同濟大學
TONGJI UNIVERSITY

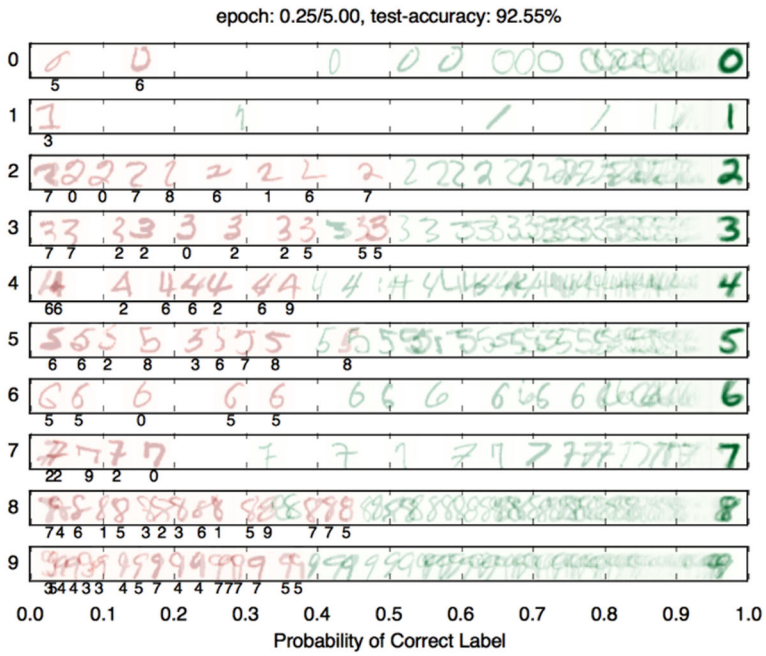
人工智能原理与技术
Project5实验报告

学　　号：2154312
姓　　名：郑博远
教　　师：王俊丽
完成日期：2023 年 5 月 20 日

1. 问题概述

1.1. 问题的直观描述

本次的项目主要围绕机器学习展开。其中，包含感知机的二类分类任务（问题 1）、正弦函数拟合（问题 2）、MNIST 数据集上的手写数字分类（问题 3，如下图所示）、单词语言识别（问题 4）等。各个问题的具体描述如下：



问题 1 中，要求实现一个感知机（perceptron），通过训练感知机来学习对输入进行二类分类。感知机根据输入和权重的线性组合来进行分类，并根据分类结果对权重进行调整，直到全部分类正确。具体来说，需要实现 run(self, x)方法，计算存储的权重向量和给定输入之间的点积；实现 get_prediction(self, x)方法，如果点积是非负的，则返回 1，否则返回-1；编写 train(self)方法，重复遍历数据集并对错误分类的示例进行更新。在数据集上完成一次完整遍历且没有发生错误分类时，训练准确性达到 100%，此时训练终止。

问题 2 中，需要训练一个神经网络来近似表示区间[-2π, 2π]上的 sin(x)函数。参考神经网络提示中的建议，搭建相对简单的网络结构来实现正弦函数拟合。使用 nn.SquareLoss 作为损失函数。具体需要完成以下任务：实现 RegressionModel.__init__方法，进行必要的初始化；实现 RegressionModel.run 方法，返回一个 batch_size×1 的节点，表示模型的预测结果；实现 RegressionModel.get_loss 方法，根据给定的输入和目标输出返回损失值；实现 RegressionModel.train 方法，使用基于梯度的更新方法来训练模型。

问题 3 中，需要训练一个神经网络，对 MNIST 数据集中的手写数字进行分类。数字输入是一个 28×28 像素的图像，其像素值存储在一个 784 维的浮点数向量中。输出是一个 10 维向量，分别对应每个数字估计的概率。在理想情况下，只有正确类别对应的位置上为 1，其他位置上都为 0。通过训练，使得模型在测试集上的准确率达到 97%以上。

问题 4 中，需要构建循环神经网络（RNN）模型对英语、西班牙语、芬兰语、荷兰语、波兰语等五种语言单词进行语言识别，即根据给定的逐个单词文本判断文本所属的语言。首先通过 RNN 将任意长度的输入单词编码为一个固定大小的向量，再通过额外的输出层将该输入单词的向量转换为用于生成语言标识的分类得分。

1.2. 项目已有代码的阅读与理解

`models.py`	感知机与神经网络模型的具体应用
`nn.py`	神经网络迷你库
`autograder.py`	自动评分程序
`backend.py`	机器学习任务的后端代码
`data`	数字分类与语言分类的数据集

1.2.1. `models.py`

该文件中包含感知机与神经网络模型的具体应用，即问题 1 到问题 4 中的感知机模型、回归模型、数字分类模型以及语言分类模型。上述四个模型对应四道题中不同的神经网络，本次 Project 需要补充其中的方法，使得神经网络完成既定任务。

其中如下几个方法较为重要：在__init__方法中，需要进行模型的初始化，即定义神经网络的结构等信息，包括设计层数、每层的输入输出大小、超参数等信息，并通过 nn.Parameter 方法初始化权重和偏置参数；在 run 方法中，对一个 batch 的输入进行计算，并返回神经网络的输出结果；在 train 方法中进行神经网络的训练，并自行指定指定某个终止条件，使得训练集上准确率达到某个范围后训练停止。

```
class PerceptronModel(object):
```

```
class RegressionModel(object):
    """
    A neural network model for approximating a function that maps from real
    numbers to real numbers. The network should be sufficiently large to be
able
```

```
    to approximate sin(x) on the interval [-2pi, 2pi] to reasonable
precision.
    """
```

```
class DigitClassificationModel(object):
    """
    A model for handwritten digit classification using the MNIST dataset.

    Each handwritten digit is a 28x28 pixel grayscale image, which is
flattened
    into a 784-dimensional vector for the purposes of this model. Each entry
in
    the vector is a floating point number between 0 and 1.

    The goal is to sort each digit into one of 10 classes (number 0 through
9).
    """
```

```
class LanguageIDModel(object):
    """
    A model for language identification at a single-word granularity.
    """
```

1.2.2. nn.py

本文件中提供了神经网络模型的迷你库，提供了权重更新、点积计算、激活函数、损失函数等方法。下面逐一介绍该文件中定义的几个重要的类和方法：

- Node 类：表示该文件中所有节点的基类。Node 类中实现了__repr__方法，用于调试时输出该节点的类型与形状信息；
- DataNode 类：Parameter 和 Constant 节点的父类，用于存储参数和常量的节点；
- Parameter 类：表示神经网络中的参数节点，用于存储网络的权重和偏置参数；
- Constant 类：表示常量节点，用于表示输入特征、输出标签和反向传播计算得到的梯度；
- FunctionNode 类：表示基于其他节点计算得到的值的节点，包含前向计算和反向传播方法；
- Add 类：实现两个矩阵的逐元素相加；

- AddBias 类：将一个偏置向量添加到每个特征向量上；
- DotProduct 类：实现批量的点积操作；
- Linear 类：对输入进行线性变换（矩阵乘法）；
- ReLU 类：ReLU 非线性激活函数，将输入中的负值替换为零；
- SquareLoss 类：计算输入之间的平方差损失；
- SoftmaxLoss 类：计算分类问题的批量 Softmax 损失；
- gradients 方法：计算并返回各个传入参数的损失梯度；
- as_scalar 方法：将大小为 1x1 的结点转换为 Python 中的 Number。

1.3. 解决问题的思路和想法

1.3.1. 问题 1 的解决思路

问题 1 中，我们需要设计一个感知机模型来分类数据点，并训练该模型直到将所有数据分类正确。解决问题的思路如下：

对于给定的数据点，通过将数据点与感知机的权重进行点积运算来计算评分。根据数据点的评分，将数据点预测为两个类别中的一个（+1 和 -1）从而实现分类。在训练感知机模型的过程中，遍历训练数据集中的每个数据点，使用当前感知机的权重来计算数据点的评分。将评分与数据点的实际类别进行比较，若分类错误则通过更新感知机的权重来改善分类准确性。重复上述步骤，直到感知机正确分类所有训练数据点。

1.3.2. 问题 2 的解决思路

问题 2 中，我们需要设计一个回归模型，在给定区间 $[-2\pi, 2\pi]$ 内对正弦函数 sin(x)进行合理精度的逼近。解决问题的思路如下：

在模型初始化过程中，设定输入数据的维度、隐藏层的大小和输出层的大小，并设置了学习率和批量大小等超参数，初始化模型中的权重参数和偏置项。在模型运行阶段，将输入数据点通过线性变换和激活函数传递到隐藏层，并进一步通过线性变换和偏置项计算输出层的值，从而得到模型对于给定数据点的预测值。

在训练模型时，采用平方损失函数来度量预测值与真实值之间的差异。使用梯度下降法来更新模型的参数以最小化损失函数。遍历训练数据集，并按批次取出数据进行训练。对于每个批次的数据，计算损失函数关于模型参数的梯度，并使用梯度乘以学习率的方式更新模型的参数。重复上述过程直到模型达到足够的精度，训练过程停止。

1.3.3. 问题 3 的解决思路

问题 3 中，我们需要设计一个模型来对 MNIST 数据集中的手写数字进行分类。每个手写数字是一个 28x28 像素的灰度图像，目标是将每个数字分类“0”到“9”的 10 个类别中。解决问题的思路如下：

在模型初始化过程中，设定输入数据的维度、隐藏层和输出层的大小，并设置了学习率和批量大小等超参数，初始化模型中的权重参数和偏置项。在模型运行阶段，将输入数据点通过线性变换和激活函数传递到隐藏层，并进一步通过线性变换和偏置项计算输出层的值，从而得到给定数据点各个分类的预测值，表示数据点属于每个类别的概率分数。

在训练模型时，采用 softmax 损失函数来度量模型对于真实标签的预测概率分数与真实概率分数之间的差异。采用梯度下降法更新模型参数以最小化损失函数，使用梯度乘以学习率的方式进行参数更新。重复上述过程直到验证准确率超过 0.98，训练过程停止。

1.3.4. 问题 4 的解决思路

问题 4 中，我们需要设计循环神经网络来进行单词级别的语言识别。数据集包含来自五种不同语言的单词，字母表总共包含 47 个不同的字符。解决问题的思路如下：

在模型初始化过程中，设定字符的数量和语言的列表等参数，并初始化模型中的权重参数和偏置项。在模型运行阶段，使用一个循环神经网络（Recurrent Neural Network，RNN）来对输入的单词序列进行建模。具体而言，将 one-hot 编码的每个字符通过线性变换传递给 RNN 的一个时间步，并将得到的隐藏状态作为输入传递到 RNN 的后续时间步。通过这种方式，RNN 能够对整个单词序列进行建模，并生成一个最终的隐藏状态。在得到最终的隐藏状态后，使用线性变换和偏置项将其映射到一个大小为 5 的输出层，对应着五种不同的语言，表示单词属于每种语言的概率分数。

在训练模型阶段，采用 softmax 损失函数来度量模型对于真实标签的预测概率分数与真实概率分数之间的差异。采用梯度下降法来更新模型的参数以最小化损失函数，我们使用梯度乘以学习率的方式更新模型的参数。重复上述过程，直到达到训练停止条件。

2. 算法设计

2.1. 问题 1 感知机

算法功能：

本算法实现了一个二类分类感知机，用于对数据点进行分类。该感知机将数据点分为属于某个类别的点（+1）和不属于该类别的点（-1）。通过学习数据点的特征权重，感知机可以对新的数据点进行分类预测。

设计思路：

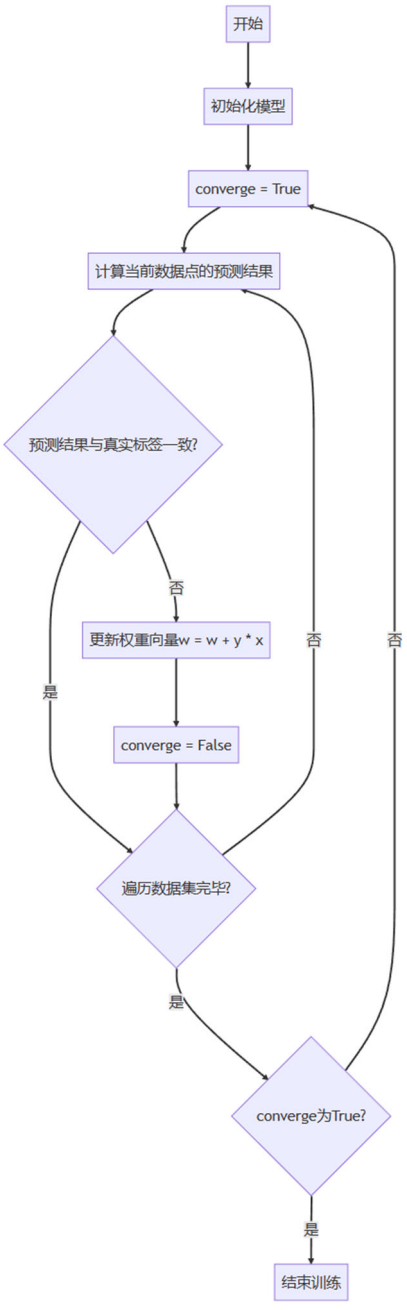
1. 初始化感知机的权重向量 w，该向量的维度与数据点的特征维度相同；

2. 根据感知机的权重向量 w，计算数据点 x 的得分，即将数据点 x 与权重向量 w 进行内积运算；

3. 根据得分确定数据点的预测类别，如果得分大于等于 0，则预测为正类别（+1），否则预测为负类别（-1）；

4. 在训练阶段，遍历训练数据集并根据预测结果更新感知机的权重向量 w，直到所有数据点都被正确分类。

算法流程图：如下图所示。



2.2. 问题 2 正弦函数拟合

算法功能：

本算法实现了一个回归模型，通过训练神经网络来逼近正弦函数。该模型能够将输入的实数映射到输出的实数，在区间[-2π, 2π]上对 sin(x)进行合理的逼近。

设计思路：

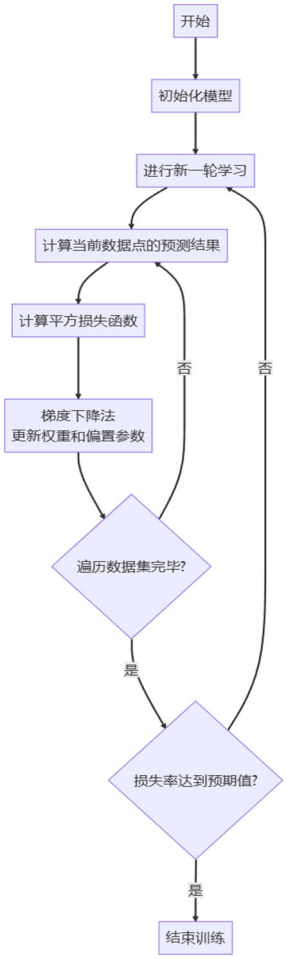
1. 模型结构：如下页图所示，该模型包含两个隐层。具体来说，输入数据的维度为 1，两个隐层分别具有 50 个、150 个神经元，输出层具有 1 个神经元。

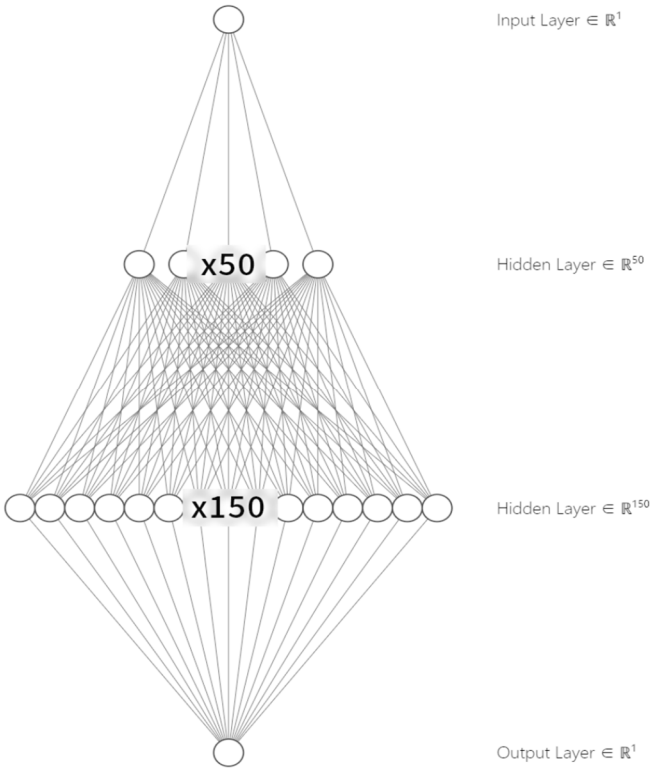
2. 激活函数：除输出层外，每层后都使用 ReLU 作为激活函数。

3. 损失函数：模型使用平方损失函数来度量预测值与真实值之间的差异。

4. 参数更新：采用随机梯度下降法（SGD）来更新模型参数，通过计算损失函数关于参数的梯度，并乘以学习率来更新参数。

算法流程图：如下图所示。





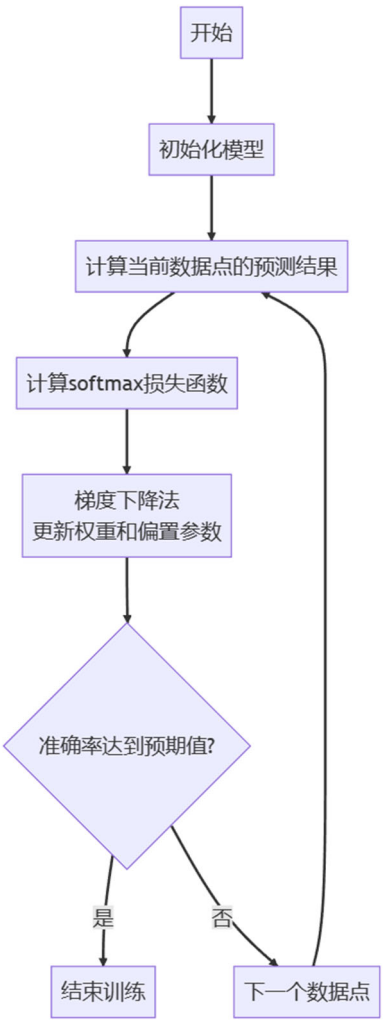
2.3. 问题 3 手写数字分类

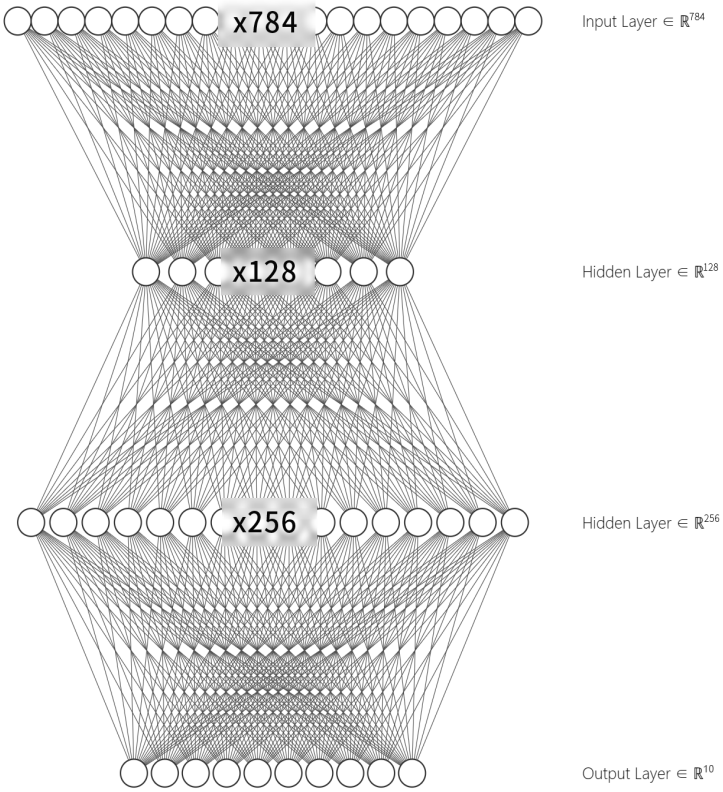
算法功能：本算法实现了一个基于神经网络的 MNIST 数字分类模型。该模型通过对输入的手写数字图像进行特征提取和分类，将每个数字图像分为 0 到 9 的 10 个类别。

设计思路：

1. 输入层：将每个手写数字图像展平为一个 784 维的向量作为模型的输入。
2. 隐层：设计两个隐层，分别具有 128 个和 256 个神经元。在每个隐层后都进行激活，使用 ReLU 作为激活函数。
3. 输出层：设计一个包含 10 个神经元的输出层，表示 10 个数字类别的分数。
4. 损失函数：使用 Softmax Loss 作为损失函数，将模型输出的分数与真实标签进行比较并计算损失。
5. 训练过程：使用梯度下降算法进行训练，通过反向传播计算梯度并更新模型参数。

算法流程图：如下图所示。





2.4. 问题 4 单词语言分类

算法功能：

本算法实现了一个语言分类模型，用于识别单词所属的语言类别。通过将单词的字符逐个输入模型，模型使用循环神经网络（Recurrent Neural Network，RNN）对输入进行处理，并将结果汇总为一个表示语言类别概率的向量。模型根据这个概率向量，判断单词所属的语言类别。

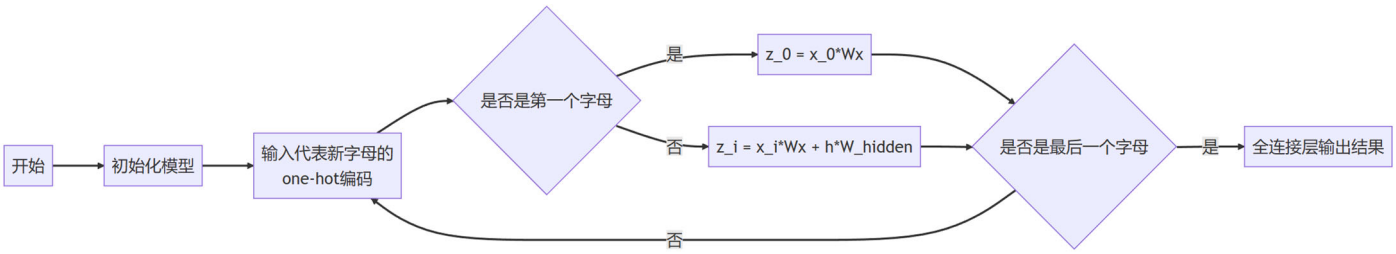
设计思路：

1. 循环神经网络：RNN 的输入是一个列表，列表的每一项代表 one-hot 编码后的字母。模型逐个处理输入单词的字符，并通过 RNN 的循环连接将当前字符的编码与前一个字符的隐藏状态进行结合。具体来说，选择将整个单词信息压缩为一个大小为 256 的向量。对于每个新输入的字母，分别对上一步输出的结果向量和代表新的字母的向量进行乘以权重做线性操作后相加。这样，模型能够在处理每个字符时考虑前面字符的影响，从而捕捉到单词中字符之间的相关性。在处理完所有字符后，模型将最后一个字符的隐藏状态作为整个单词的最终状态表示，其包含整个单词的上下文信息，可以用于进行语言分类。

2. 输出层和分类：模型使用全连接层将最终状态映射到各个语言类别上，模型预测的语言类别为具有最高概率的类别。

3. 训练过程：使用 Softmax Loss 作为损失函数，将模型输出的分数与真实标签进行比较并计算损失。使用梯度下降算法进行训练，通过反向传播计算梯度并更新模型参数。

算法流程图：如下图所示。



3. 算法实现

3.1. 问题 1 感知机

在问题 1 中，我实现了一个感知机算法以用于对数据点进行二分类。感知机能够通过学习，将输入的数据点划分为两个类别：正类和负类。

在 PerceptronModel 类的初始化方法中，我创建了一个参数 self.w，它代表了感知机的权重。权重的维度由输入数据的维度决定。

算法的核心是 run 方法，它用于计算感知机对于给定数据点 x 的评分。评分值越高，表示数据点更有可能属于正类，评分值越低，表示数据点更有可能属于负类。在 run 方法中，我使用了 nn.DotProduct 操作，将权重向量 self.w 与输入数据 x 进行点积运算，得到评分值。为了预测数据点的类别，我定义了 get_prediction 方法。它使用 run 方法计算数据点的评分，并根据评分的正负值判断数据点的类别。如果评分大于等于 0，那么将数据点预测为正类（返回 1），否则预测为负类（返回-1）。

训练方法 train 用于训练感知机模型，使其能够对数据点进行正确分类。在训练过程中，我使用了迭代的方式遍历数据集。对于每个数据点，如果感知机的预测结果与真实标签不一致，说明分类错误，需要更新权重。具体而言，我使用了 self.w.update 方法，根据感知机的预测错误与真实标签之间的差异，对权重进行调整。这样，通过不断迭代训练，感知机将逐渐调整权重，以最大程度地减少分类错误。

在训练过程中，我使用了一个循环来反复遍历数据集，并检查是否已经达到了收敛条件。如果所有数据点都被正确分类，即没有分类错误的情况发生，那么训练过程将停止，感知机模型将被认为已经收敛。具体代码如下：

```
class PerceptronModel(object):
    def __init__(self, dimensions):
        """
        Initialize a new Perceptron instance.

        A perceptron classifies data points as either belonging to a
particular
        class (+1) or not (-1). `dimensions` is the dimensionality of the
data.
        For example, dimensions=2 would mean that the perceptron must
classify
        2D points.
```

```
        """
        # 权重对应一个输入及其维度
        self.w = nn.Parameter(1, dimensions)

    def get_weights(self):
        """
        Return a Parameter instance with the current weights of the
perceptron.
        """
        return self.w

    def run(self, x):
        """
        Calculates the score assigned by the perceptron to a data point x.

        Inputs:
            x: a node with shape (1 x dimensions)
        Returns: a node containing a single number (the score)
        """
        "*** YOUR CODE HERE ***"
        return nn.DotProduct(self.get_weights(), x)

    def get_prediction(self, x):
        """
        Calculates the predicted class for a single data point `x`.

        Returns: 1 or -1
        """
        "*** YOUR CODE HERE ***"
        return 1 if nn.as_scalar(self.run(x)) >= 0 else -1

    def train(self, dataset):
        """
        Train the perceptron until convergence.
        """
        "*** YOUR CODE HERE ***"

        # 不断训练直到分类全部正确
        while True:
            converge = True
            # 遍历数据集
```

```
            for x, y in dataset.iterate_once(1):
                if(self.get_prediction(x) != nn.as_scalar(y)):
                    converge = False
                    self.w.update(x, nn.as_scalar(y))
            # 全部分类正确 训练结束
            if(converge):
                break
```

3.2. 问题 2 正弦函数拟合

在问题 2 中，我实现了一个用于正弦函数回归的神经网络模型。该模型能够通过学习训练，在区间[-2π, 2π]上逼近 sin(x)函数。

在 RegressionModel 类的初始化方法中，我首先定义了模型的参数。输入数据的维度为 1，表示每个数据点的输入是一个实数。隐层的大小通过 self.h1 和 self.h2 进行设置，我选择了 50 和 150 作为隐层的神经元数量。输出数据的大小为 1，对应模型输出也是实数。

模型的核心是 run 方法，它接受一个批次的输入数据 x，并通过神经网络模型计算出对应的预测值。在 run 方法中，我使用了一系列的线性变换、激活函数（ReLU）和偏置项加法操作来定义神经网络的前向传播过程。具体而言，我使用了 nn.Linear 操作进行线性变换，nn.ReLU 操作进行激活函数处理，以及 nn.AddBias 操作进行偏置项的加法。通过这些操作，我将输入数据逐层传递给神经网络的隐层，并最终得到输出层的预测结果。

为了训练模型，我定义了 get_loss 方法来计算模型在一批训练数据上的损失值。在 get_loss 方法中，我使用了 nn.SquareLoss 操作来计算预测值与真实标签之间的均方差损失。这个损失值用于衡量模型的预测精度，目标是最小化损失值，使得模型能够更好地逼近正弦函数。

训练过程通过 train 方法实现。在训练过程中，我使用了一个循环来迭代训练数据集。对于每个批次的数据，我首先计算损失值，然后使用 nn.gradients 方法计算出损失对于模型参数的梯度。接着，就能够通过学习率与梯度依次更新模型的参数。通过这样的参数更新过程，模型逐渐调整权重，以最小化损失函数。在训练过程中，我还计算了每个 epoch 的平均损失值 avg_loss。如果平均损失值小于设定的阈值（0.005），则认为模型已经达到了足够精度，训练过程将停止。具体代码如下：

```
class RegressionModel(object):
    """
    A neural network model for approximating a function that maps from real
```

```
    numbers to real numbers. The network should be sufficiently large to be
able
    to approximate sin(x) on the interval [-2pi, 2pi] to reasonable
precision.
    """
    def __init__(self):
        # Initialize your model parameters here
        "*** YOUR CODE HERE ***"
        # 输入数据维度
        self.i = 1
        # 隐层的大小
        self.h1 = 50
        self.h2 = 150
        # 输出数据大小
        self.o = 1
        # 学习率
        self.learning_rate = 0.05
        self.batchsize = 10

        self.W1 = nn.Parameter(self.i, self.h1)
        self.b1 = nn.Parameter(1, self.h1)
        self.W2 = nn.Parameter(self.h1, self.h2)
        self.b2 = nn.Parameter(1, self.h2)
        self.W3 = nn.Parameter(self.h2, self.o)
        self.b3 = nn.Parameter(1, self.o)

    def run(self, x):
        """
        Runs the model for a batch of examples.

        Inputs:
            x: a node with shape (batch_size x 1)
        Returns:
            A node with shape (batch_size x 1) containing predicted y-values
        """
        "*** YOUR CODE HERE ***"
        # 逐层计算
        layer1 = nn.ReLU(nn.AddBias(nn.Linear(x, self.W1), self.b1))
        layer2 = nn.ReLU(nn.AddBias(nn.Linear(layer1, self.W2), self.b2))
        # 输出层计算
        return nn.AddBias(nn.Linear(layer2, self.W3), self.b3)
```

```
    def get_loss(self, x, y):
        """
        Computes the loss for a batch of examples.

        Inputs:
            x: a node with shape (batch_size x 1)
            y: a node with shape (batch_size x 1), containing the true y-
values
                to be used for training
        Returns: a loss node
        """
        "*** YOUR CODE HERE ***"
        return nn.SquareLoss(self.run(x), y)

    def train(self, dataset):
        """
        Trains the model.
        """
        "*** YOUR CODE HERE ***"

        while True:
            avg_loss = 0
            for x, y in dataset.iterate_once(self.batchsize):
                # 计算损失
                loss = self.get_loss(x, y)
                # 更新各个参数权重
                grad_W1, grad_b1, grad_W2, grad_b2, grad_W3, grad_b3 =
nn.gradients(loss, [self.W1, self.b1, self.W2, self.b2, self.W3, self.b3])
                self.W1.update(grad_W1, -self.learning_rate)
                self.b1.update(grad_b1, -self.learning_rate)
                self.W2.update(grad_W2, -self.learning_rate)
                self.b2.update(grad_b2, -self.learning_rate)
                self.W3.update(grad_W3, -self.learning_rate)
                self.b3.update(grad_b3, -self.learning_rate)
                avg_loss += loss.data

            # 计算本 epoch 的平均 loss
            avg_loss /= self.batchsize

            # loss 小于特定值则认为足够精准
```

```
            if(avg_loss < 0.005):
                break
```

3.3. 问题 3 手写数字分类

问题 3 中，主要实现了一个手写数字分类模型，用于将 MNIST 数据集中的手写数字图像分类到 10 个不同的类别（数字 0 到 9）。

DigitClassificationModel 类的构造函数中，进行了模型参数的初始化。其中，输入数据维度为 784（图像展平后的向量大小），隐层分别设定为 128 和 256，输出层大小为 10（对应 10 个类别），学习率为 0.5，批量大小为 200。模型使用三个参数矩阵和三个偏置向量来定义网络的权重。

模型的核心是 run 方法，该方法接收一个批量的图像作为输入，通过一系列的线性变换和非线性激活函数得到预测的分数。具体来说，即通过两个 ReLU 激活的隐层和一个线性输出层，将输入数据映射到 10 维的输出空间。

模型的损失函数由 get_loss 方法定义，它计算预测结果与真实标签之间的 Softmax Loss 损失。在 train 方法中，使用迭代器从数据集中获取批量数据进行训练。对于每个批次数据，首先计算损失，然后使用反向传播计算损失相对于模型参数的梯度，并根据学习率更新模型参数。模型将持续训练直到验证准确率达到 0.98 以上。具体代码如下：

```
class DigitClassificationModel(object):
    """
    A model for handwritten digit classification using the MNIST dataset.

    Each handwritten digit is a 28x28 pixel grayscale image, which is
flattened
    into a 784-dimensional vector for the purposes of this model. Each entry
in
    the vector is a floating point number between 0 and 1.

    The goal is to sort each digit into one of 10 classes (number 0 through
9).

    (See RegressionModel for more information about the APIs of different
    methods here. We recommend that you implement the RegressionModel before
    working on this part of the project.)
    """
```

```
def __init__(self):
    # Initialize your model parameters here
    "*** YOUR CODE HERE ***"
    # 输入数据维度（图片的大小）
    self.i = 28 * 28
    # 隐层的大小
    self.h1 = 128
    self.h2 = 256
    # 输出层的大小
    self.o = 10
    # 学习率
    self.learning_rate = 0.5
    self.batchsize = 200

    self.W1 = nn.Parameter(self.i, self.h1)
    self.b1 = nn.Parameter(1, self.h1)
    self.W2 = nn.Parameter(self.h1, self.h2)
    self.b2 = nn.Parameter(1, self.h2)
    self.W3 = nn.Parameter(self.h2, self.o)
    self.b3 = nn.Parameter(1, self.o)

def run(self, x):
    """
    Runs the model for a batch of examples.

    Your model should predict a node with shape (batch_size x 10),
    containing scores. Higher scores correspond to greater probability of
    the image belonging to a particular class.

    Inputs:
        x: a node with shape (batch_size x 784)
    Output:
        A node with shape (batch_size x 10) containing predicted scores
            (also called logits)
    """
    "*** YOUR CODE HERE ***"
    # 隐层计算
    layer1 = nn.ReLU(nn.AddBias(nn.Linear(x, self.W1), self.b1))
    layer2 = nn.ReLU(nn.AddBias(nn.Linear(layer1, self.W2), self.b2))
    # 输出层计算
    return nn.AddBias(nn.Linear(layer2, self.W3), self.b3)
```

```
    def get_loss(self, x, y):
        """
        Computes the loss for a batch of examples.

        The correct labels `y` are represented as a node with shape
        (batch_size x 10). Each row is a one-hot vector encoding the correct
        digit class (0-9).

        Inputs:
            x: a node with shape (batch_size x 784)
            y: a node with shape (batch_size x 10)
        Returns: a loss node
        """
        "*** YOUR CODE HERE ***"
        return nn.SoftmaxLoss(self.run(x), y)

    def train(self, dataset):
        """
        Trains the model.
        """
        "*** YOUR CODE HERE ***"

        for x, y in dataset.iterate_forever(self.batchsize):
            # 计算损失
            loss = self.get_loss(x, y)
            # 更新各个参数权重
            grad_W1, grad_b1, grad_W2, grad_b2, grad_W3, grad_b3 =
nn.gradients(loss, [self.W1, self.b1, self.W2, self.b2, self.W3, self.b3])
            self.W1.update(grad_W1, -self.learning_rate)
            self.b1.update(grad_b1, -self.learning_rate)
            self.W2.update(grad_W2, -self.learning_rate)
            self.b2.update(grad_b2, -self.learning_rate)
            self.W3.update(grad_W3, -self.learning_rate)
            self.b3.update(grad_b3, -self.learning_rate)
            if(dataset.get_validation_accuracy() > 0.98):
                break
```

3.4. 问题 4 单词语言分类

问题 4 中，我实现了用于对单词语言分类的循环神经网络模型。该模型使用循环神经

网络（RNN）对输入的单词进行建模，并判断单词属于的语言类别。

在模型的构造函数中，题目给定的代码初始化了参数的维度，包括字符的数量和语言的类别数量。除此之外，我定义了学习率和批处理大小等超参数，并创建了权重矩阵和偏置项的参数，包括输入层到隐层的权重矩阵 W_x，隐层到隐层的权重矩阵 W_h，以及隐层到输出层的权重矩阵 W_o 和偏置项 b_o。

在模型的 run 方法中，输入的形式为一个列表，列表的每一项代表该批次单词的某个字符。循环遍历输入列表，将每个字符的独热向量作为输入进行线性变换和非线性激活操作。具体来说，在第一个字符时，使用线性变换将其与权重矩阵 W_x 相乘；对于后续字符，使用线性变换将其与权重矩阵 W_x 相乘，并将前一个字符的隐藏状态与权重矩阵 W_h 相乘，然后将两者相加。最终，我将最后一个字符的隐藏状态与权重矩阵 W_o 相乘，并添加偏置项 b_o，得到最终的输出。

在模型的 get_loss 方法中，使用 SoftmaxLoss 函数将模型的输出和正确标签之间的差异转化为损失值。train 方法中，从数据集中获取批次数据，并计算损失值及其对于各参数的梯度；使用梯度下降算法更新模型的参数。在每次迭代中，检查验证集的准确率是否达到要求，若达到准确率要求便结束训练。

```
class LanguageIDModel(object):
    """
    A model for language identification at a single-word granularity.

    (See RegressionModel for more information about the APIs of different
    methods here. We recommend that you implement the RegressionModel before
    working on this part of the project.)
    """
    def __init__(self):
        # Our dataset contains words from five different languages, and the
        # combined alphabets of the five languages contain a total of 47
unique
        # characters.
        # You can refer to self.num_chars or len(self.languages) in your code
        self.num_chars = 47
        self.languages = ["English", "Spanish", "Finnish", "Dutch", "Polish"]

        # Initialize your model parameters here
        "*** YOUR CODE HERE ***"
        # 隐层的大小
        self.h = 256
```

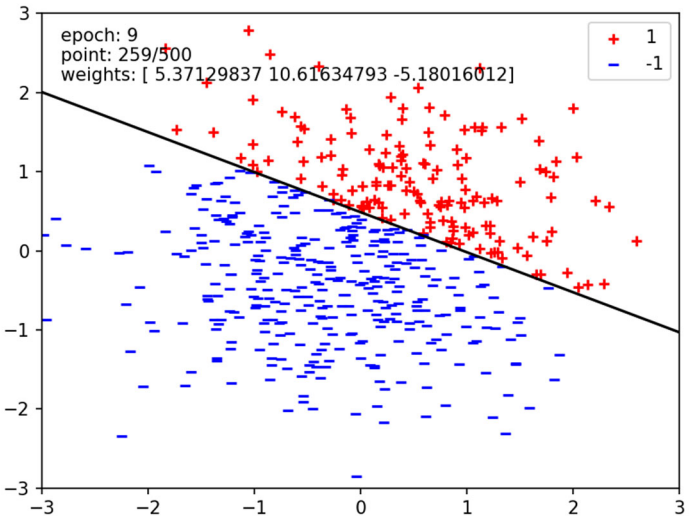
```
        # 输出层的大小
        self.o = 5

        self.learning_rate = 0.05
        self.batchsize = 100

        self.Wx = nn.Parameter(self.num_chars, self.h)
        self.Wh = nn.Parameter(self.h, self.h)

        self.Wo = nn.Parameter(self.h, self.o)
        self.bo = nn.Parameter(1, self.o)

    def run(self, xs):
        """
        Runs the model for a batch of examples.

        Although words have different lengths, our data processing guarantees
        that within a single batch, all words will be of the same length (L).

        Your model should use a Recurrent Neural Network to summarize the
list
        `xs` into a single node of shape (batch_size x hidden_size), for your
        choice of hidden_size. It should then calculate a node of shape
        (batch_size x 5) containing scores, where higher scores correspond to
        greater probability of the word originating from a particular
language.

        Inputs:
            xs: a list with L elements (one per character), where each
element
                is a node with shape (batch_size x self.num_chars)
        Returns:
            A node with shape (batch_size x 5) containing predicted scores
                (also called logits)

        "*** YOUR CODE HERE ***"
        for index in range(len(xs)):
            if(index == 0):
                layer = nn.ReLU(nn.Linear(xs[index], self.Wx))
            else:
```

```
                layer = nn.ReLU(nn.Add(nn.Linear(xs[index], self.Wx),
nn.Linear(layer, self.Wh)))

        return nn.AddBias(nn.Linear(layer, self.Wo), self.bo)


    def get_loss(self, xs, y):
        """
        Computes the loss for a batch of examples.

        The correct labels `y` are represented as a node with shape
        (batch_size x 5). Each row is a one-hot vector encoding the correct
        language.

        Inputs:
            xs: a list with L elements (one per character), where each
element
                is a node with shape (batch_size x self.num_chars)
            y: a node with shape (batch_size x 5)
        Returns: a loss node
        """
        "*** YOUR CODE HERE ***"
        return nn.SoftmaxLoss(self.run(xs), y)

    def train(self, dataset):
        """
        Trains the model.
        """
        "*** YOUR CODE HERE ***"

        for x, y in dataset.iterate_forever(self.batchsize):
            loss = self.get_loss(x, y)
            grad_Wx, grad_Wh, grad_Wo, grad_bo = nn.gradients(loss, [self.Wx,
self.Wh, self.Wo, self.bo])
            self.Wx.update(grad_Wx, -self.learning_rate)
            self.Wh.update(grad_Wh, -self.learning_rate)
            self.Wo.update(grad_Wo, -self.learning_rate)
            self.bo.update(grad_bo, -self.learning_rate)
            if(dataset.get_validation_accuracy() > 0.88):
                break
```

4. 实验结果

4.1. 问题 1 感知机

问题 1 的测试样例是平面点集的二分类问题。神经网络的输入是二维的点坐标，输出是代表划分的 1 或-1。如下图所示，可以观察到在 9 个 epoch 之后，感知机模型能够正确地对图中红色与蓝色的所有点进行分类。

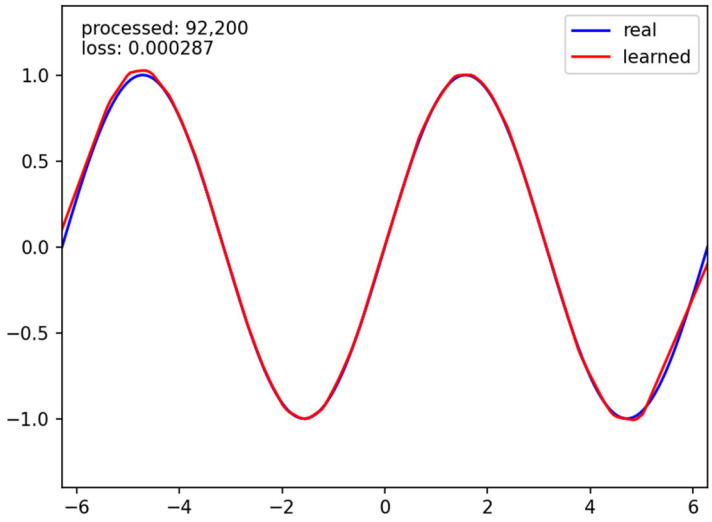


通过自动评分，截图如下：

```
*** PASS: check_perceptron

### Question q1: 6/6 ###

Finished at 16:01:47

Provisional grades
==================
Question q1: 6/6
------------------
Total: 6/6
```

4.2. 问题 2 正弦函数拟合

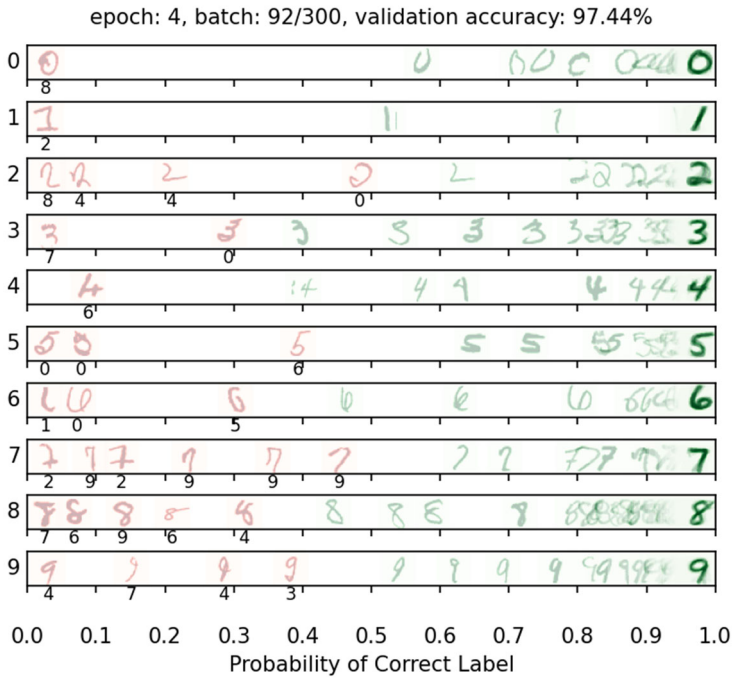
问题 2 中，需要通过神经网络对[-2π, -2π]区间上的正弦函数进行拟合，即建立从输入实数到输出实数的映射关系。在经历几轮学习后，模型能够很好地对正弦函数进行拟合，误差率较低，可视化图像如下：



以 0.001242 的损失通过自动评分，截图如下：

```
Your final loss is: 0.001242
*** PASS: check_regression

### Question q2: 6/6 ###

Finished at 16:17:31

Provisional grades
==================
Question q2: 6/6
------------------
Total: 6/6
```

4.3. 问题 3 手写数字分类



问题 3 中，需要实现对 MNIST 数据集上手写数据图片的数字分类，即将输入的 28x28 图像对应的向量转换为“0”到“9”的数字分类。根据上页图可以观察到，经历 4 个 epoch 后，模型在训练集上的准确率就超过了 97%。在训练集准确率高于 98%时停止模型训练，进行测试集上的验证。

通过自动评分，最终在测试集上的准确率为 97.7%，截图如下：

```
Your final test set accuracy is: 97.700000%
*** PASS: check_digit_classification

### Question q3: 6/6 ###

Finished at 16:29:59

Provisional grades
==================
Question q3: 6/6
------------------
Total: 6/6
```

4.4. 问题 4 单词语言分类

```
epoch 18 iteration 8 validation-accuracy 87.0%
  meantime   English ( 85.0%)              |en 85%|es 10%|fi  1%|nl  4%|pl  1%
  courthouse English ( 88.0%)              |en 88%|es  0%|fi  0%|nl 11%|pl  0%
  dispose    English ( 65.6%)              |en 66%|es 29%|fi  0%|nl  4%|pl  1%
  mercancía  Spanish ( 99.9%)              |en  0%|es100%|fi  0%|nl  0%|pl  0%
  sabiduría  Spanish ( 99.9%)              |en  0%|es100%|fi  0%|nl  0%|pl  0%
  puente     Spanish ( 90.5%)              |en  5%|es 91%|fi  1%|nl  2%|pl  1%
  vapaasti   Finnish ( 99.9%)              |en  0%|es  0%|fi100%|nl  0%|pl  0%
  normaalia  Finnish ( 86.6%)              |en  1%|es  7%|fi 87%|nl  5%|pl  1%
  väitti     Finnish (100.0%)              |en  0%|es  0%|fi100%|nl  0%|pl  0%
  doorheen   Dutch   ( 86.2%)              |en 12%|es  0%|fi  2%|nl 86%|pl  0%
  brutaal    Dutch   ( 81.0%)              |en 15%|es  1%|fi  2%|nl 81%|pl  1%
  daaruit    Dutch   ( 87.5%)              |en  9%|es  0%|fi  3%|nl 88%|pl  0%
  studia     Polish  ( 48.4%)              |en  1%|es 40%|fi 10%|nl  2%|pl 48%
  wściekły   Polish  (100.0%)              |en  0%|es  0%|fi  0%|nl  0%|pl100%
  wyłączyć   Polish  (100.0%)              |en  0%|es  0%|fi  0%|nl  0%|pl100%
Your final test set accuracy is: 85.000000%
```

问题 4 中，对英语、西班牙语、芬兰语、荷兰语、波兰语等五种语言单词进行语言识别。在 18 个 epoch 之后，模型在训练集上的准确率可以达 87%。观察上图可以得到，所有的单词均被正确地分类，且正确分类的语言预测概率较高，大部分稳定在 80%以上。可以看出，模型对单词语言分类的效果较好。

最终以 85%的准确率通过测试集，截图如下：

```
Your final test set accuracy is: 85.000000%
*** PASS: check_lang_id

### Question q4: 7/7 ###

Finished at 19:30:43

Provisional grades
==================
Question q4: 7/7
------------------
Total: 7/7
```

5. 总结与分析

在本次 Project5 实验中，我了解了深度学习中一些基本的神经网络模型，分别完成了感知机二类分类、正弦函数回归、手写数字分类和单词语言任务。通过本次实验，我对机器学习中常见的几种模型有了更深入的了解，并通过实践加深了对这些模型的应用理解。

针对问题 1 中感知机二类分类的任务，我实现了一个感知机模型。通过这个实验，我深入了解了感知机模型的原理和训练过程，并在实践中加深了对其应用的理解。针对问题 2 中的正弦函数回归任务，我学习了建立包含输入层、隐藏层和输出层的神经网络模型，并通过反向传播算法来更新模型的权重参数，从而拟合在给定区间内的正弦函数。针对问题 3 中的手写数字分类任务，我实现了一个用于对手写数字进行分类的神经网络模型。通过实验中的训练过程，我了解了神经网络模型的原理和训练方法，并通过实践加深了对神经网络在图像分类中应用的理解。针对问题 4 中的单词语言分类任务，我设计了一个循环神经网络模型（RNN）以对单词进行语言分类。该模型利用循环神经网络的记忆性质，能够捕捉到单词中的语言特征，并将其用于分类。在设计与搭建 RNN 的过程中，我逐步了解了循环神经网络模型的结构和训练过程，并通过实践掌握了如何使用循环神经网络模型进行自然语言处理任务。下面我将分别对本次实验中应用的算法进行总结分析：

5.1. 对问题 1 中感知机的总结分析

感知机是一种用于二类分类的线性分类模型，通过调整权重和阈值来实现对数据的分类。在感知机算法中，我们通过定义一个线性函数，将输入的特征向量与权重进行点积运算，并与阈值进行比较，得到分类结果。如果点积大于等于阈值，将数据分为正类；否则，将数据分为负类。在训练过程中，对于每个训练样本，如果感知机对该样本的分类结果与标签不一致，则更新权重，使得感知机更加准确地对样本进行分类。这样不断迭代训练，直到所有样本都被正确分类，就可以成功的实现二分类的任务。在这部分的实验过程

中，我整体完成的比较顺利，没有遇到太多的问题。

感知机算法是一种线性分类模型，其具有简单而直观的特点。感知机算法每次权重更新只考虑一个样本，因此算法能够实时地适应数据的变化。但是，它仅适用于线性可分问题，在处理线性不可分的数据集时会无法收敛或无法找到合适的决策边界。此外，感知机算法对噪声数据较为敏感，如果数据中存在较多的噪声点，可能会导致感知机分类错误。因此在实际应用中，感知机的应用范围是十分有限的。下面分析的其他神经网络有更多的隐藏层，且引入了激活层来解决非线性的问题，具有更广泛和强大的应用能力。

5.2. 对问题 2、3 中多层感知机（MLP）的总结分析

多层感知机（Multilayer Perceptron）是一种常用的神经网络模型，具有多个隐藏层的结构。每个隐藏层包含多个神经元，神经元之间通过权重连接。多层感知机通过将输入数据传递给隐藏层，并经过一系列非线性变换和激活函数处理，最终输出预测结果。

在正弦函数回归实验中，我使用了多层感知机模型来拟合正弦函数。模型的输入是实数，输出也是实数，需要通过训练学习到合适的权重和偏置来拟合正弦函数的曲线。在实验过程中，我首先初始化了模型的参数，包括输入维度、隐藏层大小和输出大小。然后，我通过定义模型的前向传播方法来实现正向计算，逐层进行线性变换和激活函数操作，得到最终的预测结果。通过对损失函数进行梯度下降优化，不断更新模型的参数，使得模型能够更准确地拟合正弦函数的曲线。在训练过程中，本次 Project5 的问题 2 到问题 4 都使用了批量训练的方式，每次训练一批样本以提高训练效率。

在手写数字分类的实验中，我同样采用了多层感知机模型来实现分类任务。该任务的输入是由 28x28 像素的灰度图像展开成的 784 维向量，输出是表示数字类别的 10 维向量。我通过定义模型的前向传播方法来实现预测过程，通过多层的线性变换和激活函数操作，将输入映射到 10 个输出节点，每个节点表示对应数字类别的得分。在损失函数选择上，使用 Softmax Loss 衡量预测结果与真实类别之间的差异。通过对损失函数进行梯度下降优化，不断更新模型的参数，使得模型能够更准确地分类手写数字。

从这两个问题中我不难发现，较问题 1 中的感知机，多层感知机模型具有较强的表达能力，能够适应复杂的函数拟合和分类任务。此外，多层感知机模型也可以通过调整隐藏层的大小和层数来适应不同的任务需求。然而，多层感知机模型也存在一些局限性。例如，模型的训练过程需要大量的数据和计算资源，时间复杂度较高。同时，模型的结果也受到选择合适的超参数和优化算法的影响。当模型设计的复杂或学习率设置不合适，也可能会导致训练困难或不稳定。这就需要采取重新进行模型设计、调整学习率或批次大小等超参数等措施来提升其性能。

通过这两个问题的实践，我对多层感知机模型有了更深入的理解和应用。在实验过程

中，我也遇到了一些问题，并采取了相应的解决方案。例如，起初在拟合正弦函数时，我错误地在输出层后也加了一层 ReLU 激活层，这就导致了模型无法拟合函数值为负时的正弦函数。在调试的过程中，我也更熟练地掌握了 MLP 的相关知识。

5.3. 对问题 4 中循环神经网络（RNN）的总结分析

循环神经网络（RNN）是一种在序列数据中进行建模的神经网络结构。与传统的前馈神经网络不同，RNN 具有反馈连接，可以将先前的状态信息传递到当前状态，从而对序列中的上下文进行建模。由于其能够捕捉到语言中的时序和语境信息的特性，RNN 在语言识别等任务中表现出色。

RNN 的结构包含一个隐藏层，它在不同时间步共享相同的参数。在单词语言识别任务中，输入是由单词的字符组成的序列，每个字符用独热向量表示。RNN 通过将当前字符的输入与前一个时间步的隐藏状态进行运算，得到当前时间步的隐藏状态。隐藏状态可以看作是模型对于之前序列信息的编码，它捕捉了序列中的上下文关系。然后，通过将隐藏状态与输出层的参数进行运算，得到预测的结果向量，表示每种语言的可能性。

在实验中，我使用了一个包含一个隐藏层的 RNN 模型进行单词语言识别。模型的隐藏层大小为 256，输出层的大小为 5，对应着 5 种语言。模型的参数包括输入层到隐藏层的权重矩阵 W_x，隐藏层到隐藏层的权重矩阵 W_h，隐藏层到输出层的权重矩阵 W_o，以及输出层的偏置向量 b_o。训练过程中，使用批量梯度下降法来更新模型的参数。对于每个训练样本，计算损失函数，即预测结果与真实标签之间的差异。然后，通过对损失函数进行反向传播，计算参数的梯度，并根据学习率进行更新。

在实验过程中，我也遇到了一些困难和挑战。由于是第一次接触循环神经网络，理解 RNN 的结构和原理就需要一定的时间和学习成本。此外，选择合适的超参数（如隐藏层大小、学习率等）也是一个挑战，这就需要进行反复实验和调整。

总的来说，RNN 是一种强大的模型，适用于处理序列数据和语言识别等任务。它能够捕捉上下文信息，因此适用于解决自然语言处理领域的许多问题。然而，RNN 也存在一些限制和挑战，例如处理长期依赖和梯度消失问题。近年来，出现了如长短期记忆网络（LSTM）等一些改进的 RNN 结构以缓解这些问题。因此，在实际应用中可以根据具体任务的需求，选择不同的循环神经网络结构和技巧来提高模型性能和效果。

在本次 Project5 实验中，通过完成不同的任务，我学习和实践了感知机、多层感知机、以及循环神经网络等不同的模型与算法。我不仅学习了不同神经网络模型的结构、特点以及适用问题，还深入了解了这些模型的原理和实现方式。通过调整模型的参数与超参数，设计不同的模型结构，调节隐层的数量、神经元的个数等等，我尝试着优化模型的性

能，使它们能够更好地完成题目所给定的任务，最终也都达到了较高的分类或拟合准确率。在此过程中，不管是对模型与算法本身的理解、撰写，还是对 Python 语法的学习与熟悉，不断撰写、纠错、调试的过程都让我逐渐提高了自己的分析问题、寻找解决方案和调试代码的能力，以及在解决问题时需要的耐心和毅力。

总的来说，本次实验使我逐步了解了人工智能领域的各种神经网络模型，并为我未来在该领域的学习和研究奠定了坚实的基础。通过本次实验的学习和实践，我对未来的学习更充满信心，也希望自己在未来能够通过更多的学习与实践不断进步！