**Artifact Appendix**

We trained three Convolutional Neural Network models with MNIST and cifar-10 dataset. for run the project, we need to load dataset and build model.

Dataset preparation:

Cifar-10 dataset is downloaded from torchvision.datasets.CIFAR10. The MNIST dataset is loaded from torchvision.datasets.MNIST. The format of the images stored in these two data sets is different, the training results of the model will also have different effects.

Experiment environment:

- google colab (GPU)

- python3.6

- numpy

- pytorch 0.4.0

- torchvision 0.2.0

Optional arguments:

|  |  |  |
| --- | --- | --- |
| argument | Default value | notes |
| lr | 0.001 | Learning rate |
| epoch | 200 | Training times |
| trainBatchSize | 100 | Train batch size |
| testBatchSize | 100 | Test batch size |

Configuration:

Epoch:

200 epochs for each run-through

500 batches for each training epoch

100 batches for each validating epoch

100 images for each training and validating batch

Learning Rate:

0.001 for [1,74] epochs

0.0005 for [75,149] epochs

0.0002.5 for [150,200) epochs

Run experiment step:

-python3 Alexnet.py

-python3 VGG.py

-python3 Resnet.py

We listed the code of building and training Alexnet model. We set the number of classes to 10 and design each convolution layers’ kernel and padding. The sample source code is as follow.

#build Alexmodel

import torch.nn as nn

NUM\_CLASSES = 10

class AlexNet(nn.Module):

def \_\_init\_\_(self, num\_classes=NUM\_CLASSES):

super(AlexNet, self).\_\_init\_\_()

self.features = nn.Sequential(

nn.Conv2d(3, 64, kernel\_size=3, stride=2, padding=1),

nn.ReLU(inplace=True),

nn.MaxPool2d(kernel\_size=2),

nn.Conv2d(64, 192, kernel\_size=3, padding=1),

nn.ReLU(inplace=True),

nn.MaxPool2d(kernel\_size=2),

nn.Conv2d(192, 384, kernel\_size=3, padding=1),

nn.ReLU(inplace=True),

nn.Conv2d(384, 256, kernel\_size=3, padding=1),

nn.ReLU(inplace=True),

nn.Conv2d(256, 256, kernel\_size=3, padding=1),

nn.ReLU(inplace=True),

nn.MaxPool2d(kernel\_size=2),

)

self.classifier = nn.Sequential(

nn.Dropout(),

nn.Linear(256 \* 2 \* 2, 4096),

nn.ReLU(inplace=True),

nn.Dropout(),

nn.Linear(4096, 4096),

nn.ReLU(inplace=True),

nn.Linear(4096, num\_classes),)

def forward(self, x):

x = self.features(x)

x = x.view(x.size(0), 256 \* 2 \* 2)

x = self.classifier(x)

return x

We defined two arrays Accu[] and Cost[] to store the loss and accuracy. We directly display the training results in the form of the line chart. Setting specified arguments. The default values of learning rate and batch size are 0.001 and 100 respectively. We also use CUDA to train model. The training times is 200.

#train model Alexnet.py

import torch.optim as optim

import torch.utils.data

import torch.backends.cudnn as cudnn

import torchvision

from torchvision import transforms as transforms

import numpy as np

import matplotlib.pyplot as plt

import argparse

from models import \*

from misc import progress\_bar

CLASSES = ('plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck')

Accu = []

Cost = []

def main():

parser = argparse.ArgumentParser(description="cifar-10 with PyTorch")

parser.add\_argument('--lr', default=0.001, type=float, help='learning rate')

parser.add\_argument('--epoch', default=200, type=int, help='number of epochs)

parser.add\_argument('--trainBatchSize', default=100, type=int, help='trbatchsize')

parser.add\_argument('--testBatchSize', default=100, type=int, help='testbatchsize')

parser.add\_argument('--cuda',default=torch.cuda.is\_available(),type=bool, help='')

args = parser.parse\_args()

solver = Solver(args)

solver.run()

class Solver(object):

def \_\_init\_\_(self, config):

self.model = None

self.lr = config.lr

self.epochs = config.epoch

self.train\_batch\_size = config.trainBatchSize

self.test\_batch\_size = config.testBatchSize

self.criterion = None

self.optimizer = None

self.scheduler = None

self.device = None

self.cuda = config.cuda

self.train\_loader = None

self.test\_loader = None

def load\_data(self):

train\_transform = transforms.Compose([transforms.RandomHorizontalFlip(), transforms.ToTensor()])

test\_transform = transforms.Compose([transforms.ToTensor()])

train\_set=torchvision.datasets.CIFAR10(root='./data',train=True,download =True, transform=train\_transform)

self.train\_loader=torch.utils.data.DataLoader(dataset=train\_set, batch\_size=self.train\_batch\_size, shuffle=True)

test\_set=torchvision.datasets.CIFAR10(root='./data',train=False, download=True, transform=test\_transform)

self.test\_loader = torch.utils.data.DataLoader(dataset=test\_set, batch\_size=self.test\_batch\_size, shuffle=False)

def load\_model(self):

if self.cuda:

self.device = torch.device('cuda')

cudnn.benchmark = True

else:

self.device = torch.device('cpu')

self.model = AlexNet(num\_classes=10).to(self.device)

self.optimizer = optim.Adam(self.model.parameters(), lr=self.lr)

self.scheduler = optim.lr\_scheduler.MultiStepLR(self.optimizer,milestones =[75, 150], gamma=0.5)

self.criterion = nn.CrossEntropyLoss().to(self.device)

def train(self):

print("train:")

self.model.train()

train\_loss = 0

train\_correct = 0

total = 0

for batch\_num, (data, target) in enumerate(self.train\_loader):

data, target = data.to(self.device), target.to(self.device)

self.optimizer.zero\_grad()

output = self.model(data)

loss = self.criterion(output, target)

loss.backward()

self.optimizer.step()

train\_loss += loss.item()

# second param "1" represents the dimension to be reduced

prediction = torch.max(output, 1)

total += target.size(0)

# train\_correct incremented by one if predicted right

train\_correct+=np.sum(prediction[1].cpu().numpy()==target.cpu()

.numpy())

progress\_bar(batch\_num,len(self.train\_loader),'Loss:%.4f | Acc: %.3f%% (%d/%d)' % (train\_loss / (batch\_num + 1), 100. \* train\_correct / total, train\_correct, total))

Cost.append(train\_loss / (batch\_num + 1))

Accu.append(train\_correct / total)

return train\_loss, train\_correct / total

def test(self):

print("test:")

self.model.eval()

test\_loss = 0

test\_correct = 0

total = 0

with torch.no\_grad():

for batch\_num, (data, target) in enumerate(self.test\_loader):

data, target = data.to(self.device), target.to(self.device)

output = self.model(data)

loss = self.criterion(output, target)

test\_loss += loss.item()

prediction = torch.max(output, 1)

total += target.size(0)

test\_correct+=np.sum(prediction[1].cpu().numpy()==target.cpu()

.numpy())

progress\_bar(batch\_num,len(self.test\_loader),'Loss:%.4f|Acc: %.

3f%% (%d/%d)' % (test\_loss / (batch\_num + 1), 100. \* test\_correct

/ total, test\_correct, total))

return test\_loss, test\_correct / total

def save(self):

model\_out\_path = "model.pth"

torch.save(self.model, model\_out\_path)

print("Checkpoint saved to {}".format(model\_out\_path))

def run(self):

self.load\_data()

self.load\_model()

accuracy = 0

for epoch in range(1, self.epochs + 1):

self.scheduler.step(epoch)

print("\n===> epoch: %d/200" % epoch)

train\_result = self.train()

print(train\_result)

test\_result = self.test()

accuracy = max(accuracy, test\_result[1])

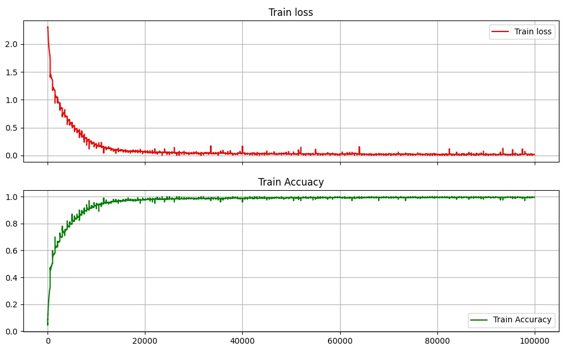
if epoch == self.epochs:

print("===> BEST ACC.%.3f%%" % (accuracy \* 100)) self.save()

if \_\_name\_\_ == '\_\_main\_\_':

main()

The Alexnet model training result is shown as.



The all models’ train result is displayed as.

|  |  |  |
| --- | --- | --- |
| Model | Dataset | Accuracy |
| Alexnet | Cifar10 | 74.74% |
| MNIST | 97.82% |
| Resnet | Cifar10 | 79.46% |
| MINST | 93.35% |
| VGGNet | Cifar10 | 89.79% |
| MINST | 90.62% |