RNN

이예린

CONTENTS

RNN

CNN: convolutional neural network

RNN: recurrent neural network

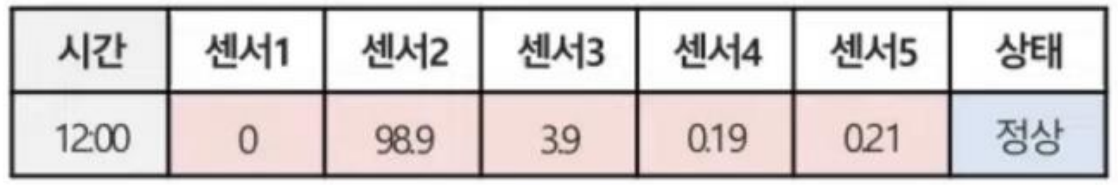
⬇

QUIZ1

“재귀성” 특징으로 인해 $\boxed{\quad\quad}$ 데이터를 처리하는 데에 사용되는 뉴럴 네트워크 구조

시계열데이터

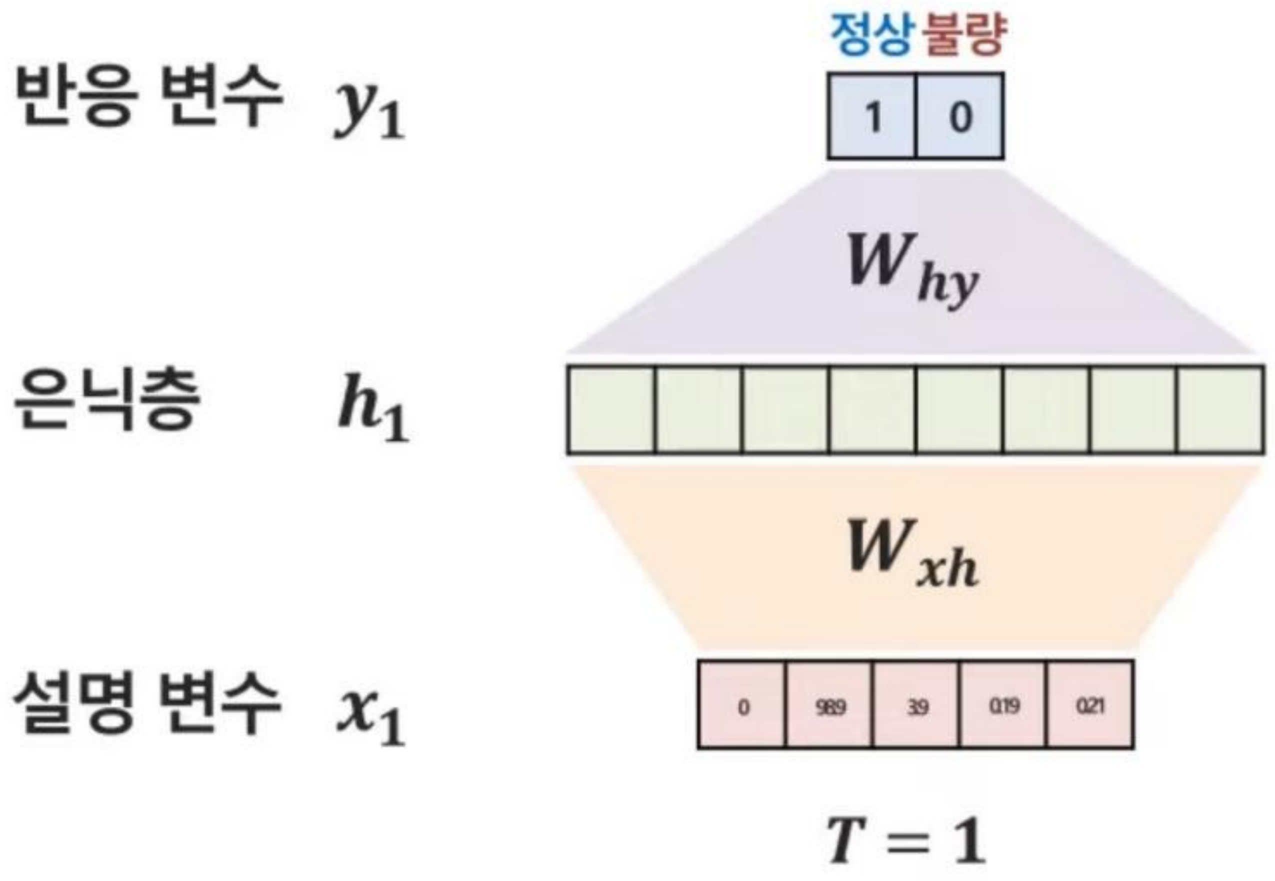
시점 1



시간	센서1	센서2	센서3	센서4	센서5	상태
12:00	0	98.9	3.9	0.19	0.21	정상

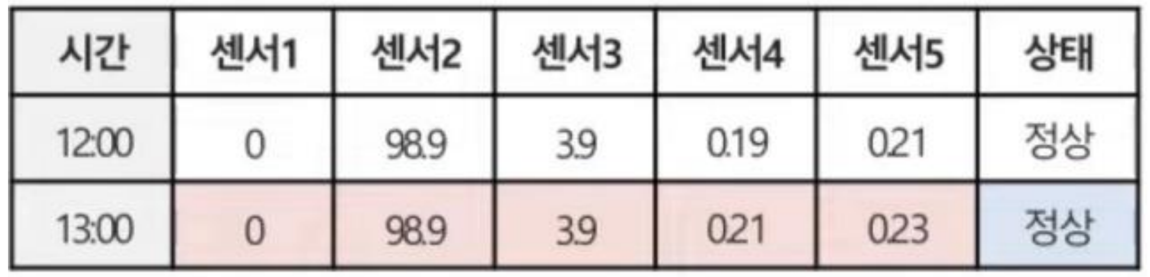
$$h_1 = f(W_{xh}\boxed{x_1})$$

$$y_1 = g(W_{hy}h_1)$$



시계열데이터

시점 2

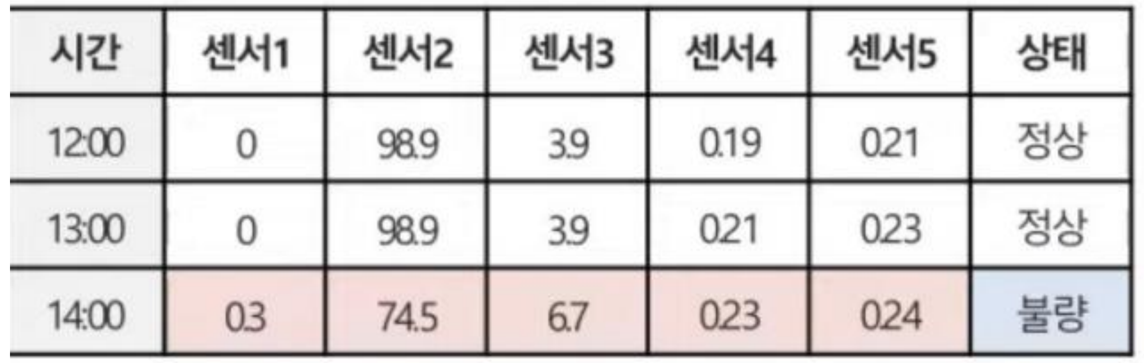


시간	센서1	센서2	센서3	센서4	센서5	상태
12:00	0	98.9	3.9	0.19	0.21	정상
13:00	0	98.9	3.9	0.21	0.23	정상

$$h_2 = f(W_{xh}\boxed{x_2})$$

$$y_2 = g(W_{hy}h_2)$$

시점 3



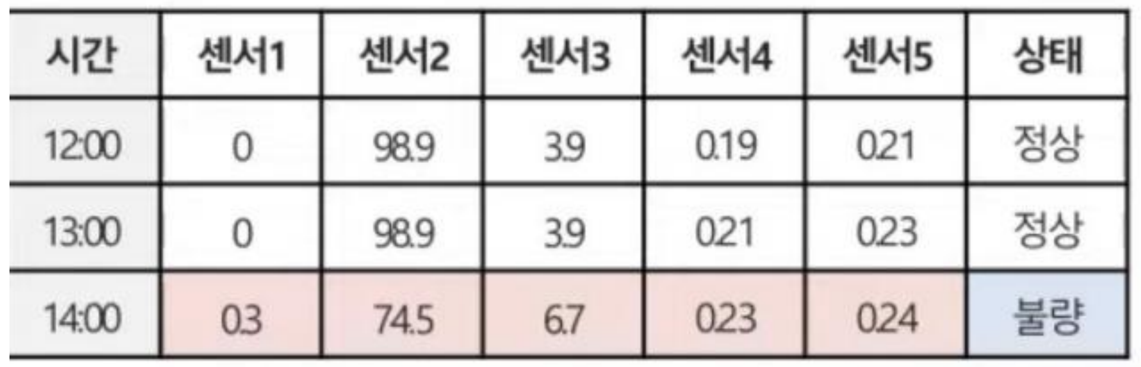
시간	센서1	센서2	센서3	센서4	센서5	상태
12:00	0	98.9	3.9	0.19	0.21	정상
13:00	0	98.9	3.9	0.21	0.23	정상
14:00	0.3	74.5	6.7	0.23	0.24	불량

$$h_3 = f(W_{xh}\boxed{x_3})$$

$$y_3 = g(W_{hy}h_3)$$

각 시점이 독립

시계열데이터



시간	센서1	센서2	센서3	센서4	센서5	상태
12:00	0	98.9	3.9	0.19	0.21	정상
13:00	0	98.9	3.9	0.21	0.23	정상
14:00	0.3	74.5	6.7	0.23	0.24	불량

$$h_3 = f(W_{xh}x_3)$$ +이전 시점 정보

$$y_3 = g(W_{hy}h_3)$$

QUIZ2

$f(\cdot) = \boxed{}$ $g(\cdot) = \boxed{}$

이전 시점의 정보까지 반영!

RNN

	시간	센서1	센서2	센서3	센서4	센서5	상태
$t-2$	12:00	0	98.9	3.9	0.19	0.21	정상
$t-1$	13:00	0	98.9	3.9	0.21	0.23	정상
t	14:00	0.3	74.5	6.7	0.23	0.24	불량

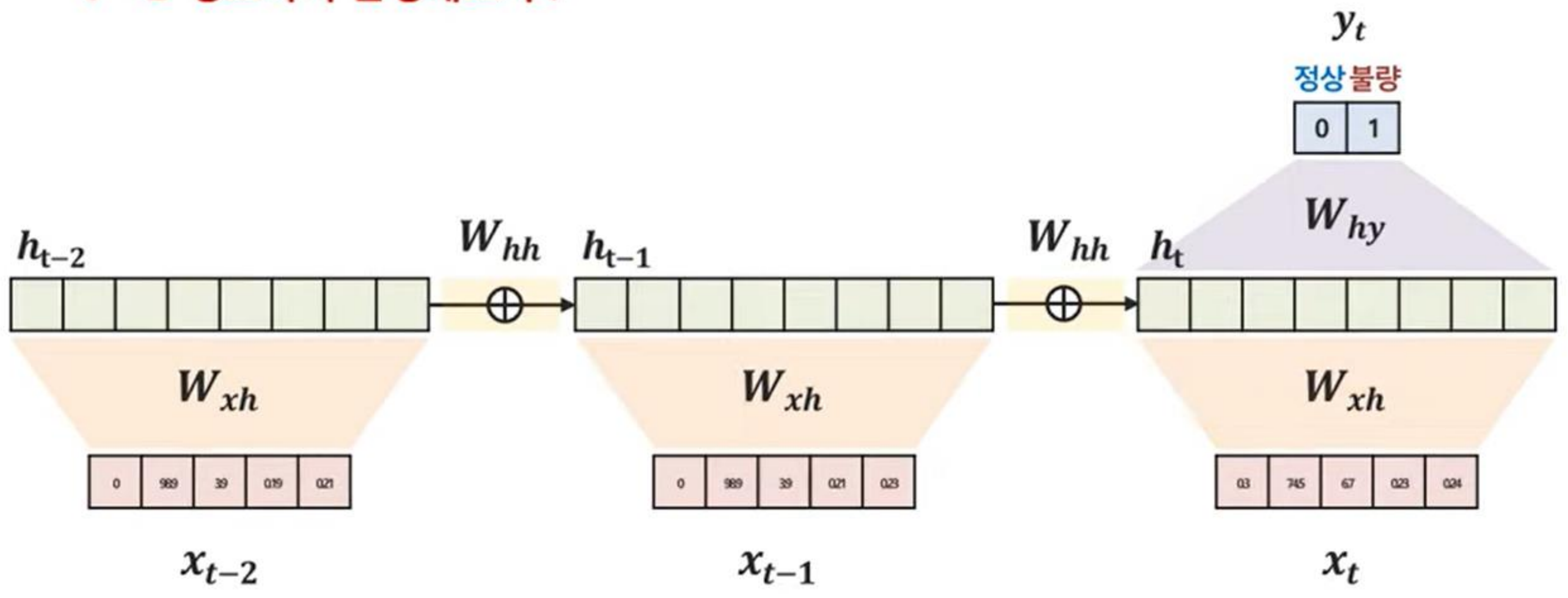
"$t-2$ 정보까지 반영해보자 !"

$$h_{t-1} = f(W_{xh}x_{t-1} + W_{hh}\boxed{h_{t-2})}$$

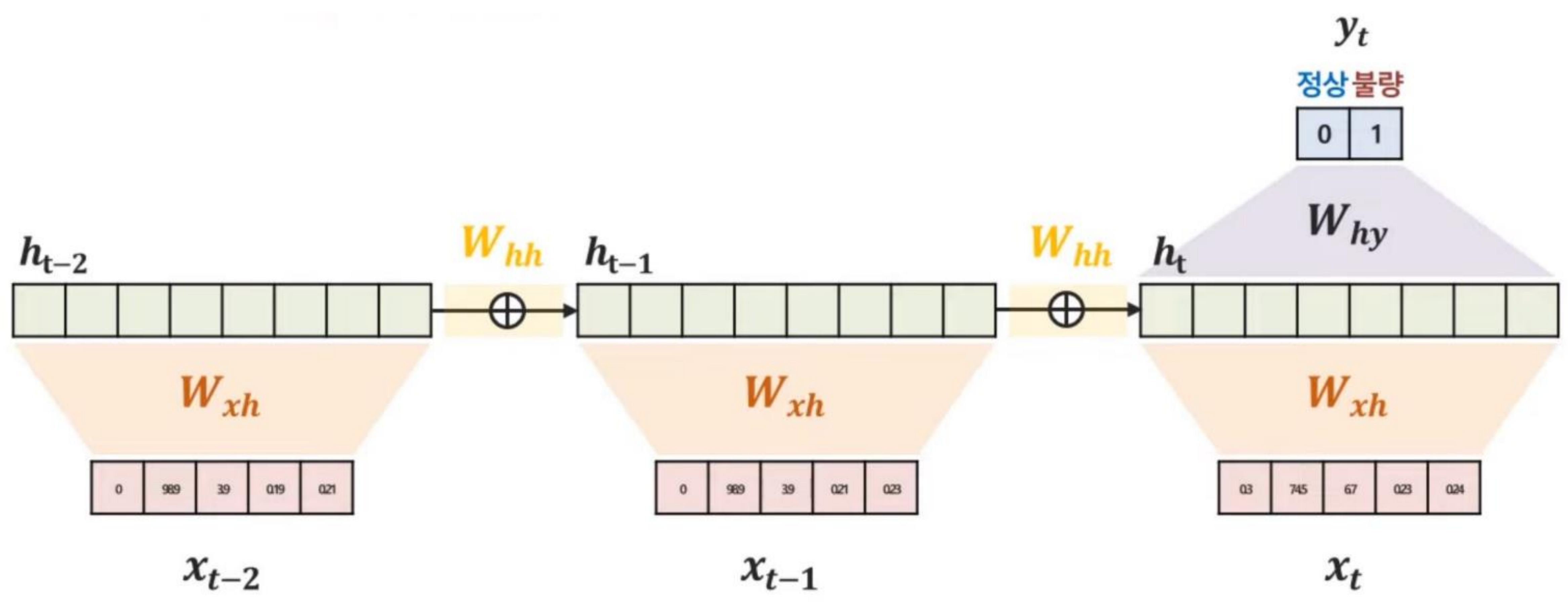
$$h_t = f(W_{xh}x_t + W_{hh}\boxed{h_{t-1})}$$

$$y_t = g(W_{hy}h_t)$$

$$f(\cdot) = tanh\ ,\ g(\cdot) = softmax$$



RNN



W_{hy} W_{xh} W_{hh}

같은 종류의 가중치 -> 같은 값

“가중치 공유”

RNN

We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

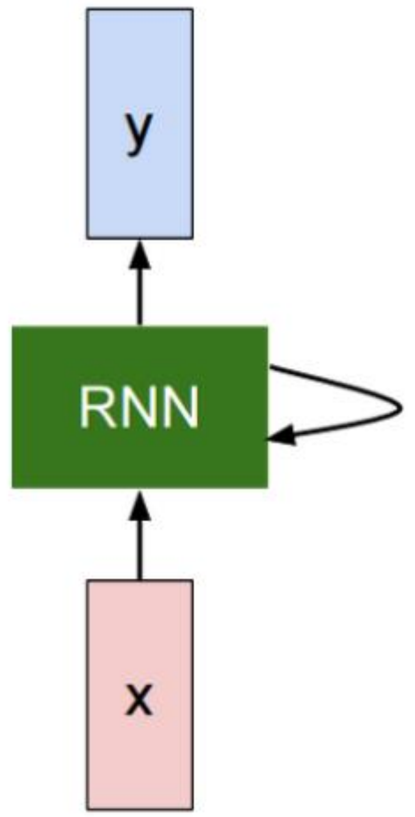
$$\boxed{h_t} = \boxed{f_W}(\boxed{h_{t-1}}, \boxed{x_t})$$

new state

old state

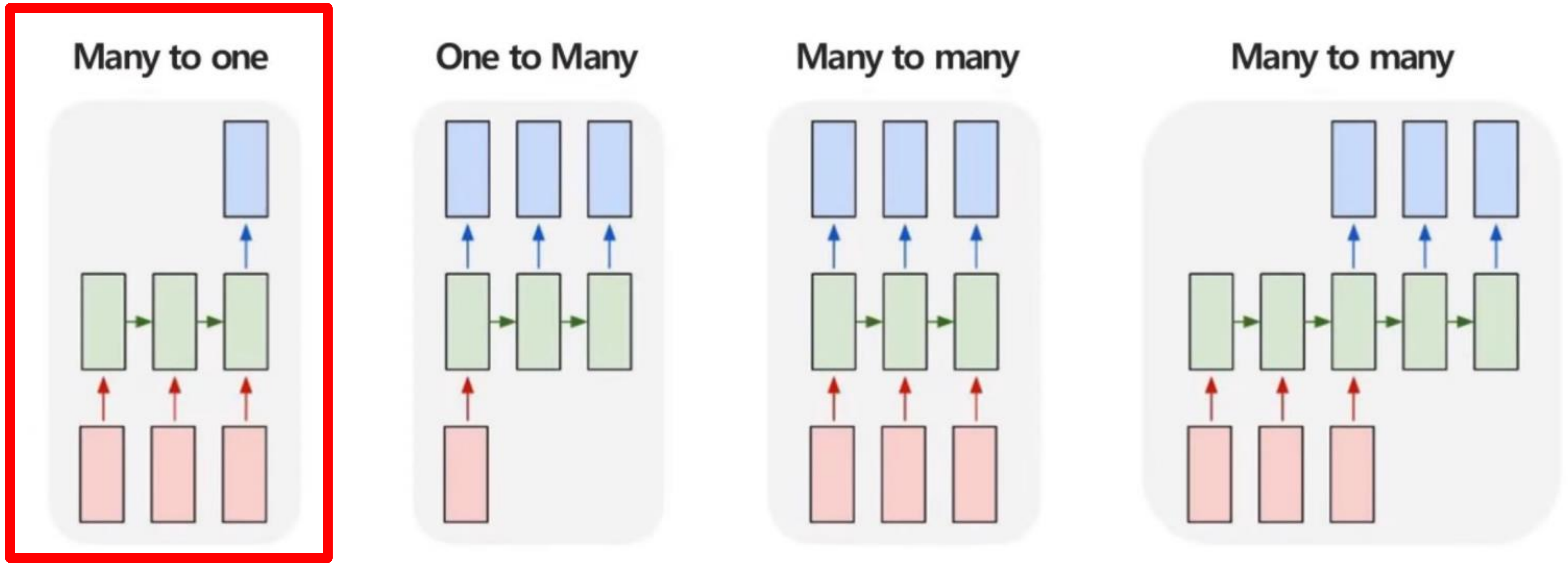
input vector at some time step

some function with parameters W



"recurrent neural network "

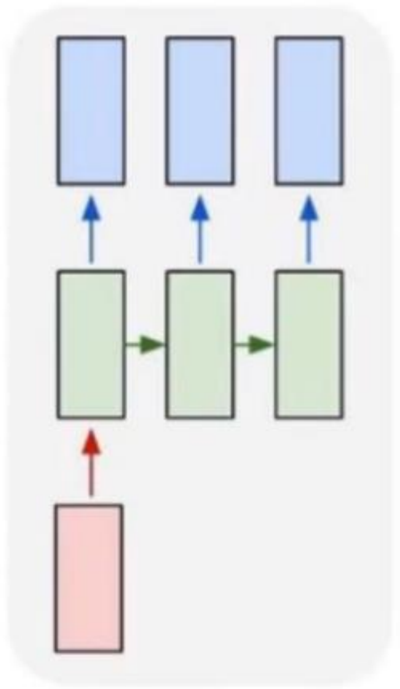
RNN



RNN

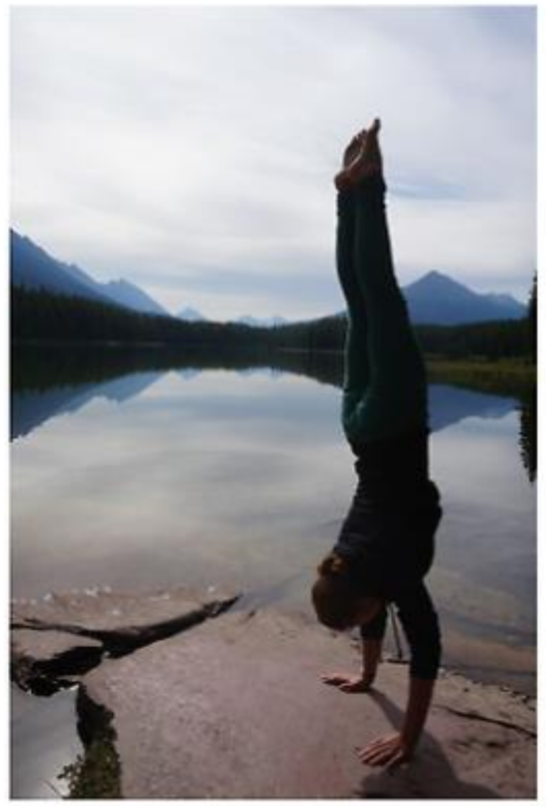
이미지 캡셔닝

One to Many



A woman is holding a cat in her hand

A person holding a computer mouse on a desk

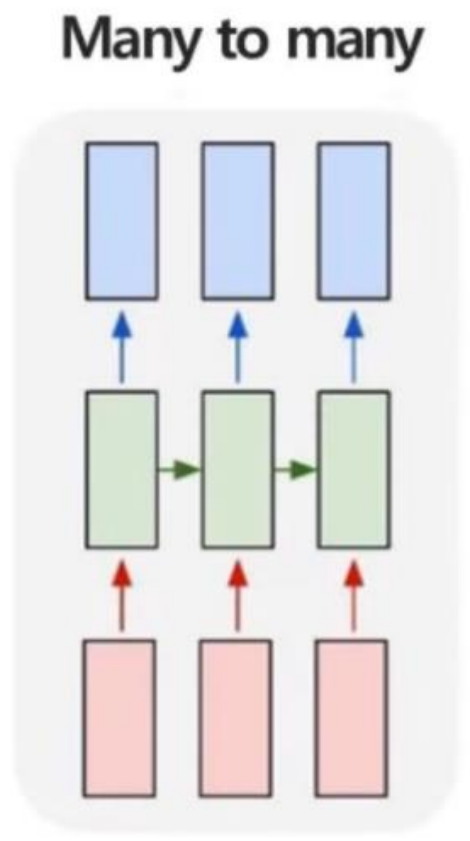

A woman standing on a beach holding a surfboard

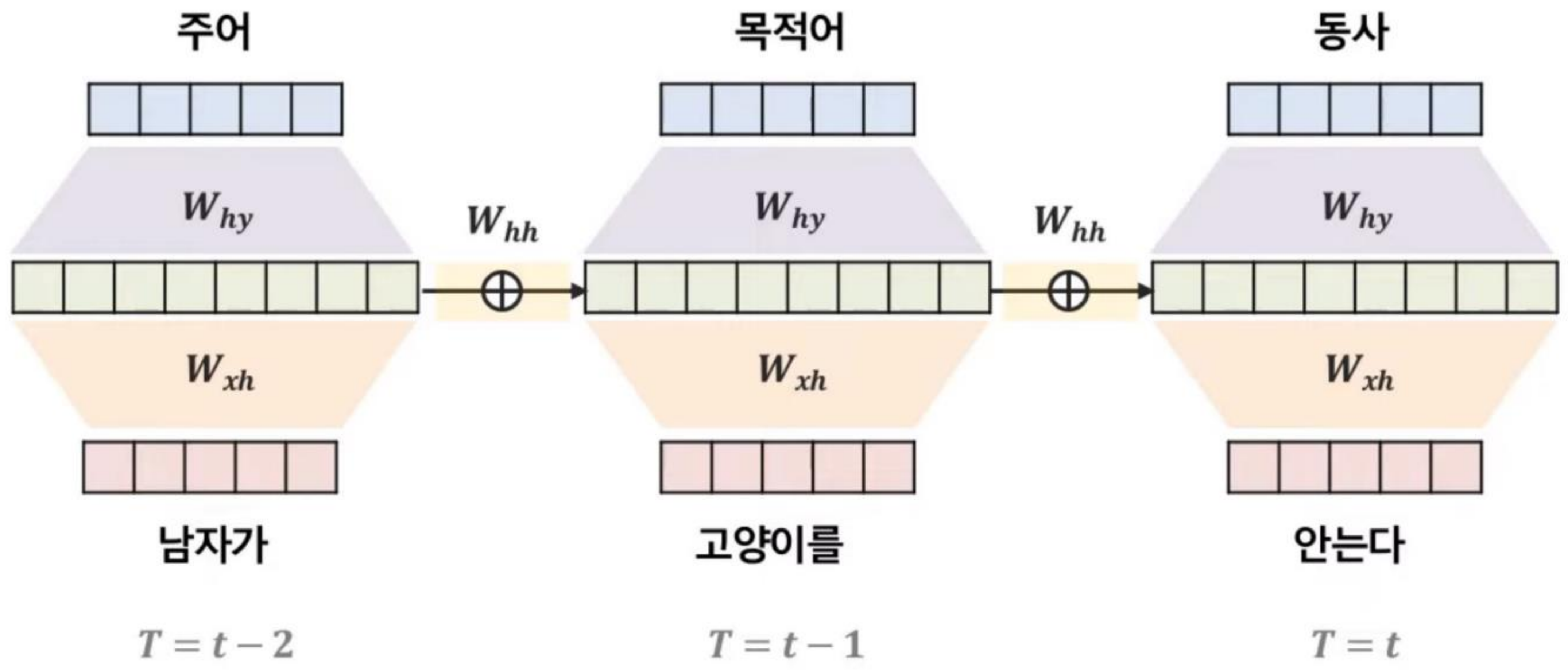
A bird is perched on a tree branch

A man in a baseball uniform throwing a ball

RNN

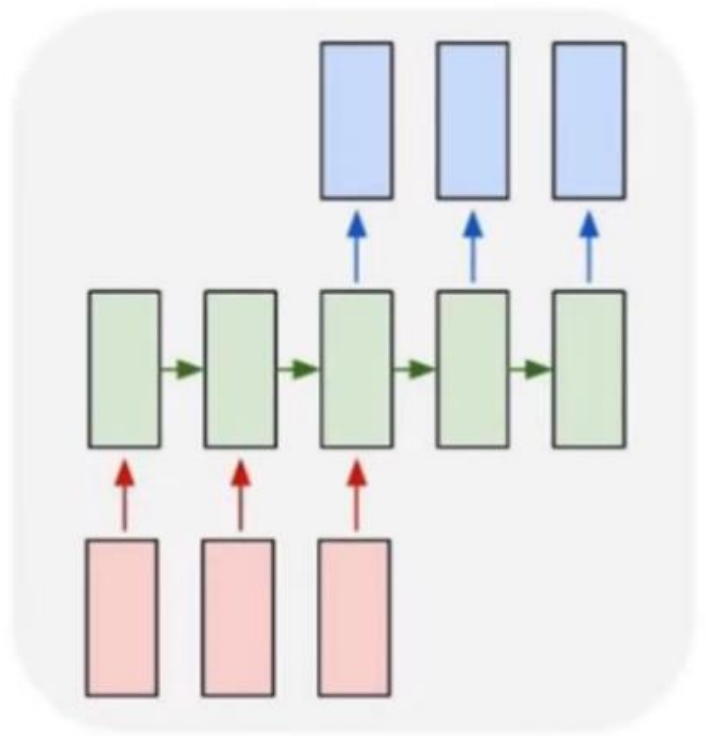
POS Tagging



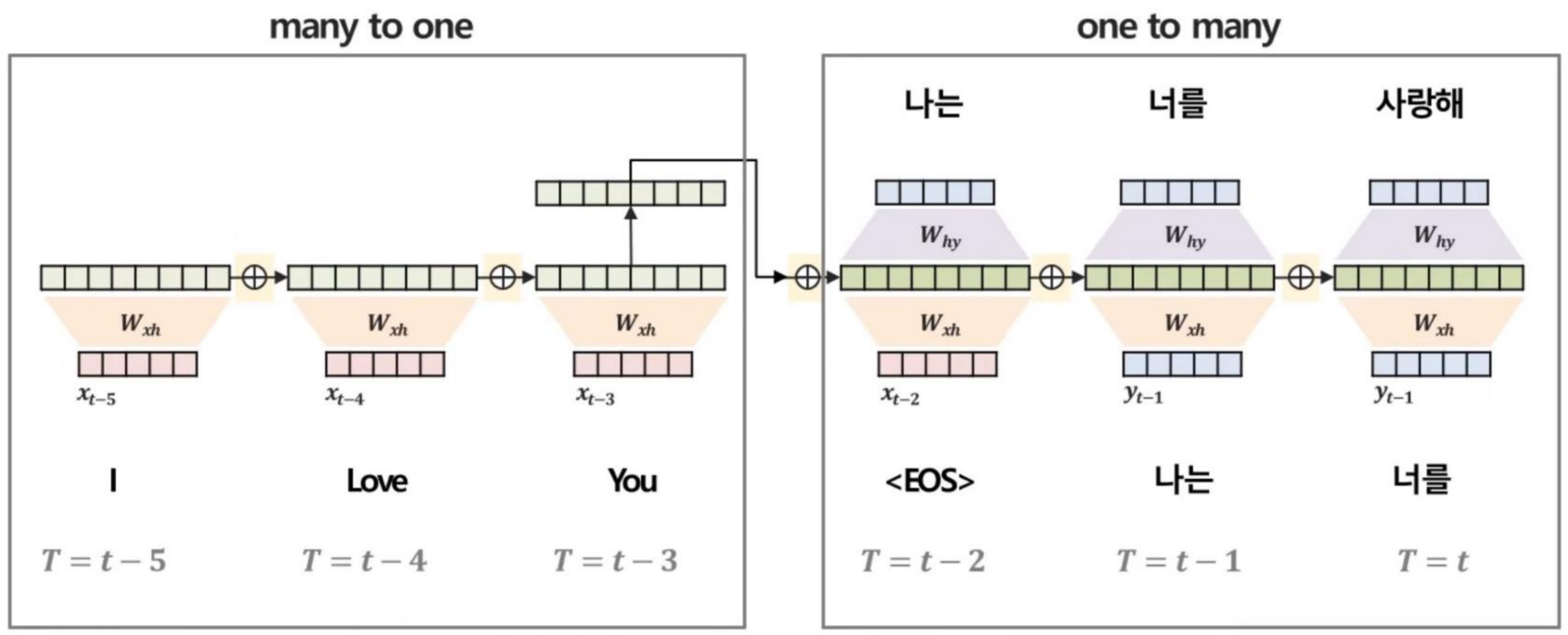


RNN

Many to many



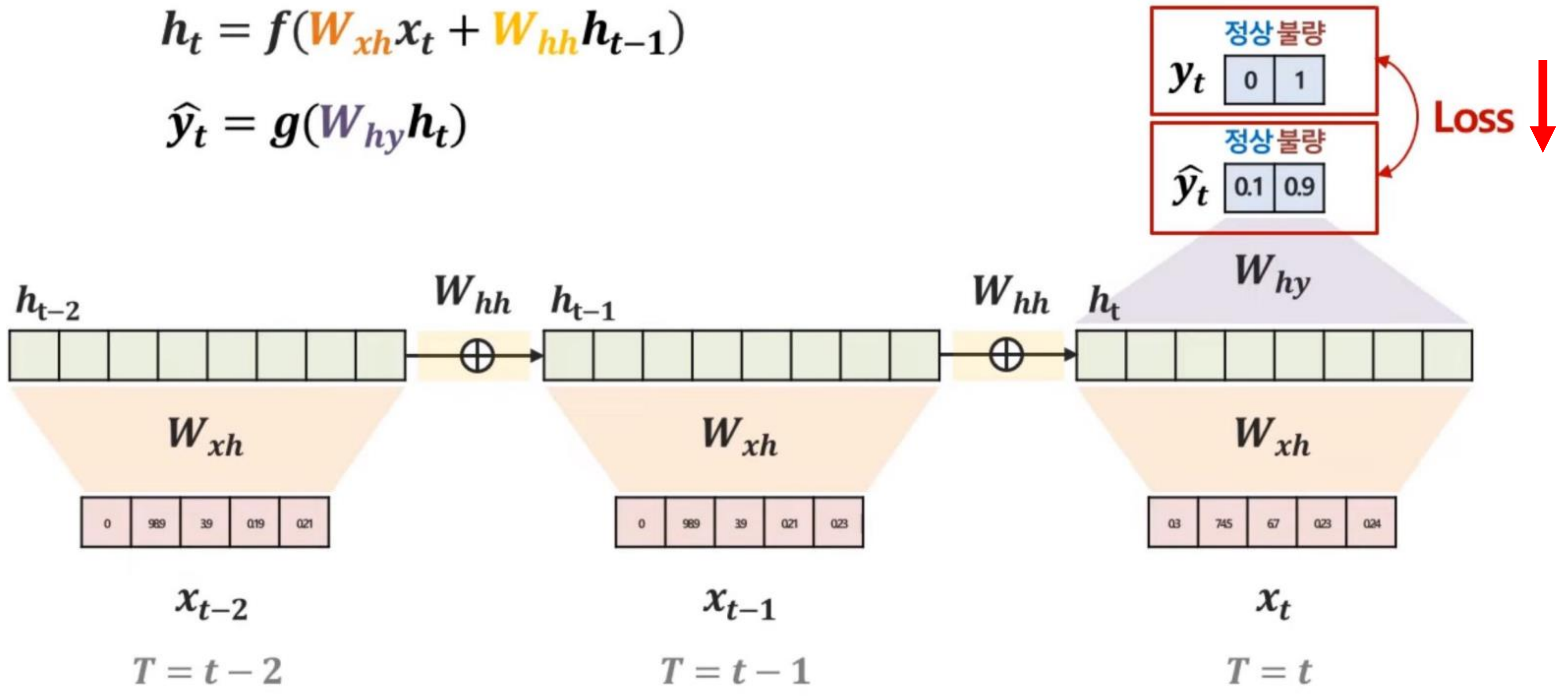
번역



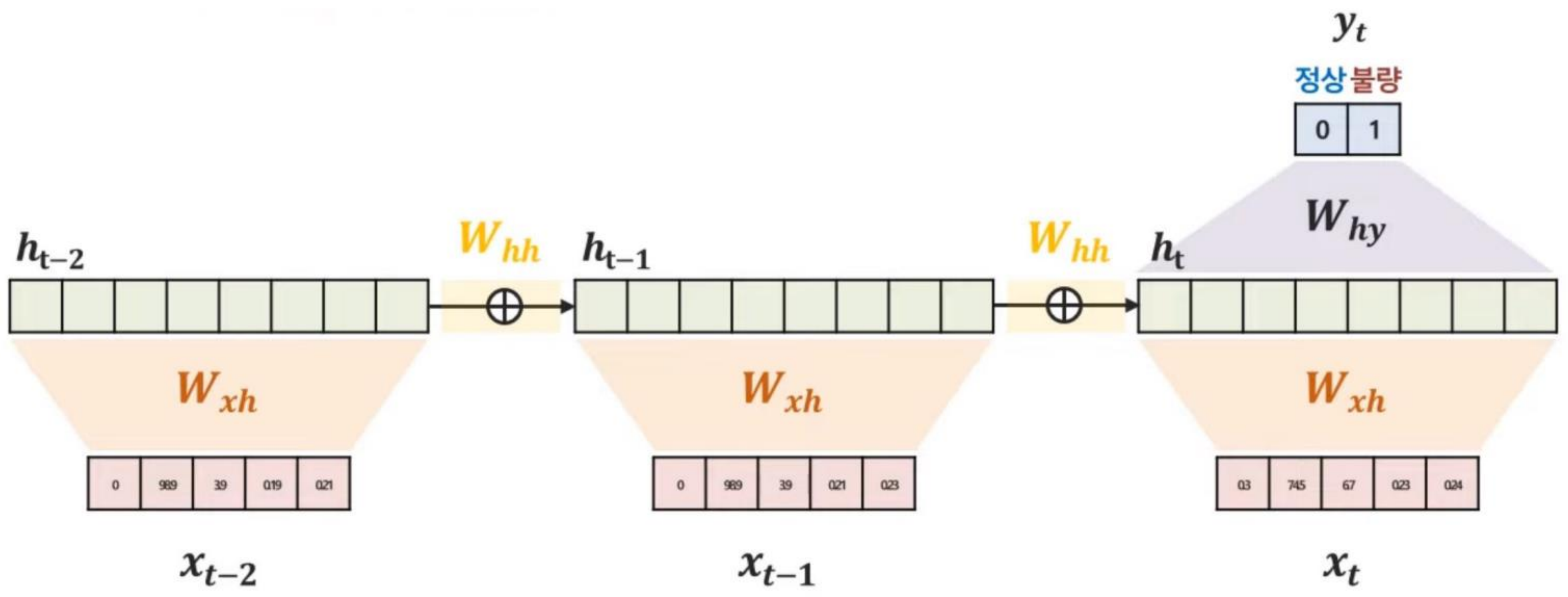
RNN

$$h_t = f(W_{xh}x_t + W_{hh}h_{t-1})$$

$$\hat{y}_t = g(W_{hy}h_t)$$



RNN



$$\frac{\partial Loss}{\partial W_{hy}} = \frac{\partial L_t}{\partial \hat{y}_t} \times \frac{\partial \hat{y}_t}{\partial W_{hy} h_t} \times \frac{\partial W_{hy} h_t}{\partial W_{hy}}$$

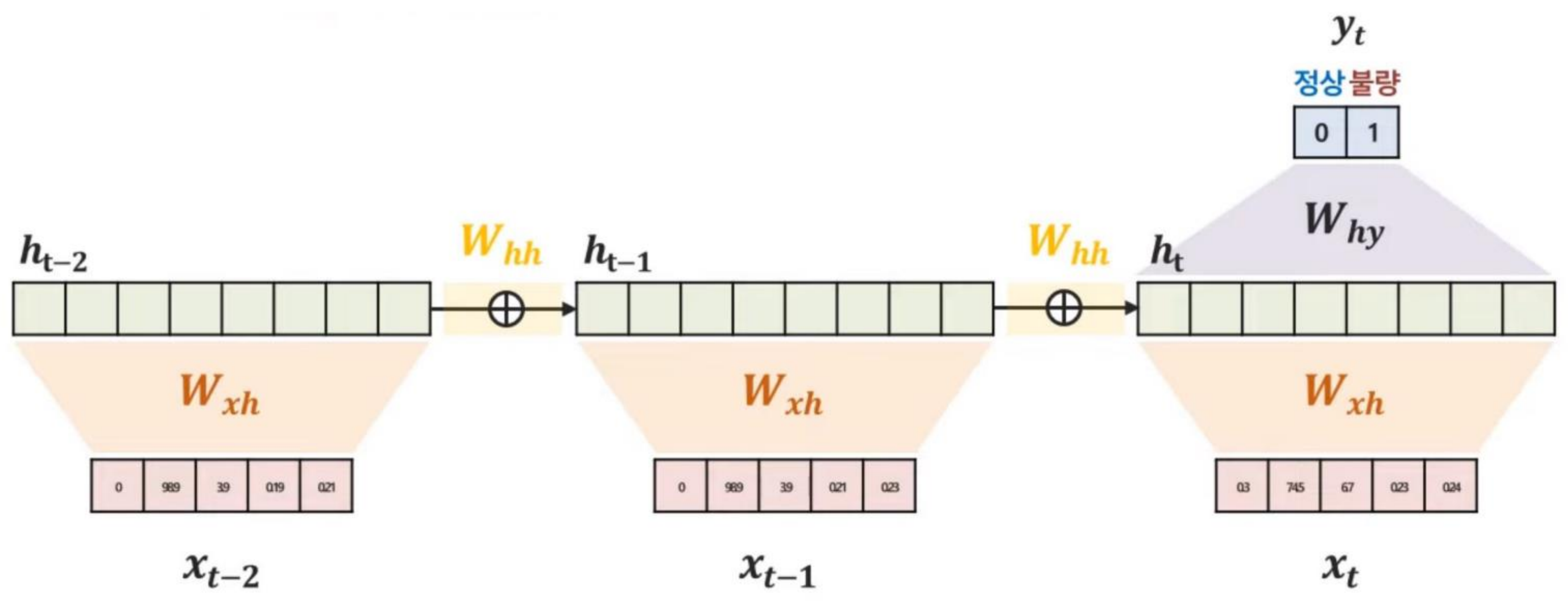
T_3: 시점 3 에서의 영향 T_2: 시점 3 으로부터 전해진 영향 고려 T_1: 시점 2 으로부터 전해진 영향 고려

$$\frac{\partial Loss}{\partial W_{hh}} = \boxed{\frac{\partial Loss}{\partial \hat{y}_t} \times \frac{\partial \hat{y}_t}{\partial \boldsymbol{h}_3} \times \frac{\partial \boldsymbol{h}_3}{\partial W_{hh}}} + \boxed{\frac{\partial Loss}{\partial \hat{y}_t} \times \frac{\partial \hat{y}_t}{\partial \boldsymbol{h}_3} \times \frac{\partial \boldsymbol{h}_3}{\partial \boldsymbol{h}_2} \times \frac{\partial \boldsymbol{h}_2}{\partial W_{hh}}} + \boxed{\frac{\partial Loss}{\partial \hat{y}_t} \times \frac{\partial \hat{y}_t}{\partial \boldsymbol{h}_3} \times \frac{\partial \boldsymbol{h}_3}{\partial \boldsymbol{h}_2} \times \frac{\partial \boldsymbol{h}_2}{\partial \boldsymbol{h}_1} \times \frac{\partial \boldsymbol{h}_1}{\partial W_{hh}}}$$

T_3: 시점 3 에서의 영향 T_2: 시점 3 으로부터 전해진 영향 고려 T_1: 시점 2 으로부터 전해진 영향 고려

$$\frac{\partial Loss}{\partial W_{xh}} = \boxed{\frac{\partial Loss}{\partial \hat{y}_t} \times \frac{\partial \hat{y}_t}{\partial \boldsymbol{h}_3} \times \frac{\partial \boldsymbol{h}_3}{\partial W_{xh}}} + \boxed{\frac{\partial Loss}{\partial \hat{y}_t} \times \frac{\partial \hat{y}_t}{\partial \boldsymbol{h}_3} \times \frac{\partial \boldsymbol{h}_3}{\partial \boldsymbol{h}_2} \times \frac{\partial \boldsymbol{h}_2}{\partial W_{xh}}} + \boxed{\frac{\partial Loss}{\partial \hat{y}_t} \times \frac{\partial y}{\partial \boldsymbol{h}_3} \times \frac{\partial \boldsymbol{h}_3}{\partial \boldsymbol{h}_2} \times \frac{\partial \boldsymbol{h}_2}{\partial \boldsymbol{h}_1} \times \frac{\partial \boldsymbol{h}_1}{\partial W_{xh}}}$$

RNN



최종 업데이트된 가중치

W_{hy}

$\rightarrow W_{hy}^{new} = W_{hy}^{old} - \eta * \frac{\partial Loss}{\partial W_{hy}}$

W_{hh}

$\rightarrow W_{hh}^{new} = W_{hh}^{old} - \eta * \frac{\partial Loss}{\partial W_{hh}}$

W_{xh}

$\rightarrow W_{xh}^{new} = W_{xh}^{old} - \eta * \frac{\partial Loss}{\partial W_{xh}}$

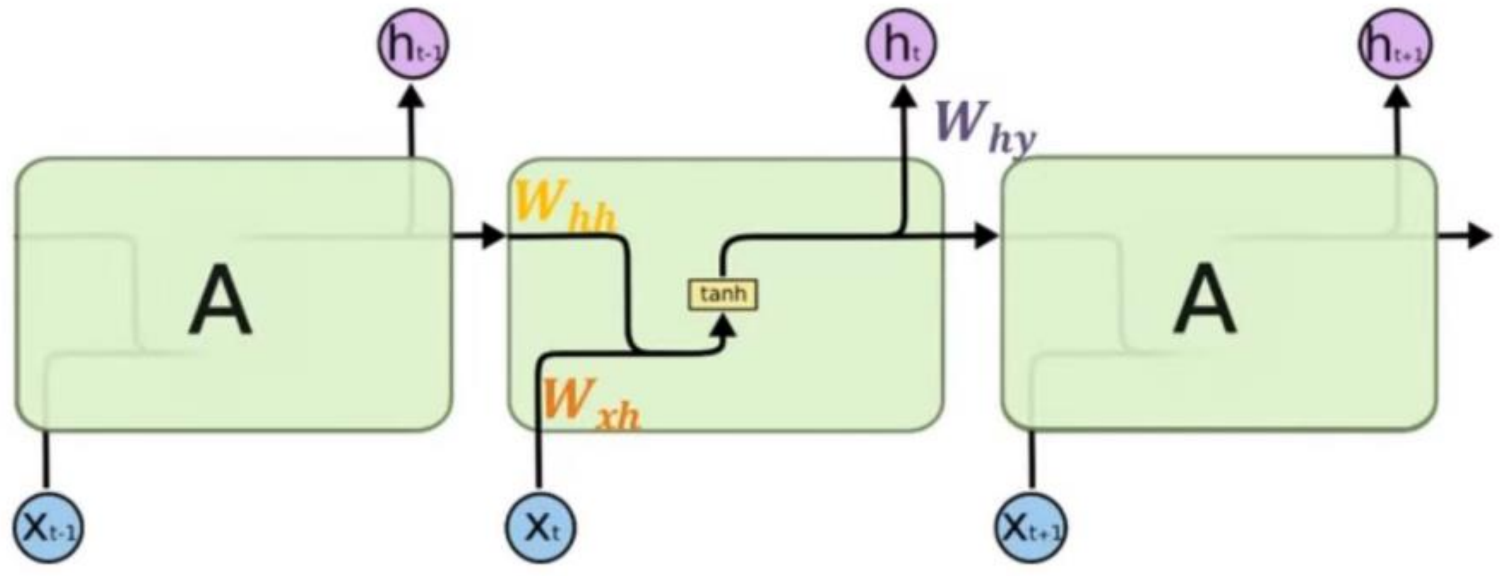
QUIZ 3) RNN이 가지는 한계점은?

LSTM

이예린

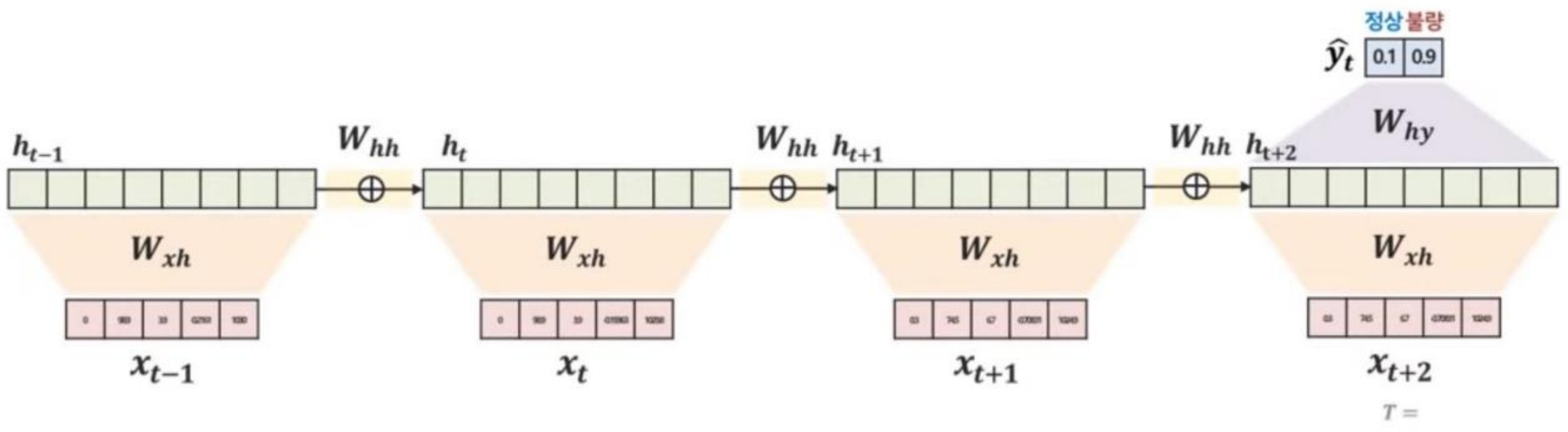
LSTM

RNN



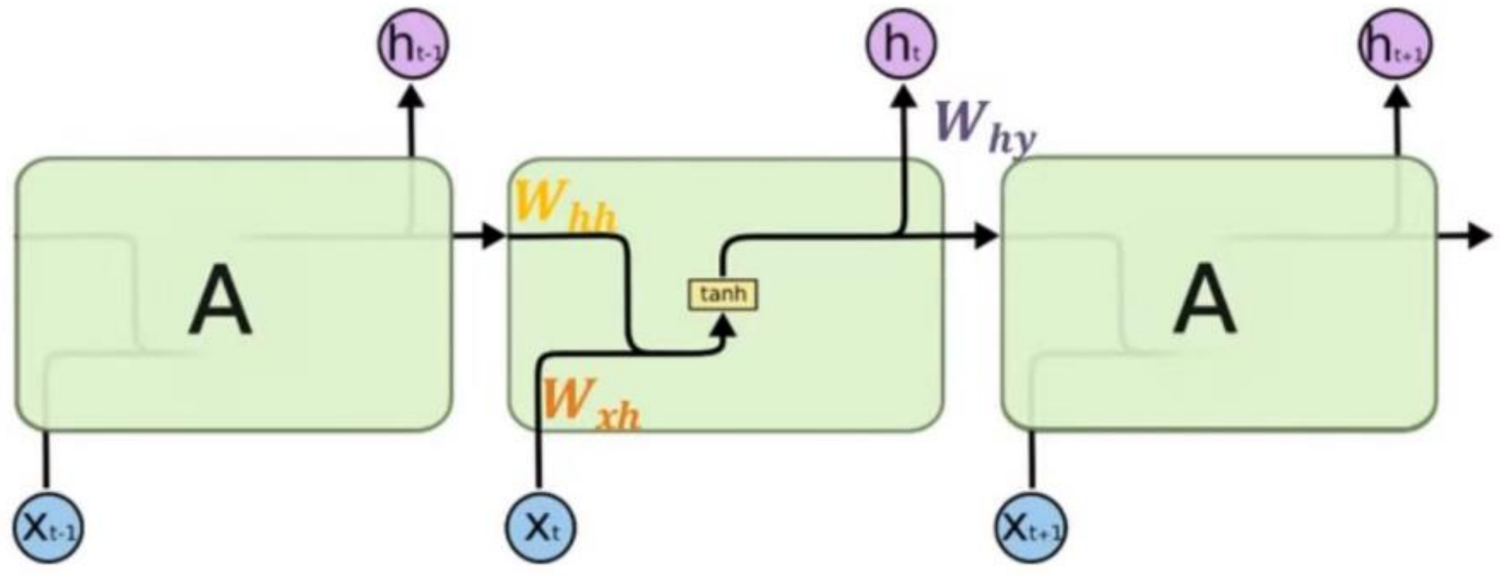
$$h_{t-1} = f(W_{xh}x_{t-1} + W_{hh}h_{t-2})$$
$$h_t = f(W_{xh}x_t + W_{hh}h_{t-1})$$
$$y_t = g(W_{hy}h_t)$$

Cell state
gate



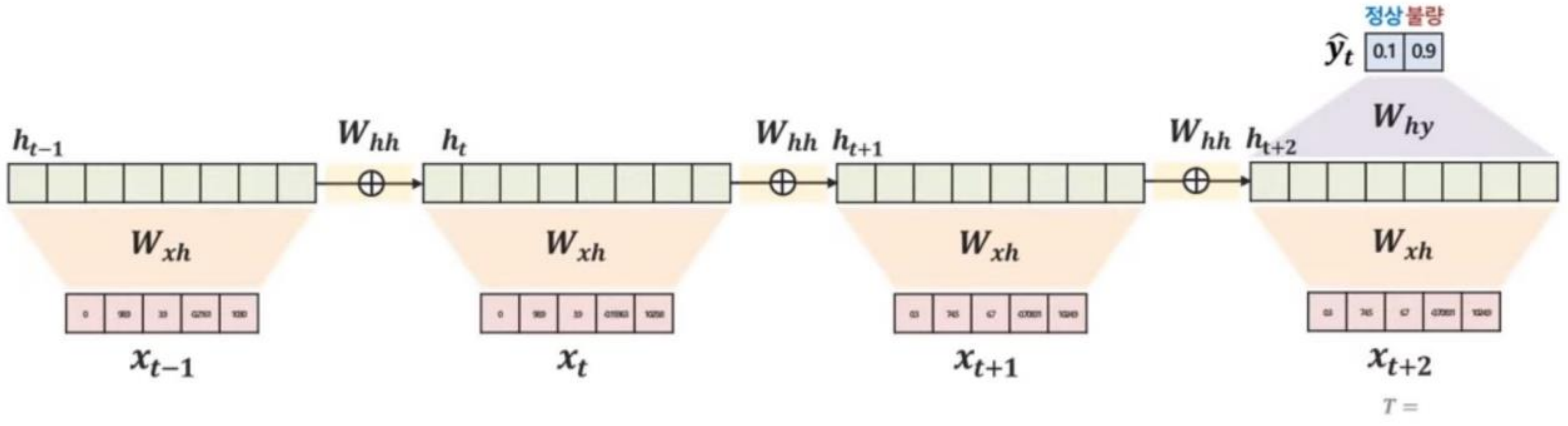
LSTM

RNN



$$h_{t-1} = f(W_{xh}x_{t-1} + W_{hh}h_{t-2})$$
$$h_t = f(W_{xh}x_t + W_{hh}h_{t-1})$$
$$y_t = g(W_{hy}h_t)$$

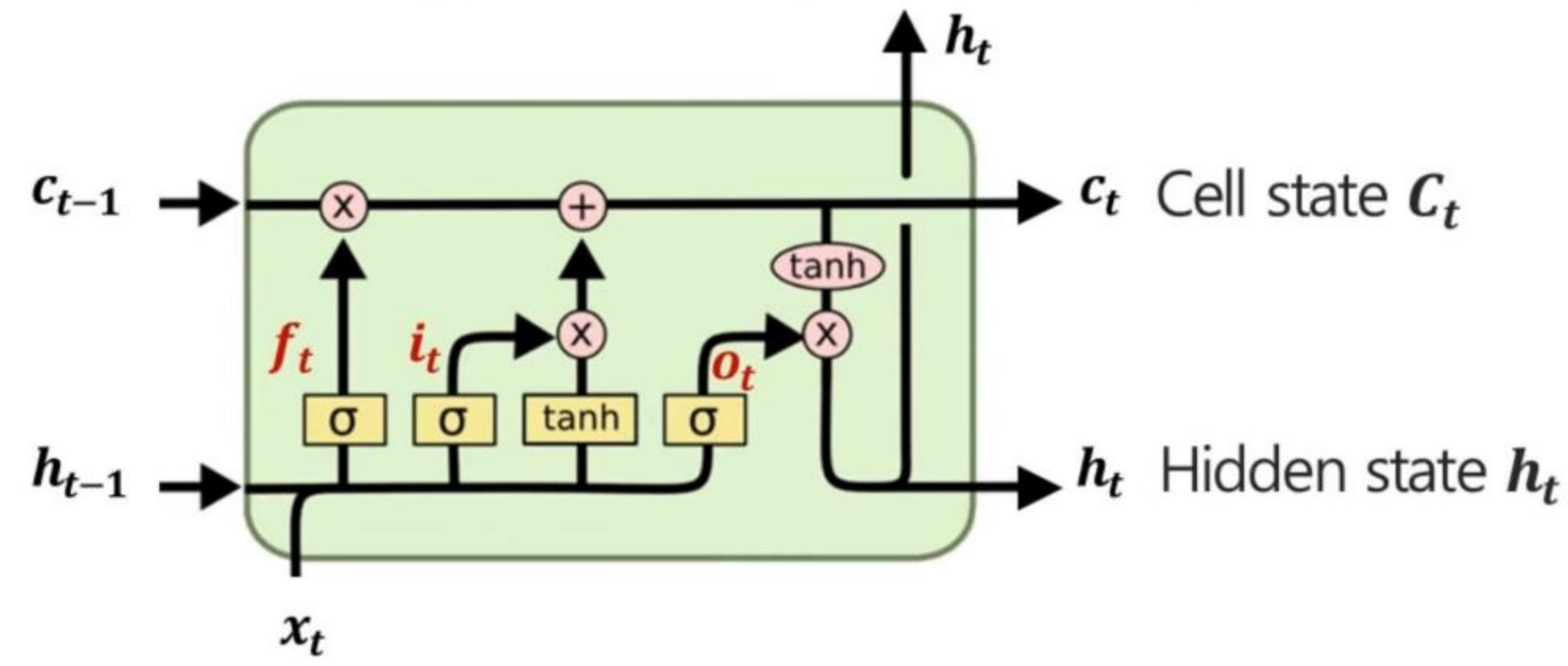
Cell state gate



LSTM

Cell state (c_t)구조

세가지 gate: Forget gate (f_t), Input gate (i_t), Output gate (o_t)



$$f_t = \sigma(W_{xh_f} x_t + W_{hh_f} h_{t-1} + b_{h_f})$$

$$i_t = \sigma(W_{xh_i} x_t + W_{hh_i} h_{t-1} + b_{h_i})$$

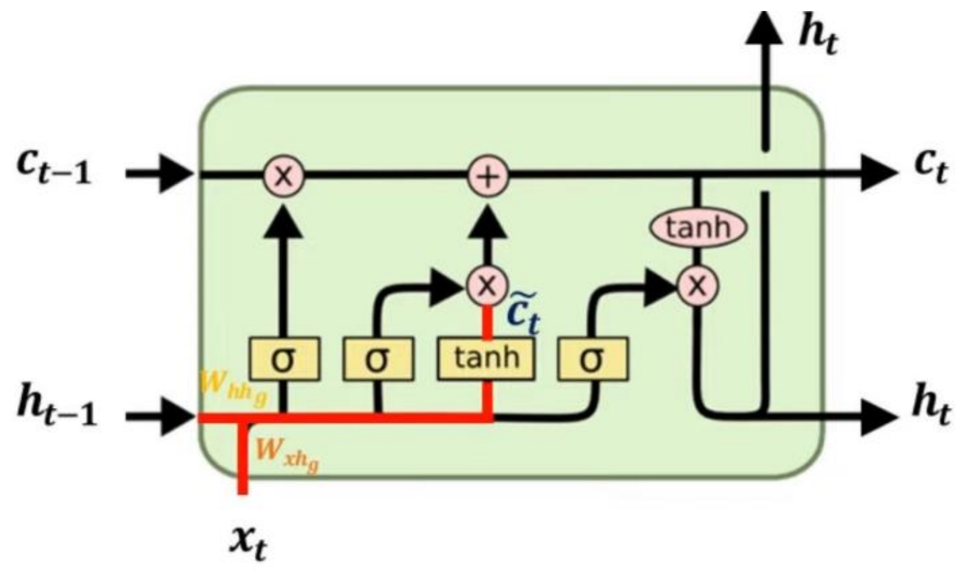
$$o_t = \sigma(W_{xh_o} x_t + W_{hh_o} h_{t-1} + b_{h_o})$$

$$\tilde{c}_t = tanh(W_{xh_g} x_t + W_{hh_g} h_{t-1} + b_{h_g})$$

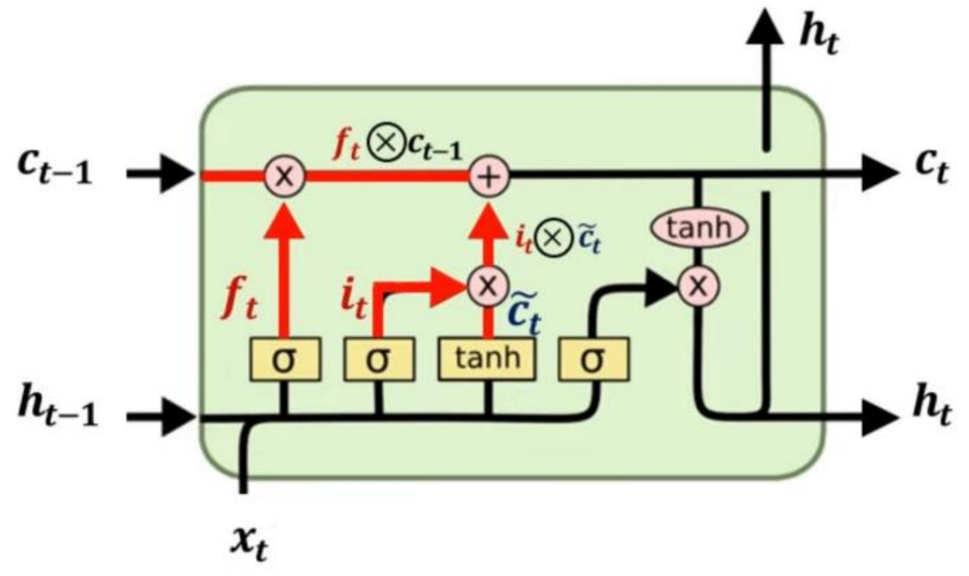
$$c_t = f_t \otimes c_{t-1} \oplus i_t \otimes \tilde{c}_t$$

$$h_t = o_t \otimes tanh(c_t)$$

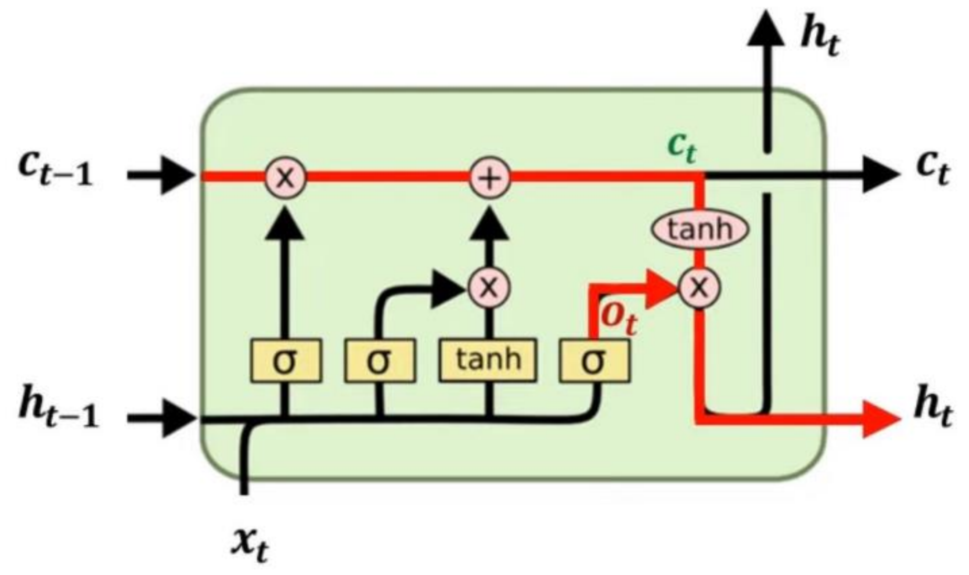
LSTM



$$\tilde{c}_t = tanh(W_{xh_g} x_t + W_{hh_g} h_{t-1} + b_{h_g})$$



$$c_t = f_t \otimes c_{t-1} \oplus i_t \otimes \tilde{c}_t$$



$$h_t = o_t \otimes tanh(c_t)$$

LSTM

Gate 계산 -〉 cell state 업데이트 -〉 hidden vector 업데이트

$$f_t = \sigma(W_{xh_f} x_t + W_{hh_f} h_{t-1} + b_{h_f})$$ forget gate

$$i_t = \sigma(W_{xh_i} x_t + W_{hh_i} h_{t-1} + b_{h_i})$$ input gate

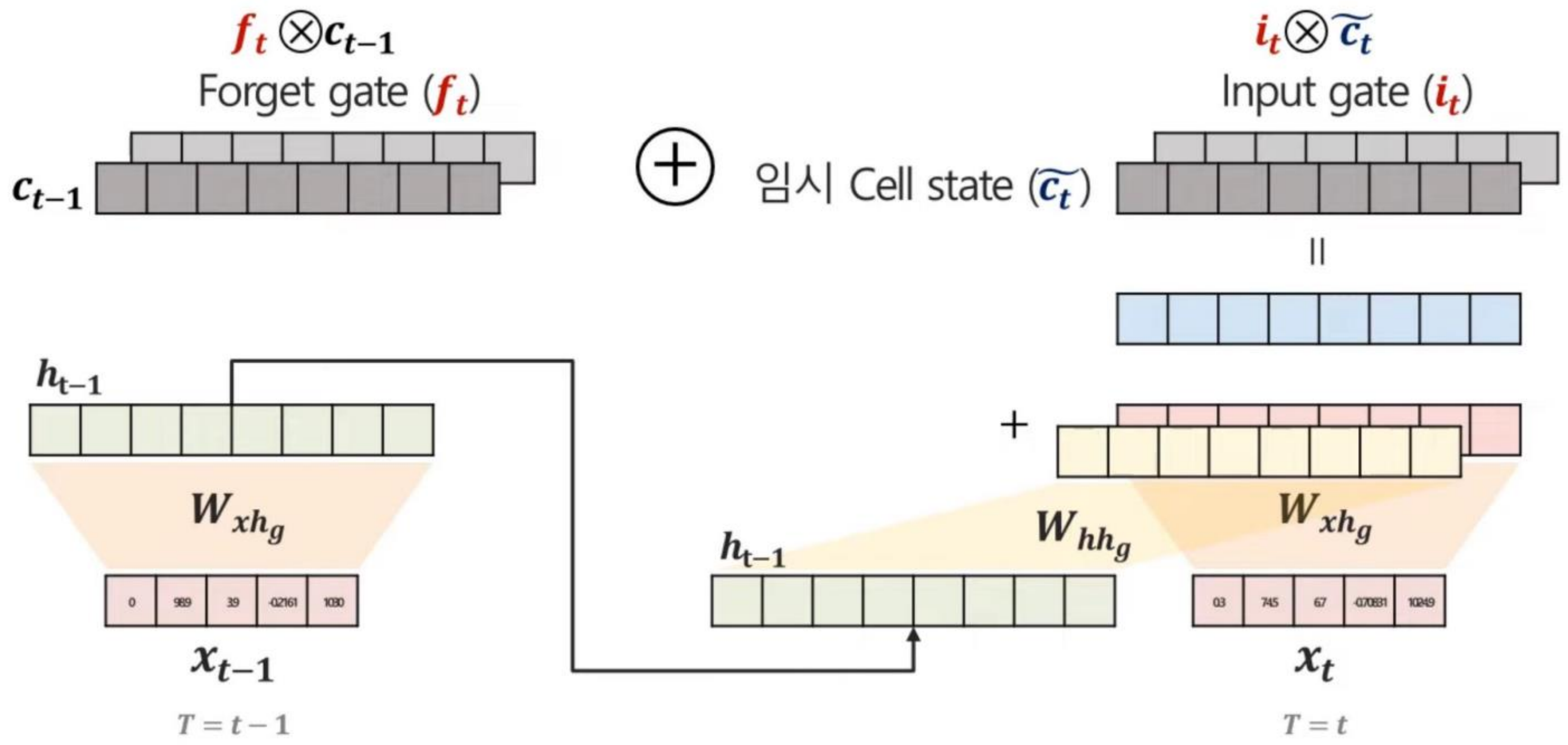
$$o_t = \sigma(W_{xh_o} x_t + W_{hh_o} h_{t-1} + b_{h_o})$$ output gate

: 0~1 값 -〉 일종의 확률

LSTM

Quiz 4) forget gate와 input gate는 어떤 시점의 정보를 조정하는 가중치일까?

ell state $c_t = f_t \otimes c_{t-1} \oplus i_t \otimes \tilde{c}_t$, $\otimes$= elementwise product

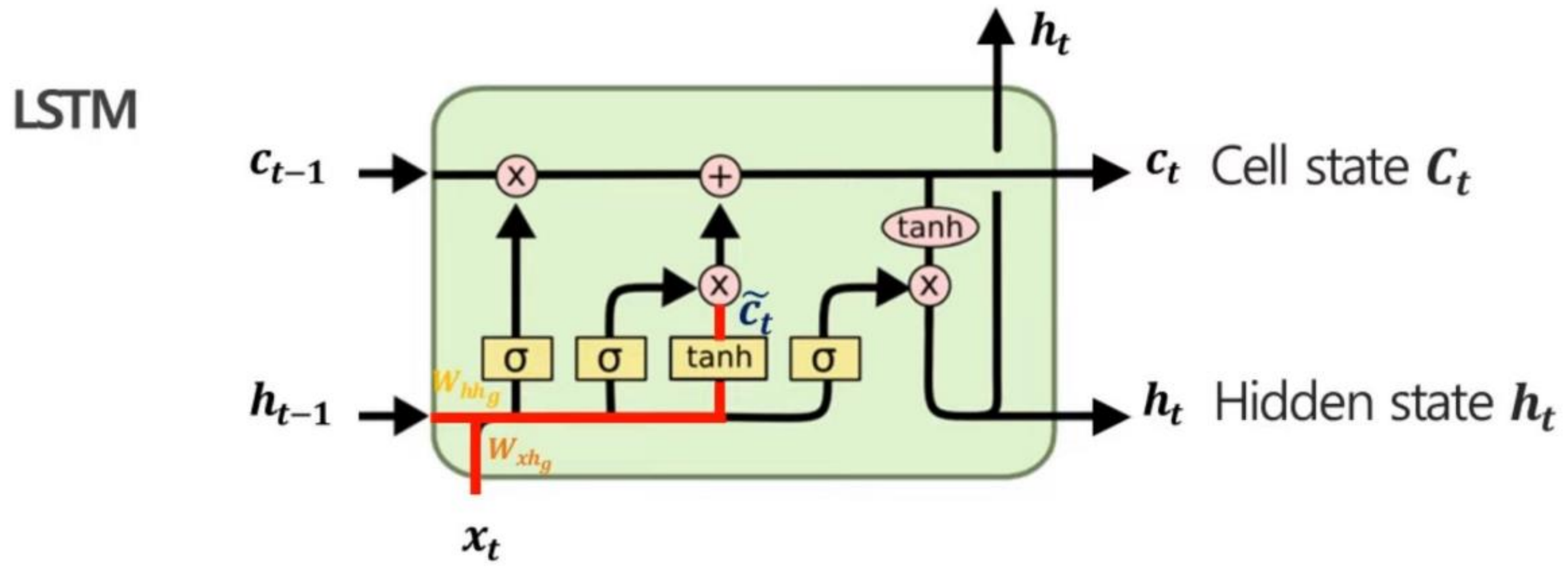


LSTM

- Output Gate $o_t = \sigma(W_{xh_o} x_t + W_{hh_o} h_{t-1} + b_{h_o})$
- Hidden state $h_t = o_t \odot tanh(c_t)$

o_t	0.5	0.4	0.1	0.9	0.2	0.3	0.8	0.7	0.1	0.9
c_t	-0.01	0.02	0.93	0.11	1.62	0.01	0.29	3.69	0.36	0.43
$tanh(c_t)$	-0.01	0.02	0.73	0.11	0.92	0.01	0.28	1.00	0.35	0.41
$h_t = o_t \odot tanh(c_t)$	0	0.08	0.073	0.099	0.184	0.003	0.224	0.7	0.035	0.369

LSTM



$$f_t = \sigma(W_{xh_f} x_t + W_{hh_f} h_{t-1} + b_{h_f})$$

$$i_t = \sigma(W_{xh_i} x_t + W_{hh_i} h_{t-1} + b_{h_i})$$

$$o_t = \sigma(W_{xh_o} x_t + W_{hh_o} h_{t-1} + b_{h_o})$$

$$\boxed{\tilde{c}_t = tanh(W_{xh_g} x_t + W_{hh_g} h_{t-1} + b_{h_g})}$$

$$c_t = f_t \otimes c_{t-1} \oplus i_t \otimes \tilde{c}_t$$

$$h_t = o_t \otimes tanh(c_t)$$

Hidden state h_t: 현 시점에 대한 단기적인 정보

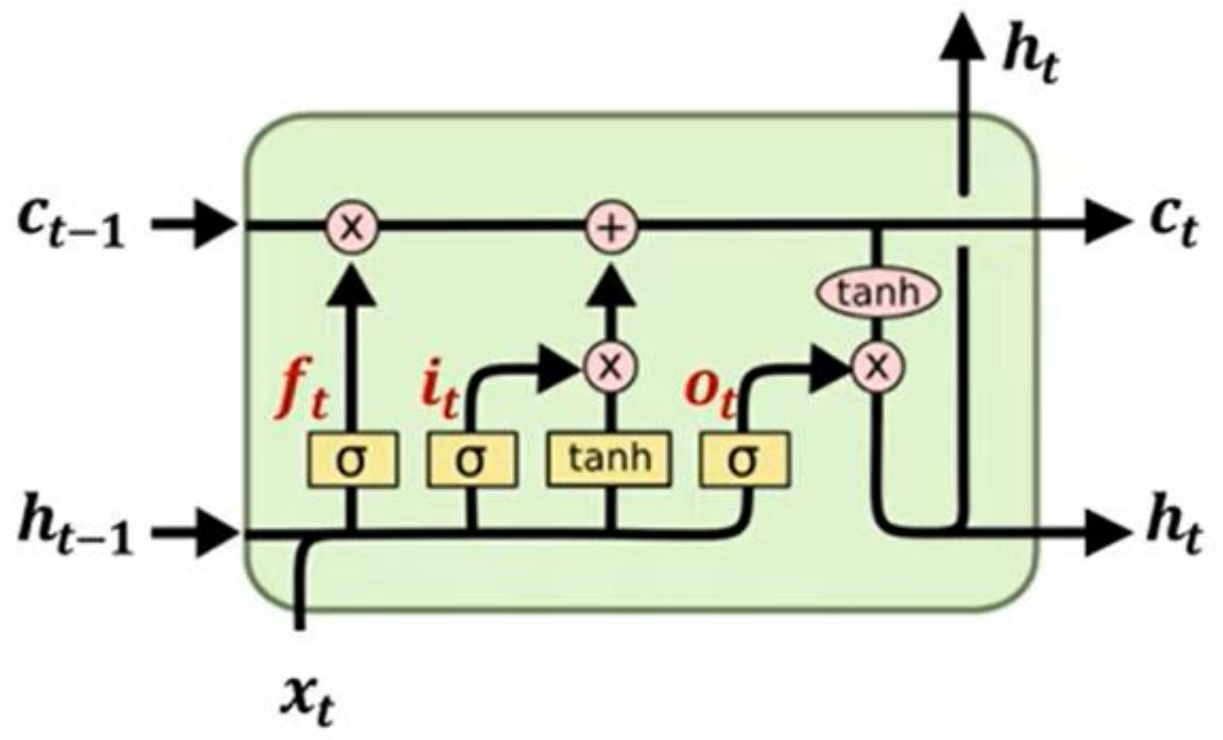
Cell state c_t: 현 시점에 대한 장기적인 정보

Peephole connection

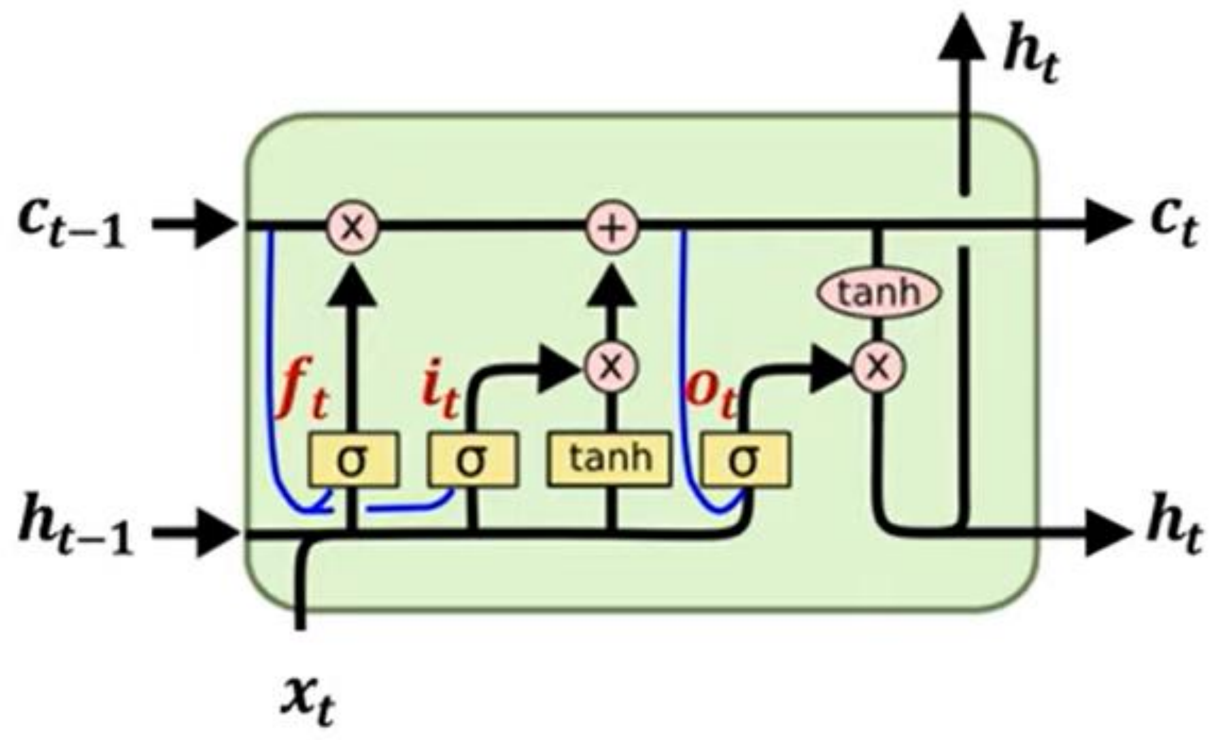
Gate에 장기정보(cell state) c_t 정보도 활용하여 더 많은 정보가 수용되도록 개선

- Forget gate $f_t = \sigma(W_{xh_f} x_t + W_{hh_f} h_{t-1} + W_{ch_f} c_{t-1} + b_{h_f})$
- Input gate $i_t = \sigma(W_{xh_i} x_t + W_{hh_i} h_{t-1} + W_{ch_i} c_{t-1} + b_{h_i})$
- Output gate $o_t = \sigma(W_{xh_o} x_t + W_{hh_o} h_{t-1} + W_{ch_o} c_t + b_{h_o})$

추가



LSTM



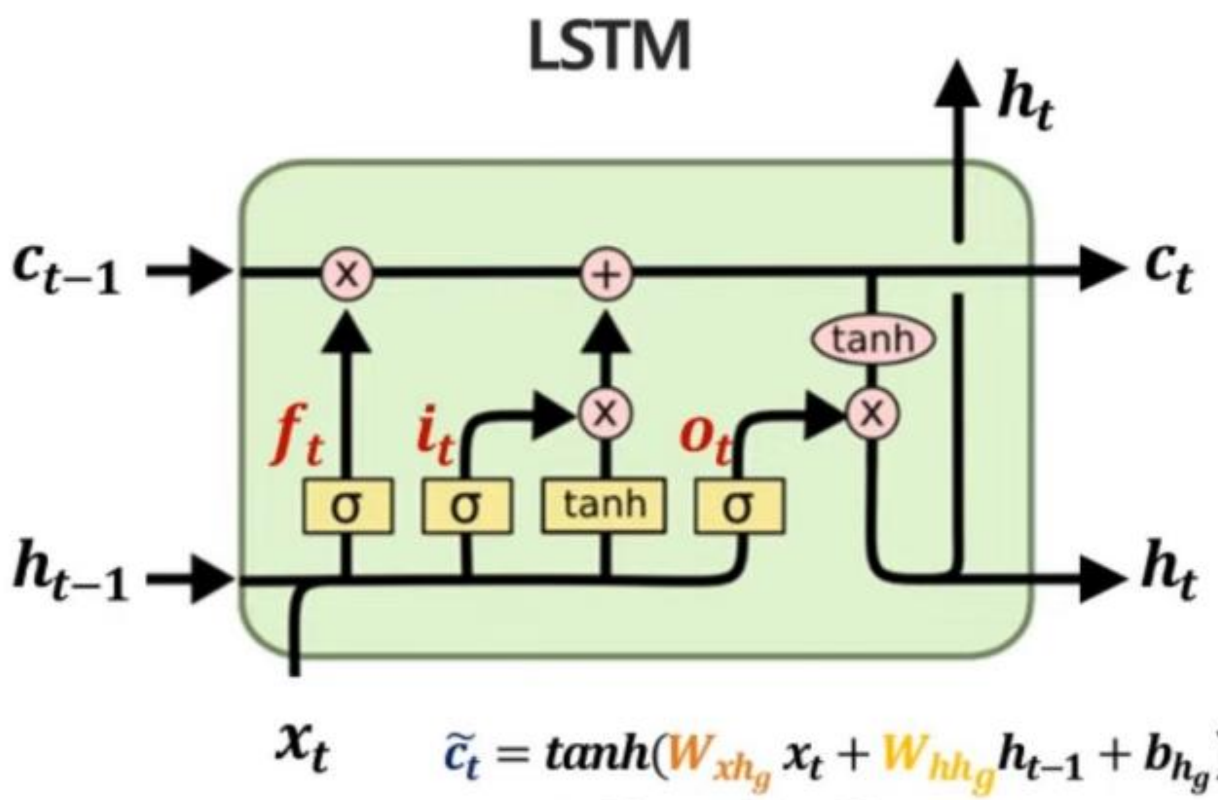
LSTM with peephole connections

GRU

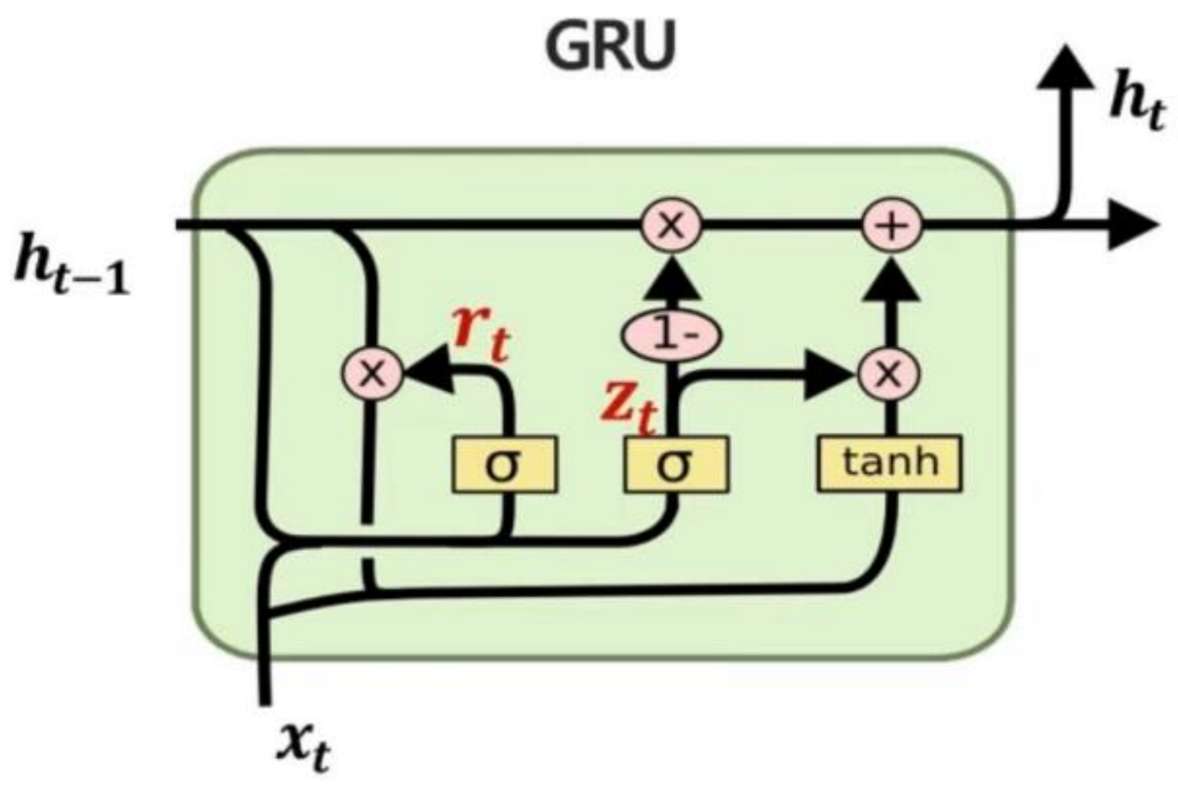
이예린

GRU

- forget gate, input gate를 update gate(z_t)로 통합, output gate를 없애고 reset gate(r_t)정의
- Cell state, hidden state를 hidden state로 통합



$$\tilde{c}_t = tanh(W_{xh_g} x_t + W_{hh_g} h_{t-1} + b_{h_g})$$
$$c_t = f_t \otimes c_{t-1} \oplus i_t \otimes \tilde{c}_t$$
$$h_t = o_t \otimes tanh(c_t)$$



GRU

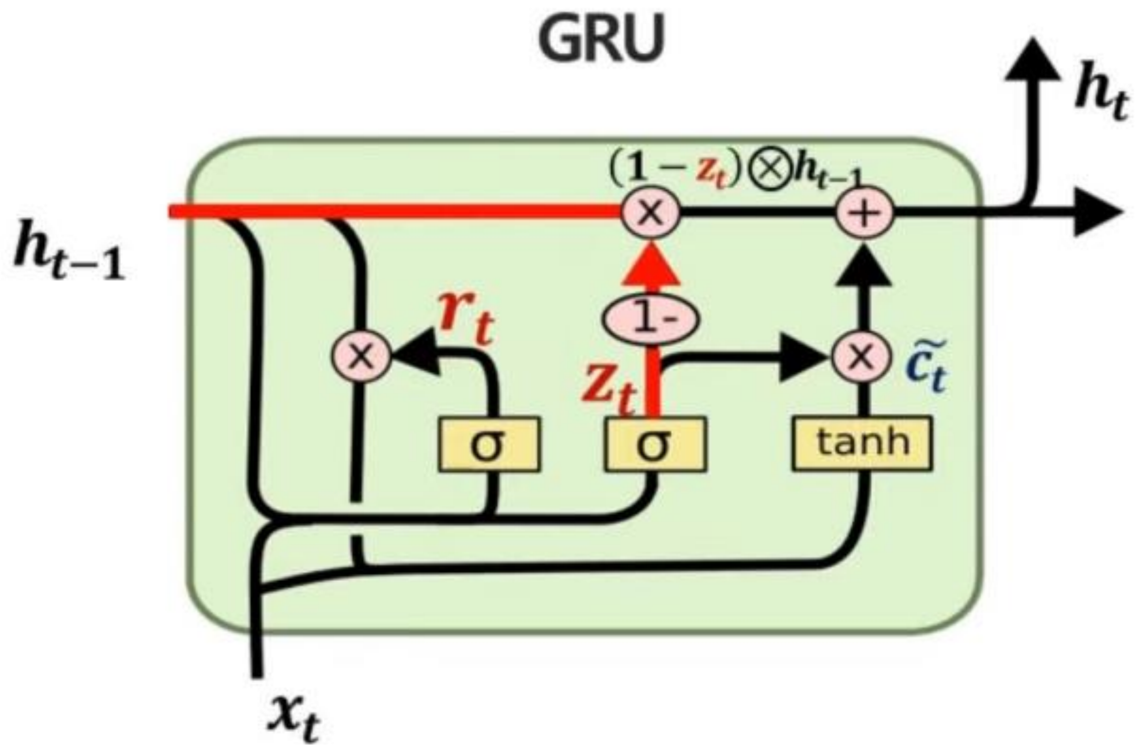
Output gate를 없애고 reset gate(r_t)정의, forget, input gate를 update gate(z_t)로 통합

- Reset gate $r_t = \sigma(W_{xh_r} x_t + W_{hh_r} h_{t-1} + b_{h_r})$
- Update gate $z_t = \sigma(W_{xh_z} x_t + W_{hh_z} h_{t-1} + b_{h_z})$
- 임시 cell state $\tilde{c_t} = tanh(W_{xh_g} x_t + W_{hh_g}(r_t \otimes h_{t-1}) + b_{h_g})$

Cell, hidden state를 hidden state로 통합

- 최종 cell state (=hidden state) $h_t = (1 - z_t) \otimes h_{t-1} \oplus z_t \otimes \tilde{c_t}$

Forget gate의 역할 Input gate의 역할



임시 cell state $\tilde{c_t} = tanh(W_{xh_g} x_t + W_{hh_g}(r_t \otimes h_{t-1}) + b_{h_g})$

최종 cell state (=hidden state) $h_t = (1 - z_t) \otimes h_{t-1} \oplus z_t \otimes \tilde{c_t}$

영화 리뷰데이터 실습

이예린

실습

1. 라이브러리 호출

```python
import torch
import torchtext
import numpy as np
import torch.nn as nn
import torch.nn.functional as F
import time
```

2. 데이터셋 다운로드 & 전처리

```python
start=time.time()

TEXT = torchtext.legacy.data.Field(sequential = True, batch_first = True, lower = True)
LABEL = torchtext.legacy.data.Field(sequential = False, batch_first = True)

from torchtext.legacy import datasets
train_data, test_data = datasets.IMDB.splits(TEXT, LABEL)
train_data, valid_data = train_data.split(split_ratio = 0.8)

TEXT.build_vocab(train_data, max_size=10000, min_freq=10, vectors=None)
LABEL.build_vocab(train_data)
```

실습

3. 데이터셋 분리

```
BATCH_SIZE = 100
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')

train_iterator, valid_iterator, test_iterator = torchtext.legacy.data.BucketIterator.splits(
    (train_data, valid_data, test_data),
    batch_size = BATCH_SIZE,
    device = device)
```

4. 변수 값 지정

```
vocab_size = len(TEXT.vocab) # 고유한 단어의 수
n_classes = 2   # 분류 문제에 대한 클래스 수
```

실습

5. RNN 계층 네트워크

```
class BasicRNN(nn.Module):                    # n_vocab: 단어 집합의 크기
    def __init__(self, n_layers, hidden_dim, n_vocab, embed_dim, n_classes, dropout_p = 0.2):
        super(BasicRNN, self).__init__()
        self.n_layers = n_layers                        # RNN의 계층 수
        self.embed = nn.Embedding(n_vocab, embed_dim)   # 임베딩(embedding) 차원의 크기
        self.hidden_dim = hidden_dim                    # 은닉 상태(hidden state)의 크기
        self.dropout = nn.Dropout(dropout_p)            # 드롭아웃(dropout) 확률
        self.rnn = nn.RNN(embed_dim, self.hidden_dim, num_layers = self.n_layers, batch_first = True)
        self.out = nn.Linear(self.hidden_dim, n_classes) # n_classes: 클래스(레이블)의 개수


    def forward(self, x):        # x: 입력 데이터로, 시퀀스의 인덱스들로 이루어진 텐서입니다.
        x = self.embed(x)        # 임베딩 적용(문자를 숫자/벡터로 변환)
        h_0 = self._init_state(batch_size = x.size(0)) # RNN의 초기 은닉 상태를 0으로 초기화
Quiz 5) x, _ =                 # 입력값과 이전 시점의 은닉 상태를 받음
        h_t = x[:, -1, :]        # RNN의 마지막 시점에서의 출력값을 선택
        self.dropout(h_t)        # 드롭아웃을 적용
        logit = torch.sigmoid(self.out(h_t)) # 0과 1 사이의 확률값으로 변환
        return logit

    # RNN 계층의 초기 은닉 상태 텐서를 생성하는 함수. 이 함수는 입력 텐서의 배치 크기에 맞춰 크기가 조절됩니
다.
    def _init_state(self, batch_size = 1):
        weight = next(self.parameters()).data
        return weight.new(self.n_layers, batch_size, self.hidden_dim).zero_() # RNN의 초기 은닉 상태를 생성하
여 반환
```

실습

6. 손실함수 & 옵티마이저 설정

```
model = BasicRNN(n_layers = 1, hidden_dim = 256, n_vocab = vocab_size, embed_dim = 128, n_classes = n_classes,
dropout_p = 0.5)
model.to(device)

loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.0001)
```

7. 모델 학습

```
def train(model, optimizer, train_iter):
    model.train() # 학습 모드

    for b, batch in enumerate(train_iter): # b: 현재 배치의 인덱스 / batch: 배치 데이터
        x, y = batch.text.to(device), batch.label.to(device) # x: text / y: label
        y.data.sub_(1) # label 값을 2/1 -> 1/0 으로 변경
        optimizer.zero_grad() # 옵티마이저 초기화

        logit = model(x) #
        loss = F.cross_entropy(logit, y)
        loss.backward()
        optimizer.step()

        if b % 50 == 0:
            print("Train Epoch: {} [{}/{} ({:.0f}%)]₩tLoss: {:.6f}".format(e,
                                                                           b * len(x),
                                                                           len(train_iter.dataset),
                                                                           100. * b / len(train_iter),
                                                                           loss.item()))
```

실습

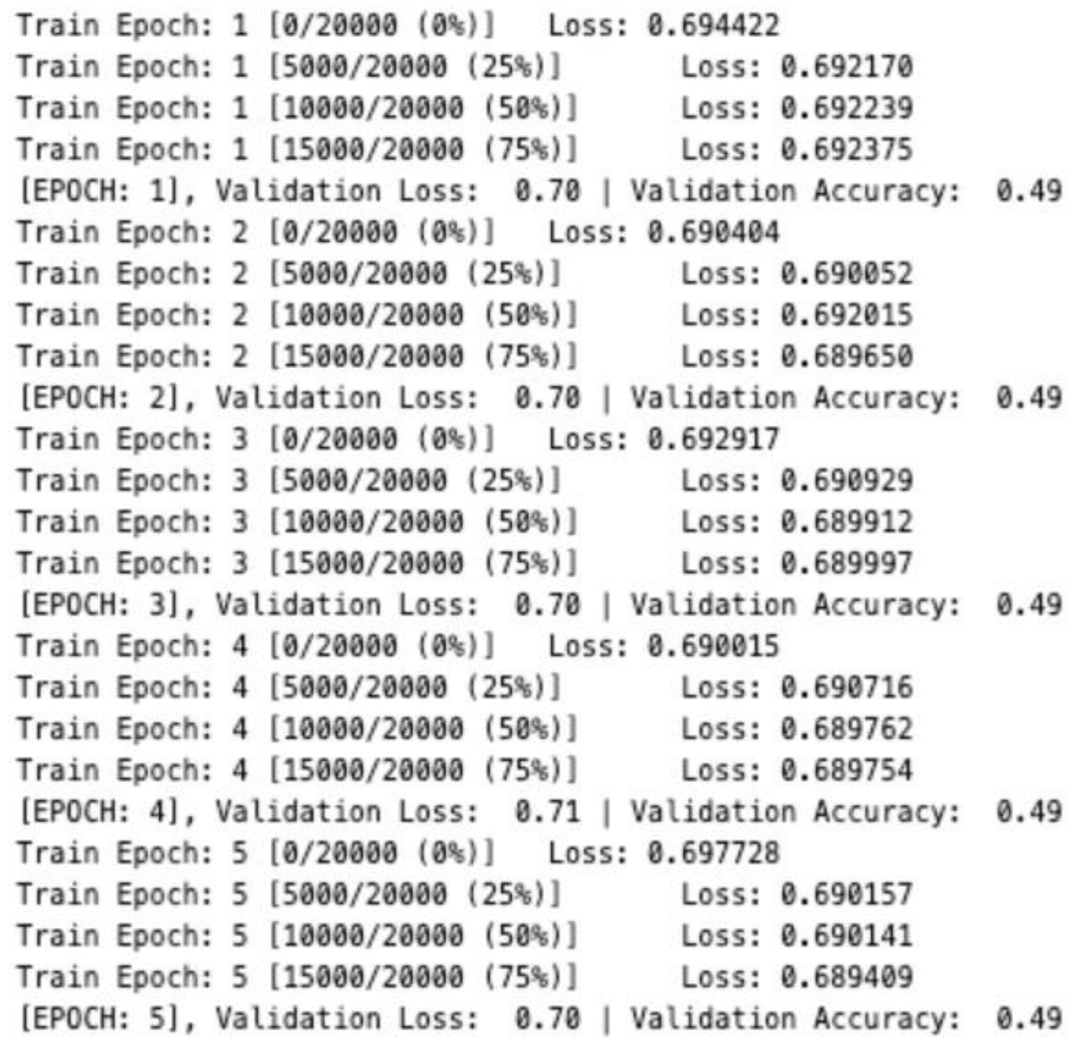
8. 모델 평가

```python
def evaluate(model, val_iter):
    model.eval() # 평가 모드

    corrects, total, total_loss = 0, 0, 0

    for batch in val_iter:
        x, y = batch.text.to(device), batch.label.to(device)
        y.data.sub_(1)
        logit = model(x)
        loss = F.cross_entropy(logit, y, reduction = "sum")
        total += y.size(0)
        total_loss += loss.item()
        corrects += (logit.max(1)[1].view(y.size()).data == y.data).sum()

    avg_loss = total_loss / len(val_iter.dataset)
    avg_accuracy = corrects / total
    return avg_loss, avg_accuracy
```

실습

9. 모델 학습 & 평가

```python
BATCH_SIZE = 100
LR = 0.001
EPOCHS = 5
for e in range(1, EPOCHS + 1):
    train(model, optimizer, train_iterator)
    val_loss, val_accuracy = evaluate(model, valid_iterator)
    print("[EPOCH: %d], Validation Loss: %5.2f | Validation Accuracy: %5.2f" % (e, val_loss, val_accuracy))
```



```
Train Epoch: 1 [0/20000 (0%)]	Loss: 0.694422
Train Epoch: 1 [5000/20000 (25%)]	Loss: 0.692170
Train Epoch: 1 [10000/20000 (50%)]	Loss: 0.692239
Train Epoch: 1 [15000/20000 (75%)]	Loss: 0.692375
[EPOCH: 1], Validation Loss:  0.70 | Validation Accuracy:  0.49
Train Epoch: 2 [0/20000 (0%)]	Loss: 0.690404
Train Epoch: 2 [5000/20000 (25%)]	Loss: 0.690052
Train Epoch: 2 [10000/20000 (50%)]	Loss: 0.692015
Train Epoch: 2 [15000/20000 (75%)]	Loss: 0.689650
[EPOCH: 2], Validation Loss:  0.70 | Validation Accuracy:  0.49
Train Epoch: 3 [0/20000 (0%)]	Loss: 0.692917
Train Epoch: 3 [5000/20000 (25%)]	Loss: 0.690929
Train Epoch: 3 [10000/20000 (50%)]	Loss: 0.689912
Train Epoch: 3 [15000/20000 (75%)]	Loss: 0.689997
[EPOCH: 3], Validation Loss:  0.70 | Validation Accuracy:  0.49
Train Epoch: 4 [0/20000 (0%)]	Loss: 0.690015
Train Epoch: 4 [5000/20000 (25%)]	Loss: 0.690716
Train Epoch: 4 [10000/20000 (50%)]	Loss: 0.689762
Train Epoch: 4 [15000/20000 (75%)]	Loss: 0.689754
[EPOCH: 4], Validation Loss:  0.71 | Validation Accuracy:  0.49
Train Epoch: 5 [0/20000 (0%)]	Loss: 0.697728
Train Epoch: 5 [5000/20000 (25%)]	Loss: 0.690157
Train Epoch: 5 [10000/20000 (50%)]	Loss: 0.690141
Train Epoch: 5 [15000/20000 (75%)]	Loss: 0.689409
[EPOCH: 5], Validation Loss:  0.70 | Validation Accuracy:  0.49
```

실습

10. test 데이터셋을 이용한 모델 예측

```
test_loss, test_acc = evaluate(model,test_iterator)
print("Test Loss: %5.2f | Test Accuracy: %5.2f" % (test_loss, test_acc))
```

```
> Test Loss: 0.68 | Test Accuracy: 0.61
```

참고자료

- [핵심 머신러닝] RNN, LSTM, and GRU (youtube.com)
- [pytorch] RNN 계층 구현하기 (tistory.com)
- 딥러닝 파이토치 교과서