Deep Learning using Linear Support Vector Machines

Abstract

- Classification방법에서 대부분의 “Deep Learning”들은 예측과 cross-entropy loss를 최소화하기 위해서 softmax activation function을 사용한다. 본 논문에서는 softmax layer를 linear support vector machine으로 대체하여 Classification을 하는 방법에 대해 소개한다.
- Cross-entropy loss를 최소화하는 것이 아니라, margin-based loss를 최소화하도록 학습한다. 지금까지 neural nets과 SVM을 결합하는 많은 연구들이 있었지만, L2-SVM을 사용하여 성능 향상을 기대한다.

Introduction

- 본 논문에서는 MNIST, CIFAR-10, ICML 2013 Representation Learning Workshop’s face expressing recognition challenge의 dataset을 사용하여 실험을 진행하였다.
- 이번 Paper-reproduction에서는 Face expression recognition dataset을 사용하여 **CNN+softmax** 와 **CNN+L2SVM**의 성능을 비교하려고 한다.

Dataset

1) Face expression recognition dataset

- 28,709개의 Face expression 이미지가 있고, 각 이미지의 크기는 모두 동일하게 48×48 크기를 갖는다. 그리고 Class Label은 {Angry, Disgust, Fear, Happy, Sad, Surprise, Neutral}로 총 7개를 갖는다.

```
for images, _ in train_Loader:

    print('images.shape:', images.shape)
    plt.figure(figsize=(16, 8))
    plt.axis("off")
    plt.imshow(make_grid(images, nrow=8).permute((1, 2, 0)))
    break

_ = plt.suptitle("Images", y=0.92, fontsize=16)
```

Code

1) Library & Device Setting & Data import

```
import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt
import torch
from torch.utils.data import Dataset, DataLoader , TensorDataset
from torchvision import transforms , utils
from torchvision.utils import make_grid

is_cuda = torch.cuda.is_available()

if is_cuda:
    device = torch.device("cuda")
    print("GPU is available")
else:
    device = torch.device("cpu")
    print("GPU not available, CPU used")
```

- 본 논문의 구현은 Pytorch를 이용하여 구현하였고, Cuda Device를 사용하여 학습을 진행

Face Expression recognition dataset import

```
path = "C:/Users/KJW/Desktop/비즈니스/train.csv/"

train_data = pd.read_csv(path + "train.csv")
test_data = pd.read_csv(path + "test.csv")

def prepare_data(data):

    image_array = torch.zeros((len(data) , 48 , 48))
    image_label = np.array(list(map(int , data["emotion"])))

    for i, row in enumerate(data.index):

        image = np.fromstring(data.loc[row , "pixels"] , dtype = int , sep = " ")
        image = image/255.0
        image = np.reshape(image , (48,48))

        image_array[i] = torch.from_numpy(image)

    return image_array , torch.tensor(image_label , dtype = torch.float32)


emotions = {0: 'Angry', 1: 'Disgust', 2: 'Fear', 3: 'Happy', 4: 'Sad', 5: 'Surprise', 6: 'Neutral'}

train_image_array , train_image_label = prepare_data(train_data)
test_image_array , test_image_label = prepare_data(test_data)

train_image_array = train_image_array.reshape((train_image_array.size(0) , 1 , 48 , 48))
test_image_array = test_image_array.reshape((test_image_array.size(0) , 1 , 48 , 48))

train_dataset = TensorDataset(train_image_array , train_image_label)
test_dataset = TEnsorDataset(test_image_array , test_image_label)

train_Loader = DataLoader(train_dataset , batch_size = 64 , shuffle = True)
test_Loader = DataLoader(test_dataset , batch_size = 64 , shuffle = True)
```

2) Convolutional Neural Network

- CNN은 Computer Vision 분야에서 image pattern recognition분야에서 많이 활용되는 알고리즘으로 기본적으로 Convolution layer, Pooling layer, Fully Connected Layer, Softmax 로 이루어진 구조이다. 기본 CNN을 기반으로 다양한 알고리즘들이 연구되어지고 있고, 이번 논문에서는 기본적인 CNN을 구조를 사용한다.

- 구현하고자 하는 구조는 2개의 Convolution layer를 이루고 있고, 각 layer는 [Convolution + ReLu + MaxPooling]로 구성되어 있다. 또한 2개의 layer를 통해 생성된 Feature Map에 연결된 Fully Connected layer(1)와 이것을 Classification을 위해 7개의 output을 생성하는 Fully Connected layer(2)로 이루어진다.

```
class CNN_Classifier(torch.nn.Module):

    def __init__(self):
        super(CNN_Classifier , self ).__init__()

        self.drop_out_prob = 0.4

        self.layer1 = torch.nn.Sequential(torch.nn.Conv2d(1 , 32 , kernel_size = 5 , stride = 1 , padding = 0),
                                          torch.nn.ReLU(),
                                          torch.nn.MaxPool2d(kernel_size = 2 , stride = 2))
        # 44 * 44
        # 22 * 22

        self.layer2 = torch.nn.Sequential(torch.nn.Conv2d(32 , 64 , kernel_size = 5 , stride = 1 , padding = 0),
                                          torch.nn.ReLU(),
                                          torch.nn.MaxPool2d(kernel_size = 2 , stride = 2))

        # 18 * 18
        # 9 * 9

        self.fc1 = torch.nn.Linear(9*9*64 ,3072 , bias = True)

        torch.nn.init.xavier_uniform_(self.fc1.weight)

        self.layer3 = torch.nn.Sequential(self.fc1,
                                          torch.nn.Dropout(p = self.drop_out_prob))

        self.fc2 = torch.nn.Linear(3072 , 7 , bias = True)
        torch.nn.init.xavier_uniform_(self.fc2.weight)

    def forward(self , input_x):

        out = self.layer1(input_x)
        out = self.layer2(out)
        out = out.view(out.size(0) , -1)
        out = self.layer3(out)
        out = self.fc2(out)

        return out
```

3) Support Vector Machine (SVM)

- SVM은 2개의 Class를 분류하기 위해 개발된 방법으로 기본적으로 서로 다른 2개의 Class를 분류하기 위한 최적의 Hyperplane을 찾기 위해 학습을 한다.

- Hyperplane을 찾기 위한 L2-SVM의 목적식은 다음과 같다.

$$\min_{\mathbf{w}} \quad \frac{1}{2}\mathbf{w}^{\mathsf{T}}\mathbf{w} + C\sum_{n=1}^{N}\max(1-\mathbf{w}^{\mathsf{T}}\mathbf{x}_n t_n, 0)^2$$

- 이를 squared Hinge Loss라 부르고, $w^T w$ 는 Euclidean norm(L2-norm), C는 penalty parameter를 뜻한다.

```
class SVM_HingeLoss(torch.nn.Module):
    def __init__(self, p=1, margin=1, weight=None, size_average=True):
        super(SVM_HingeLoss, self).__init__()

        self.p=p
        self.margin=margin
        self.weight=weight
        self.size_average=size_average

    def forward(self, output, y):

        output_y=output[torch.arange(0,y.size()[0]).long().cuda(),y.data.cuda()].view(-1,1)

        loss=output-output_y+self.margin

        loss[torch.arange(0,y.size()[0]).long().cuda(),y.data.cuda()]=0

        loss[loss<0]=0

        if(self.p!=1):
            loss=torch.pow(loss,self.p)

        if(self.weight is not None):
            loss=loss*self.weight

        loss=torch.sum(loss)

        if(self.size_average):
            loss/=output.size()[0]

        return loss
```

Experiments

- CNN + Softmax Train

```
lr = 1e-3
epochs = 50
batch_size = 64

Facial_recognition_model = CNN_Classifier().to(device)

criterion = torch.nn.CrossEntropyLoss().to(device)
optimizer = torch.optim.Adam(Facial_recognition_model.parameters() , lr = lr)

total_batch = len(train_Loader)

for epoch in range(epochs):

    loss_list = list()

    for img , label in train_Loader:

        img = img.to(device)
        label = label.to(device)
        label = label.type(torch.LongTensor).to(device)

        optimizer.zero_grad()
        y_pred = Facial_recognition_model(X)
        loss = criterion(y_pred , Y)
        loss.backward()
        optimizer.step()

        loss_list.append(loss)
```

- 기본적으로 Softmax를 사용하여 Loss를 계산하기 위해서는 pytorch에서 제공하는 torch.nn.CrossEntropyLoss를 사용한다

- CNN + L2-SVM(Hinge Loss) Train

```
lr = 1e-3
epochs = 50
batch_size = 64

Facial_recognition_model_SVM = CNN_Classifier().to(device)

criterion = SVM_HingeLoss().to(device)
optimizer = torch.optim.Adam(Facial_recognition_model_SVM.parameters() , lr = lr)

total_batch = len(train_Loader)

for epoch in range(epochs):

    loss_list = list()

    for img , label in train_Loader:

        img = img.to(device)
        label = label.to(device)
        label = label.type(torch.LongTensor).to(device)

        optimizer.zero_grad()
        y_pred = Facial_recognition_model(X)
        loss = criterion(y_pred , Y)
        loss.backward()
        optimizer.step()

        loss_list.append(loss)
```

- CNN + Softmax Test

```
with torch.no_grad():

    test_loss_list = list()

    correct_CNN_softmax = 0

    for img , label in test_Loader:

        img = img.to(device)
        label = label.to(device)
        label = label.type(torch.LongTensor).to(device)

        y_pred = Facial_recognition_model(img)
        predict_label = torch.argmax(y_pred,1)

        correct_CNN_softmax += (predict_label == label).sum().item()
```

- CNN + SVM Test

```
with torch.no_grad():

    test_loss_list = list()

    correct_CNN_SVM = 0

    for img , label in test_Loader:

        img = img.to(device)
        label = label.to(device)
        label = label.type(torch.LongTensor).to(device)

        y_pred = Facial_recognition_model_SVM(img)
        predict_label = torch.argmax(y_pred,1)

        correct_CNN_SVM += (predict_label == label).sum().item()
```

구현한 모델 성능

- Computing Power로 인해서 50 Epoch의 학습 뒤에 Test data의 정확도를 CNN+Softmax 와 CNN+SVM를 비교하였다.

	CNN+Softmax [acc %]	CNN+SVM [acc %]
Face Expression Dataset	58.84%	59.16%

- CNN+SVM 모델이 조금이지만 좀 더 성능이 좋은 것을 볼 수 있다. 논문에서 Epoch의 횟수 및 파라미터 설정의 언급이 존재하지는 않지만, 더 많은 Epoch과 Parameter optimization을 통한다면 논문에서 보인 정확도에 근접하게 다가갈 수 있을 것으로 보인다.