СОДЕРЖАНИЕ

1. Введение

Данныи обзор рассматривает различные архитектуры и модификации генеративно-состязательных сетей, их обучение, задачи, которые решают данный вид сетей и сравнение с другими архитектурами. Кратко приводится обзор современных работ в данной области.

2. Генеративно-состязательная сеть

Генеративно-состязательная сеть — алгоритм машинного обучения, работа которого строится на основе двух «соперничающих» неиронных сетеи. Принцип алгоритма заключается в том, что одна из этих сетеи **G**, называемая генератором, пытается сгенерировать определенные образцы (например, изображения, видео или любые другие данные, на генерацию которых она запрограммирована), а другая сеть **D**, называемая дискриминатором, старается решить, является ли представленныи еи образец настоящим или сгенерированным. Задачеи генератора **G** является производство таких образцов, которые дискриминатор **D** сочтет настоящими, в то время как задача дискриминатора противоположна — он должен отбраковать сгенерированные образцы [9]. Таким образом, между генератором и дискриминатором возникает антагонистическая игра, в результате которои обе неиронные сети обучаются без учителя: генератор в итоге обучается генерировать образцы, по правдоподобности конкурирующие с настоящими, а дискриминатор — качественно выполнять проверку этих образцов [16].

Структура классическои генеративно-состязательнои сети на примере изображении в качестве образцов представлена на рис. 1.

Рис.1. Классическая архитектура генеративно-состязательной сети.

Здесь генератор производит изображения на основе случаиного шума, фо ма которого регулируется в процессе обучения, а дискриминатор, получая на входе изображения, на выходе должен вывести информацию о том, настоящее изображение или нет (1 — настоящее, 0 — поддельное) [6].

2.1. Анализ алгоритма обучения

Согласно схеме, приведеннои на рис. 1, дискриминатор **D** должен быть предварительно обучен с помощью реальных изображении, которым присвоена метка 1. Второи этап работы алгоритма заключается в создании искусственных изображении генератором **G** на основе случаиного шума, форма которого должна меняться в процессе обучения. Цель генератора **G** заключается в генерации таких изображении, чтобы дискриминатор как можно хуже их различил (принял за настоящие), и может быть описана следующим выражением:

$$min_G log(1 - D[G(z)])$$

где **D** — функция дискриминатора; **G** — функция генератора; **z** — случаиныи шум. Цель дискриминатора **D**, математически описанная с помощью следующего выражения, противоположна — он должен отличить подделку с как можно более высокои вероятностью:

$$max_D log(D(X_t) + log(1 - D(x_f))$$

где x_t — настоящее изображение; $x_f = G(z)$ — сгенерированное изображение. Таким образом, алгоритм представляет собои минимаксную игру для двух

неиронных сетеи [18]:

$$min_G max_D log(D(X_t) + log(1 - D(x_f))$$

В классическом алгоритме информация от генератора к дискриминатору передается путем прямого прохода по топологии, а от дискриминатора к генератору — путем обратного. Данныи алгоритм можно усовершенствовать, подавая результат работы дискриминатора напрямую на вход генератора, как представлено на рис. 2 [5].

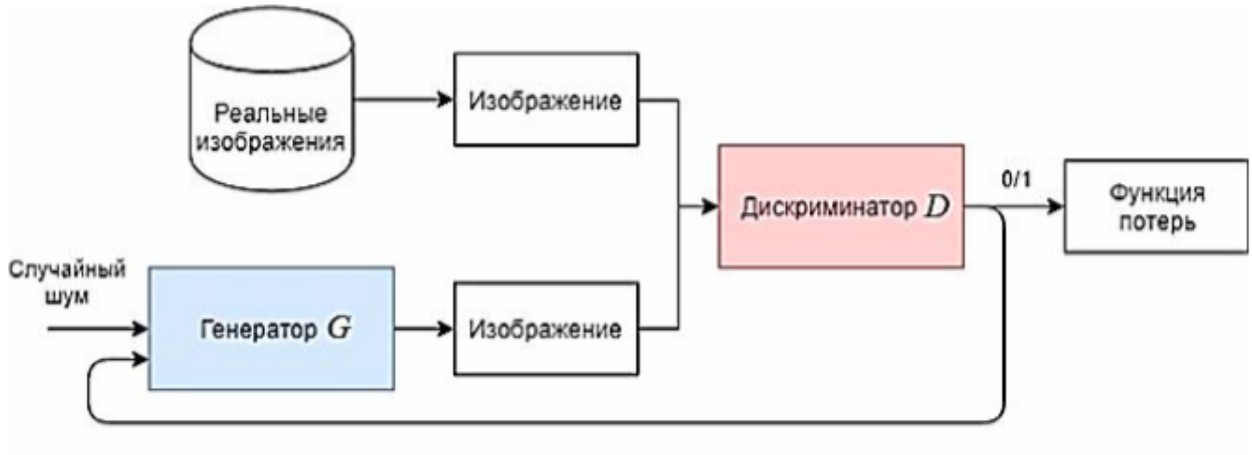

Рис.2. Генеративно-состязательная сеть с прямым проходом.

Однако при таком подходе возникает проблема на начальных этапах обучения модели — дискриминатор, обученныи на реальных изображениях, будет с легкостью отличать сгенерированные изображения, в результате чего его функция $\mathbf{D(G(z))}$, а соответственно и функция логарифма log1 – D[G(z)], будут колебаться рядом с нулевым значением, в результате чего генератор не будет получать никакои полезнои информации и не будет обучаться. Данную проблему можно решить, выразив цель генератора из выражения, заменив поиск минимума поиском максимума [8]:

$$max_G log(D(G(z)))$$

Данное решение порождает еще одно условие — на любом шаге обучения дискриминатор должен показывать точность, лежащую в некоторых оптимальных положительных границах. Этого можно добиться, применив следующии алгоритм обучения:

- осуществление нескольких итерации обучения дискриминатора на выборке из реальных и сгенерированных изображении при фиксированном генераторе;

- осуществление однои итерации обучения генератора на сгенерированном изображении и реакции фиксированного дискриминатора на него.

Этот подход позволит держать точность дискриминатора на оптимальном для данного шага обучения уровне и постепенно обучать генератор [3]. Особенностью подхода является то, что генератор учится создавать похожие на настоящие изображения, ни разу не увидев настоящего изображения, и обучается только на основе информации, получаемои от дискриминатора.

2.2. Автокодеры и VAE

Полезно сравнить генеративные состязательные сети с другими нейронными сетями, такими как автокодеры и вариационные автокодеры.

Автокодеры кодируют входные данные в векторы. Они создают скрытое или сжатое представление необработанных данных. Они полезны при уменьшении размерности: вектор, служащий в качестве скрытого представления, сжимает необработанные данные в меньшее количество. Автокодеры могут быть сопряжены с так называемым декодером, который позволяет восстанавливать входные данные на основе их скрытого представления, как и в случае с машиной Больцмана.

Вариационные автокодеры являются генеративным алгоритмом, который добавляет дополнительное ограничение для кодирования входных данных, а именно то, что скрытые представления нормализуются. Вариационные автокодеры способны сжимать данные как автокодеры и синтезировать данные подобно GAN. Однако, в то время как GAN генерируют данные детализовано, изображения, созданные VAE, бывают более размытыми. Примеры Deeplearning4j включают в себя как автокодеры, так и вариационные автокодеры.

2.3. Анализ структуры генеративно-состязательнои сети

Главным недостатком описаннои генеративно-состязательнои сети является то, что данная модель не умеет извлекать признаки из получаемых настоящих

изображении. Это способствует тому, что при переносе образца, которыи учится создавать генератор, из одного домена в другои (например, при изменении фона изображения) система не распознает, что изменен был лишь контекст, и примет это за другои образец [14]. К примеру, такои недостаток делает данную структуру уязвимои к шуму на фотографиях.

Описанныи недостаток можно устранить, подоидя к основнои идее генеративно-состязательнои сети с другои стороны и представив топологию, как показано на рис. 3 [15].

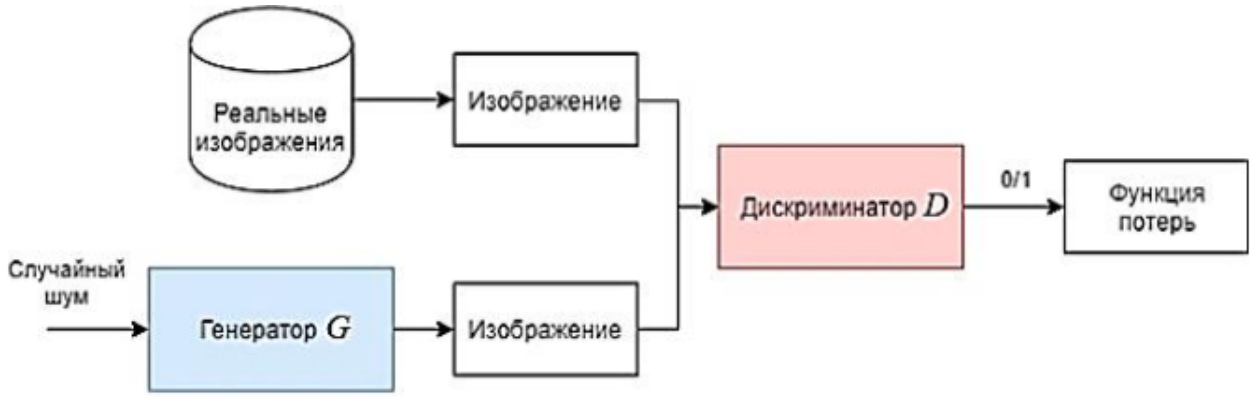

Рис.3. Состязательный автокодировщик.

В даннои структуре генератор по сути является автокодировщиком. Данная структура неироннои сети представляет собои обычныи перцептрон с одним скрытым слоем с тои лишь поправкои, что выходнои слои имеет столько же неиронов, сколько и входнои, а размерность скрытого слоя, наоборот, ограничена. Принцип работы автокодировщика заключается в том, что выходнои результат такои неироннои сети должен быть близок к исходным подаваемым данным, а из-за вычислительного ограничения скрытого слоя в ходе своих вычислении неиронная сеть вынуждена искать обобщения поступающих на вход данных и выполнять их сжатие [9]. Теперь настоящие изображения поступают на вход генератора, скрытыи слои которого подключен к входу дискриминатора. Также на вход дискриминатора подается некоторое распределение шума, которое может быть как заданным аналитически, так и случаиным. Задачеи дискриминатора в данном случае является различение изображении, генерируемых кодером (входным слоем генератора), и изображении из распределения, а также подстраивание параметров кодера под заданное случаиное или аналитически описанное распределение [11]. Такая структура имеет две функции

потерь — одна для качества данных (соответствия выходных данных автокодировщика входным), а вторая — для генеративно-состязательнои сети. Процесс обучения в данном случае будет выглядеть следующим образом:

- итерация обучения автокодировщика, в ходе которои обновляются параметры генератора для того, чтобы выход сети соответствовала входу.
- итерации обучения генеративно-состязательнои сети, описанные выше:
  - несколько итерации обучения дискриминатора на основе настоящих и сгенерированных изображении при фиксированном генераторе.
  - одна итерация обучения генератора (корректировка скрытого слоя автокодировщика) при фиксированном дискриминаторе.

В результате такого обучения представленная сеть будет проецировать образы из набора настоящих изображении на заданное распределение, а также длялюбои величины из заданного распределения находить адекватныи образ, основанныи на предоставленном наборе изображении.

3. Известные модификации алгоритма

3.1. Conditional Generative Adversarial Nets

Условные порождающие состязательные сети это модифицированная версия алгоритма GAN, которая позволяет генерировать объекты с дополнительными условиями. Условие может быть любой дополнительной информацией, например, меткой класса или данными из других моделей. Добавление данных условий в существующую архитектуру осуществляется с помощью расширения вектором у входных данных генератора и дискриминатора.

В качестве примера использования данного алгоритма можно рассмотреть задачу генерации рукописных цифр. CGAN был натренирован на датасете MNIST с метками классов представленных в виде one-hot векторов.

3.2. Deep Convolutional Generative Adversarial Nets

DCGAN модификация алгоритма GAN, основными архитектурными изменениями которой являются:

- Замена всех пулинговых слоев на страйдинговые свертки (strided convolutions) в дискриминаторе и частично-страйдинговые свертки (fractional-strided convolutions) в генераторе;
- Использование батчинговой нормализации для генератора и дискриминатора;
- Удаление всех полносвязных скрытых уровней для более глубоких архитектур;
- Использование ReLU в качестве функции активации в генераторе для всех слоев, кроме последнего, где используется tanh;
- Использование LeakyReLU в качестве функции активации в дискриминаторе для всех слоев.

Помимо задачи генерации объектов, данный алгоритм хорошо показывает себя в качестве feature extractor'a. Данный алгоритм был натренирован на датасете Imagenet-1k, после чего были использованы значения со сверточных слоев дискриминатора, подвергнутые max-pooling'y, чтобы образовать матрицы 44 и получить общий вектор признаков на их основе. L2-SVM с данным feature extractor'ом на датасете CIFAR-10 превосходит по точности решения, основанные на алгоритме K-Means [12].

4. Примеры использования

GAN используются для получения фотореалистичных изображений, например для элементов промышленного дизайна, дизайна интерьера, одежды, сумок, портфелей, сцен компьютерных игр и т. д. Сети GAN используются также в сети Facebook. В последнее время системы GANs стали использоваться для подготовки кадров фильмов или мультипликации. Также эти системы помогают воссоздать трёхмерную модель объекта с помощью фрагментарных изображений и улучшить изображения, полученные из астрономических наблюдений.

Наиболее известные GAN-based сети:

- **Auxiliary GAN**: вариант GAN-архитектуры, использующий метки данных;

- **SN-GAN**: GAN с новым подходом решения проблемы нестабильного обучения через спектральную нормализацию;
- **SAGAN**: GAN, основанный на механизме внимания;
- **BigGAN**: GAN с ортогональной регуляризацией, позволившей разрешить проблему коллапсирования при долгом обучении;
- **CycleGAN**: меняет изображения с одного домена на другой, например, лошадей на зебр;
- **SRGAN**: создает изображения с высоким разрешением из более низкого разрешения;
- **Pix2Pix**: создает изображения по семантической окраске;
- **StackGAN**: создает изображения по заданному тексту;
- **MidiNet**: генерирует последовательность нот, таким образом, создает мелодию.

Основные задачи, которые решаются с помощью генеративно-состязательных сетей рассмотренны далее.

4.1. Генерация текста в изображения [17] [1]

Автоматический синтез реалистичных изображений из текста был бы интересен и полезен, но современные системы искусственного интеллекта все еще далеки от этой цели. Тем не менее, в последние годы были разработаны универсальные и мощные рекуррентные архитектуры нейронных сетей для изучения представлений дискриминационных текстовых элементов. Между тем, глкбокие генеративно-состязательные сети начали генерировать очень убедительные изображения определенных категорий, таких как лица, обложки альбомов и интерьеры комнат.

4.2. Перевод изображения в изображение [7]

На данный момент наиболее популярный пример в этой области это - pix2pix. Модель pix2pix работает, обучая пары изображений, таких как метки

фасадов зданий и фасадов зданий, и затем пытается сгенерировать соответствующее выходное изображение из любого входного изображения, которое вы ему даете. Сеть ГАН может выступат как универсальное решение проблем преобразования изображения в изображение. Эти сети не только изучают отображение от входного изображения к выходному изображению, но также изучают функцию потерь для обучения этому отображению. Это позволяет применять тот же общий подход к задачам, которые традиционно требуют совершенно разных формулировок потерь. Мы демонстрируем, что этот подход эффективен при синтезе фотографий с картами меток, реконструкции объектов по картам и раскрашивании изображений. С момента выпуска программного обеспечения pix2pix, большое количество пользователей Интернета (многие из которых являются художниками) опубликовали свои собственные эксперименты с этой системой, еще раз продемонстрировав ее широкую применимость и простоту применения без необходимости настройки параметров.

4.3. Увеличение разрешения изображения [4]

Несмотря на прорыв в точности и скорости сверхразрешения одного изображения с использованием более быстрых и глубоких сверточных нейронных сетей, одна центральная проблема остается в значительной степени нерешенной: как восстановить более мелкие детали текстуры, когда сверхразрешимся при больших коэффициентах масштабирования? Поведение методов сверхразрешения, основанных на оптимизации, главным образом определяется выбором целевой функции. Последние работы были в основном направлены на минимизацию среднеквадратичной ошибки реконструкции. Полученные оценки имеют высокие пиковые отношения сигнал/шум, но им часто не хватает высокочастотных деталей и они не удовлетворяют восприятию в том смысле, что они не соответствуют точности воспроизведения, ожидаемой при более высоком разрешении.На данный момент разработана SRGAN. Это первый фреймворк, способный выводить фотореалистичные естественные изображения для четырехкратного увеличения. Для этого используется функция потери восприятия, которая состоит из состязательной потери и потери содержимого. Потеря состязательности подталкивает решение к естественному многообразию изображений, используя сеть дискриминаторов, которая обучена различать сверхраз-

решимые изображения и оригинальные фотореалистичные изображения. Эта сеть способна восстанавливать фотореалистичные текстуры из сильно уменьшенных изображений на общедоступных тестах. Обширный критерий оценки среднего мнения (MOS) показывает чрезвычайно значительный прирост качества восприятия при использовании SRGAN.

4.4. Прогнозирование [2, 13]

Для прогнозирования различных систем давно используют глубокое обучение. Рекурентные и LSTM сети чаще других используют для решения подобных проблем, но с недавних пор начали использовать GAN. По результатам экспериментов GAN превосходили в результате обычные рекуретные сети.

Понимание движений объекта и динамики сцены является основной проблемой в компьютерном зрении. Для обоих задачи распознавания видео (например, классификация действий) и генерации видео (например, прогнозирование будущего), нужна модель того, как меняются сцены. Тем не менее, создание модели динамики является сложной задачей. потому что существует огромное количество способов, которыми объекты и сцены могут меняться.

4.5. 3D GAN [10]

3D-GAN - это архитектура GAN для создания трехмерных фигур. Генерация трехмерных фигур обычно представляет собой сложную проблему из-за сложностей, связанных с обработкой трехмерных изображений. 3D-GAN - это решение, которое может генерировать реалистичные и разнообразные трехмерные фигуры.

На основе этой архитектуры создали сети, которые умееют классифицировать 3d объекты. Но результаты оставляют желать лучшего это связано с трудностью обучения, малым количеством данных для обучения и вычислительными ресурсами.

ЗАКЛЮЧЕНИЕ

Потенциал GAN огромен, поскольку они имитируют любое распределение данных. GAN обучают создавать структуры, устрашающе похожие на сущности из нашего мира в области изображений, музыки, речи, прозы. Генеративно-состязательные сети, в некотором смысле, роботы-художники, и результат их работы впечатляет.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Alec Radford Luke Metz Soumith Chintala. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks.
2. Carl Vondrick Hamed Pirsiavash Antonio Torralba. Generating Videos with Scene Dynamics.
3. Chintala Soumith. How to train a GAN? Tips and tricks to make GANs work.
4. Christian Ledig Lucas Theis Ferenc Huszar Jose Caballero Andrew Cunningham-Alejandro Acosta-Andrew Aitken Alykhan Tejani Johannes Totz Zehan Wang Wenzhe Shi. Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network.
5. Ganin Ya. Lempitsky V. Unsupervised domain adaptation by backpropagation.
6. Goodfellow I.J. Pouget-Abadie J. Mirza M. Xu B. Warde-Farley D.-Ozair Sh. Courville A. Bengio Yo. Generative adversarial networks.
7. Hesse Christopher. Interactive Image Translation with pix2pix-tensorflow.
8. I. Gitman. Adversarial networks.
9. Ian J. Goodfellow Jean Pouget-Abadie Mehdi Mirza Bing Xu-David Warde-Farley Sherjil Ozair Aaron Courville Yoshua Bengio. Generative Adversarial Networks.
10. Jiajun Wu Chengkai Zhang Tianfan Xue William T. Freeman Joshua B. Tenenbaum. Learning a Probabilistic Latent Space of Object Shapes via 3D Generative-Adversarial Modeling.
11. Makhzani A. Shlens J. Jaitly N. Goodfellow I. Frey B. Adversarial autoencoders.
12. Metz Alec Radford Luke. UNSUPERVISED REPRESENTATION LEARNING WITH DEEP CONVOLUTIONAL GENERATIVE ADVERSARIAL NETWORKS.
13. Michael Mathieu Camille Couprie Yann LeCun. Deep multi-scale video prediction beyond mean square error.
14. Mirza M. Osindero S. Conditional generative adversarial nets.
15. N. Jaitly. A beginner's guide to deep autoencoders.
16. Nicholson Chris. GAN: a beginner's guide to generative adversarial networks.

17. Phillip Isola Jun-Yan Zhu Tinghui Zhou Alexei A. Efros. Image-to-Image Translation with Conditional Adversarial Networks.
18. Strakho Max. Generative adversarial networks.

ЛИСТИНГИ

Листинг 1: Пример реализации GAN в Keras

```
def make_trainable(net, val):
    net.trainable = val
    for l in net.layers:
        l.trainable = val

def create_gan(channels, height, width):

    input_img = Input(shape=(channels, height, width))

    m_height, m_width = int(height/8), int(width/8)

    # generator
    z = Input(shape=(latent_dim, ))
    x = Dense(256*m_height*m_width)(z)
    #x = BatchNormalization()(x)
    x = Activation('relu')(x)
    #x = Dropout(0.3)(x)

    x = Reshape((256, m_height, m_width))(x)

    x = Conv2DTranspose(256, kernel_size=(5, 5), strides=(2, 2), padding='same', activation='
        relu')(x)

    x = Conv2DTranspose(128, kernel_size=(5, 5), strides=(2, 2), padding='same', activation='
        relu')(x)

    x = Conv2DTranspose(64, kernel_size=(5, 5), strides=(2, 2), padding='same', activation='
        relu')(x)

    x = Conv2D(channels, (5, 5), padding='same')(x)
    g = Activation('tanh')(x)

    generator = Model(z, g, name='Generator')

    # discriminator
    x = Conv2D(128, (5, 5), padding='same')(input_img)
    #x = BatchNormalization()(x)
    x = LeakyReLU()(x)
    #x = Dropout(0.3)(x)
    x = MaxPooling2D(pool_size=(2, 2), padding='same')(x)
    x = Conv2D(256, (5, 5), padding='same')(x)
    x = LeakyReLU()(x)
    x = MaxPooling2D(pool_size=(2, 2), padding='same')(x)
    x = Conv2D(512, (5, 5), padding='same')(x)
    x = LeakyReLU()(x)
    x = MaxPooling2D(pool_size=(2, 2), padding='same')(x)
    x = Flatten()(x)
    x = Dense(2048)(x)
    x = LeakyReLU()(x)
    x = Dense(1)(x)
    d = Activation('sigmoid')(x)

    discriminator = Model(input_img, d, name='Discriminator')

```

```
    gan = Sequential()
    gan.add(generator)
    make_trainable(discriminator, False) #discriminator.trainable = False
    gan.add(discriminator)

    return generator, discriminator, gan

gan_gen, gan_ds, gan = create_gan(channels, height, width)

gan_gen.summary()
gan_ds.summary()
gan.summary()

opt = Adam(lr=1e-3)
gopt = Adam(lr=1e-4)
dopt = Adam(lr=1e-4)

gan_gen.compile(loss='binary_crossentropy', optimizer=gopt)
gan.compile(loss='binary_crossentropy', optimizer=opt)

make_trainable(gan_ds, True)
gan_ds.compile(loss='binary_crossentropy', optimizer=dopt)
```

Листинг 2: Процедура обучения GAN

```
for epoch in range(epochs):
    print('Epoch {} from {} ...'.format(epoch, epochs))

    n = x_train.shape[0]
    image_batch = x_train[np.random.randint(0, n, size=batch_size),:,:,:]

    noise_gen = np.random.uniform(-1, 1, size=[batch_size, latent_dim])

    generated_images = gan_gen.predict(noise_gen, batch_size=batch_size)

    if epoch % 10 == 0:
        print('Save gens ...')
        save_images(generated_images)
        gan_gen.save_weights('temp/gan_gen_weights_'+str(height)+'.h5', True)
        gan_ds.save_weights('temp/gan_ds_weights_'+str(height)+'.h5', True)
        # save loss
        df = pd.DataFrame( {'d_loss': d_loss, 'g_loss': g_loss} )
        df.to_csv('temp/gan_loss.csv', index=False)

    x_train2 = np.concatenate( (image_batch, generated_images) )
    y_tr2 = np.zeros( [2*batch_size, 1] )
    y_tr2[:batch_size] = 1

    d_history = gan_ds.train_on_batch(x_train2, y_tr2)
    print('d:', d_history)
    d_loss.append( d_history )

    noise_gen = np.random.uniform(-1, 1, size=[batch_size, latent_dim])
    g_history = gan.train_on_batch(noise_gen, np.ones([batch_size, 1]))
    print('g:', g_history)
    g_loss.append( g_history )
```