BP 神经网络在模式识别中的应用

摘要：

建立一个四层的BP神经网络，用于成功辨识手写数字（0-9）。制定网络的结构，包括层数、每一层的神经元数量，以及单个神经元的输入输出函数；根据数字辨识的任务，规划网络的输入和输出；实现BP网络的误差反传算法，完成神经网络的培训和检测，确保最终辨识准确率超过90%。

1. 导言

在现代生活中，手写字体识别在许多场景中变得至关重要。为了赋予系统识别手写数字的能力，我们采用了BP神经网络，并通过手写数字的数据进行训练。BP算法是一种多层前馈网络，其学习规则利用误差的梯度下降法，通过误差反向传播不断调整网络的权重和阈值，以最小化网络的误差。

BP神经网络的特性：

1. **非线性映射能力：**
   - BP神经网络具备学习和存储大量的输入-输出模式映射关系，无需预先了解描述这种映射关系的数学方程。只要提供足够多的样本模式供网络学习训练，它就能完成从n维输入空间到m维输出空间的非线性映射。
2. **泛化能力：**
   - 网络在面对未曾见过的非样本数据时，仍能正确映射输入空间到输出空间，展现出优秀的泛化能力。
3. **容错能力：**
   - 即使输入样本中存在较大的误差或个别错误，BP神经网络对网络的输入输出规律的影响也较小，表现出较强的容错能力。

通过这些特性，BP神经网络在手写数字识别任务中展现出卓越的性能，使其成为解决手写字体识别问题的强大工具。

2. 实验过程

BP 算法思想流程：

1. 初始化
2. 输入训练样本，计算各层输出；
3. 计算网络输出误差；
4. 计算各层误差信号；
5. 调整各层权值；
6. 对所有样本完成一次轮训。

设计 BP 神经网络，实现手写数字识别

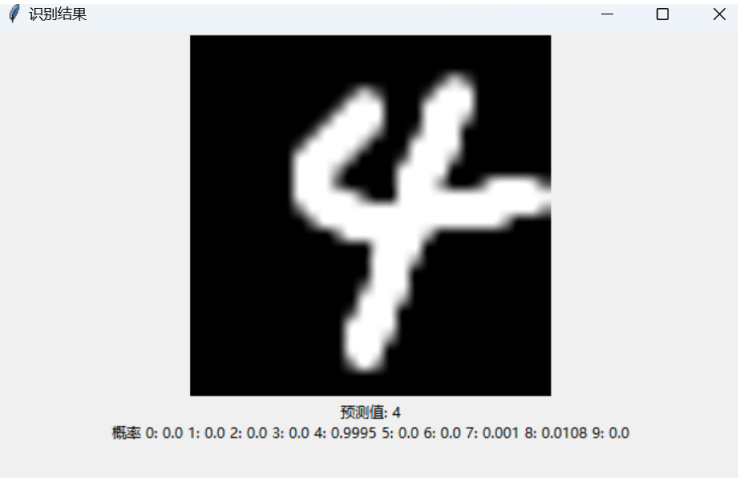
1. 使用 mnist 手写数字数据库提供的训练数据集和测试数据集，训练数据集有 60000 个样本，测试数据集有 10000 个样本，每个样本是一个 784 维的向量， 是一张手写 0-9 数字的图片按从上到下，从左到右进行向量化的结果。
2. 采用四层 BP 神经网络——输入层，隐层和输出层。输入层的神经元数根据 使用的数字识别训练集或测试集中输入向量的维数决定，为 784；隐层的神经元数可调整以提高识别率，经过多次测试取 196，49；输出层的神经元数取 10， 分别代表数字 0-9；学习率为 0.525。

3. 网络的输入：一个 784 维的输入向量。网络的输出：输出层有 10 个神经元， 分别代表数字 0-9，每个神经元的输出值表示识别出的数字为该神经元代表 的数字的可能性，值越大则可能性越高，最终识别出的数字为 10 个神经元 中输出值最大的神经元所代表的数字。

3. 结果分析

调整隐层神经元数和学习率，使用相同的训练和测试数据集，得出的识别率会发生变化，经过多次调整，最终取神经元数为 196，49，学习率为 0.525. 结果：经过多次测试运行，识别率均在 97%~99%之间，满足要求。 一次运行：

```
训练完成, 用时1403.72017秒
开始测试模型
Testing: 100%|██████████| 10000/10000 [00:05<00:00, 1746.67iteration/s]
训练数据总数: 60000
神经网络结构: 196 49 10
测试数据总数: 10000
正确结果数: 9792.0
识别率: 97.92%
期望输出中每个数字的数量: 980 1135 1032 1010 982 892 958 1028 974 1009
每个数字正确结果的数量: 973 1120 1009 990 958 878 932 1010 949 973
每个数字的识别率: 99.29% 98.68% 97.77% 98.02% 97.56% 98.43% 97.29% 98.25% 97.43% 96.43%
```



分析与改进：观察结果显示，不同数字的辨识率存在差异，这与数字的辨识难度以及训练样本数量有一定关联。为了提高辨识准确性，可能需要增加更多的训练样本，或者探索更为优化的权值调整方法。

代码：

定义net类

```python
import cupy as cp
import pandas as pd

# cp.fuse()装饰器可以将多个函数融合为一个函数，从而减少函数调用的开销，相较于numpy提高了25%左右的性能，
# 相较于未优化的cupy提高了近50%的性能(由于GPU性能波动，使得其性能最低跌至CPU的50%，上限仅仅为CPU的
# 90%，这里假定50%提高，实际上可能会由于波动有上下浮涨)

@cp.fuse()
# 激活函数
def get_act(x):
```

```
    return 1 / (1 + cp.exp(-x))
@cp.fuse()
# 激活函数的导数
def get_act_derivative(x):
    return x * (1 - x)

class DNN():
    def __init__(self, sample_config):
        # 加载数据
        self.sample_config = sample_config
        self.train_images = cp.array(pd.read_csv(self.sample_config["train_file"]).values.tolist()) / 256.0
        self.train_targets = pd.read_csv(self.sample_config["train_labels_file"]).values
        self.test_images = cp.array(pd.read_csv(self.sample_config["test_file"]).values.tolist()) / 256.0
        self.test_targets = pd.read_csv(self.sample_config["test_labels_file"]).values
        self.pkl_file = self.sample_config["savemodels_file"]
        self.dist = self.initialize_parameters()

    # 激活函数
    def get_act(self, x):
        return 1 / (1 + cp.exp(-x))

    # 激活函数的导数
    def get_act_derivative(self, x):
        return x * (1 - x)

    # 不调用原因，调用对象本身函数无法被@cp.fuse()优化，开销较大

    # 初始化权重和偏置
    def initialize_parameters(self):
        # 配置神经网络参数
        samples_num, input_num = self.train_images.shape
        output_num = self.sample_config["output_num"]
        hidden_nodes = list(map(int, self.sample_config["hidden_nodes"].split(', ')))
        hidden_num = len(hidden_nodes)
        learn_rate = self.sample_config["learn_rate"]
        # 初始化权值和偏置
        weights = [i for i in range(hidden_num)]
        weights[0] = 0.2 * cp.random.random((input_num, hidden_nodes[0])) - 0.1

        for i in range(1, hidden_num):
            weights[i] = 0.2 * cp.random.random((hidden_nodes[i-1], hidden_nodes[i])) - 0.1
        offsets = [cp.zeros(hidden_nodes[i]) for i in range(hidden_num)]

        dist = {
            "samples_num": samples_num,
            "input_num": input_num,
```

```
            "output_num": output_num,
            "hidden_num": hidden_num,
            "hidden_nodes": hidden_nodes,
            "weights": weights,
            "offsets": offsets,
            "learn_rate": learn_rate,
        }

        return dist

    # 前向传播
    def forward_propagation(self, image):
        hidden_values = [0] * self.dist["hidden_num"]
        hidden_acts = [0] * self.dist["hidden_num"]
        hidden_values[0] = cp.dot(image, self.dist["weights"][0]) +
self.dist["offsets"][0]
        hidden_acts[0] = get_act(hidden_values[0])
        for i in range(1, self.dist["hidden_num"]):
            hidden_values[i] = cp.dot(hidden_acts[i-1], self.dist["weights"]
[i]) + self.dist["offsets"][i]
            hidden_acts[i] = get_act(hidden_values[i])

        return hidden_acts

    # 反向传播
    def backward_propagation(self, image, hidden_acts, target):
        # 误差反传
        e = target - hidden_acts[-1]
        deltas = [0] * self.dist["hidden_num"]
        deltas[-1] = e * get_act_derivative(hidden_acts[-1])
        for i in range(self.dist["hidden_num"] - 2, -1, -1):
            deltas[i] = get_act_derivative(hidden_acts[i]) *
cp.dot(self.dist["weights"][i+1], deltas[i+1])

        # 调整权值和偏置
        self.dist["weights"][-1] += self.dist["learn_rate"] *
cp.outer(hidden_acts[-2], deltas[-1])
        for i in range(self.dist["hidden_num"] - 2, -1, -1):
            self.dist["weights"][i] += self.dist["learn_rate"] *
cp.outer(hidden_acts[i-1] if (i - 1) != -1 else image, deltas[i])

        for i in range(self.dist["hidden_num"]):
            self.dist["offsets"][i] += self.dist["learn_rate"] * deltas[i]
```

定义TrainDNN类

```
import cupy as cp
import time
from tqdm import tqdm
import json
import pickle
from DNN.DNN_BP.net import DNN
from PIL import Image, ImageTk

import tkinter as tk
```

```python
class TrainDNN(DNN):
    def __init__(self, sample_config):
        # 初始化
        super(TrainDNN, self).__init__(sample_config)
        self.savemodels_file = self.sample_config["savemodels_file"]

    def train(self):
        start_time = time.time()
        train_count = int(len(self.train_images))
        epochs = self.sample_config["epochs"]

        for i in range(epochs):
            for count in tqdm(range(train_count), desc="The {0}th epoch".format(i+1), unit="iteration", bar_format="{l_bar}{bar}{r_bar}", colour='blue'):

                image = self.train_images[count]
                target = cp.zeros(self.dist["output_num"]) + 0.001
                target[self.train_targets[count]] = 0.999

                self.backward_propagation(image, self.forward_propagation(image), target)

        end_time = time.time()
        print("训练完成，用时{0}秒".format(round(end_time - start_time, 5)))

        self.save(self.dist)

    def test(self):

        with open(self.savemodels_file, 'rb') as pkl_file:
            self.dist = pickle.load(pkl_file)

        print("开始测试模型")

        right_count = cp.zeros(10)
        expect_count = cp.zeros(10)
        test_count = len(self.test_images)

        for i in range(test_count):
            expect_count[self.test_targets[i]] += 1

        for count in tqdm(range(test_count), desc="Testing", unit="iteration", bar_format="{l_bar}{bar}{r_bar}", colour='green'):
            image = self.test_images[count]
            target = self.test_targets[count]

            hidden_acts = self.forward_propagation(image)
            output_act = hidden_acts[-1]

            if cp.argmax(output_act) == cp.array(target):
                right_count[target] += 1

        text = []

```

```
        text.append("训练数据总数: {0} ".format(self.dist["samples_num"]))
        text.append("神经网络结构: {0}".format(' '.join([str(i) for i in self.dist["hidden_nodes"]])))

        right_sum = right_count.sum()
        text.append("测试数据总数: {0}".format(test_count))
        text.append("正确结果数: {0}".format(right_sum))

        rate = right_sum / test_count
        text.append("识别率: {0}%".format(rate * 100))
        text.append("期望输出中每个数字的数量: {0}".format(' '.join([str(int(i)) for i in expect_count.get()])))
        text.append("每个数字正确结果的数量: {0}".format(' '.join([str(int(i)) for i in right_count.get()])))

        rate_arr = right_count / expect_count
        text.append("每个数字的识别率: {0}".format(' '.join(["{0}%".format(round(i * 100, 2)) for i in rate_arr.get()])))

        text = '\n'.join(text)

        return text

    def save(self, dist):
        with open(self.savemodels_file, 'wb') as pkl_file:
            pickle.dump(dist, pkl_file)

        return dist

    def run(self):

        with open(self.savemodels_file, 'rb') as pkl_file:
            self.dist = pickle.load(pkl_file)

        image = Image.open(self.sample_config["test_image_file"])
        image = image.resize((28, 28)).convert("L")

        # 进行预测
        hidden_acts = self.forward_propagation(cp.array(list(image.getdata())) / 256.0)
        output_act = hidden_acts[-1]

        output_probability = [str(round(i, 4)) for i in output_act.get()]
        output_probability = ' '.join([str(i) + ": " + output_probability[i] for i in range(len(output_probability))])

        # 用tkinter显示图片和结果
        root = tk.Tk()
        root.title("识别结果")
        root.geometry("625x375")

        img = ImageTk.PhotoImage(image.resize((300, 300)))
        label_img = tk.Label(root, image=img)
        label_img.pack()

```

```
        label_text = tk.Label(root, text= "预测值: " + str(cp.argmax(output_act)) + "\n" + "概率 " + output_probability)
        label_text.pack()

        root.mainloop()

def main():
    # 读取配置文件
    with open("../config/config.json", "r", encoding="utf-8") as f:
        sample_config = json.load(f)

    # cp.cuda.Device(0).use()

    dnn = TrainDNN(sample_config)
    dnn.train()
    print(dnn.test())
    dnn.run()

if __name__ == "__main__":
    main()
```

json文件用于更改训练时的参数

```
{
  "train_file": "../../test&train_csv/mnist_train.csv",
  "train_labels_file": "../../test&train_csv/mnist_train_labels.csv",
  "test_file": "../../test&train_csv/mnist_test.csv",
  "test_labels_file": "../../test&train_csv/mnist_test_labels.csv",
  "models_file": "./model.pkl",
  "savemodels_file": "./model.pkl",
  "test_image_file": "../../input_image/test.png",
  "learn_rate": 0.575,
  "?learn_rate": "调参过程中比较优秀的值",
  "hidden_nodes": "196, 49, 10",
  "?hidden_nodes": "隐藏层节点数, 可以多层, 用逗号加空格分隔, 最后一层节点数必须等于output_num",
  "output_num": 10,
  "epochs": 30,
  "?epochs": "增加epochs基本上没有太大提升, 反而会导致过拟合"
}
```

关于环境配置

鉴于`cupy`配置需要一定动手能力, 可以以`numpy`代替, 可以将`import cupy as cp`改为`import numpy as cp`, 同时将之后所有`.get()`删去, 并将`@cp.fuse()`注释掉即可. 当改为`numpy`时注意文件所提供的模型文件不可用, 请自行训练.

--- 作者: Caiki