基于BP神经网络的手写数字识别

2019211315班 胡昕韵 2019211420

北京邮电大学

1 总体设计

1.1 数据集

采用标准MNIST数据集作为本次手写数字识别的目标数据集。该数据集分为训练集(train)、验证集(validation)、测试集(test)。训练集数据规模为50,000条，验证集与测试集规模均为10,000条。本模型在训练集上进行训练，根据其在验证集上的表现调整超参数（包括神经网络的层数、每层的节点数、学习率等等等），最后在测试集上进行测试与评估。

1.2 环境

本次编程作业采用Python语言完成，不采用已有Pytorch, Tensorflow等框架，手写神经网络，若需要搭建运行环境运行程序，请参考`README.md`文件

小组成员：胡昕韵 2019211420

1.3 神经网络

基于结构模拟的方法，模拟实现全连接的BP神经网络，网络的层数与每层的节点数量，每个节点的激活函数，权值的初始化方式均可自由调整。导数通过反向传播(back-propagation)算法进行传递。

训练时采用SGD(Stochastic Gradient Descent)方式，可选标准的SGD（每次抽取一个样本）与Mini-Batch SGD（每次抽取一个小批量的样本，可指定一批次样本的数量）

- 神经网络初始化选项
  - layer_sizes：每层的节点数量
  - init_method：初始化每层权重的方法（可选`'constant'`-常数，`'uniform'`-均匀分布，`'normal'`-正态分布），默认为均匀分布
  - init_bound：初始化每层权重的范围（若初始化方法为`constant`，则应输入一个常数，否则应为两个浮点数，分别对应范围的上下界），默认为[-0.5, 0.5]
  - activation_func：节点的激活函数（可选`'sigmoid'`，`'tanh'`），默认采用Sigmoid函数
- 训练参数
  - optimizer：优化方法，默认为`'minibatch'`，可选`'standard'`
  - epoch：训练的轮次，整数
  - batch_size：若使用Mini-Batch SGD，则该选项为一个批次抽取的数据数量，默认为32
- 其他选项：
  - quiet：是否不输出模型表现，默认为False（即输出表现）

2 算法

BP神经网络实现为`Model`类，初始化该类需要定义每层节点的数量，权重的初始化方法，以及激活函数。

神经网络每层连接均为线性连接，若记第k层d_k维的节点输入值为in_k输出值为out_k，则：

$$in_{k+1} = out_k \cdot W_k + b_k$$
$$out_{k+1} = f(in_{k+1})$$

其中W_k为两层间的权重矩阵，维度为$d_k * d_{k+1}$；b_k为偏置量，是d_{k+1}维的向量，f函数为节点的激活函数，可以是sigmoid或tanh函数

```
self.weights = [self.weight_init_method(in_size, out_size, init_bound)
                for in_size, out_size in zip(layer_sizes[:-1], layer_sizes[1:])]
self.biases = [self.weight_init_method(1, size, init_bound)
               for size in layer_sizes[1:]]
```

`weights`记录所有W，`biases`记录所有b

2.1 前向更新节点

```
activations = [x.reshape(1, -1)]
before_activation = [x.reshape(1, -1)]
layer = x
for weight, bias in zip(self.weights, self.biases):
    layer = np.dot(layer, weight)+bias
    before_activation.append(layer)
    layer = self.activation_func(layer)
    activations.append(layer)
```

`before_activation`记录每层节点的输入，`activations`记录每层节点的输出（激活函数后输出的值），`layer`记录当前层的输出；对于每一层，首先计算其线性变换后的值，再计算其在激活函数后的输出

2.2 Loss计算

输出层设计为10个单元，分别对应10个数字。首先将每个数字对应的标签数字转为one-hot向量（例如：1转换为[0,1,0,...,0]），计算输出在softmax后的向量的第i位作为预测该图片为数字i的可能性。

$$S = Softmax(x) = \frac{e^x}{\sum_i e^{x_i}}$$

损失函数(loss function/cost function)采取交叉熵(cross-entropy)函数计算。

$$Loss = CrossEntropy(x, y) = -\sum_i y_i \ln S_i$$

由于y仅有一个位置为1，其余均为0，故可以简化为：

$$Loss = -\ln S_i\ , \quad where\ y_i = 1$$

对Loss求导可得：

$$\frac{\partial S_i}{\partial x_j} = \begin{cases} S_i \cdot (1 - S_j), & i = j \\ -S_i \cdot S_j, & i \neq j \end{cases}$$

$$\frac{\partial L}{\partial S_i} = -y_i \cdot \frac{1}{S_i}$$

$$\frac{\partial L}{\partial x_j} = \frac{\partial L}{\partial S_i} \cdot \frac{\partial S_i}{\partial x_j} = S_j - y_j$$

2.3 后向更新导数

梯度下降法中要求求得损失函数的值L相对于参数的梯度。对于输出层o，

$$\frac{\partial L}{\partial out_o} = Softmax(out_o) - y$$
$$\frac{\partial L}{\partial in_o} = \frac{\partial L}{\partial out_o} \cdot \frac{\partial out_o}{\partial in_o} = [Softmax(out_o) - y] \cdot f'(in_o)$$

其中$f'(x)$为激活函数在输出为x处的导数

对于连接第i层与第i+1层的权重W_i，偏置b_i，记第i层节点的输出为out_i，若已计算出$\frac{\partial L}{\partial in_{i+1}}$，则：

$$\begin{aligned}
\frac{\partial L}{\partial b_i} &= \frac{\partial L}{\partial in_{i+1}} \\
\frac{\partial L}{\partial W_i} &= \frac{\partial L}{\partial in_{i+1}} \cdot \frac{\partial in_{i+1}}{\partial W_i} = out_i^T \cdot \frac{\partial L}{\partial in_{i+1}} \\
\frac{\partial L}{\partial in_i} &= \frac{\partial L}{\partial in_{i+1}} \cdot \frac{\partial in_{i+1}}{\partial out_i} \cdot \frac{\partial out_i}{\partial in_i} \\
&= \frac{\partial L}{\partial in_{i+1}} \cdot W_i^T * f'(in_i)
\end{aligned}$$

代码实现如下：

```
delta = (softmax(activations[-1])-label) *
                    self.activation_func_derivative(before_activation[-1])
deriv_b = [np.zeros(b.shape) for b in self.biases]
deriv_w = [np.zeros(w.shape) for w in self.weights]
deriv_b[-1] = delta.reshape(deriv_b[-1].shape)
deriv_w[-1] = np.dot(activations[-2].transpose(), deriv_b[-1])
for i in range(2, self.layer_num):
    delta = np.dot(delta, self.weights[-i+1].transpose())*
                self.activation_func_derivative(before_activation[-i])
    deriv_b[-i] = delta
    deriv_w[-i] = np.dot(activations[-i-1].transpose(), delta)
return deriv_w, deriv_b
```

其中`delta`对应$\frac{\partial L}{\partial in_{i+1}}$，`deriv_w`，`deriv_b`分别记录$\frac{\partial L}{\partial W}$与$\frac{\partial L}{\partial b}$

前向更新与反向传播同时进行，在函数`forward_backward`中实现，函数返回值为$\frac{\partial L}{\partial W}$与$\frac{\partial L}{\partial b}$

2.4 更新参数

在调用`forward_backward`函数求出$\nabla J(\Theta^i)$后，更新参数Θ为：

$$\Theta^{i+1} = \Theta^i - \alpha * \nabla J(\Theta^i)$$

其中，α为学习率。代码实现如下：

```
delta_w, delta_b = self.forward_backward(x, y)
self.weights = [w-step*dw for w, dw in zip(self.weights, delta_w)]
self.biases = [b-step*db for b, db in zip(self.biases, delta_b)]
```

若采用Mini-Batch SGD，则对于一批的数据累加`delta_w`，`delta_b`之后再一起进行更新

2.5 SGD

SGD的核心在于随机抽取数据来完成梯度下降并更新参数而非在整个数据集上进行计算损失函数，大大减少了每轮训练的计算量。本算法中实现了标准的SGD与Mini-Batch SGD。

- SGD伪代码:

```
for k in range(epoch):
    shuffle(train_data)
    for x, y in train_data:
        delta_w, delta_b = self.forward_backward(x,y)
        # 更新参数
        self.weights = [w-step*dw for w, dw in zip(self.weights, delta_w)]
        self.biases = [b-step*db for b, db in zip(self.biases, delta_b)]
```

- Mini-Batch SGD伪代码:

```
for k in range(epoch):
    mini_batches = 把训练集分割为大小为batch_size的块
    for batch in mini_batches:
        for x, y in batch:
            delta_w, delta_b += self.forward_backward(x,y)
        # 更新参数
        self.weights = [w-step*dw for w, dw in zip(self.weights, delta_w)]
        self.biases = [b-step*db for b, db in zip(self.biases, delta_b)]
```

3 模型特点

- 模型实现了标准的BP神经网络，模型灵活性高，利于在多次测试与超参数的调整，可调整包括隐藏层数量、节点个数、学习率、批次大小、激活函数、权值初始化方法等在内的多种模型参数
- 模型在训练集上学习，根据在验证集上的表现调整超参，全程未接触测试集数据，最终根据在测试集数据上的表现测得的分数可信度高。
- 提供SGD与Mini-Batch SGD两种训练方式，易于比较两者的区别。
- 模型封装完善，仅需提供必要的参数与数据集即可进行训练与调用

4 遇到的问题

1. 矩阵乘法在代码实现时常常忘记其实际矩阵大小，导致运行失败

   解决：通过仔细注明变量的矩阵大小维度来减少编程错误

2. 第一次运行时，发现准确率长期维持在10%左右的水平，完全没有学习到任何有效信息

   解决：通过debug发现，sigmoid函数编写出现失误，错将函数$f(x)=\frac{1}{1+e^{-x}}$编写成$f(x)=\frac{1}{1+e^{x}}$，少写一个负号，导致算法失败

3. 学习的过程过于缓慢，模型表现长时间得不到提高

   解决：学习率过低，适当提高设定的学习率；以及，在模型表现震荡强烈时应当适当调低学习率。

4. 模型的权重初始化的方式与数据范围会强烈影响其优化速度与表现，若设定的初始化值范围过大或取值范围不当可能在Sigmoid与Tanh作为激活函数时出现梯度消失问题，训练进展及其缓慢。

   解决：设定合理的初始值范围。最终选取的初始化方式为：将权重与偏置的值均在[-0.5,0.5]的范围内均匀采样

5 结果分析

5.1 基本指标定义

基于二分类任务的TP, FN, FP, TN定义，扩展到多分类，定义对于数字i（$0 \leq i \leq 9$）：

- 标签为i，分类为i 的样本数目记为TP_i
- 标签为i，分类不为i的样本数目记为FN_i
- 标签不为i，分类为i的样本数数目记为FP_i
- 标签不为i，分类不为i的样本数目记为TN_i

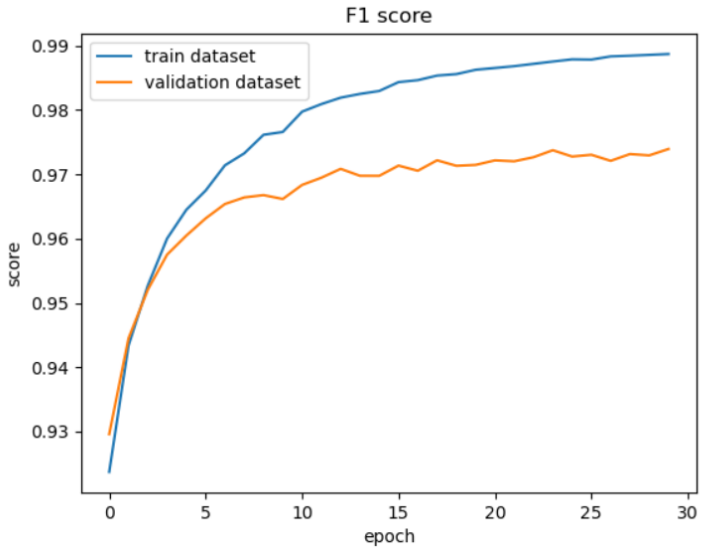
$$
\begin{aligned}
Precision_i &= \frac{TP_i}{TP_i + FP_i} \\
Recall_i &= \frac{TP_i}{TP_i + FN_i} \\
F1_i &= \frac{2 * Precision_i * Recall_i}{Precision_i + Recall_i}
\end{aligned}
$$

5.2 测试

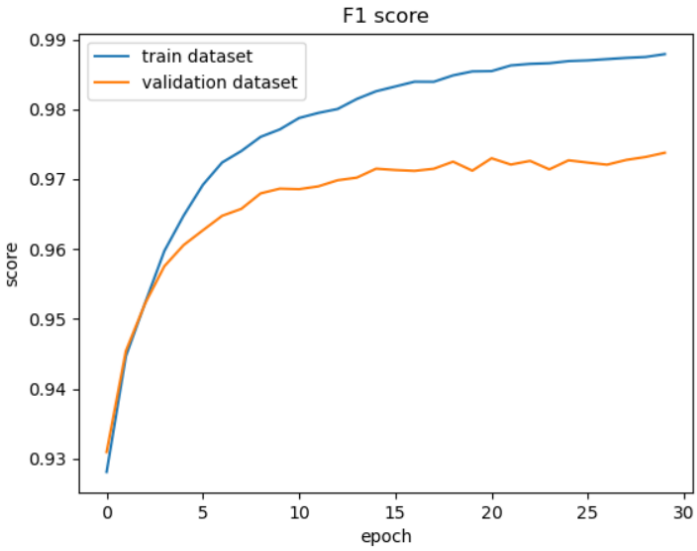
经过测试，在使用标准SGD时设置学习率`step`=0.08较为合理，可以使模型快速收敛的同时较少发生震荡；在使用mini-batch SGD时，设置`step`=0.08较为合理。

- 过拟合

  在标准SGD方法下，使用学习率为0.08，仅一个大小为100的隐藏层时进行实验，绘制训练过程中每个epoch后的结果在训练集和验证集上的得分，使模型运行30个epoch，得到的图像如下：



  而在Mini-Batch SGD方法下，使用学习率0.1得到的图像如下：



发现在20个epoch后，虽然模型可以持续在测试集上继续略微提高精度，但在验证集上的表现并不再会继续上升，表明此时模型已产生了过拟合的现象。

为了测试时间尽量缩短的同时保证模型训练充分的目的，以下测试中，均选取epoch=20，

由于时间有限，仅对激活函数为***sigmoid***进行了测试，若希望使用tanh作为激活函数，当前学习率过大，需要更改`main.py`的71与74行的step，重新确定学习率大小。

超参数调整仅对两种优化方法的学习率、batch大小、初始化方法、epoch轮次进行调整；并使用调整后的参数在测试集上进行最后的测试。由于观察到样本各类的分布较为均匀，预测结果的F1值方差一般情况下已小于0.01%，测试中选取每一类的**F1**得分的平均值作为模型的评判标准。

下表在`epoch`=20, `init_method`='uniform', `init_bound`=[-0.5, 0.5], 标准SGD方法对应学习率`step`=0.1，Mini-Batch SGD对应学习率`step`=0.08的情况下测得的结果

	网络层次架构	优化方法	**batch size**	训练集**F1**得分	验证集**F1**得分	测试集**F1**得分
1	784, 100, 10	Mini-Batch	64	0.9836	0.9721	0.9703
2	784, 100, 10	Mini-Batch	32	0.9848	0.9715	0.9699
3	784, 100, 10	Mini-Batch	16	0.9841	0.9720	0.9691
4	784, 250, 10	Mini-Batch	32	0.9887	0.9760	0.9754
5	784, 250, 100, 10	Mini-Batch	32	0.9931	0.9749	0.9763
6	784, 100, 10	Standard	-	0.9837	0.9717	0.9693
7	784, 250, 10	Standard	-	0.9891	0.9753	0.9751
8	784, 250, 100, 10	Standard	-	**0.9932**	**0.9767**	**0.9772**

可见，该实验中，Mini-Batch方法下的batch size大小影响较为微弱。

所有实验中模型均较为成功地进行了手写数字识别，各数字的F1平均分均超过了0.969

在所选取的所有BP神经网络结构中，发现每层的节点数分别为784, 250, 100, 10的神经网络的表现相对更好，在测试集上的F1得分均值达到了0.9772，预测准确率接近99.5%及以上（以上数据与随机种子有关，输入相同参数应得到相似结果，但不会完全一致）

6 讨论与思考

本次实验中，使用了结构模拟的方法，使用全连接神经网络对手写数字完成了识别。总体来讲，模型识别率较高，大部分情况下均可超过95%的成功率，非常有效，简单的结构模拟就足以完成传统算法难以完成的复杂任务。未来也许可以尝试实现Adam方法进行算法的进一步优化。

神经网络模型如今已称为人工智能界的基础，在简单的BP神经网络的基础上，逐步设计与发展出了更加复杂的各类神经网络，例如残差连接网络、注意力头、循环神经网络等等，得以在更多更加复杂的任务上取得成功。

可见，借鉴大脑的生理结构是向实现人工智能的有效道路之一。但目前人类对于大脑本身结构的理解并不透彻，巨量的细节仍有待神经科学家进一步的验证、探索。也许通过比较人工神经网络与大脑表现上的差异与增删神经网络功能后的表现，可以一定程度上帮助人们理解自己的大脑的工作方式。

视觉领域是AI迅速发展的领域之一，涌现了许多非常成功的网络框架，在一些任务上的表现得以接近甚至超过人类的水平。近年，也有很多关于使用已有的成功的视觉网络的层内活动情况与大脑的各个视觉脑区进行对比研究的工作出现，甚至通过将人脑的fMRI信号转换为其最可能对应的人工神经网络的活动情况，从而进行视觉图片的重建工作。

未来在大脑结构与机制上的发现也许会带来更多的AI领域中的技术革新，或者发生计算能力的大幅提升，使人类得以制造出更加智能、通用的AI，为人类社会解放更多的生产力。