FGSM-атака

А.С. Казимиров

Атаки на нейронные сети

# Сверточная нейронная сеть для CIFAR10

Реализуйте в PyTorch сверточную нейронную сеть для классификации изображений из набора данных CIFAR10.

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)

1. Загрузите данные и проведите предобработку

import torchvision.datasets as datasets

import torchvision.transforms as transforms

# Выбираем аугментации

train\_trans = transforms.Compose([

    transforms.RandomCrop(32, padding=4),

    transforms.RandomHorizontalFlip(),

    transforms.ToTensor(),

    transforms.Normalize((0.491, 0.482, 0.447), (0.247, 0.243, 0.262))

])

test\_trans = transforms.Compose([

    transforms.Resize(32),

    transforms.ToTensor(),

    transforms.Normalize((0.491, 0.482, 0.447), (0.247, 0.243, 0.262))

])

# Загружаем данные

dataset = datasets.CIFAR10(root='drive/temp/', download=True, transform=train\_trans)

test\_data = datasets.CIFAR10(root='drive/temp', download=True, train=False, transform=test\_trans)

1. Визуализируйте примеры данных

ind\_2\_name = {0 : 'airplane',

1 : 'automobile',

2 : 'bird',

3 : 'cat',

4 : 'deer',

5 : 'dog',

6 : 'frog',

7 : 'horse',

8 : 'ship',

9 : 'truck'}

import matplotlib.pyplot as plt

import random

def show\_examples(labels, images):

    assert len(labels) == len(images)

    plt.figure(figsize=(20,10))

    for i in range(len(labels)):

        cur\_class = ind\_2\_name[labels[i]]

        plt.subplot(1, 10, i+1)

        plt.title(cur\_class)

        plt.imshow(images[i])

        plt.axis('off')

    plt.show()

indices = [random.randint(0, len(dataset)-1) for i in range(10)]

labels = [dataset.train\_labels[r\_ind] for r\_ind in indices]

images = [dataset.train\_data[r\_ind] for r\_ind in indices]

show\_examples(labels, images)

1. Создайте генераторы батчей

from torch.utils.data import DataLoader

train\_loader = DataLoader(dataset, batch\_size=64, shuffle=True, num\_workers=4)

test\_loader = DataLoader(test\_data, batch\_size=64, num\_workers=4)

inverse\_transform = transforms.Compose([

    transforms.Normalize((-0.491/0.247, -0.482/0.243, -0.447/0.262),

                         (1/0.247, 1/0.243, 1/0.262)),

    transforms.ToPILImage(),

])

images = [

    inverse\_transform(train\_loader.dataset[r\_ind][0]) for r\_ind in indices

]

show\_examples(labels, images)

1. Реализуйте классификатор

import torch.nn as nn

class CIFAR\_NiN(nn.Module):

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

        super(CIFAR\_NiN, self).\_\_init\_\_()

        self.cfg = [[192, 160, 96],

                    'M', [192, 192, 192],

                    'M', [192, 192, num\_classes]]

        self.features = self.\_make\_layers()

        self.final\_pool = nn.AvgPool2d(kernel\_size=10)

        if init\_weights:

            self.\_init\_weights()

    def \_make\_layers(self):

        layers = []

        in\_channels = 3

        for group in self.cfg:

            if group == 'M':

                layers += [nn.MaxPool2d(kernel\_size=3, stride=2, padding=1)]

            else:

                is\_first = True

                for v in group:

                    if is\_first:

                        conv = nn.Conv2d(

                            in\_channels, v, kernel\_size=4, padding=2

                        )

                    else:

                        conv = nn.Conv2d(

                            in\_channels, v, kernel\_size=1

                        )

                    layers += [conv, nn.BatchNorm2d(v), nn.ReLU(inplace=True)]

                    in\_channels = v

        return nn.Sequential(\*layers)

    def forward(self, x):

        x = self.features(x)

        x = self.final\_pool(x)

        x = x.view(x.size(0), -1)

        return x

    def \_init\_weights(self):

        for m in self.modules():

            if isinstance(m, nn.Conv2d):

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

                if m.bias is not None:

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

            elif isinstance(m, nn.BatchNorm2d):

                nn.init.constant\_(m.weight, 1)

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

            elif isinstance(m, nn.Linear):

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

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

nin\_model = CIFAR\_NiN().to(DEVICE)

1. Настройте оптимизатор и обучите модель

import torch.optim as optim

criterion = nn.CrossEntropyLoss()

optimizer = optim.Adam(nin\_model.parameters(), lr=1e-3)

from tqdm import tqdm

for epoch in range(5):

    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 = nin\_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 = nin\_model(inputs)

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

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

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

# Атака на обученную сеть

1. Найдите правильно классифицированное изображение

r\_ind = 1110

ex\_tensor = test\_loader.dataset[r\_ind][0]

ex\_class = ind\_2\_name[test\_loader.dataset[r\_ind][1]]

ex\_img = inverse\_transform(ex\_tensor)

plt.figure(figsize=(5,5))

plt.title('True class: ' + ex\_class)

plt.imshow(ex\_img)

plt.axis('off')

plt.show()

1. Реализуйте атаку

import torch.nn.functional as F

#true\_output = torch.tensor([test\_loader.dataset[r\_ind][1]]).to(DEVICE)

ex\_input.requires\_grad = True

ex\_output = nin\_model(ex\_input)

loss = F.cross\_entropy(ex\_output, true\_output)

g = torch.autograd.grad(loss, ex\_input, allow\_unused=True)

eps = 1e-3

# FGSM is here:

# new\_input = ex\_input + eps \* torch.sign(g[0])

new\_input = ex\_input - eps \* torch.sign(g[0])

new\_output = nin\_model(new\_input)

1. Выведите результат

new\_prediction = new\_output.max(1)[1].item()

new\_class = ind\_2\_name[new\_prediction]

ex\_img = inverse\_transform(new\_input[0].detach().cpu())

plt.figure(figsize=(5,5))

plt.title('Adversarial class: ' + new\_class)

plt.imshow(ex\_img)

plt.axis('off')

plt.show()