RNN

강 의

임 경 태

CONTENTS

목적 : RNN의 구조와 RNN개념 이해

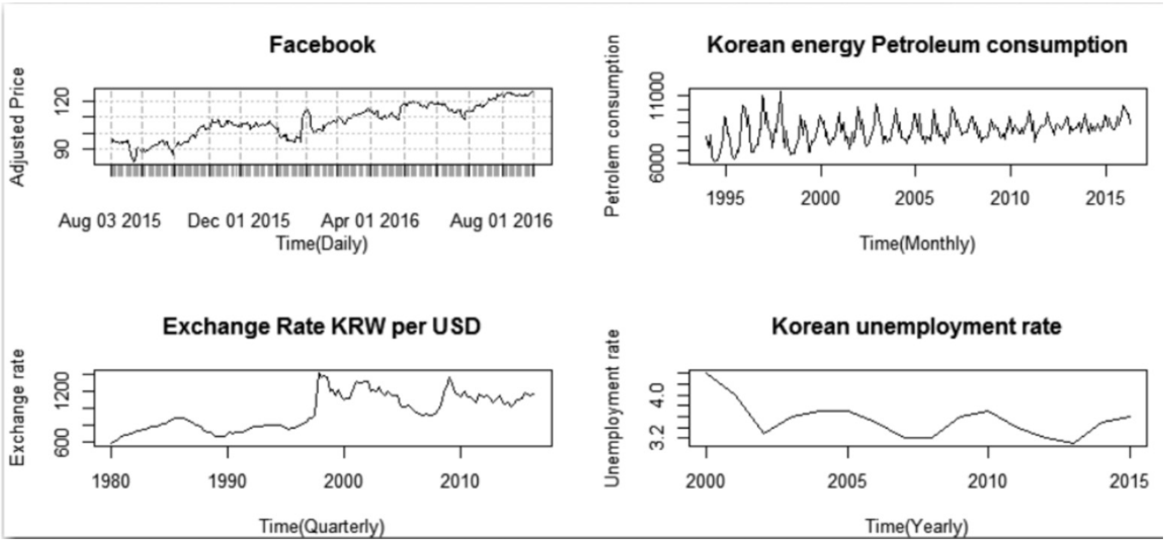
목표 : RNN에 구조와 사용하는 이유 이해

내용 : RNN 구조, 동작 원리

CONTENTS

시계열 데이터??

- 시계열(時系列, 영어: time series)은 일정 시간 간격으로 배치된 데이터들의 수열을 말한다. (위키)
  - 시계열(time series) 데이터는 관측치가 시간적 순서를 가진 데이터이다.
  - 과거의 데이터를 통해서 현재의 움직임 그리고 미래를 예측하는데 사용된다



이미지 https://velog.io/@sdubee10/ZeroBase-%EC%8B%9C%EA%B3%84%EC%97%B4%EB%8D%B0%EC%9D%B4%ED%84%B0-Time-Series-Data

RNN 개요

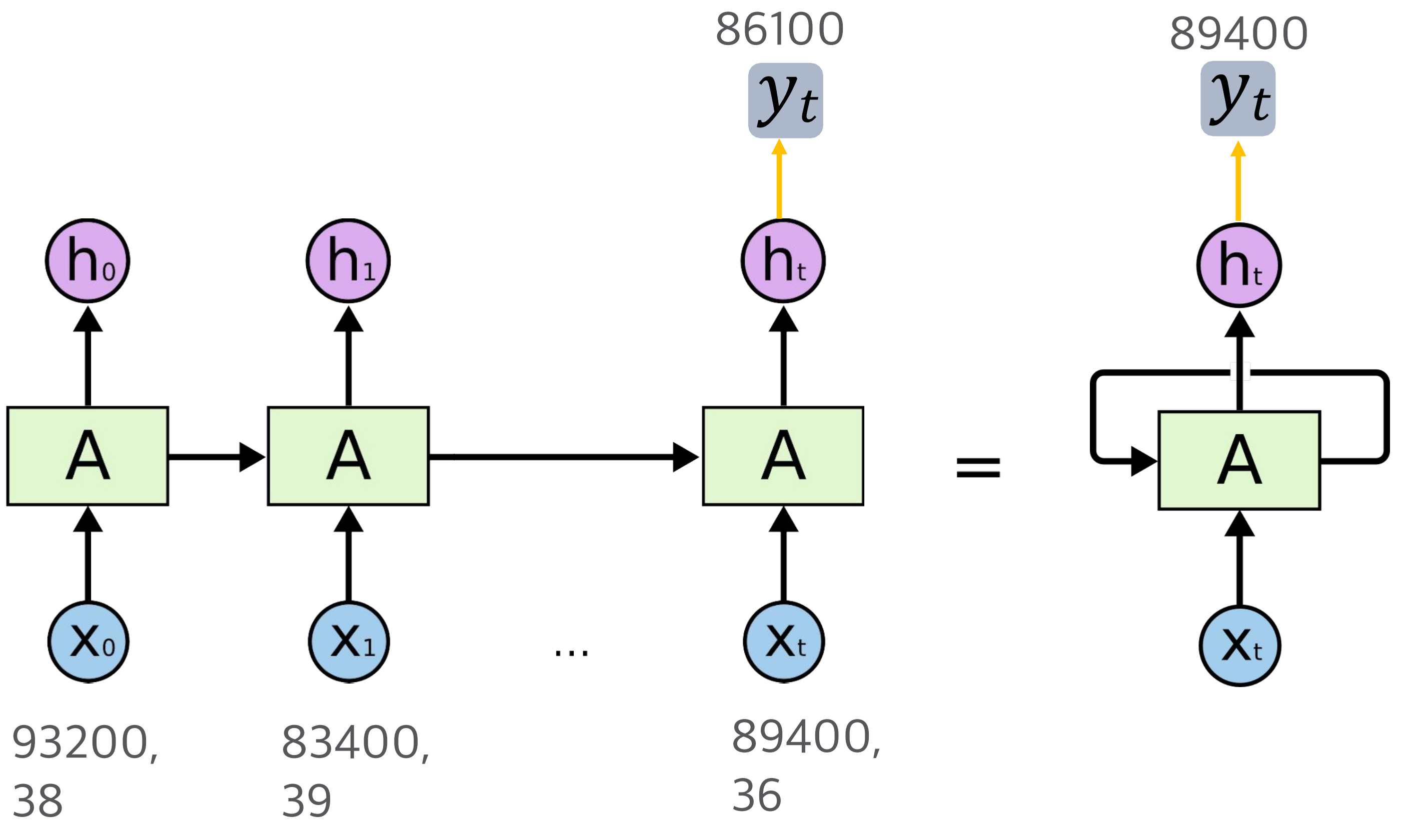
시계열 데이터??

- NCSoft의 2년 주가 흐름을 기반으로 내일 주가를 예측하려고 한다. 어떻게 하면 될까?
  - (1) 2년치 데이터 전부를 입력으로 FCN에 넣으면 될까? --> 너무 많음..
  - (2) 그러면 월~금 5일 데이터를 FCN에 넣자 --> 월->화->수 연속된 시간 순서관계 모델링 못함
  - (3) 시간에 따라 이전 가격을 고려해서 모델링 하는 방법은 없을까?

date	num_search	stock_price
2021.3.21	38	932000
2021.3.28	39	834000
2021.4.4	36	885000
2021.4.11	31	906000
2021.4.18	32	894000
2021.4.25	27	861000
2021.5.2	27	820000
2021.5.9	26	850000
2021.5.16	29	823000
2021.5.23	26	856000
2021.5.30	29	854000
2021.6.6	30	858000
2021.6.13	31	848000
2021.6.20	37	825000
2021.6.27	34	820000
2021.7.4	40	834000
2021.7.11	29	778000
2021.7.18	27	809000
2021.7.25	25	809000
2021.8.1	24	812000
2021.8.8	31	790000
2021.8.15	35	853000
2021.8.22	38	709000

What is RNN(Recurrent Neural Network)?

- 연속적이며 순서가 있는 데이터(시계열 데이터, 자연어, 음성 등)에 적합한 딥러닝 모델



A	B	C
date	num_search	stock_price
2021.3.21	38	932000
2021.3.28	39	834000
2021.4.4	36	885000
2021.4.11	31	906000
2021.4.18	32	894000
2021.4.25	27	861000
2021.5.2	27	820000
2021.5.9	26	850000
2021.5.16	29	823000
2021.5.23	26	856000
2021.5.30	29	854000
2021.6.6	30	858000
2021.6.13	31	848000
2021.6.20	37	825000
2021.6.27	34	820000
2021.7.4	40	834000
2021.7.11	29	778000
2021.7.18	27	809000
2021.7.25	25	809000
2021.8.1	24	812000
2021.8.8	31	790000
2021.8.15	35	853000
2021.8.22	38	709000

RNN 개요

intuition of RNN

(input + empty_hidden) -> hidden -> output
(input + prev_hidden) -> hidden -> output
(input + prev_hidden) -> hidden -> output
(input + prev_hidden) -> hidden -> output

사용하는 것을 색으로 표시

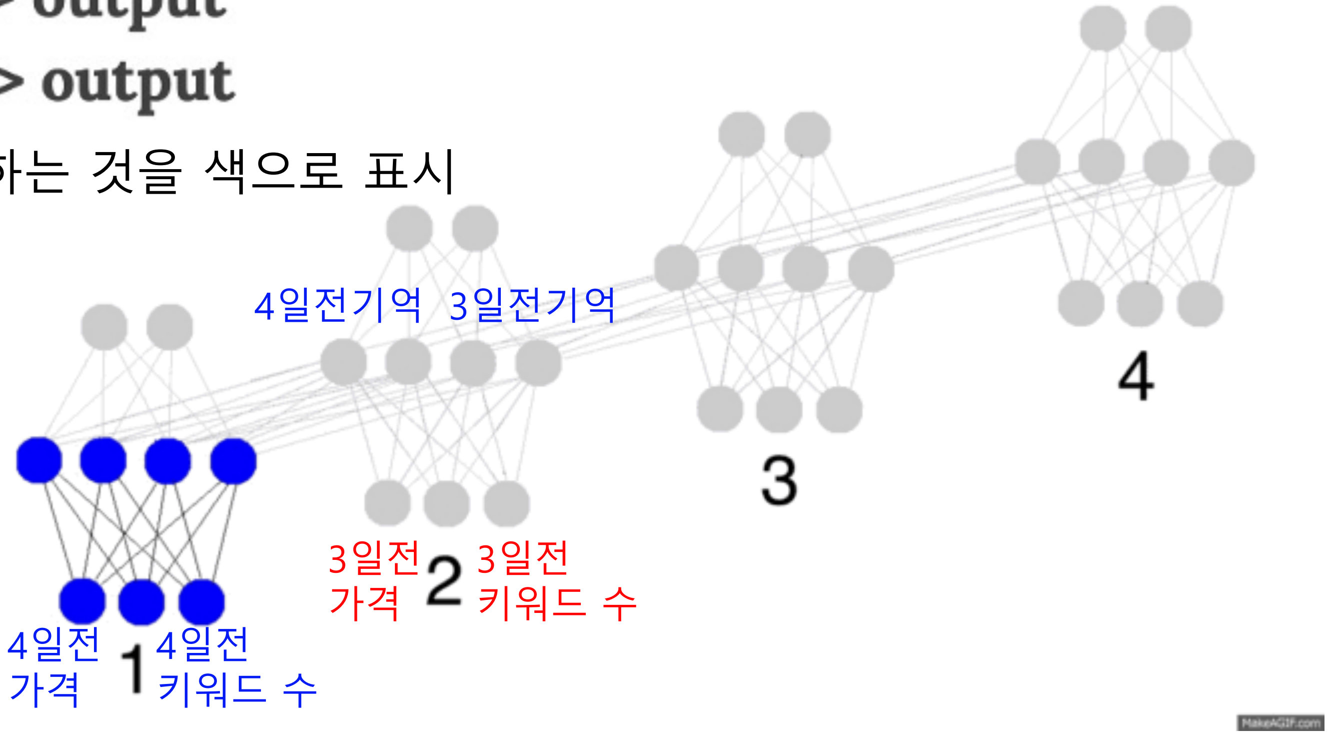
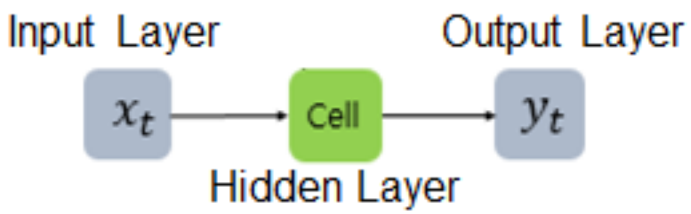


그림 출처 : https://medium.com/@serbanliviu/the-intuition-behind-recurrent-neural-networks-6fce753fe9f0

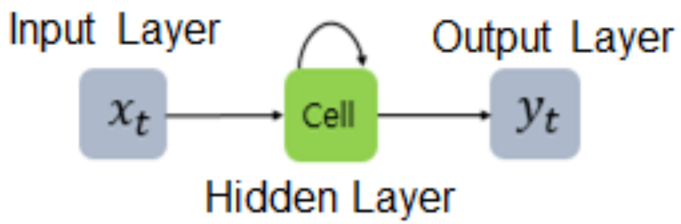
CONTENTS

FCN vs RNN의 구조 비교

Core idea:
Apply the same weights W repeatedly!



일반 신경망

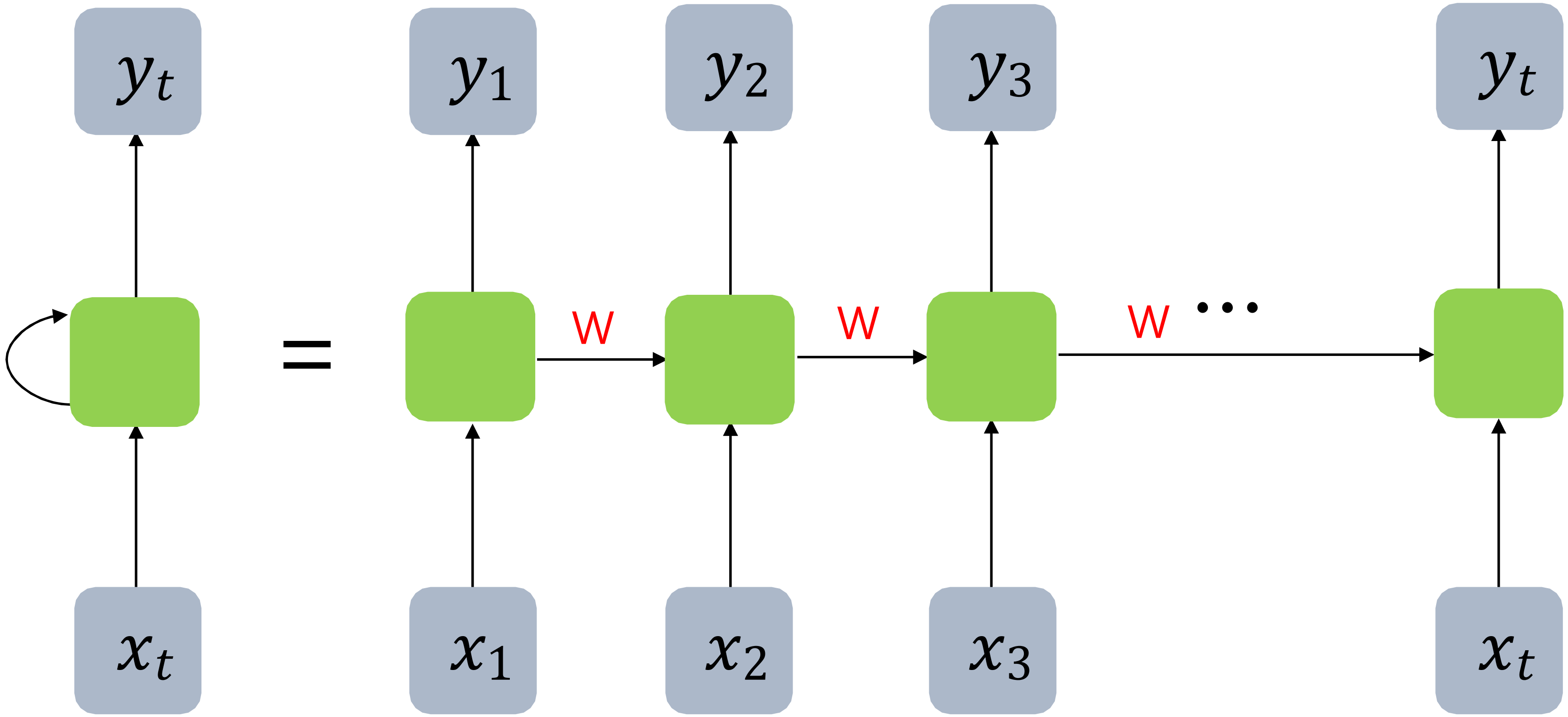


순환 신경망

그림 출처 : https://wikidocs.net/book/2155

RNN 표현

아래의 RNN은 기본적으로 입, 출력이 벡터로 가정되고 있다.



Core idea:
Apply the same weights W repeatedly!

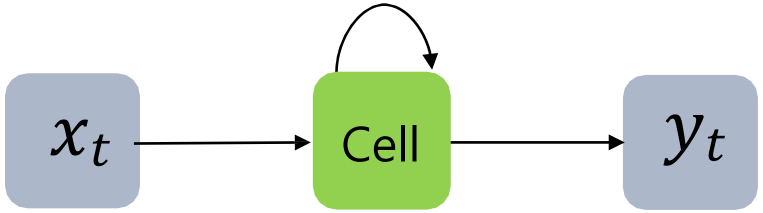
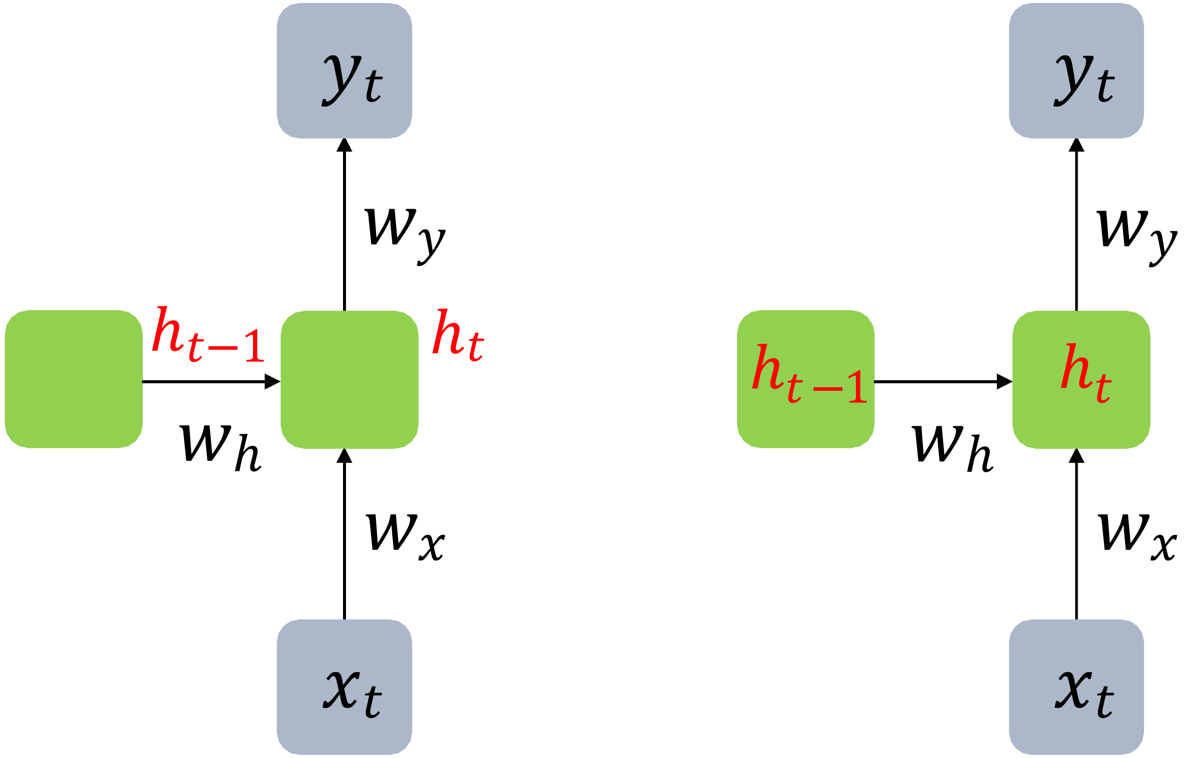


그림 출처 : https://wikidocs.net/book/2155

RNN 표현

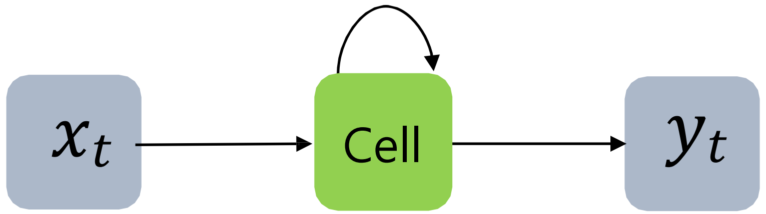


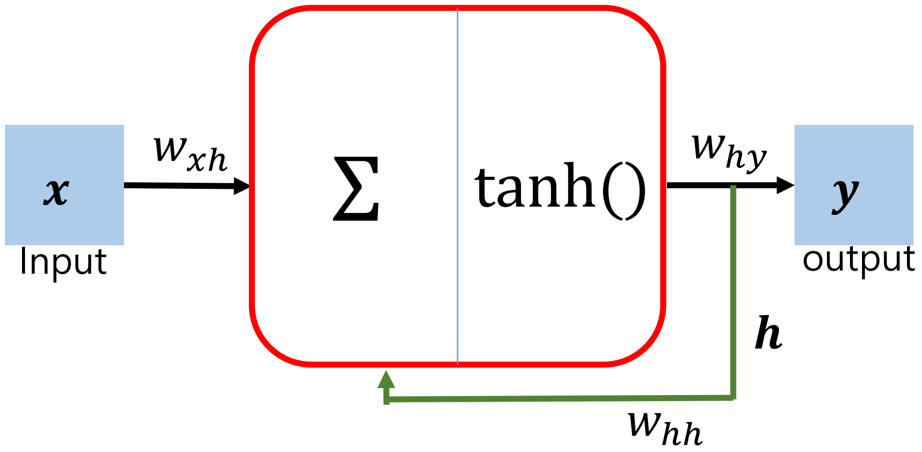
Core idea:
Apply the same weights W repeatedly!

그림 출처 : https://wikidocs.net/book/2155

RNN 구조

RNN 상세 표현





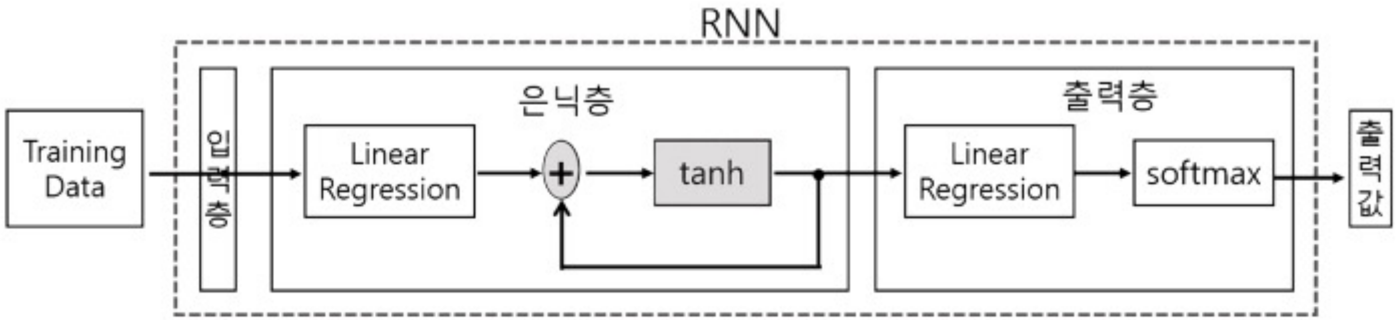
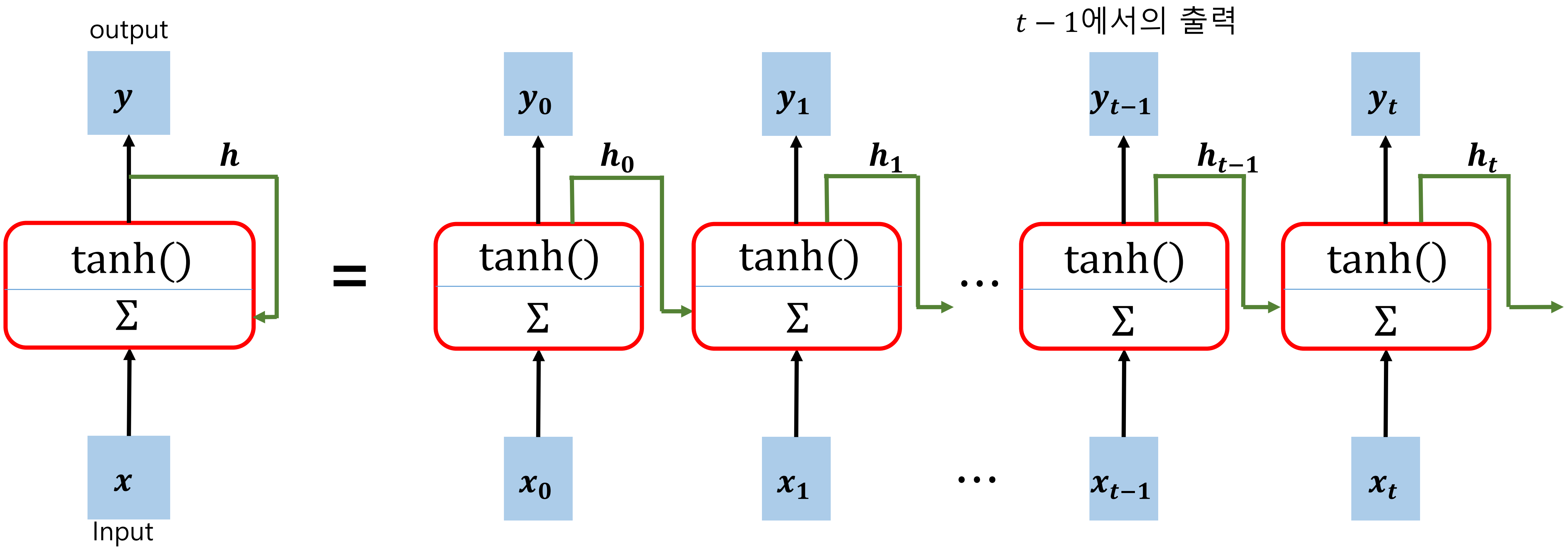


그림 출처 : 머신러닝을 위한 파이썬 한조각

RNN 구조

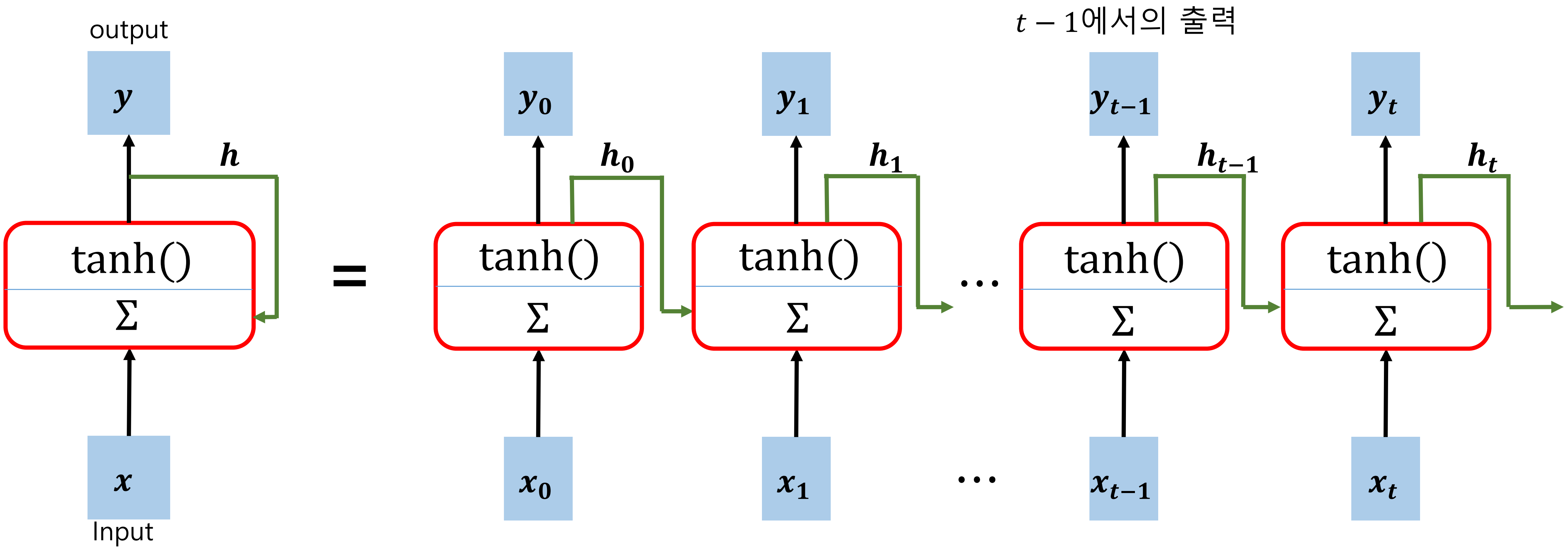
RNN 상세 표현



x_t : 모든 샘플의 입력값 (input)
y_t : 타임 스텝 t에서 각 샘플에 대한 순환 층의 출력값
h_t : hidden state, $h_t = f(h_{t-1}, x_t)$ 즉, 이전 hidden state와 입력값에 의해 현재 hidden state 결정

Hidden state : 다음 시점으로 넘겨줄 정보

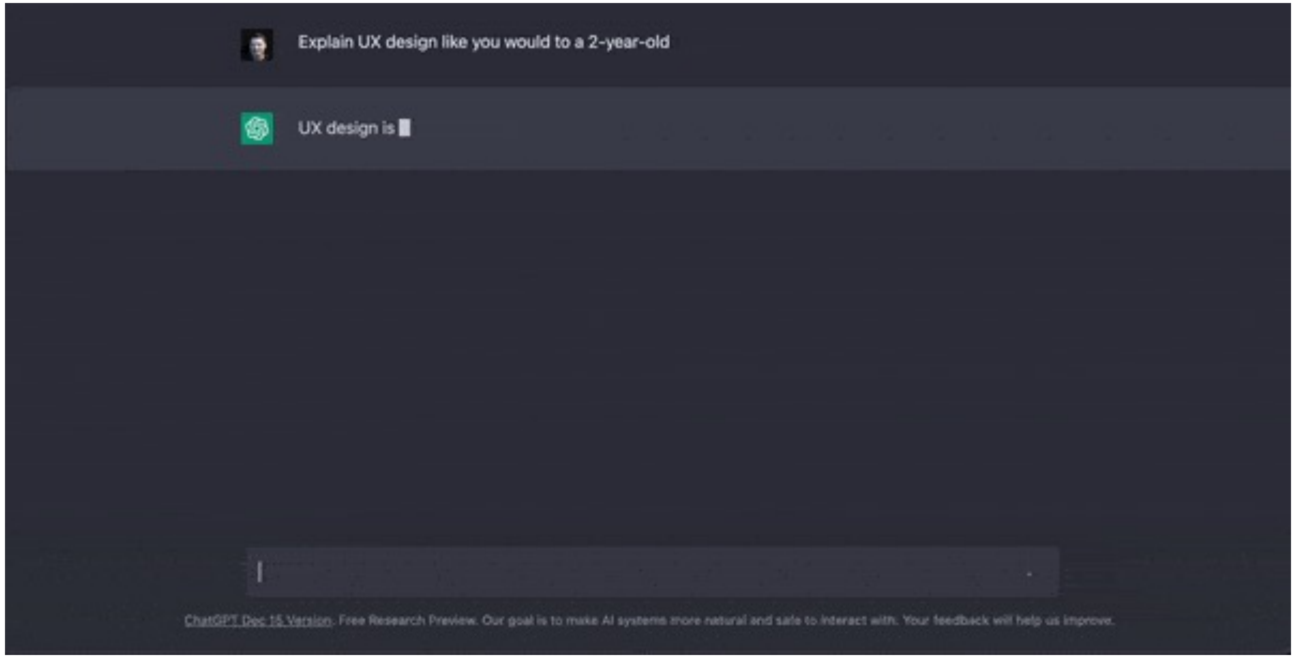
RNN 상세 표현



$$h_t = \tanh(X_t \cdot W_{xh} + h_{t-1} \cdot W_{hh} + b)$$
$$y_t = W_{hy} \cdot h_t$$

RNN을 이용한 자연어 생성 구조

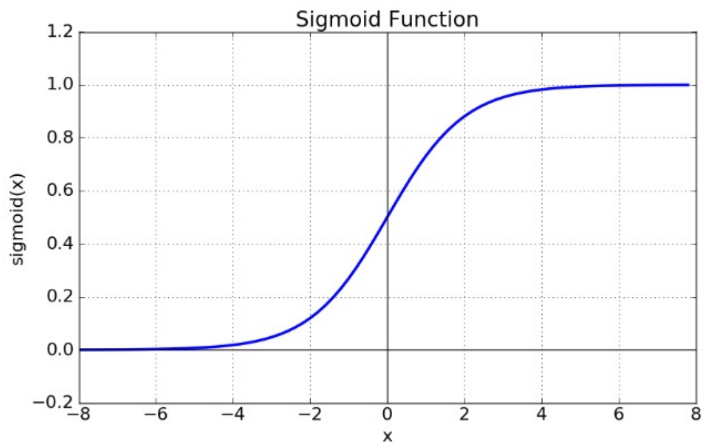
- GPT??

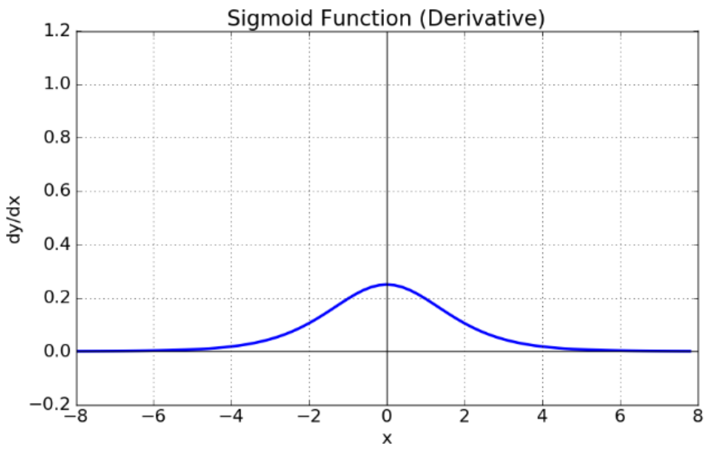


참조: Stanford 224n

Why Hyperbolic Tangent?

- Sigmoid 함수는 음수의 경우 0에 가깝게 표현되며, 이를 미분하면 최대값이 0.25으로 Vanishing Gradient발생
  - Backpropagation 할때 sigmoid의 미분값을 곱하는 과정이 포함됨 따라서, 은닉층의 깊이가 깊어 sigmoid를 많이 사용할 경우 곱해지는 미분값이 0에 가까워 지기 때문에 weight parameter (w) 값들을 업데이트 할 때 매우 작은 범위로 업데이트됨

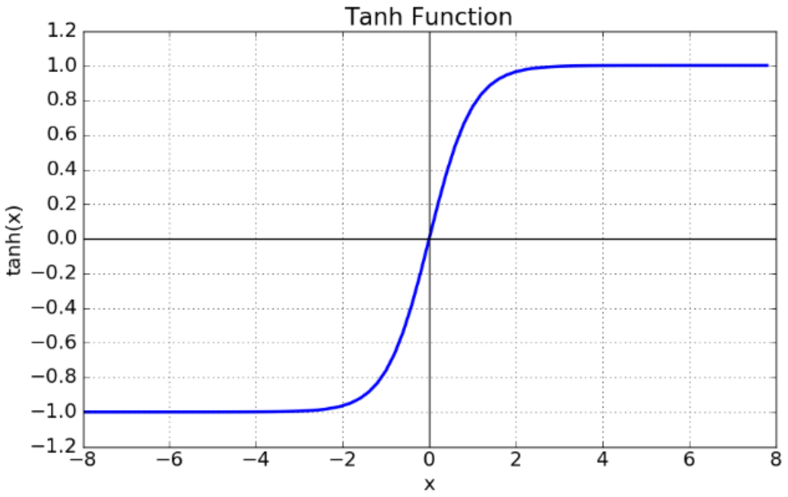


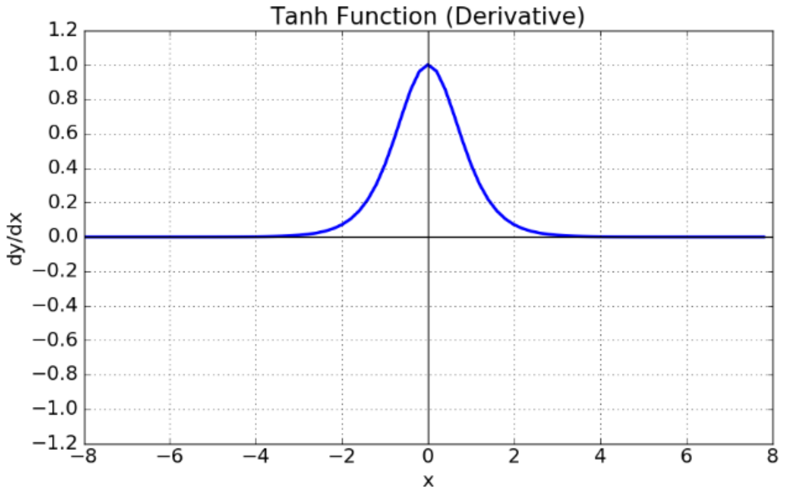


$$\frac{d}{d_x} sigmoid(x) = sigmoid(x)(1 - sigmoid(x))$$

Why Hyperbolic Tangent?

- RNN에서 Vanishing Gradient를 그나마 줄여주기 위해 Tanh Function 사용
  - Tanh 함수의 경우 미분값이 최대 1임. (여전히 1이하의 값이 계산되기 때문에 Vanishing gradient는 발생함)





$$h_t = \tanh(X_t \cdot W_{xh} + h_{t-1} \cdot W_{hh} + b)$$
$$y_t = W_{hy} \cdot h_t$$

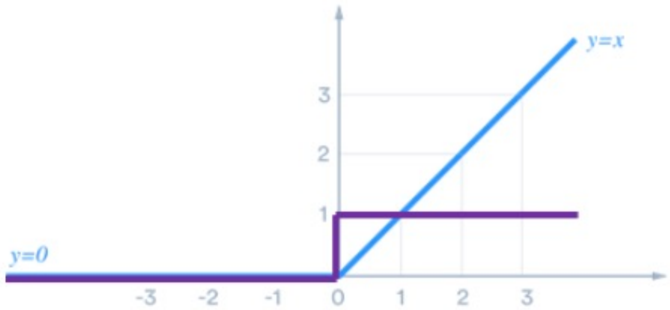
$$tanh(x) = 2\sigma(2x) - 1$$

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

$$tanh'(x) = 1 - tanh^2(x)$$

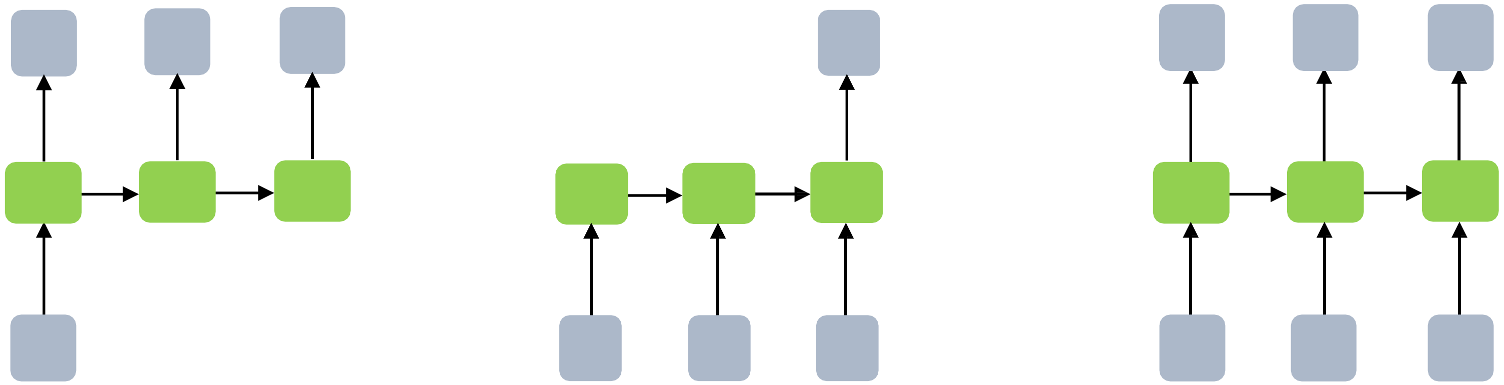
Why Hyperbolic Tangent?

- RNN에서는 Relu를 왜 안쓰죠?
  - RNN의 내부가 계속 순환하는 구조 이므로 relu를 통과한 f(x)의값이 1보다 크게 값이 발산할 수 있기 때문에 적합하지 않음.
  - 결론적으로 tanh는 기울기가 0~1 이기 때문에 normalization 기능이 포함되어 값의 발산을 막을 수 있다고 판단할 수 있음.



ReLU	
$f(x)$	$\max(0,x)$
$\frac{d}{dx}f(x)$	$\begin{cases} 1 & (x \geq 0) \\ 0 & (x < 0) \end{cases}$

RNN 구조

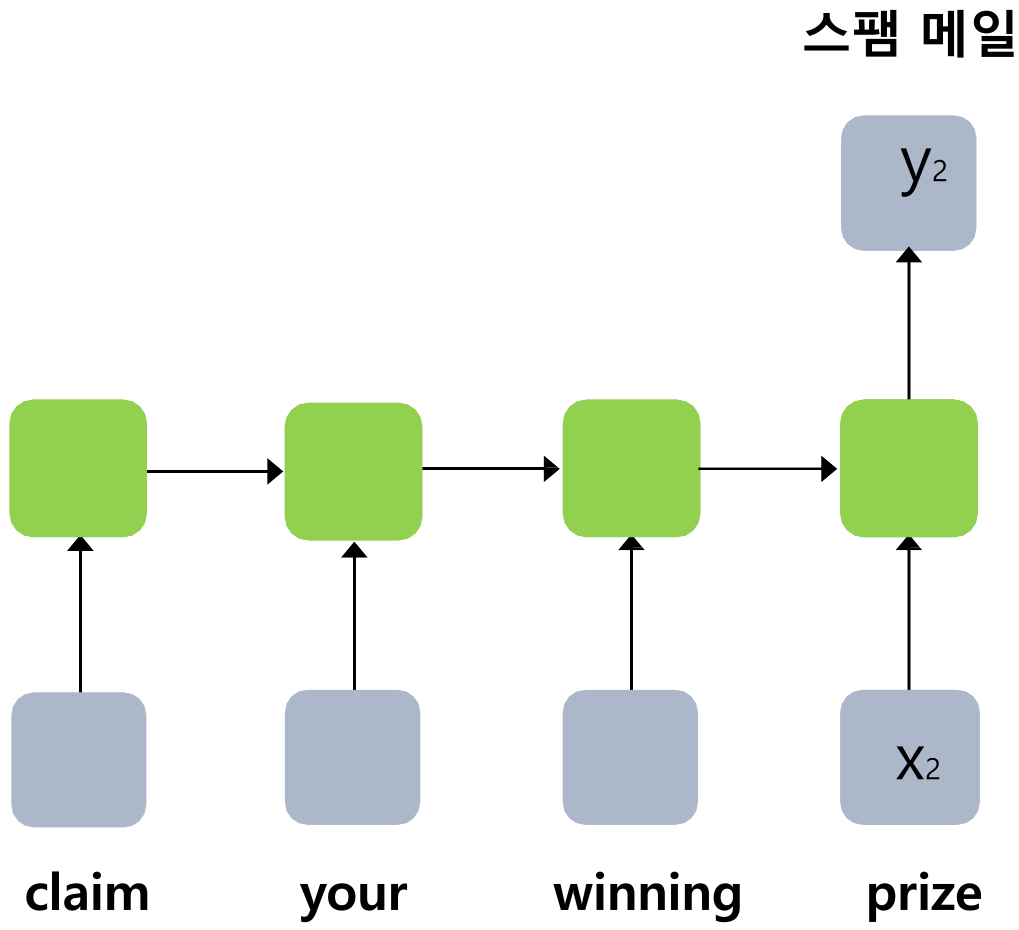


일 대 다(one-to-many)　다 대 일(many-to-one)　다 대 다(many-to-many)

그림 출처 : https://wikidocs.net/book/2155

RNN 구조

다 대 일(many-to-one) 구조의 RNN

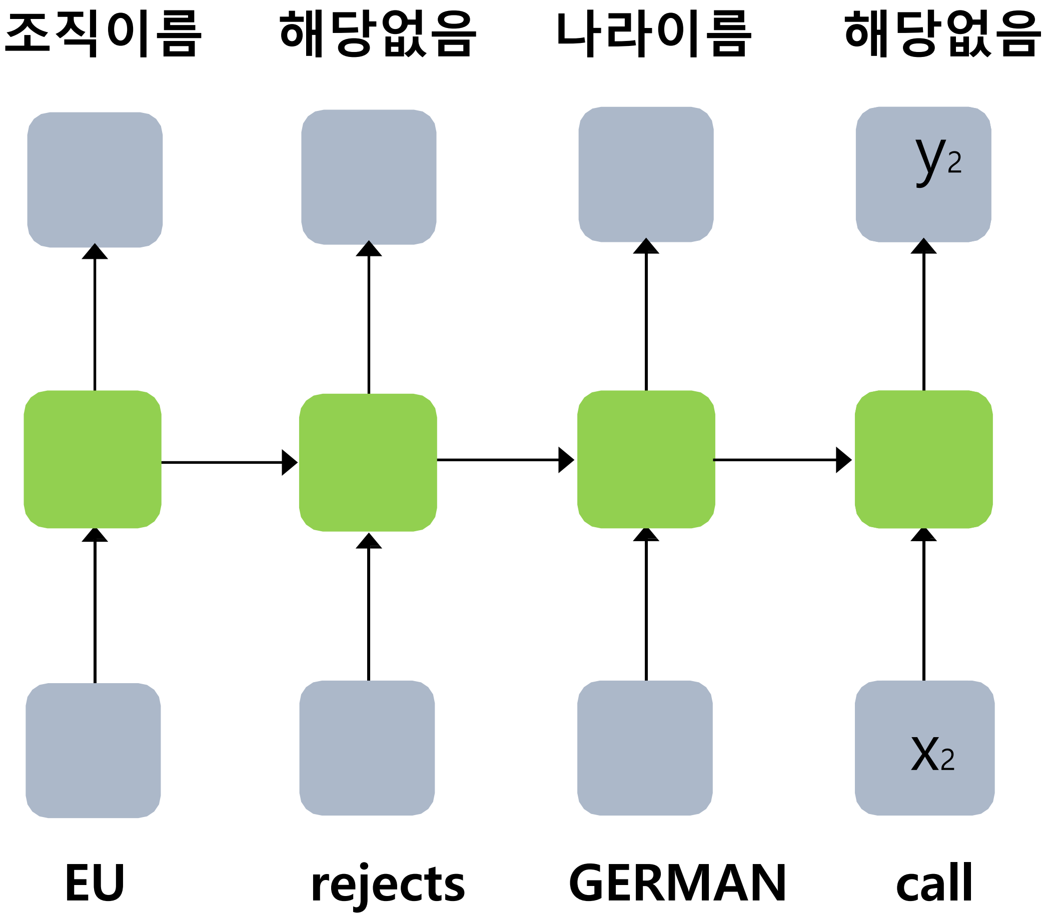


스팸 메일 분류

그림 출처 : https://wikidocs.net/book/2155

RNN 구조

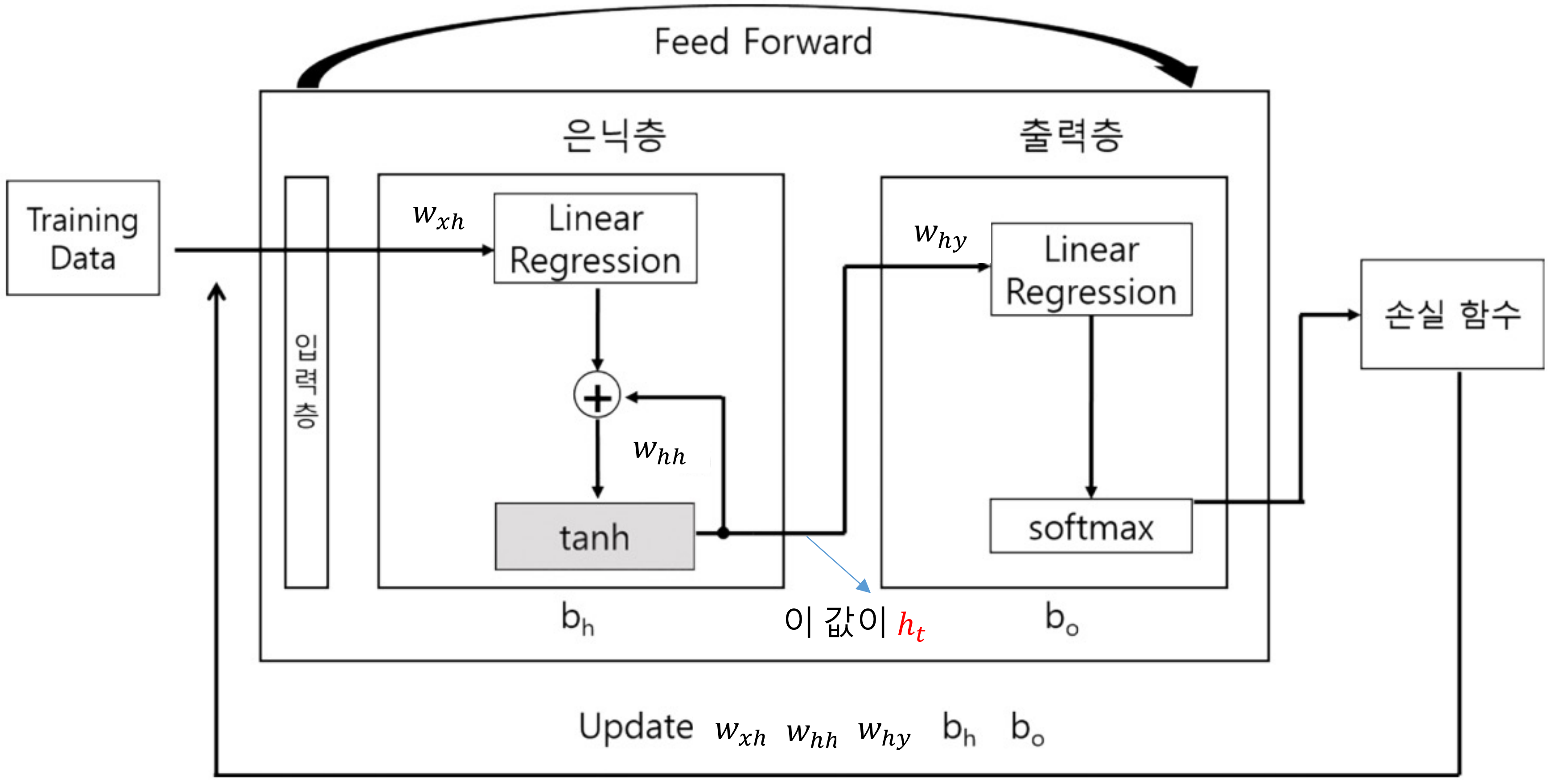
다 대 다(many-to-many) 구조의 RNN



개체명 인식

그림 출처 : https://wikidocs.net/book/2155

RNN 학습구조



$$h_t = \tanh(X_t \cdot W_{xh} + h_{t-1} \cdot W_{hh} + b)$$
$$y_t = W_{hy} \cdot h_t$$

그림 출처 : 머신러닝을 위한 파이썬 한조각

RNN 학습구조

RNN을 벡터와 행렬 연산으로 표현하면?

$$\tanh\left(\underset{D_h \times D_h}{W_h} \times \underset{D_h \times 1}{h_{t-1}} + \underset{D_h \times d}{W_x} \times \underset{d \times 1}{x_t} + \underset{D_h \times 1}{b}\right) = \underset{D_h \times 1}{h_t}$$

d : t time-step의 단어의 차원
D_h : hidden size의 크기

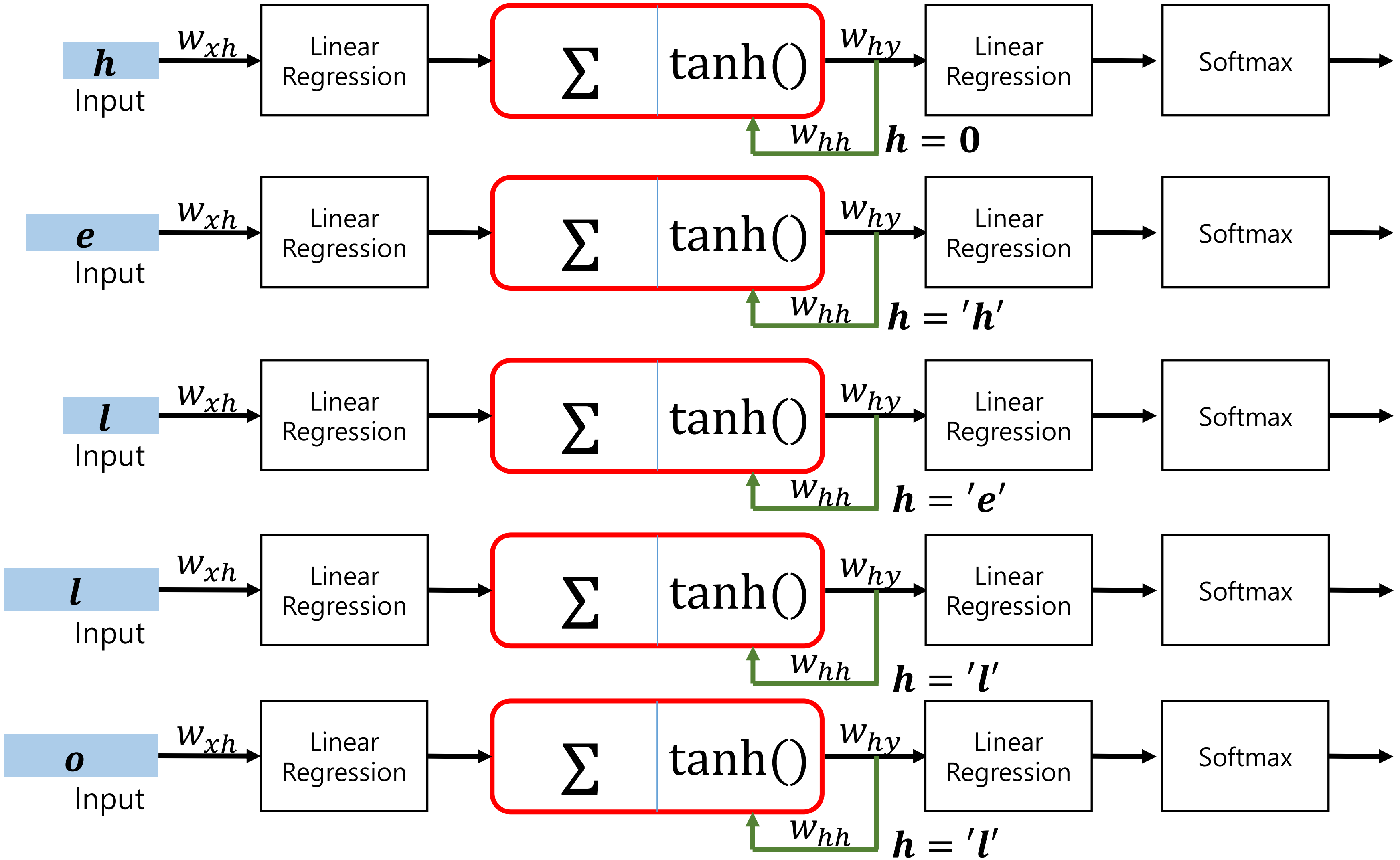
그림 출처 : https://wikidocs.net/book/2155

CONTENTS

RNN 동작원리

다음 charter 예측 모델

- h가 입력일때 → e를 예측
- e가 입력일때 → l을 예측



h	0.01
e	0.97
l	0.01
o	0.01

h	0.02
e	0.21
l	0.75
o	0.02

h	0.01
e	0.01
l	0.97
o	0.01

h	0.01
e	0.21
l	0.03
o	0.75

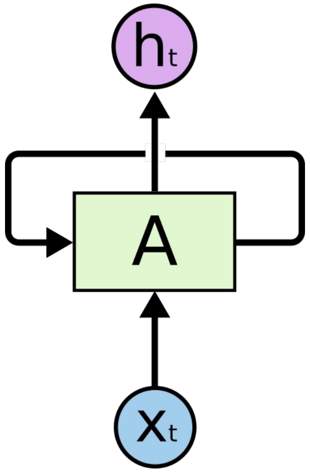
h	0.25
e	0.25
l	0.25
o	0.25

RNN 구현 (With RNN class)

RNN 구현의 기본 형태

```
rnn = torch.nn.RNN(input_size, hidden_size)

outputs, _status = rnn(input_data)
```

outputs
shape=(-,-,-)



input_data
shape=(-,-,-)

예제 참조: 모두의딥러닝 시즌2
http://colah.github.io/posts/2015-08-Understanding-LSTMs

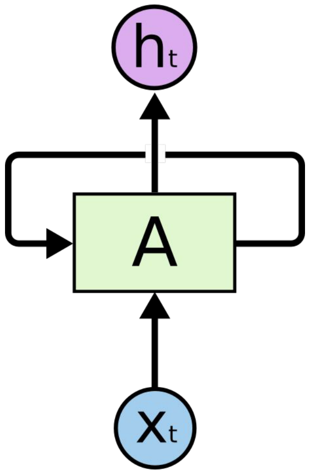
RNN의 Input size 및 형태 (one-hot)

"hello"

```
# 1-hot encoding
h = [1, 0, 0, 0]
e = [0, 1, 0, 0]
l = [0, 0, 1, 0]
o = [0, 0, 0, 1]
```

```
input_size = 4
```

outputs   s
hape=(-,-,-)



input_data
shape=(-,-,4)

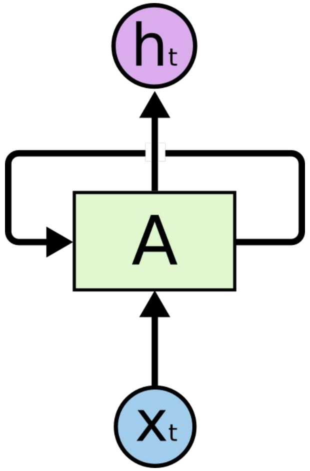
RNN 구현 (With RNN class)

RNN의 hidden state

outputs s
hape=(-,-,2)

desirable output size

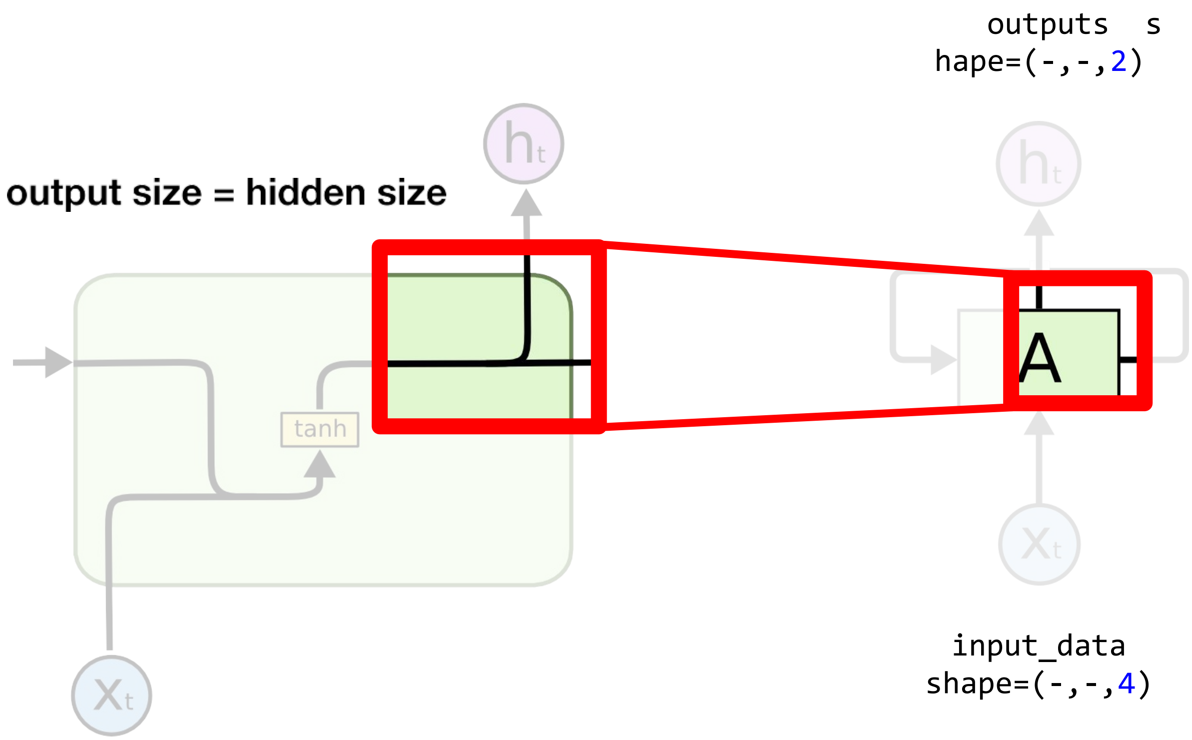
hidden_size = 2



input_data
shape=(-,-,4)

RNN 구현 (With RNN class)

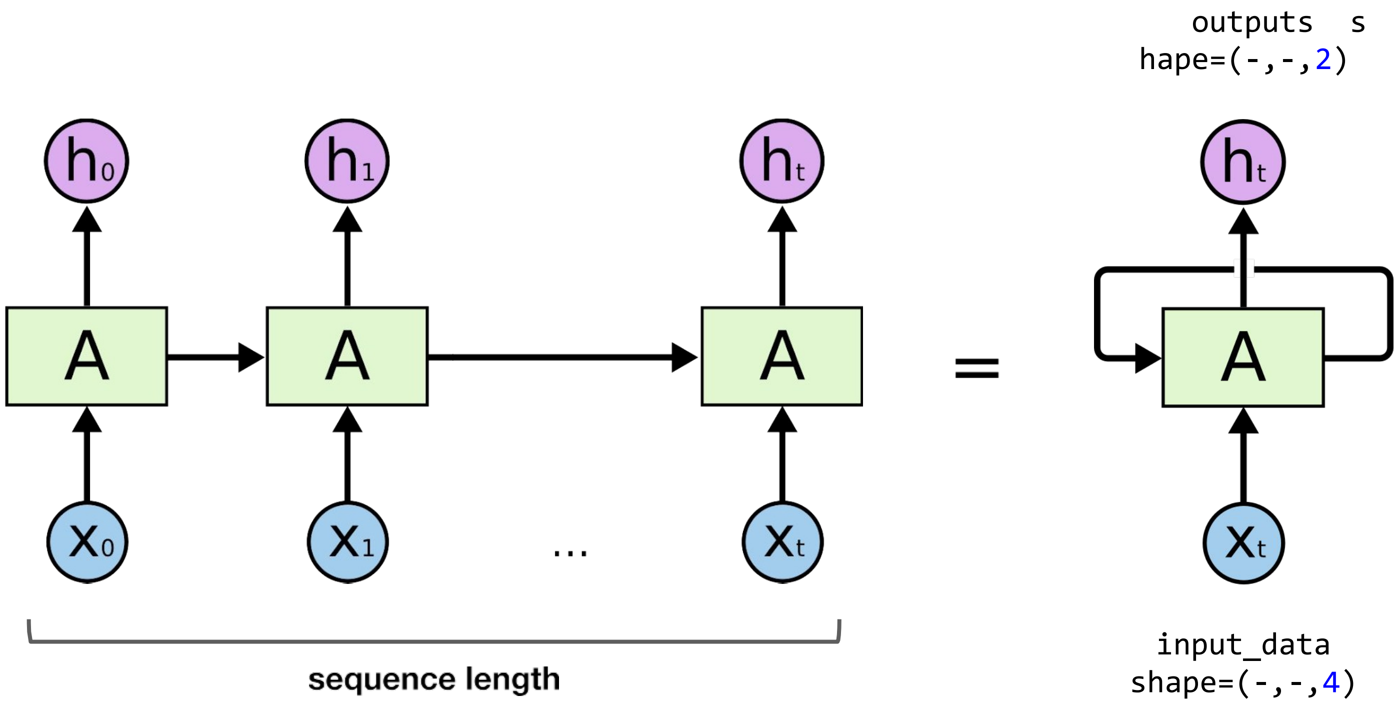
RNN의 출력의 크기 (output size)



http://colah.github.io/posts/2015-08-Understanding-LSTMs

RNN 구현 (With RNN class)

RNN의 입력길이 (sequence length)



http://colah.github.io/posts/2015-08-Understanding-LSTMs

RNN 구현 (With RNN class)

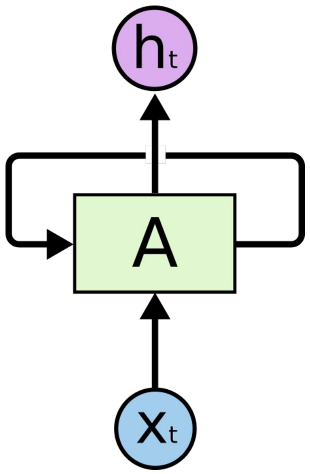
RNN의 입력길이 (sequence length)

h, e, l, l, o

$$
\begin{aligned}
x_0 &= [1, 0, 0, 0] \\
x_1 &= [0, 1, 0, 0] \\
x_2 &= [0, 0, 1, 0] \\
x_3 &= [0, 0, 1, 0] \\
x_4 &= [0, 0, 0, 1]
\end{aligned}
$$

Automatically Calculated

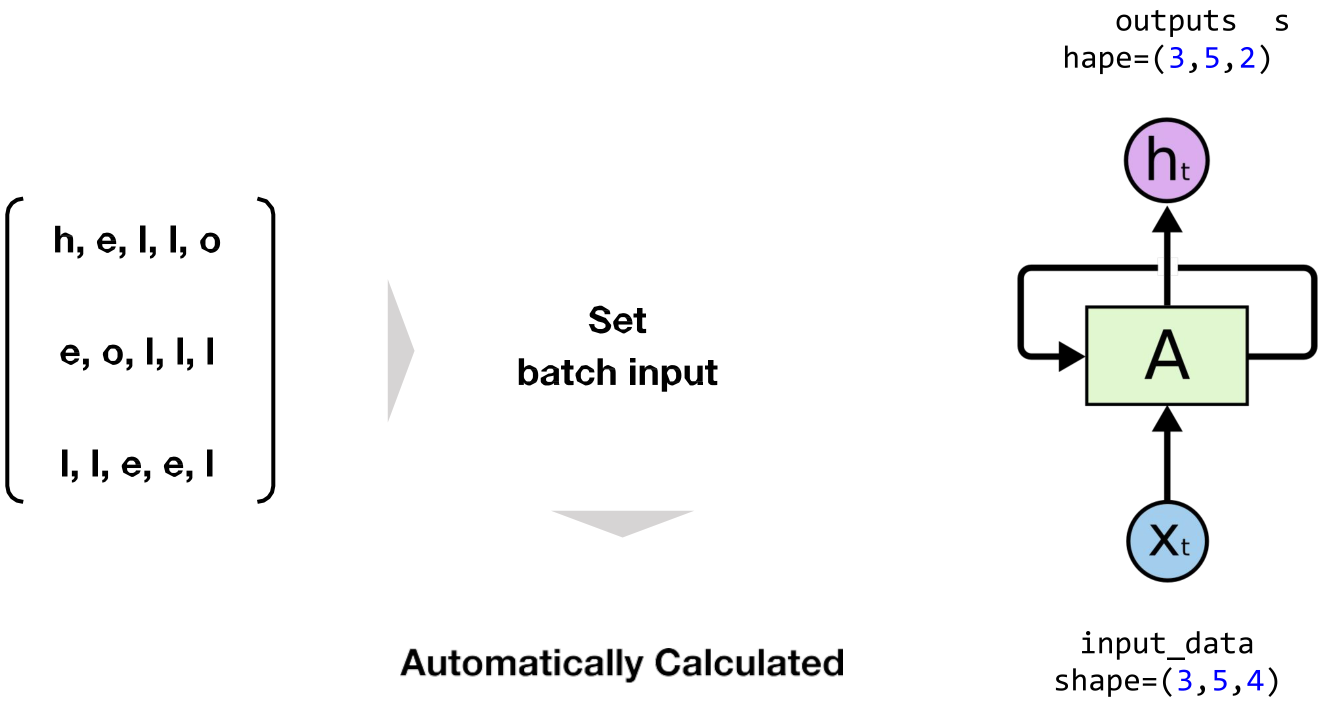
outputs s
hape=(-,5,2)



input_data
shape=(-,5,4)

RNN 구현 (With RNN class)

RNN의 한번에 처리할 샘플의 수 (batch size)



RNN 구현 (With RNN class)

Character Sequence Prediction 예제

```
import torch
import numpy as np

input_size = 4
hidden_size = 2

# 1-hot encoding

h = [1, 0, 0, 0]
e = [0, 1, 0, 0]
l = [0, 0, 1, 0]
o = [0, 0, 0, 1]
```

```
input_data_np = np.array([[h, e, l, l, o],
                          [e, o, l, l, l],
                [l, l, e, e, l]], dtype=np.float32)

# transform as torch tensor
input_data = torch.Tensor(input_data_np)

rnn = torch.nn.RNN(input_size, hidden_size)  out

puts, _status = rnn(input_data)
```

RNN 구현 (With RNN class)

'Hihello' example

- 'Hihello' problem
- Data setting
  - One hot encoding
- Cross entropy loss
- Code run through

'hihello' problem

- 'h', 'i', 'h', 'e', 'l', 'l', 'o'
- We will predict the next character!
- How can we represent characters in PyTorch?

How can we represent characters?

- We can represent them by index
  - 'h' -> 0
  - 'i' -> 1
  - 'e' -> 2
  - 'l' -> 3
  - 'o' -> 4

```
# list of available characters  char
_set = ['h', 'i', 'e', 'l', 'o']
```

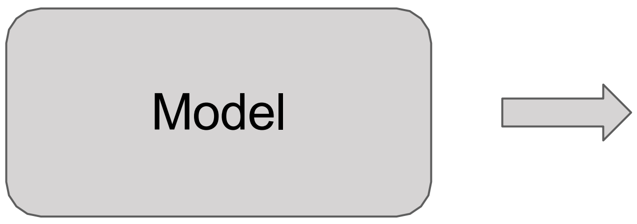
One-hot encoding

- We need to encode using one-hot encoding!

```
# list of available characters
char_set = ['h', 'i', 'e', 'l', 'o']
x_data = [[0, 1, 0, 2, 3, 3]]
x_one_hot = [[[1, 0, 0, 0, 0],
              [0, 1, 0, 0, 0],
              [1, 0, 0, 0, 0],
              [0, 0, 1, 0, 0],
              [0, 0, 0, 1, 0],
              [0, 0, 0, 1, 0]]]
y_data = [[1, 0, 2, 3, 3, 4]]
```

Cross Entropy Loss

- Loss for categorical output (usually interpreted as probability)



output	label
0.1	0
0.2	0
0.3	0
0.4	1

```python
# loss & optimizer setting
criterion = torch.nn.CrossEntropyLoss()
…
loss = criterion(outputs.view(-1, input_size), Y.view(-1))
```

RNN 구현 (With RNN class)

Code run through (hihello)

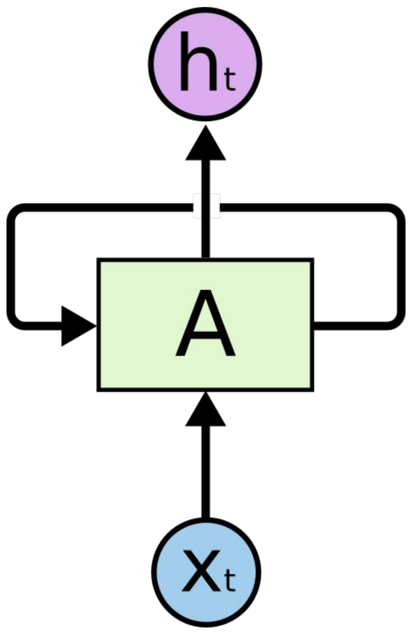
```
char_set = ['h', 'i', 'e', 'l', 'o']
# hyper parameters
input_size = len(char_set)
hidden_size = len(char_set)
learning_rate = 0.1
# data setting
x_data = [[0, 1, 0, 2, 3, 3]]
x_one_hot = [[[1, 0, 0, 0, 0],
              [0, 1, 0, 0, 0],
              [1, 0, 0, 0, 0],
              [0, 0, 1, 0, 0],
              [0, 0, 0, 1, 0],
              [0, 0, 0, 1, 0]]]
y_data = [[1, 0, 2, 3, 3, 4]]
```

```
# transform as torch tensor variable
X = torch.FloatTensor(x_one_hot)
Y = torch.LongTensor(y_data)
```

RNN 구현 (With RNN class)

Code run through

```
# declare RNN
rnn = torch.nn.RNN(input_size, hidden_size, batch_first=True)  # batch_first guarantees the order of output = (B, S, F)

# loss & optimizer setting
criterion = torch.nn.CrossEntropyLoss()
optimizer = optim.Adam(rnn.parameters(), learning_rate)

# start training
for i in range(100):  optimi
   zer.zero_grad()  outputs,
   _status = rnn(X)
   loss = criterion(outputs.view(-1, input_size), Y.view(-1))
   loss.backward()
   optimizer.step()
   result = outputs.data.numpy().argmax(axis=2)
   result_str = ''.join([char_set[c] for c in np.squeeze(result)])
   print(i, "loss: ", loss.item(), "prediction: ", result, "true Y: ", y_data, "prediction str: ", result_str)
```



Code run through (charseq)

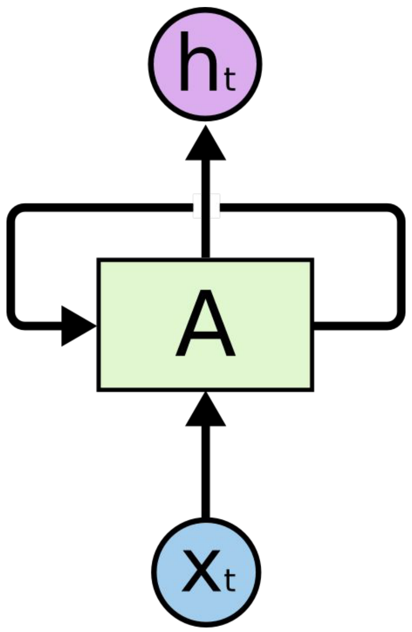
```
sample = " if you want you"
# make dictionary
char_set = list(set(sample))
char_dic = {c: i for i, c in enumerate(char_set)}

# hyper parameters  dic_
size = len(char_dic)
hidden_size = len(char_dic)
learning_rate = 0.1

# data setting
sample_idx = [char_dic[c] for c in sample]
x_data = [sample_idx[:-1]]
x_one_hot = [np.eye(dic_size)[x] for x in x_data]
y_data = [sample_idx[1:]]
```

```
# transform as torch tensor variable
X = torch.FloatTensor(x_one_hot)
Y = torch.LongTensor(y_data)
```

RNN 구현 (With RNN class)

Code run through

```
# declare RNN
rnn = torch.nn.RNN(input_size, hidden_size, batch_first=True)

# loss & optimizer setting
criterion = torch.nn.CrossEntropyLoss()
optimizer = optim.Adam(rnn.parameters(), learning_rate)

# start training
for i in range(100):  optimi
   zer.zero_grad()  outputs,
   _status = rnn(X)
   loss = criterion(outputs.view(-1, input_size), Y.view(-1))
   loss.backward()
   optimizer.step()
   result = outputs.data.numpy().argmax(axis=2)
   result_str = ''.join([char_set[c] for c in np.squeeze(result)])
   print(i, "loss: ", loss.item(), "prediction: ", result, "true Y: ", y_data, "prediction str: ", result_str)
```



longseq

- We want to use longer dataset
- But we want to train in bigger chunks
- How can we create fixed size sequence dataset from long sentence?

RNN 구현 (With RNN class)

Making sequence dataset from long sentence

```
sentence = ("if you want to build a ship, don't drum up people together to "
            "collect wood and don't assign them tasks and work, but rather "
            "teach them to long for the endless immensity of the sea.")
```

```
"if you wan" -> "f you want"

"f you want" -> " you want "

" you want " -> "you want t"

"you want t" -> "ou want to"

"ou want to" -> "u want to "
```

...

RNN 구현 (With RNN class)

Making sequence dataset from long sentence (code)

```
# data setting
x_data = []  y
_data = []

for i in range(0, len(sentence) - sequence_length):
   x_str = sentence[i:i + sequence_length]
   y_str = sentence[i + 1: i + sequence_length + 1]
   print(i, x_str, '->', y_str)
   x_data.append([char_dic[c] for c in x_str])  # x str to index
   y_data.append([char_dic[c] for c in y_str])  # y str to index

x_one_hot = [np.eye(dic_size)[x] for x in x_data]

# transform as torch tensor variable
X = torch.FloatTensor(x_one_hot)
Y = torch.LongTensor(y_data)
```

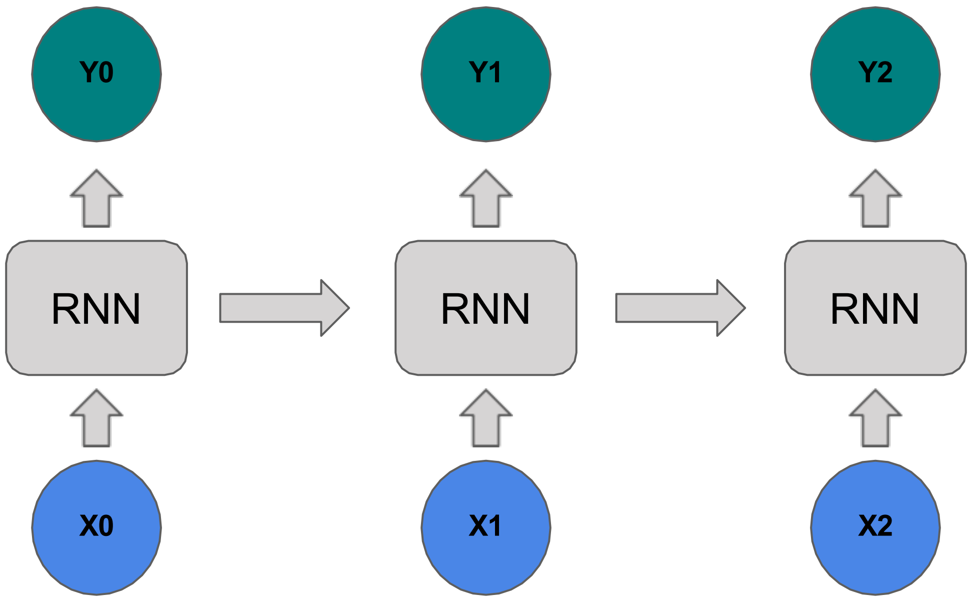
```
"if you wan" -> "f you want"

"f you want" -> " you want "

" you want " -> "you want t"

"you want t" -> "ou want to"

"ou want to" -> "u want to "
```

...

RNN 구현 (With RNN class)

Adding FC layer and stacking RNN

```
# declare RNN + FC
class Net(torch.nn.Module):
    def__init_(self, input_dim, hidden_dim, layers):
        super(Net, self).__init_()
        self.rnn = torch.nn.RNN(input_dim, hidden_dim, num_layers=layers,
batch_first=True)
        self.fc = torch.nn.Linear(hidden_dim, hidden_dim, bias=True)

    def forward(self, x):
        x, _status = self.rnn(x)
        x = self.fc(x)
        return x

net = Net(dic_size, hidden_size, 2)
```

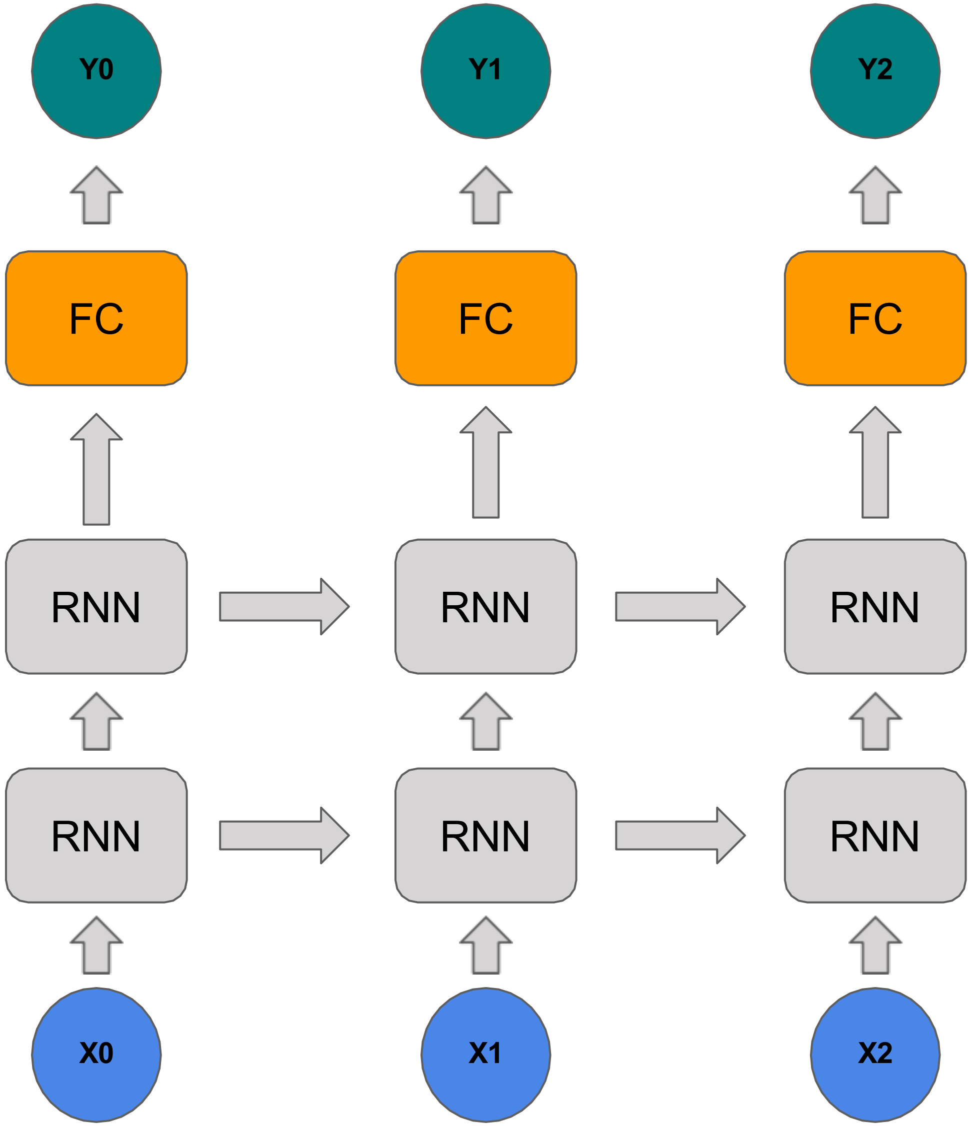


Vanilla RNN

RNN 구현 (With RNN class)

Adding FC layer and stacking RNN

```
# declare RNN + FC
class Net(torch.nn.Module):
    def__init_(self, input_dim, hidden_dim, layers):
        super(Net, self).__init_()
        self.rnn = torch.nn.RNN(input_dim, hidden_dim, num_layers=layers,
batch_first=True)
        self.fc = torch.nn.Linear(hidden_dim, hidden_dim, bias=True)


    def forward(self, x):
        x, _status = self.rnn(x)
        x = self.fc(x)
        return x


net = Net(dic_size, hidden_size, 2)
```



RNN 구현 (With RNN class)

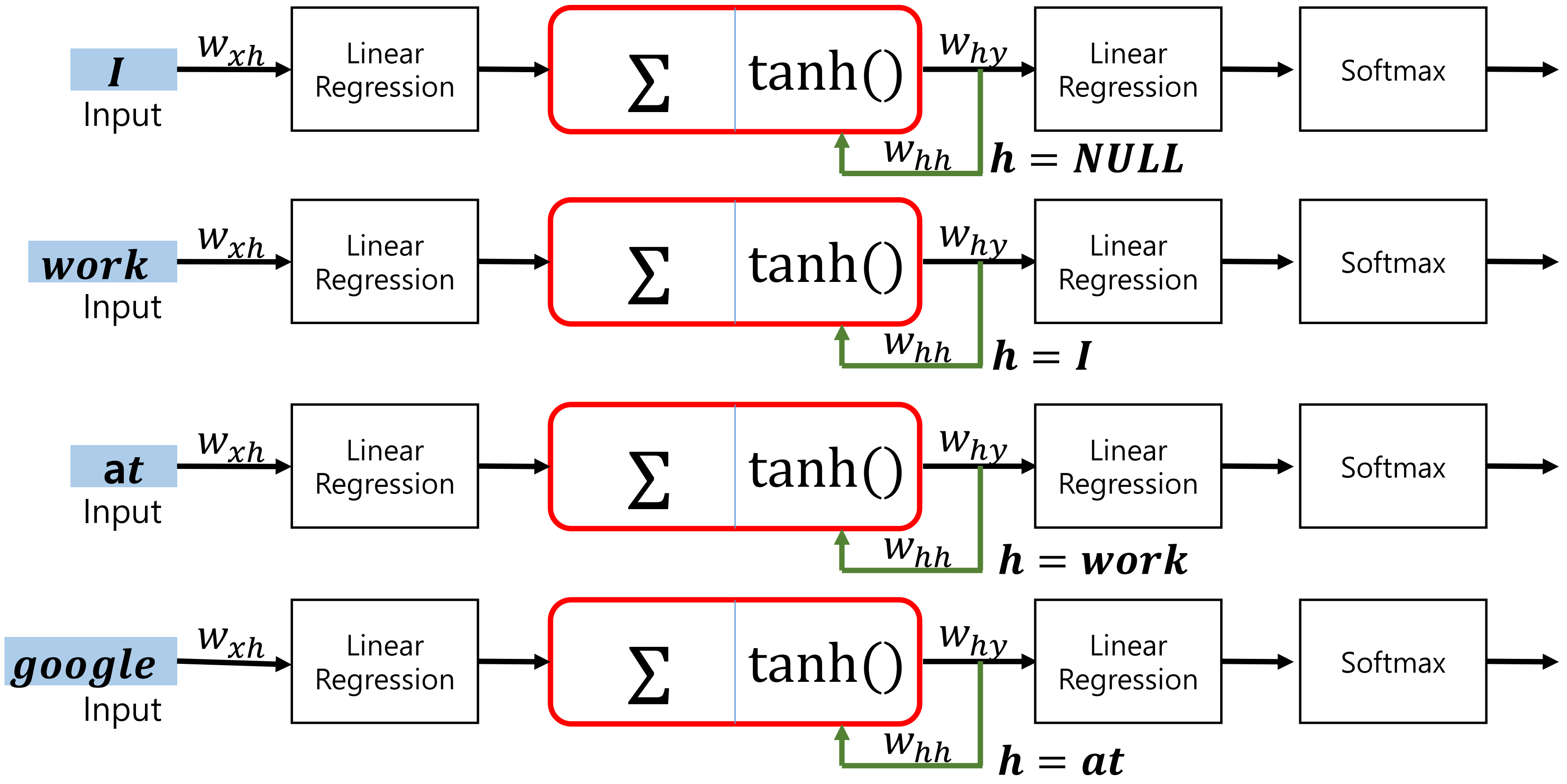
Code run through

```
# loss & optimizer setting
criterion = torch.nn.CrossEntropyLoss()
optimizer = optim.Adam(net.parameters(), learning_rate)

# start training
for i in range(100):  op
    timizer.zero_grad()
    outputs = net(X)
    loss = criterion(outputs.view(-1, dic_size), Y.view(-1))
    loss.backward()
    optimizer.step()

    results = outputs.argmax(dim=2)
    predict_str = ""
    for j, result in enumerate(results):
        print(i, j, ''.join([char_set[t] for t in result]), loss.item())
        if j == 0:
            predict_str += ''.join([char_set[t] for t in result])
        else:
            predict_str += char_set[result[-1]]
```

RNN 활용

자연어처리: 형태소 분석기 만들기

- I work at google → 나는 구글에 근무한다.
- I google at work → 나는 일하면서 구글링한다.



대명사	0.97
명사	0.01
동사	0.01
전치사	0.01

대명사	0.02
명사	0.21
동사	0.75
전치사	0.02

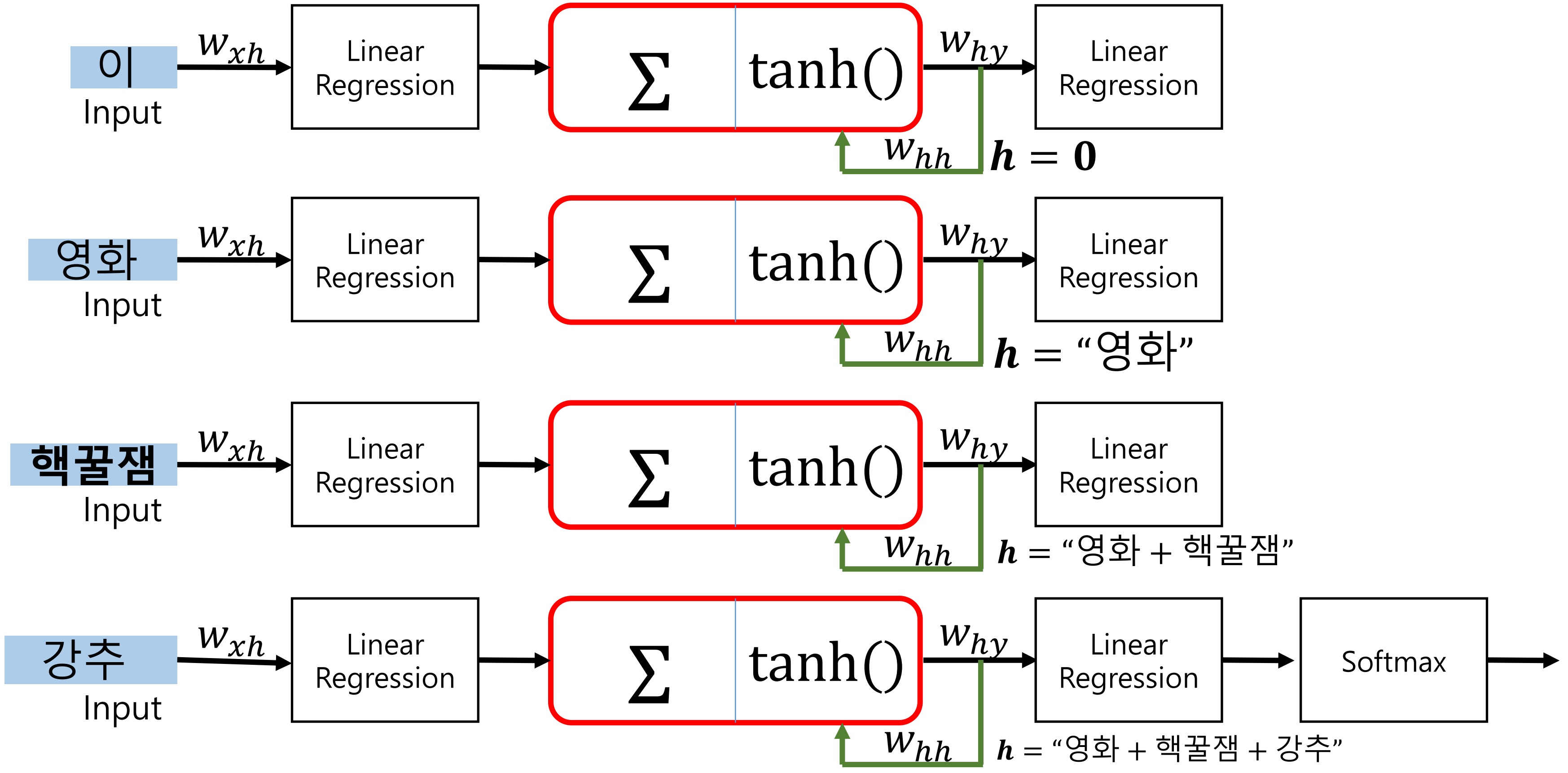
대명사	0.01
명사	0.01
동사	0.01
전치사	0.97

대명사	0.02
명사	0.75
동사	0.21
전치사	0.02

RNN 활용

자연어처리: 감정분석기

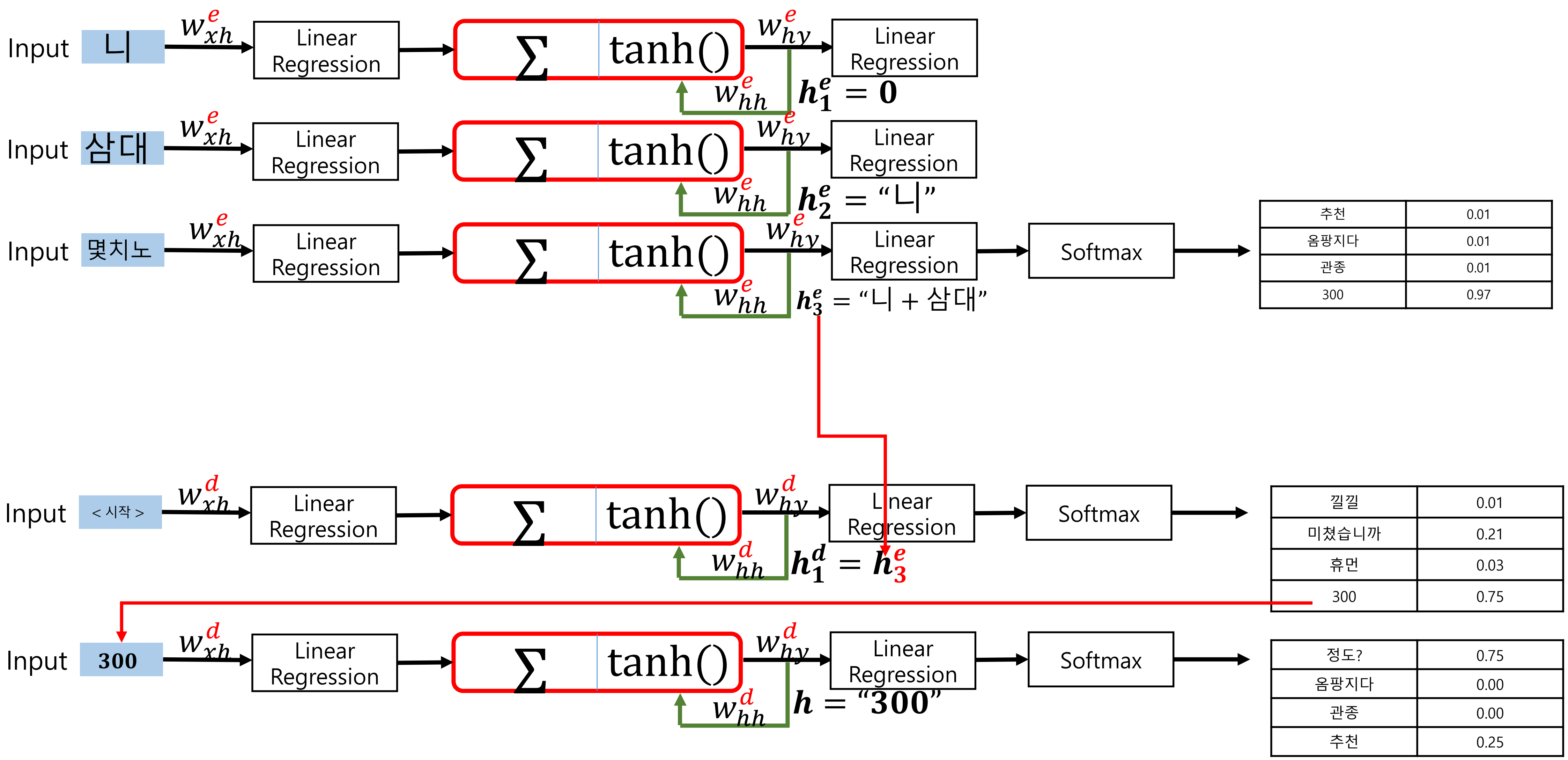
- 영화 댓글이 긍정일까 부정일까?



부정	0.04
긍정	0.75
일반	0.21

RNN 활용

자연어처리: 챗봇 (seq2seq)



RNN을 마치며

RNN 특징

- RNN 학습 방법

  ✓ RTRL(Real-time recurrent learning) : 확률적 경사하강법 사용 순환학습
  ✓ BPTT(Backpropagation though time) : 시간 기반 오차역전파

- RNN 장점
  ✓ 이전 정보를 현재의 문제해결에 사용 가능

- RNN 단점
  ✓ Long-Term Dependency : 기울기 소실(Gradient vanishing)로 인해 거리가 먼 과거 상태를 사용한 문맥 처리가 어려움
  → LSTM(Long Short Term Memory), GRU(Gated Recurrent Units)로 해결

감사합니다.