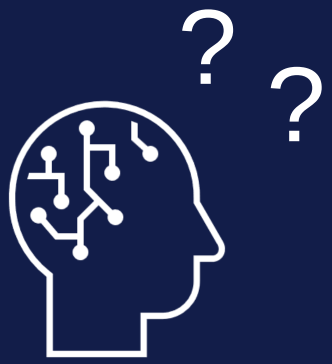
Week 1

DL Basic & RNN

HAI TEAM 5



Contents

팀원소개

팀원	학과	학년	체크사항
김한결	컴퓨터소프트웨어학과	3	하이웹(프론트엔드)
강민성	컴퓨터소프트웨어학과	1	알고리즘 경험 有
박혁주	수학과	3	NLP 및 추천 관심
조하준	컴퓨터소프트웨어학과	1	인공지능 관련 경험 HAI 1학기
박도연	수학과	1	NLP, 타 분야도 경험 원함

일정

- 일요일 19:00 구글 미팅

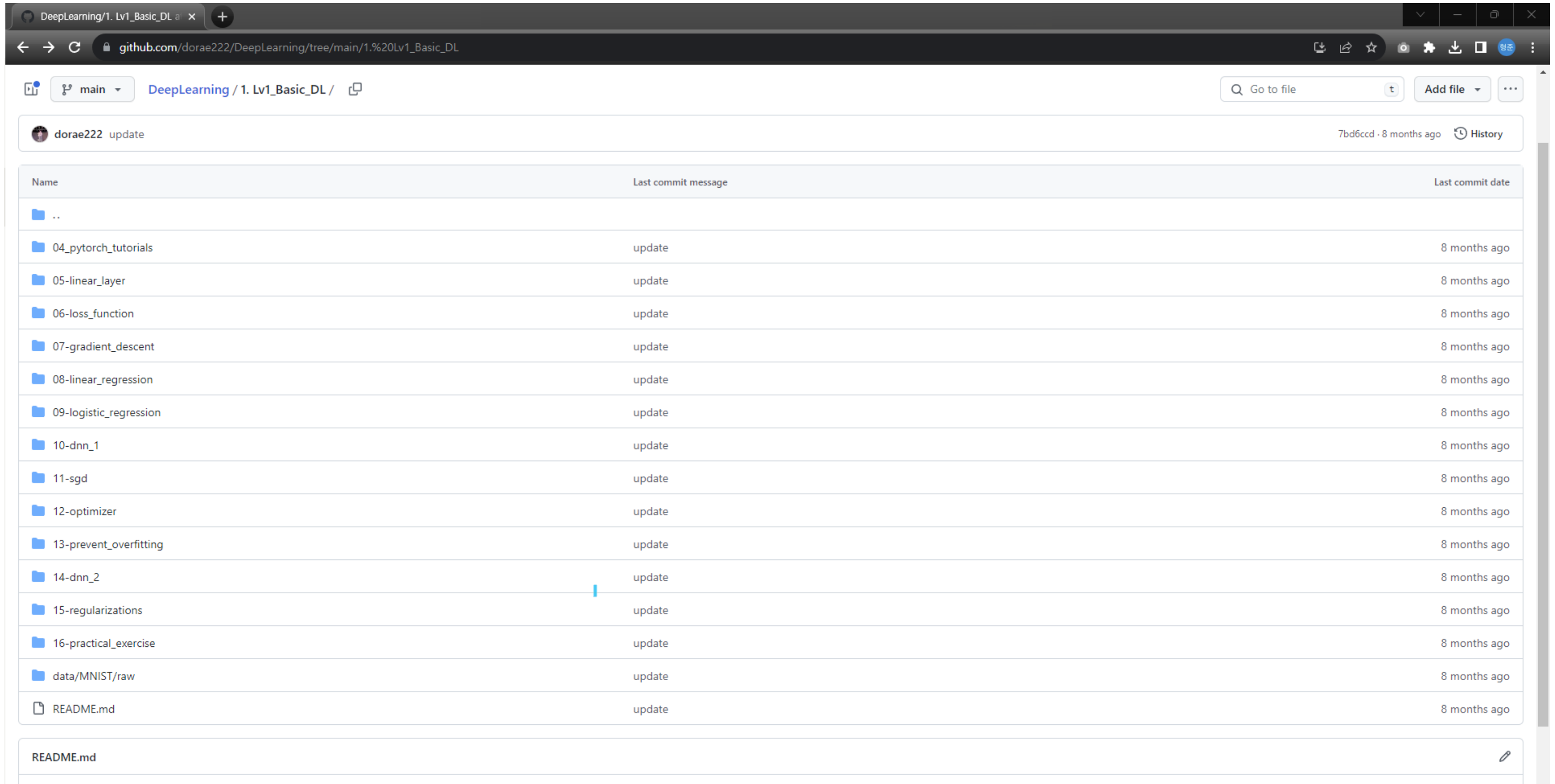
중간고사 이전 일정

9/17	DL Basic, RNN
9/24	LSTM, Seq2Seq
10/01	Seq2Seq+Attention, Transformer

중간고사 이후 일정

1	미정
2	미정
3	미정

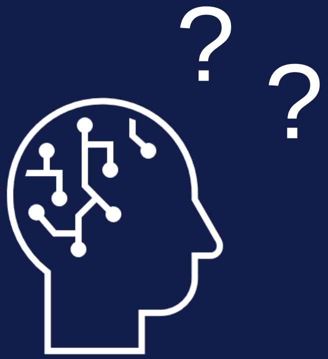
DL Basic
Pytorch 기초 문법



1. https://tutorials.pytorch.kr/recipes/recipes_index.html
2. https://github.com/dorae222/DeepLearning/tree/main/1.%20Lv1_Basic_DL

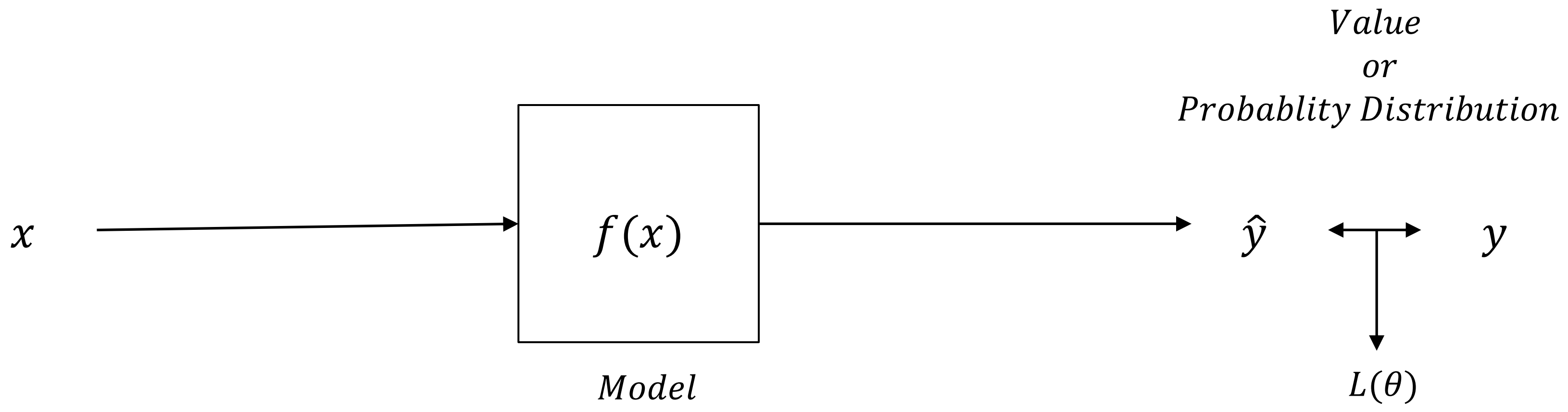
DL Basic

1. Our Objective with DL
2. Input Data
3. Model(Perceptron, ANN, DNN)
4. Whole Process with DL



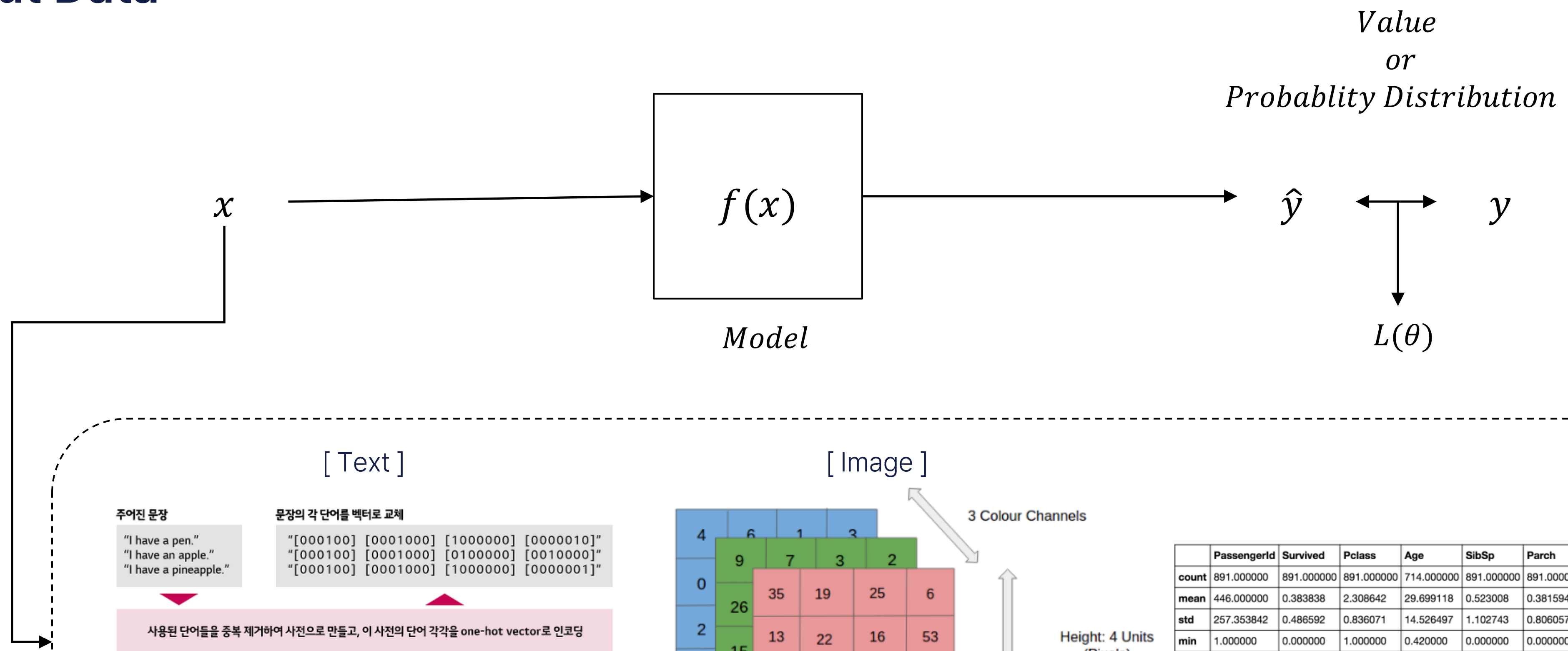
DL Basic

Our Objective

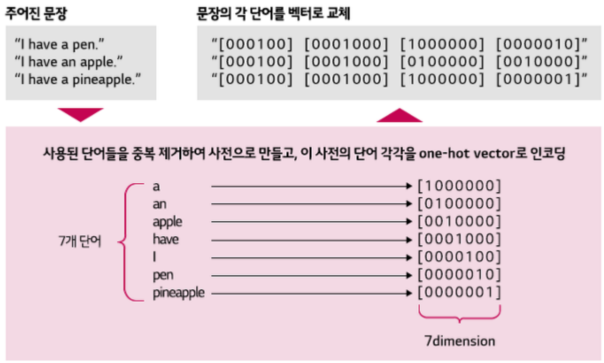


DL Basic

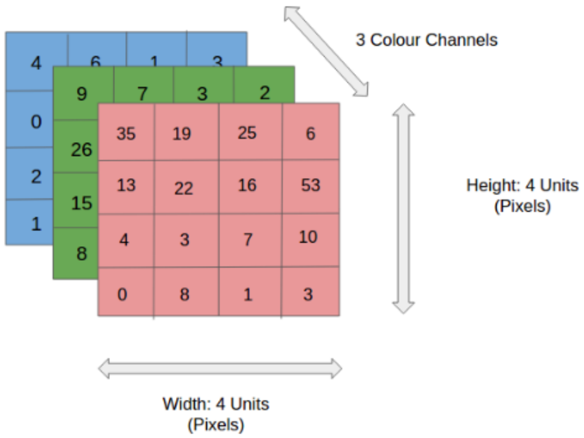
Input Data



[Text]



[Image]



	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

사진 출처: https://www.insilicogen.com/blog/346 , https://www.lgcns.com/blog/cns-tech/ai-data/2343/

DL Basic

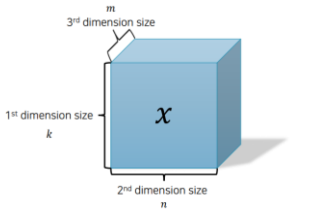
Input Data

Value
or
Probablity Distribution

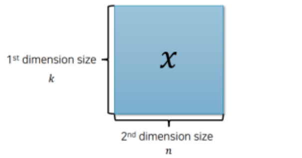
x → $f(x)$ → $\hat{y}$ ↔ y

Model

$L(\theta)$

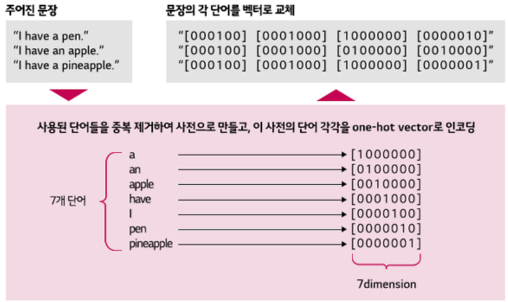


$x \in \mathbb{R}^{k \times n \times m}$ ▶ $|x| = (k, n, m)$

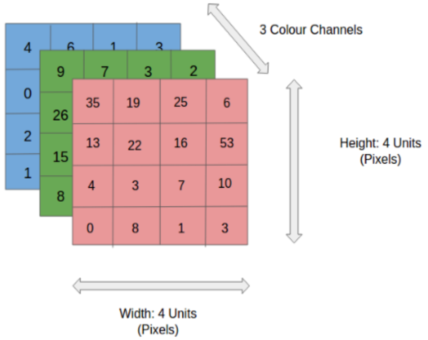


$x \in \mathbb{R}^{k \times n}$ ▶ $|x| = (k, n)$

[Text]



[Image]

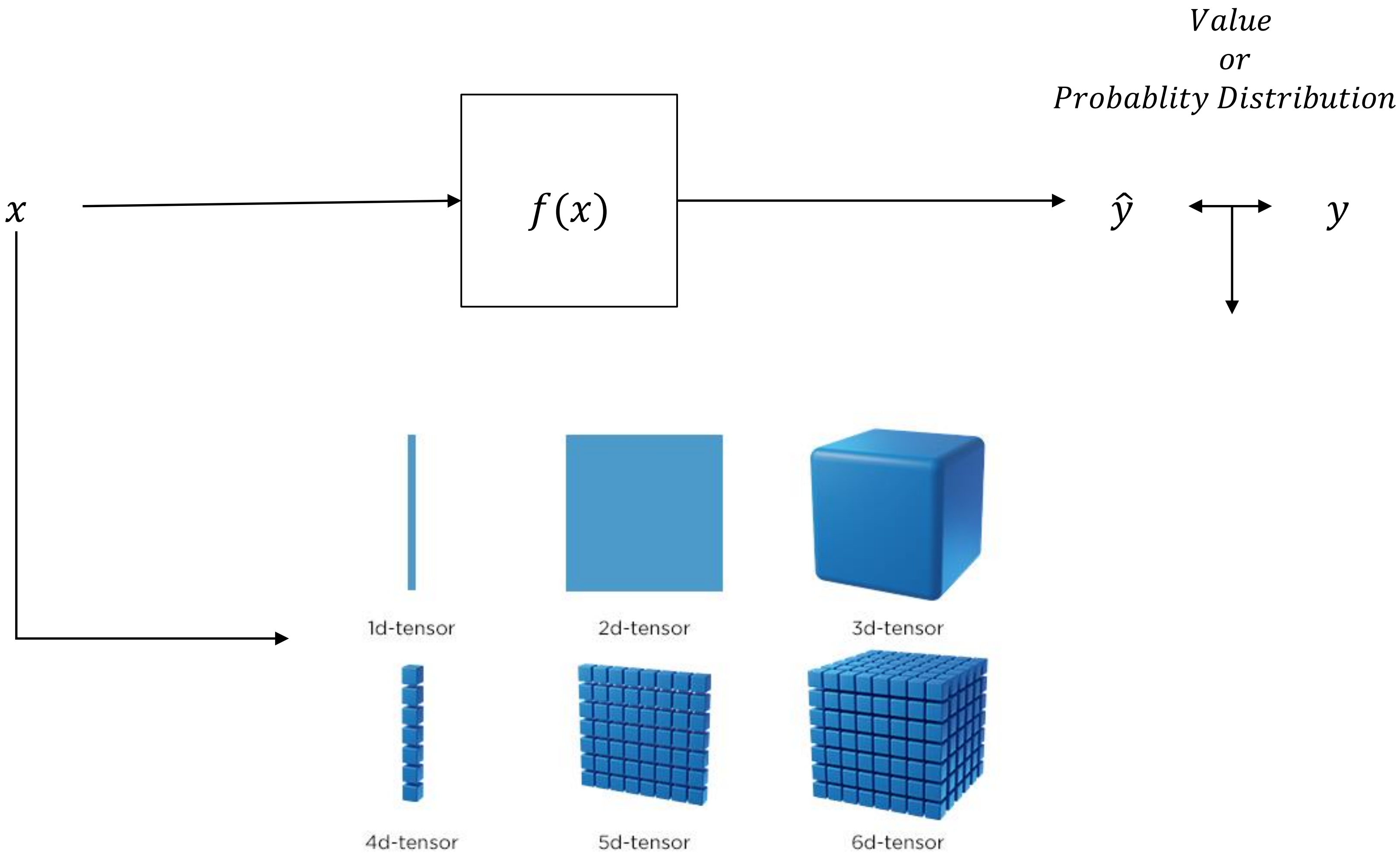


	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

사진 출처: https://www.insilicogen.com/blog/346 , https://www.lgcns.com/blog/cns-tech/ai-data/2343/

DL Basic

Input Data



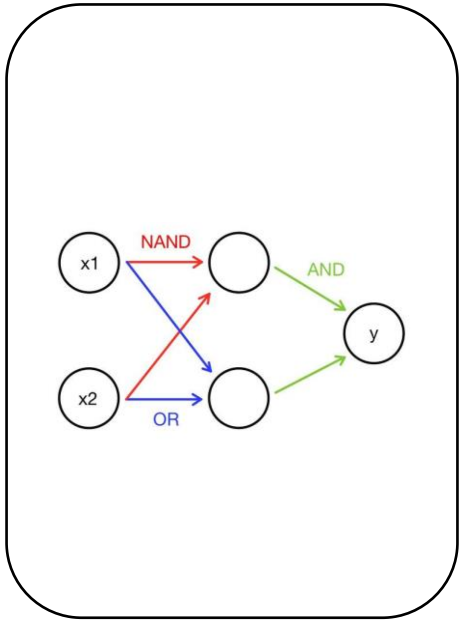
DL Basic

Model(Perceptron, ANN, DNN)

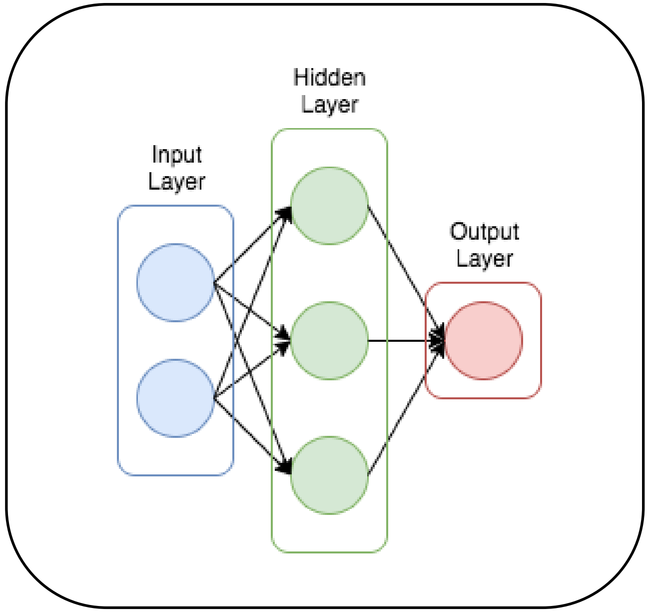
$$\boxed{f(x)}$$

Model

[Perceptron]



[ANN]



[DNN]

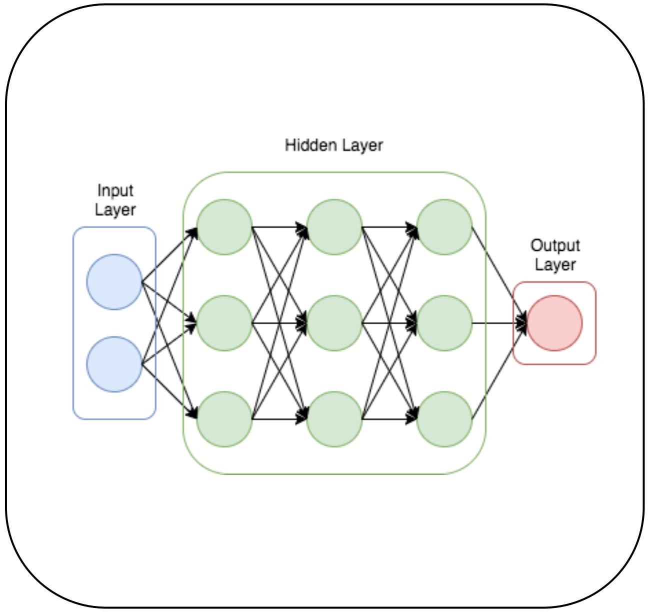


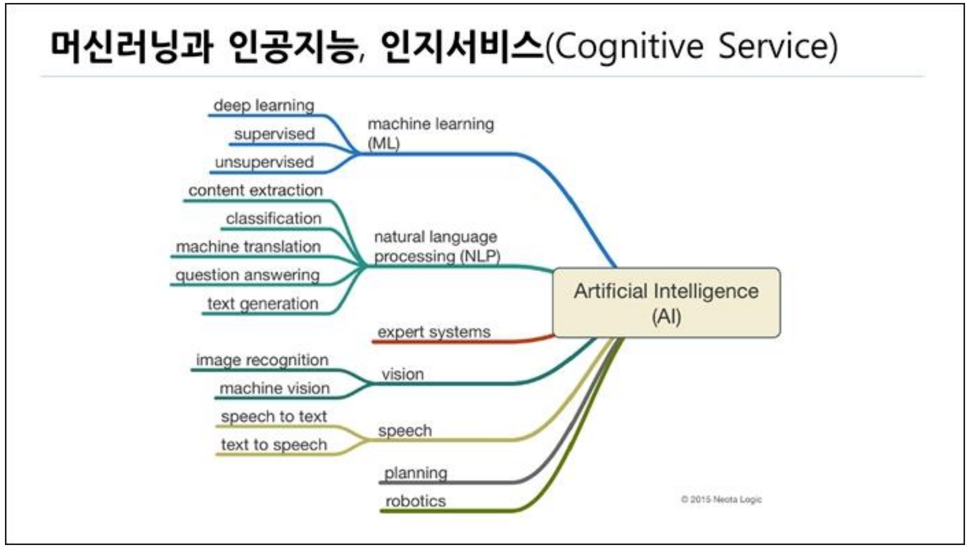
사진 출처: https://ebbnflow.tistory.com/119, https://yhyun225.tistory.com/20

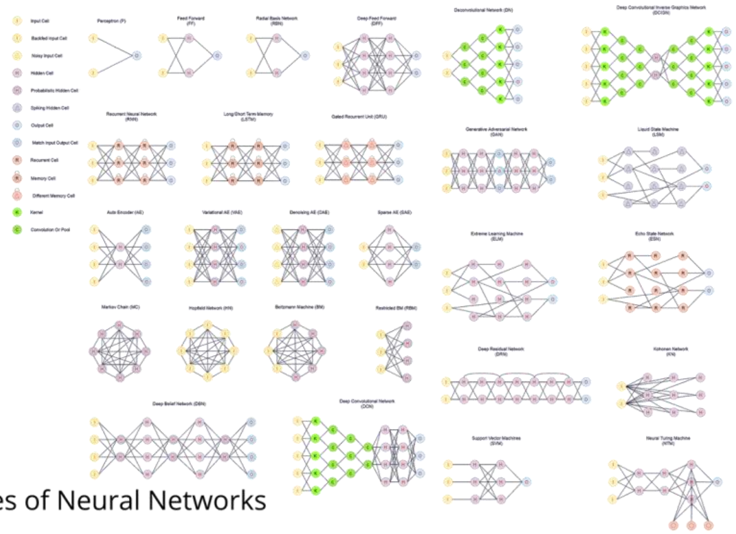
DL Basic

Model(Perceptron, ANN, DNN)

$$f(x)$$

Model

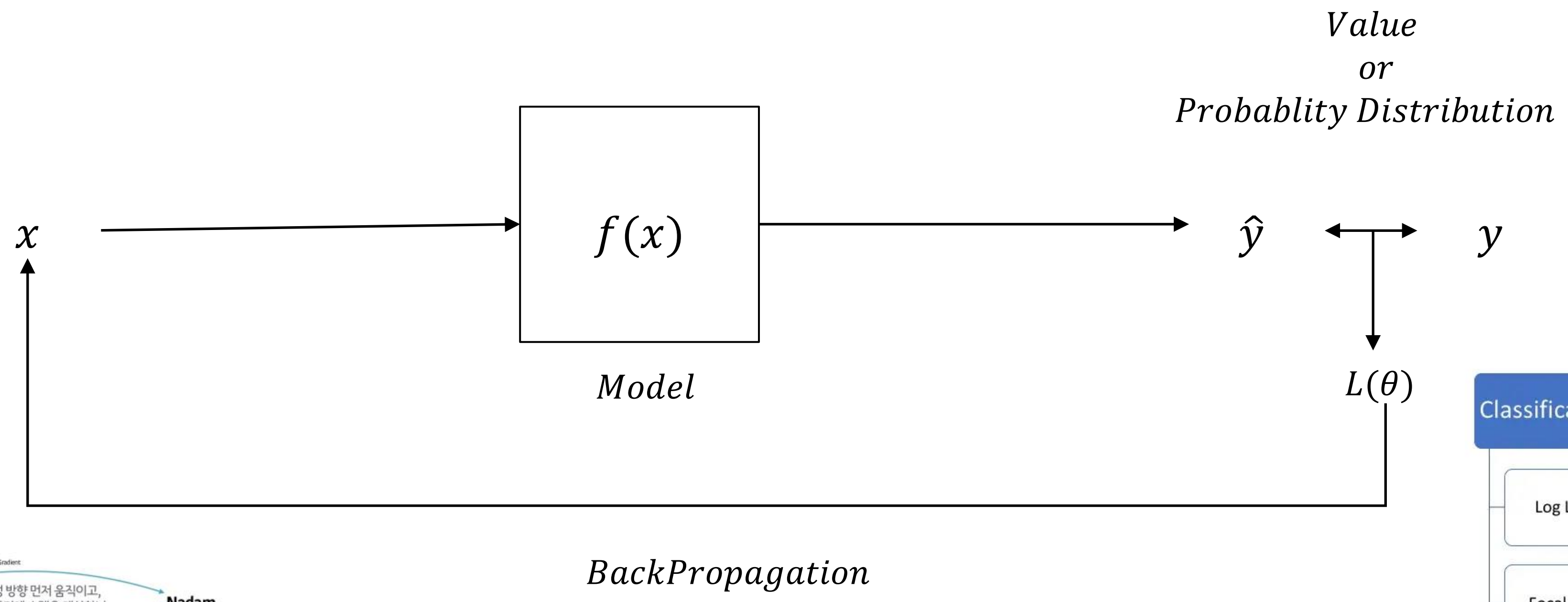




Main Types of Neural Networks

DL Basic

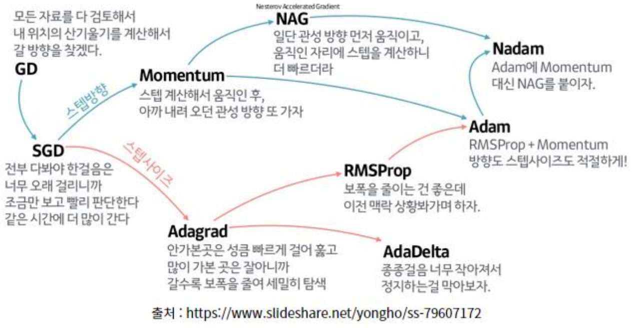
Our Objective with DL



$$\theta \leftarrow \theta - \eta \frac{\partial \mathcal{L}(\theta)}{\partial \theta}$$

or

$$\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}(\theta).$$



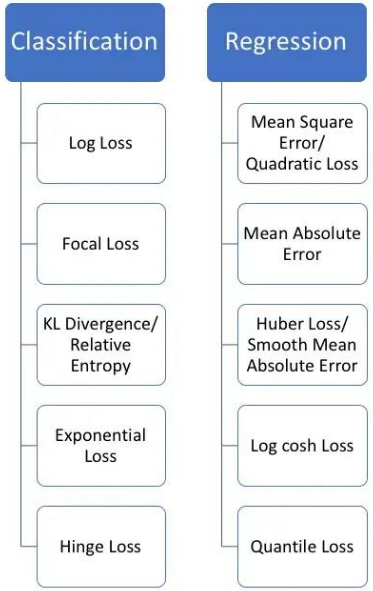
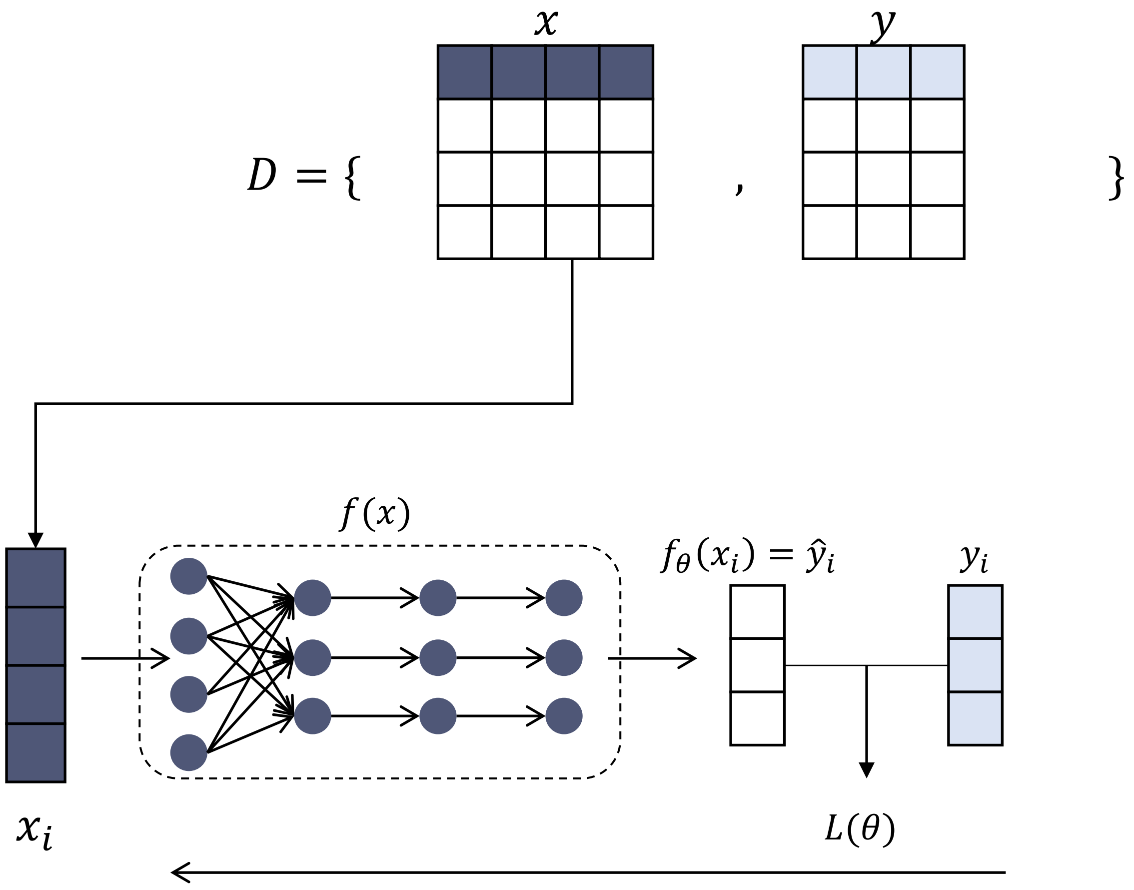


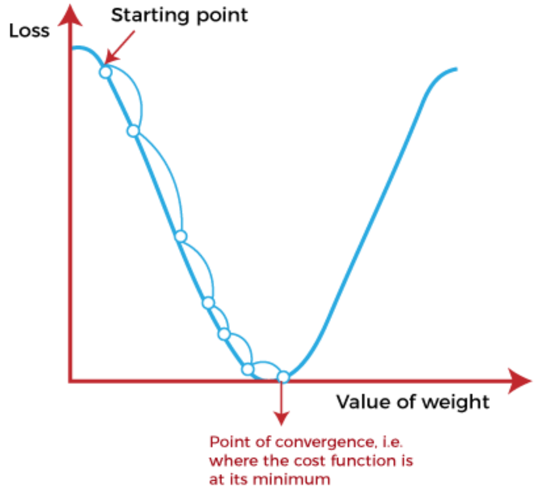
사진 출처: https://velog.io/@amzyoungchae/DL-%EC%86%90%EC%8B%A4%ED%95%A8%EC%88%98Loss-Function

DL Basic

Whole Process with DL



$$\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$$



where $\theta = \{W, b\}$

DL Basic

Model(Perceptron, ANN, DNN)

$$\boxed{f(x)}$$

$Model$

RNN → LSTM → Seq2Seq → Seq2Seq + self-Attention → Transformer

DL Basic

Whole Process with DL

통계학

1. 데이터의 패턴과 관계를 파악하기 위함
2. 불확실성을 다루고 예측을 만들기 위함
3. 실험 설계와 가설 검정을 통해 인사이트를 얻기 위함

미적분학

1. 다차원 데이터를 표현하고 조작하기
2. 기계학습 알고리즘, 특히 딥러닝에서의 연산을 이해하고 최적화하기 위함
3. 변환, 차원 축소, 클러스터링 등의 기법을 적용하기 위함

선형대수학

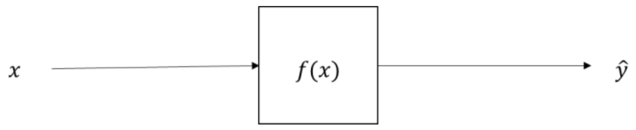
1. 최적화 문제를 해결하기 위함 (예: 경사하강법)
2. 연속 데이터의 변화율을 이해하고 모델링하기 위함
3. 확률 및 기계학습 이론에서 필요한 수학적 표현을 다루기 위함

RNN

1. What is RNN?
2. Sequential Data vs Time Series
3. Equation
4. t step vs 4 step

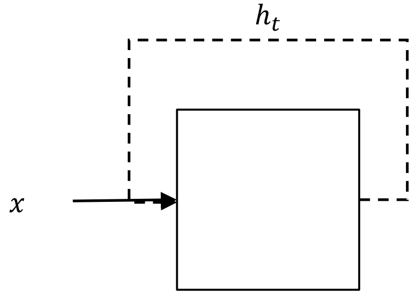
RNN
Recurrent Neural Networks

[Previous]



$$\hat{y} = f(x; \theta)$$

[RNN]



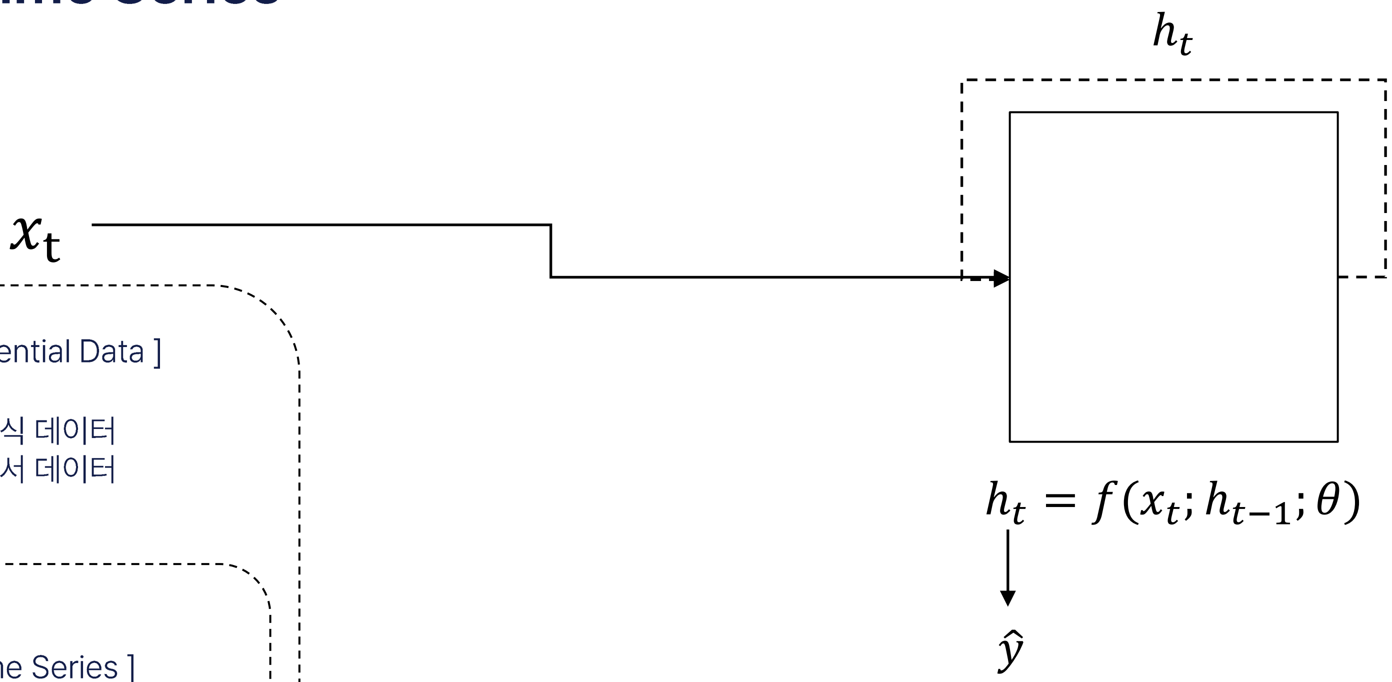
$$h_t = f(x_t; h_{t-1}; \theta)$$

$$\downarrow$$

$$\hat{y}$$

RNN

Sequential Data vs Time Series



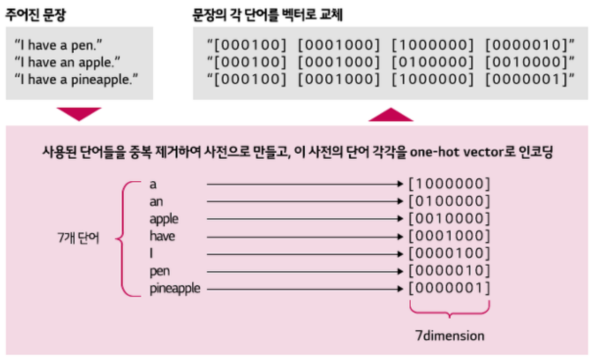
[Sequential Data]

- 주식 데이터
- 센서 데이터

[Time Series]

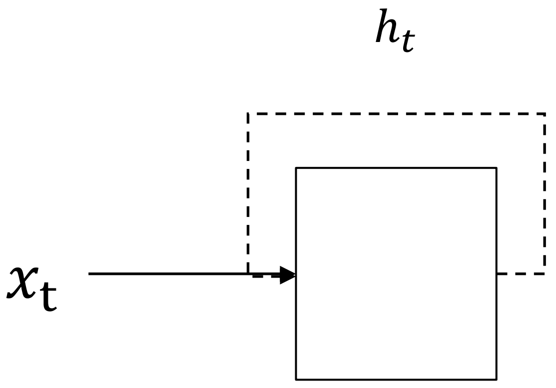
- 텍스트
- (샘플링 주기가 일정한) 영상/음성

데이터의 순서 정보가 매우 중요



RNN

Equation



$$h_t = f(x_t; h_{t-1}; \theta)$$

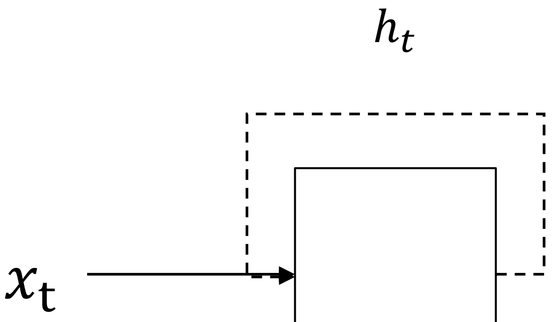
$$\begin{aligned}\hat{y}_t = h_t &= f(x_t, h_{t-1}; \theta) \\ &= \tanh(W_{ih}x_t + b_{ih} + W_{hh}h_{t-1} + b_{hh}) \\ \text{where } \theta &= \{W_{ih}, b_{ih}, W_{hh}, b_{hh}\}.\end{aligned}$$

- 현재 타임스텝 뿐만 아니라 이전 타임스텝의 출력을 입력을 받음
- RNN은 hidden state 자체가 output
- 오차역전파하기 이전까지는 같은 θ사용

RNN

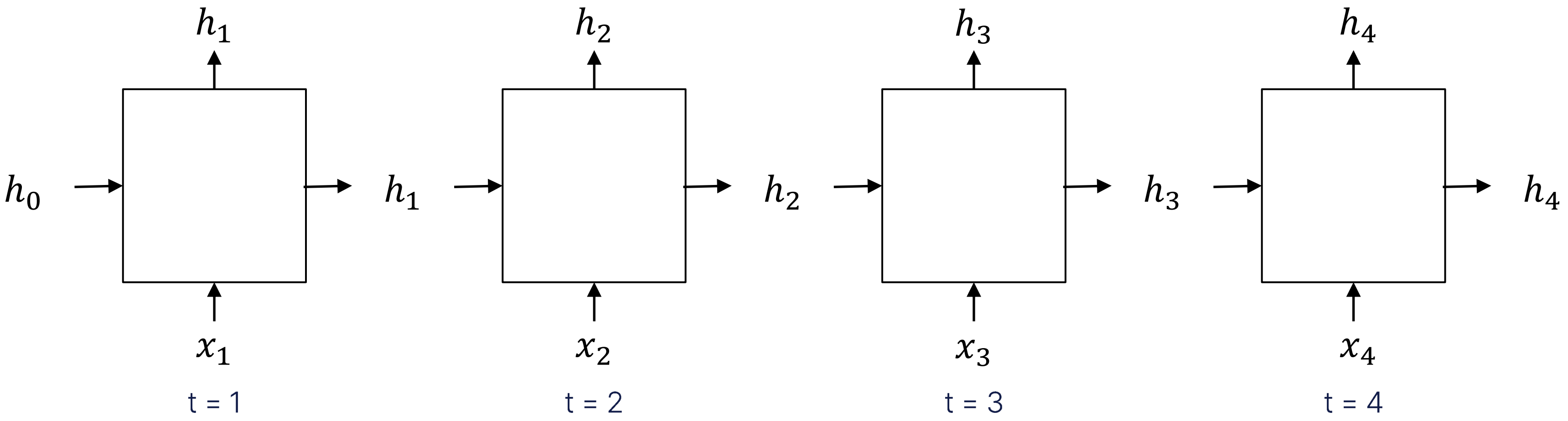
t step vs 4 step

[t step]



$$h_t = f(x_t; h_{t-1}; \theta)$$

[4 step]

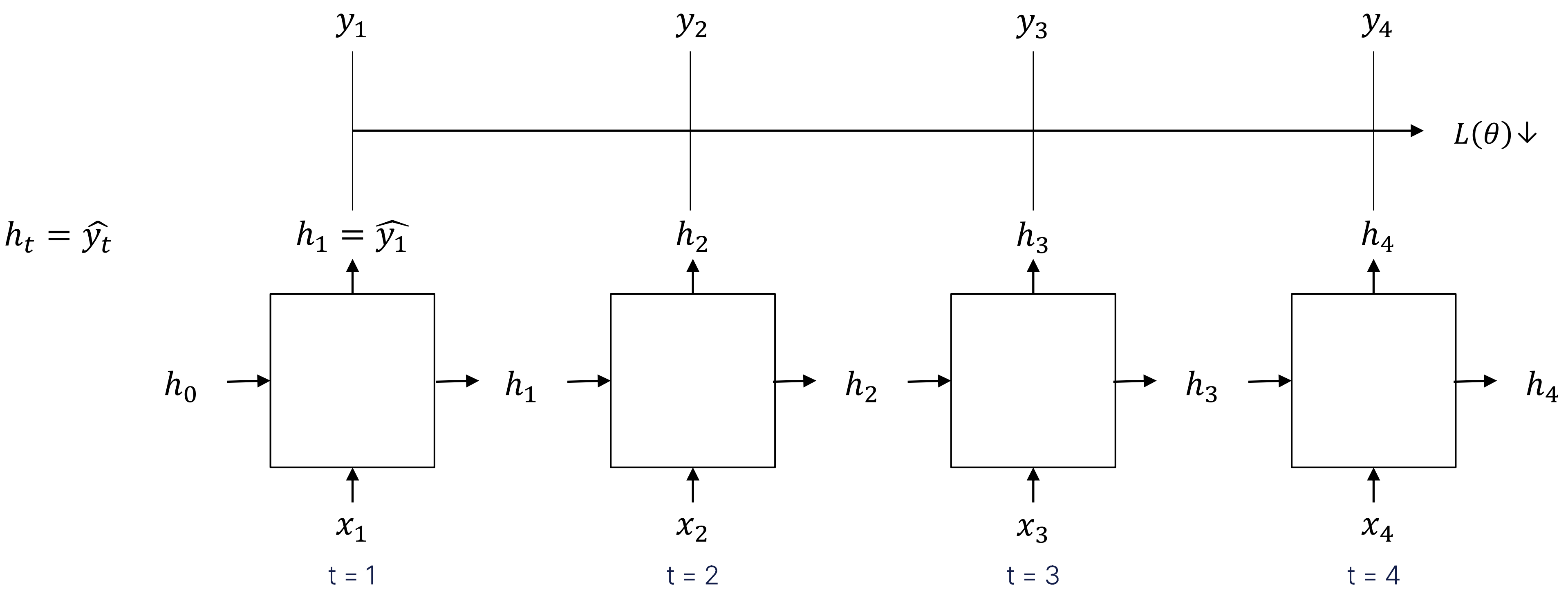


Single-layered RNN

1. About Single-layered RNN
2. Input Tensor(3D Tensor)
3. Hidden State Tensor

RNN - Single-layered RNN

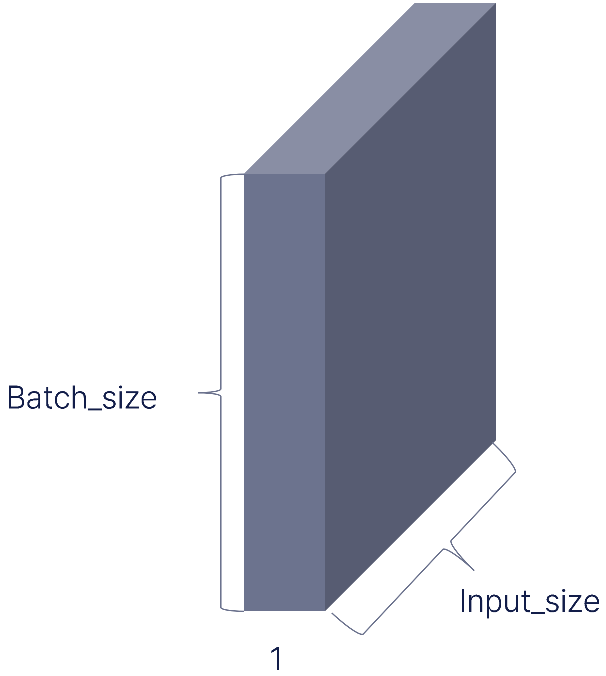
Single-layered RNN



x들이 t=4시점에서 한 번에 다 들어가고, ŷ이 한번에 튀어나옴

RNN - Single-layered RNN

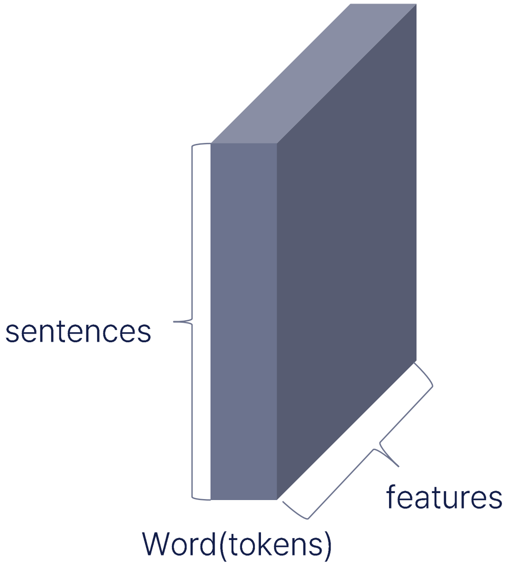
Input Tensor(3D Tensor)



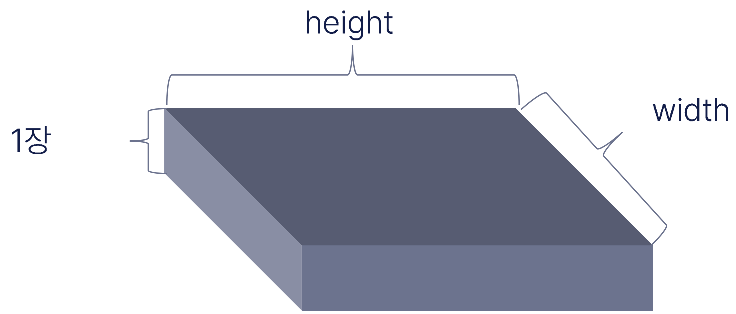
$$x_t \in \mathbb{R}^{\text{batch\_size},1,\text{input\_size}}$$

$$|x_t| = (\text{batch\_size}, 1, \text{input\_size})$$

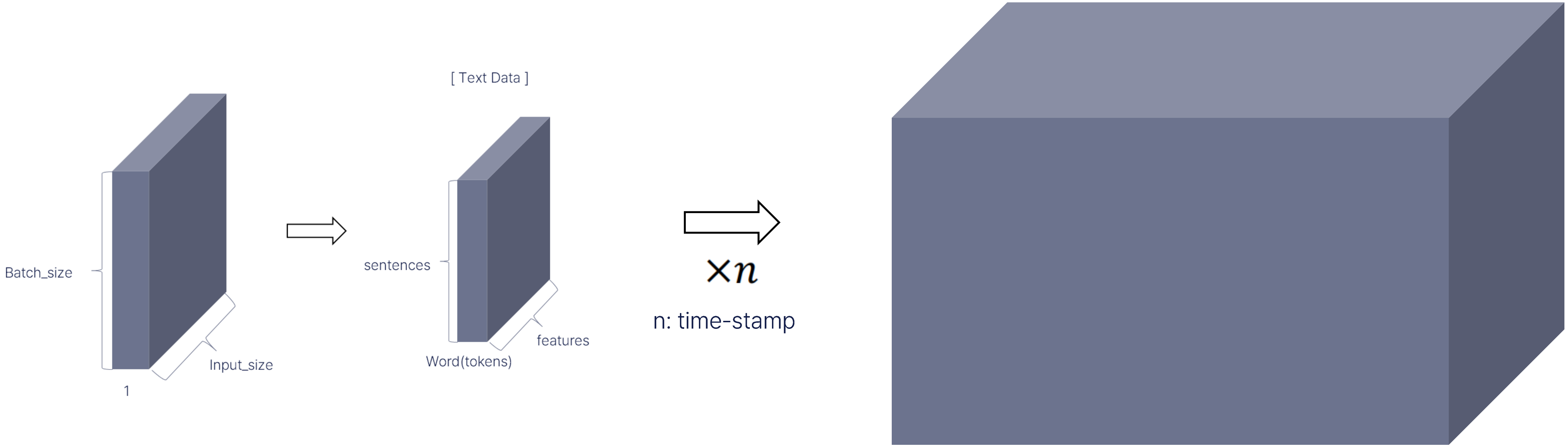
[Text Data]



[image Data]



RNN - Single-layered RNN
Input Tensor(3D Tensor)



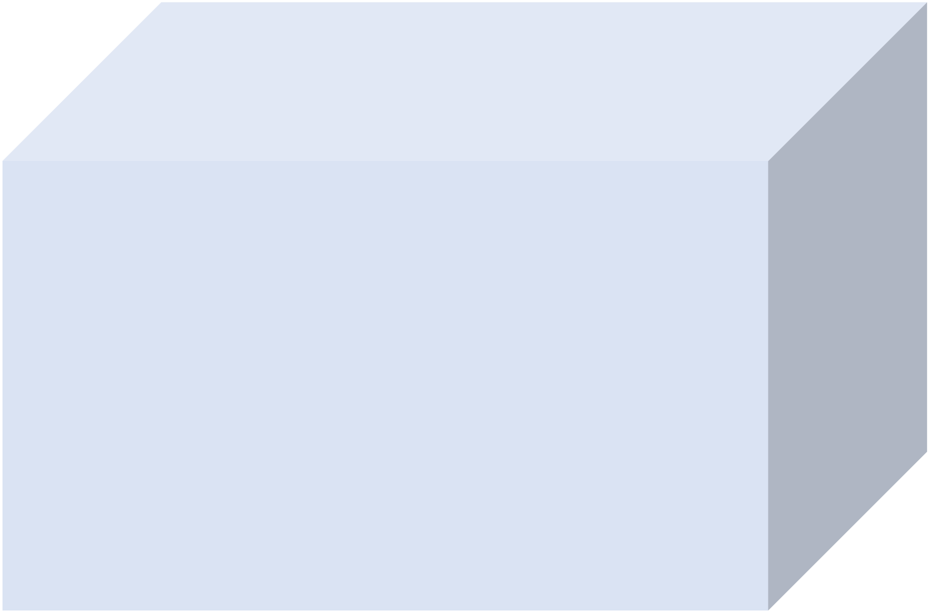
$$|X| = (\text{batch\_size}, n, \text{input\_size})$$
$$\text{where } X = \{x_1, x_2, \cdots, x_n\}$$

RNN - Single-layered RNN

Hidden State Tensor

$\times n$

n: time-stamp



$$|h_t| = (\text{batch\_size}, \text{hidden\_size})$$

$$|h_{1:n}| = (\text{batch\_size}, n, \text{hidden\_size})$$

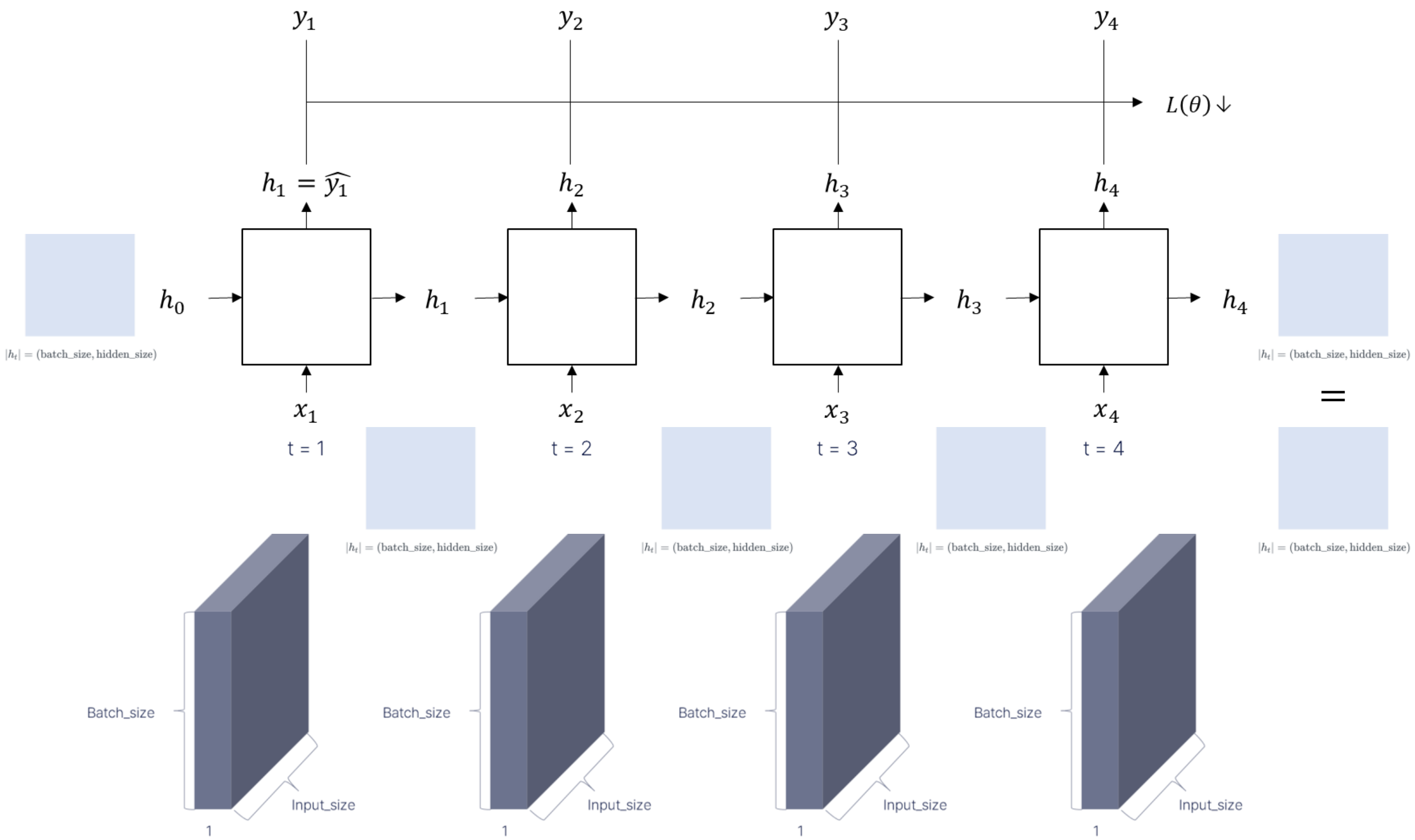
$$\text{where } h_{1:n} = [h_1; h_2; \cdots; h_n].$$

$$\Downarrow$$

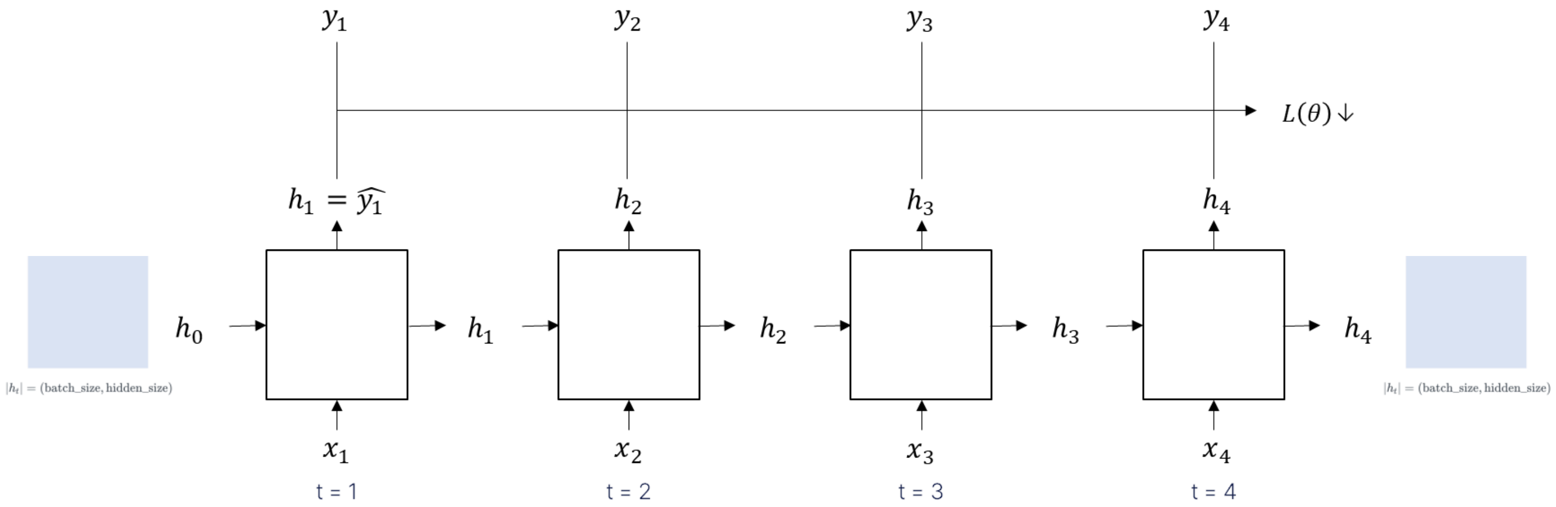
$$\hat{y}$$

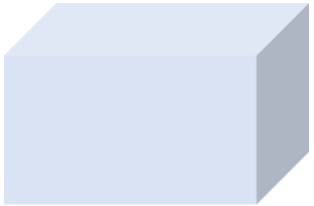
RNN - Single-layered RNN

Input & Hidden Tensor



RNN
Input & Hidden Tensor



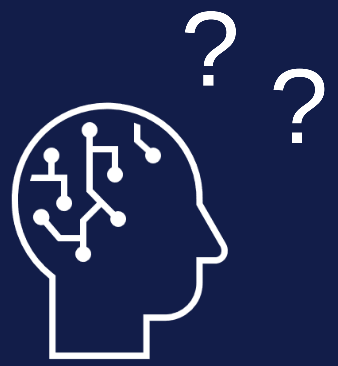


$$|h_{1:n}| = (\text{batch\_size}, n, \text{hidden\_size})$$
$$\text{where } h_{1:n} = [h_1; h_2; \cdots; h_n].$$

$$|X| = (\text{batch\_size}, n, \text{input\_size})$$
$$\text{where } X = \{x_1, x_2, \cdots, x_n\}$$

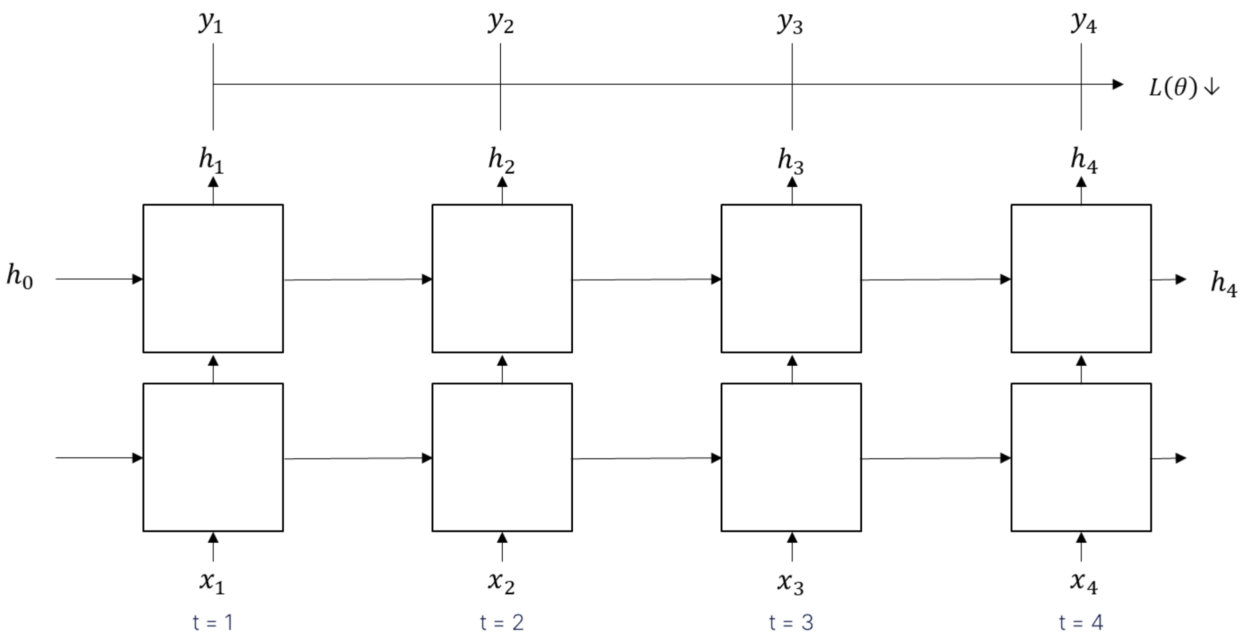
Multi-layered RNN

1. Input Tensor
2. Hidden State Tensor
3. Output Tensor



RNN - Multi-layered RNN

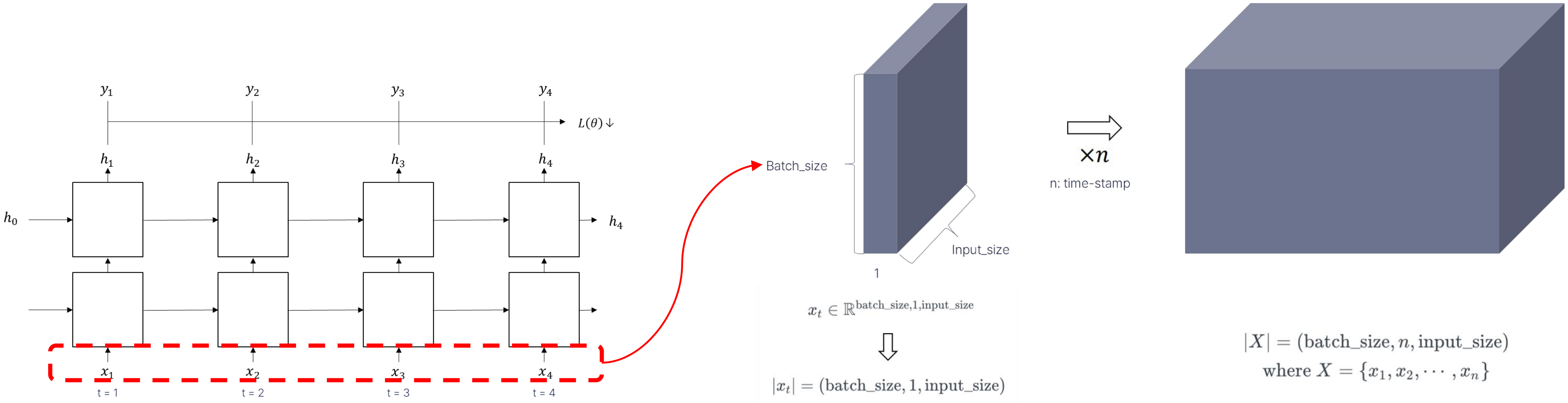
Multi-layered RNN



- 모든 층의 h값을 합쳐서, hidden state라고 부름
- PyTorch에서는 number of layers 값을 전달해주면, 알아서 층을 만들어줌
- 마지막 층의 hidden state들이 $\hat{y}$이 된다

RNN - Multi-layered RNN

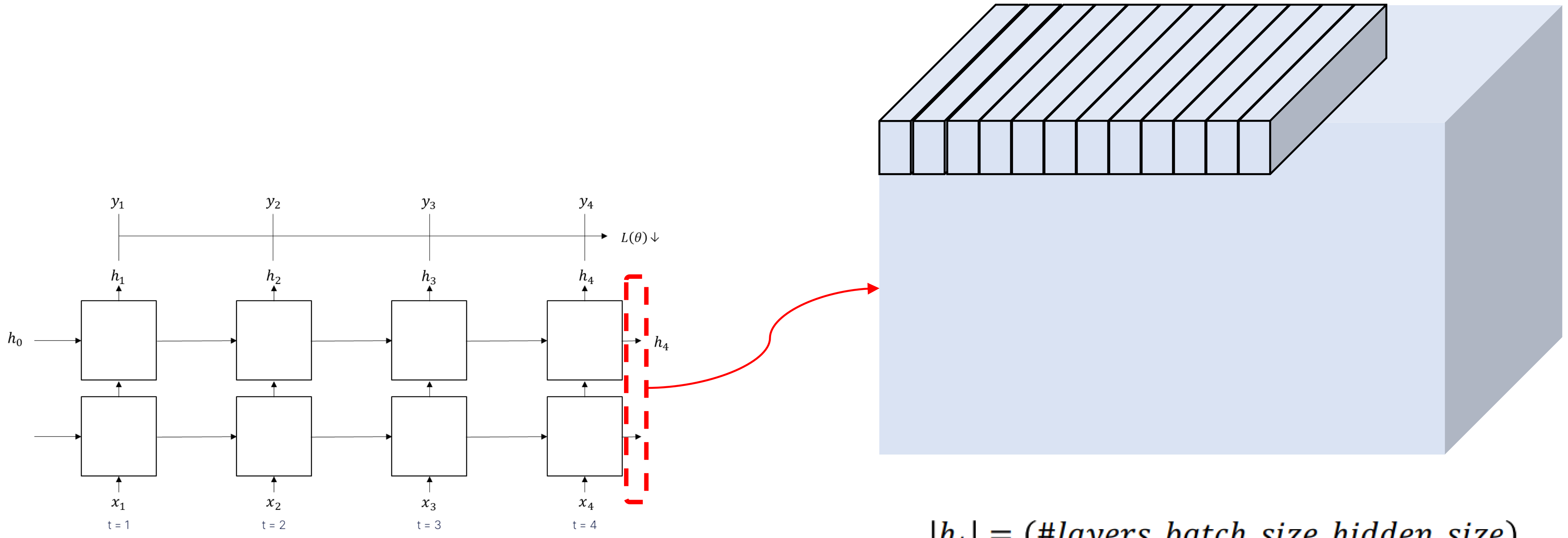
Input Tensor(Multi-layered RNN)



Single-layered RNN과 Input은 같다.

RNN - Multi-layered RNN

Hidden State Tensor(Multi-layered RNN)

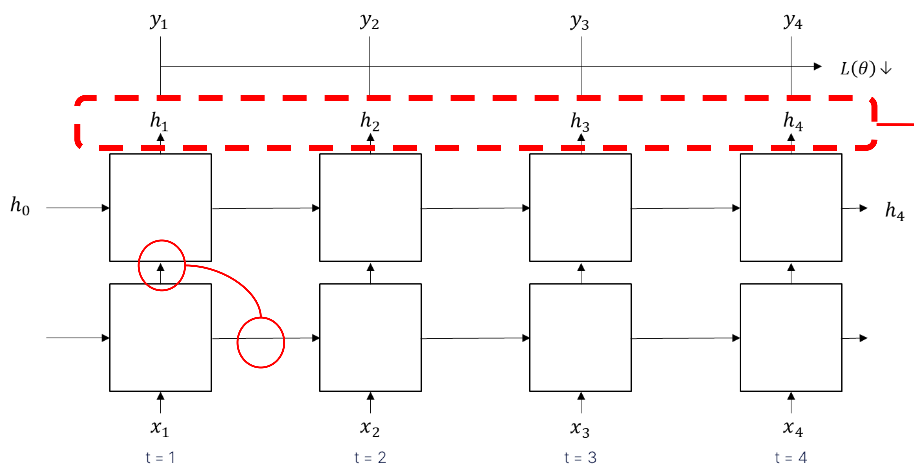


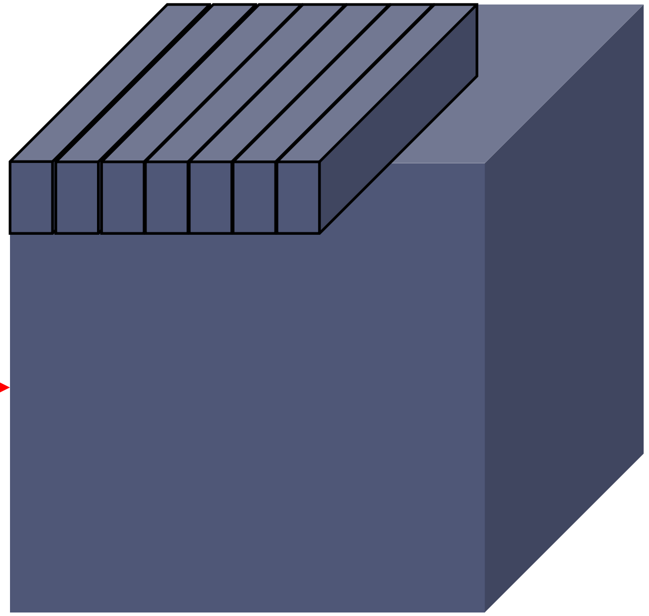
$$|h_t| = (\#layers, batch\_size, hidden\_size)$$

한 time-stamp에 대한 hidden state

RNN - Multi-layered RNN

Output Tensor(Multi-layered RNN)





$$|h_{1:n}| = (\text{batch\_size}, n, \text{hidden\_size})$$
$$\text{where } h_{1:n} = [h_1; h_2; \cdots; h_n].$$
$$\Downarrow$$
$$\widehat{y_{1:n}}$$

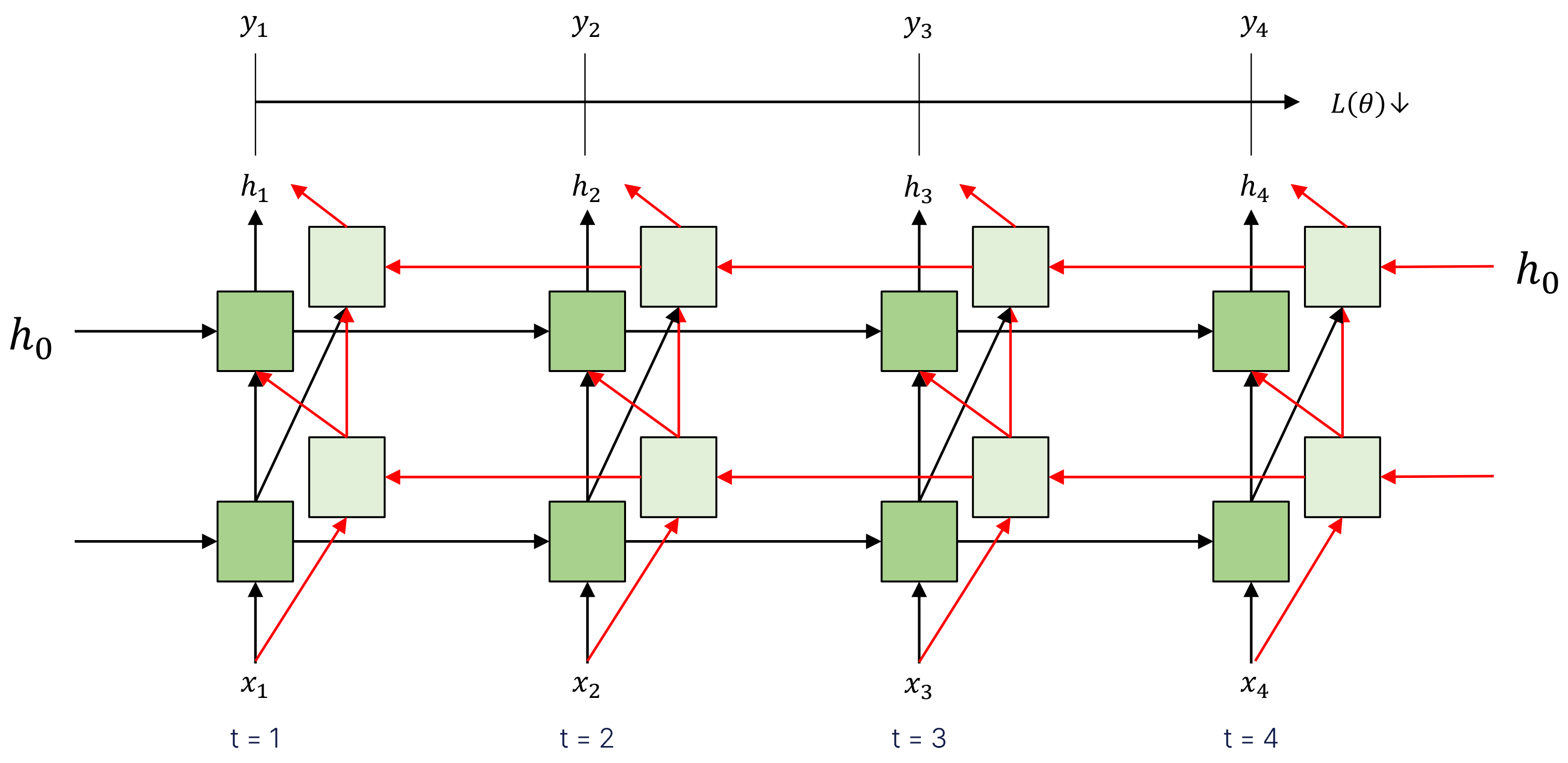
1층의 출력은 2층의 입력, 2층의 출력은 다시 3층의 입력, 3층의 출력은 ŷ

Bidirectional Multi-layered RNN

HAI

RNN - Bidirectional Multi-layered RNN

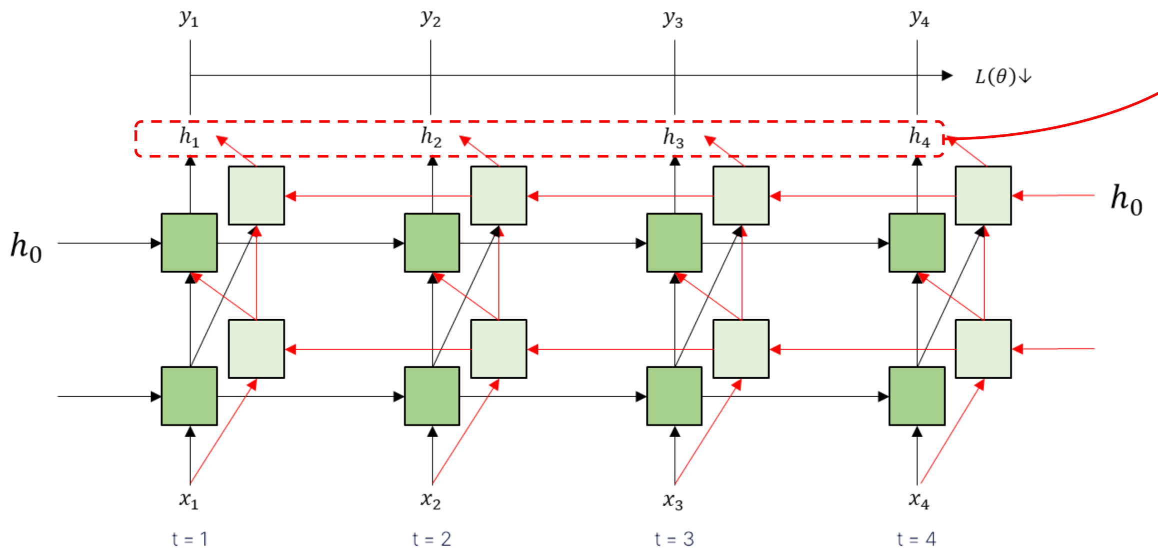
Bidirectional Multi-layered RNN

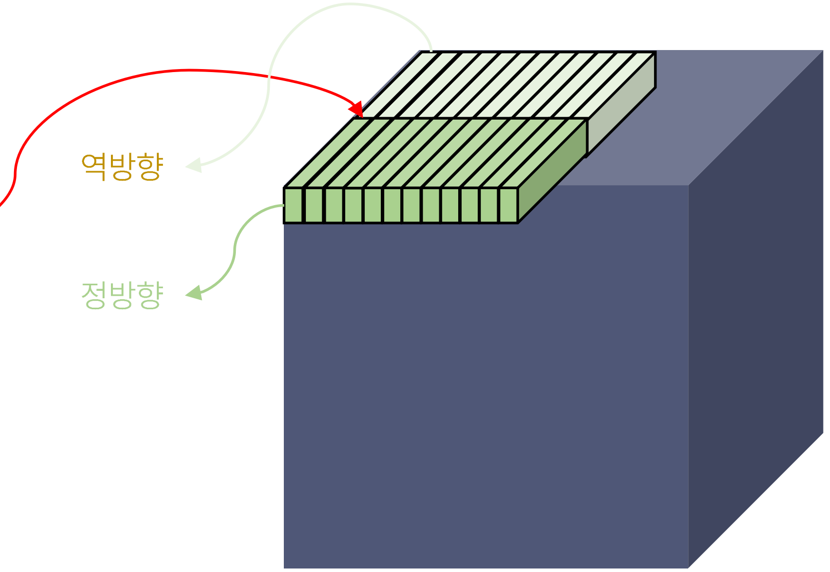


- 이전까지 그림이 층별로 색깔을 구분해놨다면
- 방향별로 색깔을 구분
  - 빨강: 역방향
  - 초록: 정방향
- 통째로 입력과 출력이 나오는 케이스에서 자주 접할 예정

RNN - Bidirectional Multi-layered RNN

Output Tensor



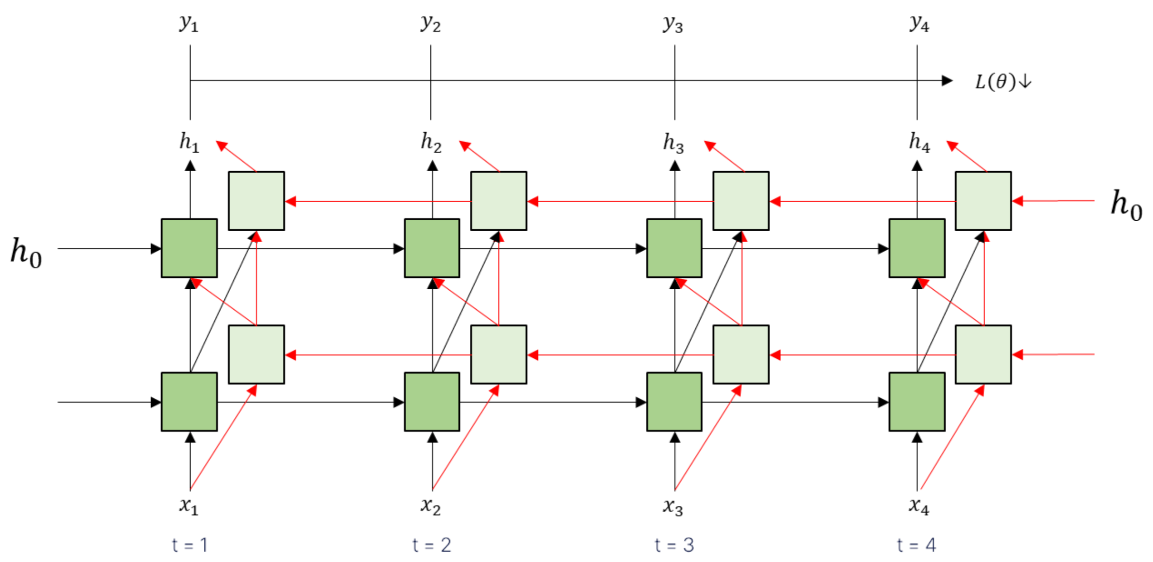


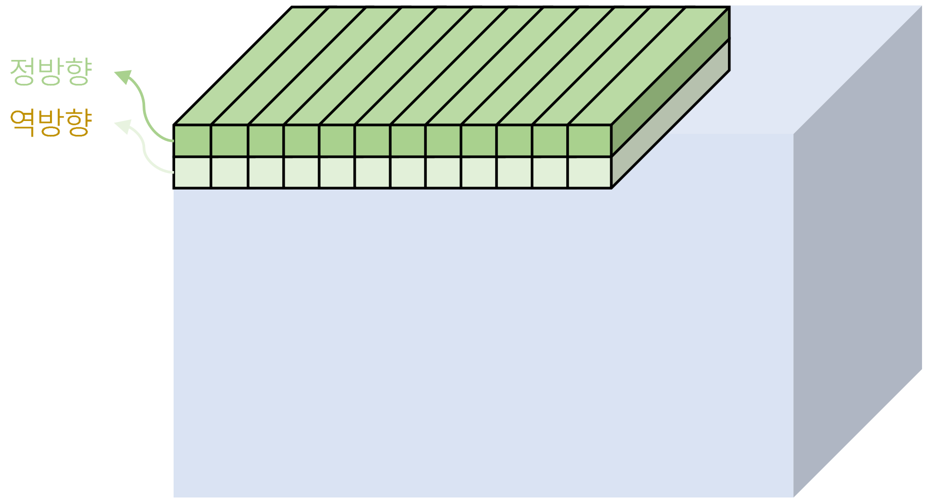
$$|h_{1:n}| = (batch\_size, n, hidden\_size \times \#directions)$$

마지막 layer의 time step별 정방향 hidden state들과 역방향 hidden state들이 모두 모인다

RNN - Bidirectional Multi-layered RNN

Hidden State Tensor





$$|h_t| = (\#\text{direction} \times \#\text{layers}, \text{batch\_size}, \text{hidden\_size})$$

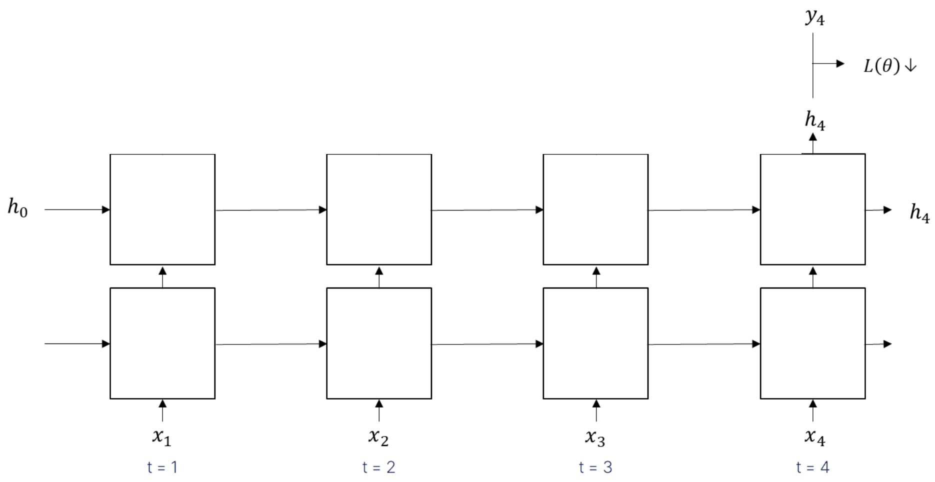
•양방향 다층 RNN에서는 hidden state가 그렇게 중요하지 않음
•hidden state를 사용하는 경우는 대게 이전 time step의 결과물을 다음 step에 사용하기 위해서이다
•하지만 전체 타임스텝이 들어오고 나가기 때문에, 다음 입력을 받기 위한 작업들을 대체로 하지 않는다

“Scratch” vs “torch.nn.RNN”

1. Scratch code
2. torch.nn.RNN

RNN - Scratch Code

Multi-layered RNN



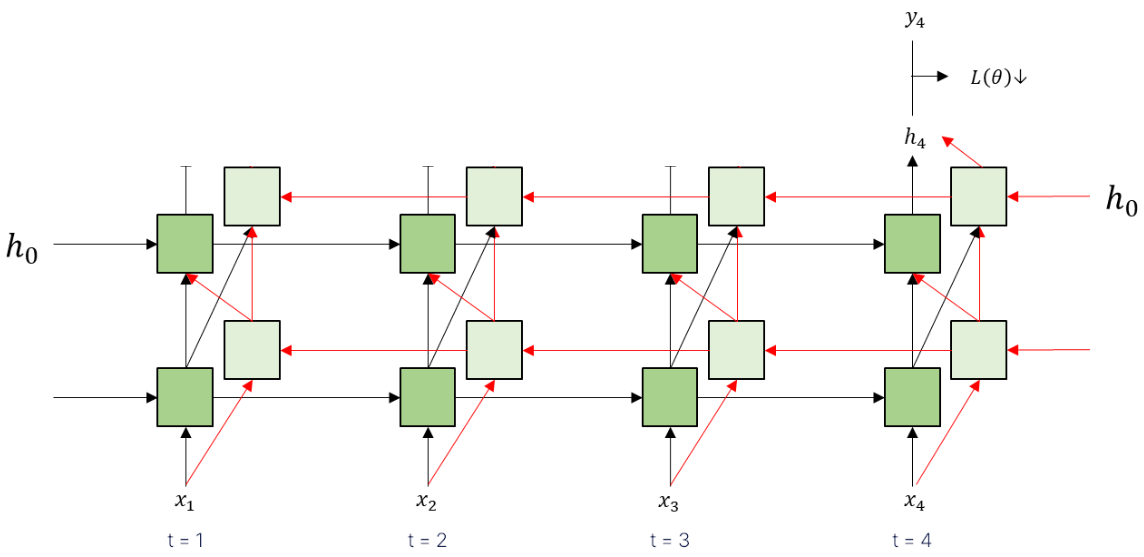

```python
class RNN(nn.Module):
    """
    기본 RNN 블록입니다. 이는 RNN의 단일 레이어를 나타냅니다.
    """
    def __init__(self, input_size: int, hidden_size: int, output_size: int) -> None:
        """
        input_size: 입력 벡터의 특징 수
        hidden_size: 은닉 뉴런의 수
        output_size: 출력 벡터의 특징 수
        """
        super().__init__()
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.output_size = output_size
        self.batch_size = batch_size
        self.i2h = nn.Linear(input_size, hidden_size, bias=False)
        self.h2h = nn.Linear(hidden_size, hidden_size)
        self.h2o = nn.Linear(hidden_size, output_size)

    def forward(self, x, hidden_state) -> tuple[torch.Tensor, torch.Tensor]:
        """
        계산된 출력 및 tanh(i2h + h2h)를 반환합니다.
        입력값
        ------
        x: 입력 벡터
        hidden_state: 이전 은닉 상태
        출력값
        -------
        out: 활성화 함수 없이 계산된 선형 출력 (파이토치의 작동 방식 때문)
        hidden_state: 새로운 은닉 상태 행렬
        """
        x = self.i2h(x)
        hidden_state = self.h2h(hidden_state)
        hidden_state = torch.tanh(x + hidden_state)
        out = self.h2o(hidden_state)
        return out, hidden_state

    def init_zero_hidden(self, batch_size=1) -> torch.Tensor:
        """
                도우미 함수입니다.
        지정된 배치 크기로 은닉 상태를 반환합니다. 기본값은 1입니다.
        """
        return torch.zeros(batch_size, self.hidden_size, requires_grad=False)
```

코드 활용: https://medium.com/@VersuS_/coding-a-recurrent-neural-network-rnn-from-scratch-using-pytorch-a6c9fc8ed4a7

RNN - Scratch Code
Bidirectional Multi-layered RNN




```python
class BidirectionalRNN(nn.Module):
    def __init__(self, input_size: int, hidden_size: int, output_size: int) -> None:
        super().__init__()

        self.input_size = input_size
        self.hidden_size = hidden_size
        self.output_size = output_size

        # 순방향 RNN 블록
        self.i2h_forward = nn.Linear(input_size, hidden_size, bias=False)
        self.h2h_forward = nn.Linear(hidden_size, hidden_size)
        self.h2o_forward = nn.Linear(hidden_size, output_size)

        # 역방향 RNN 블록
        self.i2h_backward = nn.Linear(input_size, hidden_size, bias=False)
        self.h2h_backward = nn.Linear(hidden_size, hidden_size)
        self.h2o_backward = nn.Linear(hidden_size, output_size)

    def forward(self, x, hidden_state_forward, hidden_state_backward) -> tuple[torch.Tensor, torch.Tensor]:
        # 순방향 처리
        x_forward = self.i2h_forward(x)
        hidden_state_forward = self.h2h_forward(hidden_state_forward)
        hidden_state_forward = torch.tanh(x_forward + hidden_state_forward)
        out_forward = self.h2o_forward(hidden_state_forward)

        # 역방향 처리
        x_backward = self.i2h_backward(x)
        hidden_state_backward = self.h2h_backward(hidden_state_backward)
        hidden_state_backward = torch.tanh(x_backward + hidden_state_backward)
        out_backward = self.h2o_backward(hidden_state_backward)

        # 순방향 및 역방향 결과 결합
        out = torch.cat((out_forward, out_backward), dim=-1)

        return out, (hidden_state_forward, hidden_state_backward)

    def init_zero_hidden(self, batch_size=1) -> tuple[torch.Tensor, torch.Tensor]:
        return (
            torch.zeros(batch_size, self.hidden_size, requires_grad=False),  # 순방향 은닉 상태 초기화
            torch.zeros(batch_size, self.hidden_size, reqires_grad=False)   # 역방향 은닉 상태 초기화
        )
```

기존 코드를 응용하여 Bidirectional으로 변환

RNN - torch.nn.RNN

Documentation

RNN

CLASS torch.nn.RNN(*args, **kwargs) [SOURCE]

Applies a multi-layer Elman RNN with $\tanh$ or ReLU non-linearity to an input sequence.

For each element in the input sequence, each layer computes the following function:

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

where h_t is the hidden state at time *t*, x_t is the input at time *t*, and $h_{(t-1)}$ is the hidden state of the previous layer at time *t-1* or the initial hidden state at time *0*. If `nonlinearity` is `'relu'`, then ReLU is used instead of $\tanh$.

Parameters:

- **input_size** – The number of expected features in the input *x*
- **hidden_size** – The number of features in the hidden state *h*
- **num_layers** – Number of recurrent layers. E.g., setting `num_layers=2` would mean stacking two RNNs together to form a *stacked RNN*, with the second RNN taking in outputs of the first RNN and computing the final results. Default: 1
- **nonlinearity** – The non-linearity to use. Can be either `'tanh'` or `'relu'`. Default: `'tanh'`
- **bias** – If `False`, then the layer does not use bias weights *b_ih* and *b_hh*. Default: `True`
- **batch_first** – If `True`, then the input and output tensors are provided as (*batch, seq, feature*) instead of (*seq, batch, feature*). Note that this does not apply to hidden or cell states. See the Inputs/Outputs sections below for details. Default: `False`
- **dropout** – If non-zero, introduces a *Dropout* layer on the outputs of each RNN layer except the last layer, with dropout probability equal to `dropout`. Default: 0
- **bidirectional** – If `True`, becomes a bidirectional RNN. Default: `False`

Inputs: input, h_0

- **Input**: tensor of shape (L, H_{in}) for unbatched input, (L, N, H_{in}) when `batch_first=False` or (N, L, H_{in}) when `batch_first=True` containing the features of the input sequence. The input can also be a packed variable length sequence. See `torch.nn.utils.rnn.pack_padded_sequence()` or `torch.nn.utils.rnn.pack_sequence()` for details.
- **h_0**: tensor of shape $(D * \text{num\_layers}, H_{out})$ for unbatched input or $(D * \text{num\_layers}, N, H_{out})$ containing the initial hidden state for the input sequence batch. Defaults to zeros if not provided.

where:

$$\begin{aligned} N &= \text{batch size} \\ L &= \text{sequence length} \\ D &= 2 \text{ if bidirectional=True otherwise } 1 \\ H_{in} &= \text{input\_size} \\ H_{out} &= \text{hidden\_size} \end{aligned}$$

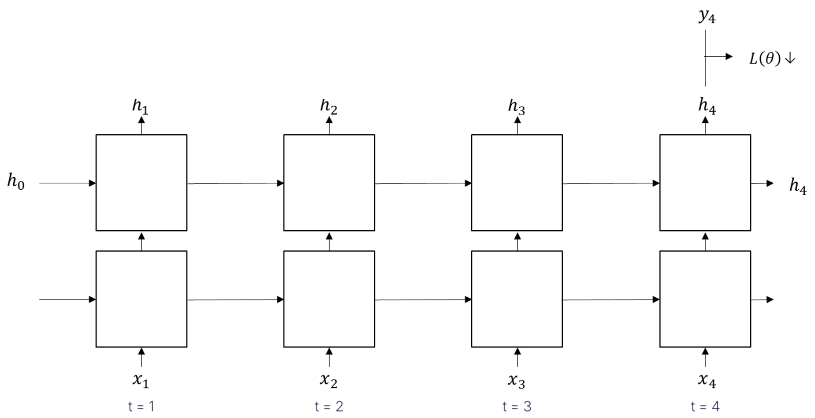
Outputs: output, h_n

- **output**: tensor of shape $(L, D * H_{out})$ for unbatched input, $(L, N, D * H_{out})$ when `batch_first=False` or $(N, L, D * H_{out})$ when `batch_first=True` containing the output features (*h_t*) from the last layer of the RNN, for each *t*. If a `torch.nn.utils.rnn.PackedSequence` has been given as the input, the output will also be a packed sequence.
- **h_n**: tensor of shape $(D * \text{num\_layers}, H_{out})$ for unbatched input or $(D * \text{num\_layers}, N, H_{out})$ containing the final hidden state for each element in the batch.

사진 출처: https://pytorch.org/docs/stable/generated/torch.nn.RNN.html

RNN - torch.nn.RNN
Using PyTorch

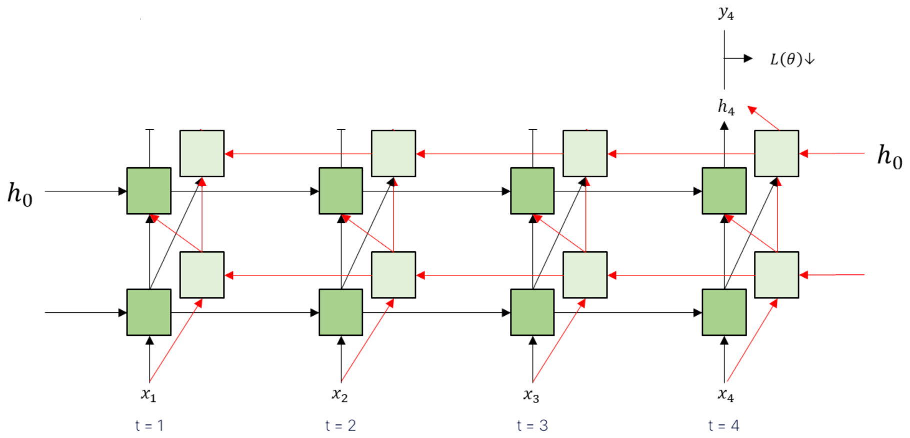
Multi-layered RNN



input_size=4, hidden_size=4, num_layers=2

```
rnn = nn.RNN(input_size=4, hidden_size=4, num_layers=2)
input = torch.randn(5, 3, 4)
h0 = torch.randn(2, 3, 4)
output, hn = rnn(input, h0)
```

Bidirectional Multi-layered RNN




```
rnn = nn.RNN(input_size=4, hidden_size=4, num_layers=2, bidirectional=True)
input = torch.randn(5, 3, 4)
h0 = torch.randn(2 * 2, 3, 4)
output, hn = rnn(input, h0)
```

사진 출처: https://pytorch.org/docs/stable/generated/torch.nn.RNN.html

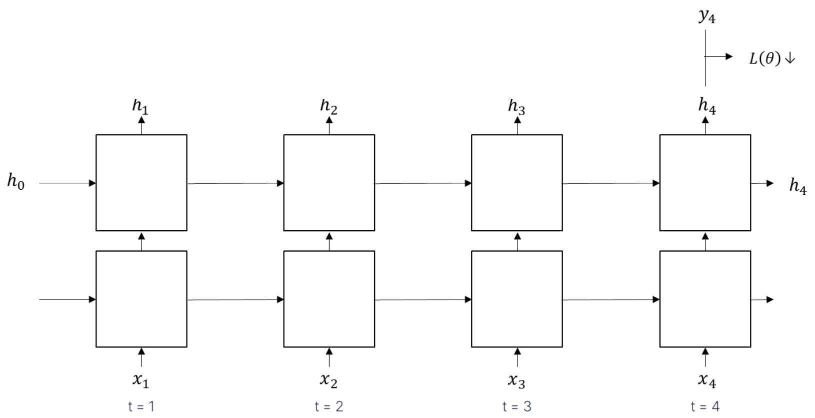
Application of RNNs

1. Application
2. Two Approaches

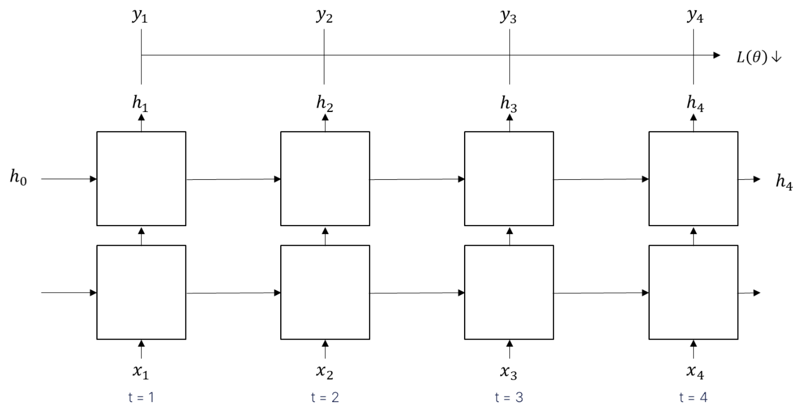
RNN - Application of RNNs

Application of RNNs

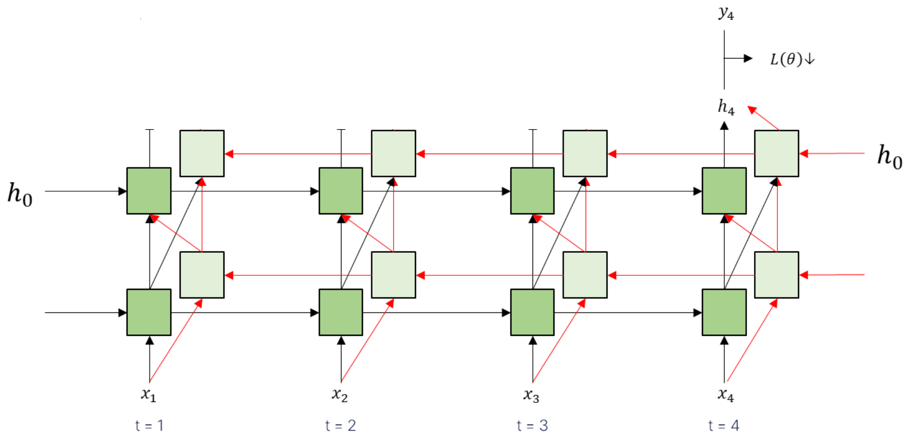
Multi-layered RNN



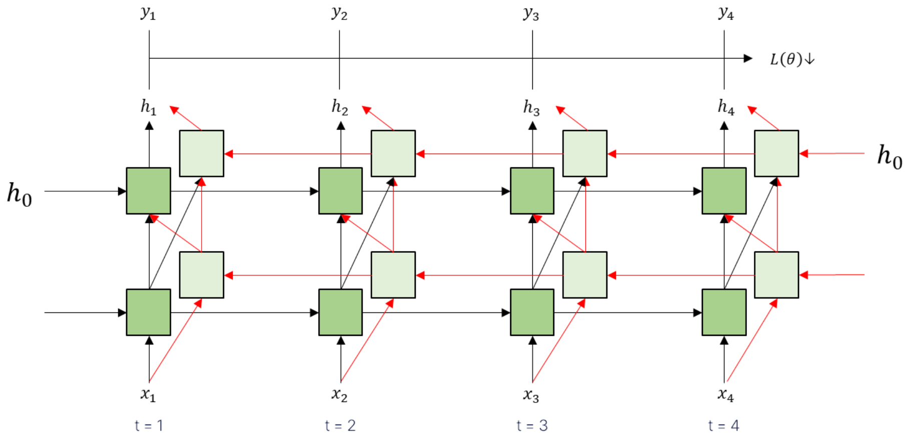
!=



Bidirectional Multi-layered RNN



!=



RNN - Application of RNNs

Application of RNNs

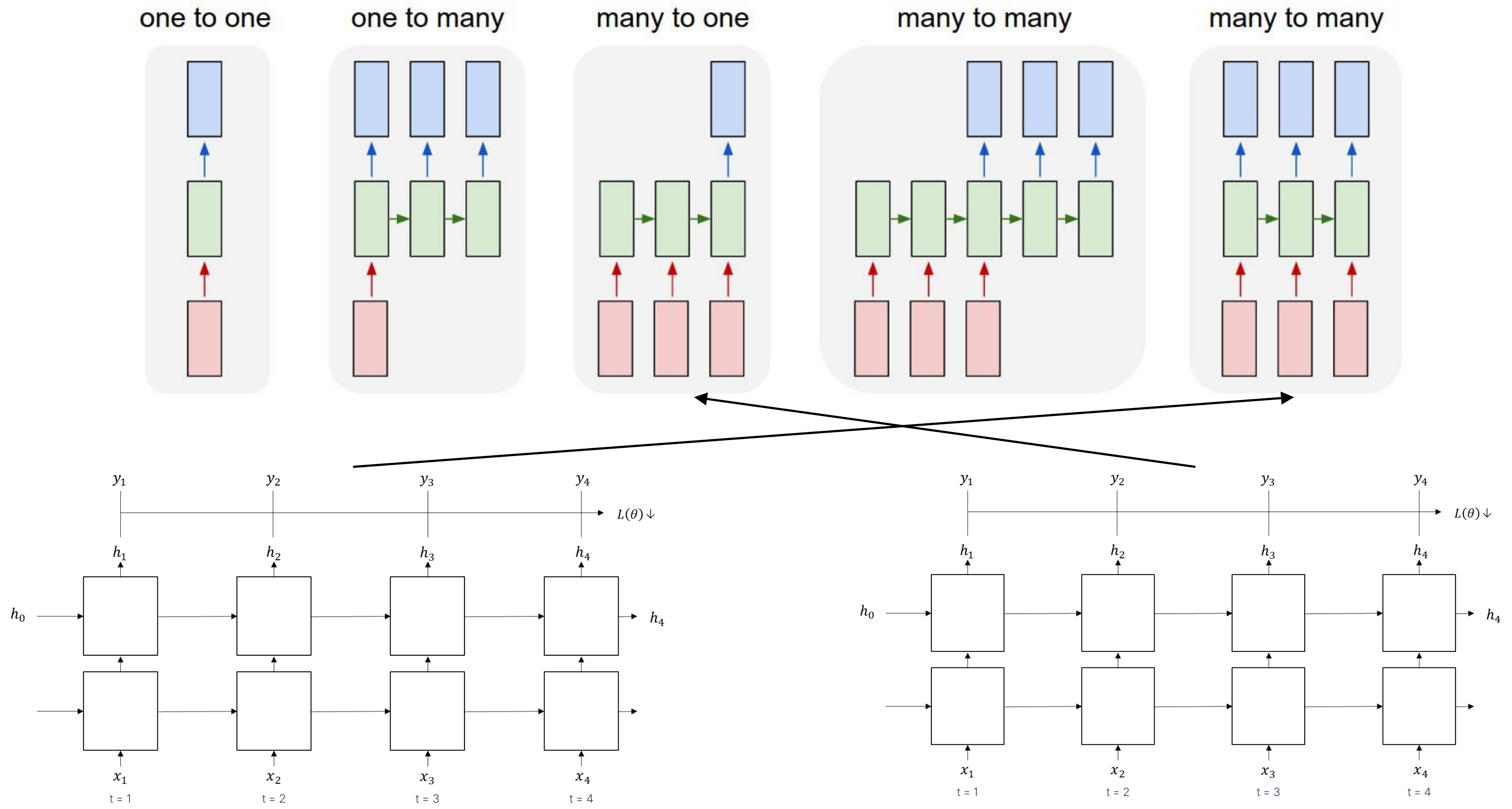
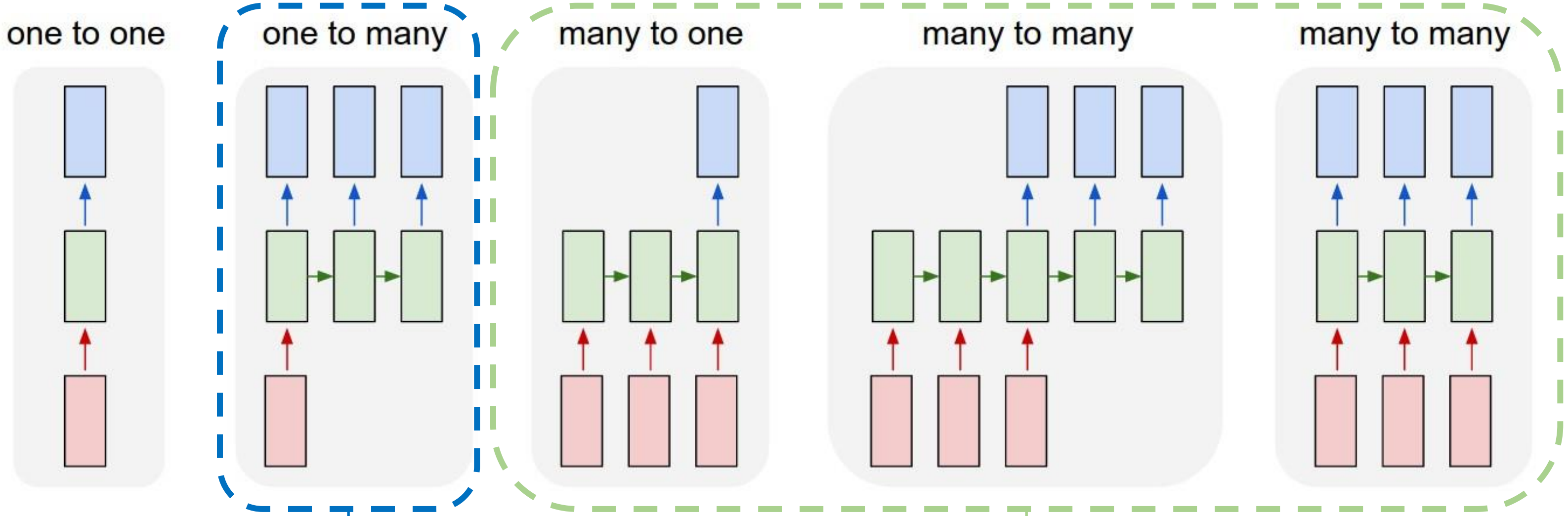


사진 출처: https://karpathy.github.io/2015/05/21/rnn-effectiveness/

RNN - Two Approaches

Two Approaches

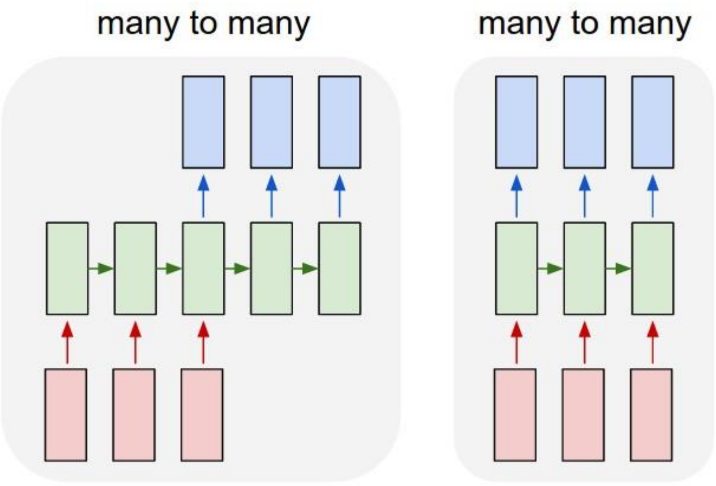


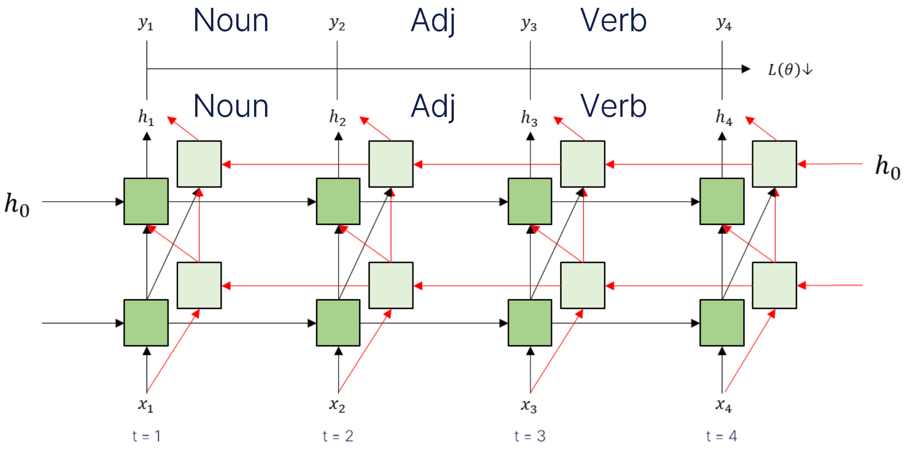
1. Non-autoregressive(Non-generative) → 문장 전체를 받음
   - 현재 상태가 앞/뒤 상태를 통해 정해지는 경우
     - e.g. POS Tagging, Text Classification
   - Bidirectional RNN 사용 권장

2. Autoregressive(Generative) → 뒷 단어를 모름
   - 현재 상태가 과거 상태에 의존하여 정해지는 경우
     - e.g. Natural Laguage Generation, Machine Translation
     - 문장 생성, 챗봇 등
   - One-to-Many Case 해당
   - Bidirectional RNN 사용 불가!

RNN - Application of RNNs

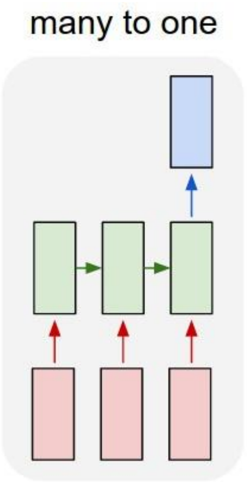
Application of RNNs

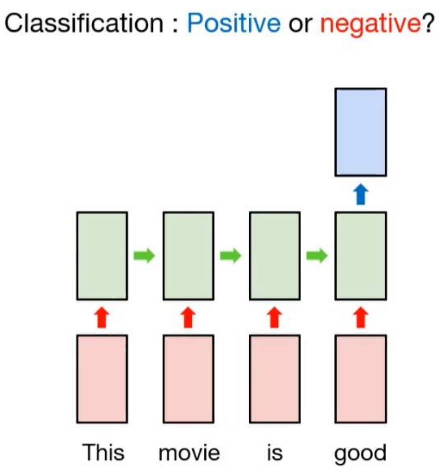


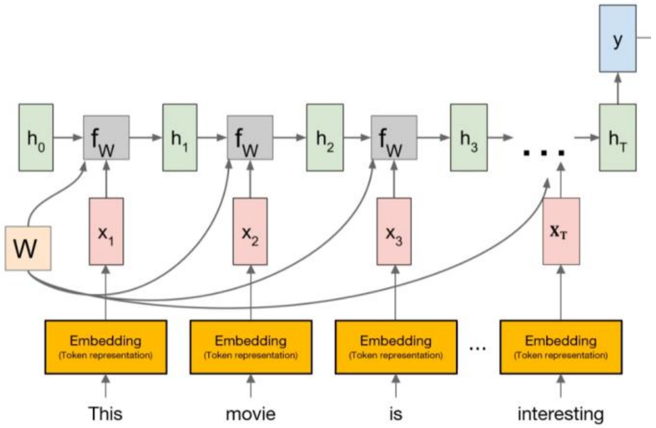


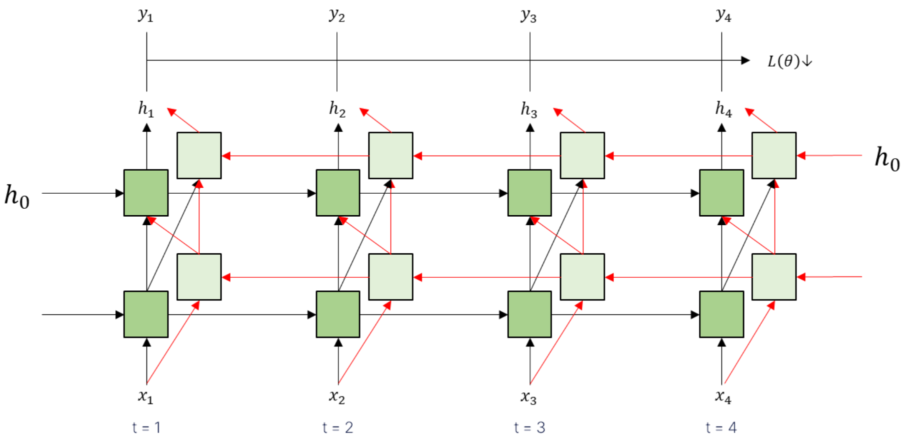
Pos Tagging, MRC

Application of RNNs





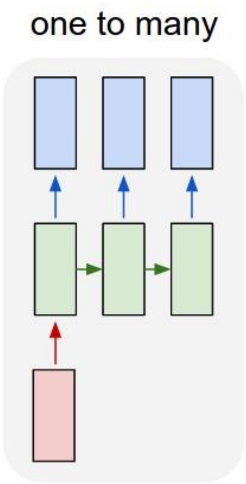


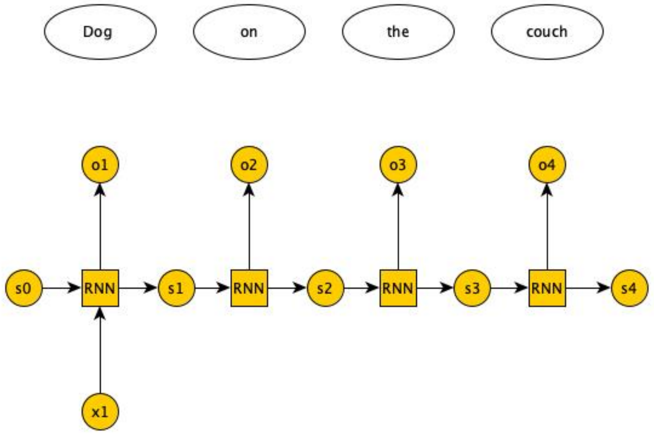


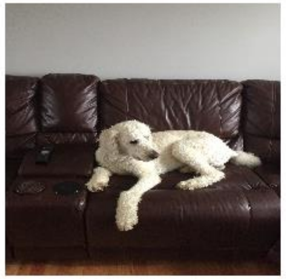
Text Classification

RNN - Application of RNNs

Application of RNNs





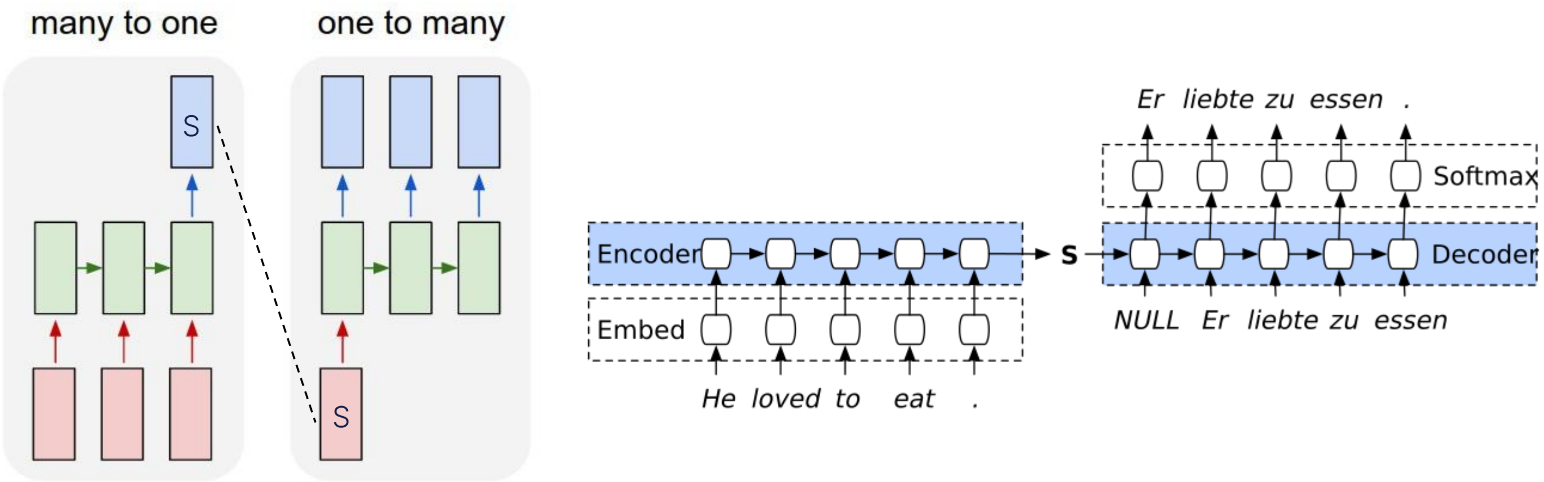


NLG, Machine Translation
- e.g. Natural Language Generation
- 대부분의 자연어 생성이 이 case에 해당됨
- x2가 오는게 아니라, 이전 time-step의 output값이 온다
- 다음 time-step을 모르기 때문에, biditrecional 불가능!

사진 출처: https://stats.stackexchange.com/questions/404955/examples-of-one-to-many-for-rnn-lstm

RNN - Application of RNNs

Application of RNNs



- AutoEncoder와의 차이는 Sequential Data가 들어오냐 아니냐의 차이
- Sequence를 한 점으로 압축하고, 그 점을 다시 Sequence로 만든다

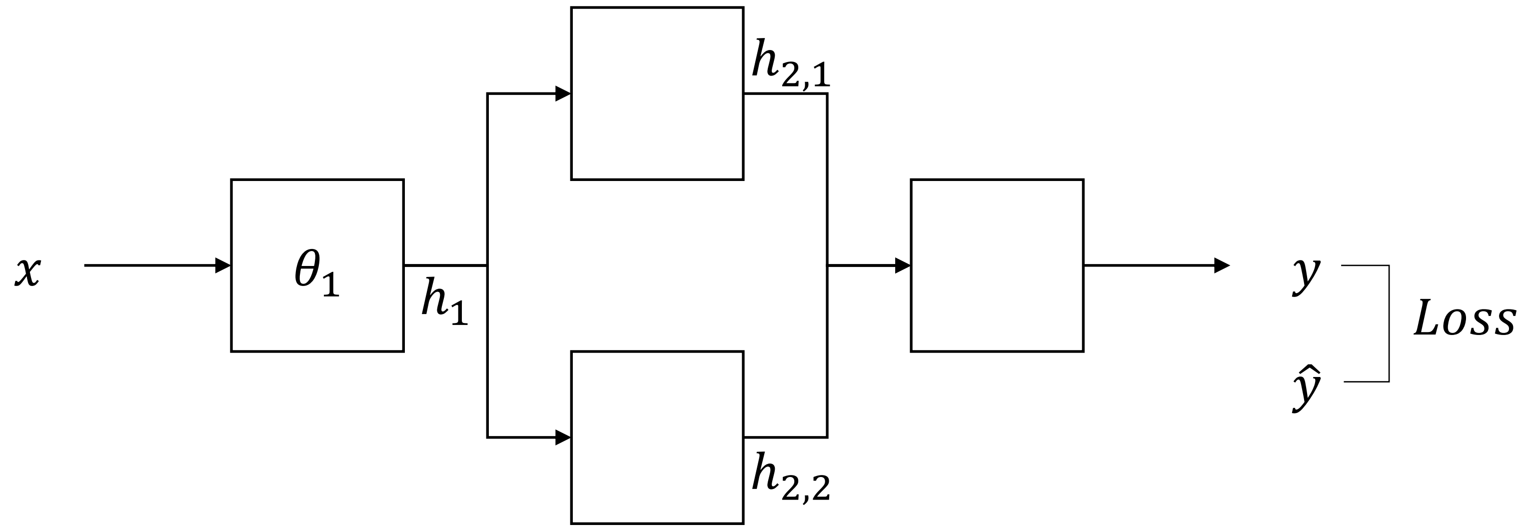
사진 출처: https://stats.stackexchange.com/questions/404955/examples-of-one-to-many-for-rnn-lstm

BackPropagation Through Time

1. Back-propagation in RNN
2. Equations

RNN - BPTT

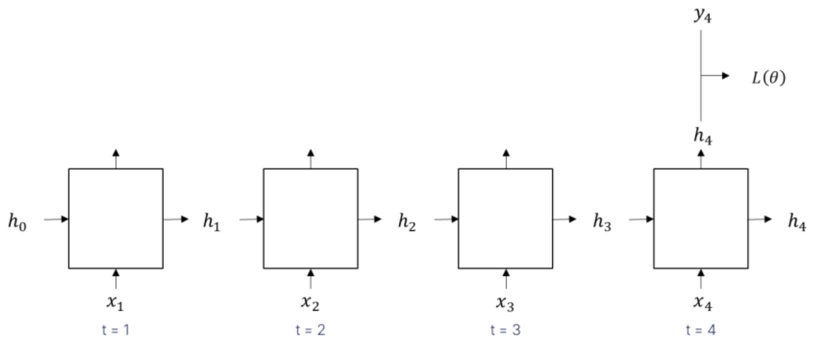
Back-propagation in RNN



$$\frac{\partial \mathcal{L}}{\partial \theta_1} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \left(\frac{\partial \hat{y}}{\partial h_{2,1}} \frac{\partial h_{2,1}}{\partial h_1} + \frac{\partial \hat{y}}{\partial h_{2,2}} \frac{\partial h_{2,2}}{\partial h_1} \right) \frac{\partial h_1}{\partial \theta_1}$$

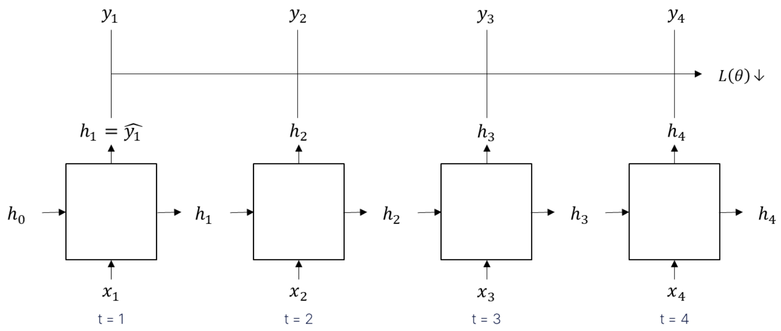
RNN - BPTT
Back-propagation in RNN

- Many to One



- 시간에 따라 gradient의 크기가 달라진다
- Gradient Exploding도 발생 가능

- Many to Many(Single layer RNN)



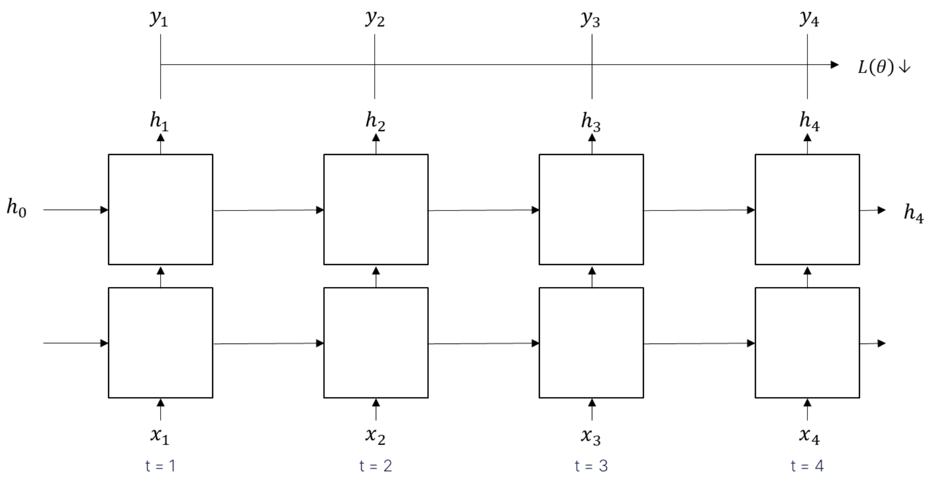
- 한 time-step 안에서도
  2개의 feedforward 경로가 존재할 수 있다

- time-step에 대해서도 더해질 뿐만 아니라,
  한 time-step 내에서도 2개의 feedforward 경로가 존재 → 이를 다 더해야 한다

RNN - BPTT

Back-propagation in RNN

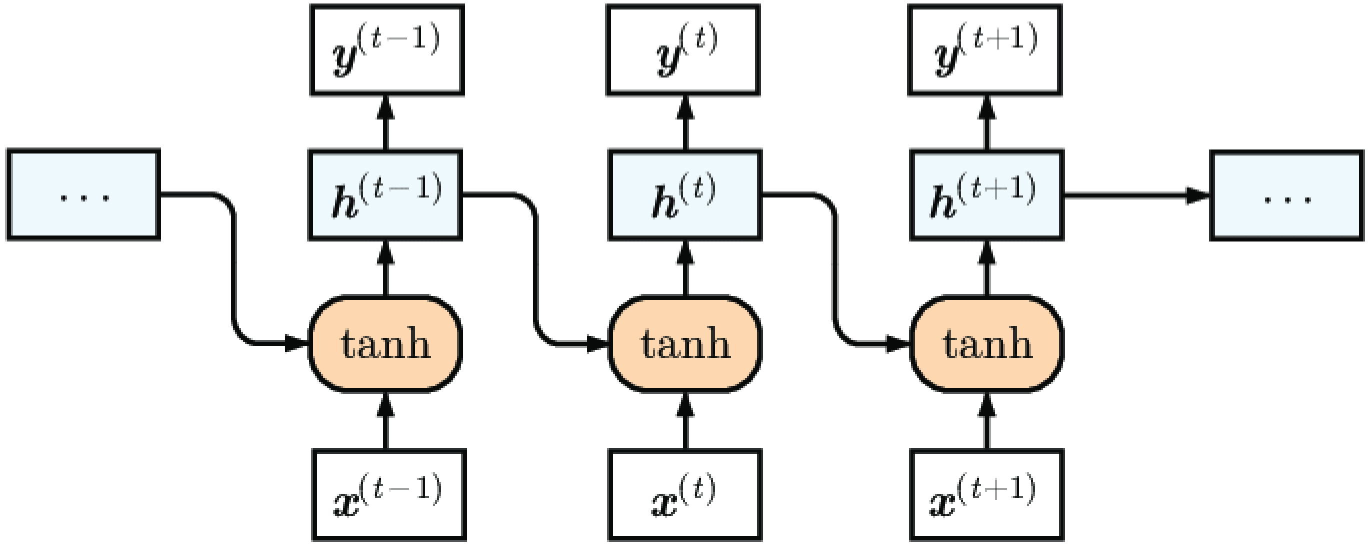
- Many to Many(Multi-layered RNN)



- 각 time-step마다 2가지 경로가 존재 → 경로가 너무 많다

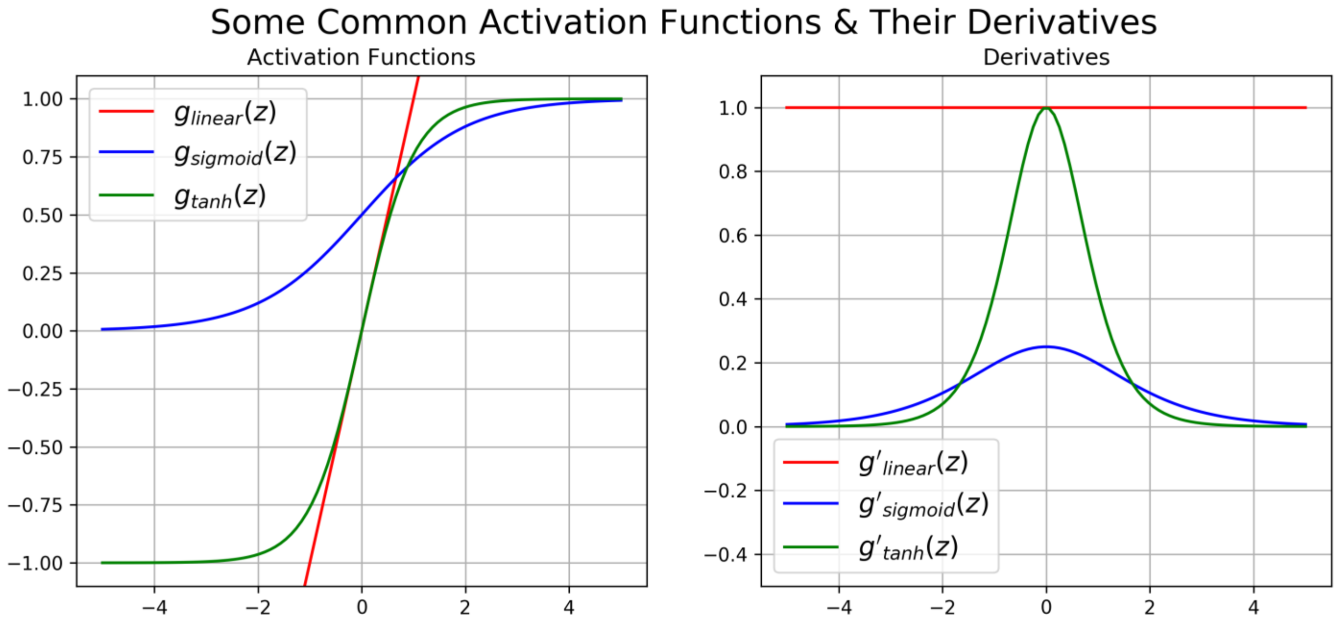
RNN - BPTT

TanH in Vanilla RNN



$$\begin{aligned}\hat{y}_t = h_t &= f(x_t, h_{t-1}; \theta) \\ &= \tanh(W_{ih}x_t + b_{ih} + W_{hh}h_{t-1} + b_{hh}) \\ &\text{where } \theta = \{W_{ih}, b_{ih}, W_{hh}, b_{hh}\}.\end{aligned}$$

RNN - BPTT
Gradient Vanishing by TanH

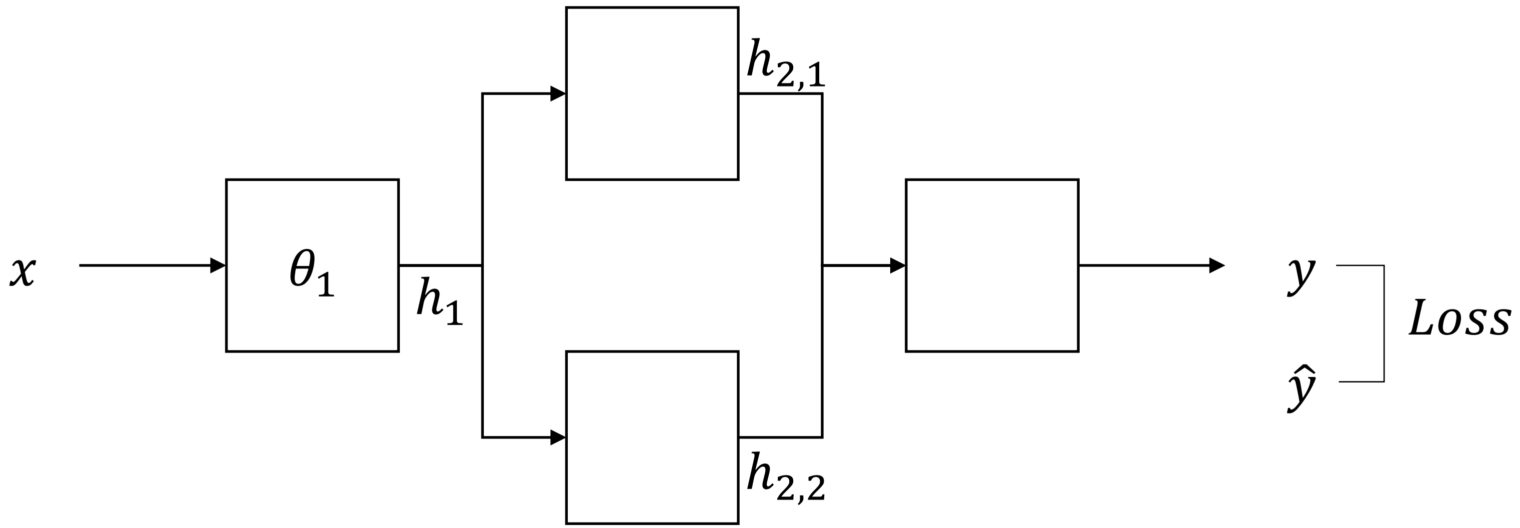


tanh에 들어오는 값이 0일 때만 1이고, 이외는 1보다 작아 Gradient가 줄어든다
초반부의 hidden state값이 중요해지지 않는다...

사진 출처: https://dustinstansbury.github.io/theclevermachine/derivation-common-neural-network-activation-functions

RNN – BPTT Equation

Back-propagation in RNN

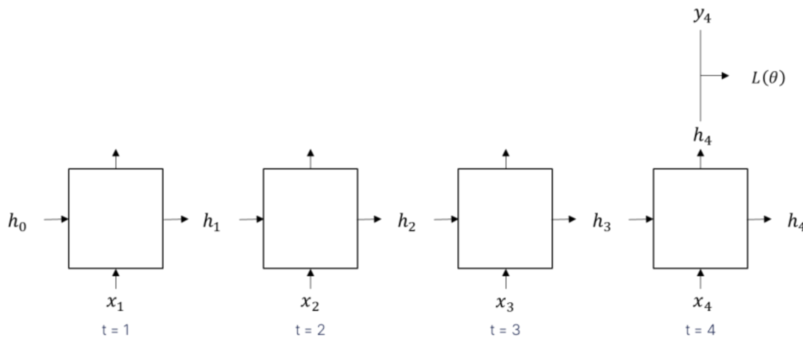


$$\frac{\partial \mathcal{L}}{\partial \theta_1} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \left(\frac{\partial \hat{y}}{\partial h_{2,1}} \frac{\partial h_{2,1}}{\partial h_1} + \frac{\partial \hat{y}}{\partial h_{2,2}} \frac{\partial h_{2,2}}{\partial h_1} \right) \frac{\partial h_1}{\partial \theta_1}$$

RNN – BPTT Equation

Back-propagation in RNN

- Many to One

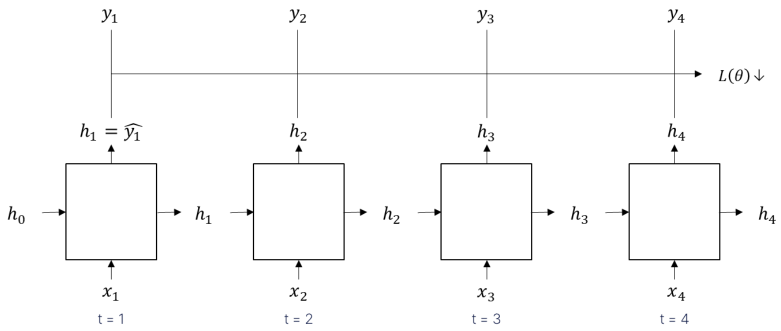


$$h_t = f(x_t, h_{t-1}; \psi)$$
$$\hat{y} = g(h_T; \phi)$$
$$\theta = \{\phi, \psi\}$$

$$\frac{\partial \mathcal{L}(\theta)}{\partial \phi} = \frac{\partial \mathcal{L}(\theta)}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \phi}$$

$$\frac{\partial \mathcal{L}(\theta)}{\partial \psi} = \frac{\partial \mathcal{L}(\theta)}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial h_4} \frac{\partial h_4}{\partial \psi} + \frac{\partial \mathcal{L}(\theta)}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial h_4} \frac{\partial h_4}{\partial h_3} \frac{\partial h_3}{\partial \psi} + \frac{\partial \mathcal{L}(\theta)}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial h_4} \frac{\partial h_4}{\partial h_3} \frac{\partial h_3}{\partial h_2} \frac{\partial h_2}{\partial \psi} + \frac{\partial \mathcal{L}(\theta)}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial h_4} \frac{\partial h}{\partial h}$$
$$= \sum_{t=1}^{T} \frac{\partial \mathcal{L}(\theta)}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial h_4} \left(\prod_{i=t}^{T-1} \frac{\partial h_{i+1}}{\partial h_i} \right) \frac{\partial h_t}{\partial \psi}$$

- Many to Many(Single layer RNN)



$$h_t = f(x_t, h_{t-1}; \psi)$$
$$\hat{y}_t = g(h_t; \phi)$$
$$\theta = \{\phi, \psi\}$$

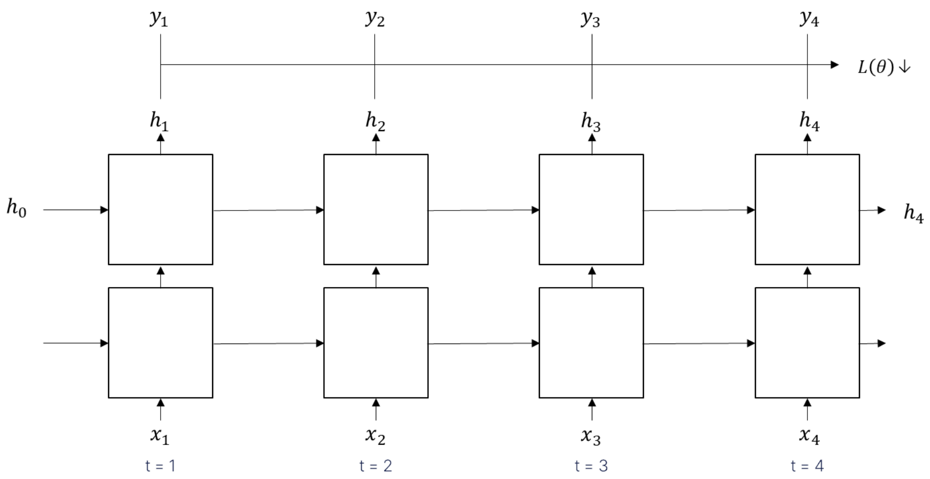
$$\frac{\partial \mathcal{L}(\theta)}{\partial \phi} = \sum_{t=1}^{T} \frac{\partial \mathcal{L}(\theta)}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial \phi}$$

$$\frac{\partial \mathcal{L}(\theta)}{\partial \psi} = \frac{\partial \mathcal{L}(\theta)}{\partial \hat{y}_4} \frac{\partial \hat{y}_4}{\partial h_4} \frac{\partial h_4}{\partial \psi} + \frac{\partial \mathcal{L}(\theta)}{\partial \hat{y}_4} \frac{\partial \hat{y}_4}{\partial h_4} \frac{\partial h_4}{\partial h_3} \frac{\partial h_3}{\partial \psi} + \frac{\partial \mathcal{L}(\theta)}{\partial \hat{y}_4} \frac{\partial \hat{y}_4}{\partial h_4} \frac{\partial h_4}{\partial h_3} \frac{\partial h_3}{\partial h_2} \frac{\partial h_2}{\partial \psi} + \frac{\partial \mathcal{L}(\theta)}{\partial \hat{y}_4} \frac{\partial \hat{y}_4}{\partial h_4} \frac{\partial h_4}{\partial h_3} \frac{\partial h_3}{\partial h_2} \frac{\partial h_2}{\partial h_1} \frac{\partial h}{\partial \psi}$$
$$+ \frac{\partial \mathcal{L}(\theta)}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial h_3} \frac{\partial h_3}{\partial \psi} + \frac{\partial \mathcal{L}(\theta)}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial h_3} \frac{\partial h_3}{\partial h_2} \frac{\partial h_2}{\partial \psi} + \frac{\partial \mathcal{L}(\theta)}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial h_3} \frac{\partial h_3}{\partial h_2} \frac{\partial h_2}{\partial h_1} \frac{\partial h_1}{\partial \psi}$$
$$+ \frac{\partial \mathcal{L}(\theta)}{\partial \hat{y}_2} \frac{\partial \hat{y}_2}{\partial h_2} \frac{\partial h_2}{\partial \psi} + \frac{\partial \mathcal{L}(\theta)}{\partial \hat{y}_2} \frac{\partial \hat{y}_2}{\partial h_2} \frac{\partial h_2}{\partial h_1} \frac{\partial h_1}{\partial \psi}$$
$$+ \frac{\partial \mathcal{L}(\theta)}{\partial \hat{y}_1} \frac{\partial \hat{y}_1}{\partial h_1} \frac{\partial h_1}{\partial \psi}$$
$$= \sum_{t=1}^{T} \sum_{k=1}^{t} \frac{\partial \mathcal{L}(\theta)}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial h_t} \left(\prod_{i=k}^{t-1} \frac{\partial h_{i+1}}{\partial h_i} \right) \frac{\partial h_k}{\partial \psi}$$

RNN – BPTT Equation

Back-propagation in RNN

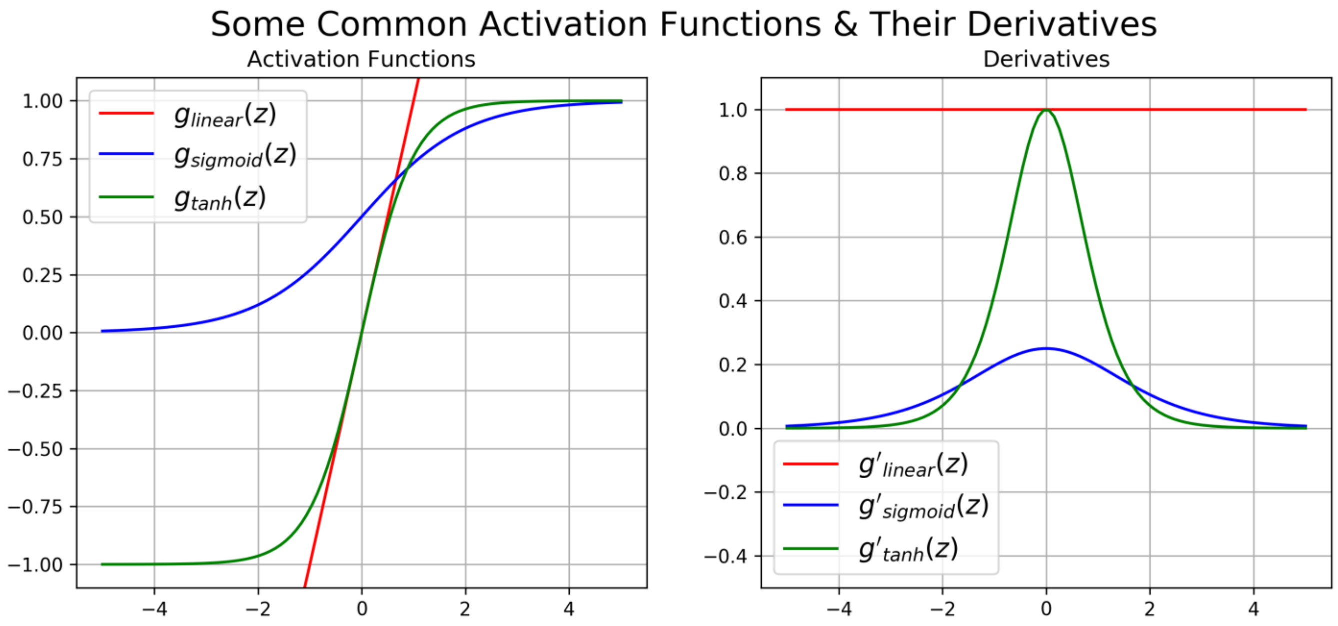
- Many to Many(Multi-layered RNN)



- 계산하기 싫어요...

RNN – BPTT Equation

Gradient Vanishing



$$\frac{\partial \mathcal{L}(\theta)}{\partial \psi} = \sum_{t=1}^{T} \frac{\partial \mathcal{L}(\theta)}{\partial \hat{y}_4} \frac{\partial \hat{y}_4}{\partial h_4} \left(\prod_{i=t}^{T-1} \frac{\partial h_{i+1}}{\partial h_i} \right) \frac{\partial h_t}{\partial \psi}$$

$$\frac{\partial h_{t+1}}{\partial h_t} \leq 1$$

사진 출처: https://www.researchgate.net/figure/Illustration-of-the-vanilla-recurrent-neural-network_fig2_320726805

With HAI, Fly High

Hanyang Artificial Intelligence