

Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное
учреждение высшего образования
«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

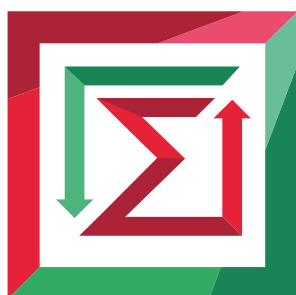


**НГТУ
НЭТИ**

Кафедра теоретической и прикладной информатики

Лабораторная работа №4
по дисциплине «Основы теории машинного обучения»

Порождающие состязательные сети



ФАКУЛЬТЕТ:	ПМИ
ГРУППА:	ПМИМ-01
СТУДЕНТЫ:	Ершов П.К. Малышкина Е.Д. Слободчикова А.Э.
ВАРИАНТ:	4
ПРЕПОДАВАТЕЛЬ:	Попов А.А.

Новосибирск

2021

1. Цель.

Получить практические навыки по решению прикладных задач с применением порождающих состязательных сетей (GAN).

2. Содержание работы.

1. Ознакомление с теоретическими основами используемых архитектур нейронных сетей (НС).
2. Ознакомление с возможностями их реализации в рамках свободных библиотек типа TensorFlow.
3. Поиск или формирование необходимых датасетов для выбранной задачи.
4. Кодирование и отладка программы. Обучение и тестирование НС.
5. Написание отчета.
6. Защита лабораторной работы.

3. Теоретическая часть

Generative Adversarial Net (GAN) это алгоритм машинного обучения, который входит в семейство порождающих моделей и состоит из двух нейронных сетей:

1. Генеративной сети (генератора), которая строит приближение данных.
2. Дискриминативной сети (дискриминатора), которая оценивает вероятность, что образец данных является тренировочным, а не сгенерирован генеративной сетью.

Целью обучения генератора является максимизация вероятности ошибки дискриминатора.

В данной работе будет использован генератор из 11 слоёв.

В качестве слоёв-активаторов будем использовать LeakyReLU.

Между основными слоями Dense и слоями-активаторами LeakyReLU будем располагать слой нормализации BatchNormalization, это позволит ускорить работу сети. В качестве оптимизатора будем использовать оптимизатор Adam. Функцией активации выходного слоя возьмём гиперболический тангенс.

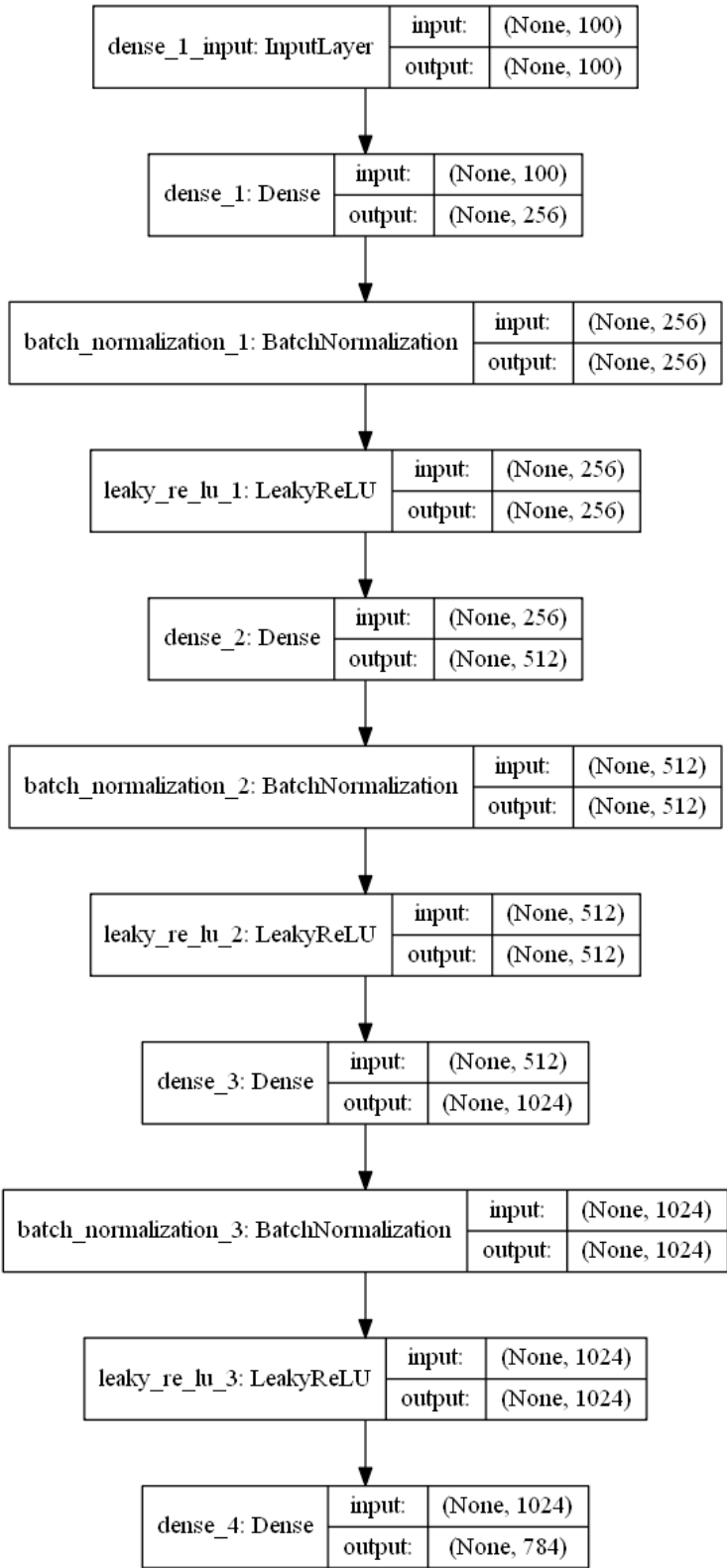


Рисунок 1. Структура генератора

В качестве дискриминатора также возьмём сеть из 11 слоёв. Но вместо слоя нормализации BatchNormalization будет использоваться слой Dropout, который позволит избежать переобучения. Также будет использоваться оптимизатор Adam. Функцией активации выходного слоя возьмём сигмоиду.

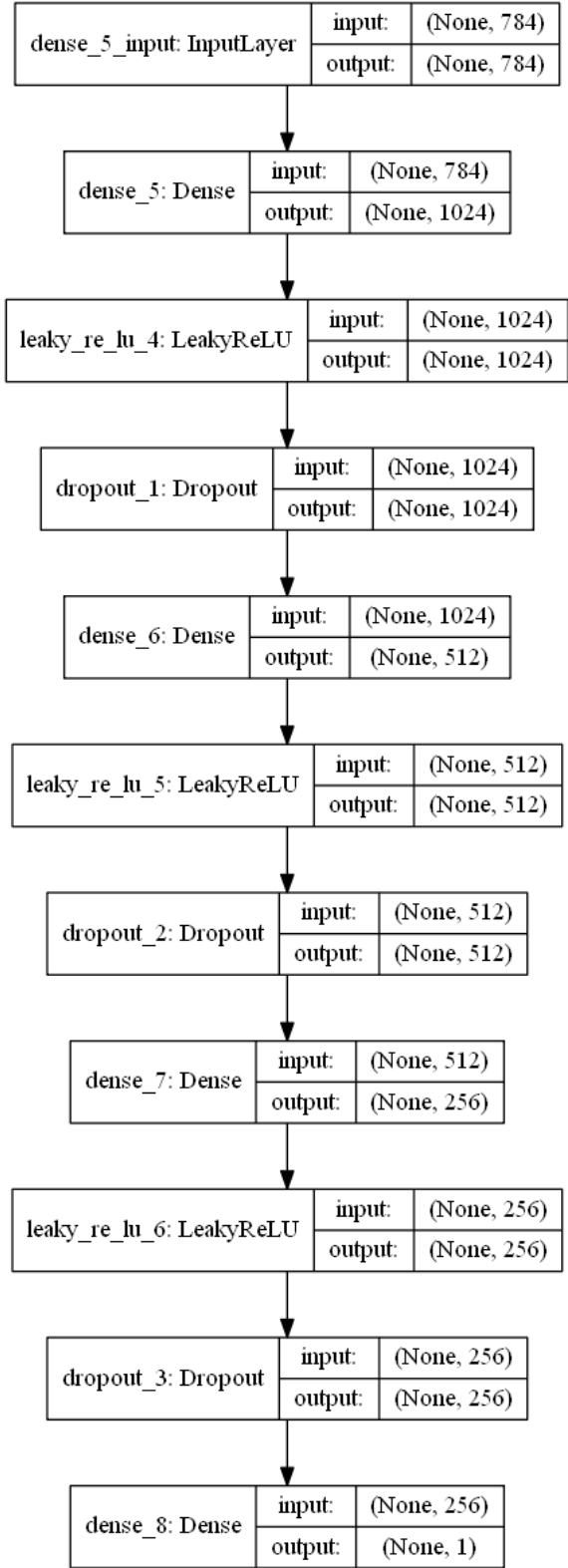


Рисунок 2. Структура дискриминатора

Проводить обучение будем с использование MNIST –набора данных, представляющих рукописные цифровые изображения. В основном MNIST используется для оценки производительности алгоритмов машинного обучения.

Число эпох зададим равным 250, число пакетов, т. е. проходов по данным, 256.

4. Ход работы

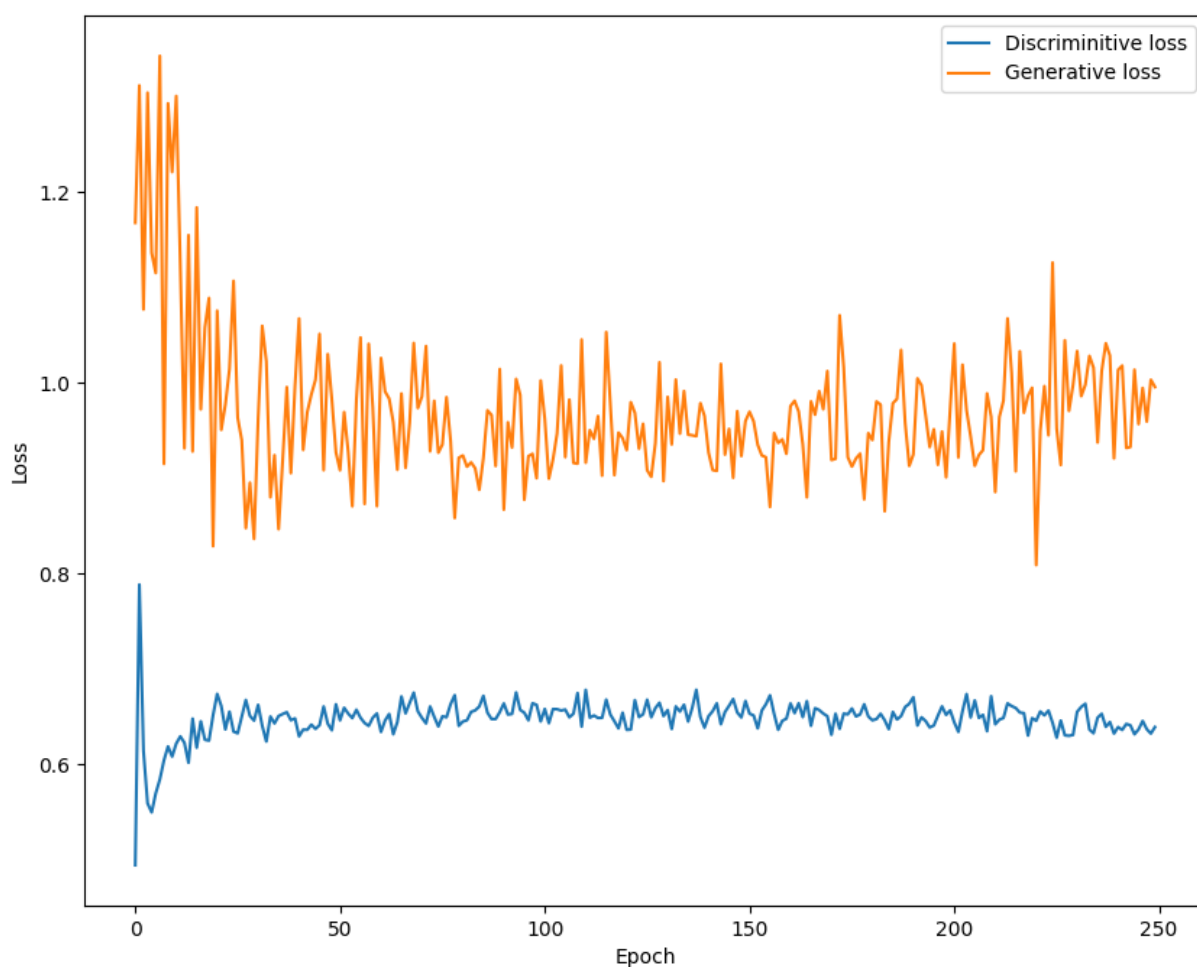



Рисунок 3. Графики функции потерь

Промежуточные результаты работы сети:

Эпоха	Результаты
1	

3	3	7	0	9	4	4	0	9	4
0	6	7	6	9	1	6	9	2	3
2	8	1	2	0	8	1	7	8	8
8	7	7	8	6	8	9	8	4	6
4	1	7	5	1	3	0	8	8	8
7	7	4	9	5	7	3	8	9	2
7	0	0	3	6	1	2	7	5	4
8	8	1	6	0	8	9	8	1	5
5	0	9	6	3	7	8	7	3	5
6	8	5	7	1	8	9	0	4	7

6	0	7	4	0	1	7	8	0	4
0	8	4	0	0	5	3	1	0	8
9	6	5	7	1	9	9	2	8	4
1	0	6	4	6	6	2	9	1	8
2	3	7	6	7	3	7	8	1	4
5	4	1	1	6	5	9	4	8	0
5	7	3	6	0	5	6	4	1	1
0	0	7	0	0	1	4	5	9	4
3	8	9	4	2	8	8	6	6	2
6	5	9	3	0	6	7	0	4	5

8	1	8	0	1	0	6	0	2	6
2	4	4	7	3	2	6	3	9	3
5	5	3	6	6	3	1	8	6	3
3	6	0	4	2	5	7	0	8	1
5	8	3	3	9	1	9	9	2	8
5	2	0	3	3	3	1	6	4	3
0	2	5	6	4	5	1	2	4	6
4	3	8	6	6	4	8	1	3	8
8	1	2	4	2	7	9	2	2	4
0	7	3	2	4	3	5	1	0	7

6	0	7	3	4	5	1	7	4	0
9	1	4	0	8	0	7	4	1	4
7	8	0	9	9	2	6	5	4	8
9	2	0	4	1	5	7	4	4	1
7	5	9	0	6	7	6	6	5	3
4	5	0	3	4	1	2	3	9	6
8	4	0	6	0	7	4	4	3	8
9	3	5	7	0	8	5	9	1	5
7	5	6	1	0	0	6	6	1	9
0	0	5	8	5	3	7	3	3	9

8	2	2	9	2	0	0	1	2	1
8	6	1	3	4	2	6	2	9	7
3	3	7	9	4	1	0	9	4	3
9	3	2	8	2	7	3	7	7	2
5	2	6	0	5	6	5	3	7	8
8	4	7	6	5	6	8	9	4	6
1	0	1	7	5	4	6	2	6	5
1	0	8	4	3	0	4	4	7	7
2	2	9	1	6	0	4	8	9	5
9	0	7	6	6	4	9	6	1	9

Результаты тестирования сети:

2	8	9	8	9	5	0	6	7	3
4	9	9	9	7	7	4	3	6	8
1	2	8	7	6	7	9	4	0	4
9	2	9	8	1	1	7	5	2	5
0	2	5	7	5	6	7	0	8	5
1	3	3	7	9	7	5	9	7	3
9	9	9	8	9	5	1	9	5	5
9	7	0	1	9	9	4	3	9	9
0	9	3	3	1	4	3	5	2	9
9	5	7	7	9	9	7	6	6	8

Рисунок 4. Результаты тестирования сети

5. Текст программы:

```
import os
os.environ["KERAS_BACKEND"] = "tensorflow"
import numpy as np
from tqdm import tqdm
import matplotlib.pyplot as plt

import math
from keras.layers import Input, Conv2D
from keras.models import Model, Sequential
from keras.layers.core import Reshape, Dense, Dropout, Flatten
from keras.layers.advanced_activations import LeakyReLU
from keras.layers.convolutional import Convolution2D, UpSampling2D
from keras.layers.normalization import BatchNormalization
from keras.datasets import mnist
from keras.optimizers import Adam
from keras import backend as K
from keras import initializers
from keras.utils.vis_utils import plot_model
import tensorflow as tf

K.set_image_data_format('channels_first')

np.random.seed(1000)

randomDim = 100

num_labels = 10
grid_size = 100
latent_size = 100

(X_train, y_train), (X_test, y_test) = mnist.load_data()
X_train = (X_train.astype(np.float32) - 127.5)/127.5
X_train = X_train.reshape(60000, 784)

adam = Adam(lr=0.0002, beta_1=0.5)

generator = Sequential()
generator.add(Dense(256, input_dim=randomDim,
kernel_initializer=initializers.RandomNormal(stddev=0.02)))
generator.add(BatchNormalization())
generator.add(LeakyReLU(0.2))

generator.add(Dense(512))
generator.add(BatchNormalization())
generator.add(LeakyReLU(0.2))

generator.add(Dense(1024))
generator.add(BatchNormalization())
generator.add(LeakyReLU(0.2))

generator.add(Dense(784, activation='tanh'))
generator.compile(loss='binary_crossentropy', optimizer=adam)

plot_model(generator, to_file='generator_plot.png', show_shapes=True,
show_layer_names=True)

discriminator = Sequential()
discriminator.add(Dense(1024, input_dim=784,
kernel_initializer=initializers.RandomNormal(stddev=0.02)))
```

```

discriminator.add(LeakyReLU(0.2))
discriminator.add(Dropout(0.3))
discriminator.add(Dense(512))
discriminator.add(LeakyReLU(0.2))
discriminator.add(Dropout(0.3))
discriminator.add(Dense(256))
discriminator.add(LeakyReLU(0.2))
discriminator.add(Dropout(0.3))
discriminator.add(Dense(1, activation='sigmoid'))
discriminator.compile(loss='binary_crossentropy', optimizer=adam)
plot_model(discriminator, to_file='discriminator_plot.png', show_shapes=True,
show_layer_names=True)
# Combined network
discriminator.trainable = False
ganInput = Input(shape=(randomDim,))
x = generator(ganInput)
ganOutput = discriminator(x)
gan = Model(inputs=ganInput, outputs=ganOutput)
gan.compile(loss='binary_crossentropy', optimizer=adam)

plot_model(gan, to_file='gan_plot.png', show_shapes=True, show_layer_names=True)

dLosses = []
gLosses = []

def plotLoss(epoch):
    plt.figure(figsize=(10, 8))
    plt.plot(dLosses, label='Discriminative loss')
    plt.plot(gLosses, label='Generative loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()
    plt.savefig('gan_loss_epoch%d.png' % epoch)

def plotGeneratedImages(epoch, examples=100, dim=(10, 10), figsize=(10, 10)):
    noise = np.random.normal(0, 1, size=[examples, randomDim])
    generatedImages = generator.predict(noise)
    generatedImages = generatedImages.reshape(examples, 28, 28)

    plt.figure(figsize=figsize)
    for i in range(generatedImages.shape[0]):
        plt.subplot(dim[0], dim[1], i+1)
        plt.imshow(generatedImages[i], interpolation='nearest', cmap='gray_r')
        plt.axis('off')
    plt.tight_layout()
    plt.savefig('gan_generated_image_epoch%d.png' % epoch)

def saveModels(epoch):
    generator.save('gan_generator_epoch%d.h5' % epoch)
    discriminator.save('gan_discriminator_epoch%d.h5' % epoch)

def plot_images(generator, noise_input, noise_labels, tag, step=0):
    os.makedirs(tag, exist_ok=True)
    images = generator.predict([noise_input, noise_labels])
    plt.figure(figsize=(10, 7))
    num_images = images.shape[0]
    image_size = images.shape[1]
    rows = int(math.sqrt(noise_input.shape[0]))
    for i in range(num_images):
        plt.subplot(10, 10, i + 1)

```

```

        image = np.reshape(images[i], [28, 28])
        plt.imshow(image, cmap='gray')
        plt.axis('off')
    plt.savefig('res_test1.png')
    plt.close()

def test_gan(generator, class_label=None):
    tag = "test_outputs"
    noise_input = np.random.uniform(-2.5, 2.5, size=[grid_size, latent_size])
    step = 0
    if class_label is None:
        noise_class = np.eye(num_labels)[np.random.choice(num_labels, grid_size)]
    else:
        noise_class = np.zeros((grid_size, num_labels))
        noise_class[:, class_label] = 1
        step = class_label
    plot_images(generator,
                tag=tag,
                noise_input=noise_input,
                noise_labels=noise_class,
                step=step)

    print('\n')
    print(tag, " Метки для генерируемых изображений: ", np.argmax(noise_class,
axis=1))

def train(epochs=1, batchSize=128):
    batchCount = int(X_train.shape[0] / batchSize)
    print('Epochs:', epochs)
    print('Batch size:', batchSize)
    print('Batches per epoch:', batchCount)

    for e in range(1, epochs+1):
        print('-'*15, 'Epoch %d' % e, '-'*15)
        for _ in tqdm(range(batchCount)):
            noise = np.random.normal(0, 1, size=[batchSize, randomDim])
            imageBatch = X_train[np.random.randint(0, X_train.shape[0],
size=batchSize)]

            generatedImages = generator.predict(noise)
            X = np.concatenate([imageBatch, generatedImages])

            yDis = np.zeros(2*batchSize)
            yDis[:batchSize] = 0.9

            discriminator.trainable = True
            dloss = discriminator.train_on_batch(X, yDis)

            noise = np.random.normal(0, 1, size=[batchSize, randomDim])
            yGen = np.ones(batchSize)
            discriminator.trainable = False
            gloss = gan.train_on_batch(noise, yGen)

            dlosses.append(dloss)
            glosses.append(gloss)

            if e == 1 or e % 50 == 0:
                plotGeneratedImages(e)
                saveModels(e)

    plotLoss(e)

```

```
if __name__ == '__main__':  
    train(250, 256)  
    m = tensorflow.keras.models.load_model("gan_generator_epoch_250.h5");  
    test_gan(m, 1)
```