KiSTi 한국과학기술정보연구원
Korea Institute of Science and Technology Information

2022

스마트교통 빅데이터 분석

시계열 데이터 분석을 위한 RNN 모델 이해

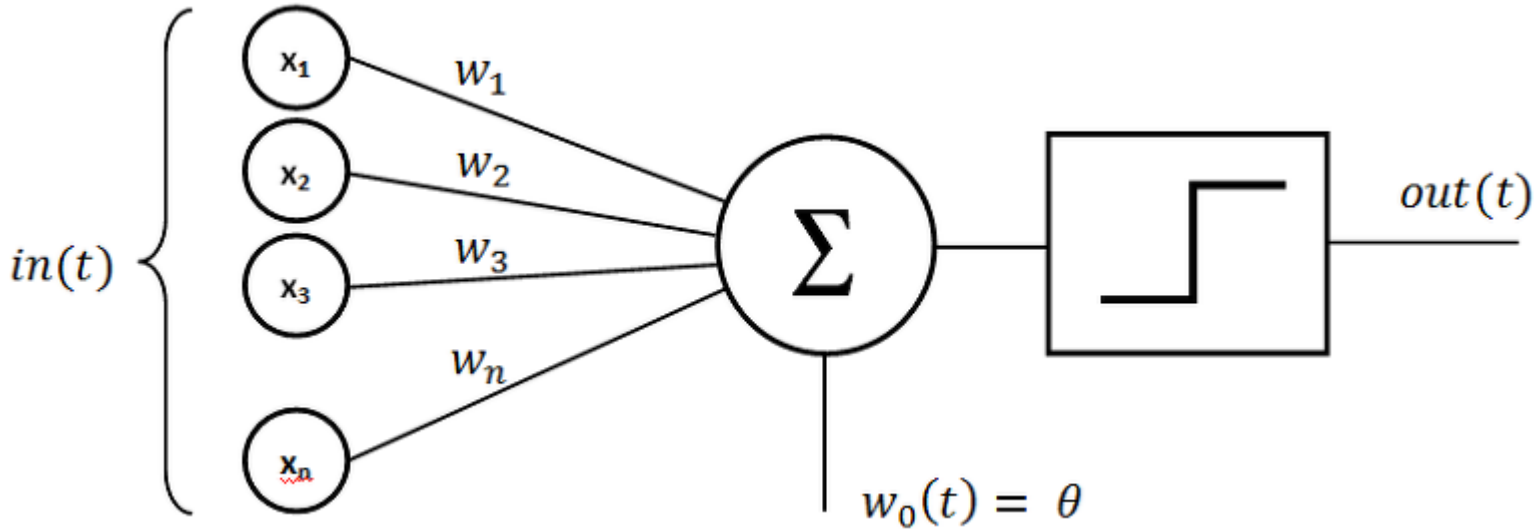
2022.6.22.

이 홍 석 (hsyi@kisti.re.kr)

Day2-Lec1
활성함수

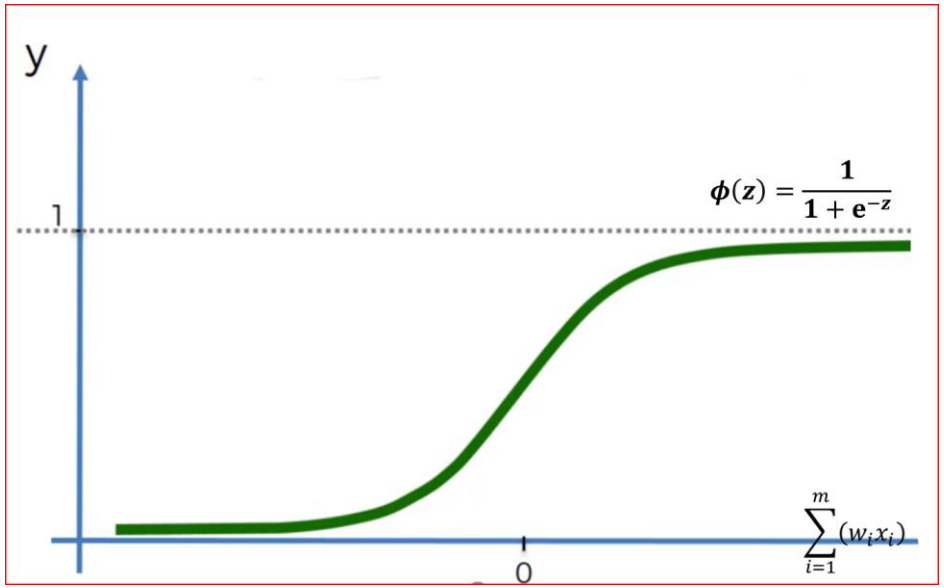
왜 딥러닝 신경망에서 활성화 기능이 필요한가요?

- **활성화 기능은 뉴런을 활성화할 필요가 있는지 없는지를 결정하는데 도움을 준다.**
  - ✓만약 필요에 의해 해당 뉴런을 활성화하면 그 신호의 강도는 어떻게 해야 될까요?
  - ✓뉴런이 신경망을 통해 정보를 처리하고 전달하는 메커니즘



Sigmoid 함수

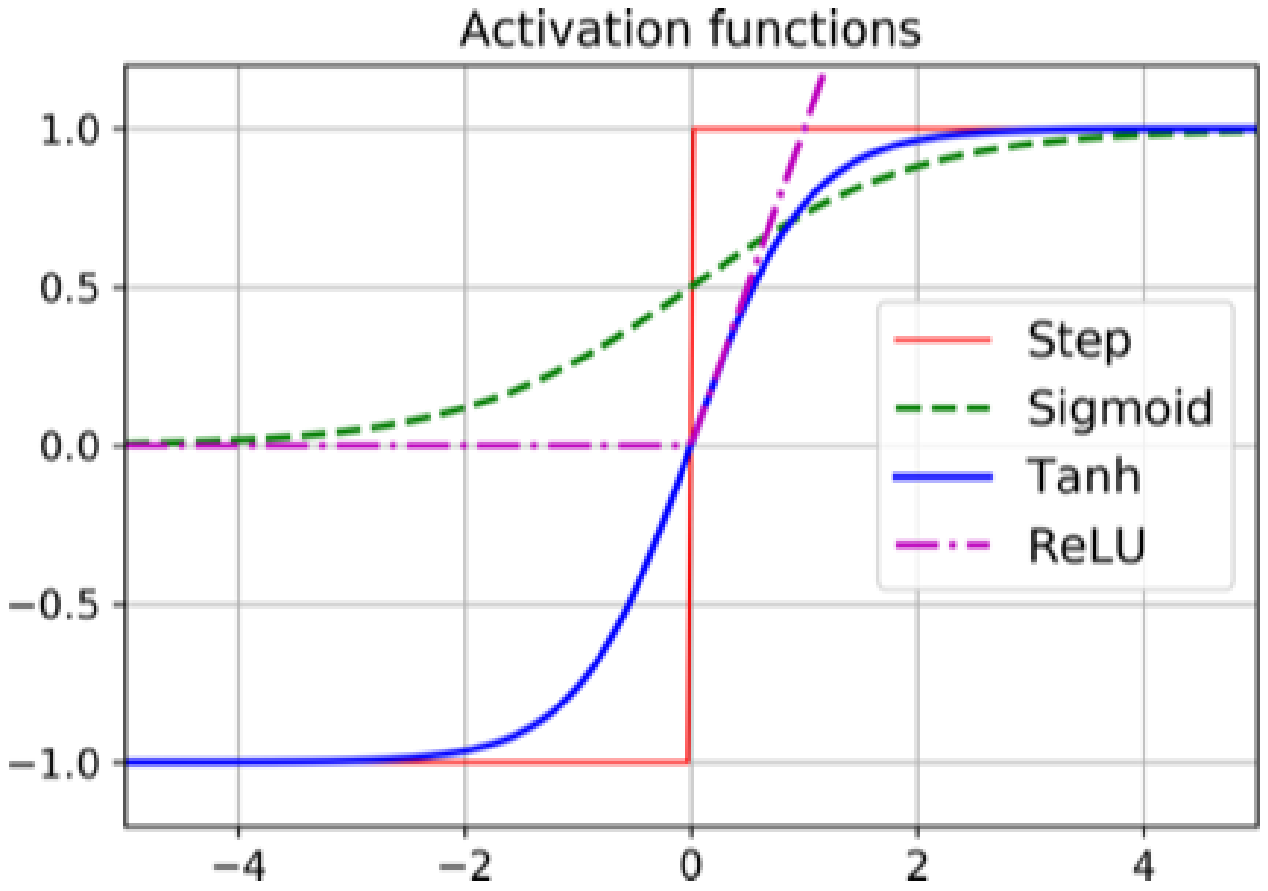
- 역전파 알고리즘 개발시 등장
  - ✓Step 함수와 달리 Sigmoid는 미분 가능하다.
  - ✓역전파 알고리즘이 가능함.
- 이중분류(Binary Classifiction)에 적용 가능
  - ✓하지만 Multi-Class 분류는 불가능하다.



Softmax

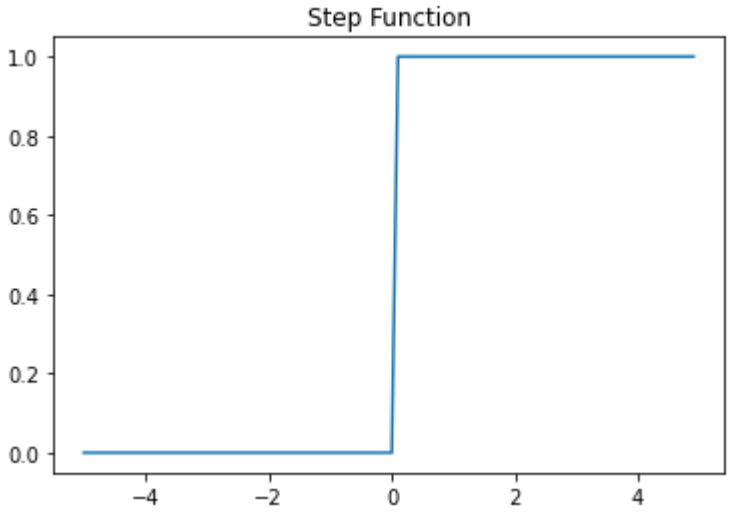
$$\phi(z) = \frac{e^i}{\sum_{j=0}^{k} e^j} \quad \text{where } i=0,1,\ldots.k$$

ReLU (Rectified Linear Unit)



활성함수의 종류 및 Step Function

In [5]:
```
import numpy as np
import matplotlib.pyplot as plt

def step(x):
    return np.array(x > 0, dtype=np.int)

# -5.0부터 5.0까지 0.1 간격 생성
x = np.arange(-5.0, 5.0, 0.1)
y = step(x)

plt.title('Step Function')
plt.plot(x,y)
```

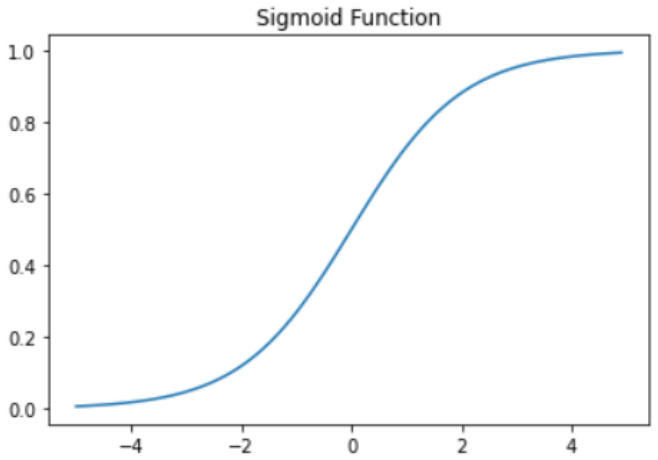


Sigmoid and tanh(x)

In [2]:
```
# 시그모이드 함수 그래프를 그리는 코드
def sigmoid(x):
    return 1/(1+np.exp(-x))
```

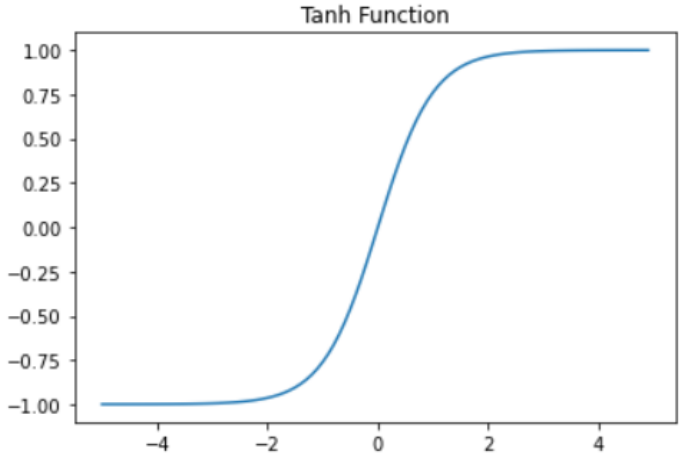
In [3]:
```
y = sigmoid(x)
plt.plot(x, y)
plt.title('Sigmoid Function')
```

Out[3]: Text(0.5, 1.0, 'Sigmoid Function')



In [4]:
```
# tanh(x) 그리기
y = np.tanh(x)
plt.plot(x, y)
plt.title('Tanh Function')
```

Out[4]: Text(0.5, 1.0, 'Tanh Function')

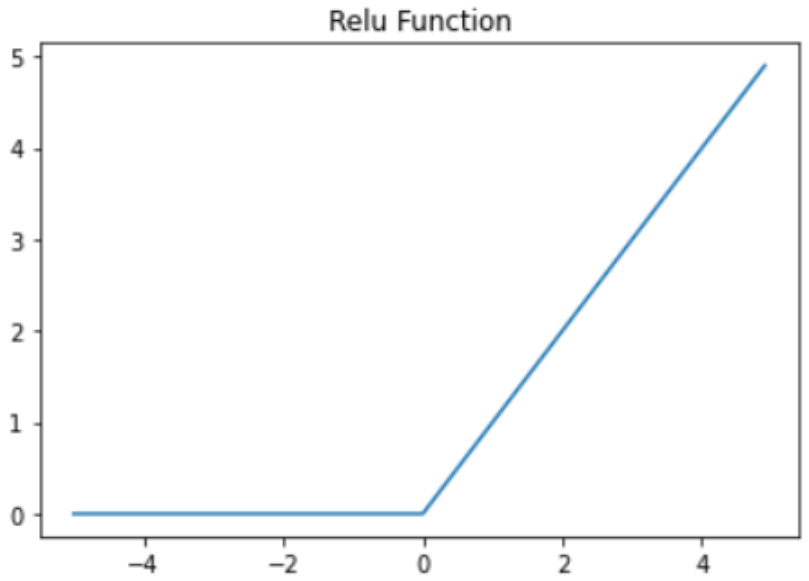


ReLU and Softmax

In [5]:
```
def relu(x):
    return np.maximum(0, x)
```

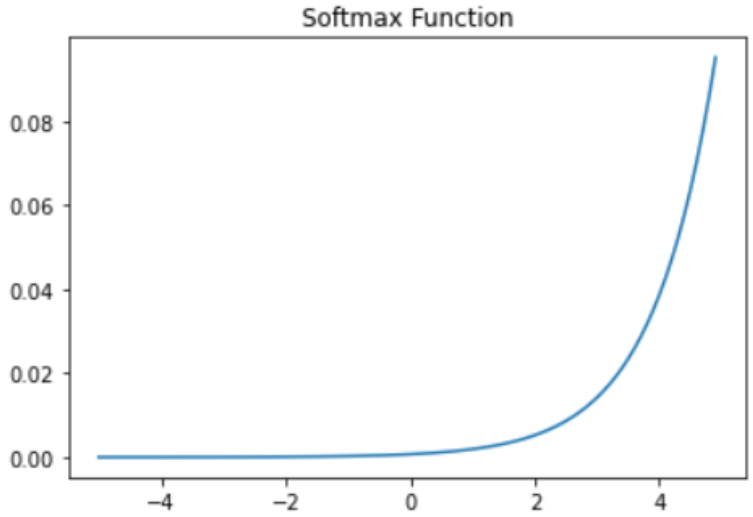
In [6]:
```
y = relu(x)
plt.plot(x, y)
plt.title('Relu Function')
```

Out[6]: Text(0.5, 1.0, 'Relu Function')



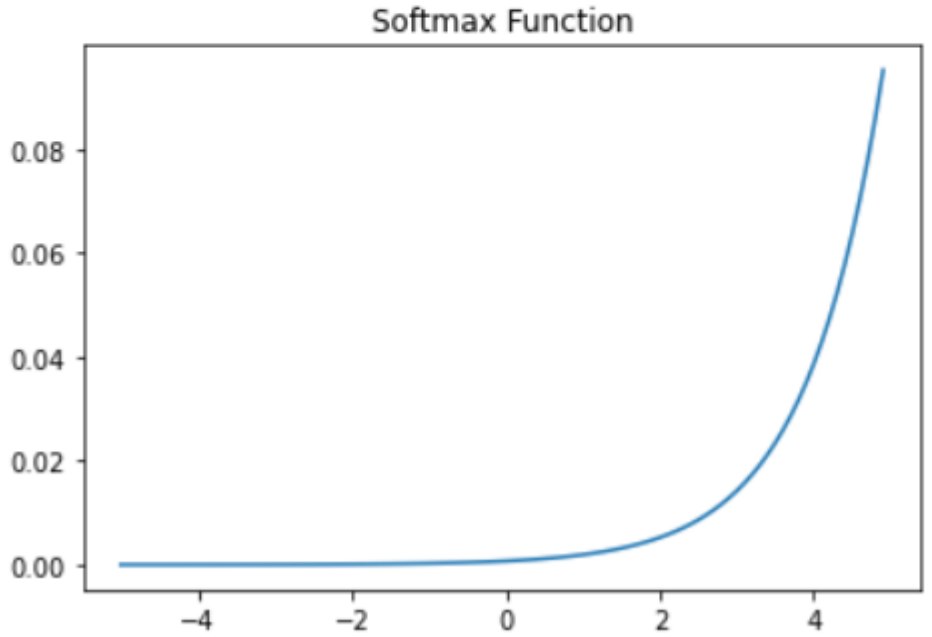
$$p_i = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}} \quad for\ i = 1, 2, ...k$$

In [7]:
```
y = np.exp(x) / np.sum(np.exp(x))
plt.plot(x, y)
plt.title('Softmax Function')
plt.show()
```



Leaky_ReLu

In [8]:
```
def leaky_relu(x):
    return np.maximum(a*x, x)
plt.plot(x, y)
plt.title('Softmax Function')
plt.show()
```



Day2-Lec2: 실습
텐서플로우를 이용한 선형회귀

회귀(Regression) 소개(1)

❖ 회귀의 역사

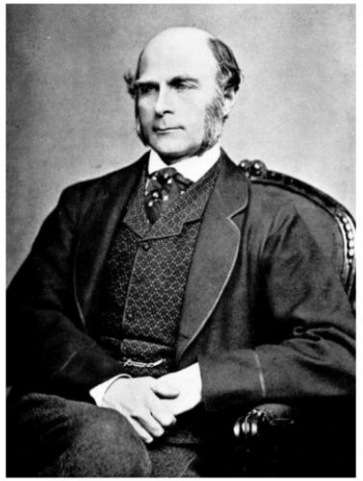
✓영국의 통계학자 갈톤(Galton)의 유전적 특성중에 부모와 자식의 키 관계
✓"사람의 키는 평균 키로 회귀(Regression)하려는 경향을 가진다는 자연의 법칙이 있다"
✓회귀 분석은 데이터 값이 평균과 같은 일정한 값으로 돌아가려는 경향을 이용한 통계학 기법

❖ 지도학습은 2가지 유형으로 나눔

✓회귀는 연속적인 숫자 값
✓분류는 예측값이 카테고리와 같은 이산형 클래스 값

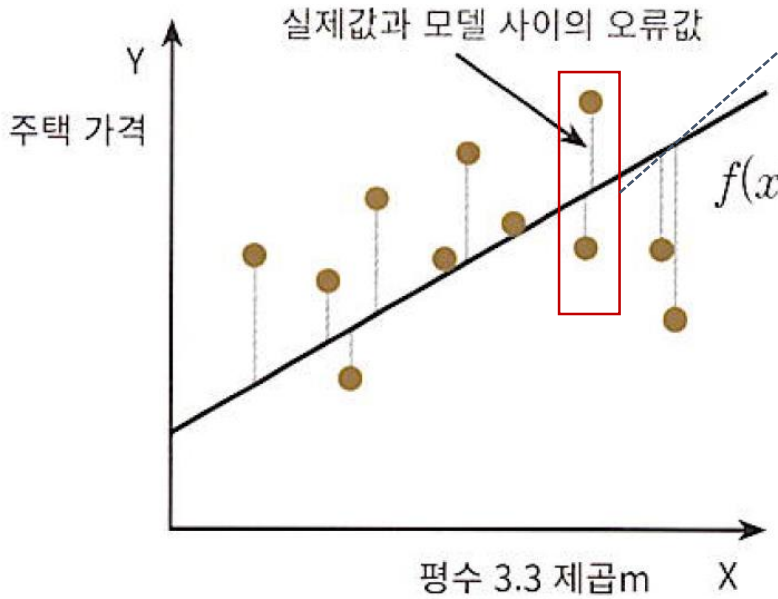
❖ 선형회귀

✓실제 값과 예측 값의 차이를 최소화하는 직선형 회귀선을 최적화하는 방식
✓단순 선형회귀는 1개의 독립변수, 1개의 종속변수.
✓예) 주택 가격이 주택의 크기로만 결정된다고 해보면,



Sir Francis Galton (1822 ~ 1911)

❖ 단순 선형회귀를 통한 회귀의 이해 : 단순 선형회귀는 1개의 독립변수, 1개의 종속변수



오차는 양수(+) 혹은 음수(-) 가능하므로, 절대값을 취하거나(MAE), 오차 값의 제곱 (MSE)를 사용.
보통 RSS(Residual Sum of Square)~(오차)의 제곱

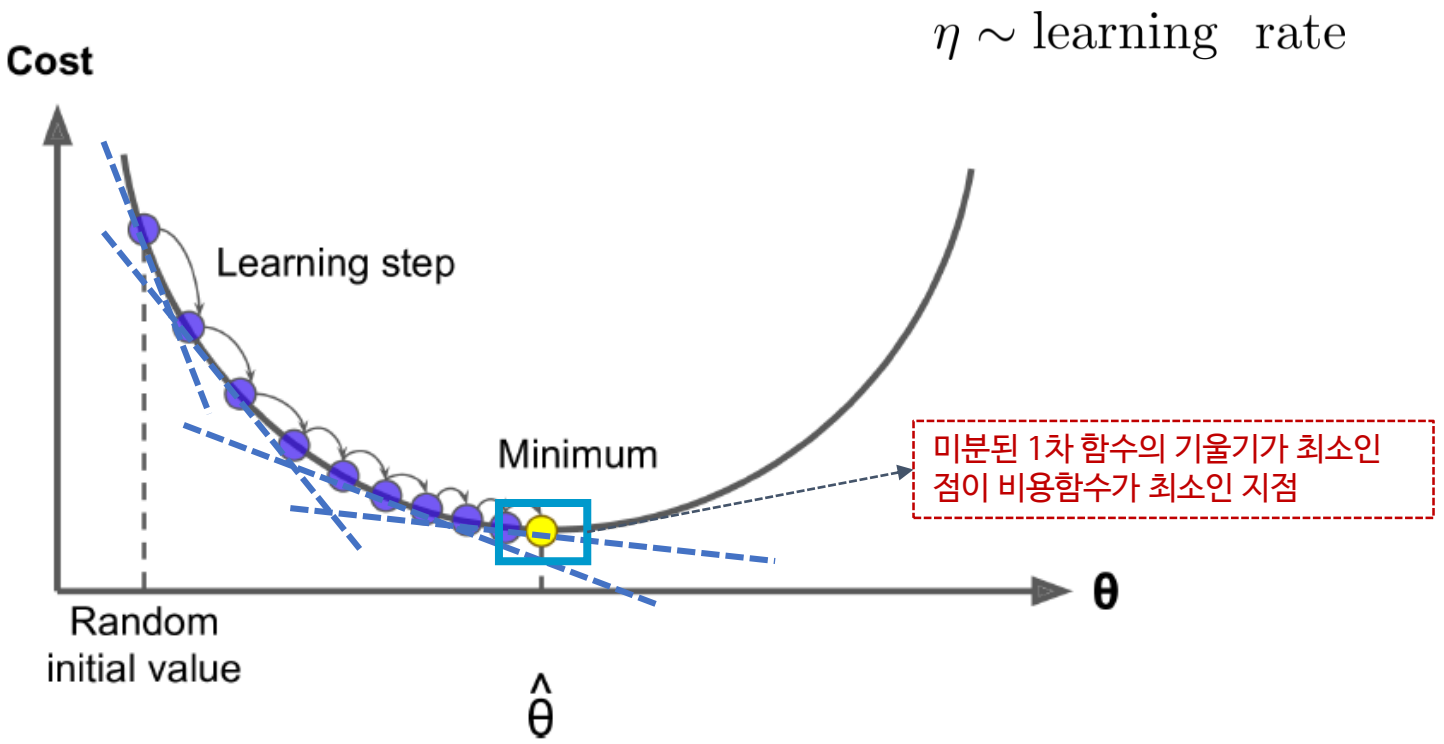
$$f(x) = w_0 + w_1 * x$$

RSS = (#1 주택 가격 – $(w_0+w_1$ #1 주택크기$)^2$
+ (#2 주택 가격 – $(w_0+w_1$ #2 주택크기$)^2$
+ (#3 주택 가격 – $(w_0+w_1$ #3 주택크기$)^2$
+ … (모든 학습 데이터에 대해 RSS 수행)

회귀에서 RSS는 비용(cost)이며, 혹은 비용함수, 혹은 손실(loss) 함수라고도 함

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

❖ 경사 하강법 (Gradient Descent)



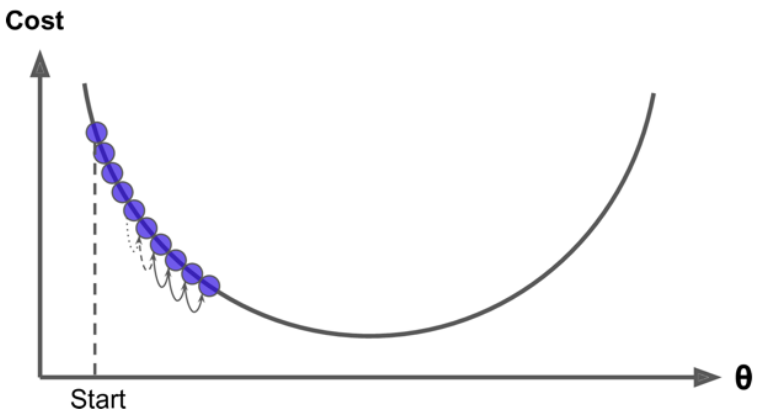
미분된 1차 함수의 기울기가 최소인 점이 비용함수가 최소인 지점

비용 최소화 _ 경사 하강법(2)

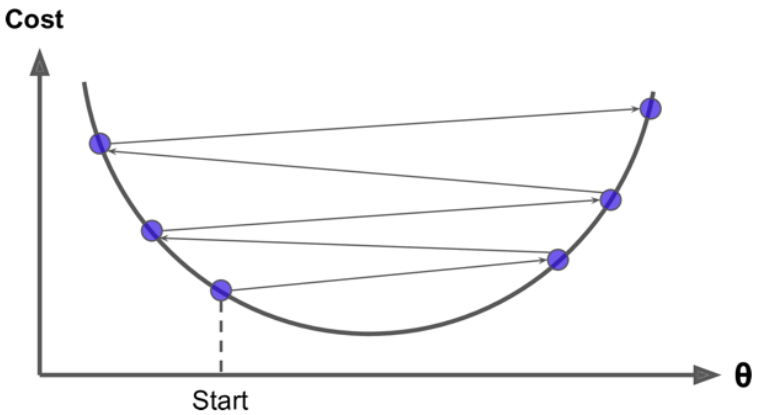
❖ 학습률

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

학습률 도입 $\eta \sim$ learning rate



a) too small

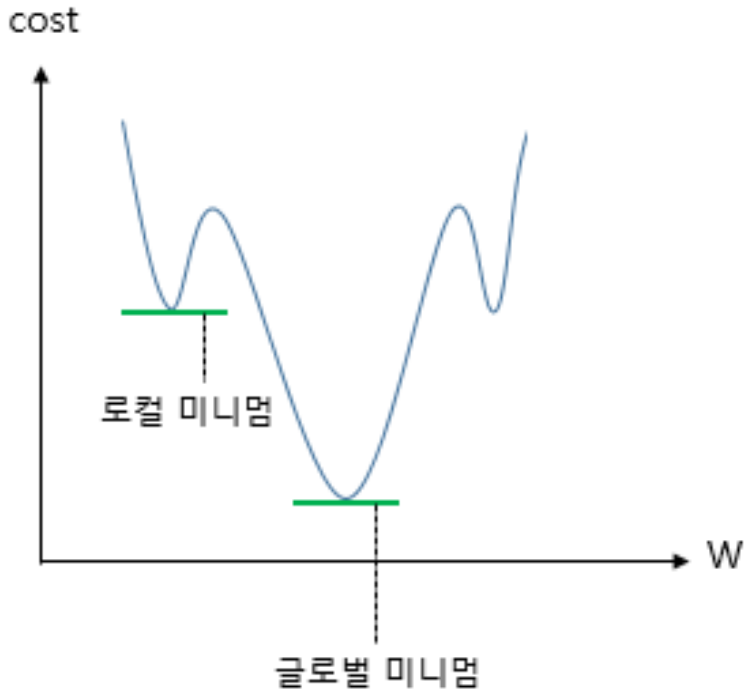


a) too big

로지스틱 비용 함수(Cost function)

❖ 로컬미니멈의 위험

- ✓로지스틱 회귀 또한 경사 하강법을 사용하여 가중치를 찾아내지만, 비용 함수로는 평균 제곱 오차를 사용하지 않습니다.
- ✓이유는 시그모이드 함수에 비용 함수를 평균 제곱 오차로 하여 그래프를 그리면 다음과 비슷한 형태가 되기 때문입니다.

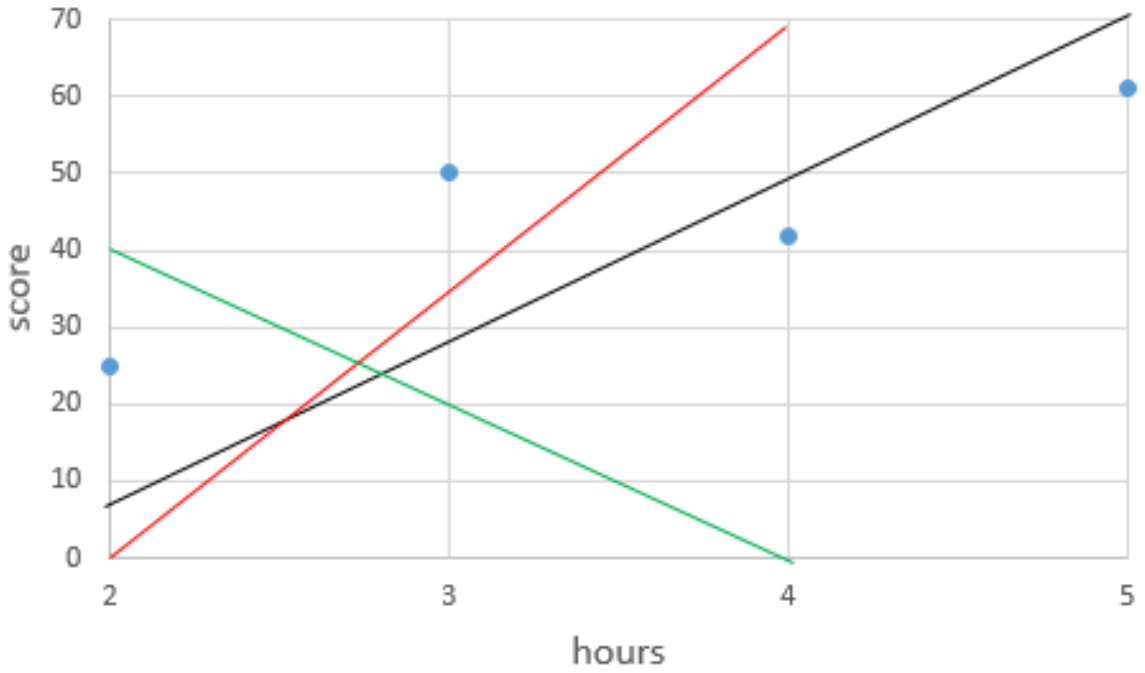


가설(Hypothesis) 세우기

- 어떤 학생의 공부 시간에 따라서 얻은 점수 데이터
- 예측을 위한 선형 가설을 세우자

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

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

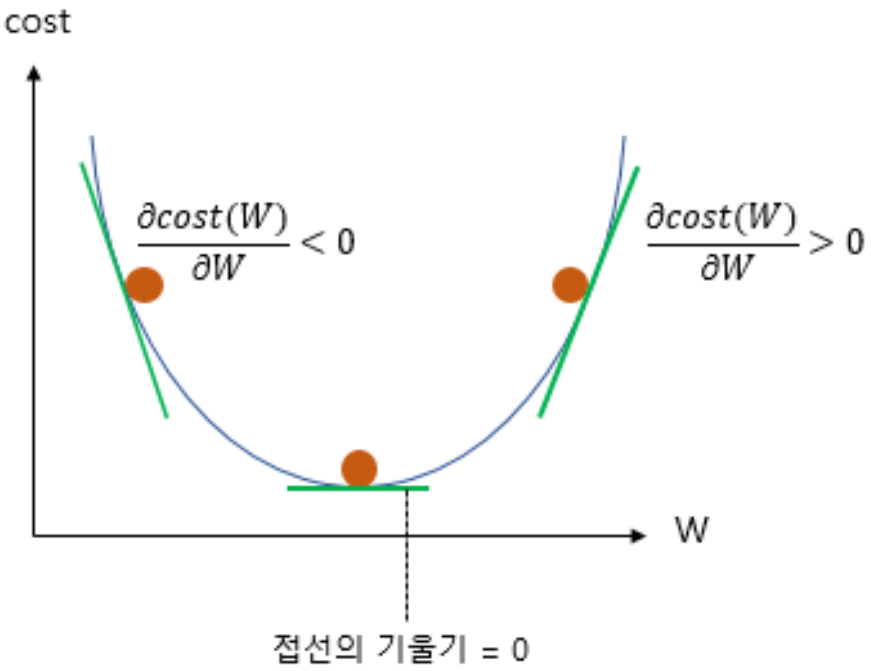


❖ 옵티마이저(Optimizer) 또는 최적화 알고리즘

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

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

$$W, b \rightarrow minimize\ cost(W, b)$$



(lab) 케라스로 구현하는 선형 회귀

- 케라스로 모델을 만드는 기본적인 형식

- model = keras.models.Sequential()
  - ✓Seauential로 모델을 이라는 이름을 만들고
- model.add(keras.layers.Dense(1, input_dim=1))
  - ✓add를 통해서 필요한 사항을 추가해 나간다.
  - ✓첫 인자 1은 출력의 차원
  - ✓두 번째 인자 input_dim은 입력의 차원을 정의

케라스로 구현하는 선형 회귀

KISTI SINCE 1962

[2]
```
import tensorflow as tf
import numpy as np
print(tf.__version__)
```

```
2.8.0
```

[3]
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras import optimizers
```

[4]
```
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])
# 각 공부하는 시간에 맵핑되는 성적
```

케라스로 구현하는 선형 회귀

```
[5] model = Sequential()

    model.add(Dense(1, input_dim=1, activation='linear'))
```

```
[6] # sgd는 경사 하강법을 의미.
    #학습률(learning rate, lr)은 0.01.
    sgd = optimizers.SGD(lr=0.01)
```

```
/usr/local/lib/python3.7/dist-packages/keras/optimizer_v2/gradient_descent.py:102:
  super(SGD, self).__init__(name, **kwargs)
```

케라스로 구현하는 선형 회귀

회귀에서

- 손실함수는 오차 MSE을 주로 사용한다.

```
[7] # 손실 함수(Loss function)은 평균제곱오차 mse를 사용합니다.
    model.compile(optimizer=sgd ,loss='mse',metrics=['mse'])
```

```
[8] # 주어진 X와 y데이터에 대해서 오차를 최소화하는 작업을 300번 시도합니다.
    history=model.fit(X,y, batch_size=1, epochs=30, shuffle=False)
```

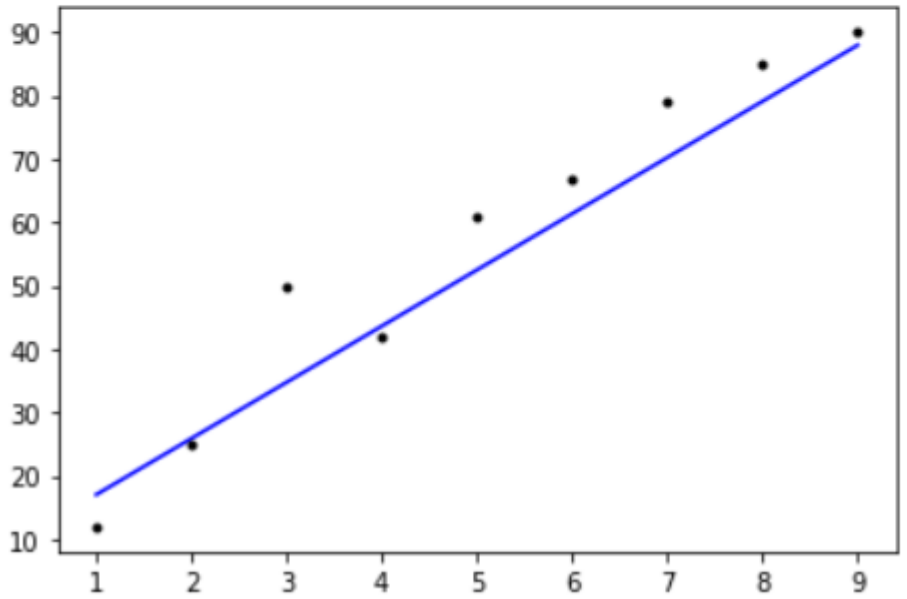
케라스로 구현하는 선형 회귀

```
%matplotlib inline
```

```
import matplotlib.pyplot as plt
```

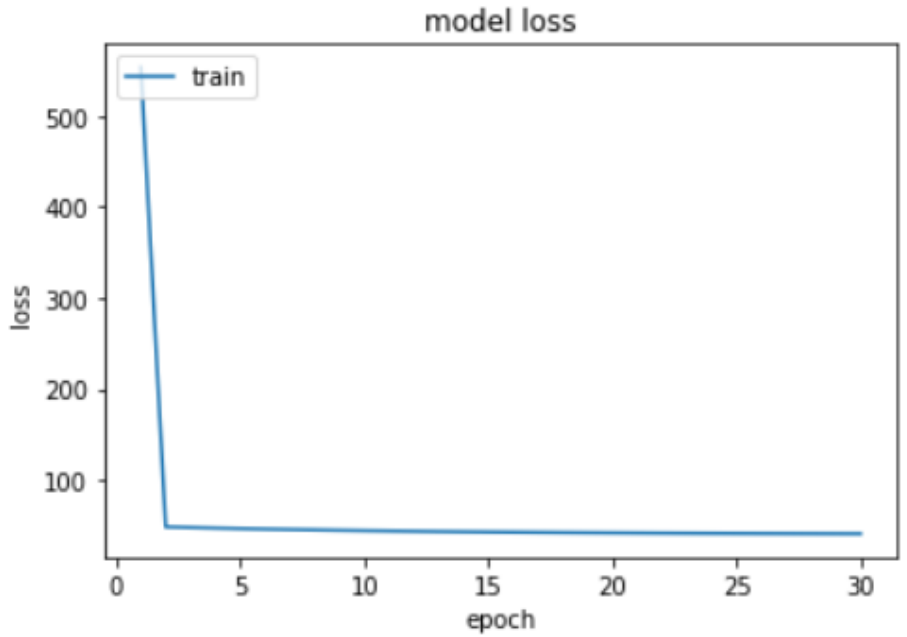
```
plt.plot(X, model.predict(X), 'b', X,y, 'k.')
```

```
[<matplotlib.lines.Line2D at 0x7fda5a760a90>,
 <matplotlib.lines.Line2D at 0x7fdad0150490>]
```



케라스로 구현하는 선형 회귀

```
epochs = range(1, len(history.history['mse']) + 1)
plt.plot(epochs, history.history['loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train'], loc='upper left')
plt.show()
```

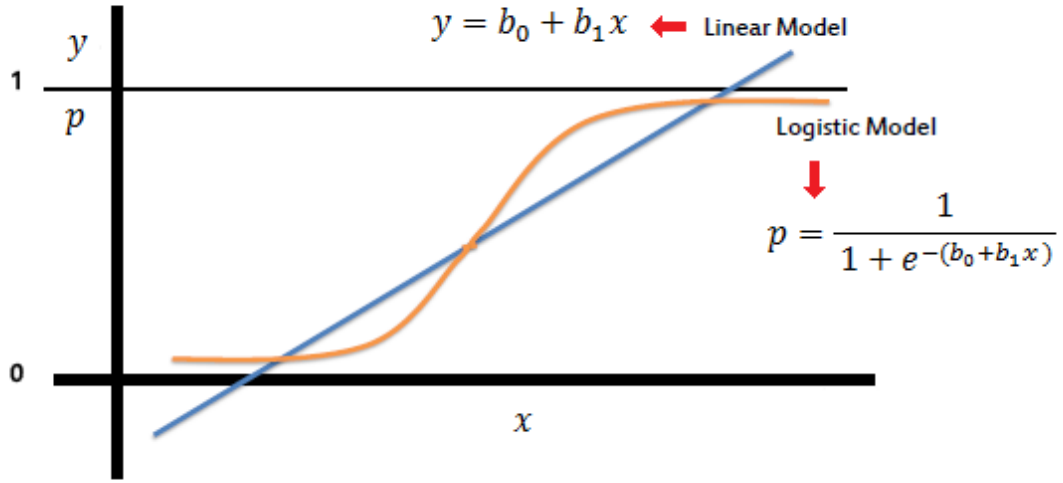


Day2-Lec3: 실습
텐서플로우를 이용한 비선형회귀

❖ 로지스틱 회귀 개요

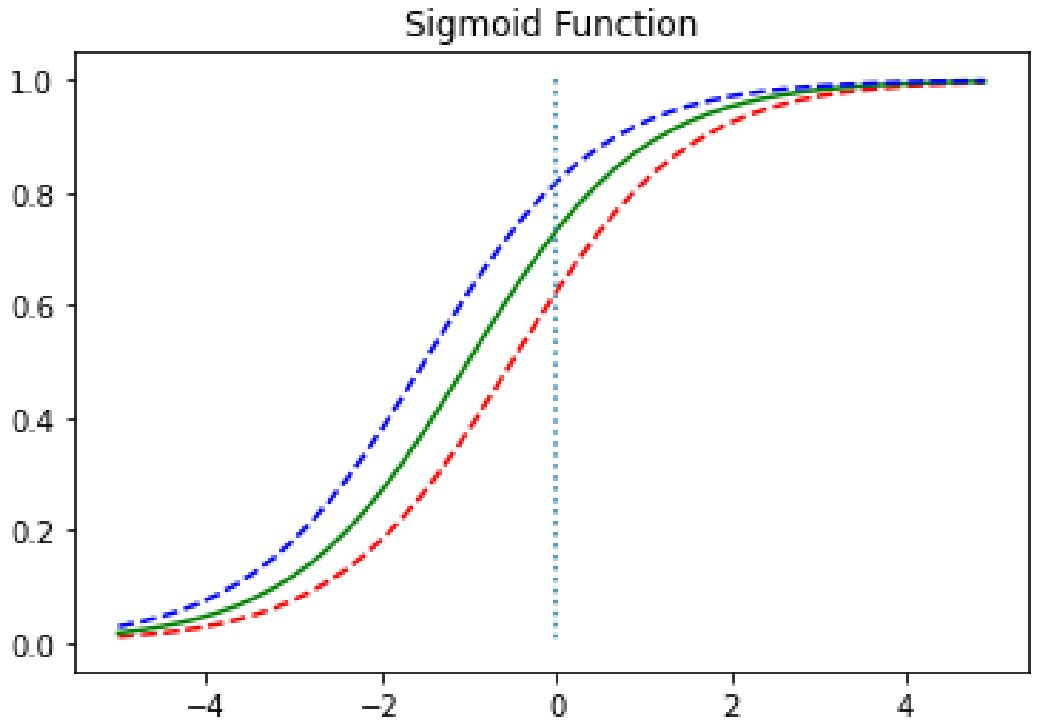
✓둘 중 하나를 결정하는 문제를 이진 분류(Binary Classification)라고 합니다.

- 이런 문제를 풀기 위한 알고리즘으로 로지스틱 회귀(Logistic Regression)가 있다.



시그모이드 함수(Sigmoid function)

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




```
# x + 0.5

# x + 1.5
```

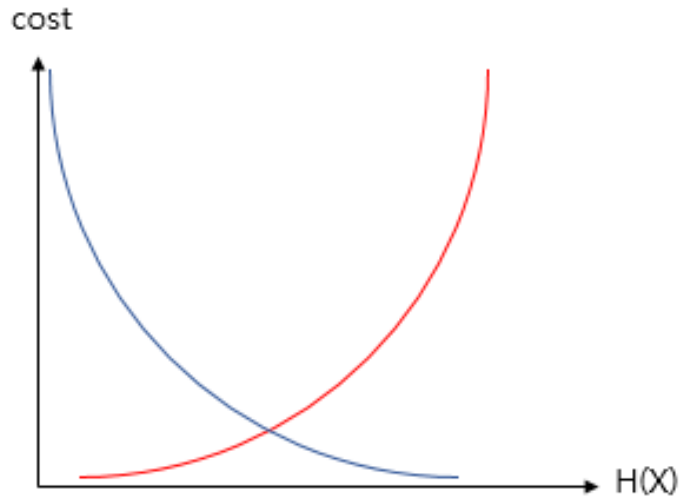
e=2.718281… 자연 상수

로지스틱 비용 함수(Cost function)

❖ 목적 함수(objective function)

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

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



❖ 크로스 엔트로피 함수 (Cross Entropy)

✓로지스틱 회귀에서 찾아낸 비용함수를 말한다

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

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

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

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

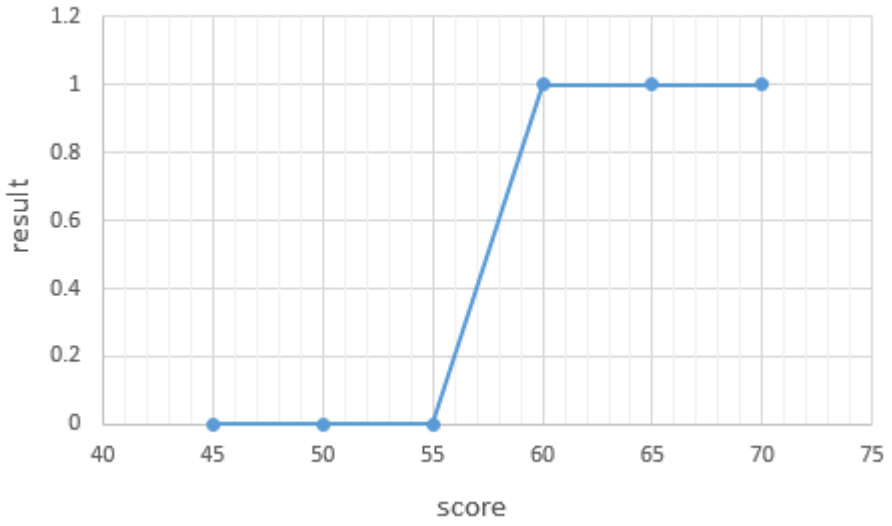
이진 분류(Binary Classification)

❖ 예제: 학생들이 시험 성적에 따라서 합격, 불합격이 기재된 데이터가 있다고 가정

✓시험 성적이 x라면, 합불 결과는 y입니다.

✓이 시험의 커트라인은 공개되지 않았는데 이 데이터로부터 특정 점수를 얻었을 때의 합격, 불합격 여부를 판정

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



```
[1] import numpy as np
```

```
[2] from tensorflow.keras.models import Sequential
    from tensorflow.keras.layers import Dense
    from tensorflow.keras import optimizers
```

```
[3] 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])
```

```
[4] model=Sequential()
```

```
[5] model.add(Dense(1, input_dim=1, activation='sigmoid'))
```

```
[6] sgd=optimizers.SGD(lr=0.01)
```

```
/usr/local/lib/python3.7/dist-packages/keras/optimizer_v2/gradient_descent.py:102: UserWarning: The
  super(SGD, self).__init__(name, **kwargs)
```

- 회귀의 손실함수는 오차(MSE)를 사용해야 하지만, 로지스틱은 특별히 Accuracy를 자주 쓴다.

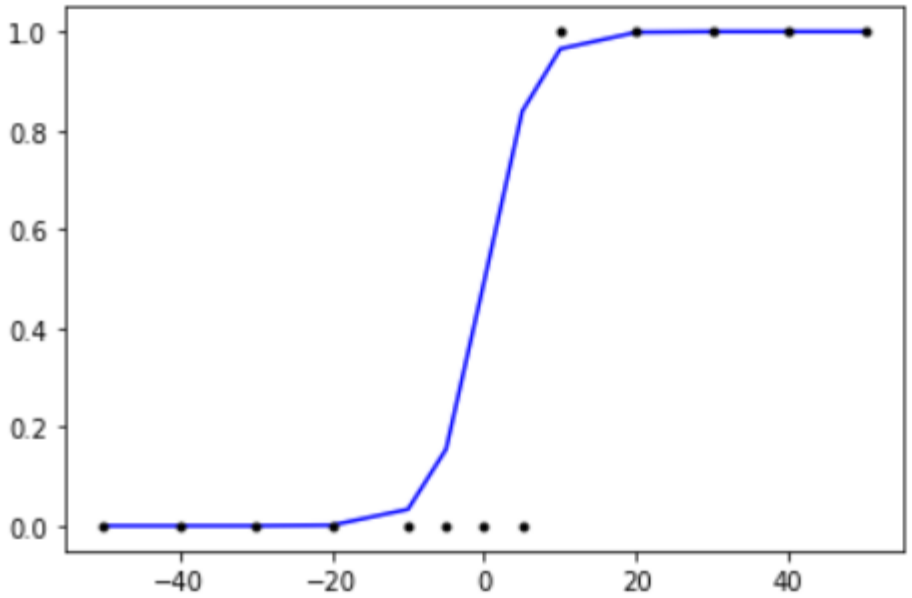
```
[7] model.compile(optimizer=sgd ,
                  loss='binary_crossentropy',metrics=['binary_accuracy'])
```

```
# 옵티마이저는 경사하강법 sgd를 사용합니다.
# 손실 함수(Loss function)는 binary_crossentropy를 사용합니다.
history = model.fit(X,y, epochs=20, shuffle=False)
```

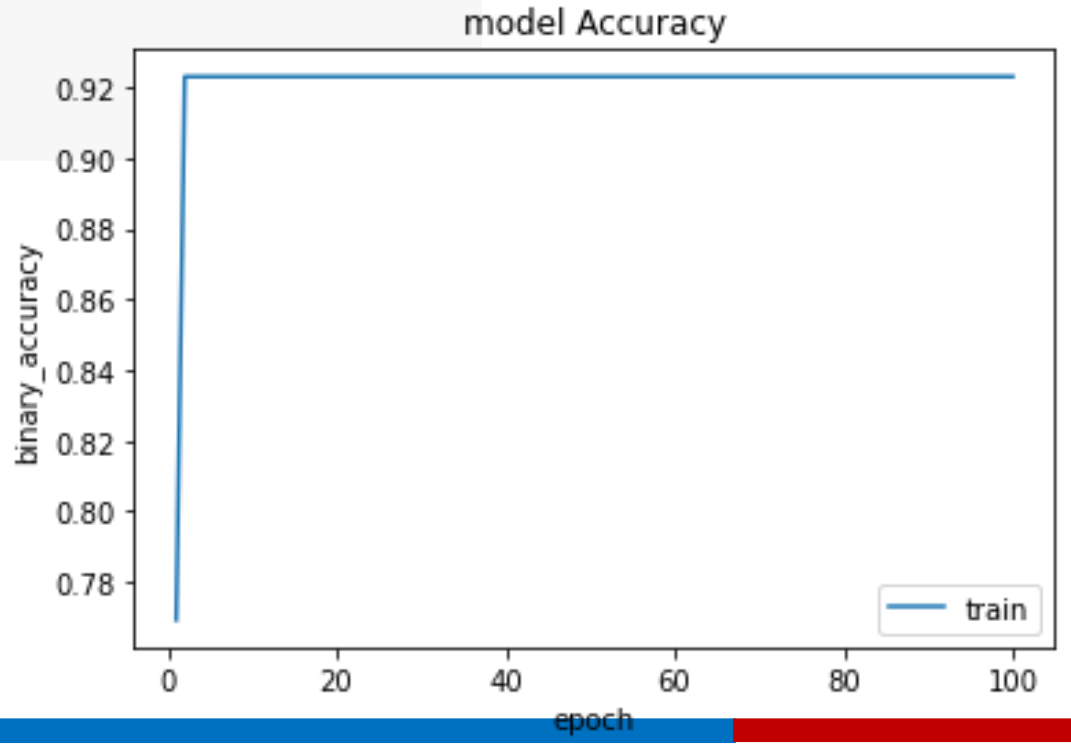
```
[9] import matplotlib.pyplot as plt
    %matplotlib inline
```

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

```
[<matplotlib.lines.Line2D at 0x7f1c7c723950>,
 <matplotlib.lines.Line2D at 0x7f1c7af7da10>]
```



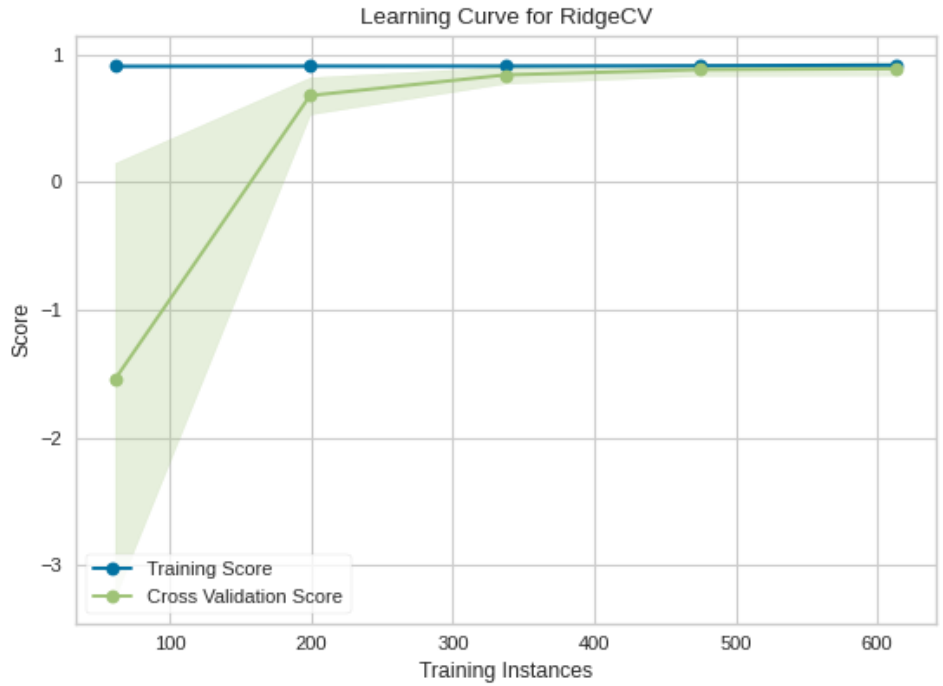
```
epochs = range(1,
               len(history.history['binary_accuracy']) + 1)
plt.plot(epochs, history.history['binary_accuracy'])
plt.title('model Accuracy')
plt.ylabel('binary_accuracy')
plt.xlabel('epoch')
plt.legend(['train'])
plt.show()
```



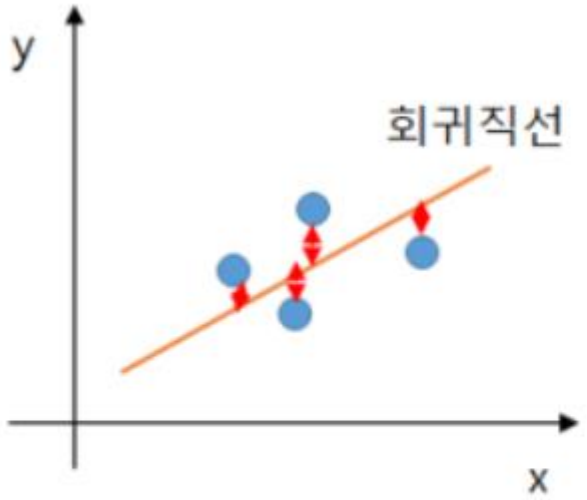
sklearn learning curve : Regression에서 R2 스코어 사용

- sklearn에서 learning_curve의 score는 높은 값이 좋은 성능으로 표현함.
  - ✓회귀에서 MSE 보다는 Negative MSE를 사용함
  - ✓R2도 사용함
- sklearn에서 Regression learning_curve의 score는 deafult로 R2를 사용함.

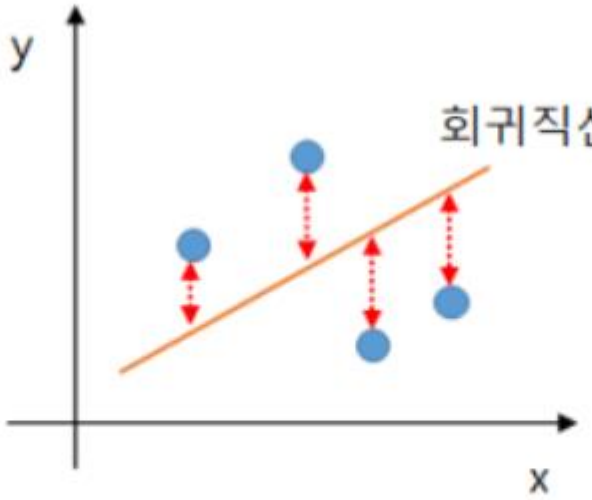
$$R^2 = 1 - \frac{SS_{RES}}{SS_{TOT}} = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2}$$



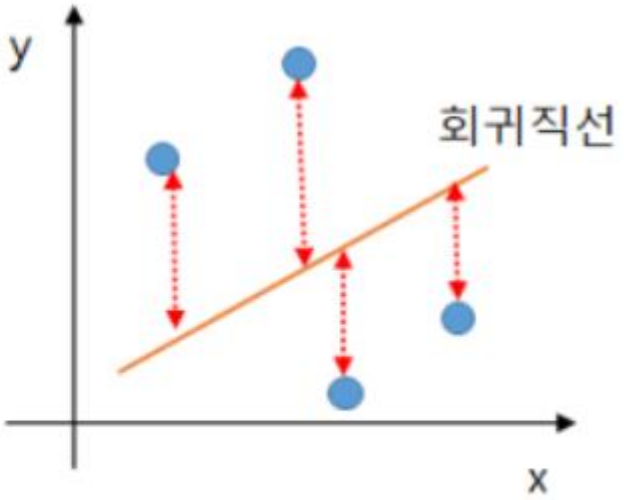
회귀의 평가지표(default): 설명력(결정계수, R2)



R^2가 1에 가까운 경우



R^2가 0.5에 가까운 경우



R^2가 0에 가까운 경우

Day2-Lec4: 분류 실습 VDS Deep Neural Network 적용하기

학습할 내용 : 2일차

- ❖ Deep Neural Network을 이용한 분류
  - ✓VDS 데이터 분류 정확도
  - ✓VDS 라벨 최적화를 통한 분류 정확도 향상
- ❖ 회귀를 통한 예측 문제 다루기
  - ✓선형 함수 회귀
  - ✓비선형 함수 회귀
  - ✓활성함수 종류 다루기
- ❖ 시계열 데이터를 이용한 예측 문제 해결
  - ✓RNN 이해
  - ✓LSTM 이해
  - ✓VDS 데이터를 이용한 SimpleRNN 적용 및 예측하기
- ❖ 교통 데이터를 활용한 Many-to-One, Many-to-Many 응용문제

교통 데이터 가저오기 VDS

```
[3] import pandas as pd
```

```
[4] from pandas import datetime
```

```
[5] def parser(x):
        return datetime.strptime(x, '%Y-%m-%d %H:%M')
```

```
[6] df = pd.read_csv('./daejeon_vds16.csv',  date_parser=parser)
```

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

	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

VDS을 이용한 머신러닝 분류

I. 데이터 가져오기

- 판다스를 이용하여 교통 데이터 가져오기

In [1]:
```
import tensorflow as tf
```

In [2]:
```
import pandas as pd
```

In [4]:
```
df = pd.read_csv('./daejeon_vds16.csv')
df.head()
```

Out[4]:

	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
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

II. 라벨을 정하기

- 지도학습을 위해서 데이터를 Feature와 Label로 나누자.
- 무엇이 Feature이며 무엇이 Label 인가?
- 라벨은 무엇으로 정하는게 좋은가?

In [5]:
```
def get_score(v):
    if v < 30:
        score = 'Jam'
    elif v < 50:
        score = 'Slow'
    else :
        score = 'Fast'
    return score
```

In [6]:
```
df["label_speed"] = df["Speed"].apply(lambda v: get_score(v))
df.head(5)
```

	Date	ToVol	SmVol	MeVol	LaVol	Speed	Occ.Rate	label_speed
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
2	2017-04-02 0:10	46	34	12	0	50.6	1.87	Normal
3	2017-04-02 0:15	45	36	9	0	50.9	1.72	Normal
4	2017-04-02 0:20	27	13	13	1	62.2	1.12	Normal

교통 흐름 분류: 지도학습을 위한 라벨링

1) 입력 X와 출력 y의 값을 정하기

In [7]:
```
import matplotlib.pyplot as plt
```

In [8]:
```
df['label_speed'].unique()
```

Out[8]: array(['Normal', 'Slow', 'Jam'], dtype=object)

- 머신러닝에서 특징(Feature)에 대하여, 그리고 라벨에 대하여 다시한번 복습합시다

In [9]:
```
feature_cols = ['ToVol','Occ.Rate','LaVol', 'MeVol']

target_col = 'label_speed'
X = df[feature_cols]
y = df[target_col]
```

교통 흐름 분류: 지도학습을 위한 라벨링

2) 출력용 라벨을 머신러닝

- 텍스트를 숫자로 바꾸자
- One-Hot Encoding
- 자연어 처리에는 Embedding을 사용함.
- Word2Vec

]:
```
class_dic = {'Jam':0, 'Slow':1, 'Normal':2}
y_ohc = y.apply(lambda z: class_dic[z])

y_ohc.head()
```

3) 데이터를 훈련과 테스트로 나누자

- (실전) 데이터를 validation을 포함해서 나눌수 있다.
- (해보기) 전체 데이터를 train : validation : test = 0.6: 0.2: 0.2 로 나누어라

[2]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y_ohc, test_size=0.20, random_state=30)

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

[4]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Deep Neural Network

- MLP와 ReLu를 활용한 신경망을 사용하자

In [15]:
```
import tensorflow as tf

from tensorflow import keras
from tensorflow.keras import Sequential, optimizers
from tensorflow.keras.layers import Flatten, Dense, Softmax

num_features = len(X_train[1])

model = Sequential([
                    Dense(64, activation='relu',input_shape=[num_features]),
                    Dense(64, activation='relu'),
                    Dense(32, activation='relu'),
                    Dense(3, activation='softmax')
])
```

모델 아키텍처 요약

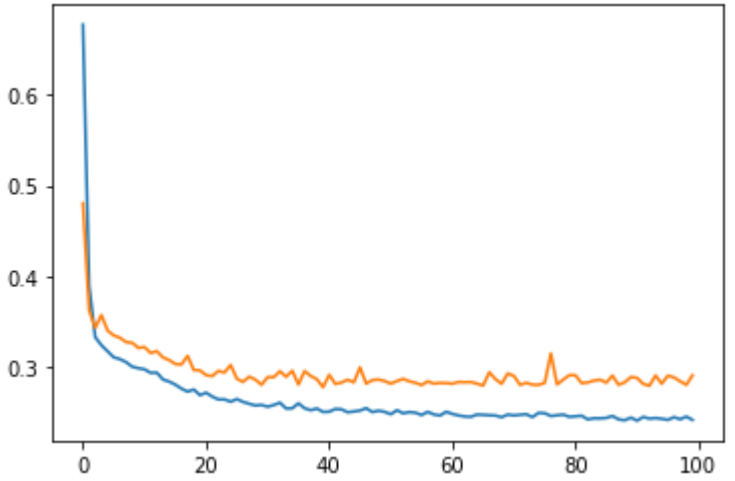
In [21]:
```
model.summary()
```

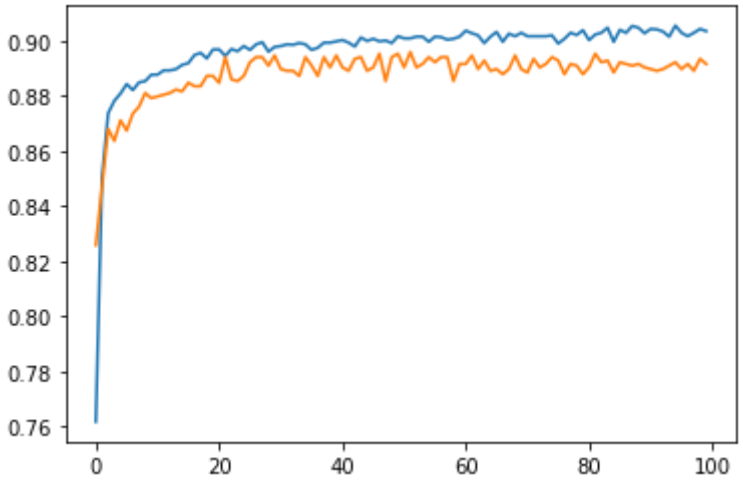
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 64)                320

dense_1 (Dense)              (None, 64)                4160

dense_2 (Dense)              (None, 32)                2080

dense_3 (Dense)              (None, 3)                 99

=================================================================
Total params: 6,659
Trainable params: 6,659
Non-trainable params: 0
_________________________________________________________________
```

모델 훈련

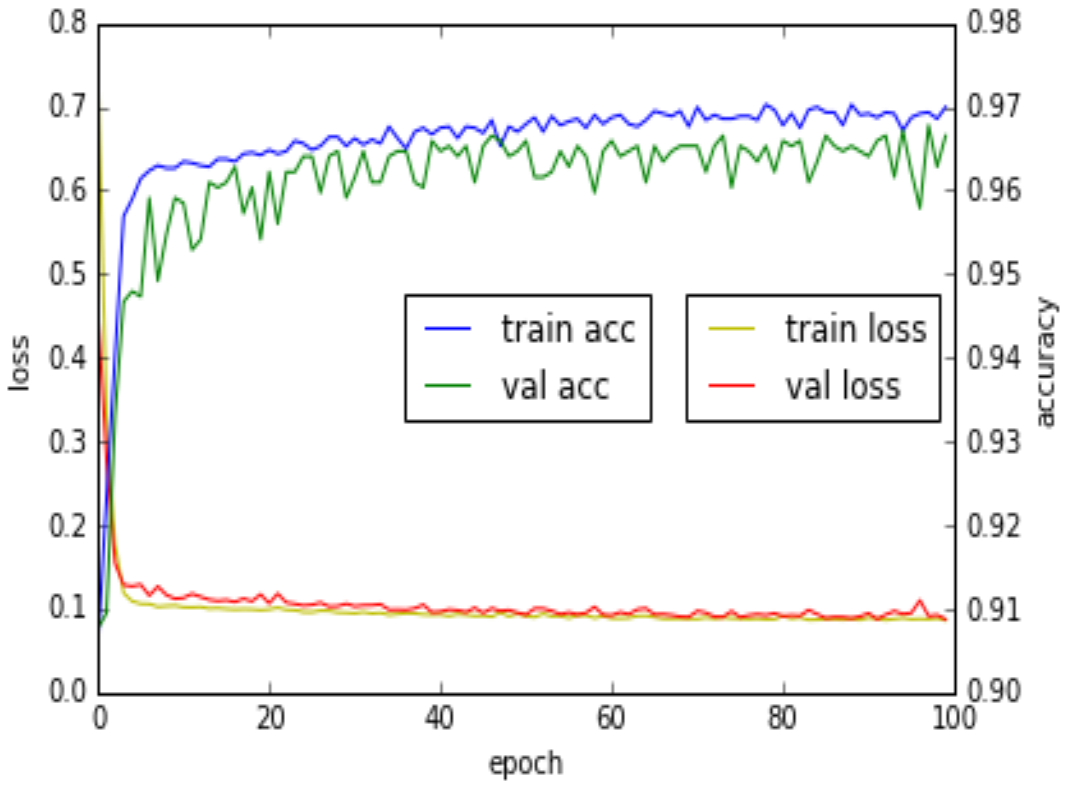
In [22]:
```
model.compile(loss='SparseCategoricalCrossentropy',
              optimizer='adam', metrics=['accuracy'])

history = model.fit(X_train,y_train, epochs=100,
                    validation_data=(X_test,y_test),
                    batch_size=128, verbose=2)

model.evaluate(X_test,y_test)

history.history.keys()
```

In [26]:
```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
```





6) DNN 딥러닝 모델을 이용한 VDS 데이터 분석




```
%matplotlib inline
import matplotlib.pyplot as plt

fig, loss_ax = plt.subplots()

acc_ax = loss_ax.twinx()

loss_ax.plot(history.history['loss'],'y',label='train loss')
loss_ax.plot(history.history['val_loss'],'r',label='val loss')
acc_ax.plot(history.history['accuracy'],'b',label='train acc')
acc_ax.plot(history.history['val_accuracy'],'g',label='val acc')

loss_ax.set_xlabel('epoch')
loss_ax.set_ylabel('loss')
acc_ax.set_ylabel('accuracy')

loss_ax.legend(loc='center right')
acc_ax.legend(loc='center')
plt.show()
```

6) DNN 딥러닝 모델을 이용한 VDS 데이터 분석

In [67]:
```
models = pd.DataFrame({
    'Model': ['Logistic Regression',  'Support Vector Machines','RandomForest',
              'K-Nearest Neighbours', 'Decision Tree','Deep Learning'],
    'Score': [acc_lr, acc_svm, acc_rf, acc_knn, acc_dt, acc_dnn]})
models.sort_values(by='Score', ascending=False)
```

Out [67] :

	Model	Score
5	Deep Learning	0.969927
1	Support Vector Machines	0.967142
3	K-Nearest Neighbours	0.967142
2	RandomForest	0.961562
4	Decision Tree	0.961562
0	Logistic Regression	0.960322

(1) 연습문제 해보기 : 속도

- VDS 데이터의 라벨을 속도(Speed)에 대하여 자신의 라벨을 정의하시오
- 예 (20, 50) 등
- 훈련(Train)과 시험(Test)의 정확도 혹은 손실(Loss)를 제출하시오

(2) 연습문제 해보기 : 교통량

교통량(ToVol) 혹은 SmVol로 라벨을 정하는 방법을 적용하세요

- 예 (100, 300) 등
- 훈련(Train)과 시험(Test)의 정확도 혹은 손실(Loss)를 제출하시오

(3) 연습문제 해보기

- VDS 데이터의 라벨을 점유률(Occ.Rate)에 대하여 자신의 라벨을 정의하시오
- 예 (8, 16) 등
- 훈련(Train)과 시험(Test)의 정확도 혹은 손실(Loss)를 제출하시오

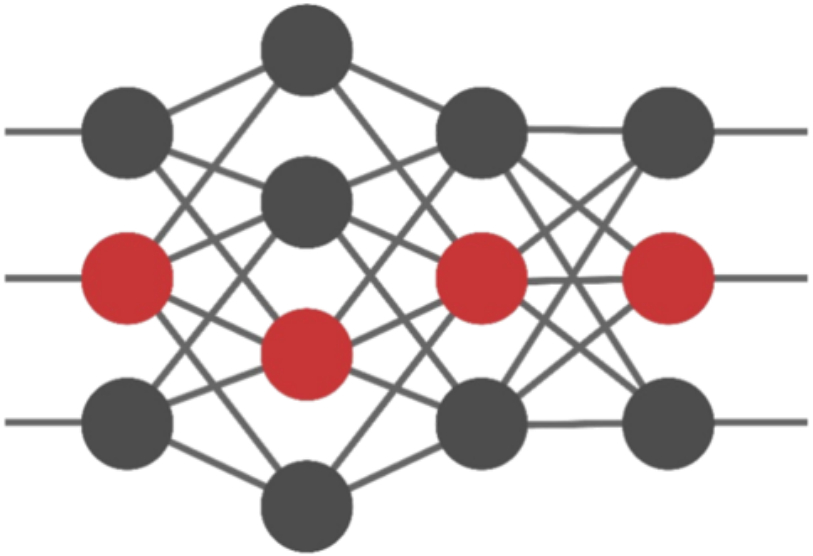
(4) 연습문제 해보기

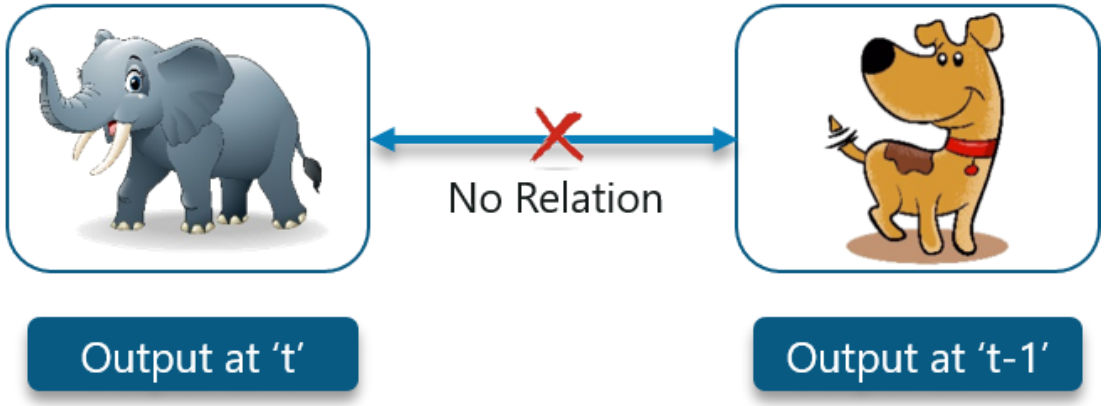
- 머신러닝 방법과 DNN 방법을 비교하여 자장 휼룡한 신경망 방법과 하이퍼파라미터를 설정하세요.

Day2-Lec5: 강의 RNN 개념 이해

Why Not Feedforward Networks

- 기존 FeedForwared Networks에서는
  - ✓입력은 개 이미지이고, 학습이 잘된 신경망은 개의 라벨을 출력한다.
  - ✓바로 다음 코끼리 이미지를 입력을 사용하면 코끼리(라벨)을 출력한다.
  - ✓하지만 코끼리와 개는 서로 관련성이 없고 독립적이다.

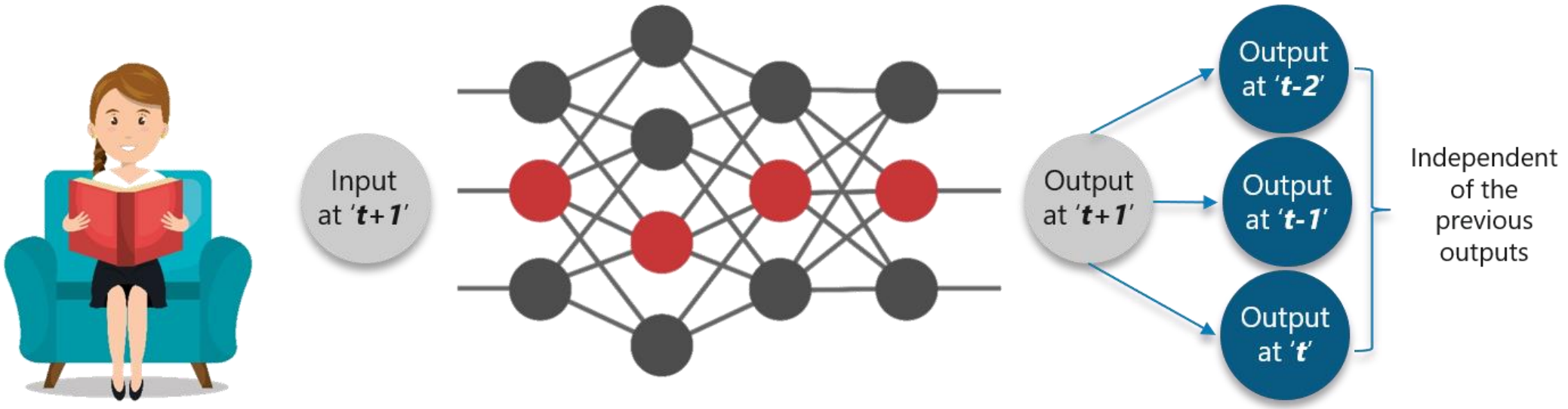




ref: https://www.edureka.co/blog/recurrent-neural-networks/

Why Not Feedforward Networks

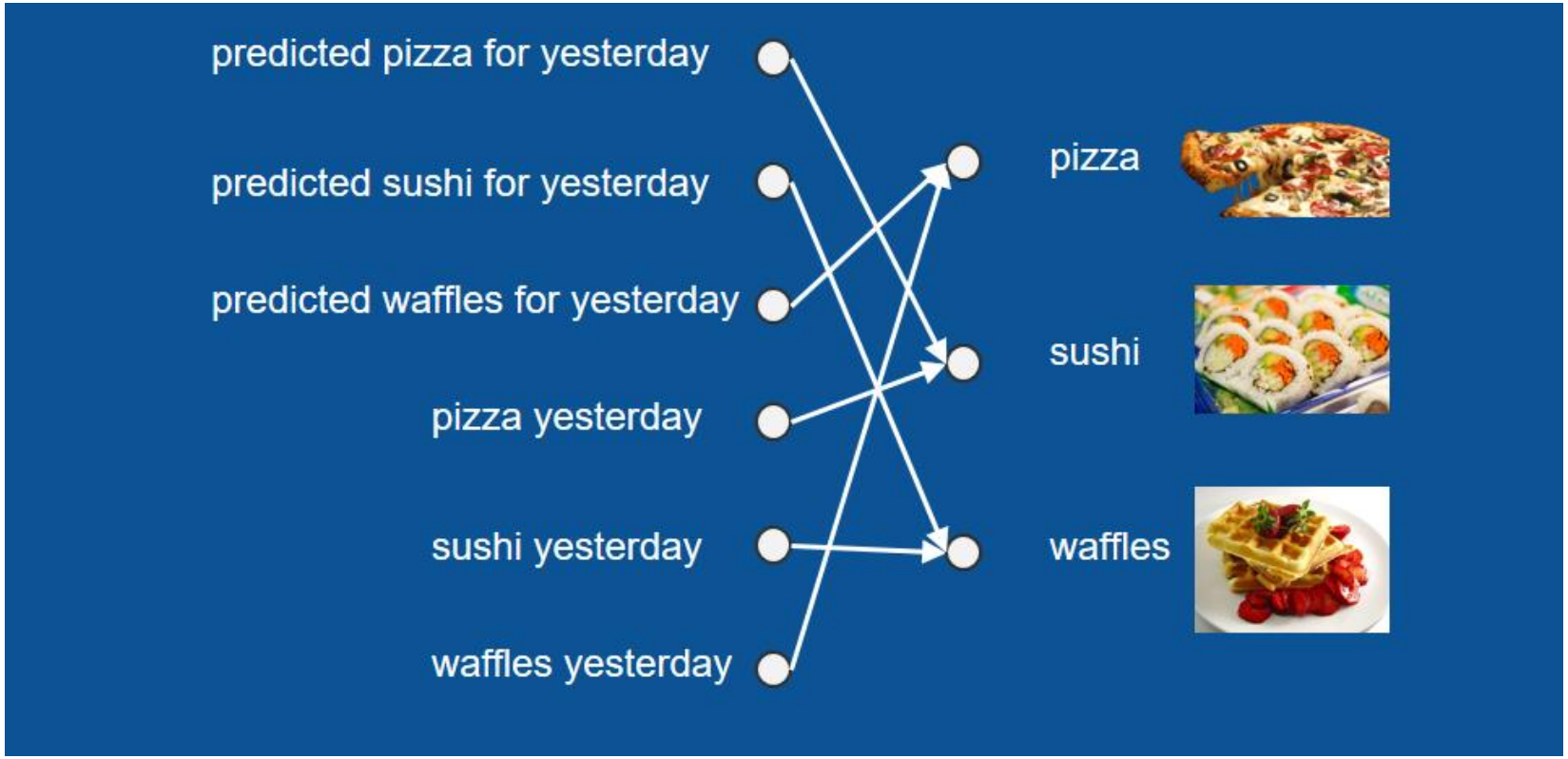
- 책을 읽고 있을 경우, 이전 페이지의 내용을 잘 알고 있어야 한다.
  - ✓책의 문맥을 이해하기 위해서는 이전의 내용 및 이전 단어를 기억하고 있어야 한다.
  - ✓그래서 Feeforward 네트워크는 이전 단어들로 다음 단어의 관계를 예측해 줄 수 없다.
  - ✓ 즉 이전의 내용(단어)를 기억해 줄 수 있는 네트워크가 필요하다.



What Are Recurrent Neural Networks?

❖ 음식을 먹을 경우도 이전 기억이 중요하다.

✓ RNNs are a **type** of **artificial neural network** designed to **recognize patterns** in sequences of data



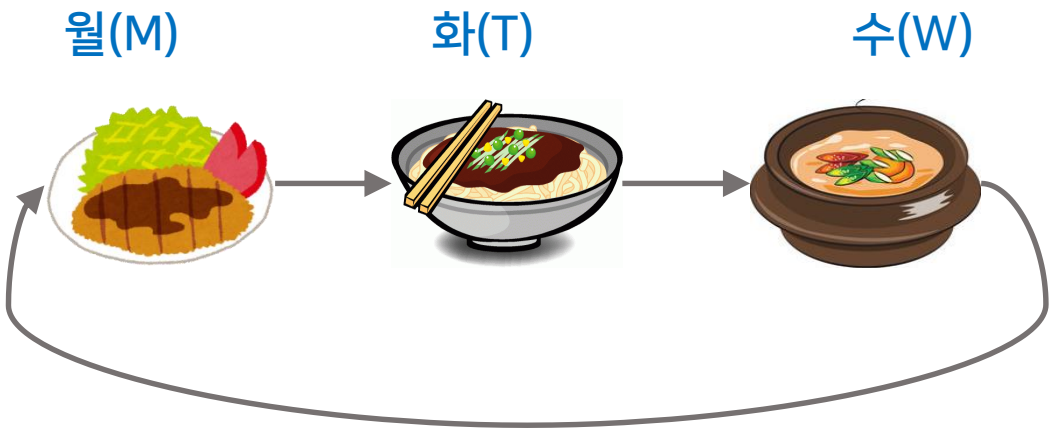
간단한 순환신경망(RNN)

❖ 함께 살고 있는 룸메이트는 요리를 하느데 3종류만 할 줄 안다.

✓친구는 저녁을 하는데 3가지 음식만 만든다.

✓월요일은 돈까스, 화요일은 짜장면, 수요일은 찌게 등 순차적으로 3가지 음식을 반복한다.

✓룸메이트가 기억하고 있는 패턴은 단순하다.

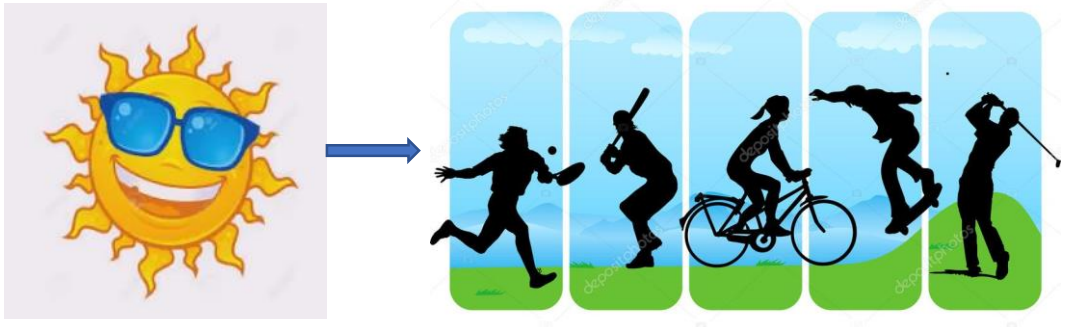


RNN = Recurrent Neural Network

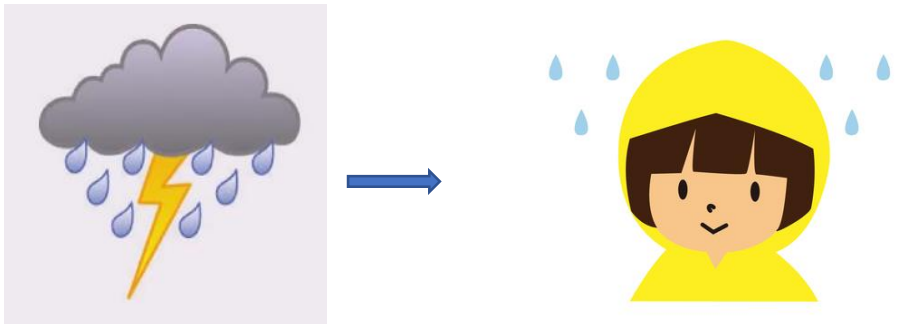
❖ 실제로 룸메이트는 날씨에 따라서 식사 메뉴에 규칙을 가지고 있다.

✓날씨가 좋은 날은 야외활동해서 늦게 귀가하며 전날 먹은 음식을 다시 먹는다.

✓하지만 비가 오는 날은 일찍 집에 귀하하여 원래 패턴을 따라서 음식을 만든다.



어제 먹었던 음식

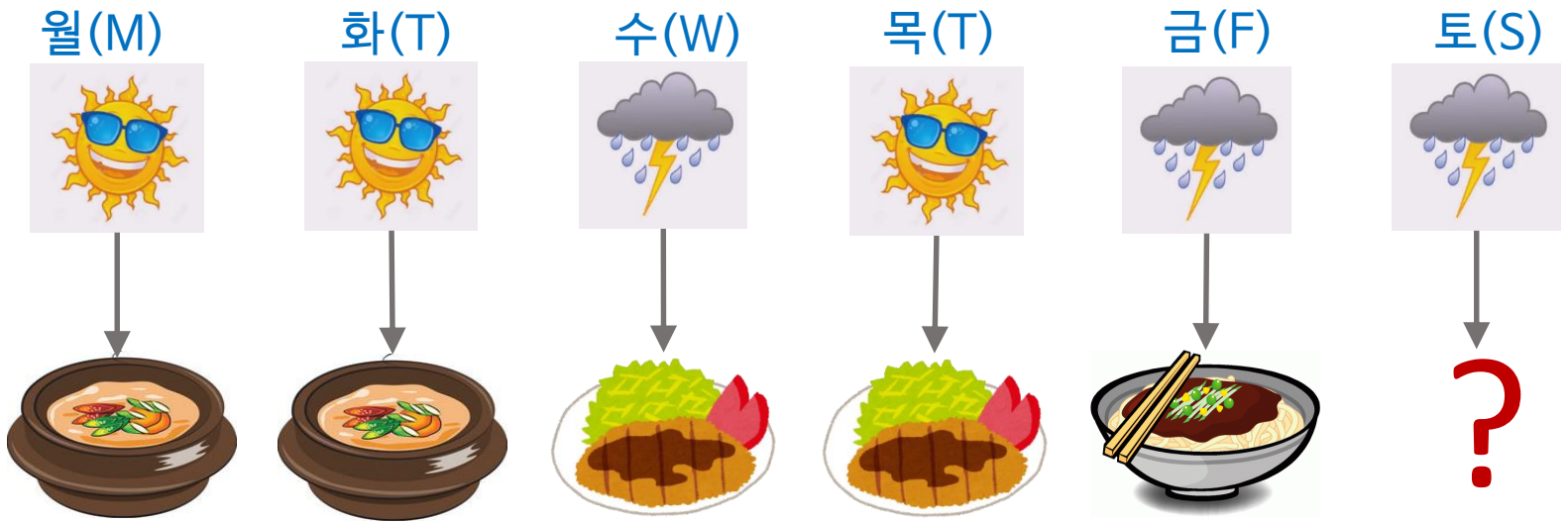


원래 RNN 패턴

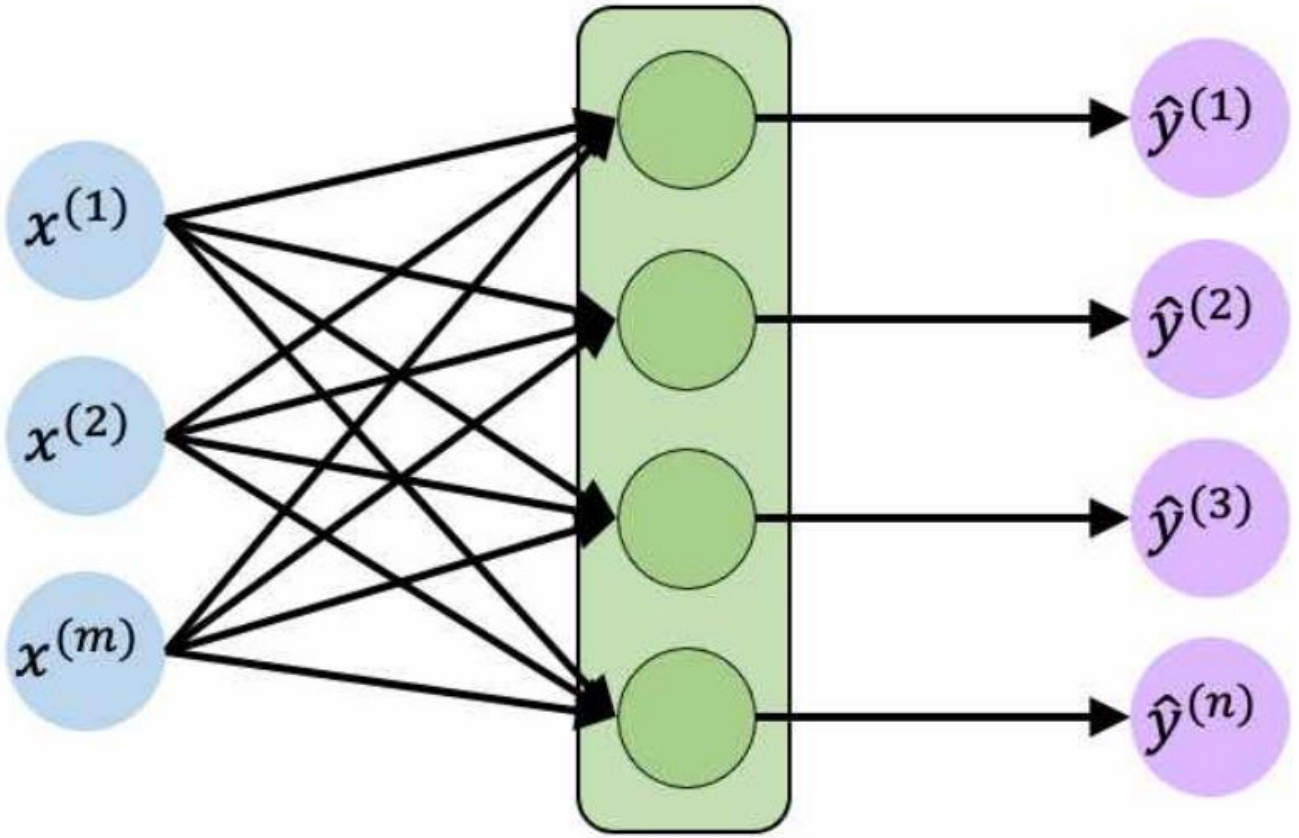
RNN depending on the weather

❖ 나는 오늘 아침 날씨에 따라서 저녁에 먹을 음식을 예측할 수 있다.

✓월요일은 저녁에 찌게를 먹었고, 화요일 아침 날씨가 좋아서 저녁에도 찌게를 먹었다.
✓수요일은 비가와서 원래 패턴인 돈까스, 목요일은 좋은 날씨라서 어제 먹은 돈까스
✓금요일은 비가와서 원래 패턴인 짜장면을 먹었다.
✓토요일에 비가 왔는데 저녁 메뉴는 무었인가?

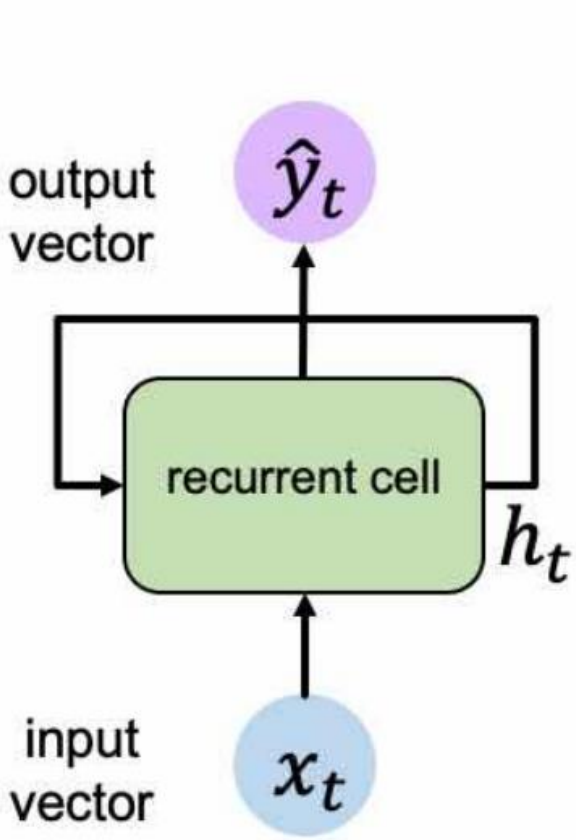


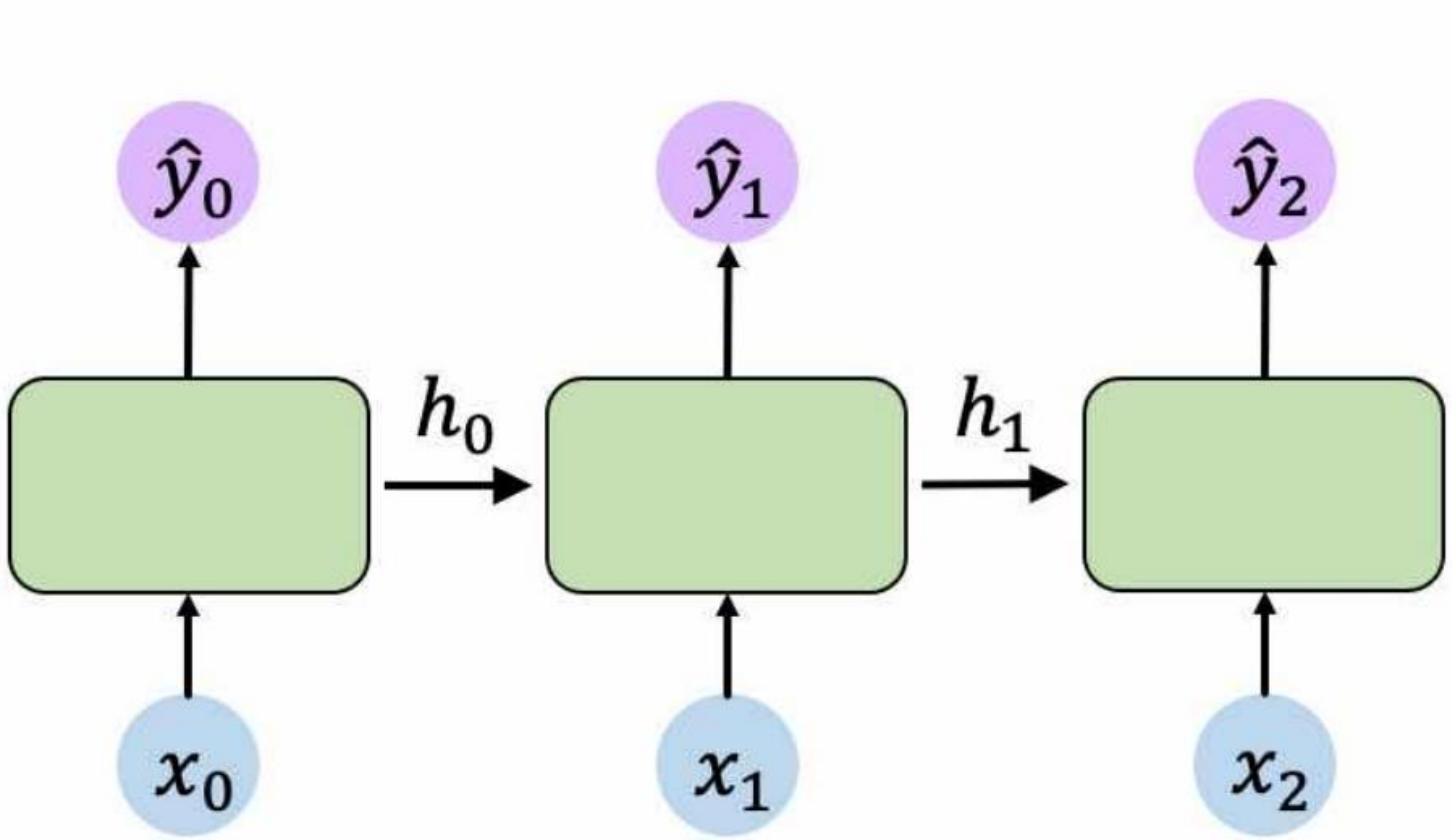
Feed-Forward Networks Revisted



$$\boldsymbol{x} \in \mathbb{R}^m \qquad \hat{\boldsymbol{y}} \in \mathbb{R}^n$$

Neurons with Recurrence

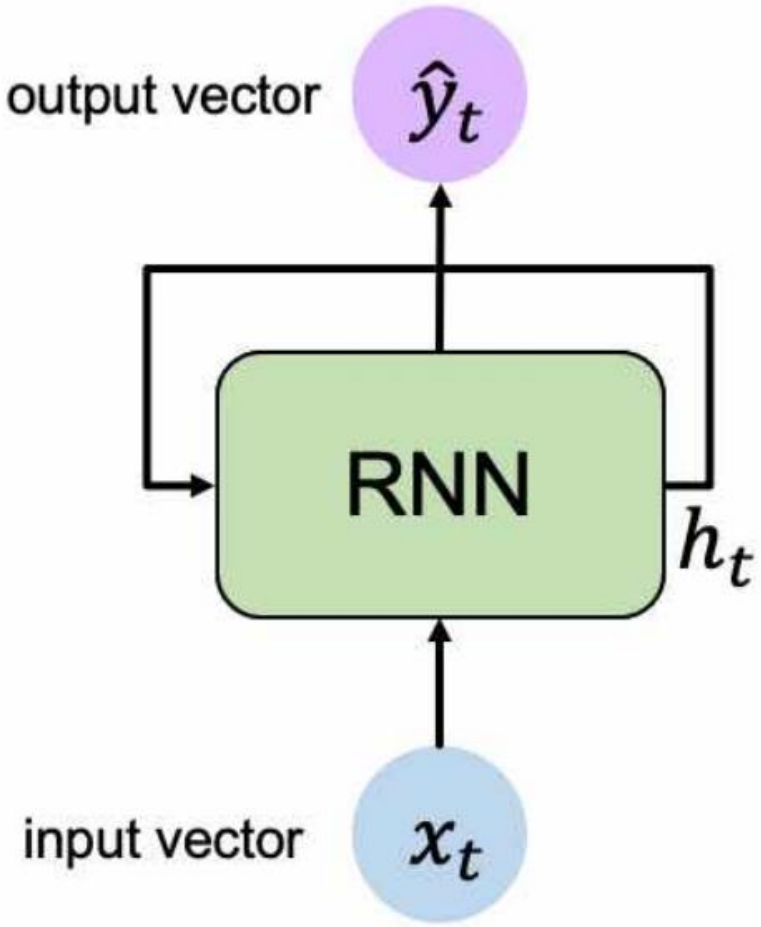




memory cell

$$\underbrace{\hat{y}_t}_{\text{output}} = f(\underbrace{x_t}_{\text{input}}, \underbrace{h_{t-1}}_{\text{past memory}})$$

RNN State Update and Output



Output Vector

$$\hat{y}_t = \boldsymbol{W}_{hy}^{T} h_t$$

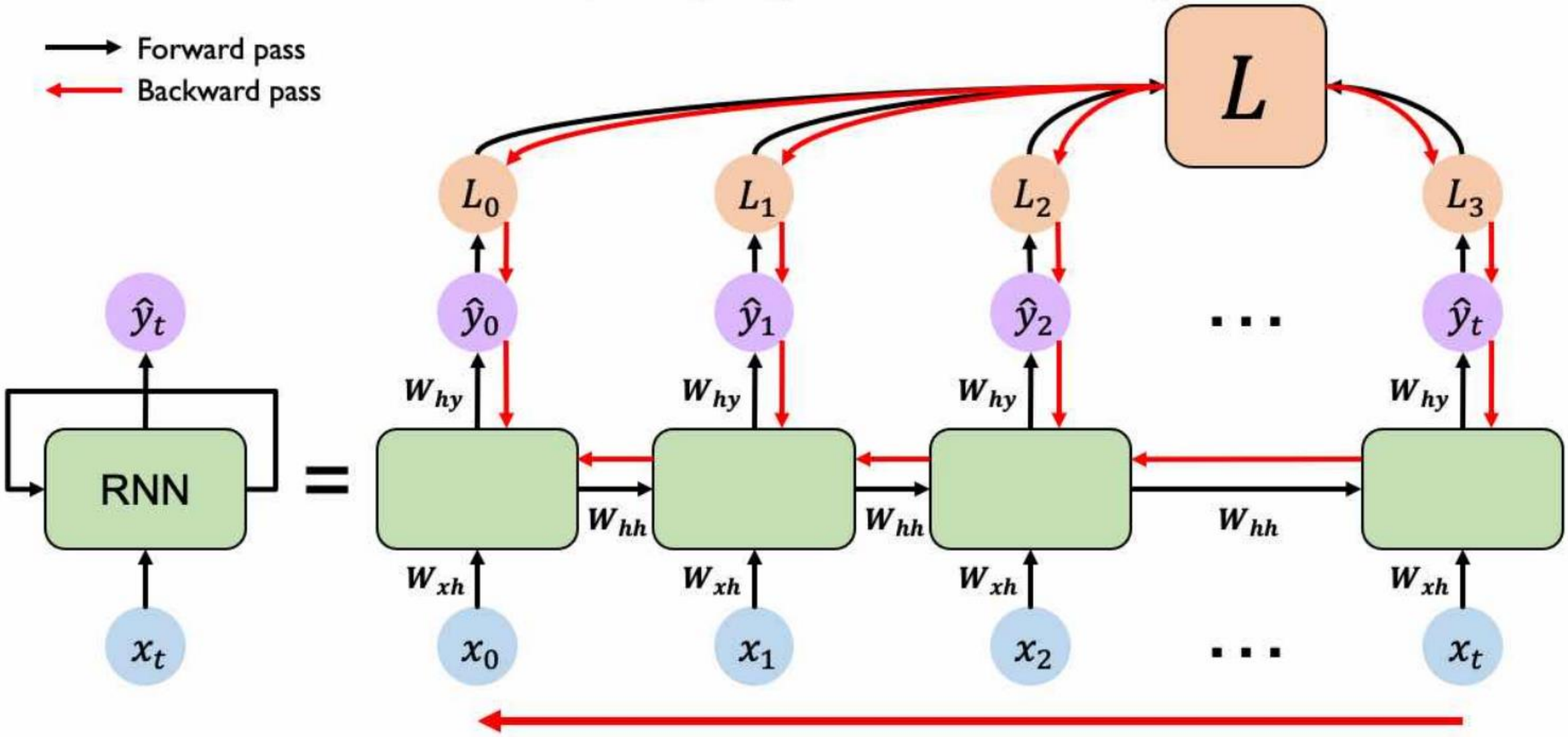
Update Hidden State

$$h_t = \tanh(\boldsymbol{W}_{hh}^{T} h_{t-1} + \boldsymbol{W}_{xh}^{T} x_t)$$

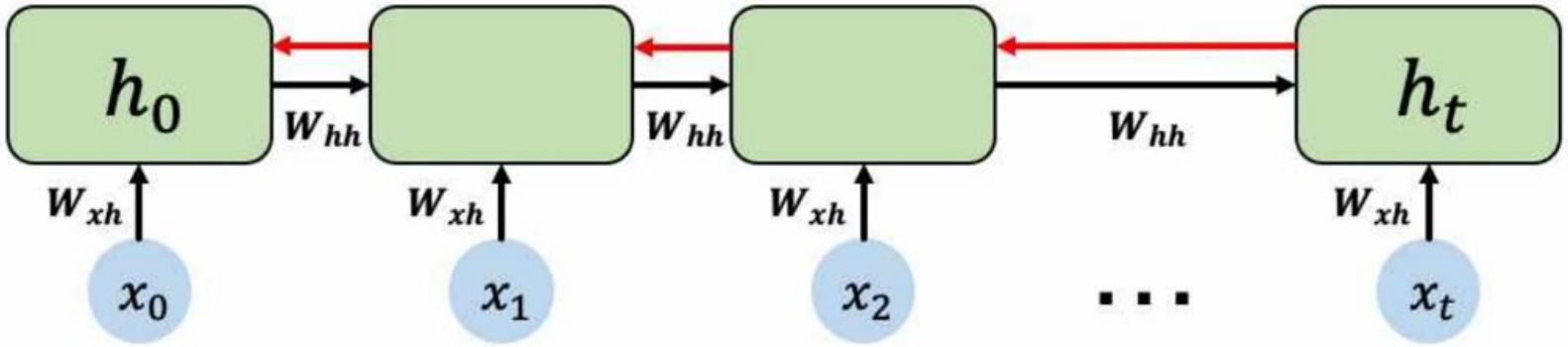
Input Vector

$$x_t$$

Backpropagation Through Time



Standard RNN Gradient Flow: Expoding Gradients



Computing the gradient wrt h_0 involves **many factors of W_{hh} + repeated gradient computation!**

$$\frac{\partial L^{(t)}}{\partial \mathbf{W}_{hh}} = \frac{\partial L^{(t)}}{\partial y^{(t)}} \cdot \frac{\partial y^{(t)}}{\partial \mathbf{h}^{(t)}} \cdot \left(\sum_{k=1}^{t} \frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{h}^{(k)}} \cdot \frac{\partial \mathbf{h}^{(k)}}{\partial \mathbf{W}_{hh}} \right)$$

computed as a multiplication of adjacent time steps:

$$\frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{h}^{(k)}} = \prod_{i=k+1}^{t} \frac{\partial \mathbf{h}^{(i)}}{\partial \mathbf{h}^{(i-1)}}$$

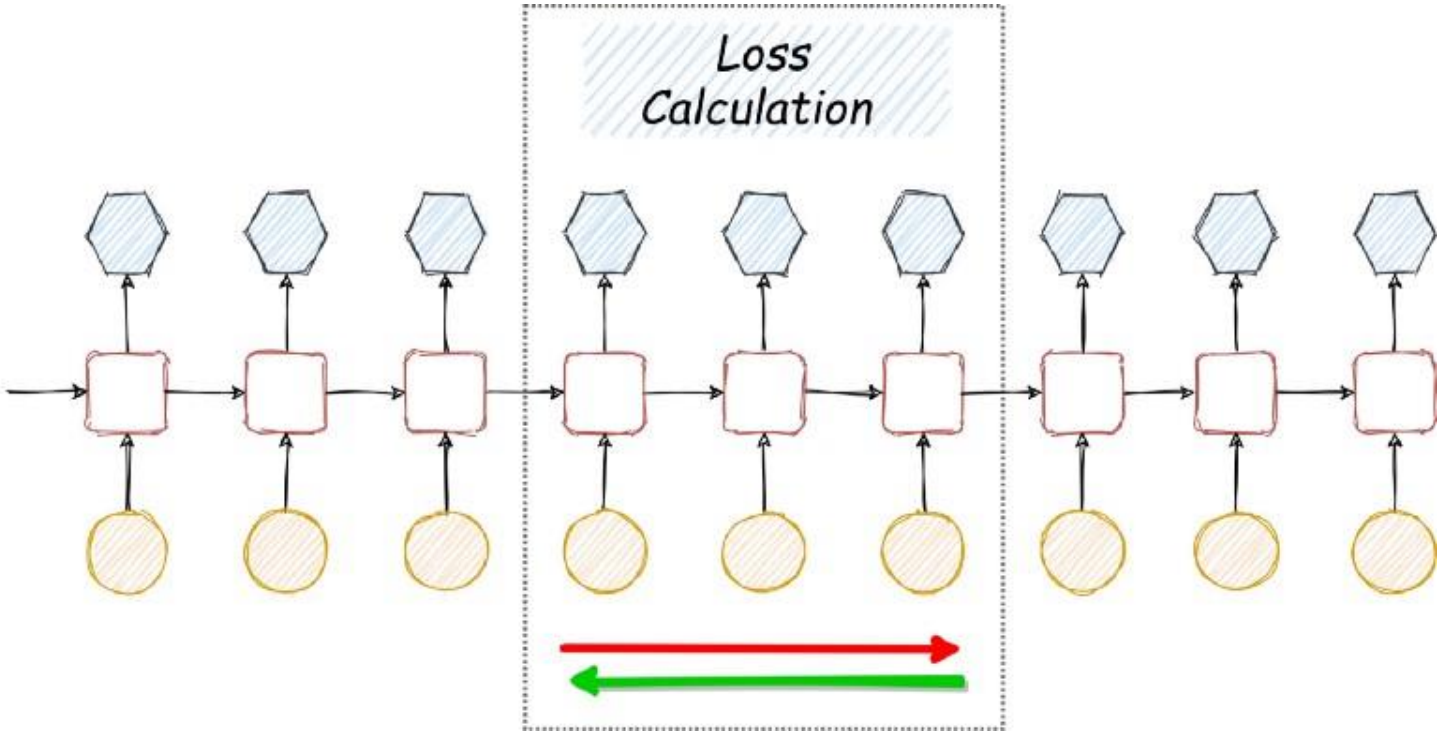
1. *Vanishing gradient* $\left\| \frac{\partial h_i}{\partial h_{i-1}} \right\|_2 < 1$
2. *Exploding gradient* $\left\| \frac{\partial h_i}{\partial h_{i-1}} \right\|_2 > 1$

Solutions to the vanishing/exploding gradient problems

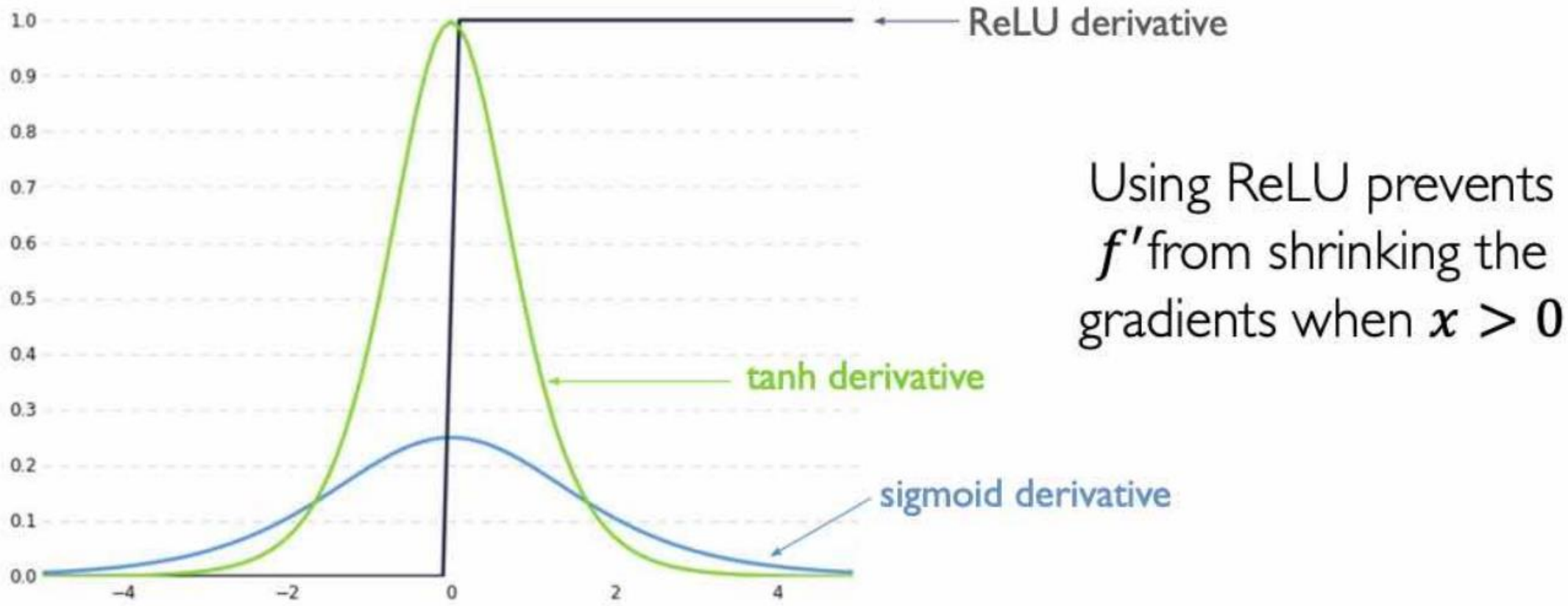
- **Truncated backpropagation through time (TBPTT)**
  - ✓ simply limits the number of time steps the signal can backpropagate after each forward pass.
- **Long short-term memory (LSTM)**
  - ✓ uses a memory cell for modeling long-range dependencies and avoid VGP
- **Gradient Clipping**
  - ✓ set a max value for gradients if they grow to large

Truncated Backpropagation Through Time

Truncated BPTT trick tries to overcome the VGP by considering a moving window through the training process



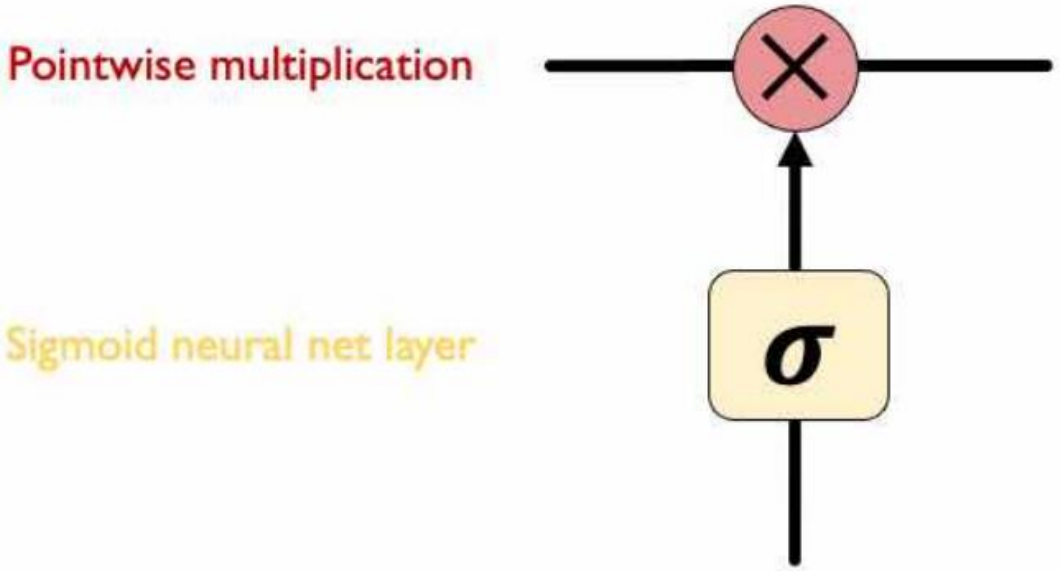
Trick : Activation Fuctions

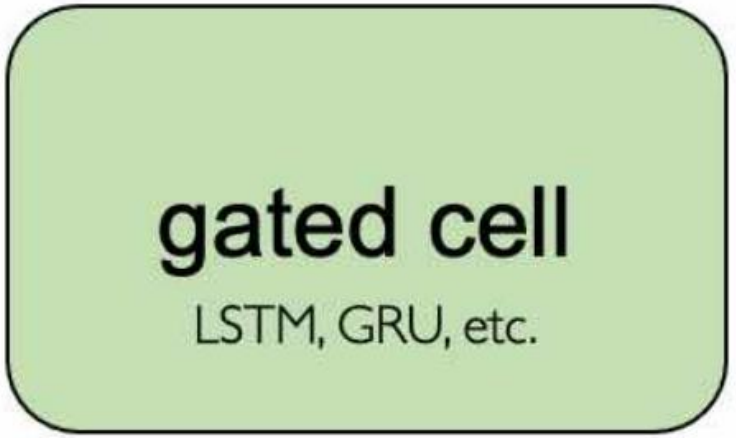


Using ReLU prevents f' from shrinking the gradients when $x > 0$

Trick : Gated Cells

Idea: use **gates** to selectively **add** or **remove** information within **each recurrent unit with**



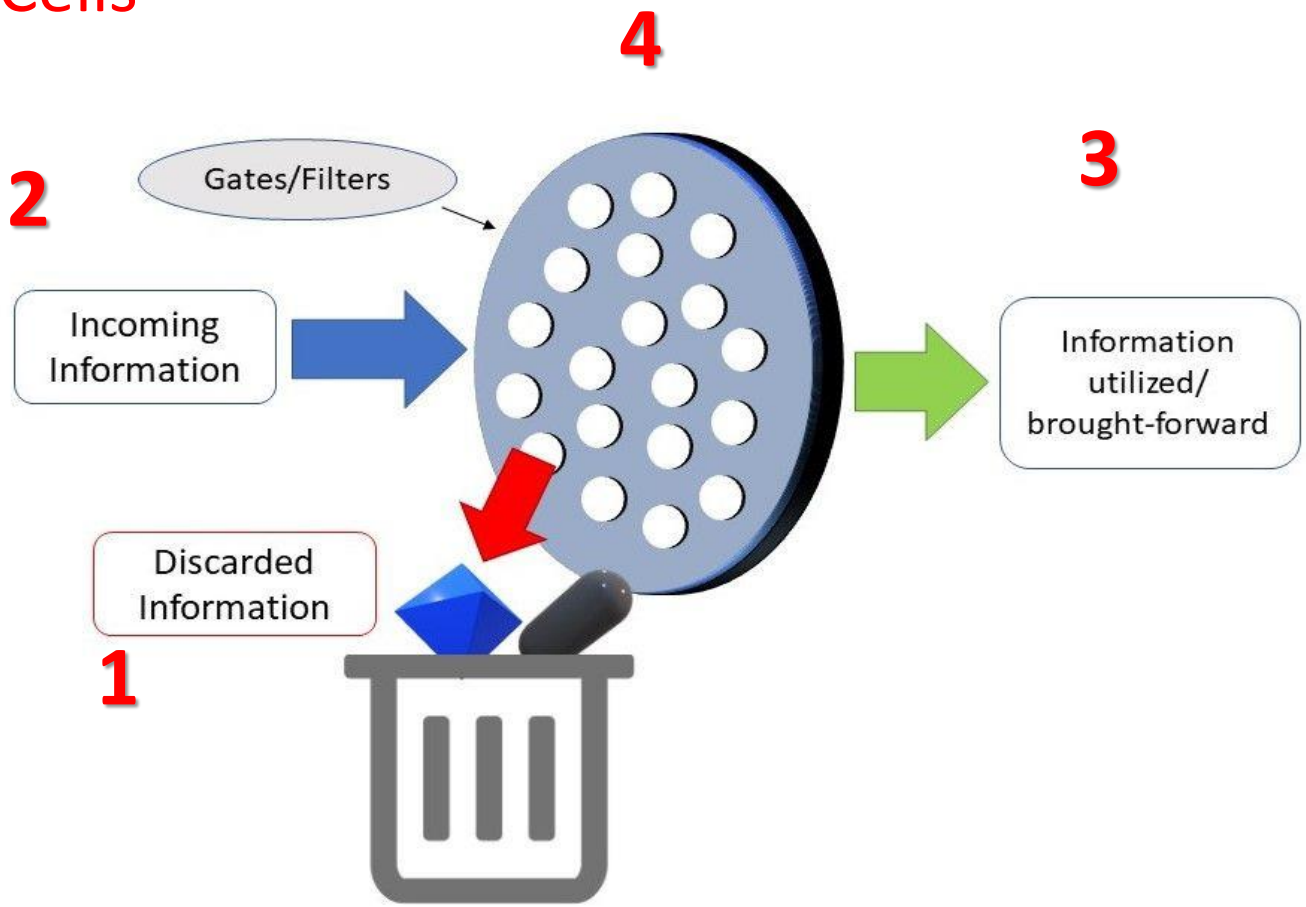


Gates optionally let information through the cell

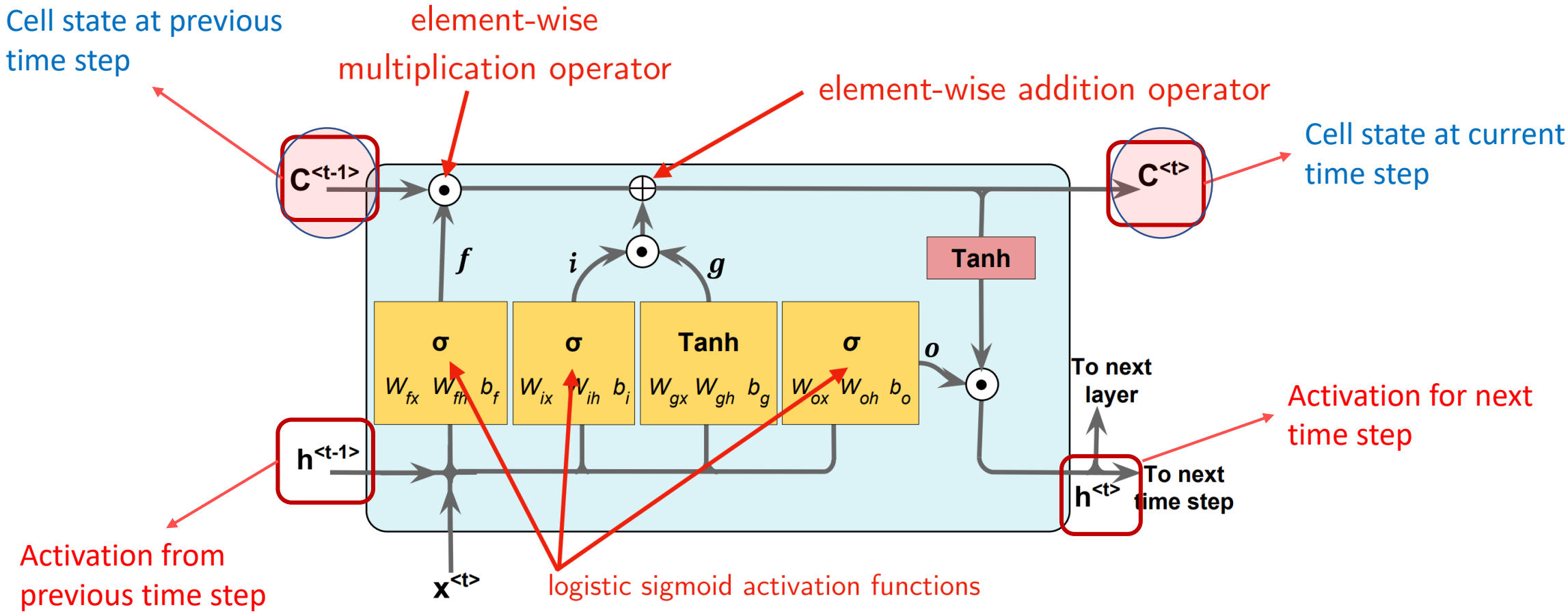
Long Short Term Memory (LSTMs) networks rely on a gated cell to track information throughout many time steps.

Gates control the flow of information to/from the memory

Trick #3: Gated Cells



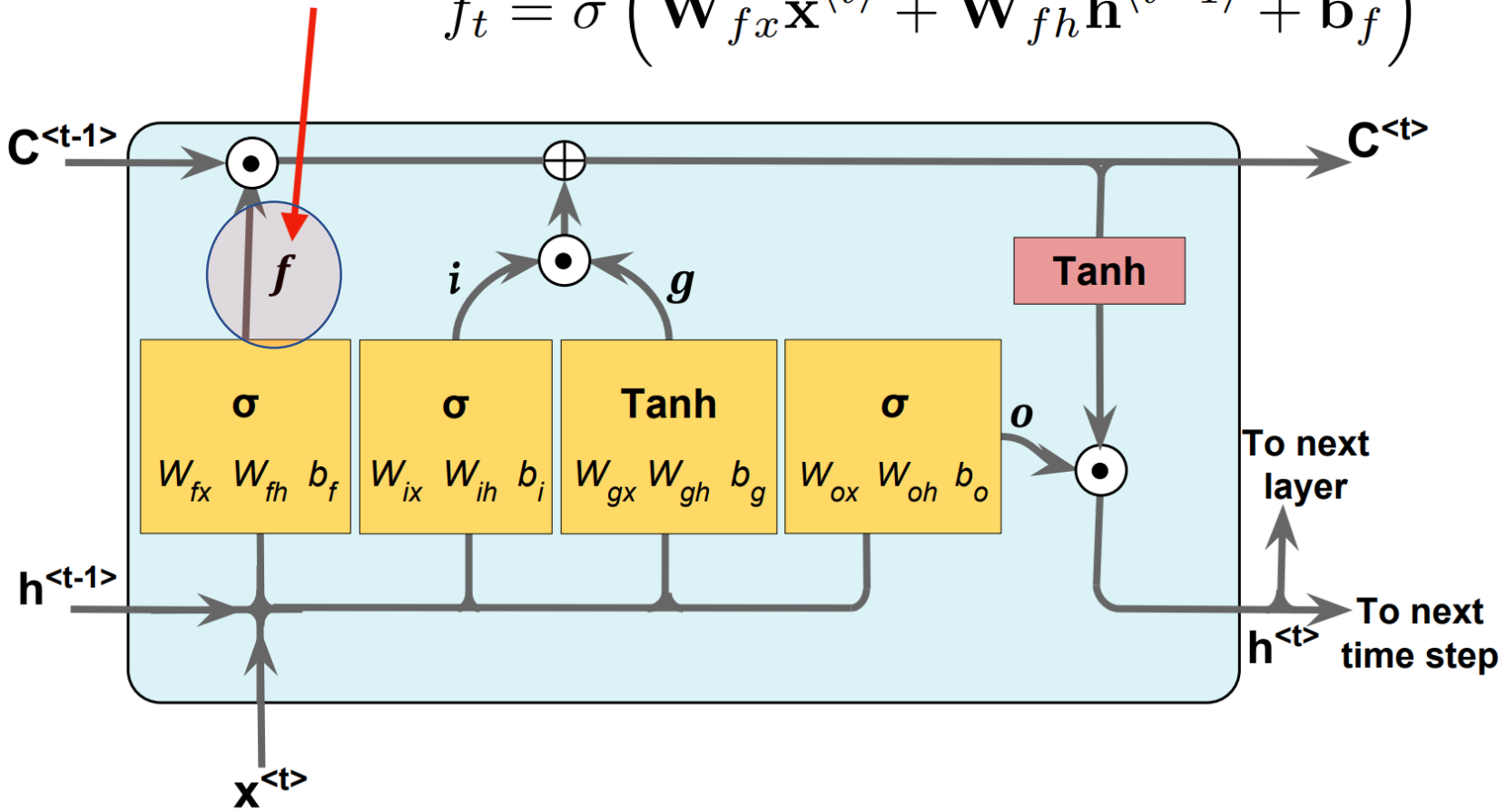
Long-short term memory (LSTM) : Cell state



Long-short term memory (LSTM) : Cell

"Forget Gate": controls which information is remembered, and which is forgotten; can reset the cell state

$$f_t = \sigma\left(\mathbf{W}_{fx}\mathbf{x}^{\langle t\rangle} + \mathbf{W}_{fh}\mathbf{h}^{\langle t-1\rangle} + \mathbf{b}_f\right)$$

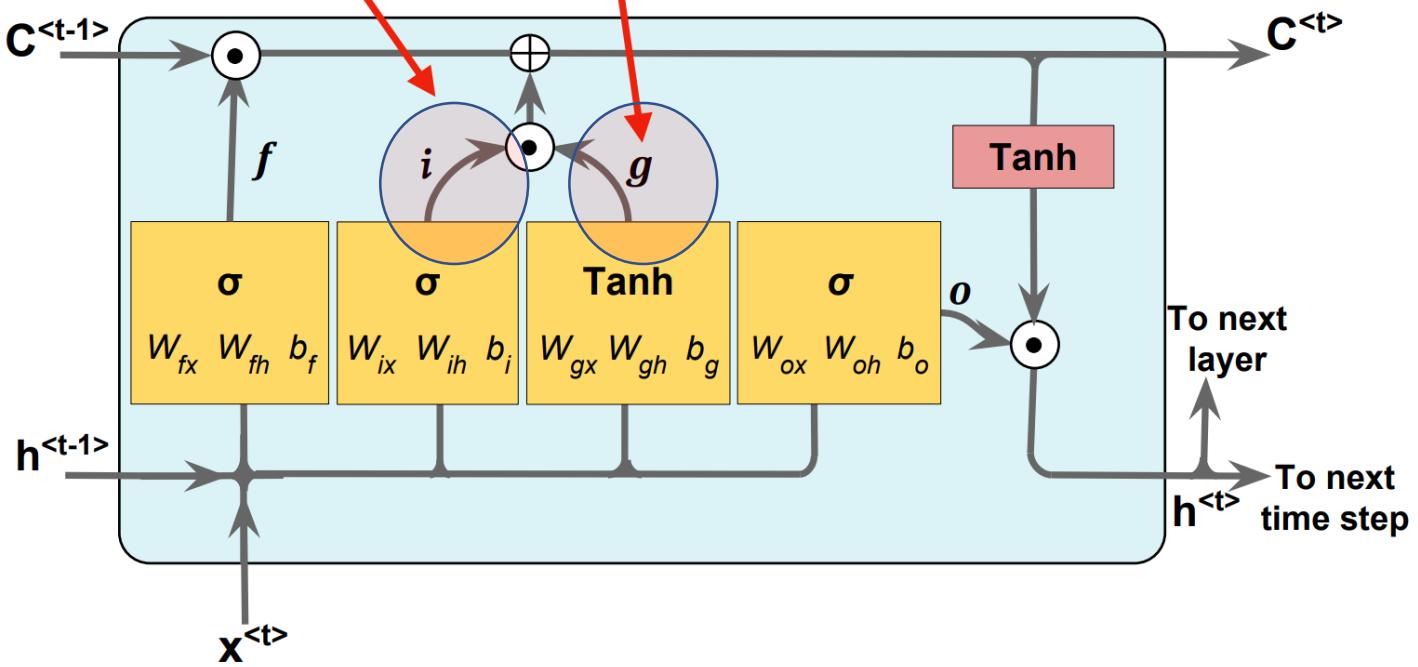


Long-short term memory (LSTM) : Cell

"Input Gate": $\mathbf{i}_t = \sigma\left(\mathbf{W}_{ix}\mathbf{x}^{\langle t\rangle} + \mathbf{W}_{ih}\mathbf{h}^{\langle t-1\rangle} + \mathbf{b}_i\right)$

"Input Node":

$$\mathbf{g}_t = \tanh\left(\mathbf{W}_{gx}\mathbf{x}^{\langle t\rangle} + \mathbf{W}_{gh}\mathbf{h}^{\langle t-1\rangle} + \mathbf{b}_g\right)$$



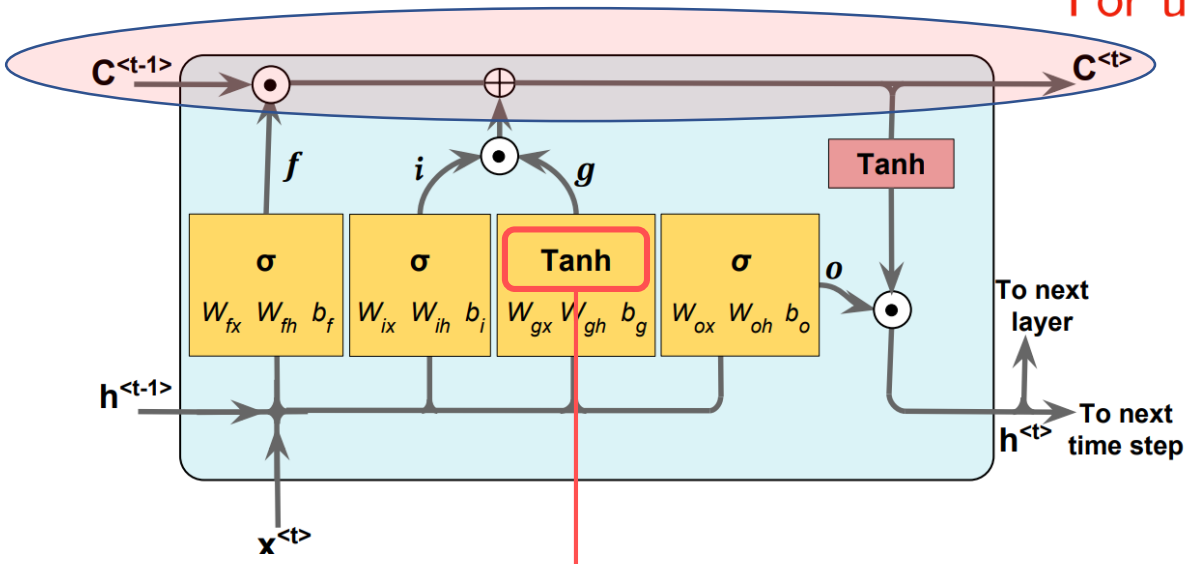
LSTM : Cell state

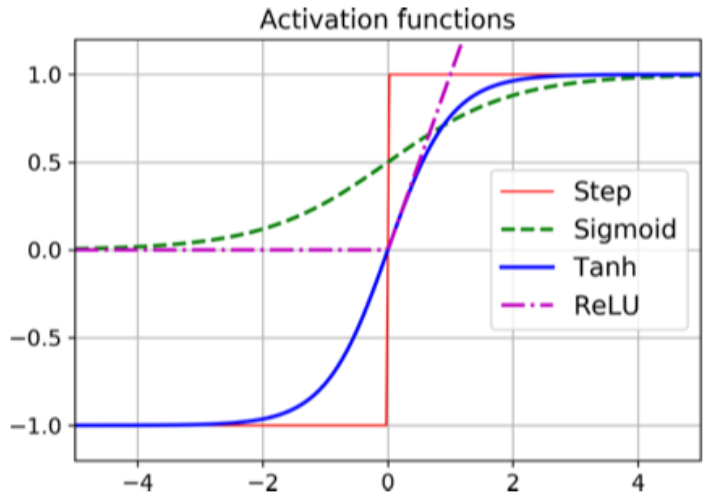
Brief summary of the gates so far ...

Forget Gate Input Node Input Gate

$$C^{\langle t \rangle} = \left(C^{\langle t-1 \rangle} \odot f_t \right) \oplus \left(i_t \odot g_t \right)$$

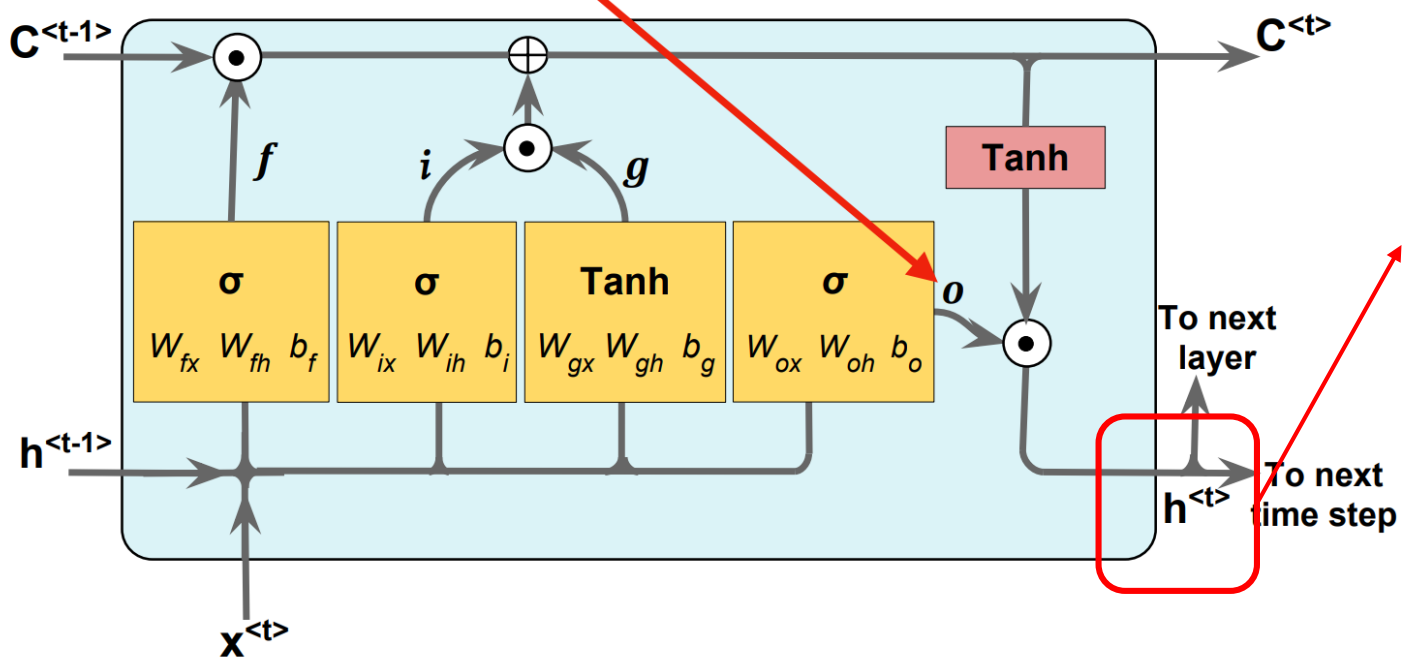
For updating the cell state





Output gate for updating the values of hidden units:

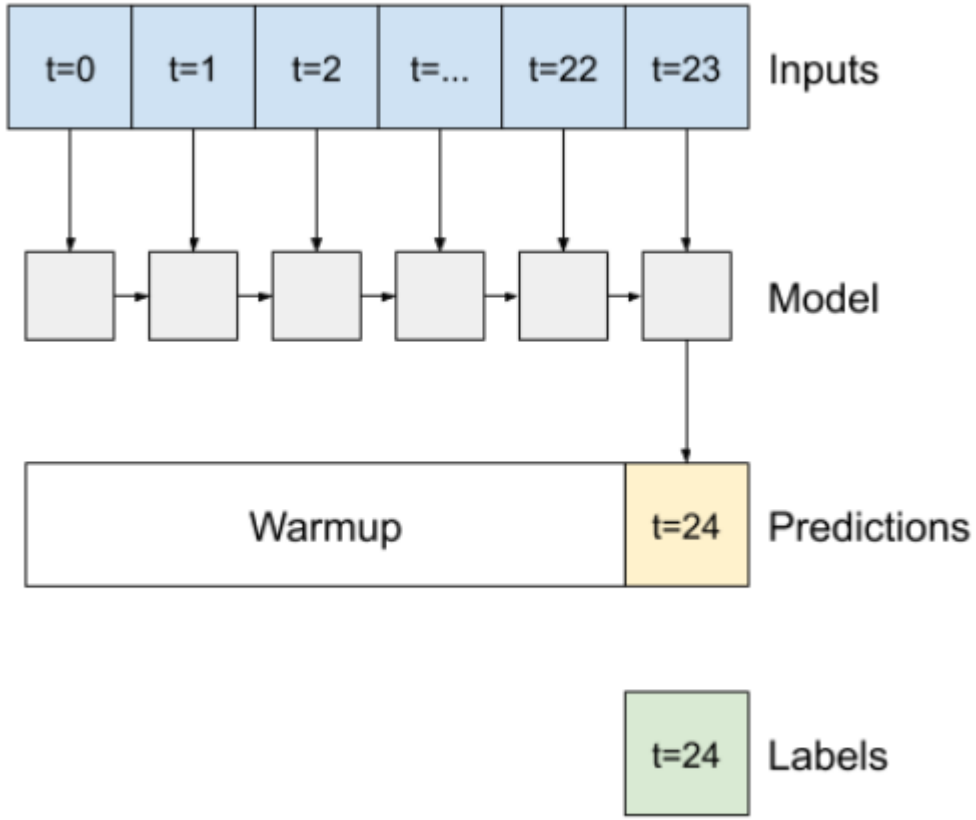
$$\mathbf{o}_t = \sigma\left(\mathbf{W}_{ox}\mathbf{x}^{\langle t\rangle} + \mathbf{W}_{oh}\mathbf{h}^{\langle t-1\rangle} + \mathbf{b}_o\right)$$



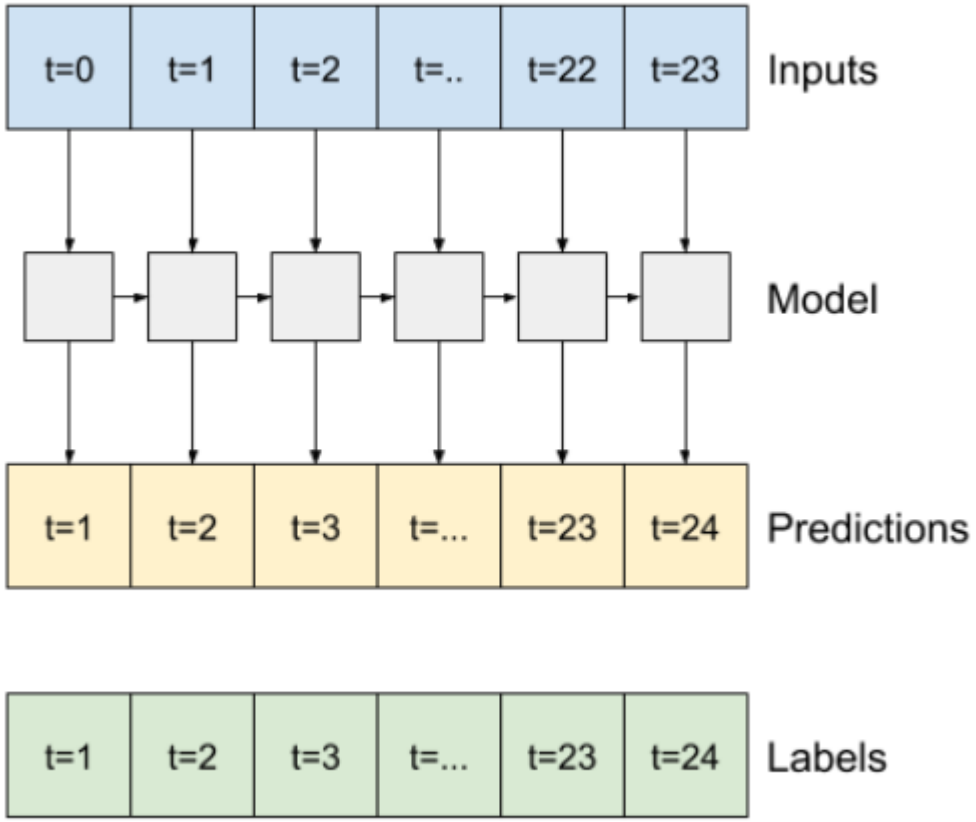
$$\mathbf{h}^{\langle t\rangle} = \mathbf{o}_t \odot \tanh\left(\mathbf{C}^{\langle t\rangle}\right)$$

Day2-Lec6: 회귀
VDS-RNN 예측하기

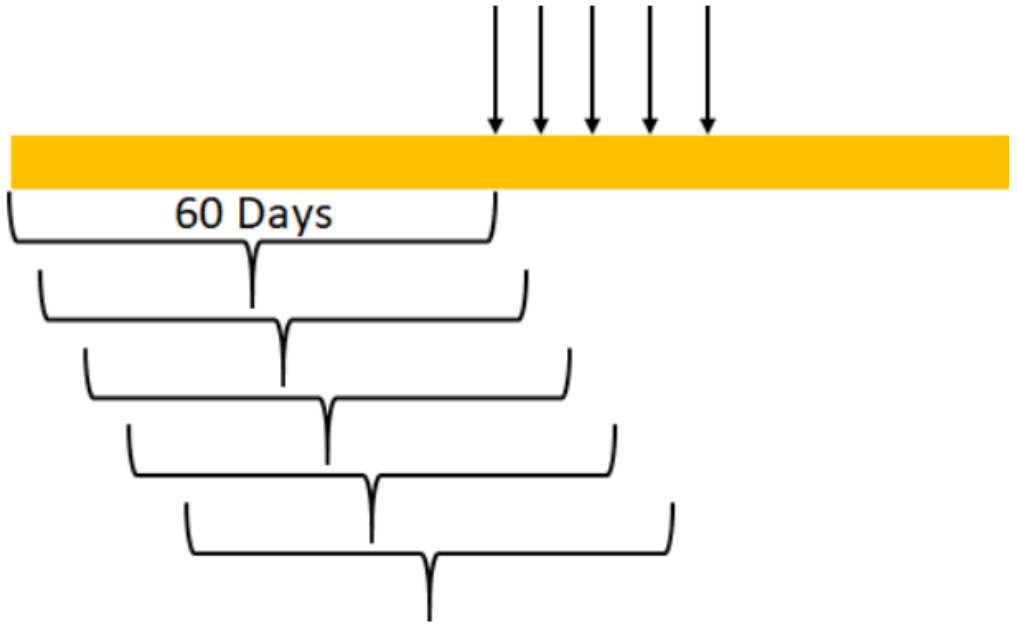
Recurrent Neural network



Stacking RNN layers



LSTM을 이용한 교통 흐름 예측

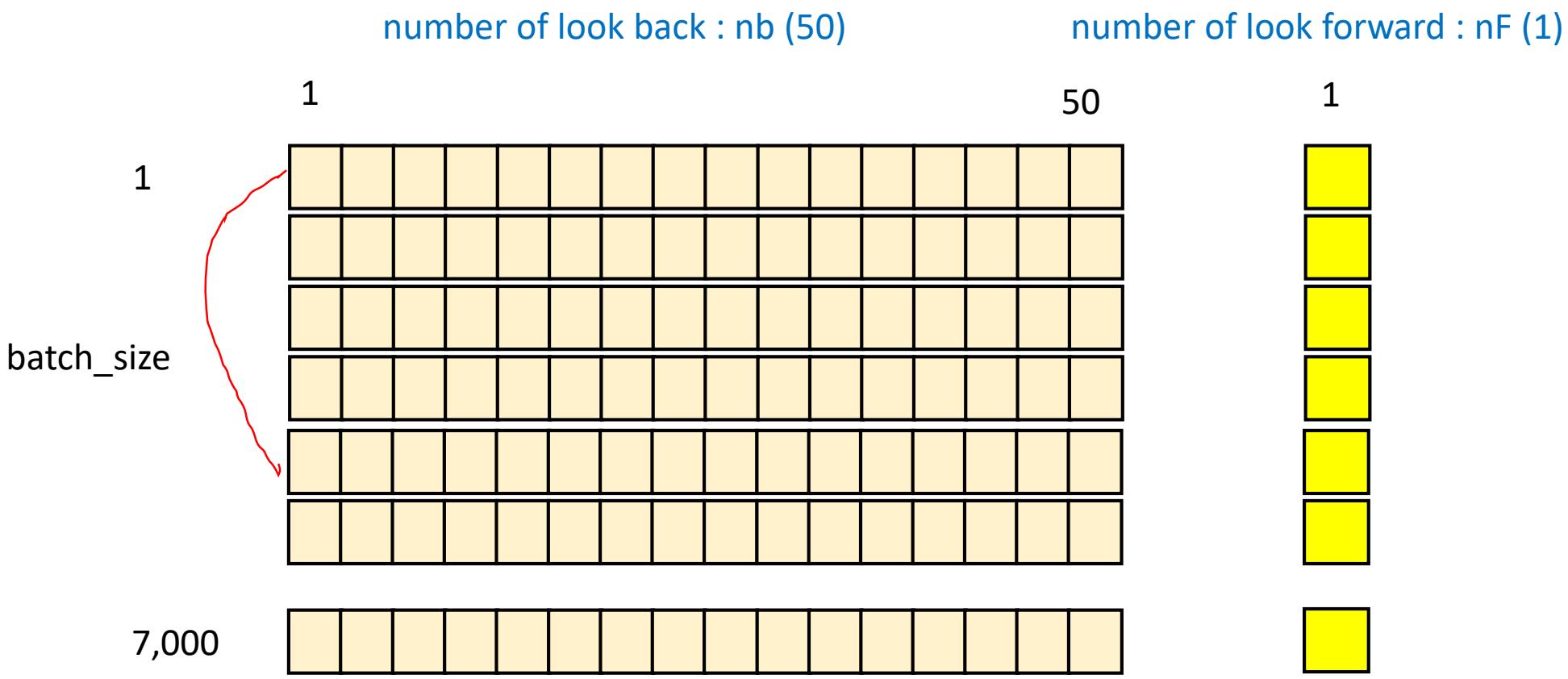


VDS 데이터를 이용한 교통 흐름 예측 모델 단계

- Data Preprocessing
- Building the RNN
- Making the prediction and visualization

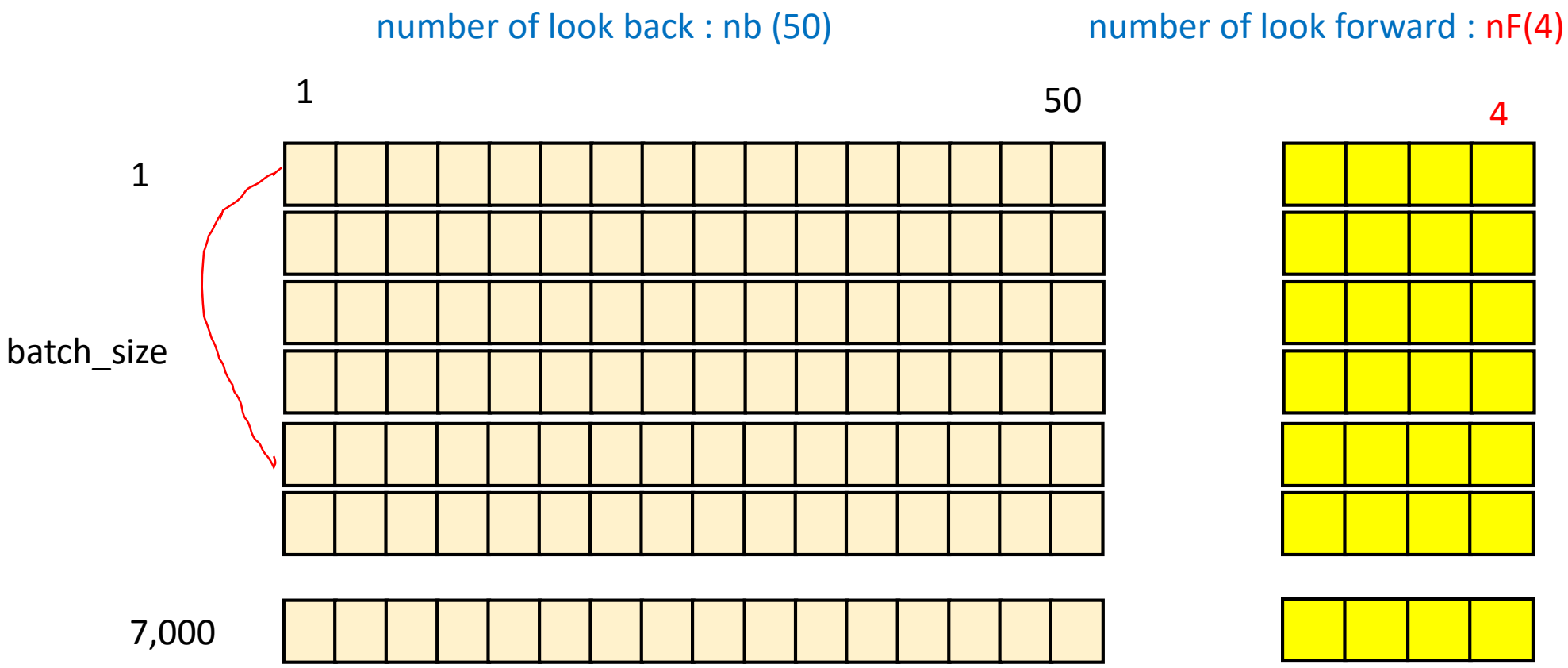
Many-to-One RNN Data Structure

X_train[7000,50,1] y_train[7000,1,1]

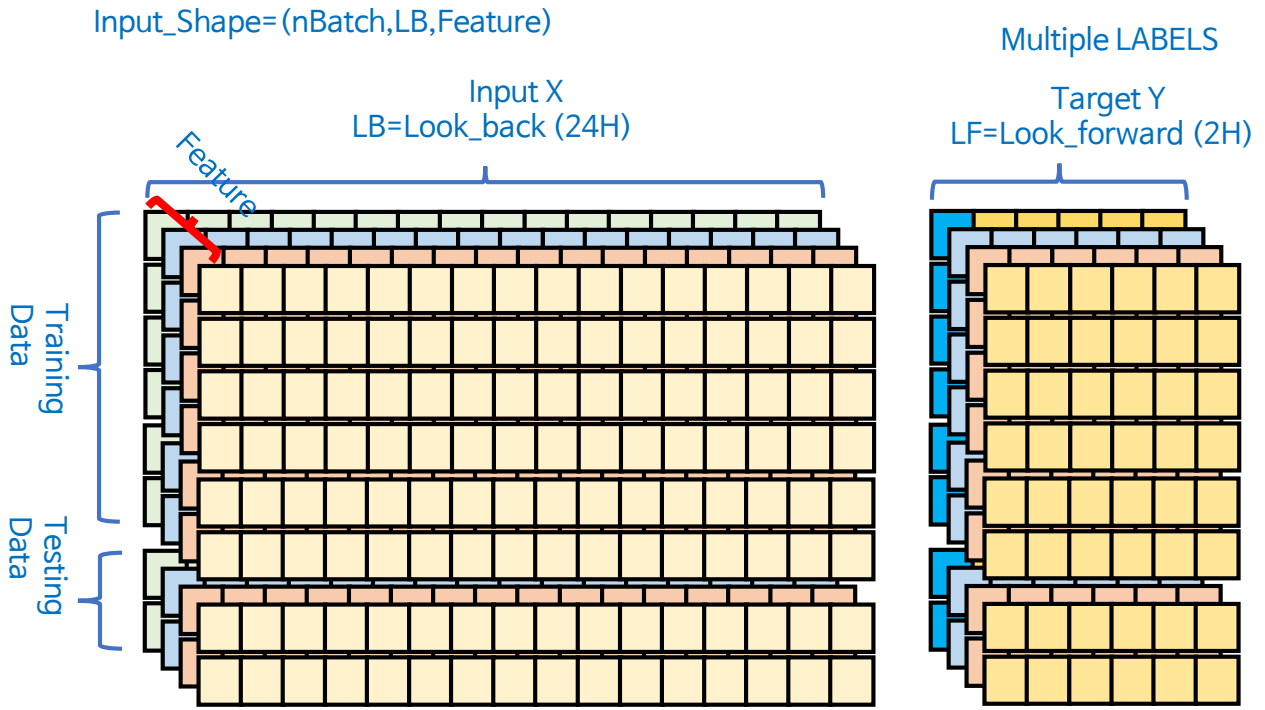


Many-to-Many RNN Data Structure

X_train[7000,50,1] y_train[7000,nF,1]



RNN Input-Output Data Structure



VDS, RSE Data Preprocessing for LSTM

교통 흐름 예측 (회귀) 실습

시계열성을 고려하지 않고 회귀를 해보자

데이터 가져오기

- 판다스를 이용하여 교통 데이터 가져오기

In [1]:
```
import tensorflow as tf

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

import pandas as pd
import numpy as np
```

교통 흐름 : 회귀 문제

In [6]:
```
df = pd.read_csv('./input/daejeon_vds16.csv')
df.head()

df['Date'] = pd.to_datetime(df['Date'])

df.head()

# set the index
df.set_index('Date', inplace=True)

df.head()
```

Out[6]:

	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
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

교통 흐름 : 회귀 문제

In [11]:
```
import matplotlib.pyplot as plt
```

In [12]:
```
X = df.drop(['Speed'],axis=1)
y = df['Speed']
```

In [13]:
```
y.head()

X.head()

# showing column wise %ge of NaN values they contains
for i in df.columns:
  print(i,"₩t-₩t", df[i].isna().mean()*100)
```

(A) 시계열성을 고려하지 않고 회귀를 해보자

속도(Speed)를 예측해봅시다.

In [16]:

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
X = pd.DataFrame(scaler.fit_transform(X), columns=X.columns)
X.head()

from sklearn.model_selection import train_test_split

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

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

X_train.shape
```

교통 흐름 : 회귀 문제

In [28]:
```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
X = pd.DataFrame(scaler.fit_transform(X), columns=X.columns)
X.head()
```

Out[28]:

	ToVol	SmVol	MeVol	LaVol	Occ.Rate
0	0.111446	0.129032	0.062069	0.0000	0.020398
1	0.117470	0.120968	0.089655	0.0000	0.019665
2	0.120482	0.129032	0.082759	0.0000	0.020032
3	0.117470	0.137097	0.062069	0.0000	0.018200
4	0.063253	0.044355	0.089655	0.0625	0.010871

In [29]:
```
Acc = []
```

A1: 시계열 패턴을 무시한 선형 회귀

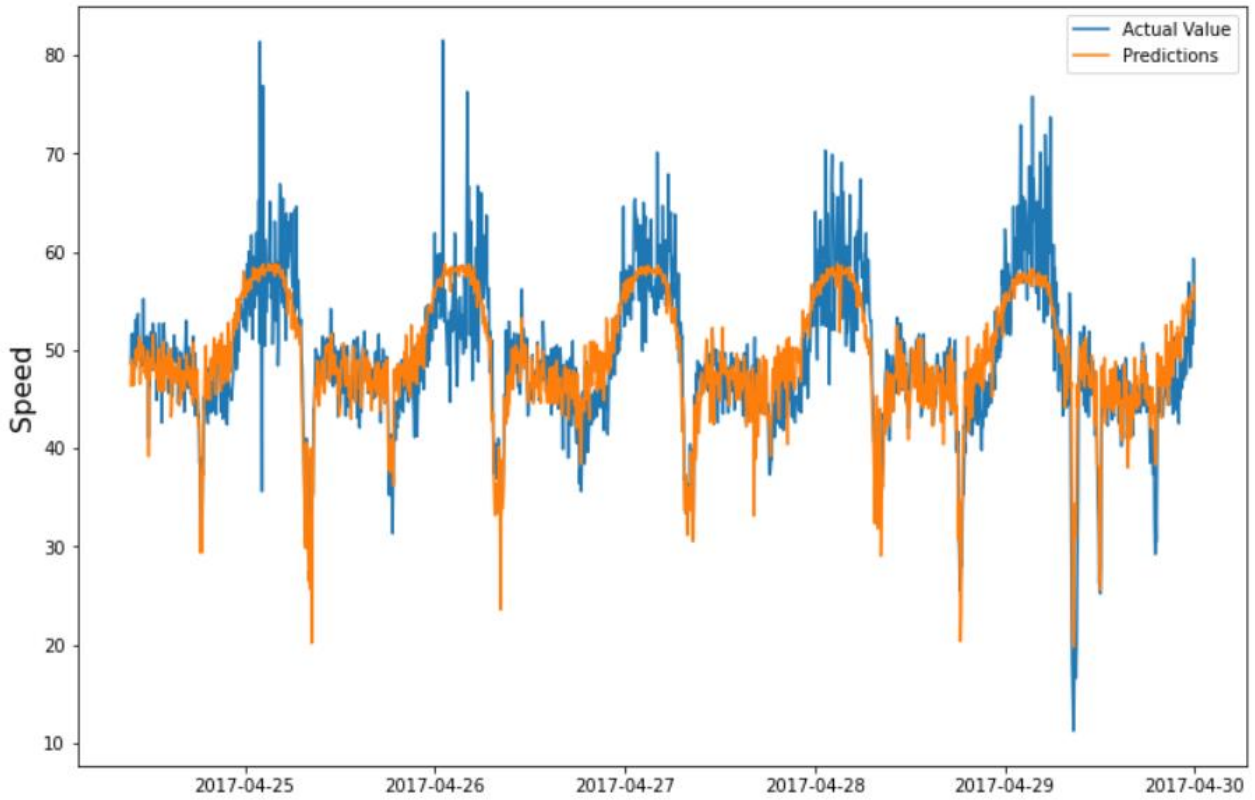
1. 선형회귀(Linear Regression)

In [30]:
```
from sklearn.linear_model import LinearRegression

model_1 = LinearRegression()
model_1.fit(X_train,y_train)

y_pred_1 = model_1.predict(X_test)

pred_df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred_1})
pred_df.head()

# Measure the Accuracy Score
from sklearn.metrics import r2_score

print("Accuracy score of the predictions: {0}".format(r2_score(y_test, y_pred_1)))
Acc.append(r2_score(y_test, y_pred_1))
```

Out[30]:
```
LinearRegression
LinearRegression()
```

A1: 시계열 패턴을 무시한 선형 회귀

In [34]:
```
plt.figure(figsize=(12,8))
plt.ylabel('Speed', fontsize=16)
plt.plot(pred_df)
plt.legend(['Actual Value', 'Predictions'])
plt.show()
```



A2: 시계열 패턴을 무시한 DNN 회귀

In [38]:

```
def reg_dnn(inp_dim):
    model = Sequential()
    model.add(Dense(20, input_dim=inp_dim, kernel_initializer='normal', activation='relu'))
    model.add(Dense(20, kernel_initializer='normal', activation='relu'))
    model.add(Dense(1, kernel_initializer='normal'))
    model.compile(loss='mean_squared_error', optimizer='adam')
    return model

# Model Training
model_2 = reg_dnn(inp_dim=X_train.shape[1])
history = model_2.fit(X_train, y_train, epochs=20, validation_split=0.2)

# Prediction
y_pred_2 = model_2.predict(X_test)
```

A2: 시계열 패턴을 무시한 DNN 회귀

In [45]:
```
pred_df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred_2.flatten()})
pred_df.head()
```

Out[45]:

Date	Actual	Predicted
2017-04-24 09:35:00	48.6	45.605270
2017-04-24 09:40:00	51.6	48.216091
2017-04-24 09:45:00	46.6	45.810997
2017-04-24 09:50:00	50.9	47.867794
2017-04-24 09:55:00	48.3	45.925426

A2: 시계열 패턴을 무시한 DNN 회귀

In [46]:

```
# Measure the Accuracy Score

from sklearn.metrics import r2_score

print("Accuracy score of the predictions: {0}".format(r2_score(y_test, y_pred_2)))
Acc.append(r2_score(y_test, y_pred_2))

plt.figure(figsize=(12,8))
plt.ylabel('Speed', fontsize=16)
plt.plot(pred_df)
plt.legend(['Actual Value', 'Predictions'])
plt.show()
```

```
Accuracy score of the predictions: 0.6904031309098866
```

3. 1D CNN 회귀

In [48]:
```
X_train = np.array(X_train).reshape(X_train.shape[0], X_train.shape[1], 1)
X_test = np.array(X_test).reshape(X_test.shape[0], X_test.shape[1], 1)
```

In [49]:
```
from tensorflow.keras import Sequential,utils
from tensorflow.keras.layers import Flatten, Dense, Conv1D, MaxPool1D, Dropout

def reg_cnn():
    model = Sequential()
    model.add(Conv1D(32, kernel_size=(3,), padding='same', activation='relu', input_shape = (X_train.shape[1],1)))
    model.add(Conv1D(64, kernel_size=(3,), padding='same', activation='relu'))
    model.add(Conv1D(128, kernel_size=(5,), padding='same', activation='relu'))
    model.add(Flatten())
    model.add(Dense(50, activation='relu'))
    model.add(Dense(20, activation='relu'))
    model.add(Dense(units = 1))

    model.compile(loss='mean_squared_error', optimizer='adam')
    return model
```

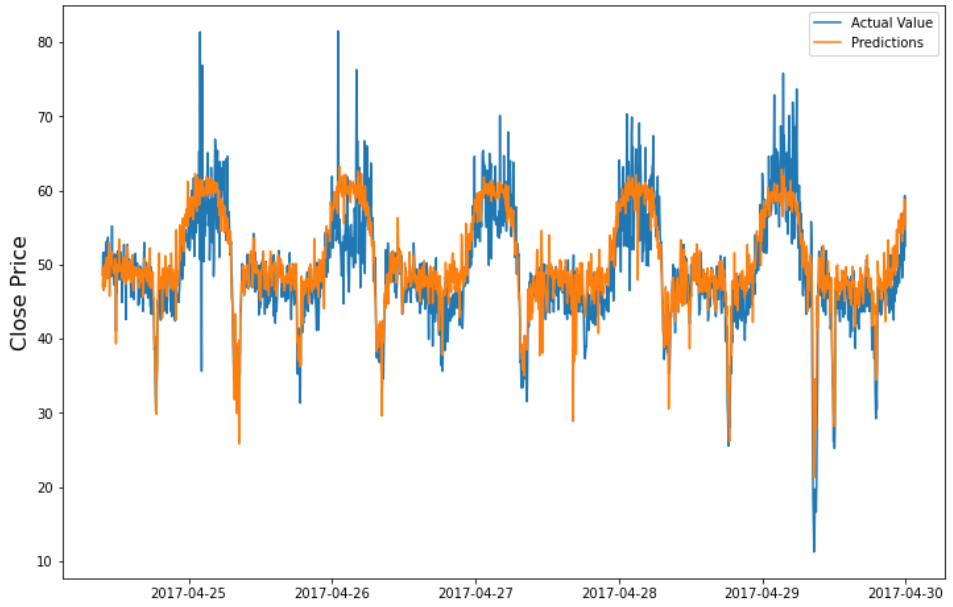
A3: 시계열 패턴을 무시한 1D CNN 회귀

In [50]:

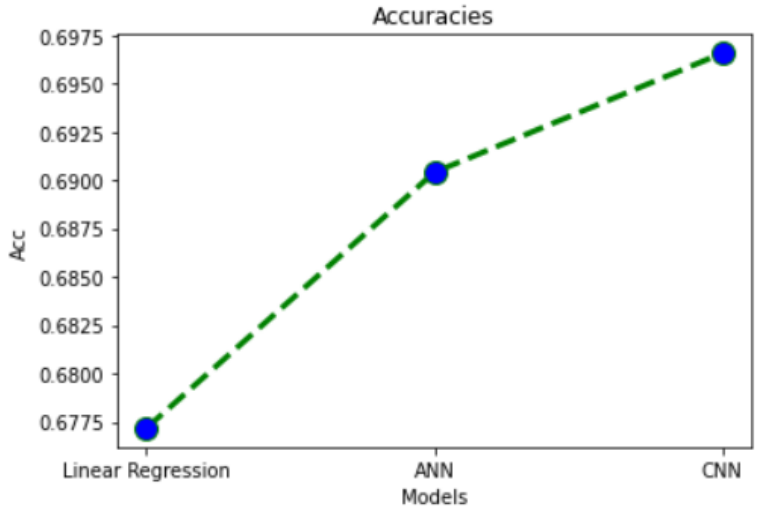
```
# Model Training
model_3 = reg_cnn()
history = model_3.fit(X_train, y_train, epochs=30, validation_split=0.2)

# Prediction
y_pred_3 = model_3.predict(X_test)
pred_df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred_3.flatten()})
pred_df.head()

# Measure the Accuracy Score
from sklearn.metrics import r2_score

print("Accuracy score of the predictions: {0}".format(r2_score(y_test, y_pred_3)))
Acc.append(r2_score(y_test, y_pred_3))
```

A3: 시계열 패턴을 무시한 1D CNN 회귀

In [54]:
```
plt.figure(figsize=(12,8))
plt.ylabel('Close Price', fontsize=16)
plt.plot(pred_df)
plt.legend(['Actual Value', 'Predictions'])
plt.show()
```



In [55]:
```
plt.plot(range(3), Acc, color='green', linestyle='dashed', linewidth = 3,
         marker='o', markerfacecolor='blue', markersize=12)
plt.ylabel('Acc')
plt.xlabel('Models')
plt.title("Accuracies")
plt.xticks(range(3), ['Linear Regression', 'ANN', 'CNN'])
plt.show()
```



KISTI SINCE 1962

2022 TRUST KISTI

Korea Institute of Science and Technology Information