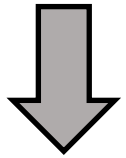
RNN
(Recurrent Neural Network)

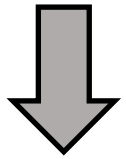
제2차 AI 겨울

이제 MLP도 해결했다.
(전문가 시스템에서) 모든 문제가 해결할 것 처럼 보였는데...



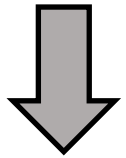
전문 지식이 없으면 못하나?

많은 데이터로 전문가를 만들자!



데이터가 없는데?

인터넷 상에 엄청 많아!!



계산량이 엄청날 텐데?

GPU 있잖아!!!

양질의 데이터가 많은 있는 임무들

단일 사진으로 판단 가능

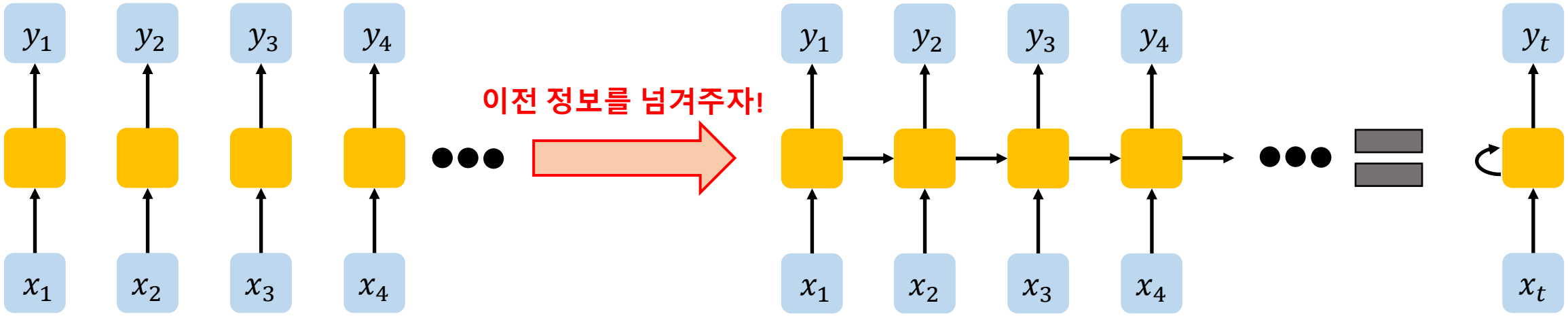
한 단어로 다음 단어 판단 불가능

Donald J. Trump
@realDonaldTrump

China has been hit much harder than the USA, not even close. They, and many other nations, have treated us unsustainably badly. We have been the dumb and helpless "whipping post," but not any longer. We are bringing back jobs and businesses like never before. Already, more than FIVE TRILLION DOLLARS OF INVESTMENT, and rising fast! THIS IS AN ECONOMIC REVOLUTION, AND WE WILL WIN. HANG TOUGH, it won't be easy, but the end result will be historic. We will, MAKE AMERICA GREAT AGAIN!!!

8.41k ReTruths **37.6k** Likes Apr 05, 2025, 9:34 PM

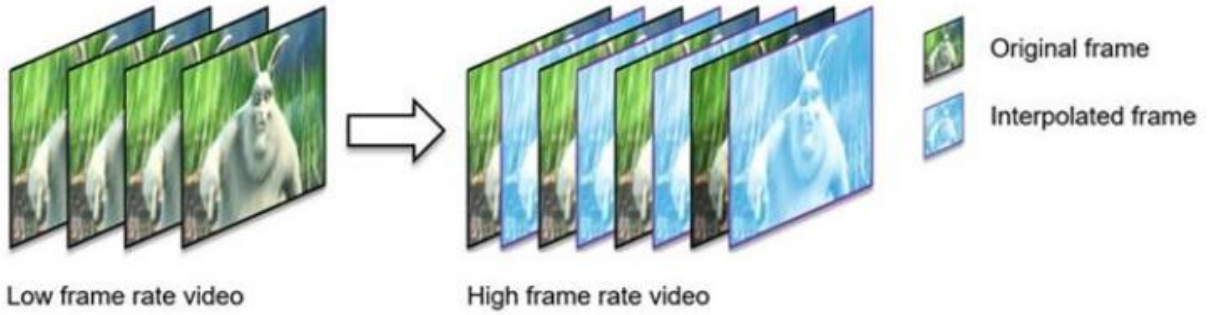
어떻게 해결할까?



이런 경우는?

(a) Masked-Image (b) Inpainted-Image

Inpainting

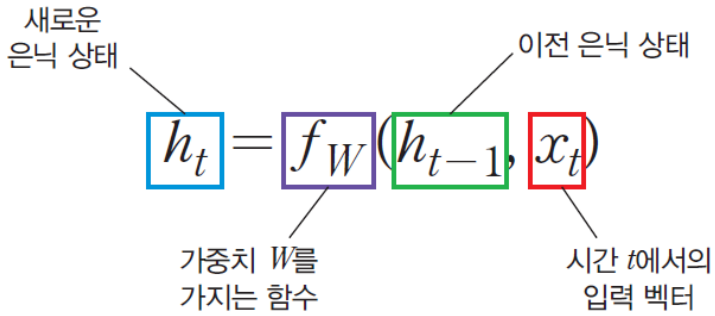


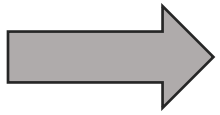
Frame Rate Up Conversion

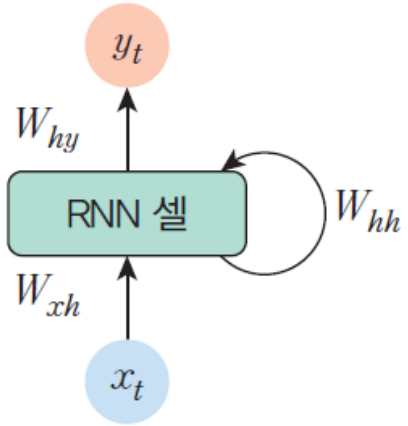
이미지라도 항상 CNN만을 하는 것은 아니다!

➔ 업무에 따라서 적절하게 선택해서 사용해야 한다!!

RNN의 동작







- 입력 벡터: x_t
- 출력 벡터: $y_t = f(W_{hy}h_t)$
- 은닉 상태: $h_t = \tanh(W_{xh}x_t + W_{hh}h_{t-1})$

RNN의 동작

샘플 번호	x(입력)	y(정답)
1	[16, 8, 32]	[9]
2	[8, 32, 9]	[23]
3	[32, 9, 23]	[11]
…	…	…
1092	[22, 15, 7]	[18]

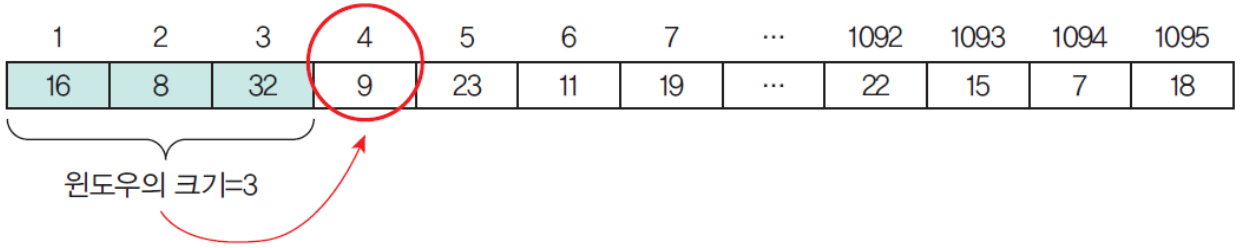
1	2	3	4	5	6	7	…	1092	1093	1094	1095
16	8	32	9	23	11	19	…	22	15	7	18

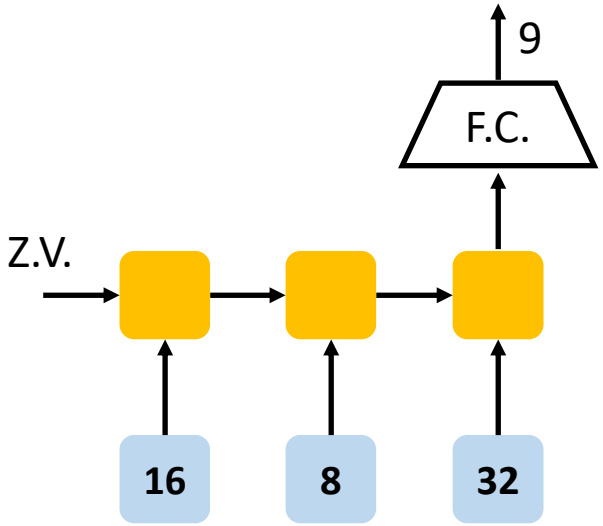
어떻게 예측할까: 이전 모든 데이터를 볼까?

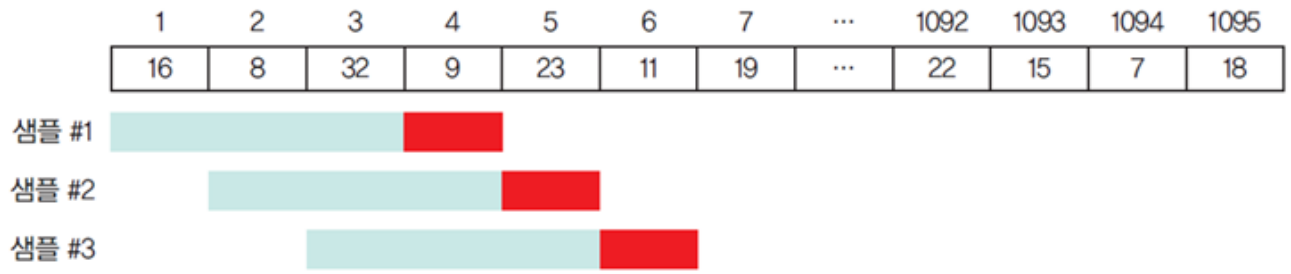
1. 관련성 높은 최근 데이터만 보자
2. 각 데이터마다 시간적 패턴이나 관계도 학습 가능

➔ **윈도우 사이즈 (최근 몇 개)내에서만 관계를 학습하자!**

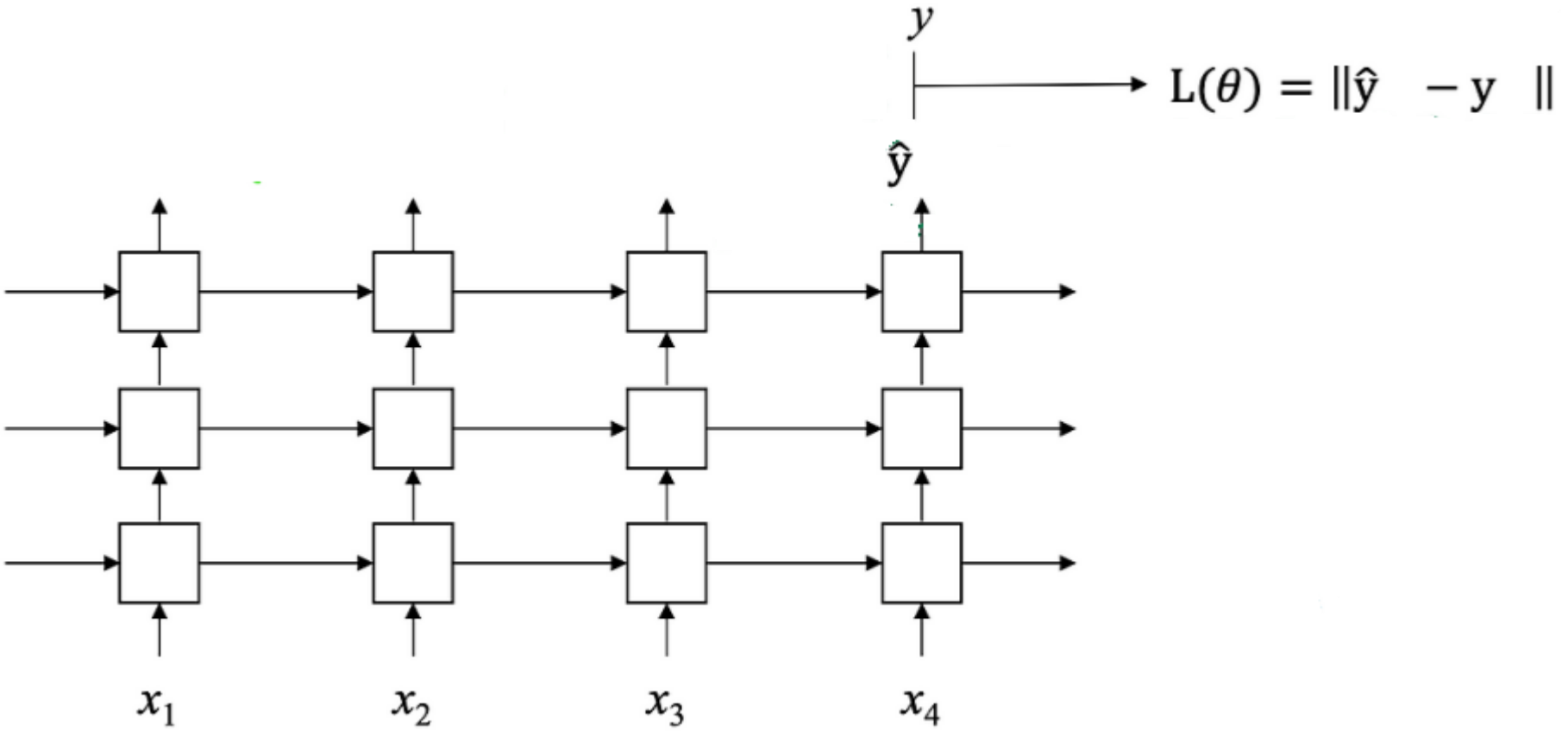
RNN의 동작


1 2 3 4 5 6 7 ··· 1092 1093 1094 1095
16 8 32 9 23 11 19 ··· 22 15 7 18
윈도우의 크기=3


9
F.C.
Z.V.
16
8
32


1 2 3 4 5 6 7 ··· 1092 1093 1094 1095
16 8 32 9 23 11 19 ··· 22 15 7 18
샘플 #1
샘플 #2
샘플 #3

RNN의 동작 – Multi-layered



예제 – S&P500 종가 예측

```
# 0: 필요한 패키지, 모듈 읽어오기
import torch
import torch.nn as nn
from torch.utils.data import TensorDataset, DataLoader
import yfinance as yf                                    # Yahoo!Finance
import numpy as np
# 사이킷런(scikit-learn) 라이브러리
### 머신러닝 모델을 학습시키기 전에 데이터를 전처리하고 변환하는 데 사용되는 다양한 도구를 제공
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt
```

예제 – S&P500 종가 예측

```
# 1: 데이터 셋
### S&P 500 데이터 다운로드
### 2010/01/01 ~ 2024/01/01까지 S&P500 자료
df = yf.download('^GSPC', start='2010-01-01', end='2024-01-01', auto_adjust=True)
data = df['Close'].values.reshape(-1, 1)          # 종가, NumPy 자료형 변환, 행(시계열)데이터로 변경

### 데이터 정규화
scaler = MinMaxScaler(feature_range=(0, 1)) # 데이터를 0 ~ 1로 정규화하기 위한 객체
scaled_data = scaler.fit_transform(data)          # fit: data에서 최대값과 최솟값을 찾기
                                                  # + transform: 구한 최대값과 최솟값을 획득하여 해당 범위로 정규화

### 학습용 및 테스트용 데이터 분리
train_size = int(len(scaled_data) * 0.8)          # training에 사용할 갯수 계산
train_data = scaled_data[:train_size]             # 계산한 갯수에 맞는 training data 할당
test_data = scaled_data[train_size:]              # 나머지 test set으로
```

예제 – S&P500 종가 예측

```
# 1: 데이터 셋
### 시퀀스 데이터셋 생성
def create_dataset(dataset, look_back=60):                    # window size에 따른 입력값, 결과값 생성하는 함수 정의
    X, Y = [], []
    for i in range(len(dataset) - look_back):
        X.append(dataset[i:(i + look_back), 0])
        Y.append(dataset[i + look_back, 0])
    return np.array(X), np.array(Y)

look_back = 60
X_train_np, Y_train_np = create_dataset(train_data, look_back)    # train set
X_test_np, Y_test_np = create_dataset(test_data, look_back)       # test set

### NumPy 배열을 PyTorch Tensor로 변환
X_train = torch.FloatTensor(X_train_np).unsqueeze(-1)
Y_train = torch.FloatTensor(Y_train_np).unsqueeze(-1)
X_test = torch.FloatTensor(X_test_np).unsqueeze(-1)
Y_test = torch.FloatTensor(Y_test_np).unsqueeze(-1)

### PyTorch DataLoader 생성
train_dataset = TensorDataset(X_train, Y_train)
train_loader = DataLoader(dataset=train_dataset, batch_size=32, shuffle=False)
```

예제 – S&P500 종가 예측

```
# 2: 모델 정의
class RNN(nn.Module):
    def __init__(self, input_size, hidden_size, num_layers, output_size):
        super(RNN, self).__init__()
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.rnn = nn.RNN(input_size, hidden_size, num_layers, batch_first=True)
        self.fc = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        h0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).to(x.device)  # 초기 hidden state와 cell state 설정
        out, _ = self.rnn(x, h0)                                                      # RNN에 입력 데이터와 hidden state 전달
        out = self.fc(out[:, -1, :])                                                  # 마지막 타임 스텝의 출력만 사용
        return out

# 모델 파라미터 설정
input_size, hidden_size, num_layers, output_size = 1, 50, 1, 1

model = RNN(input_size, hidden_size, num_layers, output_size)
```

예제 – S&P500 종가 예측

```
# 3: 손실함수 & 옵티마이저
criterion = nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
```

예제 – S&P500 종가 예측

```
# 4: 학습 루프
num_epochs = 20
for epoch in range(num_epochs):
    for X_batch, Y_batch in train_loader:
        optimizer.zero_grad()                  # 순전파 gradient 초기화
        outputs = model(X_batch)               # 순전파 (forward propagation)
        loss = criterion(outputs, Y_batch)     # 손실 계산

        loss.backward()                        # 역전파 (backward propagation, gradient 계산)
        optimizer.step()                       # 파라미터 업데이트

    print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}’)
```

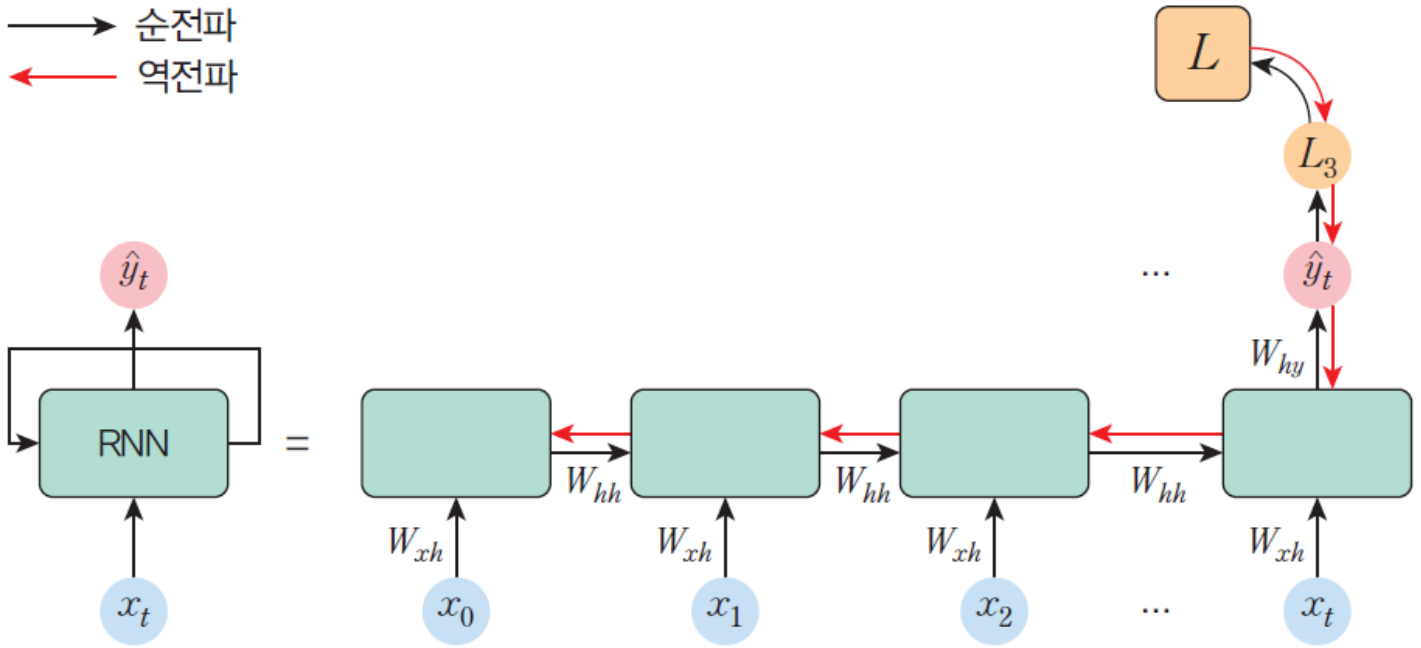
예제 – S&P500 종가 예측

```
# 5: 평가: 추론
with torch.no_grad():
    test_predict_tensor = model(X_test)
```

예제 – S&P500 종가 예측

```
# 추가: 결과 시각화
# 예측 결과를 NumPy 배열로 변환 후 원래 스케일로 되돌리기
Test_predict_np = test_predict_tensor.detach().numpy()          # detach(): 필요한 정보만/ numpy(): NumPy 배열로 변환
Test_predict = scaler.inverse_transform(test_predict_np)        # MinMaxScaler에 의해 변환 된 예측 값 복원

Actual_test_prices = df[ ' Close ' ].values[len(train_data):]   # 실제 값
Test_dates = df.index[len(train_data):]                         # X축에 들어갈 값

# Plot the actual test prices and the predicted test prices
Plt.figure(figsize=(12, 6))
Plt.plot(test_dates, actual_test_prices, label= ' Actual Price ' , color= ' blue’)                          # 실제 값 그래프 생성
Plt.plot(df.index[len(train_data) + look_back :], test_predict, label= ' Test Predict ' , color= ' red’)    # 예측 값 그래프 생성

# Set the title and labels for the plot
plt.title('S&P 500 using RNN')
plt.xlabel('Date’)                          # X축 label 생성
plt.ylabel('S&P 500 Close Price’)           # Y축 label 생성
Plt.legend()                                # 범례 생성
Plt.show()                                  # 생성된 그래프를 화면에 표시
```

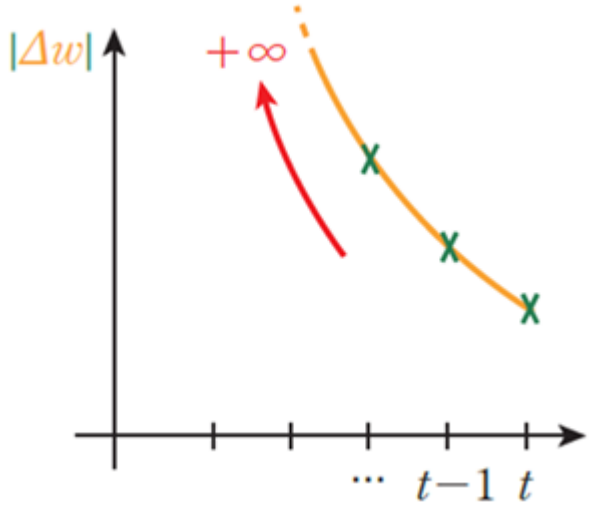
RNN의 학습

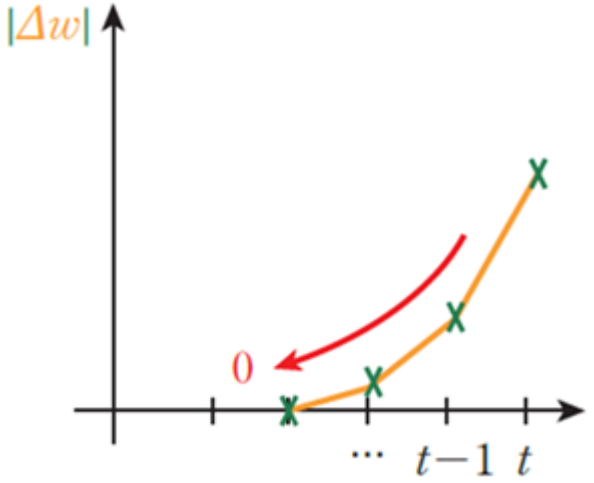


시간을 거슬러 올라가면서 역전파

➔ BPTT(Backpropagation Through Time)

RNN – 학습을 하다 보면...





$|\Delta w| > 1$ ➔ 발산
➔ 학습 X

$|\Delta w| < 1$ ➔ 0으로 수렴
➔ Gradient Vanishing

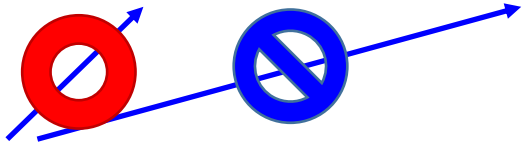
기울기 소실 문제 – 장기 기억 문제

Gradient Vanishing

➔많은 입력이 있을 수록, 초기 입력에 대한 가중치가 업데이트 되지 않는다.

➔장기 기억 상실 문제 발생

The cat saw the tiger and ran away.



It was scared.

“Without forgetfulness there can be no happiness” 니체, 『도덕의 계보학』

➔적당히 잊어야 한다.

➔but, 기억해야 하는 것 까지 잊어서는 안된다.

인간은?

인간은 단기 기억과 장기 기억을 별도로 관리한다.

구분	단기 기억 (STM / Working Memory)	장기 기억 (LTM)
정의	짧은 시간(수 초~수십 초) 동안 정보를 유지·처리	오랫동안(몇 시간~평생) 저장되는 정보
저장 용량	제한적 (일반적으로 7±2 항목; Miller, 1956)	사실상 무제한적
저장 위치	주로 전전두엽(prefrontal cortex)에서 활성화	대뇌 피질 전반에 분산 저장 (해마는 형성 과정에 핵심적)

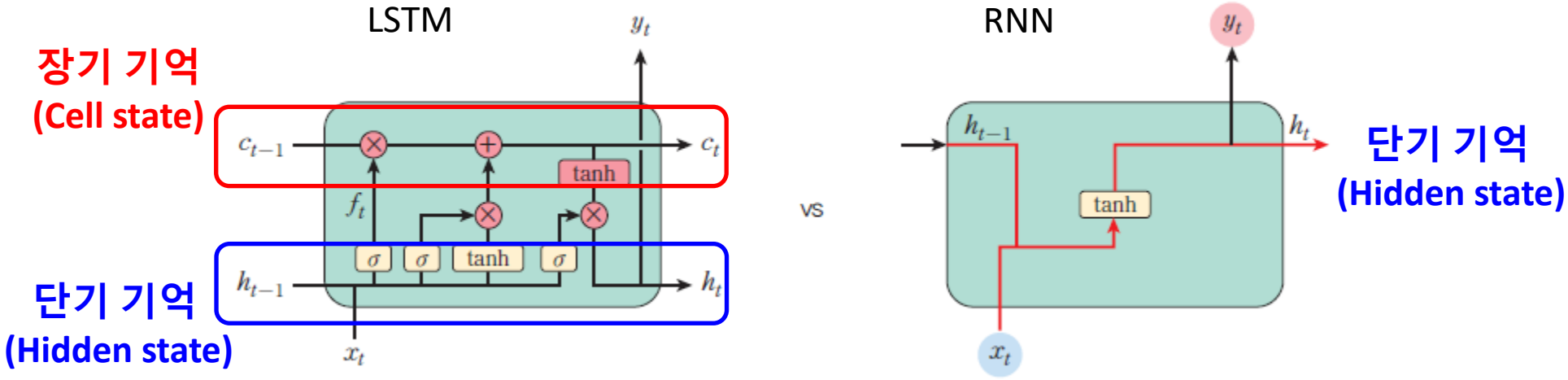
LSTM(Long Short-Term Memory)

장기 기억이 잘 안 되네?

➔ 장기 기억을 위한 새로운 state 를 추가하자. (Cell state)

각각의 정보를 어떻게 관리 하지?

➔ 세 가지 gate



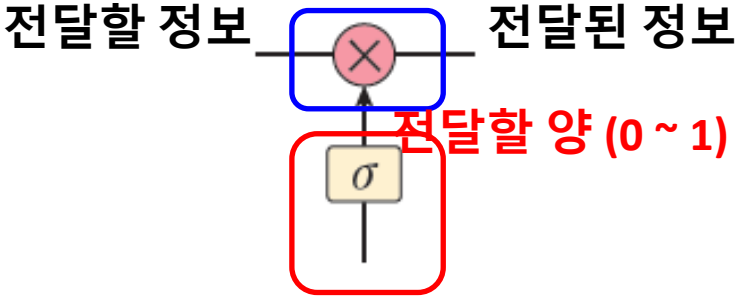
Gate

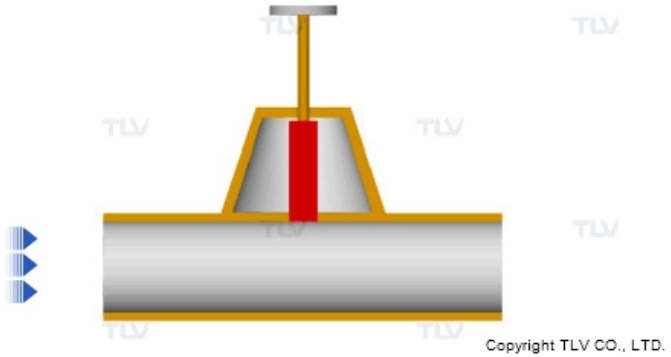
LSTM 안의 정보들은 게이트를 통하여 추가되거나 삭제

Sigmoid 함수 (0 ~ 1)로 정보의 흐름 세기 조절 (곱하기)

0: 정보 차단

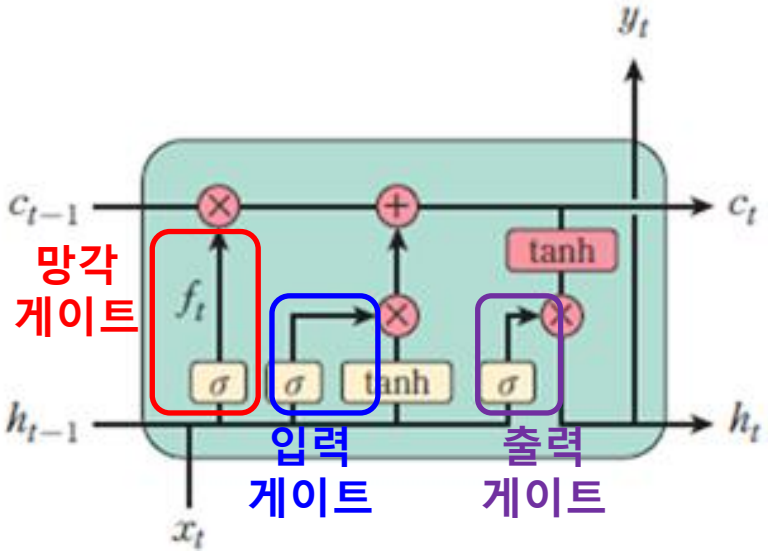
1: 정보 전달





LSTM

망각 게이트 (Forget Gate): 이전 셀 상태를 얼마나 유지할지 결정
장기 기억 관리 (잊어야 할 것들은 잊자)
입력 게이트 (Input Gate): 새로운 정보를 얼마나 추가할 지 결정
단기 기억 → 장기 기억
출력 게이트 (Output Gate): 다음 은닉 상태 결정
장기 기억 → 단기 기억



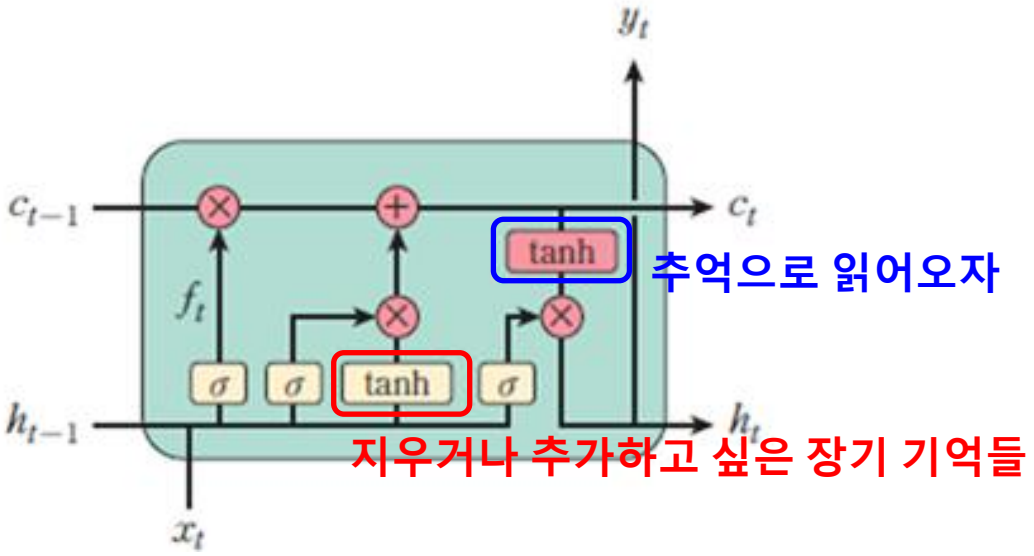
활성화 함수

Sigmoid 함수는 0 ~ 1로 전달 정보의 양을 조절

➔ 잊고 싶은 기억은(정보는)? ➔ 음수 값 필요

➔ 기억(정보) 자체를 다룰 때는 -1 ~ 1의 값이 필요

➔ tanh (-1 ~ 1) 사용

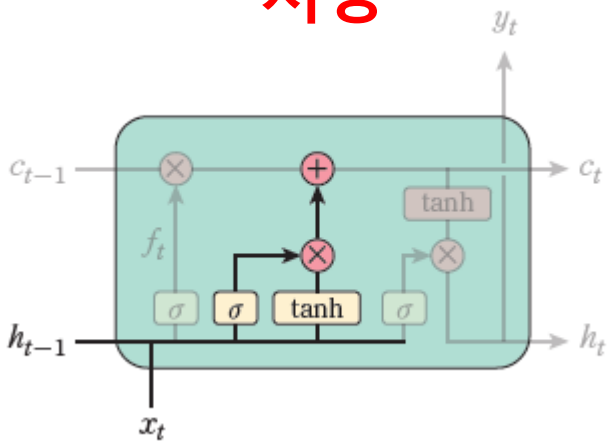


셀 상태 변화 (저장, 삭제)

입력 게이트 ➔ 선택된 기억이 추가

삭제 게이트 ➔ 기존의 일부 기억을 삭제

저장



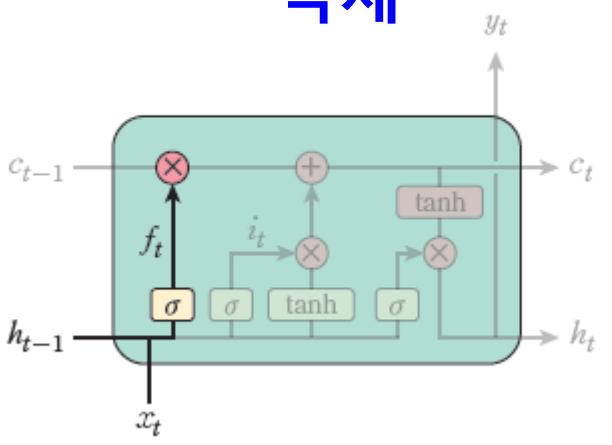
변경하고자 하는 기억들

$$g_t = \tanh(W_{xg}x_t + W_{hg}h_{t-1})$$

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1})$$

변경하고자 하는 세기

삭제



$$f_t = \sigma(W_{fx}x_t + W_{fh}h_{t-1})$$

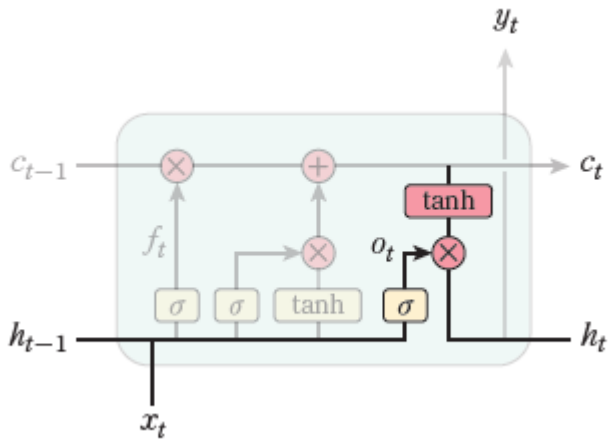
지우고 싶은 세기

$$C_t = f_t C_{t-1} + i_t g_t$$

*Matrix multiplication: 각 state의 차원 및 표현에 맞게 변환

출력(Hidden state)

출력 게이트:
현재 시점의 입력값과 이전 시점의 은닉 상태를 기반으로 다음 은닉 상태로 전달할 장기 기억을 결정



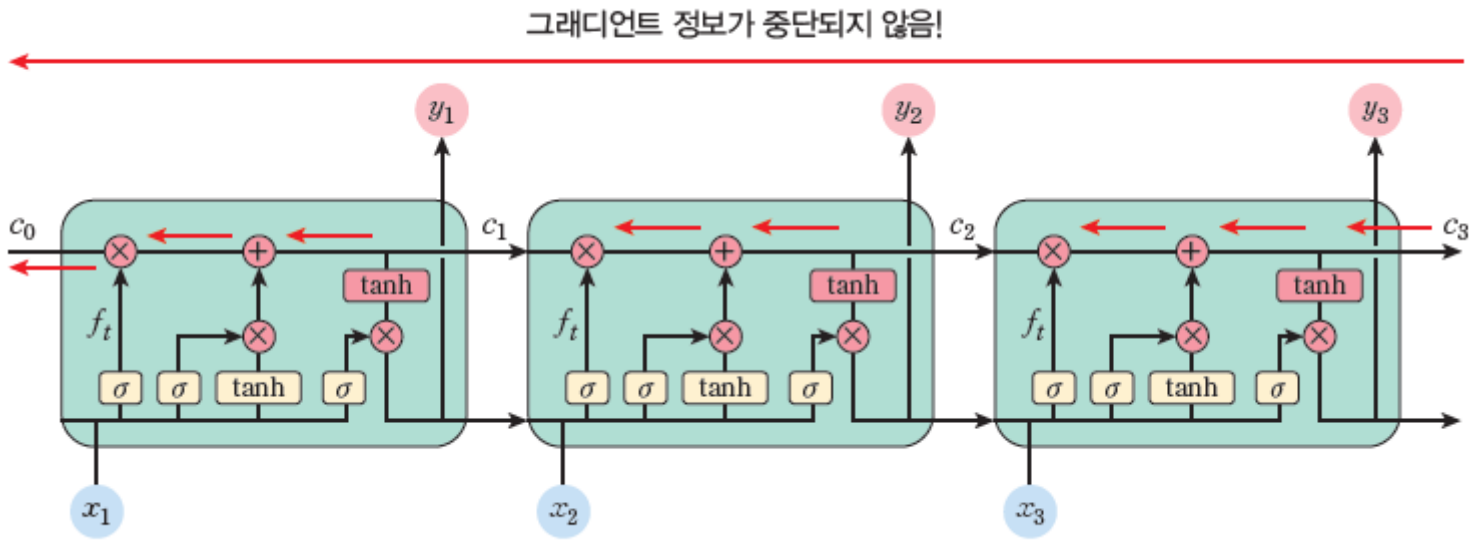
전달하는 기억의 세기

$$O_t = \sigma\left(W_{ox}x_t + W_{oh}h_{t-1}\right)$$

$$h_t = O_t \tanh(C_t)$$

단기 기억으로 전달하고하 하는 장기 기억들

장기 기억 문제 in LSTM – 해결



예제 – LSTM

```
class LSTM(nn.Module):
    def __init__(self, input_size, hidden_size, num_layers, output_size):
        super(LSTM, self).__init__()
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.lstm = nn.LSTM(input_size, hidden_size, num_layers, batch_first=True)
        self.fc = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        # 초기 hidden state와 cell state 설정
        h0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).to(x.device)
        c0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).to(x.device)        # 장기 기억을 위한 cell state

        out, _ = self.lstm(x, (h0, c0))
        out = self.fc(out[:, -1, :])
        return out

input_size, hidden_size, num_layers, output_size = 1, 50, 1, 1

model = LSTM(input_size, hidden_size, num_layers, output_size)
```

GRU(Gated Recurrent Unit)

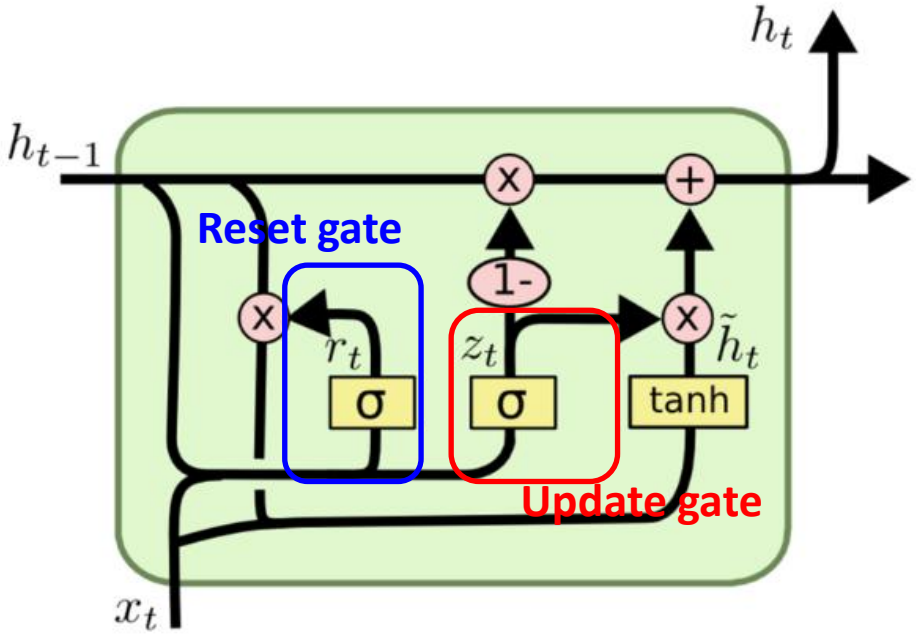
LSTM은 좋긴 한데 너무 복잡하다.

➔본질만 남기고, 간소화 해 보자

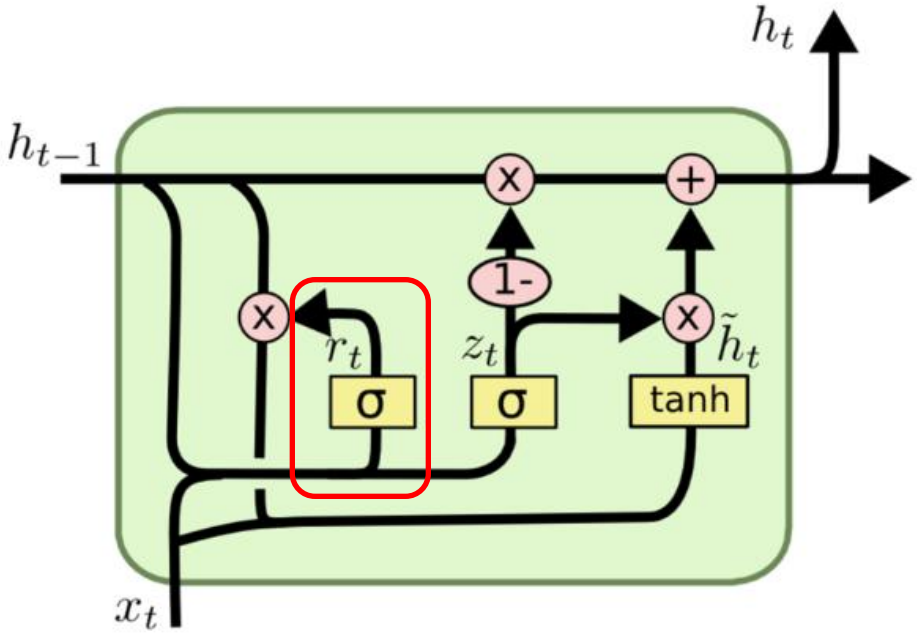
➔과연 본질은?

➔현재 정보를 기억과 조화롭게 썩어서 새로운 기억을 만들자.

= Reset gate = Update gate



GRU(Gated Recurrent Unit)



Reset gate:

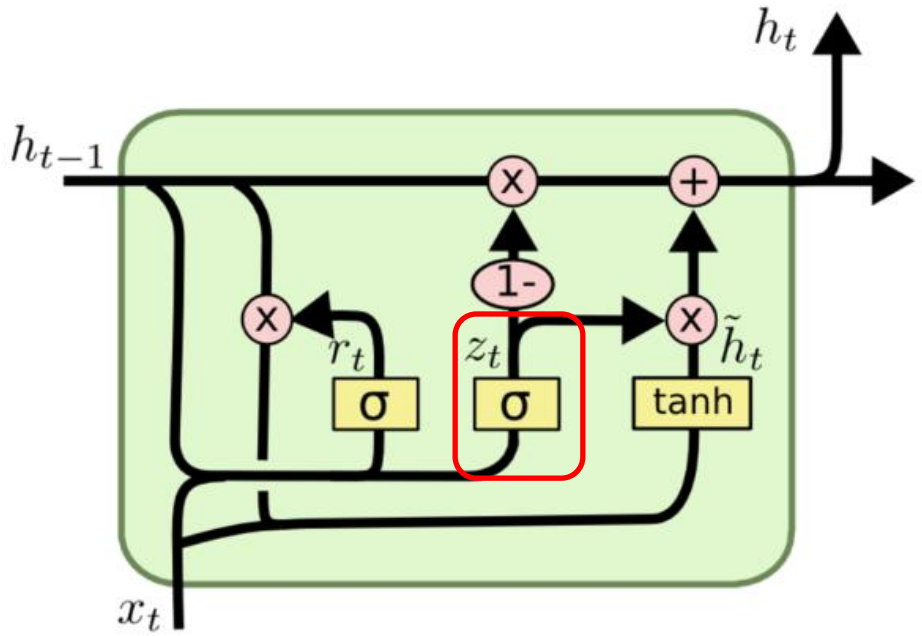
이전 상태와 **현재 입력**을 보고 **다음 상태에 반영 될 이전 상태 양** 조절

이전 상태

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t])$$

밸브의 세기 현재 입력

GRU(Gated Recurrent Unit)



Update gate:

이전 상태와 **현재 입력**을 보고,
이전 상태와 **조작된 정보**의
다음 상태에 반영 될 비율 조절

이전 상태

$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t])$$

반영 비율 현재 입력

$$\tilde{h}_t = \tanh(W \cdot [r_t \cdot h_{t-1}, x_t])$$

조작된 정보 Reset gate에 의해 조절된 이전 상태

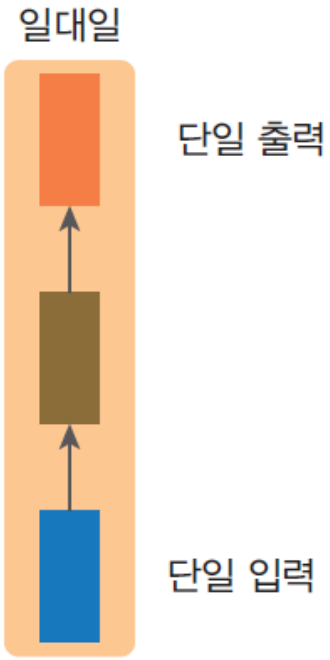
$$h_t = (1 - z_t) \cdot h_{t-1} + z_t \cdot \tilde{h}_t$$

다음 상태

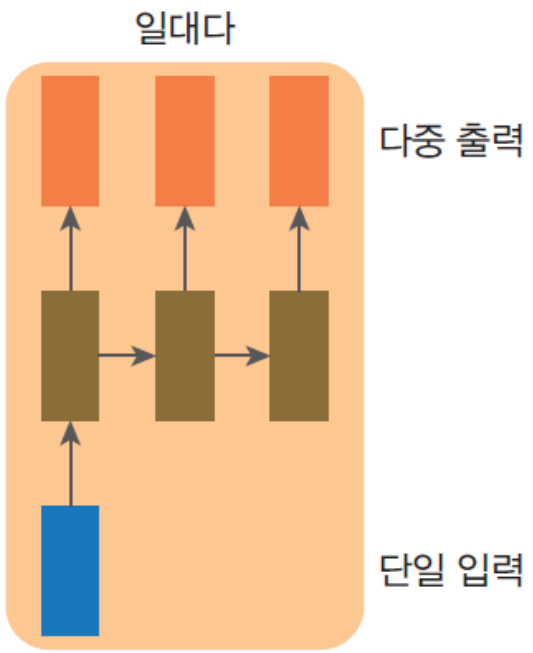
예제 – GRU

```
class GRU(nn.Module):
    def __init__(self, input_size, hidden_size, num_layers, output_size):
        super(GRU, self).__init__()
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.gru = nn.GRU(input_size, hidden_size, num_layers, batch_first=True)
        self.fc = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        h0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).to(x.device)

        out, _ = self.gru(x, h0)
        out = self.fc(out[:, -1, :])
        return out

input_size, hidden_size, num_layers, output_size = 1, 50, 1, 1

model = GRU(input_size, hidden_size, num_layers, output_size)
```

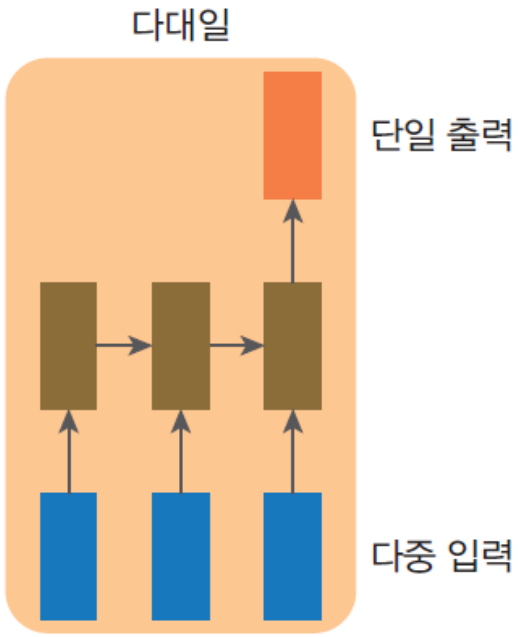
언제 쓰나?



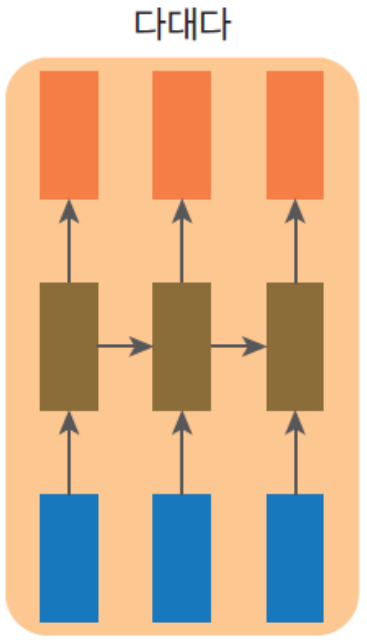
많은 머신 러닝 문제



이미지 캡션을 생성

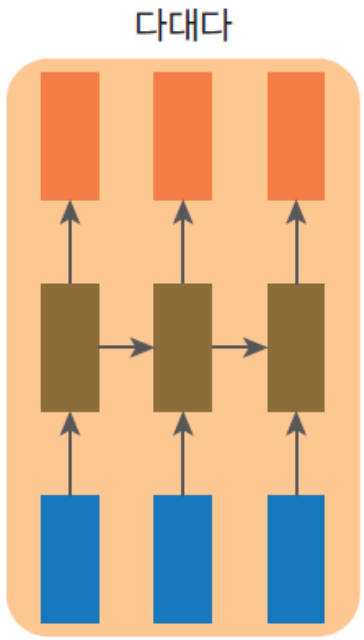


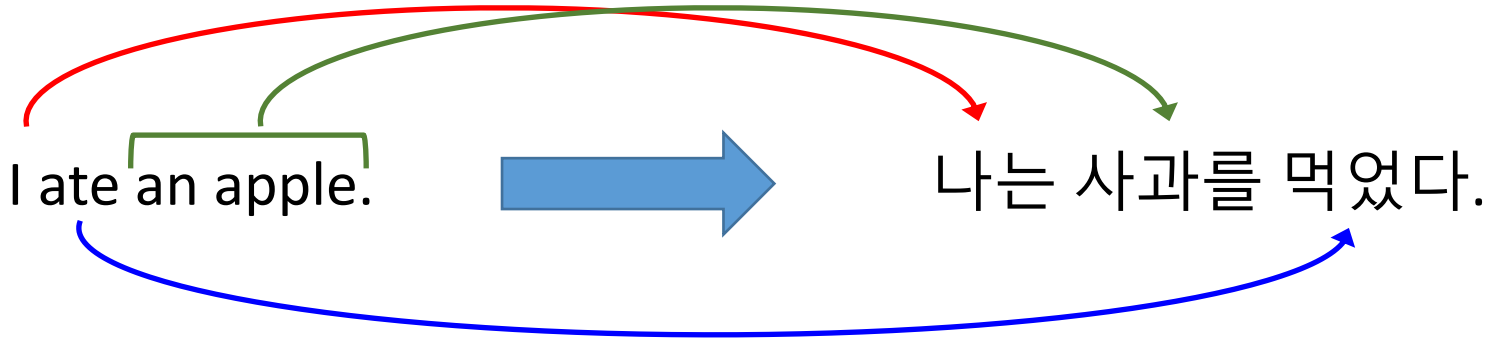
감정(Sentiment) 분석



번역

Seq2Seq 모델의 등장





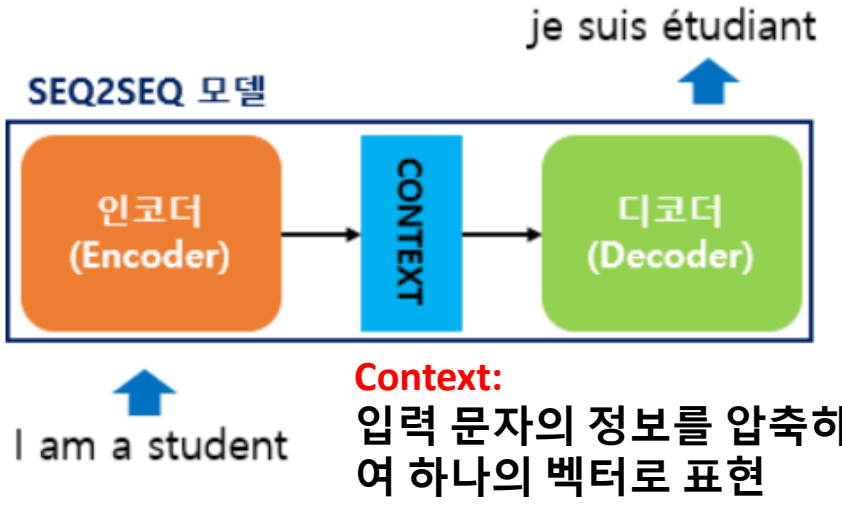
번역 순서가 다른데?
번역해서 나오는 단어의 개수가 다른데?

➔ **문장 전체의 정보**를 **잘 정리**하여 **통째로 전달** 해 보자!

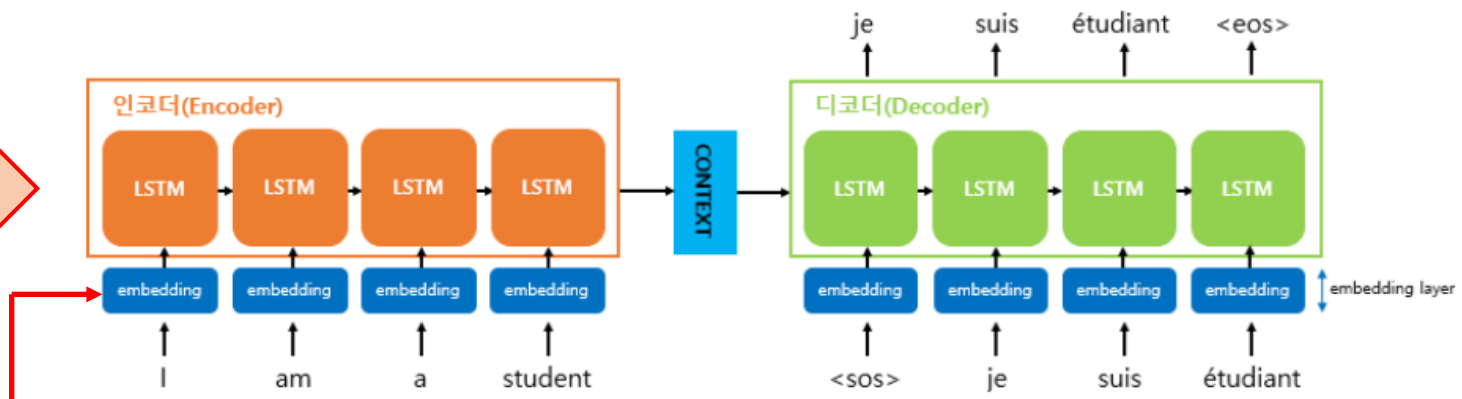
Seq2Seq

Seq2Seq: 인코더와 디코더로 구성

Encoder: 입력 문장의 모든 단어들을 순차적으로 입력 받은 뒤 마지막 state(vector)를 디코더로 전송
Decoder: Encoder로 부터 vector(context)를 받아서 번역된 단어를 하나씩 출력



Context:
입력 문자의 정보를 압축하여 하나의 벡터로 표현



Word Embedding
단어 → Vector
컴퓨터가 이해하기 쉽게 + 데이터의 의미 및 관계성 보존

"Sequence to Sequence Learning with Neural Networks,"
[1409.3215] Sequence to Sequence Learning with Neural Networks