Assignment2

说明

本次作业是个人知识与见解、大模型辅助、网络上查询的资料共同完成的作业

回答里面加粗的部分我认为是一些关键的句子，我的大部分个人知识与见解也是体现在这些加粗的地方。

大模型主要用于辅助书写数学公式、查找资料、组织概括语言，一些开放性的题目会有大模型辅助生成观点。但本次作业绝非仅靠大模型完成，融入了不少我自己的想法和理解（即使这些想法和理解与大模型写出来的确实有很多类似的地方）。

A2.1

Q

假设我们有一个10层的全连接神经网络，使用Sigmoid作为激活函数，采用均方误差损失函数。请从链式法则的角度定性分析，在反向传播过程中，靠近输入层的参数梯度可能会发生什么变化？为什么？如果将Sigmoid激活函数换成ReLU，这个问题会得到缓解吗？请解释原因。除了更换激活函数，请再列举两种缓解梯度消失/爆炸问题的方法，并简要说明其原理。

A

基础定义与建模

我们首先需要明确定义网络结构、激活函数、损失函数以及反向传播。

1.1 Sigmoid 激活函数

Sigmoid函数定义如下：

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

它的输出值域为 (0, 1)，其导数具有一个非常简洁且重要的形式：

$$\sigma'(x) = \frac{d}{dx}\sigma(x) = \sigma(x)(1 - \sigma(x))$$

这个导数形式在反向传播中至关重要。由于 $\sigma(x) \in (0, 1)$，所以 $\sigma'(x)$ 的最大值出现在 $x = 0$ 处，此时 $\sigma(0) = 0.5$，导数为 0.25。对于任何其他 x 值，导数都会小于 0.25。**这是一个关键性质：Sigmoid函数的导数有上界，且在输入值较大或较小时会趋近于0 。**

1.2 ReLU激活函数

ReLU函数定义为：

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

其导数为：

$$\mathrm{ReLU}'(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x > 0 \\ \text{undefined} & \text{if } x = 0 \end{cases}$$

在实际实现中，通常将 $x = 0$ 处的导数定义为0或1。**ReLU的关键特性是，对于所有正输入，其梯度恒为1，不会衰减 。**

1.3 网络前向传播建模

考虑一个L层（此处L=10）的全连接神经网络。我们定义：

- $\mathbf{x}^{(0)} \in \mathbb{R}^{d_0}$ 为输入向量。
- $\mathbf{W}^{(l)} \in \mathbb{R}^{d_l \times d_{l-1}}$ 为第 l 层的权重矩阵。
- $\mathbf{b}^{(l)} \in \mathbb{R}^{d_l}$ 为第 l 层的偏置向量。
- $\mathbf{z}^{(l)} \in \mathbb{R}^{d_l}$ 为第 l 层的线性变换（加权和）结果。
- $\mathbf{a}^{(l)} \in \mathbb{R}^{d_l}$ 为第 l 层的激活输出。

前向传播过程可以递归地表示为：

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

其中 $\phi(\cdot)$ 是激活函数。对于输出层（第L层），我们有 $\hat{\mathbf{y}} = \mathbf{a}^{(L)}$。

1.4 损失函数

我们使用均方误差（MSE）作为损失函数。假设真实标签为 $\mathbf{y}$，则损失为：

$$\mathcal{L} = \frac{1}{2}\|\hat{\mathbf{y}} - \mathbf{y}\|^2 = \frac{1}{2}\sum_{i=1}^{d_L}(\hat{y}_i - y_i)^2$$

1.5 反向传播的核心：误差项 $\delta^{(l)}$

我们定义第 l 层的误差项为：

$$\delta^{(l)} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{(l)}}$$

这个量是计算该层权重和偏置梯度的关键。

根据链式法则，我们可以推导出 $\delta^{(l)}$ 的递推关系。**而且很重要的一点，我们必须通过反向的顺序来计算梯度。**

对于输出层 $(l = L)$:

$$\delta^{(L)} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{(L)}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{(L)}} \odot \frac{\partial \mathbf{a}^{(L)}}{\partial \mathbf{z}^{(L)}} = (\hat{\mathbf{y}} - \mathbf{y}) \odot \phi'(\mathbf{z}^{(L)})$$

其中 $\odot$ 表示逐元素相乘。

对于隐藏层 $(l < L)$:

由于是反向传播，我们已经知道比l更深的层的梯度，我们将它们保存起来（这也是占用显存的原因），当我们计算第l层的梯度时会直接用到第$l+1$层已经计算好的梯度，间接地用到所有比l深的层计算好的梯度。

这是链式法则应用的关键步骤。我们需要将 $\mathcal{L}$ 对 $\mathbf{z}^{(l)}$ 的导数，通过 $\mathbf{z}^{(l+1)}$ 连接起来。

$$
\begin{aligned}
\delta^{(l)} &:= \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{(l)}} \\
&= \left(\frac{\partial \mathbf{z}^{(l+1)}}{\partial \mathbf{z}^{(l)}} \right)^{\top} \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{(l+1)}} \\
&= \left(\frac{\partial (\mathbf{W}^{(l+1)} \mathbf{a}^{(l)})}{\partial \mathbf{z}^{(l)}} \right)^{\top} \delta^{(l+1)} \\
&= \left(\frac{\partial (\mathbf{W}^{(l+1)} \mathbf{a}^{(l)})}{\partial \mathbf{a}^{(l)}} \cdot \frac{\partial \mathbf{a}^{(l)}}{\partial \mathbf{z}^{(l)}} \right)^{\top} \delta^{(l+1)} \\
&= \left(\mathbf{W}^{(l+1)} \cdot \operatorname{diag}\left(\phi'(\mathbf{z}^{(l)}) \right) \right)^{\top} \delta^{(l+1)} \\
&= \operatorname{diag}\left(\phi'(\mathbf{z}^{(l)}) \right) \cdot (\mathbf{W}^{(l+1)})^{\top} \delta^{(l+1)} \\
&= \left((\mathbf{W}^{(l+1)})^{\top} \delta^{(l+1)} \right) \odot \phi'(\mathbf{z}^{(l)})
\end{aligned}
$$

这个递推公式清晰地展示了梯度是如何从后向前传递的。

1.6 权重梯度的计算

有了 $\delta^{(l)}$，我们可以直接计算权重 $\mathbf{W}^{(l)}$ 的梯度：

$$
\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(l)}} = \delta^{(l)} (\mathbf{a}^{(l-1)})^T
$$

反向传播与梯度消失分析（Sigmoid情况）

现在，我们观察于 $\delta^{(l)}$ 的递推公式，并将其展开，以观察梯度在深层网络中的变化。

$$
\delta^{(l)} = \left((\mathbf{W}^{(l+1)})^T \left((\mathbf{W}^{(l+2)})^T \cdots \left((\mathbf{W}^{(L)})^T \delta^{(L)} \right) \odot \phi'(\mathbf{z}^{(L-1)}) \cdots \right) \odot \phi'(\mathbf{z}^{(l+1)}) \right) \odot \phi'(\mathbf{z}^{(l)})
$$

我们可以看到，$\delta^{(l)}$ 是一个连乘的形式，包含了从第 l 层到输出层之间所有激活函数的导数 $\phi'(\mathbf{z}^{(k)})$（其中 $k = l, l+1, \ldots, L-1$）以及所有权重矩阵的转置。

关键点在于激活函数的导数 $\phi'(z)$。

- 当 ϕ 是 Sigmoid 时，$\phi'(z) = \sigma(z)(1 - \sigma(z)) \leq 0.25$。
- 在训练初期，如果权重是随机初始化的，z 的值可能很大或很小，导致 $\sigma(z)$ 趋近于0或1，从而使 $\phi'(z)$ 趋近于0。

基于上述两点，在使用Sigmoid激活函数的深度网络中，由于其导数在大部分输入区域都小于1（甚至趋近于0），**在深度神经网络的反向传播过程中，许多小于1的数通过链式法则产生的连乘效应会导致梯度指数级地衰减。**因此，靠近输入层的参数梯度会变得非常小，几乎为零（**而在代码实现的时候，由于数字过小，会直接默认为0**）。这种现象被称为**梯度消失** 。这使得网络的浅层（靠近输入）几乎无法得到有效更新，导致训练极其缓慢甚至停滞。

ReLU激活函数的缓解作用

现在，我们将激活函数 ϕ 替换为ReLU。

ReLU的导数为：

$$
\operatorname{ReLU}'(x) = \begin{cases} 0 & x < 0 \\ 1 & x > 0 \end{cases}
$$

将这个导数代入到 $\delta^{(l)}$ 的递推公式中：

$$\delta^{(l)} = \left((\mathbf{W}^{(l+1)})^T \delta^{(l+1)} \right) \odot \text{ReLU}'(\mathbf{z}^{(l)})$$

分析：

- 对于那些 $\mathbf{z}^{(l)} > 0$ 的神经元，其导数为 1。这意味着，只要信号在前向传播中是正的，在反向传播中梯度就能无衰减地通过这一层。
- 虽然对于 $\mathbf{z}^{(l)} < 0$ 的神经元，梯度会被置为0，但只要网络中有足够多的神经元处于激活状态（即 $z > 0$），从输出层回传的梯度就不会像Sigmoid那样经历连续的、小于1的因子相乘。

因此，**ReLU通过在正区间提供恒定的、非衰减的梯度（=1），极大地缓解了梯度消失问题。**这使得训练非常深的网络成为可能，也是ReLU及其变体在现代深度学习中占据主导地位的重要原因之一 。

其他缓解梯度消失/爆炸问题的方法

方法一：合适的权重初始化

问题：不恰当的初始化（如全零初始化或过大/过小的随机初始化）会直接导致前向传播时激活值过大或过小，进而在反向传播时因激活函数导数饱和而引发梯度消失或爆炸。

解决方案：**Xavier初始化** 和 **He 初始化。**

1. Xavier初始化 (适用于Sigmoid, Tanh)
其核心思想是保持前向传播时激活值的方差和反向传播时梯度的方差在各层之间大致相等。也就是说希望每层输入和输出的方差、反向传过来的每层的梯度的方差大致相等。

- **定义**：对于一个权重矩阵 $\mathbf{W} \in \mathbb{R}^{n_{out} \times n_{in}}$，从均匀分布或正态分布中采样。
  - 均匀分布: $\mathbf{W} \sim \mathcal{U}\left(-\sqrt{\frac{6}{n_{in}+n_{out}}}, \sqrt{\frac{6}{n_{in}+n_{out}}}\right)$
  - 正态分布: $\mathbf{W} \sim \mathcal{N}\left(0, \sqrt{\frac{2}{n_{in}+n_{out}}}\right)$
- **原理简述**：
  假设输入 x 的方差为 $\text{Var}(x)$，权重 w 的方差为 $\text{Var}(w)$，且均值为0。对于线性输出 $z = \sum_{i=1}^{n_{in}} w_i x_i$，其方差为：

  $$\text{Var}(z) = n_{in} \cdot \text{Var}(w) \cdot \text{Var}(x)$$

  我们希望输入输出的方差大致相等，即 $\text{Var}(z) = \text{Var}(x)$，我们需要 $\text{Var}(w) = 1/n_{in}$。

  同理，假设损失 $\mathcal{L}$ 对本层线性输出 $\mathbf{z}$ 的梯度 $\frac{\partial \mathcal{L}}{\partial \mathbf{z}}$ 的方差为 $\text{Var}(\delta_z)$，权重 w 的方差为 $\text{Var}(w)$，且均值均为0。对于损失对上一层激活值 $\mathbf{a}$ 的梯度，其单个元素为：

  $$\frac{\partial \mathcal{L}}{\partial a_j} = \sum_{i=1}^{n_{out}} w_i \frac{\partial \mathcal{L}}{\partial z_i}$$

  其方差为：

  $$\text{Var}\left(\frac{\partial \mathcal{L}}{\partial a_j}\right) = n_{out} \cdot \text{Var}(w) \cdot \text{Var}(\delta_z)$$

  我们希望反向传播时梯度的方差大致保持不变，即 $\text{Var}\left(\frac{\partial \mathcal{L}}{\partial a_j}\right) = \text{Var}(\delta_z)$，我们需要 $\text{Var}(w) = 1/n_{out}$。

  Xavier初始化试图**同时满足前向和反向的方差稳定。**它取两个条件的折中，即 $Var(w) = 2/(n_{in} + n_{out})$

如果从均匀分布 $\mathcal{U}(-a, a)$ 采样，其方差为 $a^2/3$。令 $a^2/3 = 2/(n_{in} + n_{out})$，解得 $a = \sqrt{6/(n_{in} + n_{out})}$。- 如果从正态分布采样，直接设置方差为 $\sigma^2 = 2/(n_{in} + n_{out})$，所以标准差 $\sigma = \sqrt{2/(n_{in} + n_{out})}$。这就是定义里公式的由来。

2. He 初始化 (适用于ReLU)

- **定义**：$\mathbf{W} \sim \mathcal{N}\left(0, \sqrt{\frac{2}{n_{in}}}\right)$
- **原理简述**：假设输入激活值 $\mathbf{a}$ 的方差为 $\mathrm{Var}(a)$，权重 w 的方差为 $\mathrm{Var}(w)$，且均值均为0。对于线性输出 $z = \sum_{i=1}^{n_{in}} w_i a_i$，其方差为：

  $$\mathrm{Var}(z) = n_{in} \cdot \mathrm{Var}(w) \cdot \mathrm{Var}(a)$$

  经过 ReLU 激活函数后，由于 ReLU 会将负值置零，且假设输入 z 的分布关于0对称（均值为0），则大约一半的输出为0，另一半保持不变。因此，ReLU 输出的方差为线性输出方差的一半：

  $$\mathrm{Var}(\mathrm{ReLU}(z)) = \frac{1}{2}\mathrm{Var}(z) = \frac{1}{2} n_{in} \cdot \mathrm{Var}(w) \cdot \mathrm{Var}(a)$$

  我们希望经过 ReLU 后的输出方差与输入方差大致相等，即 $\mathrm{Var}(\mathrm{ReLU}(z)) = \mathrm{Var}(a)$，代入上式可得：

  $$\frac{1}{2} n_{in} \cdot \mathrm{Var}(w) \cdot \mathrm{Var}(a) = \mathrm{Var}(a)$$

  解得我们需要 $\mathrm{Var}(w) = 2/n_{in}$。

  因此，He 初始化将权重的方差设为 $2/n_{in}$（例如，从正态分布 $\mathcal{N}(0, \sqrt{2/n_{in}})$ 中采样），以补偿 ReLU 激活函数对方差的减半效应，从而维持前向传播中激活值方差的稳定。

方法二：批归一化（Batch Normalization, BN）

问题：随着网络深度的增加，中间层的输入分布会发生剧烈变化，称为内部协变量偏移，这会加剧梯度不稳定的问题。

解决方案：**批归一化。**

- **定义**：在每个小批量（mini-batch）数据上，对每个神经元的线性输出 $\mathbf{z}^{(l)}$ 进行归一化，然后再进行缩放和平移。
  对于一个mini-batch中的第 j 个神经元，其归一化过程为：

  $$\mu_B = \frac{1}{m}\sum_{i=1}^{m} z_{i,j}^{(l)} \qquad (\text{batch均值})$$

  $$\sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m} (z_{i,j}^{(l)} - \mu_B)^2 \qquad (\text{batch方差})$$

  $$\hat{z}_{i,j}^{(l)} = \frac{z_{i,j}^{(l)} - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \qquad (\text{归一化})$$

  $$y_{i,j}^{(l)} = \gamma_j \hat{z}_{i,j}^{(l)} + \beta_j \qquad (\text{缩放和平移})$$

  其中 γ_j 和 β_j 是可学习的参数，ϵ 是一个很小的常数用于数值稳定。

- **原理简述**：

  1. **稳定输入分布**：BN通过强制将每层的输入归一化为均值为0、方差为1的分布，极大地缓解了内部协变量偏移问题。
  2. **缓解梯度问题**：归一化后的输入 (z) 更可能落在激活函数（如Sigmoid或Tanh）的线性区域（导数较大），避免梯度饱和，从而缓解梯度消失，保障反向传播的有效性。

3. **保留模型表达能力**：BN在归一化后引入可学习的仿射参数 γ 和 β（即 $y = \gamma\hat{z} + \beta$），使网络能自主恢复任意均值和方差，既获得标准化的稳定性，又不损失表示能力。
4. **平滑损失函数景观**：研究表明（Santurkar et al., 2018），**BN的核心优势在于它显著平滑了损失函数的优化 landscape**——降低了损失函数关于权重的 Lipschitz 常数，并减小了 mini-batch 梯度的方差。这使得损失曲面更平缓、梯度方向更可靠，优化路径更稳定，收敛更快，且对学习率等超参数的敏感性降低。
5. **隐式正则化效应**：由于训练时 BN 使用 mini-batch 的统计量进行归一化，单个样本的输出会受到同 batch 其他样本的影响，引入了依赖于 batch 的噪声。这种噪声具有类似 Dropout 的正则化作用，有助于抑制过拟合；而在测试阶段，BN 切换为使用训练过程中累积的移动平均统计量，以获得确定性预测。

A2.2

Q

解释Epoch、Iteration、Batch、Step等概念及其不同，从数学的角度分析batch_size的选择依据和影响。

A

Epoch、Iteration、Batch、Step 的概念与区别

- **Epoch**：**一个 Epoch 代表模型对整个训练数据集完成了一次完整的遍历。**
- **Batch**：**一个 Batch 是训练数据集的一个子集。**由于内存限制和计算效率的考虑，我们通常不会一次性将整个数据集送入模型进行训练。相反，我们会将数据集分割成多个较小的、固定大小的块，每个块就是一个 Batch 。
- **Batch Size**：**Batch Size 是指一个 Batch 中所包含的样本数量。**这是一个非常重要的超参数
- **Iteration / Step**：**一次 Iteration/Step 对应于模型参数的一次更新。**在每次 Iteration 中，模型会执行以下操作：1) 从数据集中取出一个 Batch；2) 进行前向传播计算损失；3) 进行反向传播计算梯度；4) 使用该梯度更新模型参数。

它们之间的关系

假设你的训练集总共有 **N** 个样本，你设定的 `batch_size` 为 **B。**

- **一个 Epoch 中包含的 Iteration（或 Step）数量**为：$[\frac{N}{B}]$ 或 $[\frac{N}{B}]+1$。**是否加1取决于最后N是否整除B，如果不整除，N除以B的余数数量的样本是否作为一个不完整的Batch，是则加1否则不加1.**
- **一个 Iteration/Step 处理一个 Batch**，并更新一次模型参数 。

总结来说，Epoch 是宏观的训练周期，Batch 是数据处理的单元，Iteration/Step 是参数更新的动作。一个 Epoch 由多个 Iteration 组成，每个 Iteration 处理一个 Batch。

从数学角度分析 `batch_size` 的选择依据和影响

`batch_size` 的选择深刻影响着模型的收敛速度、收敛稳定性、泛化能力以及计算效率。我们可以从梯度估计的方差和偏差角度进行数学分析。

1. 梯度估计的视角

在训练神经网络时，我们的目标是最小化整个数据集上的损失函数：

$$\mathcal{L}(\theta) = \frac{1}{N}\sum_{i=1}^{N} \ell(f(x_i;\theta), y_i)$$

其中 θ 是模型参数，ℓ 是单个样本的损失。

- **全批量梯度下降 (Full-batch GD)**：`batch_size = N`
  - 梯度计算为：$\nabla\mathcal{L}(\theta) = \frac{1}{N}\sum_{i=1}^{N}\nabla\ell_i(\theta)$
  - **特点**：这是一个**无偏且方差为零**的梯度估计。**方向非常准确，但计算开销巨大，且容易陷入尖锐的局部极小值，泛化性能可能较差。**
- **随机梯度下降 (SGD)**：`batch_size = 1`
  - 梯度计算为：$\nabla\hat{\mathcal{L}}(\theta) = \nabla\ell_i(\theta)$，其中 i 是随机选取的样本。
  - **特点**：这是一个**无偏但方差极大**的梯度估计。**虽然每次更新计算很快，但收敛过程会剧烈震荡，需要很小的学习率。**
- **小批量梯度下降 (Mini-batch SGD)**：`1 < batch_size < N`
  - 梯度计算为：$\nabla\hat{\mathcal{L}}_B(\theta) = \frac{1}{B}\sum_{i\in\mathcal{B}}\nabla\ell_i(\theta)$，其中 $\mathcal{B}$ 是随机选取的大小为 B 的批次。
  - **特点**：这是前两者的折中。它仍然是一个**无偏估计**（因为 $\mathbb{E}[\nabla\hat{\mathcal{L}}_B(\theta)] = \nabla\mathcal{L}(\theta)$），但其**方差会随着 `batch_size` 的增大而减小。**

我们可以计算 Mini-batch 梯度的方差。假设各个样本的梯度是独立同分布的，且单个样本梯度的方差为 σ^2，那么 Mini-batch 梯度的方差为：

$$\mathrm{Var}(\nabla\hat{\mathcal{L}}_B(\theta)) = \frac{\sigma^2}{B}$$

这个公式揭示了核心权衡：

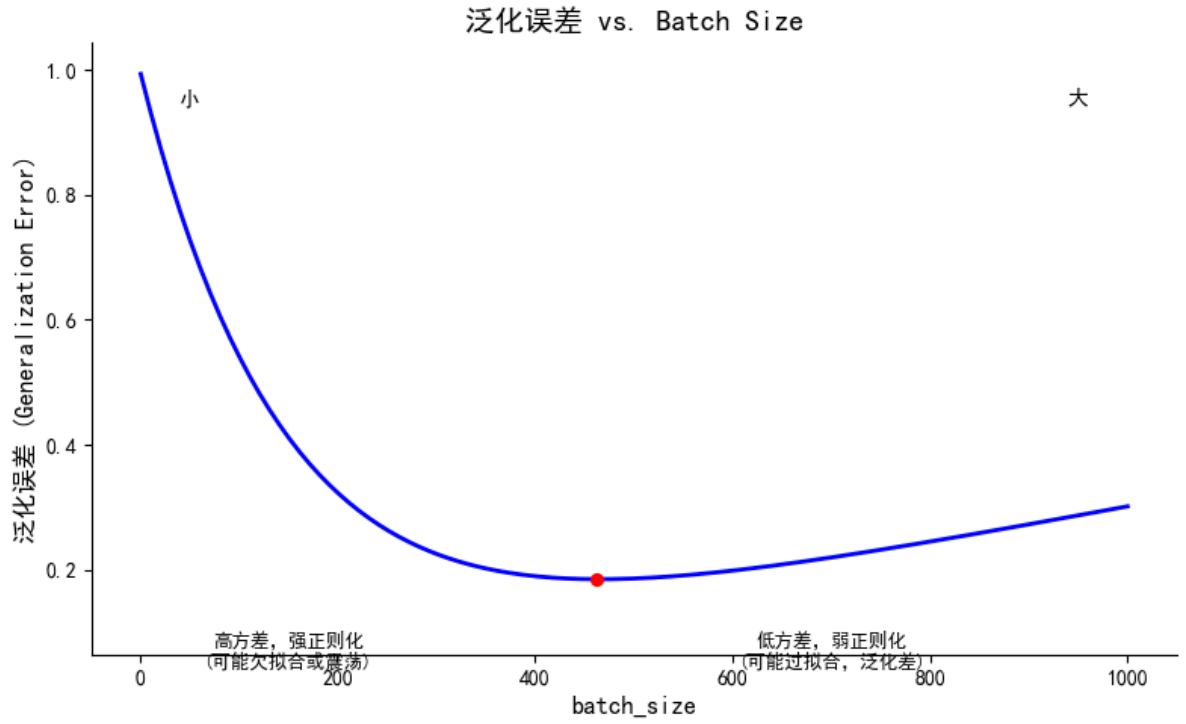
- `batch_size` **越大，梯度估计的方差越小，更新方向越稳定、越接近真实梯度，因此可以使用更大的学习率，单次迭代的收敛效果更好。但计算开销大，且可能收敛到泛化能力较差的平坦极小值。**
- `batch_size` **越小，梯度估计的方差越大，更新路径充满噪声。这种噪声在某种程度上可以看作是一种隐式的正则化，有助于模型跳出尖锐的局部极小值，找到泛化能力更强的平坦极小值。但收敛过程不稳定，需要更小的学习率，总的收敛步数可能更多。**

2. 对收敛性和泛化的影响

在一定范围内，**将 `batch_size` 增大 k 倍，同时将学习率也增大 k 倍**，可以在保持模型动态行为相似的同时，显著加速训练，因为更大的批次可以更好地利用GPU的并行计算能力。

然而，这种缩放并非没有上限。存在一个临界值。**当 `batch_size` 超过这个临界值后，再增大批次对减少梯度方差的效益会急剧下降，训练速度的提升也会停滞。此时，继续增大 `batch_size` 不仅浪费计算资源，还可能因为梯度噪声过小而损害模型的泛化能力 。**

下图可以直观地展示这种权衡关系：



理想的 `batch_size` 通常位于这个"U"形曲线的底部附近，它在收敛速度和泛化性能之间取得了最佳平衡。

3. 总结

综合以上分析，我给出下面的表格来作为总结：

表 1：Epoch、Iteration、Batch 与 Step 的概念对比

术语	定义	性质	是否触发参数更新	与数据集的关系	备注
Epoch	模型对整个训练数据集完成一次完整遍历	宏观训练周期	否	涉及全部 N 个样本	常用于衡量训练进度和调整训练时长
Batch	训练数据集的一个子集，用于单次前向/反向传播	数据处理单元	否（本身是数据）	大小由 `batch_size` 决定	是数据的物理分组
Batch Size	一个 Batch 中包含的样本数（超参数）	整数（如 32, 64）	—	决定内存占用和并行效率	通常取 2 的幂次以优化 GPU 利用率
Iteration / Step	一次参数更新过程：取一个 Batch → 前向 → 计算损失 → 反向 → 更新参数	参数更新动作	是	每次处理一个 Batch（B 个样本）	在多数框架中，Step 与 Iteration 可互换使用

表 2：Batch Size 的选择依据与影响分析

维度	小 Batch Size（如 1–32）	大 Batch Size（如 256–4096+）	理论/实践依据
梯度估计	无偏但方差大，更新噪声强	无偏且方差小($\mathrm{Var} \propto \frac{1}{B}$)，方向稳定	梯度方差公式：$\mathrm{Var}(\nabla \hat{\mathcal{L}}_B) = \frac{\sigma^2}{B}$
收敛稳定性	更新路径震荡剧烈，需较小学习率	更新平滑，可使用较大学习率	大 Batch 收敛曲线更平滑
泛化能力	噪声具有隐式正则化作用，易找到泛化性好的平坦极小值	易收敛到尖锐极小值，可能泛化性能下降	小 Batch 有助于防止过拟合
计算效率	GPU 利用率低，吞吐量小，训练速度慢	充分利用 GPU 并行能力，训练加速明显	大 Batch 减少迭代次数，提升吞吐
内存占用	显存需求低，适合资源受限设备	显存需求高，可能超出硬件限制	Batch Size 受 GPU 显存约束
学习率缩放策略	不适用	线性缩放规则：Batch Size 增大 (k) 倍，学习率也增大 (k) 倍（在临界值内有效）	保持梯度更新动态一致
适用场景	小数据集、强调泛化、资源有限	大数据集、追求训练速度、有充足算力	实践中常选 32–256 作为起点
潜在风险	收敛慢、不稳定、可能不收敛	泛化差、收敛到次优解、资源浪费	超过临界 Batch Size 后收益递减

选择 `batch_size` 时应考虑以下因素：

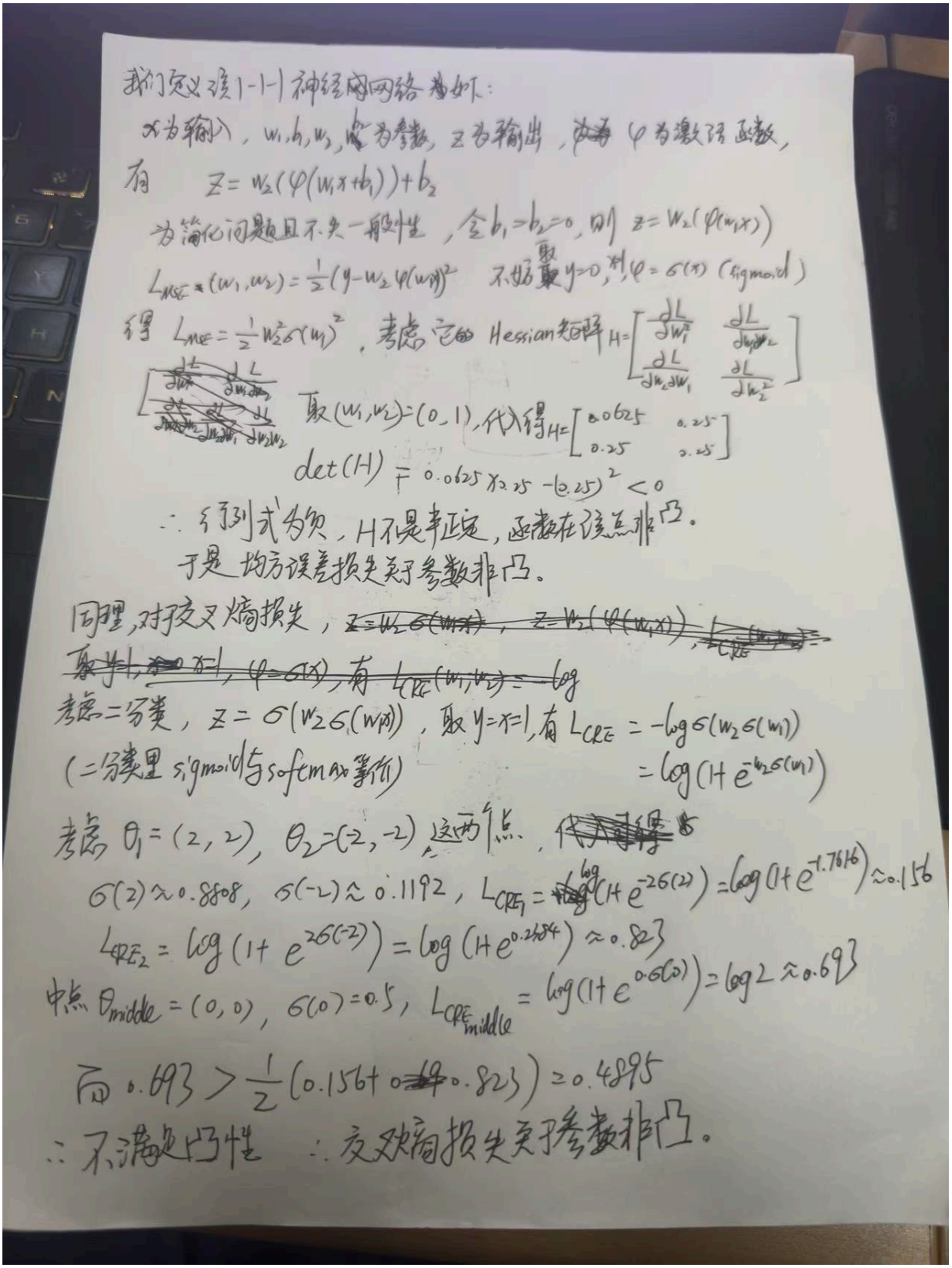
- **硬件限制**：首要限制是GPU/TPU的显存。`batch_size` 越大，显存占用越高。
- **收敛速度 vs. 泛化**：如果追求最快的训练速度且硬件允许，可以尝试较大的 `batch_size`（如256, 512, 1024），并配合相应的学习率缩放。如果更关注模型的最终性能（泛化能力），较小的 `batch_size`（如32, 64）往往是更安全的选择，但也会带来训练速度较慢的问题。
- **问题特性**：对于某些任务（如强化学习、小样本学习），小 `batch_size` 带来的噪声可能是有益的。

A2.3

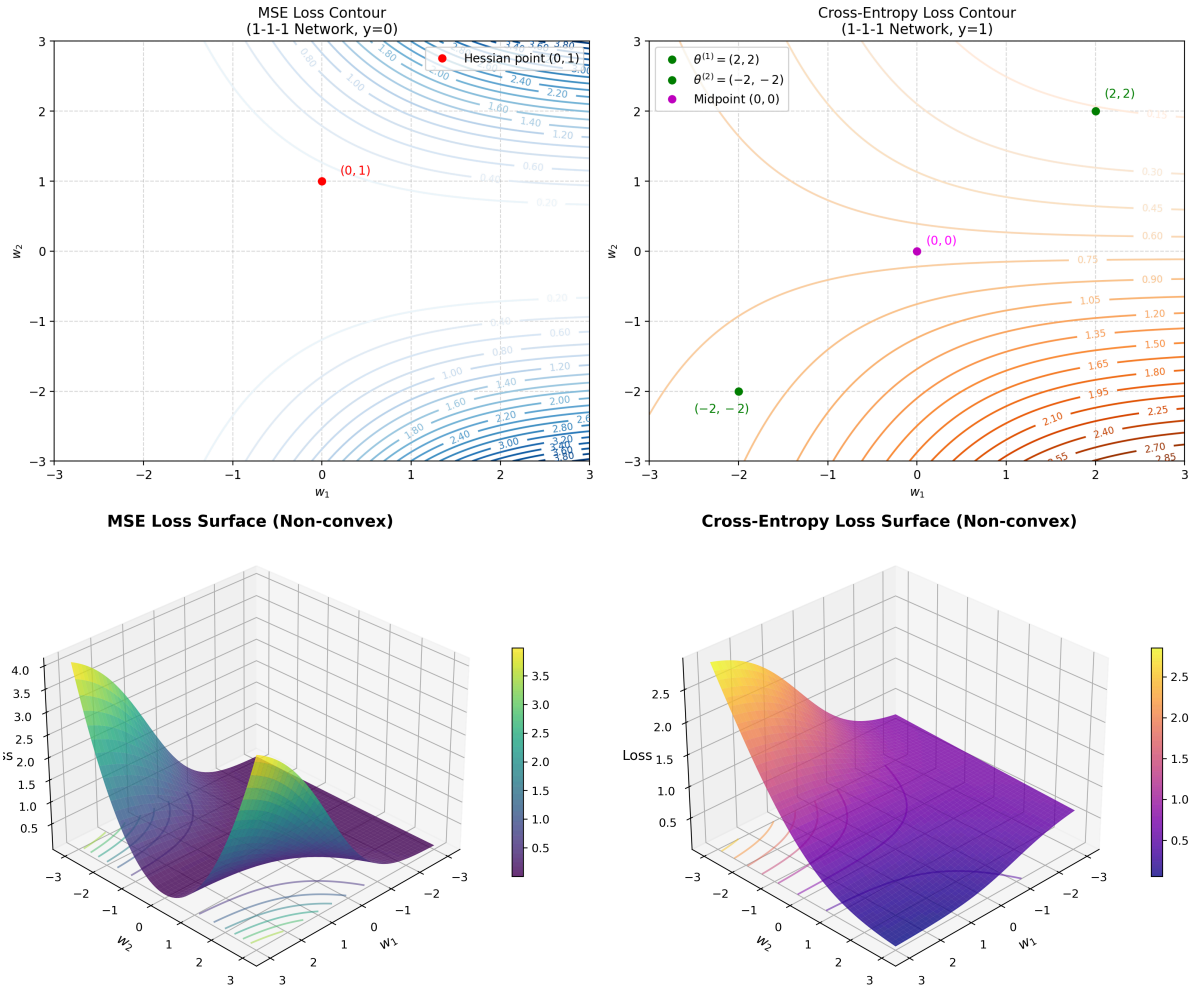
Q

以一个简单的1-1-1结构的两层神经网络为例，分别采用均方误差损失函数和交叉熵损失函数，说明这两种函数关于参数的非凸性（可作图示意和说明）。

A



下图是上图的推导的图像



均方误差损失的非凸性：在1-1-1神经网络中，由于Sigmoid激活函数的非线性作用，MSE损失函数 ($\mathcal{L} = \frac{1}{2}(w_2 \cdot \sigma(w_1))^2$) 关于参数 (w_1, w_2)呈现出明显的非凸特性。从三维曲面和等高线图可见，损失地形不是单一的"碗状"结构，而是出现了扭曲和非对称的复杂形态。特别是固定w_2时，损失随 w_1 变化呈现S形曲线，这种非线性相互作用导致Hessian矩阵不定，使得函数在各个方向的曲率不一致，无法满足凸性定义。

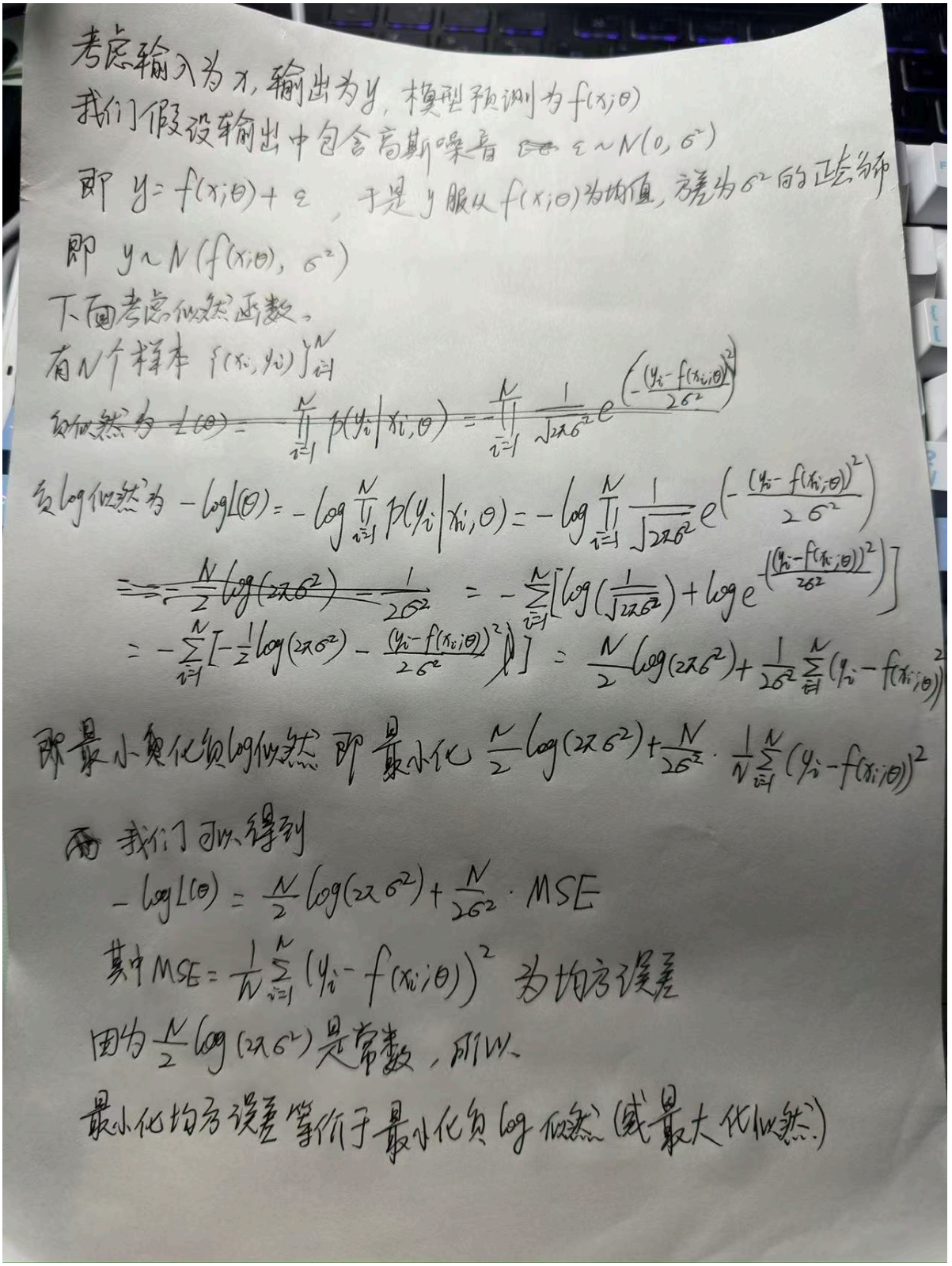
交叉熵损失的非凸性：交叉熵损失 ($\mathcal{L} = -\log(\sigma(w_2 \cdot \sigma(w_1)))$) 由于双重Sigmoid函数的嵌套组合，产生了更复杂的非凸地形。曲面显示出多个"盆地"结构和明显的非对称性，等高线呈现扭曲的非椭圆模式。在 $w_1 = 0$ 附近区域，损失函数沿不同参数方向表现出截然不同的曲率特性，这种Hessian矩阵特征值符号不一致的现象直接证明了函数的非凸性。尽管非凸性使优化变得复杂，但实践中我们只需找到性能良好的局部最优解即可。

A2.4

Q

试推导：在回归问题中，假设输出中包含高斯噪音，则最小化均方误差等价于最小化负log似然（或最大化似然）。

A

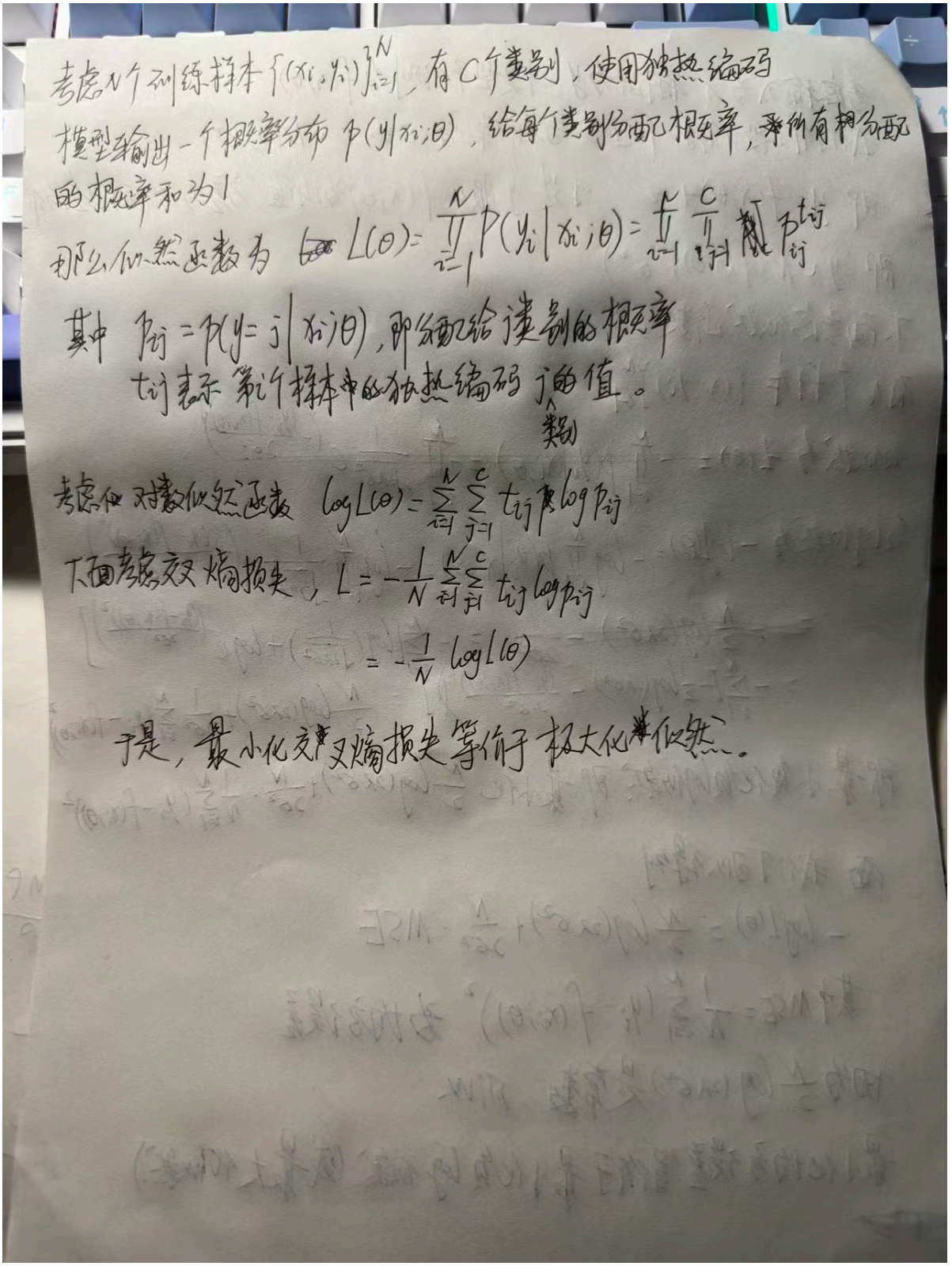


A2.5

Q

尝试推导：在分类问题中，最小化交叉熵损失等价于极大化似然。

A



A2.6

Q

分析L1正则化和L2正则化的原理，以及为什么L1正则化倾向于得到稀疏解、为什么L2正则化倾向于得到平滑的解。

A

正则化基本原理

在机器学习中，尤其是在线性模型或参数量巨大的深度学习模型中，我们常常面临过拟合的问题

L1/L2正则化是一种通过向原始损失函数添加一个惩罚项来控制模型复杂度的技术。**核心思想是：在拟合数据的同时，对模型参数的“大小”或“复杂度”进行约束，从而提高模型的泛化能力。**

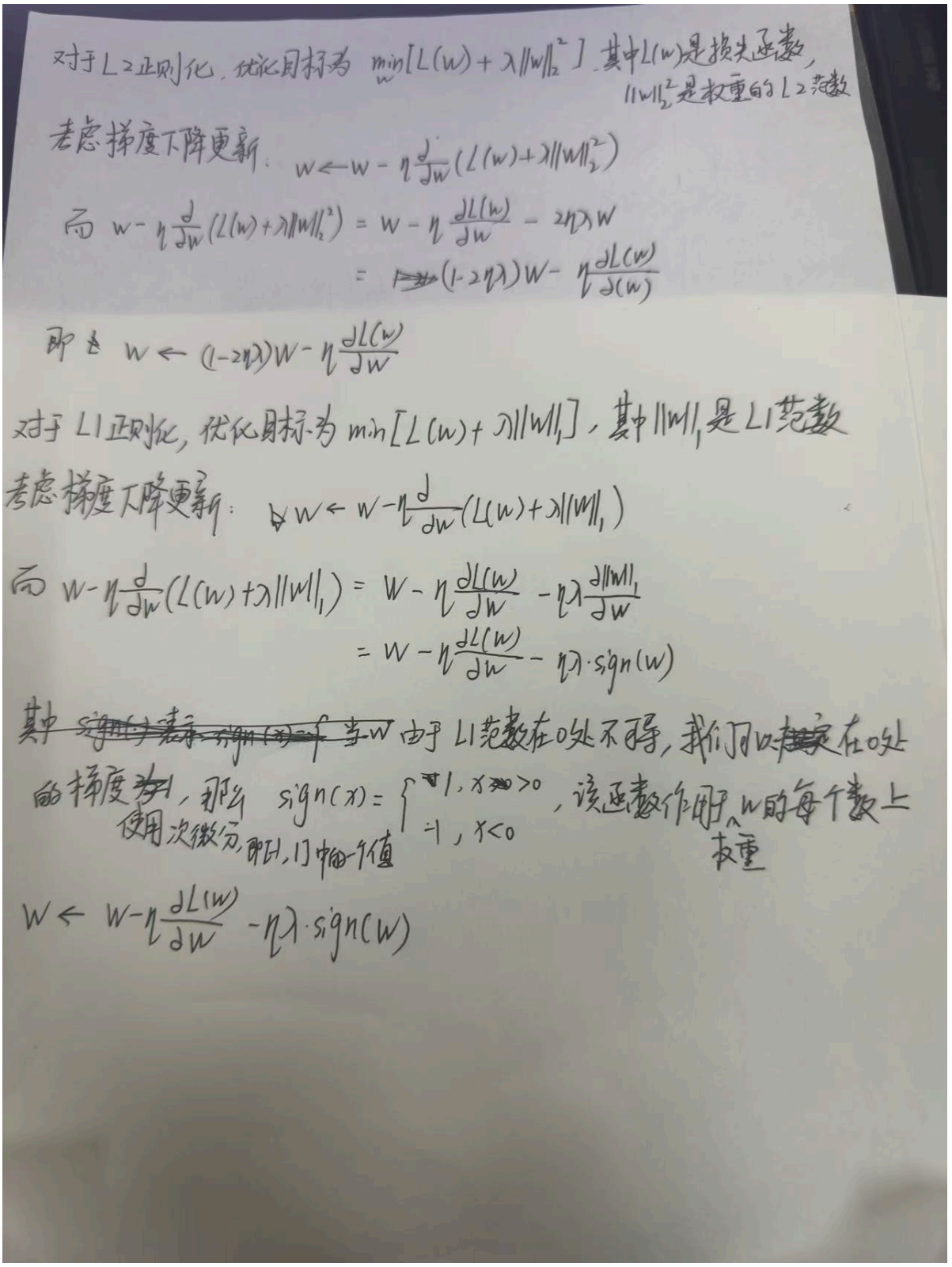
L1/L2正则化是权重衰退的方法，可以写成公式

$$\min_{\mathbf{w}} \left[\mathcal{L}(\mathbf{w}) + \lambda \Omega(\mathbf{w})\right]$$

其中：

- $\mathbf{w} \in \mathbb{R}^d$ 是模型的参数向量。
- $\mathcal{L}(\mathbf{w})$ 是原始的损失函数（如MSE、交叉熵等），衡量模型对训练数据的拟合程度。
- $\Omega(\mathbf{w})$ 是正则化项（或称惩罚项），衡量模型的复杂度。
- $\lambda \geq 0$ 是正则化强度（或称正则化系数），用于平衡拟合数据和控制复杂度这两个目标。λ 越大，对模型复杂度的惩罚越重。

数学推导



L2正则化倾向于得到平滑的解

由上面的数学推导可以得到这个式子

$$w_j \leftarrow (1 - 2\eta\lambda)w_j - \eta \frac{\partial \mathcal{L}(\mathbf{w})}{\partial w_j}$$

解释：

- **与无正则化的更新相比，L2正则化在每次更新后都对权重 w_j 进行了一个线性缩放（乘以一个小于1的因子 $(1 - 2\eta\lambda)$）。**

- **这个操作会持续地、均匀地将所有权重向零拉近，但永远不会精确地将任何权重变为零（除非它初始就是零）。**
- **结果是，所有权重的绝对值都会变得更小，并且大小相对均衡，没有特别大的权重。这种解被称为“平滑的”或“分散的”，因为它避免了模型过度依赖某几个特定的特征。**

几何解释（约束优化视角）

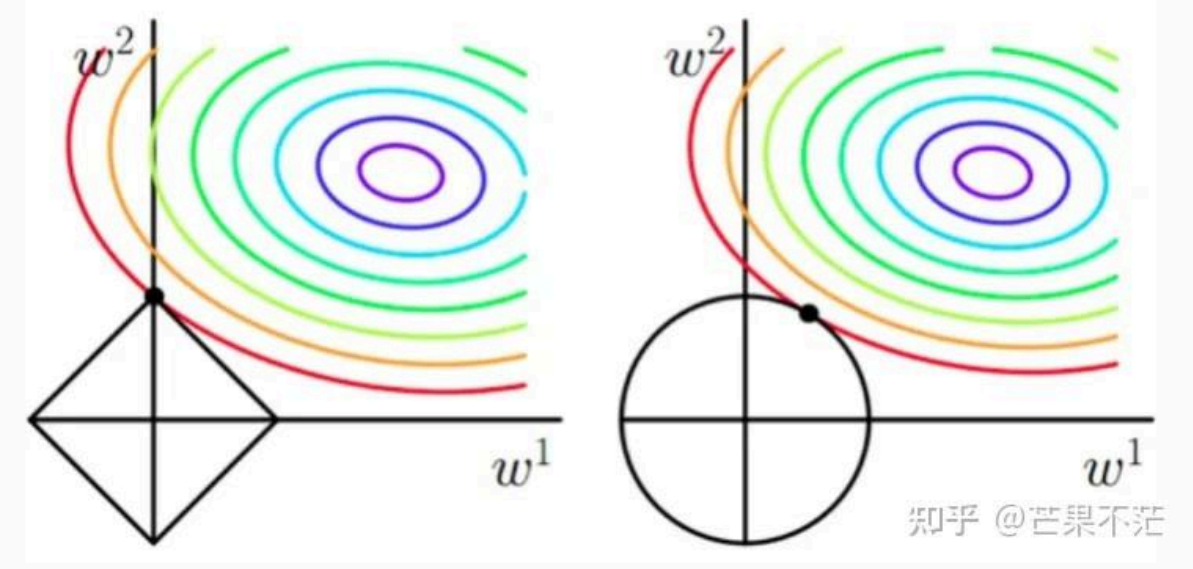
我们可以将带正则化的优化问题等价地看作一个约束优化问题：

$$\begin{aligned} \min_{\mathbf{w}} \quad & \mathcal{L}(\mathbf{w}) \\ \text{subject to} \quad & \|\mathbf{w}\|_2^2 \leq C \end{aligned}$$

其中 C 是一个与 λ 相关的常数（λ 越大，C 越小）。

- **损失函数的等高线**：$\mathcal{L}(\mathbf{w})$ 的等高线通常是椭圆形的（对于二次损失函数）或光滑的凸形。
- **L2约束区域**：$\|\mathbf{w}\|_2^2 \leq C$ 定义了一个圆形（在2D）或球形（在高维）的区域。

最优解出现在损失函数的等高线与约束圆首次接触的点。由于圆是光滑的，且在所有方向上都是对称的，这个接触点几乎不可能恰好落在坐标轴上（即 $w_1 = 0$ 或 $w_2 = 0$）。因此，解的两个分量通常都是非零的。



图示说明（右图）：
一个二维平面，原点为中心画一个圆（L2约束）。再画一组椭圆（损失函数等高线），其最小值点在圆外。最优解是椭圆与圆相切的点。这个切点几乎总是在圆的“弧”上，而不是在坐标轴的交点上。

L1正则化倾向于得到稀疏解

由上面的数学推导可以得到这个式子

$$w_j \leftarrow w_j - \eta \left(\frac{\partial \mathcal{L}(\mathbf{w})}{\partial w_j} + \lambda \cdot \text{sign}(w_j) \right)$$

其中 $\text{sign}(w_j)$ 是符号函数。

解释：

- L1正则化的梯度是一个常数（$\pm\lambda$），与权重 w_j 的大小无关。
- **这意味着，无论一个权重当前的值是多少（只要不为零），L1正则化都会以固定的力度将其推向零。**

- **产生稀疏性的核心机制：当无正则化损失 $\mathcal{L}(\mathbf{w})$ 的梯度 $\frac{\partial \mathcal{L}}{\partial w_j}$ 的绝对值小于 λ 时，L1正则化的“拉力”会完全抵消甚至超过数据驱动的梯度，从而将权重精确地推到零。一旦权重为零，在后续的更新中，只要 $\left|\frac{\partial \mathcal{L}}{\partial w_j}\right| \leq \lambda$，它就会保持为零。这个过程实现了特征选择。**

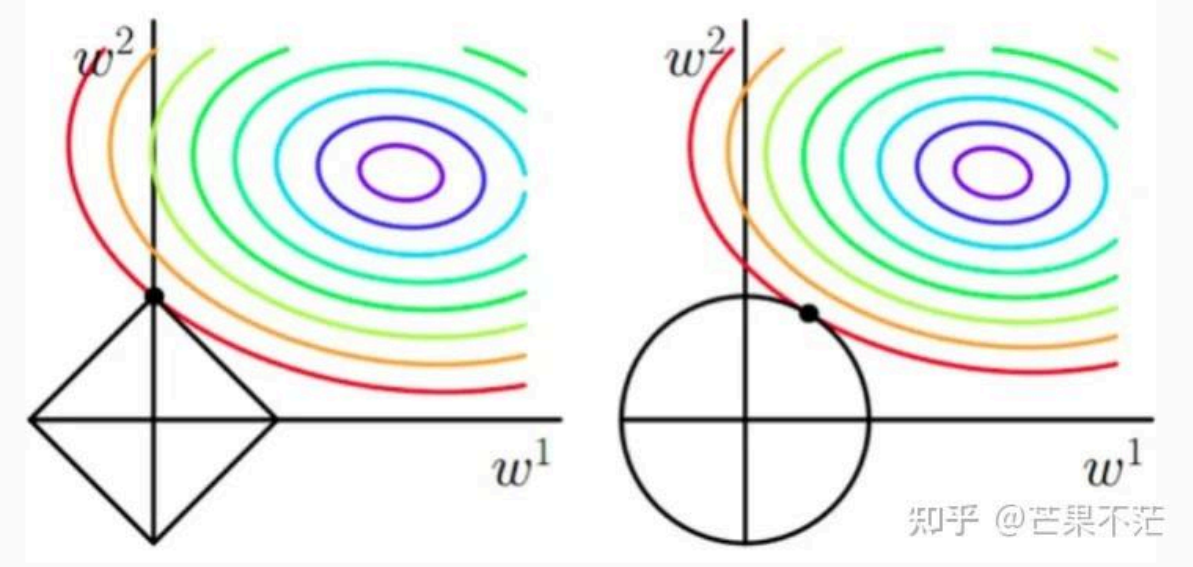
几何解释（约束优化视角）

同样，L1正则化可以看作以下约束优化问题：

$$\begin{aligned} \min_{\mathbf{w}} \quad & \mathcal{L}(\mathbf{w}) \\ \text{subject to} \quad & \|\mathbf{w}\|_1 \leq C \end{aligned}$$

- **L1约束区域**：$\|\mathbf{w}\|_1 \leq C$ 在二维平面上定义了一个菱形，其顶点位于坐标轴上。

最优解同样出现在损失函数等高线与约束菱形首次接触的点。由于菱形有尖锐的角（位于坐标轴上），损失函数的等高线非常容易与这些角相交。如果与角相交，意味着一个或多个权重精确为零，从而产生稀疏解。



图示说明（左图）：
在同一个二维平面上，用菱形（L1约束）代替之前的圆。这个菱形有四个尖角，分别在 $(C,0),(-C,0),(0,C),(0,-C)$。现在，那组椭圆（损失函数等高线）与菱形首次接触的点，有很大概率会落在菱形的边上或尖角上。一旦接触点在边上，例如在 $w_2 = 0$ 的边上，那么最优解就是 $w_2 = 0$，实现了稀疏性。

总结与对比

特性	L1 正则化 (Lasso)	L2 正则化 (Ridge)
正则化项	$\lambda \sum_j \|w_j\|$	$\lambda \sum_j w_j^2$
梯度/次梯度	常数 ($\pm\lambda$)，与 w_j 大小无关	与 w_j 成正比 ($2\lambda w_j$)
解的特性	稀疏：倾向于将不重要的特征权重精确压缩到 0，实现自动特征选择。	平滑/分散：将所有权重均匀地向 0收缩，但不会精确为0。所有特征都会被保留。
几何约束形状	菱形（有尖角）	圆形（光滑）

特性	L1 正则化 (Lasso)	L2 正则化 (Ridge)
适用场景	当怀疑只有少数特征是真正相关的时候；需要模型可解释性（通过特征选择）。	当所有特征都可能包含有用信息，但希望防止任何单个特征权重过大时；通常能提供更好的预测性能。

结论：

- **L1的稀疏性源于其非光滑的、有尖角的约束区域以及其恒定的梯度惩罚，这使得优化解很容易落在坐标轴上，从而产生零权重。**
- **L2的平滑性源于其光滑的、圆形的约束区域以及其与权重大小成正比的梯度惩罚，这使得所有权重被均匀地缩小，但不会被完全消除。**

A2.7

Q

分析Batch normalization对参数优化起到什么作用、如何起到这种作用。

A

Batch Normalization的设计动机

在训练深度神经网络时，一个长期存在的挑战是内部协变量偏移（Internal Covariate Shift, ICS）。其含义是：在网络训练过程中，由于前面层的参数不断更新，导致后面层的输入分布也随之发生剧烈变化。

这种现象带来了两个主要问题：

1. **训练不稳定**：每一层都需要不断地去适应其输入分布的变化，这使得训练过程变得困难和不稳定。
2. **梯度问题加剧**：如果某一层的输入分布偏移到了激活函数（如Sigmoid、Tanh）的饱和区（梯度接近0的区域），就会引发严重的梯度消失问题，阻碍深层网络的训练。

Batch Normalization的核心思想就是通过归一化每一层的输入，来缓解训练不稳定和梯度消失问题，从而稳定并加速训练过程。

Batch Normalization的数学定义与操作

Batch Normalization被应用在网络的线性变换之后、非线性激活函数之前。对于一个mini-batch中的某一层，我们详细描述其操作。

前向传播中的BN操作

假设我们有一个mini-batch，包含 m 个样本。对于该层的第 j 个神经元（或特征通道），其线性输出构成一个向量：

$$\mathbf{z}^{(j)} = [z_1^{(j)}, z_2^{(j)}, \ldots, z_m^{(j)}]^T$$

其中 $z_i^{(j)}$ 是第 i 个样本在该神经元上的输出。

步骤1：计算批统计量
BN首先计算该mini-batch上该神经元输出的**均值**和**方差**：

$$\mu_{\mathcal{B}}^{(j)} = \frac{1}{m}\sum_{i=1}^{m} z_i^{(j)} \qquad \text{(批均值)}$$

$$(\sigma_{\mathcal{B}}^{(j)})^2 = \frac{1}{m}\sum_{i=1}^{m}(z_i^{(j)} - \mu_{\mathcal{B}}^{(j)})^2 \qquad \text{(批方差)}$$

步骤2：归一化
使用计算出的均值和方差，对该mini-batch中所有样本的输出进行归一化，得到零均值、单位方差的激活值：

$$\hat{z}_i^{(j)} = \frac{z_i^{(j)} - \mu_{\mathcal{B}}^{(j)}}{\sqrt{(\sigma_{\mathcal{B}}^{(j)})^2 + \epsilon}}$$

其中 ϵ 是一个很小的常数（如 10^{-5}），用于防止除零错误，并确保数值稳定性。

步骤3：缩放与平移
如果直接将归一化后的 $\hat{z}_i^{(j)}$ 送入激活函数，会限制网络的表达能力（例如，无法表示恒等变换）。因此，BN引入了两个可学习的参数 $\gamma^{(j)}$ 和 $\beta^{(j)}$，对归一化后的值进行仿射变换：

$$y_i^{(j)} = \gamma^{(j)}\hat{z}_i^{(j)} + \beta^{(j)}$$

- 当 $\gamma^{(j)} = \sqrt{(\sigma_{\mathcal{B}}^{(j)})^2 + \epsilon}$ 且 $\beta^{(j)} = \mu_{\mathcal{B}}^{(j)}$ 时，该变换可以完全恢复原始的 $z_i^{(j)}$。
- 网络在训练过程中会自动学习 γ 和 β 的最优值，从而在“归一化带来的稳定性”和“保留网络表达能力”之间取得平衡。

最终，$y_i^{(j)}$ 被送入下一层的激活函数（如ReLU）。

BN如何作用于参数优化

1. **稳定输入分布**：BN通过强制将每层的输入归一化为均值为0、方差为1的分布，极大地缓解了内部协变量偏移问题。
2. **缓解梯度问题**：归一化后的输入 (z) 更可能落在激活函数（如Sigmoid或Tanh）的线性区域（导数较大），避免梯度饱和，从而缓解梯度消失，保障反向传播的有效性。
3. **保留模型表达能力**：BN在归一化后引入可学习的仿射参数 γ 和 β（即 $y = \gamma\hat{z} + \beta$），使网络能自主恢复任意均值和方差，既获得标准化的稳定性，又不损失表示能力。
4. **平滑损失函数景观**：研究表明（Santurkar et al., 2018），**BN的核心优势在于它显著平滑了损失函数的优化 landscape**——降低了损失函数关于权重的 Lipschitz 常数，并减小了 mini-batch 梯度的方差。这使得损失曲面更平缓、梯度方向更可靠，优化路径更稳定，收敛更快，且对学习率等超参数的敏感性降低。
5. **隐式正则化效应**：由于训练时 BN 使用 mini-batch 的统计量进行归一化，单个样本的输出会受到同 batch 其他样本的影响，引入了依赖于 batch 的噪声。这种噪声具有类似 Dropout 的正则化作用，有助于抑制过拟合；而在测试阶段，BN 切换为使用训练过程中累积的移动平均统计量，以获得确定性预测。

A2.8

Q

从工作原理角度，对比经典的随机梯度下降（SGD）、带动量的SGD（SGD with Momentum）、以及Adam优化器。并说明“动量”项是如何帮助SGD在相关曲面上加速并减少振荡的，Adam在泛化性方面存在哪些问题。

A

优化器工作原理对比

我们考虑一个通用的优化问题：$\min_\theta \mathcal{L}(\theta)$，其中 θ 是模型参数，$\mathcal{L}$ 是损失函数。所有优化器都基于梯度 $\nabla_\theta \mathcal{L}(\theta)$ 来更新参数。

经典随机梯度下降（SGD）

工作原理：SGD在每次迭代中，使用一个mini-batch数据计算损失函数的梯度，并沿梯度的反方向更新参数。

数学公式：

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta \mathcal{L}_t(\theta_t)$$

其中：

- θ_t 是第 t 次迭代时的参数。
- η 是学习率。
- $\nabla_\theta \mathcal{L}_t(\theta_t)$ 是在第 t 个mini-batch上计算的随机梯度。

特点：

- **简单直接**，是所有优化器的基础。
- **更新方向完全由当前梯度决定**，路径非常“嘈杂”，容易在损失函数的狭窄谷底或鞍点附近产生剧烈振荡，收敛速度慢。

带动量的SGD（SGD with Momentum）

工作原理：动量方法引入了一个“速度”（velocity）变量 $\mathbf{v}$，它累积了过去梯度的指数移动平均。参数更新不仅取决于当前梯度，还取决于历史梯度的累积方向。

数学公式：

$$\begin{aligned} \mathbf{v}_{t+1} &= \mu \mathbf{v}_t - \eta \nabla_\theta \mathcal{L}_t(\theta_t) \\ \theta_{t+1} &= \theta_t + \mathbf{v}_{t+1} \end{aligned}$$

其中 $\mu \in [0, 1)$ 是动量系数（通常设为0.9）。

特点：

- **引入惯性机制**，通过累积历史梯度方向形成“速度”，使参数更新具有动量。
- **有效抑制振荡**：在损失函数的狭窄谷底或平坦区域，动量能平滑更新路径，减少 SGD 的剧烈抖动。
- **加速收敛**：在一致的方向上，历史梯度不断累积，使更新步长增大，从而更快地穿越平坦区域、逃离鞍点。
- **对学习率更鲁棒**：由于动量提供了方向稳定性，模型对学习率的选择不那么敏感，训练过程更稳定。

Adam优化器

工作原理：Adam（Adaptive Moment Estimation）结合了动量的思想和自适应学习率的思想。它为每个参数维护两个状态变量：一阶矩（梯度的均值）和二阶矩（梯度平方的均值），并用它们来动态调整每个参数的学习率。

数学公式：

$$
\begin{aligned}
\mathbf{m}_{t+1} &= \beta_1 \mathbf{m}_t + (1-\beta_1)\nabla_t && \text{(一阶矩估计)} \\
\mathbf{v}_{t+1} &= \beta_2 \mathbf{v}_t + (1-\beta_2)\nabla_t^2 && \text{(二阶矩估计)} \\
\hat{\mathbf{m}}_{t+1} &= \mathbf{m}_{t+1} / (1-\beta_1^{t+1}) && \text{(偏差修正)} \\
\hat{\mathbf{v}}_{t+1} &= \mathbf{v}_{t+1} / (1-\beta_2^{t+1}) && \text{(偏差修正)} \\
\theta_{t+1} &= \theta_t - \eta \frac{\hat{\mathbf{m}}_{t+1}}{\sqrt{\hat{\mathbf{v}}_{t+1}} + \epsilon}
\end{aligned}
$$

其中 $\nabla_t = \nabla_\theta \mathcal{L}_t(\theta_t)$，$\beta_1$（通常0.9）和 β_2（通常0.999）是衰减率，ϵ 是小常数。

特点：

- **自适应学习率**：对于梯度变化剧烈的参数，$\hat{\mathbf{v}}$ 较大，其有效学习率 $\eta/\sqrt{\hat{\mathbf{v}}}$ 会变小；对于梯度稳定的参数，有效学习率会变大。这使得Adam对不同参数能进行更精细的调整。
- **收敛速度快**：在许多任务上，Adam能比SGD和Momentum更快地收敛到一个不错的解。

以下是一个对比 **SGD**、**SGD with Momentum** 和 **Adam** 三种优化器的核心特性的表格：

特性	SGD	SGD with Momentum	Adam
核心思想	沿当前梯度反方向更新	引入速度变量，累积历史梯度方向	结合动量（一阶矩）与自适应学习率（二阶矩）
是否使用历史梯度	仅用当前梯度	指数移动平均累积历史梯度	同时维护一阶矩（均值）和二阶矩（平方均值）
学习率	全局固定（或手动衰减）	全局固定，但有效步长受动量放大	自适应：每个参数有独立的有效学习率
收敛速度	慢，尤其在平坦区域或病态曲面	比 SGD 快，能加速穿越平坦区域	通常最快，尤其在初期
训练稳定性	噪声大，易振荡	更平滑，减少振荡	非常稳定，对超参相对鲁棒
超参数敏感性	对学习率敏感	对学习率和动量系数均敏感	对 $\beta_1, \beta_2, \epsilon$ 不太敏感，但学习率仍需调
内存开销	低（仅存参数）	中（额外存速度向量 $\mathbf{v}$）	高（额外存 $\mathbf{m}, \mathbf{v}$）
典型应用场景	理论分析、简单任务、配合学习率调度	图像分类、传统深度学习任务	NLP、Transformer、大多数现代深度学习任务

“动量”如何加速并减少振荡

类似一个球从山坡上滚下。**SGD就像每次只根据当前位置的坡度给球一个瞬时的推力，球的运动轨迹会非常不规则。而带动量的SGD则像给球赋予了惯性。**

- **加速**：当优化路径方向一致时（例如，沿着一个宽阔的山谷下降），历史梯度 $\mathbf{v}_t$ 与当前梯度方向相同。动量项 $\mu\mathbf{v}_t$ 会放大更新步长，使优化器能够更快地穿越平坦区域。
- **减少振荡**：当优化路径需要穿越狭窄的山谷时，SGD会在山谷两侧来回震荡。而动量方法中，垂直于山谷方向的梯度分量会频繁改变符号，导致其在速度 $\mathbf{v}$ 中相互抵消。而沿着山谷方向的梯度分量符号一致，会在 $\mathbf{v}$ 中不断累积。最终，速度 $\mathbf{v}$ 的方向会稳定地指向山谷底部，从而有效抑制了横向振荡，使路径更加平滑。

从数学上看，速度 $\mathbf{v}_{t+1}$ 可以展开为：

$$\mathbf{v}_{t+1} = -\eta \left(\nabla_t + \mu \nabla_{t-1} + \mu^2 \nabla_{t-2} + \dots \right)$$

这是一个指数衰减的过去梯度加权和。近期的梯度权重更大，但历史信息也被保留，起到了平滑和导向的作用。

Adam在泛化性方面存在哪些问题

尽管Adam收敛速度快，但大量实践和研究表明，使用Adam训练的模型，其最终的泛化性能常常不如精心调优的SGD（或SGD with Momentum）。

核心原因分析：

1. **自适应学习率破坏了SGD的隐式正则化效应**：
    - SGD的“嘈杂”更新路径本身具有隐式的正则化效果。这种噪声有助于模型逃离尖锐的局部极小值，而倾向于收敛到更平坦的极小值。平坦的极小值通常具有更好的泛化能力。
    - Adam通过自适应学习率平滑了更新过程，减少了这种有益的噪声，导致模型更容易收敛到尖锐的极小值，从而损害了泛化性。
2. **对梯度方差的过度补偿**：
    - Adam的二阶矩估计 $\hat{\mathbf{v}}$ 会抑制那些具有高方差但长期来看方向正确的梯度分量。这可能导致优化器过早地放弃一些对泛化至关重要的特征方向。
3. **收敛点的差异**：
    - 理论分析表明，SGD和Adam在非凸优化问题中可能会收敛到不同类型的临界点。SGD的动态特性使其更有可能找到泛化性更好的解。
4. **训练后期有效学习率过小，缺乏精细调优能力**：
    - Adam的有效学习率随训练持续衰减，难以通过学习率调度进一步优化泛化；而SGD配合学习率衰减可显著提升测试性能。

实践：在追求最佳泛化性能的任务（如计算机视觉、自然语言处理的SOTA模型）中，通常采用以下策略：

- **使用Adam进行快速预训练或初步探索。**
- **在训练后期切换到SGD with Momentum进行调整，以获得更好的最终性能。**

A2.9

Q

深度神经网络通过堆叠大量简单的非线性单元获得了将强的学习能力，你认为当前深度学习模型与大脑的学习机制之间，最大的相似之处和不同之处分别是什么？

A

最大的相似之处：分层表征学习

这是深度学习与大脑最核心、最根本的相似点。

- **在大脑中**：感觉信息（如视觉）的处理是高度分层的。例如，在视觉皮层中，初级区域（如V1）的神经元对简单的局部特征（如边缘、朝向）敏感；信息逐级传递到更高级的区域（如V2, V4, IT），神经元的感受野越来越大，能够编码越来越复杂和抽象的特征（如形状、物体部件，最终到完整物体甚至面孔）。这种“从简单到复杂”的层级加工，使得大脑能够高效地理解复杂的感官输入。
- **在深度神经网络中**：卷积神经网络（CNN）等模型完美地体现了这一思想。浅层卷积核通常学习到类似滤波器的简单模式（边缘、纹理）；随着网络深度的增加，中层开始组合这些简单模式，识别出更复杂的局部结构（如眼睛、轮子）；而深层则能够整合这些信息，形成对整个场景或物体的全局、语义级别的理解（如“一只猫”、“一辆汽车”）。

这种通过多层非线性变换，逐层构建从低级特征到高级语义的层次化表征的能力，是深度学习强大的体现，也是它与大脑学习机制非常相像之处。

最大的不同之处：学习信号与能量效率

- **在深度学习中**：
  - 模型依赖于全局的、精确的误差信号。在监督学习中，这个信号就是预测输出与真实标签之间的差异（如交叉熵损失）。
  - 反向传播算法会将这个全局误差精确地、逐层地分解，并计算出网络中每一个参数对最终误差的贡献（梯度）。这个过程需要完美的对称性（前向和反向通路的权重必须转置对称）和微分性。
  - 这是一个高度协调、集中式的优化过程，但其生物学合理性备受质疑。大脑中似乎不存在一个能够将误差信号从顶层精确地、瞬时地传回底层每一个突触的物理机制。
  - 当前深度学习模型依赖同步、密集计算的架构，无论输入是否相关，几乎所有神经元在每次前向传播中都会被激活和计算，这种“始终在线”的工作方式导致其训练过程需要消耗堪比一个小城市的巨大电力。
- **在大脑中**：
  - 大脑的学习被认为主要是局部的和自监督的。神经元的可塑性（如赫布学习，“一起激活的神经元会连接在一起”）主要取决于其前馈输入和局部反馈（如来自邻近神经元或特定调制系统如多巴胺的信号），而非一个全局的、精确的误差梯度。
  - 大脑利用大量的无标签数据进行学习。我们通过观察世界、与环境互动（强化学习）、预测感官输入（预测编码理论）等方式，从数据本身的结构中提取规律，而不是依赖于大量人工标注的标签。
  - 学习信号可能是稀疏的、延迟的、嘈杂的（如多巴胺信号作为奖励预测误差），而不是像反向传播那样精确和即时。
  - 大脑采用异步、事件驱动的稀疏激活架构，神经元仅在处理信息时才消耗能量，这种高效机制使其能以约20瓦的极低功耗运行。

以一张表格作为总结：

对比维度	当前深度学习模型	生物大脑
表征结构	采用多层非线性网络，逐层构建从低级（边缘、纹理）到高级（语义、物体）的层次化特征。	感知系统（如视觉皮层）呈层级组织，V1→V2→IT 等区域逐步编码从简单到复杂的抽象特征。
学习信号来源	依赖全局、精确的误差信号（如标签与预测的差异），通过反向传播计算每个参数的梯度。	依赖局部、稀疏、延迟的信号（如神经调质、脉冲时序），无证据支持全局误差反传；大量使用自监督和无监督学习。
学习规则	基于端到端可微的优化（如SGD + 反向传播），要求计算图处处光滑、权重更新高度协调。	基于局部可塑性规则（如赫布学习、STDP），突触更新仅依赖邻近神经元活动，无需全局协调。
数据与监督方式	主要依赖大量人工标注数据（监督学习）或精心设计的代理任务（自监督）。	主要通过与环境互动、预测感官输入、稀疏奖励（如多巴胺）进行学习，极少依赖精确标签。
计算与能耗特性	同步、密集计算：每次前向/反向传播激活几乎所有单元；训练能耗极高（兆瓦级）。	异步、事件驱动、稀疏激活：神经元仅在需要时放电；整体功耗极低（约20瓦），能效极高。

A2.10

Q

由通用近似定理可知，即使只有一个隐藏层的浅层网络，只要有足够的神经元，就可以近似任何函数。那么，为什么还需要构建深层的神经网络（例如ResNet-101有100多层）？请从模型容量、计算效率、特征抽象层次三个角度阐述深度网络的优势。

A

1. 模型容量

- **通用近似定理的代价**：通用近似定理虽然保证了浅层网络的表达能力，但它没有告诉我们需要多少个神经元。**对于许多现实世界中的复杂函数（如图像识别、自然语言理解），用单层网络去近似，可能需要指数级增长的神经元数量。**例如，要精确表示一个涉及多个变量交互的函数，浅层网络可能需要为每一种可能的交互组合都分配一个神经元，这在计算和存储上是完全不可行的。
- **深度网络的优势**：深层网络通过层次化的组合，能够用少得多的参数来表示同样复杂甚至更复杂的函数。每一层都可以看作是在前一层的输出上进行一次“特征变换”或“特征组合”。这种组合是乘法性的，而非加法性的。因此，**深度网络能以多项式级别的参数数量，高效地表示那些浅层网络需要用指数级别参数才能表示的函数。**这使得深度网络在模型容量和参数效率上具有压倒性优势。

2. 计算效率

- **浅层网络的计算瓶颈**：一个拥有数百万甚至数十亿神经元的单层网络，其计算图是一个巨大的、完全连接的二分图。**每一次前向传播都需要进行一次规模为输入维度 × 隐藏层神经元数的巨型矩阵乘法。这不仅内存消耗巨大，而且无法有效利用现代硬件（如GPU）的并行计算能力，因为计算过于集中在一个步骤，同时浅层网络的模型并行也较为困难。**

- **深度网络的优势**：**深层网络将计算分解为多个连续的、较小的步骤（层）。每个步骤的计算量相对较小，并且这些步骤可以被高效地流水线化。**更重要的是，像卷积这样的操作引入了稀疏连接和参数共享，极大地减少了计算量和参数量。**这种分而治之的策略使得深度网络在现代计算硬件上能够被高效地训练和推理，而同等表达能力的浅层网络则完全无法在实践中运行。**

3. 特征抽象层次

- **浅层网络的局限**：单层网络试图一步到位地从原始输入（如像素）映射到最终输出（如类别标签）。**它缺乏一个中间的、结构化的表示过程。所有复杂的模式识别和决策逻辑都被"压扁"在一个巨大的权重矩阵里，这使得模型难以学习到数据中固有的、层次化的结构。**
- **深度网络的优势**：这是深度网络最核心、最直观的优势。深度网络天然地支持层次化的特征学习：
  - **浅层**（靠近输入）：学习低级特征，例如图像中的边缘、角点、简单纹理。
  - **中层**：组合低级特征，形成中级特征，例如物体的部件（眼睛、轮子、门把手）。
  - **深层**（靠近输出）：整合中级特征，形成高级语义特征，例如完整的物体（人脸、汽车）、场景（厨房、海滩）甚至抽象概念。

  这种从"具体"到"抽象"的渐进式理解过程，与人类感知和认知世界的方式高度吻合。它不仅让模型更强大，也让学习过程更稳定、更符合数据的内在规律。例如，在ResNet-101这样的网络中，前面的几十层可能在构建通用的视觉基元，而最后的几十层则专注于特定任务的精细判别。这种分工协作是浅层网络无法实现的。
- **灵活性与复用性**：这种层次化的表示不仅提升了模型的表达能力，还带来了强大的灵活性与复用性。**通过迁移学习和微调，我们可以将在大规模数据上学到的通用低级和中级特征，高效地迁移到数据稀缺的新任务中。这种'预训练 + 微调'的范式已成为现代深度学习的标准实践，而浅层网络由于缺乏中间表示层次，完全无法支持这种知识迁移。**

总结

通用近似定理告诉我们浅层网络"能做什么"（What it can do），但它没有告诉我们"如何高效地去做"（How to do it efficiently）。深层网络通过其层次化的结构，在参数效率、计算效率和特征抽象能力上实现了对浅层网络的全面超越。它不是简单地增加了模型的宽度，而是通过增加深度，构建了一个能够分步骤、有层次地理解复杂世界的计算架构。这就是为什么在通用近似定理成立的今天，我们依然需要并广泛使用像深度神经网络。