A2.1 (5 points) 尝试解释Epoch、Iteration、Batch几个概念及其不同，尝试说明batch\_size的选择依据和影响。

答：

Epoch（轮次）：

一个 Epoch 表示完整的训练数据集（全部样本）在神经网络中前向传播和反向传播的一次迭代。

在一个 Epoch 中，模型会遍历整个训练数据集一次，计算损失并更新模型的参数。

Epoch 的数量通常是一个超参数，控制着模型训练的总轮次。增加 Epoch 数可以提高模型的性能，但可能会导致过拟合。

Iteration（迭代）：

一个 Iteration 表示一次参数更新的操作，通常是由一个小批次（Batch）的数据执行前向传播和反向传播后进行的参数更新。

在一个 Epoch 中，可以有多个 Iterations，每个 Iteration 处理一个 Batch 的数据。

Iteration 的数量通常是由数据集大小和 Batch 大小来决定。

Batch（批次）：

一个 Batch 表示一组训练样本，它是数据集的一个子集，用于一次前向传播和反向传播的计算。

Batch 的大小（batch size）是一个重要的超参数，控制了每次参数更新使用的样本数量。

Batch 大小的选择会直接影响训练过程的速度和性能。常见的 Batch 大小有 32、64、128 等。

综合来说：从迭代次数角度，一个Epoch中有许多个Iteration。从数据量角度，一个Epoch中包含着全部样本，一个Iteration中包含着Batch决定的样本数量。

使用我自己在A1全连接神经网络作业中的代码来举例：

有MiniBatch训练函数如下：



其中Epoch和Iteration分别为大小两个循环，小循环中的batchsize即为batch：

1. **for** i **in** range(iter\_num):
2. # 随机打乱数据和标签，以创建随机的小批次
3. indices = list(range(m))
4. random.shuffle(indices)
5. totalcost = 0
6. **for** j **in** range(0, m, batch\_size):
7. batch\_indices = indices[j:j + batch\_size]
8. X\_batch = X[batch\_indices]
9. y\_batch = y[batch\_indices]

Batch size 的选择依据和影响：

训练速度：较大的 Batch size 可以加速训练过程，因为每次参数更新处理更多的样本，减少了参数更新的频率。但过大的 Batch size 可能会导致内存不足或计算资源不足的问题。

泛化能力：较小的 Batch size 可以帮助模型更好地泛化，因为它在每个 Iteration 中接触到不同的数据，有助于模型更好地适应数据的多样性。但较小的 Batch size 可能会导致训练过程中的噪声，使得收敛不稳定。

超参数调整：Batch size 通常需要与学习率一起进行调整，因为它们之间有关联。较大的 Batch size 可能需要较大的学习率，而较小的 Batch size 可能需要较小的学习率，以确保参数更新的稳定性。

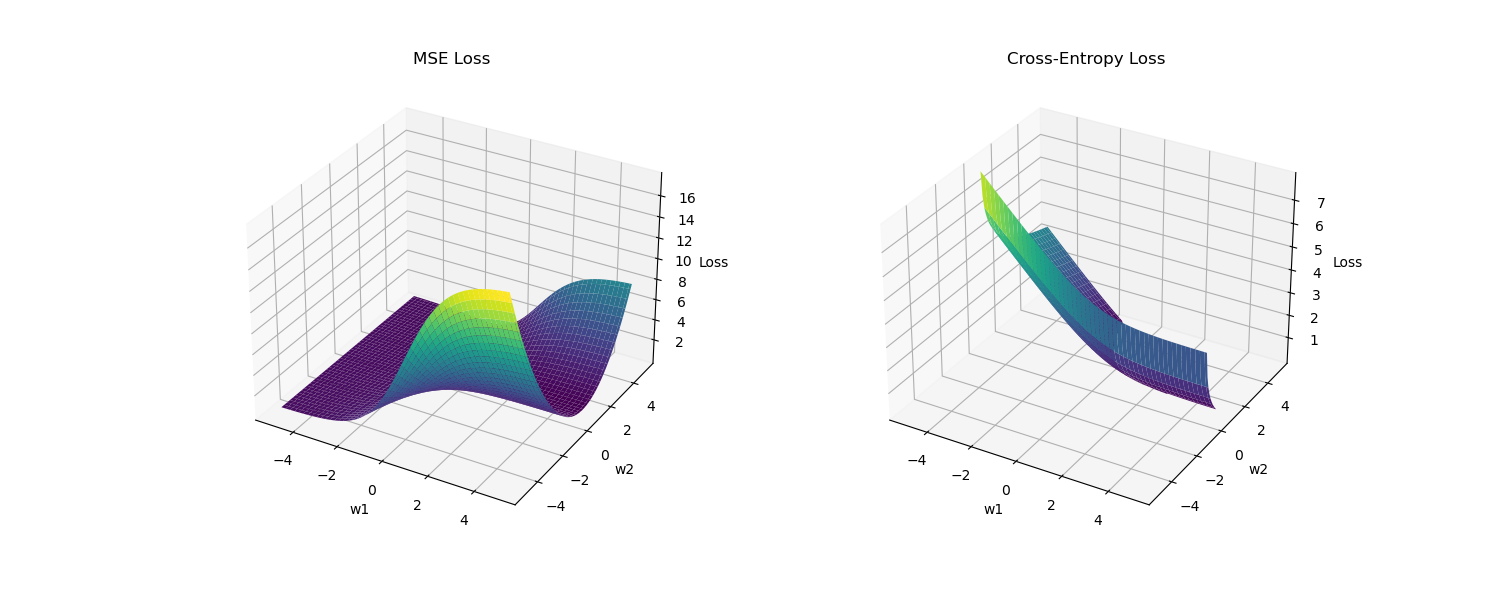
计算资源：Batch size 的选择还受限于可用的计算资源。在具有有限内存和计算能力的硬件上，需要选择合适的 Batch size。

A2.2 (5 points) 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）。

答：

写python代码构建神经网络，并绘图如下：

1. **import** numpy as np
2. **import** matplotlib.pyplot as plt
3. **from** mpl\_toolkits.mplot3d **import** Axes3D
5. # 定义损失函数
6. **def** mse\_loss(w1, w2):
7. # 这里假设输入为1，输出为1
8. input\_data = 1
9. actual\_output = 1
10. hidden\_output = 1 / (1 + np.exp(-(w1 \* input\_data)))  # Sigmoid激活函数
11. predicted\_output = w2 \* hidden\_output
12. **return** 0.5 \* (predicted\_output - actual\_output) \*\* 2
14. **def** cross\_entropy\_loss(w1, w2):
15. input\_data = 1
16. actual\_output = 1
17. hidden\_output = 1 / (1 + np.exp(-(w1 \* input\_data)))  # Sigmoid激活函数
18. predicted\_output = w2 \* hidden\_output
19. **return** - (actual\_output \* np.log(predicted\_output) + (1 - actual\_output) \* np.log(1 - predicted\_output))
21. # 创建参数空间
22. w1\_range = np.linspace(-5, 5, 100)
23. w2\_range = np.linspace(-5, 5, 100)
24. W1, W2 = np.meshgrid(w1\_range, w2\_range)
26. # 计算损失函数值
27. MSE\_loss = np.zeros\_like(W1)
28. CE\_loss = np.zeros\_like(W2)
30. **for** i **in** range(len(w1\_range)):
31. **for** j **in** range(len(w2\_range)):
32. MSE\_loss[i][j] = mse\_loss(W1[i][j], W2[i][j])
33. CE\_loss[i][j] = cross\_entropy\_loss(W1[i][j], W2[i][j])
35. # 绘制三维图像
36. fig = plt.figure(figsize=(15, 6))
38. # 均方误差损失函数图像
39. ax1 = fig.add\_subplot(121, projection='3d')
40. ax1.plot\_surface(W1, W2, MSE\_loss, cmap='viridis')
41. ax1.set\_title('MSE Loss')
42. ax1.set\_xlabel('w1')
43. ax1.set\_ylabel('w2')
44. ax1.set\_zlabel('Loss')
46. # 交叉熵损失函数图像
47. ax2 = fig.add\_subplot(122, projection='3d')
48. ax2.plot\_surface(W1, W2, CE\_loss, cmap='viridis')
49. ax2.set\_title('Cross-Entropy Loss')
50. ax2.set\_xlabel('w1')
51. ax2.set\_ylabel('w2')
52. ax2.set\_zlabel('Loss')
54. plt.show()



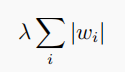
由凸函数定义可知，凸函数的一个特征是，从任意两点连成的线段在函数图像上方。由图像可发现，这两种函数关于参数具有非凸性。

A2.5 (5 points) 分析为什么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。

答：

L1正则化：

L1正则化对模型的损失函数添加了参数的绝对值之和作为惩罚项。即：

损失 = 原始损失 + 

其中，是w模型参数，是lambda正则化系数。

为什么L1倾向于得到稀疏解？

惩罚项鼓励模型的权重尽量小，而对于不重要的特征，L1正则化有可能直接将其对应的权重压缩到0。因为L1惩罚是不可导的（在0处），这导致了许多权重直接变为0，从而得到一个稀疏解。

L2正则化

L2正则化对模型的损失函数添加了参数的平方和作为惩罚项。即：

损失=原始损失 + IMG_256

为什么L2倾向于得到平滑的解？

L2正则化鼓励模型的权重尽量小，但与L1正则化不同，L2正则化不会将权重压缩到0（除非λ非常大）。相反，它会将所有权重均匀地压缩，这导致了权重值都接近但不等于0，从而得到了一个平滑的解。

为了形象理解，可以想象一个二维空间中的等高线图，L1正则化的等高线是由菱形组成，因此优化路径可能更容易与坐标轴相交，导致某个参数变为0；而L2正则化的等高线是圆形，优化路径会更趋于平滑，不容易与坐标轴相交，因此参数不容易变为0。

L1正则化能产生稀疏解，有助于特征选择。

L2正则化产生非稀疏解，但可以防止过拟合，使得权重分布更平滑。

A2.6 (5 points) 分析Batch normalization对参数优化起到什么作用、如何起到这种作用。

答：

缓解内部协变量偏移（Internal Covariate Shift）:

在深度神经网络中，每一层的输入分布随着上一层的参数更新而变化，这会导致每一层都需要不断地适应其输入的分布变化，从而减缓训练速度。

BN作用：通过对每个mini-batch进行归一化，BN强制每一层的输入都有相似的分布，这减少了内部协变量偏移的影响。

正则化效应:

防止过拟合

BN作用：尽管BN的主要目的不是正则化，但在实践中，BN为模型引入了轻微的噪声，这有助于防止过拟合，因此，使用BN有时可以减少或替代其他正则化技巧如Dropout。

梯度流:

问题：在深层网络中，梯度消失或梯度爆炸是常见问题，导致网络难以训练。

BN作用：通过确保每一层的激活具有相似的尺度，BN有助于保持稳定的梯度流，这样可以减轻梯度消失或梯度爆炸的问题。

允许更高的学习率:

较高的学习率可能导致训练不稳定。

BN作用：由于BN提供了稳定的输入分布，并且有助于缓解梯度问题，因此它允许使用更高的学习率，从而加速训练。

减少对权重初始化的敏感性:

深度神经网络的训练对权重初始化非常敏感。

BN作用：由于BN调整了各层的激活分布，这使得网络对于初始权重的选择不那么敏感，因此可以减少对特定权重初始化策略的依赖。

Batch Normalization的核心思想是，在每一层之后加入一个归一化步骤，使得激活值均值为0，方差为1。然后引入两个可学习的参数（scale和shift），以便网络可以学习到最佳的数据分布。

Batch Normalization通过调整和规范化每一层的输入，帮助神经网络更稳定、更快速地训练，并有助于提高模型的泛化能力。