7, 8장 발표

이상윤, 김태우, 송선영

Start

목차

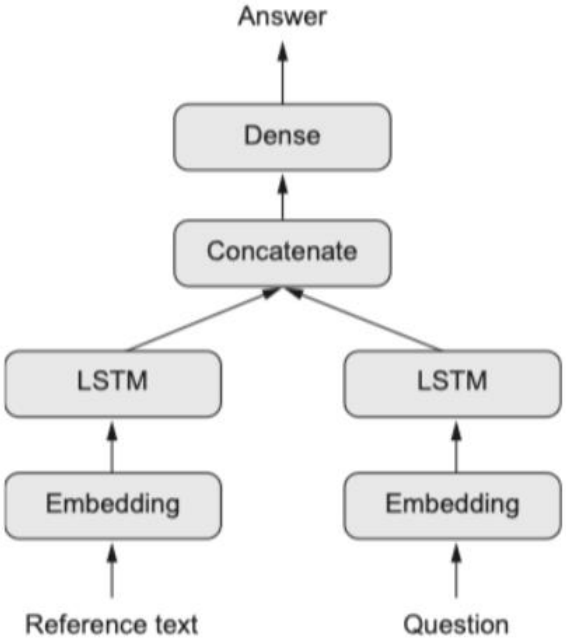
7.1 Sequential 모델을 넘어서: 케라스의 함수형 API

1) 함수형 API 소개
- 함수처럼 층을 사용하여 직접 텐서를 입력받고 출력한다.

```
# Sequential
seq_model = Sequential()
seq_model.add(layers.Dense(32, activation='relu', input_shape=(64,)))
seq_model.add(layers.Dense(32, activation='relu'))
seq_model.add(layers.Dense(10, activation='softmax'))

# 함수형 API
input_tensor = Input(shape=(64,))   # 텐서
x = layers.Dense(32, activation='relu')(input_tensor) # 함수처럼 사용하기 위해 층 객체를 만듦
x = layers.Dense(32, activation='relu')(x)
output_tensor = layers.Dense(10, activation='softmax')(x)

model = Model(input_tensor, output_tensor) # 입력과 출력 텐서를 지정해 model 클래스의 객체를 만듦
```

7.1 Sequential 모델을 넘어서: 케라스의 함수형 API

2) 다중 입력 모델

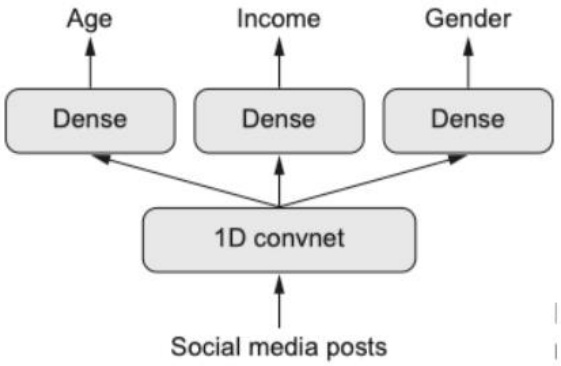


* **keras.layers.add / keras.layers.concatenate**
  -> 텐서를 더하거나 이어붙일때 사용하는 함수
  -> 그 외에 layers.average(), layers.maximum(), layers.multiply(), layers.subtract(), layers.dot() 등이 있음

```
# 리스트 입력을 사용하여 학습
model.fit([text, question], answers, epochs=10, batch_size=128)

# # 입력에 이름(name)을 지정했을 때만 사용가능
model.fit({'text': text, 'question': question}, answers,
          epochs=10, batch_size=128)
```

7.1 Sequential 모델을 넘어서: 케라스의 함수형 API

3) 다중 출력 모델




```
vocabulary_size = 50000
num_income_groups = 10

posts_input = Input(shape=(None,), dtype='int32', name='posts')
embedded_posts = layers.Embedding(vocabulary_size, 256)(posts_input)
x = layers.Conv1D(128, 5, activation='relu')(embedded_posts)
x = layers.MaxPooling1D(5)(x)
x = layers.Conv1D(256, 5, activation='relu')(x)
x = layers.Conv1D(256, 5, activation='relu')(x)
x = layers.MaxPooling1D(5)(x)
x = layers.Conv1D(256, 5, activation='relu')(x)
x = layers.Conv1D(256, 5, activation='relu')(x)
x = layers.GlobalMaxPooling1D()(x)
x = layers.Dense(128, activation='relu')(x)

age_prediction = layers.Dense(1, name='age')(x) # 출력 층에 이름을 지정
income_prediction = layers.Dense(num_income_groups,
                                 activation='softmax',
                                 name='income')(x) # 출력 층에 소득수준을 지정
gender_prediction = layers.Dense(1, activation='sigmoid', name='gender')(x) # 출력 층에 성별을 지정

model = Model(posts_input,
              [age_prediction, income_prediction, gender_prediction])
```

7.1 Sequential 모델을 넘어서: 케라스의 함수형 API

3) 다중 출력 모델

```
# 리스트 형태
model.compile(optimizer='rmsprop',
              loss=['mse', 'categorical_crossentropy', 'binary_crossentropy'])

# 딕셔너리 형태: 출력 층에 이름을 지정해야함
model.compile(optimizer='rmsprop',
              loss={'age': 'mse',
                    'income': 'categorical_crossentropy',
                    'gender': 'binary_crossentropy'})          # 다중 손실
```

```
# 리스트 형태
model.compile(optimizer='rmsprop',
              loss=['mse', 'categorical_crossentropy', 'binary_crossentropy'],
              loss_weights=[0.25, 1., 10])

# 딕셔너리 형태: 출력 층에 이름을 지정해야함
model.compile(optimizer='rmsprop',
              loss={'age': 'mse',
                    'income': 'categorical_crossentropy',
                    'gender': 'binary_crossentropy'},
              loss_weights={'age': 0.25,
                            'income': 1.,
                            'gender': 10.})          # 손실 가중치 지정
```

age: 스칼라 회귀 문제 -> mse
income: 다중 클래스 문제 -> binary_crossentropy
gender: 이진 클래스 문제 -> categorical_crossentropy

- 각 출력에 대한 손실들을 **하나의 값으로 합쳐**줘야한다.

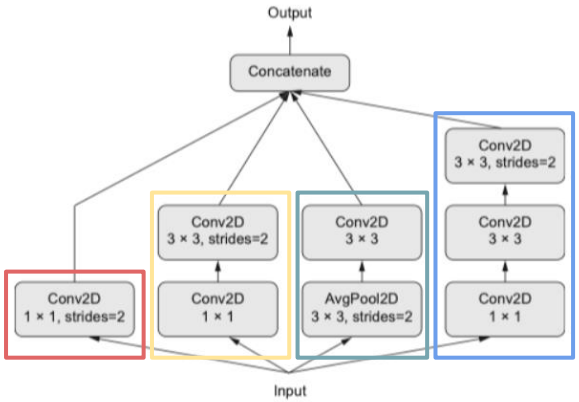
  -> 이럴 경우 손실값이 불균형하면, 개별 손실이 가장 큰 작업에 치우칠 수 있음

→ **손실 가중치 지정**
  - 손실 값의 스케일이 다를 때 유용

7.1 Sequential 모델을 넘어서: 케라스의 함수형 API

4-1) 인셉션 모듈

- 가장 기본적인 형태는 3~4개의 가지를 가짐
- 네트워크가 따로따로 공간 특성과 채널 방향의 특성을 학습하도록 도움



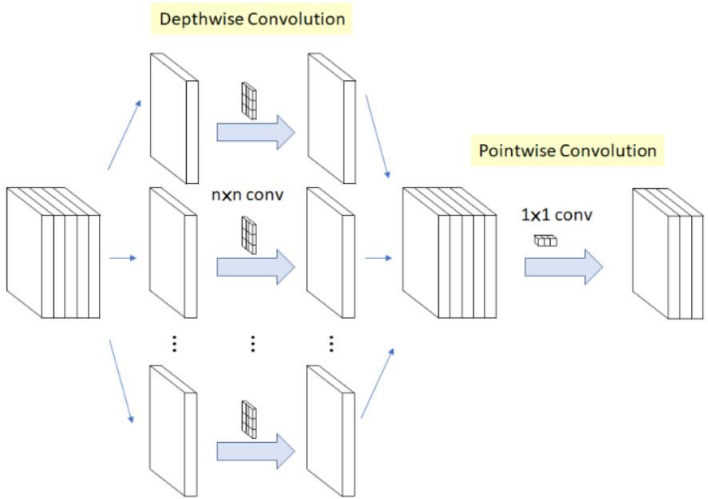
인셉션 V3

```
branch_a = layers.Conv2D(128, 1, padding='same',
                         activation='relu', strides=2)(inputs)

branch_b = layers.Conv2D(128, 1, padding='same',
                         activation='relu')(inputs)
branch_b = layers.Conv2D(128, 3, padding='same',
                         activation='relu', strides=2)(branch_b)

branch_c = layers.AveragePooling2D(3, strides=2, padding='same')(inputs)
branch_c = layers.Conv2D(128, 3, padding='same', activation='relu')(branch_c)

branch_d = layers.Conv2D(128, 1, padding='same', activation='relu')(inputs)
branch_d = layers.Conv2D(128, 3, padding='same', activation='relu')(branch_d)
branch_d = layers.Conv2D(128, 3, padding='same', activation='relu', strides=2)(branch_d)
```

- padding='same' 추가하여 출력 크기 더 정확하게 맞출 수 있음

7.1 Sequential 모델을 넘어서: 케라스의 함수형 API

4-2) **엑셉션**



깊이별 분리 합성곱

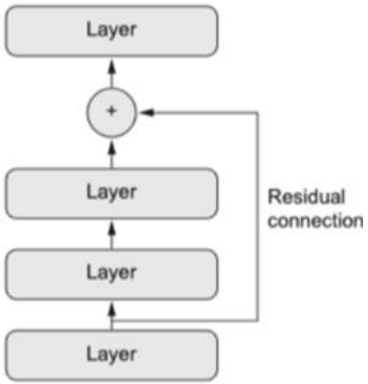
- **극단적인 인셉션**을 의미

- 인셉션 모듈을 깊이별 분리 합성곱으로 바꿈
  -> 그런 다음, 1x1 합성곱을 적용
  -> **공간 특성과 채널 방향 특성을 완전히 분리함**

- 인셉션 V3와 거의 동일한 개수의 모델 파라미터를 가짐
- 인셉션 V3보다 실행 속도가 더 빠르고 대규모 데이터셋(ex. ImageNet)에서 정확도가 더 높음
  -> 모델 파라미터를 더 효율적으로 사용하기 때문

7.1 Sequential 모델을 넘어서: 케라스의 함수형 API

4-3) 잔차 연결
- 그래디언트 소실과 표현 병목 문제 해결
- 10개 층 이상을 가진 모델에 잔차 연결을 추가하면 도움됨



Residual

- 하위 층의 출력을 상위 층의 출력에 더해서 상위 층의 입력으로 사용함
  -> 두 출력의 크기가 동일해야 함
- 하위 층에서 학습된 정보가 데이터 처리 과정에서 손실되는 것을 방지합니다.

7.1 Sequential 모델을 넘어서: 케라스의 함수형 API

5) 층 가중치 공유
- 함수형 API는 층 객체를 여러 번 재사용할 수 있음
  -> 공유 가지를 가진 모델을 만들 수 있음
- **같은 표현을 공유하고 이런 표현을 다른 입력에서 함께 학습함**

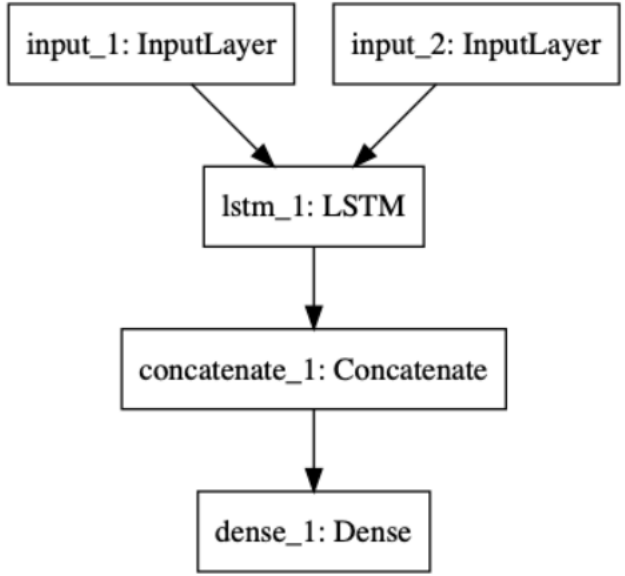
Ex. 두 문장 사이의 의미가 비슷한지 측정하는 모델
- A,B 2개의 문장을 입력받아 0과 1 사이의 점수를 출력함
  (0: 관련없는 문장, 1: 두 문장이 동일하거나 재구성됨)

-> A에서 B에 대한 유사도가 B에서 A에 대한 유사도와 같음
-> 각 입력 문장을 처리하는 2개의 독립된 모델을 학습하는 것 보다는 하나의 LSTM층으로
양쪽 모두를 처리하는 것이 좋음

-> **샴 LSTM 모델** or **공유 LSTM** 이라고 부름

7.1 Sequential 모델을 넘어서: 케라스의 함수형 API

```
# LSTM층 객체를 하나 만듦
lstm = layers.LSTM(32)

# Input: 크기가 128인 벡터의 가변길이 시퀀스
left_input = Input(shape=(None, 128)) # 모델의 왼쪽 가지
left_output = lstm(left_input)

right_input = Input(shape=(None, 128)) # 모델의 오른쪽 가지
right_output = lstm(right_input)

# Classifier
merged = layers.concatenate([left_output, right_output], axis=-1)
prediction = layers.Dense(1, activation='sigmoid')(merged)

# 모델 객체를 만들고 훈련함
model = Model([left_input, right_input], prediction)
model.fit([left_data, right_data], targets)
```



7.1 Sequential 모델을 넘어서: 케라스의 함수형 API

6) 층과 모델
- 함수형 API에서는 모델을 층처럼 사용할 수 있음 (모델을 '커다란 층'으로 생각)
- 모델 객체를 호출할때 모델의 가중치가 재사용됨

```
# 입력 텐서와 출력텐서가 여러개면 텐서의 리스트로 호출
y1, y2 = model([x1, x2])
```

7.2 케라스 콜백과 텐서보드를 사용한 딥러닝 모델 검사와 모니터링

1) 콜백을 사용하여 모델의 훈련 과정 제어하기
   - keras의 callback을 사용해 검증 손실이 더 이상 향상되지 않을때 훈련을 멈추는 것 구현 가능

   - **콜백:** 모델의 fit()메서드가 호출될 때 전달되는 객체
     -> 훈련하는 동안 모델은 여러 지점에서 콜백을 호출함
     -> 콜백은 모델의 상태와 성능에 대한 모든 정보에 접근하고
     훈련 중지, 모델 저장, 가중치 적재 또는 모델 상태 변경 등을 처리할 수 있음

   - 콜백을 사용하는 예시
     - 모델 체크포인트 저장
     - 조기 종료
     - 훈련하는 동안 하이퍼 파라미터 값을 동적으로 조정
     - 훈련과 검증 지표를 로그에 기록하거나 모델이 학습한 표현이 업데이트될 때마다 시각화

7.2 케라스 콜백과 텐서보드를 사용한 딥러닝 모델 검사와 모니터링

1-1) ModelCheckpoint와 EarlyStopping 콜백

```
import keras

# fit() 메서드의 callbacks 매개변수를 사용하여
# callbacks_list를 모델로 전달한다. 몇 개의 콜백이라도 전달할 수 있다.
callbacks_list = [
    keras.callbacks.EarlyStopping(  # 성능 향상이 멈추면 훈련을 중지한다.
        monitor='val_acc',  # 모델의 검증 정확도를 모니터링 한다. default는 'val_loss'
        patience=1  # 1 epoch보다 더 길게 (즉 2 epoch 동안) 정확도가 향상되지 않으면 훈련이 중지된다.
    ),
    keras.callbacks.ModelCheckpoint(  # epoch 마다 현재 가중치를 저장한다.
        filepath='./model/my_model.h5',  # 모델 파일의 경로
        monitor='val_loss',  # 'val_loss'가 좋아지지 않으면 모델 파일을 덮어쓰지 않는다는 뜻
        save_best_only=True  # 훈련하는 동안 가장 좋은 모델만 저장
    )
]

model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['acc']) # 정확도를 모니터링하므로 모델 지표에 포함되어야 함

model.fit(x, y,
          epochs=10,
          batch_size=32,
          callbacks=callbacks_list,# 콜백이 검증 손실과 검증 정확도를 모니터링하기 때문에
          validation_data=(x_val, y_val)) # validation_data 매기변수에 검증 데이터를 전달해야 함
```

1-2) ReduceLROnPlateau 콜백

```
callbacks_list = [
    keras.callbacks.ReduceLROnPlateau(
        monitor='val_loss',  # 모델의 검증 손실을 모니터링 한다.
        factor=0.1,  # 콜백이 호출될 때 학습률을 10배로 줄임
        patience=10,  # 검증 손실이 10 epoch동안 좋아지지 않으면 콜백이 호출됨
    )
]

model.fit(x, y,
          epochs=10,
          batch_size=32,
          callbacks=callbacks_list, # 콜백이 검증 손실과 검증 정확도를 모니터링하기 때문에
          validation_data=(x_val, y_val)) # validation_data 매기변수에 검증 데이터를 전달해야 함
```

1-3) 자신만의 콜백 만들기

on_epoch_begin : 각 epoch이 시작할 때 호출
on_epoch_end : 각 epoch이 끝날 때 호출
on_batch_begin : 각 배치 처리가 시작되기 전에 호출
on_batch_end : 각 배치 처리가 끝난 후에 호출
on_train_begin : 훈련이 시작될 때 호출
on_train_end : 훈련이 끝날 때 호출

7.2 케라스 콜백과 텐서보드를 사용한 딥러닝 모델 검사와 모니터링

2) 텐서보드
- 텐서플로의 시각화 프레임워크

- 텐서보드의 기능
  - 훈련하는 동안 측정 지표를 시각적으로 모니터링함
  - 모델 구조를 시각화함
  - 활성화 출력과 그래디언트의 히스토그램을 그림
  - 3D로 임베딩을 표현함

7.3 모델의 성능을 최대로 끌어올리기

1) 고급 구조 패턴

1. 배치 정규화
   - 정규화: ML 모델에 주입되는 샘플들을 균일하게 만드는 광범위한 방법

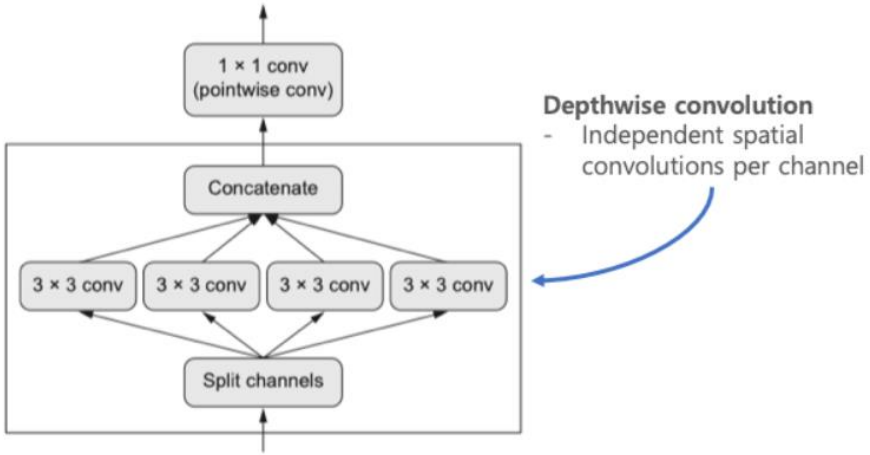
$$N(\mu, \sigma^2) \rightarrow N(0, 1)$$

```
normalized_data = (data - np.mean(data, axis=...)) / np.std(data, axis=...)
```

   - 배치 정규화: 훈련하는 동안 평균과 분산이 바뀌더라도 이에 적응해 데이터를 정규화

7.3 모델의 성능을 최대로 끌어올리기

2. 깊이별 분리 합성곱
   - Conv2D를 대체하면서 더 가볍고 더 빠른 합성곱 층
   - 입력 채널별로 따로따로 공간 방향의 합성곱을 수행함
   - 공간 특성의 학습과 채널 방향 특성의 학습을 분리




```
from keras.models import Sequential, Model
from keras import layers

height = 64
width = 64
channels = 3
num_classes = 10

model = Sequential()
model.add(layers.SeparableConv2D(32, 3,
                                 activation='relu',
                                 input_shape=(height, width, cannels,)))
model.add(layers.SeparableConv2D(64, 3, activation='relu'))
model.add(layers.MaxPooling2D(2))

model.add(layers.SeparableConv2D(64, 3, activation='relu'))
model.add(layers.SeparableConv2D(128, 3, activation='relu'))
model.add(layers.MaxPooling2D(2))

model.add(layers.SeparableConv2D(64, 3, activation='relu'))
model.add(layers.SeparableConv2D(128, 3, activation='relu'))
model.add(layers.GlobalAveragePooling2D())

model.add(layers.Dense(32, activation='relu'))
model.add(layers.Dencs(num_classes, activation='softmax'))

model.compile(optimizer='rmsprop', loss='categorical_crossentropy')
```

7.3 모델의 성능을 최대로 끌어올리기

2) 하이퍼파라미터 최적화
   - 과정
     1. 일련의 하이퍼파라미터를 자동으로 선택
     2. 선택된 하이퍼파라미터로 모델 생성
     3. 훈련 데이터에 학습하고 검증 데이터에서 최종 성능 측정
     4. 다음으로 시도할 하이퍼파라미터를 자동으로 선택
     5. 이 과정 반복
     6. 테스트 데이터에서 성능 측정

   - 하이퍼파라미터를 선택하는 알고리즘: 베이지안 최적화, 유전 알고리즘, 간단한 랜덤 탐색

3) 모델 앙상블
   - 여러 개 다른 모델의 예측을 합쳐서 더 좋은 예측을 만드는 기법
   - **모델의 다양성**이 가장 중요

     1. 예측을 평균내는 것
     2. 가중 평균하는 것

7.3 모델의 성능을 최대로 끌어올리기

3) 모델 앙상블
- 여러 개 다른 모델의 예측을 합쳐서 더 좋은 예측을 만드는 기법
- **모델의 다양성**이 가장 중요

- 앙상블 방법
  1. 예측을 평균내는 것
  2. 가중 평균하는 것

```
preds_a = model_a.predict(val_x)
preds_b = model_b.predict(val_x)
preds_c = model_c.predict(val_x)
preds_d = model_d.predict(val_x)

# 예측 평균
final_preds = 0.25 * (preds_a + preds_b + preds_c + preds_d)

# 가중 평균
final_preds = 0.5 * preds_a + 0.25 * preds_b + 0.1 * preds_c + 0.15 * preds_d
```

8.1 LSTM으로 텍스트 생성하기

- 순환 신경망으로 시퀀스 데이터를 생성하는 방법

  이전 토큰을 입력으로 사용하여 시퀀스의 **다음 1개 또는 몇 개의 토큰을 예측**하는 것.

- 시퀀스 데이터의 종류

  음표의 시퀀스로 새로운 음악을 만들거나, 붓질 시퀀스에 적용하여 새로운 그림을 그리거나 대화, 이미지, 음성, 분자 등

- 샘플링 과정에서 확률의 양을 조절하기 위해 **소프트맥스 온도** 라는 파라미터 사용.

  온도가 낮으면 무작위성이 작고, 높으면 무작위성이 커진다.

8.1 LSTM으로 텍스트 생성하기

- 니체의 글로 실습해보기 -> 일반적인 영어 모델이 아닌 니체의 문체와 특정 주제를 따르는 모델

1. 데이터 전처리
1. 네트워크 구성
1. 언어 모델 훈련과 샘플링

8.1 LSTM으로 텍스트 생성하기

1. 데이터 전처리

말뭉치를 내려받고 소문자로 바꿔준다.

In [2]:
```
import keras
import numpy as np

path = keras.utils.get_file(
    'nietzsche.txt',
    origin='https://s3.amazonaws.com/text-datasets/nietzsche.txt')
text = open(path).read().lower()
print('말뭉치 크기:', len(text))
```

```
말뭉치 크기: 600893
```

In [3]:
```
type(text)
```

Out[3]:
```
str
```

8.1 LSTM으로 텍스트 생성하기

1. 데이터 전처리

- 글자 시퀀스 벡터화하기

60개의 글자로 된 시퀀스를 추출, 3글자 씩 건너뛰며 새로운 시퀀스를 샘플링한다.

Chars 에는 text를 set로 만들어 고유한 글자만 담아 정렬된 리스트로 저장한다.

이 후 고유한 글자와 인덱스로 매핑한 딕셔너리를 만들고 벡터화한다.

In [4]:

```
# 60개 글자로 된 시퀀스를 추출합니다.
maxlen = 60

# 세 글자씩 건너 뛰면서 새로운 시퀀스를 샘플링합니다.
step = 3

# 추출한 시퀀스를 담을 리스트
sentences = []

# 타깃(시퀀스 다음 글자)을 담을 리스트
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))

# 말뭉치에서 고유한 글자를 담은 리스트
chars = sorted(list(set(text)))
print('고유한 글자:', len(chars))
# chars 리스트에 있는 글자와 글자의 인덱스를 매핑한 딕셔너리
char_indices = dict((char, chars.index(char)) for char in chars)

# 글자를 원-핫 인코딩하여 0과 1의 이진 배열로 바꿉니다.
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...
```

8.1 LSTM으로 텍스트 생성하기

2. 네트워크 구성

하나의 LSTM 층과 Dense 분류기로 구성

분류기는 가능한 모든 글자에 대한 소프트맥스 출력을 만든다.

타깃이 원-핫 인코딩되어 있기에 훈련을 위해 categorical_crossentropy 를 loss로 사용하는 모습

In [5]:
```
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'))
```

In [6]:
```
optimizer = keras.optimizers.RMSprop(lr=0.01)
model.compile(loss='categorical_crossentropy', optimizer=optimizer)
```

8.1 LSTM으로 텍스트 생성하기

3. 언어 모델 훈련과 샘플링

1. 텍스트를 주입하여 모델에서 다음 글자에 대한 확률 분포를 뽑는다.
2. 특정 온도로 확률 분포의 가중치를 조정
3. 가중치가 조정된 분포에서 무작위로 새로운 글자를 샘플링
4. 새로운 글자를 생성된 텍스트의 끝에 추가

In [7]:
```
def sample(preds, temperature=1.0):
    preds = np.asarray(preds).astype('float64')
    preds = np.log(preds) / temperature
    exp_preds = np.exp(preds)
    preds = exp_preds / np.sum(exp_preds)
    probas = np.random.multinomial(1, preds, 1)
    return np.argmax(probas)
```

- 모델의 예측이 주어졌을 때 새로운 글자를 샘플링하는 함수
  예측값을 log화 하여 온도로 나누고, exp로 자연상수의 지수함수 형태로 만든다.

8.1 LSTM으로 텍스트 생성하기

3. 언어 모델 훈련과 샘플링

각 에포크마다 학습 후 0.2, 0.5, 1.0, 1.2의 온도를 사용하여 텍스트를 생성

총 60 에포트 * 4개 온도로 240개의 출력이 나온다.

In [8]:

```
import random
import sys

random.seed(42)
start_index = random.randint(0, len(text) - maxlen - 1)

# 60 에포크 동안 모델을 훈련합니다
for epoch in range(1, 60):
    print('에포크', epoch)
    # 데이터에서 한 번만 반복해서 모델을 학습합니다
    model.fit(x, y, batch_size=128, epochs=1)

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

    # 여러가지 샘플링 온도를 시도합니다
    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 글자를 생성합니다
        for i in range(400):
            # 지금까지 생성된 글자를 원-핫 인코딩으로 바꿉니다
            sampled = np.zeros((1, maxlen, len(chars)))
            for t, char in enumerate(generated_text):
                sampled[0, t, char_indices[char]] = 1.

            # 다음 글자를 샘플링합니다
            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()
```

8.1 LSTM으로 텍스트 생성하기

3. 언어 모델 훈련과 샘플링

```
에포크 1
Epoch 1/1
200278/200278 [==============================] - 78s 390us/step - loss: 1.9812
--- 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 a
nd 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 the 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 sthobaligncurally
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
```

8.1 LSTM으로 텍스트 생성하기

3. 언어 모델 훈련과 샘플링

```
에포크 20
Epoch 1/1
200278/200278 [==============================] - 79s 395us/step - loss: 1.3370
--- 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 same taste of a man is the sense of the problem of the sense of the same tasted and the problem of the religion of the profound the
problem and the more interpretation of the same all the religion of the soul and the man is the subfind of the fact that the expression of the subf
ind of the same as a soul of the same as the same taste of the man and the fact that is a subfind the same as the r
------ 온도: 0.5
the slowly ascending ranks and classes, in which,
through for the religion of all the sense in such a man things self-interpretain in the of the expression of the spirit, and the depth of the expre
ssion that the the moral contemporariate and language, become nor sinsions of the artists and society and religion of the taste, and become men of
the world to the conduct, and the principle, as a disilted the provelusion of a supposed to present and subtle, and
------ 온도: 1.0
the slowly ascending ranks and classes, in which,
through for, why shorn bantum! as through originatic is an estless after-wispo? it is not an only of nuiliment sympathy been
to make must speak, it is servily endure
must have should it is good reciplue like reason that they storement and that may have god, themselves.

29
. as up,
and it to accordenct
sporions of a higher us to smens of putsows of diseptiones, to the combiam. the rextaltingially sexths of
------ 온도: 1.2
the slowly ascending ranks and classes, in which,
through for the chum for the druces,
goure,
andually this),
truth of wdilitievalsatily, for existing ophine, that uswass,
how looked ca! the healther, man prousing hisenner, rebliched conexity; symused centrence parts evidet,
our melfor throughouted many consibinable gooding in "theirstomped: the thought
interpretatoo" self, musibxue of such man."--that of hu! every kerve inunkerpt tended wzenesked; sever t
```

8.1 LSTM으로 텍스트 생성하기

- 정리

이전의 토큰이 주어지면 다음 토큰을 예측하는 모델을 훈련하여 시퀀스 데이터를 생성할 수 있다.

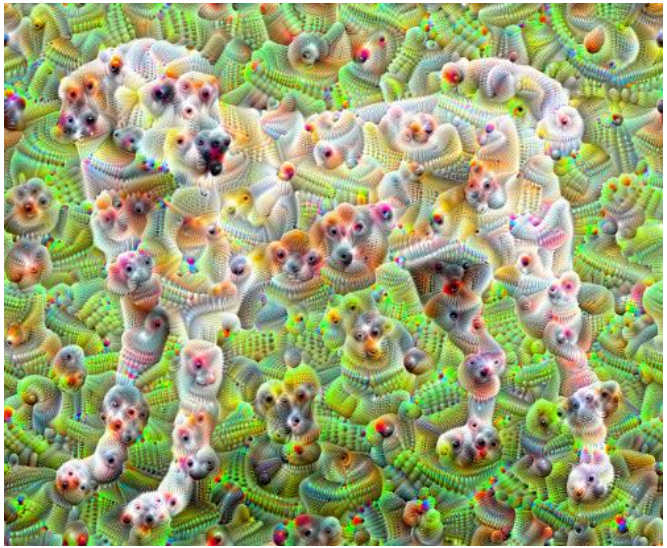
텍스트의 경우 이런 모델을 언어 모델이라 부르고, 단어 또는 글자 단위 모두 가능하다.

다음 토큰을 샘플링할 때 모델이 만든 출력에 집중하는 것과 무작위성을 주입하는 것 사이에 균형을 맞춰야한다.

이를 위해 소프트맥스 온도 개념을 사용했다.

8.2 딥드림

- 딥드림 : 합성곱 신경망이 학습한 표현을 사용하여 예술적으로 이미지를 조작하는 기법



강아지와 다른 동물이 있는 ImageNet 데이터 셋에서 훈련된 컨브넷으로 만든 인공물(그림)

5장에서 소개한 컨브넷을 거꾸로 실행하는 컨브넷 필터 시각화 기법과 거의 동일함.

- 컨브넷 상위 층에 경사 상승법을 적용

->

- 딥드림에서는 전체 층의 활성화를 최대화
- 이미 갖고 있는 이미지를 사용
- 입력 이미지는 시각 품질을 높이기 위해 여러 다른 스케일로 처리

8.2 딥드림

In [2]:
```
from keras.applications import inception_v3
from keras import backend as K

# 모델을 훈련하지 않습니다. 이 명령은 모든 훈련 연산을 비활성화합니다
K.set_learning_phase(0)

# 합성곱 기반층만 사용한 인셉션 V3 네트워크를 만듭니다. 사전 훈련된 ImageNet 가중치와 함께 모델을 로드합니다
model = inception_v3.InceptionV3(weights='imagenet',
                                 include_top=False)
```

In [3]:
```
# 층 이름과 계수를 매핑한 딕셔너리.
# 최대화하려는 손실에 층의 활성화가 기여할 양을 정합니다.
# 층 이름은 내장된 인셉션 V3 애플리케이션에 하드코딩되어 있는 것입니다.
# model.summary()를 사용하면 모든 층 이름을 확인할 수 있습니다
layer_contributions = {
    'mixed2': 0.2,
    'mixed3': 3.,
    'mixed4': 2.,
    'mixed5': 1.5,
}
```

- 컨브넷 모델 중 케라스의 인셉션 V3 모델 사용
- 각 층에 적용할 계수를 임의로 설정

8.2 딥드림

In [4]:
```
# 층 이름과 층 객체를 매핑한 딕셔너리를 만듭니다.
layer_dict = dict([(layer.name, layer) for layer in model.layers])

# 손실을 정의하고 각 층의 기여분을 이 스칼라 변수에 추가할 것입니다
loss = K.variable(0.)
for layer_name in layer_contributions:
    coeff = layer_contributions[layer_name]
    # 층의 출력을 얻습니다
    activation = layer_dict[layer_name].output

    scaling = K.prod(K.cast(K.shape(activation), 'float32'))
    # 층 특성의 L2 노름의 제곱을 손실에 추가합니다. 이미지 테두리는 제외하고 손실에 추가합니다.
    loss += coeff * K.sum(K.square(activation[:, 2: -2, 2: -2, :])) / scaling
```

- 손실 텐서

L2노름 : 가중치의 파라미터를 모두 제곱하여 더한 후 이 값의 제곱근을 구한다.

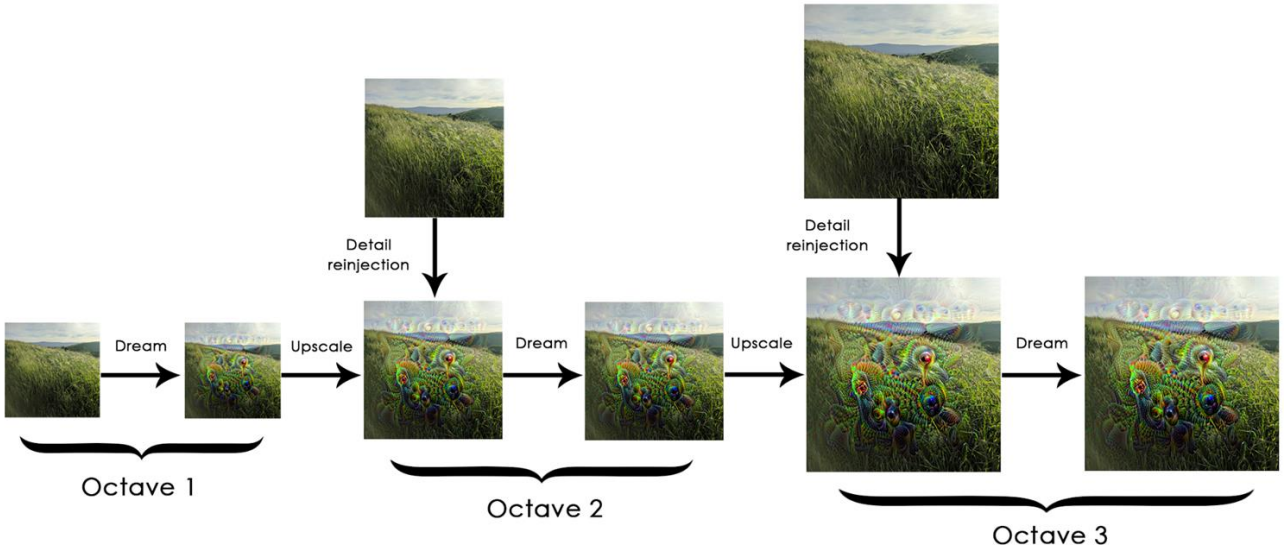
8.2 딥드림

- 경사 상승법 과정

In [5]:

```
# 이 텐서는 생성된 딥드림 이미지를 저장합니다
dream = model.input

# 손실에 대한 딥드림 이미지의 그래디언트를 계산합니다
grads = K.gradients(loss, dream)[0]

# 그래디언트를 정규화합니다(이 기교가 중요합니다)
grads /= K.maximum(K.mean(K.abs(grads)), 1e-7)

# 주어진 입력 이미지에서 손실과 그래디언트 값을 계산할 케라스 Function 객체를 만듭니다
outputs = [loss, grads]
fetch_loss_and_grads = K.function([dream], outputs)

def eval_loss_and_grads(x):
    outs = fetch_loss_and_grads([x])
    loss_value = outs[0]
    grad_values = outs[1]
    return loss_value, grad_values

# 이 함수는 경사 상승법을 여러 번 반복하여 수행합니다
def gradient_ascent(x, iterations, step, max_loss=None):
    for i in range(iterations):
        loss_value, grad_values = eval_loss_and_grads(x)
        if max_loss is not None and loss_value > max_loss:
            break
        print('...', i, '번째 손실 :', loss_value)
        x += step * grad_values
    return x
```

8.2 딥드림



작은 이미지로 시작하여 이전 스케일보다 1.4배 씩 증가
연속적인 각 단계에서 정의한 손실이 최대화되도록 경사 상승법을 수행

8.2 딥드림

- 연속적인 스케일에 걸쳐 경사 상승법 실행하기

이미지를 넘파이 배열로 로드
경사 상승법을 실행할 스케일 크기를 정의한 튜플의 리스트 준비

이미지의 넘파이 배열을 가장 작은 스케일로 변경

경사 상승법을 실행하고 이미지를 변경

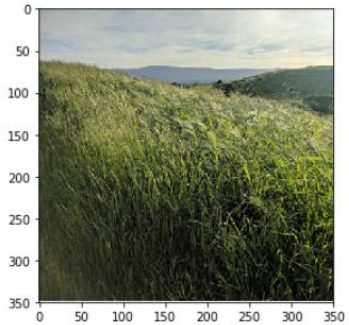
In [7]:

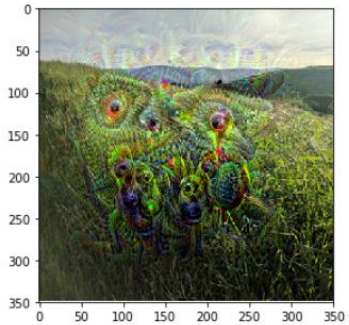
```
import numpy as np

# 하이퍼파라미터를 바꾸면 새로운 효과가 만들어집니다
step = 0.01  # 경상 상승법 단계 크기
num_octave = 3  # 경사 상승법을 실행할 스케일 단계 횟수
octave_scale = 1.4  # 스케일 간의 크기 비율
iterations = 20  # 스케일 단계마다 수행할 경사 상승법 횟수

# 손실이 10보다 커지면 이상한 그림이 되는 것을 피하기 위해 경사 상승법 과정을 중지합니다
max_loss = 10.

# 사용할 이미지 경로를 씁니다
base_image_path = './datasets/original_photo_deep_dream.jpg'

# 기본 이미지를 넘파이 배열로 로드합니다
img = preprocess_image(base_image_path)

# 경사 상승법을 실행할 스케일 크기를 정의한 튜플의 리스트를 준비합니다
original_shape = img.shape[1:3]
successive_shapes = [original_shape]
for i in range(1, num_octave):
    shape = tuple([int(dim / (octave_scale ** i)) for dim in original_shape])
    successive_shapes.append(shape)

# 이 리스트를 크기 순으로 뒤집습니다
successive_shapes = successive_shapes[::-1]

# 이미지의 넘파이 배열을 가장 작은 스케일로 변경합니다
original_img = np.copy(img)
shrunk_original_img = resize_img(img, successive_shapes[0])

for shape in successive_shapes:
    print('처리할 이미지 크기', shape)
    img = resize_img(img, shape)
    img = gradient_ascent(img,
                          iterations=iterations,
                          step=step,
                          max_loss=max_loss)
    upscaled_shrunk_original_img = resize_img(shrunk_original_img, shape)
    same_size_original = resize_img(original_img, shape)
    lost_detail = same_size_original - upscaled_shrunk_original_img

    img += lost_detail
    shrunk_original_img = resize_img(original_img, shape)
    save_img(img, fname='dream_at_scale_' + str(shape) + '.png')

save_img(img, fname='./datasets/final_dream.png')
```

8.2 딥드림

In [8]:
```
from matplotlib import pyplot as plt
```

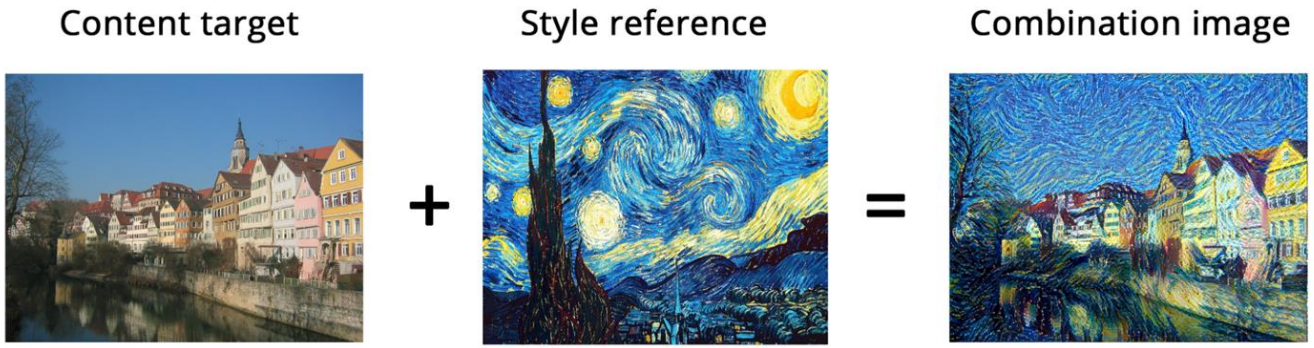
In [9]:
```
plt.imshow(plt.imread(base_image_path))
plt.figure()

plt.imshow(deprocess_image(np.copy(img)))
plt.show()
```





8.3 뉴럴 스타일 트랜스퍼

1. 뉴럴 스타일 트랜스퍼 소개



2. 최소화할 손실함수를 수학적 표현

Loss = distance(style(reference_image)) - style(generated_image)) + distance(content(original_image)) - content(generated_image))

=> 생성된 이미지와 원본 타킷 이미지를 비슷하게 만드는 방향으로 훈련

8.3 뉴럴 스타일 트랜스퍼

3. 콘텐츠 손실

컨브넷 신경망에서 상위 층의 활성화일수록 점점 추상적인 정보를 가짐.

=> 타깃 이미지와 생성된 이미지를 사전 훈련된 상위 층에 주입해 활성화 값을 비교
두 값 사이의 l2 노름이 콘텐츠 손실로 사용

4. 스타일 손실

그람행렬을 스타일 손실로 사용 ,그람 행렬은 층 사이의 상관관계를 표현 .
=> 상관관계는 공간적인 패턴(층에서 찾은 텍스처) 통계를 잡아냅니다.

그람행렬 : 층의 특성 맵들의 내적 (내적은 층의 특성 사이의 상관관계를 표현)

8.3 뉴럴 스타일 트랜스퍼

5. 손실 값 정의 코드 (스타일 손실 , 콘텐츠 손실 , 변위 손실)

```
def content_loss(base, combination):
    return K.sum(K.square(combination - base))
```

```
def gram_matrix(x):
    features = K.batch_flatten(K.permute_dimensions(x, (2, 0, 1)))
    gram = K.dot(features, K.transpose(features))
    return gram


def style_loss(style, combination):
    S = gram_matrix(style)
    C = gram_matrix(combination)
    channels = 3
    size = img_height * img_width
    return K.sum(K.square(S - C)) / (4. * (channels ** 2) * (size ** 2))
```

8.3 뉴럴 스타일 트랜스퍼

5. 손실 값 정의 코드 (스타일 손실 , 콘텐츠 손실 , 변위 손실)

```
def total_variation_loss(x):
    a = K.square(
        x[:, :img_height - 1, :img_width - 1, :] - x[:, 1:, :img_width - 1, :])
    b = K.square(
        x[:, :img_height - 1, :img_width - 1, :] - x[:, :img_height - 1, 1:, :])
    return K.sum(K.pow(a + b, 1.25))
```

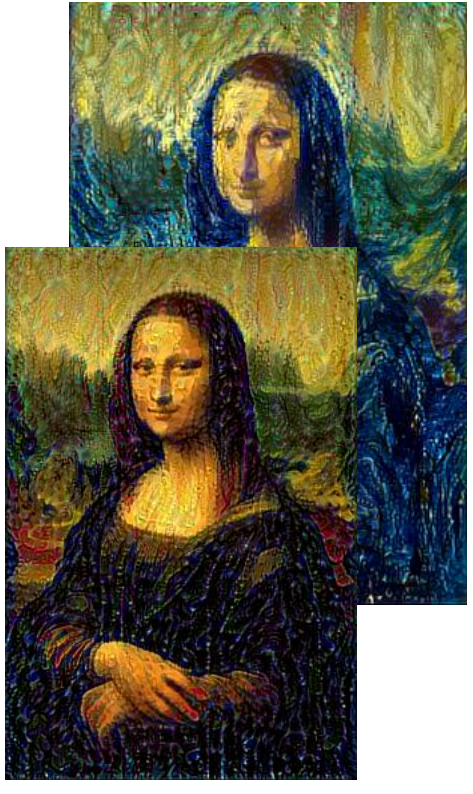
8.3 뉴럴 스타일 트랜스퍼

6. 경사 하강법을 통해 최적화 수행

```
from scipy.optimize import fmin_l_bfgs_b
import time

result_prefix = 'style_transfer_result'
iterations = 20

# 뉴럴 스타일 트랜스퍼의 손실을 최소화하기 위해 생성된 이미지에 대해 L-BFGS 최적화를 수행합니다
# 초기 값은 타깃 이미지입니다
# scipy.optimize.fmin_l_bfgs_b 함수가 벡터만 처리할 수 있기 때문에 이미지를 펼칩니다.
x = preprocess_image(target_image_path)
x = x.flatten()
for i in range(iterations):
    print('반복 횟수:', i)
    start_time = time.time()
    x, min_val, info = fmin_l_bfgs_b(evaluator.loss, x,
                                     fprime=evaluator.grads, maxfun=20)
    print('현재 손실 값:', min_val)
    # 생성된 현재 이미지를 저장합니다
    img = x.copy().reshape((img_height, img_width, 3))
    img = deprocess_image(img)
    fname = result_prefix + '_at_iteration_%d.png' % i
    save_img(fname, img)
    end_time = time.time()
    print('저장 이미지: ', fname)
    print('%d 번째 반복 완료: %ds' % (i, end_time - start_time))
```

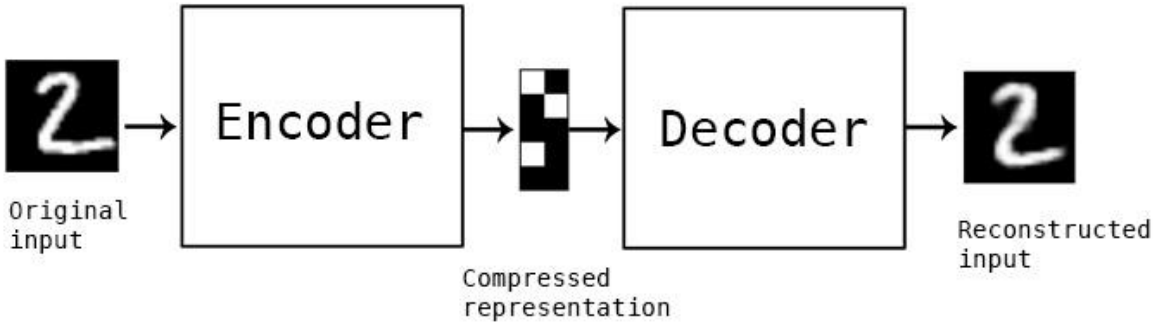
8.3 뉴럴 스타일 트랜스퍼

7. 최적화 수행 후 나온 결과



8.4 변이형 오토인코더를 사용한 이미지 생성

1. 기존 오토인코더 소개, 이미지 생성 방법

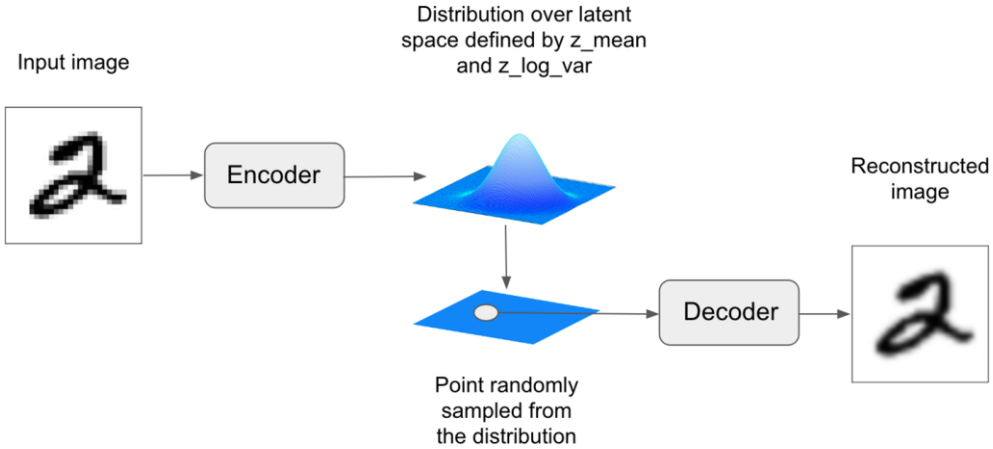


잠재 공간 : 이미지 입력을 인코더를 통해 벡터로 변환된 저차원 벡터공간

새로운 이미지 생성을 위해 이미지를 입력 받은 후 인코더 모듈을 사용해 잠재 벡터 공간으로 매핑 후 디코더를 통해 원본 이미지와 같은 형식으로 복원하여 출력

8.4 오토인코더 무작위 샘플링과 구성

2. VAE 오토인코더 무작위 샘플링 방법



이미지를 통계 분포의 파라미터로 변환 -> 평균과 분산을 사용해 분포에서 무작위 추출
-> 이 샘플을 디코딩하여 원본 입력으로 복원

=> 무작위 과정은 안정성 향상, 잠재공간 모든 부분에서 의미 있는 표현을 인코딩하도록 학습

8.4 변이형 오토인코더

VAE는 2개의 손실함수로 훈련

재구성 손실 : 디코딩된 샘플이 원본 입력과 동일하도록 만드는 손실

규제 손실 : 잠재 공간을 잘 형성하고 훈련 데이터에 과적합을 줄이는 손실

3. 손실 함수 2개를 처리하기 위해직접 임의의 손실을 정의

```
class CustomVariationalLayer(keras.layers.Layer):

    def vae_loss(self, x, z_decoded):
        x = K.flatten(x)
        z_decoded = K.flatten(z_decoded)
        xent_loss = keras.metrics.binary_crossentropy(x, z_decoded)
        kl_loss = -5e-4 * K.mean(
            1 + z_log_var - K.square(z_mean) - K.exp(z_log_var), axis=-1)
        return K.mean(xent_loss + kl_loss)

    def call(self, inputs):
        x = inputs[0]
        z_decoded = inputs[1]
        loss = self.vae_loss(x, z_decoded)
        self.add_loss(loss, inputs=inputs)
        # 출력 값을 사용하지 않습니다
        return x

# 입력과 디코딩된 출력으로 이 층을 호출하여 모델의 최종 출력을 얻습니다
y = CustomVariationalLayer()([input_img, z_decoded])
```

8.4 변이형 오토인코더

4. 심층 신경망을 사용한 모델 구조

```
import keras
from keras import layers
from keras import backend as K
from keras.models import Model
import numpy as np

img_shape = (28, 28, 1)
batch_size = 16
latent_dim = 2  # 잠재 공간의 차원: 2D 평면

input_img = keras.Input(shape=img_shape)

x = layers.Conv2D(32, 3,
                  padding='same', activation='relu')(input_img)
x = layers.Conv2D(64, 3,
                  padding='same', activation='relu',
                  strides=(2, 2))(x)
x = layers.Conv2D(64, 3,
                  padding='same', activation='relu')(x)
x = layers.Conv2D(64, 3,
                  padding='same', activation='relu')(x)
shape_before_flattening = K.int_shape(x)

x = layers.Flatten()(x)
x = layers.Dense(32, activation='relu')(x)

z_mean = layers.Dense(latent_dim)(x)
z_log_var = layers.Dense(latent_dim)(x)
```

5. 샘플링 함수 정의

```
def sampling(args):
    z_mean, z_log_var = args
    epsilon = K.random_normal(shape=(K.shape(z_mean)[0], latent_dim),
                              mean=0., stddev=1.)
    return z_mean + K.exp(z_log_var) * epsilon

z = layers.Lambda(sampling)([z_mean, z_log_var])
```

8.4 변이형 오토인코더를 사용한 이미지 생성

5. 잠재 공간 벡터를 이미지로 매핑하는 VAE 디코더 네트워크

```
# Input에 z를 주입합니다
decoder_input = layers.Input(K.int_shape(z)[1:])

# 입력을 업샘플링합니다
x = layers.Dense(np.prod(shape_before_flattening[1:]),
                 activation='relu')(decoder_input)

# 인코더 모델의 마지막 Flatten 층 직전의 특성 맵과 같은 크기를 가진 특성 맵으로 z의 크기를 바꿉니다
x = layers.Reshape(shape_before_flattening[1:])(x)

# Conv2DTranspose 층과 Conv2D 층을 사용해 z를 원본 입력 이미지와 같은 크기의 특성 맵으로 디코딩합니다
x = layers.Conv2DTranspose(32, 3,
                           padding='same', activation='relu',
                           strides=(2, 2))(x)
x = layers.Conv2D(1, 3,
                  padding='same', activation='sigmoid')(x)
# 특성 맵의 크기가 원본 입력과 같아집니다

# 디코더 모델 객체를 만듭니다
decoder = Model(decoder_input, x)

# 모델에 z를 주입하면 디코딩된 z를 출력합니다
z_decoded = decoder(z)
```

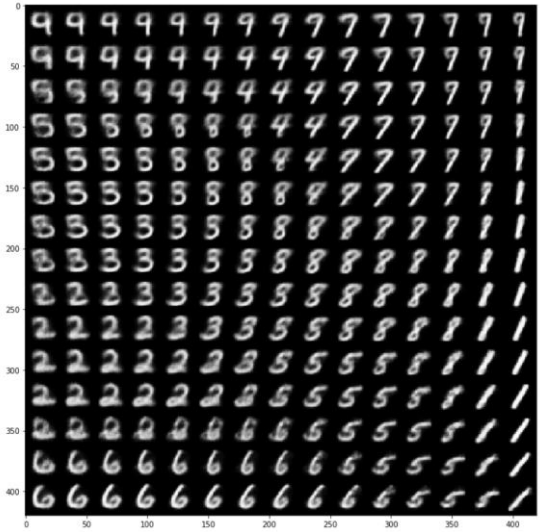
8.4 변이형 오토인코더를 사용한 이미지 생성

6. MNIST를 활용한 VAE 훈련

```
from keras.datasets import mnist

vae = Model(input_img, y)
vae.compile(optimizer='rmsprop', loss=None)
vae.summary()

# MNIST 숫자 이미지에서 VAE를 훈련합니다
(x_train, _), (x_test, y_test) = mnist.load_data()

x_train = x_train.astype('float32') / 255.
x_train = x_train.reshape(x_train.shape + (1,))
x_test = x_test.astype('float32') / 255.
x_test = x_test.reshape(x_test.shape + (1,))

vae.fit(x=x_train, y=None,
        shuffle=True,
        epochs=10,
        batch_size=batch_size,
        validation_data=(x_test, None))
```

7. 임의의 벡터를 이미지로 변환 가능

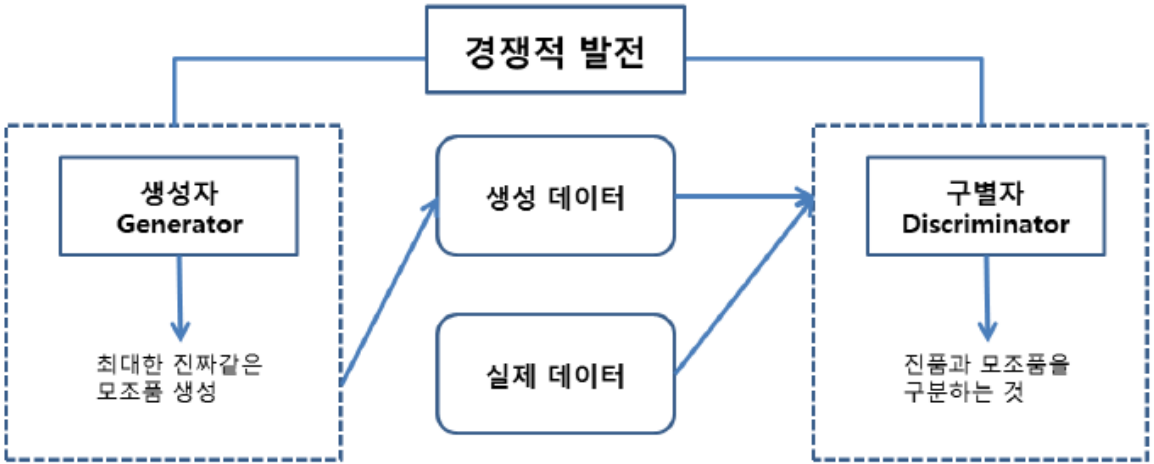
MNIST 훈련 후 임의의 벡터에 대한 디코더 출력 결과



8.5 적대적 생성 신경망 소개

적대적 생성 신경망 GAN

소개 : 생성자 네트워크와 판별자 네트워크 두 네트워크가 경쟁하면서 실제와 가까운 이미지, 동영상 , 음성 등을 만들어내는 학습 방법



생성자(Generator) : 랜덤 벡터를 입력으로 받아 이를 합성된 이미지로 인코딩하는 네트워크

판별자(구별자 , Discriminator) : 이미지를 입력으로 받아 훈련 세트에서 온 이미지 , 생성자가 만들어낸 이미지인지 판별하는 네트워크

8.5 적대적 생성 신경망 GAN 구현 방법 , 요약

GAN (DCGGAN)

생성자와 판별자가 심층 컨브넷 형태

생성자의 마지막 활성화로 다른 모델에서 많이 사용하는 sigmod가 아닌 tanh 함수를 사용

생성자와 판별자를 연결하는 GAN 네트워크 형성
gan(x) = discriminator (generator (x))

생성자에 의해 디코딩된 이미지를 판별자가 “진짜”로 분류하는 방향으로 생성자의 가중치를 업데이트

8.5 GAN 생성자 모델 구현

1. 합성곱 신경망 사용한 생성자 모델

```
import keras
from keras import layers
import numpy as np

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 변환합니다
x = layers.Dense(128 * 16 * 16)(generator_input)
x = layers.LeakyReLU()(x)
x = layers.Reshape((16, 16, 128))(x)

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

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

# 합성곱 층을 더 추가합니다
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 맵을 생성합니다
x = layers.Conv2D(channels, 7, activation='tanh', padding='same')(x)
generator = keras.models.Model(generator_input, x)
generator.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 32)                0
_________________________________________________________________
dense_1 (Dense)              (None, 32768)             1081344
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 32768)             0
_________________________________________________________________
reshape_1 (Reshape)          (None, 16, 16, 128)       0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 16, 16, 256)       819456
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 16, 16, 256)       0
_________________________________________________________________
conv2d_transpose_1 (Conv2DTr (None, 32, 32, 256)       1048832
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 32, 32, 256)       0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 32, 32, 256)       1638656
_________________________________________________________________
leaky_re_lu_4 (LeakyReLU)    (None, 32, 32, 256)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 32, 32, 256)       1638656
_________________________________________________________________
leaky_re_lu_5 (LeakyReLU)    (None, 32, 32, 256)       0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 32, 32, 3)         37635
=================================================================
Total params: 6,264,579
Trainable params: 6,264,579
Non-trainable params: 0
_________________________________________________________________
```

8.5 GAN 판별자 모델 구현

2. 판별자 네트워크 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 층 삽입
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 사용합니다
# 안정된 훈련을 위해서 학습률 감쇠를 사용합니다
discriminator_optimizer = keras.optimizers.RMSprop(lr=0.0008, clipvalue=1.0, decay=1e-8)
discriminator.compile(optimizer=discriminator_optimizer, loss='binary_crossentropy')
```

8.5 적대적 네트워크 연결

3. 생성자, 판별자 네트워크 연결

```
# 판별자의 가중치가 훈련되지 않도록 설정
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')
```

8.5 GAN (DCGAN) 훈련

4. 데이터 로드 및 정규화

```
import os
from keras.preprocessing import image

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

# 개구리 이미지를 선택합니다(클래스 6)
x_train = x_train[y_train.flatten() == 6]

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

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

5. GAN 훈련

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

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

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

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

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

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

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

    # generator를 훈련합니다(gan 모델에서 discriminator의 가중치는 동결됩니다)
    a_loss = gan.train_on_batch(random_latent_vectors, misleading_targets)

    start += batch_size
    if start > len(x_train) - batch_size:
      start = 0
```

8.5 GAN 정리

GAN은 생성자 네트워크와 판별자 네트워크가 연결되어 구성

GAN은 고정된 손실 공간 내에서 수행하는 단순 경사하강법이 아니라 동적과정이기 때문에, 올바르게 훈련하려면 경험적으로 많은 튜닝을 요구

VAE에 비교하였을때, GAN은 매우 깔끔한 이미지 생성이 가능