学习报告（第二周）

25325186 刘金河

一、Neural Networks and Backpropagation

1、学习了神经网络与反向传播的原理，认识了几种不同的激活函数，实践构建了两层的神经网络对 Cifer 10 数据集进行模型的训练与预测，还搭建了全连接的神经网络，进一步提升模型的预测准确度，通过不断调整超参数，最终在测试集上达到 60%左右的准确度。

2、神经网络的基本结构：

①2 层神经网络：$f = W_2\max(0, W_1x)$

其中，W_1为第一层权重矩阵，$\max(0,\cdot)$为激活函数，W_2为第二层权重矩阵；

②3 层神经网络：$f = W_3\max(0, W_2\max(0, W_1x))$

其中，W_1为第一层权重矩阵，$\max(0,\cdot)$为激活函数，W_2为第二层权重矩阵，W_3为第二层权重矩阵；

③维度匹配原则：

输入$x \in \mathbb{R}^D$（D 为输入特征数），

$W_1 \in \mathbb{R}^{H_1 \times D}$（$H_1$为第一层隐藏层神经元数），

$W_2 \in \mathbb{R}^{H_2 \times H_1}$（$H_2$为第二层隐藏层神经元数），

输出层维度由任务需求决定（如分类任务的类别数,在 CIFER-10 中维度为 10）；

④偏置项：实际应用中，通常为每一层添加可学习偏置项b_1, b_2等，即$f = W_2\max(0, W_1x + b_1) + b_2$，以增强模型表达能力（从“线性变换”变成“仿射变换”）。

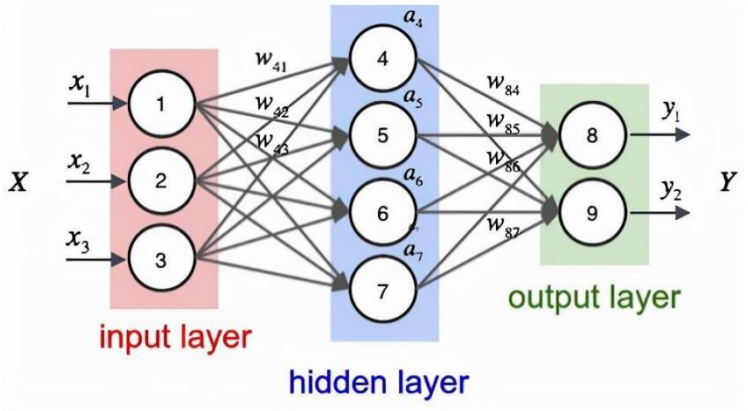

图 1 一个三层的神经网络示例

3、激活函数：

定义：

激活函数是对神经元输出进行非线性变换的函数，记为$\sigma(\cdot)$。其位置在每一层的“线性变换（$Wx + b$）”之后，即每一层的完整计算为：输出 $\mathrm{h} = \sigma(Wx + b)$。

必要性：

激活函数使得神经网络的各层之间不再是简单的线性变化，如果没有激活函数，那么无论多少层的神经网络都最终会退化为一个一维的线性变换，相较于普通的线性变换性能并无显著提升。

常见类型如下表所示：

表 1　常用激活函数

激活函数	公式（简化版）	核心特性	优点	缺点
ReLU	$\sigma(z) = \max(0, z)$	分段线性：$z > 0$ 时输出 z，$z \leq 0$ 时输出 0	1. 计算简单：无指数或三角函数； 2. 梯度稳定：z> 0 时梯度为 1，无梯度时消失。	神经元死亡：$z \leq 0$ 时梯度为 0，参数无法更新
Leaky ReLU	$\sigma(z) = \max(\alpha z, z))$ $(\alpha = 0.01)$	解决 ReLU 函数神经元死亡问题：$z < 0$ 时输出αz（小斜率），$z \geq 0$)时输出 z	避免神经元死亡：$z < 0$ 时梯度=α≠0	α 需手动调整
Sigmoid	$\sigma(z) = \frac{1}{1 + e^{-z}}$	S 形曲线：输出$\sigma(z) \in (0,1)$，可表示“概率”	1. 输出可作为概率 2. 梯度平滑	当 z 偏小或偏大时梯度消失
Tanh	$\sigma(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$	双 S 形曲线：输出∈(-1,1)，零中心	非线性强	与 Sigmoid 相似，存在梯度消失问题，但相对轻微

下面是以上 4 个函数的函数图像。

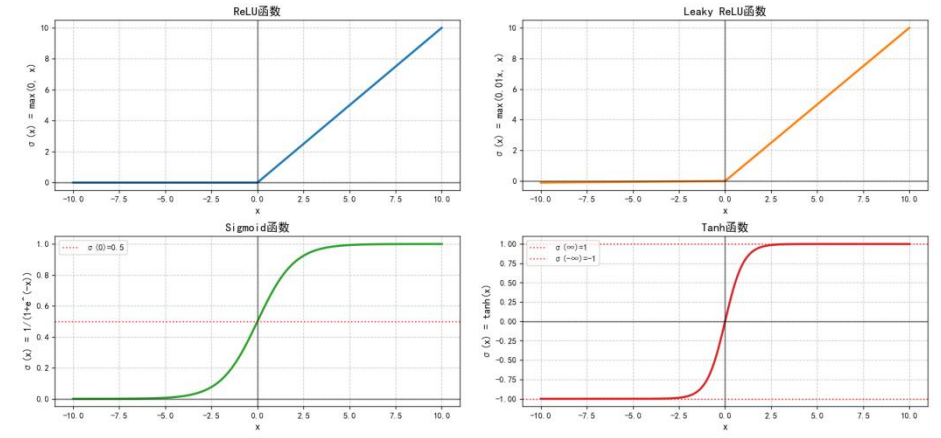

图 2　常用激活函数对应的图像

4、损失函数与正则化：（关联第三课的学习内容）

得分函数：神经网络的输出结果，形如$W_2 \max(0, W_1 x + b_1) + b_2$；

正则化：惩罚模型权重的过大值，避免过拟合，这里我们采用 L2 正则化函数：$R(W) = \sum_k W_k^2$，即所有权重参数的平方和；

总损失：$L = \frac{1}{N}\sum_{i=1}^{N} L_i + \lambda R(W_1) + \lambda R(W_2)$，其中，N 为样本总数，λ为正则化强度（λ越大，正则化越强）。

5、反向传播：

由于神经网络训练的核心是通过梯度下降不断更新权重参数，因此需计算总损失对每个权重的梯度$\nabla_W L$，但由于计算量随着神经网络的层数增大而呈指数级增加，因此需要找到一个方法，用于自动计算并更新神经网络的梯度，所以，在此处我们引入了反向传播。

①核心逻辑

反向传播的核心其实是链式法则，即通过对各层函数的各个参数进行偏导数运算，再通过“局部梯度”与“上游梯度”的乘积，得到最终梯度。

1.前向传播：计算中间变量与最终输出（损失函数值为 L）；

2.反向传播：从 L 出发，计算每个中间变量对 L 的梯度（上游梯度），再结合局部梯度（即偏导数），通过链式法则得到输入对 L 的梯度。

②示例：

定义函数 $f(x,y,z)=(x+y)z$，对其进行梯度计算（当然，这并不算是一个神经网络，毕竟只是简单的线性变化，仅是作为示例）：

1）分解运算：

第一步：$q=x+y$，局部梯度（偏导数）为$\frac{\partial q}{\partial x}=1$，$\frac{\partial q}{\partial y}=1$；

第二步：$f=q\cdot z$，局部梯度为$\frac{\partial f}{\partial q}=z$，$\frac{\partial f}{\partial z}=q$；

2）链式法则计算梯度：若已知上游梯度$\frac{\partial L}{\partial f}$，则：

$$\frac{\partial L}{\partial z}=\frac{\partial L}{\partial f}\times\frac{\partial f}{\partial z}$$

$$\frac{\partial L}{\partial x}=\frac{\partial L}{\partial f}\times\frac{\partial f}{\partial q}\times\frac{\partial q}{\partial x}$$

$$\frac{\partial L}{\partial y}=\frac{\partial L}{\partial f}\times\frac{\partial f}{\partial q}\times\frac{\partial q}{\partial y}$$

上面这个计算过程的代码实现如下图所示：

```python
def compute_gradients_forwars(x, y, z):
    # 前向传播：计算中间变量和输出
    q = x + y
    f = q * z
    #储存中间变量以供反向传播使用
    cache = (x, y, z, q, f)
    return f, q, cache
def compute_gradients_backwars(dL_df,cache):
    #反向传播：计算梯度
    #回溯中间变量
    z, q, f = cache
    # 计算 f=q*z 的局部梯度
    df_dq = z
    df_dz = q
    # 链式法则：传递梯度到q和z
    dL_dq = dL_df * df_dq
    dL_dz = dL_df * df_dz
    # 计算 q=x+y 的局部梯度
    dq_dx = 1
    dq_dy = 1
    # 链式法则：传递梯度到x和y
    dL_dx = dL_dq * dq_dx
    dL_dy = dL_dq * dq_dy
    return f , dL_dx, dL_dy, dL_dz

```

图 3　示例函数的梯度计算

6、反向传播中几种常见的模型

神经网络反向传播中的基本运算被称为“计算门”，主要有以下几种模型：

①Add Gate（加法门）：$q = a + b$

输入梯度等于输出梯度，即$(\frac{\partial L}{\partial a} = \frac{\partial L}{\partial q})$，$(\frac{\partial L}{\partial b} = \frac{\partial L}{\partial q})$；

②Mul Gate（乘法门）：$q = a \times b$

输入梯度为输出梯度乘另一个输入，即$\frac{\partial L}{\partial a} = \frac{\partial L}{\partial q} \times b$，$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial q} \times a$；

③Copy Gate（复制门）：$q = a$，$r = a$（将输入 a 复制到两个输出）

输入梯度为所有输出梯度之和，即$\frac{\partial L}{\partial a} = \frac{\partial L}{\partial q} + \frac{\partial L}{\partial r}$；

④Max Gate（最大值门）：$q = \max(a, b)$

最大值对应的输入局部梯度为 1，其余为 0，即若 $a > b$，则$\frac{\partial L}{\partial a} = \frac{\partial L}{\partial q}$，$\frac{\partial L}{\partial b} = 0$；

7、学习率调整：

学习率是梯度下降算法的核心超参数（$\theta = \theta - \alpha \cdot \nabla_\theta L$，$\alpha$ 为学习率），动态调整学习率可有效提升模型收敛速度和预测准确度，以下是常用的学习率调整的方法：

Step 调度（阶梯式衰减）：

固定训练轮次（epoch）将学习率乘以一个关于衰减系数与训练轮次的函数；

公式：$\alpha_t = \alpha_0 \times \gamma^{t/s}$（$\gamma$ 为衰减系数，s 为衰减步长，t 为训练轮次）；

Cosine 调度（余弦周期衰减）：

学习率随训练轮次呈余弦函数周期性变化，后期学习率缓慢下降，利于模型收敛到更优解；

公式：$\alpha_t = \frac{1}{2}\alpha_0(1 + \cos(\frac{t\pi}{T}))$（T 为总训练轮次，$\alpha_0$为初始学习率）；

Linear 调度（线性衰减）：

学习率从初始值线性下降至 0；

公式：$\alpha_t = \alpha_0(1 - \frac{t}{T})$；

Inverse sqrt 调度（平方根衰减）：

学习率随训练轮次的平方根反比例衰减，适用于数据量较大、训练轮次较长的场景；

公式：$\alpha_t = \alpha_0/\sqrt{t + \epsilon}$（$\epsilon$是一个为避免分母为 0 的小常数）。

下图是上面四种调度方式对应的关于训练轮次与学习率的函数图像。

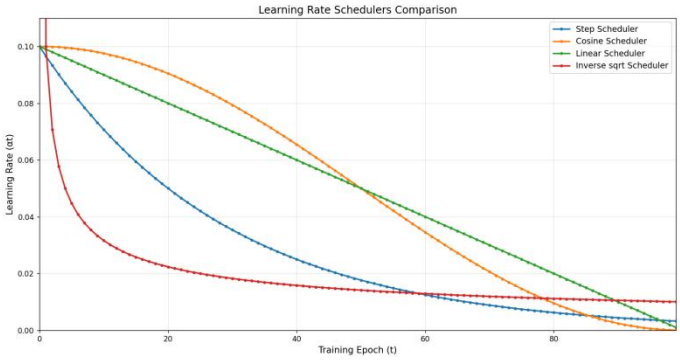

图 4　关于训练轮次与学习率的四种函数图像

二、Image Classification with CNNs

1、背景：由于此前的所有训练方法，包括矩阵变换、全连接神经网络，都破坏了图像的二维结构，因此为了保留图像的二维结构特征，我们引入了 CNNs（卷积神经网络）算法，用于预测图像的所属类别。

2、CNN 的核心组件：

(1) 卷积层（Convolutional Layer）

卷积层通过滑动卷积核（Filter/Kernel）读取输入数据中的局部数据，是 CNN 算法的核心模块。

数学原理：输入为三维张量（通道数C_{in}× 高度H_{in}× 宽度W_{in}），单个卷积核尺寸为 K×K×C_{in}（与输入通道数一致），输出特征图（Feature Map）的尺寸由公式严格计算：

$$H_{out} = \left\lfloor \frac{H_{in} - K + 2P}{S} \right\rfloor + 1$$

$$W_{out} = \left\lfloor \frac{W_{in} - K + 2P}{S} \right\rfloor + 1$$

其中：

P 为填充（Padding）：在输入边缘补充 0 值，用于控制输出尺寸（避免特征图过度收缩）；

S 为步幅（Stride）：卷积核滑动的步长，S>1 时可实现下采样；

⌊ · ⌋ 为向下取整，确保输出尺寸为整数。

(2) 池化层（Pooling Layer）

池化层通过下采样降低特征图的空间维度，减少计算量与过拟合风险。

常见的池化层有以下两个类型：

①最大池化（Max Pooling）：取池化窗口内的最大值，保留局部最显著特征。

例如输入大小为 64×224×224 的数据经(2×2)最大池化(S=2)后，输出为数据大小为(64×112×112)（通道数不变，空间维度减半）；

②平均池化（Average Pooling）：取池化窗口内的平均值，输出的数据整体上比最大池化输出的更加平滑。

(3) 激活函数

激活函数用于为神经网络引入非线性，使 CNN 能够拟合复杂的非线性映射，在 1.4 中已经有详细描述，在此不再赘述。

(4) 全连接层

全连接层将池化层输出的三维特征图展平为一维向量（如将大小为 64×112×112 的数据展平为 64×112×112=802,816 维），再将特征映射到输出类别得分（如 10 类图像分类则输出 10 维得分向量）。

3、CNN 算法的完整架构

(1).输入层：接收原始图像数据（如大小为 3×32×32 的 RGB 图像）；

(2).特征提取模块：多轮 “卷积（Conv）→激活（ReLU）→池化（Pool）” 迭代；

(3).分类决策模块：展平（Flat）→全连接层（FC）→输出层（Softmax，将得分转化为概率分布）。

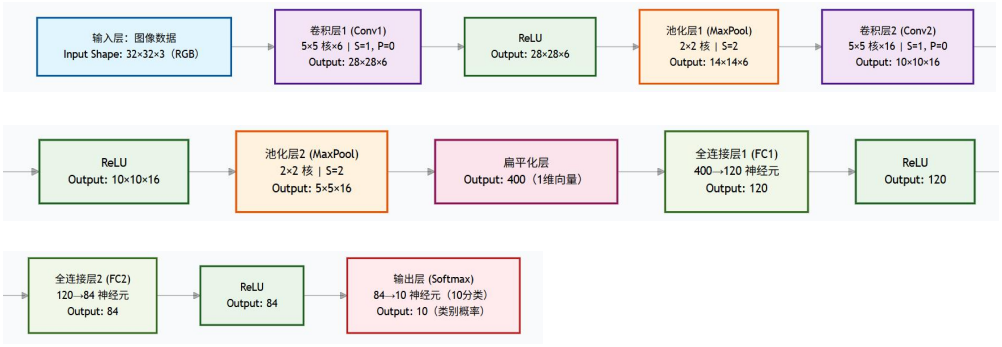

图 5 CNN 算法完整架构示例

4、CNN 算法各层的梯度传播规则

1.卷积层梯度：

输出特征图的梯度需通过 “转置卷积” 反向传播至输入，同时计算卷积核权重的梯度（每个权重的梯度为输入对应区域与输出梯度对应区域的元素乘积和）；

2.池化层梯度：

最大池化：梯度仅传递给池化窗口内最大值对应的输入位置（其他位置梯度为 0）；

平均池化：梯度均匀分配给池化窗口内的所有输入位置；

3.激活函数梯度（以下仅给出 ReLU 和 Leaky ReLU 的反向传播规则，详见 1.4）：

ReLU：$f'(x)=1$（$x>0$），$f'(x)=0$（$x\leq 0$）；

Leaky ReLU：$f'(x)=1$（$x>0$），$f'(x)=0.1$（$x\leq 0$）。

三、CNN Architectures

1、CNN 算法的优化

(1)学习率调度（Learning Rate Scheduling）

学习率是控制参数更新幅度的核心超参数，如果学习率过大，会导致训练震荡不收敛，过小则训练速度极慢。

以下是三种常用学习率调度策略：

①Step 调度：固定训练轮次（Epoch）将学习率乘以衰减因子（如 ResNet 在 Epoch 30、60、90 时学习率 ×0.1）；

②Cosine 调度：学习率随训练轮次呈余弦曲线衰减，公式为 $\alpha_t=\frac{1}{2}\alpha_0(1+\cos(\frac{t\pi}{T}))$（T 为总轮次），后期学习率缓慢下降，有助于模型收敛到更优解；

③Linear 调度：学习率从初始值线性衰减至 0，适用于短训练周期；

④Inverse sqrt 调度：$\alpha_t=\alpha_0/\sqrt{t}$，前期学习率下降较快，后期趋于稳定。

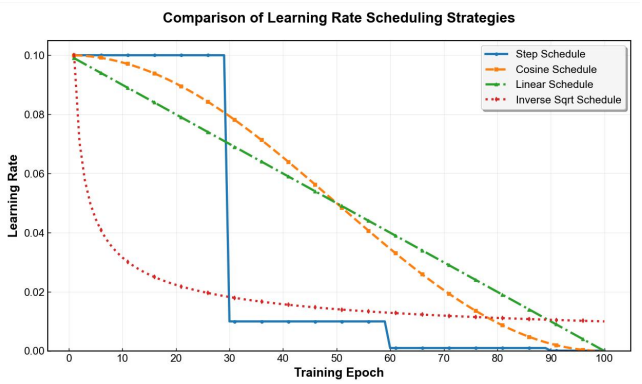

图 6　四种调度方式的学习率曲线

(2)正则化（Regulation）

CNN 算法通常使用正则化来降低模型过拟合的风险，以下是两种常用的正则化方法：

①L2 正则化：通过在损失函数中添加参数平方和，惩罚过大的参数值，使模型权重趋于平缓，减少过拟合；

②Dropout：训练时随机将部分神经元的输出置为 0（失活概率 p，通常取 0.5），迫使模型学习冗余的特征表示，避免依赖单一神经元。测试时不进行失活，而是将所有神经元的输出乘以$(1-p)$，相当于在 CNN 的某一层中使用的神经元占该层神经元的比重为$(1-p)$；

(3)归一化（Normalization）：

归一化通过标准化输入数据或中间特征，缓解梯度消失的问题，并且可以加速训练收敛，以下是几种常用的归一化的方法：

①Batch Normalization（BN）：对每个批量的特征图进行归一化（均值为 0、方差为 1），再通过可学习参数γ（缩放）和β（偏移）调整特征分布，公式为：

$$\hat{x}=\frac{x-\mu_B}{\sqrt{\sigma_B^2+\epsilon}},\quad y=\gamma\hat{x}+\beta$$

其中μ_B、σ_B^2为批量数据的均值和方差，ϵ为防止分母为 0 的微小值（通常设置为$1e-8$）。

②输入图像的通道归一化

在将图像输入 CNN 前，需对每个颜色通道单独进行归一化（课程图示），步骤如下：

1.预计算数据集每个颜色通道（RGB）的均值μ_c与标准差σ_c；

2.对输入图像的每个像素，按通道应用：

$$norm_{pixel}[i,j,c]=\frac{pixel[i,j,c]-\mu_c}{\sigma_c+\epsilon}$$

其中ϵ为防止分母为 0 的微小值（通常取$1e-8$）。该操作使每个通道的输入

的数据均值为 0、方差为 1，有效避免了因像素值尺度差异导致的梯度不稳定。

2、数据增强（Data Augmentation）

为了使得模型能够适应不同的环境，降低模型对输入数据的依赖，我们可以使用下面这些方法对输入数据进行处理。

(1) 随机裁剪：随机选取输入数据中的一部分数据，输入到训练模型中；

(2) 图像缩放：将输入的图像进行缩放，如最大池和平均池；

(3) 几何变换：随机翻转（水平 / 垂直）、旋转、仿射变换；

(4) 颜色变换：随机调整亮度、对比度、饱和度、色相，添加高斯噪声；

原理：变换后的图像仍保留原始标签（如猫的图像翻转后仍是猫），这使得模型必须学习与图像姿态、颜色等无关的核心特征，从而达到减少过拟合的目的。

3、迁移学习策略

当我们的数据集较小时，直接使用深层的 CNN 算法进行训练容易导致模型过拟合，迁移学习通过复用预训练模型的特征提取能力，从而提升模型的泛化能力：

预训练基础：通常使用在 ImageNet（1000 类、百万级图像）上预训练的模型（如 VGG、ResNet 等），其低层卷积层已学习到通用图像特征（如边缘、纹理等）；

策略选择：

i.小数据集 + 相似任务：冻结预训练模型的所有层，仅替换顶层全连接层为目标类别数的分类器，仅针对顶层的超参数进行微调；

ii.大数据集 + 相似任务：解冻预训练模型的所有层，用较小的学习率微调所有参数；

iii.小数据集 + 不同任务：尝试其他模型或扩充数据；

iv.大数据集 + 不同任务：可微调所有层，或直接从头训练模型；

核心逻辑：预训练模型的低层特征常常具有通用性，而高层特征更与任务相关，因此在数据集较小的情况下通常冻结低层、训练高层。

四、总结与体会

第二周我围绕着神经网络与反向传播、卷积、CNN 算法等内容的学习，深入了解了如何搭建 CNN 算法进行模型训练，还学会了一些处理数据集的方法，使得模型能够应用于更多的场景。

在学习中，我深刻体会到矩阵运算和偏导数对于神经网络和反向传播的重要性，对 CNN 算法的深入学习为我后续学习其他模型打下了坚实基础。