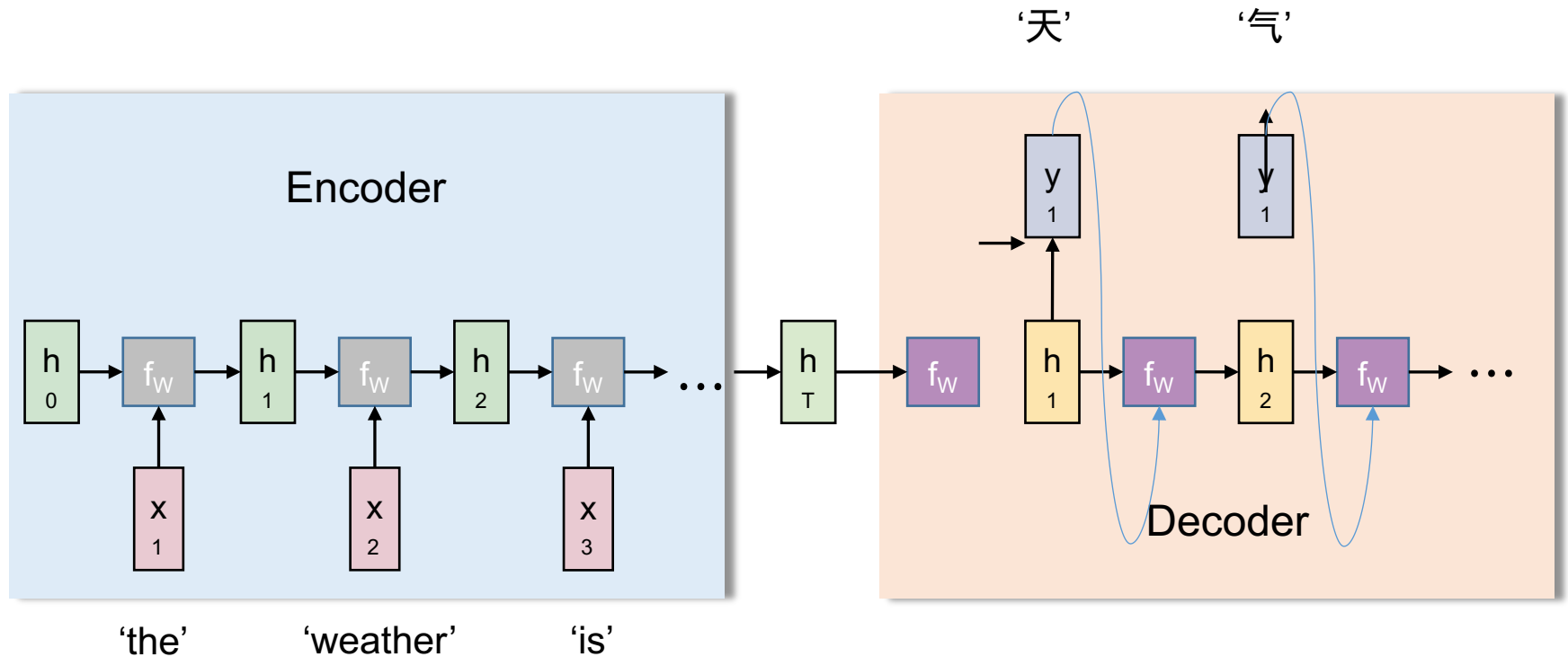
- One to many

Two people walking on the beach with surfboards



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

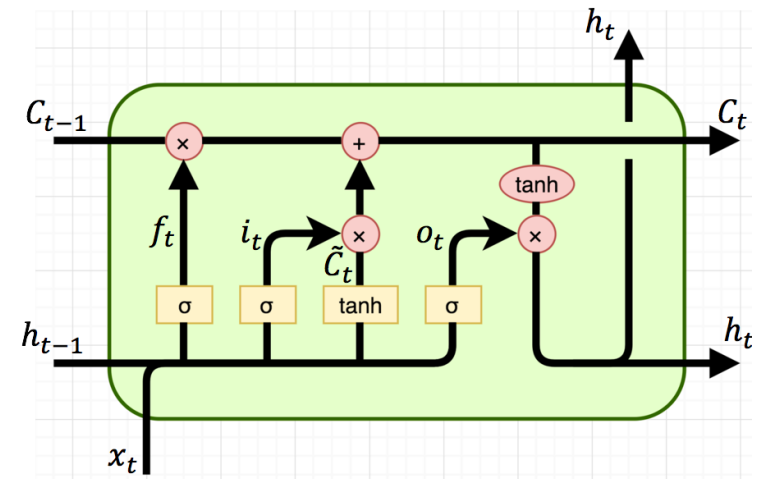
4) $\tilde{C}_t = \tanh(W_g[x_t, h_{t-1}])$

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

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

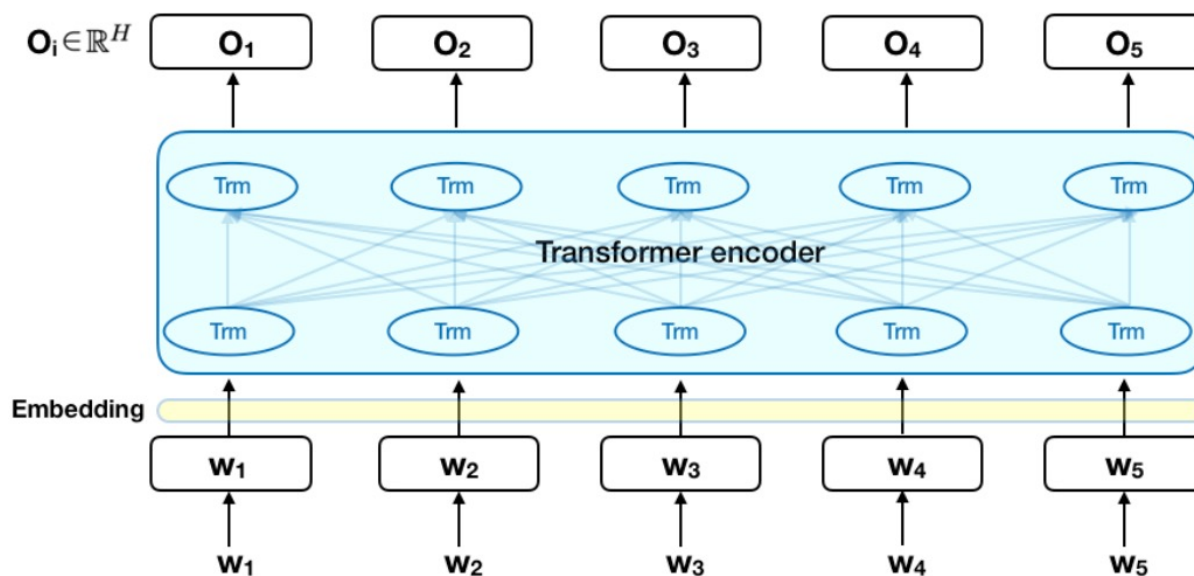
Gates

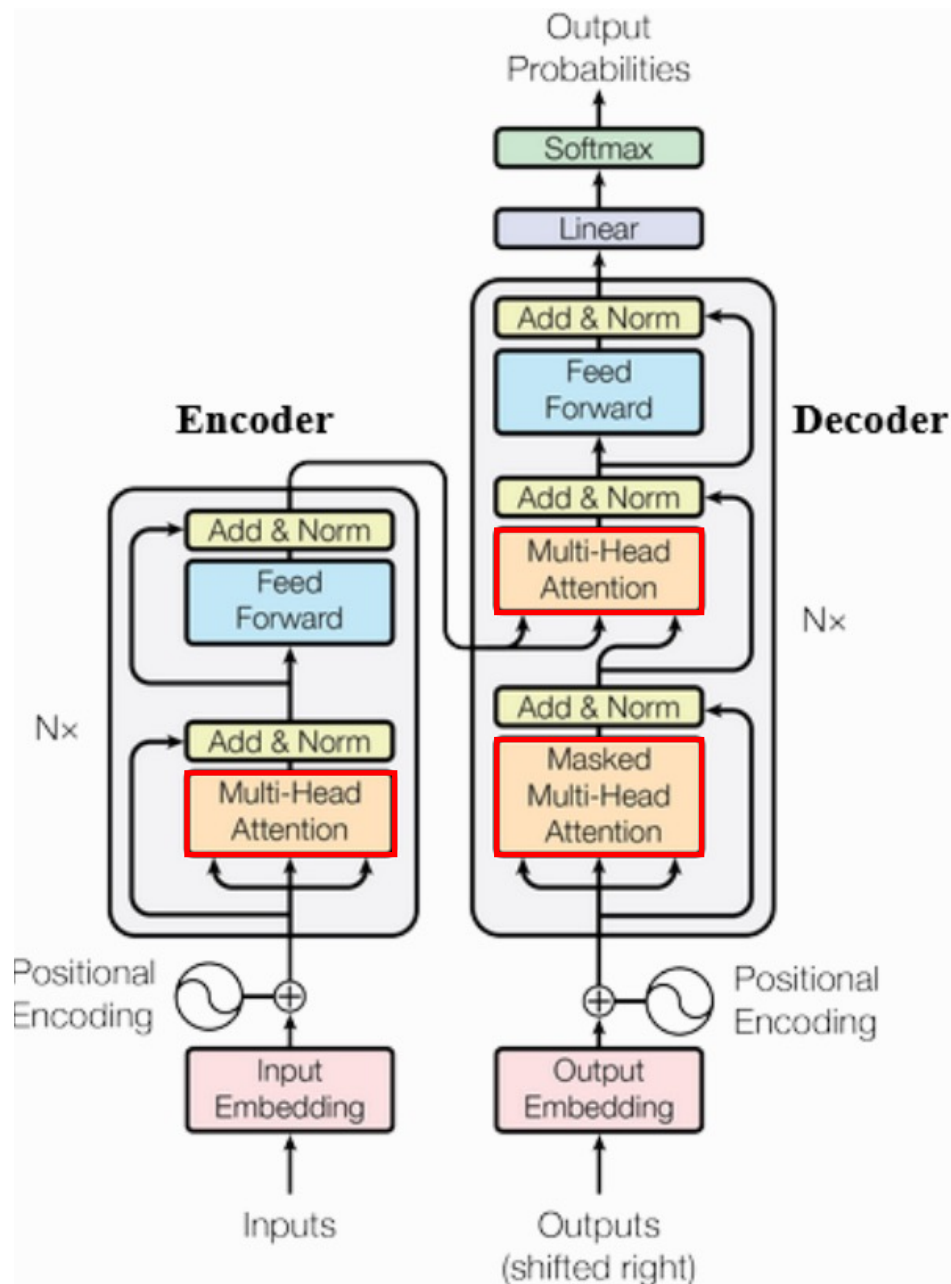
Outputs

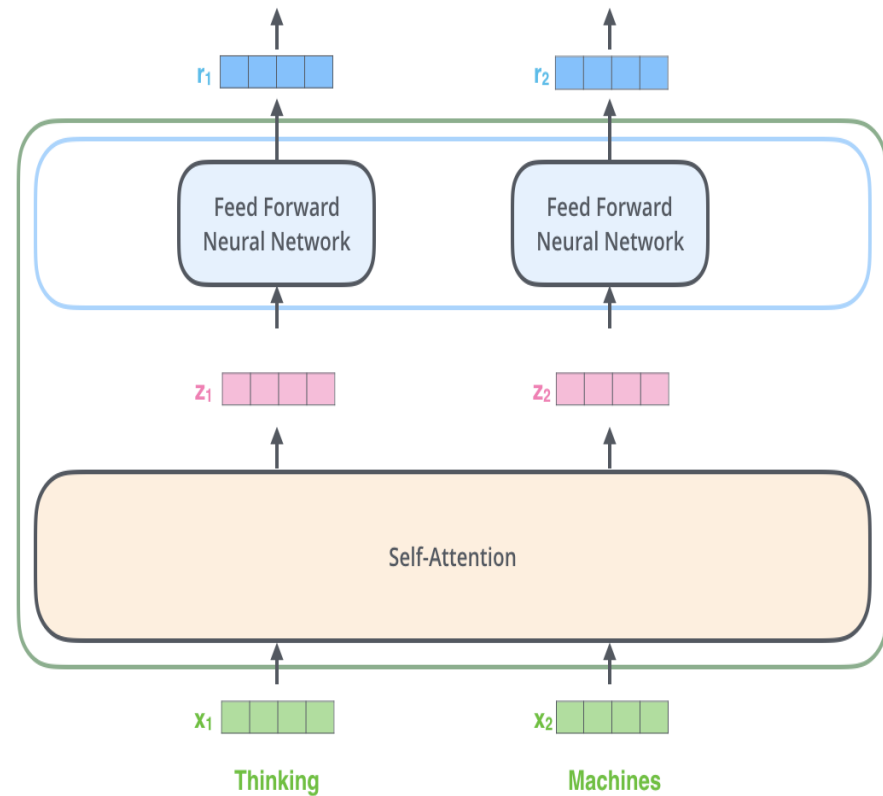
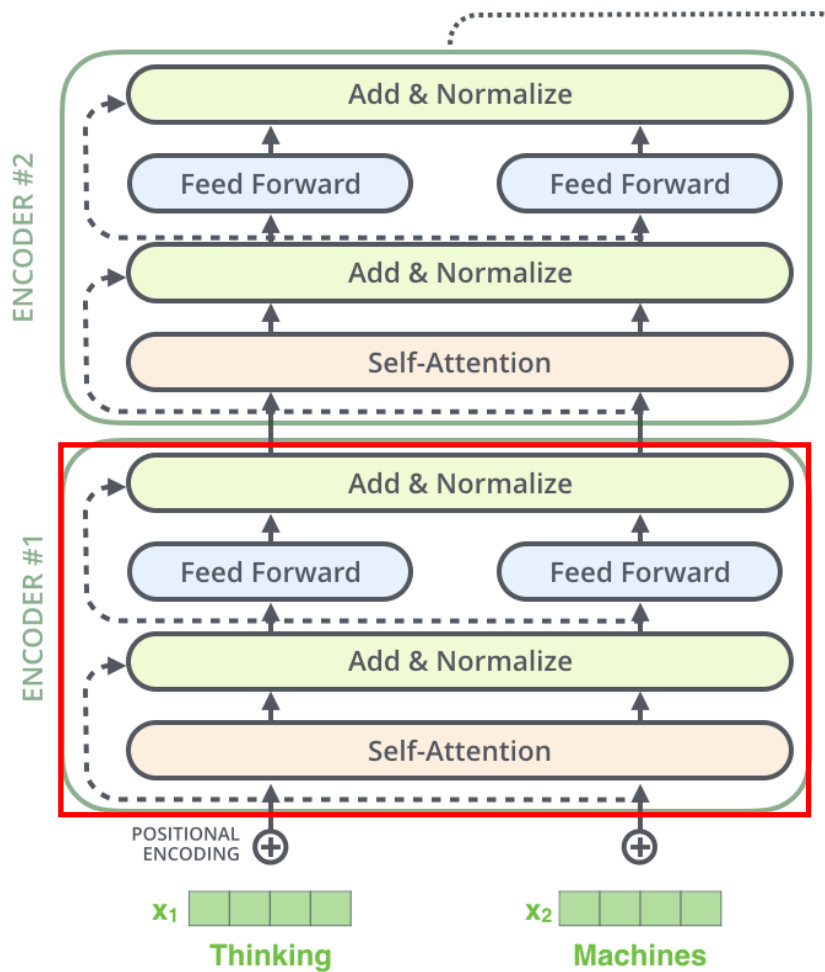


Transformer-based Neural Networks

- Problems with RNN
 - 1) Difficult to capture **long-term dependencies** of words in documents
 - 2) Only admit 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**

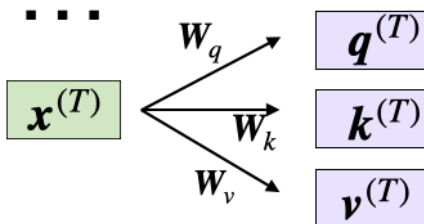
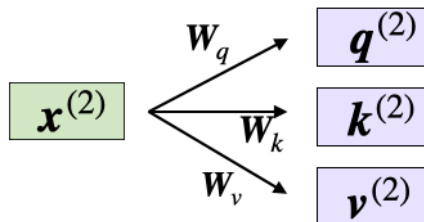
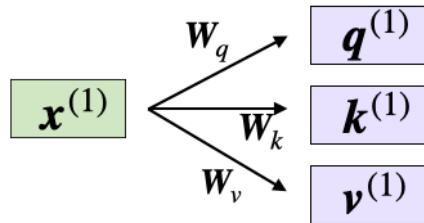




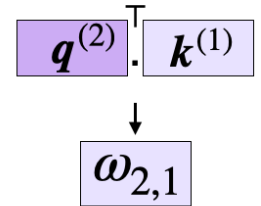
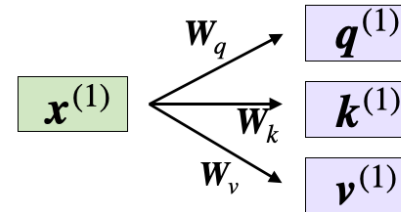


- Self-attention mechanism

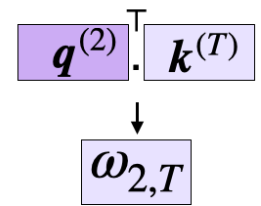
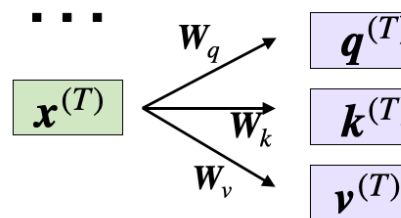
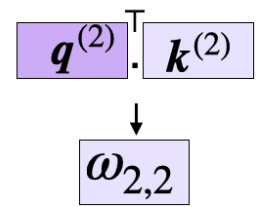
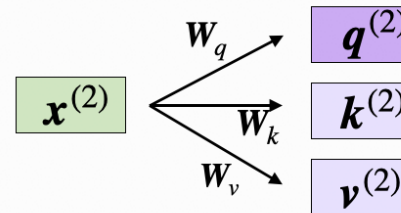
(1) query, key, value



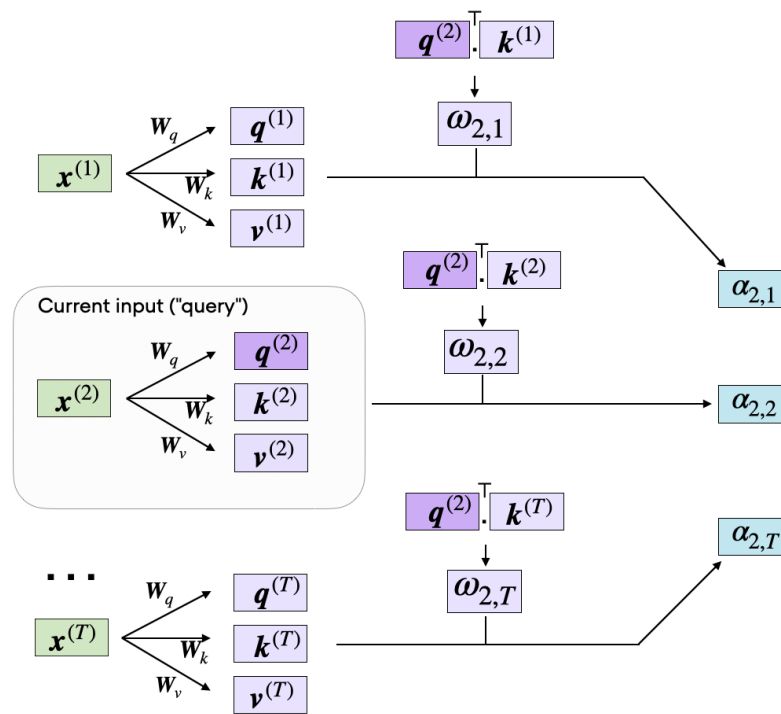
(2) query-key similarity



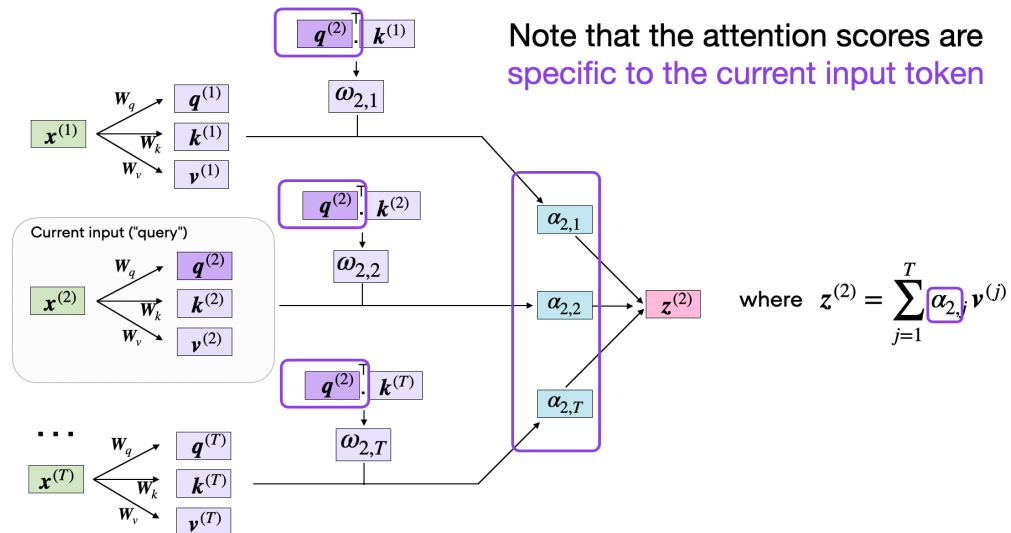
Current input ("query")



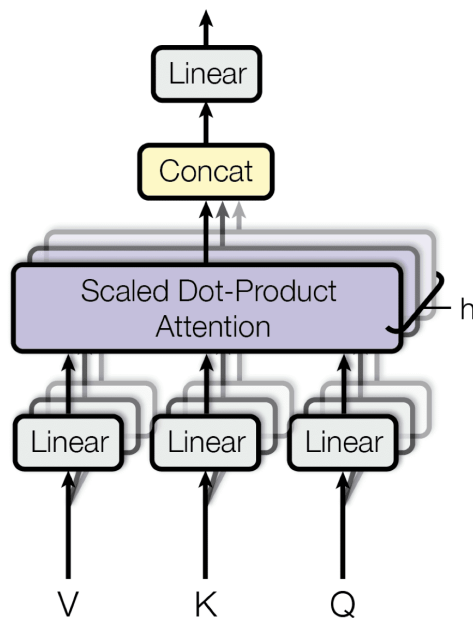
(3) attention weigh



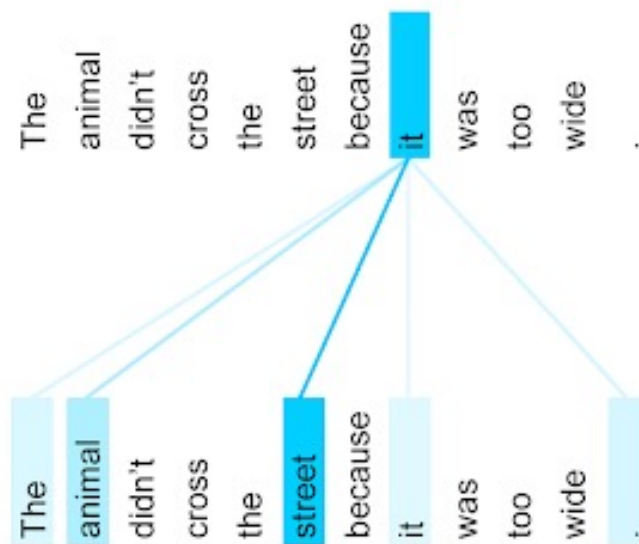
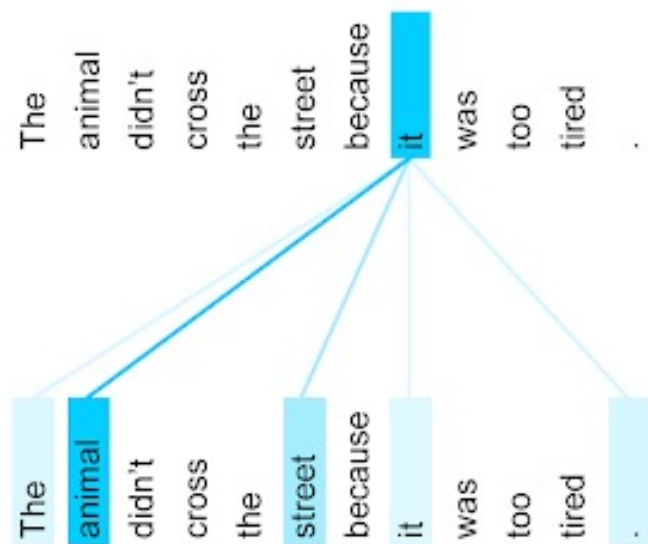
(4) output



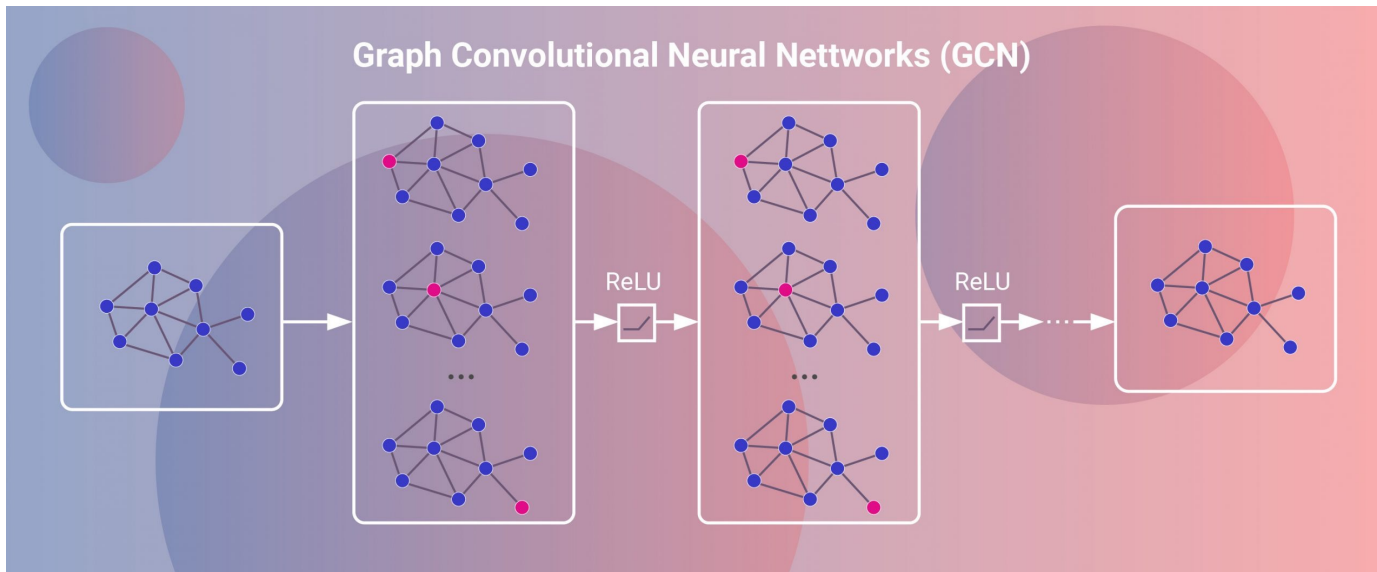
(5) multi-head attention



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

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