同济大学

实验报告

实验名称：Pacman吃豆人 对抗搜索

学号：2153538

姓名：刘博洋

完成日期：2023.4.23

1. 项目分析与概述

1.1.项目总体描述

本项目是机器学习与神经网络的简单介绍与实践，通过编写神经网络模型解决回归和分类问题，并在数据集上予以实践。

1.2.项目已有代码结构及类分析

需要编辑的文件	
models.py	神经网络的模型编写
需要阅读的文件	
nn.py	包含许多构建神经网络与处理数据的方法供调用

项目已有代码包括一个供编写的model.py,包括多个构建神经网络的类以及方法库nn.py，提供数据处理的方法以及遍历数据集的函数。

1.3.子问题概述及简要分析

1.3.1. Q1：感知器（Perceptron）

感知器是一种单层前馈神经网络，所有输入直接连接到输出，本题用感知器制作线性分类器，将所有点分为两类。

1.3.2. Q2：非线性回归（Non-linear Regression）

神经网络可以运用于回归模型，将已知函数作为训练模型，运用梯度下降，不断计算损失函数并更新权重， 可以不断拟合近似于目标函数

1.3.3. Q3：手写数字识别（Digit Classification）

本题是经典的神经网络问题，通过编写多层神经网络以及使用MINIST的手写数字训练集，将图片转化为张量并用训练集进行训练，可以对成千上万张图片进行识别。

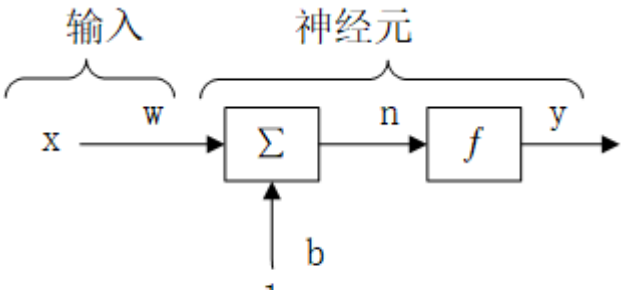
1.3.4. Q4: 语言分类（Language Identification）

在得到一个单词或者一个句子后我们常常需要判断其是由什么语言书写的，但是计算机没有人脑这样的判断力，于是便需要用神经网络模型首先接收带有不同标签的语言和单词进行训练，但是由于不同的单词含有不同的长度，所以我们需要用到循环神经网络进行训练（RNN）,即每一层都与上一层训练的结果有关，这样的话计算机才能更加整体地了解到整个单词的意思，从而作出分类，而不是限于单个字母。

2. 算法设计与实现

2.1.1. Q1：感知器（Perceptron）

感知器也就是单层神经网络，或者是神经元，是组成神经网络中最小的单位。



感知器结构图

感知器的输出为：y=f(n)=f(wx+b)，其中，w和b为感知器模型参数，w表示权值，b表示偏置，wx表示w和x的内积。在感知层进行学习时，每一个样本都将作为一个刺激输入神经元。输入信号是每一个样本的特征，期望的输出是该样本的类别。当输出与类别不同时，可以通过调整突触权值和偏置值，直到每个样本的输出与类别相同。本题中要将二维平面上的点（x,y）分为两类-1,1.

```
    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 ***"
        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 ***"

    return 1 if nn.as_scalar(self.run(x)) >= 0 else -1


def train(self, dataset):
    """
    Train the perceptron until convergence.
    """
    "*** YOUR CODE HERE ***"
    while True:
        # 记录权重是否被更新过
        updated = False
        for x, y in dataset.iterate_once(1):
            pre = self.get_prediction(x)# 计算预测值
            if pre != nn.as_scalar(y): # 如果预测值和实际类别不同则更新权重
                # nn.Parameter.update(x,nn.as_scalar(y))
                self.w.update(x, nn.as_scalar(y))
                updated = True
        if not updated:
            break
```

在感知器中，run（x）表示按照当前权值和偏移得到的点的分数，然后通过get_prediction函数计算预测类别是-1还是1，这样在训练模型的过程中，如果预测值和实际的类别不同，就需要反向更新权值，调用update（）函数进行更新，通过训练得到感知器模型，对于新的输入向量便可比较准确的预测其类别。

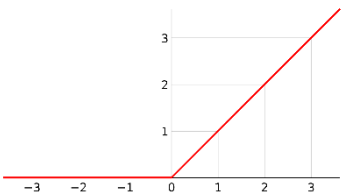
2.1.2. Q2：非线性回归（Non-linear Regression）

非线性回归也是神经网络的一个重要应用。由于感知器中输入层关于权值的计算总是线性的，要想拟合非线性函数，必须要在双层或者多层的神经网络中，必须加入激活函数，加入非线性的因素。

本题中的激活函数已有代码以及帮我们编写好了，为ReLU函数，ReLU，全称为：**Rectified Linear Unit**，是一种人工神经网络中常用的激活函数，通常意义下，其指代数学中的斜坡函数，即

$$f(x) = \max(x, 0)$$

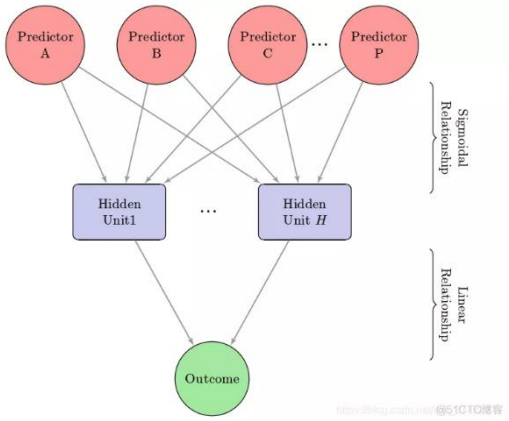
对应的函数大致如下图所示



而在神经网络中，ReLU函数作为神经元的激活函数，为神经元在线性变换WTx+b 之后的非线性输出结果。换言之，对于进入神经元的来自上一层神经网络的输入向量 x xx，使用ReLU函数的神经元会输出

$$\max(0, W^T X + b)$$

到下一个神经元或作为输出。



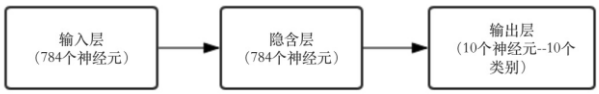

```
    class RegressionModel(object):
    """
    A neural network model for approximating a function that maps from real
    numbers to real numbers. The network should be sufficiently large to be
able
    to approximate sin(x) on the interval [-2pi, 2pi] to reasonable
precision.
    """
    def __init__(self):
        # Initialize your model parameters here
        "*** YOUR CODE HERE ***"
        # 设置权重参数和偏移修正值,一共两层，一个隐含层，学习率 0.05
        self.batch_size=200;
        self.w0=nn.Parameter(1,512)
        self.b0=nn.Parameter(1,512)
        self.w1=nn.Parameter(512,1)
        self.b1=nn.Parameter(1,1)
        self.learning_rate=0.05


    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 ***"
        xw1=nn.Linear(x,self.w0)
        r=nn.ReLU(nn.AddBias(xw1,self.b0))
        xw2=nn.Linear(r,self.w1)
        return nn.AddBias(xw2,self.b1)

    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 ***"
        # 计算真实值与预测值的损失
        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.w0,self.b0,self.w1,self.b1])
                self.w0.update(grad[0],-self.learning_rate)
                self.b0.update(grad[1], -self.learning_rate)
                self.w1.update(grad[2], -self.learning_rate)
                self.b1.update(grad[3], -self.learning_rate)
            if
nn.as_scalar(self.get_loss(nn.Constant(dataset.x),nn.Constant(dataset.y)))<=
0.005:
                return
```

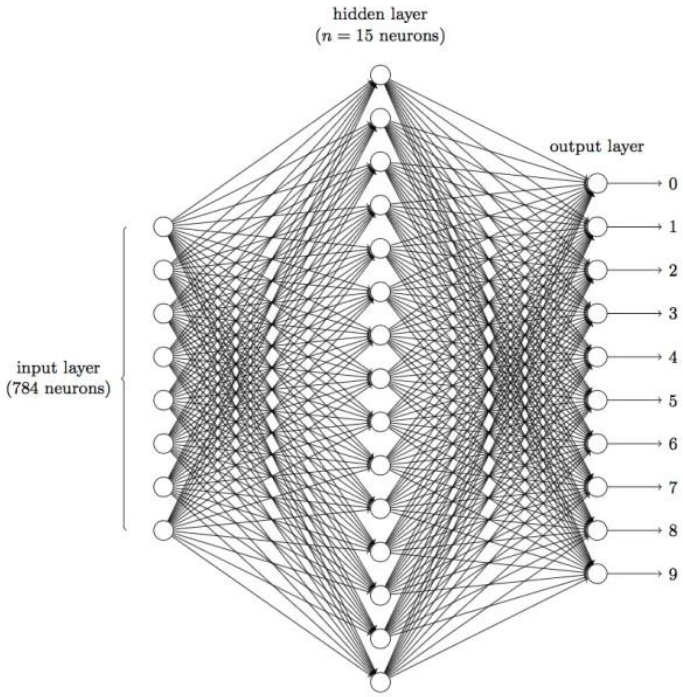
首先要设置两个神经网络层的权值和偏移量参数，以及按照题目要求设置学习率，然后run函数中，通过前向传播，调用ReLU函数激活神经元，并且加上偏置值，实现对输入值的预测。随后在getloss函数中，由于这里是回归问题，所以选用Squareloss，即平方损失函数，来衡量模型预测的准确度，在train函数中，通过每一个点的损失函数，反向更新两层神经网络的权值和偏置参数。最后设置训练停止的条件，即损失值小于0.005时停止训练。

2.1.3. Q3：手写数字识别（Digit Classification）

手写数字识别问题是最经典的神经网络应用问题，不同于回归问题，这是一个处理图像数据并分类的问题，我们知道，图像是由一个个像素点构成，每个像素点有三个通道，分别代表RGB颜色，那么，如果一个图像的尺寸是（28，28，1），即代表这个图像的是一个长宽均为28，channel为1的图像（channel也叫depth,此处1代表灰色图像）。如果使用全连接的网络结构，即，网络中的神经与与相邻层上的每个神经元均连接，那就意味着我们的网络有28 * 28 =784个神经元，然后通过训练集，对每一个神经元经过线性变换和激活函数得到最终的得分，而得分高低标志着图像属于某一类别的概率高低，这样就可以比较好的识别出手写数字的特征并成功归类。



一种可能的神经网络结构如下，在隐含层里包含了15个神经元采用全连接模式进行训练和预测。




```
    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 ***"
        # 设置模型参数
        self.batch_size=100
        self.w0=nn.Parameter(784,200)
        self.b0=nn.Parameter(1,200)
        self.w1=nn.Parameter(200,10)
        self.b1=nn.Parameter(1,10)
        self.learning_rate=0.5

    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 ***"
        xw1=nn.Linear(x,self.w0)
        r=nn.ReLU(nn.AddBias(xw1,self.b0))
        xw2=nn.Linear(r,self.w1)
        return nn.AddBias(xw2,self.b1)

    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 ***"
        # 由于是分类问题，所以采用 SoftmaxLoss 损失函数
        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.w0,self.b0,self.w1,self.b1])
                self.w0.update(grad[0],-self.learning_rate)
                self.b0.update(grad[1],-self.learning_rate)
                self.w1.update(grad[2],-self.learning_rate)
                self.b1.update(grad[3],-self.learning_rate)
            if dataset.get_validation_accuracy()>0.98:
                return
```

同样采用全连接的传统神经网络，这个数字识别问题的解决步骤与回归问题基本类似，关键就在于输入层和隐含层的计算和映射，同时，有一个细节是项目要求我们在最后一层映射时不要加入 ReLU 激活函数，这是因为最后一层是输出层，输出层的作用是将输入映射到输出空间，不需要进行非线性变换。另外，加入 ReLU 激活函数可能会破坏输出的概率解释性，不利于分类问题的解释和可视化。因此，在最后一层不加入 ReLU 激活函数是合理的。

2.1.4. Q4：期望最优（ExpectiMax）

Ghost每次都进行最优操作是可遇不可求的，我们必须也要能在对手不作出最佳操作时进行最合理的操作，扩大自己的优势，这里可以把ghost的行动看成随机的，

```
    class ExpectimaxAgent(MultiAgentSearchAgent):
    """
      Your expectimax agent (question 4)
    """
    def getAction(self, gameState):
        """
        Returns the expectimax action using self.depth and
self.evaluationFunction

        All ghosts should be modeled as choosing uniformly at random from
their
        legal moves.
        """
        "*** YOUR CODE HERE ***"
        GhostIndex = [i for i in range(1, gameState.getNumAgents())]

        # 目标状态：游戏结束或者搜索到一定深度
        def target(state, d):
            return state.isWin() or state.isLose() or d == self.depth

        # ghost 不一定每次作出最好决定，要计算期望值
        def exp_value(state, d, ghost):

            if target(state, d):
                return self.evaluationFunction(state)

            value = 0
            prob = 1 / len(state.getLegalActions(ghost))# 每种情况的概率

            for action in state.getLegalActions(ghost):
                if ghost == GhostIndex[-1]:# 递归的查找期望值，如果最后一个 ghost 已经
作出行动了，下一次便是轮到 pacman
                    value += prob * max_value(state.generateSuccessor(ghost,
action), d + 1)
                else:# 否则就遍历所有 ghost，每一个 ghost 作出行动后再由 pacman 行动
                    value += prob * exp_value(state.generateSuccessor(ghost,
action), d, ghost + 1)

            return v

        def max_value(state, d):

            if target(state, d):
                return self.evaluationFunction(state)

            MAX = -float("inf")
            for action in state.getLegalActions(0):
                MAX = max(MAX, exp_value(state.generateSuccessor(0, action), d,
1))
            return MAX
```

```
        ans = [(action, exp_value(gameState.generateSuccessor(0, action), 0,
1)) for action in
               gameState.getLegalActions(0)]
        ans.sort(key=lambda k: k[1])

        return ans[-1][0]
```

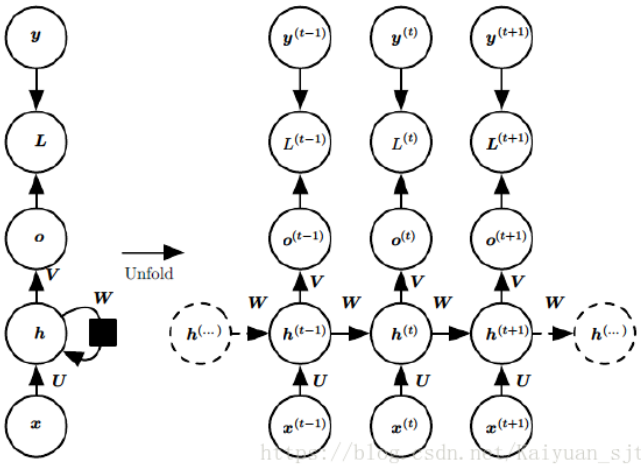
设置ghost每个行动的概率都相等，然后每一次搜索后都把所得效用值加入value,然后pacman再根据ghost生成的效用值进行行动。

2.1.5 Q4: 语言分类（Language Identification）

对于分类问题，不论是用传统神经网络还是卷积神经网络都可以实现，但是在语言处理、分词等领域，则不得不用到循环神经网络（RNN）

在传统神经网络和卷积神经网络中训练样本的输入和输出是比较的确定的。传统的神经网络模型中，是从输入层到隐含层再到输出层，层与层之间是全连接的，每层之间的节点是无连接的。但是有一类问题它们不好解决，就是训练样本输入是连续的序列,且序列的长短不一，比如基于时间的序列：一段段连续的语音，一段段连续的手写文字。这些序列比较长，且长度不一，比较难直接的拆分成一个个独立的样本来进行训练。

而对于这类问题，RNN则比较的擅长。那么RNN是怎么做到的呢？RNN假设我们的样本是基于序列的。比如是从序列索引1到序列索引 τ 的。对于这其中的任意序列索引号t,它对应的输入是对应的样本序列中的x(t)。而模型在序列索引号t位置的隐藏状态h(t)，则由x(t)和在t-1位置的隐藏状态h(t-1)共同决定。在任意序列索引号t，我们也有对应的模型预测输出o(t)。通过预测输出o(t)和训练序列真实输出y(t),以及损失函数L(t)，我们就可以用DNN类似的方法来训练模型，接着用来预测测试序列中的一些位置的输出。



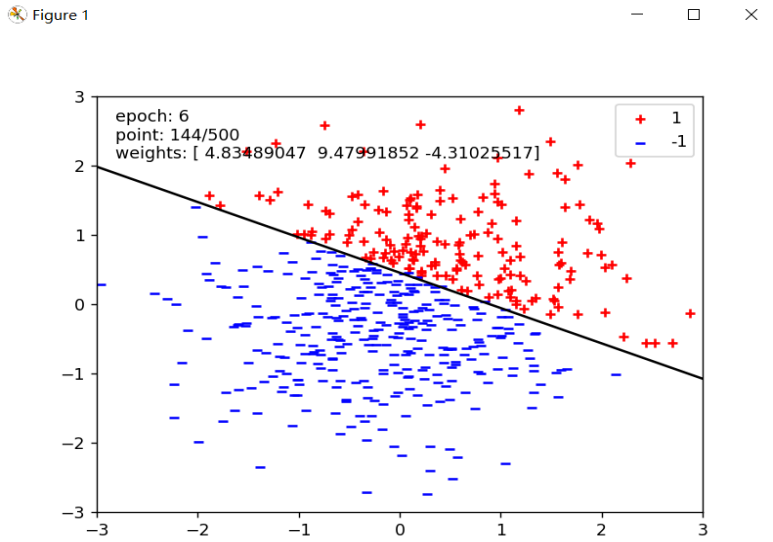
从网络结构上，RNN会记忆之前的信息，并利用之前的信息影响后面结点的输出。在实现语言分类时，由于许多语言公用相同的字母，所以如果只对单个的字母进行训练是肯定不够的，还要考虑其前后字母之间的关系，这就涉及到构词法、常见词尾、词头等等，这就是RNN可以做到的地方，即记录前面已经读取的单词或语句，进行训练之后便可以对一段陌生的文字提取特征来计算其属于某个语言类别的概率最大。

```
    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=50
        self.learning_rate=0.05
        self.w0=nn.Parameter(self.num_chars,300)
        self.b0 = nn.Parameter(1, 300)  # b0
        self.w1 = nn.Parameter(300, 300)  # w1
        self.b1 = nn.Parameter(1, 300)  # b1
        self.wf = nn.Parameter(300, 5)  # wf
        self.bf = nn.Parameter(1, 5)  # bf

    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 ***"
        initial=nn.ReLU(nn.AddBias(nn.Linear(xs[0],self.w0),self.b0))
        z=initial
        for i,x in enumerate(xs[1:]):
            z=nn.Add(nn.ReLU(nn.AddBias(nn.Linear(z, self.w1),
self.b1)),initial)
            z=nn.Add(nn.ReLU(nn.Linear(z, self.w1)), nn.ReLU(nn.Linear(x,
self.w0)))
        return nn.AddBias(nn.Linear(z, self.wf), self.bf)


    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.w0, self.b0, self.w1, self.b1,
self.wf, self.bf])
                self.w0.update(grad[0],-self.learning_rate)
                self.b0.update(grad[1],-self.learning_rate)
                self.w1.update(grad[2], -self.learning_rate)
                self.b1.update(grad[3], -self.learning_rate)
                self.wf.update(grad[4], -self.learning_rate)
                self.bf.update(grad[5], -self.learning_rate)
            if dataset.get_validation_accuracy()>=0.85:
                return
```

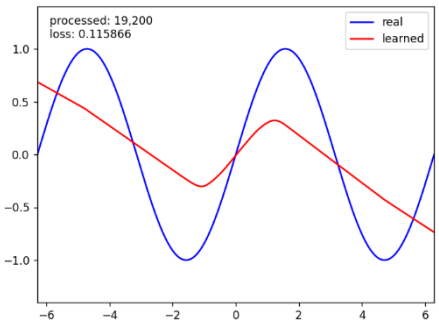
3. 实验结果

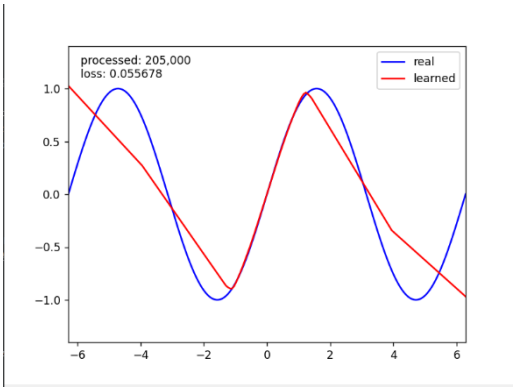
Q1：

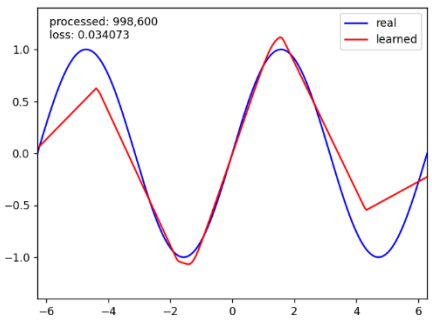


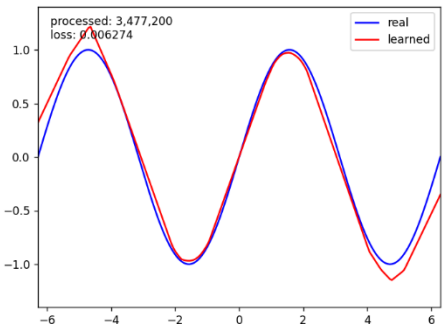
可以看到构建的感知器较好的将二维点集分为两类。

Q2：



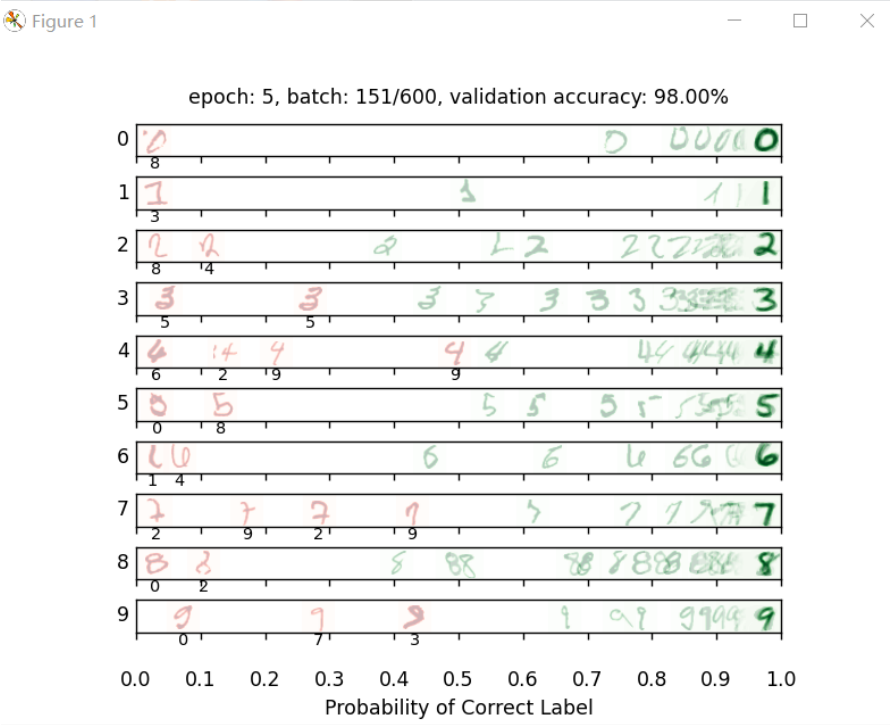






可以看到拟合效果是比较好的，从一开始的接近一条直线到最后与目标曲线基本一致，拟合速度也较快，说明神经网络的设计是比较成功的，而且用来解决回归和拟合问题也有较好的效果。

Q3：



随着训练数据量的加大，多层神经网络可以比较准确的读取图片中数字的特征。

Q4：

```
epoch 13 iteration 309 validation-accuracy 83.4%
  forth      English ( 87.2%)               |en 87%|es  4%|fi  0%|nl  6%|pl  3%
  follows    English ( 98.4%)               |en 98%|es  0%|fi  0%|nl  0%|pl  1%
  supplies   English ( 72.8%)               |en 73%|es  8%|fi  0%|nl 19%|pl  0%
  gatito     Spanish ( 92.6%)               |en  5%|es 93%|fi  0%|nl  0%|pl  2%
  imágenes   Spanish ( 45.3%) Pred: English |en 52%|es 45%|fi  0%|nl  2%|pl  0%
  ejecutivo  Spanish ( 97.1%)               |en  3%|es 97%|fi  0%|nl  0%|pl  0%
  käveli     Finnish (100.0%)               |en  0%|es  0%|fi100%|nl  0%|pl  0%
  elämme     Finnish ( 99.4%)               |en  0%|es  0%|fi 99%|nl  0%|pl  0%
  kutsuvat   Finnish (100.0%)               |en  0%|es  0%|fi100%|nl  0%|pl  0%
  attentie   Dutch   ( 29.0%) Pred: Spanish |en  7%|es 56%|fi  7%|nl 29%|pl  1%
  legt       Dutch   ( 68.8%)               |en 28%|es  0%|fi  0%|nl 69%|pl  3%
  ritme      Dutch   ( 36.0%)               |en 19%|es 23%|fi 17%|nl 36%|pl  5%
  momencie   Polish  ( 15.7%) Pred: Spanish |en  5%|es 72%|fi  0%|nl  8%|pl 16%
  wariat     Polish  (  2.2%) Pred: English |en 93%|es  1%|fi  1%|nl  3%|pl  2%
  spraw      Polish  ( 30.5%) Pred: English |en 54%|es  9%|fi  0%|nl  7%|pl 30%
epoch 13 iteration 331 validation-accuracy 84.8%
  forth      English ( 67.7%)               |en 68%|es  6%|fi  0%|nl 14%|pl 11%
  follows    English ( 95.3%)               |en 95%|es  0%|fi  0%|nl  1%|pl  3%
  supplies   English ( 49.6%)               |en 50%|es 44%|fi  1%|nl  5%|pl  1%
  gatito     Spanish ( 94.4%)               |en  4%|es 94%|fi  0%|nl  0%|pl  2%
  imágenes   Spanish ( 58.5%)               |en 40%|es 59%|fi  0%|nl  1%|pl  0%
  ejecutivo  Spanish ( 94.3%)               |en  5%|es 94%|fi  0%|nl  0%|pl  0%
```

可以看到训练了13个epoch之后准确率已经达到了84.8%，这说明循环神经网络确实比较适合用来做文本处理这种带有时序逻辑的内容。

4. 总结与分析

4.1 算法分析

神经网络是机器学习中的一种模型，是一种模仿动物神经网络行为特征，进行分布式并行信息处理的算法数学模型。这种网络依靠系统的复杂程度，通过调整内部大量节点之间相互连接的关系，从而达到处理信息的目的。本项目用的较多的是 BP 神经网络，后向传播学习的前馈型神经网络（Back Propagation Feed-forward Neural Network，BPFNN/BPNN），是一种应用最为广泛的神经网络。在 BPNN 中，后向传播是一种学习算法，体现为 BPNN 的训练过程，该过程是需要教师指导的；前馈型网络是一种结构，体现为 BPNN 的网络构架。

反向传播算法通过迭代处理的方式，不断地调整连接神经元的网络权重，使得最终输出结果和预期结果的误差最小。

BPNN 是一种典型的神经网络，广泛应用于各种分类系统，它也包括了训练和使用两个阶段。由于训练阶段是 BPNN 能够投入使用的基础和前提，而使用阶段本身是一个非常简单的过程，也就是给出输入，BPNN 会根据已经训练好的参数进行运算，得到输出结果。

BP 神经网络的算法流程图如下：

BPNN 的训练过程具体如下所示，相应的流程图和伪代码如图 2.11 所示。

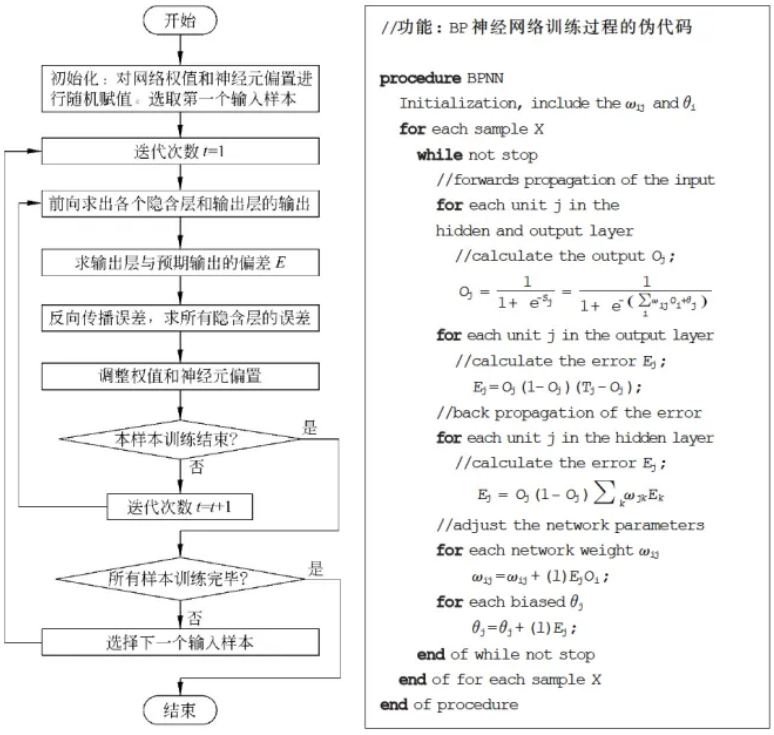


图 2.11　BP 神经网络算法训练阶段的流程图和伪代码

简单来说，对于一个 BP 神经网络，我们首先是初始化网络权重，即参数的值。通常取值范围是-1.0~1.0。然后向前传播，计算每一层的输出（k 层的输出即为 k+1 层的输入）。随后进行反向误差传播，就是通过与预期输出的比较得到每个输出单元的误差，且得到的误差需要从后往前传播，前面一层的误

差可以通过和它连接的后面的一层所有单元的误差计算所得，从后往前依次得到每一层每一个神经元的误差。最后进行网络权重与神经元偏置的调整。以此构建的模型可以得到较好的预测值。

而有一个比较重要的指标就是**学习率**：即要利用多少个误差点（即真实值与预测值不同的数据），学习率越低，损失函数的变化速度就越慢，容易过拟合。虽然使用低学习率可以确保我们不会错过任何局部极小值，但也意味着我们将花费更长的时间来进行收敛，特别是在被困在局部最优点的时候。而学习率过高容易发生梯度爆炸，loss 振动幅度较大，模型难以收敛。

4.2 项目总结

遇到的困难及解决：

在完成本项目过程中遇到的主要困难是开始项目初期对神经网络结构不够熟悉，只是比较简单的知道其原理，但是具体到代码每一步还是需要不断琢磨。而且对于项目的各个文件函数的简介也需要有充分的了解。

项目总结与收获：

通过这次项目，通过循序渐进的4个问题，我不但增长了我对神经网络问题的理解，而且利用神经网络解决了数字识别和文本分类问题，给了我一定的成就感，是一次非常有意义的实践。