神经网络与深度学习

吉建民

USTC
jianmin@ustc.edu.cn

2024 年 5 月 19 日

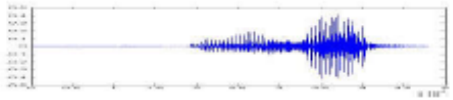
Used Materials

Disclaimer: 本课件大量采用了台大李宏毅教授及复旦大学邱锡鹏副教授的深度学习课件内容，也采用了 GitHub 中深度学习开源代码，以及部分网络博客内容
建议学习：《动手学深度学习》

Table of Contents

Machine Learning ≈ Looking for a Function

- Speech Recognition

$f($  $)=$ “How are you”

- Image Recognition

$f($  $)=$ “Cat”

- Playing Go

$f($  $)=$ “5-5” (next move)

- Dialogue System

$f($ “Hi” $)=$ “Hello”

(what the user said) (system response)

Framework

Image Recognition:

$f($  $)=$ "cat"

A set of function

Model

$f_1, f_2 \cdots$

$f_1($  $)=$ "cat"

$f_2($  $)=$ "money"

$f_1($  $)=$ "dog"

$f_2($  $)=$ "snake"

Framework

Image Recognition:

$f($  $) =$ "cat"

A set of function

Model

$f_1, f_2 \cdots$

Goodness of function f

Training Data

$f_1($  $) =$ "cat"

$f_2($  $) =$ "money"

Better!

$f_1($  $) =$ "dog"

$f_2($  $) =$ "snake"

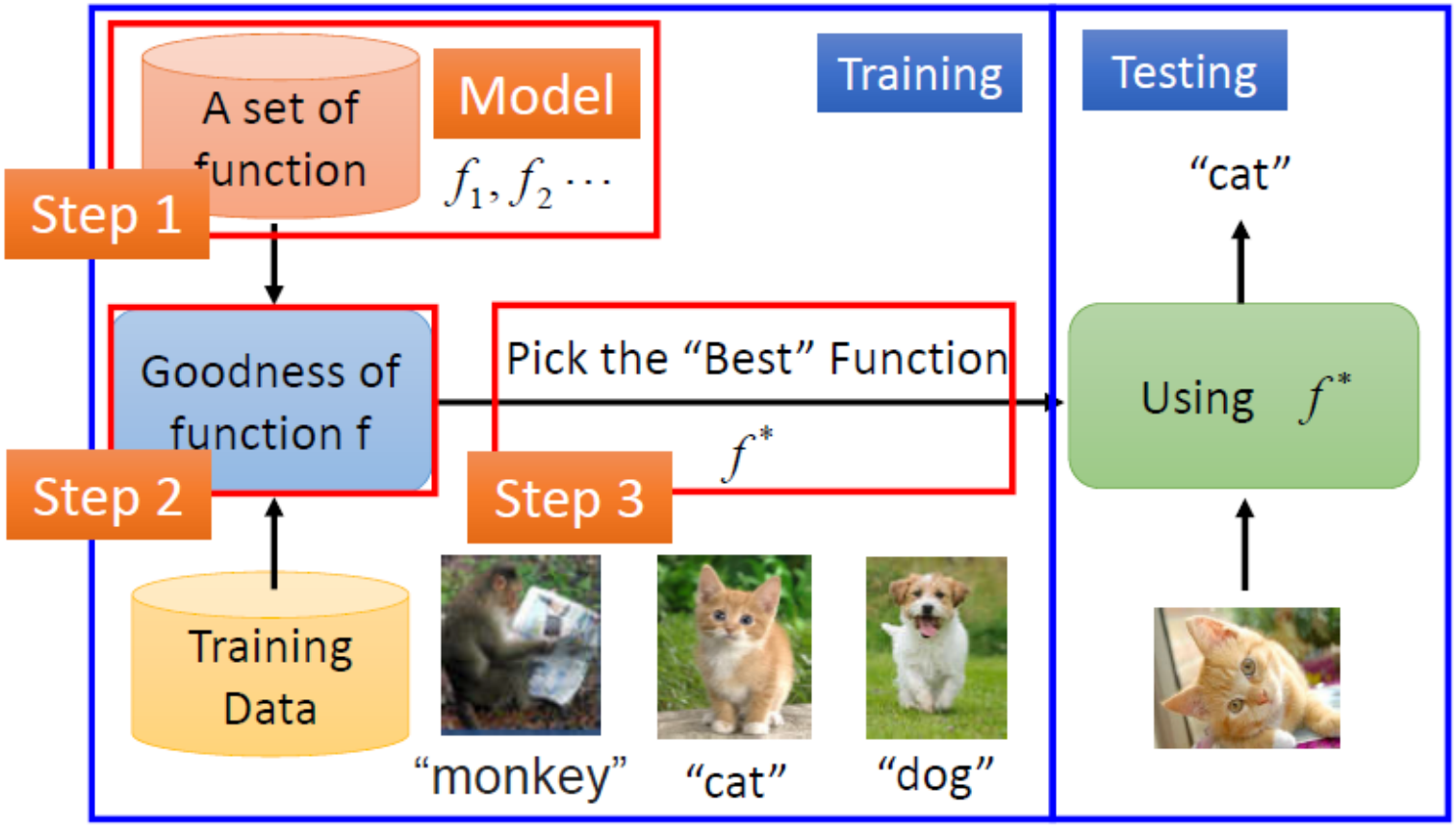
Supervised Learning

function input:   

function output: "monkey" "cat" "dog"

Framework

Image Recognition:



传统机器学习

目前机器学习解决问题的思路

中间的三部分，概括起来就是特征表达。良好的特征表达，对最终算法的准确性起了非常关键的作用，而且系统主要的计算和测试工作都耗在这一大部分。但这块实际中一般都是人工完成的。

靠人工提取特征

手工选取特征不太好，那么能不能自动地学习一些特征呢？能！

Deep Learning (Unsupervised Feature Learning)

Three Steps for Deep Learning

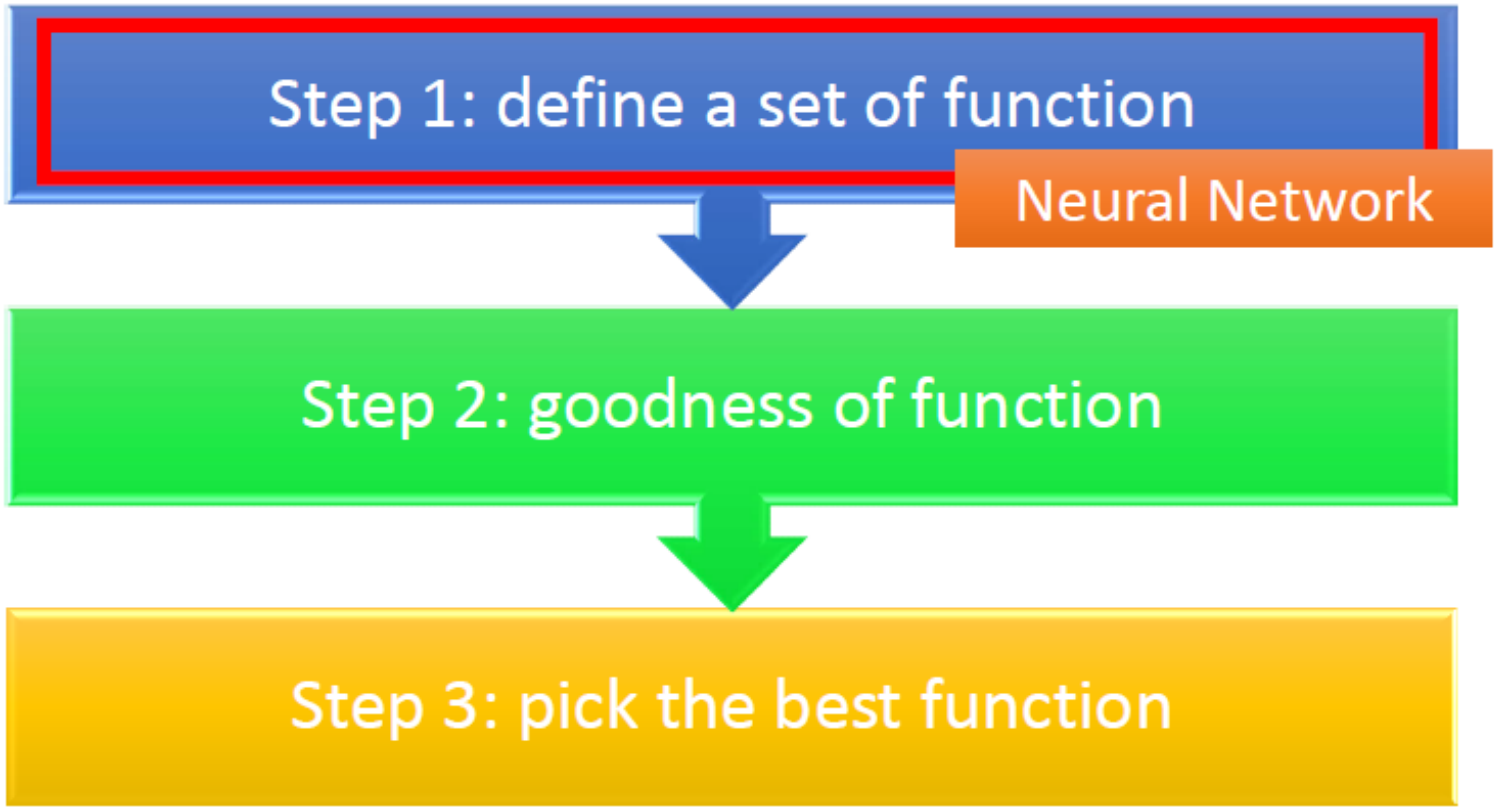
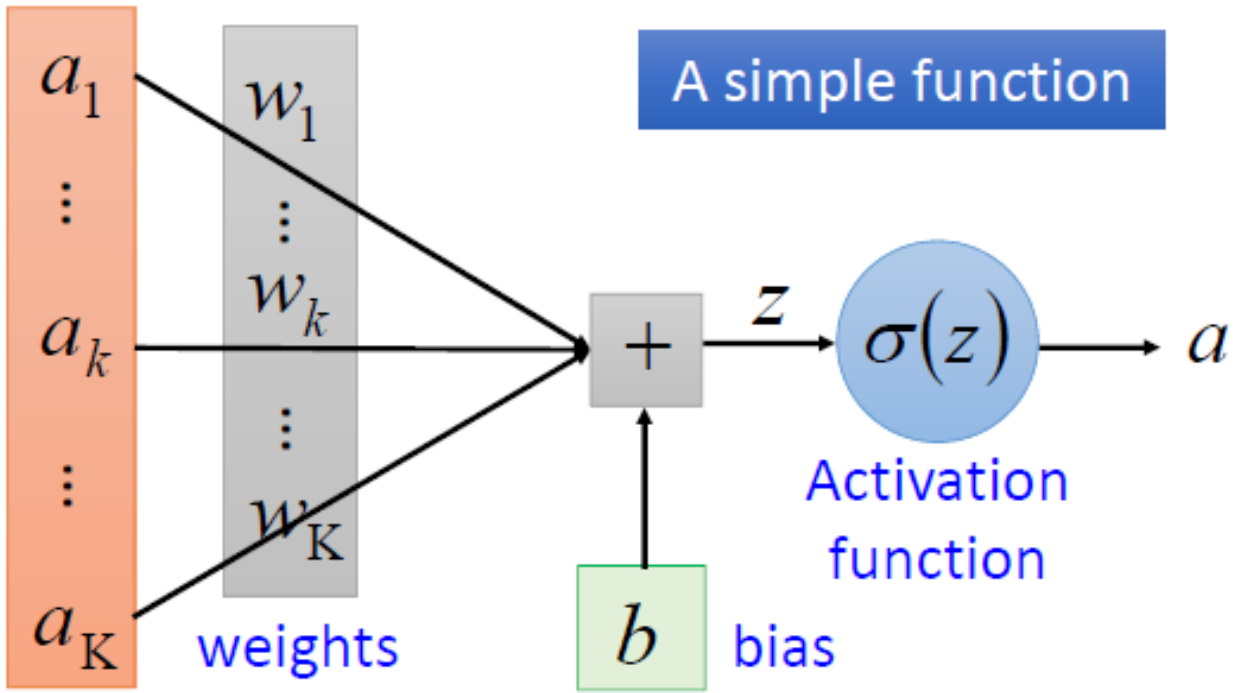


Table of Contents

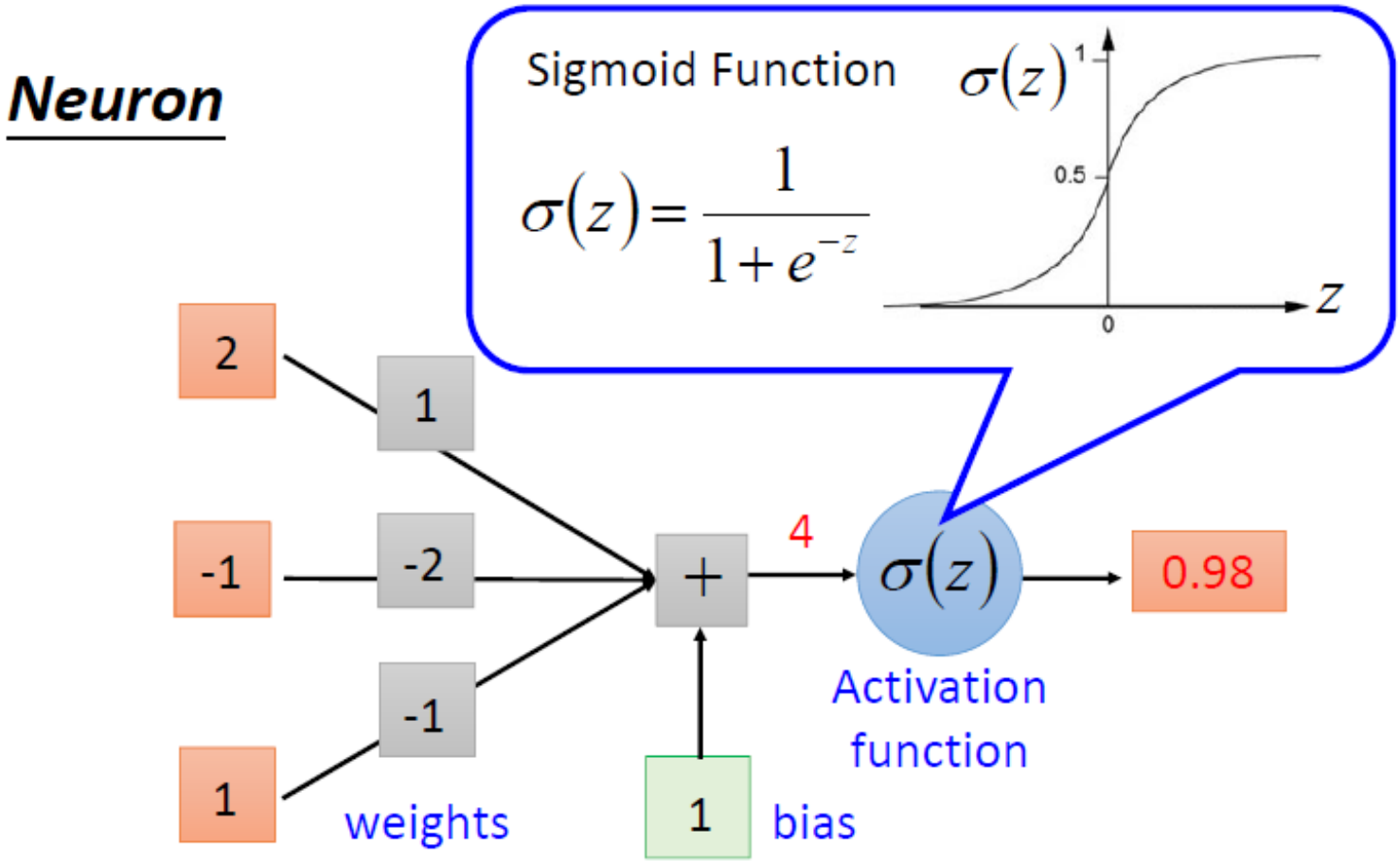
神经元（Neuron）

Neuron

$$z = a_1 w_1 + \cdots + a_k w_k + \cdots + a_K w_K + b$$



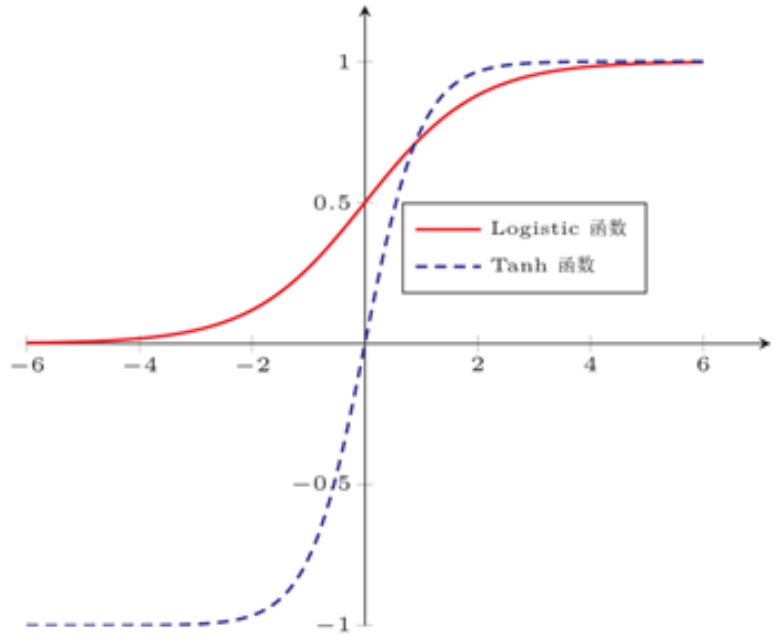
神经元（Neuron）


Neuron
Sigmoid Function
$\sigma(z) = \frac{1}{1+e^{-z}}$
$\sigma(z)$
1
0.5
0
z
2
1
-1
-2
-1
1
+
4
$\sigma(z)$
0.98
Activation function
weights
1
bias

激活函数

$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$

$$\tanh(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}$$



激活函数

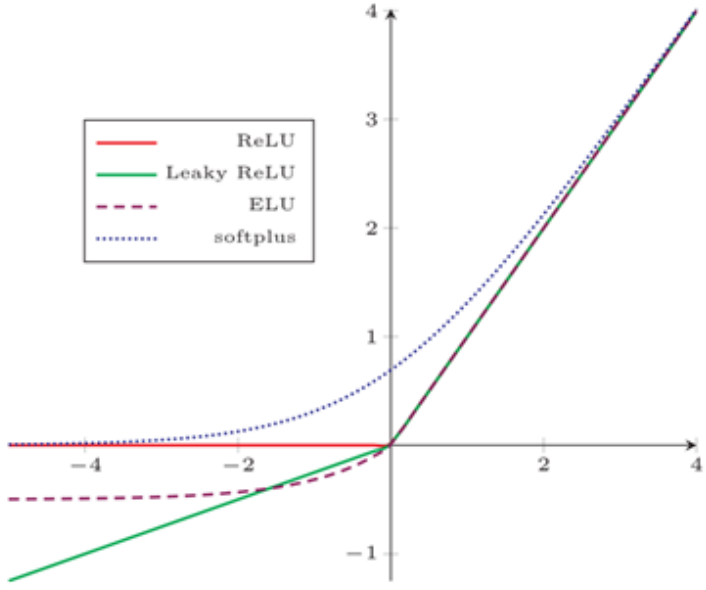
$$\mathrm{ReLU}(x) = \begin{cases} x & x \geq 0 \\ 0 & x < 0 \end{cases}$$
$$= \max(0, x).$$

$$\mathrm{LeakyReLU}(x) = \begin{cases} x & \text{if } x > 0 \\ \gamma x & \text{if } x \leq 0 \end{cases}$$
$$= \max(0, x) + \gamma \min(0, x)$$

$$\mathrm{PReLU}_i(x) = \begin{cases} x & \text{if } x > 0 \\ \gamma_i x & \text{if } x \leq 0 \end{cases}$$
$$= \max(0, x) + \gamma_i \min(0, x)$$

$$\mathrm{ELU}(x) = \begin{cases} x & \text{if } x > 0 \\ \gamma(\exp(x) - 1) & \text{if } x \leq 0 \end{cases}$$
$$= \max(0, x) + \min(0, \gamma(\exp(x) - 1))$$

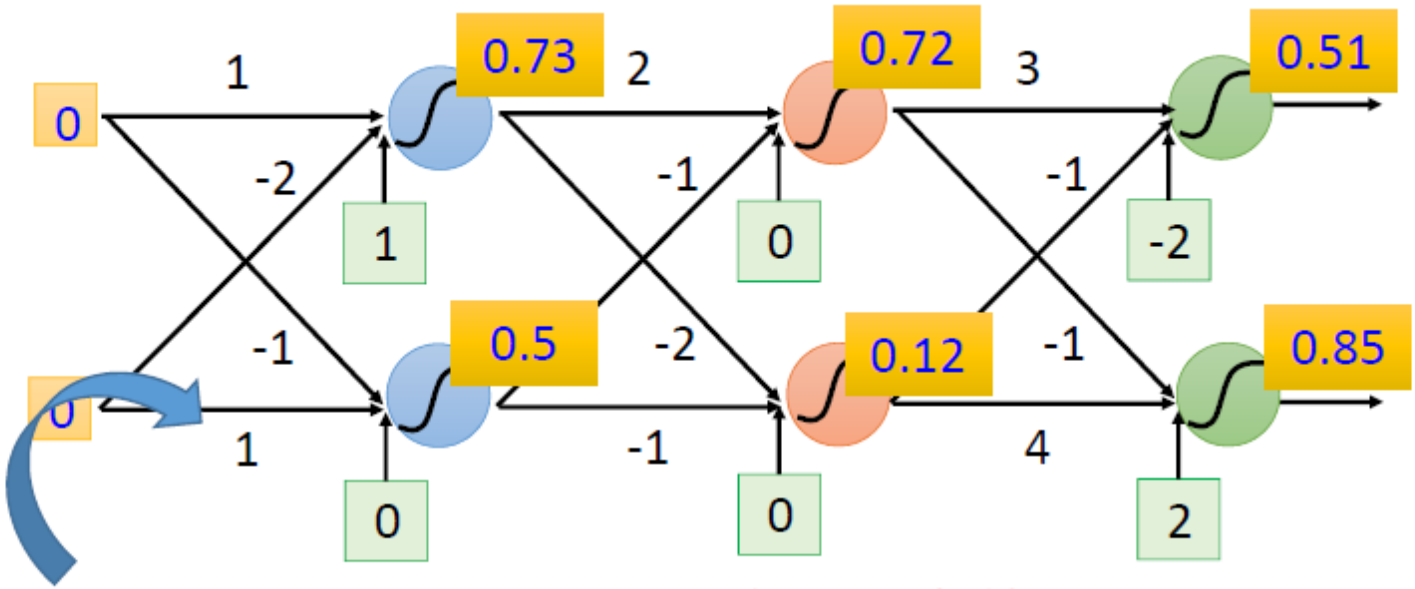
$$\mathrm{softplus}(x) = \log(1 + \exp(x))$$



激活函数

激活函数	函数	导数
Logistic 函数	$f(x) = \frac{1}{1+\exp(-x)}$	$f'(x) = f(x)\big(1 - f(x)\big)$
Tanh 函数	$f(x) = \frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)}$	$f'(x) = 1 - f(x)^2$
ReLU	$f(x) = \max(0, x)$	$f'(x) = I(x > 0)$
ELU	$f(x) = \max(0, x) + \min\big(0, \gamma(\exp(x) - 1)\big)$	$f'(x) = I(x > 0) + I(x \leq 0) \cdot \gamma \exp(x)$
SoftPlus 函数	$f(x) = \log\big(1 + \exp(x)\big)$	$f'(x) = \frac{1}{1+\exp(-x)}$

Fully Connect Feedforward Network



This is a function.

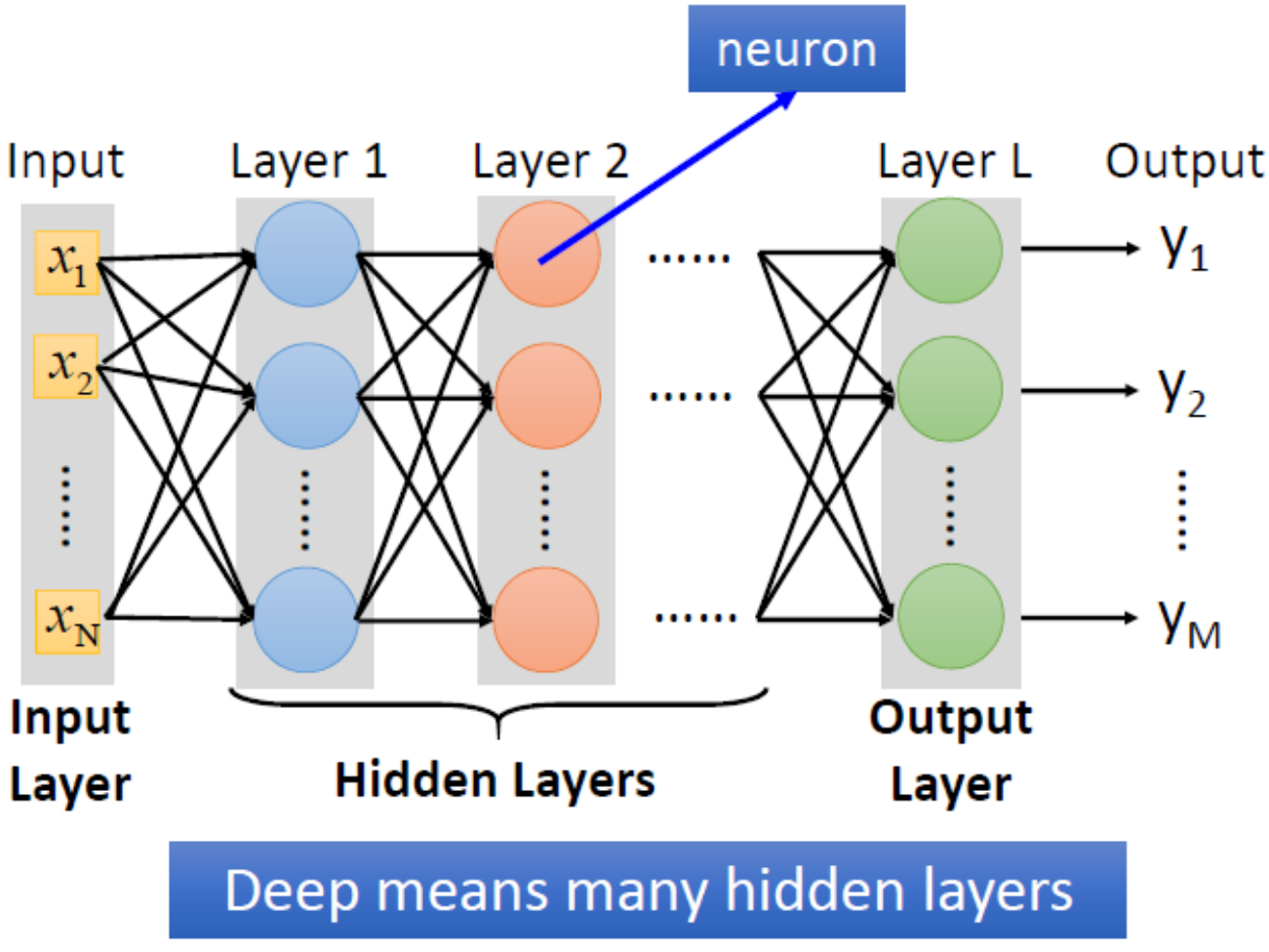
Input vector, output vector

$$f\left(\begin{bmatrix}1\\-1\end{bmatrix}\right)=\begin{bmatrix}0.62\\0.83\end{bmatrix} \quad f\left(\begin{bmatrix}0\\0\end{bmatrix}\right)=\begin{bmatrix}0.51\\0.85\end{bmatrix}$$

Given parameters θ, define a function

Given network structure, define ***a function set***

Fully Connect Feedforward Network



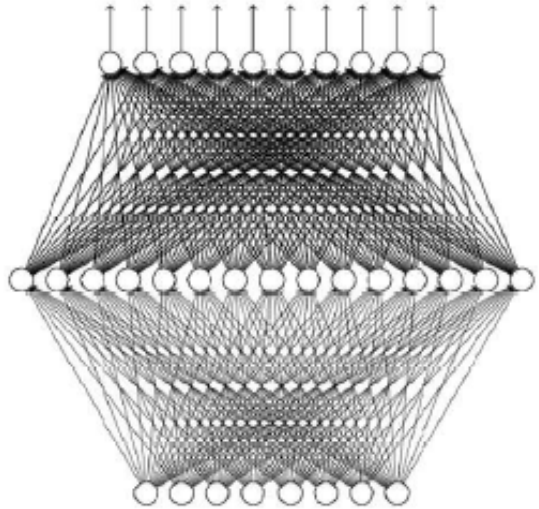
Why Deep? Universality Theorem

Any continuous function f

$$f : R^N \rightarrow R^M$$

Can be realized by a network with one hidden layer

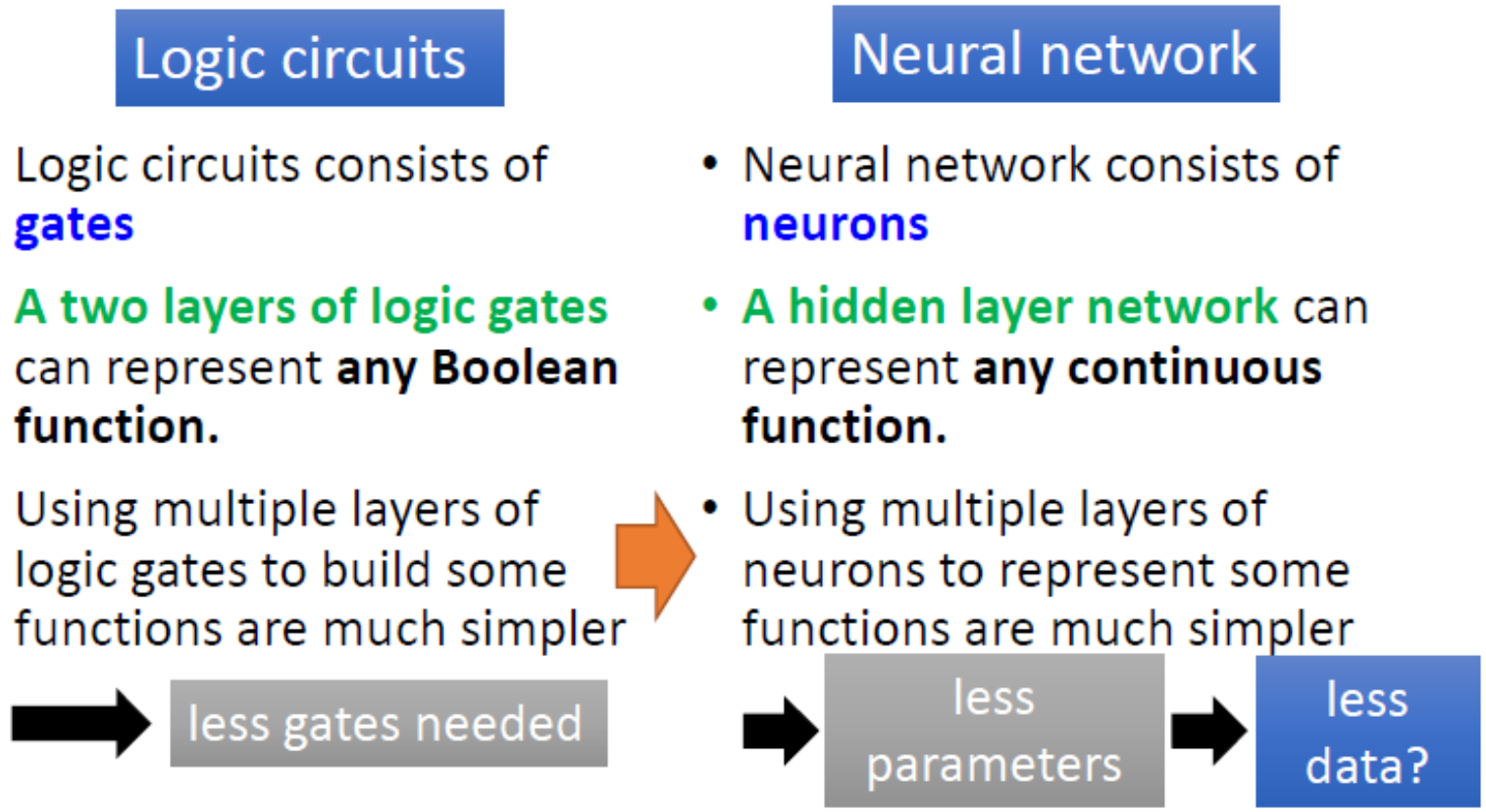
(given **enough** hidden neurons)



Reference for the reason:
http://neuralnetworksanddeeplearning.com/chap4.html

Why "Deep" neural network not "Fat" neural network?

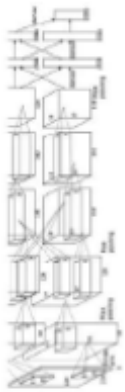
Why Deep? Analogy



Deep = Many hidden layers

http://cs231n.stanford.edu/slides/winter1516_lecture8.pdf

8 layers



16.4%

AlexNet (2012)

19 layers

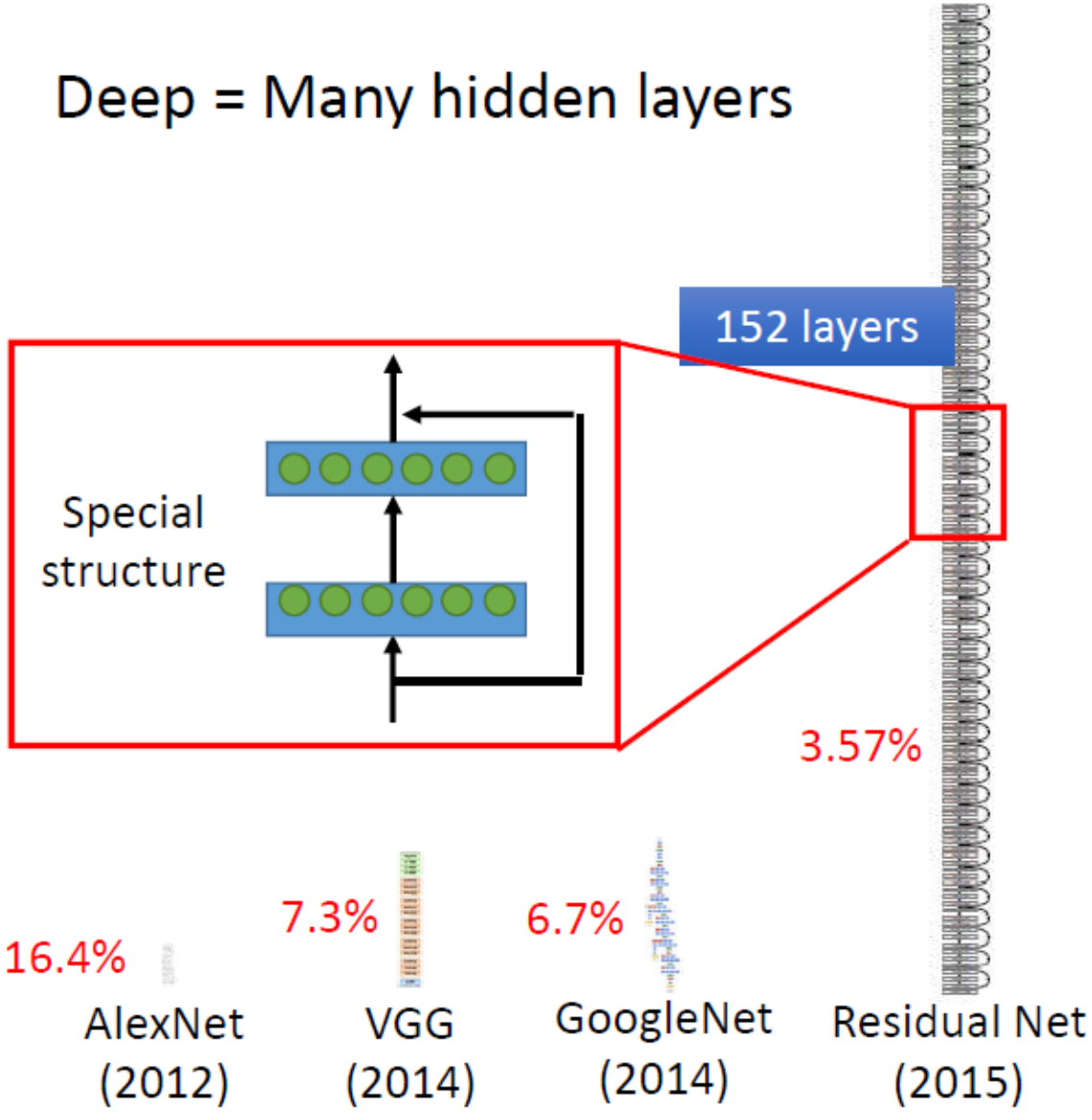
7.3%

VGG (2014)

22 layers

6.7%

GoogleNet (2014)


Deep = Many hidden layers
152 layers
Special structure
3.57%
16.4%
7.3%
6.7%
AlexNet (2012)
VGG (2014)
GoogleNet (2014)
Residual Net (2015)

Output Layer

- Softmax layer as the output layer

Probability:

- $1 > y_i > 0$
- $\sum_i y_i = 1$

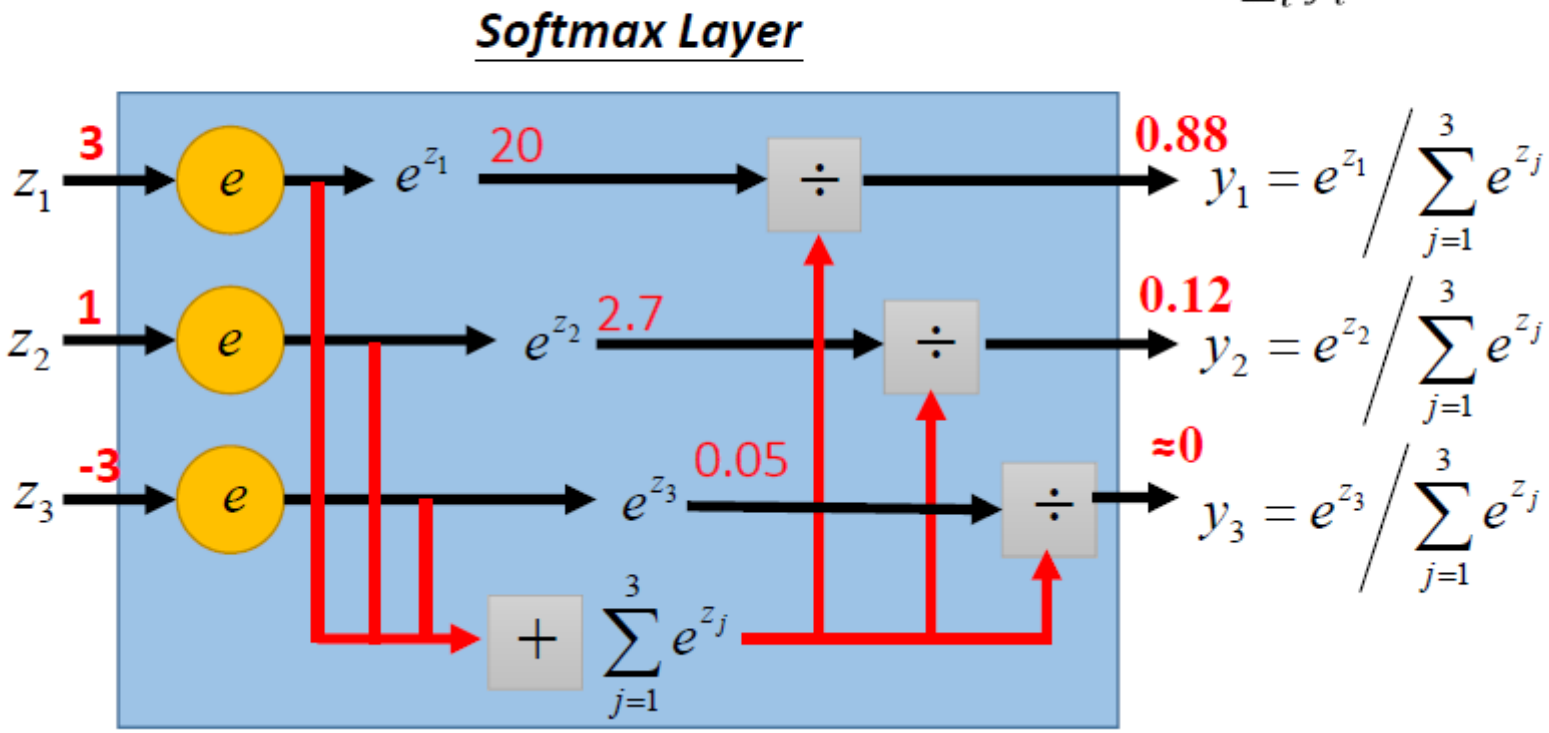
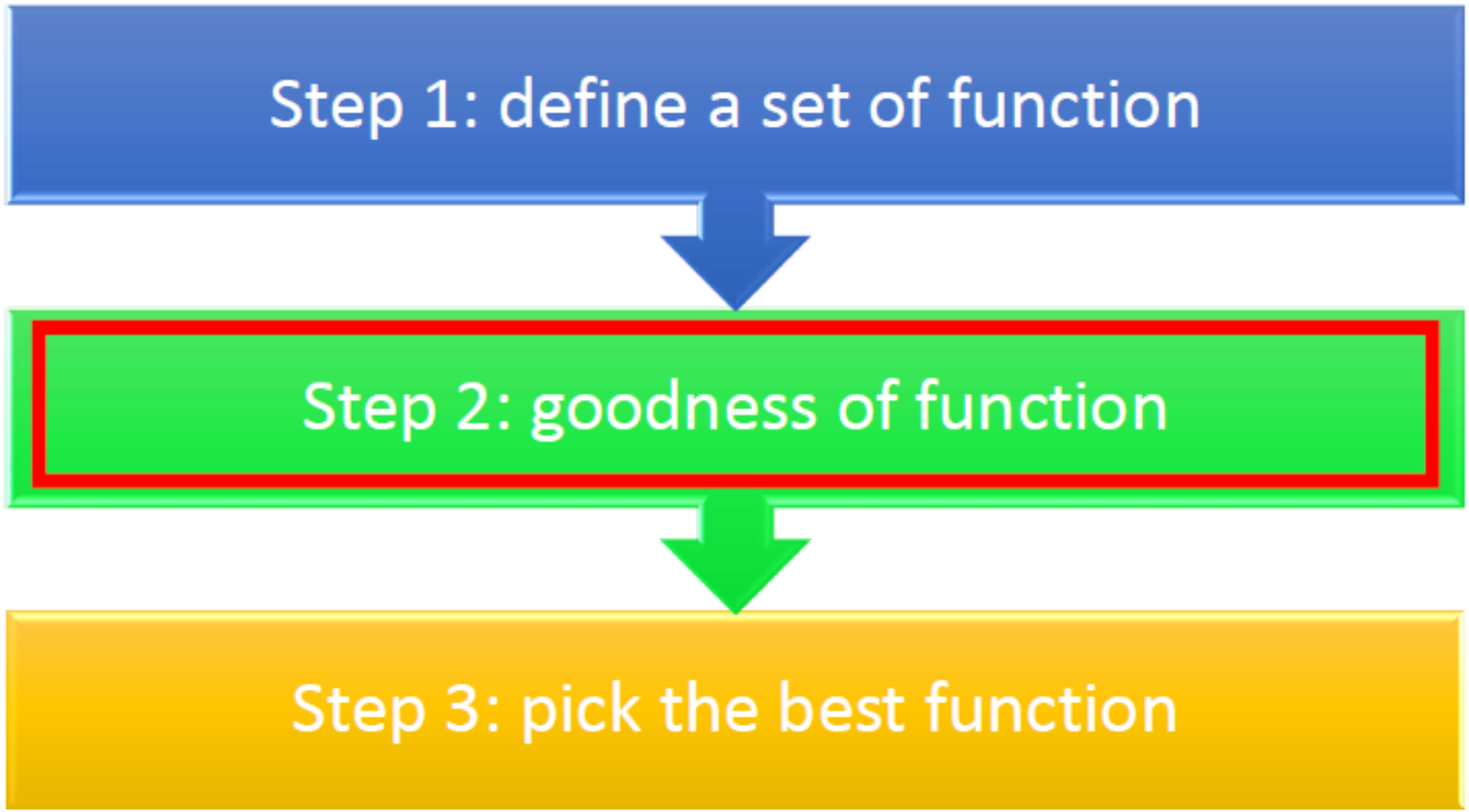


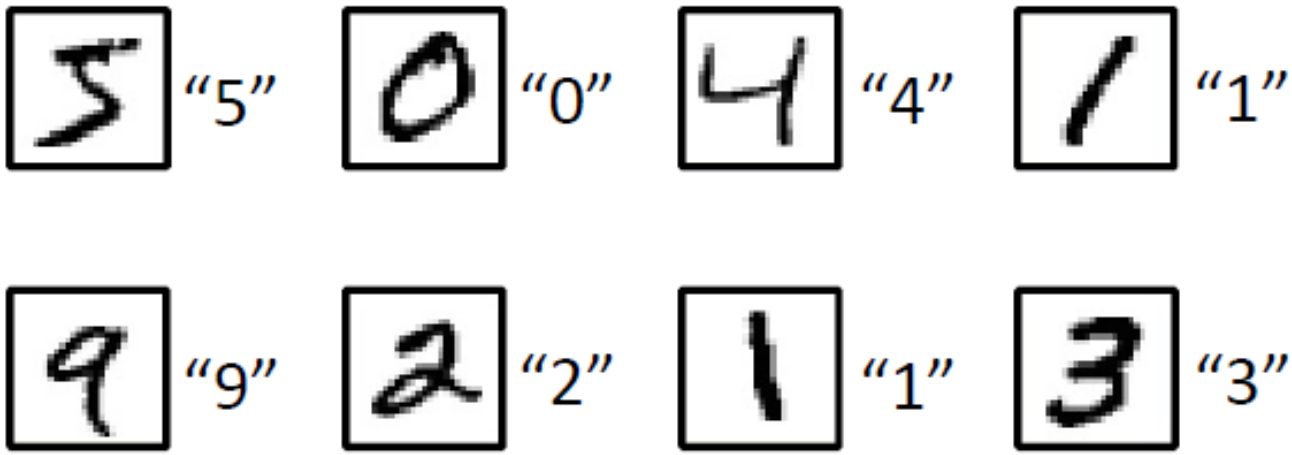
Table of Contents

Three Steps for Deep Learning



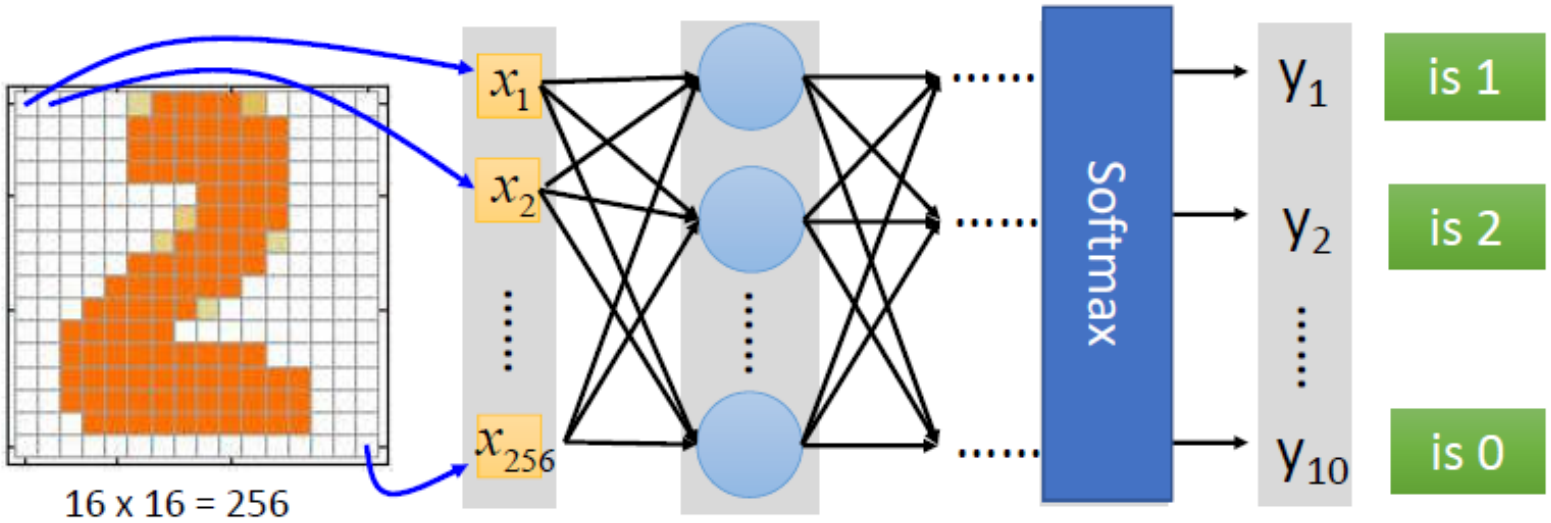
Training Data

- Preparing training data: images and their labels



The learning target is defined on the training data.

Learning Target



Ink → 1

No ink → 0

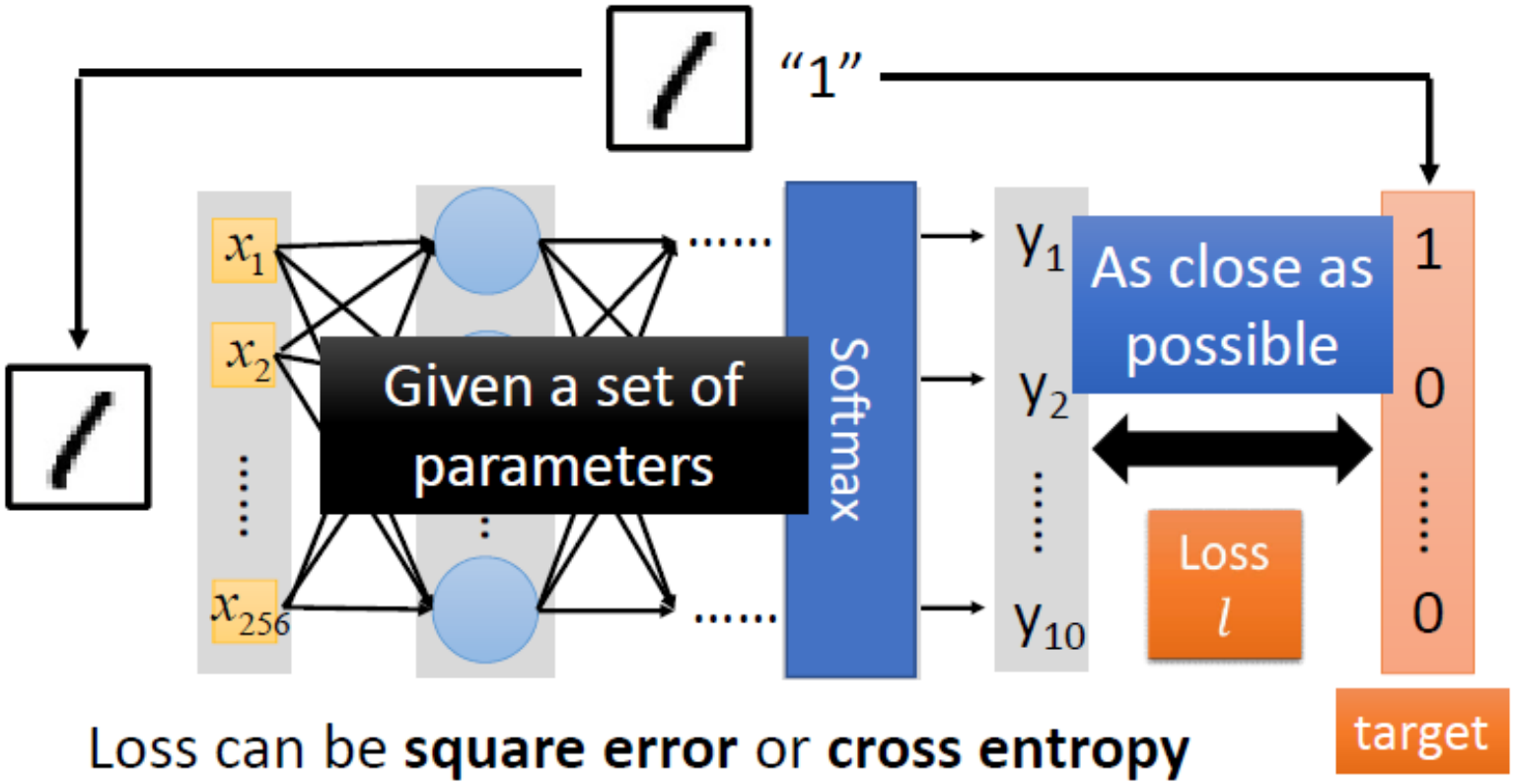
The learning target is .....

Input:  → y_1 has the maximum value

Input:  → y_2 has the maximum value

Loss

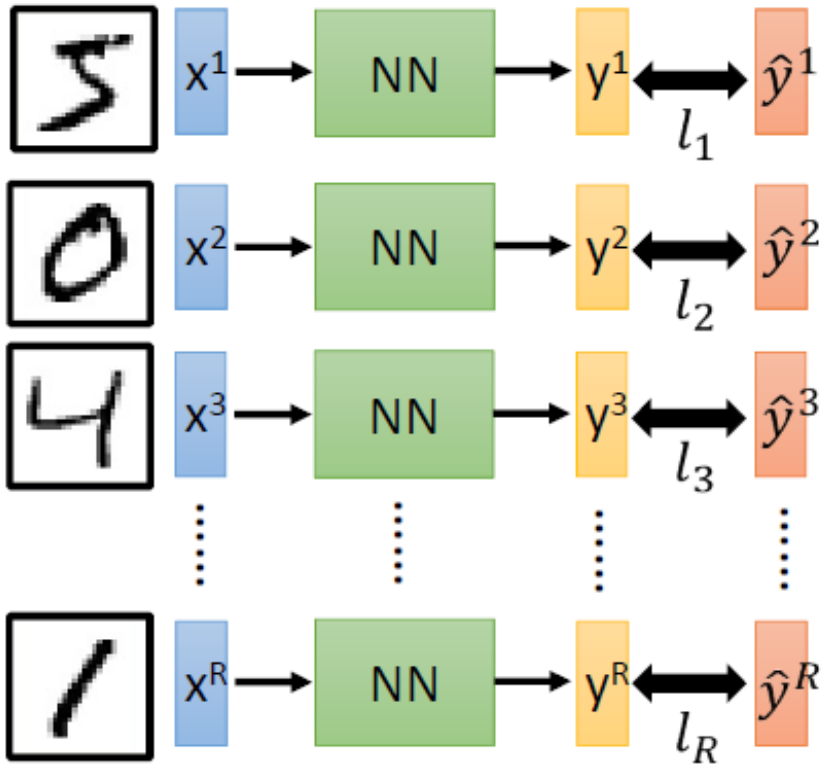
A good function should make the loss of all examples as small as possible.



Loss can be **square error** or **cross entropy** between the network output and target

Total Loss

For all training data ...



Total Loss:

$$L = \sum_{r=1}^{R} l_r$$

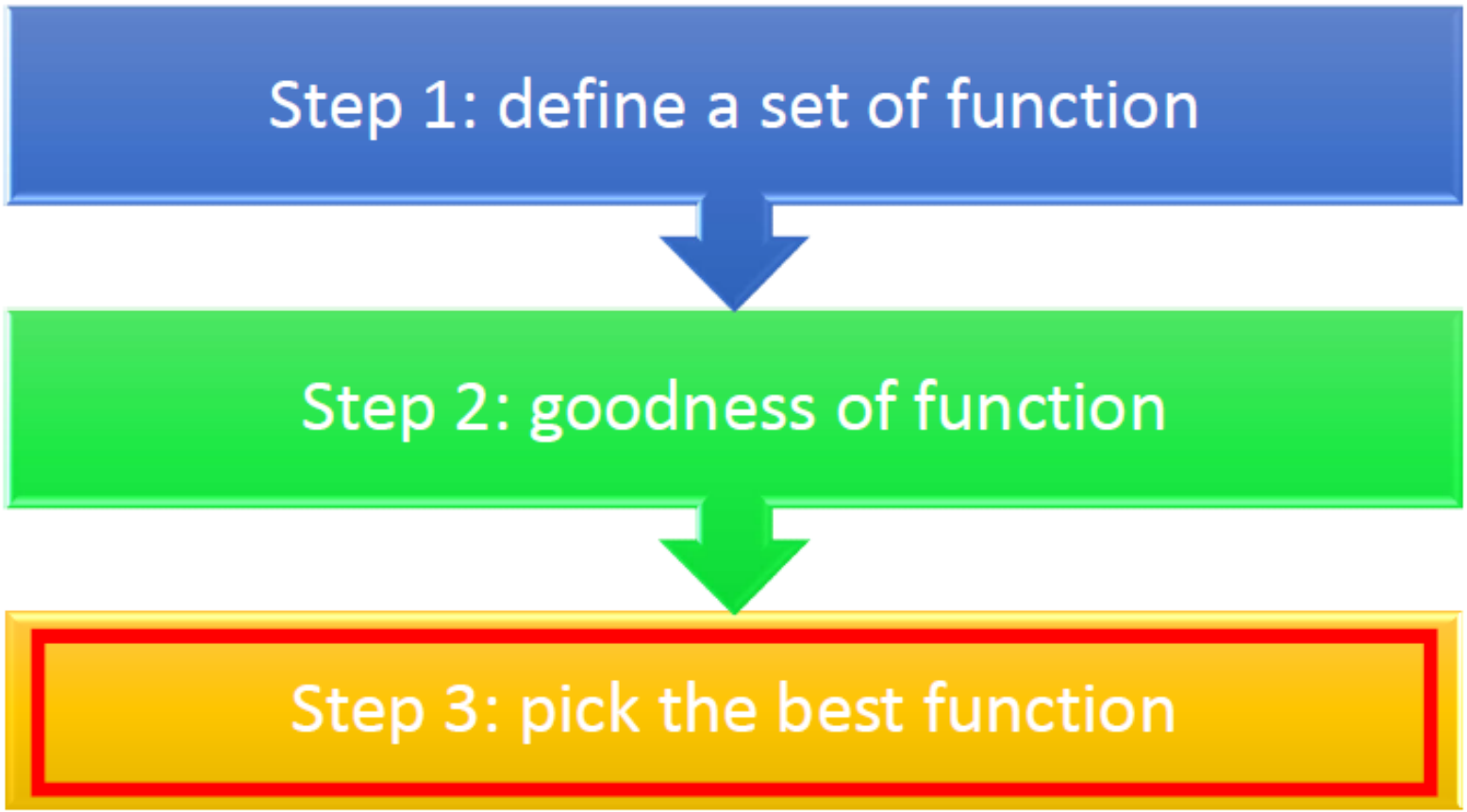
As small as possible

Find ***a function in function set*** that minimizes total loss L

Find ***the network parameters*** θ^* that minimize total loss L

Table of Contents

Three Steps for Deep Learning



How to pick the best function

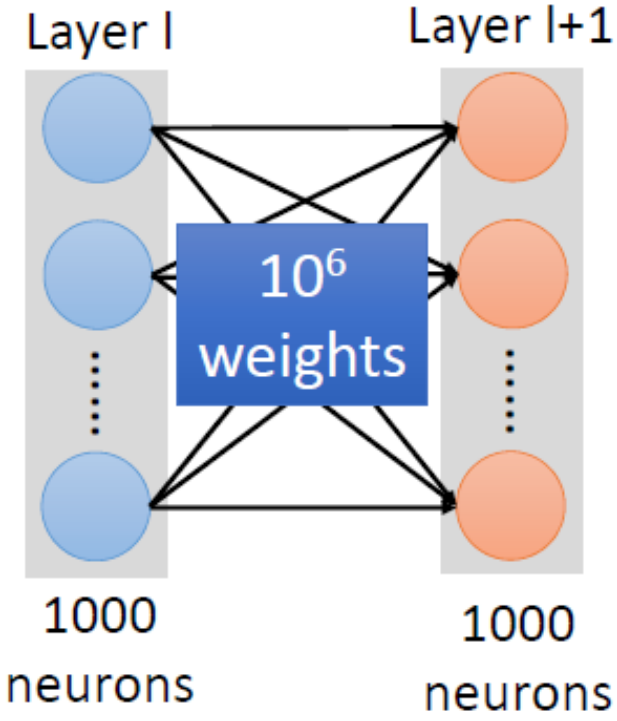
Find ***network parameters*** θ^* that minimize total loss L

Enumerate all possible values

Network parameters $\theta = \{w_1, w_2, w_3 \cdots, b_1, b_2, b_3, \cdots\}$

Millions of parameters

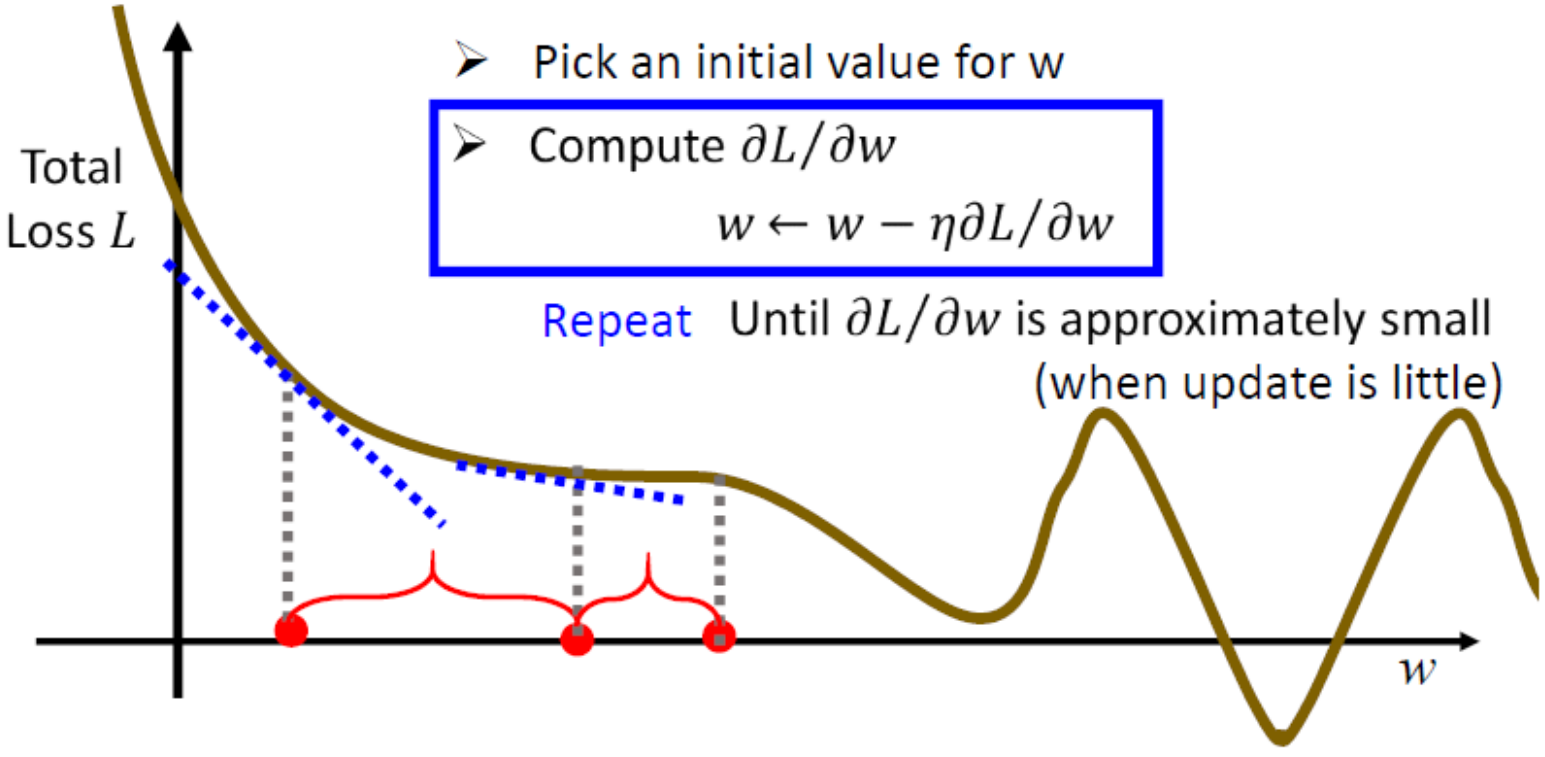
E.g. speech recognition: 8 layers and 1000 neurons each layer



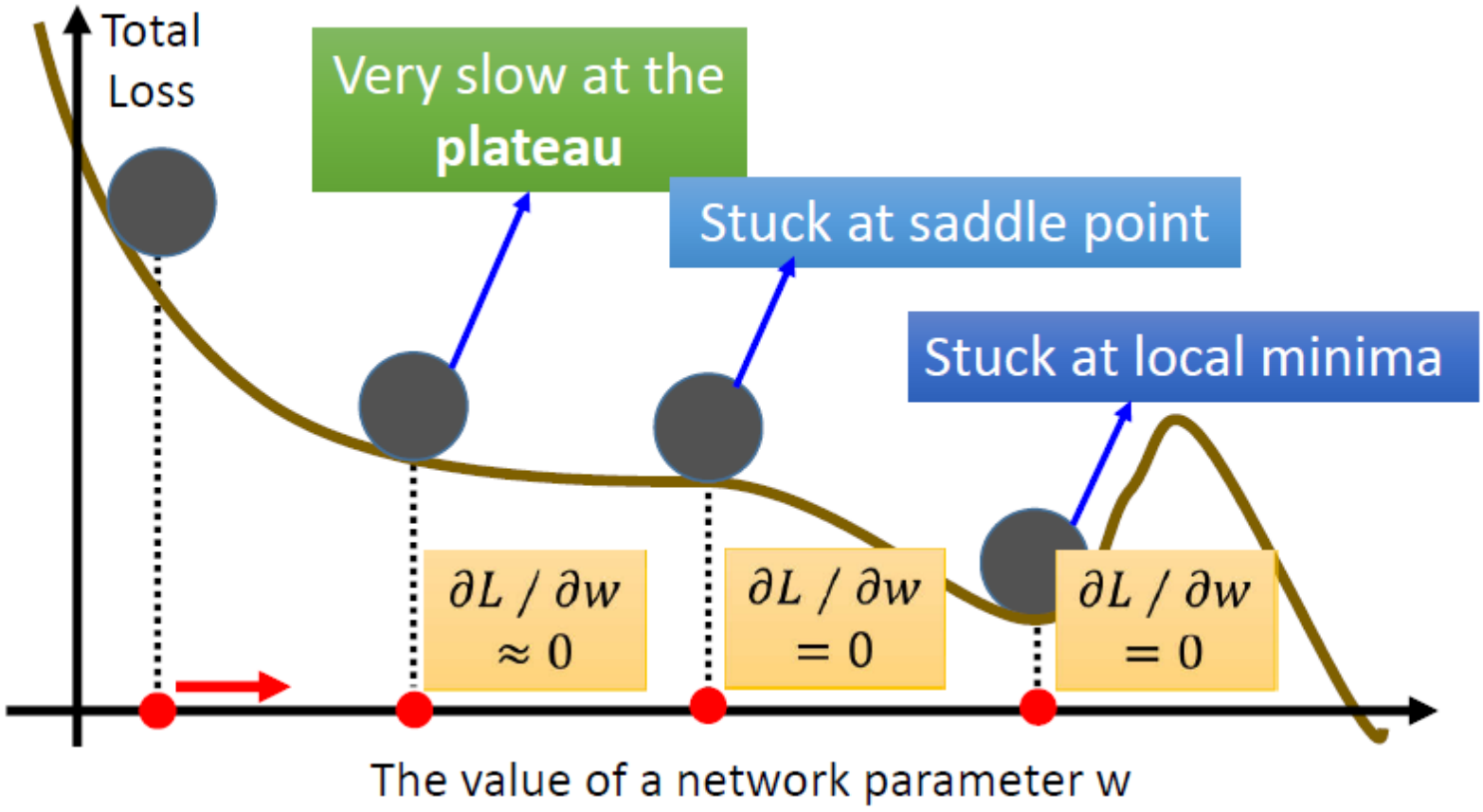
Gradient Descent

Network parameters $\theta = \{w_1, w_2, \cdots, b_1, b_2, \cdots\}$

Find ***network parameters*** θ^* that minimize total loss L

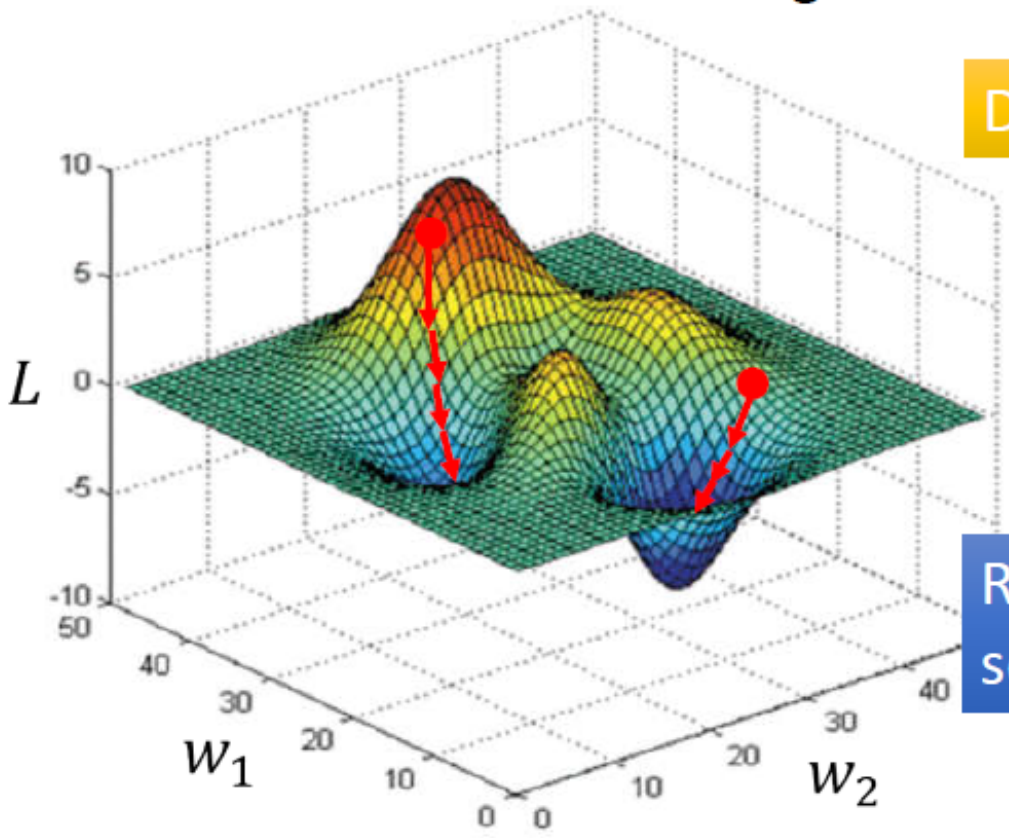


Local Minima



Local Minima

- Gradient descent never guarantee global minima



Different initial point

Reach different minima, so different results

Backpropagation

- Backpropagation: an efficient way to compute $\partial L/\partial w$ in neural network

theano

Caffe

Microsoft
CNTK

Deep Learning library produced by Amazon
DSSTNE

mxnet

libdnn
台大周伯威
同學開發

Ref: https://www.youtube.com/watch?v=ibJpTrp5mcE

TensorFlow 2.0, PyTorch

计算图

机器学习系统：设计和实现

https://openmlsys.github.io/index.html

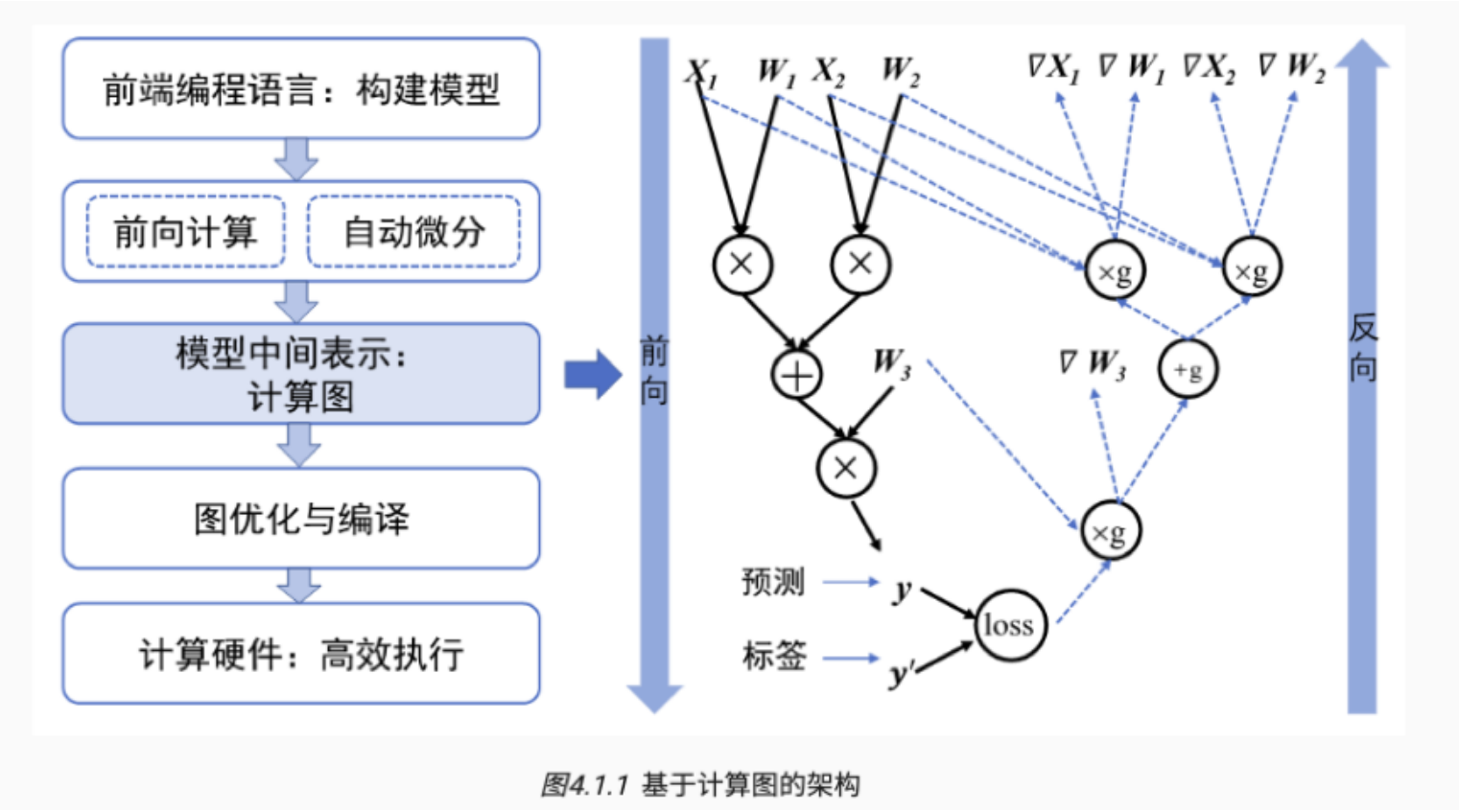


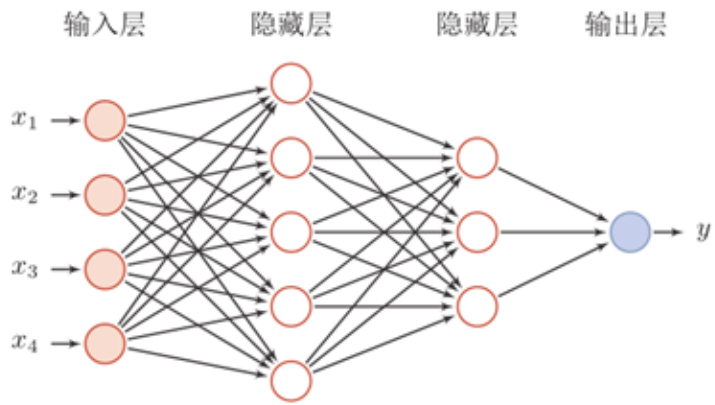
图4.1.1 基于计算图的架构

自动微分：将计算机程序中的运算操作分解为一个有限的基本操作集合，且集合中基本操作的求导规则均为已知，在完成每一个基本操作的求导后，使用链式法则将结果组合得到整体程序的求导结果。

Table of Contents

前馈神经网络（Feedforward Neural Network）

- 前馈神经网络（全连接神经网络、多层感知器）
  - 各神经元分别属于不同的层，层内无连接。
  - 相邻两层之间的神经元全部两两连接。
  - 整个网络中无反馈，信号从输入层向输出层单向传播，可用一个有向无环图表示。



前馈神经网络

- 给定一个前馈神经网络，我们用下面的记号来描述这样网络。

  - L：表示神经网络的层数；
  - n^l：表示第l层神经元的个数；
  - $f_l(\cdot)$：表示l层神经元的激活函数；
  - $W^{(l)} \in \mathbb{R}^{n^l \times n^{l-1}}$：表示$l-1$层到第$l$层的权重矩阵；
  - $\mathbf{b}^{(l)} \in \mathbb{R}^{n^l}$：表示$l-1$层到第$l$层的偏置；
  - $\mathbf{z}^{(l)} \in \mathbb{R}^{n^l}$：表示$l$层神经元的净输入（净活性值）；
  - $\mathbf{a}^{(l)} \in \mathbb{R}^{n^l}$：表示$l$层神经元的输出（活性值）。

前馈计算

前馈神经网络通过下面公式进行信息传播。

$$\mathbf{z}^{(l)} = W^{(l)} \cdot \mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}$$

$$\mathbf{a}^{(l)} = f_l(\mathbf{z}^{(l)})$$

上述公式也可以合并写为：

$$\mathbf{z}^{(l)} = W^{(l)} \cdot f_l(\mathbf{z}^{(l-1)}) + \mathbf{b}^{(l)}$$

这样，前馈神经网络可以通过逐层的信息传递，得到网络最后的输出 $\mathbf{a}^{L}$。

$$\mathbf{x} = \mathbf{a}^{(0)} \to \mathbf{z}^{(1)} \to \mathbf{a}^{(1)} \to \mathbf{z}^{(2)} \to \cdots \to \mathbf{a}^{(L-1)} \to \mathbf{z}^{(L)} \to \mathbf{a}^{(L)} = y$$

将前馈网络应用于机器学习

给定一组样本 $(\mathbf{x}^{(i)}, y^{(i)}), 1 \leq i \leq N$，用前馈神经网络的输出为 $f(\mathbf{x}|\mathbf{w}, \mathbf{b})$，目标函数为：

$$J(W, \mathbf{b}) = \sum_{i=1}^{N} L(y^{(i)}, f(\mathbf{x}^{(i)}|W, \mathbf{b})) + \frac{1}{2}\lambda||W||_F^2,$$
$$= \sum_{i=1}^{N} J(W, \mathbf{b}; \mathbf{x}^{(i)}, y^{(i)}) + \frac{1}{2}\lambda||W||_F^2,$$

这里，W 和 $\mathbf{b}$ 包含了每一层的权重矩阵和偏置向量，

$||W||_F^2$ 是正则化项，用来防止过拟合：λ 是为正数的超参数。λ 越大，W 越接近于 0。这里的 $||W||_F^2$ 一般使用 Frobenius 范数:

$$||W||_F^2 = \sum_{l=1}^{L} \sum_{i=1}^{m^{(l)}} \sum_{j=1}^{m^{(l-1)}} (W_{ij}^{(l)})^2$$

参数估计

我们的目标是最小化 $J(W, \mathbf{b}; \mathbf{x}, y)$。如果采用梯度下降方法，我们可以用如下方法更新参数：

$$
\begin{aligned}
W^{(l)} &= W^{(l)} - \alpha \frac{\partial J(W, \mathbf{b})}{\partial W^{(l)}}, \\
&= W^{(l)} - \alpha \sum_{i=1}^{N} \left(\frac{\partial J(W, \mathbf{b}; \mathbf{x}^{(i)}, y^{(i)})}{\partial W^{(l)}} \right) - \lambda W, \\
\mathbf{b}^{(l)} &= \mathbf{b}^{(l)} - \alpha \frac{\partial J(W, \mathbf{b}; \mathbf{x}^{(i)}, y^{(i)})}{\partial \mathbf{b}^{(l)}}, \\
&= \mathbf{b}^{(l)} - \alpha \sum_{i=1}^{N} \left(\frac{\partial J(W, \mathbf{b}; \mathbf{x}^{(i)}, y^{(i)})}{\partial \mathbf{b}^{(l)}} \right),
\end{aligned}
$$

这里 α 是参数的更新率。

反向传播算法

我们首先来看下 $\frac{\partial J(W,\mathbf{b};\mathbf{x},y)}{\partial W^{(l)}}$ 怎么计算。

根据公式24定义的链式法则，$\frac{\partial J(W,\mathbf{b};\mathbf{x},y)}{\partial W_{ij}^{(l)}}$ 可以写为

$$\frac{\partial J(W,\mathbf{b};\mathbf{x},y)}{\partial W_{ij}^{(l)}} = \left(\frac{\partial J(W,\mathbf{b};\mathbf{x},y)}{\partial \mathbf{z}^{(l)}}\right)^{\top} \frac{\partial \mathbf{z}^{(l)}}{\partial W_{ij}^{(l)}}.$$

对于第 l 层，我们定义一个**误差项** $\delta^{(l)} = \frac{\partial J(W,\mathbf{b};\mathbf{x},y)}{\partial \mathbf{z}^{(l)}} \in \mathbb{R}^{n^{(l)}}$ 为目标函数关于第 l 层的神经元 $\mathbf{z}^{(l)}$ 的偏导数，来表示第 l 层的神经元对最终误差的影响。$\delta^{(l)}$ 也反映了最终的输出对第 l 层的神经元对最终误差的敏感程度。

若 $X \in \mathbb{R}^{p\times q}$ 为矩阵，$\mathbf{y} = g(X) \in \mathbb{R}^{s}$，$z = f(\mathbf{y}) \in \mathbb{R}$，则

$$\frac{\partial z}{\partial X_{ij}} = \left(\frac{\partial z}{\partial \mathbf{y}}\right)^{\top} \frac{\partial \mathbf{y}}{\partial X_{ij}} \tag{24}$$

反向传播算法

因为 $\mathbf{z}^{(l)} = W^{(l)} \cdot \mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}$，所以

$$\frac{\partial \mathbf{z}^{(l)}}{\partial W_{ij}^{(l)}} = \frac{\partial (W^{(l)} \cdot \mathbf{a}^{(l-1)} + \mathbf{b}^{(l)})}{\partial W_{ij}^{(l)}} = \begin{bmatrix} 0 \\ \vdots \\ a_j^{(l-1)} \\ \vdots \\ 0 \end{bmatrix} \leftarrow \text{第 } i \text{ 行}$$

因此，

$$\frac{\partial J(W, \mathbf{b}; \mathbf{x}, y)}{\partial W_{ij}^{(l)}} = \delta_i^{(l)} a_j^{(l-1)}$$

反向传播算法

因此

$$\frac{\partial J(W,\mathbf{b};\mathbf{x},y)}{\partial W_{ij}^{(l)}} = \left(\frac{\partial J(W,\mathbf{b};\mathbf{x},y)}{\partial \mathbf{z}^{(l)}}\right)^{\top} \frac{\partial \mathbf{z}^{(l)}}{\partial W_{ij}^{(l)}}.$$

可以等价写为:

$$\frac{\partial J(W,\mathbf{b};\mathbf{x},y)}{\partial W^{(l)}} = \delta^{(l)}(\mathbf{a}^{(l-1)})^{\top}.$$

同理可得,

$$\frac{\partial J(W,\mathbf{b};\mathbf{x},y)}{\partial \mathbf{b}^{(l)}} = \delta^{(l)}.$$

误差项

现在我们再来看下第 l 层的**误差项** $\delta^{(l)}$ 怎么计算。

$$\begin{aligned}\delta^{(l)} &\triangleq \frac{\partial J(W,\mathbf{b};\mathbf{x},y)}{\partial \mathbf{z}^{(l)}} \\ &= \frac{\partial \mathbf{a}^{(l)}}{\partial \mathbf{z}^{(l)}} \cdot \frac{\partial \mathbf{z}^{(l+1)}}{\partial \mathbf{a}^{(l)}} \cdot \frac{\partial J(W,\mathbf{b};\mathbf{x},y)}{\partial \mathbf{z}^{(l+1)}} \\ &= \mathbf{diag}(f_l'(\mathbf{z}^{(l)})) \cdot (W^{(l+1)})^{\top} \cdot \delta^{(l+1)} \\ &= f_l'(\mathbf{z}^{(l)}) \odot ((W^{(l+1)})^{\top}\delta^{(l+1)}),\end{aligned}$$

其中 $\odot$ 是向量的点积运算符，表示每个元素相乘。

对角矩阵 A 可以记为 $diag(a)$，a 为一个 n 维向量，并满足 $A_{ii} = a_i$

$$\delta_j^L = \frac{\partial J(W,b;x,y)}{\partial z_j^L} = \frac{\partial J(W,b;x,y)}{\partial a_j^L}\frac{\partial a_j^L}{\partial z_j^L} = \frac{\partial J(W,b;x,y)}{\partial a_j^L} f_L'(z_j^L)$$

反向传播

- 从上式可以看出，第 l 层的误差项可以通过第 $l+1$ 层的误差项计算得到。这就是误差的**反向传播** (Backpropagation，BP)。反向传播算法的含义是：第 l 层的一个神经元的误差项 (或敏感性) 是所有与该神经元相连的第 $l+1$ 层的神经元的误差项的权重和。然后，在乘上该神经元激活函数的梯度。
- 在计算出每一层的误差项之后，我们就可以得到每一层参数的梯度。因此，前馈神经网络的训练过程可以分为以下三步:
  1. 先前馈计算每一层的状态和激活值，直到最后一层；
  2. 反向传播计算每一层的误差；
  3. 计算每一层参数的偏导数，并更新参数。

反向传播算法

算法 4.1: 基于随机梯度下降的反向传播算法

输入: 训练集 $\mathcal{D} = \{(\mathbf{x}^{(n)}, y^{(n)})\}_{n=1}^{N}$, 验证集 $\mathcal{V}$, 学习率 α, 正则化系数 λ, 网络层数 L, 神经元数量 $m^{(l)}$, $1 \leq l \leq L$.

随机初始化 $W, \mathbf{b}$;

repeat

  对训练集 $\mathcal{D}$ 中的样本随机重排序;

  for $n = 1 \cdots N$ **do**

    从训练集 $\mathcal{D}$ 中选取样本 $(\mathbf{x}^{(n)}, y^{(n)})$;

    前馈计算每一层的净输入 $\mathbf{z}^{(l)}$ 和激活值 $\mathbf{a}^{(l)}$，直到最后一层;

    反向传播计算每一层的误差 $\delta^{(l)}$; // 公式 (4.59)

    // 计算每一层参数的导数

    $\forall l, \quad \frac{\partial \mathcal{L}(\mathbf{y}^{(n)}, \hat{\mathbf{y}}^{(n)})}{\partial W^{(l)}} = \delta^{(l)} (\mathbf{a}^{(l-1)})^{\mathrm{T}}$; // 公式 (4.61)

    $\forall l, \quad \frac{\partial \mathcal{L}(\mathbf{y}^{(n)}, \hat{\mathbf{y}}^{(n)})}{\partial \mathbf{b}^{(l)}} = \delta^{(l)}$; // 公式 (4.62)

    // 更新参数

    $W^{(l)} \leftarrow W^{(l)} - \alpha(\delta^{(l)} (\mathbf{a}^{(l-1)})^{\mathrm{T}} + \lambda W^{(l)})$;

    $\mathbf{b}^{(l)} \leftarrow \mathbf{b}^{(l)} - \alpha \delta^{(l)}$;

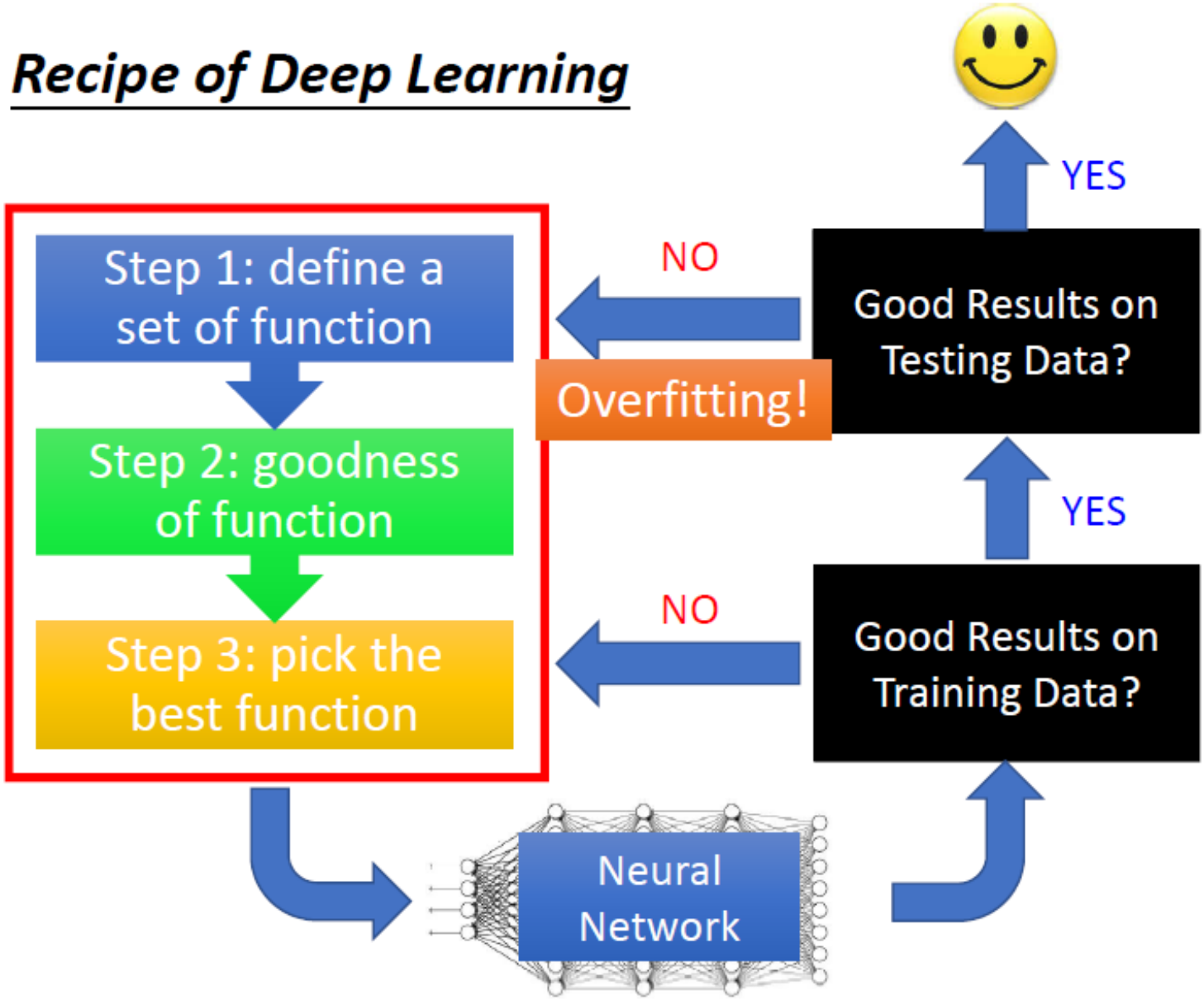
  end

until 神经网络模型在验证集 $\mathcal{V}$ 上的错误率不再下降;

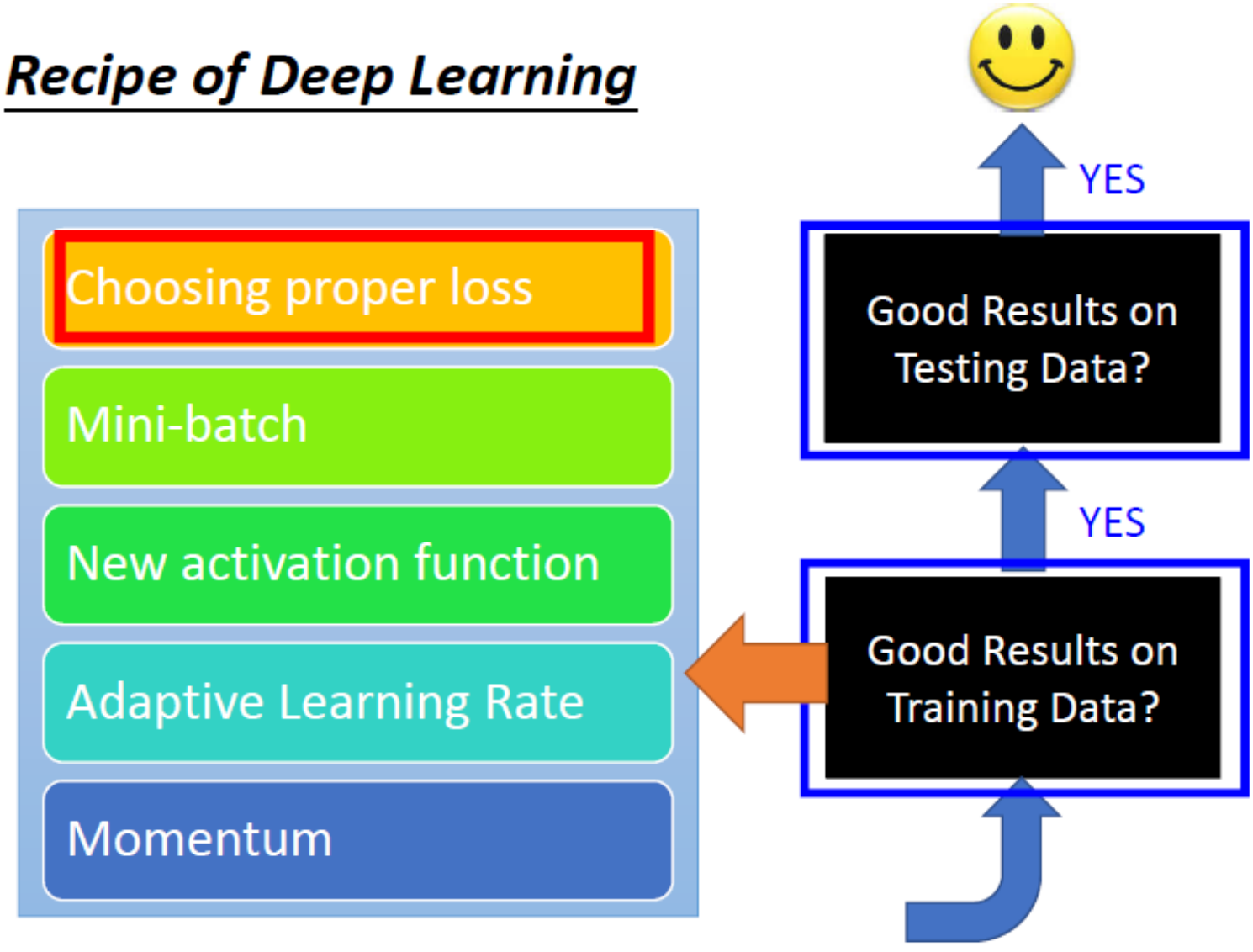
输出: $W, \mathbf{b}$

Table of Contents

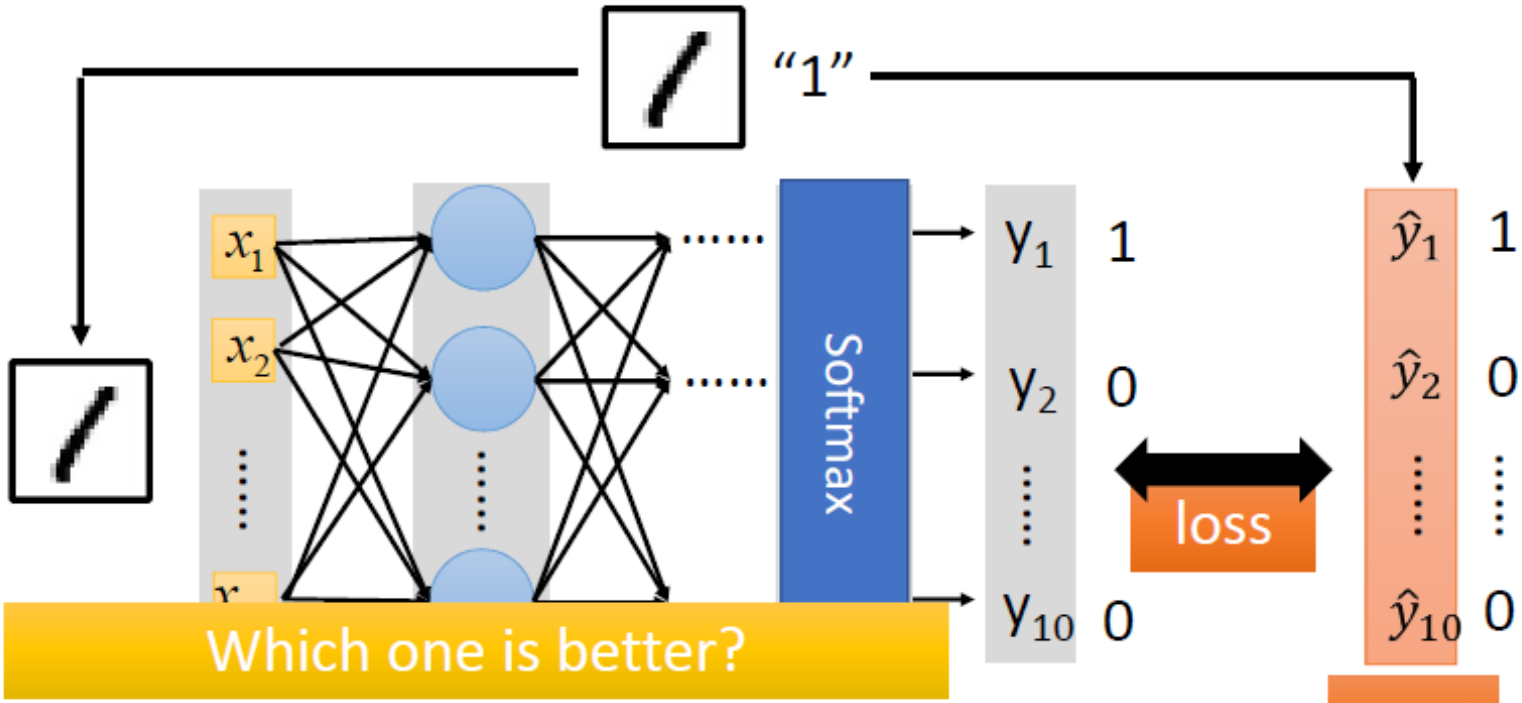
Recipe of Deep Learning



Recipe of Deep Learning



Choosing Proper Loss

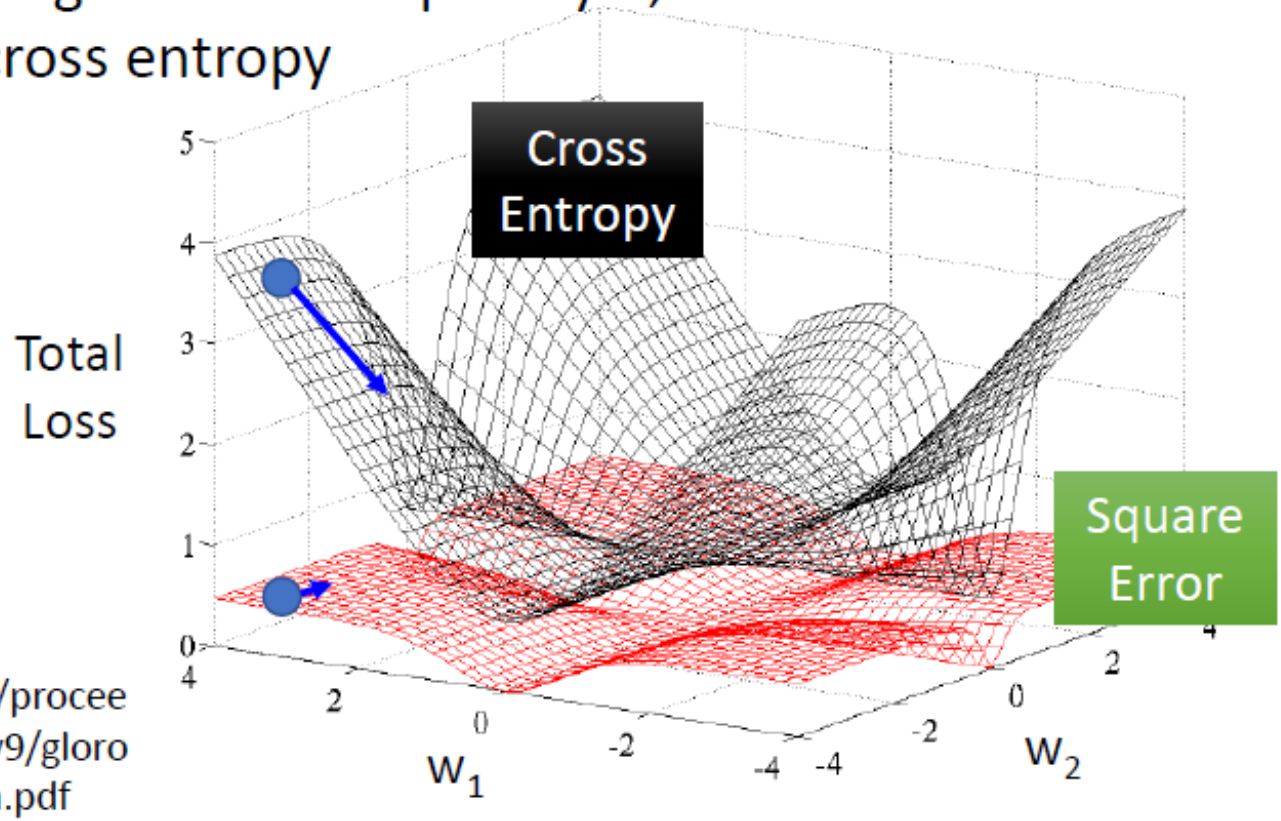


Square Error $\sum_{i=1}^{10}(y_i - \hat{y}_i)^2$ =0

Cross Entropy $-\sum_{i=1}^{10}\hat{y}_i ln y_i$ =0

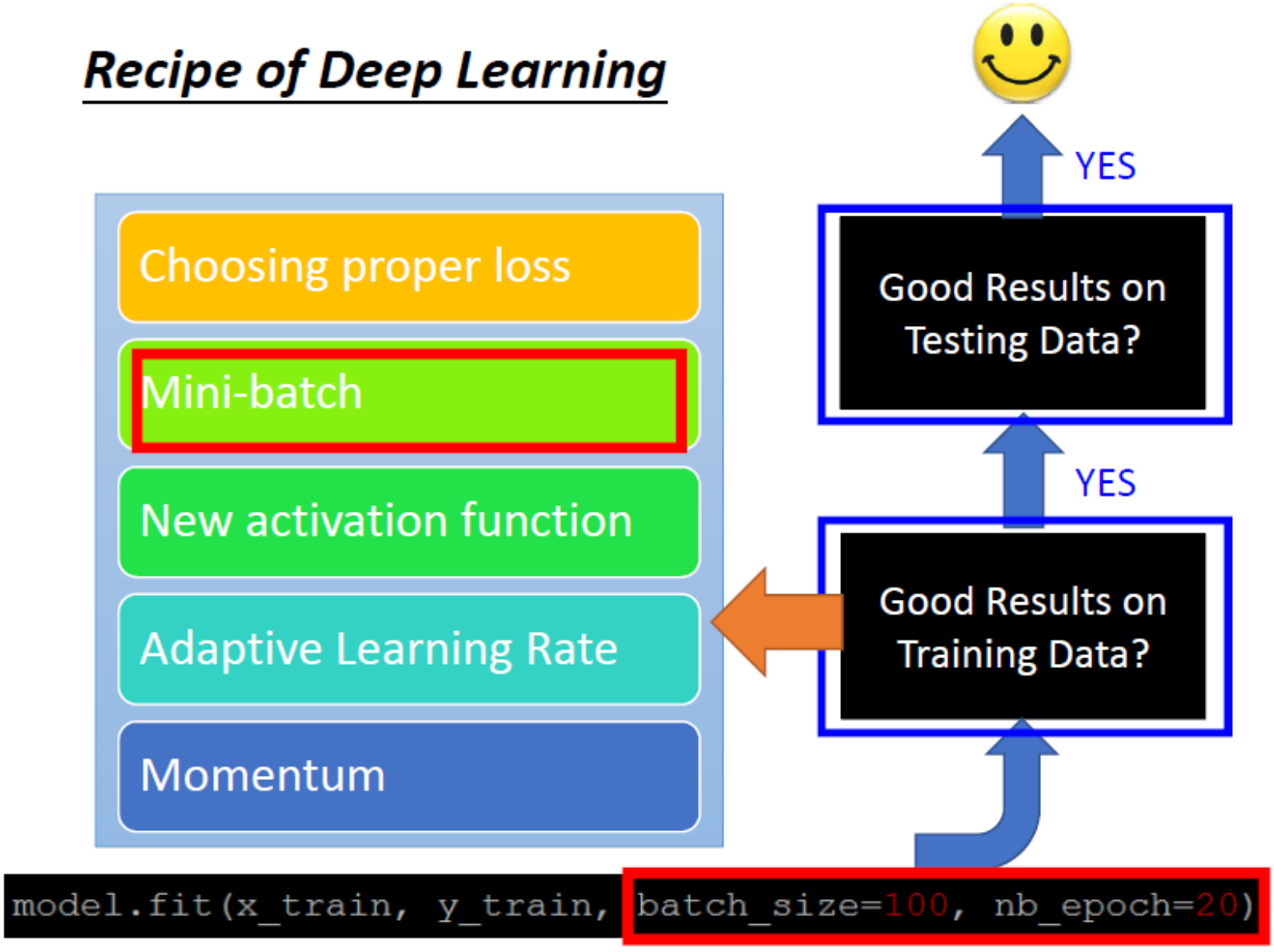
Choosing Proper Loss

When using softmax output layer, choose cross entropy



http://jmlr.org/proceedings/papers/v9/glorot10a/glorot10a.pdf

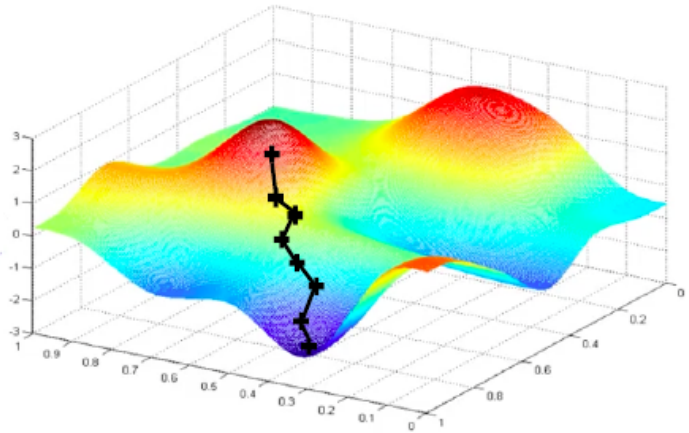
Recipe of Deep Learning



Stochastic Gradient Descent: Mini-batch

Algorithm

1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Pick batch of B data points
4. Compute gradient, $\frac{\partial J(\theta)}{\partial \theta} = \frac{1}{B} \sum_{k=1}^{B} \frac{\partial J_k(\theta)}{\partial \theta}$
5. Update weights, $\theta \leftarrow \theta - \eta \frac{\partial J(\theta)}{\partial \theta}$
6. Return weights

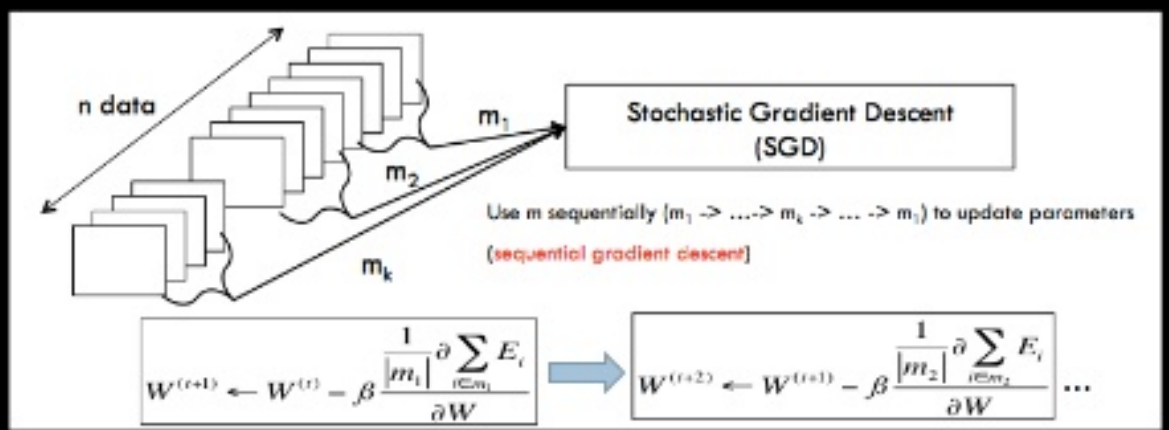


Fast to compute and a much better estimate of the true gradient!

Mini-batch

SGD - Mini batch

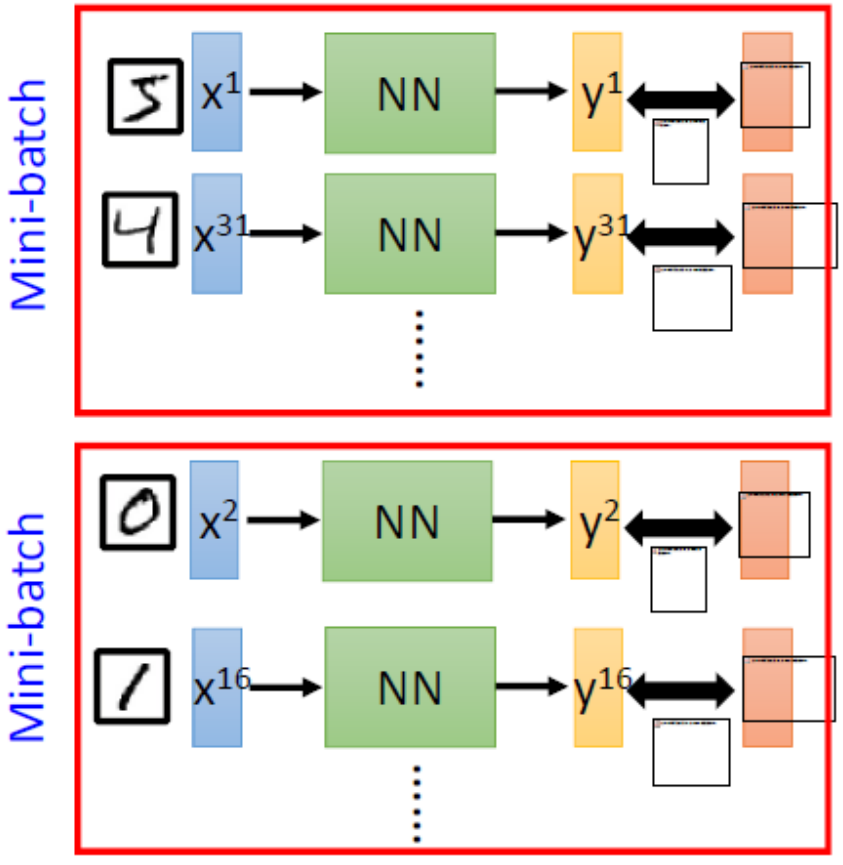
- Batch learning: update all samples in each update (over-fitting)
- Mini batch: update parameters with some samples



20

We do not really minimize total loss!

Mini-batch



- Randomly initialize network parameters
- Pick the 1st batch

  Update parameters once
- Pick the 2nd batch

  Update parameters once

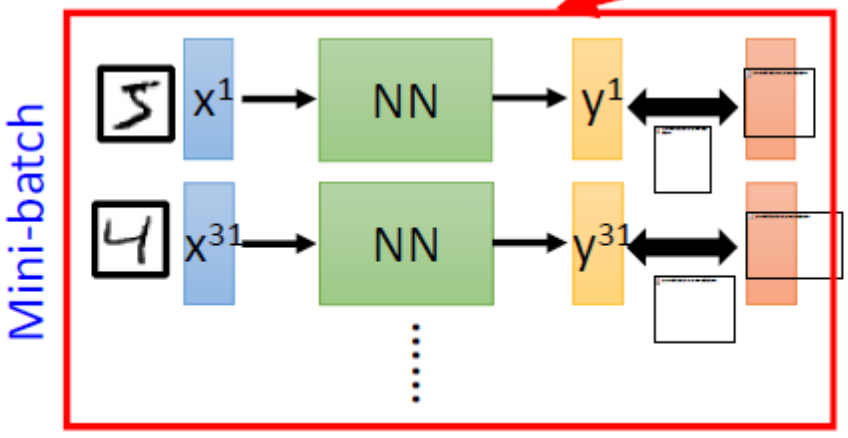
  ⋮
- Until all mini-batches have been picked

one epoch

Repeat the above process

Mini-batch

```
model.fit(x_train, y_train, batch_size=100, nb_epoch=20)
```



100 examples in a mini-batch

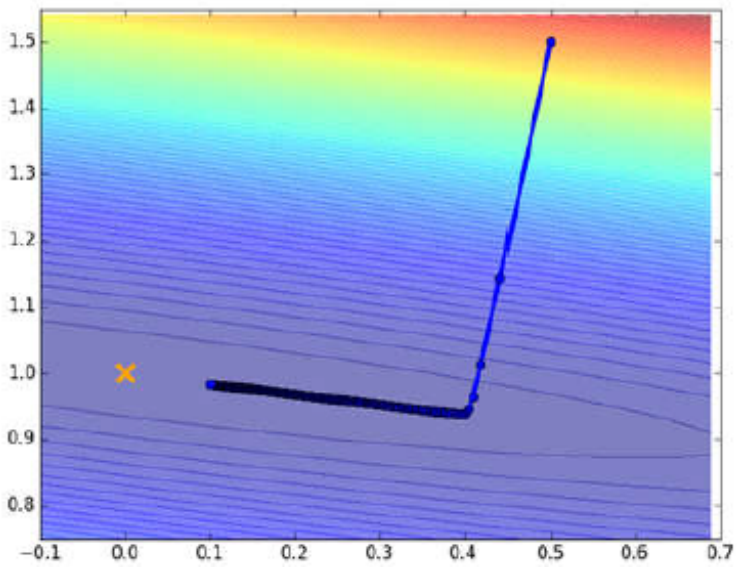
Repeat 20 times

- Pick the 1st batch
- Update parameters once
- Pick the 2nd batch
- Update parameters once
- ⋮
- Until all mini-batches have been picked

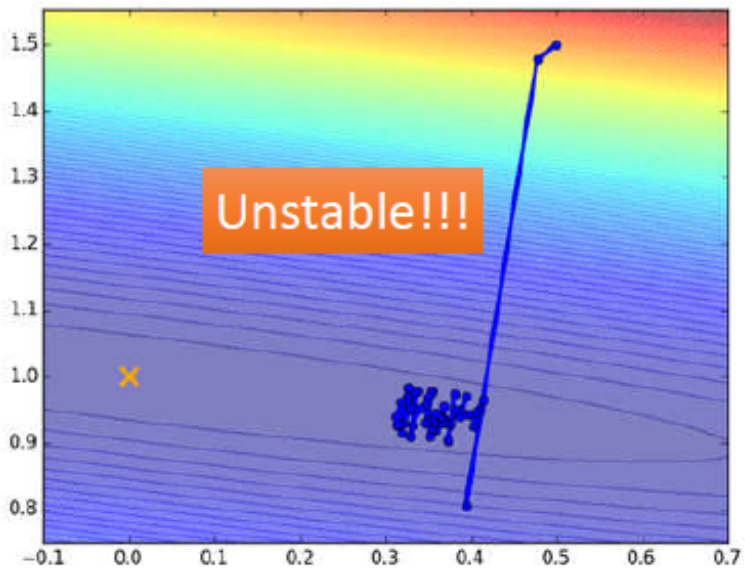
one epoch

Mini-batch

Original Gradient Descent



With Mini-batch



The colors represent the total loss.

Mini-batch is Faster

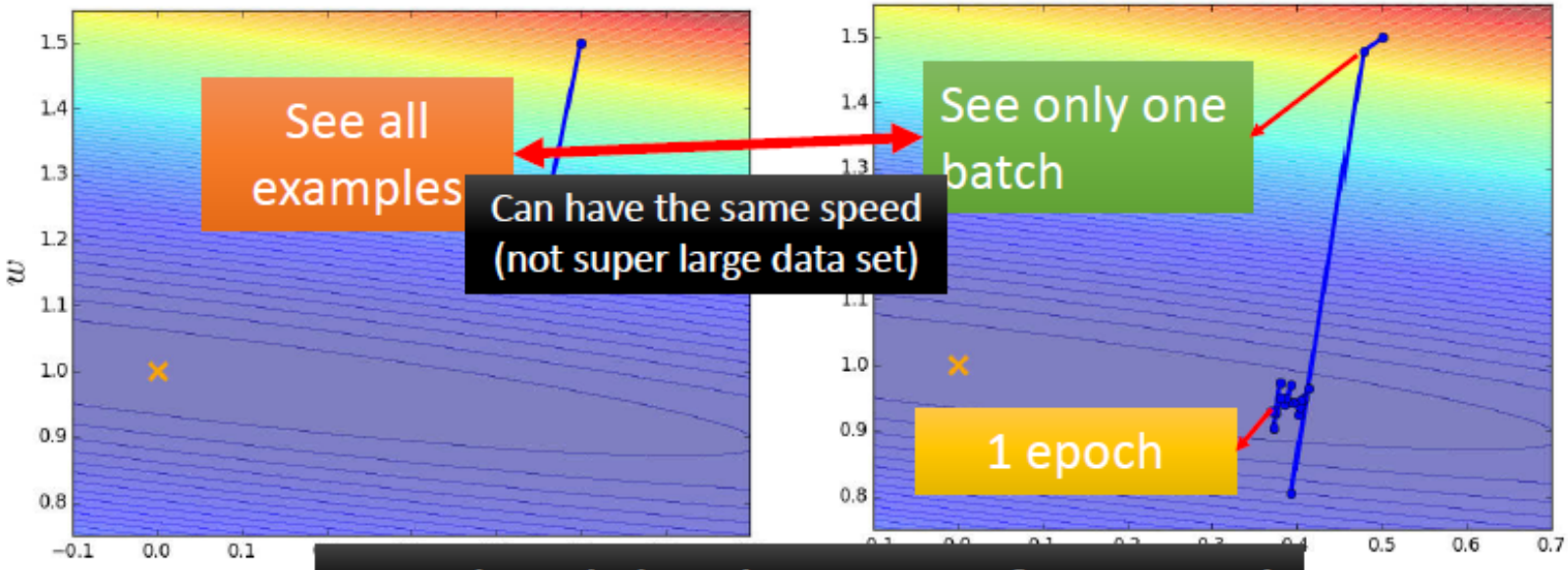
Not always true with parallel computing.

Original Gradient Descent

Update after seeing all examples

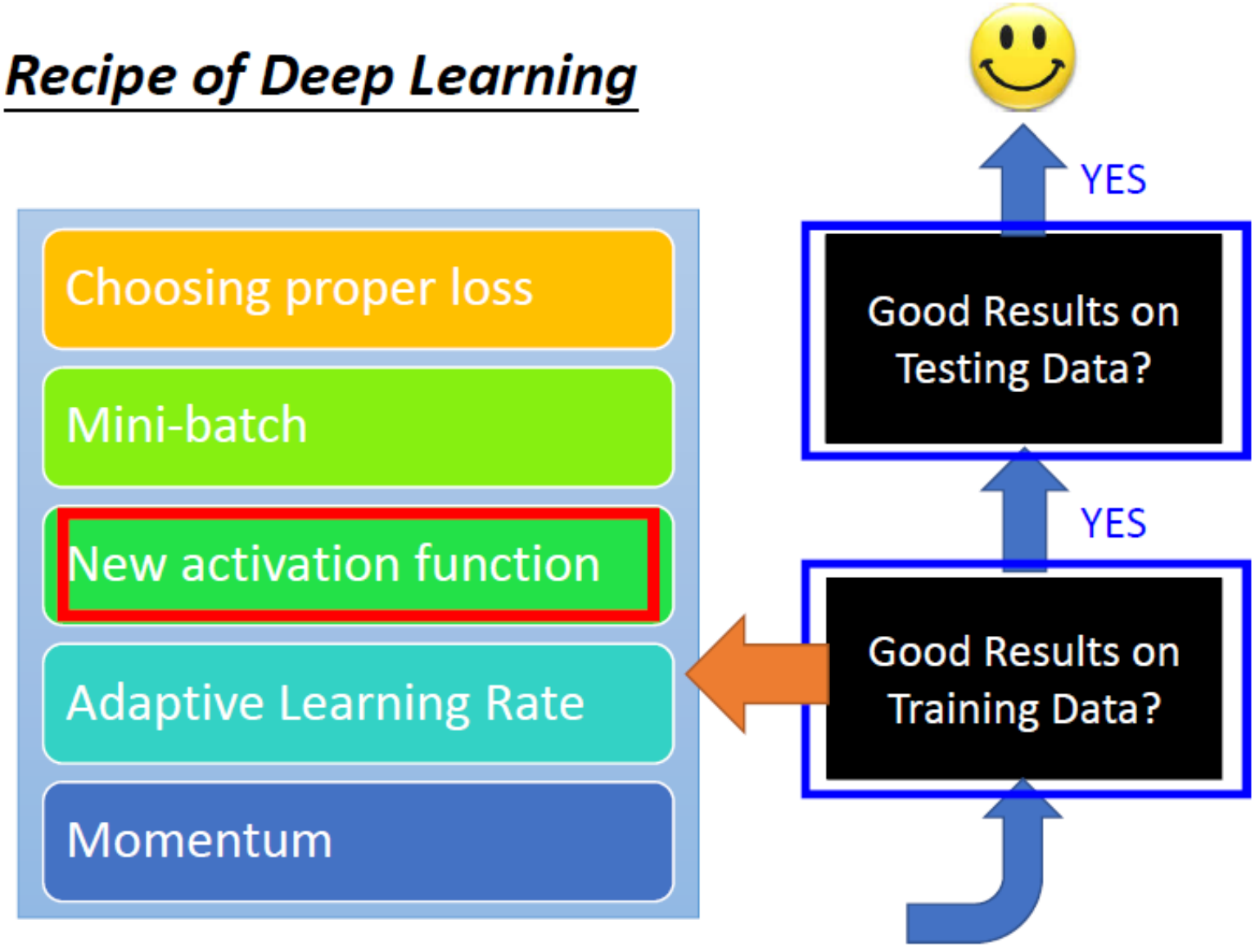
With Mini-batch

If there are 20 batches, update 20 times in one epoch.



Mini-batch has better performance!

Recipe of Deep Learning



梯度消失问题（Vanishing Gradient Problem）

在神经网络中误差反向传播的迭代公式为

$$\delta^{(l)} = f_l'(\mathbf{z}^{(l)}) \odot ((W^{(l+1)})^\top \delta^{(l+1)},$$

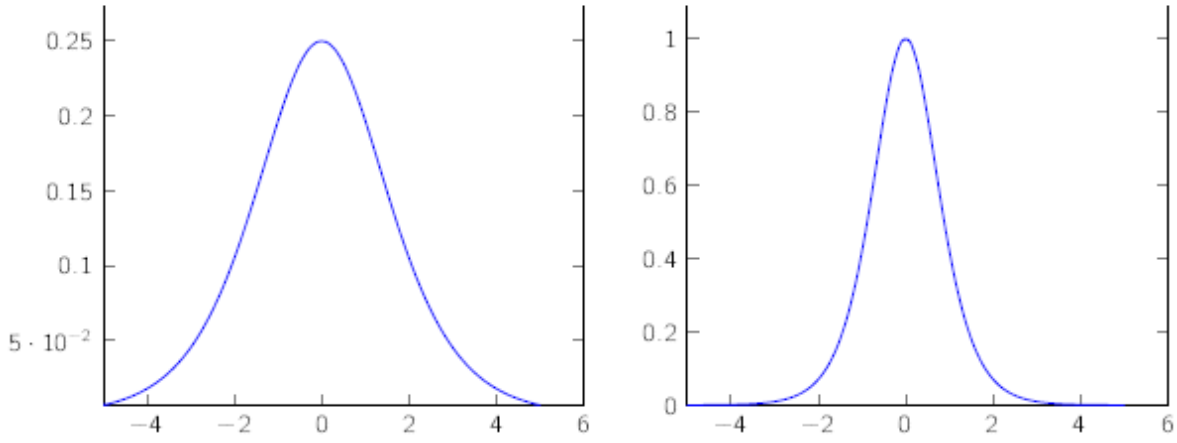
其中需要用到激活函数 f_l 的导数。误差从输出层反向传播时，在每一层都要乘以该层的激活函数的导数。

当我们使用 sigmoid 型函数：logistic 函数 $\sigma(x)$ 或 tanh 函数时，其导数为

$$\sigma'(x) = \sigma(x)(1 - \sigma(x)) \in [0, 0.25]$$
$$\tanh'(x) = 1 - (\tanh(x))^2 \in [0, 1].$$

软饱和：$\lim_{x \to \infty} f'(x) = 0$
硬饱和：$f'(x) = 0, \quad \text{if } |x| > c$

梯度消失问题

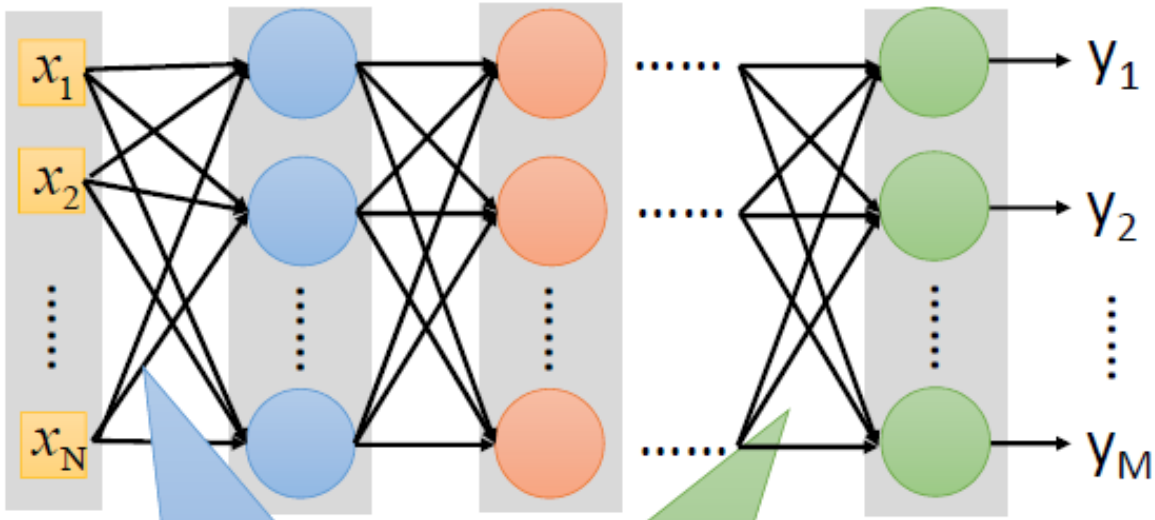


(a) logistic 函数的导数　　(b) tanh 函数的导数

图: 激活函数的导数

对于 Sigmoid 型函数一旦输入落入饱和区，$f'(x)$ 就会变得接近于 0，导致了向底层传递的梯度也变得非常小；当网络层数很深时，梯度就会不停的衰减，甚至消失，使得整个网络很难训练。

梯度消失问题



Smaller gradients

Learn very slow

Almost random

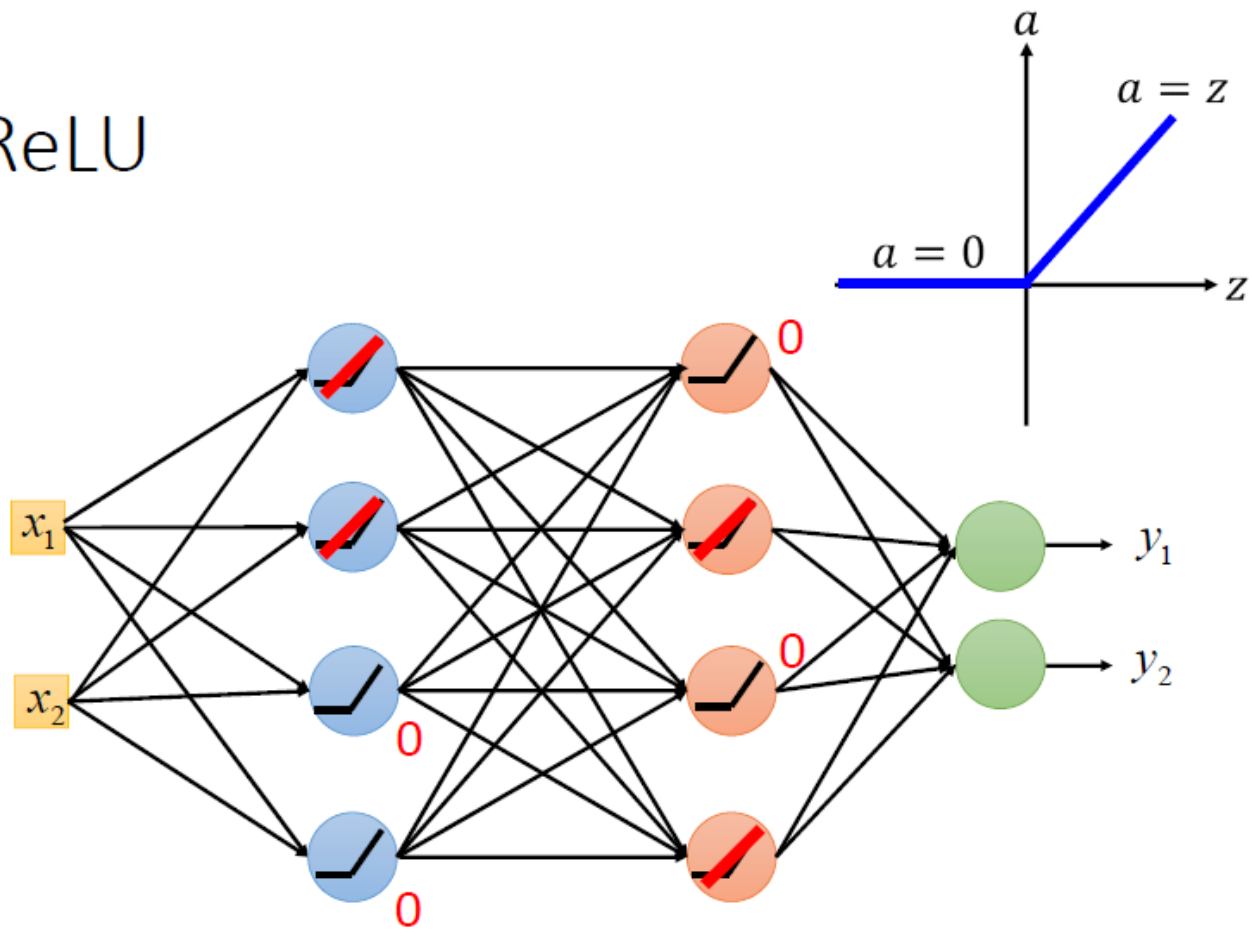
Larger gradients

Learn very fast

Already converge

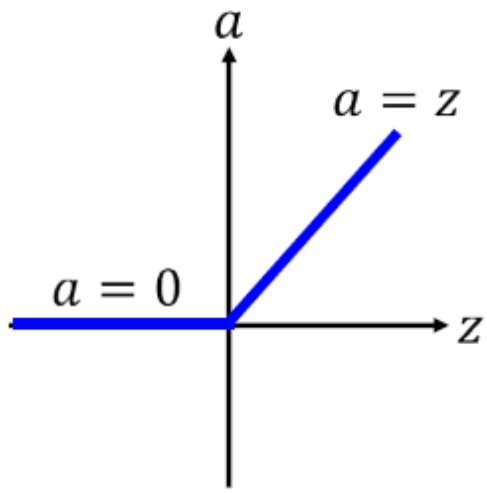
based on random!?

ReLU

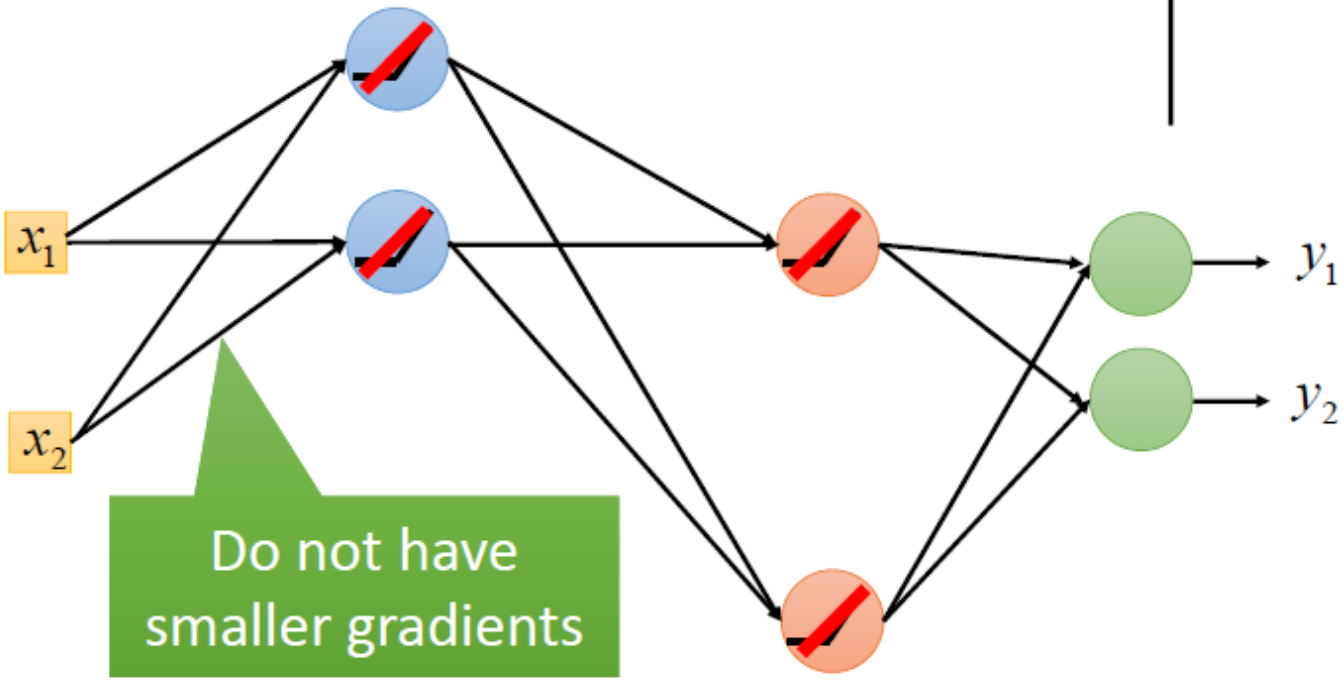


在深层神经网络中，减轻梯度消失问题的方法有很多种。一种简单有效的方式是使用导数比较大的激活函数，比如 ReLU

ReLU



A Thinner linear network

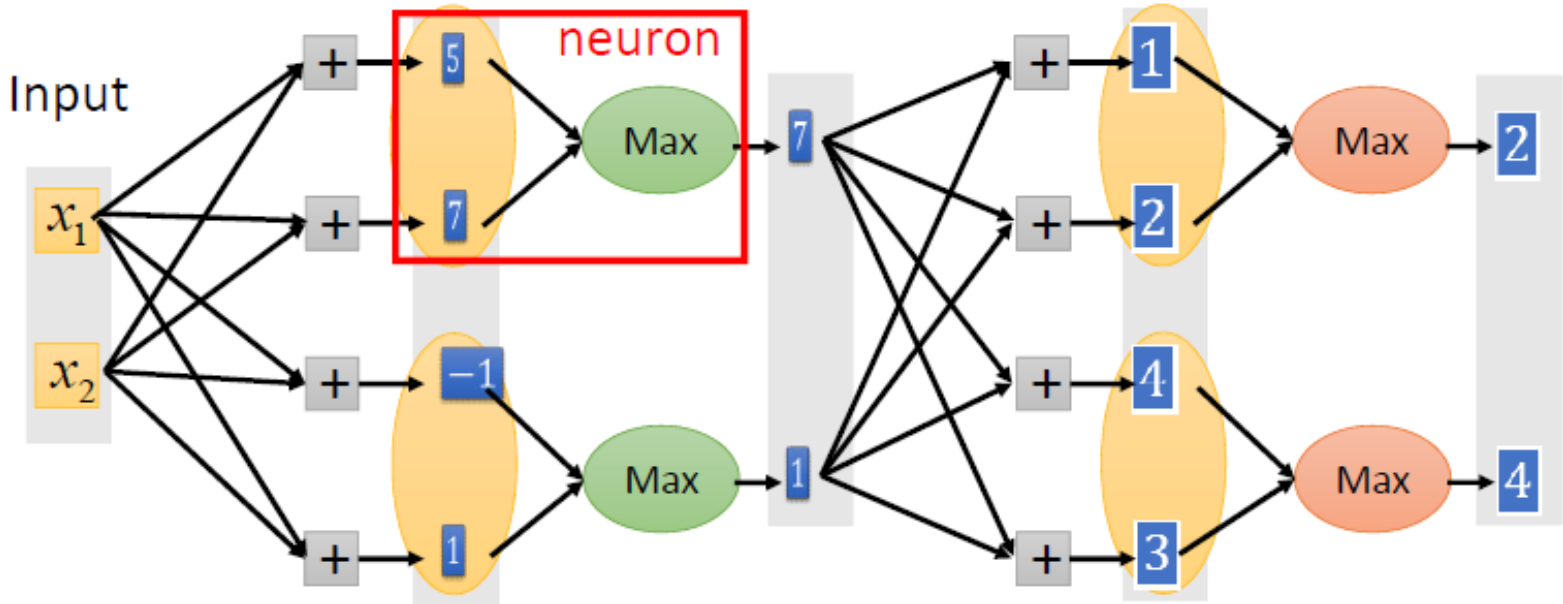


Do not have smaller gradients

Maxout

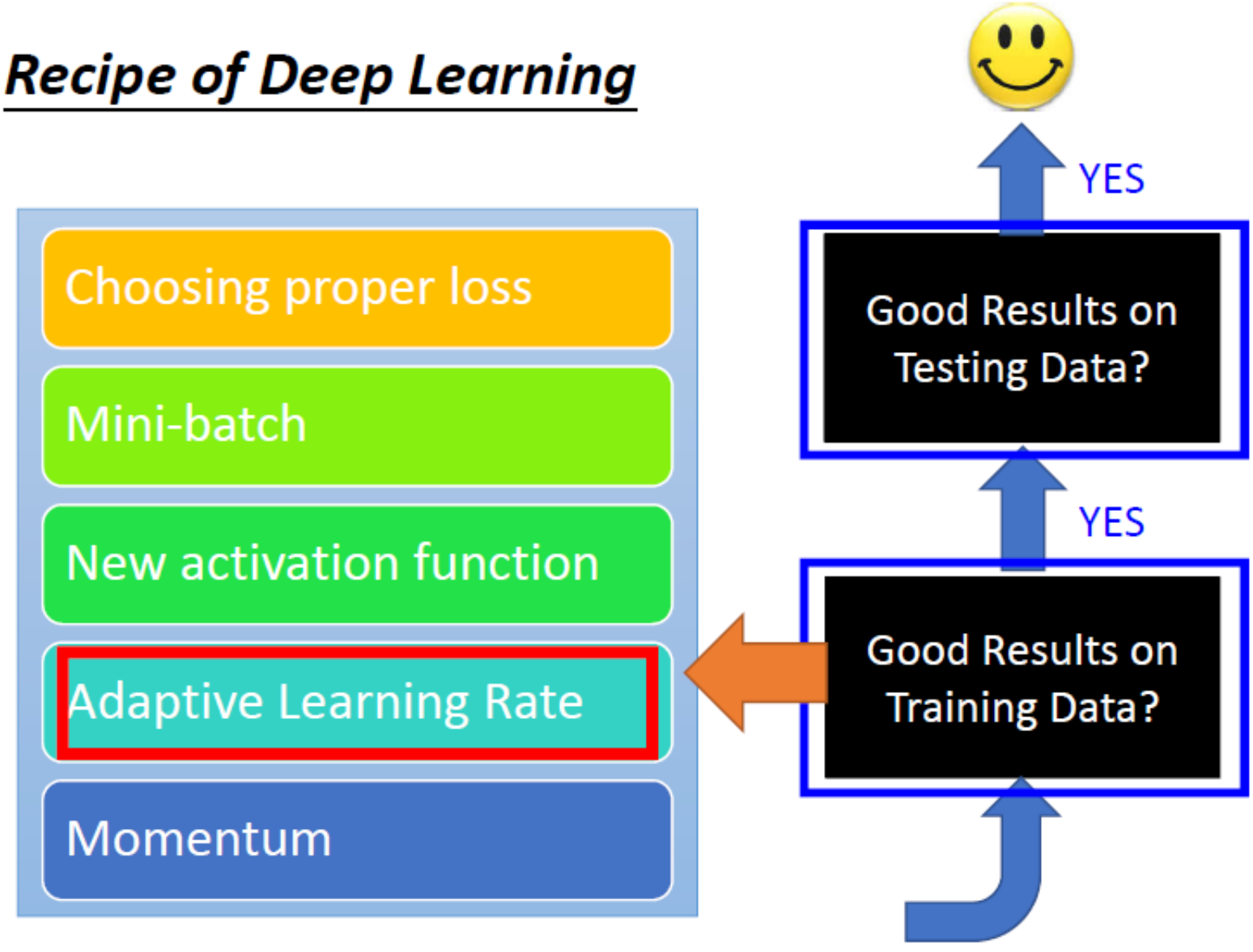
ReLU is a special cases of Maxout

- Learnable activation function [Ian J. Goodfellow, ICML'13]



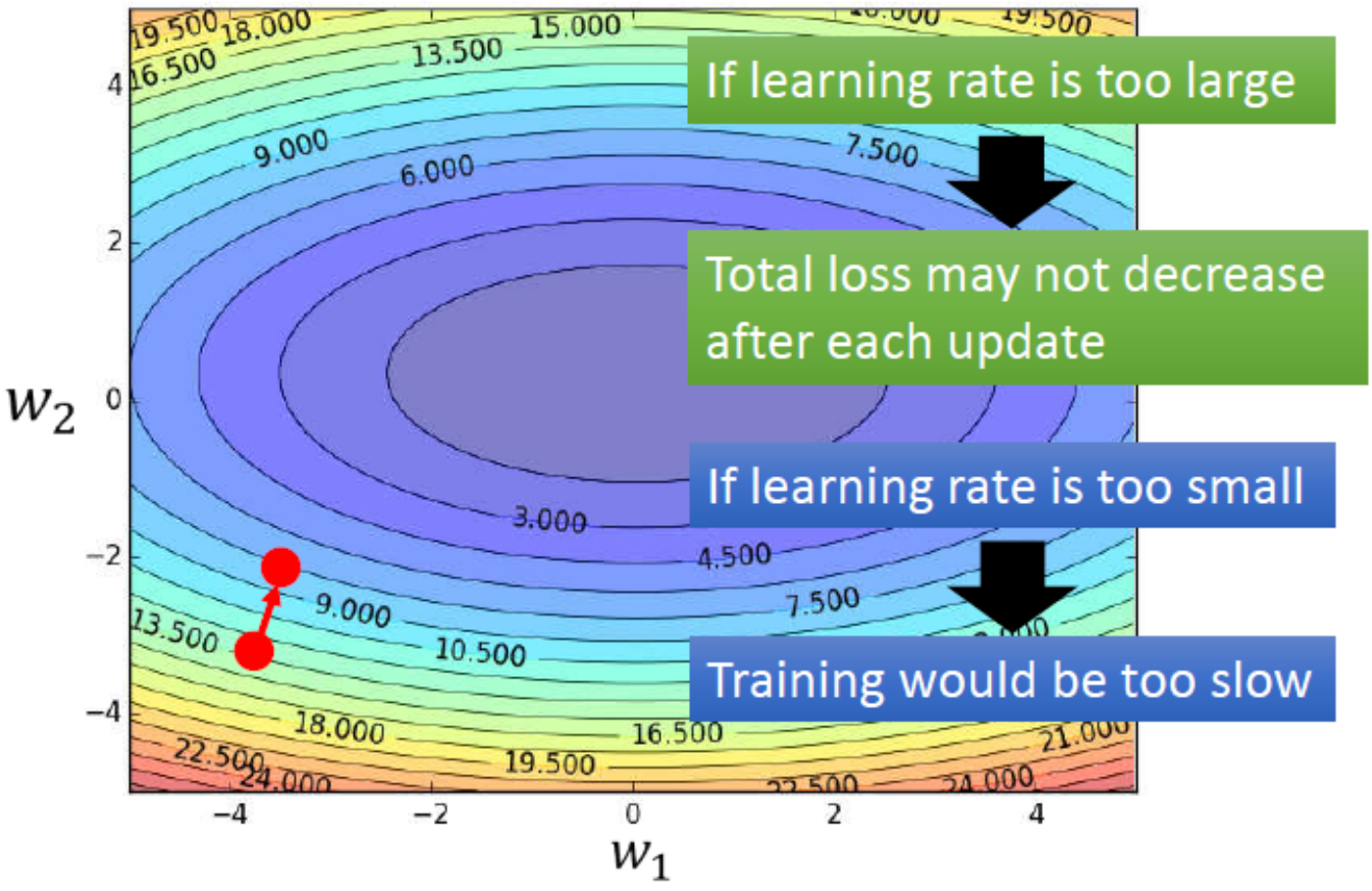
You can have more than 2 elements in a group.

Recipe of Deep Learning



Learning Rates

Set the learning rate η carefully



If learning rate is too large

Total loss may not decrease after each update

If learning rate is too small

Training would be too slow

Learning Rates

- Popular & Simple Idea: Reduce the learning rate by some factor every few epochs.
  - At the beginning, we are far from the destination, so we use larger learning rate
  - After several epochs, we are close to the destination, so we reduce the learning rate
  - E.g. 1/t decay: $\eta^t = \eta/\sqrt{t+1}$
- Learning rate cannot be one-size-fits-all
  - Giving different parameters different learning rates

Adagrad

Original: $w \leftarrow w - \eta \partial L / \partial w$

Adagrad: $w \leftarrow w - \eta_w \partial L / \partial w$

Parameter dependent learning rate

$$\eta_w = \frac{\eta}{\sqrt{\sum_{i=0}^{t} (g^i)^2}}$$

η: constant

g^i is $\partial L / \partial w$ obtained at the i-th update

Summation of the square of the previous derivatives

Adagrad

$$\eta_w = \frac{\eta}{\sqrt{\sum_{i=0}^{t}(g^i)^2}}$$

w_1	g^0
	0.1

w_2	g^0
	20.0

Learning rate:

$$\frac{\eta}{\sqrt{0.1^2}} = \frac{\eta}{0.1}$$

$$\frac{\eta}{\sqrt{0.1^2 + 0.2^2}} = \frac{\eta}{0.22}$$

Learning rate:

$$\frac{\eta}{\sqrt{20^2}} = \frac{\eta}{20}$$

$$\frac{\eta}{\sqrt{20^2 + 10^2}} = \frac{\eta}{22}$$

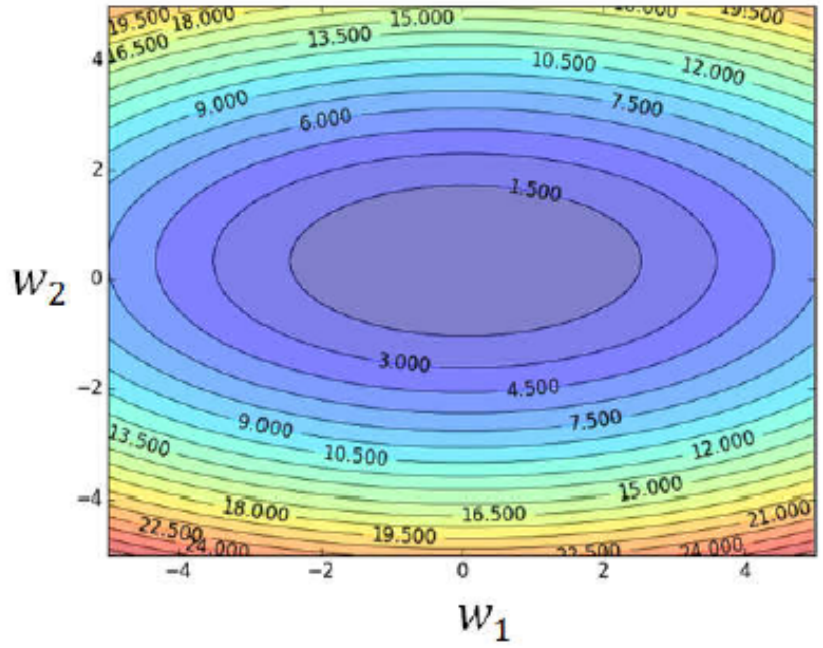
Observation: 1. Learning rate is smaller and smaller for all parameters

2. Smaller derivatives, larger learning rate, and vice versa

Why?

Larger derivatives

Smaller Learning Rate



Smaller Derivatives

Larger Learning Rate

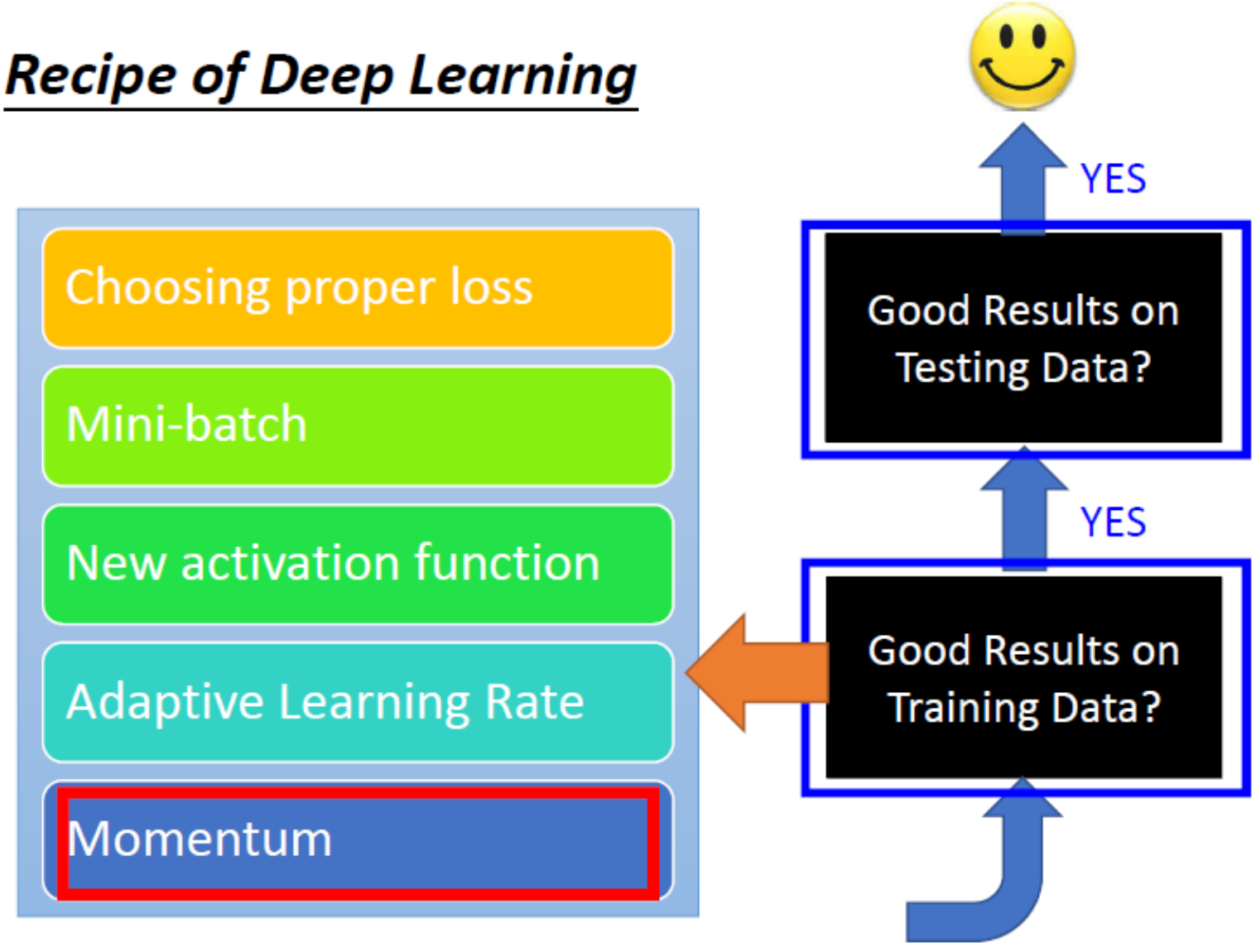
2. Smaller derivatives, larger learning rate, and vice versa

Why?

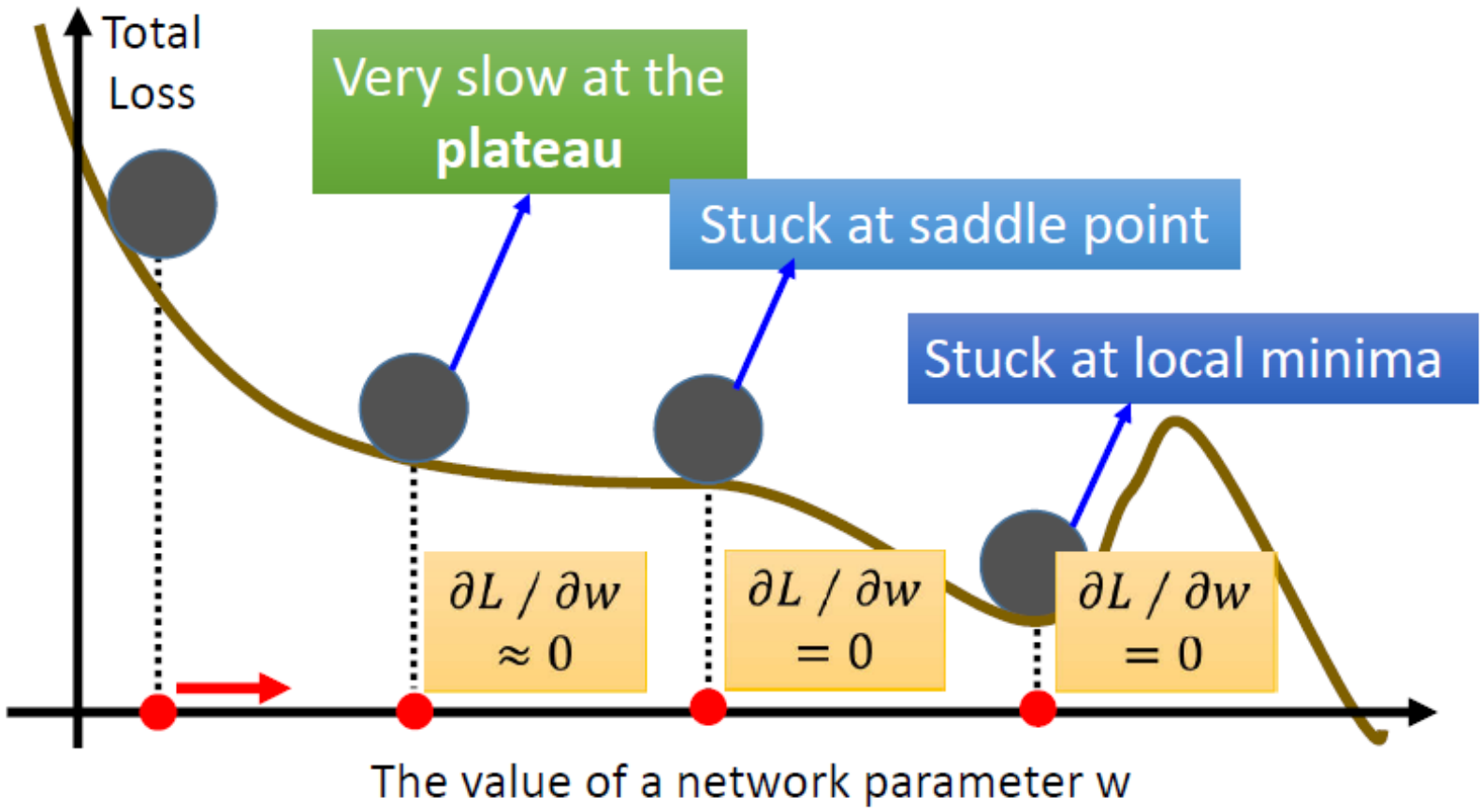
其他方法

- Adagrad [John Duchi, JMLR'11]
- RMSprop
  - https://www.youtube.com/watch?v=O3sxAc4hxZU
- Adadelta [Matthew D. Zeiler, arXiv'12]
- "No more pesky learning rates" [Tom Schaul, arXiv'12]
- AdaSecant [Caglar Gulcehre, arXiv'14]
- Adam [Diederik P. Kingma, ICLR'15]
- Nadam
  - http://cs229.stanford.edu/proj2015/054_report.pdf

Recipe of Deep Learning

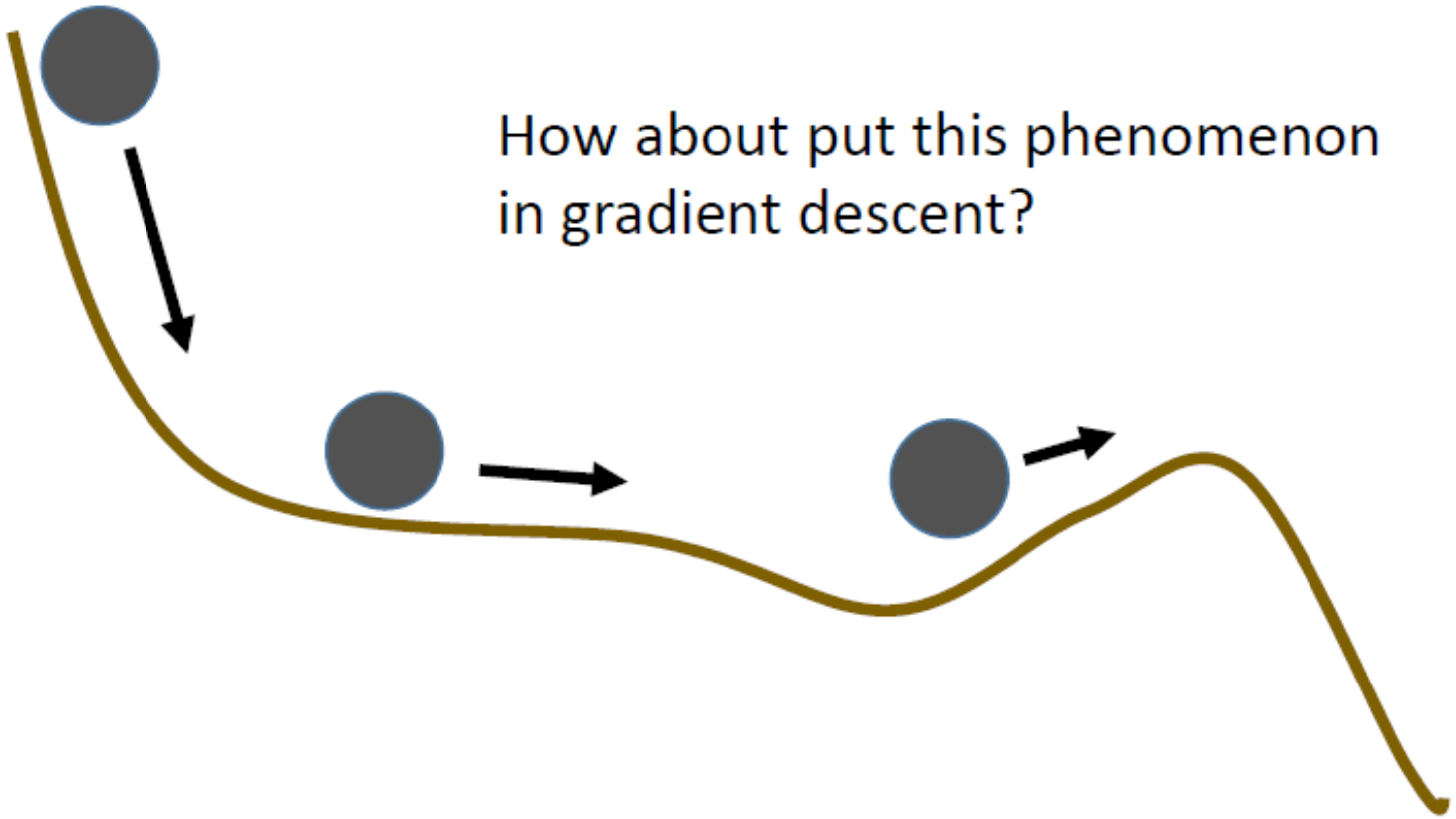


Hard to find optimal network parameters



In physical world ……

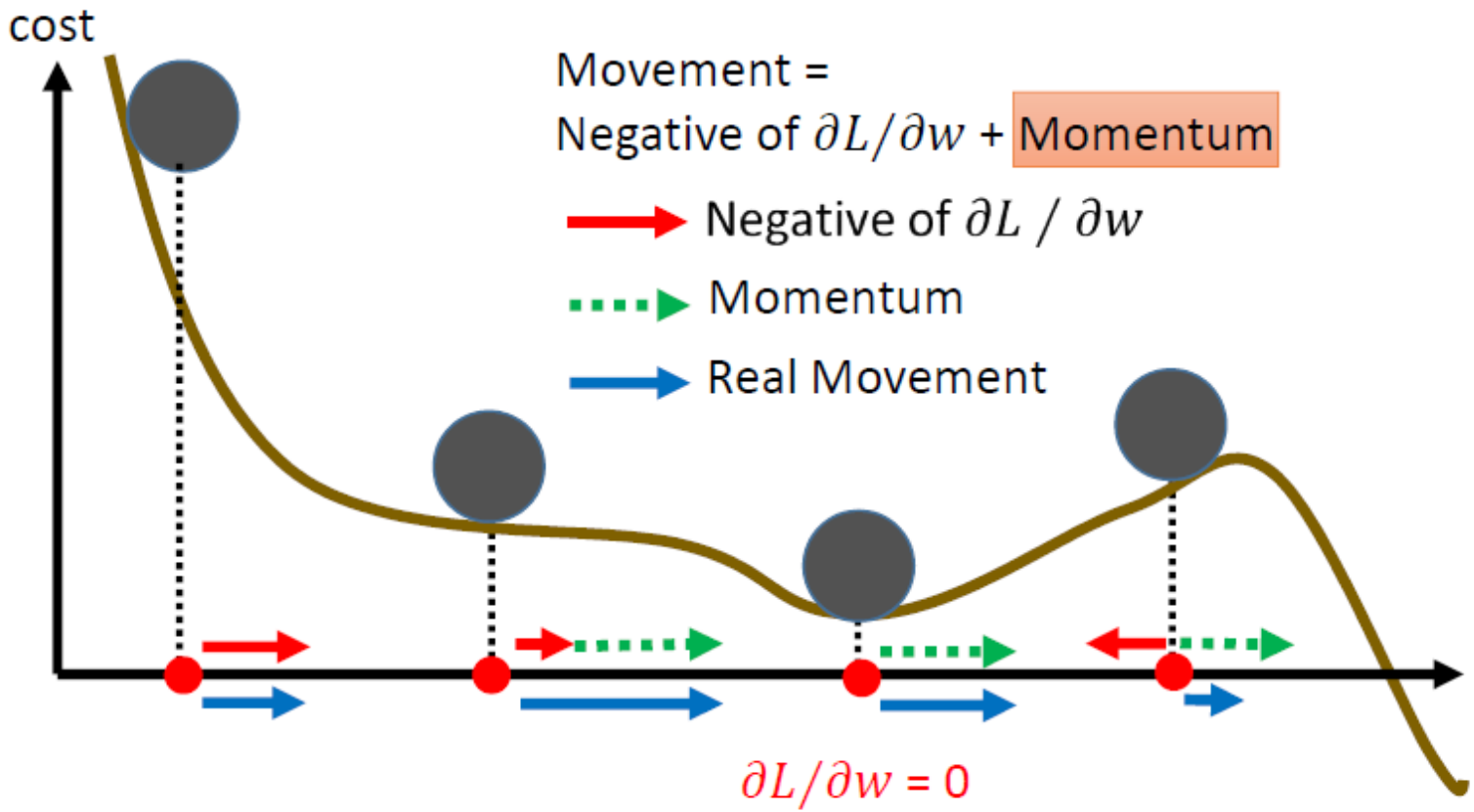
- Momentum



How about put this phenomenon in gradient descent?

Momentum

Still not guarantee reaching global minima, but give some hope ……



Momentum

$$v_t = \mu_{t-1} v_{t-1} - \epsilon_{t-1} \nabla f(\theta_{t-1})$$
$$\theta_t = \theta_{t-1} + v_t$$
$$= \theta_{t-1} + \mu_{t-1} v_{t-1} - \epsilon_{t-1} \nabla f(\theta_{t-1})$$

Standard momentum

$$v_t = \mu_{t-1} v_{t-1} - \epsilon_{t-1} \nabla f(\theta_{t-1} + \mu_{t-1} v_{t-1})$$
$$\theta_t = \theta_{t-1} + v_t$$

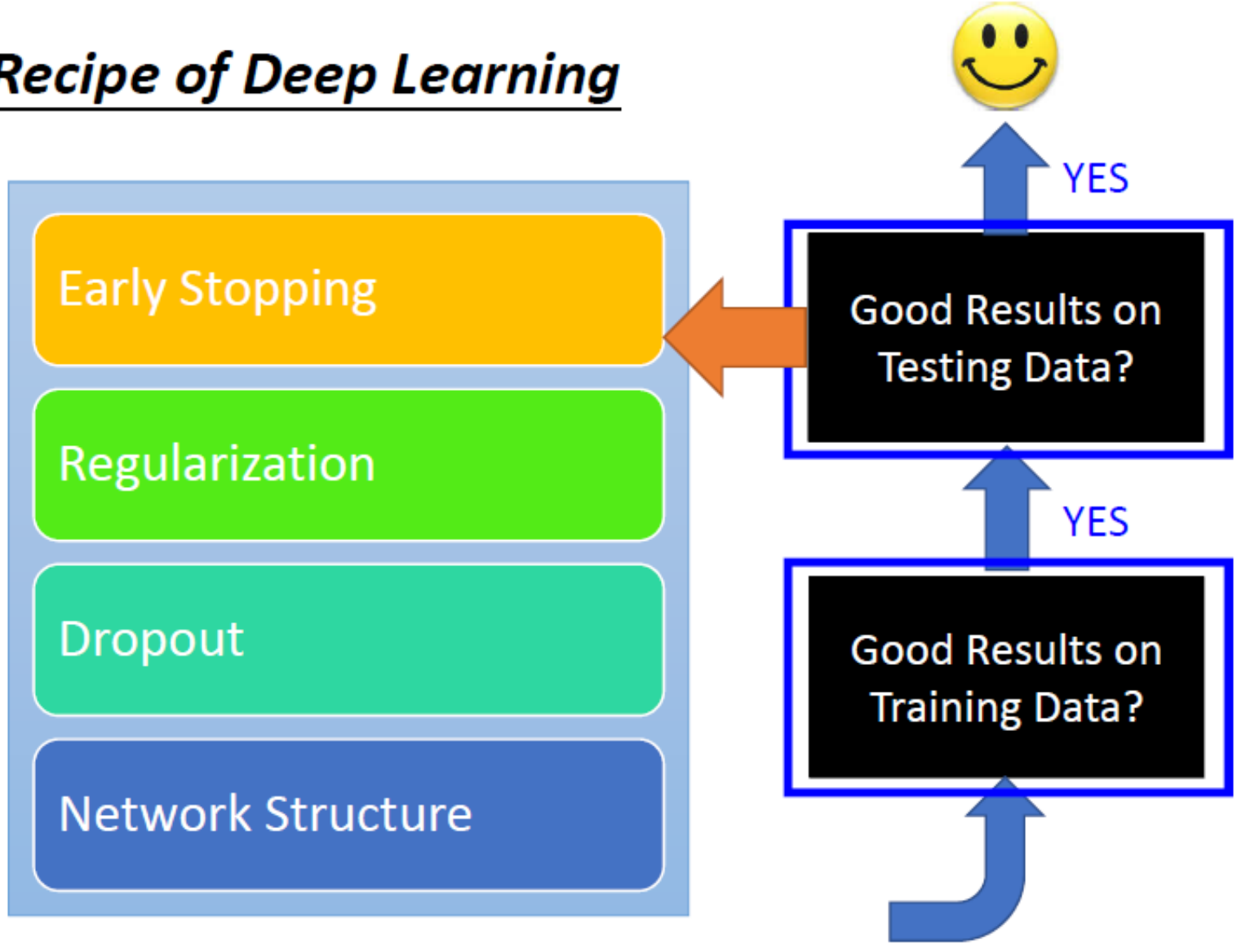
Nesterov momentum (definition)

$$v_t = \mu_{t-1} v_{t-1} - \epsilon_{t-1} \nabla f(\Theta_{t-1})$$
$$\Theta_t = \Theta_{t-1} - \mu_{t-1} v_{t-1} + \mu_t v_t + v_t$$
$$= \Theta_{t-1} + \mu_t \mu_{t-1} v_{t-1} - (1 + \mu_t) \epsilon_{t-1} \nabla f(\Theta_{t-1})$$

Nesterov momentum (simplified computation)
Note: weights are slightly offset from original

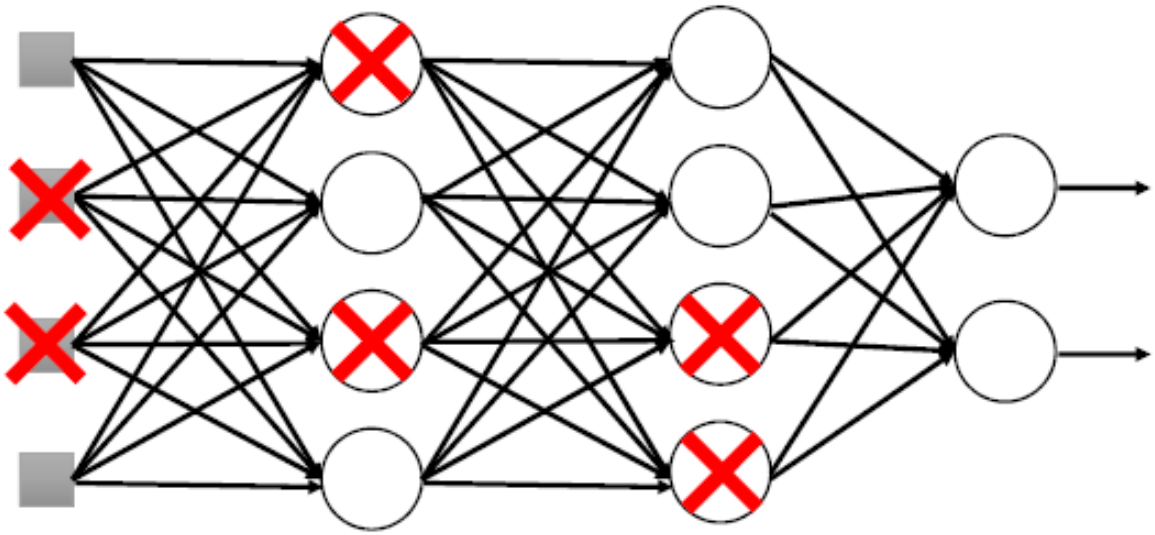
"Advances in Optimizing Recurrent Neural Networks", December 14, 2012.
Online: https://arxiv.org/pdf/1212.0901v2.pdf

Recipe of Deep Learning



Dropout

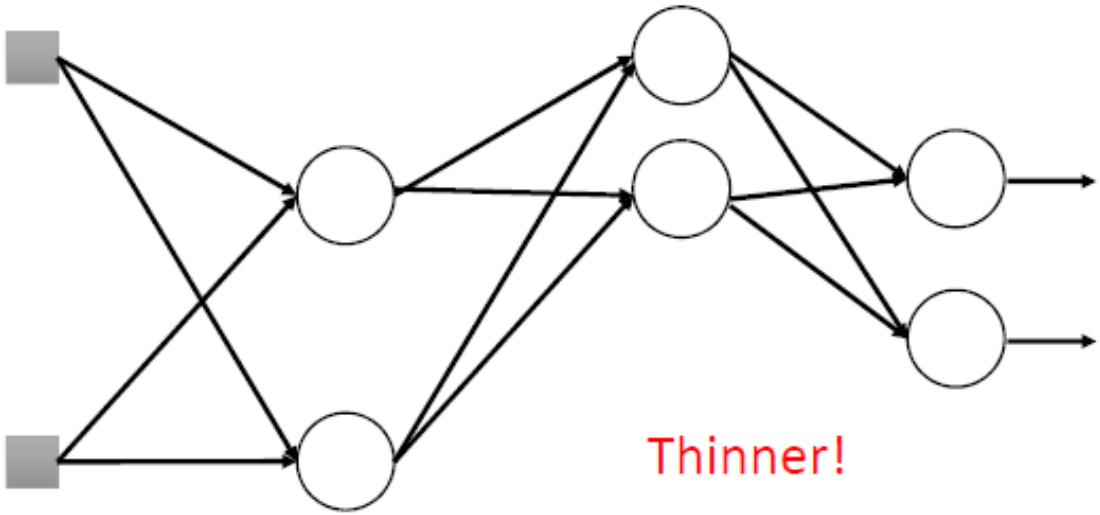
Training:



- **Each time before updating the parameters**
  - Each neuron has p% to dropout

Dropout

Training:



- **Each time before updating the parameters**
  - Each neuron has p% to dropout
    - **The structure of the network is changed.**
  - Using the new network for training

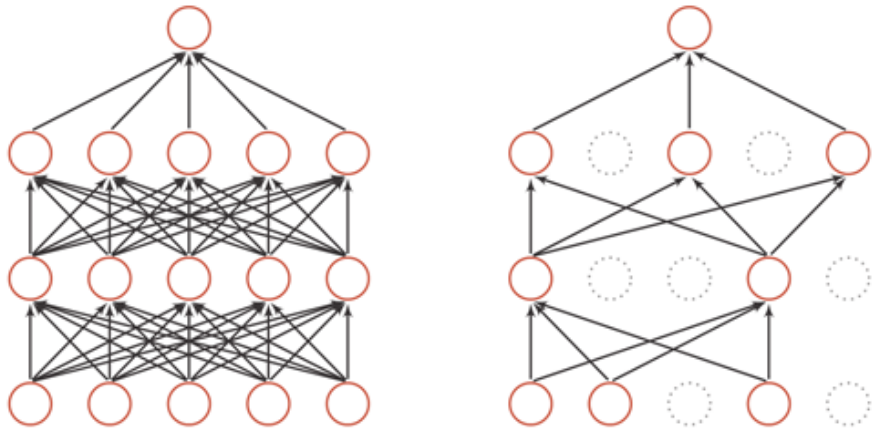
For each mini-batch, we resample the dropout neurons

Dropout

- 对于一个神经层$y = f(Wx + b)$，引入一个丢弃函数$d(\cdot)$使得$y = f(Wd(x) + b)$。

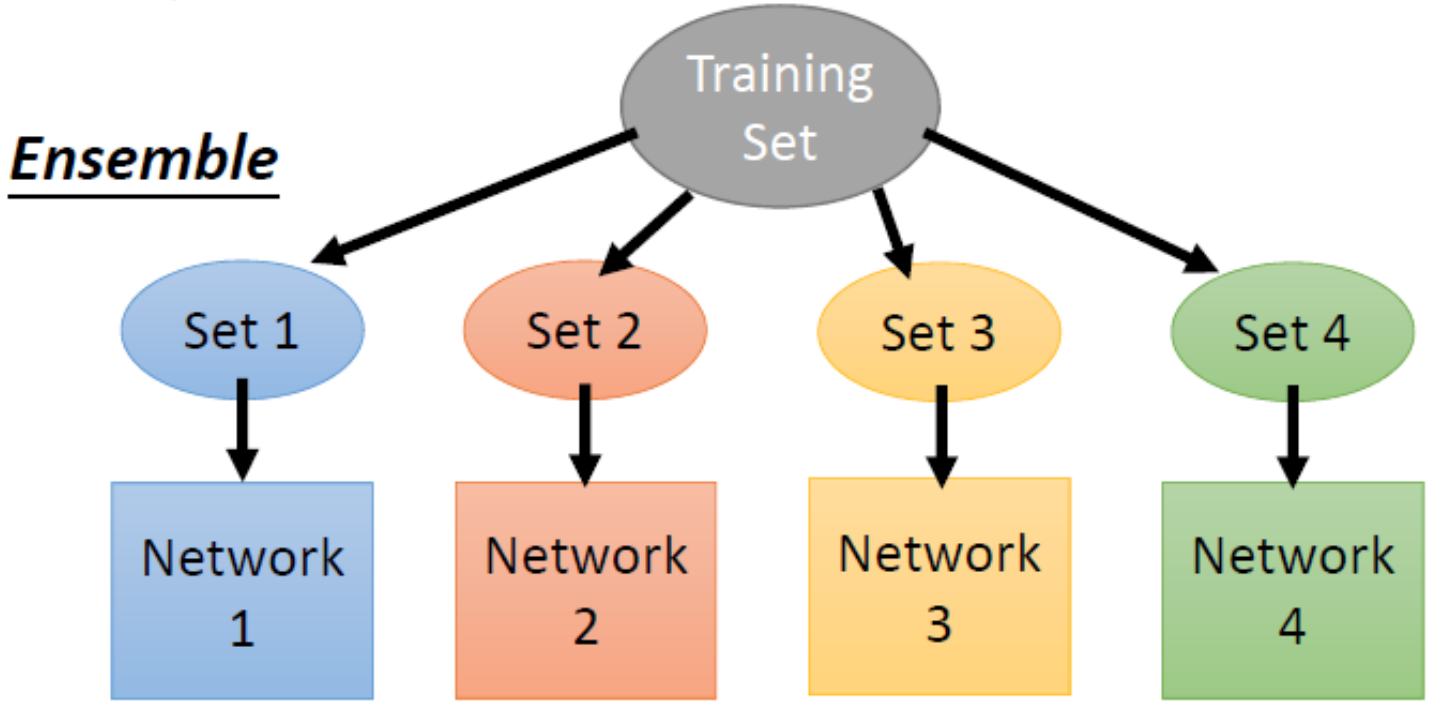
$$d(\mathbf{x}) = \begin{cases} \mathbf{m} \odot \mathbf{x} & \text{当训练阶段时} \\ p\mathbf{x} & \text{当测试阶段时} \end{cases}$$

  - 其中$m \in \{0,1\}^d$是丢弃掩码（dropout mask），通过以概率为p的贝努力分布随机生成。



(a) 标准网络　　(b) Dropout 后的网络

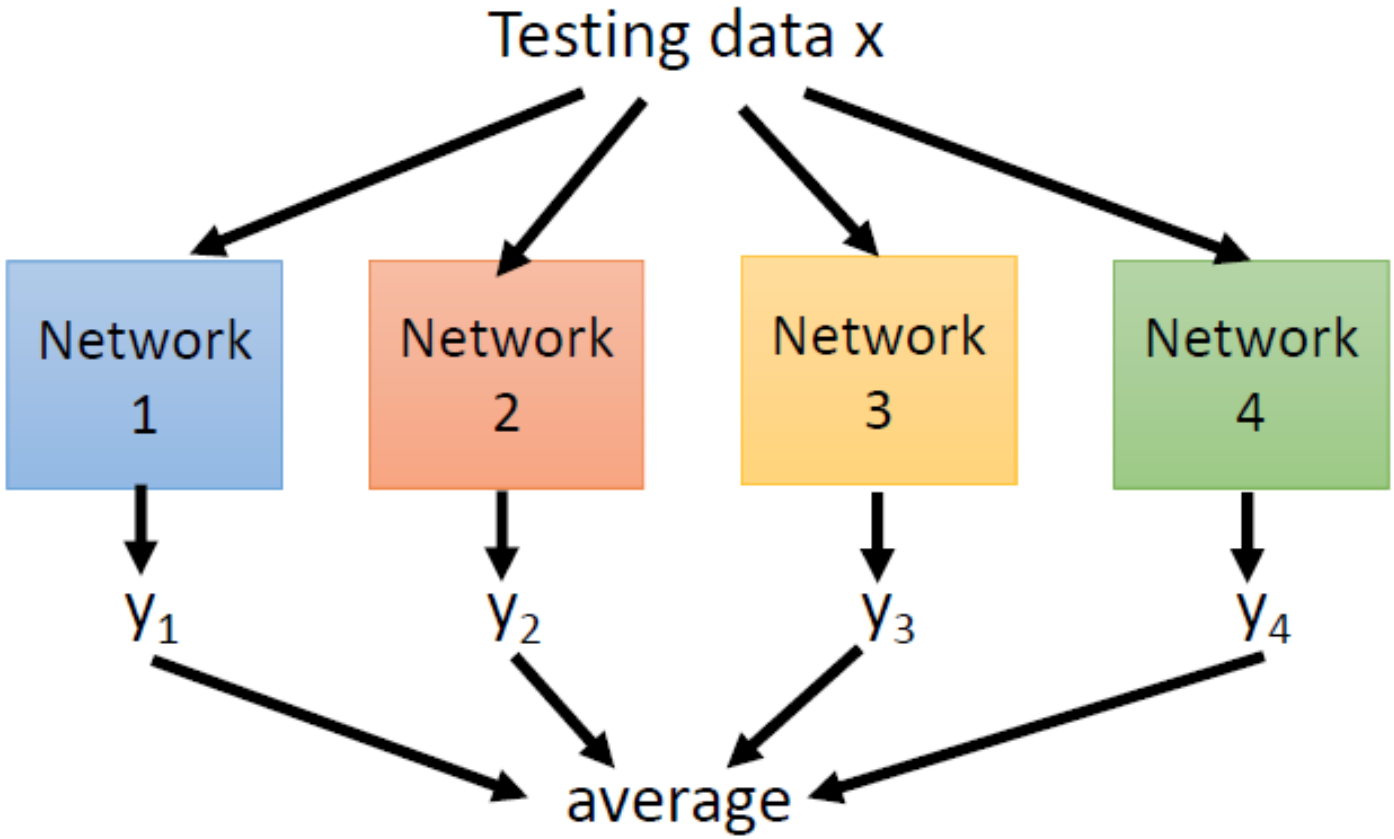
Dropout is a kind of ensemble.



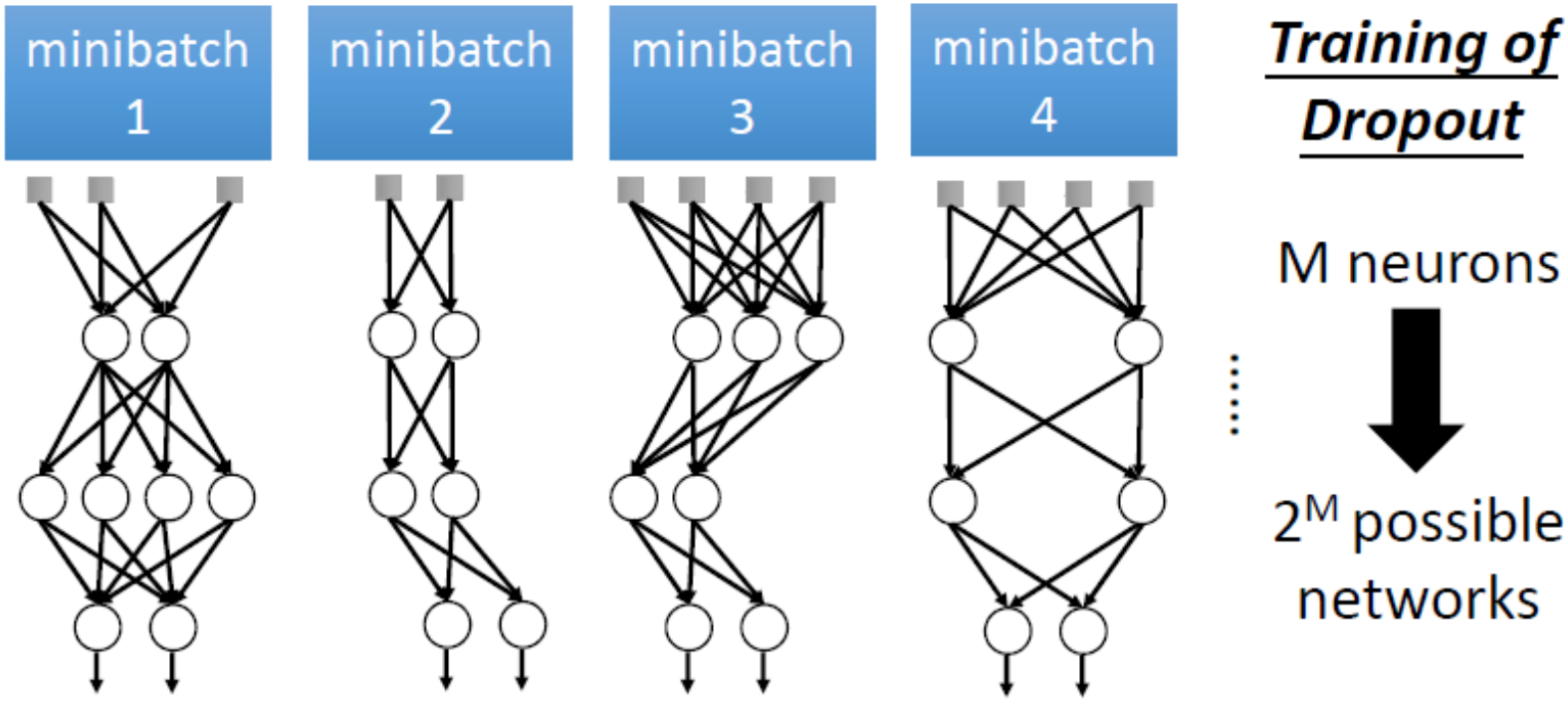
Train a bunch of networks with different structures

Dropout is a kind of ensemble.

Ensemble



Dropout is a kind of ensemble.



- Using one mini-batch to train one network
- Some parameters in the network are shared

Dropout is a kind of ensemble.

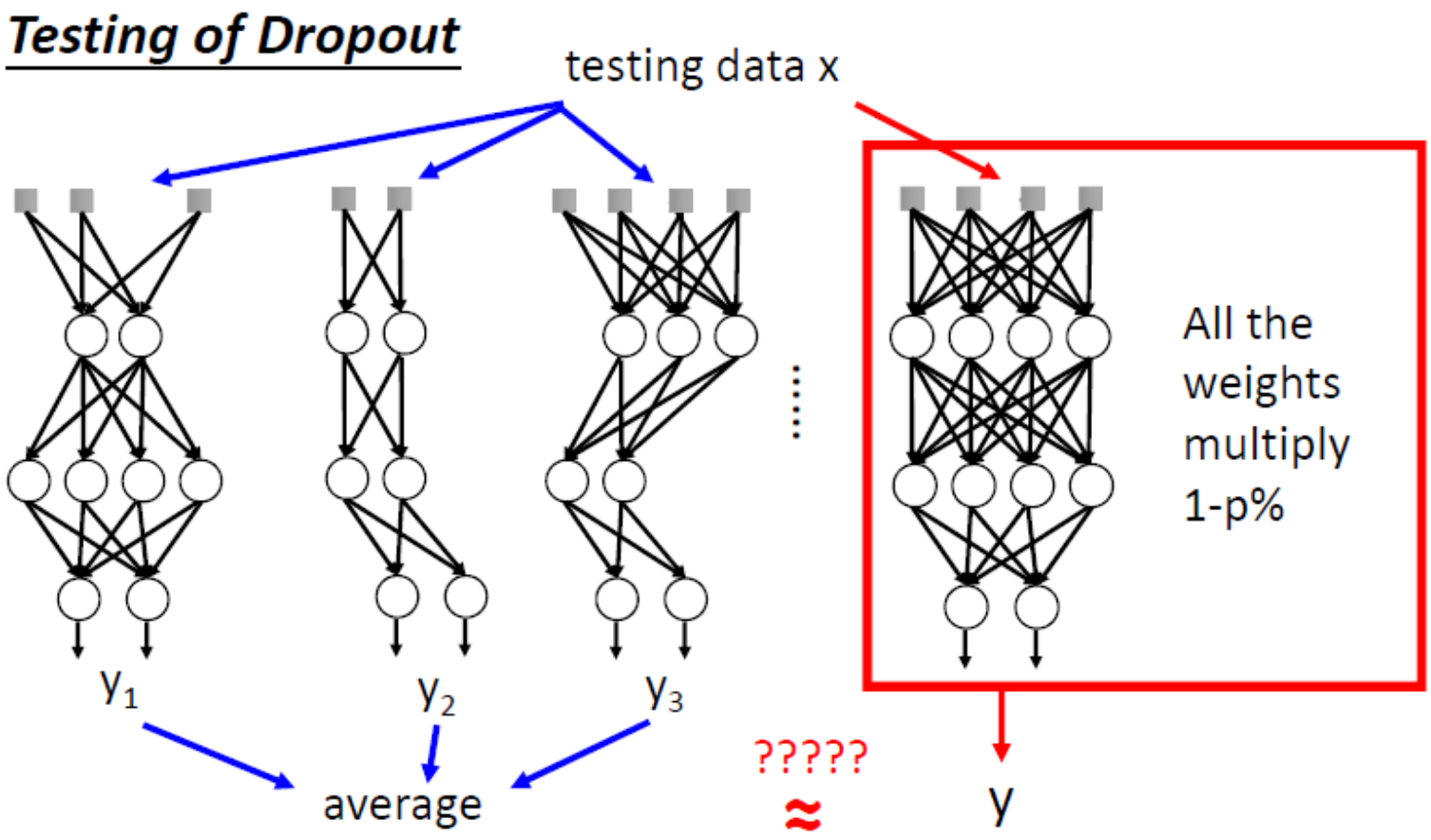


Table of Contents

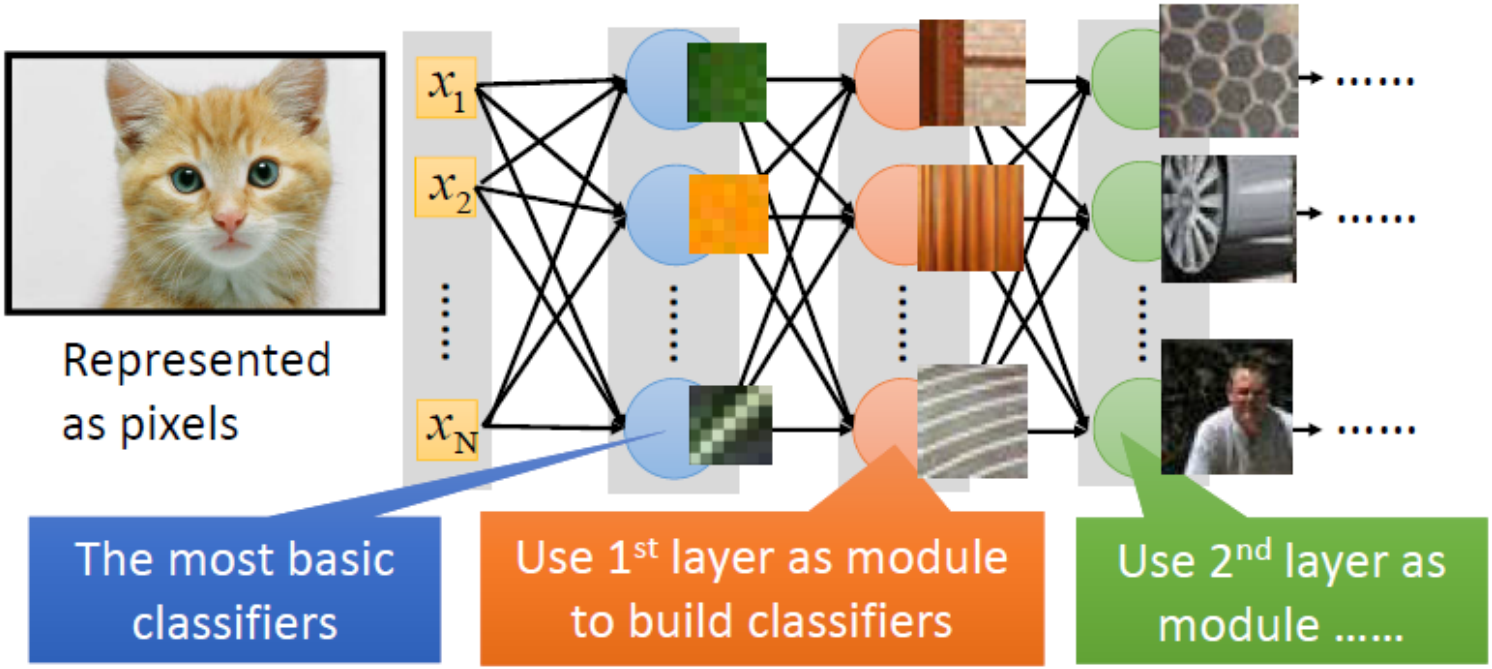
全连接前馈网络处理图像

如果用全连接前馈网络来处理图像时，会存在以下两个问题:

- 参数太多：如果输入图像大小为 $100 \times 100 \times 3$(即图像高度为 100，宽度为 100，3 个颜色通道:RGB)。在全连接前馈网络中，第一个隐藏层的每个神经元到输入层都有 $100 \times 100 \times 3 = 30,000$ 个相互独立的连接，每个连接都对应一个权重参数。随着隐藏层神经元数量的增多，参数的规模也会急剧增加。这会导致整个神经网络的训练效率会非常低，也很容易出现过拟合
- 局部不变性特征：自然图像中的物体都具有局部不变性特征，比如在尺度缩放、平移、旋转等操作不影响其语义信息。而全连接前馈网络很难提取这些局部不变特征，一般需要进行数据增强来提高性能。

Why CNN for Image?

[Zeiler, M. D., *ECCV 2014*]



Can the network be simplified by considering the properties of images?

Why CNN for Image

- Some patterns are much smaller than the whole image

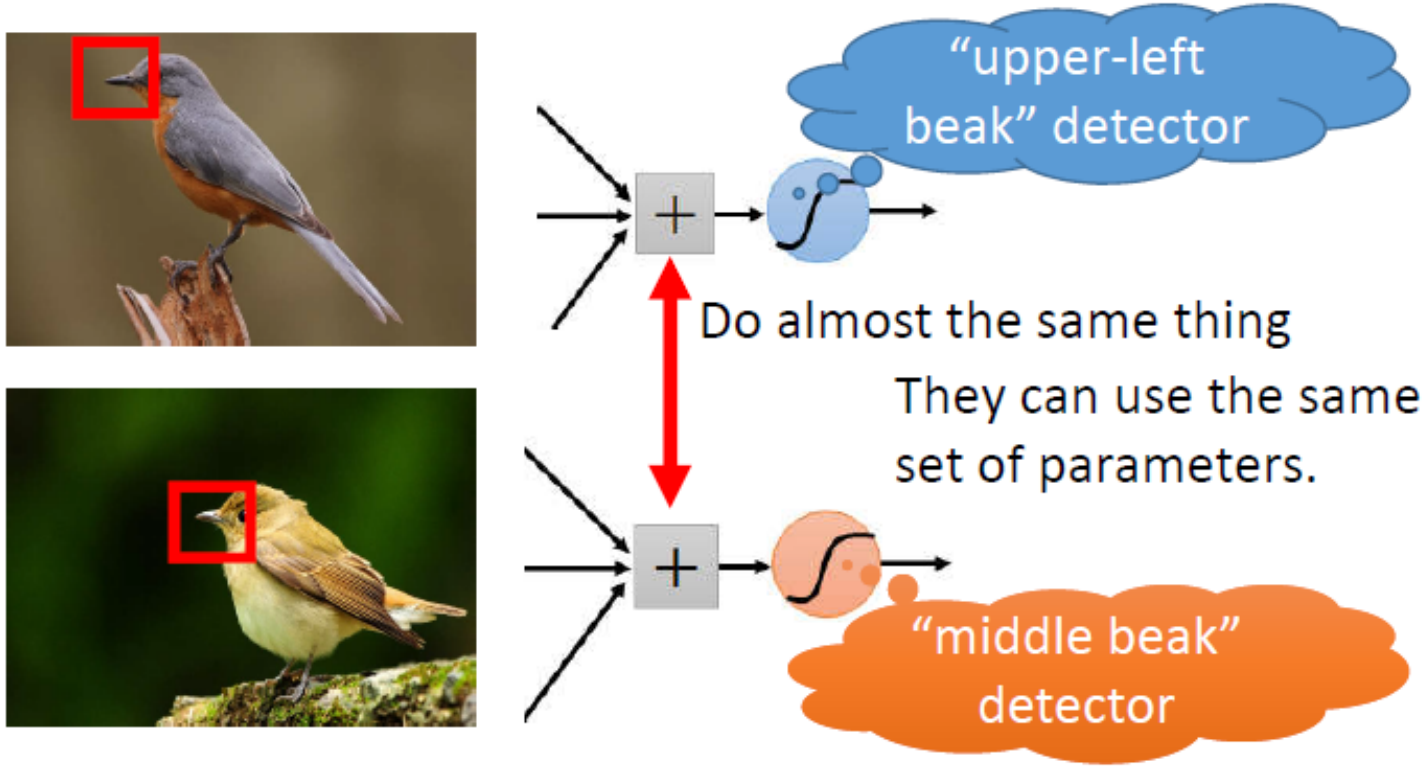
A neuron does not have to see the whole image to discover the pattern.

Connecting to small region with less parameters

局部连接

Why CNN for Image

- The same patterns appear in different regions.



权重共享

Why CNN for Image

- Subsampling the pixels will not change the object

bird

subsampling

bird

We can subsample the pixels to make image smaller

Less parameters for the network to process the image

次采样

The whole CNN

Convolution

Max Pooling

Convolution

Max Pooling

Can repeat many times

Flatten

Fully Connected Feedforward network

cat dog .....

The whole CNN

Property 1

- Some patterns are much smaller than the whole image

Property 2

- The same patterns appear in different regions.

Property 3

- Subsampling the pixels will not change the object

Convolution

Max Pooling

Convolution

Max Pooling

Can repeat many times

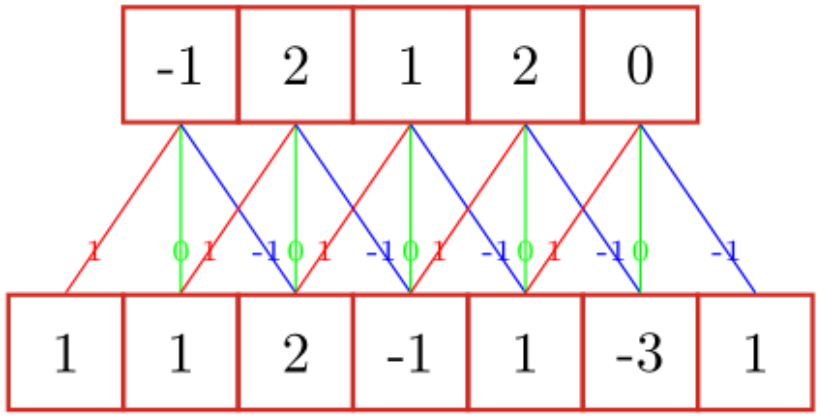
Flatten

卷积

- 给定一个输入信号序列x和滤波器f,卷积的输出为:

$$y_t = \sum_{k=1}^{m} w_k x_{t-k+1}$$

Filter: [-1,0,1]

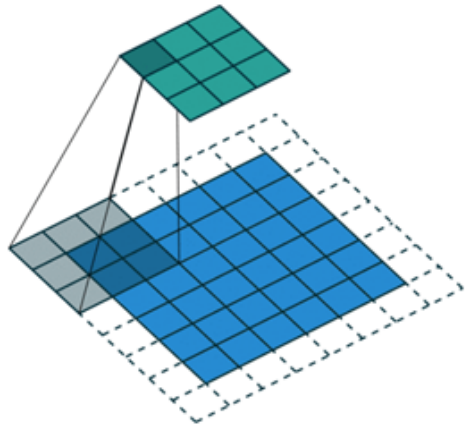


二维卷积

- 在图像处理中，图像是以二维矩阵的形式输入到神经网络中，因此我们需要二维卷积。

$$\mathbf{y} = \mathbf{w} \otimes \mathbf{x},$$

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



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

$\otimes$

1	0	0
0	0	0
0	0	-1

$=$

0	-2	-1
2	2	4
-1	0	0

二维卷积示例

原始图像

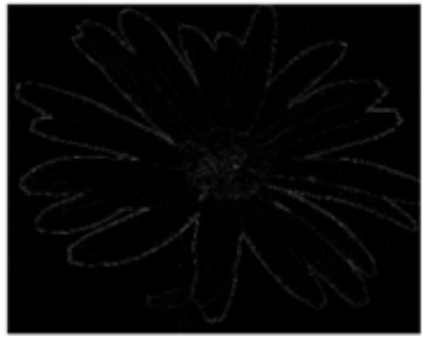
$\otimes$

$\frac{1}{16}$	$\frac{1}{8}$	$\frac{1}{16}$
$\frac{1}{8}$	$\frac{1}{4}$	$\frac{1}{8}$
$\frac{1}{16}$	$\frac{1}{8}$	$\frac{1}{16}$

$=$

0	1	0
1	-4	1
0	1	0

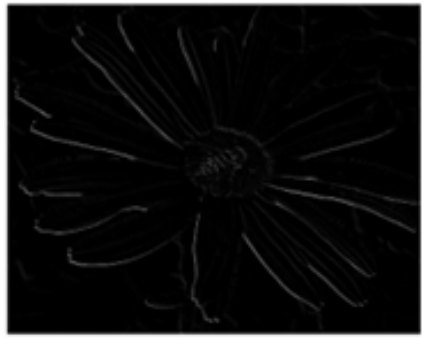
$=$



0	1	1
-1	0	1
-1	-1	0

滤波器

$=$



输出图像

CNN – Convolution

Those are the network parameters to be learned.

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

Matrix

-1	1	-1
-1	1	-1
-1	1	-1

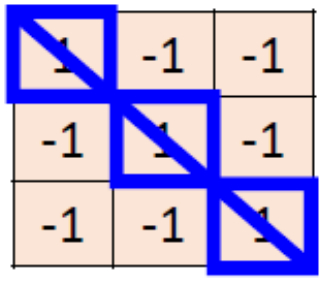
Filter 2

Matrix

⋮

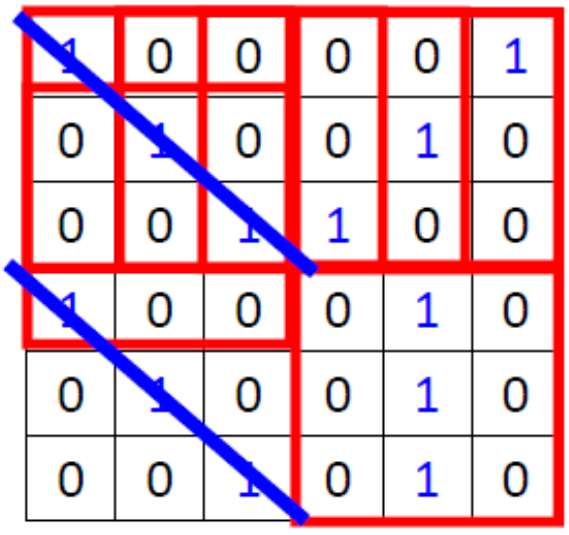
Property 1 Each filter detects a small pattern (3 x 3).

CNN – Convolution

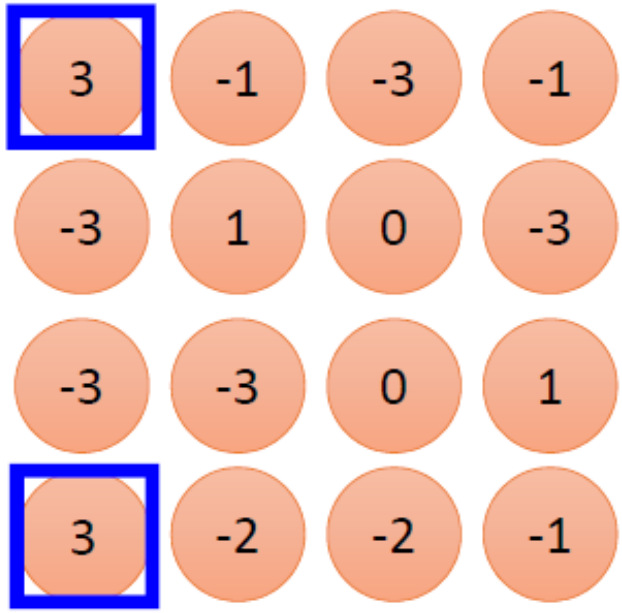


Filter 1

stride=1



6 x 6 image



Property 2

CNN – Convolution

-1	1	-1
-1	1	-1
-1	1	-1

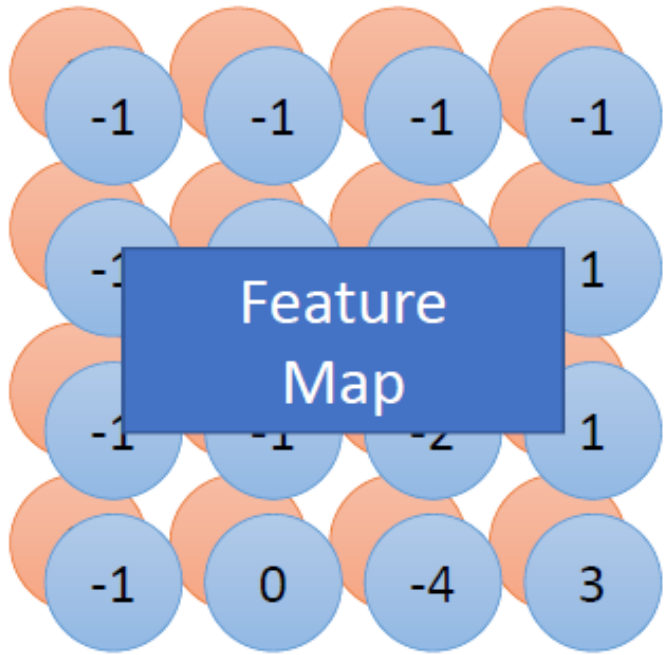
Filter 2

stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

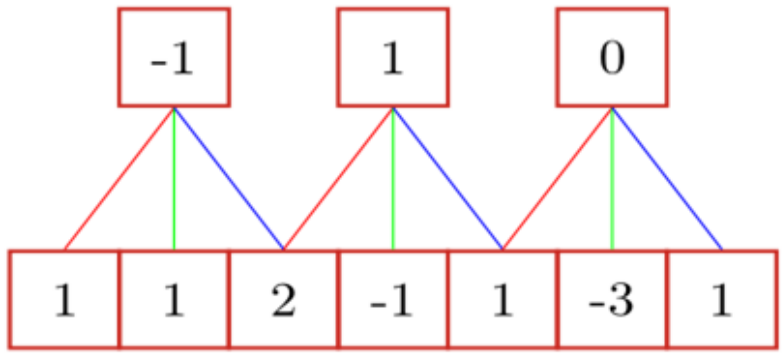
Do the same process for every filter



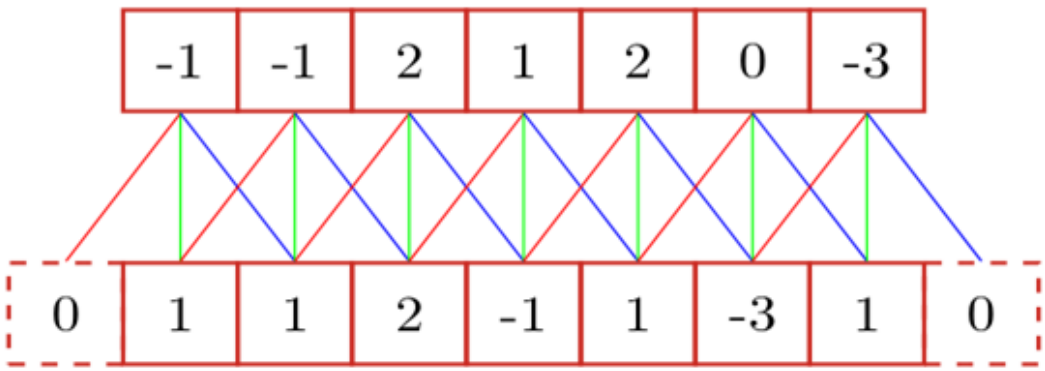
4 x 4 image

卷积扩展

- 引入滤波器的滑动步长s和零填充p



(a) 步长 $s = 2$



(b) 零填充 $p = 1$

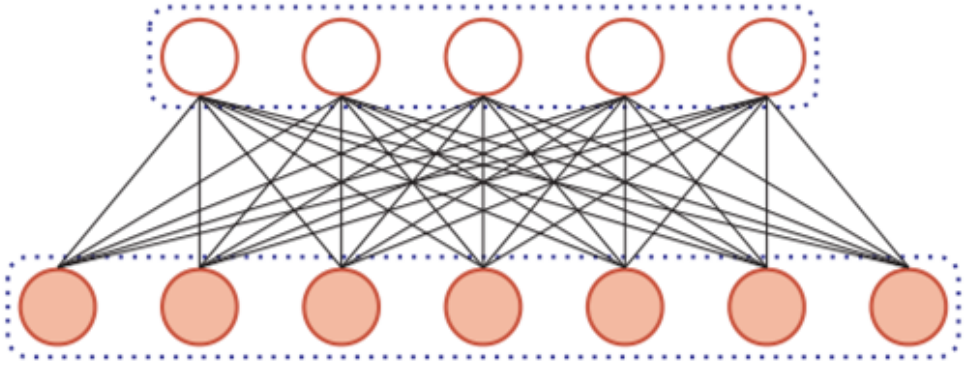
卷积类型

- 卷积的结果按输出长度不同可以分为三类：
  - 窄卷积：步长$s = 1$，两端不补零$p = 0$，卷积后输出长度为$n - m + 1$。
  - 宽卷积：步长$s = 1$，两端补零$p = m - 1$，卷积后输出长度$n + m - 1$。
  - 等长卷积：步长$s = 1$，两端补零$p = (m - 1)/2$，卷积后输出长度n。

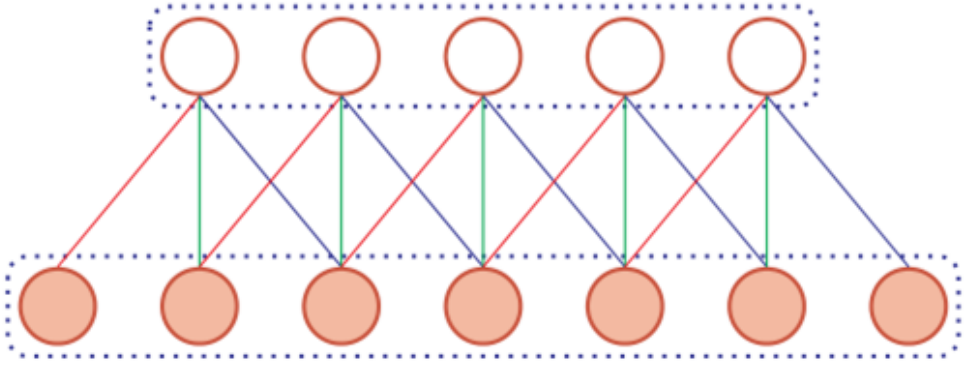
- 在早期的文献中，卷积一般默认为窄卷积。
- 而目前的文献中，卷积一般默认为等宽卷积。

卷积神经网络

- 用卷积层代替全连接层

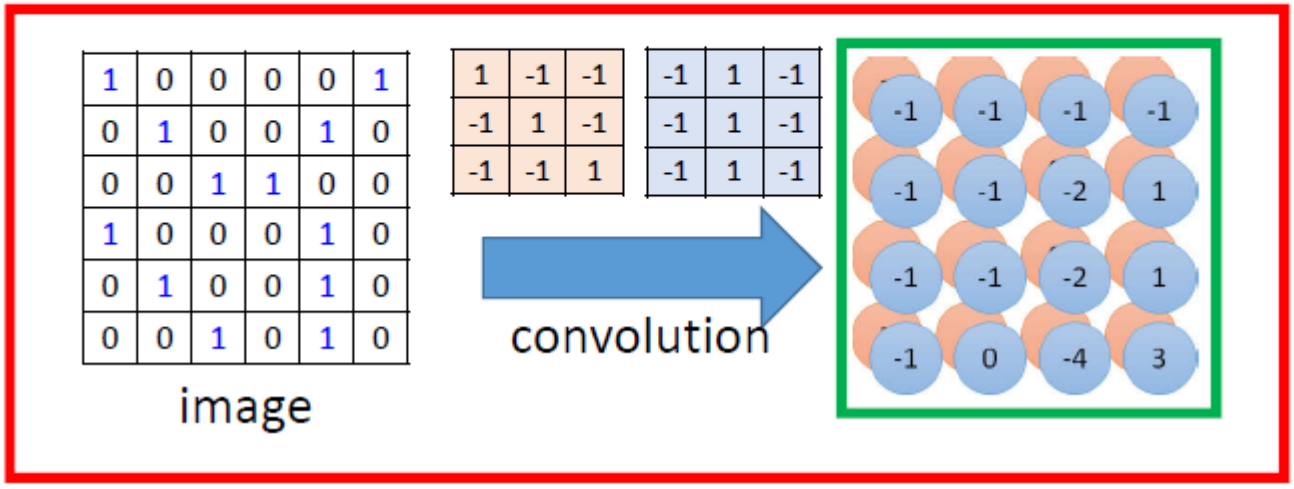


(a) 全连接层

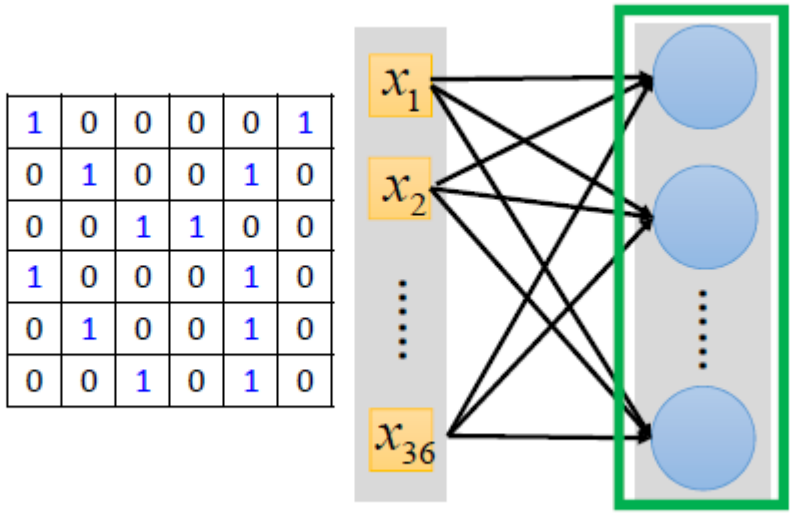


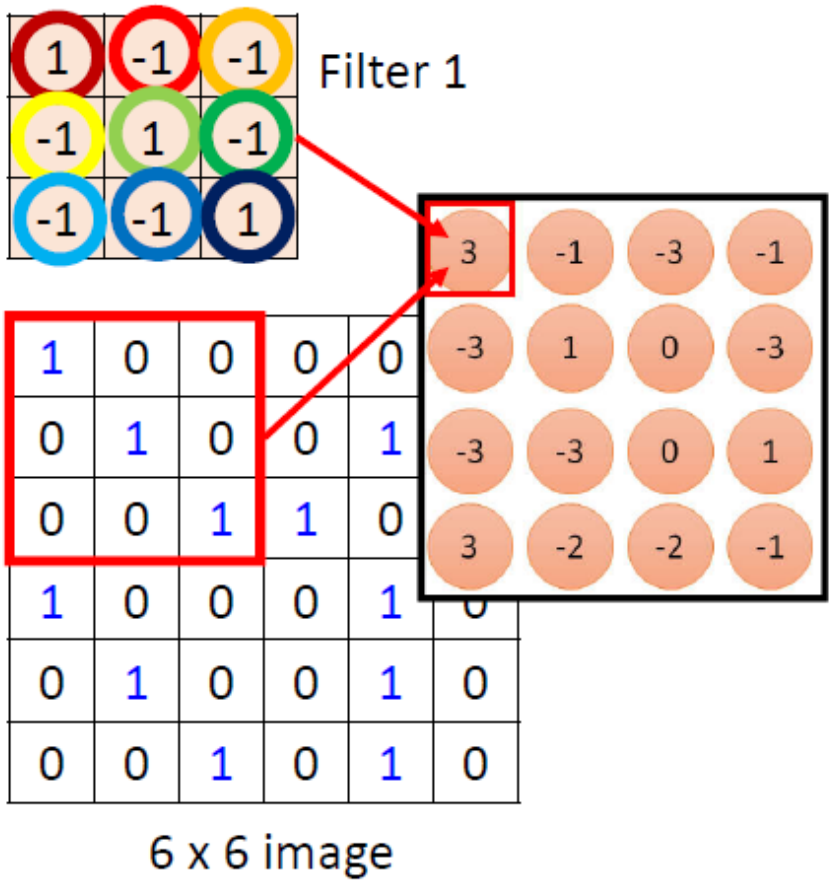
(b) 卷积层

Convolution v.s. Fully Connected

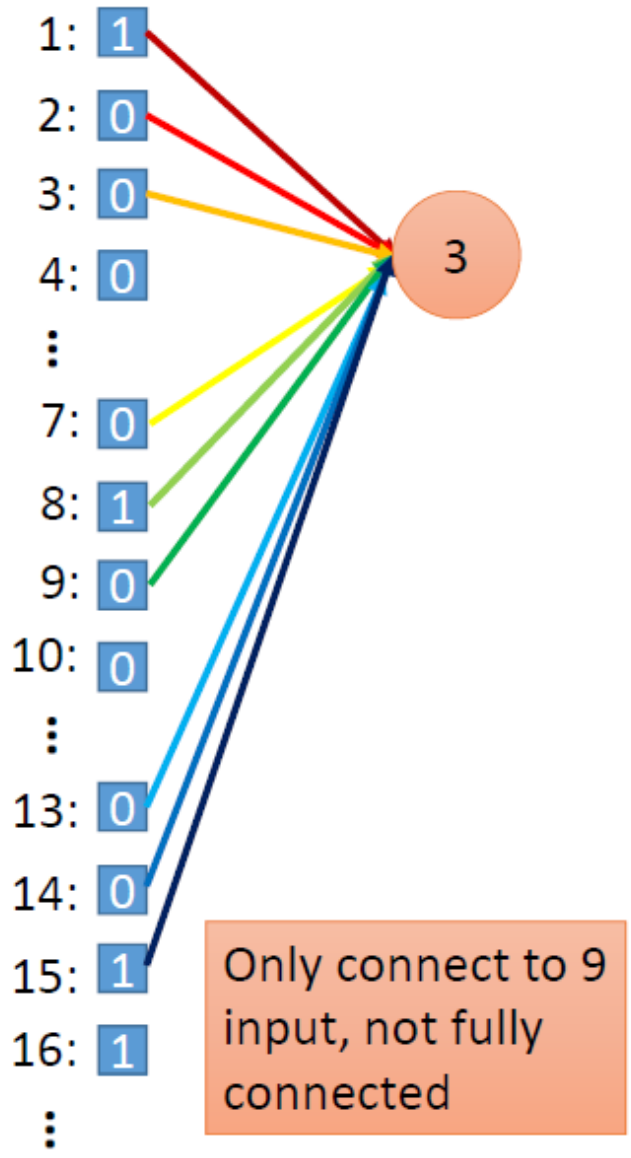


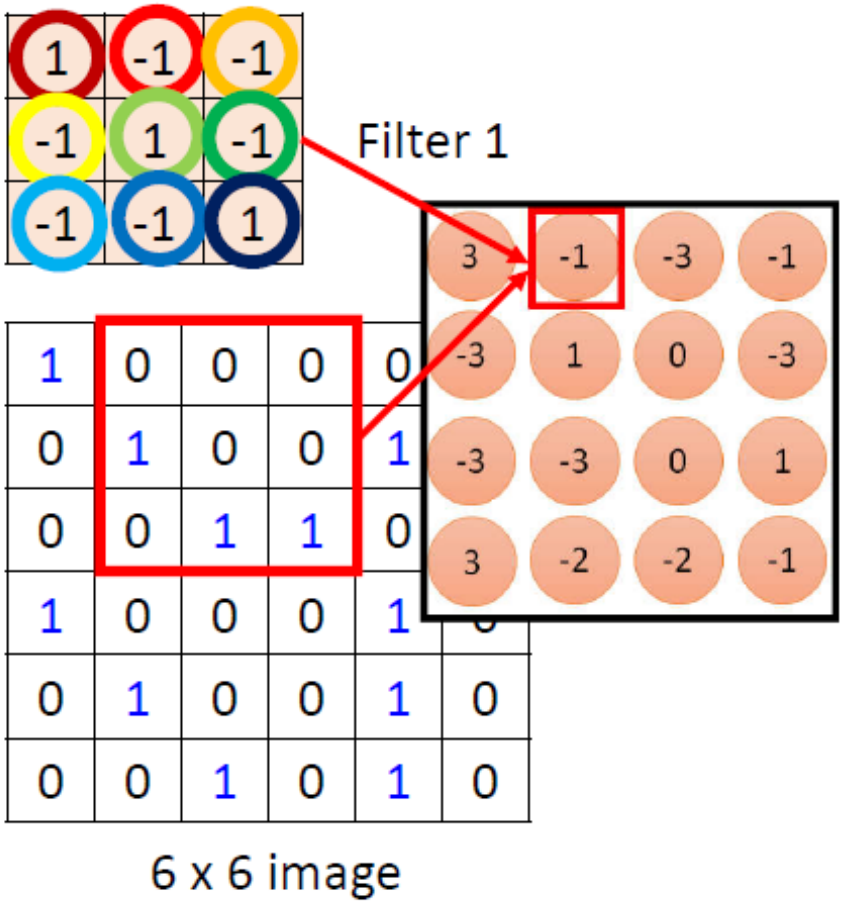
Fully-connected





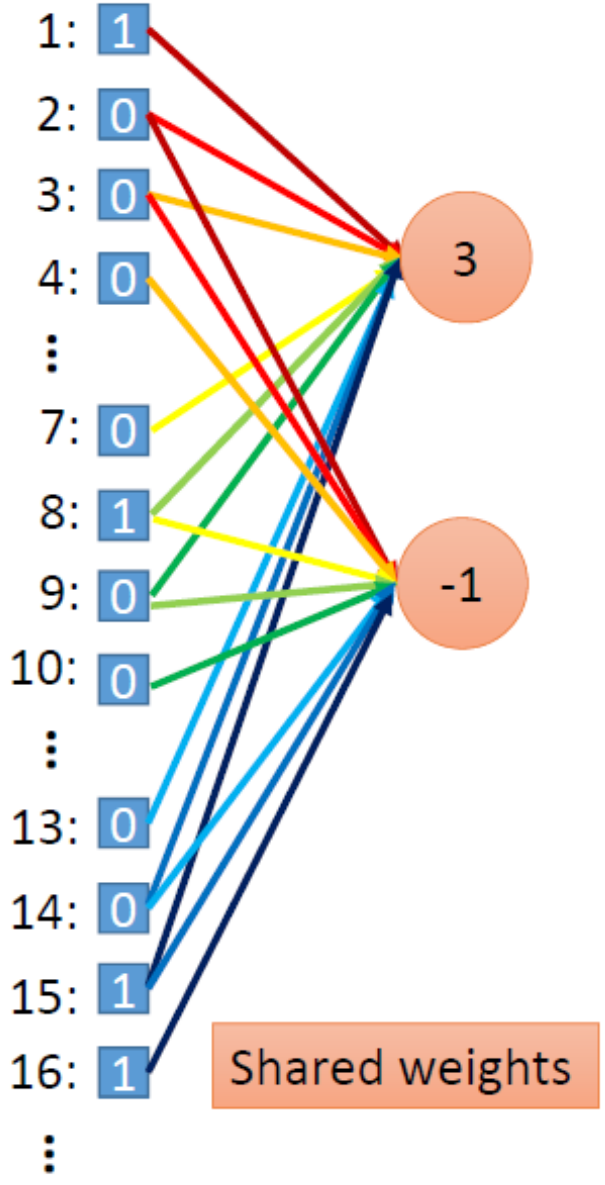
Less parameters!





Less parameters!

Even less parameters!



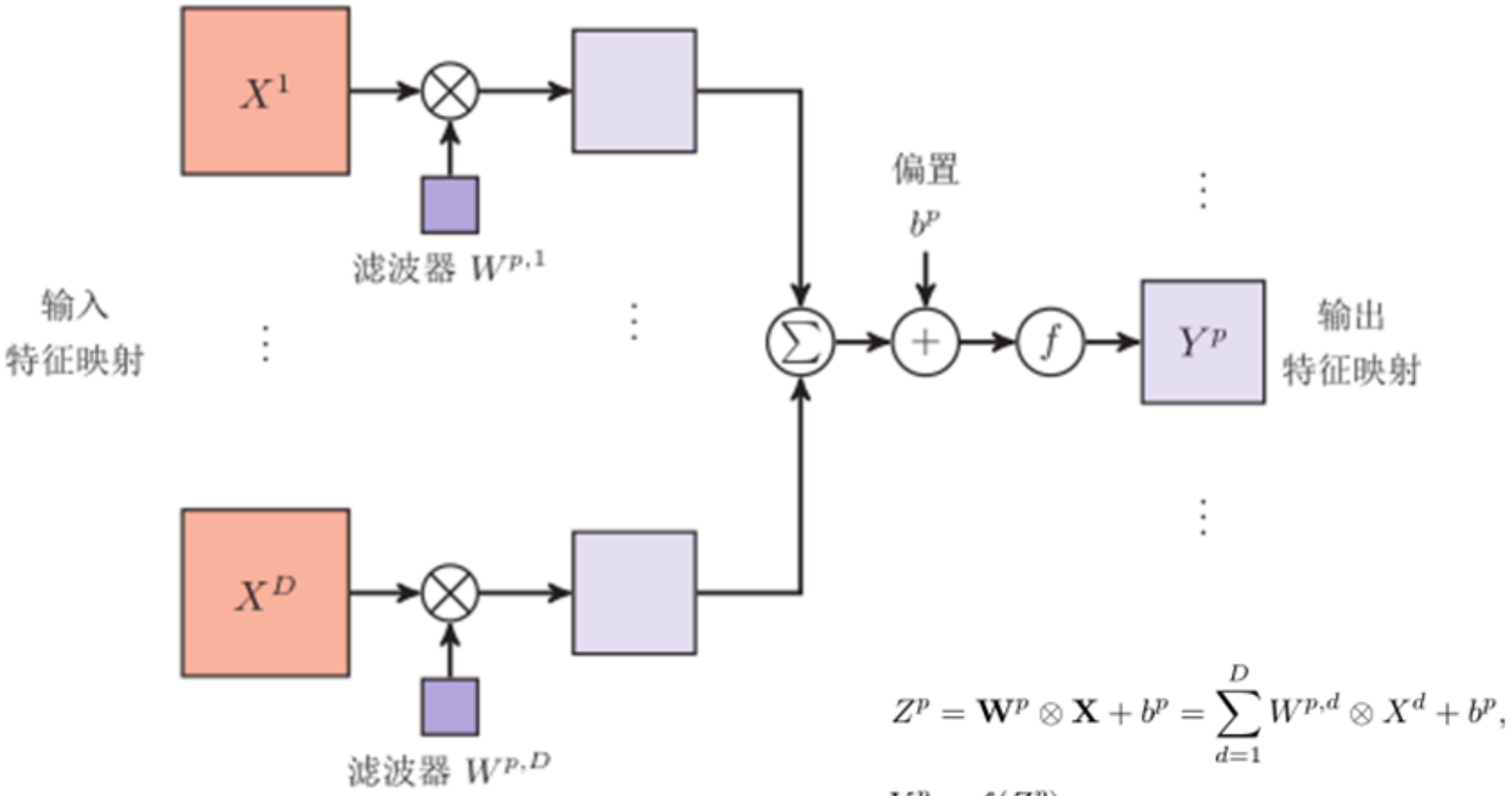
卷积层

- 输入：D个特征映射 $M \times N \times D$
- 输出：P个特征映射 $M' \times N' \times P$

- 特征映射（Feature Map）：一幅图像经过卷积后得到的特征。
  - 卷积核看成一个特征提取器

- 典型的卷积层可以表示成3维结构

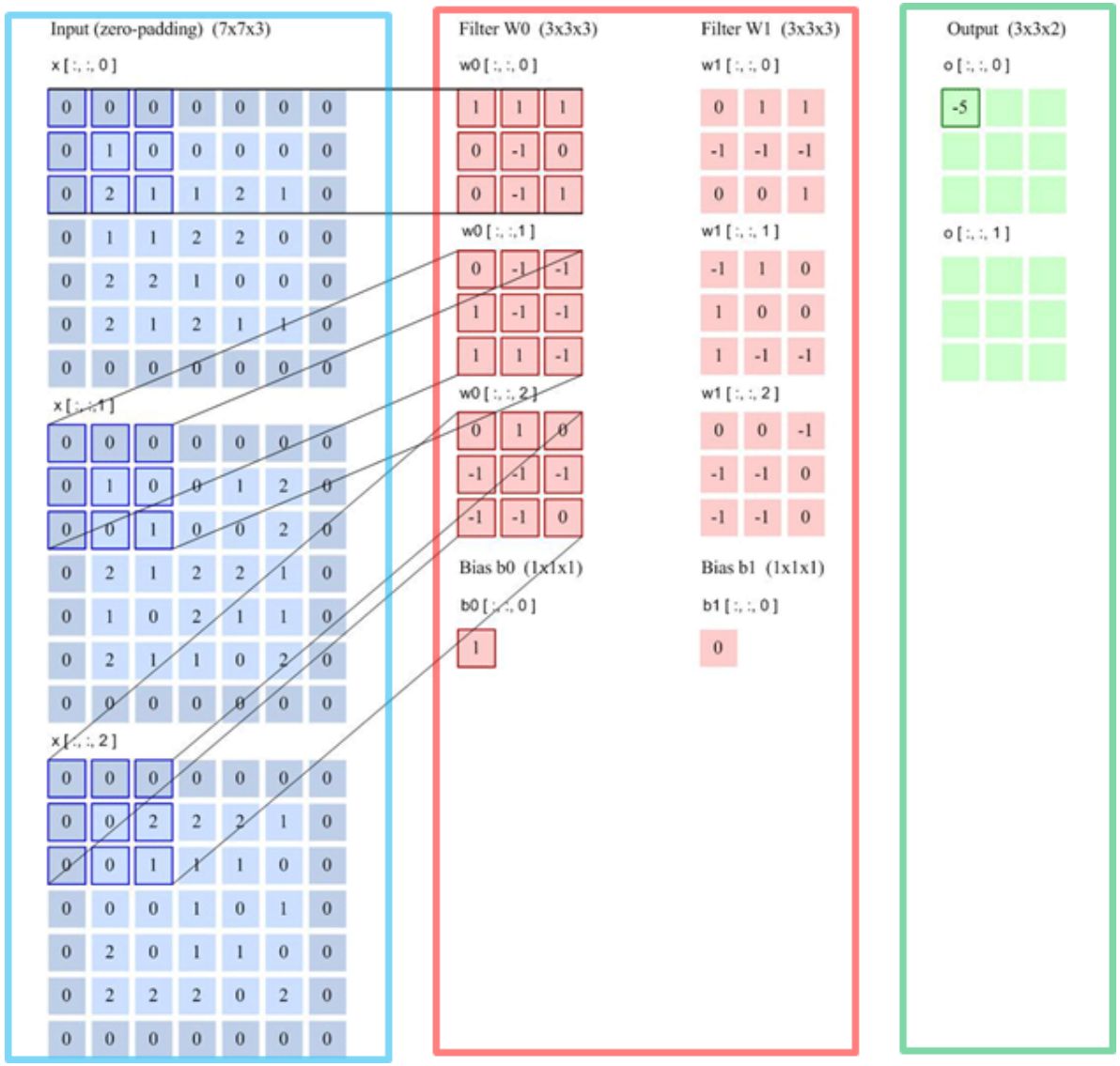
卷积层的映射关系



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

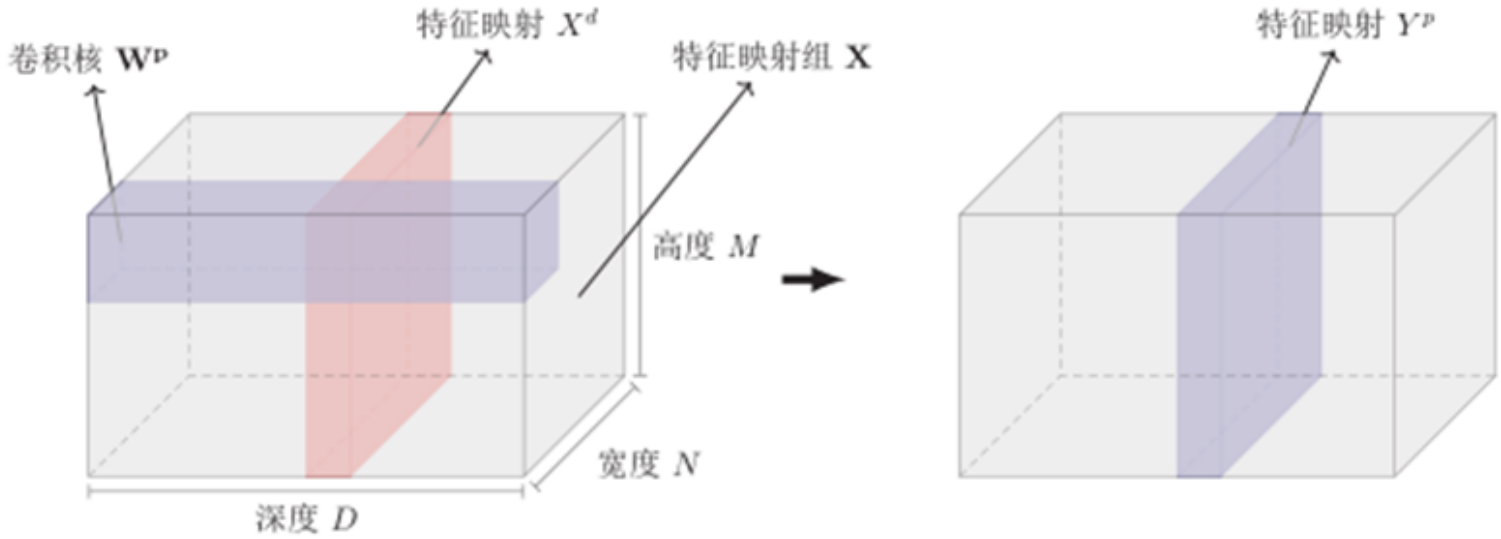
$$Y^p = f(Z^p).$$

步长 2 filter 个数 2 3×3 填充



卷积层

- 典型的卷积层为3维结构

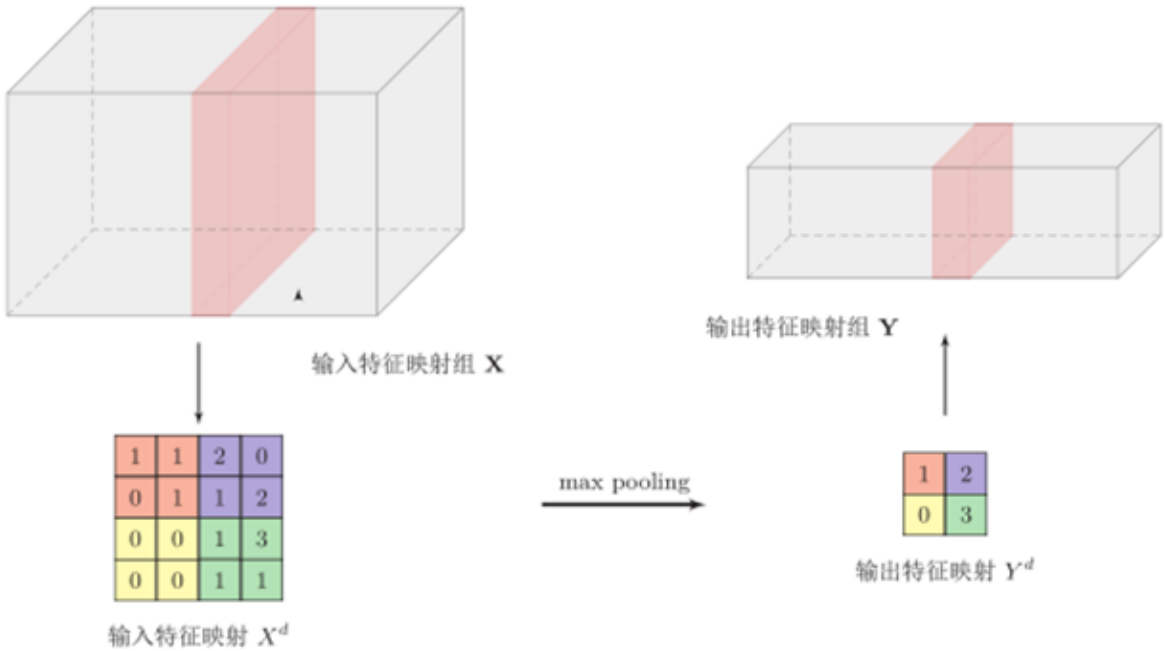


$$Z^p = \mathbf{W}^p \otimes \mathbf{X} + b^p = \sum_{d=1}^{D} W^{p,d} \otimes X^d + b^p,$$

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

汇聚层（Polling Layer or Subsampling Layer）

- 卷积层虽然可以显著减少连接的个数，但是每一个特征映射的神经元个数并没有显著减少。



常用汇聚函数

- 最大汇聚 (maximum pooling): 一般是取一个区域内所有神经元的最大值

$$Y_{m,n}^{d} = \max_{i \in R_{m,n}^{d}} x_i$$

- 平均汇聚 (mean pooling): 一般是取区域内所有神经元的平均值

$$Y_{m,n}^{d} = \frac{1}{|R_{m,n}^{d}|} \sum_{i \in R_{m,n}^{d}} x_i$$

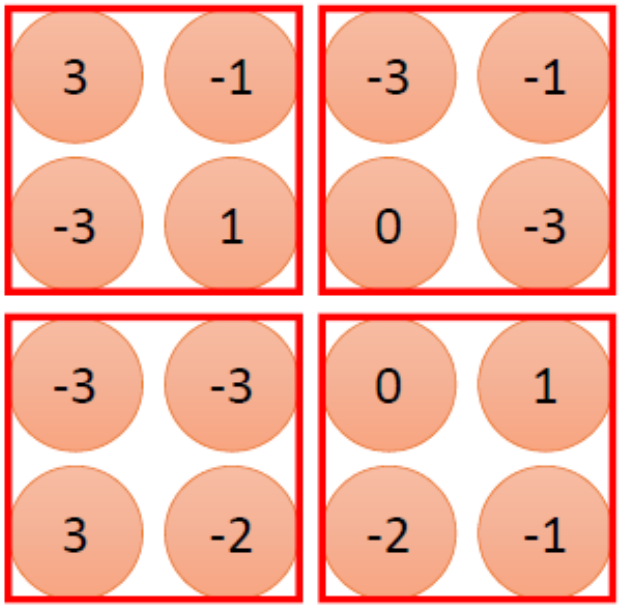
CNN – Max Pooling

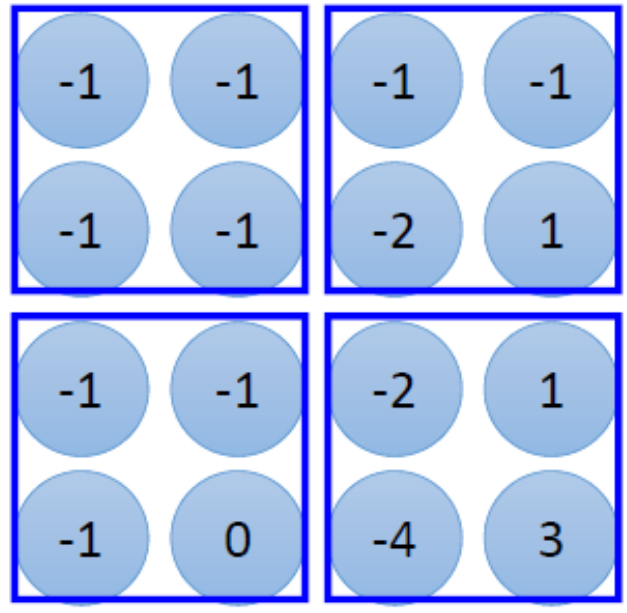
1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

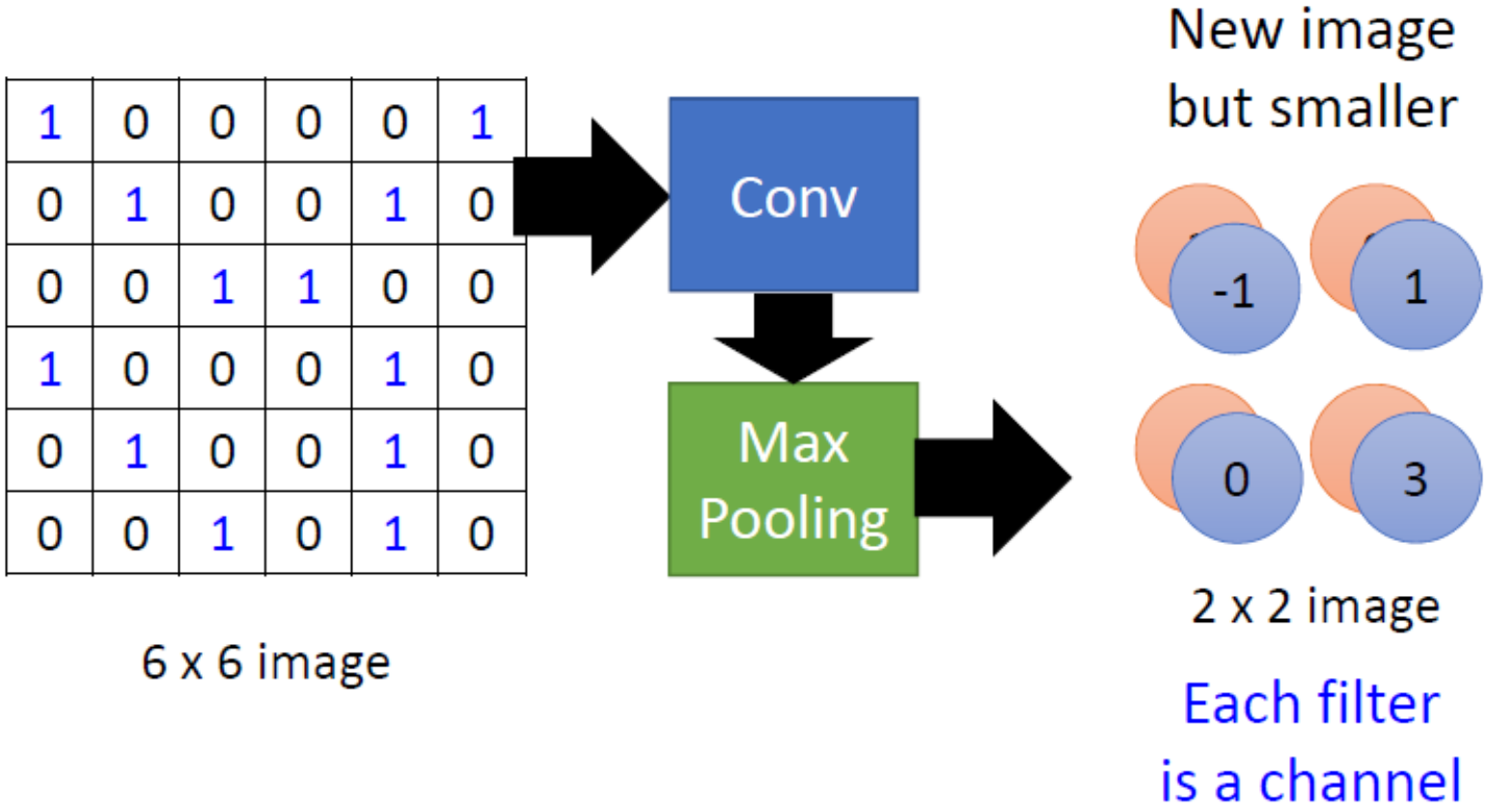
-1	1	-1
-1	1	-1
-1	1	-1

Filter 2

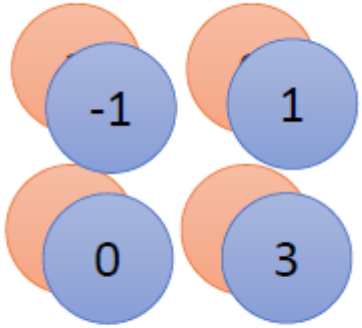




CNN – Max Pooling



The whole CNN



A new image

Convolution

Max Pooling

Convolution

Max Pooling

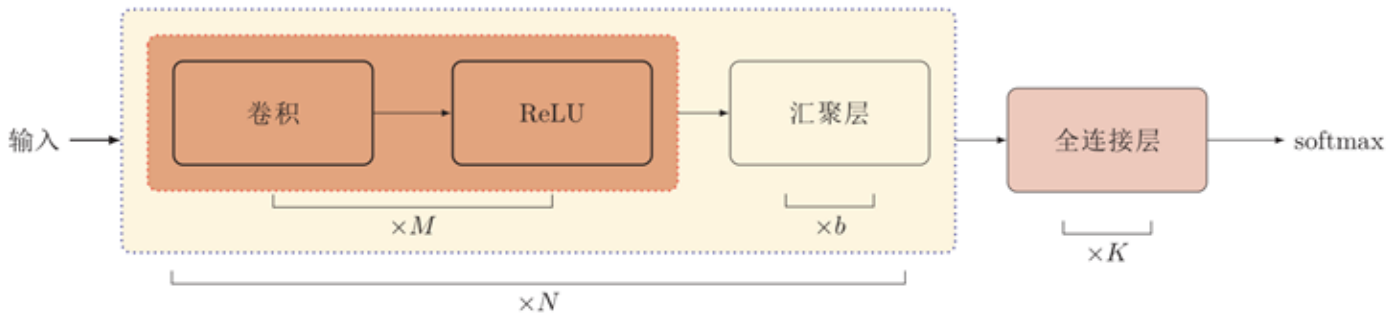
Can repeat many times

Smaller than the original image

The number of the channel is the number of filters

卷积网络结构

- 卷积网络是由卷积层、子采样层、全连接层交叉堆叠而成。
  - 趋向于小卷积、大深度
  - 趋向于全卷积
- 典型结构



- 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）。一个卷积网络中可以堆叠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）。

The whole CNN

Convolution

Max Pooling

A new image

Convolution

Max Pooling

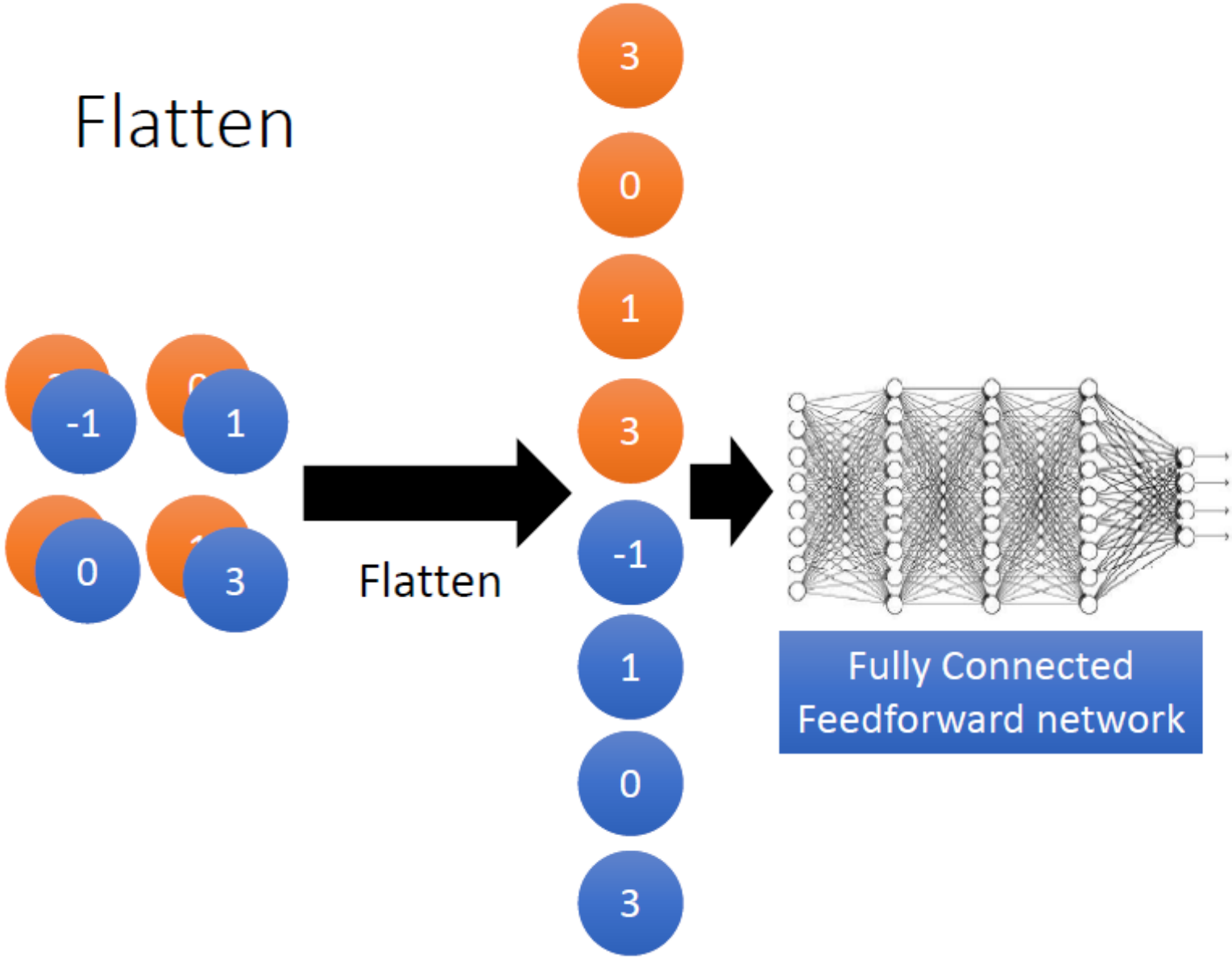
A new image

Flatten

Fully Connected Feedforward network

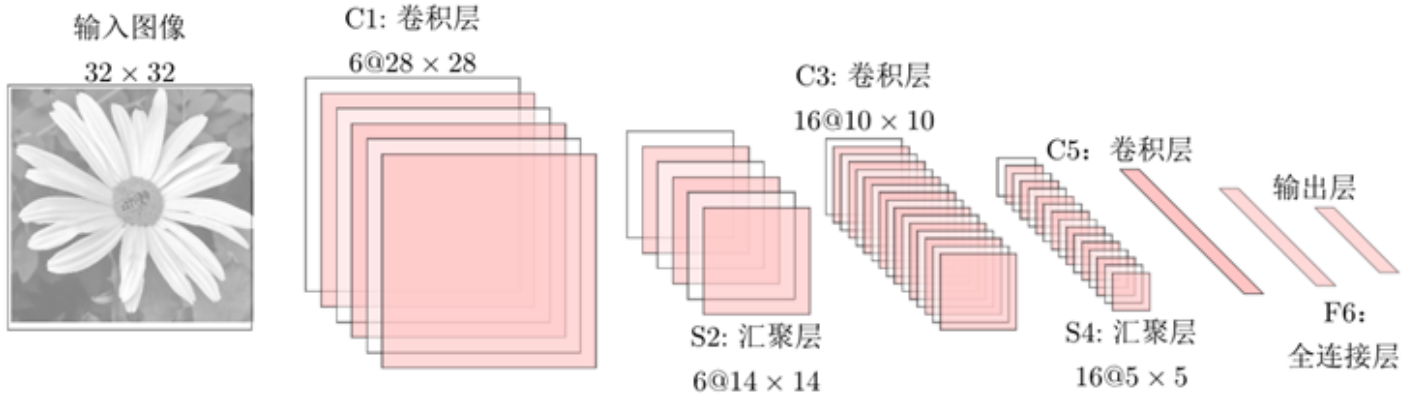
cat dog ……

Flatten


-1
1
0
3
Flatten
3
0
1
3
-1
1
0
3
Fully Connected
Feedforward network

LeNet-5

- LeNet-5 是一个非常成功的神经网络模型。
  - 基于 LeNet-5 的手写数字识别系统在 90 年代被美国很多银行使用，用来识别支票上面的手写数字。
  - LeNet-5 共有 7 层。



AlexNet

- 2012 ILSVRC winner
  - (top 5 error of 16% compared to runner-up with 26% error)
  - 第一个现代深度卷积网络模型，首次使用了很多现代深度卷积网络的一些技术方法，
    - 比如使用GPU进行并行训练，采用了ReLU作为非线性激活函数，使用Dropout防止过拟合，使用数据增强
  - 共有8层，其中前5层卷积层，后边3层全连接层

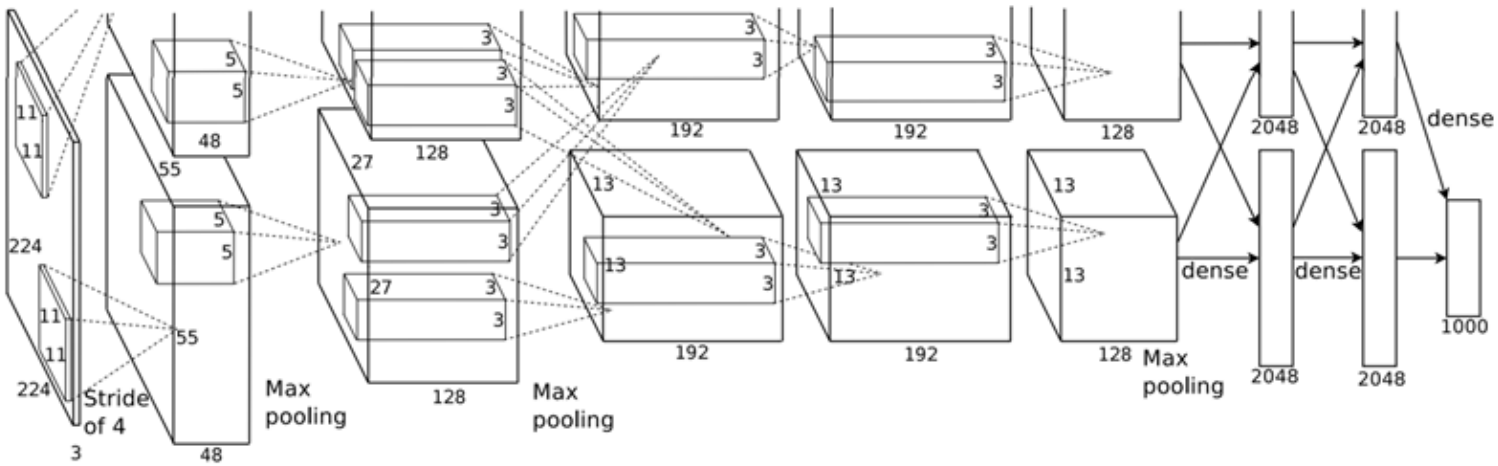


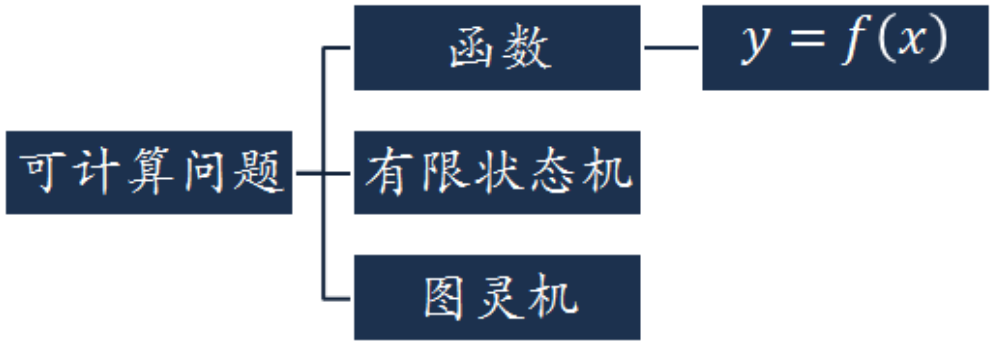
Table of Contents

前馈网络的一些不足

- 连接存在层与层之间，每层的节点之间是无连接的。（无循环）
- 输入和输出的维数都是固定的，不能任意改变。无法处理变长的序列数据。
- 假设每次输入都是独立的，也就是说每次网络的输出只依赖于当前的输入。

可计算问题

- 前馈网络
  - 通用近似定理

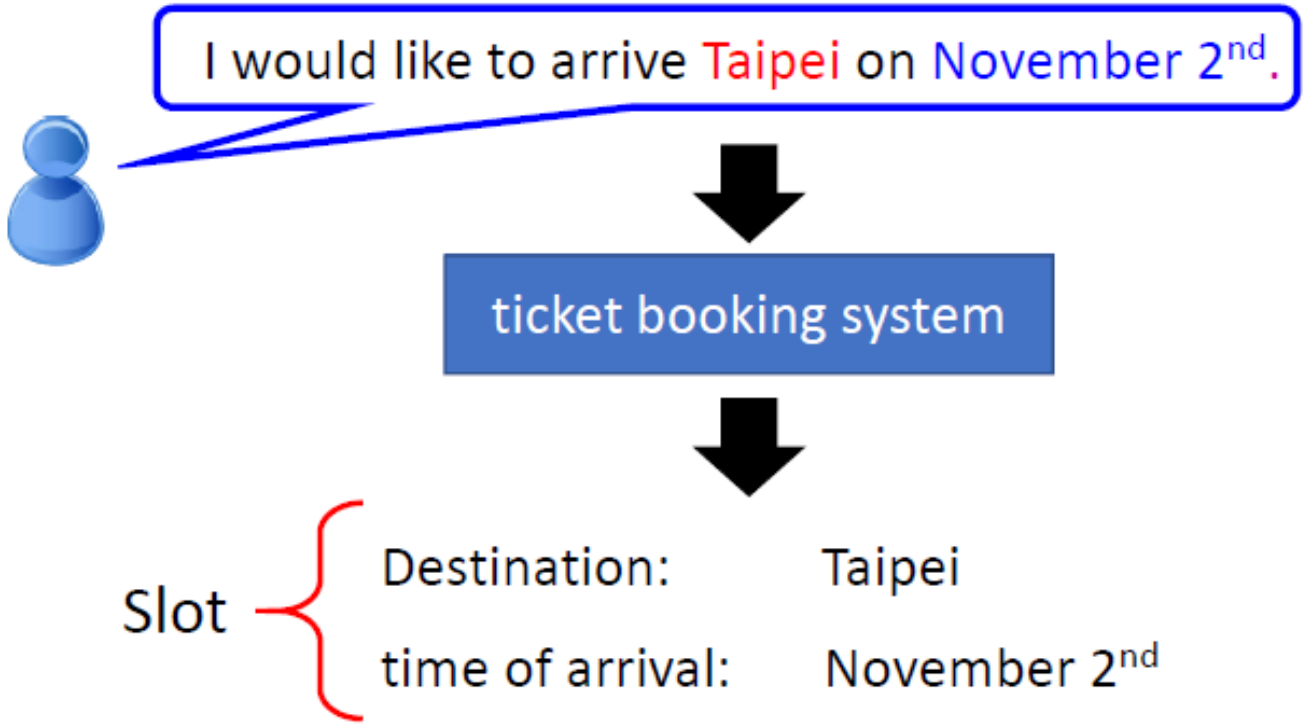


前馈网络

如何用FNN去模拟一个有限状态自动机?

Example

- Slot Filling

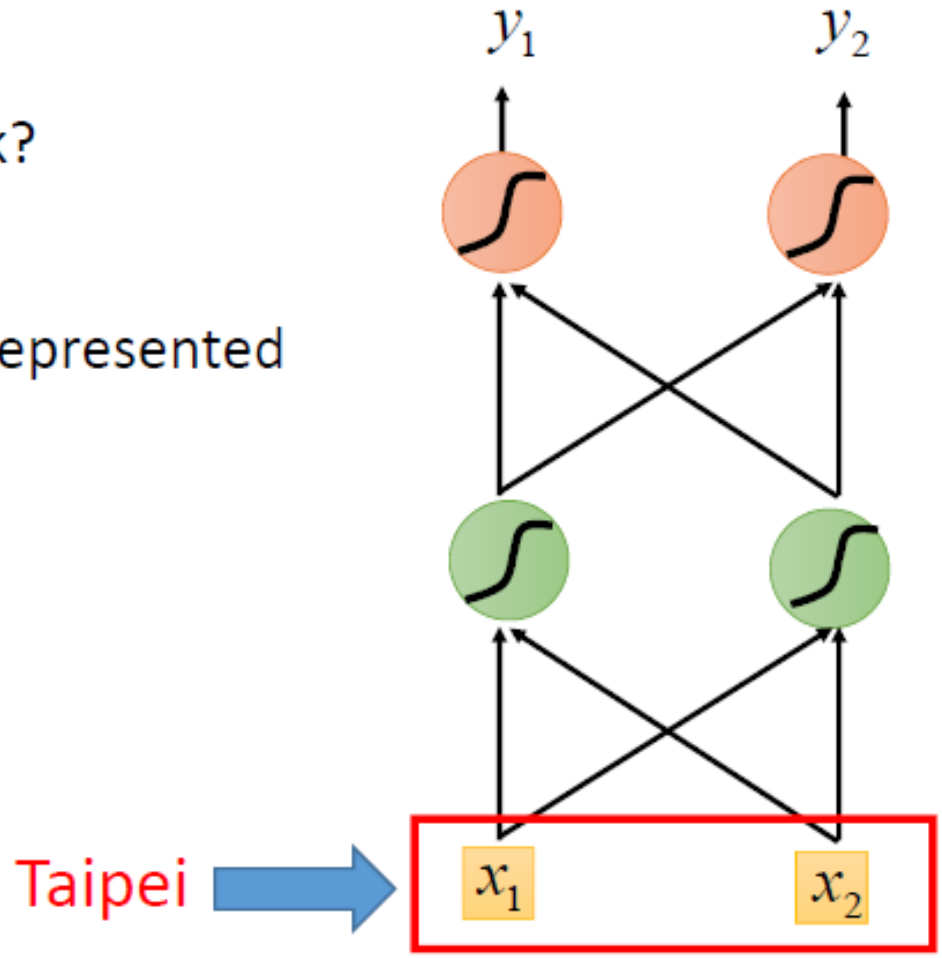


Example

Solving slot filling by Feedforward network?

Input: a word

(Each word is represented as a vector)



Example Application

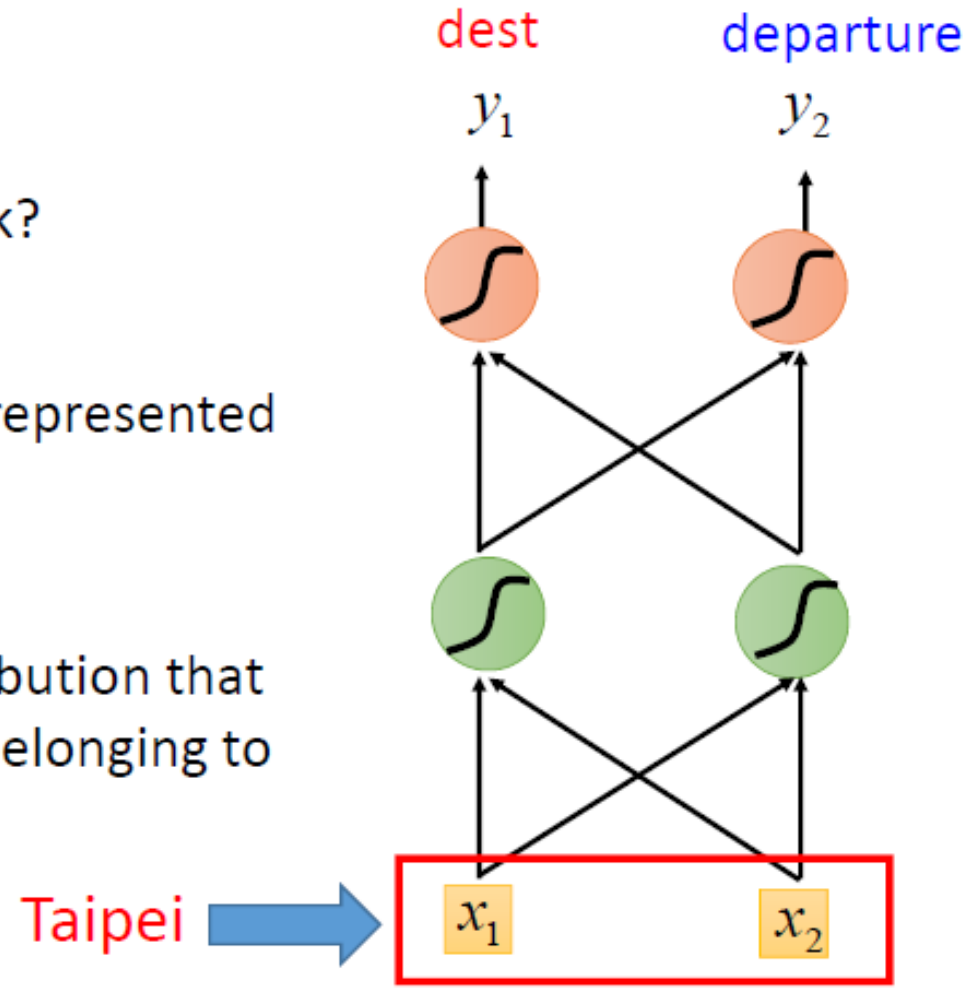
Solving slot filling by Feedforward network?

Input: a word

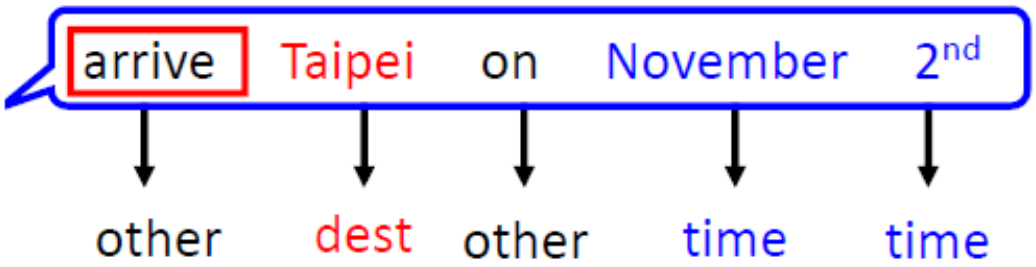
(Each word is represented as a vector)

Output:

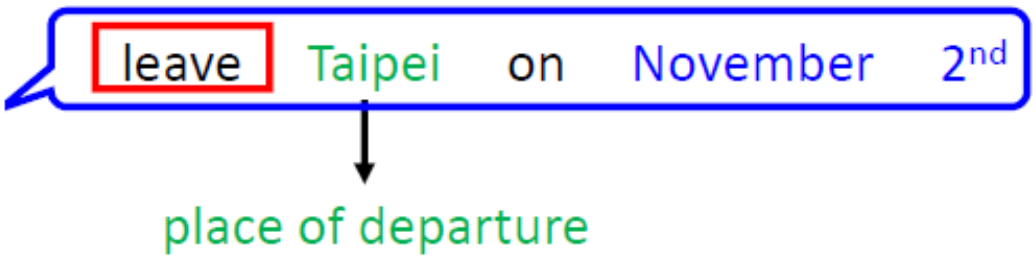
Probability distribution that the input word belonging to the slots



Example Application

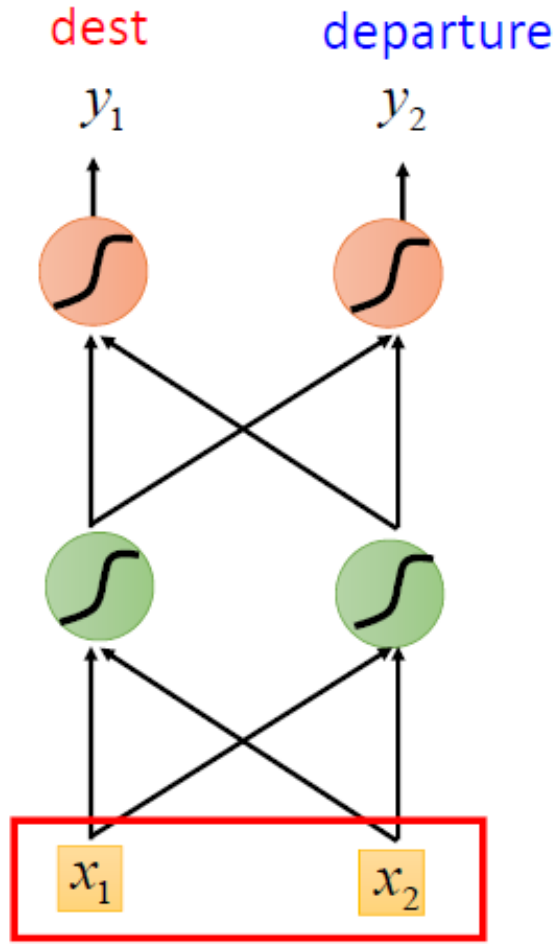


Problem?



Neural network needs memory!

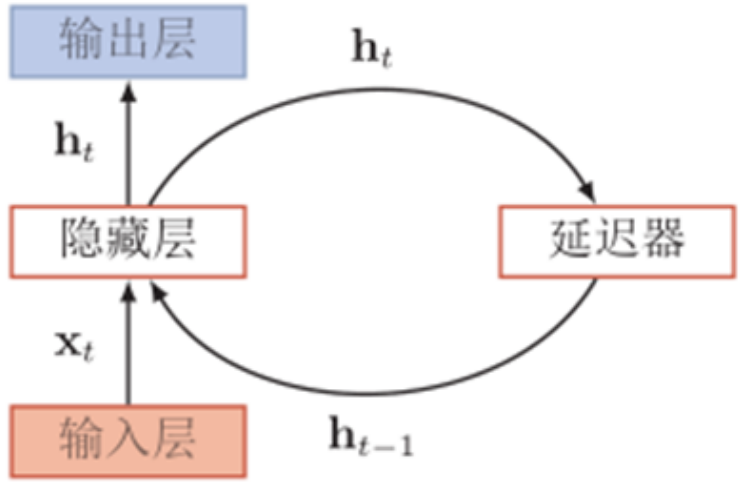
Taipei



循环神经网络

给定一个输入序列$\mathbf{x}_{1:T} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_t, \ldots, \mathbf{x}_T)$，循环神经网络通过下面公式更新带反馈边的隐藏层的**活性值**$\mathbf{h}_t$：

$$\mathbf{h}_t = \begin{cases} 0 & t = 0 \\ f(\mathbf{h}_{t-1}, \mathbf{x}_t) & \text{otherwise} \end{cases}$$

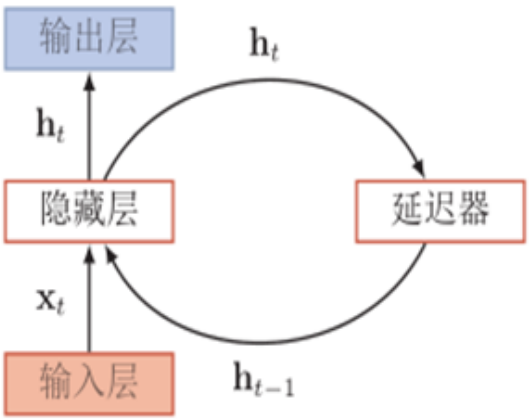


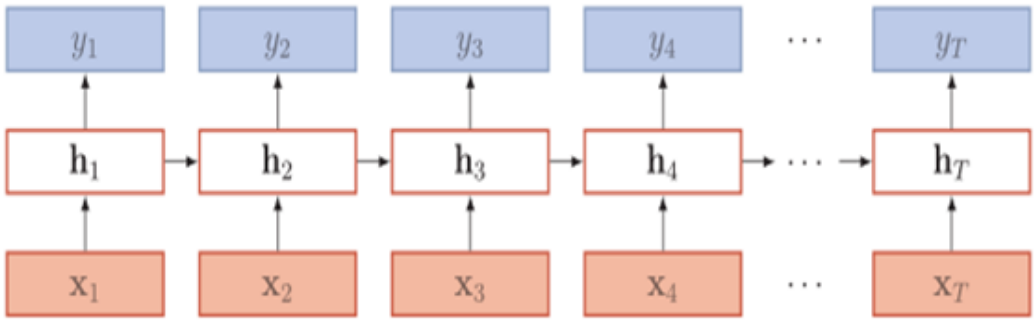
RNN 是图灵完全等价的 (Siegelmann and Sontag, 1995)

FNN：模拟任何函数

RNN：模拟任何程序（计算过程）

按时间展开


输出层
$\mathbf{h}_t$
$\mathbf{h}_t$
隐藏层
延迟器
$\mathbf{x}_t$
输入层
$\mathbf{h}_{t-1}$


y_1
y_2
y_3
y_4
$\cdots$
y_T
$\mathbf{h}_1$
$\mathbf{h}_2$
$\mathbf{h}_3$
$\mathbf{h}_4$
$\cdots$
$\mathbf{h}_T$
$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_3$
$\mathbf{x}_4$
$\cdots$
$\mathbf{x}_T$

循环神经网络

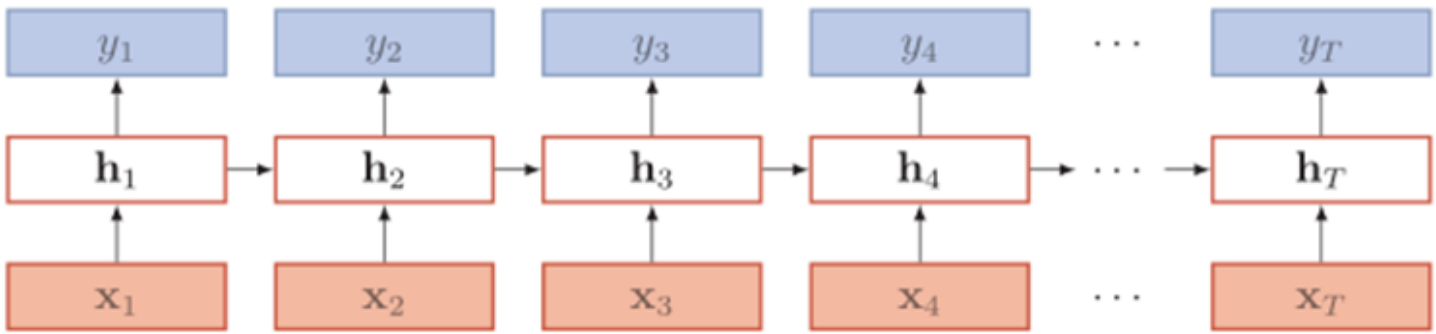
- 循环神经网络
  - 循环神经网络通过使用带自反馈的神经元，能够处理任意长度的序列。
  - 循环神经网络比前馈神经网络更加符合生物神经网络的结构。
  - 循环神经网络已经被广泛应用在语音识别、语言模型以及自然语言生成等任务上。

简单循环网络（Simple Recurrent Network）

只有一个隐藏层的神经网络，隐藏层状态 h_t 不仅与输入 x_t 相关，也与上一时刻的隐藏层状态 h_{t-1} 相关

- 状态更新：

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



图灵完备

- 图灵完备（Turing Completeness）是指一种数据操作规则，比如一种计算机编程语言，可以实现图灵机的所有功能，解决所有的可计算问题。

> **定理 6.2 – 图灵完备 [Siegelmann and Sontag, 1991]：** 所有的图灵机都可以被一个由使用 sigmoid 型激活函数的神经元构成的全连接循环网络来进行模拟。

- 一个完全连接的循环神经网络可以近似解决所有的可计算问题。

参数学习

- 机器学习
  - 给定一个训练样本(x,y)，
    - 其中$x = (x_1, \cdots, x_T)$为长度是T 的输入序列，
    - $y = (y_1, \cdots, y_T)$是长度为T 的标签序列。

- 时刻t的瞬时损失函数为

$$\mathcal{L}_t = \mathcal{L}(\mathbf{y}_t, g(\mathbf{h}_t)),$$

- 总损失函数

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

梯度

- 链式法则

$$\frac{\partial \mathcal{L}_t}{\partial U_{ij}} = \sum_{k=1}^{t} tr\left((\frac{\partial \mathcal{L}_t}{\partial \mathbf{z}_k})^{\mathrm{T}} \frac{\partial^+ \mathbf{z}_k}{\partial U_{ij}} \right)$$

$$= \sum_{k=1}^{t} \left(\frac{\partial^+ \mathbf{z}_k}{\partial U_{ij}} \right)^{\mathrm{T}} \frac{\partial \mathcal{L}_t}{\partial \mathbf{z}_k},$$

$\delta_{t,k}$为第t时刻的损失对第k步隐藏神经元的净输入z_k的导数

$$\frac{\partial^+ \mathbf{z}_k}{\partial U_{ij}} = \begin{bmatrix} 0 \\ \vdots \\ [\mathbf{h}_k]_j \\ \vdots \\ 0 \end{bmatrix} \triangleq \mathbb{I}_i([\mathbf{h}_k]_j),$$

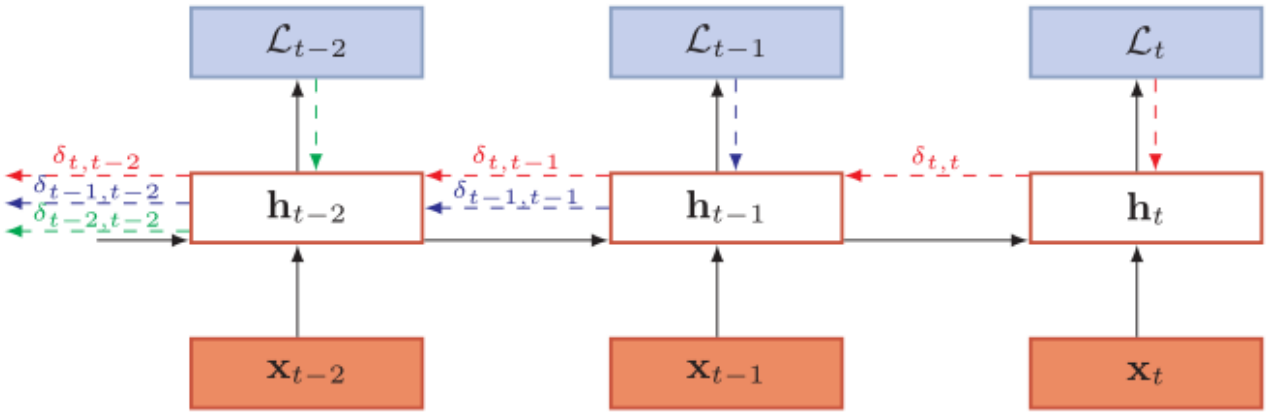
$$\delta_{t,k} = \frac{\partial \mathcal{L}_t}{\partial \mathbf{z}_k}$$

$$= \frac{\partial \mathbf{h}_k}{\partial \mathbf{z}_k} \frac{\partial \mathbf{z}_{k+1}}{\partial \mathbf{h}_k} \frac{\partial \mathcal{L}_t}{\partial \mathbf{z}_{k+1}}$$

$$= \mathbf{diag}(f'(\mathbf{z}_k)) U^{\mathrm{T}} \delta_{t,k+1}.$$

梯度

- 随时间反向传播算法



梯度消失/爆炸

- 梯度

$$\frac{\partial \mathcal{L}}{\partial U} = \sum_{t=1}^{T}\sum_{k=1}^{t} \delta_{t,k}\mathbf{h}_k^{\mathrm{T}}.$$

- 其中

$$\delta_{t,k} = \prod_{i=k}^{t-1}\Big(\mathbf{diag}(f'(\mathbf{z}_i))U^{\mathrm{T}}\Big)\delta_{t,t}.$$

如果定义$\gamma \cong \|\mathrm{diag}(f'(\mathbf{z}_i))U^{\mathrm{T}}\|$，则

$$\delta_{t,k} = \gamma^{t-k}\delta_{t,t}. \tag{6.46}$$

若$\gamma > 1$，当$t-k \to \infty$时，$\gamma^{t-k} \to \infty$，会造成系统不稳定，称为梯度爆炸问题（Gradient Exploding Problem）；相反，若$\gamma < 1$，当$t-k \to \infty$时，$\gamma^{t-k} \to 0$，会出现和深度前馈神经网络类似的梯度消失问题（gradient vanishing problem）

由于梯度爆炸或消失问题，实际上只能学习到短周期的依赖关系。这就是所谓的长期依赖问题。

长期依赖问题

- 循环神经网络在时间维度上非常深！
  - 梯度消失或梯度爆炸
- 如何改进？
  - 梯度爆炸问题
    - 权重衰减
    - 梯度截断
  - 梯度消失问题
    - 改进模型

长期依赖问题

- 改进方法
  - 循环边改为线性依赖关系

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

但这种改变也丢失了神经元在反馈边上的非线性激活的性质，因此也降低了模型的表示能力

- 增加非线性

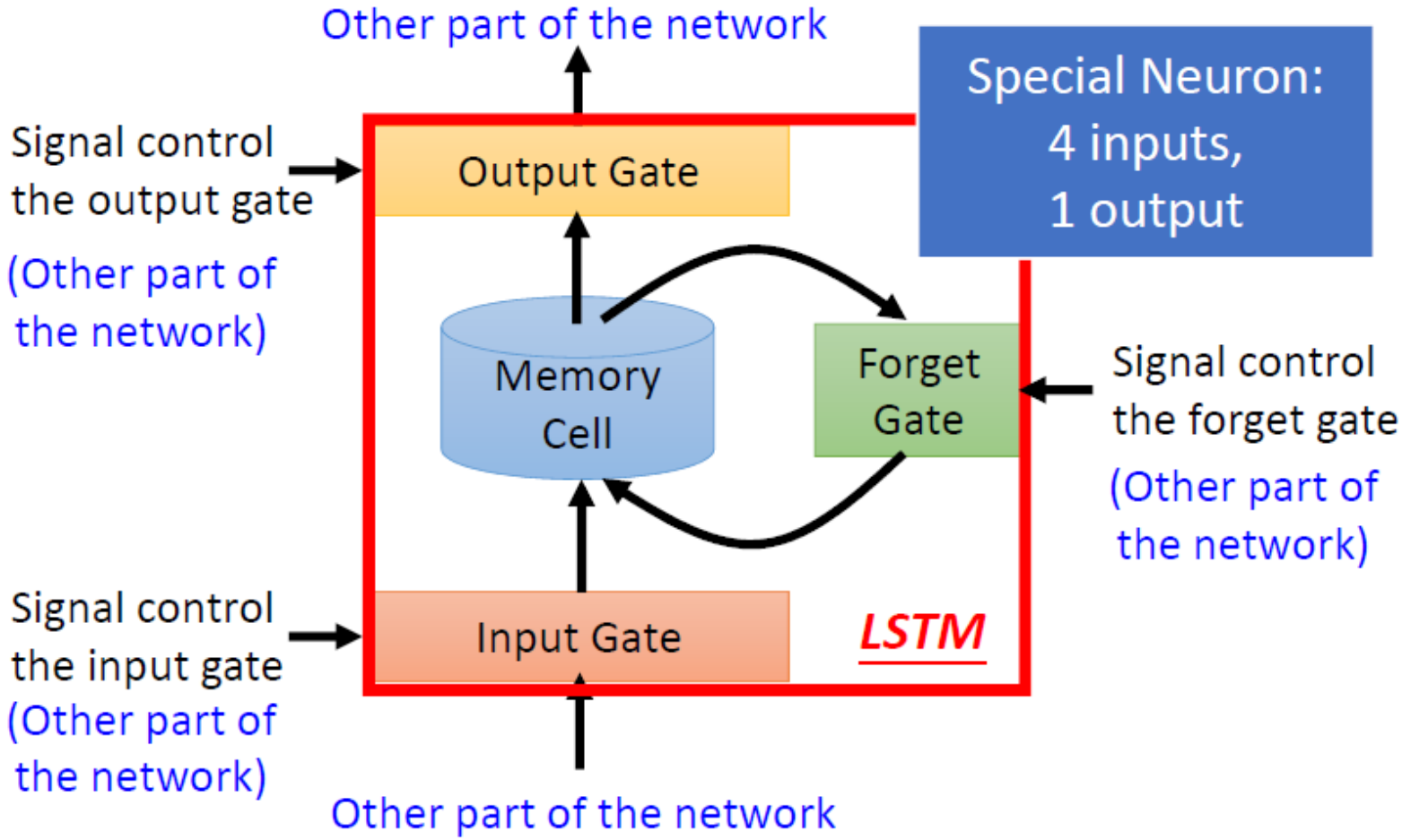
$$\mathbf{h}_t = \mathbf{h}_{t-1} + g(\mathbf{x}_t, \mathbf{h}_{t-1}; \theta),$$

这样 h_t 和 h_{t-1} 之间为既有线性关系，也有非线性关系，在一定程度上可以缓解梯度消失问题

这种改进依然有一个问题就是记忆容量 (memory capacity)。随着 h_t 不断累积存储新的输入信息，会发生饱和现象。即，隐藏状态 h_t 可以存储的信息是有限的，随着记忆单元存储的内容越来越多，其丢失的信息也越来越多

可以进行选择性的遗忘，同时也进行有选择的更新，LSTM 就是这个目的

Long Short-term Memory (LSTM)



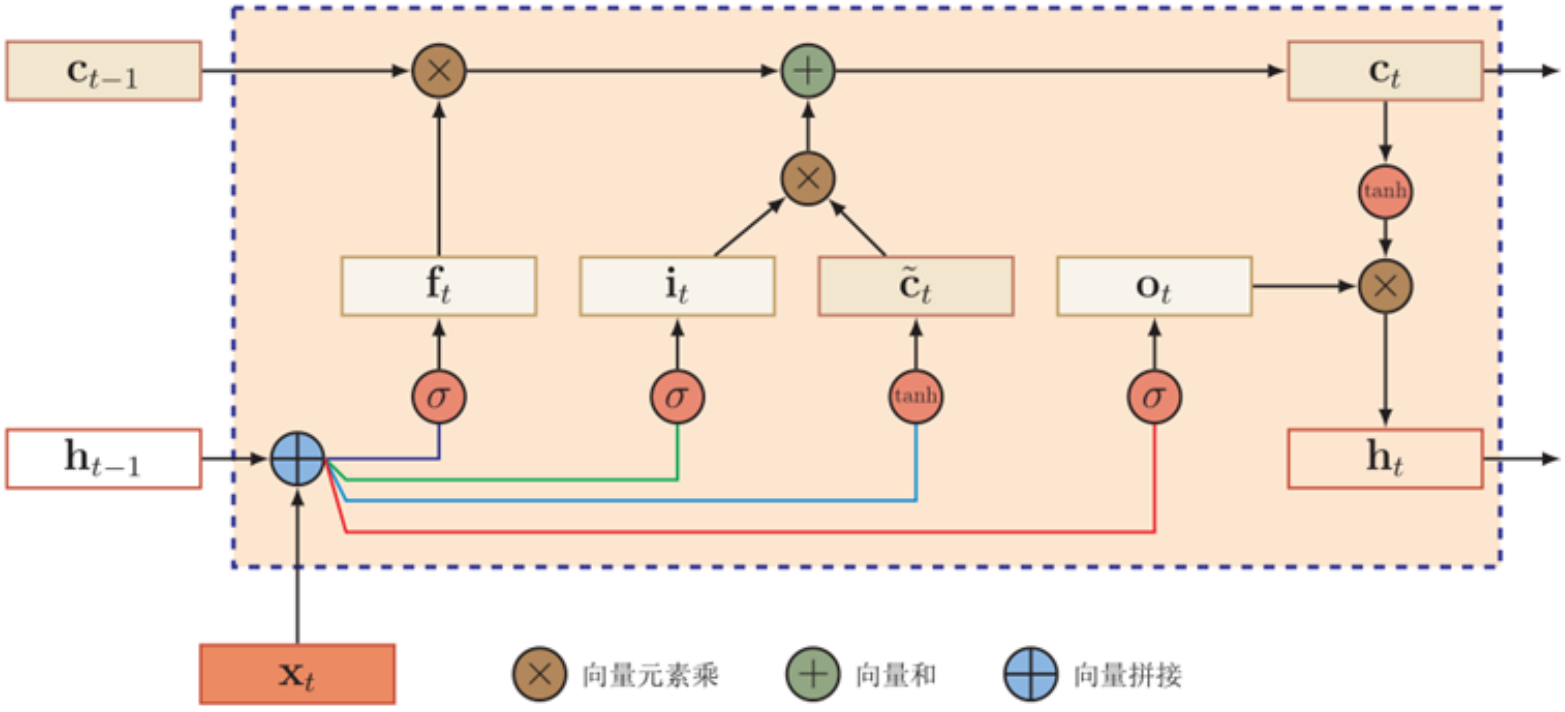
长短时记忆神经网络：LSTM

- 在下式的基础上，LSTM 网络主要改进在以下两方面：

$$h_t = h_{t-1} + g(x_t, h_{t-1}; \theta)$$

  - 新的内部状态（internal state）c_t 专门进行线性的循环信息传递，同时（非线性）输出信息给隐藏层的外部状态 h_t；其中 f_t，i_t 和 o_t 为三个门 (gate) 来控制信息传递的路径；在每个时刻 t，LSTM 网络的内部状态 c_t 记录了到当前时刻为止的历史信息
  - 门机制（gating mechanism）来控制信息传递的路径；取值在 $(0, 1)$ 之间，表示以一定的比例运行信息通过
    - 遗忘门 f_t 控制上一个时刻的内部状态 c_{t-1} 需要遗忘多少信息
    - 输入门 i_t 控制当前时刻的候选状态 $\tilde{c}_t$ 有多少信息需要保存
    - 输出门 o_t 控制当前时刻的内部状态 c_t 有多少信息需要输出给外部状态 h_t

长短时记忆神经网络：LSTM



$$\mathbf{i}_t = \sigma(W_i\mathbf{x}_t + U_i\mathbf{h}_{t-1} + \mathbf{b}_i),$$

$$\mathbf{f}_t = \sigma(W_f\mathbf{x}_t + U_f\mathbf{h}_{t-1} + \mathbf{b}_f),$$

$$\mathbf{o}_t = \sigma(W_o\mathbf{x}_t + U_o\mathbf{h}_{t-1} + \mathbf{b}_o),$$

$$\tilde{\mathbf{c}}_t = \tanh(W_c\mathbf{x}_t + U_c\mathbf{h}_{t-1} + \mathbf{b}_c)$$

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

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

长短时记忆神经网络：LSTM

- 循环神经网络中的隐状态 h 存储了历史信息，可以看作是一种记忆 (memory)
  - 在简单循环网络中，隐状态每个时刻都会被重写，因此可以看作是一种短期记忆 (short-term memory)
  - 长期记忆 (long-term memory) 可以看作是网络参数，隐含了从训练数据中学到的经验，并更新周期要远远慢于短期记忆
  - 在 LSTM 网络中，记忆单元 c 可以在某个时刻捕捉到某个关键信息，并有能力将此关键信息保存一定的时间间隔
  - 记忆单元 c 中保存信息的生命周期要长于短期记忆 h，但又远远短于长期记忆，因此称为长的短期记忆 (long short-term memory)

LSTM 的各种变体

- 没有遗忘门

$$\mathbf{c}_t = \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t.$$

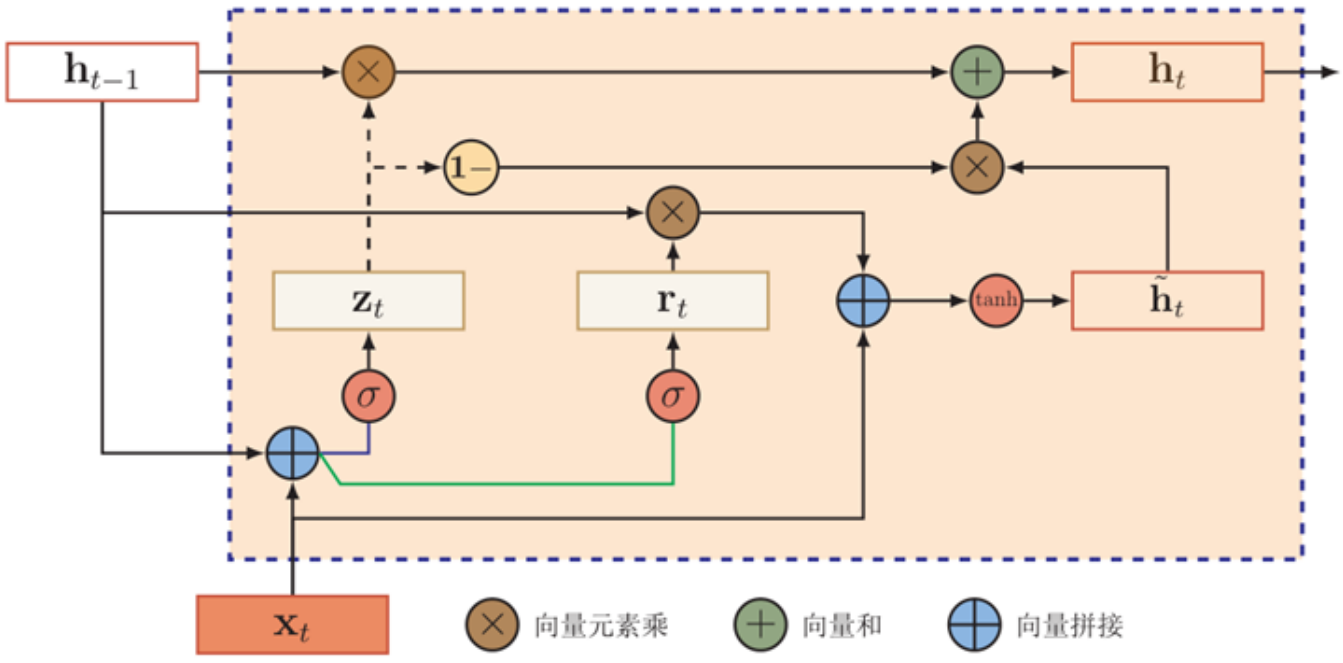
- 耦合输入门和遗忘门

$$\mathbf{f}_t + \mathbf{i}_t = \mathbf{1}.$$

- peephole连接

$$\begin{aligned}
\mathbf{i}_t &= \sigma(W_i \mathbf{x}_t + U_i \mathbf{h}_{t-1} + V_i \mathbf{c}_{t-1} + \mathbf{b}_i), \\
\mathbf{f}_t &= \sigma(W_f \mathbf{x}_t + U_f \mathbf{h}_{t-1} + V_f \mathbf{c}_{t-1} + \mathbf{b}_f), \\
\mathbf{o}_t &= \sigma(W_o \mathbf{x}_t + U_o \mathbf{h}_{t-1} + V_o \mathbf{c}_t + \mathbf{b}_o),
\end{aligned}$$

门控循环单元网络：GRU



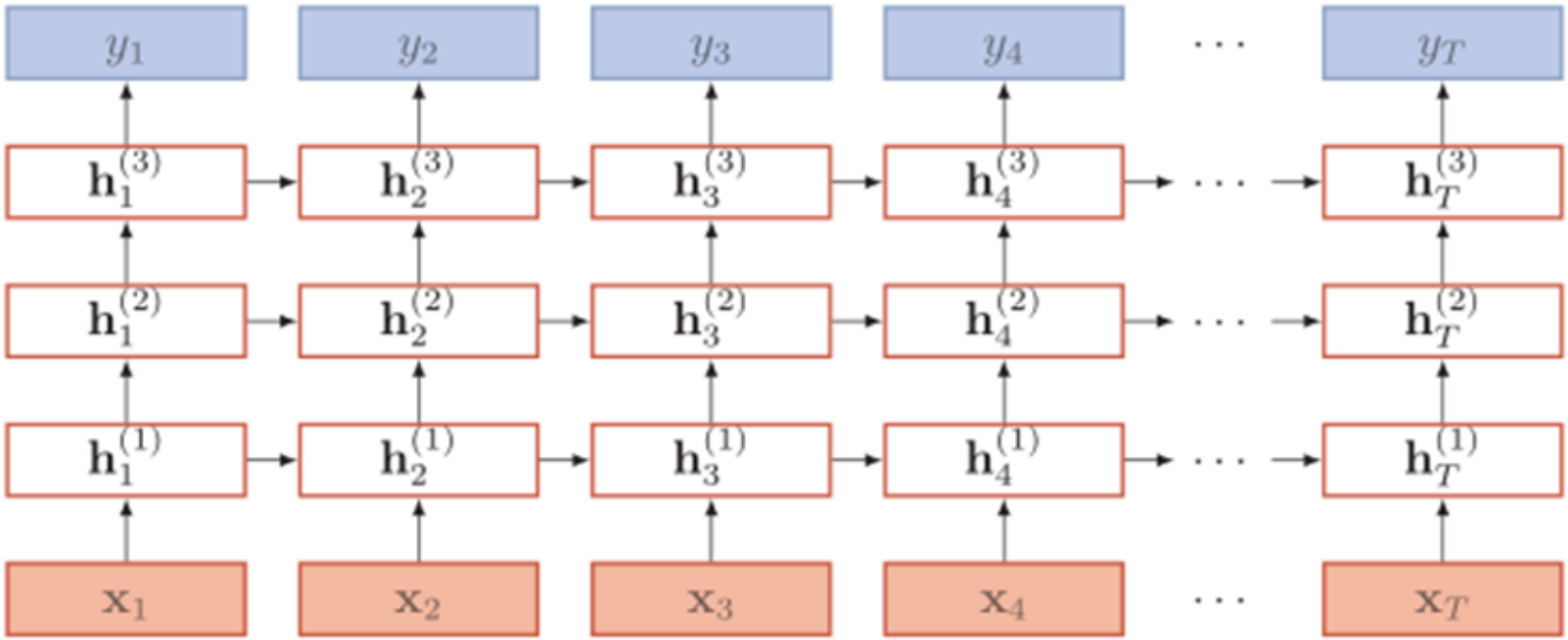
重置门

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

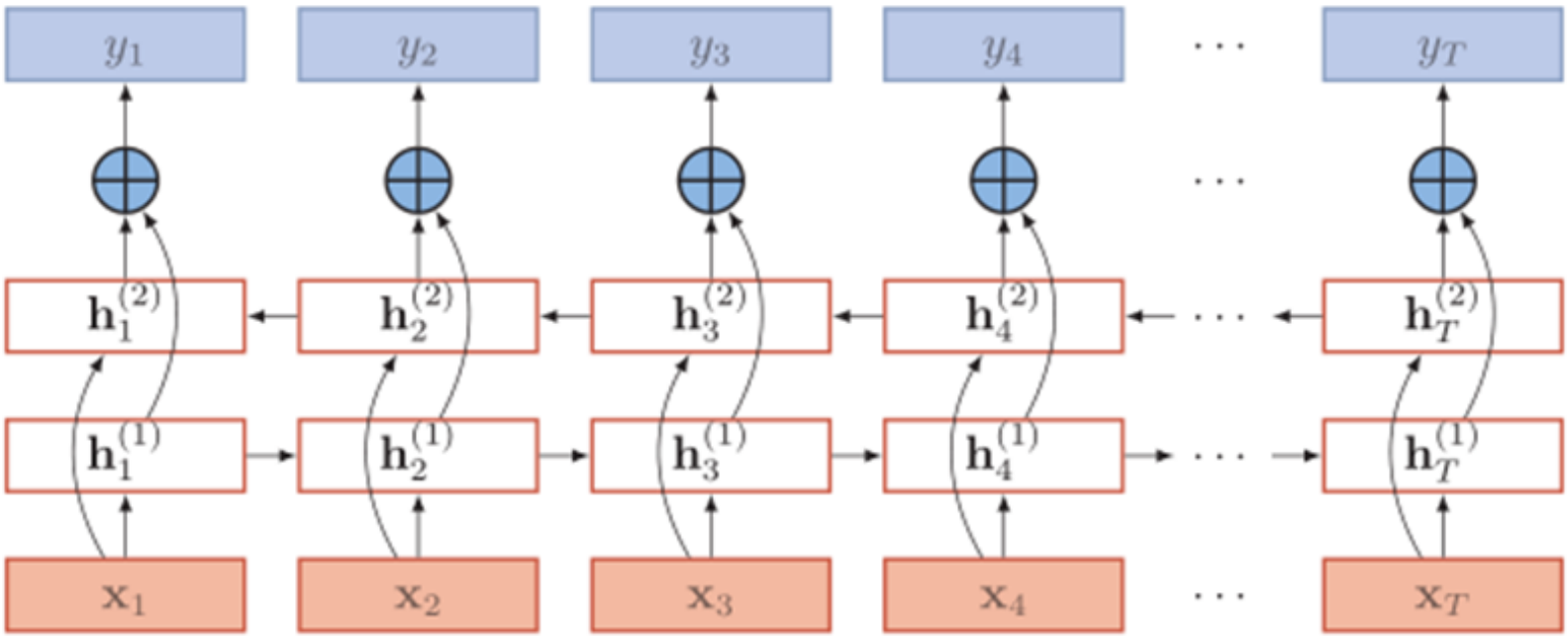
$$\mathbf{z}_t = \sigma(\mathbf{W}_z\mathbf{x}_t + \mathbf{U}_z\mathbf{h}_{t-1} + \mathbf{b}_z), \qquad \mathbf{h}_t = \mathbf{z}_t \odot \mathbf{h}_{t-1} + (1 - \mathbf{z}_t) \odot \tilde{\mathbf{h}}_t,$$

更新门

堆叠循环神经网络


y_1 y_2 y_3 y_4 $\cdots$ y_T
$\mathbf{h}_1^{(3)}$ $\mathbf{h}_2^{(3)}$ $\mathbf{h}_3^{(3)}$ $\mathbf{h}_4^{(3)}$ $\cdots$ $\mathbf{h}_T^{(3)}$
$\mathbf{h}_1^{(2)}$ $\mathbf{h}_2^{(2)}$ $\mathbf{h}_3^{(2)}$ $\mathbf{h}_4^{(2)}$ $\cdots$ $\mathbf{h}_T^{(2)}$
$\mathbf{h}_1^{(1)}$ $\mathbf{h}_2^{(1)}$ $\mathbf{h}_3^{(1)}$ $\mathbf{h}_4^{(1)}$ $\cdots$ $\mathbf{h}_T^{(1)}$
$\mathbf{x}_1$ $\mathbf{x}_2$ $\mathbf{x}_3$ $\mathbf{x}_4$ $\cdots$ $\mathbf{x}_T$

双向循环神经网络


y_1
y_2
y_3
y_4
$\cdots$
y_T
$\mathbf{h}_1^{(2)}$
$\mathbf{h}_2^{(2)}$
$\mathbf{h}_3^{(2)}$
$\mathbf{h}_4^{(2)}$
$\mathbf{h}_T^{(2)}$
$\mathbf{h}_1^{(1)}$
$\mathbf{h}_2^{(1)}$
$\mathbf{h}_3^{(1)}$
$\mathbf{h}_4^{(1)}$
$\mathbf{h}_T^{(1)}$
$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_3$
$\mathbf{x}_4$
$\mathbf{x}_T$

- RNN-based network is not always easy to learn

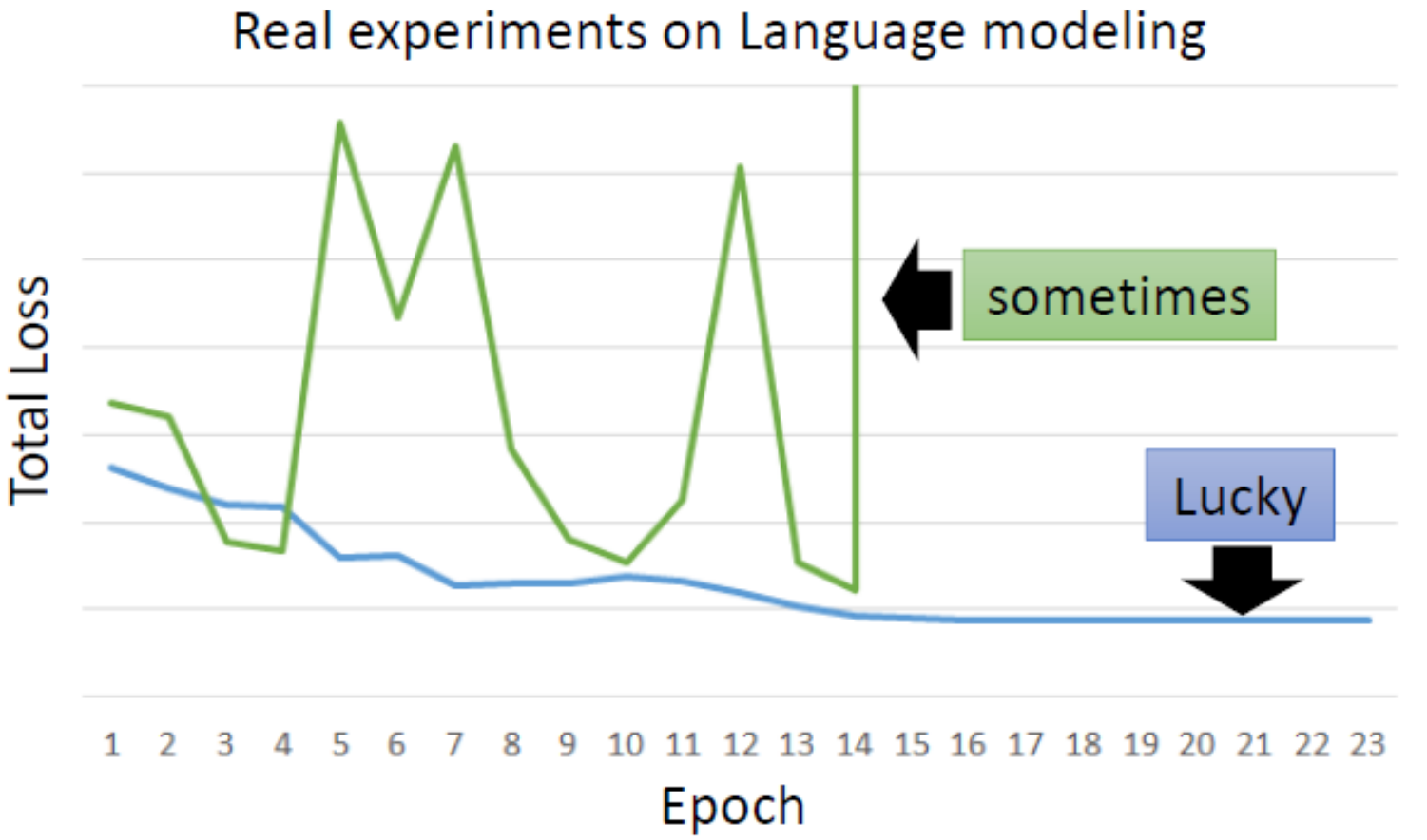
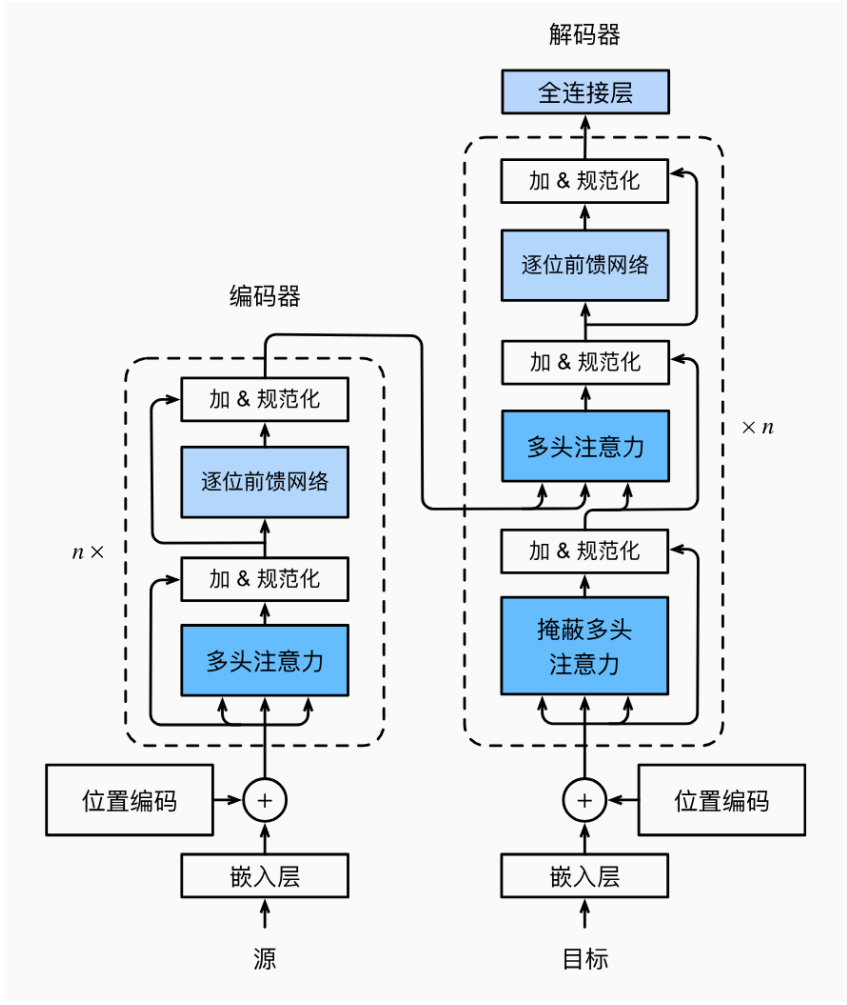


Table of Contents

Transformer

Transformer：继 MLP, CNN, RNN 之外的第 4 大类模型



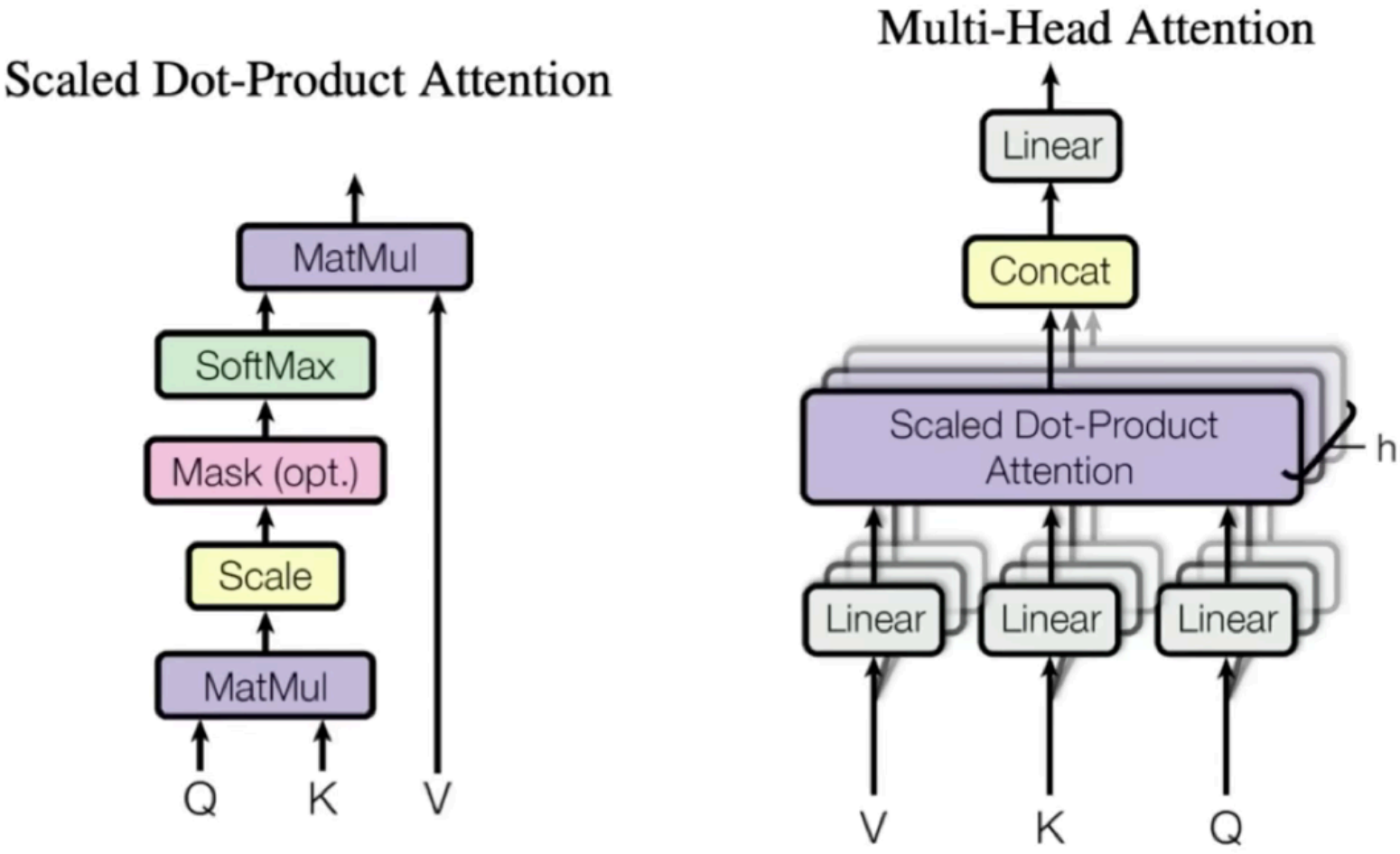
Vaswani, Ashish, et al. "Attention is all you need." Advances in neural information processing systems 30 (2017).

Transformer 编码器

Transformer 的编码器是由多个相同的层叠加而成的，每个层都有两个子层（子层表示为 sublayer）

- 第一个子层是多头自注意力（multi-head self-attention）汇聚
  - 在计算编码器的自注意力时，查询（Q）、键（K）和值（V）都来自前一个编码器层的输出
- 第二个子层是基于位置的前馈网络（positionwise feed-forward network）
- 受残差网络的启发，每个子层都采用了残差连接（residual connection）
  - 在 Transformer 中，对于序列中任何位置的任何输入 $\mathbf{x} \in \mathbb{R}^d$，都要求满足 $\text{sublayer}(\mathbf{x}) \in \mathbb{R}^d$，以便残差连接满足 $\mathbf{x} + \text{sublayer}(\mathbf{x}) \in \mathbb{R}^d$
  - 在残差连接的加法计算之后，紧接着应用层规范化（layer normalization）
- 因此，输入序列对应的每个位置，Transformer 编码器都将输出一个 d 维表示向量

多头注意力


Scaled Dot-Product Attention
MatMul
SoftMax
Mask (opt.)
Scale
MatMul
Q
K
V
Multi-Head Attention
Linear
Concat
Scaled Dot-Product Attention
h
Linear
Linear
Linear
V
K
Q

多头注意力

Scaled Dot-Product Attention:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

n 个 d_k 维度 Query Q，对应 m 个 d_k 维度 Key K，和 m 个 d_v 维度 Value V

Multi-Head Attention:

- Linear 层：将输入投影到比较低的维度，再进 Scaled Dot-Product Attention
- 重复做 h 次，合并在一起，最后再一次线性
- 因为 Scaled Dot-Product Attention 里面没有参数，这里重复 h 次，里面有 linear 投影到低维，这里有一个参数 W，学习 h 次，学习不同的投影方式；使得在投影进入的空间里面，能够匹配不同模式下需要的不同的相似函数，最后再做一次投影
- 公式中看起来有很多小矩阵的乘法，但其实可以一次大矩阵乘法来实现

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

多头注意力用法

Multi-Head Attention 有三种使用方法：

- Encoder 中 Multi-Head Attention
  - 对 Encoder 输入，句子长度为 n，实际输入为 n 个长为 d 的向量
  - 对其中 Multi-Head Attention 输入，是输入直接复制 3 份，作为相同的 Query, Key, Value；所以称之为自注意力机制
  - Multi-Head Attention 的输出让然是 n 个长为 d 的向量
- Decoder 中一开始的 Maksed Multi-Head Attention，与 Encoder 中类似，只是加了 Mask 层，对未来的值加非常大的负数（从而在 softmax 后几乎为 0）
  - 输入是 m 个长为 d 的向量
- Decoder 中之后的 Multi-Head Attention 其中有 Encoder 的输出作为其输入
  - 其中 Key 和 Value 来自 Encoder 的输出 $(n \times d)$
  - Query 来自 Decoder 对应的输出 $(m \times d)$

逐位前馈网络和位置编码

Position-wise Feed-Forward Networks:

- 就是单隐藏层的 MLP：

$$\mathrm{FFN}(\mathbf{x}) = \max(0, \mathbf{x}W_1 + b_1)W_2 + b_2$$

- 由于 multi-head attention 已经把历史关联信息已经获取了，所以之后只用 position-wise 的 MLP 就可以了

Positional Encoding:

- Attention 没有序列位置，因此需要在输入里面加入序列信息

$$PE_{pos,2i} = \sin\left(pos/10000^{2i/d_{model}}\right)$$

$$PE_{pos,2i+1} = \cos\left(pos/10000^{2i/d_{model}}\right)$$

- 这个位置编码在 [-1, 1] 之间变化

Transformer 应用

- RNN 需要串行输入获取模型序列信息，Transformer 通过 Multi-head attention 来获得序列信息，可以并行化
- Attention 对模型假设更少，所以要训练到 RNN 和 CNN 同样的效果，需要更多的数据和更大的模型
- Bert 系列基于 Transformer Encoder，采用 Mask 方式自监督的获得完形填空式的数据，学习双向语言模型
- GPT 系列相当于 Autoregressive Transformer Decoder，each token is predicted and conditioned on the previous token，学习单向语言模型
- Transformer 可应用于 NLP（Bert，GPT-3 等），也可以用在 Vision（ViT 等），多模态（ViLT，CLIP，GPT-4 等），各类任务的基础模型（大模型），决策（决策过程看作 sequential modeling, Decision Transformer）

Table of Contents

Keras

If you want to learn theano:

http://speech.ee.ntu.edu.tw/~tlkagk/courses/MLDS_2015_2/Lecture/Theano%20DNN.ecm.mp4/index.html

http://speech.ee.ntu.edu.tw/~tlkagk/courses/MLDS_2015_2/Lecture/RNN%20training%20(v6).ecm.mp4/index.html

or theano

Very flexible

Need some effort to learn

Interface of TensorFlow or Theano

keras

Easy to learn and use

(still have some flexibility)

You can modify it if you can write TensorFlow or Theano

Keras

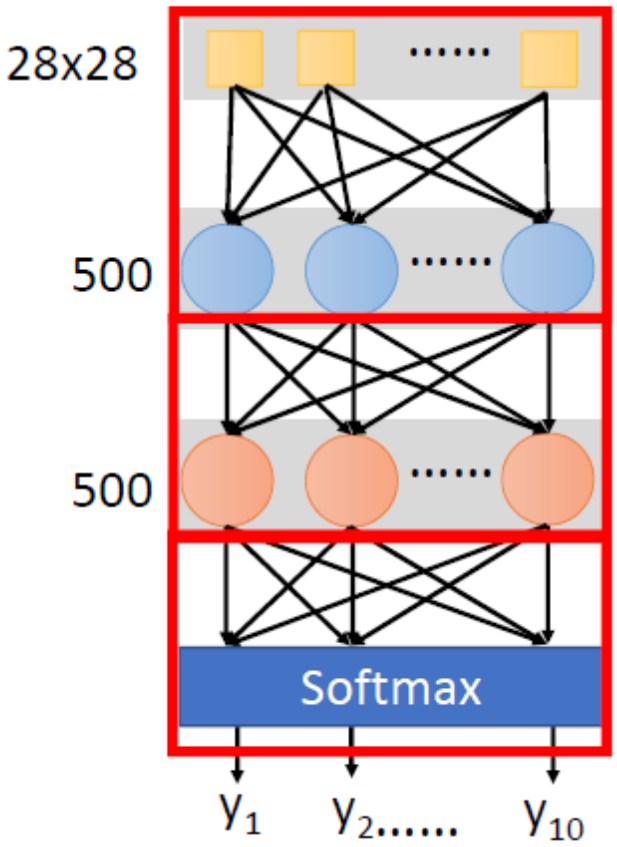
- François Chollet is the author of Keras.
  - He currently works for Google as a deep learning engineer and researcher.
- Keras means *horn* in Greek
- Documentation: http://keras.io/
- Example: https://github.com/fchollet/keras/tree/master/examples

Keras

Step 1: define a set of function

Step 2: goodness of function

Step 3: pick the best function



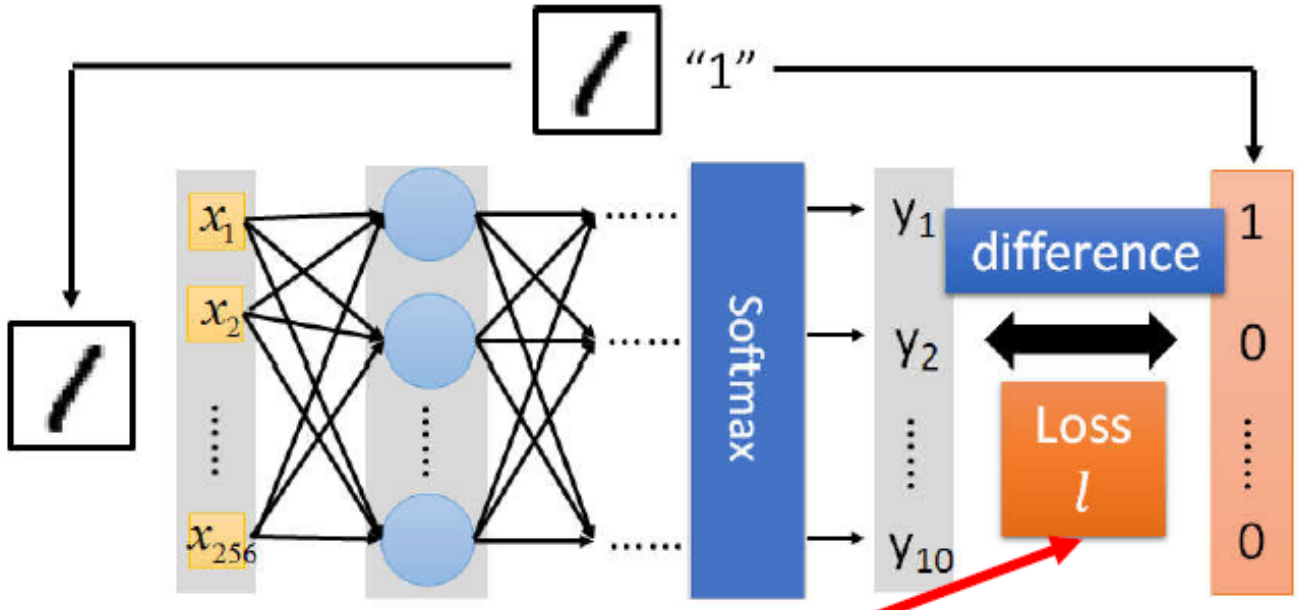

```
model = Sequential()
```

```
model.add( Dense( input_dim=28*28,
                  output_dim=500 ))
model.add( Activation('sigmoid') )
```

```
model.add( Dense( output_dim=500 ) )
model.add( Activation('sigmoid') )
```

```
model.add( Dense(output_dim=10 ) )
model.add( Activation('softmax') )
```

Keras




```
model.compile(loss='mse',
              optimizer=SGD(lr=0.1),
              metrics=['accuracy'])
```

Keras

Step 3.1: Configuration

```
model.compile(loss='mse',
              optimizer=SGD(lr=0.1),
              metrics=['accuracy'])
```

$$w \leftarrow w - \eta \partial L / \partial w$$

0.1

Step 3.2: Find the optimal network parameters

```
model.fit(x_train, y_train, batch_size=100, nb_epoch=20)
```

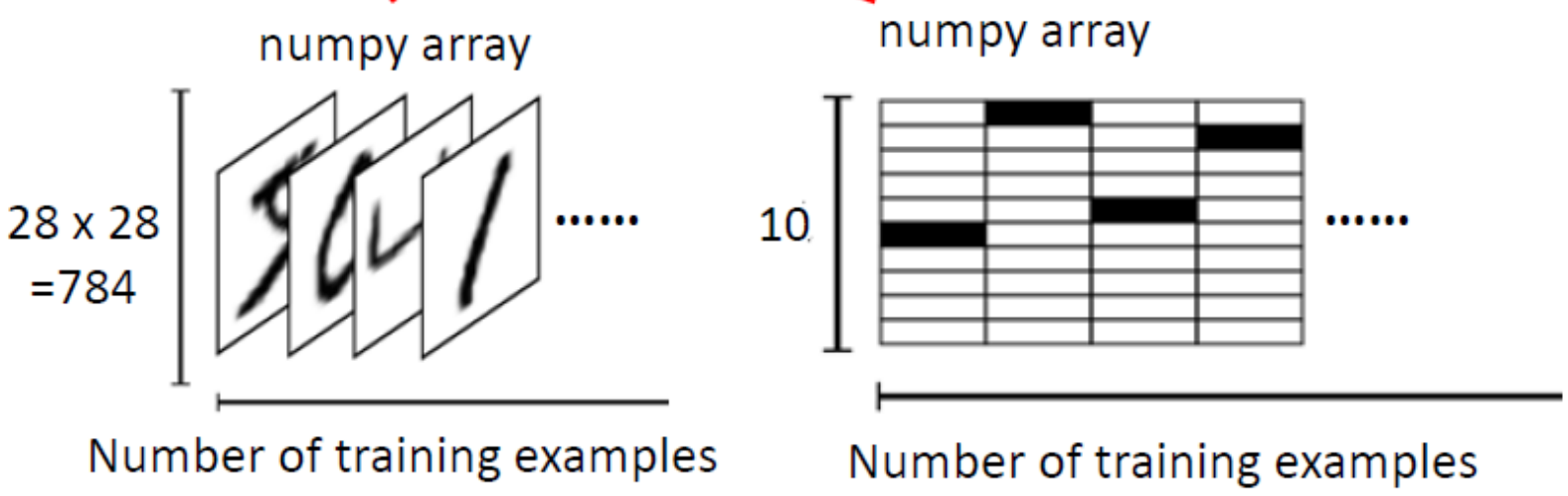
Training data (Images)

Labels (digits)

Keras

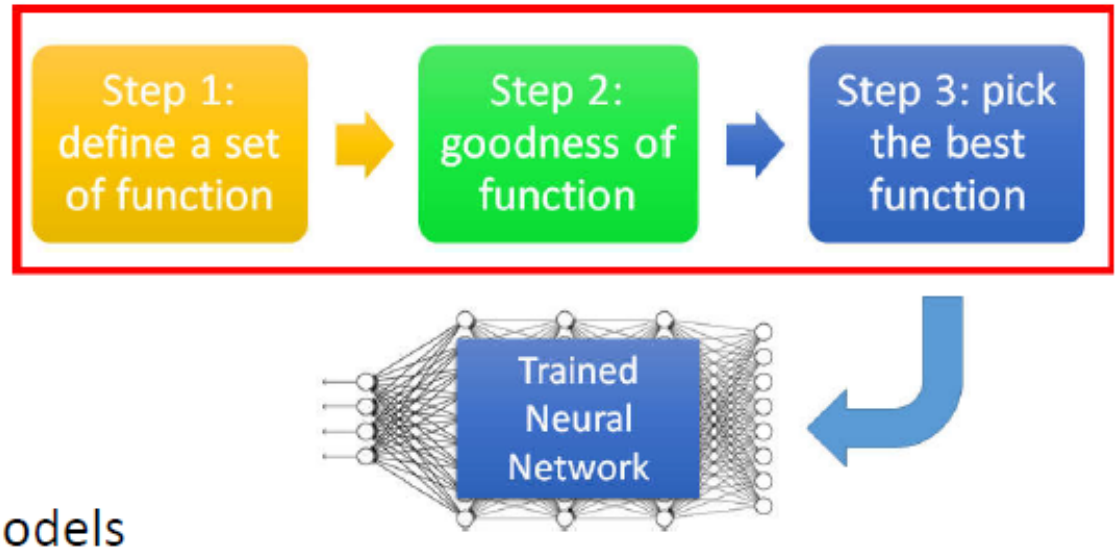
Step 3.2: Find the optimal network parameters

```
model.fit(x_train, y_train, batch_size=100, nb_epoch=20)
```



https://www.tensorflow.org/versions/r0.8/tutorials/mnist/beginners/index.html

Keras



Save and load models

http://keras.io/getting-started/faq/#how-can-i-save-a-keras-model

How to use the neural network (testing):

case 1:

```
score = model.evaluate(x_test,y_test)
print('Total loss on Testing Set:', score[0])
print('Accuracy of Testing Set:', score[1])
```

case 2:

```
result = model.predict(x_test)
```

Keras

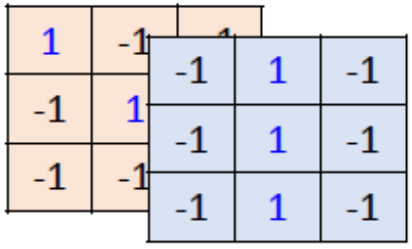
- Using GPU to speed training
  - Way 1
    - THEANO_FLAGS=device=gpu0 python YourCode.py
  - Way 2 (in your code)
    - import os
    - os.environ["THEANO_FLAGS"] = "device=gpu0"

Table of Contents

CNN in Keras

Only modified the ***network structure*** and ***input format (vector -> 3-D tensor)***

```
model2.add( Convolution2D( 25,3,3,
            input_shape=(1,28,28) ) )
```



.....

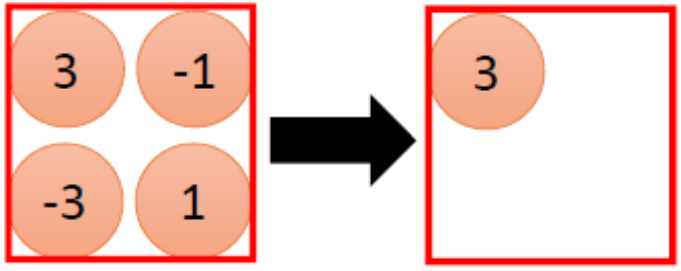
There are 25 3x3 filters.

Input_shape = (1 , 28 , 28)

1: black/weight, 3: RGB

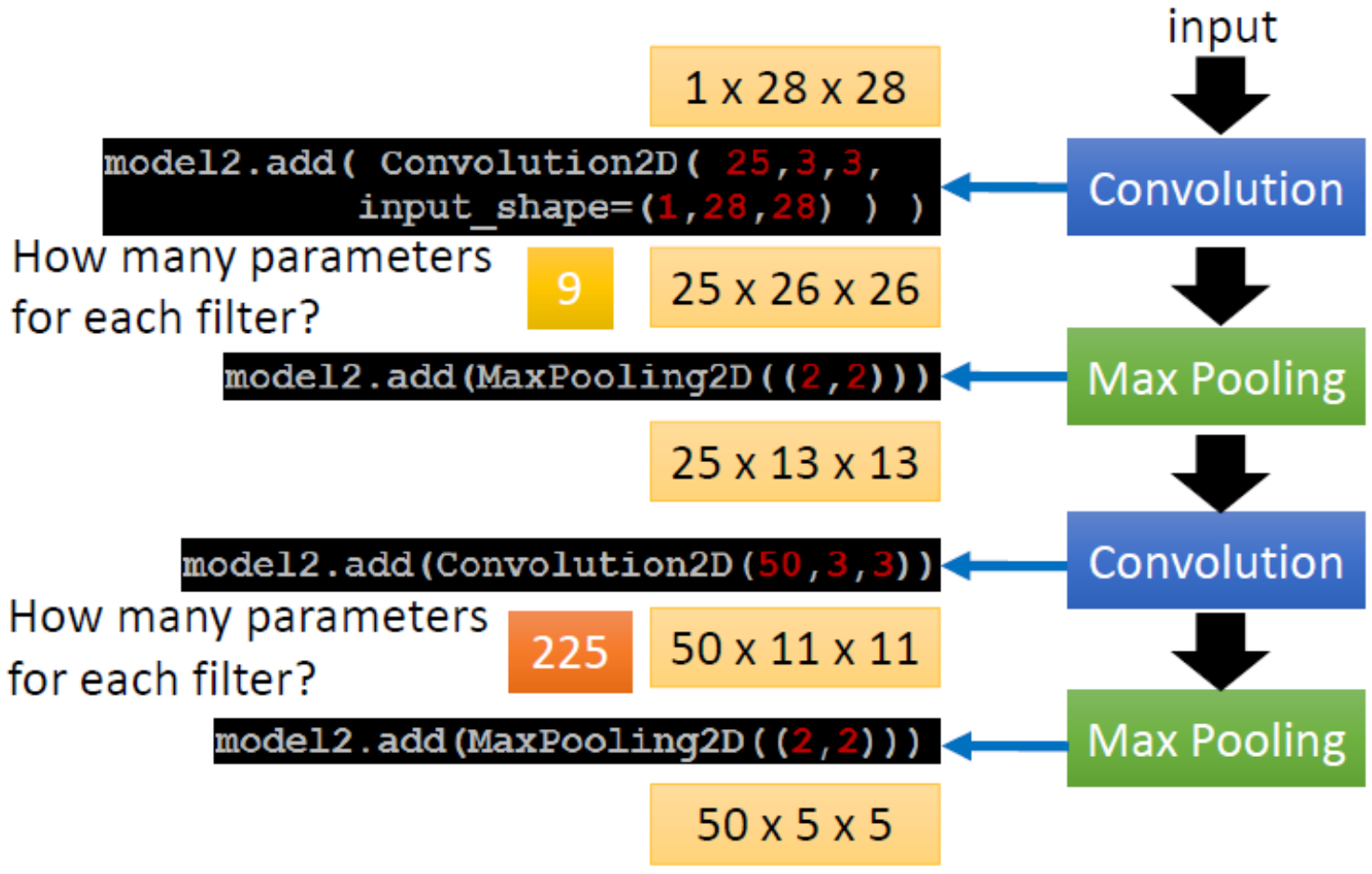
28 x 28 pixels

```
model2.add(MaxPooling2D((2,2)))
```



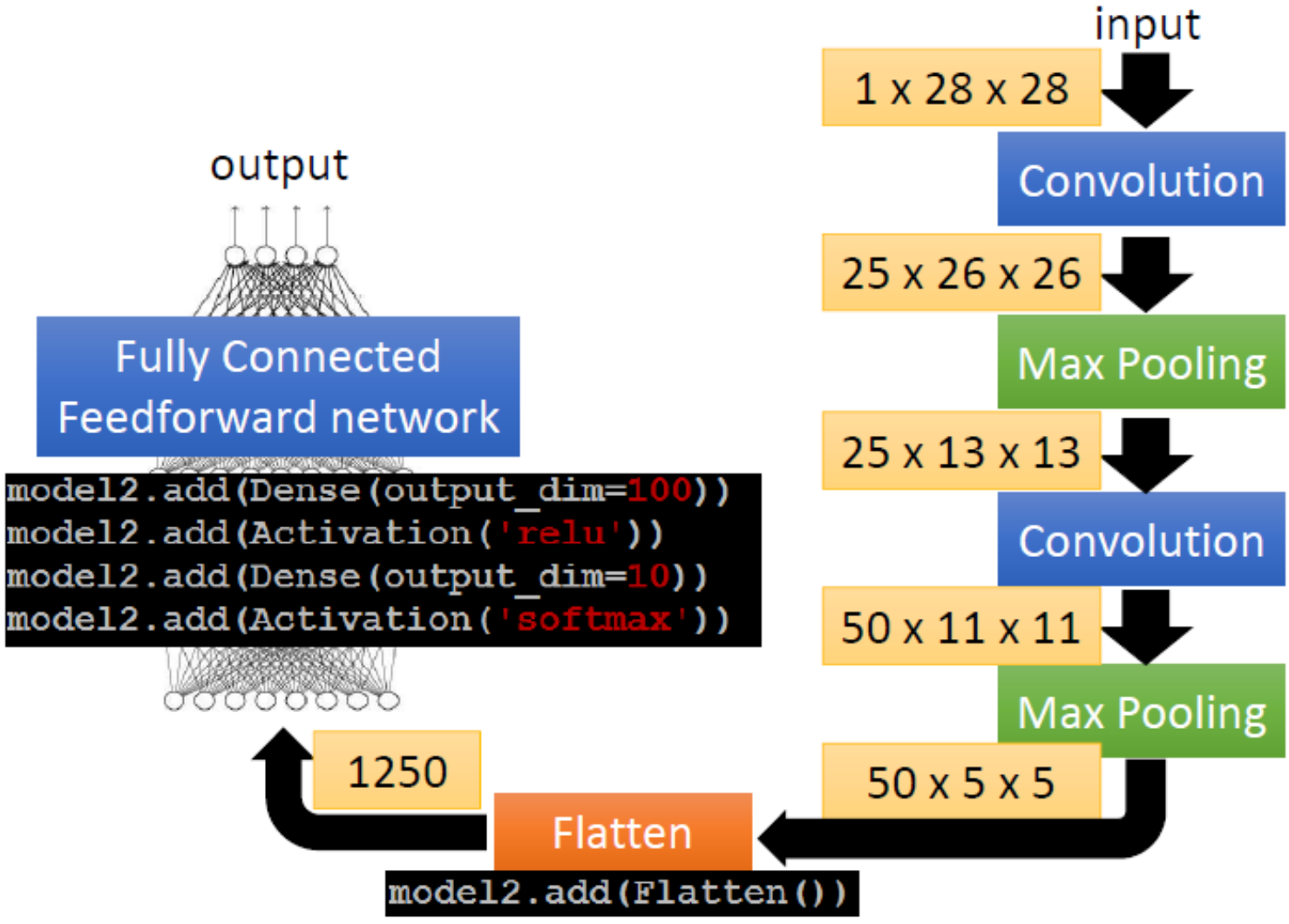
CNN in Keras

Only modified the ***network structure*** and ***input format (vector -> 3-D tensor)***



CNN in Keras

Only modified the ***network structure*** and ***input format (vector -> 3-D tensor)***



实现 LeNet

- 卷积网络发展的特别迅速，最早是由 Lecun 提出来的，LeNet 成为 cnn 的鼻祖
- 接下来他的学生 Alex 提出了层数更深的 Alexnet，然后 2013 年又提出了 VGGnet，有 16 层和 19 层两种，这些都只是在层数上面的加深，并没有什么其他的创新
- 之后 google 提出了 inception net 在网络结构上实现了创新，提出了一种 inception 的机构
- Facebook AI 实验室又提出了 resnet，残差网络，实现了 150 层的网络结构可训练化
- 下面我们就来实现一下最简单的 LeNet，使用 mnist 手写子体作为训练集

LeNet

- 导入必要的库和数据集

```
import keras
from keras.datasets import mnist
(x_train, y_train), (x_test,y_test) =mnist.load_data()
```

- 处理数据，让数据的 shape 是 (28, 28, 1)，然后 label 做一个 one-hot encoding 处理，比如类别是 3，那么变成 [0, 0, 1 ,0, 0, 0, 0, 0, 0, 0]

```
x_train=x_train.reshape(-1,28,28,1)
x_test=x_test.reshape(-1,28,28,1)
x_train=x_train/255.
x_test=x_test/255.
y_train=keras.utils.to_categorical(y_train)
y_test=keras.utils.to_categorical(y_test)
```

LeNet

构建 LeNet

```
from keras.layers import Conv2D,MaxPool2D,Dense,Flatten
from keras.models import Sequential
lenet=Sequential()
lenet.add(Conv2D(filters=6, kernel_size=(5,5), padding='vali
            input_shape=(1,28,28), activation='relu'))
lenet.add(MaxPool2D(pool_size=2,strides=2))
lenet.add(Conv2D(16,kernel_size=(5,5),strides=1,padding='val
lenet.add(MaxPool2D(pool_size=2,strides=2))
lenet.add(Flatten())
lenet.add(Dense(120))
lenet.add(Dense(84))
lenet.add(Dense(10,activation='softmax'))
lenet.compile('sgd',loss='categorical_crossentropy',
              metrics=['accuracy'])
lenet.fit(x_train,y_train,batch_size=64,epochs=50,
          validation_data=[x_test,y_test])
```

LeNet

训练 50 次，得到结果如下

```
60000/60000 [==============================] - 3s - loss: 0.0231 - acc: 0.9925 - val_loss: 0.0491 - val_acc: 0.9855
Epoch 43/50
60000/60000 [==============================] - 3s - loss: 0.0233 - acc: 0.9926 - val_loss: 0.0478 - val_acc: 0.9870
Epoch 44/50
60000/60000 [==============================] - 3s - loss: 0.0226 - acc: 0.9931 - val_loss: 0.0522 - val_acc: 0.9850
Epoch 45/50
60000/60000 [==============================] - 3s - loss: 0.0218 - acc: 0.9930 - val_loss: 0.0502 - val_acc: 0.9854
Epoch 46/50
60000/60000 [==============================] - 3s - loss: 0.0216 - acc: 0.9934 - val_loss: 0.0491 - val_acc: 0.9848
Epoch 47/50
60000/60000 [==============================] - 3s - loss: 0.0209 - acc: 0.9935 - val_loss: 0.0501 - val_acc: 0.9855
Epoch 48/50
60000/60000 [==============================] - 3s - loss: 0.0206 - acc: 0.9937 - val_loss: 0.0493 - val_acc: 0.9858
Epoch 49/50
60000/60000 [==============================] - 3s - loss: 0.0200 - acc: 0.9940 - val_loss: 0.0493 - val_acc: 0.9858
Epoch 50/50
60000/60000 [==============================] - 3s - loss: 0.0202 - acc: 0.9939 - val_loss: 0.0509 - val_acc: 0.9852
```

训练 50 次得到的训练准确率已经达到 0.9939，测试准确率达到 0.9852

Table of Contents

RNN 文本生成

- RNN 在 NLP 领域有着广泛的应用，其中一个应用就是构建语言模型
- 语言模型让我们可以在给定前文的情况下预测下一个词的可能性
- 我们基于语言模型，在给定前十个字符的情况下去预测下一个字符

导入库和数据

```
from keras.layers import Dense,Activation
from keras.layers.recurrent import SimpleRNN
from keras.models import Sequential
from keras.utils.vis_utils import plot_model
import numpy as np
####load data
fin=open('./11-0.txt','rb')
lines=[]
for line in fin:
    line=line.strip().lower()
    line=line.decode('ascii','ignore')
    if len(line)==0:
        continue
    lines.append(line)
fin.close()
text=' '.join(lines)
```

处理数据

```
####creat the lookup tables
chars=set([c for c in text])
nb_chars=len(chars)
char2index=dict((c,i) for i,c in enumerate(chars))
index2char=dict((i,c) for i,c in enumerate(chars))
####create the input and label texts
SEQLEN=10
STEP=1
input_chars=[]
label_chars=[]
for i in range(0,len(text)-SEQLEN,STEP):
    input_chars.append(text[i:i+SEQLEN])
    label_chars.append(text[i+SEQLEN])
####vectorize teh input data and label texts
X=np.zeros((len(input_chars),SEQLEN,nb_chars),dtype=np.bool)
y=np.zeros((len(input_chars),nb_chars),dtype=np.bool)
for i,input_char in enumerate(input_chars):
    for j,ch in enumerate(input_char):
        X[i,j,char2index[ch]]=1
    y[i,char2index[label_chars[i]]]=1
```

Simple RNN 模型定义

```
####define the model
HIDDEN_SIZE=128
BATCH_SIZE=128
NUM_ITERATIONS=25
NUM_EPOCHES_PER_ITERATION=1
NUM_PERDS_PER_EPOCH=100
model=Sequential()
model.add(SimpleRNN(HIDDEN_SIZE,return_sequences=False,
                    input_shape=(SEQLEN,nb_chars),
                    unroll=True))
model.add(Dense(nb_chars,activation='softmax'))
model.compile(loss='categorical_crossentropy',
              optimizer='rmsprop')
```

LSTM RNN 模型定义

```
####define the model
hidden_size = 500
model = Sequential()
model.add(Embedding(vocabulary, hidden_size,
                    input_length=num_steps))
model.add(LSTM(hidden_size, return_sequences=True))
model.add(LSTM(hidden_size, return_sequences=True))
model.add(Dropout(0.5))
model.add(TimeDistributed(Dense(vocabulary)))
model.add(Activation('softmax'))
optimizer = Adam()
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['categorical_accuracy'])
```