심층 신경망

```
# !pip install tensorflow==2.15.1
```

```
# 실행마다 동일한 결과를 얻기 위해 케라스에 랜덤 시드를 사용하고 텐서플로 연산을 결정적으로 만듭니다.
import tensorflow as tf

tf.keras.utils.set_random_seed(42)
tf.config.experimental.enable_op_determinism()
```

2개의 층

```
from tensorflow import keras

(train_input, train_target), (test_input, test_target) = keras.datasets.fashion_mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz
29515/29515 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz
26421880/26421880 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz
5148/5148 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz
4422102/4422102 [==============================] - 0s 0us/step
```

```
from sklearn.model_selection import train_test_split

train_scaled = train_input / 255.0
train_scaled = train_scaled.reshape(-1, 28*28)

train_scaled, val_scaled, train_target, val_target = train_test_split(
    train_scaled, train_target, test_size=0.2, random_state=42)
```

```
dense1 = keras.layers.Dense(100, activation='sigmoid', input_shape=(784,))
dense2 = keras.layers.Dense(10, activation='softmax')
```

심층 신경망 만들기

```
model = keras.Sequential([dense1, dense2])
```

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 100)               78500

 dense_1 (Dense)             (None, 10)                1010

=================================================================
Total params: 79510 (310.59 KB)
Trainable params: 79510 (310.59 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

층을 추가하는 다른 방법

```
model = keras.Sequential([
    keras.layers.Dense(100, activation='sigmoid', input_shape=(784,), name='hidden'),
    keras.layers.Dense(10, activation='softmax', name='output')
], name='패션 MNIST 모델')
```

```
model.summary()
```

```
Model: "패션 MNIST 모델"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 hidden (Dense)              (None, 100)               78500     
                                                                 
 output (Dense)              (None, 10)                1010      
                                                                 
=================================================================
Total params: 79510 (310.59 KB)
Trainable params: 79510 (310.59 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
model = keras.Sequential()
model.add(keras.layers.Dense(100, activation='sigmoid', input_shape=(784,)))
model.add(keras.layers.Dense(10, activation='softmax'))
```

```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_2 (Dense)             (None, 100)               78500     
                                                                 
 dense_3 (Dense)             (None, 10)                1010      
                                                                 
=================================================================
Total params: 79510 (310.59 KB)
Trainable params: 79510 (310.59 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
model.compile(loss='sparse_categorical_crossentropy', metrics=['accuracy'])

model.fit(train_scaled, train_target, epochs=5)
```

```
Epoch 1/5
1500/1500 [==============================] - 7s 3ms/step - loss: 0.5710 - accuracy: 0.8064
Epoch 2/5
1500/1500 [==============================] - 5s 4ms/step - loss: 0.4132 - accuracy: 0.8509
Epoch 3/5
1500/1500 [==============================] - 4s 3ms/step - loss: 0.3776 - accuracy: 0.8646
Epoch 4/5
1500/1500 [==============================] - 4s 3ms/step - loss: 0.3530 - accuracy: 0.8732
Epoch 5/5
1500/1500 [==============================] - 6s 4ms/step - loss: 0.3344 - accuracy: 0.8782
<keras.src.callbacks.History at 0x7c26afe9c250>
```

렐루 활성화 함수

```
model = keras.Sequential()
model.add(keras.layers.Flatten(input_shape=(28, 28)))
model.add(keras.layers.Dense(100, activation='relu'))
model.add(keras.layers.Dense(10, activation='softmax'))
```

```
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 flatten (Flatten)           (None, 784)               0         
                                                                 
 dense_4 (Dense)             (None, 100)               78500     
                                                                 
 dense_5 (Dense)             (None, 10)                1010      
                                                                 
=================================================================
Total params: 79510 (310.59 KB)
Trainable params: 79510 (310.59 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
(train_input, train_target), (test_input, test_target) = keras.datasets.fashion_mnist.load_data()

train_scaled = train_input / 255.0

train_scaled, val_scaled, train_target, val_target = train_test_split(
    train_scaled, train_target, test_size=0.2, random_state=42)
```

```
model.compile(loss='sparse_categorical_crossentropy', metrics=['accuracy'])

model.fit(train_scaled, train_target, epochs=5)
```

```
Epoch 1/5
1500/1500 [==============================] - 6s 3ms/step - loss: 0.5290 - accuracy: 0.8113
Epoch 2/5
1500/1500 [==============================] - 5s 3ms/step - loss: 0.3920 - accuracy: 0.8576
Epoch 3/5
1500/1500 [==============================] - 4s 3ms/step - loss: 0.3525 - accuracy: 0.8726
Epoch 4/5
1500/1500 [==============================] - 6s 4ms/step - loss: 0.3301 - accuracy: 0.8821
Epoch 5/5
1500/1500 [==============================] - 5s 3ms/step - loss: 0.3141 - accuracy: 0.8867
<keras.src.callbacks.History at 0x7c26b0c8ccd0>
```

```
model.evaluate(val_scaled, val_target)
```

```
375/375 [==============================] - 1s 2ms/step - loss: 0.3683 - accuracy: 0.8726
[0.3683287501335144, 0.8725833296775818]
```

옵티마이저

```
model.compile(optimizer='sgd', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

```
sgd = keras.optimizers.SGD()
model.compile(optimizer=sgd, loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

```
sgd = keras.optimizers.SGD(learning_rate=0.1)
```

```
sgd = keras.optimizers.SGD(momentum=0.9, nesterov=True)
```

```
adagrad = keras.optimizers.Adagrad()
model.compile(optimizer=adagrad, loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

```
rmsprop = keras.optimizers.RMSprop()
model.compile(optimizer=rmsprop, loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

```
model = keras.Sequential()
model.add(keras.layers.Flatten(input_shape=(28, 28)))
model.add(keras.layers.Dense(100, activation='relu'))
model.add(keras.layers.Dense(10, activation='softmax'))
```

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

model.fit(train_scaled, train_target, epochs=5)
```

```
Epoch 1/5
1500/1500 [==============================] - 7s 4ms/step - loss: 0.5262 - accuracy: 0.8153
Epoch 2/5
1500/1500 [==============================] - 5s 3ms/step - loss: 0.3946 - accuracy: 0.8596
Epoch 3/5
1500/1500 [==============================] - 5s 3ms/step - loss: 0.3544 - accuracy: 0.8709
Epoch 4/5
1500/1500 [==============================] - 6s 4ms/step - loss: 0.3277 - accuracy: 0.8794
Epoch 5/5
1500/1500 [==============================] - 5s 3ms/step - loss: 0.3062 - accuracy: 0.8871
<keras.src.callbacks.History at 0x7c26b0f06b90>
```

```
model.evaluate(val_scaled, val_target)
```

```
375/375 [==============================] - 1s 2ms/step - loss: 0.3440 - accuracy: 0.8769
[0.3439626097679138, 0.8769166469573975]
```