# 심층 신경망

구글 코랩에서 실행하기

## 2개의 층

In [1]:
```
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
```

In [2]:
```
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)
```

In [3]:
```
dense1 = keras.layers.Dense(100, activation='sigmoid', input_shape=(784,)) # 신경망의 첫번째 층은 input_shape의 매개변수로 반드시 입력해야 함
# 노드는 100, activate_func는 sigmoid,입력 크기는 784
dense2 = keras.layers.Dense(10, activation='softmax') # 10개의 클래스 분류, softmax
```

## 심층 신경망 만들기

In [4]:
```
model = keras.Sequential([dense1, dense2])
```

In [5]:
```
model.summary() # 각 층별로 정보 확인
```
```
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

In [6]:
```
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') # Sequential 클래스를 사용해서 층 추가하기
```

In [7]:
```
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)
_________________________________________________________________
```

In [8]:
```
model = keras.Sequential()
model.add(keras.layers.Dense(100, activation='sigmoid', input_shape=(784,)))
model.add(keras.layers.Dense(10, activation='softmax'))
```

In [9]:
```
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)
_________________________________________________________________
```

In [10]:
```
model.compile(loss='sparse_categorical_crossentropy', metrics='accuracy')

model.fit(train_scaled, train_target, epochs=5)
```
```
Epoch 1/5
1500/1500 [==============================] - 9s 3ms/step - loss: 0.5699 - accuracy: 0.8063
Epoch 2/5
1500/1500 [==============================] - 4s 3ms/step - loss: 0.4122 - accuracy: 0.8524
Epoch 3/5
1500/1500 [==============================] - 4s 3ms/step - loss: 0.3773 - accuracy: 0.8648
Epoch 4/5
1500/1500 [==============================] - 5s 3ms/step - loss: 0.3535 - accuracy: 0.8728
Epoch 5/5
1500/1500 [==============================] - 5s 4ms/step - loss: 0.3354 - accuracy: 0.8783
```
Out[10]:
```
<keras.src.callbacks.History at 0x7e7119923490>
```

## 렐루 활성화 함수

In [11]:
```
model = keras.Sequential()
model.add(keras.layers.Flatten(input_shape=(28, 28))) # flatten 클래스는 배치 차원을 제외하고 나머지 입력 차원을 모두 일렬로 펼치는 역할
model.add(keras.layers.Dense(100, activation='relu')) # activate func 를 relu로 설정
model.add(keras.layers.Dense(10, activation='softmax'))
```

In [12]:
```
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)
_________________________________________________________________
```

In [13]:
```
(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)
```

In [14]:
```
model.compile(loss='sparse_categorical_crossentropy', metrics='accuracy')

model.fit(train_scaled, train_target, epochs=5)
```
```
Epoch 1/5
1500/1500 [==============================] - 7s 4ms/step - loss: 0.5300 - accuracy: 0.8133
Epoch 2/5
1500/1500 [==============================] - 4s 3ms/step - loss: 0.3949 - accuracy: 0.8569
Epoch 3/5
1500/1500 [==============================] - 5s 3ms/step - loss: 0.3561 - accuracy: 0.8713
Epoch 4/5
1500/1500 [==============================] - 4s 3ms/step - loss: 0.3345 - accuracy: 0.8790
Epoch 5/5
1500/1500 [==============================] - 4s 3ms/step - loss: 0.3179 - accuracy: 0.8864
```
Out[14]:
```
<keras.src.callbacks.History at 0x7e710d941540>
```

In [16]:
```
model.evaluate(val_scaled, val_target)
```
```
375/375 [==============================] - 1s 2ms/step - loss: 0.3563 - accuracy: 0.8777
```
Out[16]:
```
[0.3562949299812317, 0.8777499794960022]
```

## 옵티마이저

In [17]:
```
model.compile(optimizer='sgd', loss='sparse_categorical_crossentropy', metrics='accuracy')
# 케라스는 다양한 종류의 경사 하강법 알고리즘 제공 -> optimizer
```

In [18]:
```
sgd = keras.optimizers.SGD() # SGD 클래스의 객체를 만듦
model.compile(optimizer=sgd, loss='sparse_categorical_crossentropy', metrics='accuracy')
```

In [19]:
```
sgd = keras.optimizers.SGD(learning_rate=0.1)
```

In [21]:
```
sgd = keras.optimizers.SGD(momentum=0.9, nesterov=True) # 모멘텀을 0.9로 설정, nesterov=True로 하여 네스테로프 모멘텀 최적화를 사용
```

In [22]:
```
adagrad = keras.optimizers.Adagrad() # adagrad 클래스 객체를 만듦
model.compile(optimizer=adagrad, loss='sparse_categorical_crossentropy', metrics='accuracy')
```

In [23]:
```
rmsprop = keras.optimizers.RMSprop() # RMSProp 클래스 객체를 만듦
model.compile(optimizer=rmsprop, loss='sparse_categorical_crossentropy', metrics='accuracy')
```

In [24]:
```
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'))
```

In [25]:
```
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics='accuracy')

model.fit(train_scaled, train_target, epochs=5)
```
```
Epoch 1/5
1500/1500 [==============================] - 7s 3ms/step - loss: 0.5338 - accuracy: 0.8151
Epoch 2/5
1500/1500 [==============================] - 4s 3ms/step - loss: 0.4019 - accuracy: 0.8556
Epoch 3/5
1500/1500 [==============================] - 5s 3ms/step - loss: 0.3582 - accuracy: 0.8703
Epoch 4/5
1500/1500 [==============================] - 4s 3ms/step - loss: 0.3332 - accuracy: 0.8777
Epoch 5/5
1500/1500 [==============================] - 4s 3ms/step - loss: 0.3126 - accuracy: 0.8857
```
Out[25]:
```
<keras.src.callbacks.History at 0x7e710d9b5720>
```

In [26]:
```
model.evaluate(val_scaled, val_target)
```
```
375/375 [==============================] - 1s 2ms/step - loss: 0.3577 - accuracy: 0.8698
```
Out[26]:
```
[0.35767918825149536, 0.8697500228881836]
```

In [ ]: