同济大学计算机系

人工智能课程设计实验报告

学　　号 2152809

姓　　名 曾崇然

专　　业 计算机科学与技术

授课老师 武妍老师

一. 问题概述

1. 直观描述

在这次的实验中，需要通过搭建不同的神经网络来完成一些不同的任务，包括一个二进制感知机，实现对已有结点的分类；模拟近似一个正弦函数；训练一个网络来识别手写的数字；训练一个网络来识别一个单词属于哪种语言。以上的任务都要求识别的准确率或者模拟的精度达到一定的程度。

2. 已有代码的阅读和理解

a) 需要完成的代码 models.py

PerceptronModel 类：包含训练一个二进制感知机的方法的类

RegressionModel 类：实现用神经网络拟合近似正弦函数的方法

DigitClassificationModel 类：训练神经网络来实现手写数字的识别

LanguageIDModel 类：训练神经网络来识别一个单词属于哪种语言

__init__函数：初始化参数，包括不同层神经网络的权重矩阵和偏移矩阵

run 函数：使用现有的参数来预测对应输入的输出

get_loss 函数：返回现有参数预测的损失

train 函数：从数据集中取数据，使用参数来进行预测对应输出，计算损失值，并根据损失值调整参数矩阵的值，重复这个过程直到精度到达一定程度

b) 需要参考的代码 nn.py

包含一些用于训练神经网络的库函数：

nn.Parameter 代表一个可训练的感知器或神经网络的参数。

nn.DotProduct 计算其输入的点积。

nn.as_scalar 可以从节点中提取一个 Python 浮点数。

nn.Add 按元素对矩阵进行加法。

nn.AddBias 将偏置向量添加到每个特征向量。

nn.Linear 对输入应用线性变换（矩阵乘法）。

nn.ReLU 应用逐元素的修正线性单元非线性函数

nn.SoftmaxLoss 计算批次的 softmax 损失，用于分类问题。

nn.gradients 计算相对于提供的参数的损失梯度。

c) 需要参考的代码 backend.py

通过调用 dataset.iterate_once（batch_size）来获取训练示例的批次.

dataset.iterate_forever(batch_size)生成一个无限的批次示例序列。

dataset.get_validation_accuracy()返回模型在验证集上的准确率。

3. 解决问题的思路与方法

a) 二进制感知机

通过计算输入和参数的点积并以此计算出输出预测值，根据预测值和真实值的差别来调整参数，直到数据集中所有数都能够被正确分类

b) 近似正弦函数

搭建一个神经网络，根据这个神经网络中的参数值来计算输入对应的输出值，根据预测值的损失来对参数进行更新，直到损失值低到一定程度，这表明拟合的精度到达了一定的高度

c) 手写数字识别

搭建一个神经网络，根据这个神经网络对输入的手写图片进行映射，映射到 0-9 的分类中，不断的根据损失值来调整参数的值，直到准确率到达一定的水准

d) 语言分类

搭建一个神经网络，每读入一个字母，就使用输入层对其进行映射，并使用隐藏层对之前得到的映射结果进行映射，将二者之和作为新的映射结果，直到最后一个字母读入完毕，再使用输出层映射得到最后的预测结果，根据损失值不停调整参数值，直到精度到达一定的程度

二. 算法设计

1. 算法功能

a) 二进制感知机

对输入结点的 x 用参数矩阵进行二分类，结果为-1 或 1，根据结果调整参数的值，使得分类更加准确

b) 近似正弦函数

对输入的结点 x 用神经网络进行近似，使其输出结果逼近一个正弦函数，不断根据结果更新神经网络中的参数值，使近似更加精确

c) 手写数字识别

对输入的手写数字图片的像素信息使用神经网络进行映射，使映射结果为手写图片对应的数字，不断根据分类的结果修改神经网络中的参数，使分类结果更加精确

d) 语言分类

对输入的单词使用神经网络进行映射，将其所属的语言进行识别，根据结果修改参数值，使识别更加精确

2. 设计思路

a) 二进制感知机

初始化参数矩阵，计算输入和参数矩阵的点积并据此对输入进行分类，判断分类的预测值与真实值是否相同，若不相同则调整参数值，直到数据集中所有的数据都能够被正确的分类，结束。

b) 近似正弦函数

搭建神经网络，包含一个隐藏层，确定神经网络的深度和大小，使用该神经网络对输入进行映射，根据损失值来进行参数的调整，直到近似的精度到达一定的程度，结束。

c) 手写数字识别

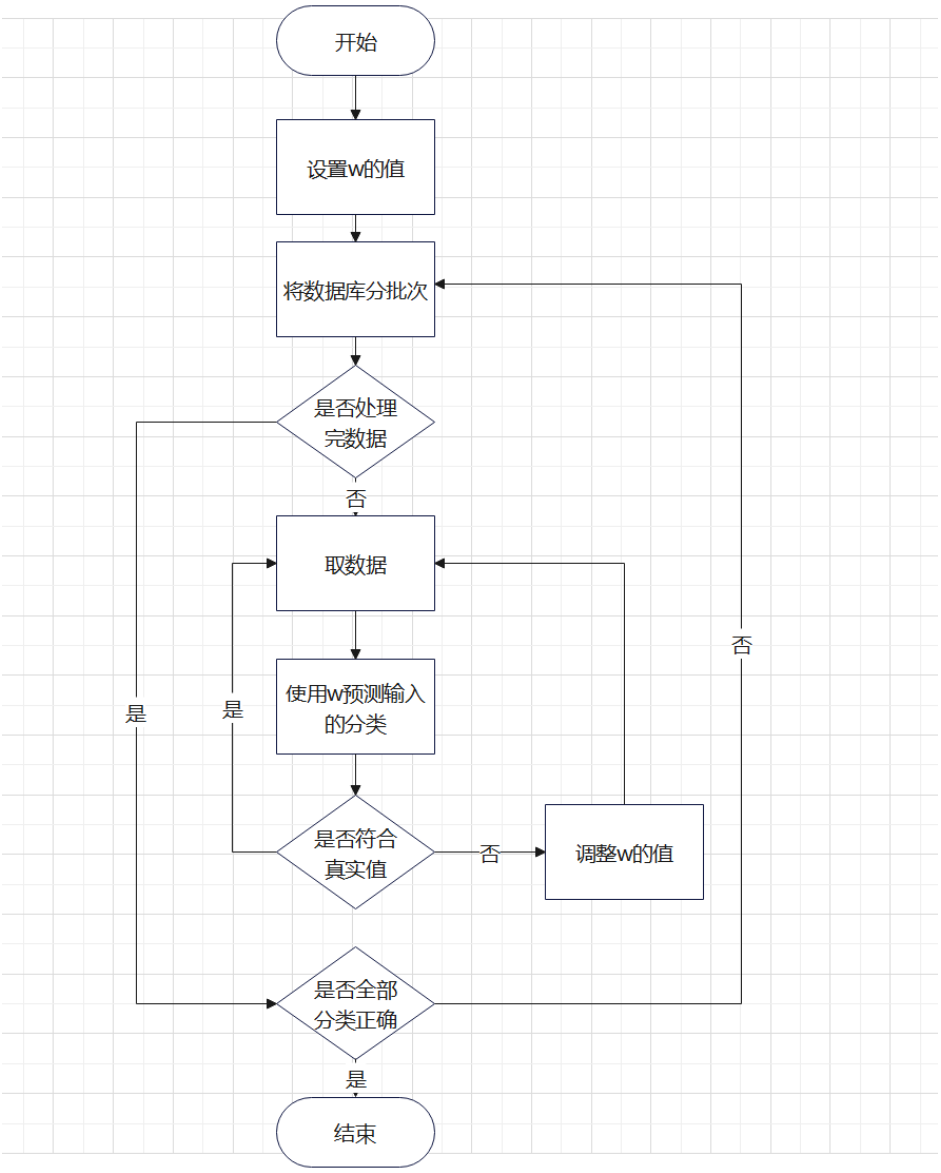
搭建神经网络，确定神经网络的深度和隐藏层的大小，使用该神经网络对输入结点进行映射，识别是哪一个数字，根据损失值调整参数，直到识别的精度到达一定的程度，结束。

d) 语言分类

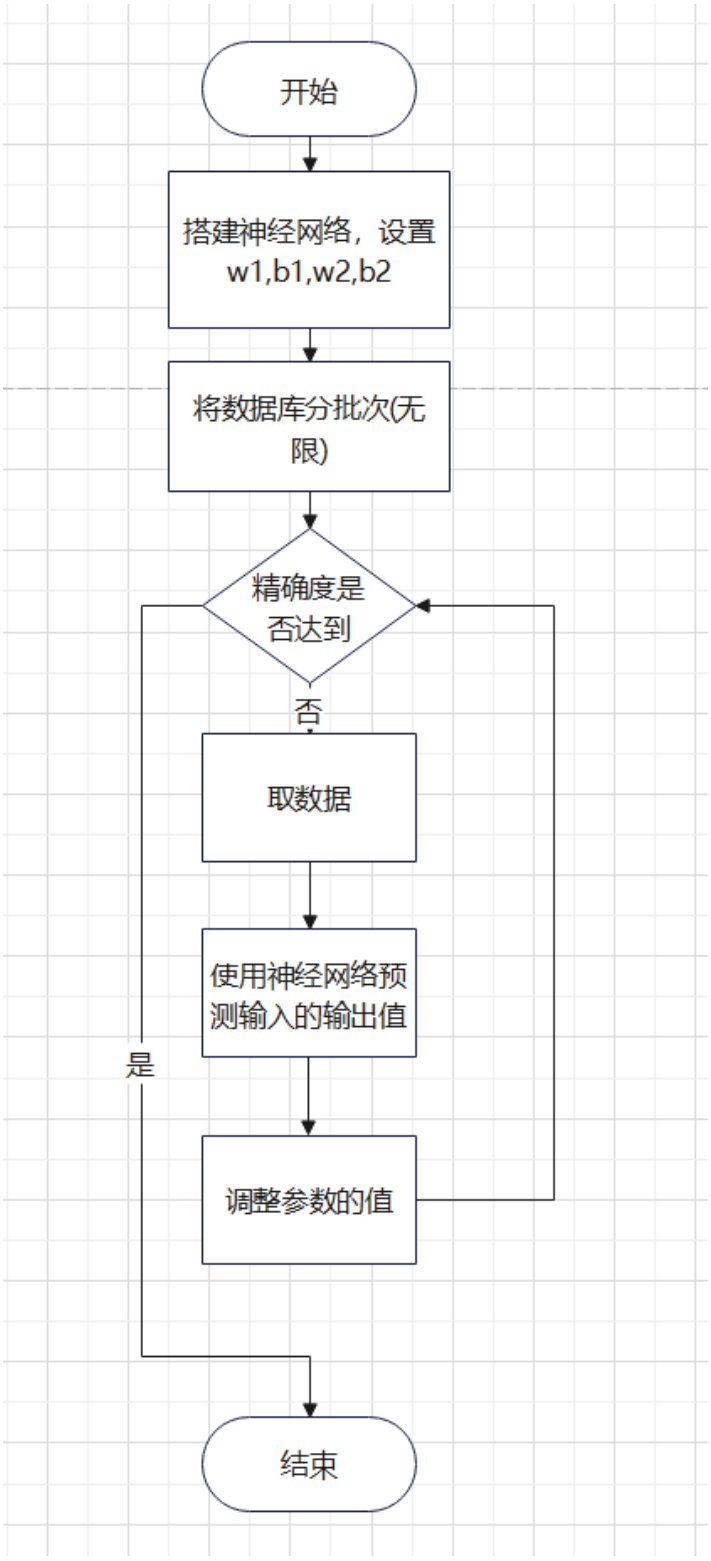
搭建神经网络，包含三个部分，输入层，隐藏层，输出层，不断的读入字符直到单词被读取完，在读取的过程中反复的使用输入层和隐藏层进行映射得到新的结果，在读入完毕之后将得到的结点使用输出层映射输出，根据损失值调整参数，直到准确率达到一定程度，结束。

3. 流程图

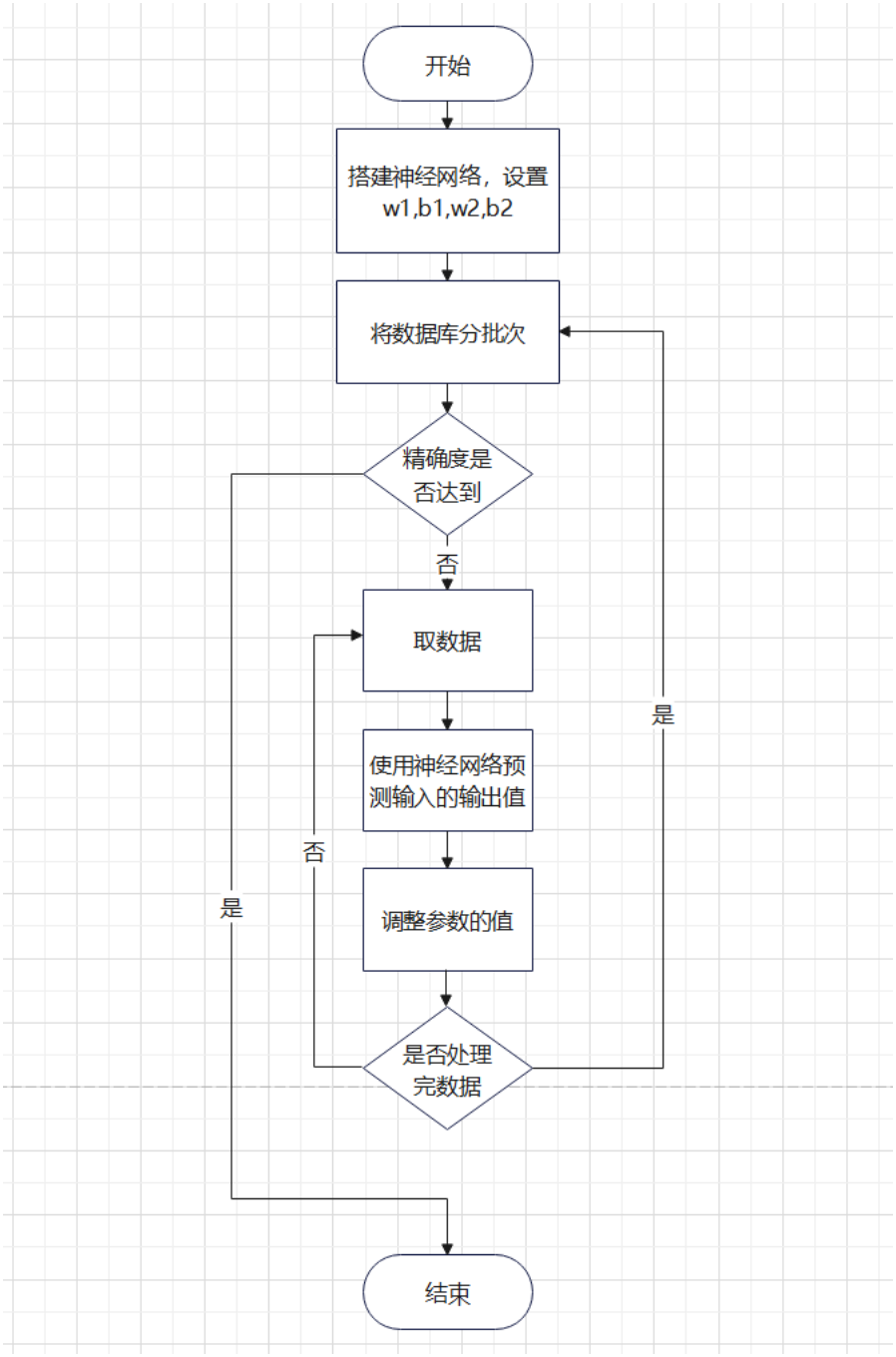
a) 二进制感知机


开始
设置w的值
将数据库分批次
是否处理完数据
否
取数据
使用w预测输入的分类
是否符合真实值
否
调整w的值
是
是
否
是否全部分类正确
是
结束

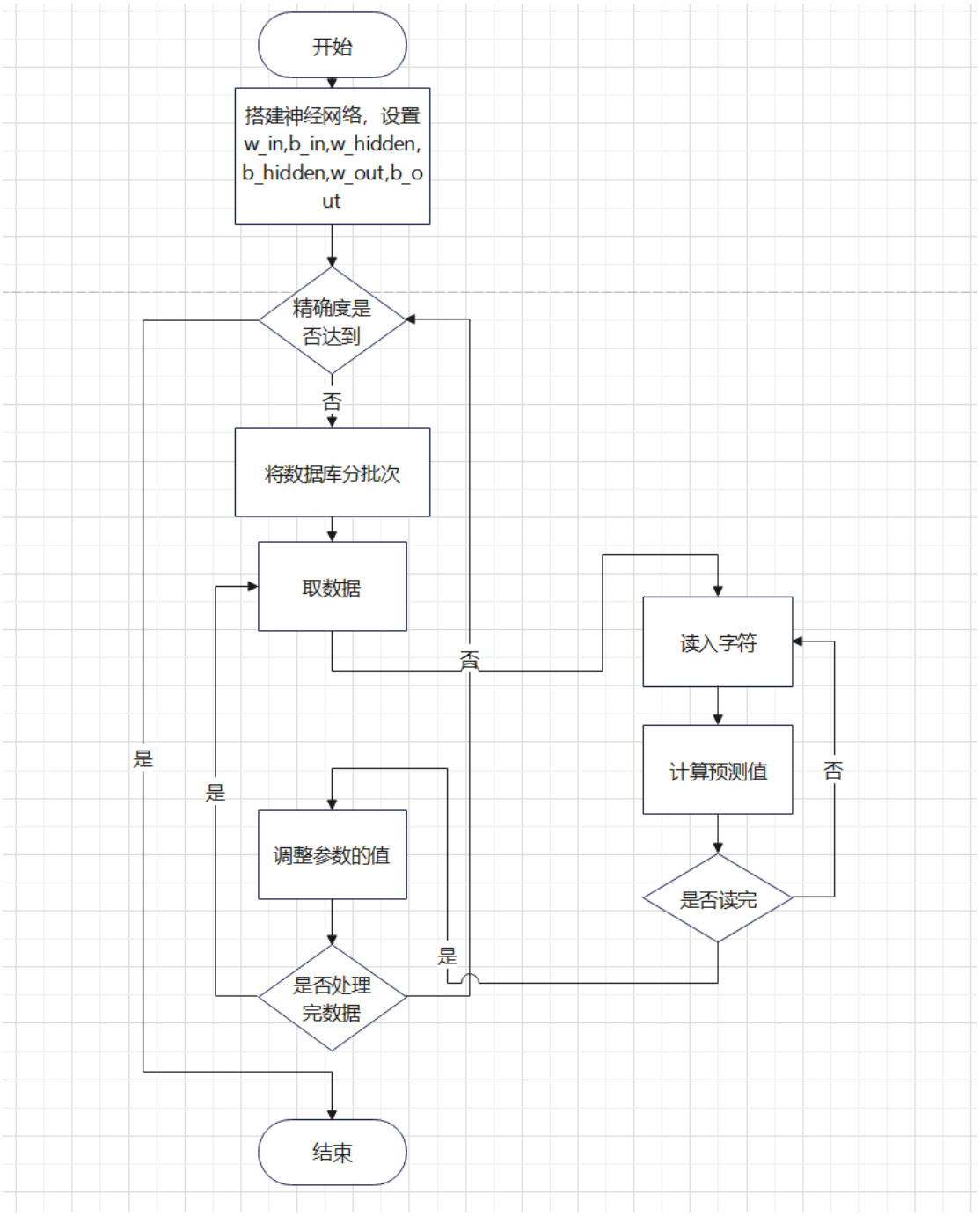
b) 近似正弦函数


开始
搭建神经网络，设置w1,b1,w2,b2
将数据库分批次(无限)
精确度是否达到
否
取数据
使用神经网络预测输入的输出值
是
调整参数的值
结束

c) 手写数字识别


开始
搭建神经网络，设置w1,b1,w2,b2
将数据库分批次
精确度是否达到
否
取数据
使用神经网络预测输入的输出值
调整参数的值
是否处理完数据
否
是
是
结束

d) 语言分类



三. 算法实现

1. 实现细节

a) 二进制感知机

权重矩阵的大小设置时已经给定的，同时已经规定 batchs 的值为 1，即每次处理一个数据，因此在该题目中无需自己去搭建神经网络，只需要不停的更新参数的值直到符合要求即可。

b) 近似正弦函数

神经网络的搭建：参考题目给出的参数设置建议，我搭建的神经网络有一个隐藏层（两个线性层），线性层间用非线性的 relu 进行分隔，同时将隐藏层的大小设置为 512，参数如下图：

```
self.w1 = nn.Parameter(1, 512)
self.b1 = nn.Parameter(1, 512)
self.w2 = nn.Parameter(512, 1)
self.b2 = nn.Parameter(1, 1)
```

结构如下图：

```
hidden = nn.ReLU(nn.AddBias(nn.Linear(x, self.w1), self.b1))
output = nn.AddBias(nn.Linear(hidden, self.w2), self.b2)
```

batch size 和学习率的设置：根据题目的推荐，我将 batch size 设置为 200，学习率设置为 0.05，但是发现准确率达不到要求的水平，于是我修改了学习率，将其改为 0.01，使得拟合更加的精准，达到了要求（可能时题目的推荐没有和题目要求吻合？）

c) 手写数字识别

神经网络的搭建：根据题目的推荐，我搭建的神经网络具有一个隐藏层（两个线性层，用非线性的 relu 分隔）我将隐藏层的大小设置为 200，参数设置如下图：

```
self.w1 = nn.Parameter(784, 200)
self.b1 = nn.Parameter(1, 200)
self.w2 = nn.Parameter(200, 10)
self.b2 = nn.Parameter(1, 10)
```

结构如下图：

```
hidden = nn.ReLU(nn.AddBias(nn.Linear(x, self.w1), self.b1))
output = nn.AddBias(nn.Linear(hidden, self.w2), self.b2)
```

batch size 和学习率的设置：根据题目的推荐，我将 batch size 设置为 100，学习率设置为 0.5

d) 语言分类

神经网络的搭建：在处理单词时，由于单词长度的不同，不便于直接通过类似于上两题的神经网络进行映射，因此采用循环神经网络的方法进行映射：每读入一个字母，使用输入层进行映射，将结果保留，在下一次读入字母时，将上一次的结果使用隐藏层进行映射，读入的字母使用输入层进行映射，将两个矩阵求和作为这次读入的结果，重复上述操作，直到将字母读取完为止，最后将结果用输出层映射输出，为了保证非线性，每层之间用 relu 分隔（输出层不添加 relu），参数设置如下图（因为要保证隐藏层足够大，所以我设置隐藏层为 512）：

```
self.w_in = nn.Parameter(47, 512)
self.b_in = nn.Parameter(1, 512)
self.w_hidden = nn.Parameter(512, 512)
self.b_hidden = nn.Parameter(1, 512)
self.w_out = nn.Parameter(512, 5)
self.b_out = nn.Parameter(1, 5)
```

结构如下图：

```
for i in range(len(xs)):
    if i == 0:
        h = nn.ReLU(nn.AddBias(nn.Linear(xs[i], self.w_in), self.b_in))
    else:
        h = nn.ReLU(nn.AddBias(nn.Add(nn.Linear(xs[i], self.w_in), nn.Linear(h, self.w_hidden)), self.b_hidden))
return nn.AddBias(nn.Linear(h, self.w_out), self.b_out)
```

batch size 和学习率设置：我尝试设置 batch size 为 50，学习率为 0.1

2. **核心函数**

a) **二进制感知机**

run 函数：

```
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(x, self.get_weights())
```

train 函数：

```
def train(self, dataset):
    """
    Train the perceptron until convergence.
    """
    "*** YOUR CODE HERE ***"
    while 1:
        flag = 1;
        for x, y in dataset.iterate_once(1):
            if self.get_prediction(x) != nn.as_scalar(y):
                nn.Parameter.update(self.w, x, nn.as_scalar(y))
                flag = 0;
        if flag == 1:
            break
```

b) **近似正弦函数**

run 函数：

```python
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 ***"
    hidden = nn.ReLU(nn.AddBias(nn.Linear(x, self.w1), self.b1))
    output = nn.AddBias(nn.Linear(hidden, self.w2), self.b2)
    return output
```

train 函数：

```python
def train(self, dataset):
    """
    Trains the model.
    """
    "*** YOUR CODE HERE ***"
    parts = dataset.iterate_forever(200)
    for x, y in parts:
        loss = self.get_loss(x, y)
        if nn.as_scalar(loss) < 0.01:
            break
        grad_wrt_w1, grad_wrt_b1, grad_wrt_w2, grad_wrt_b2 = nn.gradients(loss, [self.w1, self.b1, self.w2, self.b2])
        self.w1.update(grad_wrt_w1, -0.01)
        self.b1.update(grad_wrt_b1, -0.01)
        self.w2.update(grad_wrt_w2, -0.01)
        self.b2.update(grad_wrt_b2, -0.01)
```

c) 手写数字识别

run 函数：

```python
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 ***"
    hidden = nn.ReLU(nn.AddBias(nn.Linear(x, self.w1), self.b1))
    output = nn.AddBias(nn.Linear(hidden, self.w2), self.b2)
    return output
```

train 函数：

```
def train(self, dataset):
    """
    Trains the model.
    """
    "*** YOUR CODE HERE ***"
    while 1:
        if dataset.get_validation_accuracy() > 0.975:
            break
        parts = dataset.iterate_once(100)
        for x, y in parts:
            loss = self.get_loss(x, y)
            grad_wrt_w1, grad_wrt_b1, grad_wrt_w2, grad_wrt_b2 = nn.gradients(loss, [self.w1, self.b1, self.w2, self.b2])
            self.w1.update(grad_wrt_w1, -0.5)
            self.b1.update(grad_wrt_b1, -0.5)
            self.w2.update(grad_wrt_w2, -0.5)
            self.b2.update(grad_wrt_b2, -0.5)
```

d) 语言分类

run 函数：

```
for i in range(len(xs)):
    if i == 0:
        h = nn.ReLU(nn.AddBias(nn.Linear(xs[i], self.w_in), self.b_in))
    else:
        h = nn.ReLU(nn.AddBias(nn.Add(nn.Linear(xs[i], self.w_in), nn.Linear(h, self.w_hidden)), self.b_hidden))
return nn.AddBias(nn.Linear(h, self.w_out), self.b_out)
```

train 函数：

```
def train(self, dataset):
    """
    Trains the model.
    """
    "*** YOUR CODE HERE ***"
    while 1:
        if dataset.get_validation_accuracy() > 0.9:
            break
        for x, y in dataset.iterate_once(50):
            grad_wrt_w_in, grad_wrt_b_in, grad_wrt_w_hidden, grad_wrt_b_hidden, grad_wrt_w_out, grad_wrt_b_out = nn.gradients(self.get_loss(x, y), [self.w_in
            self.w_in.update(grad_wrt_w_in, -0.1)
            self.b_in.update(grad_wrt_b_in, -0.1)
            self.w_hidden.update(grad_wrt_w_hidden, -0.1)
            self.b_hidden.update(grad_wrt_b_hidden, -0.1)
            self.w_out.update(grad_wrt_w_out, -0.1)
            self.b_out.update(grad_wrt_b_out, -0.1)
```

3. 模块输入输出

a) 二进制感知机

输入数据：x: 一个形状为 (1 x dimensions)的结点

预测结果：1 或者-1（代表分类）

b) 近似正弦函数

输入数据：x: 一批数据，形状为（batch_size x 1）

预测结果：y: 一个包含有对应的预测值的向量，形状同 x 一样

c) 手写数字识别

输入数据：x: 一批数据，行为每批的数据个数，列为 784 个像素点的浮点值

预测结果：y: 一个存有每个数据对应分类的矩阵，行为每批数据的数量，列为分类情况

d) 语言分类

输入数据：xs: 一批数据，一个列表，长度为单词的长度，每个元素是所有单词对应的位置的表示的集合

预测结果：对应的批次内每个数据的分类结果

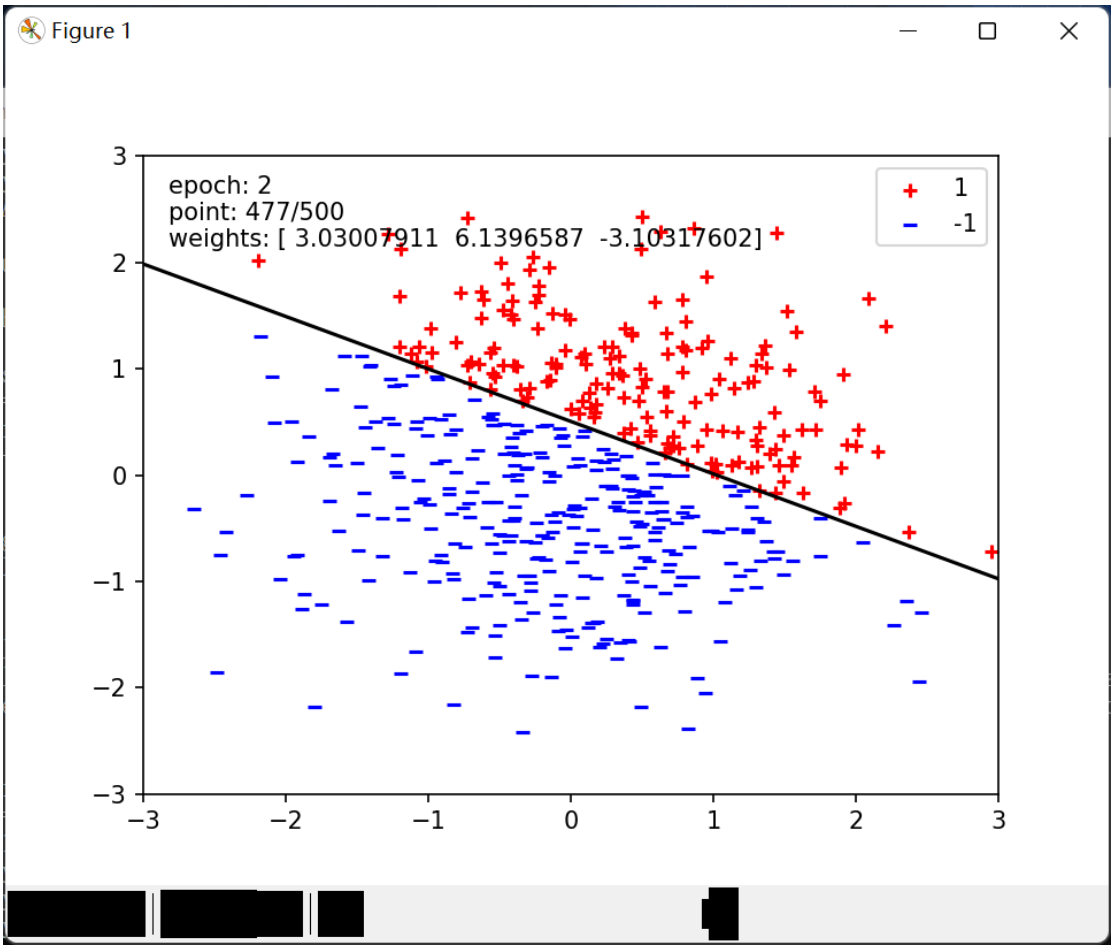
4. 数据结构定义

无数据结构定义

四. 实验结果

1. 结果展示

a) 二进制感知机



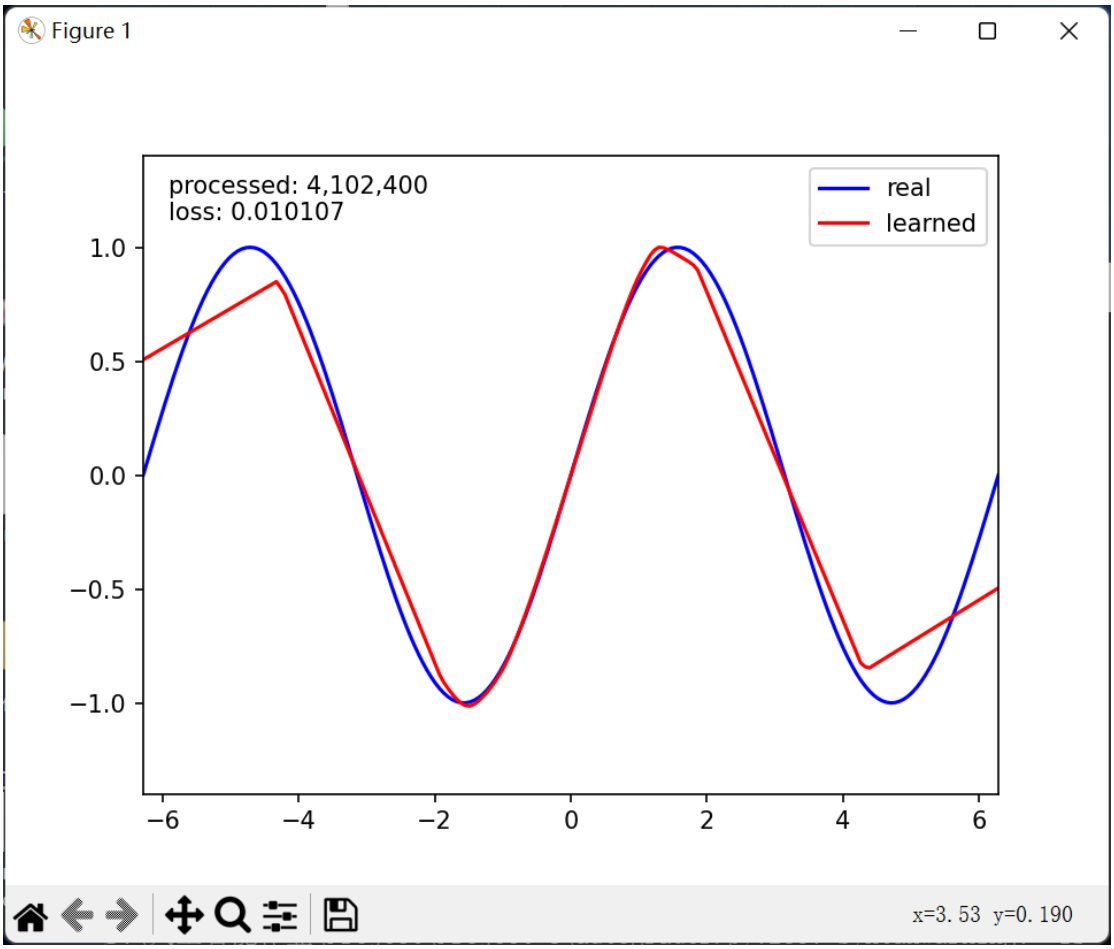

```
Question q1
===========
*** q1) check_perceptron
Sanity checking perceptron...
Sanity checking perceptron weight updates...
Sanity checking complete. Now training perceptron
*** PASS: check_perceptron

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

Finished at 18:05:46

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

b) 近似正弦函数



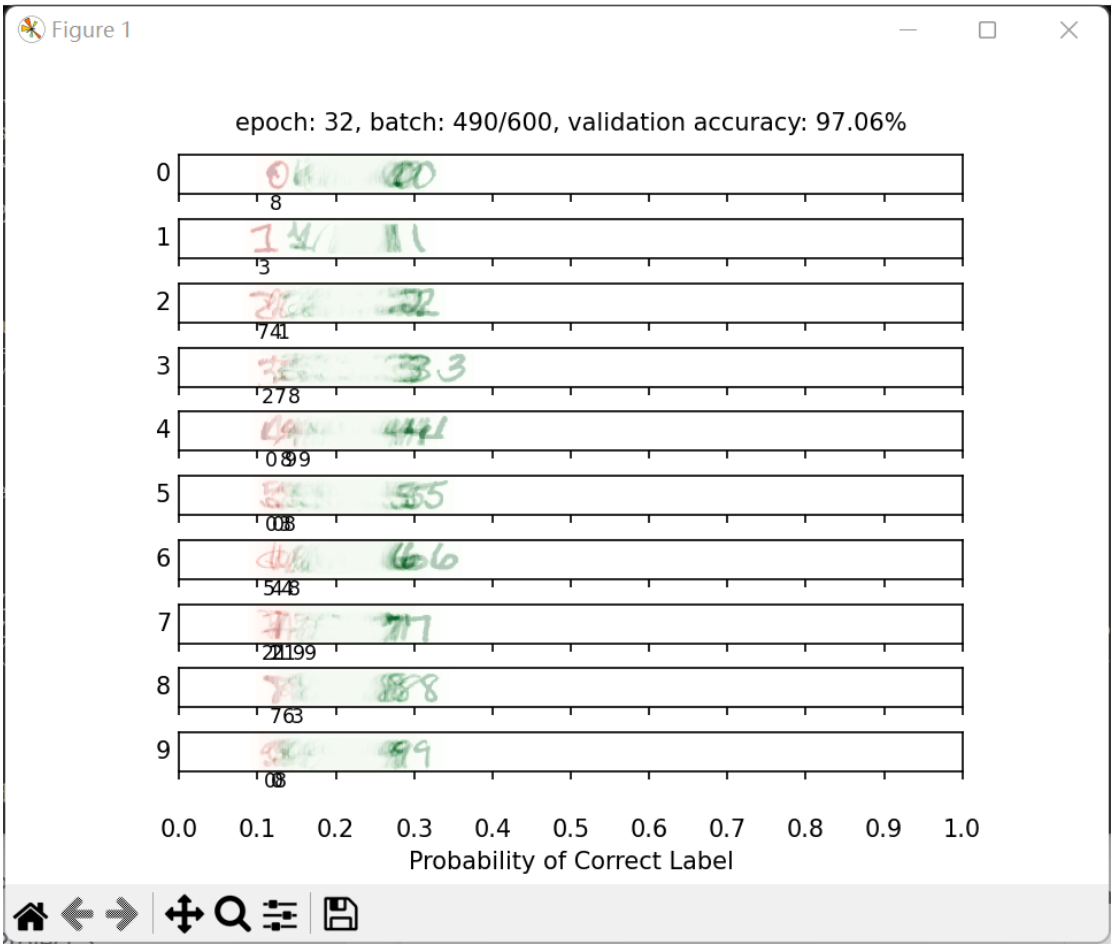

```
Question q2
===========
*** q2) check_regression
Your final loss is: 0.009999
*** PASS: check_regression

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

Finished at 18:07:38

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

c) 手写数字识别




```
Question q3
===========
*** q3) check_digit_classification
Your final test set accuracy is: 97.320000%
*** PASS: check_digit_classification

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

Finished at 18:12:29

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

d) 语言分类

```
epoch 14 iteration 322 validation-accuracy 88.2%
  raging     English ( 73.8%)                |en 74%|es  0%|fi  0%|nl 26%|pl  0%
  fruits     English ( 92.6%)                |en 93%|es  0%|fi  0%|nl  7%|pl  0%
  rides      English ( 98.5%)                |en 99%|es  1%|fi  0%|nl  0%|pl  0%
  equivocas  Spanish ( 99.8%)                |en  0%|es100%|fi  0%|nl  0%|pl  0%
  órganos    Spanish (100.0%)                |en  0%|es100%|fi  0%|nl  0%|pl  0%
  hierba     Spanish ( 12.3%) Pred: Dutch    |en  0%|es 12%|fi  1%|nl 84%|pl  3%
  katsoo     Finnish (100.0%)                |en  0%|es  0%|fi100%|nl  0%|pl  0%
  tulessa    Finnish (100.0%)                |en  0%|es  0%|fi100%|nl  0%|pl  0%
  valitsit   Finnish ( 97.6%)                |en  0%|es  0%|fi 98%|nl  2%|pl  0%
  verspreid  Dutch   ( 99.9%)                |en  0%|es  0%|fi  0%|nl100%|pl  0%
  vuurwerk   Dutch   (100.0%)                |en  0%|es  0%|fi  0%|nl100%|pl  0%
  lagere     Dutch   ( 99.9%)                |en  0%|es  0%|fi  0%|nl100%|pl  0%
  wział      Polish  (100.0%)                |en  0%|es  0%|fi  0%|nl  0%|pl100%
  pannę      Polish  ( 99.4%)                |en  0%|es  0%|fi  1%|nl  0%|pl 99%
  strażnik   Polish  ( 99.9%)                |en  0%|es  0%|fi  0%|nl  0%|pl100%
Your final test set accuracy is: 88.200000%
*** PASS: check_lang_id
```

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

Finished at 18:14:57

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

2. **描述说明**

上面的可视化截图均非最后的结果（因为训练完会自动关闭，所以是在训练过程中的截图）

a) 二进制感知机

可视化界面表示将点分为了两部分（对应 1 和-1），之所以为直线是因为只做了线性的变化，只有一层且没有添加非线性的函数

b) 近似正弦函数

可视化界面中蓝色的线条是正弦函数，红色的线条是近似出来的函数，可以看到随着训练过程的推进，红色线条越发接近蓝色线条，同时损失值也越来越小，表明在训练过程中，神经网络的映射结果能不断的逼近正弦函数

第二张图表明最后的损失值为 0.009999，满足题目要求

c) 手写数字识别

可视化显示了随着训练的进行各个数字识别准确率的变化。

第二张图是结果，最终的准确率为 97.32%，达到了题目的要求

d) 语言分类

第一张截图是部分的训练过程，显示了每个批次的预测结果变化和总准确率的变化，训练完后的准确率为 88.2%

五. 总结与分析

1. **调试过程**

在调试过程中遇到的主要困难是神经网络的搭建，尤其是在搭建最后一个语言识别的时候，其明显区别于前两道题的结构，同时由于题目没有给出一些参数和网络搭建的建议，这道题花费了我很多时间去尝试，但是在尝试的过程中我逐渐理解了循环神经网络的原理，最终搭建了一个三层了网络，并选择了合适的参数，完成了这

道题目。

2. 收获与思考

a) 二进制感知机

理解感知机的基本原理和二进制分类问题。学会使用感知机模型来解决简单的分类任务。掌握参数更新和训练过程中的梯度计算。

b) 模拟正弦函数

掌握如何使用神经网络来拟合和预测函数值。

c) 手写数字识别

了解图像分类问题和数字识别的基本概念和简单方法

d) **识别给定的单词是哪种语言**

理解文本分类问题和语言识别的基本原理。

在这次的实验中，我学习并理解了机器学习的基本概念，了解了神经网络的结果，并能够搭建简单的神经网络来解决一些简单的问题，明白了参数的选择对于机器学习的意义所在。