케라스 창시자에게 배우는 딥러닝

생성 모델을 위한 딥러닝

- LSTM으로 텍스트 생성하기
- 적대적 생성 신경망 소개

한윤진 qkr030@naver.com

LSTM으로 텍스트 생성하기

- 모델을 학습하기 위한 텍스트 데이터 : 독일의 철학자 니체의 글 사용
- 모델은 데이터의 단어들만 학습되는 것이 아니라 니체의 문체와 특정 주제 또한 학습될 것

1. 데이터 불러오기

```
1 import keras
2 import numpy as np
3
4 path = keras.utils.get_file(
5     'nietzsche.txt',
6     origin='https://s3.amazonaws.com/text-datasets/nietzsche.txt')
7 text = open(path).read().lower()
8 print('말뭉치 크기:', len(text))
```

```
Downloading data from https://s3.amazonaws.com/text-datasets/nietzsche.txt
606208/600901 [==============================] - 0s 0us/step
말뭉치 크기: 600893
```

- keras.utils.get_file 함수를 사용하여 데이터셋 다운로드
- 다운로드 된 데이터셋을 읽고 소문자로 변환

LSTM으로 텍스트 생성하기

2. 데이터 전처리

```
maxlen = 60

step = 3

sentences = []

next_chars = []

for i in range(0, len(text) - maxlen, step):
    sentences.append(text[i: i + maxlen])
    next_chars.append(text[i + maxlen])
print('시퀀스 개수:', len(sentences))

chars = sorted(list(set(text)))
print('고유한 글자:', len(chars))

char_indices = dict((char, chars.index(char)) for char in chars)

print('벡터화...')
x = np.zeros((len(sentences), maxlen, len(chars)), dtype=np.bool)
y = np.zeros((len(sentences), len(chars)), dtype=np.bool)
for i, sentence in enumerate(sentences):
    for t, char in enumerate(sentence):
        x[i, t, char_indices[char]] = 1
    y[i, char_indices[next_chars[i]]] = 1
```

```
시퀀스 개수: 200278
고유한 글자: 57
벡터화...
```

① 변수 초기화

- Maxlen : 시퀀스의 글자 개수
- Step : 새로운 시퀀스를 만들기 위해 뛰어넘을 수
- Sentences : 추출한 시퀀스를 담을 리스트
- Next_chars : 타깃(시퀀스 다음 글자)를 담을 리스트

② 시퀀스 분리

- range(start, stop, step) : start부터 stop까지 step씩 뛰어 넘으며 반복
- Text[i:i+maxlen] : 0-60 / 3-63 / 6-66 · · · 60개 글자씩
- Text[i+maxlen] : 시퀀스의 다음 글자로 어떤 게 오는지

③ 중복 제거

- Set() : 집합 생성자, 중복 제거
- Char_indices : 글자와 글자의 인덱스를 매핑하여 딕셔너리에 저장

④ 벡터화

- 크기가 (sequences, maxlen, unique_characters)인 3D 넘파이 배열 x 생성
- 타깃을 담을 배열 y 생성
- x[i, t, char_indices[char]] = 1 : i번째 시퀀스의 t번째 글자가 어떤 글자인지
- Y[I, char_indices[next_chars[i]]] = 1 : i번째 시퀀스 다음 글자는 어떤 글자가 오는지

LSTM으로 텍스트 생성하기

3. 네트워크 구성

```
from keras import layers

model = keras.models.Sequential()
model.add(layers.LSTM(128, input_shape=(maxlen, len(chars))))
model.add(layers.Dense(len(chars), activation='softmax'))
```

```
optimizer = keras.optimizers.RMSprop(lr=0.01)
model.compile(loss='categorical_crossentropy', optimizer=optimizer)
```

- 하나의 LSTM 층과 Dense 분류기로 구성
- 고유한 글자 수에 대한 Softmax 출력
- 여러 개 라벨로 원-핫 인코딩 되어있기 때문에 categorical_crossentropy 손실함수 사용

LSTM으로 텍스트 생성하기

4. 언어 모델 훈련과 샘플링

샘플링 함수를 사용한 새로운 텍스트 생성

- 지금까지 생성된 텍스트를 주입하여 모델에서 다음 글자에 대한 확률 분포를 뽑음
- 특정 온도로 이 확률 분포의 가중치를 조정
- 가중치가 조정된 분포에서 무작위로 새로운 글자를 샘플링
- 새로운 글자를 생성된 텍스트의 끝에 추가

```
1 def sample(preds, temperature=1.0):
2     preds = np.asarray(preds).astype('float64')
3     preds = np.log(preds) / temperature
4     exp_preds = np.exp(preds)
5     preds = exp_preds / np.sum(exp_preds)
6     probas = np.random.multinomial(1, preds, 1)
7     return np.argmax(probas)
```

- preds = np.asarray(preds).astype('float64') : 예측 값을 numpy 배열 형태로 변환
- preds = np.log(preds) / temperature : temperature를 여러 값으로 설정하여 예측 값을 확장시킴
- softmax 적용
  - exp_preds = np.exp(preds)
  - preds = exp_preds / np.sum(exp_preds)
- probas = np.random.multinomial(1, preds, 1) : 원-핫 인코딩
- return np.argmax(probas) : 가장 큰 값의 인덱스 반환

Softmax 함수

```
1 import numpy as np
2
3 def softmax(a) :
4     exp_a = np.exp(a)
5     y = exp_a / np.sum(exp_a)
6
7     return y
8
9 a = np.array([0.3, 2.9, 4.0])
10
11 print(softmax(a)) # softmax 결과값 출력
12 print (sum(softmax(a))) # softmax 결과값들의 합은 1
```

```
[0.01821127 0.24519181 0.73659691]
1.0
```

LSTM으로 텍스트 생성하기

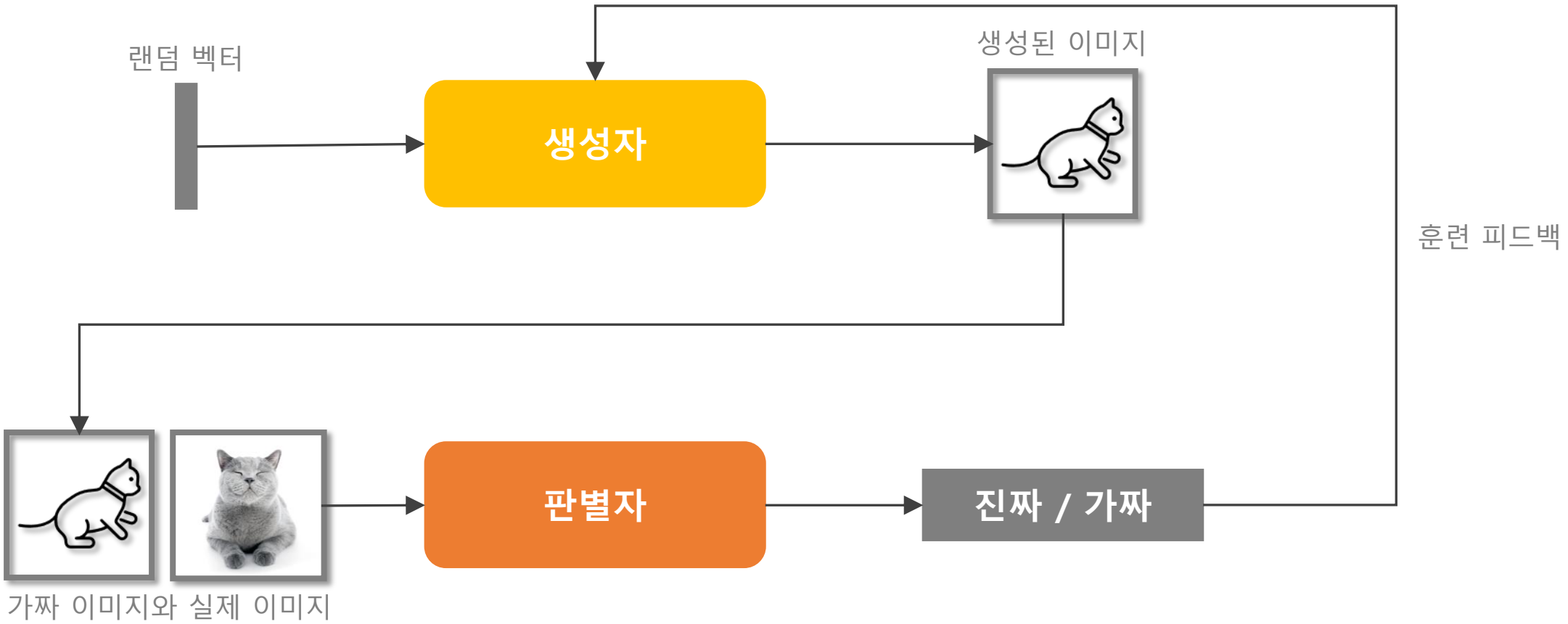
4. 언어 모델 훈련과 샘플링

```
# 60 에포크 동안 모델 훈련
for epoch in range(1, 60):
    print('에포크', epoch)
    # 데이터를 한 번만 반복해서 모델 학습
    model.fit(x, y, batch_size=128, epochs=1)

    # 무작위로 시드 텍스트 선택
    seed_text = text[start_index: start_index + maxlen]
    print('--- 시드 텍스트: "' + seed_text + '"')

    # 여러가지 샘플링 온도 시도
    for temperature in [0.2, 0.5, 1.0, 1.2]:
        print('------ 온도:', temperature)
        generated_text = seed_text
        sys.stdout.write(generated_text)

        # 시드 텍스트에서 시작해서 400개의 글자 생성
        for i in range(400):
            # 지금까지 생성된 글자를 원-핫 인코딩으로 바꿈
            sampled = np.zeros((1, maxlen, len(chars)))
            for t, char in enumerate(generated_text):
                sampled[0, t, char_indices[char]] = 1.

            # 다음 글자 샘플링
            preds = model.predict(sampled, verbose=0)[0]
            next_index = sample(preds, temperature)
            next_char = chars[next_index]

            generated_text += next_char
            generated_text = generated_text[1:]

            sys.stdout.write(next_char)
            sys.stdout.flush()
        print()
```

```
--- 시드 텍스트: "the slowly ascending ranks and classes, in which,
through fo"
------ 온도: 0.2
the slowly ascending ranks and classes, in which,
through for the resting in the still to the spirity of the resting and that is and and that has and that is intermand and that the stare and some and
and and properses that is a so and that the resting that who has and that the resting in the restination of the stares and that is and the stare of t
he restination of the will and the resting and that the senses and the resting of the stare of the spirity that
------ 온도: 0.5
the slowly ascending ranks and classes, in which,
through for the "there so for and fang, in such and so parhy can that is it is and in the something that his somewhish propes to so and the diling of
consequent procoest there
so its as a song, without ranger mistorally and dispossible and and and the them and and the remations and that is
at the sund that its has and the result. the baltical that its to the longer spirity of into the spience in the stare o
------ 온도: 1.0
the slowly ascending ranks and classes, in which,
through forming arding religed expra ever not sicquestsed
fortunce is to clarties, like bre.. the respures of somedityows, and we this was aowes an our obgines pur antidogard to
fone one lifting k,orlidity and
toerally, be fren, any posservery and -to the parly it is i abjecl than his hastistily
are thore
ame with that yay
stingls-are man why
shore proarsppens of antimaina--couls lance that was
lyader in th
------ 온도: 1.2
the slowly ascending ranks and classes, in which,
through for upstiching.


186a attenechp--much, imposic, grady
upenvairishicem. wishitm. tran meny nee be wheeghher ye re.--where recrotises,
ammors" itflence, than sthobalignc urally
it
 s rabiemve thathersterd, comethily, should sotifcofies, though an inviyousmmentlyy. bren cating
ouncull that intetrory. and
preed, thyer: tajte pessery itser europe, easming, asgaph, "order befonce inten he ang is trascer-i
```

적대적 생성 신경망(Generative Adversarial Network, GAN)

- 생성자 네트워크(Generator network) : 랜덤 벡터를 입력으로 받아 이를 이미지로 디코딩
- 판별자 네트워크(Discriminator network) : 이미지(실제 또는 합성 이미지)를 입력으로 받아 데이터셋의 이미지인지, 생성자 네트워크가 만든 이미지인지 판별
- 생성자 네트워크는 판별자 네트워크를 속이도록 훈련
- 훈련이 계속될수록 점점 더 실제와 같은 이미지를 생성하게 됨
- 실제 이미지와 구분할 수 없는 인공적인 이미지를 만들어 판별자 네트워크가 두 이미지를 동일하게 보도록 만듦
- 한편 판별자 네트워크는 생성된 이미지가 실제인지 판별하는 기준을 설정하면서 생성자의 능력 향상에 적응함



GAN 구현 방법

- 심층 합성곱 적대적 생성 신경망 구현
- CIFAR10 데이터셋 이미지 사용(32X32 크기의 RGB 이미지 5만개/10개의 클래스)
- 'frog' 클래스의 이미지만 사용하여 테스트

심층 합성곱 적대적 적대 신경망 (Deep Convolutional Generative Adversarial Networks, DCGAN)

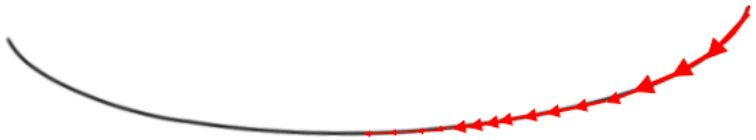
- 생성자와 판별자에 Convolution을 깊게 쌓아 더 안정적으로 학습시킬 수 있는 네트워크
- 생성자 네트워크에선 이미지 업샘플링을 위해 Conv2DTranspose 층을 사용함

GAN 구조

gan(x) = discriminator(generator(x))	
생성자(generator)	판별자(discriminator)
▪ 벡터를 (32,32,3) 크기의 이미지로 매핑 ▪ gan 모델의 경사하강법 사용하여 훈련 ▪ 매 단계마다 생성자가 디코딩한 이미지를 판별자가 '진짜'로 분류하도록 가중치 수정	▪ (32,32,3) 크기의 이미지가 진짜일 확률을 추정하여 이진 값으로 매핑 ▪ 판별자는 생성자가 벡터를 디코딩한 것이 얼마나 현실적인지 평가 ▪ 이미지는 진짜/가짜 라벨과 함께 판별자 네트워크를 훈련함

GAN 훈련 방법

- sigmoid[0~1] 대신 tanh[-1~1] 함수 사용
- 생성자에 입력될 샘플 벡터로 정규 분포(가우시안 분포)를 사용하여 추출한 값을 사용



- 약한 그래디언트는 GAN 훈련에 좋지 않음
  - ✓ 최대 풀링과 ReLU 활성화는 그래디언트를 약하게 만듦
  - ✓ 최대 풀링 대신 스트라이드 합성곱을 사용하여 다운샘플링
  - ✓ ReLU 활성화 대신 LeakyReLU 층을 사용 : ReLU와 비슷하지만 음수의 활성화 값을 조금 허용하기 때문에 희소가 조금 완화

생성자 네트워크

- 무작위로 샘플링된 벡터를 이미지로 변환하는 생성자 모델

```
latent_dim = 32
height = 32
width = 32
channels = 3

generator_input = keras.Input(shape=(latent_dim,))

# 입력을 16 × 16 크기의 128개 채널을 가진 특성 맵으로 변환
x = layers.Dense(128 * 16 * 16)(generator_input)
x = layers.LeakyReLU()(x)
x = layers.Reshape((16, 16, 128))(x)

# 합성곱 층을 추가
x = layers.Conv2D(256, 5, padding='same')(x)
x = layers.LeakyReLU()(x)

# 32 × 32 크기로 업샘플링
x = layers.Conv2DTranspose(256, 4, strides=2, padding='same')(x)
x = layers.LeakyReLU()(x)

# 합성곱 층을 더 추가
x = layers.Conv2D(256, 5, padding='same')(x)
x = layers.LeakyReLU()(x)
x = layers.Conv2D(256, 5, padding='same')(x)
x = layers.LeakyReLU()(x)

# 32 × 32 크기의 1개 채널을 가진 특성 맵 생성
x = layers.Conv2D(channels, 7, activation='tanh', padding='same')(x)
generator = keras.models.Model(generator_input, x)
generator.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              
=================================================================
input_1 (InputLayer)         (None, 32)                
_________________________________________________________________
dense_1 (Dense)              (None, 32768)             
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 32768)             
_________________________________________________________________
reshape_1 (Reshape)          (None, 16, 16, 128)       
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 16, 16, 256)       
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 16, 16, 256)       
_________________________________________________________________
conv2d_transpose_1 (Conv2DTr (None, 32, 32, 256)       
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 32, 32, 256)       
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 32, 32, 256)       
_________________________________________________________________
leaky_re_lu_4 (LeakyReLU)    (None, 32, 32, 256)       
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 32, 32, 256)       
_________________________________________________________________
leaky_re_lu_5 (LeakyReLU)    (None, 32, 32, 256)       
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 32, 32, 3)         
=================================================================
```

판별자 네트워크

- 이미지를 입력으로 받고 두 개의 클래스로 분류하는 판별자 모델

```
discriminator_input = layers.Input(shape=(height, width, channels))
x = layers.Conv2D(128, 3)(discriminator_input)
x = layers.LeakyReLU()(x)
x = layers.Conv2D(128, 4, strides=2)(x)
x = layers.LeakyReLU()(x)
x = layers.Conv2D(128, 4, strides=2)(x)
x = layers.LeakyReLU()(x)
x = layers.Conv2D(128, 4, strides=2)(x)
x = layers.LeakyReLU()(x)
x = layers.Flatten()(x)

# 드롭아웃 층을 넣는 것이 아주 중요
x = layers.Dropout(0.4)(x)

# 분류 층
x = layers.Dense(1, activation='sigmoid')(x)

discriminator = keras.models.Model(discriminator_input, x)
discriminator.summary()

# 옵티마이저에서 (값을 지정하여) 그래디언트 클리핑을 사용
# 안정된 훈련을 위해서 학습률 감쇠를 사용
discriminator_optimizer = keras.optimizers.RMSprop(lr=0.0008, clipvalue=1.0, decay=1e-8)
discriminator.compile(optimizer=discriminator_optimizer, loss='binary_crossentropy')
```

```
_________________________________________________________________
Layer (type)                 Output Shape              
=================================================================
input_2 (InputLayer)         (None, 32, 32, 3)         
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 30, 30, 128)       
_________________________________________________________________
leaky_re_lu_6 (LeakyReLU)    (None, 30, 30, 128)       
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 14, 14, 128)       
_________________________________________________________________
leaky_re_lu_7 (LeakyReLU)    (None, 14, 14, 128)       
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 6, 6, 128)         
_________________________________________________________________
leaky_re_lu_8 (LeakyReLU)    (None, 6, 6, 128)         
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 2, 2, 128)         
_________________________________________________________________
leaky_re_lu_9 (LeakyReLU)    (None, 2, 2, 128)         
_________________________________________________________________
flatten_1 (Flatten)          (None, 512)               
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 
=================================================================
```

적대적 네트워크

- 생성자와 판별자를 연결하여 GAN 설정
- 생성자가 판별자를 속이는 능력이 커지도록 학습

```
# 판별자의 가중치가 훈련되지 않도록 설정(gan 모델에만 적용)
discriminator.trainable = False

gan_input = keras.Input(shape=(latent_dim,))
gan_output = discriminator(generator(gan_input))
gan = keras.models.Model(gan_input, gan_output)

gan_optimizer = keras.optimizers.RMSprop(lr=0.0004, clipvalue=1.0, decay=1e-8)
gan.compile(optimizer=gan_optimizer, loss='binary_crossentropy')
```

GAN 학습 단계

I. 랜덤 벡터 생성
II. 랜덤 벡터를 사용하여 생성자 네트워크가 이미지 생성
III. 생성된 이미지와 진짜 이미지를 섞음
IV. 진짜와 가짜가 섞인 이미지를 사용하여 판별자 학습
V. 랜덤 벡터 생성
VI. 랜덤 벡터 전부 '진짜 이미지'로 라벨링하고 GAN 학습
VII. 판별자가 이미지를 전부 '진짜 이미지'로 예측하도록 생성자 가중치 업데이트

적대적 네트워크 학습

1. 데이터 불러오기

```
# CIFAR10 데이터를 로드
(x_train, y_train), (_, _) = keras.datasets.cifar10.load_data()

# 개구리 이미지 선택(클래스 6)
x_train = x_train[y_train.flatten() == 6]

# 데이터 정규화
x_train = x_train.reshape(
    (x_train.shape[0],) + (height, width, channels)).astype('float32') / 255.

iterations = 10000
batch_size = 20
save_dir = './datasets/gan_images/'
if not os.path.exists(save_dir):
    os.mkdir(save_dir)
```

- Cifar10 데이터셋 다운로드
- 개구리 이미지만 사용
- 데이터는 (이미지 개수, 세로 크기, 가로 크기, 채널) 형태로 정규화됨
- 10000번 반복
- 생성자 네트워크가 생성한 이미지를 저장시킬 Save_dir 디렉토리 생성

적대적 네트워크 학습

2. 학습

- 진짜 이미지와 생성자가 만들어낸 이미지로 판별자 학습

```
# 훈련 반복 시작
start = 0
for step in range(iterations):
    # 잠재 공간에서 무작위로 포인트를 샘플링
    random_latent_vectors = np.random.normal(size=(batch_size, latent_dim))

    # 가짜 이미지를 디코딩
    generated_images = generator.predict(random_latent_vectors)

    # 진짜 이미지와 연결
    stop = start + batch_size
    real_images = x_train[start: stop]
    combined_images = np.concatenate([generated_images, real_images])

    # 진짜와 가짜 이미지를 구분하여 레이블을 합침
    labels = np.concatenate([np.ones((batch_size, 1)),
                             np.zeros((batch_size, 1))])
    # 레이블에 랜덤 노이즈를 추가
    labels += 0.05 * np.random.random(labels.shape)

    # discriminator를 훈련
    d_loss = discriminator.train_on_batch(combined_images, labels)
```

적대적 네트워크 학습

2. 학습

- 생성자가 만들어낸 이미지를 '진짜 이미지' 라벨링하여 GAN 학습(생성자만 가중치 업데이트)

```
    # 잠재 공간에서 무작위로 포인트를 샘플링
    random_latent_vectors = np.random.normal(size=(batch_size, latent_dim))

    # 모두 "진짜 이미지"라고 레이블 생성
    misleading_targets = np.zeros((batch_size, 1))

    # generator를 훈련(gan 모델에서 discriminator의 가중치는 동결)
    a_loss = gan.train_on_batch(random_latent_vectors, misleading_targets)
```

3. 저장

- 모델 가중치 저장 및 생성된 이미지/비교할 진짜 이미지 저장

```
    # 중간 중간 저장하고 그래프를 그립니다
    if step % 100 == 0:
        # 모델 가중치를 저장합니다
        gan.save_weights('gan.h5')

        # 측정 지표를 출력합니다
        print('스텝 %s에서 판별자 손실: %s' % (step, d_loss))
        print('스텝 %s에서 적대적 손실: %s' % (step, a_loss))

        # 생성된 이미지 하나를 저장합니다
        img = image.array_to_img(generated_images[0] * 255., scale=False)
        img.save(os.path.join(save_dir, 'generated_frog' + str(step) + '.png'))

        # 비교를 위해 진짜 이미지 하나를 저장합니다
        img = image.array_to_img(real_images[0] * 255., scale=False)
        img.save(os.path.join(save_dir, 'real_frog' + str(step) + '.png'))
```

생성자 네트워크가 생성한 가짜 이미지