AI 프로그래밍

- 9주차

인하 공전 컴퓨터 정보과

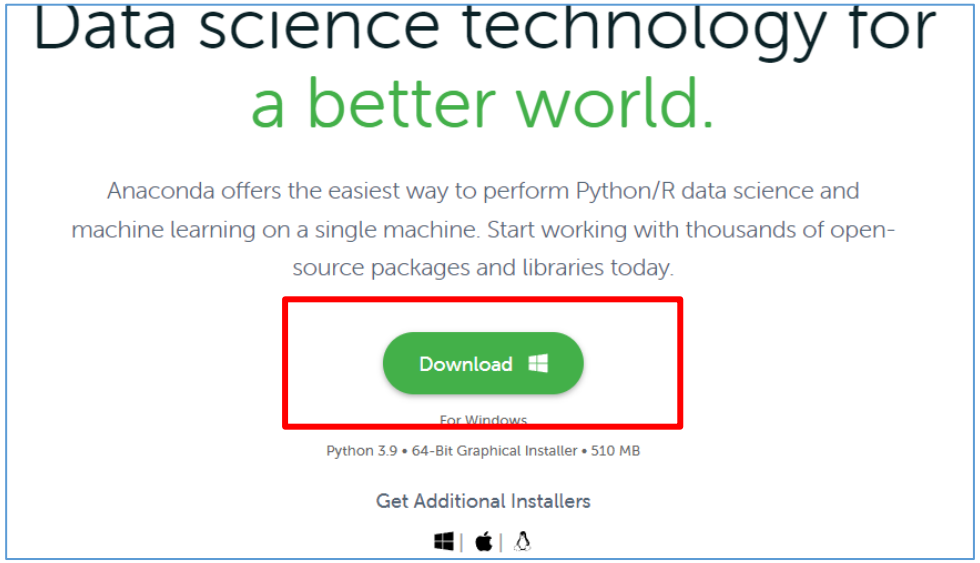
민 정혜

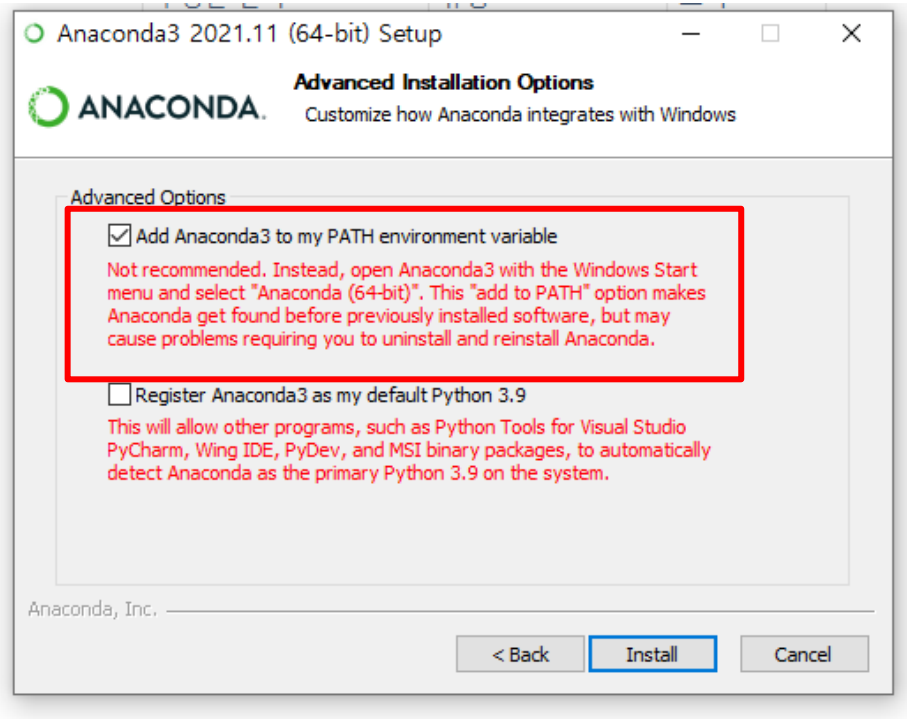
오늘 수업 순서

- Anaconda, jupyter 설치
- 케라스 딥러닝 코드 분석
- 케라스 딥러닝 파라미터 분석

아나콘다&주피터 설치 하기

https://www.anaconda.com/

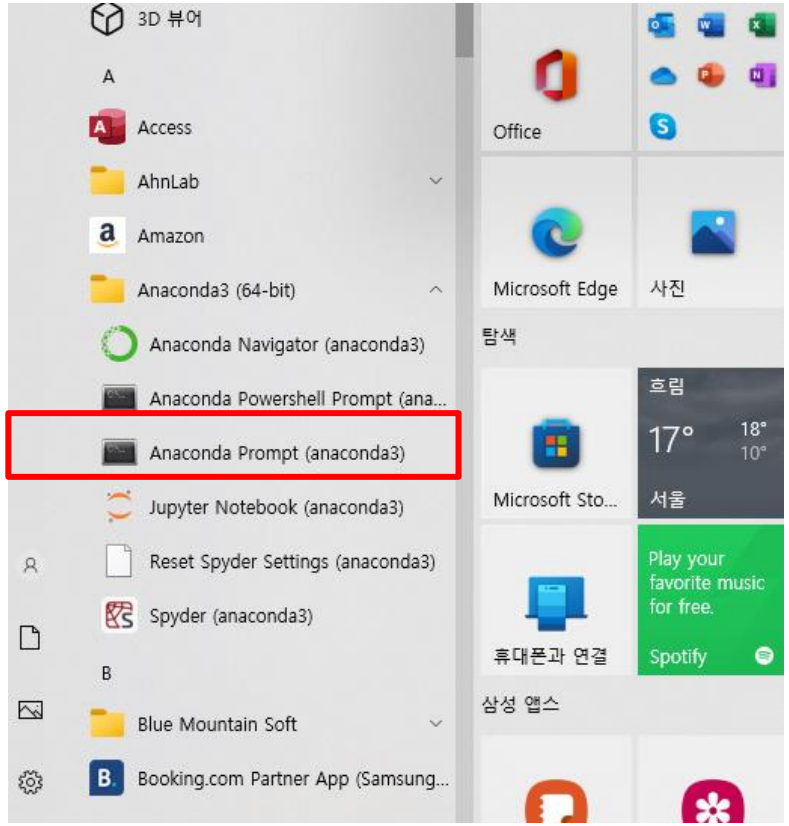




아나콘다 : 가상 환경 제공

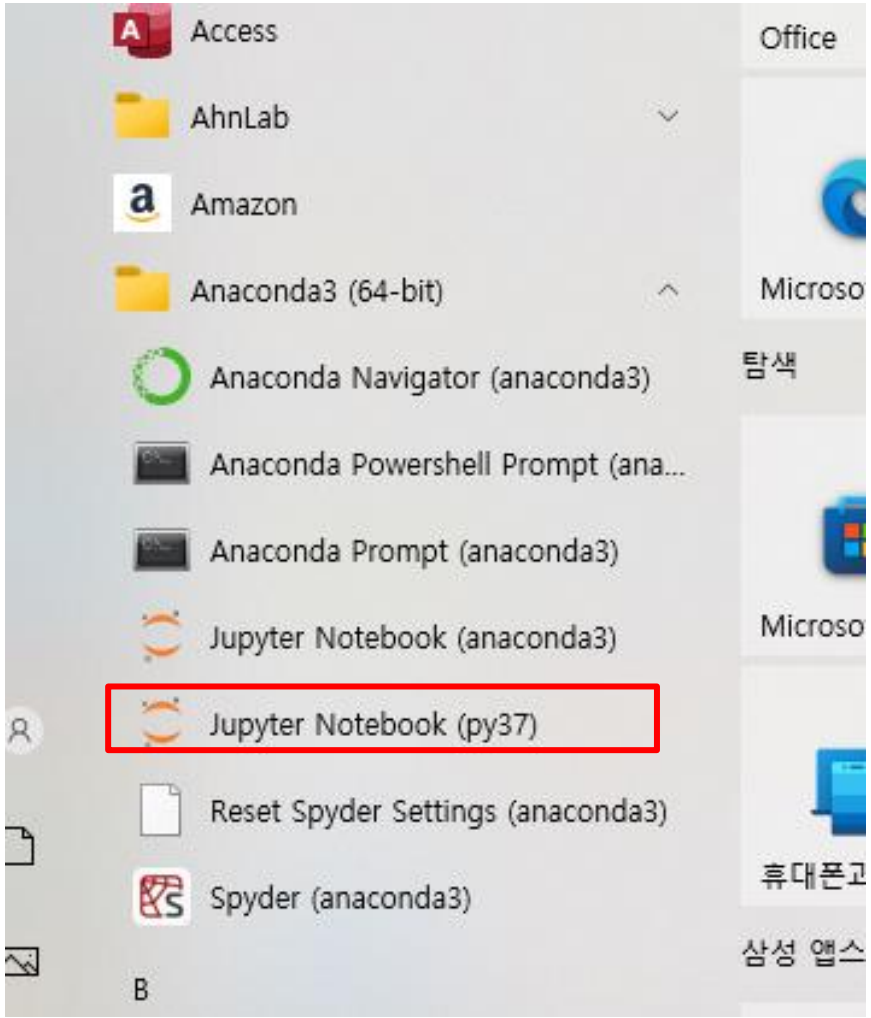
아나콘다&주피터 설치 하기

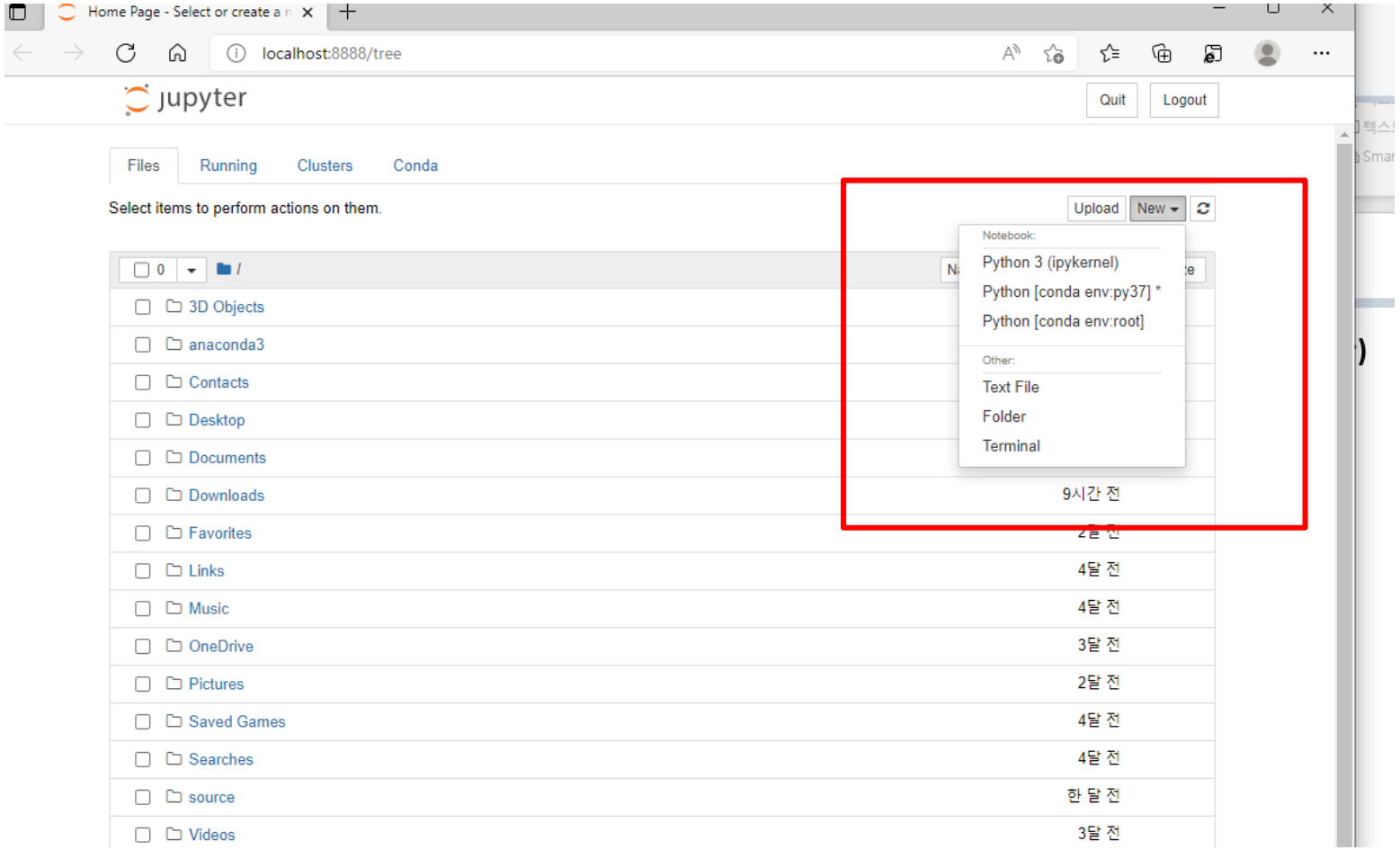
https://www.anaconda.com/




```
base>>conda create –n py37 python=3.7
base>>conda activate py37

py37>>pip install tensorflow==2.0.0
py37>>pip install keras==2.3
py37>>pip install scikit-learn
py37>>pip install matplotlib
py37>>pip install opencv-python
py37>>pip install pandas
py37>>pip install seaborn

py37>>conda install nb_conda
```

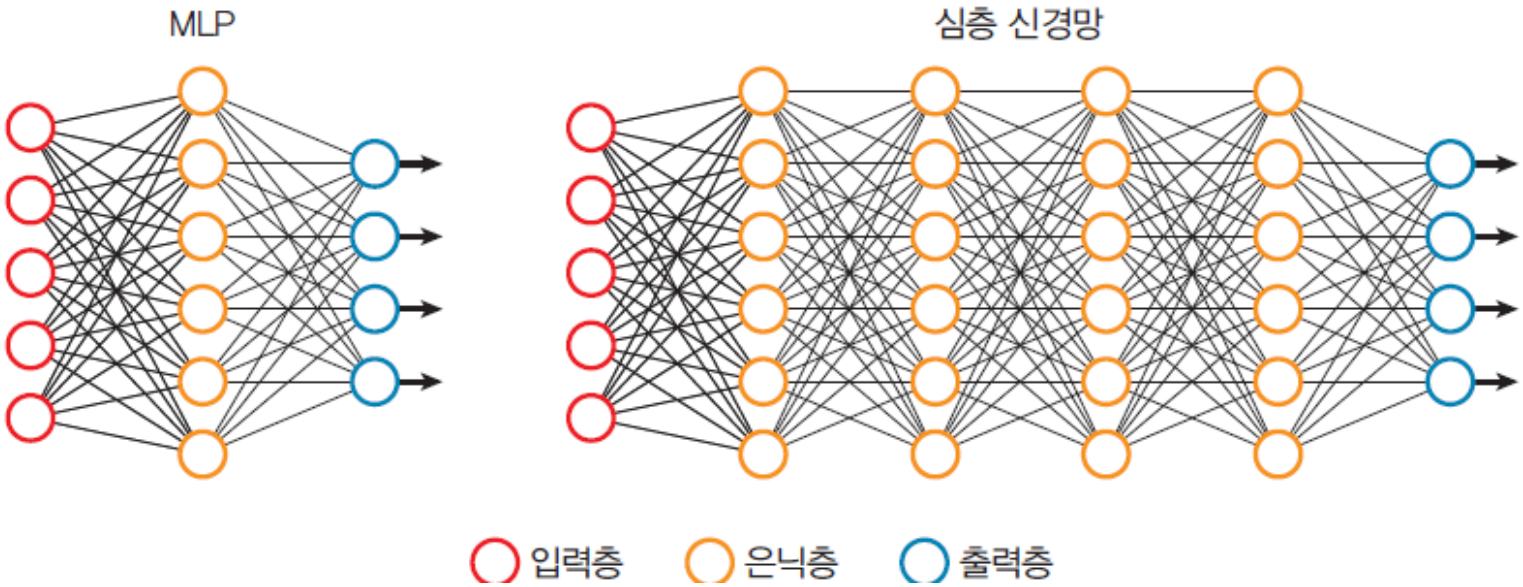
아나콘다&주피터 설치 하기





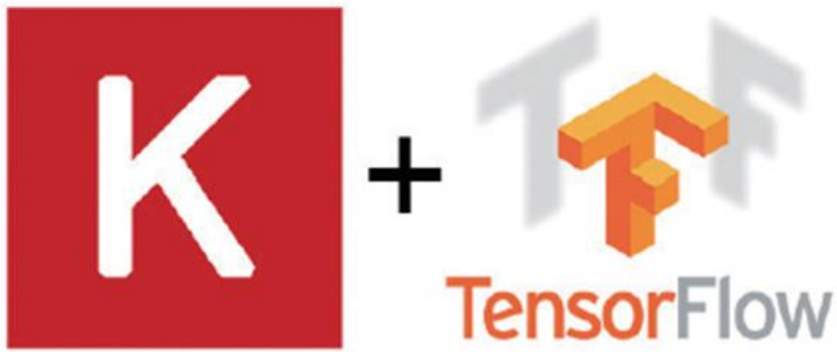
딥러닝 (Deep learning)

- **심층 신경망 (Deep Neural Network)을 사용하는 학습 방법**
- 심층 신경망(DNN: Deep Neural Networks)은 MLP(다층 퍼셉트론)에서 은닉층의 개수를 증가시킨 것이다.
- 은닉층을 하나만 사용하는 것이 아니고 여러 개를 사용한다.
- 최근에 딥러닝은 컴퓨터 시각, 음성 인식, 자연어 처리, 소셜 네트워크 필터링, 기계 번역 등에 적용되어서 인간 전문가에 필적하는 결과를 얻고 있다.



딥러닝 (Deep learning) 실습

- 케라스로 모델 설계 하기 (keras 라이브러리 이용)
- Tensorflow + keras 사용
- Import tensorflow.keras



케라스로 신경망을 작성 하는 절차

- 케라스의 핵심 데이터 구조는 모델(model)이며 이것은 레이어를 구성하는 방법을 나타낸다.
- 가장 간단한 모델 유형은 Sequential 선형 스택 모델이다. Sequential 모델은 레이어를 선형으로 쌓을 수 있는 신경망 모델이다

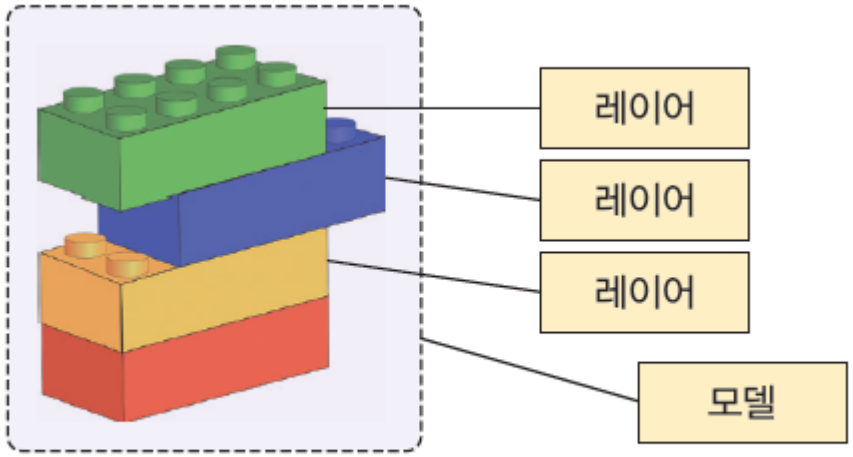
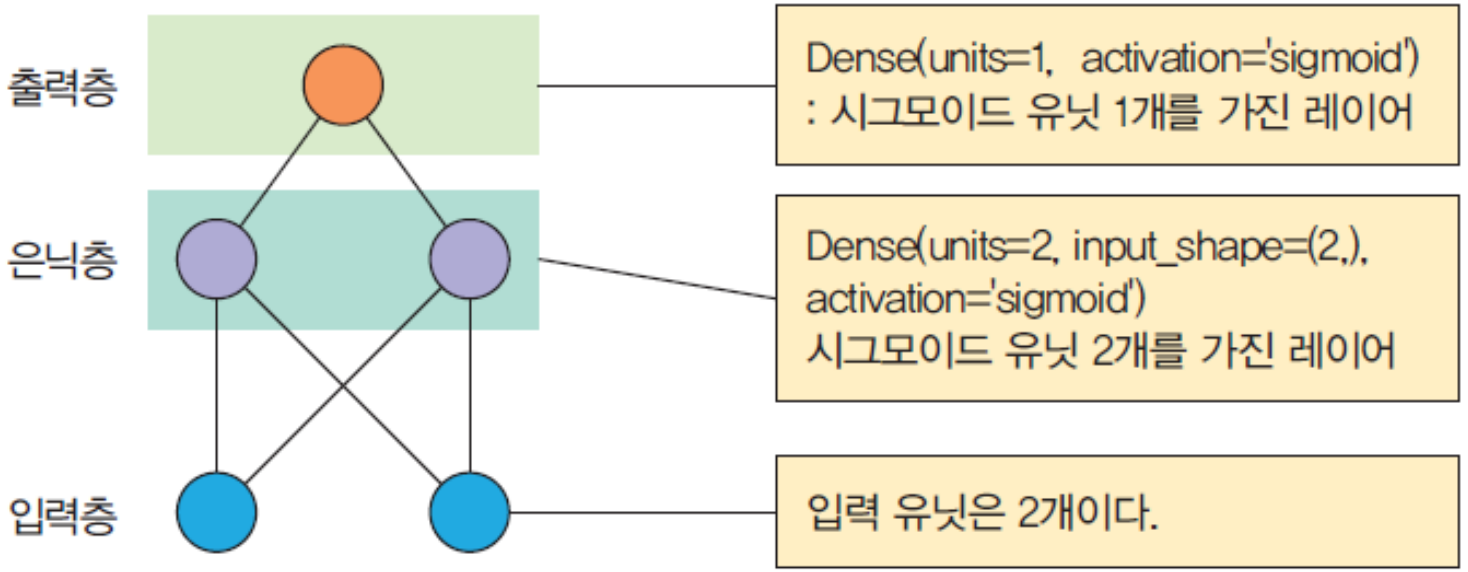


그림 7-5 케라스의 기본 개념

케라스로 신경망을 작성 하는 절차



케라스로 신경망을 작성 하는 절차


```
model = tf.keras.models.Sequential()


model.add(tf.keras.layers.Dense(units=2, input_shape=(2,), activation='sigmoid')) #①
model.add(tf.keras.layers.Dense(units=1, activation='sigmoid')


model.compile(loss='mean_squared_error', optimizer=keras.optimizers.SGD(lr=0.3)
              , metrics='accuracy')


model.fit(X, y, batch_size=1, epochs=10000)


print( model.predict(X) )
```

Sequential 모델을 생성

Add함수를 이용하여 layer 추가

컴파일

학습을 수행

모델 테스트

[Keras API reference](https://keras.io/api/)

https://keras.io/api/

OR 연산을 수행하는 신경망 실습


```
import numpy as np
import tensorflow as tf

model = tf.keras.models.Sequential()

model.add(tf.keras.layers.Dense(units=2, input_shape=(2,), activation='sigmoid'))
model.add(tf.keras.layers.Dense(units=1,  activation='sigmoid'))
model.compile(loss='mean_squared_error', optimizer=tf.keras.optimizers.SGD(lr=0.3),metrics='accuracy')

model.summary()

X = np.array([[0, 0],[0, 1],[1, 0],[1, 1]])
y = np.array([[0], [1], [1], [1]])

model.fit(X, y, batch_size=1, epochs=500)

print( model.predict(X) )
```

케라스 신경망 파라미터

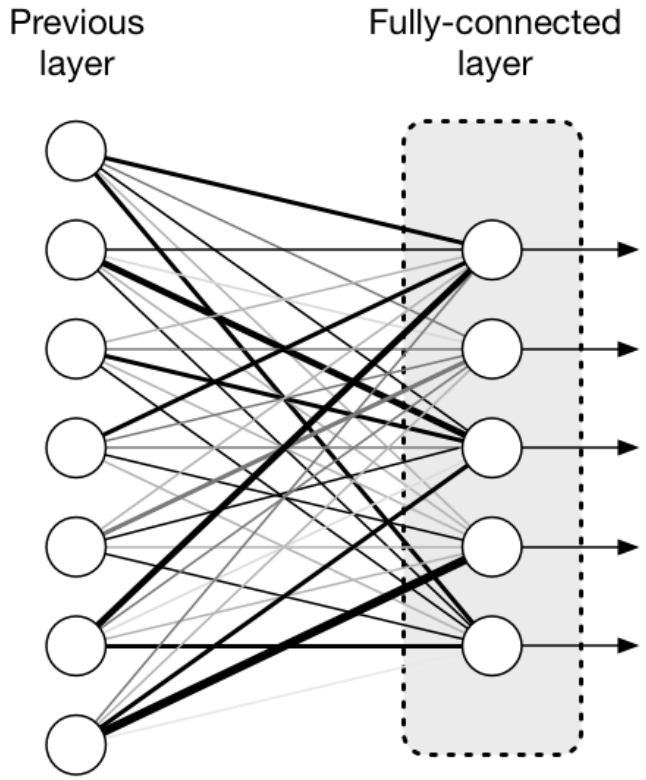

```
model = tf.keras.models.Sequential()


model.add(tf.keras.layers.Dense(units=2, input_shape=(2,), activation='sigmoid')) #①
model.add(tf.keras.layers.Dense(units=1, activation='sigmoid')



model.compile(loss='mean_squared_error', optimizer=keras.optimizers.SGD(lr=0.3),
, metrics='accuracy'))


model.fit(X, y, batch_size=1, epochs=10000)


print( model.predict(X) )
```

케라스 신경망 파라미터

tf.keras.layers.**Dense**

완전 연결 계층 : (Fully Connected Layer, Densely connected layer)

한 층 (layer)의 모든 뉴런이 그 다음 층의 모든 뉴런과 연결된 상태



https://keras.io/api/layers/core_layers/

케라스 신경망 파라미터 –loss (손실함수)

평균 제곱 계열	mean_squared_error	평균 제곱 오차 계산: mean(square(yt - yo))
	mean_absolute_error	평균 절대 오차(실제 값과 예측 값 차이의 절댓값 평균) 계산: mean(abs(yt - yo))
	mean_absolute_percentage_error	평균 절대 백분율 오차(절댓값 오차를 절댓값으로 나눈 후 평균) 계산: mean(abs(yt - yo)/abs(yt)) (단, 분모 ≠ 0)
	mean_squared_logarithmic_error	평균 제곱 로그 오차(실제 값과 예측 값에 로그를 적용한 값의 차이를 제곱한 값의 평균) 계산: mean(square((log(yo) + 1) - (log(yt) + 1)))
교차 엔트로피 계열	categorical_crossentropy	범주형 교차 엔트로피(일반적인 분류)
	binary_crossentropy	이항 교차 엔트로피(두 개의 클래스 중에서 예측할 때)

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2$$

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|x_i - x|$$

케라스 신경망 파라미터 –loss (손실함수)

교차 엔트로피 : 2개의 확률 분포 간의 거리.

- 교차 엔트로피는 2개의 확률 분포 p, q에 대해서 다음과 같이 정의된다.

$$H(p, q) = -\sum_x p(x) \log_n q(x)$$

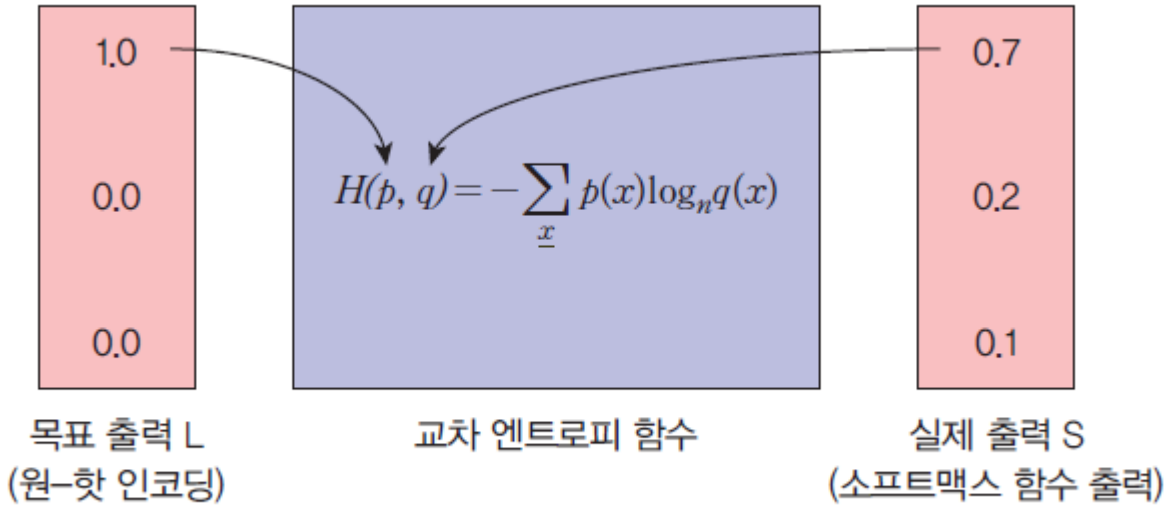


그림 8–9 교차 엔트로피 함수

케라스 신경망 파라미터 –loss (손실함수)

교차 엔트로피

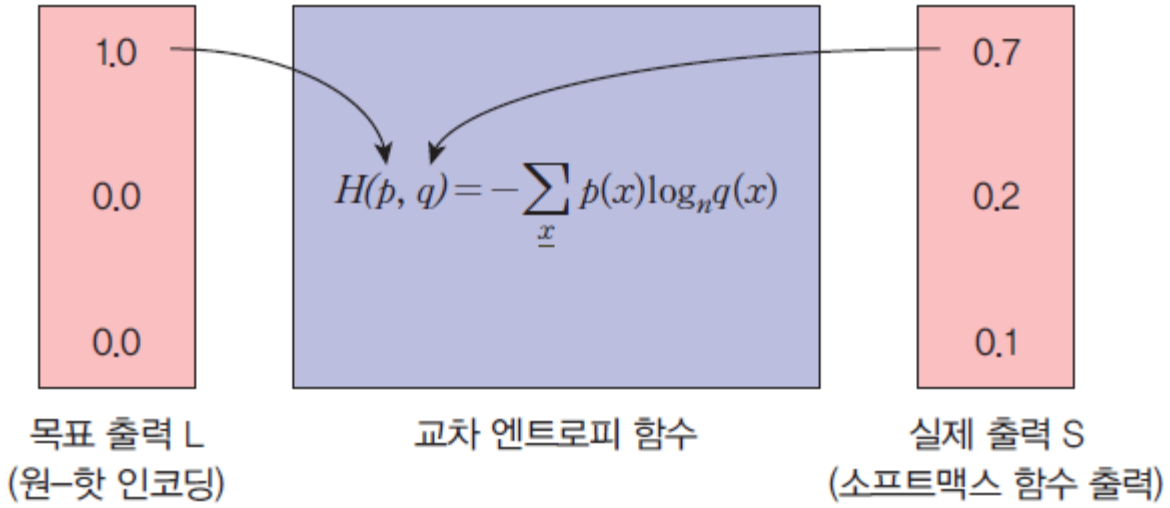


그림 8–9 교차 엔트로피 함수

$$
\begin{aligned}
H(p, q) &= -\sum_x p(x)\log_n q(x) \\
&= -(1.0 * \log 0.7 + 0.0 * \log 0.2 + 0.0 * \log 0.1) \\
&= 0.154901
\end{aligned}
$$

케라스 신경망 파라미터 –loss (손실함수)

교차 엔트로피

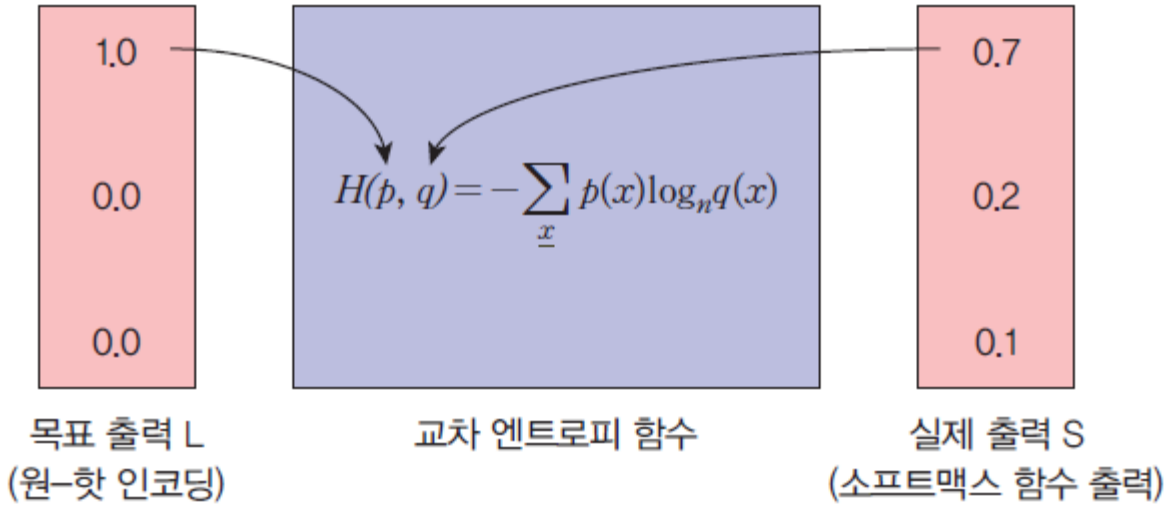
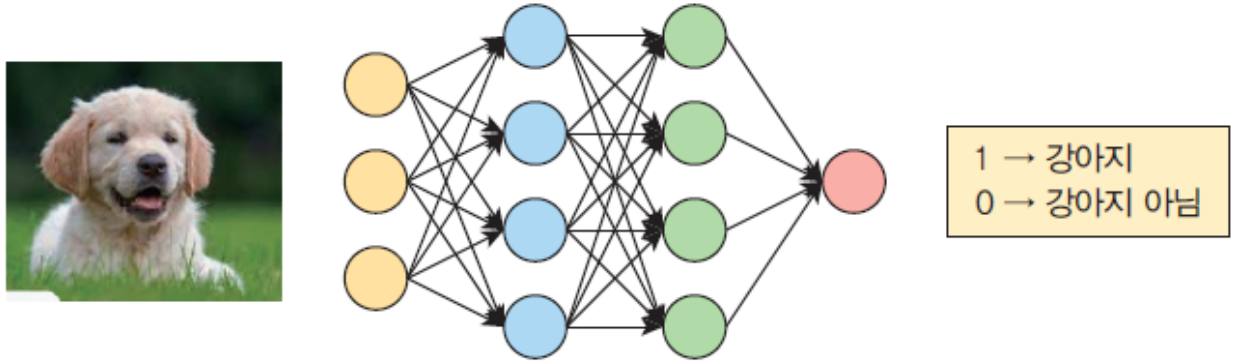


그림 8–9 교차 엔트로피 함수

$$
\begin{aligned}
H(p, q) &= -\sum_x p(x)\log_n q(x) \\
&= -(1.0 * \log 1.0 + 0.0 * \log 0.0 + 0.0 * \log 0.0) \\
&= 0
\end{aligned}
$$

케라스 신경망 파라미터 –loss (손실함수)

- **BinaryCrossentropy:** 이진 교차 엔트로피(BCE)는 이진 분류 문제를 해결하는 데 사용되는 손실 함수이다.
- 즉 우리가 분류해야 하는 부류가 두 가지뿐일 때 사용한다. 예를 들어 이미지를 "강아지", "강아지 아님"의 두 부류로 분류할 때 BinaryCrossentropy를 사용한다.



$$BCE = -\frac{1}{n}\sum_{i=1}^{n}(y_i \log_n(\hat{y}_i) + (1-y_i)\log_n(1-\hat{y}_i))$$

케라스 신경망 파라미터 –loss (손실함수)

- BinaryCrossentropy

실제 레이블 y	1	0	0	1
예측 값 ŷ	0.8	0.3	0.5	0.9

$$\text{샘플 1}: \ BCE1 = -(1 \cdot \log(0.8) + (1-1) \cdot \log(1-0.8))$$
$$\text{샘플 2}: \ BCE2 = -(0 \cdot \log(0.3) + (1-0) \cdot \log(1-0.3))$$
$$\text{샘플 3}: \ BCE3 = -(0 \cdot \log(0.5) + (1-0) \cdot \log(1-0.5))$$
$$\text{샘플 4}: \ BCE4 = -(1 \cdot \log(0.9) + (1-1) \cdot \log(1-0.9))$$

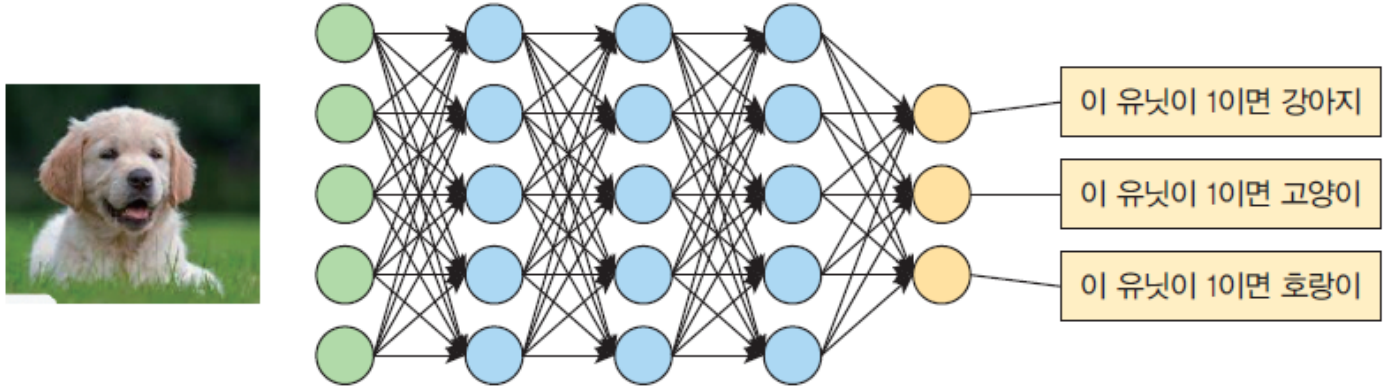
$$BCE = \frac{BCE1 + BCE2 + BCE3 + BCE4}{4} \approx 0.345$$

```
y_true = [  [1],   [0],   [0],   [1]]
y_pred = [[0.8], [0.3], [0.5], [0.9]]
bce = tf.keras.losses.BinaryCrossentropy()
print(bce(y_true, y_pred).numpy())
```

```
0.3445814
```

케라스 신경망 파라미터 –loss (손실함수)

- **CategoricalCrossentropy**: 우리가 분류해야 할 부류가 두 개 이상이라면(즉 **다중 분류 문제**라면) 을 사용
- 정답 레이블은 원-핫 인코딩으로 제공한다. 예를 들어서 입력 이미지를 "강아지(1,0, 0)", "고양이(0, 1, 0)", "호랑이(0, 0, 1)" 중의 하나로 분류




```
y_true = [[0.0, 1.0, 0.0], [0.0, 0.0, 1.0], [1.0, 0.0, 0.0]] # 고양이, 호랑이, 강아지
y_pred = [[0.6, 0.3, 0.1], [0.3, 0.6, 0.1], [0.1, 0.7, 0.2]]
cce = tf.keras.losses.CategoricalCrossentropy ()
print(cce(y_true, y_pred).numpy ())
```

```
1.936381
```

케라스 신경망 파라미터 –optimizer

고급 경사 하강법	개요	효과	케라스 사용법
확률적 경사 하강법 (SGD)	랜덤하게 추출한 일부 데이터를 사용해 더 빨리, 자주 업데이트를 하게 하는 것	속도 개선	keras.optimizers.SGD(lr = 0.1) 케라스 최적화 함수를 이용합니다.
모멘텀 (Momentum)	관성의 방향을 고려해 진동과 폭을 줄이는 효과	정확도 개선	keras.optimizers.SGD(lr = 0.1, momentum = 0.9) 모멘텀 계수를 추가합니다.
네스테로프 모멘텀 (NAG)	모멘텀이 이동시킬 방향으로 미리 이동해서 그레이디언트를 계산. 불필요한 이동을 줄이는 효과	정확도 개선	keras.optimizers.SGD(lr = 0.1, momentum = 0.9, nesterov = True) 네스테로프 옵션을 추가합니다.
아다그라드 (Adagrad)	변수의 업데이트가 잦으면 학습률을 적게 하여 이동 보폭을 조절하는 방법	보폭 크기 개선	keras.optimizers.Adagrad(lr = 0.01, epsilon = 1e - 6) 아다그라드 함수를 사용합니다. ※ 참고: 여기서 epsilon, rho, decay 같은 파라미터는 바꾸지 않고 그대로 사용하기를 권장하고 있습니다. 따라서 lr, 즉 learning rate(학습률) 값만 적절히 조절하면 됩니다.

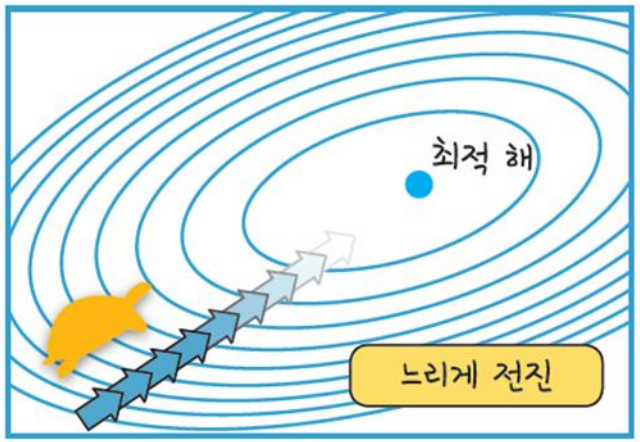
케라스 신경망 파라미터 –optimizer

고급 경사 하강법	개요	효과	케라스 사용법
알엠에스프롭 (RMSProp)	아다그라드의 보폭 민감도를 보완한 방법	보폭 크기 개선	keras.optimizers.RMSprop(lr = 0.001, rho = 0.9, epsilon = 1e - 08, decay = 0.0) 알엠에스프롭 함수를 사용합니다.
아담(Adam)	모멘텀과 알엠에스프롭 방법을 합친 방법	정확도와 보폭 크기 개선	keras.optimizers.Adam(lr = 0.001, beta_1 = 0.9, beta_2 = 0.999, epsilon = 1e - 08, decay = 0.0) 아담 함수를 사용합니다.

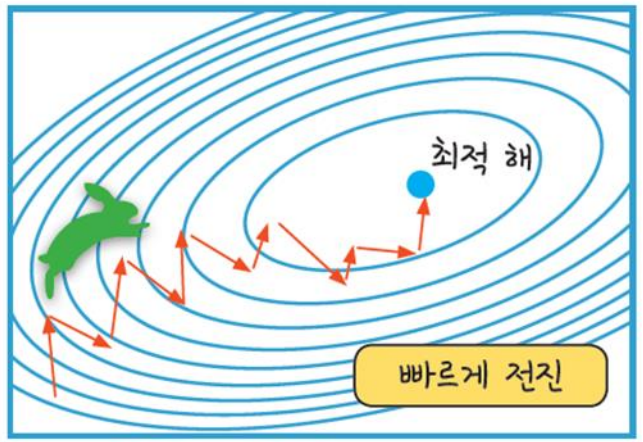
고급 경사 하강법

경사 하강법 (**GD** : Gradient Descent) : 정확 하지만 한번 업데이트 할 때마다 전체 데이터를 미분 해야 함. 속도 문제

➡ 확률적 경사 하강법 (**SGD** : Stochastic Gradient Descent)
전체 데이터를 사용하지 않고 랜덤하게 추출한 일부 데이터를 사용
일부 데이터를 사용하기 때문에 진폭이 크고 불안정 할수 있음.
속도 개선 효과



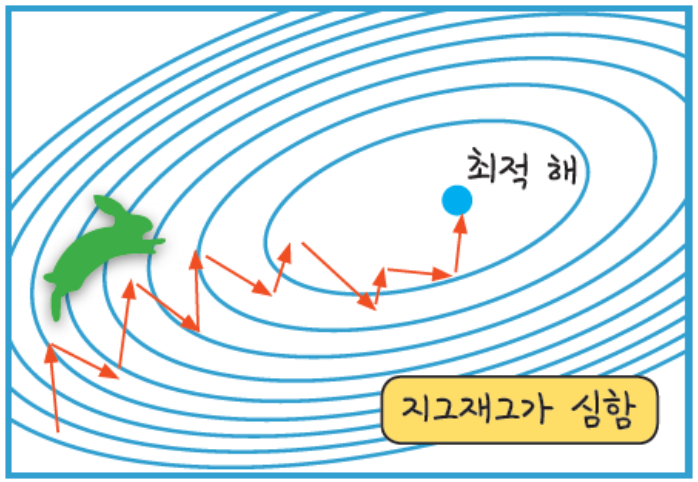
경사 하강법



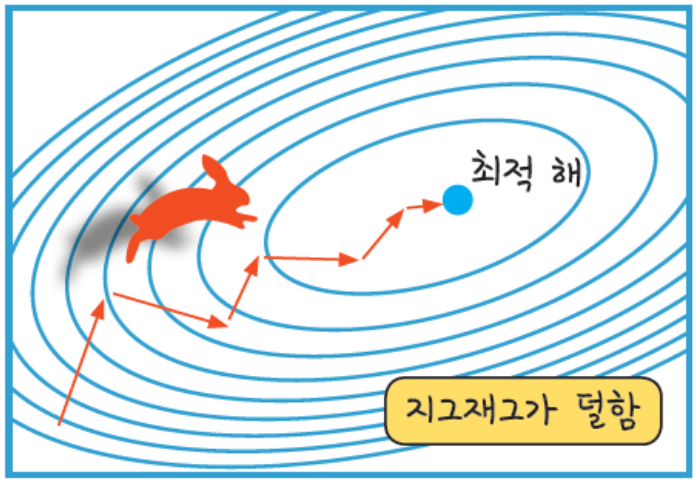
확률적 경사 하강법

고급 경사 하강법

모멘텀 (**MOMENTUM**) : 관성의 방향을 고려해 속도를 개선
안정성 정확도 향상.



확률적 경사 하강법

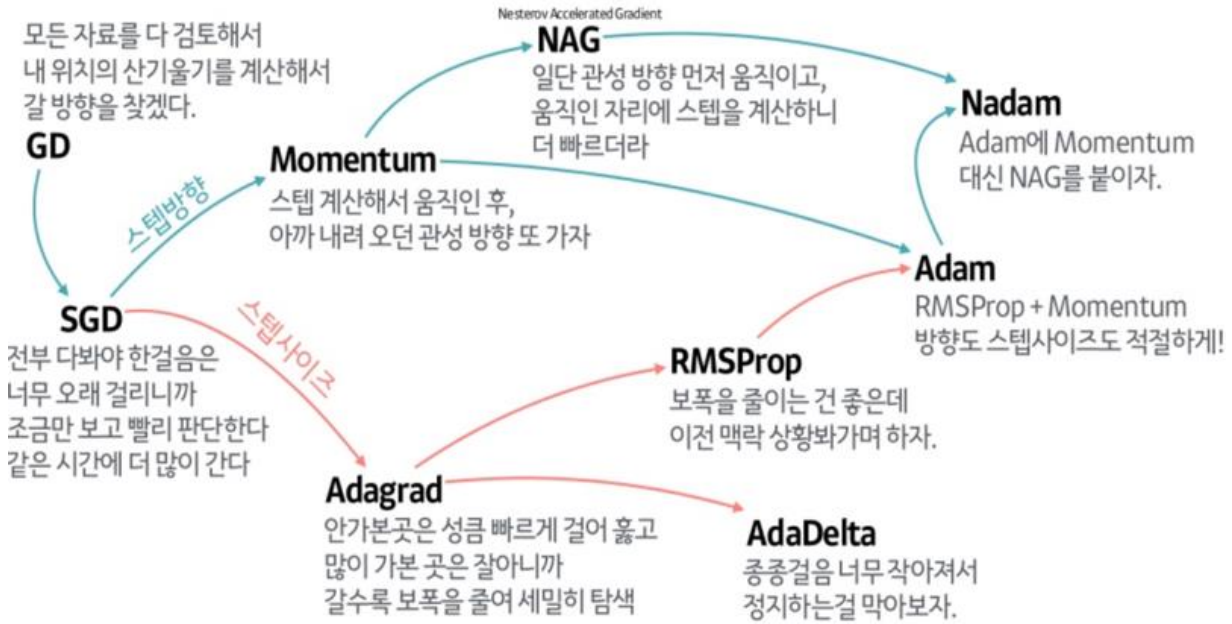


모멘텀을 적용한 확률적 경사 하강법

고급 경사 하강법

- 네스테로프 모멘텀 (**NAG**) : 모멘텀 개선. 불필요한 계산을 줄임.
- 아다그라드 (**Adagrad**) : 가변 학습률 적용. 변수 업데이트가 너무 크면 학습률을 줄여줌.
- 알엠에스프롭 (**RMSprop**): Adagrad 개선. 학습률이 너무 작아지는것을 방지
- 아담 (**Adam**: Adaptive Moment Estimation)
  : RMSprop+ Momentum
  : **현재 가장 인기 있는 최적화 방법**.

주로 Adam을 사용



경사 하강법 비교 1

https://cs231n.github.io/assets/nn3/opt2.gif

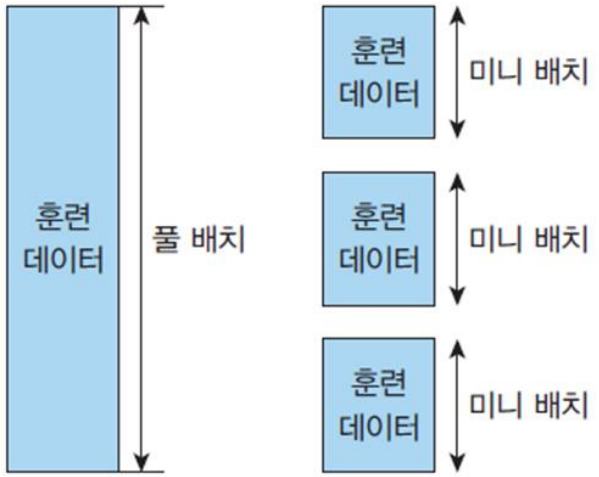
경사 하강법 비교 2

https://cs231n.github.io/assets/nn3/opt1.gif

- **자료실 colab_01_My_First_Deeplearning_basic.ipynb 실습**
- **데이터 ThoracicSurgery**

케라스 신경망 파라미터 –batch_size

- batch_size
  - 샘플을 한 번에 몇 개씩 처리할지를 정하는 부분으로 batch_size=10은 전체 샘플을 10개씩 끊어서 집어넣으라는 뜻
  - batch_size가 너무 크면 학습 속도가 느려지고, 너무 작으면 각 실행 값의 편차가 생겨서 전체 결과값이 불안정해질 수 있음
  - 변수 (w,b)가 업데이트 되는 단위.
  - 자신의 컴퓨터 메모리가 감당할 만큼의 batch_size를 찾아 설정해 주는 것이 좋음



1 epoch는 전체 샘플이 처리되는 기준

전체 샘플이 이 300 개이고 batch size가 30이면 1 epoch동안 10번 가중치가 update됨.
전체 샘플이 이 300 개이고 batch size가 10이면 1 epoch동안 30번 가중치가 update됨.

케라스 신경망 파라미터 –metrics

모델의 성능을 평가 하는 항목

accuracy :

- **긍정으로 올바르게 예측하면 TP(True Positive)라고 한다.**
- **긍정을 부정으로 잘못 예측하면 FN(False Nagative)라고 한다.**
- **부정을 긍정으로 잘못 예측하면 FP(False Positive)라고 한다.**
- **부정을 부정으로 올바르게 예측하면 TN(True Nagative)라고 한다. 긍정을**

혼동 행렬

$$정확도(accuarcy) = \frac{올바르게\ 분류한\ 샘플\ 수}{전체\ 샘플\ 수}$$

올바르게 분류한 샘플 수=TP + TN

Precision =TP / (TP + FP) True로 예측 한 것 중에서 실제로 True 인 비율

recall = TP /(TP + FN) : 실제로 True인 것 중에서 True로 예측된 비율

딥러닝 (Deep learning) 실습

- 자료실 colab_01_My_First_Deeplearning_no_drive.ipynb 실습
- 데이터 ThoracicSurgery

폐암 환자 생존율 실습

속성 / 클래스

	정보 1	정보 2	정보 3	…	정보 17	생존 여부
1번째 환자	293	1	3.8	…	62	0
2번째 환자	1	2	2.88	…	60	0
3번째 환자	8	3	3.19	…	66	1
…	…	…	…	…	…	…
470번째 환자	447	8	5.2	…	49	0

샘플

표 10-2 폐암 환자 생존율 예측 데이터의 샘플, 속성, 클래스 구분

17개의 속성이 있을때 생존 여부를 예측 하는 모델 생성

폐암 환자 생존율 실습

```
# 딥러닝을 구동하는 데 필요한 케라스 함수 호출
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# 필요한 라이브러리 불러옴
import numpy as np
import tensorflow as tf

# 실행할 때마다 같은 결과를 출력하기 위해 설정하는 부분
np.random.seed(3)
tf.random.set_seed(3)
```

폐암 환자 생존율 실습

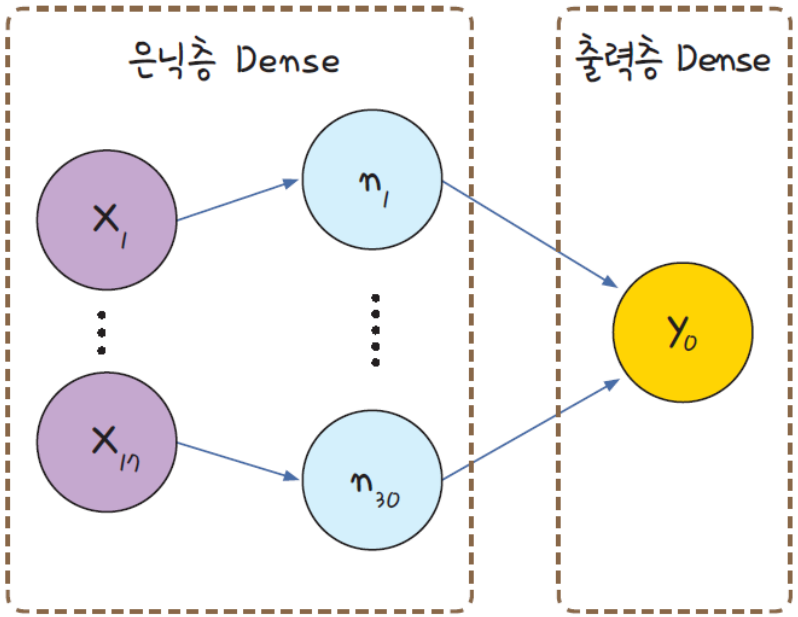
```
# 준비된 수술 환자 데이터를 불러옴
Data_set = np.loadtxt("../dataset/ThoracicSurgery.csv",
delimiter=",")

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

# 딥러닝 구조를 결정(모델을 설정하고 실행하는 부분)
model = Sequential()
model.add(Dense(30, input_dim=17, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

*MLP 정의

폐암 환자 생존율 실습

```
# 딥러닝 실행
model.compile(loss=' binary_crossentropy  ', optimizer='adam',
metrics=['accuracy'])
model.fit(X, Y, epochs=100, batch_size=10)
```

폐암 환자 생존율 실습




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

폐암 환자 생존율 실습

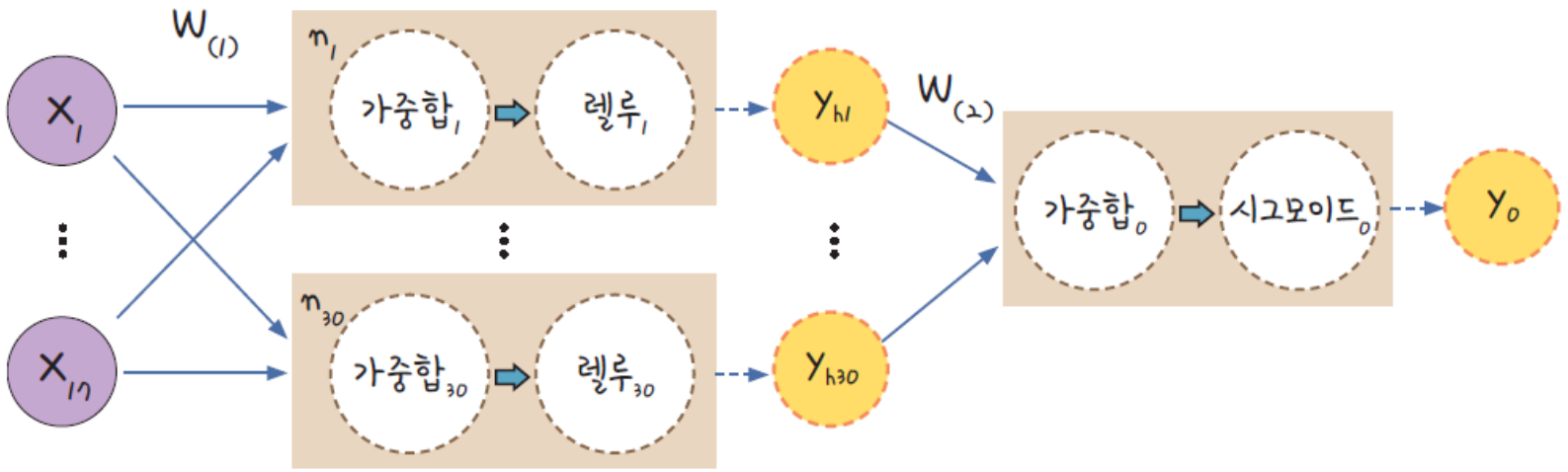


그림 10-2 폐암 환자 생존율 예측 신경망 모델의 도식화. 여기서 W는 각 층별 가중치(w)들의 집합을 말함.

과제

- **폐암 환자 생존율 분석**
  - 고급 경사 하강 법 성능 비교
  - Batch size 성능 비교
  - 다음 주 수업 전날 까지 제출

수고 하셨습니다

jhmin@inhatc.ac.kr