机器学习实验 2023春

实验四 神经网络

实验内容：

使用pytorch构件神经网络,完成简单回归,分类问题.

环境要求：

Python，numpy支持多维度的数组和矩阵运算，pandas数据处理和分析工具，Matplotlib图形化工具，pytorch深度学习库.

```
#为避免环境冲突,使用anaconda管理python环境
#由于使用ARM架构的设备,为了增加计算效率,考虑安装ARM原生的python和pytorch

CONDA_SUBDIR=osx-arm64 conda create -n Pytorch python=3.10
conda activate Pytorch
pip install torch torchvision torchaudio
```

模型推导:

梯度下降是一种基于反向传播算法来更新神经网络权重的方法。具体而言，根据损失函数对权重的偏导数计算出梯度，然后利用梯度的反方向来更新权重，使得损失函数的值随着训练逐步减小，从而提高模型的准确性.下面结合两个问题做简要说明:

任务一:

通过梯度下降法来拟合线性回归问题 $y = ax + b$

定义损失函数: $J(a,b) = \frac{1}{m}\sum_{i=1}^{m}(y_i - (ax_i + b))^2$, 通过求解损失函数关于参数 a 和 b 的偏导数，得到梯度 $\nabla_{a,b}J(a,b)$.

使用梯度下降法不断迭代更新模型参数 a 和 b，直到损失函数收敛，即取到最小值.

任务二:

通过梯度下降法拟合Logistic回归问题:

输入数据 $x=(x_1,x_2,\cdots,x_n)^T$，对应的标签为 y

Logistic回归的目标是寻找一组参数 $w=(w_1,w_2,\cdots,w_n)^T$ 和 b，使得对于所有的输入数据 x，都有:

$\hat{y}=\sigma(w^Tx+b)=\frac{1}{1+e^{-(w^Tx+b)}}$ 其中 σ 表示Sigmoid函数: $\sigma(x)=\frac{1}{1+e^{-x}}$

Logistic回归的参数学习过程就是最小化损失函数 $J(w,b)$:

$J(w,b)=-\frac{1}{m}\sum_{i=1}^{m}(y^{(i)}\log\hat{y}^{(i)}+(1-y^{(i)})\log(1-\hat{y}^{(i)}))$

通过梯度下降等优化算法来更新参数 w 和 b，从而最小化损失函数 $J(w,b)$.

任务一：回归模型

本任务中你将搭建神经网络模型完成回归任务
假设你是一家特许餐厅的首席执行官，正在考虑在不同的城市开设一家新的分店.该连锁店已经在不同的城市有分店，你有这些城市的利润和人口数据,希望使用这些数据来帮助选择下一个要扩展到的城市.文件ex1data.csv包含我们的线性回归问题的数据集,Population代表一个城市的人口，profit代表此个城市的餐厅利润.利润的负值表示亏损.
请使用pytorch搭建一个三层神经网络，构建回归模型，选择合适的损失函数，使用反向传播来优化模型参数，记录模型训练过程损失.请留出20%作为测试集评估模型，并分析结果.

任务分析:

这是一个典型的分类问题,通过构建神经网络,完成回归分析,具体流程如下:

1.加载CSV格式数据集，将其分成训练集和测试集
2.定义三层全连接神经网络模型
3.定义损失函数和优化器，并使用训练集对模型进行训练
4.使用测试集评估了模型的性能

实验代码:

```
# -*- coding: utf-8 -*-
# @Time : 5/2/23 11:18
# @Author : ANG

import torch
import torch.nn as nn
import pandas as pd
from sklearn.model_selection import train_test_split

# 加载数据集
data = pd.read_csv('/Users/wallanceleon/Desktop/机器学习/机器学习实
验/20201060287-李昂-实验四/数据集/ex1data.csv')
X = data.iloc[:, :-1].values
y = data.iloc[:, -1].values

# 将数据集分成训练集和测试集
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=0)

# 将数据转换为张量
X_train = torch.from_numpy(X_train).float()
y_train = torch.from_numpy(y_train).float()
X_test = torch.from_numpy(X_test).float()
y_test = torch.from_numpy(y_test).float()

# 定义三层神经网络
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(1, 10)
        self.fc2 = nn.Linear(10, 5)
        self.fc3 = nn.Linear(5, 1)

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        x = self.fc3(x)
        return x

# 定义模型、损失函数和优化器
net = Net()
criterion = nn.MSELoss()
```

```
optimizer = torch.optim.SGD(net.parameters(), lr=0.01)

# 训练模型
for epoch in range(1000):
    optimizer.zero_grad()
    outputs = net(X_train)
    loss = criterion(outputs, y_train)
    loss.backward()
    optimizer.step()
    if (epoch+1) % 100 == 0:
        print('Epoch [{}/{}], Loss: {:.4f}'.format(epoch+1, 1000,
loss.item()))

# 在测试集上评估模型
with torch.no_grad():
    predicted = net(X_test)
    test_loss = criterion(predicted, y_test)
    print('Test Loss: {:.4f}'.format(test_loss.item()))
```

实验结果:

进行1000次迭代,每100次迭代后输出均方误差 $mse = (1/n) * \Sigma(y - y_hat)^2$ 和测试误差,通过调整学习率,观察模型拟合效果:

Lr = 0.001	Lr = 0.01
Epoch [100/1000], Loss: 38.9518	Epoch [100/1000], Loss: 33.0021
Epoch [200/1000], Loss: 38.0776	Epoch [200/1000], Loss: 32.1567
Epoch [300/1000], Loss: 37.2534	Epoch [300/1000], Loss: 32.0541
Epoch [400/1000], Loss: 36.3925	Epoch [400/1000], Loss: 32.0419
Epoch [500/1000], Loss: 35.4144	Epoch [500/1000], Loss: 32.0404
Epoch [600/1000], Loss: 34.3107	Epoch [600/1000], Loss: 32.0402
Epoch [700/1000], Loss: 33.2499	Epoch [700/1000], Loss: 32.0401
Epoch [800/1000], Loss: 32.5112	Epoch [800/1000], Loss: 32.0401
Epoch [900/1000], Loss: 32.1737	Epoch [900/1000], Loss: 32.0401
Epoch [1000/1000], Loss: 32.0701	Epoch [1000/1000], Loss: 32.0401
Test Loss: 18.6968	Test Loss: 18.7845
学习率较小时,迭代1000次仍未达到收敛	学习率较大时,迭代400步完成收敛,最终误差为:18.7845

任务二：分类模型

本任务中你将搭建神经网络模型完成回归任务
假设你是一所大学系的管理员，你想根据两次考试的成绩来决定每个申请人的录取机会.你有以前申请者的历史数据，可以用作逻辑回归的训练集.对于每个培训示例，你都有申请人在两次考试中的分数和录取决定. 你的任务是建立一个分类模型，根据这两次考试的分数来估计申请人的录取概率.文件ex1data2.csv包含我们的逻辑回归问题的数据集,学生的两门成绩Exam1，Exam2和是否被录取Accepted(1为录取，0为未录取)
请使用pytorch搭建一个四层神经网络，构建分类模型，选择合适的损失函数，使用反向传播来优化模型参数，记录模型训练过程损失.留出20%作为测试集评估模型，并分析结果.

任务分析:

这是一个典型的Logistic回归问题,通过构建神经网络,完成分析,具体流程如下:

```
1.加载CSV格式数据集，将其分成训练集和测试集
2.定义三层全连接神经网络模型
3.定义损失函数和优化器，并使用训练集对模型进行训练
4.使用测试集评估了模型的性能
```

实验代码:

```
# -*- coding: utf-8 -*-
# @Time : 5/2/23 12:37
# @Author : ANG

import pandas as pd
import torch
import torch.nn as nn
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split

# 读取数据集
data = pd.read_csv('/Users/wallanceleon/Desktop/机器学习/机器学习实
验/20201060287-李昂-实验四/数据集/ex1data2.csv')

# 将数据集分为特征和标签
X = data.iloc[:, :-1].values
y = data.iloc[:, -1].values
```

```
# 将数据集分为训练集和测试集
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=0)


class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(2, 16)
        self.fc2 = nn.Linear(16, 8)
        self.fc3 = nn.Linear(8, 1)
        self.sigmoid = nn.Sigmoid()  # Sigmoid激活函数

    def forward(self, x):
        x = self.fc1(x)
        x = self.sigmoid(x)
        x = self.fc2(x)
        x = self.sigmoid(x)
        x = self.fc3(x)
        x = self.sigmoid(x)
        return x


net = Net()

criterion = nn.BCELoss()  # 二元交叉熵损失函数
optimizer = torch.optim.SGD(net.parameters(), lr=0.1)  # 随机梯度下降优
化器

for epoch in range(1000):
    inputs = torch.Tensor(X_train).float()
    labels = torch.Tensor(y_train).float()

    optimizer.zero_grad()  # 梯度清零

    outputs = net(inputs)
    loss = criterion(outputs, labels.unsqueeze(1))
    loss.backward()  # 反向传播
    optimizer.step()  # 更新权重

    if epoch % 100 == 0:
        print('Epoch [{}/{}], Loss: {:.4f}'.format(epoch + 1, 1000,
loss.item()))
```

```
# 在测试集上预测
with torch.no_grad():
    inputs = torch.Tensor(X_test).float()
    labels = torch.Tensor(y_test).float()

    outputs = net(inputs)
    predicted = torch.round(outputs)

# 计算准确率
accuracy = accuracy_score(labels, predicted)

print('Accuracy:', accuracy)
```

实验结果:

进行1000次迭代,每100次迭代后输出均方误差,并调用sk-learn库,评估模型准确率:

准确率
Epoch [1/1000], Loss: 0.6992 Epoch [101/1000], Loss: 0.6709 Epoch [201/1000], Loss: 0.6697 Epoch [301/1000], Loss: 0.6676 Epoch [401/1000], Loss: 0.6661 Epoch [501/1000], Loss: 0.6640 Epoch [601/1000], Loss: 0.6610 Epoch [701/1000], Loss: 0.6580 Epoch [801/1000], Loss: 0.6545 Epoch [901/1000], Loss: 0.6506 Accuracy: 0.6