同济大学

TONGJI UNIVERSITY

# 人工智能 project5 报告


学　　院　电子与信息工程学院

专　　业　计算机科学与技术专业

学生姓名　吕博文

学　　号　2151769

指导教师　王俊丽


日　　期　2023 年 06 月 01 日

# 目 录



装 订 线

# 1 问题概述

## 1.1 问题描述

本项目主要是机器学习的一个简单应用，主要包括四个部分，分别是二分类器的设计，非线性函数的回归拟合，手写数字识别以及语言识别等，主要通过机器学习中的一些简单的实验，由浅入深，通过具体的项目实践使得我们加深对机器学习基本概念的理解以及自己动手完成一些简单的分类器，简单的网络搭建的能力。在本项目中，我们使用项目本身提供的一个小型神经网络库 *nn.py* 来帮助我们搭建网络，解决问题，我们主要完成的是 *model.py* 文件中的四个 *python* 类，编写完成类中的训练函数，损失函数，初始化函数以及网络函数，四个类中需要实现的框架大致类似，具体细节有所不同。

## 1.2 项目已有代码阅读和理解

本次项目中提供了许多已有代码如下：(1)*nn.py*: 神经网络搭建的小型库，可以帮助我们实现一些网络搭建过程中的常见函数，比如损失函数，梯度计算函数，节点构造函数，线性回归函数等等。(2)*model.py*: 内含四个 *python* 类，对应四个 *task*, 是需要我们自己编写的一个文件。(3)*data*: 提供神经网络的训练集和测试集，不需要我们改动。(4)*backend.py*: 提供神经网络内部函数实现的一些细节。

## 1.3 解决问题思路与方法

本项目属于机器学习的一个简单项目实践，在完成该项目前，我们首先需要对基本的神经网络前置知识有足够多的了解，我们需要了解神经网络的基本构成：输入层，隐藏层，输出层；需要了解神经网络激活函数的作用，需要了解神经网络的迭代更新，梯度下降优化网络的算法，具体项目子任务分析如下：

(1)*PerceptronModel*: 简单的二分类模型构建，我们只需要按照题目要求一步步构建网络，补充函数即可完成，主要参考项目本身提供的代码讲解部分 *providedcode*1

(2)*RegressionModel*: 非线性函数拟合问题：因为涉及到非线性函数拟合，我们需要在网络中引入激活函数，使得整个网络具有模拟非线性函数的功能。

(3)*DigitClassificationModel*: 数字识别问题：大部分实现过程和 *task*2 基本一致，因为考虑到识别精度的问题，这个问题我采用了 3 层来搭建网络。

(4)*LanguageIDModel*: 语言文字识别问题：按照题意所说，本题我们采用 *RNN* 算法来改进神经网络中的具体计算，在 *run* 函数的编写中，可以仿照 *task*4 中的给定例子来通过循环实现，其他函数部分和 *task*3 基本一致。

# 2 算法设计

## 2.1 *PerceptronModel* 设计

在 *PerceptronModel* 类中，我们按照题意模拟搭建网络，对于初始化函数，我们构建一个 1 ×*dimensions* 的节点类型；对于 *run* 函数，我们通过 *nn.DotProduct* 返回输入数据 *x* 和设计权重 *w* 的乘积矩阵；对于 $get_prediction$ 我们按题意模拟，节点权重化为浮点型后大于等于 0 则返回 1，否则返回-1；对于 *train* 函数，我们采用循环更新 w 的值直至预测值和标签相一致为止。

## 2.2 *RegressionModel* 设计

在 *RegressionModel* 类设计中，我们需要搭建具有激活函数的网络，在初始化函数中，我们把隐藏层的维度设为 100，学习率设为 0.01，并随机初始化权重矩阵；在 *run* 函数中，我们分别调用 $nn.Linear(), nn.AddBias()$ 函数进行层内线性回归，调用 *nn.ReLU*() 函数进行层之间非线性激活；在 *train*() 函数中，我们按照题意模拟，不断梯度更新网络中的权重值，直至损失率小于 0.01.

## 2.3 *DigitClassificationModel* 设计

在 *DigitClassificationModel* 中，我们同样设计具有激活函数的神经网络，大致框架和 *task*2 一致，不同的是我们在本任务中设计三层网络实现，同时为了加快学习过程，将学习率调至 0.5

## 2.4 *LanguageIDModel* 设计

在 *LanguageIDModel* 设计中，我们主要函数在于 *run* 函数中 *RNN* 算法的编写，这里我们根据项目中的提示，对于输入层的第一个字符，进行线性回归后利用非线性函数激活，对于之后的输入字符，我们进行 RNN 处理，即进行 $z = nn.Add(nn.Linear(x, W), nn.Linear(h, W_hidden))$ 处理，最后对输出层再进行一次线性回归处理。

# 3 算法实现

## 3.1 *PerceptronModel* 实现

```
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.
        """
```

```
        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 ***"
        #返回对应乘积，第一个参数是Node类型，大小batch_size * dimensions
        #第二个参数是权重weights参数，大小是1 * dimensions
        return nn.DotProduct(x,self.w)

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

        Returns: 1 or -1
        """
        "*** YOUR CODE HERE ***"
        #按题意模拟，节点权重乘积化为浮点型后大于等于0则返回1，否则为-1
        if nn.as_scalar(nn.DotProduct(x,self.w))>=0:
            return 1
        else:
            return -1

    def train(self, dataset):
        """
        Train the perceptron until convergence.
        """
        "*** YOUR CODE HERE ***"
        OK = True
        batch_size = 1
        while True:
            OK = True
            for x,y in dataset.iterate_once(batch_size):
                #因为二分类问题中题目确定y只会是1或-1，我们就可以通过
                #x的预测值是否和标签对照来确定是否更新完全
                if self.get_prediction(x) != nn.as_scalar(y):
                    OK = False
```

```
                    self.w.update(x,nn.as_scalar(y))
            if OK:
                break
```

Listing 1 PerceptronModel

通过上述代码可以看出，在 *PerceptronModel* 的实现过程中，我们主要通过一个线性回归的方程进行二分类器的训练，利用 *nn.py* 文件里的构造函数实现网络初始化以及网络训练的过程，在网络权重更新的过程中同样利用 *update* 函数进行梯度更新。

### 3.2 *RegressionModel* 实现

```
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 ***"
        #测试数据取100个
        self.batch_size = 100
        #我们采用两个隐藏层，每个隐藏层中使用100个节点组成,每个层进行线性回归f1 = k1*x+b1,f2 = k2*x+
    b2l
        #层之间采用非线性激活函数Relu来使得整个神经网络模拟非线性函数
        self.k1 = nn.Parameter(1,100)
        self.b1 = nn.Parameter(1,100)
        self.k2 = nn.Parameter(100,1)
        self.b2 = nn.Parameter(1,1)
        #定义学习率
        self.alpha = 0.01

    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 ***"
        #第一层线性回归
        g1 = nn.Linear(x,self.k1)
        f1 = nn.AddBias(g1,self.b1)
        #层之间非线性激活
```

```
        new_f = nn.ReLU(f1)
        #第二层线性回归
        g2 = nn.Linear(new_f,self.k2)
        f2 = nn.AddBias(g2,self.b2)
        return f2


    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:
            for x,y in dataset.iterate_once(self.batch_size):
                loss = self.get_loss(x,y)
                grad = nn.gradients(loss,[self.k1,self.b1,self.k2,self.b2])

                self.k1.update(grad[0],-self.alpha)
                self.b1.update(grad[1],-self.alpha)
                self.k2.update(grad[2],-self.alpha)
                self.b2.update(grad[3],-self.alpha)

                if nn.as_scalar(self.get_loss(nn.Constant(dataset.x),nn.Constant(dataset.y)))<0.01:
                    return
```

Listing 2 RegressionModel

在 *RegressionModel* 代码中，我们第一次引入了激活函数来构建神经网络，根据题目要求，我们采用 *ReLU* 函数来搭建神经网络，采用两个隐藏层进行网络实现与更新，在网络训练最后通过损失函数的值是否小于 0.01 来判断是否结束训练。

### 3.3 *DigitClassificationModel* 实现

```
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 ***"
        #测试数据
        self.batch_size = 100
        #设计两层网络中的参数
        self.k1 = nn.Parameter(784,200)
        self.b1 = nn.Parameter(1,200)
        self.k2 = nn.Parameter(200,100)
        self.b2 = nn.Parameter(1,100)
        self.k3 = nn.Parameter(100,10)
        self.b3 = nn.Parameter(1,10)
        #学习率
        self.alpha = 0.1

    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 ***"
        f1 = nn.AddBias(nn.Linear(x,self.k1),self.b1)
        new_f1 = nn.ReLU(f1)
        f2 = nn.AddBias(nn.Linear(new_f1,self.k2),self.b2)
        new_f2 = nn.ReLU(f2)
        f3 = nn.AddBias(nn.Linear(new_f2,self.k3),self.b3)
```

```
        return f3

    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 ***"
        while True:
            for x,y in dataset.iterate_once(self.batch_size):
                loss = self.get_loss(x,y)
                grad = nn.gradients(loss,[self.k1,self.b1,self.k2,self.b2,self.k3,self.b3])

                #更新
                self.k1.update(grad[0],-self.alpha)
                self.b1.update(grad[1],-self.alpha)
                self.k2.update(grad[2],-self.alpha)
                self.b2.update(grad[3],-self.alpha)
                self.k3.update(grad[4],-self.alpha)
                self.b3.update(grad[5],-self.alpha)

                if dataset.get_validation_accuracy() >0.973:
                    return
```

Listing 3 DigitClassificationModel

*DigitClassificationModel* 代码大体架构和 *RegressionModel* 一致，不同之处仅仅在于我们在本子任务中多了一个隐藏层进行网络训练，同时增加了隐藏层的维数，与此对应的，为了加快网络的训练速度，我们也对应的修改了学习率以更好的适应这个模型。

### 3.4 *LanguageIDModel* 实现

```python
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 ***"
        self.batch_size = 200
        #神经网络设计中，我们对于f_initial函数设计两层网络，实现方式基本和task2,task3一致
        #对于RNN中的hidden层，我们同样设计两层
        self.k = nn.Parameter(self.num_chars,200)
        self.k_x = nn.Parameter(200,200)
        self.k_f = nn.Parameter(200,5)

        self.alpha = 0.1

    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).

        Here `xs` will be a list of length L. Each element of `xs` will be a
        node with shape (batch_size x self.num_chars), where every row in the
        array is a one-hot vector encoding of a character. For example, if we
        have a batch of 8 three-letter words where the last word is "cat", then
        xs[1] will be a node that contains a 1 at position (7, 0). Here the
        index 7 reflects the fact that "cat" is the last word in the batch, and
        the index 0 reflects the fact that the letter "a" is the inital (0th)
        letter of our combined alphabet for this task.

        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 i in range(len(xs)):
            if i == 0:
                #对第一个字符做特殊处理
                h = nn.ReLU(nn.Linear(xs[i],self.k))
            else :
                h = nn.ReLU(nn.Add(nn.Linear(xs[i],self.k),nn.Linear(h,self.k_x)))
        return nn.Linear(h,self.k_f)

    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 ***"
        while True:
            for x,y in dataset.iterate_once(self.batch_size):
                loss = self.get_loss(x,y)
                grad = nn.gradients(loss,[self.k,self.k_x,self.k_f])

                self.k.update(grad[0],-self.alpha)
                self.k_x.update(grad[1],-self.alpha)
                self.k_f.update(grad[2],-self.alpha)

```

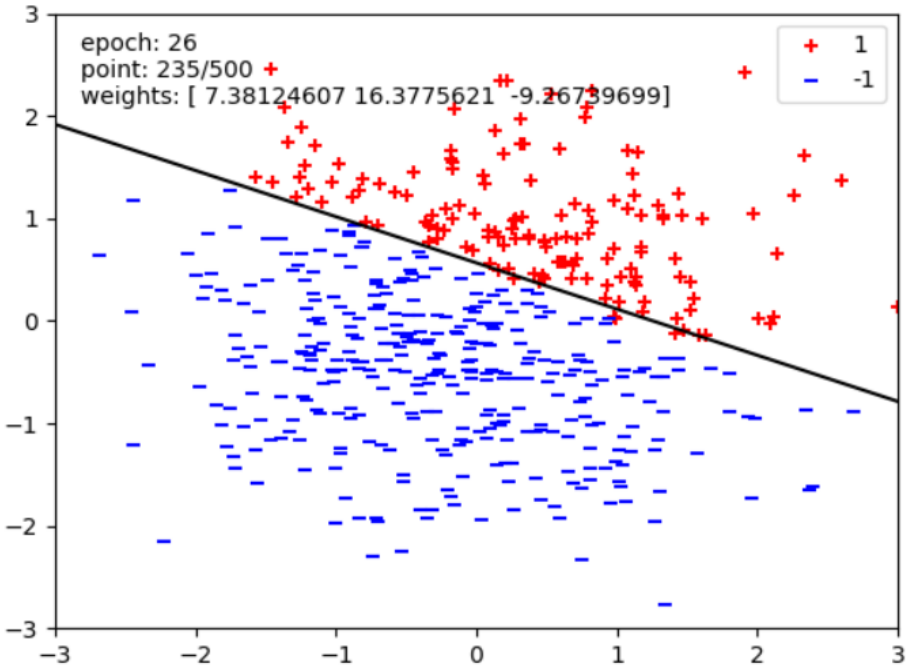
```
97                 if dataset.get_validation_accuracy() >0.85:
98                     return
```

Listing 4 LanguageIDModel

*LanguageIDModel* 中实现的重点在于 *run* 函数的编写，在该函数中，不同于前三问中我们进行顺序的网络参数更新，这次我们采用 *RNN* 算法进行运算更新，对第一个读入的字符进行线性回归和激活后，之后的每个字符读入的结果不仅和读入的字符有关，同时还和前一个输出的结果有关，相当于继承了上一次训练的结果，最后输出的是整个字符串识别的结果而不是单个字符识别的结果。

# 4 实验结果

4.1 *Question*1(6*points*) : *Perceptron*



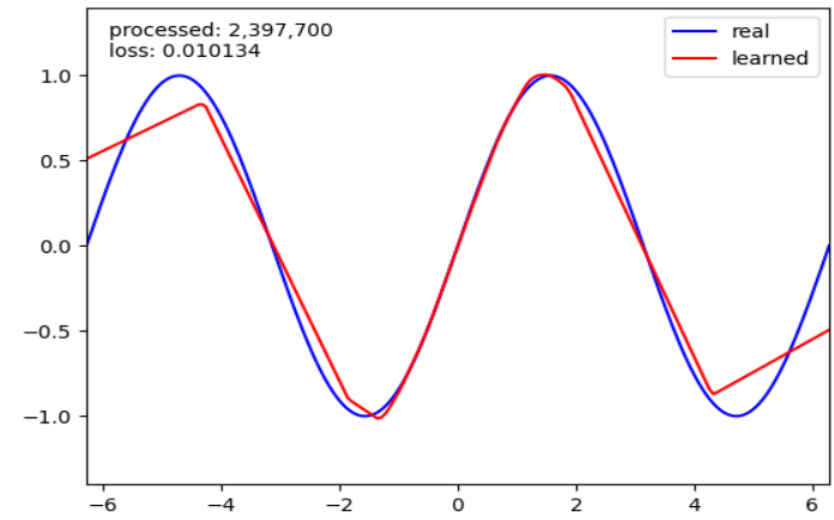
```
Question q1
===========
*** q1) check_perceptron
Sanity checking perceptron...
Sanity checking perceptron weight updates...
Sanity checking complete. Now training perceptron
*** PASS: check_perceptron

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

Finished at 16:59:58

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

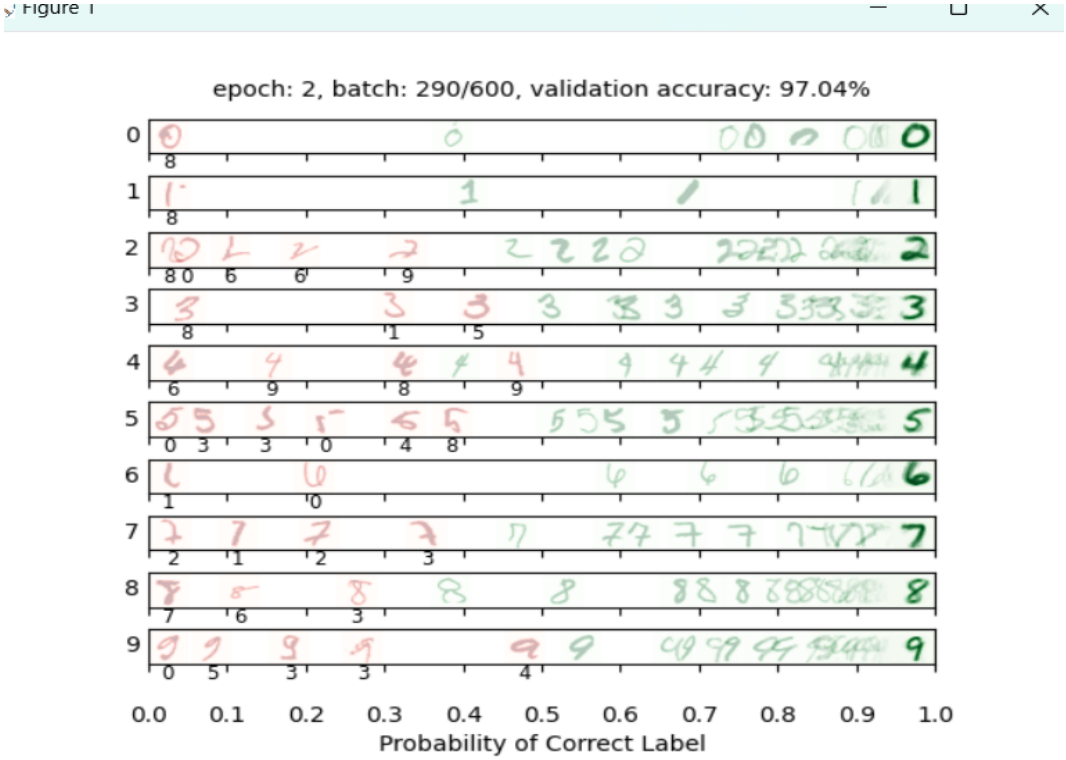
4.2 *Question*2(6*points*) : *Non* − *linearRegression*



```
Question q2
===========
*** q2) check_regression
Your final loss is: 0.009998
*** PASS: check_regression

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

4.3 *Question*3(6*points*) : *DigitClassification*



```
Question q3
===========
*** q3) check_digit_classification
Your final test set accuracy is: 97.480000%
*** PASS: check_digit_classification

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

4.4 *Question*4(7*points*) : *LanguageIdentification*

```
Your final test set accuracy is: 84.400000%
*** PASS: check_lang_id

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

4.5 **最终结果**

```
Provisional grades
==================
Question q1: 6/6
Question q2: 6/6
Question q3: 6/6
Question q4: 7/7
------------------
Total: 25/25
```

# 5 总结与分析

## 5.1 完成项目中遇到的问题

(1) 问题一：在进行 *task*2，通过非线性回归模拟 *sin*(*x*) 时，总是得不到正确的结果，自己拟合的曲线不会随着时间的推移而逐渐贴近 *sin*(*x*) 的曲线形状，在进行代码的认真检查后，发现了两个漏

洞，一是在网络运行时，我们只需要在第一层输出时采用激活函数处理，而之前我错误的把最后需要拟合的结果也经过了激活函数然后通过函数传出，因为激活函数会使得结果呈现分布式的结果，所以自然无法完成拟合连续的 $sin(x)$ 的效果；二是在进行模型的 *update* 时，使用的梯度和权重函数未照应。

(2) 问题二：在进行 *task*3 时，基本神经网络框架构建没有出现问题，和 *task*2 相差不多，但是在运行代码时，网络在测试集上的正确率始终达不到 0.97, 在反复检查代码以及阅读项目要求后，发现是自己神经网络的层数设置的太少以及学习率设置的太低，导致在有限的时间和训练集上没有训练出想要的结果。

### 5.2 本项目的收获与思考

本项目是人工智能中机器学习的一次简单应用，通过该项目，我首先熟悉并了解了简单神经网络搭建所需要的步骤和过程；在该项目中，我先后完成了二分类器、非线性函数拟合、手写数字识别以及语言字符识别等几个小项目，通过这几个有趣且实用的小项目，我对神经网络的基础知识有了更深的了解，也为之后继续深入了解神经网络以及机器学习的相关内容打好基础。