

实验三 神经网络

1 问题描述

利用神经网络算法对 MNIST 数据集进行分类

2 实现步骤与流程

BP 算法通过链式法则将误差对参数的梯度逐级传播，并通过梯度下降算法对参数进行优化

加载数据集

实验提供的 MNIST 数据集文件为.idx3-ubyte 和.idx1-ubyte 格式，需要对读取到的二进制序列进行处理，转化为 numpy 可以进行运算的 ndarray 的格式

定义基础模块

神经网络可能由多个模块构成，为了简化网络模型的构建过程，需要对神经网络中的模块概念进行封装，并定义模块的前向传播以及反向传播机制，本次实验实现了多种基本模块，包括全连接层（Linear）、标准化层（LayerNormalization）以及常见的激活函数 relu 层、sigmoid 层、tanh 层以及 softmax 层，并提供了简单的序列神经网络模块（Sequence）用于堆叠各种模块形成网络

定义损失函数

为了对网络中的参数进行优化，还需要定义损失函数，包括损失函数的计算以及梯度的回传。本次实验提供了交叉熵损失函数（CrossEntropy）用于多分类任务的优化

构建训练网络

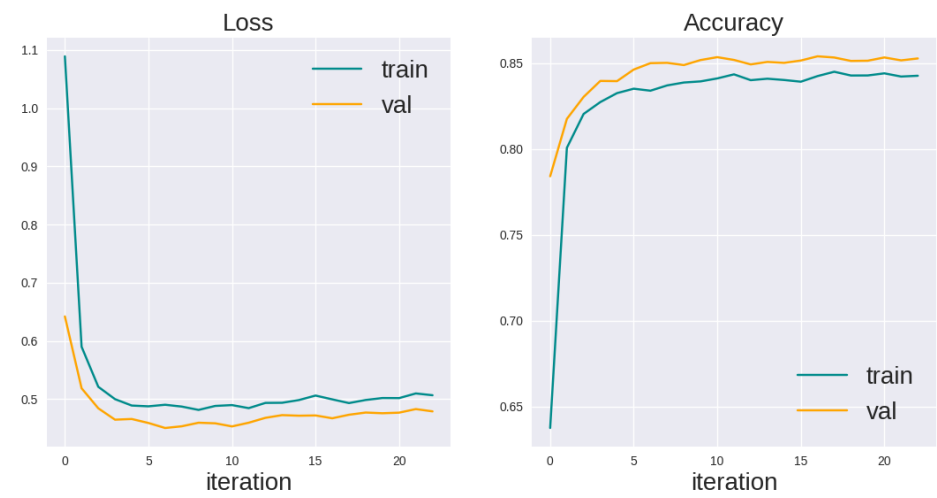
在初始化数据集以及构建好神经网络后，即可对网络中的参数进行学习训练。本次实验采用小批次梯度下降算法，为了加快网络的收敛速度的同时保证网络收敛的稳定性，实验中采用了学习率的指数衰减策略，并且

为了减少网络过拟合的可能，实验中实现了网络的提前体制（early-stop）。在训练过程中记录网络模型在训练集以及验证集上的平均损失函数以及准确率的历史数据，训练结束后可视化历史数据，观察网络的训练过程

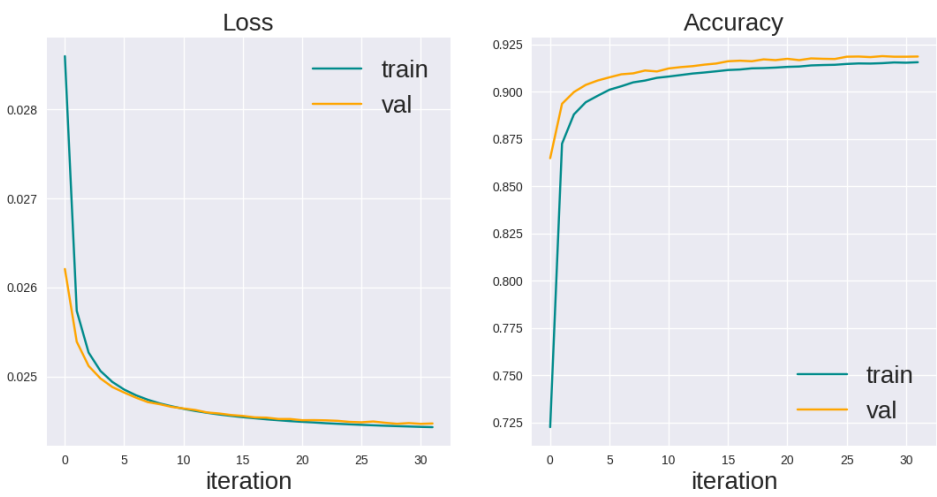
3 实验结果与分析

由于实验要求的 MindSpore 框架相关的资料较为匮乏，因此本次实验采取深度学习框架 pytorch 与实验中的算法进行对比。pytorch 是一个开源的 python 深度学习库，可以用于构建能够自动求导的神经网络。

以下是实验中构建的神经网络的训练历史和相同结构的网络在 pytorch 下的训练历史



(a) 历史数据（实验）



(b) 历史数据（pytorch）

图 1: 神经网络训练历史

可以看出，随着训练轮数的增加，网络模型在分类的准确率上不断上升最终达到收敛。与 pytorch 的算法

相比，实验的算法最终的收敛准确率较低，在 85% 左右，而 pytorch 最终的准确率可以达到接近 92%，并且在实验算法每个轮次需要约 1.6s 的运行时间，而 pytorch 每个轮次需要 1.2s 的运行时间，在时间效率上更高。

经过实际测试，本次实验采用的单层网络结构在训练效率以及准确率上的综合表现较好。

4 MindSpore 学习使用心得体会

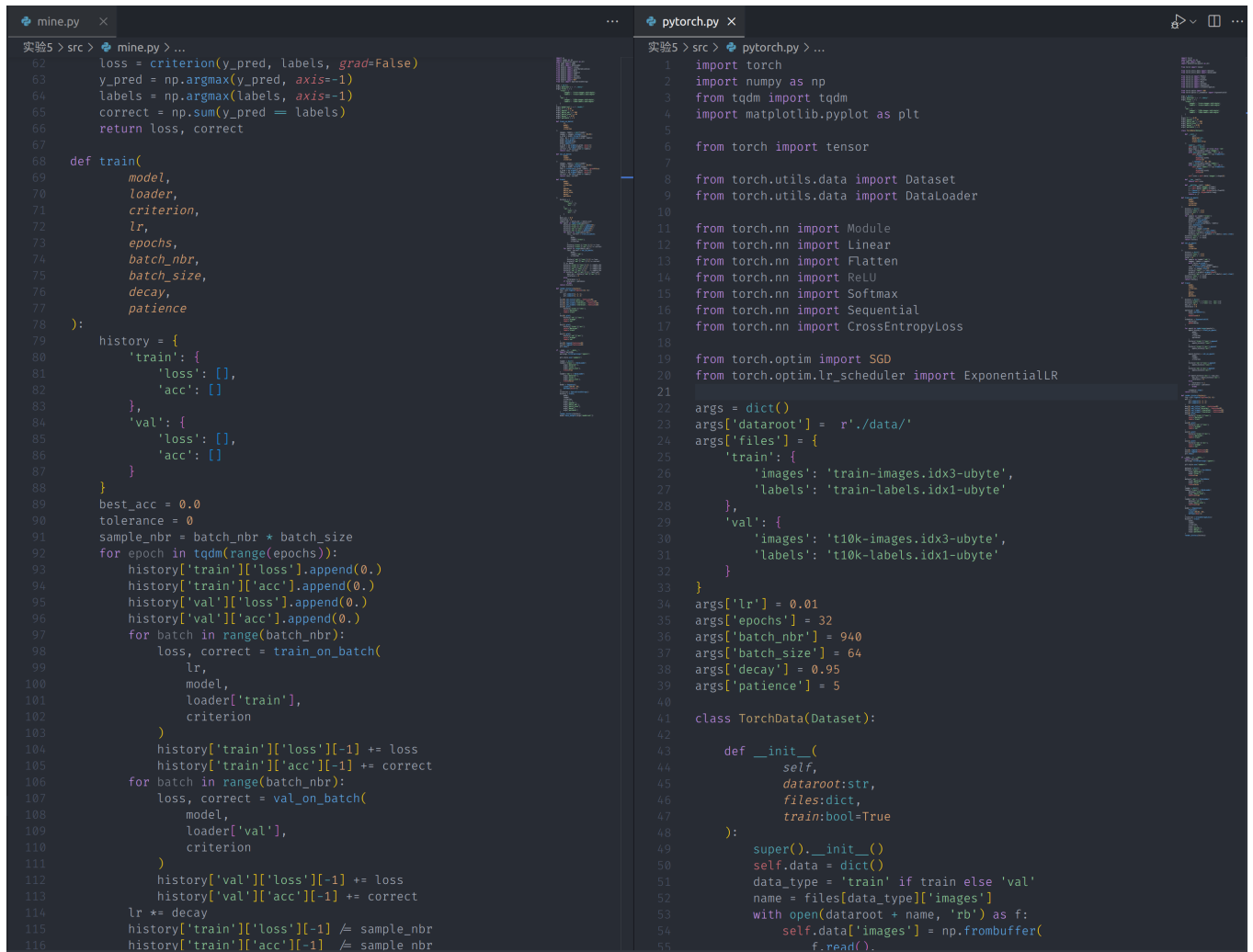
由于本次实验并未采用 MindSpore，此部分用 pytorch 的学习使用心得体会来代替。

本次实验中调用了 pytorch 中的以下算法接口

- 神经网络模块 (nn)
- 数据集接口 (utils data Dataset、DataLoader)
- SGD 优化器 (optim SGD)
- 学习率指数衰减 (optim lr_scheduler ExponentialLR)

作为实验算法的对照算法，pytorch 提供的算法接口方便调用，并且执行效率较高，能够高效地实现实验的需求。

5 代码附录



```
mine.py
实验5 > src > mine.py > ...
62     loss = criterion(y_pred, labels, grad=False)
63     y_pred = np.argmax(y_pred, axis=-1)
64     labels = np.argmax(labels, axis=-1)
65     correct = np.sum(y_pred == labels)
66     return loss, correct
67
68 def train(
69     model,
70     loader,
71     criterion,
72     lr,
73     epochs,
74     batch_nbr,
75     batch_size,
76     decay,
77     patience
78 ):
79     history = {
80         'train': {
81             'loss': [],
82             'acc': []
83         },
84         'val': {
85             'loss': [],
86             'acc': []
87         }
88     }
89     best_acc = 0.0
90     tolerance = 0
91     sample_nbr = batch_nbr * batch_size
92     for epoch in tqdm(range(epochs)):
93         history['train']['loss'].append(0.)
94         history['train']['acc'].append(0.)
95         history['val']['loss'].append(0.)
96         history['val']['acc'].append(0.)
97         for batch in range(batch_nbr):
98             loss, correct = train_on_batch(
99                 lr,
100                 model,
101                 loader['train'],
102                 criterion
103             )
104             history['train']['loss'][-1] += loss
105             history['train']['acc'][-1] += correct
106         for batch in range(batch_nbr):
107             loss, correct = val_on_batch(
108                 model,
109                 loader['val'],
110                 criterion
111             )
112             history['val']['loss'][-1] += loss
113             history['val']['acc'][-1] += correct
114         lr *= decay
115         history['train']['loss'][-1] /= sample_nbr
116         history['train']['acc'][-1] /= sample_nbr

pytorch.py
实验5 > src > pytorch.py > ...
1  import torch
2  import numpy as np
3  from tqdm import tqdm
4  import matplotlib.pyplot as plt
5
6  from torch import tensor
7
8  from torch.utils.data import Dataset
9  from torch.utils.data import DataLoader
10
11  from torch.nn import Module
12  from torch.nn import Linear
13  from torch.nn import Flatten
14  from torch.nn import ReLU
15  from torch.nn import Softmax
16  from torch.nn import Sequential
17  from torch.nn import CrossEntropyLoss
18
19  from torch.optim import SGD
20  from torch.optim.lr_scheduler import ExponentialLR
21
22  args = dict()
23  args['dataroot'] = 'r'./data/'
24  args['files'] = {
25      'train': {
26          'images': 'train-images.idx3-ubyte',
27          'labels': 'train-labels.idx1-ubyte'
28      },
29      'val': {
30          'images': 't10k-images.idx3-ubyte',
31          'labels': 't10k-labels.idx1-ubyte'
32      }
33  }
34  args['lr'] = 0.01
35  args['epochs'] = 32
36  args['batch_nbr'] = 940
37  args['batch_size'] = 64
38  args['decay'] = 0.95
39  args['patience'] = 5
40
41  class TorchData(Dataset):
42
43      def __init__(
44          self,
45          dataroot:str,
46          files:dict,
47          train:bool=True
48      ):
49          super().__init__()
50          self.data = dict()
51          data_type = 'train' if train else 'val'
52          name = files[data_type]['images']
53          with open(dataroot + name, 'rb') as f:
54              self.data['images'] = np.frombuffer(
55                  f.read(),
```

图 2: 部分代码截图