Model Subclassing

최석재 *lingua@naver.com*

모델 서브클래싱

- 2019년 Tensorflow 2.0과 함께 발표된 방법
- Pytorch에 대응하기 위해 나온 것으로 케라스 모델링 방법 중 가장 자유도가 높다

- 공통된 기능은 간단한 방법으로 제공하는 Keras의 장점과
- 자유도 높은 모델링이 가능한 PyTorch의 장점을 모두 가지고 있다

- 성능면에서는 앞에서 다룬 Sequential Model이나 Functional API와 차이가 없으며,
- 대부분 이 두 방법으로 필요한 모델을 만들 수 있다
- 단, 모델이 순환적 성격을 가질 때는 객체를 반복할 수 있는 Model Subclassing 방법이 적절하다

- 앞의 두 방법에 비하여 부가 기능은 다소 부족하나, 계속 개발 중에 있다
- 예) 모델 요약, 모델 그래프, 층의 이름 활용

모델 서브클래싱

- 특징으로는 call() 메소드를 사용한다는 점으로서
- call() 메소드는 파이썬의 __call()__ 와는 다른 것이며,
- .fit() 메소드를 통해 데이터를 입력할 때 실행된다

- 메인 클래스를 생성할 때는 tf.keras.Model 클래스를 상속한다
- 즉, Model 클래스의 기능을 상속받아 새로운 Model 클래스를 만든다
- .fit(), .evaluate(), .predict() 메소드 등을 활용할 수 있어 개발에 유용하다

모델 설계 (1/2)

- import tensorflow as tf

- class MyModel(tf.keras.Model):
  def __init__(self, class_num): # 객체 생성시 전달할 파라미터는 이곳으로 전달한다
  super(MyModel, self).__init__() # 부모 클래스인 tf.keras.Model의 __init__() 호출해야 한다

  # 사용할 layer 정의
  self.conv1 = tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)) # 32개 필터 Conv2D 층
  self.pool1 = tf.keras.layers.MaxPooling2D((2, 2)) # MaxPooling 층
  self.conv2 = tf.keras.layers.Conv2D(64, (3, 3), activation='relu') # 64개 필터 Conv2D 층
  self.pool2 = tf.keras.layers.MaxPooling2D((2, 2)) # MaxPooling 층
  self.flatten = tf.keras.layers.Flatten() # Flatten 층
  self.dense = tf.keras.layers.Dense(class_num, activation='softmax') # 출력노드 class_num의 Dense 층

※ tf.keras.layers 의 다양한 layer 클래스를 이용하여 층을 생성할 수 있다

모델 설계 (2/2)

call()은 fit()이 호출될 때 사용된다. 생성된 객체에 전달할 파라미터는 이곳으로 전달한다

- def call(self, input):
  - x = self.conv1(input)
  - x = self.pool1(x)
  - x = self.conv2(x)
  - x = self.pool2(x)
  - x = self.flatten(x)
  - x = self.dense(x)
  - return x

```
import tensorflow as tf

class MyModel(tf.keras.Model):
  def __init__(self, class_num):
    super(MyModel, self).__init__()

    # 사용할 layer 정의
    self.conv1 = tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3))
    self.pool1 = tf.keras.layers.MaxPooling2D((2, 2))
    self.conv2 = tf.keras.layers.Conv2D(64, (3, 3), activation='relu')
    self.pool2 = tf.keras.layers.MaxPooling2D((2, 2))  # MaxPooling 층
    self.flatten = tf.keras.layers.Flatten()           # Flatten 층
    self.dense = tf.keras.layers.Dense(class_num, activation='softmax')

  def call(self, input):
    x = self.conv1(input)
    x = self.pool1(x)
    x = self.conv2(x)
    x = self.pool2(x)
    x = self.flatten(x)
    x = self.dense(x)
    return x
```

모델 생성 및 컴파일

- 일반적으로는 다음의 방법으로 모델을 생성하고, 컴파일한다

모델 생성

- mymodel = MyModel(class_num=10) # 출력층 노드 10개

모델 컴파일

- mymodel.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['acc'])

모델 요약 보기

모델 생성

- mymodel = MyModel(class_num=10) # 출력층 노드는 10개

모델 이름 설정 (생략 가능)

- mymodel._name = 'My_Test_Model'

모델 input shape 정의

- mymodel(tf.keras.layers.Input(shape=(32, 32, 3))) # 특성수

모델 요약을 보기 위해서는 Input Shape를 설정해야 한다

모델 요약. input shape를 정의해야 summary()를 쓸 수 있다

- mymodel.summary()

모델 컴파일

- mymodel.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['acc'])

그러나 output shape는 자세히 나오지 않는다

Model: "My_Test_Model"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	multiple	896
max_pooling2d (MaxPooling2D)	multiple	0
conv2d_1 (Conv2D)	multiple	18496
max_pooling2d_1 (MaxPooling2D)	multiple	0
flatten (Flatten)	multiple	0
dense (Dense)	multiple	23050

Total params: 42442 (165.79 KB)
Trainable params: 42442 (165.79 KB)
Non-trainable params: 0 (0.00 Byte)

모델 그래프

- from tensorflow.keras.utils import plot_model
- plot_model(mymodel, show_shapes=True, show_layer_names=True)

그러나 모델 그래프는 제대로 나오지 않는다

다중 입력 모델

다중 입력 모델

- Functional API로 만든 다중 입력 모델을 Model Subclassing 방법으로 구현해본다

- import tensorflow as tf
- from keras.models import Model
- from keras import layers

모델 설계 (1/2) – MyModel 클래스

```
class MyModel(Model):
    def __init__(self, class_num):                          # 객체 생성시 전달할 파라미터는 이곳으로 전달한다
        super(MyModel, self).__init__()                     # 부모 클래스인 tf.keras.Model의 __init__()을 호출해야 한다

        # text1 Input 층
        self.embedded1 = layers.Embedding(text1_max_words, 64)      # 노드를 64개 갖는 임베딩 층
        self.lstm1 = layers.LSTM(32)                                # 노드를 32개 갖는 LSTM 층

        # text2 Input 층
        self.embedded2 = layers.Embedding(text2_max_words, 32)      # 노드를 32개 갖는 임베딩 층
        self.lstm2 = layers.LSTM(16)                                # 노드를 16개 갖는 LSTM 층

        # concatenate 층
        self.concatenated = layers.Concatenate(axis=-1)             # 두 데이터를 열 차원 기준으로 연결하는 층

        # 출력층
        self.dense = layers.Dense(class_num, activation='softmax')  # 출력노드를 10개 갖는 Dense 층
```

모델 설계 (2/2) – MyModel 클래스

- def call(self, inputs): # call()은 fit()이 호출될 때 사용된다. 생성된 객체에 전달할 파라미터는 이곳으로 전달한다

```
    embedded_text1 = self.embedded1(inputs['text1'])        # 첫 번째 입력
    lstm_text1 = self.lstm1(embedded_text1)

    embedded_text2 = self.embedded2(inputs['text2'])        # 두 번째 입력
    lstm_text2 = self.lstm2(embedded_text2)

    concatenated = self.concatenated([lstm_text1, lstm_text2])        # 두 입력을 연결
    answer = self.dense(concatenated)

    return answer
```

key

key

학습 데이터

- import numpy as np
- from keras.utils import to_categorical

- text1_max_words = 10000 # text1 단어의 최대 크기 숫자
- text2_max_words = 10000 # text2 단어의 최대 크기 숫자
- answer_max_words = 500 # 답변이 가질 수 있는 단어의 최대 크기 숫자

- num_samples = 1000 # 행의 개수. 두 데이터의 입력되는 샘플의 수가 같아야 한다
- max_len1 = 100 # text1의 최대 길이
- max_len2 = 200 # text2의 최대 길이. 열 연결로서 maxlen은 text1과 달라도 된다

입력 데이터 랜덤 생성. (행, 열)의 2차원 데이터로 만든다
- text1 = np.random.randint(low=0, high=text1_max_words, size=(num_samples, max_len1))
- text2 = np.random.randint(low=0, high=text2_max_words, size=(num_samples, max_len2))

답변 데이터 랜덤 생성. 답변이 가질 수 있는 단어의 최대 크기로 슬롯을 만든다
- answers = np.random.randint(low=0, high=answer_max_words, size=num_samples)
- answers = to_categorical(answers)

Error!

- 랜덤 함수로 데이터 생성 중 다음과 같은 에러가 발생할 수 있다

```
ValueError: Shapes (None, 499) and (None, 500) are incompatible
```

- answers의 생성된 숫자가 500개에 미치지 못하여 발생하는 경우로서,
  이때는 앞의 코드를 다시 실행한다

high=answer_max_words가 500이 입력되었으나, 랜덤 종류가 500이 안 된 경우로서,
이렇게 되면 answers = to_categorical(answers) 에서 500개의 슬롯이 만들어지지 못한다

그런데 출력층 생성시 tf.keras.layers.Dense(answer_max_words, activation='softmax')(concatenated)
로 500개를 쓸 것이라고 설정해놓았기 때문에 문제가 된다

모델 생성 및 컴파일

모델 생성

- mymodel = MyModel(class_num=answer_max_words) # 출력층 노드는 answer_max_words 개

모델 컴파일

- mymodel.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['acc'])

데이터 학습

key key

- mymodel.fit({'text1':text1, 'text2':text2}, answers, epochs=3, batch_size=128)

```
Epoch 1/3
8/8 [==============================] - 8s 324ms/step - loss: 6.2148 - acc: 0.0010
Epoch 2/3
8/8 [==============================] - 2s 189ms/step - loss: 6.2031 - acc: 0.0550
Epoch 3/3
8/8 [==============================] - 2s 192ms/step - loss: 6.1889 - acc: 0.0380
<keras.src.callbacks.History at 0x7b399bf7efb0>
```

예측

- prediction = mymodel.predict({'text1':text1, 'text2':text2})
- label = np.argmax(prediction[0])
- print(label) # 293번 단어로 예측하였다

```
32/32 [==============================] - 2s 22ms/step
293
```

다중 출력 모델

다중 출력 모델

- Functional API로 만든 다중 입력 모델을 Model Subclassing 방법으로 구현해본다

- import tensorflow as tf
- from keras.models import Model
- from keras import layers

모델 설계 (1/3) – MyModel 클래스

- class MyModel(Model):
- def __init__(self, **num_income_groups**): # 객체 생성시 전달할 파라미터는 이곳으로 전달한다

```
    super(MyModel, self).__init__()                 # 부모 클래스인 tf.keras.Model의 __init__()을 호출해야 한다

    self.embedded1 = layers.Embedding(max_words, 256)     # 노드를 256 갖는 임베딩 층. 파라미터로 전달 가능
    self.conv1 = layers.Conv1D(filters=128,kernel_size=5,activation='relu')
    self.pool1 = layers.MaxPooling1D(pool_size=5)
    self.conv2 = layers.Conv1D(filters=256, kernel_size=5, activation='relu')
    self.conv3 = layers.Conv1D(filters=256, kernel_size=5, activation='relu')
    self.pool2 = layers.MaxPooling1D(pool_size=5)
    self.conv4 = layers.Conv1D(filters=256, kernel_size=2, activation='relu')
    self.pool3 = layers.GlobalMaxPooling1D()
    self.dense = layers.Dense(128, activation='relu')
```

모델 설계 (2/3) – MyModel 클래스

```
# 출력층
self.pred_age = layers.Dense(1, name='age')                                          # 연령 예측
self.pred_income = layers.Dense(num_income_groups, activation='softmax', name='income') # 소득 수준 분류
self.pred_gender = layers.Dense(1, activation='sigmoid', name='gender')              # 성별 예측
```

모델 설계 (3/3) – MyModel 클래스

- def call(self, input): # call()은 fit()이 호출될 때 사용된다. 생성된 객체에 전달할 파라미터는 이곳으로 전달

```
x = self.embedded1(input)
x = self.conv1(x)
x = self.pool1(x)
x = self.conv2(x)
x = self.conv3(x)
x = self.pool2(x)
x = self.conv4(x)
x = self.pool3(x)

age = self.pred_age(x)
income = self.pred_income(x)
gender = self.pred_gender(x)

return {'age': age, 'income': income, 'gender': gender}  # key로 사용할 수 있게 한다
```

학습 데이터

- import numpy as np

- max_words = 10000
- num_income_groups = 10

- num_samples = 1000
- max_len = 100

```
# 입력 데이터 생성
# (행, 열)의 2차원 데이터로 만듦
# 3개 데이터의 입력되는 샘플의 수는 같아야 한다
```

- posts = np.random.randint(low=0, high=max_words, size=(num_samples, max_len))

```
# 출력 데이터 생성
```

- target_age = np.random.randint(low=0, high=100, size=num_samples) # 연령은 0~99
- target_income = np.random.randint(low=0, high=10, size=num_samples) # 소득수준 그룹은 0~9
- target_gender = np.random.randint(low=0, high=2, size=num_samples) # 성별은 0 또는 1

모델 생성 및 컴파일

모델 생성

- mymodel = MyModel(num_income_groups=num_income_groups)

모델 컴파일

- mymodel.compile(optimizer='rmsprop', loss={'age': 'mse', 'income': 'sparse_categorical_crossentropy', 'gender': 'binary_crossentropy'}, metrics={'age': ['mse'], 'income': ['acc'], 'gender': ['acc']})

※ 3개의 예측 결과를 내야 하므로 3개의 loss가 필요하다

학습 및 예측

학습

- mymodel.fit(posts, {'age': target_age, 'income': target_income, 'gender': target_gender}, epochs=3, batch_size=128)

```
Epoch 1/3
8/8 [==============================] - 6s 555ms/step - loss: 1683.7950 - output_1_loss: 1680.114(
Epoch 2/3
8/8 [==============================] - 6s 743ms/step - loss: 727.0842 - output_1_loss: 724.0245 -
Epoch 3/3
8/8 [==============================] - 4s 554ms/step - loss: 466.5356 - output_1_loss: 463.4557 -
32/32 [==============================] - 2s 52ms/step
```

예측

- result = mymodel.predict(posts)

예측

```
# 연령 예측 (0~99)
print(result['age'][:10])
```

```
[[86.61336 ]
 [60.45579 ]
 [83.82495 ]
 [94.34294 ]
 [27.66571 ]
 [36.381546]
 [39.5737  ]
 [69.46188 ]
 [85.07294 ]
 [33.257767]]
```

```
# 소득수준 예측 (0~9)
print(result['income'][:5])
```

```
[[0.05312871 0.05464975 0.04803489 0.07145705 0.14625278 0.
  0.13314325 0.04180322 0.3057043  0.10303494]
 [0.06177144 0.09281629 0.05019458 0.07484925 0.1455449  0.
  0.16628297 0.06907307 0.18139955 0.09592415]
 [0.05086607 0.06806539 0.04483266 0.07035603 0.15833452 0.
  0.1495871  0.05012209 0.260216   0.09956857]
 [0.04515883 0.0673733  0.03796273 0.06249646 0.16524206 0.
  0.16582893 0.04721602 0.272968   0.09455997]
 [0.07736695 0.10862355 0.06722409 0.08682331 0.12686914 0.
  0.14021742 0.09254426 0.12042799 0.09445616]]
```

```
# 성별 예측 (바이너리)
print(result['gender'][:10])
```

```
[[0.53095806]
 [0.49183348]
 [0.5073338 ]
 [0.5094429 ]
 [0.48215818]
 [0.4832531 ]
 [0.49702263]
 [0.48870966]
 [0.5110466 ]
 [0.49252605]]
```

※ 1이 될 확률

여러 방식을 혼합하기

여러 방식을 혼합하여 사용하기

- Sequential 모델, Functional API 모델, Model Subclassing 모델은
- 서로 간 통합이 가능하다

- 다른 방식으로 작성된 커스텀 모델을 하나의 층으로 활용할 수 있다

Sequential – Functional API

함수형 API 방법으로 작성된 모델을
Sequential 모델에서
하나의 층으로 사용하고 있다

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_1 (Dense)             (None, 64)                64064

 dense_2 (Dense)             (None, 32)                2080

 model (Functional)          (None, 1)                 33

=================================================================
Total params: 66,177
Trainable params: 66,177
Non-trainable params: 0
_________________________________________________________________
```

```
from tensorflow.keras.models import Sequential
from tensorflow import keras
from tensorflow.keras import layers

inputs = keras.Input(shape=(32,))
outputs = layers.Dense(1, activation="sigmoid")(inputs)
binary_classifier = keras.Model(inputs=inputs, outputs=outputs)

model = Sequential()
model.add(layers.Dense(64, activation="relu", input_shape=(1000,)))
model.add(layers.Dense(32, activation="relu"))
model.add(binary_classifier)

model.summary()
```

Model Subclassing – Functional API

모델 서브클래싱 방법으로 작성된 클래스를 함수형 API에서 하나의 층으로 사용하고 있다

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 64)]              0

 dense (Dense)               (None, 32)                2080

 my_classifier (my_classifie  (None, 1)                33
 r)

=================================================================
Total params: 2,113
Trainable params: 2,113
Non-trainable params: 0
_________________________________________________________________
```

```
from tensorflow import keras
from tensorflow.keras import layers

class my_classifier(keras.Model):
    def __init__(self, num_classes=2):
        super().__init__()
        if num_classes == 2:
            num_units = 1
            activation = "sigmoid"
        else:
            num_units = num_classes
            activation = "softmax"
        self.dense = layers.Dense(num_units, activation=activation)

    def call(self, inputs):
        return self.dense(inputs)


inputs = keras.Input(shape=(64,))
features = layers.Dense(32, activation="relu")(inputs)
outputs = my_classifier(num_classes=1)(features)
model = keras.Model(inputs=inputs, outputs=outputs)

model.summary()
```

Functional API Model Subclassing

```
from tensorflow import keras
from tensorflow.keras import layers

inputs = keras.Input(shape=(64,))
outputs = layers.Dense(1, activation="sigmoid")(inputs)
binary_classifier = keras.Model(inputs=inputs, outputs=outputs)

class MyModel(keras.Model):
    def __init__(self):
        super().__init__()
        self.dense1 = layers.Dense(64, activation="relu")
        self.dense2 = layers.Dense(32, activation="relu")
        self.classifier = binary_classifier

    def call(self, inputs):
        features = self.dense1(inputs)
        features = self.dense2(features)
        outputs = self.classifier(features)
        return outputs

model = MyModel()
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc'])
```

함수형 API 방법으로 작성된 모델을
모델 서브클래싱 방법에서
하나의 층으로 사용하고 있다