南京大学
NANJING UNIVERSITY

人工智能导论

神经网络与深度学习
(Neural Networks)

郭兰哲

南京大学 智能科学与技术学院

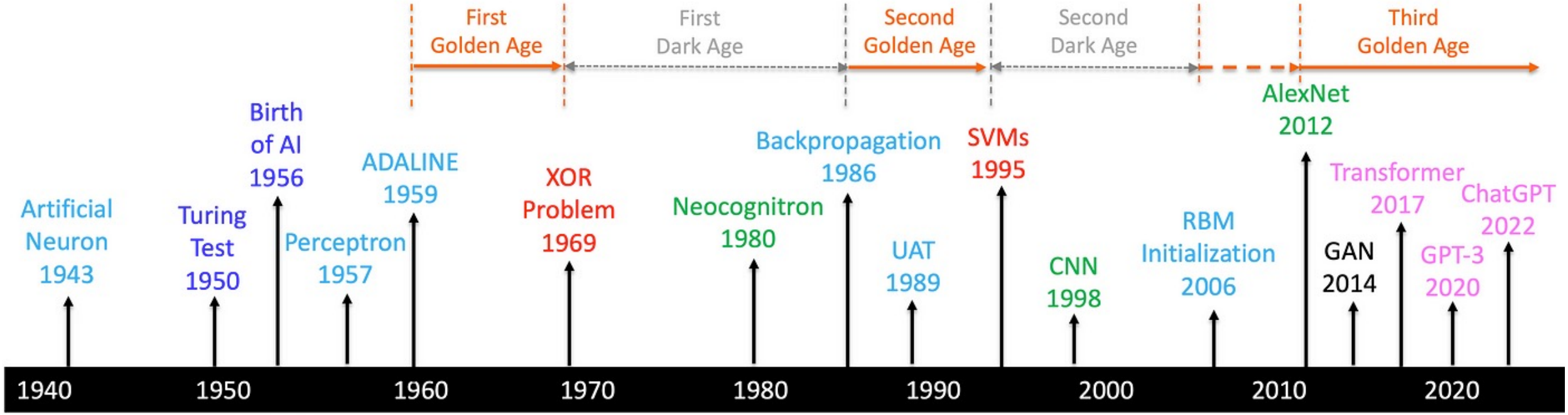
https://www.lamda.nju.edu.cn/guolz

Email: guolz@nju.edu.cn

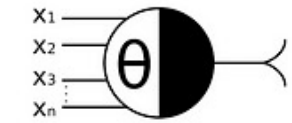
大纲

- 神经元模型到前馈神经网络
- 参数优化：BP算法
- 深度学习

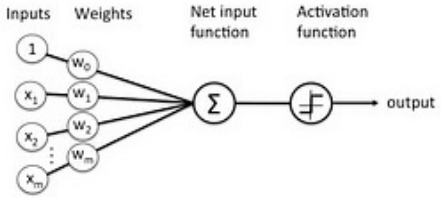
神经网络的“几起几落”



McCulloch-Pitts

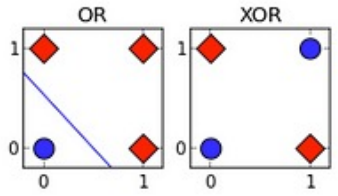


Rosenblatt

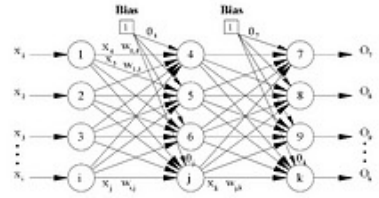


Widrow-Hoff

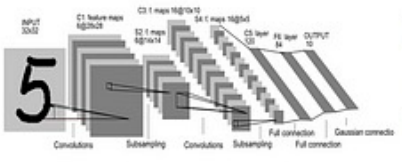
Minsky-Papert



Rumelhart, Hinton et al.

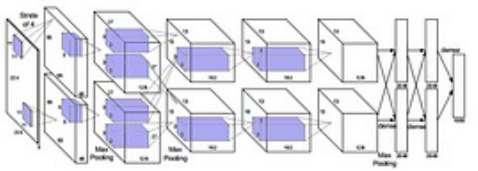


LeCun



Hinton-Ruslan

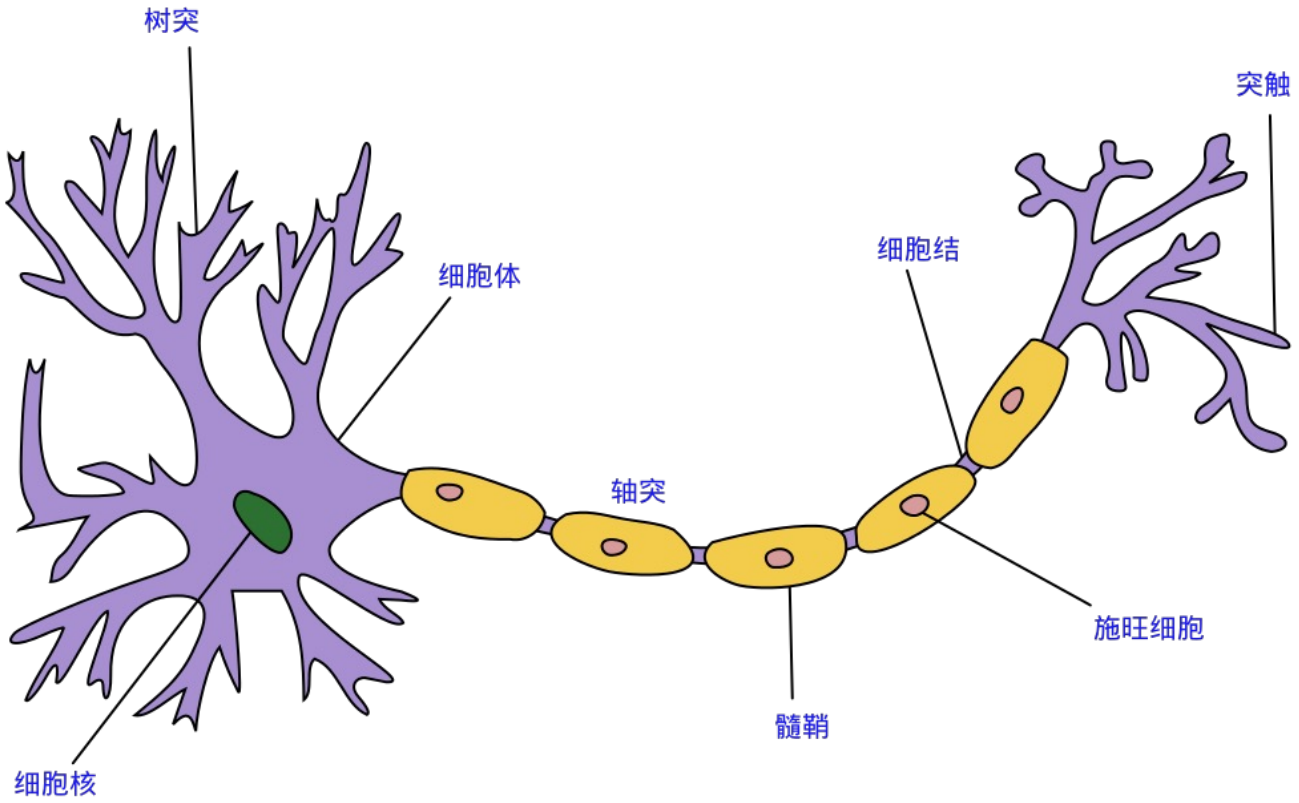
Krizhevsky et al.



Vaswani

生物神经元结构

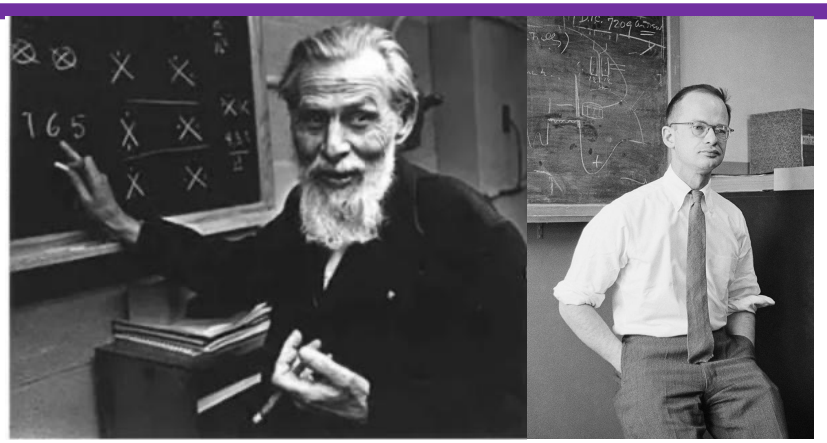
- 每个神经元与其他神经元相连，当它“兴奋”时，就会向相连的神经元发送化学物质，从而改变这些神经元内的电位
- 如果某神经元的电位超过一个“阈值”，那么它就会被激活，向其它神经元发送化学物质



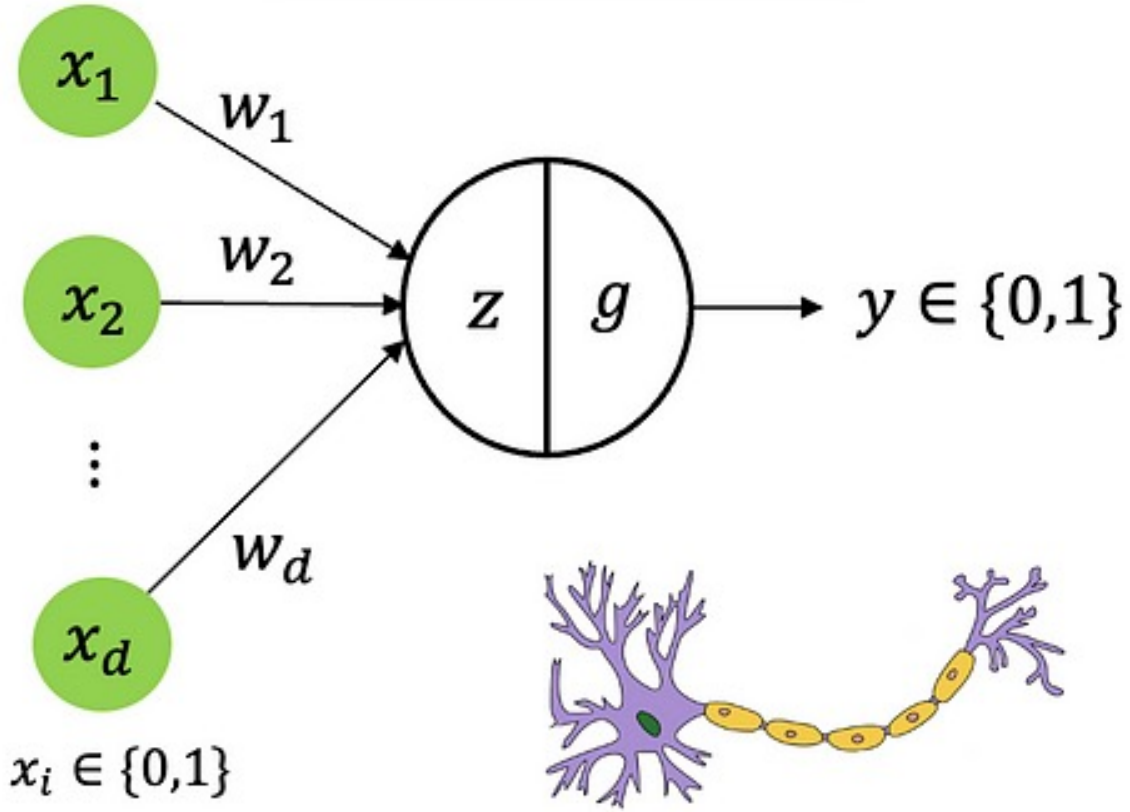
M-P神经元(1943年)

1943年，麦卡洛克(Warren mcCulloch)和皮茨(Walter Pitts）

提出了神经元的数学模型：M-P神经元模型



Warren mcCulloch　　Walter Pitts



$$x_i \in \{0,1\} \text{ and } y \in \{0,1\}$$

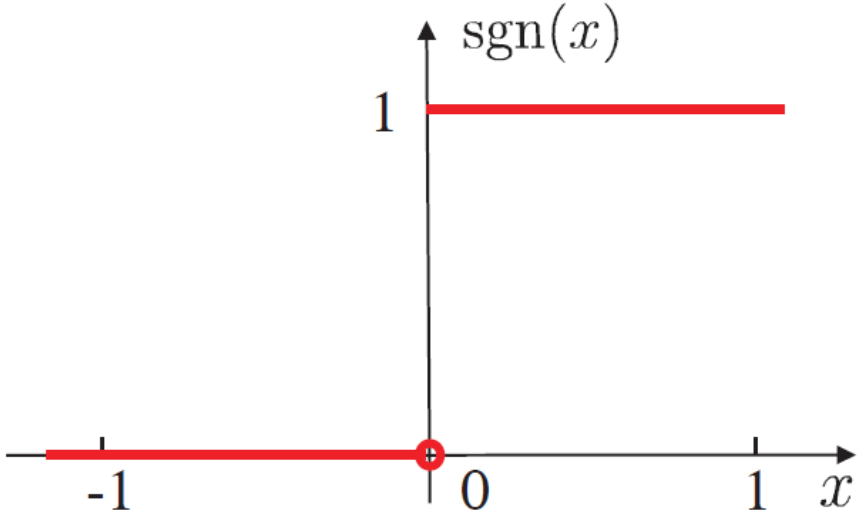
$$z(x_1, x_2, , \ldots, x_d) = z(\mathbf{x}) = \sum_{j=1}^{d} w_j \cdot x_j$$

$$y = g(z(\mathbf{x})) = \begin{cases} 1 & \text{if } z(\mathbf{x}) \geq T \\ 0 & \text{if } z(\mathbf{x}) < T \end{cases}$$

Threshold Activation function

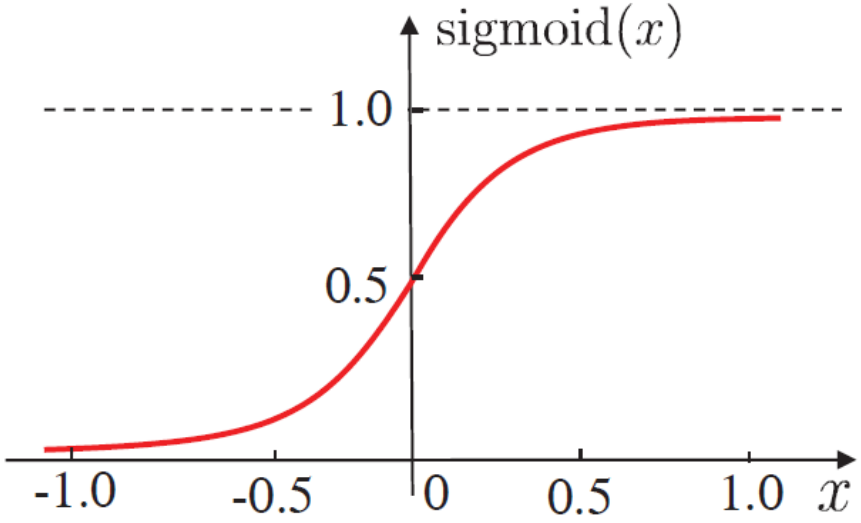
神经元的激活函数

- 理想激活函数是阶跃函数, 0表示抑制神经元而1表示激活神经元
- 阶跃函数具有不连续、不光滑等不好的性质, 常用的是 Sigmoid 函数



$$\text{sgn}(x) = \begin{cases} 1, & \text{if } x \geq 0; \\ 0, & \text{if } x < 0. \end{cases}$$

(a) 阶跃函数



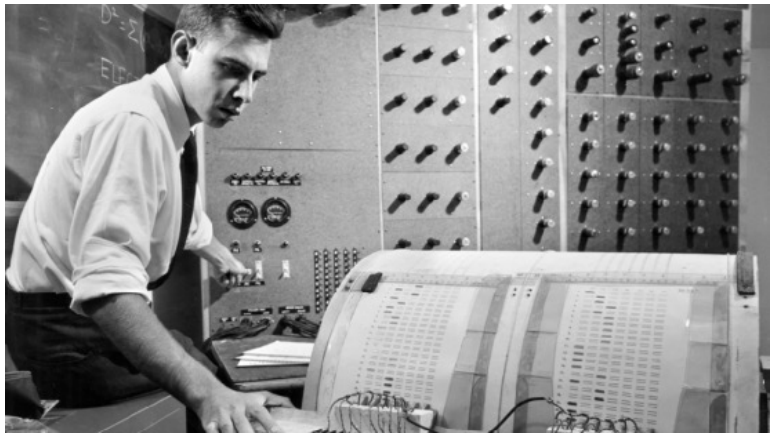
$$\text{sigmoid}(x) = \frac{1}{1+e^{-x}}$$

(b) Sigmoid 函数

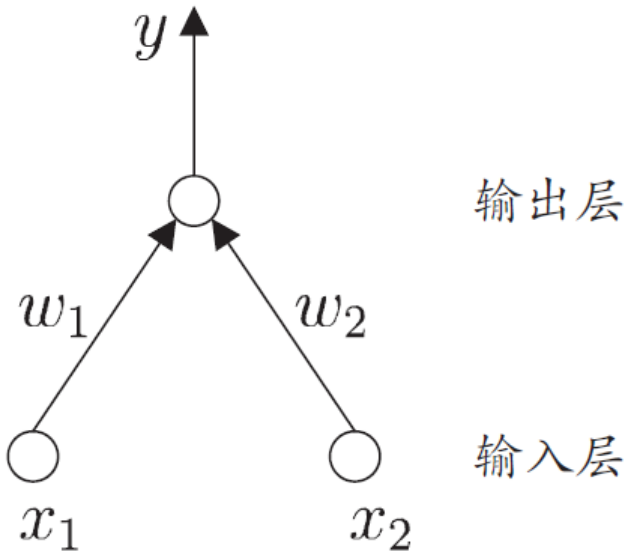
感知机(Perceptron)(1957年)

1957年，康奈尔大学的罗森布拉特(Frank Rosenblatt)在一台IBM-704计算机上模拟实现了感知机模型

旨在处理实数输入，并通过调整权重减少分类错误



Frank Rosenblatt

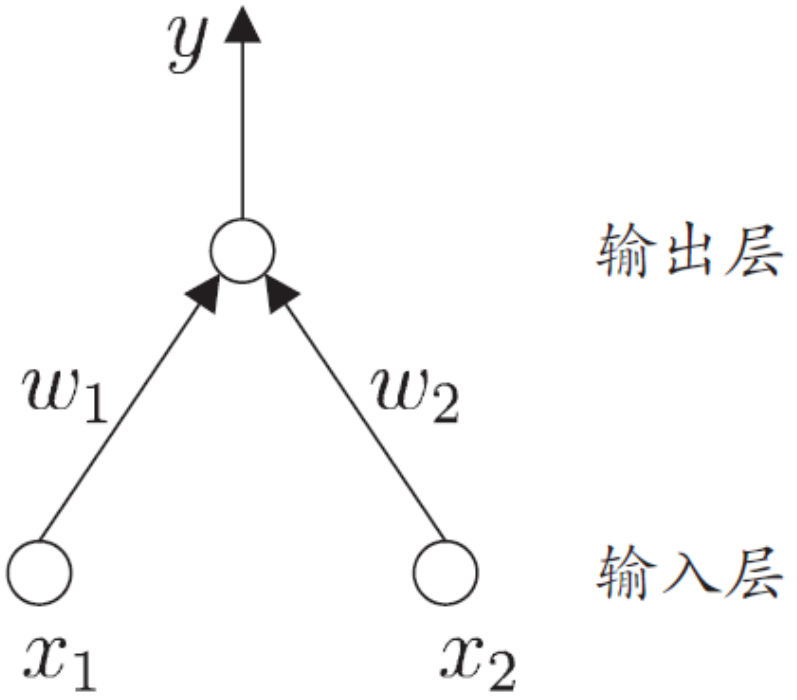


单层感知机

罗森布拉特在理论上证明了单层感知机在处理线性可分的问题时，可以收敛，并以此为基础做了若干感知机有学习能力的实验

感知机(Perceptron)(1957年)

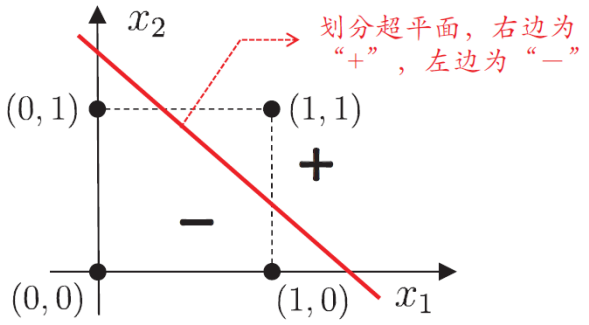
- 感知机能够容易实现逻辑与、或、非运算
- 当w_1, w_2, θ为多少时，可以实现与运算（$x_1 \wedge x_2$）
- 当w_1, w_2, θ为多少时，可以实现或运算（$x_1 \vee x_2$）
- 当w_1, w_2, θ为多少时，可以实现非运算（$\neg x_1$）



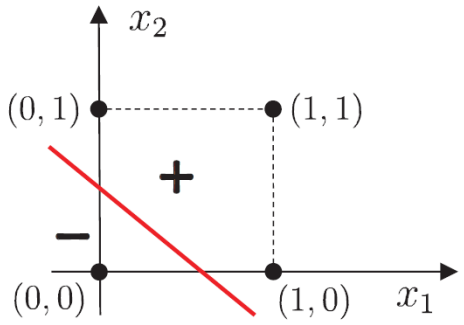
异或问题(1969)

1969年，Marvin Minsky和Seymour Papert在《Perceptrons》中证明

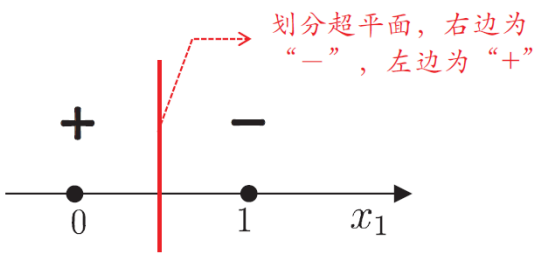
单层感知机不能解决异或XOR问题



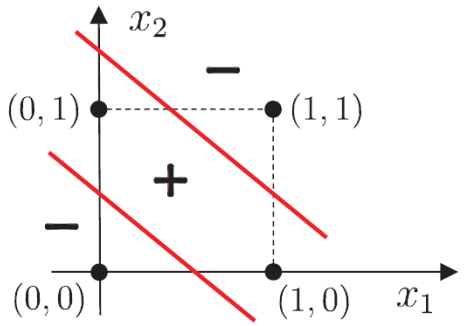
(a) “与”问题 $(x_1 \wedge x_2)$



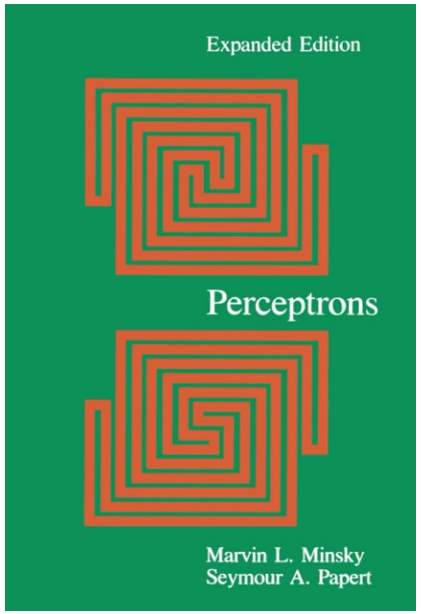
(b) “或”问题 $(x_1 \vee x_2)$



(c) “非”问题 $(\neg x_1)$



(d) “异或”问题 $(x_1 \oplus x_2)$

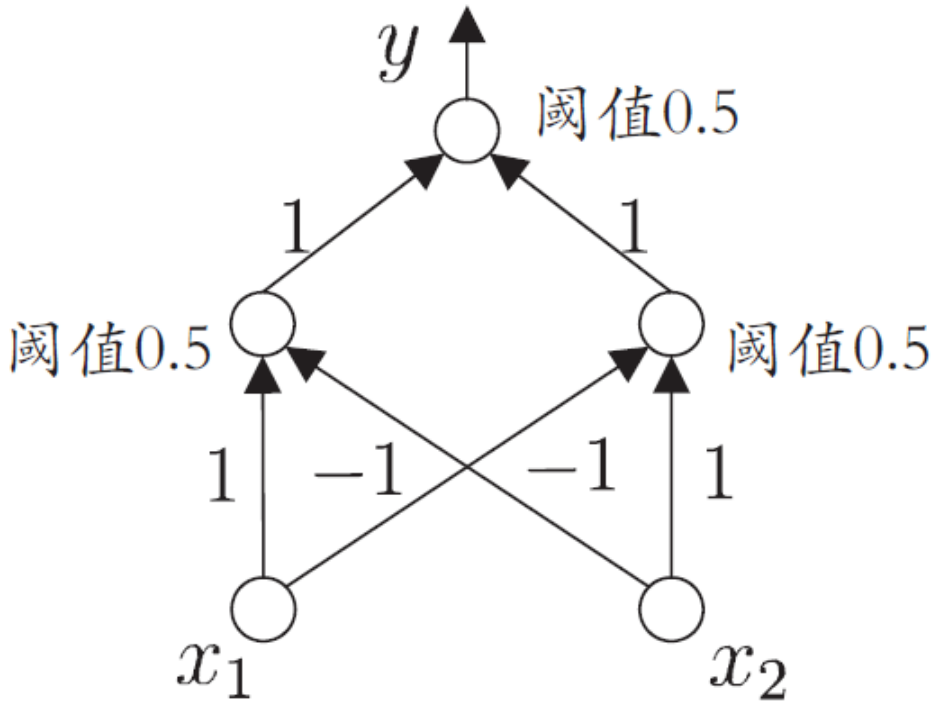


当两类模式线性可分时，则感知机的学习过程一定会收敛；否则感知机的学习过程将会发生震荡

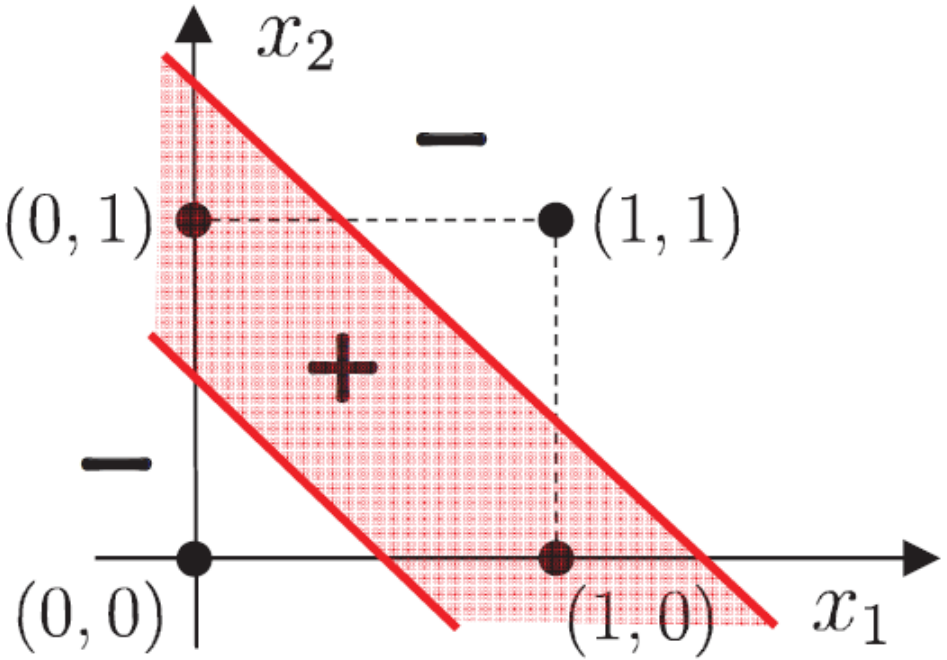
[Minsky and Papert, 1969]

多层感知机(Multi-Layerd Perceptron, MLP)

苏联科学家A. G. Ivakhnenko和V. Lapa在Perceptron的基础上，尝试引入多层结构，为神经网络的发展做出了重要贡献



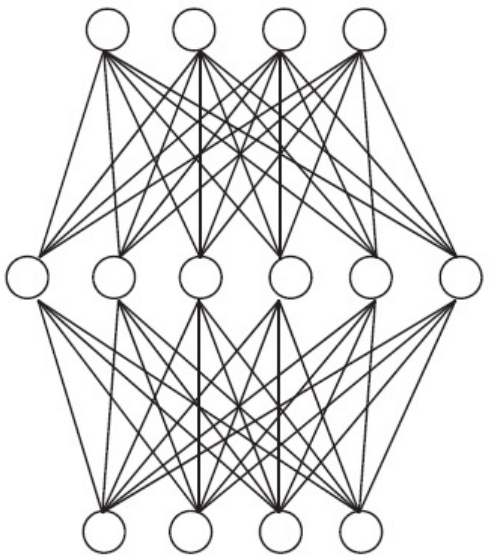
(a) 网络结构



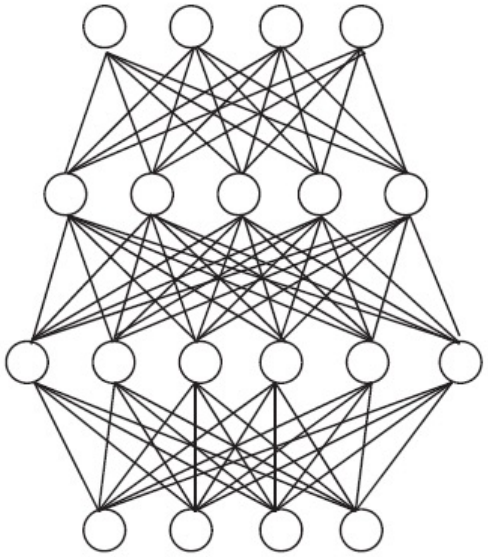
(b) 分类区域

多层前馈神经网络

- 将输出层与输入层之间的一层神经元，称为**隐层或隐含层**
- 隐含层和输出层神经元都是具有激活函数的**功能神经元**
- **前馈网络**：层与层 全连接，但不能存在同层链接和跨层连接



(a) 单隐层前馈网络



(b) 双隐层前馈网络

George Cybenko：多层前馈网络有强大的表示能力（“万有逼近性”）

仅需一个包含足够多神经元的隐层，多层前馈神经网络就能以任意精度逼近任意复杂度的连续函数

并非神经网络特有

挑战

多层神经网络标志着神经网络研究的重大进步，展示了神经网络架构解决复杂问题的潜力。然而，在1960年代和1970年代，若干挑战阻碍了神经网络的发展：

- **缺乏训练算法**：早期的MLP模型缺乏可以有效调整网络权重的有效训练算法，很难训练具有多层的深度网络
- **计算资源受限**：当时可用的计算能力不足以处理训练深度神经网络所需的复杂计算，这种限制减缓了MLP研究和开发的进展
- **数据资源不足**：当时可用的数据不足，难以支撑神经网络这种复杂模型的训练，容易造成过拟合

大纲

- 神经元模型到前馈神经网络
- 参数优化：BP算法
- 深度学习

反向传播算法(1970-1980年代)

- Paul Werbos（1974）：在博士学位论文中第一次正式完整描述反向传播算法的概念，将其作为一种通过误差梯度来调整多层网络权重的方法
- David Rumelhart、Geoffrey Hinton 和 Ronald Williams1986年发表文章《Learning Representations by Back-Propagating Errors》引发了机器学习领域的热潮，也推动了神经网络的复兴，让反向传播算法成为神经网络训练的标准方法，并奠定了现代深度学习的基础

David Rumelhart

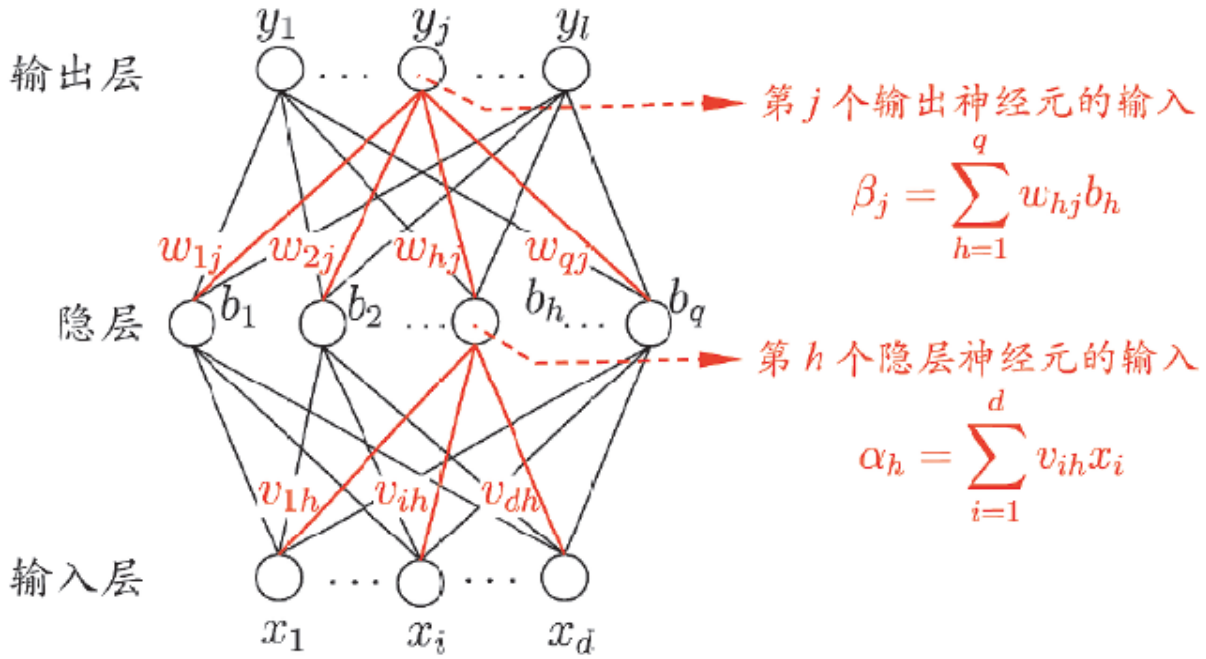
Geoffrey Hinton

Ronald Williams

网络参数优化

给定训练集$D = \{(x_1, y_1), (x_2, y_2), \cdots, (x_m, y_m)\}, x_i \in \mathbb{R}^d, y_i \in \mathbb{R}^l$

- 输入：d 维特征向量
- 输出：l 个输出值
- 隐层：q 个隐层神经元
- 假定功能单元均采用 Sigmoid激活函数



- 需通过学习确定的参数数目：$(d + l + 1)q + l$

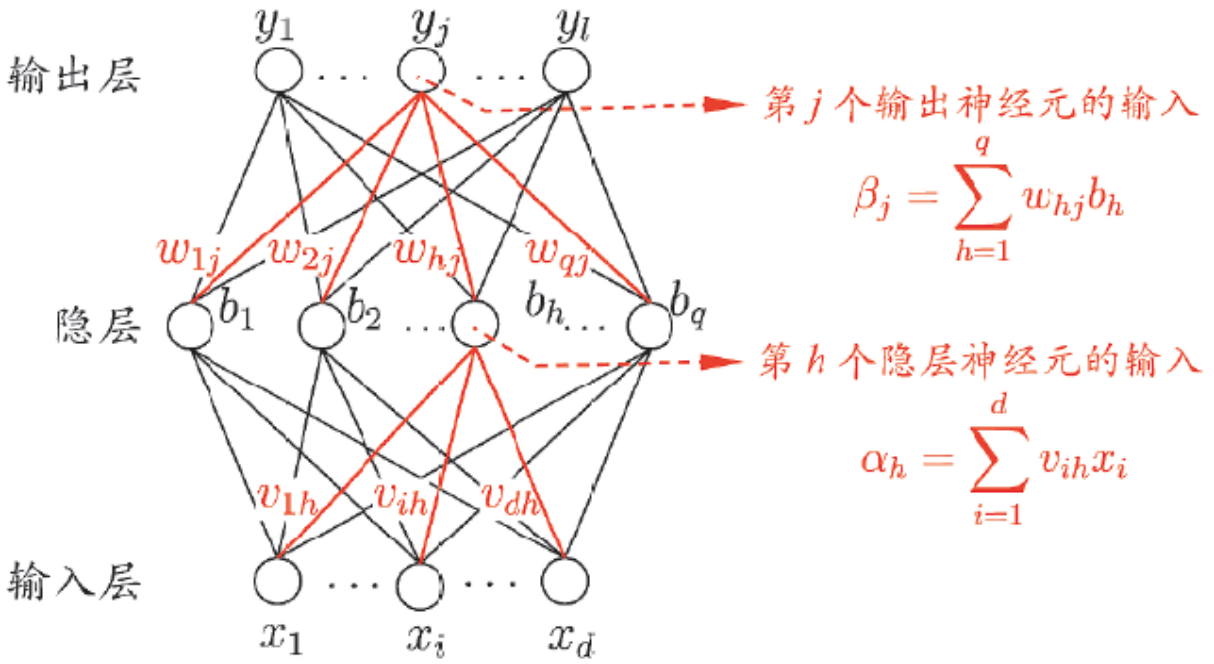
网络参数优化-前向计算

对于训练样本(x_k, y_k), 假定网络的实际输出为$\hat{y}_k = (\hat{y}_1^k, \hat{y}_2^k, \cdots \hat{y}_l^k)$

$$\hat{y}_j^k = f(\beta_j - \theta_j)$$

则网络在样本 (x_k, y_k) 上的均方误差为：

$$E_k = \frac{1}{2}\sum_{j=1}^{l}(\hat{y}_j^k - y_j^k)^2$$



如何根据误差优化网络的参数？

BP (BackPropagation：误差逆传播算法)

迄今最成功、最常用的神经网络算法，可用于多种任务（不仅限于分类）

对于训练样本(x_k, y_k), 假定网络的实际输出为$\hat{y}_k = (\hat{y}_1^k, \hat{y}_2^k, \cdots \hat{y}_l^k)$

$$\hat{y}_j^k = f(\beta_j - \theta_j)$$

则网络在样本 (x_k, y_k) 上的均方误差为：

$$E_k = \frac{1}{2}\sum_{j=1}^{l}\left(\hat{y}_j^k - y_j^k\right)^2$$

BP 是一个基于梯度下降的迭代学习算法, 对于网络中的每一个参数w在迭代的每一轮中更新方式如下：

$$w = w - \eta\frac{\partial E}{\partial w}$$

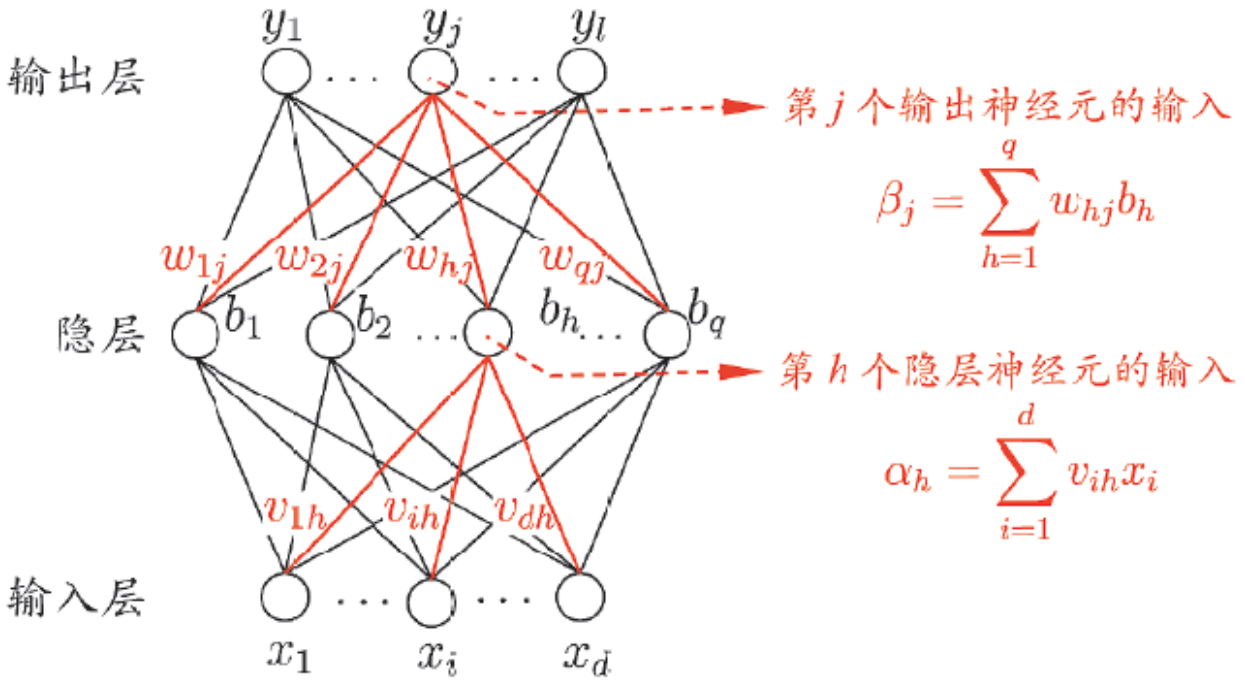
BP (BackPropagation：误差逆传播算法)

以w_{hj}为例

对误差$E_k = \frac{1}{2}\sum_{j=1}^{l}(\hat{y}_j^k - y_j^k)^2$,

如何计算

$$\frac{\partial E_k}{\partial w_{hj}}$$



注意到w_{hj}先影响到β_j,

再影响到$\hat{y}_j^k$, 然后才影响到E_k, 有:

$$\frac{\partial E_k}{\partial w_{hj}} = \frac{\partial E_k}{\partial \hat{y}_j^k} \cdot \frac{\partial \hat{y}_j^k}{\partial \beta_j} \cdot \frac{\partial \beta_j}{\partial w_{hj}}$$

⇐ “链式法则”

BP (BackPropagation：误差逆传播算法)

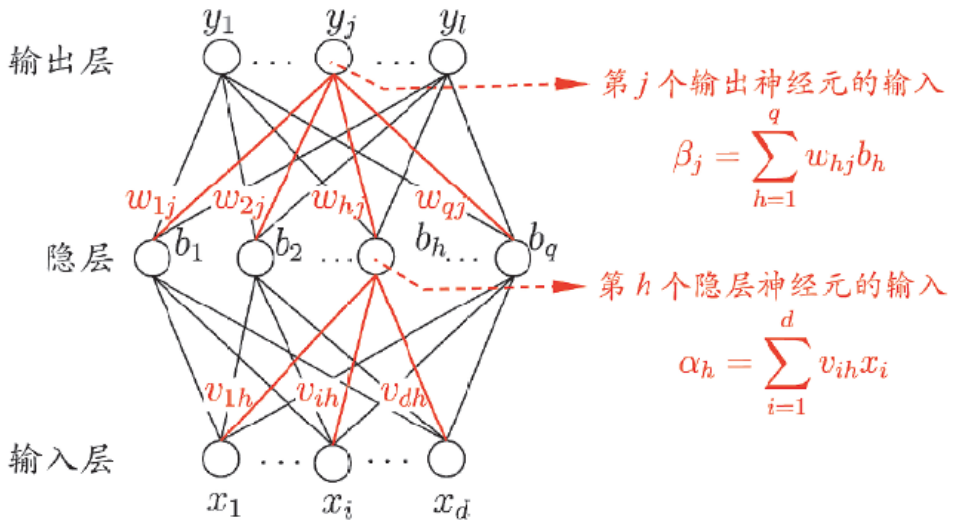
$$\frac{\partial E_k}{\partial w_{hj}} = \frac{\partial E_k}{\partial \hat{y}_j^k} \cdot \frac{\partial \hat{y}_j^k}{\partial \beta_j} \cdot \frac{\partial \beta_j}{\partial w_{hj}}$$

$$\frac{\partial E_k}{\partial \hat{y}_j^k} = (\hat{y}_j^k - y_j^k)$$

$$\frac{\partial \beta_j}{\partial w_{hj}} = b_h$$

$$\frac{\partial \hat{y}_j^k}{\partial \beta_j} = f'(\beta_j - \theta_j) = \hat{y}_j^k(1 - \hat{y}_j^k)$$

注意到 $\hat{y}_j^k = f(\beta_j - \theta_j)$



对 $\text{sigmoid}(x) = \frac{1}{1+e^{-x}}$

$$f'(x) = f(x)(1 - f(x))$$

令 $g_j = -\frac{\partial E_k}{\partial \hat{y}_j^k} \cdot \frac{\partial \hat{y}_j^k}{\partial \beta_j}$

$$= \hat{y}_j^k(1 - \hat{y}_j^k)(y_j^k - \hat{y}_j^k)$$

于是，$\Delta w_{hj} = -\eta \frac{\partial E_k}{\partial w_{hj}} = \eta g_j b_h$

BP (BackPropagation：误差逆传播算法)

类似地，有：

$$\Delta\theta_j = -\eta g_j$$

$$\Delta v_{ih} = \eta e_h x_i$$

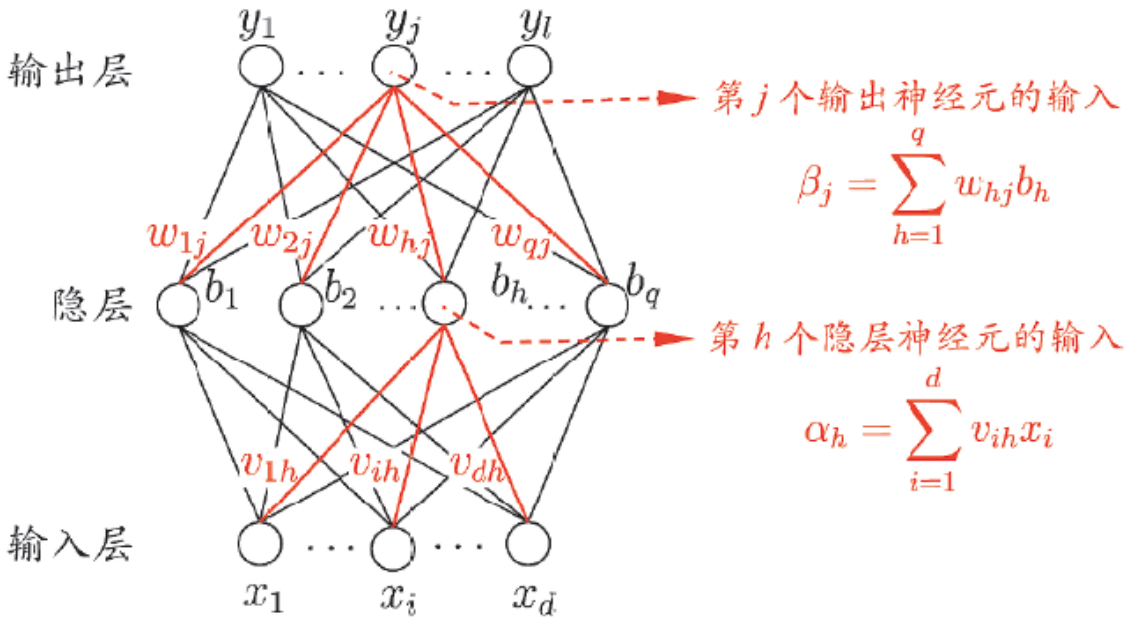
$$\Delta\gamma_h = -\eta e_h$$

其中：

$$e_h = -\frac{\partial E_k}{\partial b_h}\cdot\frac{\partial b_h}{\partial \alpha_h}$$

$$= -\sum_{j=1}^{l}\frac{\partial E_k}{\partial \beta_j}\cdot\frac{\partial \beta_j}{\partial b_h}f'(\alpha_h-\gamma_h) \quad = \sum_{j=1}^{l} w_{hj}g_j f'(\alpha_h-\gamma_h)$$

$$= b_h(1-b_h)\sum_{j=1}^{l} w_{hj}g_j$$



多种实现方式

批量梯度下降 (batch gradient descent)

- 每次在整个训练集上计算损失误差
- 每次需要读取完整数据，消耗内存，收敛速度较慢

随机梯度下降 (stochastic gradient descent)

- 每次随机选一个样本计算损失误差
- 参数更新频繁，不同样例可能抵消

小批量梯度下降 (mint-batch gradient descent)

- 每次随机选一批样本计算损失误差读取整个训练集
- 训练过程更稳定，如何设置batch-size是个难题

一些变种：SGD、SGD with Momentum、Adagrad (Adaptive Gradient Algorithm)、RMSprop (Root Mean Square Propagation)、Adam (Adaptive Moment Estimation), etc

An overview of gradient descent optimization algorithms. https://arxiv.org/pdf/1609.04747

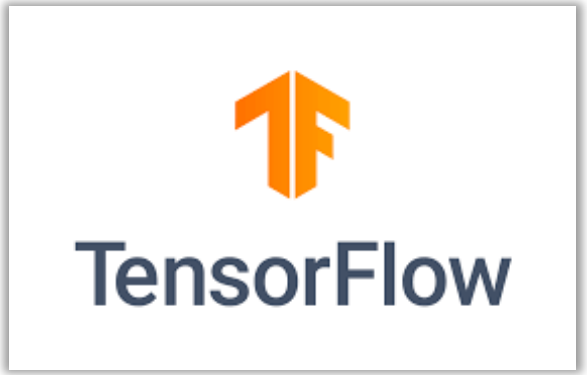
两个名词：Epoch和Iteration

- Epoch：轮次，整个训练集被完整的训练一遍
- Iteration：迭代步，一次参数更新

```
for epoch in range(num_epochs):                # epoch 层
    for i, (x_batch, y_batch) in enumerate(train_loader):  # iteration 层
        y_pred = model(x_batch)
        loss = criterion(y_pred, y_batch)
        loss.backward()
        optimizer.step()
```

深度学习框架

- 自动求导
- 内置常用优化算法与损失函数
- 无缝CPU和GPU切换

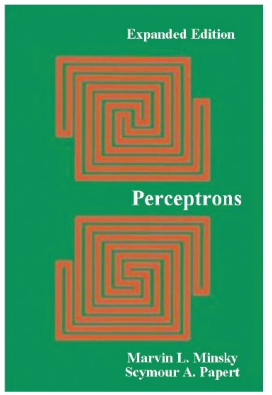


神经网络发展回顾

1940年代-萌芽期：M-P模型 (1943)

1956左右-1969左右~繁荣期：感知机 (1958), Adaline (1960), …

1969年：Minsky & Papert “Perceptrons”



马文·闵斯基
(1927-2016)
1969年图灵奖

冰河期

1984左右 -1997左右~繁荣期：Hopfield (1983), BP (1986), …

1997年左右：SVM文本分类成功 及 统计学习 兴起

沉寂期

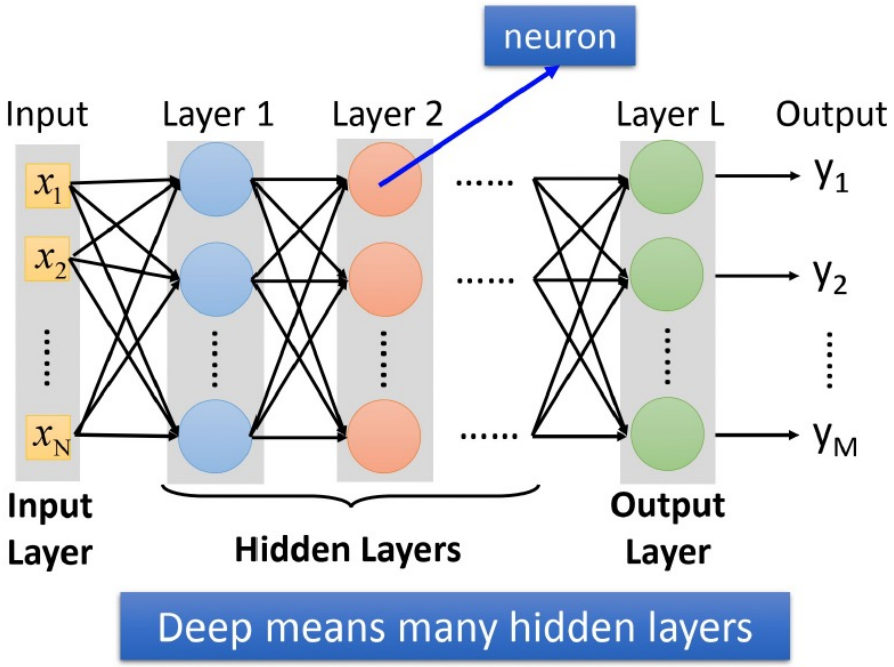
2012-至今~繁荣期：深度学习

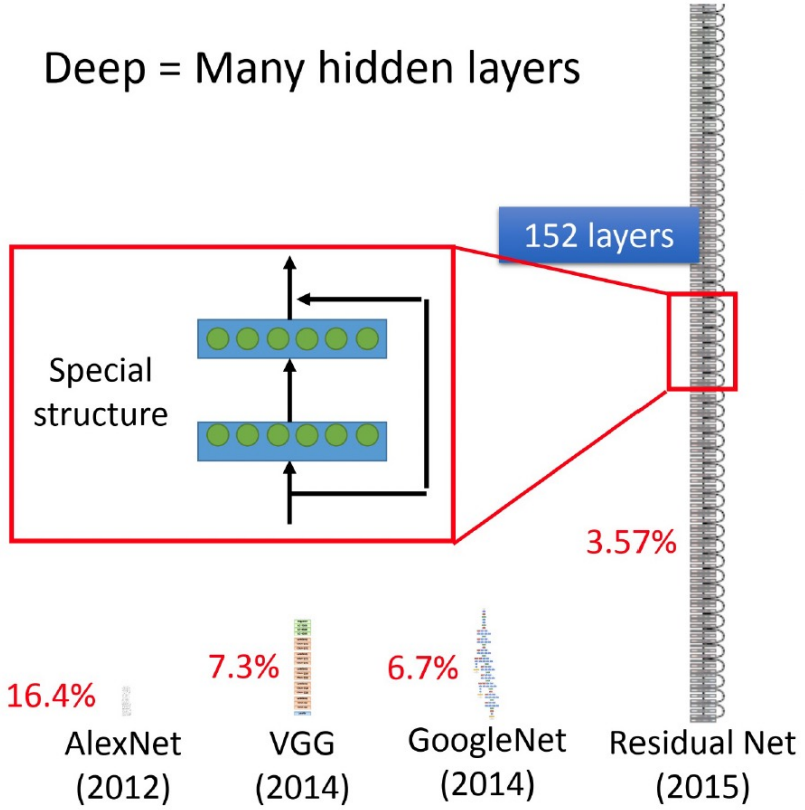
大纲

- 神经元模型到前馈神经网络
- 参数优化：BP算法
- 深度学习

深度学习的兴起

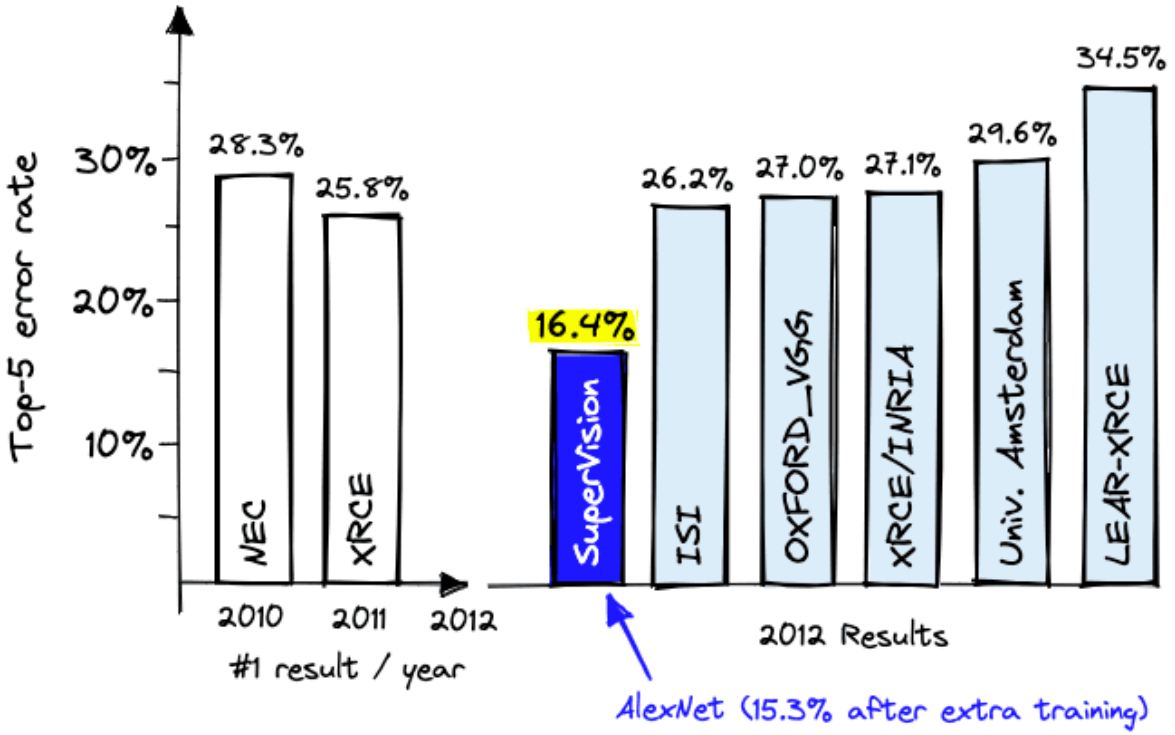
- 2006年, Hinton 在 Science 发表文章《Reducing the dimensionality of data with neural networks》，提出深度信念网络（Deep Belief Networks，DBNs）
- **深度学习模型就是具有很多个隐层的神经网络**

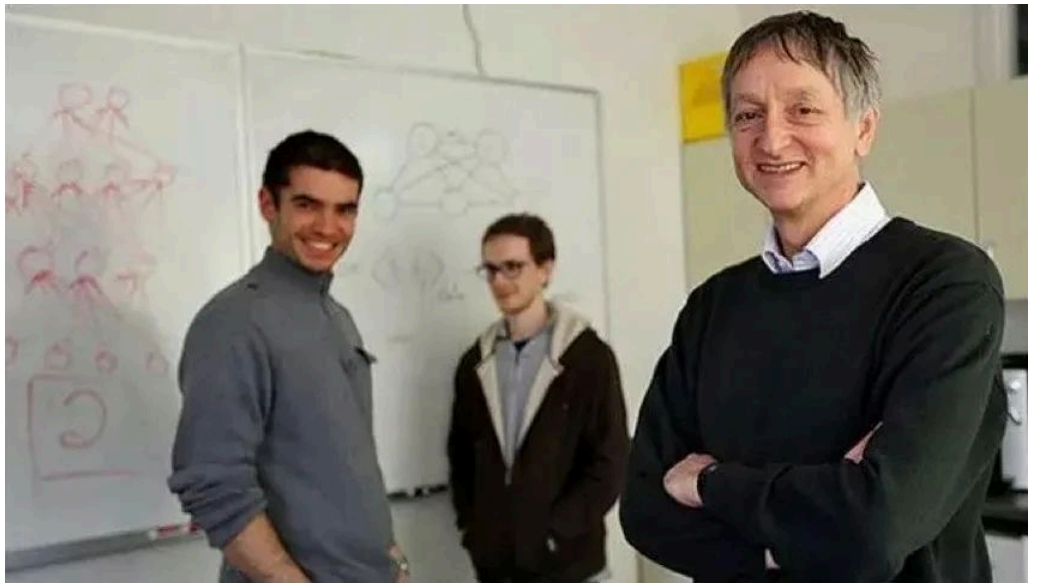




深度学习的兴起

2012年, Hinton 组参加ImageNet 竞赛, 使用一个名为AlexNet的CNN 模型以超过第二名10个百分点的成绩夺得当年竞赛的冠军





Ilya Sutskever， Alex Krizhevsky

深度学习的兴起

多层前馈网络有强大的表示能力(“万有逼近性”)

仅需一个包含足够多神经元的隐层，多层前馈神经网络就能以任意精度逼近任意复杂度的连续函数

为什么之前没人做深度神经网络？

一方面，计算能力的大幅提高缓解了训练效率

另一方面，训练数据的大幅增加降低了过拟合风险

因此，以深度神经网络为代表的复杂机器学习模型成为了合适的选择

深度学习的兴起

多层前馈网络有强大的表示能力（“万有逼近性”）

仅需一个包含足够多神经元的隐层，多层前馈神经网络就能以任意精度逼近任意复杂度的连续函数

- 增加模型复杂程度的方式
  - 模型宽度：增加隐层神经元的数目
  - 模型深度：增加隐层数目

为何是深度学习不是宽度学习？

理解深度学习

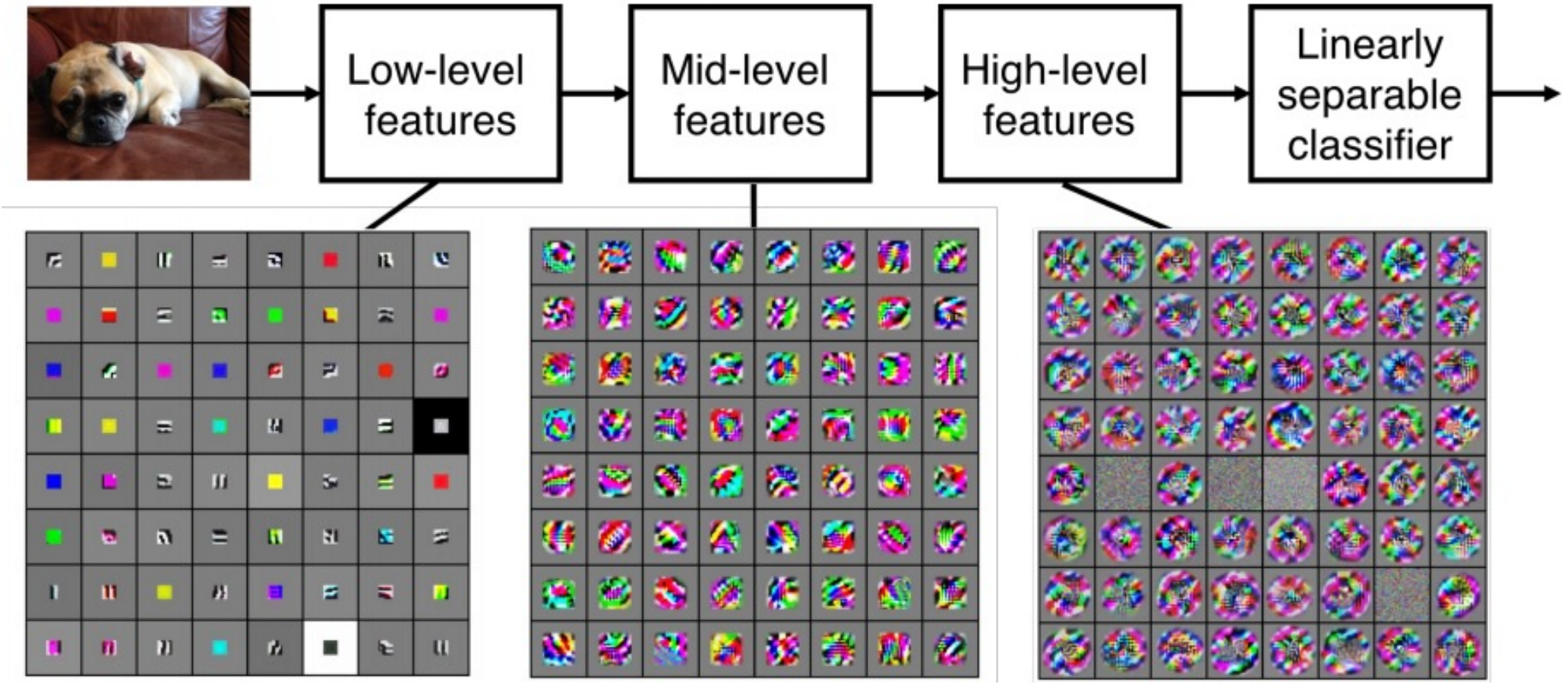
从“特征工程” 到“特征学习”或“表示学习”

特征工程由人类专家根据现实任务来设计，特征提取与识别是分开的两个阶段

特征学习通过深度学习自动产生有益于分类的特征，是一个端到端(end-to-end)的学习框架

理解深度学习

深度学习提供了层次化特征提取的能力



深度学习的挑战

- 复杂优化问题
- 梯度消失与梯度爆炸
- 超参数选择困难
- 过拟合
- 计算资源开销
- 可解释性
- 稳健性
- …

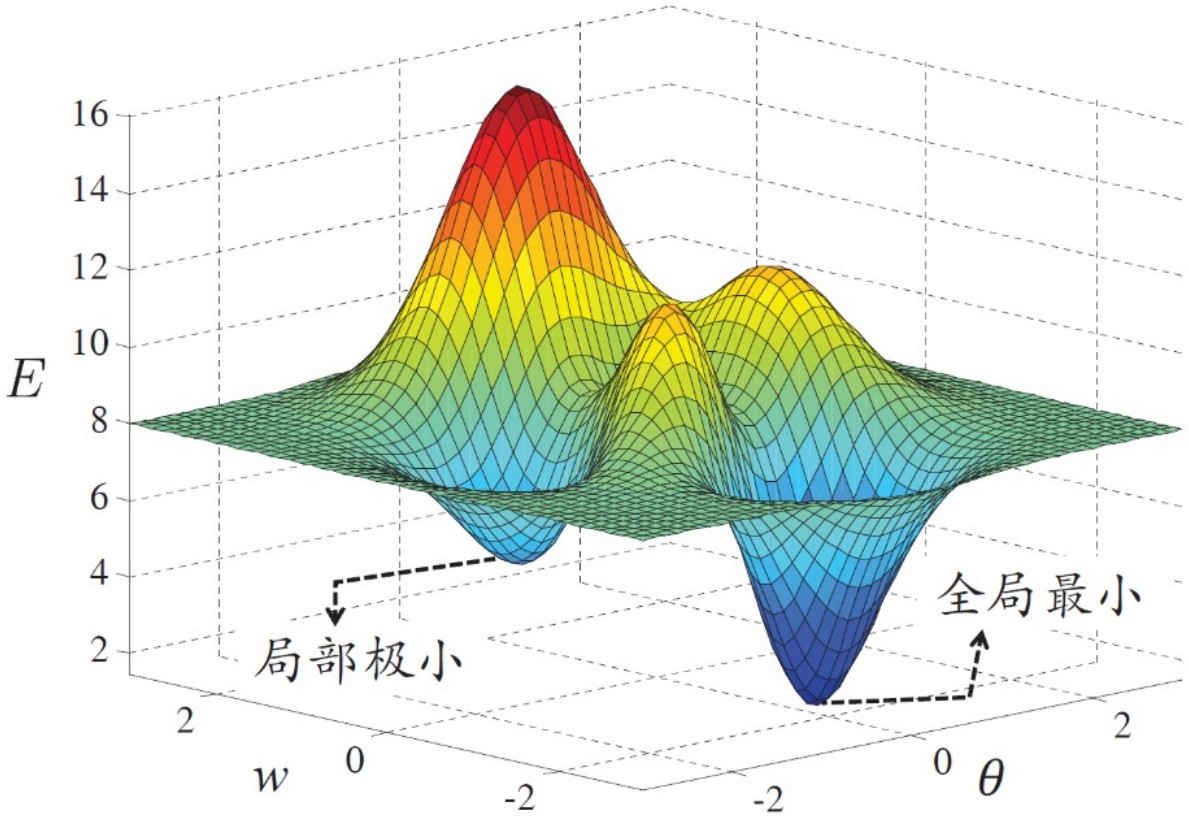
深度学习工程师 → 炼丹术士

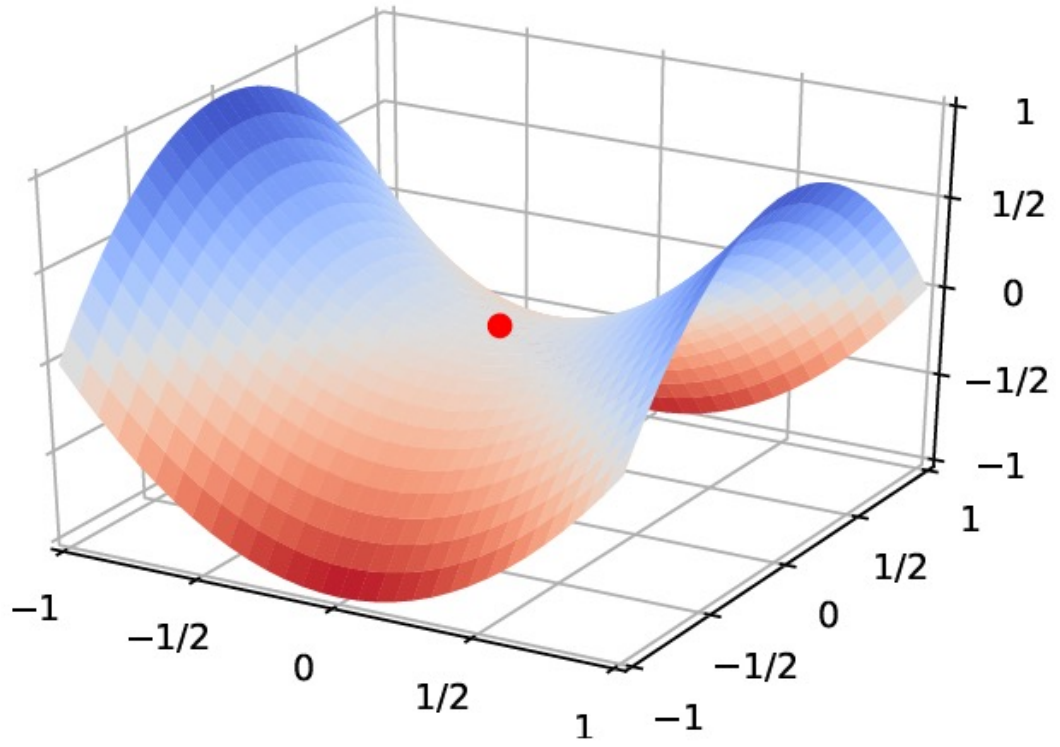
深度学习的挑战

- 复杂优化问题
- 梯度消失与梯度爆炸
- 超参数选择困难
- 过拟合
- 计算资源开销
- 可解释性
- 稳健性
- …

非凸优化(Non-Convex Optimization)

神经网络通常是非凸的：难以找到全局最优解、鞍点(saddle point)问题





非凸优化(Non-Convex Optimization)

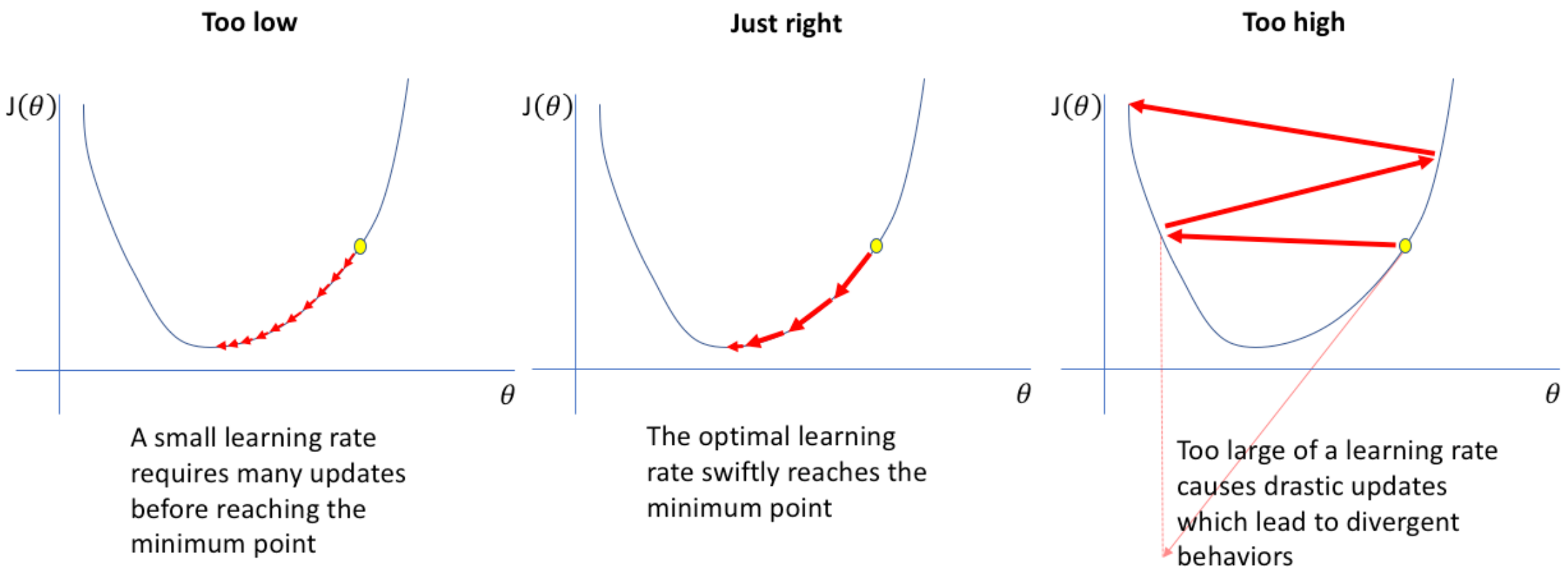
神经网络最大的谜团之一：优化是如此困难，为什么效果还能这么好？

目前的理解：

- **过参数化**：过参数化让许多不同参数组合都能实现相同的低损失，从而形成大量“良性平坦极小值”
- **足够好的局部最优**：在大型、过度参数化的神经网络中，大部分局部最优解可能已经足够好了
- **逃离鞍点**：SGD的随机性，往往可以跳出鞍点

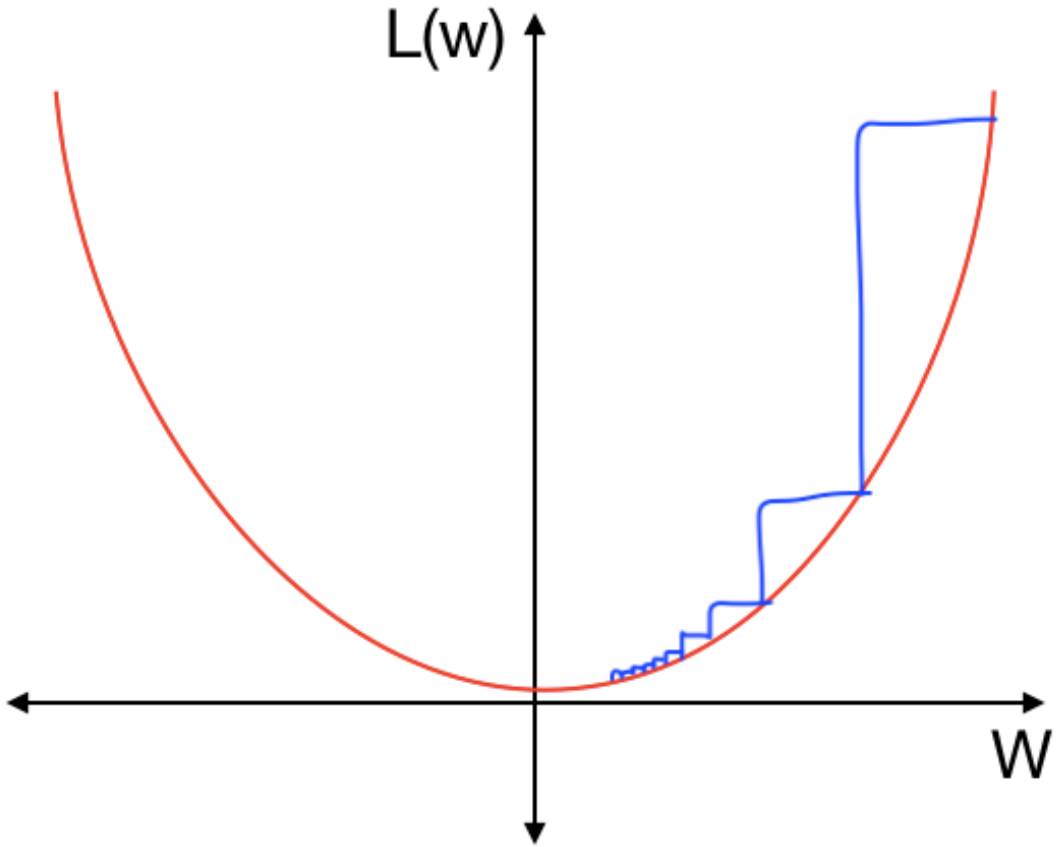
Some Tricks for optimization

学习率的影响



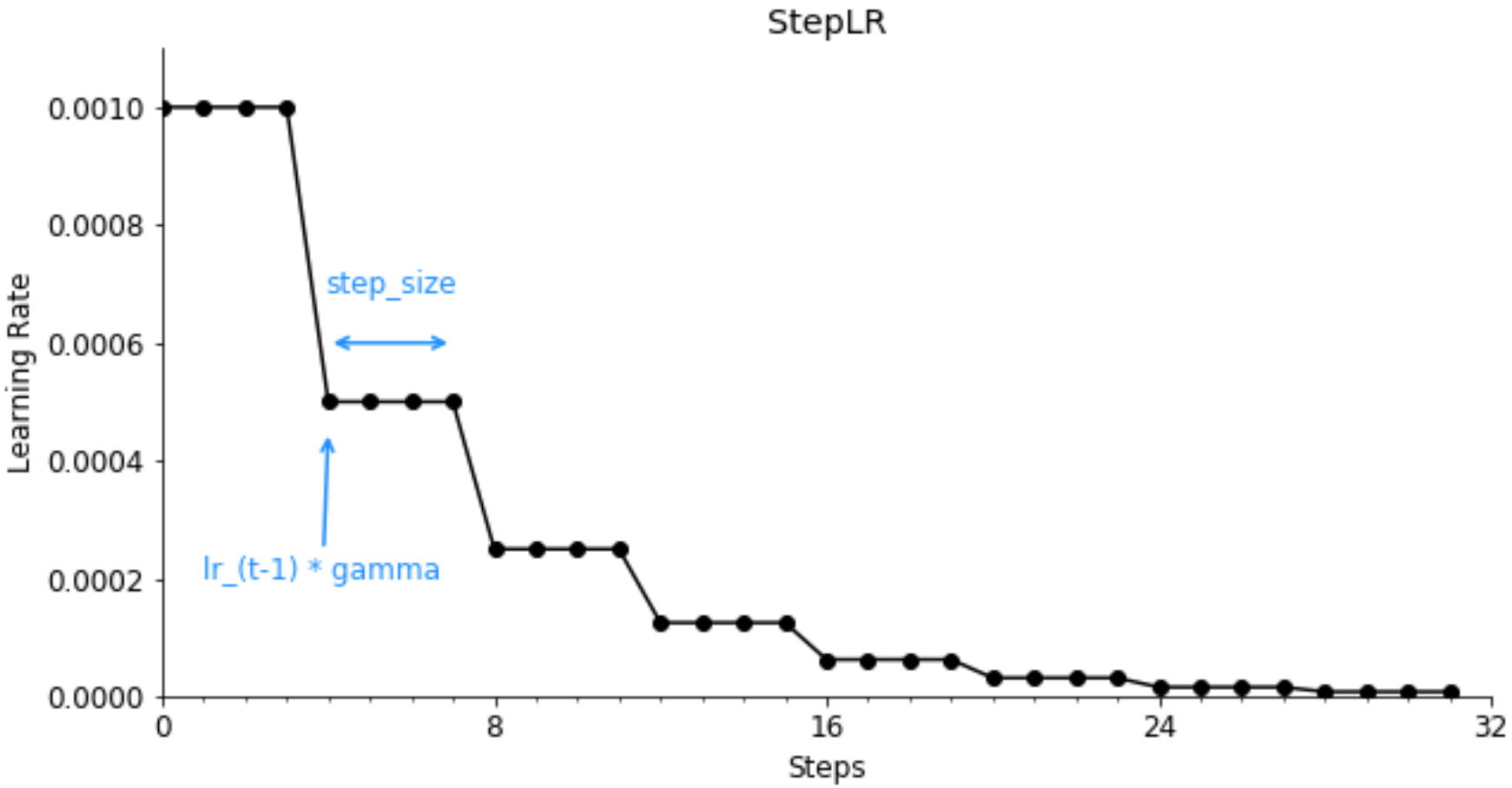
Some Tricks for optimization

学习率衰减 (learning rate decay)



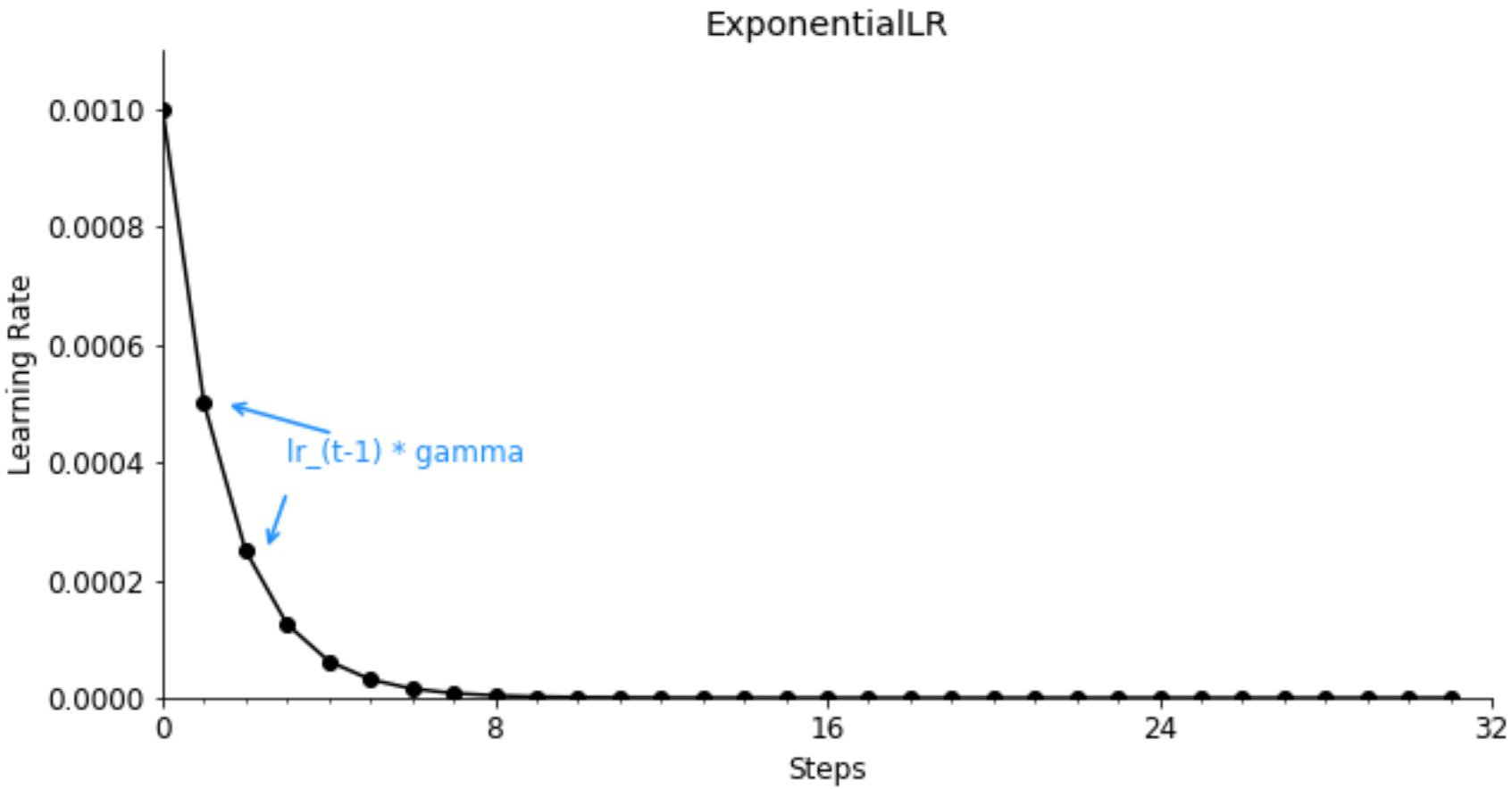
Some Tricks for optimization

学习率衰减 (learning rate decay)



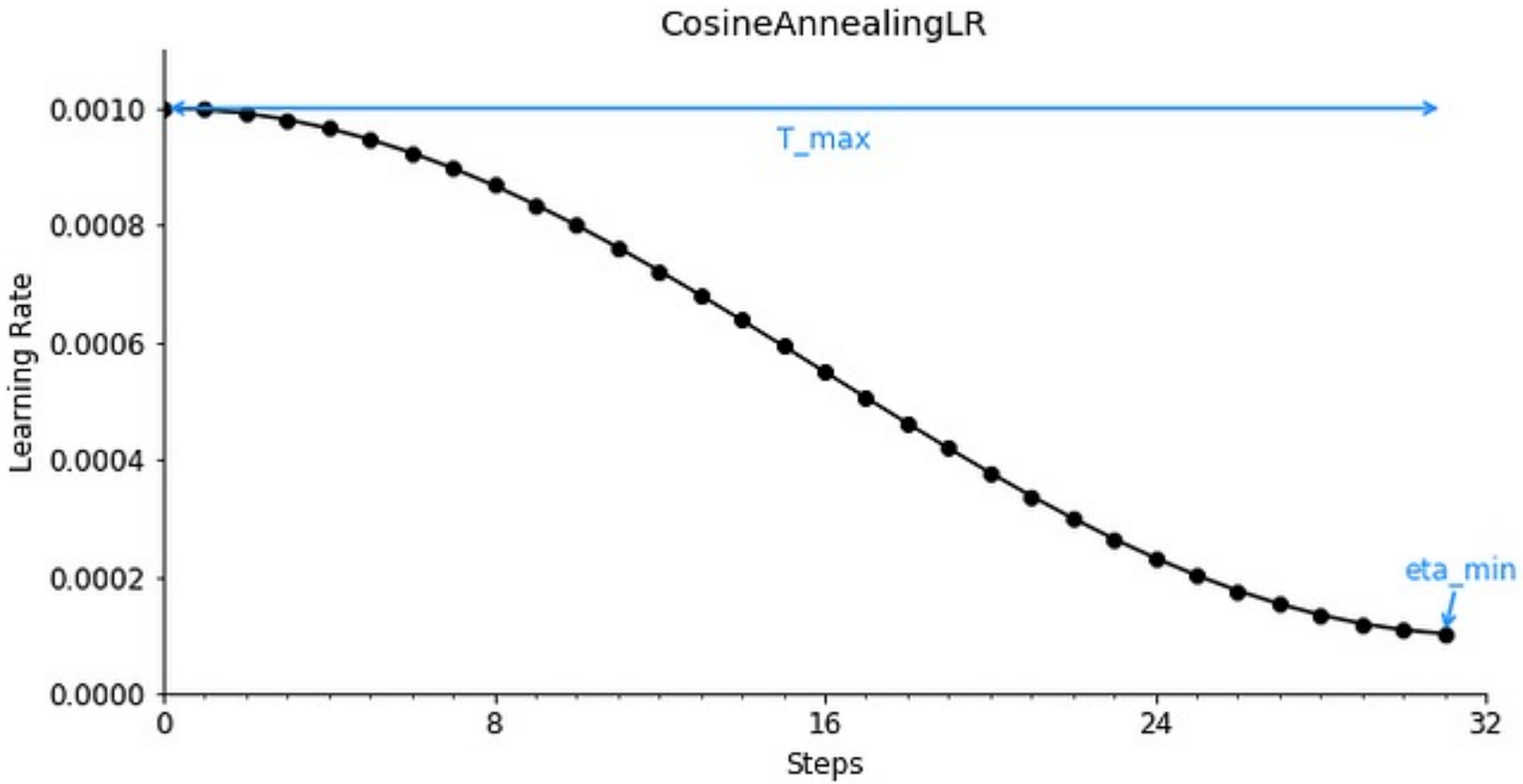
Some Tricks for optimization

学习率衰减 (learning rate decay)



Some Tricks for optimization

学习率衰减 (learning rate decay)



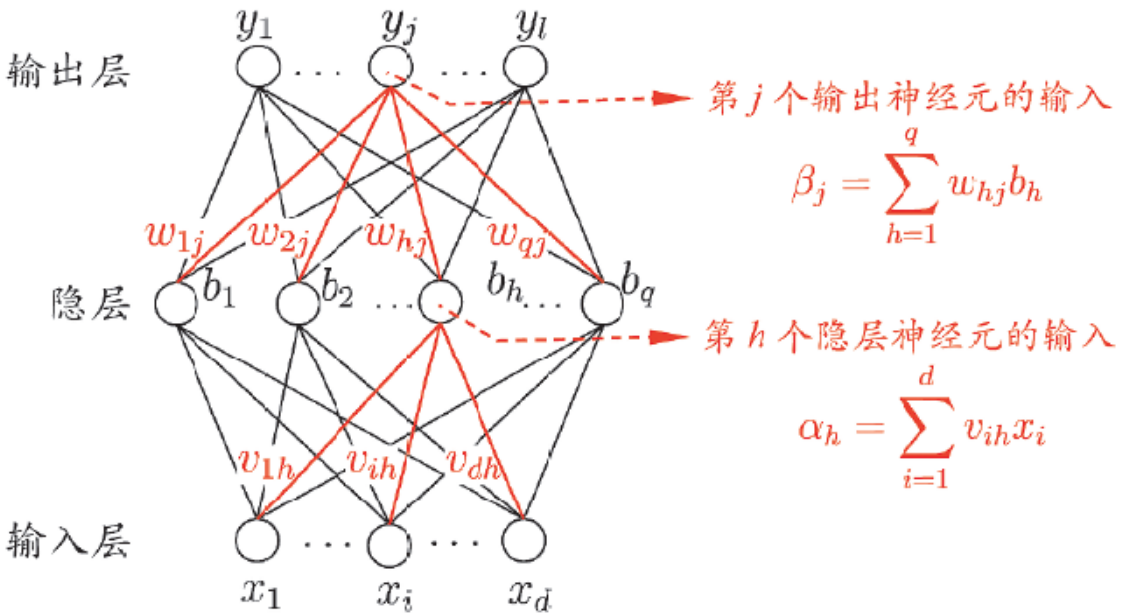
Some Tricks for optimization

```python
import torch
import torch.nn as nn
import torch.optim as optim

# 模拟一个简单模型
model = nn.Linear(10, 2)
optimizer = optim.SGD(model.parameters(), lr=0.1)

# StepLR: 每隔 step_size 个 epoch，将学习率乘以 gamma
scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=10, gamma=0.1)

num_epochs = 30
for epoch in range(num_epochs):
    # 假设每轮训练过程
    optimizer.zero_grad()
    output = model(torch.randn(8, 10))
    loss = output.mean()
    loss.backward()
    optimizer.step()

    # 更新学习率（每个 epoch 后调用一次）
    scheduler.step()

    # 打印当前学习率
    lr = optimizer.param_groups[0]['lr']
    print(f"Epoch [{epoch+1}/{num_epochs}] - Learning Rate: {lr:.5f}")
```

深度学习的挑战

- 复杂优化问题
- **梯度消失与梯度爆炸**
- 超参数选择困难
- 过拟合
- 计算资源开销
- 可解释性
- 稳健性
- …

梯度消失与梯度爆炸



$$\frac{\partial E_k}{\partial w_{hj}} = \frac{\partial E_k}{\partial \hat{y}_j^k} \cdot \frac{\partial \hat{y}_j^k}{\partial \beta_j} \cdot \frac{\partial \beta_j}{\partial w_{hj}}$$

反向传播链式法则中的导数连乘导致梯度消失或爆炸

缓解梯度消失的Tricks

新型激活函数

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

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

$$= \max(0, x).$$

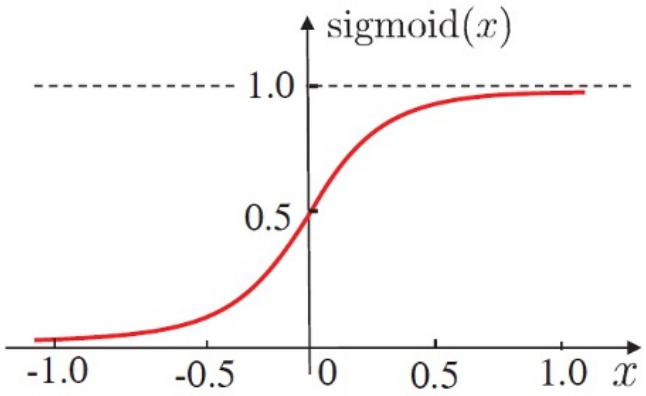
$$\mathrm{LeakyReLU}(x) = \begin{cases} x & \text{if } x > 0 \\ \gamma x & \text{if } x \leq 0 \end{cases}$$

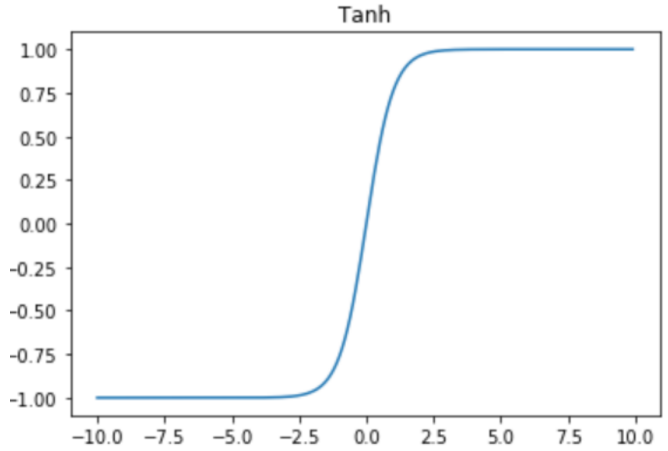
$$= \max(0, x) + \gamma \min(0, x)$$

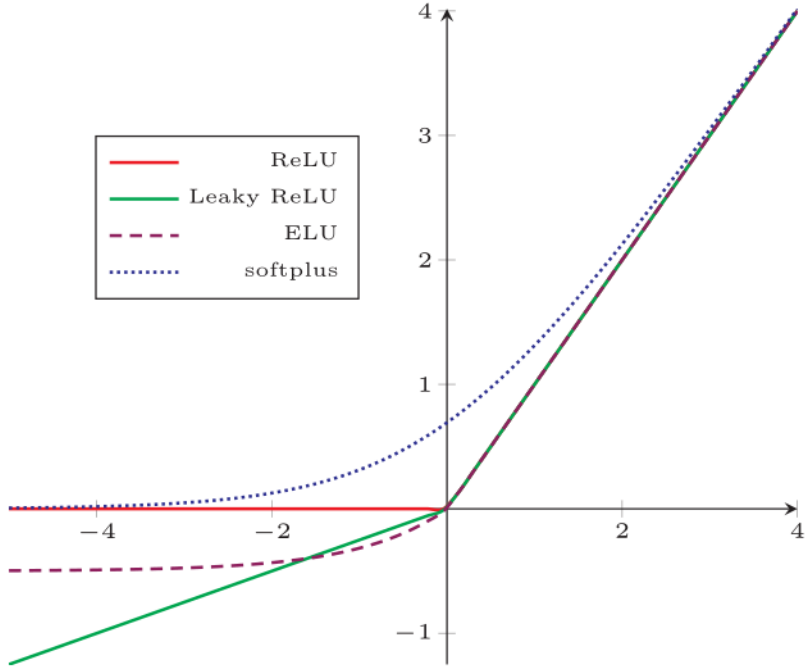
$$\mathrm{ELU}(x) = \begin{cases} x & \text{if } x > 0 \\ \gamma(\exp(x) - 1) & \text{if } x \leq 0 \end{cases}$$

$$= \max(0, x) + \min(0, \gamma(\exp(x) - 1))$$

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







分析：求导更容易、导数值更大

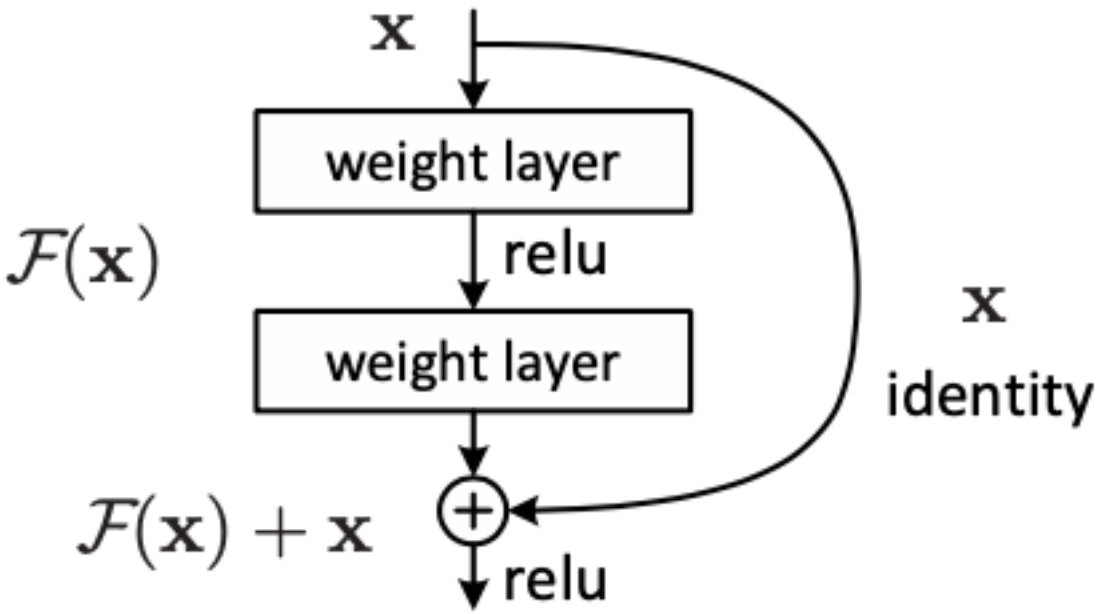
缓解梯度消失的Tricks

残差连接

- 假设在一个深度网络中，我们期望 f(x,θ) 去逼近目标函数为 h(x)
- 将目标函数拆分成两部分：恒等函数和残差函数

$$h(\mathbf{x}) = \underbrace{\mathbf{x}}_{\text{恒等函数}} + \underbrace{(h(\mathbf{x}) - \mathbf{x})}_{\text{残差函数}}$$

$$f(\mathbf{x}, \theta)$$



缓解梯度消失的Tricks

Batch Normalization

在每层网络中，都对输入特征进行归一化，让分布稳定在均值0、方差1附近

假设输入是一个batch的样本B, 样本x是这个batch中的样本, γ和β是可学习参数

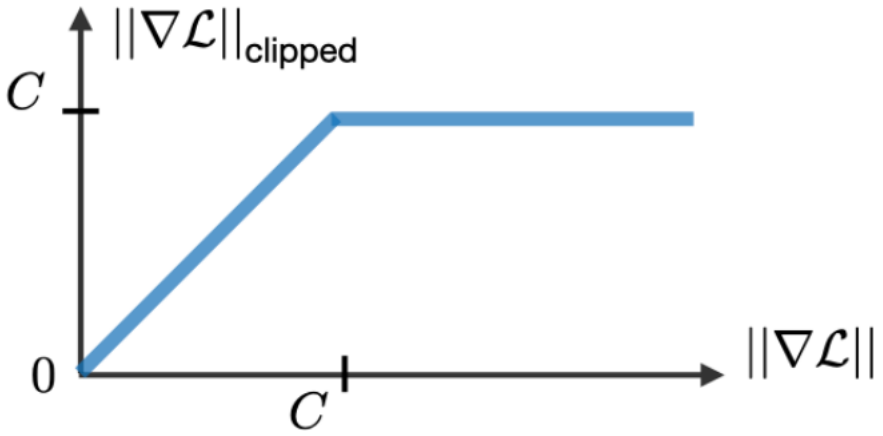
$$BN(x) = \gamma \frac{x - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} + \beta$$

分析：避免进入激活函数的梯度饱和区

Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. https://arxiv.org/abs/1502.03167

缓解梯度爆炸的Tricks

- **梯度裁剪(Gradient Clipping)：**限制梯度的阈值，如果超过某个设定的阈值，就进行截断
- **合理的权重初始化**
- **Batch Normalization**
- **权重正则化(L1/L2 Regularization)**
- ……



参数初始化的Tricks

- 参数不能初始化为0！为什么？
  - **对称性问题**

- 一些常用初始化方法
  - Gaussian分布初始化：参数从一个固定均值（比如0）和固定方差（比如0.01）的Gaussian分布进行随机初始化
  - 均匀分布初始化：参数可以在区间[−r, r]内采用均匀分布进行初始化

参数初始化的Tricks

- Xavier 初始化：
  - 均值为0，方差为 $\left(\frac{2}{n_{in}+n_{out}}\right)$ 的高斯分布
  - 适用于以Sigmoid、Tanh为激活函数的MLP网络

- He 初始化：
  - 均值为0，方差为 $\left(\frac{2}{n_{in}}\right)$ 的高斯分布
  - 适用于以Relu为激活函数的CNN网络

深度学习的挑战

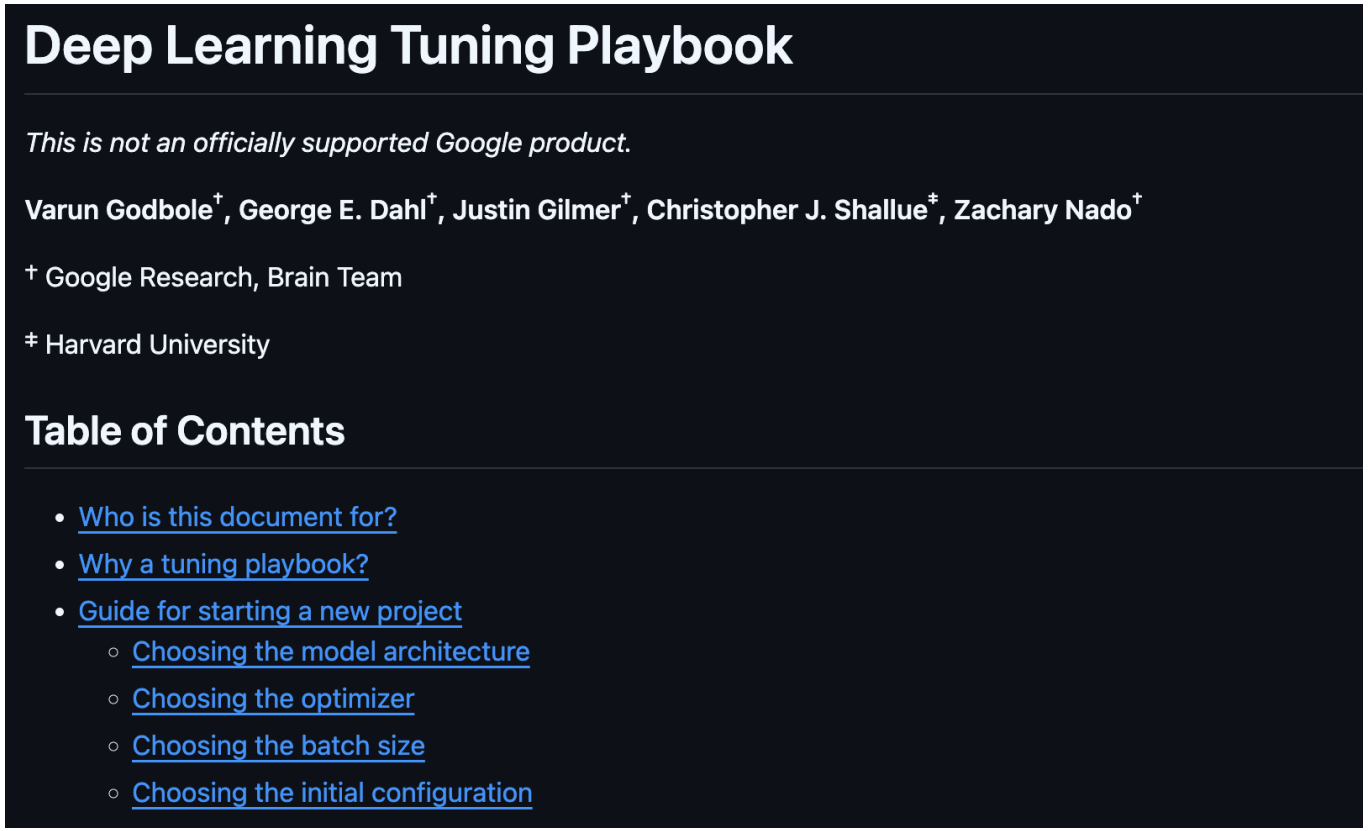
- 复杂优化问题
- 梯度消失与梯度爆炸
- **超参数选择困难**
- 过拟合
- 计算资源开销
- 可解释性
- 稳健性
- …

超参数优化

超参数

- 网络层数
- 每层神经元的个数
- 激活函数
- 优化算法
- 学习率
- Mini-batch大小
- ……

没有理论解法，实际只能采用“试错法”



Deep Learning Tuning Playbook

This is not an officially supported Google product.

Varun Godbole[†], George E. Dahl[†], Justin Gilmer[†], Christopher J. Shallue[‡], Zachary Nado[†]

[†] Google Research, Brain Team

[‡] Harvard University

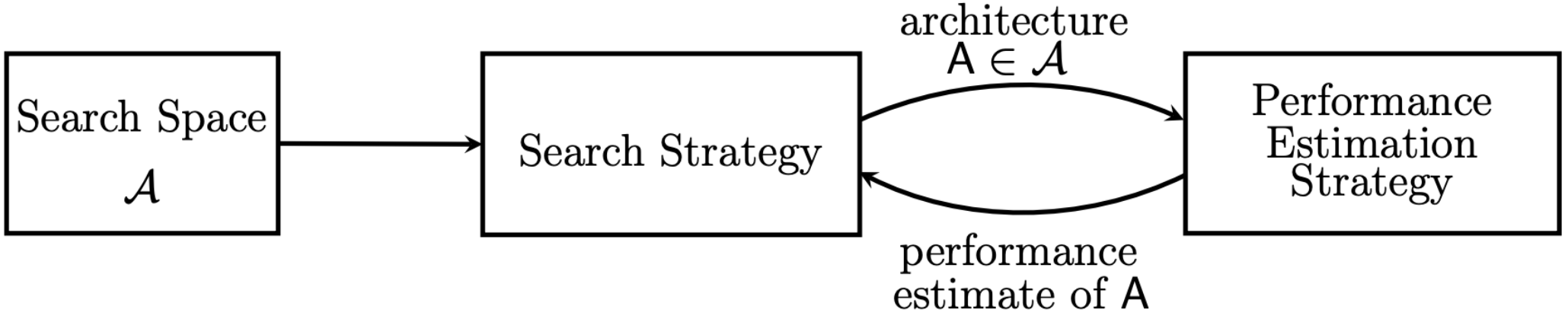
Table of Contents

https://github.com/schrodingercatss/tuning_playbook_zh_cn

神经网络架构搜索(Neural Architecture Search，NAS)

基于各种优化策略，如随机搜索，贝叶斯优化，遗传算法，

强化学习等等，从搜索空间中自动选择网络结构



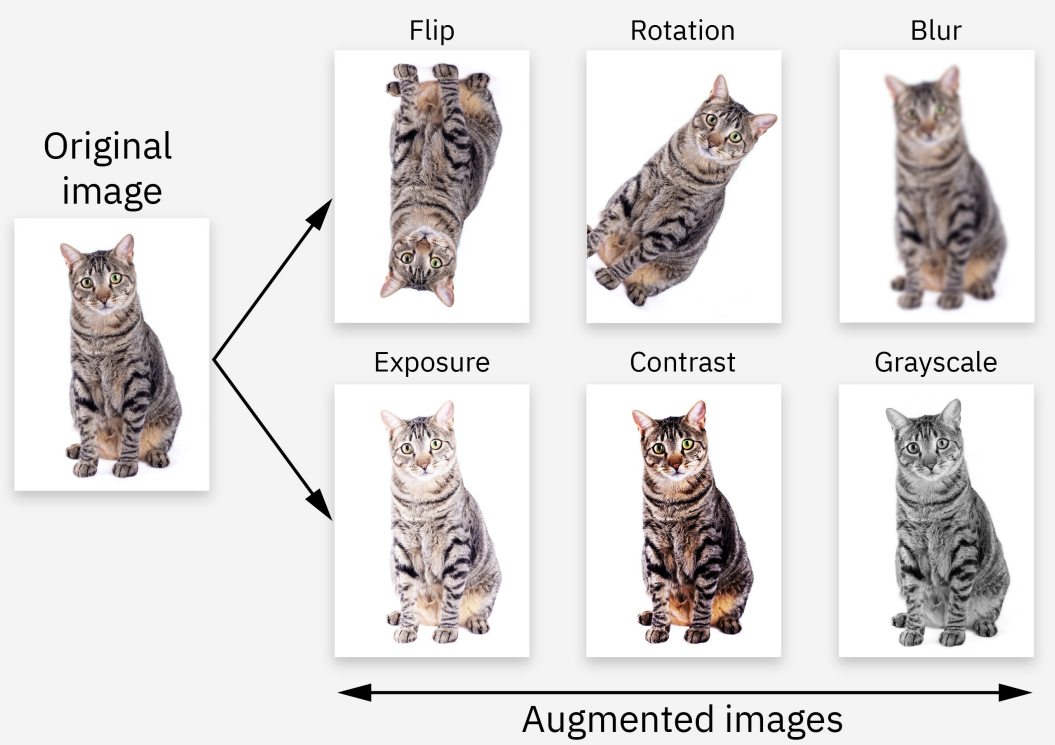
Neural Architecture Search: A Survey. https://arxiv.org/pdf/1808.05377

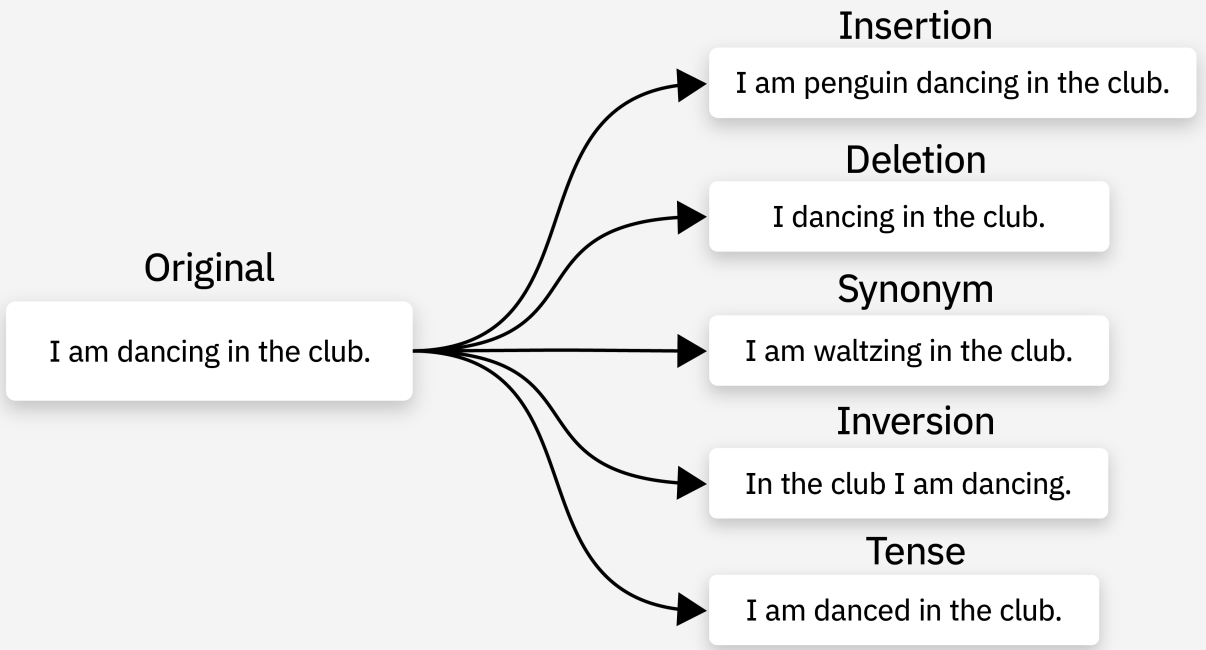
深度学习的挑战

- 复杂优化问题
- 梯度消失与梯度爆炸
- 超参数选择困难
- **过拟合**
- 计算资源开销
- 可解释性
- 稳健性
- …

Some Tricks

数据增强(data augmentation)





Some Tricks

早停(Early Stopping)

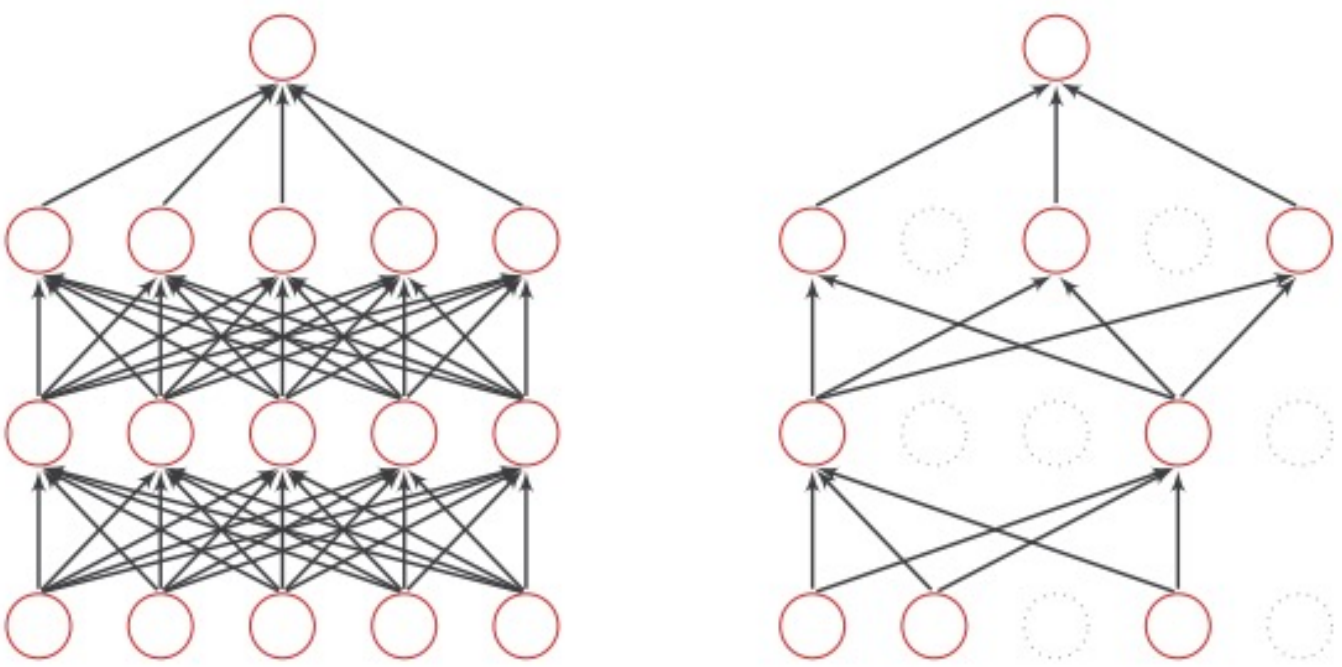
当验证集误差上升时或者
连续多轮优化没有下降时，
提前终止训练

```
        参数:
        val_loss (float): 当前epoch的验证损失
        model (torch.nn.Module): 待保存的模型
        """

        if self.best_loss - val_loss > self.min_delta:
            # 损失有足够的改善
            self.best_loss = val_loss
            self.counter = 0
            # 保存最佳模型
            torch.save(model.state_dict(), self.save_path)
            print(f"Validation loss decreased ({self.best_loss:.6f}). Saving model
        else:
            # 损失没有改善
            self.counter += 1
            print(f"EarlyStopping counter: {self.counter} out of {self.patience}")
            if self.counter >= self.patience:
                print("Early stopping triggered.")
                self.early_stop = True
```

Some Tricks

丢弃法(Dropout)

当训练一个深度神经网络时，我们可以随机丢弃一部分神经元以及其对应的连接边



```
self.layers = nn.Sequential(
    # 第 1 个隐藏层
    nn.Linear(input_size, hidden_size1),
    nn.ReLU(),
    nn.Dropout(p=dropout_p),

    # 第 2 个隐藏层
    nn.Linear(hidden_size1, hidden_size2),
    nn.ReLU(),
    nn.Dropout(p=dropout_p),

    # 输出层
    # (注意: 输出层之前通常不加 Dropout)
    nn.Linear(hidden_size2, output_size)
)
```

Some Tricks

权重衰减(weight decay) / L2正则化

$$L_{reg} = L + \frac{\lambda}{2}||W||^2$$

$$w_{t+1} = (1 - \eta\lambda)w_t - \eta\frac{\partial L}{\partial w_t}$$

```
import torch.optim as optim

# 在这里设置 weight_decay (即 L2 惩罚系数 λ)
optimizer = optim.Adam(model.parameters(),
                       lr=0.001,
                       weight_decay=1e-5)
```

深度学习的挑战

- 复杂优化问题
- 梯度消失与梯度爆炸
- 超参数选择困难
- 过拟合
- **计算资源开销**
- 可解释性
- 稳健性
- …

高性能计算设备

国内的AI芯片

华为昇腾 (Ascend)

寒武纪 (Cambricon)

壁仞科技 (BIREN)

摩尔线程 (Moore Threads)

沐曦集成 (MetaX)

燧原科技 (Enflame)

……

了解寒武纪思元370智能加速卡

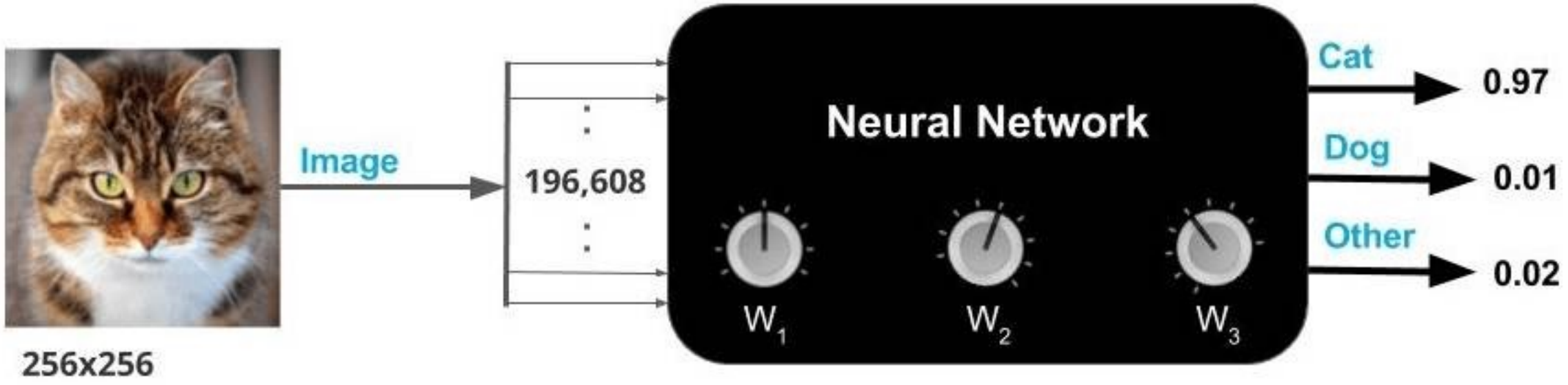
MLU370-S4板卡

MLU370-X4板卡

深度学习的挑战

- 复杂优化问题
- 梯度消失与梯度爆炸
- 超参数选择困难
- 过拟合
- 计算资源开销
- **可解释性**
- 稳健性
- …

深度学习的挑战



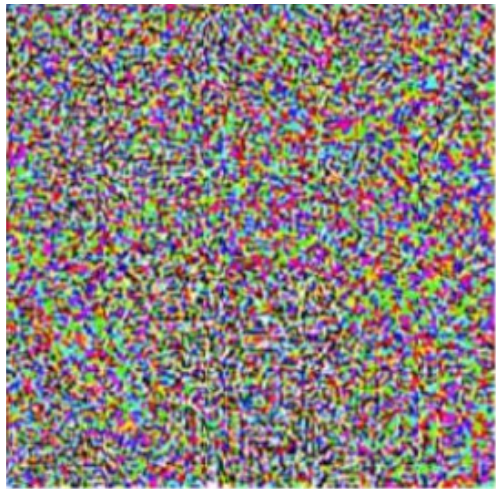
可解释性挑战：模型黑盒，难以得知模型决策逻辑

深度学习的挑战

- 复杂优化问题
- 梯度消失与梯度爆炸
- 超参数选择困难
- 过拟合
- 计算资源开销
- 可解释性
- **稳健性**
- …

对抗攻击(adversarial attack)

+



=

Original Top-3 inferred captions:

1. A red stop sign sitting on the side of a road.
2. A stop sign on the corner of a street.
3. A red stop sign sitting on the side of a street.

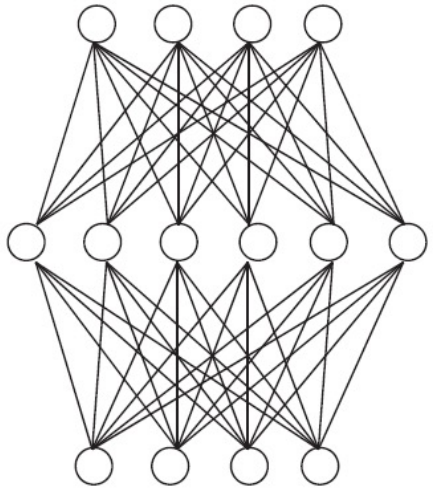
Adversarial Top-3 captions:

1. A brown teddy bear laying on top of a bed.
2. A brown teddy bear sitting on top of a bed.
3. A large brown teddy bear laying on top of a bed.

深度学习模型脆弱，易受攻击

Other Tricks

- Softmax： $P_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$



分析：将输出归一化为概率分布

- 交叉熵(cross-entropy)

- BP算法中以交叉熵 $-\frac{1}{m}\sum_{i=1}^{m} y_i \log \hat{y}_i$ 代替均方误差 $\frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2$

分析：更能体现分类任务的特性

深度学习不仅仅是算法的创新

更是算力、数据共同发展的结果

同时，深度学习仍有许多悬而未决的问题

深度学习是“模拟人脑”吗？

《IEEE 深度对话 Facebook 人工智能负责人 Yann LeCun》

Yann LeCun

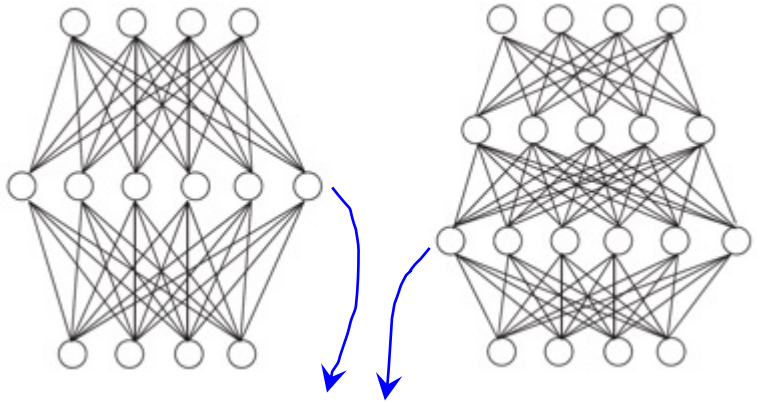
CNN的主要发明人
深度学习“三架马车”之一
2019年图灵奖得主

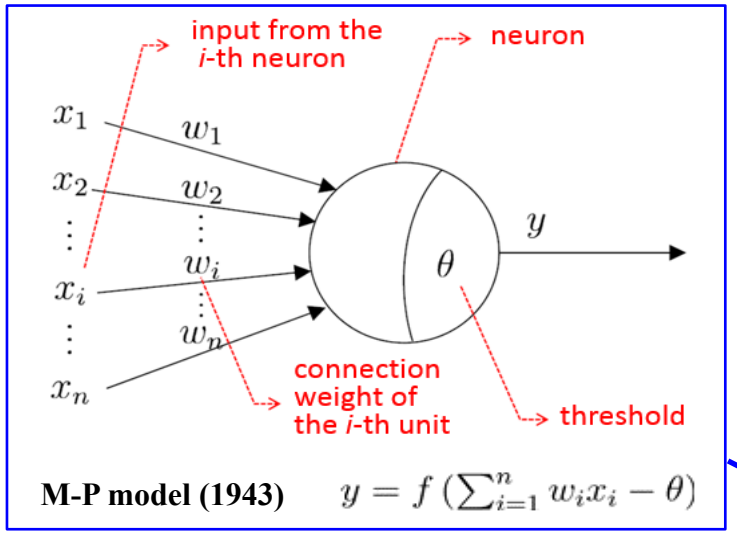
IEEE Spectrum：这些天我们看到了许多关于深度学习的新闻 ……

Yann LeCun：我最不喜欢的描述是「它像大脑一样工作」，我不喜欢人们这样说的原因是，虽然深度学习从生命的生物机理中获得灵感，但它与大脑的实际工作原理差别非常非常巨大。将它与大脑进行类比给它赋予了一些神奇的光环，这种描述是危险的。

深度神经网络

以往神经网络采用单或双隐层结构

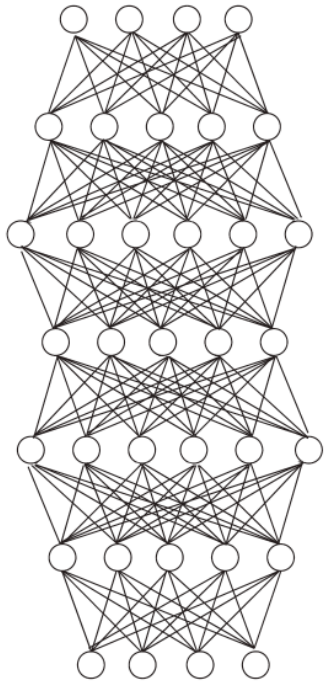




例如，ImageNet 胜者:
2012: 8 层
2015: 152 层
2016: 1207 层

deep

深度神经网络：
很多层



神经网络实质上是多层函数嵌套形成的数学模型

可以说受到了一点生物神经机制的“启发”，但远没有“受指导”

至今最常用的算法：BP [Rumelhart et al., 1986]，是完全从数学上推导出来的

深度神经网络

深度学习并非“突然出现”的

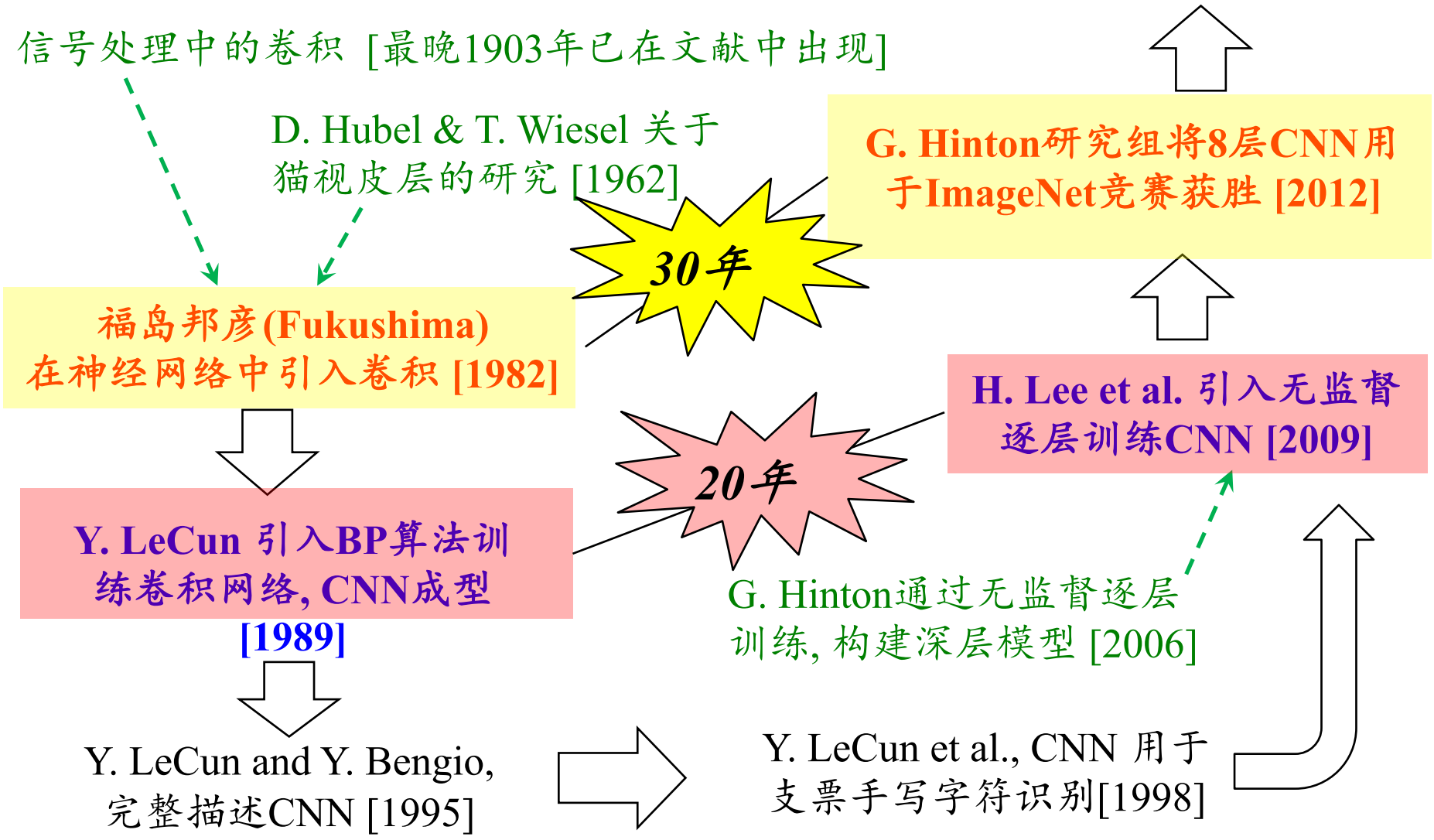
“颠覆性技术”

而是经过了长期发展、

很多研究者做出贡献，

“冷板凳”坐“热”的结果

例如，卷积神经网络


引发深度学习热潮，
被广泛应用
信号处理中的卷积 [最晚1903年已在文献中出现]
D. Hubel & T. Wiesel 关于
猫视皮层的研究 [1962]
30年
G. Hinton研究组将8层CNN用
于ImageNet竞赛获胜 [2012]
福岛邦彦(Fukushima)
在神经网络中引入卷积 [1982]
20年
H. Lee et al. 引入无监督
逐层训练CNN [2009]
Y. LeCun 引入BP算法训
练卷积网络, CNN成型
[1989]
G. Hinton通过无监督逐层
训练, 构建深层模型 [2006]
Y. LeCun and Y. Bengio,
完整描述CNN [1995]
Y. LeCun et al., CNN 用于
支票手写字符识别[1998]

深度学习

科学的发展总是
“螺旋式上升”

三十年河东
三十年河西

坚持才能有结果

2019年3月27日，ACM宣布：

Geoffrey Hinton, Yann LeCun, Yoshua Bengio

因对深度学习的卓越贡献获得图灵奖