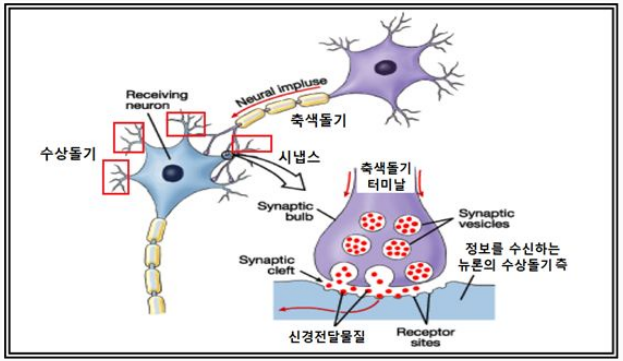
Deep Learning with Tensorflow Keras

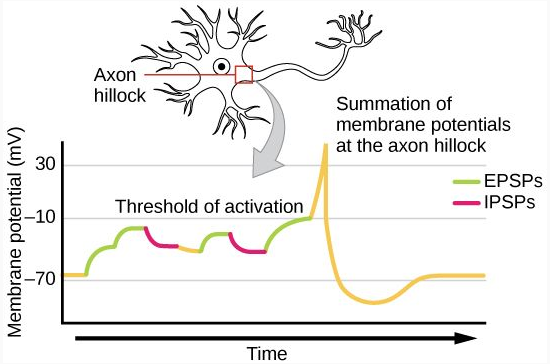
2강 인공신경망 기초

신유주

뉴런(Neuron)

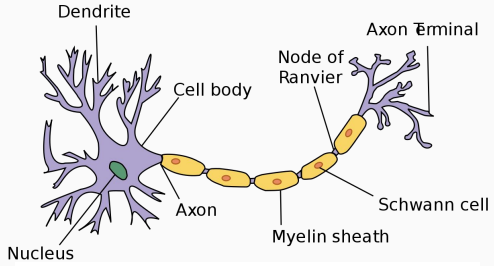
- 신호 수신용인 수상돌기의 시냅스 영역, 내보내는 축색돌기가 있는 뉴런
- 뉴런의 수신 신호량이 일정 수준 초과시 축색돌기 통해 다른 뉴런에 전파
  - All-or-none law
  - 역치를 넘지 않으면 아예 반응하지 않고, 역치를 넘으면 최대한 반응함

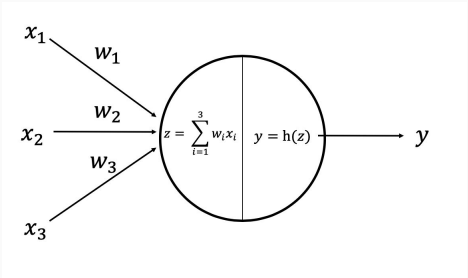




인공신경망 (Artificial Neural Network)

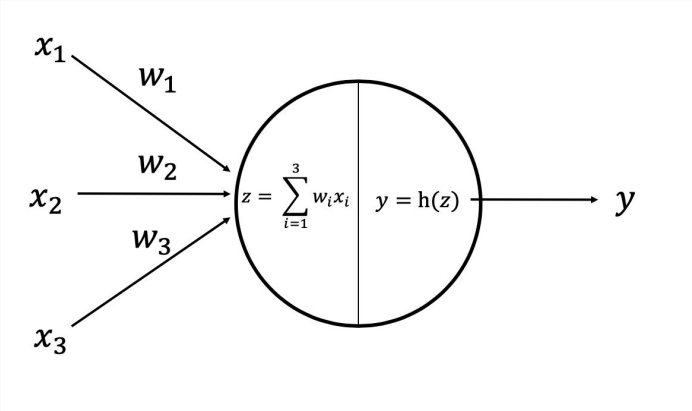
- 생물학적 신경망을 흉내내어 복잡한 의사결정을 학습하도록 함
- 실제 뇌에 있는 신경망과 차이가 존재함
  - 단순한 네트워크
  - 분자 전달이 없는 수학적 모델
- 그러나 비슷한 구조를 가짐
  - 수상돌기
  - 축색돌기





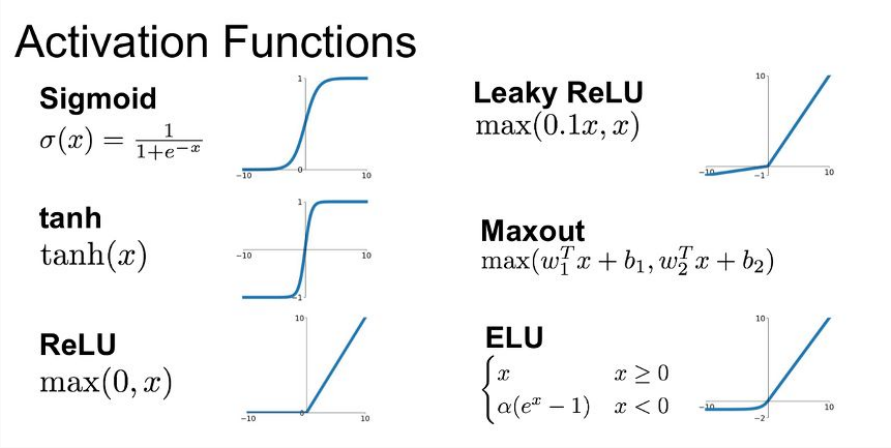
퍼셉트론(Perceptron)

- 인공신경망의 한 형태로, 여러 신호를 선형 결합 후(x1*w1+…+w3*w3) 활성화 함수를 거쳐서 나온 값(y)을 출력
- 활성화 함수는 다양한 '비'선형적 함수가 사용됨



활성화 함수(Activation Function)

- 비선형 함수를 사용하는 이유: 선형 함수를 사용하면 은닉층이 여러 개여도 학습에 아무런 영향을 주지 못함
  - 이는 선형함수에 의해 여러층의 퍼셉트론 연산이 한 층의 퍼셉트론 연산과 같아지기 때문

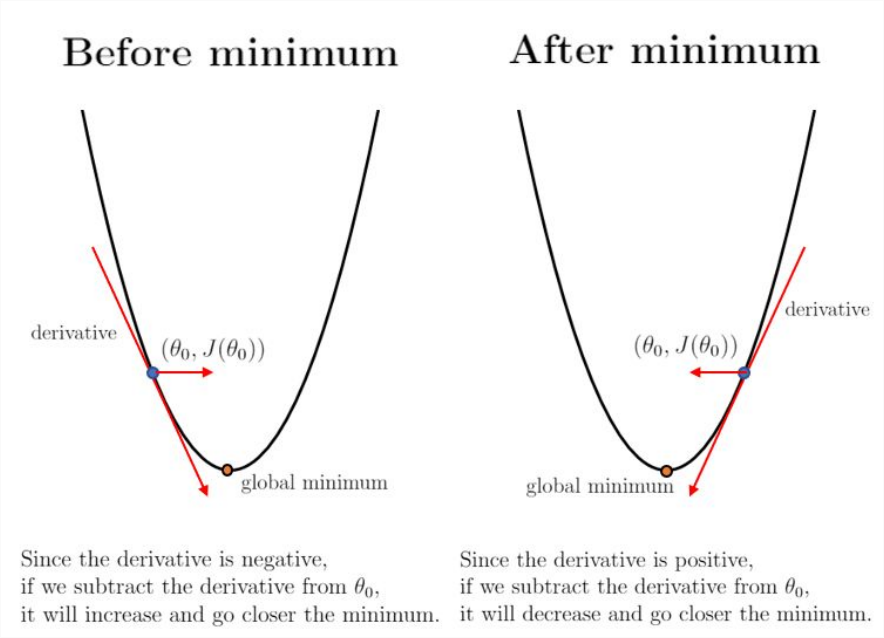


손실함수(Loss Function)

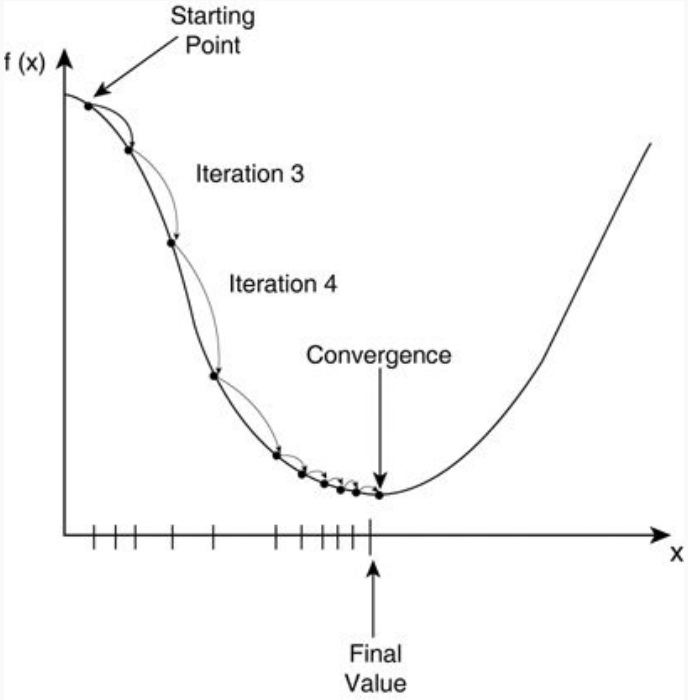
- 머신러닝 모델이 최소화하고자 하는 함수
  - Linear regression에서의 MSE(실제 y값과 모델에서 얻은 y값의 차이를 제곱해서 더한 것)
- 머신러닝 모델이 원하는 y 값과 가장 유사하게 나오도록 손실함수를 디자인
- 학습을 통해서 손실함수를 최소화하는 모델 파라미터를 탐색함
- 여기서 학습은 경사하강법을 이용함

$$\min_{\theta} \sum_{k=1}^{N} \|\mathbf{y}_k - f_\theta(\mathbf{x}_k)\|^2$$

경사하강법

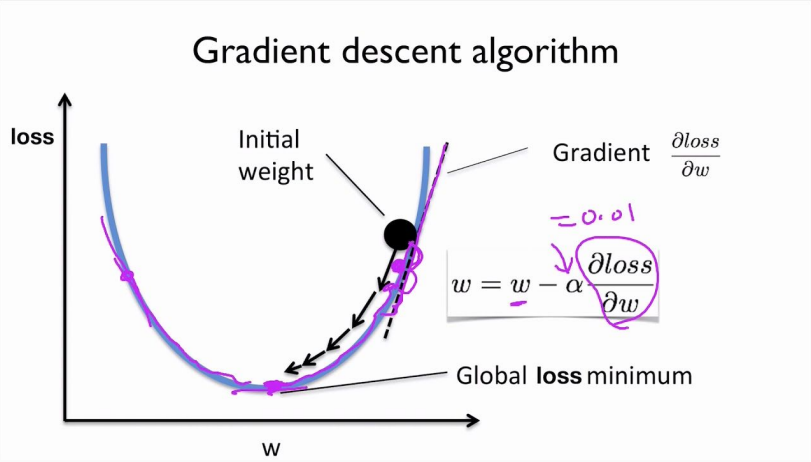

Before minimum
derivative
$(\theta_0, J(\theta_0))$
global minimum
Since the derivative is negative,
if we subtract the derivative from θ_0,
it will increase and go closer the minimum.
After minimum
derivative
$(\theta_0, J(\theta_0))$
global minimum
Since the derivative is positive,
if we subtract the derivative from θ_0,
it will decrease and go closer the minimum.

경사하강법(Gradient Descent)



- 손실 함수의 최소값을 알기 위해 딥러닝에서 사용하는 최적화기법
- 통계적 경사하강법(Stochastic Gradient Descent)를 통해 데이터 일부만 가지고 경사를 계산한 후 이 경사값을 바탕으로 모델의 파라미터를 업데이트함

통계적 경사하강법(Stochastic Gradient Descent)

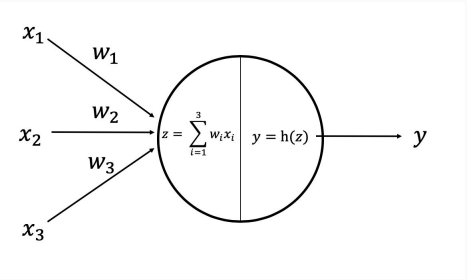


$$\min_{\theta} \sum_{k=1}^{N} \|\mathbf{y}_k - f_\theta(\mathbf{x}_k)\|^2$$

- loss: 손실함수로, 데이터셋 일부만 가지고 계산하게 됨
  - 이 데이터셋 일부를 mini-batch라고 부름
  - 일부만 사용하는 이유는 딥러닝에 쓰이는 데이터 크기가 너무 커서 한번에 경사 하강을 하는 데 필요한 계산량이 너무 많기 때문
- α: Learning rate
  - Gradient의 크기를 조절해 파라미터가 학습되는 속도를 결정함
- N: mini-batch의 크기

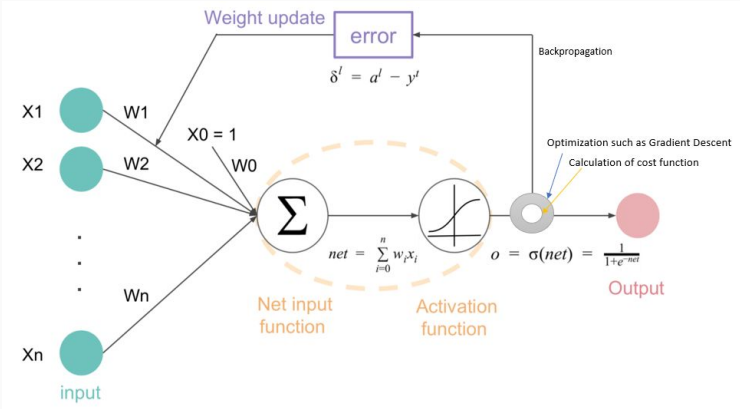
순방향 전파(Forward Propagation)

- 입력받은 신호(데이터)를 기반으로
- 입력층에서 출력층으로 퍼셉트론 작동
  - 입력 데이터값에 가중치를 곱해 더한 후 활성화함수를 거쳐 y 출력
  - Inference라고도 함
- 출력층에는 인공지능 모델이 내린 '결론' 존재
- '결론'과 실제 값과의 차이(틀린 정도): Loss
- '결론'을 만드는 Weight를 변화시키는 정도는
- Loss의 양에 따라서 결정됨(많이 틀린다: 많이 배운다)



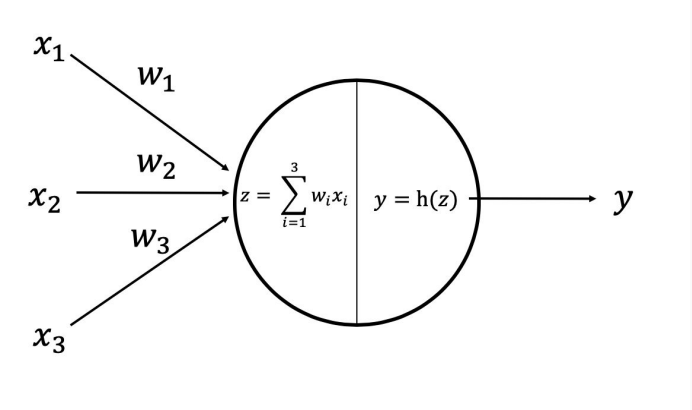
역방향 전파(Backward Propagation)

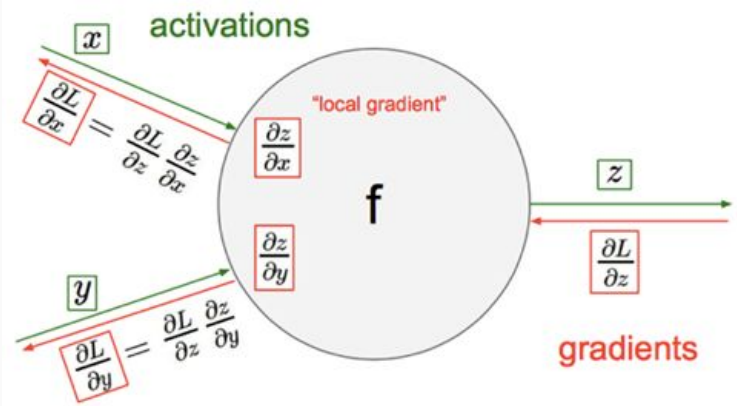
- 출력값(인공지능의 '결론')은 일련의 수식 계산
- 출력값 미분=일련의 수식 계산 미분
  =합성함수의 미분
  =구성하는 각 함수의 미분의 곱
- 연쇄 법칙을 활용하여 미분값 계산 가능
- 깊어질수록 많은 미분값을 곱하게 됨



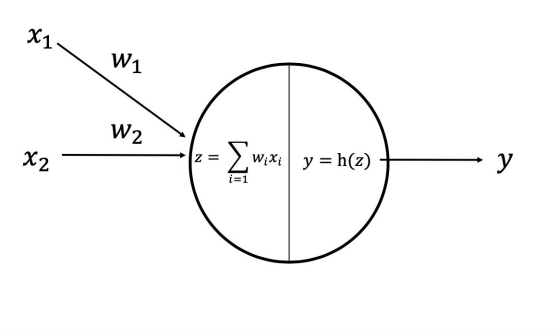
- 미분값 * Loss * Learning rate = 파라미터 업데이트해야할 양

역방향 전파(Backward Propagation)

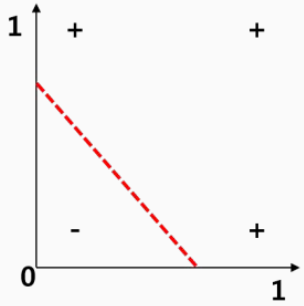




퍼셉트론의 한계

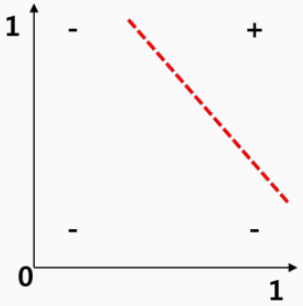


OR



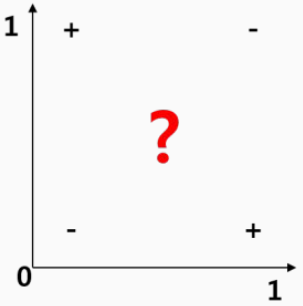
x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

AND



x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

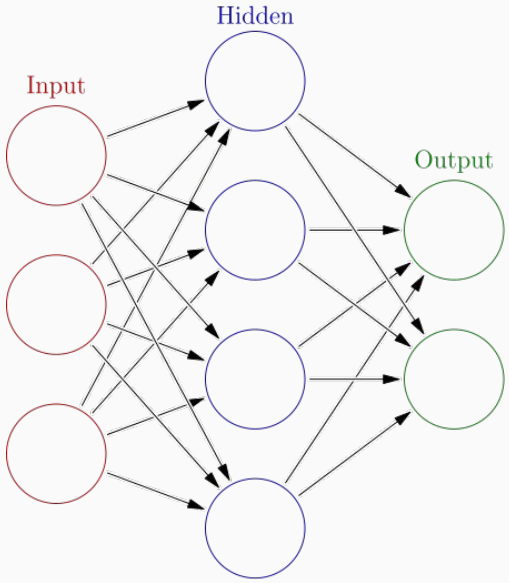
XOR



x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

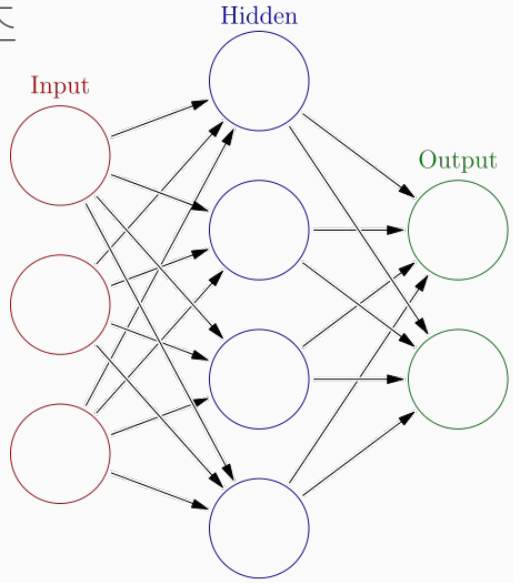
다층 퍼셉트론(Multi-layer Perceptron)

- 퍼셉트론 여러개를 층 구조로 쌓을 수 있음
  - 데이터가 들어오는 입력층
  - 입력층과 출력층 사이에 있는 은닉층
  - 결과를 출력하는 출력층
  - **= Dense Layer = Fully connected network**
- 심층신경망(**Deep** Neural Network)
  - 은닉층이 1개 이상 존재하는 인공신경망
- 심층신경망을 이용한 기계학습
  = 딥러닝(Deep Learning)

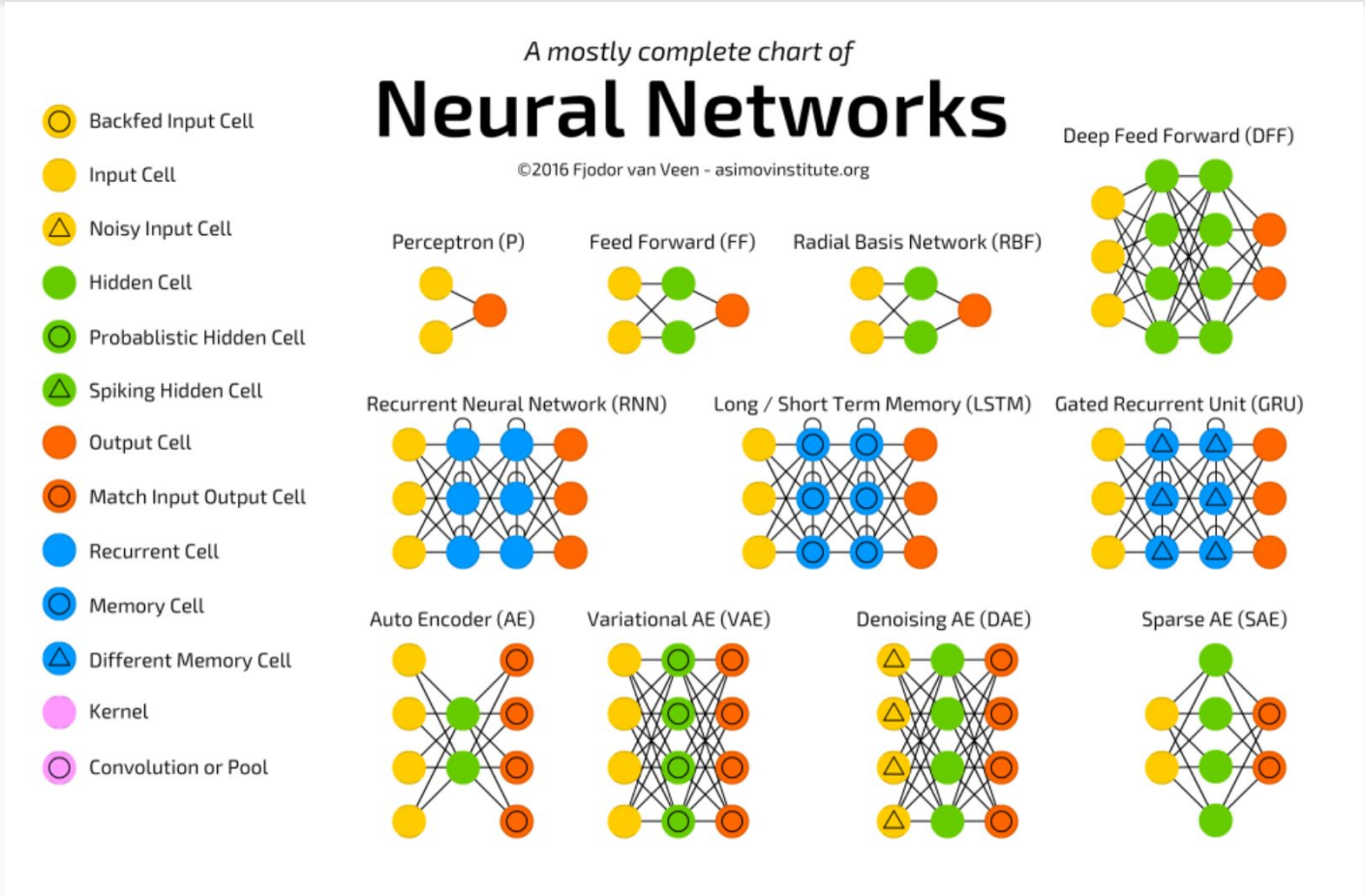


완전연결신경망(Fully Connected NN)

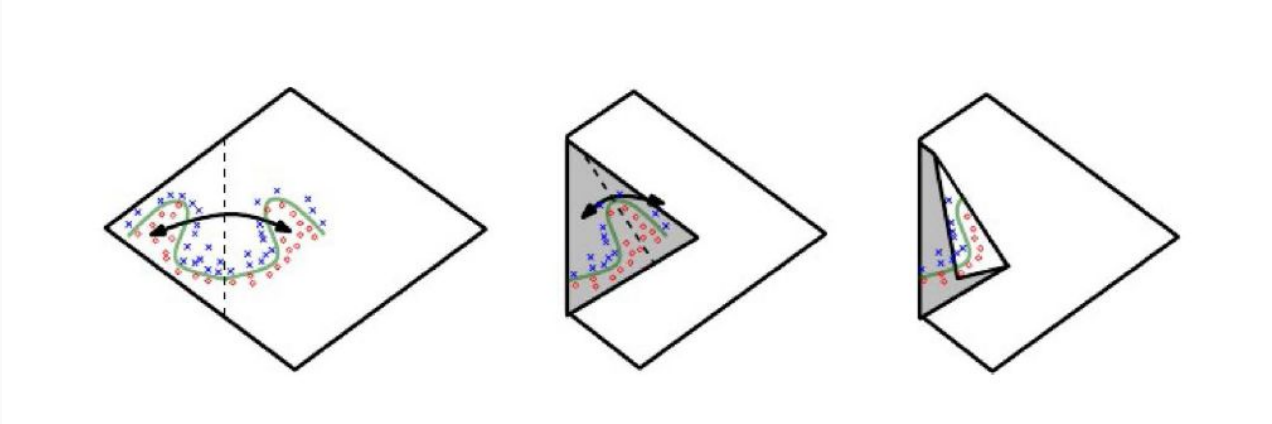
- 파라미터 수가 많은 편에 속하는 기본적인 신경망 구조
- 은닉층에 들어가는 뉴런의 수는 제한이 없음
  - 너무 많아지면 Overfitting 발생할 수 있음
- 은닉층 수도 바꿀 수 있음
  - 깊어지면 깊어질수록 보통 성능이 좋아짐
  - 하지만 역시 파라미터가 너무 많아져 overfitting 발생가능
  - 또한 **Gradient Diminishing** 현상 나타남
- 파라미터 수가 너무 많으면 학습도 느려짐
  - 역전파시 파라미터 수만큼 업데이트해야함



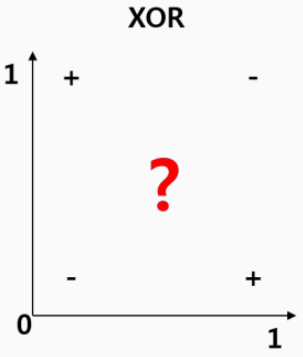
DNN 아키텍쳐 종류


A mostly complete chart of
Neural Networks
©2016 Fjodor van Veen - asimovinstitute.org
Backfed Input Cell
Input Cell
Noisy Input Cell
Hidden Cell
Probablistic Hidden Cell
Spiking Hidden Cell
Output Cell
Match Input Output Cell
Recurrent Cell
Memory Cell
Different Memory Cell
Kernel
Convolution or Pool
Perceptron (P)
Feed Forward (FF)
Radial Basis Network (RBF)
Deep Feed Forward (DFF)
Recurrent Neural Network (RNN)
Long / Short Term Memory (LSTM)
Gated Recurrent Unit (GRU)
Auto Encoder (AE)
Variational AE (VAE)
Denoising AE (DAE)
Sparse AE (SAE)

DNN의 작동원리



layer 1 layer 2 layer 3

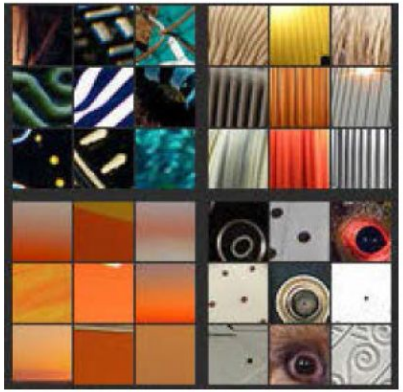


x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

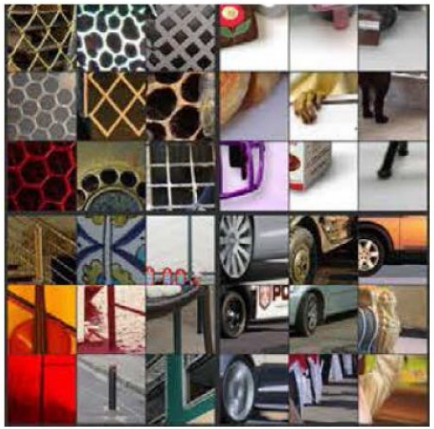
DNN의 작동원리 2

Layer 1

Layer 2



Layer 3



어제 코드 다시 보기

- model은 어떻게 생겼는가?
- model.fit()이 하고있는 일
- optimizer가 하고 있는 일
- Loss는 왜 줄어드는가?

  ...

- GPU 사용해보기

Colaboratory CPU/GPU 시간차이 체감

- 런타임->런타임 유형 변경->GPU선택
- Epoch=100

- import time
- s = time.time()
- print(time.time()-s)

실습 목표

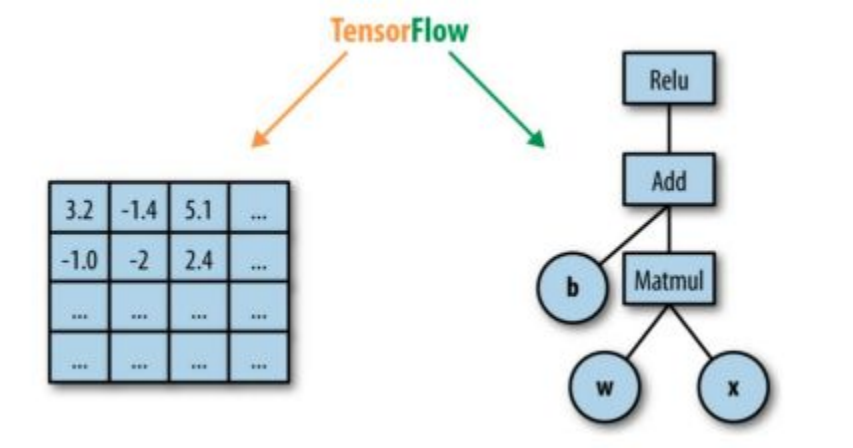
- Fashion MNIST 데이터 분류하기

텐서플로우 케라스?

- 케라스는 많은 이들이 딥러닝을 쉽게 접할 수 있도록, 다양한 플랫폼(텐서플로우, Theano 등) 위에서 딥러닝 모델을 만들 수 있는 응용 프로그램 프로그래밍 인터페이스(API)임
- 케라스는 텐서 곱(tensor products), 합성곱(convolutions)과 같은 저수준 작업을 자체적으로 수행하지 않고 백엔드(ex. Tensorflow)에 의존함
- 케라스로 모델을 쉽게 만들고, 그 내부의 복잡한 연산은 텐서플로우가 알아서 하게 됨
- 이미 알려진 모델은 쉽게 만들 수 있지만(사용자 친화성) 새 모델을 만들 때 케라스가 아닌 텐서플로우를 사용해야 할 수도 있음

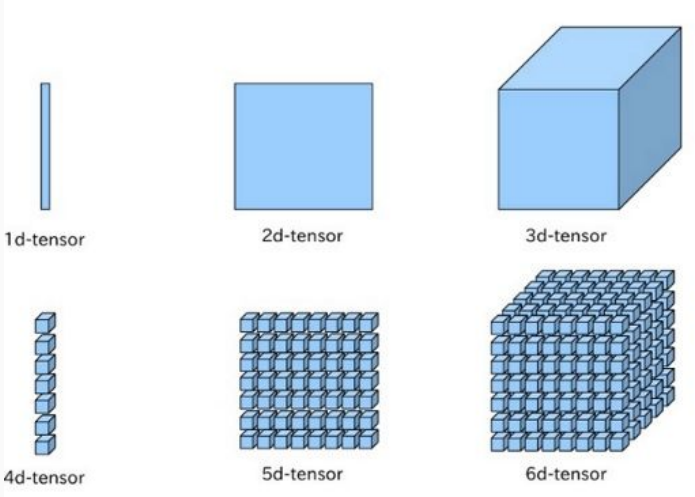
텐서플로우

- 텐서라는 데이터 구조와 오퍼레이션이라는 연산을 가짐
- 텐서와 오퍼레이션을 통해 계산 그래프를 그리고 이를 계산해줌



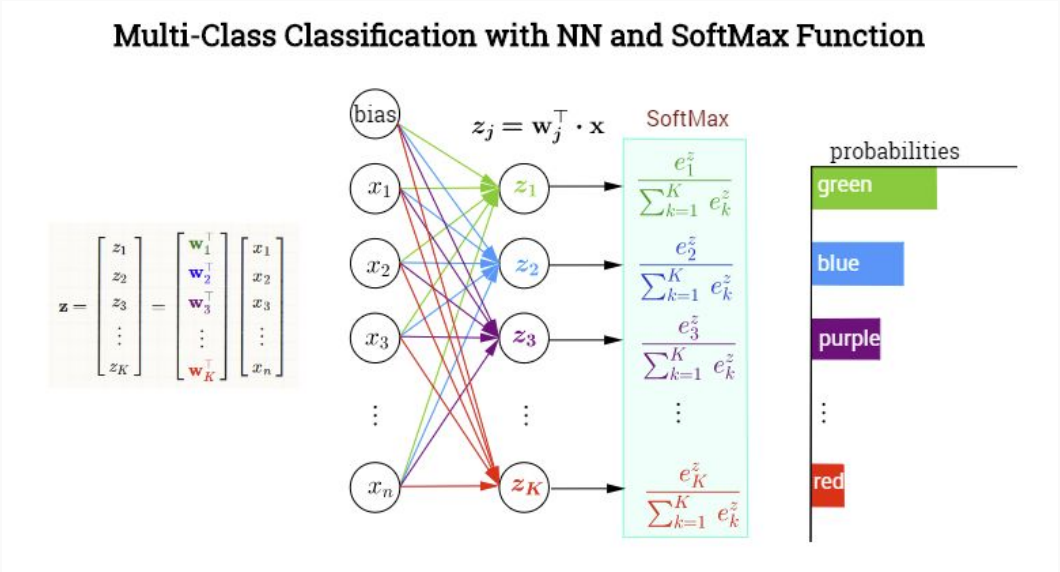
데이터 구조

- 텐서(Tensor): 값들이 들어있는 N차원의 데이터
  - 0차원: 스칼라
  - 1차원 텐서: 벡터
  - 2차원 텐서: 행렬
- Fashion MNIST 데이터셋
  - 3차원 텐서가 됨
  - 첫번째 차원은 데이터 인덱스
  - 두번째 차원은 가로 길이
  - 세번째 차원은 세로 길이



Softmax 함수

- 출력값을 확률값으로 바꿔주는 역할
  - 0~1사이 값으로 변환
- 지수함수(exponential)를 사용해 값 차이가 크면 클수록 더 큰 확률값 차이를 보임
- 즉, 우리가 만들 모델은 주어진 입력 이미지가 각 클래스일 확률을 학습함



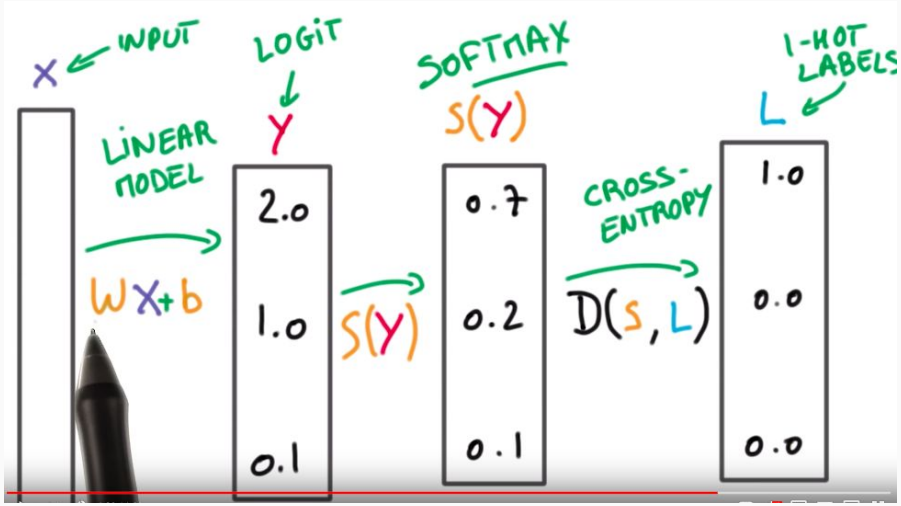
이미지 분류를 위한 손실함수

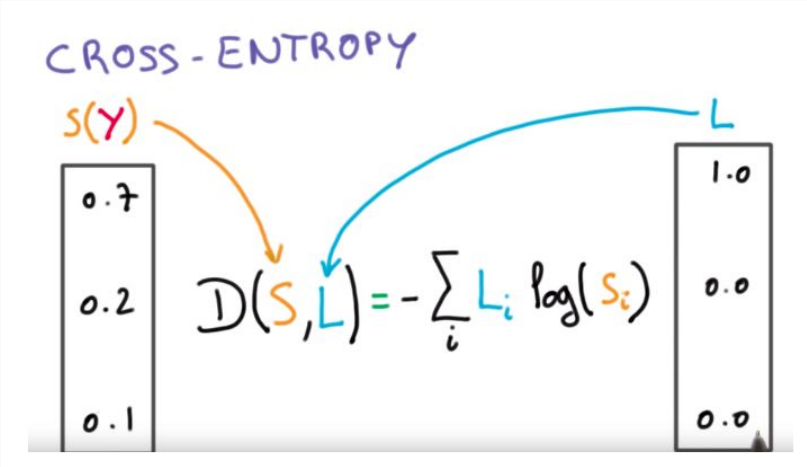
- Cross Entropy Loss
  - 두 확률분포 사이의 거리를 측정함(KL-Divergence로 나타낼 수도 있음)
  - X라는 데이터가 들어왔을 때 모델이 주는 각 클래스일 확률값 벡터와
    - 이 확률값 벡터는 Softmax함수에 의해 얻어짐
  - 원래 클래스의 one-hot encoding 벡터를 비교하여 거리를 측정함

$$J(\boldsymbol{\theta}) = -\mathbb{E}_{\mathbf{x},\mathbf{y}\sim\hat{p}_{\text{data}}} \log p_{\text{model}}(\mathbf{y}|\mathbf{x})$$

$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1-y_i) \cdot log(1-p(y_i))$$

Cross Entropy Loss





케라스의 시퀀셜 API

```
model = keras.models.Sequential()
model.add(keras.layers.Flatten(input_shape=[28, 28]))
model.add(keras.layers.Dense(300, activation="relu"))
model.add(keras.layers.Dense(100, activation="relu"))
model.add(keras.layers.Dense(10, activation="softmax"))
```

or

```
model = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[28, 28]),
    keras.layers.Dense(300, activation="relu"),
    keras.layers.Dense(100, activation="relu"),
    keras.layers.Dense(10, activation="softmax")
])
```

모델 컴파일, 훈련, 예측

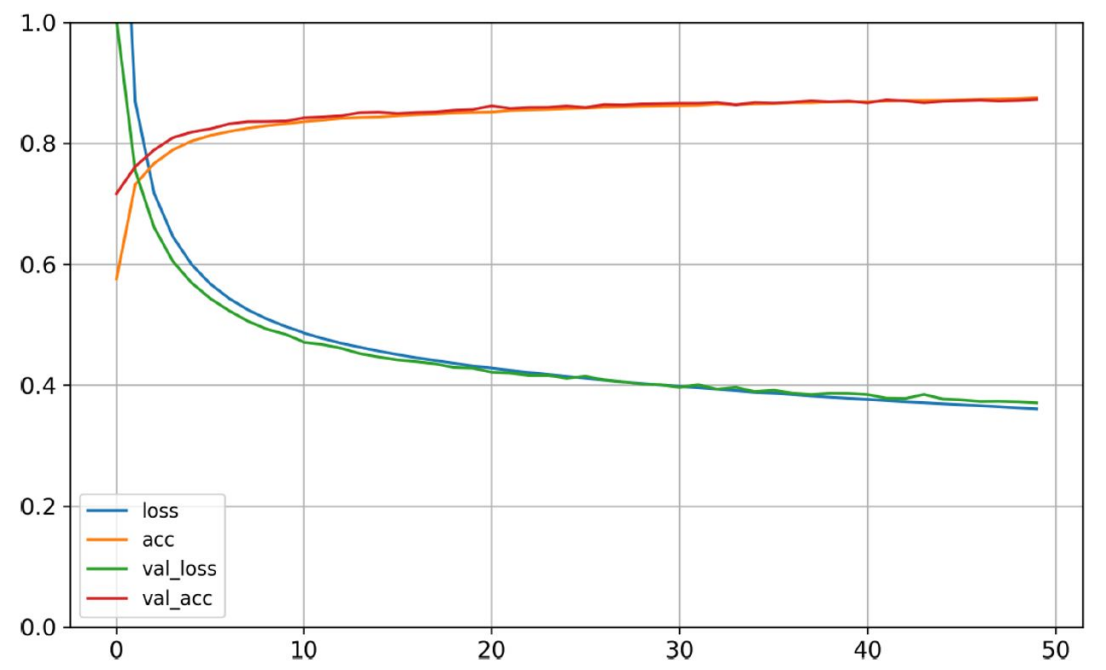
```
model.compile(loss="sparse_categorical_crossentropy",
              optimizer="sgd",
              metrics=["accuracy"])
```

```
history = model.fit(X_train, y_train, epochs=30,
                    validation_data=(X_valid, y_valid))
```

```
entropy_test = model.evaluate(X_test, y_test)
```

```
y_proba = model.predict(X_new)
y_pred = model.predict_classes(X_new)
```

학습 곡선(Learning Curves)



자가 실습

- 30분 제한
- FCN의 층 수를 바꿔보며 성능 측정하기
- FCN의 노드 수를 바꿔보며 성능 측정하기
- 활성화 함수 바꿔보기

  층이나 노드 수가 너무 많아지면 학습하는 데 걸리는 시간이 매우 길어지니 주의!

FCN의 층을 늘린다면?

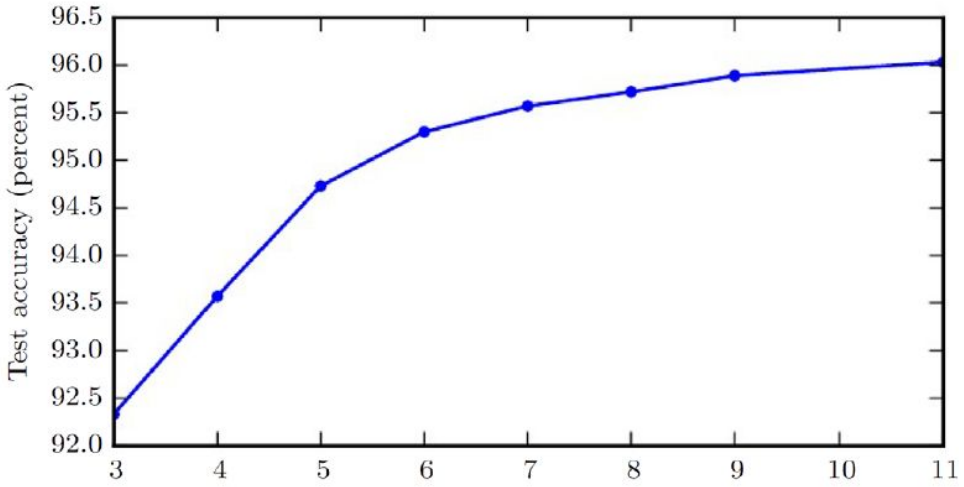
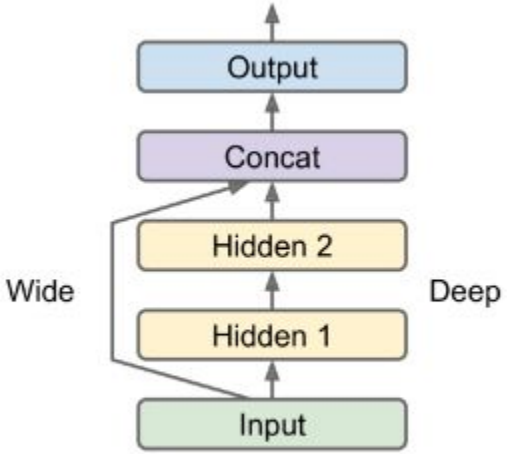


Fig. 6.6

케라스의 추가 기능(Functional API)

```
input = keras.layers.Input(shape=X_train.shape[1:])
hidden1 = keras.layers.Dense(30, activation="relu")(input)
hidden2 = keras.layers.Dense(30, activation="relu")(hidden1)
concat = keras.layers.Concatenate()([input, hidden2])
output = keras.layers.Dense(1)(concat)
model = keras.models.Model(inputs=[input], outputs=[output])
```



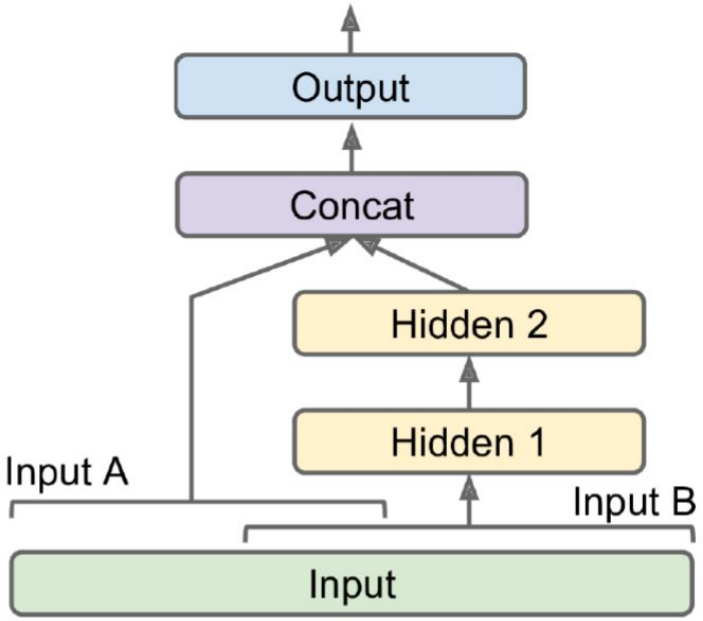
케라스의 추가 기능(Functional API)

```
input_A = keras.layers.Input(shape=[5])
input_B = keras.layers.Input(shape=[6])
hidden1 = keras.layers.Dense(30, activation="relu")(input_B)
hidden2 = keras.layers.Dense(30, activation="relu")(hidden1)
concat = keras.layers.concatenate([input_A, hidden2])
output = keras.layers.Dense(1)(concat)
model = keras.models.Model(inputs=[input_A, input_B], outputs=[output])

model.compile(loss="mse", optimizer="sgd")

X_train_A, X_train_B = X_train[:, :5], X_train[:, 2:]
X_valid_A, X_valid_B = X_valid[:, :5], X_valid[:, 2:]
X_test_A, X_test_B = X_test[:, :5], X_test[:, 2:]
X_new_A, X_new_B = X_test_A[:3], X_test_B[:3]

history = model.fit((X_train_A, X_train_B), y_train, epochs=20,
                    validation_data=((X_valid_A, X_valid_B), y_valid))
mse_test = model.evaluate((X_test_A, X_test_B), y_test)
y_pred = model.predict((X_new_A, X_new_B))
```



케라스의 추가 기능(Functional API)

```
[...] # Same as above, up to the main output layer
output = keras.layers.Dense(1)(concat)
aux_output = keras.layers.Dense(1)(hidden2)
model = keras.models.Model(inputs=[input_A, input_B],
                           outputs=[output, aux_output])

model.compile(loss=["mse", "mse"], loss_weights=[0.9, 0.1], optimizer="sgd")

history = model.fit(
    [X_train_A, X_train_B], [y_train, y_train], epochs=20,
    validation_data=([X_valid_A, X_valid_B], [y_valid, y_valid]))

total_loss, main_loss, aux_loss = model.evaluate(
    [X_test_A, X_test_B], [y_test, y_test])
```