Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Теоретическая информатика и компьютерные технологии»

Домашняя работа №5

по курсу «Теория искусственных нейронных сетей»

«Свёрточные нейронные сети»

Студент группы ИУ9-71Б Афанасьев И.

Преподаватель Каганов Ю.Т.

Москва 2024

1 Цель работы

1. Изучение основных архитектур свёрточных нейронных сетей: LeNet-5, VGG-16, ResNet-34.

2. Обучение свёрточных сетей с использованием различных оптимизаторов и фреймворка PyTorch.

2 Реализация

В листинге 1 приводится исходный код программы на языке Python с использованием фреймворка PyTorch.

Листинг 1: Файл hw5.py

```
import datetime
import torch
import torch.nn as nn
import torch.optim as optim

from torchvision import datasets, transforms, models

DATA_PATH = '../../datasets/'
BATCH_SIZE = 100
MOMENTUM = 0.9
EPOCHS = 20


class LeNet5(nn.Module):
    def __init__(self, num_classes):
        super(LeNet5, self).__init__()
        self.layer1 = nn.Sequential(
            nn.Conv2d(1, 6, kernel_size=5, stride=1, padding=0),
            nn.BatchNorm2d(6),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2))
        self.layer2 = nn.Sequential(
            nn.Conv2d(6, 16, kernel_size=5, stride=1, padding=0),
            nn.BatchNorm2d(16),
```

```
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2))
        self.fc = nn.Linear(400, 120)
        self.relu = nn.ReLU()
        self.fc1 = nn.Linear(120, 84)
        self.relu1 = nn.ReLU()
        self.fc2 = nn.Linear(84, num_classes)

    def forward(self, x):
        out = self.layer1(x)
        out = self.layer2(out)
        out = out.reshape(out.size(0), -1)
        out = self.fc(out)
        out = self.relu(out)
        out = self.fc1(out)
        out = self.relu1(out)
        out = self.fc2(out)
        return out


def train(n_epochs, optimizer, model, loss_fn, train_loader):
    for epoch in range(1, n_epochs + 1):
        loss_train = 0.0
        for imgs, labels in train_loader:
            imgs = imgs.to(device=device)
            labels = labels.to(device=device)
            outputs = model(imgs)
            loss = loss_fn(outputs, labels)

            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            loss_train += loss.item()

        print('{} Epoch {}, Training loss {}'.format(
            datetime.datetime.now(), epoch,
            loss_train / len(train_loader)))
```

```python
def calculate_accuracy(model, train_loader, test_loader):
    accdict = {}
    for name, loader in [("train", train_loader), ("test", test_loader)]:
        correct = 0
        total = 0

        with torch.no_grad():
            for imgs, labels in loader:
                imgs = imgs.to(device=device)
                labels = labels.to(device=device)
                outputs = model(imgs)
                _, predicted = torch.max(outputs, dim=1)
                total += labels.shape[0]
                correct += int((predicted == labels).sum())

        print("Accuracy {}: {:.3f}".format(name, correct / total))
        accdict[name] = correct / total
    return accdict


if __name__ == '__main__':
    device = (torch.device('cuda') if torch.cuda.is_available()
              else torch.device('cpu'))
    print(f'Using {device}')

    mnist_train = datasets.MNIST(
        DATA_PATH, train=True, download=True, transform=transforms.Compose([
            transforms.Resize((32, 32)),
            transforms.ToTensor(),
            transforms.Normalize(mean=(0.1307,), std=(0.3081,))]))
    mnist_test = datasets.MNIST(
        DATA_PATH, train=False, download=True, transform=transforms.Compose([
            transforms.Resize((32, 32)),
            transforms.ToTensor(),
            transforms.Normalize(mean=(0.1325,), std=(0.3105,))]))

    cifar10_train = datasets.CIFAR10(
        DATA_PATH, train=True, download=True, transform=transforms.Compose([
            transforms.ToTensor(),
            transforms.Normalize((0.4915, 0.4823, 0.4468),
```

```
                                (0.2470, 0.2435, 0.2616))
    ]))
cifar10_test = datasets.CIFAR10(
    DATA_PATH, train=False, download=True, transform=transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize((0.4915, 0.4823, 0.4468),
                             (0.2470, 0.2435, 0.2616))
    ]))

loss_fn = nn.CrossEntropyLoss()

print('LeNet5, MNIST')
model = LeNet5(num_classes=10).to(device=device)
train_loader = torch.utils.data.DataLoader(
    mnist_train, batch_size=BATCH_SIZE, shuffle=True)
test_loader = torch.utils.data.DataLoader(
    mnist_test, batch_size=BATCH_SIZE, shuffle=True)

print('SGD')
optimizer = optim.SGD(model.parameters(), lr=1e-2)
train(n_epochs=EPOCHS, optimizer=optimizer, model=model,
      loss_fn=loss_fn, train_loader=train_loader)
calculate_accuracy(model, train_loader, test_loader)

print('Adadelta')
optimizer = optim.Adadelta(model.parameters(), lr=1e-2)
train(n_epochs=EPOCHS, optimizer=optimizer, model=model,
      loss_fn=loss_fn, train_loader=train_loader)
calculate_accuracy(model, train_loader, test_loader)

print('NAG')
optimizer = optim.SGD(model.parameters(), lr=1e-2,
                      momentum=MOMENTUM, nesterov=True)
train(n_epochs=EPOCHS, optimizer=optimizer, model=model,
      loss_fn=loss_fn, train_loader=train_loader)
calculate_accuracy(model, train_loader, test_loader)

print('Adam')
optimizer = optim.Adam(model.parameters(), lr=1e-2)
train(n_epochs=EPOCHS, optimizer=optimizer, model=model,
```

```
        loss_fn=loss_fn, train_loader=train_loader)
    calculate_accuracy(model, train_loader, test_loader)

    print('VGG16, CIFAR10')
    model = models.vgg16(num_classes=10, dropout=0.5).to(device=device)
    train_loader = torch.utils.data.DataLoader(
        cifar10_train, batch_size=BATCH_SIZE, shuffle=True)
    test_loader = torch.utils.data.DataLoader(
        cifar10_test, batch_size=BATCH_SIZE, shuffle=True)

    print('SGD')
    optimizer = optim.SGD(model.parameters(), lr=1e-2)
    train(n_epochs=EPOCHS, optimizer=optimizer, model=model,
        loss_fn=loss_fn, train_loader=train_loader)
    calculate_accuracy(model, train_loader, test_loader)

    print('Adadelta')
    optimizer = optim.Adadelta(model.parameters(), lr=1e-2)
    train(n_epochs=EPOCHS, optimizer=optimizer, model=model,
        loss_fn=loss_fn, train_loader=train_loader)
    calculate_accuracy(model, train_loader, test_loader)

    print('NAG')
    optimizer = optim.SGD(model.parameters(), lr=1e-2,
                          momentum=MOMENTUM, nesterov=True)
    train(n_epochs=EPOCHS, optimizer=optimizer, model=model,
        loss_fn=loss_fn, train_loader=train_loader)
    calculate_accuracy(model, train_loader, test_loader)

    print('Adam')
    optimizer = optim.Adam(model.parameters(), lr=1e-2)
    train(n_epochs=EPOCHS, optimizer=optimizer, model=model,
        loss_fn=loss_fn, train_loader=train_loader)
    calculate_accuracy(model, train_loader, test_loader)

    print('ResNet34, CIFAR10')
    model = models.resnet34(num_classes=10).to(device)

    print('SGD')
    optimizer = optim.SGD(model.parameters(), lr=1e-2)
```

```
    train(n_epochs=EPOCHS, optimizer=optimizer, model=model,
          loss_fn=loss_fn, train_loader=train_loader)
    calculate_accuracy(model, train_loader, test_loader)

    print('Adadelta')
    optimizer = optim.Adadelta(model.parameters(), lr=1e-2)
    train(n_epochs=EPOCHS, optimizer=optimizer, model=model,
          loss_fn=loss_fn, train_loader=train_loader)
    calculate_accuracy(model, train_loader, test_loader)

    print('NAG')
    optimizer = optim.SGD(model.parameters(), lr=1e-2,
                          momentum=MOMENTUM, nesterov=True)
    train(n_epochs=EPOCHS, optimizer=optimizer, model=model,
          loss_fn=loss_fn, train_loader=train_loader)
    calculate_accuracy(model, train_loader, test_loader)

    print('Adam')
    optimizer = optim.Adam(model.parameters(), lr=1e-2)
    train(n_epochs=EPOCHS, optimizer=optimizer, model=model,
          loss_fn=loss_fn, train_loader=train_loader)
    calculate_accuracy(model, train_loader, test_loader)
```

3 Результаты сравнения

3.1 Архитектура LeNet-5

Для обучения нейронной сети с архитектурой LeNet-5 используется датасет MNIST. В таблице 1 приводятся результаты обучения. Точность измеряется на тестовых данных

3.2 Архитектура VGG-16

Для обучения нейронной сети с архитектурой VGG-16 используется датасет CIFAR-10. В таблице 2 приводятся результаты обучения.

Оптимизатор	Эпохи	Коэффициент обучения	Точность, %
SGD	20	10^{-2}	98.7
AdaDelta	20	10^{-2}	98.8
NAD	20	10^{-2}	99.1
Adam	20	10^{-2}	98.9

Таблица 1: Вариация гиперпараметров LeNet-5.

Оптимизатор	Эпохи	Коэффициент обучения	Dropout	Точность, %
SGD	20	10^{-2}	0.5	73.9
AdaDelta	20	10^{-2}	0.5	75.9
NAD	20	10^{-2}	0.5	80.9
Adam	20	10^{-2}	0.5	10.0

Таблица 2: Вариация гиперпараметров VGG-16.

По таблице 2 видно, что с использованием оптимизатора Adam достигается очень низкая точность — идёт застревание в локальном минимуме. Изменением гиперпараметров (количество эпох, коэффициент обучения, dropout) решить проблему не удаётся.

3.3 Архитектура ResNet-34

Для обучения нейронной сети с архитектурой ResNet-34 используется датасет CIFAR-10. В таблице 3 приводятся результаты обучения.

Оптимизатор	Эпохи	Коэффициент обучения	Dropout	Точность, %
SGD	40	10^{-2}	0.5	65.4
AdaDelta	40	10^{-2}	0.5	66.3
NAD	40	10^{-2}	0.5	75.2
Adam	20	10^{-2}	0.5	74.8

Таблица 3: Вариация гиперпараметров ResNet-34.