Recurrent Neural Network (RNN) for NLP

jujbob@gmail.com

KyungTae Lim

CONTENTS

CONTENTS

Time Series?

- A time series is a sequence of data points arranged at consistent time intervals. (Wikipedia)
  - Time series data consists of observations ordered chronologically.
  - It is used to predict current trends and future movements based on historical data.

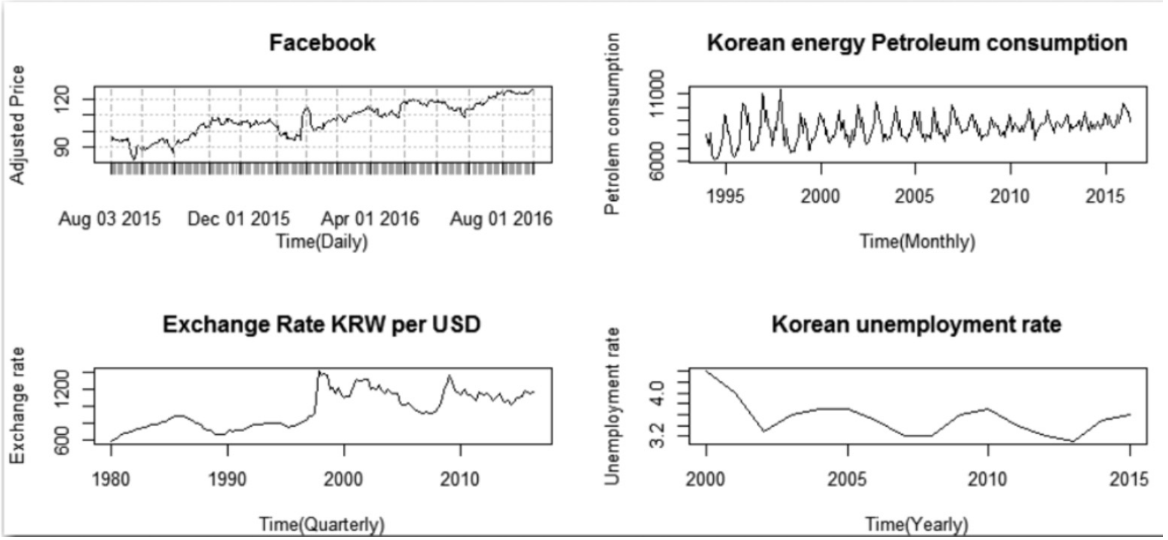


Figure Ref. https://velog.io/@sdubee10/ZeroBase-%EC%8B%9C%EA%B3%84%EC%97%B4%EB%8D%B0%EC%9D%B4%ED%84%B0-Time-Series-Data

Introduction of RNN

Time Series?

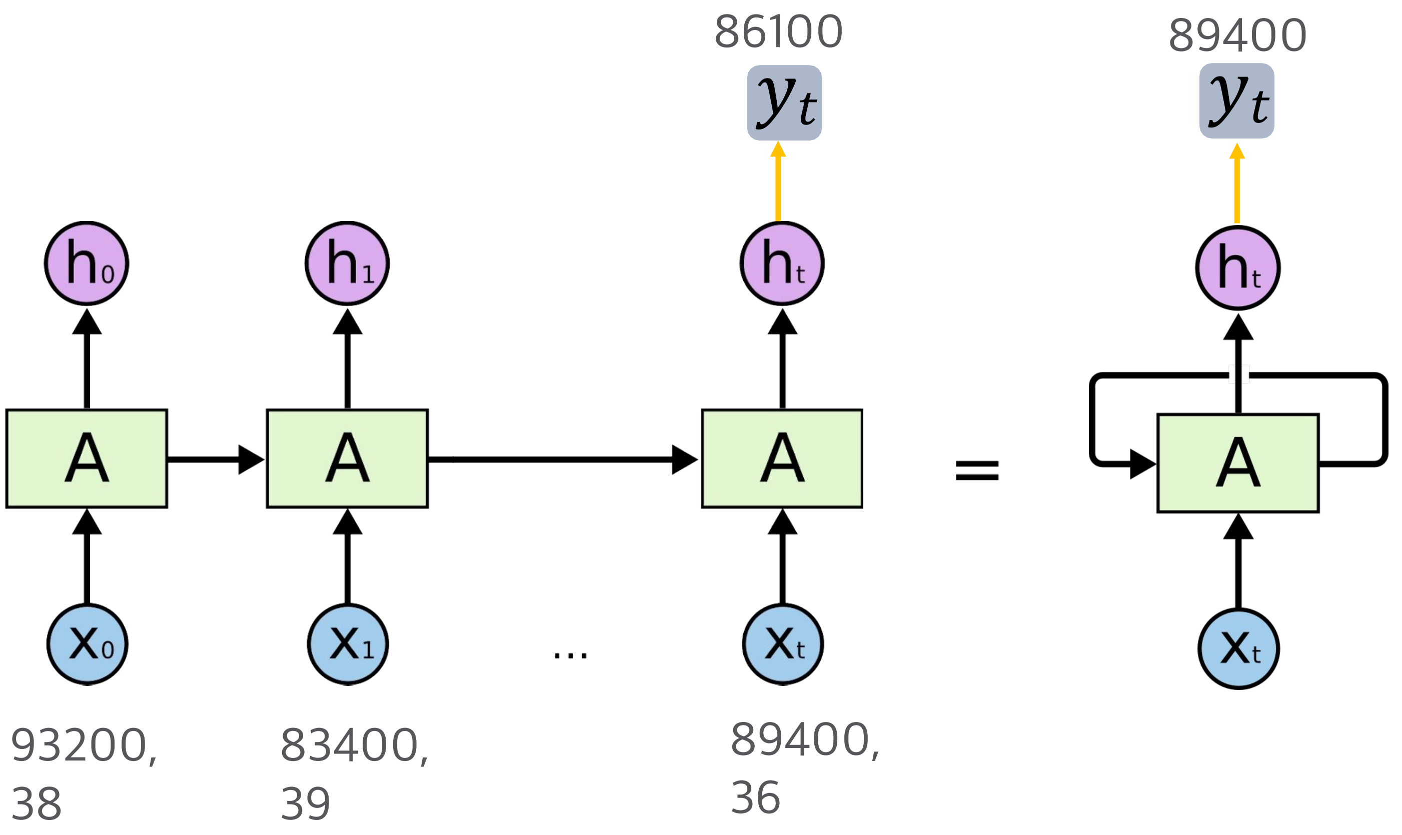
- We aim to predict stock prices based on NCSoft's search frequency and stock price data.
  How might we approach this?
  - (1) Should we input all 2 years of data into an FFN? --> Too much data.
  - (2) Then let's input data for five into an FFN. --> Unable to model sequential relationships
  - (3) Isn't there a method to model considering previous prices over time? → RNN!!!

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

Introduction of RNN

What is RNN(Recurrent Neural Network)?

- RNN (Recurrent Neural Network) is a deep learning model suitable for continuous and sequential data (e.g., time series data, natural language, speech).

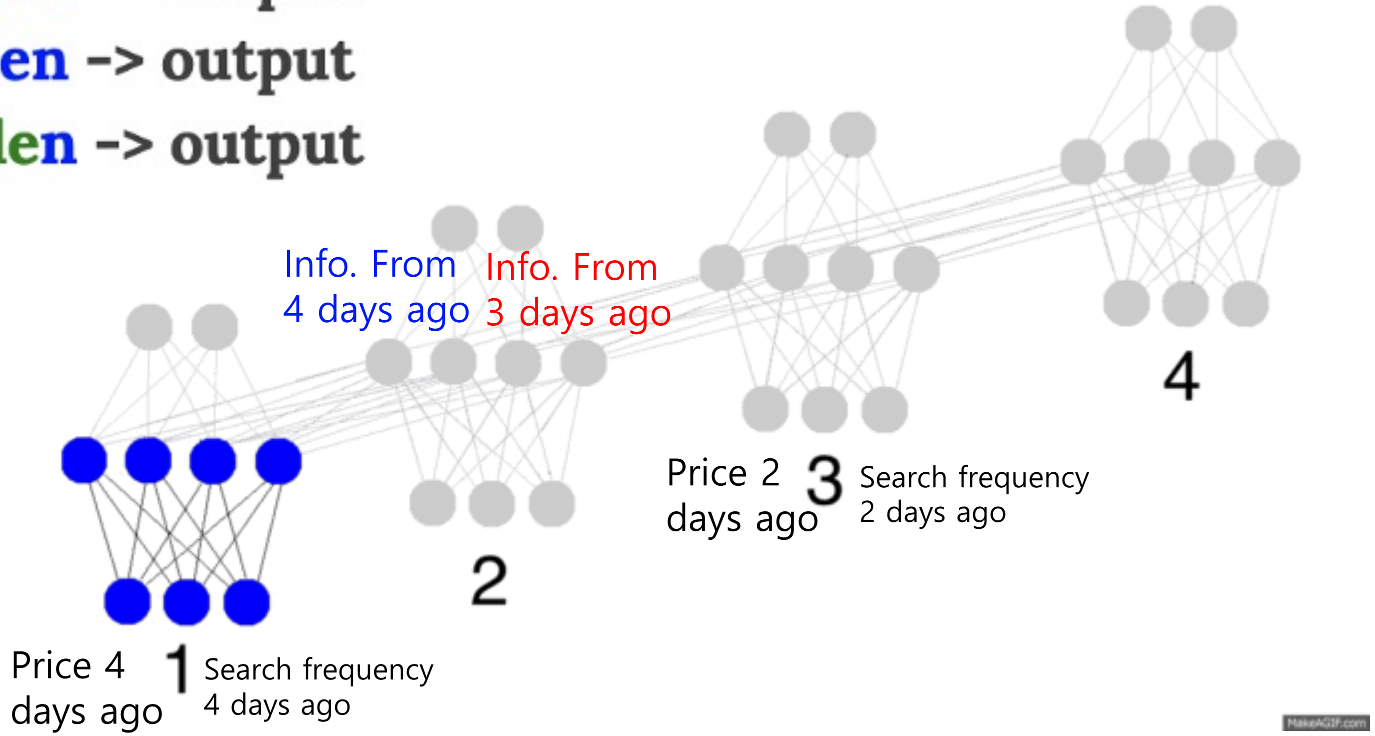


A	B	C
date	num_search	stock_price
2021.3.21	38	932000
2021.3.28	39	834000
2021.4.4	36	885000
2021.4.11	31	906000
2021.4.18	32	894000
2021.4.25	27	861000
2021.5.2	27	820000
2021.5.9	26	850000
2021.5.16	29	823000
2021.5.23	26	856000
2021.5.30	29	854000
2021.6.6	30	858000
2021.6.13	31	848000
2021.6.20	37	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

intuition of RNN

Let's illustrate the transmitted information with colors for each time step.

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

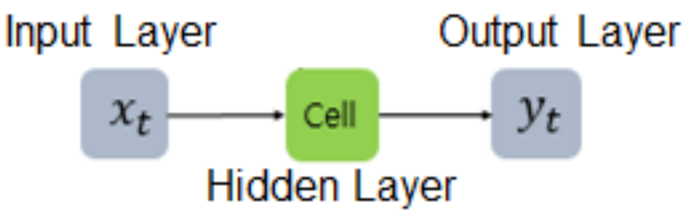


Ref. : https://medium.com/@serbanliviu/the-intuition-behind-recurrent-neural-networks-6fce753fe9f0

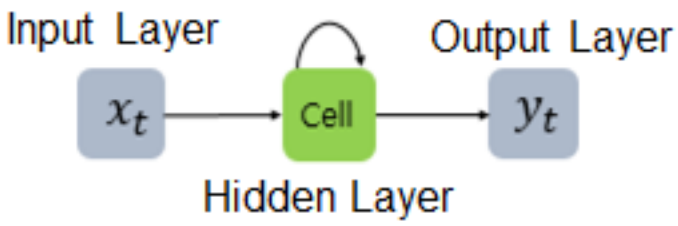
CONTENTS

Comparison of FFN and RNN

Core idea:
Apply the same weights W repeatedly!



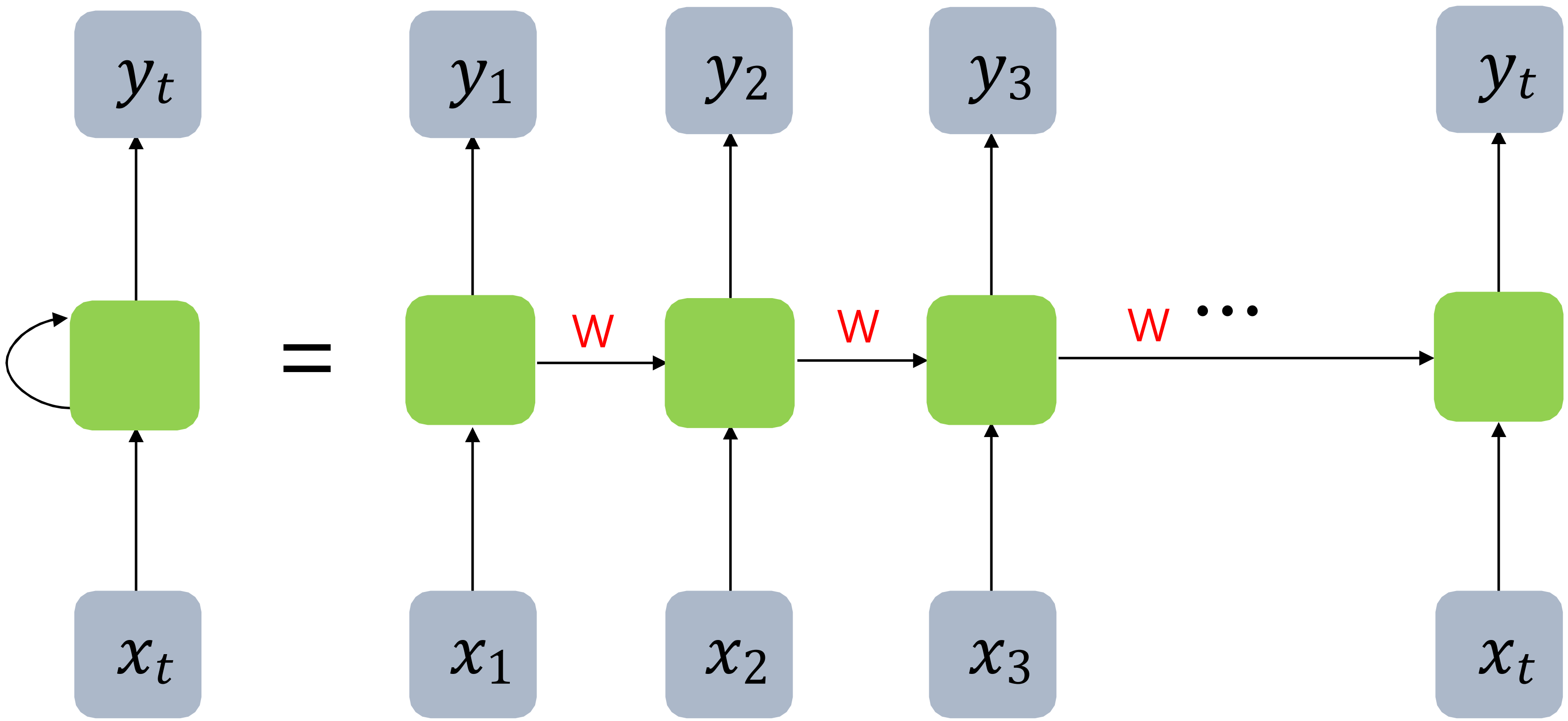
FFN



RNN

Figure Ref. : https://wikidocs.net/book/2155

Visualization of RNN



Core idea:
- Apply the same weights W repeatedly!
- The RNN illustrated below fundamentally assumes input and output as vectors.

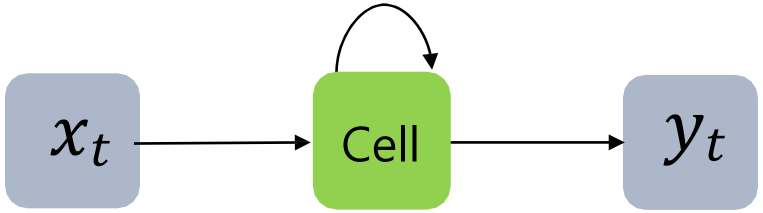
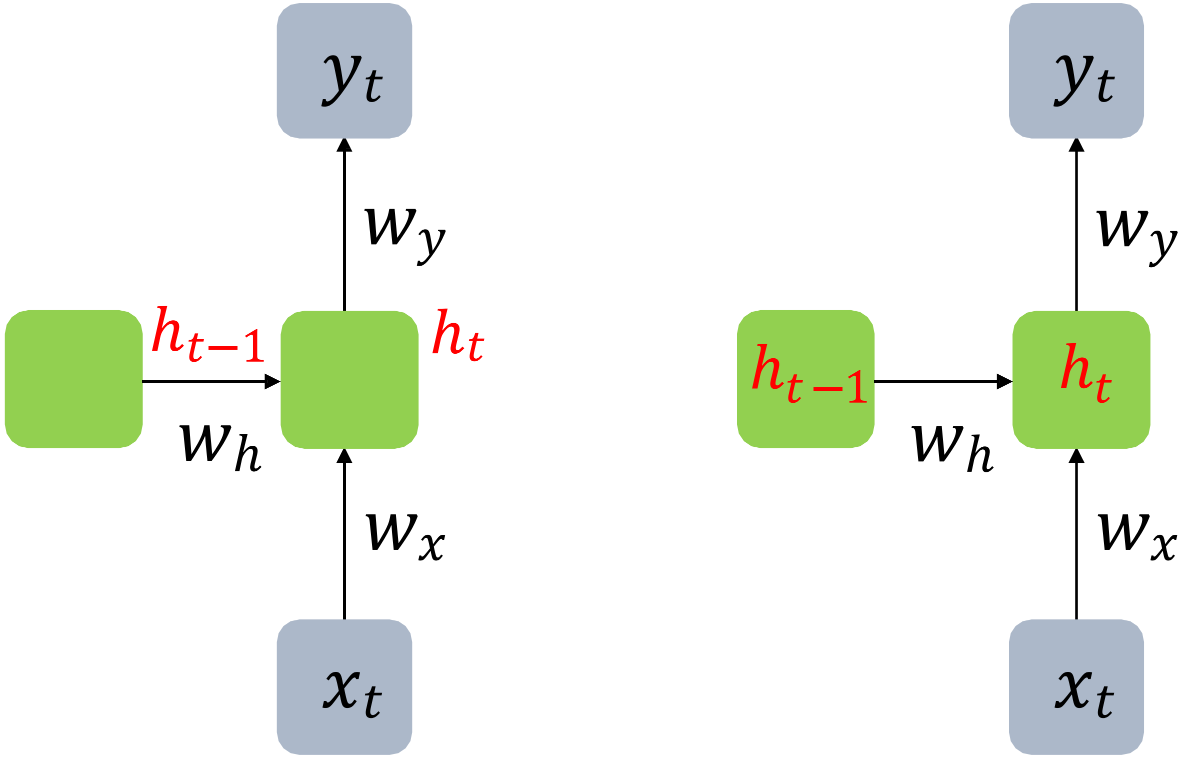


Figure Ref.: https://wikidocs.net/book/2155

Structure of RNN

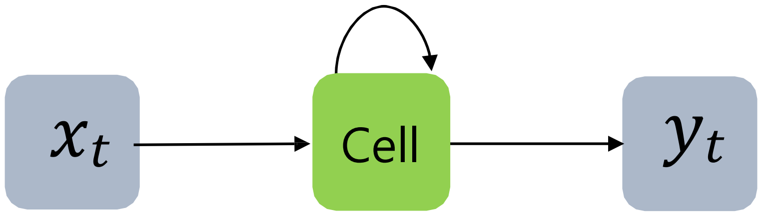
Visualization of RNN

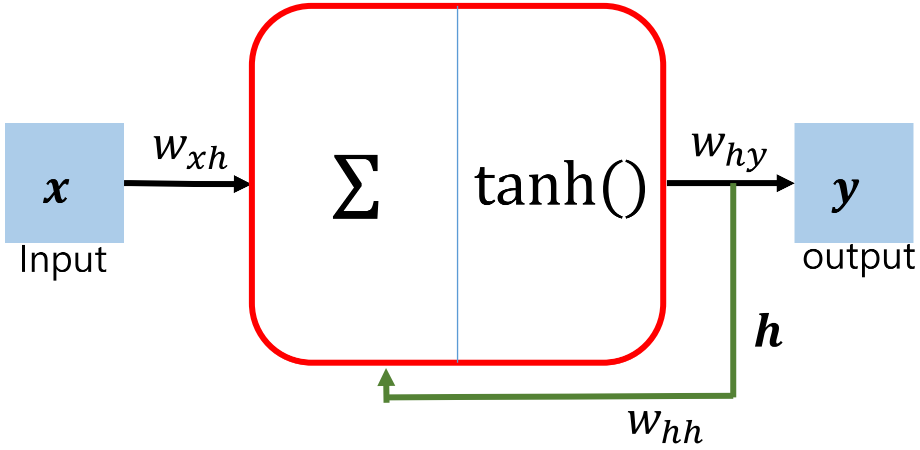


Core idea:
Apply the same weights W repeatedly!

Figure Ref. : https://wikidocs.net/book/2155

Visualization of RNN





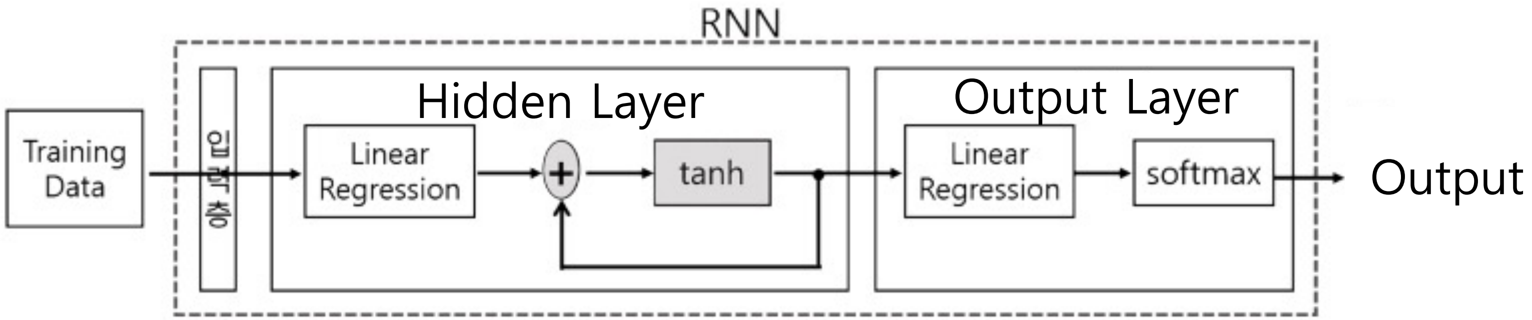
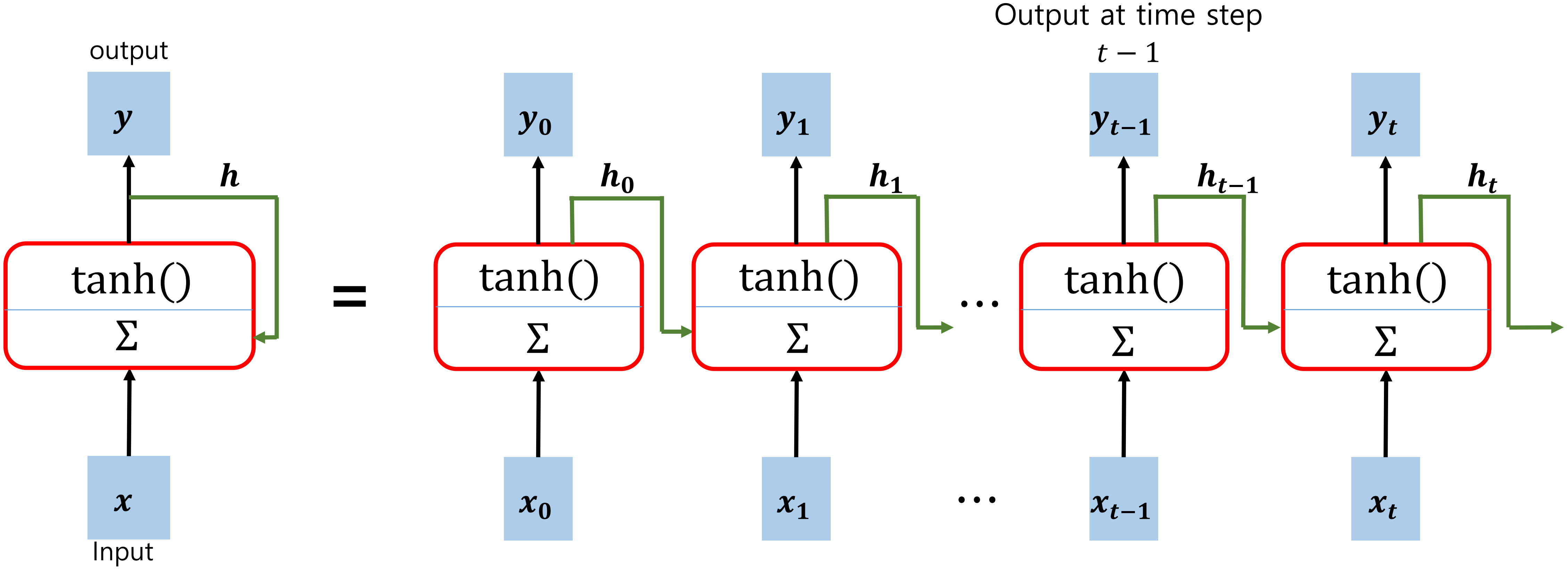


Figure Ref. : 머신러닝을 위한 파이썬 한조각

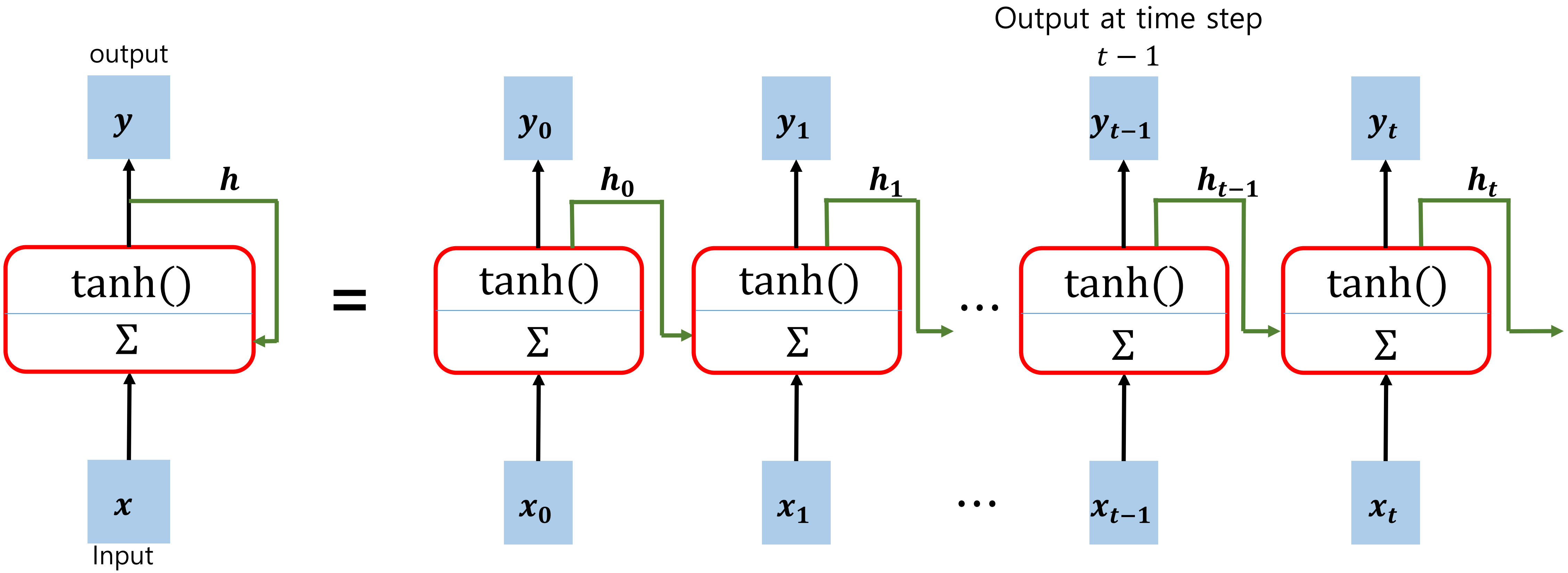
Visualization of RNN



x_t : Input values for all samples
y_t : Output of the recurrent layer for each sample at time step t
h_t : hidden state, defined as $h_t = f(h_{t-1}, x_t)$, meaning the current hidden state is determined by the previous hidden state and the current input.
Hidden state : Information to be passed on to the next time step

Structure of RNN

Visualization of RNN



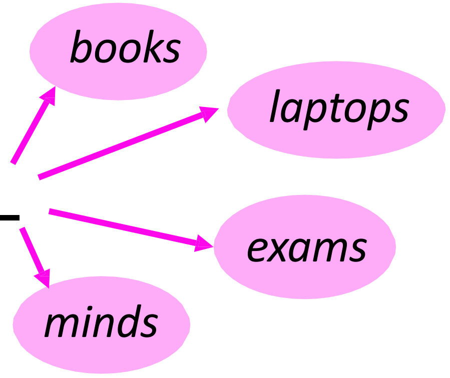
$$h_t = \tanh(X_t \cdot W_{xh} + h_{t-1} \cdot W_{hh} + b)$$
$$y_t = W_{hy} \cdot h_t$$

Structure of RNN

Language Model using RNN

- **Language Modeling** is the task of predicting what word comes next

the students opened their ______



- More formally: given a sequence of words $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(t)}$,
  compute the probability distribution of the next word $\boldsymbol{x}^{(t+1)}$:

$$P(\boldsymbol{x}^{(t+1)} \mid \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(1)})$$

  where $\boldsymbol{x}^{(t+1)}$ can be any word in the vocabulary $V = \{\boldsymbol{w}_1, \ldots, \boldsymbol{w}_{|V|}\}$

- A system that does this is called a **Language Model**

Ref.: Stanford 224n

Language Model using RNN

You can also use a Language Model to generate text

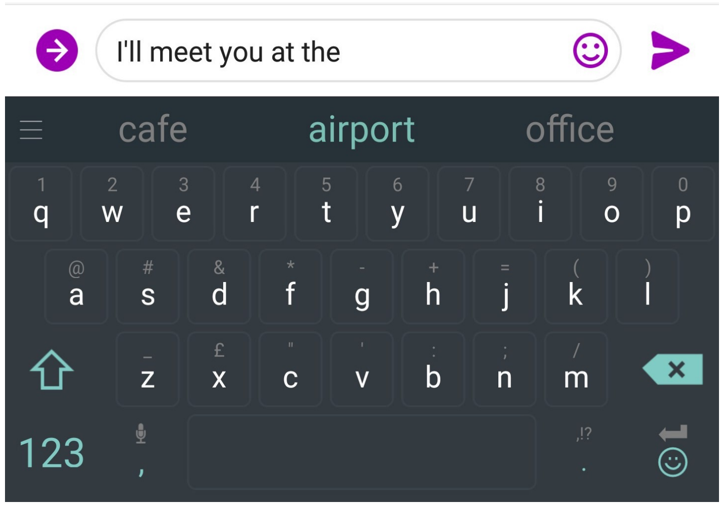
today the ________

condition on this

get probability distribution

company	0.153
bank	0.153
price	0.077
italian	0.039
emirate	0.039
…	

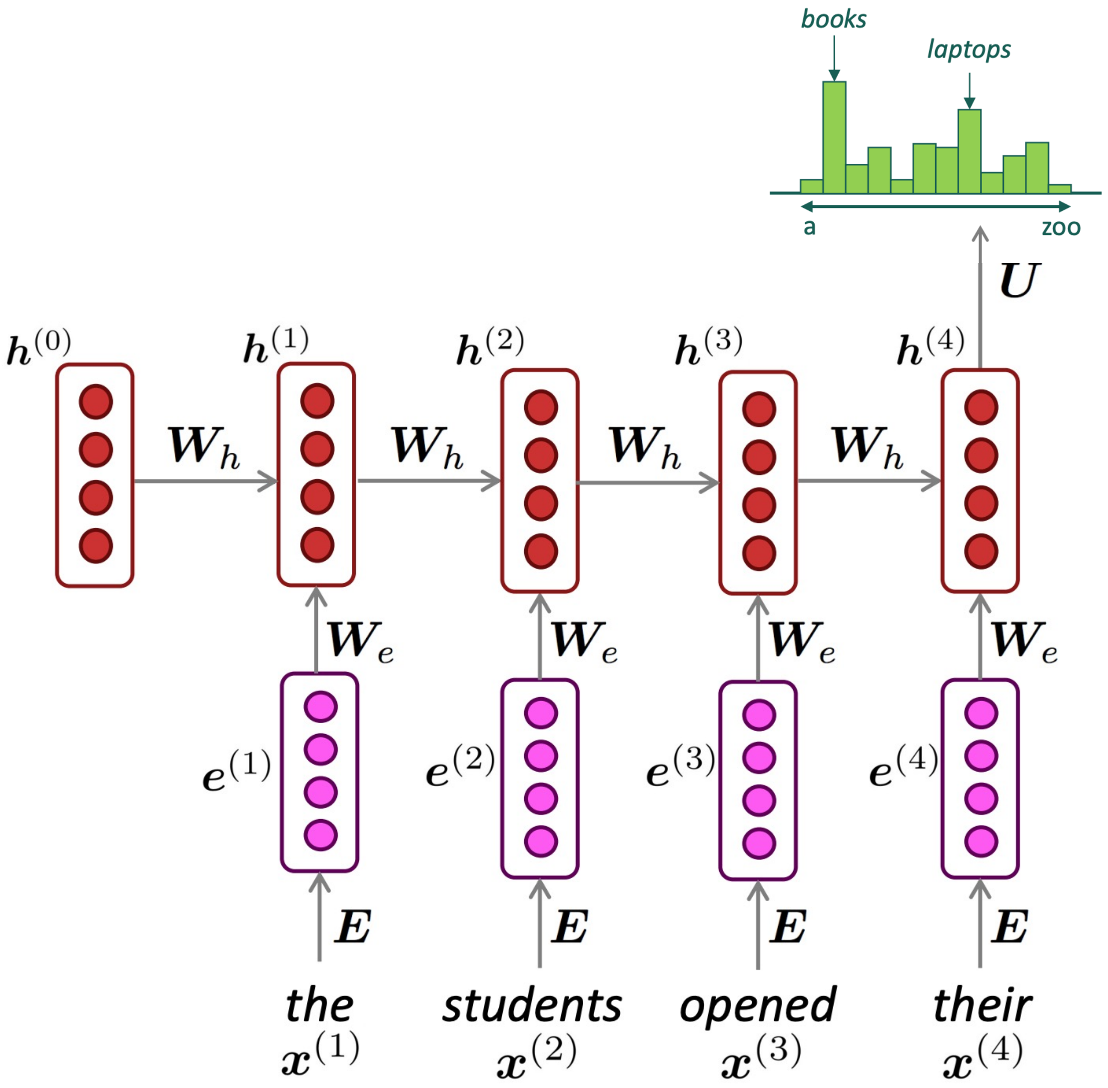
sample



Structure of RNN

Language Model using RNN

$$\hat{\boldsymbol{y}}^{(4)} = P(\boldsymbol{x}^{(5)}|\text{the students opened their})$$



output distribution

$$\hat{\boldsymbol{y}}^{(t)} = \text{softmax}\left(\boldsymbol{U}\boldsymbol{h}^{(t)} + \boldsymbol{b}_2\right) \in \mathbb{R}^{|V|}$$

hidden states

$$\boldsymbol{h}^{(t)} = \sigma\left(\boldsymbol{W}_h\boldsymbol{h}^{(t-1)} + \boldsymbol{W}_e\boldsymbol{e}^{(t)} + \boldsymbol{b}_1\right)$$

$\boldsymbol{h}^{(0)}$ is the initial hidden state

word embeddings

$$\boldsymbol{e}^{(t)} = \boldsymbol{E}\boldsymbol{x}^{(t)}$$

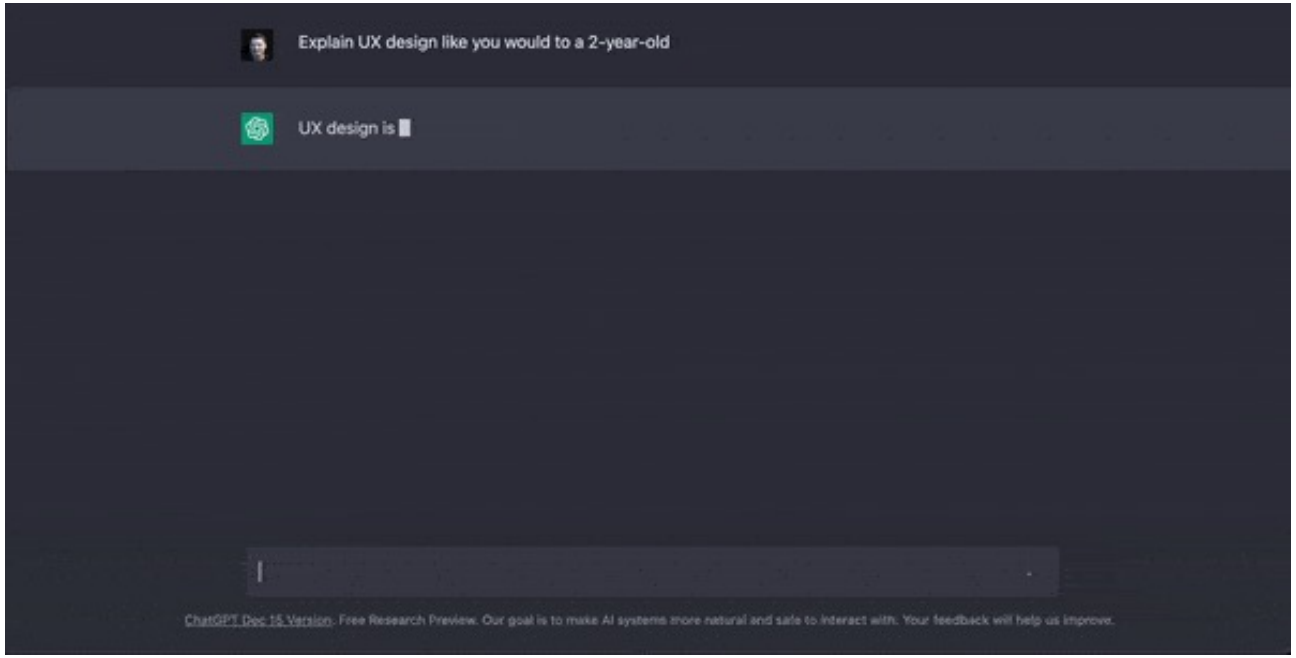
words / one-hot vectors

$$\boldsymbol{x}^{(t)} \in \mathbb{R}^{|V|}$$

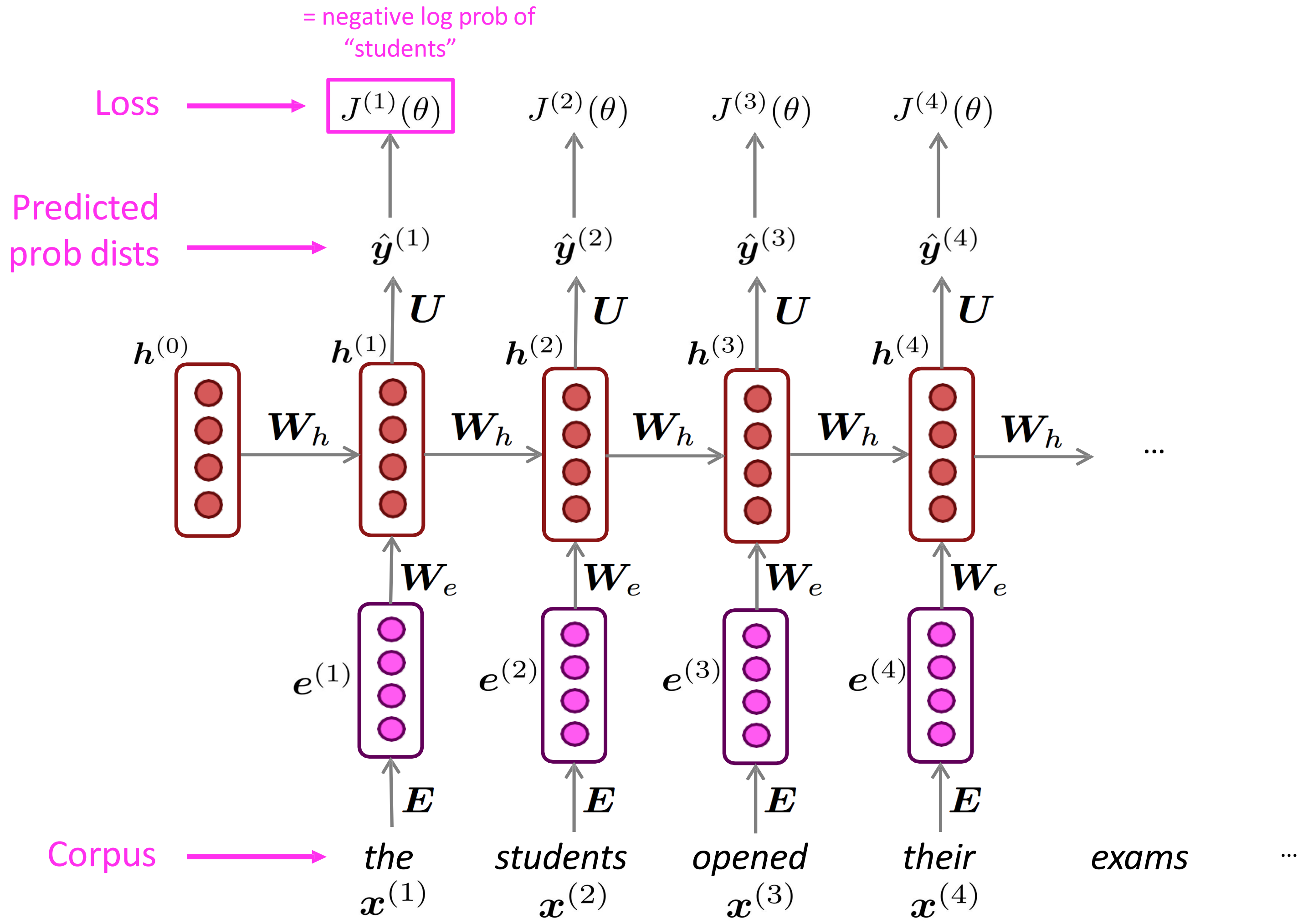
Ref.: Stanford 224n

Language Model using RNN

- GPT??



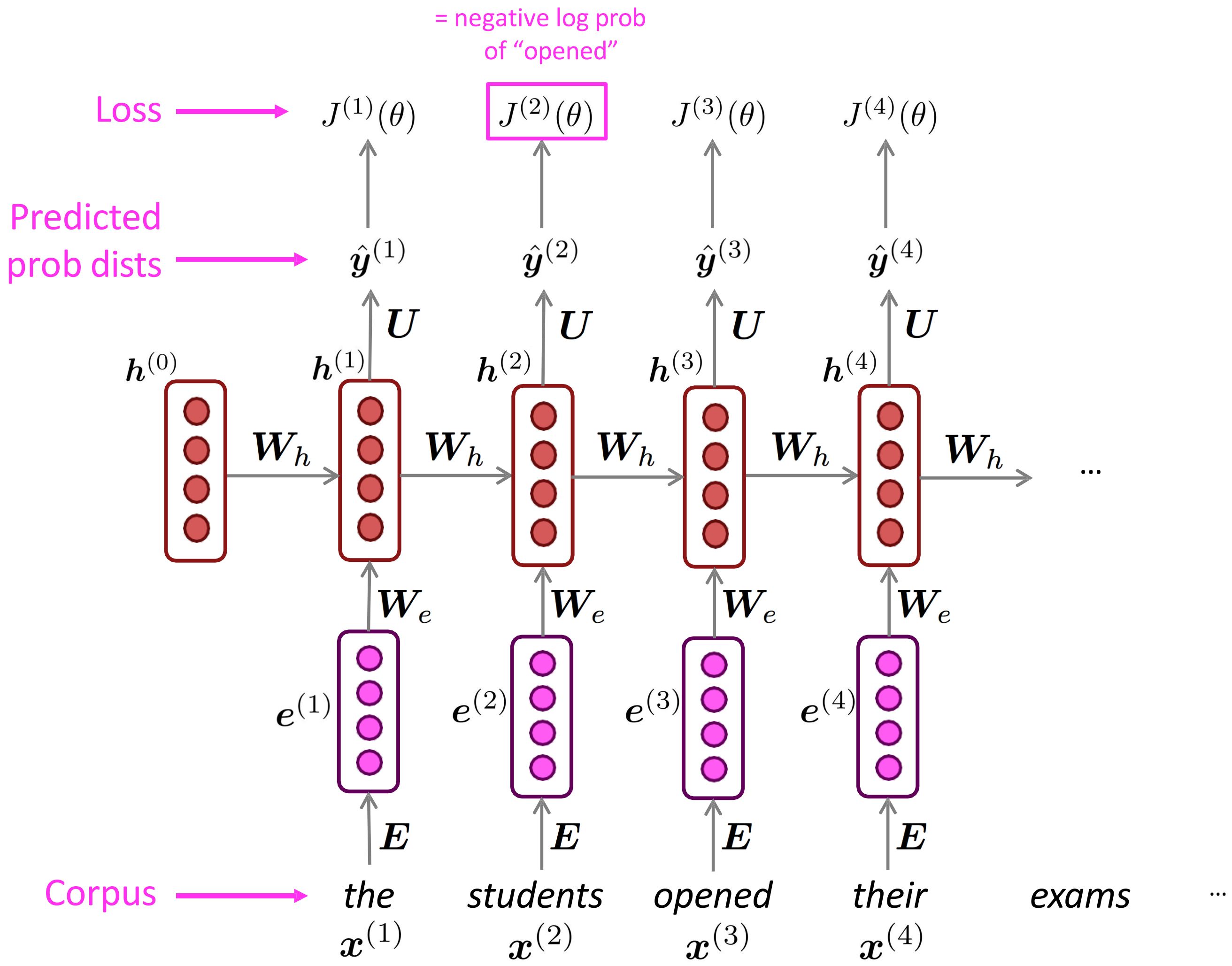
Language Model using RNN



Ref.: Stanford 224n

Structure of RNN

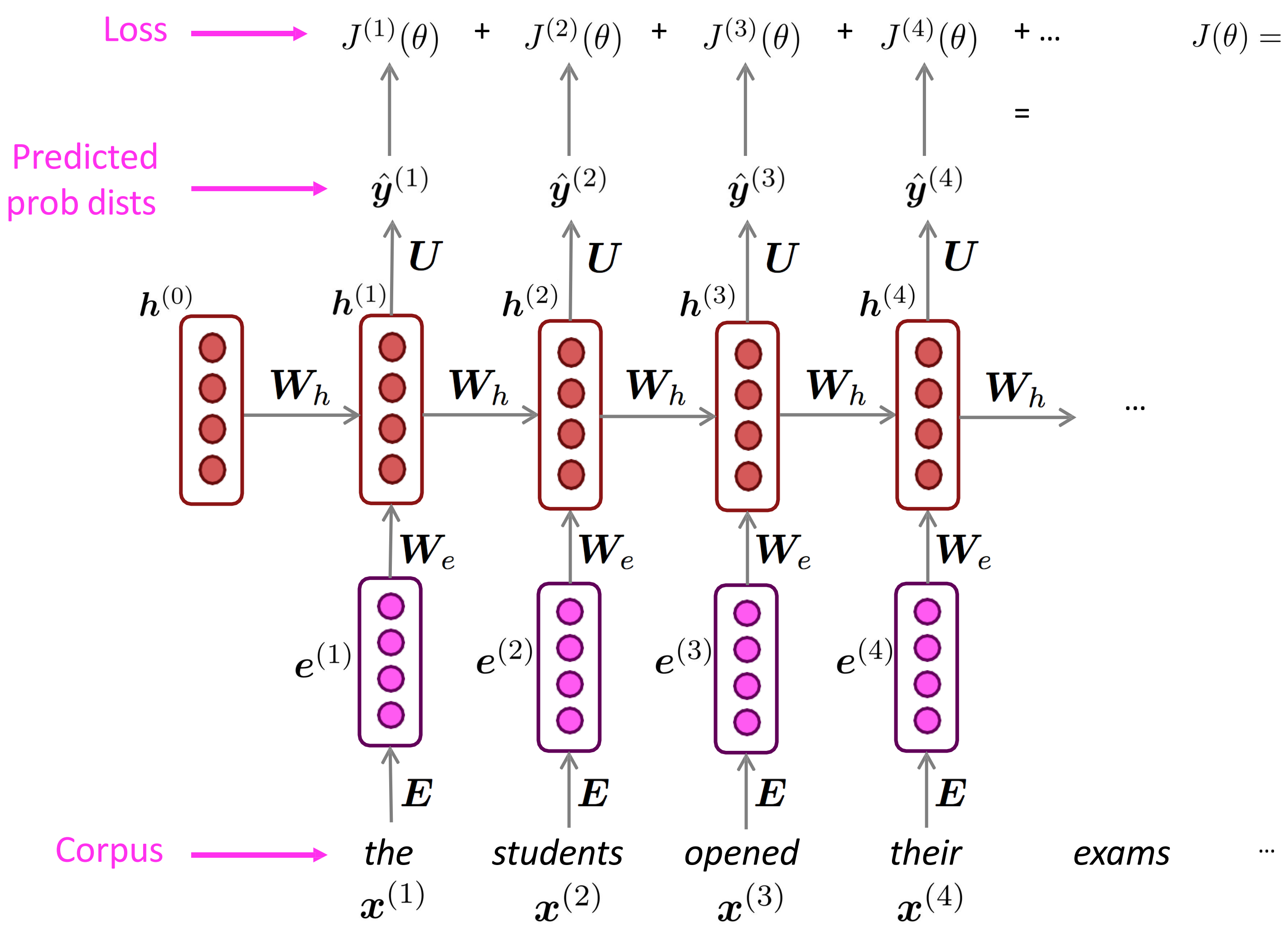
Language Model using RNN



Ref.: Stanford 224n

Language Model using RNN

“Teacher forcing”

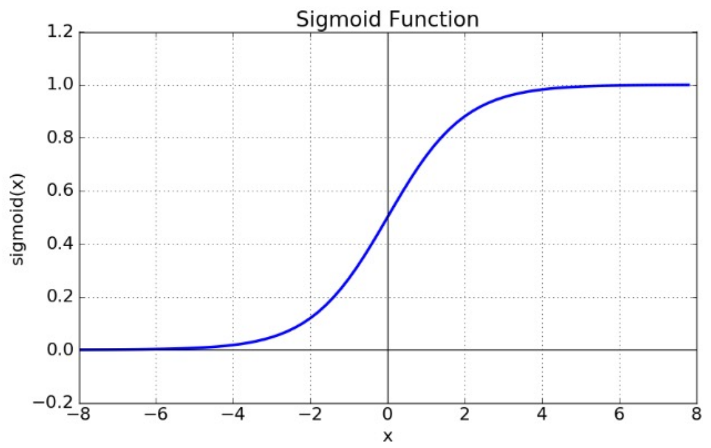


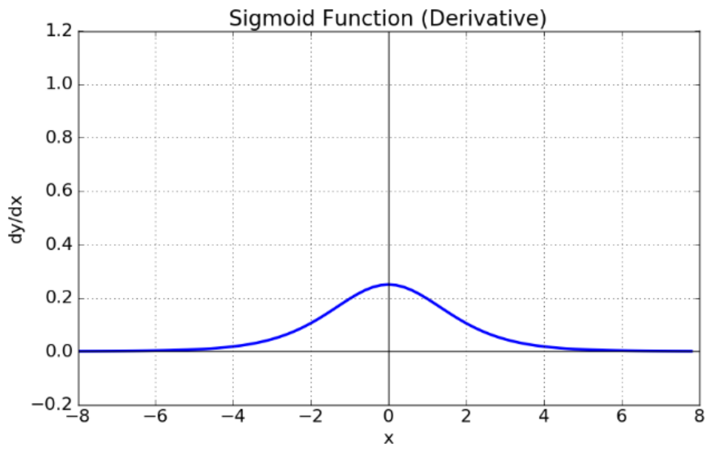
$$J(\theta) = \frac{1}{T}\sum_{t=1}^{T} J^{(t)}(\theta)$$

Why Hyperbolic Tangent?

- The sigmoid function outputs values close to 0 for negative inputs.

  Its derivative reaches a maximum of 0.25, leading to the problem of Vanishing Gradient.

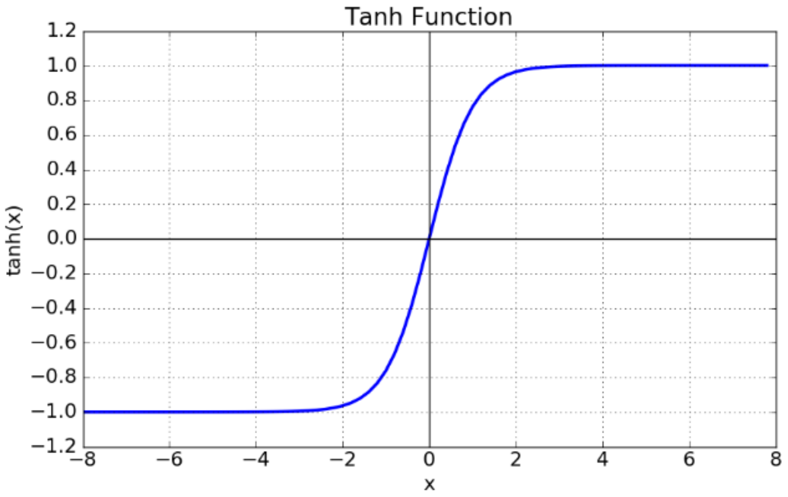


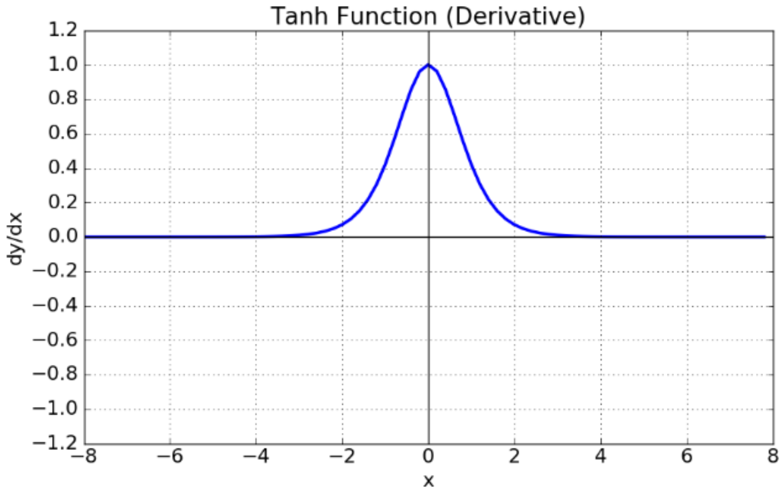


$$\frac{d}{d_x} sigmoid(x) = sigmoid(x)(1 - sigmoid(x))$$

Why Hyperbolic Tangent?

- To mitigate the Vanishing Gradient issue in RNNs, the Tanh function is used
  - The derivative of Tanh function has a maximum value of 1





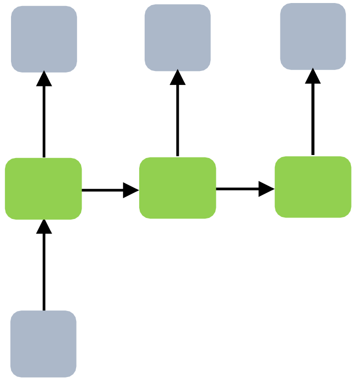
$$\boldsymbol{h_t = \tanh(X_t \cdot W_{xh} + h_{t-1} \cdot W_{hh} + b)}$$
$$\boldsymbol{y_t = W_{hy} \cdot h_t}$$

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

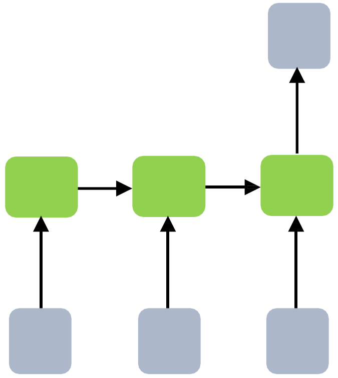
$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

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

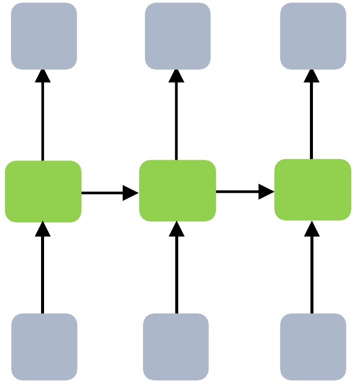
Structure of RNN



(one-to-many)



(many-to-one)

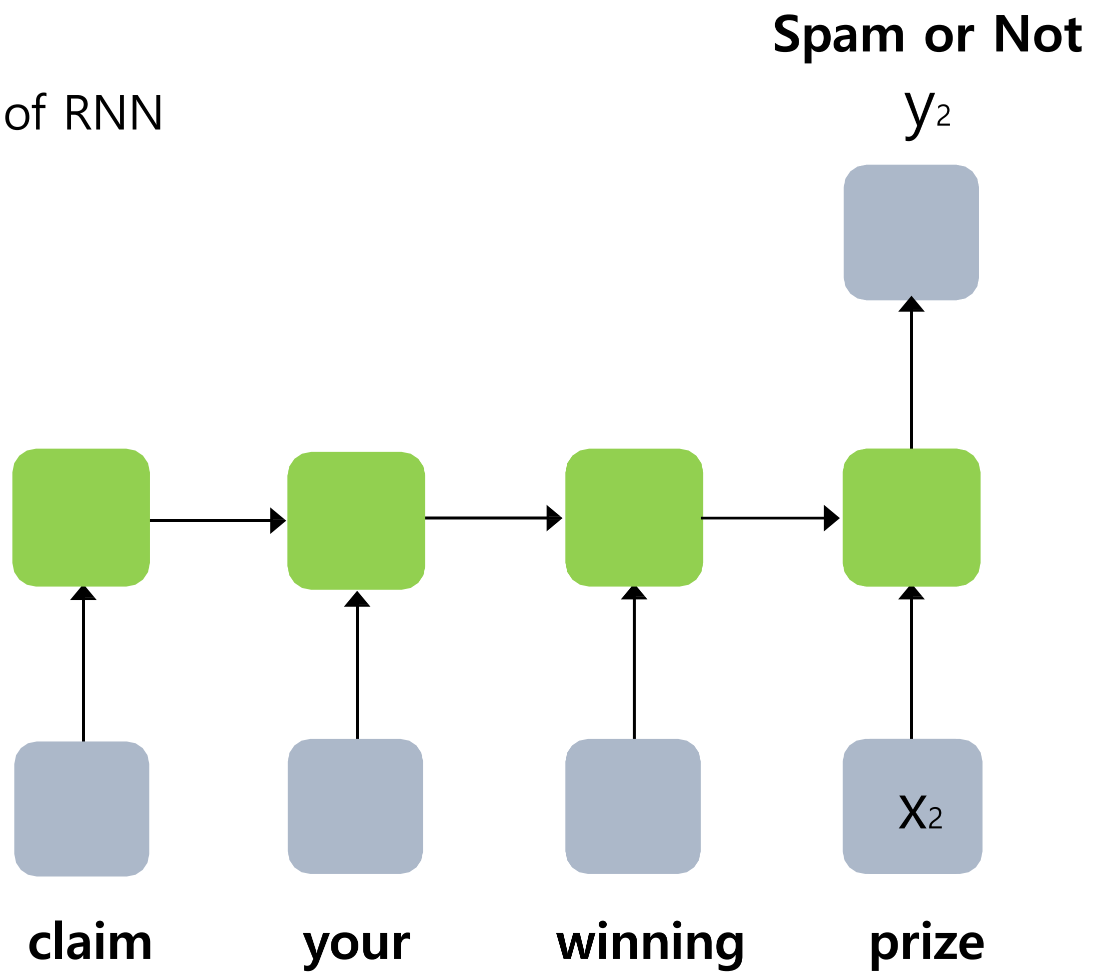


(many-to-many)

Figure Ref: https://wikidocs.net/book/2155

Structure of RNN

many-to-one Structure of RNN

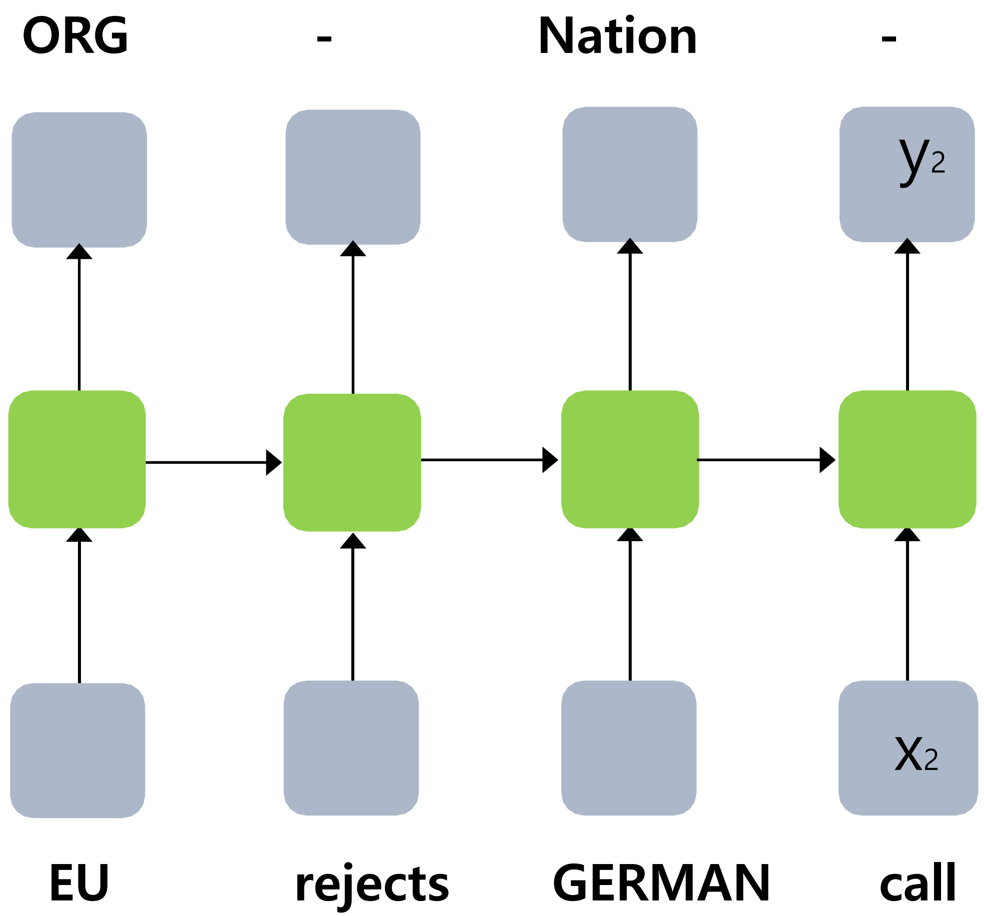


Spam Mail Recognizer

Figure Ref : https://wikidocs.net/book/2155

Structure of RNN

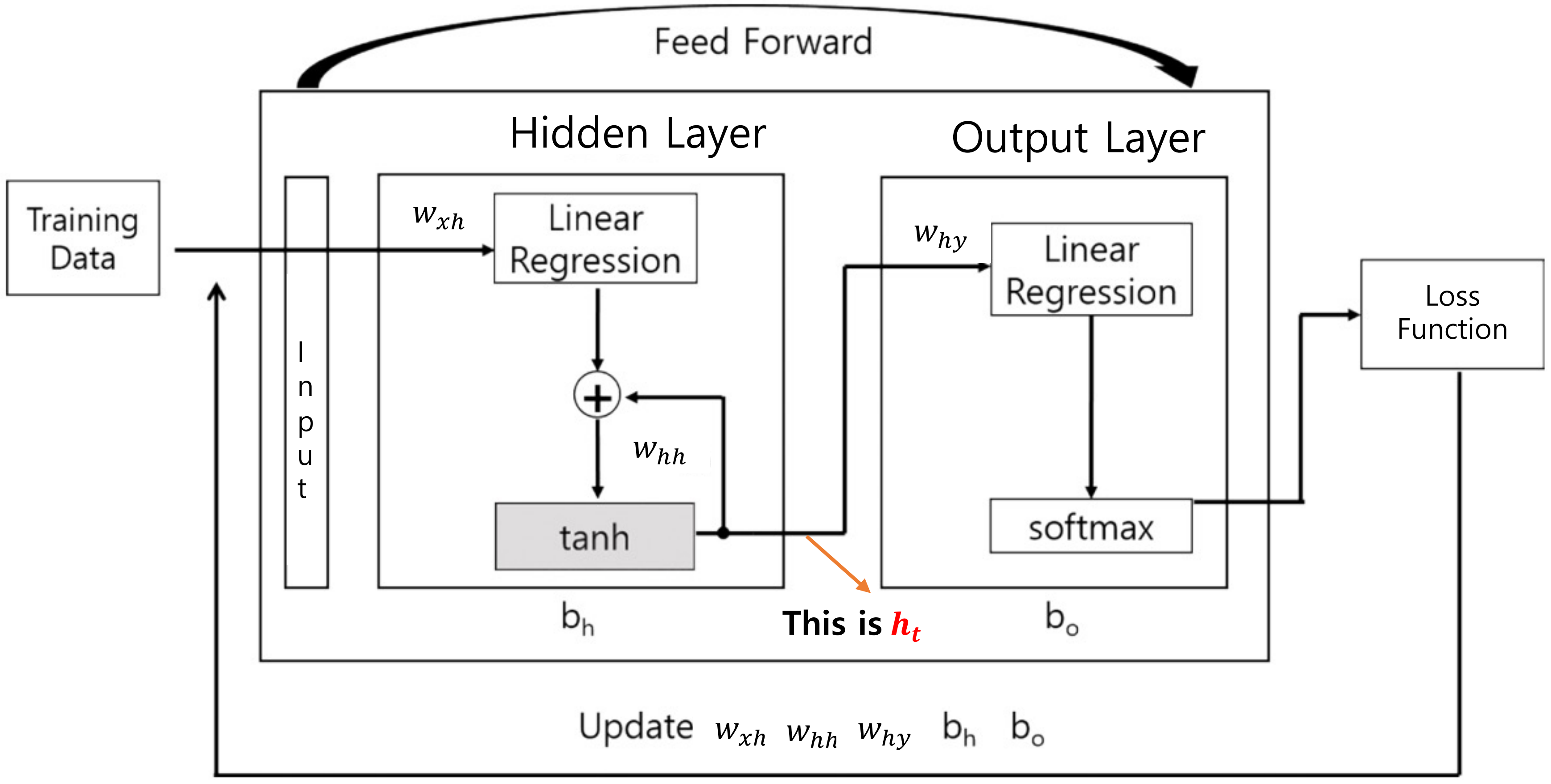
many-to-many Structure of RNN



Named Entity Recognition

Figure Ref : https://wikidocs.net/book/2155

Training Structure of RNN



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

$$y_t = W_{hy} \cdot h_t$$

Figure Ref: 머신러닝을 위한 파이썬 한조각

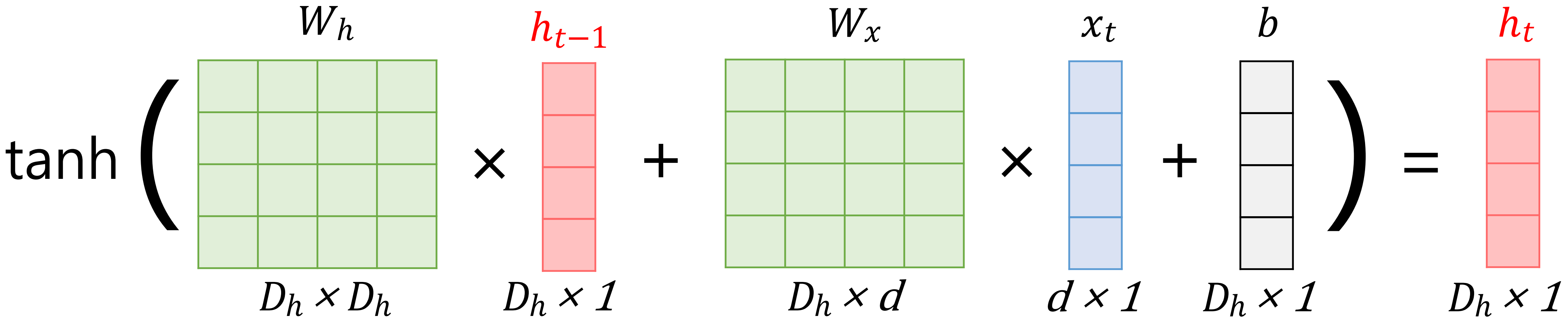
Assignment!

- Build your won language model with any corpora!
  - You should train your LM using RNN
  - Try not to use nn.RNN! (if it’s too hard just use it!)
  - No need to generate predicted results! just only train it yourself
  - (Optional) Implements generation function to generate text!!

Please Submit “xx.ipynb” file

Training Structure of RNN

Representation of RNN with vector and matrix operations



d: Dimension of the word at time-step t

D_h : Size of the hidden layer

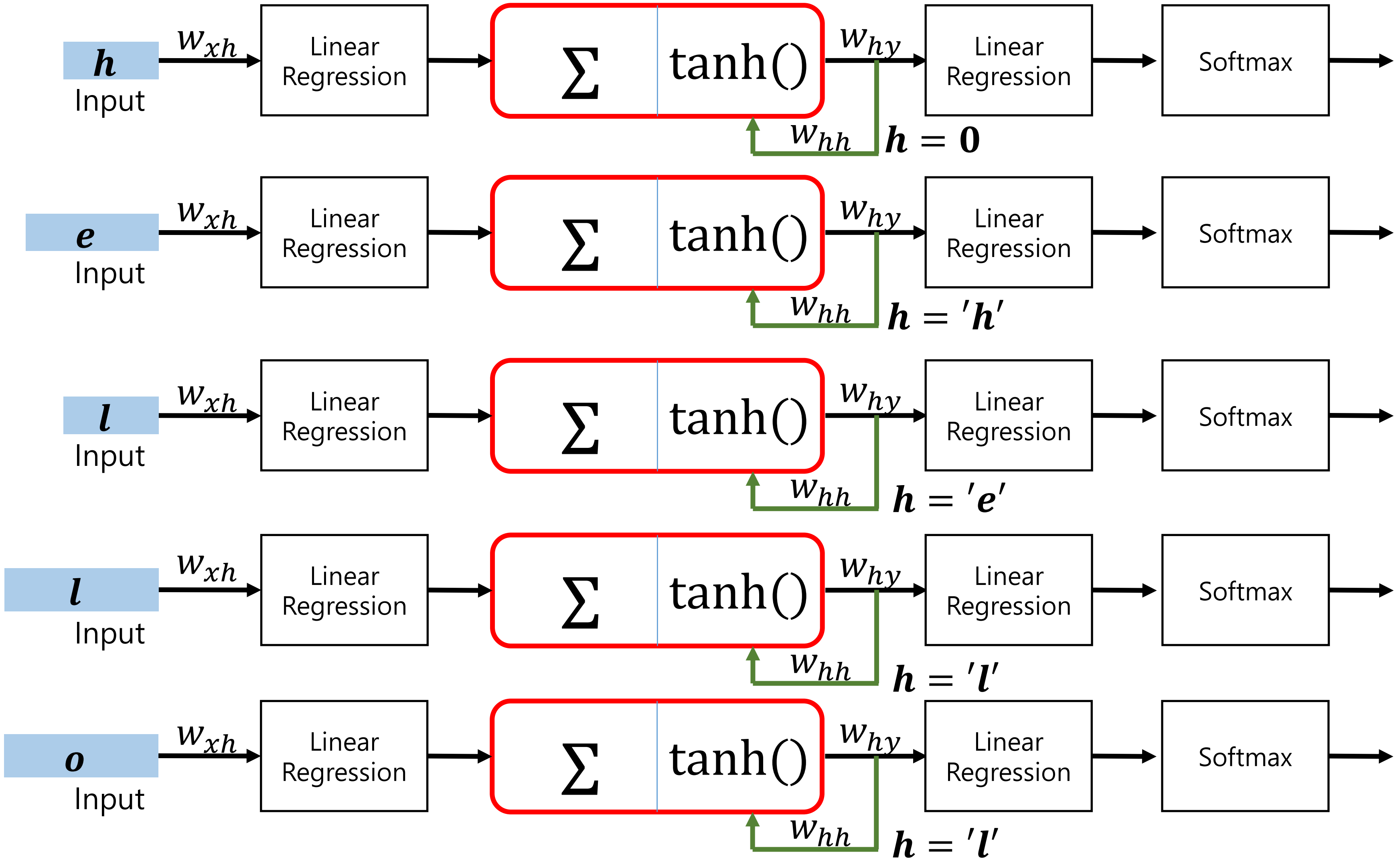
Figure Ref. : https://wikidocs.net/book/2155

CONTENTS

RNN Operation

Next character prediction model

When 'h' is input, predict 'e'
When 'e' is input, predict 'l'



h	0.01
e	0.97
l	0.01
o	0.01

h	0.02
e	0.21
l	0.75
o	0.02

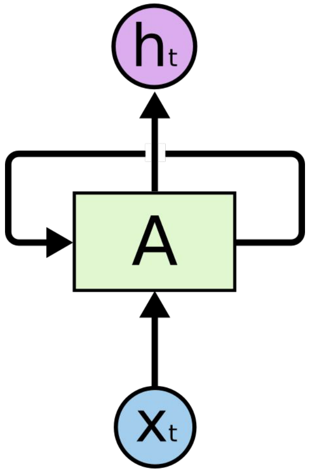
h	0.01
e	0.01
l	0.97
o	0.01

h	0.01
e	0.21
l	0.03
o	0.75

h	0.25
e	0.25
l	0.25
o	0.25

Basic implementation form of RNN

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

outputs, _status = rnn(input_data)
```



Ref: Deep Learning for All
http://colah.github.io/posts/2015-08-Understanding-LSTMs

Implementation of RNN (With RNN class)

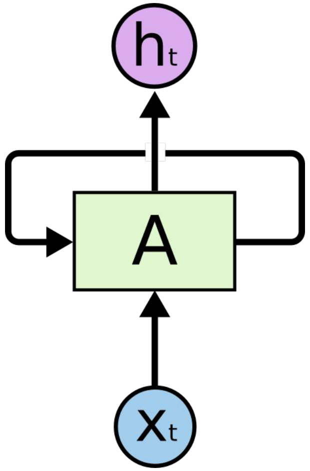
RNN input size and format (one-hot)

“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=(-,-,-)



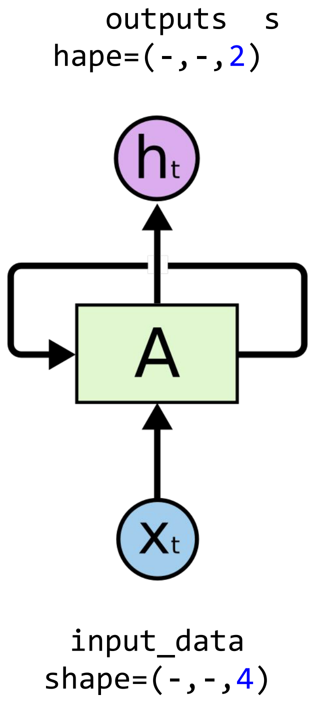
input_data
shape=(-,-,4)

Implementation of RNN (With RNN class)

RNN’s hidden state

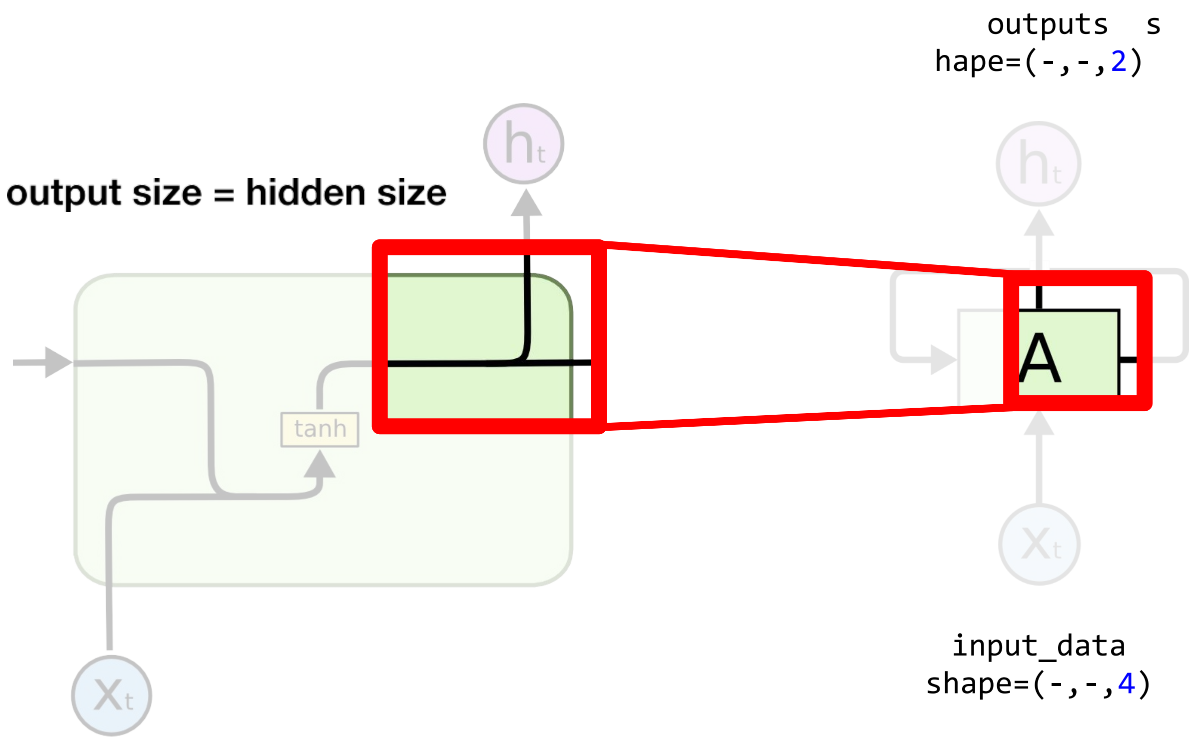
desirable output size

hidden_size = 2



Implementation of RNN (With RNN class)

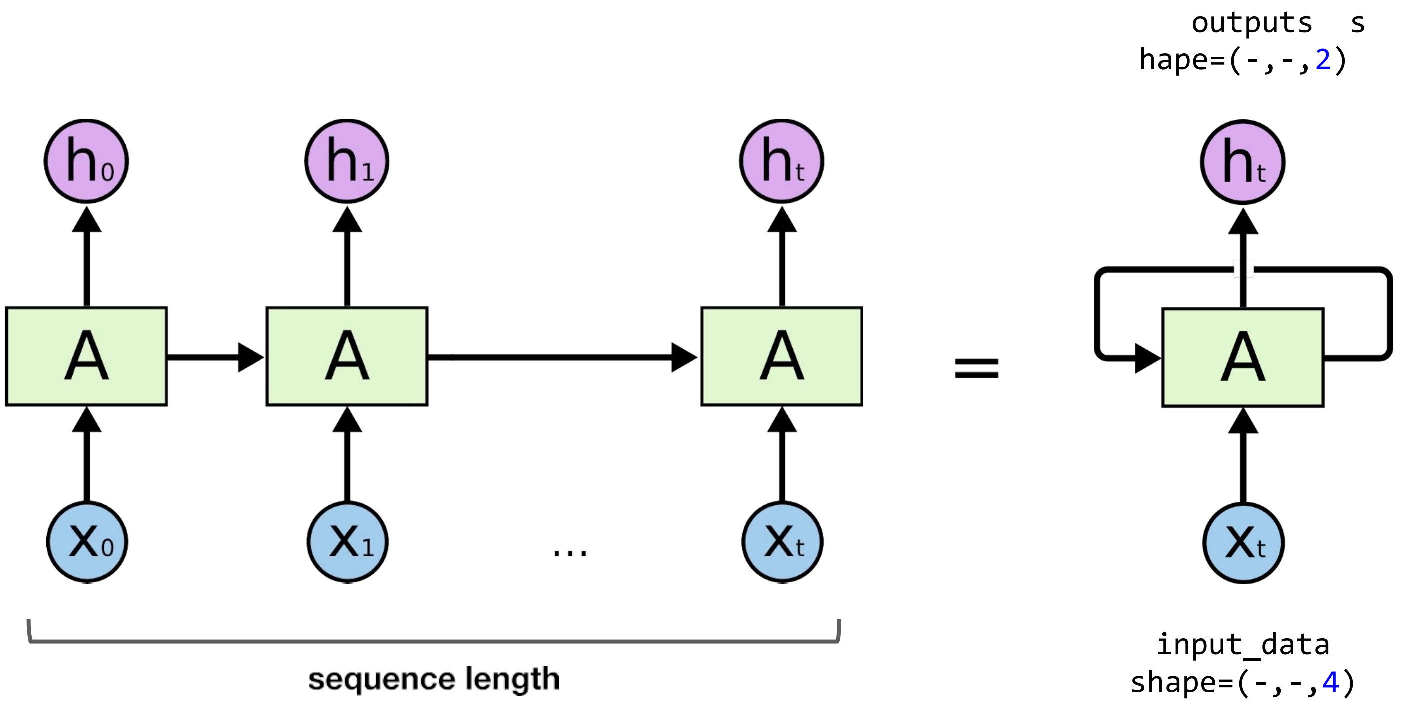
RNN's output size



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

Implementation of RNN (With RNN class)

Sequence length of RNN input

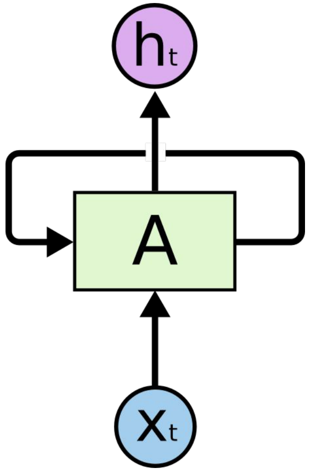


http://colah.github.io/posts/2015-08-Understanding-LSTMs

Implementation of RNN (With RNN class)

Sequence length of RNN input

outputs s
hape=(-,5,2)



input_data
shape=(-,5,4)

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

Implementation of RNN (With RNN class)

Batch size for processing multiple samples simultaneously

h, e, l, l, o

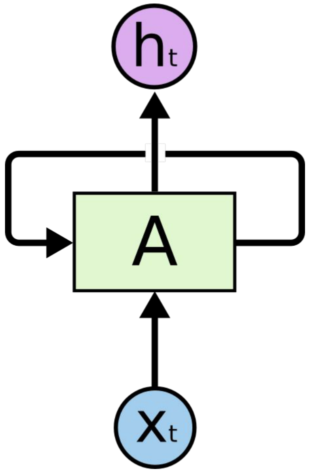
e, o, l, l, l

l, l, e, e, l

Set
batch input

Automatically Calculated

outputs s
hape=(3,5,2)



input_data
shape=(3,5,4)

Implementation of RNN (With RNN class)

Example of Character Sequence Prediction

```
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)
```

'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?

Implementation of RNN (With RNN class)

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']
```

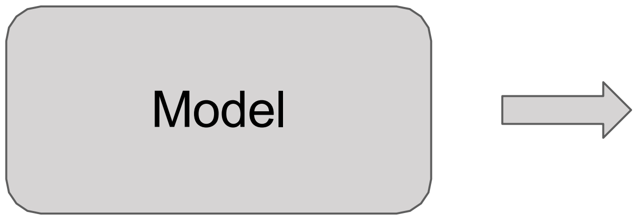
One-hot encoding

- 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

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

Code run through (hihello)

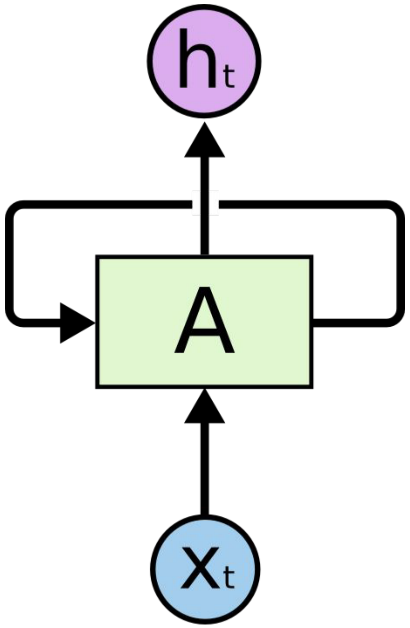
```
char_set = ['h', 'i', 'e', 'l', 'o']
# hyper parameters
input_size = len(char_set)
hidden_size = len(char_set)
learning_rate = 0.1
# data setting
x_data = [[0, 1, 0, 2, 3, 3]]
x_one_hot = [[[1, 0, 0, 0, 0],
              [0, 1, 0, 0, 0],
              [1, 0, 0, 0, 0],
              [0, 0, 1, 0, 0],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

Implementation of RNN (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)
```



Implementation of RNN (With RNN class)

Code run through (charseq)

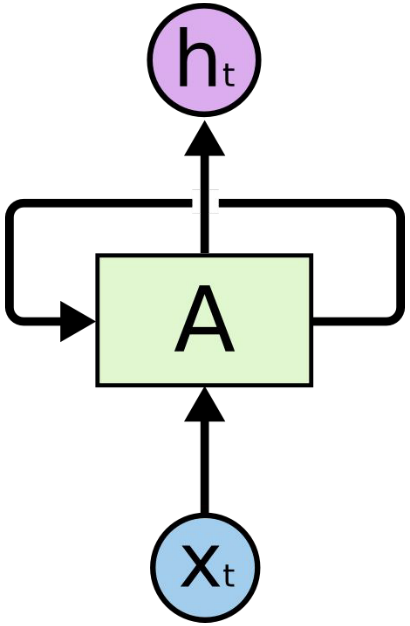
```
sample = " if you want you"
# make dictionary
char_set = list(set(sample))
char_dic = {c: i for i, c in enumerate(char_set)}

# 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)
```

Implementation of RNN (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?

Implementation of RNN (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 "
```

...

Implementation of RNN (With RNN class)

Making sequence dataset from long sentence (code)

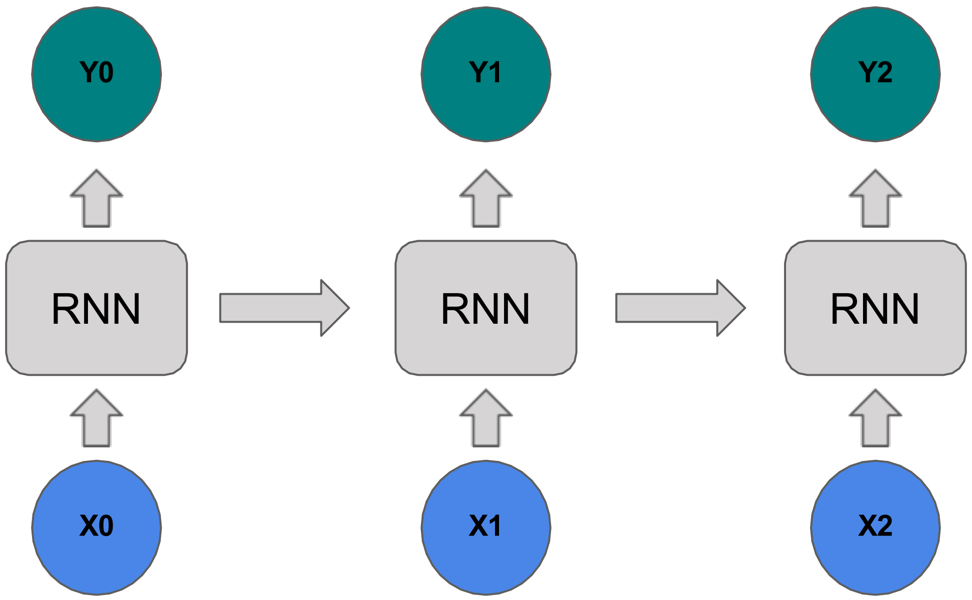
```
# data setting
x_data = []  y
_data = []

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 "

...
```

Implementation of RNN (With RNN class)

Adding FC layer and stacking RNN

```python
# 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)
```

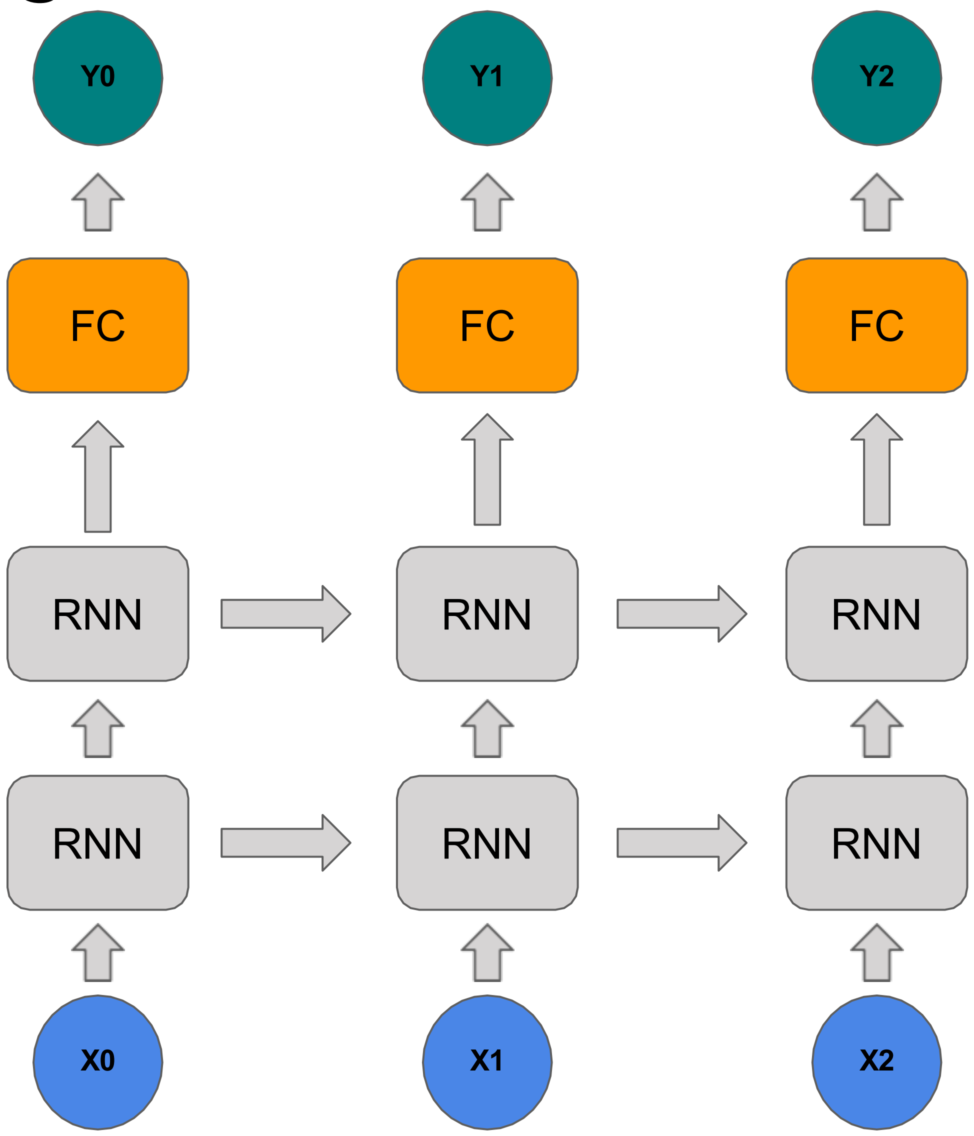


Vanilla RNN

Implementation of RNN (With RNN class)

Adding FC layer and stacking RNN

```python
# 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)
```



Implementation of RNN (With RNN class)

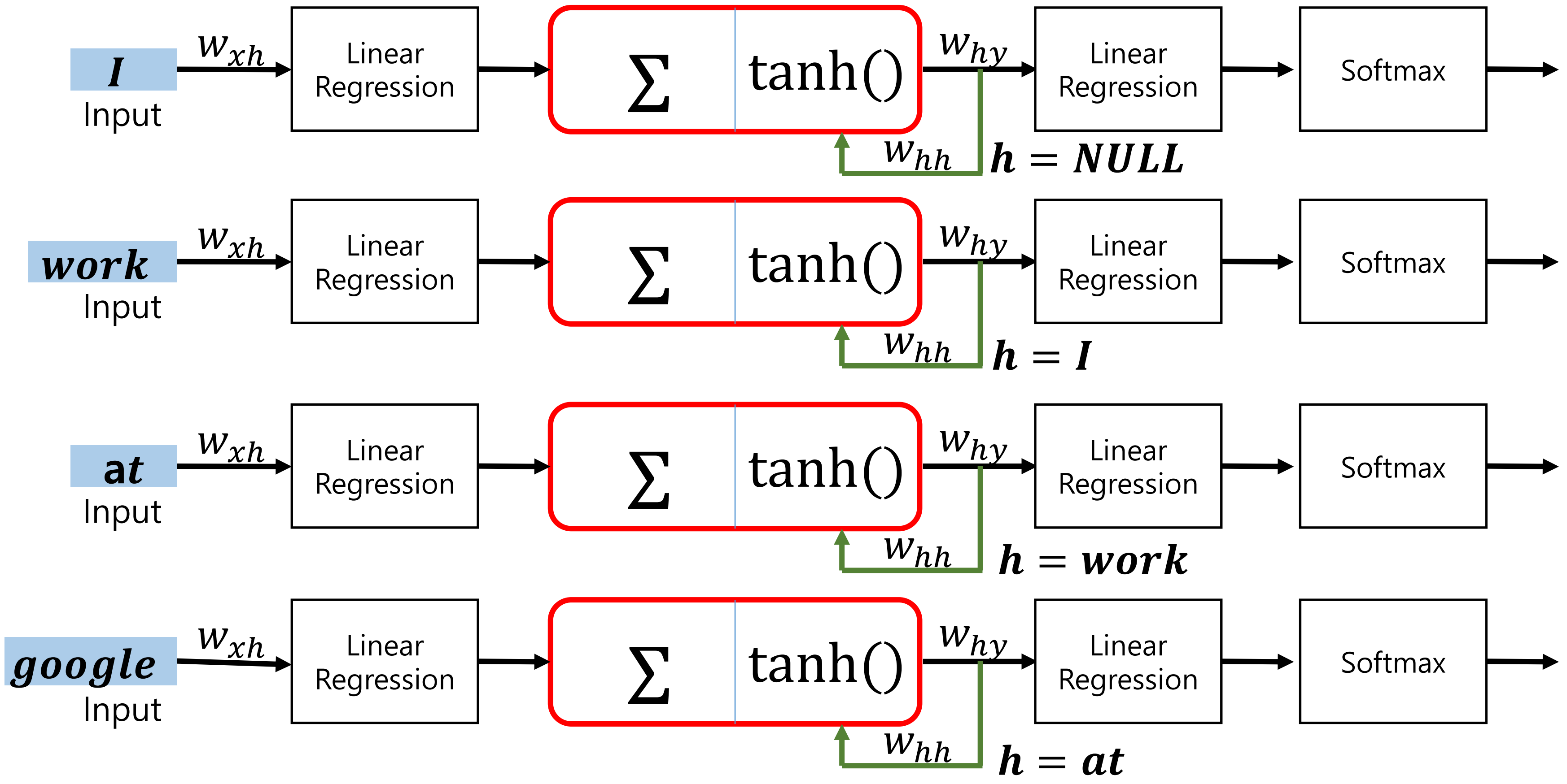
Code run through

```
# 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]]
```

Application of RNN

Sentiment analysis: Is a movie review positive or negative?

- 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

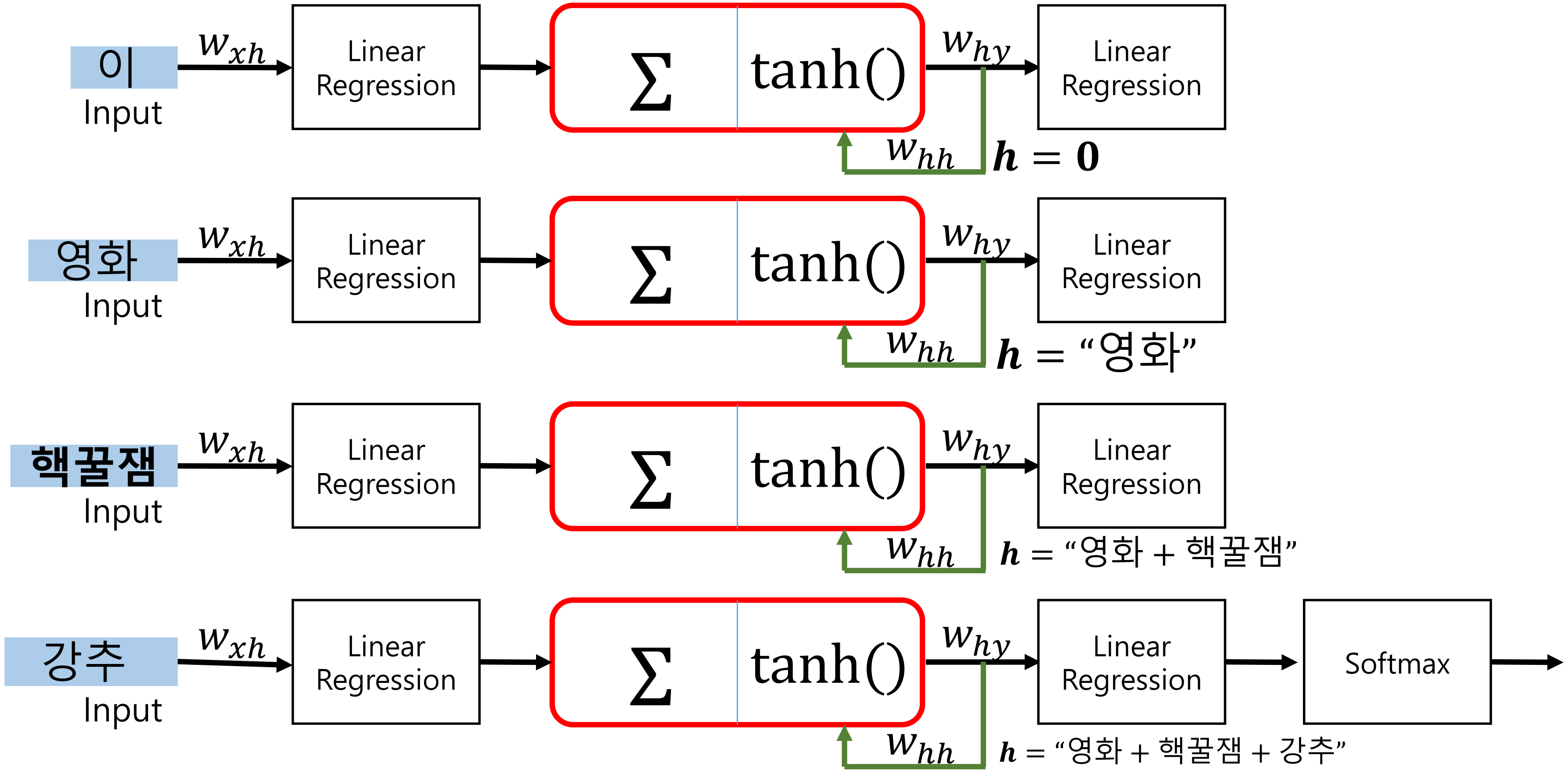
대명사	0.02
명사	0.21
동사	0.75
전치사	0.02

대명사	0.01
명사	0.01
동사	0.01
전치사	0.97

대명사	0.02
명사	0.75
동사	0.21
전치사	0.02

Application of RNN

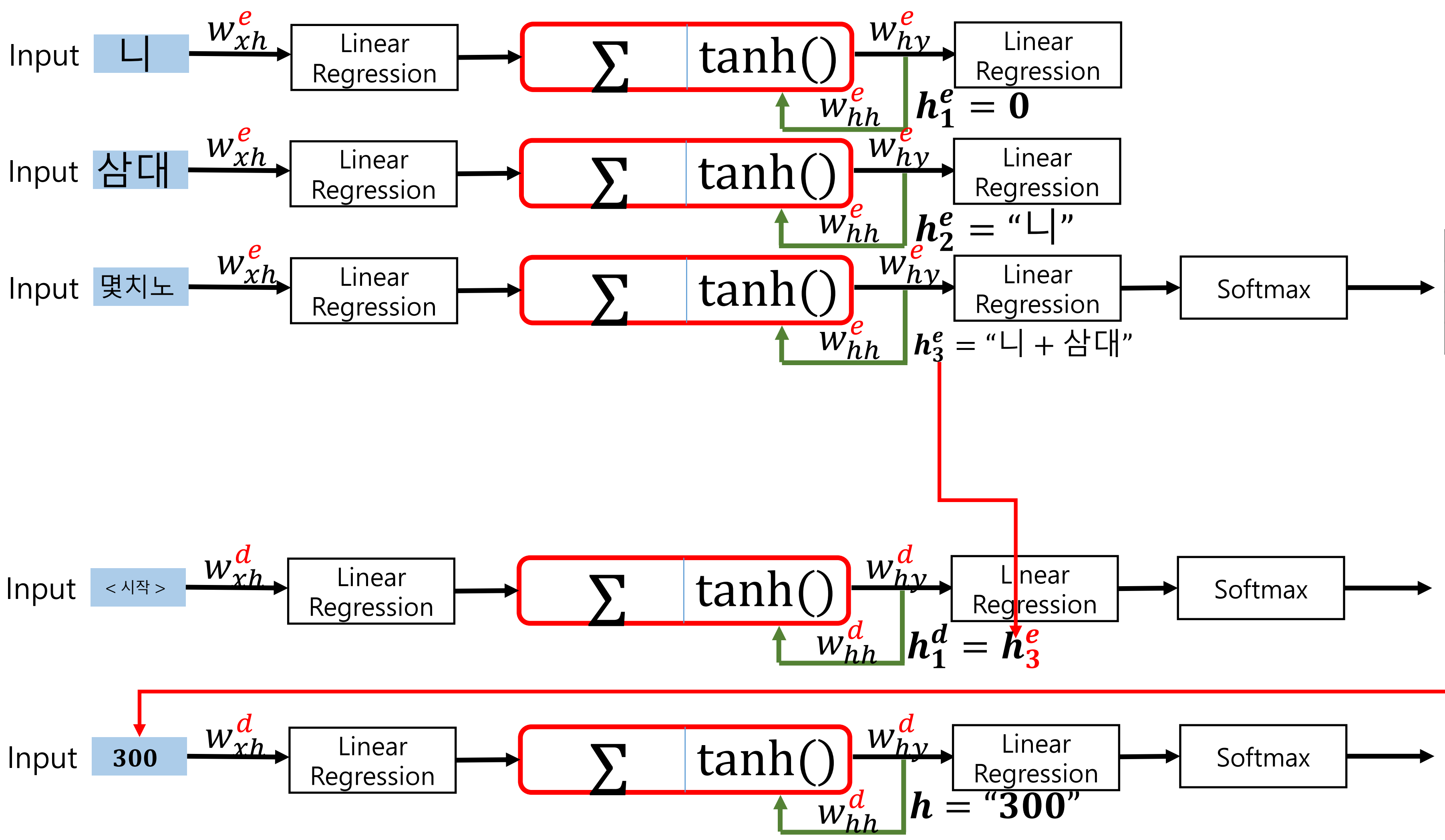
Sentiment analysis: Is a movie review positive or negative?



부정	0.04
긍정	0.75
일반	0.21

Application of RNN

Chatbot (seq2seq)



추천	0.01
움팡지다	0.01
관종	0.01
300	0.97

낄낄	0.01
미쳤습니까	0.21
휴먼	0.03
300	0.75

정도?	0.75
움팡지다	0.00
관종	0.00
추천	0.25

Further Discussion

- RNN Training Methods

  ✓ RTRL (Real-time recurrent learning): Uses stochastic gradient descent for online learning
  ✓ BPTT (Backpropagation through time): Time-based error backpropagation

- Advantages of RNN
  - Utilizes previous information for solving current problems

- Disadvantages of RNN
  ✓ Long-term dependency: Difficulty handling context from distant past states due to gradient vanishing
  → Resolved by LSTM (Long Short Term Memory) and GRU (Gated Recurrent Units)

Thank you.