KiSTi 한국과학기술정보연구원
Korea Institute of Science and Technology Information

2023

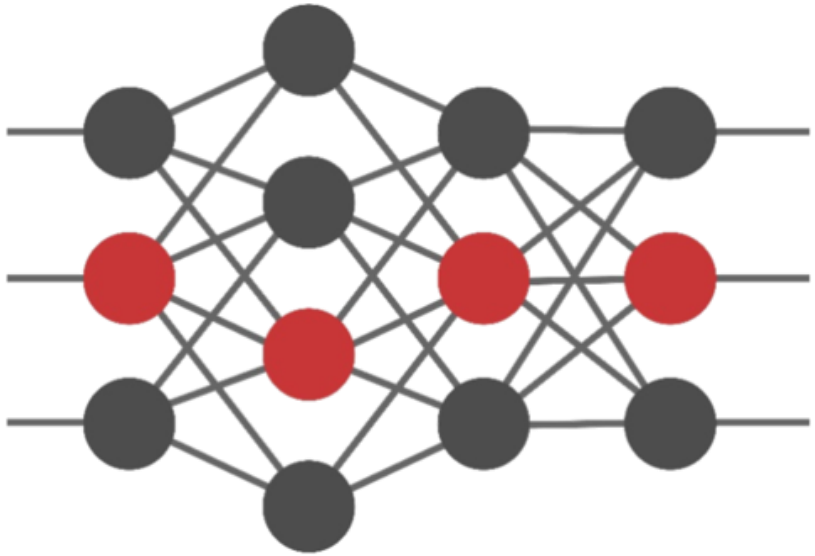
인공지능특론
(Advanced Artificial Intelligence)
11주 신경망 소개

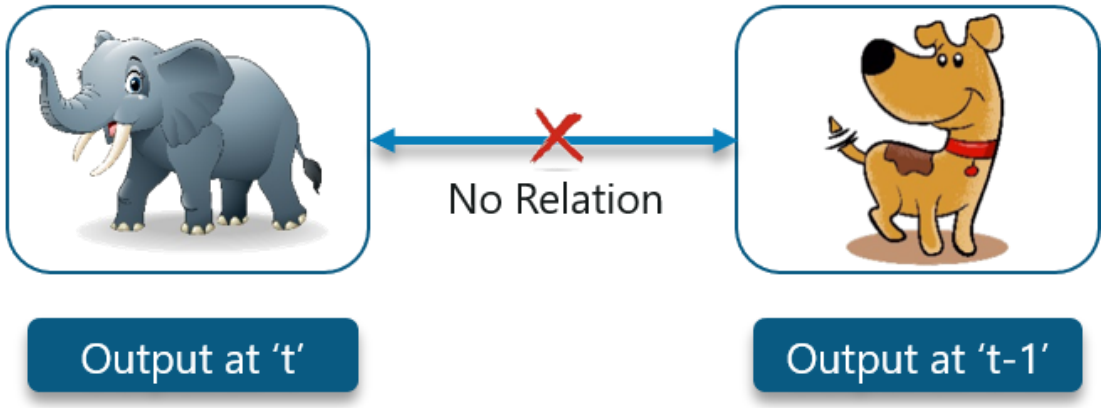
Hongsuk Yi KISTI

10
시계열 데이터 기반 RNN 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/

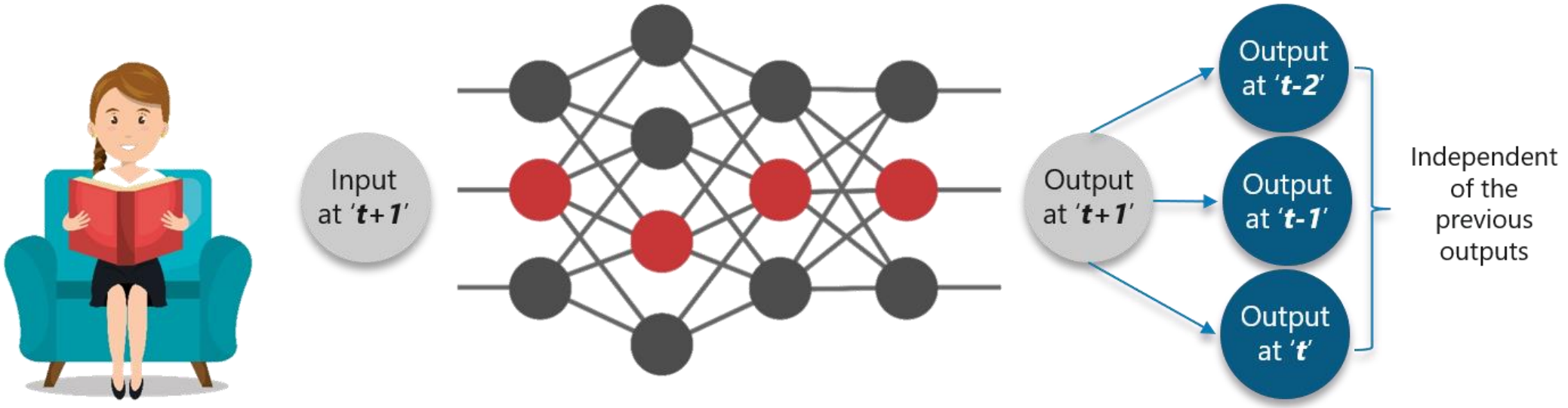
Why Not Feedforward Networks

❖ **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.



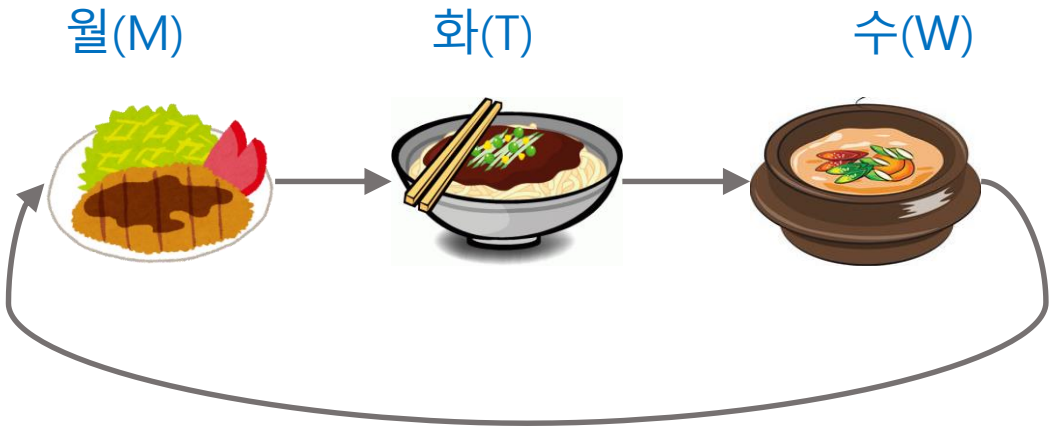
간단한 순환신경망(RNN)

❖ 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.



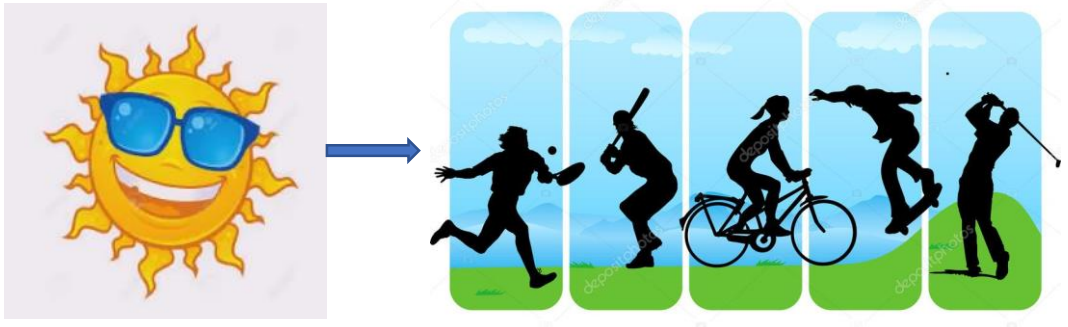
RNN = Recurrent Neural Network

간단한 순환신경망(RNN)

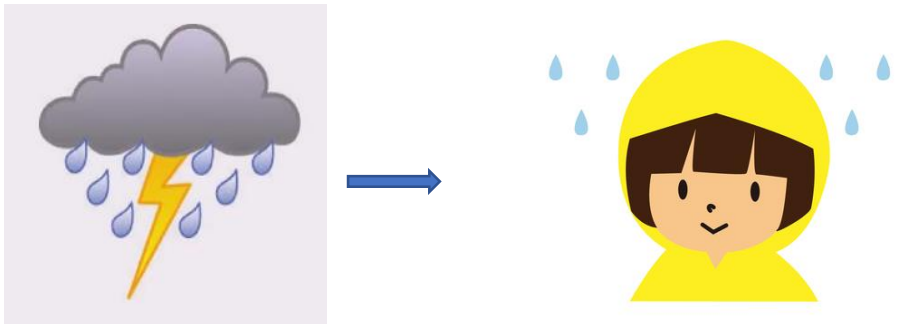
❖ 실제로 룸메이트는 날씨에 따라서 식사 메뉴에 규칙을 가지고 있다.

✓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

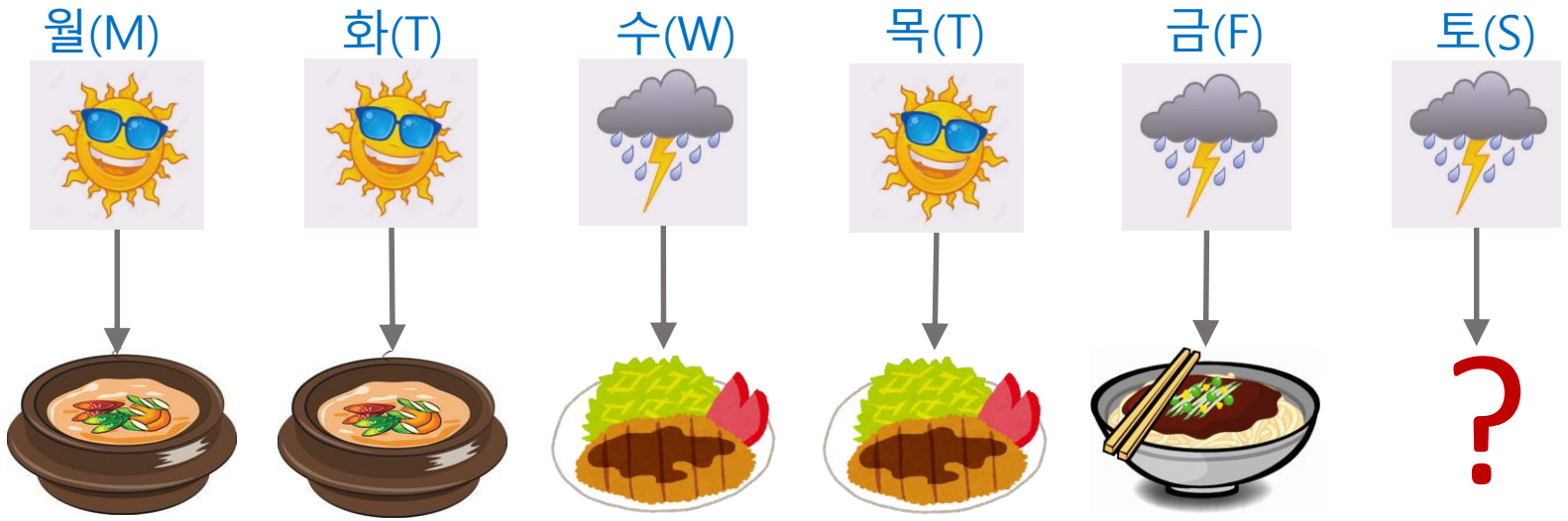


원래 RNN 패턴

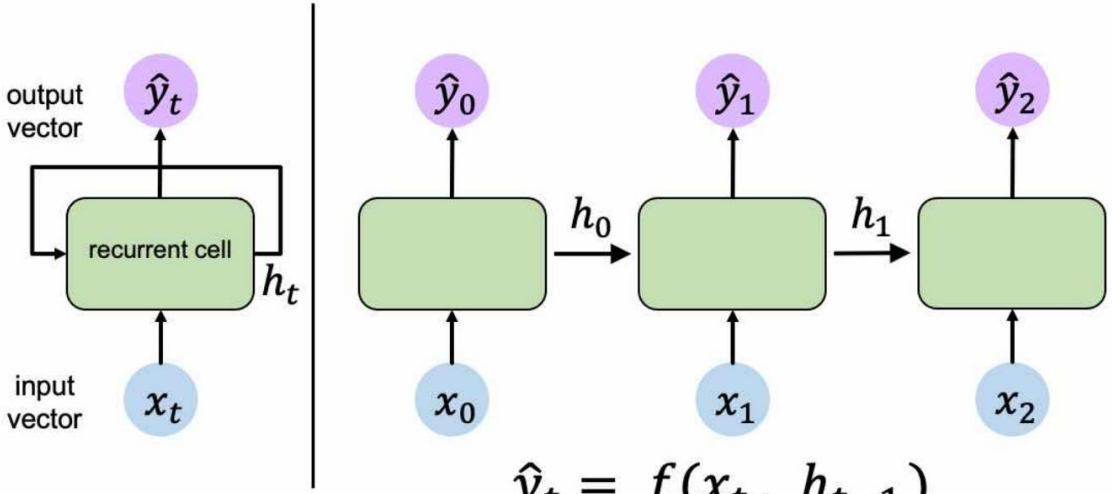
RNN depending on the weather

❖ 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?



Neurons with Recurrence



memory cell

Output Vector

$$\hat{y}_t = W_{hy}^T h_t$$

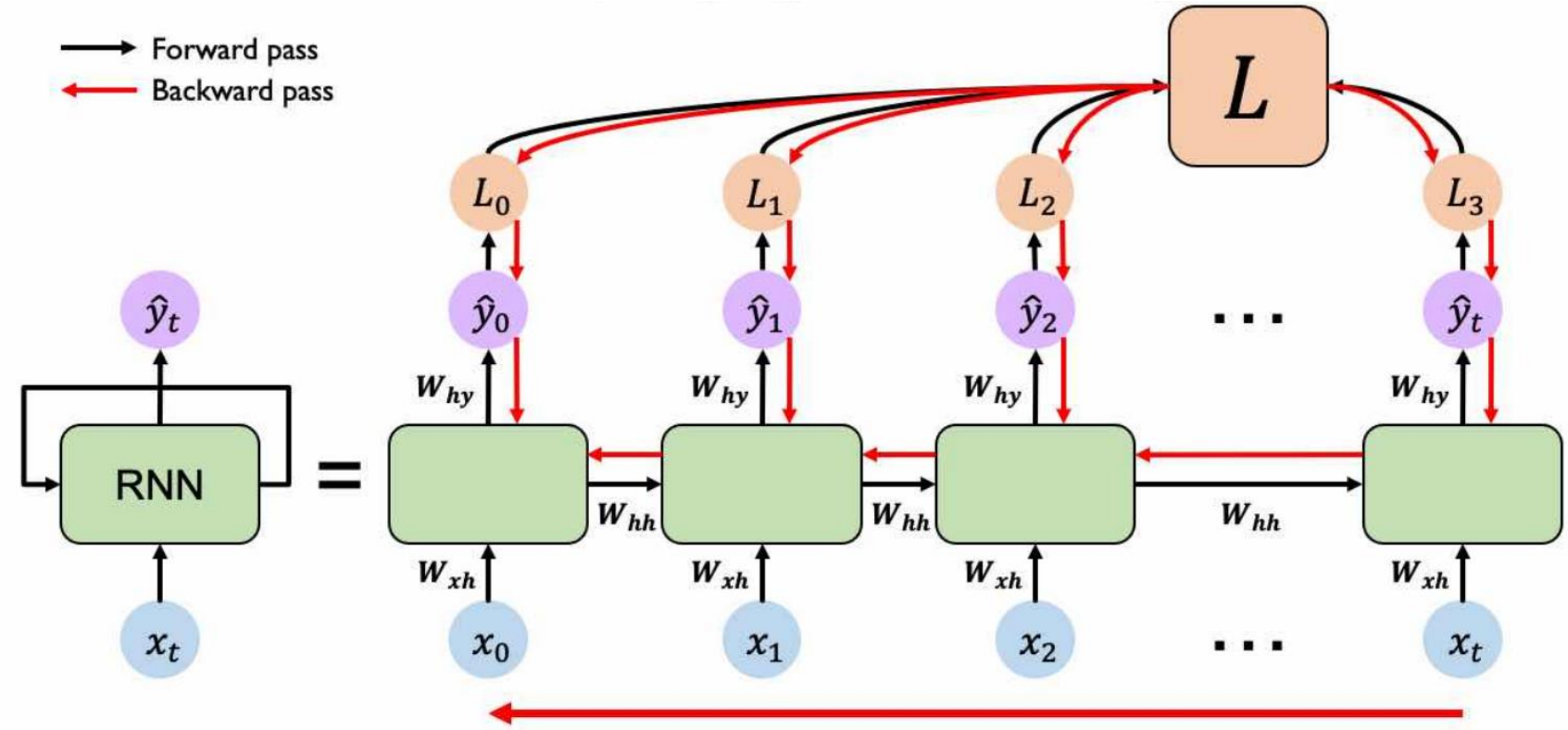
Update Hidden State

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

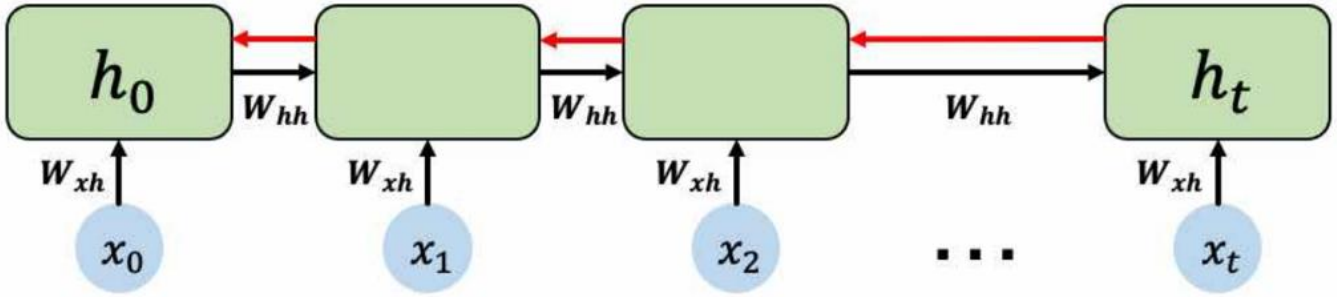
Input Vector

$$x_t$$

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:

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

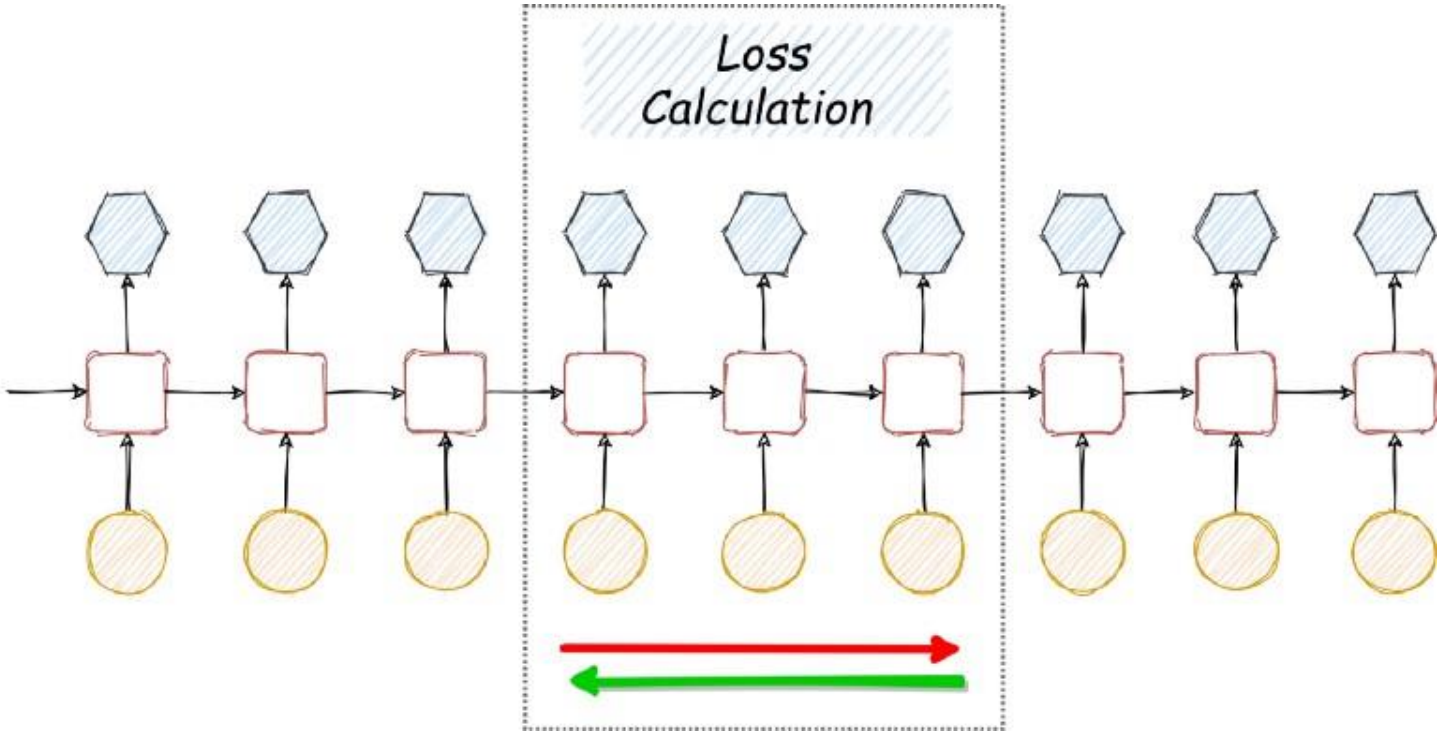
1. ***Vanishing gradient*** $\left\| \frac{\partial h_i}{\partial h_{i-1}} \right\|_2 < 1$
2. ***Exploding gradient*** $\left\| \frac{\partial h_i}{\partial h_{i-1}} \right\|_2 > 1$

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

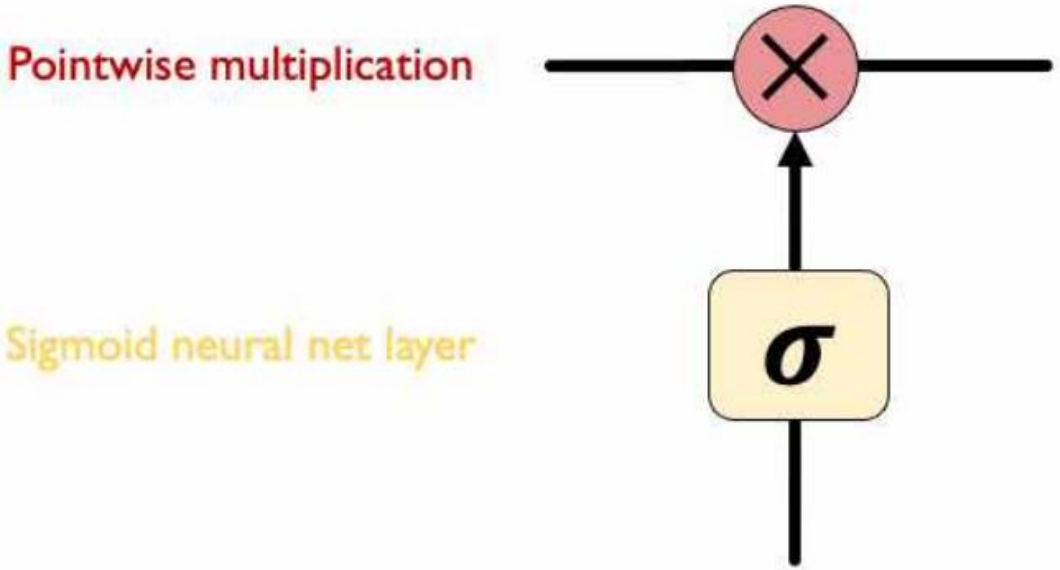
Truncated BPTT trick tries to overcome the VGP by considering a moving window through the training process



13
LSTM로 교통 흐름 예측

Trick : Gated Cells

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



gated cell

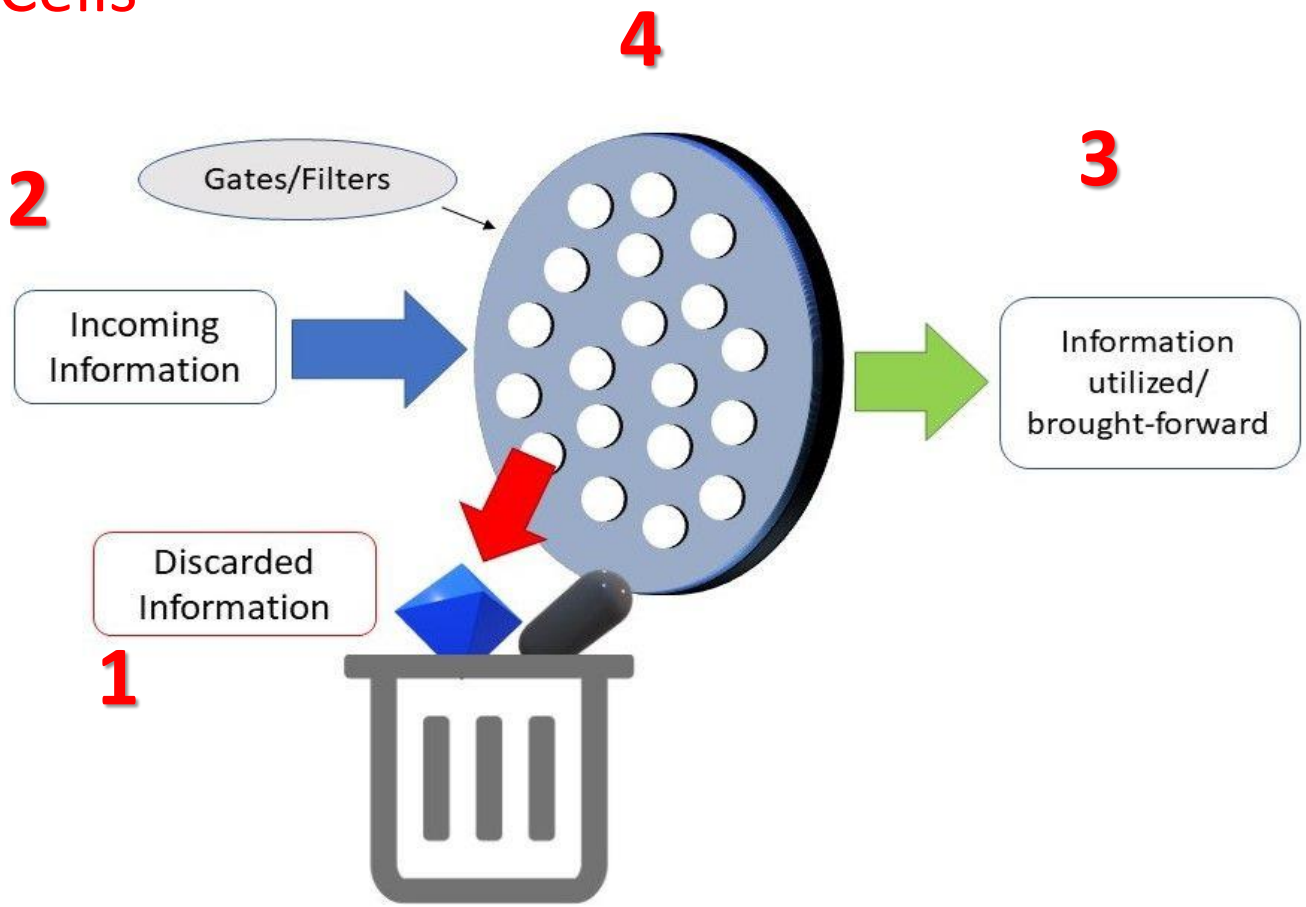
LSTM, GRU, etc.

Gates optionally let information through the cell

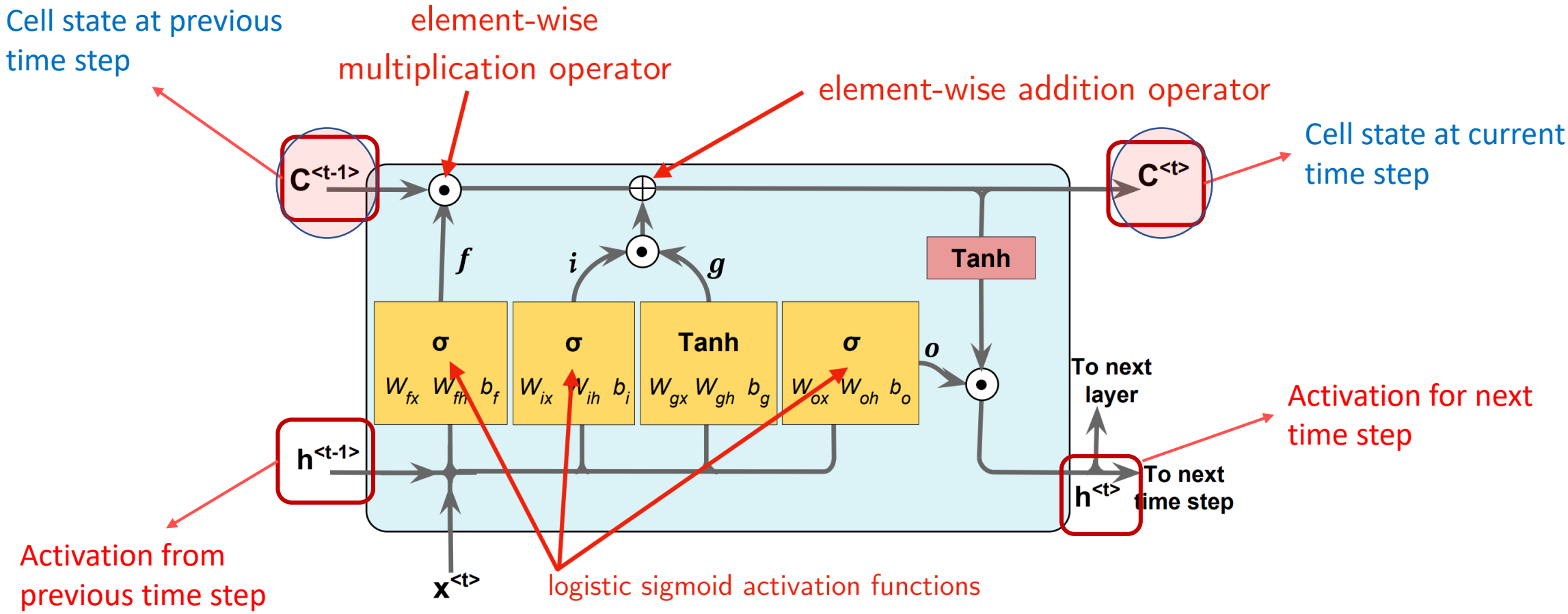
Long Short Term Memory (LSTMs) networks rely on a gated cell to track information throughout many time steps.

Gates control the flow of information to/from the memory

Trick #3: Gated Cells



Long-short term memory (LSTM) : Cell state



핵심 사항 정리

❖ 데이터 전처리

✓시계열 데이터의 과거 기억의 길이를 windows 한다면, 그 길이는 중요한 파라미터이다.
- windows=12 (1시간)을 권장한다.

✓RNN의 한계점을 극복하고 LSTM의 장점을 반영하기 위해서는 긴 기억 계산을 시도해 보자
- windows=144 (12시간)을 설정하면 컴퓨터 계산 시간이 많이 필요로 한다.

✓Many-to-One 형태의 응용 모델로 설계하자.
- 과거 기억으로 바로 다음 순간(미래) 값을 예측한다.

✓단항(Uni-variant) 값을 사용한ㅏ.
- 여기서는 속도만을 고려한다.

❖ LSTM 모델 설정 및 모델 이해

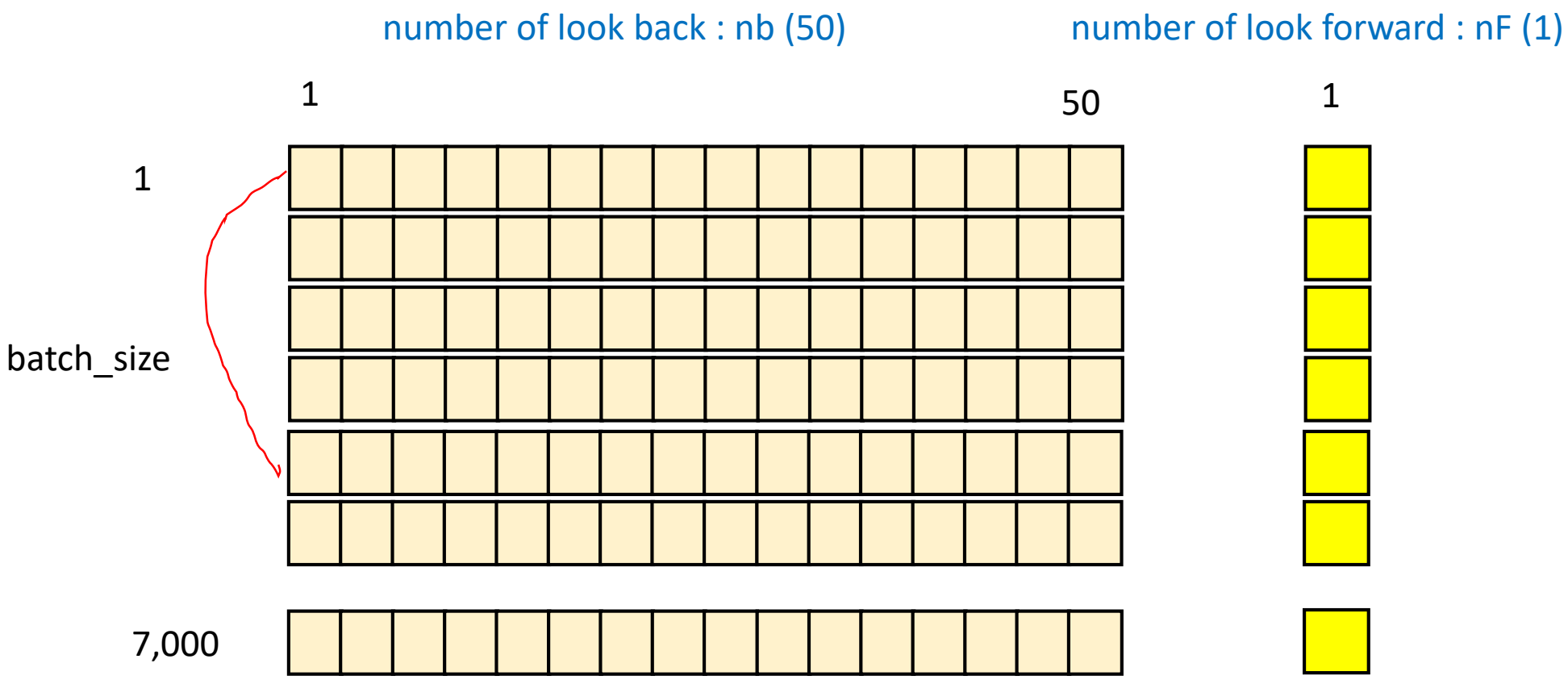
✓return_sequences=True는 Many-to-Many 형태에서 이전 Layer의 정보를 다음 층에 전달한다.
- 웨이트가 초기화할 필요없이 유지된다.

✓디폴트 활성함수는 tanh이다. relu도 사용할 수 있지만 GPU에서 Blas를 이용한 고속 처리는 안 된다.

✓뉴런에 대하여
- 각각 LSTM에서 기억의 길이마다 뉴런 개수를 정할 수 있다. (여기서는 60으로 설정함)
- 출력층은 1개의 뉴런을 갖는다. (Many-to-One 문제임)

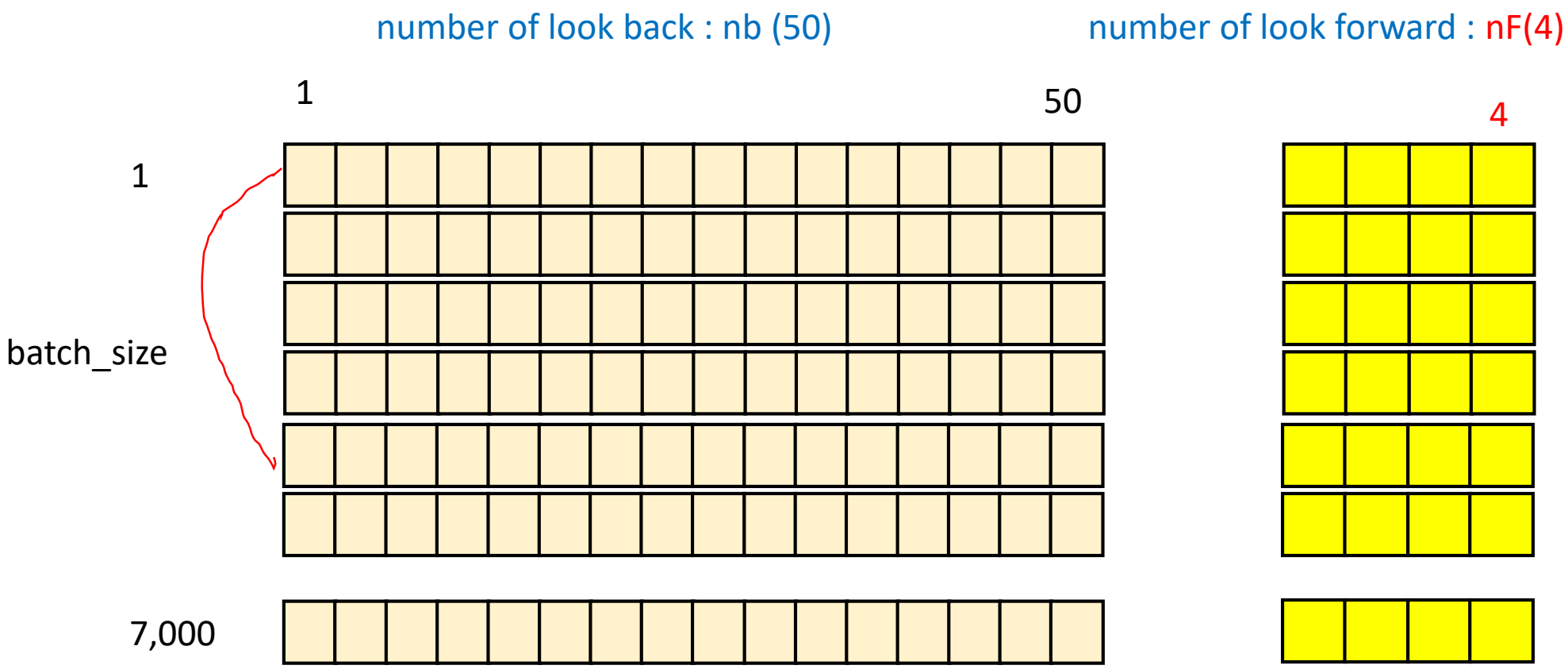
Many-to-One RNN Data Structure

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

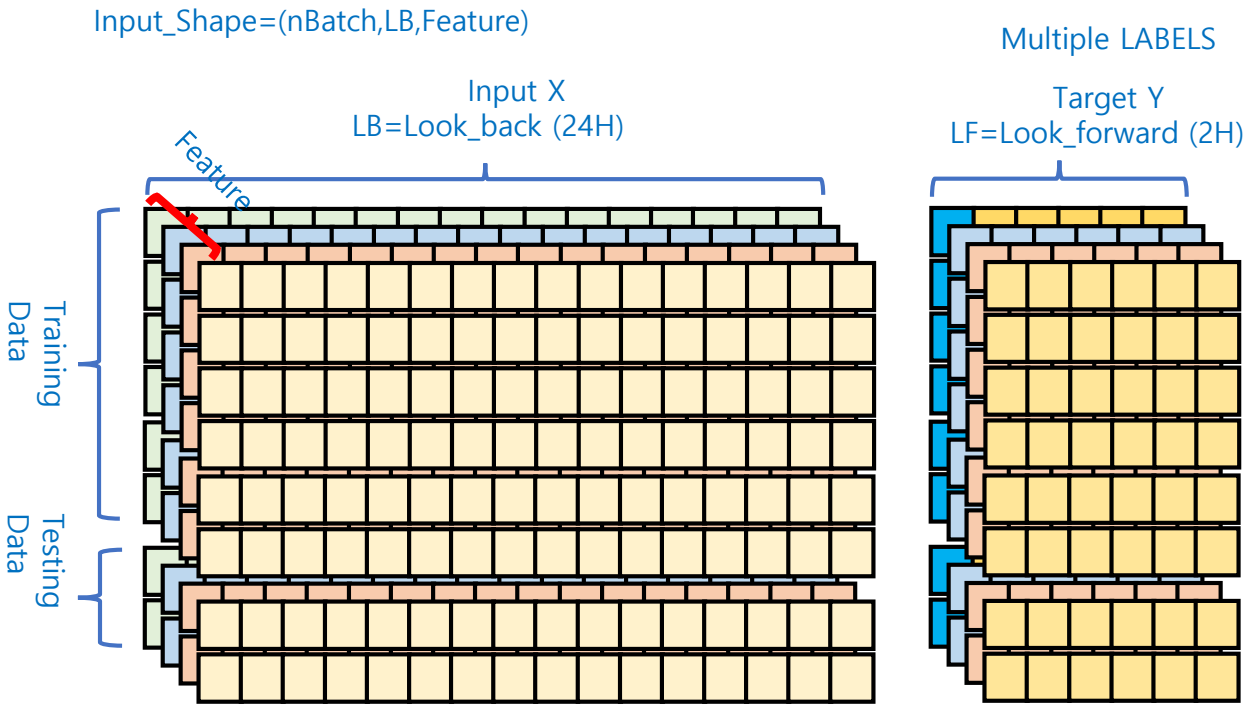


Many-to-Many RNN Data Structure

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



RNN Input-Output Data Structure



VDS, RSE Data Preprocessing for LSTM

RNN으로 교통흐름 예측 실습

- VDS 데이터, SimpleRNN

VDS 데이터 다운로드

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

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [3]:
```
df = pd.read_csv('https://raw.githubusercontent.com/hongsukyi/Lectures/main/vds/vds.csv')
df.head(2)
```

Out[3]:

	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

차량 속도를 입력으로 사용

In [4]:
```
df_uni = df.iloc[:,5:6].values
```

In [5]:
```
df_uni
```

Out[5]:
```
array([[50.3],
       [58.9],
       [50.6],
       ...,
       [50.6],
       [59.3],
       [52.5]])
```

In [6]:
```
print(df_uni.shape)
```

```
(8064, 1)
```

입력을 정규화

In [7]:
```
from sklearn.preprocessing import MinMaxScaler

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

scaled_df
```

Out[7]:
```
array([[0.52350699],
       [0.63278272],
       [0.52731893],
       ...,
       [0.52731893],
       [0.63786531],
       [0.55146125]])
```

RNN 데이터 구조 만들기

In [8]:
```
look_back = 144
```

In [9]:
```
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)
```

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

```
(7919, 144, 1) (7919, 1)
```

훈련과 테스트 데이터 나누기

In [11]:
```
from sklearn.model_selection import train_test_split

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

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

```
(6335, 144, 1) (6335, 1)
(1584, 144, 1) (1584, 1)
```

In [12]:
```
neuron_1 = 32
neuron_2 = 16
```

간단한 RNN 모델 아키텍처 : SimpleRNN

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

In [14]:
```
def simple_rnn():
    model = Sequential([
        SimpleRNN(neuron_1, input_shape=(X_train.shape[1],1)),
        Dense(neuron_2),
        Dense(1)
    ])

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

모델 파리미터 계산

In [15]:

```
model = simple_rnn()
model.summary()
```

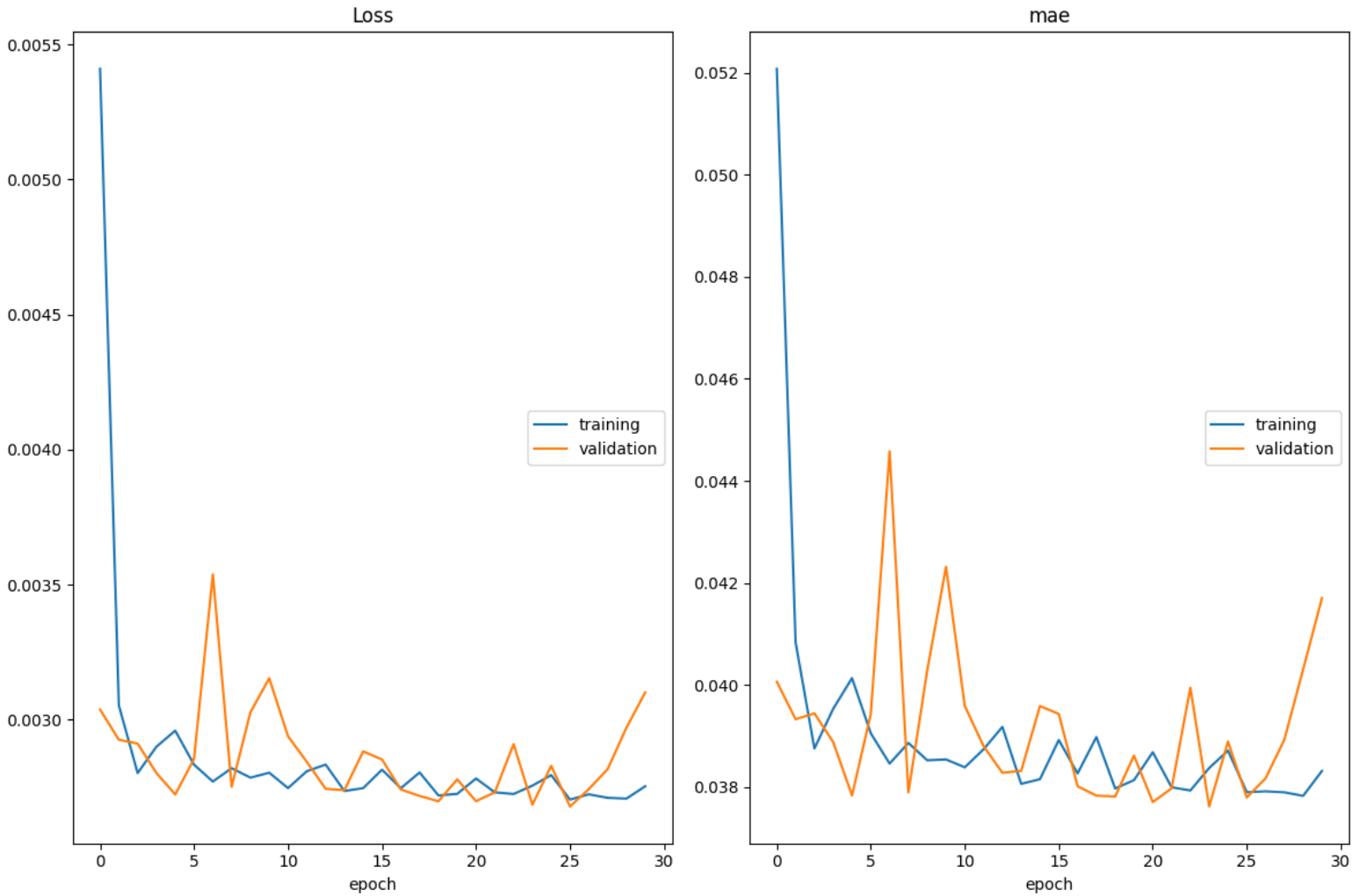
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 simple_rnn (SimpleRNN)      (None, 32)                1088

 dense (Dense)               (None, 16)                528

 dense_1 (Dense)             (None, 1)                 17

=================================================================
Total params: 1,633
Trainable params: 1,633
Non-trainable params: 0
_________________________________________________________________
```

모델 훈련과 실시간 가시화

In [16]:
```
# !pip install livelossplot
from livelossplot import PlotLossesKeras
```

In [17]:
```
history = model.fit(X_train,y_train, epochs=30,
                    validation_split=0.1,
                    batch_size=32,
                    callbacks=[PlotLossesKeras()])
```

PlotLossesKeras 결과



훈련과정 손실값

```
Loss
        training                (min:    0.003, max:    0.005, cur:    0.003)
        validation              (min:    0.003, max:    0.004, cur:    0.003)
mae
        training                (min:    0.038, max:    0.052, cur:    0.038)
        validation              (min:    0.038, max:    0.045, cur:    0.042)
179/179 [==============================] - 6s 32ms/step - loss: 0.0028 - mae: 0.0383 - val_loss: 0.0031 - val_mae: 0.0417
```

숙제 제출 할 때 사용할 것

In [17]:

```
mse, mae = model.evaluate(X_test, y_test, verbose=2)

print("\nlook_back={:3d},  MSE={:8.6f}".format(look_back,mse))
```

```
50/50 - 1s - loss: 0.0028 - mae: 0.0376 - 514ms/epoch - 10ms/step

look_back=144,  MSE=0.002782
```

테스트 데이터 활용한 예측

KISTI SINCE 1962

In [18]:
```
X_test.shape
```

Out[18]: (1584, 144, 1)

In [19]:
```
# Prediction
y_pred = model.predict(X_test)
```

```
50/50 [==============================] - 1s 11ms/step
```

In [20]:
```
y_pred.shape
```

Out[20]: (1584, 1)

In [21]:
```
y_test.shape
```

Out[21]: (1584, 1)

예측값 대 실제값 비교

In [22]:
```
pred_df = pd.DataFrame({'Actual': y_test.flatten(), 'Predicted': y_pred.flatten()})
pred_df.head()
```

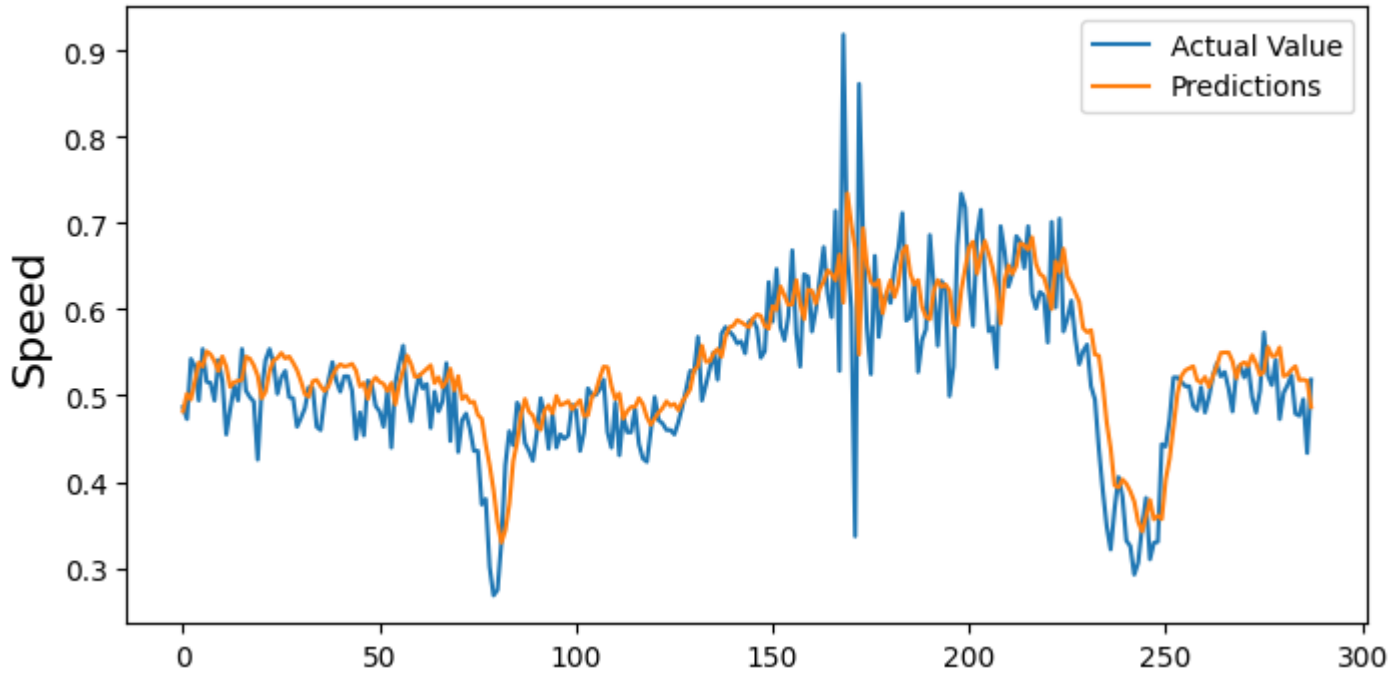
Out[22]:

	Actual	Predicted
0	0.486658	0.481714
1	0.472681	0.502077
2	0.542567	0.495385
3	0.533672	0.517115
4	0.494282	0.538158

정규화된 차량 속도 예측

In [23]:

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



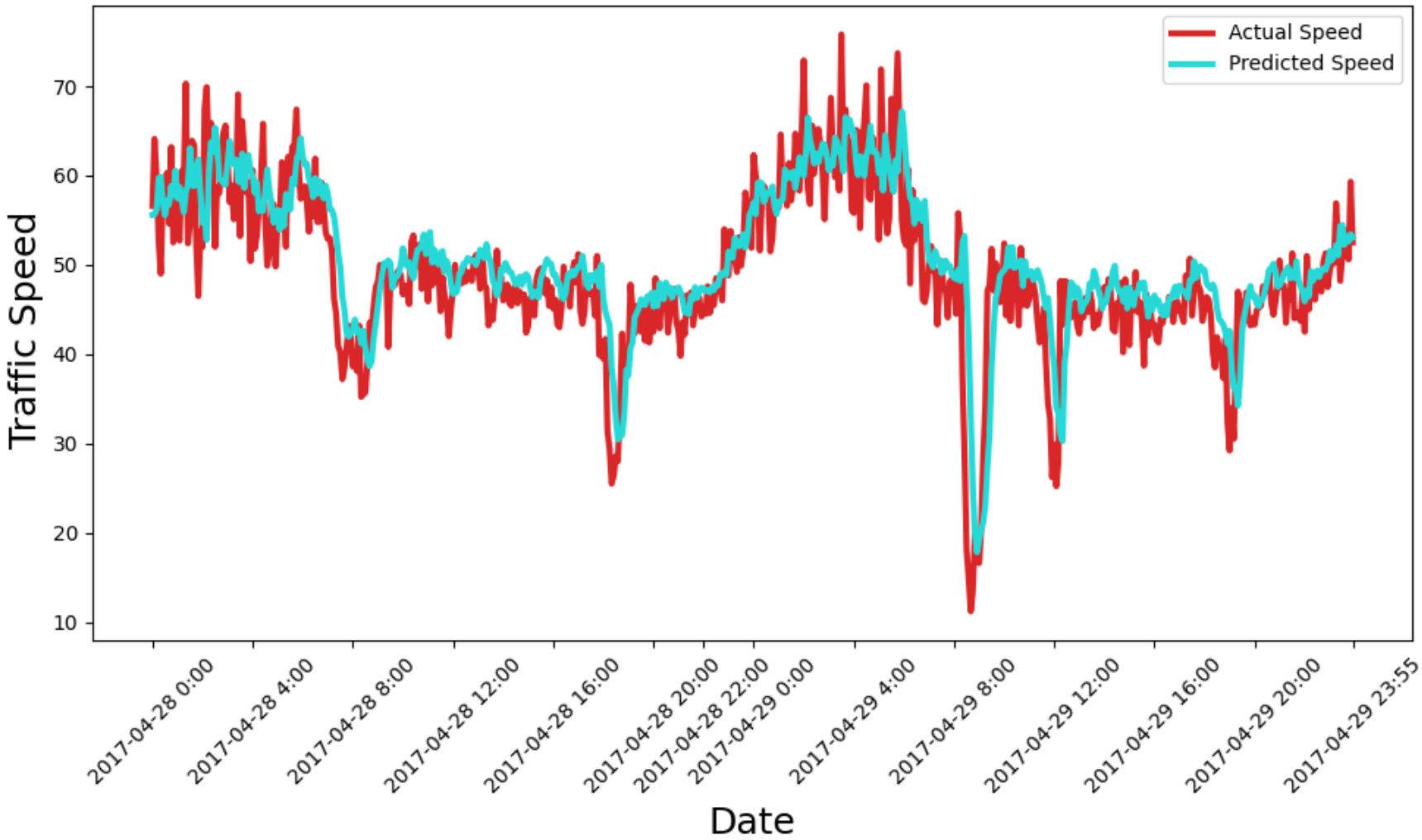
차량속도 원래 값으로 예측

In [24]:
```
y_pred = model.predict(X_test)
y_pred = scaler.inverse_transform(y_pred)
```

```
50/50 [==============================] - 1s 10ms/step
```

In [25]:
```
plt.figure(figsize=(15, 8))
#plt.style.use("dark_background")

plt.plot(df["Date"].iloc[-576:], df_uni[-576:], linewidth= 5,  color='#d92628', label='Actual Speed')
plt.plot(y_pred[-576:], color='#26D9D7', linewidth= 5, label='Predicted Speed')
plt.xlabel("Date", size="18")
plt.ylabel("Traffic Speed ", size="18")
plt.xticks([0, 24, 48,72, 96,120, 144, 168, 192,216,240,264, 287],rotation=45)

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

실제값으로 차량 속도 예측 (2일)



원래 속도로 환원

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

```
(1584, 1) (1584, 1)
```

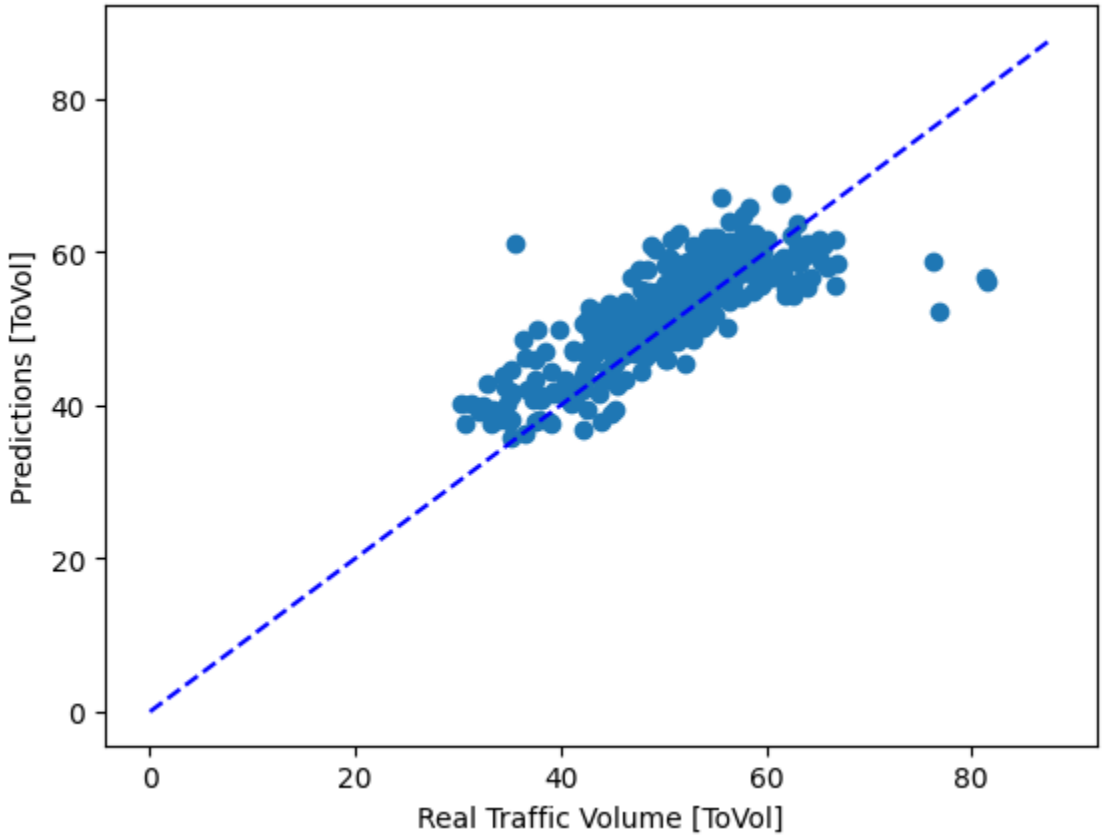
```
y_pred = scaler.inverse_transform(y_pred)
y_test = scaler.inverse_transform(y_test)
```

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

	Actual	Predicted
0	47.4	47.543636

실제값과 예측 오차

```
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--')
```



숙제

❖ SimpleRNN 하이퍼파라미터 최적화

✓ 속도(Speed)와 교통량에 대하여 아래 표를 작성하세요

- Look_back은 수정하지 말것
- 뉴런 개수와 은닉층은 수정 가능함

✓가장 작은 MSE 값을 제출하고 그때 파이미터 값도 제시함

항목	Look_back	뉴런	은닉층	MSE
Speed	12			
Speed	288			
ToVol	288			

```
print("\nlook_back={:3d},  MSE={:8.6f}".format(look_back,mse))
```

```
look_back=144,  MSE=0.002782
```

KISTI SINCE 1962

2023 TRUST KISTI

Korea Institute of Science and Technology Information