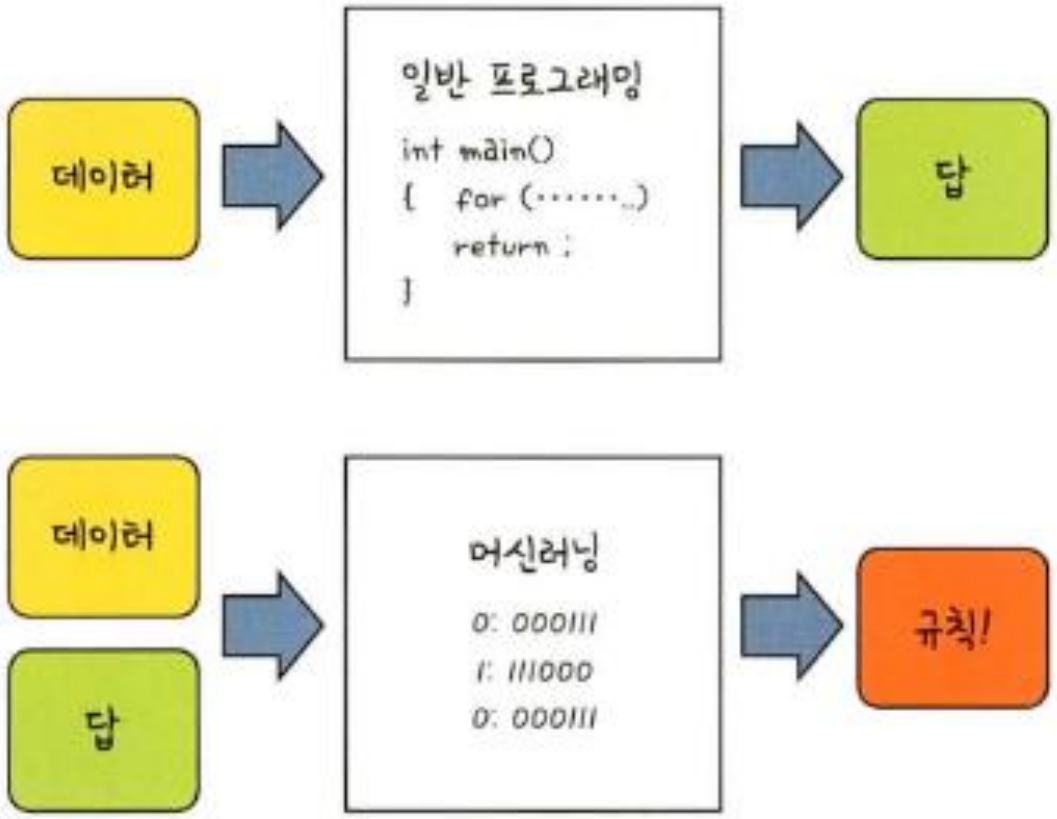
3. 처음 접하는 딥러닝

미지의 일을 예측하는 힘

- “기존 환자의 데이터를 이용해 새로운 환자의 상사를 예측하는 프로그램을 짜봐!”
- 이런 과제를 받았다고 해 봅시다.
- 기존의 프로그래밍 기법으로 이런 프로그램을 만들려면 쉽지 않습니다.
- 하지만 머신러닝은 이를 매우 쉽게 해결합니다.
- 기존에 우리가 했던 프로그래밍은 데이터를 입력해서 답을 구하는 데 초점이 맞춰져 있었다면, 머신러닝은 데이터 안에서 규칙을 발견하고 그 규칙을 새로운 데이터에 적용해서 새로운 결과를 도출하는 데 초점 이 맞춰져 있기 때문입니다.

미지의 일을 예측하는 힘

- 머신러닝은 기존 데이터를 이용해 아직 일어나지 않은 미지의 일을 예측하기 위해 만들어진 기법입니다
- 다음 그림은 이를 잘 설명해 줍니다.



미지의 일을 예측하는 힘

- 실제 예를 들어 머신러닝을 활용하는 방법에 대해 살펴보겠습니다.
- 중환자를 전문으로 수술하는 어느 병원의 의사가 수많은 환자를 수술해 오던 중 다음과 같은 질문을 던져 보았습니다
- “혹시 수술하기 전에 수술 후의 생존율을 수치로 예측할 방법이 없을까?"

미지의 일을 예측하는 힘

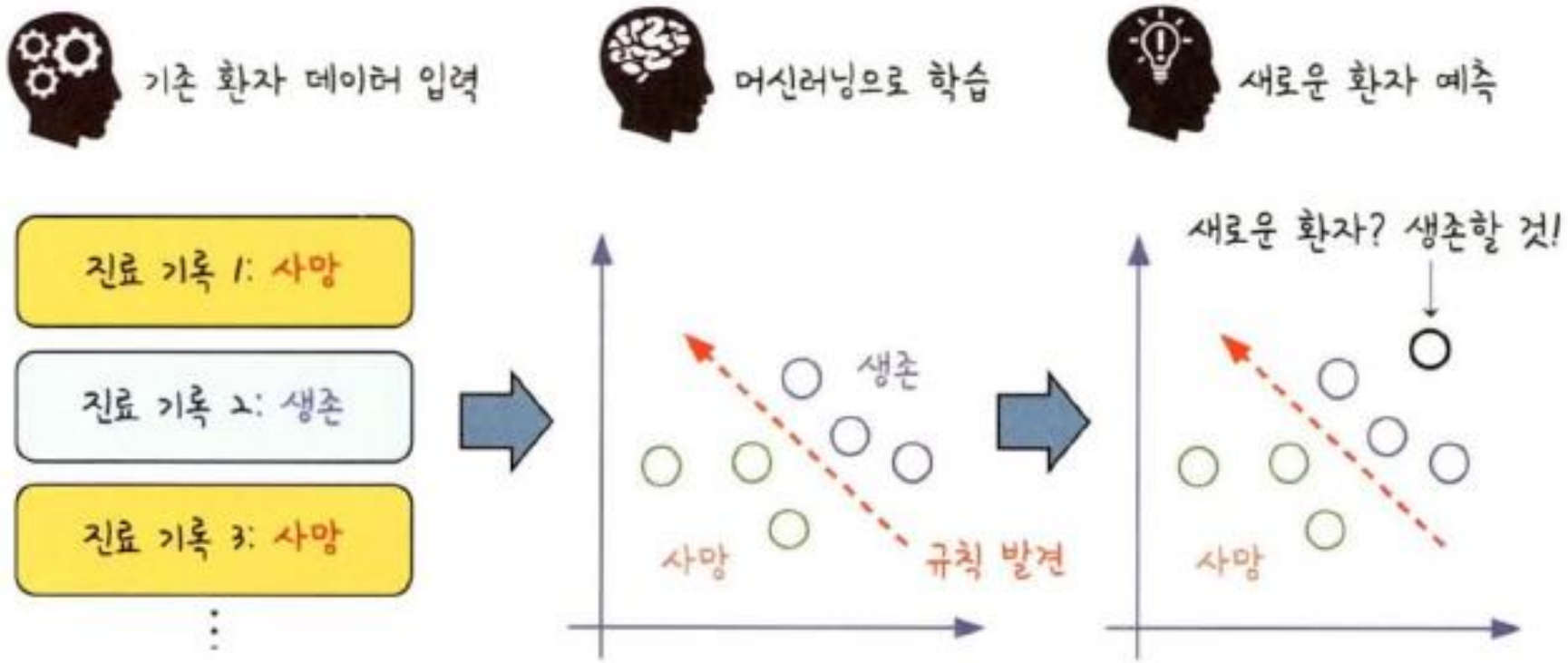
- 방법이 있습니다.
- 자신이 그동안 집도한 수술 환자의 수술 전 상태와 수술 후의 생존율을 정리해 놓은 데이터를 머신러닝 알고리즘에 넣는 것입니다.
- 기존 환자의 데이터는 머신러닝에 입력되는 순간 그 패턴과 규칙이 분석됩니다.
- 그리고 분석 결과를 새로운 환자의 데이터와 비교하여 생존 가능성이 몇 퍼센트(%) 인지 알려 줍니다.
- 이것이 바로 머신러닝이 하는 일입니다.

미지의 일을 예측하는 힘

- 여기서 데이터가 입력되고 패턴이 분석되는 과정을 학습(training)이라고 부릅니다
- 학습 과정을 다시 말하면 깨끗한 좌표 평면에 기존 환자들을 하나씩 배치하는 과정이라고 할 수 있습니다.
- 예를 들어 환자들의 분포를 그래프 위에 펼쳐 놓고 이 분포도 위에 생존과 사망 여부를 구분짓는 경계를 그려넣습니다
- 이를 잘 저장해 놓았다가 새로운 환자가 오면 분포도를 다시 꺼냅니다.
- 그리고 새 환자가 분포도의 어디쯤 위치하는지를 정합니다.

미지의 일을 예측하는 힘

- 그리고 아까 그려둔 생존과 사망 경계선을 기준으로 이 환자의 생사를 판단하는 것입니다.
- 이를 그림으로 표현하면 아래 그림과 같습니다.



미지의 일을 예측하는 힘

- 우리가 지금 배우려는 것이 바로 이러한 학습과 예측의 구체적인 과정입니다.
- 머신러닝의 예측 성공률은 결국 얼마나 정확한 경계선을 긋느냐에 달려 있습니다.
- 1950년대부터 학자들이 더 정확한 선을 긋고자 여러 가지 방법을 고안하였고, 그 결과 랜덤 포레스트(random forest, 서포트 벡터 머신 (support vector machines) 등 많은 방법들이 세상에 소개되었습니다.
- 딥러닝은 이러한 수많은 머신러닝 방법 가운데 가장 효과적인 방법입니다.

폐암수술 환자의 생존율 예측하기

- 이제 실제 딥러닝 코드를 보며 그 구성을 살펴보겠습니다.
- 불과 몇 년 전만 해도 딥러닝 알고리즘을 만들려면 코드를 길게 작성해야 했지만, 지금은 단 몇 줄의 코드로도 완벽히 구동할 수 있게 되었습니다.

폐암수술 환자의 생존율 예측하기

- 앞서 이야기한 수술환자의 생존율 예측 알고리즘을 실제로 만들어 보면 다음 코드와 같습니다.

```
from keras.models import Sequential
from keras.layers import Dense

import numpy
import tensorflow as tf

seed = 0
numpy.random.seed(seed)
tf.set_random_seed(seed)

Data_set = numpy.loadtxt("/content/gdrive/My Drive/data/ThoracicSurgery.csv", delimiter=",")

X = Data_set[ : , 0 : 17]
Y = Data_set[ : , 17]
```

폐암수술 환자의 생존율 예측하기

- 앞서 이야기한 수술환자의 생존율 예측 알고리즘을 실제로 만들어 보면 다음 코드와 같습니다.

```
model = Sequential()
model.add(Dense(30, input_dim=17, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='mean_squared_error', optimizer='adam', metrics=['accuracy'])
model.fit(X, Y, epochs=30, batch_size=10)

print("\n Accuracy: %.4f" % (model.evaluate(X, Y)[1]))
```

폐암수술 환자의 생존율 예측하기

- 이를 실행하면 다음과 같은 값이 화면에 출력됩니다.
- 출력 내용의 마지막 부분을 옮겨보면 다음과 같습니다.

```
Epoch 27/30
470/470 [==============================] - 0s 100us/step - loss: 0.1473 - acc: 0.8511
Epoch 28/30
470/470 [==============================] - 0s 95us/step - loss: 0.1471 - acc: 0.8489
Epoch 29/30
470/470 [==============================] - 0s 87us/step - loss: 0.1471 - acc: 0.8489
Epoch 30/30
470/470 [==============================] - 0s 104us/step - loss: 0.1462 - acc: 0.8532
470/470 [==============================] - 0s 70us/step

 Accuracy: 0.8511
```

폐암수술 환자의 생존율 예측하기

- 이를 실행하면 다음과 같은 값이 화면에 출력됩니다.
- 출력 내용의 마지막 부분을 옮겨보면 다음과 같습니다.

```
Epoch 27/30
470/470 [==============================] - 0s 100us/step - loss: 0.1473 - acc: 0.8511
Epoch 28/30
470/470 [==============================] - 0s 95us/step - loss: 0.1471 - acc: 0.8489
Epoch 29/30
470/470 [==============================] - 0s 87us/step - loss: 0.1471 - acc: 0.8489
Epoch 30/30
470/470 [==============================] - 0s 104us/step - loss: 0.1462 - acc: 0.8532
470/470 [==============================] - 0s 70us/step

 Accuracy: 0.8511
```

폐암수술 환자의 생존율 예측하기

- 여기서 눈여겨 봐야 할 값은 맨 아래 줄의 정확도 (Accuracy) 입니다.
- 정확도가 1.0이면 예측 정확도가 100%라는 뜻입니다.
- 만약 정확도가 0.8511 이라면 이는 이 스크립트를 통해 구현된 딥러닝의 예측 정확도가 85.11 %라는 뜻입니다.
- 다시 말해서, 기존 폐암 환자의 데이터를 딥러닝에 집어넣고 학습시킨 결과, 새로운 환자의 수술 후 생존을 10번 중 8~9 번은 맞힐 수 있다는 뜻입니다.

폐암수술 환자의 생존율 예측하기

- 단 몇 줄의 코드가 해내는 성과가 놀라워 보입니다.
- 하지만 아직은 놀랄 때가 아닙니다.
- 예측 성공률은 입력 데이터의 면밀한 분석과 프로젝트에 맞는 옵션을 가미할 때 더욱 높아지기 때문입니다.
- 그런데 이러한 '최적화 단계'를 제대로 해내려면 딥러닝의 구동 원리를 이해해야 합니다.
- 그리고 이 책을 통해 배울 내용은 곧 최적화를 잘하기 위한 준비 과정이기도 합니다.

딥러닝 코드 분석

- 첫 번째 부분: 데이터 분석과 입력
  - 먼저 다음은 데이터를 불러와서 사용할 수 있게 만들어 주는 부분입니다.

```
import numpy
(중략)

# 준비된 수술 환자 데이터를 불러들입니다
Data_set = numpy.loadtxt("/content/gdrive/My Drive/data/ThoracicSurgery.csv", delimiter=",")

# 환자의 기록과 수술 결과를 X와 Y로 구분하여 저장합니다
X = Data_set[:, 0:17]
Y = Data_set[:, 17]
```

딥러닝 코드 분석

- 첫 번째 부분: 데이터 분석과 입력
  - 그래서 코드 첫줄이 다음과 같이 시작합니다.

```
import numpy
```

  - 넘파이 (numpy) 라는 라이브러리를 불러오라는 뜻입니다.
  - 넘파이는 수치 계산을 위해 만들어진 라이브러리로 데이터 분석에 많이 사용됩니다.

  - 넘파이 라이브러리 안에는 여러 함수가 내장되어 있습니다.

```
Data_set = numpy.loadtxt("/content/gdrive/My Drive/data/ThoracicSurgery.csv", delimiter=",")
```

딥러닝 코드 분석

- 첫 번째 부분: 데이터 분석과 입력

```
Data_set = numpy.loadtxt("/content/gdrive/My Drive/data/ThoracicSurgery.csv", delimiter=",")
```

- 위 코드에서는 Data set 이라는 임시 저장소를 만들고 넘파이 라이브러리 안에 있는 loadtxt( ) 라는 함수를 사용했습니다.\
- 이를 이용해 ThoracicSurgery.csv'라는 외부 데이터 셋을 불러왔습니다.
- 머신러닝에서 알고리즘이나 좋은 컴퓨터 환경 만큼 중요한 것이 바로 제대로 된 데이터를 준비하는 일입니다.
- 따라서 매번 데이터를 정밀히 관찰하고 효율적으로 다루는 연습을 하는 것이 중요합니다.

딥러닝 코드 분석

- 첫 번째 부분: 데이터 분석과 입력
  - 우선은 첫 딥러닝에 사용할 데이터 인 'ThoracicSurgery. csv' 파일에 관해 좀 더 살펴보겠습니다.
  - 이 파일은 폴란드의 브로츠와프 의과대학에서 2013년 공개한 폐암 수술 환자의 수술 전 진단 데이터와 수술 후 생존 결과를 기록한 실제 의료 기록 데이터 입니다.
  - 이 파일을 열어보면 모두 470개의 라인으로 이루어져 있고 각 라인은 18개 항목으로 구분되어 있음을 알 수 있습니다.

딥러닝 코드 분석

- 첫 번째 부분: 데이터 분석과 입력
  - 이를 좀 더 알아보기 쉽게 정리하면 그림과 같습니다.

줄 / 항목	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	293	1	3.8	2.8	0	0	0	0	0	0	12	0	0	0	1	0	62	0
2	1	2	2.88	2.16	1	0	0	0	1	1	14	0	0	0	1	0	60	0
3	8	2	3.19	2.5	1	0	0	0	1	0	11	0	0	1	1	0	66	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
470	447	8	5.2	4.1	0	0	0	0	0	0	12	0	0	0	0	0	49	0

딥러닝 코드 분석

- 첫 번째 부분: 데이터 분석과 입력
  - 한 줄 한 줄이 서로 다른 환자의 상태를 기록한 정보입니다.
  - 총 470행이므로 470명의 환자에 대한 정보입니다.
  - 쉼표(,)로 이어진 항목이 각 줄마다 18개가 있습니다
  - 이는 환자마다 18개의 정보를 순서에 맞춰 정리했다는 뜻입니다.
  - 앞의 17개 정보는 종양의 유형, 폐활량, 호흡 곤란 여부, 고통 정도, 기침, 흡연, 천식 여부 등 17가지의 환자 상태를 조사해서 기록해 놓은 것입니다.
  - 그리고 마지막 18 번째 정보는 수술 후 생존 결과입니다.

딥러닝 코드 분석

- 첫 번째 부분: 데이터 분석과 입력
  - 1 또는 0 이라는 값이 많이 보이는데, 1은 true, 즉 해당 사항이 있다는 것을 뜻하고 0은 true, 즉 해당 사항이 없다는 것을 뜻합니다.
  - 맨 마지막 열인 18 번째 항목에서 1은 수술 후 생존했음을, 0은 수술 후 사망했음을 의미합니다.
  - 이번 프로젝트의 목적은 1번째부터 17 번째까지의 항목을 분석해서 18 번째 항목, 즉 수술 후 생존 또는 사망을 맞히는 것입니다.
  - 1번째 항목부터 17번째 항목까지를 '속성(attribute)'이라고 하고 정답에 해당하는 18 번째 항목을 '클래스(class)'라고 합니다.

딥러닝 코드 분석

- 첫 번째 부분: 데이터 분석과 입력
  - 딥러닝을 구동시키려면 '속성'만을 뽑아 데이터셋을 만들고, '클래스'를 담는 데이터 셋을 또 따로 만들어 줘야합니다.
  - 속성 데이터셋 X를 다음과 같이 생성합니다.

```
X = Data_set[:, 0:17]
```

  - 클래스 데이터셋 Y는 18 번째 항목을 이용해 다음과 같이 만들어 줍니다.

```
Y = Data_set[:, 17]
```

딥러닝 코드 분석

- 두 번째 부분: 딥러닝 실행

```
from keras.models import Sequential
from keras.layers import Dense

(중략)

model = Sequential()
model.add(Dense(30, input_dim=17, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='mean_squared_error', optimizer='adam', metrics=['accuracy'])
model.fit(X, Y, epochs=30, batch_size=10)
```

  - 이제 딥러닝을 실제로 실행하는 부분입니다.
  - 본 강의에서는 주로 케라스(keras)를 사용해 딥러닝을 실행시킵니다

딥러닝 코드 분석

- 두 번째 부분: 딥러닝 실행
  - 케라스가 구동되려면 텐서플로(TensorFlow) 또는 씨아노(theano)이라는 두 가지 라이브러리 중 하나가 미리 설치되어 있어야 하는데, 우리는 이 중 텐서플로를 선택해서 실습하겠습니다
  - 딥러닝 프로젝트를 '여행'으로 비유해 본다면 텐서플로는 목적지까지 빠르게 이동시켜주는 '비행기'에 해당하고, 케라스는 비행기의 이륙 및 정확한 지점까지의 도착을 책임지는 '파일럿'에 비유할 수 있습니다.

딥러닝 코드 분석

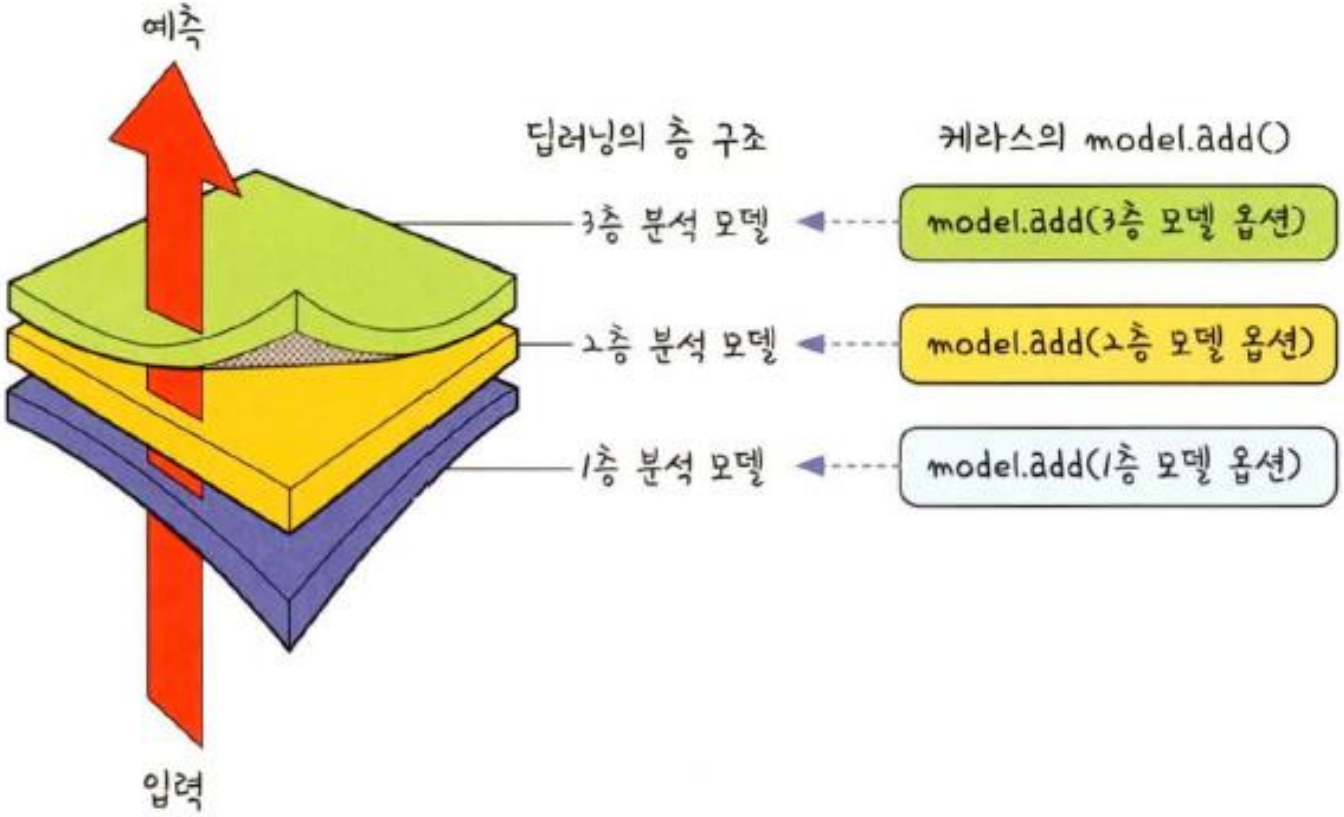
- 두 번째 부분: 딥러닝 실행
  - 설치가 모두 올바로 되었다면 다음과 같은 방법으로 설치된 케라스 라이브러리를 불러올 수 있습니다.

```
from keras.models import Sequential
from keras.layers import Dense
```

  - 케라스 라이브러리 중에서 Sequential 함수와 Dense 함수를 불러왔습니다.

딥러닝 코드 분석

- 두 번째 부분: 딥러닝 실행
  - 다음 장에서 자세히 다루겠지만, 딥러닝은 아래 그림과 같이 여러 층이 쌓여 결과를 만들어 냅니다.



딥러닝 코드 분석

- 두 번째 부분: 딥러닝 실행
  - Sequential 함수는 딥러닝의 구조를 한층 한층 쉽게 쌓아올릴 수 있게 해줍니다.
  - Sequential 함수를 선언하고 나서 model.add() 함수를 사용해 필요한 층을 차례로 추가하면 됩니다.
  - 우리가 살펴보고 있는 코드에서는 model.add() 함수를 이용해 두 개의 층을 쌓아 올렸습니다.

```
model = Sequential()
model.add(Dense(30, input_dim=17, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

딥러닝 코드 분석

- 두 번째 부분: 딥러닝 실행
  - 층을 몇 개 쌓을지는 데이터에 따라 그때 그때 결정합니다.
  - 케라스의 가장 큰 장점 중 하나는 model.add() 함수를 이용해 필요한 만큼의 층을 빠르고 쉽게 쌓아 올릴 수 있다는 것입니다.
  - model.add () 안에는 Dense() 함수가 포함되어 있습니다.
  - 영어 단어 dense는 '조밀하게 모여있는 집합'이란 뜻으로, 여기서는 각 층이 제각각 어떤 특성을 가질지 옵션을 설정하는 역할을 합니다

딥러닝 코드 분석

- 두 번째 부분: 딥러닝 실행
  - 딥러닝의 구조와 층별 옵션을 정하고 나면 compile() 함수를 이용해 이를 실행시킵니다.

```
model.compile(loss='mean_squared_error', optimizer='adam', metrics=['accuracy'])
model.fit(X, Y, epochs=30, batch_size=10)
```

  - loss, optimizer 같은 키워드가 나옵니다
    - activation: 다음 층으로 어떻게 값을 넘길지 결정하는 부분입니다. 여기서는 가장 많이 사용되는 relu와 sigmoid 함수를 사용하게끔 지정하고 있습니다.
    - loss: 한 번 신경망이 실행될 때마다 오차 값을 추적하는 함수입니다.
    - optimizer: 오차를 어떻게 줄여 나갈지 정하는 함수입니다.

딥러닝 코드 분석

- 마지막 부분: 결과 출력

```
print("\n Accuracy: %.4f" % (model.evaluate(X,Y)[1]))
```

  - 출력 부분에서는 model.evaluate() 함수를 이용해 앞서 만든 딥러닝의 모델이 어느 정도 정확하게 예측하는지를 점검할 수 있습니다.
  - 이 코드를 통해 출력되는 정확도(Accuracy)는 학습대상이 되는 기존 환자들의 데이터 중에 일부를 랜덤하게 추출해 새 환자인 것으로 가정하고 테스트한 결과입니다.
  - 좀 더 신뢰할 수 있는 정확도를 측정하려면, 학습단계에서 미리 일부를 떼어내어 태스트셋으로 저장하고 테스트할 때는 오직 이 테스트셋만을 사용합니다.

블랙 박스를 극복하려면?

- 지금까지 간단하게나마 딥러닝 프로젝트의 골격을 확인해 보았습니다.
- 케라스 라이브러리 덕분에 딥러닝을 구현하는 것이 그리 어렵지 않아 보입니다.
- 하지만 딥러닝이 어떤 원리로 작동되고 딥러닝 내부에서 어떤 방식으로 결과를 도출하는지를 알지 못하면 딥러닝은 속 안을 들여다 볼 수 없는 캄캄한 '블랙 박스'가 되어 버립니다.
- 결과를 도출하긴 했지만 이 결과가 어떻게 나왔는지를 설명하지 못하면 더 나은 결과를 만들기가 어렵겠지요?

블랙 박스를 극복하려면?

- 이 것이 바로 딥러닝이라는 '블랙박스'를 열어 그 안에서 구동되는 여러 가지 원리를 공부해야 하는 이유입니다.
- 다음 장에서부터 설명할 선형 회귀, 로지스틱 회귀, 신경망, 역전파 등을 배우다보면 블랙박스 안이 조금씩 보이기 시작할 것입니다.

Q&A