# 1부 딥러닝의 기초 - 3장 신경망 시작하기

| | |
|---|---|
| 시작일 | @2022년 7월 19일 |
| 링크 | |
| 목표 | 월/0725 |
| 종료일 | |

## 3장 신경망 시작하기

이진분류, 다중 분류, 스칼라 값을 예측ㅎ나느 회귀에 배운 것들을 적용하는 장

2장 층, 네트워크, 목적 함수, 옵티마이저 같은 핵심 구성 요소 살펴봄

- 영화 리뷰를 긍정 또는 부정으로 분류하기(이진 분류)
- 신문 기사를 토픽으로 분류하기(다중 분류)
- 부동산 데이터를 바탕으로 주택 가격을 예측하기(회귀)

### 3.1 신경망의 구조

- `layers:` 네트워크(또는 모델)을 구성하는 층
- `input & output` : 입력 데이터와 그에 상응하는 타깃
- `loss function:` 학습에 사용할 피드백 신호를 정의하는 손실 함수
- `optimizer` : 학습 진행 방식을 결정하는 옵티마이저

#### 3.1.1 층: 딥러닝의 구성 단위

- 층은 상태가 없지만 대부분의 경우 `가중치`라는 층의 상태를 가짐

일반적으로 적절한 텐서 포맷과 데이터 처리 방식이 다름

  - 2D텐서 완전 연결층(fully connected layer) / 밀집 층(dense layer)
  - 3D텐서(시퀀스 데이터): LSTM 같은 순환 층(recurrent layer)
  - 4D텐서: 2D 합성곱 층(convolution layer)

- 호환 가능한 층들을 엮어 데이터 변환 파이프라인(pipeline)을 구성

- 층 호환성(layer compatibility)
  - 층이 특정 크기의 입력 텐서만 받고
  - 특정 크기의 출력 텐서를 반환함

```
# 1. 첫 번째 차원이 784인 2D 텐서만 입력으로 받는 층
from keras import layers
layer = layers.Dense(32, input_shape = (784,)) # 32개의 유닛으로 된 밀집 층

# 32로 변환된 텐서로 출력할 것임
# 32차원의 벡터를 입력으로 받는 하위 층이 연결되어야 함

from keras import models
from keras import layers

model = models.Sequential()
model.add(layers.Dense(32, input_shape = (784,)))
model.add(layers.Dense(10))
```

## 3.1.2 모델: 층의 네트워크

- 딥러닝 모델: 층으로 만든 비순환 유향 그래프(Directed Acyclic Graph, DAG) 네트워크 구조
  - 가지(brach)가 2개인 네트워크
  - 출력이 여러 개인 네트워크
  - 인셉션(Inception) 블록

## 3.1.3 손실 함수와 옵티마이저: 학습 과정을 조절하는 열쇠

- 네트워크 구조를 정의하고 1) 손실 함수 2) 옵티마이저 선택해야 함
- `손실 함수(loss function) 목적 함수(objective function)`
  - 훈련하는 동안 최소화될 값
  - 주어진 문제에 대한 성공 지표
  - 분류, 회귀, 시퀀스 예측 같은 일반적 문제 → 올바른 손실 함수

    분류
    - 2개의 클래스가 있는 분류 문제 (이진 크로스엔트로피 binary crossentropy))
    - 여러 개의 클래스가 있는 분류 문제 (범주형 크로스엔트로피(categorical crossentropy))

회귀

- 평균 제곱 오차

시퀀스 학습 문제

- CTC(Connection Temporal Classification)

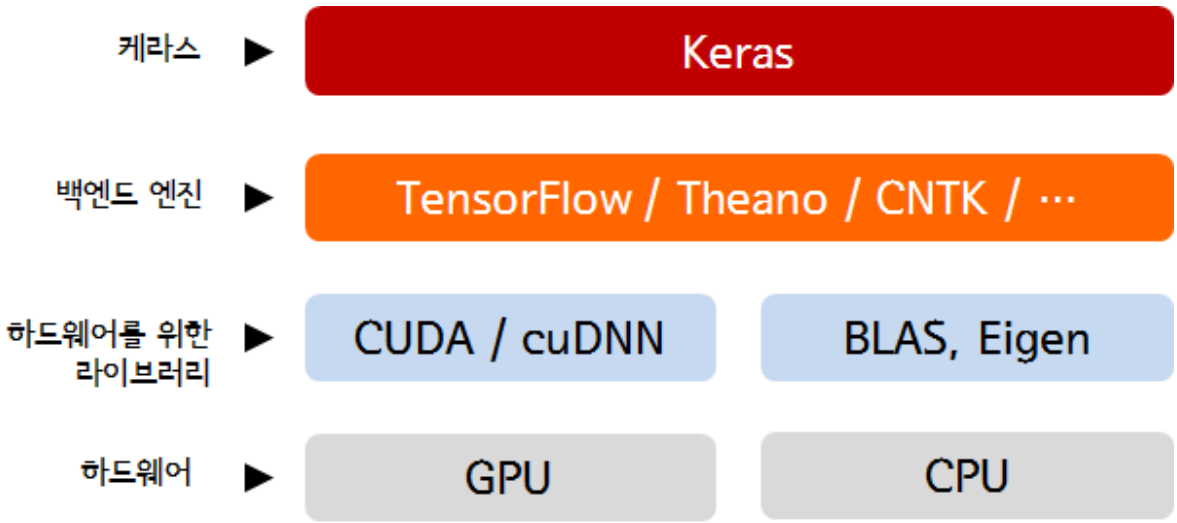
- `옵티마이저(optimzier)`
  - 손실 함수를 기반으로 네트워크가 어떻게 업데이트될지 결정함
  - 특정 종류의 확률적 경사 하강법(SGD) 을 구현

## 3.2 케라스 소개

- Keras: 파이썬 딥러닝 프레임워크
  - 동일한 코드로 CPU & GPU로 실행할 수 있음
  - 사용하기 쉬운 API
  - MIT 라이선스를 이용 → 상업적 프로젝트에서도 자유롭게 사용할 수 있음

### 3.2.1 케라스, 텐서플로, 씨아노, CNTK

- 케라스의 백엔드 엔진(backend engine)에서 제공하는 최적화,특화된 텐서 라이브러리를 사용
- 모듈 구조로 구성 → 백엔드 엔진이 케라스와 연동됨
- 텐서플로, 씨아노, CNTK 3개를 백엔드 엔진으로 사용할 수 있음




### 3.2.2 케라스를 사용한 개발: 빠르게 둘러보기

케라스 모델

- 입력 텐서, 타켓 텐서로 이루어진 훈련 데이터를 정의
- 입력 & 타깃을 매핑하는 층으로 이루어진 네트워크(또는 모델)을 정의
- 손실 함수, 옵티마이저, 모니터링하기 위한 측정 지표를 선택 → 학습 과정을 설정
- 훈련 데이터에 대해 모델의 fit() 메서드를 반복적으로 호출합니다.
- `모델 종류 : Sequential, Functional`
  - Sequential
    - 층을 순서대로 쌓아 올린 네트워크

```
# Sequential

from keras import layers
from keras import models
from keras import optimizers

# 모델 네트워크 정의
model = models.Sequential()
model.add(layers.Dense(32, activation = 'relu', input_shape = (784,))
model.add(layers.Dense(10, activation = 'softmax'))

# 컴파일
model.comiple(loss = 'mse',
              optimizer =optimizer.RMSprop(lr = 0.001),
              metrics = ['accuracy'] )

# 모델 학습
model.fit(input_tensor, target_tensor, batch_size = 128, epoch = 10)
```

  - Functional (합수형 API)
    - 완전히 임의의 구조를 만들 수 있는 비순환 유향 그래프

```
# Functional
  # 모델이 처리할 데이터 텐서를 만들고
  # 마치 함수처럼 이 텐서에 층을 적용함

# 모델 네트워크 정의
input_tensor = layers.Input(shape = (784,))
x = layer.Dense(32, activation = 'relu')(input_tensor)
output_tensor = layers.Dense(10, activation = 'softmax')(x)

model = models.Model(inputs = input_tensor, outputs = ouput_tensor)

# 컴파일
model.comiple(loss = 'mse',
```

```
                optimizer =optimizer.RMSprop(lr = 0.001),
                metrics = ['accuracy'] )

# 모델 학습
model.fit(input_tensor, target_tensor, batch_size = 128, epoch = 10)
```

## 3.3 딥러닝 컴퓨터 셋팅

### 3.3.1 주피터 노트북: 딥러닝 실험을 위한 최적의 방법

### 3.3.2 케라스 시작하기: 두 가지 방법

### 3.3.3 클라우드에서 딥러닝 작업을 수행했을 때 장단점

### 3.3.4 어떤 GPU 카드가 딥러닝에 최적일까?

## 3.4 영화 리뷰 분류: 이진 분류 예제

### 3.4.1 IMDB 데이터셋

- IMDB(Internet Movie Database) : 인터넷 영화 데이터베이스
- 양극단의 리뷰 5만개 , 훈련 데이터셋 (2만 5천 훈련데이터 , 2만 5천 테스트데이터) - 긍부정 50%
- MNIST 처럼 Keras에 포함되어 있으며, 전처리되어 있어 리뷰가 숫자 시퀀스로 변환됨

```
# IMBD 데이터셋 로드하기
from keras.datasets import imdb

(train_data, train_labels), (test_data, test_labels) = imdb.load_data(num_words = 1000
0)

# num_words = 10000
  # 훈련 데이터에서 가장 자주 나타나는 단어 1만 개만 사용하겠다는 의미

# train_data, test_data
  # 리뷰의 목록
  # 각 리뷰는 단어 인덱스의 리스트 (단어 시퀀스 인코딩)

# train_labels, test_labels
  # 부정을 나타내는 0 , 긍정: 1

train_data[0] # [1,14,22,16,....]

train_labels[0] # 1

# 자주 등장한 단어 1만개, 단어 인덱스 9,999 넘지 않음
```

```
max([max(sequence) for sequence in train_data])  # 9999

word_index = imdb.get_word_index() # word_index- {단어:정수} dictionary
reverse_word_index = dict(
  [(value, key) for (key,value) in word_index.items()]) # 정수인덱스-단어 매핑
decoded_review = ' '.join(
  [reverse_word_index.get(i-3, '?') for i in train_data[0]]) # 리뷰를 인코딩
# 0,1,2 는 패딩, 문서 시작, 사전에 없음을 위한 인덱스이므로 3을 뺌
```

## 3.4.2 데이터 준비

- 신경망 숫자 리스트 대입 X → `리스트를 텐서로 바꿔줘야 함`

Method 1 : 임베딩

- 같은 길이가 되도록 리스트에 패딩(padding)추가
- (samples, sequence_length)크기의 정수 텐서로 변환
- 정수 텐서를 다룰 수 있는 층을 신경망의 첫 번째 층으로 사용 (Embedding)

Method 2: 리스트를 원-핫 인코딩

- 원핫인코딩(one-hot encoding) 하여 0,1의 벡터로 변환함

```
# 정수 시퀀스를 이진 행렬로 인코딩하기
import numpy as np

def vectorize_sequences(sequences, dimension = 10000):
  results= np.zeros((len(sequences), dimension)) # 크기:len(sequence), dimension /0행렬
  for i, sequence in enumerate(sequences):
    results[i, sequence] = 1. # results[i] 에서 특정 인덱스의 위치를 1로 만듦
  return results

x_train = vectorize_sequences(train_data) # 훈련 데이터를 벡터로 변환
x_test = vectorize_sequences(test_data) # 훈련 데이터를 벡터로 변환
```

## 3.4.3 신경망 모델 만들기

```
# 모델 정의하기
from keras import models
from keras import layers

model = models.Sequential()
model.add(layers.Dense(16, activation = 'relu', input = (10000,)))
model.add(layers.Dense(8, activation = 'relu'))
model.add(layers.Dense(1, activation = 'sigmoid'))

# 모델 컴파일하기
model.compile(optimizer = 'rmsprop',
```

```
              loss= 'binary_crossentropy',
              metrics = ['accuracy'])
```

```
# 자신만의 옵티마이저, 손실 함수, 측정 지표를 매개변수로 전달할 경우
# 1. 옵티마이저 설정하기
from keras import optimizers

model.compile(optimizer = optimizers.RMSprop(lr = 0.001),
              loss = 'binary_crossentropy',
              metrics = ['accuracy'])
```

```
# 2. 손실과 측정을 함수 객체로 지정하기
from keras import losses
from keras import metrics

model.compile(optimizer = optimizers.RMSprop(lr = 0.001),
              loss = losses.binary_crossentropy,
              metrics = ['accuracy'])
```

## 3.4.4 훈련 검증

```
# 검증 세트 준비하기
x_val = x_train[:10000]
partial_x_train = x_train[10000:]

y_val = y_train[:10000]
partial_y_train = [10000:]
```

- 512개 샘플씩 미니 배치를 만들어 20번의 에포크 동안 훈련

```
# 모델 훈련하기

model.compile(optimizer = 'rmsprop',
              loss= 'binary_crossentropy',
              metrics = ['acc'])

history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs = 20,
                    batch_size = 512,
                    validation_data = (x_val, y_val))
```

```
# model.fit 은 history객체를 반환 - > 훈련 정보 딕셔너리에 속성 가짐
history_dict= history.history
history_dict.keys() # acc, loss, val_acc, val_loss
```

```
# 훈련, 검증 그리기
import matplotlib.pyplot as plt

history_dict = history.history
loss = history_dict['loss']
val_loss= history_dict['val_loss']

epochs= range(1, len(loss) + 1)

plt.plot(epochs,loss, 'bo', label = 'Training loss') # 'bo'는 파란색 점을 의미
plt.plot(epochs, val_loss, 'b', label = 'Validation loss') # 'b'는파란색 실선을 의미
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
```

```
# 훈련과 검증 정확도 그리기
plt.clf() # 그래프 초기화
acc = history_dict['acc']
val_acc = history_dict['val_acc']

plt.plot(epochs, acc, label = 'training accuracy')
plt.plot(epochs, val_acc, label = 'validation accuracy')
plt.label('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
```

```
# 모델을 처음부터 다시 훈련하기
model = models.Sequential()
model.add(layers.Dense(16, activation = 'relu', input = (10000,)))
model.add(layers.Dense(16, activation = 'relu'))
model.add(layers.Dense(1, activation = 'sigmoid'))

# 모델 컴파일
model.compile(optimizer= 'rmsprop',
              loss = 'binary_crossentropy',
              metrics = ['accuracy'])

# 모델 훈련
mdodel.fit(x_train,
            y_train,
            epochs = 4,
            batch_size = 512)

results= model.evaluate(x_test, y_test)
```

## 3.4.5 훈련된 모델로 새로운 데이터에 대해 예측하기

```
# predict 메서드를 사용해서 어떤 리뷰가 긍정일 확률
model.predict(x_test)
```

## 3.4.6 추가 실험

```
# 1. 2개의 은닉층을 사용함
  # 1개 또는 3개의 은닉 층을 사용하고 검증과 테스트 정확도에 어떤 영향을 미치는지 보기

# 2. 층의 은닉 유닛을 추가/줄여보기
  # 32개 유닛, 64개의 유닛 등

# 3. binary_crossentropy 대신에 mse 손실 함수를 사용해보세요

# 4. relu 대신에 tanh 활성화 함수를 사용해 보세요

#---------- 모델 구성
model = models.Sequential()
model.add(layers.Dense(64, activation = 'relu', input= (10000, )))
model.add(layers.Dense(32, activation = 'relu'))
model.add(layers.Dense(16, activation = 'tanh'))
model.add(layers.Dense(8, activation = 'tanh'))
model.add(layers.Dense(1, activation= 'sigmoid'))

#---------- 모델 컴파일
model.compile( optimizer= 'rmsprop',
               loss= 'mse',
               metrics= ['accuracy'])

#---------- 모델 훈련
model.fit(x_train,
          y_train,
          epochs = 4,
          batch_size = 512)

#--------- 모델 평가
results= model.evaluate(x_test, y_test)
model.predict(x_test)
```

## 3.4.7 정리

- 원본 데이터를 신경망에 텐서로 주입하기 위해 많은 전처리 필요
  - 단어 시퀀스는 이진 벡터로 인코딩될 수 있고 다른 인코딩 방식도 있음
- relu 활성화 함수와 함께 Dense 층을 쌓은 네트워크
  - 여러 종류의 문제에 적용할 수 있음
- (출력 클래스가 2개인) 이진 부류 문제에서 네트워크는
  - 하나의 유닛과 sigmoid 활성화 함수를 가진 Dense 층으로 끝나야 함

    - 신경망의 출력은 확률을 나타내는 0/1 사이의 스칼라 값
- 이진 분류 문제에서 이런 스칼라 시그모이드 출력에 대해 사용할 손실 함수
    - binary_crossentropy
- rmsprop 옵티마이저
    - 문제에 상관없이 일반적으로 충분히 좋은 선택
- 훈련데이터에 대해 성능이 향상됨에 따라 신경망은 과대적합되기도 함
    - 이전에 본적 없는 데이터에서 결과가 점점 나빠지게 됨
    - 훈련 세트 이외의 데이터에서 성능을 모니터링 해야 함

# 3.5 뉴스 기사 분류 : 다중 분류 문제

## 3.5.1 로이터 데이터셋

- 로이터(Reuter) 뉴스: 46개 배타적 토픽으로 분류
- 다중 분류(multiclass classification) 문제
- 각 데이터 포인트 → 하나의 범주
    - 단일 레이블 다중 분류(single-label, multiclass classification) 문제
- 각 데이터 포인트 → 여러 개의 범주(토픽)
    - 다중 레이블 다중 분류 (multi-label, multiclass classification) 문제

```
# 로이터 데이터셋 로드하기
from keras.datasets import reuters

(train_data, train_labels), (test_data, test_labels) = reuters.load_data(num_words= 10
000)) # num_words , 데이터에서 자주 등장하는 단어 1만개로 제한

len(train_data)
len(test_data)
train_data[10] # 각 샘플은 정수 리스트(단어 인덱스)
```

```
# 로이터 데이터셋을 텍스트로 디코딩하기
word_index = reuters.get_word_index()
reverse_word_index = dict([(value,key) for (key,value) in word_index.itmes())])
decoded_newswird = ' '.join([reverse_word_index.get(i-3, '?') for i in train_data[0]])
# 0,1,2 는 패딩, 문서 시작, 사전에 없음을 위한 인덱스로 3을 제외함

train_labels[10]
```

## 3.5.2 데이터 준비

```
# 데이터 인코딩하기
import numpy as np

def vectorize_sequences(sequences, dimension = 10000):
  results = np.zeros(len(sequences), dimension))
  for i,sequence in enumerate(sequences):
    results[i, sequence] = 1.
  return results

x_train = vectorize_sequences(train_data) # 훈련 데이터 벡터 변환
x_test = vectorize_sequences(test_data) # 테스트 데이터 벡터 변환
```

```
# 레이블 변환
  # 1. 원핫 인코딩, 범주형 인코딩(categorical encoding)

def to_one_hot(labels, dimension = 46):
  results= np.zeros(len(labels), dimension))
  for i, label in enumerate(labels):
    results[i, label] = 1.
  return results

one_hot_train_labels = to_one_hot(train_labels) # 훈련 레이블 벡터 변환
one_hot_test_labels = to_one_hot(test_labels) # 테스트 레이블 벡터 변환

# 케라스 내장 인코딩 함수
from keras.np.utils import to_categorical
one_hot_train_labels = to_categorical(train_labels)
one_hot_test_labels = to_categorical(test_labels)
```

## 3.5.3 모델 구성

- 클래스(46)개 보다 적은 유닛(차원)을 가진 은닉층을 사용할 경우,
  - 정보의 손실 / 정보의 병목(bottleneck) 가능
  - 따라서, 클래스의 개수보다 작은 은닉층 X
- 다중 분류 /마지막 층 Dense
  - Dense : 46/ 각 원소는 다른 출력 클래스 인코딩
  - activation : softmax
    - 46개 출력 클래스에 대한 확률 분포를 출력
    - output[i] 는 어떤 샘플이 클래스 i에 속할 활률 : 46개 모두 합: 1
  - 손실 함수: categorical_crossentropy

- 두 확률 분포 사이의 거리 (진짜 레이블 분포 사이의 거리)

```
# 모델 정의하기
from keras import models
from keras import layers

model = models.Sequential()
model.add(layers.Dense(64, activation ='relu', input = (10000,))
model.add(layers.Dense(64, activation ='relu'))
model.add(layers.Dense(46, activation ='softmax'))

# 모델 컴파일
model.compile(optimizer= 'rmsprop',
              loss ='categorical_crossentropy',
              metrics= ['accuracy'])
```

## 3.5.4 훈련 검증

- 훈련 데이터 1,000개의 샘플 → 검증 샘플로 이용

```
# 검증 세트 준비하기
x_val = x_train[:1000]
partial_x_train = x_train[1000:]

y_val = one_hot_train_labels[:1000]
partial_y_train = one_hot_train_labels[1000:]

# 모델 훈련
history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs = 20,
                    batch_size = 512,
                    validation_data = (x_val, y_val))
```

```
# 훈련과 검증 손실 그리기
import matplotlib.pyplot as plt

loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(loss) + 1)

plt.plot(epochs, loss, 'bo', label = 'Training loss')
plt.plot(epochs, val_loss, 'b', label = 'Validation loss')
plt.title('Training and validation loss')
plt.xlable('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()
```

```
# 훈련과 검증 정확도 그리기
import matplotlib.pyplot as plt

acc = history.history['acc']
val_acc = history.history['val_acc']

epochs = range(1, len(loss) + 1)

plt.plot(epochs, acc, 'bo', label = 'Training accuracy')
plt.plot(epochs, val_acc, 'b', label = 'Validation accuracy')
plt.title('Training and validation accuracy')
plt.xlable('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```

```
# 모델을 처음부터 다시 훈련하기
model = models.Sequential()
model.add(layers.Dense(64, activation = 'relu', input_shape = (10000,)))
model.add(layers.Dense(64, activation ='relu'))
model.add(layers.Dense(46, activation ='softmax'))

model.compile(optimizer= 'rmsprop',
              loss = 'categorical_crossentropy',
              metrics = ['accuracy'])

model.fit(partial_x_train,
          partial_y_train,
          epochs = 9,
          batch_size= 512,
          validation_data = (x_val, y_val))

results= model.evaluate(x_test, one_hot_test_labels)
results
```

- 균형잡힌 데이터에서 이진 분류 하면 50% 정확도
- 불균형 데이터에서 무작위 분류하면 18% 정도

```
import copy
test_labels_copy = copy.copy(test_labels)
np.random.shuffle(test_labels_copy)
hits_array = np.array(test_labels) == np.array(test_labels_copy)
float(np.sum(hits_array)) / len(test_labels)
```

## 3.5.5 새로운 데이터에 대해 예측하기

- 모델 객체의 predict 메서드는 46개 토픽에 대한 확률 분포를 반환

```
# 새로운 데이터에 대해서 예측하기
predictions = model.predict(x_test)
predictions[0].shape # (46,) # 46개 len를 가진 벡터
np.sum(predictions[0]) # 원소들의 합 1
np.argmax(predictions[0]) # 확률이 제일 높은 클래스
```

## 3.5.6 레이블과 손실을 다루는 다른 방법

- 레이블 인코딩 - 정수 텐서로 변환

```
y_train = np.array(train_labels)
y_test = np.array(test_labels)

# 손실 함수
  # categorical_crossentropy - 레이블이 범주형 인코딩 해야 함
  # 정수 레이블 사용시 - sparse_categorical_crossentropy

model.compile(optimizer= 'rmsprop',
              loss = 'sparse_categorical_crossentropy',
              metrics = ['acc'])
```

## 3.5.7 충분히 큰 중간층을 두어야 하는 이유

- 마지막 출력 : 46 / 중간층이 → 히든 유닛이 46개보다 많이 적어서는 안됨

```
# 정보 병목이 있는 모델
model = models.Sequential()
model.add(layers.Dense(64, activation = 'relu', input_shape = (10000, )))
model.add(layers.Dense(4, activation ='relu'))
model.add(layers.Dense(46, activation = 'softmax'))

model.compile(optimizer= 'rmsprop',
              loss = 'categorical_crossentropy',
              metrics= ['accuracy'])

model.fit(partial_x_train,
          partial_y_train,
          epochs = 20,
          batch_size = 128,
          validation_data = (x_val, y_val))
```

## 3.5.8 추가 실험

- 더 크거나 작은 층 사용 / 32개의 유닛, 128개의 유닛 등

- 여기에서 2개의 은닉 층 → 1개/3개의 은닉층 사용

### 3.5.9 정리

- N개의 클래스로 데이터 포인트를 분류하려면 네트워크 → 마지막 Dense 층의 크기는 N
- 단일 레이블, 다중 분류 문제에서 N개의 클래스에 대한 확률 분포 출력 → softmax 활성함수
- 범주형 crossentropy
  - 모델이 출력한 확률 분포와 타깃 분포 사이의 거리를 최소화
- 다중 분류 레이블
  - 레이블을 범주형 인코딩(또는 원-핫 인코딩)
    - categorical_crossentropy 손실 함수 사용
  - 레이블을 정수로 인코딩
    - sparse_categorical_crossentropy 손실 함수 사용
- 중간층의 크기가 너무 작아 네트워크에 정보의 병목이 생기지 않도록 함

## 3.6 주택 가격 예측 : 회귀 문제

### 3.6.1 보스턴 주택 가격 데이터셋

```
# 보스턴 주택 데이터셋 로드하기
  # 범죄율, 지방 세율 등 -> 주택 가격의 중간 값 에측
  # 506 - 404 훈련/ 102 테스트 샘플
  # 특성 스케일 각자 다름
     # 0~1 / 1~12 / 1~100 사이의 값 등
from keras.datasets import boston_housing

(train_data, train_targets), (test_data, test_targets) = boston_hosing.load_data()
train_data.sheape # (404, 13)
test_data.shape # (102,13)
train_targets # 달러(집가격)
```

### 3.6.2 데이터 준비

- 상이한 스케일 → 특성별로 정규화하는 것이 좋음
- (값 - 평균) / 표준편차 → 특성의 중앙이 0 근처에 맞춰짐, 표준 편차: 1

```
# 데이터 정규화하기
mean = train_data.mean(axis = 0)
train_data -= mean
std = train_data.std(axis = 0)
train_data /= std

test_data -= mean
test_data /= std
```

## 3.6.3 모델 구성

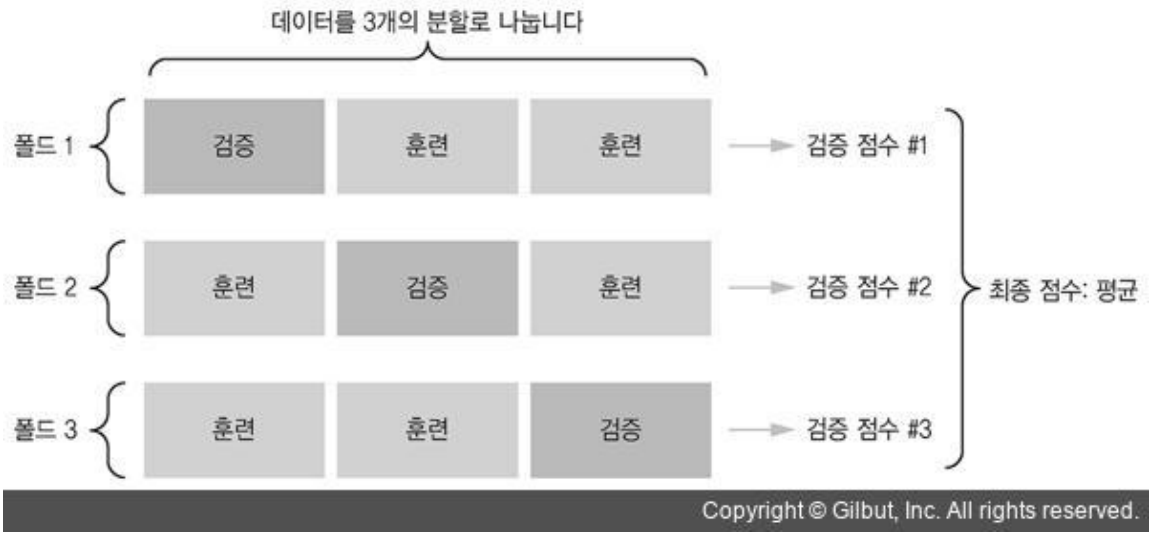
```
# 모델 정의하기
from keras import models
from keras import layers

def build_model():
  # 동일한 모델을 여러 번 생성할 것이므로 함수를 만들어 사용
  model = models.Sequential()
  model.add(layers.Dense(64, activation ='relu',
                         input_shape = (train_data.shape[1],)))
  model.add(layers.Dense(64, activation ='relu'))
  model.add(layers.Dense(1)) # 유닛, 활성함수 X, 선형 층

  # 모델 컴파일
    # mse 손실 함수 사용하여 컴파일
      # 평균 제곱 오차(mean squared error) - 에측과 타깃 사이 거리의 제곱
    # mae 모니터링 - 새로운 지표 평균 절대 오차
      # 평균 절대 오차(mean absolute error, mae) - 에측 타깃 사이 거리의 절댓값
  model.compile(optimizer= 'rmsprop', loss = 'mse', metrics= ['mae'])
  return model
```

## 3.6.4 K-겹 검증을 사용한 훈련 검증

- K겹 교차 검증(K-fold cross-validation)
  - 데이터를 K개의 분할(폴드, fold)로 나누고
  - K개의 모델을 각각 만들어 K-1개의 분할에서 훈련, 나머지 분할에서 평가
  - 모델의 검증 점수는 K개의 검증 점수 평균이 됨




```
# K 겹 검증하기
import numpy as np

k = 4
num_val_sample = len(train_data) // k
num_epochs = 100
all_score = []

for i in range(k):
  print('처리중인 폴드#', i)
  # 검증 데이터 준비: k번째 분할
  val_data = train_data[i * num_val_samples : (i+1) * num_val_samples]
  val_targets = train_targets[i * num_val_samples : (i+1) * num_val_samples]

  partial_train_data = np.concatenate( # 훈련 데이터 준비 : 다른 분할 전체
    [train_data[:i * num_val_samples],
    train_data[(i+1) * num_val_samples:]],
    axis = 0)

  partial_train_targets= np.concatenate(
    [train_targets[:i * num_val_samples],
    train_targets[(i+1) * num_val_samples:]],
    axis = 0)

model = build_model() # 케라스 모델 구성(컴파일 포함)
model.fit(partial_train_data,
          partial_train_targets,
          epochs = num_epochs,
          batch_size = 1,
          verbose = 0)
          # 모델 훈련 verbose = 0 , 훈련 과정이 출력되지 않음

# 검증 데이터로 모델 평가
val_mse, val_mae = model.evaluate(val_data, val_targets, verbose= 0)
all_scores.append(val_mae)

# num_epochs = 100 으로 실행한 결과
```

```
all_scores
np.mean(all_scores)
```

```
# 각 폴드에서 검증 점수를 로그에 저장하기
num_epochs = 500
all_mae_histories = []

for i in range(k):
  print('처리중인 폴드 #', i)
  val_data = train_data[i * num_val_samples , (i+1) * num_val_samples]
  val_targets = train_targets[i * num_val_samples , (i+1) * num_val_samples]

  partial_train_data = np.concatenate( # 훈련 데이터 준비: 다른 분할 전체
    [train_data[: i * num_val_samples],
     train_data[(i+1) * num_val_samples:]],
      axis = 0)

  partial_train_targets = np.concatenate( # 훈련 데이터 준비: 다른 분할 전체
    [train_targets[: i * num_val_samples],
     train_targets[(i+1) * num_val_samples:]],
      axis = 0)

  model = build_model() # 케라스 모델 구성 (컴파일 포함)
  history = model.fit(partial_train_data,
                      partial_train_targets,
                      validation_data = (val_data, val_targets),
                      epochs = num_epochs,
                      batch_size= 1,
                      verbose=  0)

  mae_history = history.history['val_mean_absolute_error']
  all_mae_histories.append(mae_history)
```

```
# K 겹 검증 점수 평균을 기록하기
average_mae_history = [
    np.mean([x[i] for x in all_mae_histories]) for i in range(num_epochs)]
```

```
# 검증 점수 그래프
import matplotlib.pyplot as plt

plt.plot(range(1, len(average_mae_history) + 1), average_mae_history)
plt.xlabel('Epochs')
plt.ylabel('Validation MAE')
plt.show()
```

- 첫 10개 데이터 포인트가 너무 커서 결과가 잘 안보임 → 제외
- 부드러운 곡선 → 이전 포인트의 지수 이동 평균(exponential moving average)로 대체

```
# 처음 10개의 데이터 포인트를 제외한 검증 점수 그리기
def smooth_curve(points, factor = 0.9):
  smoothed_points= []
  for point in points:
    if smoothed_points:
      previous = smoothed_points[-1]
      smoothed_points.append(previous * factor + point * (1-factor))
    else:
      smoothed_points.append(point)
  return smoothed_points

smoothe_mae_history = smooth_curve(average_mae_history[10:])

plt.plot(range(1, len(smooth_mae_history) + 1), smooth_mae_history)
plt.xlabel('Epochs')
plt.ylabel('Validation MAE')
plt.show()
```

```
# 최종 모델 훈련하기
model = build_model() # 새롭게 컴파일된 모델을 얻음
model.fit(train_data,
          train_targets, # 전체 데이터로 훈련시킴
          epochs = 80,
          batch_size= 16,
          verbose = 0)

test_mse_score, test_mae_score= model.evaluate(test_data, test_targets)

# 최종 결과
test_mae_score
```

### 3.6.5 정리

- 회귀
  - 손실 함수 : 평균 제곱 오차(MSE)
  - 평가 지표: 평균 절대 오차(MAE)
- 스케일 조정
  - 입력 데이터의 특성이 서로 다른 범위 가지면 → 스케일 조정
- 가용할 데이터가 적다면
  - K-겹 검증을 사용하는 것이 신뢰할 수 있는 모델 평가 방법
  - 과대적합 피하기 위해 은닉층의 수를 줄인 모델이 좋음 (1~2개)

## 3.7 요약

- 머신 러닝 - 이진 분류, 다중 분류, 스칼라 회귀
  - 원본 데이터를 신경망에 주입하기 전에 전처리
  - 데이터 범위 다른 특성 → 각 특성 독립적으로 스케일 조정
  - 훈련이 진행되며, 신경망 과대적합 , 새로운 데이터에 대해 나쁜 결과
  - 훈련 데이터 적을때, 과대적합 피하기 위해 → 1~2개 은닉층
  - 데이터 많은 범주로 나뉘어 있을때 → 중간층 작으면 → 정보의 병목 발생
  - 회귀 - 다른 손실 함수 / 평가 지표
  - 적은 데이터 - K겹 검증 → 모델 평가