

스마트교통 빅데이터 분석



강사소개

❖ 소속

- ✓ KISTI 데이터기반문제해결연구단 책임연구원
- ✓ 과학기술연합대학원(UST) 응용AI전공 교수
- ✓ 한국기계연구원 융합연구단 (~2026.6)

❖ 연구내용

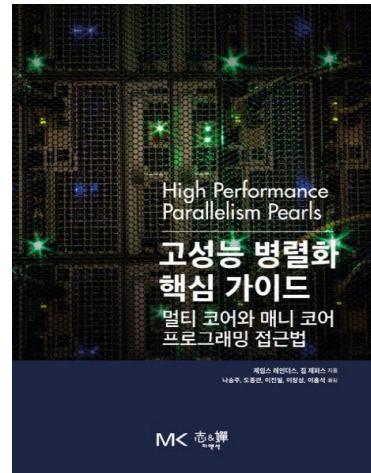
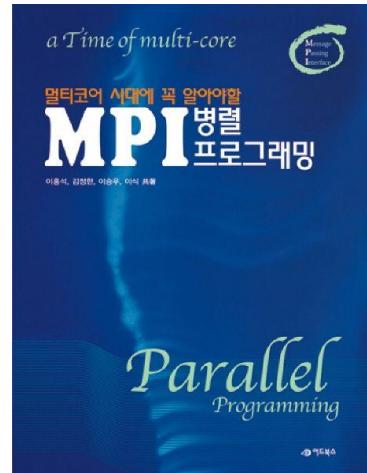
- ✓ 딥러닝기반 도심지 교통혼잡 예측 (2018~2021)
- ✓ 도심지 교통예측을 위한 트랜스포머 모델 연구(2022~2025)
- ✓ 노지 사과 병충해 예방을 위한 AI 기술 연구(2023~2026)

❖ 교육 경험

- ✓ 한국교통학회 텐서플로우 튜토리얼 강좌 (2016~2018)
- ✓ 교통학회/KIRD/KISTI 과학데이터스쿨 강의 진행 (2018~2022)
- ✓ OpenCL, OpenMP, MPI 프로그래밍

❖ 컴퓨팅 기술 및 저서

- ✓ MPI (공저) 병렬프로그래밍
- ✓ ManyCore Computing (공번역)



❖ 1일차 : 분류 모델

- ✓ 데이터의 이해 및 교통 데이터 분석 (2H)
- ✓ 머신러닝을 이용한 교통 흐름 분류 (2H)
- ✓ Tensorflow를 이용한 DNN 분류 모델 적용 (2H)

❖ 2일차 : 회귀 및 예측

- ✓ 회귀 소개와 교통 흐름 회귀 모델 DNN 소개 (2H)
- ✓ 시계열데이터 특성 및 RNN 예측 모델 (2H)
- ✓ LSTM, GRU 모델을 이용한 예측 (2H)

❖ 3일차 : 예측

- ✓ CNN 소개와 교통흐름 예측을 위한 Conv1D 모델 (2H)
- ✓ (예측) 멀티스텝 예측 모델 : DNN, RNN, LSTM, CNN (2H)
- ✓ 프로젝트 (2H)

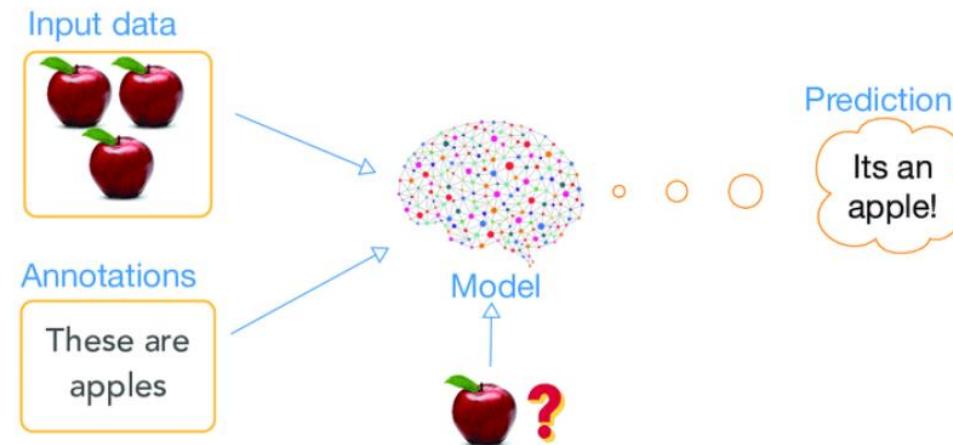
1-1

교통데이터 소개

데이터의 종류

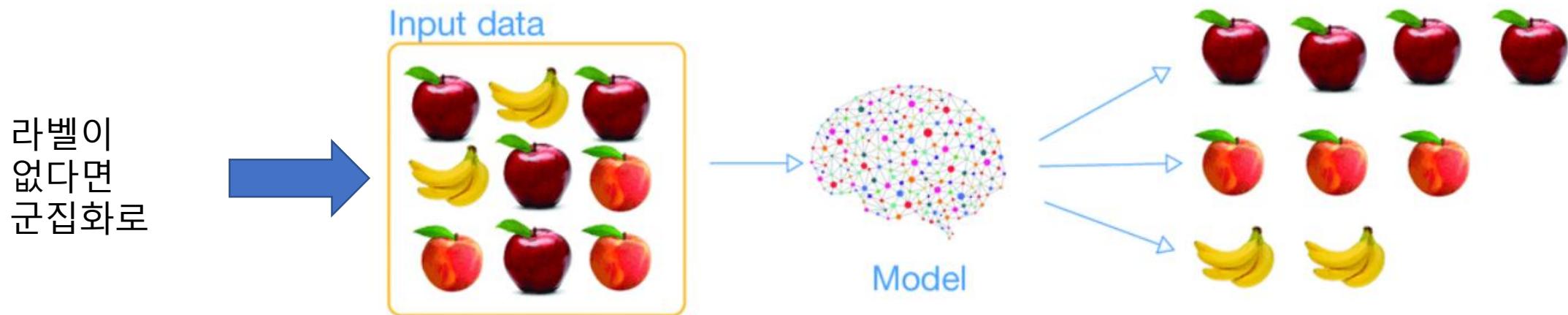
❖ 데이터의 종류(?)

- ✓ 숫자
- ✓ 그림
- ✓ 주식 데이터
 - 시간 순서가 중요하다
- ✓ 텍스트
 - 글자 순서가 중요하다.



❖ 지도학습과 비지도학습

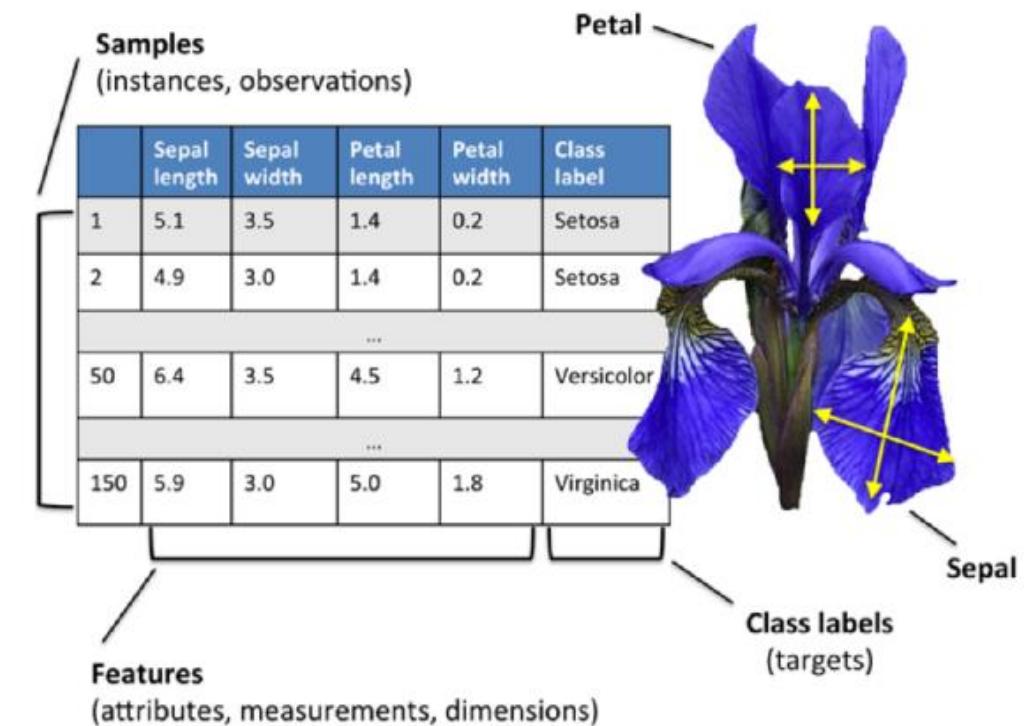
- ✓ 라벨이 있고 실제와 예측값의 최소화



1) 숫자 표현

❖ 숫자 데이터

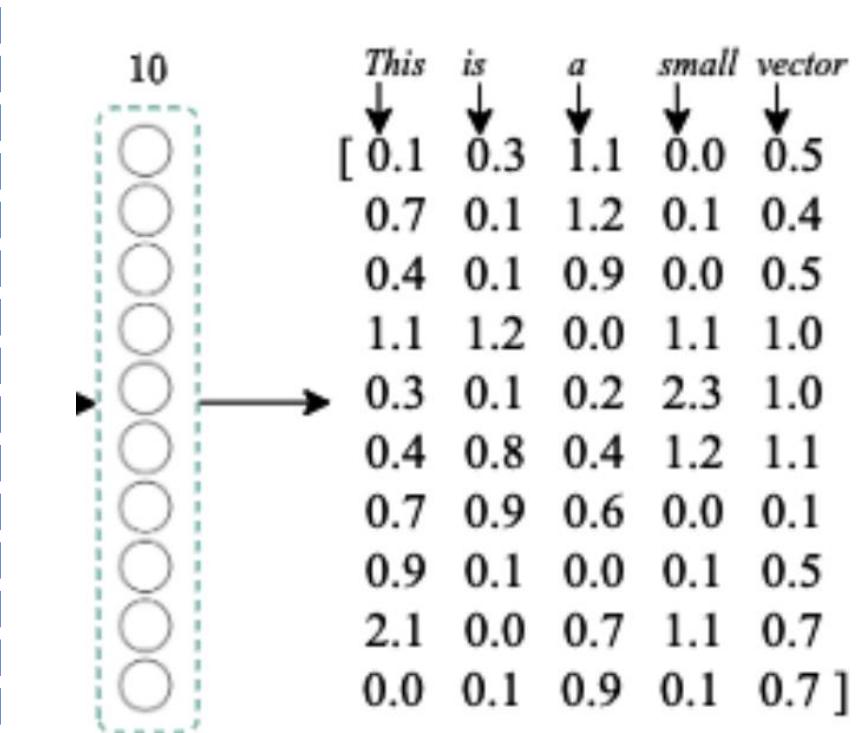
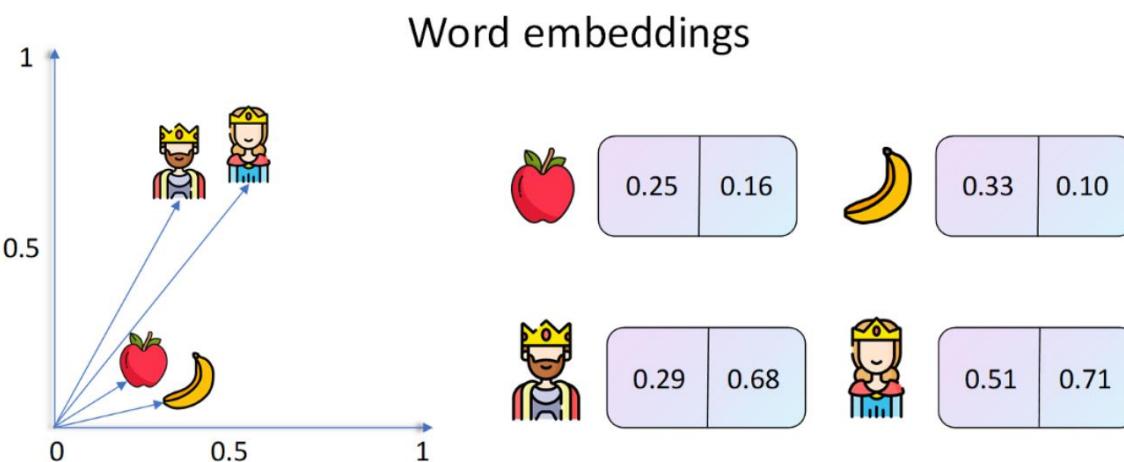
- ✓ 주식, 교통, IRIS 꽃



2) 글자 데이터

❖ 글자(Text)를 숫자로 표현 방법

- ✓ Word Embedding (워드 임베딩)



3) 이미지 데이터

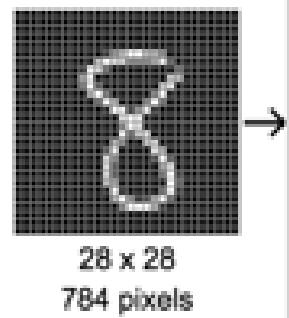
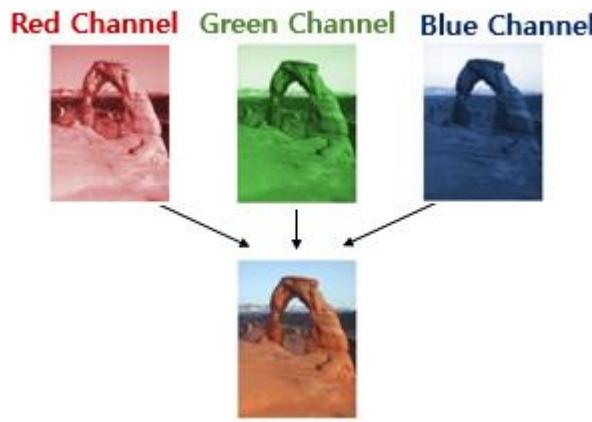
❖ 이미지를 숫자화 방법

✓ 픽셀과 채널(RGB)

- $28 \times 28 \times 1 \rightarrow 784$ pixels
 - 각 픽셀은 256 숫자(2^8)으로 최대값

✓ 채널은 컬러(RGB)

- 3개로 표현



- ❖ 도시 교통 문제의 원인
 - ✓ 교통혼잡은 온실가스, 미세먼지 원인
 - ✓ 도시문제는 지속 가능성에 대한 글로벌한 도전
- ❖ 도시문제 해결을 위한 핵심 키워드는 스마트 시티(Smart city)

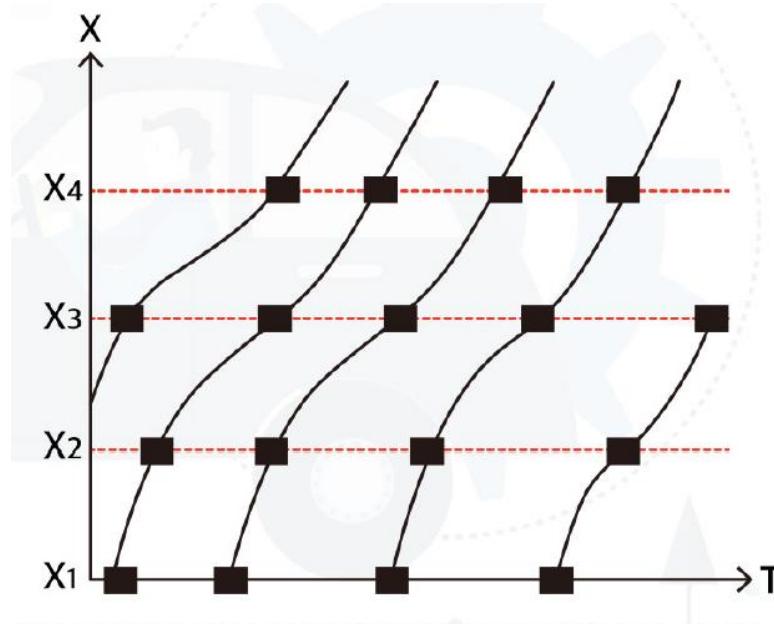


❖ 고정위치 관측

- ✓ 도로상의 고정된 위치에서 지나가는 차량들을 관측
- ✓ 고정된 시각에 일정 구간의 교통 상태를 관측
- ✓ 항공 사진 촬영

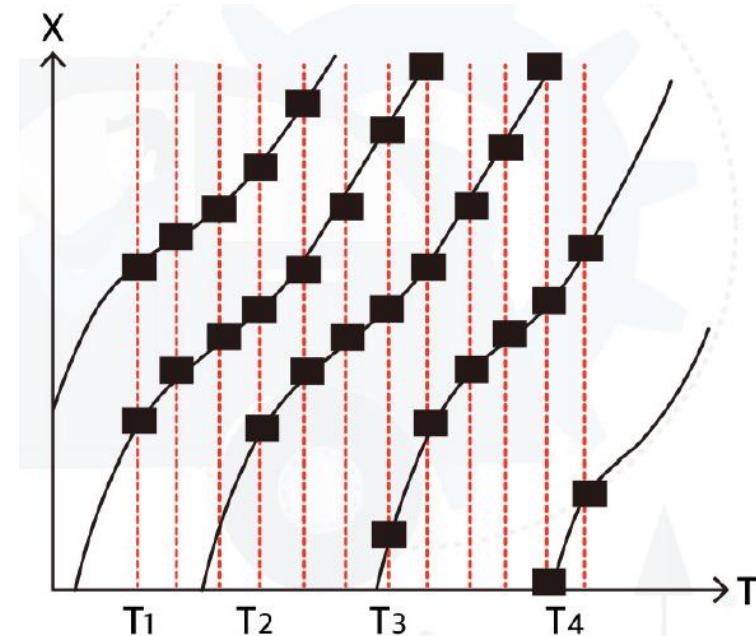
❖ 관측가능한 데이터는

- ✓ 통과 차량수, 차량 사이의 시간 간격, 차량 속도

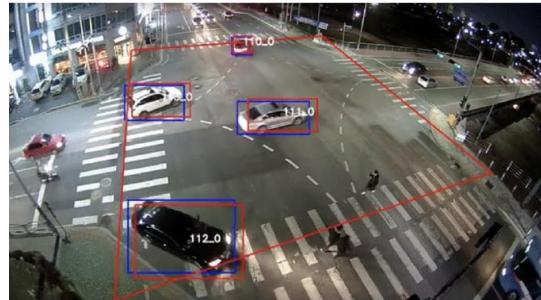


● 연속 시간 관측

- ✓ 일정 구간내의 교통상태를 연속된 시간동안 관측
- ✓ 교통 카메라
- ✓ 관측데이터



영상 검지기



RSU(Road Side Unit)



도로를 운행하는 차량에 설치된 단말기와 WAVE 무선통신을 수행 차량 단말기에서 전송하는 각종 정보를 수집 저장하여 센터로 전송하는 기능

VDS(Vehicle Detection System)



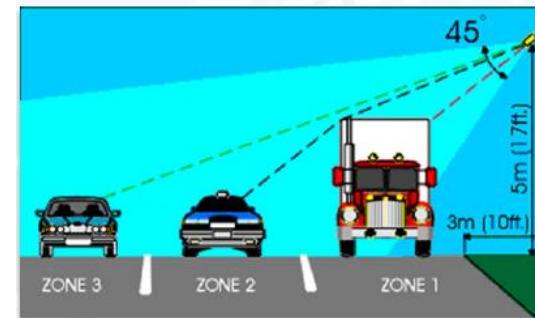
교통량
점유율
속도
시간(1분,5분)

루프 검지기



- 교통량 산정
- 도로 용량 분석
- 차량 통과 속도 감시

교통 레이다

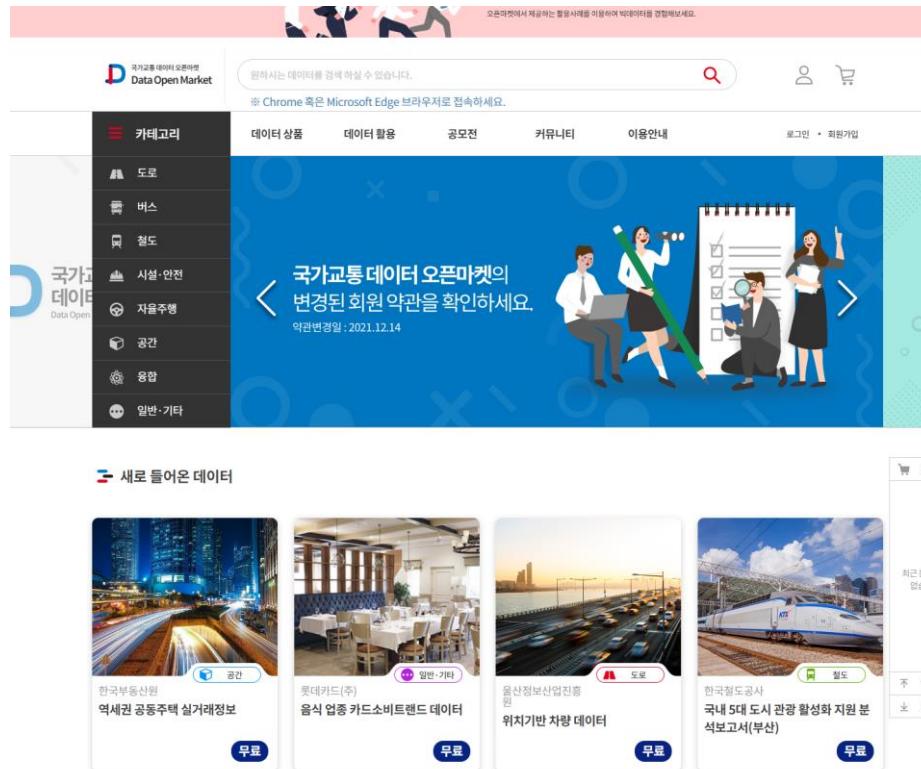


교통 데이터 다운로드

국가교통데이터 오픈마켓 (로그인 필수로 하세요)

<https://www.bigdata-transportation.kr/>

(예제) 도로/대전시청을 선택



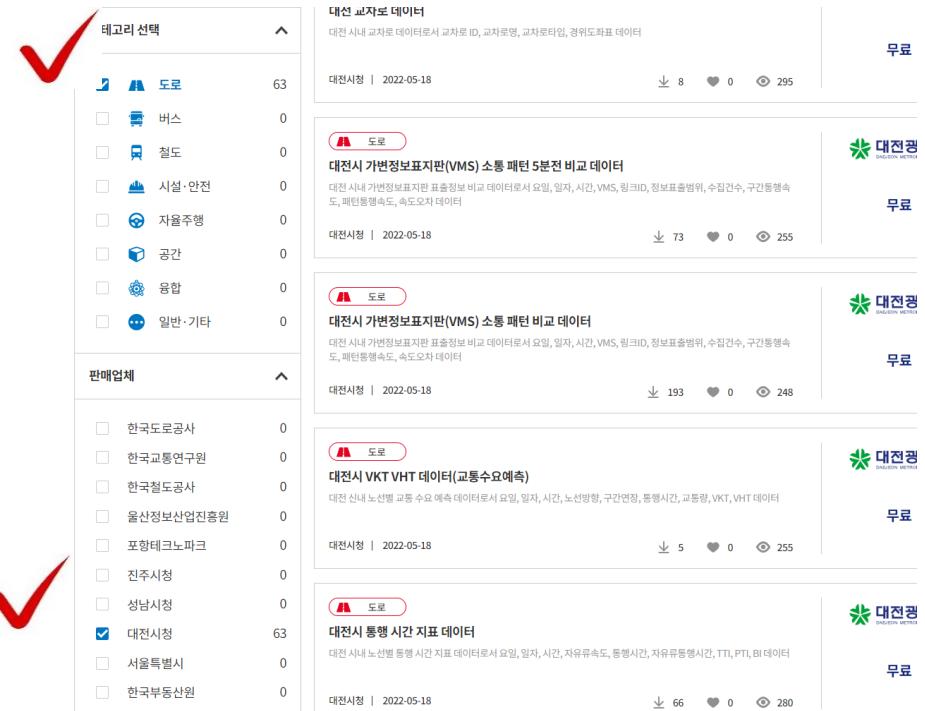
국가교통데이터 오픈마켓
DataOpenMarket

도로 버스 철도 시설·안전 자율주행 공간 융합 일반·기타

국가교통 데이터 오픈마켓의 변경된 회원 약관을 확인하세요.
약관변경일: 2021.12.14

새로 들어온 데이터

- 한국부동산원 역세권 공동주택 실거래정보 (무료)
- 롯데카드(주) 음식 업종 카드소비트랜드 데이터 (무료)
- 울산정보산업진흥원 위치기반 차량 데이터 (무료)
- 한국철도공사 국내 5대 도시 관광 활성화 지원 분석보고서(부산) (무료)



태고리 선택

도로 63

- 버스 0
- 철도 0
- 시설·안전 0
- 자율주행 0
- 공간 0
- 융합 0
- 일반·기타 0

판매업체

- 한국도로공사 0
- 한국교통연구원 0
- 한국철도공사 0
- 울산정보산업진흥원 0
- 포항테크노파크 0
- 진주시청 0
- 성남시청 0
- 대전시청 63
- 서울특별시 0
- 한국부동산원 0

대전 교차로 데이터

대전 시내 교차로 데이터로서 교차로 ID, 교차로명, 교차로타입, 경위도좌표 데이터

대전시청 | 2022-05-18 ↓ 8 ❤ 0 ⚡ 295

도로 대전시 가변정보표지판(VMS) 소통 패턴 5분전 비교 데이터

대전 시내 가변정보표지판 표출정보 비교 데이터로서 요일, 일자, 시간, VMS, 링크ID, 정보표출범위, 수집건수, 구간통행속도, 패턴통행속도, 속도오차 데이터

대전시청 | 2022-05-18 ↓ 73 ❤ 0 ⚡ 255

도로 대전시 가변정보표지판(VMS) 소통 패턴 비교 데이터

대전 시내 가변정보표지판 표출정보 비교 데이터로서 요일, 일자, 시간, VMS, 링크ID, 정보표출범위, 수집건수, 구간통행속도, 패턴통행속도, 속도오차 데이터

대전시청 | 2022-05-18 ↓ 193 ❤ 0 ⚡ 248

도로 대전시 VKT VHT 데이터(교통수요예측)

대전 신내 노선별 교통 수요 예측 데이터로서 요일, 일자, 시간, 노선방법, 구간연장, 통행시간, 교통량, VKT, VHT 데이터

대전시청 | 2022-05-18 ↓ 5 ❤ 0 ⚡ 255

도로 대전시 통행 시간 지표 데이터

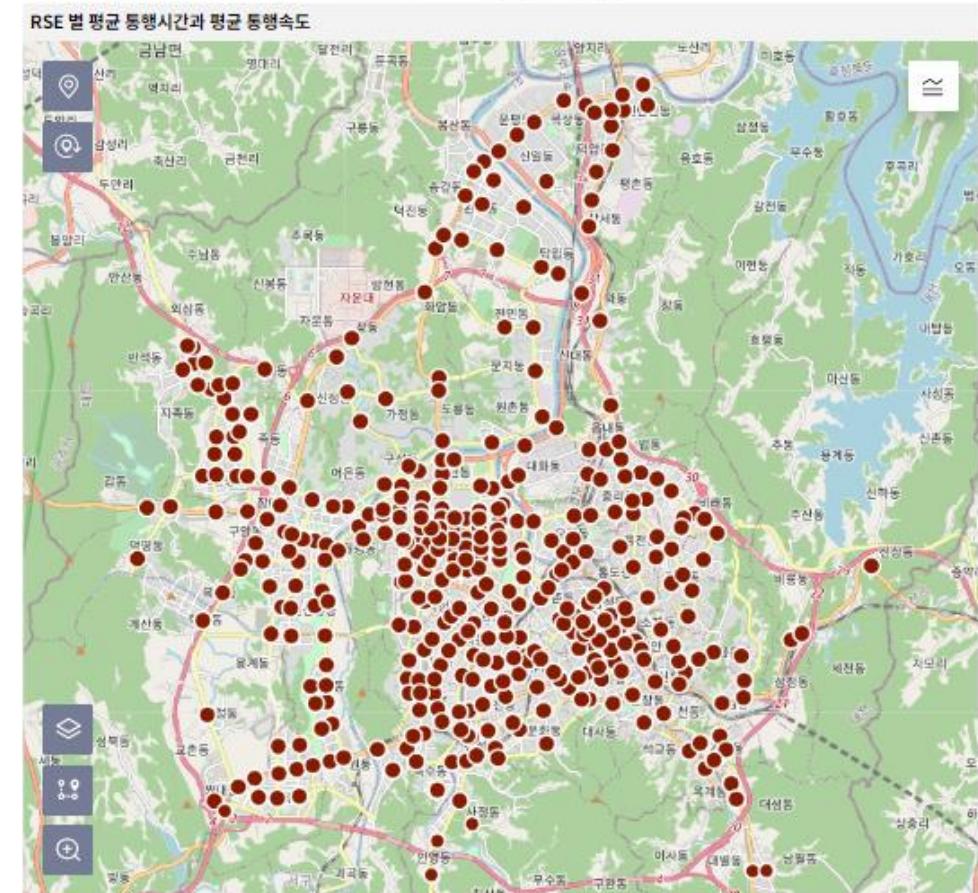
대전 시내 노선별 통행 시간 지표 데이터로서 요일, 일자, 시간, 자유유속도, 통행시간, 자유유동행사간, TTI, PTI, BI 데이터

대전시청 | 2022-05-18 ↓ 66 ❤ 0 ⚡ 280

❖ 첨단 교통량 관리 시스템

- ✓ (ATMS, Advanced Transportation Management System)
 - 실시간 교통상황 정보를 토대로 도로와 같은 교통시설의 이용률을 극대화하기 위한 교통관제체계
- ✓ RSE란
 - 교차로나 차로변에 위치한 CCTV 지주 등에 설치되어 차량의 운행정보를 수집하는 기기
- ✓ 대전시 노변장치(RSE) 정보 5분 단위 데이터
 - 일자 ID, 시 ID, 분 ID, 링크 ID, RSE 통행 속도, RSE 통행 시간 등
- ✓ RSE 위치 및 각 RSE 별 평균 통행시간/통행속도

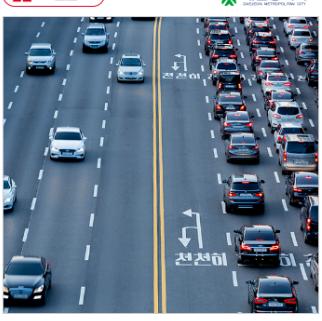
대전시 ATMS RSE 소통 이력 정보 2020년 9월 8일 기준



대전시 VDS 운행 원시 이력 데이터

데이터 상품 상세

도로 대전광역시



1,170 3 3,555

VDS(Vehicle Detection System): 차량검지기
도로 상에 약 1km 간격으로 설치되어 실시간으로 교통량, 점유율, 속도, 대기행렬길이, 차량길이 등의 정보를 검지하여 소통 및 둘별상황 등을 감시하는 장치로 도로 환경적 특성에 따라 설치하며 종류는 루프식, 영상식, 레이더식 등이 있다.

대전시 VDS 운행 원시 이력 데이터 (등록일자, 검지기 ID, VDS ID, VDS 구간 ID, 요일 구분, 교통량 (소/중/대형), 속도, 점유율, 차두 길이, 차두 시간 등)

기본 이용료 무료

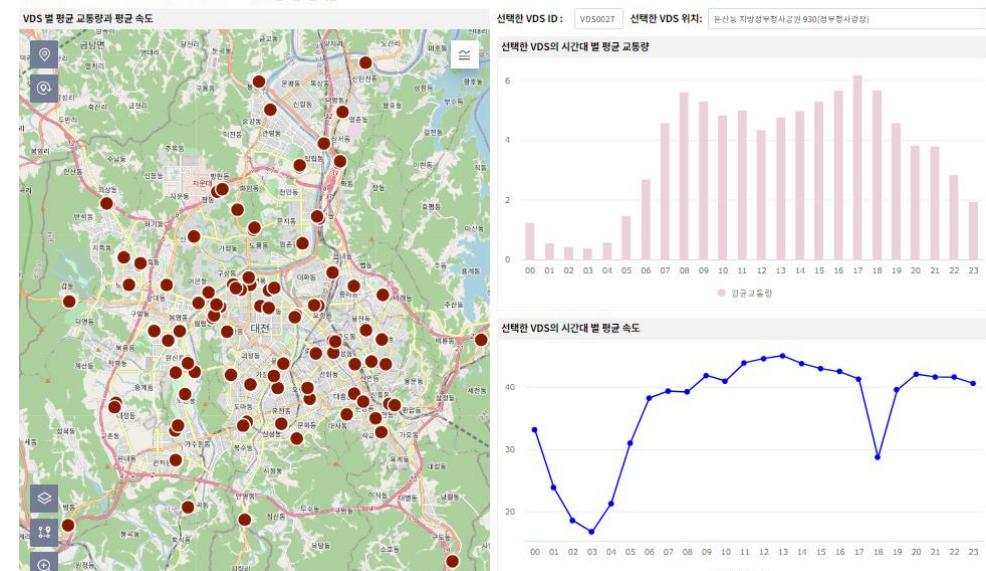
카테고리 도로

제공 기관 대전시청

상품 키워드 #차량검지시스템 #방루프원형검지기 #원시이력 #검지기 #대전시 #교통량 #속도 #매핑 #영상검지 #VDS #루프검지기 #레이더

3 장바구니 데이터 구매하기

대전시 VDS 운행 원시 이력 정보



VDS 데이터 상세 설명

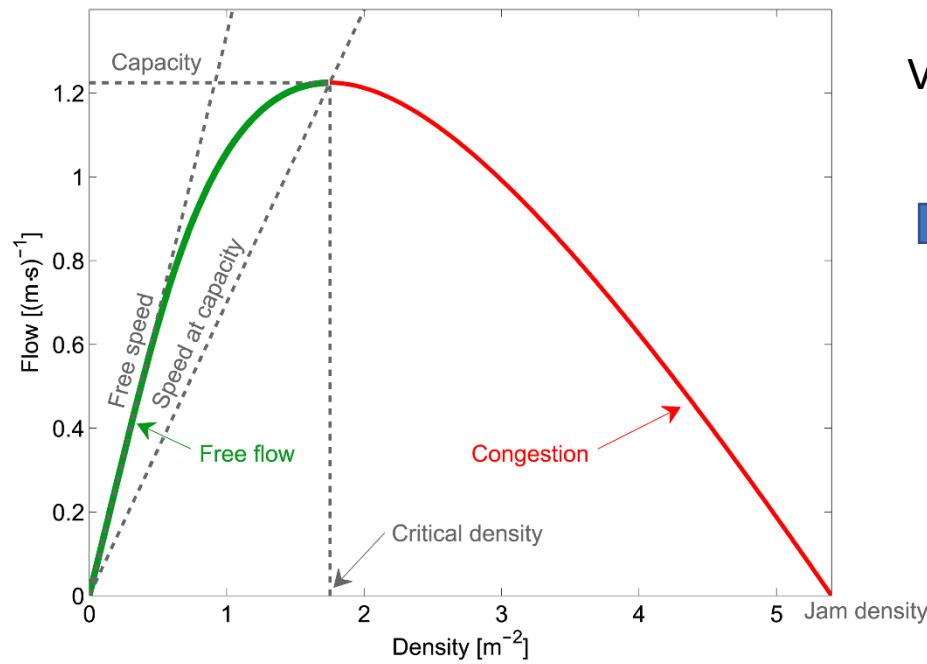
- ❖ 연속류의 대표적인 효과척도라 할 수 있는 교통량, 속도, 밀도의 상관관계
- ❖ 교통량(Q)
 - ✓ 단위시간(일, 시간, 15분 등)에 어떤 지점을 통과하는 차량의 총대수(대)
- ❖ 평균통행속도(km/hr)
 - ✓ 일정구간거리를 통행시간으로 나눈 평균값
- ❖ 밀도(승용차/km/차로)
 - ✓ 어느시간에 단위구간 도로상에 있는 차량대수 (대/km/시간)

| | |
|------------------|-----------------------|
| $Q = V \times K$ | Q : 교통량 V : 속도 K : 밀도 |
|------------------|-----------------------|

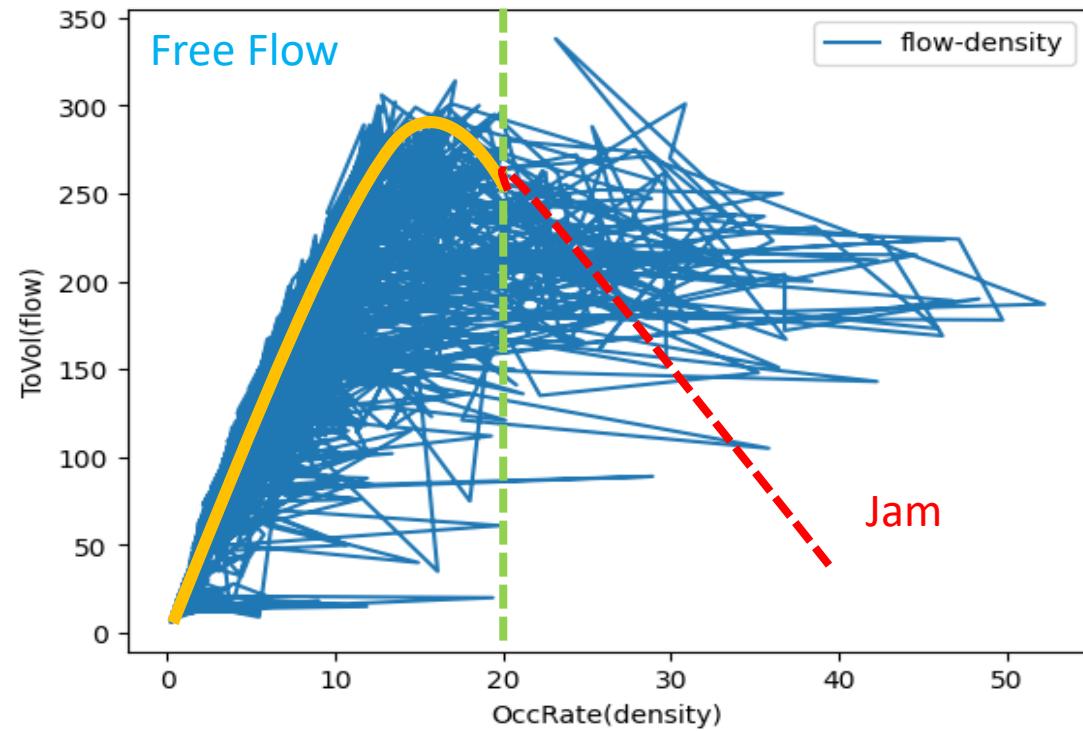
- 교통량이 일정할때 밀도가 높으면 속도저하 발생
- 3요소 상호간의 작용에 의해 각 요소의 성격이 규명된다는 가정하에 접근

<https://moneyjang.tistory.com/305#gsc.tab=0>

- ❖ 교통량이 많으면 밀도 증가 (자유류)
- ❖ 교통량이 용량을 초과하면 밀도는 증가하지만 교통량은 감소(혼잡류)



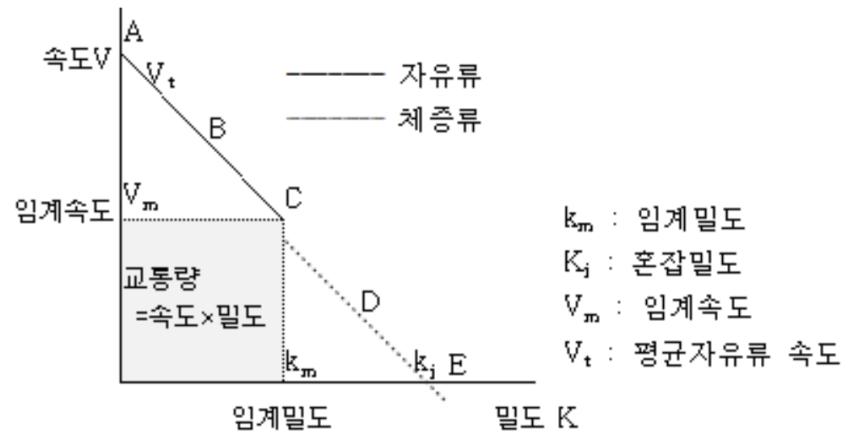
VDS16



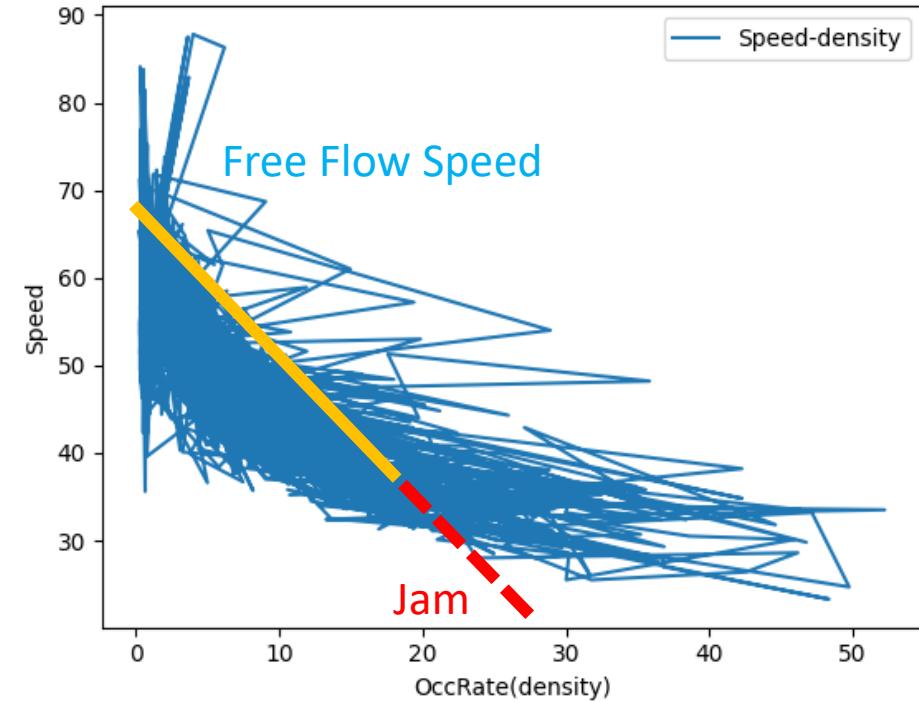
참고자료: <https://journals.plos.org/plosone/article/figure?id=10.1371/journal.pone.0208496.g001>

속도-밀도(점유율)

- 밀도가 증가하면 속도 저하
- 도로의 최대교통류율은 도로의 용량이 되며
이때 밀도가 임계밀도, 이때 속도가 임계속도

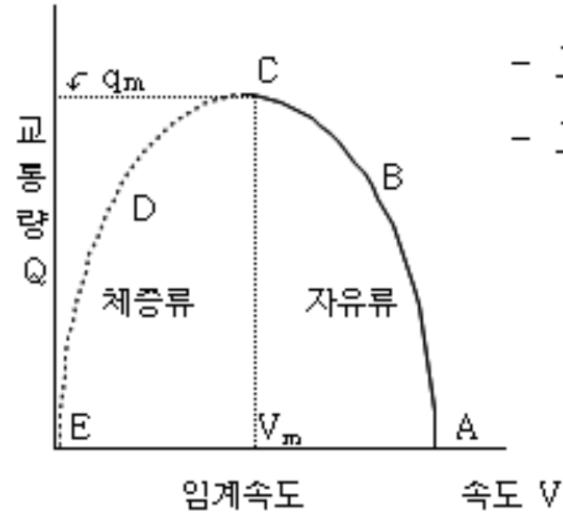


VDS16



참고문서: <https://rashidfaridi.com/2018/12/11/concept-of-traffic-flow-diagram/>

- ❖ 교통량이 많으면 속도 저하

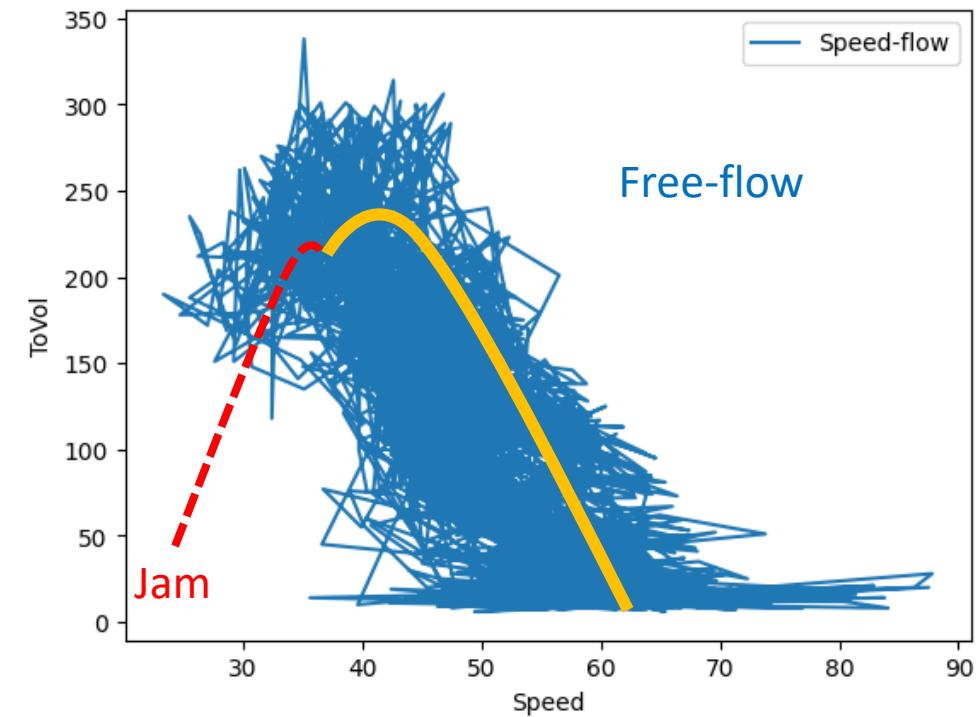
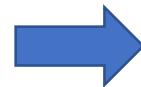


- 교통량이 많아지면 속도 저하(자유류)
- 교통량이 용량을 초과하게 되면

교통량 및 속도 감소(체증류)

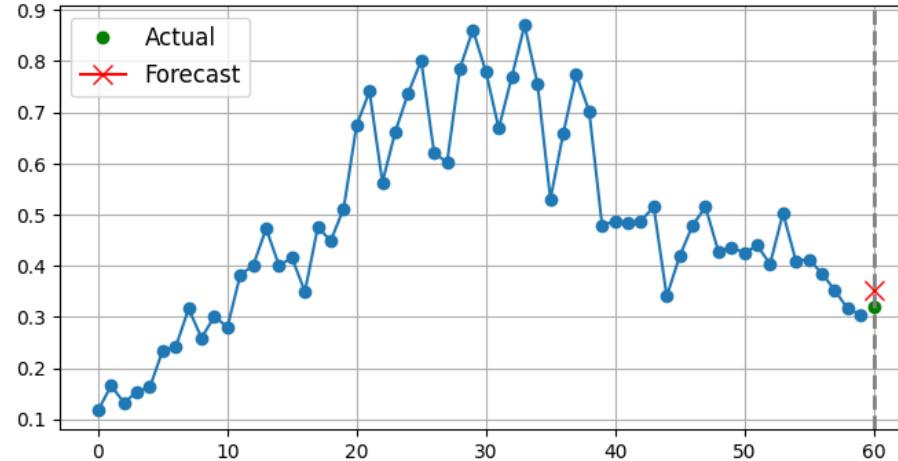
q_m : 최대교통류율(용량)
 V_m : 임계속도

VDS16

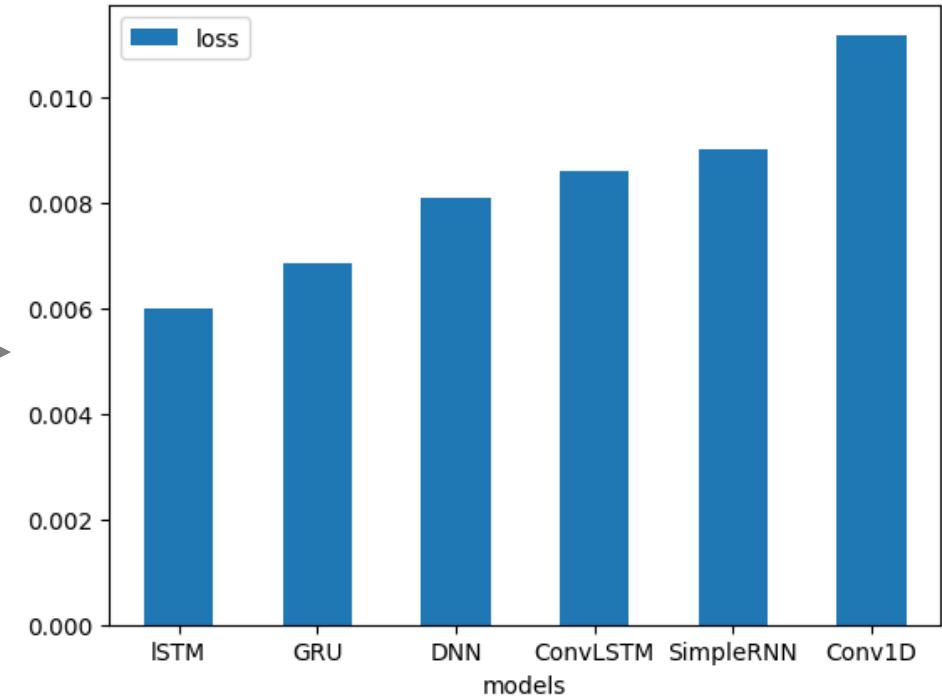
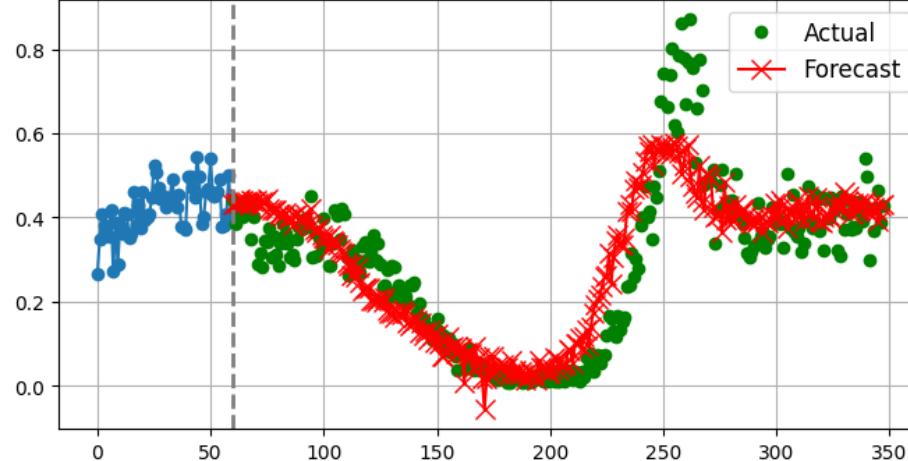


교통 데이터를 활용한 딥러닝 모델 비교 (예시)

LSTM 예측 : One-step ahead prediction



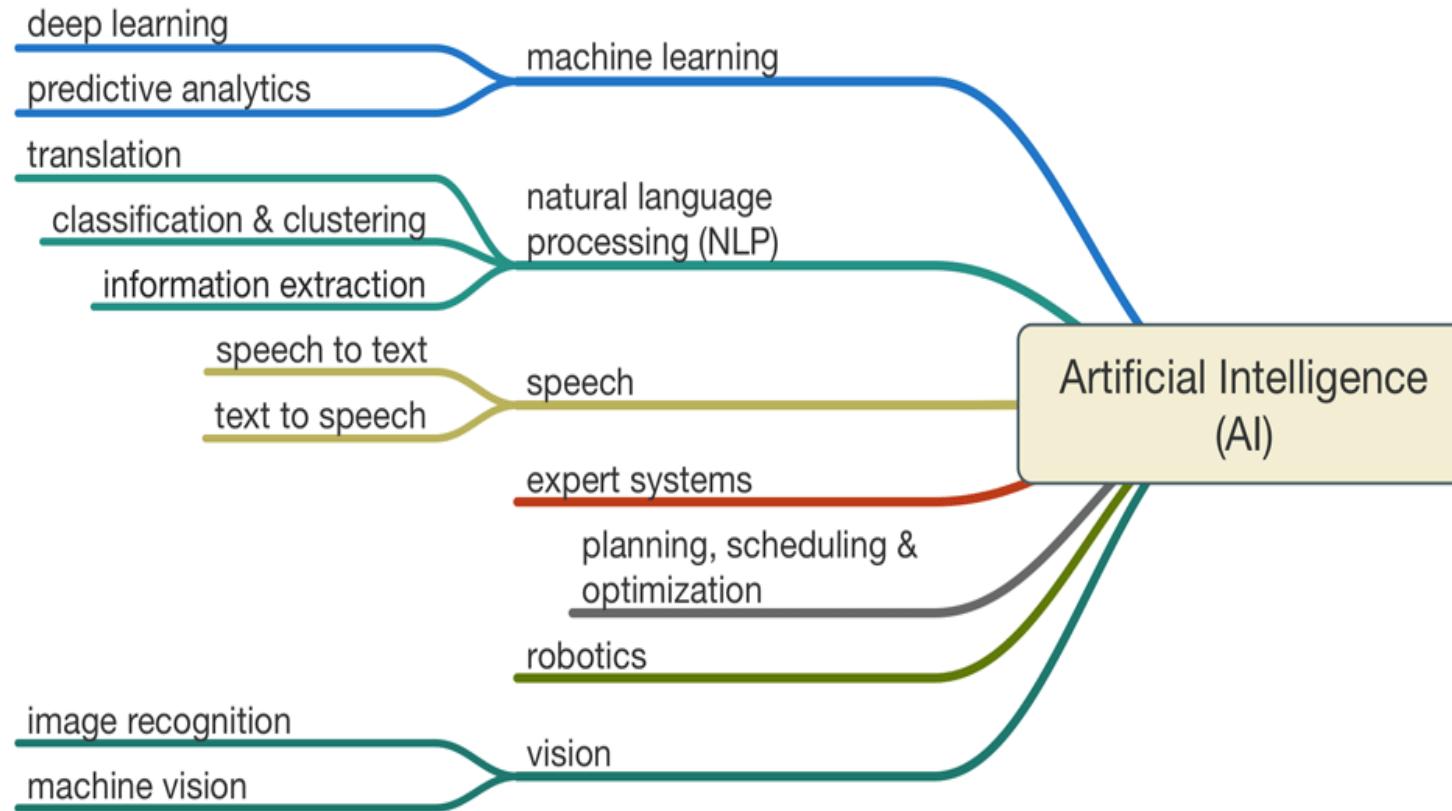
LSTM 예측 : Multi-step ahead prediction



1-2

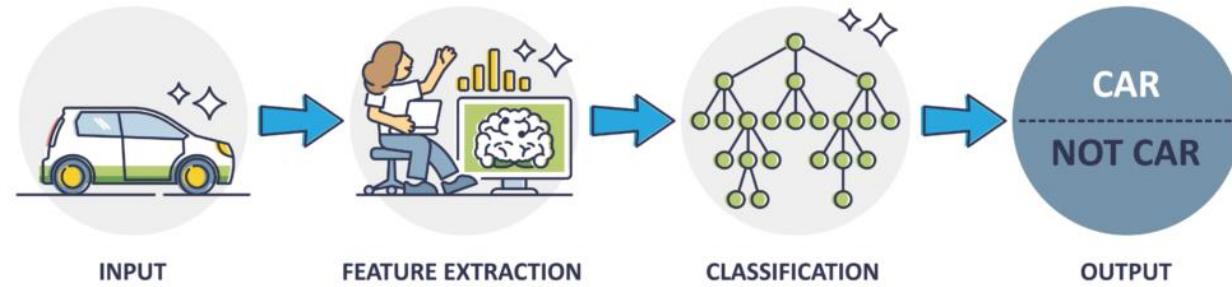
데이터 레이블링과 지도학습 소개

인공지능의 분류



자료: <https://www.eyerys.com/articles/paving-roads-artificial-intelligence-its-either-us-or-them>

MACHINE LEARNING



DEEP LEARNING



(source) <https://www.ait.de/en/deep-learning/>

1) 데이터의 전처리 과정

- ✓ 피처와 라벨을 결정함,
- ✓ 필요한 피처를 추출하고 가공함,
- ✓ 입력과 출력을 위한 피처의 정규화(표준화)

2) 머신러닝 모델을 선정

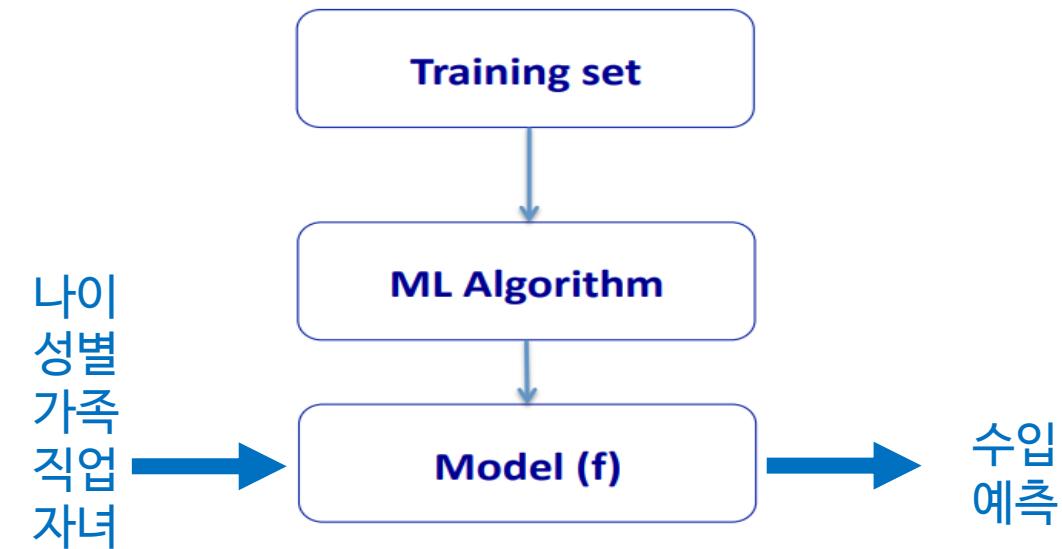
- ✓ 학습, 검증, 테스트 데이터로 나누자
- ✓ 적용할 머신러닝 알고리즘을 선정하자.
 - 분류 혹은 회귀 문제인지 먼저 정하자.

3) 모델 훈련

- ✓ 모델을 훈련시키자.
- ✓ 과적합 발생등 확인하자.

4) 모델 평가하자.

- ✓ 모델의 테스트로 평가하자.



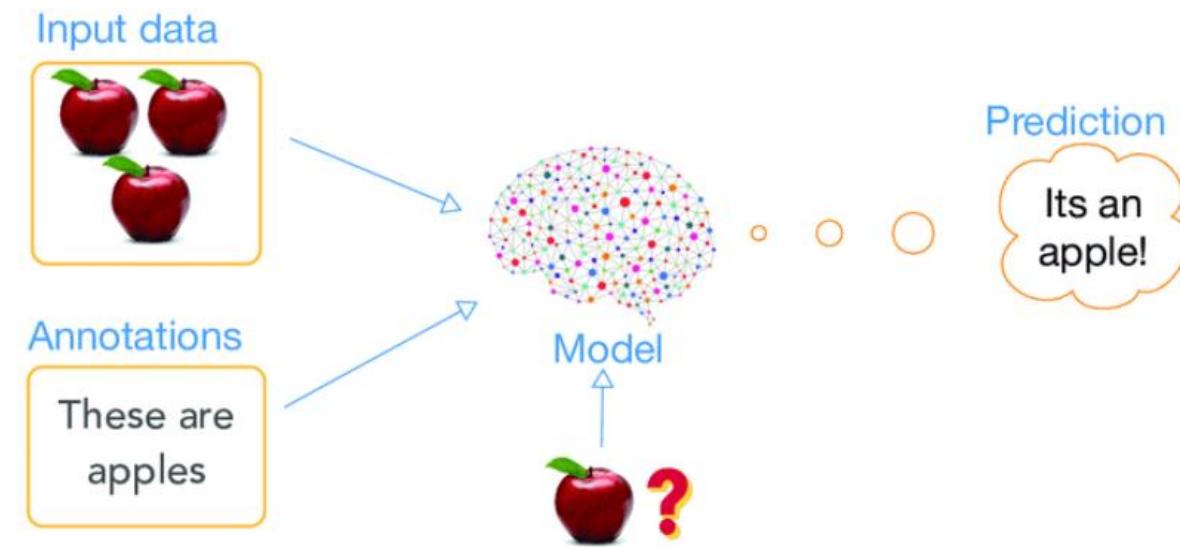
❖ 데이터의 라벨

- ✓ MNIST 데이터로 이미지 분류
 - 10개의 숫자 분류/라벨
- ✓ Fashion_MNIST 데이터
 - 10개의 패션 라벨
- ✓ IMDB 데이터로 감성분류
 - 영화 리뷰

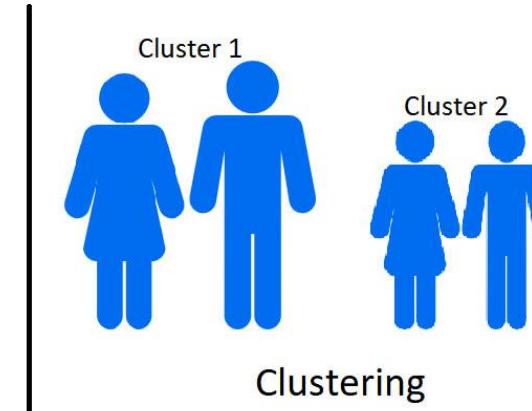
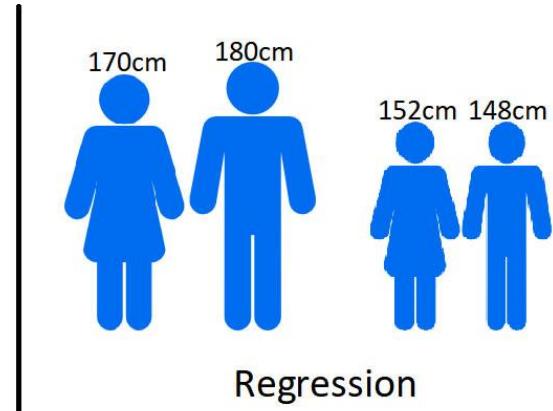
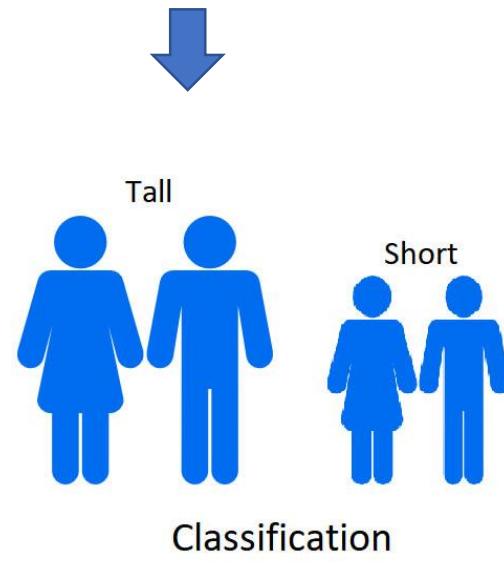
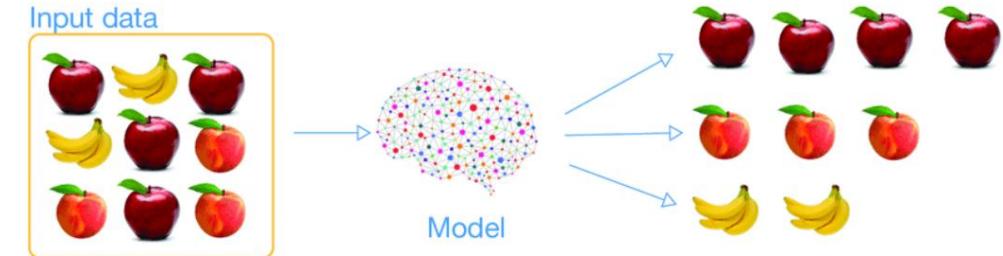
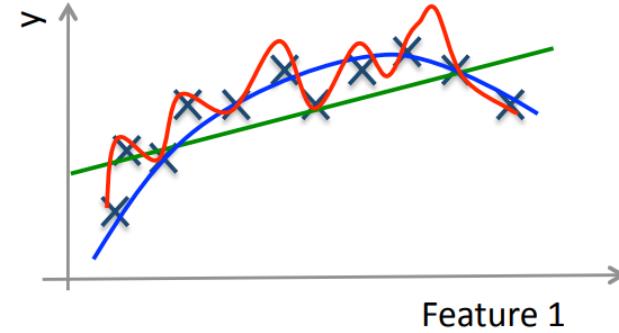
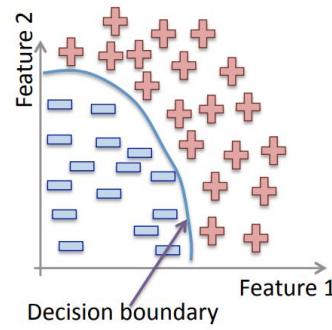
❖ 지도학습은 레이블이 있어야 함

- ✓ 라벨이 있고 실제와 예측값의 최소화

❖ 누가 레이블을 만드나?



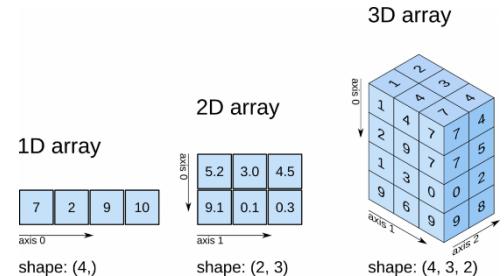
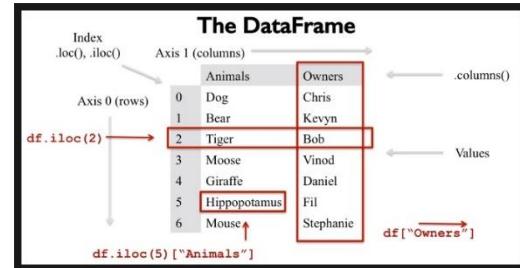
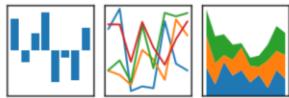
데이터를 이용한 ML/DL 모델 종류



머신러닝에 많이 사용되는 도구(라이브러리)

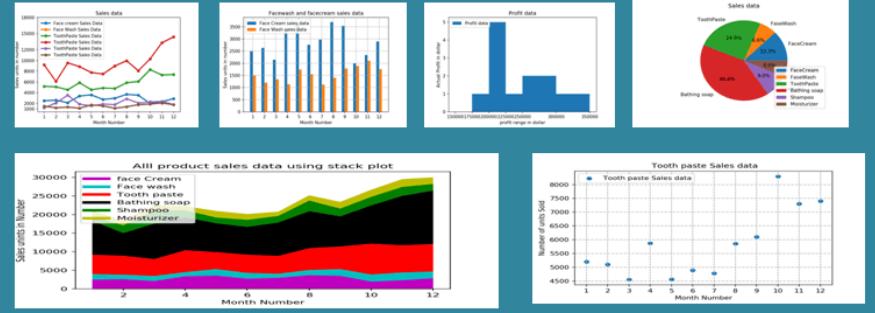
pandas

$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$



Python Matplotlib

Practice Data Visualization In, Practice Questions Online, Solution Provided for Each Question



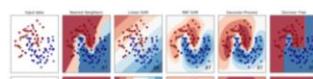
scikit-learn

Classification

Identifying which category an object belongs to.

Applications: Spam detection, image recognition.

Algorithms: SVM, nearest neighbors, random forest, and more...

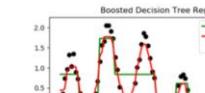


Regression

Predicting a continuous-valued attribute associated with an object.

Applications: Drug response, Stock prices.

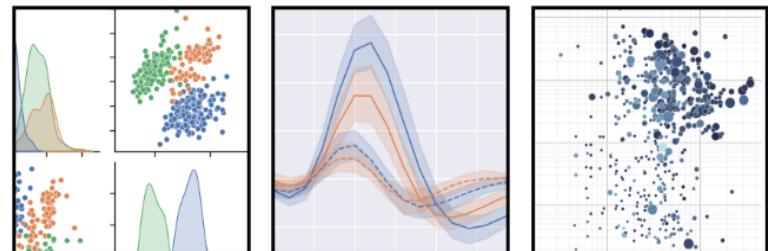
Algorithms: SVR, nearest neighbors, random forest, and more...



TensorFlow PYTORCH



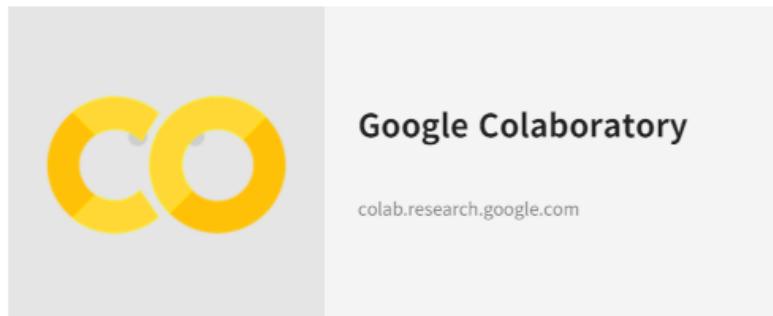
seaborn: statistical data visualization



Google Colab 스펙

- ❖ 코랩이란?
 - ✓ 클라우드에서 실행되는 무료 Jupyter 노트 환경
- ❖ Google Colaboratory 접속하기
 - ✓ <https://colab.research.google.com> 접속
- ❖ Google Colab 스펙
 - ✓ CPU : 제온
 - ✓ Memory : 13GB
 - ✓ HDD : 320GB
 - ✓ GPU : NVIDIA Tesla K80

<https://colab.research.google.com/>

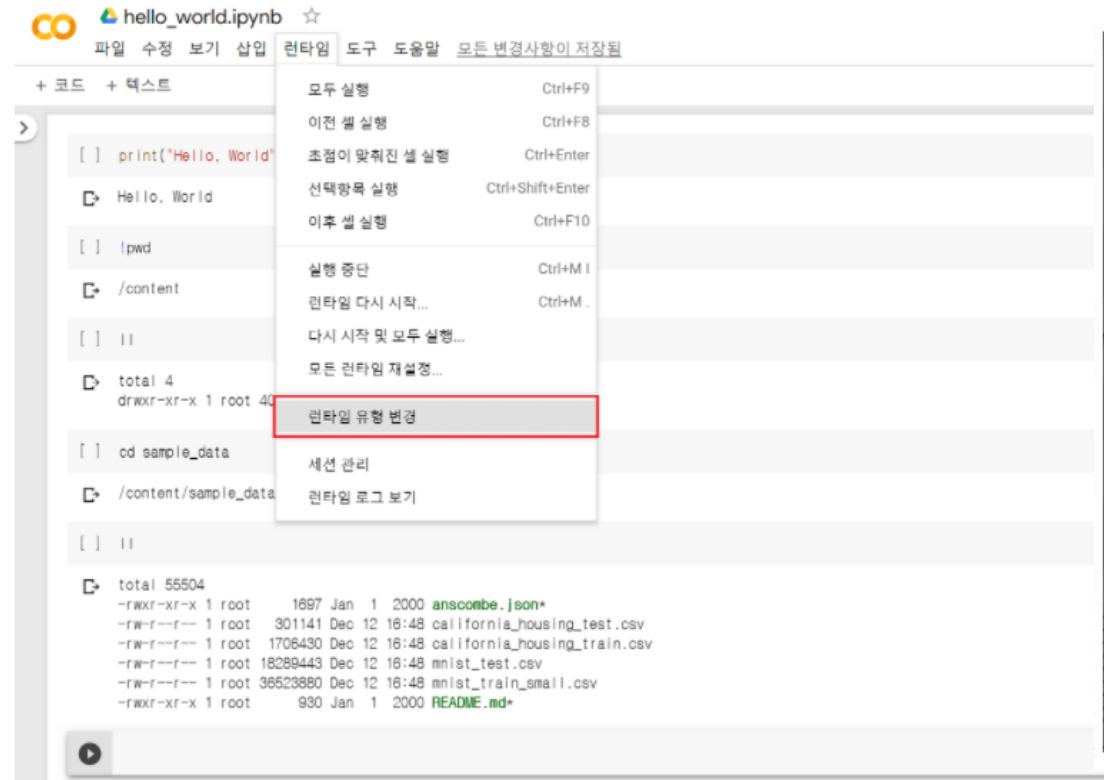


런타임 유형 GPU, TPU 설정하기

무료답지 않게 엄청난 서비스가 무료입니다.

GPU, TPU 등을 지원하기 때문에 요즘 핫한 Tensorflow등을 동작할 수 있습니다.

Colab의 GPU 정보 확인 방법:
!nvidia-smi



hello_world.ipynb

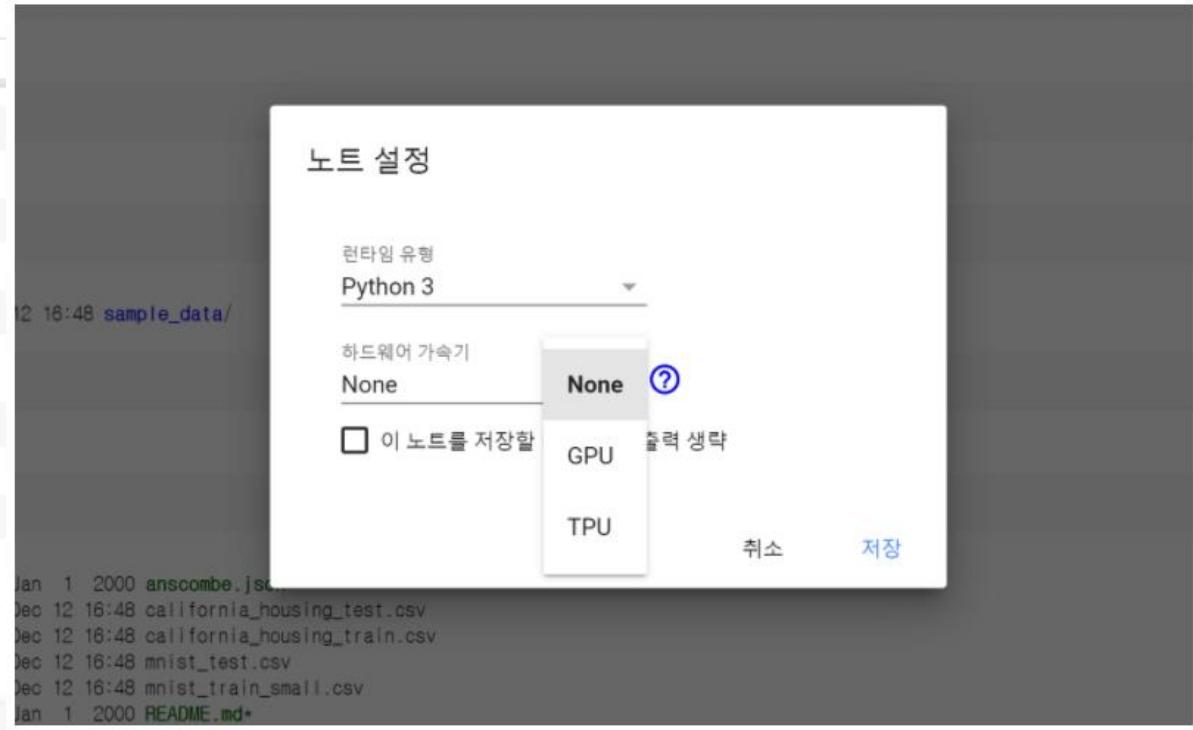
파일 수정 보기 삽입 런타임 도구 도움말 모든 변경사항이 저장됨

+ 코드 + 텍스트

```
[ ] print("Hello, World")
Hello, World
[ ] pwd
/content
[ ] !ls
Hello, World
[ ] total 4
drwxr-xr-x 1 root 4096 Dec 12 16:48 sample_data
[ ] cd sample_data
[ ] ls
```

런타임 유형 변경

모두 실행 Ctrl+F9
이전 셀 실행 Ctrl+F8
초점이 맞춰진 셀 실행 Ctrl+Enter
선택항목 실행 Ctrl+Shift+Enter
이후 셀 실행 Ctrl+F10
실행 중단 Ctrl+M I
런타임 다시 시작... Ctrl+M .
다시 시작 및 모두 실행... Ctrl+M ..
모든 런타임 재설정...



1-2

코랩에서 교통 데이터 가시화

```
▶ import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
import seaborn as sns
```

```
[2] # 맷플랏립에서 한글을 사용하고자 한다면 아래와 같이  
!pip install koreanize-matplotlib  
import koreanize_matplotlib
```

```
[3] df = pd.read_csv("https://raw.githubusercontent.com/hongsukyi/Lectures/main/data/vds16.csv")
```

```
▶ df.head()
```

| | Date | ToVol | SmVol | MeVol | LaVol | Speed | OccRate |
|---|-----------------|-------|-------|-------|-------|-------|---------|
| 0 | 2017-04-02 0:00 | 43 | 34 | 9 | 0 | 50.3 | 1.90 |
| 1 | 2017-04-02 0:05 | 45 | 32 | 13 | 0 | 58.9 | 1.84 |
| 2 | 2017-04-02 0:10 | 46 | 34 | 12 | 0 | 50.6 | 1.87 |
| 3 | 2017-04-02 0:15 | 45 | 36 | 9 | 0 | 50.9 | 1.72 |
| 4 | 2017-04-02 0:20 | 27 | 13 | 13 | 1 | 62.2 | 1.12 |

[5] df.tail(3)



Date ToVol SmVol MeVol LaVol Speed OccRate



| | | | | | | | |
|------|------------------|----|----|----|---|------|------|
| 8061 | 2017-04-29 23:45 | 32 | 28 | 4 | 0 | 50.6 | 1.36 |
| 8062 | 2017-04-29 23:50 | 31 | 21 | 10 | 0 | 59.3 | 1.40 |
| 8063 | 2017-04-29 23:55 | 39 | 33 | 6 | 0 | 52.5 | 1.74 |

▶ df.info()

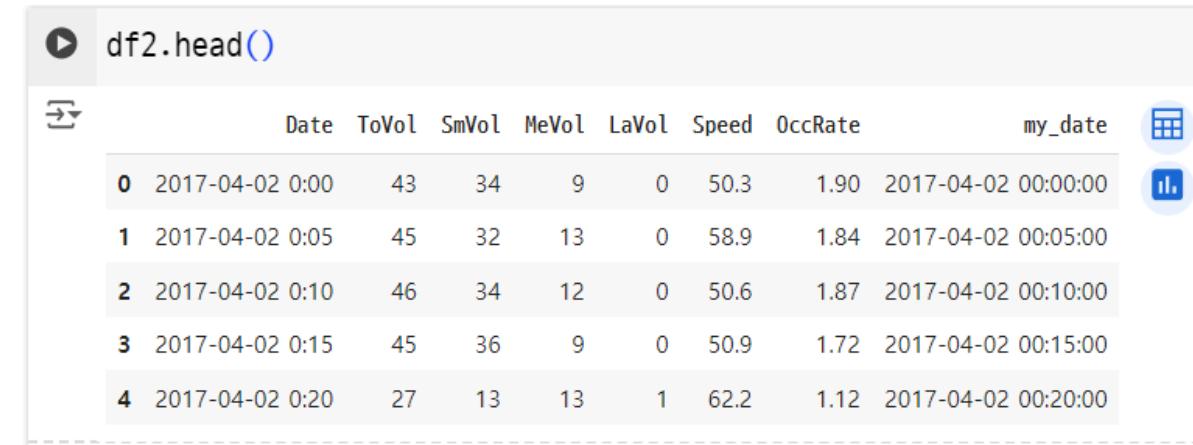
```
>>> <class 'pandas.core.frame.DataFrame'>
RangeIndex: 8064 entries, 0 to 8063
Data columns (total 7 columns):
 #   Column   Non-Null Count  Dtype  
--- 
 0   Date      8064 non-null   object 
 1   ToVol     8064 non-null   int64  
 2   SmVol    8064 non-null   int64  
 3   MeVol     8064 non-null   int64  
 4   LaVol     8064 non-null   int64  
 5   Speed      8064 non-null   float64
 6   OccRate   8064 non-null   float64
dtypes: float64(2), int64(4), object(1)
memory usage: 441.1+ KB
```

```
[8] df2['my_date']=pd.to_datetime(df['Date'], format='%Y-%m-%d %H:%M')
```

```
[9] df2.info()
```

```
→ <class 'pandas.core.frame.DataFrame'>
RangeIndex: 8064 entries, 0 to 8063
Data columns (total 8 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   Date        8064 non-null    object  
 1   ToVol       8064 non-null    int64  
 2   SmVol       8064 non-null    int64  
 3   MeVol       8064 non-null    int64  
 4   LaVol       8064 non-null    int64  
 5   Speed        8064 non-null    float64 
 6   OccRate     8064 non-null    float64 
 7   my_date     8064 non-null    datetime64[ns]
dtypes: datetime64[ns](1), float64(2), int64(4), object(1)
memory usage: 504.1+ KB
```

df2.head()



| | Date | ToVol | SmVol | MeVol | LaVol | Speed | OccRate | my_date |
|---|-----------------|-------|-------|-------|-------|-------|---------|---------------------|
| 0 | 2017-04-02 0:00 | 43 | 34 | 9 | 0 | 50.3 | 1.90 | 2017-04-02 00:00:00 |
| 1 | 2017-04-02 0:05 | 45 | 32 | 13 | 0 | 58.9 | 1.84 | 2017-04-02 00:05:00 |
| 2 | 2017-04-02 0:10 | 46 | 34 | 12 | 0 | 50.6 | 1.87 | 2017-04-02 00:10:00 |
| 3 | 2017-04-02 0:15 | 45 | 36 | 9 | 0 | 50.9 | 1.72 | 2017-04-02 00:15:00 |
| 4 | 2017-04-02 0:20 | 27 | 13 | 13 | 1 | 62.2 | 1.12 | 2017-04-02 00:20:00 |

- ❖ 데이터 프레임에 인덱스를 선언하고, 2일간 288x2의 교통량을 그림으로 그리자

```
[11] df=df.set_index('Date')
```

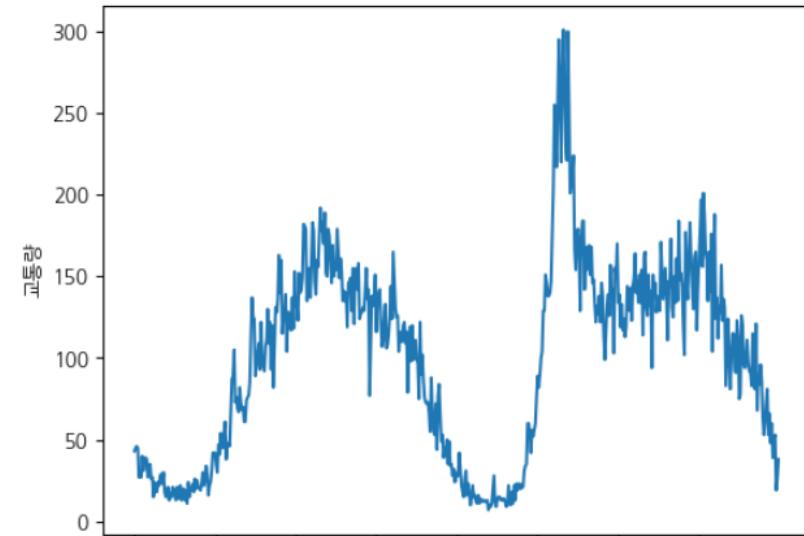
```
[12] df.head()
```

| | ToVol | SmVol | MeVol | LaVol | Speed | OccRate |
|-----------------|-------|-------|-------|-------|-------|---------|
| Date | | | | | | |
| 2017-04-02 0:00 | 43 | 34 | 9 | 0 | 50.3 | 1.90 |
| 2017-04-02 0:05 | 45 | 32 | 13 | 0 | 58.9 | 1.84 |
| 2017-04-02 0:10 | 46 | 34 | 12 | 0 | 50.6 | 1.87 |
| 2017-04-02 0:15 | 45 | 36 | 9 | 0 | 50.9 | 1.72 |
| 2017-04-02 0:20 | 27 | 13 | 13 | 1 | 62.2 | 1.12 |

```
[13] # 5분 단위로 가공된 데이터를 2일 간 그리고자 한다면  
nsize = 288*2
```

```
▶ # 교통량을 2일 동안 그려보자 한다면,  
plt.plot(df["ToVol"][:nsize])  
plt.xticks(np.arange(0,nsize,72),rotation=45)  
plt.xlabel('날자')  
plt.ylabel('교통량')
```

▶ Text(0, 0.5, '교통량')



❖ 속도, 교통량, 점유률 2일간 그리기

```
[18] # 속도 그리기
```

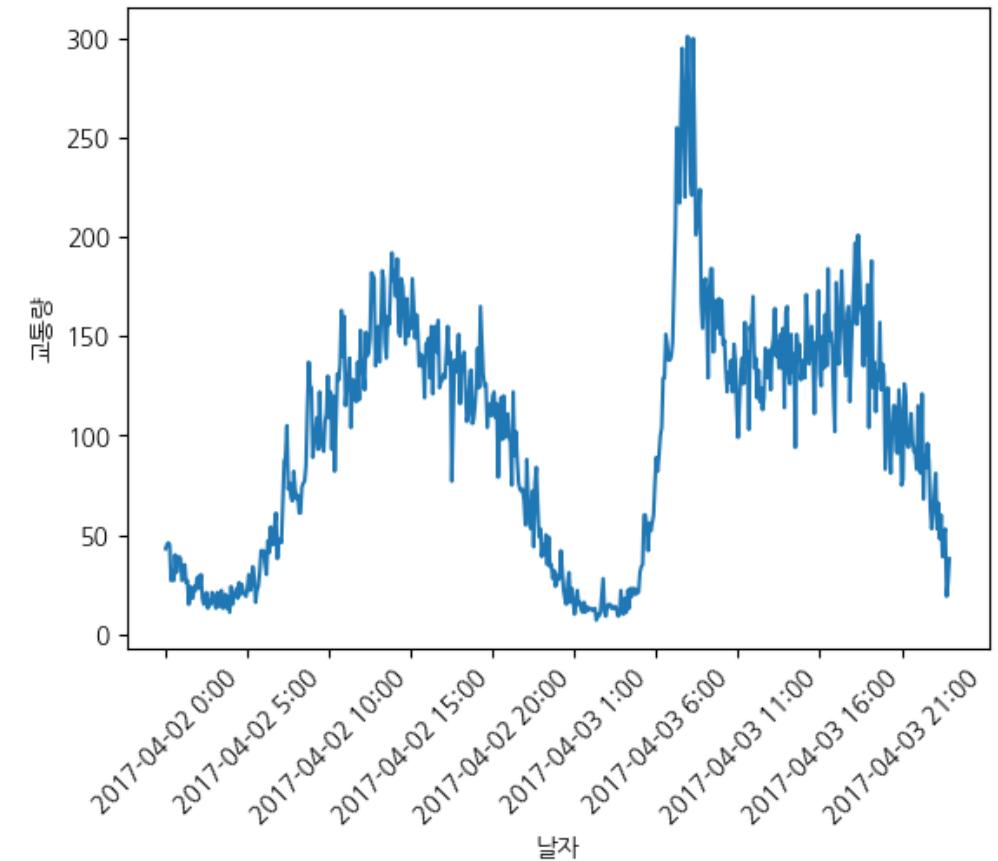
```
plt.plot(df.index[0:nsize],df["Speed"][0:nsize])
plt.xticks(np.arange(0,nsize,72),rotation=45)
plt.ylabel('Speed')
```

▶

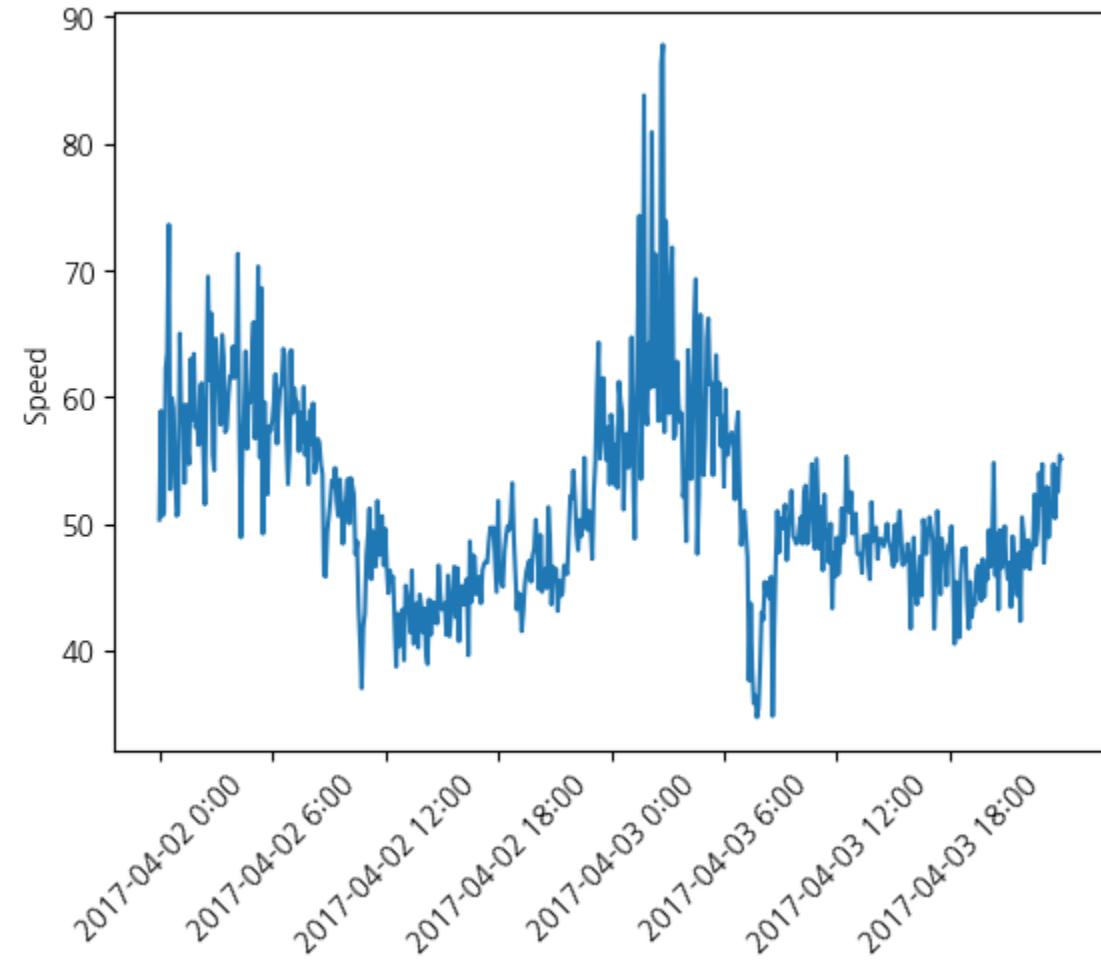
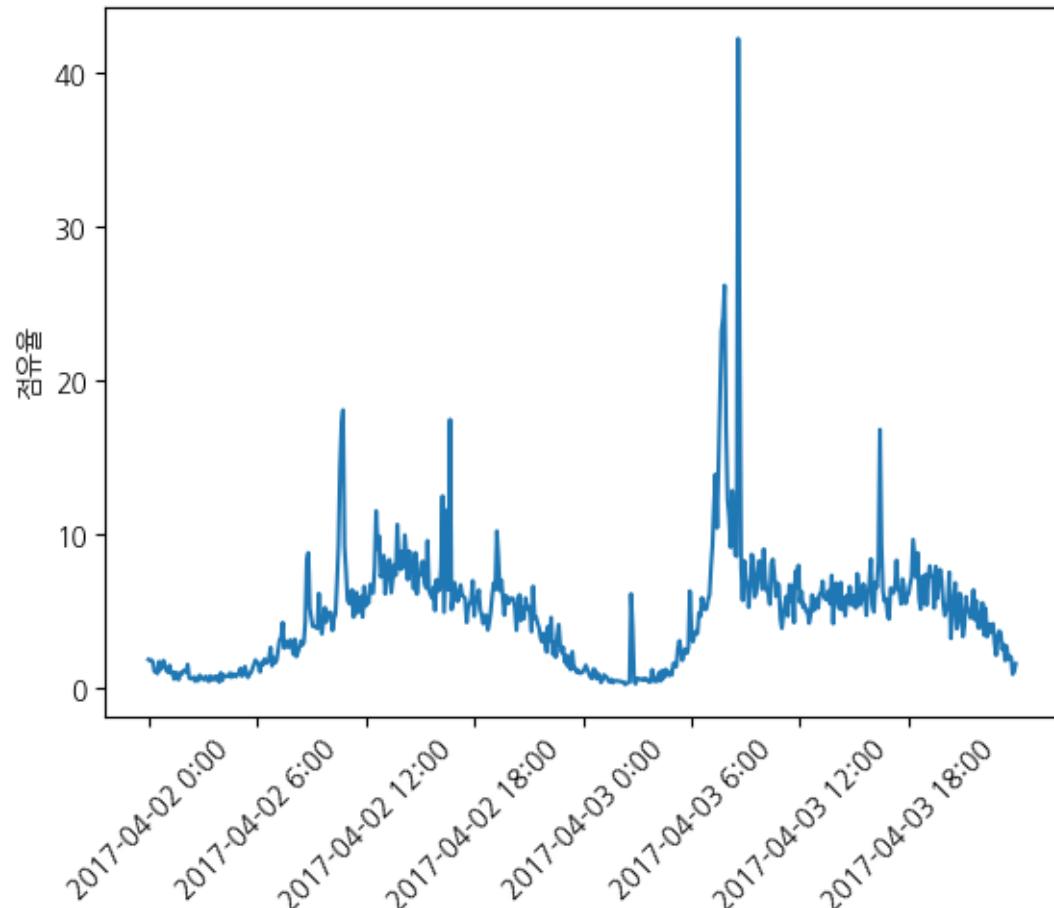
```
plt.plot(df.index[0:nsize],df["OccRate"][0:nsize])
plt.xticks(np.arange(0,nsize,72),rotation=45)
plt.ylabel('점유율')
```

▶ # x-축으로 날짜를 표시하고자 한다면

```
plt.plot(df.index[0:nsize],df["ToVol"][0:nsize],label='ToVol')
plt.xticks(np.arange(0,nsize,60),rotation=45)
plt.xlabel('날자')
plt.ylabel('교통량')
```



VDS 가시화



❖ 교통량과 점유율관계

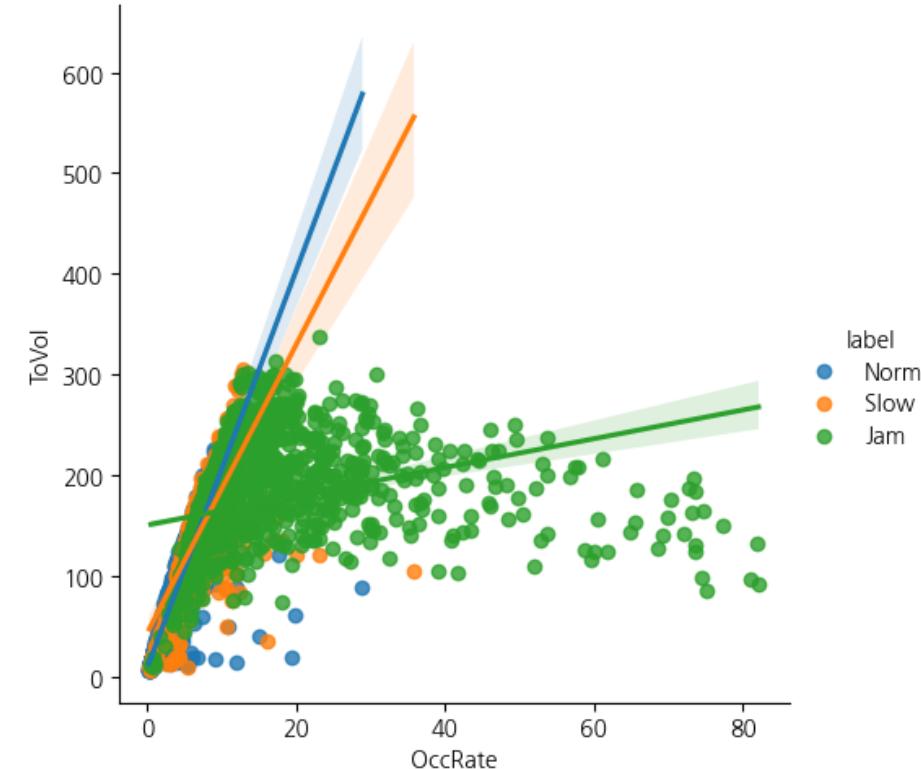
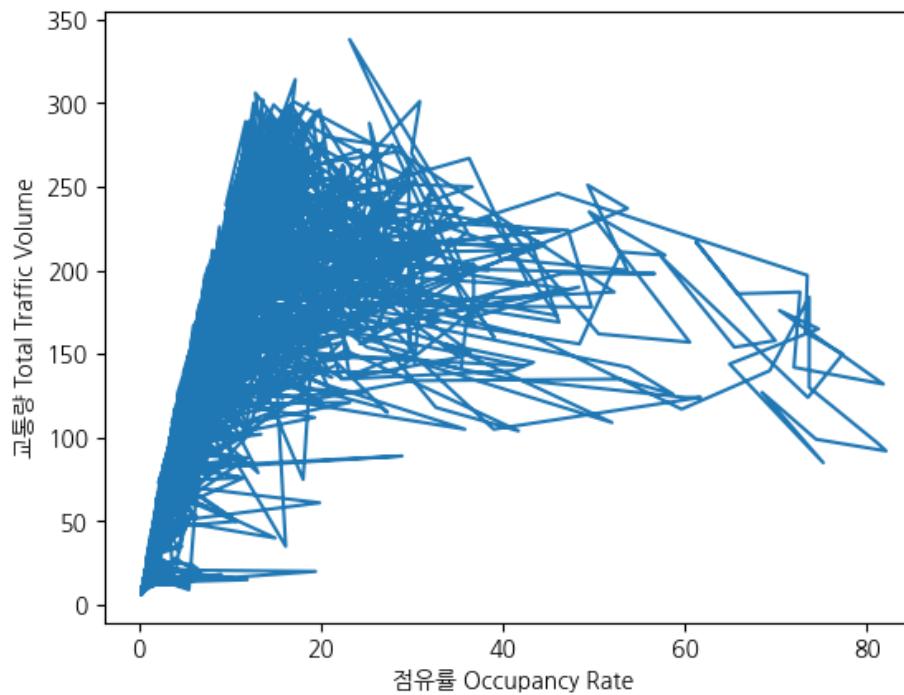


sns.lmplot(x="OccRate", y="ToVol", hue="label", data=df)



교통량과 점유률 관계

```
plt.plot(df["OccRate"],df["ToVol"])
plt.ylabel('교통량 Total Traffic Volume')
plt.xlabel('점유률 Occupancy Rate')
```



❖ 라벨을 정하기

- ✓ 지도학습을 위해서 데이터를 Feature와 Label로 나누자.
- ✓ 무엇이 Feature이며 무엇이 Label 인가?

❖ 라벨은 무엇으로 정하는게 좋은가?

- ✓ 속도
 - 혼잡(Jam) : 0~25km
 - 서행(Slow): 25~50km
 - 원활(Fast): 50~
- ✓ 점유률
- ✓ 교통량 (대형, 중형, 소형, 통합)

| Traffic-jam Level | Speed Performance Index | Traffic State | Traffic Context |
|-------------------|-------------------------|---------------|--|
| 1 | [0, 0.25] | Traffic jam | The average speed is the lowest; the road traffic state is very poor. |
| 2 | (0.25, 0.50] | Slow | The average speed is low; the road traffic state is poor. |
| 3 | (0.5, 0.75] | Moderate | The average speed is moderate; the road traffic state is a little congested. |
| 4 | (0.75, 1.0] | Fast | The average speed is high; the road traffic state is good. |

https://www.researchgate.net/figure/The-criterion-of-traffic-jam-level-and-the-speed-performance-index_tbl1_322834098

❖ 데이터 레이블 전략 : 속도로 해보자

[35] # 분류를 위한 클래스가 3개인 경우

```
num_classes = 3
class_labels= ['Jam', 'Slow', 'Fast']
```

[36] # 라벨을 위한 함수를 만들자

```
def speed_score(speed):
    if speed < 45:
        label = 'Jam'
    elif speed < 50:
        label = 'Slow'
    else :
        label = 'Fast'
    return label
```

[42] # speed로 라벨한 경우

```
df["label"] = df["Speed"].apply(lambda x: speed_score(x))
```

[45] df.head()

| | Date | ToVol | SmVol | MeVol | LaVol | Speed | OccRate | label |
|--|-----------------|-------|-------|-------|-------|-------|---------|--------|
| | 2017-04-02 0:00 | 43 | 34 | 9 | 0 | 50.3 | 1.90 | Normal |
| | 2017-04-02 0:05 | 45 | 32 | 13 | 0 | 58.9 | 1.84 | Normal |
| | 2017-04-02 0:10 | 46 | 34 | 12 | 0 | 50.6 | 1.87 | Normal |
| | 2017-04-02 0:15 | 45 | 36 | 9 | 0 | 50.9 | 1.72 | Normal |
| | 2017-04-02 0:20 | 27 | 13 | 13 | 1 | 62.2 | 1.12 | Normal |

다음 단계: df변수로 코드 생성

추천 차트 보기

[46] print('label:', df['label'].unique())

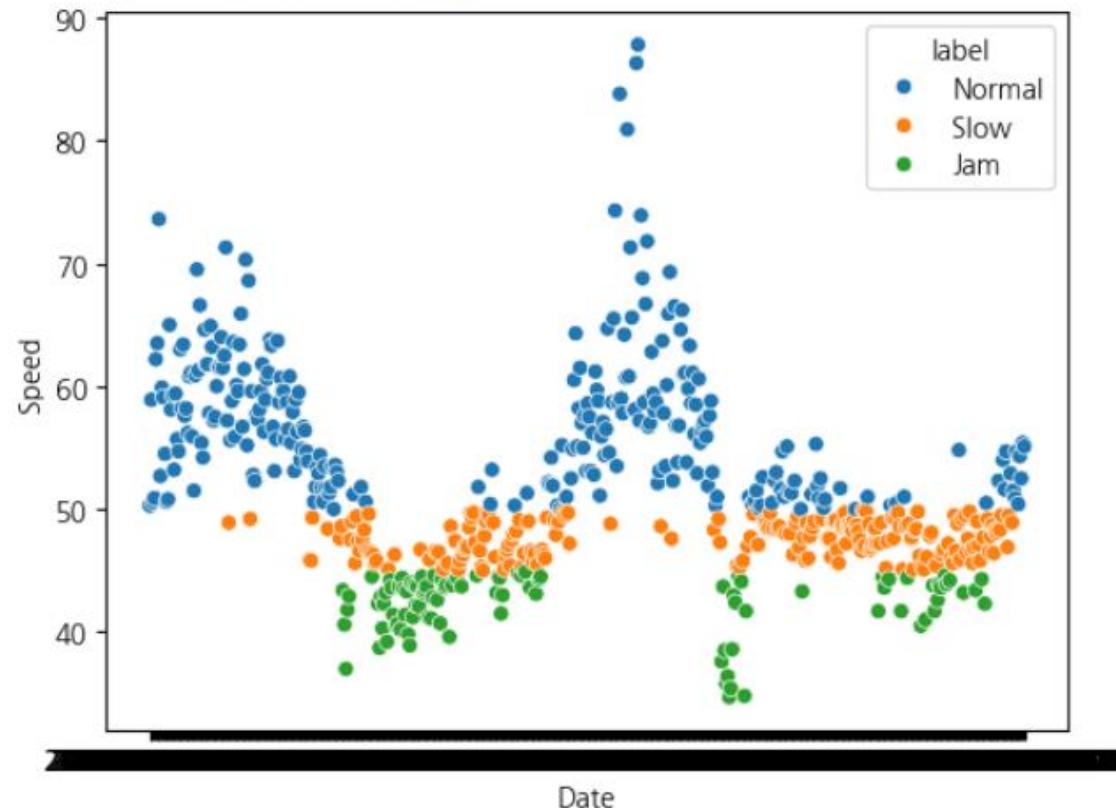
↳ label: ['Normal' 'Slow' 'Fast']

❖ Seaborn을 이용하여 가시화해보자

▶ # hue는 'label'에 따라서 색상으로 표시해준다.

```
sns.scatterplot(data=df[:nsize], x = 'Date', y = 'Speed', hue='label')
```

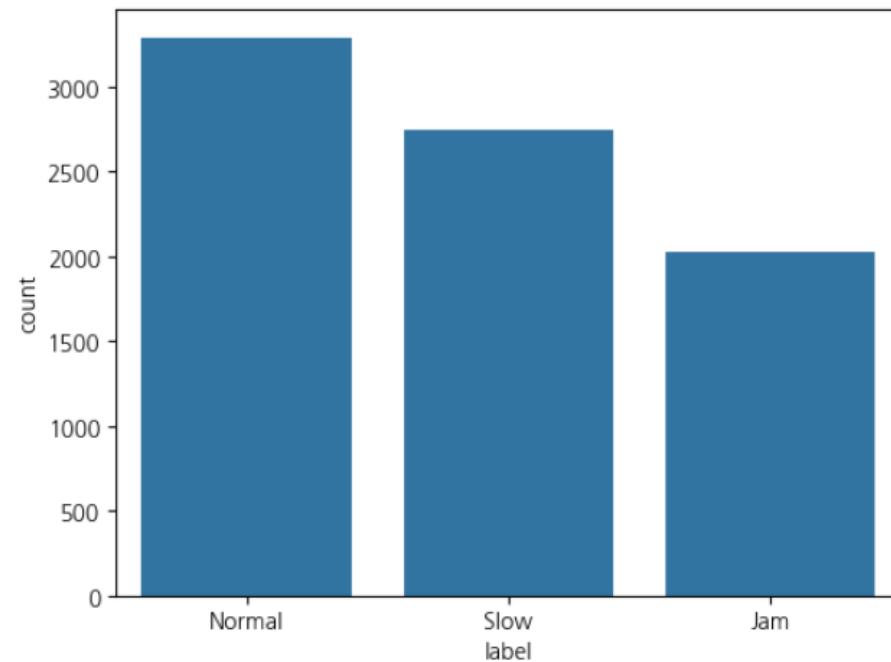
→ <Axes: xlabel='Date', ylabel='Speed'>



- ❖ 라벨의 개수가 균등하게 분포되었다면 베이스라인 모델로 도전해 보자

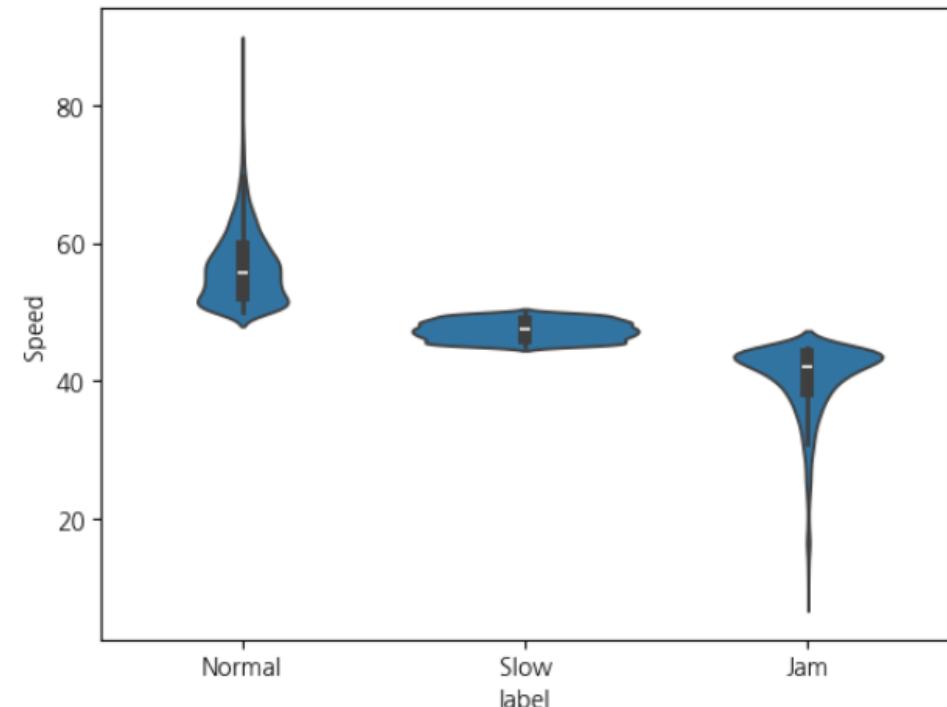
```
▶ sns.countplot(x='label',data=df)
df.loc[:, 'label'].value_counts()
```

```
label
Normal    3290
Slow      2749
Jam       2025
Name: count, dtype: int64
```



```
[53] sns.violinplot(x='label',y='Speed',data=df)
```

```
☞ <Axes: xlabel='label', ylabel='Speed'>
```

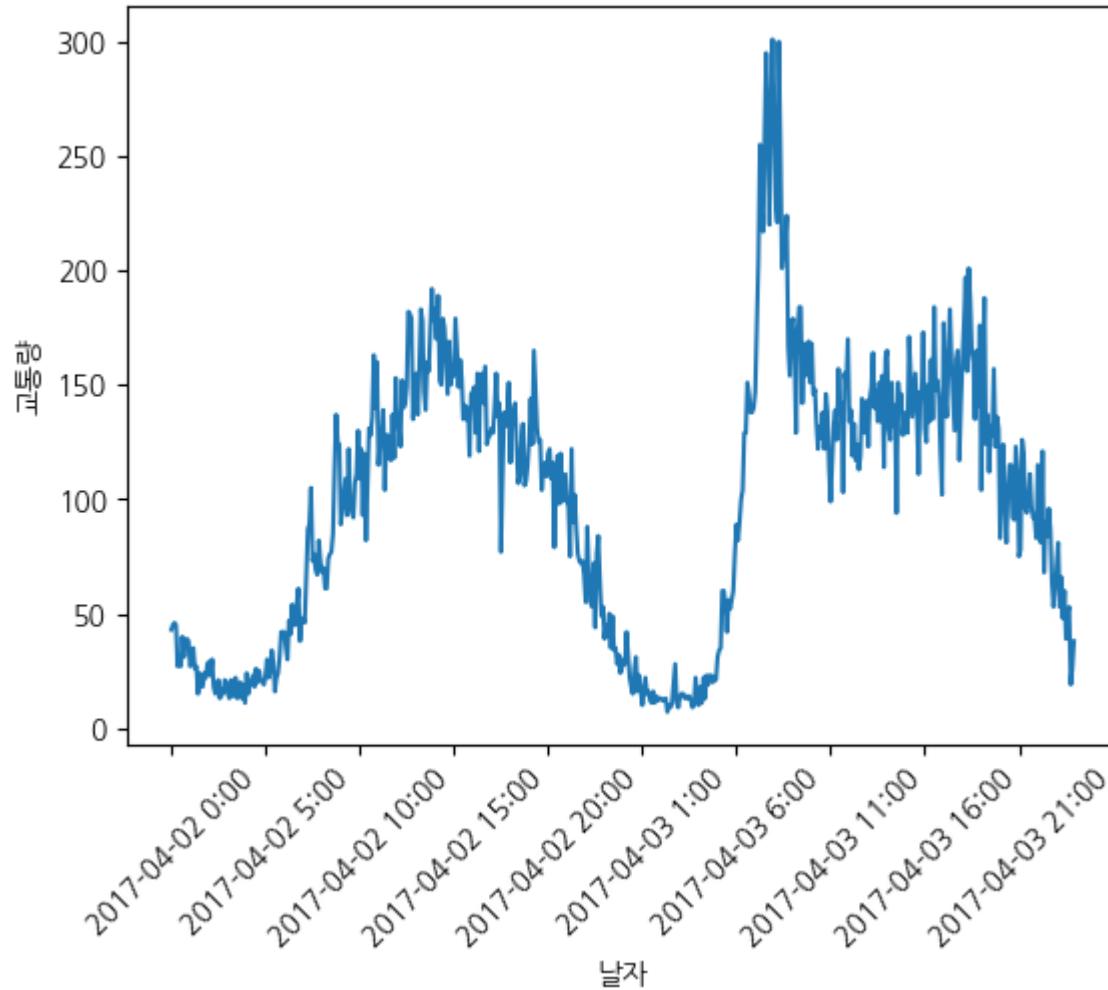


1-3

코랩에서 교통 데이터 가시화 실습

숙제 1 : 교통량으로 라벨 3개 만들어라

❖ 교통량으로 라벨을 전략



```
[35] # 분류를 위한 클래스가 3개인 경우  
num_classes = 3  
class_labels= ['Jam', 'Slow', 'Fast']
```

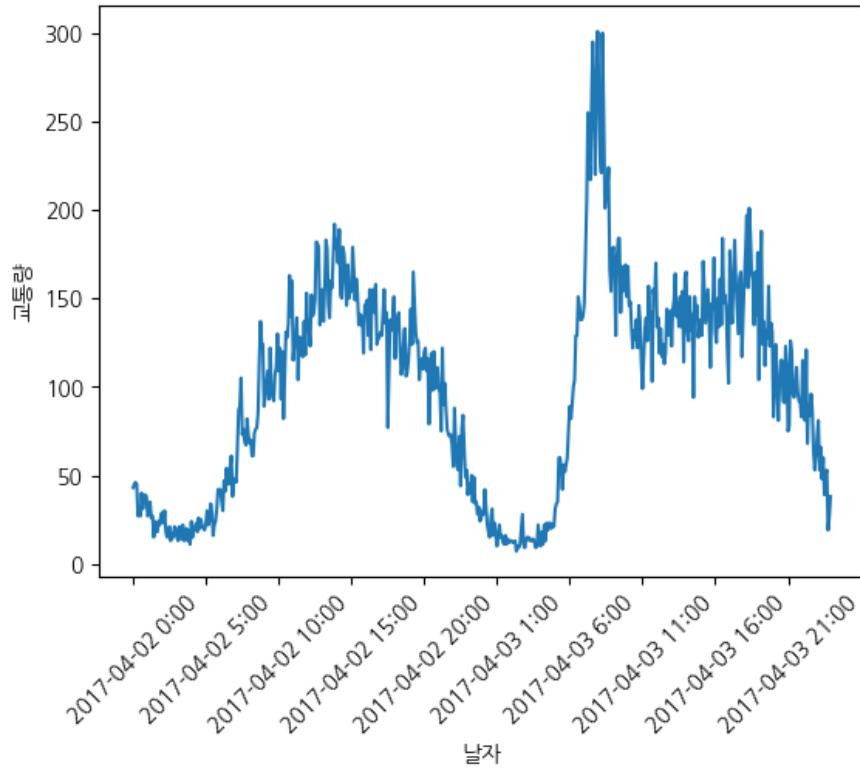
```
def volume_score(x):  
    if x < 100:  
        label = 'Fast'  
    elif x > 200:  
        label = 'Jam'  
    else :  
        label = 'Slow'  
    return label
```

숙제 2 : 시간 대 별로 라벨 3개 만들어라

- ❖ 시간대를 고려하여 라벨을 하여라
 - ✓ 3개의 라벨을 만들어라
- ❖ 교통량, 속도, 시간대 중에 어느 것이 최적인 것인가

[35] # 분류를 위한 클래스가 3개인 경우

```
num_classes = 3
class_labels= ['Jam', 'Slow', 'Fast']
```



```
def hour_score(hour):
    if ( hour < 7 and hour >0) or (hour <=24 and hour > 20):
        label = 'Fast'
    elif (hour >= 7 and hour <= 9) or (hour >= 18 and hour <= 20 ) :
        label = 'Jam'
    else :
        label = 'Slow'
    return label
```

실습 코드 : vds_baseline.ipynb (1/4)

```
[1] import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# 맷플랏립에서 한글을 사용하고자 한다면 아래와 같이
!pip install koreanize-matplotlib
import koreanize_matplotlib
df = pd.read_csv("https://raw.githubusercontent.com/hongsukyi/Lectures/main/data/vds16.csv")

df.tail(3)

df.info()
# 'Date'는 시간정보 얻어오기
# 'Date' 오브젝트를 시간으로 읽어오는 방법. 새로운 프레임워크에 저장
df2=df.copy()
df2.head()
df2['my_date']=pd.to_datetime(df['Date'], format='%Y-%m-%d %H:%M')
df2.info()

df=df.set_index('Date')
df.head()
```

실습 코드 : vds_baseline.ipynb (2/4)

```
### <a > Matplotlib을 이용하여 특성 가시화 </a>  
# 5분 단위로 가공된 데이터를 2일 간 그리고자 한다면  
nsize = 288*2
```

```
# x-축으로 날짜를 표시하고자 한다면  
plt.plot(df.index[0:nsize],df["ToVol"][0:nsize],label='ToVol')  
plt.xticks(np.arange(0,nsize,60),rotation=45)  
plt.xlabel('날자')  
plt.ylabel('교통량')  
plt.plot(df.index[0:nsize],df["OccRate"][0:nsize])  
plt.xticks(np.arange(0,nsize,72),rotation=45)  
plt.ylabel('점유율')  
# 속도 그리기  
plt.plot(df.index[0:nsize],df["Speed"][0:nsize])  
plt.xticks(np.arange(0,nsize,72),rotation=45)  
plt.ylabel('Speed')  
# 교통량과 점유률 관계  
plt.plot(df["OccRate"],df["ToVol"])  
plt.ylabel('교통량 Total Traffic Volume')  
plt.xlabel('점유률 Occupancy Rate')  
### <a > VDS 데이터 라벨링 전략 </a>  
# 분류를 위한 클래스가 3개인 경우  
num_classes = 3  
class_labels= ['Jam', 'Slow', 'Fast']  
# 라벨을 위한 함수를 만들자
```

실습 코드 : vds_baseline.ipynb (3/4)

```
def speed_score(speed):
    if speed < 45:
        label = 'Jam'
    elif speed < 50:
        label = 'Slow'
    else :
        label = 'Fast'
    return label
```

```
def volume_score(x):
    if x < 100:
        label = 'Fast'
    elif x > 200:
        label = 'Jam'
    else :
        label = 'Slow'
    return label
```

```
def hour_score(hour):
    if ( hour < 7 and hour >0) or (hour <=24 and hour > 20):
        label = 'Fast'
    elif (hour >= 7 and hour <= 9) or (hour >= 18 and hour <= 20 ) :
        label = 'Jam'
    else :
        label = 'Slow'
    return label
```

```
# speed로 라벨한 경우
df["label"] = df["Speed"].apply(lambda x: speed_score(x))
```

실습 코드 : vds_baseline.ipynb (4/4)

```
# speed로 라벨한 경우
df["label"] = df["Speed"].apply(lambda x: speed_score(x))

# 시간대로 라벨한 경우
# df["label"] = df["hour"].apply(lambda x: hour_score(x))
# 총 교통량으로 라벨한 경우
# df["label"] = df["ToVol"].apply(lambda x: volume_score(x))
df.head()
print('label:', df['label'].unique())
### <a > Seaborn을 이용한 상관관계 등 가시화 </a>
# hue는 'label'에 따라서 색상으로 표시해준다.
sns.scatterplot(data=df[:nsize], x = 'Date', y = 'Speed', hue='label')
sns.scatterplot(data=df[:576], x = 'Date', y = 'ToVol', hue='label')
sns.scatterplot(data=df[:576], x = 'Date', y = 'OccRate', hue='label')
df.hist('ToVol',bins=100)
sns.countplot(x='label',data=df)
df.loc[:, 'label'].value_counts()

sns.lmplot(x="OccRate", y="ToVol", hue="label", data=df)
sns.violinplot(x='label',y='Speed',data=df)
```

1-4

머신러닝을 적용해보자

❖ 머신러닝은 사용전에 데이터를 전처리 해야함

- ✓ 결손값, NaN, Null는 허용하지 않음
- ✓ 문자열 값을 입력으로 사용하지 않음
 - 숫자로 인코딩해서 변환 함
 - 원-핫 인코딩 (One-Hot Encoding)
 - 레이블 인코딩 (Label Encoding)
 - 텍스트 데이터는 텍스트 임베딩함 (Word2Vec)
- ✓ 피처는 카데고리형은 코드 값으로 변환
- ✓ 문자형 피처는 피처 벡터화(Vectorization)을 사용
- ✓ 불필요한 피처는 삭제 가능

❖ 전처리의 영향

- ✓ 전처리에 따라서 알고리즘의 복잡도가 정해지며, 예측 성능을 떨어뜨리기도 함

❖ 피처 스케일링(feature scaling)

- ✓ 서로 다른 변수의 값 범위를 일정하게 맞추는 작업

❖ 표준화(Standardization)

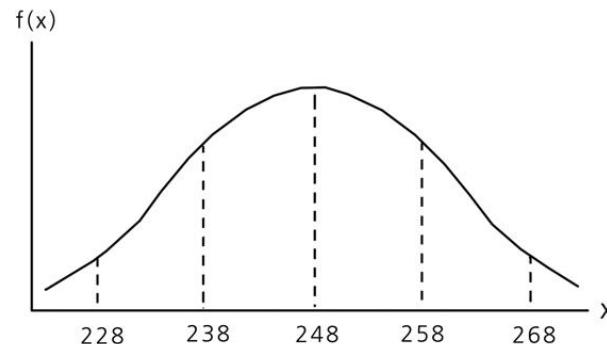
- ✓ 평균이 0이고 분산이 1인 가우시안 분포로 변환
- ✓ StandardScaler는 표준화를 쉽게 지원하는 클래스
 - SVM, 선형회귀, 로지스틱 회귀는 가우시안 분포를 가지고 있다고 가정하고 구현
 - 따라서 표준화를 적용하는 것이 성능향상 예측에 도움이 됨

❖ 정규화(Normalization)

- ✓ 데이터 값이 0과 1사이로 변환
- ✓ 음수 값이 있으면 (-1과 1 사이로 변환)
- ✓ 데이터 분포가 가우시안이 아닐 경우 적용

$$x_{i_new} = \frac{x_i - \min(x)}{\max(x) - \min(x)}$$

$$x_{i_new} = \frac{x_i - mean(x)}{stdev(x)}$$

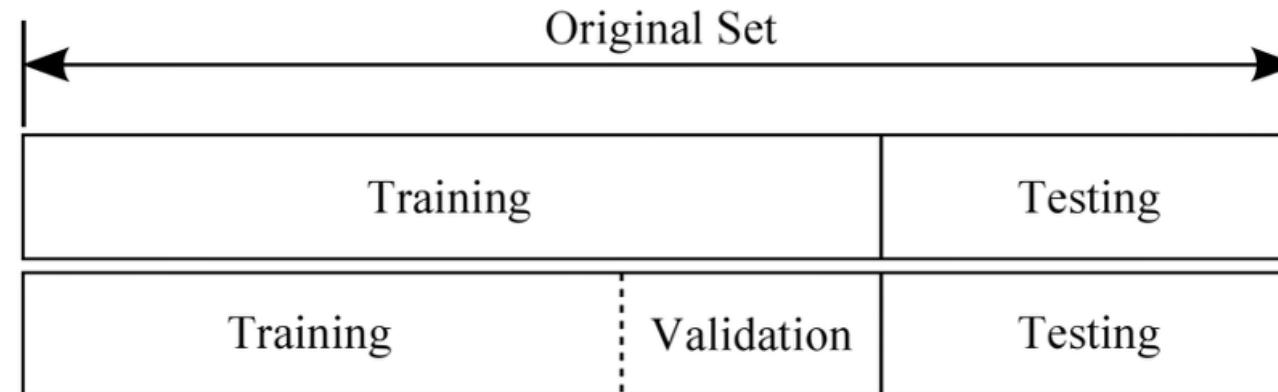


❖ 데이터 분할

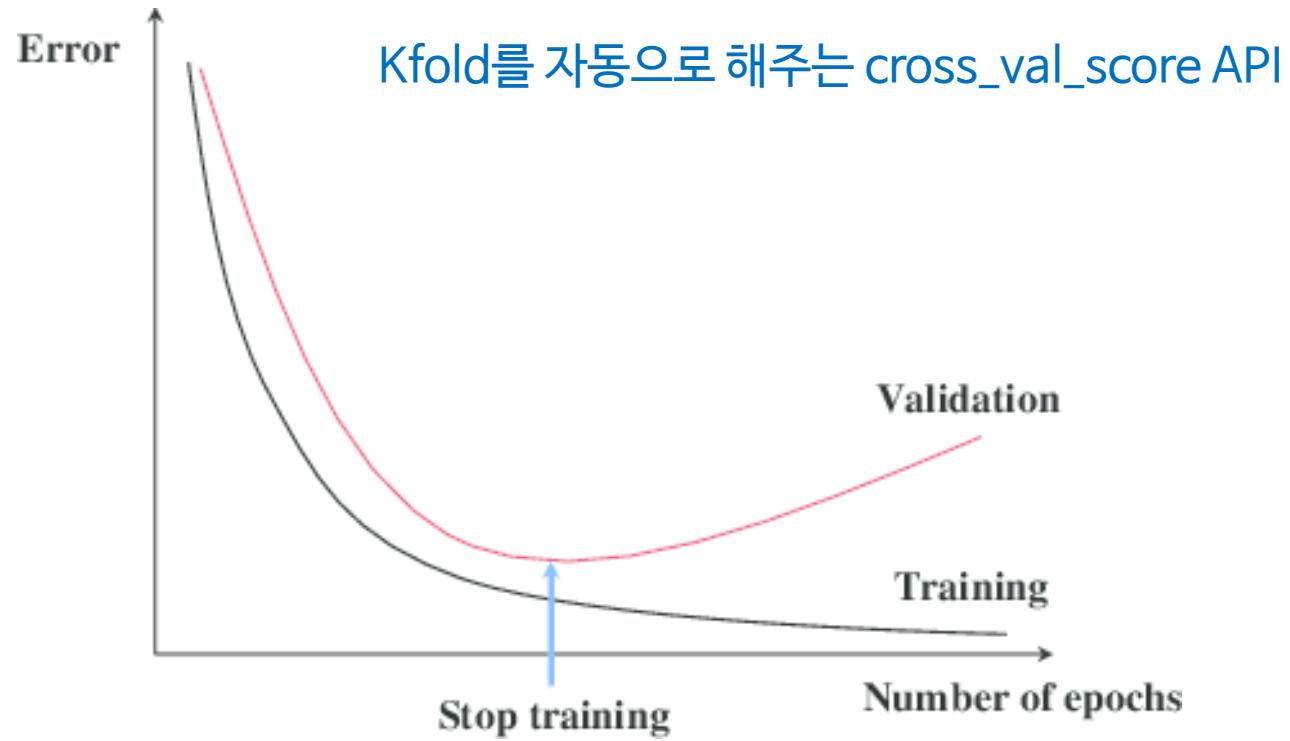
- ✓ 학습데이터
 - 학습데이터와 검증데이터로 나눔
- ✓ 테스트 데이터

❖ 머신러닝 및 딥러닝 모델 작성할때 Training 데이터 개수에 유의

- ✓ 같은 용어(Trainging) 데이터 사용에 혼란이 있을 수 있음.



과적합(Overfitting)과 검증



```
sklearn.model_selection.cross_val_score(estimator, X, y=None, groups=None, scoring=None,  
cv=None, n_jobs=None, verbose=0, fit_params=None, pre_dispatch='2*n_jobs', error_score=nan)
```

- ❖ 하이퍼 파라미터(Hyper-parameter)는 무엇인가?

- ✓ 참고, 신경망에서 Learning Rate, Batch size, # Layers 등

- ❖ Hyper-parameter tuning 방법

- ✓ Grid Search
 - ✓ Random Search
 - ✓ Bayesian Optimization
 - ✓ AutoML

- ❖ GridSearchCV API

- ✓ 균일한 그리드 사용
 - ✓ 촘촘한 파라미터의 최적값을 구함
 - ✓ 가장 기초적이며 많은 계산시간을 요구한다.
 - ✓ High Throughput 계산에 적합

❖ Confusion Matrix 이해

- ✓ 임신테스트 예제로 임신이면 +(1), 아니면 -(0)
 - TP (True Positive) : 임신 했을 때, 테스트 결과는 1
 - TN (True Negative) : 임신은 안 했을 때, 테스트 결과는 0
 - FP (False Positive) : 임신은 안 했을 때, 테스트 결과는 1 (에러)
 - FN (False Negative) : 임신 했는데, 테스트 결과는 0

❖ 민감도(Sensitivity)

- ✓ 병충해에 걸린 것을 양성으로 판정한다
- ✓ $TP/(TP+FN)$

❖ 특이도(Specificity)

- ✓ 민감도와 반대로 병충해에 걸리지 않은 것을 음성
- ✓ $TN/(TN+FP)$ 임



❖ 모델 평가 지표

- ✓ 정확도 (Accuracy)
- ✓ 정밀도 (Precision)
- ✓ 재현율(Recall)
- ✓ F1-스코어 (F1-score)

| | | Predicted | | |
|--|--------------|---|---|--|
| | | Negative (0) | Positive (1) | |
| Actual | Negative (0) | True Negative TN | False Positive FP (Type I error) | Specificity $= \frac{TN}{TN + FP}$ |
| | Positive (1) | False Negative FN (Type II error) | True Positive TP | Recall, Sensitivity, True positive rate (TPR) $= \frac{TP}{TP + FN}$ |
| Accuracy $= \frac{TP + TN}{TP + TN + FP + FN}$ | | Precision, Positive predictive value (PPV) $= \frac{TP}{TP + FP}$ | | F1-score $= 2 \times \frac{Recall \times Precision}{Recall + Precision}$ |

❖ 머신러닝 분류 모델을 위해 특성과 라벨을 정하자

▼ 특성과 라벨에 대하여 논의사항

- 특성 중에서 $ToVol=LaVol+MeVol+SmVol$ 이다
- Speed를 이용하여 label을 만들었다.
- Speed를 특성 포함하는 것이 올바른가? 아니면 제외하는 것이 올바른 것인가?

[12] # 라벨을 'Speed'에서 만들어서 'Speed'는 특성에서 제외하자.

```
X = df[['ToVol', 'LaVol', 'MeVol', 'SmVol', 'OccRate']]  
y = df['label']
```

▼ sklearn.preprocessing에 LabelEncoder를 이용하자

- 라벨(y)는 텍스트로 되어 있어서 숫자로 바꾸자
- 라벨 개수 num_classes에 맞도록 LabelEncoder()로 인코딩 해보자

❖ 표준화

```
[13] from sklearn.preprocessing import LabelEncoder, StandardScaler
```

```
[14] encoder = LabelEncoder()  
y = encoder.fit_transform(y)
```

```
[15] y
```

```
→ array([0, 0, 0, ..., 0, 0, 0])
```

- ▼ X_train을 위하여 sklearn.preprocessing에 StandardScaler를 이용하자.

```
[16] X[:4]
```

| | ToVol | LaVol | MeVol | SmVol | OccRate | grid | bar |
|---|-------|-------|-------|-------|---------|------|-----|
| 0 | 43 | 0 | 9 | 34 | 1.90 | | |
| 1 | 45 | 0 | 13 | 32 | 1.84 | | |
| 2 | 46 | 0 | 12 | 34 | 1.87 | | |
| 3 | 45 | 0 | 9 | 36 | 1.72 | | |

❖ 결정경계

```
[17] scaler = StandardScaler()  
X_scaled = scaler.fit_transform(X)
```

```
[18] X_scaled[:4]
```

```
→ array([[-1.05487777, -0.75688677, -1.09793441, -0.96910406, -0.6331216 ],  
        [-1.02360358, -0.75688677, -0.88829029, -1.01183983, -0.6420243 ],  
        [-1.00796648, -0.75688677, -0.94070132, -0.96910406, -0.63757295],  
        [-1.02360358, -0.75688677, -1.09793441, -0.92636829, -0.6598297 ]])
```

▼ 의사결정 경계를 정하기 위해 교통량-점유률 경계를 알아보자

- X1 : 점유률과 교통량 관계
- X2 : 속도와 교통량 관계

```
[19] X1=df[['OccRate', 'ToVol']].values
```

```
X2=df[['Speed', 'ToVol']].values
```

```
▶ import random
  from sklearn.model_selection import train_test_split
  from sklearn.linear_model import LogisticRegression
  from sklearn.svm import SVC
  from sklearn.naive_bayes import GaussianNB
  from sklearn.tree import DecisionTreeClassifier
  from sklearn.ensemble import RandomForestClassifier
  from sklearn.neighbors import KNeighborsClassifier

  from sklearn.metrics import classification_report, accuracy_score, confusion_matrix

  from mlxtend.plotting import plot_decision_regions
  #!pip install mlxtend
```

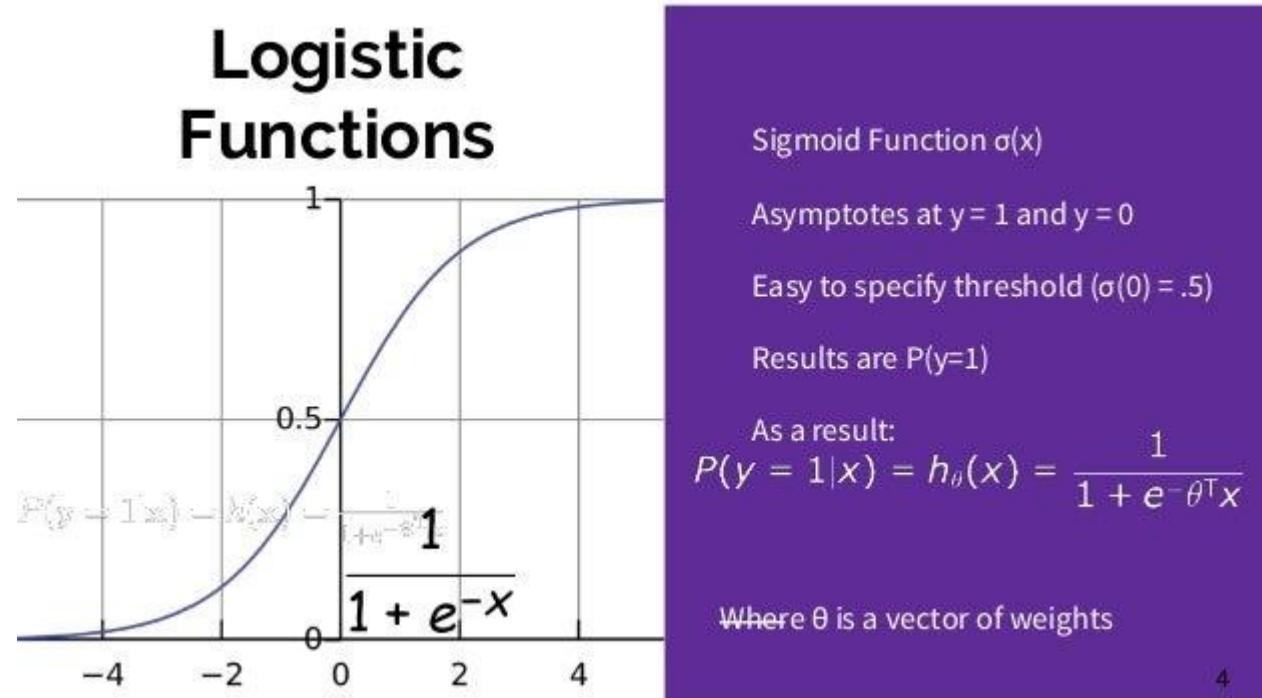
```
[21] X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=0)
```

```
[22] print(X_train.shape, y_train.shape)
    print(X_test.shape, y_test.shape)
```

```
→ (6451, 5) (6451,)
    (1613, 5) (1613,)
```

Logistic regression은 분류 모델로 사용가능

- ❖ Logistic regression은 라벨이 범주형일때 사용한다.
 - ✓ 시그모이드 함수와 같다.



Logistic regression

```
[24] # LogisticRegression
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
lr.fit(X_train, y_train)
y_pred = lr.predict(X_test)

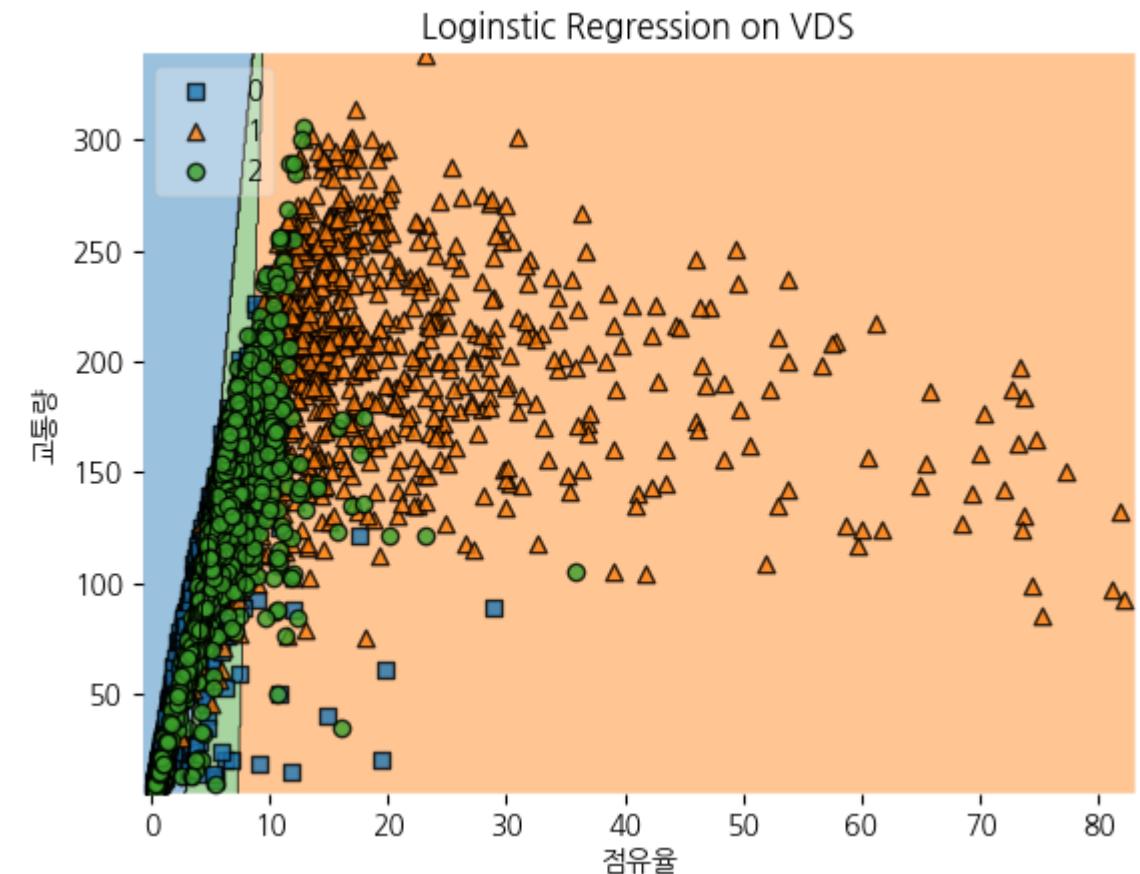
# Summary of the predictions made by the classifier
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
# Accuracy score
acc_lr=accuracy_score(y_pred,y_test)
print('accuracy is',accuracy_score(y_pred,y_test))
```

| | precision | recall | f1-score | support |
|--------------------------------|-----------|--------|----------|---------|
| 0 | 0.88 | 0.84 | 0.86 | 665 |
| 1 | 0.82 | 0.62 | 0.71 | 398 |
| 2 | 0.63 | 0.77 | 0.69 | 550 |
| accuracy | | | | 0.76 |
| macro avg | | | | 0.75 |
| weighted avg | | | | 0.76 |
| [[559 3 103] | | | | |
| [5 247 146] | | | | |
| [73 52 425]] | | | | |
| accuracy is 0.7631742095474272 | | | | |

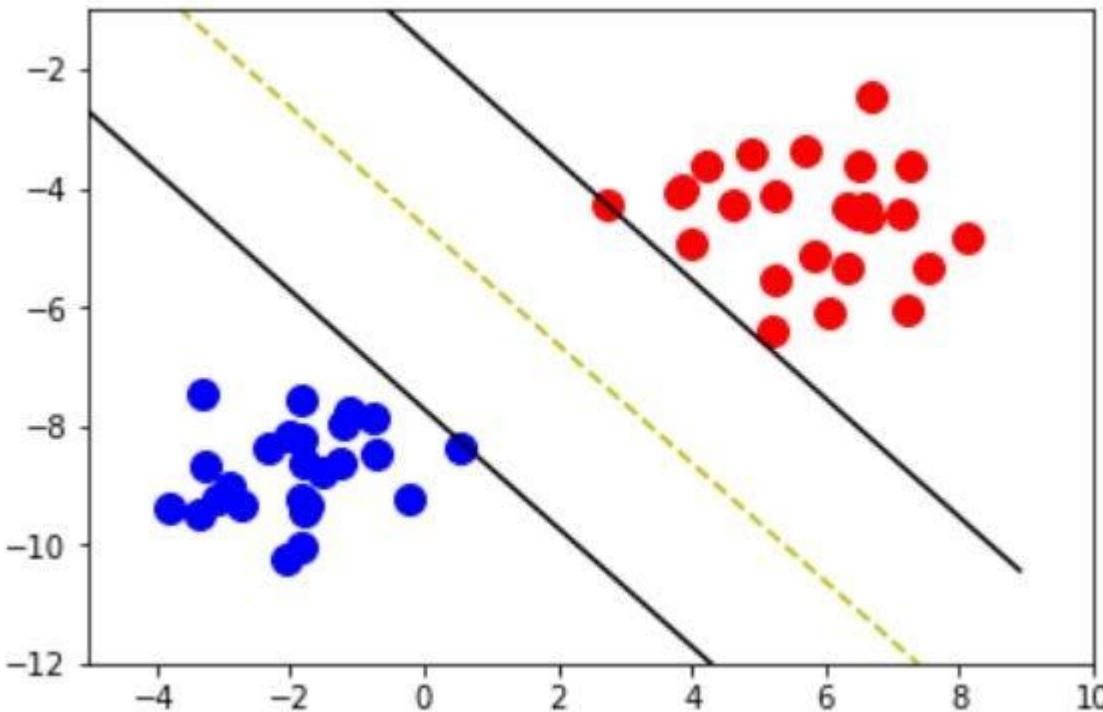
Logistic regression

```
▶ # Training a classifier
lr = LogisticRegression()
lr.fit(X1, y)
plot_decision_regions(X1, y, clf=lr, legend=2)

# Adding axes annotations
plt.xlabel('점유율')
plt.ylabel('교통량')
plt.title('Loginstic Regression on VDS')
plt.show()
```



서포터 벡터 머신을 이용한 분류

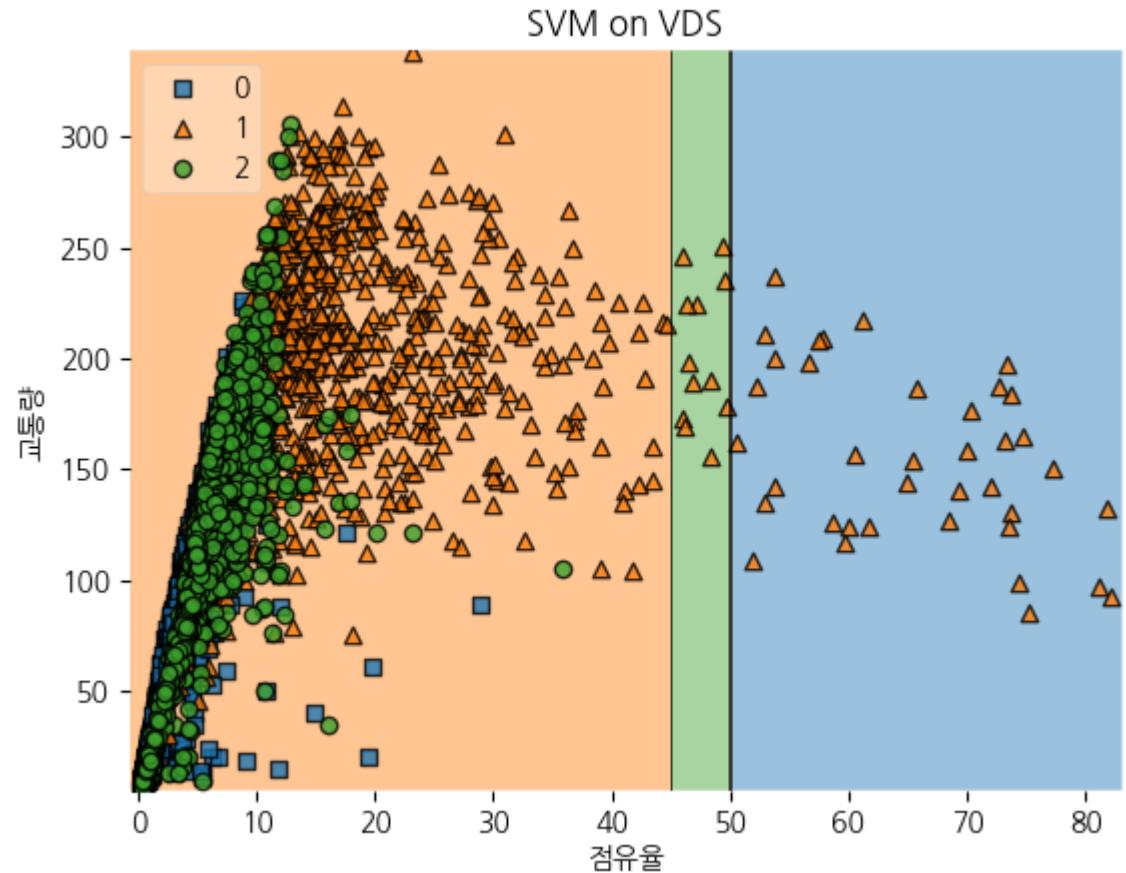


```
| Model = SVC()  
| Model.fit(X_train, y_train)  
| y_pred = Model.predict(X_test)  
  
| print(classification_report(y_test, y_pred))  
| print(confusion_matrix(y_test, y_pred))  
| acc_svc=accuracy_score(y_pred,y_test)  
| print('accuracy is',accuracy_score(y_pred,y_test))
```

```
[[534  1 130]  
 [ 6 236 156]  
 [ 61  44 445]]  
accuracy is 0.7532548047117174
```

서포터 벡터 머신을 이용한 분류

```
[28] svm = SVC()  
svm.fit(X1, y)  
plot_decision_regions(X1, y, clf=lr, legend=2)  
  
plt.xlabel('점유율')  
plt.ylabel('교통량')  
plt.title('SVM on VDS')  
plt.show()
```



1-5

의사결정나무

결정 트리?**

- 결정 트리는 분류 문제를 해결하기 위하여 가장 많이 사용된 지도(supervised) 기계학습 알고리즘의 한 종류
- 물론 회귀(Regression)에도 사용 가능함
- 결정 트리의 결과는 나무 같은 간단한 그래프으로 설명이 되어짐
- 그래서 블랙박스가 아닌 실제로 일어난 일들을 볼수 있음.

결정 트리 용어

루트 노드 - 부모 노드로 알려짐. 데이터셋의 길이와 모든 가지가 여기서 출발 함.

브랜치 - 가지는 루트 노드의 서브 노드로 나눈다. 물론 데이터도 나눔

결정 노드 - 결정노드들은 서브노드를 더 깊게 나눔. 더 이상 나눌 것이 없으면 리프노드임.

리프노드: 더이상 나눌수 없는 노드.

알고리즘 - '지니'(Gini)는 경제학에서 불평등 지수를 나타냄. 0이 가장 평등하며, 1이 불평등하다.

- 즉, 데이터가 다양한 값을 가질 경우 평등(0)하며
- 특정 값으로 쓸릴경우 불평등(1에 가까움)
- 엔트로피는 무질서도를 나타내며, 무질서도(혼잡도)는 서로 다른 값이 섞여 있으면 높다. 혼잡도가 높으면 1, 적으면 0

$$\text{지니 지수: } G = 1 - \sum_i^c p_i^2 , \quad 0 \leq G \leq 1/2$$

$$\text{엔트로피 지수: } E = - \sum_i^c p_i \log_2 p_i , \quad 0 \leq E \leq 1$$

- ❖ 기본적으로 gini를 사용 Entropy (엔트로피) 불순도를 사용해도됨

- ✓ 엔트로피는 문자의 무질서도를 측정하는 것 (열역학의 개념)
 - 문자가 안정되고 질서 정연하면 엔트로피는 0에 가까움
 - 정보이론에서 모든 메시기가 동일할때 엔트로피는 0에 가까움

- ❖ 머신러닝에서

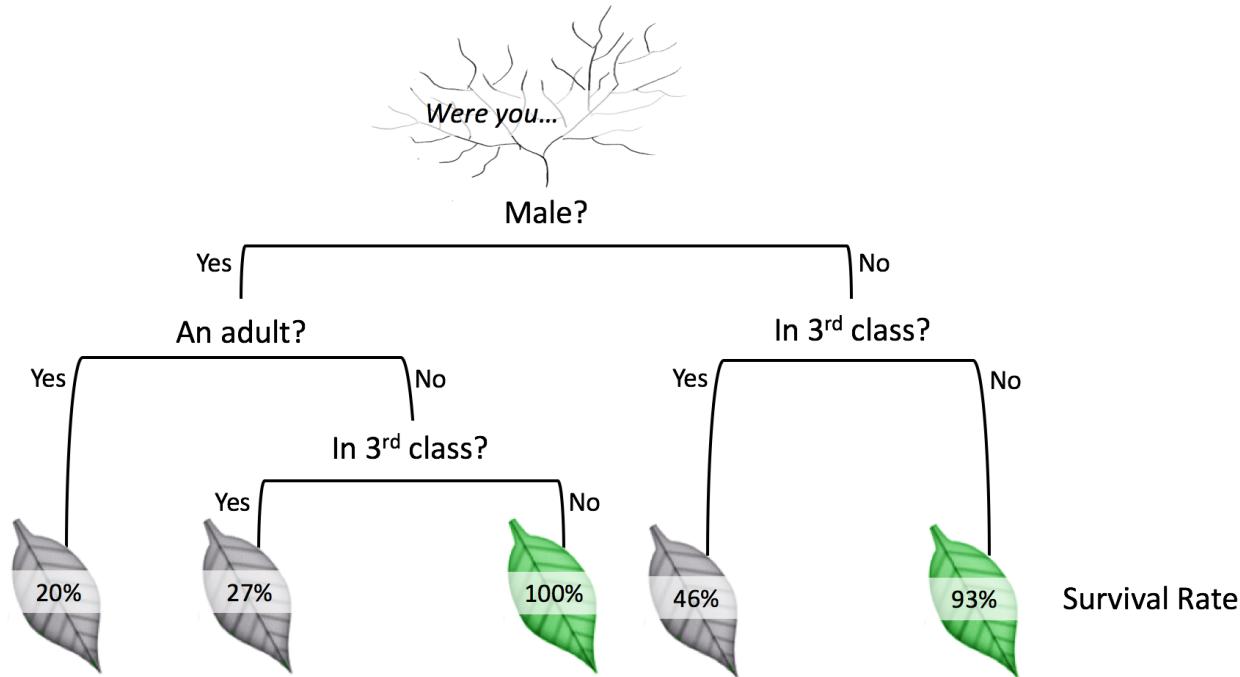
- ✓ 어떤 세트가 한 클래스의 샘플만 있는 경우는 엔트로피가 0이다.
- ✓ Gini와 엔트로피 중 어떤 것을 사용해도 실제로는 큰 차이가 없다.
- ✓ Gini 불순도가 계산이 조금 더 빠르기 때문에 기본값으로 좋다.

$$H_i = - \sum_{k=1}^n p_{i,k} \log_2(p_{i,k})$$

Decision Tree: 의사결정나무

- ❖ 의사결정나무는 의사결정을 도와주는 툴로 나무 같은 그래프 모델을 사용한다.

- ✓ 20고개 같은 방식
- ✓ 각각의 잎 노드(leaf)는 클래스 라벨(범주)를 나타낸다.

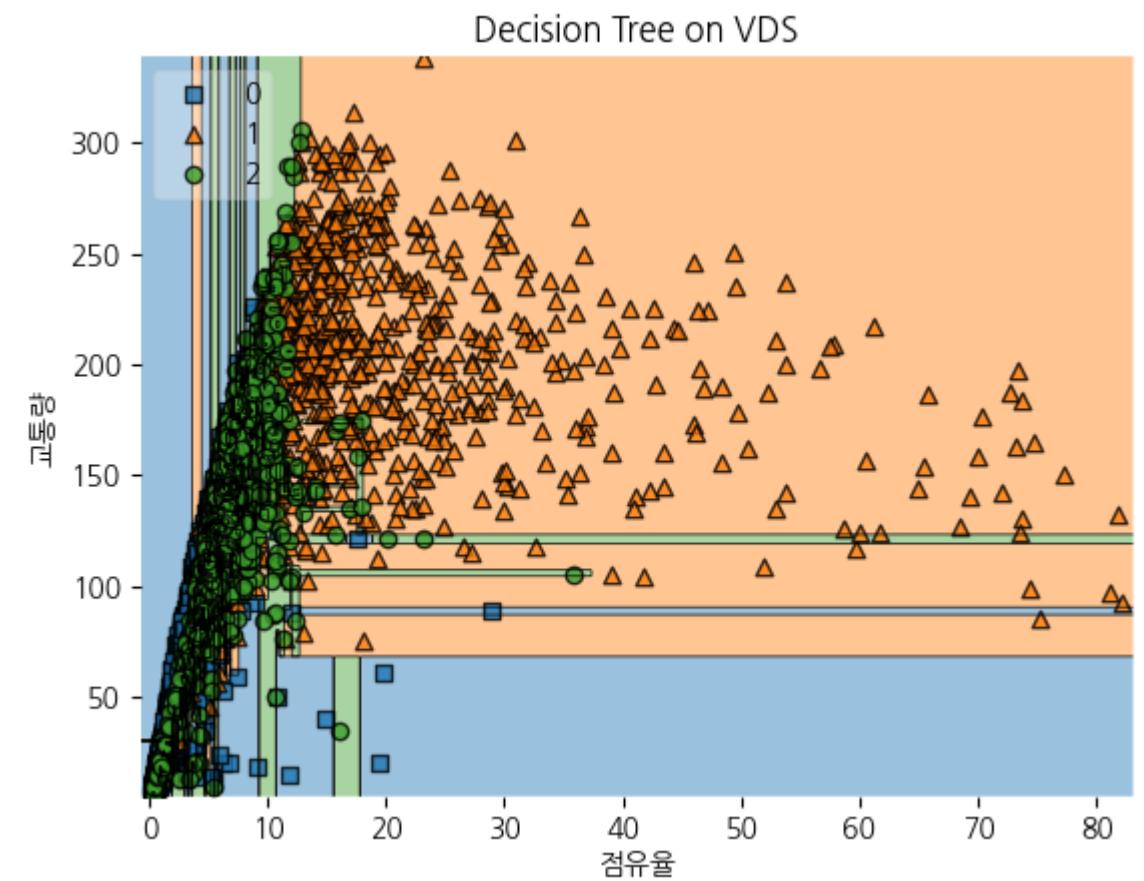


Decision Tree: 의사결정나무

```
▶ tree = DecisionTreeClassifier()  
tree.fit(X_train, y_train)  
y_pred = tree.predict(X_test)  
  
print(classification_report(y_test, y_pred))  
print(confusion_matrix(y_test, y_pred))  
acc_dt=accuracy_score(y_pred,y_test)  
print('accuracy is',accuracy_score(y_pred,y_test))
```

| | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0 | 0.82 | 0.81 | 0.82 | 665 |
| 1 | 0.66 | 0.72 | 0.69 | 398 |
| 2 | 0.59 | 0.56 | 0.57 | 550 |
| accuracy | | | 0.70 | 1613 |
| macro avg | 0.69 | 0.70 | 0.69 | 1613 |
| weighted avg | 0.70 | 0.70 | 0.70 | 1613 |

```
[[540 14 111]  
 [ 7 285 106]  
 [112 130 308]]  
accuracy is 0.7024178549287042
```



램덤포레스트

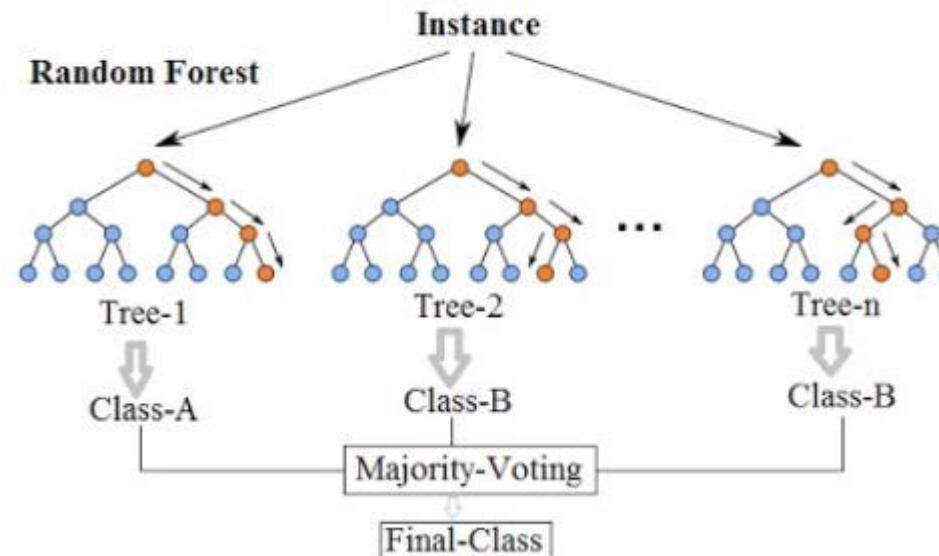
❖ 앙상블 학습 개요

- ✓ 여러 개의 분류기를 생성하고 그 예측을 결합하여 보다 정확한 최종 예측을 도달함
- ✓ 대중의 지혜(wisdom of the crowd)
- ✓ 단일 분류기 보다 신뢰성이 높은 예측 값을 얻는 것이 핵심
 - 앙상블이 뛰어난 분야는 램덤포레스트와, 그레이언트 부스팅
 - XGboost, 훨씬 빠른 LightGBM, 메타 모델을 수립하는 스택킹(Stacking)

❖ 양상블 학습 유형

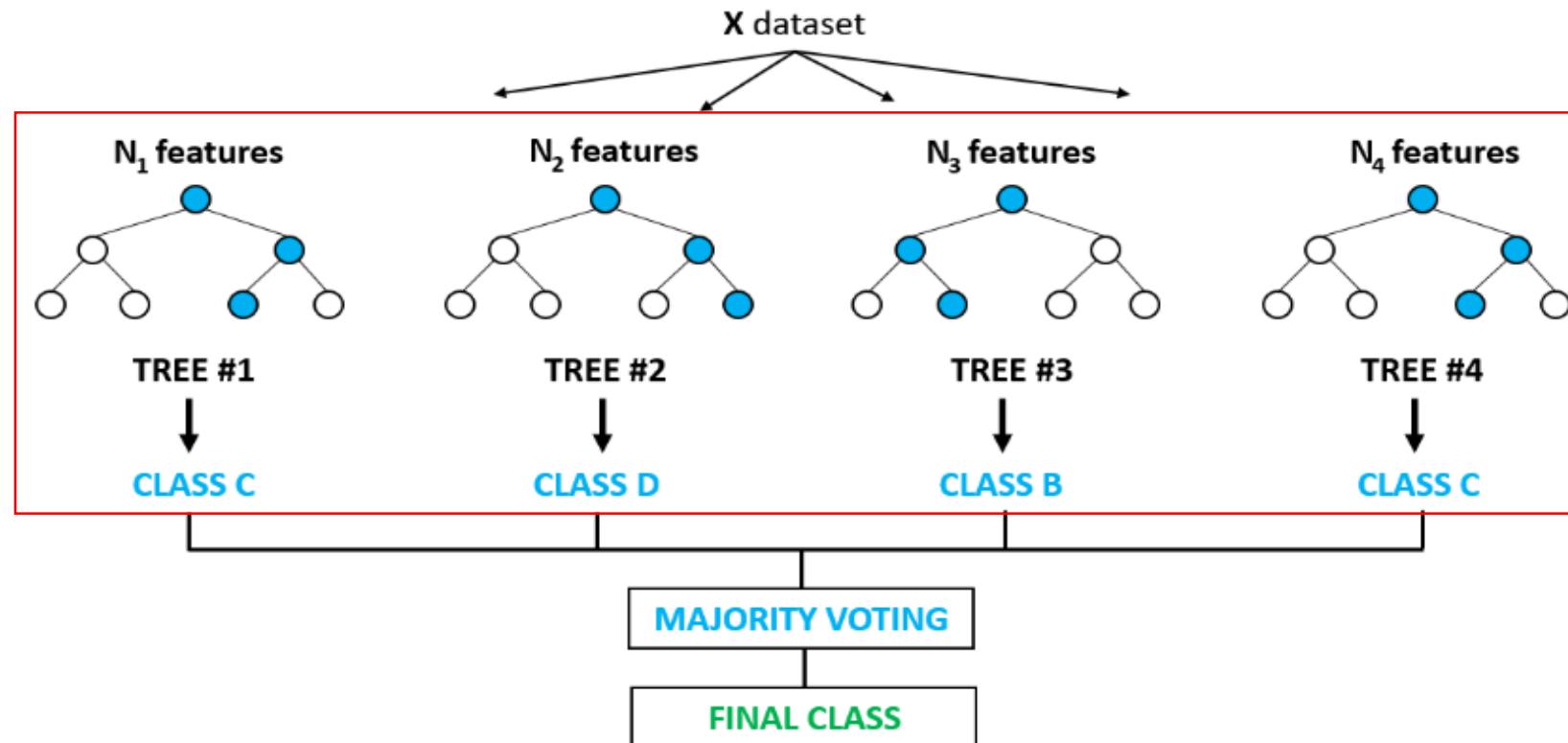
- ✓ 보팅(Voting) : 여러 개의 분류기가 투표를 통해 최종 예측
 - 서로 다른 알고리즘(knn, svm, kmeans)을 가진 분류기를 결합하는 방식
- ✓ 배깅(Bagging) : 여러 개의 분류기가 투표를 통해 최종 예측
 - 대표적인 배깅 방식은 램덤 포레스트 알고리즘이다.
 - 분류기는 같은 유형의 알고리즘이지만, 데이터 샘플링을 서로 다르게 학습
 - 배깅은 교차 검증과 달리 데이터 셋트 간의 중첩을 허용한다.

Random Forest Simplified



- ❖ 랜덤포레스트는 의사결정나무의 앙상블 집합으로

- ✓ 훈련 세트로부터 무작위로 각기 다른 서브셋(포레스트)을 만들어 일련의 결정 트리 분류기를 훈련시킴.
- ✓ 의사결정 나무들 투표로 테스트 라벨의 최종 범주를 결정한다.



Random Forest 분류

```
[33] Model=RandomForestClassifier(max_depth=2)
    Model.fit(X_train,y_train)
    y_pred=Model.predict(X_test)

    # Summary of the predictions made by the classifier
    print(classification_report(y_test,y_pred))
    print(confusion_matrix(y_pred,y_test))
    #Accuracy Score
    acc_rf=accuracy_score(y_pred,y_test)
    print('accuracy is ',accuracy_score(y_pred,y_test))
```

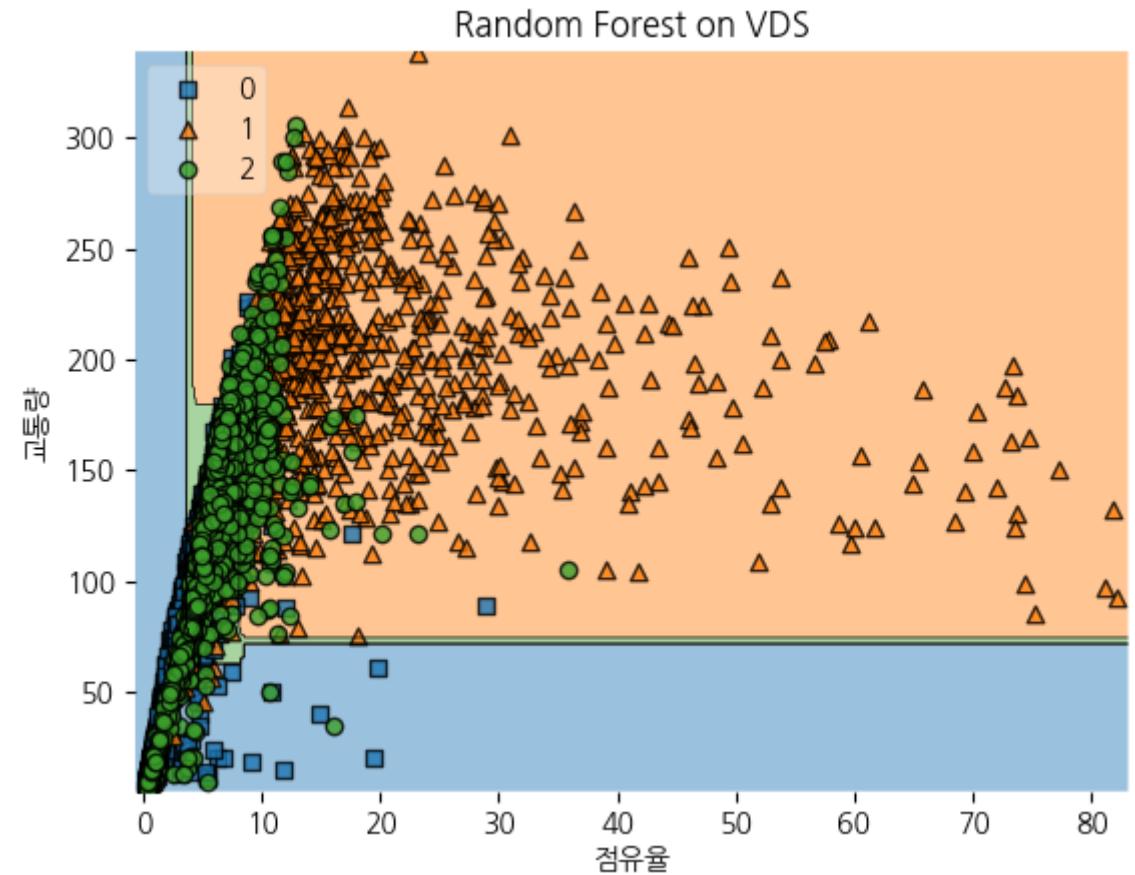
| | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0 | 0.90 | 0.76 | 0.82 | 665 |
| 1 | 0.73 | 0.63 | 0.67 | 398 |
| 2 | 0.58 | 0.75 | 0.66 | 550 |
| accuracy | | | 0.72 | 1613 |
| macro avg | 0.74 | 0.71 | 0.72 | 1613 |
| weighted avg | 0.75 | 0.72 | 0.73 | 1613 |

```
[[504  5 53]
 [ 9 249 82]
 [152 144 415]]
accuracy is  0.7241165530068195
```

Random Forest 분류

```
forest = RandomForestClassifier(max_depth=2)
forest.fit(X1, y)
plot_decision_regions(X1, y, clf=forest, legend=2)

# Adding axes annotations
plt.xlabel('점유율')
plt.ylabel('교통량')
plt.title('Random Forest on VDS')
plt.show()
```



1-6

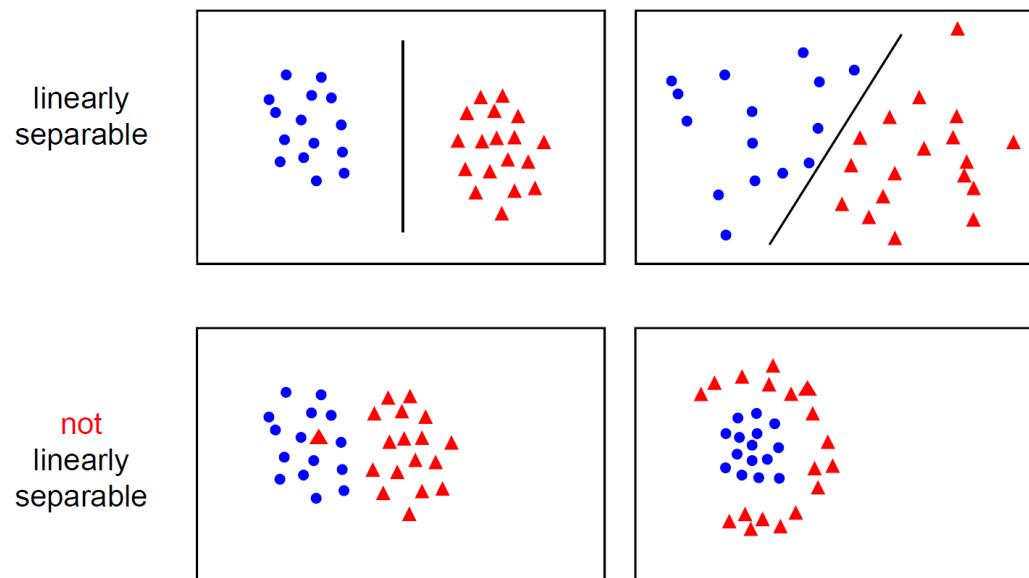
SVM, KNN, GBM

❖ SVM은 통계 학습 이론과 관련

- ✓ SVM은 1992년에 처음 소개됨
 - SVM은 손글씨 인식 문제에서 성능을 보임
 - SVM의 테스트 오차율은 1.1%로, 신경망 LeNet 4과 비슷한 성능

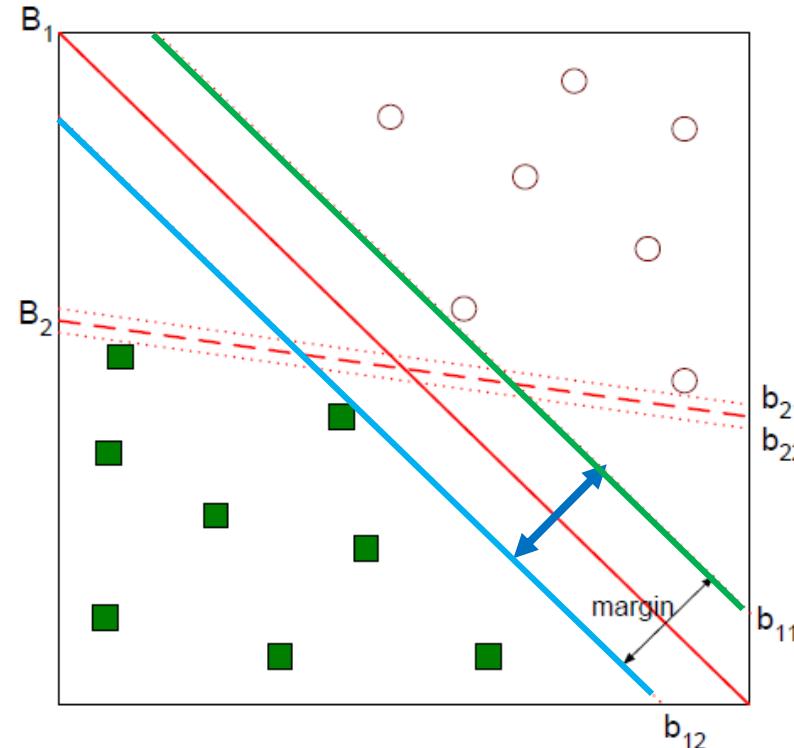
❖ 결정 경계 (decision boundary)

- ✓ 2개의 범주를 나누는 선형 분리가 가능할 때, 결정 경계가 가능하다.
- ✓ 선형분리와 비선형 분리 예제



마진(margin)이란 무엇인가?

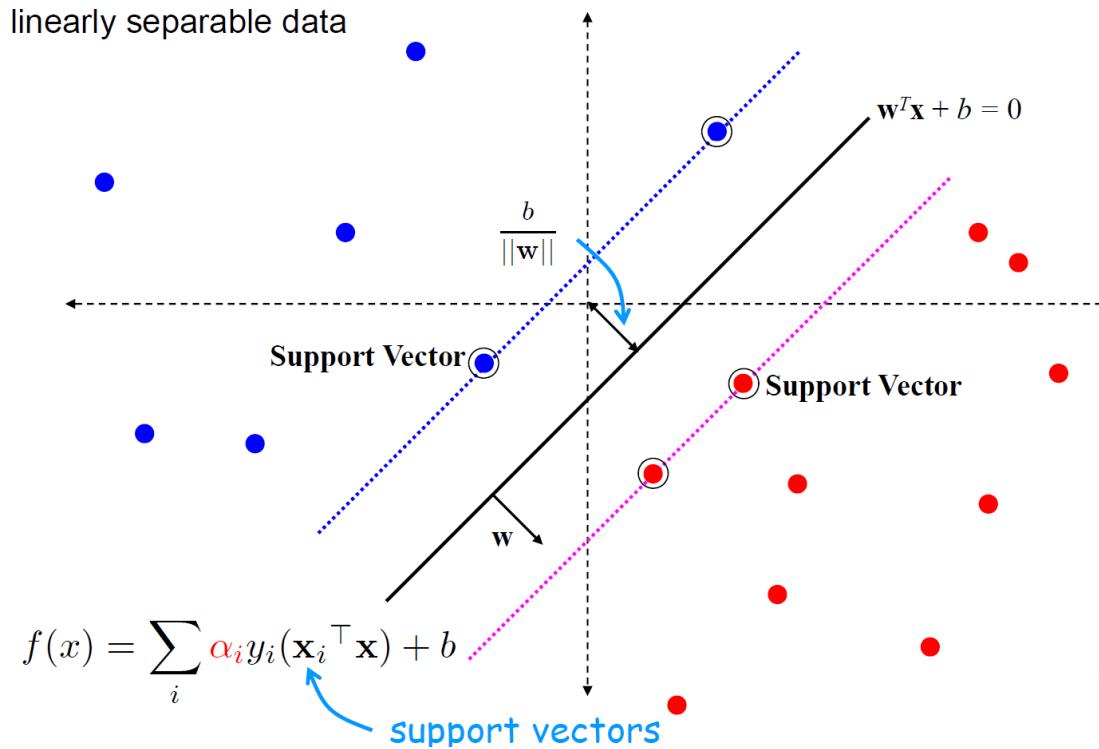
- ❖ 결정 경계를 정하기 위해 마진(margin)을 도입하자.
 - ✓ 마진은 두 경계면 사이의 거리이며, 이 거리가 최대일때 좋은 결정 경계를 얻음.
 - ✓ 마진은 결정 경계 두 샘플 거리가 최대가 되도록 정한다



서포트 벡터 머신 (SVM)의 해

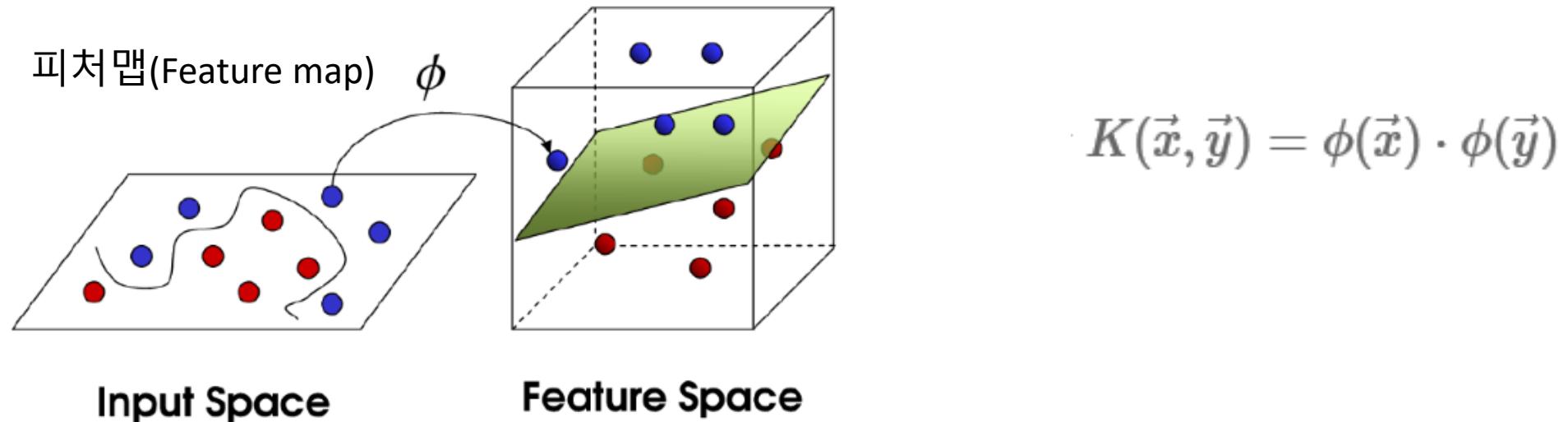
❖ SVM은 결정경계(최적의 라인, 초평면)를 만든다

- ✓ 초평면에 가장 가까이 있는 벡터를 지지벡터(Support Vector)라 한다.
- ✓ 알파는 해당 벡터(x)가 경계선을 정하는 샘플이라는 뜻



$$\begin{aligned} \text{Margin} &= \text{distance}(x^+, x^-) \\ &= \|x^+ - x^-\|_2 \\ &= \|x^- + \lambda w - x^-\|_2 \\ &= \|\lambda w\|_2 \\ &= \lambda \sqrt{w^T w} \\ &= \frac{2}{w^T w} \sqrt{w^T w} \\ &= \frac{2}{\|w\|_2} \sqrt{w^T w} \\ &= \frac{2}{\|w\|_2} \quad \lambda = \frac{2}{w^T w} \end{aligned}$$

- ❖ 비선형 SVM은 선형으로 분리 되지 않을 경우 적용
 - ✓ 입력 자료의 다차원 공간상으로의 맵핑(mapping) 기법을 사용하여 비선형분류도 효율적으로 수행
- ❖ 피처맵
 - ✓ 고차원 매핑과 내적을 한방에 할 수는 커널 트릭(Kernel trick)
 - ✓ 샘플 공간에서 선형적으로 나눌 수 있는 공간에 샘플을 보내주고 SVM을 적용함



Soft Vector Machine

```
[27] Model = SVC()
    Model.fit(X_train, y_train)
    y_pred = Model.predict(X_test)

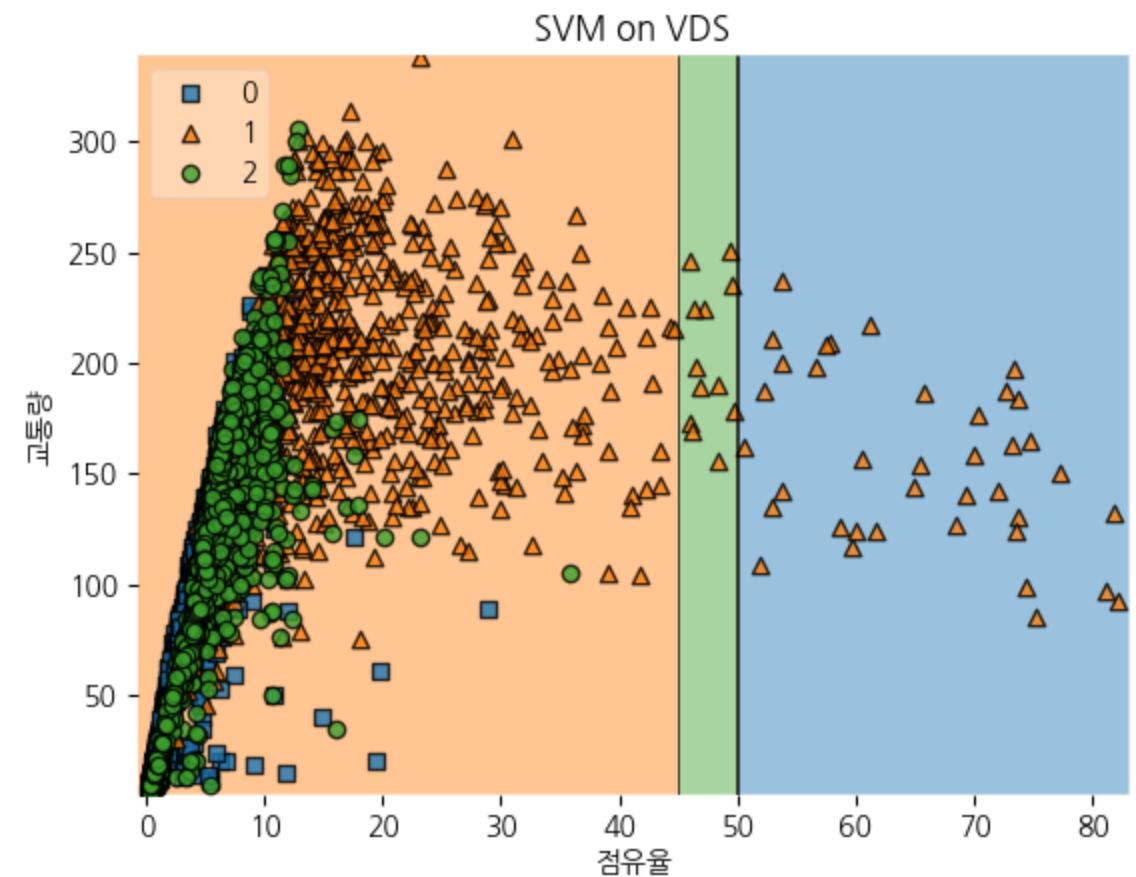
    print(classification_report(y_test, y_pred))
    print(confusion_matrix(y_test, y_pred))
    acc_svc=accuracy_score(y_pred,y_test)
    print('accuracy is',accuracy_score(y_pred,y_test))
```

| | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0 | 0.89 | 0.80 | 0.84 | 665 |
| 1 | 0.84 | 0.59 | 0.70 | 398 |
| 2 | 0.61 | 0.81 | 0.69 | 550 |
| accuracy | | | 0.75 | 1613 |
| macro avg | 0.78 | 0.74 | 0.74 | 1613 |
| weighted avg | 0.78 | 0.75 | 0.76 | 1613 |

```
[[534  1 130]
 [ 6 236 156]
 [ 61 44 445]]
accuracy is 0.7532548047117174
```

Soft Vector Machine

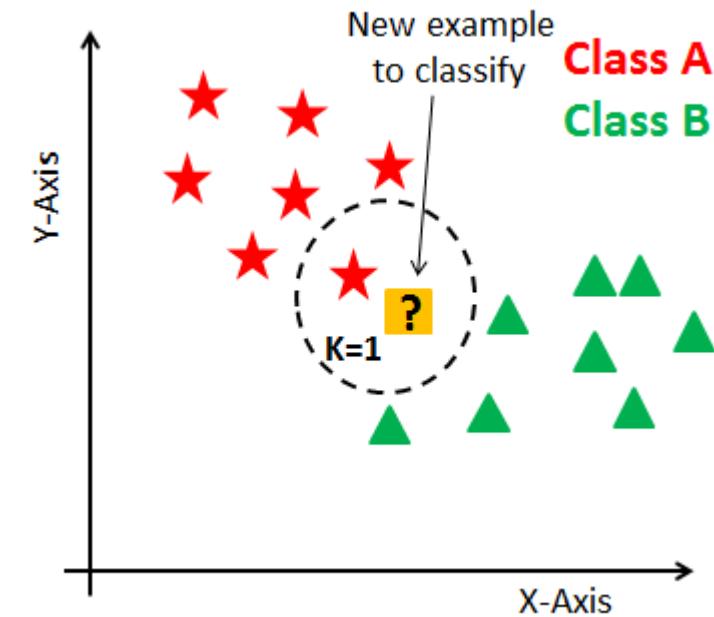
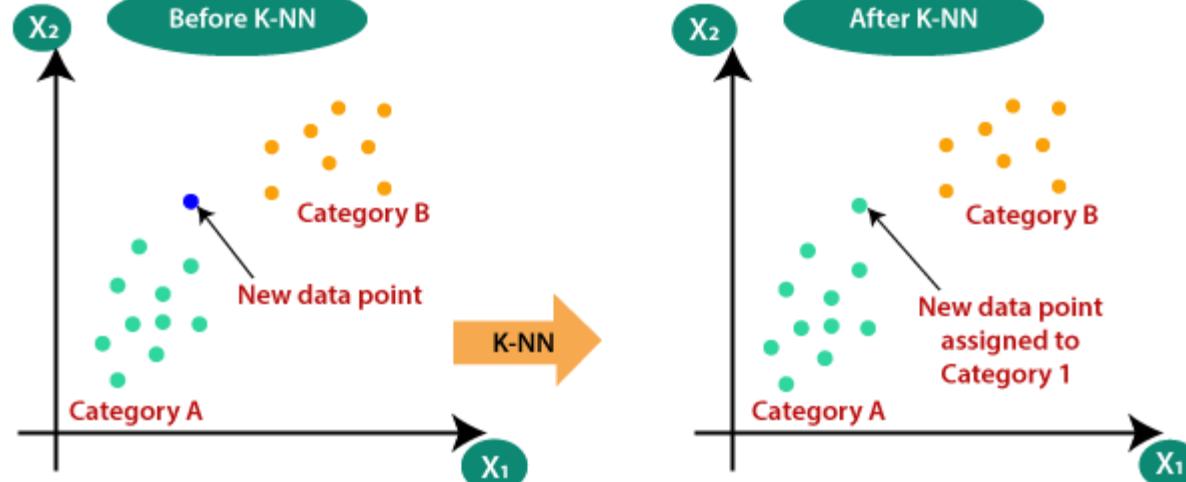
```
▶ svm = SVC()  
svm.fit(X1, y)  
plot_decision_regions(X1, y, clf=lr, legend=2)  
  
plt.xlabel('점유율')  
plt.ylabel('교통량')  
plt.title('SVM on VDS')  
plt.show()
```



KNN

K nearest neighbors : KNN

- ❖ KNN 알고리즘은 가까운 경계에 유사한 점이 있다고 가정한다. (유유상종 같은 개념)
 - ✓ KNN은 분류와 회귀에 둘 다 사용되지만, 분류에 더 많이 사용한다.



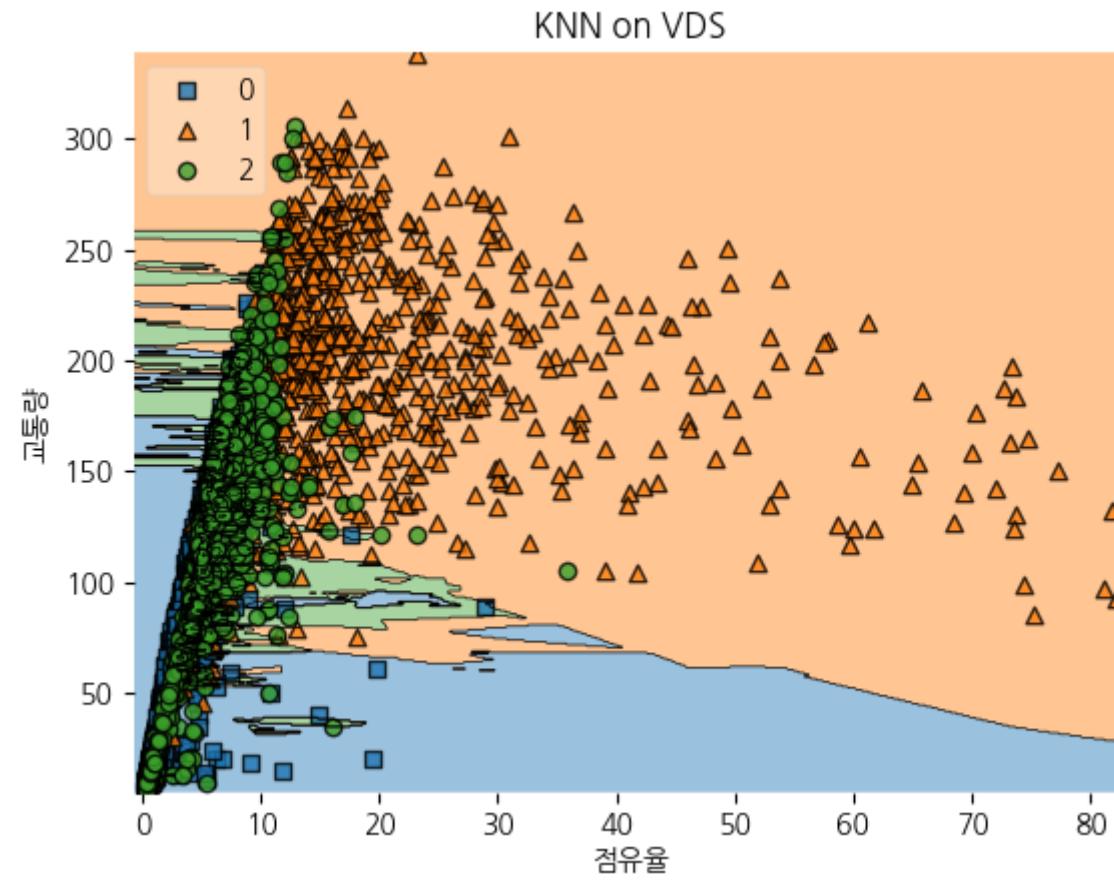
그림출처) http://localhost:8888/notebooks/lab_vds/vds01_label_visualization.ipynb

```
▶ # K-Nearest Neighbours
from sklearn.neighbors import KNeighborsClassifier
Model = KNeighborsClassifier(n_neighbors=8)
Model.fit(X_train, y_train)
y_pred = Model.predict(X_test)
# Summary of the predictions made by the classifier
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
# Accuracy score
acc_knn=accuracy_score(y_pred,y_test)
print('accuracy is',accuracy_score(y_pred,y_test))
```

| | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0 | 0.86 | 0.84 | 0.85 | 665 |
| 1 | 0.70 | 0.72 | 0.71 | 398 |
| 2 | 0.65 | 0.65 | 0.65 | 550 |
| accuracy | | | 0.75 | 1613 |
| macro avg | 0.74 | 0.74 | 0.74 | 1613 |
| weighted avg | 0.75 | 0.75 | 0.75 | 1613 |


```
[[557 15 93]
 [ 10 288 100]
 [ 81 110 359]]
accuracy is 0.7464352138871667
```

```
# Training a classifier
knn = KNeighborsClassifier(n_neighbors=8)
knn.fit(X1, y)
plot_decision_regions(X1, y, clf=knn, legend=2)
```



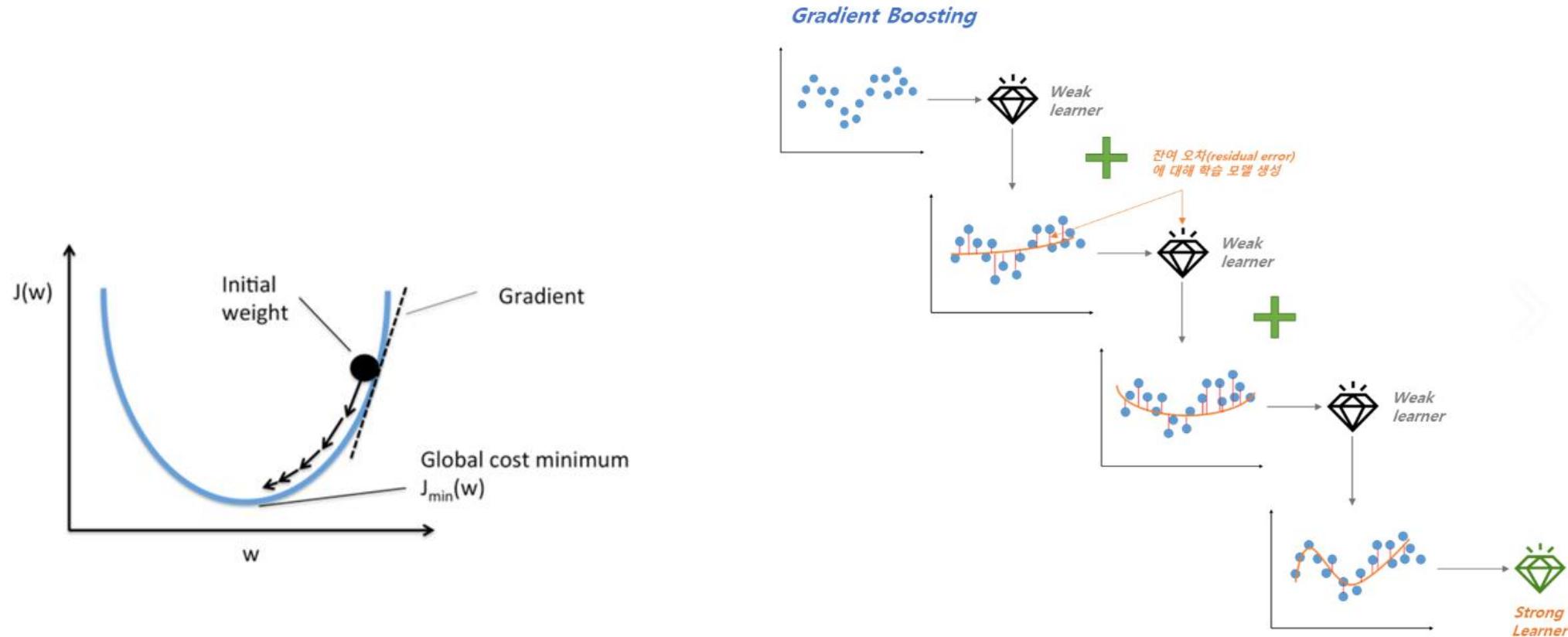
GBM (Gradient Boost Machine) 부스팅 알고리즘

❖ 양상블 학습 유형

- ✓ 보팅(Voting) : 여러 개의 분류기가 투표를 통해 최종 예측
 - 서로 다른 알고리즘(knn, svm, kmeans)을 가진 분류기를 결합하는 방식
- ✓ 배깅(Bagging) : 여러 개의 분류기가 투표를 통해 최종 예측
 - 대표적인 배깅 방식은 램덤 포레스트 알고리즘이다.
- ✓ 부스팅(Boosting)
 - 앞에서 학습한 분류기가 예측이 틀린 데이터에 대해서 올바른 예측을 할 수 있도록 다음 분류기에는 가중치(weight)를 부여하면서 학습
- ✓ 스태킹(Stacking)
 - 여러 가지 다른 모델의 예측 결과 값을 다시 학습 데이터로 만들어서 다른 모델(메타모델)로 재학습시켜 결과를 예측하는 방법

부스팅 알고리즘

- ❖ AdaBoost와 GBM(Gradient Boost Machine)이 있다.
 - ✓ AdaBoost: 오류 데이터에 가중치를 부여하면서 부스팅을 수행하는 알고리즘
 - ✓ GBM : 가중치 업데이트를 경사 하강법을 이용하는 것이 큰 차이다.



GradientBoosting

- ❖ Gradient Boosting은 약한 러너를 최적화 한다.

- ✓ 따라서 Greedy 알고리즘으로 과적합 발생한다.

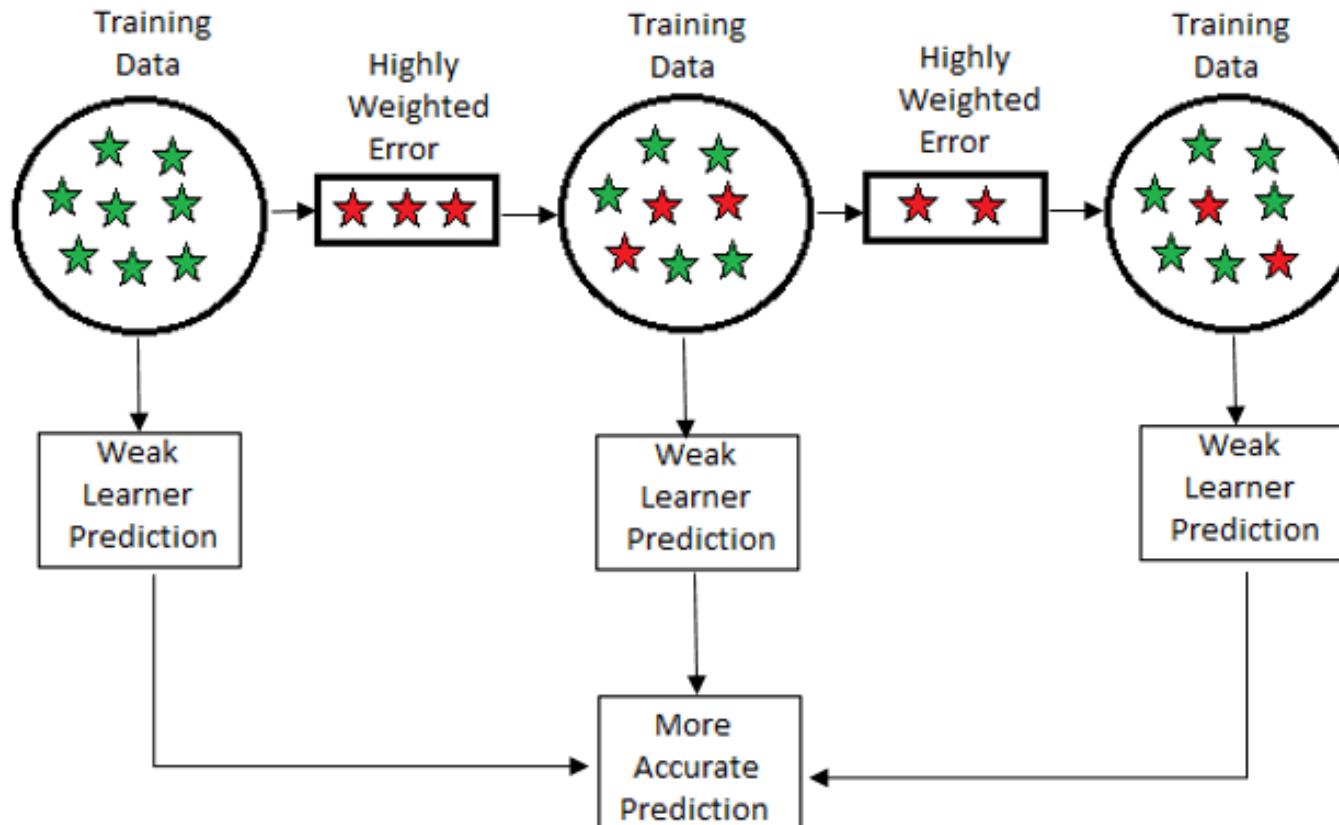


그림 출처) <https://pub.towardsai.net/fully-explained-gradient-boosting-technique-in-supervised-learning-d3e293ca70e1>

GradientBoosting

```
[39] from sklearn.ensemble import GradientBoostingClassifier
Model=GradientBoostingClassifier()
Model.fit(X_train,y_train)
y_pred=Model.predict(X_test)

# Summary of the predictions made by the classifier
print(classification_report(y_test,y_pred))
print(confusion_matrix(y_pred,y_test))

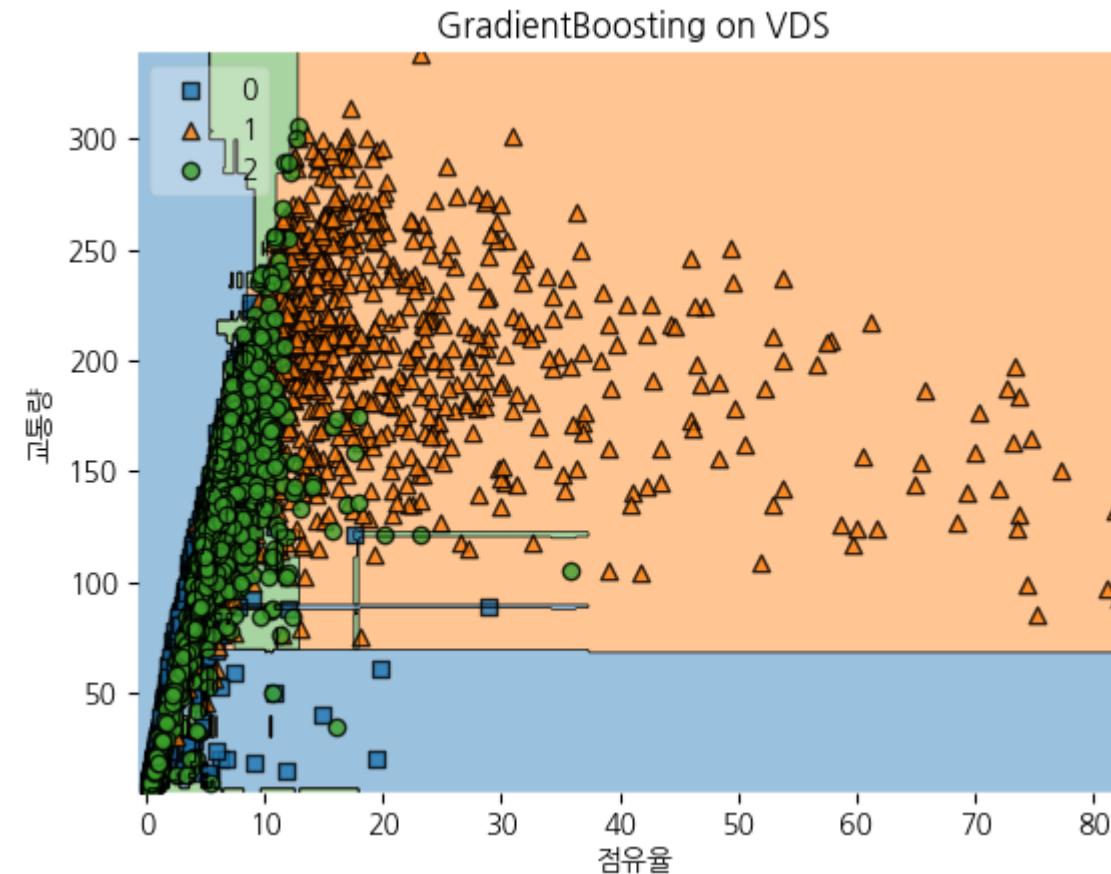
#Accuracy Score
acc_gb=accuracy_score(y_pred,y_test)
print('accuracy is ',accuracy_score(y_pred,y_test))
```

| | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0 | 0.91 | 0.83 | 0.87 | 665 |
| 1 | 0.75 | 0.72 | 0.73 | 398 |
| 2 | 0.65 | 0.74 | 0.69 | 550 |
| accuracy | | | 0.77 | 1613 |
| macro avg | 0.77 | 0.76 | 0.77 | 1613 |
| weighted avg | 0.78 | 0.77 | 0.78 | 1613 |

```
[[555  2 54]
 [ 4 286  91]
 [106 110 405]]
accuracy is  0.7724736515809052
```

GradientBoosting

```
▶ # Training a classifier
gboost=GradientBoostingClassifier()
gboost.fit(X1, y)
plot_decision_regions(X1, y, clf=gboost, legend=2)
```

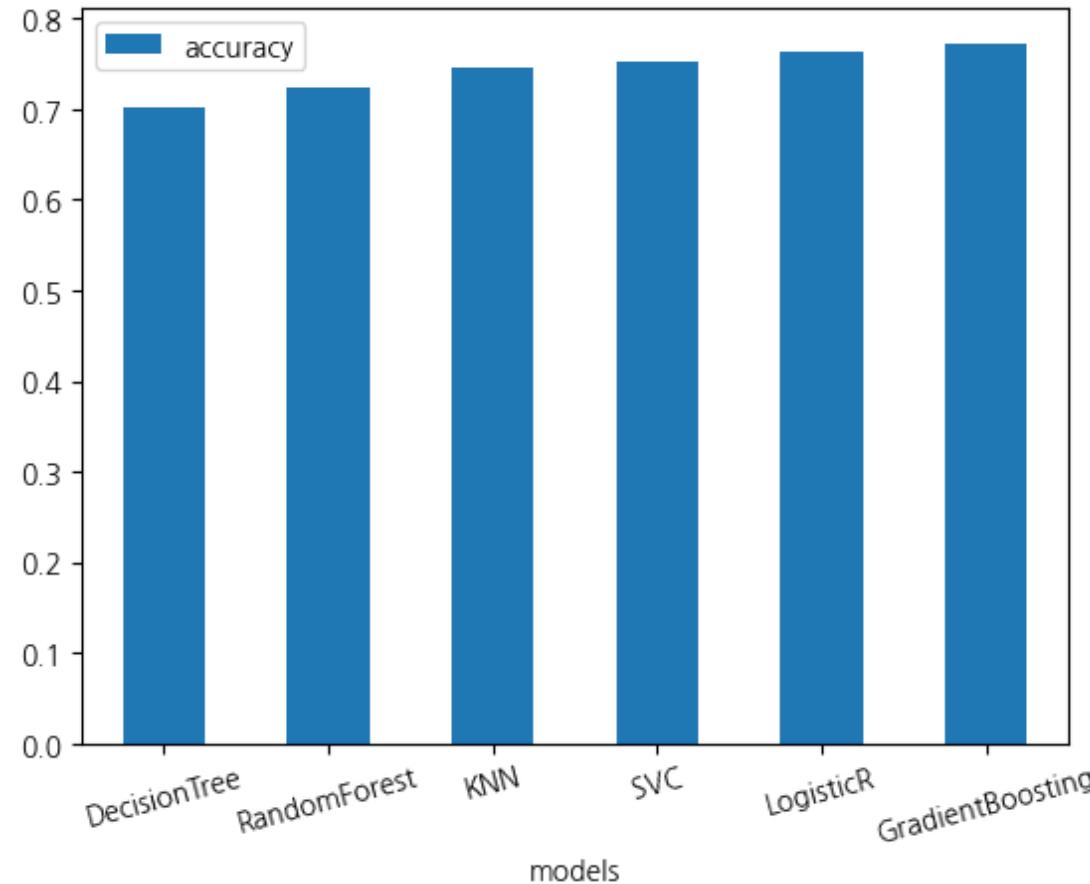


```
▶ df = pd.DataFrame({  
    'models': ['LogisticR', 'SVC', 'DecisionTree','RandomForest','KNN', 'GradientBoosting'],  
    'accuracy':[acc_lr, acc_svc, acc_dt, acc_rf, acc_knn, acc_gb]})  
df=df.sort_values(by='accuracy', ascending=True)  
  
df.plot.bar(x='models',y='accuracy',rot=15)
```

교통 데이터 분류(속도) 모델 정확도 비교

- ❖ 머신러닝 모델중에 GBM(GradientBoosting)이 가장 성능이 좋다.

- ✓ 실험한 조건에서, 모델의 하이퍼파라미터를 최적화 하면 결과는 바뀔 수 있다.
- ✓ 입력으로 교통량과 점유률을 사용한 경우 출력으로 속도를 예측하는 것인데 약 80% 이상의 정확도에 대하여 장/단점은 무엇인가?

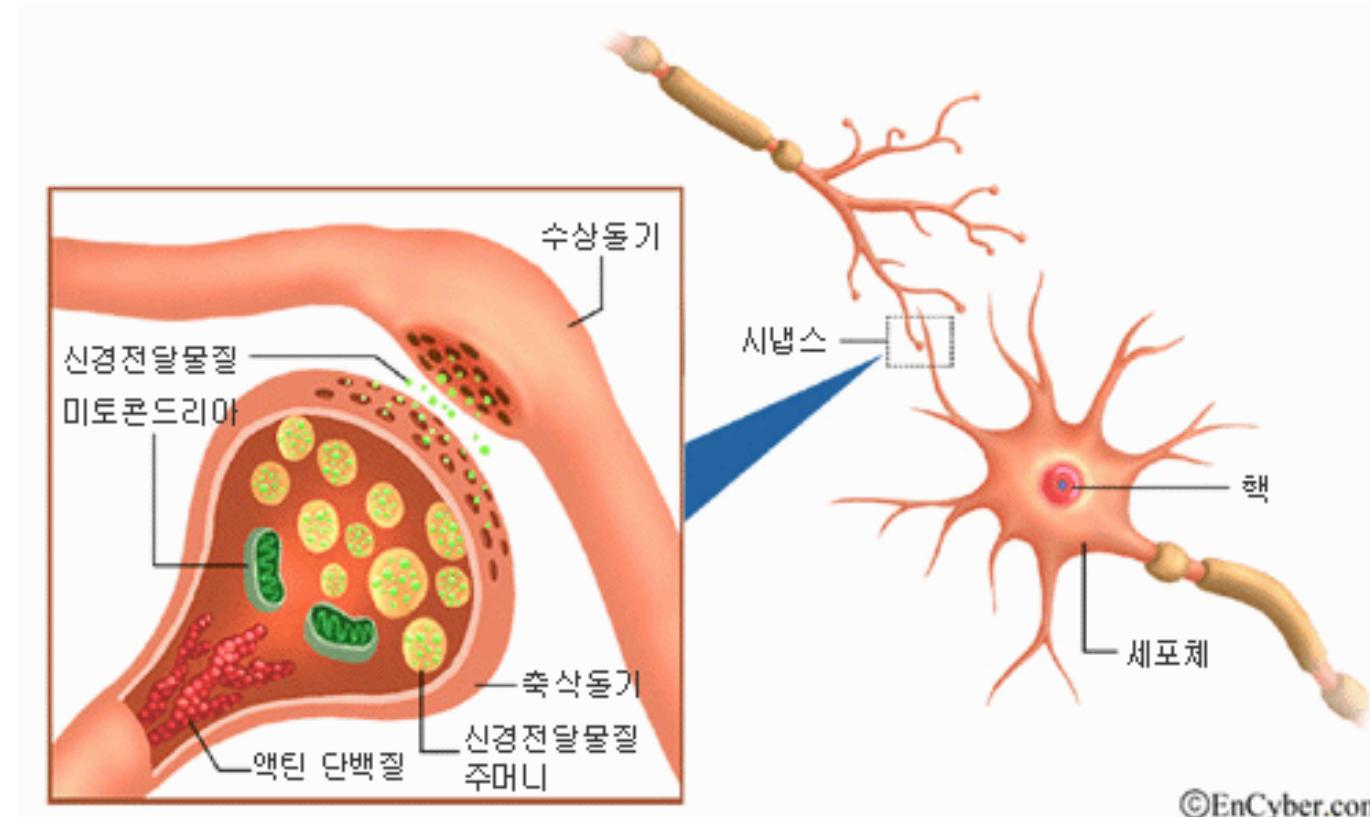


2.1

AI 간략 소개

A Biological Neuron

- ❖ 뉴런의 집합체로 한 뉴런의 축삭돌기 말단과 다음 뉴런의 수상돌기 사이의 연접 부위
- ❖ 시냅스 (Synapse)
 - ✓ 자극의 전달

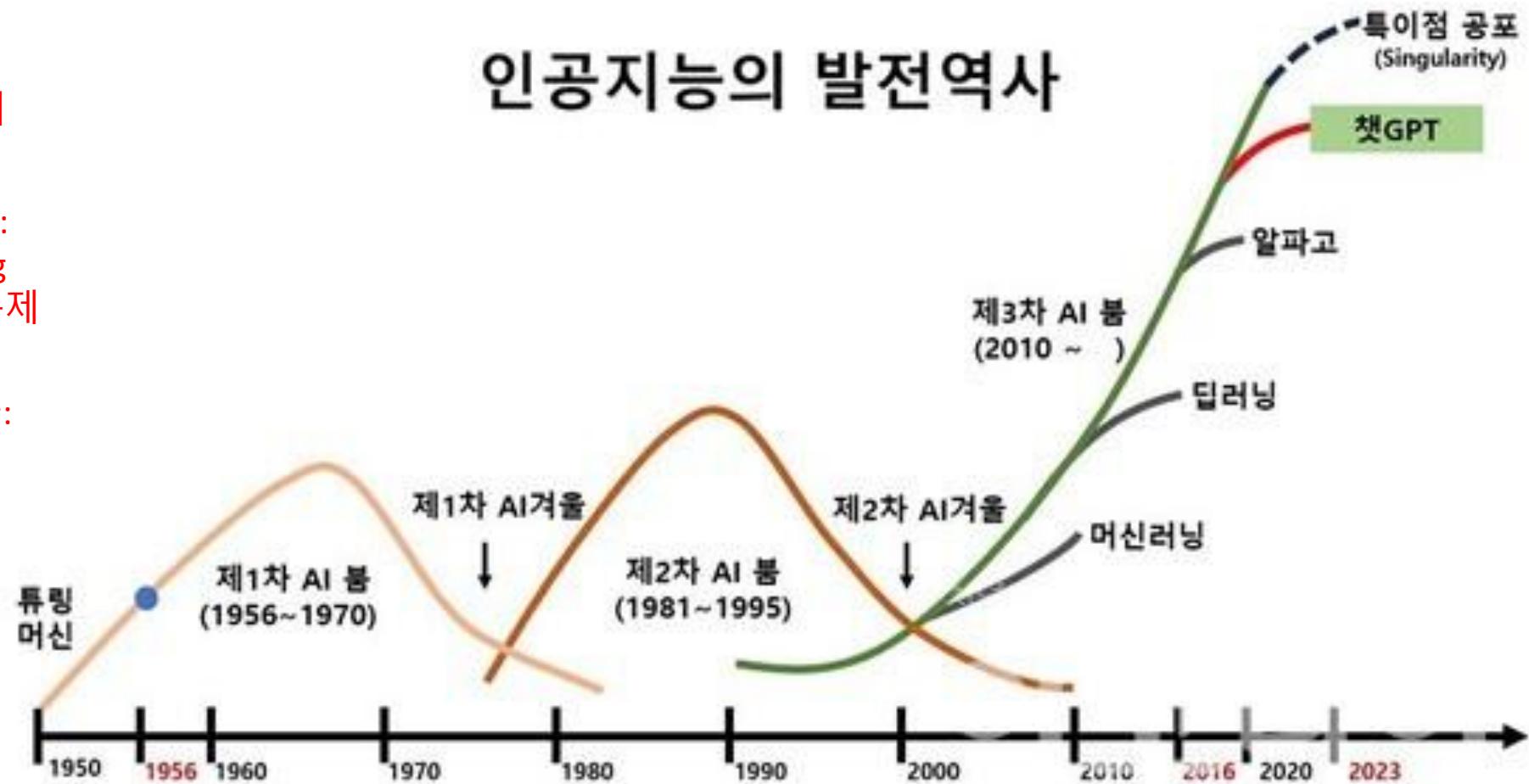


1차 겨울:
XOR 문제

2차 겨울:
Vanishing
Gradient 문제

3차 겨울:
?

인공지능의 발전역사

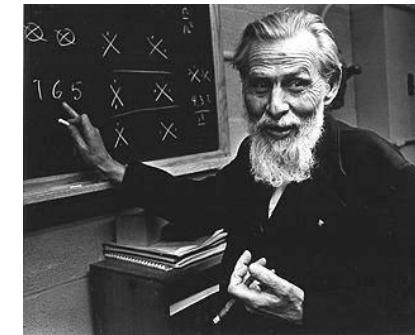
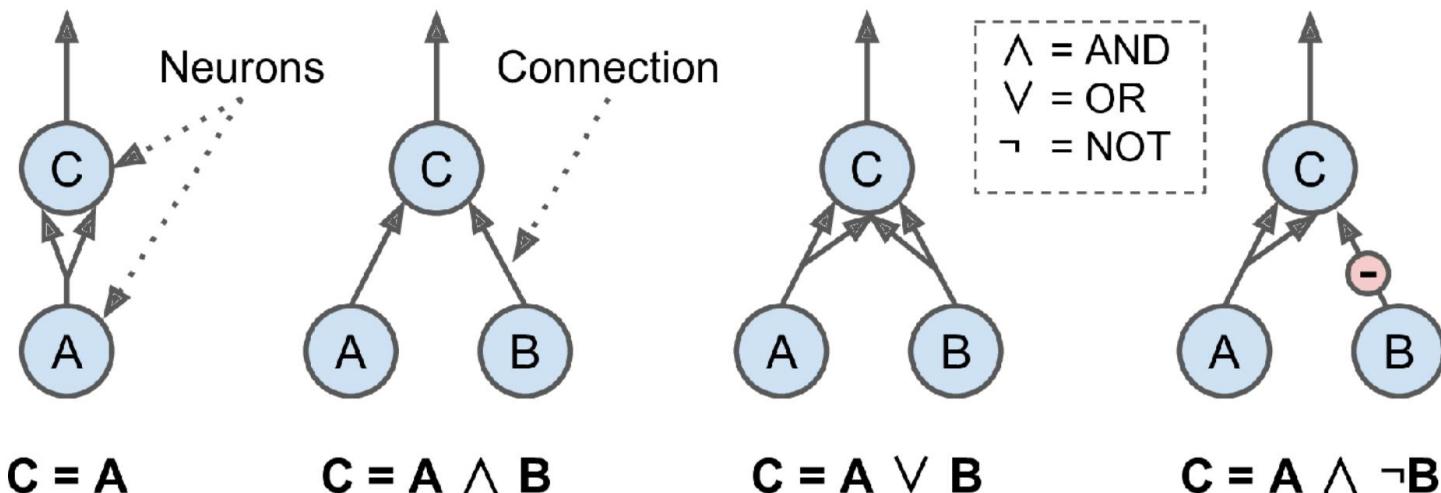


(자료) <https://www.ekoreanews.co.kr/news/articleView.html?idxno=67937>

인공지능의 역사 : 1943년

❖ 인공신경망 개념 최초 제안 : 맥컬록-피츠 신경망 모델 (1943)

- ✓ McCulloch와 Pitts가 1943년에 ANN 최초 논문이 발표
- ✓ 인간의 신경 구조를 복잡한 스위치들이 연결된 네트워크로 표현할 수 있다



맥컬록

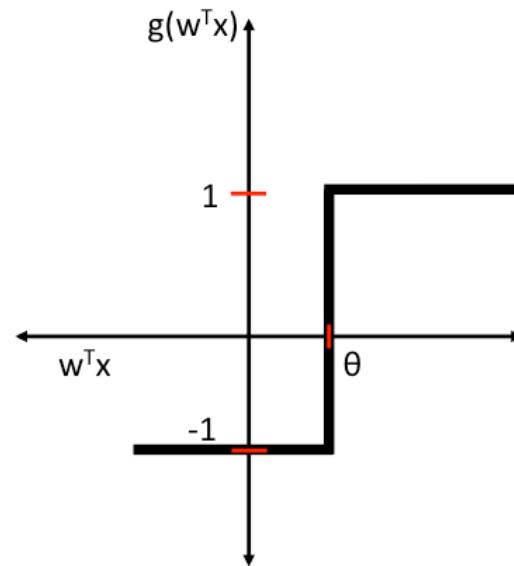


피츠

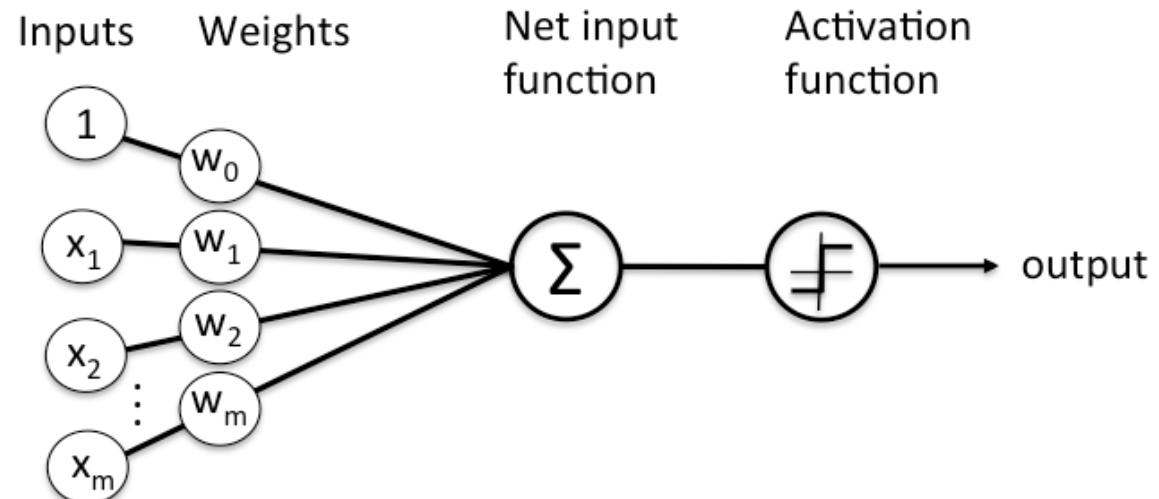
The perceptron (1958)

❖ Rosenblatt's perceptron

✓ 활성함수는 TLU(계단함수)



Unit step function.



Schematic of Rosenblatt's perceptron.

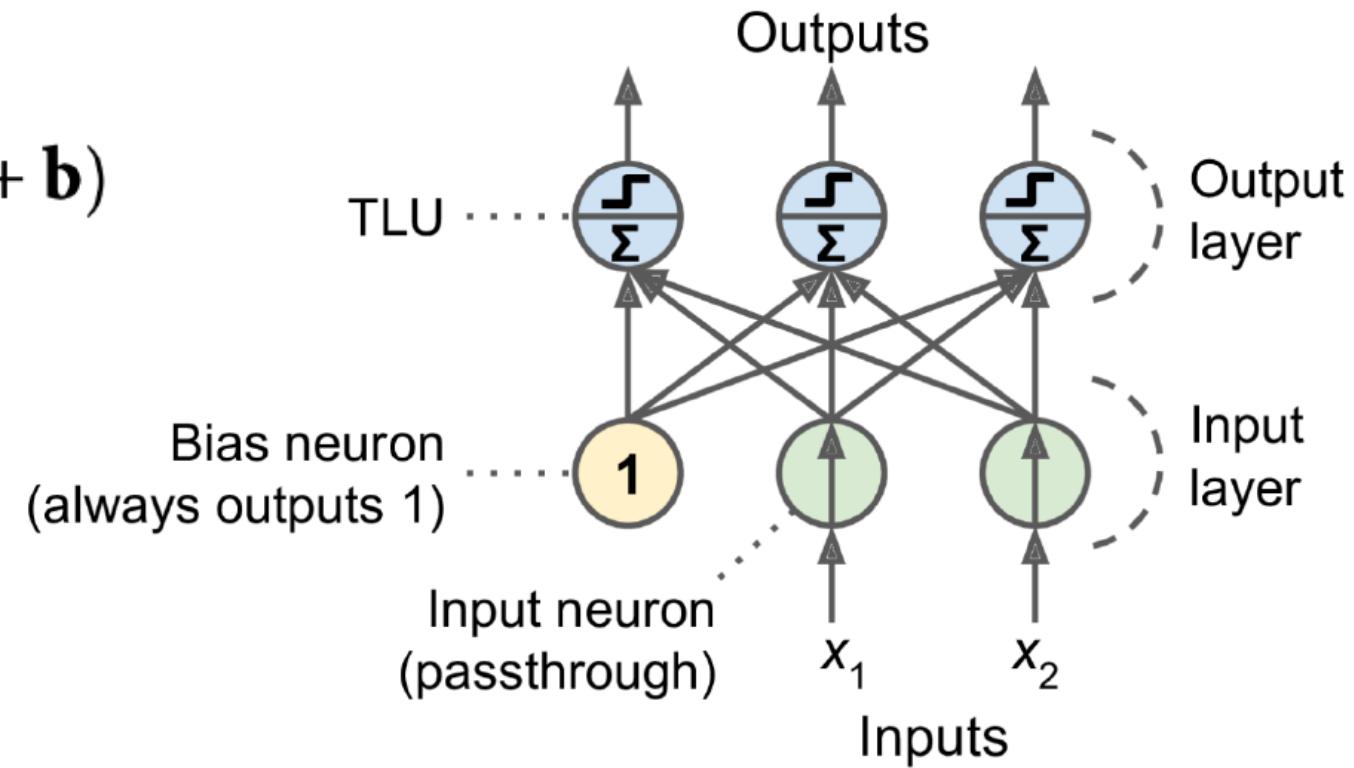
(source) https://sebastianraschka.com/Articles/2015_singlelayer_neurons.html

The Perceptron : Threshold Logic Unit

❖ Architecture of a Perceptron

- ✓ with two input neurons, one bias neuron, and three output neurons:
- ✓ Computing the outputs of a fully connected layer

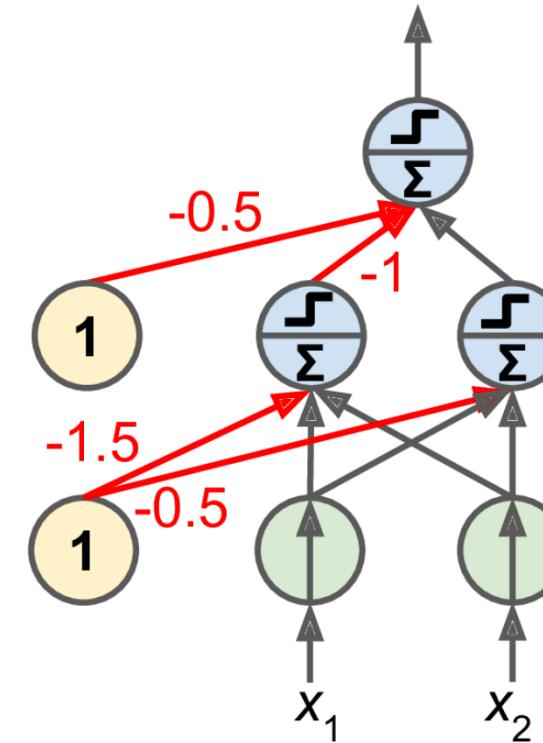
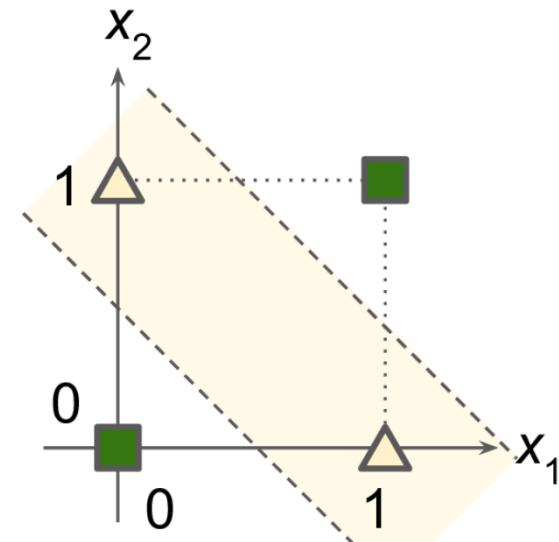
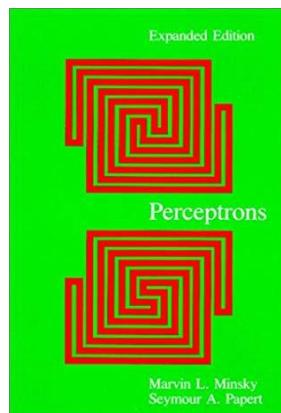
$$h_{\mathbf{W}, \mathbf{b}}(\mathbf{X}) = \phi(\mathbf{X}\mathbf{W} + \mathbf{b})$$



ANN의 역사 (4): 1차 겨울 (1969)

❖ 퍼셉트론의 무용론 등장으로 1차 인공지능 겨울이 시작된다.

- ✓ 1969년 마빈 민스키는 '퍼셉트론'은 단순 선형 분류기이며,
- ✓ 'XOR' 분류도 할 수 없는 미미한 선형분류기라는 것을 수학적으로 증명함
- ✓ 퍼셉트론 인기가 사그라 들면서 인공지능 1차 암흑기가 도래한다.

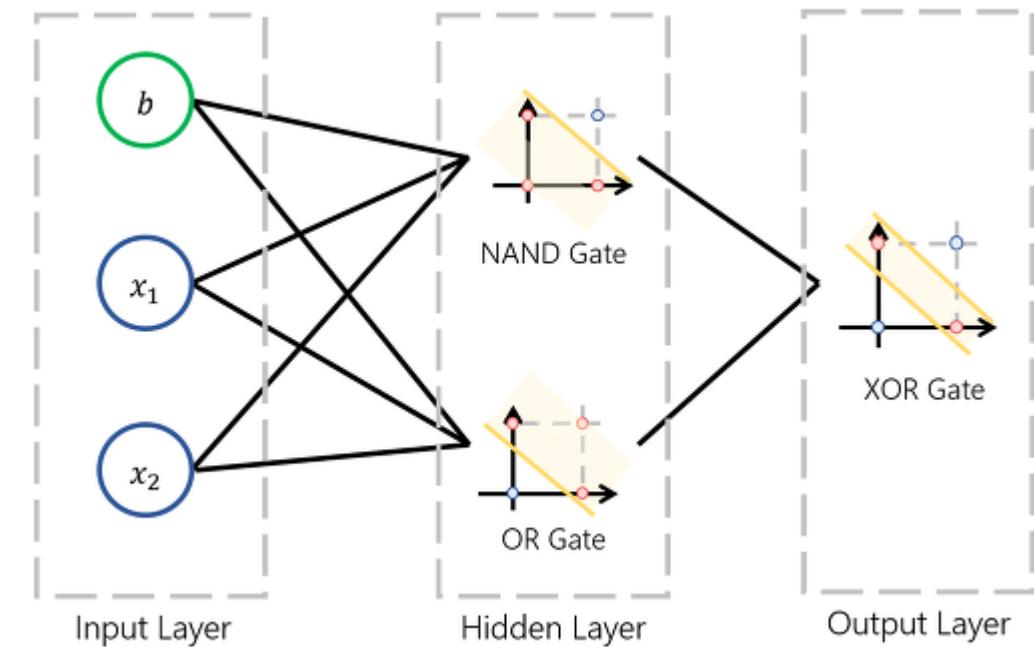
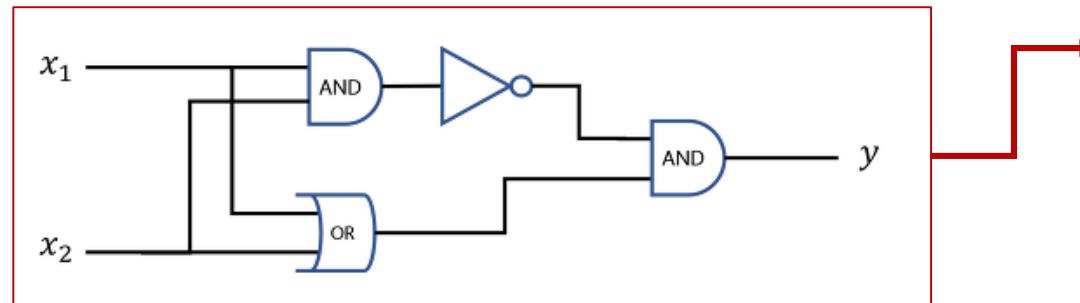


MLP : Multilayer Perceptron

❖ MLP로 'XOR' 문제 해결

- ✓ XOR 게이트는 'AND'와 'OR', 'NOT' 게이트의 조합
- ✓ Perceptron으로는 'AND'나 'OR' 게이트 만들 수 있다.
- ✓ 여러 개의 Perceptron을 조합하면 'XOR' 게이트 만들 수 있다.

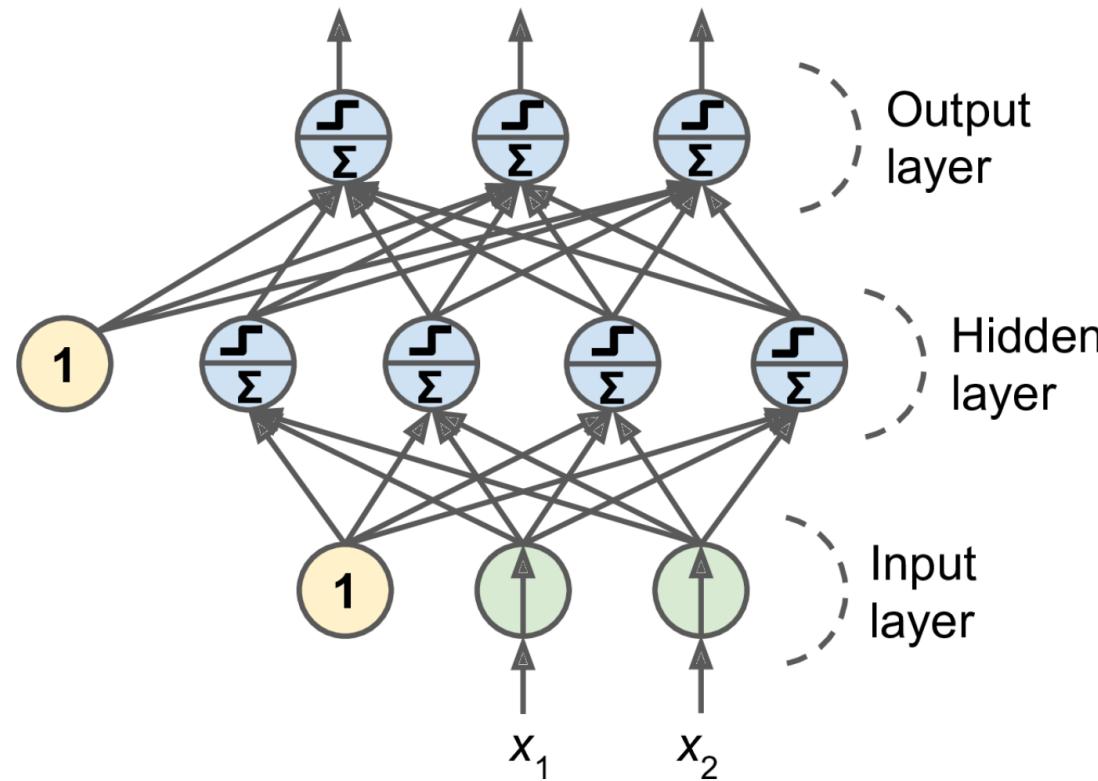
$$\text{XOR} = \text{!AND} \And \text{OR}$$



(source) <https://gomguard.tistory.com/178>

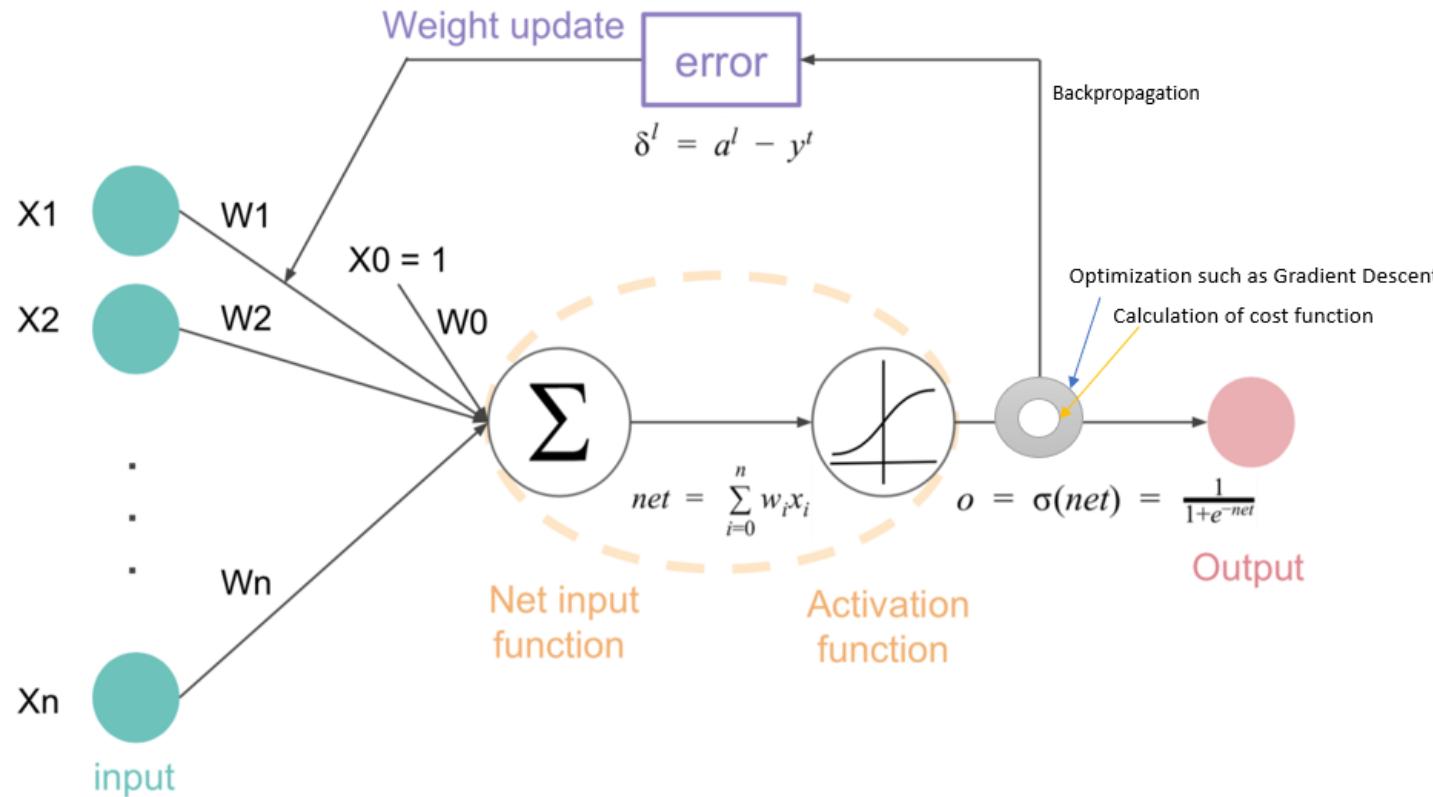
❖ 다층 퍼셉트론과 역전파 알고리즘 등장

- ✓ 다층 퍼셉트론은 전방향(feed-forward) 신경망으로 중간에 은닉층을 추가함
- ✓ 다층 퍼셉트론은 은닉층(hidden layer)라는 중간 레이어를 추가로 XOR 문제를 해결

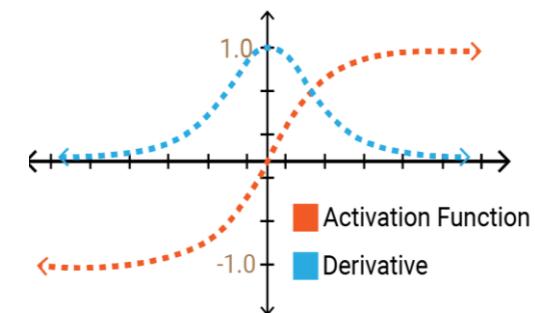


Backpropagation로 AI 다시 봄이 시작된다

- ❖ 역전파알고리즘 (Geoffrey E. Hinton, 1986)



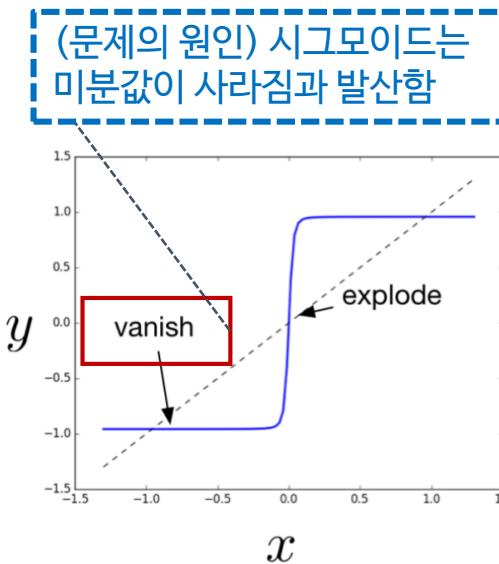
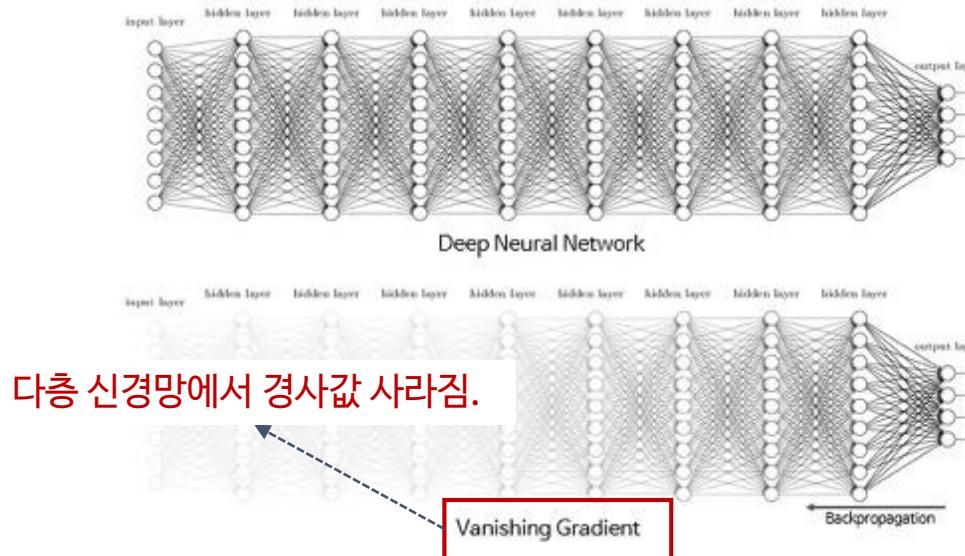
활성함수로 TLU(계단함수) 대신
미분가능한
Sigmoid 함수 사용



(source) <https://datascience.stackexchange.com/questions/44703/how-does-gradient-descent-and-backpropagation-work-together>

❖ 다층 퍼셉트론의 경사 발산과 소멸 문제 등장(1997)

- ✓ RNN에서 VGP(Vanishing Gradient Problem) 문제 (1993)
- ✓ LSTM(Long Short Term Memory)로 VGP 해결(1997), Hochreiter
- ✓ 이시기는 기계학습 SVM, Random Forest 등이 큰 인기가 있음.

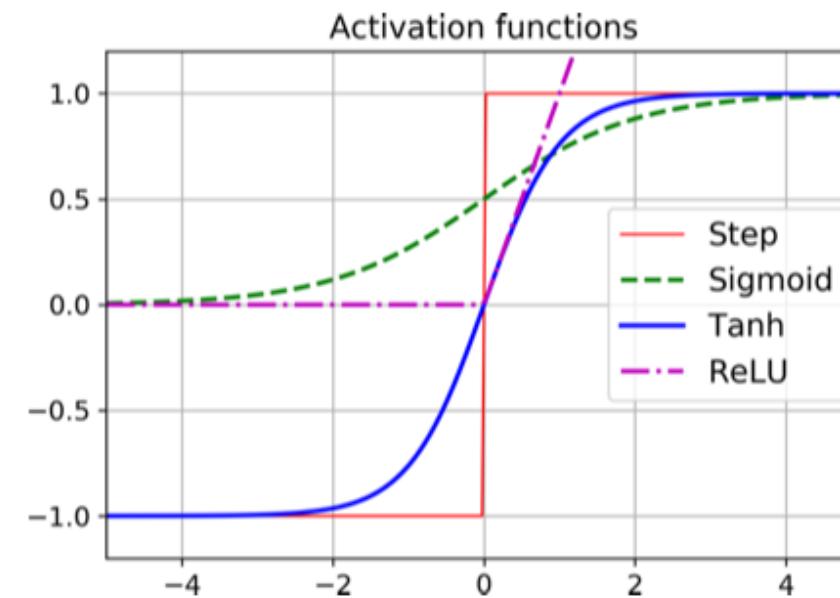


❖ MLP의 문제점

- ✓ BP에서 Vanishing Gradient Problem 발생
- ✓ MLP의 오버피팅(Overfitting) 문제

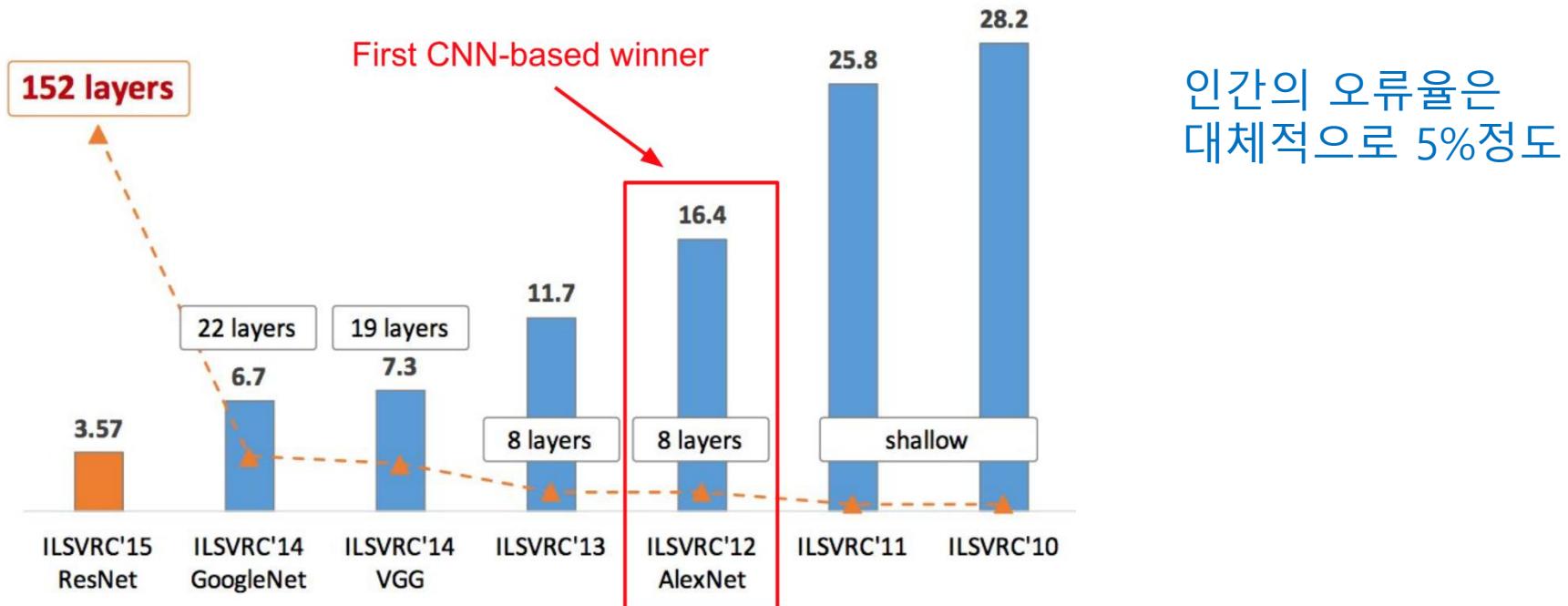
❖ MLP 문제 해결책

- ✓ RNN, CNN의 가중치 공유
- ✓ ReLU로 Deep Learning 가능
- ✓ GPU 등 가속기 등장



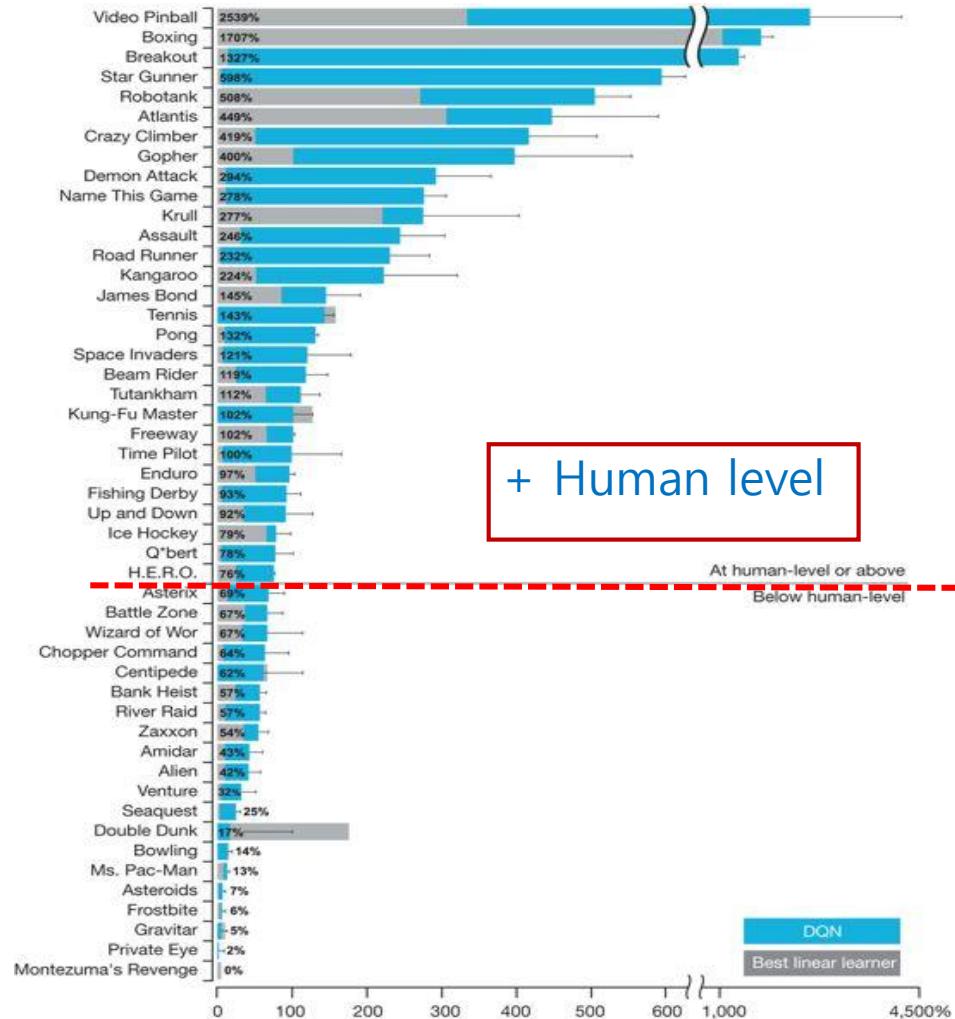
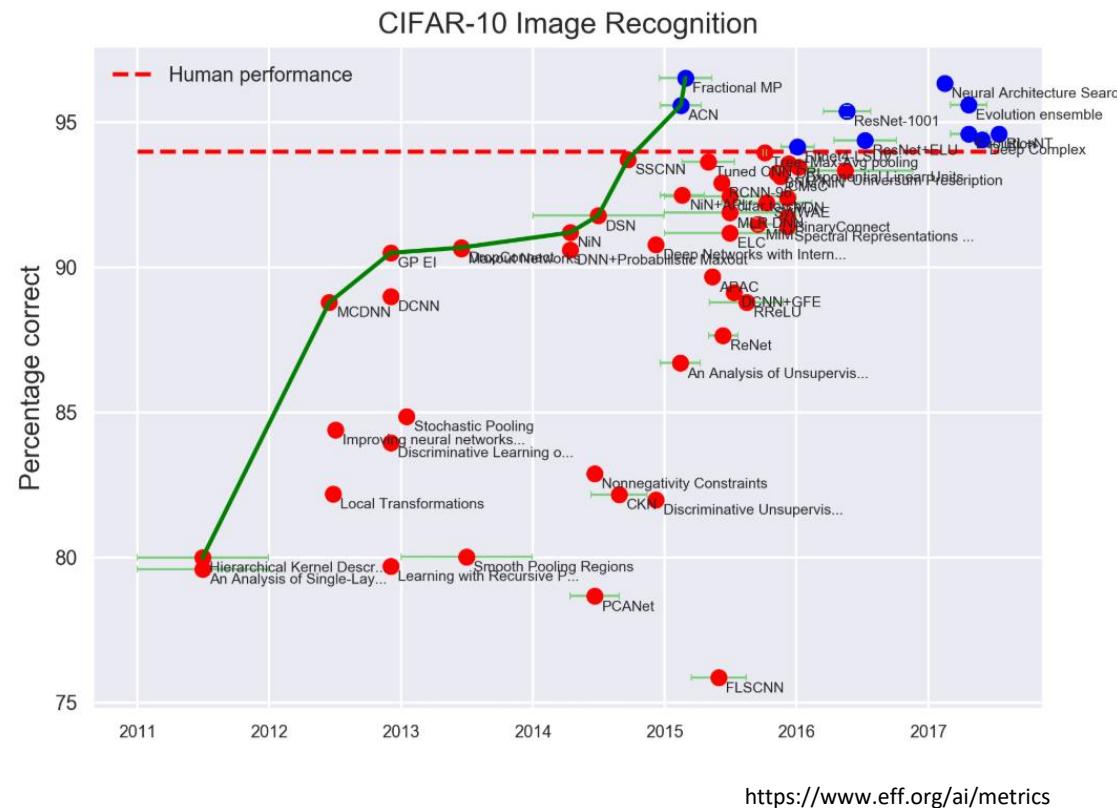
❖ 신경망 대신 딥러닝으로 인공지능 분야의 부활 (2006~)

- ✓ 드롭아웃 층(dropout layer) 도입으로 과적합 문제를 해결
- ✓ ReLU(Rectified Linear Unit) 활성함수 도입으로 기울기 사라짐 문제 해결
- ✓ GPU 컴퓨팅과 고속 최적화 알고리즘 등장
- ✓ ILSVRC에서 오류율을 15%로 획기적으로 낮춤



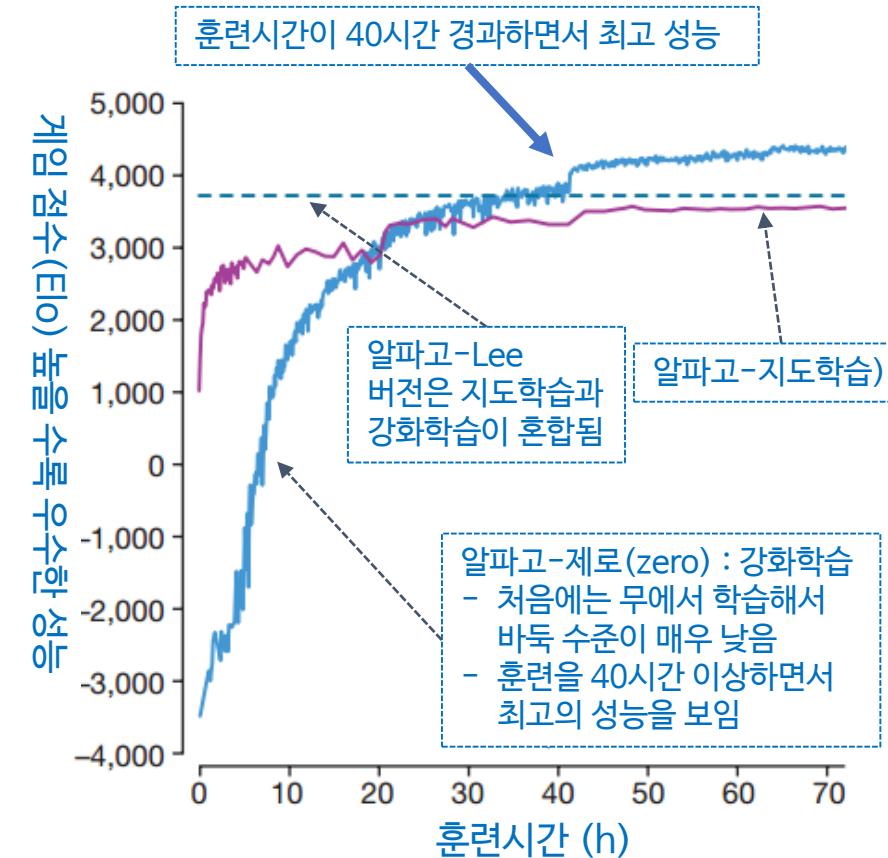
ANN의 역사 (12): 이미지 및 비전

구글 딥마인드, Nature 2015



알파고(AlphaGo)는 3개 버전

- ① 알파고-지도학습(바둑 기보)
- ② 알파고-리 (이세돌 9단)
- ③ 알파고-제로 (강화학습)



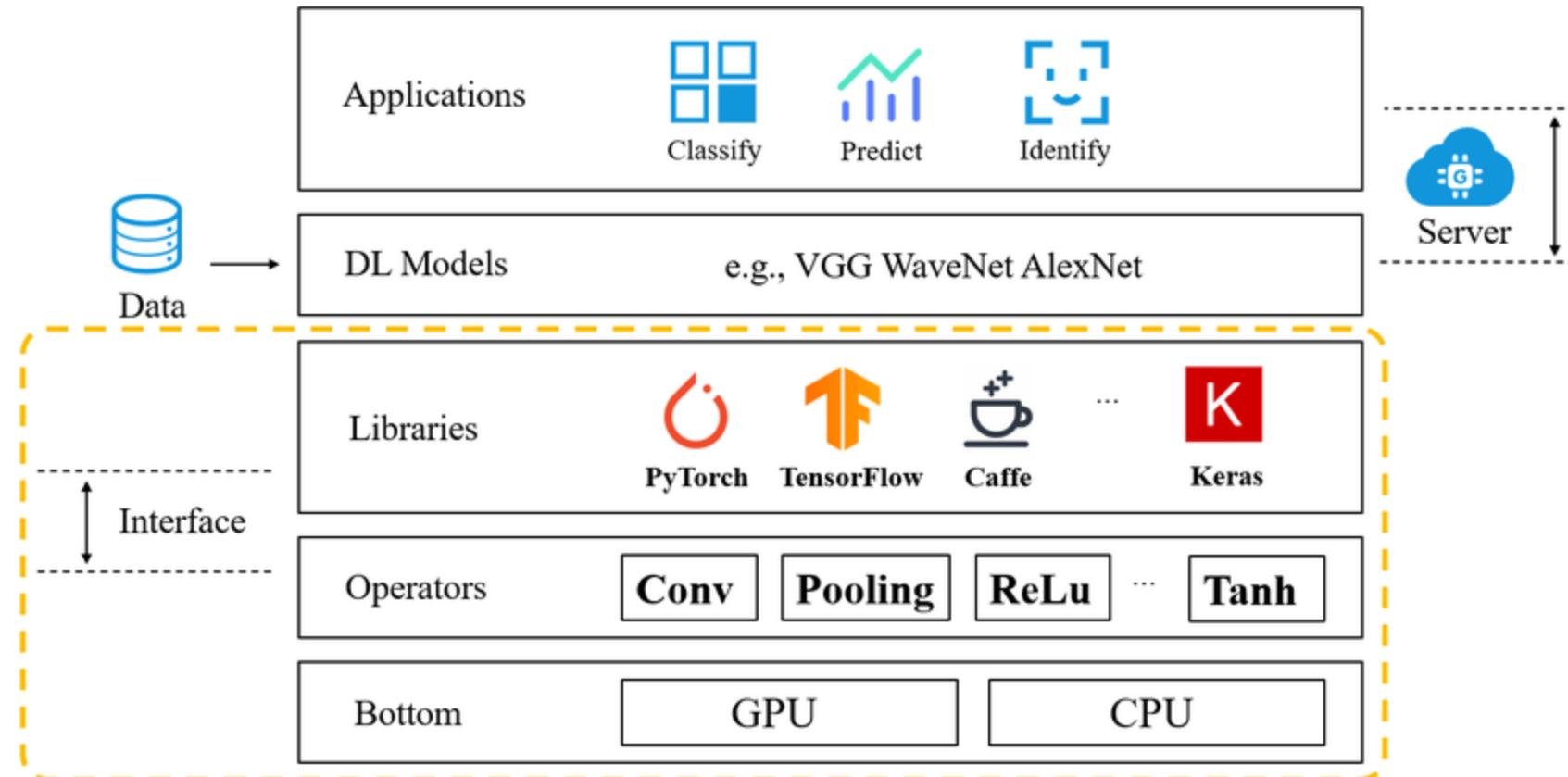
자료: Mastering the game of Go without human knowledge , David Silver, et al. Nature(2017)

2-2

Tensorflow를 이용한 딥러닝 모델

Deep learning frameworks

❖ deep learning framework 다이어그램

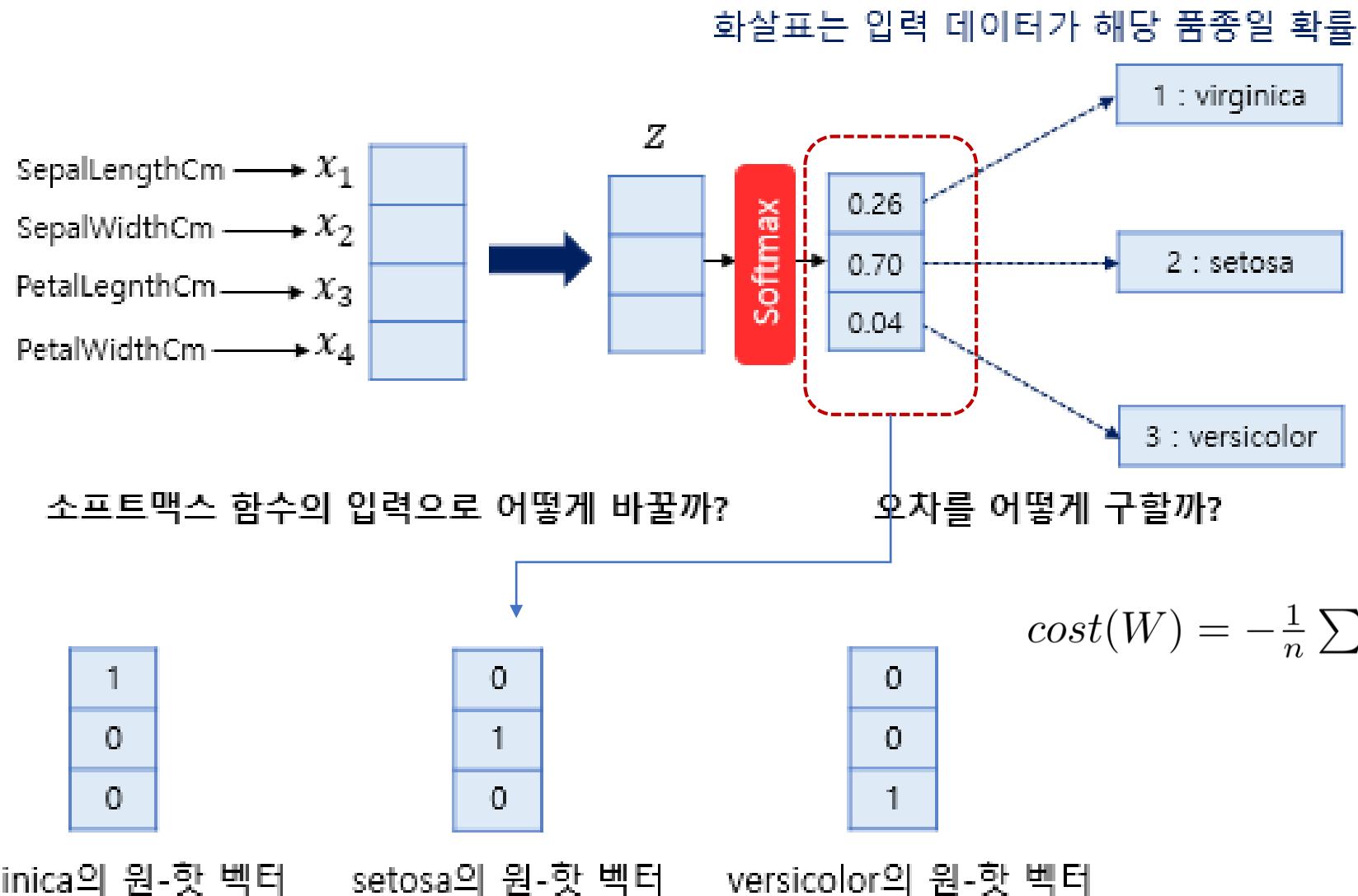


텐서플로우 2.0 시작하기: 초보자용 참고

- ❖ 데이터셋 로드
- ❖ 모델 만들기
 - ✓ tf.keras.Sequential
- ❖ 모델은 훈련하기
 - ✓ model.compile
 - ✓ model.fit
- ❖ 모델 평가 하기
 - ✓ model.evaluate
- ❖ 모델 결과 가시화
 - ✓ plt.plot(history.history['loss'])



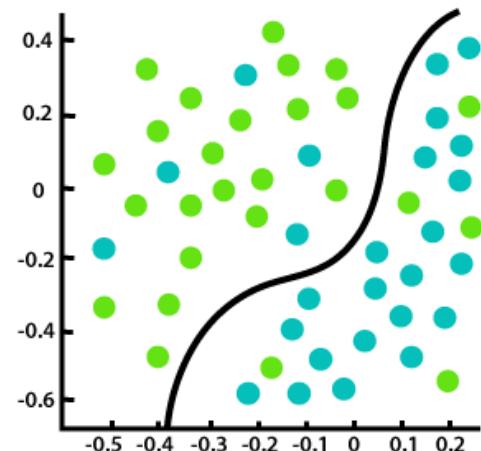
Softmax



❖ *Typical classification MLP architecture*

Table 10-2. Typical classification MLP architecture

| Hyperparameter | Binary classification | Multilabel binary classification | Multiclass classification |
|-------------------------|-----------------------|----------------------------------|---------------------------|
| Input and hidden layers | Same as regression | Same as regression | Same as regression |
| # output neurons | 1 | 1 per label | 1 per class |
| Output layer activation | Logistic | Logistic | Softmax |
| Loss function | Cross entropy | Cross entropy | Cross entropy |

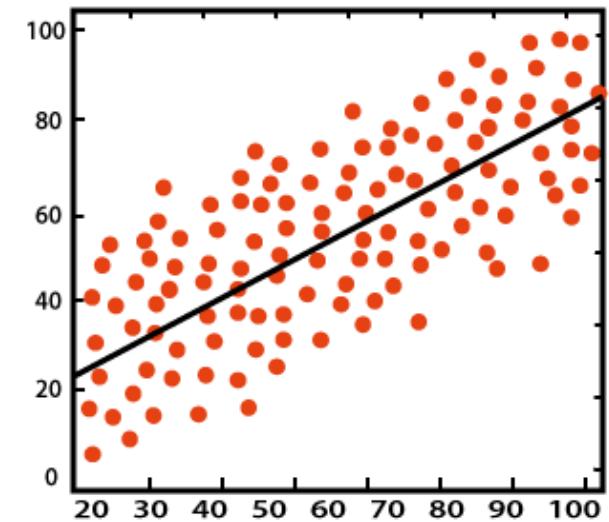


Classification

Regression MLPs

❖ Typical regression MLP architecture

| Hyperparameter | Typical value |
|----------------------------|--|
| # input neurons | One per input feature (e.g., $28 \times 28 = 784$ for MNIST) |
| # hidden layers | Depends on the problem, but typically 1 to 5 |
| # neurons per hidden layer | Depends on the problem, but typically 10 to 100 |
| # output neurons | 1 per prediction dimension |
| Hidden activation | ReLU (or SELU, see Chapter 11) |
| Output activation | None, or ReLU/softplus (if positive outputs) or logistic/tanh (if bounded outputs) |
| Loss function | MSE or MAE/Huber (if outliers) |

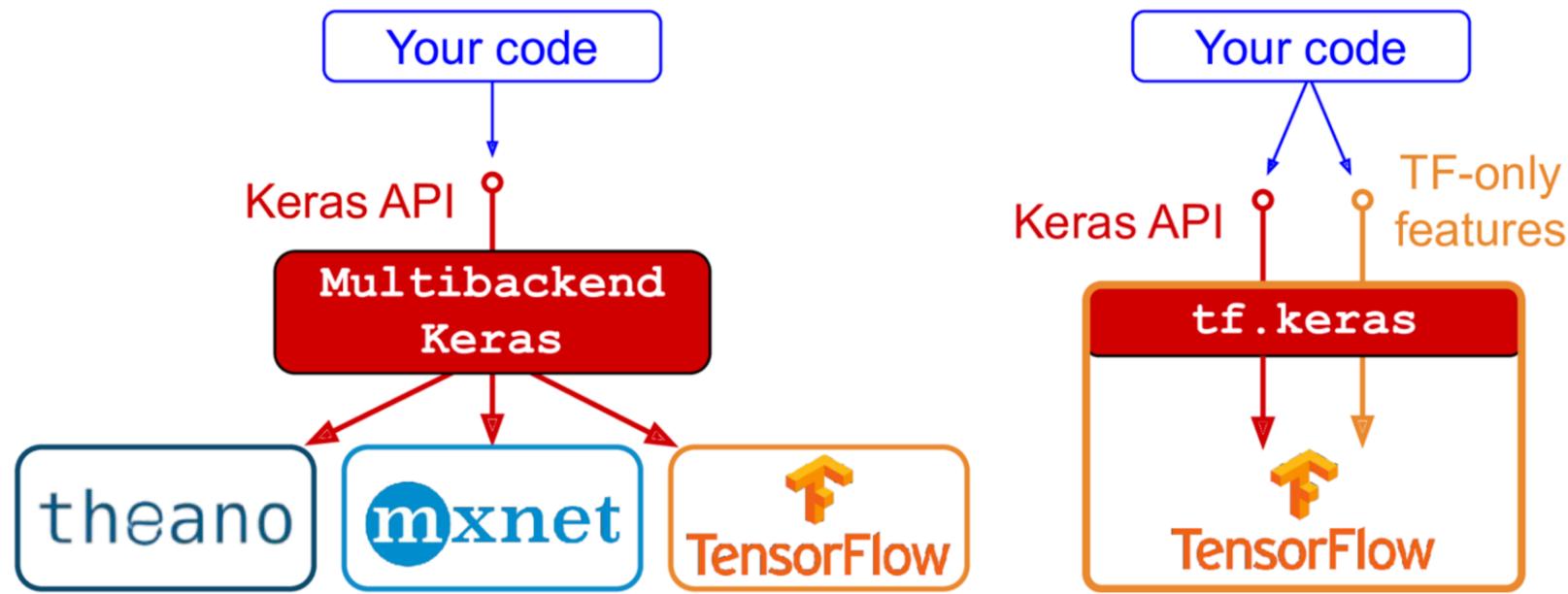


Regression

Implementing MLPs with Keras

❖ Keras is a high-level Deep Learning API

- ✓ that allows you to easily build, train, evaluate, and execute all sorts of neural networks.
- ✓ **tf.keras**: Extended Keras implementation based on TF with TF-specific features.
- ✓ Computation backend: TensorFlow, Microsoft Cognitive Toolkit (CNTK), Theano, Apache MXNet, Apple's Core ML, JavaScript or TypeScript, and PlaidML.



분류를 위한 데이터 라벨 전략 수립: 속도로 해보자!

```
num_classes = 3
class_labels= ['Jam', 'Slow', 'Normal']
```

```
def get_score(speed):
    if speed < 45:
        label = 'Jam'
    elif speed < 50:
        label = 'Slow'
    else :
        label = 'Normal'
    return label
```

```
df[ "label" ] = df[ "Speed" ].apply(lambda spd: get_score(spd))
df.head()
```

| | Date | ToVol | SmVol | MeVol | LaVol | Speed | OccRate | label |
|---|-----------------|-------|-------|-------|-------|-------|---------|--------|
| 0 | 2017-04-02 0:00 | 43 | 34 | 9 | 0 | 50.3 | 1.90 | Normal |
| 1 | 2017-04-02 0:05 | 45 | 32 | 13 | 0 | 58.9 | 1.84 | Normal |

데이터를 특성과 라벨(숫자로 변환)

```
X = df[['ToVol', 'LaVol', 'MeVol', 'SmVol', 'Speed', 'OccRate']]  
y = df['label']
```

```
encoder = LabelEncoder()  
y = encoder.fit_transform(y)
```

```
y
```

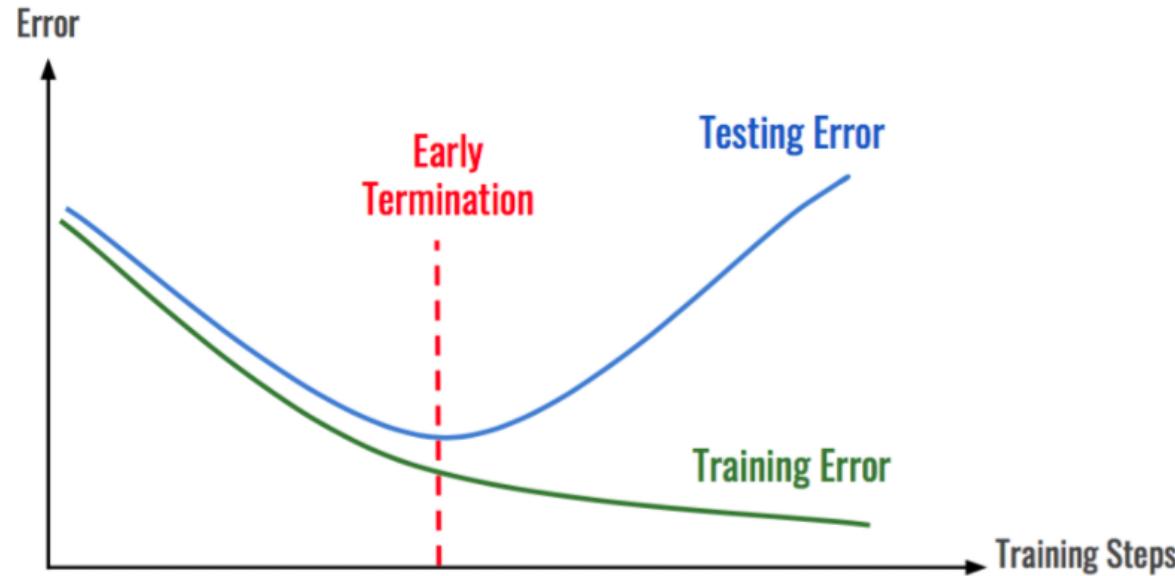
```
array([1, 1, 1, ..., 1, 1, 1])
```

```
X[:3]
```

| | ToVol | LaVol | MeVol | SmVol | Speed | OccRate |
|---|-------|-------|-------|-------|-------|---------|
| 0 | 43 | 0 | 9 | 34 | 50.3 | 1.90 |
| 1 | 45 | 0 | 13 | 32 | 58.9 | 1.84 |
| 2 | 46 | 0 | 12 | 34 | 50.6 | 1.87 |

```
import numpy as np  
from keras.utils import to_categorical  
### Categorical data to be converted to numeric data  
colors = [ "red", "green", "yellow", "red", "blue" ]  
  
### Universal list of colors  
total_colors = [ "red", "green", "blue", "black", "yellow" ]  
  
### map each color to an integer  
mapping = {}  
for x in range(len(total_colors)):  
    mapping[total_colors[x]] = x  
  
# integer representation  
for x in range(len(colors)):  
    colors[x] = mapping[colors[x]]  
  
one_hot_encode = to_categorical(colors)  
print(one_hot_encode)
```

Sparse_Categoricalcrossentropy를 사용함



Train/Test 데이터 나누기

```
scaler = StandardScaler()  
X_scaled = scaler.fit_transform(X)
```

```
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3, random_state=1)
```

```
print(X_train.shape, y_train.shape)  
print(X_test.shape, y_test.shape)
```

```
(5644, 6) (5644,)  
(2420, 6) (2420,)
```

```
X_train[:2]
```

```
array([[-0.67958742, -0.1033334 , -0.36417997, -0.7767931 ,  1.34731445,  
       -0.52777298],  
      [-1.55526491, -0.75688677, -1.46481163, -1.50330118,  0.66561375,  
       -0.81414318]])
```

```
#import keras
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout

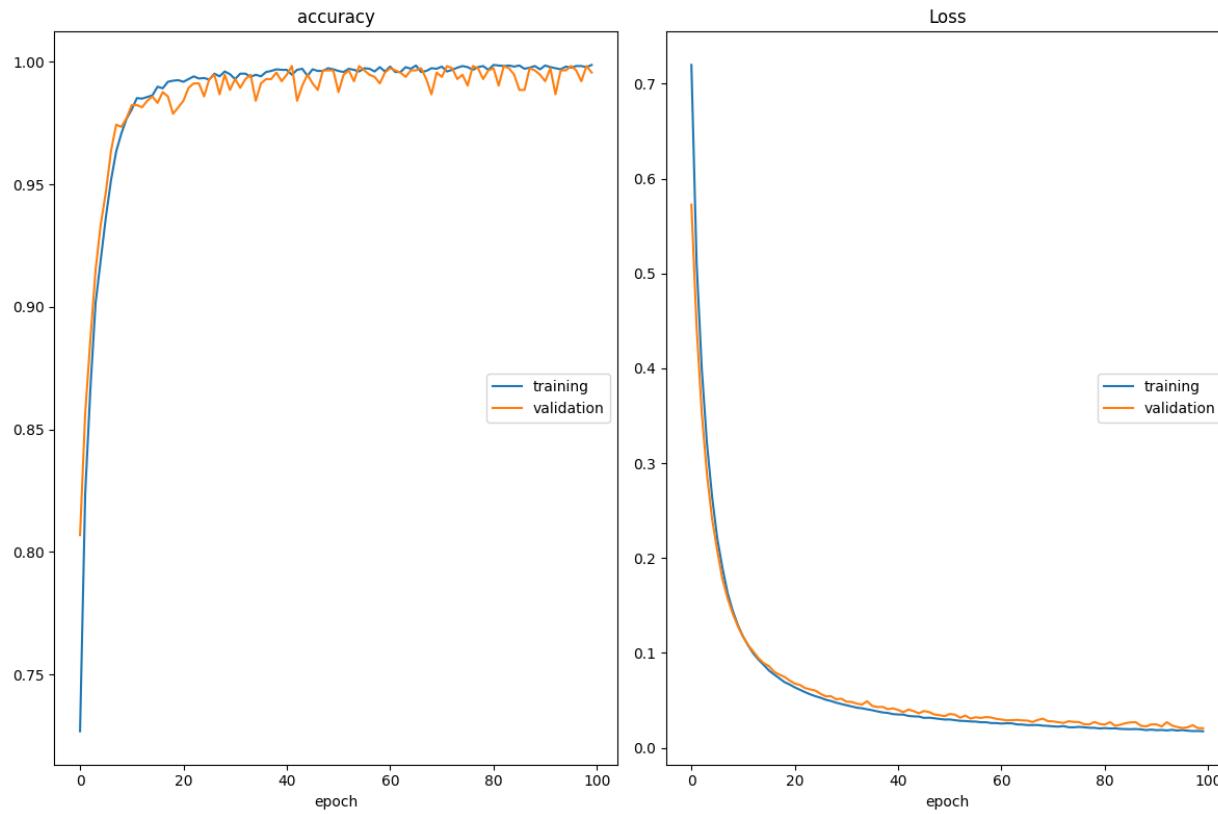
from livelossplot import PlotLossesKeras
```

Shallow Deep learning

```
def model_shallow():
    model = Sequential([
        Dense(32, input_dim=X_train.shape[1], activation = 'relu'),
        Dense(num_classes, activation= 'softmax')
    ])
    model.compile(loss = 'sparse_categorical_crossentropy', optimizer = 'adam', metrics = ['accuracy'])
    return model
```

모델 훈련과정

```
model = model_shallow()
#history = model.fit(X_train,y_train, epochs=100, validation_split=0.2)
history = model.fit(X_train,y_train, epochs=100, validation_split=0.2, callbacks=[PlotLossesKeras()])
```

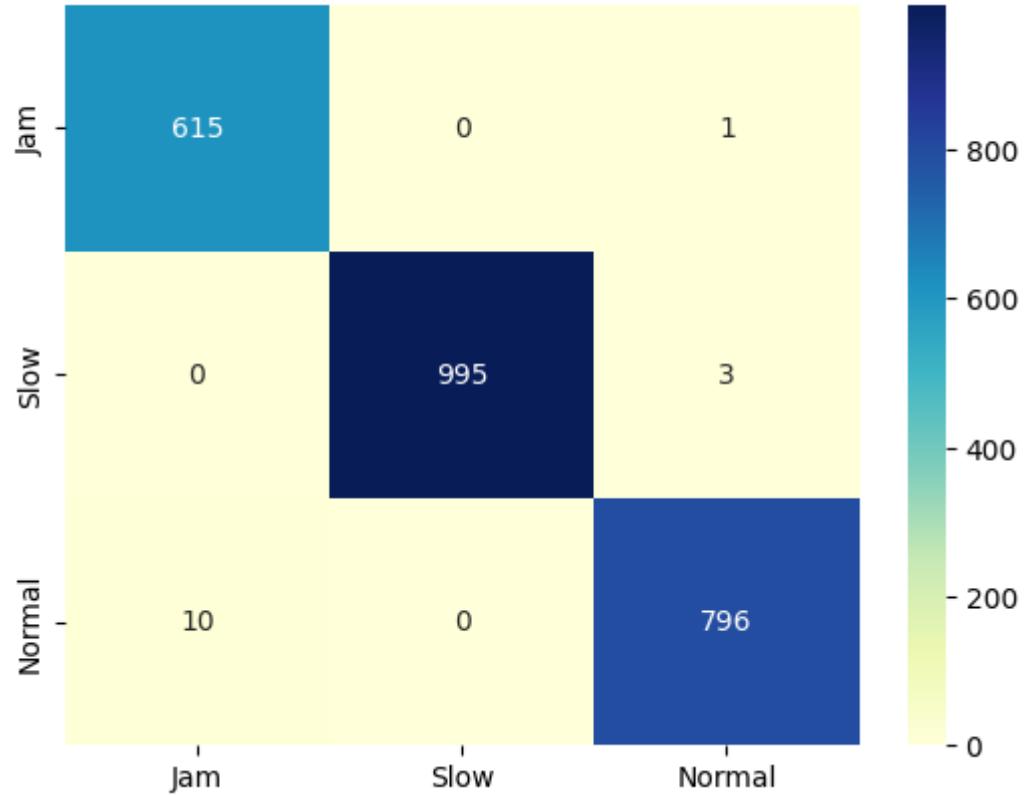


```
model.evaluate(X_test,y_test)
76/76 [=====] - 0s 2ms/step - loss: 0.0187 - accuracy: 0.9942
[0.01872224360704422, 0.9942148923873901]
```

Confusion Matrix 분석

```
from sklearn.metrics import confusion_matrix
import numpy as np

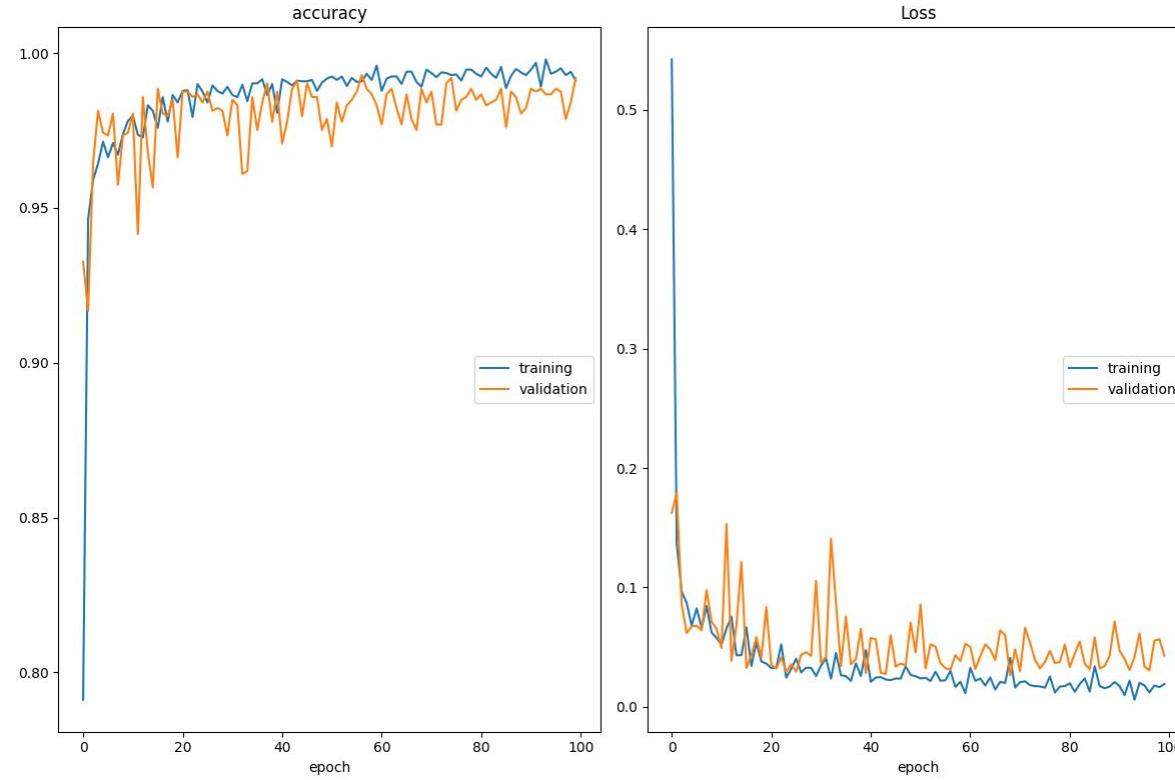
y_pred_labels = [np.argmax(label) for label in y_pred]
cm = confusion_matrix(y_test,y_pred_labels)
sns.heatmap(cm , annot = True, cmap='YlGnBu',
            fmt = 'd',xticklabels = class_labels,
            yticklabels = class_labels)
```



```
def model_deep():
    model = Sequential([
        Dense( 32, input_dim=X_train.shape[1], activation = 'relu'),
        Dense( units = 32, activation= 'relu'),
        Dense( num_classes, activation= 'softmax')
    ])
    model.compile(loss = 'sparse_categorical_crossentropy', optimizer = 'adam', metrics = ['accuracy'])
    return model

model = model_deep()
history = model.fit(X_train,y_train, epochs=100, validation_split=0.2, callbacks=[PlotLossesKeras()])
```

DNN : 깊은 은닉층



```
model.evaluate(X_test, y_test)
```

```
76/76 [=====] - 0s 2ms/step  
[0.06313834339380264, 0.9851239919662476]
```

loss: 0.0631 - accuracy: 0.9851

Model_shallow2()

```
def model_shallow2():
    model = Sequential([
        Dense(16, input_dim=X_train.shape[1], activation = 'relu'),
        Dense(16, activation = 'relu'),
        Dense(num_classes, activation= 'softmax')
    ])
    model.compile(loss = 'sparse_categorical_crossentropy', optimizer = 'adam', metrics = ['accuracy'])
    return model
```

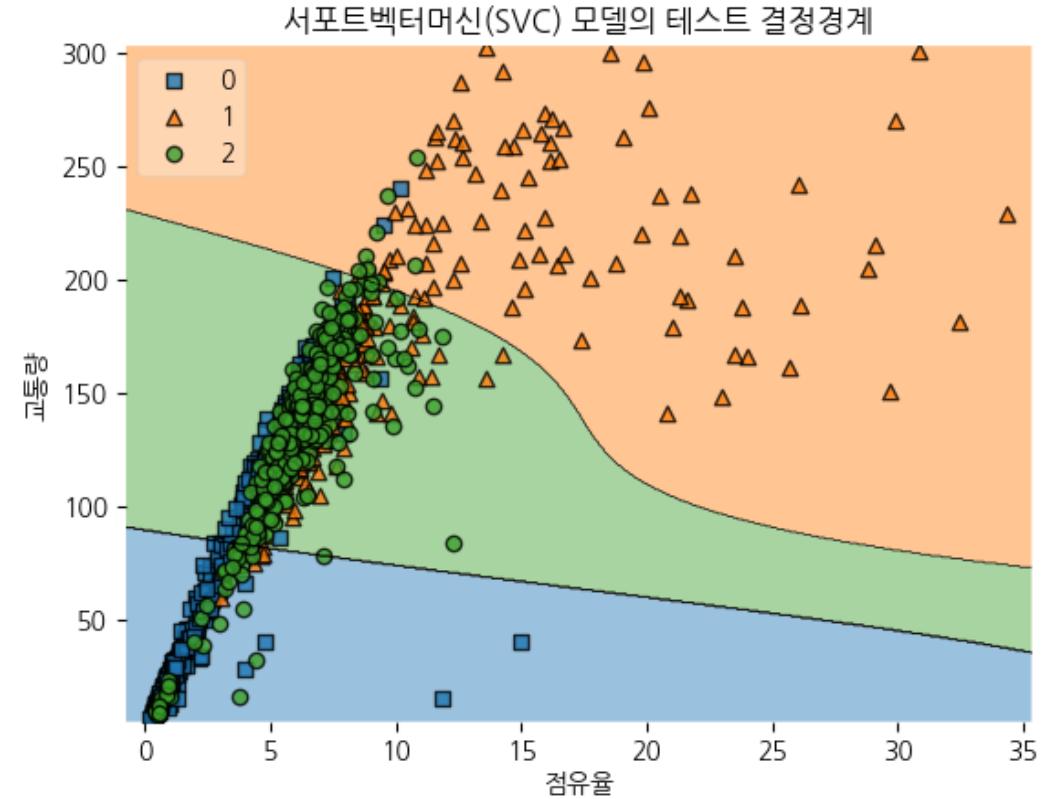
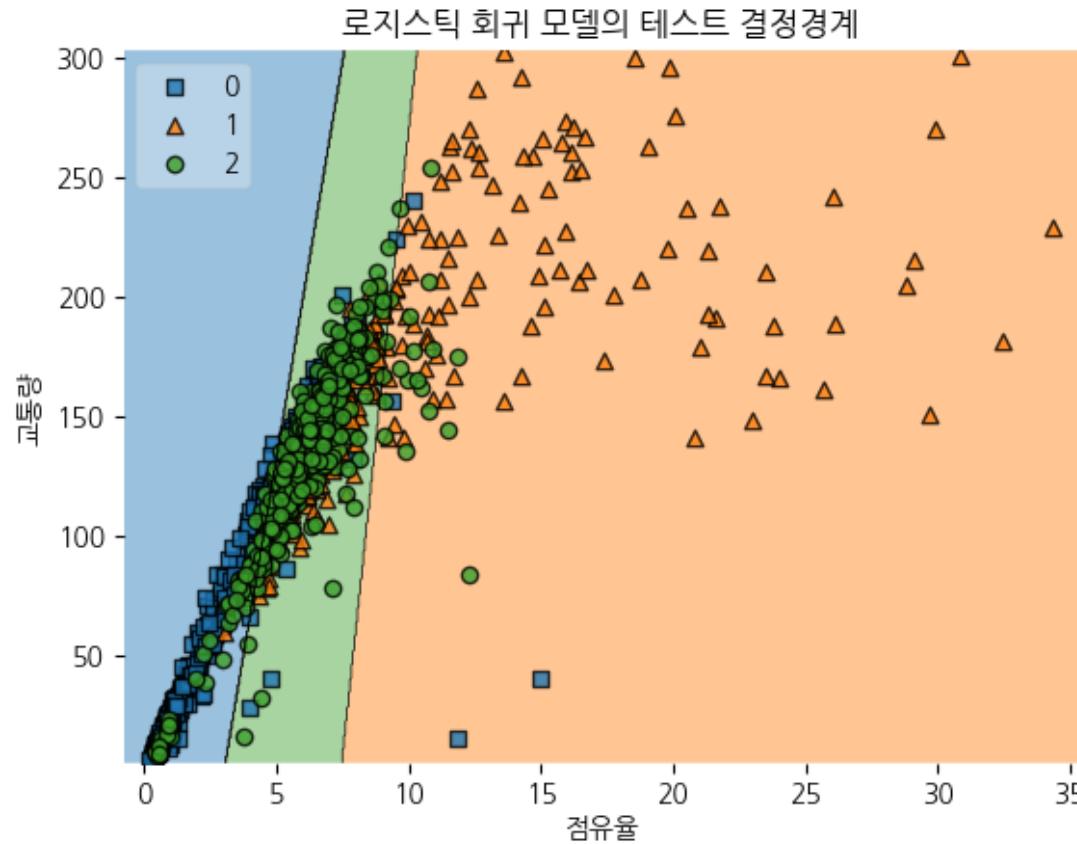
```
model = model_shallow2()
history = model.fit(X_train,y_train, epochs=100, validation_split=0.2, callbacks=[PlotLossesKeras()])
```

```
model.evaluate(X_test, y_test)
```

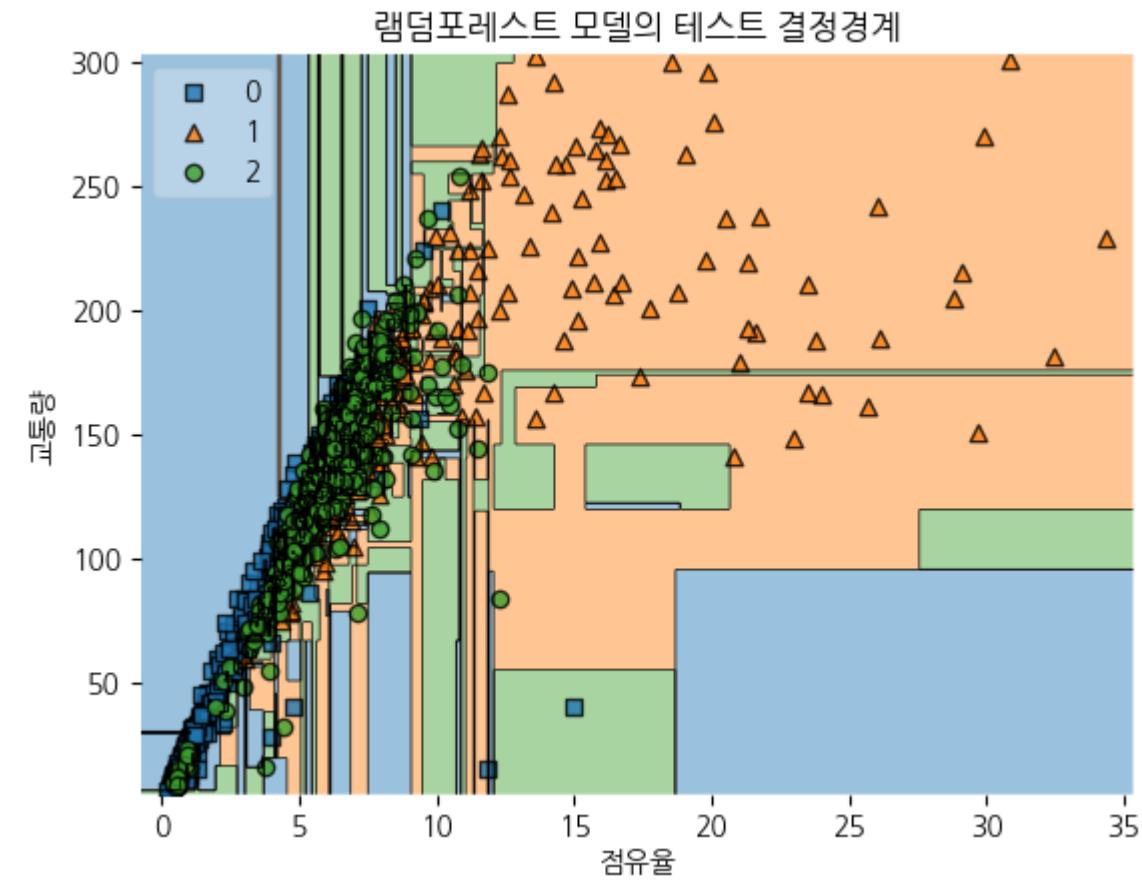
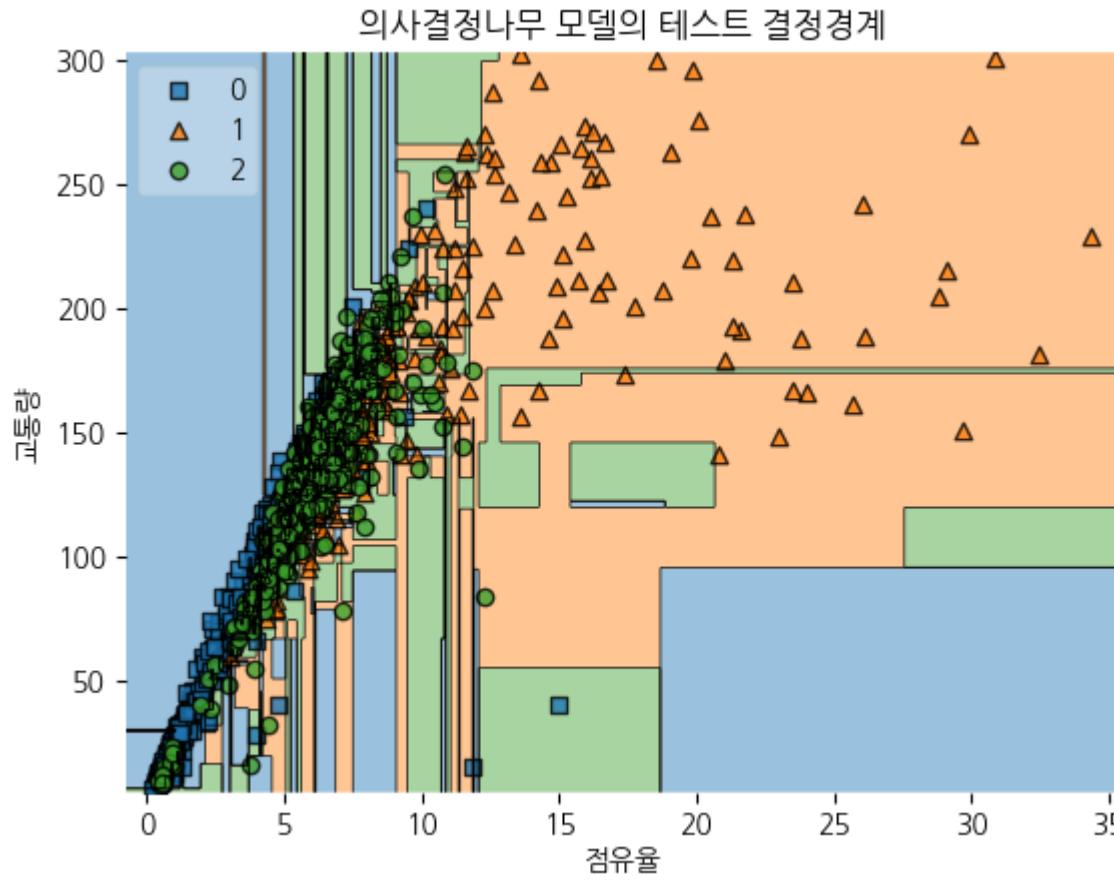
```
76/76 [=====] - 0s 2ms/step - loss: 0.0152 - accuracy: 0.9942
```

```
[0.015153169631958008, 0.9942148923873901]
```

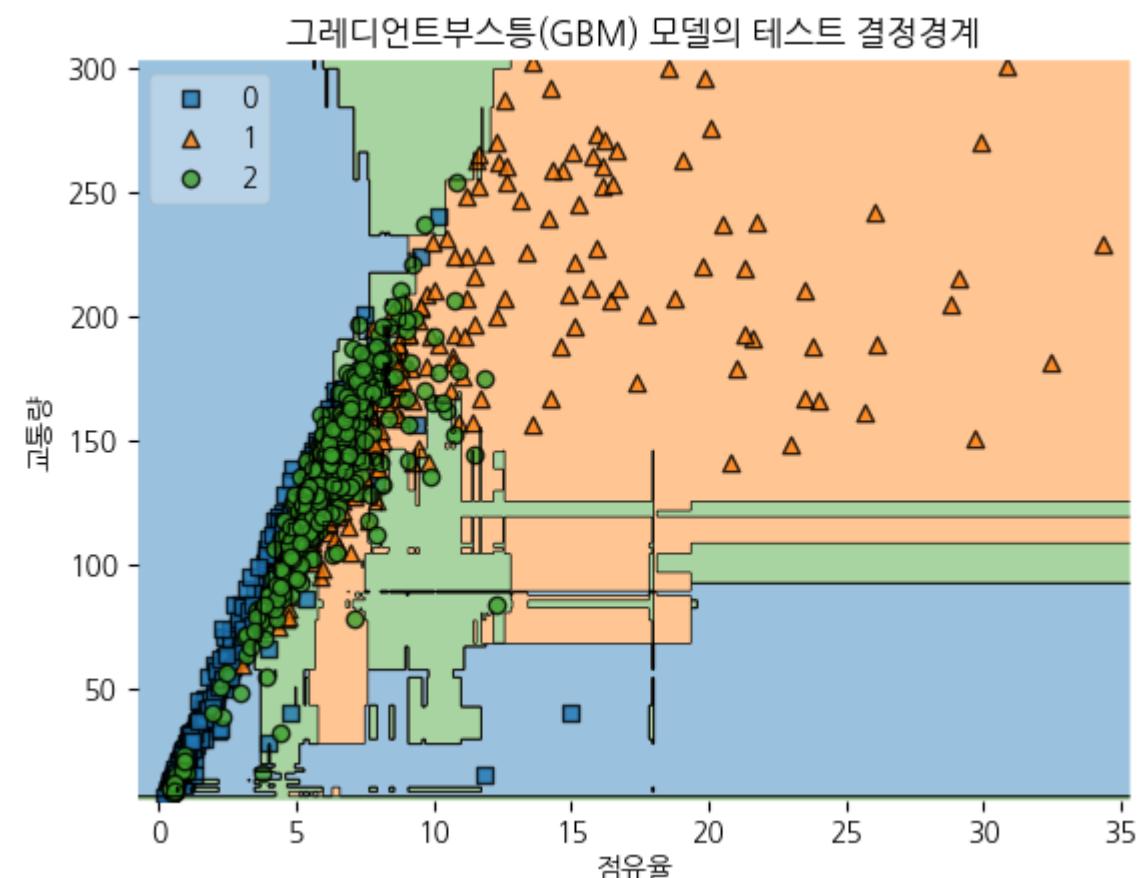
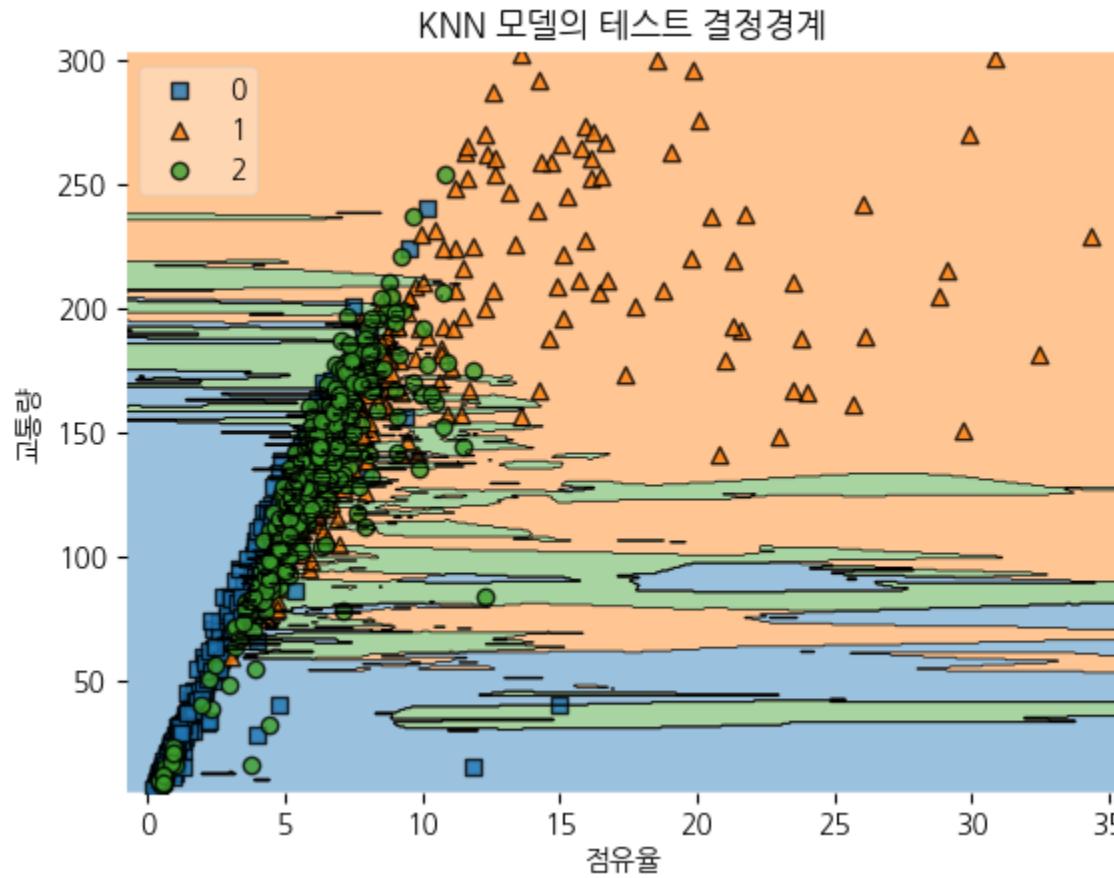
로지스틱 및 SVC 결정경계 : 교통 흐름데이터



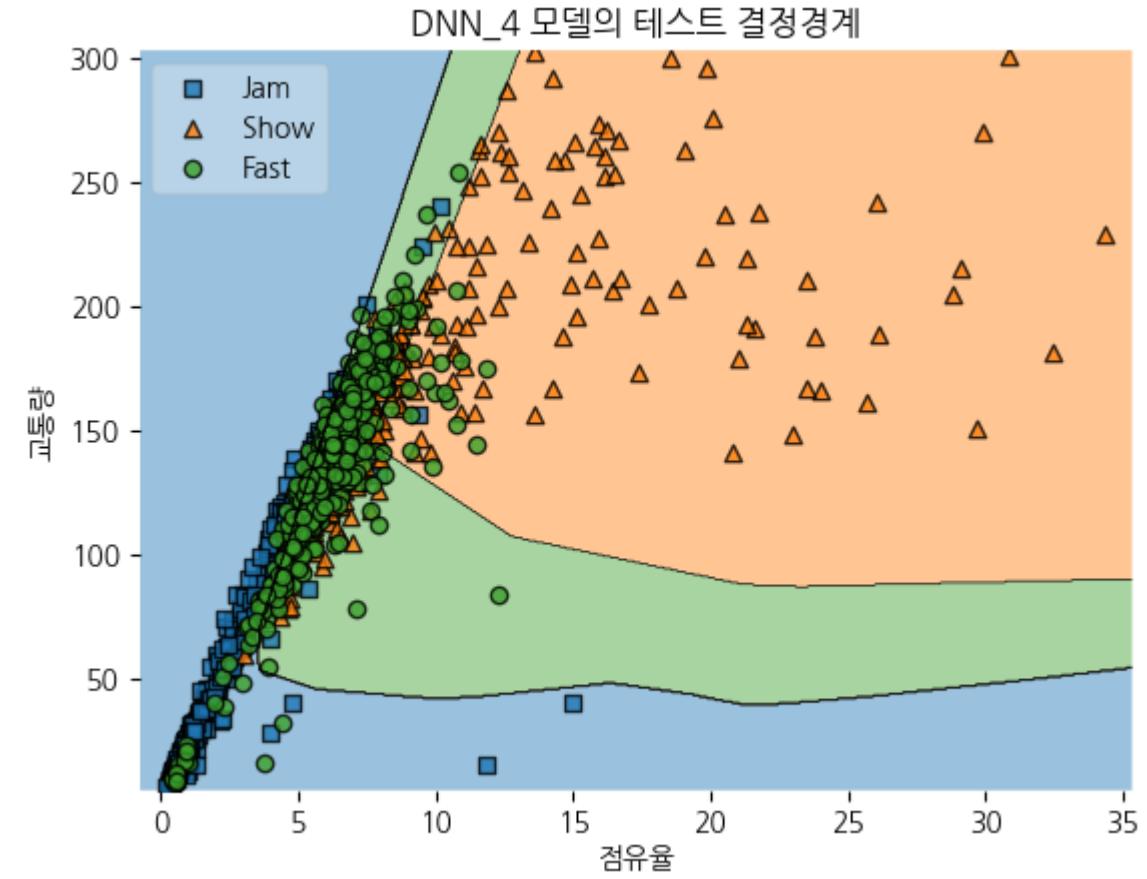
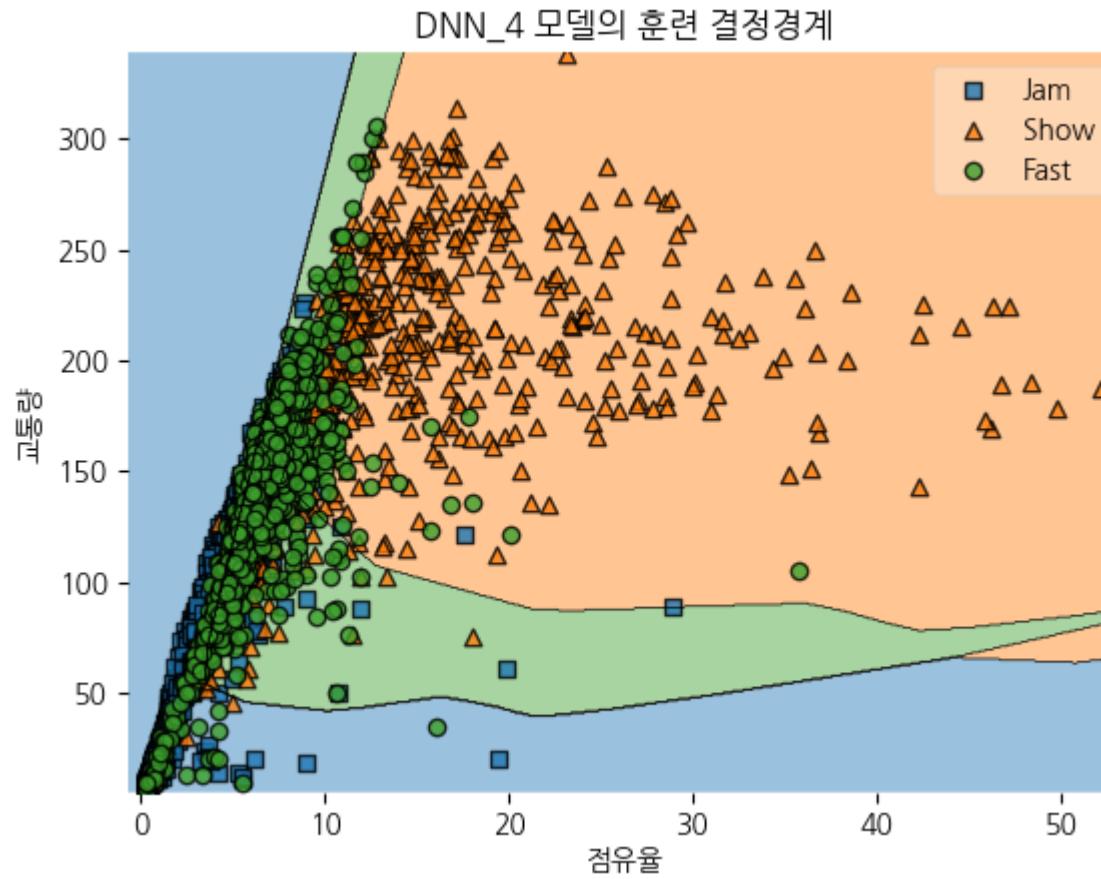
DT와 RF 결정경계 : 교통 흐름데이터



KNN 및 GBM 결정경계 : 교통 흐름데이터



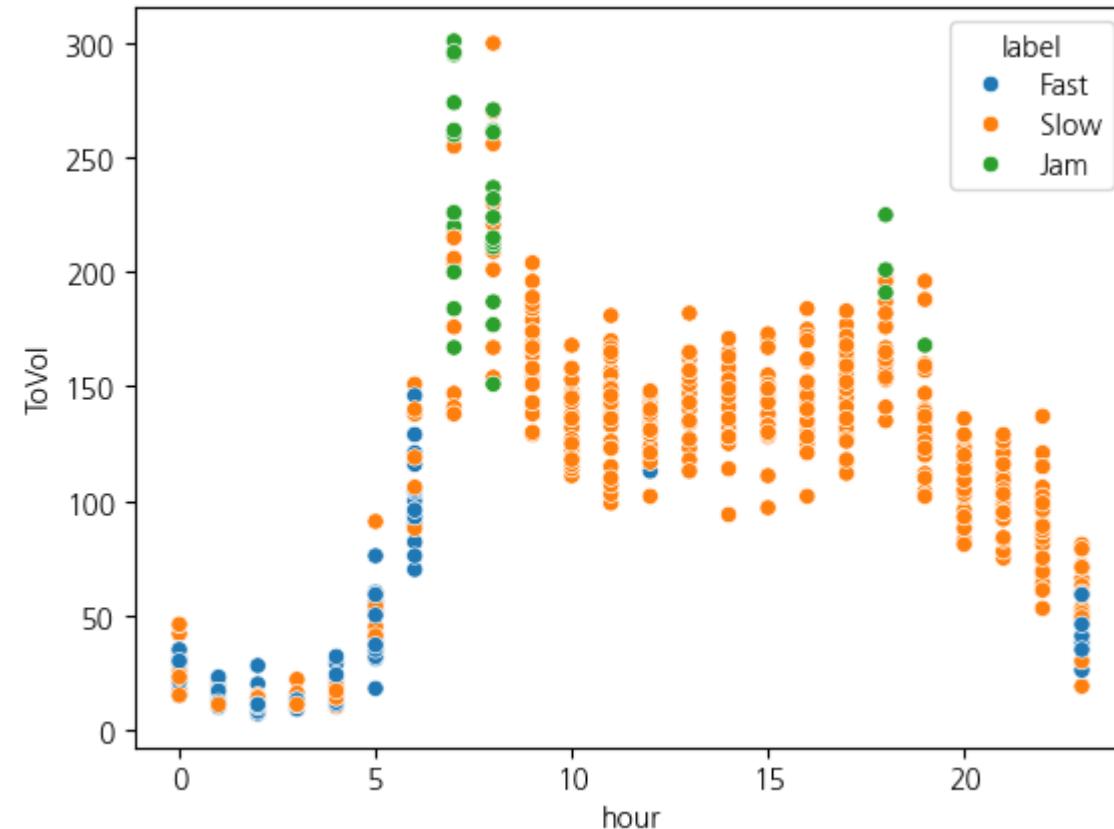
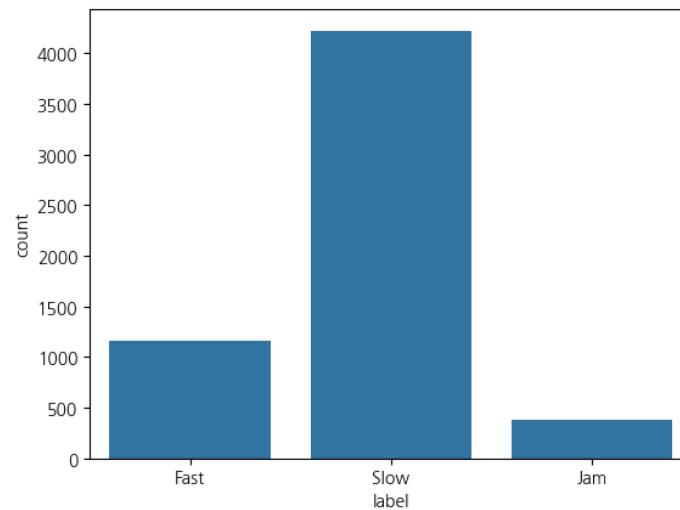
딥러닝 모델 (DNN) 결정경계 : 교통 흐름데이터



방법 2: 레이블을 다시 정해보자

❖ 속도로 레이블 판단

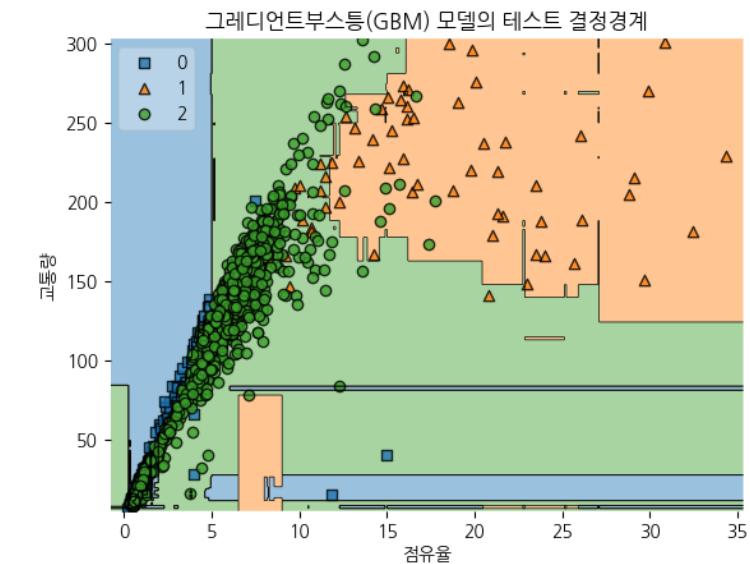
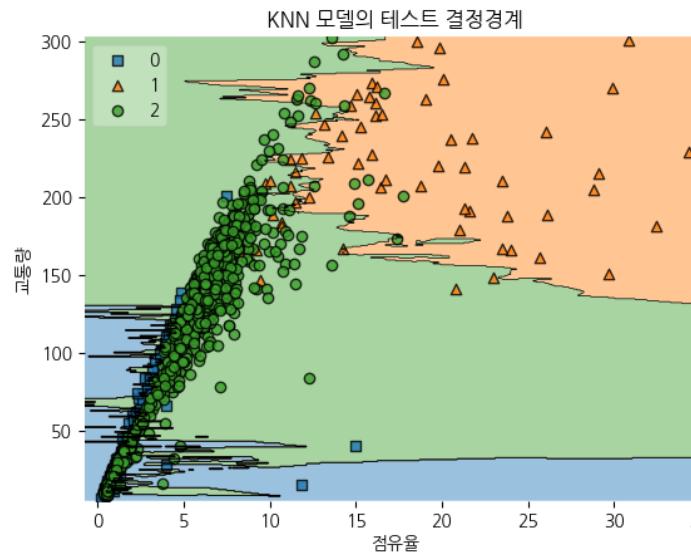
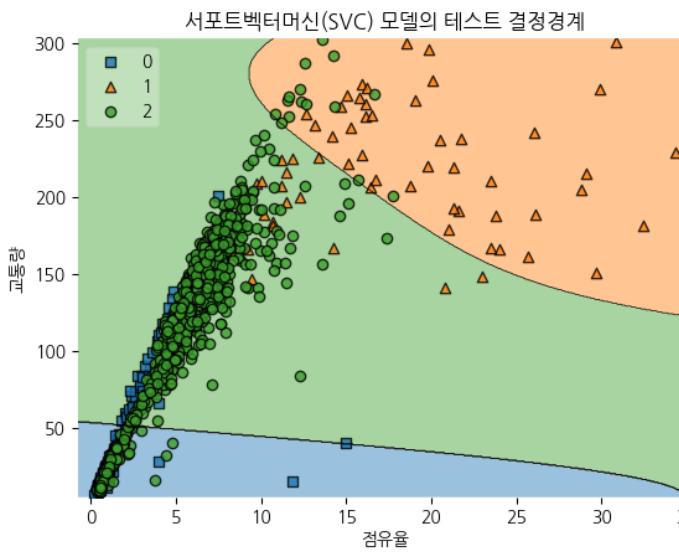
- ✓ 40 이하 Jam
- ✓ 55 이상 Fast
- ✓ 그 사이 Slow



방법 2: 레이블을 다시 정해보자

❖ 속도로 레이블 판단

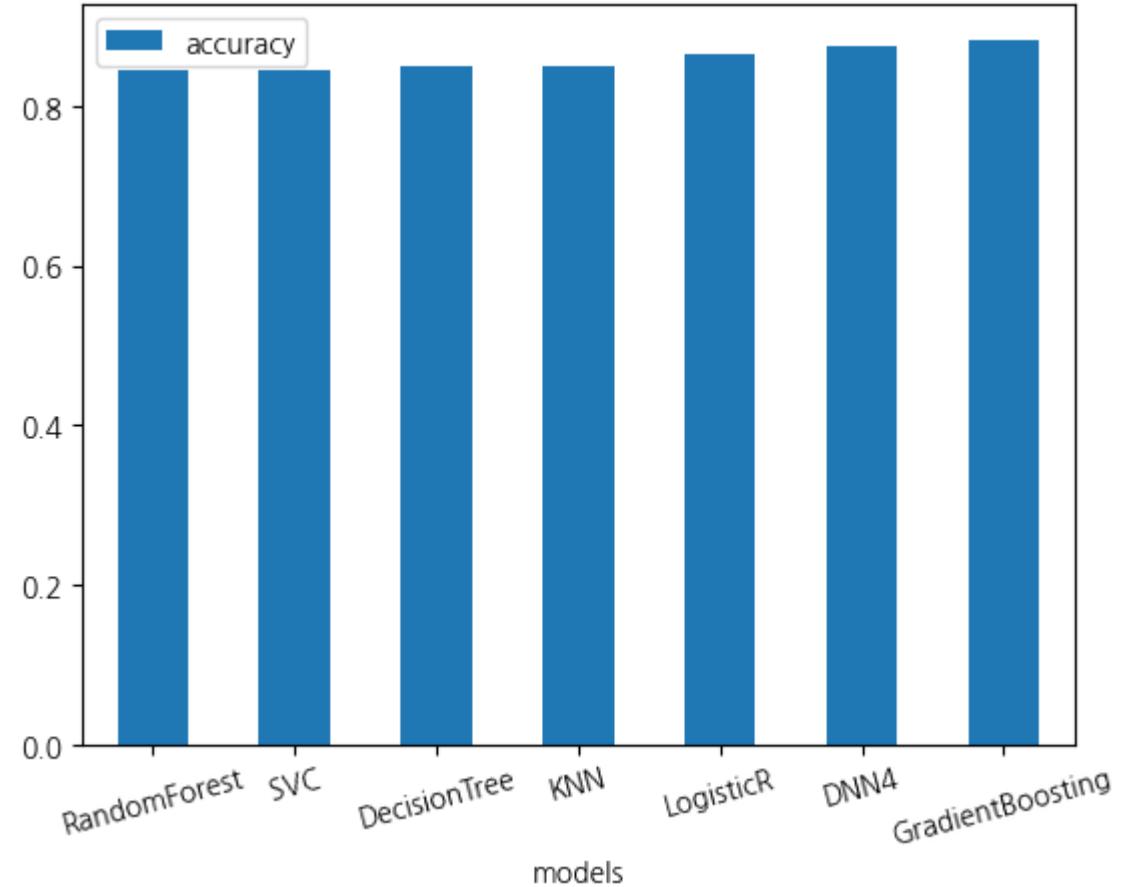
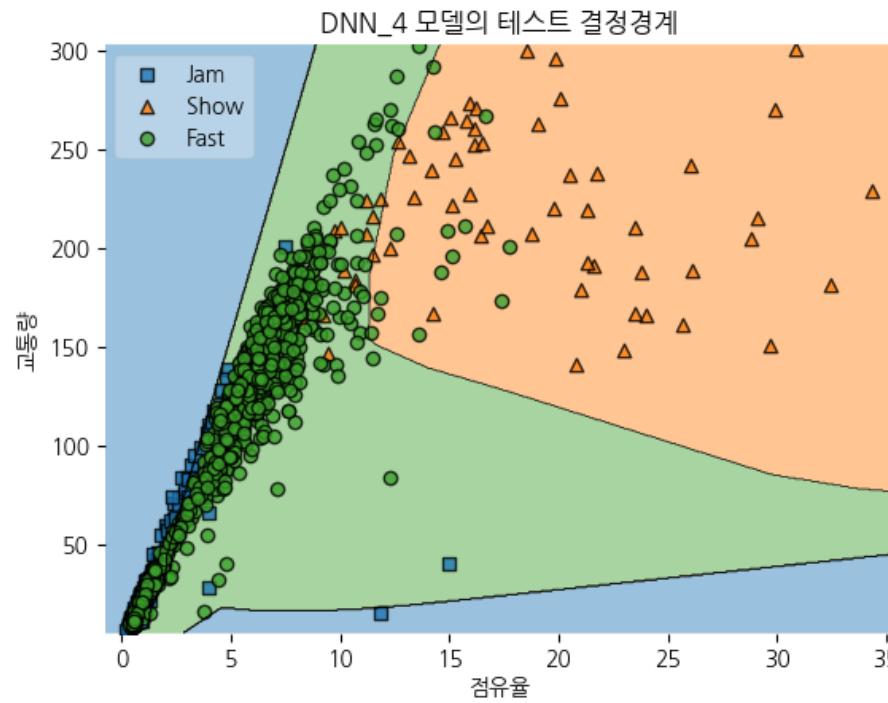
- ✓ 40 이하 Jam
- ✓ 55 이상 Fast
- ✓ 그 사이 Slow



방법 2: 레이블을 다시 정해보자

❖ 속도로 레이블 판단

- ✓ 40 이하 Jam
- ✓ 55 이상 Fast
- ✓ 그 사이 Slow



2-3

선형 회귀와 로지스틱 회귀 소개

❖ 회귀의 역사

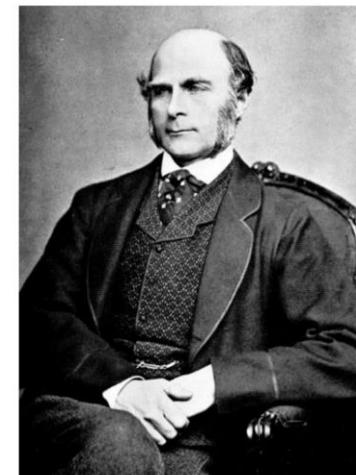
- ✓ 영국의 통계학자 갈톤(Galton)의 유전적 특성중에 부모와 자식의 키 관계
 - “사람의 키는 평균 키로 회귀(Regression)하려는 경향을 가진다는 자연의 법칙이 있다”
- ✓ 회귀 분석은 데이터 값이 평균과 같은 일정한 값으로 돌아가려는 경향을 이용한 통계학 기법

❖ 지도학습은 2가지 유형으로 나눔

- ✓ 회귀
 - 연속적인 숫자 값
- ✓ 분류
 - 예측값이 카테고리와 같은 이산형 클래스 값

❖ 선형회귀

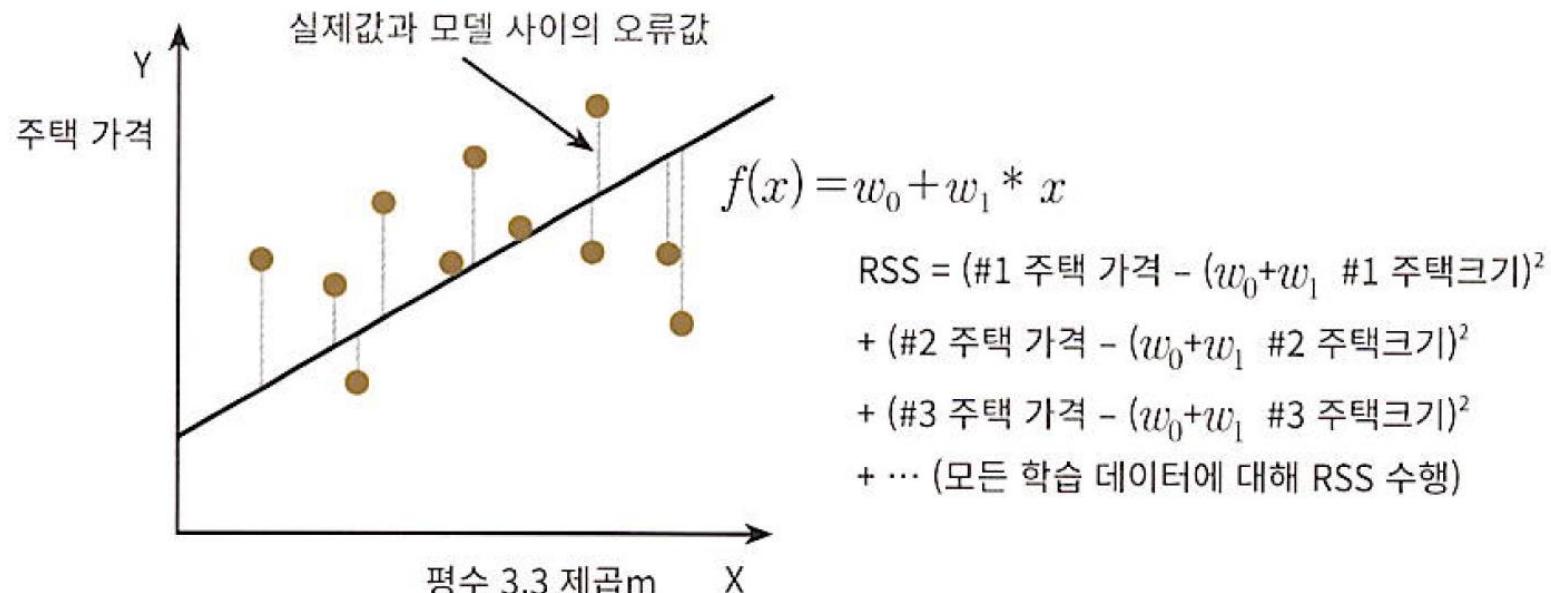
- ✓ 실제 값과 예측 값의 차이를 최소화하는 직선형 회귀
 - 주택 가격이 주택의 크기로만 결정



Sir Francis Galton (1822 ~ 1911)

❖ 단순 선형회귀

- ✓ 1개의 독립변수, 1개의 종속변수



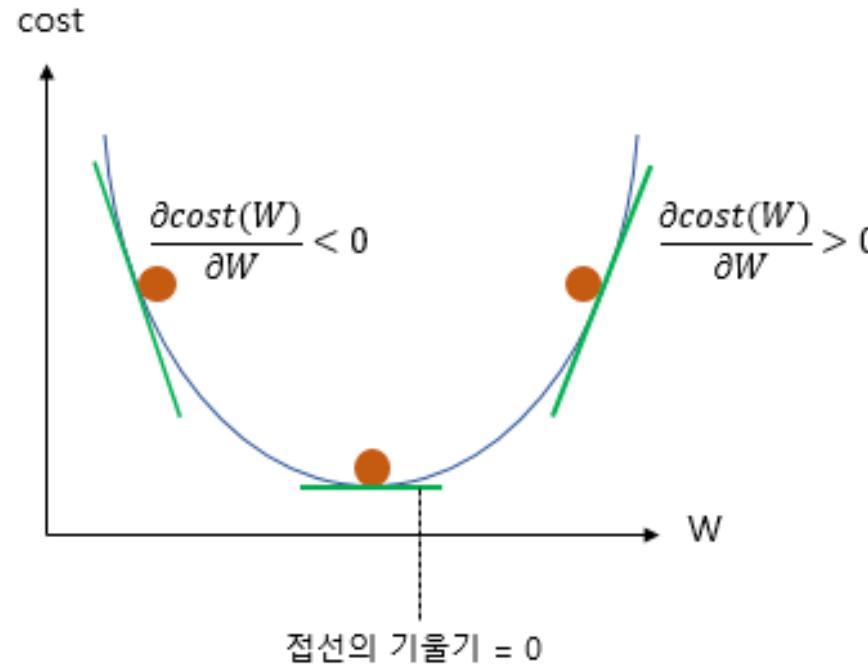
$$RSS(w_0, w_1) = \frac{1}{N} \sum_{i=1}^N (y_i - (w_0 + w_1 \times x_i))^2$$

❖ 옵티마이저(Optimizer) 또는 최적화 알고리즘

- ✓ 머신 러닝 학습은 결국 비용 함수를 최소화하는 가중치, 편향을 찾기 위한 작업을 수행

$$w_{\text{updated}} \approx w - \alpha \cdot \frac{\partial \text{cost}}{\partial w}$$

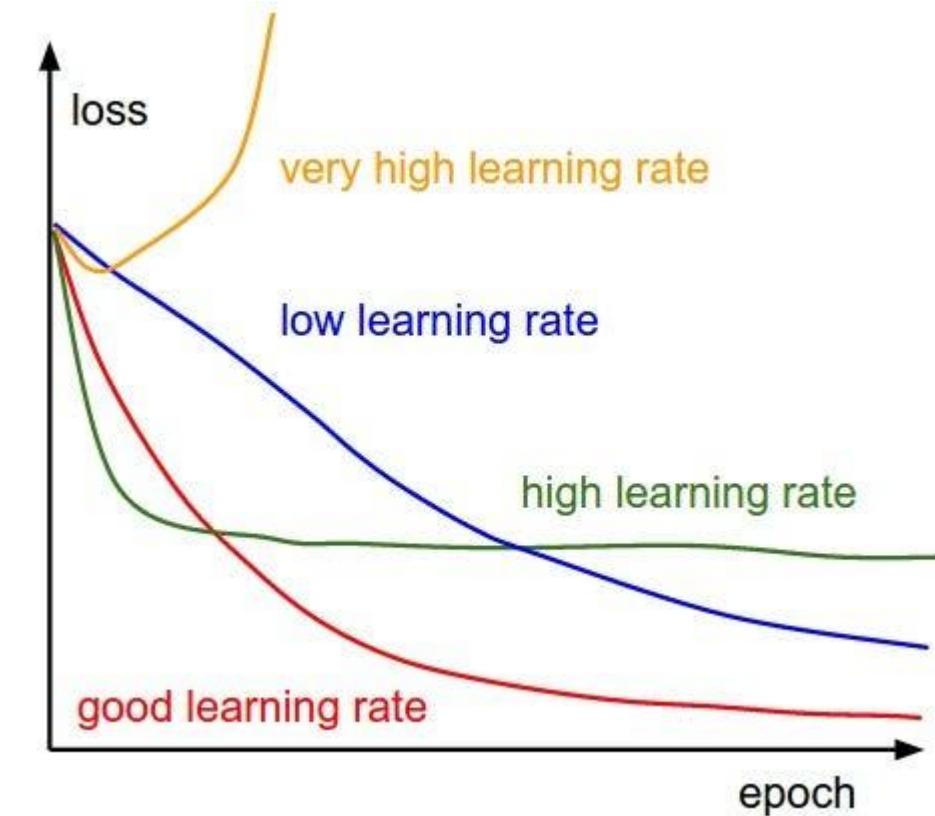
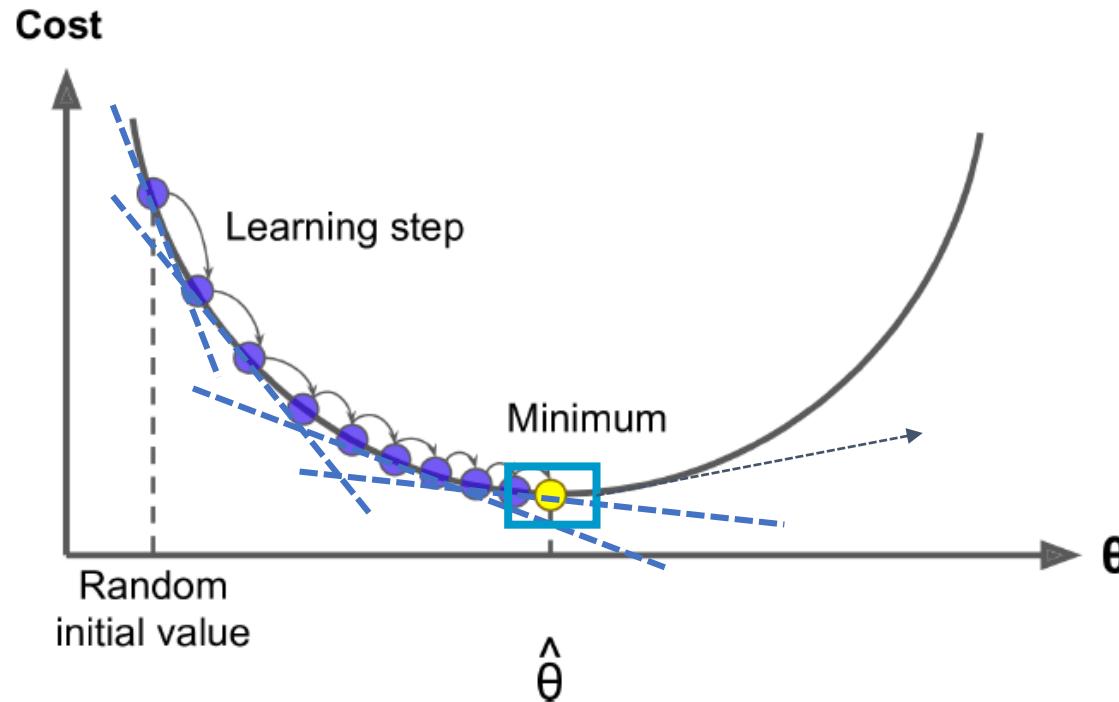
$$W, b \rightarrow \text{minimize } \text{cost}(W, b)$$



경사 하강법 (Gradient Descent)

$\eta \sim \text{learning rate}$

$$w = w - \eta \frac{2}{N} \sum_{i=1}^N (\text{real}_i - \text{pred}_i)$$



<https://towardsdatascience.com/https-medium-com-dashingaditya-rakhecha-understanding-learning-rate-dd5da26bb6de>

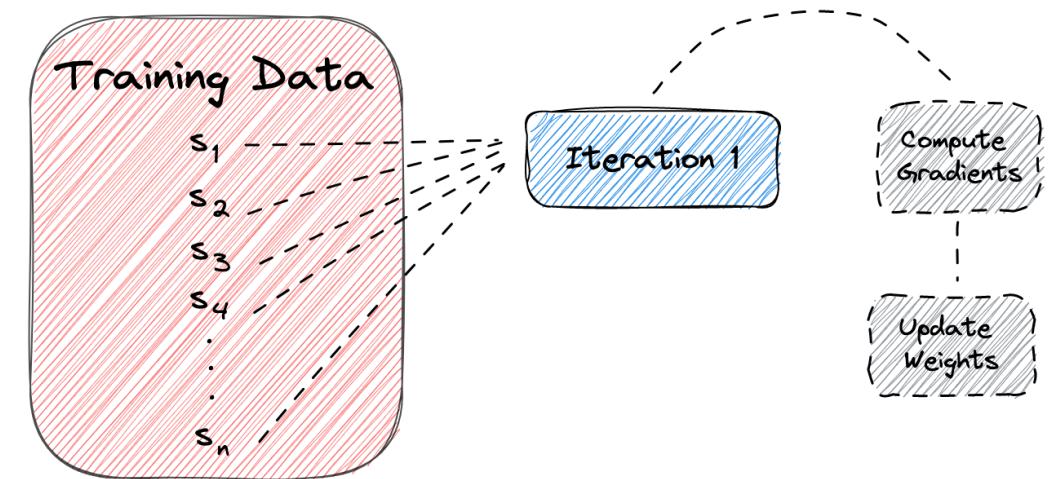
배치와 이포크(Epoch) 개념

- 전체 학습 데이터를 한 번씩 사용 한 것을 '에폭'이라 함
- 훈련 데이터 묶음을 배치라고 한다.

Batch Gradient



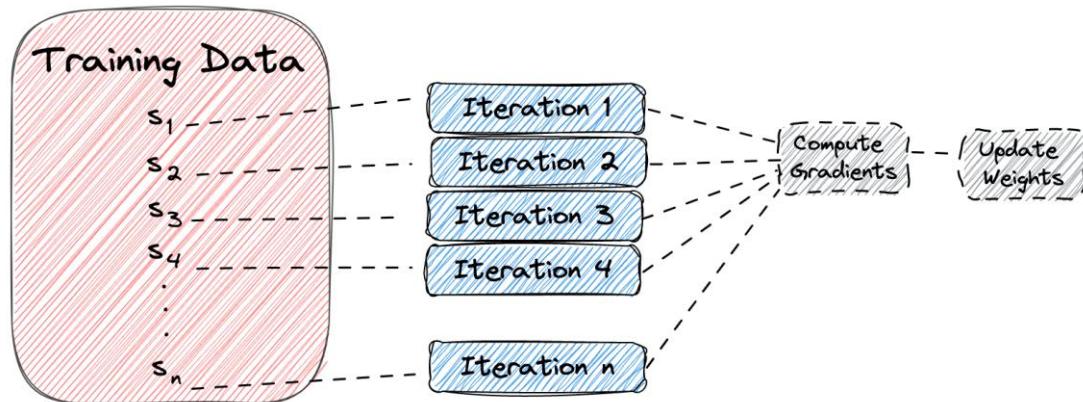
<https://otugi.tistory.com/m/350>



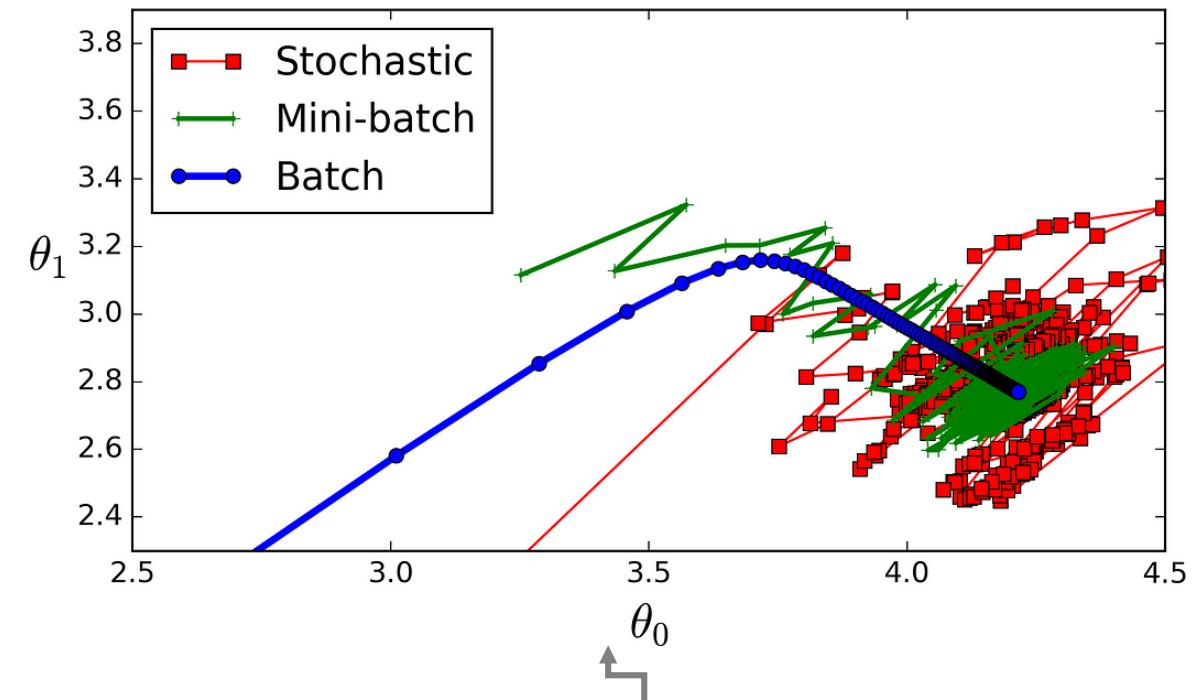
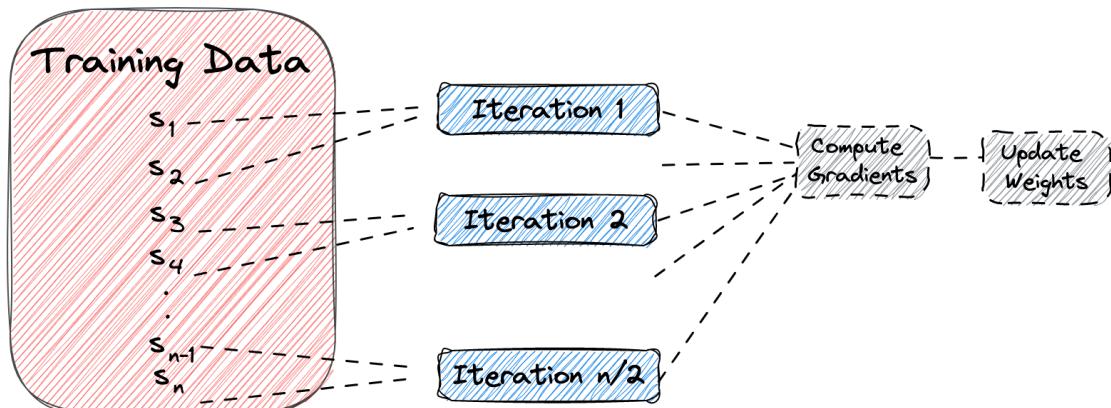
(source) <https://www.baeldung.com/cs/epoch-vs-batch-vs-mini-batch#1-batch-gradient-descent>

Types of Gradient Descent

Stochastic Gradient



Mini-Batch Gradient



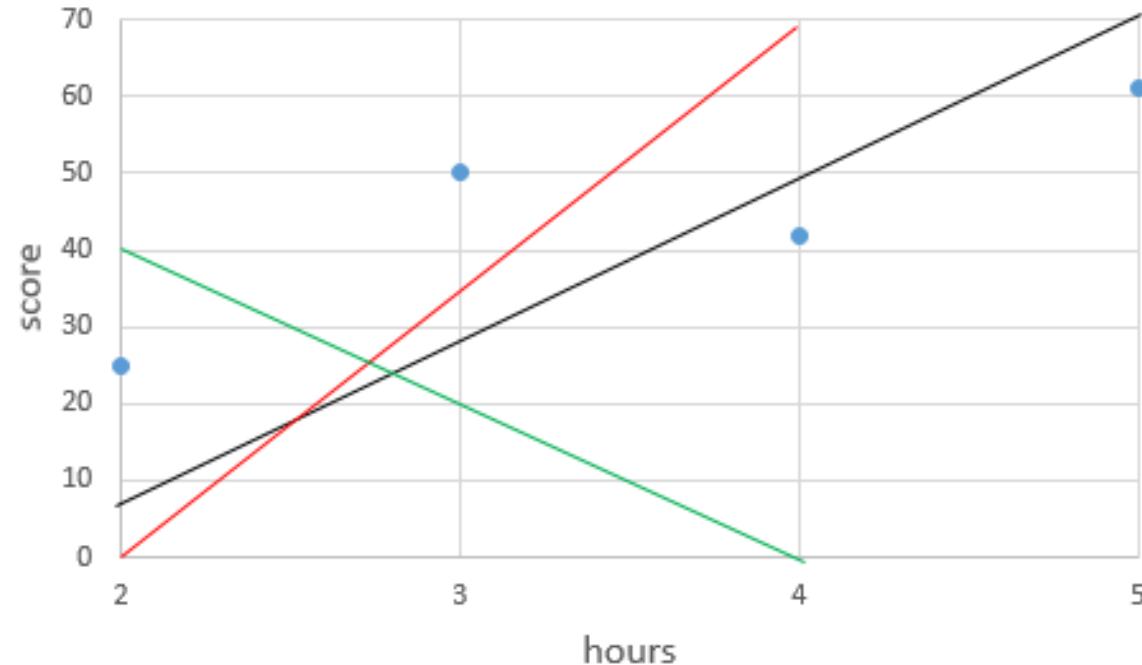
(source) <https://sweta-nit.medium.com/batch-mini-batch-and-stochastic-gradient-descent-e9bc4cacd461>

(source) <https://www.baeldung.com/cs/epoch-vs-batch-vs-mini-batch#1-batch-gradient-descent>

선형 회귀 : 가설(Hypothesis)

- ❖ 어떤 학생의 공부 시간에 따라서 얻은 점수 데이터

| hours(x) | score(y) |
|--------------|--------------|
| 2 | 25 |
| 3 | 50 |
| 4 | 42 |
| 5 | 61 |



$$H(x) = Wx + b$$

케라스로 구현하는 선형 회귀

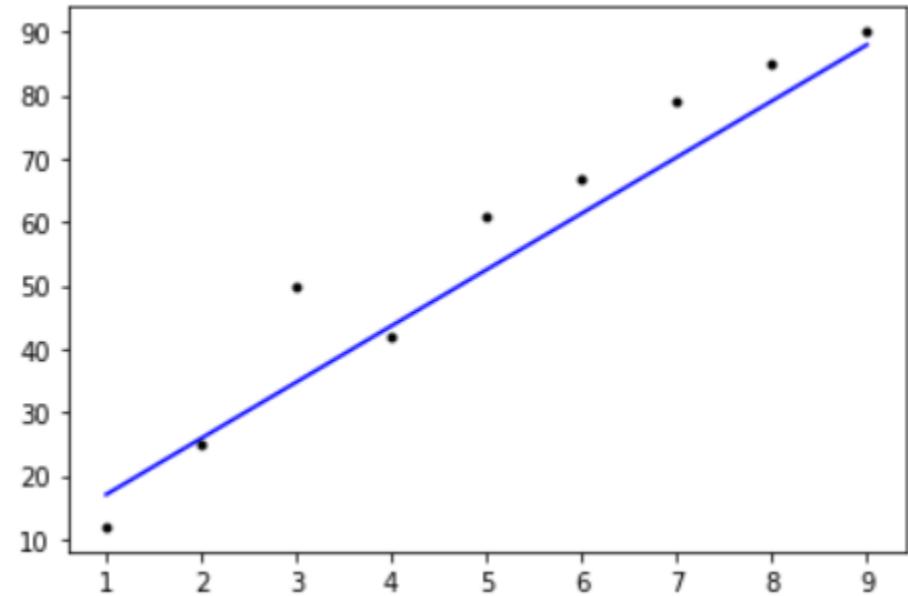
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras import optimizers

X=np.array([1,2,3,4,5,6,7,8,9])
# 공부하는 시간
y=np.array([12,25,50,42,61, 67, 79, 85, 90])

model = Sequential()

model.add(Dense(1, input_dim=1, activation='linear'))
sgd = optimizers.SGD(lr=0.01)
model.compile(optimizer=sgd ,loss='mse',metrics=['mse'])
history=model.fit(X,y, batch_size=1, epochs=30, shuffle=False)

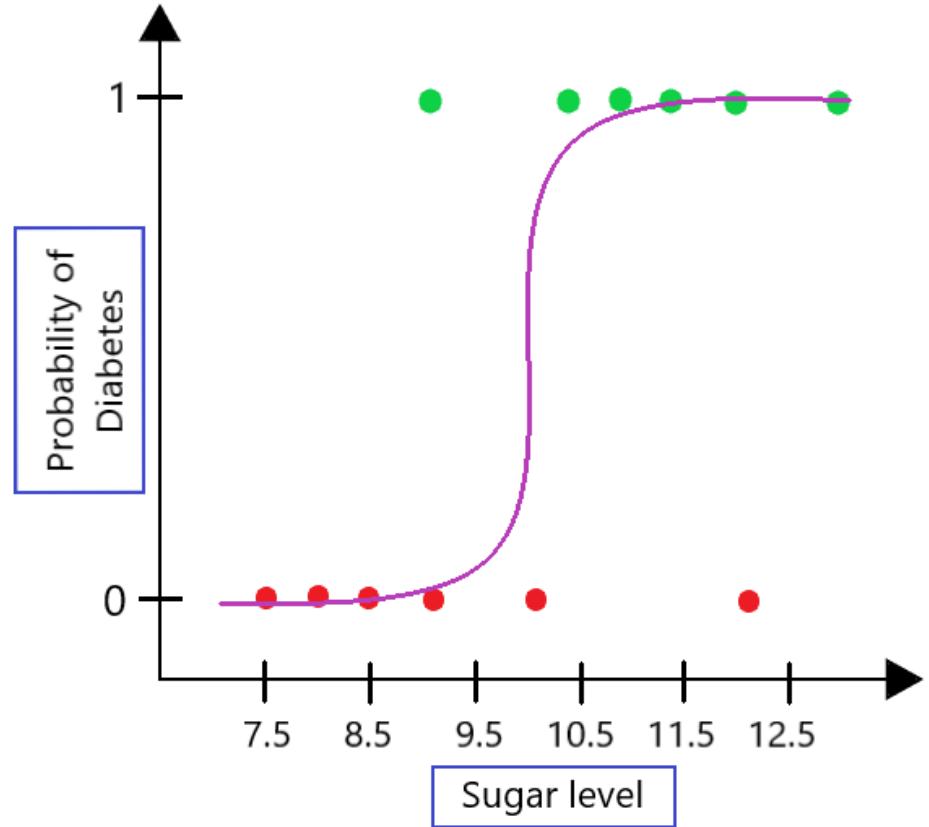
plt.plot(X, model.predict(X), 'b', X,y, 'k.')
```



비선형 회귀 문제

이진 분류(Binary Classification): 둘 중 하나를 결정하는 문제

이런 문제를 풀기 위한 알고리즘으로 로지스틱 회귀(Logistic Regression)



비용함수는 볼록한 모양이 아니고,
여러 개의 로컬미니엄이 있는 모양이다.



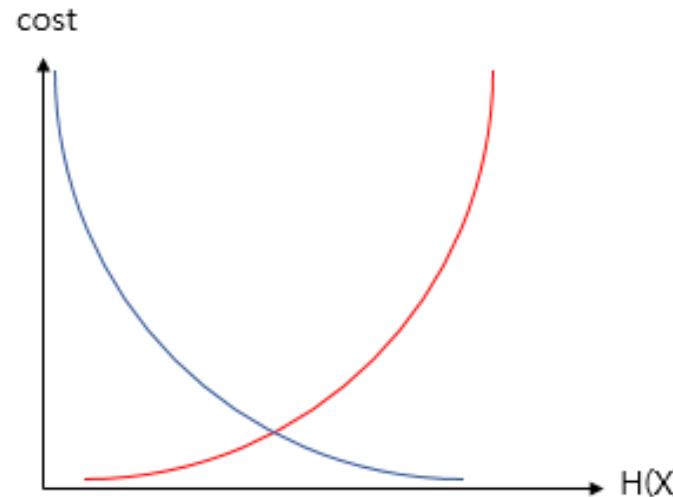
<https://www.geeksforgeeks.org/ml-cost-function-in-logistic-regression/>

Cross Entropy

분류 평가 지표로 Log Loss (Cross Entropy)를 고려하자.

$$J(W) = -\frac{1}{n} \sum_{i=1}^n [y^{(i)} \log H(x^{(i)}) + (1 - y^{(i)}) \log(1 - H(x^{(i)}))]$$

$$J(W) = \frac{1}{n} \sum_{i=1}^n \text{cost}(H(x^{(i)}), y^{(i)})$$

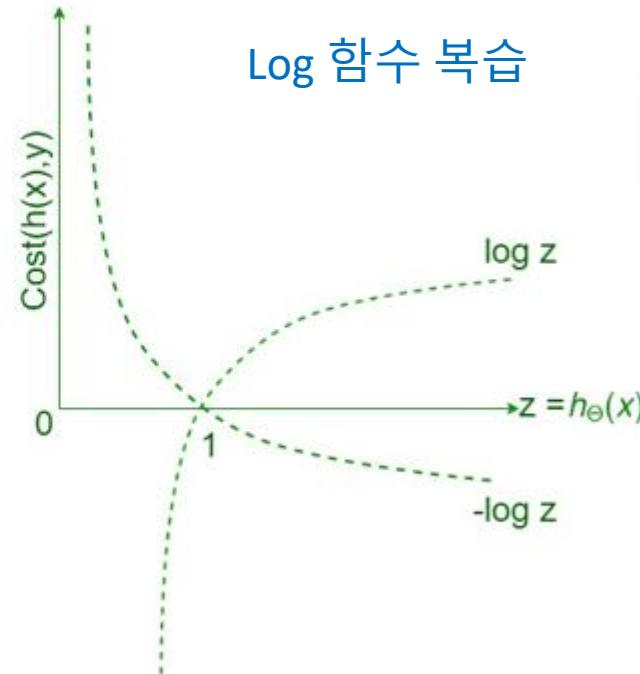
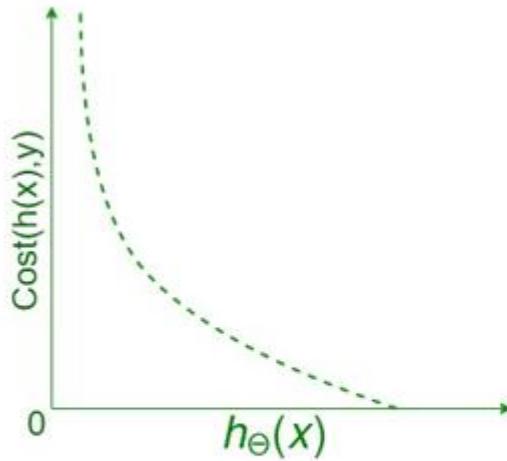


$$\text{cost}(H(x), y) = -[y \log H(x) + (1 - y) \log(1 - H(x))]$$

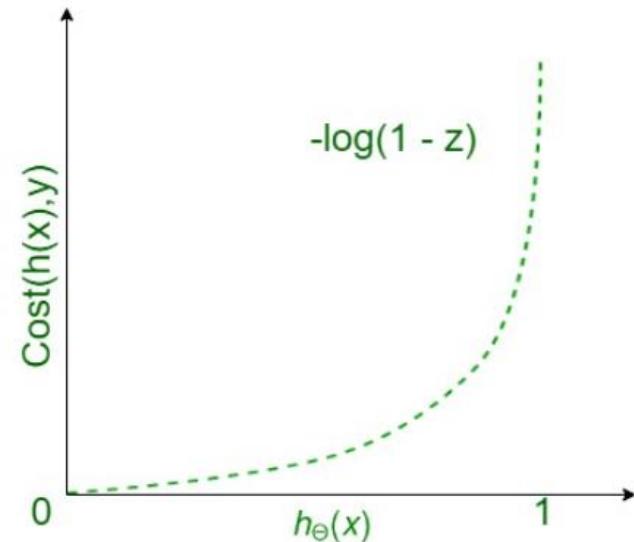
$$H(X) = \frac{1}{1+e^{-(Wx+b)}} = \sigma(Wx + b)$$

로지스틱 회귀의 비용함수

if $y = 0 \rightarrow \text{cost}(H(x), y) = -\log(1 - H(x))$



if $y = 1 \rightarrow \text{cost}(H(x), y) = -\log(H(x))$



<https://www.geeksforgeeks.org/ml-cost-function-in-logistic-regression/>

선형회귀 최적화 과정

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

$$h_\theta(x) = \theta^\top x$$

비선형회귀 최적화 과정

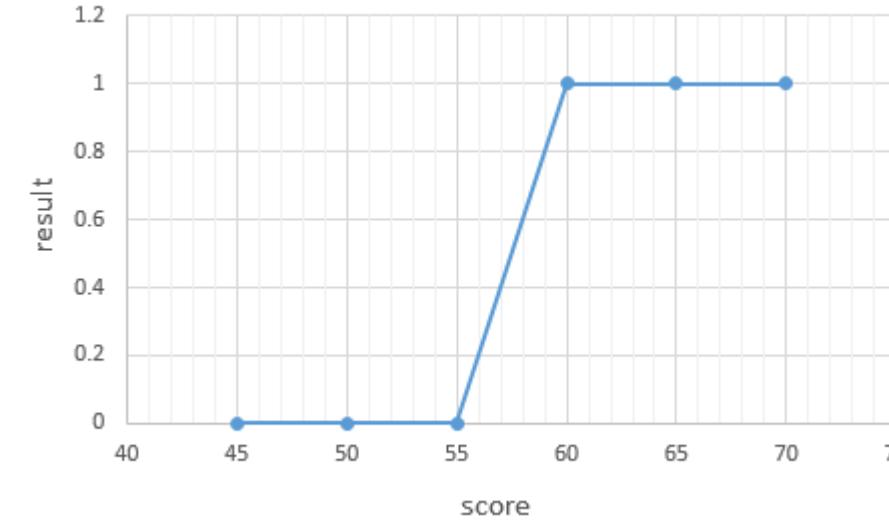
$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^\top x}}$$

비선형회귀 예제: 이진 분류(Binary Classification)

- ❖ 학생들이 시험 성적에 따라서 합격, 불합격 데이터

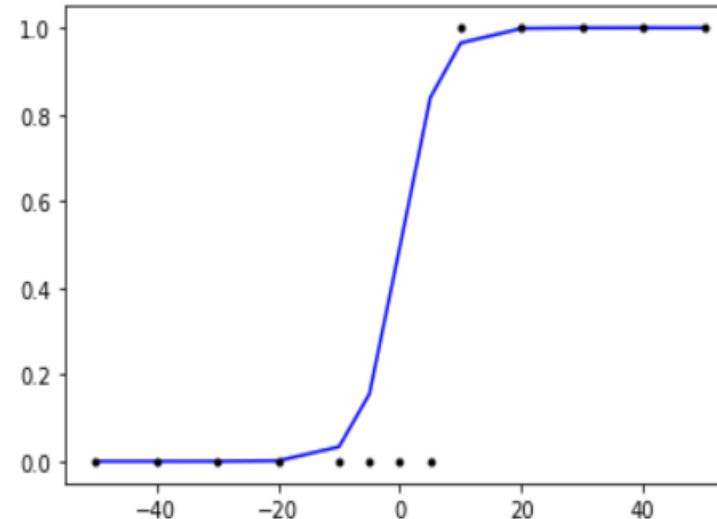
| score(x) | result(y) |
|----------|-----------|
| 45 | 불합격 |
| 50 | 불합격 |
| 55 | 불합격 |
| 60 | 합격 |
| 65 | 합격 |
| 70 | 합격 |



```
X=np.array([-50, -40, -30, -20, -10, -5, 0, 5, 10, 20, 30, 40, 50])
y=np.array([0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1])
```

Tensorflow 2.x로 시작하는 로지스틱 회귀

```
import numpy as np  
  
from tensorflow.keras.models import Sequential  
from tensorflow.keras.layers import Dense  
from tensorflow.keras import optimizers  
  
model=Sequential()  
  
model.add(Dense(1, input_dim=1, activation='sigmoid'))  
sgd=optimizers.SGD(lr=0.01)  
  
model.compile(optimizer=sgd ,  
              loss='binary_crossentropy',metrics=['binary_accuracy'])  
  
history = model.fit(X,y, epochs=20, shuffle=False)  
  
plt.plot(X, model.predict(X), 'b', X,y, 'k.')
```



2-4

DNN 회귀 소개 및 실습

❖ 분류 모델인가 회귀 모델인가?

- ✓ 라벨을 사용하지 않으면 회귀모델
- ✓ DNN 회귀 모델의 특성과 분류기(?)는 어떻게 정할 것인가?
 - 특성으로 교통량, 점유율, 속도
 - 라벨은 3개 Slow, Jam, Fast로 이미 전처리한 데이터가 있다.

❖ 풀고자 하는 문제를 명확히 하고 모델을 만들자

- ✓ (예) 교통량을 특성(입력)으로 하여 속도를 알아내보자(예측)
- ✓ (예) 속도와 점유율을 입력으로 하여 교통량을 예측해 보자

❖ 특성과 타깃을 만들어 주어야 한다.

- ✓ 타깃의 속성이 연속적인 숫자이면 Dense 층의 활성함수를 선형활성함수를 사용한다.
 - Linear(), ReLU() 등
- ✓ 타깃이 범주형(Categorical)이면 Softmax() 활성함수
 - Sortmax() 활성

코랩에서 github 데이터 읽어오기

```
import pandas as pd
```

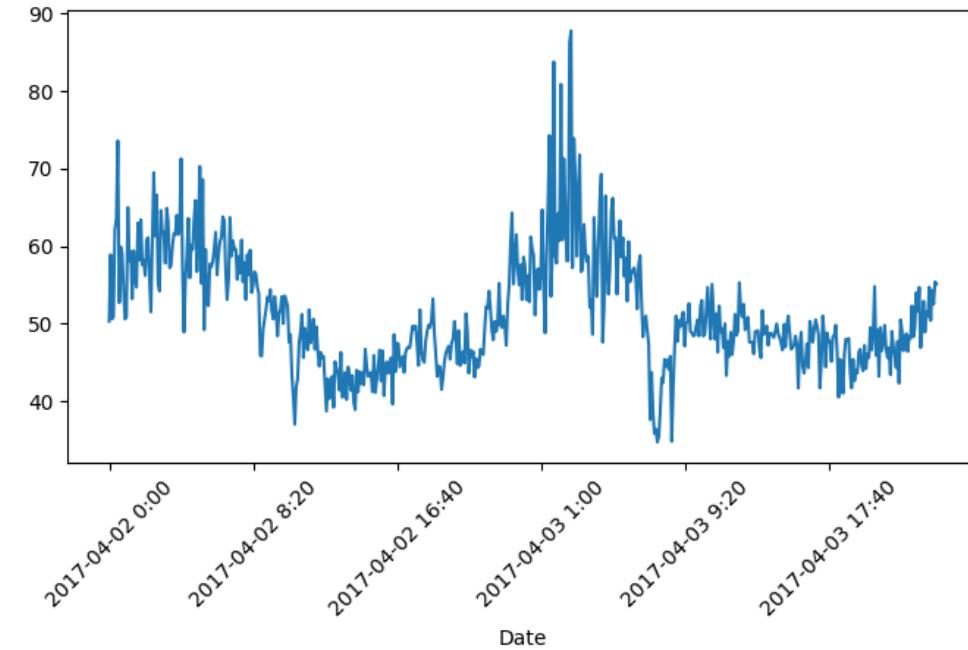
```
df = pd.read_csv('https://raw.githubusercontent.com/hongsukyi/Lectures/main/vds/vds.csv')
```

```
df.set_index('Date', inplace=True)
```

```
df.head(2)
```

| Date | ToVol | SmVol | MeVol | LaVol | Speed | Occ. Rate |
|-----------------|-------|-------|-------|-------|-------|-----------|
| 2017-04-02 0:00 | 43 | 34 | 9 | 0 | 50.3 | 1.90 |
| 2017-04-02 0:05 | 45 | 32 | 13 | 0 | 58.9 | 1.84 |

```
df['Speed'][:576].plot(rot=45, figsize=(8,4))
```



Multivariate Regression

```
features = ['ToVol', 'LaVol', 'MeVol', 'SmVol', 'Occ.Rate']
X = df[features]
y = df.iloc[:, 4:5].values
```

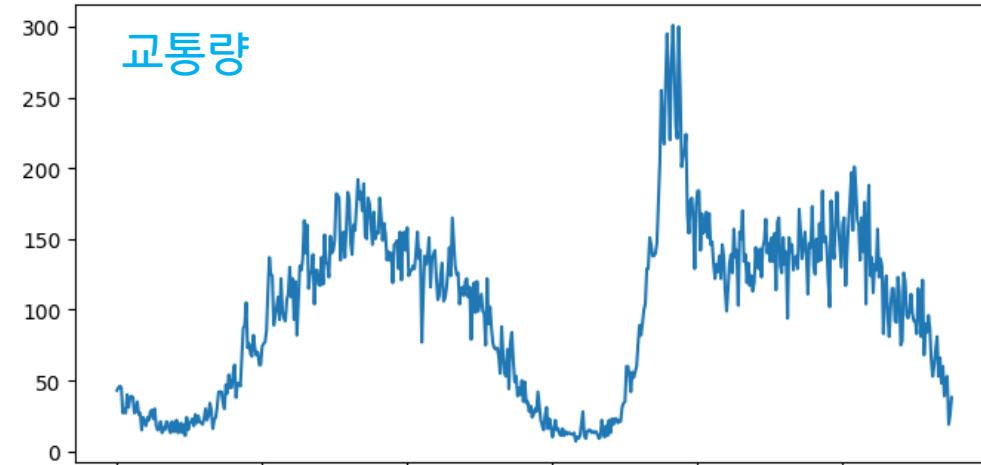
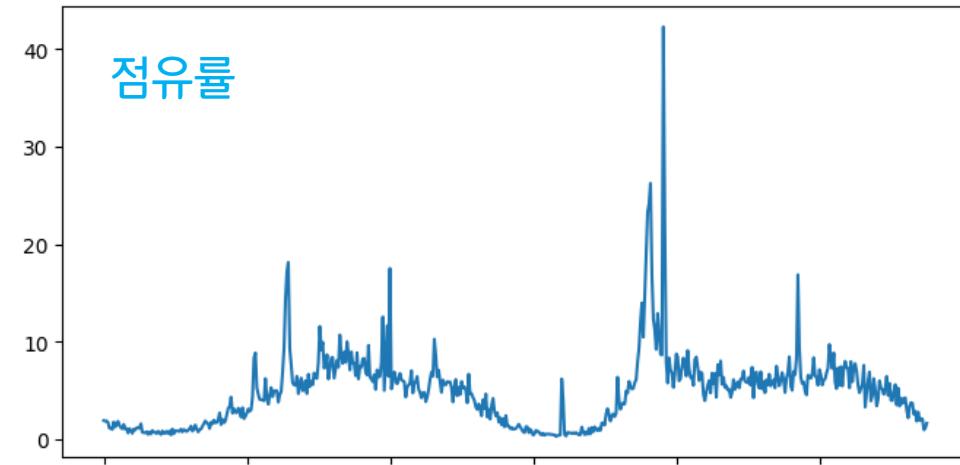
```
print(y)
```

```
[[50.3]
 [58.9]
 [50.6]
 ...
 [50.6]
 [59.3]
 [52.5]]
```

```
max_speed = y.max(); max_speed
```

```
87.8
```

```
df['ToVol'][:576].plot(rot=45, figsize=(8,4))
```



정규화

```
from sklearn.preprocessing import StandardScaler, MinMaxScaler
#scaler = StandardScaler()
scaler = MinMaxScaler()
scaled_X = scaler.fit_transform(X)
scaled_y = scaler.fit_transform(y)
```

```
print(scaled_X)
print(scaled_y)
```



```
[[0.11144578 0.          0.06206897 0.12903226 0.02039819]
 [0.11746988 0.          0.08965517 0.12096774 0.01966532]
 [0.12048193 0.          0.08275862 0.12903226 0.02003176]
 ...
 [0.07831325 0.          0.02758621 0.10483871 0.01380237]
 [0.0753012  0.          0.06896552 0.0766129  0.01429095]
 [0.09939759 0.          0.04137931 0.125       0.01844387]]
[[0.52350699]
 [0.63278272]
 [0.52731893]]
```

Train과 test를 8:2로 분할

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(scaled_X, scaled_y,
                                                    test_size=0.2, shuffle=False)
```

```
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
```

```
(6451, 5) (6451, 1)
(1613, 5) (1613, 1)
```

```
num_features = len(X_train[1])
print('number of features :', num_features)  number of features : 5
```

MLP 회귀모델

```
def dnn_reg1():
    model = Sequential([
        Dense(n1, activation='relu', input_shape=[num_features]),
        Dense(1)
    ])
    model.compile(optimizer='adam', loss='mse', metrics=['mae'])
    return model
```

n1 = 32
n2 = 32
n3 = 32
n4 = 32

```
def dnn_reg2():
    model = Sequential([
        Dense(n1, activation='relu',
              input_shape=[num_features]),
        Dense(n2, activation='relu'),
        Dense(1)
    ])
```

```
def dnn_reg4():
    model = Sequential([
        Dense(n1, activation='relu',
              input_shape=[num_features]),
        Dense(n2, activation='relu'),
        Dense(n3, activation='relu'),
        Dense(n4, activation='relu'),
        Dense(1)
    ])
```

회귀 모델 아키텍처 및 파라미터 구하기

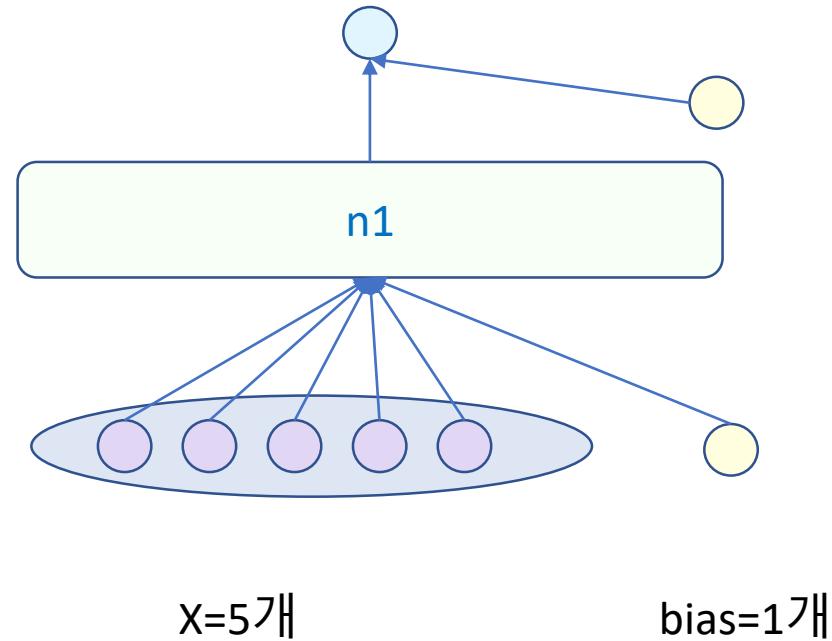
```
def dnn_reg1():
    model = Sequential([
        Dense(n1, activation='relu', input_shape=[num_features]),
        Dense(1)
    ])
```

```
model = dnn_reg1()
model.summary()
```

Model: "sequential"

| Layer (type) | Output Shape | Param # |
|-----------------|--------------|---------|
| dense (Dense) | (None, 32) | 192 |
| dense_1 (Dense) | (None, 1) | 33 |

파라미터 개수 계산하기
 $Dense_1: (32+1)*1=33$
 $Dense: (5+1)*32=192$



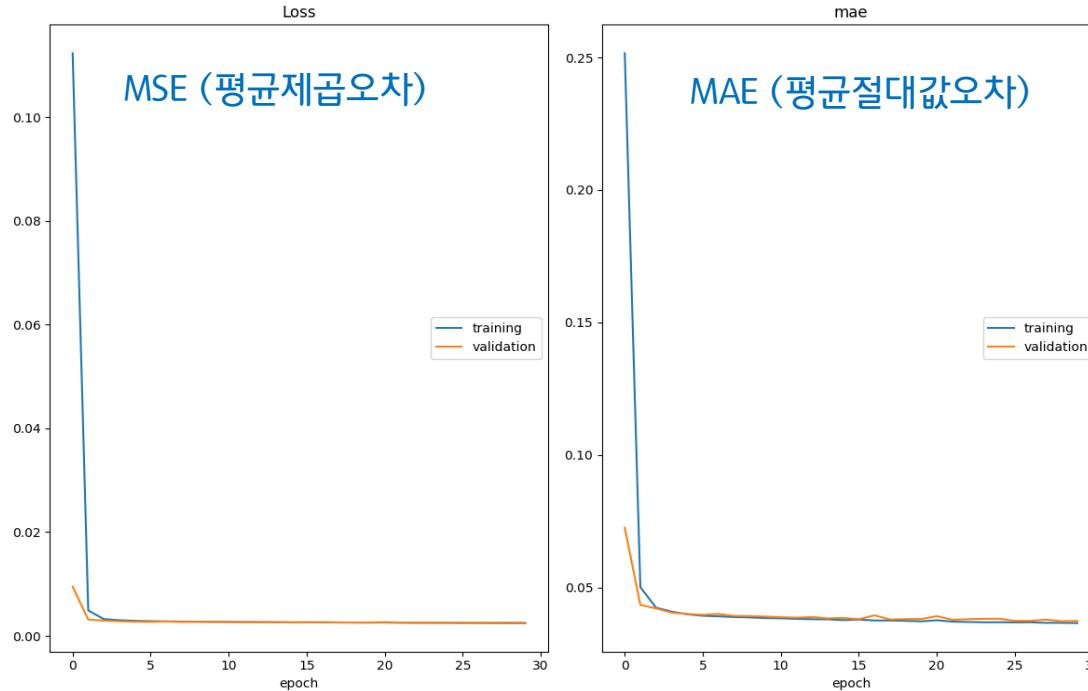
모델 훈련 및 실시간 가시화

Colab에서 꼭 미리 실행이 필요합니다.

!pip install livelossplot

```
from livelossplot import PlotLossesKeras
```

```
history = model.fit(X_train,y_train, epochs=30, validation_split=0.2, batch_size=32, callbacks=[PlotLossesKeras()])
```



```
# 테스트 데이터 세트로 모델 평가: Loss 값 확인  
mse, mae = model.evaluate(X_test, y_test, verbose=0)
```

Neuron1= 32, MSE= 0.0025, MAE= 0.0362

훈련과정 Loss와 MAE: history

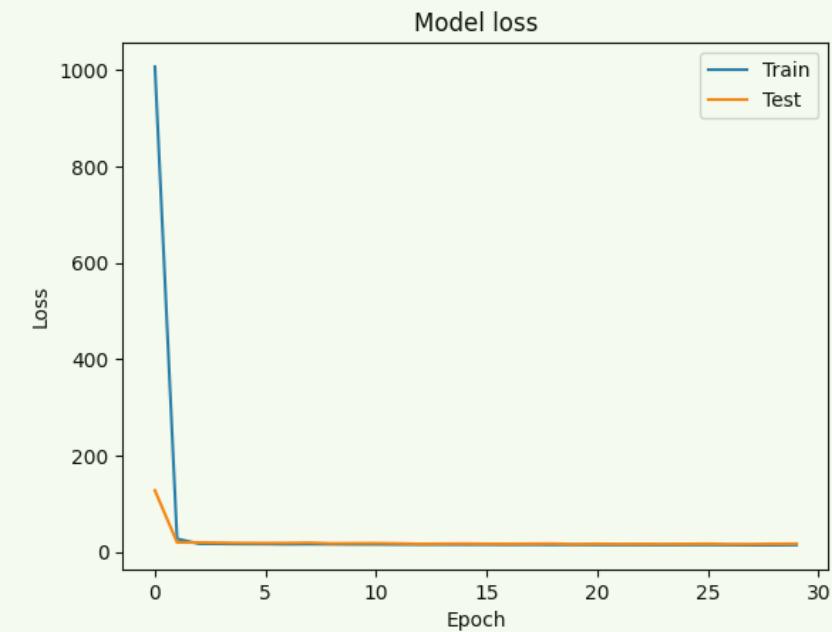
```
history.history.keys()
```

```
dict_keys(['loss', 'mae', 'val_loss', 'val_mae'])
```

```
hist = pd.DataFrame(history.history)
hist['epoch'] = history.epoch
hist.tail()
```

| | loss | mae | val_loss | val_mae | epoch |
|----|-----------|----------|-----------|----------|-------|
| 25 | 14.801393 | 2.847822 | 17.192307 | 2.992428 | 25 |
| 26 | 14.521421 | 2.806103 | 16.137362 | 2.960289 | 26 |
| 27 | 14.246248 | 2.754538 | 16.351469 | 2.987771 | 27 |
| 28 | 14.358523 | 2.774629 | 16.986986 | 2.998828 | 28 |
| 29 | 14.627845 | 2.818771 | 17.263212 | 3.201360 | 29 |

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['Train', 'Test'], loc='upper right')
```



이제 DNN 모델로 예측: 속도

표준화(정규화)로 스케일 상태 → 원래 속도로 환원해주자

```
y_pred = model.predict(X_test, verbose=0)  
print(y_pred.shape)
```

(1613, 1)

```
print('(scaled) predictions : \n', y_pred[:2])  
print('\n (scaled) actural values :\n',y_test[:2])
```

(scaled) predictions :
[[0.4665929]
[0.4981718]]

(scaled) actural values :
[[0.50190597]
[0.54002541]]

```
y_inv_pred = scaler.inverse_transform(y_pred)  
y_inv_test = scaler.inverse_transform(y_test)
```

```
print('predictions : \n', y_inv_pred[:2])  
print('\nactural values :\n',y_inv_test[:2])
```

predictions :
[[45.82086]
[48.30612]]

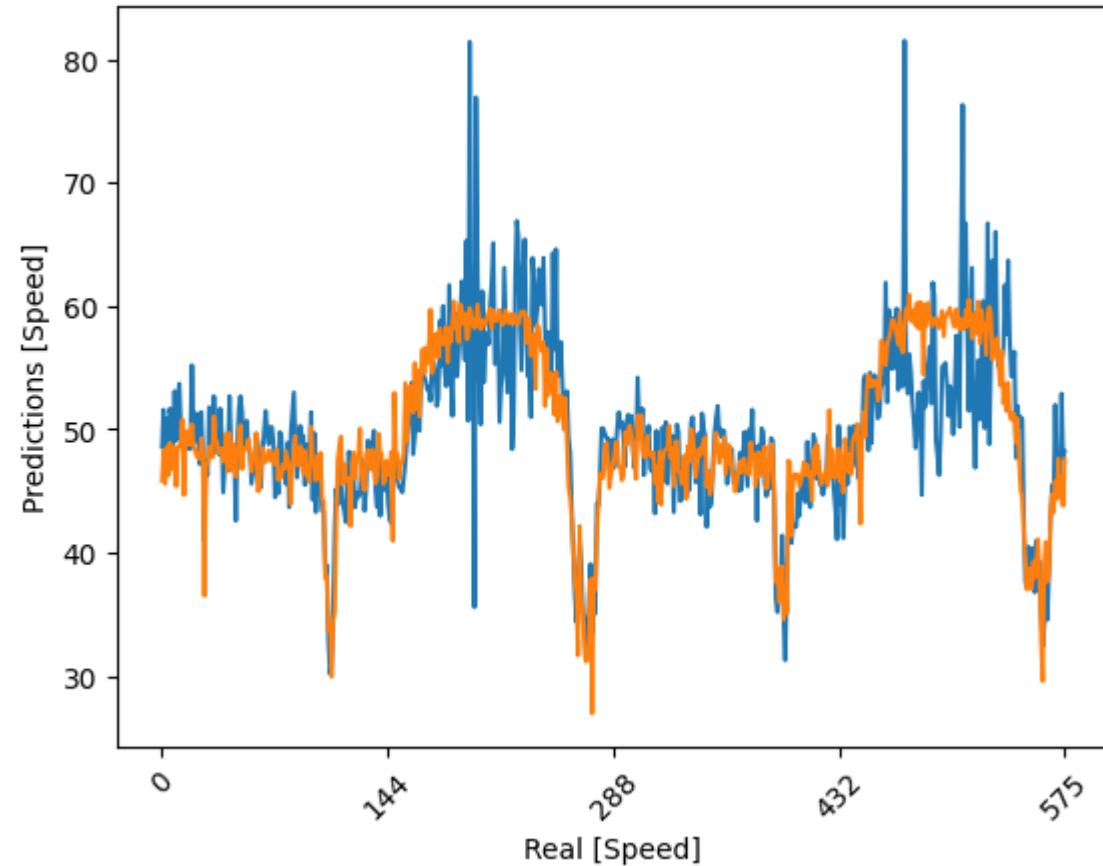
actural values :
[[48.6]
[51.6]]

DNN으로 2일간 속도 예측

```
# 가시화를 위해서 2차원을 1차원으로
test_prediction = y_inv_pred.flatten()
test_real = y_inv_test.flatten()
```

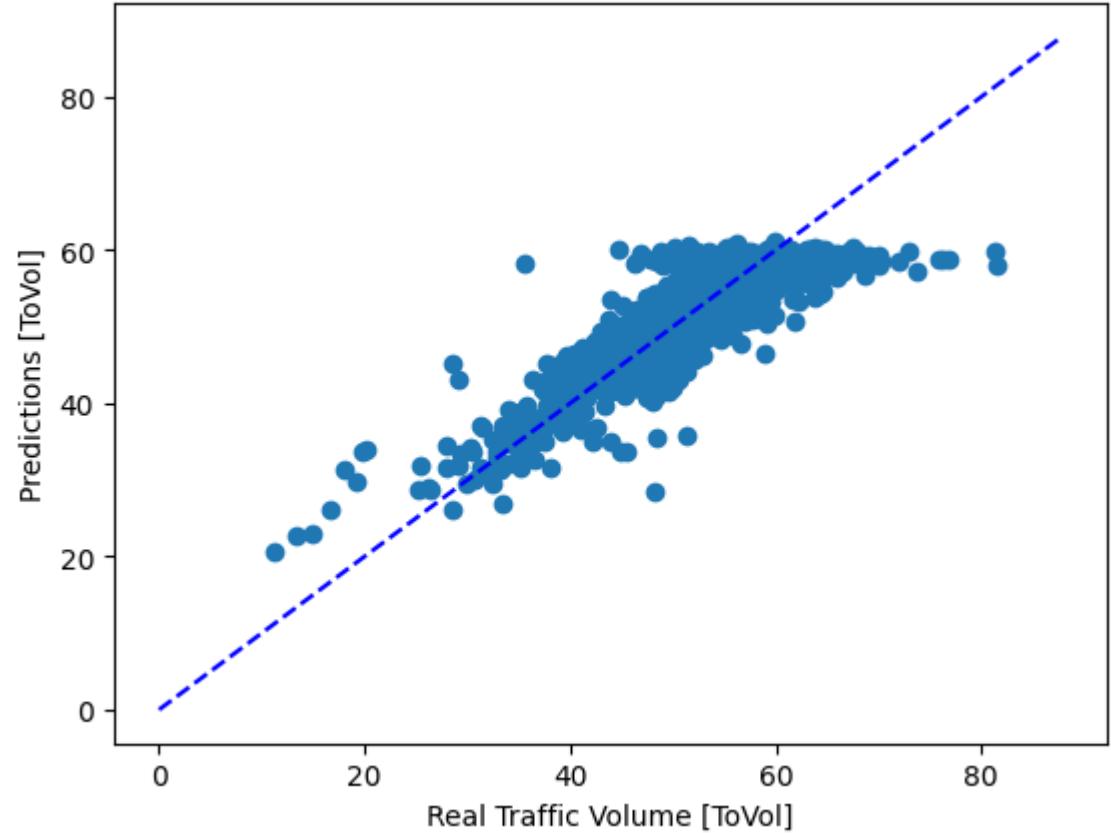
```
plt.plot(test_real[:576])
plt.plot(test_prediction[:576])
plt.xticks(rotation=45)
plt.xticks([0, 144, 288, 432, 575])
plt.xlabel('Real [Speed]')
plt.ylabel('Predictions [Speed]')
```

대전시 유성구 대학로 (vds16)



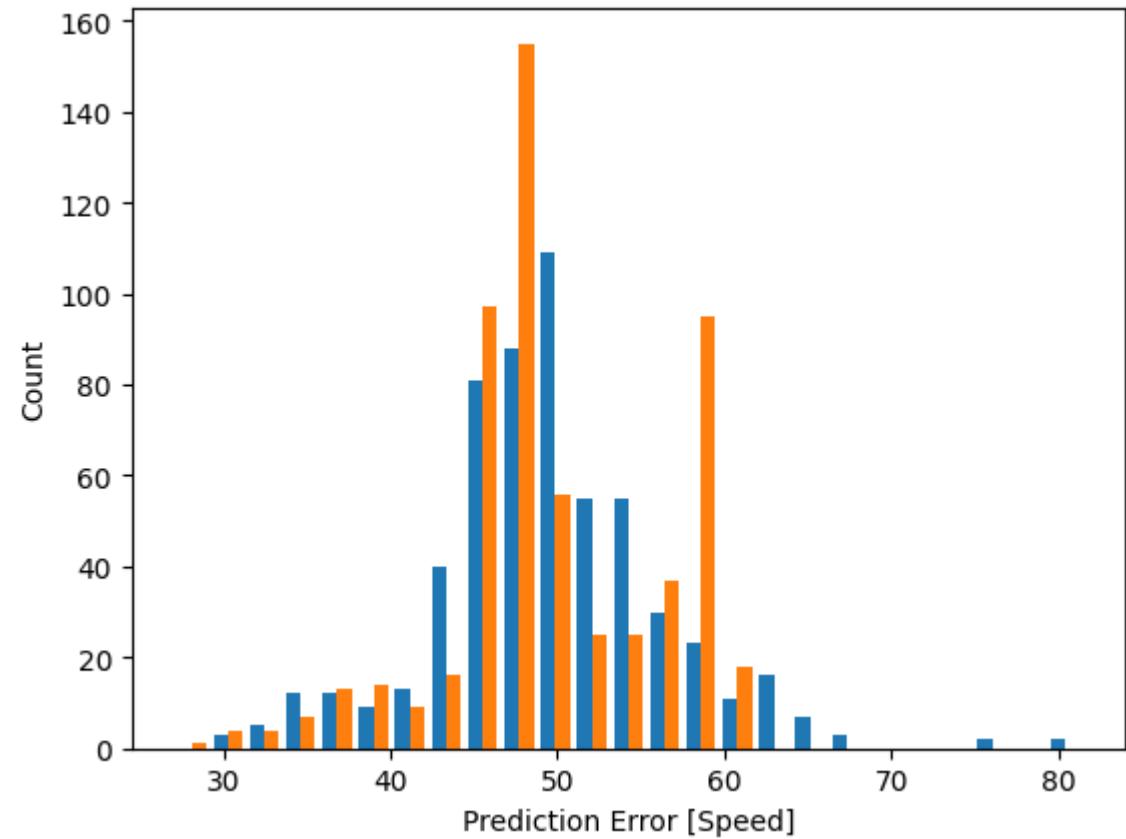
DNN 회귀 예측 오차

```
plt.scatter(test_real, test_prediction)
plt.xlabel('Real Traffic Volume [ToVol]')
plt.ylabel('Predictions [ToVol]')
plt.plot([0, max_speed], [0, max_speed], 'b--')
```



회귀 예측 오차

```
error = test_real[:576], test_prediction[:576]
plt.hist(error, bins = 25)
plt.xlabel("Prediction Error [Speed]")
plt.ylabel("Count")
```

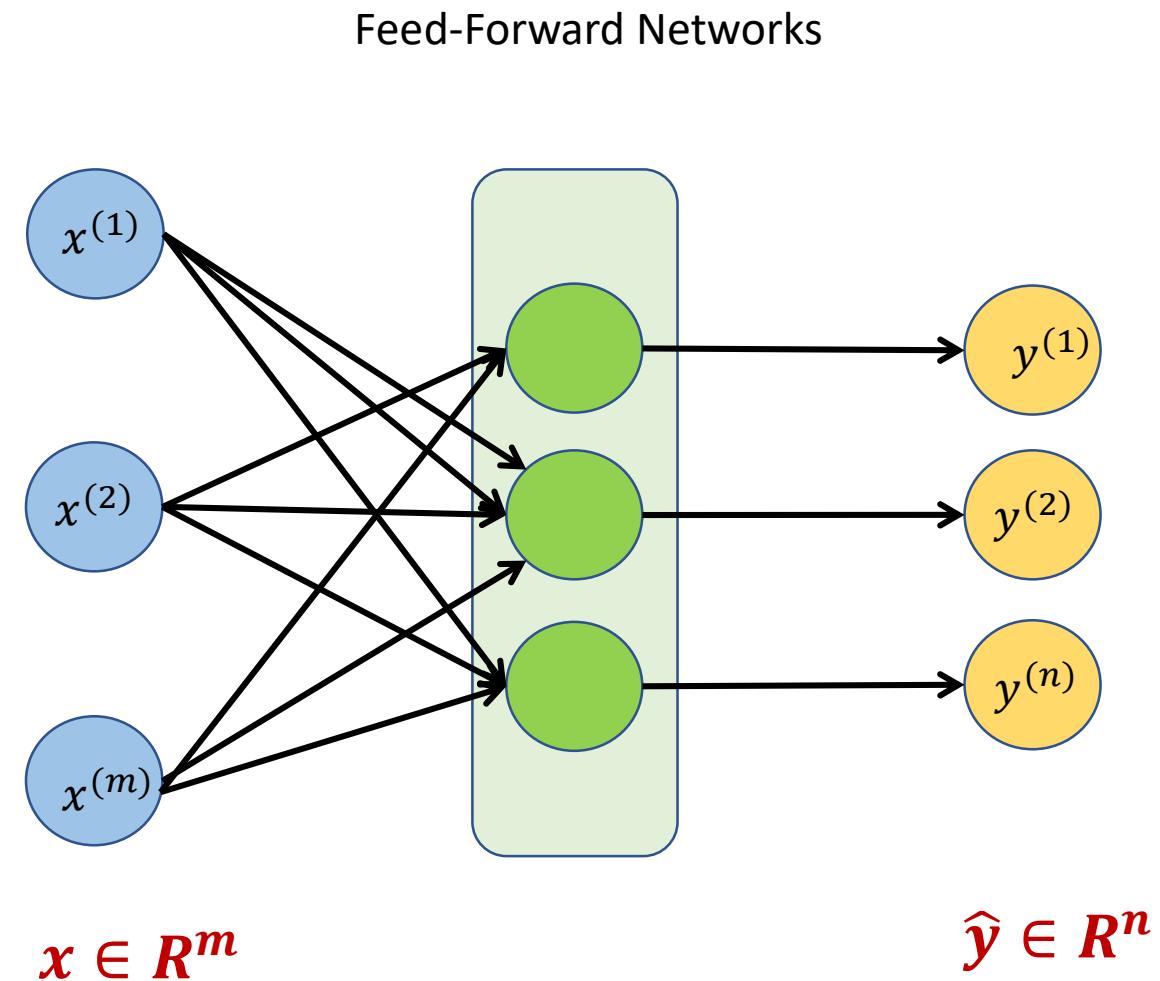
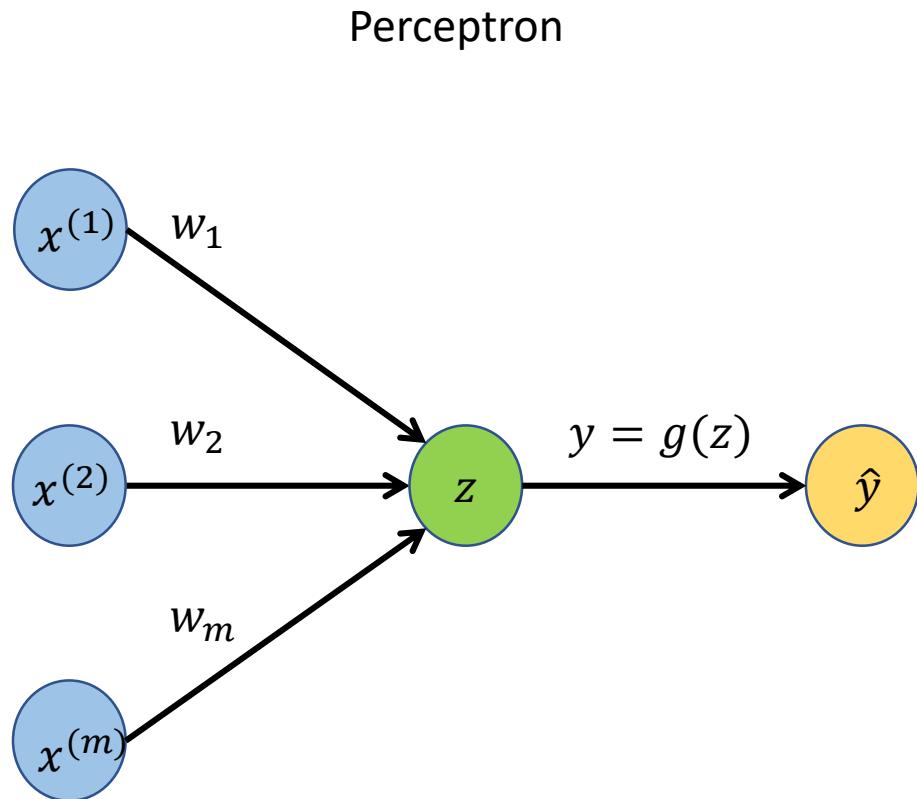


2-5

시계열데이터의 순서가 있는 RNN 모델 이해

- ❖ 순서(Order)가 있는 데이터
 - ✓ 시계열데이터
 - ✓ 텍스트 (NLP, 언어모델)
- ❖ 시계열데이터의 순서를 이해하자
 - ✓ 기억, 메모리, 상태(state)
 - ✓ 기억의 길이(?)
 - ✓ 상태(cell state)를 이해하자
 - LSTM에서는 state와 cell state 2개를 사용한다.
 - GRU에서 state 1개
 - SimpleRNN에서는 state 1개
- ❖ 순서로 생기는 변수
 - ✓ 과거 기억의 길이를 표현해보자
 - Look_back, windows
 - ✓ 미래 예측의 길이를 표현해보자
 - Look_forward, horizons
- ❖ 순서가 있는 데이터의 특성을 이해해보자
 - ✓ Text 데이터는 좋은 예이다.

The Perceptron Revisited

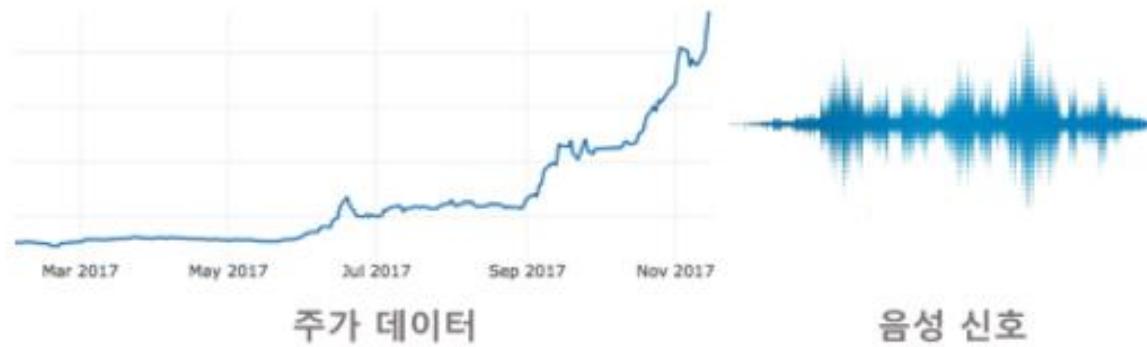


What is a time series problem? Order

- ❖ Time series deals with data over time: Predicting
 - ✓ weather (short term forecasts), climate (long term forecasts)

This sentence is a sequence of words...

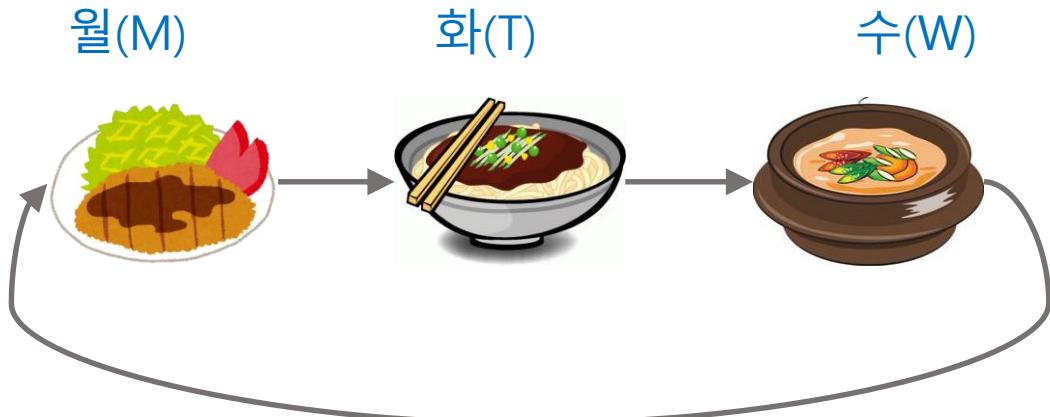
↑
 $t = 1$ ↑
 $t = 2$ ↑
 $t = 3$...



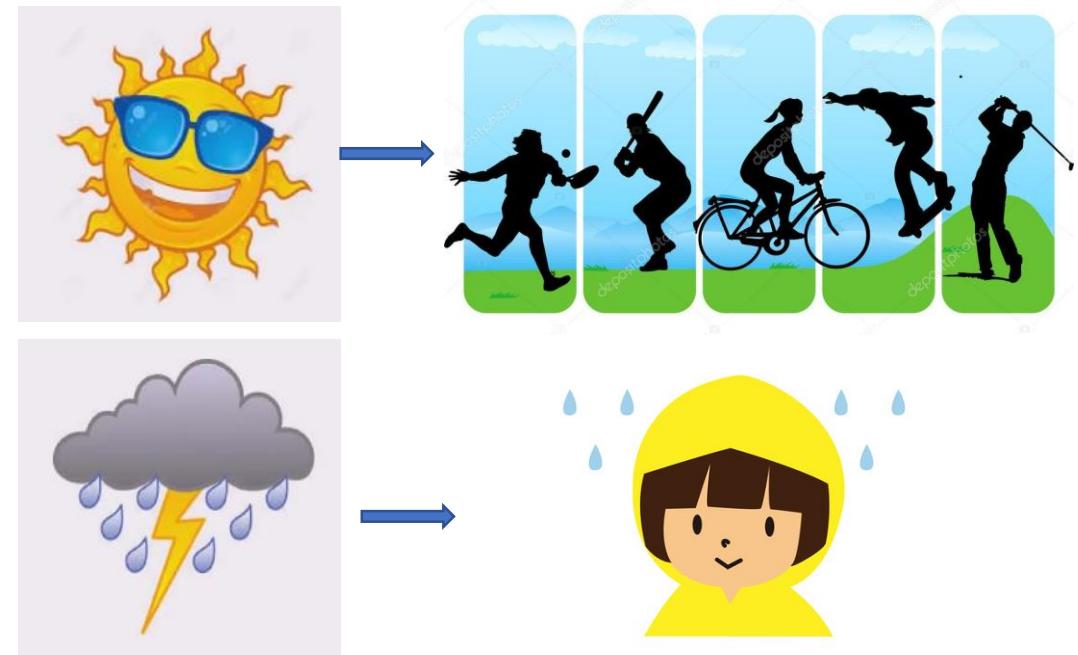
Order is important in the sequence data



3-days
cooking pattern

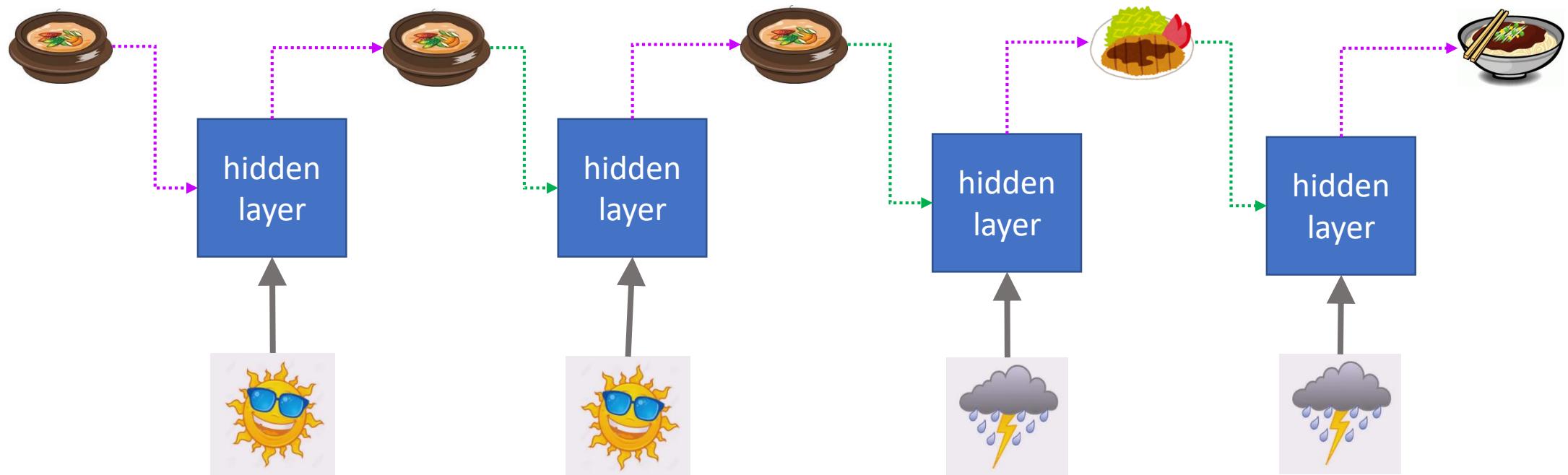


It's sunny, the same food as yesterday
It's rainy, following the pattern



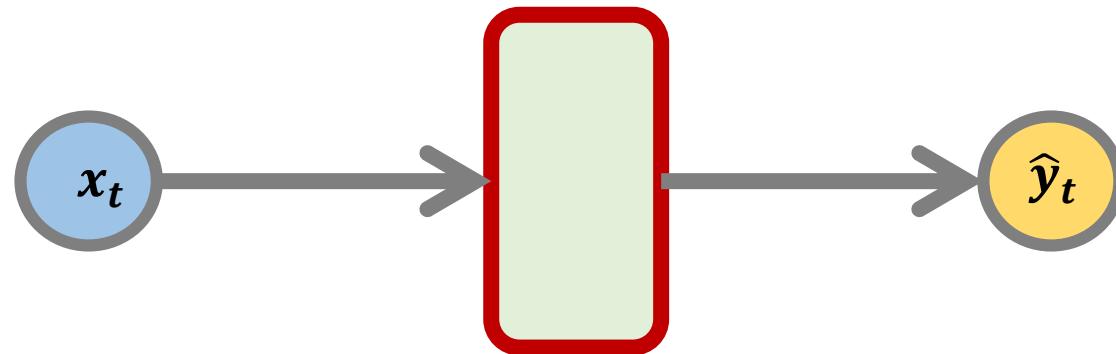
Input, Memory, and Recurrent

SimpleRNN(1, input_shape=(4,1))



Feed-Forward Networks Revisited

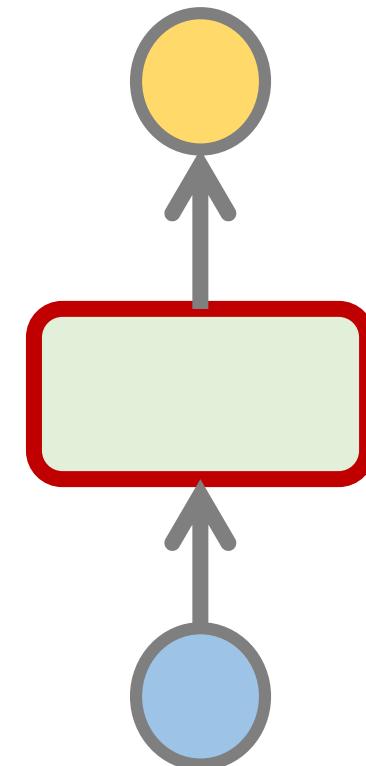
뉴런 개수를 1개로 만들고, 가중치(weights)를 1개로 표시



$$x_t \in R^m$$

$$\hat{y} \in R^n$$

90도 회전

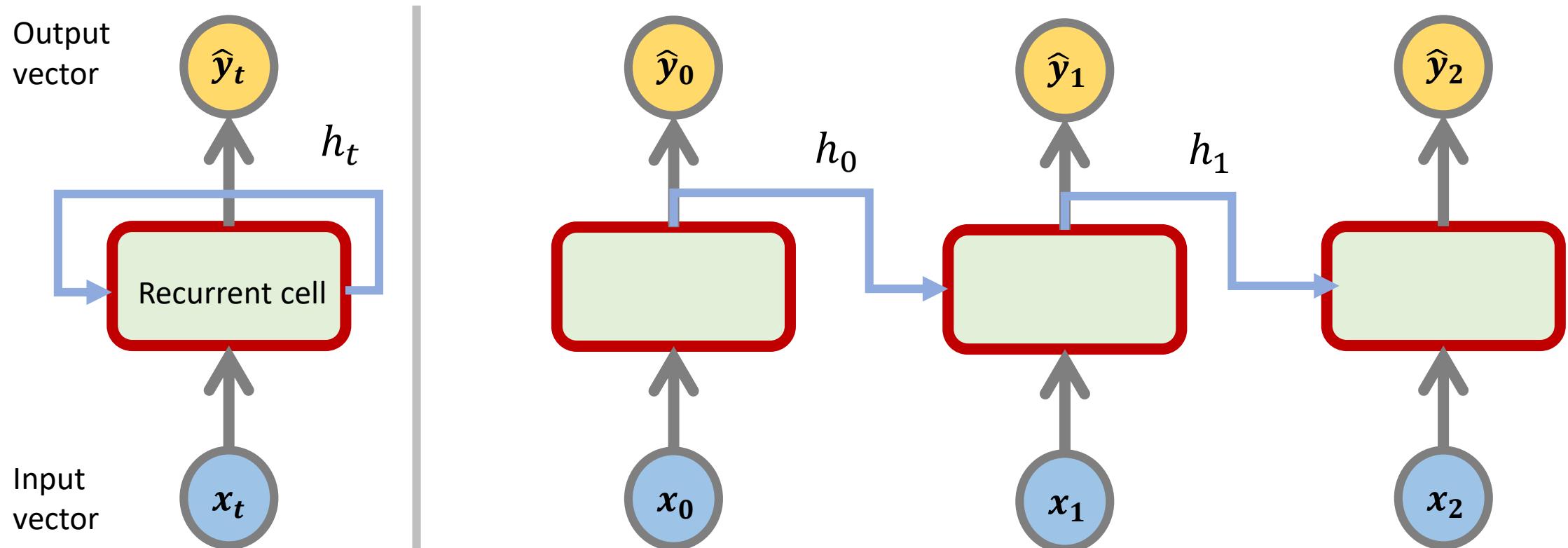


Neurons with Recurrence

$$\hat{y}_t = f(x_t, h_{t-1})$$

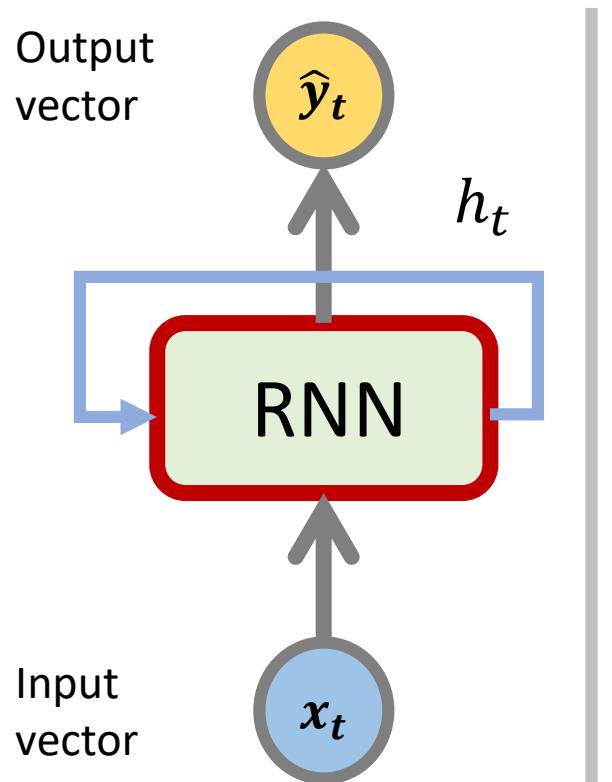
output input past memory

기억: 과거 상태 정보



Recurrent Neural Networks (RNNs)

RNNs have a **state**, h_t , that is updated at each time step as a sequence is processed



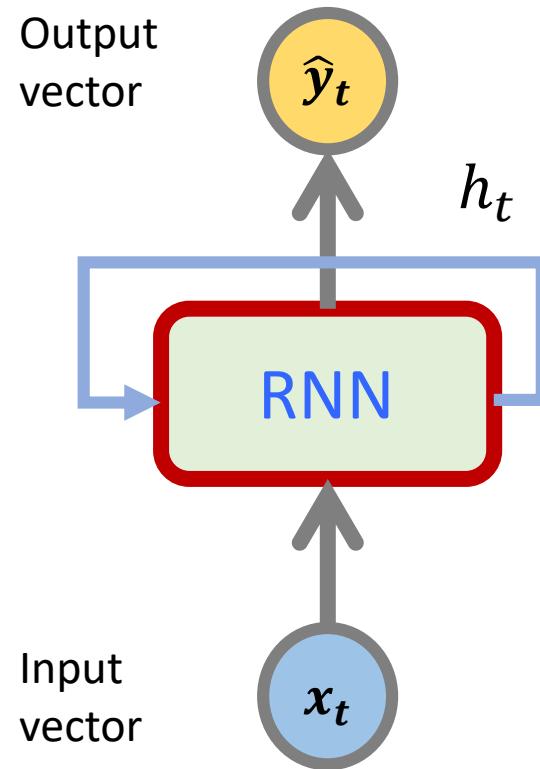
Apply a recurrence relation at every time step to process a sequence:

$$h_t = f_W(x_t, h_{t-1})$$

cell state input past cell state

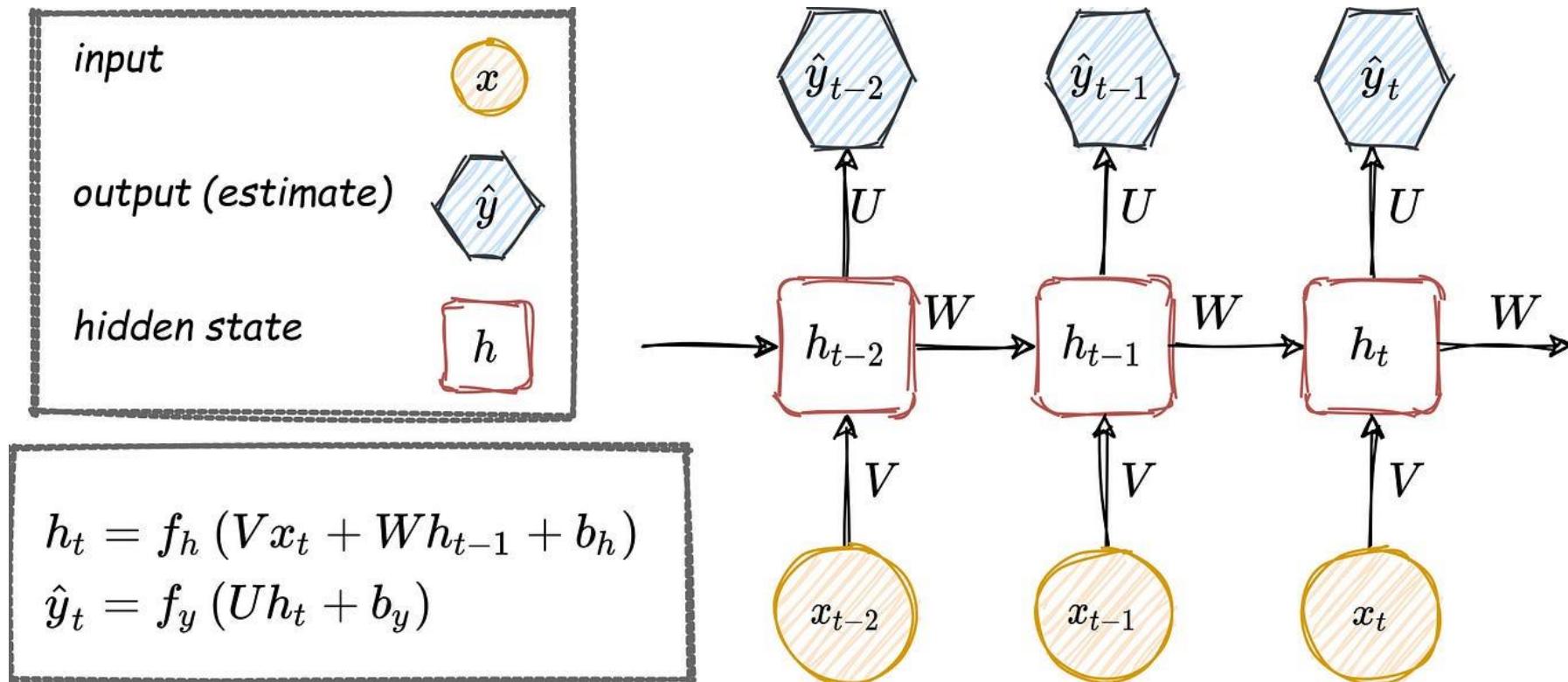
Activation function with weights W

RNN Intuition



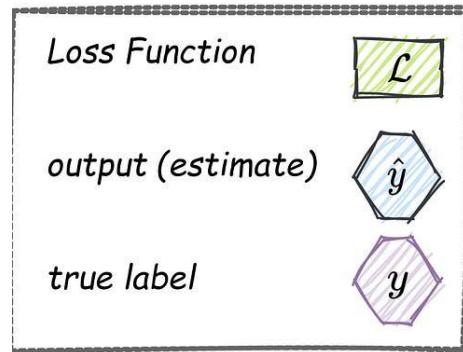
```
model = myRNN()  
hidden_state = [0,0,0,0]  
  
sentence = ["I", "love", "recurrent", "neural"]  
  
for word in sentence:  
    prediction, hidden_state = model(word, hidden_state)  
  
    next_word_prediction = prediction  
    # >>> "networks"
```

Multy Layers Perceptrons



(source) <https://towardsdatascience.com/the-exploding-and-vanishing-gradients-problem-in-time-series-6b87d558d22>

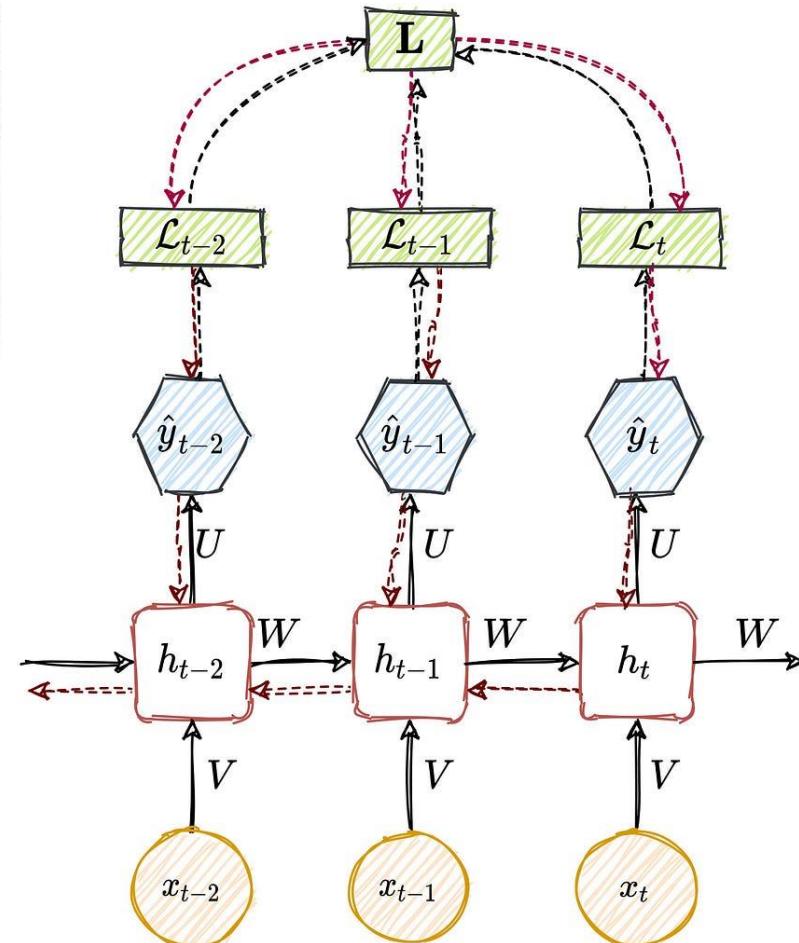
Backpropagation Through Time (BPTT)



$$\mathbf{L} = \sum_i \mathcal{L}_i (\hat{y}_t, y_t)$$

Forward Pass:
 $h_t, \hat{y}_t, \mathcal{L}_t, \mathbf{L}$

Backward Pass:
 $\frac{\partial \mathbf{L}}{\partial U}, \frac{\partial \mathbf{L}}{\partial V}, \frac{\partial \mathbf{L}}{\partial W}, \frac{\partial \mathbf{L}}{\partial b_h}, \frac{\partial \mathbf{L}}{\partial b_y}$



$$\frac{\partial \mathbf{L}}{\partial W} = \sum_{i=0}^T \frac{\partial \mathcal{L}_i}{\partial W} \propto \sum_{i=0}^T \left(\prod_{j=k+1}^y \frac{\partial h_j}{\partial h_{j-1}} \right) \frac{\partial h_k}{\partial W}$$

$$\frac{\partial \mathbf{L}}{\partial W} \propto \sum_{i=0}^T \left(\prod_{j=k+1}^y \frac{\partial h_j}{\partial h_{j-1}} \right) \frac{\partial h_k}{\partial W}$$

(source) <https://towardsdatascience.com/the-exploding-and-vanishing-gradients-problem-in-time-series-6b87d558d22>

The Vanishing and Exploding Gradients Problem

1. *Vanishing gradient*

$$\left\| \frac{\partial h_i}{\partial h_{i-1}} \right\|_2 < 1$$

2. *Exploding gradient*

$$\left\| \frac{\partial h_i}{\partial h_{i-1}} \right\|_2 > 1$$

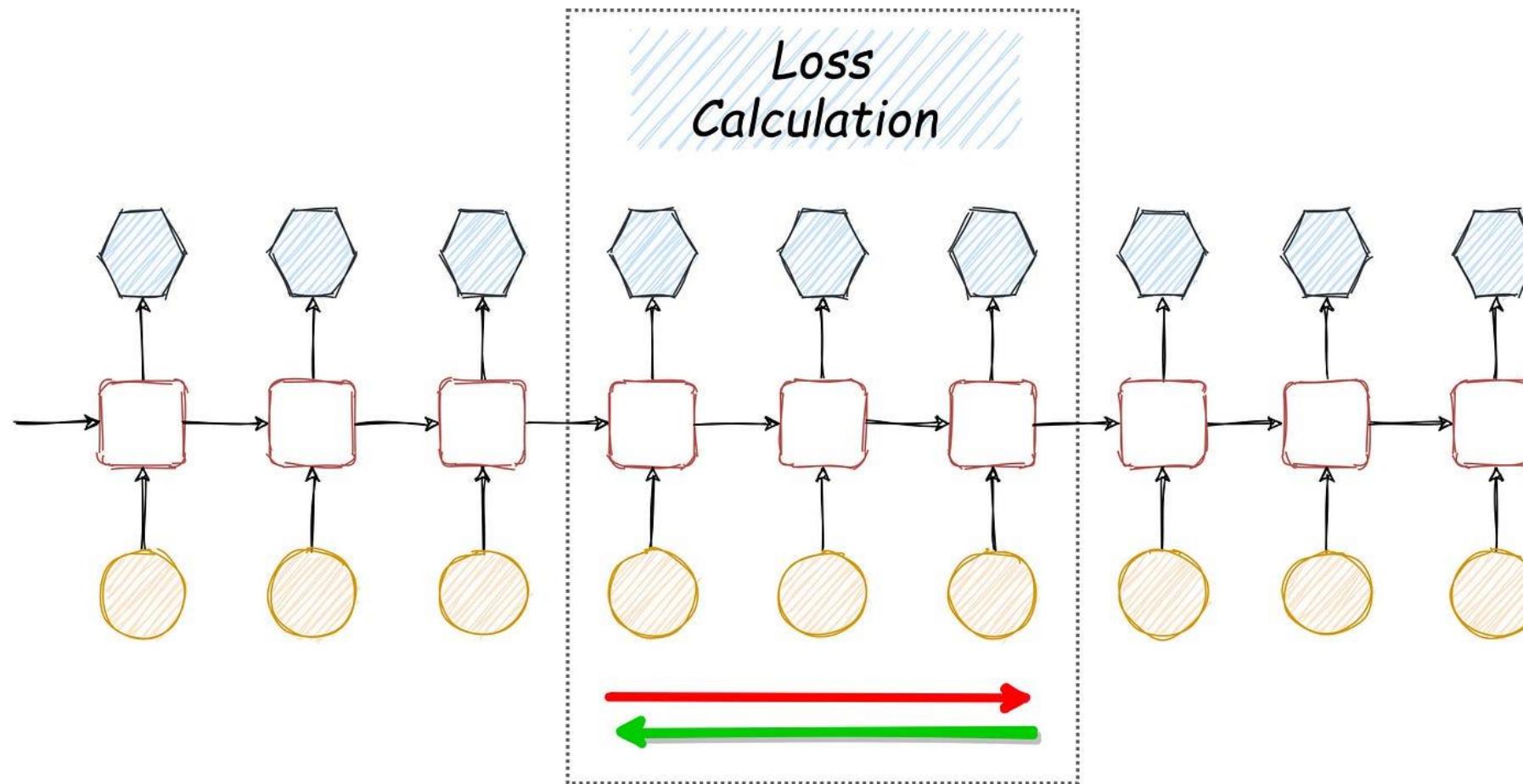


Solutions

- ① Truncated backpropagation through time (TBPTT)
- ② Long short-term memory (LSTM)
- ③ Gradient Clipping

(source) <https://towardsdatascience.com/the-exploding-and-vanishing-gradients-problem-in-time-series-6b87d558d22>

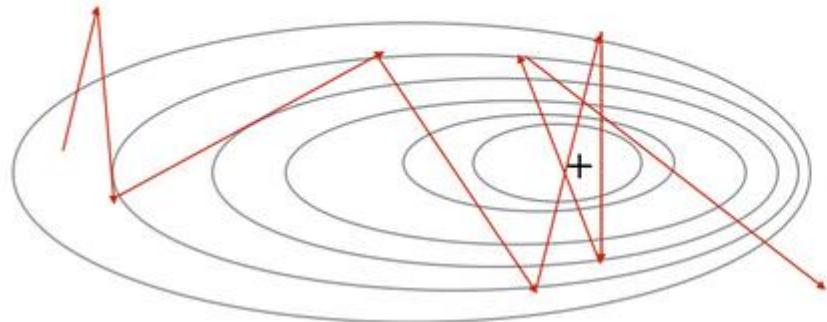
Truncated Backpropagation Through Time



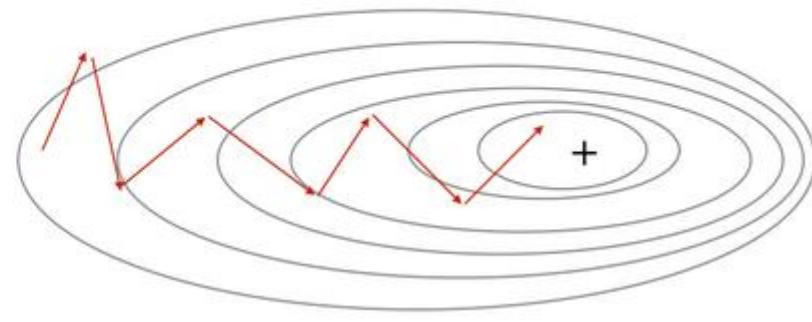
(source) <https://towardsdatascience.com/the-exploding-and-vanishing-gradients-problem-in-time-series-6b87d558d22>

Gradient Clipping

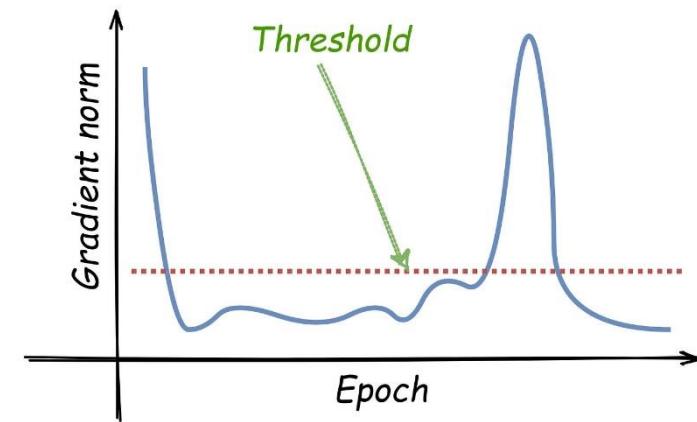
Without gradient clipping



With gradient clipping



$$clip(g, threshold) = g \cdot \max\left(1, \frac{threshold}{\|g\|}\right)$$



(source) <https://towardsdatascience.com/the-exploding-and-vanishing-gradients-problem-in-time-series-6b87d558d22>

2-6

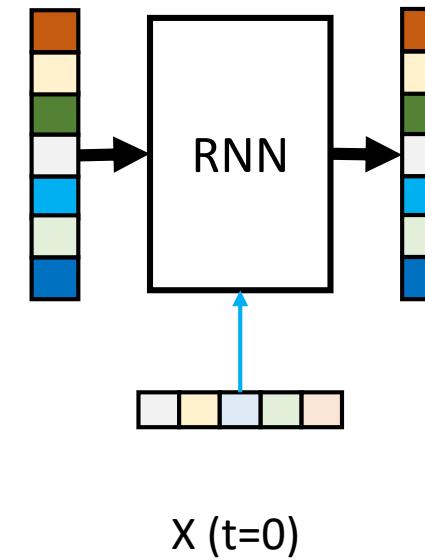
시계열 데이터를 이용한 RNN 모델 전략

❖ 시계열 데이터의 특성을 먼저 파악하자.

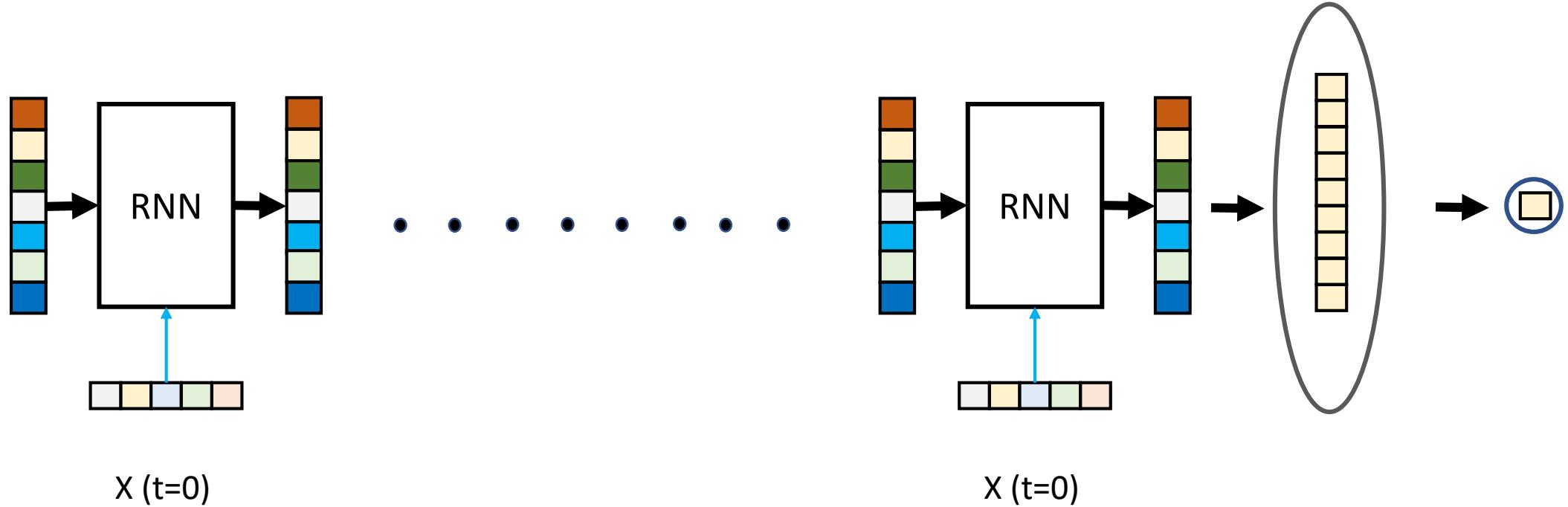
- ✓ 순서가 있는 데이터이다.
- ✓ 기억의 길이는 순서를 유지해야 하는 파라이터이다.
 - Windows, look_back로 표시

❖ RNN 모델의 특징을 파악하자

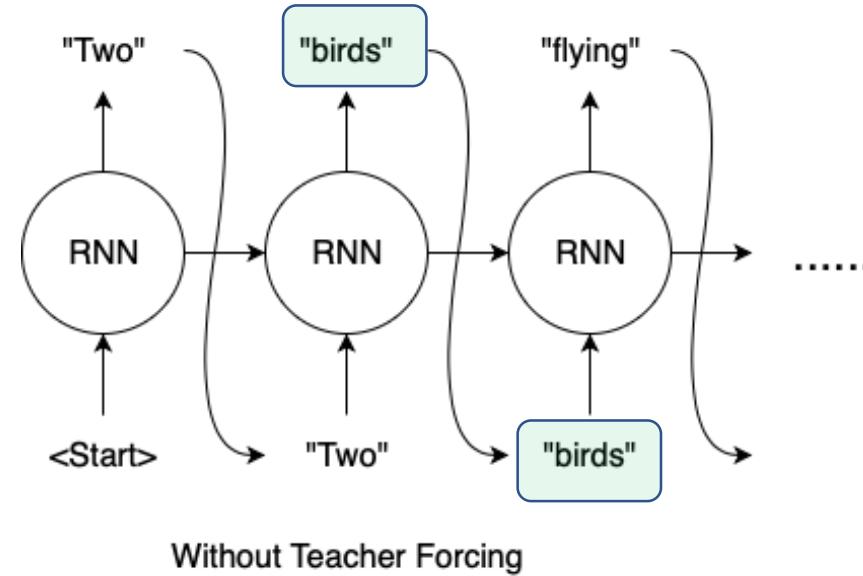
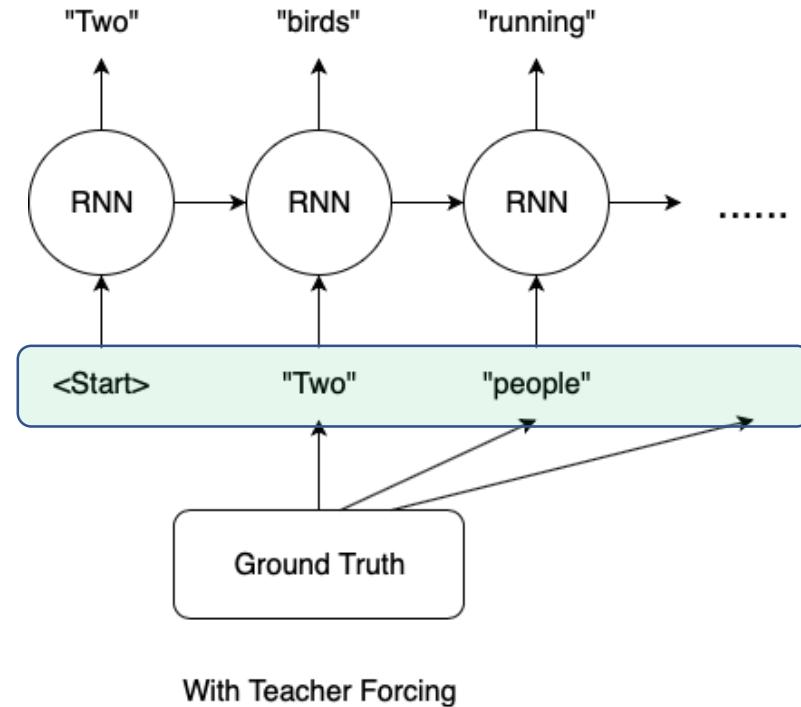
- ✓ 응용 모델이 Many-to-One인지 Many-to-Many인지에 따라서
 - look_back 혹은 windows의 개수
 - look_forward 혹은 horizon의 개수를 정한다.
- ✓ 시간에 따른 순서의 개수는 RNN에서는 무시한다.
 - 왜냐하면 가중치를 공유하기 때문에
 - 1개의 시간에 가중치만 고려한다.
- ✓ RNN 상태(state), 혹은 기억(memory)의 뉴런 개수를 정한다.
 - 뉴런 개수는 파라미터이다.
 - 보통 RNN 출력의 뉴런과 같다. 왜냐하면 기억의 업데이트의 결과이다.
- ✓ 교사 학습 (Teacher Forcing Learning) 에 대하여 이해한다.



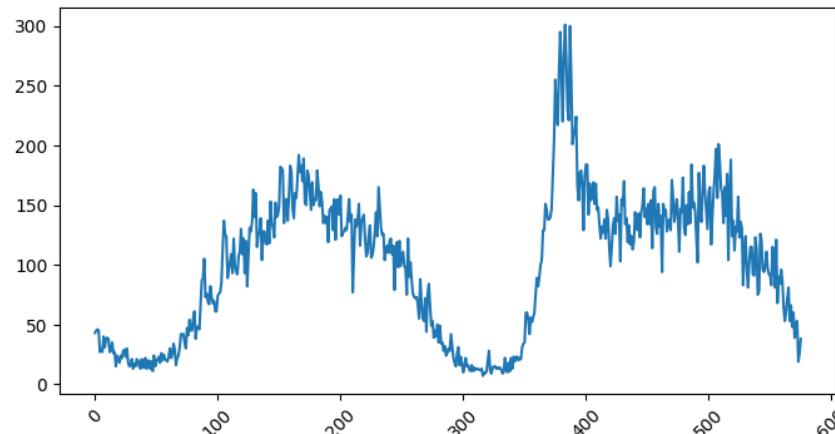
시계열 데이터를 이용한 RNN 모델링 전략



With Teacher Forcing Learing



총 교통량(ToVol)을 예측해보자



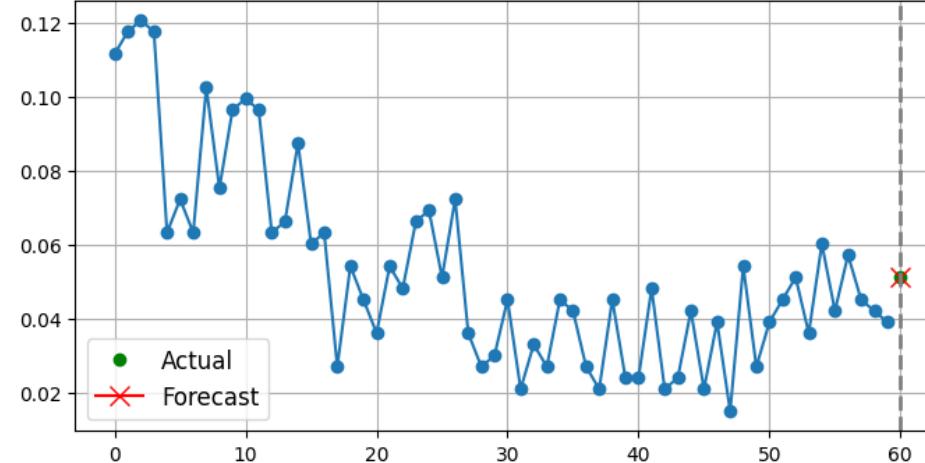
```
look_back = 12*5
look_forward = 1

X, y = create_dataset(df_scaled, look_back, look_forward)

X.shape, y.shape
((8003, 60, 1), (8003, 1, 1))

plot_windows(X,y,y)
```

```
# convert an array of values into a dataset matrix
def create_dataset(dataset, look_back, look_forward):
    dataX, dataY = [], []
    np.array(dataY)
    for i in range(len(dataset)-look_back-1):
        a = dataset[i:(i+look_back)]
        if len(dataset[i + look_back:i+look_back+look_forward]) == look_forward:
            dataX.append(a)
            dataY.append(dataset[i + look_back:i+look_back+look_forward])
    return np.array(dataX), np.array(dataY)
```



RNN 모델로 교통흐름 예측하기

- 목표는 $X_1 + X_2 + X_3 = X_4$ 임을 예측한다.

SmVol, MeVol, LaVol을 특성으로하여 ToVol 계
산

RNN으로 VDS 데이터의 교통량 예측

```
▶ import tensorflow as tf
    import pandas as pd
    import numpy as np
    import matplotlib.pyplot as plt

[2] !pip install livelossplot
    from livelossplot import PlotLossesKeras
```

RNN으로 VDS 데이터의 교통량 예측

```
▶ df = pd.read_csv("https://raw.githubusercontent.com/hongsukyi/Lectures/main/data/vds16.csv")
df.head(5)
```

→

| | Date | ToVol | SmVol | MeVol | LaVol | Speed | OccRate |
|---|-----------------|-------|-------|-------|-------|-------|---------|
| 0 | 2017-04-02 0:00 | 43 | 34 | 9 | 0 | 50.3 | 1.90 |
| 1 | 2017-04-02 0:05 | 45 | 32 | 13 | 0 | 58.9 | 1.84 |
| 2 | 2017-04-02 0:10 | 46 | 34 | 12 | 0 | 50.6 | 1.87 |
| 3 | 2017-04-02 0:15 | 45 | 36 | 9 | 0 | 50.9 | 1.72 |
| 4 | 2017-04-02 0:20 | 27 | 13 | 13 | 1 | 62.2 | 1.12 |

grid icon

line icon

```
[4] df.set_index('Date', inplace=True)
```

```
[5] df.head()
```

→

| | ToVol | SmVol | MeVol | LaVol | Speed | OccRate |
|-----------------|-------|-------|-------|-------|-------|---------|
| Date | | | | | | |
| 2017-04-02 0:00 | 43 | 34 | 9 | 0 | 50.3 | 1.90 |
| 2017-04-02 0:05 | 45 | 32 | 13 | 0 | 58.9 | 1.84 |

grid icon

line icon

RNN으로 VDS 데이터의 교통량 예측

```
▶ df_vol = df.iloc[:,0:4].values #Volume  
print(df_vol)  
max_vol = df_vol.max()  
print(max_vol)
```

```
[[43 34 9 0]  
 [45 32 13 0]  
 [46 34 12 0]  
 ...  
 [32 28 4 0]  
 [31 21 10 0]  
 [39 33 6 0]]  
338
```

```
[7] from sklearn.preprocessing import StandardScaler, MinMaxScaler  
  
mm_scaler = MinMaxScaler()  
scaled_df = mm_scaler.fit_transform(df_vol)  
  
scaled_df
```

```
▶ look_back = 24      # 2시간 = 12*2
  states_neurons = 20    # RNN 기억의 뉴런갯수
```

```
▶ X = []
  y = []
  for i in range(len(scaled_df)-look_back-1):
      X.append(scaled_df[i:(i+look_back),1:4])
      y.append(scaled_df[(i+look_back),0:1])

  X = np.array(X)
  y = np.array(y)
```

```
[10] print(X.shape, y.shape)
```

```
→ (8039, 24, 3) (8039, 1)
```

RNN으로 VDS 데이터의 교통량 예측

```
[11] from sklearn.model_selection import train_test_split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_s
        print(X_train.shape, y_train.shape)
        print(X_test.shape, y_test.shape)

→ (6431, 24, 3) (6431, 1)
(1608, 24, 3) (1608, 1)
```

```
[12] from tensorflow.keras.layers import Dense, SimpleRNN, LSTM
    from tensorflow.keras.models import Sequential
```

RNN으로 VDS 데이터의 교통량 예측

```
[13] def model_rnn1():
    model = Sequential([
        SimpleRNN(states_neurons, input_shape=(X_train.shape[1],X_train.shape[2]),
                  return_sequences=False),
        Dense(states_neurons, activation='linear'),
        Dense(1,activation='linear')
    ])

    model.compile(optimizer='adam', loss= 'mse', metrics=['mae'])
    return model
```

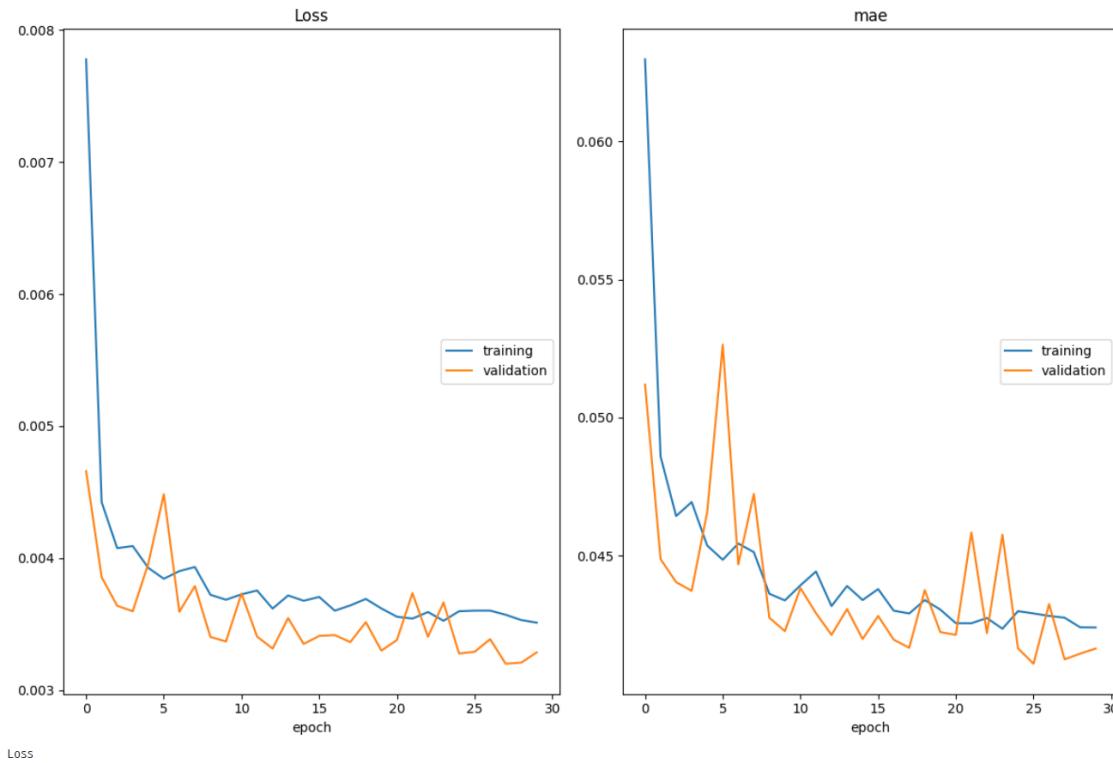
```
[15] model = model_rnn1()
      model.summary()
```

Model: "sequential"

| Layer (type) | Output Shape | Param # |
|------------------------|--------------|---------|
| simple_rnn (SimpleRNN) | (None, 20) | 480 |
| dense (Dense) | (None, 20) | 420 |

RNN으로 VDS 데이터의 교통량 예측

```
▶ history = model.fit(X_train,y_train, epochs=30,  
                      validation_split=0.2,  
                      batch_size=32,  
                      callbacks=[PlotLossesKeras()])
```



RNN으로 VDS 데이터의 교통량 예측

```
[17] # 테스트 데이터 세트로 모델 평가: Loss 값 확인
    mse, mae = model.evaluate(X_test, y_test, verbose=0)
    print("\nlook_back={:3d}, MSE={:8.4f}, MAE={:8.4f}".format(look_back,mse,mae))
```

```
→ look_back= 24, MSE= 0.0037, MAE= 0.0432
```

```
▶ y_pred = model.predict(X_test, verbose=0)
    print(y_pred.shape,y_test.shape)
```

```
→ (1608, 1) (1608, 1)
```

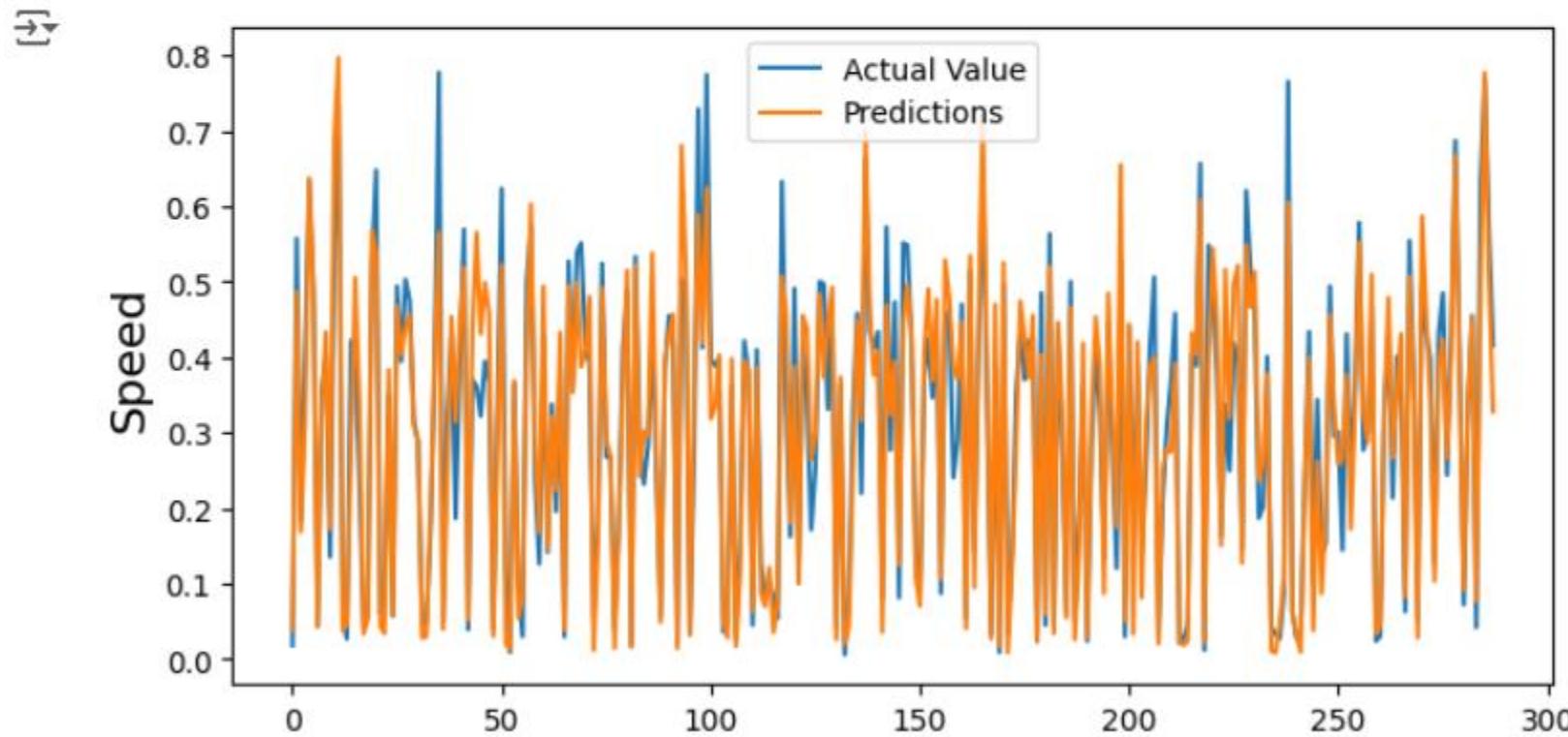
RNN으로 VDS 데이터의 교통량 예측

```
[21] pred_df = pd.DataFrame({'Actual': y_test.flatten(), 'Predicted': y_pred.flatten()})
pred_df.head()
```

| | Actual | Predicted | grid icon |
|---|----------|-----------|-----------|
| 0 | 0.018072 | 0.039698 | info icon |
| 1 | 0.557229 | 0.486437 | |
| 2 | 0.183735 | 0.168845 | |
| 3 | 0.397590 | 0.317322 | |
| 4 | 0.632530 | 0.637335 | |

RNN으로 VDS 데이터의 교통량 예측

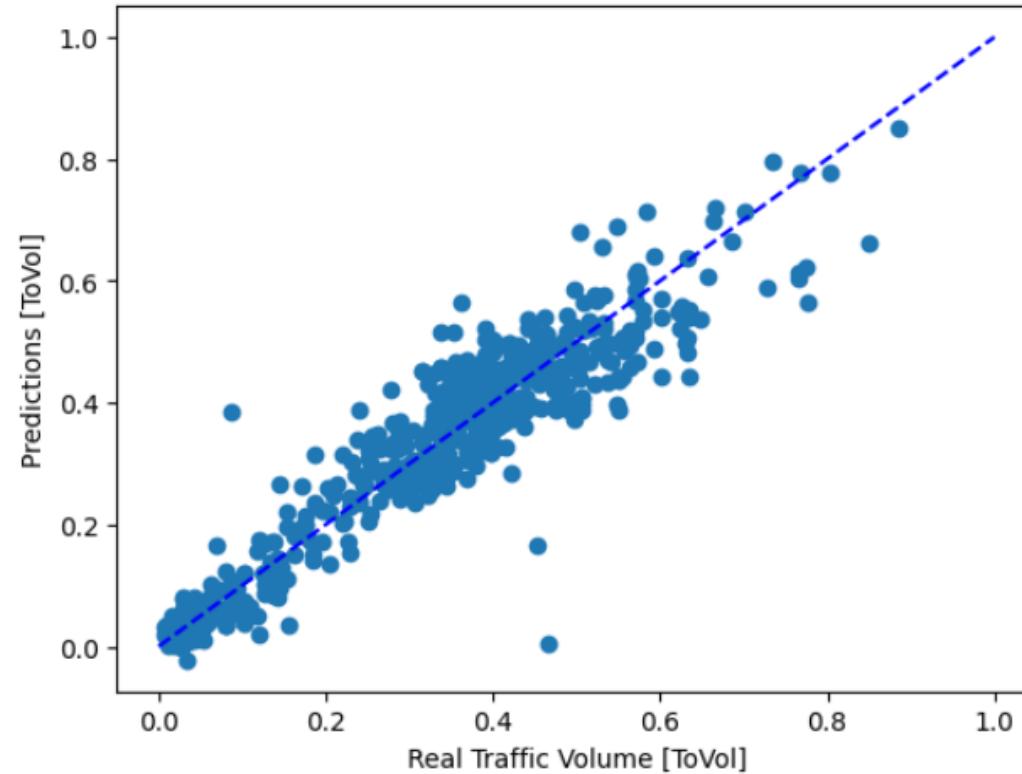
```
▶ plt.figure(figsize=(8,4))
    plt.ylabel('Speed', fontsize=16)
    plt.plot(pred_df[:288])
    plt.legend(['Actual Value', 'Predictions'])
    plt.show()
```



RNN으로 VDS 데이터의 교통량 예측

```
[23] plt.scatter(y_test.flatten()[:576], y_pred.flatten()[:576])
plt.xlabel('Real Traffic Volume [ToVol]')
plt.ylabel('Predictions [ToVol]')
plt.plot([0, max_vol/max_vol], [0, max_vol/max_vol], 'b--')
```

→ [<matplotlib.lines.Line2D at 0x7be457c9ca30>]



3-1

RNN 모델로 교통흐름 예측하기

RNN으로 VDS 데이터 속도 예측

```
[1] import tensorflow as tf  
import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt
```

```
[2] !pip install livelossplot  
from livelossplot import PlotLossesKeras
```

```
[3] df = pd.read_csv("https://raw.githubusercontent.com/hongsukyi/Lectures/main/data/vds16.csv")  
df.head(5)
```

| | Date | ToVol | SmVol | MeVol | LaVol | Speed | OccRate |
|---|-----------------|-------|-------|-------|-------|-------|---------|
| 0 | 2017-04-02 0:00 | 43 | 34 | 9 | 0 | 50.3 | 1.90 |
| 1 | 2017-04-02 0:05 | 45 | 32 | 13 | 0 | 58.9 | 1.84 |
| 2 | 2017-04-02 0:10 | 46 | 34 | 12 | 0 | 50.6 | 1.87 |
| 3 | 2017-04-02 0:15 | 45 | 36 | 9 | 0 | 50.9 | 1.72 |
| 4 | 2017-04-02 0:20 | 27 | 13 | 13 | 1 | 62.2 | 1.12 |

```
[4] df.set_index('Date', inplace=True)
```

```
[5] df.head()
```

| Date | ToVol | SmVol | MeVol | LaVol | Speed | OccRate |
|-----------------|-------|-------|-------|-------|-------|---------|
| 2017-04-02 0:00 | 43 | 34 | 9 | 0 | 50.3 | 1.90 |
| 2017-04-02 0:05 | 45 | 32 | 13 | 0 | 58.9 | 1.84 |
| 2017-04-02 0:10 | 46 | 34 | 12 | 0 | 50.6 | 1.87 |

RNN으로 VDS 데이터 속도 예측

```
✓ [7] df_spd = df.iloc[:,4:5].values #Speed  
print(df_spd)  
max_speed = df_spd.max()  
print(max_speed)
```

```
[[50.3]  
[58.9]  
[50.6]  
...  
[50.6]  
[59.3]  
[52.5]]  
87.8
```

```
✓ [8] from sklearn.preprocessing import StandardScaler, MinMaxScaler
```

```
scaler = StandardScaler()  
scaled_df = scaler.fit_transform(df_spd)
```

```
scaled_df
```

```
array([[0.12277801],  
[1.20844949],  
[0.16065027],  
...  
[0.16065027],  
[1.25894584],  
[0.40050793]])
```

```
✓ [9] look_back = 24      # 2시간 = 12*2  
states_neurons = 20     # RNN 기억의 뉴런갯수
```

RNN으로 VDS 데이터 속도 예측

```
[10] X = []
    y = []
    for i in range(len(scaled_df)-look_back-1):
        X.append(scaled_df[i:(i+look_back)])
        y.append(scaled_df[(i+look_back)])
    X = np.array(X)
    y = np.array(y)

[11] print(X.shape, y.shape)
→ (8039, 24, 1) (8039, 1)

[12] from sklearn.model_selection import train_test_split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

    print(X_train.shape, y_train.shape)
    print(X_test.shape, y_test.shape)
→ (6431, 24, 1) (6431, 1)
(1608, 24, 1) (1608, 1)
```

RNN으로 VDS 데이터 속도 예측

```
[13] from tensorflow.keras.layers import Dense, SimpleRNN, LSTM
from tensorflow.keras.models import Sequential

[14] def model_rnn1():
    model = Sequential([
        SimpleRNN(states_neurons, input_shape=(X_train.shape[1],1),
                  return_sequences=False),
        Dense(states_neurons, activation='linear'),
        Dense(1,activation='linear')
    ])

    model.compile(optimizer='adam', loss= 'mse', metrics=['mae'])
    return model
```

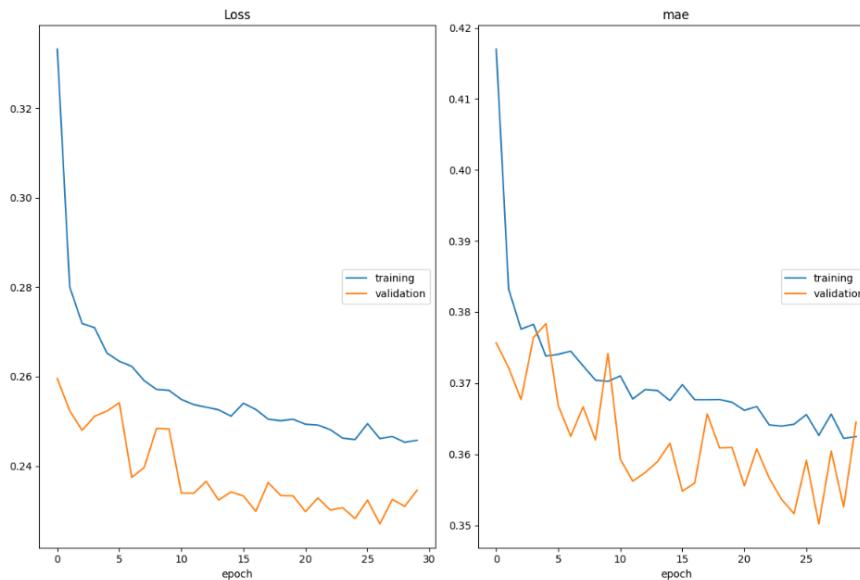
```
[15] model = model_rnn1()
model.summary()
```

↳ Model: "sequential"

| Layer (type) | Output Shape | Param # |
|-------------------------------------|--------------|---------|
| simple_rnn (SimpleRNN) | (None, 20) | 440 |
| dense (Dense) | (None, 20) | 420 |
| dense_1 (Dense) | (None, 1) | 21 |
| <hr/> | | |
| Total params: 881 (3.44 KB) | | |
| Trainable params: 881 (3.44 KB) | | |
| Non-trainable params: 0 (0.00 Byte) | | |
| <hr/> | | |

RNN으로 VDS 데이터 속도 예측

```
[16] history = model.fit(X_train,y_train, epochs=30,  
                      validation_split=0.2,  
                      batch_size=32,  
                      callbacks=[PlotLossesKeras()])
```



```
[17] # 테스트 데이터 세트로 모델 평가: Loss 값 확인  
mse, mae = model.evaluate(X_test, y_test, verbose=0)  
print("\nlook_back={:3d}, MSE={:8.4f}, MAE={:8.4f}".format(look_back,mse,mae))  
↪ look_back= 24, MSE= 0.2545, MAE= 0.3708
```

```
[18] y_pred = model.predict(X_test, verbose=0)  
print(y_pred.shape,y_test.shape)  
↪ (1608, 1) (1608, 1)
```

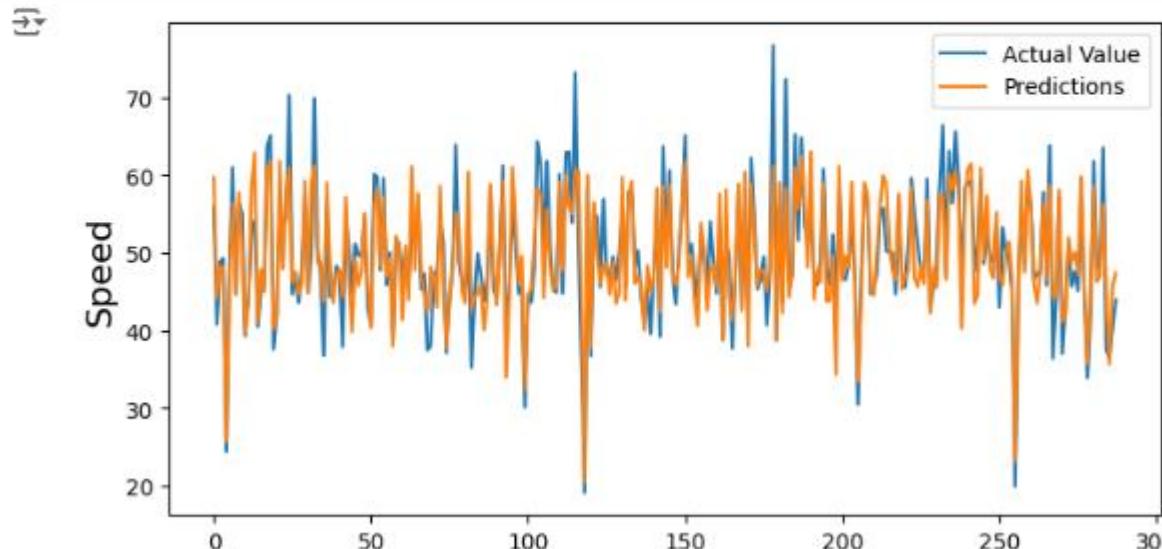
```
[19] y_pred = scaler.inverse_transform(y_pred)  
y_test = scaler.inverse_transform(y_test)
```

```
[20] pred_df = pd.DataFrame({'Actual': y_test.flatten(), 'Predicted': y_pred.flatten()})  
pred_df.head()  
↪
```

| | Actual | Predicted |
|---|--------|-----------|
| 0 | 55.9 | 59.723019 |
| 1 | 40.8 | 44.140583 |
| 2 | 48.8 | 48.223026 |
| 3 | 49.3 | 48.136997 |
| 4 | 24.4 | 25.747105 |

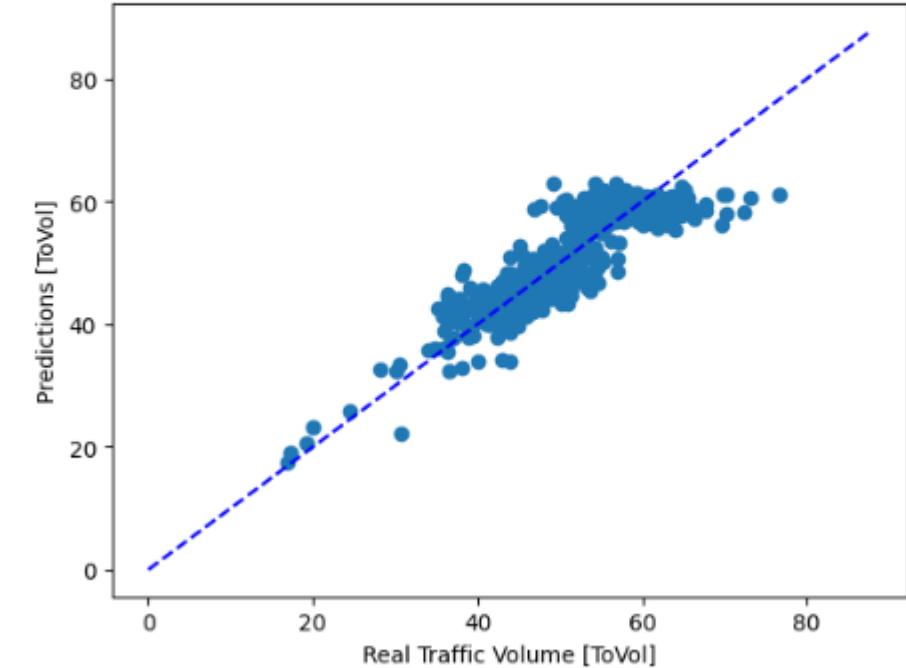
RNN으로 VDS 데이터 속도 예측

```
plt.figure(figsize=(8,4))
plt.ylabel('Speed', fontsize=16)
plt.plot(pred_df[:288])
plt.legend(['Actual Value', 'Predictions'])
plt.show()
```



```
[22] plt.scatter(y_test.flatten()[:576], y_pred.flatten()[:576])
plt.xlabel('Real Traffic Volume [ToVol]')
plt.ylabel('Predictions [ToVol]')
plt.plot([0, max_speed], [0, max_speed], 'b--')
```

```
[<matplotlib.lines.Line2D at 0x7963ba255b70>]
```

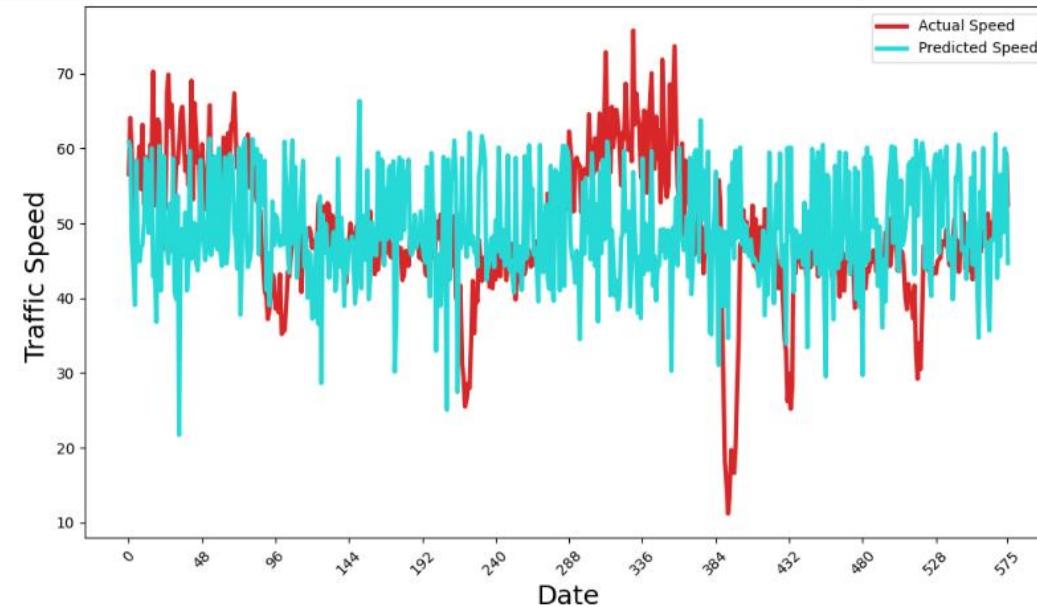


RNN으로 VDS 데이터 속도 예측

```
❶ plt.figure(figsize=(10,6))
# plt.style.use("dark_background")

plt.plot(df_spd[-576:], linewidth= 3, color="#d92628", label='Actual Speed')
# plt.plot(df["Date"].iloc[-576:], df_uni[-576:], linewidth= 3, color="#d92628", label='Actual Speed')
plt.plot(y_pred[-576:], color="#26D9D7", linewidth= 3, label='Predicted Speed')
plt.xlabel("Date", size="18")
plt.ylabel("Traffic Speed ", size="18")
plt.xticks([0, 48, 96, 144, 192, 240, 288, 336, 384, 432, 480, 528, 575], rotation=45)

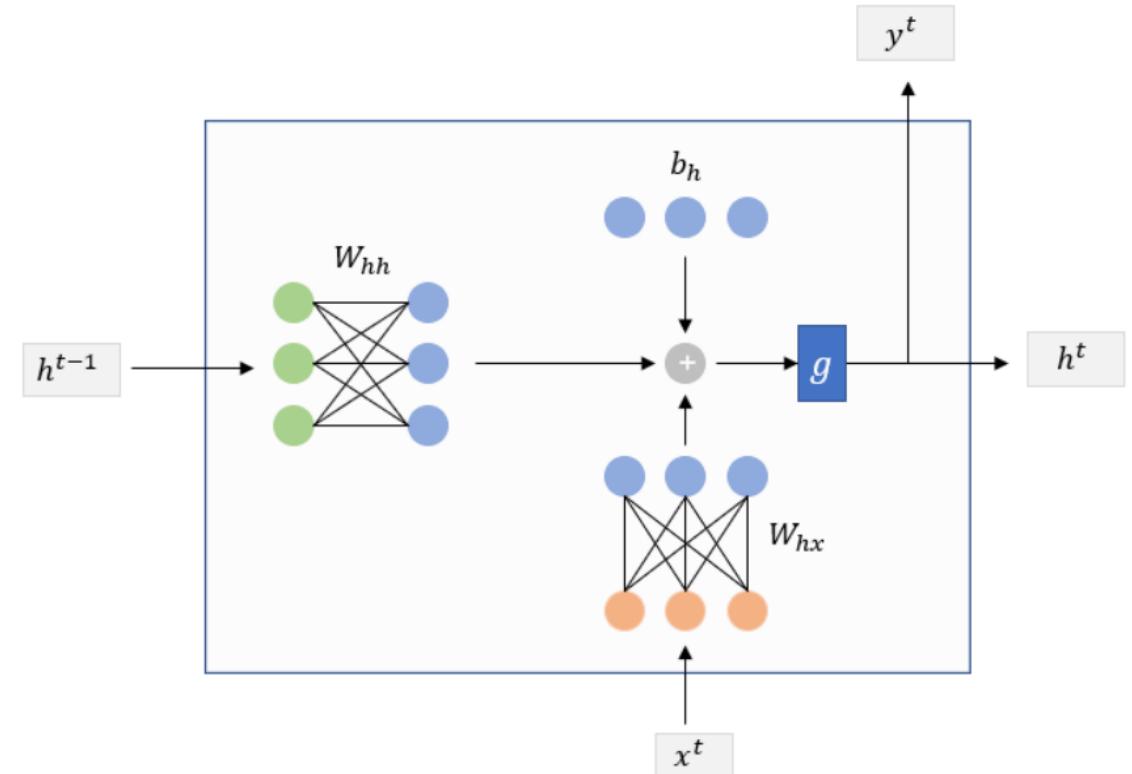
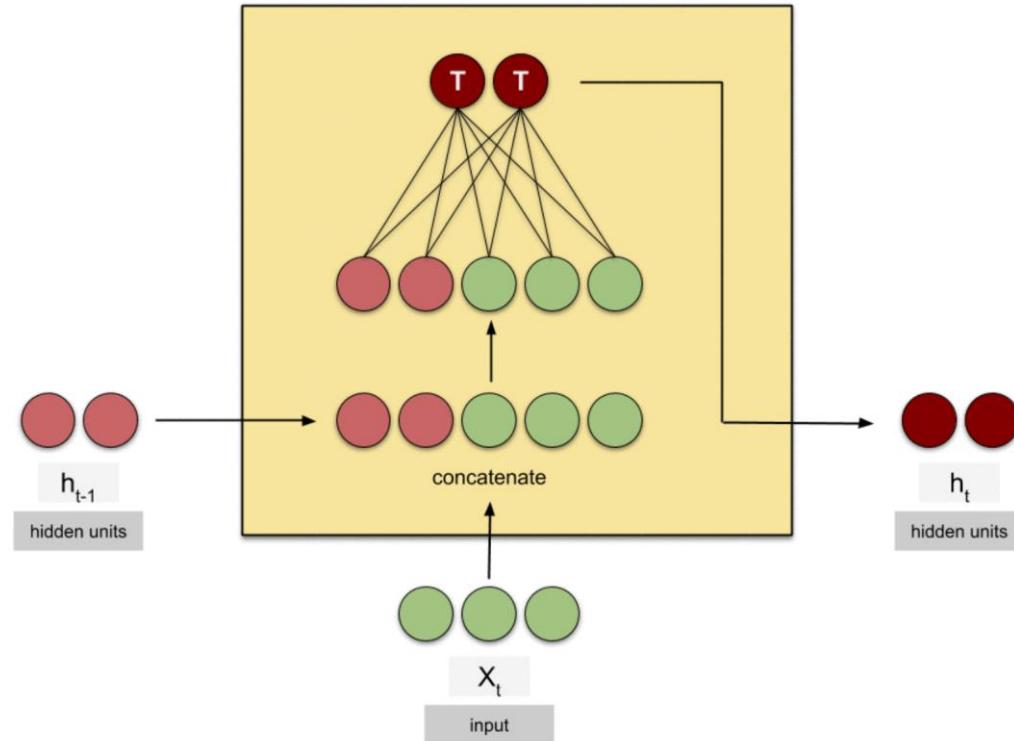
plt.tight_layout()
plt.legend()
```



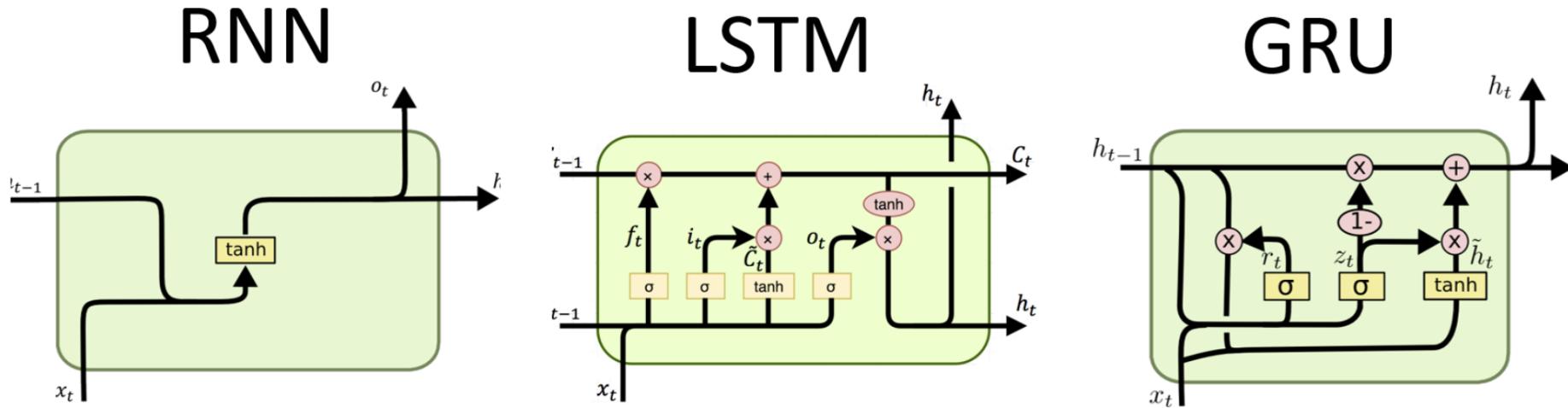
3-2

LSTM 모델 소개 및 실습

Vanilla RNN and SimpleRNN

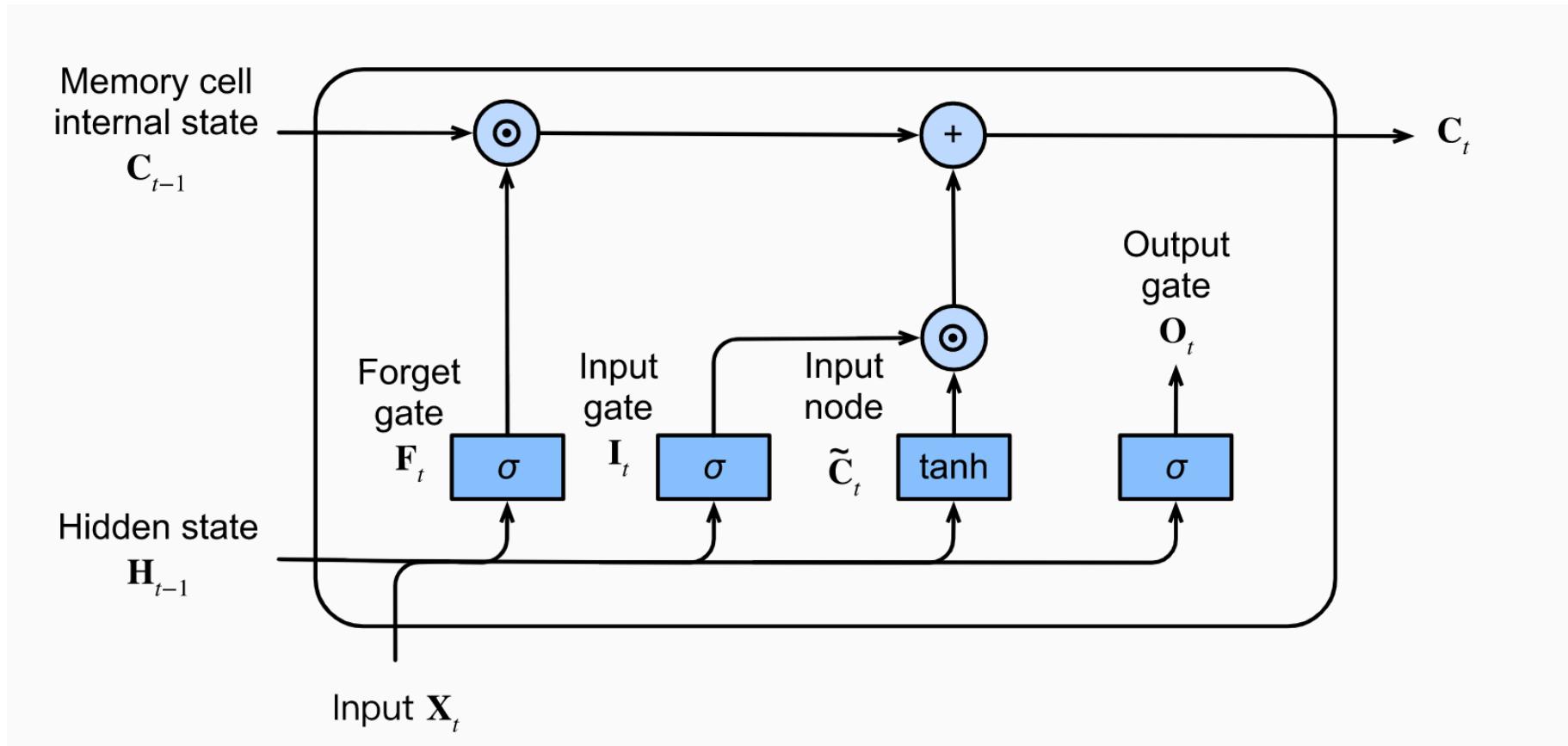


https://www.tensorflow.org/api_docs/python/tf/keras/layers/SimpleRNN



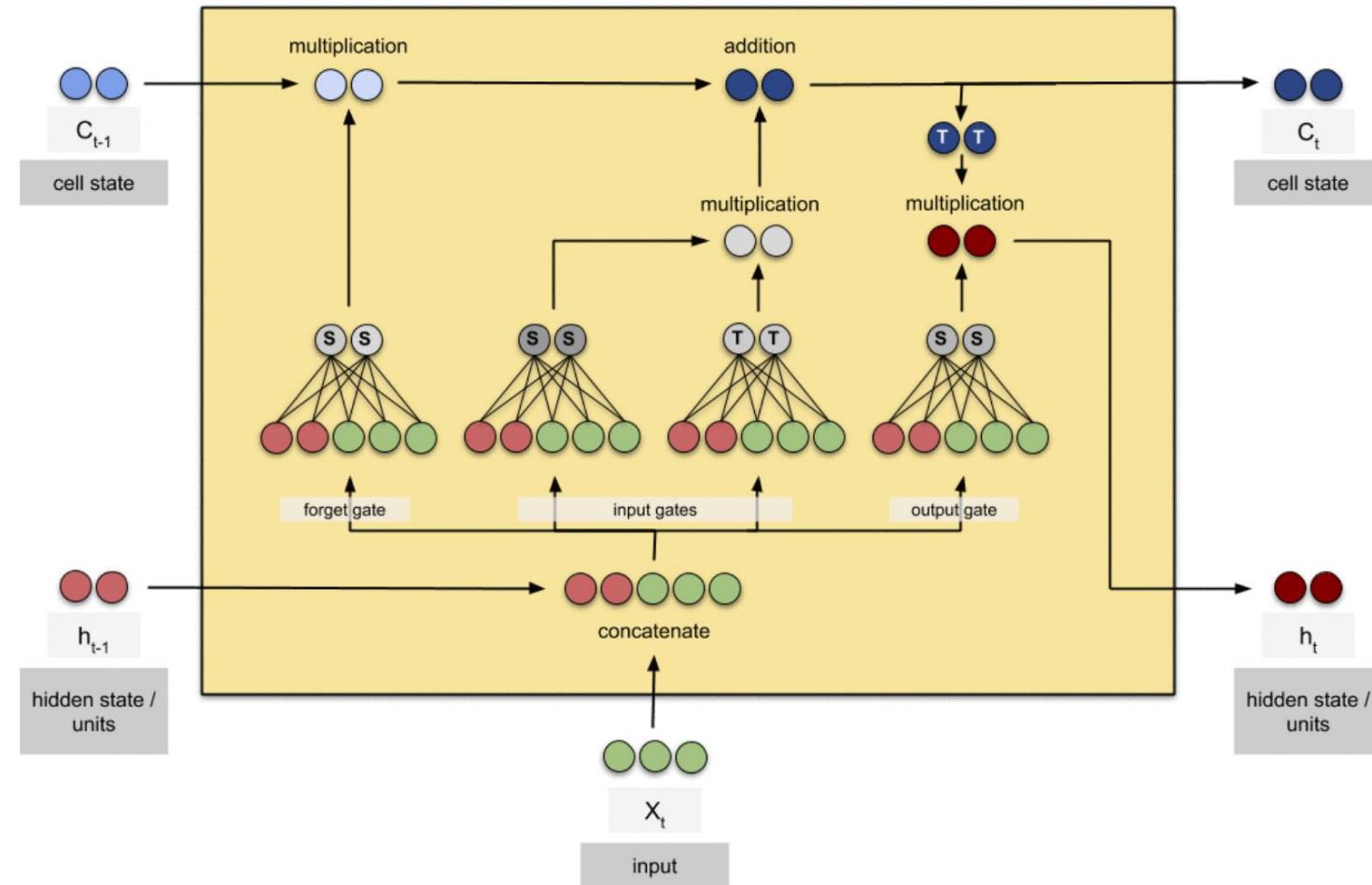
<http://dprogrammer.org/rnn-lstm-gru>

Long-Short Term Memory



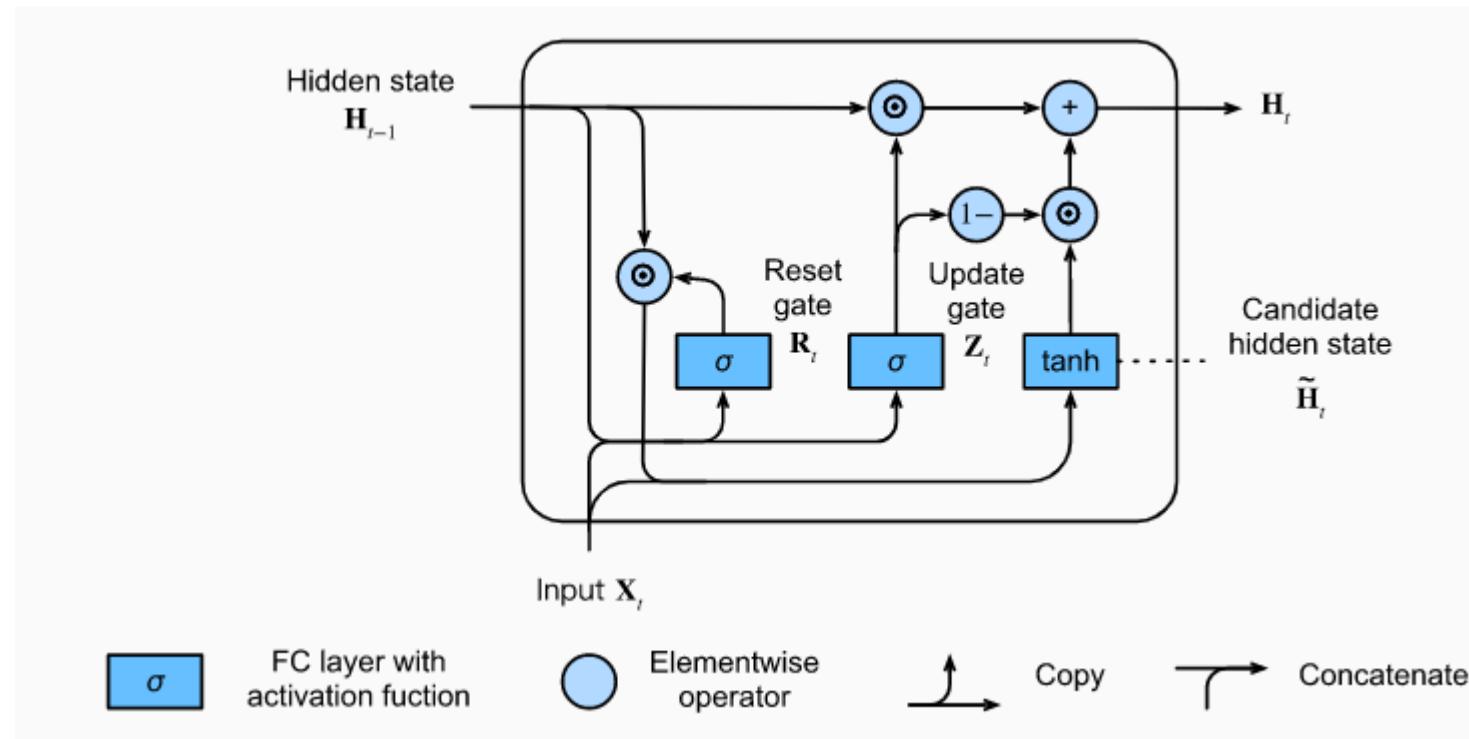
(source) https://d2l.ai/chapter_recurrent-modern/lstm.html

LSTM: Long-Short Term Memory



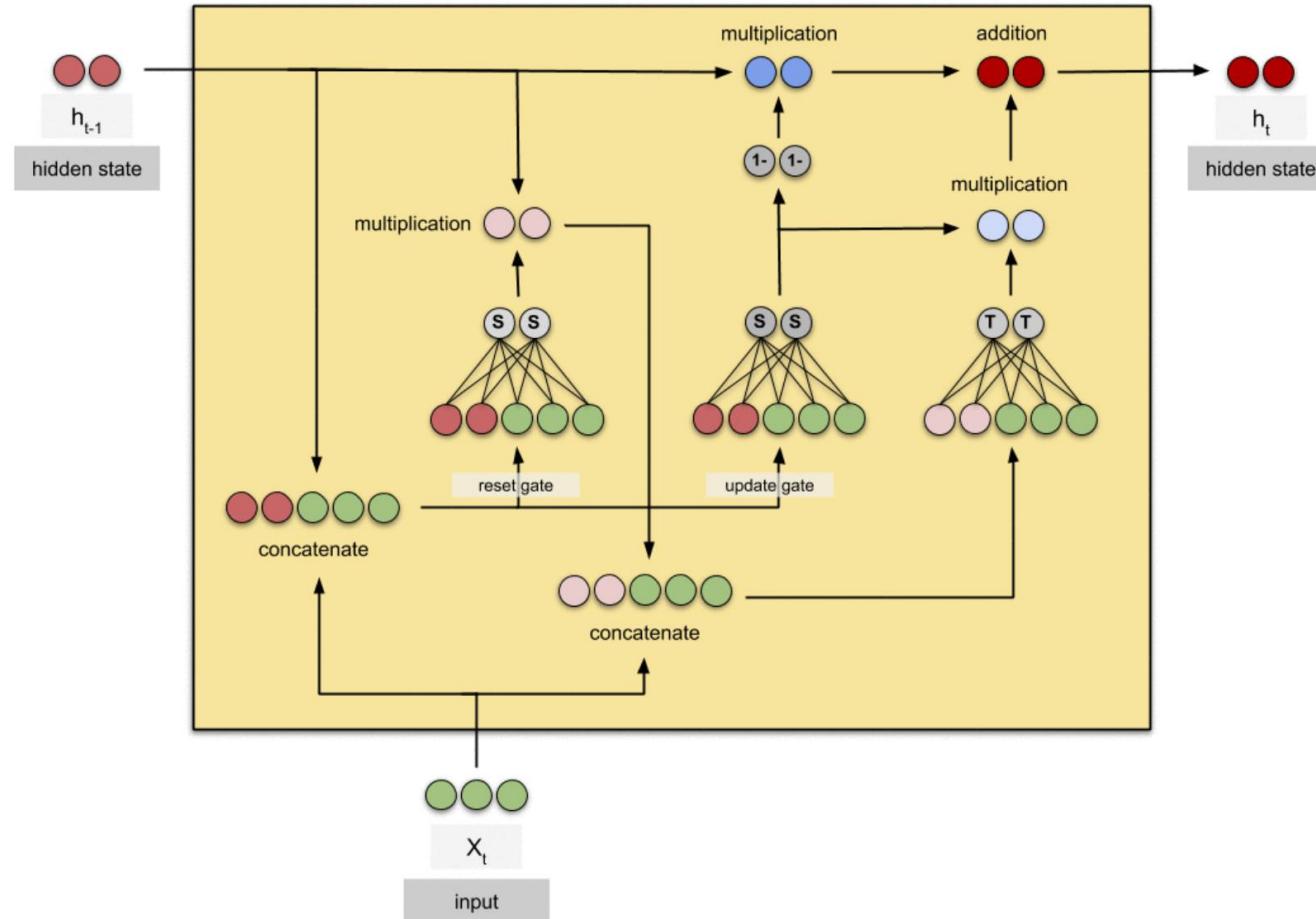
<https://towardsdatascience.com/animated-rnn-lstm-and-gru-ef124d06cf45>

Gated Recurrent Units



https://d2l.ai/chapter_recurrent-modern/gru.html

Animated GRU

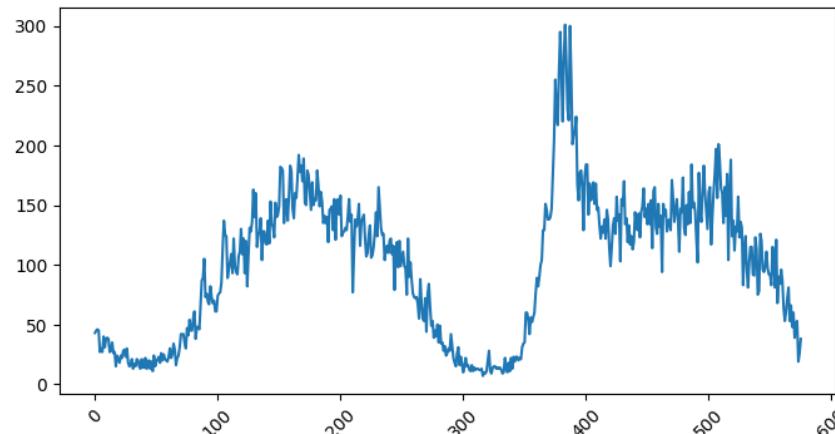


<https://towardsdatascience.com/animated-rnn-lstm-and-gru-ef124d06cf45>

필요한 라이브러리 추가

```
def plot_windows(X, Y, Y_pred):  
    plt.figure(figsize=(8, 4))  
    plt.plot(X[0,:], "o-")  
    if Y is not None:  
        plt.plot(np.arange(look_back, look_back + look_forward), Y[0, :], "go", label="Actual")  
    if Y_pred is not None:  
        plt.plot(np.arange(look_back, look_back + look_forward), Y_pred[0, :], "rx-", label="Forecast", markersize=10)  
    plt.legend(fontsize=12)  
    plt.grid(True)  
    plt.axvline(look_back, color='gray', linestyle='--', linewidth=2)  
    plt.show()  
  
def plot_pred():  
    plt.figure(figsize=(5, 4))  
    plt.scatter(y_test.flatten(), y_pred.flatten())  
    plt.xlabel('Actual')  
    plt.ylabel('Predictions [Sine Wave]')  
    plt.plot([0,1], [0,1],color='gray',linestyle='--',linewidth=2)  
    plt.show()
```

총 교통량(ToVol)을 예측해보자



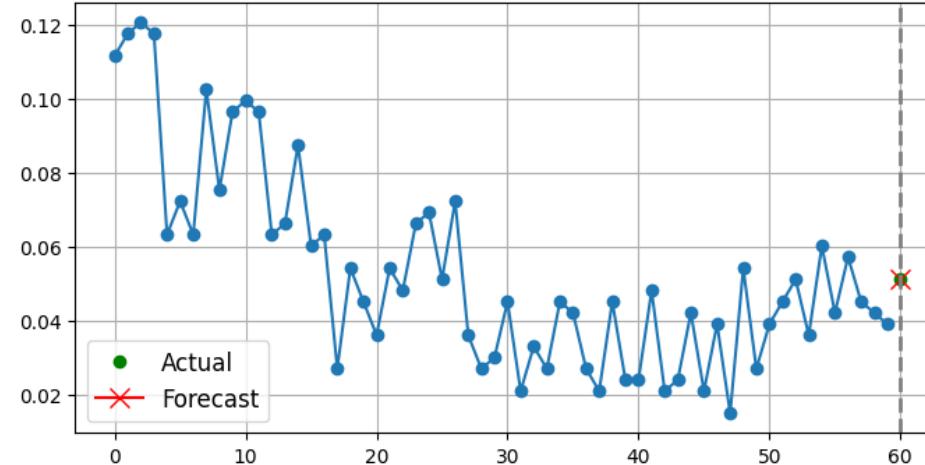
```
look_back = 12*5
look_forward = 1

X, y = create_dataset(df_scaled, look_back, look_forward)

X.shape, y.shape
((8003, 60, 1), (8003, 1, 1))

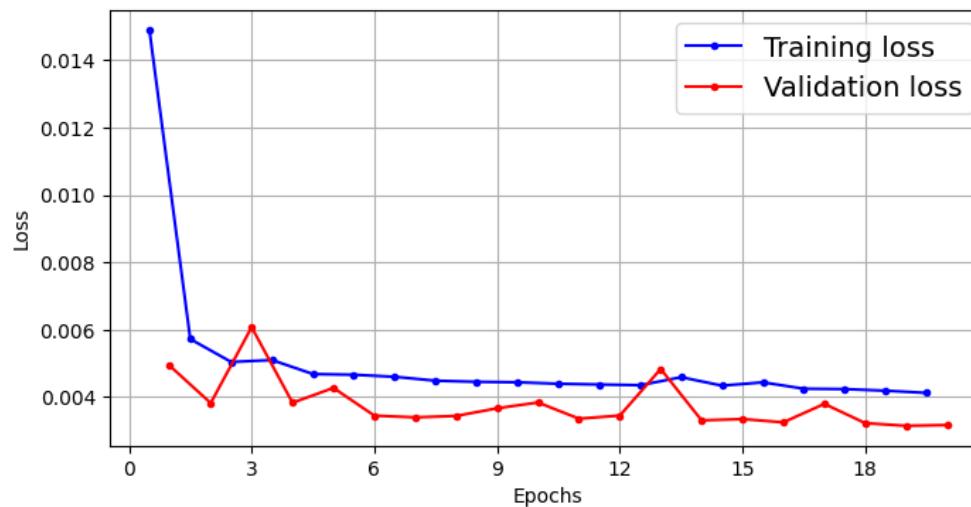
plot_windows(X,y,y)
```

```
# convert an array of values into a dataset matrix
def create_dataset(dataset, look_back, look_forward):
    dataX, dataY = [], []
    np.array(dataY)
    for i in range(len(dataset)-look_back-1):
        a = dataset[i:(i+look_back)]
        if len(dataset[i + look_back:i+look_back+look_forward]) == look_forward:
            dataX.append(a)
            dataY.append(dataset[i + look_back:i+look_back+look_forward])
    return np.array(dataX), np.array(dataY)
```

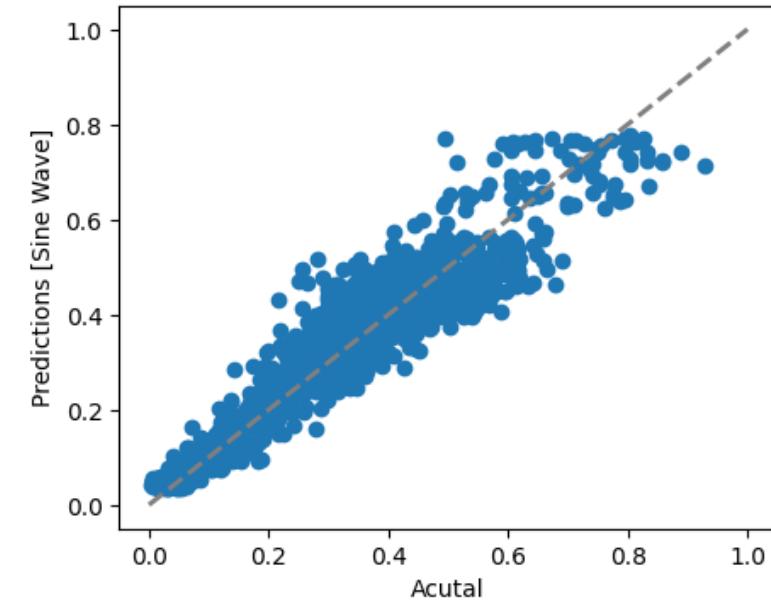
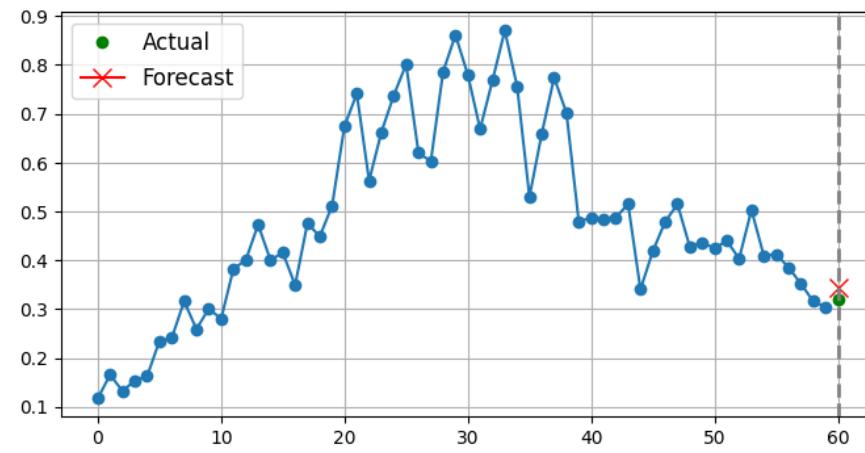


4) many-to-one with LSTM

```
def model_lstm():
    model = Sequential([
        LSTM(32, return_sequences=True, input_shape=[look_back, 1]),
        LSTM(32, return_sequences=True),
        LSTM(look_forward, return_sequences=False)
    ])
    model.compile(loss="mse", optimizer='adam', metrics=['mae'])
    model.summary()
    return model
```

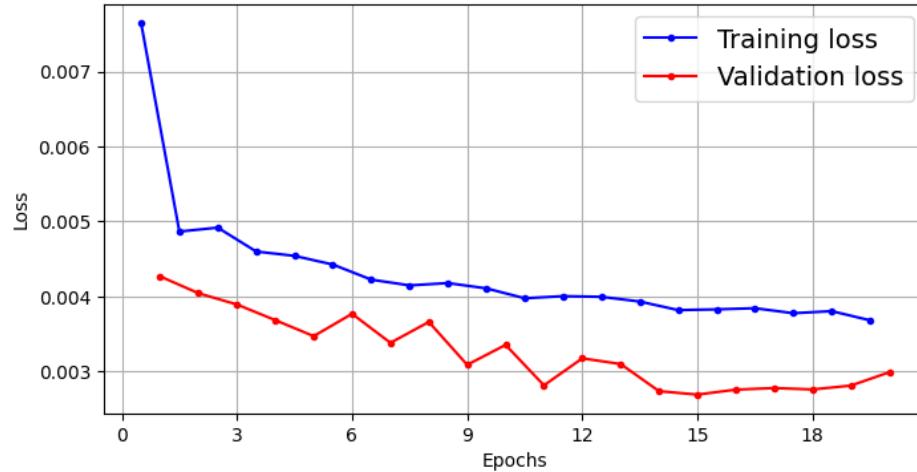


51/51 [=====]
m1_LSTM: 0.003614206099882722

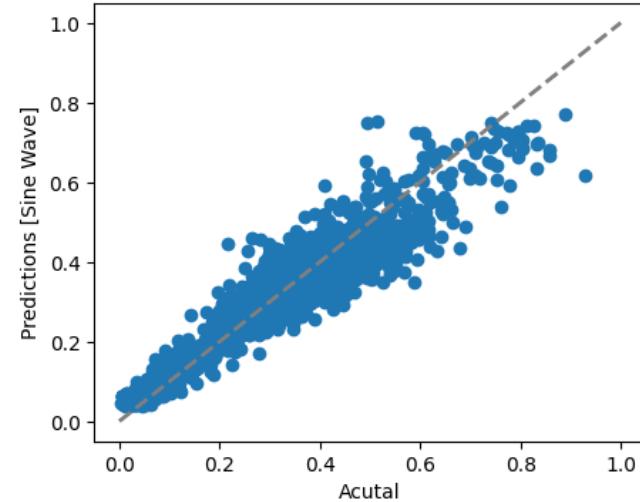
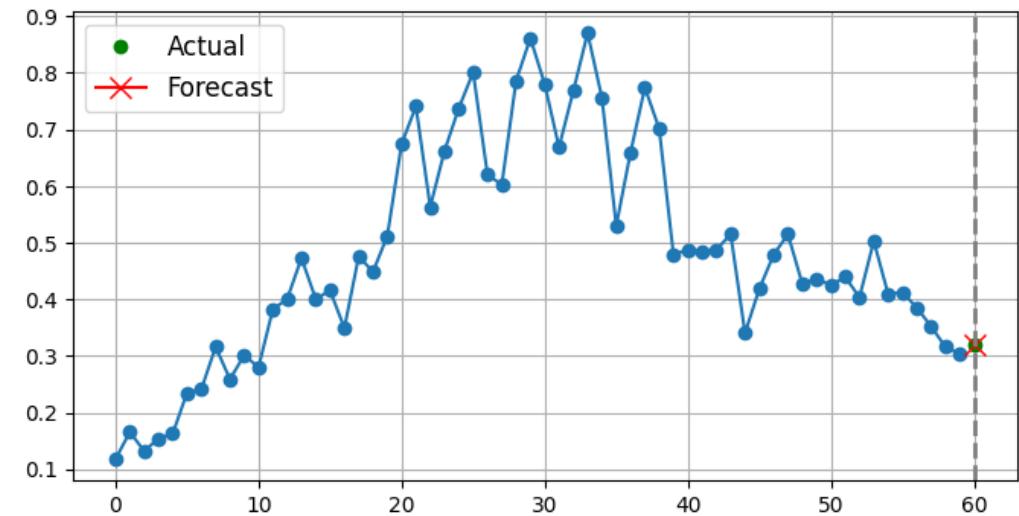


5) many-to-one with GRU

```
def model_gru():
    model = Sequential([
        GRU(32, return_sequences=True, input_shape=[look_back, 1]),
        GRU(32, return_sequences=True),
        GRU(look_forward, return_sequences=False),
    ])
    model.compile(loss="mse", optimizer='adam', metrics=['mae'])
    model.summary()
    return model
```



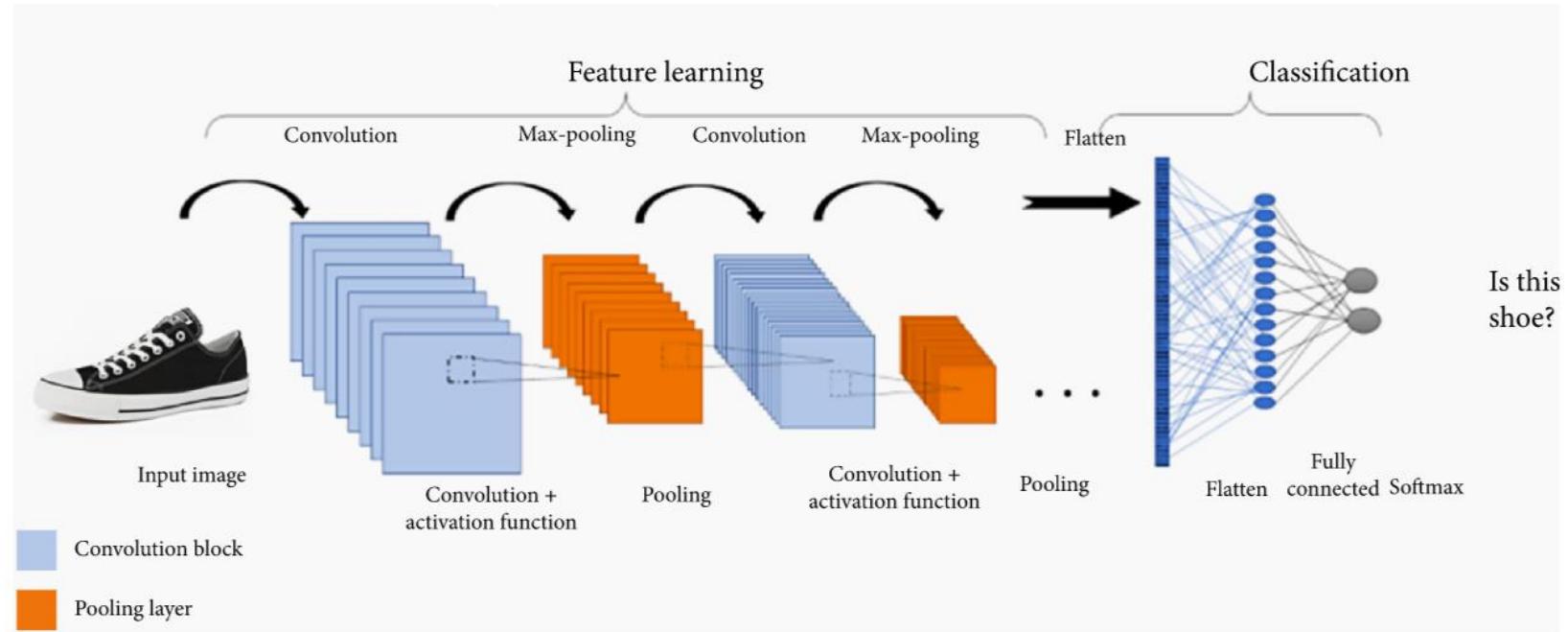
51/51 [=====]
m1_GRU: 0.0037777069956064224



3-3

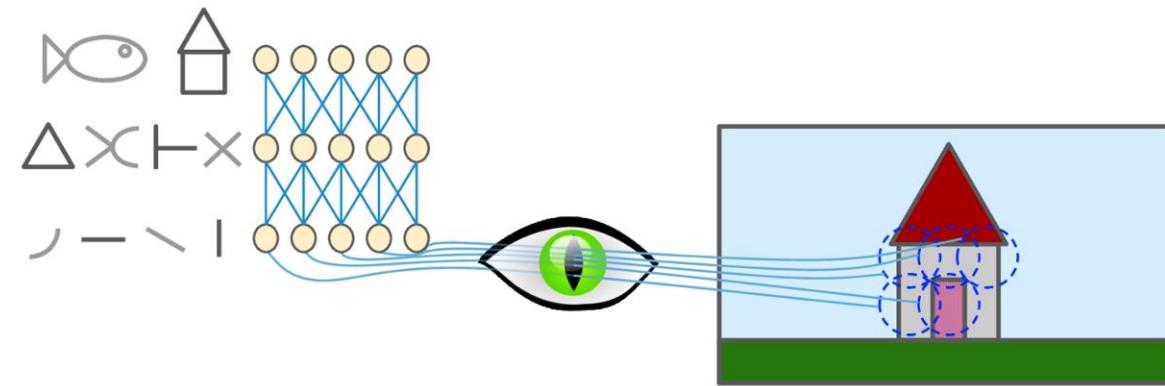
CNN 모델 소개

CNNs for Image Classification



source : <https://www.hindawi.com/journals/wcmc/2022/7549397/fig2/>

- ❖ Hubel과 Wiesel은 시각피질의 세포들을 여러 유형으로 구별 (1959)

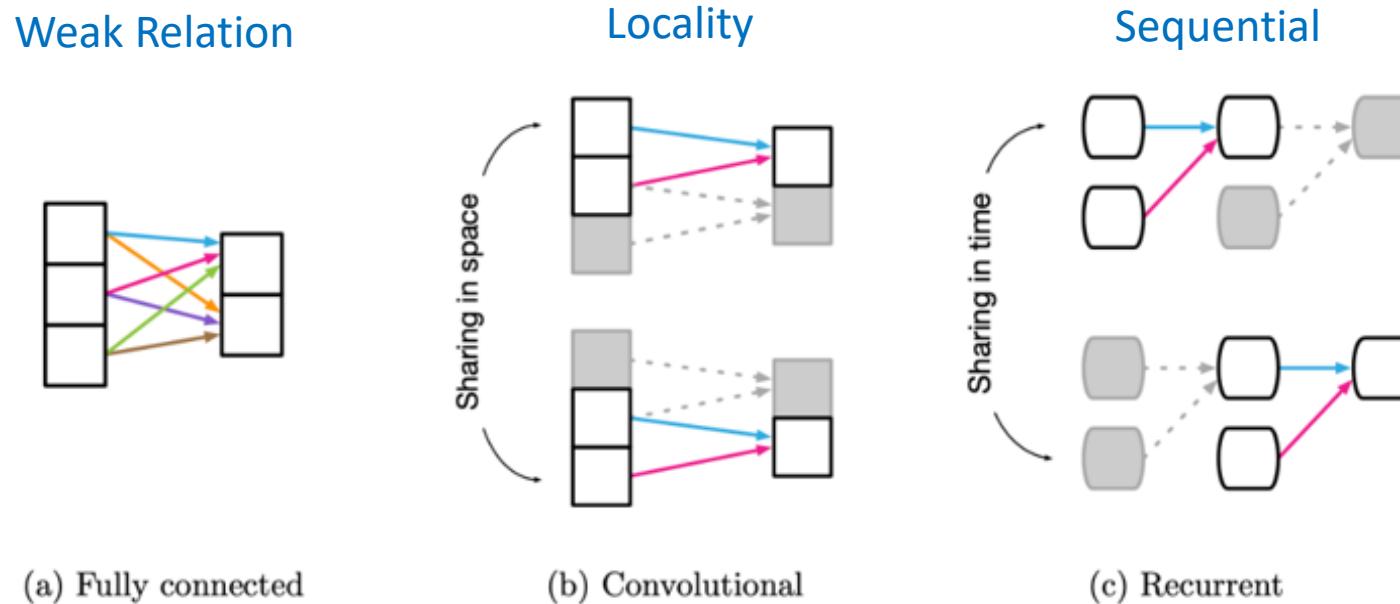


합성곱 신경망은 뉴런 사이의 연결 패턴이 동물 시각 피질의 조직과 유사하다는 점에 영감을 받았다. 개별 피질 뉴런은 수용장(receptive field)으로 알려진 시야의 제한된 영역에서만 자극에 반응한다. 상이한 뉴런의 수용 필드는 전체 시야를 볼 수 있도록 부분적으로 중첩된다.[\[출처 필요\]](#)

The Main Concepts Behind Convolutional Neural Networks

❖ 딥러닝에서 관계 귀납 편향(Relational inductive biases)

학습시 정확한 예측을 위해 추가적인 가정을 하는 것

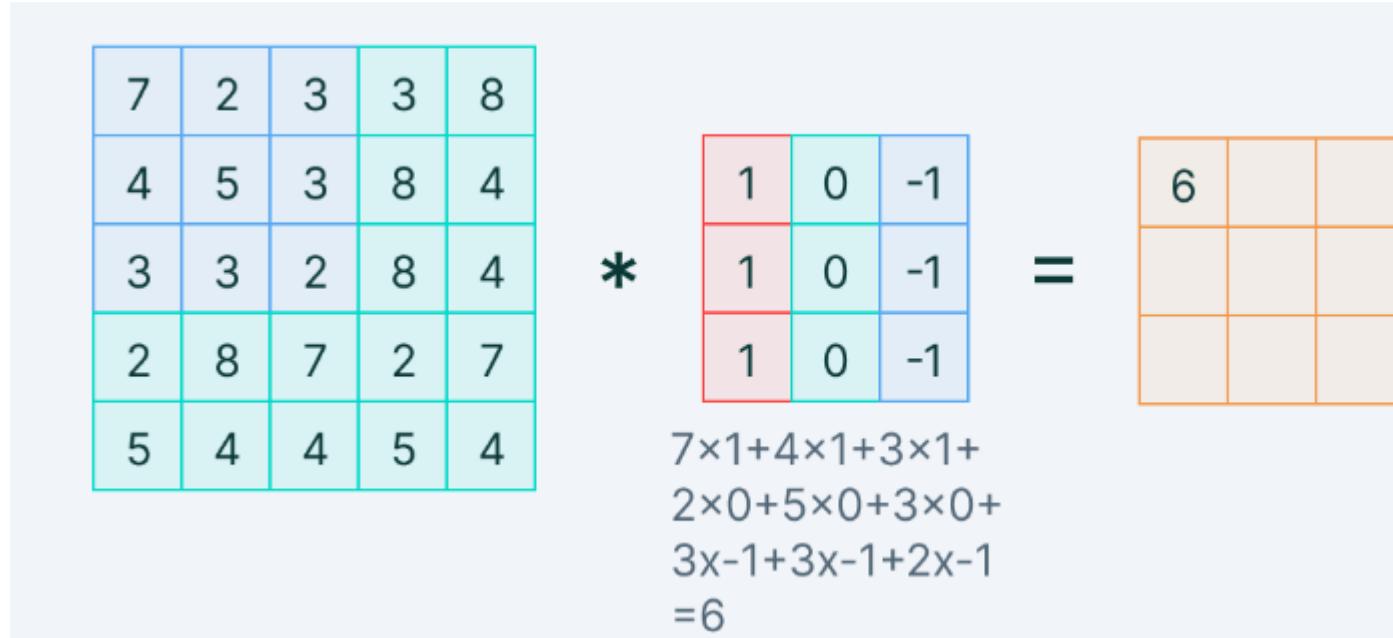


source : <https://www.baeldung.com/cs/ml-inductive-bias>

CNN works by comparing images piece by piece

❖ Convolution Operation (합성곱 연산)

- ✓ Element-wise multiplication between the filter-sized patch of the input image
- ✓ And filter is done, which is then summed.



The diagram illustrates a convolution operation. On the left is a 5x5 input image with values:

| | | | | |
|---|---|---|---|---|
| 7 | 2 | 3 | 3 | 8 |
| 4 | 5 | 3 | 8 | 4 |
| 3 | 3 | 2 | 8 | 4 |
| 2 | 8 | 7 | 2 | 7 |
| 5 | 4 | 4 | 5 | 4 |

In the center, a 3x3 filter is shown with values:

| | | |
|---|---|----|
| 1 | 0 | -1 |
| 1 | 0 | -1 |
| 1 | 0 | -1 |

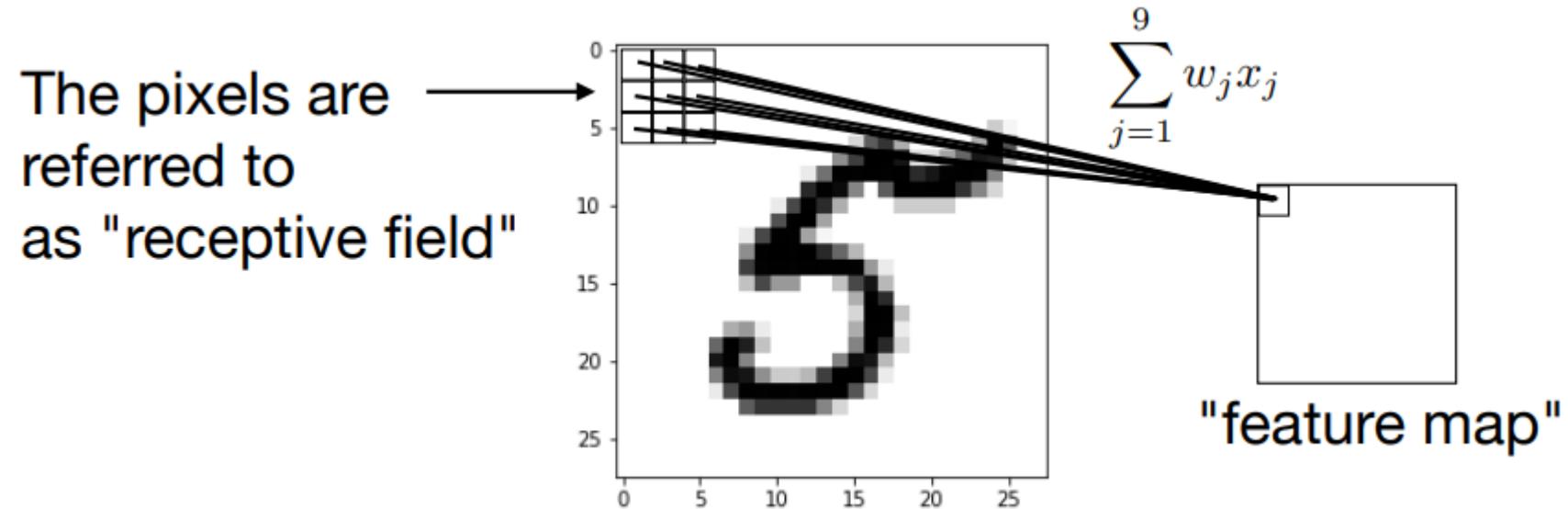
The operation is represented by the formula:

$$7 \times 1 + 4 \times 1 + 3 \times 1 + \\ 2 \times 0 + 5 \times 0 + 3 \times 0 + \\ 3 \times -1 + 3 \times -1 + 2 \times -1 \\ = 6$$

(source) <https://www.oreilly.com/library/view/learning-tensorflow/9781491978504/ch04.html>

RNN 처럼 Weight Sharing

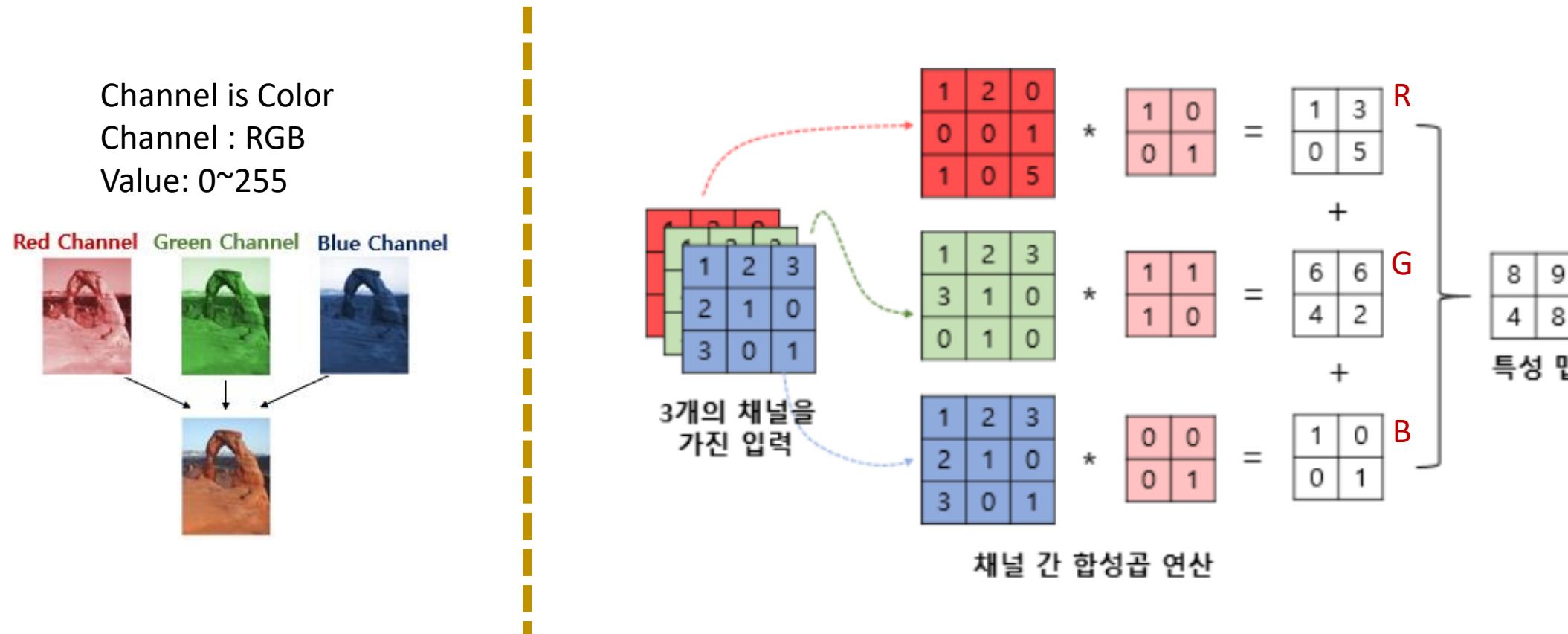
- ❖ A filter(kernel) slides over the inputs to generate a feature map



source : <https://pages.stat.wisc.edu/~sraschka/teaching/stat453-ss2021/>

Convolutions with Color Channels

- ❖ For multiple channels, the feature map does not contain RGB meanings

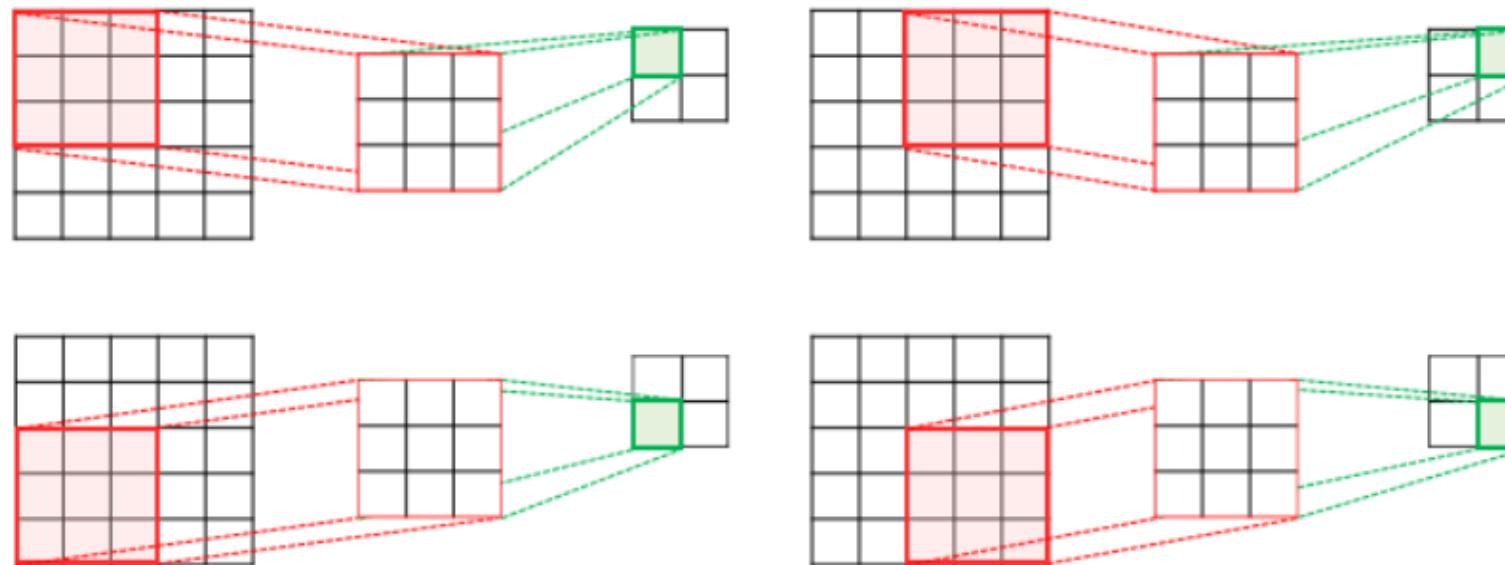


(source) <https://wikidocs.net/64066>

Strided Convolution

❖ Stride is range of movement

- ✓ 5×5 input (spatially) assume 3×3 filter applied with stride,- ✓ and finally 2×2 size feature map

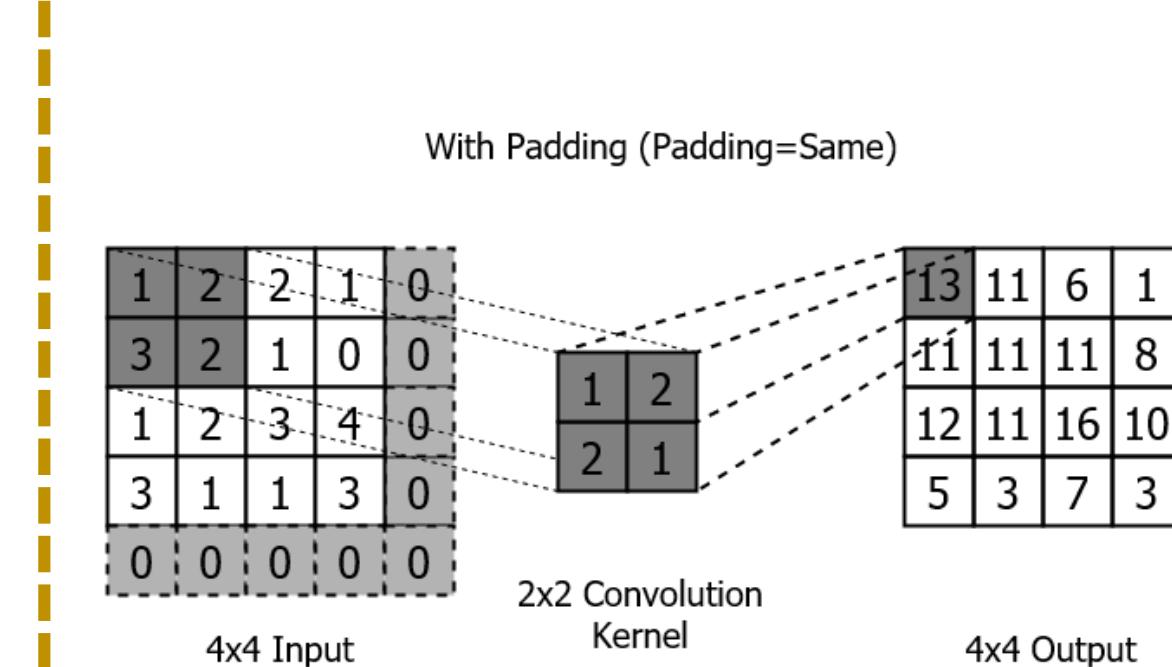
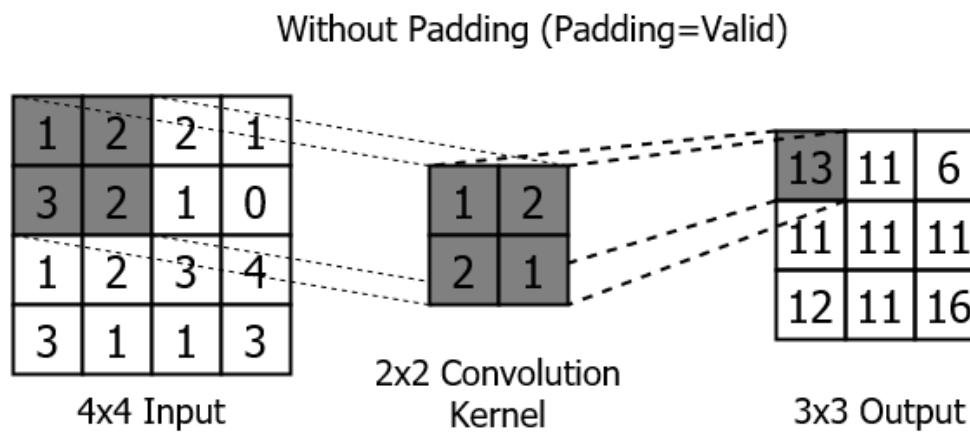


(source) <https://wikidocs.net/64066>

Padding basically extends the area of an image

- ❖ Zero padding fills zero

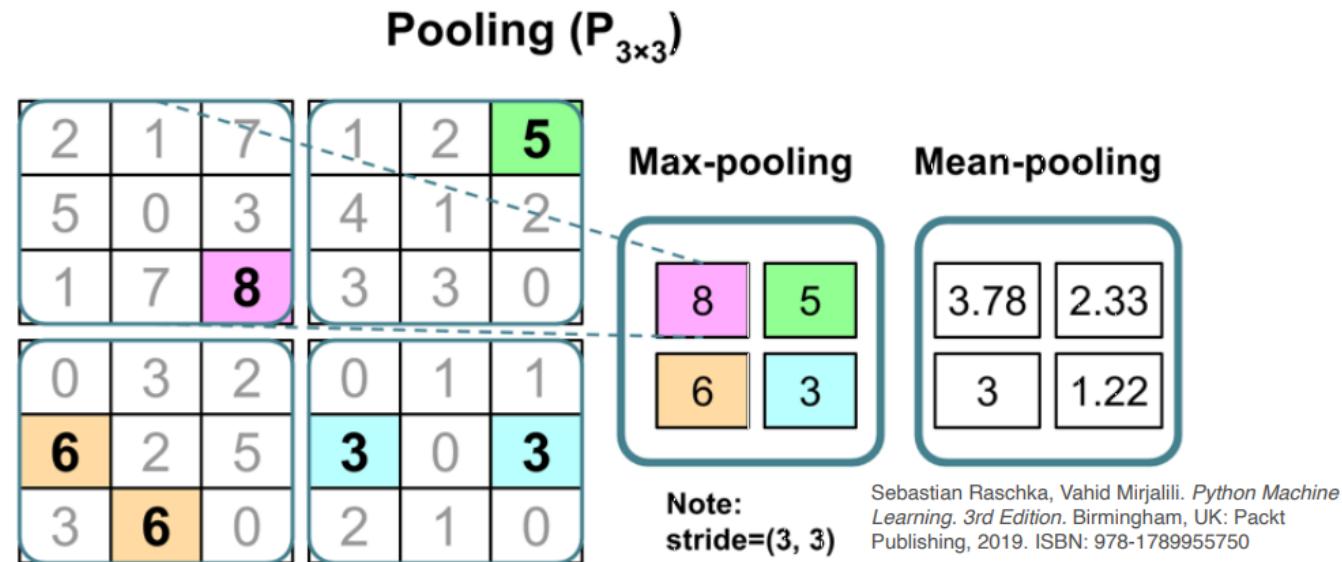
- ✓ padding="valid", padding="same"



(source) <https://livebook.manning.com/book/tensorflow-in-action/chapter-4/v-1/>

Pooling Layers Can Help With Local Invariance

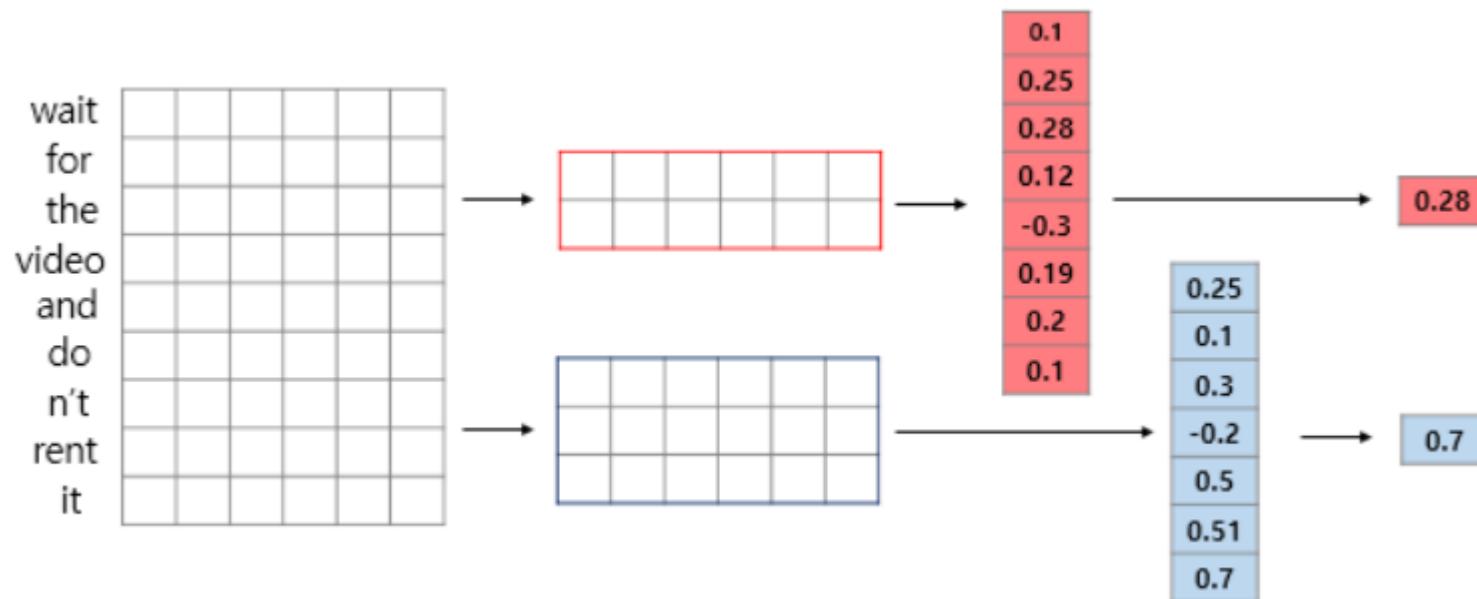
- ❖ Pooling : information is lost.
 - ✓ relative position is important (like face recognition)



source : <https://pages.stat.wisc.edu/~sraschka/teaching/stat453-ss2021/>

1D CNN에서 Max Pooling

- ❖ 1D CNN 예시
 - ✓ 합성곱+활성함수 다음에 풀링층을 적용
 - ✓ 맥스풀링 사용 (커널크기는 2 혹은 3)
- ❖ 커널 너비는 특성(feature)의 길이이고 커널의 높이는 1차원 CNN의 커널 크기



입력 데이터 다운로드

```
import tensorflow as tf
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
df = pd.read_csv('https://raw.githubusercontent.com/hongsukyi/Lectures/main/vds/vds.csv')
df.head(2)
```

| | Date | ToVol | SmVol | MeVol | LaVol | Speed | Occ.Rate |
|---|-----------------|-------|-------|-------|-------|-------|----------|
| 0 | 2017-04-02 0:00 | 43 | 34 | 9 | 0 | 50.3 | 1.90 |
| 1 | 2017-04-02 0:05 | 45 | 32 | 13 | 0 | 58.9 | 1.84 |

속도 예측을 위한 정규화

```
df_uni = df.iloc[:,5:6].values
```

```
df_uni
```

```
array([[50.3],  
       [58.9],  
       [50.6],  
       ...,  
       [50.6],  
       [59.3],  
       [52.5]])
```

```
print(df_uni.shape)
```

```
(8064, 1)
```

```
from sklearn.preprocessing import MinMaxScaler
```

```
scaler = MinMaxScaler(feature_range = (0,1))  
scaled_df = scaler.fit_transform(df_uni)
```

```
scaled_df
```

```
array([[0.52350699],  
       [0.63278272],  
       [0.52731893],  
       ...,  
       [0.52731893],  
       [0.63786531],  
       [0.55146125]])
```

```
look_back = 12*24
```

RNN 처럼 Conv1d를 위한 데이터 구조:

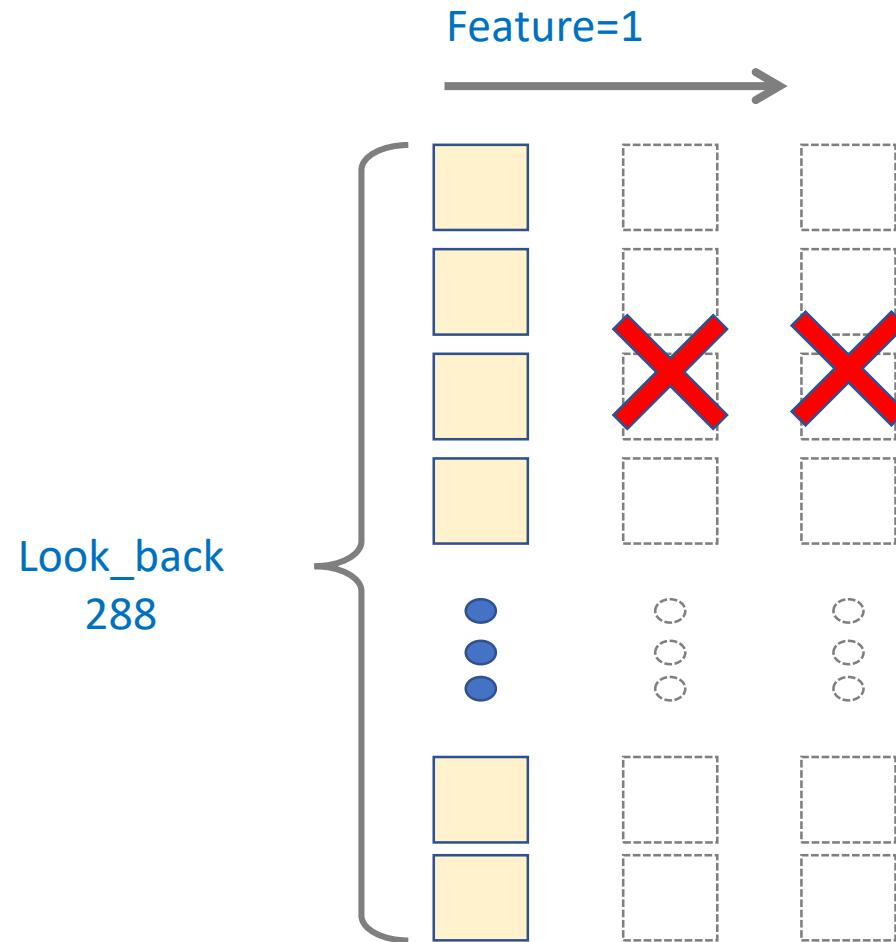
```
look_back = 12*24
```

```
X = []
y = []
for i in range(len(scaled_df)-look_back-1):
    X.append(scaled_df[i:(i+look_back)])
    y.append(scaled_df[(i+look_back)])
```

```
X = np.array(X)
y = np.array(y)
```

```
print(X.shape, y.shape)
```

```
(7775, 288, 1) (7775, 1)
```

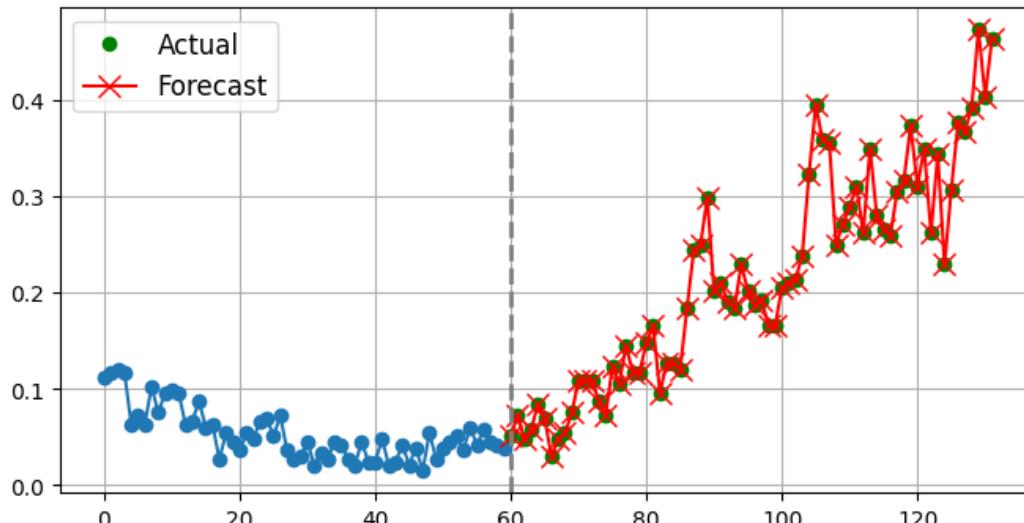


Multi-step ahead prediction model

```
look_back = 12*5  
look_forward = 12*6
```

```
X, y = create_dataset(df_scaled, look_back, look_forward)
```

```
plot_windows(X,y,y)
```



```
X_train, X_test, y_train, y_test = train_test_split(X, y,  
test_size=0.20, shuffle=False)
```

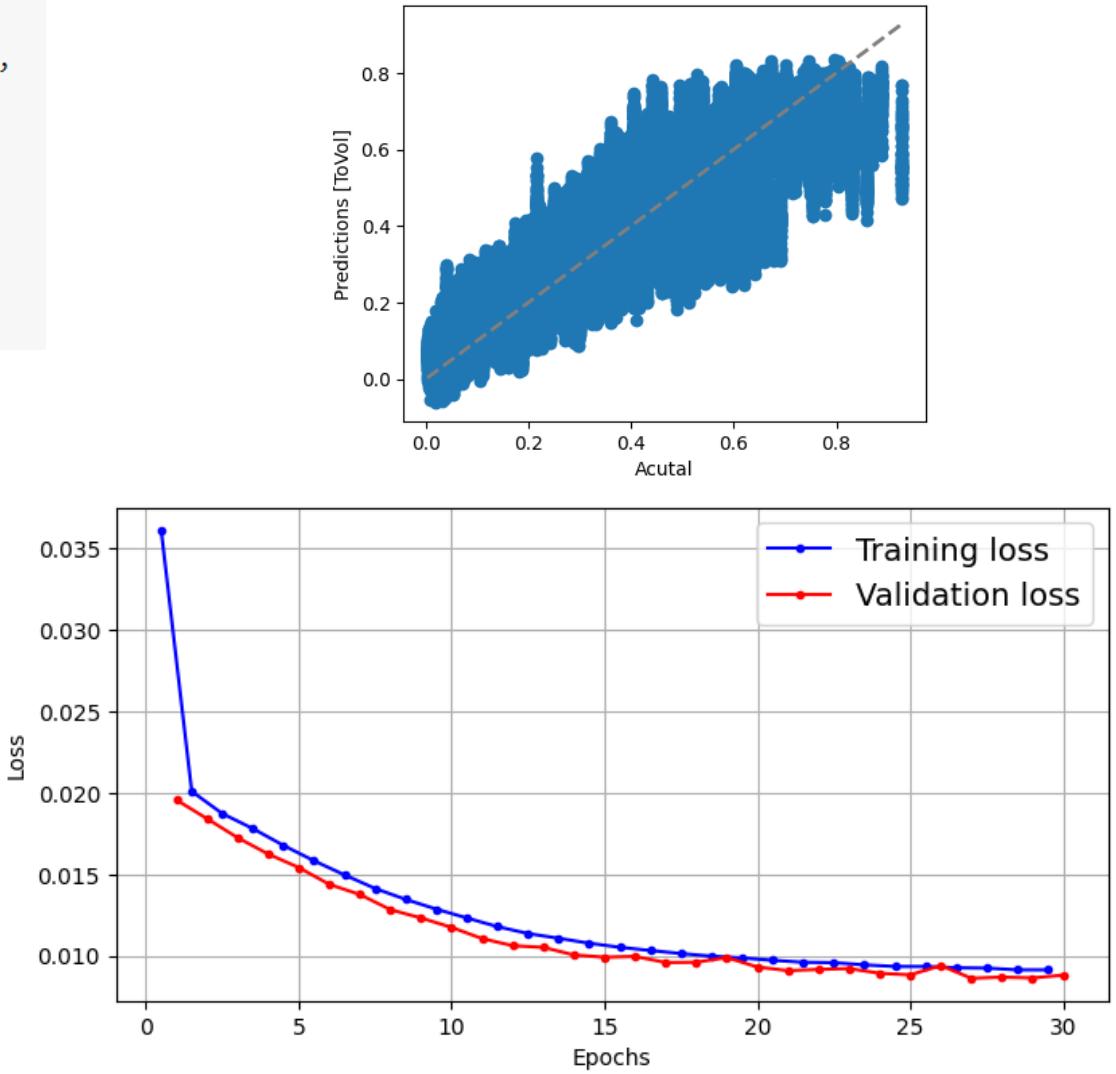
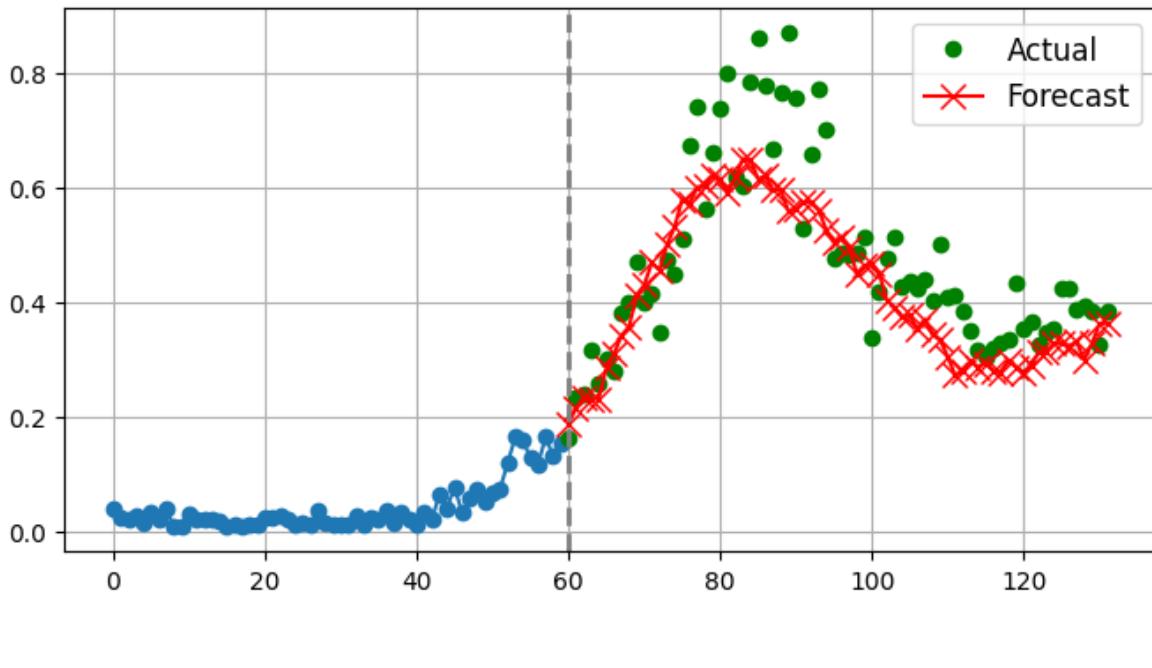
```
print('X_train:', X_train.shape)  
print('X_test :', X_test.shape)  
print('y_train:', y_train.shape)  
print('y_test :', y_test.shape)
```

```
X_train: (6346, 60, 1)  
X_test : (1587, 60, 1)  
y_train: (6346, 72, 1)  
y_test : (1587, 72, 1)
```

```
def model_dnn():  
    model = Sequential([ Flatten(input_shape=[look_back,1]),  
                        Dense(100, activation='relu'),  
                        Dense(look_forward) ])  
    optimizers = tf.keras.optimizers.Adam(learning_rate=0.01)  
    model.compile(loss="mse", optimizer=optimizers,metrics =['mae'])  
    model.summary()  
    return model
```

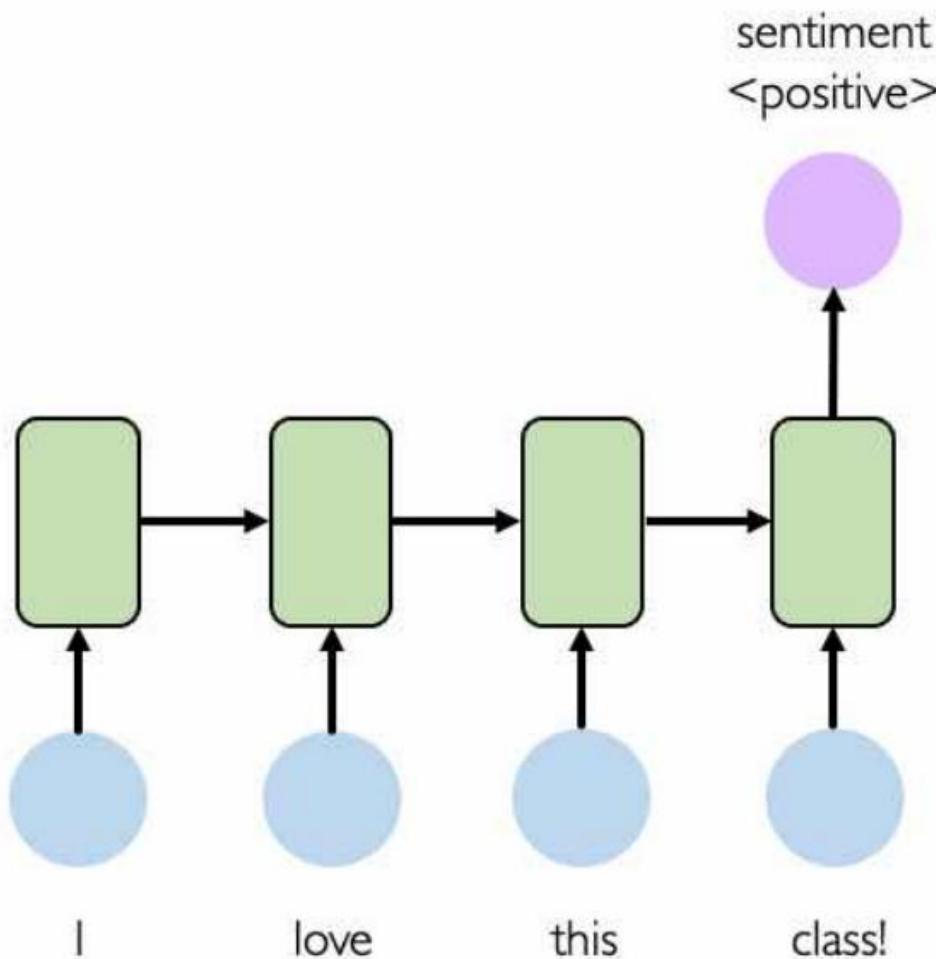
(b5) many to many with Conv1D

```
def model_conv1d():
    model = Sequential([
        Conv1D(filters=32, kernel_size=3, strides=1, padding='valid', activation='relu',
               input_shape = (look_back,1)),
        MaxPool1D(2, padding = 'valid'),
        Flatten(),
        Dense(32, activation='relu'),
        Dense(look_forward, activation='relu')
    ])
    model.compile(optimizer='adam', loss= 'mse', metrics=['mae'])
    return model
```



3-4

Attention Is All You Need



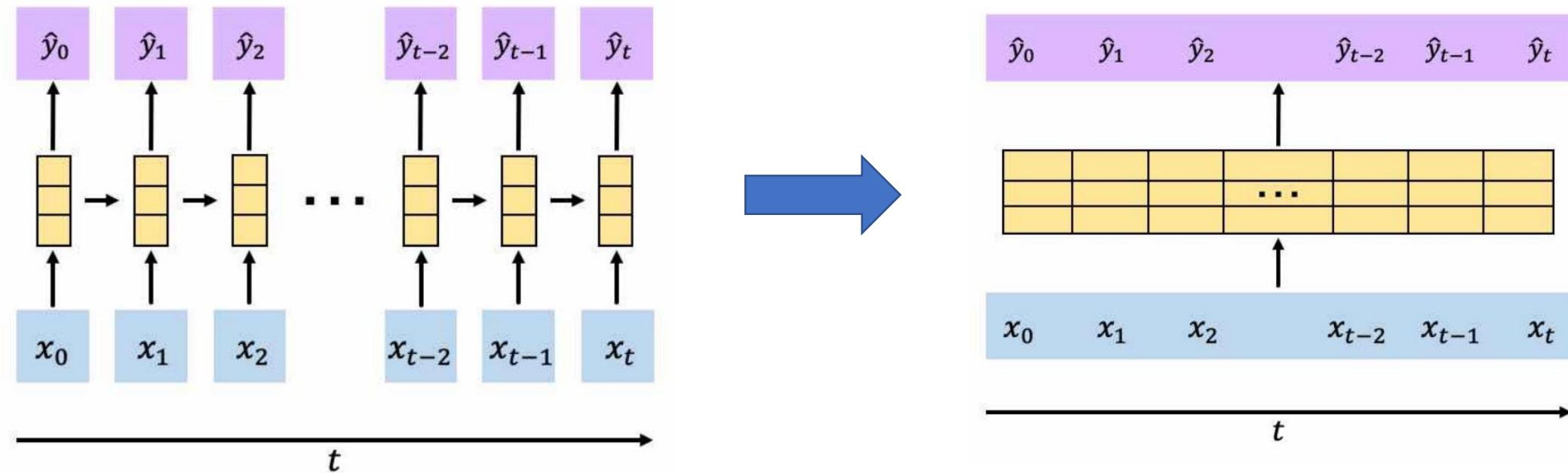
Limitations of RNNs

- Encoding bottleneck
- Slow, no parallelization
- Not long memory

Goal of Sequence Modeling

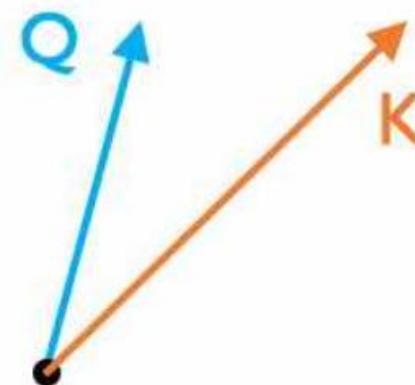
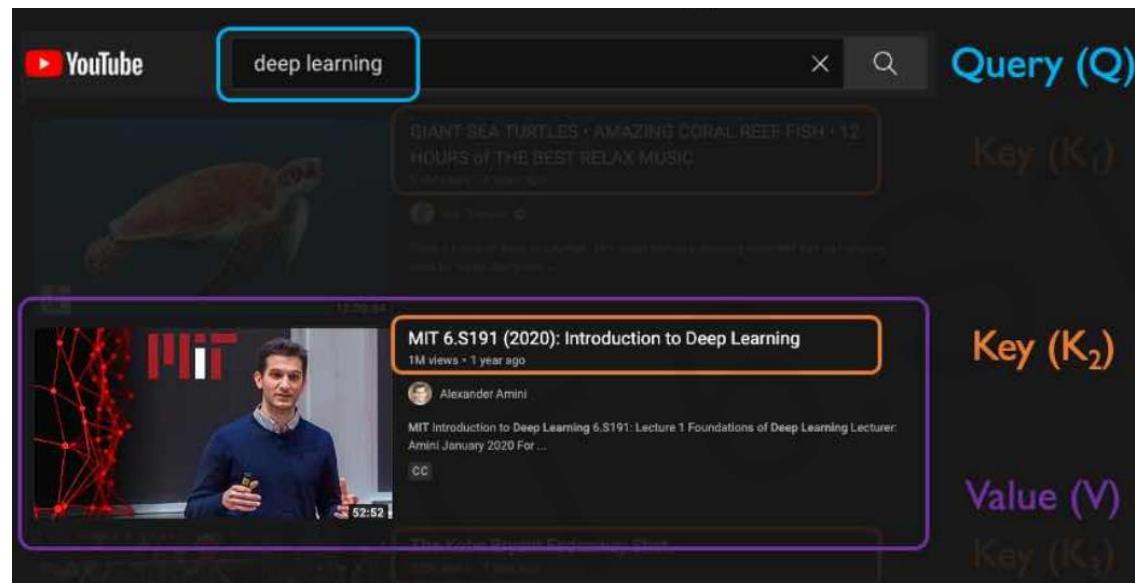
❖ RNNs: recurrence to model sequence dependencies

- ✓ Limitation of RNNs → Desired Capabilities
 - Encoding bottleneck → Continuous stream
 - No parallelization → Parallelization



Intuition Behind Self-Attention

- ❖ Attending to the most important parts of an input.
 - ✓ Identify which parts to attend to → Similar to Search Process
 - ✓ Extract the features with high attentions
- ❖ A simple example of Search : Query, Key, Value
 - ✓ How similar is the key to the query?



Dot product → Scale

$$\frac{Q \cdot K^T}{\text{scaling}}$$

Similarity metric

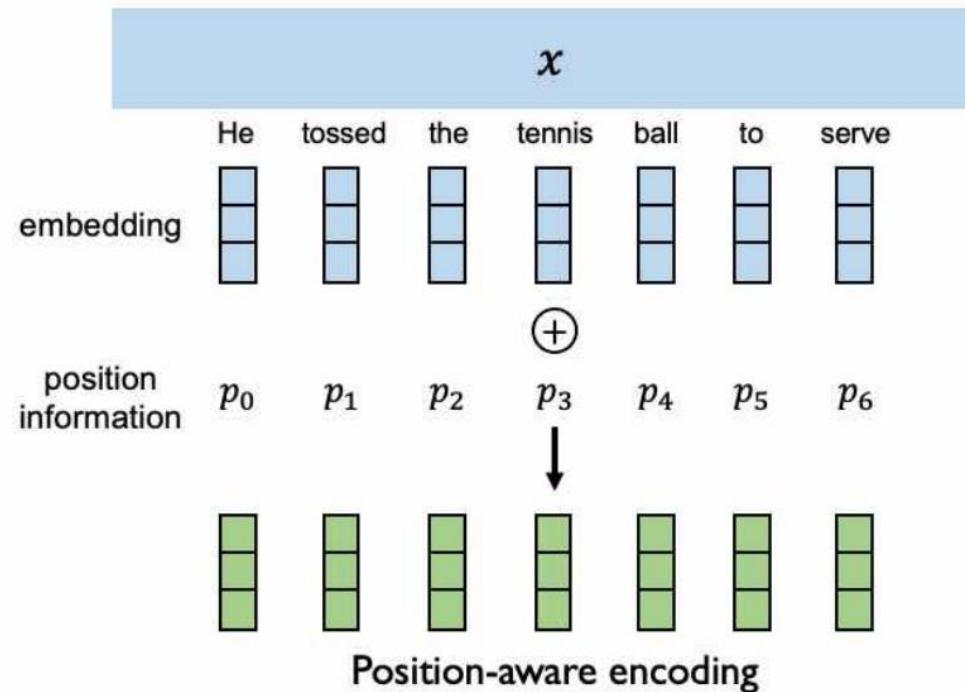
Also known as the "cosine similarity"

Learning Self-Attention with Neural Networks

- ❖ Goal: identify and attend to most important features in input!!

- ✓ Encoding position information

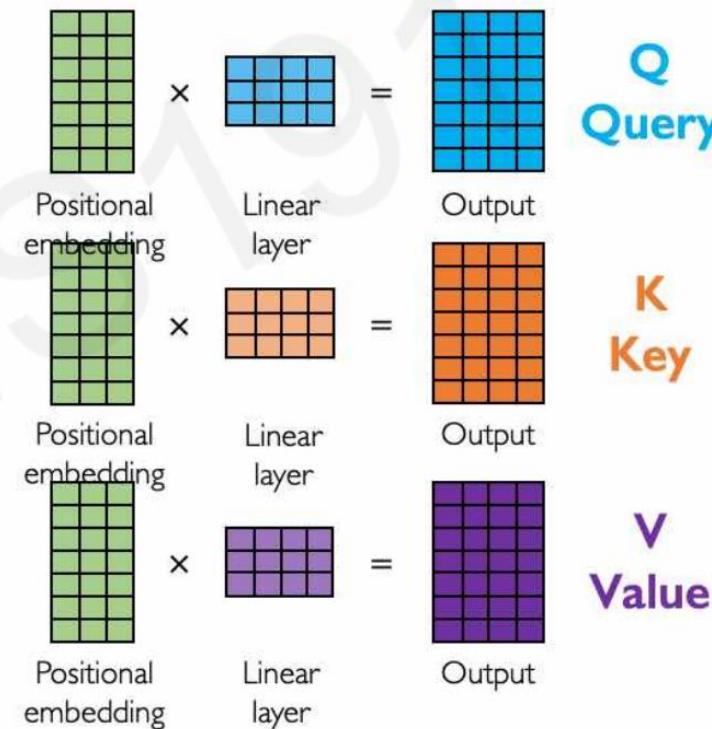
- Data is fed in all at once!
 - Need to encode position information to understand order



Learning Self-Attention with Neural Networks

- ❖ Goal: identify and attend to most important features in input!!

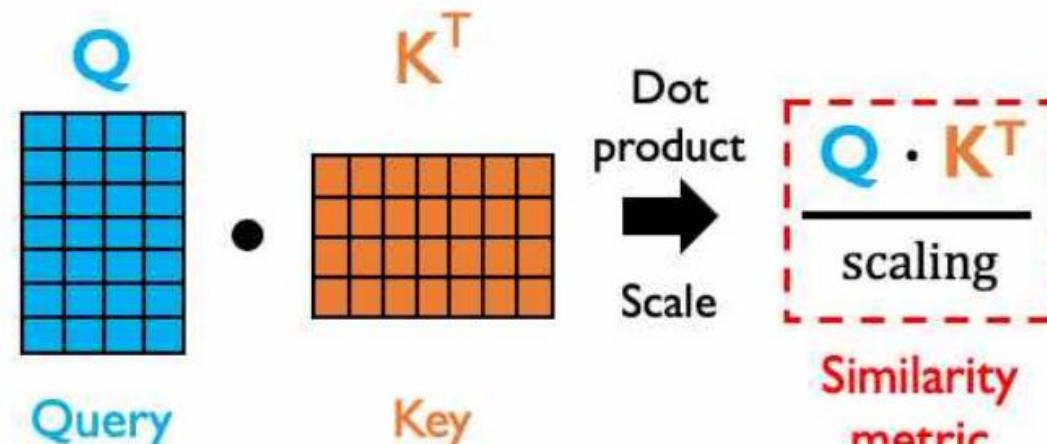
- ✓ Encoding position information
- ✓ Extract query(Q), key(K), value(V) for search



Learning Self-Attention with Neural Networks

- ❖ Goal: identify and attend to most important features in input!!

- ✓ Encoding position information
- ✓ Extract query(Q), key(K), value(V) for search
- ✓ Compute Attention weighting
 - Attention score: compute pairwise similarity between each query(Q) and key(K)

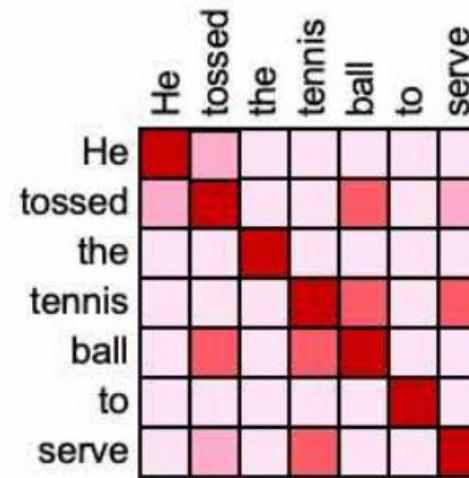


Also known as the "cosine similarity"

Learning Self-Attention with Neural Networks

- ❖ Goal: identify and attend to most important features in input!!

- ✓ Encoding position information
- ✓ Extract query(Q), key(K), value(V) for search
- ✓ Compute Attention weighting: where to attend to!
 - How similar is the key to the query?



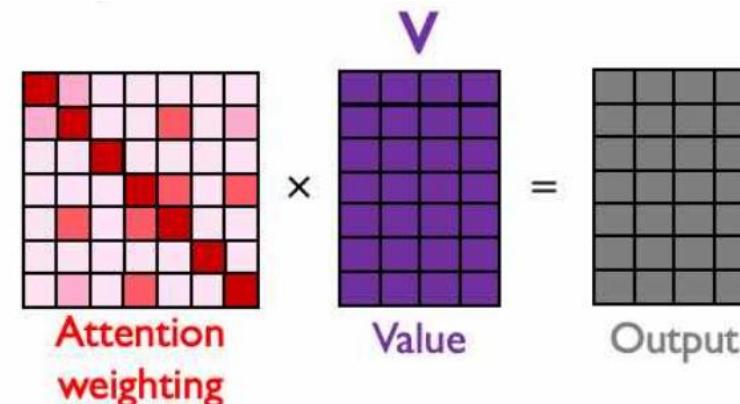
$$\text{softmax} \left(\frac{Q \cdot K^T}{\text{scaling}} \right)$$

Attention weighting

Learning Self-Attention with Neural Networks

- ❖ Goal: identify and attend to most important features in input!!

- ✓ Encoding position information
- ✓ Extract query(Q), key(K), value(V) for search
- ✓ Compute Attention weighting
- ✓ Extract features with high attention
 - Self-attend to extract features



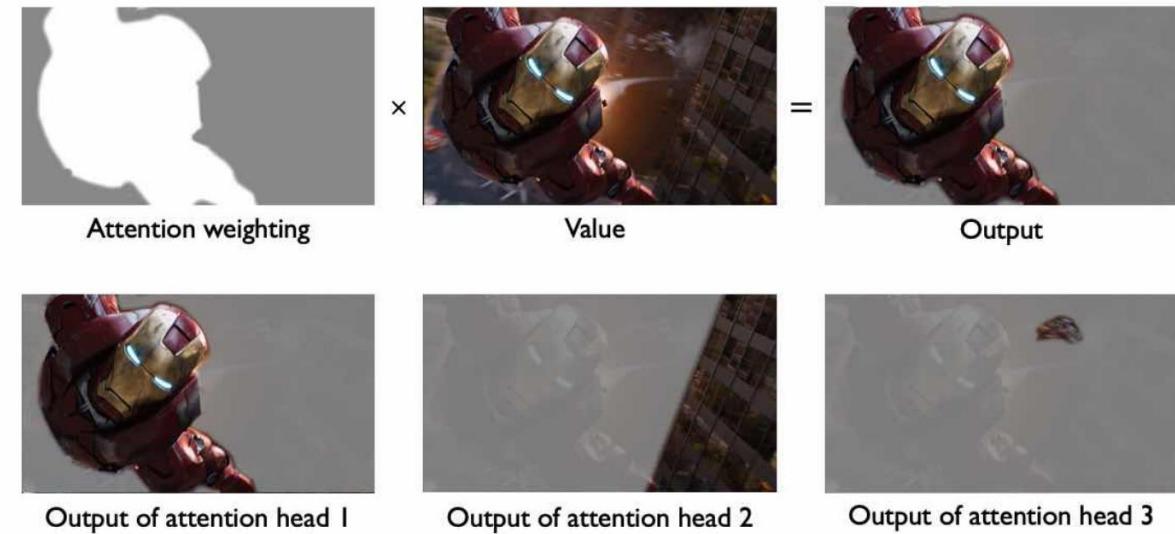
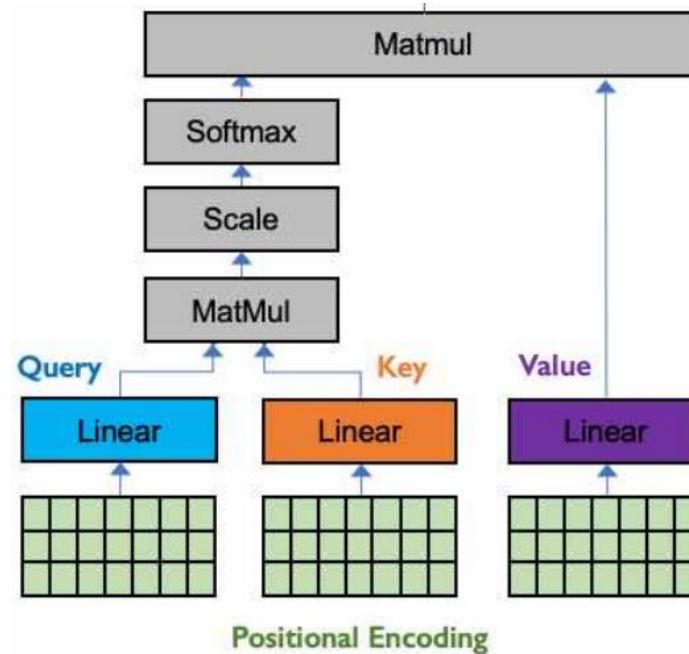
$$\text{softmax} \left(\frac{Q \cdot K^T}{\text{scaling}} \right) \cdot V = A(Q, K, V)$$

Learning Self-Attention with Neural Networks

❖ Goal: identify and attend to most important features in input!!

- ✓ Encoding position information
- ✓ Extract query(Q), key(K), value(V) for search
- ✓ Compute Attention weighting
- ✓ Extract features with high attention

These operations form a self-attention head
that can plug into a larger network.
Each head attends to a different part of input.

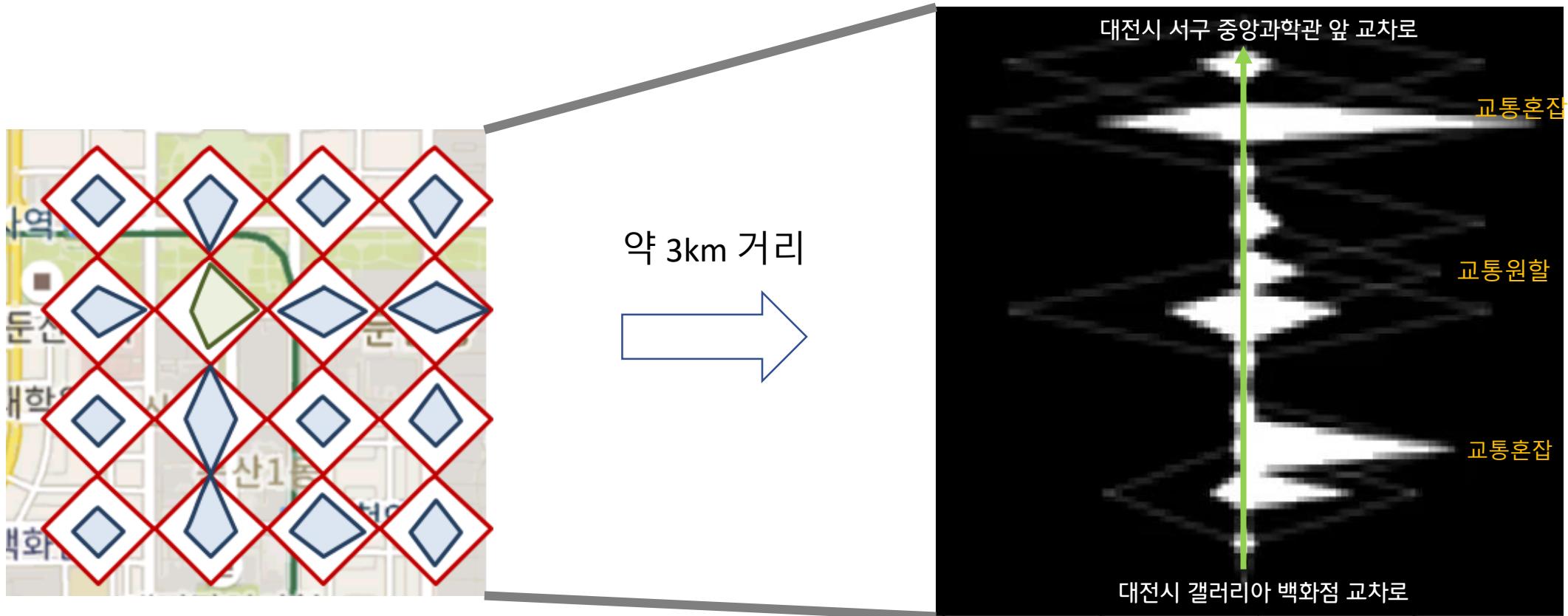


3-5

교통 프로젝트 소개

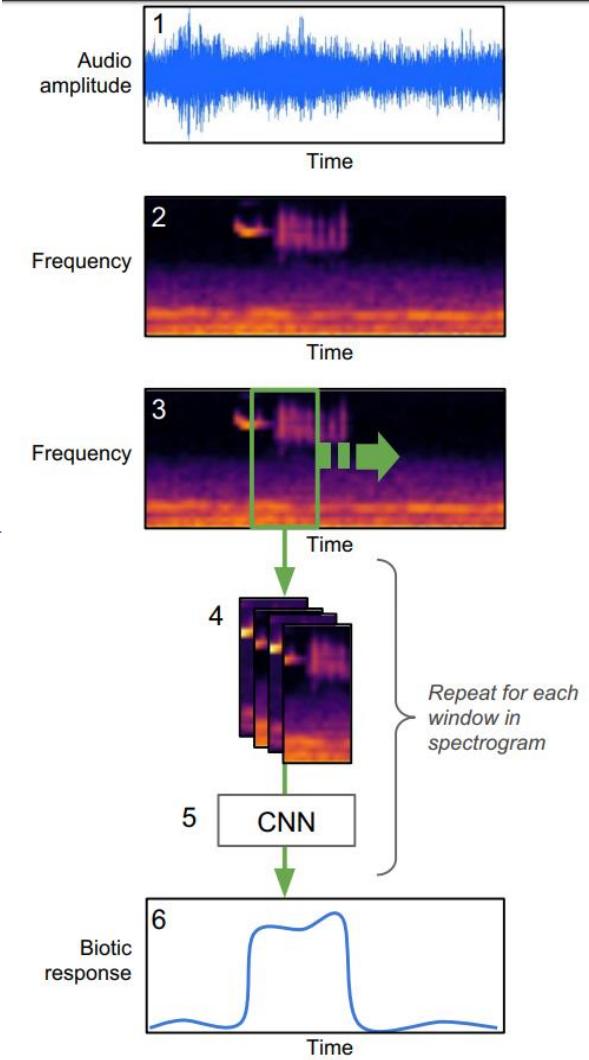
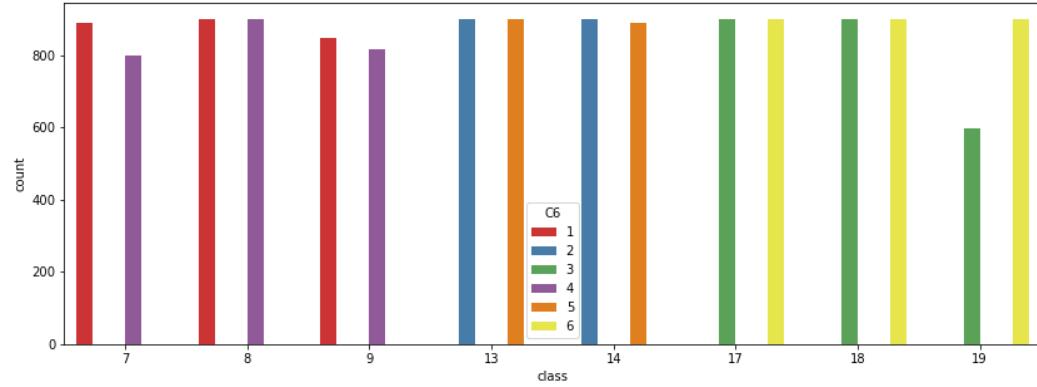
- ❖ RSE 교통 데이터를 직접 CNN을 적용하기 어렵다.

- ✓ RSE를 이미지로 변환하는 전략이 필요함
 - 대전시 대덕대로 RSE 데이터



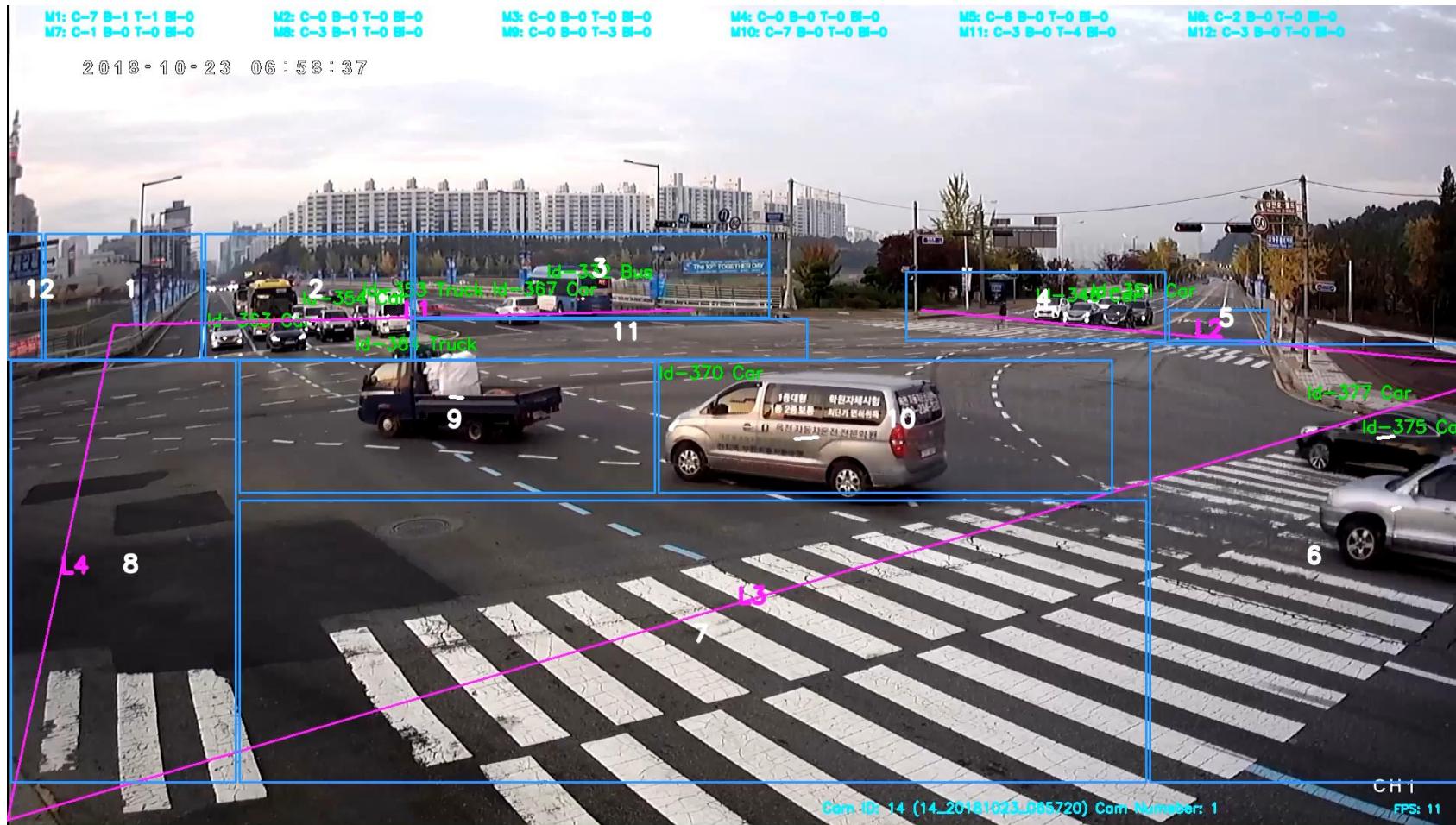
교통 데이터 활용한 AI 연구 소개 : CNN 적용

소음 빅데이터 자체 수집 : DJ_TrafficSound14K (대덕대로)



YOLO 적용: CCTV 영상에서 차량 탐색, 카운트, 추적 기술

- ❖ 세계 최고 AI 컨퍼런스인 CVPR20의 'AI City Challenge' 경진대회 참가 및 (7위, KISTI)

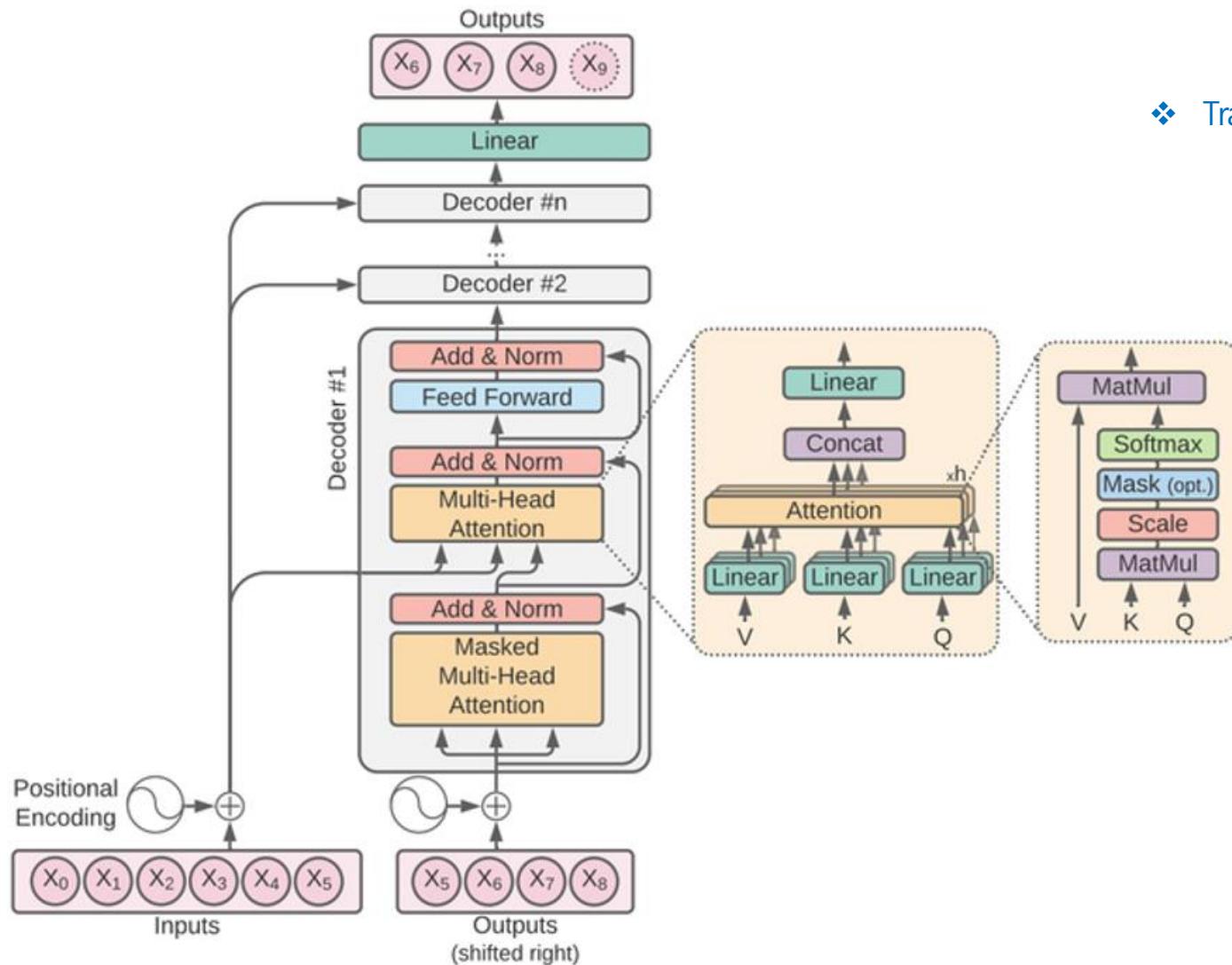


대상 도로 : 대학로 (유성구청네거리-구성삼거리)

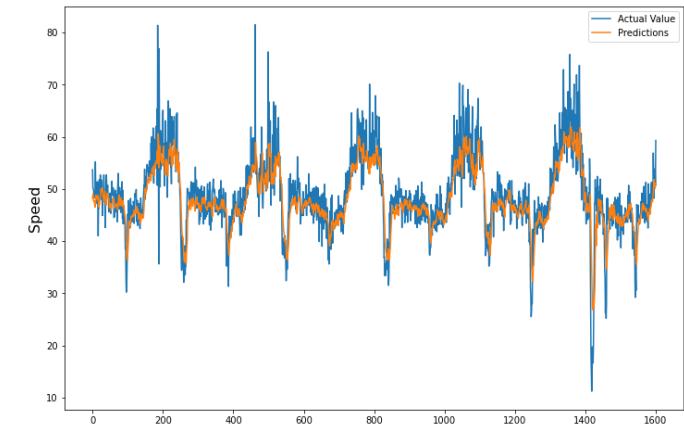
- ❖ 약 1.4km 왕복 4차로, RSE 3개, 영상검지기 5개(현재 3개, 향후 2개)



멀티 헤드 어텐션 메커니즘



❖ TransformerVDS: Transformer+TimeEmbedding



- ❖ 교통 데이터 VDS 분석 및 분류를 위한 라벨링
 - ✓ 교통량과 점유률 관계 및 상관계수
- ❖ DNN, RNN, LSTM, GRU로 교통 예측
 - ✓ RNN, LSTM, GRU 모델 및 가중치 공유
 - ✓ 파라미터 개수 구하기
 - ✓ Many-to-One, Many-to-Many 예측 문제 옵션 조정하기
- ❖ 1차원 CNN 모델로 교통 예측
 - ✓ 가중치 공유 알고리즘
 - ✓ 입력 데이터 형태(숫자, 텍스트, 이미지)에 따른 입력 변화
- ❖ 멀티헤드 어텐션과 트랜스포머 모델로 교통 예측하기
 - ✓ RNN, LSTM의 한계
 - ✓ Encoder-Decoder 모델
 - ✓ 쿼리, 키, 벨류 (Q,V,K) 기반 어텐션 메커니즘 이해
- ❖ 프로젝트
 - ✓ Multi-Step Ahead Prediction model에 적용하기

TRUST **KISTI**

Korea Institute of Science
and Technology Information

