ВЫСШАЯ ШКОЛА ЭКОНОМИКИ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

Прикладные задачи анализа данных

Домашнее задание: генеративно-состязательные сети

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.m
```

Введение

MAGIC – Major Atmospheric Gamma Imaging Cherenkov Telescope

MAGIC (Major Atmospheric Gamma Imaging Cherenkov) - это система, состоящая из двух черенковских телескопов диаметром 17 м. Они предназначены для наблюдения гамма-ических источников в диапазоне очень высоких

Сохранено

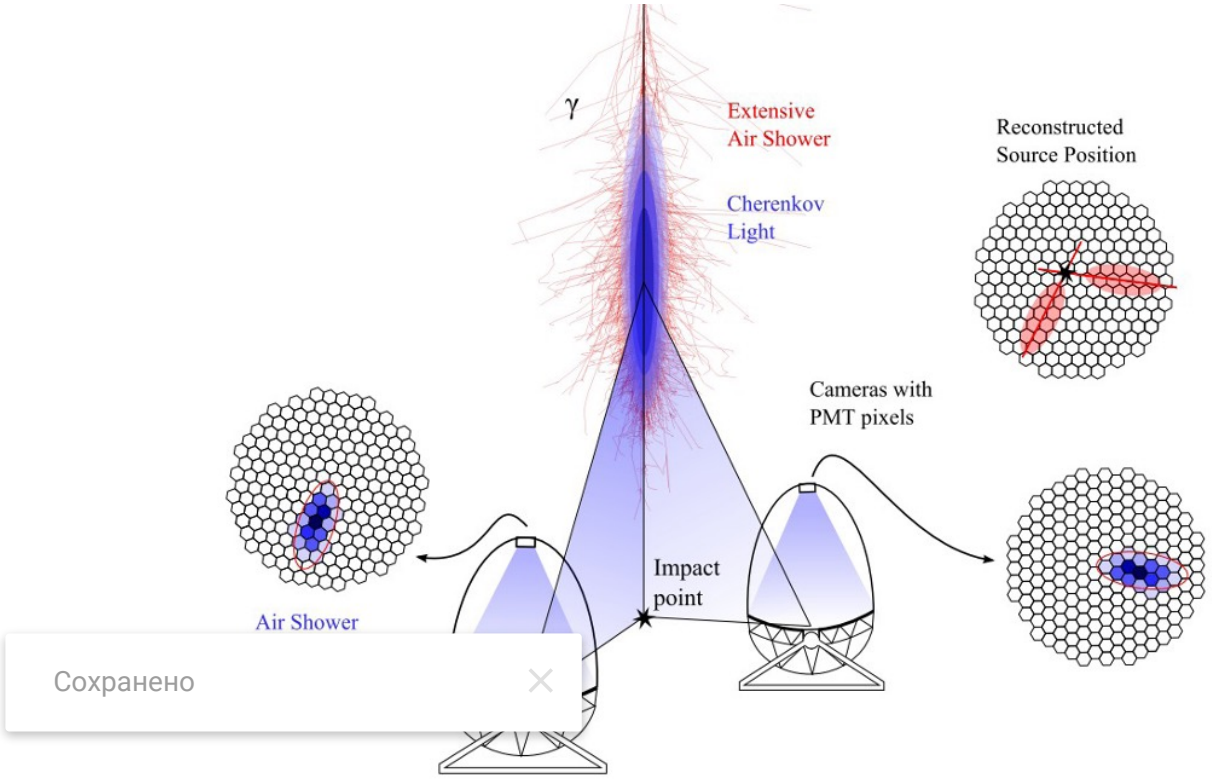
Телескопами MAGIC в настоящее время управляют около 165 астрофизиков из 24 организаций и консорциумов из 12 стран. MAGIC позволил открыть и исследовать новые классы источников гамма-излучения, таких как, например, пульсары и гамма-всплески (GRB).

Источник: https://magic.mpp.mpg.de/

Youtube video: https://youtu.be/mjcDSR2vSU8

Частицы из космоса

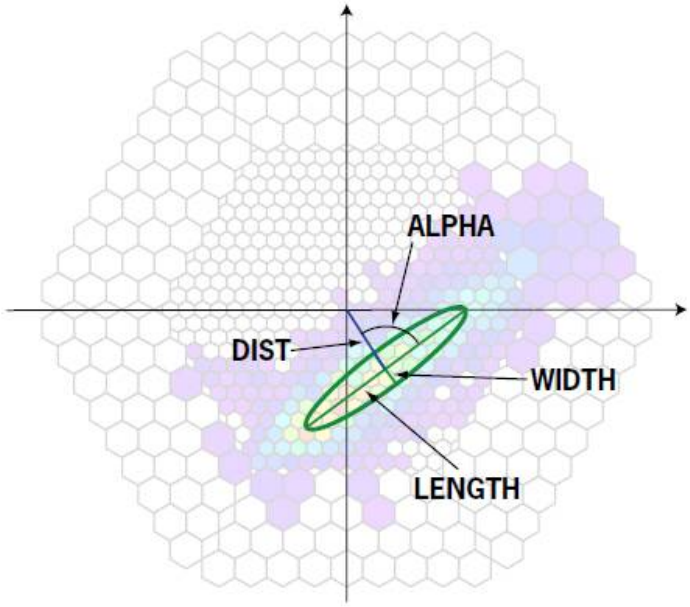
Космические частицы, γ-кванты (фотоны) и адроны (протоны), взаимодействуют с атмосферой и порождают ливни вторичных частиц. Двигаясь с околосветовой скоростью, эти частицы излучают Черенковское излучение. Телескопы фотографируют это излучение. По фотографиям можно определить тип частицы из космоса: фотон или протон.



Сохранено

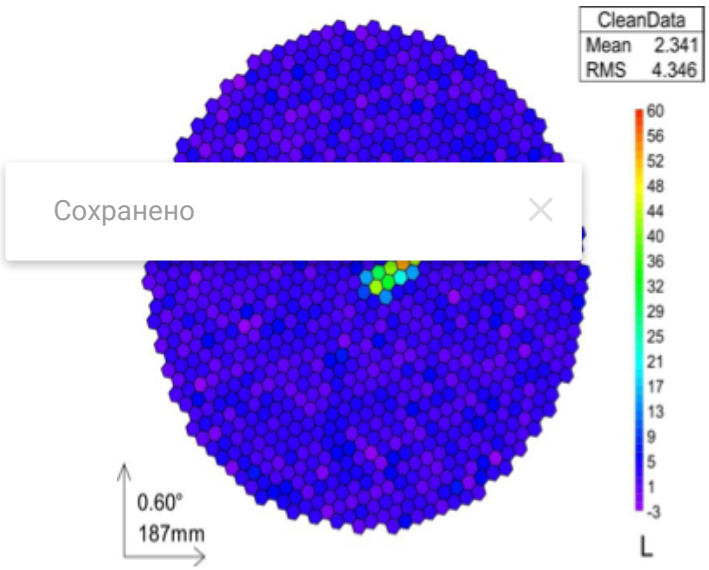
Фотографии

Задача атмосферного черенковского телескопа - получить изображение ливня путем измерения черенковского света от частиц ливня. Это изображение представляет собой геометрическую проекцию ливня на детектор. Для анализа этих изображений были введены параметры изображения или так называемые параметры Хилласа. Есть два вида параметров изображения: параметры формы и параметры ориентации. (Источник: http://ihp-lx.ethz.ch/Stamet/magic/parameters.html)

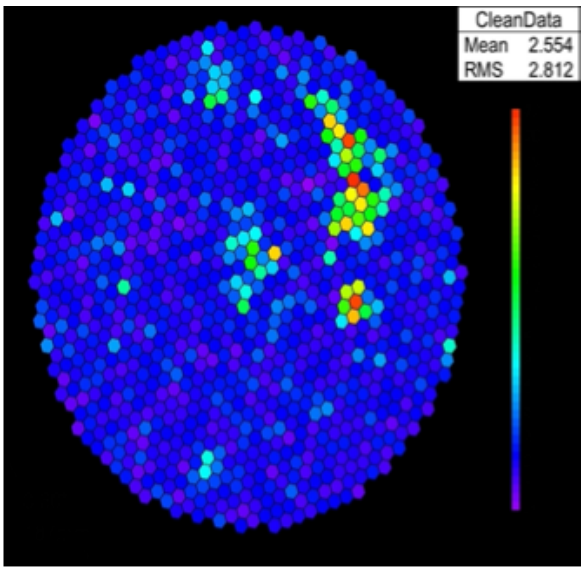


Фотоны vs адронов

Изображения для γ-квантов (фотонов) и адронов (протонов) отличаются по форме кластеров. Астрономы используют модели машинного обучения для классификации этих изображений. Для обучения моделей ученые искусственно генерируют такие изображения для каждого типа частиц с помощью сложных физических симуляторов.



Фотоны



Адроны

Ускорение симуляции

Сложные физические симуляторы требуют больших вычислительных ресурсов. Они моделируют прилет частиц из космоса, их взаимодействие с атмосферой, рождение

ливней, черенковского излучения и работы телескопов для получения изображений. Но мы можем использовать генеративно-состязательные сети для быстрой симуляции!

```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
```

Данные

Будем использовать данные телескопа MAGIC из UCI репозитория https://archive.ics.uci.edu/ml/datasets/MAGIC+Gamma+Telescope. Каждый объект в данных - параметры одного изображения кластера и метка этого кластера (фотон или адрон):

0. Length: major axis of ellipse [mm]
1. Width: minor axis of ellipse [mm]
2. Size: 10-log of sum of content of all pixels [in #phot]
3. Conc: ratio of sum of two highest pixels over fSize [ratio]
4. Conc1: ratio of highest pixel over fSize [ratio]
5. Asym: distance from highest pixel to center, projected onto major axis [mm]
6. M3Long: 3rd root of third moment along major axis [mm]
7. M3Trans: 3rd root of third moment along minor axis [mm]
8. Alpha: angle of major axis with vector to origin [deg]
9. Dist: distance from origin to center of ellipse [mm]
10. ...on (background)


```
# read data
names = np.array(['Length', 'Width', 'Size', 'Conc', 'Conc1', 'Asym', 'M3Long', 'M3Trans',
data = pd.read_csv("/content/drive/My Drive/Учеба/data/magic04.data", header=None)
data.columns = names
data.head()
```

```
data.shape
```

```
(19020, 11)
```

Постановка задачи

Ваша задача заключается в том, чтобы с помощью генеративно-состязательных сетей научится генерировать параметры кластеров на изображениях телекопа для каждого типа частиц (фотона или адрона):

- X - матрица реальных объектов, которые нужно начиться генерировать;
- y - метки классов, которые будем использовать как условие при генерации.

```
# параметры кластеров на изображениях
X = data[names[:-1]].values
X = np.abs(X)

# метки классов
labels = data[names[-1]].values
y = np.ones((len(labels), 1))
y[labels == 'h'] = 0


# примеры
X[:2]
```

```
array([[2.87967e+01, 1.60021e+01, 2.64490e+00, 3.91800e-01, 1.98200e-01,
        2.77004e+01, 2.20110e+01, 8.20270e+00, 4.00920e+01, 8.18828e+01],
       [3.16036e+01, 1.17235e+01, 2.51850e+00, 5.30300e-01, 3.77300e-01,
        2.62722e+01, 2.38238e+01, 9.95740e+00, 6.36090e+00, 2.05261e+02]])
```



```
array([[1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.]])
```

Визуализация данных

Каждое изображение описывается 10 параметрами. Давайте построим распделения значений каждого параметра для каждого типа частиц.

```
def plot_hists(X1, X2, names, label1, label2, bins=np.linspace(-3, 3, 61)):
    plt.figure(figsize=(4*4, 4*2))
    for i in range(X1.shape[1]):
        plt.subplot(3, 4, i+1)
        plt.hist(X1[:, i], bins=bins, alpha=0.5, label=label1, color='C0')
        plt.hist(X2[:, i], bins=bins, alpha=0.5, label=label2, color='C1')
        plt.xlabel(names[i], size=14)
        plt.legend(loc='best')
    plt.tight_layout()


plot_hists(X[y[:, 0]==0], X[y[:, 0]==1], names, label1="Hadrons", label2="Photons", bins=5
```


Предобработка данных

Из графика видим, что распределения для многих признаков имеют тяжелые хвосты. Это делает обучение генеративных моделей тяжелее. Поэтому, нужно как-то преобразовать данные, чтобы убрать эти тяжелые хвосты.

Задание 1 (1 балл)

Используя функцию `sklearn.preprocessing.QuantileTransformer` трансформируйте входные данные `X`. Это преобразование делает так, чтобы распределение каждого параметра было нормальным. Описание функции доступно по ссылке http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.QuantileTransformer.html. Используйте значение параметра `output_distribution='normal'`.

```
### YOUR CODE IS HERE ######
from sklearn.preprocessing import QuantileTransformer
rng = np.random.RandomState(668)
qt = QuantileTransformer(n_quantiles=150, random_state=668, output_distribution='normal')
X_qt = qt.fit_transform(X)
### THE END OF YOUR CODE ###
```

```
plot_hists(X_qt[y[:, 0]==0], X_qt[y[:, 0]==1], names, label1="Hadrons", label2="Photons",
```

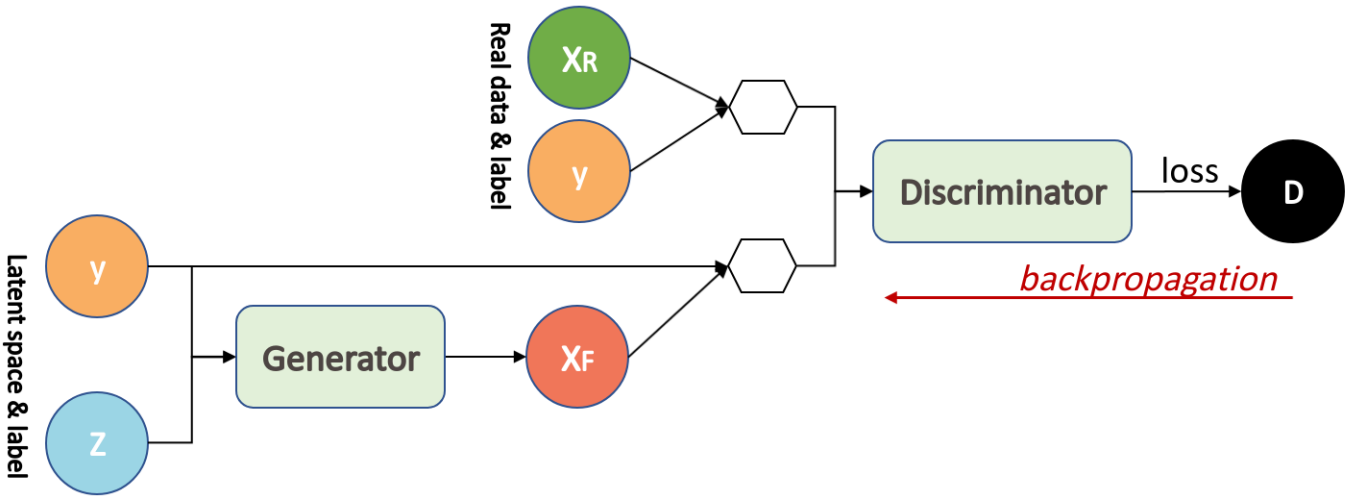
Сохранено

Обучающая и тестовая выборки

```
from sklearn.model_selection import train_test_split

# train / test split
X_train, X_test, y_train, y_test = train_test_split(X_qt, y, stratify=y, test_size=0.5, sh


plot_hists(X_train, X_test, names, label1="Train", label2="Test")
```


Conditional WGAN

Мы будем использовать `Conditional WGAN`, который изображен на рисунке. В качестве условия `y` мы будем использовать метку класса: **0** - адрон, **1** - фотон. Таким образом, мы будем сообщать генератору для какой частицы нужно генерировать параметры изображения.



Генератор $\hat{x} = G(z, y)$ будет принимать на вход шумовой вектор z и вектор условий y, а выдавать будет сгенерированный (фейковый) вектор параметров $\hat{x}$.

Дискриминатор $D(x, y)$ будет принимать на вход вектор параметров x и вектор условий y, а возвращать будет рациональное число.

Обучать `Conditional WGAN` будем с такой функцией потерь:

$$L(G, D) = -\frac{1}{n} \sum_{x_i \in X, y_i \in y} D(x_i, y_i) + \frac{1}{n} \sum_{z_i \in Z, y_i \in y} D(G(z_i, y_i), y_i) \to \max_G \min_D$$

```
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import TensorDataset, DataLoader
from torch.autograd import Variable
                                                    tput
```

Сохранено

```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
```

```
device
```

```
    device(type='cuda', index=0)
```

Задание 2 (1 балл)

Реализуйте нейронную сеть для генератора со следующими слоями:

- Полносвязный слой со 100 нейронами;
- Слой батч-нормализации;
- ReLU функцию активации;
- Полносвязный слой со 100 нейронами;

- Слой батч-нормализации;
- ReLU функцию активации;
- Выходной слой.

Подсказка: используйте функцию `nn.Sequential()`.

```
BATCH_SIZE=32

class Generator(nn.Module):

    def __init__(self, n_inputs, n_outputs):
        super(Generator, self).__init__()

        ### YOUR CODE IS HERE ######
        self.net = nn.Sequential(
            nn.Linear(n_inputs, 100),
            nn.BatchNorm1d(100),
            nn.LeakyReLU(),
            nn.Linear(100, 100),
            nn.BatchNorm1d(100),
            nn.LeakyReLU(),
            nn.Linear(100, n_outputs)

        )
        ### THE END OF YOUR CODE ###

    def forward(self, z, y, num_objects):
        #print(z)
        #print(y)
        y = nn.functional.one_hot(y.to(torch.int64), num_classes=2).reshape(num_objects, 2
        zy = torch.cat((z, y), dim=1).to(device)
        return self.net(zy)
```

Сохранено

Задание 3 (1 балл)

Реализуйте нейронную сеть для дискриминатора со следующими слоями:

- Полносвязный слой со 100 нейронами;
- ReLU функцию активации;
- Полносвязный слой со 100 нейронами;
- ReLU функцию активации;
- Выходной слой.

Подсказка: используйте функцию `nn.Sequential()`.

```
class Discriminator(nn.Module):

    def __init__(self, n_inputs):
```

```
        super(Discriminator, self).__init__()

        ### YOUR CODE IS HERE ######
        self.net = nn.Sequential(
            nn.Linear(n_inputs, 100),
            nn.ReLU(),
            nn.Linear(100, 100),
            nn.ReLU(),
            nn.Linear(100, 1)
        )
        ### THE END OF YOUR CODE ###

    def forward(self, x, y):
        xy = torch.cat((x, y), dim=1).to(device)
        return self.net(xy)
```

Задание 4 (2 балла)

Реализуйте класс для обучения генеративной модели.

- Подсказка 1: не забывайте ограничивать веса дискриминатора. Для этого используйте `p.data.clamp_(-0.01, 0.01)`, где `p` веса дискриминатора.
- Подсказка 2: `n_critic` - число итераций обучения дискриминатора на одну итерацию обучения генератора.
- Подсказка 3: Используйте `X_tensor = torch.tensor(X_numpy, dtype=torch.float, device=DEVICE)` для перевода numpy в тензор.

```
from typing import Optional
from typing_extensions import Literal
```

Сохранено

```
def wgan_loss(real_objects_scores: torch.Tensor,
              generated_objects_scores: torch.Tensor,
              by: Literal['generator', 'discriminator'] = 'generator') -> torch.Tensor:
  '''
  Имплементирует подсчет лосса для генератора и дискриминатора.
  '''
  if by=='generator':
    # возвращает лосс генератора
    return -generated_objects_scores.mean()#YOUR CODE HERE
  elif by == 'discriminator':
    # возвращает лосс дискриминатора
    return  generated_objects_scores.mean() - real_objects_scores.mean()

def train_gan(tr_dataloader,
              gen, discr,
              gen_opt, discr_opt,
              loss_func, prior,
              num_epochs, gen_steps, discr_steps,
```

```
          discr_params_clip_value=None,
          verbose_num_iters=100,
          data_type='2d', latent_dim_= 10):
'''
Имплементирует подсчет лосса для генератора и дискриминатора.
'''
gen.to(device)
discr.to(device)
gen.train()
discr.train()
gen_loss_trace = []
discr_loss_trace = []


iter_i = 0

for epoch_i in tqdm(range(num_epochs)):
    print(f'Epoch {epoch_i + 1}')
    for batch in tr_dataloader:
        # берем реальные объекты
        for p in discr.parameters():
            p.data.clamp_(-0.01, 0.01)


        real_objects, y2 = batch
        real_objects = real_objects.to(device)
        y2 = y2.to(device)

        # генерируем новые объекты
        num_objects = real_objects.shape[0]
        z = torch.randn(num_objects, latent_dim_).to(device)
        gen_objects = gen(z.to(device), y2.to(device), num_objects).cuda()

Сохранено                                                    ×
        Check\n',  gen_objects, torch.cat([real_objects, gen_o
        _objects_scores = torch.split(discr(
            torch.cat([real_objects, gen_objects], dim=0).to(device),
            torch.cat([y2, y2], dim=0).to(device)), num_objects)

        if (iter_i % (gen_steps + discr_steps)) < gen_steps:
          # делаем шаг обучения генератора
            gen_opt.zero_grad()
            gen_loss = loss_func(real_objects_scores, gen_objects_scores, 'generator')
            gen_loss.backward()
            gen_opt.step()
            gen_loss_trace.append((iter_i, gen_loss.item()))
        else:
          # делаем шаг обучения дискриминатора
            discr_opt.zero_grad()
            discr_loss = loss_func(real_objects_scores, gen_objects_scores, 'discrimin
            discr_loss.backward()
            discr_opt.step()
            discr_loss_trace.append((iter_i, discr_loss.item()))
```

```
            iter_i += 1

            # раз в verbose_num_iters визуализируем наши лоссы и семплы
            if iter_i % verbose_num_iters == 0:
                clear_output(wait=True)
                plt.figure(figsize=(10, 10))

                plt.subplot(1, 3, 1)
                plt.xlabel('Iteration')
                plt.ylabel('Generator loss')
                plt.plot([p[0] for p in gen_loss_trace],
                         [p[1] for p in gen_loss_trace])

                plt.subplot(1, 3, 2)
                plt.xlabel('Iteration')
                plt.ylabel('Discriminator loss')
                plt.plot([p[0] for p in discr_loss_trace],
                         [p[1] for p in discr_loss_trace], color='orange')

                gen.eval()


                plt.show()
                gen.train()

    gen.eval()
    discr.eval()
```

Обучение

Обучим модель на данных.

Сохранено

```
=torch.float, device=device)
y_cond = torch.tensor(y_train, dtype=torch.float, device=device)

# tensor to dataset
dataset_real = TensorDataset(X_real, y_cond)
loader = DataLoader(dataset_real, batch_size=BATCH_SIZE, shuffle=True, drop_last=True)



latent_dim = 32
generator     = Generator(n_inputs=latent_dim+y.shape[1] + 1,
                          n_outputs=X_train.shape[1])


discriminator = Discriminator(n_inputs=X_train.shape[1]+y.shape[1])

gen_opt = torch.optim.RMSprop(generator.parameters(), lr=3e-4)
discr_opt = torch.optim.RMSprop(discriminator.parameters(), lr=3e-4)
```

```
prior_2d = torch.distributions.Normal(torch.zeros(2).cuda(), torch.ones(2).cuda()) # not u
train_gan(loader, generator, discriminator, gen_opt, discr_opt,
          wgan_loss, prior_2d, num_epochs=82, gen_steps=1, discr_steps=5,
          verbose_num_iters=100, latent_dim_ = latent_dim)
```


Задание 5 (1 балл)

Реализуйте функцию для генерации новый объектов X по вектору условий y.

```
def generate(generator_, y, latent_dim_):
    ### YOUR CODE IS HERE ######
    z = torch.randn(len(y), latent_dim_).to(device)
    y = torch.tensor(y, dtype=torch.float, device=device)
    X_fake = generator_(z, y, len(y)).cuda()
    ### THE END OF YOUR CODE ###
    return X_fake # numpy
```

Теперь сгенерируем фейковые матрицы `X_fake_train` и `X_fake_test`. Сравним их с матрицами реальных объектов `X_train` и `X_test`

```
X_fake_train = generate(generator, y_train, latent_dim).cpu().detach().numpy()

plot_hists(X_train, X_fake_train , names, label1="Real", label2="Fake", bins=50)
```

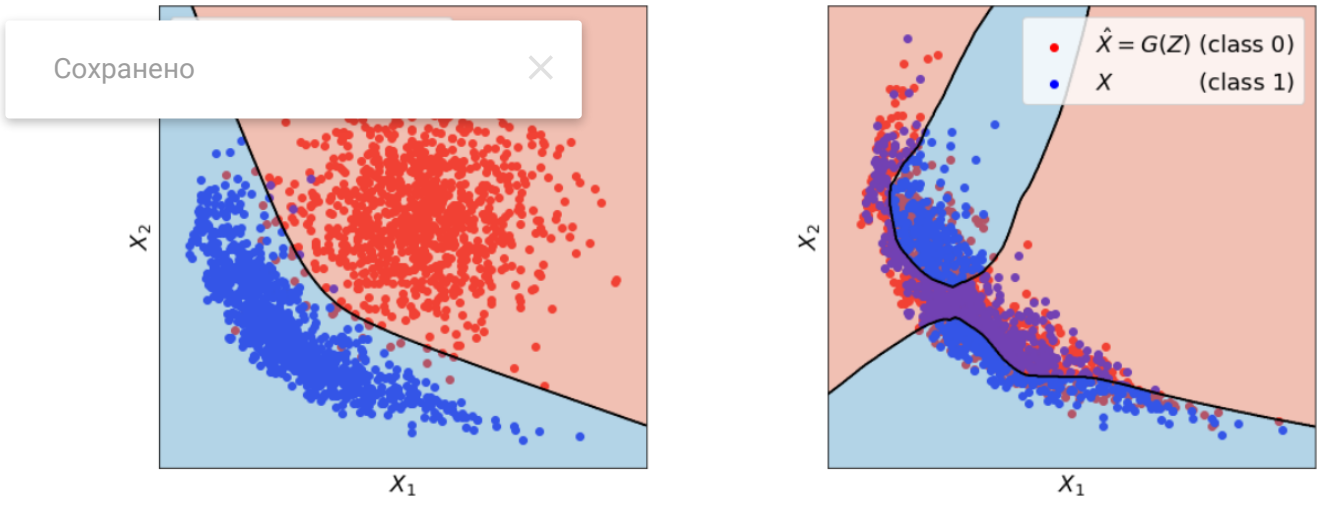


```
X_fake_test = generate(generator, y_test, latent_dim).cpu().detach().numpy()

plot_hists(X_test, X_fake_test, names, label1="Real", label2="Fake", bins=50)
```

Вывод 1:

Визуально мы видим сходство реальных и фейковых данных. Однако это только проекции 10-мерных объектов на одну ось.

Измерение качества генерации



ROC AUC = 0.995
Log Loss = 0.135

ROC AUC = 0.554
Log Loss = 0.689

Измерим сходство распределений классификатором.

```
# собираем реальный и фейковые матрицы в одну
XX_train = np.concatenate((X_fake_train, X_train), axis=0)
XX_test = np.concatenate((X_fake_test, X_test), axis=0)
```

```
yy_train = np.array([0]*len(X_fake_train) + [1]*len(X_train))
yy_test = np.array([0]*len(X_fake_test) + [1]*len(X_test))


from sklearn.ensemble import GradientBoostingClassifier

# обучаем классификатор
clf = GradientBoostingClassifier()
clf.fit(XX_train, yy_train)

# получаем прогнозы
yy_test_proba = clf.predict_proba(XX_test)[:, 1]


from sklearn.metrics import roc_auc_score

auc = roc_auc_score(yy_test, yy_test_proba)
print("ROC AUC = ", auc)
```

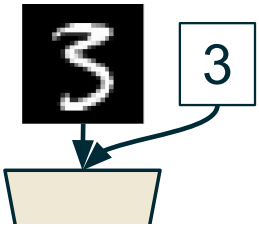
```
ROC AUC =  0.752788541808335
```

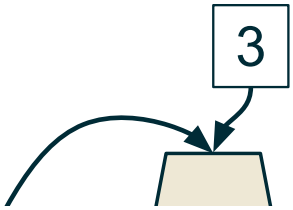
Вывод 2

Идеальное значение ROC AUC равно 0.5. Это соответствует случаю, когда классификатор не может разделить реальные и фейковые данные. В нашем случае ROC AUC около 0.7, что говорит о том, что есть куда улучшать качество генеративной модели :)

Условные вариационные автокодировщики

Сохранено





Задание 6 (1 балл)

Реализуйте нейронную сеть для декодеровщика со следующими слоями:

- Полносвязный слой со 100 нейронами;
- Слой батч-нормализации;
- ReLU функцию активации;
- Полносвязный слой со 100 нейронами;
- Слой батч-нормализации;
- ReLU функцию активации;
- Выходной слой для mu; Выходной слой для log_sigma;

Подсказка: используйте функцию `nn.Sequential()`.

вашим семинаром по теме вариационных автокодировщиков выполните следующие

```
class Encoder(nn.Module):
    def __init__(self, n_inputs, lat_size):
        super(Encoder, self).__init__()

        ### YOUR CODE IS HERE ######
        self.enc_net = nn.Sequential(
            nn.Linear(n_inputs, 100),
            nn.BatchNorm1d(100),
            nn.LeakyReLU(),
```

Сохранено

```
        )

        self.mu = nn.Linear(100, lat_size)
        self.log_sigma =  nn.Linear(100, lat_size)
        ### THE END OF YOUR CODE ###

    def forward(self, x, y):
        z = torch.cat((x, y), dim=1)
        z = self.enc_net(z)
        mu = self.mu(z)
        log_sigma = self.log_sigma(z)
        return mu, log_sigma
```

Задание 7 (1 балл)

Реализуйте нейронную сеть для декодеровщика со следующими слоями:

- Полносвязный слой со 100 нейронами;
- ReLU функцию активации;
- Полносвязный слой со 100 нейронами;
- ReLU функцию активации;
- Выходной слой.

Подсказка: используйте функцию `nn.Sequential()`.

```
class Decoder(nn.Module):
    def __init__(self, n_inputs, n_outputs):
        super(Decoder, self).__init__()

        ### YOUR CODE IS HERE ######
        self.dec_net = nn.Sequential(
            nn.Linear(n_inputs, 100),
            nn.LeakyReLU(),
            nn.Linear(100, 100),
            nn.LeakyReLU(),
            nn.Linear(100, n_outputs)
            )
        ### THE END OF YOUR CODE ###

    def forward(self, z, y):
        z_cond = torch.cat((z, y), dim=1)
        x_rec = self.dec_net(z_cond)
        return x_rec
```

Задание 8 (1 балл)

Сохранено

мационного автокодировщика.

Подсказка: используйте `X_tensor = torch.tensor(X_numpy, dtype=torch.float, device=DEVICE)` для перевода numpy в тензор.

```
DEVICE = device
class VAEFitter(object):

    def __init__(self, encoder, decoder, batch_size=32, n_epochs=10, latent_dim=1, lr=0.00

        self.encoder = encoder
        self.decoder = decoder
        self.batch_size = batch_size
        self.n_epochs = n_epochs
        self.latent_dim = latent_dim
        self.lr = lr
        self.KL_weight = KL_weight

        self.criterion = nn.MSELoss()
        self.opt  = torch.optim.RMSprop(list(self.encoder.parameters()) + list(self.decode
```

```
        self.encoder.to(DEVICE)
        self.decoder.to(DEVICE)

    def encode(self, x: torch.Tensor, y: torch.Tensor) -> torch.Tensor:
        return self.encoder(x, y)

    def decode(self, z: torch.Tensor, y: torch.Tensor) -> torch.Tensor:
        #print(z.shape)
        #print(y.shape)
        #print('Cicly done')
        return self.decoder(z, y)



    def sample_z(self, mu, log_sigma):
        eps = torch.randn(mu.shape).to(DEVICE)
        return mu + torch.exp(log_sigma / 2) * eps

    def custom_loss(self, x, rec_x, mu, log_sigma):
        KL = torch.mean(-0.5 * torch.sum(1 + log_sigma - mu ** 2 - log_sigma.exp(), dim =
        recon_loss = self.criterion(x, rec_x)
        return KL*self.KL_weight + recon_loss


    def compute_loss(self, x_batch, cond_batch):
              ### YOUR CODE IS HERE ######

        mu, log_sigma = self.encode(x_batch, cond_batch)
        z = self.sample_z(mu, log_sigma)
        reconstructed_x = self.decode(z, cond_batch)

        loss = self.custom_loss(x_batch, reconstructed_x, mu, log_sigma)
```



```
        return loss


    def fit(self, X, y):

        # numpy to tensor
        X_real = torch.tensor(X, dtype=torch.float, device=DEVICE)
        y_cond = torch.tensor(y, dtype=torch.float, device=DEVICE)

        # tensor to dataset
        dataset_real = TensorDataset(X_real, y_cond)

        # Turn on training
        self.encoder.train(True)
        self.decoder.train(True)

        self.loss_history = []

        # Fit CVAE
```

```
        for epoch in range(self.n_epochs):
            for i, (x_batch, cond_batch) in enumerate(DataLoader(dataset_real, batch_size=

                # caiculate loss
                loss = self.compute_loss(x_batch, cond_batch)

                # optimization step
                self.opt.zero_grad()
                loss.backward()
                self.opt.step()

            # caiculate and store loss after an epoch
            loss_epoch = self.compute_loss(X_real, y_cond)
            print(f'Train Epoch number {epoch+1}, Train loss = {loss_epoch:.3f}')
            self.loss_history.append(loss_epoch.detach().cpu())


        # Turn off training
        self.encoder.train(False)
        self.decoder.train(False)
clear_output()
```

Обучение

Обучим модель на данных.

```
%%time

latent_dim = 10

encoder = Encoder(n_inputs=X_train.shape[1]+y.shape[1], lat_size=latent_dim)
decoder = Decoder(n_inputs=latent_dim+y.shape[1], n_outputs=X_train.shape[1])

vae_fitter = VAEFitter(encoder, decoder, batch_size=50, n_epochs=230, latent_dim=latent_di
vae_fitter.fit(X_train, y_train)
```

```
Train Epoch number 105, Train loss = 0.045
Train Epoch number 106, Train loss = 0.043
Train Epoch number 107, Train loss = 0.046
Train Epoch number 108, Train loss = 0.049
Train Epoch number 109, Train loss = 0.042
Train Epoch number 110, Train loss = 0.035
Train Epoch number 111, Train loss = 0.036
Train Epoch number 112, Train loss = 0.040
Train Epoch number 113, Train loss = 0.033
Train Epoch number 114, Train loss = 0.037
Train Epoch number 115, Train loss = 0.038
Train Epoch number 116, Train loss = 0.107
Train Epoch number 117, Train loss = 0.034
Train Epoch number 118, Train loss = 0.032
Train Epoch number 119, Train loss = 0.037
Train Epoch number 120, Train loss = 0.049
Train Epoch number 121, Train loss = 0.060
Train Epoch number 122, Train loss = 0.044
```

```
Train Epoch number 123, Train loss = 0.074
Train Epoch number 124, Train loss = 0.039
Train Epoch number 125, Train loss = 0.053
Train Epoch number 126, Train loss = 0.037
Train Epoch number 127, Train loss = 0.037
Train Epoch number 128, Train loss = 0.047
Train Epoch number 129, Train loss = 0.037
Train Epoch number 130, Train loss = 0.068
Train Epoch number 131, Train loss = 0.044
Train Epoch number 132, Train loss = 0.036
Train Epoch number 133, Train loss = 0.045
Train Epoch number 134, Train loss = 0.031
Train Epoch number 135, Train loss = 0.037
Train Epoch number 136, Train loss = 0.068
Train Epoch number 137, Train loss = 0.043
Train Epoch number 138, Train loss = 0.061
Train Epoch number 139, Train loss = 0.044
Train Epoch number 140, Train loss = 0.033
Train Epoch number 141, Train loss = 0.033
Train Epoch number 142, Train loss = 0.040
Train Epoch number 143, Train loss = 0.036
Train Epoch number 144, Train loss = 0.043
Train Epoch number 145, Train loss = 0.037
Train Epoch number 146, Train loss = 0.040
Train Epoch number 147, Train loss = 0.039
Train Epoch number 148, Train loss = 0.039
Train Epoch number 149, Train loss = 0.064
Train Epoch number 150, Train loss = 0.032
Train Epoch number 151, Train loss = 0.035
Train Epoch number 152, Train loss = 0.034
Train Epoch number 153, Train loss = 0.039
Train Epoch number 154, Train loss = 0.036
Train Epoch number 155, Train loss = 0.037
Train Epoch number 156, Train loss = 0.039
Train Epoch number 157, Train loss = 0.047
Train Epoch number 158, Train loss = 0.039
Train Epoch number 159, Train loss = 0.037
Train Epoch number 160, Train loss = 0.049
Train Epoch number 161, Train loss = 0.033
Train Epoch number 162, Train loss = 0.038
```

Сохранено

```
# WGAN learning curve
plt.figure(figsize=(9, 5))
plt.plot(vae_fitter.loss_history)
plt.xlabel("Epoch Number", size=14)
plt.ylabel("Loss Function", size=14)
plt.xticks(size=14)
plt.yticks(size=14)
plt.title("Conditional VAE Learning Curve", size=14)
plt.grid(b=1, linestyle='--', linewidth=0.5, color='0.5')
plt.show()
```

Задание 9 (1 балл)

Реализуйте функцию для генерации новый объектов X по вектору условий y.

```
def generate(decoder, y, latent_dim):
    ### YOUR CODE IS HERE ######
    y = torch.tensor(y_train, dtype=torch.float, device=DEVICE)
    z = torch.randn((len(y), latent_dim)).to(DEVICE)
    X_fake = decoder(z, y)
    ### THE END OF YOUR CODE ###
    return X_fake # numpy
```

Теперь сгенерируем фейковые матрицы `X_fake_train` и `X_fake_test`. Сравним их с матрицами реальных объектов `X_train` и `X_test`

```
# garbage
```

Сохранено

```
ecoder, y_train, latent_dim).cpu().detach().numpy()
```

```
plot_hists(X_train, X_fake_train, names, label1="Real", label2="Fake", bins=50)
```

```
X_fake_test = generate(vae_fitter.decoder, y_test, latent_dim).cpu().detach().numpy()

plot_hists(X_test, X_fake_test, names, label1="Real", label2="Fake", bins=50)
```


Измерение качества генерации

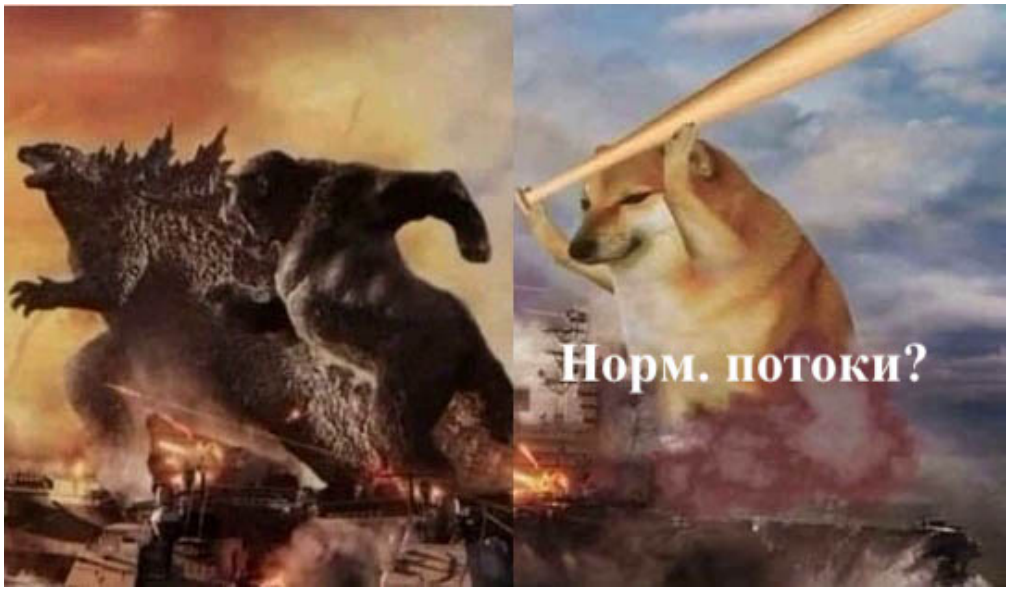
Измерим сходство распределений классификатором.

```
# собираем реальный и фейковые матрицы в одну
XX_train = np.concatenate((X_fake_train, X_train), axis=0)
XX_test = np.concatenate((X_fake_test, X_test), axis=0)

yy_train = np.array([0]*len(X_fake_train) + [1]*len(X_train))
yy_test = np.array([0]*len(X_fake_test) + [1]*len(X_test))


from sklearn.ensemble import GradientBoostingClassifier

# обучаем классификатор
clf = GradientBoostingClassifier()
clf.fit(XX_train, yy_train)

# получаем прогнозы
yy_test_proba = clf.predict_proba(XX_test)[:, 1]


from sklearn.metrics import roc_auc_score

auc = roc_auc_score(yy_test, yy_test_proba)
print("ROC AUC = ", auc)
```

```
ROC AUC =  0.6833309726548289
```

Вывод 3

Для CVAE получили ROC AUC около 0.7 (меньше лучше). Таким образом видим, что в
бя схожим образом. Но может их можно как-то
одель? :)



Бонус (0 баллов)

Попробуйте настроить параметры обучения каждой модели или еще как-нибудь их улучшить, чтобы получить как можно меньший ROC AUC. Что получилось? Какая модель лучше?

Сохранено