Сверточные нейронные сети

А.С. Казимиров

Атаки на нейронные сети

# Логистическая регрессия

Реализуйте в PyTorch простую нейронную сеть для классификации цифр из набора данных MNIST.

1. Подключите необходимые библиотеки:

# install dependencies & import libs

!pip3 install torch torchvision

import torch

# Наличие cuda определяется так

DEVICE = 'cuda' if torch.cuda.is\_available() else 'cpu'

print('Using', DEVICE)

import torchvision.datasets as datasets

import torchvision.transforms as transforms

1. Загрузите данные и проведите предобработку

# Выбираем аугментации

train\_trans = transforms.Compose([

    transforms.RandomCrop(28, padding=4),

    transforms.ToTensor(),

    transforms.Normalize((0.1307,), (0.3081,))

])

test\_trans = transforms.Compose([

    transforms.Resize(28),

    transforms.ToTensor(),

    transforms.Normalize((0.1307,), (0.3081,))

])

# Загружаем данные

dataset = datasets.MNIST(root='drive/temp/', download=True,

                         transform=train\_trans)

test\_data = datasets.MNIST(root='drive/temp', download=True,

                           train=False, transform=test\_trans)

1. Визуализируйте примеры данных

import matplotlib.pyplot as plt

import random

plt.figure(figsize=(20,10))

for i in range(10):

    r\_ind = random.randint(0, len(dataset)-1)

    plt.subplot(1, 10, i+1)

    plt.title(dataset.train\_labels[r\_ind].numpy())

    plt.imshow(dataset.train\_data[r\_ind])

    plt.axis('off')

plt.show()

1. Создайте генераторы батчей

from torch.utils.data import DataLoader

train\_loader = DataLoader(dataset, batch\_size=256, num\_workers=4, shuffle=True)

test\_loader = DataLoader(test\_data, batch\_size=256, num\_workers=4)

1. Реализуйте классификатор

import torch.nn as nn

class DummyClassifier(nn.Module):

    def \_\_init\_\_(self, num\_classes=10, init\_weights=True):

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

        self.classifier = nn.Linear(28\*28\*1, num\_classes)

        if init\_weights:

            self.\_init\_weights()

    def forward(self, x):

        x = x.view(x.size(0), -1)

        return self.classifier(x)

    def \_init\_weights(self):

        nn.init.normal\_(self.classifier.weight, 0, 1e-3)

        nn.init.constant\_(self.classifier.bias, 0)

linear\_model = DummyClassifier().to(DEVICE)

1. Настройте оптимизатор и обучите модель

import torch.optim as optim

criterion = nn.CrossEntropyLoss()

optimizer = optim.SGD(linear\_model.parameters(), lr=1e-2, momentum=0.8)

from tqdm import tqdm

reporter = tqdm(enumerate(train\_loader, 0), total=len(train\_loader))

# for epoch in range...

for i, data in reporter:

    inputs, targets = data[0].to(DEVICE), data[1].to(DEVICE)

    # Forward pass

    predictions = linear\_model(inputs)

    # Estimate predictions quality

    loss = criterion(predictions, targets)

    # Backward pass

    optimizer.zero\_grad()

    loss.backward()

    optimizer.step()

    if i % 50 == 0:

        cur\_loss = loss.item()

        reporter.set\_description('loss %.2f'%cur\_loss)

1. Проверьте качество модели

reporter2 = tqdm(enumerate(test\_loader), total=len(test\_loader))

total\_examples = len(test\_loader.dataset)

num\_matches = 0

for i, data in reporter2:

    with torch.no\_grad():

        inputs, targets = data[0].to(DEVICE), data[1].to(DEVICE)

        predictions = linear\_model(inputs)

        \_, label\_predictions = predictions.max(1)

        num\_matches += label\_predictions.eq(targets).sum().item()

print('\nTest accuracy: %.2f%%'%(100.\*num\_matches/total\_examples))

# Свёрточная нейронная сеть

1. Реализуйте сеть LeNet

class LeNet3(nn.Module):

    def \_\_init\_\_(self, num\_classes=10, init\_weights=True):

        super(LeNet3, self).\_\_init\_\_()

        self.conv1 = nn.Conv2d(in\_channels=1, out\_channels=10, kernel\_size=8)

        self.pool1 = nn.AvgPool2d(kernel\_size=2, stride=2)

        self.relu = nn.ReLU()

        self.clf1 = nn.Linear(10\*10\*10, 128)

        self.clf2 = nn.Linear(128, 10)

        if init\_weights:

            self.\_init\_weights()

    def forward(self, x):

        # возможно, тут что-то не так...

        x = self.relu(self.conv1(x))

        x = self.relu(self.pool1(x))

        x = x.view(x.size(0), -1)

        x = self.relu(self.clf1(x))

        x = self.clf2(x)

        return x

    def \_init\_weights(self):

        for m in self.modules():

            if isinstance(m, nn.Linear):

                nn.init.normal\_(m.weight, 0, 1e-2)

                nn.init.constant\_(m.bias, 0)

            elif isinstance(m, nn.Conv2d):

                nn.init.kaiming\_uniform\_(m.weight, nonlinearity='relu')

                nn.init.constant\_(m.bias, 0)

conv\_model = LeNet3().to(DEVICE)

1. Обучите модель

optimizer = optim.SGD(conv\_model.parameters(), lr=1e-2, momentum=0.8)

from tqdm import tqdm

reporter = tqdm(enumerate(train\_loader, 0), total=len(train\_loader))

for i, data in reporter:

    inputs, targets = data[0].to(DEVICE), data[1].to(DEVICE)

    # Forward pass

    predictions = conv\_model(inputs)

    # Estimate predictions quality

    loss = criterion(predictions, targets)

    # Backward pass

    optimizer.zero\_grad()

    loss.backward()

    optimizer.step()

    if i % 50 == 0:

        cur\_loss = loss.item()

        reporter.set\_description('loss %.2f'%cur\_loss)

1. Проверьте качество модели

reporter2 = tqdm(enumerate(test\_loader), total=len(test\_loader))

total\_examples = len(test\_loader.dataset)

num\_matches = 0

for i, data in reporter2:

    with torch.no\_grad():

        inputs, targets = data[0].to(DEVICE), data[1].to(DEVICE)

        predictions = conv\_model(inputs)

        \_, label\_predictions = predictions.max(1)

        num\_matches += label\_predictions.eq(targets).sum().item()

print('\nTest accuracy: %.2f%%'%(100.\*num\_matches/total\_examples))

# Задание

Распишите, как меняются: размеры картинок, receptive field при прохождении по слоям LeNet3. С помощью этого исправьте размерность на входе классификатора.

Сколько всего параметров у LeNet3? Сравните с количеством параметров у линейной модели.

Насколько важны регуляризация и инициализация параметров? Попробуйте с/без dropout/инициализации, сравните результаты.

Напишите свою модель, использующую параметров не больше, чем логистическая регрессия, при этом имеющую качество не хуже, чем LeNet3 на тесте (можно обернуть обучение и тест модели в отдельные методы train и evaluate).