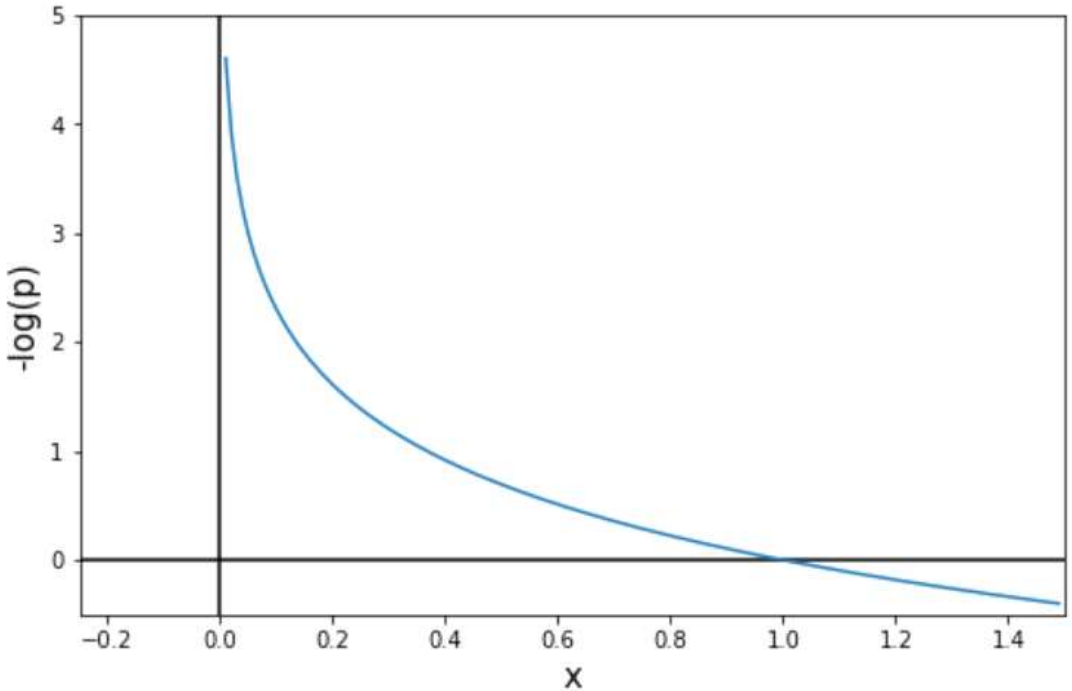
혼자 공부하는 머신러닝 + 딥러닝 7-3장

손실함수: 크로스 엔트로피 비용함수

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K} y_i \log(\hat{p}_i)$$

$\theta = (w_1\ w_2\ w_3\ ...)^T$ 가중치들의 벡터

y_i: 타깃 데이터, $\hat{p}_i$: 예측확률(softmax)



경사 하강법(기본형)

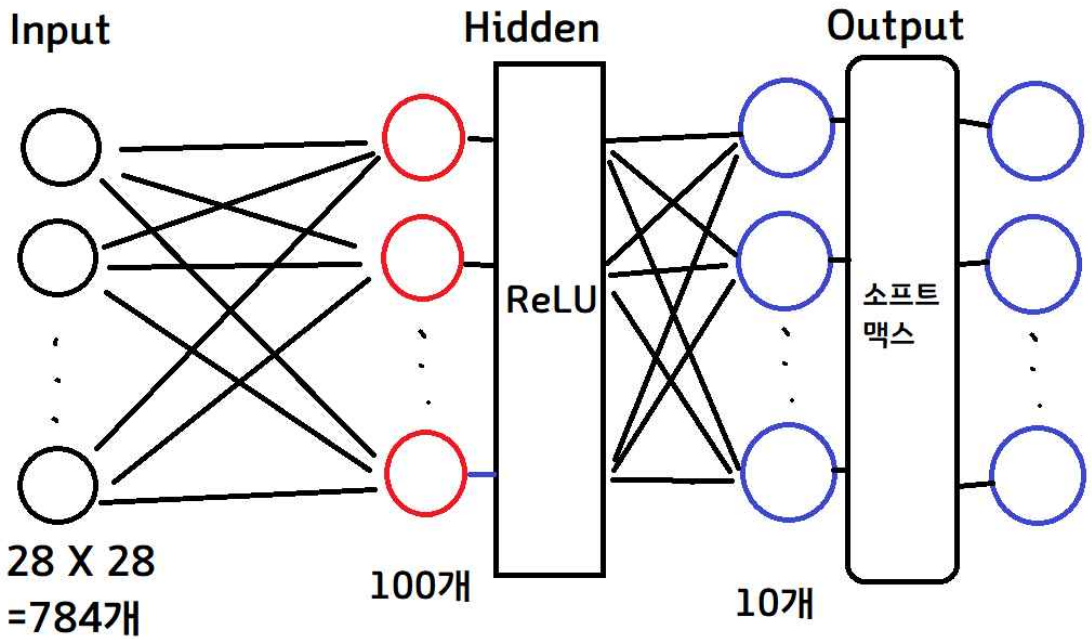
$$\theta^{+} = \theta - \gamma \nabla J(\theta)$$

γ: learning rate

신경망 모델 만들기 (Keras)

```
# 신경망 모델 만드는 과정 간소화 함수 정의
def model_fn(a_layer=None):
  model = keras.Sequential()
  model.add(keras.layers.Flatten(input_shape=(28, 28)))
  model.add(keras.layers.Dense(100, activation='relu'))
  if a_layer:
    model.add(a_layer)
  model.add(keras.layers.Dense(10, activation='softmax'))
  return model
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_1 (Flatten)          (None, 784)               0
_________________________________________________________________
dense (Dense)                (None, 100)               78500
_________________________________________________________________
dense_1 (Dense)              (None, 10)                1010
=================================================================
Total params: 79,510
Trainable params: 79,510
Non-trainable params: 0
_________________________________________________________________
```

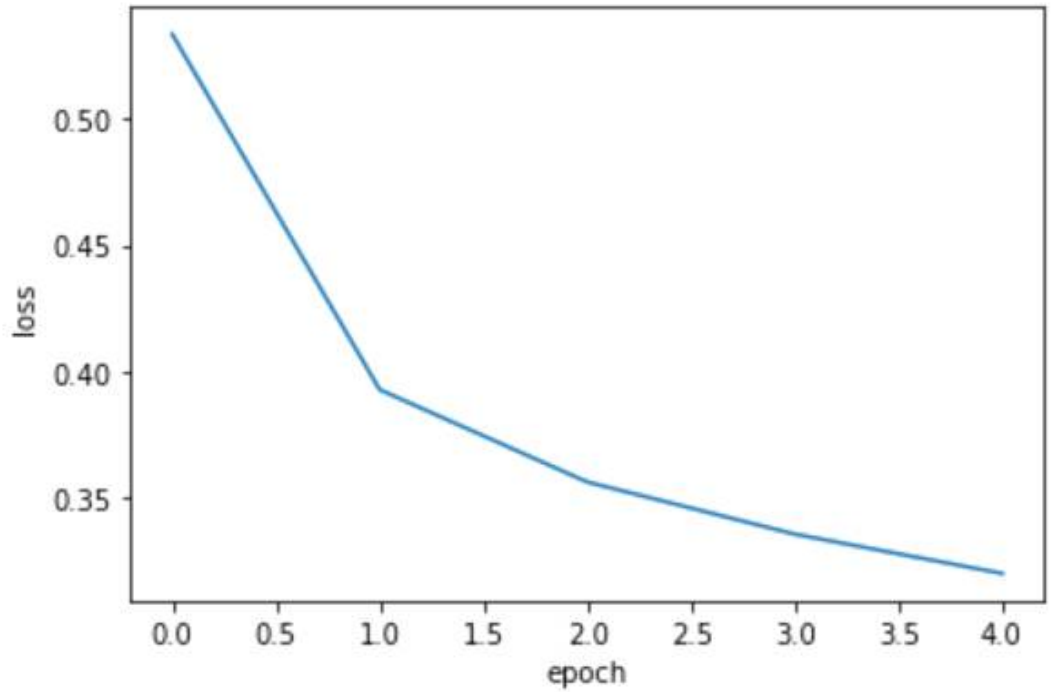


손실 곡선

fit() 메서드는 각 에포크마다 손실함수를 계산하여 딕셔너리 형태로 저장

```
# 모델을 훈련하고 fit() 메서드의 결과를 history 변수에 담기
model.compile(loss = 'sparse_categorical_crossentropy', metrics='accuracy')
history = model.fit(train_scaled, train_target, epochs=5, verbose=0)
```

```
# 에포크마다 손실함수 변화 그래프 그리기
import matplotlib.pyplot as plt
plt.plot(history.history['loss'])
plt.xlabel('epoch')
plt.ylabel('loss')
plt.show()
```

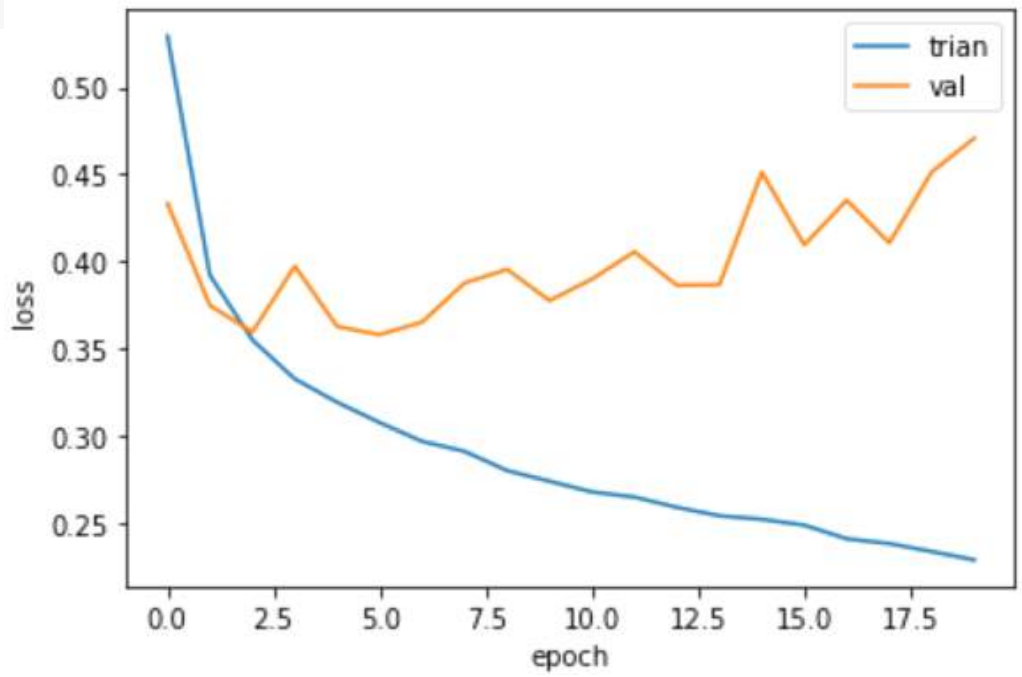


에포크는 다다익선? 과대적합!

과대적합: 훈련 세트에 과도하게 맞춰져 다른 데이터에는 성능 저하

```
# 검증 데이터 사용
model = model_fn()
model.compile(loss = 'sparse_categorical_crossentropy', metrics = 'accuracy')
history = model.fit(train_scaled, train_target, epochs = 20, verbose = 0,
                    validation_data = (val_scaled, val_target))
```

```
# 에포크마다 훈련 손실과 검증 손실의 변화 그리기
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.xlabel('epoch')
plt.ylabel('loss')
plt.legend(['trian', 'val'])
plt.show()
```



최적화 (경사 하강법) 알고리즘 바꾸기

모멘텀 최적화: (마찰 있는 경사면을 굴러가는 공처럼) 등속도 운동 전까지 가속

그래디언트 $\nabla J(\theta)$를 속도가 아닌 가속도처럼 활용

$$m \leftarrow \beta m + \gamma \nabla J(\theta), \qquad \theta \leftarrow \theta + m$$

RMSprop: 학습률(γ)를 점점 감소시킴, 경사가 가파른 경우에 더 빠르게 감소

$$s \leftarrow \beta s + (1-\beta) \nabla J(\theta) \otimes \nabla J(\theta)$$

$$\theta \leftarrow \theta - \gamma \nabla J(\theta) \oslash \sqrt{s + \epsilon}$$

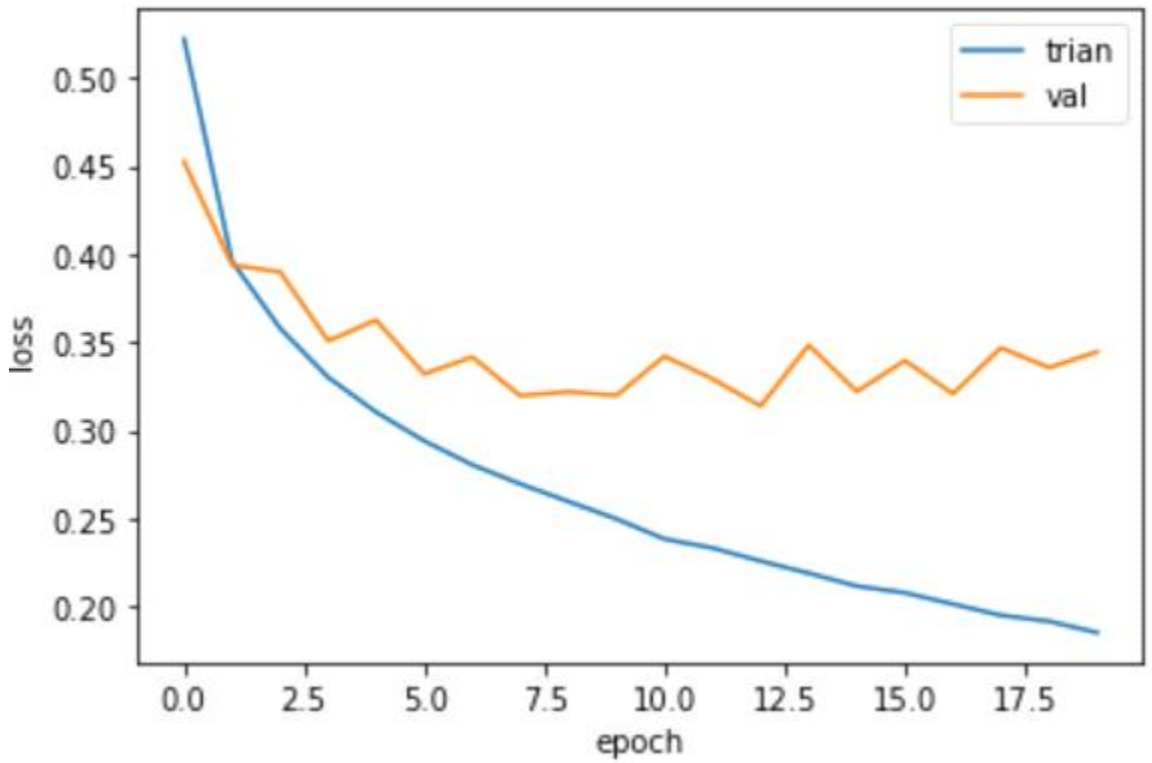
Adam (adaptive moment estimation) 최적화: 위 두 방법의 장점을 합침

(「Adam: A Method for Stochastic Optimization」, D.Kingma, J. Ba (2015), https://arxiv.org/pdf/1412.6980v8.pdf)

Adam 옵티마이저 적용

```
# 옵티마이저로 Adam 알고리즘 사용
model = model_fn()
model.compile(optimizer = 'adam', loss = 'sparse_categorical_crossentropy', metrics = 'accuracy')
history = model.fit(train_scaled, train_target, epochs = 20, verbose = 0,
                    validation_data = (val_scaled, val_target))
```

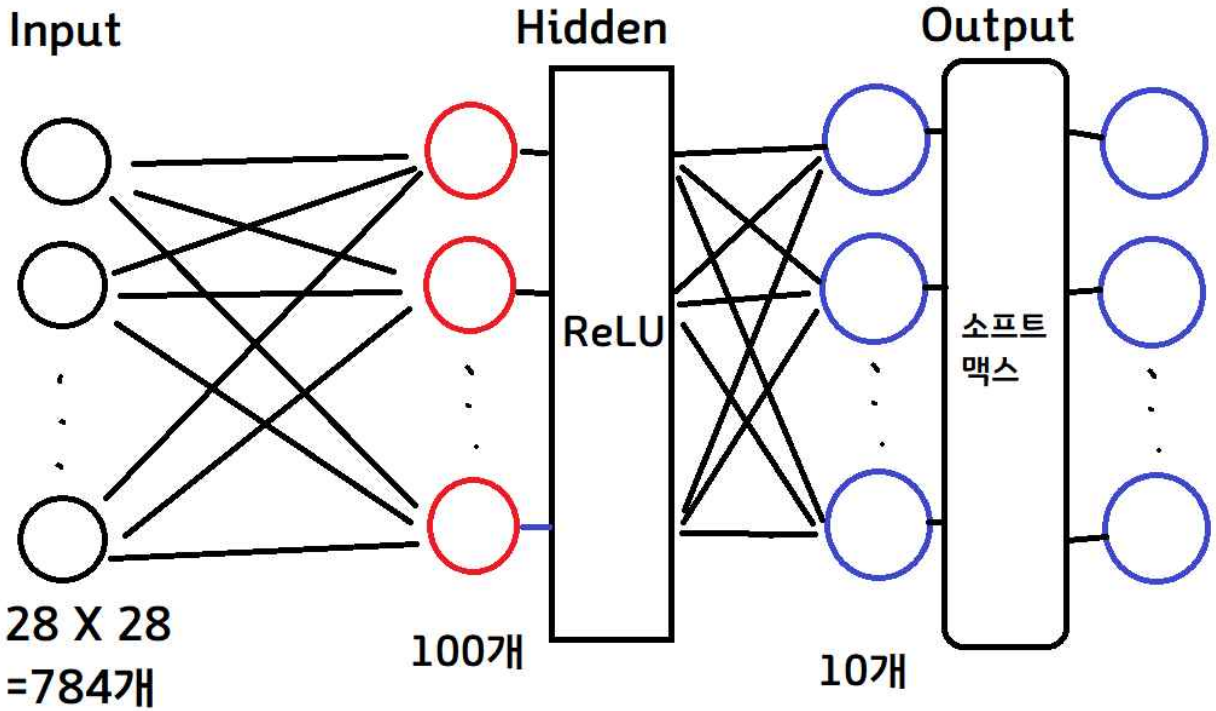
```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.xlabel('epoch')
plt.ylabel('loss')
plt.legend(['trian', 'val'])
plt.show()
```



이전보다 과대적합이 개선됨.

드롭아웃(Dropout)

은닉층의 일부 뉴런을 랜덤하게 비활성화(각 에포크마다)



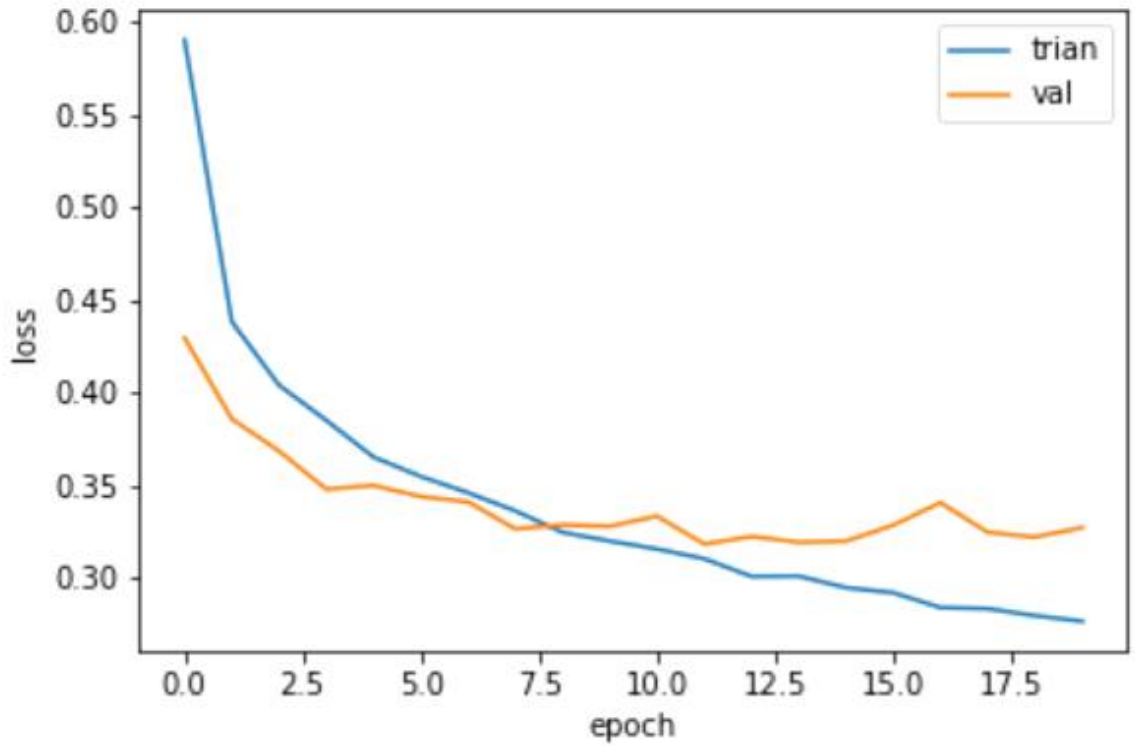
특정 뉴런을 과대하게 의존하는 것을 줄일 수 있음

모든 입력에 주의를 기울여야 함

드롭아웃(Dropout)

```
# 드롭아웃 이용하여 모델 훈련
model = model_fn(keras.layers.Dropout(0.3))
model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_5 (Flatten)          (None, 784)               0
_________________________________________________________________
dense_8 (Dense)              (None, 100)               78500
_________________________________________________________________
dropout (Dropout)            (None, 100)               0
_________________________________________________________________
dense_9 (Dense)              (None, 10)                1010
=================================================================
Total params: 79,510
Trainable params: 79,510
Non-trainable params: 0
_________________________________________________________________
```



과대적합이 개선됨.

조기 종료(Early Stopping)

콜백: 훈련 과정 중간에 작업 수행

Keras의 keras.callbacks 패키지 아래에 있는 클래스들로 사용

keras.callbacks.ModelCheckpoint(): 최상의 검증 점수를 만드는 모델을 저장

keras.callbacks.EarlyStopping(): 과대적합이 커지기 전에 (검증 세트의 손실 점수가 커지기 시작할 때) 훈련을 종료하여 리소스 사용 절약

keras.callbacks.EarlyStopping(patience=2, restore_best_weigts=True)

두 번 연속 검증 점수가 향상되지 않으면 훈련을 중지하고, 다시 최상의 파라메터로 되돌림(2 에포크 전으로)

조기 종료(Early Stopping)

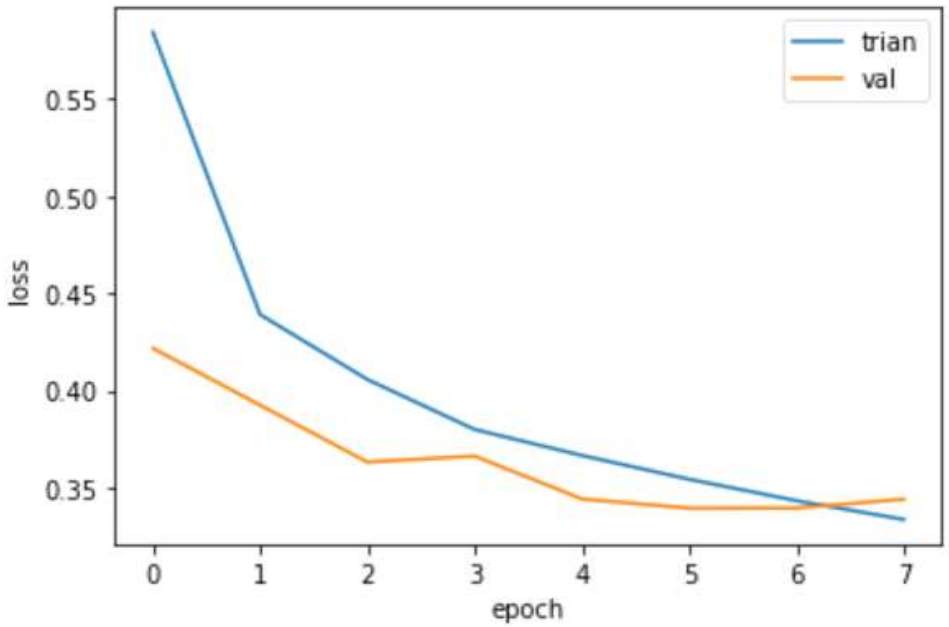
```
# 조기종료 콜백 적용하기
model = model_fn(keras.layers.Dropout(0.3))
model.compile(optimizer = 'adam', loss = 'sparse_categorical_crossentropy', metrics = 'accuracy')
checkpoint_cb = keras.callbacks.ModelCheckpoint('best-model.h5')
early_stopping_cb = keras.callbacks.EarlyStopping(patience=2,
                                                  restore_best_weights=True)
history = model.fit(train_scaled, train_target, epochs = 20, verbose = 0,
                    validation_data = (val_scaled, val_target),
                    callbacks=[checkpoint_cb, early_stopping_cb])
```

```
# 어느 에포크에서 조기종료했는지 출력
print(early_stopping_cb.stopped_epoch)
```

```
7
```

에포크는 0부터 시작, 8번째 에포크에서 조기종료. 최상의 모델은 2 에포크 전인 6번째 에포크의 모델

조기 종료(Early Stopping)



6번째 에포크 (그래프에서 5)에서 검증세트의 손실 점수가 가장 낮음.

```
# 모델 성능 확인
model.evaluate(val_scaled, val_target)
```

```
375/375 [==============================] - 1s 2ms/step - loss: 0.3394 - accuracy: 0.8775
[0.3393799960613251, 0.8774999976158142]
```

참고 자료

- 박해선, *혼자 공부하는 머신러닝 + 딥러닝* (한빛미디어, 2021), 392-411
- 오헬리앙 제롱, *핸즈온 머신러닝 1판* (한빛미디어, 2018), 376-384