머신러닝 vs 딥러닝

- 머신러닝
  1. 특징 추출을 인간이 해야함! => 머신러닝 모델 => 결과
  2. Scikit-learn(sklearn)
- 딥러닝
  1. 데이터 넣음 => 딥러닝(특징 추출) => 결과
  2. Tensorflow(KERAS) => 고수준 API
  3. PyTorch
     - 빅데이터분석기사, ADsP, tensorflow => kaggle Master

In [2]:

```
from tensorflow import keras
```

In [3]:

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

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-ker
as-datasets/train-labels-idx1-ubyte.gz (https://storage.googleapis.co
m/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz)
32768/29515 [================================] - 0s 0us/step
40960/29515 [========================================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-ker
as-datasets/train-images-idx3-ubyte.gz (https://storage.googleapis.co
m/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz)
26427392/26421880 [==============================] - 0s 0us/step
26435584/26421880 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-ker
as-datasets/t10k-labels-idx1-ubyte.gz (https://storage.googleapis.com/
tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz)
16384/5148 [===============================================================
==================================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-ker
as-datasets/t10k-images-idx3-ubyte.gz (https://storage.googleapis.com/
tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz)
4423680/4422102 [==============================] - 0s 0us/step
4431872/4422102 [==============================] - 0s 0us/step
```

2. 데이터 탐색

In [4]:

```
print(train_input.shape, train_target.shape)
```

```
(60000, 28, 28) (60000,)
```

In [5]:

```
print(test_input.shape, test_target.shape)
```

```
(10000, 28, 28) (10000,)
```

In [6]:

```
import matplotlib.pyplot as plt

fig, axs = plt.subplots(1, 10, figsize=(10,10))
for i in range(10):
  axs[i].imshow(train_input[i], cmap='gray_r')
  axs[i].axis('off')
plt.show()
```

In [7]:

```
import numpy as np
# 패션 mnist 는 0-9(10개의 패션 아이템 카테고리)
# 티셔츠, 바지, 스웨터, 드레스, 코트, 샌달, 셔츠, 스니커즈, 가방, 앵클부츠

print(np.unique(train_target, return_counts=True))
```

```
(array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9], dtype=uint8), array([6000, 600
0, 6000, 6000, 6000, 6000, 6000, 6000, 6000, 6000]))
```

3. 데이터 전처리

In [8]:

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

In [9]:

```
print(train_scaled.shape)
```

```
(60000, 784)
```

4. 인공신경망

In [10]:

```
import tensorflow as tf
from tensorflow import keras
```

In [11]:

```
from sklearn.model_selection import train_test_split

train_scaled, val_scaled, train_target, val_target = train_test_split(train_scaled,
```

In [15]:

```
print(train_scaled.shape, train_target.shape)  # 학습데이터를 train과 val로 나눔, 48000개
```

```
(48000, 784) (48000,)
```

In [16]:

```
print(val_scaled.shape, val_target.shape)  #12000개의 val 데이터
```

```
(12000, 784) (12000,)
```

In [17]:

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

layer가 하나만 있으면 shallow learning, layer가 최소 2개 이상 있어야 딥러닝 모델이라고 함

- layers: 신경망 층
- dense: 밀집 : 784개의 input과 10개의 뉴런이 모두 선으로 연결(위의 코드)

option

- 10: 뉴런 개수
- activation: 뉴런의 출력에 적용할 함수
- input_layer: 입력 크기

In [20]:

```
# model 생성 - sequential 모델, 순차적 모델
model = keras.Sequential(dense)
```

In [23]:

```
#loss = 손실함수(정답값과 예측값의 차이)
#metrics = 평가지표(n번 학습했을 때 정확도가 얼마인지 알려줌)
model.compile(loss="sparse_categorical_crossentropy", metrics="accuracy")
```

In [24]:

```
# 모델 학습
# fit(입력, 정답, 학습횟수)
# epochs: 에포크, 몇 번 학습할 건지
model.fit(train_scaled, train_target, epochs=5)
```

```
Epoch 1/5
1500/1500 [==============================] - 3s 2ms/step - loss: 0.612
5 - accuracy: 0.7914
Epoch 2/5
1500/1500 [==============================] - 2s 1ms/step - loss: 0.478
5 - accuracy: 0.8400
Epoch 3/5
1500/1500 [==============================] - 2s 2ms/step - loss: 0.456
8 - accuracy: 0.8471
Epoch 4/5
1500/1500 [==============================] - 2s 2ms/step - loss: 0.444
4 - accuracy: 0.8521
Epoch 5/5
1500/1500 [==============================] - 2s 2ms/step - loss: 0.437
1 - accuracy: 0.8550
```

Out[24]:

```
<keras.callbacks.History at 0x7f02da992090>
```

In [25]:

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

```
375/375 [==============================] - 1s 2ms/step - loss: 0.4768
- accuracy: 0.8438
```

Out[25]:

```
[0.47675248980522156, 0.843833327293396]
```

5. 심층신경망(딥러닝)

- 활성화 함수(activation function) : 신경망 층의 선형 방정식의 계산 값에 적용하는 함수
- 입력(X), 히든, 출력
- 출력층에 적용하는 활성화 함수는 제한적
- 분류문제에서는 클래스에 대한 확률을 출력하기 위해 활성화 함수(soft max) 사용
- 활성화 함수 사용 이용:

2개의 선형 방적식이 있음

- $a * 4 + 2 = b$
- $b * 3 - 5 = c$
- ==> $a * 12 + 1 = c$

> 선형식으로는 하나의 식으로 변경 가능, 층을 추가해도 의미가 없음, 활성화 함수(값을 비선형으로 바꿈) 사용!

=> 선형계산을 적당하게 비선형적으로 비틀어줌

- $a * 4 + 2 = b$
- $log(b) = k$

- $k * 3 - 5 = c$

In [42]:

```
# 조금 더 깊은 신경망 작성
# sigmoid: 가장 기본적인 활성화 함수(단점 많음, 사용 잘 안함)
# - 출력 z값을 0~1 사이의 값으로 압축!
# - activation function='relu' 고정적으로 많이 씀
dense1 = keras.layers.Dense(100, activation='sigmoid', input_shape=(784,))  #input 
dense2 = keras.layers.Dense(10, activation='softmax')  #10개의 층 추가??100개의 층 추가-교
```

In [33]:

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

In [34]:

```
model.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_6 (Dense)             (None, 100)               78500     

 dense_7 (Dense)             (None, 10)                1010      

=================================================================
Total params: 79,510
Trainable params: 79,510
Non-trainable params: 0
_________________________________________________________________
```

param = 계산횟수 78500번의 연산이 끝나야 한 번 끝난 것 100개 마다 각각의 bias 덧셈 = > 78400+100

- input data = 784개, 100번 붙으면 78400 + 100(bias)
- input: 784, hidden = 100, bias = 100

$784 * 100 + 100$

In [35]:

```
### 6. 층을 추가하는 다른 방법
```

In [36]:

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

In [37]:

```
model.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 hidden (Dense)              (None, 100)               78500

 output (Dense)              (None, 10)                1010

=================================================================
Total params: 79,510
Trainable params: 79,510
Non-trainable params: 0
_________________________________________________________________
```

In [39]:

```
# 2번째(가장 많이 사용)
model = keras.Sequential()
model.add(keras.Sequential(keras.layers.Dense(100, activation='sigmoid', input_shape
model.add(keras.layers.Dense(10, activation='softmax'))
```

In [41]:

```
model.compile(loss='sparse_categorical_crossentropy', metrics= 'accuracy')
model.fit(train_scaled, train_target, epochs=5)
```

```
Epoch 1/5
1500/1500 [==============================] - 5s 3ms/step - loss: 0.566
3 - accuracy: 0.8054
Epoch 2/5
1500/1500 [==============================] - 4s 3ms/step - loss: 0.406
5 - accuracy: 0.8543
Epoch 3/5
1500/1500 [==============================] - 4s 3ms/step - loss: 0.371
1 - accuracy: 0.8655
Epoch 4/5
1500/1500 [==============================] - 4s 3ms/step - loss: 0.348
7 - accuracy: 0.8734
Epoch 5/5
1500/1500 [==============================] - 4s 3ms/step - loss: 0.332
0 - accuracy: 0.8799
```

Out[41]:

```
<keras.callbacks.History at 0x7f02da4e0b50>
```

In [43]:

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

```
375/375 [==============================] - 2s 3ms/step - loss: 0.3499
- accuracy: 0.8747
```

Out[43]:

```
[0.34991133213043213, 0.874666690826416]
```

층 추가, 학습방법, loss 바꾸기.. 등을 이용해서 모델 튜닝 가능 가장 좋은 건 좋은 데이터를 사용하는 것