**Лекция 8**

**Модель нейронной сети GAN**

GAN были впервые представлены в работе 2014-го года Ian’a Goodfellow’a и других исследователей университета Монреаля, включая Yoshua Bengio. Директор Facebook по исследованиям [искусственного интеллекта](https://neurohive.io/ru/osnovy-data-science/iskusstvennyj-intellekt-voprosy-i-otvety/) Yann LeCun назвал состязательную претренировку сетей “самой интересной идеей в [машинном обучении](https://neurohive.io/ru/osnovy-data-science/vvedenie-v-mashinnoe-obuchenie-kto-ego-primenjaet-i-kak-stat-razrabotchikom/) за последние 10 лет”.

Вряд ли вы подумаете, что программиста можно назвать артистом, но, на самом деле, программирование это очень творческая профессия. Творчество, основанное на логике — John Romero

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-состязательная нейросеть, в некотором смысле, робот-художник, и результаты ее работы впечатляют.



Реалистичные изображения несуществующих знаменитостей, созданные с помощью GAN

Рассмотрим, как работают алгоритмы дикриминатора и генератора.

**Дискриминатор**

Дискриминационные алгоритмы пытаются классифицировать входные данные. Учитывая особенности полученных данных, они стараются определить категорию, к которой они относятся.

К примеру, пробегая все слова в письме дискриминационный алгоритм может предсказать, является сообщение спамом или не спамом. Спам — это категория, а пакет слов, собранный из электронной почты — образы, которые составляют входные данные. Математически категории обозначают **y**, а образы обозначают **x**. Запись **p(y|x)** используется для обозначения «вероятности **y** при заданном **x**», которая обозначает «вероятность того, что электронное письмо является спамом при имеющемся наборе слов».

Итак, дискриминационные функции сопоставляют образы с категорией. Они заняты только этой корреляцией.

**Генератор**

Генеративные алгоритмы заняты обратным. Вместо того, чтобы предсказывать категорию по имеющимся образам, они пытаются подобрать образы к данной категории.

В то время как дискриминационные алгоритмы волнует взаимосвязь между y и x, генеративные алгоритмы волнует “откуда берутся **x**”. Они позволяют находить **p(x|y)**, вероятность **x** при данном **y** или вероятность образов при данном классе (генеративные алгоритмы также могут использоваться в качестве классификаторов. Они могут делать больше, чем классифицировать входные данные.)

Еще одно представление о работе генеративных алгоритмов можно получить, разделяя дискриминационные модели от генеративных таким образом:

* Дискриминационные модели изучают границу между классами;
* Генеративные модели моделируют распределение отдельных классов.

**Как работает генеративно-состязательная нейросеть**

Одна нейронная сеть, называемая генератором, генерирует новые экземпляры данных, а другая — дискриминатор, оценивает их на подлинность; т.е. дискриминатор решает, относится ли каждый экземпляр данных, который он рассматривает, к набору тренировочных данных или нет.

Предположим, мы пытаемся сделать что-то более банальное, чем повторить портрет Моны Лизы. Мы сгенерируем рукописные цифры, подобные тем, что имеются [в наборе данных MNIST](http://yann.lecun.com/exdb/mnist/). Цель дискриминатора — распознать подлинные экземпляры из набора.

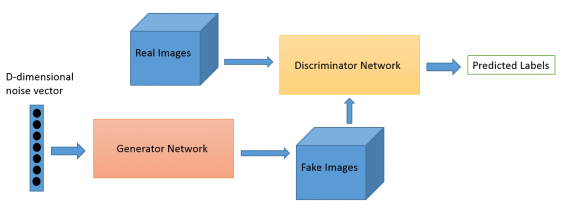
Между тем, генератор создает новые изображения, которые он передает дискриминатору. Он делает это в надежде, что они будут приняты подлинными, хотя являются поддельными. Цель генератора состоит в том, чтобы генерировать рукописные цифры, которые будут пропущены дискриминатором. Цель дискриминатора — определить, является ли изображение подлинным.

Шаги, которые проходит GAN:

* Генератор получает рандомное число и возвращает изображение.
* Это сгенерированное изображение подается в дискриминатор наряду с потоком изображений, взятых из фактического набора данных.
* Дискриминатор принимает как реальные, так и поддельные изображения и возвращает вероятности, числа от 0 до 1, причем 1 представляет собой подлинное изображение и 0 представляет фальшивое.

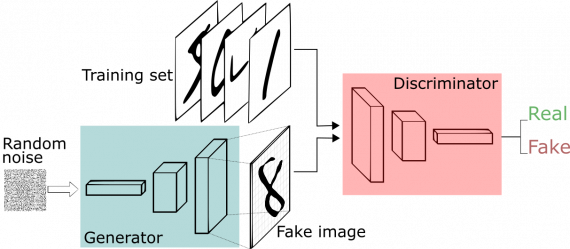
Таким образом, у есть двойной цикл обратной связи:

* Дискриминатор находится в цикле с достоверными изображениями.
* Генератор находится в цикле вместе с дискриминатором



Вы можете представить GAN как фальшивомонетчика и полицейского, играющих в кошки мышки, где фальсификатор учится изготавливать ложные купюры, а полицейский учится их обнаруживать. Оба динамичны; т. е. полицейский тоже тренируется (возможно, центральный банк отмечает пропущенные купюры), и каждая сторона приходит к изучению методов другого в постоянной эскалации.

Сеть дискриминаторов представляет собой стандартную [сверточную сеть](https://neurohive.io/vidy-nejrosetej/glubokaya-svertochnaja-nejronnaja-set/), которая может классифицировать изображения, подаваемые на нее с помощью биномиального классификатора, распознающего изображения как реальные или как поддельные. Генератор в некотором смысле представляет собой обратную сверточную сеть: хотя стандартный сверточный классификатор принимает изображение и уменьшает его разрешение, чтобы получить вероятность, генератор принимает вектор случайного шума и преобразует его в изображение. Первый отсеивает данные с помощью методов понижения дискретизации, таких как maxpooling, а второй генерирует новые данные.

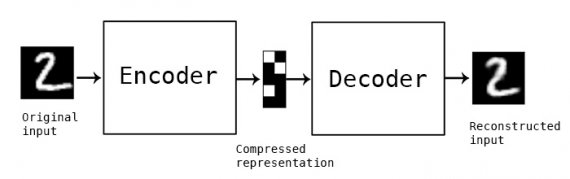


Обе сети пытаются оптимизировать целевую функцию или функцию потерь в игре zero-zum. Это, по сути, модель актера-критика (actor-critic). Когда дискриминатор меняет свое поведение, то и генератор меняет, и наоборот.

**Автокодеры и VAE**

Полезно сравнить генеративные состязательные сети с другими нейронными сетями, такими как **автокодеры (автоэнкодеры) 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 генеративно-состязательная нейросеть генерирует данные детализировано, созданные VAE изображение, бывают более размытыми. Примеры Deeplearning4j включают в себя как автокодеры, так и вариационные автокодеры.

Вы можете разделить генеративные алгоритмы на три типа, которые имея:

* категорию, предсказывают связанные функции (Naive Bayes);
* скрытое представление, предсказывают связанные функции (VAE, GAN);
* некоторые образы, предсказывают остальное ([inpainting](https://neurohive.io/ru/papers/rekonstrukcija-fotografij-metodom-chastichnoj-svertki-ot-nvidia/), imputation);

**Советы по обучению GAN**

Когда вы тренируете дискриминатор, удерживайте значения генератора постоянными; и когда вы тренируете генератор, удерживайте дискриминатор на одном уровне. Каждый должен тренироваться против статичного противника. Например, генератору это позволит лучше считывать градиент, по которому он должен учиться.

Точно так же предварительная тренировка дискриминатора против MNIST будет способствовать установлению более четкого градиента.

**Каждая часть GAN может одолеть другую.** Если дискриминатор слишком хорош, он будет возвращать значения очень близкие к 0 или к 1, так что генератор будет испытывать трудности в чтении градиента. Если генератор слишком хорош, он будет постоянно использовать недостатки дискриминатора, приводящие к неправильным негативам.

GAN требуют много времени на тренировку. На одном GPU тренировка может занимать часы, а на одном CPU — более одного дня. Несмотря на сложность настройки и, следовательно, использования, GAN стимулировали создание многих интересных исследований и статей, например:

* [PGGAN — прогрессивная генеративная нейросеть от Nvidia](https://neurohive.io/ru/papers/pggan-progressivnaja-generativnaja-nejroset-ot-nvidia/)
* [TwinGAN — междоменный перенос человеческих портретов](https://neurohive.io/ru/papers/twingan-mezhdomennyj-perenos-chelovecheskih-portretov/)
* [XOGAN: преобразование одного изображения в несколько](https://neurohive.io/ru/papers/xogan-preobrazovanie-odnogo-izobrazhenija-v-neskolko-bez-uchitelja/)
* [Метод восстановления глаз при помощи Exemplar GAN от Facebook AI Research](https://neurohive.io/ru/papers/metod-vosstanovlenija-glaz-pri-pomoshhi-exemplar-gan-ot-facebook-ai-research/)

**Просто покажите код**

Вот пример GAN, запрограммированной в библиотеке Keras, из которой модели могут в дальнейшем быть импортированы в Deeplearning4j.

class GAN():

def \_\_init\_\_(self):

self.img\_rows = 28

self.img\_cols = 28

self.channels = 1

self.img\_shape = (self.img\_rows, self.img\_cols, self.channels)

optimizer = Adam(0.0002, 0.5)

# Build and compile the discriminator

self.discriminator = self.build\_discriminator()

self.discriminator.compile(loss='binary\_crossentropy',

optimizer=optimizer,

metrics=['accuracy'])

# Build and compile the generator

self.generator = self.build\_generator()

self.generator.compile(loss='binary\_crossentropy', optimizer=optimizer)

# The generator takes noise as input and generated imgs

z = Input(shape=(100,))

img = self.generator(z)

# For the combined model we will only train the generator

self.discriminator.trainable = False

# The valid takes generated images as input and determines validity

valid = self.discriminator(img)

# The combined model (stacked generator and discriminator) takes

# noise as input => generates images => determines validity

self.combined = Model(z, valid)

self.combined.compile(loss='binary\_crossentropy', optimizer=optimizer)

def build\_generator(self):

noise\_shape = (100,)

model = Sequential()

model.add(Dense(256, input\_shape=noise\_shape))

model.add(LeakyReLU(alpha=0.2))

model.add(BatchNormalization(momentum=0.8))

model.add(Dense(512))

model.add(LeakyReLU(alpha=0.2))

model.add(BatchNormalization(momentum=0.8))

model.add(Dense(1024))

model.add(LeakyReLU(alpha=0.2))

model.add(BatchNormalization(momentum=0.8))

model.add(Dense(np.prod(self.img\_shape), activation='tanh'))

model.add(Reshape(self.img\_shape))

model.summary()

noise = Input(shape=noise\_shape)

img = model(noise)

return Model(noise, img)

def build\_discriminator(self):

img\_shape = (self.img\_rows, self.img\_cols, self.channels)

model = Sequential()

model.add(Flatten(input\_shape=img\_shape))

model.add(Dense(512))

model.add(LeakyReLU(alpha=0.2))

model.add(Dense(256))

model.add(LeakyReLU(alpha=0.2))

model.add(Dense(1, activation='sigmoid'))

model.summary()

img = Input(shape=img\_shape)

validity = model(img)

return Model(img, validity)

def train(self, epochs, batch\_size=128, save\_interval=50):

# Load the dataset

(X\_train, \_), (\_, \_) = mnist.load\_data()

# Rescale -1 to 1

X\_train = (X\_train.astype(np.float32) - 127.5) / 127.5

X\_train = np.expand\_dims(X\_train, axis=3)

half\_batch = int(batch\_size / 2)

for epoch in range(epochs):

# ---------------------

# Train Discriminator

# ---------------------

# Select a random half batch of images

idx = np.random.randint(0, X\_train.shape[0], half\_batch)

imgs = X\_train[idx]

noise = np.random.normal(0, 1, (half\_batch, 100))

# Generate a half batch of new images

gen\_imgs = self.generator.predict(noise)

# Train the discriminator

d\_loss\_real = self.discriminator.train\_on\_batch(imgs, np.ones((half\_batch, 1)))

d\_loss\_fake = self.discriminator.train\_on\_batch(gen\_imgs, np.zeros((half\_batch, 1)))

d\_loss = 0.5 \* np.add(d\_loss\_real, d\_loss\_fake)

# ---------------------

# Train Generator

# ---------------------

noise = np.random.normal(0, 1, (batch\_size, 100))

# The generator wants the discriminator to label the generated samples

# as valid (ones)

valid\_y = np.array([1] \* batch\_size)

# Train the generator

g\_loss = self.combined.train\_on\_batch(noise, valid\_y)

# Plot the progress

print ("%d [D loss: %f, acc.: %.2f%%] [G loss: %f]" % (epoch, d\_loss[0], 100\*d\_loss[1], g\_loss))

# If at save interval => save generated image samples

if epoch % save\_interval == 0:

self.save\_imgs(epoch)

def save\_imgs(self, epoch):

r, c = 5, 5

noise = np.random.normal(0, 1, (r \* c, 100))

gen\_imgs = self.generator.predict(noise)

# Rescale images 0 - 1

gen\_imgs = 0.5 \* gen\_imgs + 0.5

fig, axs = plt.subplots(r, c)

cnt = 0

for i in range(r):

for j in range(c):

axs[i,j].imshow(gen\_imgs[cnt, :,:,0], cmap='gray')

axs[i,j].axis('off')

cnt += 1

fig.savefig("gan/images/mnist\_%d.png" % epoch)

plt.close()

if \_\_name\_\_ == '\_\_main\_\_':

gan = GAN()

gan.train(epochs=30000, batch\_size=32, save\_interval=200)