Лабораторная работа № 7

Автоассоциативные сети с узким горлом

Воронов К.М., М8О-407Б-19

Цель работы: исследование свойств автоассоциативных сетей с узким горлом, алгоритмов обучения, а также применение сетей для выполнения линейного и нелинейного анализа главных компонент набора данных.

Вариант 19

[ ]:
```
import matplotlib.pyplot as plt
import numpy as np
import torch
import torch.nn as nn
from torchvision import datasets
from torchvision.transforms import ToTensor
from torch.utils.data import DataLoader
```

[ ]:
```
!pip install matplotlib==3.1.3
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
wheels/public/simple/
Requirement already satisfied: matplotlib==3.1.3 in
/usr/local/lib/python3.8/dist-packages (3.1.3)
Requirement already satisfied: numpy>=1.11 in /usr/local/lib/python3.8/dist-
packages (from matplotlib==3.1.3) (1.21.6)
Requirement already satisfied: kiwisolver>=1.0.1 in
/usr/local/lib/python3.8/dist-packages (from matplotlib==3.1.3) (1.4.4)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.8/dist-
packages (from matplotlib==3.1.3) (0.11.0)
Requirement already satisfied: python-dateutil>=2.1 in
/usr/local/lib/python3.8/dist-packages (from matplotlib==3.1.3) (2.8.2)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in
/usr/local/lib/python3.8/dist-packages (from matplotlib==3.1.3) (3.0.9)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.8/dist-
packages (from python-dateutil>=2.1->matplotlib==3.1.3) (1.15.0)
```

[ ]:
```
device = "cuda" if torch.cuda.is_available() else "cpu"
print(device)
```

```
cuda
```

[ ]:
```
data = datasets.CIFAR10(
    root = "data", train = False, download = True, transform = ToTensor()
)
```

```
Files already downloaded and verified
```

[ ]:
```
class_type = 9
```

[ ]:
```
images = []

for img, label in data:
    if label == class_type:
        images.append(img.numpy().transpose(1, 2, 0))
images = np.array(images)
np.random.shuffle(images)
```

[ ]:
```
number_images = 10

figure, ax = plt.subplots(1, number_images, figsize=(3 * number_images, 4))
for i in range(number_images):
    ax[i].axis("off")
    ax[i].imshow((images[i]).astype(np.float64))

plt.show()
```

[ ]:
```
images_tensor = []

for i in range(len(images)):
    images[i] = images[i] * 2 - 1
    images_tensor.append(torch.Tensor(images[i].flatten()))
    images_tensor[i] = images_tensor[i].to(device)
```

[ ]:
```
encoder = nn.Sequential(
    nn.Linear(3072, 1024),
    nn.Tanh(),
    nn.Linear(1024, 512),
    nn.Tanh(),
    nn.Linear(512, 128),
    nn.Tanh(),
).cuda().to(device)

decoder = nn.Sequential(
    nn.Linear(128, 512),
    nn.Tanh(),
    nn.Linear(512, 1024),
    nn.Tanh(),
    nn.Linear(1024, 3072),
    nn.Tanh(),
).cuda().to(device)
```

[ ]:
```
lr = 0.001
epochs = 1050
batch_size = 32
optim_enc = torch.optim.Adam(encoder.parameters(), lr = lr)
optim_dec = torch.optim.Adam(decoder.parameters(), lr = lr)
```

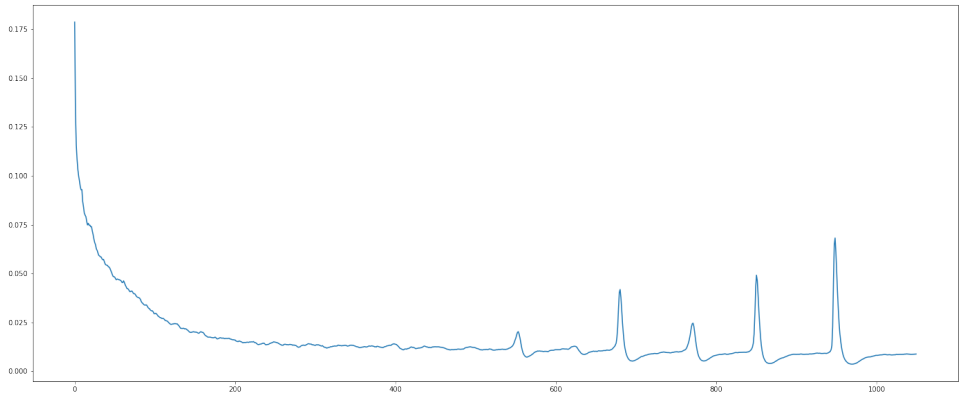
[ ]:
```
dataloader = DataLoader(images_tensor, batch_size = batch_size)
```

[ ]:
```
encoder.train()
decoder.train()
loss_epoch_sum = []
for epoch in range(epochs):
    loss_epoch = 0.0
    for img in dataloader:
        enc = encoder(img)
        dec = decoder(enc)
        loss = torch.nn.MSELoss()(img, dec)
        loss.backward()

        optim_dec.step()
        optim_enc.step()

        optim_dec.zero_grad()
        optim_enc.zero_grad()

        loss_epoch += loss.cpu().detach().item() / len(dataloader)
    loss_epoch_sum.append(loss_epoch)
    print("Loss ", epoch, " epoch: ", loss_epoch_sum[len(loss_epoch_sum) - 1])
```

```
Loss  0  epoch:  0.1787474942393601
Loss  1  epoch:  0.1321283089928329
Loss  2  epoch:  0.11592263518832624
Loss  3  epoch:  0.10827818443067372
Loss  4  epoch:  0.10312776500359178
Loss  5  epoch:  0.0998982188757509
Loss  6  epoch:  0.09718525619246066
Loss  7  epoch:  0.09451376204378903
Loss  8  epoch:  0.09283375577069819
Loss  9  epoch:  0.09293617098592222
Loss  10  epoch:  0.08684870461001992
...
Loss  1039  epoch:  0.008665082772495225
Loss  1040  epoch:  0.008646685251733288
Loss  1041  epoch:  0.008629526244476438
Loss  1042  epoch:  0.008529180806363001
Loss  1043  epoch:  0.008506038575433195
Loss  1044  epoch:  0.008539525108062662
Loss  1045  epoch:  0.008539340400602669
Loss  1046  epoch:  0.008578880559070967
```

```
Loss  1047  epoch:  0.008639386185677722
Loss  1048  epoch:  0.008673707488924265
Loss  1049  epoch:  0.008706651424290612
```

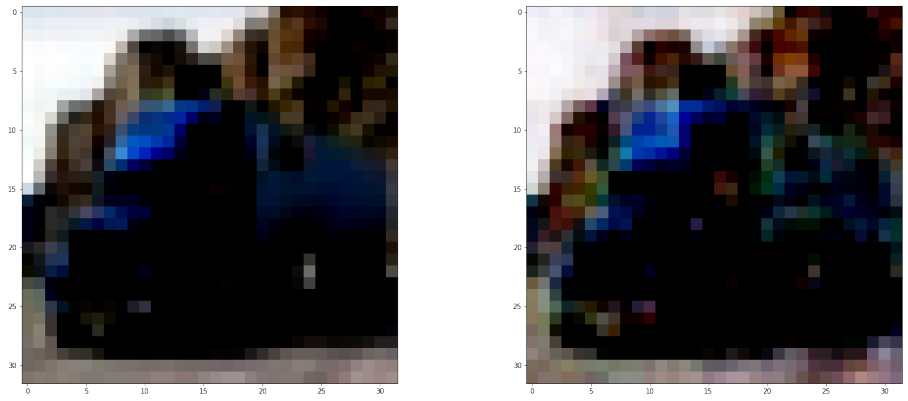
[ ]:
```
figure = plt.figure(figsize = (24, 10))

loss_epoch_sum = np.array(loss_epoch_sum)
print()
tt = np.arange(0, epochs, 1)

plt.plot(tt, loss_epoch_sum)
plt.show()
```



[ ]:
```
encoder.eval()
decoder.eval()

figure = plt.figure(figsize = (24, 10))

ax1 = figure.add_subplot(1, 2, 1)
ax2 = figure.add_subplot(1, 2, 2)


ax1.imshow(images[1].astype(np.float64))

img = images_tensor[1].flatten().to(device)

enc = encoder(img)
dec = decoder(enc)

dec = dec.reshape(32, 32, 3)
ax2.imshow((dec.cpu().detach().numpy()).astype(np.float64))
```

```
plt.show()
```

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with
RGB data ([0..1] for floats or [0..255] for integers).
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with
RGB data ([0..1] for floats or [0..255] for integers).
```



Выводы

Выполнив данную лаюораторную работу, я изучил строение автоассоциативные сетей с узким горлом и реализовал одну из них, продемонстрировав её работу на датасете CIFAR-10

```
[ ]:
```