实验报告

（2023~2024 学年第二学期）

课程名称 人工智能课程设计

实验名称 课程设计作业 Project5

姓名 范潇 学号 2254298

1. 项目概述

1.1. 主要内容与目标

本次项目的主要内容为神经网络的搭建与应用。
首先，项目要求我们实现感知机，然后搭建神经网络以完成非线性回归、数字分类以及语言识别。

1.2. 已有代码

下面对已有代码中与本项目的具体实现相关的代码进行分析。

1.2.1. nn.py

该文件提供了用于搭建神经网络的一系列节点：

1. Consant：由浮点数组成的二维数组；
2. Parameter：可训练的感知机或神经网络，提供了用于更新参数的方法 update；
3. DotProduct：对它的输入进行点积运算；
4. Add：逐元素的矩阵加法；
5. AddBias：为特征向量添加一个偏置值；
6. Linear：对输入进行线性变换；
7. ReLU：ReLU 激活函数；
8. SquareLoss：平方损失函数；
9. SoftmaxLoss：SoftmaxLoss 损失函数。

同时，还提供了一下两个函数：

1. gradients：给定损失函数对于给定参数的梯度；
2. as_scalar：将给定节点以 Python 中的数值类型返回。

1.2.2. backend.py

该文件提供了本项目中所需要使用的数据集，提供了以下方法：

1. iterate_once：返回数据集中的一批数据，返回的类型为一个二元组 (x,y)，其中 x,y 分别为特征和标签，类型均为 Constant 结点，大小分别为“批的大小 $\times$ 特征数量”，“批的大小 $\times$ 输出大小”；
2. iterate_forever：一个不断从数据集中返回数据的迭代器；
3. get_validation_accuracy：返回在验证集上得到的准确率。

2. Question1

2.1. 问题概述

本问题要求实现二元感知器。

2.2. 算法设计

这里假设所给的数据是可以被线性函数所完全分隔分隔开来的。对于二分类问题，感知器通过自身权重向量于数据的特征向量之间的点积的符号来进行分类。对于正例，点积符号为正，感知器输出 1；对于负例，点积符号为负，感知器输出-1。为了使得感知器能够将数据的正负例完全分离开来，需要进行学习，即根据分类结果不断对感知器中的权重进行调整。

如果感知器给出的分类结果与真实类别一致，则无需进行调整。否则，说明点积

$$\mathbf{W} \cdot \mathbf{x}$$

的符号错误，其中 $\mathbf{W}$ 为感知器的权重向量，$\mathbf{x}$ 为样例的特征向量。为此，如果感知器返回的点积为正，需要调整权重向量使其返回的点积减小；如果感知器返回的点积为负，需要调整权重向量使其返回的点积增大。由于 $\mathbf{x} \cdot \mathbf{x} \geq 0$，所以可以将 $\mathbf{W}$ 调整为

$$\mathbf{W}' = \mathbf{W} - \mathbf{sgn}(\mathbf{W} \cdot \mathbf{x})\mathbf{x},$$

从而有

$$\mathbf{W}' \cdot \mathbf{x} = \mathbf{W} \cdot \mathbf{x} - \mathbf{sgn}(\mathbf{W} \cdot \mathbf{x})\mathbf{x} \cdot \mathbf{x},$$

达到了上述的需求。

只需不断调整，直到感知器能够完全正确分类所有样例即可。

2.3. 算法实现

```
def run(self, x):
    return nn.DotProduct(x,self.w)

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

def train(self, dataset):
    while True:
        OK = True
        for x,y in dataset.iterate_once(1):
            res = self.get_prediction(x)
            if nn.as_scalar(y) != res:
```

```
                self.w.update(x,-res)
                OK = False
                break
        if OK:
            break
```

2.4. 实验结果

我成功获得了本问题的所有分数。本问所用的测试样例便是一个二元分类问题，要求感知器能够正确分类所有样例。

```
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 ###
```

图 2.1: Question1 实验结果

3. Question2

3.1. 问题概述

本问题要求利用神经网络拟合 $[-2\pi, 2\pi]$ 范围内的正弦函数。

3.2. 算法设计

理论上，二层神经网络便能以任意精度拟合任何给定的连续函数。对于本问题，只需要拟合 $[-2\pi, 2\pi]$ 范围内的正弦函数，因此二层神经网络就足够了。最终我使用的神经网络架构为：

$$\text{输入} \rightarrow \text{线性层} \rightarrow \text{非线性层} \rightarrow \text{线性层} \rightarrow \text{输出}$$

其中，线性层中包含偏置值，非线性层为使用 ReLU 函数。

在训练的过程中，我依据模型在当前批次样例上的损失函数值来判断是否应当结束训练。

3.3. 算法实现

```
class RegressionModel(object):
    def __init__(self):
        self.layer_size = 512
        self.feature_num = 1
        self.output_num = 1
        self.learning_rate = 0.05
        self.batch_size = 200
        self.threshold = 0.01
        self.w1 = nn.Parameter(self.feature_num,self.layer_size)
        self.b1 = nn.Parameter(1,self.layer_size)
        self.w2 = nn.Parameter(self.layer_size,self.output_num)
        self.b2 = nn.Parameter(1,self.output_num)
        self.W = [self.w1,self.b1,self.w2,self.b2]

    def run(self, x):
        return nn.AddBias( nn.Linear( nn.ReLU( nn.AddBias( nn.Linear(x,self.
            w1),self.b1)),self.w2),self.b2)

    def get_loss(self, x, y):
        return nn.SquareLoss(self.run(x),y)

    def train(self, dataset):
```

```
        for x,y in dataset.iterate_forever(self.batch_size):
            gradients = nn.gradients(self.get_loss(x,y),self.W)
            for i,w in enumerate(self.W):
                w.update(gradients[i],-self.learning_rate)
            loss = self.get_loss(x,y)
            if nn.as_scalar(loss) <= self.threshold:
                break
```

3.4. 实验结果

我成功获得了本问题的所有分数。本问的评分依据便是训练好的神经网络在测试集上得到的损失函数值。

```
Question q2
===========
*** q2) check_regression
Your final loss is: 0.009986
*** PASS: check_regression

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

图 3.1: Question2 实验结果

4. Question3

4.1. 问题概述

本问题要求利用神经网络进行数字分类。

4.2. 算法设计

所用的神经网络架构与上一题一样，唯一的区别在于特征向量维数从 1 变为了 28×28，输出向量维数由 1 变为了 10，同时判断训练是否结束的依据变为了由在验证集上得到的准确率。

4.3. 算法实现

```
class DigitClassificationModel(object):
    def __init__(self):
        self.layer_size = 512
        self.feature_num = 28*28
        self.output_num = 10
        self.learning_rate = 0.5
        self.batch_size = 200
        self.threshold = 0.98
        self.w1 = nn.Parameter(self.feature_num,self.layer_size)
        self.b1 = nn.Parameter(1,self.layer_size)
        self.w2 = nn.Parameter(self.layer_size,self.output_num)
        self.b2 = nn.Parameter(1,self.output_num)
        self.W = [self.w1,self.b1,self.w2,self.b2]

    def run(self, x):
        return nn.AddBias( nn.Linear( nn.ReLU( nn.AddBias( nn.Linear(x,self.
            w1),self.b1)),self.w2),self.b2)

    def get_loss(self, x, y):
        return nn.SoftmaxLoss(self.run(x),y)

    def train(self, dataset):
        for x,y in dataset.iterate_forever(self.batch_size):
            gradients = nn.gradients(self.get_loss(x,y),self.W)
            for i,w in enumerate(self.W):
                w.update(gradients[i],-self.learning_rate)
```

```
26              if dataset.get_validation_accuracy() >= self.threshold:
27                  break
```

4.4. 实验结果

我成功获得了本问题的所有分数。本问的评分依据便是训练好的神经网络在测试集上得到的准确率。

```
Question q3
===========
*** q3) check_digit_classification
Your final test set accuracy is: 97.560000%
*** PASS: check_digit_classification

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

图 4.1: Question3 实验结果

5. Question4

5.1. 问题概述

本问题要求利用神经网络实现语言识别功能。

5.2. 算法设计

项目要求使用 RNN 解决该问题。它的核心思想是，对于输入的单词，以字母为单位不断更新预测的结果，并且除了当前的字母外，还以上一次的得出的结果作为输入（除首字母外）。这要求我们构造两个函数 f_{init} 和 f_{mid} 来分别处理首字母和后续字母的情形。

由于不排除只由一个字母组成的单词的可能，所以 f_{init} 的输出大小也必须符合最终结果的要求，特别地，大小必须相同。

由于 f_{init} 的作用只占一小部分，所以可以直接沿用之前的神经网络结构，即两层线性层加上一层非线性层。

对于 f_{mid}，首先要处理的是将上一层得到的输出和当前的输入结合在一起。为此，我先对当前输入和上一层的输出分别使用一层线性层，以统一它们的形状，然后使用逐元素的矩阵加法将它们合并在一起，以方便后续神经网络的搭建。最终，我在此基础之上又添加了两层隐藏层。

具体神经网络架构如下图所示，其中 last 指模型在上一个输入中得到的结果，线性层中均包含了偏置值，非线性层均使用 ReLU 函数。

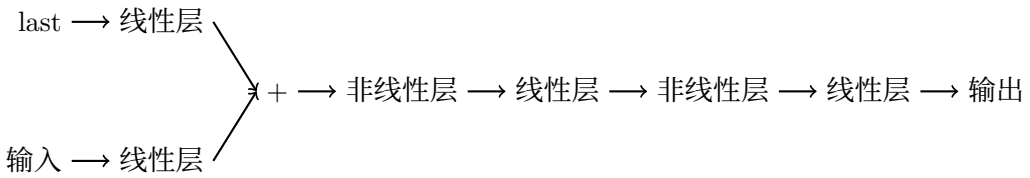

图 5.1: 神经网络架构

训练是否结束的依据仍然是在验证集上得到的准确率。

5.3. 算法实现

为了避免学习率设置不当而导致训练后期出现梯度消失或梯度爆炸的问题，我让学习率以已训练批次的对数的倒数的速率递减。

```
class LanguageIDModel(object):
    def __init__(self):
        self.num_chars = 47
        self.languages = ["English", "Spanish", "Finnish", "Dutch", "Polish"
            ]
```

```
        self.layer_size = 1<<9
        self.second_layer_size = 1<<9
        self.feature_num = self.num_chars
        self.output_num = len(self.languages)
        self.learning_rate = 0.1
        self.batch_size = 200
        self.threshold = 0.87
        self.cnt = 1

        self.w1 = nn.Parameter(self.feature_num,self.layer_size)
        self.b1 = nn.Parameter(1,self.layer_size)
        self.w2 = nn.Parameter(self.layer_size,self.second_layer_size)
        self.b2 = nn.Parameter(1,self.second_layer_size)
        self.w3 = nn.Parameter(self.second_layer_size,self.output_num)
        self.b3 = nn.Parameter(1,self.output_num)
        self.w4 = nn.Parameter(self.layer_size,self.output_num)
        self.b4 = nn.Parameter(1,self.output_num)
        self.w_hidden = nn.Parameter(self.output_num,self.layer_size)
        self.W = [self.w1,self.b1,self.w2,self.b2,self.w_hidden,self.w3,self
            .b3,self.w4,self.b4]

    def init(self,x0):
        return nn.AddBias( nn.Linear( nn.ReLU( nn.AddBias( nn.Linear(x0,self
            .w1),self.b1)),self.w4),self.b4)

    def mid(self,h,x):
        return \
            nn.AddBias( nn.Linear(
        nn.ReLU(
                    nn.AddBias( nn.Linear(
                        nn.ReLU(
                                    nn.Add( nn.AddBias( nn.Linear(x,self.w1)
                                        ,self.b1),nn.Linear(h,self.w_hidden))
                                ),
                    self.w2),self.b2)
                ),
            self.w3),self.b3)

    def run(self, xs):
        last = None
        for i,x in enumerate(xs):
            if not i:
```

```
                last = self.init(x)
            else:
                last = self.mid(last,x)
        return last

    def get_loss(self, xs, y):
        return nn.SoftmaxLoss(self.run(xs),y)

    def train(self, dataset):
        for x,y in dataset.iterate_forever(self.batch_size):
            self.cnt += 1
            gradients = nn.gradients(self.get_loss(x,y),self.W)
            for i,w in enumerate(self.W):
                w.update(gradients[i],-self.learning_rate*(1/log(self.cnt)))
            if dataset.get_validation_accuracy() >= self.threshold:
                break
```

5.4. 实验结果

我成功获得了本问题的所有分数。本文的评分标准是模型在测试集上的准确率达到 81% 及以上。

```
Your final test set accuracy is: 84.000000%
*** PASS: check_lang_id


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

图 5.2: Question4 实验结果

6. 总结与分析

通过本次实验，我深刻体会到了机器学习以及神经网络的强大之处。

在实验过程中，我主要遇到了三个问题。

其一，在完成问题二的过程中，起初，我并没有在线性层中增加偏置值。从可视化的结果中可以看出，由模型得到的曲线只是两段从原点引出的射线。而当我添加了偏置值后，由模型得到的曲线很快便接近正弦函数的形态。

其二，在完成神经网络的相关题目时，我发现如果编程过程中把有关问题的参数以硬编码的方式保留下来，将会大大降低代码的可移植性和可读性，同时，还不便于推导所需矩阵的形状。为此，我重构了自己的代码，将参数在 ___init___ 函数中便初始化，而非以魔数的形式在后续的训练过程中使用。这使得我在完成第三题时，可以直接使用第二题的代码，只需修改输入输出的相关参数即可。

其三，在完成第四题的过程中，我体会到了超参数的重要性。一开始，我沿用前两题的两层线性层的神经网络架构，但是尝试了如 0.5，0.1，0.01 等多个学习率后，结果都是在达到所需的准确率之前便产生了梯度消失或梯度爆炸的现象。为此，我将神经网络架构调整为了总共有三层线性层的架构。在相同的其他超参数的条件下，新得到的神经网络便能成功达到所需的准确率。

在训练神经网络的过程中，我也观察到神经网络的准确率不一定时随着训练时间递增的，会有一定的波动存在，为此，在最后一题中，我设定停止训练的准确率阈值为 87%，以增加在测试集上达到 81% 准确率的概率。当然这样做的代价便是训练时间会增长一倍左右。我最终使用的神经网络能够在训练了 60 个 epoch 后达到 80% 左右的准确率，训练了 100 个 epoch 后能达到 85% 左右的准确率，而为了达到 87% 的准确率，需要训练 180 个 epoch 左右，这需要 20 分钟左右的时间。