深度学习

作　　者:罗裕辉

时　　间:2024 年上

目录

一、 概论

1.1 深度学习路线

如下：

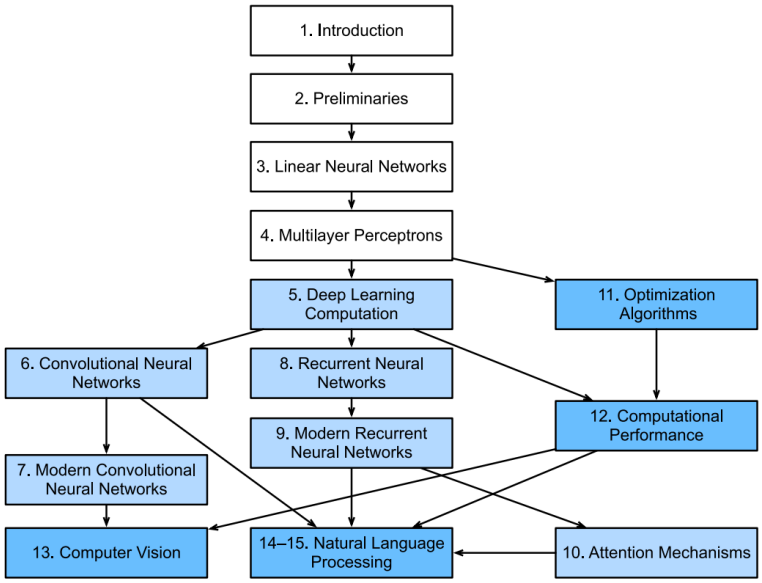


图 **1**

1.2 引言

传统的计算机程序是由软件开发人员从 first principles 开始编写的，这需要设计合适的业务逻辑，涉及应用程序和数据库的交互。

然而，对于某些问题，如下：

- 写一个程序预测明天的天气
- 写一个程序，接受一个问题，然后给出正确答案
- 写一个程序完成人脸识别。
- 写一个程序向用户推荐他们喜欢的产品。

这些问题的可能会随时间变化，内在关系也可能比较复杂，传统的程序不足以解决。这就需要 machine learning，它可以从经验中学习，性能表现不断提高，尤其是 deep learning，在 computer vision，natural language processing，speech recognition 等领域大放异彩。

1.3 机器学习的关键组件

机器学习的过程大概可以分为以下三步：

1. define a function set，or model family

2. define the loss function
3. pick the best funtion

机器学习中也有四个关键组件，包括 data、model、objective funtion、algorithm。

1.3.1 Data

通常来说，每个数据集由一个个样本组成，而样本由一组特征组成。

如果该样本的特征数量固定，我们可以说它是一个长度固定的向量。当然，很多情况下，样本的特征是变化的。

此外，数据的多少，也十分影响 deep learning 的训练效果，通常 deep learning 需要的数据是十分多的，否则未必比得上传统机器学习方法。

1.3.2 Models

深度学习主要关注功能强大的模型，这些模型由神经网络错综复杂地交织在一起，包含层层数据转换。

1.3.3 Objective Functions

我们会需要目标函数，以便评测我们 model 的好坏。并且通常来说，objective function 是越低越好的，所以它也叫 loss function(通常 squared error loss function)。

此外，由于在 training data 上表现好，不一定在 unseen data 上表现好，因此，我们通常还会将原始数据划分成 training dataset 和 test dataset

1.3.4 Optimization Algorithms

还需要有优化算法，它就是寻找到使得 loss function 最小的参数，通常是用 gradient descent 实现，也就是朝着使 loss 下降的方向更新参数。

1.4 机器学习的分类

1.4.1 supervised learning

监督学习就是 data 带标签。

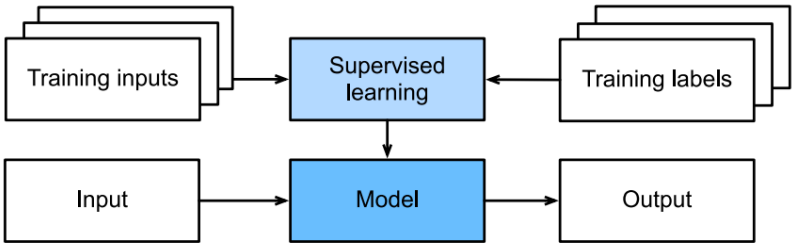


Fig. 1.3.1: Supervised learning.

图 **2**

监督学习包括以下几类：

- regression：针对 how many，loss function 通常是 squared error。
- classification: 针对 which one，并且通常是输出概率。loss function 通常是 cross-entropy。包括二分类、多分类和层次分类。
- tagging: tagging 就是贴标签，而且通常是多个标签，也就是 multi-label classification.
- search: 主要关注的是搜索结果的排序
- recommender systems：推荐系统关注对特定用户的个性化。
- sequence learning：输入序列和输出序列长度不固定，并且输入序列之间可能有关系。
- deep generative models. 能估计数据的密度。

1.4.2 semi-supervised learning

半监督学习就是标签数据远小于无标签数据。通常需要作出一些假设后进行训练。比如低密度分离假设、平滑假设、基于熵的正则化等。

1.4.3 unsupervised and self-supervised learning

无监督学习下数据不带标签，包括以下几类。

- clustering。聚类
- principal component analysis。分析数据的特性，主成分分析。
- generative adversarial networks. 生成对抗网络。

1.4.4 reinforcement learning

强化学习涉及智能体在一系列时间步骤中和环境交互。它会从环境中接收观察，选择一个动作，然后通过某种机制传输回环境，最后从环境中获得奖励，重复这个流程多次。强化学习目标是找到一个好的策略，能够选择好的动作。

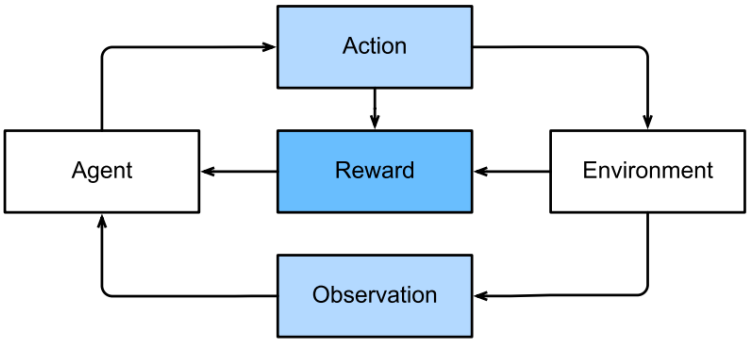


图 **3**

强化学习可以解决很多监督学习不能解决的问题。但是强化学习需要明确哪些动作可以导致好的 reward。

二、 预备知识

学习 deep learning 需要一些前置知识，包括线性代数、微积分、概率论与数理统计和信息论等。这里先复习部分，后面再慢慢补充完善。

2.1 Data manipulation 数据操作

首先我们需要知道如何存储和处理数据。通常我们使用的是 tensor，也就是它有点类似 numpy 的 ndarray(n-dimensional array)，但是 tensor 有几个有点，它可以用 GPU 加速以及支持自动微分等，deep learning 中我们经常使用它。

一个 tensor 就是 n 维数组，一维叫 vector，二维叫 matrix，更高维没有特殊的名字。

1. 0-d:0 维是一个标量，可以看作一个类比。
2. 1-d:1 维是一个向量，可以看作一个特征向量。
3. 2-d:2 维是一个矩阵，可以看作是一个特征矩阵，代表一个样本。
4. 3-d:3 维，例子是 RGB 图片 (宽度 x 高度 x 通道)。
5. 4-d:4 维，一个 RGB 图片批量 (批量大小 x 宽度 x 高度 x 通道)。
6. 5-d:5 维，一个视频批量 (批量大小 x 时间 x 宽度 x 高度 x 通道)。

- tensor 之间的二元运算会变成 tensor 每个元素之间的二元运算，并且即使 size 不匹配，也存在广播机制。
- tensor 可以用索引访问和修改多个 axis，如 X[0:2, :]。
- 为了减少内存开销以及多个位置使用同一个 tensor，我们会想要 in place 操作 tensor，可以使用切片，这样不会分配新的内存。

2.2 Data Preprocessing 数据预处理

实际问题中，我们首先要对原始数据进行预处理，将其转换成 tensor，常用的数据分析的包是 pandas。

2.3 Linear Algebra 线性代数

向量的范数可以理解为是向量的大小，包括 L2 范数和 L1 范数。

2.4 Calculus 微积分

2.4.1 梯度

参考下图 (注意标量只是代表它是一维，可能是多元的)，梯度实际上就是偏导，梯度指向值变化最大的方向，这里是分子布局法。

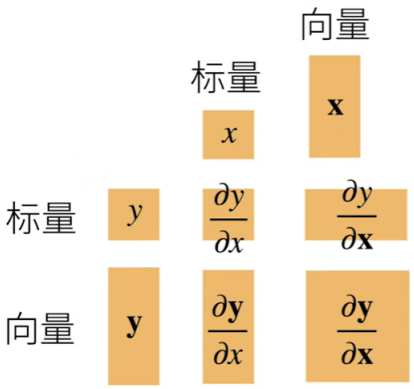


图 **4**

特别的，当 y 是一个向量，x 是一个向量，结果是一个矩阵

$$\partial \mathbf{y}/\partial \mathbf{x} \qquad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}$$

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y_1}{\partial \mathbf{x}} \\ \frac{\partial y_2}{\partial \mathbf{x}} \\ \vdots \\ \frac{\partial y_m}{\partial \mathbf{x}} \end{bmatrix} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1}, \frac{\partial y_1}{\partial x_2}, \ldots, \frac{\partial y_1}{\partial x_n} \\ \frac{\partial y_2}{\partial x_1}, \frac{\partial y_2}{\partial x_2}, \ldots, \frac{\partial y_2}{\partial x_n} \\ \vdots \\ \frac{\partial y_m}{\partial x_1}, \frac{\partial y_m}{\partial x_2}, \ldots, \frac{\partial y_m}{\partial x_n} \end{bmatrix}$$

图 **5**

2.4.2 自动求导

自动求导计算一个函数在指定值上的导数，可以通过计算图理解。计算图将代码分解成操作子，将计算表示成一个无环图

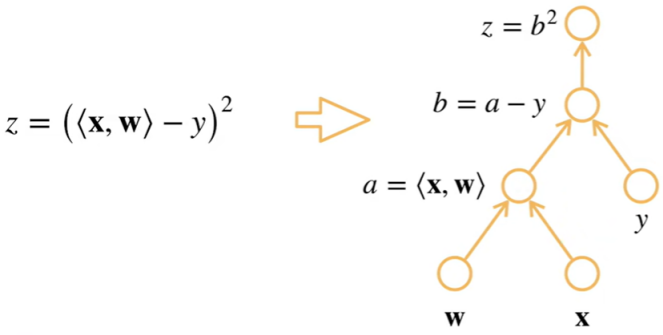


图 **6**

根据链式法则，计算梯度有两种方向：

自动求导的两种模式

- 链式法则：$\frac{\partial y}{\partial x}=\frac{\partial y}{\partial u_n}\frac{\partial u_n}{\partial u_{n-1}}\cdots\frac{\partial u_2}{\partial u_1}\frac{\partial u_1}{\partial x}$
- 正向累积 $\frac{\partial y}{\partial x}=\frac{\partial y}{\partial u_n}\left(\frac{\partial u_n}{\partial u_{n-1}}\left(\cdots\left(\frac{\partial u_2}{\partial u_1}\frac{\partial u_1}{\partial x}\right)\right)\right)$
- 反向累积、又称反向传递

$$\frac{\partial y}{\partial x}=\left(\left(\left(\frac{\partial y}{\partial u_n}\frac{\partial u_n}{\partial u_{n-1}}\right)\cdots\right)\frac{\partial u_2}{\partial u_1}\right)\frac{\partial u_1}{\partial x}$$

图 **7**

反向累积

$z=\left(\langle \mathbf{x},\mathbf{w}\rangle-y\right)^2$

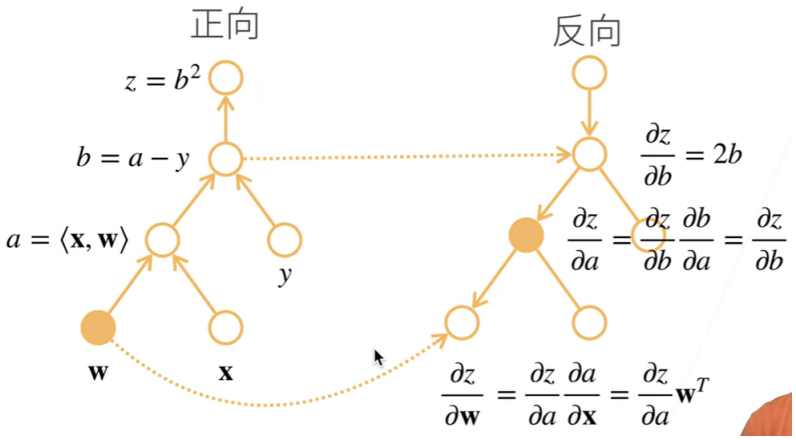


图 **8**

通常我们使用反向累积更新参数，虽然它需要计算图的中间结果，空间复杂度 O(n)，但是更新所有参数总体时间复杂度为 O(n)，牺牲空间换时间。

2.5 Probability and Mathematical Statistics 概率论与数理统计

2.6 Information theory 信息论

三、 线性回归

线性回归一般是深度学习遇到的第一类问题了，我们首先了解它，下面以购房为例进行介绍。

3.1 问题介绍

假设需要买一个房子，我们主要关心的是房子的价格，我们希望得到一个精准的房子预估价，而房子的价格与多种因素有关，比如卧室个数，卫生间个数和居住面积等，假设房子的价格与这些因素成线性关系，我们要训练一个模型，给定相关因素，能输出准确的预估价。

3.2 模型建立

1. 输入：给定与房子有关的 n 个因素，即 $\mathbf{x} = [x_1, x_2, ..., x_n]^T$
2. 权重和偏差：n 个因素有 n 个权重，即 $\mathbf{w} = [w_1, w_2, ..., w_n]^T$，此外，还存在一个标量偏差 b。
3. 输出：模型输出是预估价，它是输入的加权和。$y = w_1x_1 + w_2x_2 + ... + w_nx_n + b$，即 $y =< \mathbf{x},\mathbf{w} > +b$。

考虑 m 个样本集合 $\boldsymbol{X}_{\boldsymbol{m},\boldsymbol{n}}$，其每行代表一个样本的 n 个特征，则模型表达式为:

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{w} + b$$

事实上，线性模型可以看作是单层的神经网络 (有权重的层只有输入层)，它只有输入层和输出层，没有隐藏层。

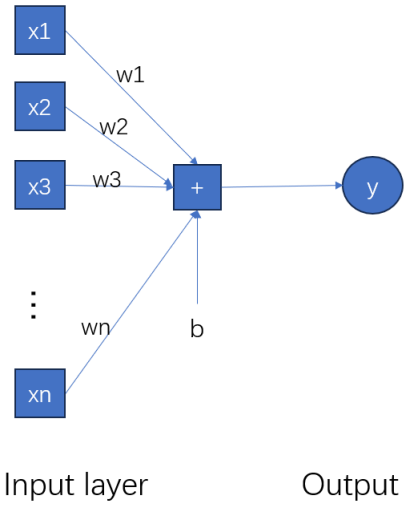


图 **9**

3.3 评估函数

我们的目标是根据已有的数据训练出模型，尽可能是模型的输出预估价 $\hat{y}$ 接近真实的购入价 y，可以定义损失函数 (**均方误差**)，n 为样本数 (小批量梯度下降则为批量大小)：

$$l(y,\hat{y}) = \frac{1}{2n}\sum(y-\hat{y})^2$$

3.4 模型训练

我们首先需要收集过往的数据，它们包括各种因素以及最终的成交价，通常越多越好，假设有 n 个样本，即

$$\mathbf{X} = [\boldsymbol{x_1}, \boldsymbol{x_2}, ..., \boldsymbol{x_n}]^T$$

$$\mathbf{y} = [y_1, y_2, ..., y_n]^T$$

其中，$\boldsymbol{X}$ 中的每个 $\boldsymbol{x_i}$ 经过转置后都是一个行向量，对应一个样本的不同因素，$\boldsymbol{y}$ 中同样 y_i 对应不同样本最后的成交价格。

我们训练的方法是利用训练数据最小化 loss function，并找到对应的参数 $\boldsymbol{w}$ 和 $\boldsymbol{b}$ 即：

$$\boldsymbol{w*}, \boldsymbol{b*} = arg\min_{w,b} l(y,\hat{y}) = \frac{1}{2n}\sum_{i=1}^{n}(y_i - < \boldsymbol{x_i}, , \boldsymbol{w_i} > -b)^2$$

3.4.1 显式解——由凸函数性质直接给出最优解

由于我们的 loss function 是一个凸函数 (简单的说，就是两个点上的函数值连线的位置总是位于该函数图像的或之上)，凸函数的局部最优就是全局最优，根据数学原理，

我们知道当梯度为 0 时取最小值。

为了方便，将 b 纳入 w, 同时 X 加一列全 1，即 $\boldsymbol{X} \leftarrow [\boldsymbol{X}, \boldsymbol{1}], w \leftarrow \begin{bmatrix} w \\ b \end{bmatrix}$。

此时 $l = \frac{1}{2n}||\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}||^2$，其中 $||\ ||$ 代表 L2 范数，于是

$$\frac{\partial l}{\partial \boldsymbol{w}} = -\frac{1}{n}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w})^T\boldsymbol{X} = 0$$

$$\boldsymbol{w*} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}$$

3.4.2 梯度下降

然而，通常来说深度学习的处理的问题不是凸问题，属于 NP-hard 问题，一般采用梯度下降法求解。

1. 选择一个参数的初始值 $\boldsymbol{w_0}$。
2. 选择一个学习率 η，**不能过大也不能过小**。
3. 重复迭代一定次数 $t = 1, 2, 3, ...$，沿着梯度方向的反方向更新参数。即：

$$\boldsymbol{w_t} = \boldsymbol{w_{t-1}} - \eta\frac{\partial l}{\partial \boldsymbol{w_{t-1}}}$$

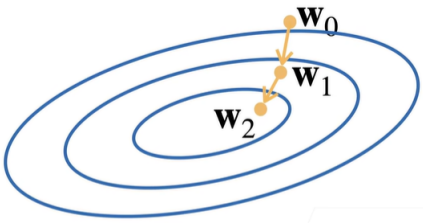


图 **10**

当然，考虑到训练数据一般很大，在整个训练集上计算梯度耗时太长，通常我们使用小批量梯度下降 (mini-batch)。

也就是随机从整个样本中采样 b 个样本，**不能过大也不能过小**，$I_b = i_1, i_2, ..., i_b$，计算这 b 个样本的损失，近似为整个样本的损失，即

$$l = \frac{1}{b}\sum_{I_b} l(y_i, \hat{y_i}) = \frac{1}{2b}\sum_{I_b}(y_i - < \boldsymbol{x_i}, , \boldsymbol{w_i} > -b)^2$$

3.5 代码实现

3.5.1 不使用框架

3.5.2 使用框架

四、 softmax 回归

softmax 回归虽然叫做回归，但实际上是处理分类问题。

- 回归预测的是一个连续值。
- 分类预测的是一个离散类比。

4.1 问题介绍

假设我们要对图像进行分类，输出它属于哪个类别。

- 假设输入是 2x2 的灰度图像，那么就有 x_1, x_2, x_3, x_4 四个 feature。
- 假设有 3 个类别，分类问题通常和类之间的 natural order 无关，我们使用 one-hot encoding 对类别编码，第 i 位是 1，其他位是 0 代表第 i 类。

4.2 模型建立

我们需要输出属于哪个类别，每个类别都有可能，需要计算属于每个类别的概率，假设每个类别和图像所有像素值之间依然是线性关系，每个输出都有 $d+1$ 个参数，则有:

$$
\begin{aligned}
o_1 &= x_1 w_{11} + x_2 w_{12} + x_3 w_{13} + x_4 w_{14} + b_1, \\
o_2 &= x_1 w_{21} + x_2 w_{22} + x_3 w_{23} + x_4 w_{24} + b_2, \\
o_3 &= x_1 w_{31} + x_2 w_{32} + x_3 w_{33} + x_4 w_{34} + b_3.
\end{aligned}
$$

即 $\boldsymbol{o} = \boldsymbol{x}^{\boldsymbol{T}}\boldsymbol{W} + \boldsymbol{b}$，其中，$\boldsymbol{x_{1,d}} = [x_1, x_2, ..., x_d]^T$，$\boldsymbol{W_{d,q}}$ 代表每个图像有 d 个特征，有 q 个类别。

这依然可以视为一个单层的线性神经网络。

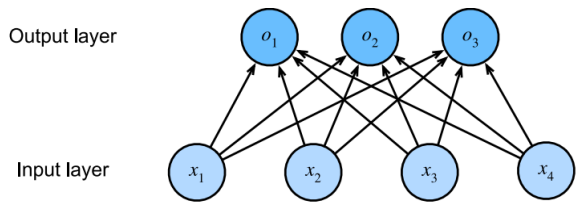


图 **11**

我们希望能够给出每个类别的置信度，更准确说是概率，那么就需要满足概率的几条基本公理，如非负且和为 1。一个好的办法是在输出层 logits **o** 后面接上一个 softmax 函数，即：

$$\hat{\boldsymbol{y}} = \text{softmax}(\boldsymbol{o}).$$

$$\hat{y}_j = \frac{\exp(o_j)}{\sum_k \exp(o_k)}.$$

输出概率最大对应的类别，即：

$$\arg\max_j \hat{y}_j$$

汇总，如果 batch_size 的大小是 n，那么模型如下：

$$\boldsymbol{O} = \boldsymbol{X}\boldsymbol{W} + \boldsymbol{b},$$

$$\hat{\boldsymbol{Y}} = \text{softmax}(\boldsymbol{O}).$$

$$\arg\max_j \hat{y}_j$$

其中，$\boldsymbol{X} \in \mathbb{R}^{n \times d}$，$\boldsymbol{W} \in \mathbb{R}^{d \times q}$，$\boldsymbol{b} \in \mathbb{R}^{1 \times q}$。而 softmax 是对 $\boldsymbol{O}$ 的每一行也就是对应每个样本进行 softmax。

4.3 损失函数

对于分类问题，我们需要衡量的是真实值和预测值之间的分布差距，通常使用交叉熵 (KL 散度或者说相对熵去掉了常数项) 衡量，即损失函数为:

$$l(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{j=1}^{q} y_j \log \hat{y}_j.$$

由于通常真实值中只有真实 label 为 1，其余为 0，所以损失函数实际上是分配给真实标签的预测概率的负对数似然值，即：

$$l(\mathbf{y}, \hat{\mathbf{y}}) = -\log \hat{y}_j.$$

4.4 代码实现

4.4.1 简洁实现

由于 softmax 需要取指数，传入 softmax 的某个 o_j 可能非常大导致溢出，因此将每个 $o_j - max(o_k)$，此外，o_j 也可能很小，导致后面 cross-entropy 取对数时负无穷，因此一种好的方法是将 softmax 和 crsoo-entropy 结合，先得出公式再代入计算，即：

$$\begin{aligned}\hat{y}_j &= \frac{\exp(o_j - \max(o_k))\exp(\max(o_k))}{\sum_k \exp(o_k - \max(o_k))\exp(\max(o_k))} \\ &= \frac{\exp(o_j - \max(o_k))}{\sum_k \exp(o_k - \max(o_k))}.\end{aligned}$$

$$\begin{aligned}\log(\hat{y}_j) &= \log\left(\frac{\exp(o_j - \max(o_k))}{\sum_k \exp(o_k - \max(o_k))}\right) \\ &= \log(\exp(o_j - \max(o_k))) - \log\left(\sum_k \exp(o_k - \max(o_k))\right) \\ &= o_j - \max(o_k) - \log\left(\sum_k \exp(o_k - \max(o_k))\right).\end{aligned}$$

五、 多层感知机

5.1 感知机历史

1957 年感知机被提出，它根据通过线性决策边界将数据分为两类，将线性组合的结果传递给激活函数，通常是一个阶跃函数，其形式如下：

$$\text{activation}(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$

其中 $z = w_1x_1 + w_2x_2 + \cdots + w_nx_n + b$。

然而，它不能拟合 XOR 等非线性复杂函数，导致了第一次 AI 寒冬。

后来，通过引入非线性激活函数和多隐藏层，可以解决非线性的复杂问题。

5.2 多层感知机

以单隐藏层多层感知机为例，假设有 n 个样本，输入为 **x**，有 d 个特征，输出为 **O**，有 q 类，隐藏层为一层 h 个隐藏单元，每个隐藏层还有一个非线性激活函数，隐藏层输出为 **H**(如果没有在隐藏层引入非线性激活函数，那么这样多层的架构相当于感知机，

依然是线性的)，即 (当然后面还要跟一个 softmax)：

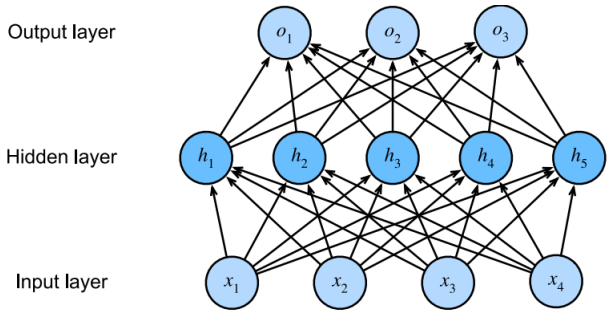


Fig. 4.1.1: An MLP with a hidden layer of 5 hidden units.

图 **12**

$$\mathbf{H} = \sigma(\mathbf{X}\mathbf{W}^{(1)} + \mathbf{b}^{(1)}),$$

$$\mathbf{O} = \mathbf{H}\mathbf{W}^{(2)} + \mathbf{b}^{(2)}.$$

其中，$\mathbf{X} \in \mathbb{R}^{n\times d}$，$\mathbf{W}^{(1)} \in \mathbb{R}^{d\times h}$，$\mathbf{H} \in \mathbb{R}^{n\times h}$, $\mathbf{W}^{(2)} \in \mathbb{R}^{h\times q}$，$\mathbf{O} \in \mathbb{R}^{n\times q}$。$\sigma$ 为非线性激活函数，按元素操作，其输出称为活性值。

把前 L−1 层看作表示，把最后一层看作线性预测器，这种架构通常称为多层感知机。

这里超参数多了隐藏层层数和每层大小，其选取有一定经验依据。通常深些 (尽管一个单隐层网络能学习任何函数) 并且宽度先递增再递减 (扩张 + 压缩) 或者直接递减 (压缩)。

5.3 激活函数

隐藏层输入除了经过仿射函数外还要经过激活函数，激活函数通过计算加权和并加上偏置来确定神经元是否应该被激活，通常是非线性的。

5.3.1 ReLU

在最早的神经网络中，科学家们感兴趣的是对“激发”或“不激发”的生物神经元进行建模。

修正线性单元 (Rectified linear unit，ReLU) 是一个简单好求导高效的激活函数，它将相应的活性值设为 0，仅保留正元素并丢弃所有负元素，其表达式如下：

$$\mathrm{ReLU}(x) = \max(x, 0).$$

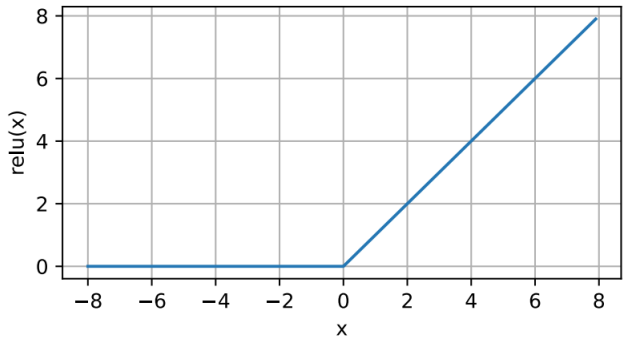


图 **13**

5.3.2 sigmoid 函数

挤压函数（squashing function），它将范围（-inf, inf）中的任意输入压缩到区间（0, 1）中的某个值:

$$\text{sigmoid}(x) = \frac{1}{1 + \exp(-x)}.$$

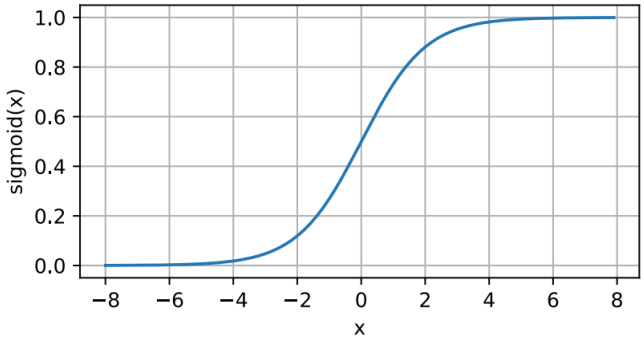


图 **14**

以前基于梯度的学习时，sigmoid 函数被使用，尤其是二分类问题中，它可以看作 softmax 的特例。但存在梯度消失问题，现在隐藏层中已经被 ReLu 取代。

5.3.3 tanh 函数

tanh(双曲正切) 函数将其输入压缩转换到区间 (-1, 1) 上，公式如下:

$$\tanh(x) = \frac{1 - \exp(-2x)}{1 + \exp(-2x)}.$$

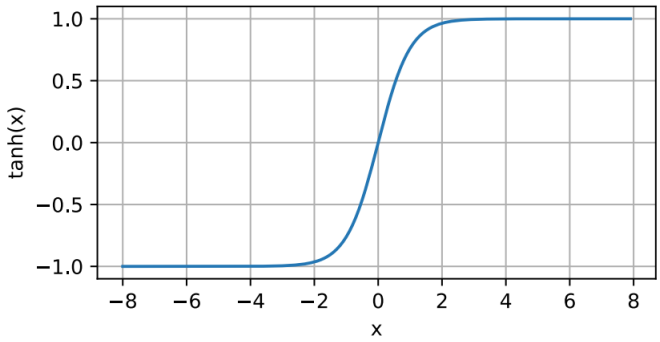


图 **15**

5.4 logistic、sgmoid 和 softmax

这三者有点类似，我们区分一下它们。

- logistic 函数。
  用于二分类的函数 (softmax 的二分类版本)，表达式如下：

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

  其中，$z = w_n x_n + w_{n-1} x_{n-1} + ... + w_1 x_1 + b$。
- sgmoid 函数
  神经网络中的激活函数，也是 logistic 函数，表达式如下 (这里 z 也经过了仿射变换)：

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

- softmax 函数
  常用于多分类问题的输出层，将一个向量的所有元素转换为概率分布，公式如下：

$$\text{Softmax}(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

5.5 代码实现

六、 相关知识

我们希望模型是发现模式而不是简单地记住样本。而如果模型在训练数据集上表现比测试数据集上要好则称为过拟合，需要用正则化方法避免。

6.1 模型选择

6.1.1 训练集、验证集和测试集

我们需要对不同模型 (模型类型不同或者超参数不同) 进行评估选择最佳模型，因此通常会引入验证集。

- 训练集：用于训练模型参数的数据集。
- 验证集: 用于选择模型超参数的数据集 (与测试集的边界被模糊，测试集应该最终用来评估，不能用来调优，因此通常我们其实是在用验证集)。
- 测试集: 用于评估模型最终性能的数据集。

6.1.2 K 折交叉验证

如果将训练集的 50% 作为验证集可能会导致数据稀缺问题，非大数据集通常使用 K 折 (K 取 5 或 10) 交叉验证。

将原始训练数据分成 K 个不重叠的子集。每次在 K − 1 个子集上进行训练，并在剩余的一个子集（在该轮中没有用于训练的子集）上进行验证。最后，通过对 K 次实验的结果取平均来估计训练和验证误差。

6.2 欠拟合和过拟合

- 欠拟合: 模型在训练集上表现不佳，模型过于简单，不能很好地拟合训练数据。
- 过拟合: 模型在训练集上表现得非常好，但在验证集或测试集上表现不佳。模型过于复杂，记住了训练数据中的噪声。

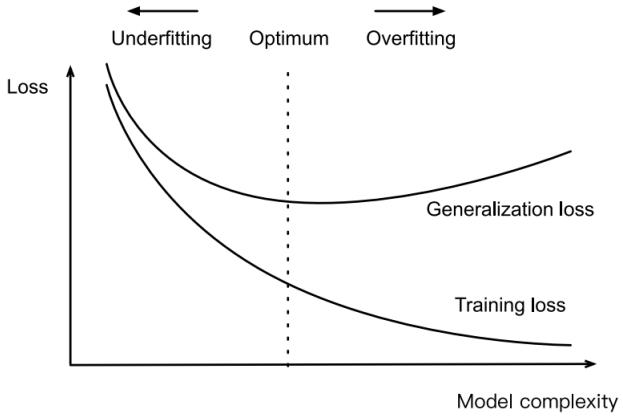


Fig. 4.4.1: Influence of model complexity on underfitting and overfitting

图 **16**

深度模型本质上就是要针对数据集选择复杂度合适的模型。

6.3 正则化技术-权重衰减

6.3.1 范数

范数可以看作是衡量向量的大小，L_p 范数的定义如下：

$$\|\mathbf{x}\|_p = \left(\sum_{i=1}^{n} |x_i|^p\right)^{1/p}.$$

- 当 $p = 1$ 时，为 L_1 范数:

$$\|\mathbf{x}\|_1 = \sum_{i=1}^{n} |x_i|.$$

- 当 $p = 2$ 时，为 L_2 范数，深度学习常用 L_2 范数的平方，下标 2 也常省略:

$$\|\mathbf{x}\|_2 = \sqrt{\sum_{i=1}^{n} x_i^2},$$

6.3.2 权重衰减

权重衰减（weight decay）是最广泛使用的正则化的技术之一，它通常也被称为 L_2 正则化，是一项缓解过拟合的常用手段。

它调整损失为为最小化预测损失和惩罚项之和：

$$\min \ell(\mathbf{w}, b) + \frac{\lambda}{2}\|\mathbf{w}\|^2$$

其中，λ 为正则化参数，越大对权重的限制越强，使权重越小。

梯度下降则为:

$$\mathbf{w}_{t+1} = (1 - \eta\lambda)\mathbf{w}_t - \eta\frac{\partial \ell(\mathbf{w}_t, b_t)}{\partial \mathbf{w}_t}$$

通常，$\eta\lambda < 1$，这也是为什么叫权重衰减，这也使得模型参数不会过大，控制模型复杂度，进而缓解过拟合。

6.4 正则化技术-丢弃法 dropout

一个好的模型需要对输入数据的扰动鲁棒。

相关数学知识证明，在输入中加入随机噪声等价于 Tikhonov 正则化。

而丢弃法 dropout 则是对隐藏层的输出每次训练前向传播时以一定概率丢弃 (输出

为 0)，但保持期望不变，即：

$$h' = \begin{cases} 0 & \text{with probability } p \\ \frac{h}{1-p} & \text{otherwise} \end{cases}$$

dropout 只在训练时使用，测试时不使用，且主要用于全连接层，p 一般取 0.1，0.5 或 0.9。

6.5 正则化技术-数据增强

6.6 数值稳定性

6.6.1 梯度公式推导

考虑一个具有 d 层的神经网络，第 t 层的输出为 $\mathbf{h}^t(\mathbf{h}^0 = \mathbf{x})$，损失函数为 l，则函数变换如下：

$$\mathbf{h}^t = f_t(\mathbf{h}^{t-1})$$

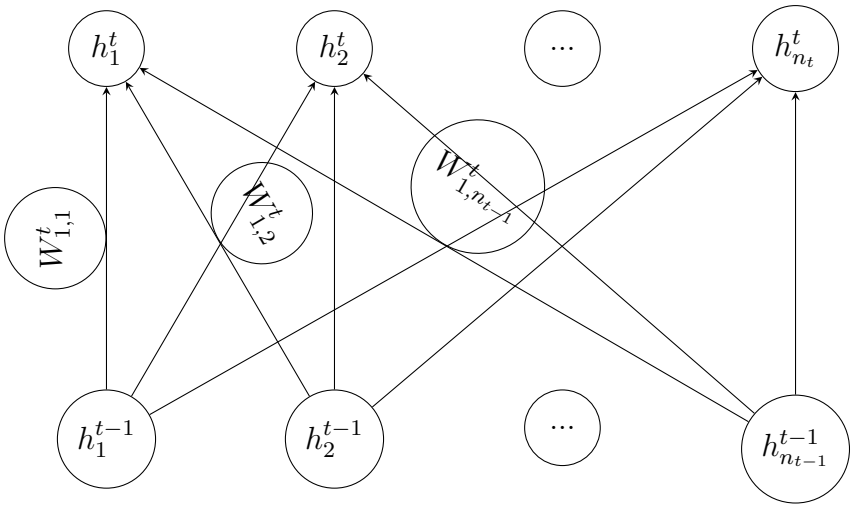
$$\mathbf{y} = \ell \circ f_d \circ \cdots \circ f_1(\mathbf{x})$$

对于第 t 层，损失关于参数 $\mathbf{W}_t$ 的梯度计算如下:

$$\frac{\partial \ell}{\partial \mathbf{W}_t} = \frac{\partial \ell}{\partial \mathbf{h}^d} \frac{\partial \mathbf{h}^d}{\partial \mathbf{h}^{d-1}} \cdots \frac{\partial \mathbf{h}^{t+1}}{\partial \mathbf{h}^t} \frac{\partial \mathbf{h}^t}{\partial \mathbf{W}_t}$$

中间是向量对于向量的导数，结果是一个矩阵，因此中间实际上是 $d-t$ 次矩阵乘法运算。

我们进一步具体分析其求导结果 (这里省去了偏移)，首先给出图示方便理解 (假设第 $t-1$ 层有 n_{t-1} 个神经元，第 t 层有 n_t 个神经元)：



$$\mathbf{W}^t = \begin{pmatrix} W_{1,1}^t & W_{1,2}^t & \cdots & W_{1,n_{t-1}}^t \\ W_{2,1}^t & W_{2,2}^t & \cdots & W_{2,n_{t-1}}^t \\ \vdots & \vdots & \ddots & \vdots \\ W_{n_t,1}^t & W_{n_t,2}^t & \cdots & W_{n_t,n_{t-1}}^t \end{pmatrix} \quad \mathbf{h}^{t-1} = \begin{pmatrix} h_1^{t-1} \\ h_2^{t-1} \\ \vdots \\ h_{n_{t-1}}^{t-1} \end{pmatrix}$$

$$\mathbf{h}^t = f_t(\mathbf{h}^{t-1}) = \sigma(\mathbf{W}^t\mathbf{h}^{t-1}) \quad \sigma\text{是激活函数}$$

$$\text{不妨记}\mathbf{z^t} = \mathbf{W}^t\mathbf{h}^{t-1}, \mathbf{z}^t = \begin{pmatrix} z_1^t \\ z_2^t \\ \vdots \\ z_{n_t}^t \end{pmatrix}$$

$$\text{根据链式法则,} \frac{\partial \mathbf{h}^t}{\partial \mathbf{h}^{t-1}} = \frac{\partial \mathbf{h}^t}{\partial \mathbf{z}^t} \cdot \frac{\partial \mathbf{z}^t}{\partial \mathbf{h}^{t-1}}$$

进一步根据向量对向量的求导结果得：$\frac{\partial \mathbf{h}^t}{\partial \mathbf{h}^{t-1}} = \text{diag}\left(\sigma'(\mathbf{W}^t\mathbf{h}^{t-1})\right)(\mathbf{W}^t)$ ，σ'是σ 的导数函数，diag 是结果写成对角矩阵。

因此，中间 $d-t$ 次乘法运算结果为：

$$\frac{\partial \mathbf{h}^d}{\partial \mathbf{h}^{d-1}} \cdots \frac{\partial \mathbf{h}^{t+1}}{\partial \mathbf{h}^t} = \prod_{i=t}^{d-1} \frac{\partial \mathbf{h}^{i+1}}{\partial \mathbf{h}^i} = \prod_{i=t}^{d-1} \text{diag}\left(\sigma'(\mathbf{W}^i\mathbf{h}^{i-1})\right)(\mathbf{W}^i)$$

注意：这里必须使用链式法则借助变量 $\mathbf{z^t}$，因为 $\mathbf{h}^t = \sigma(\mathbf{z}^t)$ 只是按元素作用，并不是严格的函数关系。

6.6.2 梯度爆炸

按照上面的结果，如果权重比较大且层数比较深就会存在梯度爆炸，可能使得：

1. 值超出值域
2. 学习率敏感，不好调整

6.6.3 梯度消失

梯度消失主要是 sigmoid 激活函数的缺点，其函数和导数如下：

$$\sigma(x) = \frac{1}{1+e^{-x}} \quad \sigma'(x) = \sigma(x)(1-\sigma(x))$$

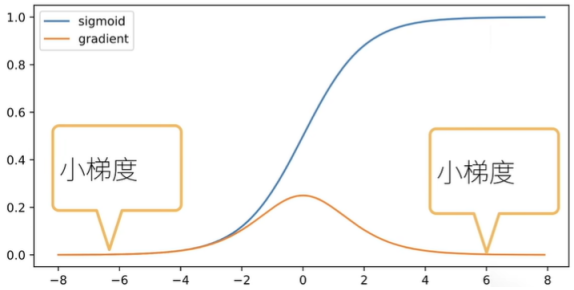


图 **17**

输入值离 0 越远，梯度越接近 0，并且乘积导致梯度消失。

1. 底层梯度变成 0。
2. 不管如何选择学习率，训练没有进展。

6.6.4 训练稳定方法

使训练稳定通常有以下方法：

- 将乘法变加法：ResNet，LSTM
- 归一化: 梯度归一化，梯度裁剪
- 合理的权重初始化和激活函数

这里我们首先研究**合理的权重初始化和激活函数**。

对于中小神经网络，直接正态分布初始化权重 $N(0, 0.01)$ 是有效的，但是对于更大的神经网络需要更合理的权重初始化。

将每层的输出和权重看作随机变量，我们的目标是让它们的均值和方差保持一致，且均值通常取 0，即 $\forall i, t$：

$$\text{正向} \quad \mathbb{E}[h_i^t] = 0 \quad \text{Var}[h_i^t] = a$$

$$\text{反向} \quad \mathbb{E}\left[\frac{\partial \ell}{\partial h_i^t}\right] = 0 \quad \text{Var}\left[\frac{\partial \ell}{\partial h_i^t}\right] = b$$

公式推导

假设第 t 层权重 w_{ij} 均值为 0，方差为 σ^2，输入 h_i^t 的均值为 0，方差为 γ^2，忽略激活函数，则有：

- 正向：

$$h_i^t = \sum_j W_{ij}^t h_j^{t-1}$$

$$\mathbb{E}[h_i^t] = \sum_j \mathbb{E}[W_{ij}^t]\mathbb{E}[h_j^{t-1}] = 0$$

$$\begin{aligned}
\text{Var}[h_i^t] &= \mathbb{E}[(h_i^t)^2] - \mathbb{E}[h_i^t]^2 \\
&= \mathbb{E}\left[\sum_j (W_{ij}^t h_j^{t-1})^2\right] \\
&= \sum_j \mathbb{E}[(W_{ij}^t)^2]\mathbb{E}[(h_j^{t-1})^2] \\
&= n_{t-1}\sigma^2 \text{Var}[h_j^{t-1}]
\end{aligned}$$

$$令\text{Var}[h_i^t] = \text{Var}[h_j^{t-1}]$$

$$则 n_{t-1}\sigma^2 = 1$$

- 反向：

$$h_i^t = \sum_j W_{ij}^t h_j^{t-1},$$

$$\begin{aligned}
\frac{\partial \ell}{\partial h_j^{t-1}} &= \sum_i \frac{\partial \ell}{\partial h_i^t}\frac{\partial h_i^t}{\partial h_j^{t-1}} \\
&= \sum_i \frac{\partial \ell}{\partial h_i^t} W_{ij}^t,
\end{aligned}$$

$$\begin{aligned}\mathbb{E}\left[\frac{\partial \ell}{\partial h_j^{t-1}}\right] &= \sum_i \mathbb{E}\left[\frac{\partial \ell}{\partial h_i^t}\right] \mathbb{E}\left[W_{ij}^t\right] \\ &= 0\end{aligned}$$

$$\begin{aligned}\text{Var}\left[\frac{\partial \ell}{\partial h_j^{t-1}}\right] &= \mathbb{E}\left[\left(\frac{\partial \ell}{\partial h_j^{t-1}}\right)^2\right] - \mathbb{E}\left[\frac{\partial \ell}{\partial h_j^{t-1}}\right]^2 \\ &= \mathbb{E}\left[\left(\sum_i \frac{\partial \ell}{\partial h_i^t} W_{ij}^t\right)^2\right] \\ &= \sum_i \mathbb{E}\left[\left(\frac{\partial \ell}{\partial h_i^t}\right)^2\right] \mathbb{E}\left[\left(W_{ij}^t\right)^2\right] \\ &= n_t \sigma^2 \text{Var}\left[\frac{\partial \ell}{\partial h_i^t}\right]\end{aligned}$$

$$\begin{aligned}\text{令Var}\left[\frac{\partial \ell}{\partial h_j^{t-1}}\right] &= \text{Var}\left[\frac{\partial \ell}{\partial h_i^t}\right] \\ \text{则} n_t \sigma^2 &= 1.\end{aligned}$$

按照这个要求，我们需要满足：

$$\begin{cases} n_{t-1}\sigma^2 &= 1 \\ n_t \sigma^2 &= 1 \end{cases} \tag{1}$$

这很难同时满足，$Xavier$ 提出让 $\frac{n_{t-1}+n_t}{2}\sigma^2 = 1$，则 $\sigma = \sqrt{\frac{2}{n_{t-1}+n_t}}$，而实现这一点有两种方法 (不一定要正态分布)：

- 初始化权重为正态分布 $N(0, \sqrt{\frac{2}{n_{t-1}+n_t}})$
- 初始化权重为均匀分布 $U(-\sqrt{\frac{6}{n_{t-1}+n_t}}, \sqrt{\frac{6}{n_{t-1}+n_t}})$

此外同理可考虑激活函数对均值和方差的影响，也因此发现 Relu、tanh 优于 sigmoid，如果要用 sigmoid，可以对其进行变换，改成 $4\text{sigmoid} - 2$

七、 卷积神经网络

7.0.1 从全连接层到卷积

MLP 十分适合处理表格数据，每行可以看作一个样本，每列可以看作特征，尽管它也能用于分类图片，将图片平铺，但这忽视了图像的空间结构信息。

图像的识别具有以下特征：

- 平移不变性（translation invariance）：不管检测对象出现在图像中的哪个位置，我们都能识别它，即神经网络的前面几层应该对相同的图像区域具有相似的反应，即为“平移不变性”。
- 局部性（locality）：图像的识别只需看局部区域，即神经网络的前面几层应该只探索输入图像中的局部区域，而不过度在意图像中相隔较远区域的关系，这就是“局部性”原则。最终，可以聚合这些局部特征，以在整个图像级别进行预测。

前面我们已经知道，当输入是一维时，隐藏表示 **H** 与输入 **X** 的关系如下：

$$\mathbf{H}_i = \sum_j \mathbf{W}_{i,j}\mathbf{X}_j + \mathbf{U}_i$$

这表示，每个隐藏表示为所有输入元素的加权求和。

而当输入是二维图像 (假设是 mxn) 时，用二维张量表示时，同样，每个隐藏表示 (i,j) 为所有输入像素 (k,l) 的加权求和，即：

$$\mathbf{H}_{i,j} = \sum_k \sum_l \mathbf{W}_{i,j,k,l}\mathbf{X}_{k,l} + \mathbf{U}_{i,j}$$

我们重新索引下标 (k,l), 使$k = i + a, l = j + b$(a, b遍历负值和正值之间)，再令 $\mathbf{V}_{i,j,a,b} = \mathbf{W}_{i,j,k,l}$，则有：

$$\mathbf{H}_{i,j} = \sum_a \sum_b \mathbf{V}_{i,j,a,b}\mathbf{X}_{i+a,j+b} + \mathbf{U}_{i,j}$$

我们重新审视这个公式

- 平移不变性：根据平移不变性，对于 $\mathbf{H}_{i,j}$ 的值不应该依赖于图像的位置，也就是和 (i,j) 无关，可以删去这两维，即：

$$\mathbf{H}_{i,j} = \sum_a \sum_b \mathbf{V}_{a,b}\mathbf{X}_{i+a,j+b} + u$$

- 局部性：为了确定 $\mathbf{H}_{i,j}$ 的值，不应该考虑离它太远的地方，即在 |a| > Δ 或 |b| > Δ 的

范围之外，$\mathbf{V}_{a,b} = 0$，则将上述公式再次重写为：

$$\mathbf{H}_{i,j} = \sum_{a=-\Delta}^{\Delta} \sum_{b=-\Delta}^{\Delta} \mathbf{V}_{a,b} \mathbf{X}_{i+a,j+b} + u$$

上述公式中 **H** 便是卷积层，**V** 是卷积核（convolution kernel）或者滤波器（filter）。

如果考虑通道，增加通道数 c，则公式如下：

$$\mathbf{H}_{i,j,c} = \sum_{a=-\Delta}^{\Delta} \sum_{b=-\Delta}^{\Delta} \sum_{d} \mathbf{V}_{a,b,c,d} \mathbf{X}_{i+a,j+b,d} + \mathbf{u}_c$$

这个公式是对于输出通道，将所有输入通道的二维张量和相应卷积核的二维张量进行互相关运算，再对通道求和。每个输出通道可以识别特定模式，而输入通道核识别并组合输入通道的模式。

7.1 卷积

实际上我们所说的卷积并不准确，它是互相关运算。输入大小 $n_h \times n_w$, 输出大小 $k_h \times k_w$, 输出大小为 $(n_h - k_h + 1) \times (n_w - k_w + 1)$

7.2 填充和步幅

填充（padding）可以使得输入和输出具有相同的高度和宽度。并且通常我们卷积核的高度和宽度为奇数，这样方便填充。(注意代码中的 padding 单指一侧)

如果我们添加 p_h 行填充（大约一半在顶部，一半在底部）和 p_w 列填充（左侧大约一半，右侧一半），则输出形状将为 $(n_h - k_h + 1 + p_h) \times (n_w - k_w + 1 + p_w)$

步幅可以降低图像的宽度和高度。当垂直步幅为 s_h、水平步幅为 s_w 时, 输出形状为:

$$\left\lfloor \frac{n_h - k_h + p_h + s_h}{s_h} \right\rfloor \times \left\lfloor \frac{n_w - k_w + p_w + s_w}{s_w} \right\rfloor.$$

更易理解应该是

$$\left\lfloor \frac{n_h - k_h + p_h}{s_h} + 1 \right\rfloor \times \left\lfloor \frac{n_w - k_w + p_w}{s_w} + 1 \right\rfloor.$$

7.3 多输入多输出通道

输入通道数等于卷积核通道数，输出通道数等于卷积核个数。多输入通道：

- 输入 $\mathbf{X}$: $c_i \times n_h \times n_w$
- 核 $\mathbf{W}$: $c_i \times k_h \times k_w$

- 输出 $\mathbf{Y}$: $m_h \times m_w$

$$\mathbf{Y} = \sum_{i=0}^{c_i} \mathbf{X}_{i,:,:} \star \mathbf{W}_{i,:,:}$$

多输出通道，每个三维卷积核生成一个输出通道：

- 输入 $\mathbf{X}$: $c_i \times n_h \times n_w$
- 核 $\mathbf{W}$: $c_o \times c_i \times k_h \times k_w$
- 输出 $\mathbf{Y}$: $c_o \times m_h \times m_w$

$$\mathbf{Y}_{i,:,:} = \mathbf{X} \star \mathbf{W}_{i,:,:,:} \quad \text{for } i = 1, \ldots, c_o$$

7.4 汇聚层

汇聚（pooling）层降低卷积层对位置的敏感性，同时降低对空间降采样表示的敏感性。分为最大汇聚层（maximum pooling）和平均汇聚层（average pooling）。

汇聚层不改变通道数。默认情况下，深度学习框架中的步幅与汇聚窗口的大小相同。

7.5 超参数选择

- 核大小：通常为小的奇数，如 3x3
- 填充：通常取核大小-1，保证形状不变
- 步幅：通常为 2，使得图片每次减半
- 通道数：
- 学习率: 1e-1,1e-2,1e-3
- 丢弃率: 0.1，0.5 或 0.9。
- L2 正则：1
- batch_size：128

7.6 LeNet

LeNet 是最早发布的卷积神经网络之一，其架构如下：

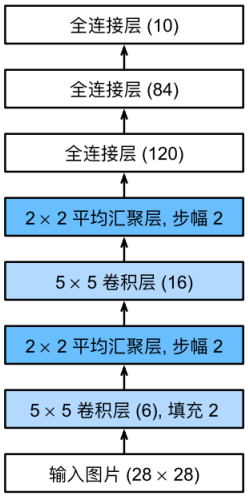


图 **18**

八、 现代卷积神经网络

8.1 ImageNet 历年冠军

- 2012：AlexNet
- 2014: GooLeNet、VGG
- 2015: ResNet

将人类直觉和相关数学见解结合后，诞生了一系列现代的卷积神经网络架构。

8.2 深度卷积神经网络（AlexNet）

对于计算机视觉研究人员来说，80% 的时间实际上都是花在数据上。

由于大数据集和硬件的进步，深度卷积神经网络取得突破，AlexNet 证明了学习到的特征可以超越手工设计的特征。此后计算机视觉领域方法发生改变：

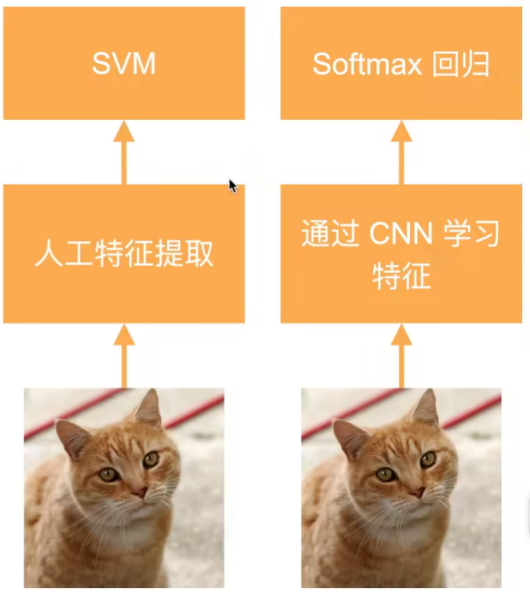


图 **19**

AlexNet 的设计架构如下：

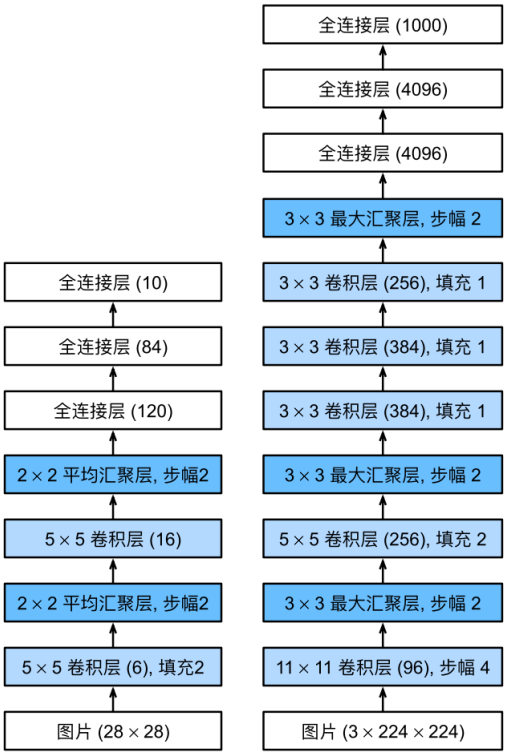


图7.1.2: 从LeNet（左）到AlexNet（右）

图 **20**

8.3 使用块的网络 (VGG)

VGG 从块的角度考虑神经网络的设计，使用可重复的块来构建神经网络。一个 VGG 块是有若干个卷积层和一个最大汇聚层组成，架构如下：

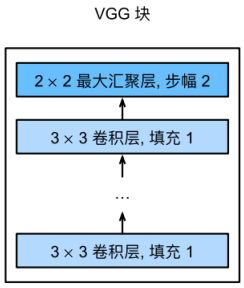


图 **21**

为 VGG-11 有 5 个 VGG 块，其中前两个块各有一个卷积层，后三个块各包含两个卷积层，最后是三个全连接层。

8.4 网络中的网络 (NiN)

之前的卷积神经网络后面都有全连接层，它占用的参数较多，并且容易过拟合，NiN 提出了 NiN 块，一个 NiN 块的组成是一个卷积层后面接两个 1x1 的卷积层 (带有 ReLU 激活函数的逐像素全连接层)，进行跨通道特征学习。

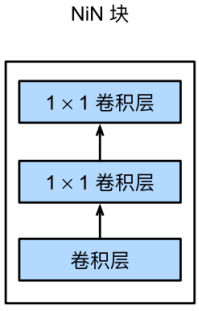


图 **22**

每个 NiN 块后面通常接最大汇聚层，逐步减半尺寸并增大通道数，最后使用全局平均汇聚层来得到输出 (设置成所需的输出通道数)：

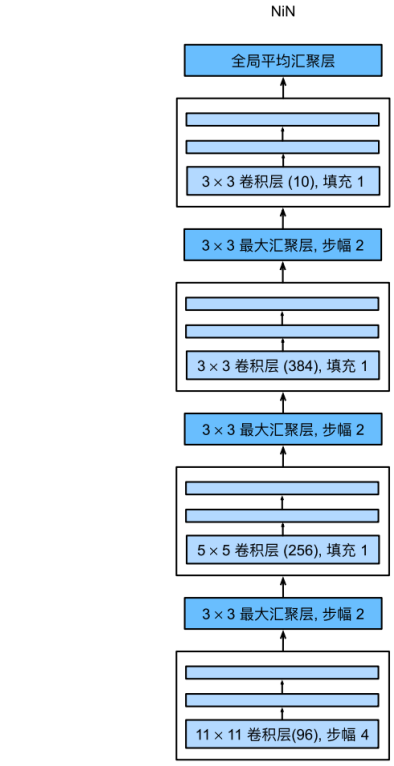


图 **23**

8.5 含并行连结的网络（GoogLeNet）

GoogLeNet 吸收了 NiN 中串联网络的思想，基本的卷积块被称为 Inception 块，由四条并行路径组成，从不同层面抽取信息，然后在输出通道维度合并。

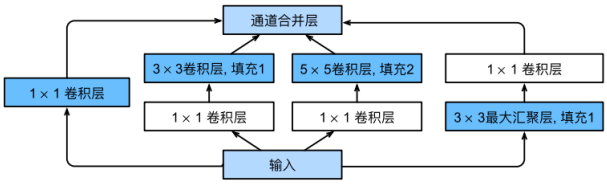


图 **24**

以一个 inception 为例：

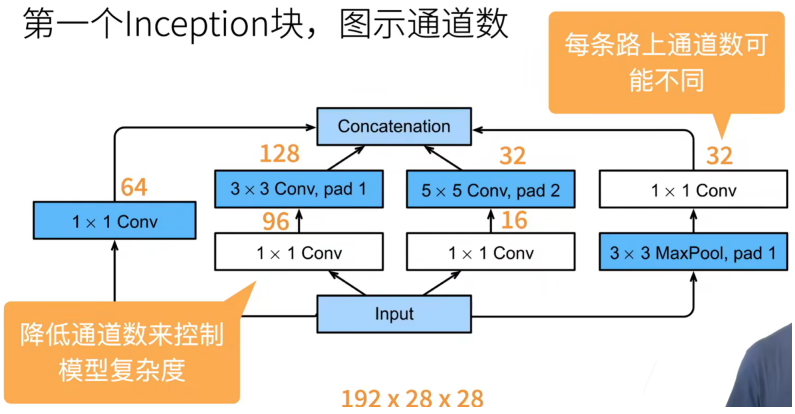


图 **25**

Inception 集合了多种卷积核，提前的信息更多样，并且复杂度更低。GoogLeNet 则是包含多个 Inception 的组合：

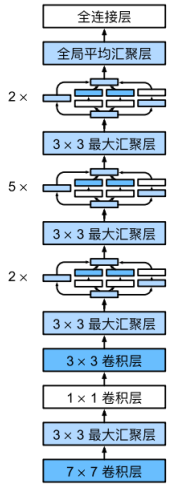


图 **26**

8.6 批量归一化

当神经网络层数较深时，底部层可能出现梯度消失现象，导致收敛变慢，batch normalization 可以固定每一层**小批量**输入的均值和方差，通过在每个小批量加入噪音来控制模型复杂度，加速收敛速度，即：

$$\mathbf{BN}(\mathbf{x}) = \gamma \odot \frac{\mathbf{x} - \boldsymbol{\mu}_{\mathcal{B}}}{\boldsymbol{\sigma}_{\mathcal{B}}} + \boldsymbol{\beta}.$$

其中，

$$\boldsymbol{\mu}_{\mathcal{B}} = \frac{1}{|\mathcal{B}|} \sum_{\mathbf{x} \in \mathcal{B}} \mathbf{x},$$

$$\boldsymbol{\sigma}_{\mathcal{B}}^2 = \frac{1}{|\mathcal{B}|}\sum_{\mathbf{x}\in\mathcal{B}}(\mathbf{x}-\boldsymbol{\mu}_{\mathcal{B}})^2 + \epsilon.$$

拉伸参数（scale）γ 和偏移参数（shift）β 是需要与其他模型参数一起学习的参数。

- 批量归一化层通常作用在激活函数前。
- 对于全连接层，作用在特征维上。
- 对于卷积层，作用在通道维上。(可以将图像每个元素看成一个样本，通道就是它的特征)

有了 batchnormalization 就可以实际训练出超过 100 层的神经网络了。

8.7 残差网络 ResNet

ResNet 的设计思路是复杂的函数包含原始较小的函数，确保性能不会降低，并且缓解了梯度消失现象。

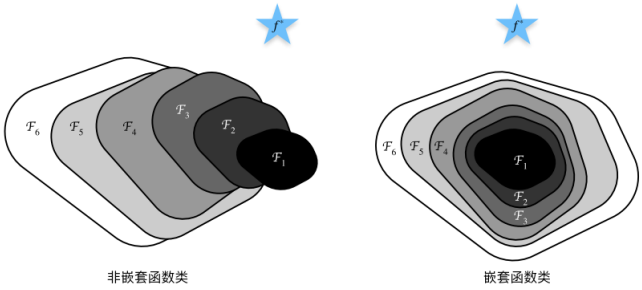


图 **27**

架构如下，需要的话可以使用 1 × 1 卷积可以调整通道和分辨率。

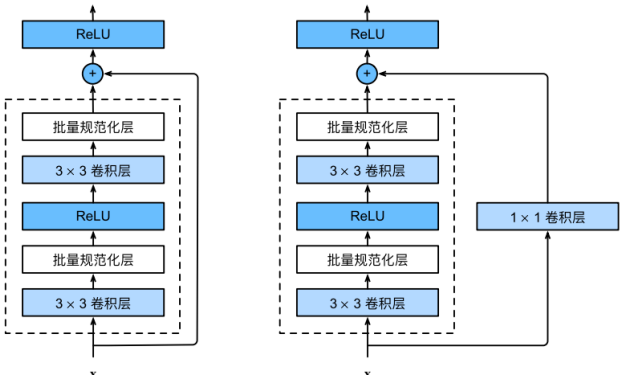


图 **28**

ResNet-18 架构如下：

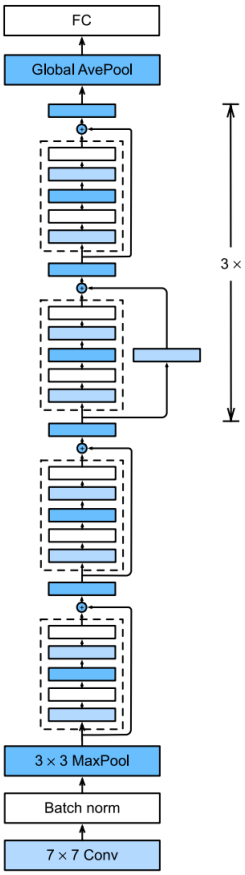


图 **29**

8.8 ResNet 的对抗攻击

利用模型损失函数相对于输入样本的梯度信息来生成对抗样本，沿梯度方向增加扰动，使得损失函数增加最快。

$$x_{\text{adv}} = x + \epsilon \cdot \text{sign}(\nabla_x J(\theta, x, y))$$

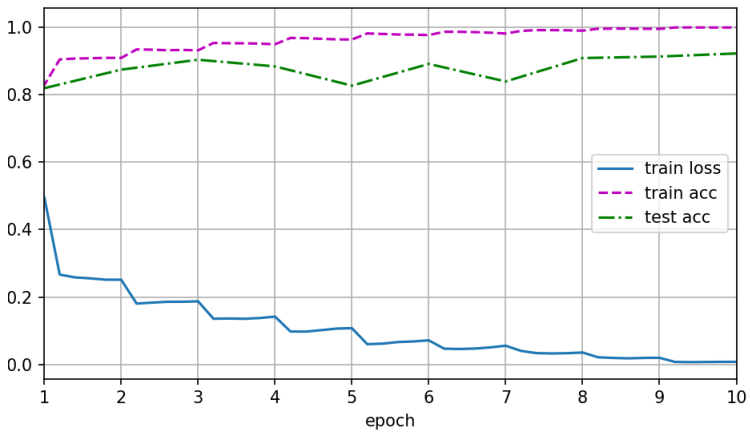


图 **30**

```
loss 0.008, train acc 0.998, test acc 0.921
932.7 examples/sec on cuda:0
对抗样本的准确率: 33.20%
```

图 **31**

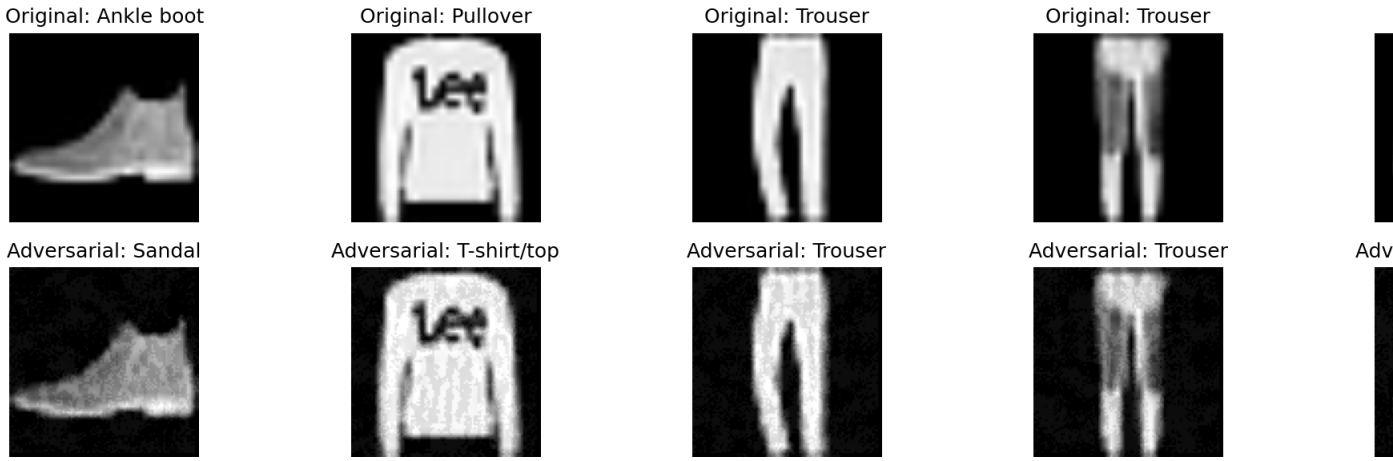


图 **32**

参考文献

[1]《动手学深度学习》