Handwritten digit recognition based on MLP

Yifan Liang
Department of Electrical Engineering
Stevens Institute of Technology
`yliang33@stevens.edu`

Abstract

This project is aimed to design a Multilayer Perception model, which could be trained to recognize the hand-written number in a single picture with considerable accuracy. The training set and test set are both chosen from the popular mature datasets MNIST. It is showed how different learning rates will influence the iterative recognition accuracy. This project is also the base for future attempt to achieve precise recognition for the whole hand-written text.

1 Introduction

Image recognition is an important part in today's computer vision and relevant fields. The development of image recognition has gone through three stages: character recognition, digital image processing and recognition, and object recognition. The research on character recognition began in 1950, and generally recognizes letters, numbers and symbols. From printed character recognition to handwritten character recognition, it is widely used. So it is of great value to find more efficient methods for image recognition, especially with the rapidly developing machine learning technology.

1.1 Development of computer vision and image recognition

Computer vision is a science that studies how to make machines "see". More specifically, it refers to the use of cameras and computers instead of human eyes to identify, track and measure targets, and further image processing , Processed by computer into images more suitable for human eye observation or transmitted to the instrument for detection.

Image recognition refers to the technology of using computers to process, analyze, and understand images to identify targets and objects in various modes. It is a practical application of deep learning algorithms. At present, practical image recognition technology is generally divided into face recognition and product recognition. Face recognition is mainly used in security inspection, identity verification and mobile payment; product recognition is mainly used in the process of product circulation, especially unmanned retail, smart retail cabinets, etc.

Image recognition is an important field of artificial intelligence. In order to compile computer programs that simulate human image recognition activities, different image recognition models have been proposed. For example, template matching model. This model believes that to recognize an image, the memory pattern of this image must be known in the past experience, also known as the template. If the current stimulus matches the template in the brain, this image will be recognized. For example, there is a letter A. If there is an A template in the brain, the size, orientation, and shape of the letter A are exactly the same as the A template, and the letter A is recognized. This model is simple and clear, and easy to get practical application. However, this model emphasizes that the image must be fully consistent with the template in the brain to be recognized. In fact, people can not only recognize images that are completely consistent with the template in the brain, but also recognize images that are not completely consistent with the template. For example, people can recognize not only a specific letter A, but also various printed letters, handwritten letters, misaligned letters, and letters A of different sizes. At the same time, there are a large number of images that humans can recognize. It is also impossible if each recognized image has a corresponding template in the brain.

1.2 Relationship between computer vision and image recognition

Computer vision, image processing, image analysis, robot vision and machine vision are closely related disciplines. If we open the textbook with the above names, you will find that they have a considerable overlap in technology and application fields. This shows that the basic theories of these disciplines are roughly the same, and even makes people suspect that they are the same discipline and are given different names.

In fact, the research objects of computer vision are mainly three-dimensional scenes mapped onto single or multiple images, such as the reconstruction of three-dimensional scenes. Computer vision research is largely directed at the content of images.

The research objects of image processing and image analysis are mainly two-dimensional images to realize image conversion, especially for pixel-level operations, such as improving image contrast, edge extraction, denoising and geometric transformations such as image rotation. This feature indicates that whether it is image processing or image analysis, its research content has nothing to do with the specific content of the image. The mathematical essence of the image recognition problem belongs to the mapping problem from pattern space to category space. At present, in the development of image recognition, there are three main recognition methods: statistical pattern recognition, structural pattern recognition, and fuzzy pattern recognition.

As a result, for every future researcher focusing on such related fields should have the literacy to establish corresponding basic models and the ability to solve practical problems. The MNIST dataset is actually a good start to do so.

1.3 Multiplayer perception

There are a lot of machine learning method developed these years to better solve image recognition problem, such as the k-nearest neighbors (KNN) algorithm, support vector machine (SVM), badger prairie needs network, convolutional neural network(CNN). We are going to use Multiplayer perception as the simpler model to handle this problem.

Multilayer Perceptron (MLP) is a forward-structured artificial neural network that maps a set of input vectors to a set of output vectors. MLP can be regarded as a directed graph, composed of multiple node layers, each layer is fully connected to the next layer. Except for the input node, each node is a neuron (or processing unit) with a nonlinear activation function. MLP sometimes plays useful role when handling non-linear problem.

2 Analysis on specific problem

First of all, we should get the MNIST hand-written number dataset, which includes the training set and test set. The required datasets can be downloaded from the website easily. It's also one of the most important reason why MNIST becomes so popular.

`http://yann.lecun.com/exdb/mnist/`

After download, we get four files which each other contain the training and test set and their labels.

To show the datasets more directly, we can use following python codes to split them into single png files. For instance, we perform the seperation on training set:

```
import numpy as np
import struct
from PIL import Image
import os
```

```
data_file = './data/raw/train-images-idx3-ubyte'
data_file_size = 47040016
data_file_size = str(data_file_size - 16) + 'B'

data_buf = open(data_file, 'rb').read()

magic, numImages, numRows, numColumns = struct.unpack_from(
'>IIII', data_buf, 0)
datas = struct.unpack_from(
'>' + data_file_size, data_buf, struct.calcsize('>IIII'))
datas = np.array(datas).astype(np.uint8).reshape(
numImages, 1, numRows, numColumns)

label_file = './data/raw/train-labels-idx1-ubyte'

label_file_size = 60008
label_file_size = str(label_file_size - 8) + 'B'

label_buf = open(label_file, 'rb').read()

magic, numLabels = struct.unpack_from('>II', label_buf, 0)
labels = struct.unpack_from(
'>' + label_file_size, label_buf, struct.calcsize('>II'))
labels = np.array(labels).astype(np.int64)

datas_root = 'mnist_train'
if not os.path.exists(datas_root):
os.mkdir(datas_root)

for i in range(10):
file_name = datas_root + os.sep + str(i)
if not os.path.exists(file_name):
os.mkdir(file_name)

for ii in range(numLabels):
img = Image.fromarray(datas[ii, 0, 0:28, 0:28])
label = labels[ii]
file_name = datas_root + os.sep + str(label) + os.sep + \
'mnist_train_' + str(ii) + '.png'
img.save(file_name)
```

According to the official introduction of MNIST, the data file size is 47040016B, but we should set to 47040000B. For the same reason, the label file size should be set to 60000B from 60008B.

Now we can get see the result:

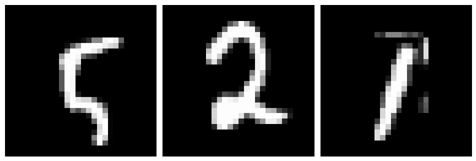


Figure 1: Examples of handwritten numbers.

3 Design on MLP model

3.1 The simple Multilayer Perception network

First we try to establish a simple full connected linear network with two hidden layers. We should introduce the activation function. Here we choose ReLU:

```
import torch
from  torch.autograd import *
from  torch import nn,optim
from  torch.utils.data import DataLoader
from  torchvision import datasets,transforms

class Activation_Net(nn.Module):
def __init__(self, in_dim, n_hidden_1, n_hidden_2, out_dim):
super().__init__()
self.layer1 = nn.Sequential(nn.Linear(in_dim, n_hidden_1),nn.ReLU(True))
self.layer2 = nn.Sequential(nn.Linear(n_hidden_1, n_hidden_2),nn.ReLU(True))
self.layer3 = nn.Sequential(nn.Linear(n_hidden_2, out_dim))

def forward(self, x):
x = self.layer1(x)
x = self.layer2(x)
x = self.layer3(x)
return x
```

In theory, this network should be able to get trained, but in some practical cases, the simple model will meet with difficulties.

3.2 Batch Normalization

We need to consider a new concept:Internal Covariate Shift, which describes a specific difficulty in training often occurring when training deep networks. After each parameter is iteratively updated, after the output data of the upper layer network is calculated by this layer of network, the distribution of the data will change, which brings difficulties to the learning of the next layer of network. If the distribution keeps changing, learning is difficult.

In addition to increasing the difficulty of model learning, Internal Covariate Shift also causes the problem of gradient disappearance.

The activation input value of the deep neural network before the non-linear transformation's distribution as the network deepens or during the training process, gradually shifts or changes. Generally the overall distribution is gradually approaching the upper and lower limits of the value range of the nonlinear function (for the Sigmoid function as an instance, it means that the activation input value is a large negative or positive value), so this leads to disappearance of the gradient of the low-level neural network during propagation, which is the essential reason why the training deep neural network converges more and more slowly.

Another difficulty is called Covariate Shift. nternal Covariate Shift and Covariate Shift are similar, but not one thing. The former occurs inside the neural network, so it is InternalInternal, and the latter occurs on the input data. Covariate Shift mainly describes that due to the difference in the distribution of training data and test data, it affects the generalization and training speed of the network. The method we often use is normalization or whitening.

Solving these problems means promoting the model performance. BatchNorm is to forcibly pull the distribution of the input value of any neuron in each layer of the neural network back to the normal distribution through certain normalization means, based on the Independent Identical Distribution hypothesis(IID). This makes the activation input value fall in the area where the nonlinear function is more sensitive to the input, so that a small change in the input will cause a larger change in the loss function, which means that the gradient will be larger to avoid the problem of gradient disappearance, and the gradient will become larger which means fast learning convergence and can greatly speed up training.

For each hidden layer neuron, the input distribution that is gradually mapped to the nonlinear function and then close to the limit saturation region is forced to return to a more normal normal distribution, so that the input value of the nonlinear transformation function falls into more sensitive areas to avoid the problem of gradient disappearance. Because the gradient can always maintain a relatively large state, it is obvious that the parameter adjustment efficiency of the neural network is relatively high.

Here is the revised model:

```
class Batch_Net(nn.Module):
def __init__(self, in_dim, n_hidden_1, n_hidden_2, out_dim):
super().__init__()
self.layer1 = nn.Sequential(nn.Linear(in_dim, n_hidden_1),
nn.BatchNorm1d(n_hidden_1),nn.ReLU(True))
self.layer2 = nn.Sequential(nn.Linear(n_hidden_1, n_hidden_2),
nn.BatchNorm1d(n_hidden_2),nn.ReLU(True))
self.layer3 = nn.Sequential(nn.Linear(n_hidden_2, out_dim))

def forward(self, x):
x = self.layer1(x)
x = self.layer2(x)
x = self.layer3(x)
return x
```

3.3 Training model

After establish the MLP model, we can start training it. In order to further improve the training speed, we check for the availabilty of GPU, and show the loss change after every iteration:

```
batch_size=64
learning_rate=1e-1
num_epoches=20

data_tf=transforms.Compose([transforms.ToTensor(),
transforms.Normalize([0.5],[0.5])])
train_dataset=datasets.MNIST(root='./data',train=True,
transform=data_tf,download=True)
test_dataset=datasets.MNIST(root="./data",train=False,transform=data_tf)
train_loader=DataLoader(train_dataset,batch_size=batch_size,shuffle=True)
test_loader=DataLoader(test_dataset,batch_size=batch_size,shuffle=False)

model=Batch_Net(28*28,300,100,10)
if torch.cuda.is_available():
model=model.cuda()
criterion=nn.CrossEntropyLoss()
optimizer=optim.SGD(model.parameters(),lr=learning_rate)

for epoch in range(num_epoches):
loss_sum, cort_num_sum,acc = 0.0, 0,0
for data in train_loader:
img,label=data
img=img.view(img.size(0),-1)
if torch.cuda.is_available():
inputs = Variable(img).cuda()
```

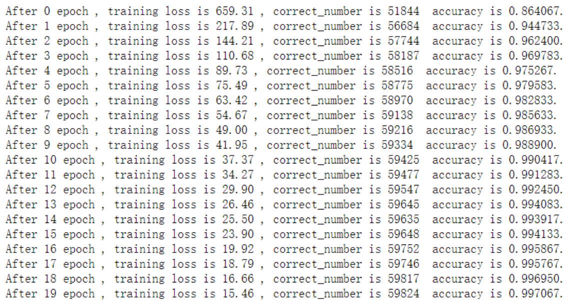
```
target = Variable(label).cuda()
else:
inputs = Variable(img)
target = Variable(label)
output =model(inputs)
loss = criterion(output, target)
optimizer.zero_grad()
loss.backward()
optimizer.step()
loss_sum += loss.data
_, pred = output.data.max(1)
num_correct = pred.eq(target).sum()
cort_num_sum += num_correct
acc=cort_num_sum.float()/len(train_dataset)
print( "After %d epoch , training loss is %.2f ,
correct_number is %d  accuracy is %.6f. "%(epoch,loss_sum,cort_num_sum,acc))

model.eval()
eval_loss=0
eval_acc=0
for data in test_loader:
img ,label =data
img=img.view(img.size(0),-1)
if torch.cuda.is_available():
img=Variable(img,volatile=True)
label=Variable(label,volatile=True)
else:
img = Variable(img, volatile=True)
label = Variable(label, volatile=True)
out=model(img)
loss=criterion(out,label)
eval_loss+=loss.data*label.size(0)
_,pred=out.data.max(1)
num_correct=pred.eq(label).sum()
eval_acc+=num_correct.data
print('Test loss: {:.6f},ACC: {:.6f}'.format(eval_loss.float()/(len(test_dataset)),
eval_acc.float()/(len(test_dataset))))
```

4 MLP performance and possible improvement

4.1 Training result

We can clearly see the iteration result with learning rate 0.01:

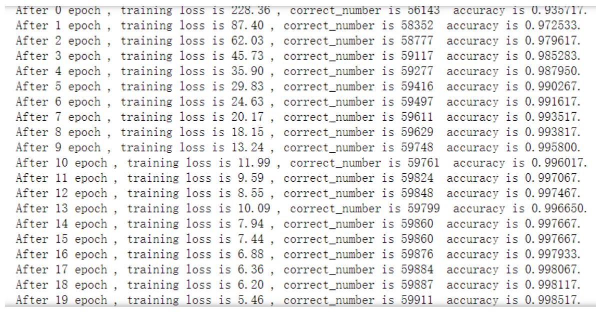


```
After 0 epoch , training loss is 659.31 , correct_number is 51844  accuracy is 0.864067.
After 1 epoch , training loss is 217.89 , correct_number is 56684  accuracy is 0.944733.
After 2 epoch , training loss is 144.21 , correct_number is 57744  accuracy is 0.962400.
After 3 epoch , training loss is 110.68 , correct_number is 58187  accuracy is 0.969783.
After 4 epoch , training loss is 89.73 , correct_number is 58516  accuracy is 0.975267.
After 5 epoch , training loss is 75.49 , correct_number is 58775  accuracy is 0.979583.
After 6 epoch , training loss is 63.42 , correct_number is 58970  accuracy is 0.982833.
After 7 epoch , training loss is 54.67 , correct_number is 59138  accuracy is 0.985633.
After 8 epoch , training loss is 49.00 , correct_number is 59216  accuracy is 0.986933.
After 9 epoch , training loss is 41.95 , correct_number is 59334  accuracy is 0.988900.
After 10 epoch , training loss is 37.37 , correct_number is 59425  accuracy is 0.990417.
After 11 epoch , training loss is 34.27 , correct_number is 59477  accuracy is 0.991283.
After 12 epoch , training loss is 29.90 , correct_number is 59547  accuracy is 0.992450.
After 13 epoch , training loss is 26.46 , correct_number is 59645  accuracy is 0.994083.
After 14 epoch , training loss is 25.50 , correct_number is 59635  accuracy is 0.993917.
After 15 epoch , training loss is 23.90 , correct_number is 59648  accuracy is 0.994133.
After 16 epoch , training loss is 19.92 , correct_number is 59752  accuracy is 0.995867.
After 17 epoch , training loss is 18.79 , correct_number is 59746  accuracy is 0.995767.
After 18 epoch , training loss is 16.66 , correct_number is 59817  accuracy is 0.996950.
After 19 epoch , training loss is 15.46 , correct_number is 59824  accuracy is 0.997067.
```

Figure 2: Iteration when learning rate=0.01.

And the result loss after test is 0.062430. The recognition accuracy is 0.950500.

Here the iteration result with larger learning rate 0.1 is also showed:



```
After 0 epoch , training loss is 228.36 , correct_number is 56143  accuracy is 0.935717.
After 1 epoch , training loss is 87.40 , correct_number is 58352  accuracy is 0.972533.
After 2 epoch , training loss is 62.03 , correct_number is 58777  accuracy is 0.979617.
After 3 epoch , training loss is 45.73 , correct_number is 59117  accuracy is 0.985283.
After 4 epoch , training loss is 35.90 , correct_number is 59277  accuracy is 0.987950.
After 5 epoch , training loss is 29.83 , correct_number is 59416  accuracy is 0.990267.
After 6 epoch , training loss is 24.63 , correct_number is 59497  accuracy is 0.991617.
After 7 epoch , training loss is 20.17 , correct_number is 59611  accuracy is 0.993517.
After 8 epoch , training loss is 18.15 , correct_number is 59629  accuracy is 0.993817.
After 9 epoch , training loss is 13.24 , correct_number is 59748  accuracy is 0.995800.
After 10 epoch , training loss is 11.99 , correct_number is 59761  accuracy is 0.996017.
After 11 epoch , training loss is 9.59 , correct_number is 59824  accuracy is 0.997067.
After 12 epoch , training loss is 8.55 , correct_number is 59848  accuracy is 0.997467.
After 13 epoch , training loss is 10.09 , correct_number is 59799  accuracy is 0.996650.
After 14 epoch , training loss is 7.94 , correct_number is 59860  accuracy is 0.997667.
After 15 epoch , training loss is 7.44 , correct_number is 59860  accuracy is 0.997667.
After 16 epoch , training loss is 6.88 , correct_number is 59876  accuracy is 0.997933.
After 17 epoch , training loss is 6.36 , correct_number is 59884  accuracy is 0.998067.
After 18 epoch , training loss is 6.20 , correct_number is 59887  accuracy is 0.998117.
After 19 epoch , training loss is 5.46 , correct_number is 59911  accuracy is 0.998517.
```

Figure 3: Iteration when learning rate=0.1.

The result loss after test is 0.061904. The recognition accuracy is 0.9839.

It is proved that for some reason in this model with the higher learning rate 0.1 comes a better recognition result.

4.2 Possible future improvement

After reviewing the whole model and design process, it is ensured that this MLP model can finish handwritten digit recognition in single image with high accuracy. The future attemt is supposed to be about designing additional algorithm to split digit in whole handwritten text or image and do recognition, to make it practically of great use.

References

[1] Ngiam J , Khosla A , Kim M , et al. Multimodal Deep Learning[C]// Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011. DBLP, 2011.

[2] Hayet Boughrara, Mohamed Chtourou, Chokri Ben Amar. MLP neural network based face recognition system using constructive training algorithm[M]. 2012.

[3] Rion Snow, Brendan O'connor, Daniel Jurafsky. Jurafsky D, Ng AY: Cheap and fast—but is it good?: evaluating non-expert annotations for natural language tasks[M]// Conference on Empirical Methods in Natural Language Processing 2008. 2008.