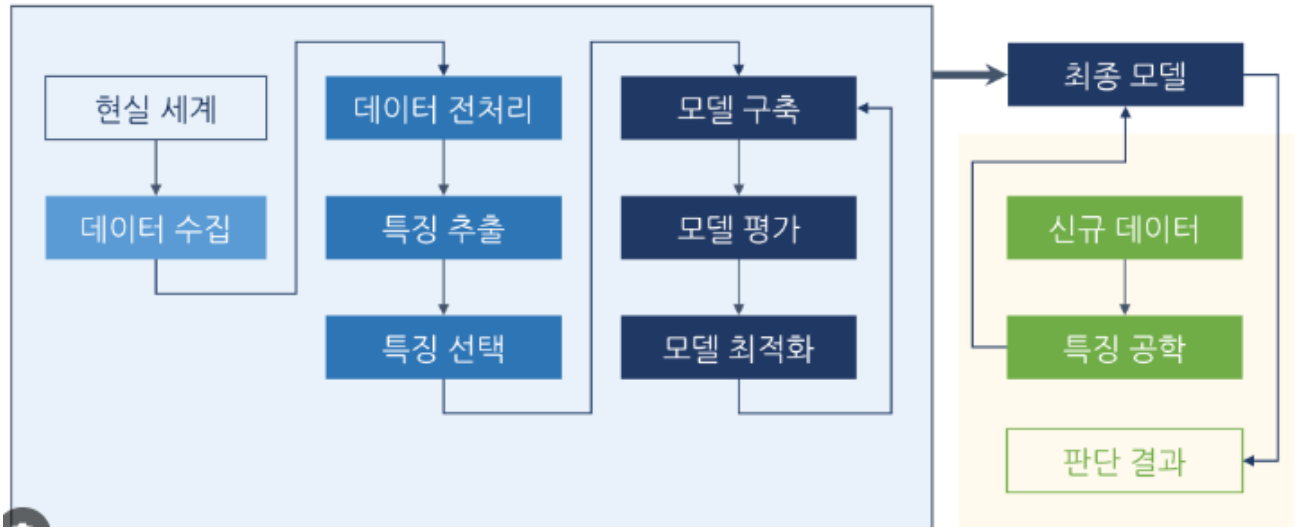
디지털 영상처리 연구실 연구보고서

김우헌

#딥러닝 프로세스



#텐서플로를 이용한 단순한 모델구축

- 케라스의 Sequential API이용

1. 모델 구조생성

```
import tensorflow as tf
```

```
# 리스트형
model = tf.keras.Sequential([
    tf.keras.layers.Dense(10),
    tf.keras.layers.Dense(5),
    tf.keras.layers.Dense(1),
])
```

```
# add 함수로 레이어 추가
model = tf.keras.Sequential()
model.add(tf.keras.layers.Dense(10))
model.add(tf.keras.layers.Dense(5))
model.add(tf.keras.layers.Dense(1))
```

2. 모델 컴파일과 훈련

```
model.compile(optimizer='sgd', loss='mean_squared_error',
              metrics=['mean_squared_error', 'mean_absolute_error'])
```

```
# 훈련
history = model.fit(x, y, epochs=1200)
```

3. 모델 검증과 예측

검증

```
# 검증
model.evaluate(x, y)
```

```
1/1 [==============================] - 0s 215ms/step - loss: 9.0305e-05 - mae: 0.0082
[9.030506771523505e-05, 0.00815649051219225]
```

예측

```
# 예측
model.predict([10])
```

```
array([[32.03942]], dtype=float32)
```

#MNIST 데이터를 이용한 딥러닝

1. 데이터 로드

```
import tensorflow as tf

# 케라스의 내장 데이터셋에서 mnist 데이터셋을 로드
mnist = tf.keras.datasets.mnist
```

```
# load_data()로 데이터셋을 로드 합니다.
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

⬇

```
train set:  (60000, 28, 28) (60000,)
test  set:  (10000, 28, 28) (10000,)
```

⬇

2. 데이터 전처리

```
# 데이터 정규화
x_train = x_train / x_train.max()

# 정규화 후 최소/최대 값 확인
print(f'정규화 후] 최소값: {x_train.min()}, 최대값: {x_train.max()}')
```

```
정규화 전] 최소값: 0, 최대값: 255
정규화 후] 최소값: 0.0, 최대값: 1.0
```

->수렴속도 GOOD, 국소 최적에 빠지는 현상 방지

3. 모델 구조 생성

```
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(32, activation='relu'),
    # 노드=10개 (클래스 개수와 동일)
    tf.keras.layers.Dense(10, activation='softmax'),
])
```

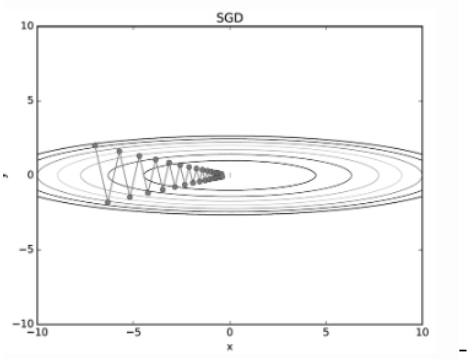
```
# 출력층 노드 = 1인 경우, sigmoid
tf.keras.layers.Dense(1, activation='sigmoid')

# 출력층 노드 = 2개 이상인 경우, softmax
tf.keras.layers.Dense(10, activation='softmax')
```

4. 모델 컴파일

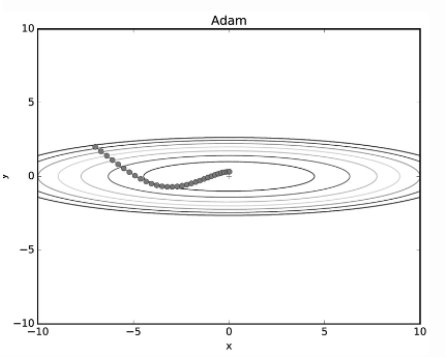
```
# 문자열로 지정
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

4-1. optimizer

-> 손실함수를 최소화하는 파라미터를 추정하는 알고리즘



->sgd



->adam

4-2. 손실함수

->손실함수의 최솟값을 찾기 위한 대상

```
# 이진 분류 (출력 노드 개수 = 1, sigmoid 인 경우)
model.compile(loss='binary_crossentropy')
```

출력 노드가 2개 이상, softmax 활성화 함수를 적용한 경우

```
# y가 원 핫 벡터인 경우
# [0., 0., 0., 0., 0., 1., 0., 0., 0., 0.]
model.compile(loss='categorical_crossentropy')

# y가 원 핫 벡터가 아닌경우
# [5]
model.compile(loss='sparse_categorical_crossentropy')
```

5. 훈련

```
# 훈련
model.fit(x_train, y_train,
          # 검증셋 지정
          validation_data=(x_test, y_test),
          epochs=10,
          )
```

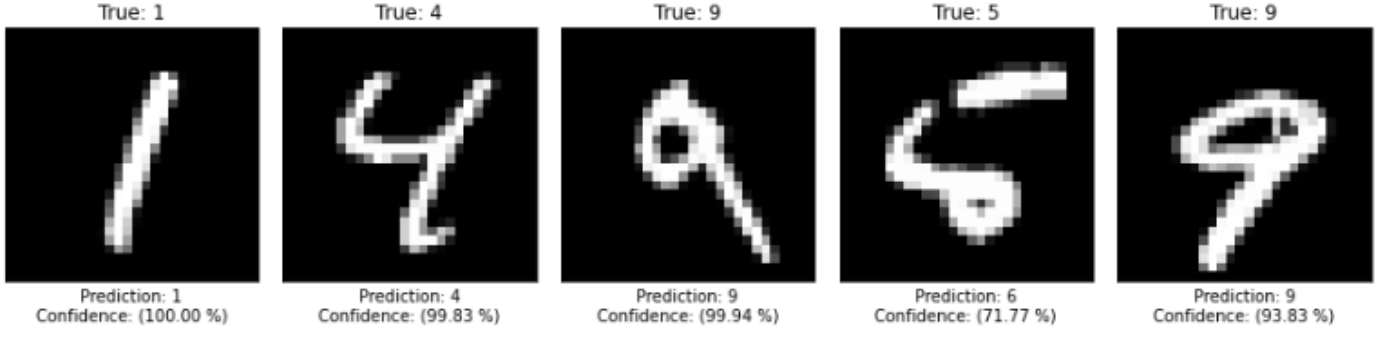
6. 평가

```
# 검증
test_loss, test_acc = model.evaluate(x_test,  y_test)

print('검증셋 정확도:', test_acc)
```

```
313/313 [==============================] - 1s 2ms/step - loss: 0.0856 - accuracy: 0.9799
검증셋 정확도: 0.9799000024795532
```

7. 예측

```
# 예측
predictions = model.predict(x_test)
```



#복잡한 모델 생성-

Sequential API

```
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax'),|
])
```

Functional API

```
# 모델의 레이어를 체인 구조로 연결 Input 레이어 정의
input_layer = tf.keras.Input(shape=(28, 28), name='InputLayer')

# 모델의 레이어를 체인 구조로 연결
x1 = tf.keras.layers.Flatten(name='Flatten')(input_layer)
x2 = tf.keras.layers.Dense(256, activation='relu', name='Dense1')(x1)
x3 = tf.keras.layers.Dense(64, activation='relu', name='Dense2')(x2)
x4 = tf.keras.layers.Dense(10, activation='softmax', name='OutputLayer')(x3)
```

```
# 모델 생성
func_model = tf.keras.Model(inputs=input_layer, outputs=x4, name='FunctionalModel')
```

```
Model: "FunctionalModel"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
InputLayer (InputLayer)      [(None, 28, 28)]          0
_________________________________________________________________
Flatten (Flatten)            (None, 784)               0
_________________________________________________________________
Dense1 (Dense)               (None, 256)               200960
_________________________________________________________________
Dense2 (Dense)               (None, 64)                16448
_________________________________________________________________
OutputLayer (Dense)          (None, 10)                650
=================================================================
Total params: 218,058
Trainable params: 218,058
Non-trainable params: 0
_________________________________________________________________
```