GAN: LIKE I'M FIVE

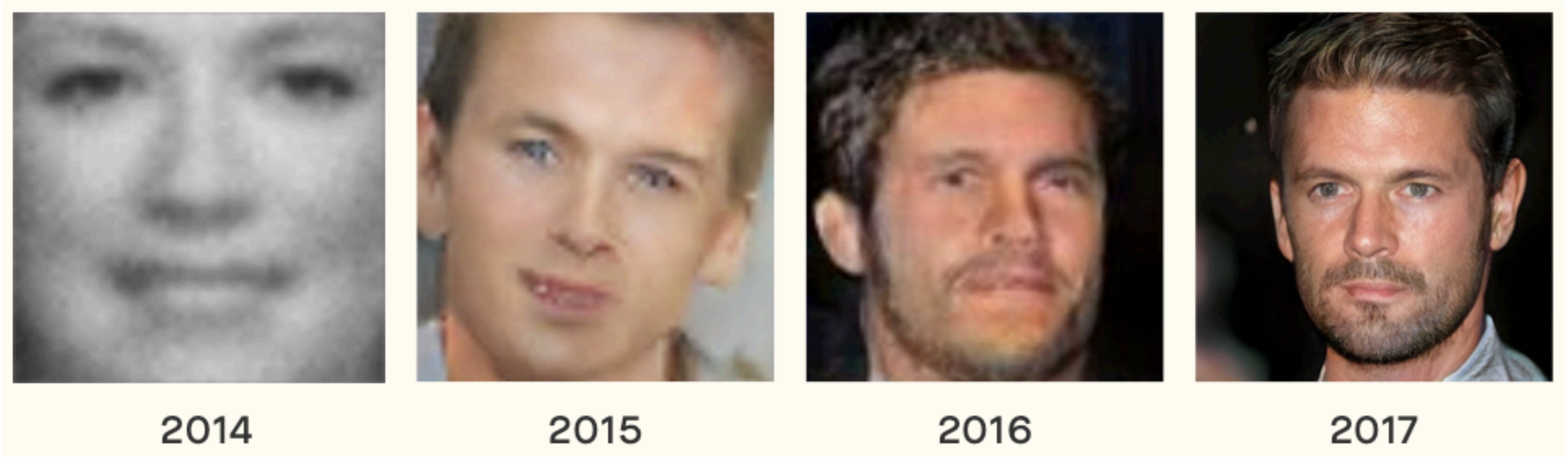
GENERATIVE ADVERSARIAL NETWORKS

GENERATIVE ADVERSARIAL NETWORKS

- 생성적 적대 신경망, 혹은 적대적 생성 신경망
- 임의의 인코딩(노이즈)이 주어졌을 때 새로운 유의미한 출력을 생성할 수 있음

GENERATIVE ADVERSARIAL NETWORKS

- Increasingly realistic synthetic faces generated by variations on Generative Adversarial Networks (GANs)



- Brundage, Miles, et al. "The malicious use of artificial intelligence: Forecasting, prevention, and mitigation." arXiv preprint arXiv:1802.07228 (2018).

BASIC OF GAN

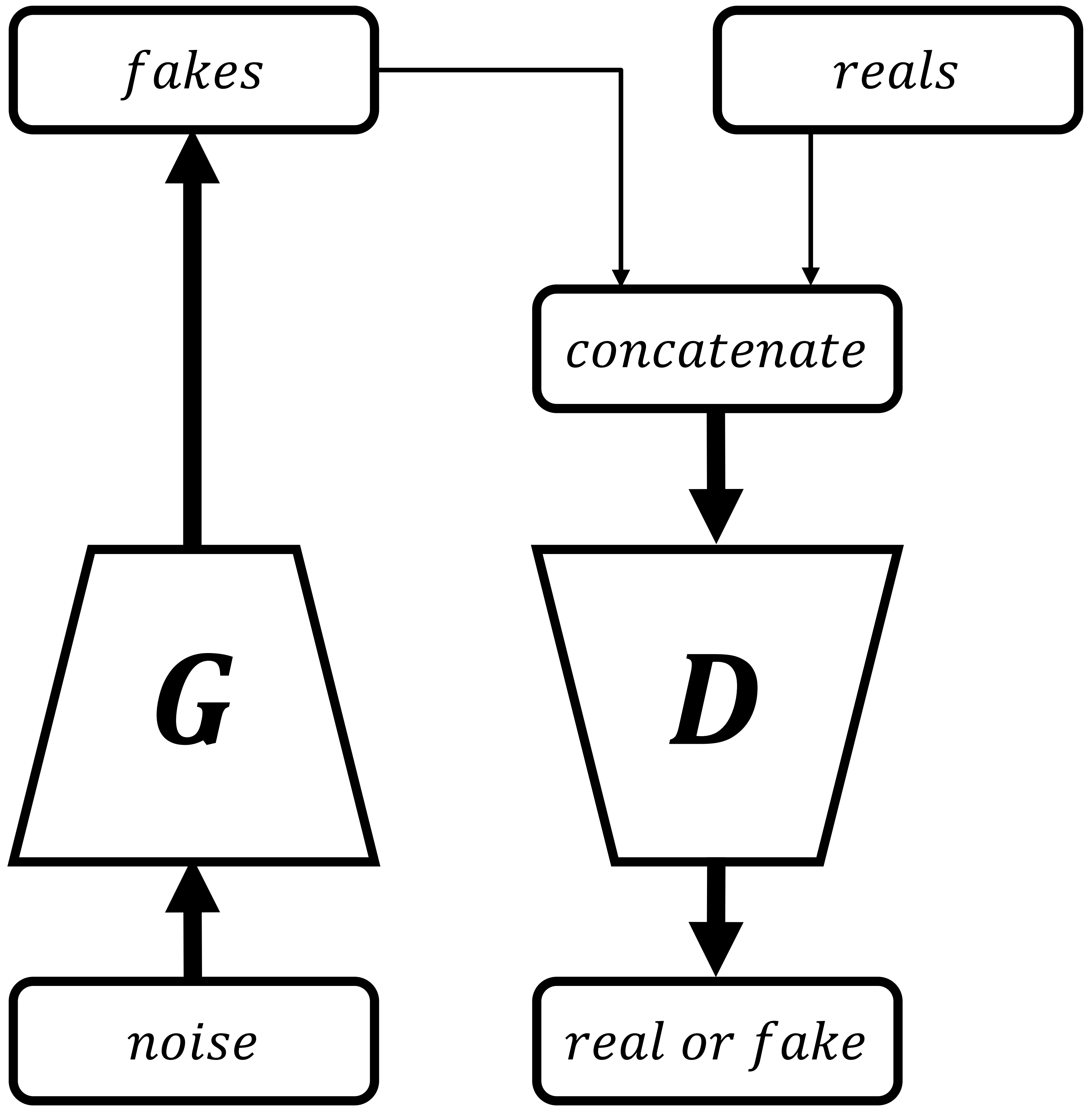
GENERATOR AND DISCRIMINATOR

- Generator: 위조지폐범
- Discriminator: 경찰
- 경찰이 위조지폐범에게 어떻게 검출했는지 알려줌 (!)
  - 서로 경쟁하면서 협업

GENERATOR AND DISCRIMINATOR

- 실제와 거의 구분되지 않는 정교한 위조지폐를 만들어내면 목표 달성
- 일반적으로 생성기를 취하고 판별기를 폐기

GENERATOR AND DISCRIMINATOR



GENERATOR AND DISCRIMINATOR

- Training GANs:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

  - Min-max game
  - Ian Goodfellow et at., “Generative Adversarial Nets”, NIPS 2014

GENERATOR AND DISCRIMINATOR

- Training GANs:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log \boxed{D_{\theta_d}(x)} + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

  - Discriminator output for real data x: 0~1
  - Discriminator wants to maximize it (1)

GENERATOR AND DISCRIMINATOR

- Training GANs:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(\boxed{G_{\theta_g}(z)})) \right]$$

  - Generated fake data from noise z
  - Generator try to fool the discriminator

GENERATOR AND DISCRIMINATOR

- Training GANs:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - \boxed{D_{\theta_d}(G_{\theta_g}(z))}) \right]$$

  - Discriminator output for fake data from z: 0~1
  - Generator wants to maximize it (1)
  - Discriminator wants to minimize it (0)

GENERATOR AND DISCRIMINATOR

- Training GANs:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log \boxed{\left(1 - D_{\theta_d}(G_{\theta_g}(z))\right)} \right]$$

  - Inverse of discriminator output for fake data from z: 0~1
  - Generator wants to minimize it (0)
  - Discriminator wants to maximize it (1)

GENERATOR AND DISCRIMINATOR

- Training GANs:

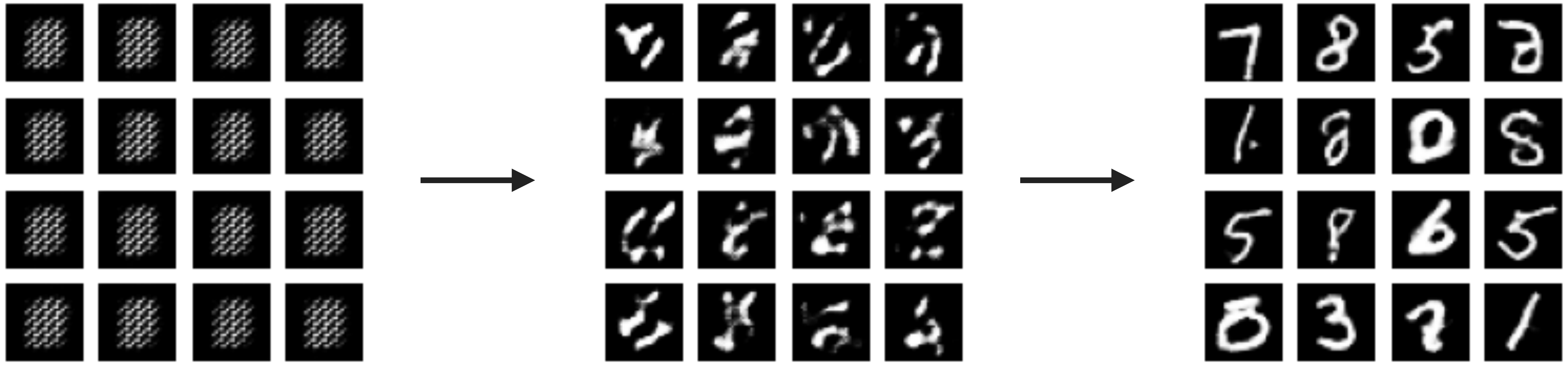
$$\min_{\theta_g} \max_{\theta_d} \left[\boxed{\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))} \right]$$

    - Discriminator wants to maximize it
    - Generator wants to minimize it
- Min-max game이라 불리는 이유임

IMPLEMENTATION

DCGAN USING KERAS ONLY

- Deep Convolution-NN GAN
- https://github.com/lukepark327/keras-only-GAN



DCGAN USING KERAS ONLY

```
# Discriminator
inputs = Input(shape=img_shape, name='discriminator_input')
discriminator = build_discriminator(inputs)
optimizer = RMSprop(lr=lr, decay=decay)
discriminator.compile(loss='binary_crossentropy', optimizer=optimizer, metrics=['accuracy'])
discriminator.summary()

# Generator
inputs = Input(shape=z_shape, name='generator_input')
generator = build_generator(inputs, img_shape[0])
generator.summary()
```

- There is no `generator.compile( )`

DCGAN USING KERAS ONLY

```
# Adversarial
discriminator.trainable = False  # Fix weights # Boolean flag at compiling

inputs = Input(shape=z_shape, name='generator_input')
adversarial = Model(inputs, discriminator(generator(inputs)), name=model_name)
optimizer = RMSprop(lr=lr * 0.5, decay=decay * 0.5)
adversarial.compile(loss='binary_crossentropy', optimizer=optimizer, metrics=['accuracy'])
adversarial.summary()
```

- Compiling `adversarial` instead of `generator`

DCGAN USING KERAS ONLY

```
x = np.concatenate([real_imgs, fake_imgs])
y = np.concatenate([np.ones([batch_size]), np.zeros([batch_size])]).reshape(-1, 1)
loss, acc = discriminator.train_on_batch(x, y)
log = "%05d: [discriminator loss: %f, acc: %f]" % (i, loss, acc)

# Train adversarial
x = np.random.uniform(-1.0, 1.0, size=[batch_size, latent_size])  # noise vectors
y = np.ones([batch_size, 1])
loss, acc = adversarial.train_on_batch(x, y)
log = "%s [adversarial loss: %f, acc: %f]" % (log, loss, acc)
print(log)
```

GAN: LIKE I'M FIVE

GENERATIVE ADVERSARIAL NETWORKS