4장 머신러닝의 기본 요소

케라 수요일 3시

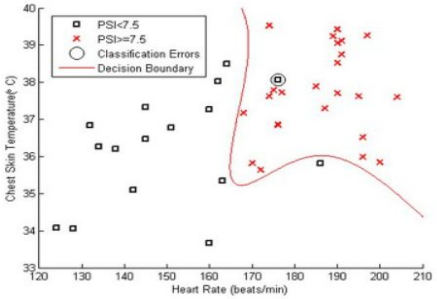
머신 러닝의 기본 요소

1. 머신 러닝의 네 가지 분류 -가영님
2. 머신 러닝 모델 평가 - 우혁님
3. 데이터 전처리, 특성 공학, 특성 학습 - 민지님
4. 과대적합과 과소적합 - 성곤님
5. 보편적인 머신 러닝 작업 흐름 - 주희님

4.1 머신 러닝의 네 가지 분류

4.1.1 지도학습

- 샘플 데이터 주어지면 타깃에 입력 데이터를 매핑하는 방법을 학습
- [data, label] 형태로 학습시킴
- 가장 흔함
- 대부분 분류와 회귀로 구성됨

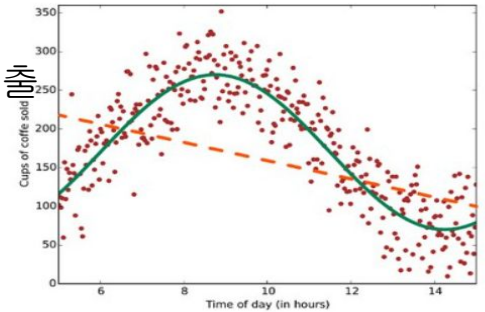


분류(classification)

범주형 데이터에서 주어진 입력 벡터가 어떤 종류의 값인지 구분
예측하는 결과값이 이산값

회귀(regression)

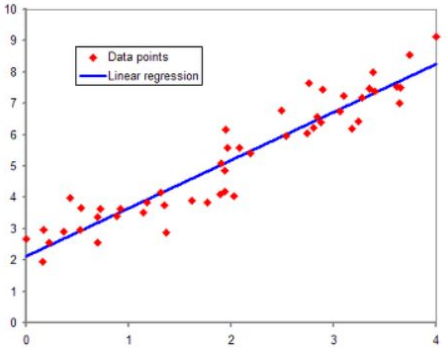
연속형 데이터에서 입력과 출력값 사이의 일반적인 관계 특성을 도출
(연속적인 값 예측)
예측하는 결과값이 연속값



지도학습 알고리즘

- 선형 회귀

  임의로 분포한 데이터들을 하나의 직선으로 일반화



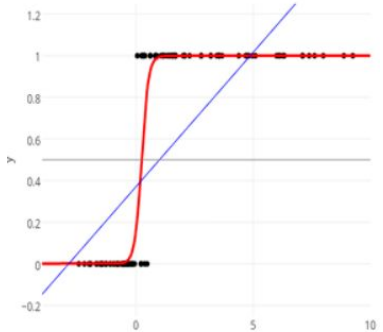
- 로지스틱 회귀

  출력값이 범주형 데이터일 때 사용할 수 있음

  로지스틱 함수

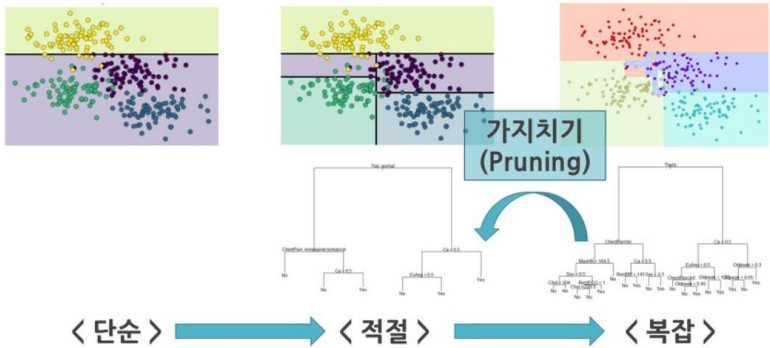
  최솟값, 최댓값이 특정 값으로 수렴하고 그 사이는 S자로 굴곡이 진 함수

  범주형으로 완전 분리된 값을 구분하기 좋음



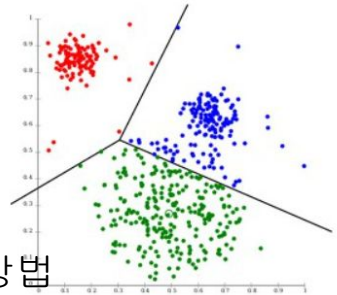
- 의사결정나무

  주어진 입력값에 대해 여러 번의 질문을 통해 답을 찾음



4.1.2 비지도학습

- 데이터에 대한 타깃이 주어지지 않은 상태에서 학습시킴
- 정답 라벨이 없는 데이터를 비슷한 특징끼리 군집화해 새로운 데이터에 대한 결과를 예측함
- 지도 학습의 전처리 방법으로도 사용함
- [data]의 형태로 학습함



비지도학습 알고리즘

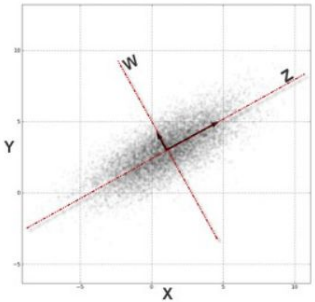
- 군집 분석

  입력된 데이터의 값에 따라 비슷한 것들끼리 군집으로 묶어주는 분석 방법

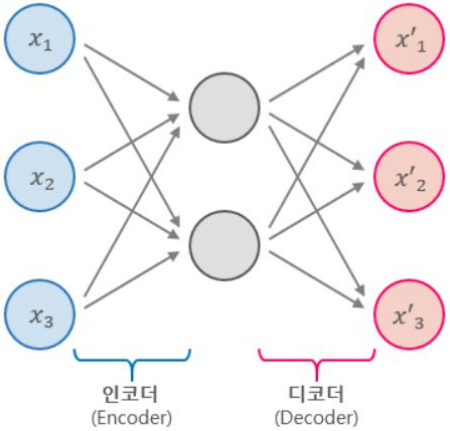
- 주성분 분석

  데이터 입력값의 특성을 분석해서 출력값을 더 잘 설명해줄 수 있는 새로운 입력 특성을 추출

  입력 특성의 개수를 적절하게 줄이는 효과가 있음
  → 데이터의 크기를 축소해 지도 학습을 시킬 때 성능 개선에 활용됨



4.1.3 자기 지도 학습

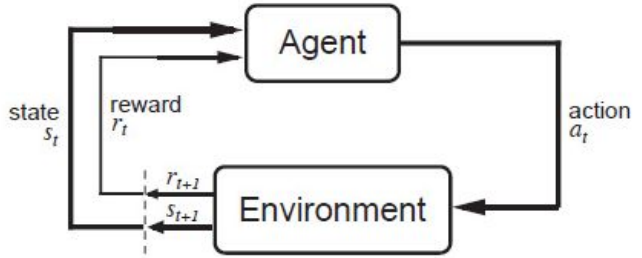


- 지도 학습의 특별한 경우
- 사람이 만든 레이블을 사용하지 않음
- 레이블이 없는 데이터를 이용해서 지도 학습의 방식으로 학습
- 경험적인 알고리즘을 사용해 입력 데이터로부터 레이블을 생성함

4.1.4 강화 학습

- 행동 심리학에서 나온 이론
- 분류할 수 있는 데이터 존재 X, 데이터 있어도 정답이 X, 자신이 한 행동의 보상을 받으며 학습
- 에이전트가 환경에 대한 정보를 받아 보상을 최대화하는 행동 선택하도록 학습됨

강화 학습에 신경망 적용하면서 복잡한 문제(자율 주행 자동차, 바둑)에 적용 가능하게 됨
(신경망으로 근삿값을 구해서)



4.2 머신 러닝 모델 평가

4.21 훈련,검증,테스트 세트

- 단순 홀드아웃 검증

- K-겹 교차 검증

- 셔플링을 사용한 반복 K-겹 교차 검증

4.22 기억해야 할 것

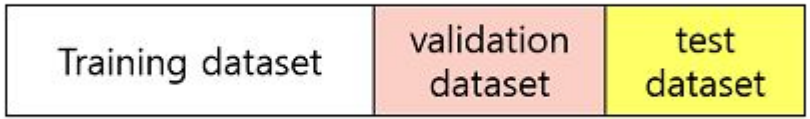
4.21 훈련,검증, 테스트 세트

딥러닝 모델을 구축할 때, 훈련 데이터와 테스트 데이터만으로도 훈련의 척도를 판단할 수 있다. 하지만, 훈련 데이터에 대한 학습만을 바탕으로 모델의 설정(Hyperparameter)를 튜닝하게 되면 과대적합(overfitting)이 일어날 가능성이 매우 크다. 또한, 테스트 데이터는 학습에서 모델에 간접적으로라도 영향을 미치면 안 되기 때문에 테스트 데이터로 검증을 해서는 안 된다. 그래서 검증(validation) 데이터셋을 따로 두어 매 훈련마다 검증 데이터셋에 대해 평가하여 모델을 튜닝해야 한다.

하지만, 검증 데이터셋이 훈련에 사용되지 않더라도 검증 데이터셋에 대한 성능을 기반으로 hyperparameter를 튜닝하므로 **정보 누설 (information leak)**이 일어나게 된다. 즉, 검증 데이터셋을 사용함으로써 훈련 데이터셋에 대한 overfitting은 어느정도 피할 수 있어도 오히려 검증 데이터셋에 대해 overfitting될 수 있다는 것이다.

이를 위해 여러 가지 검증 방법이 존재한다.

단순 홀드 아웃 검증



아주 기본적인 검증 방법으로 단순히 트레이닝 데이터와 테스트 데이터로 나누고, 나눠진 트레이닝 데이터에서 다시 검증 데이터셋을 따로 떼어내는 방법이다.

단순 홀드 아웃 검증 - 코드

```
(train_data, train_targets), (test_data, test_targets) = boston_housing.load_data()

num_train_samples = 323 # 404*0.8

validation_data = train_data[num_train_samples:]
validation_targets = train_targets[num_train_samples:]

train_data = train_data[:num_train_samples]
train_targets = train_targets[:num_train_samples]

model = build_model()
model.fit(train_data, train_targets,
          epochs=100, batch_size=16, verbose=0)
validation_score = model.evaluate(validation_data, validation_targets)

model = build_model()
model.fit(np.concatenate([train_data,validation_data]), np.concatenate([train_targets,validation_targets]),
          epochs=80, batch_size=16, verbose=0)
validation_score = model.evaluate(test_data, test_targets)
```

```
3/3 [==============================] - 0s 2ms/step - loss: 51.8100 - mean_absolute_error: 5.3235
4/4 [==============================] - 0s 2ms/step - loss: 33.4284 - mean_absolute_error: 4.4368
```

```
from sklearn.model_selection import train_test_split

(train_data, train_targets), (test_data, test_targets) = boston_housing.load_data()

# train_test_split
train_data, validation_data,train_targets, validation_targets = train_test_split(train_data, train_targets, test_size=0.2, shuffle=True, random_state=100)
print(train_data.shape)
```

```
(323, 13)
```

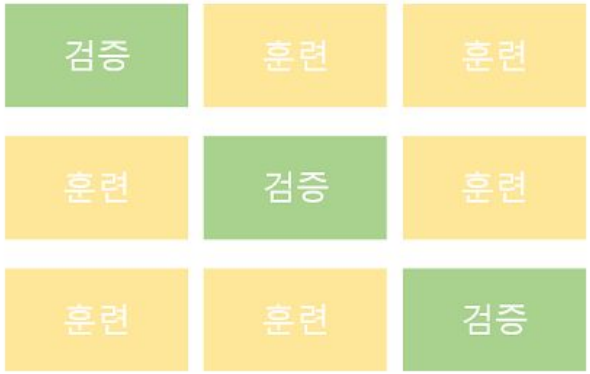
단순 홀드 아웃 검증 - 문제점

검증 세트와 테스트 세트의 샘플이 너무 적어 주어진 데이터를 통계적으로 대표하지 못함. 다른 난수 초깃값으로 셔플링해서 데이터를 나누었을 때

모델의 성능이 다름.

```
3/3 [==============================] - 0s 3ms/step - loss: 66.0687 - mean_absolute_error: 6.2421
4/4 [==============================] - 0s 2ms/step - loss: 40.1109 - mean_absolute_error: 4.4577
3/3 [==============================] - 0s 2ms/step - loss: 47.3933 - mean_absolute_error: 4.6738
4/4 [==============================] - 0s 2ms/step - loss: 80.7496 - mean_absolute_error: 7.2992
```

K-겹 교차 검증



위 그림처럼 정해진 K번만큼 검증 데이터셋과 훈련 데이터셋을 변경해가면서 거의 모든 데이터에 대해 검증을 하는 방법이다.
모델의 총 validation score는 각 훈련의 validation score의 평균이다.

K-겹 교차 검증- 코드

```python
import numpy as np

(train_data, train_targets), (test_data, test_targets) = boston_housing.load_data()

k = 4
num_val_samples = len(train_data) // k
num_epochs = 80
all_scores = []
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]

    # 훈련 데이터 준비: 다른 분할 전체
    partial_train_data = np.concatenate(
        [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)

    # 케라스 모델 구성(컴파일 포함)
    model = build_model()
    # 모델 훈련(verbose=0 이므로 훈련 과정이 출력되지 않습니다)
    model.fit(partial_train_data, partial_train_targets,
              epochs=num_epochs, batch_size=1, verbose=0, shuffle = True)
    # 검증 세트로 모델 평가
    val_mse, val_mae = model.evaluate(val_data, val_targets, verbose=0)
    all_scores.append(val_mae)

print(np.mean(all_scores))

model = build_model()
model.fit(train_data,train_targets,
          epochs=80, batch_size=16, verbose=0,shuffle=False)
validation_score = model.evaluate(test_data, test_targets)
```

```
처리중인 폴드 # 0
처리중인 폴드 # 1
처리중인 폴드 # 2
처리중인 폴드 # 3
3.179085373878479
4/4 [==============================] - 0s 2ms/step - loss: 29.3609 - mae: 3.9628
```

셔플링을 사용한 반복 K-겹 교차 검증- 코드

```
import numpy as np
import random
(train_data, train_targets), (test_data, test_targets) = boston_housing.load_data(seed = random.randint(1,100))


k = 4
num_val_samples = len(train_data) // k
num_epochs = 80
all_scores = []
for j in range(3):
  (train_data, train_targets), (test_data, test_targets) = boston_housing.load_data(seed = random.randint(1,100))
  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]

    # 훈련 데이터 준비: 다른 분할 전체
    partial_train_data = np.concatenate(
        [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)

    # 케라스 모델 구성(컴파일 포함)
    model = build_model()
    # 모델 훈련(verbose=0 이므로 훈련 과정이 출력되지 않습니다)
    model.fit(partial_train_data, partial_train_targets,
              epochs=num_epochs, batch_size=1, verbose=0, shuffle = False)
    # 검증 세트로 모델 평가
    val_mse, val_mae = model.evaluate(val_data, val_targets, verbose=0)
    all_scores.append(val_mae)
```

```
3.4889694849650064
4/4 [==============================] - 0s 2ms/step - loss: 55.4296 - mae: 5.3
```

4.22 기억해야 할 것 (주의 해야 할 점)

- 대표성 있는 데이터 :예를 들어 0~9의 이미지가 순서대로 있는 데이터를 훈련 시킬 때 0~7은 훈련, 8~9는 테스트로 사용할 수 있기 때문에 데이터를 나누기전에 섞어줌으로써 대표성을 가지게 해야 된다.

- 시간의 방향 : 과거로부터 미래를 예측해야 하기 때문에 데이터를 섞어서는 안됩니다. 훈련 세트에 있는 데이터는 과거, 테스트 세트에 있는 데이터는 미래여야 한다

- 데이터 중복 : 한 데이터가 두번 등장하면 신뢰도가 크게 떨어진다. 따라서 훈련 세트와 검증 세트가 중복되지 않는지 확인해야 한다.

TMI: 데이터 Set의 적절한 비율

전통적인 머신러닝 방법론에 따르면 Train : Val: Test를 6:2:2정도의 비율로 나눈다고 한다.

이는 10여 년 전만 해도 확보 가능한 데이터의 수가 적기 때문에 위의 비율이 적절하다고 한다.

하지만 데이터의 수가 엄청 많을 때는 (ex:100만개)

오히려 98:1:1이 훨씬 효율이 좋다고 한다.

https://www.youtube.com/watch?v=_Fe5kKmFieg&index=6&list=PLkDaE6sCZn6E7jZ9sN_xHwSHOdjUxUW_b

(앤드류 응 교수 딥러닝 강의)

4.3.1 신경망을 위한 데이터 전처리

데이터 전처리 목적 = 원본 데이터를 신경망에 적용하기 쉽도록

1. 벡터화
2. 값 정규화
3. 누락된 값 다루기
4. 특성추출

데이터 벡터화(data vectorization)

사운드, 이미지, 텍스트등 신경망에서 모든 입력과 타깃을 텐서로 변환하는 것

3.5.2 데이터 준비

레이블을 벡터로 바꾸는 방법

1. 레이블의 리스트를 정수 텐서로 변환
2. 원-핫 인코딩 사용

```
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
```

원-핫 인코딩

```
from keras.utils.np_utils import to_categorical

one_hot_train_labels = to_categorical(train_labels)
one_hot_test_labels = to_categorical(test_labels)
```

값 정규화(data vectorization)

숫자 이미지 분류 :

```
[ ]  1 train_images = train_images.reshape((60000, 28*28))
     2 train_images = train_images.astype('float32') / 255
     3
     4 test_images = test_images.reshape((10000, 28*28))
     5 test_images = test_images.astype('float32') / 255
```

주택 가격 예측 :

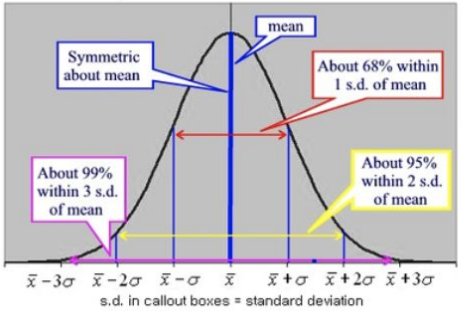
3.6.2 데이터 준비

상이한 스케일을 가진 다양한 데이터의 경우 각 특성별로 정규화를 한 후 신경망에 주입한다.

- 입력 데이터 각 특성의 평균을 빼고 표준 편차로 나누어준다.
- 특성의 중앙이 0근처에 맞추어지고
- 표준편차가 1이 된다.

```
1 # 데이터 정규화하기
2 mean = train_data.mean(axis = 0)
3 train_data -= mean
4
5 std = train_data.std(axis = 0)
6 train_data /= std
7
8 test_data -= mean
9 test_data /= std
```

$$Z = \frac{X - \mu}{\sigma}$$ → "평균값에서 표준편차의 몇배 정도 떨어져 있다"는 것을 평가하는 수치



표준편차 ±1배의 범위내에 약 68% 데이터가 들어감.
표준편차 ±2배의 범위내에 약 95% 데이터가 들어감.
표준편차 ±3배의 범위내에 약 99% 데이터가 들어감.

네트워크를 쉽게 학습시키기 위한 데이터 특징

1) 0 ~ 1의 작은 값
2) 모든 특성이 대체로 비슷한 범위를 갖도록 균일성
3) 평균이 0이 되도록 정규화
4) 표준 편차가 1이 되도록 정규화

누락된 값 다루기

누락 데이터에 특정 값을 지정

1) 0으로 채우기
2) 평균값으로 채우기
3) 중간값으로 채우기

4.4 과대적합과 과소적합

- 최적화: 훈련 데이터에 대한 최고의 성능

- 일반화: 처음 보는 데이터에 대한 최고의 성능

최적화와 일반화 사이의 줄다리기(trade off)

과대적합과 과소적합

- 과대적합: 훈련 데이터에 특화된 패턴을 학습
- 과소적합: 훈련 데이터에 있는 관련 특성을 모두 학습하지 못함

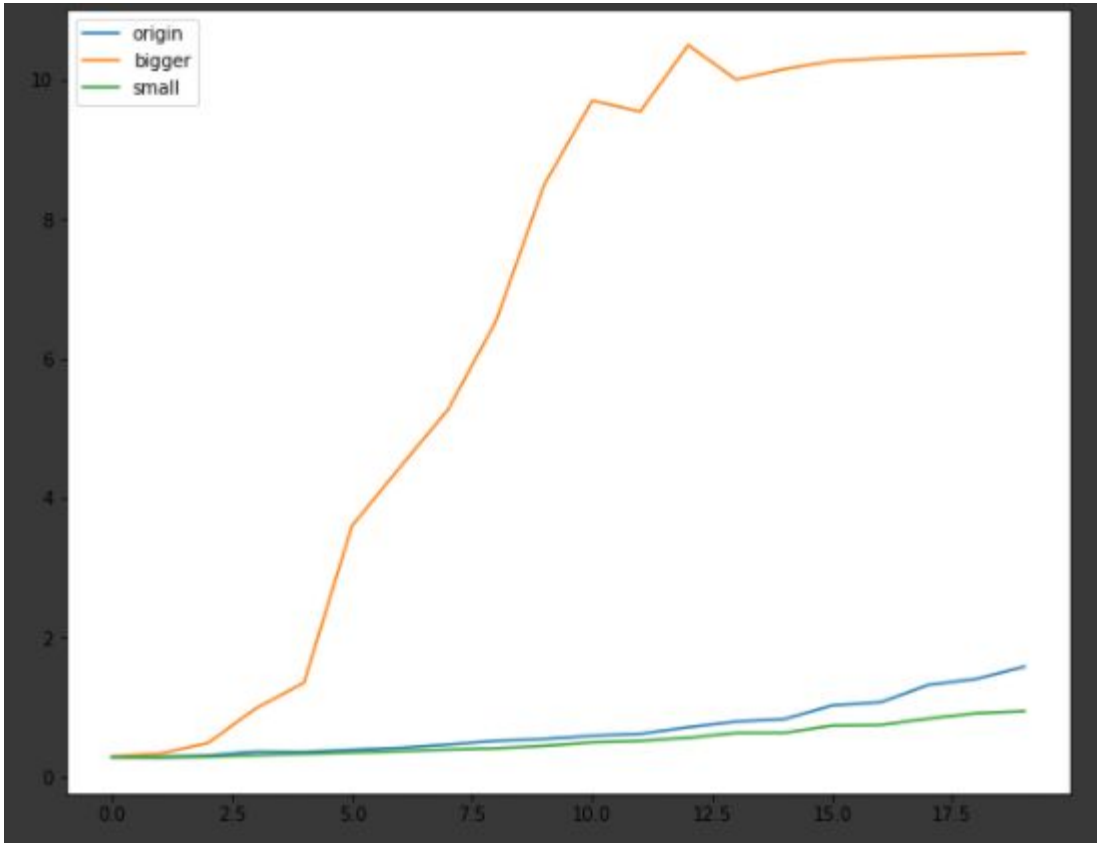
해결방법

1. 더 많은 데이터 수집
2. 규제

규제 1. 네트워크 크기 축소

- 모델의 크기를 줄인다 -> 기억 용량을 줄인다
  - 층의 수
  - 유닛의 수

- 비교적 적은 수의 층과 파라미터로 시작해서 검증 손실이 감소되기 시작할 때까지 조금씩 늘려간다.



용량이 크고 복잡한 모델은 순식간에 과대적합

기존 모델

단순해진 모델, 기존 모델보다 과대적합으로부터 안전

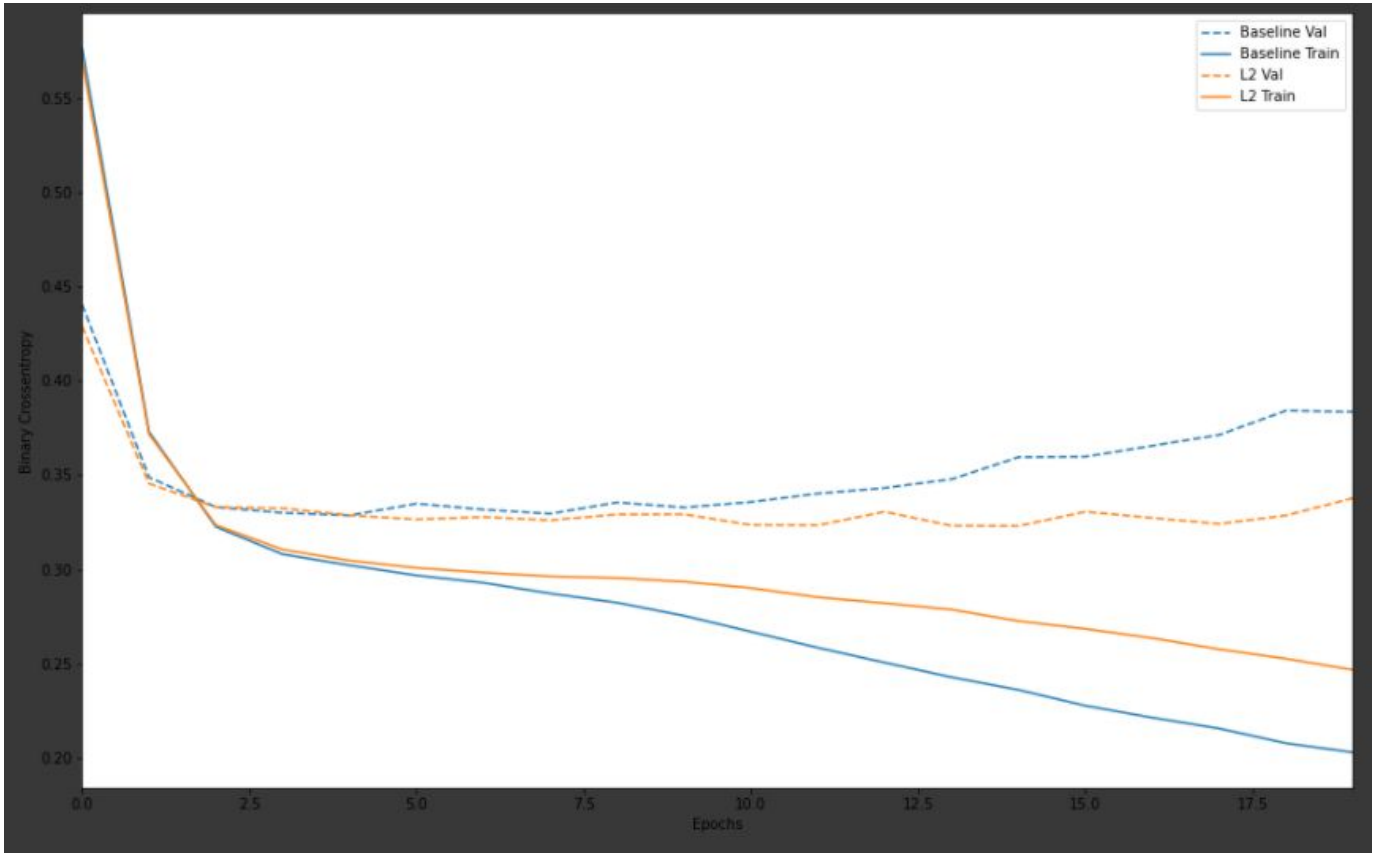
규제2. 가중치 규제 추가

- L1 규제: 가중치의 **절댓값**에 비례하는 비용 추가
- L1 Norm 은 벡터 p, q 의 각 원소들의 차이의 절대값의 합입니다.
- p = (3, -5, 8) , q = (2, 1, 6) -> |3 -1| + |-5-1| + |8-6| = 2+6+2 = 10

- L2 규제: 가중치의 **제곱**에 비례하는 비용이 추가
- $\|\boldsymbol{x}\|_2 := \sqrt{x_1^2 + \cdots + x_n^2}.$
- p = (x_1, x_2, ... , x_n), q = (0, 0, ... , 0)

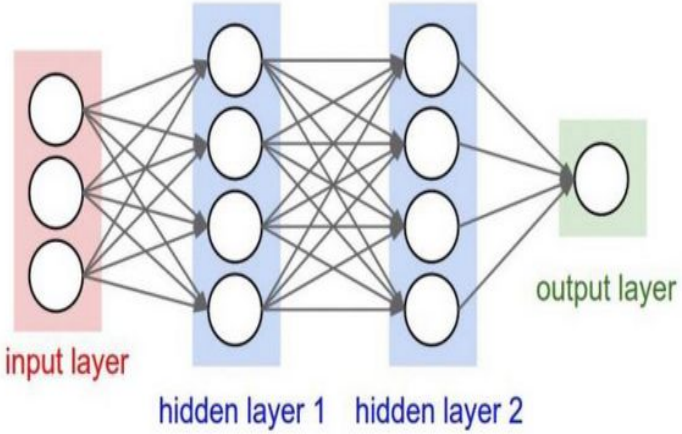
https://light-tree.tistory.com/125

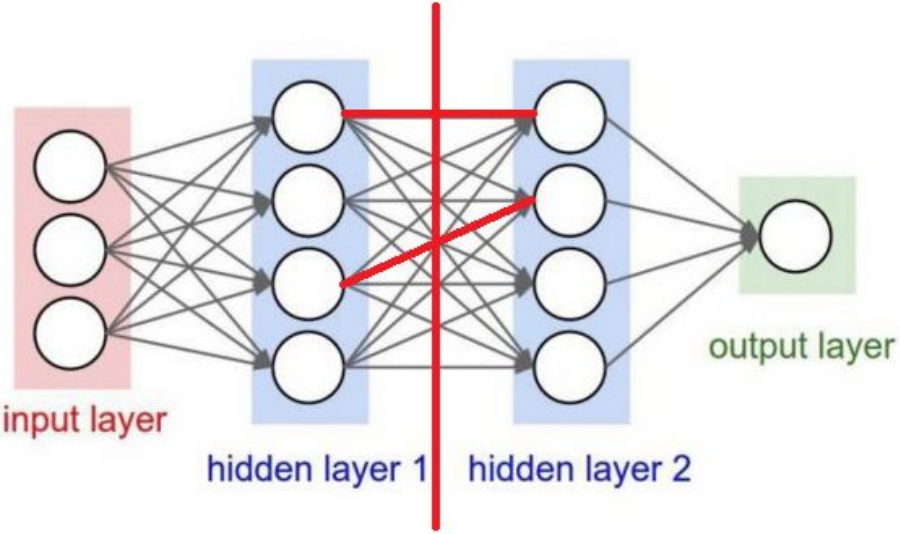
https://wordbe.tistory.com/entry/Regulariz
A%B7%9C%EC%A0%9C-%EB%AA%A8

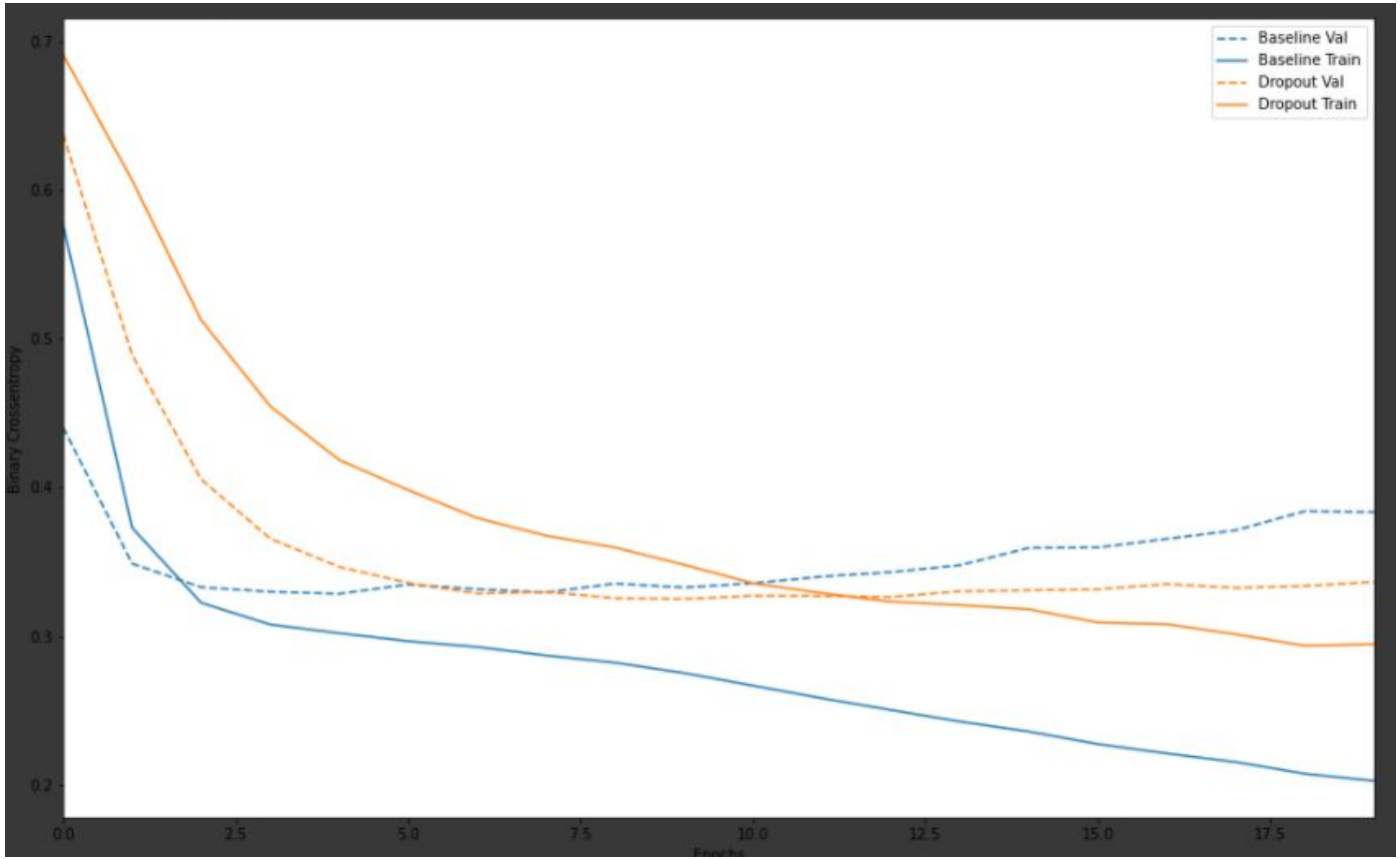


노랑(L2 규제)이가 과대적합으로 부터 잘 버티고 있다.

규제3: 드롭아웃 추가


input layer
hidden layer 1
hidden layer 2
output layer


input layer
hidden layer 1
hidden layer 2
output layer



과대적합도 적고, 성능도 좋아졌다.

정리 : 과대적합 방지를 위한 네가지 방법

1. 더 많은 데이터 수집
2. 네트워크 용량 감소 (모델 단순화, 경량화)
3. 가중치 규제 추가(L1, L2)
4. 드롭아웃 추가

4.5 보편적인 머신러닝 작업 흐름

1) 문제 정의와 데이터셋 수집
2) 성공지표 선택
3) 평가 방법 선택
4) 데이터 준비
5) 모델 훈련하기
6) 모델 구축
7) 모델 규제와 하이퍼파라미터 튜닝

4.5.1 문제 정의와 데이터셋 수집

1. 주어진 문제 정의

입력 데이터가 무엇인지

예측하고자 하는 것이 무엇인지

→ 이에 따라 가용한 훈련데이터가 있어야 예측하도록 학습할 수 있다.

ex) 영화리뷰와 감성 테이블이 태깅되어있어야 영화리뷰의 감성분류를 학습할 수 있음.

2. 당면한 문제의 종류가 무엇인지?

이진 분류, 다중 분류, 스칼라 회귀, 벡터 회귀, 다중 레이블 다중 분류, 군집, 생성또는 강화학습

→ 문제의 유형 식별은 모델의 구조와 손실 함수 선택에 도움이 된다.

→ 이후 해당 데이터를 통해 모델을 구현하고 검증 후에 문제 해결 여부를 판단할 수 있다.

4.5.2 성공 지표 선택

성공의 지표는 모델이 최적화할 손실 함수를 선택하는 기준

- 클래스 분포가 **균일한** 분류 문제: **정확도와 ROC AUC**
- 클래스 분포가 **균일하지 않은** 문제: **정밀도와 재현율**
- **랭킹** 문제나 **다중 레이블** 문제: **평균 정밀도**

		실제 정답	
		True	False
분류 결과	True	True Positive	False Positive
	False	False Negative	True Negative

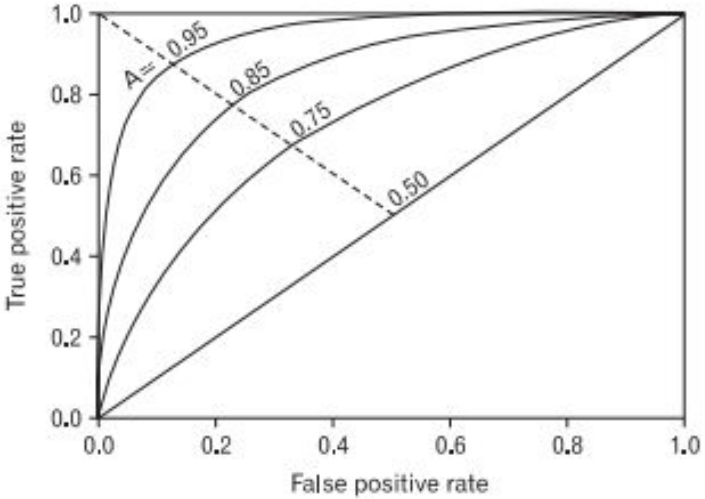
- True Positive(TP) : 실제 True인 정답을 True라고 예측
- False Positive(FP) : 실제 False인 정답을 True라고 예측
- False Negative(FN) : 실제 True인 정답을 False라고 예측
- True Negative(TN) : 실제 False인 정답을 False라고 예측

$$(Accuracy) = \frac{TP + TN}{TP + FN + FP + TN}$$

Precision(정밀도): 모델이 True라고 분류한 것 중에서 실제 True인 것의 비율
Recall(재현율): 실제 True인 것 중에서 모델이 True라고 예측한 것의 비율

$$(Precision) = \frac{TP}{TP + FP}$$

$$(Recall) = \frac{TP}{TP + FN}$$



ROC curve (Receiver Operating Characteristic curve) :
TPR과 FPR을 각각 x,y축으로 놓은 그래프
면적은 항상 0.5~1의 범위 (0.5이면 랜덤에 가까운 성능, 1이면 최고의 성능)

ROC AUC (ROC Area Under the Curve):
ROC curve의 밑면적을 구한 값. 값이 1에 가까울수록 성능이 좋다.

- **TPR** : True Positive Rate (=민감도):
1인 케이스에 대해 **1**로 잘 예측한 비율.**(**암환자를 진찰해서 암이라고 진단**)**

- **FPR** : False Positive Rate (**=1-**특이도, false accept rate)**:**
0인 케이스에 대해 **1**로 잘못 예측한 비율**. (**암환자가 아닌데 암이라고 진단**)**

True positive rates :
(= recall, sensitivity)
$$TPR = R = \frac{TP}{TP + FN}$$

True negative rates :
(= specificity)
$$TNR = \frac{TN}{TN + FP}$$

4.5.3 평가 방법 선택

단순 홀드아웃 검증: 데이터셋을 train과 validation으로 나누어 사용. 데이터가 풍부할 때 사용

k-겹 교차 검증: 단순 홀드아웃 검증을 사용하기에 샘플의 수가 너무 적을 때 사용.

(보통 딥러닝을 사용할 때는 데이터가 풍부한 상황이므로 홀드아웃 방법으로 대체 가능. 대규모 딥러닝 모델은 훈련 비용이 너무 커서 교차 검증을 적용하기 어려울 때가 많다.)

반복 k-겹 교차 검증: 데이터가 적고 매우 정확한 모델 평가가 필요할 때

4.5.4 데이터 준비

머신러닝 모델에 주입할 데이터를 구성.

- 데이터는 텐서로 구성
- 입력 데이터와 타깃 데이터의 텐서를 준비.
- 특성마다 범위가 다르면(여러 종류의 값으로 이루어진 데이터라면) 정규화해야 한다.

* 주어진 입력으로 출력을 예측할 수 있고, 가용한 데이터에 입력과 출력 사이의 관계를 학습하는 데에 충분한 정보가 있다고 가정

4.5.5 모델 훈련하기

마지막 층의 활성화 함수: 전달받은 값을 출력할 때 일정 기준에 따라 출력값을 변화시키는 비선형 함수.
네트워크의 출력에 필요한 제한을 가한다.

손실 함수

- 모델의 출력값과 사용자가 원하는 출력값의 차이, 즉 오차를 의미한다.
- 신경망이 학습할 수 있도록 해주는 지표.
- 손실 함수 값이 최소화하는 W, b를 찾아가는 것이 학습목표이다.
- 일반적으로 회귀문제는 평균 제곱 오차, 분류문제는 교차 엔트로피를 사용한다.

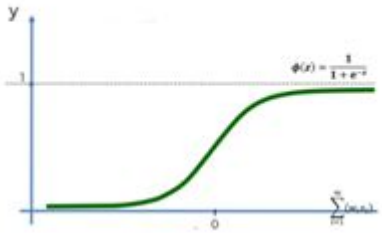
최적화 설정: 어떤 옵티마이저를 사용할 지. 학습률은 얼마인지.
최적화(Optimization) 과정을 통해 Loss Function에서 cost(loss)가 최소가 되는 부분을 찾는다.

모델에 맞는 마지막 층의 활성화 함수와 손실 함수 선택

문제유형	마지막 층의 활성화 함수	손실함수
이진 분류	시그모이드	Binary_crossentropy
단일 레이블 다중 분류	소프트맥스	Categorical_crossentropy
다중 레이블 다중 분류	시그모이드	Binary_ crossentropy
임의 값에 대한 회귀	없음	mse
0과 1 사이의 값에 대한 회귀	시그모이드	mse 또는 Binary_ crossentropy

마지막 층의 활성화 함수

시그모이드 (Sigmoid)



- 입력을 (0,1) 사이로 정규화(normalization)
- Backpropagation 단계에서 NN layer 를 거칠 때마다 작은 미분 값이 곱해져, 모델이 깊을수록 기울기가 사라지는 **Gradient Vanishing 문제** 를 야기한다. 여러 개의 Layer를 쌓으면 신경망 학습이 잘 되지 않는 원인
- 위와 같은 이유로 DNN layer에서 잘 사용하지 않는다.

소프트맥스(softmax)

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

- **다중 분류** 문제를 풀때 가장 자주 사용되는 활성화 함수
- 입력받은 값을 0~1 사이의 출력이 되도록 정규화하여 출력되는 layer의 output들의 합이 1이 되도록 출력하는 함수.

ex) MNIST예제

10개의 숫자를 분류하는 작업에서, 10개 layer의 output의 합이 1이 되도록 한다. 숫자를 분류 할때 0~9 중 하나일 확률은 1이며 그 중 실제 해당되는 숫자의 확률을 가장 높이 나오도록 학습하는 것.

4.5.6 모델 구축

이상적인 모델: 과소적합과 과대적합 사이의 경계에 적절히 위치한 모델

1 step 과대 적합 모델을 만들어 보기.

: layer 추가, layer 크기 키우기, epoch 늘리기.

훈련과 검증지표, train loss와 validation loss를 모니터링해서 검증 데이터에서 모델 성능이 감소하기 시작하면 과대적합에 도달한 것이다.

2 step 이상적인 모델로 만들기

규제와 모델 튜닝을 통해 과소적합도 아니고 과대적합도 아닌 이상적인 모델에 가깝도록 만든다.

4.5.7 모델 규제와 하이퍼파라미터 튜닝

1. 모델 수정 → 훈련 → 평가 과정의 반복

반복적으로 모델을 수정하고 훈련하고 검증 데이터에서 평가한다. (이때 테스트 데이터를 사용하지 않음.)

다시 수정하고 가능한 좋은 모델을 얻을 때까지 반복한다.

* 적용해 볼 수 있는 것
dropout 추가
layer 추가 또는 제거
L1 이나 **L2** 또는 두 가지 모두 추가
하이퍼파라미터 바꿔보기 (층의 유닛 수, optimizer의 학습률 등)
선택적으로 **특성 공학** 시도. (새로운 특성 추가 또는 유용하지 않은 특성 제거.)

2. 최종 모델 훈련시키기

만족할 만한 모델 설정을 얻었다면 가용한 모든 데이터(훈련 데이터와 검증데이터)를 사용해서 제품에 투입할 최종 모델을 훈련시킨다.

3. 마지막에 딱 한번 테스트 세트에서 평가한다.

테스트 세트의 성능이 검증 데이터에서 측정한 것보다 많이 나쁘다면, 검증과정에 전혀 신뢰성이 없거나 모델의 하이퍼파라미터를 튜닝하는 동안 검증 데이터에 과대 적합된 것이다. 이런 경우에는 좀 더 신뢰할 만한 평가 방법으로 바꾸는 것이 좋다.(반복 k겹 교차 검증)

* 유념 사항: **모델 튜닝**을 많이 반복하면 모델이 검증과정에 **과대적합** 될 것이다.

검증과정에서 얻은 피드백을 사용해서 모델을 튜닝할 때마다 검증과정에 대한 정보를 모델에 누설하고 있는 것이다. 많이 반복하게 되면 결국 모델이 검증과정에 과대적합 될 것이다. 이는 검증 과정의 신뢰도를 감소시킨다.