学习笔记

Zhengjv

2021 年 6 月 11 日

目录

1 深度学习

1.1 实验（基于 pytorch）

1.1.1 使用 CNN 进行多分类

数据集准备 下载 MNIST 数据集，对手写数字进行识别分类。

原理

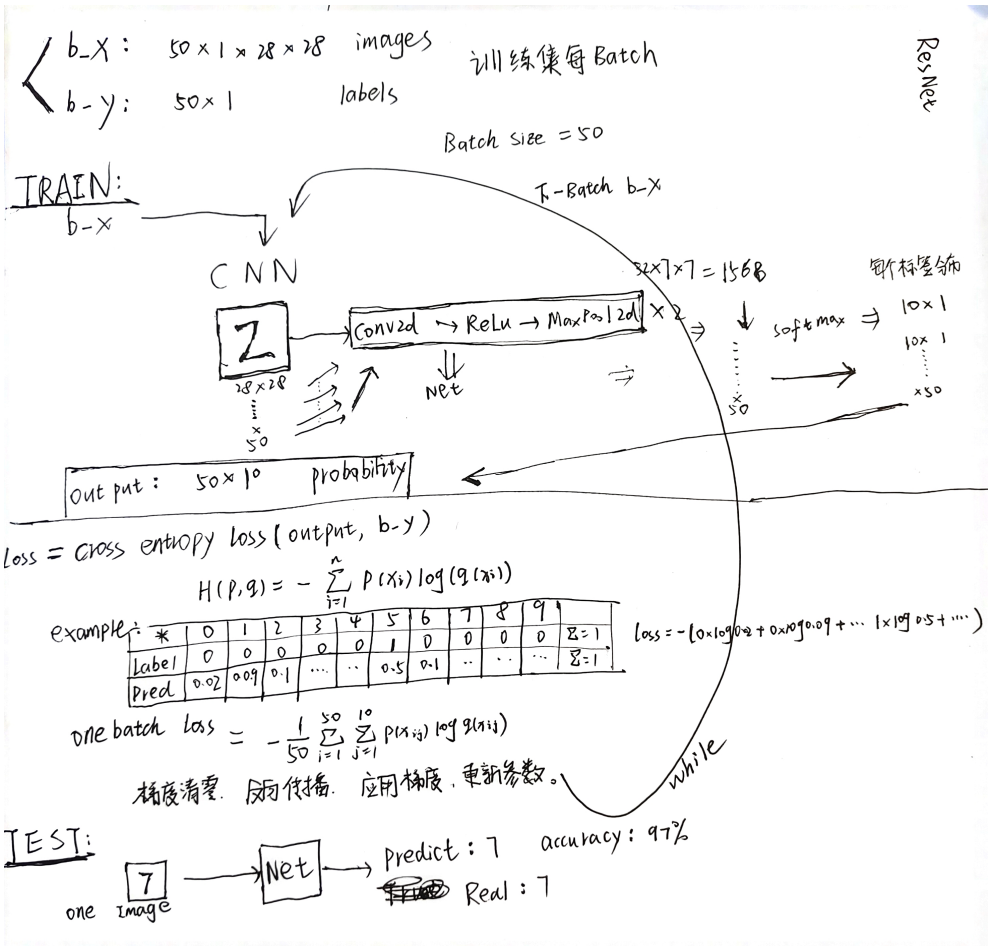

图 1: 原理

超参数设置 选择 batch size=50; LR = 0.001 进行训练。只训练一个 epoch

网络 图像大小为 28*28，网络结构如图：

第一层卷积，16 个 5*5 滤波器；激活函数为 RELU；最大池化；

第二层卷积，16 个 5*5 滤波器；激活函数为 RELU；最大池化；

全连接层 +softmax，计算出每个标签预测概率分布。

训练 采用交叉熵损失；Adam 方法进行梯度下降

```
CNN(
  (conv1): Sequential(
    (0): Conv2d(1, 16, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (conv2): Sequential(
    (0): Conv2d(16, 32, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (out): Linear(in_features=1568, out_features=10, bias=True)
)
```

图 2: 网络结构

测试

代码 ：

```
import os
import torch
import torch.nn as nn
import torch.utils.data as Data
import torchvision
import matplotlib.pyplot as plt


EPOCH = 1
BATCH_SIZE = 50
LR = 0.001
DOWNLOAD_MNIST = False

if not(os.path.exists('./mnist/')) or not os.listdir('./mnist/'):
    DOWNLOAD_MNIST = True

train_data = torchvision.datasets.MNIST(
    root='./mnist/',
    train=True,
    transform=torchvision.transforms.ToTensor(),
```

```
    download=DOWNLOAD_MNIST,
)

print(train_data.train_data.size())
# (60000, 28, 28)
print(train_data.train_labels.size())
# (60000)

plt.imshow(train_data.train_data[0].numpy(), cmap='gray')
plt.title('%i' % train_data.train_labels[0])
plt.show()

# batch_size = (50, 1, 28, 28)
train_loader = Data.DataLoader(dataset=train_data, batch_size=
    BATCH_SIZE, shuffle=True)

# pick 2000 samples to speed up testing
test_data = torchvision.datasets.MNIST(root='./mnist/', train=
    False)
test_x = torch.unsqueeze(test_data.test_data, dim=1).type(torch
    .FloatTensor)[:2000]/255.
# shape from (2000, 28, 28) to (2000, 1, 28, 28), value in
    range(0,1)
test_y = test_data.test_labels[:2000]


class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Sequential(# input shape (1, 28, 28)
            nn.Conv2d(
                in_channels=1,      # input height
                out_channels=16,    # 16_filters
                kernel_size=5,      # kernel size=5
                stride=1,           # step
                padding=2,    # padding=(kernel_size-1)/2 if
                    stride=1
            ),                      # output shape (16, 28, 28)
            nn.ReLU(),
```

```
55              nn.MaxPool2d(kernel_size=2),
56            # choose max value in 2x2 area, output shape (16, 14,
                  14)
57          )
58          self.conv2 = nn.Sequential(    # input shape (16, 14,
                14)
59              nn.Conv2d(16, 32, 5, 1, 2), # output shape (32, 14,
                     14)
60              nn.ReLU(),
61              nn.MaxPool2d(2),           # output shape (32, 7, 7)
62          )
63          self.out = nn.Linear(32 * 7 * 7, 10)
64            # fully connected layer, output 10 classes
65
66      def forward(self, x):
67          x = self.conv1(x)
68          x = self.conv2(x)
69          x = x.view(x.size(0), -1)
70          output = self.out(x)
71          return output,x
72
73  cnn = CNN()
74  print(cnn)  # 网络结构
75
76
77  optimizer = torch.optim.Adam(cnn.parameters(), lr=LR)
78  loss_func = nn.CrossEntropyLoss()
79
80
81  # training and testing
82  for epoch in range(EPOCH):
83      for step, (b_x, b_y) in enumerate(train_loader):
84
85
86          output = cnn(b_x)[0]
87
88          #每批50张图输入神经网络, output.shape() = (50,10)
89
90          loss = loss_func(output, b_y)
```

```
91                  #计算交叉熵损失
92          optimizer.zero_grad()
93                  #梯度清零
94          loss.backward()
95                  #反向传播，计算梯度
96          optimizer.step()
97                  #应用梯度
98
99          if step % 50 == 0:
100             test_output, last_layer = cnn(test_x)
101             pred_y = torch.max(test_output, 1)[1].data.numpy()
102             accuracy = float((pred_y == test_y.data.numpy()).
                    astype(int).sum()) / float(test_y.size(0))
103             print('Epoch: ', epoch, '| train loss: %.4f' % loss
                    .data.numpy(), '| test accuracy: %.2f' %
                    accuracy)
104 #测试
105 test_output, _ = cnn(test_x[:10])
106 pre_y = torch.max(test_output, 1)[1].data.numpy()
107 print(pre_y, 'prediction number')
108 print(test_y[:10].numpy(), 'real number')
```

```
Epoch:  0 | train loss: 0.1402 | test accuracy: 0.97
Epoch:  0 | train loss: 0.0526 | test accuracy: 0.97
Epoch:  0 | train loss: 0.0391 | test accuracy: 0.97
Epoch:  0 | train loss: 0.1452 | test accuracy: 0.97
Epoch:  0 | train loss: 0.0306 | test accuracy: 0.98
Epoch:  0 | train loss: 0.0440 | test accuracy: 0.97
Epoch:  0 | train loss: 0.1290 | test accuracy: 0.97
Epoch:  0 | train loss: 0.2930 | test accuracy: 0.97
Epoch:  0 | train loss: 0.2125 | test accuracy: 0.98
[7 2 1 0 4 1 4 9 5 9] prediction number
[7 2 1 0 4 1 4 9 5 9] real number

Process finished with exit code 0
```

图 3: 结果

结果

2 刷题笔记

2.1 电话号码的字母组合

给定一个仅包含数字 2-9 的字符串，返回所有它能表示的字母组合。答案可以按任意顺序返回。

给出数字到字母的映射如下（与电话按键相同）。注意 1 不对应任何字母。

2.1.1 回溯

思路 对于这种求排列组合的枚举问题，用递归回溯的方法是最为合理的，可以尝试建立一个长度为 8 的新的链表，其中储存每个数字所代表的字母。

定义回溯函数，以及结束条件。当输入的数字长度递减为 0 时结束，并将其得到的字符串添加入结果中。

若判断不为结束条件，则进入循环，递归调用回溯函数，并且将当前字符添加，当前输入字符串长度-1。

代码 ：

```
1  class Solution(object):
2      def letterCombinations(self, digits):
3          if len(digits)==0:
4              return []
5          list=["abc",'def','ghi','jkl','mno','pqrs','tuv','wxyz'
               ]
6          res=[]
7
8          def backtrack(con,digit):
9              if len(digit)==0:
10                 res.append(con)
11             else:
12
13                 for i in list[int(digit[0]) - 2]:
14                     backtrack(con+i,digit[1:])
15         backtrack('',digits)
```

```
16          return re
```

2.2 四数之和

给定一个包含 n 个整数的数组 nums 和一个目标值 target，判断 nums 中是否存在四个元素 a，b，c 和 d ，使得 a + b + c + d 的值与 target 相等?

找出所有满足条件且不重复的四元组。

注意：答案中不可以包含重复的四元组。

2.2.1 暴力 + 双指针优化

思路 类似于之前的三数之和，对于四个数字采取暴力枚举的方法来判断，并将其排序以剔除重复字段。对于第三和第四轮枚举，可以采用双指针的方法来优化，从而减少算法的时间复杂度。

代码 ：

```
1  class Solution:
2      def fourSum(self, nums, target)
3          res=[]
4          if not nums or len(nums) < 4:
5              return res
6          nums.sort()
7          length = len(nums)
8          for i in range(length - 3):
9              if i > 0 and nums[i] == nums[i - 1]:
10                 continue
11             if nums[i] + nums[i + 1] + nums[i + 2] + nums[i +
                   3] > target:
12                 break
13             if nums[i] + nums[length - 3] + nums[length - 2] +
                   nums[length - 1] < target:
14                 continue
15             for j in range(i + 1, length - 2):
16                 if j > i + 1 and nums[j] == nums[j - 1]:
17                     continue
```

```
18                 if nums[i] + nums[j] + nums[j + 1] + nums[j +
                       2] > target:
19                     break
20                 if nums[i] + nums[j] + nums[length - 2] + nums[
                       length - 1] < target:
21                     continue
22                 left, right = j + 1, length - 1
23                 while left < right:
24                     total = nums[i] + nums[j] + nums[left] +
                           nums[right]
25                     if total == target:
26                         res.append([nums[i], nums[j], nums[left
                               ], nums[right]])
27                         while left < right and nums[left] ==
                               nums[left + 1]:
28                             left += 1
29                         left += 1
30                         while left < right and nums[right] ==
                               nums[right - 1]:
31                             right -= 1
32                         right -= 1
33                     elif total < target:
34                         left += 1
35                     else:
36                         right -= 1
37
38         return res
```

2.3 删除链表的倒数第 N 个节点

给你一个链表，删除链表的倒数第 n 个结点，并且返回链表的头结点。

进阶：你能尝试使用一趟扫描实现吗?

示例 1：

输入：head = [1,2,3,4,5], n = 2

输出：[1,2,3,5]

示例 2：

输入：head = [1], n = 1

输出：[]
示例 3：
输入：head = [1,2], n = 1
输出：[1]

2.3.1 解决方法

思路 使用双指针，来得以实现一次遍历，在常数空间下解决问题。使用两个指针 first 和 second 同时遍历链表，但是 first 比 second 领先 n 个节点，

这样一来，当 first 遍历到 null 时，second 正好在第 n 个节点，这时候只要将 second 所在节点删除即可。

代码 ：

```
1  class Solution:
2      def removeNthFromEnd(self, head: ListNode, n: int) ->
           ListNode:
3          res=ListNode(0,head)
4          first=head
5          second=res
6          while n:
7              # first.
8
9              first=first.next
10             n=n-1
11
12         while first:
13
14             first=first.next
15             second=second.next
16
17         second.next=second.next.next
18         return res.next
```

2.4 有效的括号

给定一个只包括'('，')'，''，''，'['，']' 的字符串 s ，判断字符串是否有效。

有效字符串需满足：
左括号必须用相同类型的右括号闭合。
左括号必须以正确的顺序闭合。

2.4.1 解决方法

思路 利用栈来实现括号匹配，建立左右括号的字典键值对，若遇到左括号，将其入栈。若遇到右括号，将当前字符与栈顶元素匹配，若不匹配或栈空，则返回 false。

若匹配，将栈顶元素 pop 出。最终遍历完后，若字符串为空，则返回 True。否则返回 false。

代码 ：

```
class Solution:
    def isValid(self, s: str) -> bool:
        l=[]
        d={
            ")": "(",
            "]": "[",
            "}": "{",
        }
        for i in s:
            if i in d:
                if not l or l[-1]!=d[i]:
                    return False
                l.pop()
            else:
                l.append(i)
        return not l
```