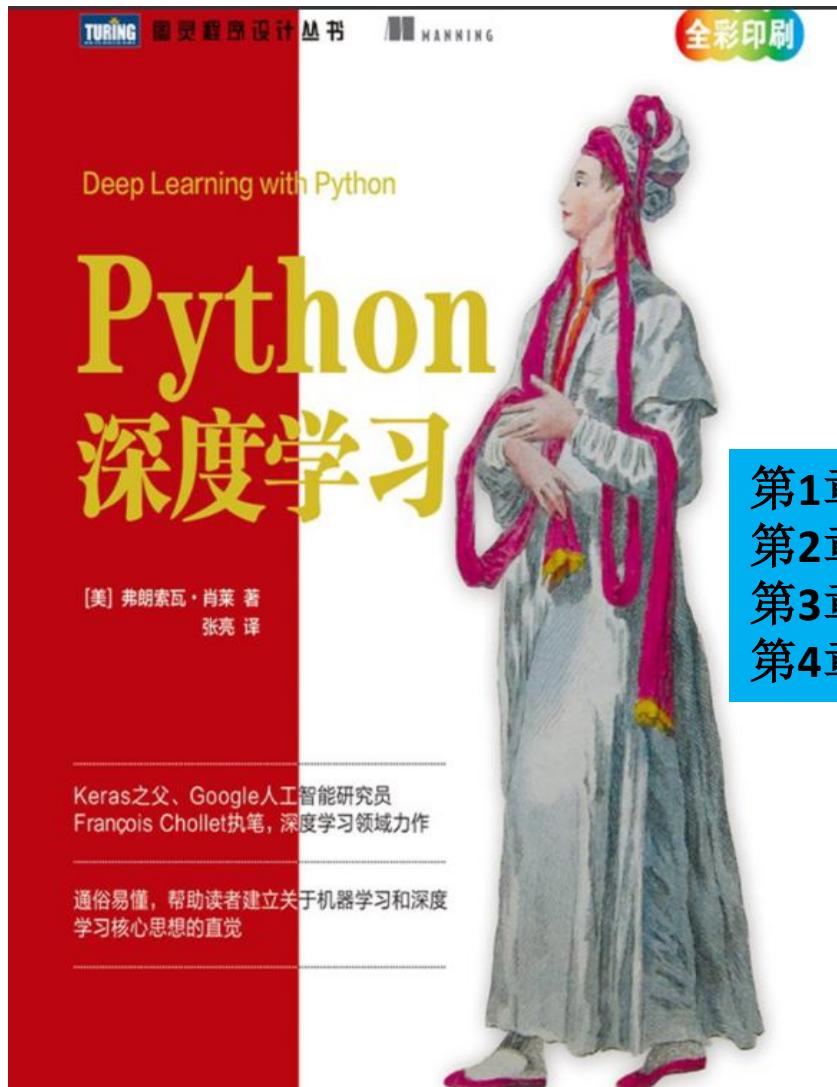


深度学习基础

参考资料



- 第1章 什么是深度学习
- 第2章 神经网络的数学基础
- 第3章 神经网络入门
- 第4章 机器学习基础

教材代码：<https://github.com/fchollet/deep-learning-with-python-notebooks>

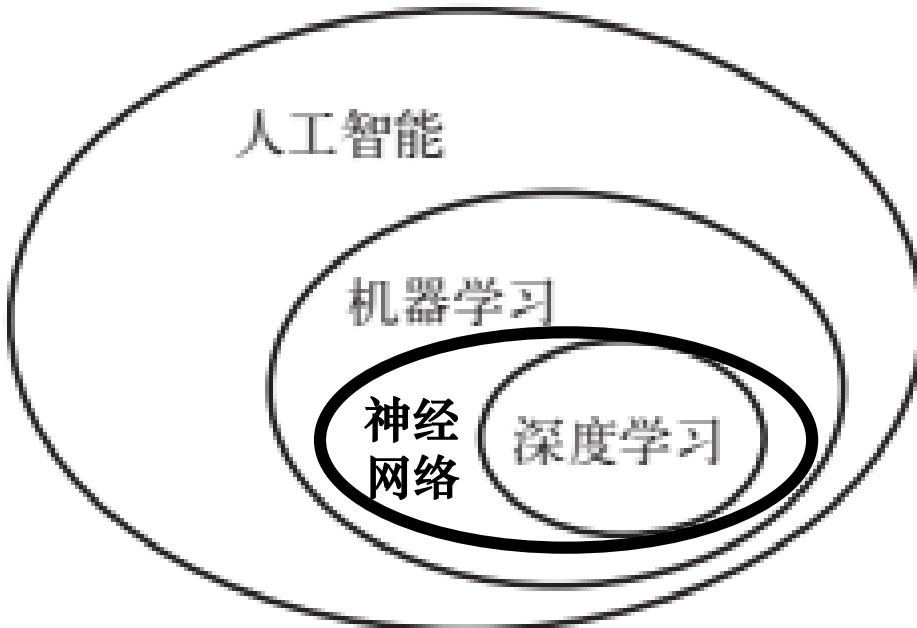
本章内容

- 1 什么是深度学习
- 2 神经网络的数学基础
- 3 神经网络入门
- 4 神经网络的通用工作流程

1 什么是深度学习？

- 基本概念的定义
- 机器学习发展的时间线
- 深度学习日益流行的关键因素及其未来潜力

人工智能、机器学习与深度学习



人工智能是一个综合性的领域。

- 符号主义：用符号逻辑表达一些东西，目前并不是十分流行。
- 行为主义：行为能力不断增强，并没有很好的实现。
- 连接主义：最主要的智能技术，比如人工神经网络。

图 1-1 人工智能、机器学习与深度学习

机器学习

1. 机器学习：通过观察数据自动学会数据处理规则。

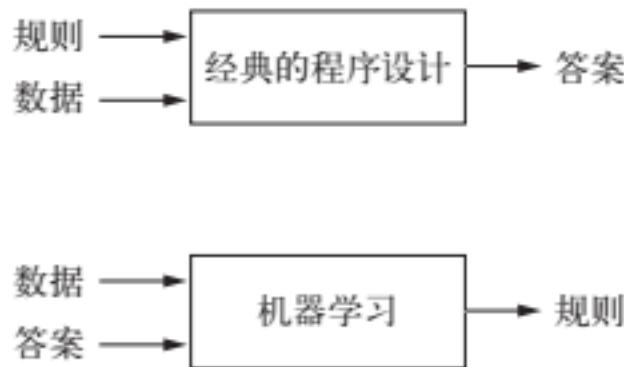


图 1-2 机器学习：一种新的编程范式

2. 机器学习中的**学习**指的是，寻找更好< b>数据表示的**自动搜索**过程。
3. 计算机程序利用**经验E**学习**任务T**，性能是**P**，如果针对**任务T**的**性能P**随着**经验E**不断增长，则称为机器学习。——汤姆·米切尔，1997

神经网络

- 从连续的层中进行学习
- 深度：模型中的层数
- 结构特点：逐层堆叠

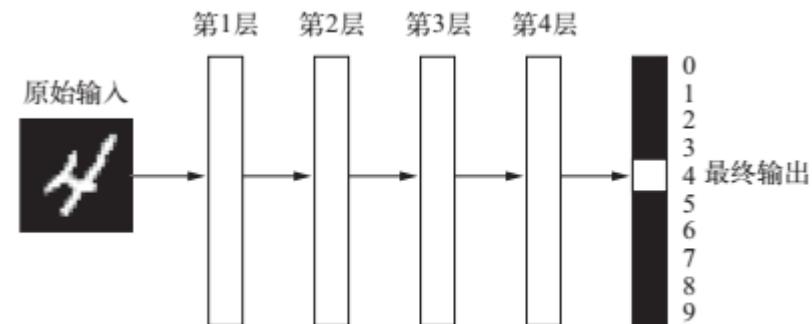
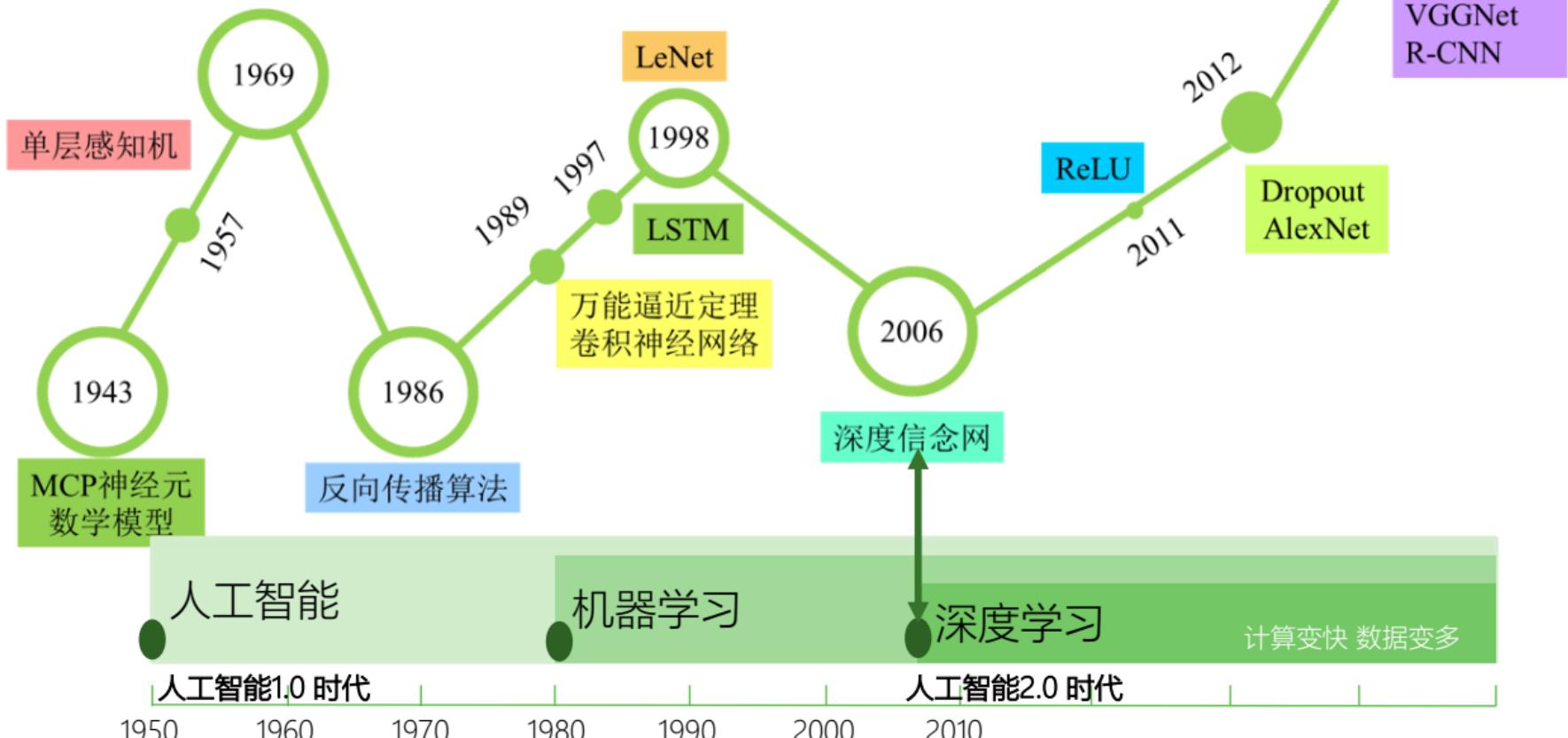


图 1-5 用于数字分类的深度神经网络

- 神经网络 **VS.** 机器学习
 - 神经网络：将特征工程完全自动化
 - 第一，通过渐进的、逐层的方式形成越来越复杂的表示；
 - 第二，对中间这些渐进的表示共同进行学习
 - 机器学习：
 - 也被称为浅层学习（shallow learning），往往是仅仅学习一两层的数据表示。
 - 需手动为数据设计好的表示层。

深度学习里程碑

Minsky和Seymour Papert专著Perceptron
：单层感知机不能解决XOR问题

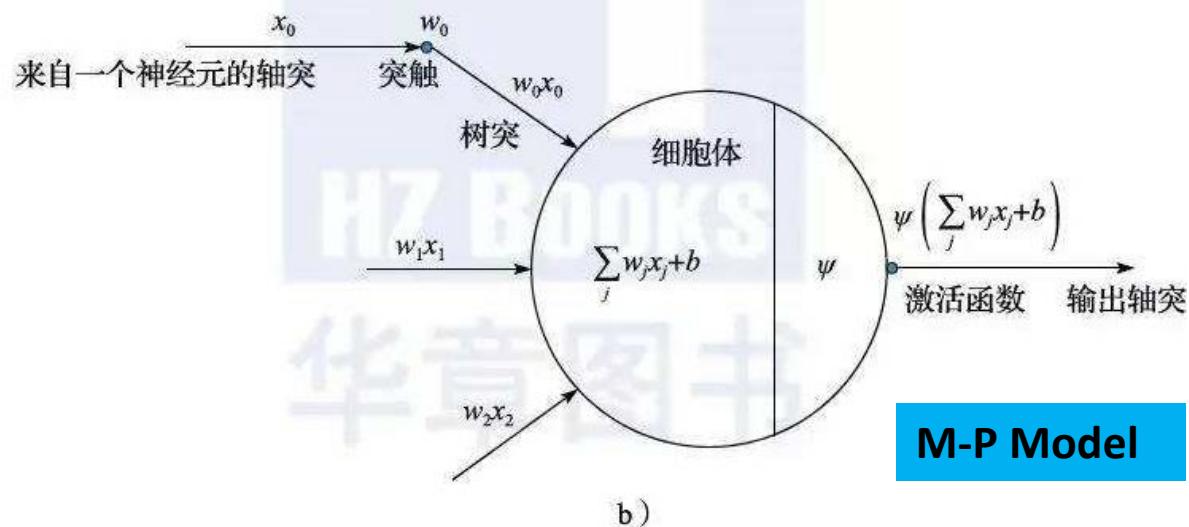
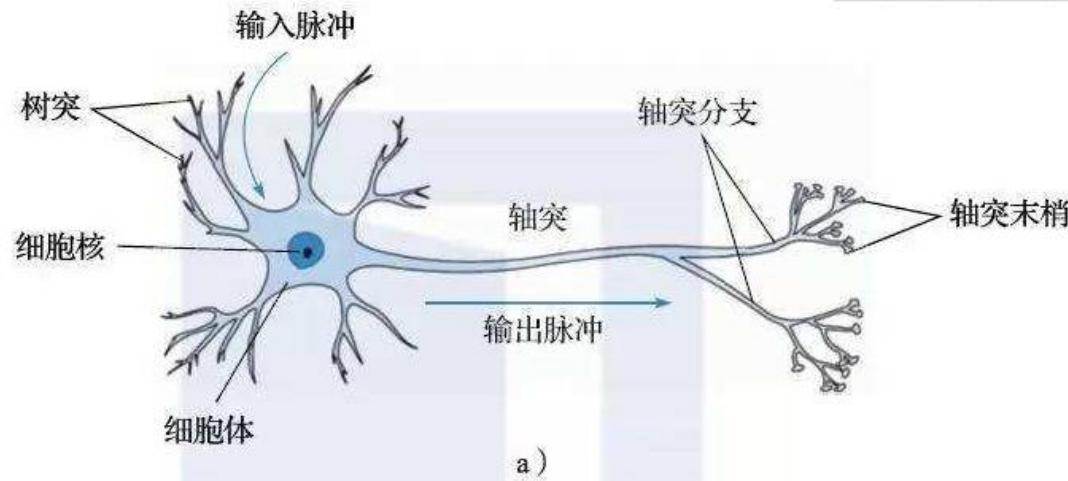


引自 Prof. Meina Kan, PHD Student Xin Liu and Shuzhe Wu 5

- 1943: M-P model (McCulloch-Pitts Model)
- 1958: Perceptron (linear model)
- 1969: Perceptron has limitation
- 1980s: Multi-layer perceptron
 - Do not have significant difference from DNN today
- 1986: Backpropagation
 - Usually more than 3 hidden layers is not helpful
- 1989: 1 hidden layer is “good enough”, why deep?
- 2006: “Deep Learning”; RBM(Restricted Boltzmann machine) initialization
- 2009: GPU
- 2011: Start to be popular in speech recognition
- 2012: win ILSVRC image competition
- 2015.2: Image recognition surpassing human-level performance
- 2016.3: Alpha GO beats Lee Sedol
- 2016.10: Speech recognition system as good as humans

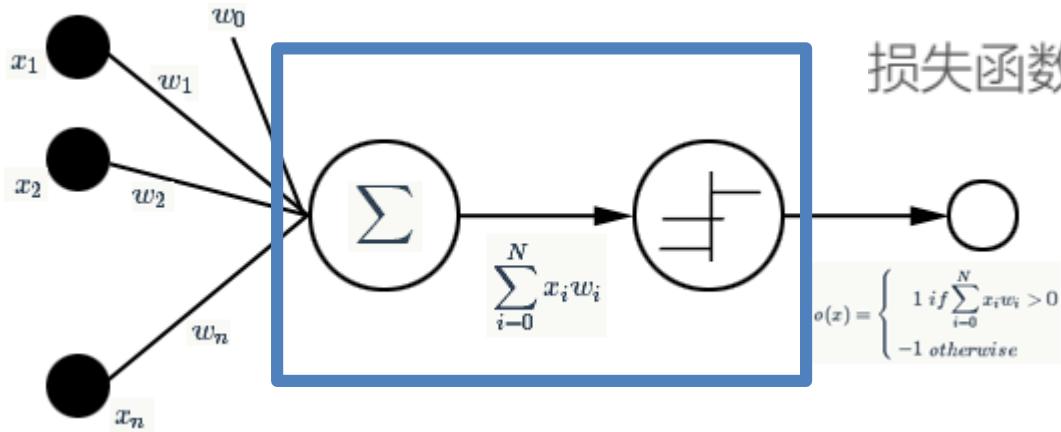
浅层神经网络 -> 深层神经网络
(深度学习)

人工神经元 VS. 生物神经元



感知机 (Perceptron) 模型

- 目标：找到一个 (\mathbf{w}, b) ，将线性可分的数据集中的所有样本点都正确地分为两类。
- 问题设定：寻找 (\mathbf{w}, b) ，使得损失函数极小化的最优问题。 $f(\mathbf{x}) = \text{sign}(\mathbf{w}^T \mathbf{x} + b)$



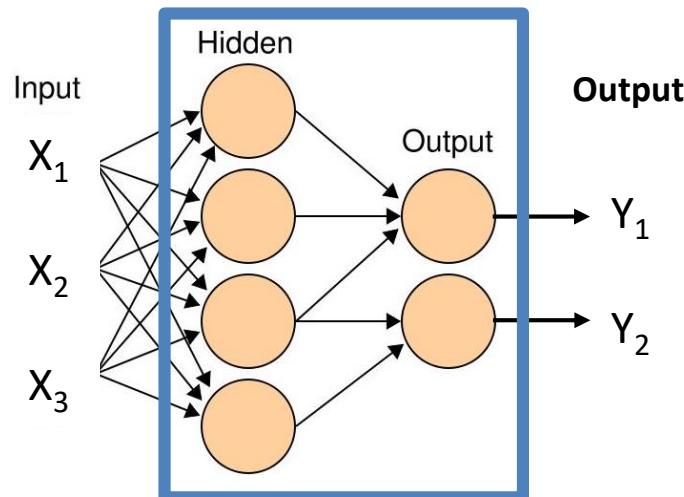
$$\text{sign}(x) = \begin{cases} +1 & x \geq 0 \\ -1 & x < 0 \end{cases}$$

损失函数 $L(\mathbf{w}, b) = - \sum_{x_j \in M} y_j (\mathbf{w}^T \mathbf{x}_j + b)$

不能解决异或问题！

多层感知机 (Multi-layer perceptron)

- 是一种全连接的两层神经网络
- 相较于感知机，有三点改进：
 1. 加入了隐藏层，增强模型的表达能力，但同时也增加了模型的复杂度
 2. 输出层的神经元也可以不止一个输出；
 3. 对激活函数做扩展，增强模型的表达能力。
 - 感知机的激活函数是 $\text{sign}(z)$ ，虽然简单但是处理能力有限。

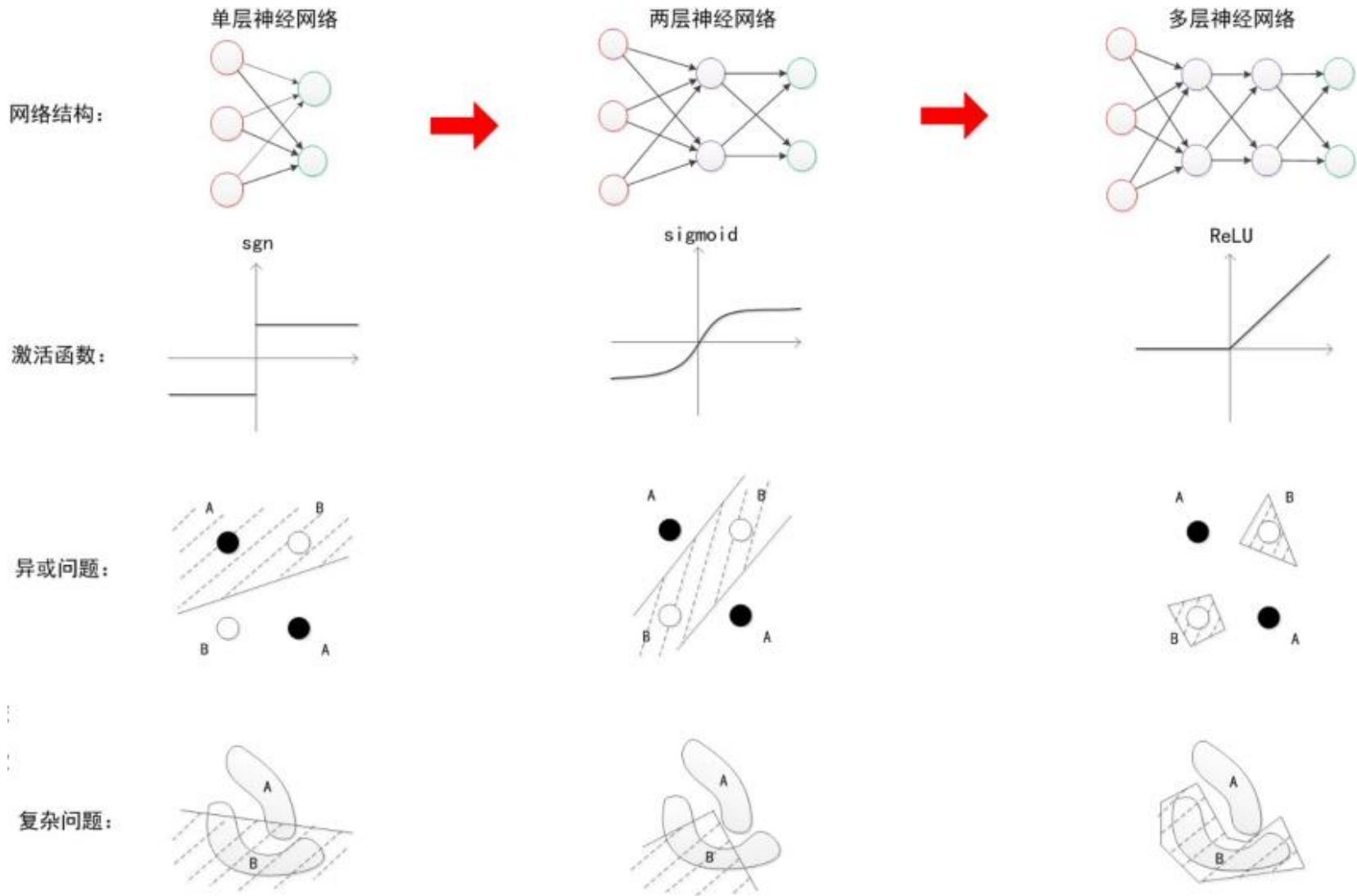


总结：浅层神经网络

- 优势：
 - 需要的数据量小
 - 训练快
- 局限性：对复杂问题的表示能力有限，泛化性弱。
- Why not go deeper?
 - Kurt Hornik证明：理论上两层神经网络足以拟合任何函数。
 - 数据量和计算能力的约束

多层神经网络（深度学习）

- 深度学习：最早是由多伦多大学的G. E.Hinton等于2006年在Science上发表的论文“Reducing the dimensionality of data with neural networks”中提出。
- 深度学习三位开创者：Hinton, LeCun, Bengio
 - Nature杂志为纪念AI 60周年推出的综述文献：《Deep Learning》



随着网络层数的增加，以及激活函数的调整，
神经网络对非线性问题的拟合能力越来越强。

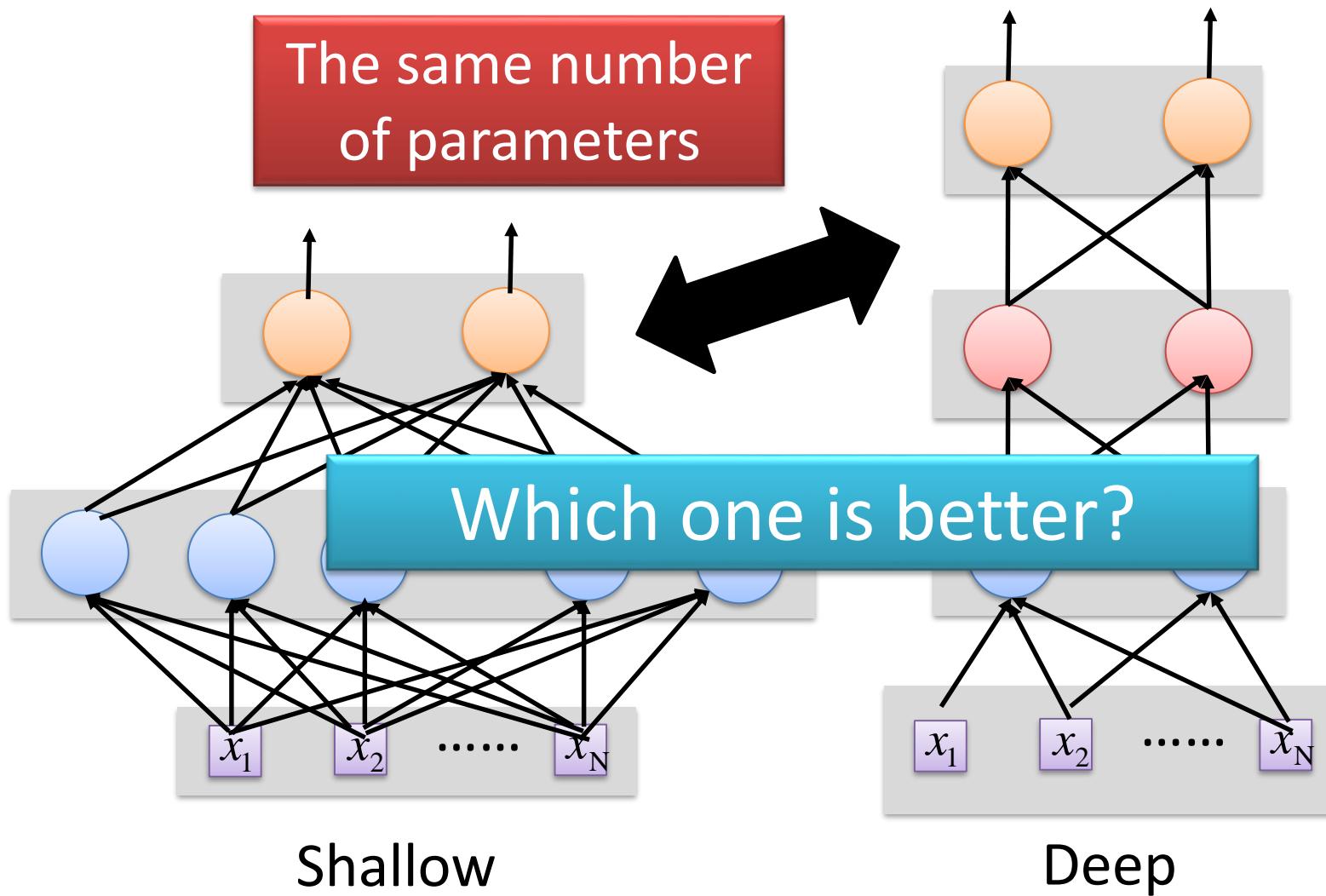
Deeper is Better?

Layer X Size	Word Error Rate (%)
1 X 2k	24.2
2 X 2k	20.4
3 X 2k	18.4
4 X 2k	17.8
5 X 2k	17.2
7 X 2k	17.1

Not surprised, more parameters, better performance

Seide, Frank, Gang Li, and Dong Yu. "Conversational Speech Transcription Using Context-Dependent Deep Neural Networks." *Interspeech*. 2011.

Fat + Short v.s. Thin + Tall



Fat + Short v.s. Thin + Tall

Layer X Size	Word Error Rate (%)	Layer X Size	Word Error Rate (%)
1 X 2k	24.2		
2 X 2k	20.4		
3 X 2k	18.4		
4 X 2k	17.8		
5 X 2k	17.2	1 X 3772	22.5
7 X 2k	17.1	1 X 4634	22.6
		1 X 16k	22.1

Why?

Seide, Frank, Gang Li, and Dong Yu. "Conversational Speech Transcription Using Context-Dependent Deep Neural Networks." *Interspeech*. 2011.

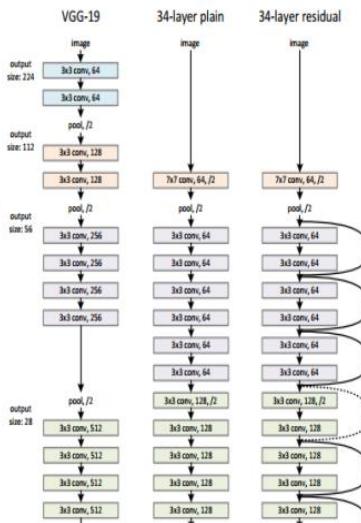
What Makes Deep Learning Succeed Now?

Big data:



- Massive labeled datasets

Algorithm: Improved models



- Easy programming
 - Fast model evolution
 - Fast training and inferencing

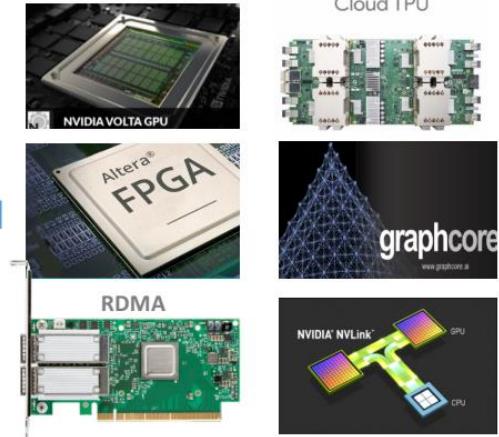
Software frameworks

Microsoft
CNTK

dmlc
mxnet

PYTORCH

Computing:



- Massive computing power
 - Fast communication

Deep Learning Approach

训练过程：正向计算 & 反向传播（backpropagation）

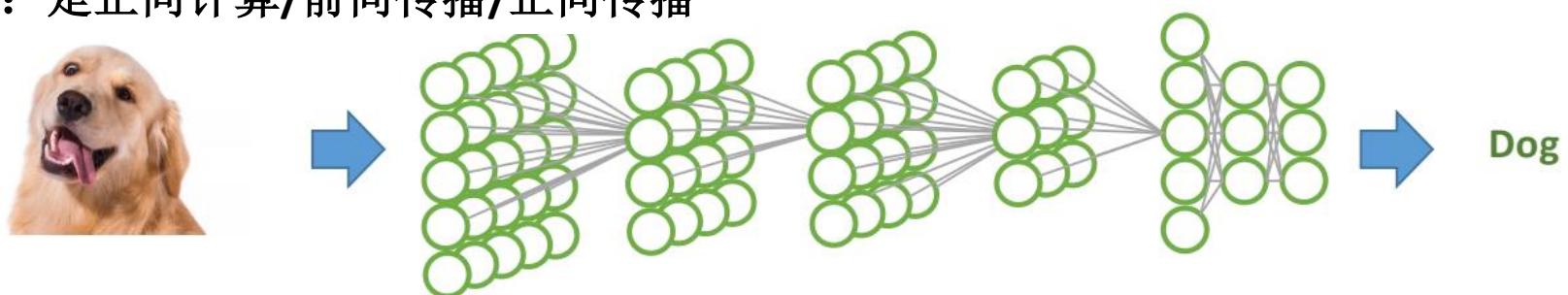
预测过程：是正向计算/前向传播/正向传播

来自：微软亚洲研究院 周礼栋《大数据系统的演化：理论、实践和展望》报告

Deep Learning Approach

训练过程：正向计算 & 反向传播（backpropagation）

预测过程：是正向计算/前向传播/正向传播

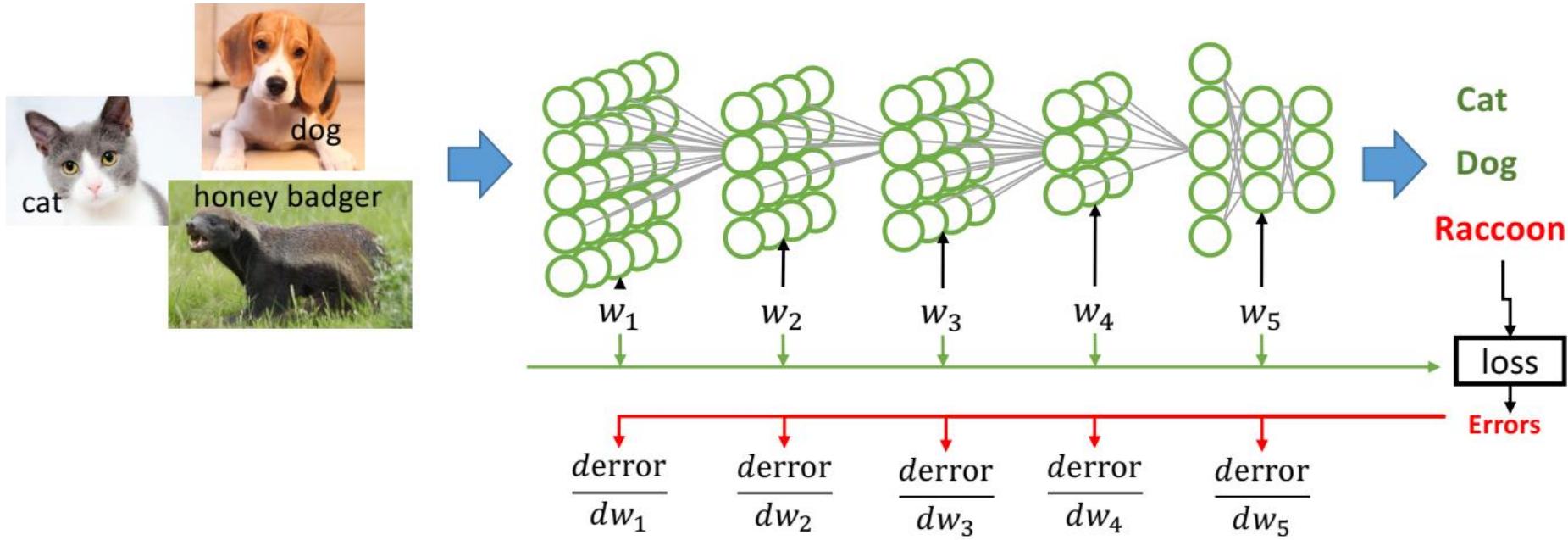


来自：微软亚洲研究院 周礼栋《大数据系统的演化：理论、实践和展望》报告

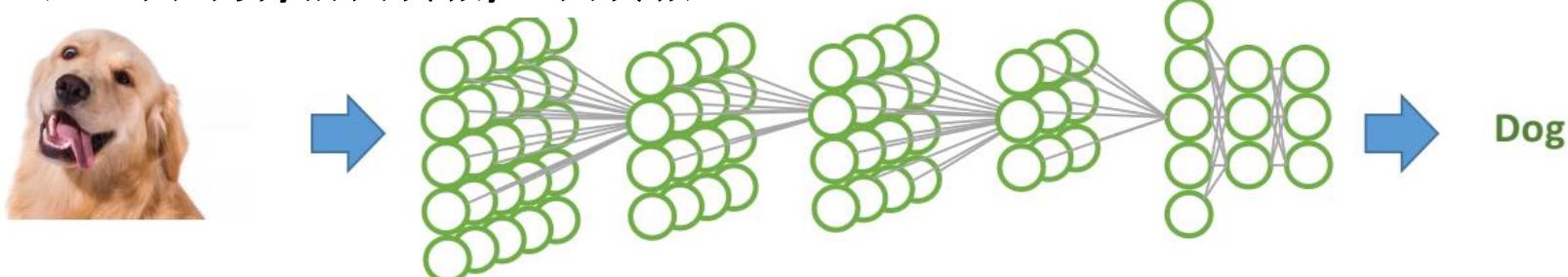
- 2个基本核心运算:
1. 正向计算
 2. 反向传播

Deep Learning Approach

训练过程: 正向计算 & 反向传播 (backpropagation)



预测过程: 是正向计算/前向传播/正向传播



来自: 微软亚洲研究院 周礼栋《大数据系统的演化: 理论、实践和展望》报告

本章内容

- 1 什么是深度学习
- 2 神经网络的数学基础
- 3 神经网络入门
- 4 神经网络的通用工作流程

- MNIST数据集
 - 包含70000张尺寸较小的手写数字图片，由美国的高中生和美国人口调查局的职员手写而成。
 - 图像被编码为Numpy数组，每个样本，即每图片有784个特征，因为每个图片都是 28×28 像素的。每个特征对应一个介于0~255之间的像素值
 - 标签是数字数组，取值范围为0~9。
 - 图像和标签一一对应。
- 任务：将手写数字的灰度图像（ 28 像素 $\times 28$ 像素）划分到10个类别中（0~9）。
- 问题设定：多分类问题
 - 机器学习中的“HelloWorld”

0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1
2 2 2 2 2 2 2 2 2 2
3 3 3 3 3 3 3 3 3 3
4 4 4 4 4 4 4 4 4 4
5 5 5 5 5 5 5 5 5 5
6 6 6 6 6 6 6 6 6 6
7 7 7 7 7 7 7 7 7 7
8 8 8 8 8 8 8 8 8 8
9 9 9 9 9 9 9 9 9 9



数字5的图片

• 安装Keras

```
C:\WINDOWS\system32>pip install keras
```

```
Collecting keras
```

```
  Downloading https://files.pythonhosted.org/packages/ad/fd/6bfe87920d7f4  
479832bdc0fe9e589a60ceb/Keras-2.3.1-py2.py3-none-any.whl (377kB)
```



```
Requirement already satisfied: six>=1.9.0 in c:\program files (x86)\micro  
d\anaconda3_64\lib\site-packages (from keras) (1.12.0)
```

```
Requirement already satisfied: pyyaml in c:\program files (x86)\microsoft
```

代码清单 2-1 加载 Keras 中的 MNIST 数据集

```
from keras.datasets import mnist  
  
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

我们来看一下训练数据：

```
>>> train_images.shape  
(60000, 28, 28)  
>>> len(train_labels)  
60000  
>>> train_labels  
array([5, 0, 4, ..., 5, 6, 8], dtype=uint8)
```

下面是测试数据：

```
>>> test_images.shape  
(10000, 28, 28)  
>>> len(test_labels)  
10000  
>>> test_labels  
array([7, 2, 1, ..., 4, 5, 6], dtype=uint8)
```

神经网络的数学基础

- 神经网络的数据表示：张量（tensor）
- 神经网络的“齿轮”：张量运算
- 神经网络的“引擎”：基于梯度的优化

神经网络的数据表示：张量 (tensor)

- 是机器学习中的基本数据结构
- 是一个数据容器。它包含的数据几乎总是数值数据，因此它是数字的容器。
- 张量的维度 (dimension)：通常叫作轴 (axis)
 - 可以用`ndim`属性来查看一个Numpy张量的轴的个数。
 - 张量轴的个数也叫作阶 (rank)
- 张量是矩阵向任意维度的推广。比如，矩阵是二维张量。
- 张量是由以下三个关键属性来定义的。
 - 轴的个数（阶）。在Numpy等Python库中也叫张量的`ndim`。
 - 形状。这是一个整数元组，表示张量沿每个轴的维度大小（元素个数）。
 - 数据类型（在Python库中通常叫作`dtype`）。这是张量中所包含数据的类型。
 - 注意，Numpy（以及大多数其他库）中不存在字符串张量，因为张量存储在预先分配的连续内存段中，而字符串的长度是可变的，无法用这种方式存储。

- 观察MNIST数据集中的数据

```
from keras.datasets import mnist  
  
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

接下来，我们给出张量 `train_images` 的轴的个数，即 `ndim` 属性。

```
>>> print(train_images.ndim)  
3
```

下面是它的形状。

```
>>> print(train_images.shape)  
(60000, 28, 28)
```

下面是它的数据类型，即 `dtype` 属性。

```
>>> print(train_images.dtype)  
uint8
```

神经网络的数据表示：张量 (tensor)

- 按照维度，张量分为：
 - 标量（0D张量）：仅包含一个数字的张量叫作标量（scalar，也叫标量张量、零维张量、0D张量）。
 - 向量（1D张量）
 - 矩阵（2D张量）
 - 3D张量与更高维张量

- **标量（0D张量）**：仅包含一个数字的张量叫作标量（scalar，也叫标量张量、零维张量、0D张量）。
- 比如，在Numpy中，一个float32或float64的数字就是一个标量张量（或标量数组）。
- 标量张量有0个轴（ndim == 0）。

```
>>> import numpy as np  
>>> x = np.array(12)  
>>> x  
array(12)  
>>> x.ndim  
0
```

• 向量（1D张量）

数字组成的数组叫作向量（vector）或一维张量（1D 张量）。一维张量只有一个轴。下面是一个 Numpy 向量。

```
>>> x = np.array([12, 3, 6, 14, 7])
>>> x
array([12, 3, 6, 14, 7])
>>> x.ndim
1
```

这个向量有 5 个元素，所以被称为 5D 向量。不要把 5D 向量和 5D 张量弄混！5D 向量只有一个轴，沿着轴有 5 个维度，而 5D 张量有 5 个轴（沿着每个轴可能有任何个维度）。维度（dimensionality）可以表示沿着某个轴上的元素个数（比如 5D 向量），也可以表示张量中轴的个数（比如 5D 张量），这有时会令人感到混乱。对于后一种情况，技术上更准确的说法是 5 阶张量（张量的阶数即轴的个数），但 5D 张量这种模糊的写法更常见。

• 矩阵（2D张量）

向量组成的数组叫作矩阵（matrix）或二维张量（2D 张量）。矩阵有 2 个轴（通常叫作行和列）。你可以将矩阵直观地理解为数字组成的矩形网格。下面是一个 Numpy 矩阵。

```
>>> x = np.array([[5, 78, 2, 34, 0],  
                 [6, 79, 3, 35, 1],  
                 [7, 80, 4, 36, 2]])  
  
>>> x.ndim  
2
```

第一个轴上的元素叫作行（row），第二个轴上的元素叫作列（column）。在上面的例子中，`[5, 78, 2, 34, 0]` 是 `x` 的第一行，`[5, 6, 7]` 是第一列。

• 3D张量与更高维张量

将多个矩阵组合成一个新的数组，可以得到一个 3D 张量，你可以将其直观地理解为数字组成的立方体。下面是一个 Numpy 的 3D 张量。

```
>>> x = np.array([[[5, 78, 2, 34, 0],  
                 [6, 79, 3, 35, 1],  
                 [7, 80, 4, 36, 2]],  
                [[[5, 78, 2, 34, 0],  
                  [6, 79, 3, 35, 1],  
                  [7, 80, 4, 36, 2]],  
                 [[5, 78, 2, 34, 0],  
                  [6, 79, 3, 35, 1],  
                  [7, 80, 4, 36, 2]]])  
  
>>> x.ndim  
3
```

将多个 3D 张量组合成一个数组，可以创建一个 4D 张量，以此类推。深度学习处理的一般是 0D 到 4D 的张量，但处理视频数据时可能会遇到 5D 张量。

在Numpy中操作张量

- 使用语法`train_images[i]`来选择沿着第一个轴的特定数字。选择张量的特定元素叫作张量切片（**tensor slicing**）。
- Numpy数组上的张量切片运算示例：
 - 选择第10~100个数字（不包括第100个），并将其放在形状为(90, 28, 28)的数组中。

```
>>> my_slice = train_images[10:100]
>>> print(my_slice.shape)
(90, 28, 28)
```

注意，`:`等同于选择整个轴。

```
>>> my_slice = train_images[10:100, :, :] ← 等同于前面的例子
>>> my_slice.shape
(90, 28, 28)
>>> my_slice = train_images[10:100, 0:28, 0:28] ← 也等同于前面的例子
>>> my_slice.shape
(90, 28, 28)
```

数据批量

- 通常来说，深度学习中所有数据张量的第一个轴（0轴，因为索引从0开始）都是**样本轴**（samples axis，有时也叫**样本维度**）。
 - 比如，在MNIST的例子中，样本就是数字图像。
- 深度学习模型不会同时处理整个数据集，而是将数据拆分成小批量。
 - 比如，下面是MNIST数据集的一个批量，批量大小为128。

```
batch = train_images[:128]
```
 - 然后是下一个批量。

```
batch = train_images[128:256]
```
 - 然后是第n个批量。

```
batch = train_images[128 * n:128 * (n + 1)]
```
- 对于这种批量张量，第一个轴（0轴）叫作**批量轴**（batch axis）或**批量维度**（batch dimension）。

现实世界中的数据张量

- 向量数据： 2D张量， 形状为(samples, features)。
- 时间序列数据或序列数据： 3D张量， 形状为(samples, timesteps, features)。
- 图像： 4D张量， 形状为(samples, height, width, channels)或(samples, channels, height, width)。
- 视频： 5D张量， 形状为(samples, frames, height, width, channels)或(samples, frames, channels, height, width)。

现实世界中的数据张量 — 向量数据

- 是最常见的数据。
- 对于MNIST数据集，每个数据点(即每个样本/每张图像)都被编码为一个向量，因此一个数据批量就被编码为2D张量（即向量组成的数组），其中第一个轴是样本轴，第二个轴是特征轴。
- 人口统计数据集，其中包括每个人的年龄、邮编和收入。每个人可以表示为包含3个值的向量，而整个数据集包含100 000个人，因此可以存储在形状为 $(100000, 3)$ 的2D张量中。
- 文本文档数据集，我们将每个文档表示为每个单词在其中出现的次数（字典中包含20 000个常见单词）。每个文档可以被编码为包含20 000个值的向量（每个值对应于字典中每个单词的出现次数），整个数据集包含500个文档，因此可以存储在形状为 $(500, 20000)$ 的张量中。

现实世界中的数据张量 — 时间序列数据或序列数据

- 当时间（或序列顺序）对于数据很重要时，应该将数据存储在带有时间轴的3D张量中。每个样本可以被编码为一个向量序列（即2D张量），因此一个数据批量就被编码为一个3D张量（见图2-3）。

- 根据惯例，时间轴始终是第2个轴（索引为1的轴）

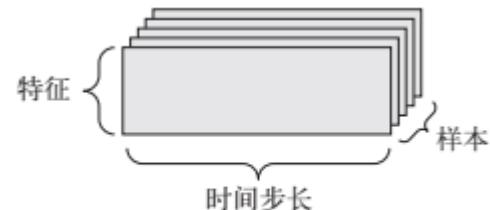


图 2-3 时间序列数据组成的 3D 张量

- 股票价格数据集。每一分钟，我们将股票的当前价格、前一分钟的最高价格和前一分钟的最低价格保存下来。因此每分钟被编码为一个3D向量，整个交易日被编码为一个形状为 $(390, 3)$ 的2D张量（一个交易日有390分钟），而250天的数据则可以保存在一个形状为 $(250, 390, 3)$ 的3D张量中。这里每个样本是一天的股票数据。
- 推文数据集。我们将每条推文编码为280个字符组成的序列，而每个字符又来自于128个字符组成的字母表。在这种情况下，每个字符可以被编码为大小为128的二进制向量（只有在该字符对应的索引位置取值为1，其他元素都为0）。那么每条推文可以被编码为一个形状为 $(280, 128)$ 的2D张量，而包含100万条推文的数据集则可以存储在一个形状为 $(1000000, 280, 128)$ 的张量中。

现实世界中的数据张量 – 图像数据

- 描述一张图像张量始终都是3D张量。
 - 图像通常具有三个维度：高度、宽度和颜色深度。
- 考虑到图像的通道数的不同，
 - 灰度图像的彩色通道只有一维。因此，如果图像大小为 256×256 ，那么128张灰度图像组成的批量可以保存在一个形状为(128, 256, 256, 1)的张量中
 - 灰度图像（比如MNIST数字图像）只有一个颜色通道，因此可以保存在2D张量中
 - 128张彩色图像组成的批量则可以保存在一个形状为(128, 256, 256, 3)的张量中。
- 图像张量的形状有两种约定：
 - 通道在后（channels-last）的约定（在TensorFlow中使用）
 - (samples, height, width, color_depth)
 - 通道在前（channels-first）的约定（在Theano中使用）。
 - (samples, color_depth, height, width)
- Keras框架同时支持这两种格式。

现实世界中的数据张量 — 图像数据

- 描述一张图像张量始终都是3D张量。
 - 图像通常具有三个维度：高度、宽度和颜色深度。
- 考虑到图像的通道数的不同，图像数据是4D张量。
 - 灰度图像的彩色通道只有一维。因此，如果图像大小为 256×256 ，那么128张灰度图像组成的批量可以保存在一个形状为 $(128, 256, 256, 1)$ 的张量中
 - 灰度图像（比如MNIST数字图像）只有一个颜色通道，因此可以保存在2D张量中
 - 128张彩色图像组成的批量则可以保存在一个形状为 $(128, 256, 256, 3)$ 的张量中。
- 图像张量的形状有两种约定：
 - 通道在后（channels-last）的约定（在TensorFlow中）
 - $(\text{samples}, \text{height}, \text{width}, \text{color_depth})$
 - 通道在前（channels-first）的约定（在TensorFlow和Keras中）
 - $(\text{samples}, \text{color_depth}, \text{height}, \text{width})$
- Keras框架同时支持这两种格式。

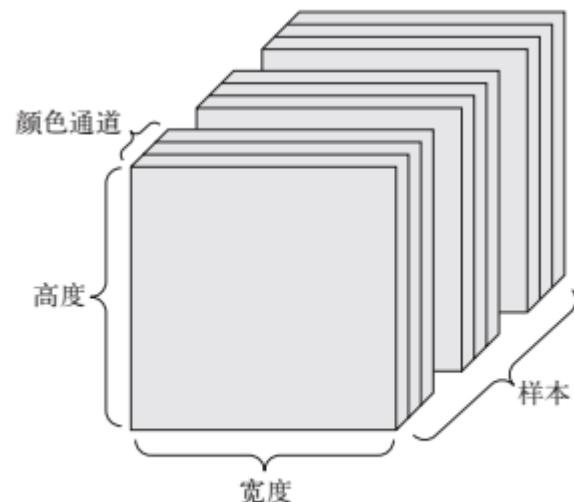


图 2-4 图像数据组成的 4D 张量（通道在前的约定）

现实世界中的数据张量 – 视频数据

- 视频数据是现实生活中需要用到5D张量的少数数据类型之一。
 - 视频可以看作一系列帧，每一帧都是一张彩色图像。每一帧都可以保存在一个形状为(`height, width, color_depth`)的3D张量中，因此一系列帧可以保存在一个形状为(`frames, height, width, color_depth`)的4D张量中，而不同视频组成的批量则可以保存在一个5D张量中，其形状为(`samples, frames, height, width, color_depth`)。
- 举个例子，一个以每秒4帧采样的60秒YouTube视频片段，视频尺寸为 144×256 ，这个视频共有240帧。4个这样的视频片段组成的批量将保存在形状为(`4, 240, 144, 256, 3`)的张量中。总共有106 168 320个值！如果张量的数据类型（`dtype`）是`float32`，每个值都是32位，那么这个张量共有405MB。
 - 现实生活中遇到的视频要小得多，因为它们不以`float32`格式存储，而且通常被大大压缩，比如MPEG格式。

神经网络的“齿轮”：张量运算

- **逐元素运算：**该运算独立地应用于张量中的每个元素。
 - `relu`运算和加法都是逐元素（**element-wise**）的运算
 - 非常适合大规模并行实现（向量化实现，这一术语来自于1970—1990年间向量处理器超级计算机架构）。

• 加法运算（+）

```
import numpy as np

x = np.random.random((64, 3, 32, 10))      ← x 是形状为 (64, 3, 32, 10) 的随机张量
y = np.random.random((32, 10))            ← y 是形状为 (32, 10) 的随机张量

z = np.maximum(x, y)          ← 输出 z 的形状是 (64, 3, 32, 10), 与 x 相同
```

• **relu**运算

```
import numpy as np

z = x + y      ← 逐元素的相加

z = np.maximum(z, 0.)    ← 逐元素的 relu
```

神经网络的“齿轮”：张量运算

- **广播：**两个形状不同的张量进行运算时，如果没有歧义的话，较小的张量会被广播（broadcast），以匹配较大张量的形状。广播包含以下两步。
 1. 向较小的张量添加轴（叫作广播轴），使其`ndim`与较大的张量相同。
 2. 将较小的张量沿着新轴重复，使其形状与较大的张量相同。

```
import numpy as np

x = np.random.random((64, 3, 32, 10))      ← x 是形状为 (64, 3, 32, 10) 的随机张量
y = np.random.random((32, 10))            ← y 是形状为 (32, 10) 的随机张量

z = np.maximum(x, y)          ← 输出 z 的形状是 (64, 3, 32, 10), 与 x 相同
```

神经网络的“齿轮”：张量运算

- **点积运算 (dot)**：也叫张量积 (tensor product, 不要与逐元素的乘积弄混)，是最常见也最有用的张量运算。与逐元素的运算不同，它将输入张量的元素合并在一起。
- 在Numpy、Keras、Theano和TensorFlow中，都是用`*`实现逐元素乘积。TensorFlow中的点积使用了不同的语法，但在Numpy和Keras中，都是用标准的`dot`运算符来实现点积。
- 注意：`dot(x, y) ≠ dot(y, x)`

对于两个矩阵 x 和 y ，当且仅当：

$x.shape[1] == y.shape[0]$ 时，才可以对 x 和 y 做点积，点积结果是一个形状为 $(x.shape[0], y.shape[1])$ 的矩阵。

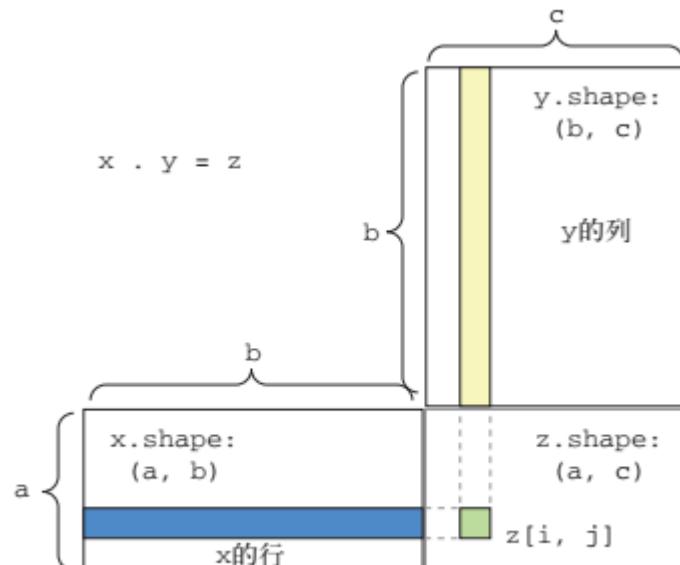


图 2-5 图解矩阵点积

神经网络的“齿轮”：张量运算

- 张量变形

```
>>> x = np.array([[0., 1.],  
                 [2., 3.],  
                 [4., 5.]])  
>>> print(x.shape)  
(3, 2)  
>>> x = x.reshape((6, 1))  
>>> x  
array([[ 0.],  
       [ 1.],  
       [ 2.],  
       [ 3.],  
       [ 4.],  
       [ 5.]])  
>>> x = x.reshape((2, 3))  
>>> x  
array([[ 0.,  1.,  2.],  
       [ 3.,  4.,  5.]])
```

- 转置（transposition）：特殊的张量变形
 - 对矩阵做转置是指将行和列互换，使 $x[i, :]$ 变为 $x[:, i]$ 。

```
>>> x = np.zeros((300, 20))    ← 创建一个形状为 (300, 20) 的零矩阵  
>>> x = np.transpose(x)  
>>> print(x.shape)  
(20, 300)
```

神经网络的“引擎”：基于梯度的优化

- 一个训练循环（**training loop**）包含的具体过程如下：
 - (1)抽取训练样本 x 和对应目标 y 组成的数据批量。
 - (2)在 x 上运行网络，得到预测值 y_{pred} 。〔这一步叫作前向传播（**forward pass**）〕
 - (3)计算网络在这批数据上的损失，用于衡量 y_{pred} 和 y 之间的距离。
 - (4)更新网络的所有权重，使网络在这批数据上的损失略微下降。
- 经过多轮的训练循环后，最终得到的网络在训练数据上的损失非常小，即预测值 y_{pred} 和预期目标 y 之间的距离非常小。网络“学会”将输入映射到正确的目标。
- 存在问题：如何使得每次训练循环后，损失都下降？
 - 利用网络中所有运算都是可微（**differentiable**）的这一事实，计算损失相对于网络系数的梯度（**gradient**），然后向梯度的反方向改变系数，从而使损失降低。

- 什么是导数？

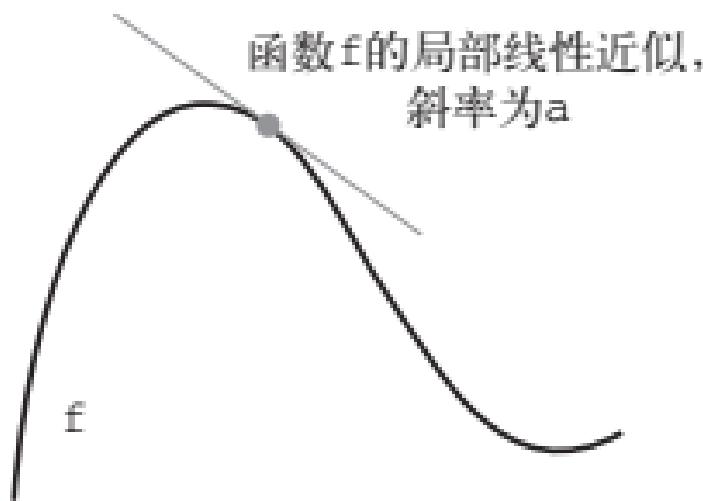


图 2-10 f 在 p 点的导数

- 某函数 $f(x)$ 可微，是指“可以被求导”。
- 梯度：张量运算的导数
 - 梯度（gradient）是张量运算的导数。它是导数这一概念向多元函数导数的推广。多元函数是以张量作为输入的函数。

链式求导：反向传播算法(Backpropagation)

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



Caffe



theano



Gradient Descent

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

Starting Parameters $\theta^{<0>} \rightarrow \theta^{<1>} \rightarrow \theta^{<2>} \rightarrow \dots$

$$\nabla L(\theta) = \begin{bmatrix} \partial L(\theta)/\partial w_1 \\ \partial L(\theta)/\partial w_2 \\ \vdots \\ \partial L(\theta)/\partial b_1 \\ \partial L(\theta)/\partial b_2 \\ \vdots \end{bmatrix}$$

Compute $\nabla L(\theta^{<0>})$ $\theta^{<1>} = \theta^{<0>} - \eta \nabla L(\theta^{<0>})$

Compute $\nabla L(\theta^{<1>})$ $\theta^{<2>} = \theta^{<1>} - \eta \nabla L(\theta^{<1>})$

Millions of parameters

To compute the gradients efficiently,
we use **backpropagation**.

Chain Rule

Case 1

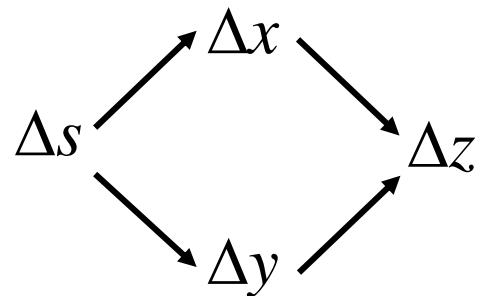
$$y = g(x) \quad z = h(y)$$

$$\Delta x \rightarrow \Delta y \rightarrow \Delta z$$

$$\frac{dz}{dx} = \frac{dz}{dy} \frac{dy}{dx}$$

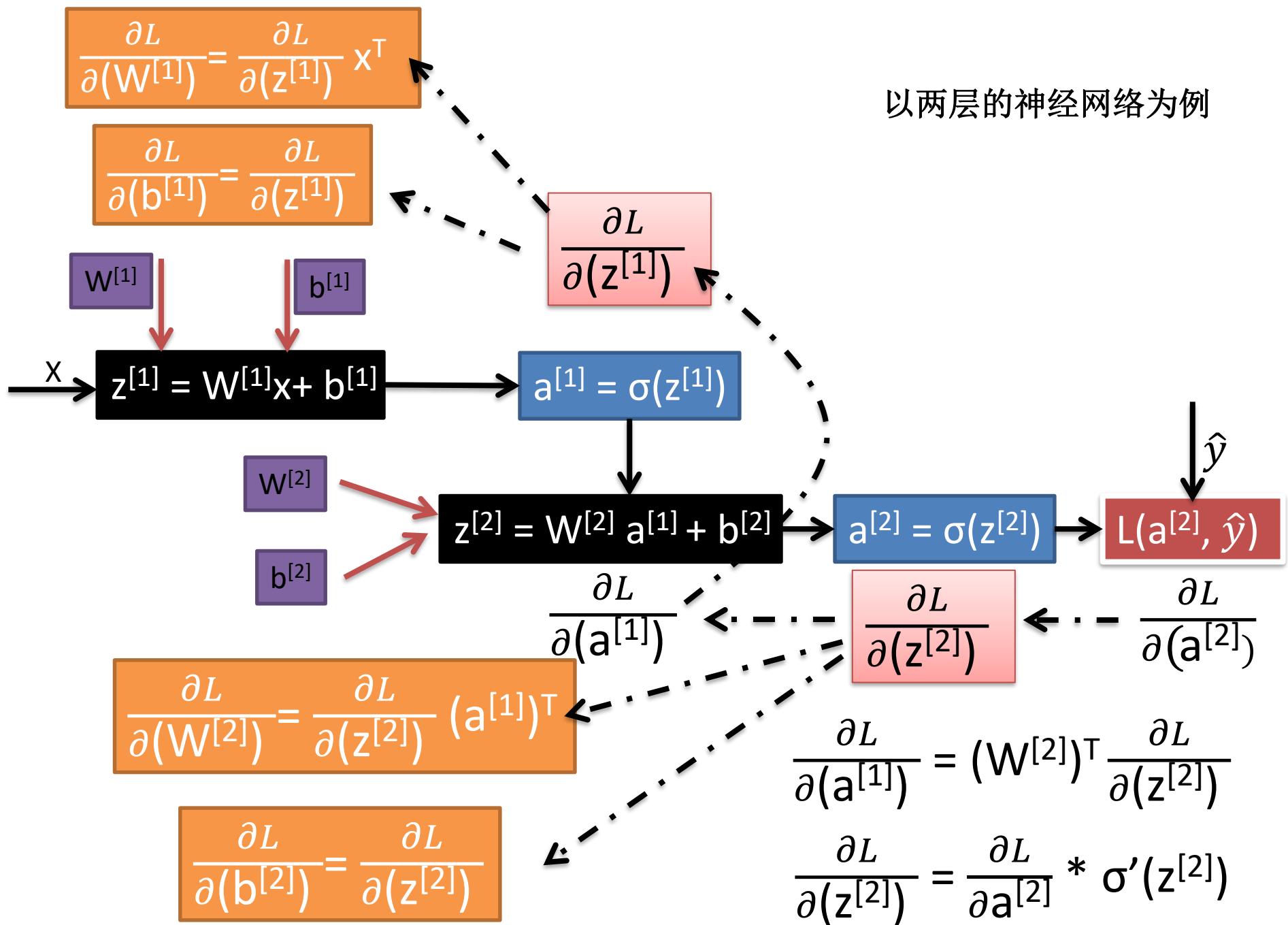
Case 2

$$x = g(s) \quad y = h(s) \quad z = k(x, y)$$

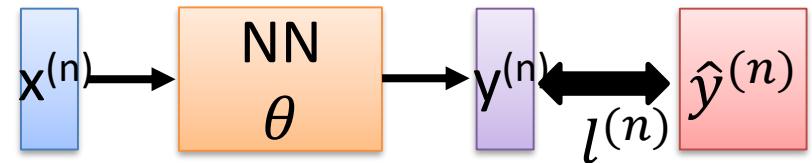


$$\frac{dz}{ds} = \frac{\partial z}{\partial x} \frac{dx}{ds} + \frac{\partial z}{\partial y} \frac{dy}{ds}$$

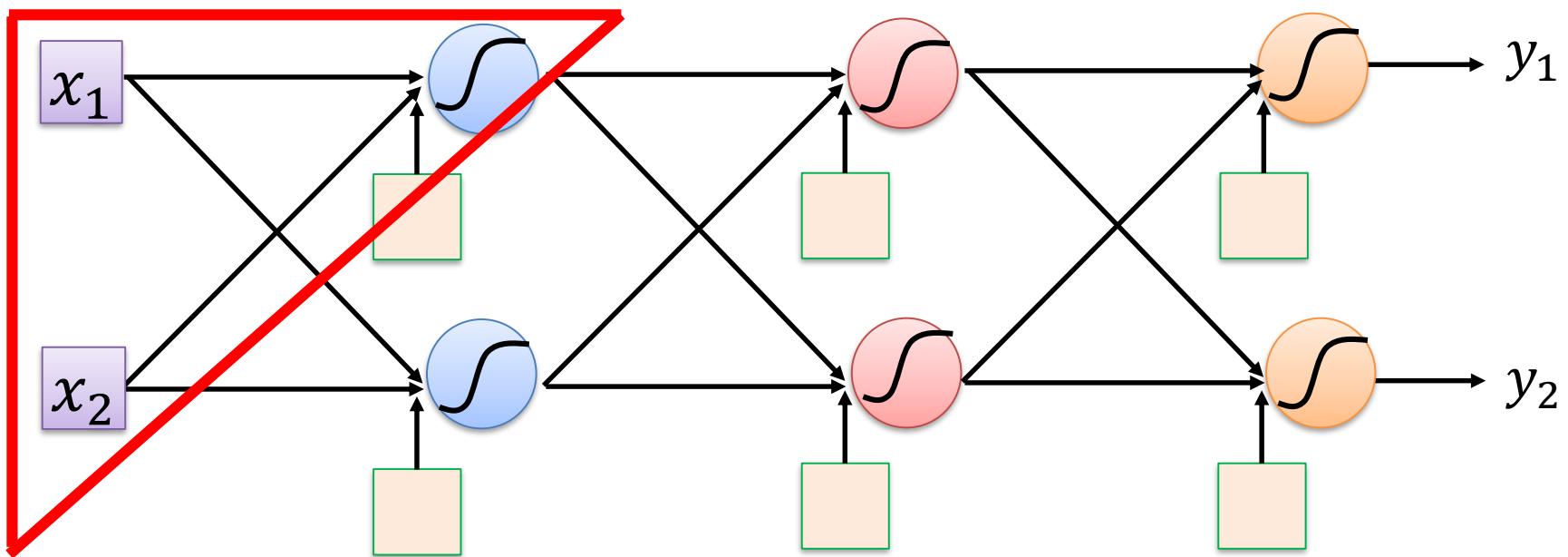
以两层的神经网络为例



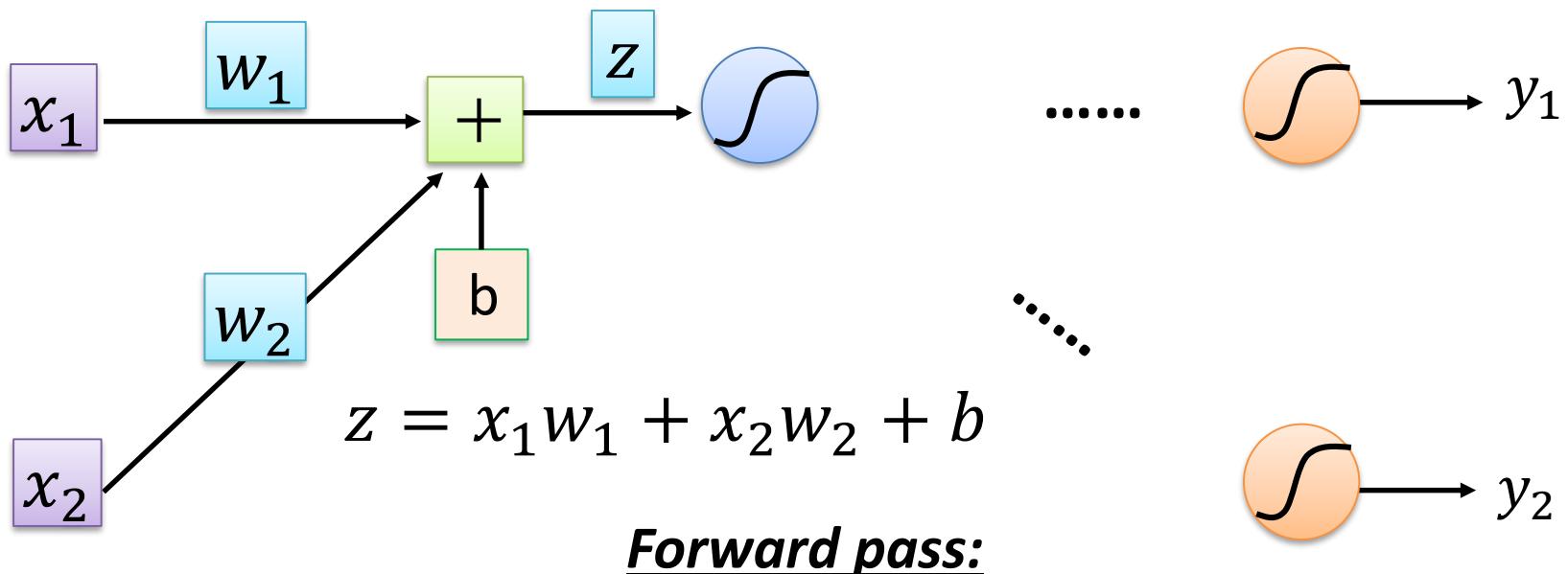
Backpropagation



$$L(\theta) = \sum_{n=1}^N l^{(n)}(\theta) \rightarrow \frac{\partial L(\theta)}{\partial w} = \sum_{n=1}^N \frac{\partial l^{(n)}(\theta)}{\partial w}$$



Backpropagation



$$\frac{\partial l}{\partial w} = ? \quad \frac{\partial z}{\partial w} \frac{\partial l}{\partial z}$$

(Chain rule)

Compute $\partial z / \partial w$ for all parameters

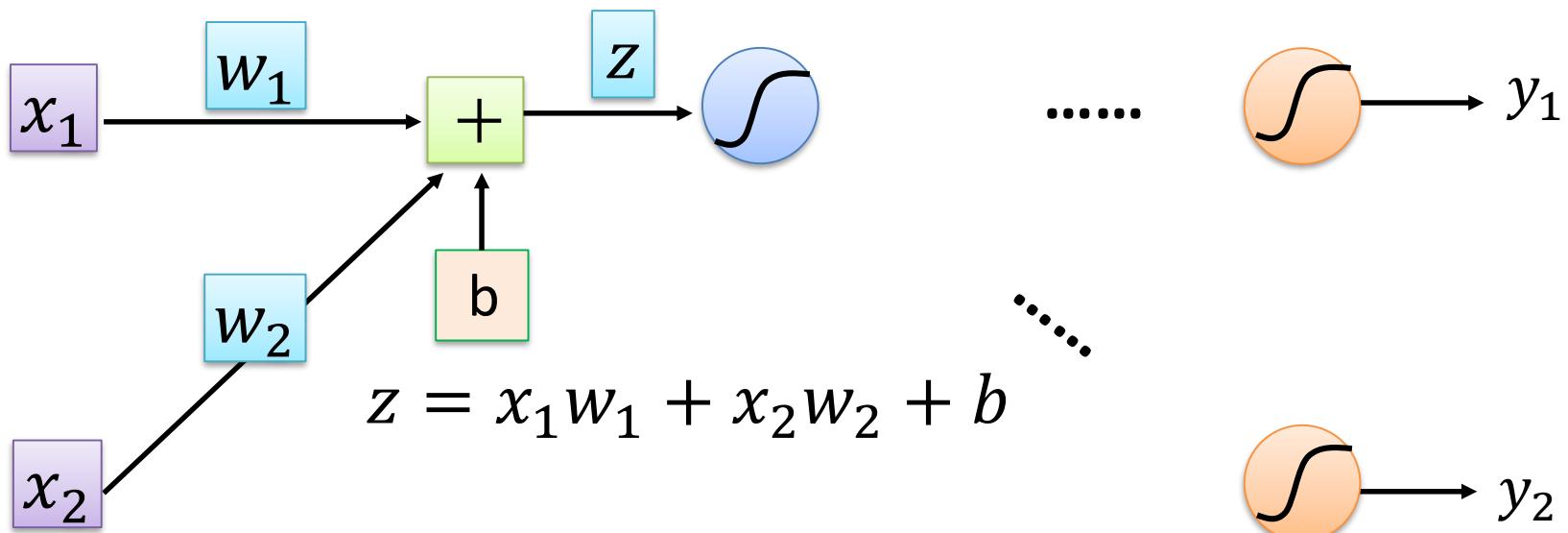
Backward pass:

Compute $\partial l / \partial z$ for all activation function inputs z

Backpropagation – Forward pass

Compute $\partial z / \partial w$ for all parameters

$$\frac{\partial l}{\partial w} = \frac{\partial z}{\partial w} \frac{\partial l}{\partial z} \text{ (Chain rule)}$$



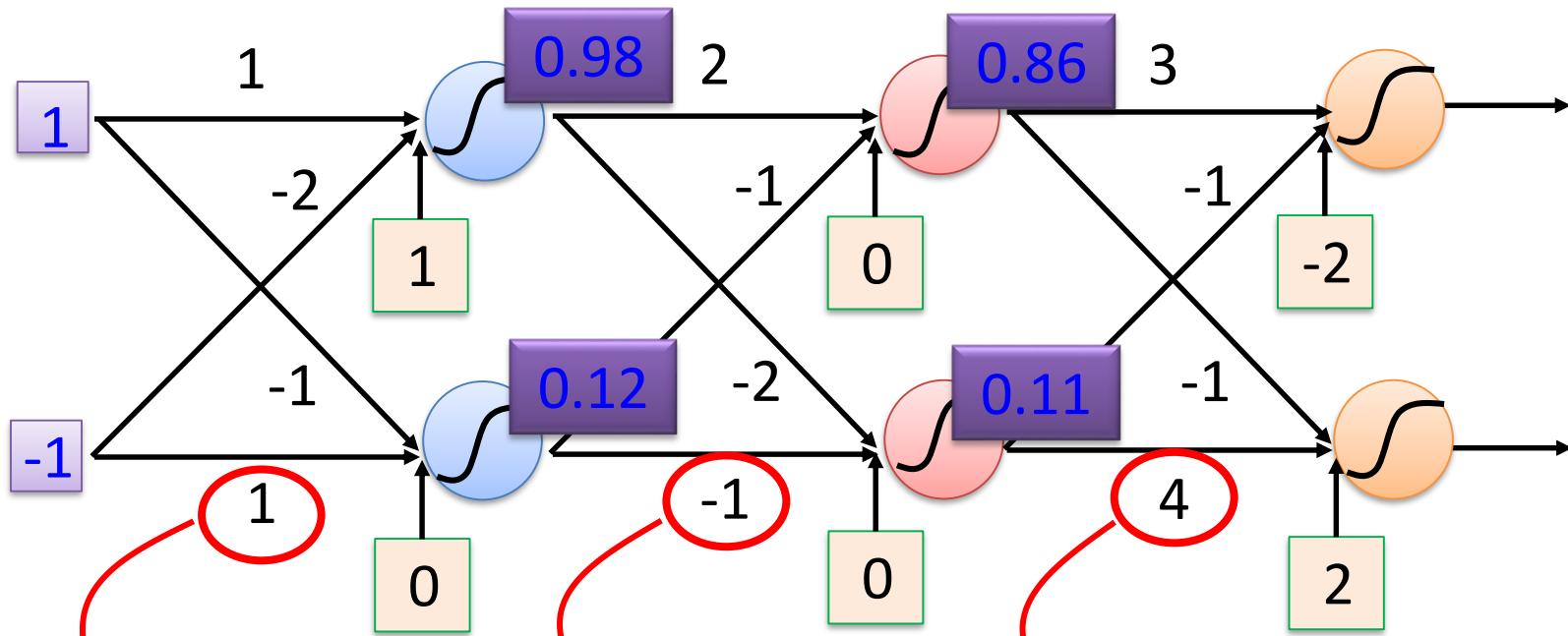
$$\begin{aligned}\partial z / \partial w_1 &=? x_1 \\ \partial z / \partial w_2 &=? x_2\end{aligned}$$

The value of the input
connected by the weight

Backpropagation – Forward pass

$$\frac{\partial l}{\partial w} = \frac{\partial z}{\partial w} \frac{\partial l}{\partial z} \text{ (Chain rule)}$$

Compute $\frac{\partial z}{\partial w}$ for all parameters



$$\frac{\partial z}{\partial w} = -1$$

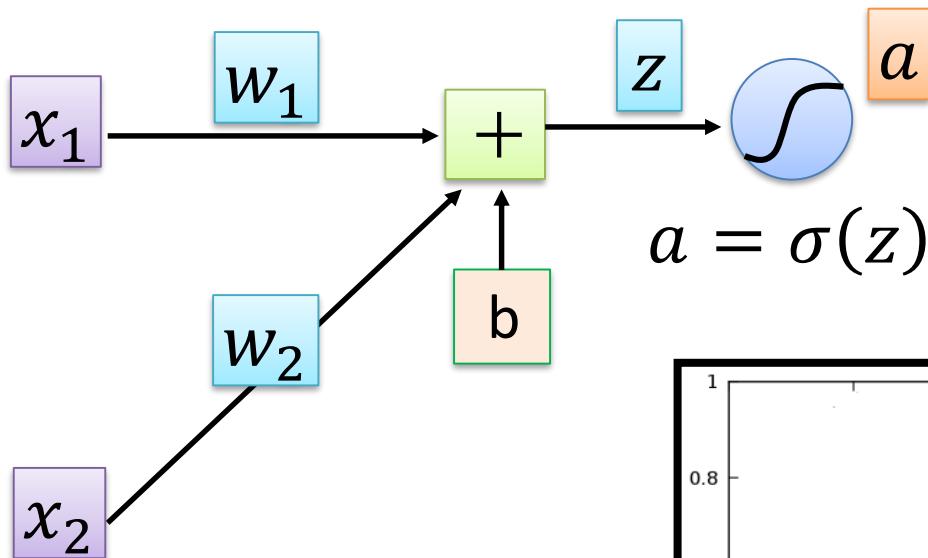
$$\frac{\partial z}{\partial w} = 0.12$$

$$\frac{\partial z}{\partial w} = 0.11$$

Backpropagation – Backward pass

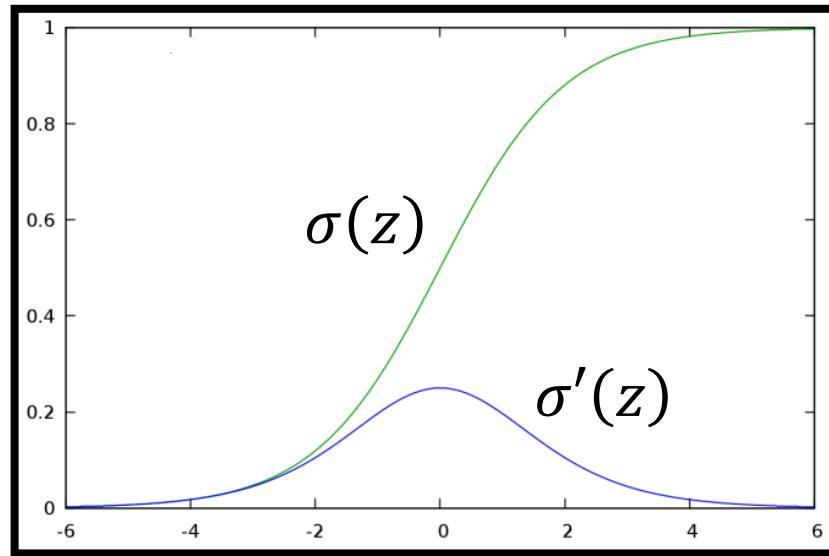
$$\frac{\partial l}{\partial w} = \frac{\partial z}{\partial w} \frac{\partial l}{\partial z} \quad (\text{Chain rule})$$

Compute $\partial l / \partial z$ for all activation function inputs z



$$\frac{\partial l}{\partial z} = \frac{\partial a}{\partial z} \frac{\partial l}{\partial a}$$

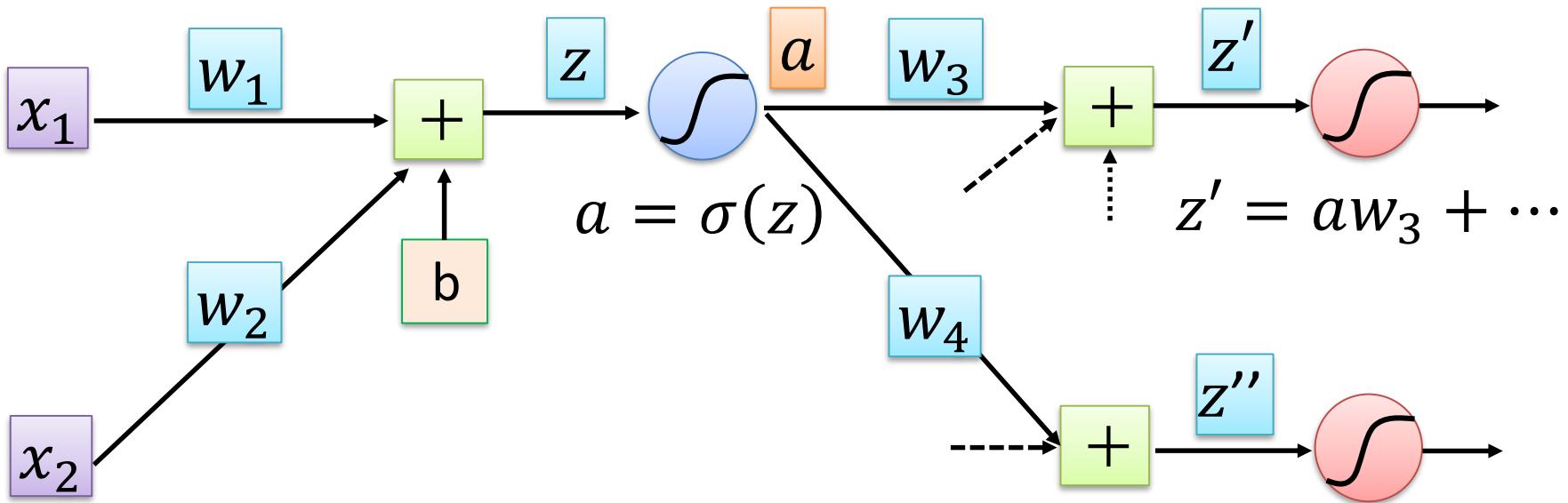
→ $\sigma'(z)$



Backpropagation – Backward pass

$$\frac{\partial l}{\partial w} = \frac{\partial z}{\partial w} \frac{\partial l}{\partial z} \quad (\text{Chain rule})$$

Compute $\partial l / \partial z$ for all activation function inputs z



$$\frac{\partial l}{\partial z} = \frac{\partial a}{\partial z} \frac{\partial l}{\partial a}$$

$$\frac{\partial l}{\partial a} = \frac{\partial z'}{\partial a} \frac{\partial l}{\partial z'} + \frac{\partial z''}{\partial a} \frac{\partial l}{\partial z''} \quad (\text{Chain rule})$$

w_3

?

w_4

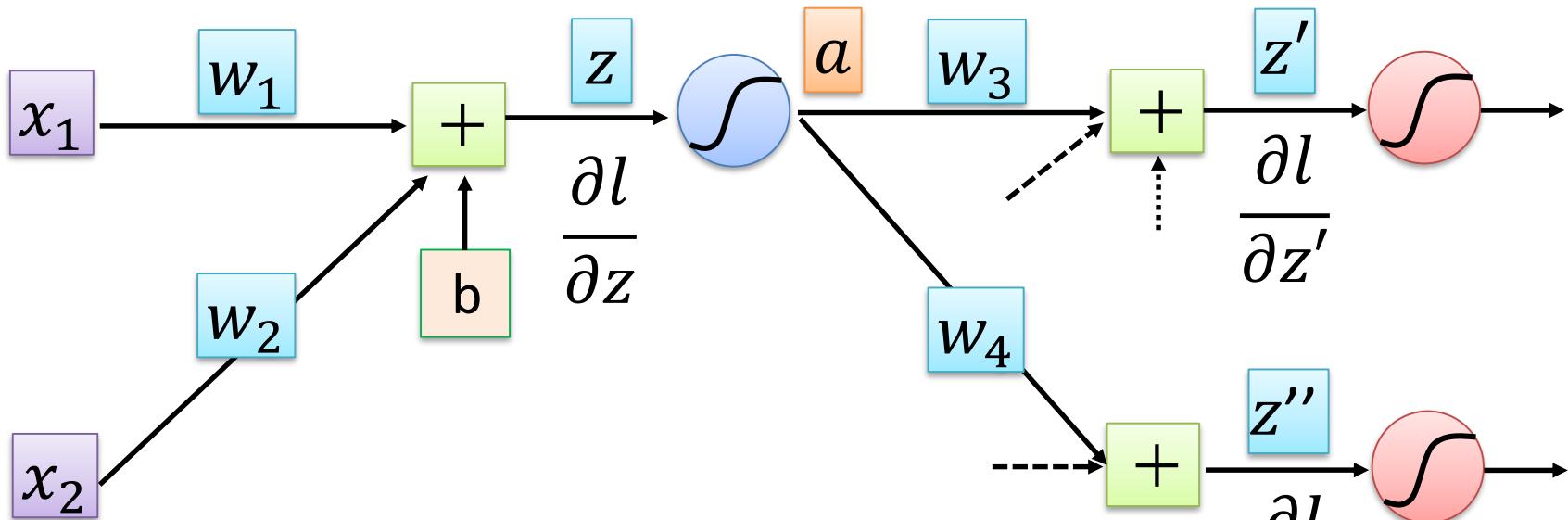
?

Assumed
it's known

Backpropagation – Backward pass

$$\frac{\partial l}{\partial w} = \frac{\partial z}{\partial w} \frac{\partial l}{\partial z} \quad (\text{Chain rule})$$

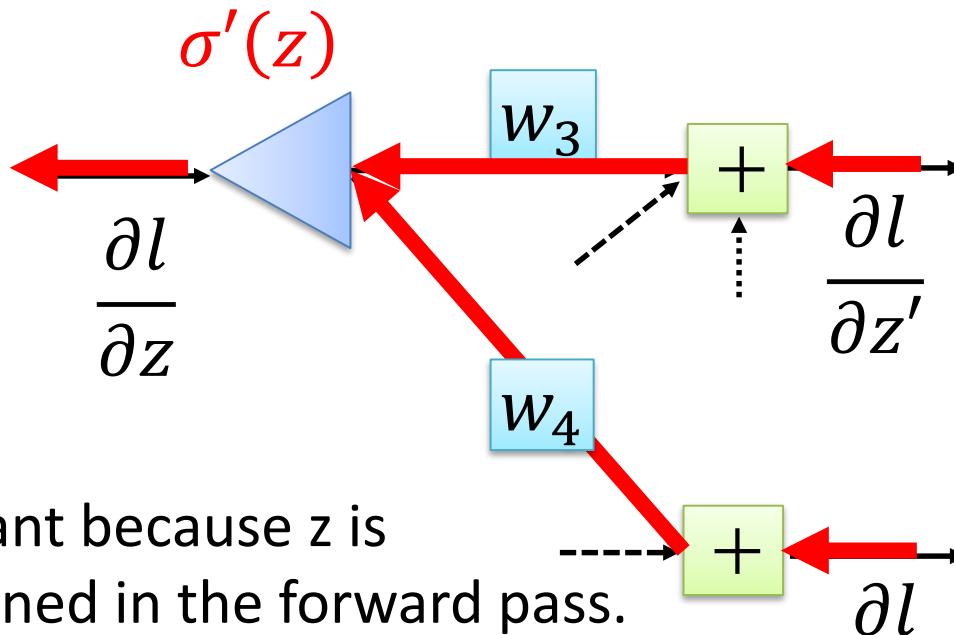
Compute $\frac{\partial l}{\partial z}$ for all activation function inputs z



$$\frac{\partial l}{\partial z} = \sigma'(z) \left[w_3 \frac{\partial l}{\partial z'} + w_4 \frac{\partial l}{\partial z''} \right]$$

Backpropagation – Backward pass

$$\frac{\partial l}{\partial w} = \frac{\partial z}{\partial w} \frac{\partial l}{\partial z} \quad (\text{Chain rule})$$



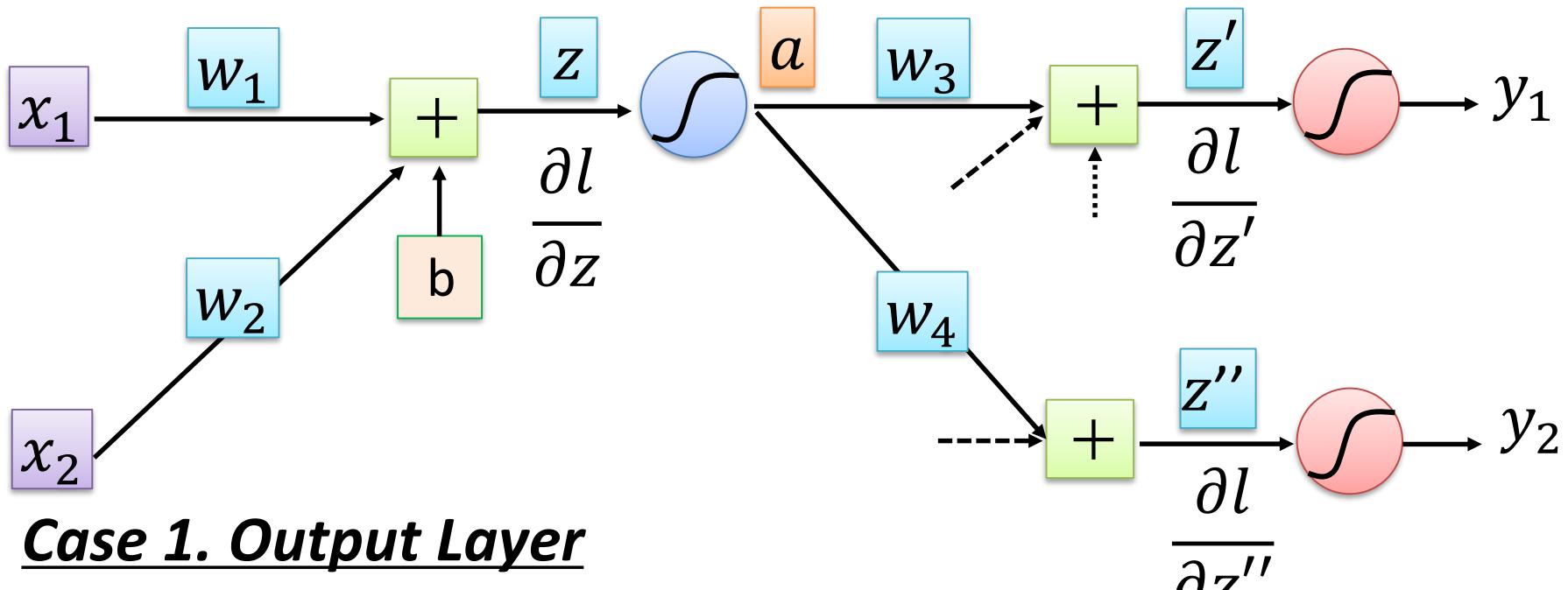
$\sigma'(z)$ is a constant because z is already determined in the forward pass.

$$\frac{\partial l}{\partial z} = \sigma'(z) \left[w_3 \frac{\partial l}{\partial z'} + w_4 \frac{\partial l}{\partial z''} \right]$$

Backpropagation – Backward pass

$$\frac{\partial l}{\partial w} = \frac{\partial z}{\partial w} \frac{\partial l}{\partial z} \quad (\text{Chain rule})$$

Compute $\frac{\partial l}{\partial z}$ for all activation function inputs z



Case 1. Output Layer

$$\frac{\partial l}{\partial z'} = \frac{\partial y_1}{\partial z'} \frac{\partial l}{\partial y_1}$$

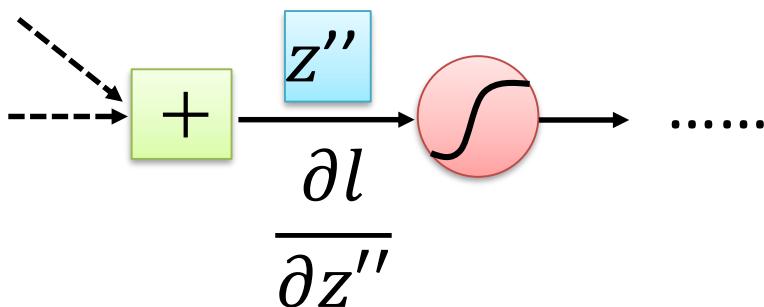
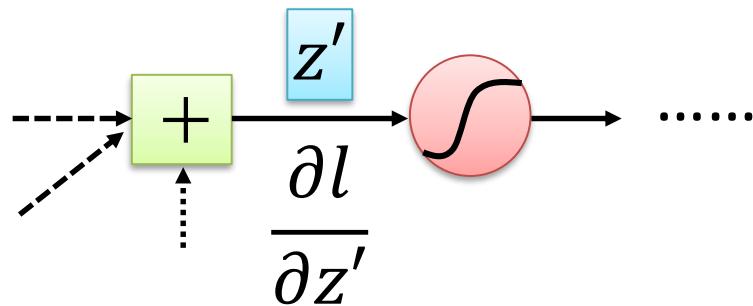
$$\frac{\partial l}{\partial z''} = \frac{\partial y_2}{\partial z''} \frac{\partial l}{\partial y_2}$$

Done!

Backpropagation – Backward pass

Compute $\frac{\partial l}{\partial z}$ for all activation function inputs z

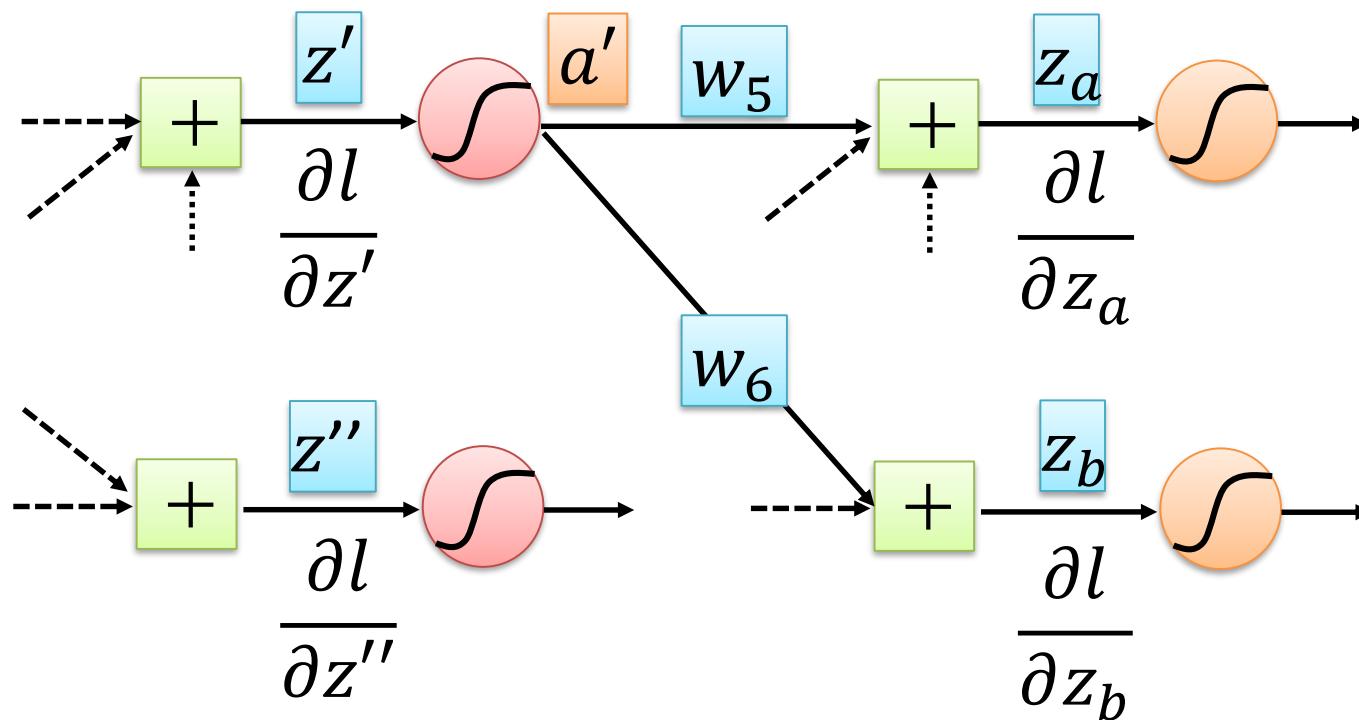
Case 2. Not Output Layer



Backpropagation – Backward pass

Compute $\frac{\partial l}{\partial z}$ for all activation function inputs z

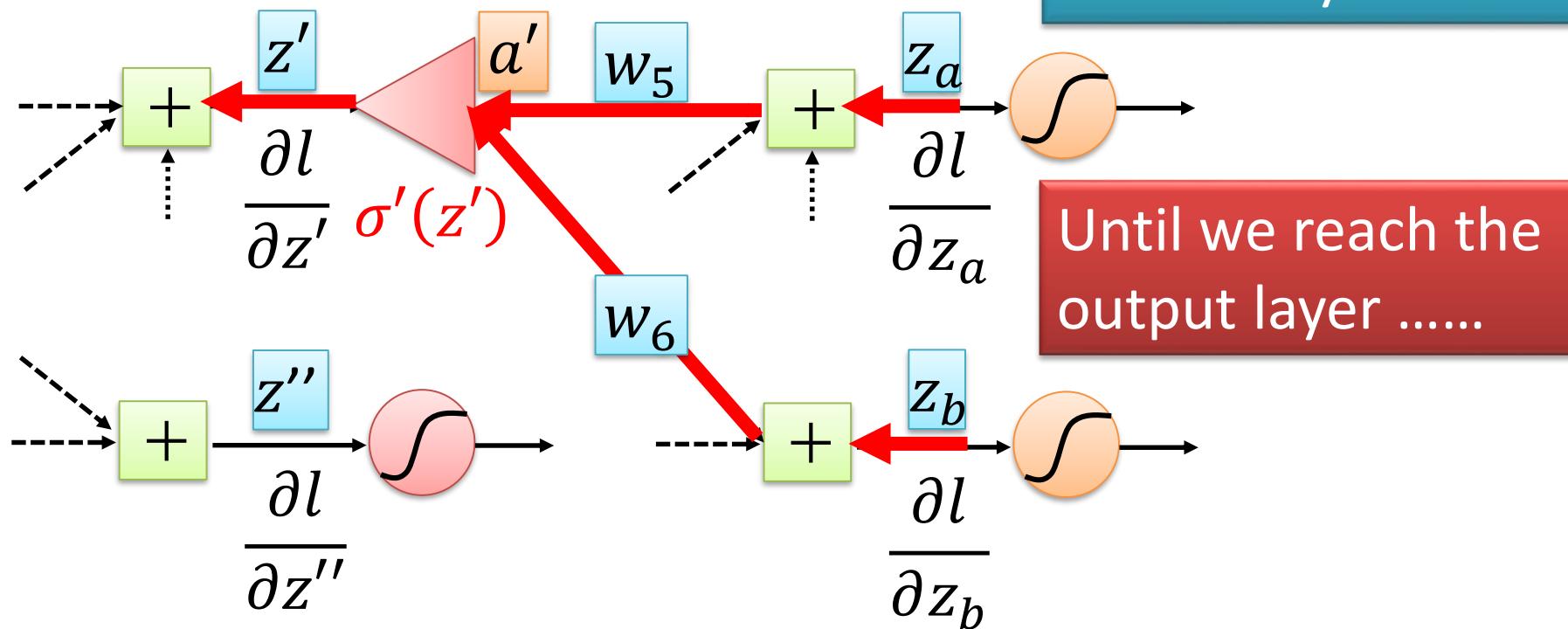
Case 2. Not Output Layer



Backpropagation – Backward pass

Compute $\frac{\partial l}{\partial z}$ for all activation function inputs z

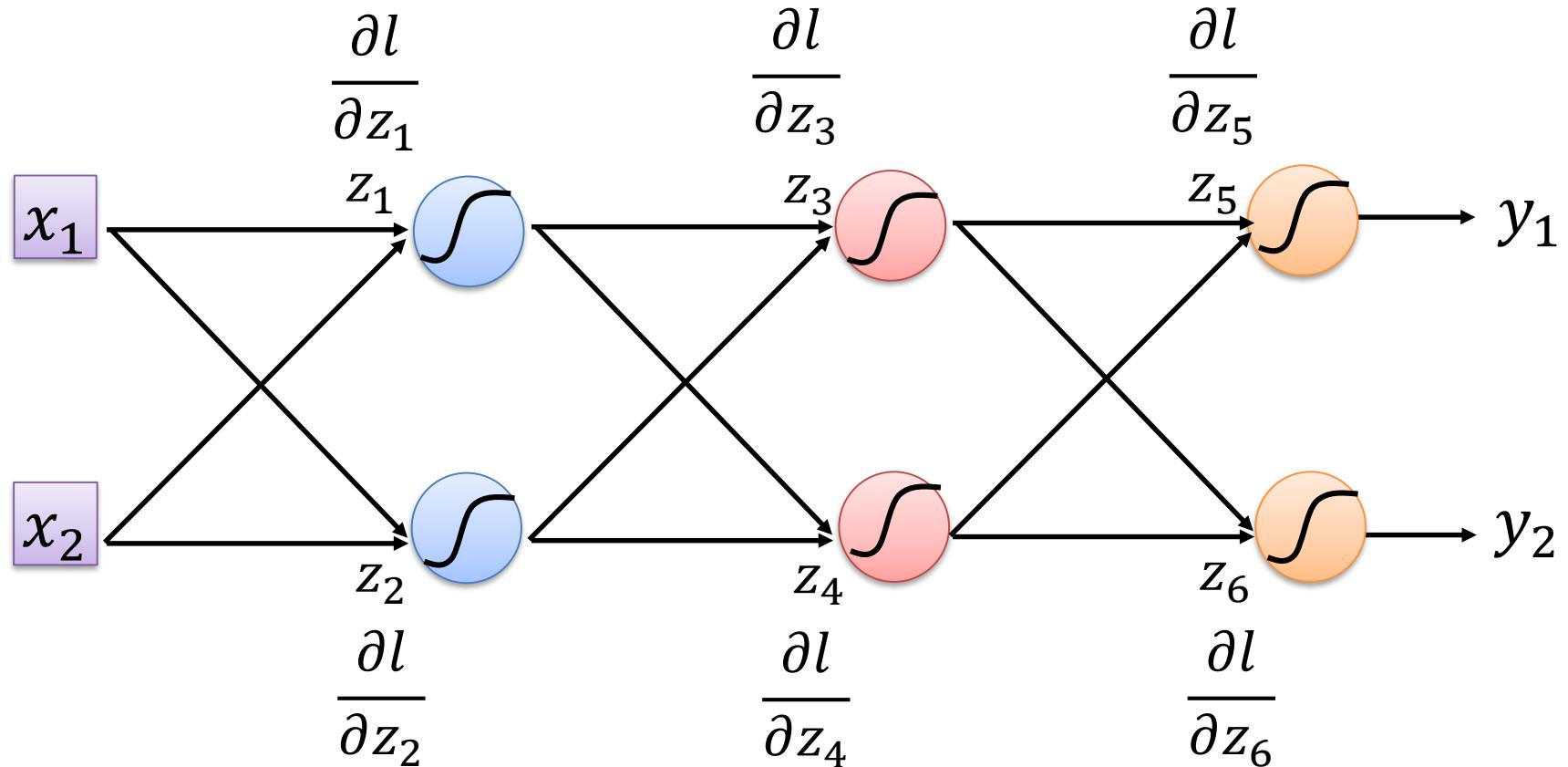
Case 2. Not Output Layer



Backpropagation – Backward Pass

Compute $\frac{\partial l}{\partial z}$ for all activation function inputs z

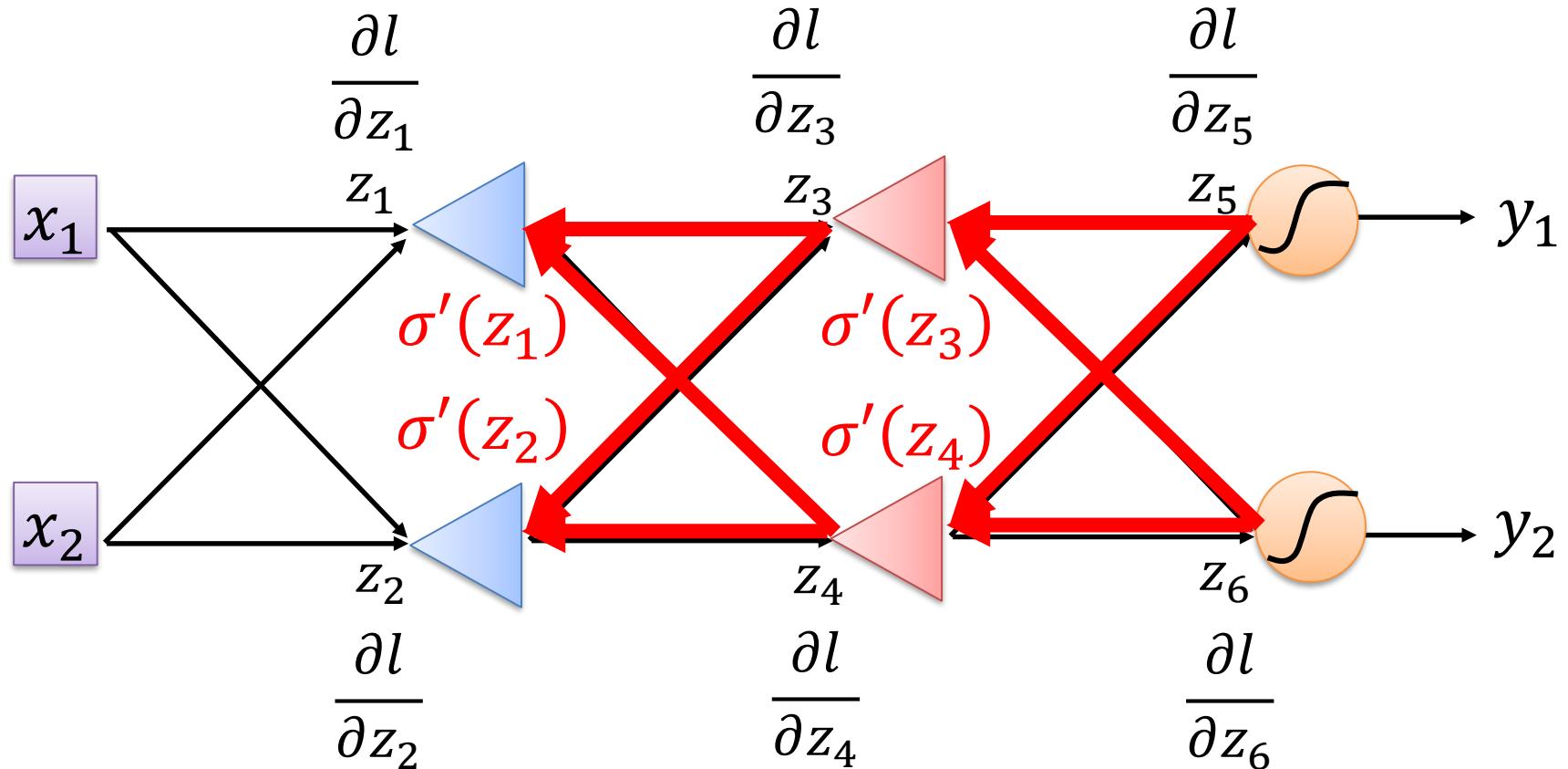
Compute $\frac{\partial l}{\partial z}$ from the output layer



Backpropagation – Backward Pass

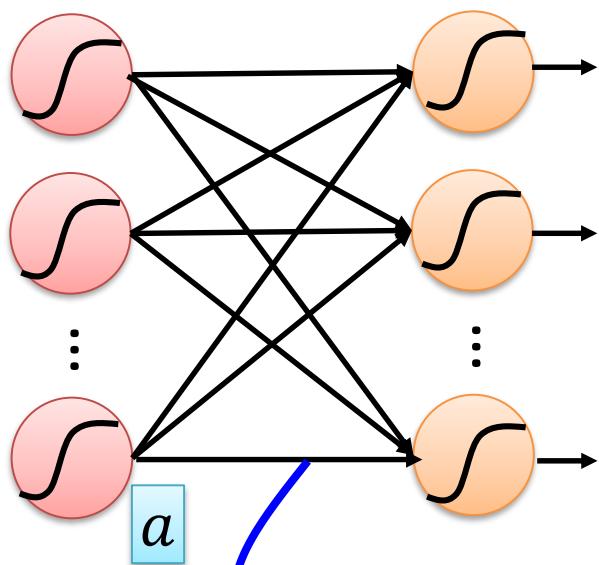
Compute $\partial l / \partial z$ for all activation function inputs z

Compute $\partial l / \partial z$ from the output layer



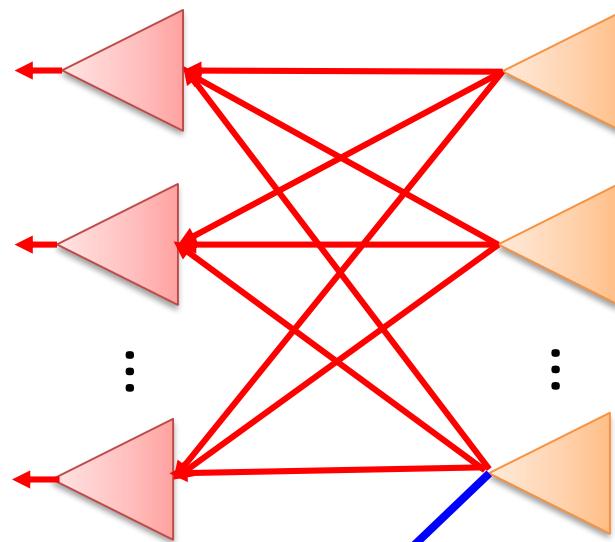
Backpropagation – Summary

Forward Pass



$$\frac{\partial z}{\partial w} = a$$

Backward Pass



$$X \quad \frac{\partial l}{\partial z} = \frac{\partial l}{\partial w}$$

for all w

链式求导：反向传播算法(Backpropagation)

- 输入数据。

```
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()  
train_images = train_images.reshape((60000, 28 * 28))  
train_images = train_images.astype('float32') / 255  
test_images = test_images.reshape((10000, 28 * 28))  
test_images = test_images.astype('float32') / 255
```

- 构建网络。

```
network = models.Sequential()  
network.add(layers.Dense(512, activation='relu', input_shape=(28 * 28,)))  
network.add(layers.Dense(10, activation='softmax'))
```

- 网络的编译。

```
network.compile(optimizer='rmsprop',  
loss='categorical_crossentropy',  
metrics=['accuracy'])
```

- 训练循环。

```
network.fit(train_images, train_labels, epochs=5, batch_size=128)
```

- 调用fit时发生了什么？

- 网络开始在训练数据上进行迭代（每个小批量包含128个样本），共迭代5次〔在所有训练数据上迭代一次叫作一个轮次（epoch）〕。在每次迭代过程中，网络会计算批量损失相对于权重的梯度，并相应地更新权重。5轮之后，网络进行了2345次梯度更新（每轮469次），网络损失值将变得足够小，使得网络能够以很高的精度对手写数字进行分类。

输入图像保存在float32格式的Numpy张量中，形状分别为(60000,784)（训练数据）和(10000, 784)（测试数据）

本章内容

1 什么是深度学习

2 神经网络的数学基础

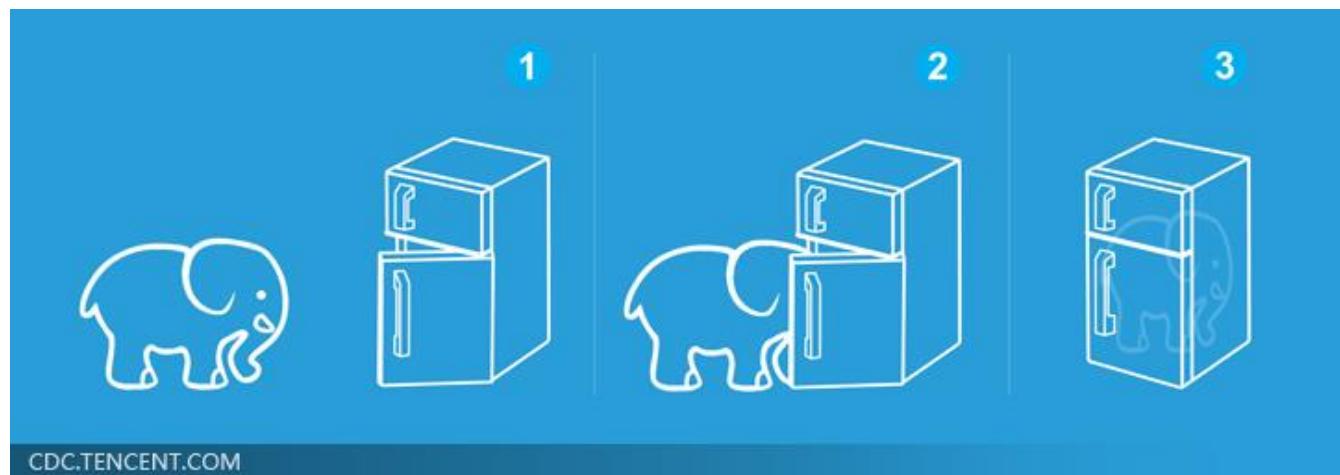
3 神经网络入门

4 神经网络的通用工作流程

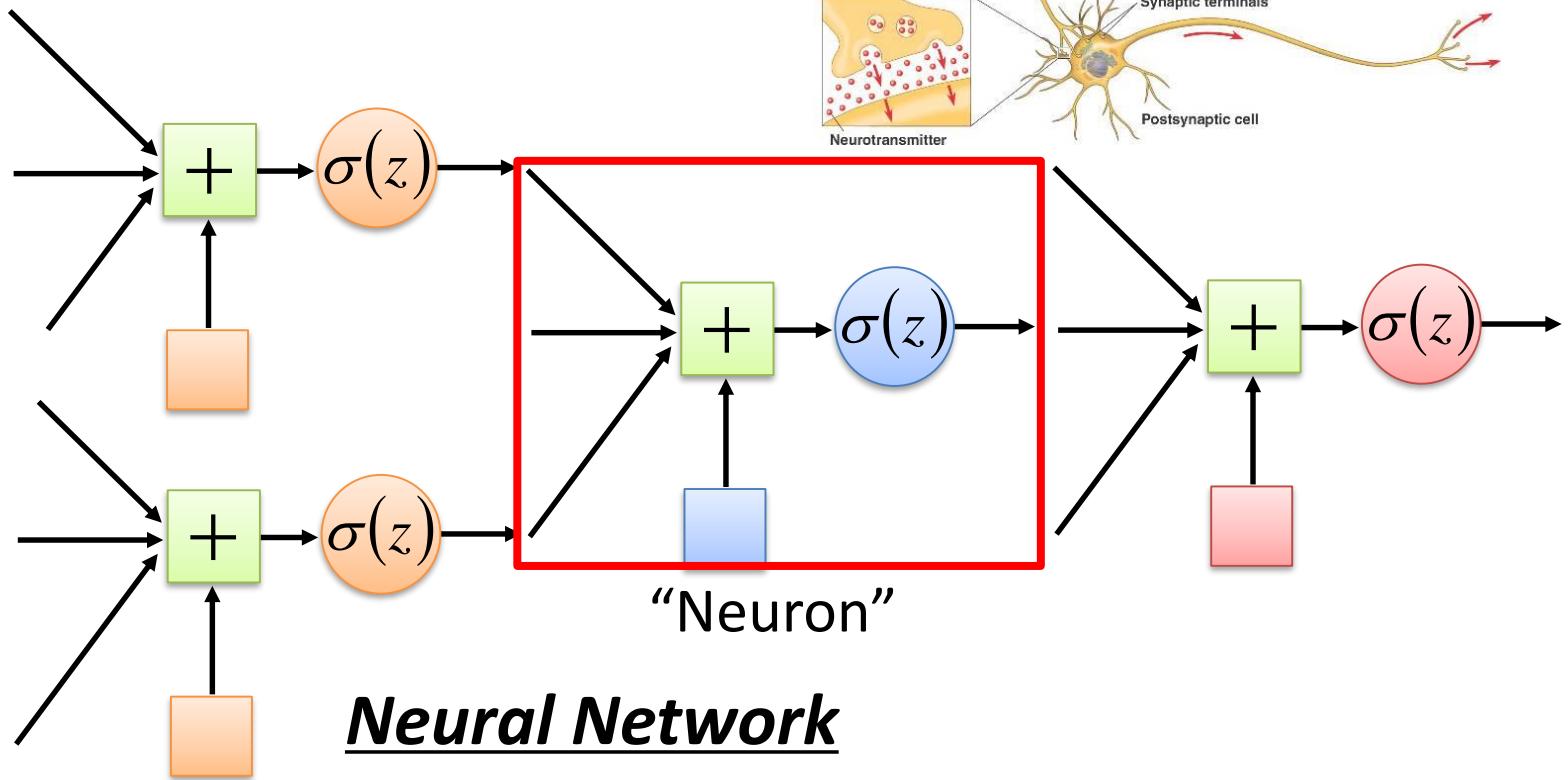
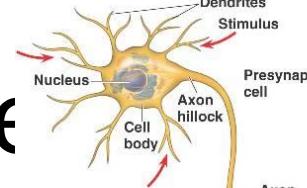
Three Steps for Deep Learning



Deep Learning is so simple



Neural Ne

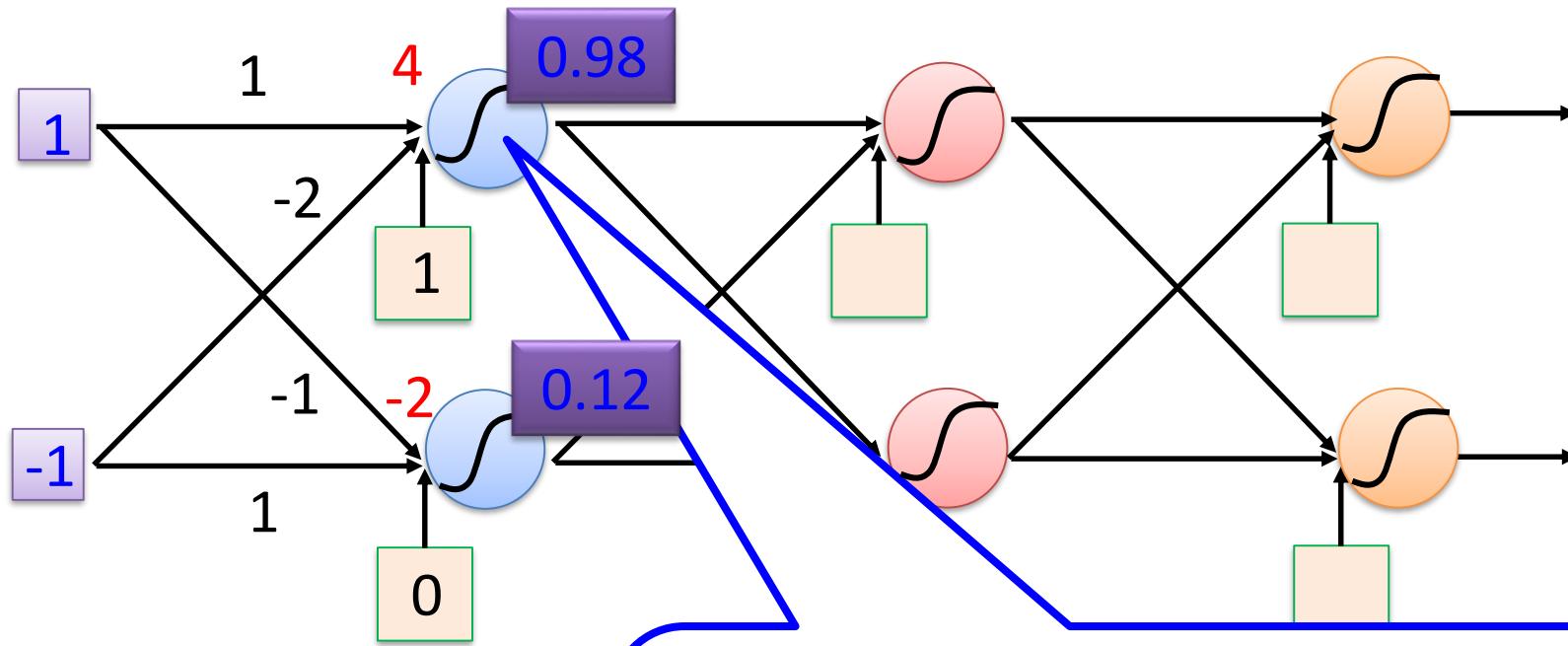


Neural Network

Different connection leads to different network structures

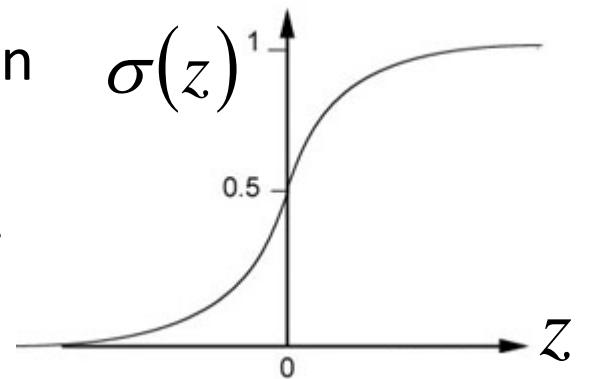
Network parameter θ : all the weights and biases in the “neurons”

Fully Connect Feedforward Network

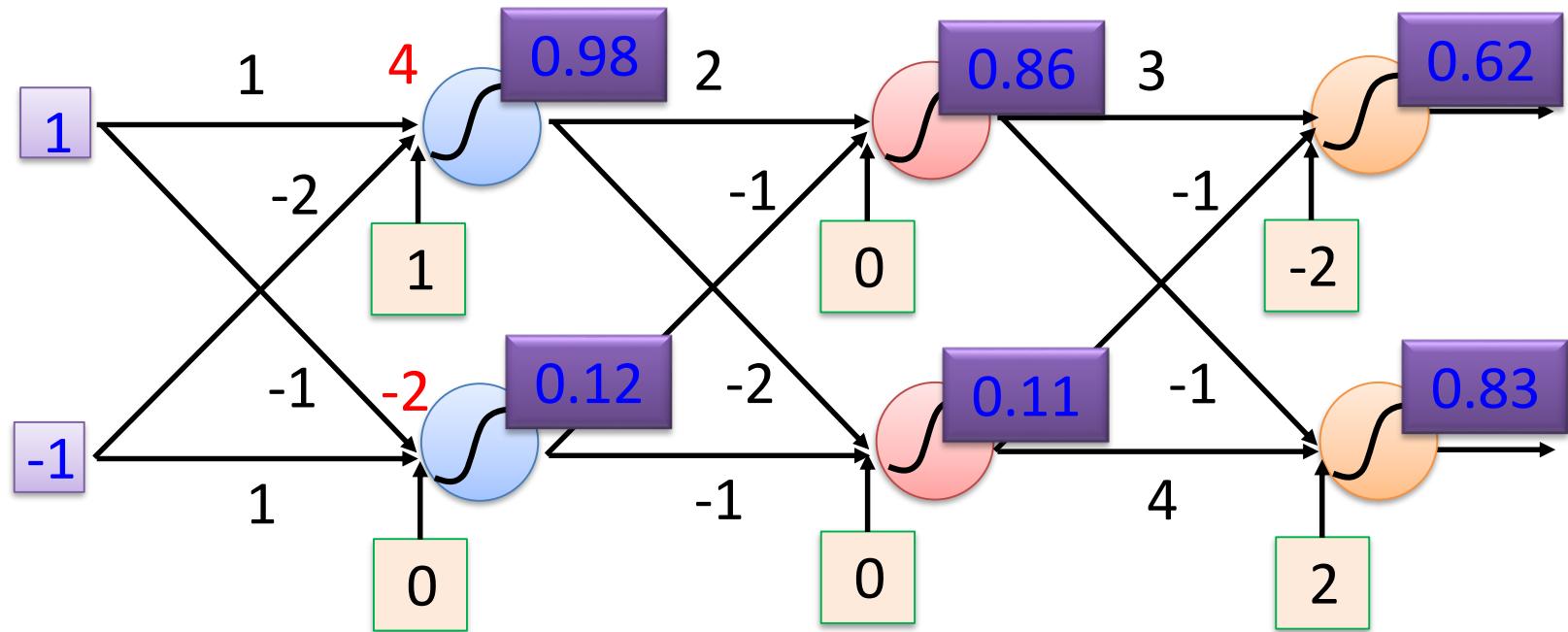


Sigmoid Function

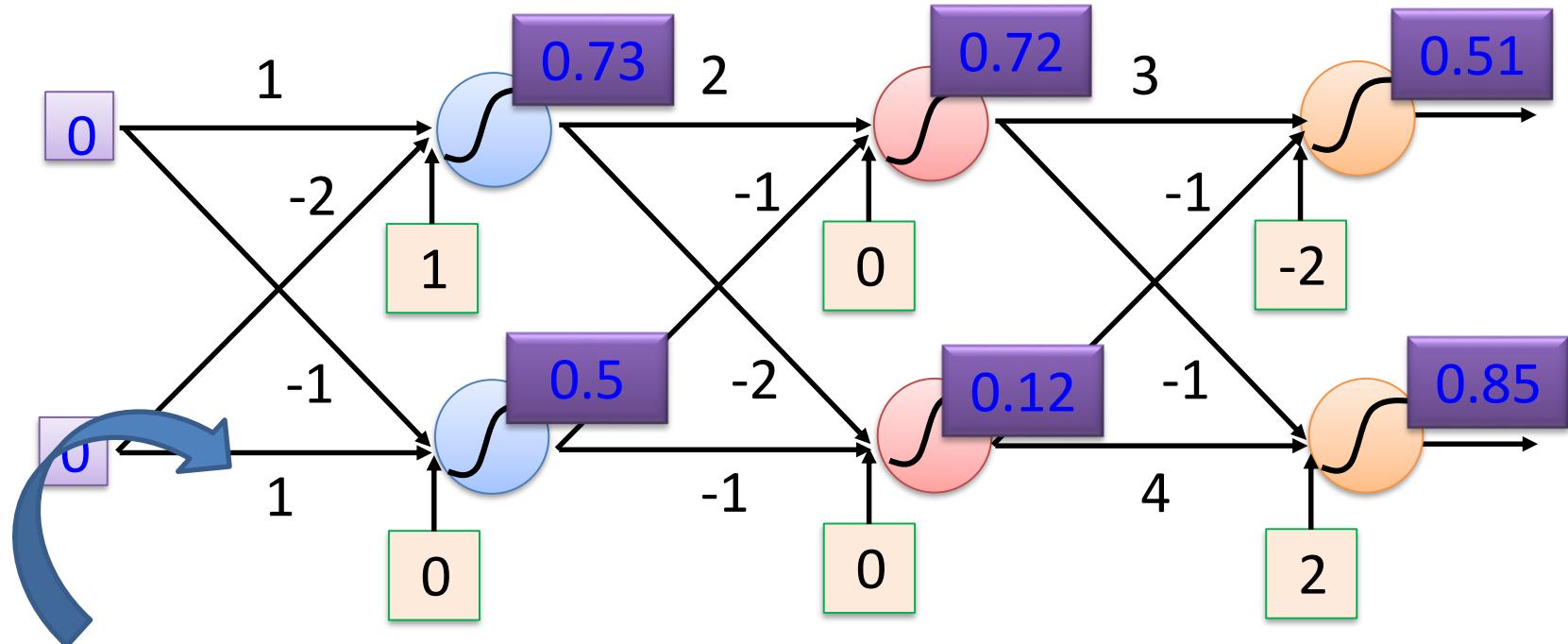
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$



Fully Connect Feedforward Network



Fully Connect Feedforward Network

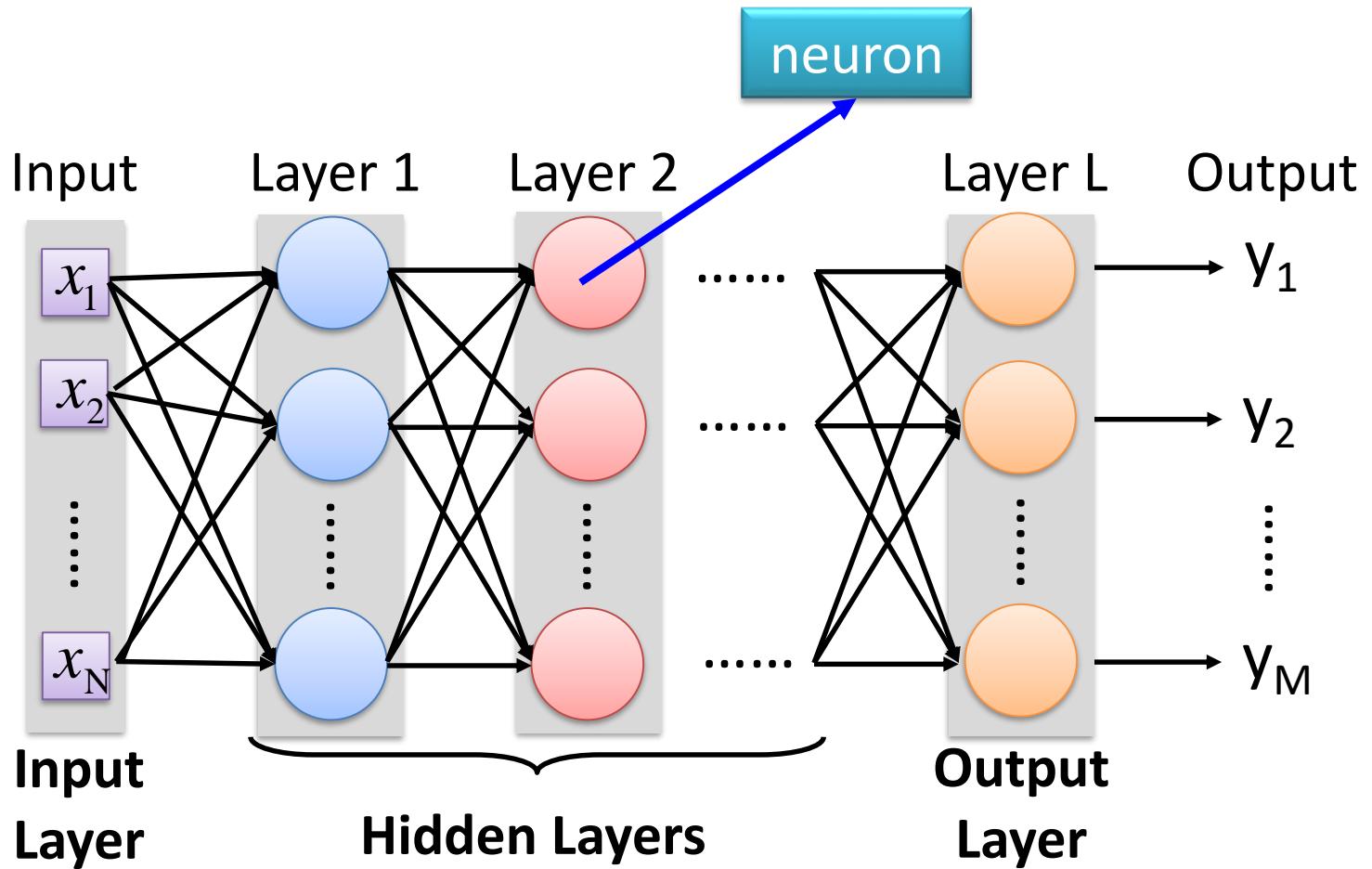


This is a function.
Input vector, output vector

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

Given network structure, define a function set

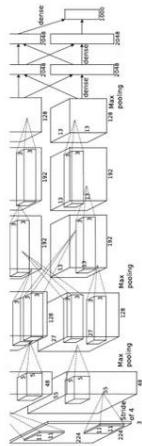
Fully Connect Feedforward Network



Deep = Many hidden layers

http://cs231n.stanford.edu/slides/winter1516_lecuture8.pdf

16.4%



AlexNet (2012)

8 layers

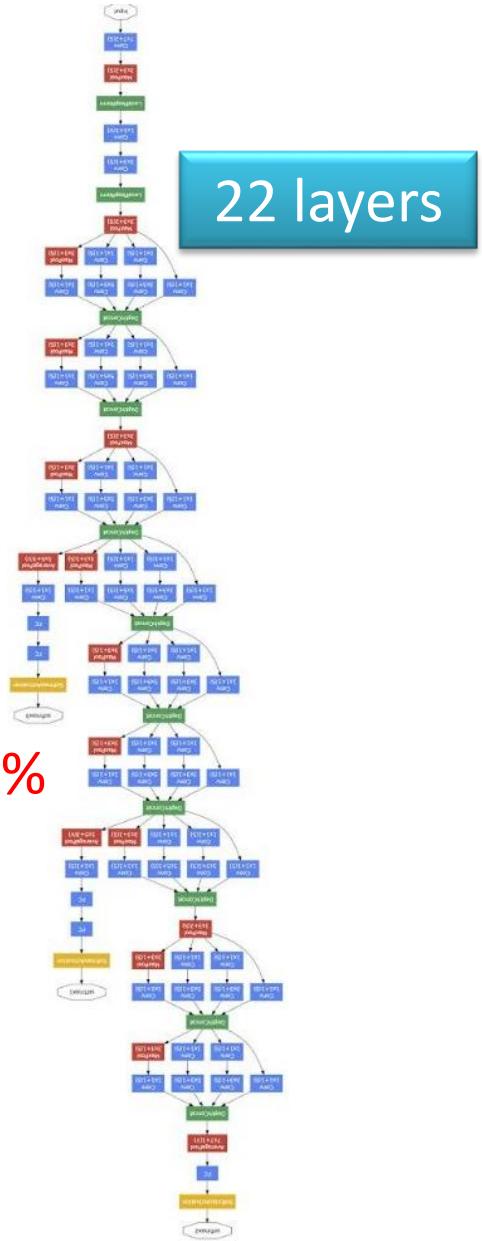
7.3%



VGG (2014)

19 layers

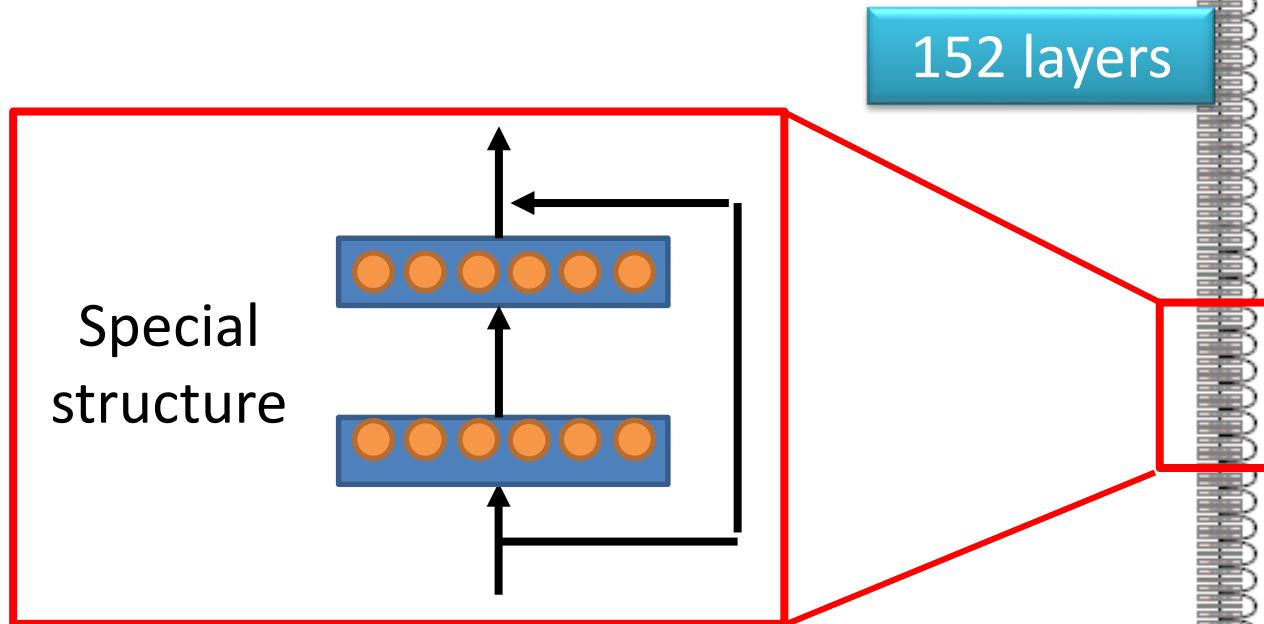
6.7%



GoogleNet (2014)

22 layers

Deep = Many hidden layers



Ref:

<https://www.youtube.com/watch?v=dxB6299gpvl>

3.57%

16.4%

7.3%

6.7%

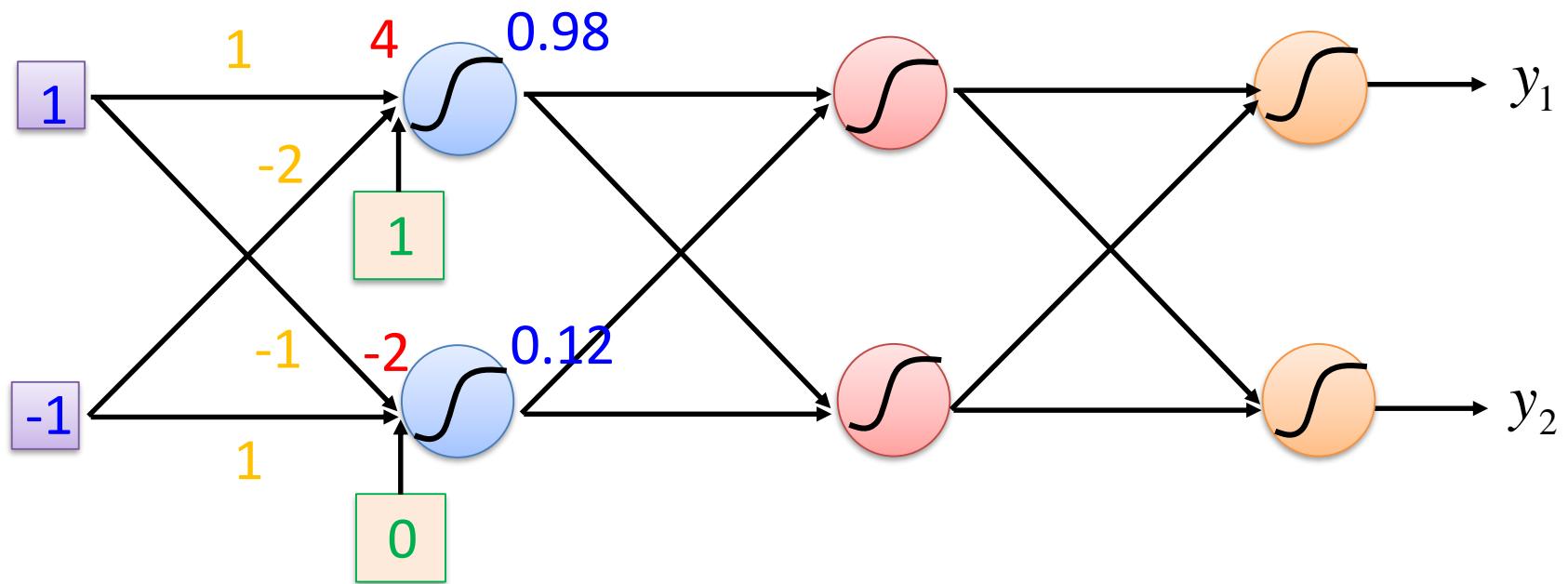
AlexNet
(2012)

VGG
(2014)

GoogleNet
(2014)

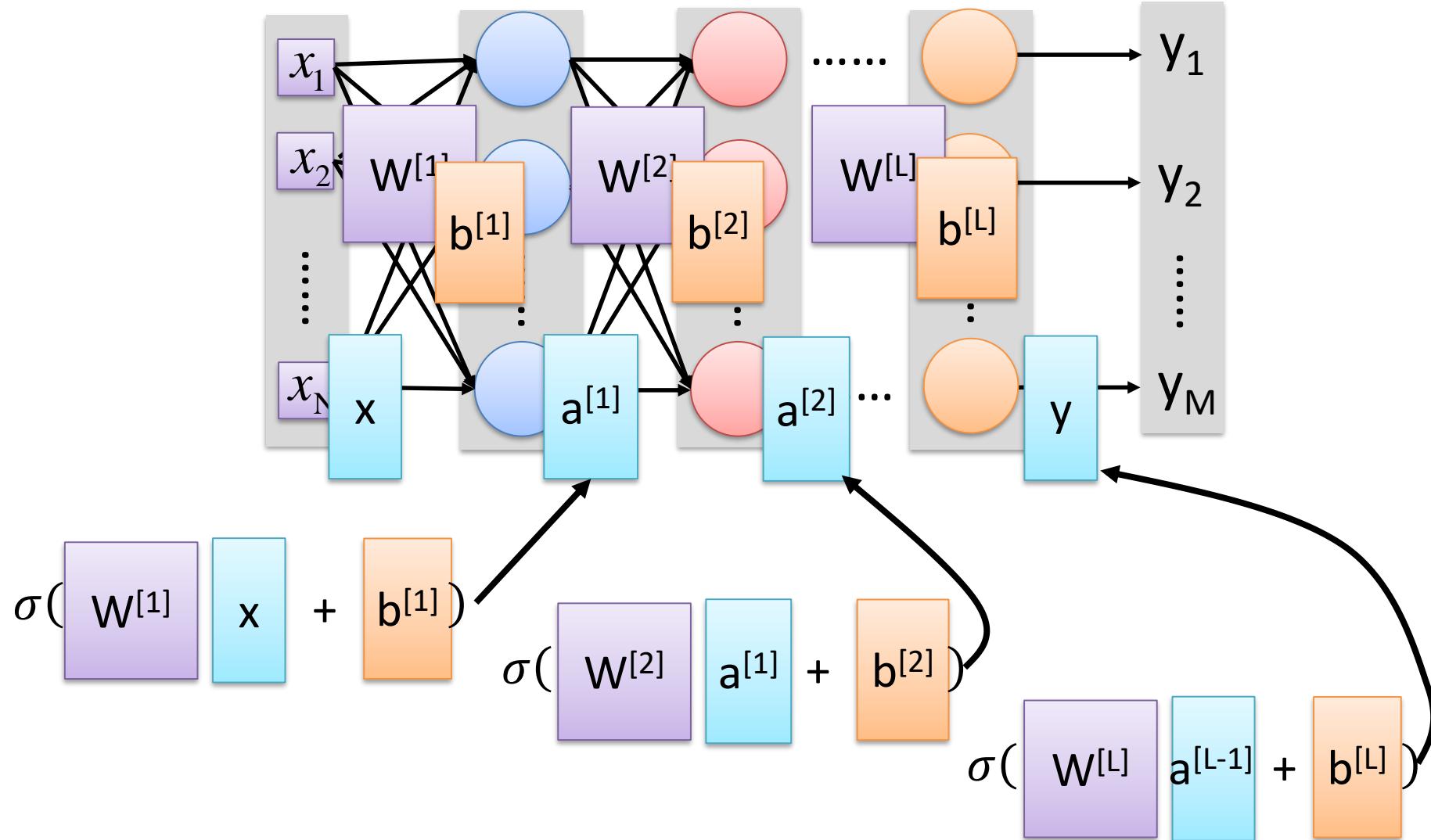
Residual Net
(2015)

Matrix Operation

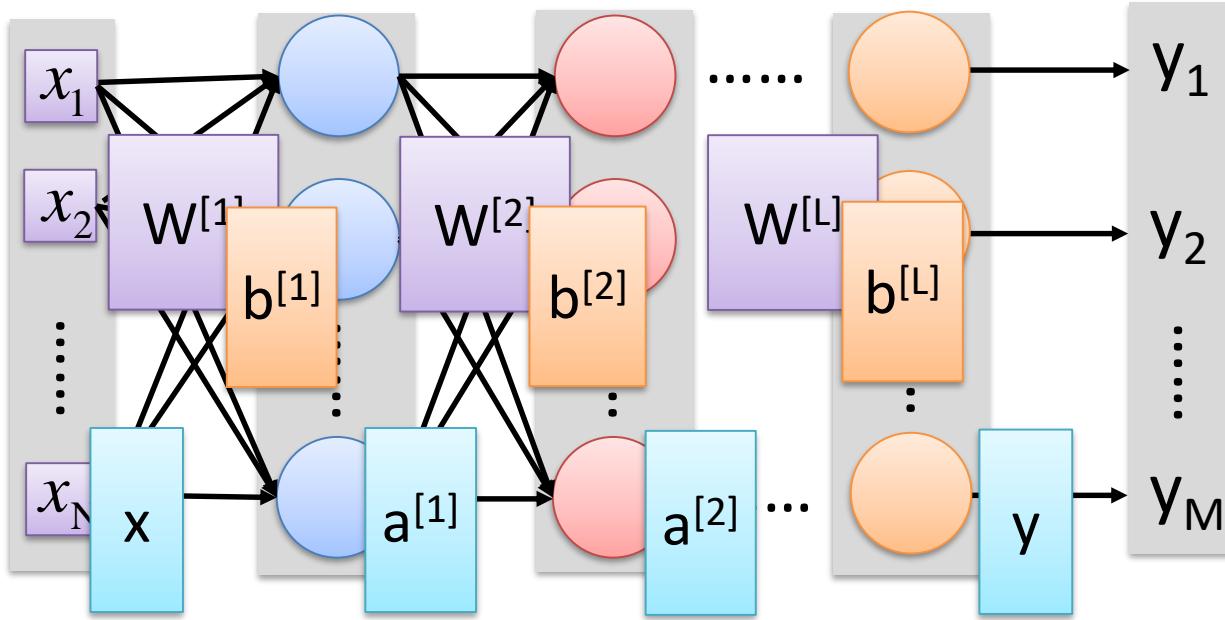


$$\sigma \left(\underbrace{\begin{bmatrix} 1 & -2 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix}}_{\begin{bmatrix} 4 \\ -2 \end{bmatrix}} \right) = \begin{bmatrix} 0.98 \\ 0.12 \end{bmatrix}$$

Neural Network



Neural Network



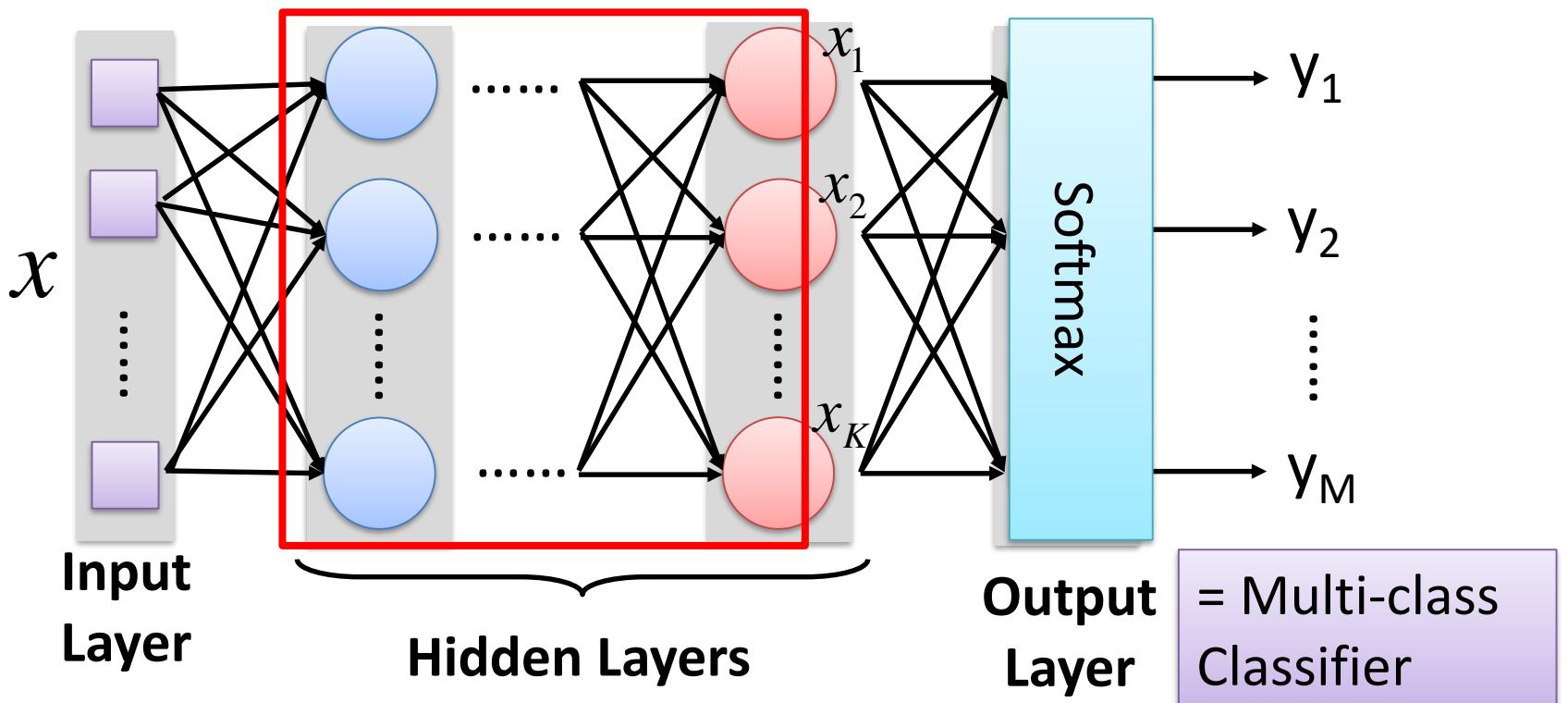
$$y = f(x)$$

Using parallel computing techniques
to speed up matrix operation

$$= \sigma(W^{[L]} \cdots \sigma(W^{[2]} \sigma(W^{[1]} x + b^{[1]}) + b^{[2]}) \cdots + b^{[L]})$$

Output Layer as Multi-Class Classifier

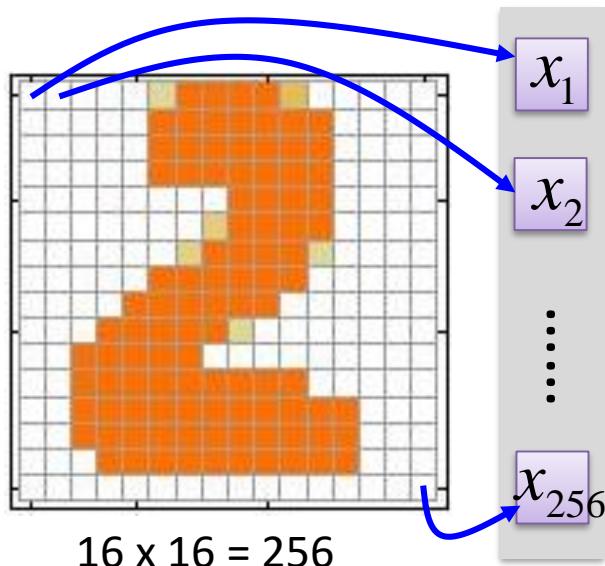
Feature extractor replacing
feature engineering



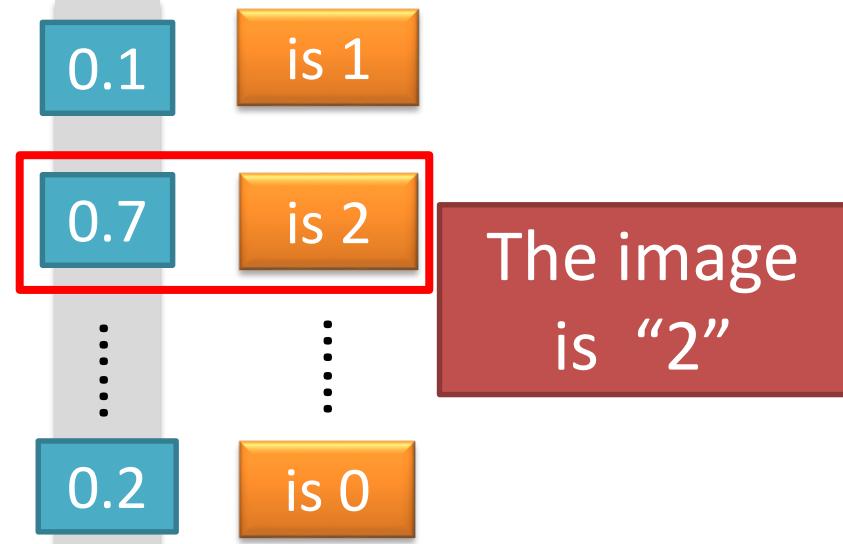
Example Application



Input



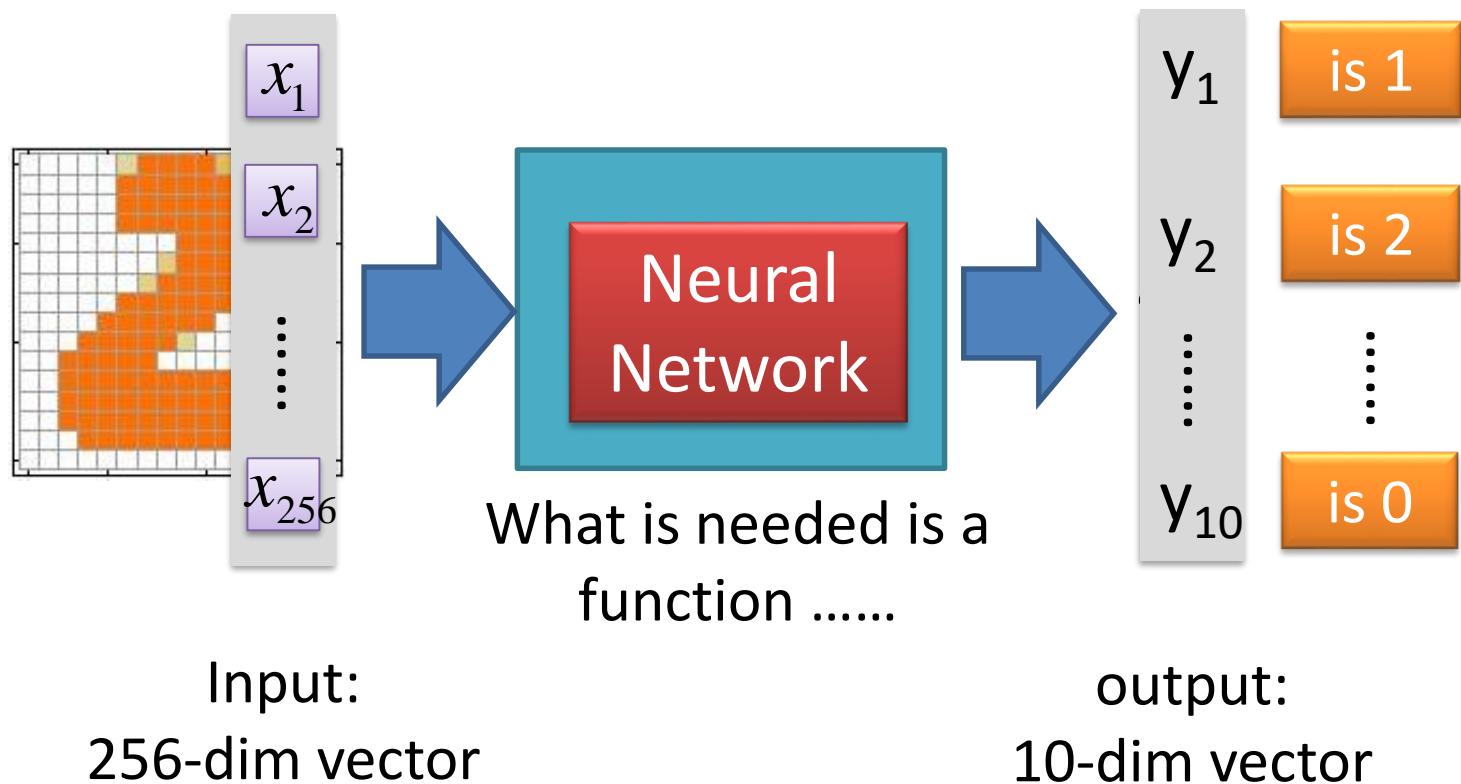
Output



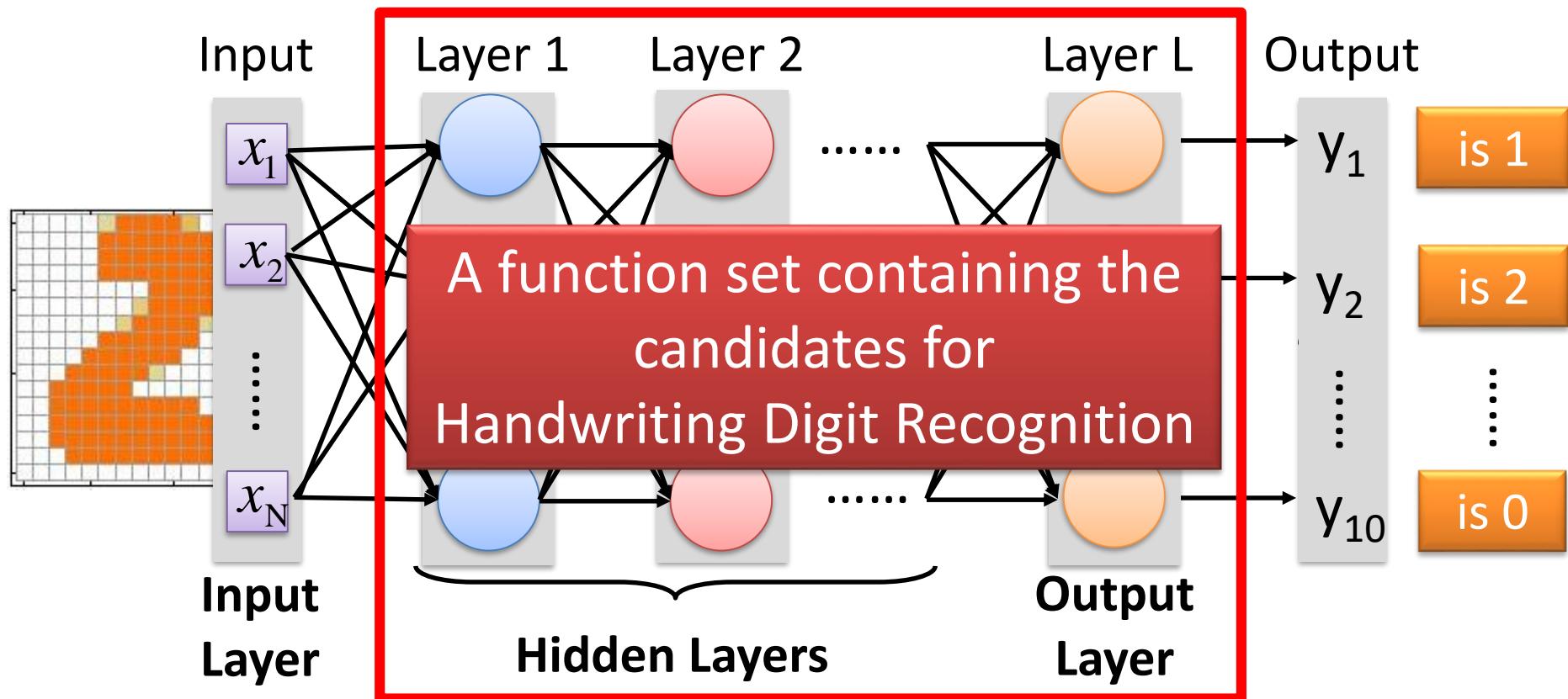
Each dimension represents the confidence of a digit.

Example Application

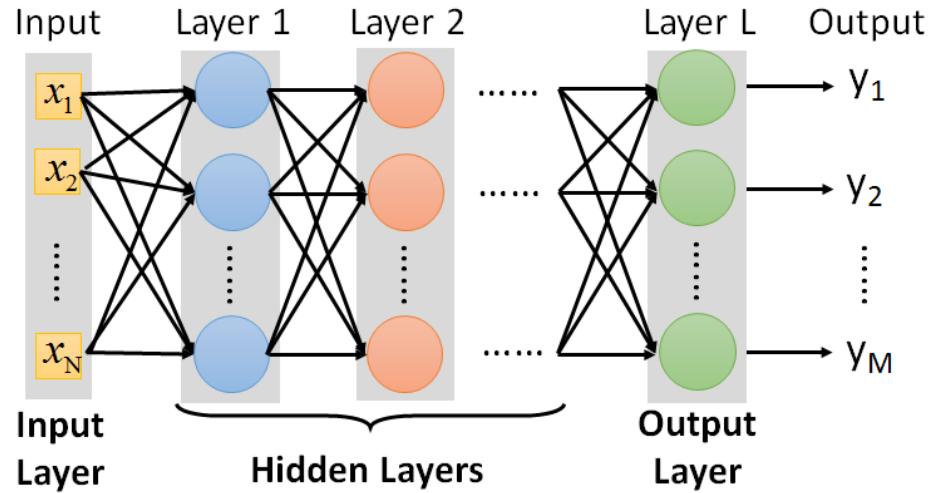
- Handwriting Digit Recognition



Example Application



You need to decide the network structure to let a good function in your function set.



- Q: How many layers? How many neurons for each layer?

Trial and Error

+

Intuition

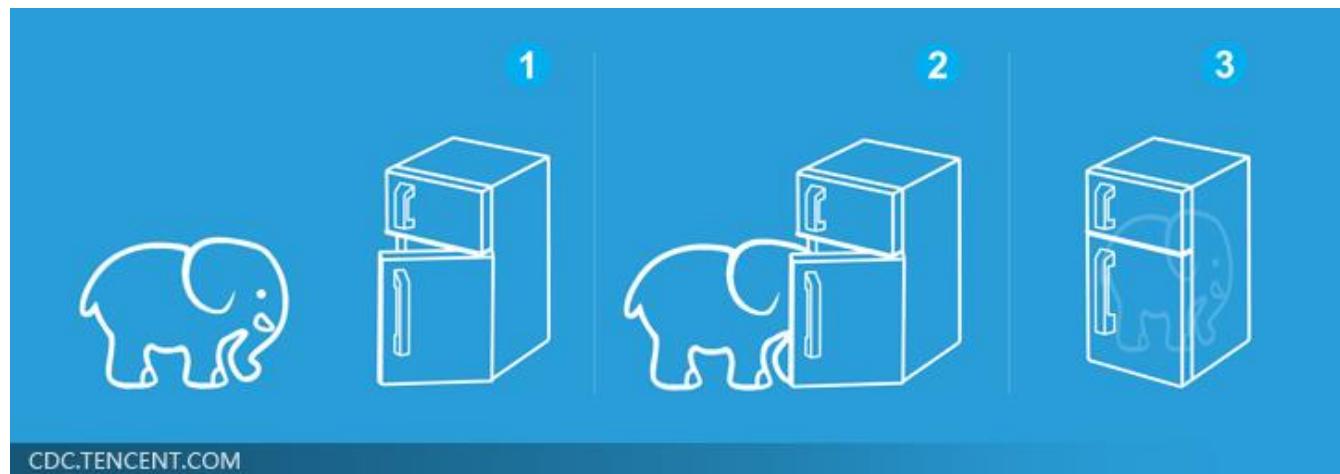
- Q: Can the structure be automatically determined?
 - E.g. Evolutionary Artificial Neural Networks
- Q: Can we design the network structure?

Convolutional Neural Network (CNN)

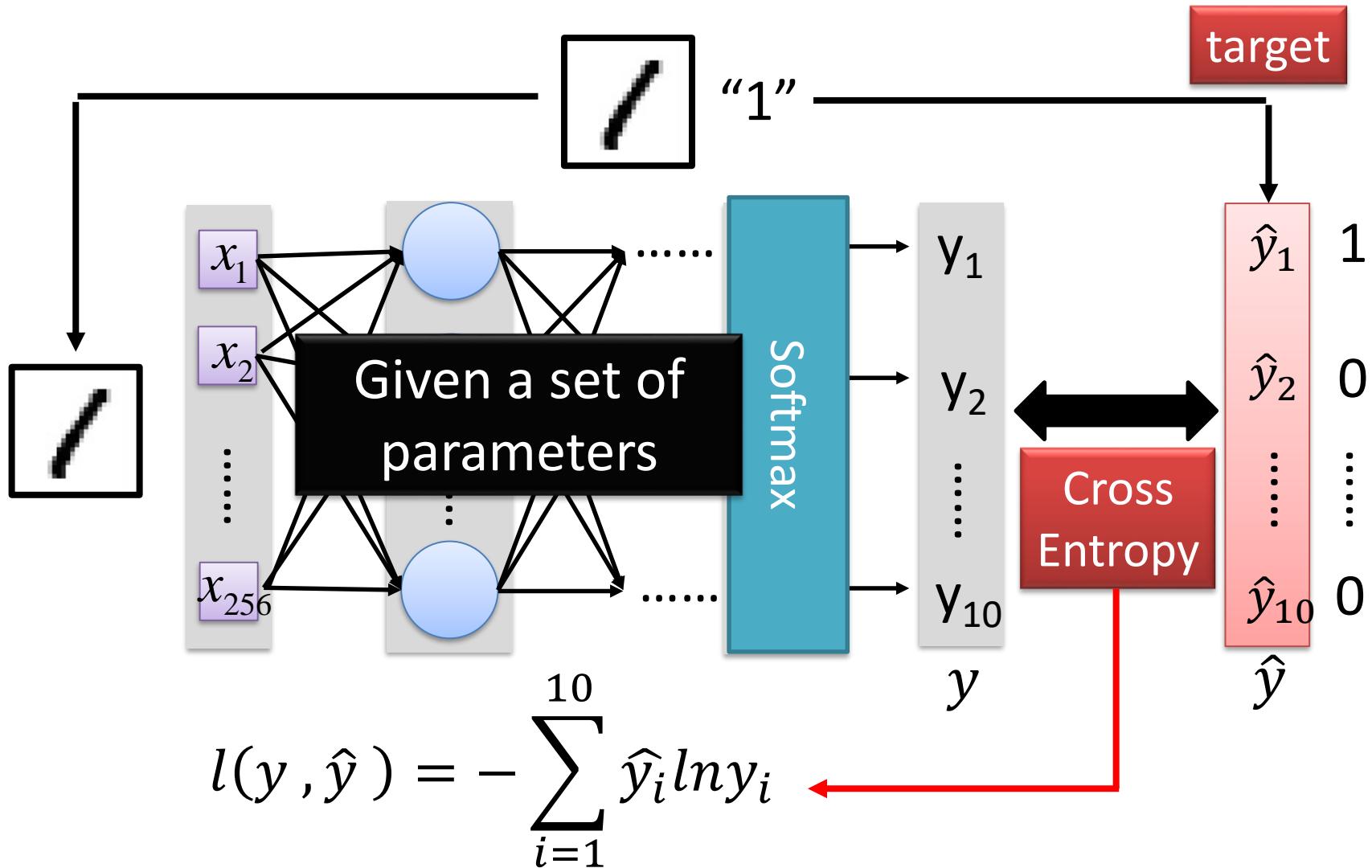
Three Steps for Deep Learning



Deep Learning is so simple

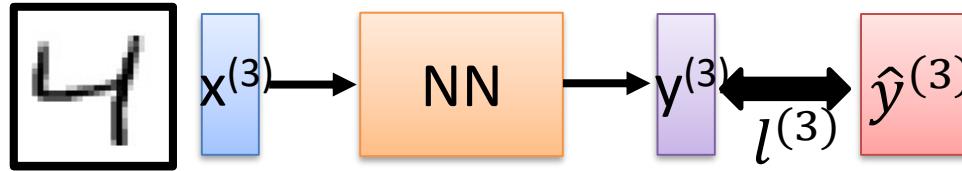
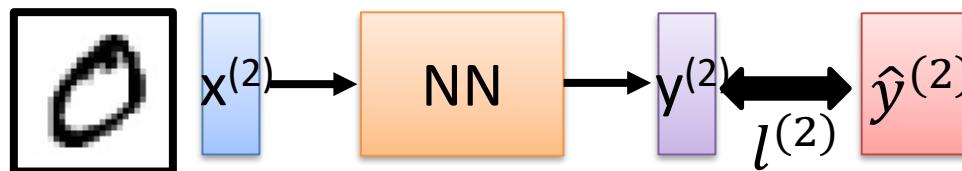
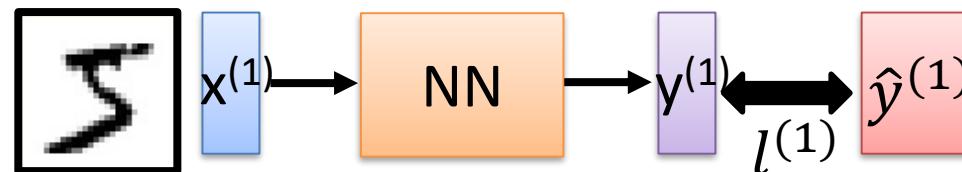


Loss for an Example



Total Loss

For all training data ...

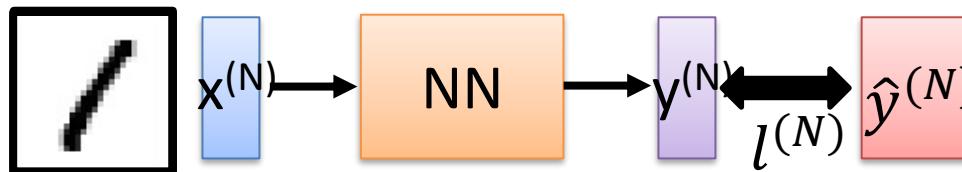


⋮

⋮

⋮

⋮



Total Loss:

$$L = \sum_{n=1}^N l^{(n)}$$



Find a function in function set that minimizes total loss L

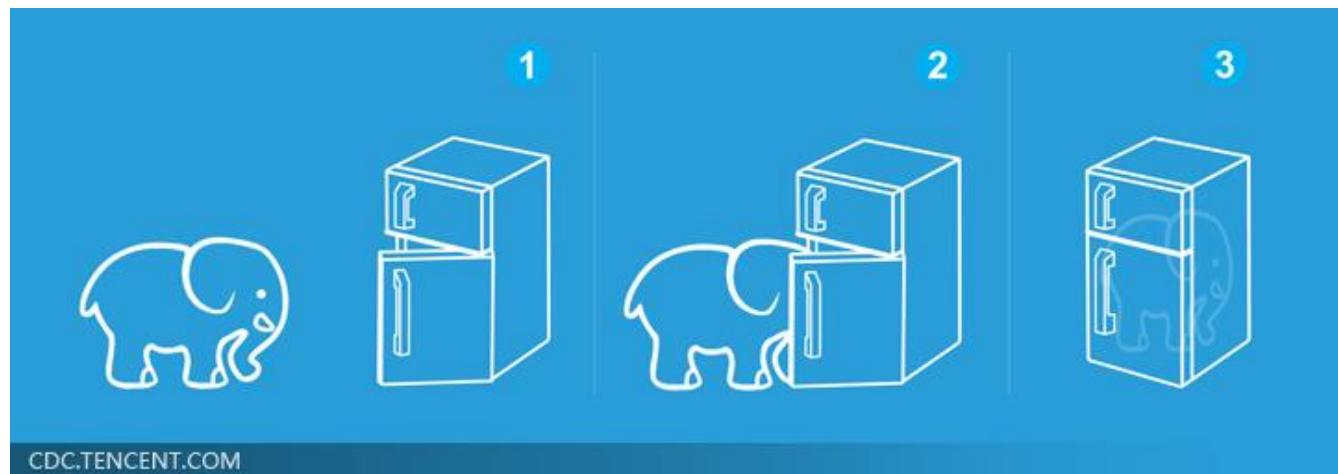


Find the network parameters θ^* that minimize total loss L

Three Steps for Deep Learning



Deep Learning is so simple



How to pick the best function

Find network parameters θ^* that minimize total loss L

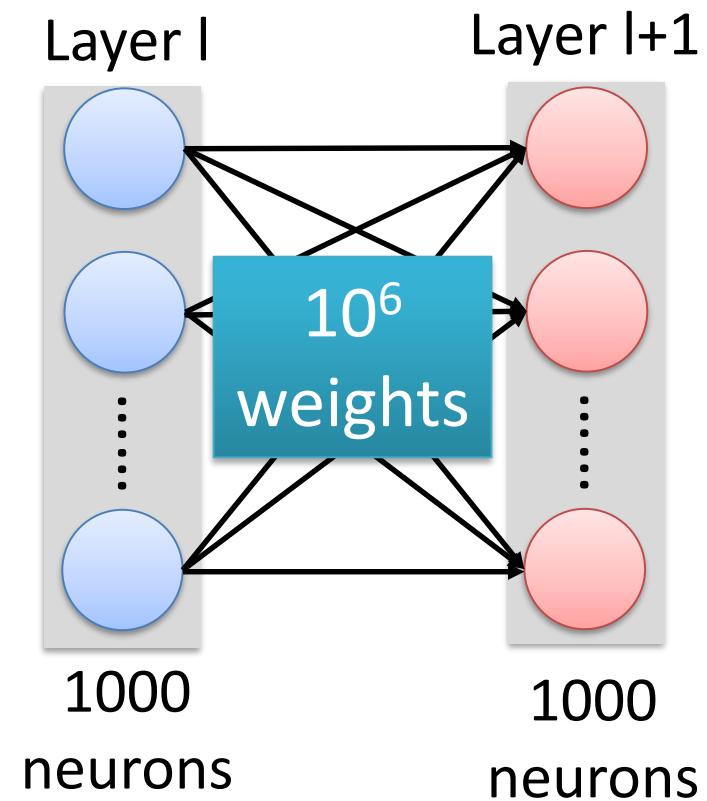
Enumerate all possible values

Network parameters

$$\theta = \underbrace{\{w_1, w_2, w_3, \dots, b_1, b_2, b_3, \dots\}}_{\text{Millions of parameters}}$$

Millions of parameters

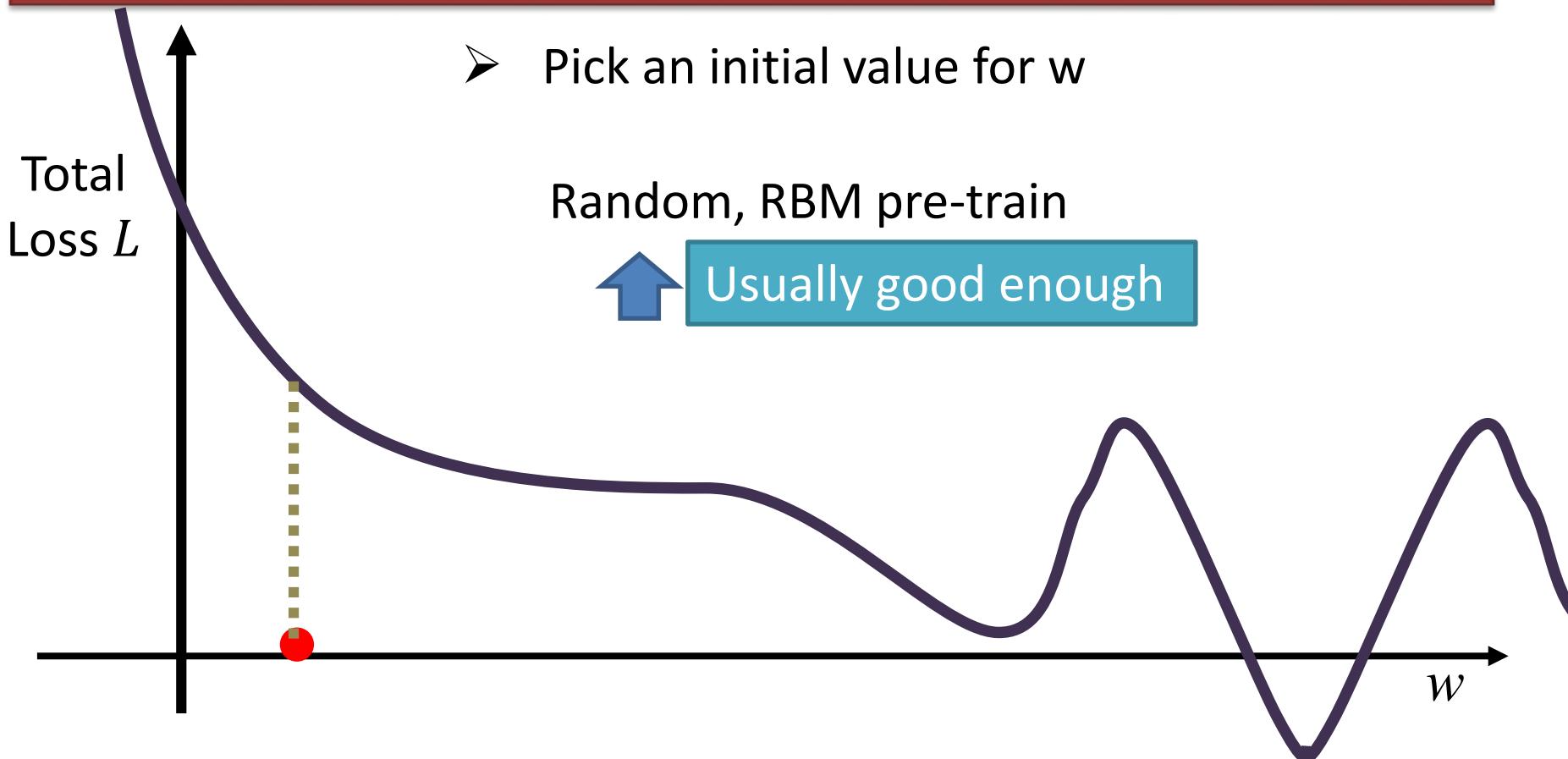
E.g. speech recognition: 8 layers and
1000 neurons each layer



Gradient Descent

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

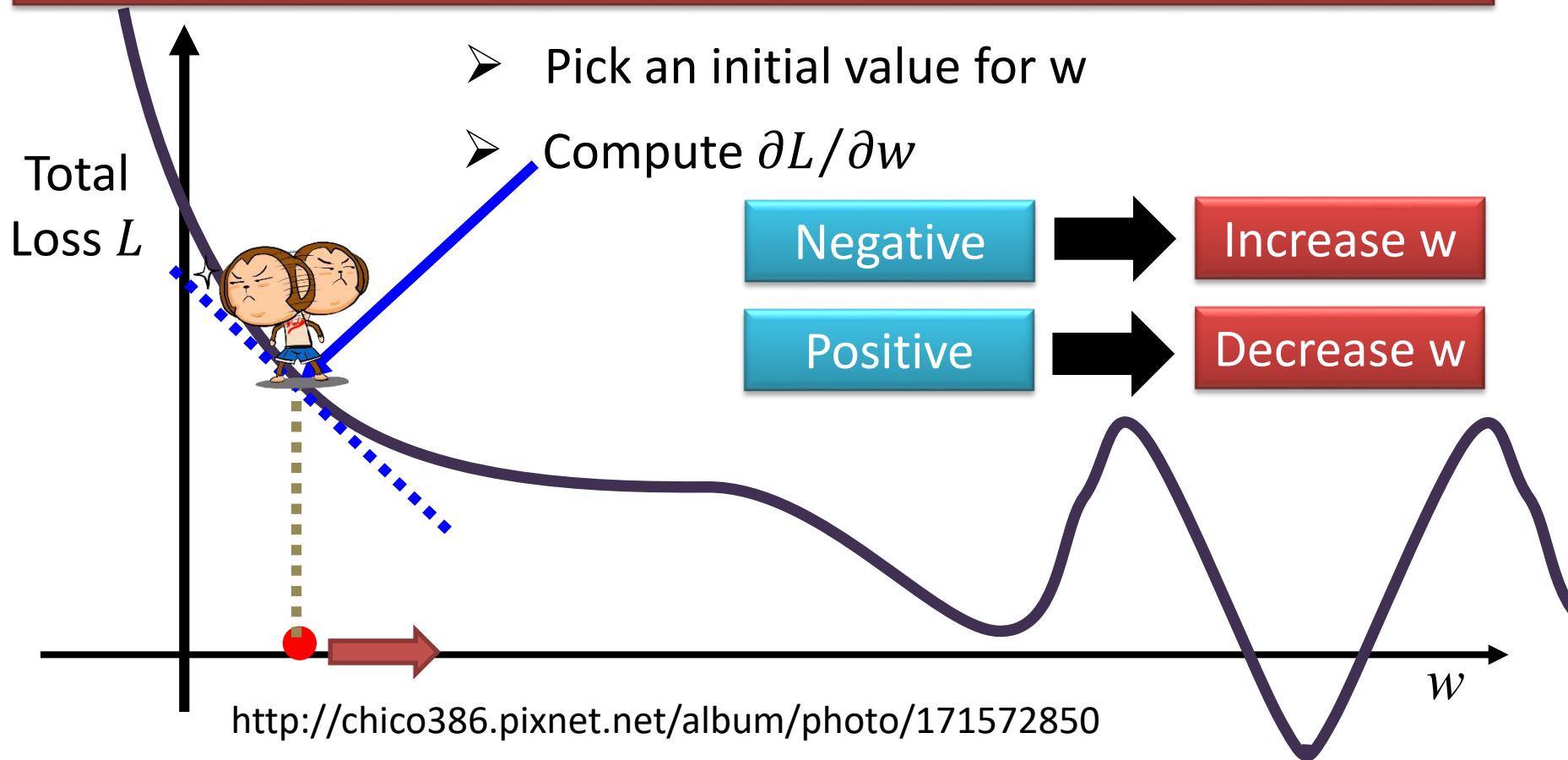
Find network parameters θ^* that minimize total loss L



Gradient Descent

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

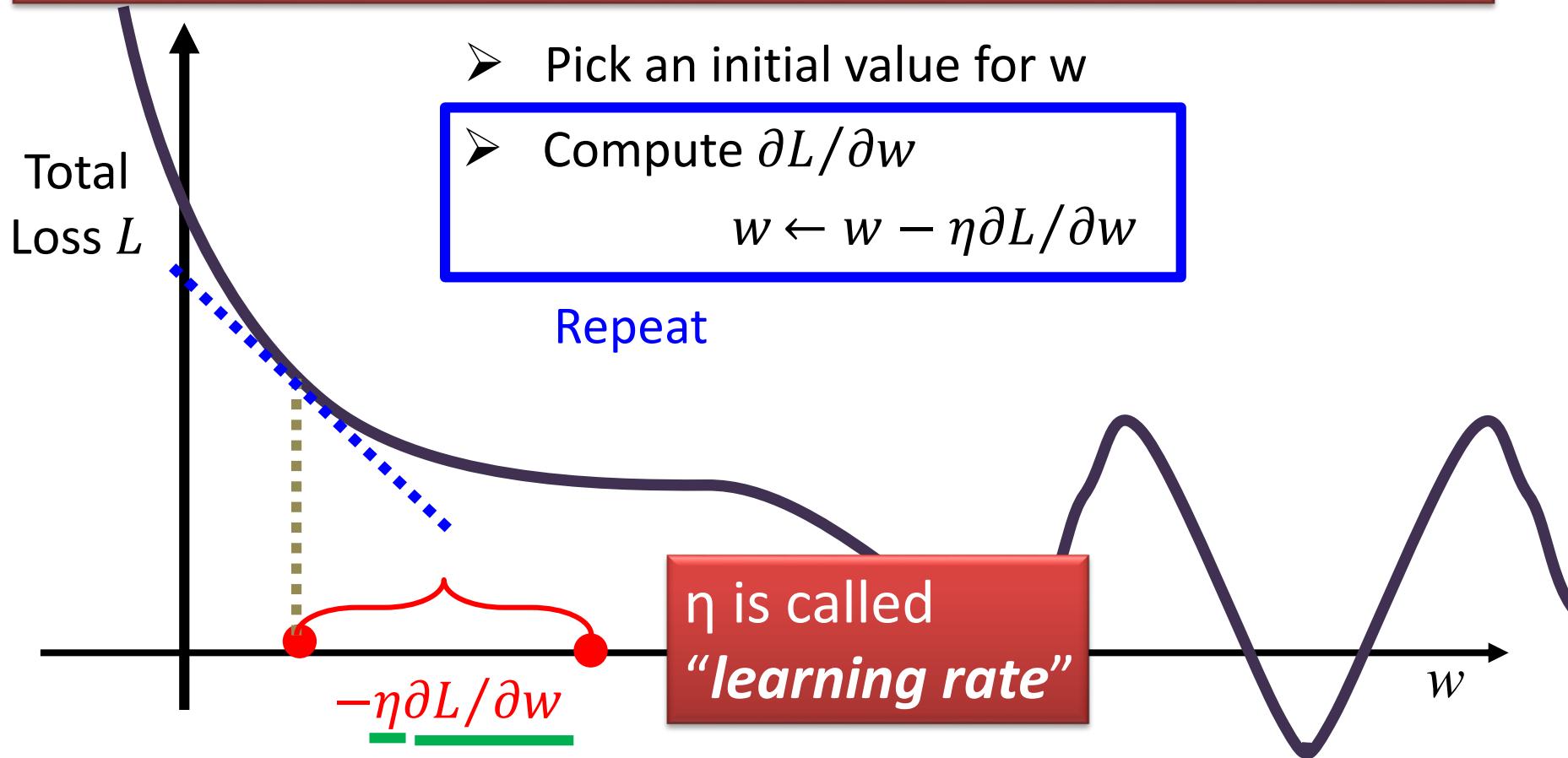
Find network parameters θ^* that minimize total loss L



Gradient Descent

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

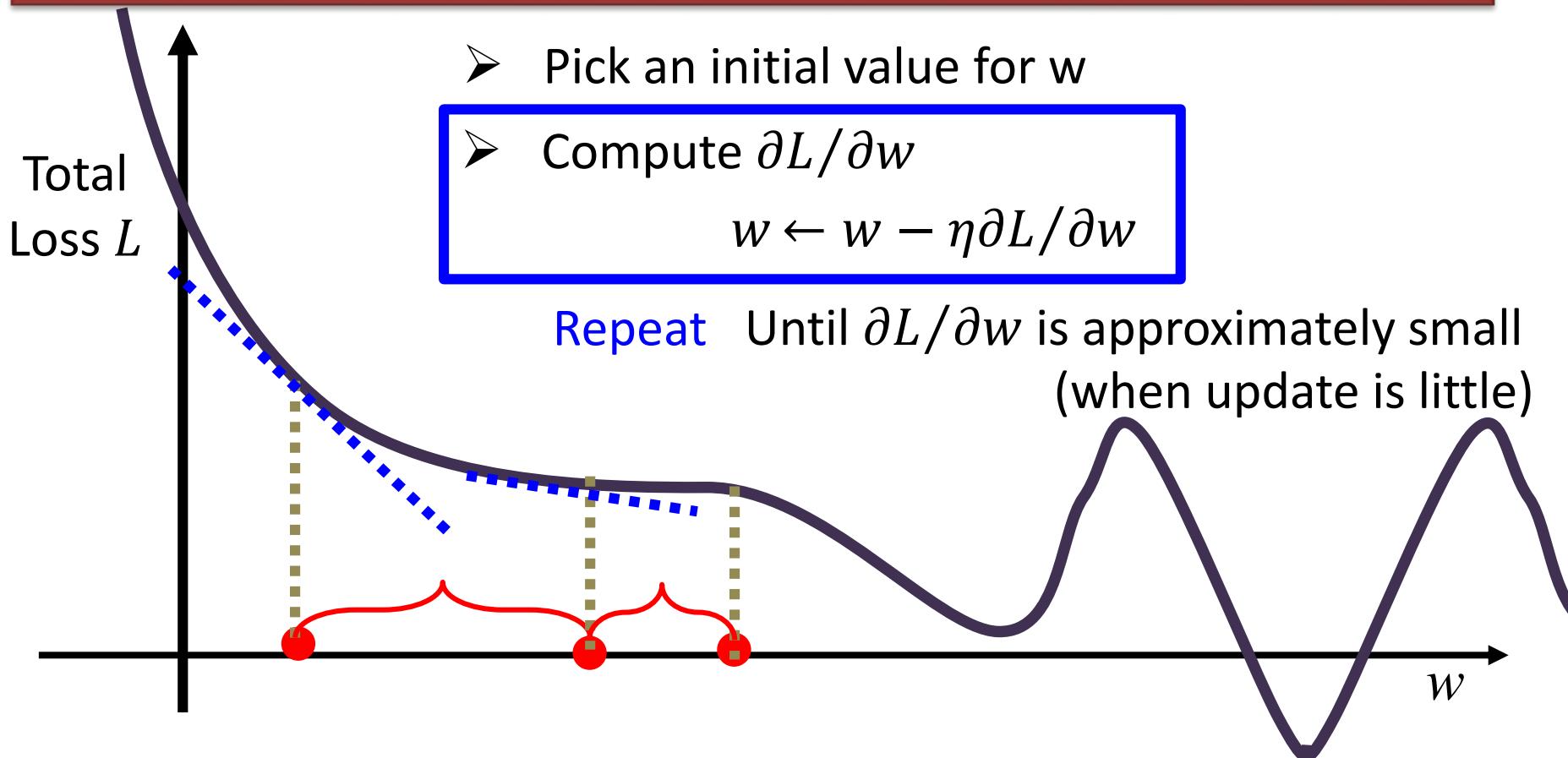
Find network parameters θ^* that minimize total loss L



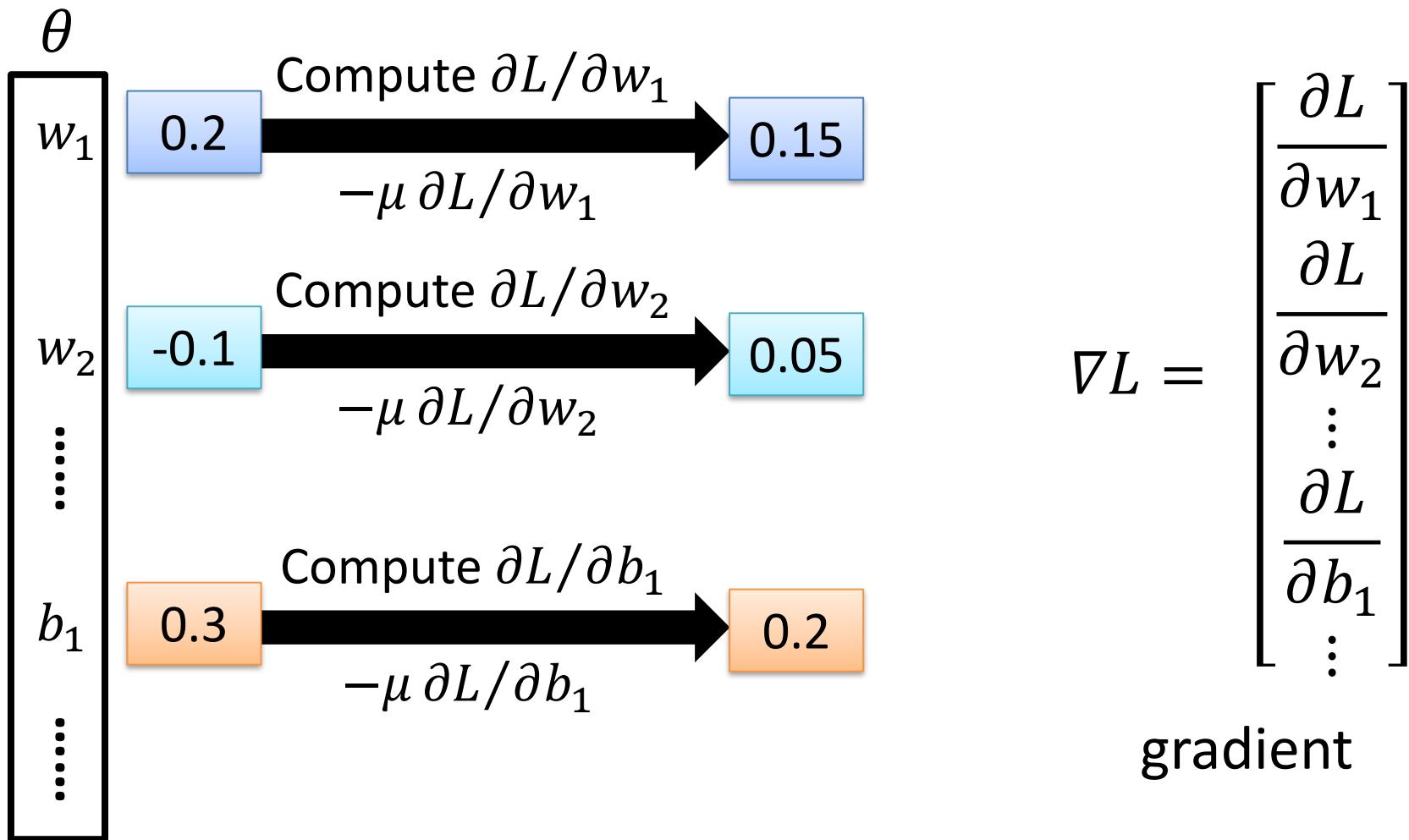
Gradient Descent

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

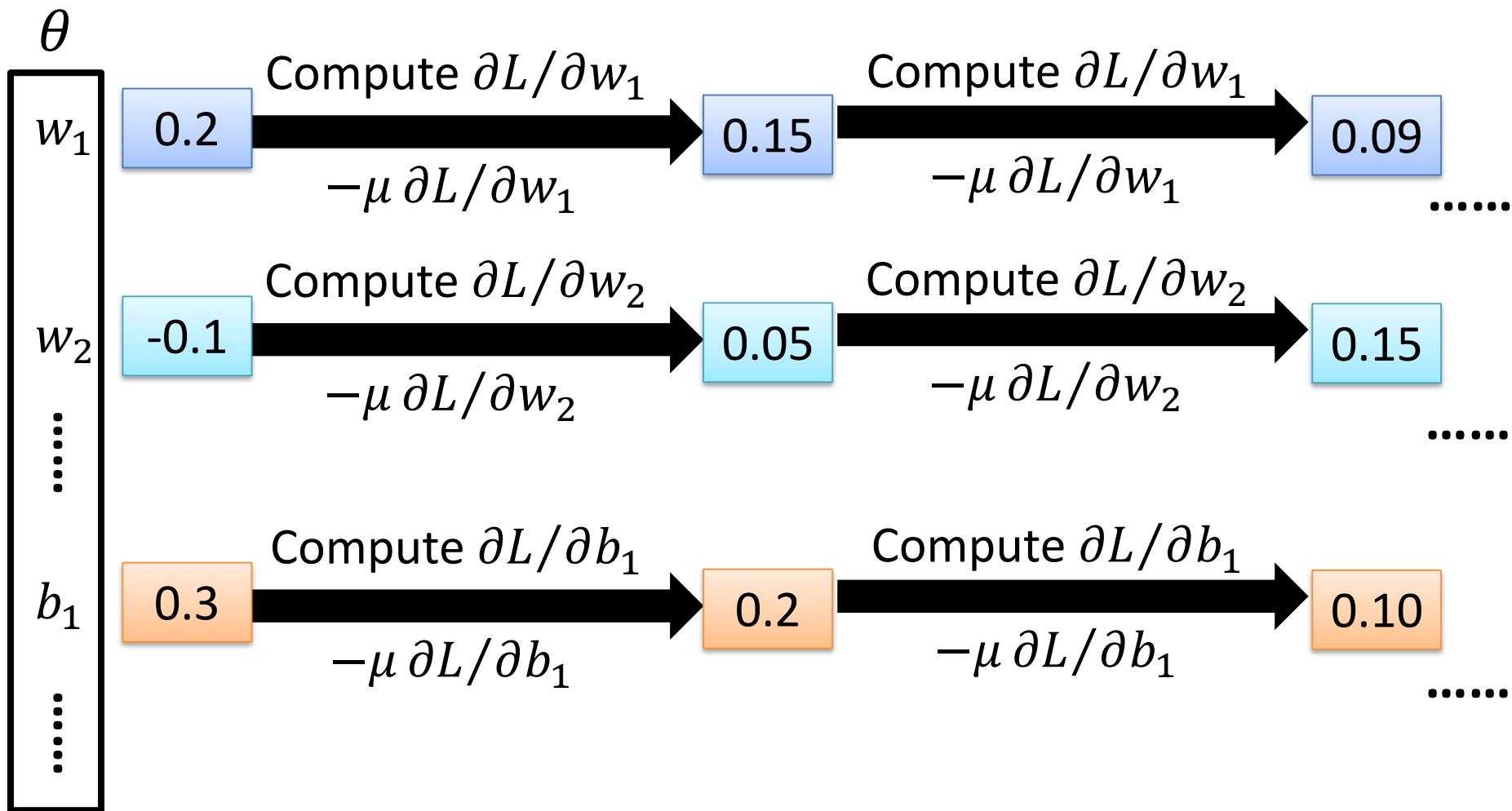
Find network parameters θ^* that minimize total loss L



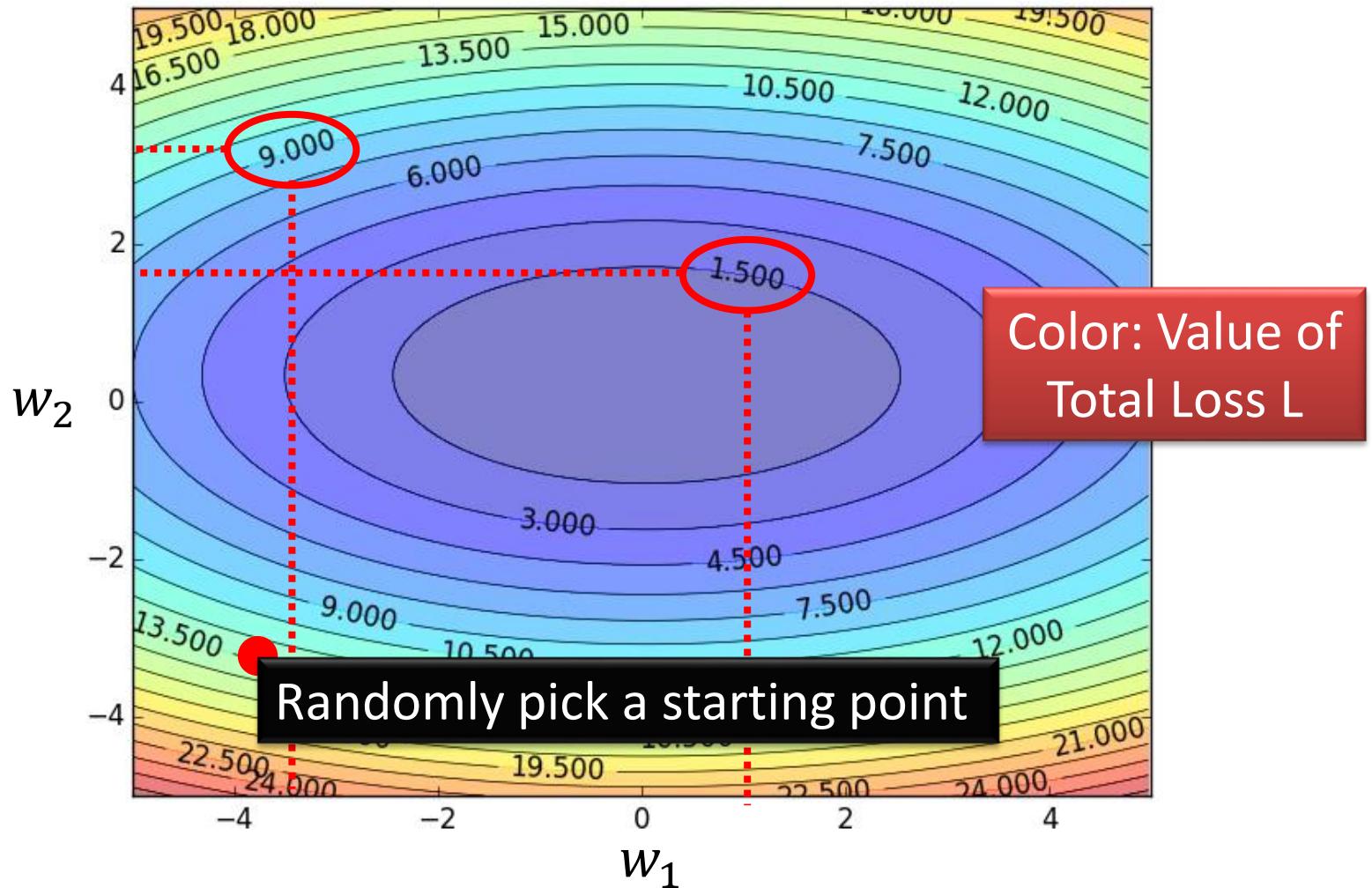
Gradient Descent



Gradient Descent

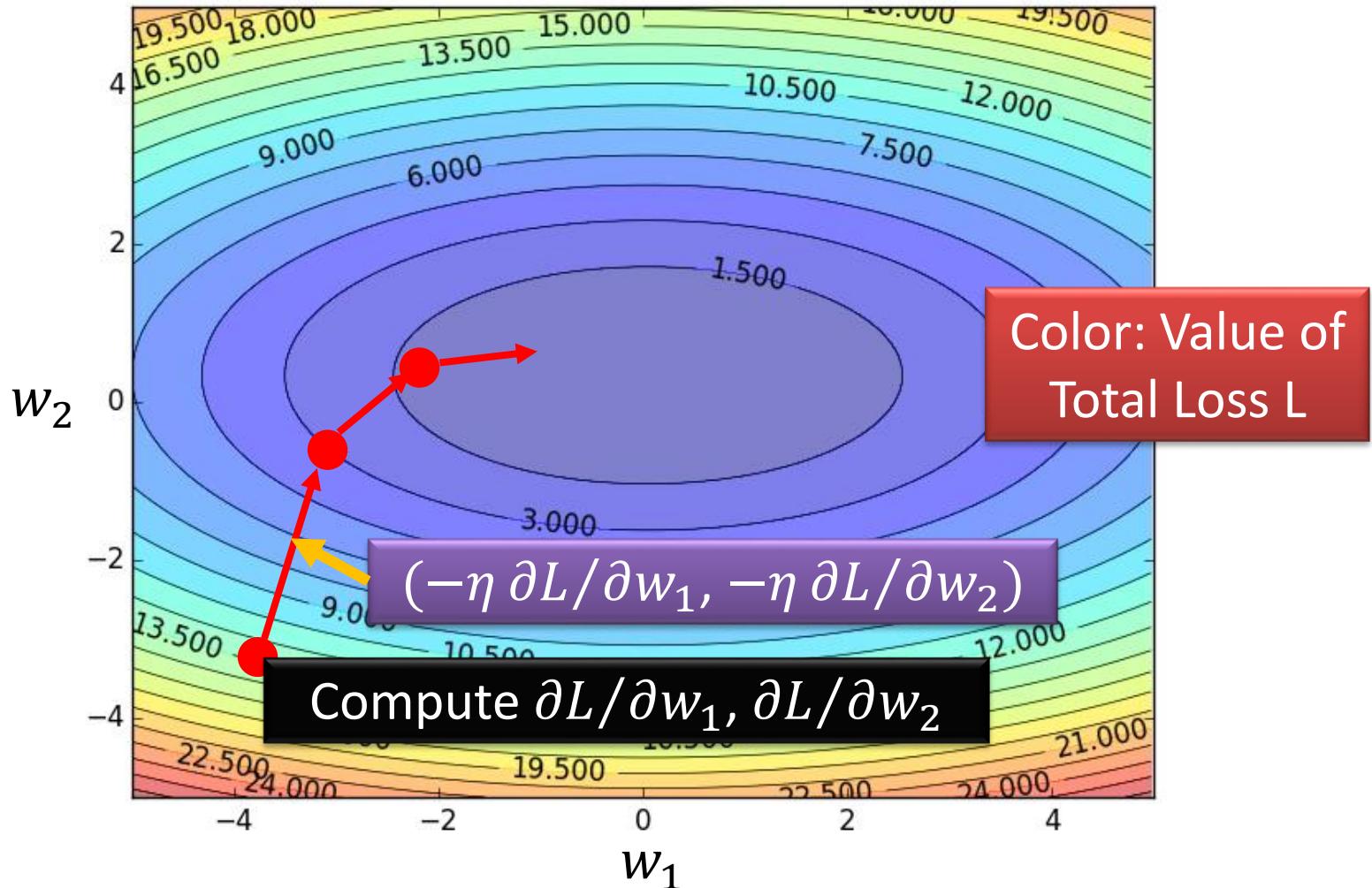


Gradient Descent



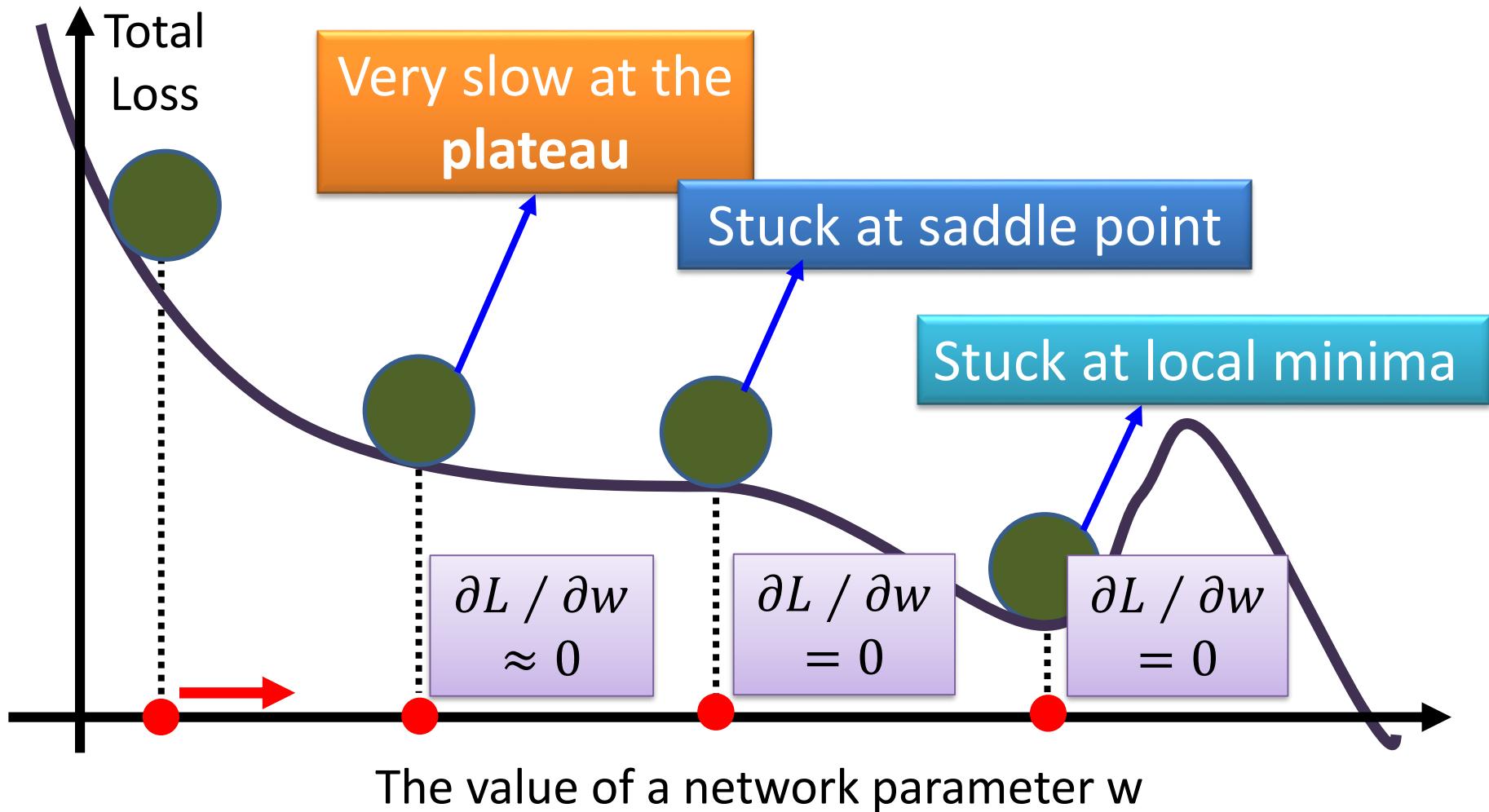
Gradient Descent

Hopefully, we would reach a minima



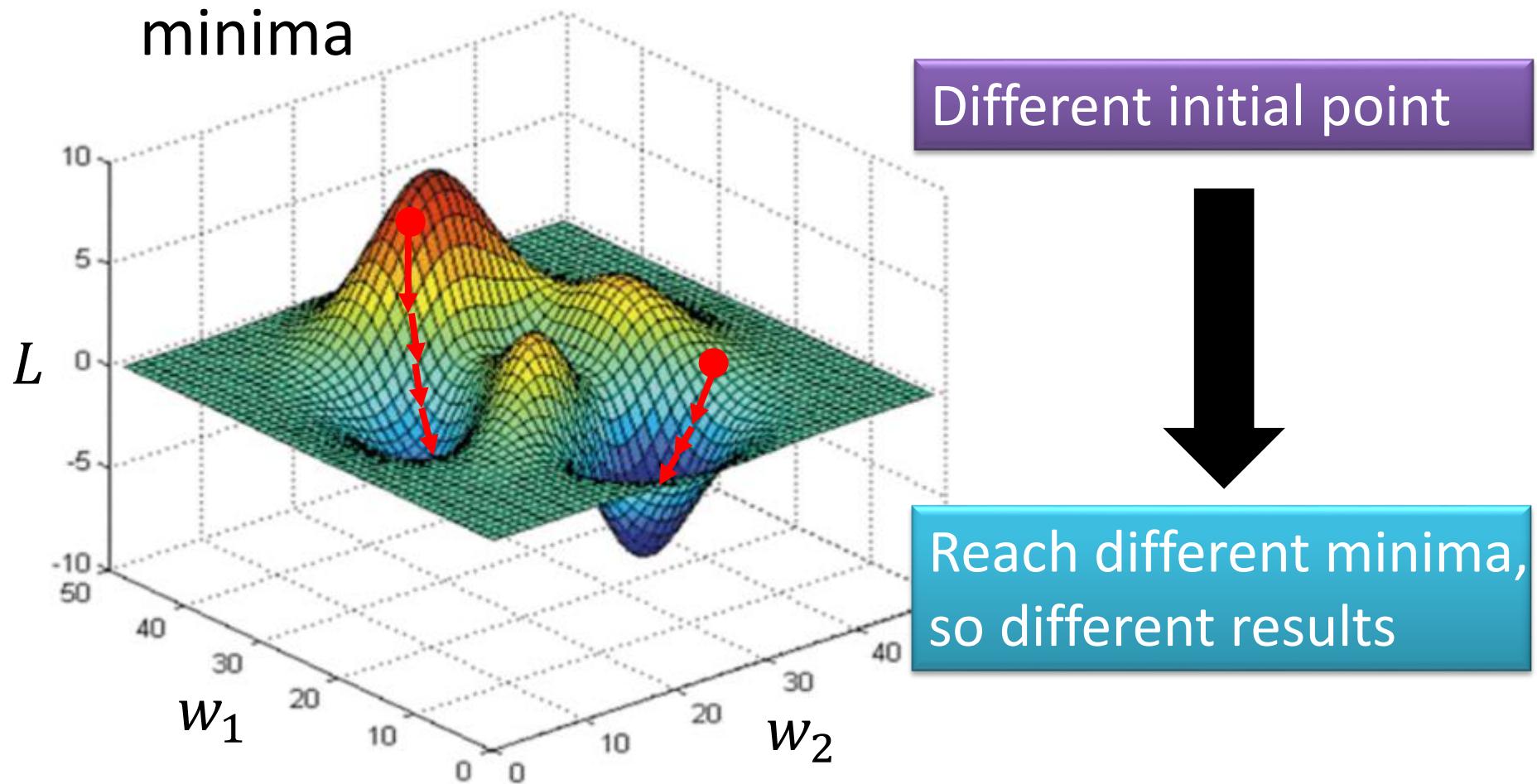
Local Minima

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



Local Minima

- Gradient descent never guarantee global minima



Gradient Descent

This is the “learning” of machines in deep learning



Even alpha go using this approach.

Three Steps for Deep Learning



Deep Learning is so simple

Now If you want to find a function

If you have lots of function input/output (?) as training data

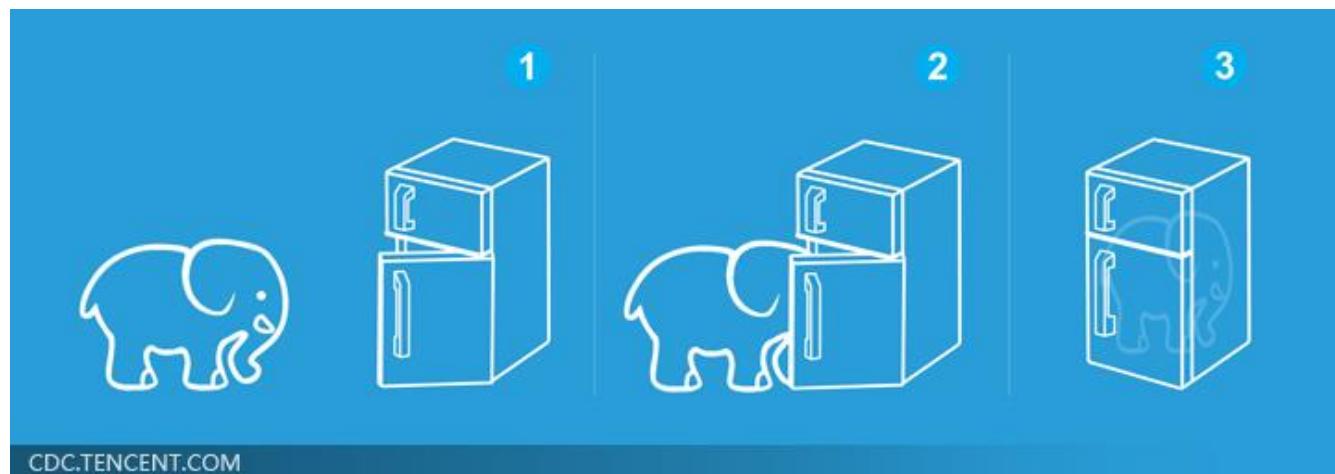


You can use deep learning

Three Steps for Deep Learning



Deep Learning is so simple

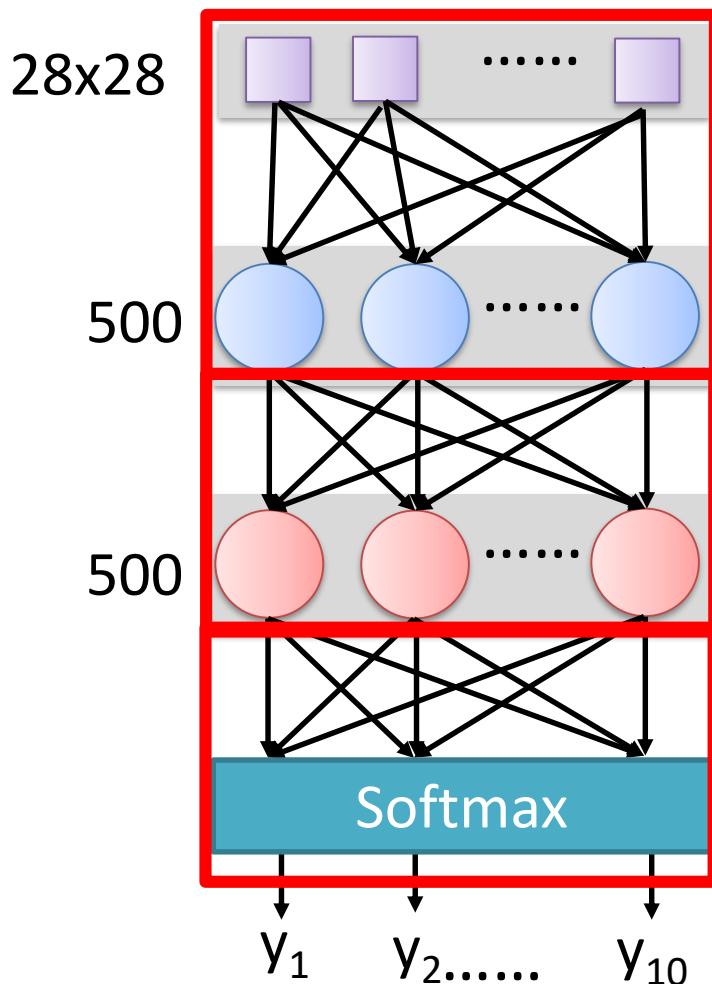


DNN in Keras

Step 1:
define a set
of function

Step 2:
goodness of
function

Step 3: pick
the best
function



```
model = Sequential()
```

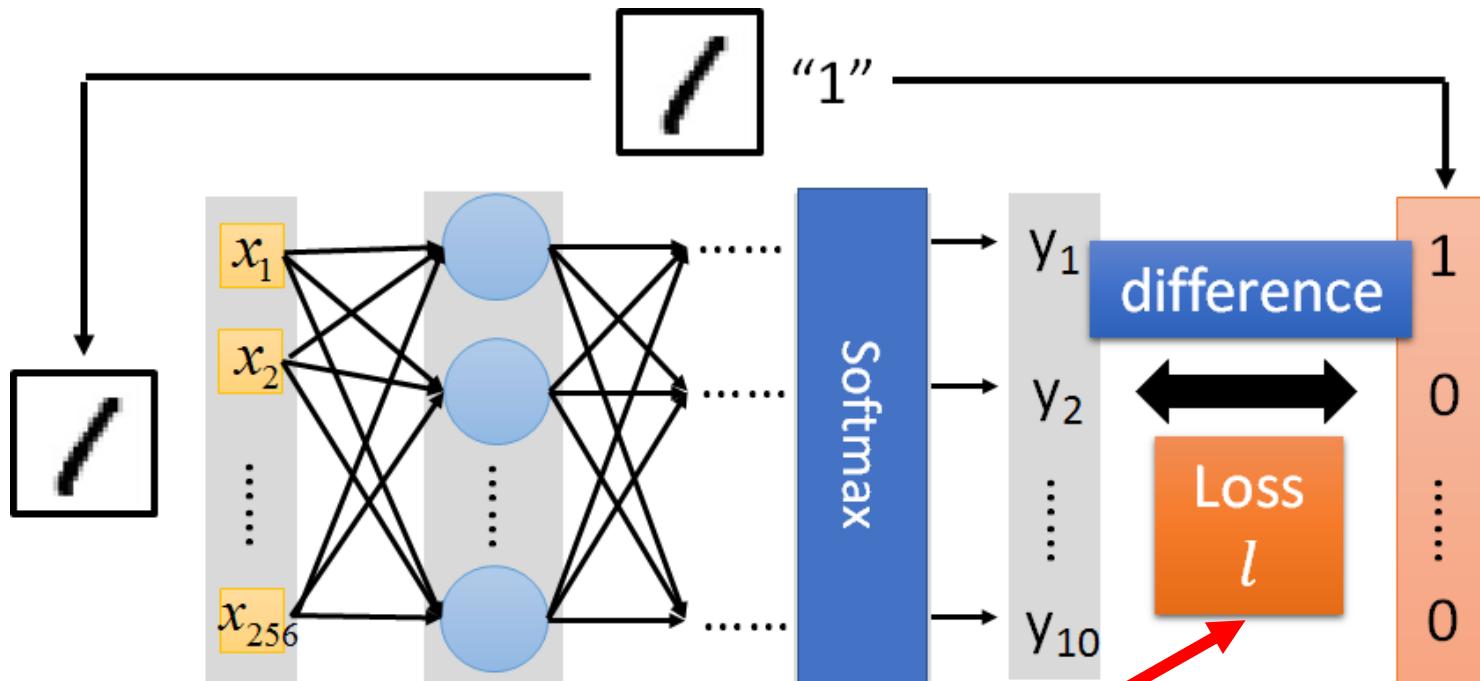
```
model.add( Dense( input_dim=28*28,  
                  output_dim=500 ) )  
model.add( Activation('sigmoid') )
```

softplus, softsign, relu, tanh,
hard_sigmoid, linear

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

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

Keras



```
model.compile(loss='categorical_crossentropy',  
              optimizer='adam',  
              metrics=['accuracy'])
```

Several alternatives: <https://keras.io/objectives/>

Keras

Step 1:
define a set
of function

Step 2:
goodness of
function

Step 3: pick
the best
function

Step 3.1: Configuration

```
model.compile(loss='categorical_crossentropy',  
               optimizer='adam',  
               metrics=['accuracy'])
```

SGD, RMSprop, Adagrad, Adadelta, Adam, Adamax, Nadam

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)

In the following slides

Keras

Step 1:
define a set
of function

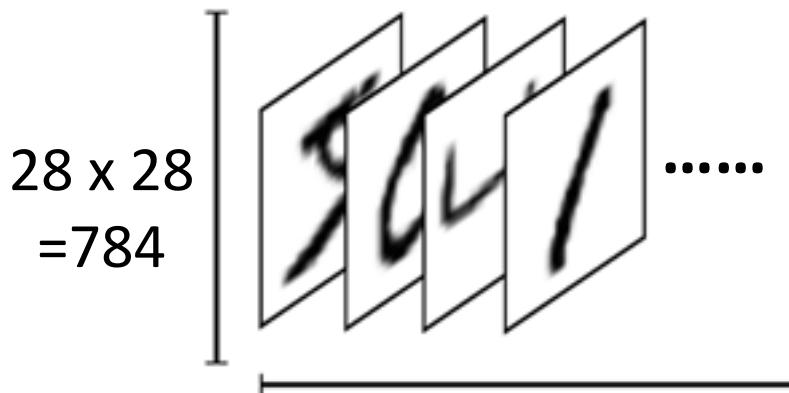
Step 2:
goodness of
function

Step 3: pick
the best
function

Step 3.2: Find the optimal network parameters

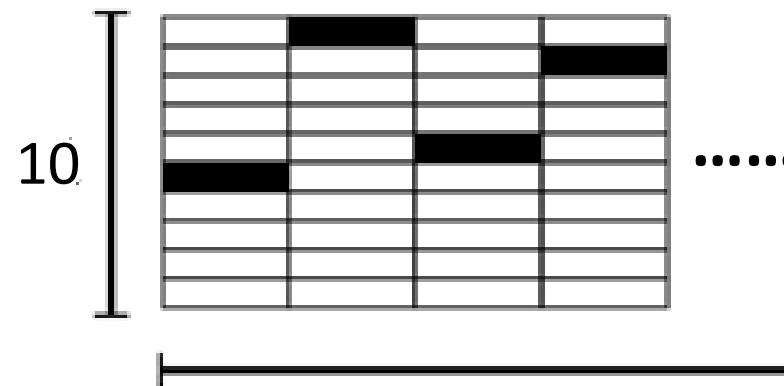
```
model.fit(x_train, y_train, batch_size=100, nb_epoch=20)
```

numpy array



Number of training examples

numpy array



Number of training examples

总结：神经网络的核心组件——层

- 层是神经网络的基本数据结构。
- 层是一个数据处理模块，将一个或多个输入张量转换为一个或多个输出张量。
- 有些层是无状态的，但大多数的层是有状态的，即层的权重。权重是利用随机梯度下降学到的一个或多个张量，其中包含网络的知识。
- 不同的张量格式与不同的数据处理类型需要用到不同的层。
 - 例如，简单的向量数据保存在形状为(samples, features)的2D张量中，通常用密集连接层 [densely connected layer，也叫全连接层 (fully connected layer) 或密集层 (dense layer)，对应于Keras的Dense类] 来处理。
 - 序列数据保存在形状为(samples, timesteps, features)的3D张量中，通常用循环层 (recurrent layer，比如Keras的LSTM层) 来处理。
 - 图像数据保存在4D张量中，通常用二维卷积层 (Keras的Conv2D) 来处理。
- 层兼容性 (layer compatibility) 具体指的是每一层只接受特定形状的输入张量，并返回特定形状的输出张量。

```
from keras import models
from keras import layers

model = models.Sequential()
model.add(layers.Dense(32, input_shape=(784,)))
model.add(layers.Dense(32))
```

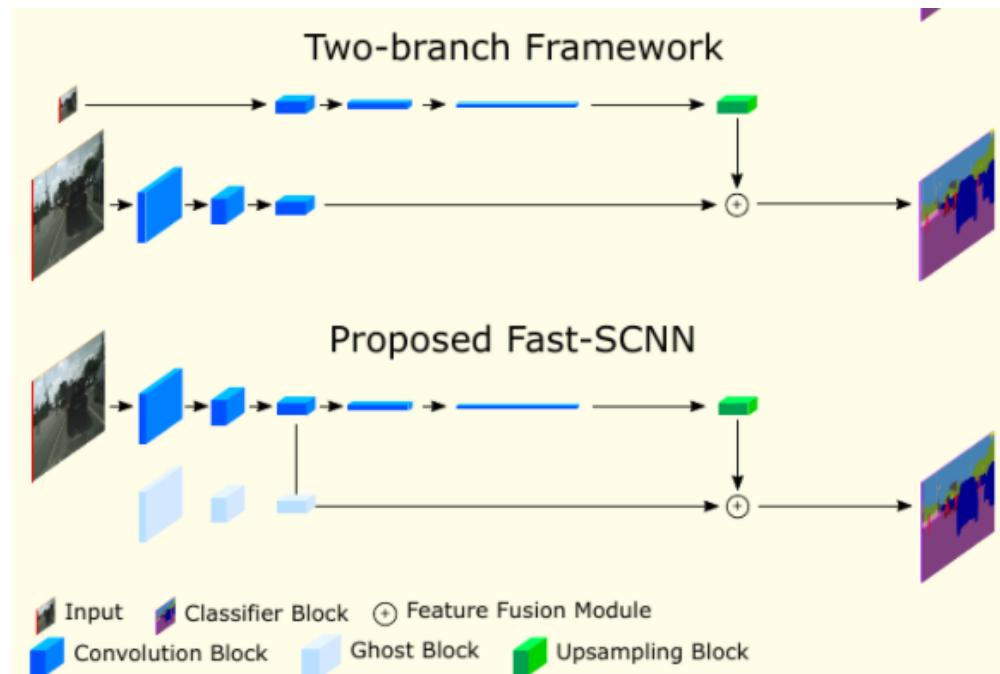
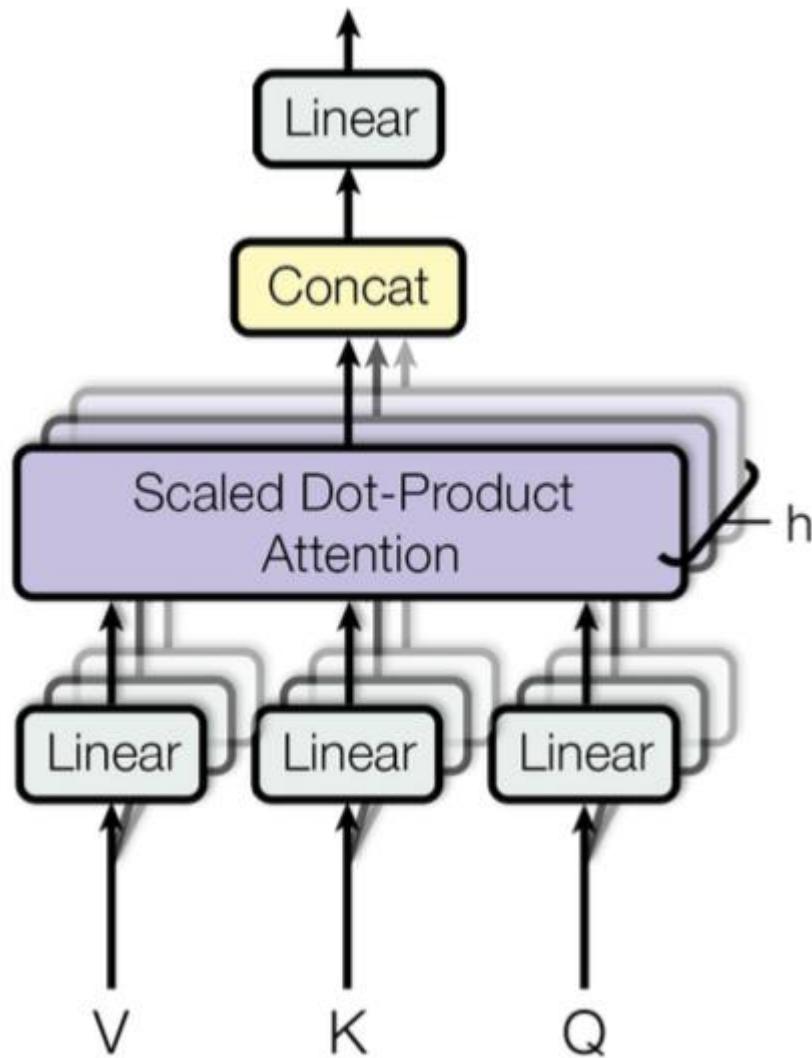
总结：神经网络的核心组件——网络

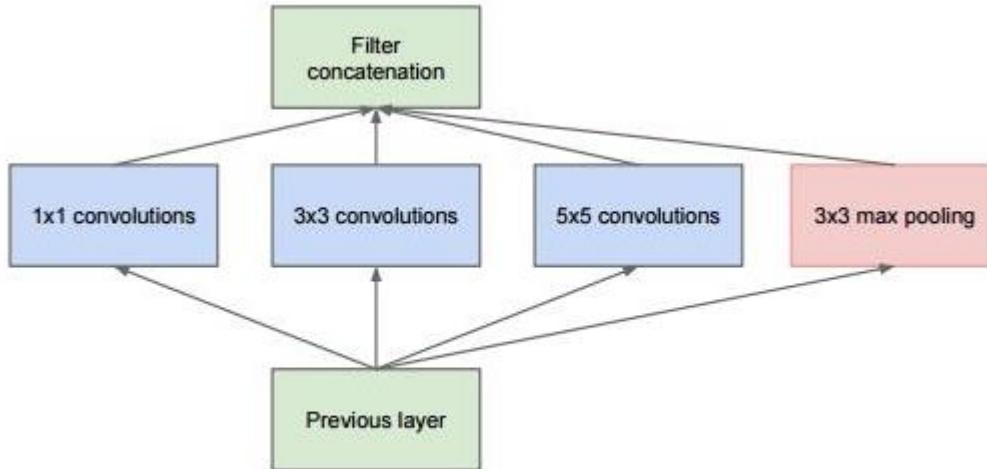
- 层：
- 模型：
- 层是神经网络的基本数据结构。
- 层是一个数据处理模块，将一个或多个输入张量转换为一个或多个输出张量。
- 有些层是无状态的，但大多数的层是有状态的，即层的权重。权重是利用随机梯度下降学到的一个或多个张量，其中包含网络的知识。
- 不同的张量格式与不同的数据处理类型需要用到不同的层。
 - 例如，简单的向量数据保存在形状为(samples, features)的2D张量中，通常用密集连接层 [**densely connected layer**，也叫全连接层 (**fully connected layer**) 或密集层 (**dense layer**)，对于Keras的**Dense**类] 来处理。
 - 序列数据保存在形状为(samples, timesteps, features)的3D张量中，通常用循环层 (**recurrent layer**，比如Keras的**LSTM**层) 来处理。
 - 图像数据保存在4D张量中，通常用二维卷积层 (Keras的**Conv2D**) 来处理。

总结：神经网络的核心组件——模型

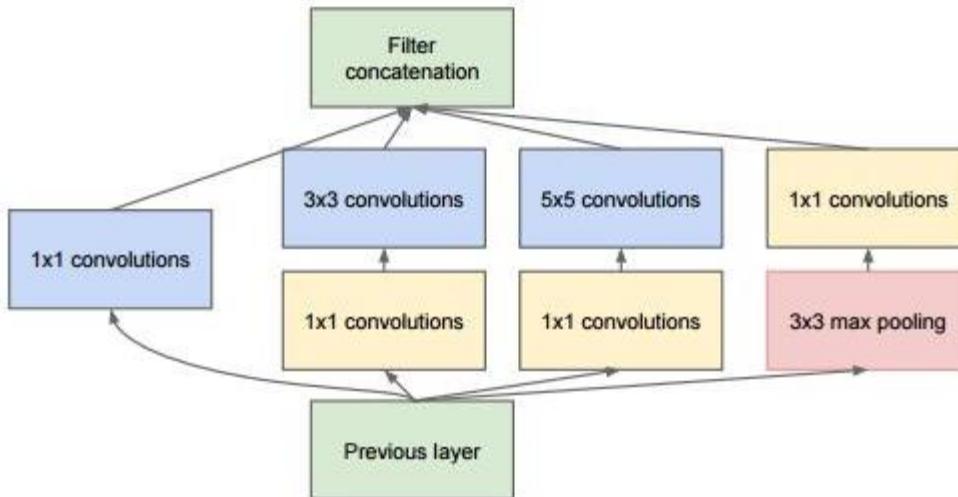
- 深度学习模型是层构成的有向无环图。
- 模型：由层堆叠构成。常见的网络拓扑结构：
 - 层的线性堆叠：将单一输入映射为单一输出
 - 双分支（two-branch）网络
 - 多头（multihead）网络
 - Inception模块

Multi-Head Attention

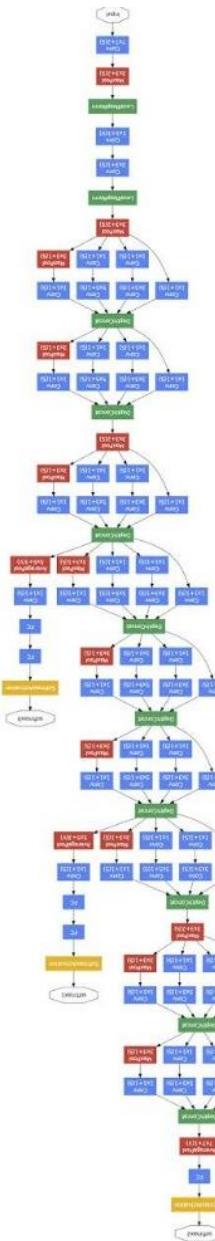




Inception module

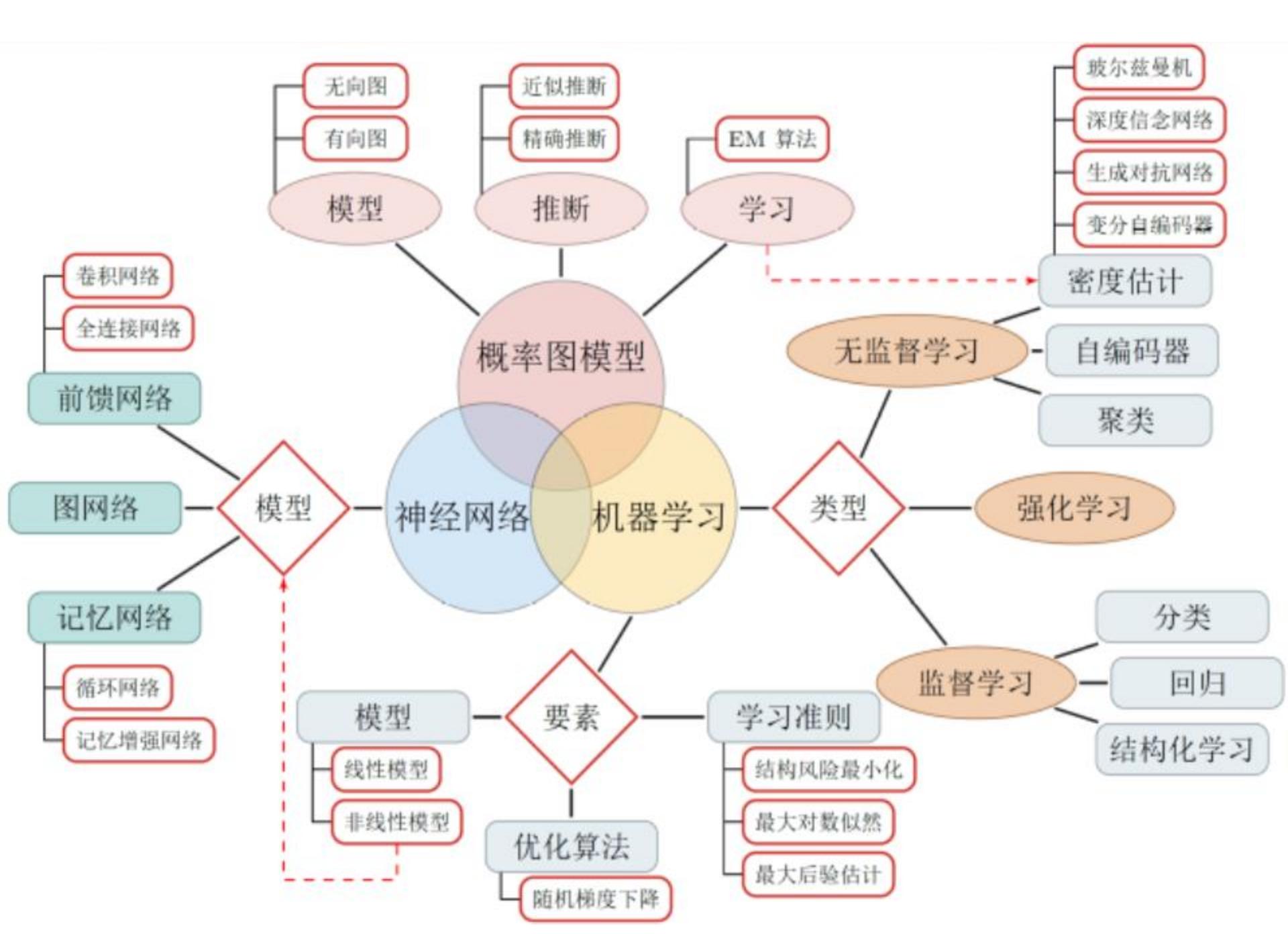


(b) Inception module with dimension reductions



GoogleNet (2014)

- **损失函数与优化器**: 配置学习过程的关键
 - 损失函数（目标函数）——在训练过程中需要将其最小化。它能够衡量当前任务是否已成功完成。
 - 如何选择正确的损失函数?
 - 对于二分类问题，你可以使用二元交叉熵（binary crossentropy）损失函数；
 - 对于多分类问题，可以用分类交叉熵（categorical crossentropy）损失函数；
 - 对于回归问题，可以用均方误差（mean-squared error）损失函数；
 - 对于序列学习问题，可以用联结主义时序分类（CTC, connectionist temporal classification）损失函数。
 - 优化器——决定如何基于损失函数对网络进行更新。它执行的是随机梯度下降（SGD）的某个变体。



Keras

- Keras是一个Python深度学习框架，可以方便地定义和训练几乎所有类型的深度学习模型。Keras最开始是为研究人员开发的，其目的在于快速实验。
- Keras具有以下重要特性：
 - 相同的代码可以在CPU或GPU上无缝切换运行。
 - 具有用户友好的API，便于快速开发深度学习模型的原型。
 - 内置支持卷积网络（用于计算机视觉）、循环网络（用于序列处理）以及二者的任意组合。
 - 支持任意网络架构：多输入或多输出模型、层共享、模型共享等。这也就是说，Keras能够构建任意深度学习模型，无论是生成式对抗网络还是神经图灵机。
- Keras基于宽松的MIT许可证发布，这意味着可以在商业项目中免费使用它。它与所有版本的Python都兼容（截至2017年年中，从Python 2.7到Python 3.6都兼容）。
- Keras已有200000多个用户，既包括创业公司和大公司的学术研究人员和工程师，也包括研究生和业余爱好者。Google、Netflix、Uber、CERN、Yelp、Square以及上百家创业公司都在用Keras解决各种各样的问题。Keras还是机器学习竞赛网站Kaggle上的热门框架，最新的深度学习竞赛中，几乎所有的优胜者用的都是Keras模型，

Keras

- Keras是一个模型级（model-level）的库，为开发深度学习模型提供了高层次的构建模块。
- Keras不处理张量操作、求微分等低层次的运算。相反，它依赖于一个专门的、高度优化的张量库——Keras的后端引擎（backend engine）来完成这些运算。
- Keras有三个后端实现：
 - TensorFlow后端
 - Theano后端
 - 微微软认知工具包（CNTK）后端



图 3-3 深度学习的软件栈和硬件栈

Keras工作流程

1. 定义训练数据：输入张量和目标张量。
2. 定义层组成的网络（或模型），将输入映射到目标。
 - 定义模型有两种方法：
 - 一种是使用**Sequential**类（仅用于层的线性堆叠，这是目前最常见的网络架构）；

```
from keras import models
from keras import layers
model = models.Sequential()
model.add(layers.Dense(32, activation='relu', input_shape=(784,)))
model.add(layers.Dense(10, activation='softmax'))
```
 - 另一种是**函数式API**（functional API，用于层组成的有向无环图，让你可以构建任意形式的架构）。

```
input_tensor = layers.Input(shape=(784,))
x = layers.Dense(32, activation='relu')(input_tensor)
output_tensor = layers.Dense(10, activation='softmax')(x)
model = models.Model(inputs=input_tensor, outputs=output_tensor)
```

Keras工作流程

3. 配置学习过程：选择损失函数、优化器和需要监控的指标。

```
from keras import optimizers

model.compile(optimizer=optimizers.RMSprop(lr=0.001),
              loss='mse',
              metrics=['accuracy'])
```

4. 调用模型的**fit**方法在训练数据上进行迭代。

```
model.fit(input_tensor, target_tensor, batch_size=128, epochs=10)
```

Keras示例

- 电影评论分类：二分类问题
 - IMDB数据集
- 新闻分类：多分类问题
 - 路透社数据集
- 预测房价：回归问题
 - 波士顿房价数据集

电影评论分类：二分类问题

- IMDB数据集：

- 它包含来自互联网电影数据库（IMDB）的50 000条严重两极分化的评论。数据集被分为用于训练的25 000条评论与用于测试的25 000条评论，训练集和测试集都包含50%的正面评论和50%的负面评论。
- 是由评论组成的列表，每条评论是单词索引组成的单词列表。标签中：0代表负面（negative），1代表正面

代码清单 3-1 加载 IMDB 数据集

```
from keras.datasets import imdb

(train_data, train_labels), (test_data, test_labels) = imdb.load_data(
    num_words=10000)

>>> train_data[0]
[1, 14, 22, 16, ... 178, 32]

>>> train_labels[0]
1
```

仅保留训练数据中前10 000个最常出现的单词。低频单词将被舍弃。

电影评论分类：二分类问题

• 准备数据

你不能将整数序列直接输入神经网络。你需要将列表转换为张量。转换方法有以下两种。

- 填充列表，使其具有相同的长度，再将列表转换成形状为 (samples, word_indices) 的整数张量，然后网络第一层使用能处理这种整数张量的层（即 Embedding 层，本书后面会详细介绍）。
- 对列表进行 one-hot 编码，将其转换为 0 和 1 组成的向量。举个例子，序列 [3, 5] 将会被转换为 10 000 维向量，只有索引为 3 和 5 的元素是 1，其余元素都是 0。然后网络第一层可以用 Dense 层，它能够处理浮点数向量数据。

代码清单 3-2 将整数序列编码为二进制矩阵

```
import numpy as np

def vectorize_sequences(sequences, dimension=10000):
    results = np.zeros((len(sequences), dimension)) ← 创建一个形状为 (len(sequences),
    for i, sequence in enumerate(sequences):                                dimension) 的零矩阵
        results[i, sequence] = 1. ← 将 results[i] 的指定索引设为 1
    return results

x_train = vectorize_sequences(train_data) ← 将训练数据向量化
x_test = vectorize_sequences(test_data) ← 将测试数据向量化
```

样本现在变成了这样：

```
>>> x_train[0]
array([ 0.,  1.,  1., ...,  0.,  0.,  0.])
```

你还应该将标签向量化，这很简单。

```
y_train = np.asarray(train_labels).astype('float32')
y_test = np.asarray(test_labels).astype('float32')
```

电影评论分类：二分类问题

- 构建网络

- 明确任务：输入数据是向量，而标签是标量（1和0），是二分类问题。

两个关键点：

1. 网络有多少层；
2. 每层有多少个隐藏单元。

代码清单 3-3 模型定义

```
from keras import models
from keras import layers

model = models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

代码清单 3-4 编译模型

```
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

代码清单 3-5 配置优化器

```
from keras import optimizers

model.compile(optimizer=optimizers.RMSprop(lr=0.001),
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

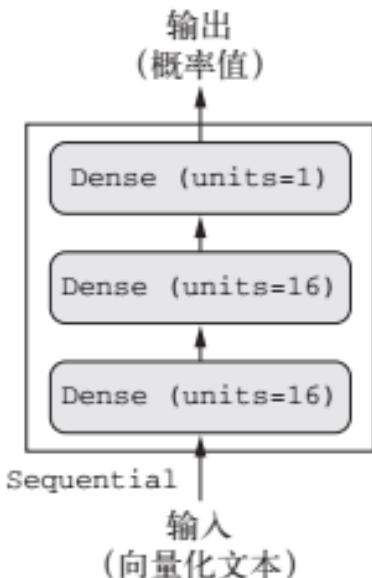


图 3-6 三层网络

电影评论分类：二分类问题

- 验证你的方法

代码清单 3-7 留出验证集

```
x_val = x_train[:10000]
partial_x_train = x_train[10000:]

y_val = y_train[:10000]
partial_y_train = y_train[10000:]
```

现在使用 512 个样本组成的小批量，将模型训练 20 个轮次（即对 `x_train` 和 `y_train` 两个张量中的所有样本进行 20 次迭代）。与此同时，你还要监控在留出的 10 000 个样本上的损失和精度。你可以通过将验证数据传入 `validation_data` 参数来完成。

代码清单 3-8 训练模型

```
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['acc'])

history = model.fit(partial_x_train,
                     partial_y_train,
                     epochs=20,
                     batch_size=512,
                     validation_data=(x_val, y_val))
```

`History` 对象有一个成员 `history`，它是一个字典，包含训练过程中的所有数据，该字典中包含 4 个条目，对应训练过程和验证过程中监控的指标。

- History对象有一个成员history，它是一个字典，包含训练过程中的所有数据，该字典中包含4个条目，对应训练过程和验证过程中监控的指标。

```
>>> history_dict = history.history
>>> history_dict.keys()
dict_keys(['val_acc', 'acc', 'val_loss', 'loss'])
```

代码清单 3-9 绘制训练损失和验证损失

```
import matplotlib.pyplot as plt

history_dict = history.history
loss_values = history_dict['loss']
val_loss_values = history_dict['val_loss']

epochs = range(1, len(loss_values) + 1)

plt.plot(epochs, loss_values, 'bo', label='Training loss') ←
plt.plot(epochs, val_loss_values, 'b', label='Validation loss') ←
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()
```

'bo' 表示蓝色圆点
'b' 表示蓝色实线

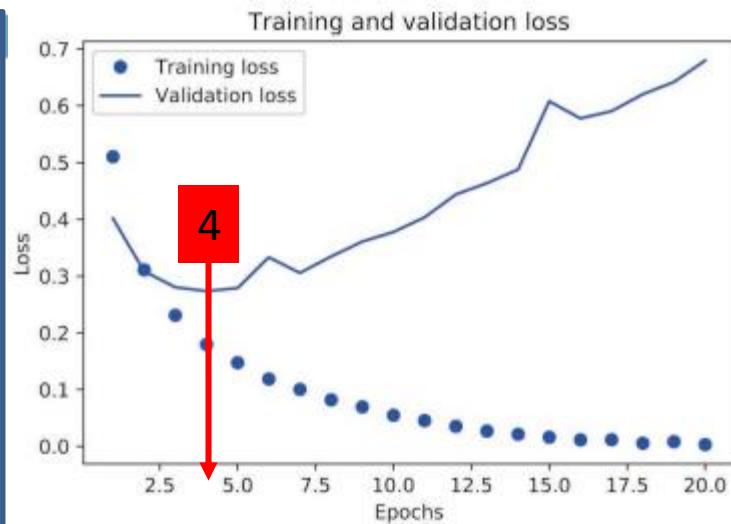


图 3-7 训练损失和验证损失

代码清单 3-11 从头开始重新训练一个模型

```
model = models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=4, batch_size=512)
results = model.evaluate(x_test, y_test)
```

最终结果如下所示。

```
>>> results
[0.2929924130630493, 0.8832799999999995]
```

结论：得到了88%的精度。利用最先进的方法，你应该能够得到接近95%的精度。

电影评论分类：二分类问题

- 使用训练好的网络在新数据上生成预测结果

```
>>> model.predict(x_test)  
array([[ 0.98006207]  
       [ 0.99758697]  
       [ 0.99975556]  
       ...,  
       [ 0.82167041]  
       [ 0.02885115]  
       [ 0.65371346]], dtype=float32)
```

观察到：网络对某些样本的结果非常确信（大于等于**0.99**，或小于等于**0.01**），但对其他结果却不那么确信（**0.6**或**0.4**）。

- 进一步的实验（课后作业）

- 前面使用了两个隐藏层。你可以尝试使用一个或三个隐藏层，然后观察对验证精度和测试精度的影响。
- 尝试使用更多或更少的隐藏单元，比如32个、64个等。
- 尝试使用mse损失函数代替binary_crossentropy。
- 试使用tanh激活（这种激活在神经网络早期非常流行）代替relu。

电影评论分类：二分类问题

1. 通常需要对原始数据进行大量预处理，以便将其转换为张量输入到神经网络中。单词序列可以编码为二进制向量，但也有其他编码方式。
2. 带有**relu**激活的**Dense**层堆叠，可以解决很多种问题（包括情感分类），你可能会经常用到这种模型。
3. 对于二分类问题（两个输出类别），网络的最后一层应该是只有一个单元并使用**sigmoid**激活的**Dense**层，网络输出应该是0~1范围内的标量，表示概率值。
4. 对于二分类问题的**sigmoid**标量输出，你应该使用**binary_crossentropy**损失函数。
5. 无论你的问题是什么，**rmsprop**优化器通常都是足够好的选择。这一点你无须担心。
6. 随着神经网络在训练数据上的表现越来越好，模型最终会过拟合，并在前所未见的数据上得到越来越差的结果。一定要一直监控模型在训练集之外的数据上的性能。

新闻分类：多分类问题

- 是单标签、多分类（single-label, multiclass classification）问题。
- 路透社数据集
 - 它包含许多短新闻及其对应的主题，由路透社在1986年发布。它是一个简单的、广泛使用的文本分类数据集。它包括46个不同的主题：某些主题的样本更多，但训练集中每个主题都有至少10个样本。
 - 是Keras的内置数据集。

代码清单 3-12 加载路透社数据集

```
from keras.datasets import reuters

(train_data, train_labels), (test_data, test_labels) = reuters.load_data(
    num_words=10000)
```

与 IMDB 数据集一样，参数 `num_words=10000` 将数据限定为前 10 000 个最常出现的单词。我们有 8982 个训练样本和 2246 个测试样本。

```
>>> len(train_data)
8982
>>> len(test_data)
2246
```

与 IMDB 评论一样，每个样本都是一个整数列表（表示单词索引）。

```
>>> train_data[10]
[1, 245, 273, 207, 156, 53, 74, 160, 26, 14, 46, 296, 26, 39, 74, 2979,
3554, 14, 46, 4689, 4329, 86, 61, 3499, 4795, 14, 61, 451, 4329, 17, 12]
```

如果好奇的话，你可以用下列代码将索引解码为单词。

代码清单 3-13 将索引解码为新闻文本

```
word_index = reuters.get_word_index()
reverse_word_index = dict([(value, key) for (key, value) in word_index.items()])
decoded_newswire = ' '.join([reverse_word_index.get(i - 3, '?') for i in
    train_data[0]])
```

注意，索引减去了 3，因为 0、1、2 是为“padding”（填充）、“start of sequence”（序列开始）、“unknown”（未知词）分别保留的索引

样本对应的标签是一个 0~45 范围内的整数，即话题索引编号。

```
>>> train_labels[10]
```

新闻分类：多分类问题

- 准备数据：将数据向量化
 - 将数据集中的输入向量化

代码清单 3-14 编码数据

```
import numpy as np

def vectorize_sequences(sequences, dimension=10000):
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        results[i, sequence] = 1.
    return results

x_train = vectorize_sequences(train_data) ← 将训练数据向量化
x_test = vectorize_sequences(test_data)   ← 将测试数据向量化
```

- 将标签向量化。

- 将标签列表转换为整数张量
- 或者使用**one-hot**编码。**one-hot**编码是分类数据广泛使用的一种格式，也叫分类编码（**categorical encoding**）

```
def to_one_hot(labels, dimension=46):
    results = np.zeros((len(labels), dimension))
    for i, label in enumerate(labels):
        results[i, label] = 1.
    return results

one_hot_train_labels = to_one_hot(train_labels) ← 将训练标签向量化
one_hot_test_labels = to_one_hot(test_labels)   ← 将测试标签向量化
```

```
from keras.utils.np_utils import to_categorical

one_hot_train_labels = to_categorical(train_labels)
one_hot_test_labels = to_categorical(test_labels)
```

新闻分类：多分类问题

- 构建网络

最后一层是大小为46的Dense层；
最后一层使用了softmax激活。

代码清单 3-15 模型定义

```
from keras import models
from keras import layers

model = models.Sequential()
model.add(layers.Dense(64, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(46, activation='softmax'))
```

代码清单 3-16 编译模型

```
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

标签编码采用One-hot编码时

```
model.compile(optimizer='rmsprop',
              loss='sparse_categorical_crossentropy',
              metrics=['acc'])
```

标签编码采用整数张量时

新闻分类：多分类问题

- 验证你的方法
 - 在训练数据中留出1000个样本作为验证集。

代码清单 3-17 留出验证集

```
x_val = x_train[:1000]
partial_x_train = x_train[1000:]

y_val = one_hot_train_labels[:1000]
partial_y_train = one_hot_train_labels[1000:]
```

现在开始训练网络，共 20 个轮次。

代码清单 3-18 训练模型

```
history = model.fit(partial_x_train,
                     partial_y_train,
                     epochs=20,
                     batch_size=512,
                     validation_data=(x_val, y_val))
```

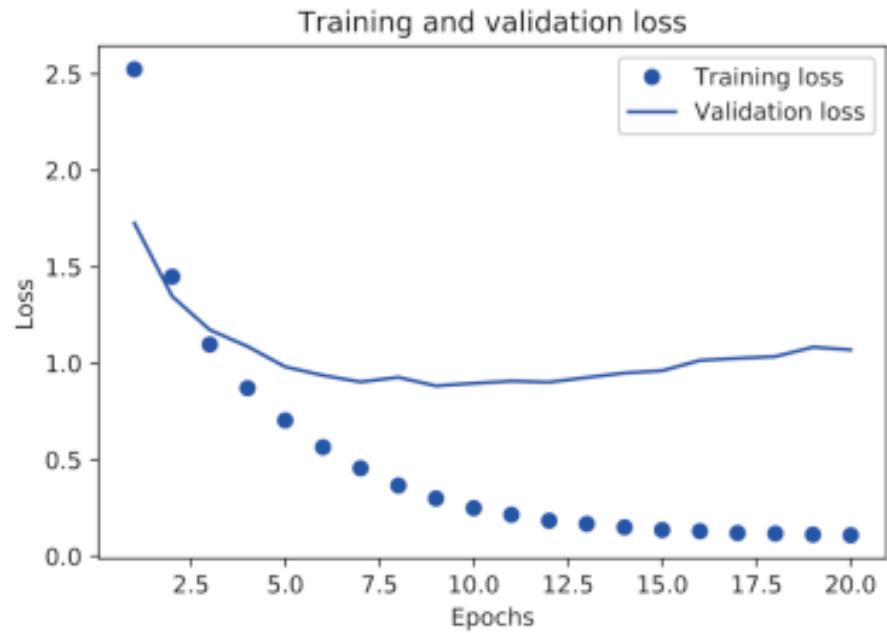


图 3-9 训练损失和验证损失

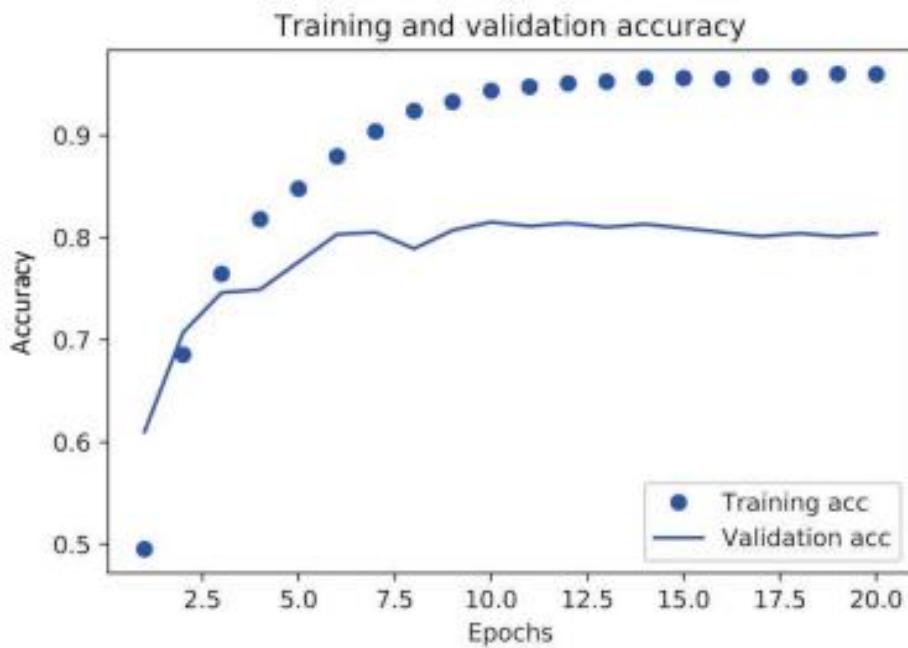


图 3-10 训练精度和验证精度

观察到：网络在训练9轮后开始过拟合。
因此，从头开始训练一个新网络，共9个轮次，然后在测试集上评估模型。

代码清单 3-21 从头开始重新训练一个模型

```
model = models.Sequential()
model.add(layers.Dense(64, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(46, activation='softmax'))

model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
model.fit(partial_x_train,
          partial_y_train,
          epochs=9,
          batch_size=512,
          validation_data=(x_val, y_val))
results = model.evaluate(x_test, one_hot_test_labels)
```

最终结果如下。

```
>>> results
[0.9565213431445807, 0.79697239536954589]
```

结论：这种方法可以得到约80%的精度

新闻分类：多分类问题

- 使用训练好的网络在新数据上生成预测结果

代码清单 3-22 在新数据上生成预测结果

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

predictions 中的每个元素都是长度为 46 的向量。

```
>>> predictions[0].shape  
(46,)
```

这个向量的所有元素总和为 1。

```
>>> np.sum(predictions[0])  
1.0
```

最大的元素就是预测类别，即概率最大的类别。

```
>>> np.argmax(predictions[0])  
4
```

- 中间层维度足够大的重要性
 - 前面提到，最终输出是46维的，因此中间层的隐藏单元个数不应该比46小太多。现在来看一下，如果中间层的维度远远小于46（比如4维），造成了信息瓶颈，那么会发生什么？

代码清单 3-23 具有信息瓶颈的模型

```
model = models.Sequential()
model.add(layers.Dense(64, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(4, activation='relu'))
model.add(layers.Dense(46, activation='softmax'))
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
model.fit(partial_x_train,
          partial_y_train,
          epochs=20,
          batch_size=128,
          validation_data=(x_val, y_val))
```

结论：网络的验证精度最大约为**71%**，比前面下降了**8%**。导致这一下降的主要原因在于，你试图将大量信息（这些信息足够恢复**46**个类别的分割超平面）压缩到维度很小的中间空间。网络能够将大部分必要信息塞入这个四维表示中，但并不是全部信息。

新闻分类：多分类问题

- 进一步的实验（作业）
 - 尝试使用更多或更少的隐藏单元，比如32个、128个等。
 - 前面使用了两个隐藏层，现在尝试使用一个或三个隐藏层。

新闻分类：多分类问题

- 小结

1. 如果要对N个类别的数据点进行分类，网络的最后一层应该是大小为N的Dense层。
2. 对于单标签、多分类问题，网络的最后一层应该使用softmax激活，这样可以输出在N个输出类别上的概率分布。
3. 这种问题的损失函数几乎总是应该使用分类交叉熵。它将网络输出的概率分布与目标的真实分布之间的距离最小化。
4. 处理多分类问题的标签有两种方法。
 - 通过分类编码（也叫one-hot编码）对标签进行编码，然后使用categorical_crossentropy作为损失函数。
 - 将标签编码为整数，然后使用sparse_categorical_crossentropy损失函数。
5. 如果你需要将数据划分到许多类别中，应该避免使用太小的中间层，以免在网络中造成信息瓶颈。

预测房价：回归问题

- 任务：预测一个连续值而不是离散的标签
- 波士顿房价数据集
 - 将要预测20世纪70年代中期波士顿郊区房屋价格的中位数（单位是千美元），已知当时郊区的一些数据点，比如犯罪率、当地房产税率等。
 - 它包含的数据点相对较少，只有506个，分为404个训练样本和102个测试样本。每个样本都有13个数值特征，比如人均犯罪率、每个住宅的平均房间数、高速公路可达性等。输入数据的每个特征（比如犯罪率）都有不同的取值范围。例如，有些特性是比例，取值范围为0~1；有的取值范围为1~12；还有的取值范围为0~100，等等。

代码清单 3-24 加载波士顿房价数据

```
from keras.datasets import boston_housing  
  
(train_data, train_targets), (test_data, test_targets) = boston_housing.load_data()
```

我们来看一下数据。

```
>>> train_data.shape          >>> train_targets  
(404, 13)                   array([ 15.2,  42.3,  50. ... 19.4, 19.4, 29.1])  
>>> test_data.shape  
(102, 13)
```

预测房价：回归问题

- 准备数据

代码清单 3-25 数据标准化

```
mean = train_data.mean(axis=0)
train_data -= mean
std = train_data.std(axis=0)
train_data /= std

test_data -= mean
test_data /= std
```

注意，用于测试数据标准化的均值和标准差都是在训练数据上计算得到的。

在工作流程中，你不能使用在测试数据上计算得到的任何结果，即使是像数据标准化这么简单的事情也不行。

预测房价：回归问题

- 构建网络

由于样本数量很少，我们将使用一个非常小的网络，其中包含两个隐藏层，每层有64个单元。

代码清单 3-26 模型定义

```
from keras import models
from keras import layers

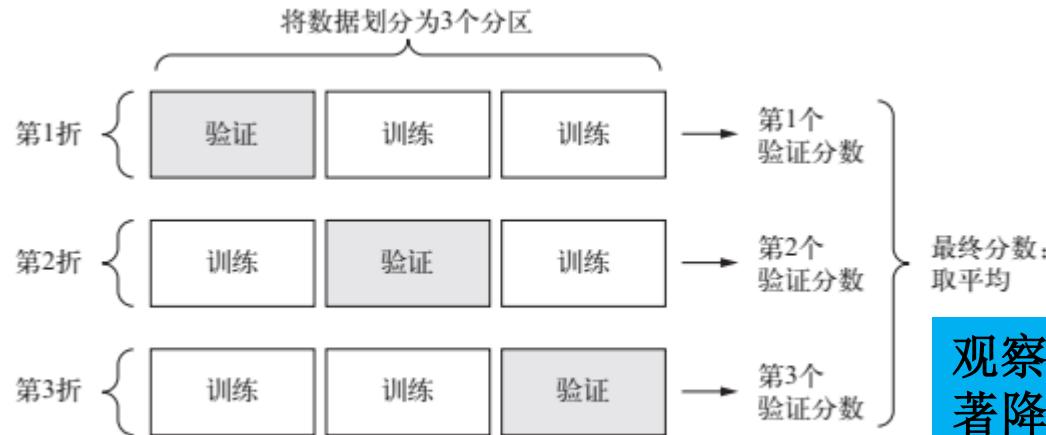
def build_model():
    model = models.Sequential() ← 因为需要将同一个模型多次实例化,
    model.add(layers.Dense(64, activation='relu',
                           input_shape=(train_data.shape[1],)))
    model.add(layers.Dense(64, activation='relu'))
    model.add(layers.Dense(1))
    model.compile(optimizer='rmsprop', loss='mse', metrics=['mae'])
    return model
```

注意：

1. 网络的最后一层只有一个单元，没有激活，是一个线性层。这是标量回归（标量回归是预测单一连续值的回归）的典型设置。
2. 编译网络用的是mse损失函数，即均方误差（MSE， mean squared error），预测值与目标值之差的平方。这是回归问题常用的损失函数。
3. 在训练过程中还监控一个新指标：平均绝对误差（MAE， mean absolute error）。它是预测值与目标值之差的绝对值。

预测房价：回归问题

- 利用K折验证来验证你的方法



观察到：验证MAE在80轮后不再显著降低，之后就开始过拟合。

图 3-11 3 折交叉验证

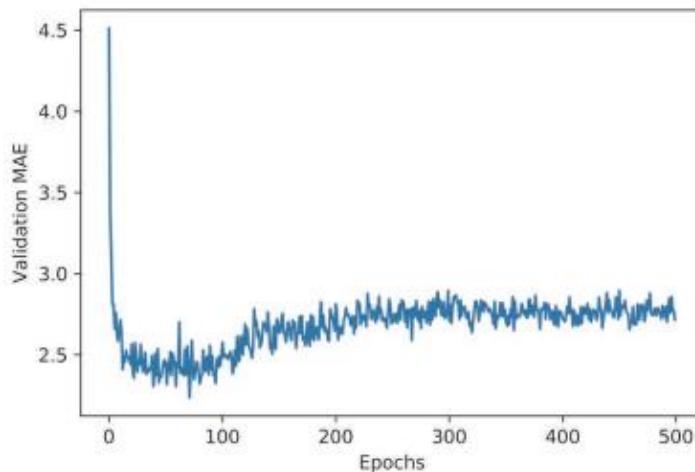


图 3-12 每轮的验证 MAE

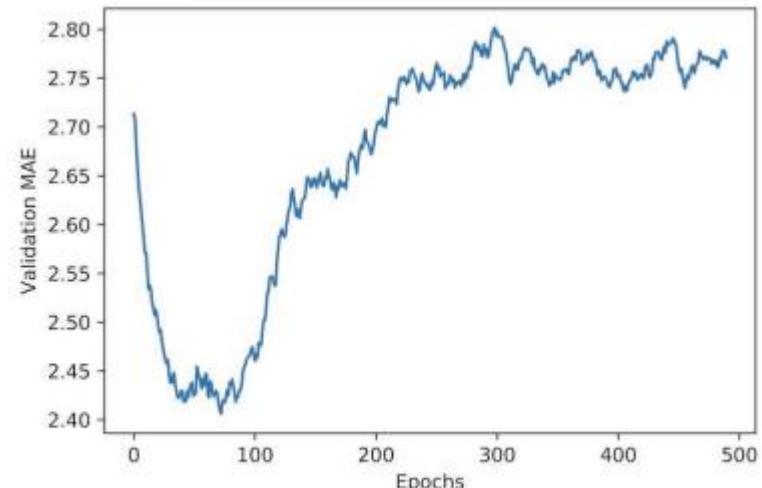


图 3-13 每轮的验证 MAE (删除前 10 个数据点)

- 使用训练好的网络在新数据上生成预测结果

代码清单 3-32 训练最终模型

```
model = build_model()      ← 一个全新的编译好的模型
model.fit(train_data, train_targets,   ← 在所有训练数据上训练模型
          epochs=80, batch_size=16, verbose=0)
test_mse_score, test_mae_score = model.evaluate(test_data, test_targets)
```

最终结果如下。

```
>>> test_mae_score
2.5532484335057877
```

- 小结
1. 回归问题使用的损失函数与分类问题不同。回归常用的损失函数是均方误差（MSE）。
 2. 同样，回归问题使用的评估指标也与分类问题不同。显而易见，精度的概念不适用于回归问题。常见的回归指标是平均绝对误差（MAE）。
 3. 如果输入数据的特征具有不同的取值范围，应该先进行预处理，对每个特征单独进行缩放。
 4. 如果可用的数据很少，使用K折验证可以可靠地评估模型。
 5. 如果可用的训练数据很少，最好使用隐藏层较少（通常只有一到两个）的小型网络，以避免严重的过拟合。

总结：关于交叉验证

- 交叉验证的方式充分利用了所有样本信息，给了每个样本作为训练集和验证集的机会，保障了鲁棒性，防止过拟合。
- 选择多种模型进行训练时，使用交叉验证能够评判各模型的鲁棒性/泛化性。
- 选择同一个模型的不同超参数组合时，使用交叉验证评估各超参数组合下的模型稳定性。

本章内容

1 什么是深度学习

2 神经网络的数学基础

3 神经网络入门

4 神经网络的通用工作流程

神经网络的通用工作流程

1. 定义问题，收集数据集
2. 选择衡量成功的指标
 - 要在验证数据上监控哪些指标？
3. 确定评估方法
 - 留出验证？K折验证？你应该将哪一部分数据用于验证？
4. 准备数据
5. 开发比基准更好的模型
6. 扩大模型规模：开发过拟合的模型
7. 模型正则化与调节超参数：基于模型在验证数据上的性能（目前被过度关注）

神经网络的通用工作流程

1. 定义问题，收集数据集
 - 要在验证数据上监控哪些指标？
2. 选择衡量成功的指标
3. 确定评估方法
 - 留出验证？K折验证？你应该将哪一部分数据用于验证？
4. 准备数据
5. 开发比基准更好的模型
6. 扩大模型规模：开发过拟合的模型
7. 模型正则化与调节超参数：基于模型在验证数据上的性能（目前被过度关注）

- 1 定义问题：软件架构设计、确定评价指标
- 2 获取数据：自动化方式
- 3 研究数据：可视化方式，相关性研究等
- 4 准备数据：数据清理、特征选择及处理
- 5 研究模型：确定验证集上的评估方法、列出可能的模型并训练，选择最有希望的3~5个模型
- 6 微调模型：寻找最佳超参数，模型融合，评估泛化性能
- 7 展示解决方案：将工作进行文档化总结展示
- 8 启动、监视、维护系统：投入使用

定义问题，收集数据集

- 定义问题，收集数据集
 - 你的输入数据是什么？你要预测什么？
 - 你面对的是什么类型的问题？是二分类问题、多分类问题、标量回归问题、向量回归问题，还是多分类、多标签问题？或者是其他问题，比如聚类、生成或强化学习？
 - 确定问题类型有助于你选择模型架构、损失函数等。
- 能成功解决问题的2个前提假设：
 - 假设输出是可以根据输入进行预测的。
 - 假设可用数据包含足够多的信息，足以学习输入和输出之间的关系。
 - 假设测试数据集与训练数据集来自同一分布。

- 虽然监督学习主要包括分类和回归，但还有更多的奇特变体，主要包括如下几种：
 - 序列生成（sequence generation）。给定一张图像，预测描述图像的文字。序列生成有时可以被重新表示为一系列分类问题，比如反复预测序列中的单词或标记。
 - 语法树预测（syntax tree prediction）。给定一个句子，预测其分解生成的语法树。
 - 目标检测（object detection）。给定一张图像，在图中特定目标的周围画一个边界框。这个问题也可以表示为分类问题（给定多个候选边界框，对每个框内的目标进行分类）或分类与回归联合问题（用向量回归来预测边界框的坐标）。
 - 图像分割（image segmentation）。给定一张图像，在特定物体上画一个像素级的掩模（mask）。

选择衡量成功的指标

- 业务成功的定义：精度？准确率（precision）和召回率（recall）？客户保留率？
- 衡量成功的指标将指引你选择损失函数，即模型要优化什么。它应该直接与你的目标（如业务成功）保持一致。
- 如何选择衡量成功的指标？
 - 对于平衡分类问题（每个类别的可能性相同），精度和接收者操作特征曲线下面积（area under the receiver operating characteristic curve, ROC AUC）是常用的指标。
 - 对于类别不平衡的问题，你可以使用准确率和召回率。
 - 对于排序问题或多标签分类，你可以使用平均准确率均值（mean average precision）。
 - 自定义衡量成功的指标也很常见。
 - 浏览Kaggle网站上的数据科学竞赛，上面展示了各种各样的问题和评估指标。

确定评估方法

- 三种常见的评估方法：
 - 留出验证集。数据量很大时可以采用这种方法。
 - K折交叉验证。如果留出验证的样本量太少，无法保证可靠性，那么应该选择这种方法。
 - 重复的K折验证。如果可用的数据很少，同时模型评估又需要非常准确，那么应该使用这种方法。
- 只需选择三者之一。
- 大多数情况下，第一种方法足以满足要求。

准备数据

- 将数据（包含输入数据和目标数据）格式化为张量，使其可以输入到机器学习模型中（这里假设模型为深度神经网络）。
- 这些张量的取值通常应该缩放为较小的值，比如在 $[-1, 1]$ 区间或 $[0, 1]$ 区间。
- 如果不同的特征具有不同的取值范围（异质数据），那么应该做数据标准化。
- 你可能需要做特征工程，尤其是对于小数据问题。

开发比基准更好的模型

- 目标：获得统计功效（statistical power），即开发一个小型模型，它能够打败纯随机的基准（dumb baseline）。
 - 比如在MNIST数字分类的例子中，任何精度大于0.1的模型都可以说具有统计功效；
 - 比如在IMDB的例子中，任何精度大于0.5的模型都可
- 如果你尝试了多种合理架构之后仍然无法打败随机基准，那么原因可能是训练数据集有问题。

- 如果能打败随机基准，那还需要选择三个关键参数来构建第一个工作模型。
 - 最后一层的激活函数的选择。它对网络输出进行有效的限制。
 - 例如，IMDB分类的例子在最后一层使用了sigmoid，回归的例子在最后一层没有使用激活，等等。
 - 损失函数。它应该匹配你要解决的问题的类型。
 - 例如，IMDB的例子使用binary_crossentropy、回归的例子使用mse，等等。
 - 优化配置。你要使用哪种优化器？学习率是多少？大多数情况下，使用rmsprop及其默认的学习率是稳妥的。

表 4-1 为模型选择正确的最后一层激活和损失函数

问题类型	最后一层激活	损失函数
二分类问题	sigmoid	binary_crossentropy
多分类、单标签问题	softmax	categorical_crossentropy
多分类、多标签问题	sigmoid	binary_crossentropy
回归到任意值	无	mse
回归到 0~1 范围内的值	sigmoid	mse 或 binary_crossentropy

扩大模型规模：开发过拟合的模型

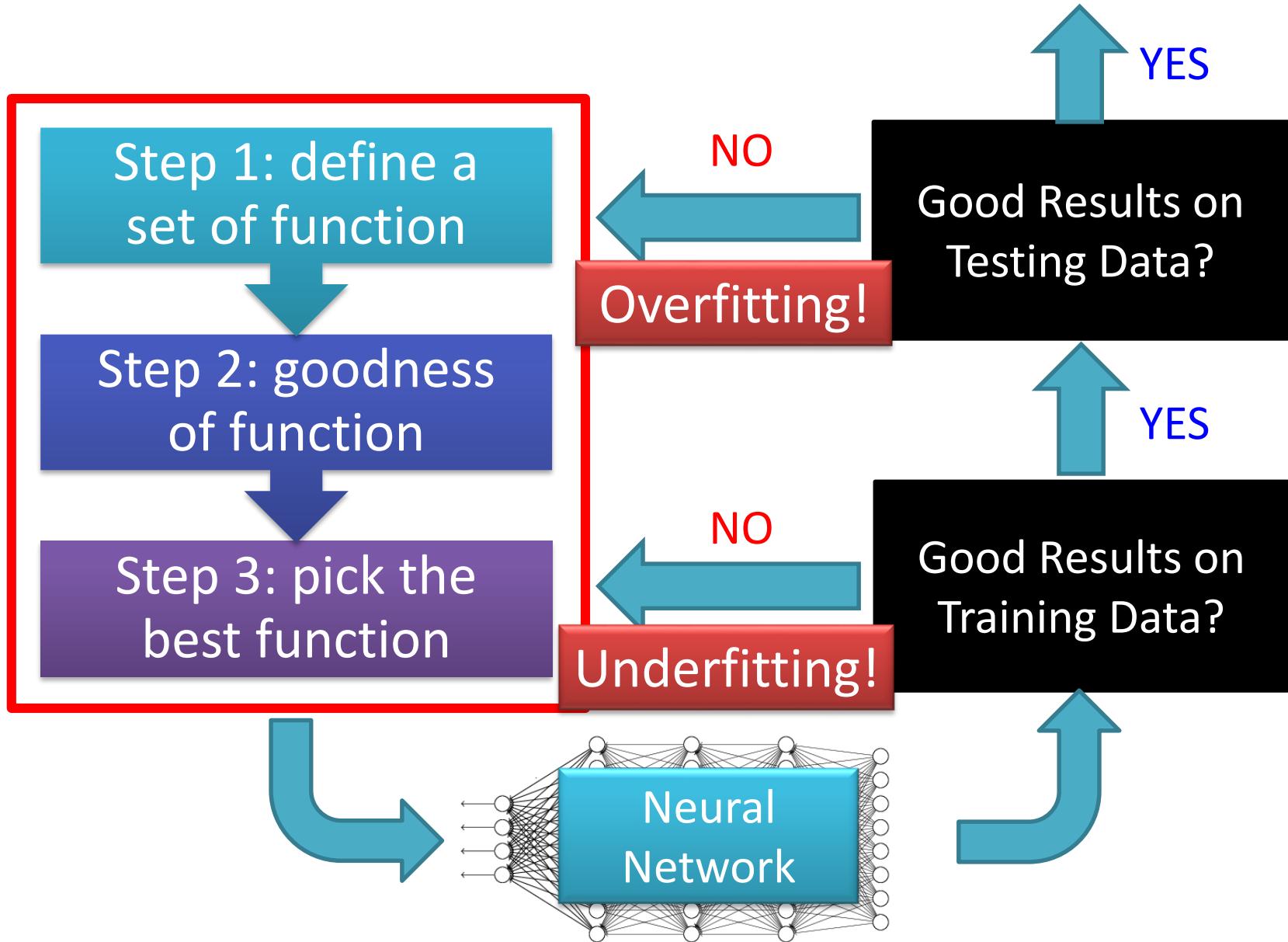
- 理想的模型：刚好在欠拟合和过拟合的界线上，在容量不足和容量过大的界线上。
- 要搞清楚你需要多大的模型，就必须开发一个过拟合的模型。
 - (1)添加更多的层。
 - (2)让每一层变得更大。
 - (3)训练更多的轮次。
- 要始终监控训练损失和验证损失，以及你所关心的指标的训练值和验证值。如果你发现模型在验证数据上的性能开始下降，那么就出现了过拟合。

模型正则化与调节超参数

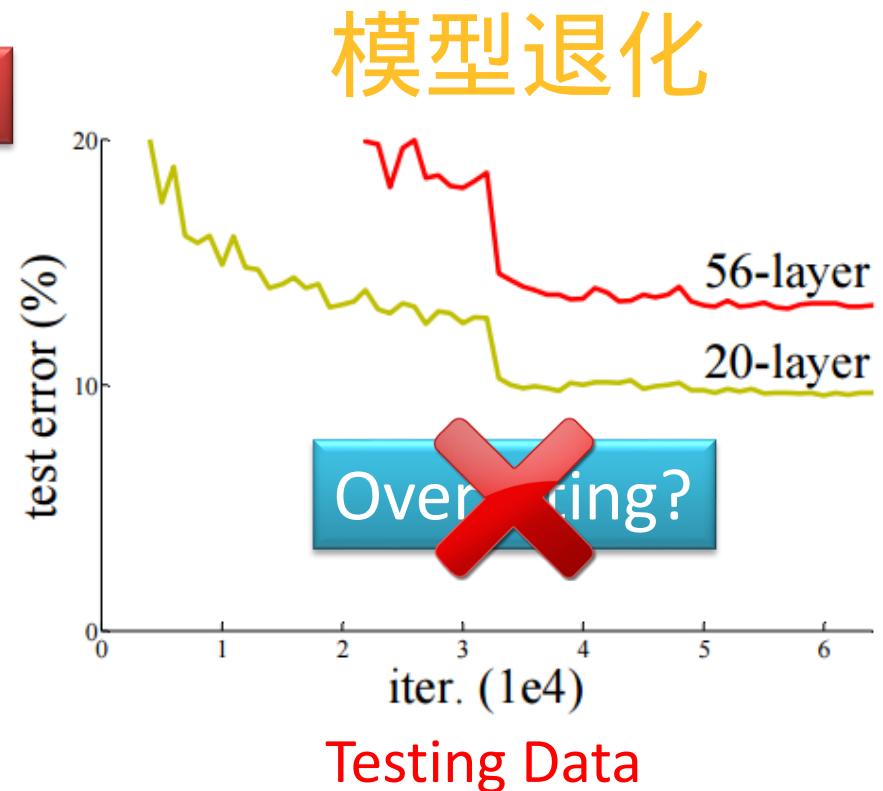
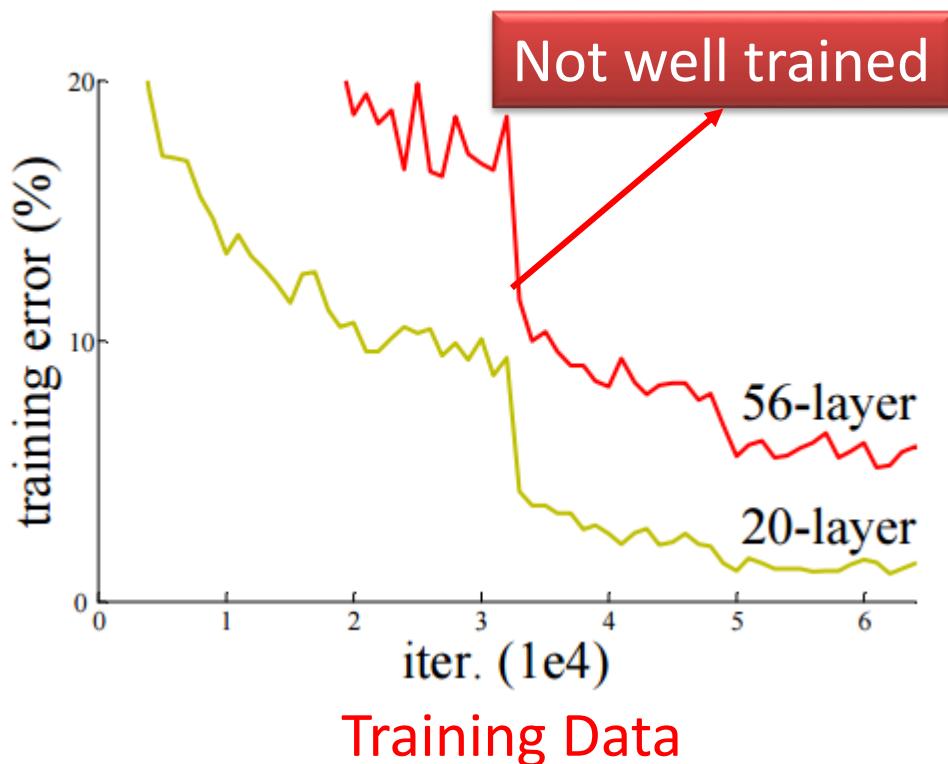
- 这一步是最费时间的：你将不断地调节模型的超参数、训练模型、在验证数据上评估（这里不是测试数据）、再次调节模型，然后重复这一过程，直到模型达到最佳性能。
- 你应该尝试以下几项：
 - 添加dropout。
 - 尝试不同的架构：增加或减少层数。
 - 添加L1和/或L2正则化。
 - 尝试不同的超参数（比如每层的单元个数或优化器的学习率），以找到最佳配置。
 - （可选）反复做特征工程：添加新特征或删除没有信息量的特征。
- 请注意：每次使用验证过程的反馈来调节模型，都会将有关验证过程的信息泄露到模型中。如果只重复几次，那么无关紧要；但如果系统性地迭代许多次，最终会导致模型对验证过程过拟合（即使模型并没有直接在验证数据上训练）。这会降低验证过程的可靠性。
- 一旦开发出令人满意的模型配置，你就可以在所有可用数据（训练数据+验证数据）上训练最终的生产模型，然后在测试集上最后评估一次。
- 如果测试集上的性能比验证集上差很多，那么这可能意味着你的验证流程不可靠，或者你在调节模型参数时在验证数据上出现了过拟合。在这种情况下，你可能需要换用更加可靠的评估方法，比如重复的K折验证。

Under-fitting and Overfitting for Deep Learning

Recipe of Deep Learning



Do not always blame Overfitting

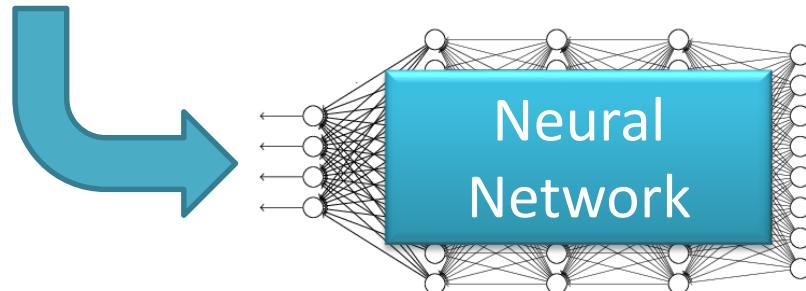


Deep Residual Learning for Image Recognition
<http://arxiv.org/abs/1512.03385>

Recipe of Deep Learning

Different approaches for different problems.

e.g. dropout for good results on testing data



Good Results on Testing Data?

Good Results on Training Data?



YES



Recipe for underfitting problem of Deep Learning

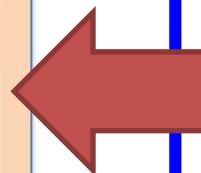
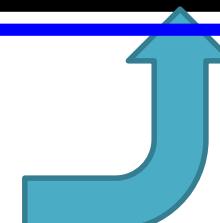


YES

Good Results on
Testing Data?

YES

Good Results on
Training Data?



Choosing proper loss

Mini-batch & Batch Norm

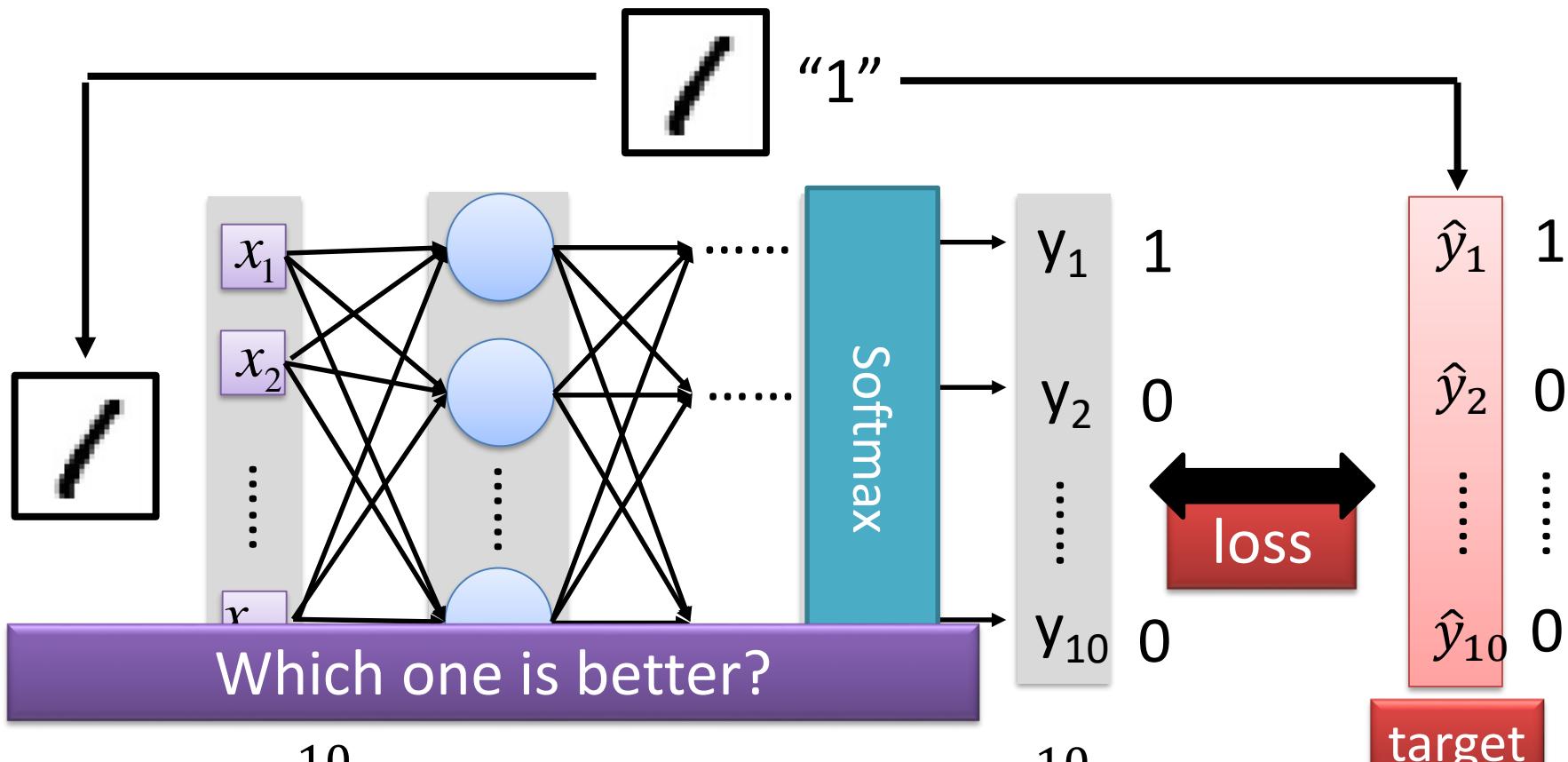
New activation function

Adaptive Learning Rate

Momentum

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

Choosing Proper Loss



Square
Error

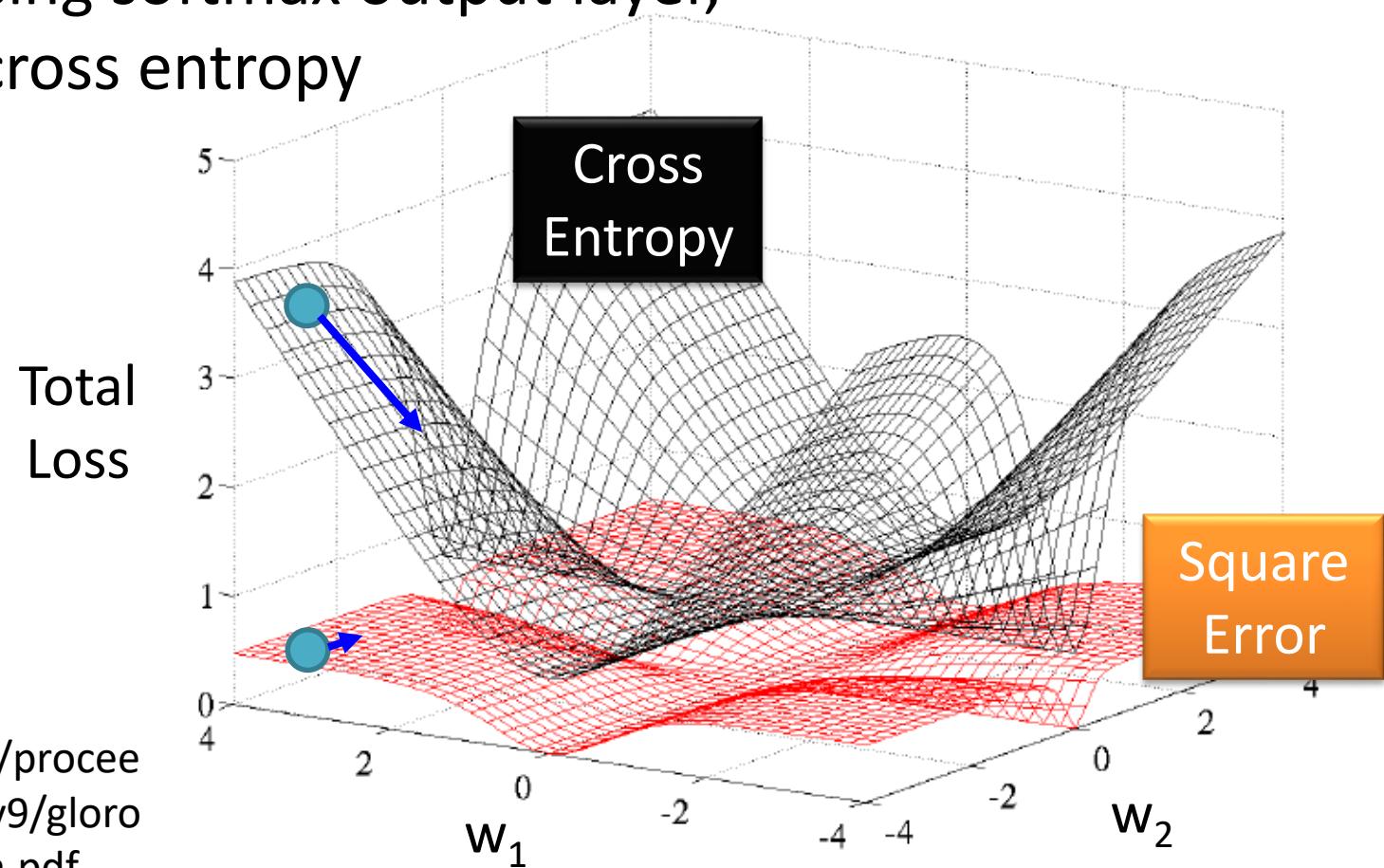
$$\sum_{i=1}^{10} (y_i - \hat{y}_i)^2 = 0$$

Cross
Entropy

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

Choosing Proper Loss

When using softmax output layer,
choose cross entropy



Demo

Square Error

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

Cross Entropy

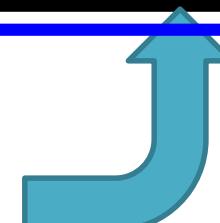
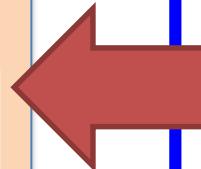
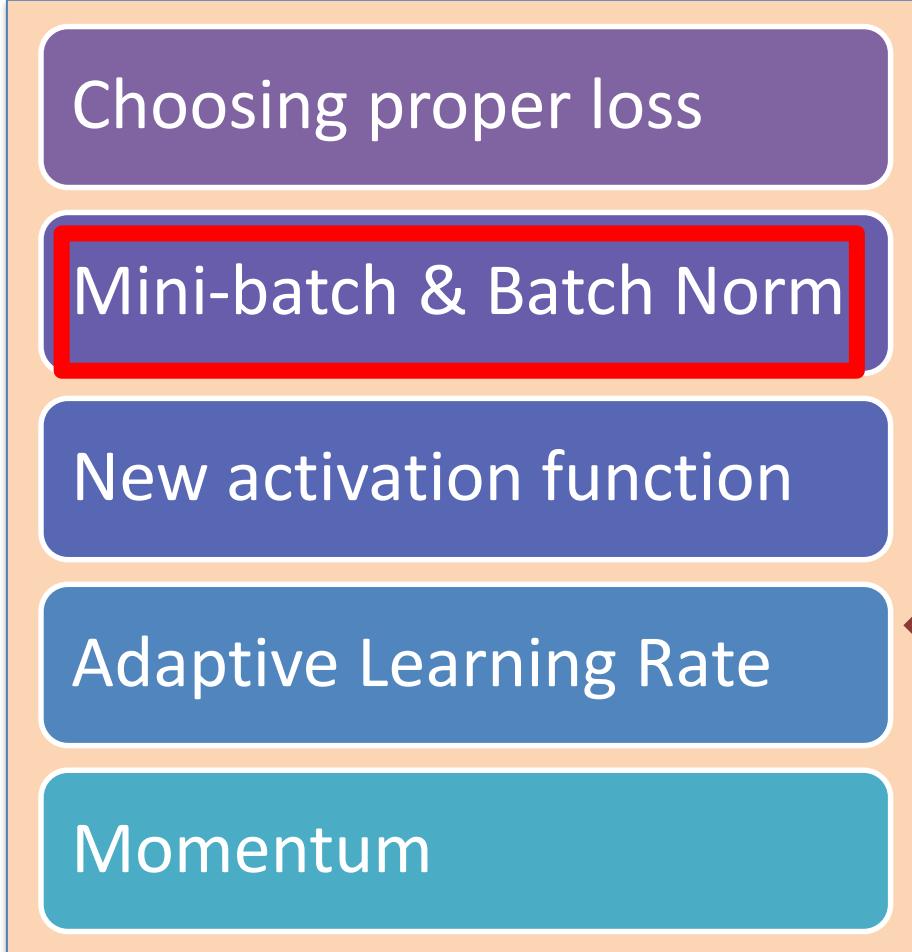
```
model.compile(loss='categorical_crossentropy',  
              optimizer=SGD(lr=0.1),  
              metrics=['accuracy'])
```

Several alternatives: <https://keras.io/objectives/>

Recipe for underfitting problem of Deep Learning



YES

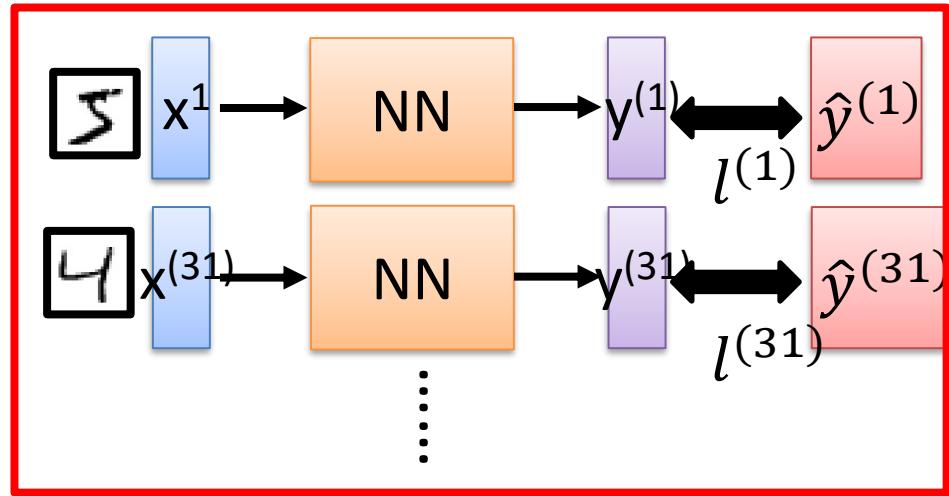


```
model.fit(x_train, y_train, batch_size=100, nb_epoch=20)
```

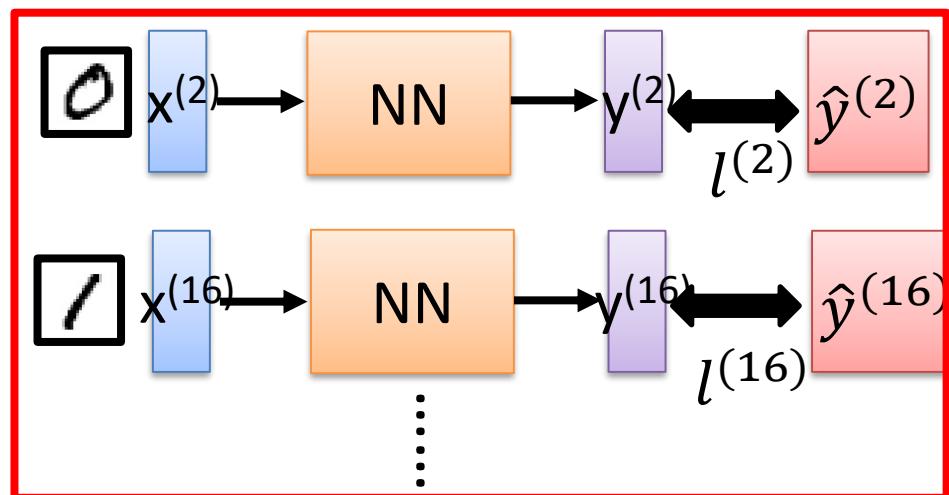
We do not really minimize total loss!

Mini-batch

Mini-batch



Mini-batch



- Randomly initialize network parameters
- Pick the 1st batch
 $L' = l^{(1)} + l^{(31)} + \dots$
Update parameters once
- Pick the 2nd batch
 $L'' = l^{(2)} + l^{(16)} + \dots$
Update parameters once
- ⋮
- Until all mini-batches have been picked

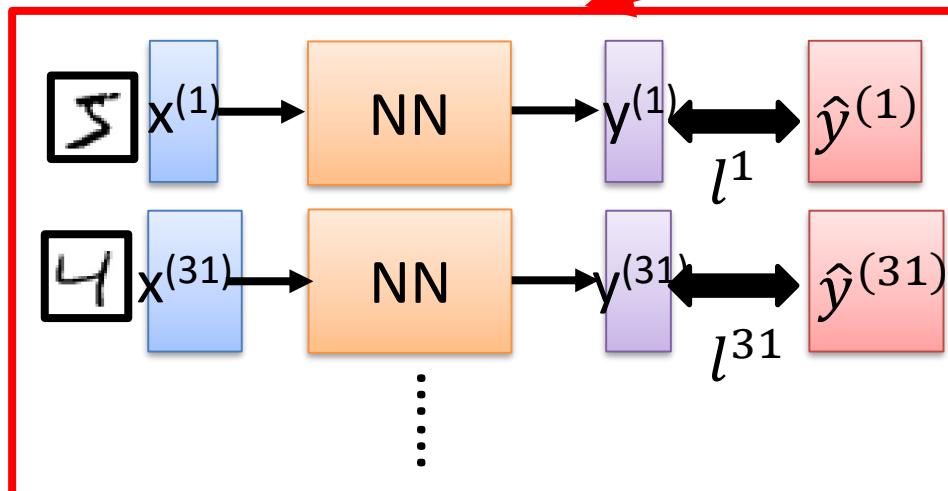
one epoch

Repeat the above process

Mini-batch

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

Mini-batch



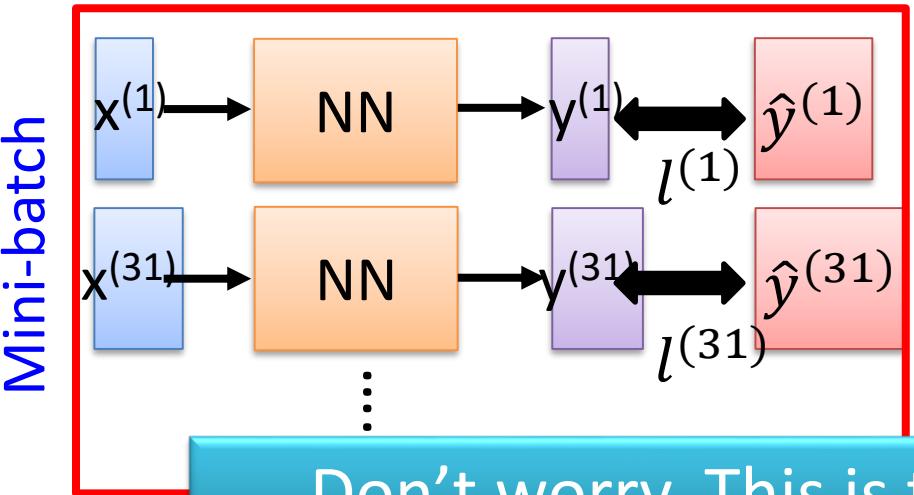
100 examples in a mini-batch

Repeat 20 times

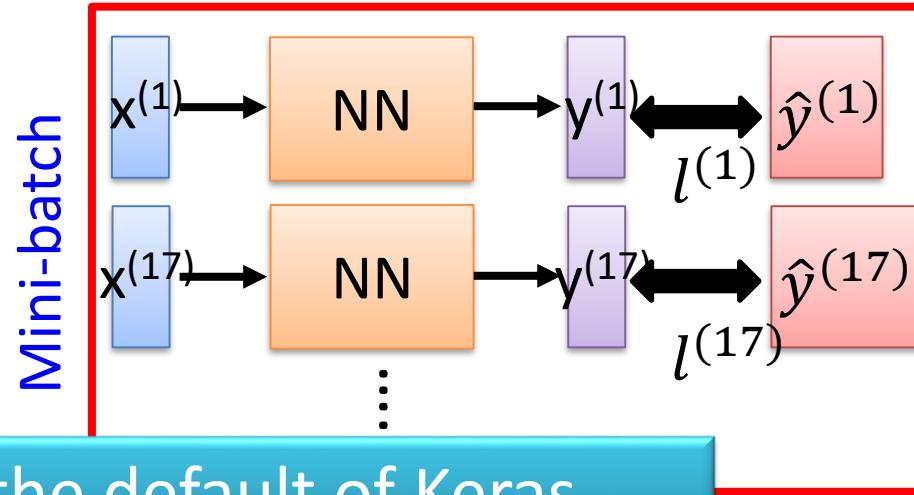
- Pick the 1st batch
 $L' = l^{(1)} + l^{(31)} + \dots$
Update parameters once
 - Pick the 2nd batch
 $L'' = l^{(2)} + l^{(16)} + \dots$
Update parameters once
 - ⋮
 - Until all mini-batches have been picked
- one epoch

Shuffle the training examples for each epoch

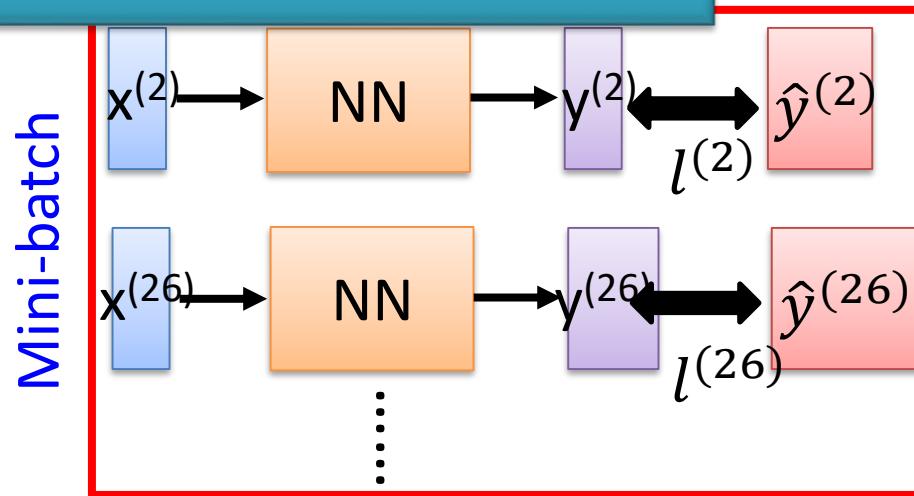
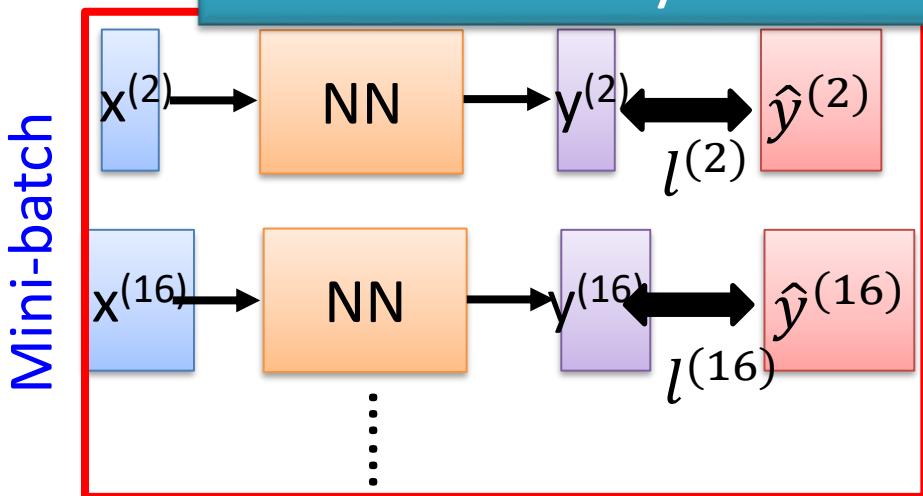
Epoch 1



Epoch 2



Don't worry. This is the default of Keras.



Batch Normalization

- 为什么要进行 Batch Normalization ?
 - 神经网络的训练目标是在输出层得到原始输入数据分布的一个映射，即在训练过程中，应该力图保证数据分布的映射关系；若数据分布在训练过程中发生了偏移，则会降低网络的泛化能力；
 - 在网络训练过程中，后一层网络的输入是前一层的输出，因此，前一层网络参数的变化，将导致后一层输入数据分布的改变，且这种改变会在训练过程中向后传递并被逐步放大；这种在训练过程中，数据分布的改变称为“Internal Covariate Shift”；
 - 在 Batch-based Training 中，若个 Batch 的分布各不相同，网络需在每个 Batch 的训练中适应不同的分布，从而大大降低训练速度；
- 针对每层网络的每个 Batch 进行Batch Normalization. 可有效提高训练速度和效果。

激活之前归一化处理

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_1 \dots m\}$;

Parameters to be learned: γ, β

Output: $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i \quad \begin{matrix} \text{减去均值} \\ \text{除以方差} \\ \text{达到标准化} \end{matrix} \quad // \text{mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \quad // \text{mini-batch variance}$$

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad // \text{normalize}$$

$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i) \quad // \text{scale and shift}$$

加了缩放和偏移，通过学习设定
Keras自动处理

反向传播阶段的计算：

$$\frac{\partial \ell}{\partial \hat{x}_i} = \frac{\partial \ell}{\partial y_i} \cdot \gamma$$

$$\frac{\partial \ell}{\partial \sigma_{\mathcal{B}}^2} = \sum_{i=1}^m \frac{\partial \ell}{\partial \hat{x}_i} \cdot (x_i - \mu_{\mathcal{B}}) \cdot \frac{-1}{2} (\sigma_{\mathcal{B}}^2 + \epsilon)^{-3/2}$$

$$\frac{\partial \ell}{\partial \mu_{\mathcal{B}}} = \left(\sum_{i=1}^m \frac{\partial \ell}{\partial \hat{x}_i} \cdot \frac{-1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \right) + \frac{\partial \ell}{\partial \sigma_{\mathcal{B}}^2} \cdot \frac{\sum_{i=1}^m -2(x_i - \mu_{\mathcal{B}})}{m}$$

$$\frac{\partial \ell}{\partial x_i} = \frac{\partial \ell}{\partial \hat{x}_i} \cdot \frac{1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} + \frac{\partial \ell}{\partial \sigma_{\mathcal{B}}^2} \cdot \frac{2(x_i - \mu_{\mathcal{B}})}{m} + \frac{\partial \ell}{\partial \mu_{\mathcal{B}}} \cdot \frac{1}{m}$$

$$\frac{\partial \ell}{\partial \gamma} = \sum_{i=1}^m \frac{\partial \ell}{\partial y_i} \cdot \hat{x}_i$$

$$\frac{\partial \ell}{\partial \beta} = \sum_{i=1}^m \frac{\partial \ell}{\partial y_i}$$

数据测试阶段：

因为测试阶段输入数据可能只有一个，因此，我们使用所有 Batch 的 μ_B 的期望值代替上述公式中的 $E[x]$ ；使用所有 Batch 方差 δ_B^2 的无偏估计代替上述公式中的 $\text{Var}[x]$ ，即：

$$E[x] \leftarrow E_{\mathcal{B}}[\mu_{\mathcal{B}}]$$

$$\text{Var}[x] \leftarrow \frac{m}{m-1} E_{\mathcal{B}}[\sigma_{\mathcal{B}}^2]$$

又因为，上述公式中：

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$$

则，Batch Normalization 层计算的公式为：

$$y = \frac{\gamma}{\sqrt{\text{Var}[x]+\epsilon}} \cdot x + \left(\beta - \frac{\gamma E[x]}{\sqrt{\text{Var}[x]+\epsilon}} \right)$$

Recipe for underfitting problem of Deep Learning



YES

Choosing proper loss

Mini-batch & Batch Norm

New activation function

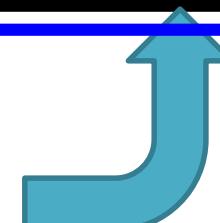
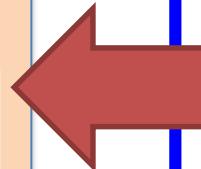
Adaptive Learning Rate

Momentum

Good Results on
Testing Data?

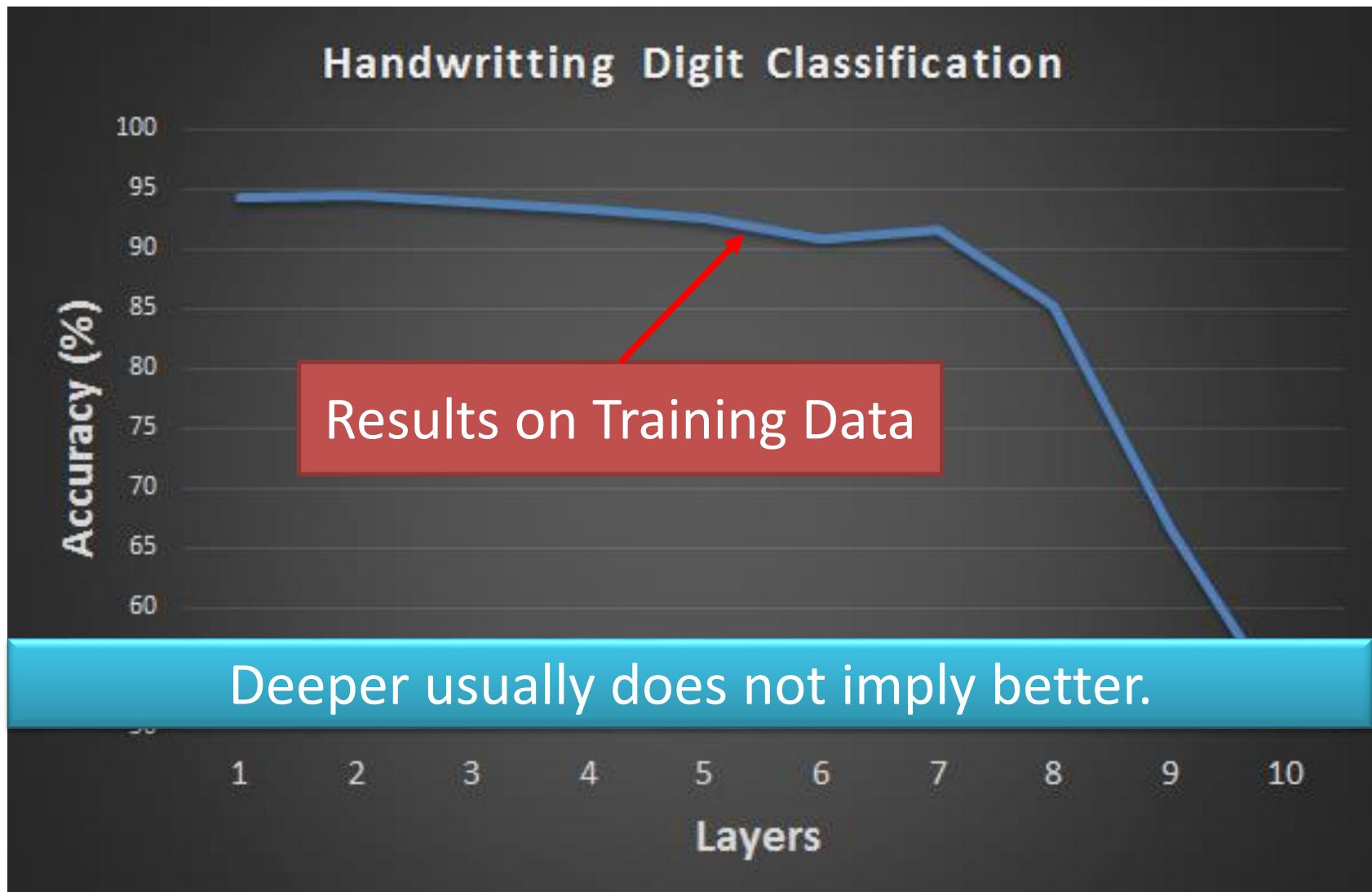
YES

Good Results on
Training Data?

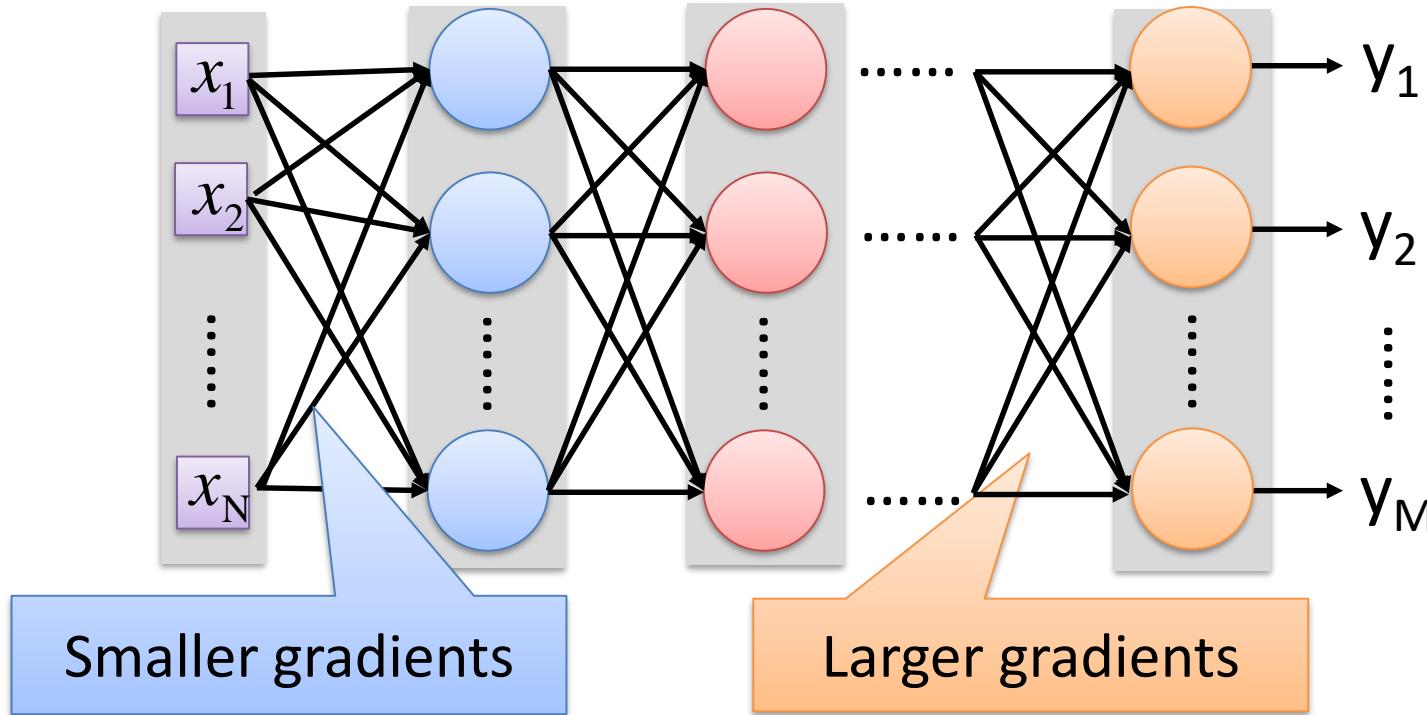


```
model.fit(x_train, y_train, batch_size=100, nb_epoch=20)
```

Hard to get the power of Deep ...

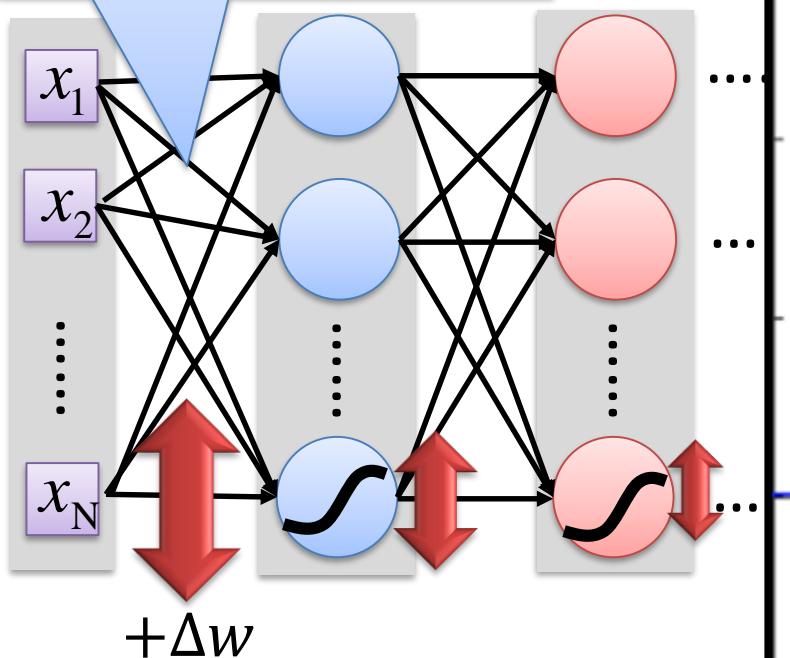


Vanishing Gradient Problem

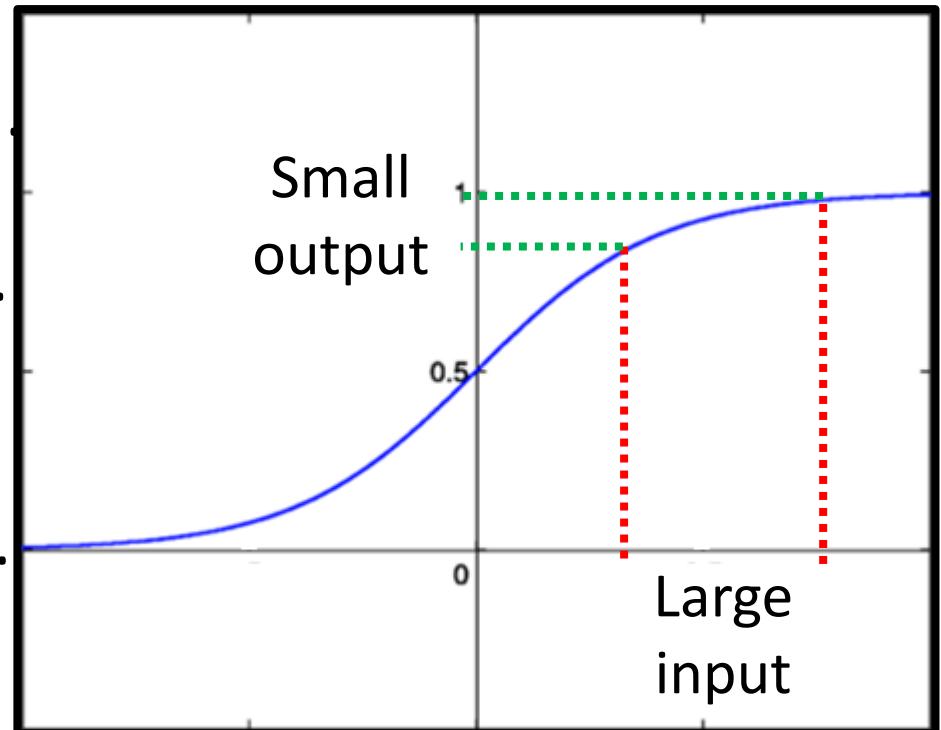


梯度消失问题 Vanishing Gradient Problem

Smaller gradients



Sigmoid激活函数

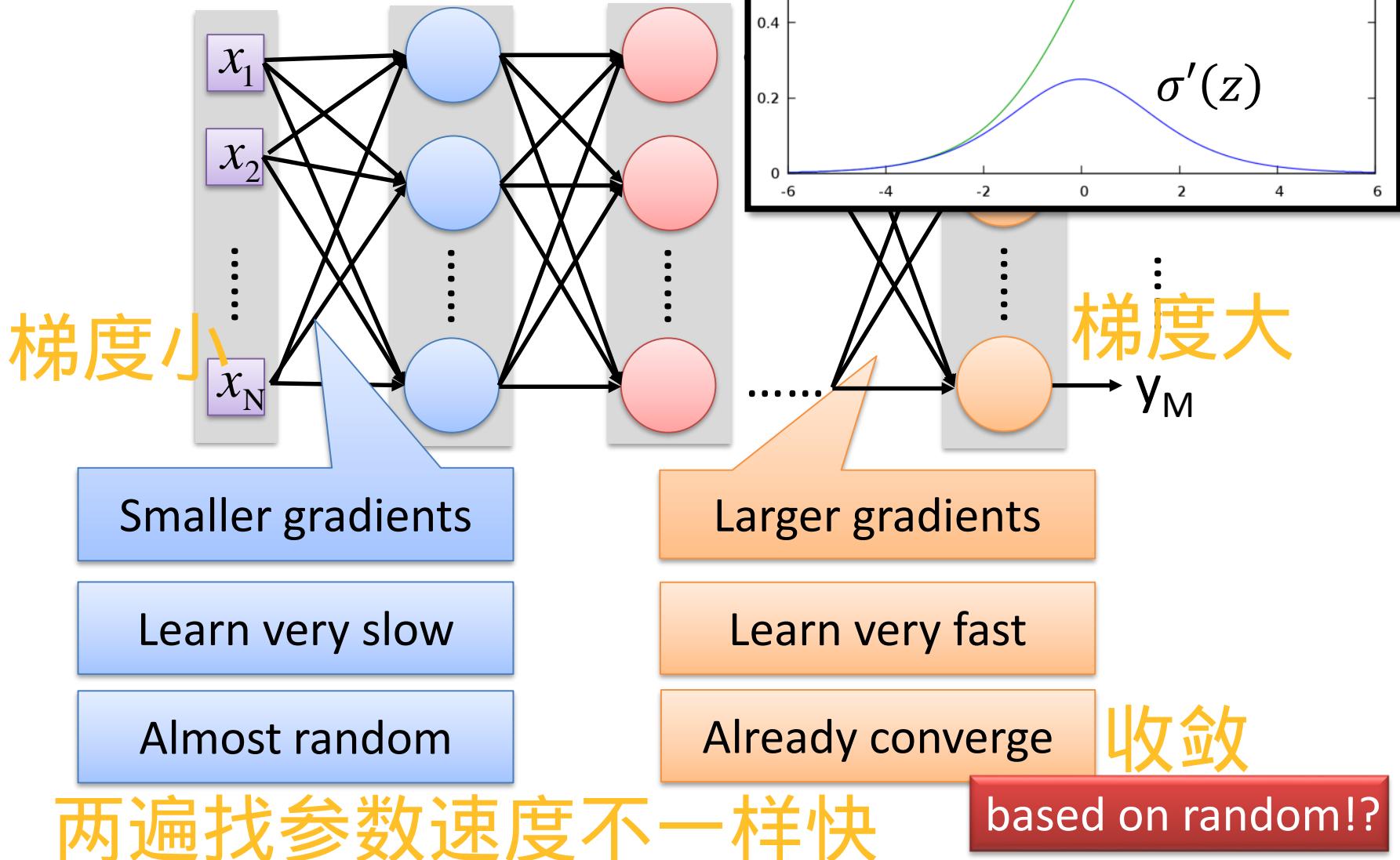


Intuitive way to compute the derivatives ...

直观

$$\frac{\partial l}{\partial w} = ? \quad \frac{\Delta l}{\Delta w}$$

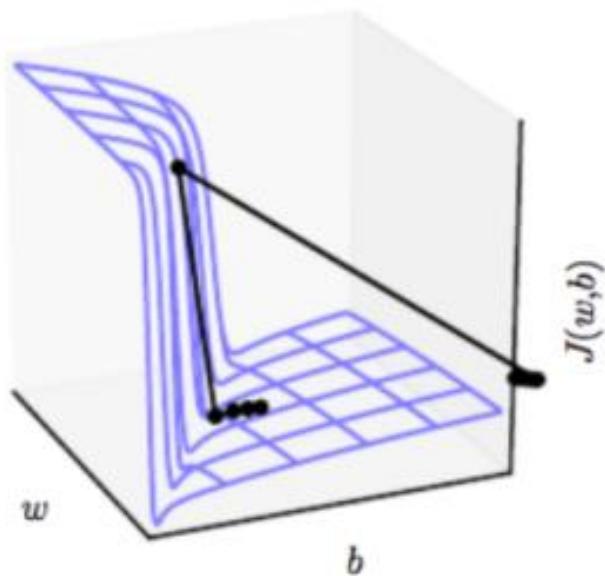
Vanishing Gradient



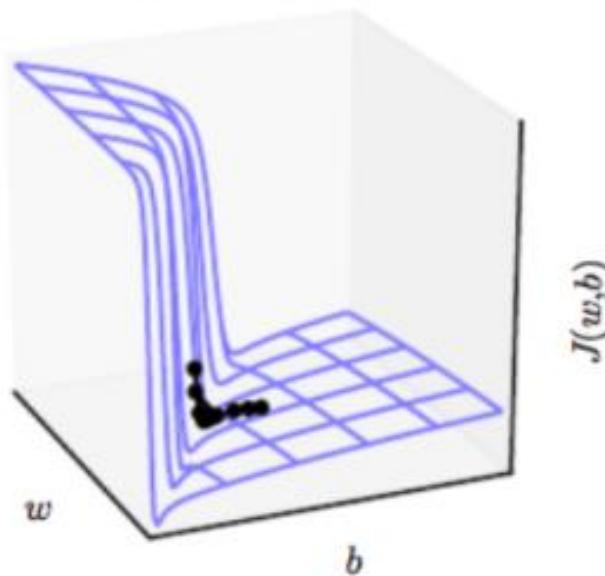
Exploding Gradient

梯度爆炸问题

Without clipping



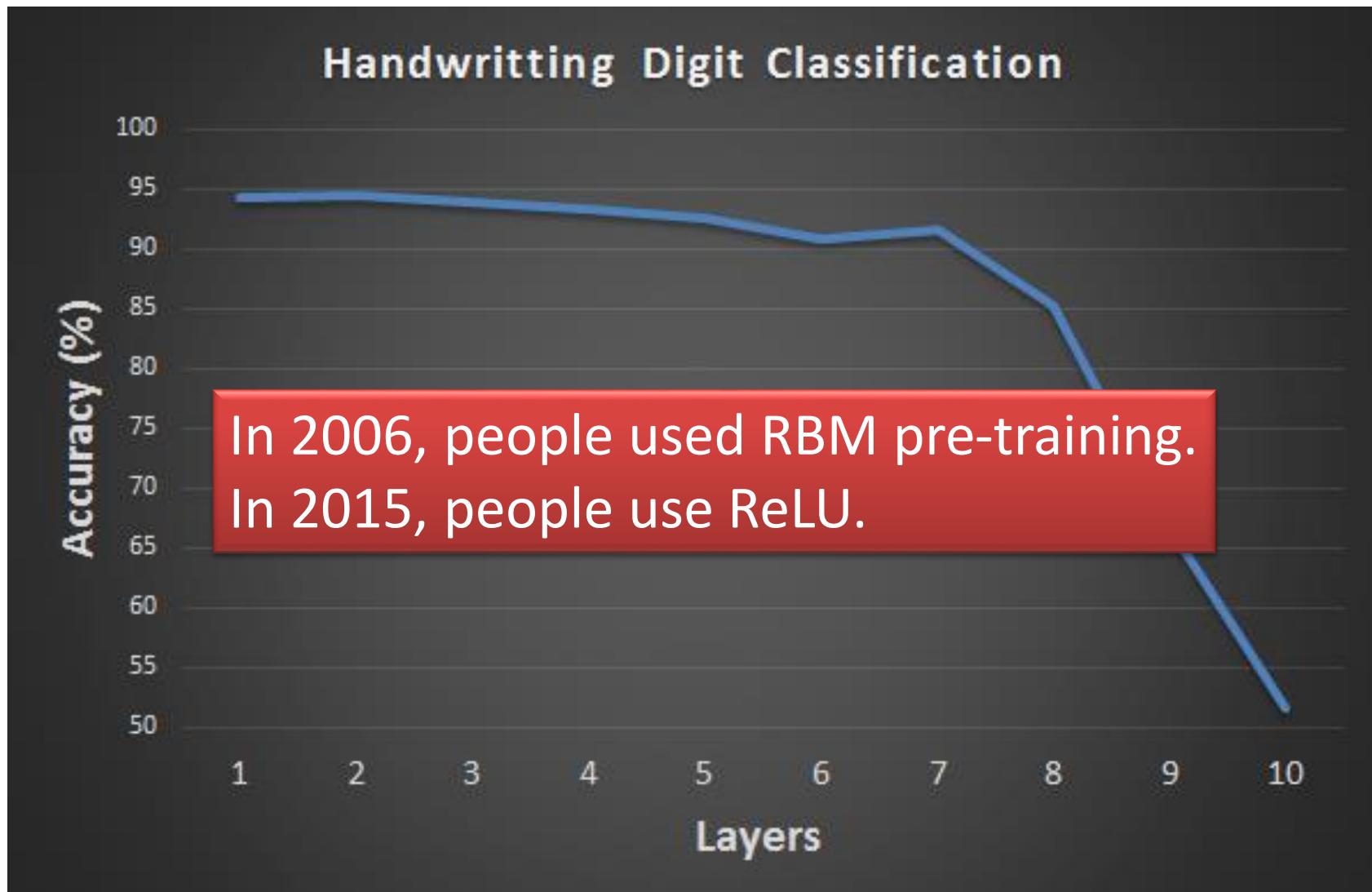
With clipping



强制截断

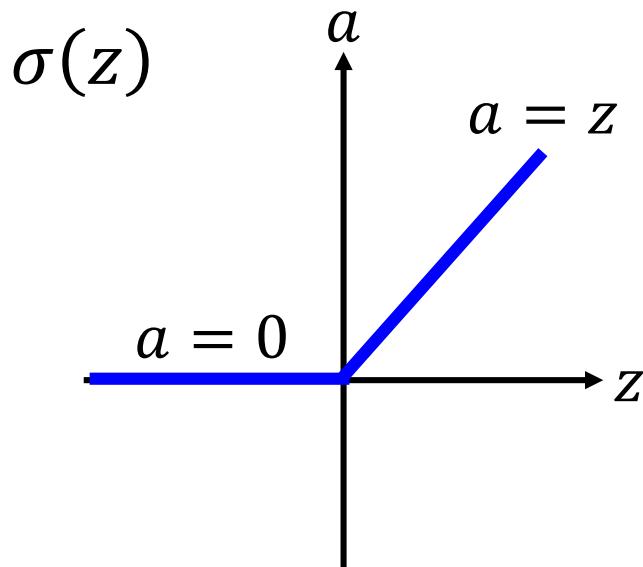
— Goodfellow et al., Deep Learning

Hard to get the power of Deep ...



ReLU

- Rectified Linear Unit (ReLU)

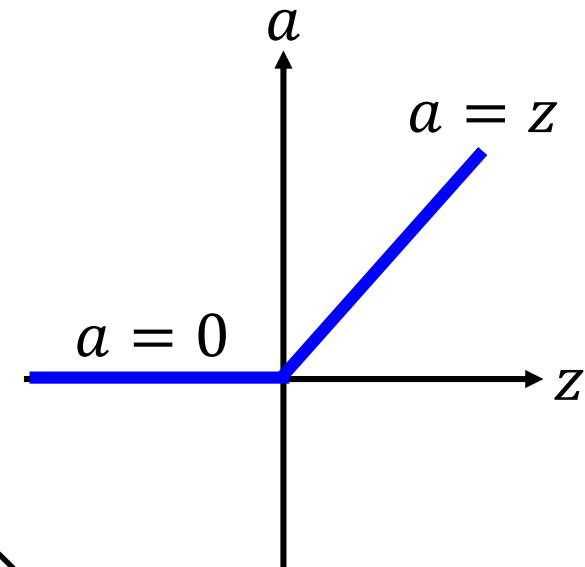
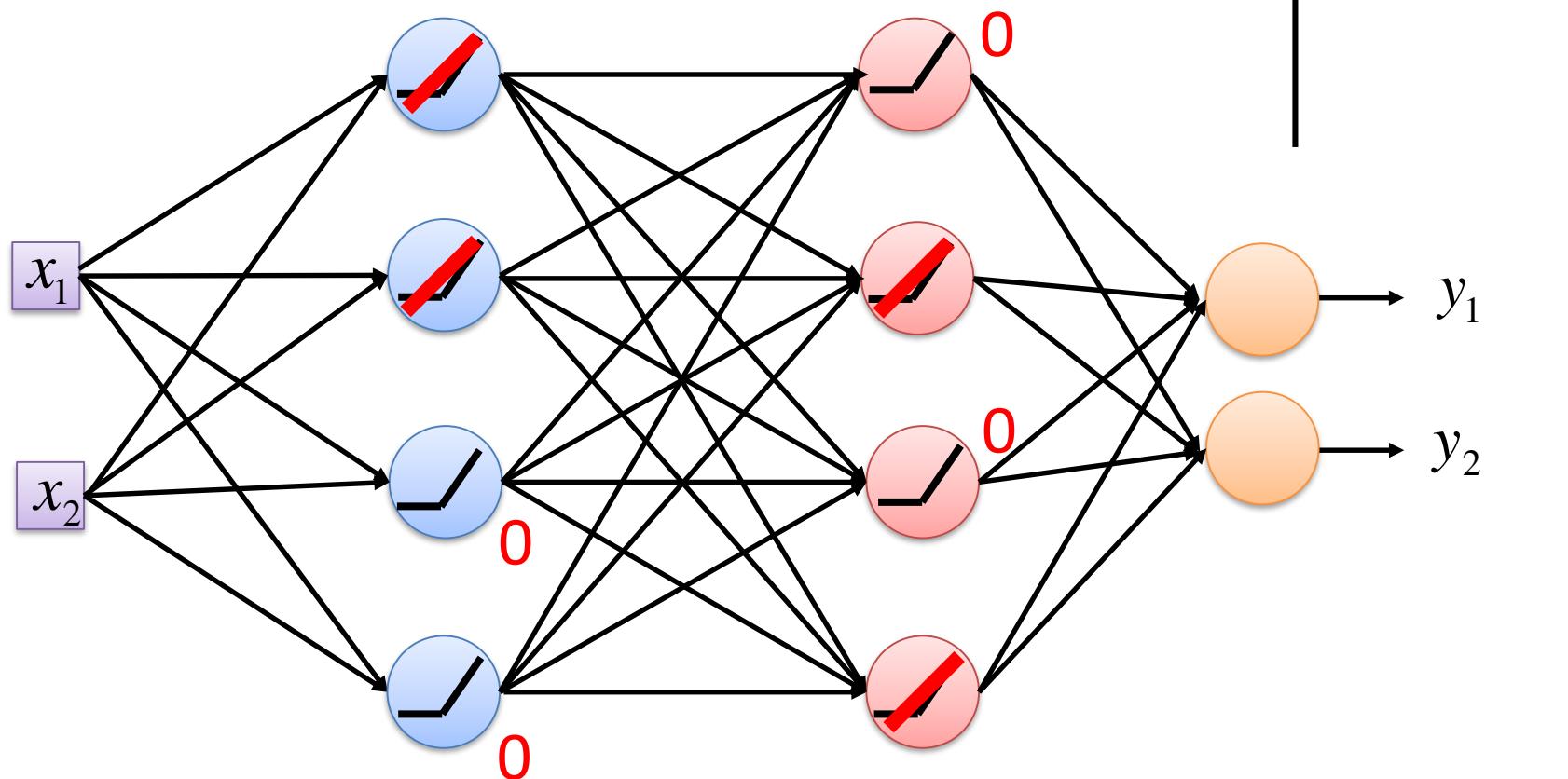


[Xavier Glorot, AISTATS'11]
[Andrew L. Maas, ICML'13]
[Kaiming He, arXiv'15]

Reason:

- Fast to compute
计算方便
- Biological reason
- Infinite sigmoid
with different biases
- Vanishing gradient
problem

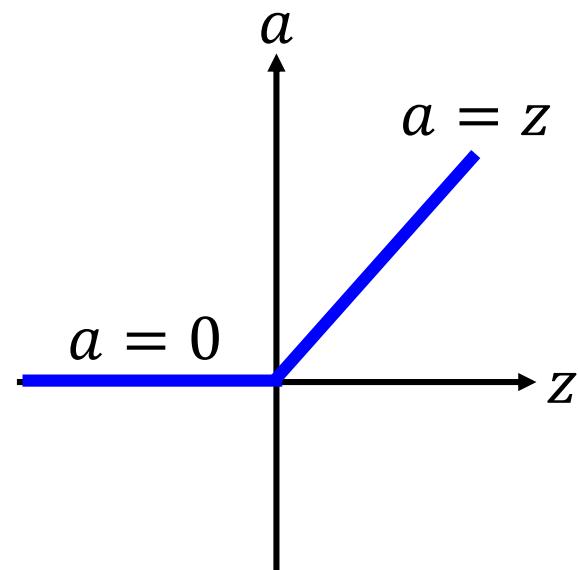
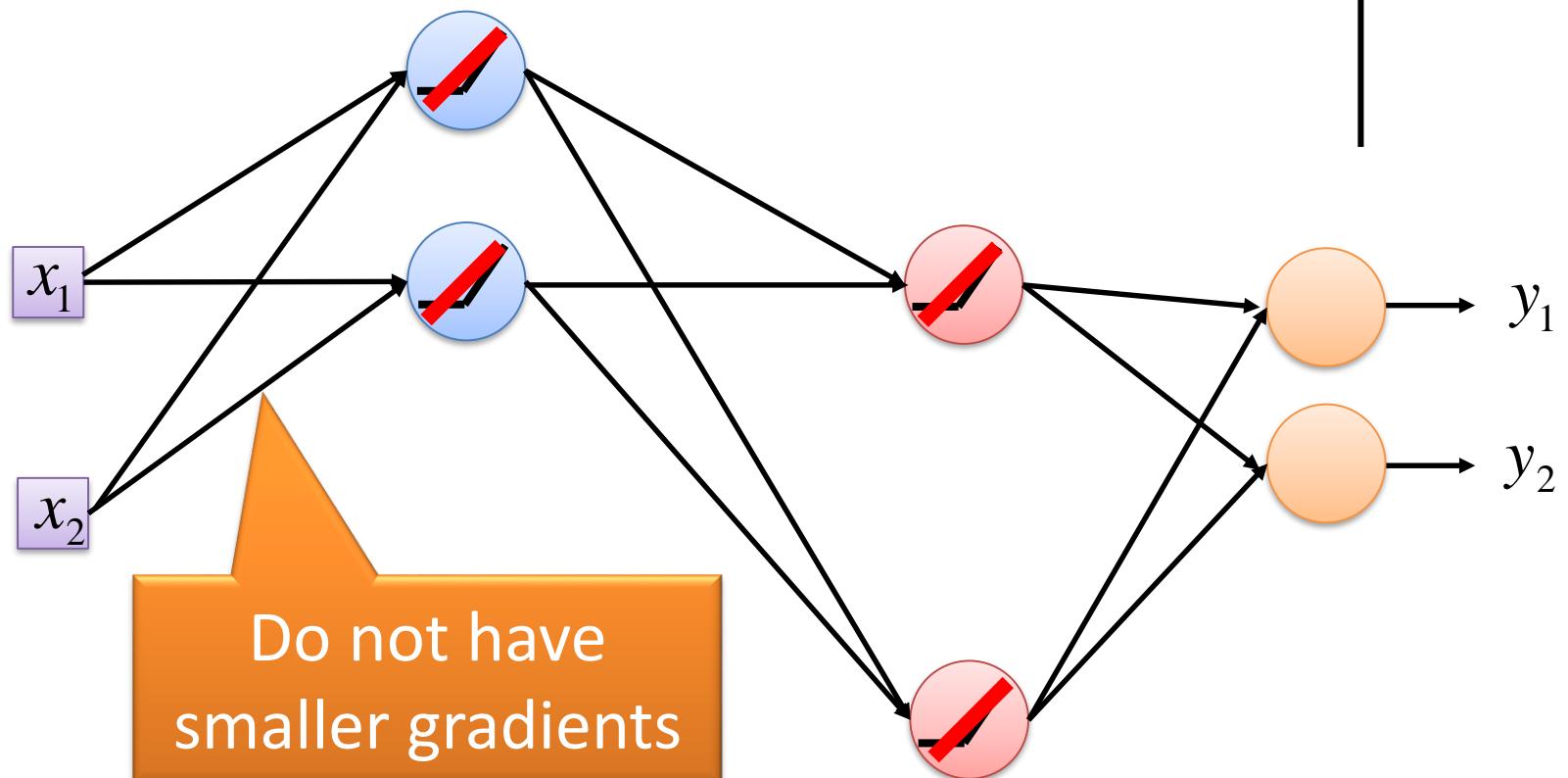
ReLU



ReLU

让网络更加简单

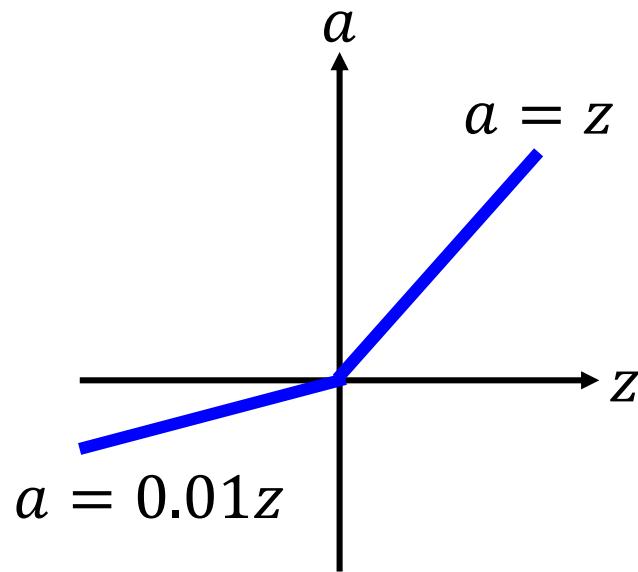
A Thinner linear network



ReLU - variant

变异

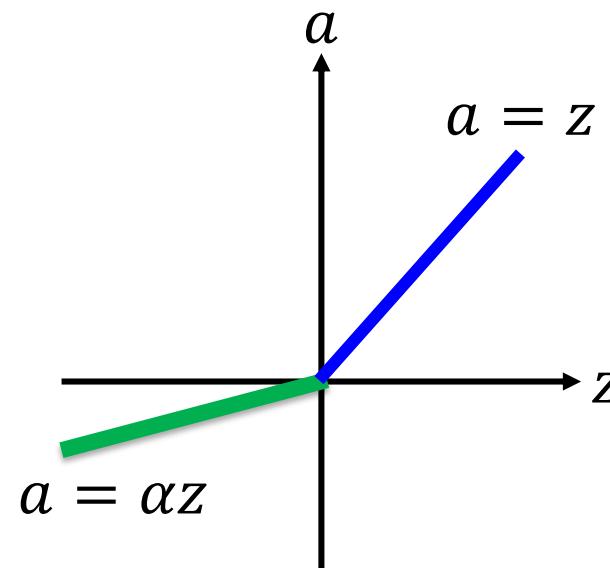
Leaky ReLU



Exponential Linear Unit (ELU)

[FAST AND ACCURATE DEEP NETWORK LEARNING BY EXPONENTIAL LINEAR UNITS \(ELUS\) \(2016, Djork-Arn'e Clevert, Thomas Unterthiner & Sepp Hochreiter\)](#)

Parametric ReLU

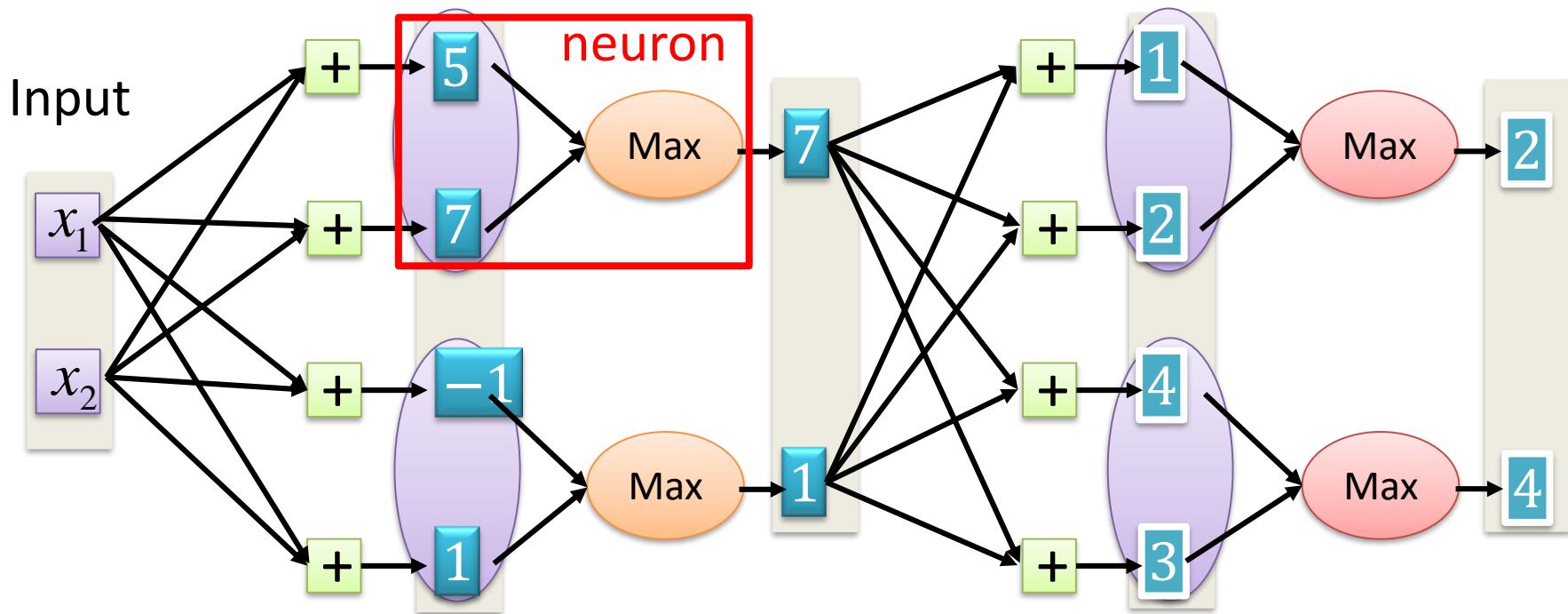


α also learned by gradient descent

Maxout

ReLU is a special cases of Maxout

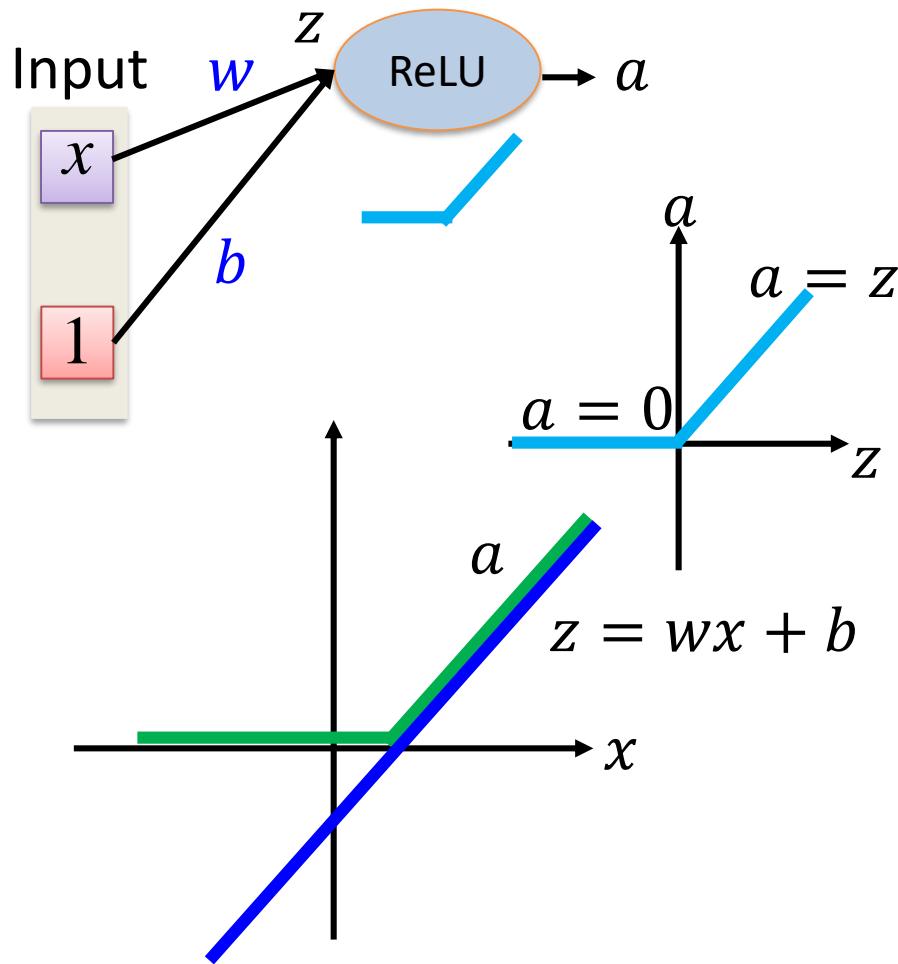
- Learnable activation function [Ian J. Goodfellow, ICML'13]



You can have more than 2 elements in a group.

Maxout

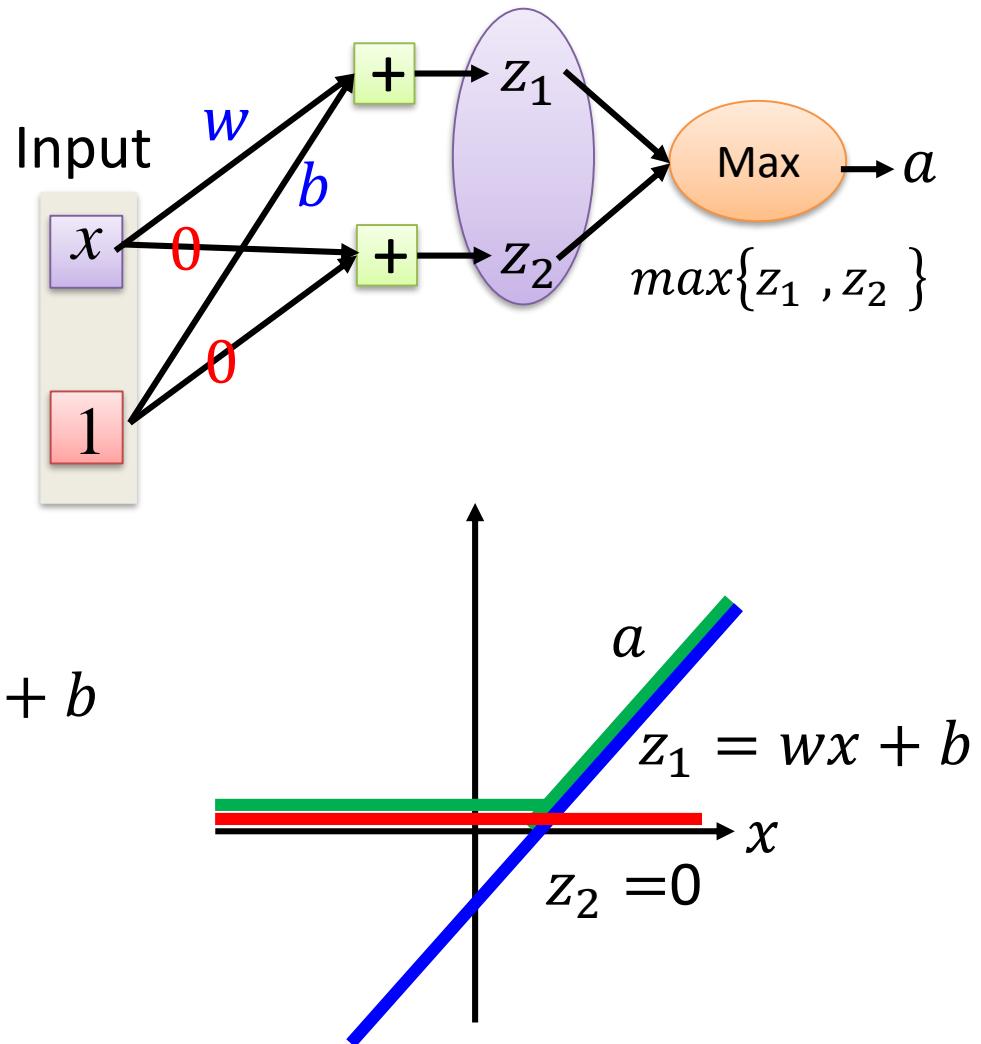
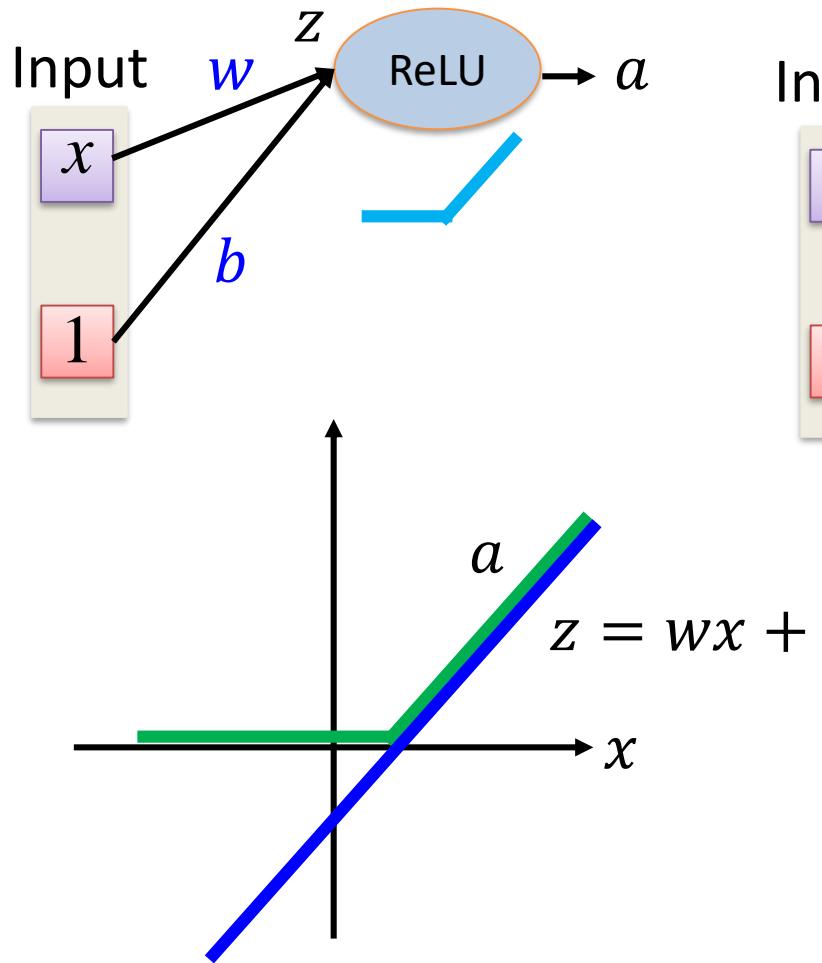
ReLU is a special cases of Maxout



$$a = \max\{ 0, x \}$$

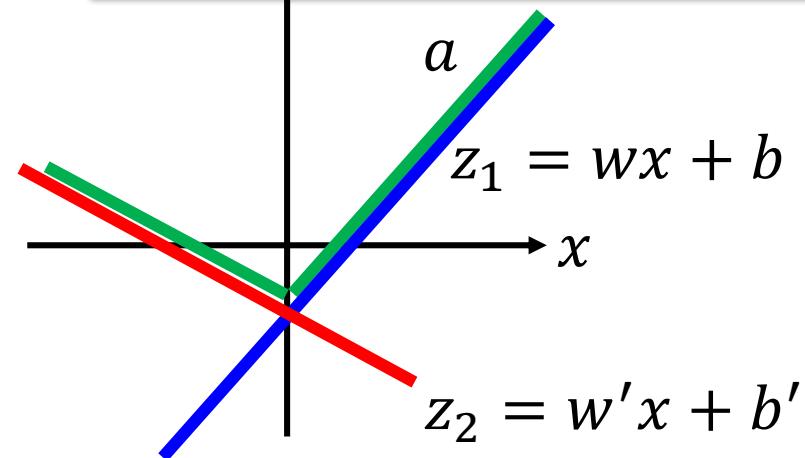
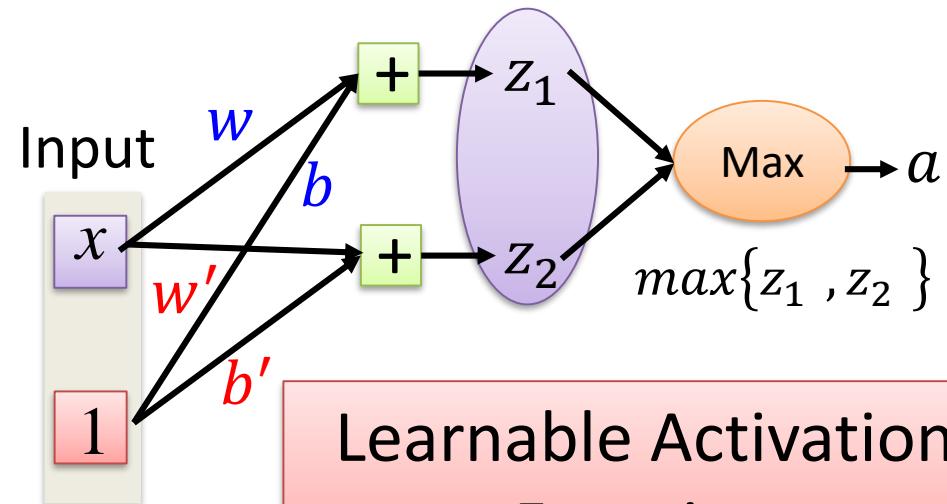
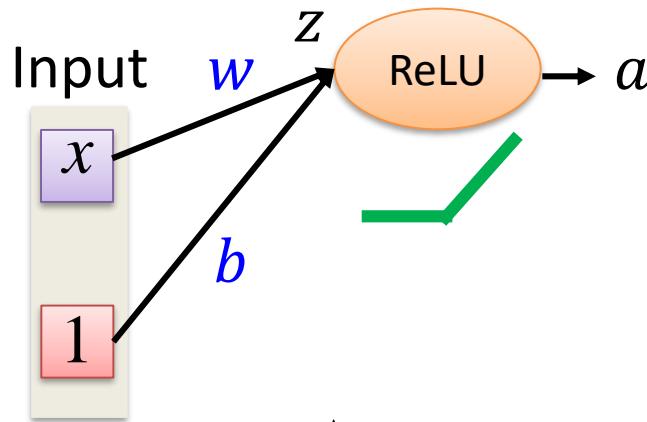
Maxout

ReLU is a special cases of Maxout



Maxout

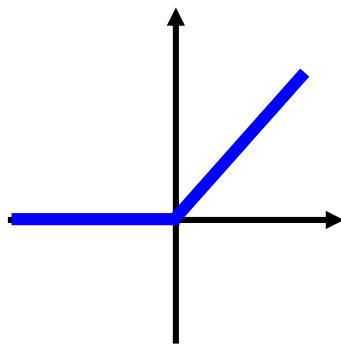
More than ReLU



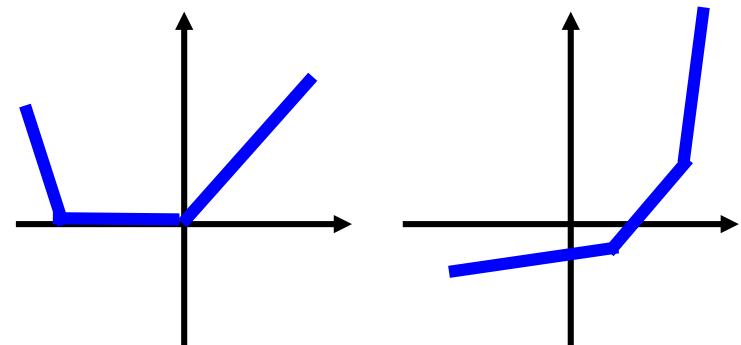
Maxout

- Learnable activation function [Ian J. Goodfellow, ICML'13]
 - Activation function in maxout network can be any piecewise linear convex function
 - How many pieces depending on how many elements in a group

2 elements in a group

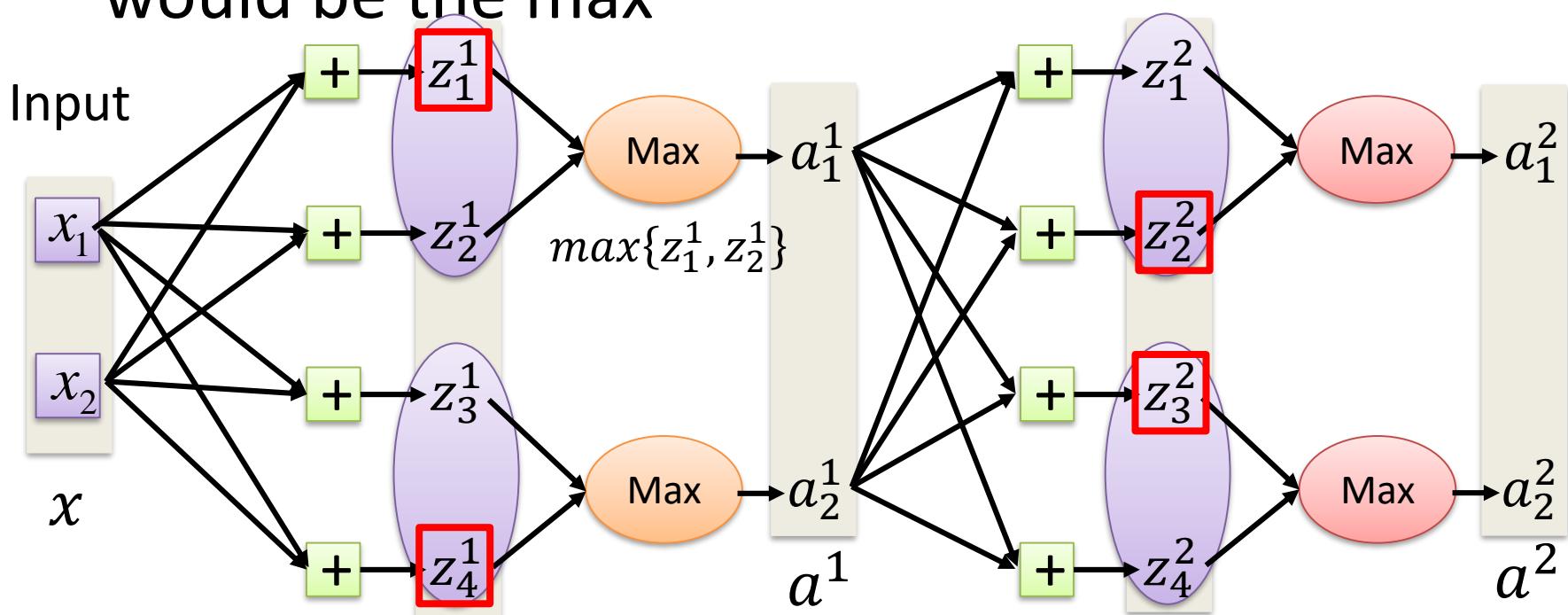


3 elements in a group



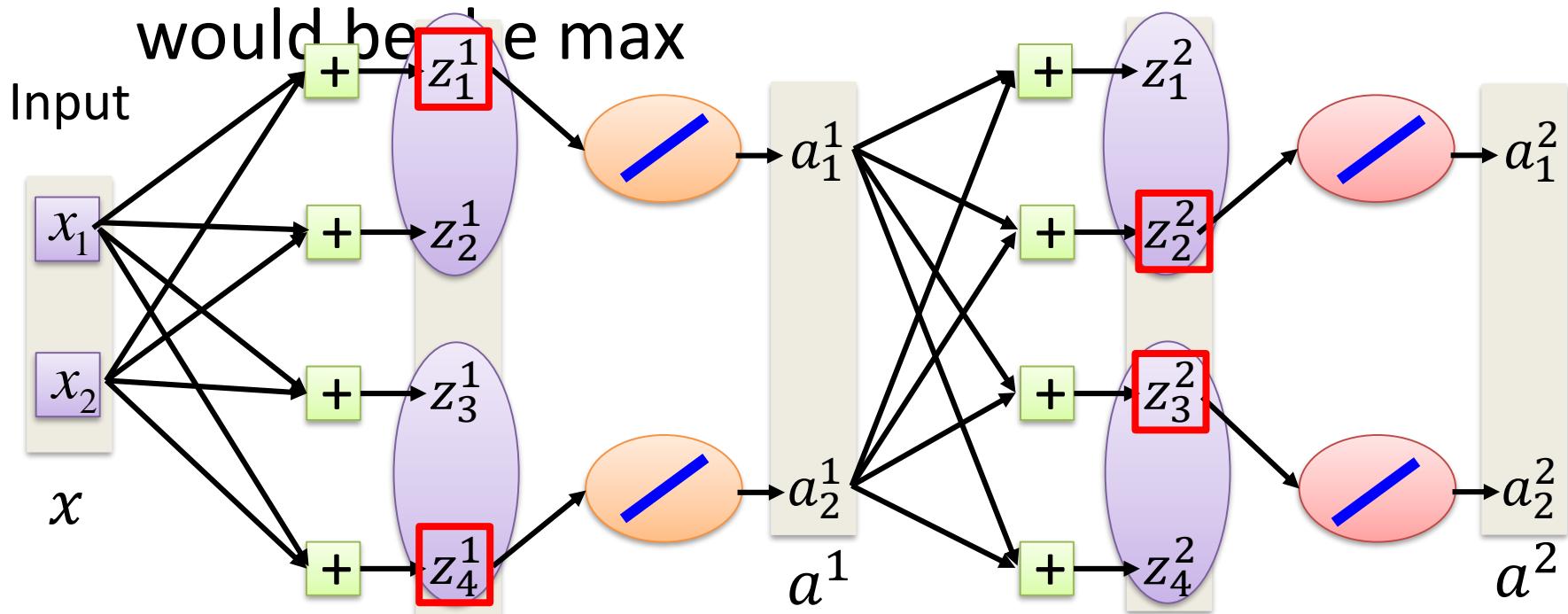
Maxout - Training

- Given a training data x , we know which z would be the max



Maxout - Training

- Given a training data x , we know which z would be the max



- Train this thin and linear network

Different thin and linear network for different examples

Recipe for underfitting problem of Deep Learning



YES

Choosing proper loss

Mini-batch & Batch Norm

New activation function

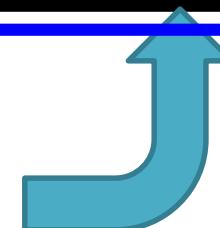
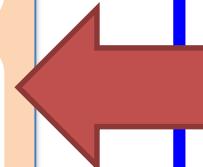
Adaptive Learning Rate

Momentum

Good Results on
Testing Data?

YES

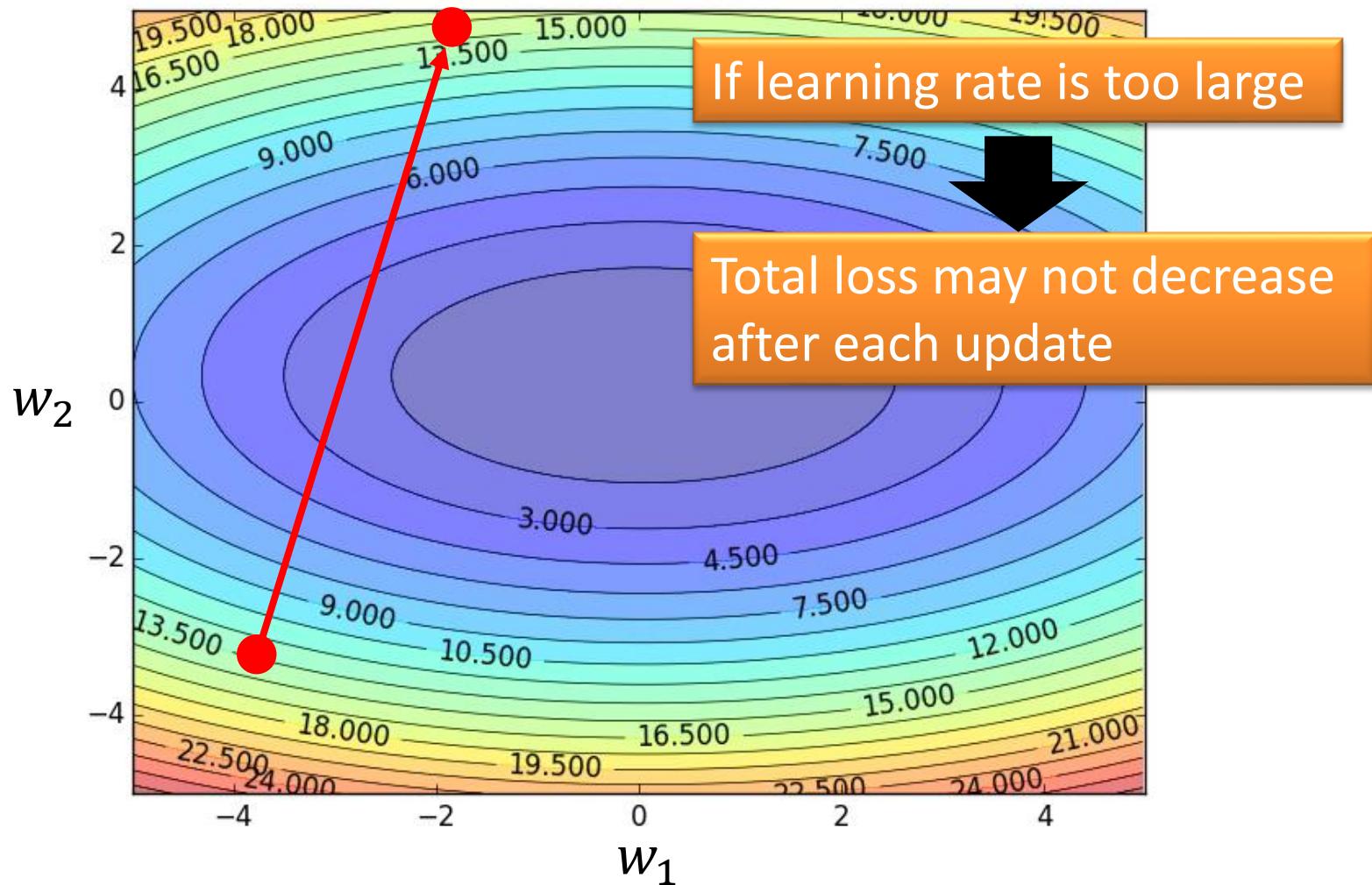
Good Results on
Training Data?



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

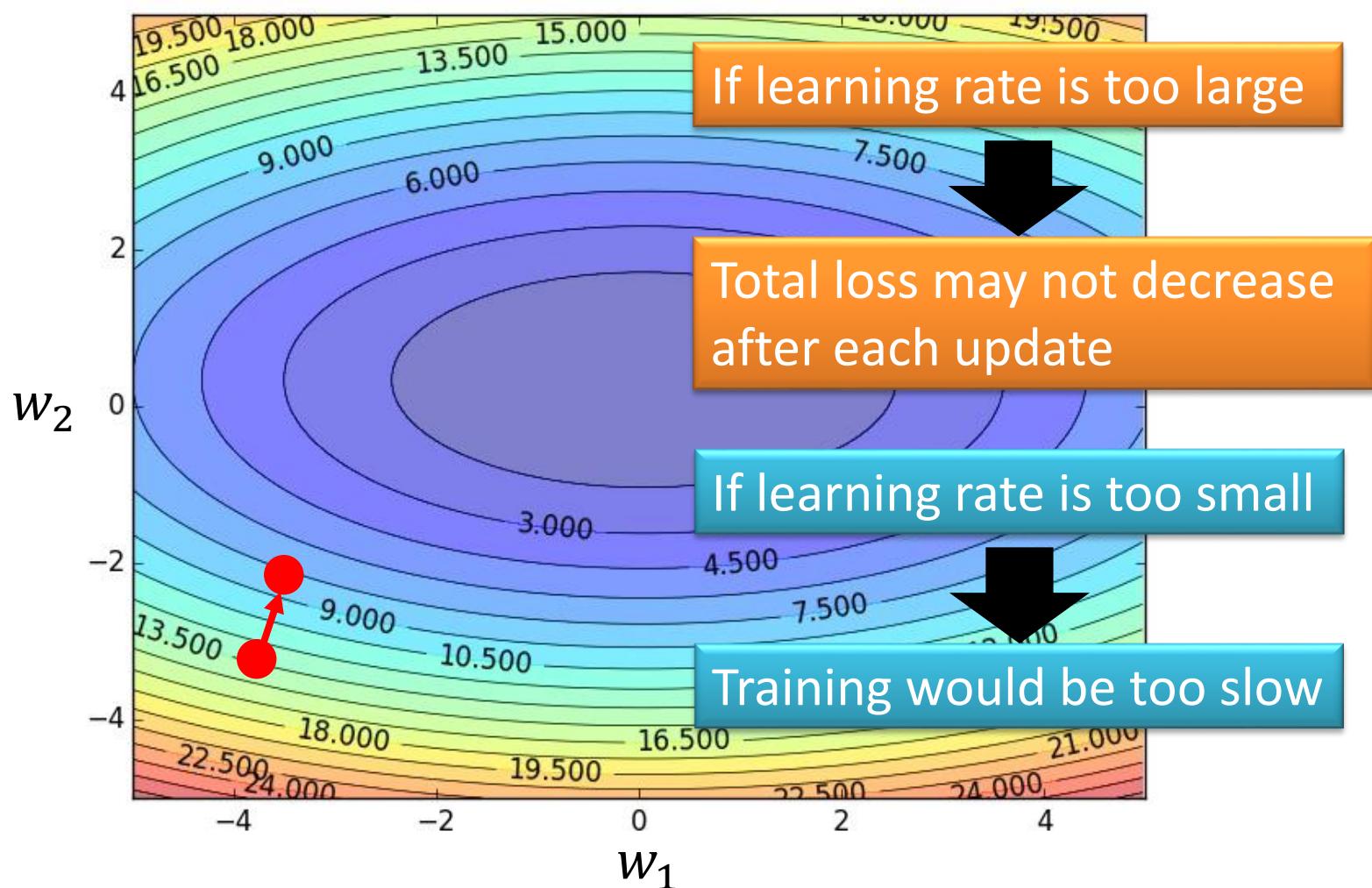
Learning Rates

Set the learning rate η carefully



Learning Rates

Set the learning rate η carefully



Learning Rates 选择合适的学习率

- 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 自适应学习率

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

Adagrad: $w \leftarrow w - \boxed{\eta_w} \partial L / \partial w$

Parameter dependent learning rate

近似二阶动量

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

η → constant

$\sqrt{\sum_{i=0}^t (g^i)^2}$ → 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	$\mathbf{g^0}$
	0.1

Learning rate:

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

$$= \frac{\eta}{0.1}$$

$$\frac{\eta}{\sqrt{0.1^2 + 0.2^2}} = \frac{\eta}{\sqrt{0.01 + 0.04}} = \frac{\eta}{\sqrt{0.05}} = \frac{\eta}{\sqrt{0.05}} = \frac{\eta}{0.22}$$



w_2	$\mathbf{g^0}$
	20.0

Learning rate:

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

$$= \frac{\eta}{20}$$

$$\frac{\eta}{\sqrt{20^2 + 10^2}} = \frac{\eta}{\sqrt{400 + 100}} = \frac{\eta}{\sqrt{500}} = \frac{\eta}{\sqrt{500}} = \frac{\eta}{22}$$

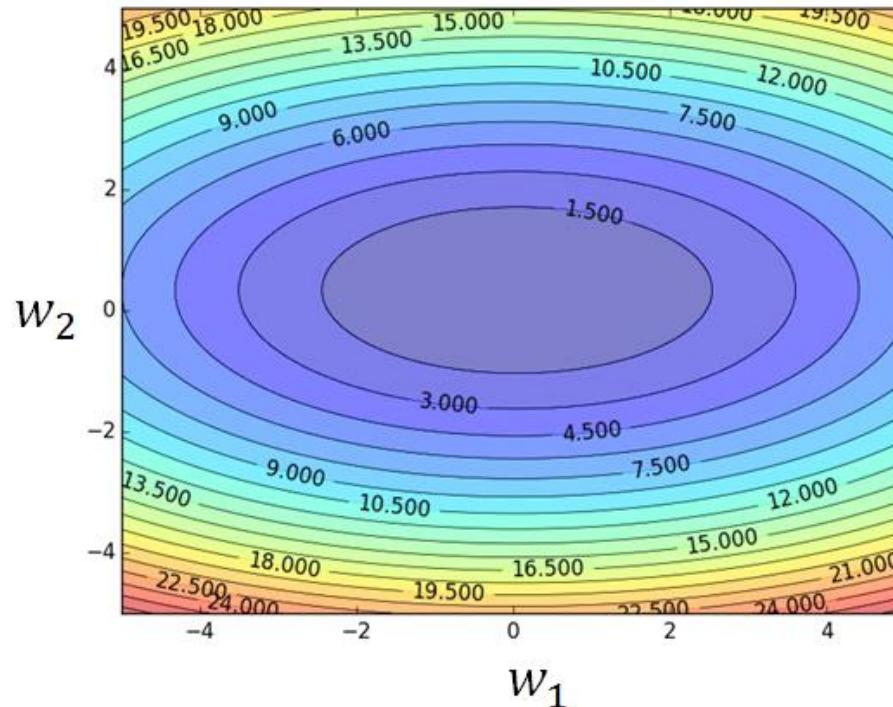


- Observation:**
1. Learning rate is smaller and smaller for all parameters
 2. Smaller derivatives, larger learning rate, and vice versa

Why?

Larger derivatives

Smaller Learning Rate



Smaller Derivatives

Larger Learning Rate

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

Why?

Not the whole story

- **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 for underfitting problem of Deep Learning



YES

Choosing proper loss

Mini-batch & Batch Norm

New activation function

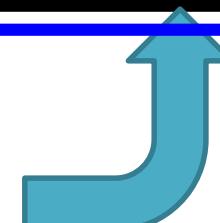
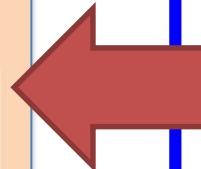
Adaptive Learning Rate

Momentum

Good Results on
Testing Data?

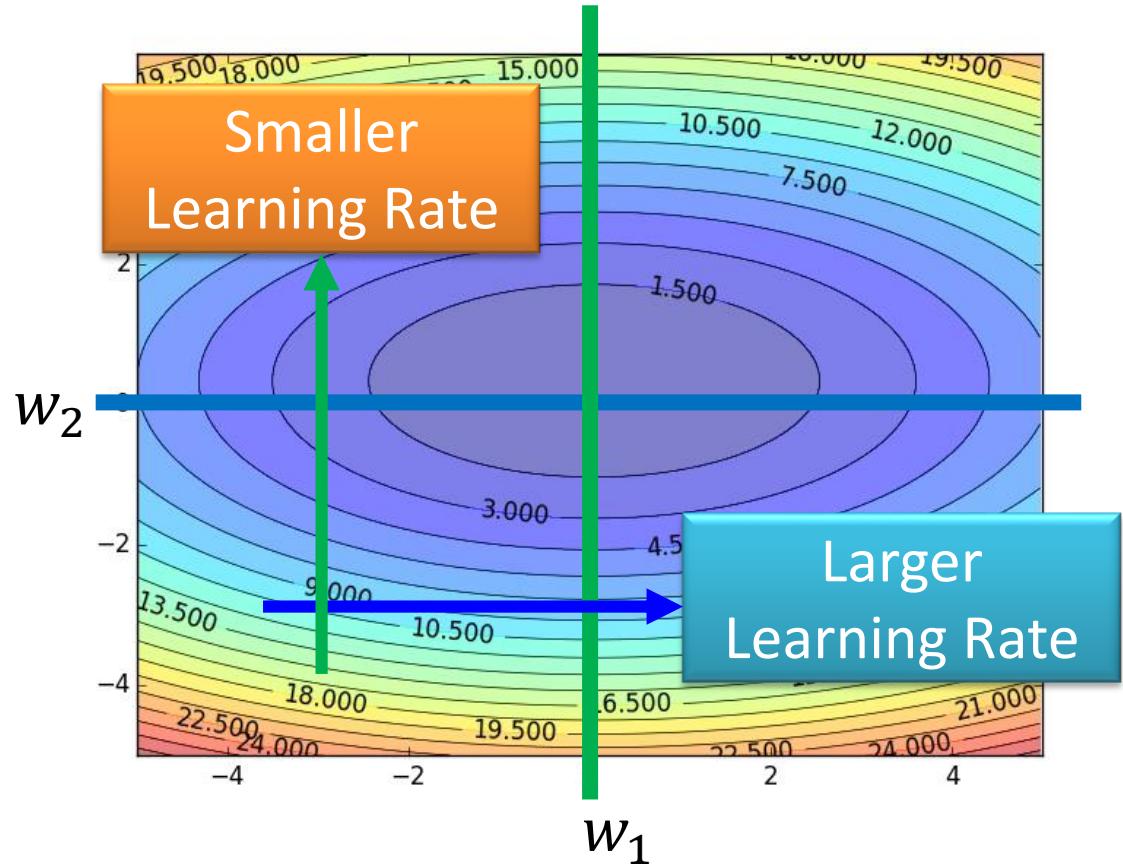
YES

Good Results on
Training Data?



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

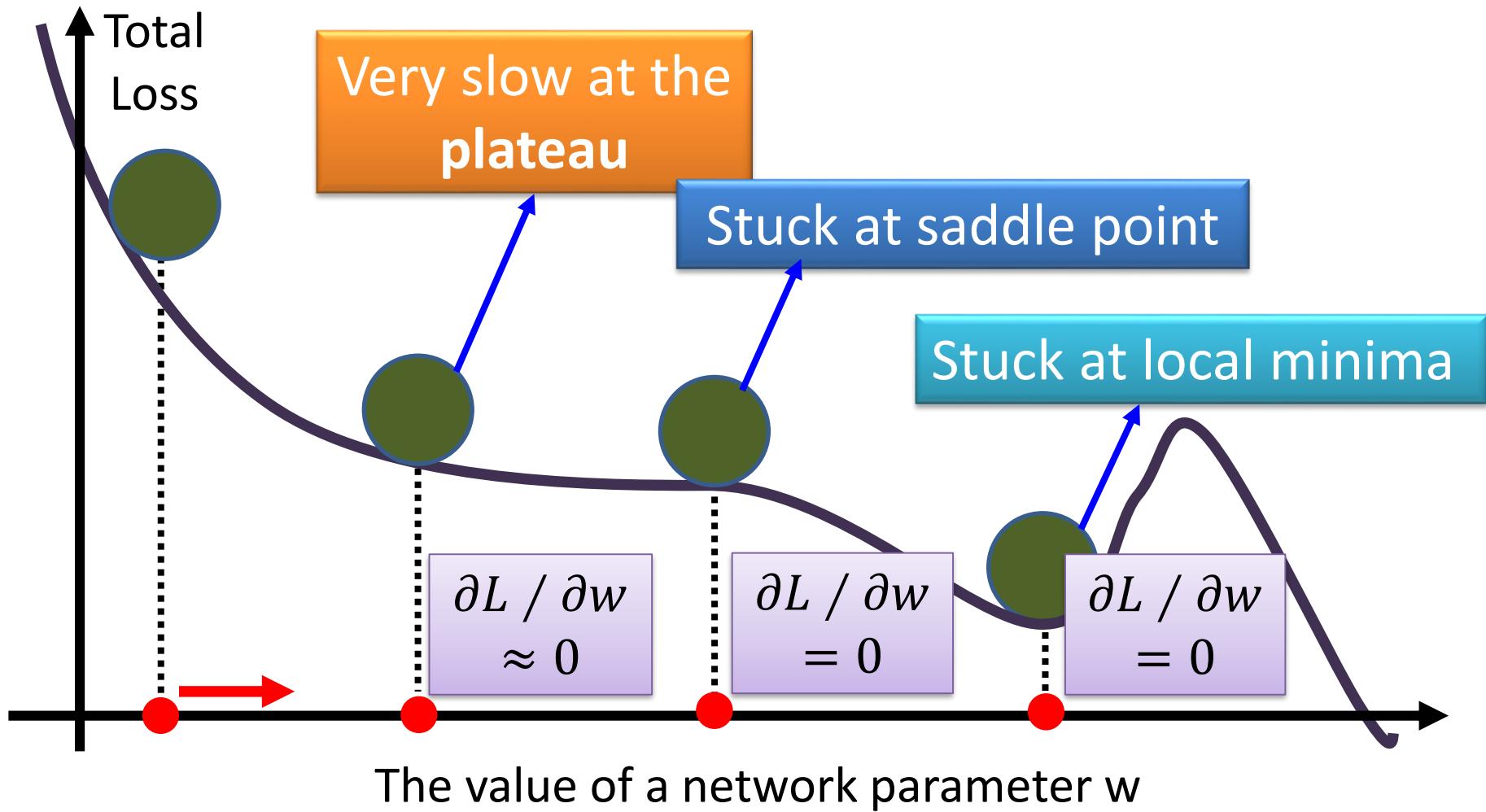
Adagrad



$$w^{t+1} \leftarrow w^t - \frac{\eta}{\sqrt{\sum_{i=0}^t (g^i)^2}} g^t$$

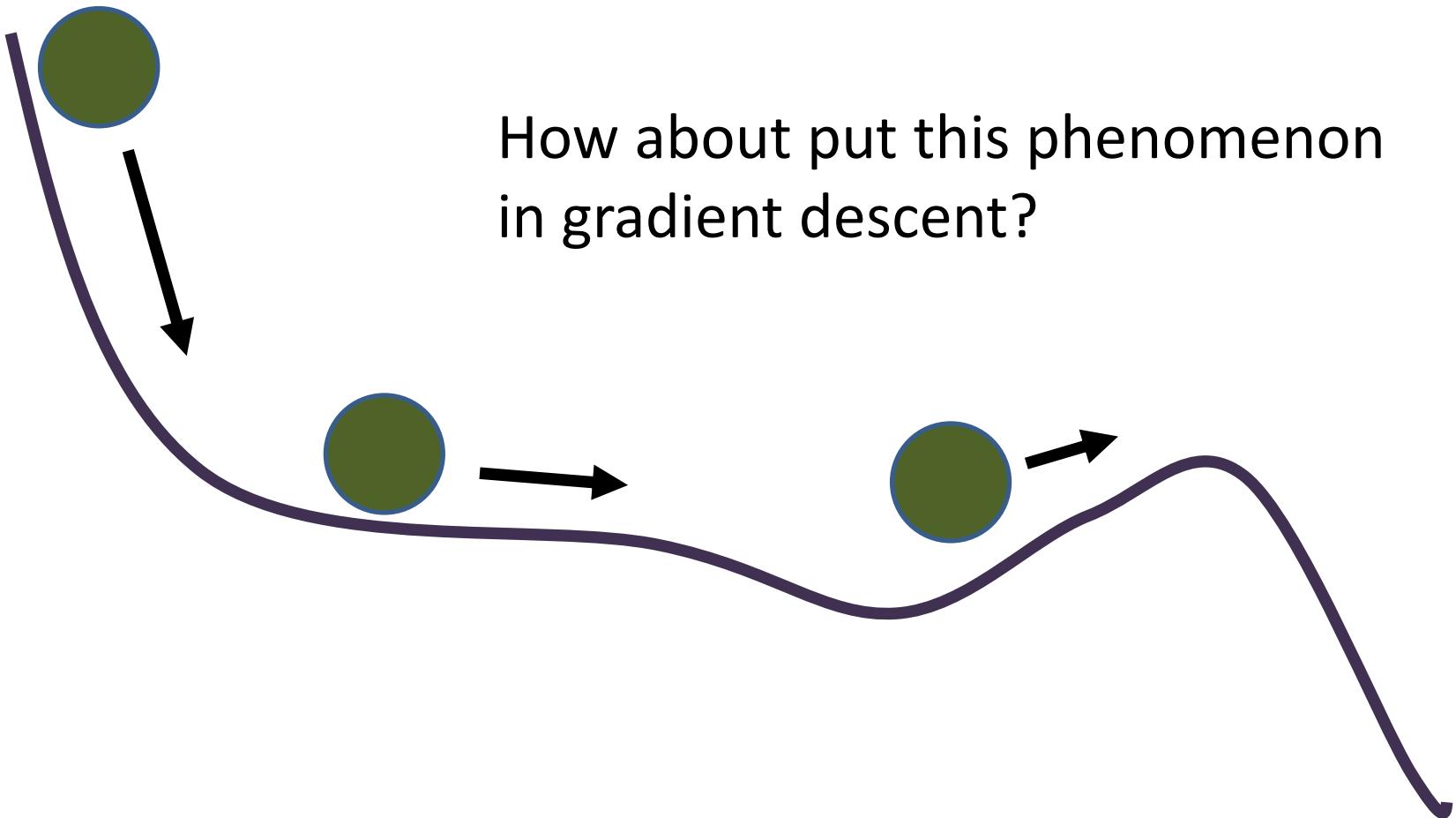
Use first derivative to estimate second derivative

Hard to find optimal network parameters



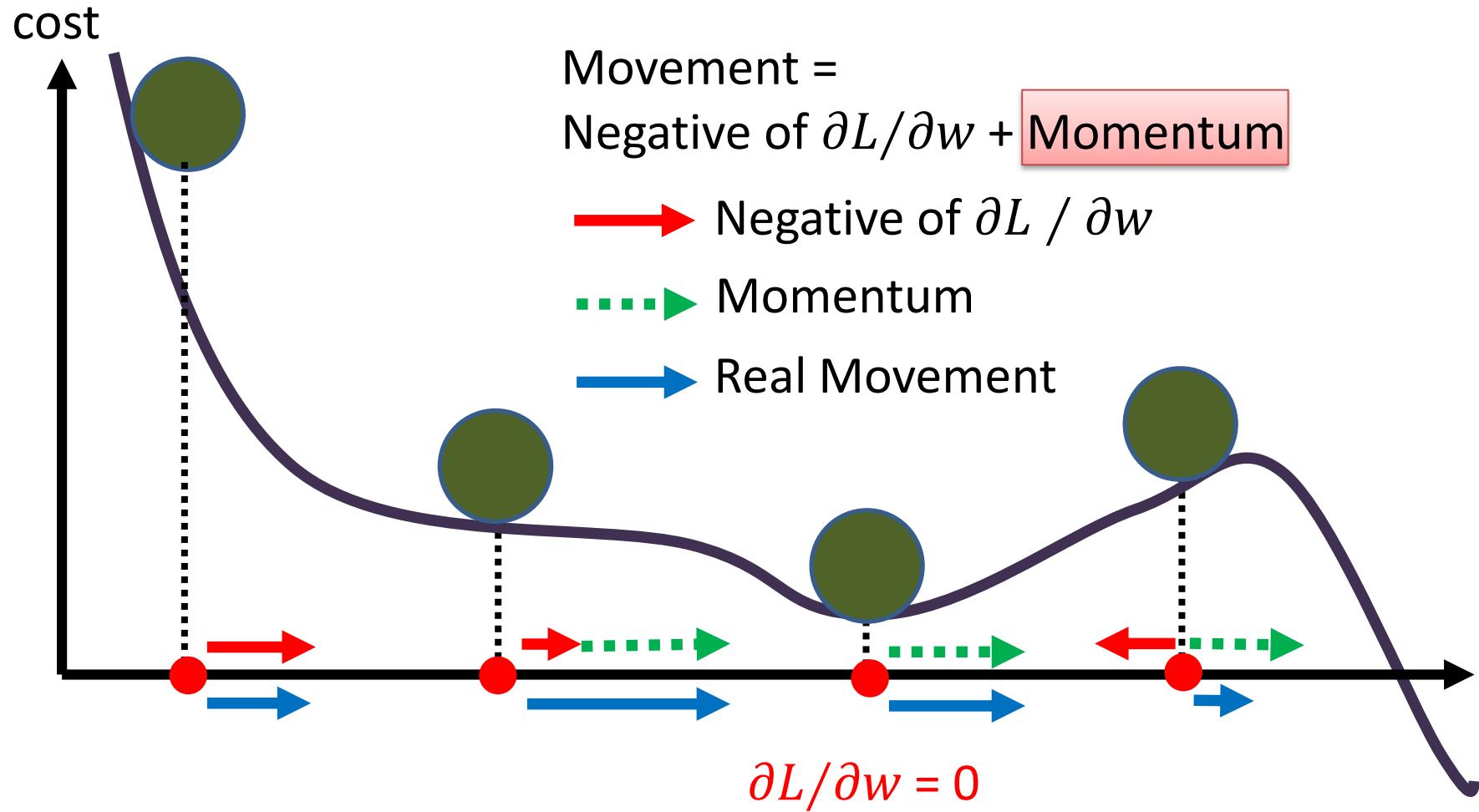
In physical world

- Momentum



Momentum

Still not guarantee reaching global minima, but give some hope



RMSProp (Advanced Adagrad) + Momentum

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

```
model.compile(loss='categorical_crossentropy',
               optimizer=Adam(),
               metrics=['accuracy'])
```

Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t .

Require: α : Stepsize
Require: $\beta_1, \beta_2 \in [0, 1]$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0 : Initial parameter vector
 $m_0 \leftarrow 0$ (Initialize 1st moment vector)
 $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
 $t \leftarrow 0$ (Initialize timestep)
while θ_t not converged **do**
 $t \leftarrow t + 1$
 $g_t \leftarrow \nabla_{\theta} f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
 $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
 $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
 $\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
 $\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
 $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$ (Update parameters)
end while
return θ_t (Resulting parameters)

Gradient Descent Method

$$\Delta\theta (= \theta^{t+1} - \theta^t) g^{t+1} = \frac{\partial L(\theta^{t+1})}{\partial \theta}$$

SGD, BGD, MBGD

Adagrad

$$-\eta g^{t+1}$$

$$-\frac{\eta}{\sqrt{\sum_{i=0}^t (g^{i+1})^2}} g^{t+1}$$

引入二阶动量：
加速度

RMSProp

$$-\frac{\eta}{\sigma^{t+1}} g^{t+1}$$

$$(\sigma^{0+1} = g^{0+1}, \\ \sigma^{t+1} =$$

$$\sqrt{\alpha(\sigma^{t-1})^2 + (1-\alpha)(g^{t+1})^2})$$

SGD with Momentum (SGDM)

$$v^{t+1} = \lambda v^t - \eta g^{t+1}$$

$$v^{0+1} = 0$$

引入一阶动量：
惯性速度

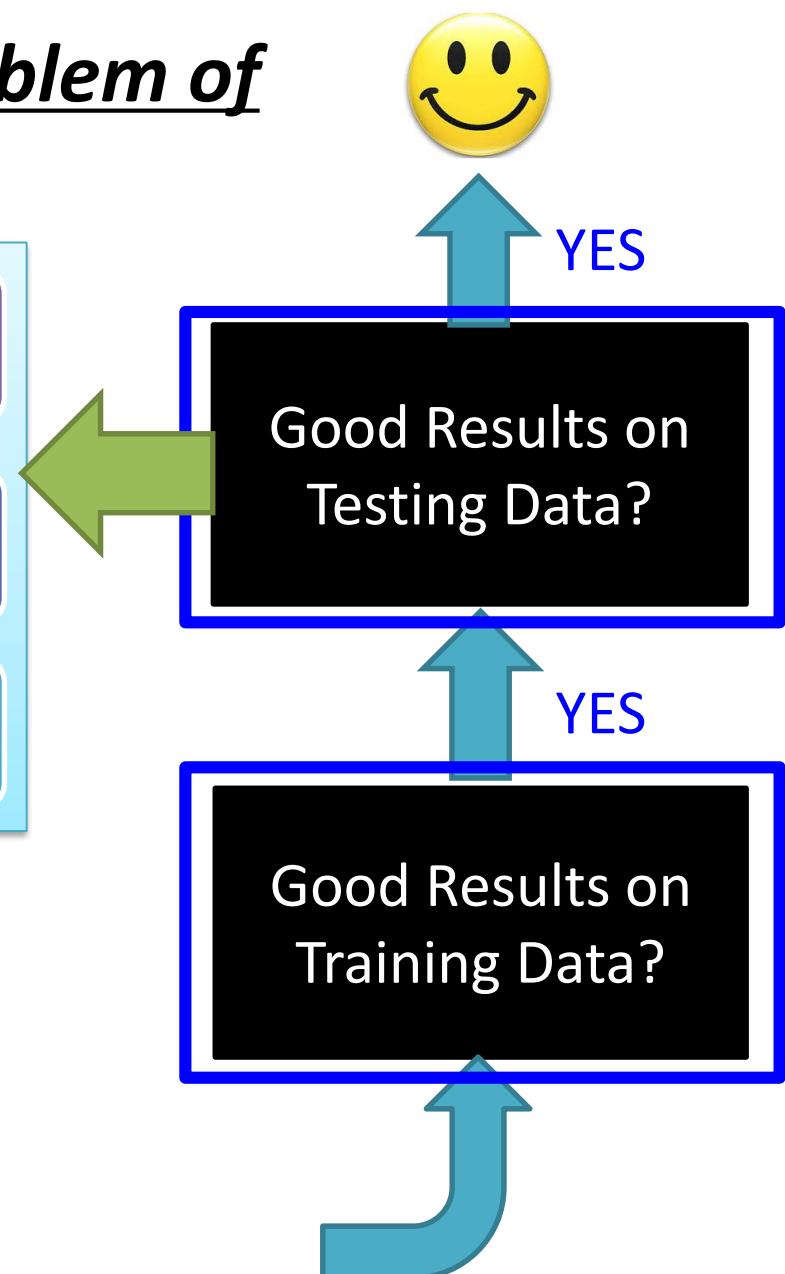
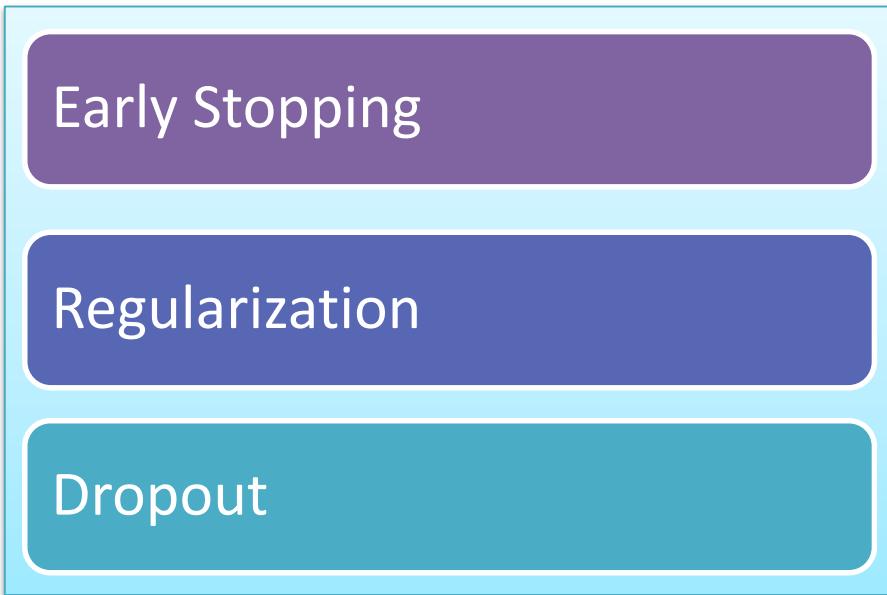
SGD with Nesterov (NAG)

$$v^{t+1} = \lambda v^t - \eta \frac{\partial L^n(\theta^t + v^t)}{\partial \theta}$$

Adam(RMSProp + Momentum)

Nadam (Nesterov+Adam)

Recipe for overfitting problem of Deep Learning



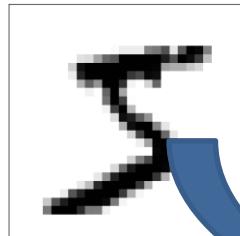
解决过拟合

Panacea(万能药) for Overfitting

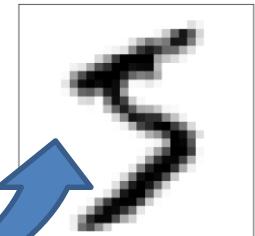
- Have more training data
- *Create* more training data (?)

Handwriting recognition:

Original
Training Data:

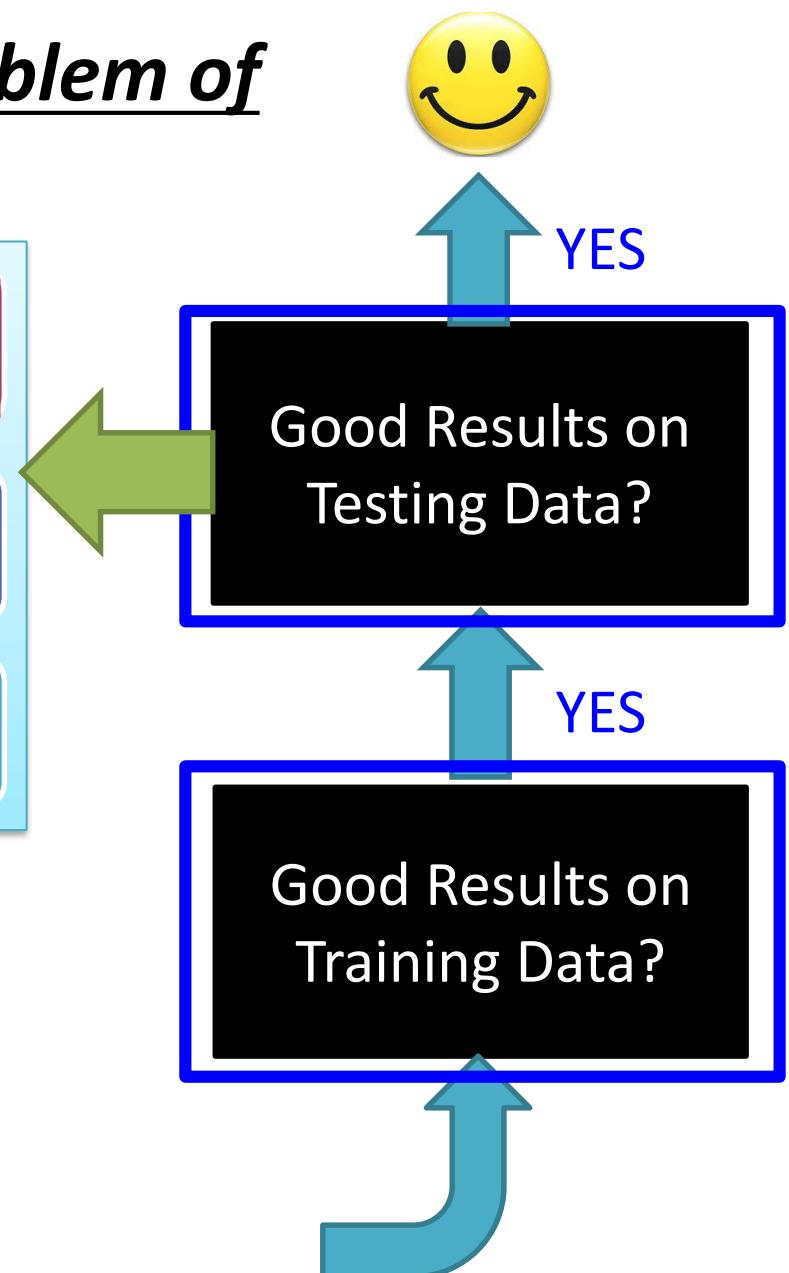


Created
Training Data:

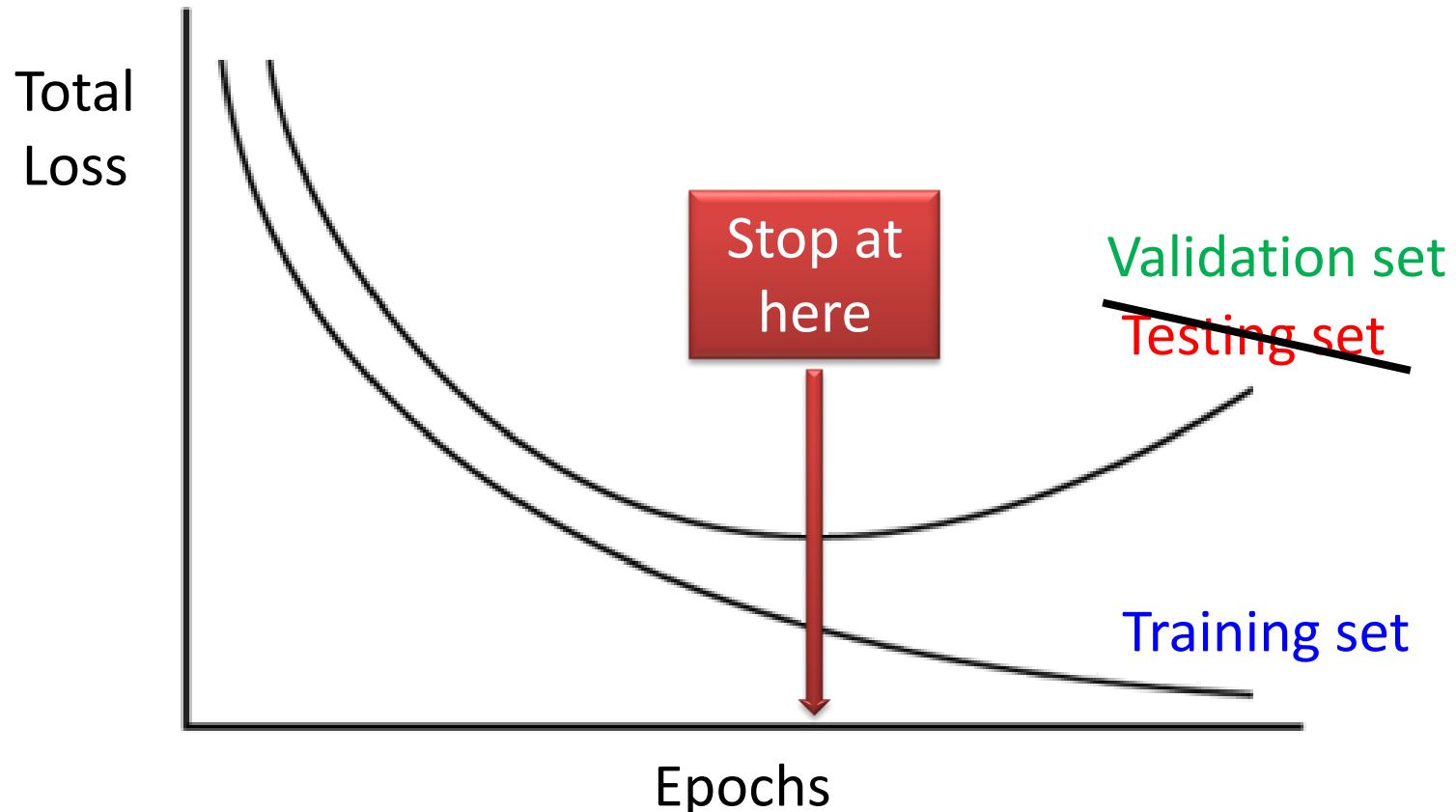


Shift 15°

Recipe for overfitting problem of Deep Learning

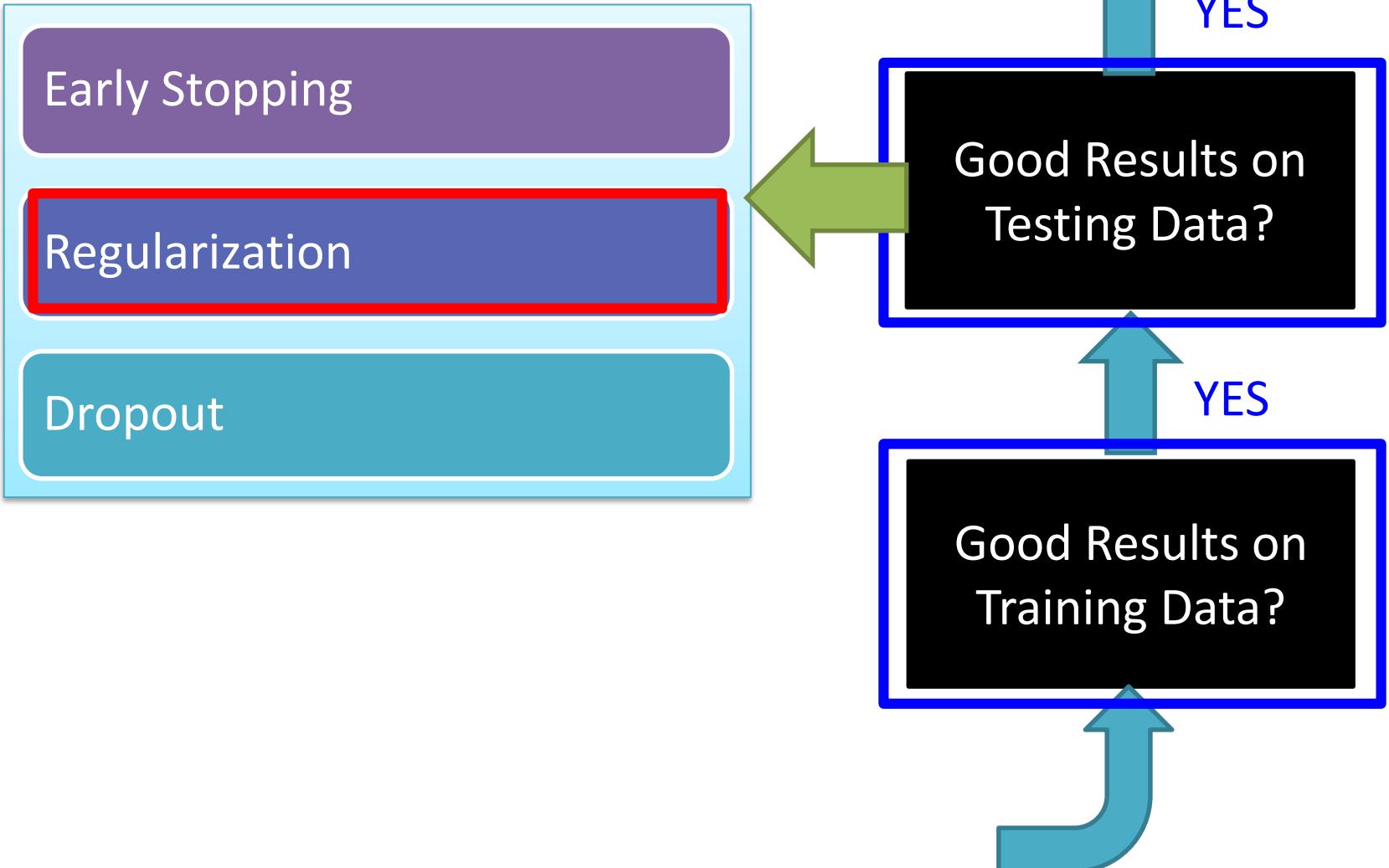


Early Stopping



Keras: <http://keras.io/getting-started/faq/#how-can-i-interrupt-training-when-the-validation-loss-isnt-decreasing-anymore>

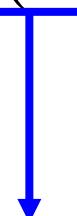
Recipe for overfitting problem of Deep Learning



Regularization

- New loss function to be minimized
 - Find a set of weight not only minimizing original cost but also close to zero

$$L'(\theta) = \underline{L(\theta)} + \lambda \frac{1}{2} \underline{\|\theta\|_2} \rightarrow \text{Regularization term}$$



Original loss
(e.g. minimize square error, cross entropy ...)

$$\theta = \{w_1, w_2, \dots\}$$

L2 regularization:

$$\|\theta\|_2 = (w_1)^2 + (w_2)^2 + \dots$$

(usually not consider biases)

Regularization

L1 regularization:

$$\|\theta\|_1 = |w_1| + |w_2| + \dots$$

- New loss function to be minimized

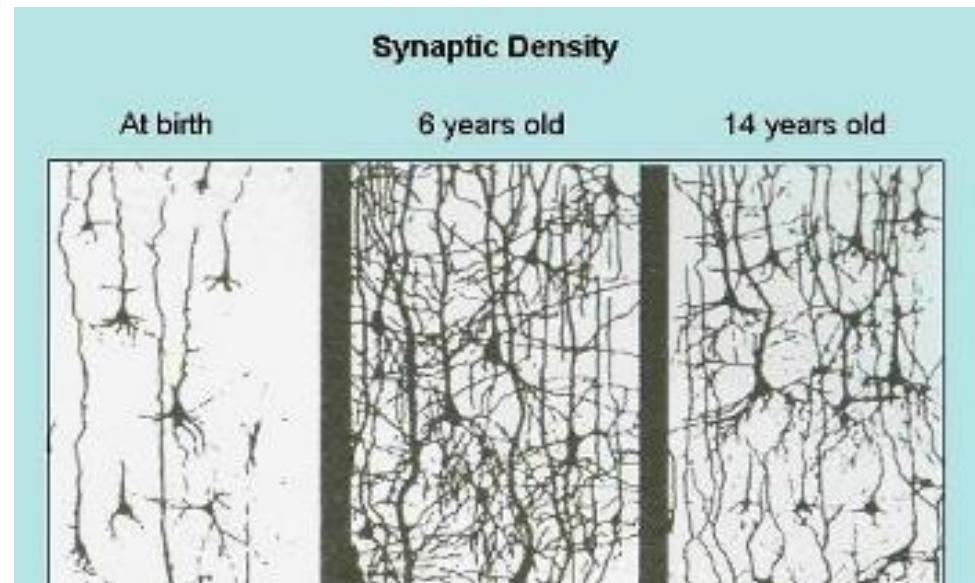
$$L'(\theta) = L(\theta) + \lambda \frac{1}{2} \|\theta\|_1 \quad \frac{\partial L'}{\partial w} = \frac{\partial L}{\partial w} + \lambda \operatorname{sgn}(w)$$

Update:

$$\begin{aligned} w^{t+1} &\rightarrow w^t - \eta \frac{\partial L'}{\partial w} = w^t - \eta \left(\frac{\partial L}{\partial w} + \lambda \operatorname{sgn}(w^t) \right) \\ &= w^t - \eta \frac{\partial L}{\partial w} - \underline{\eta \lambda \operatorname{sgn}(w^t)} \quad \text{Always delete} \\ &= (1 - \eta \lambda) w^t - \eta \frac{\partial L}{\partial w} \quad \dots \text{L2} \end{aligned}$$

Regularization - Weight Decay

- Our brain prunes out the useless link between neurons.

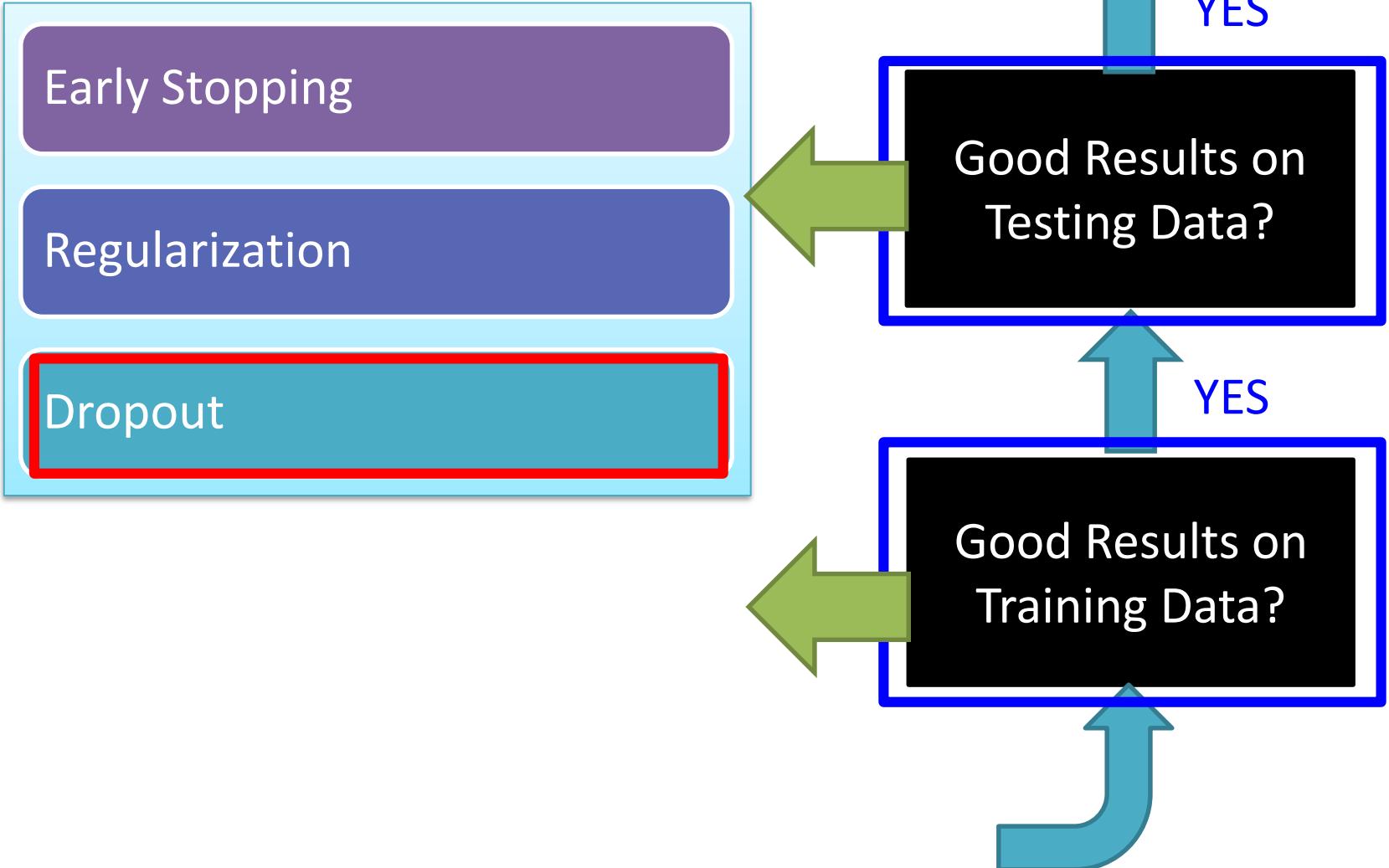


Doing the same thing to machine's brain improves the performance.



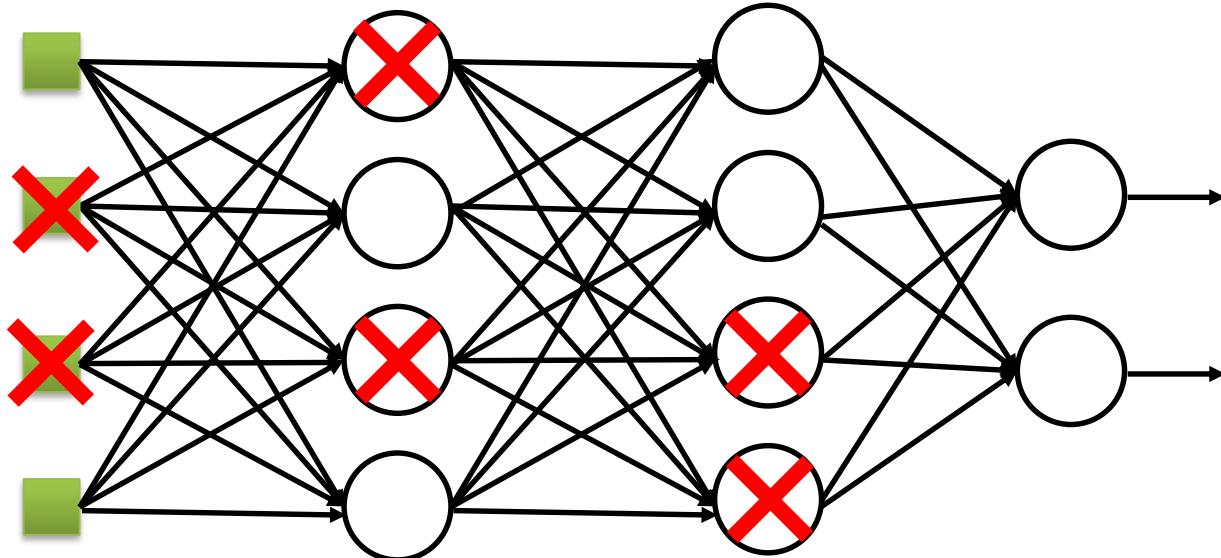
Source: Rethinking the Brain, Families and Work Institute, Rita Shore, 1997; Founders Network slide

Recipe for overfitting problem of Deep Learning



Dropout(随机失活)

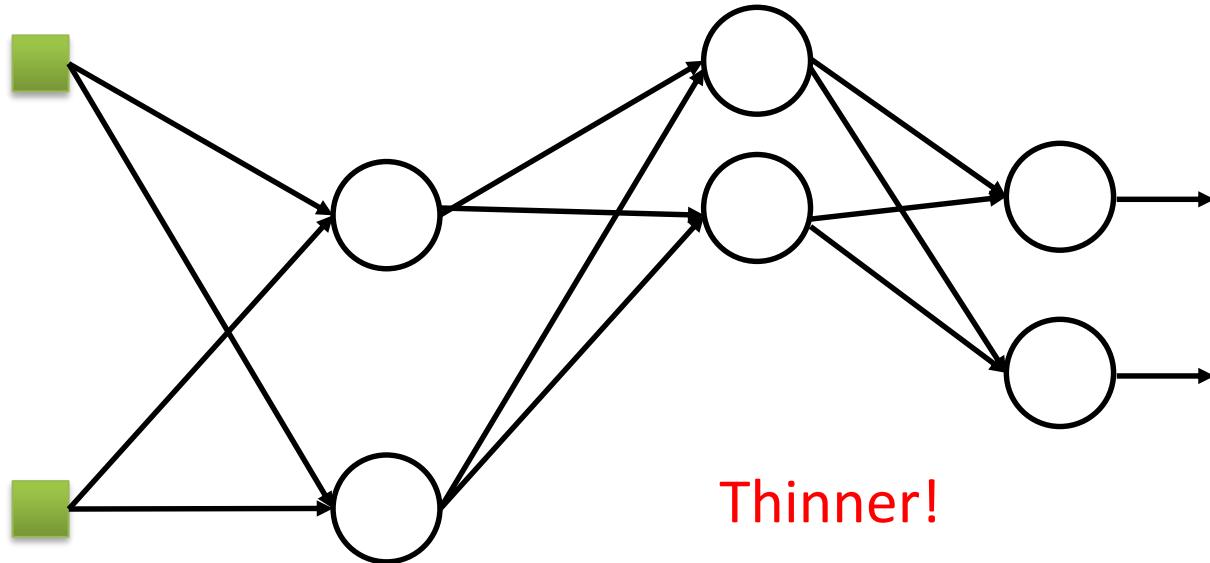
Training:



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

Dropout

Training:

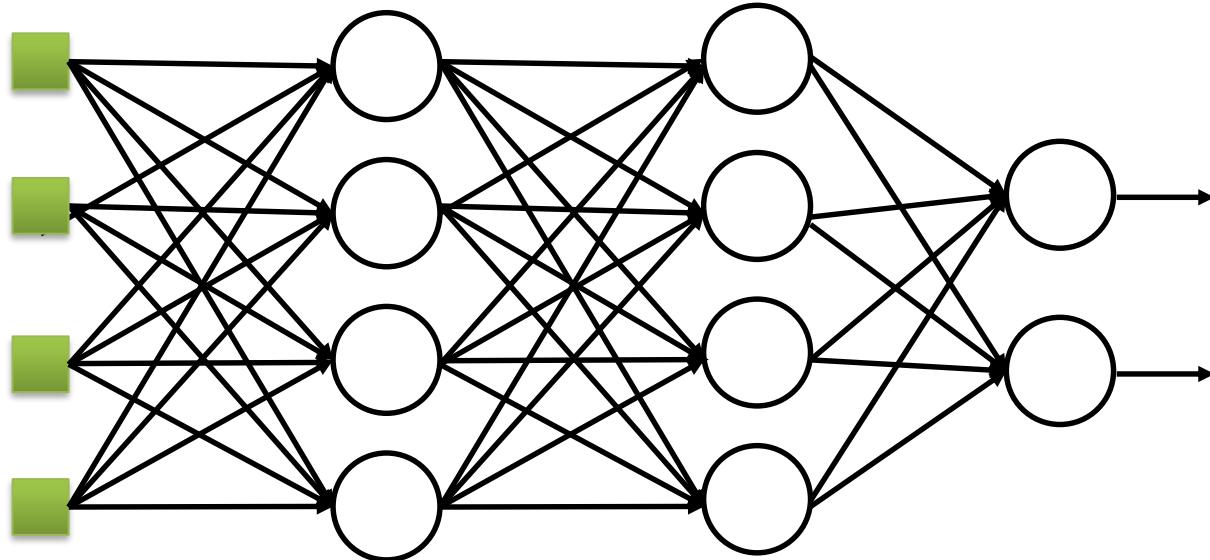


- **Each time before updating the parameters**
 - Each neuron has $p\%$ to dropout
 - ➡ **The structure of the network is changed.**
 - Using the new network for training

For each mini-batch, we resample the dropout neurons

Dropout

Testing:

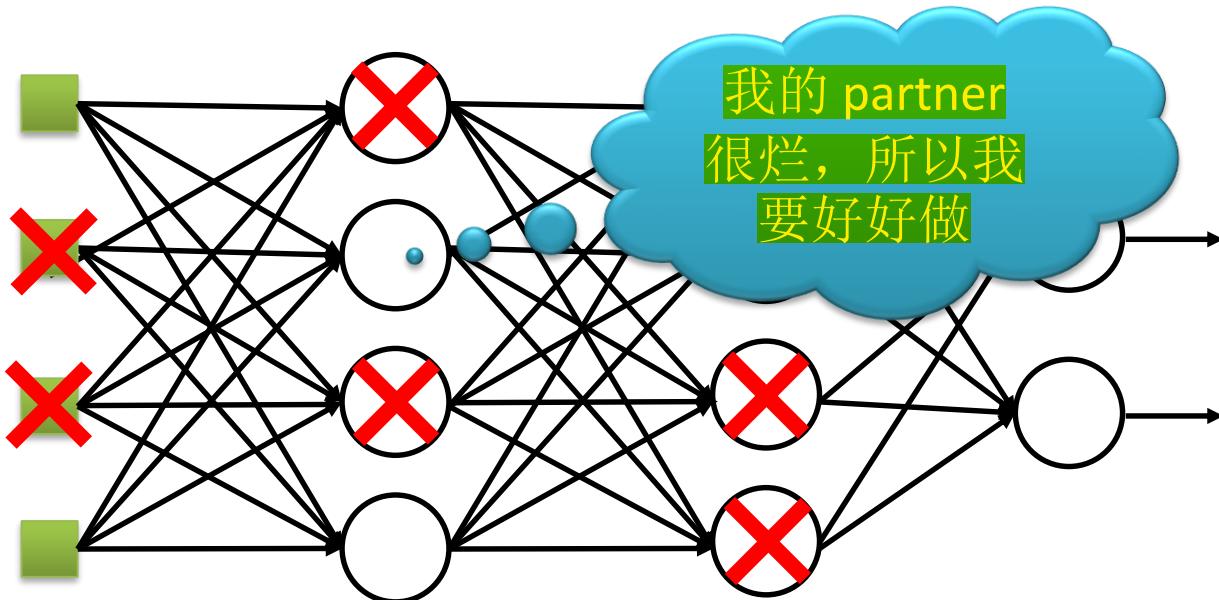


➤ No dropout

- If the dropout rate at training is $p\%$,
all the weights times $1-p\%$
- Assume that the dropout rate is 50%.
If a weight $w = 1$ by training, set $w = 0.5$ for testing.

[mnist_nn_dropout.py](#)

Dropout - Intuitive Reason



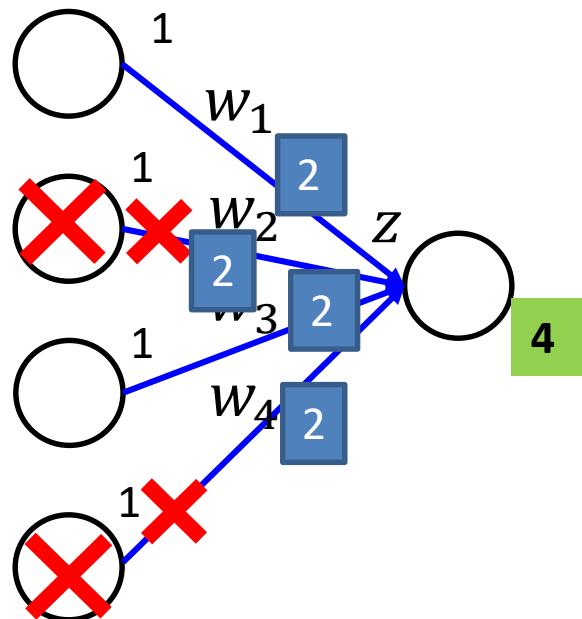
- When teams up, if everyone expect the partner will do the work, nothing will be done finally.
- However, if you know your partner will dropout, you will do better.
- When testing, no one dropout actually, so obtaining good results eventually.

Dropout - Intuitive Reason

- Why the weights should multiply $(1-p)\%$ (dropout rate) when testing?

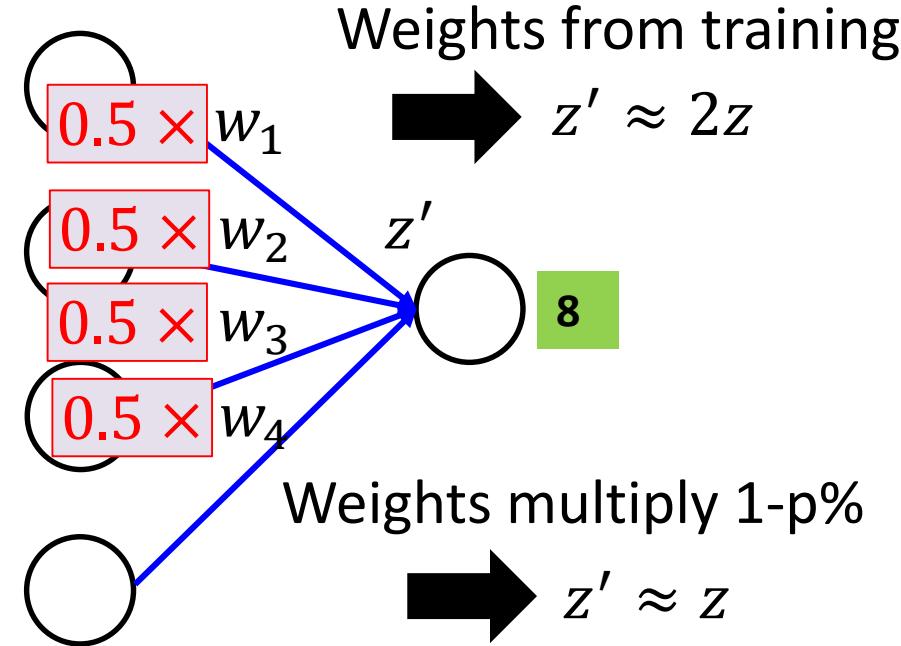
Training of Dropout

Assume dropout rate is 50%



Testing of Dropout

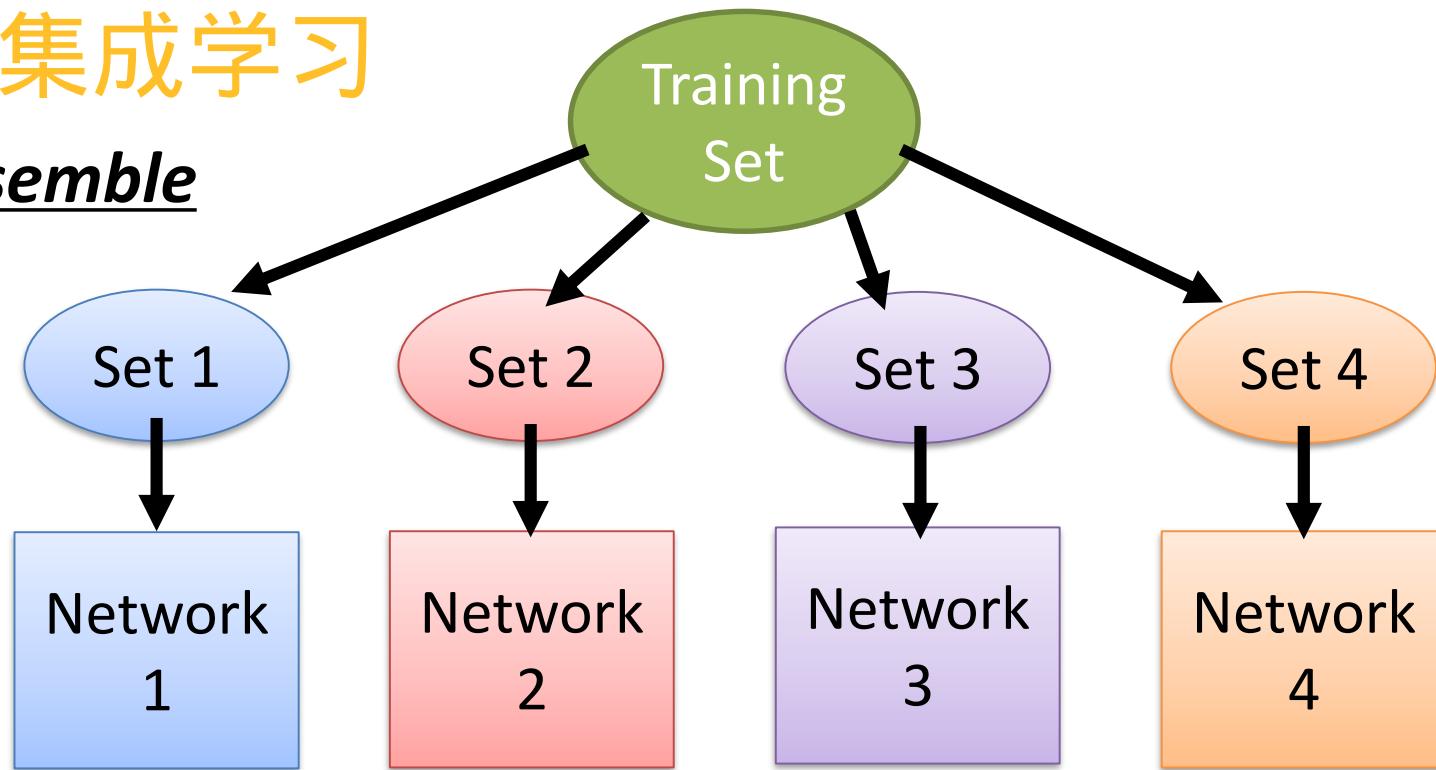
No dropout



Dropout is a kind of ensemble.

集成学习

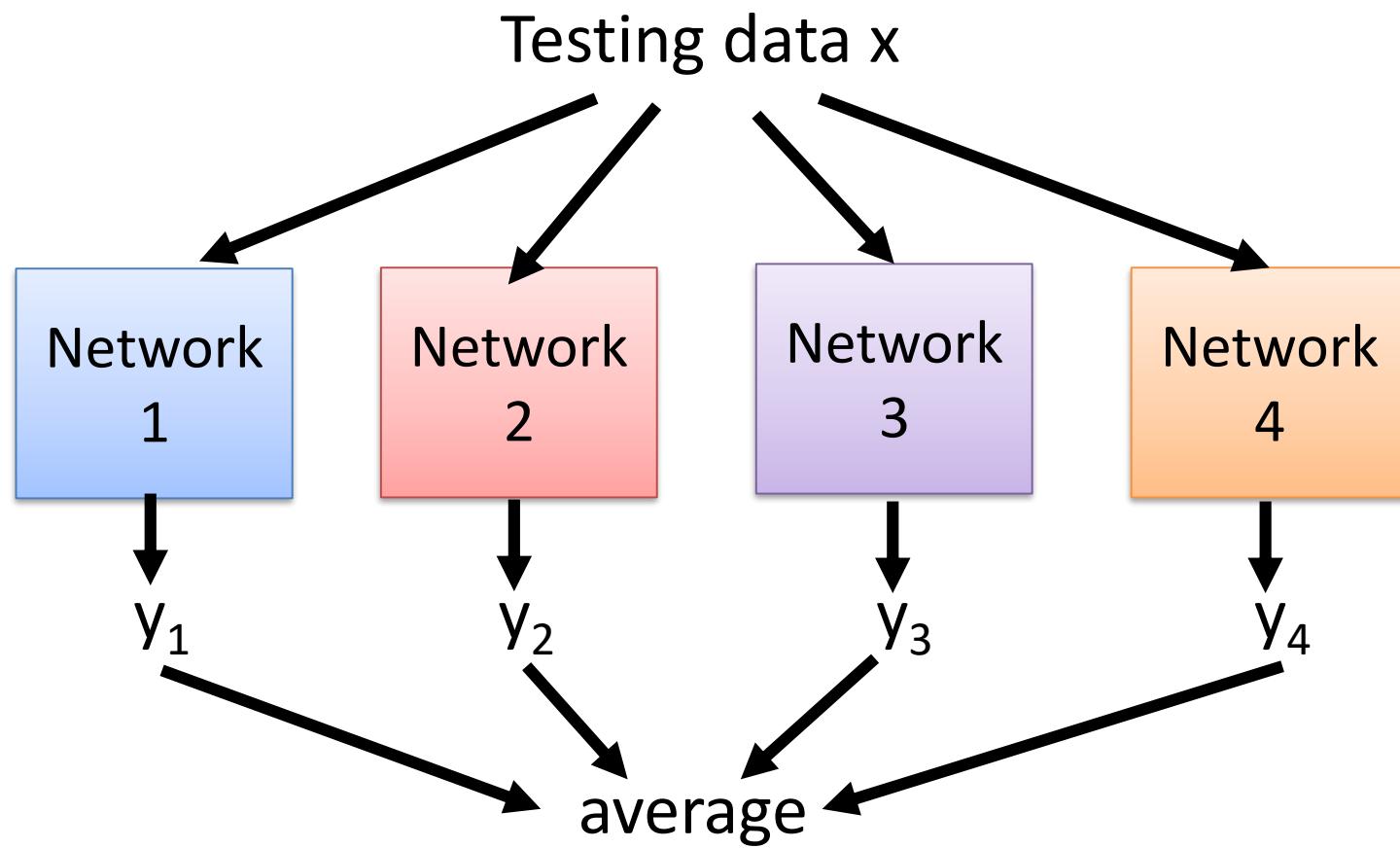
Ensemble



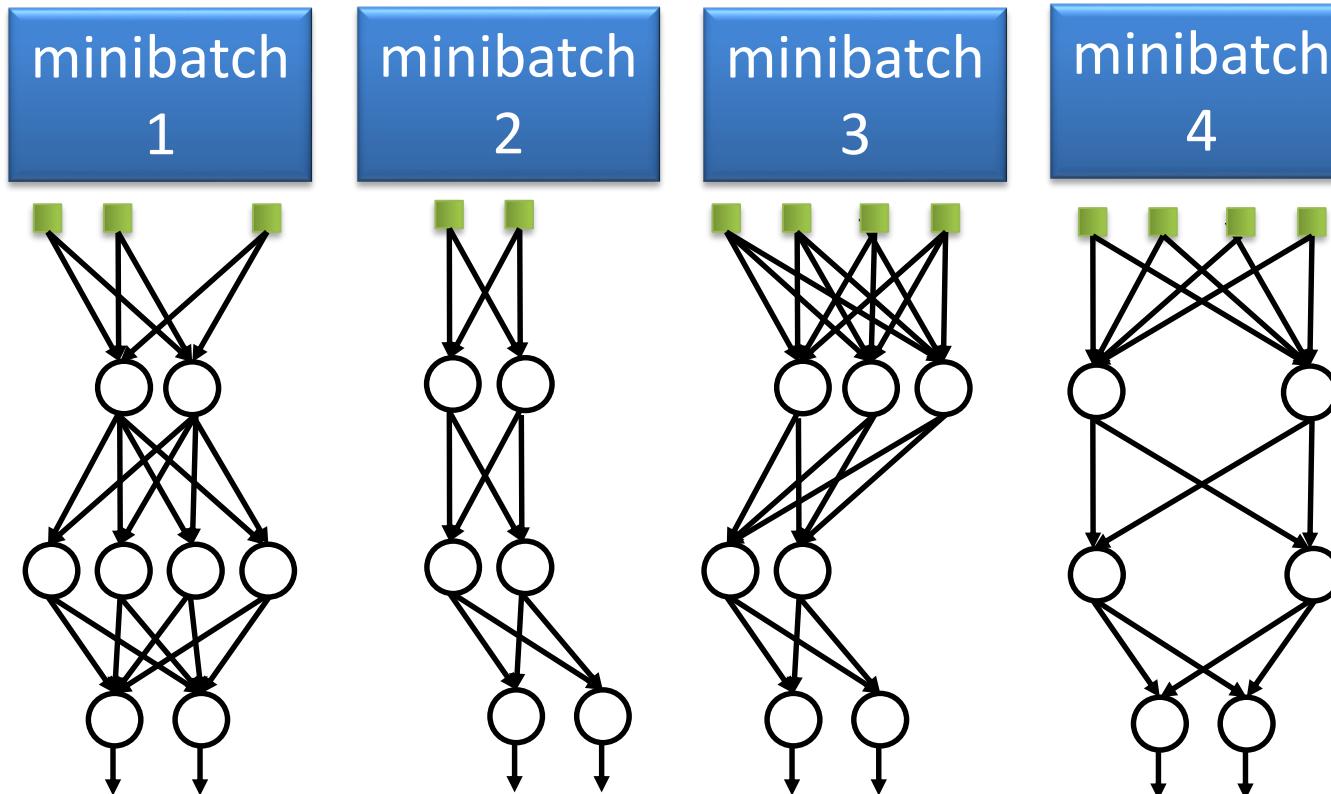
Train a bunch of networks with different structures

Dropout is a kind of ensemble.

Ensemble



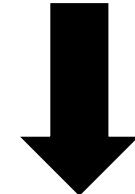
Dropout is a kind of ensemble.



Training of
Dropout

M neurons

⋮

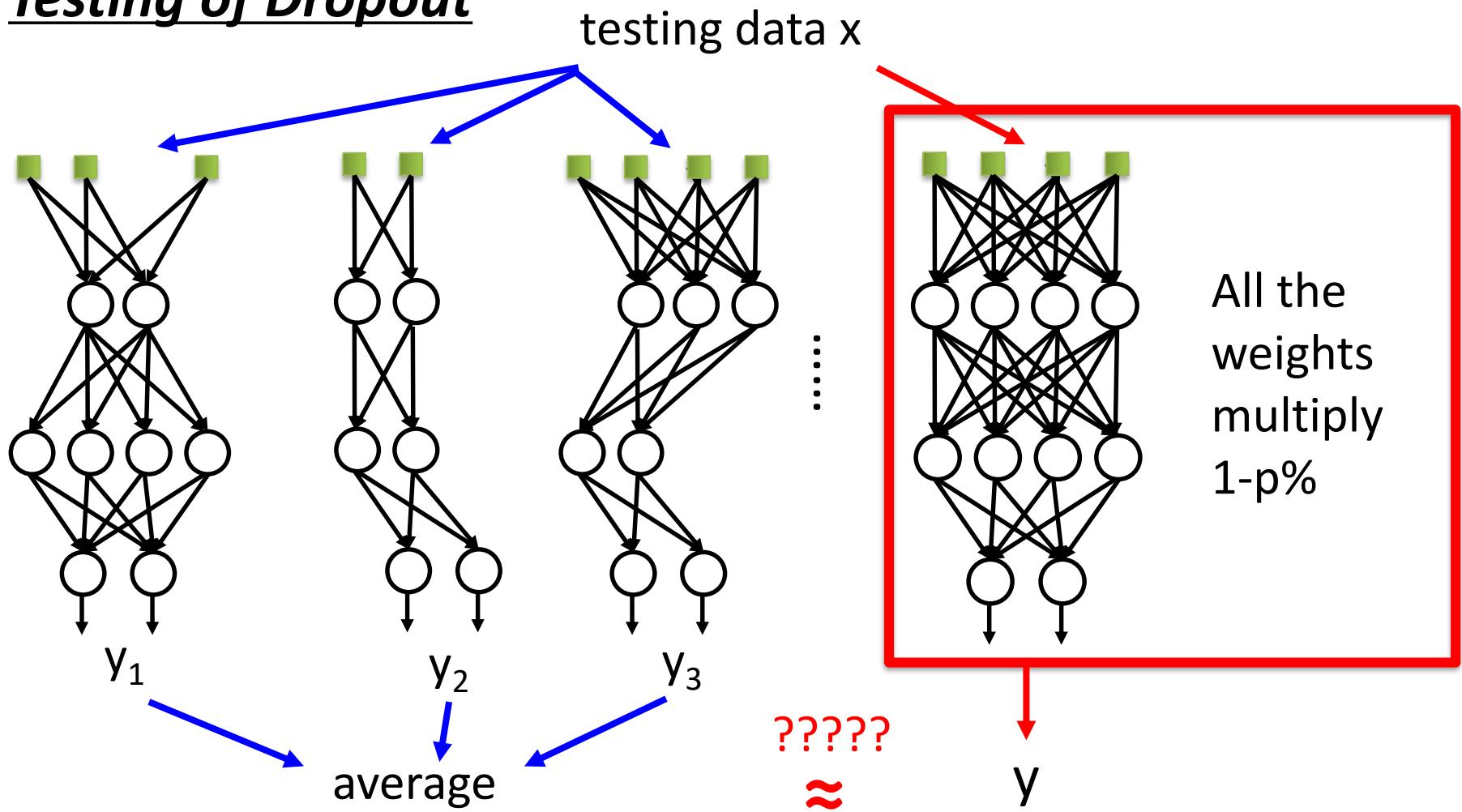


2^M possible
networks

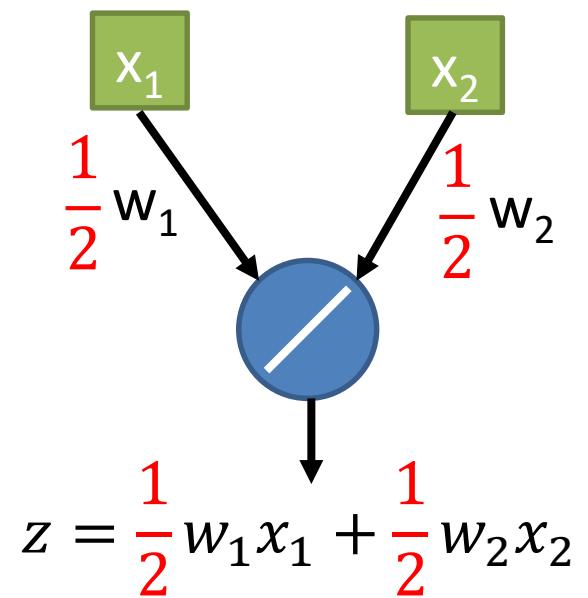
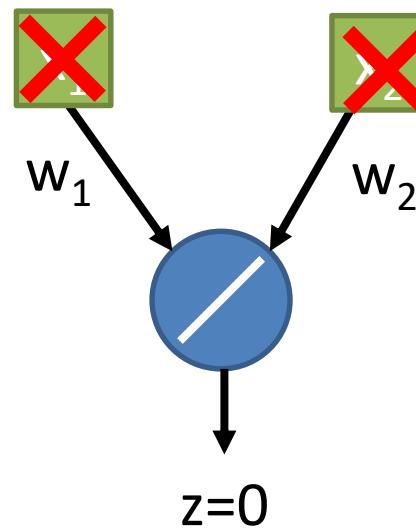
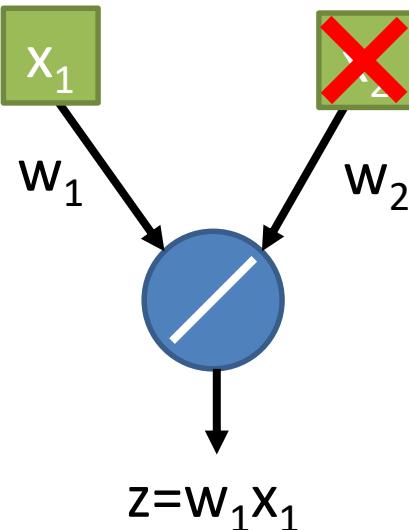
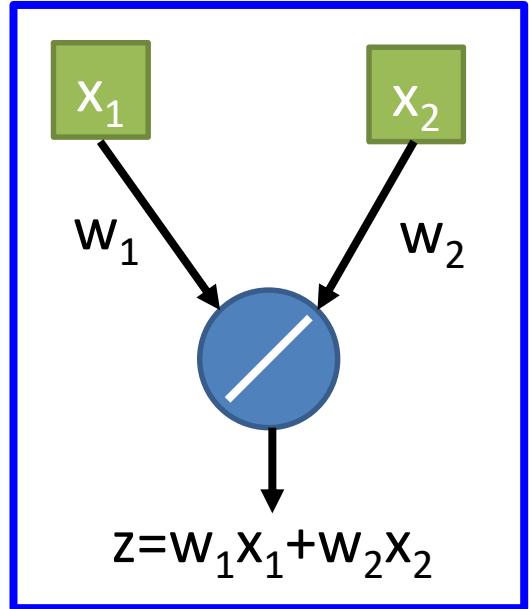
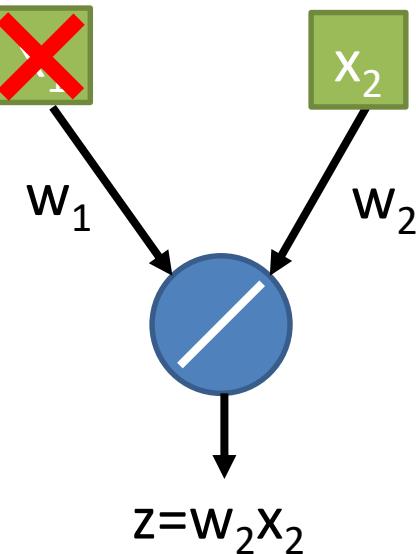
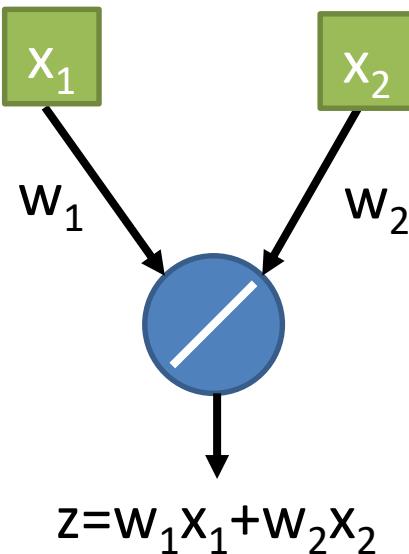
- Using one mini-batch to train one network
- Some parameters in the network are shared

Dropout is a kind of ensemble.

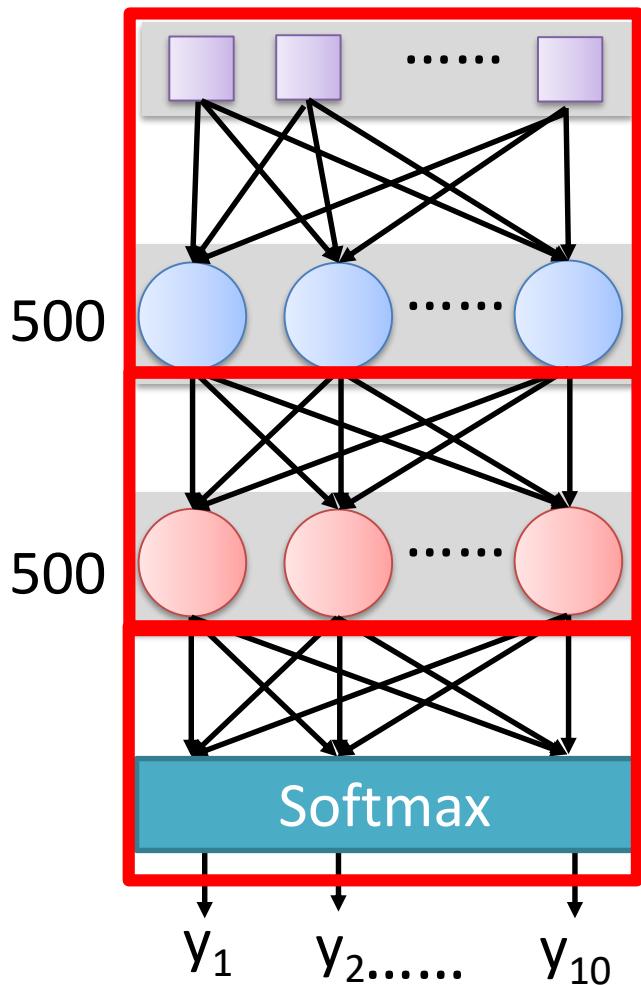
Testing of Dropout



Testing of Dropout



Dropout in Keras



```
model = Sequential()
```

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

model.add(dropout(0.8))

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

model.add(dropout(0.8))

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

Recipe for underfitting problem of Deep Learning



YES

Choosing proper loss

Mini-batch & Batch Norm

New activation function

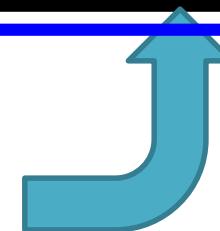
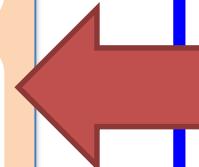
Adaptive Learning Rate

Momentum

Good Results on
Testing Data?

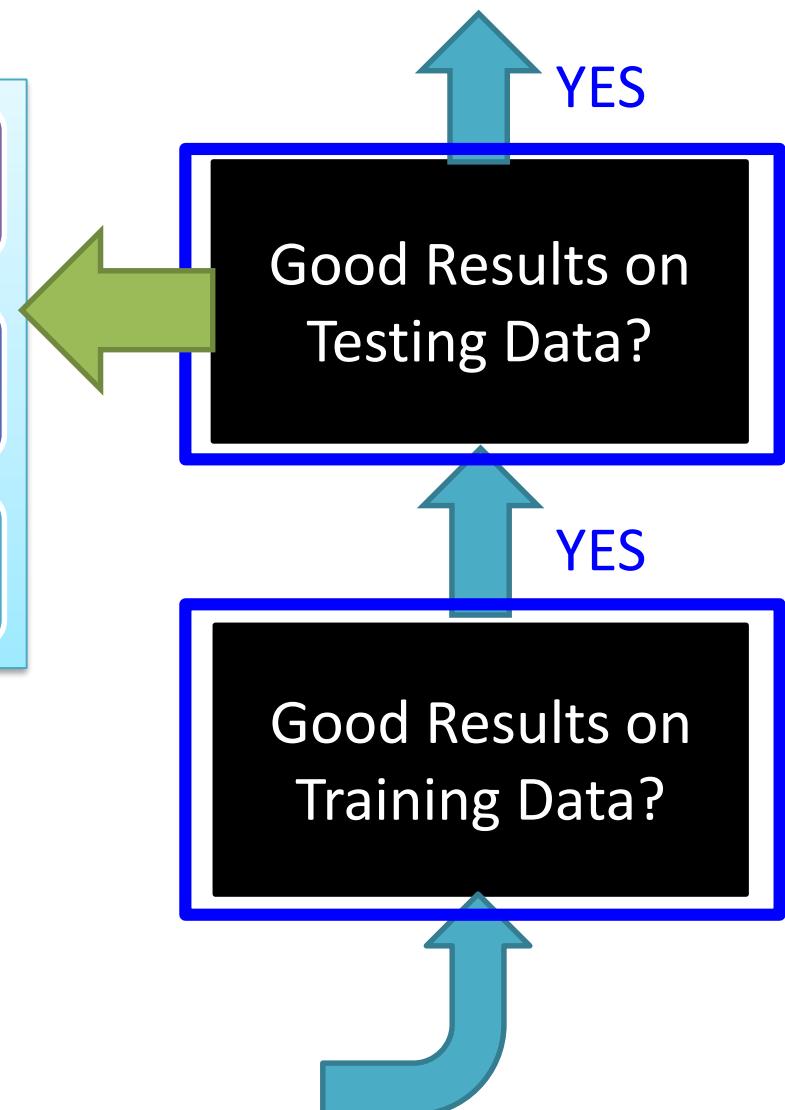
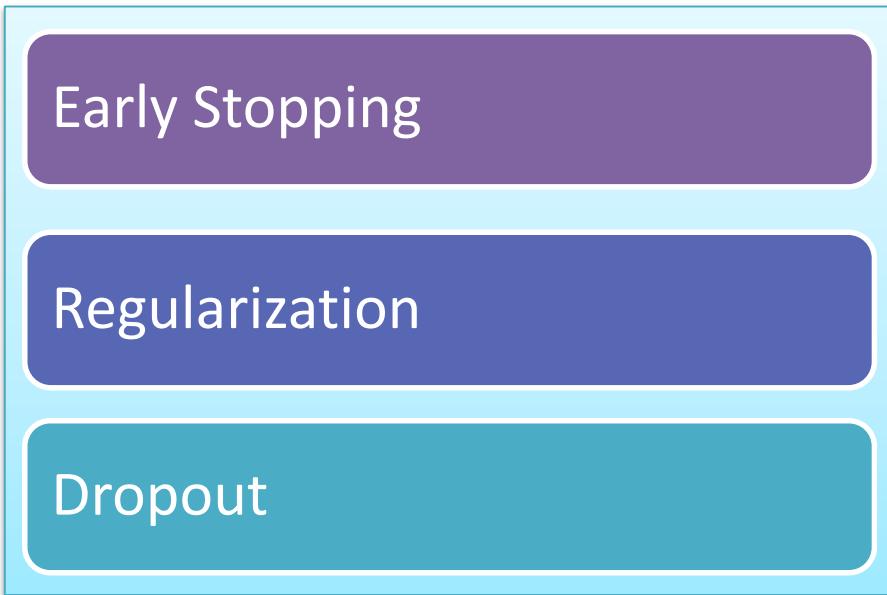
YES

Good Results on
Training Data?



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

Recipe for overfitting problem of Deep Learning



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

謝 謝 !