# 机器学习实验报告

实验名称：建立全连接神经网络

学生姓名：蒋雨初

学生学号：58121102

完成日期：2023/5/27

# 任务描述

通过两种方式实现全连接神经网络，并对图片分类任务行进行测试与实验。

1. 手动实现简单的全连接神经网络
2. 使用 Pytorch 库简洁实现全连接神经网络

Fashion-MNIST 图片分类数据集包含 10 个类别的时装图像，训练集有 60,000 张图片，测试集中有 10,000 张图片。图片为灰度图片，高度（$h$）和宽度（$w$）均为 28 像素，通道数（channel）为 1。10 个类别分别为：t-shirt(T 恤), trouser（裤子）, pullover（套衫），dress（连衣裙）, coat（外套）， sandal（凉鞋），shirt（衬衫），sneaker（运动鞋），bag（包），ankle boot（短靴）。使用训练集数据进行训练，测试集数据进行测试。

图 1 Fashion-MNIST 数据示例

# 实验原理

## 单隐层神经网络

## Fashion-MNIST 数据集

## 内容

Fashion-MNIST 是一个经典的图像分类数据集，包含了 10 个类别共 70000 张 28x28 像素的灰度图像，每个类别包含了 6000 张训练样本和 1000 张测试样本。这个数据集的目的是替代 MNIST 数据集，以便更好地评估图像分类算法的性能。具体内容如下：

1. 类别： Fashion-MNIST 数据集包含以下 10 个类别的图像：

- T 恤/上衣
- 裤子
- 套头衫
- 连衣裙
- 外套
- 凉鞋

- 衬衫
- 运动鞋
- 包
- 短靴

2. 图像特征：每个图像的尺寸为 28x28 像素，灰度图像只有一个通道。因此，每个图像可以表示为一个 28x28 的矩阵，共计 784 个特征。
3. 训练集和测试集：Fashion-MNIST 数据集被划分为训练集和测试集。训练集包含 60000 张图像，其中每个类别有 6000 张图像。测试集包含 10000 张图像，用于评估算法在未见过的数据上的性能。


**图** 2 Fashion-MNIST **部分数据**

# 数据预处理

## 数据增强

数据增强是在训练过程中对原始数据进行一系列变换操作，以产生更多的训练样本，从而增加数据的多样性。数据增强可以有效地扩展训练集的规模，减轻过拟合问题，并提高模型的泛化能力。

以下是一些常见的数据增强技术：

1. 翻转（Flipping）：包括水平翻转和垂直翻转，通过翻转图像可以增加数据的多样性。
2. 旋转（Rotation）：对图像进行旋转操作，以增加不同角度的训练样本。
3. 裁剪（Cropping）：随机裁剪图像的一部分，可以使模型对目标物体的位置具有更好的鲁棒性。
4. 缩放（Scaling）：将图像进行缩放操作，可以模拟不同尺度的目标物体。
5. 平移（Translation）：将图像进行平移操作，可以模拟不同位置的目标物体。
6. 亮度、对比度和饱和度调整（Brightness, Contrast, and Saturation Adjustment）：通过调整图像的亮度、对比度和饱和度，可以使模型对不同光照条件和颜色变化具有更好的适应能力。
7. 噪声添加（Noise Addition）：向图像中添加随机噪声，以增加数据的多样性和鲁棒性。
8. 变换（Transformation）：应用各种几何变换，如扭曲、拉伸等，以模拟不同视角和形变。

这些数据增强技术可以单独应用，也可以组合使用，具体选择哪些数据增强技术取决于应用场景和数据集的特性。通过数据增强，可以提高模型的泛化能力，并在有限的数据集上获得更好的性能。

此外，当使用数据增强技术时，需要小心过拟合的问题。如果应用过多的增强技术，可能会导致模型在训练集上过拟合，但在真实数据上表现不佳。因此，需要进行适当的监控和调整，确保模型在训练和测试集上都能获得好的性能。

考虑到 Fashion-MNIST 中的图像皆为灰度图，在本实验中我们可以仅使用来自 torchvision.transforms 中的 RandomRotation 和 RandomHorizontalFlip。

# 数据归一化

数据归一化是数据预处理中的一项重要步骤，它将数据按照一定规则进行缩放和标准化，以使数据具有相似的范围和统计特性。以下是几个原因说明为什么需要数据归一化：

1. 梯度下降算法：在机器学习中，很多模型的训练过程使用梯度下降算法来更新参数。梯度下降算法对输入数据的规模和范围敏感，如果不对数据进行归一化，可能会导致梯度更新过快或过慢，从而影响模型的收敛速度和稳定性。
2. 特征权重：在一些模型中，例如支持向量机（SVM）和 K 近邻（KNN），特征的尺度差异会对模型的学习和预测产生影响。如果某些特征的尺度较大，它们会在模型中起主导作用，而忽略其他尺度较小的特征。通过归一化，可以确保每个特征对模型的贡献相对平衡。
3. 模型收敛和稳定性：在训练神经网络等深度学习模型时，数据归一化可以帮助加速模型的收敛过程。对输入数据进行归一化可以使每个特征具有相似的范围，减小了梯度更新的差异，提高了模型的稳定性。
4. 避免数值计算问题：在进行数值计算时，如果输入数据具有不同的尺度，可能会导致数值溢出或下溢等问题。通过归一化，可以将数据限制在合理的范围内，避免这些数值计算问题的发生。
5. 提高模型的泛化能力：通过数据归一化，可以减小特征之间的相关性，提高特征的独立性，从而帮助模型更好地泛化到未见过的数据。

综上所述，数据归一化可以帮助调整数据的尺度和范围，提高模型的训练效果、稳定性和泛化能力。它是数据预处理中的一个重要步骤，通常在训练模型之前应用于输入数据。

不同的数据集有不同的统计学特性，所以中值和标准差都需要单独计算。在 Fashion-MNIST 上，中值和标准差分别是 0.1307 和 0.3081。

在实践上，可以使用来自 torchvision.transforms 中的 Normalize。

# 模型原理

单隐层神经网络是一种基本的神经网络模型，它包含一个输入层、一个隐藏层和一个输出层。其中输入层的节点数为 784，对应于输入图像的 784 个像素；隐藏层的节点数为 256，它是一个可调整的参数，可以根据实际需求进行调整；输出层的节点数为 10，对应于 10 个不同的类别。

在单隐层神经网络中，每个节点都与前一层的所有节点相连，每个连接都有一个权重，它决定了该连接的重要性。隐藏层的节点通过激活函数（如 sigmoid 函数, 线性整流函数 ReLU）将输入信号加权求和后，输出一个非线性的结果。输出层的节点也通过激活函数（如 softmax 函数）将输入信号加权求和后，输出一个概率分布，表示输入图像属于每个

类别的概率。

# 任务一：手动实现单隐层全连接神经网络

训练单隐层神经网络通常使用反向传播算法，该算法通过比较网络输出和实际标签之间的差异来调整权重，以最小化误差。在训练过程中，网络逐渐学习到输入数据的特征，从而提高分类的准确性。

算法概要如下：

a) 正向传播，得到各层输出$\boldsymbol{h}^{(0)}, \boldsymbol{h}^{(1)}, \boldsymbol{h}^{(2)}$

$$\boldsymbol{z}^{(0)} = \boldsymbol{x}, \boldsymbol{h}^{(0)} = \boldsymbol{z}^{(0)}$$
$$\boldsymbol{z}^{(1)} = \boldsymbol{W}^{(1)}\boldsymbol{h}^{(0)} + \boldsymbol{b}^{(1)}, \boldsymbol{h}^{(1)} = relu\left(\boldsymbol{z}^{(1)}\right)$$
$$\boldsymbol{z}^{(2)} = \boldsymbol{W}^{(2)}\boldsymbol{h}^{(1)} + \boldsymbol{b}^{(2)}, \boldsymbol{h}^{(2)} = softmax(\boldsymbol{z}^{(2)})$$

b) 反向传播，得到输出层误差$\boldsymbol{\delta}^{(2)}$

$$\boldsymbol{\delta}^{(2)} = \frac{\partial L}{\partial \boldsymbol{z}^{(2)}} = \boldsymbol{h}^{(2)} - \boldsymbol{y}$$

其中 L 是损失函数。再计算各层梯度和隐藏层误差$\boldsymbol{\delta}^{(1)}$

$$\nabla_{\boldsymbol{W}^{(2)}} L = {\boldsymbol{h}^{(2)}}^{T} \cdot \boldsymbol{\delta}^{(2)}$$
$$\nabla_{\boldsymbol{b}^{(2)}} L = \boldsymbol{\delta}^{(2)}$$
$$\boldsymbol{\delta}^{(1)} = \frac{\partial relu\left(\boldsymbol{z}^{(1)}\right)}{\partial \boldsymbol{z}^{(1)}} \odot \left(\boldsymbol{\delta}^{(2)} \cdot {\boldsymbol{W}^{(2)}}^{T}\right)$$
$$\nabla_{\boldsymbol{W}^{(1)}} L = {\boldsymbol{h}^{(1)}}^{T} \cdot \boldsymbol{\delta}^{(1)}$$
$$\nabla_{\boldsymbol{b}^{(1)}} L = \boldsymbol{\delta}^{(1)}$$

其中 relu 的导数为

$$\frac{\partial relu(x)}{\partial x} = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases}$$

c) 更新各层参数

$$\boldsymbol{W}^{(t)} \leftarrow \boldsymbol{W}^{(t)} - \eta \nabla_{\boldsymbol{W}^{(t)}} L$$
$$\boldsymbol{b}^{(t)} \leftarrow \boldsymbol{b}^{(t)} - \eta \nabla_{\boldsymbol{b}^{(t)}} L$$

# 任务二：使用 PyTorch 库简洁实现全连接神经网络

由三层组成：

1. 展平层：将输入的 28x28 图像展开为 784 个神经元输入。
2. 全连接层：784 个结点。激活函数：relu
3. 输出层：10 个节点。激活函数：softmax

# 实验设置

## 运行环境

Platform: google colab

GPU: RTX 3090

# 隐藏层结点数（任务一）

对于单层全连接神经网络，隐藏层结点数（也称为隐藏单元数）可以对最终的预测结果产生影响。隐藏层结点数指定了神经网络中隐藏层的神经元数量。

影响因素如下：

1. 表示能力：隐藏层结点数的增加可以增加神经网络的表示能力。更多的隐藏结点可以提供更多的自由度，使得网络能够学习更复杂的函数。因此，增加隐藏层结点数可能会提高神经网络对数据的拟合能力。
2. 模型复杂度：增加隐藏层结点数会增加神经网络的模型复杂度。一个更复杂的模型可能会更好地拟合训练数据，但也可能导致过拟合问题。过拟合指的是模型过度学习了训练数据中的噪声和细节，而无法很好地泛化到新的数据。
3. 训练时间：增加隐藏层结点数可能会增加训练神经网络的时间。更多的隐藏层结点意味着更多的参数需要优化，这可能会增加训练时间的成本。如果数据集较大或训练资源有限，增加隐藏层结点数可能会导致训练时间过长或资源消耗过高。
4. 过拟合风险：随着隐藏层结点数的增加，神经网络的容量也增加，可能会增加过拟合的风险。如果隐藏层结点数过多，模型可能过于复杂，无法泛化到新的数据，导致在训练集上表现很好但在测试集上表现较差。

因此，在选择隐藏层结点数时，需要根据具体问题的复杂性、可用的训练数据量和计算资源进行权衡。常见的做法是通过交叉验证或验证集来选择合适的隐藏层结点数，以达到较好的预测性能和泛化能力。

在本实验中，测试了当结点数目为 128、256、512、1024、2048 时的情况，并绘制了图像。

# 参数初始化（任务二）

针对**任务二**进行参数初始化实验。网络训练的过程中, 容易出现梯度消失(梯度特别的接近 0)和梯度爆炸(梯度特别的大)的情况,导致大部分反向传播得到的梯度不起作用或者起反作用. 研究人员希望能够有一种好的权重初始化方法: 让网络前向传播或者反向传播的时候, 卷积的输出和前传的梯度比较稳定. 合理的方差既保证了数值一定的不同, 又保证了数值一定的稳定。下面介绍目前 DNN 权重初始化的方法。

## 1. Constant Initialization

将神经网络中的模型全部初始化为某个常数，意味着将所有计算单元初始化为完全相同的状态，这会使每个计算单元对同一样例的输出和反向更新的梯度存在某种对称关系或甚至完全相同，导致神经网络的灵活性大打折扣。

## 2. Random Initialization

Random Initialization 将每个计算单元初始化成不同的状态，但却无法很好选择概率模型中的超参数，例如正态分布$W \sim N(\mu, \sigma^2)$中的$\mu$和$\sigma$。

在本实验中默认的初始化即为此初始化。具体而言，使用 torch.randn 将权重随机初始化，使用 torch.zeros 将偏置设置为 0.

## 3. Xavier Initialization[1]

通常不使用常量初始化，而随机初始化又存在无法很好选择概率模型中的超参数的问题，那么如何既保证输入输出的差异性，又能让模型稳定而快速的收敛呢？我们要求正向传播时候数据流经每一层前后的方差一致，并且反向传播时候数据流经每一层，该层梯度前后方差一致。

要描述“差异性”，首先就能想到概率统计中的方差这个基本统计量。对于每个神经元的输入$z=\sum_{i=1}^{n} w_i x_i$，其中 n 是上一层神经元的数量。因此，根据概率统计里的两个随机变量乘积的方差展开式：

$$D(w_i x_i) = E\left(w_i^2\right)D(x_i) + E(x_i)^2 D(w_i) + D(w_i)D(x_i)$$

可以得到，如果$E(x_i) = E(w_i) = 0$（可以通过批量归一化 Batch Normalization 来满足），那么就有：

$$D(\boldsymbol{z}) = \sum_{i=1}^{n} D(w_i)D(x_i)$$

如果随机变量$x_i$和$w_i$再满足独立同分布的话：

$$D(\boldsymbol{z}) = \sum_{i=1}^{n} D(w_i)D(x_i) = nD(\boldsymbol{w})D(\boldsymbol{x})$$

注意到整个大型前馈神经网络的工作无非是将原始样本稳定的映射成它的类别，也就是将样本空间映射到类别空间。试想，如果样本空间与类别空间的分布差异很大，比如说类别空间特别稠密，样本空间特别稀疏辽阔，那么在类别空间得到的用于反向传播的误差丢给样本空间后简直变得微不足道，也就是会导致模型的训练非常缓慢。同样，如果类别空间特别稀疏，样本空间特别稠密，那么在类别空间算出来的误差丢给样本空间后简直是爆炸般的存在，即导致模型发散震荡，无法收敛。因此，我们要让样本空间与类别空间的分布差异（密度差别）不要太大，也就是要让它们的方差尽可能相等。

因此为了得到$D(\boldsymbol{z}) = D(\boldsymbol{x})$，只能让$nD(\boldsymbol{w}) = 1$，也就是$D(\boldsymbol{w}) = 1/n$。

然而，正向传播时是从前往后计算的，因此 $D(\boldsymbol{w}) = 1/n_{in}$，反向传播时是从后往前计算的，因此 $D(\boldsymbol{w}) = 1/n_{out}$。通常$n_{in} \neq n_{out}$，所以需要取调和均值

$$D(\boldsymbol{w}) = \frac{2}{1/\frac{1}{n_{in}} + 1/\frac{1}{n_{out}}} = \frac{2}{n_{in} + n_{out}}$$

假设$w$为均匀分布，由$w$在区间$[a, b]$内均匀分布时的方差为

$$D(\boldsymbol{w}) = \frac{(b-a)^2}{12}$$

联立两式，即得**均匀分布下的 Xavier 初始化**

$$\boldsymbol{w} \sim U\left(-\sqrt{\frac{6}{n_{in} + n_{out}}}, \sqrt{\frac{6}{n_{in} + n_{out}}}\right)$$

[1] Glorot X, Bengio Y. Understanding the difficulty of training deep feedforward neural networks[J]. Journal of Machine Learning Research, 2010, 9:249-256.

同理也可以得到**高斯分布下的 Xavier 初始化**

$$\boldsymbol{w}\sim N\left(0,\sqrt{\frac{2}{n_{in}+n_{out}}}\right)$$

此外，对于不同的激活函数还需要乘上不同的增益系数。如下图所示

| nonlinearity | gain |
|---|---|
| Linear / Identity | $1$ |
| Conv{1,2,3}D | $1$ |
| Sigmoid | $1$ |
| Tanh | $\frac{5}{3}$ |
| ReLU | $\sqrt{2}$ |
| Leaky Relu | $\sqrt{\frac{2}{1+\text{negative\_slope}^2}}$ |
| SELU | $\frac{3}{4}$ |

**图 3 不同激活函数下，w 的分布**[2]

实践上，可以使用 torch.nn.init.xavier_uniform_或 torch.nn.init.xavier_normal_。

## 4. Kaiming Initialization[3]

尽管 Xavier 初始化能够在 Sigmoid 和 tanh 激活函数叠加的神经网络中起到一定的效果，但由于 ReLU 激活函数属于非饱和类激活函数，并不会出现类似 Sigmoid 和 tanh 激活函数使用过程中可能存在的梯度消失或梯度爆炸问题，反而因为 ReLU 激活函数的不饱和特性，ReLU 激活函数的叠加极有可能出现神经元活性消失的问题，很明显，该类问题无法通过 Xavier 初始化解决。

没有时间看论文了，此处给出结论：

1. 前向传播的时候, 每一层的卷积计算结果的方差为 1.
2. 反向传播的时候, 每一 层的继续往前传的梯度方差为 1(因为每层会有两个梯度的计算, 一个用来更新当前层的权重, 一个继续传播, 用于前面层的梯度的计算.)

**均匀分布下的 Kaiming 初始化**

$$\boldsymbol{w}\sim U\left(-\sqrt{\frac{3}{n_{in}}},\sqrt{\frac{3}{n_{in}}}\right)\text{ or }\boldsymbol{w}\sim U\left(-\sqrt{\frac{3}{n_{out}}},\sqrt{\frac{3}{n_{out}}}\right)$$

**高斯分布下的 Kaiming 初始化**

$$\boldsymbol{w}\sim N\left(0,\frac{1}{\sqrt{n_{in}}}\right)\text{ or }\boldsymbol{w}\sim N\left(0,\frac{1}{\sqrt{n_{out}}}\right)$$

实践上，可以使用 torch.nn.init.kaiming_uniform_和 torch.nn.init.kaiming_normal_。

[2] https://pytorch.org/docs/stable/nn.init.html
[3] Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification arXiv:1502.01852 [cs.CV]

在本实验中，主要是比较了以 ReLU 为激活函数，随机初始化、均匀分布下的 Xavier 初始化和 Kaiming 初始化、高斯分布下的 Xavier 初始化和 Kaiming 初始化对神经网络的影响。

## 学习率（Learning rate）

学习率（Learning rate）是指在神经网络的训练过程中，用于调整模型权重的步长或速率。学习率对神经网络训练有以下影响：

1. 收敛速度：学习率决定了参数更新的步长。较大的学习率可以加快模型的收敛速度，因为每次参数更新的幅度更大。然而，如果学习率过大，可能会导致参数在最优解附近波动或震荡，使模型难以稳定地收敛。相反，较小的学习率会使模型收敛速度较慢，但更有可能找到更好的最优解。
2. 网络性能：学习率直接影响模型在训练数据和测试数据上的性能。过大的学习率可能会导致模型在训练集上表现良好，但在测试集上泛化能力较差，产生过拟合。过小的学习率可能会导致模型无法充分学习数据的模式和特征，表现为欠拟合。选择适当的学习率是优化模型性能的重要因素之一。
3. 跳出局部最优解：在训练过程中，模型可能会陷入局部最优解，而无法达到全局最优解。适当的学习率可以帮助模型跳出局部最优解并继续寻找更好的解决方案。较大的学习率可能有助于跳出局部最优，但也可能导致不稳定的训练过程。较小的学习率可以使模型更稳定地收敛，但可能陷入较差的局部最优。
4. 调整学习率策略：学习率还与调整策略相关。在训练过程中，学习率可能需要进行动态调整，以便在后期阶段更加细致地调整模型参数。例如，学习率衰减、学习率衰减和动量等技术可以用来优化学习率的变化方式，以获得更好的训练结果。

在本实验当中，我们考虑实践 One Cycle Policy。

## LR Range Test（任务一）

2015 年，Leslie N. Smith 提出了该技术。其核心是将模型进行几次迭代，在最初的时候，将学习率设置的足够小，然后，随着迭代次数的增加，逐渐增加学习率，记录下每个学习率对应的损失，并绘图：（LR 的初始值仅为 1e-7，然后增加到 10）

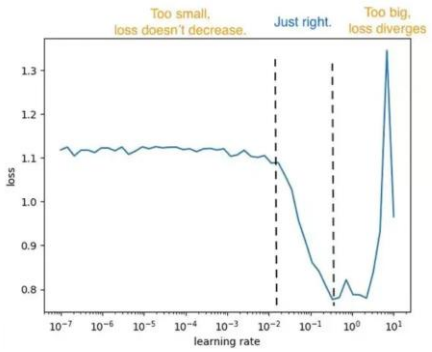



**图 4 Loss 随 LR 的变化**

LR Range Test 图应该包括三个区域，第一个区域中学习率太小以至于损失几乎没有减少，第二个区域里损失收敛很快，最后一个区域中学习率太大以至于损失开始发散。因此，第二个区域中的学习率范围就是我们在训练时应该采用的。我们把第二个区域的左端叫做下界学习率（base_lr），右端叫做上界学习率（max_lr）。

所以，这个方法字如其名，就是学习率范围测试，为训练寻找一个合适的学习率范围。

# Cyclic Learning Rates

在一些经典方法中，学习率总是逐步下降的，从而保证模型能够稳定收敛，但 Leslie Smith 对此提出了质疑，Leslie Smith 认为让学习率在合理的范围内周期性变化（即 Cyclical LR： 在 base_lr 和 max_lr 范围内循环学习率）是更合理的方法，能够以更小的步骤提高模型准确率。

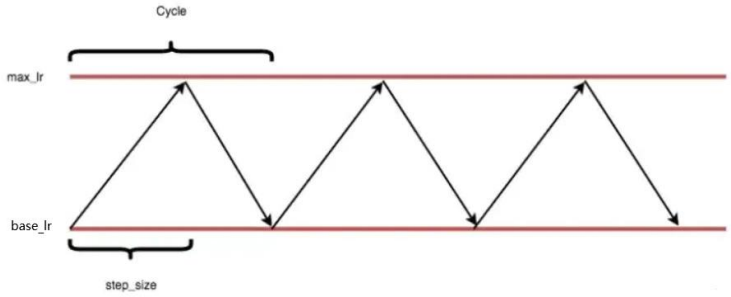



图 5 Cyclic LR 策略

# One Cycle Policy（任务二）

在 Cyclical LR 和 LR Range Test 的基础上，Leslie 继续改进，提出了 The one cycle policy。在一周期策略中，最大学习率被设置为 max_lr 中可以找到的最高值，最小学习率比最大学习率小几个数量级（比如设为 0.1 倍的 max_lr）。

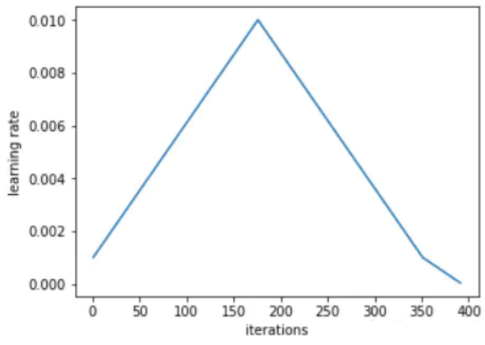



图 6 One Cycle Policy 策略

如上图，假设整个训练周期约 400 个 iter，前 175 个 iter 用来 warm-up。中间 175 个 iter 用来退火到初始学习率, 由于此前有相当大的时间模型处于较高的学习率，作者认为，这将起到一定的正则化作用，防止模型在陡峭最小值驻留，从而更倾向于寻找平坦的局部最小值。最后几十个 iter 学习率进行进一步衰减至 0，这将使得模型在一个‘平坦’区域内收敛至一个较为‘陡峭’的局部最小值。

在实践上，我们可以直接使用 torch.optim.lr_scheduler.OneCycleLR。在训练时记录每一步 learning rate 并绘图，如下：

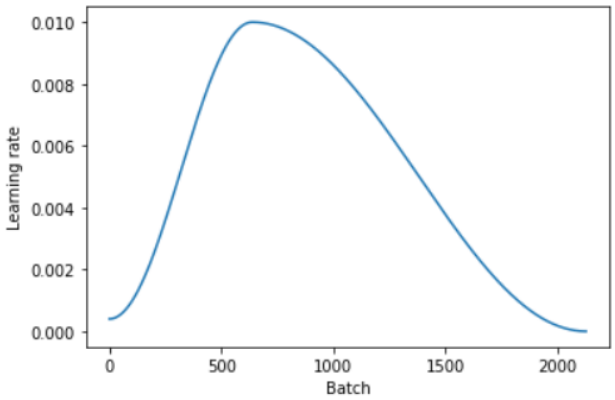



**图 7 在 epochs=30,max_lr=0.01,optimizer=SGD 条件下，LR 随 Batch 变化的情况**

# 梯度截断（Gradient clipping）

梯度截断（gradient clipping）是一种在深度学习中用于稳定训练过程的技术。它主要应用于梯度爆炸的情况，即在网络训练过程中，梯度的数值变得非常大，导致训练不稳定或发散。具体来说，当计算出的梯度的范数（即梯度的绝对值的平方和开根号）超过了预先设定的阈值时，就将梯度进行缩放，使其范数不超过阈值。这个阈值通常称为截断值或阈值，可以手动设置。

在本实验中，我们将它设置为 **0.1**。

# 权重衰退（Weight decay）

权重衰减（weight decay）是一种在神经网络训练中常用的正则化技术。它的目的是通过在损失函数中引入一个附加项，来惩罚模型中较大的权重值。

在神经网络中，权重（或参数）的大小对模型的性能和泛化能力起着重要的影响。权重衰减通过在损失函数中增加一个正则化项，通常是权重的平方和（L2 范数），来限制权重的大小。这个额外的正则化项使得模型倾向于选择较小的权重值，从而降低过拟合的风险。

具体来说，权重衰减通过将权重的平方和乘以一个较小的正则化系数（也称为衰减因子）添加到损失函数中。这个正则化项会在训练过程中对较大的权重进行惩罚，使得模型倾向于选择较小的权重值。损失函数中的权重衰减项可以表示为：

$$L(w) = Loss(y, \hat{y}) + \lambda ||w||^2$$

其中，$L(w)$是加入权重衰减的损失函数，$Loss(y, \hat{y})$是原始的损失函数（例如，均方误差或交叉熵），$\lambda$是正则化系数，$||w||^2$是权重的平方和。

通过调整正则化系数$\lambda$的值，可以控制权重衰减的程度。较大的$\lambda$会对权重施加更大的惩罚，使得权重更加趋向于零。较小的$\lambda$会对权重施加较小的惩罚，允许权重值相对较大。

通过使用权重衰减，可以帮助防止模型在训练数据上过拟合，提高其泛化能力。这是因为权重衰减鼓励模型学习简单的权重分布，减少了过多依赖个别训练样本的风险。

在本实验中，我们选取$\lambda = \mathbf{10^{-4}}$。这是一个经验值，有些论文证明这是一个较优的阈值。

# 其他设置

- 性能指标：准确率

# 实验结果

## 参数实验

### 隐藏层结点数

测试了当结点数目为 128、256、512、1024、2048 时的情况，并绘制误差-轮数图像。由图中可以看出，最佳的隐藏层结点数应该为 1024。然而较大的神经网络训练会消耗更多的时间，且训练损失差异不大，所以仍然选择结点数为 256.

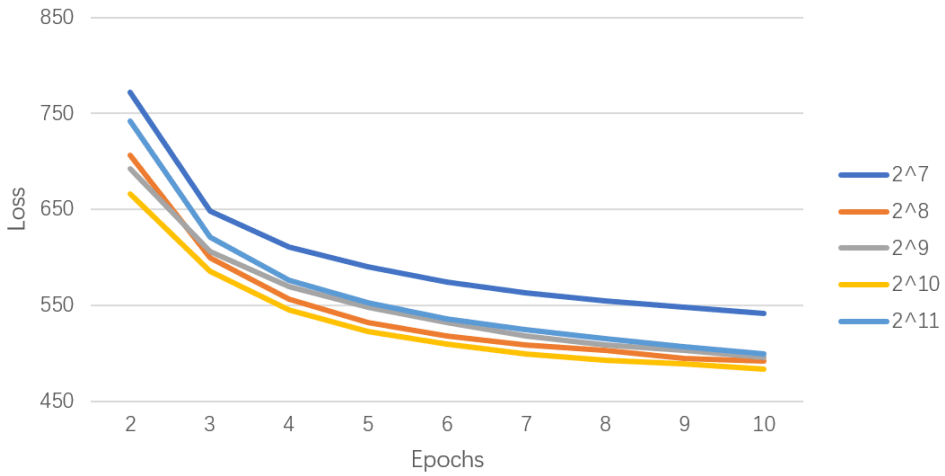



图 8 不同隐藏层结点数的测试集误差

### 参数初始化

比较了以 ReLU 为激活函数，随机初始化、均匀分布下的 Xavier 初始化和 Kaiming 初始化、高斯分布下的 Xavier 初始化和 Kaiming 初始化对神经网络的影响，并画出了在测试集上的 loss 随轮数变化的图像。

由下图可见，不同的初始化方式对最终收敛的结果影响不大，但是收敛过程则各有差异。通过计算各个初始化方式的测试集误差的方差，可见波动性较小的是 kaiming uniform 初始化方法。此外，xavier 初始化在 relu 上的表现确实不佳，符合之前的预期。

```
default:  0.008267246109966599
xavier uniform:  0.009727791152846202
xavier normal:  0.011576973632159735
kaiming uniform:  0.005557279300623649
kaiming normal:  0.007156463735207943
```

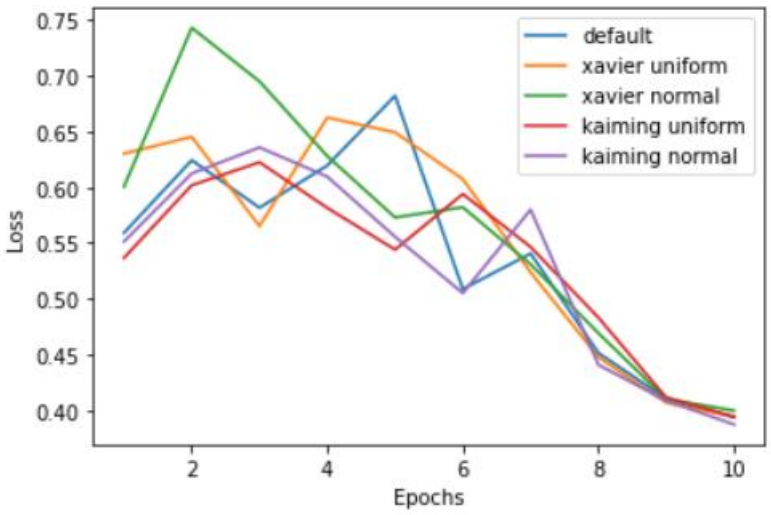
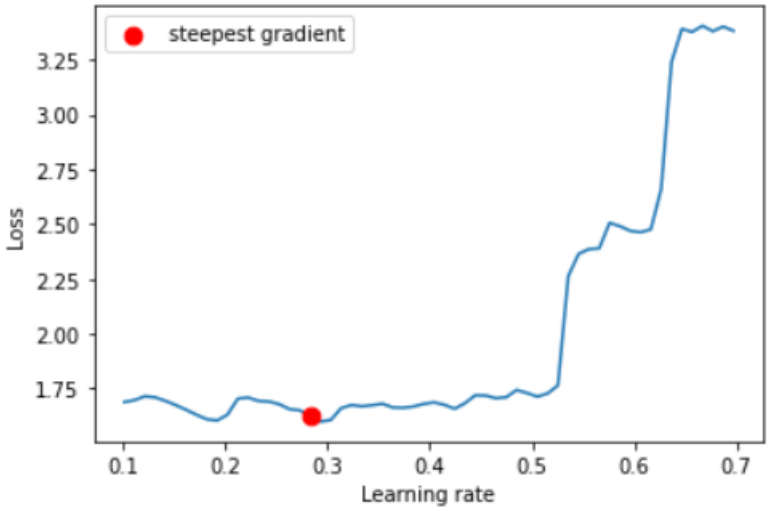



图 9 不同初始化方式对神经网络的影响

# 学习率

运用前文提到的方法，从$10^{-7}$遍历到1，得到最佳的学习率应为 0.283。




# 模型训练

# 任务一

采用上文得到的最优学习率训练，绘制 train loss 和 test loss 随 epoch 变化的曲线

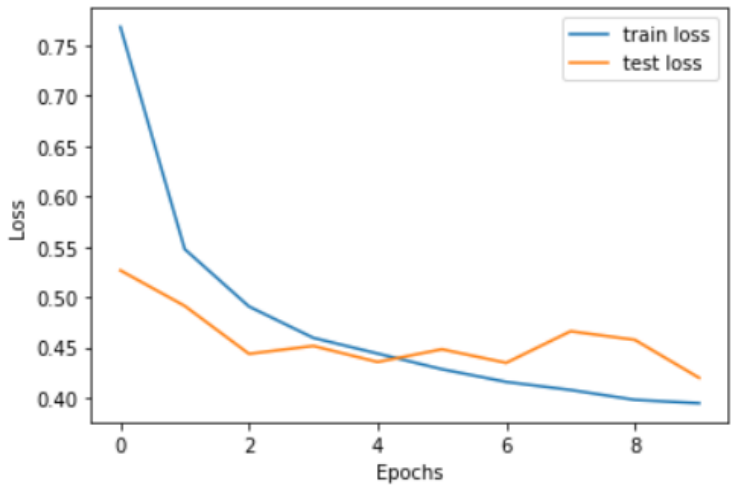



图 10 手动实现的单隐层神经网络的损失随轮数的变化曲线

在测试集上，预测准确率达到 85.67%。

# 任务二

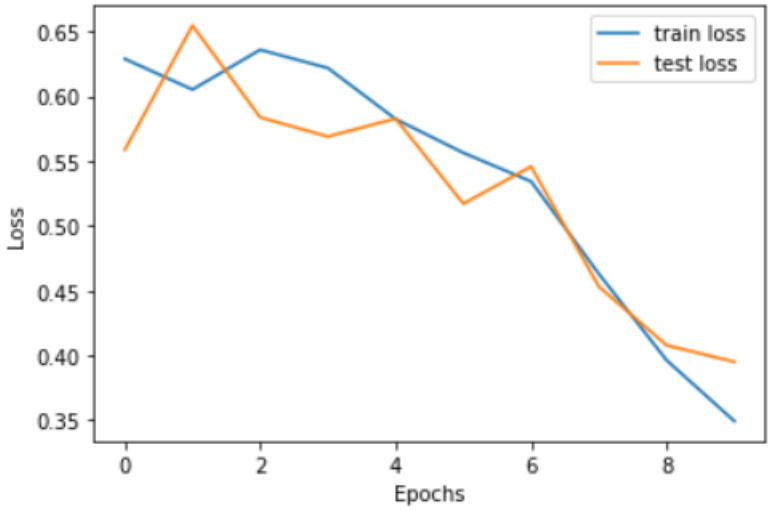



图 11 基于 pytorch 的单隐层神经网络的损失随轮数的变化曲线

在测试集上，预测准确率达到 86.00%。