주교재: 혼자 공부하는 머신러닝 + 딥러닝,
저자 박해선, 한빛미디어 → 실습코드

인공 신경망

Hyuntae Cho

Dept. of Digital Media Engineering

Tongmyong University

동명대학교
TONGMYONG UNIVERSITY

신경망을 활용하여 분류를 해보자

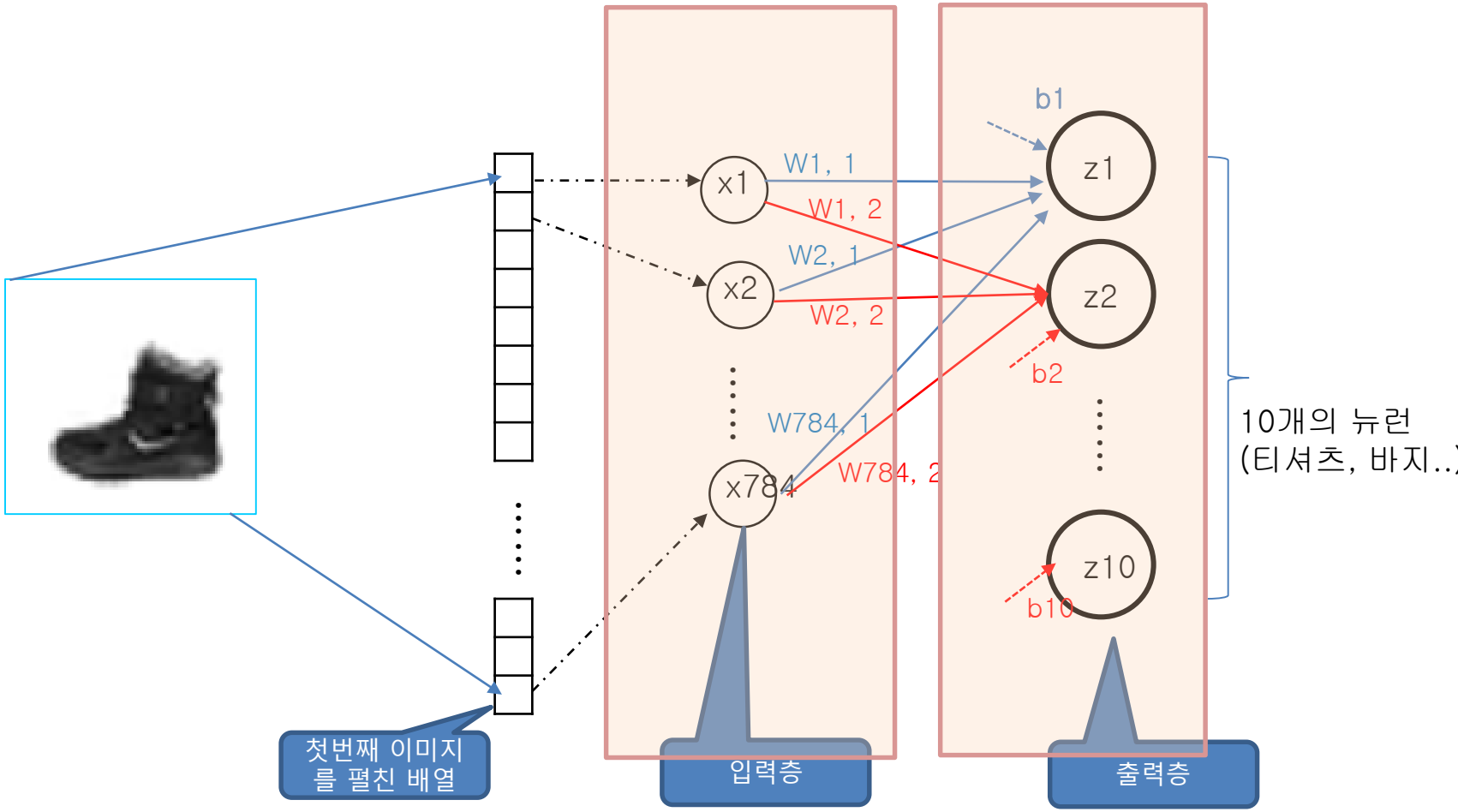
- **인공지능 모델로 만들어 보자**
  - 무작위로 뽑았을 경우, 각 아이템이 나올 확률을 계산

- **패션 아이템으로 이루어진 데이터 ' 패션 MNIST'**

- **각 이미지의 size = 28 x 28 pixels (784)**

인공 신경망 (ANN: Artificial Neural Network)

- **패션 아이템 분류를 인공 신경망으로 표현 해보자**
  - 10개의 class라 z1~z10까지 계산함 → 출력층이라 함.



인공 신경망 (ANN: Artificial Neural Network)

- **z1~z10를 계산하고 이를 바탕으로 class를 예측하기 때문에 “출력층 " 이라 함.**
  - 신경망의 최종값을 만든다는 의미
- **인공 신경망에서는 z값을 계산하는 단위를 뉴런 (neuron) 이라고 부름**
  - 유닛(unit)이라고도 부름
- **입력층**
  - x1~x784까지 픽셀값 자체를 나타냄
- **가중치**
  - x1에 곱해지는 가중치를 w1,1
- **절편**
  - 뉴런(z)마다 1개씩

케라스 API를 사용해 보기

- 데이터 준비

- 텐서플로 및 케라스

```
import tensorflow as tf


from tensorflow import keras
```

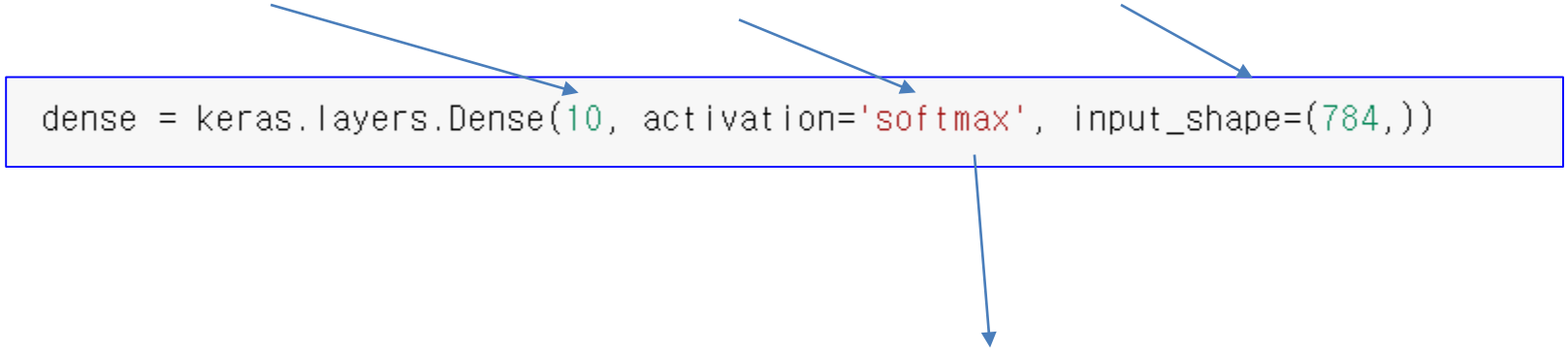
- 데이터 나누기 : 훈련 셋 vs. 테스트 셋

```
from sklearn.model_selection import train_test_split

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

케라스 API를 사용해 보기

- **1. 케라스의 dense 클래스를 이용하여 밀집층을 만들어 보자**
  - 필요 매개 변수:
    - 뉴런 개수, 뉴런의 출력에 적용할 함수, 입력의 크기



```
dense = keras.layers.Dense(10, activation='softmax', input_shape=(784,))
```

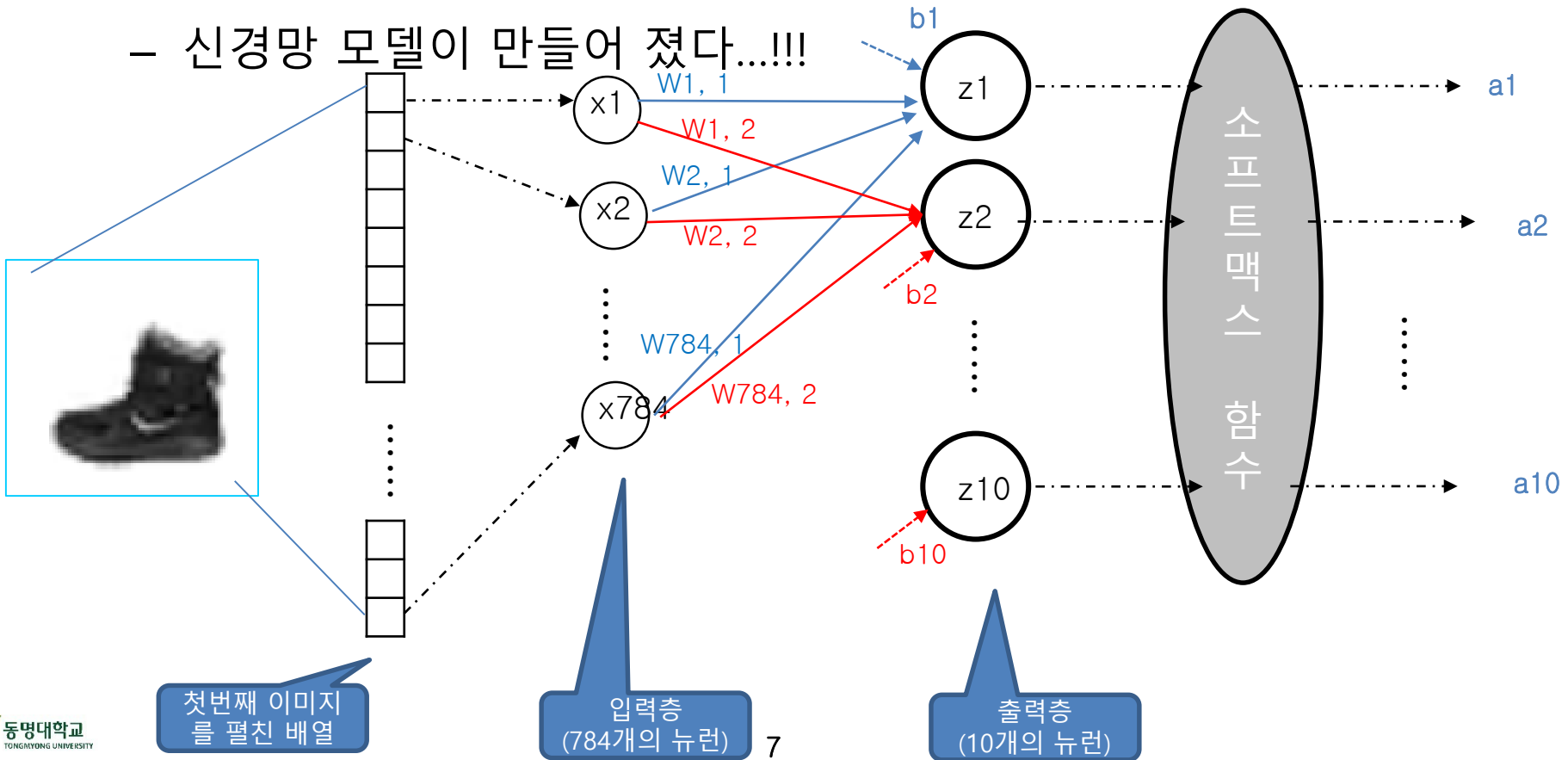
2진 분류일 경우 시그모이드 함수

케라스 API를 사용해 보기

- **2. 밀집층을 가진 신경망 모델을 만들자.**
  - 예: Keras의 sequential 클래스

```
model = keras.Sequential(dense)
```

  - 신경망 모델이 만들어 졌다...!!!



케라스 API를 사용해 보기

- **아이템 분류하기**
  - 사이킷 런과는 모델을 만드는 방식이 조금 다름
  - 3. 훈련하기 전에 model 객체의 **compile() 메소드를 통한 설정 단계**가 있음

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

- 이진분류: loss="binary_corssentropy"
- 다중분류: loss="categorical_corssentropy"

훈련할때
정확도 출력

케라스 API를 사용해 보기

- **아이템 분류하기**
  - 패션 MNIST 데이터의 타겟(정답)값은 어떻게 되어있나?
    - 처음 10개를 출력해보자

```
print(train_target[:10])
```

```
[7 3 5 8 6 9 3 3 9 9]
```

  - 4. 훈련

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

걸린시간 손실 정확도

```
Epoch 1/5
1500/1500 [==============================] - 5s 2ms/step - loss: 0.6096 - accuracy: 0.7932
Epoch 2/5
1500/1500 [==============================] - 2s 2ms/step - loss: 0.4790 - accuracy: 0.8395
Epoch 3/5
1500/1500 [==============================] - 2s 1ms/step - loss: 0.4564 - accuracy: 0.8472
Epoch 4/5
1500/1500 [==============================] - 2s 2ms/step - loss: 0.4436 - accuracy: 0.8529
Epoch 5/5
1500/1500 [==============================] - 2s 1ms/step - loss: 0.4370 - accuracy: 0.8555
<tensorflow.python.keras.callbacks.History at 0x7ff7f00685d0>
```

케라스 API를 사용해 보기

- 5. 평가

검증데이터

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

```
375/375 [==============================] - 1s 1ms/step - loss: 0.4434 - accuracy: 0.8529
[0.4433653652667999, 0.8529166579246521]
```

코드를 가지고 실습 해보자

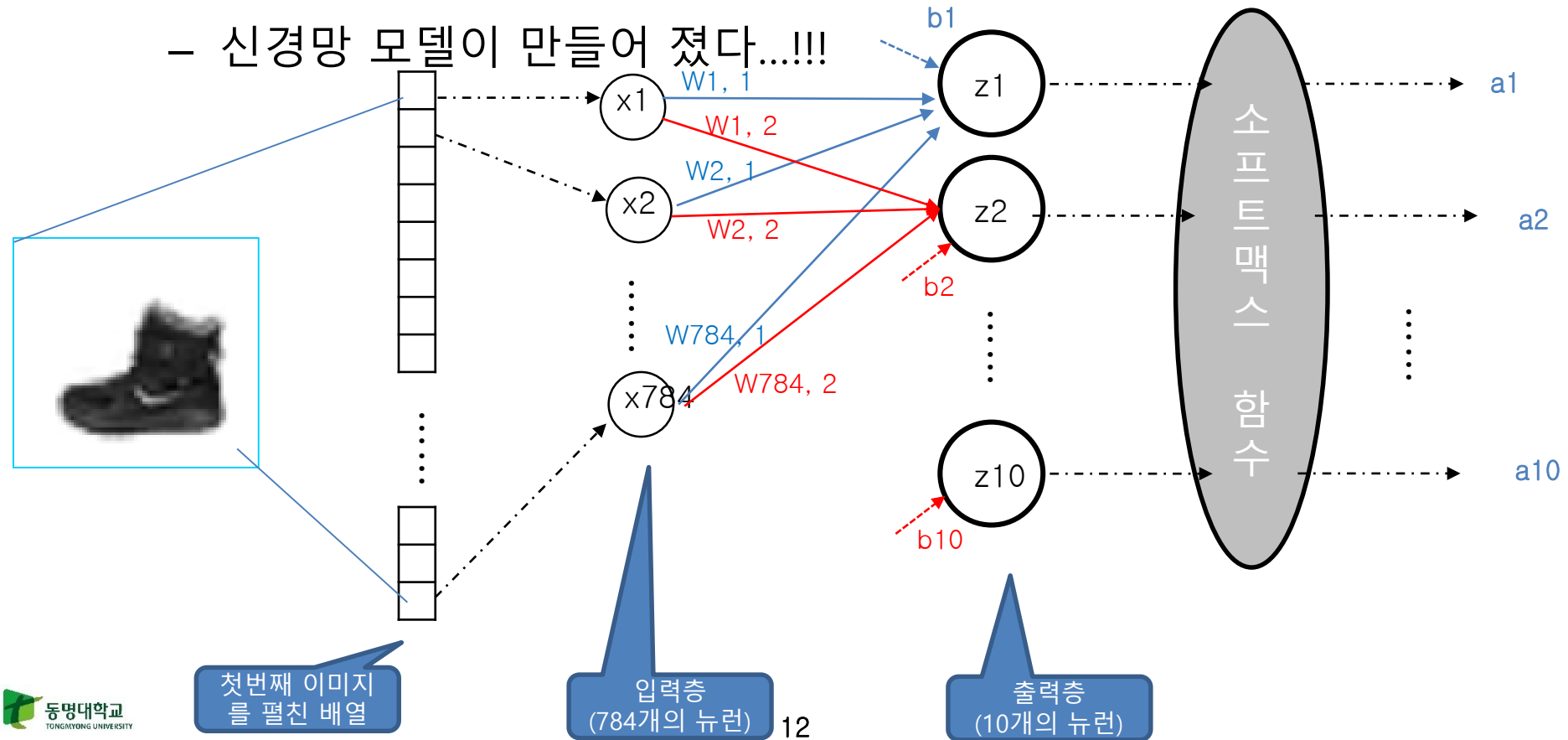
- **Github fork → Colab에 탑재하여 테스트**
  - Github:

신경망을 확장 해보자

- 지난 실습은 **1개의 층**으로 이루어짐.
  - 예: Keras의 sequential 클래스

```
model = keras.Sequential(dense)
```

  - 신경망 모델이 만들어 졌다...!!!



2개의 층 (layer)

- **심층 신경망을 만들어 성능을 높여보자**
  - 2개의 층부터~

- **1. 케라스 API를 사용하여 패션 MNIST 데이터 셋을 로드**

```
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
32768/29515 [================================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz
26427392/26421880 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz
8192/5148 [===============================================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz
4423680/4422102 [==============================] - 0s 0us/step
```

2개의 층 (layer)

- **2. 정규화, 2차원배열 → 1차원배열 변환, 데이터셋 분리 의 과정을 거침**

  – 0~255 사이의 픽셀값을 (0~1) 사이의 값으로..

```
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)
```

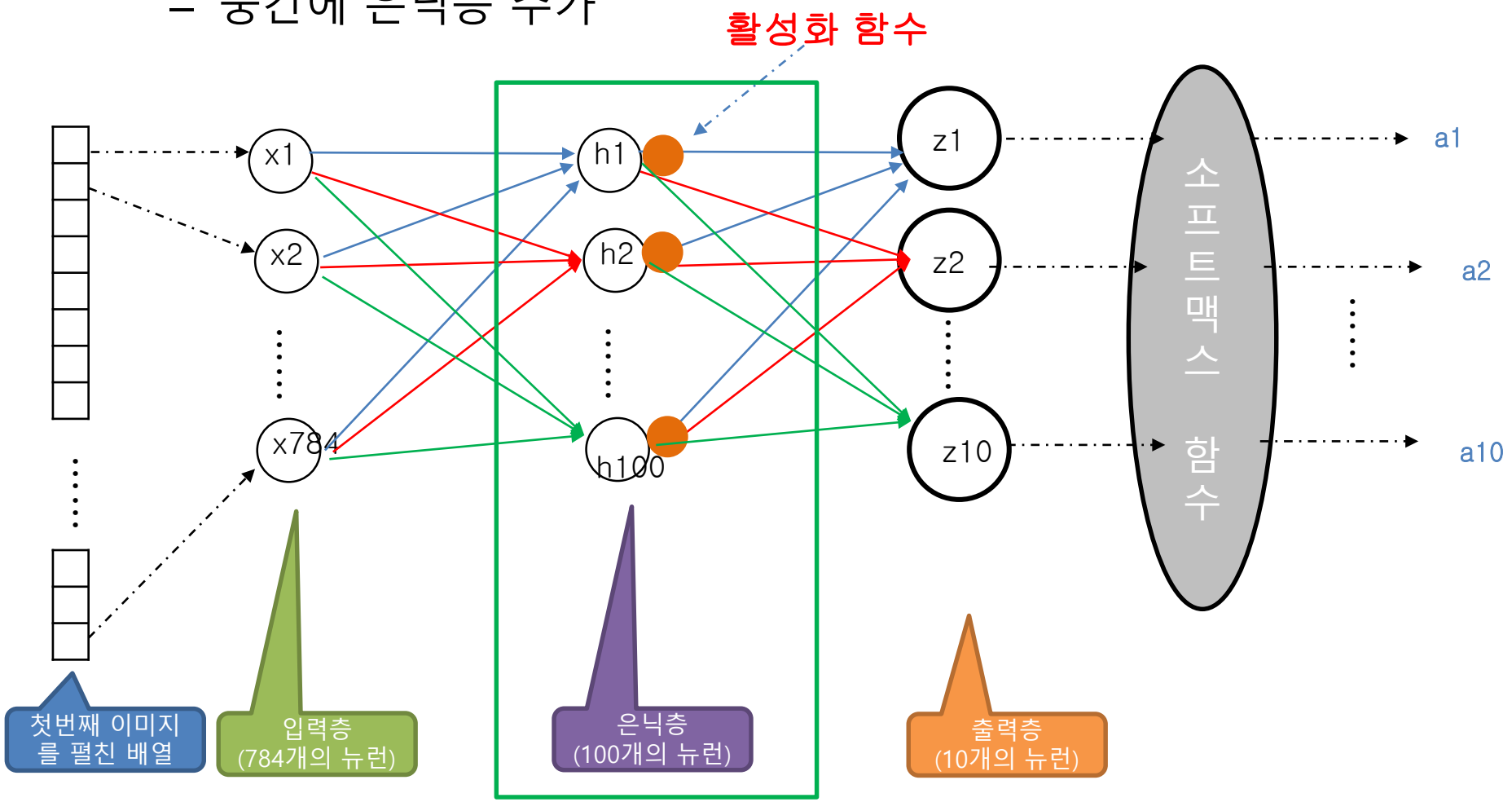
0~255사이의 픽셀값을 0~1사이의 값으로 변환

28x28 픽셀 크기 이미지를 784개의 1차원 배열로 변환

훈련 데이터 셋과 검증 데이터 셋으로 분리

2개의 층 (layer)

- 3. 신경망이 2개의 층을 가지도록 수정하자
  - 중간에 은닉층 추가



2개의 층 (layer) 신경망

- 시그모이드 활성화 함수를 사용한 은닉층
- 소프트맥스 함수를 사용한 출력층이 있는 신경망을 만들어 보자

은닉층

은닉층 개수

입력층

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

출력층

  - 은닉층의 개수에 따라 성능이 달라짐
  - 많은 경험이 필요함 그러나 출력층의 개수보다는 많아야 함.

2개의 층 (layer) 신경망

- **dense1(은닉층)과 dense2(출력층) 객체를 Sequential 클래스에 추가하여 심층 신경망을 생성**

출력층

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

  - 여러 개의 층을 만들때, 리스트[]로 만들어 인자값으로 전달해야 함
  - 출력층은 가장 마지막에 위치

- **이런 식으로 여러 개의 신경망 층을 추가 할 수 있음**

2개의 층 (layer) 신경망

- 케라스에서는 summary() 메소드를 통해 층에 대한 정보를 획득 가능

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 100)               78500
_________________________________________________________________
dense_1 (Dense)              (None, 10)                1010
=================================================================
Total params: 79,510
Trainable params: 79,510
Non-trainable params: 0
_________________________________________________________________
```

은닉층

출력층

784 입력 x 100개 은닉층 조합 + 뉴런마다 절편이 1개씩 (100)

100개 은닉층 조합 x 출력층 10개 뉴런

+ 출력층 뉴런마다 절편 1개씩 (10)

2개의 층 (layer) 신경망

- **이제 훈련을 시켜보자**
  - compile() 및 fit()
    - 층이 몇 개라도 같은 과정을 반복함

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

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

```
Epoch 1/5
1500/1500 [==============================] - 6s 2ms/step - loss: 0.5643 - accuracy: 0.8087
Epoch 2/5
1500/1500 [==============================] - 3s 2ms/step - loss: 0.4081 - accuracy: 0.8530
Epoch 3/5
1500/1500 [==============================] - 3s 2ms/step - loss: 0.3737 - accuracy: 0.8633
Epoch 4/5
1500/1500 [==============================] - 3s 2ms/step - loss: 0.3504 - accuracy: 0.8722
Epoch 5/5
1500/1500 [==============================] - 3s 2ms/step - loss: 0.3332 - accuracy: 0.8784
```

Epoch 단계별로 성능을 계산

코드를 가지고 실습 해보자

- **Github fork → Colab에 탑재하여 테스트**
  - Github: