Chapter06

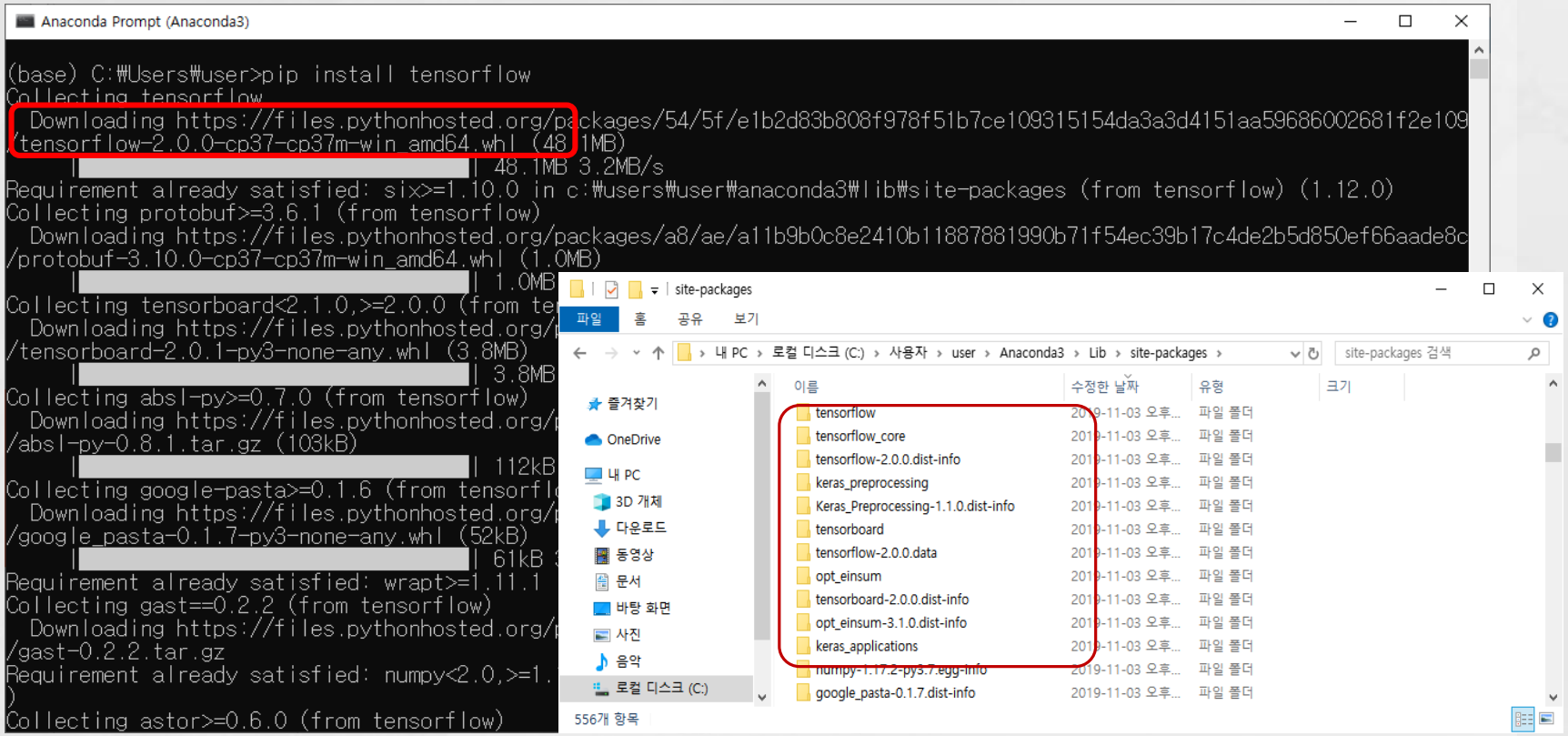
Keras model

작성자 : 김진성

목차

1. Keras 개요 & 설치

- ✓ Hyper parameter 설정이 잘 되어 있어서 튜닝 없이 사용 가능
- ✓ DNN model 생성을 위한 고수준 API
- ✓ Tensorflow2.0을 설치하면 Keras 라이브러리는 자동으로 설치

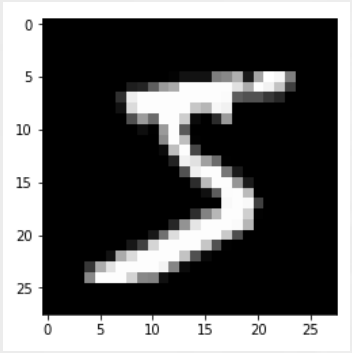


2. Keras Dataset

1) mnist

DNN, CNN 분석용 data set

train image : 60,000개(28x28), label(0~9),
Grayscale 이미지
test image : 10,000개




```
# DNN 분석용 data set
x_train = x_train.reshape(x_train.shape[0], 28*28)
x_test = x_test.reshape(x_test.shape[0], 28*28)
print(x_train.shape) # (60000, 784)
print(x_test.shape) # (10000, 784)
```

2) Cifar10

train image : 50,000개(32x32), label(0~9), RGB 컬러 이미지
test image : 10,000개

label : airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck.

3) IMDB Movie Review Sentiment Analysis

영화 사이트 IMDB의 review data, Text mining의 감성 분석용 데이터 셋
x_data : 25,000개 영화 review 텍스트 -> 단어 출현빈도 index
y_data : review에 대한 긍정(1), 부정(0) label

텍스트 벡터화

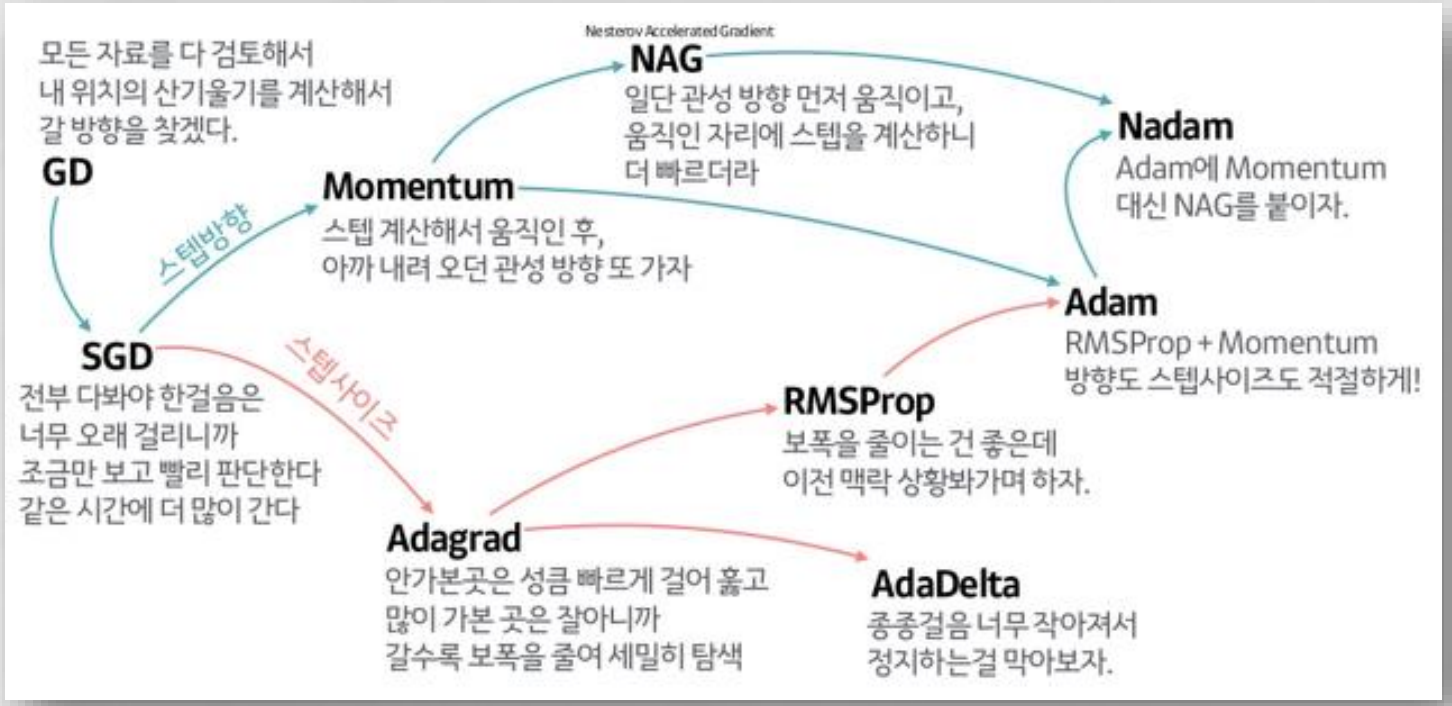
첫번째 영화 review data

x_train[0] :
[1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 458, 4468, 66, 3941, 4, 173, 36, 256, 5, 25, 100, 43, 838, 112, 50, 670, 2, 9, 35, 480, 284, 5, 150, 4, 172, 112, 167, 2, 336, 385, 39, 4, 172, 4536, 1111, 17, 546, 38, 13, 447, 4, 192, 50, 16, 6, 147, 2025, 19, 14, 22, 4, 1920, 4613, 469, 4, 22, 71, 87, 12, 16, 43, 530, 38, 76, 15, 13, 1247, 4, 22, 17, 515, 17, 12, 16, 626, 18, 2, 5, 62, 386, 12, 8, 316, 8, 106, 5, 4, 2223, 5244, 16, 480, 66, 3785, 33, 4, 130, 12, 16, 38, 619, 5, 25, 124, 51, 36, 135, 48, 25, 1415, 33, 6, 22, 12, 215, 28, 77, 52, 5, 14, 407, 16, 82, 2, 8, 4, 107, 117, 5952, 15, 256, 4, 2, 7, 3766, 5, 723, 36, 71, 43, 530, 476, 26, 400, 317, 46, 7, 4, 2, 1029, 13, 104, 88, 4, 381, 15, 297, 98, 32, 2071, 56, 26, 141, 6, 194, 7486, 18, 4, 226, 22, 21, 134, 476, 26, 480, 5, 144, 30, 5535, 18, 51, 36, 28, 224, 92, 25, 104, 4, 226, 65, 16, 38, 1334, 88, 12, 16, 283, 5, 16, 4472, 113, 103, 32, 15, 16, 5345, 19, 178, 32]
y_train[0] : 1

3. Keras Optimizers

Optimizer : model 최적화 도구, 모델을 컴파일 하는 데 필요한 두 개의 인수 중 하나

model.compile(loss='mean_squared_error', optimizer='sgd')



참고 https://seamless.tistory.com/38

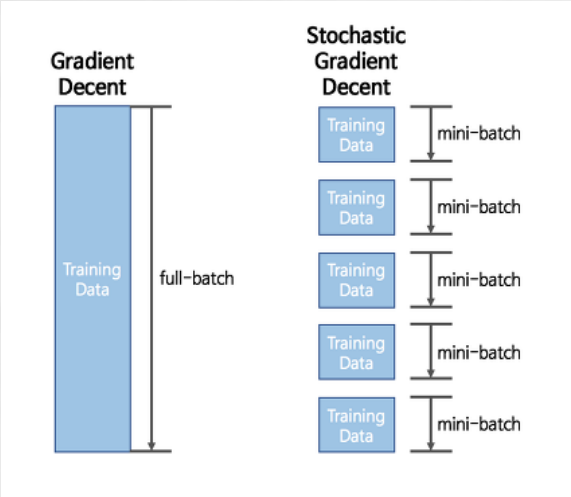
GD(Gradient Descent) vs SGC(Stochastic Gradient Descent)

- GD

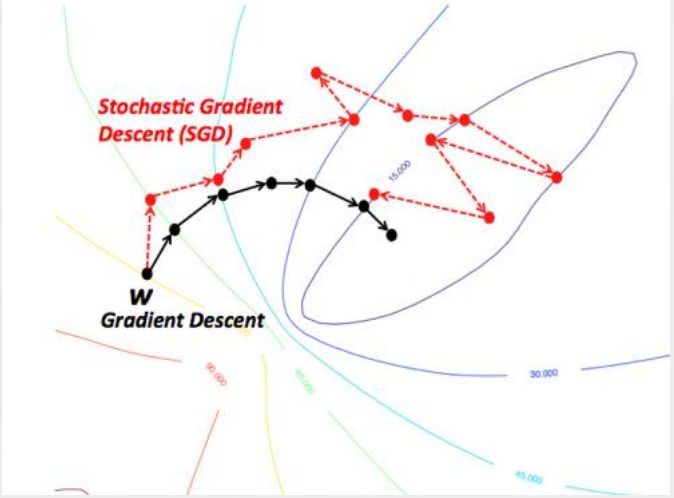
모든 데이터 계산(소요시간 1시간)
속도 느림 전진
수렴 속도 느림, 일정한 운동량(Momentum)

- SGD

일부 데이터만 계산(소요시간 5분)
빠르게 전진한다.
수렴 속도 빠름, 심한 운동량(Momentum)



Batch size



수렴 과정

1) SGD : 확률적 경사하강

```
keras.optimizers.SGD(lr=0.01, momentum=0.0, decay=0.0, nesterov=False)
```

운동량(momentum), 학습 속도 감소(decay), 운동량 적용(nesterov)

lr : float> = 0. 학습 속도.
momentum : float> = 0. SGD 가속, 진동을 줄이는 변수
decay : float> = 0. 각 업데이트에 대한 학습 속도 감소
nesterov : 운동량 적용 여부

2) RMSprop

```
keras.optimizers.RMSprop(lr=0.001, rho=0.9, epsilon=None, decay=0.0)
```

매개 변수를 기본값으로 두는 것이 좋다. (학습 속도는 자유롭게 조정 가능)
대부분 반복적인 순환신경망(RNN)에 적합

3) Adagrad

```
keras.optimizers.Adagrad(lr=0.01, epsilon=None, decay=0.0)
```

학습 과정에서 매개변수의 Update가 많아질 수록 학습 속도 더 작아짐
학습 속도 자동 조정, 매개 변수를 기본값으로 두는 것이 좋다.

4) Adadelta

```
keras.optimizers.Adadelta(lr=1.0, rho=0.95, epsilon=None, decay=0.0)
```

Adagrad 확장판, 매개 변수를 기본값으로 두는 것이 좋다.

5) Adam(*Adaptive moment estimation)*

```
keras.optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=None,
decay=0.0, amsgrad=False)
```

학습 속도를 조정하는 Adagrad 확장판,매개 변수를 기본값으로 두는 것이 좋다.

4. Keras Model

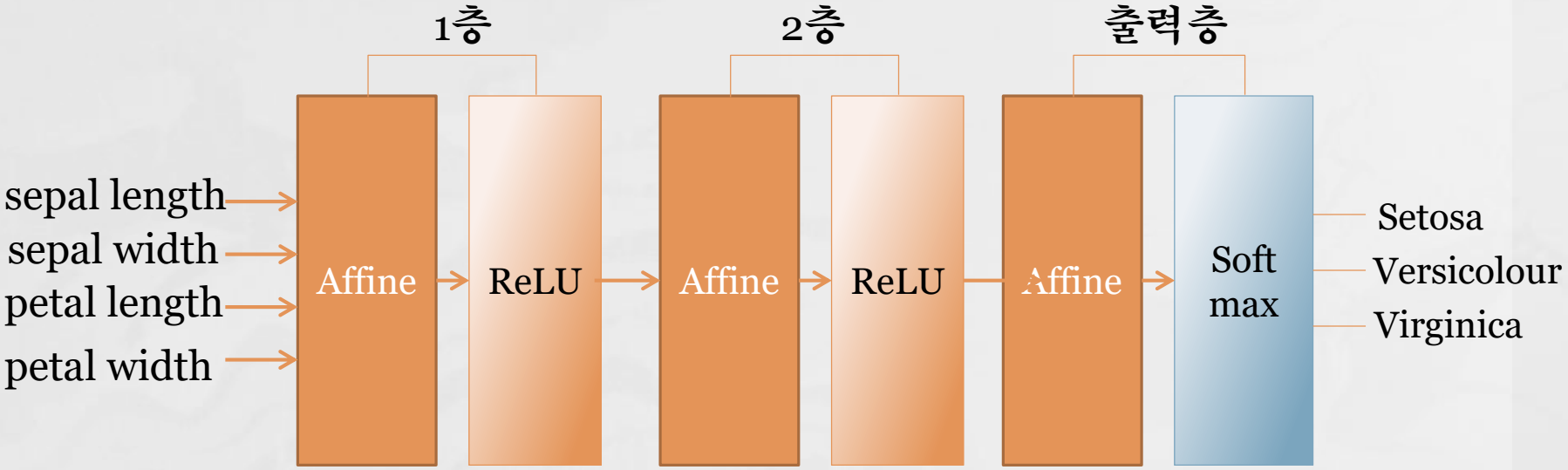
```
# 모델 학습 환경
model.compile(optimizer = 'adam',
              loss = 'categorical_crossentropy', # one hot encoding
              metrics = ['accuracy'])

# 모델 학습
model.fit(x_train, y_train, # 훈련 data
          epochs = 500, # 학습횟수
          verbose=1, # 출력여부
          validation_data=(x_test, y_test)) # 검증 data

# 모델 평가
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

주요 Model 환경

```
# 회귀 모델 학습 환경
model.compile(optimizer = 'sgd',
              loss = ‘mse',
              metrics = [‘mae'])

# 이항 분류기 모델 학습 환경
model.compile(optimizer = 'adam ',
              loss = 'binary_crossentropy',     # one hot encoding
              metrics = [‘'accuracy'])

# 다항 분류기 모델 학습 환경
model.compile(optimizer = 'adam ',
              loss = 'categorical_crossentropy',   # one hot encoding
              metrics = [‘'accuracy'])
```

5. Keras DNN layer

- iris dataset 적용




```
model = Sequential()

# hidden layer1 : shape = [4, 12]
model.add(Dense(12, input_shape=(4,), activation = 'relu')) # 1층(hidden1)
# hidden layer2 : shape = [12, 6]
model.add(Dense(6, activation = 'relu')) # 2층(hidden2)
# hidden layer2 : shape = [6, 3]
model.add(Dense(3, activation = 'softmax')) # 출력층(output)
```

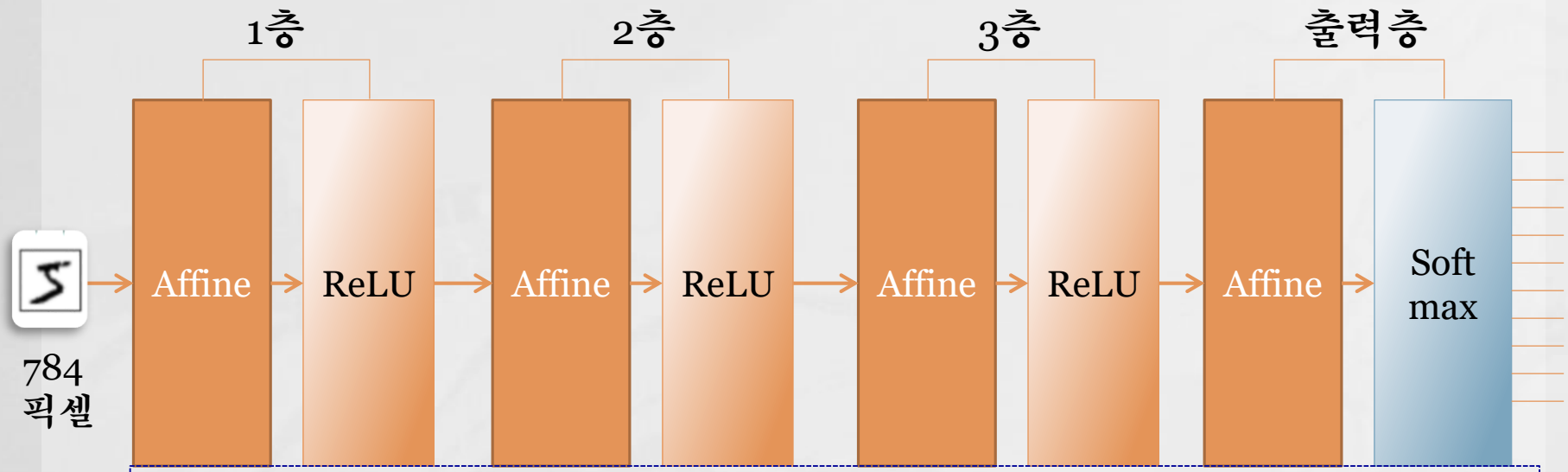
Tensorflow DNN(ver1.x)

```
hideen1_nodes = 12
hideen2_nodes = 6

# Hidden1 layer : 1층
w1 = tf.Variable(tf.random_normal([4, hideen1_nodes])) # [X_in, h1]
b1 = tf.Variable(tf.random_normal([hideen1_nodes]))# [h1]
hidden1_out = tf.nn.relu(tf.matmul(X, w1) + b1)

# Hidden2 layer : 2층
w2 = tf.Variable(tf.random_normal([hideen1_nodes, hideen2_nodes]))#[h1, h2]
b2 = tf.Variable(tf.random_normal([hideen2_nodes]))# [h2]
hidden2_out = tf.nn.relu(tf.matmul(hidden1_out, w2) + b2)

# Output layer : 3층
w3 = tf.Variable(tf.random_normal([hideen2_nodes, 3])) #[h2, Y_out]
b3 = tf.Variable(tf.random_normal([3])) #[Y_out]

# Softmax 분류기(1~3)
# 1.model
model = tf.matmul(hidden2_out, w3) + b3 # output 계산

# softmax(예측치)
softmax = tf.nn.softmax(model) # 0~1 확률값(전체 합=1)

# 2. cost function : sotfmax + entropy
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(
    labels=Y, logits=model))

# 3. Optimizer
train = tf.train.AdamOptimizer(0.01).minimize(cost)
```

Keras DNN(ver2.x)

```
# Hidden layer : 1층 [4, 12]
model.add(Dense(12, input_shape = (4,), activation = 'relu'))

# hidden layer : 2층 [12, 6]
model.add(Dense(6, activation = 'relu'))

# output layer : 3층 [6, 3]
model.add(Dense(3, activation = 'softmax'))


# model compile : Softmax 분류기  설정
model.compile(optimizer = 'adam',
          loss = 'categorical_crossentropy', # one hot encoding
          metrics = ['accuracy'])
```

- MNIST dataset 적용




```
model = Sequential()

# hidden layer1 : shape = [784, 128]
model.add(Dense(128, input_shape=(784,), activation = 'relu')) # 1층(hidden1)
# hidden layer2 : shape = [128, 64]
Model.add(Dense(64, activation =  ' relu ' )) # 2층(hidden2)
# hidden layer3 : shape = [64, 32]
Model.add(Dense(32, activation =  ' relu ' )) # 3층(hidden3)
# output layer : shape = [32, 10]
Model.add(Dense(10, activation =  ' softmax')) # 출력층(output)
```

Tensorflow DNN(ver1.x)

```
hidden1_nodes = 128
hidden2_nodes = 64

# Hideen1 layer : 1층
W1 = tf.Variable(tf.random_normal([784, hidden1_nodes]))
b1 = tf.Variable(tf.random_normal([hidden1_nodes]))
L1 = tf.nn.relu(tf.matmul(X, W1) + b1)

# Hideen2 layer : 2층
W2 = tf.Variable(tf.random_normal([hidden1_nodes, hidden2_nodes]))
b2 = tf.Variable(tf.random_normal([hidden2_nodes]))
L2 = tf.nn.relu(tf.matmul(L1, W2) + b2)

# Output layer : 3층
W3 = tf.Variable(tf.random_normal([hidden2_nodes, 10]))
b3 = tf.Variable(tf.random_normal([10]))

# 1. model
model = tf.matmul(L2, W3) + b3

# softmax(예측치)
softmax = tf.nn.softmax(model) # 0~1 확률값(전체 합=1)

# 2. cost function : sotfmax + entropy
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(
    labels=Y, logits=model))

# 3. Optimizer
train = tf.train.AdamOptimizer(lr).minimize(cost)
```

Keras DNN(ver2.x

```
# Hidden layer : 1층
model.add(Dense(128, input_shape = (784,), activation = 'relu'))

# hidden layer : 2층 [128, 64]
Model.add(Dense(64, activation =  ' relu ' ))

# hidden layer : 3층 [64, 32]
model.add(Dense(32, activation = 'relu'))

# output layer : 4층 [32, 10]
model.add(Dense(10, activation = 'softmax'))

# model compile : Softmax 분류기 설정
model.compile(optimizer = 'adam',
        loss = 'categorical_crossentropy', # one hot encoding
        metrics = ['accuracy'])
```

- Keras DNN model

```
# model compile : 학습과정 설정
model.compile(optimizer = 'adam', # 'adam', 'sgd'
              loss='categorical_crossentropy', # 'binary_crossentropy','mse'
              metrics = ['accuracy']) # ['mae']

# model train : train set(1회 학습 data : 100 image feed)
model.fit(x_train, y_train, batch_size=100, epochs=10)

# model test : test set
score = model.evaluate(x_test, y_test)
```

optimizer : 최적화 알고리즘
loss : 손실함수(cost function)
metrics : model 평가

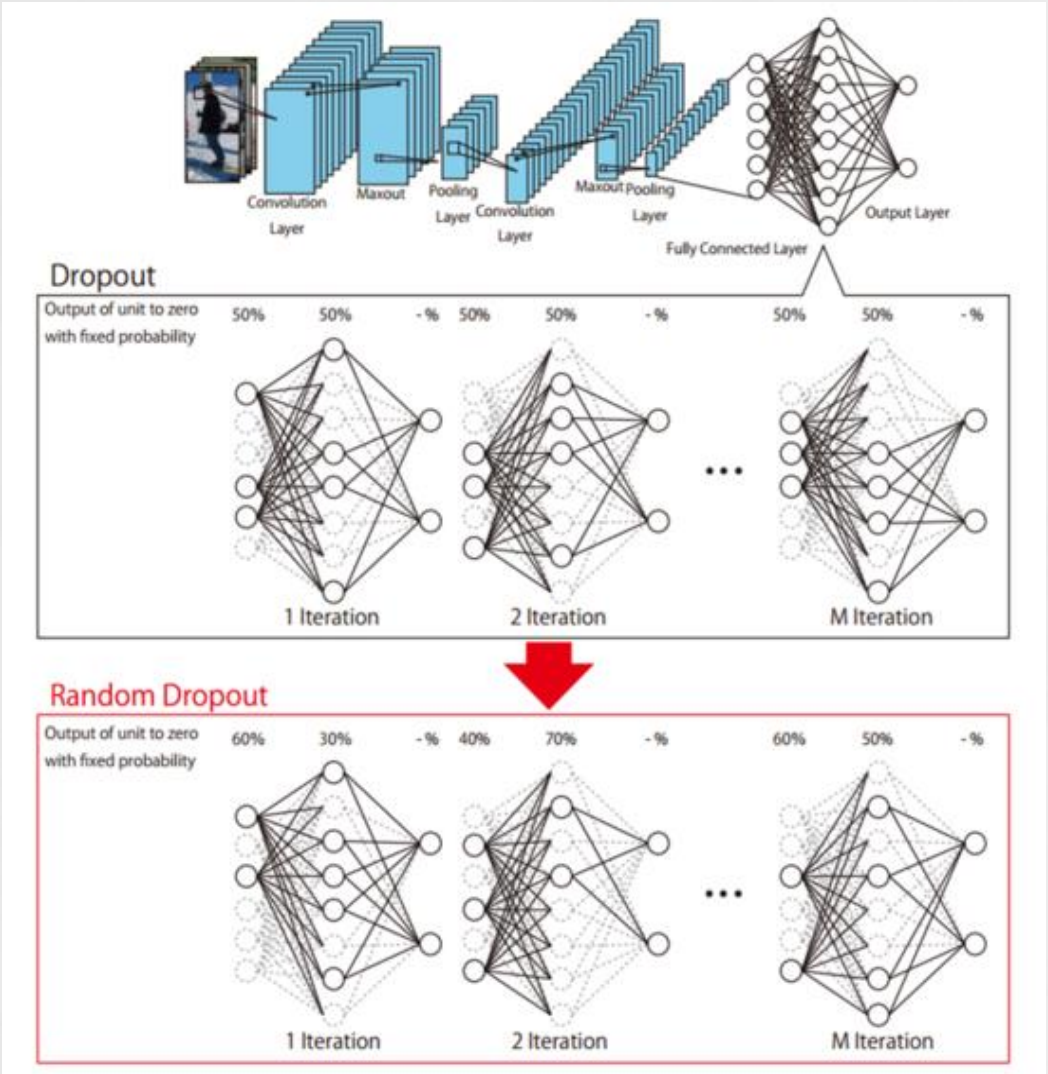
6. Deep Learning Overfitting

- 딥러닝 과적합(Over Fitting) 원인
  - ✓ 훈련데이터 적은 경우, 가중치가 큰 경우
  - ✓ Layer, Node 많은 모델(표현력이 높은 모델)
- 해결방안 : 정형화(Regularization)
  - ✓ 가중치 감소(weight decay) : 큰 가중치에 대한 패널티 부과 방법
    - L1법칙(Rasso) : cost + $\lambda |W|$
    - L2법칙(Ridge) : cost + $1/2\lambda W^2$

    ❖λ : 정형화 세기 조절 Hyper 파라미터

    Ex) reg = 0.001 * tf.reduce_sum(tf.square(W))
  - ✓ 드롭아웃(Drop out) : 뉴런 임의 삭제
    - 훈련 시 은닉층의 뉴런 무작위 삭제, 신호 전달 차단

1) Dropout & Random Dropout



매 학습시 은닉층에서 모든 neuron을 사용하는게 아니라 70%정도의 neuron을 사용
하나의 딥러닝에서 여러 개의 작은 neural net이 앙상블되어진 효과가 있고, 앙상블은 과적합을 크게 줄어준다.

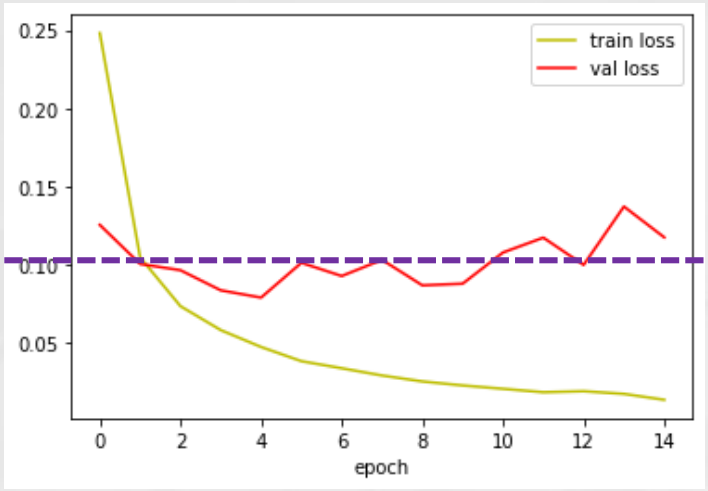
또한, 비슷한 weight를 갖는 뉴런들이 줄어들게 되어서(중복된 판단을 하는 뉴런들이 줄어들게 되어) 뉴런을 효율적으로 사용 가능

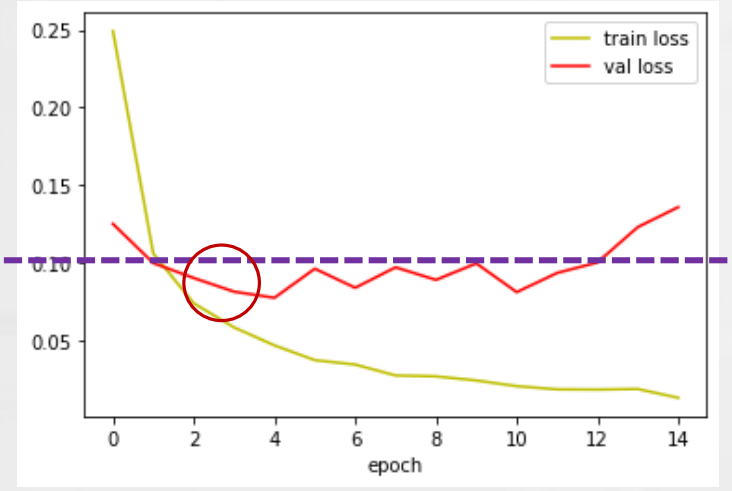
Keras layer Dropout 적용 예

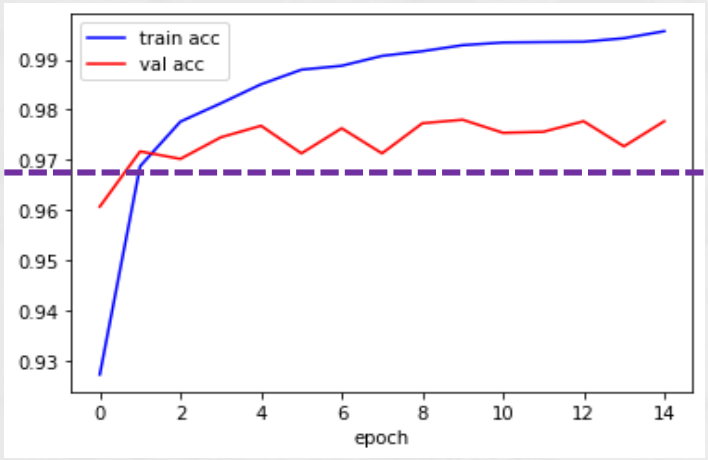
```
# keras DNN model layer

model.add(Dense(128, input_shape=(784,), activation='relu'))#1층
Dropout(rate = 0.5)

model.add(Dense(64, activation = 'relu')) # 2층
Dropout(rate = 0.3)

model.add(Dense(32, activation = 'relu')) # 3층
Dropout(rate = 0.3)

model.add(Dense(10, activation = 'softmax')) # 4층
```

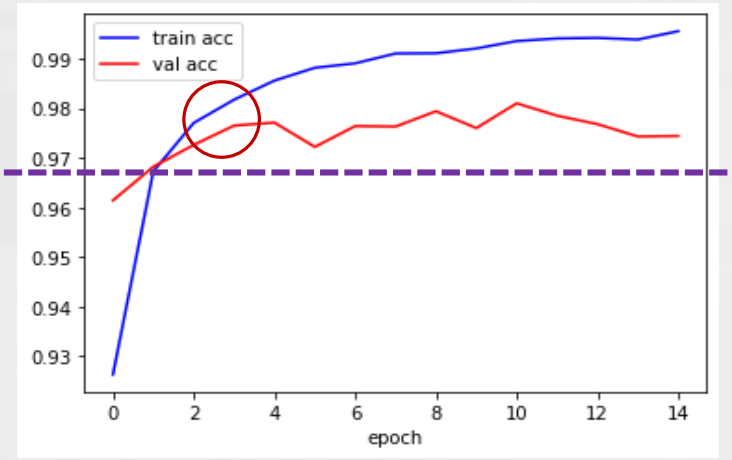
Output layer
적용 안함

Dropout 적용 전과 후







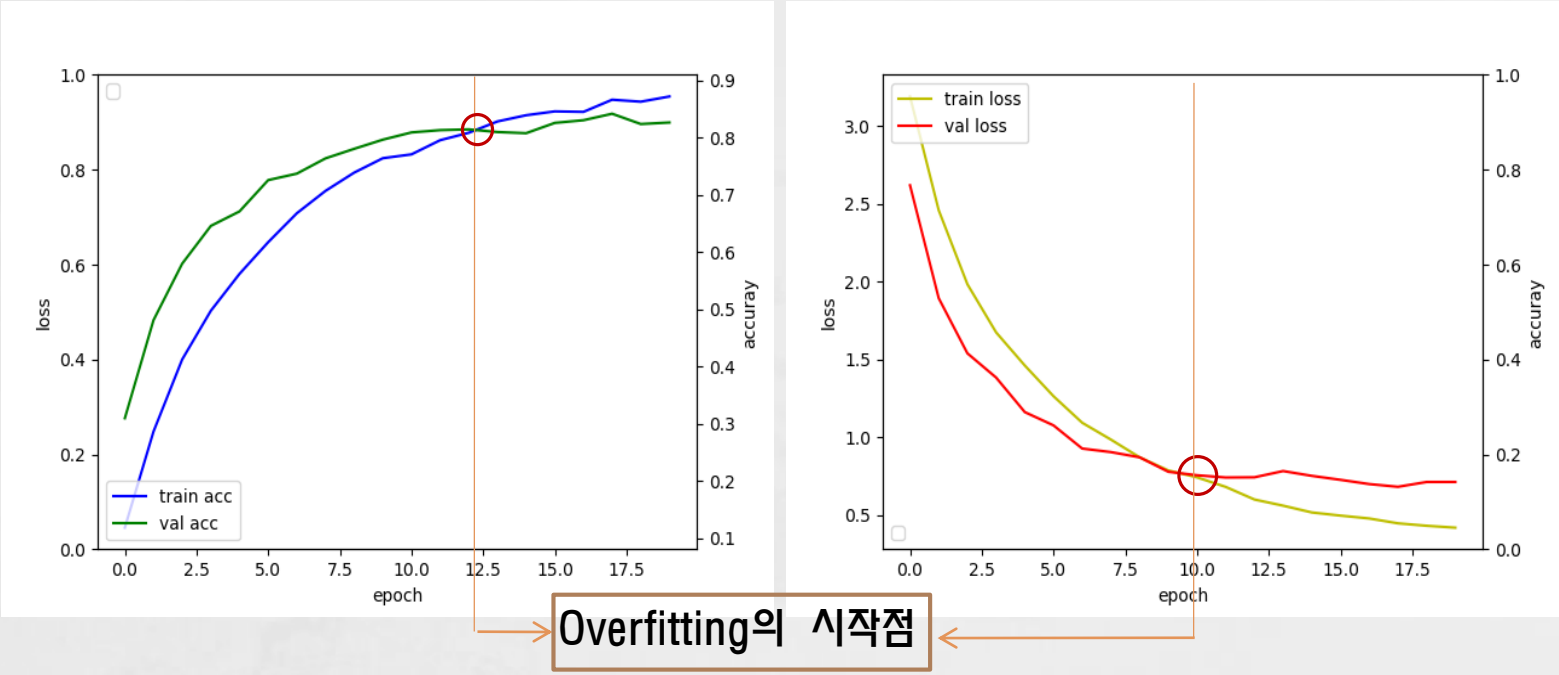


Dropout 전

Dropout 후

2) 과적합(overfitting)을 고려한 반복회수 결정

✓ Train 예측치와 Test 예측치의 교차점



3) EarlyStopping

✓ Epoch 수행 중 특정 시점에서 학습 조기 종료 기법

```
callback = EarlyStopping(monitor='val_loss', patience=2)
# epoch=2 이후 평가 손실이 개선되지 않으면 조기 종료

model_fit = model.fit(x_train, y_train,
        epochs = 15, # 학습횟수
        verbose=1, # 출력여부
        validation_data=(x_test, y_test), # 검증data
        callbacks=[callback]) # 조기종료
```

- **EarlyStopping 적용 결과**

  ✓ 15 Epoch 수행 중 7 Epoch에서 학습 조기 종료