



LAB 5: RECURRENT NEURAL NETWORKS (RNNs)

University of Washington, Seattle

Spring 2022



OUTLINE

[Part 1: Introduction to RNNs](#) (video link)

- [Why do we need RNNs?](#)
- [RNN Architecture](#)
- [RNN in PyTorch](#)
- [Embedding and Decoder](#)

[Part 2: Training RNNs](#) (video link)

- [Backpropagation in RNNs](#)
- [Vanishing/Exploding Gradient Problem](#)
- [Training with Teacher Forcing](#)

[Part 3: RNN Problem Types](#) (video link)

- [RNN Configurations](#)
- [RNN Extensions](#)

[Part 4: RNN Implementation in PyTorch](#) (video link)

- [Character Level Generation Shakespeare Dataset](#)

[Lab Assignment](#) (video link)

- [Create Arthur Conan Doyle AI](#)



INTRODUCTION TO RNNs

Why do we need RNNs?

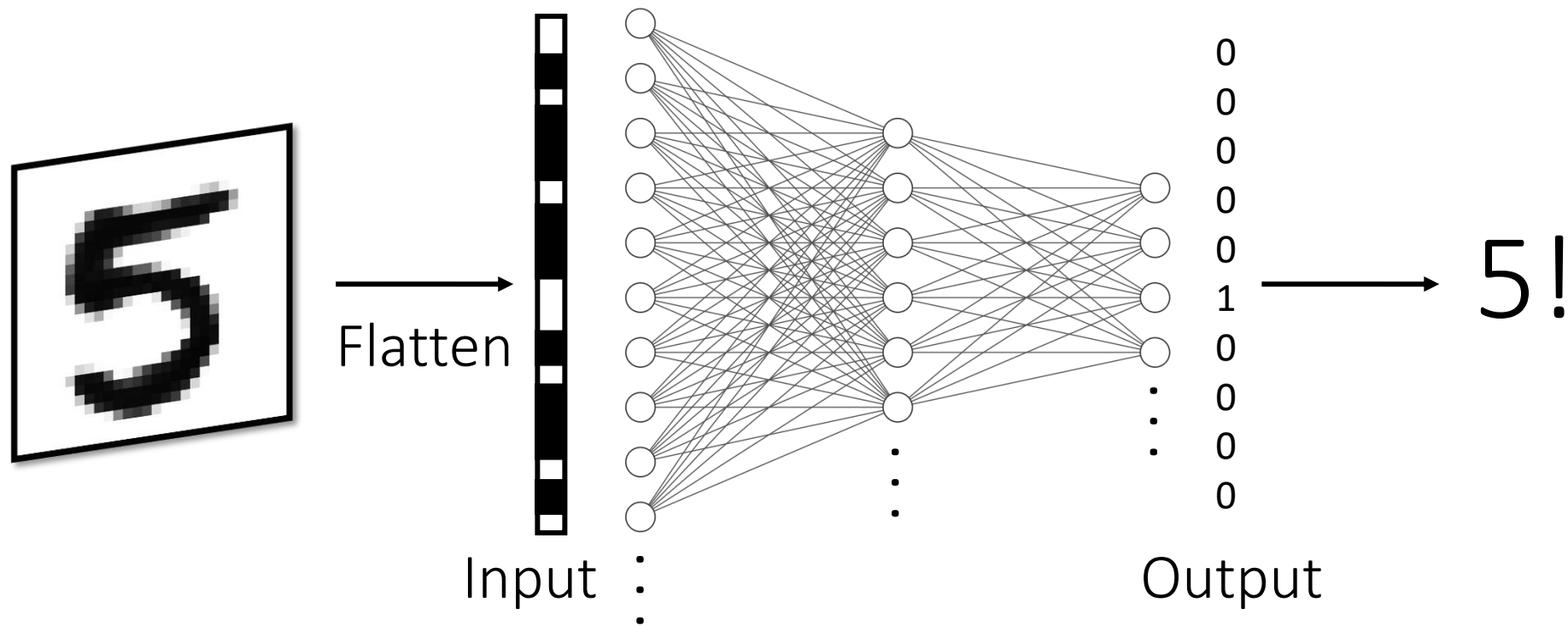
RNN Architecture

RNN in PyTorch

Embedding and Decoder



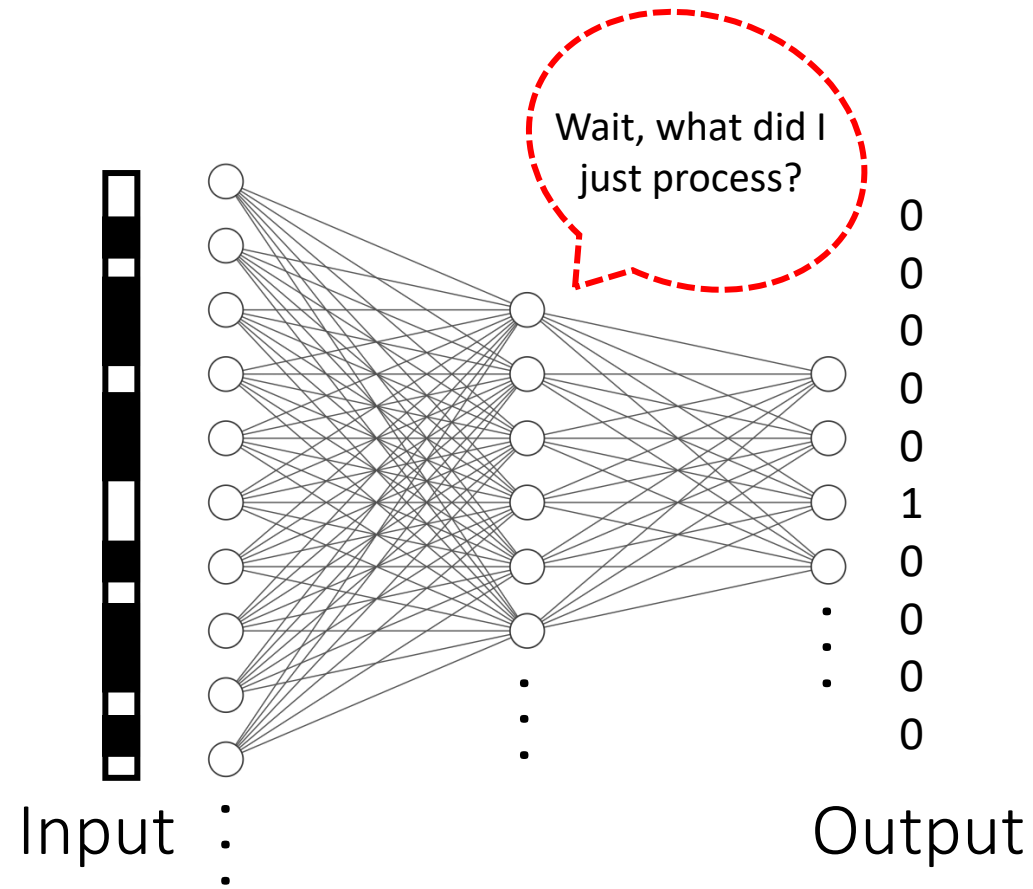
Why Do We Need RNNs?



Feed-Forward Network



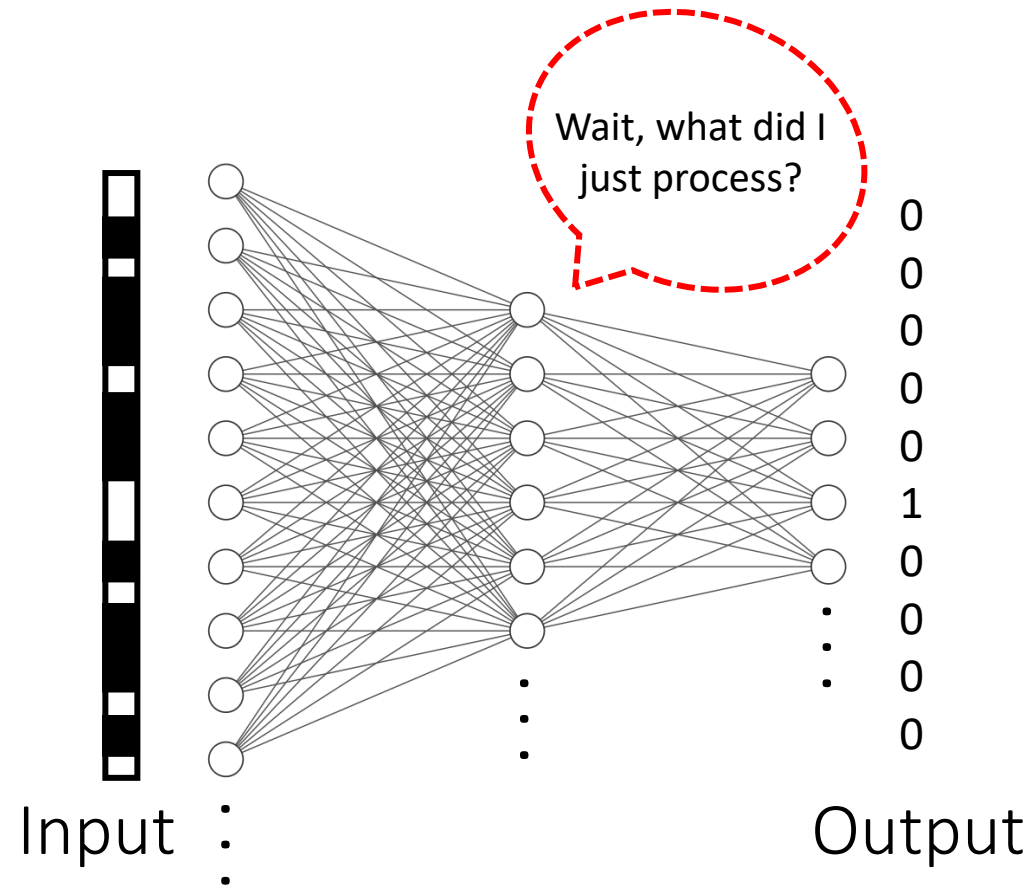
Why Do We Need RNNs?



Feed-Forward Network



Why Do We Need RNNs?



Feed-Forward Network has no memory of past inputs



Why Do We Need RNNs?

Korean

안녕하세요, 제 이름은 지민이에요



English

Hello, my name is Jimin



Why Do We Need RNNs?

Korean

안녕하세요,

English

Hello,



Why Do We Need RNNs?

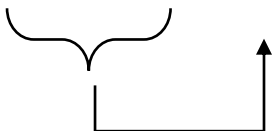
Korean

안녕하세요, 제



English

Hello, my

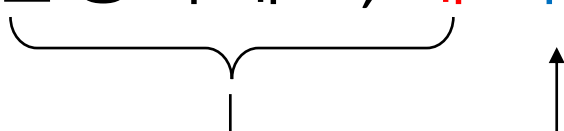




Why Do We Need RNNs?

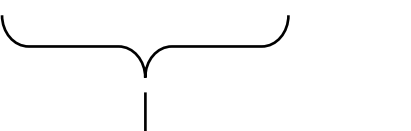
Korean

안녕하세요, 제 이름



English

Hello, my name






Why Do We Need RNNs?

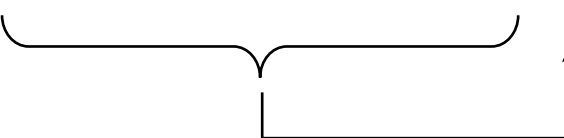
Korean

안녕하세요, 제 이름은



English

Hello, my name is





Why Do We Need RNNs?

Korean

안녕하세요, 제 이름은 지민이에요

The diagram illustrates the sequential nature of Korean. A bracket under '안녕하세요,' is followed by another bracket under '제 이름은', with an arrow pointing from the second bracket to '지민이에요', indicating that the meaning of the second part depends on the first.

English

Hello, my name is Jimin

The diagram illustrates the sequential nature of English. A bracket under 'Hello,' is followed by another bracket under 'my name is', with an arrow pointing from the second bracket to 'Jimin', indicating that the meaning of the second part depends on the first.



Why Do We Need RNNs?

Korean

안녕하세요, 제 이름은 지민이에요

English

Hello, my name is Jimin

Each word in a sentence is dependent to the past words →

Need memory



Why Do We Need RNNs?

Korean 안녕하세요, 제 이름은 지민이에요, 그리고 저는 비디오게임을 좋아해요

English Hello, my name is Jimin, and I like videogames

A sentence (input) could have **different sizes**



Why Do We Need RNNs?

We need a neural network architecture that can handle:



Why Do We Need RNNs?

We need a neural network architecture that can handle:

- Data order



Why Do We Need RNNs?

We need a neural network architecture that can handle:

- Data order
- Temporal dependencies



Why Do We Need RNNs?

We need a neural network architecture that can handle:

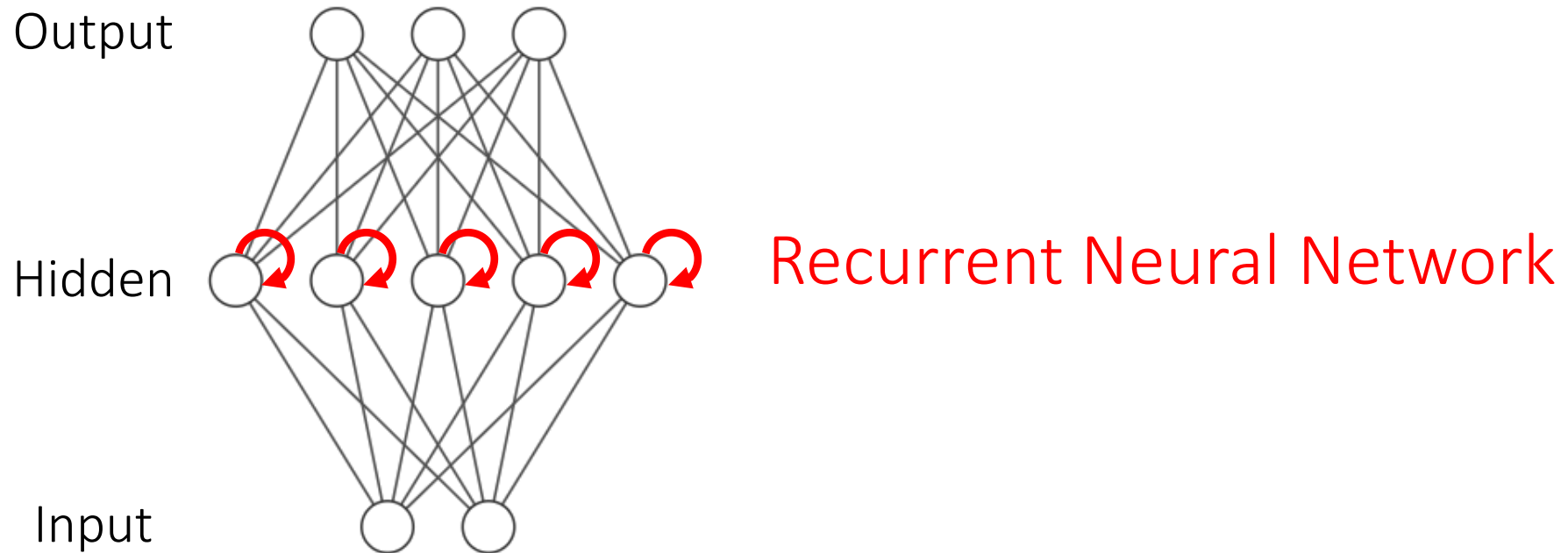
- Data order
- Temporal dependencies
- Variable input sizes



Why Do We Need RNNs?

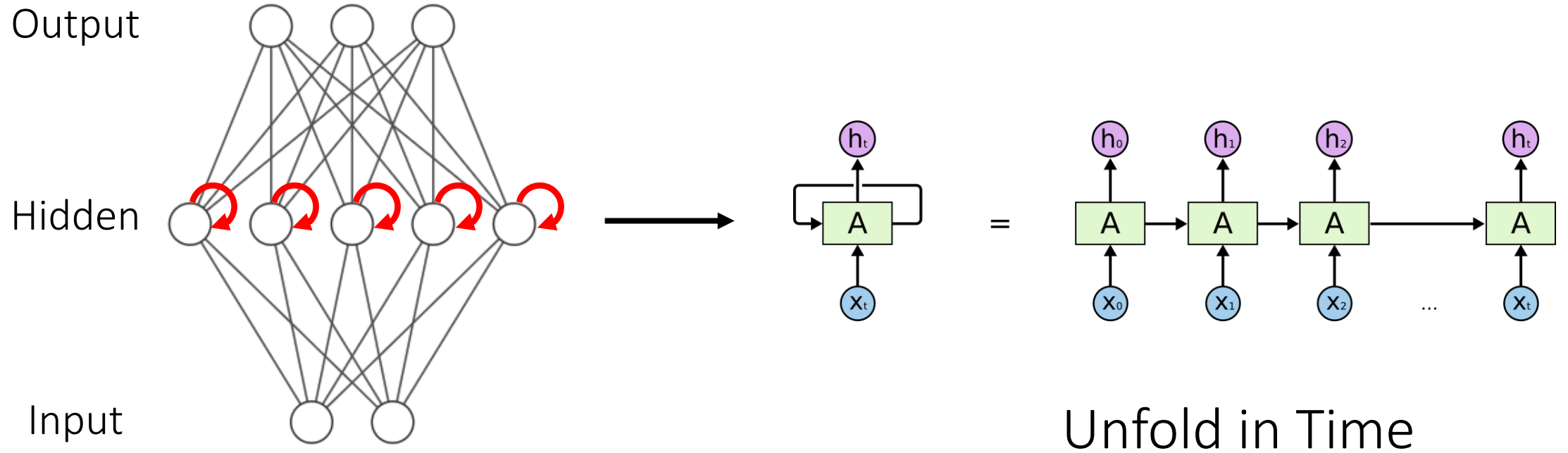
We need a neural network architecture that can handle:

- Data order
- Temporal dependencies
- Variable input sizes



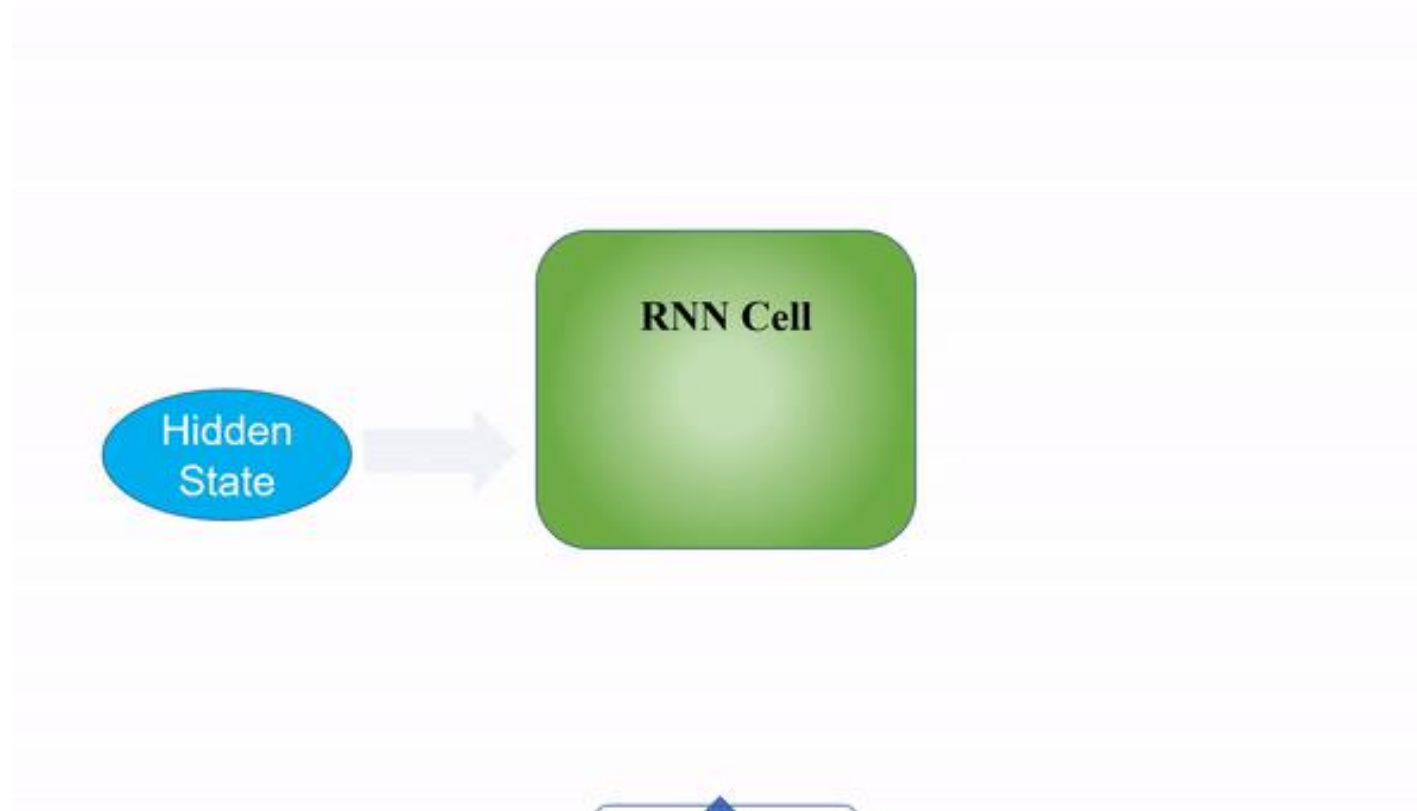


RNN Architecture



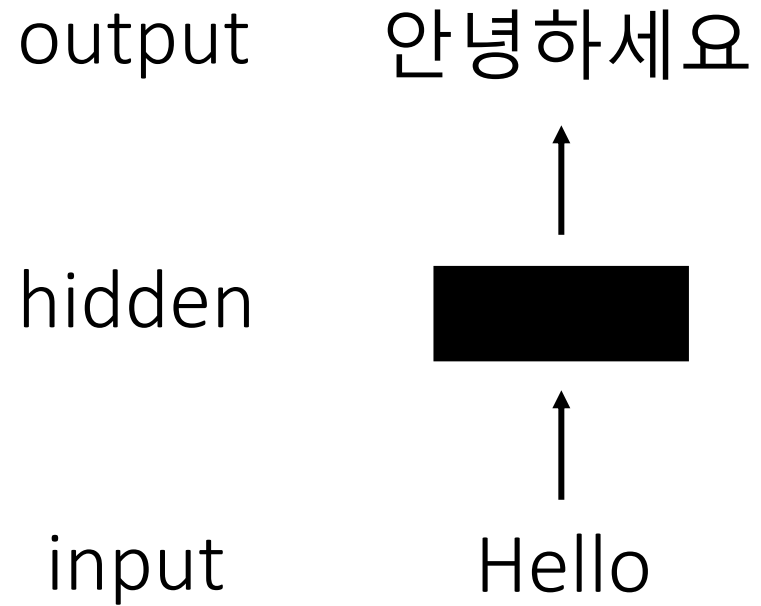


RNN Architecture



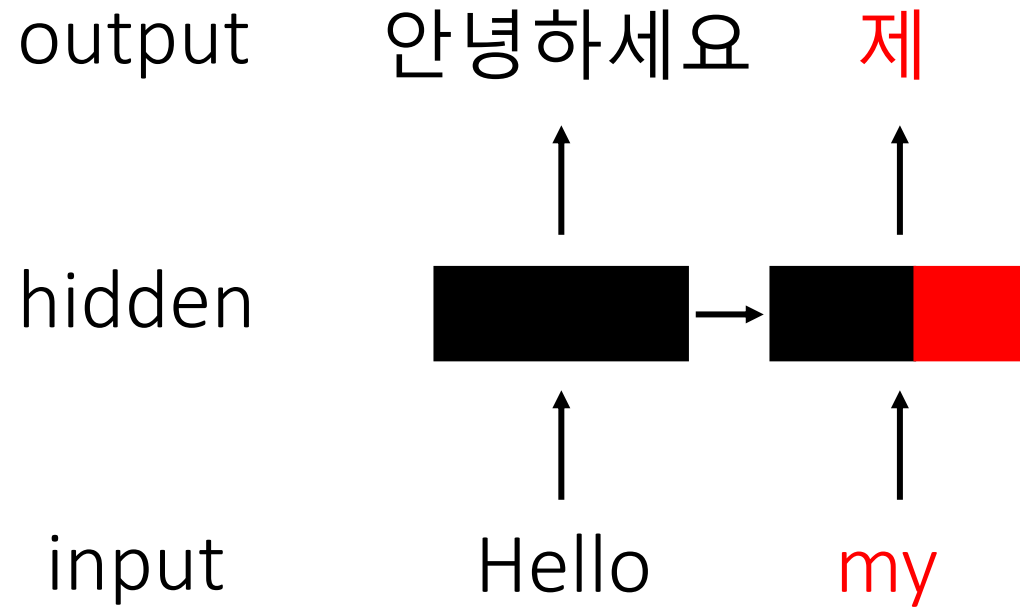


RNN Architecture



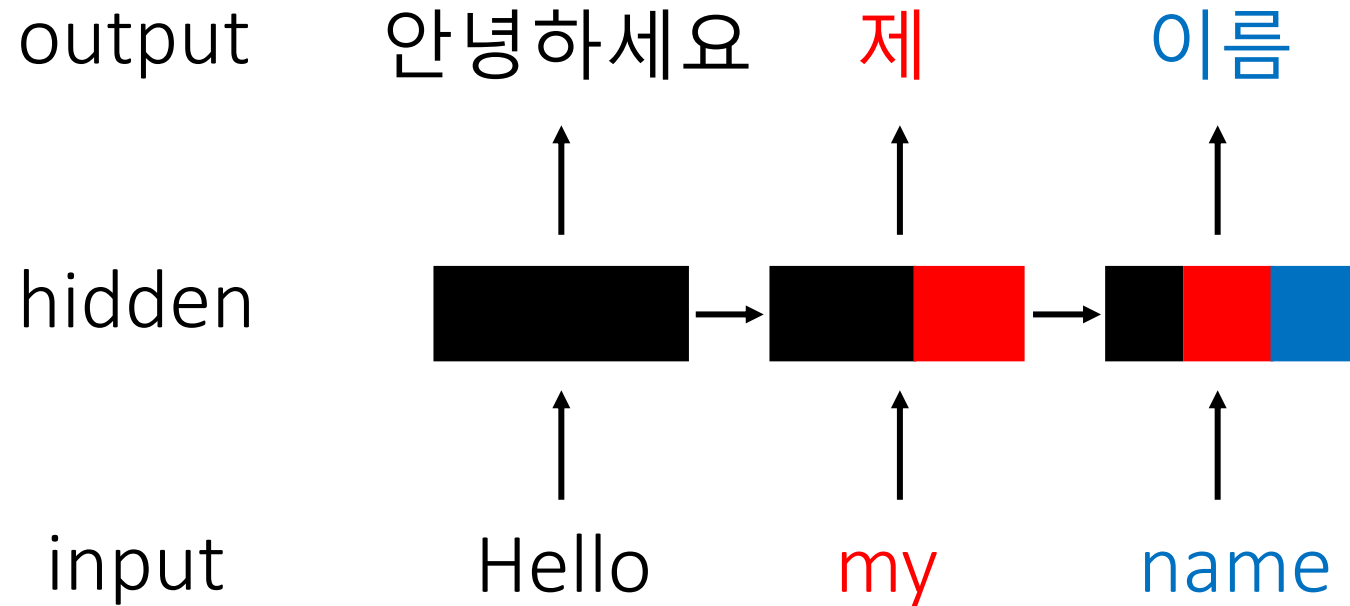


RNN Architecture



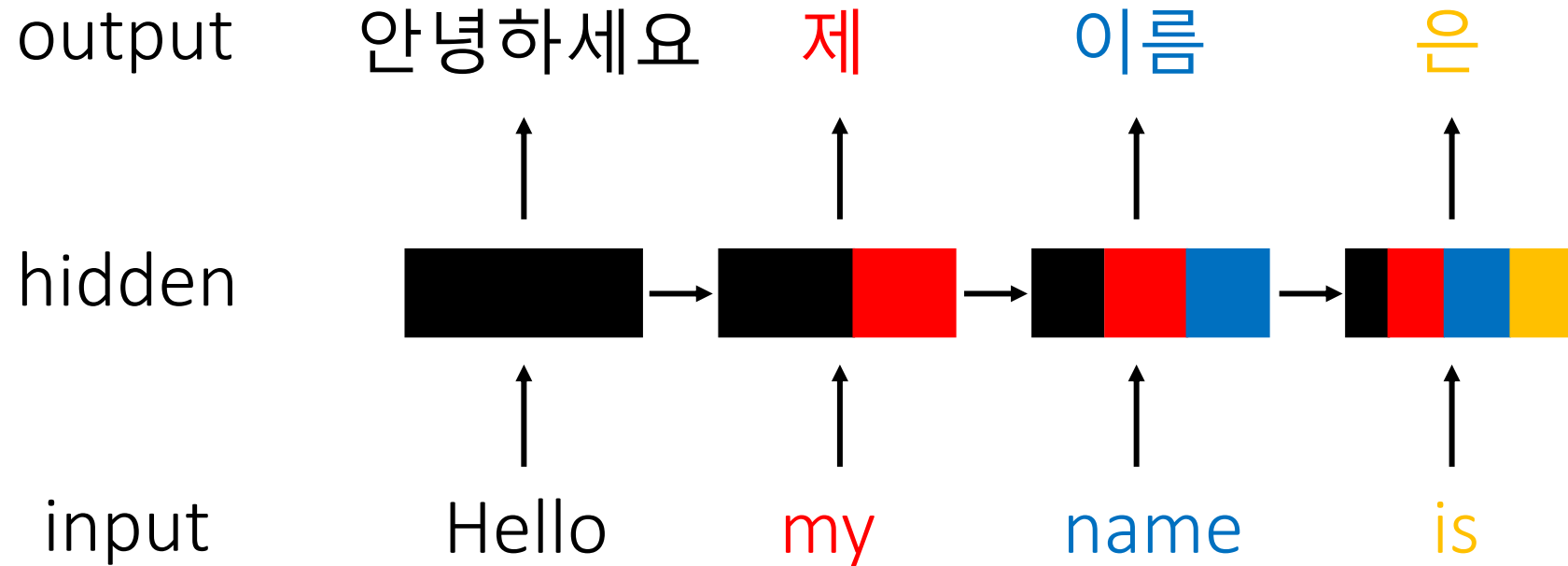


RNN Architecture



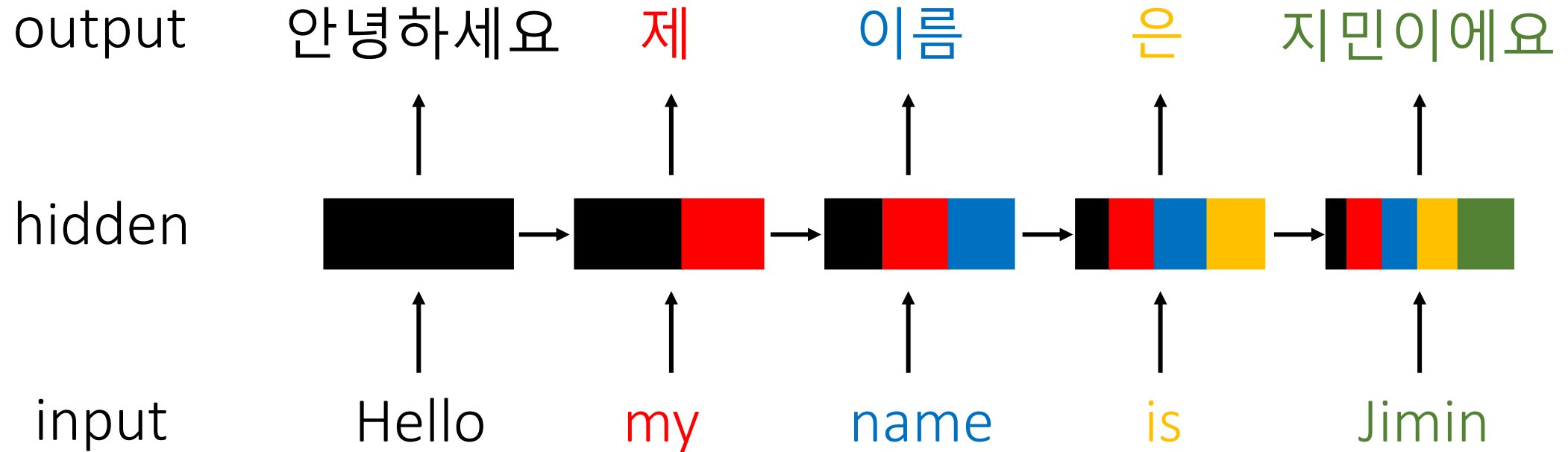


RNN Architecture





RNN Architecture





Sequential Data

Speech recognition



"The quick brown fox jumped
over the lazy dog."

Music generation

∅



Sentiment classification

"There is nothing to like
in this movie."



DNA sequence analysis

AGCCCCTGTGAGGAACTAG



AG**CCCCTGTGAGGAACT**AG

Machine translation

Voulez-vous chanter avec
moi?



Do you want to sing with
me?

Video activity recognition



Running

Name entity recognition

Yesterday, Harry Potter
met Hermione Granger.



Yesterday, **Harry Potter**
met **Hermione Granger**.



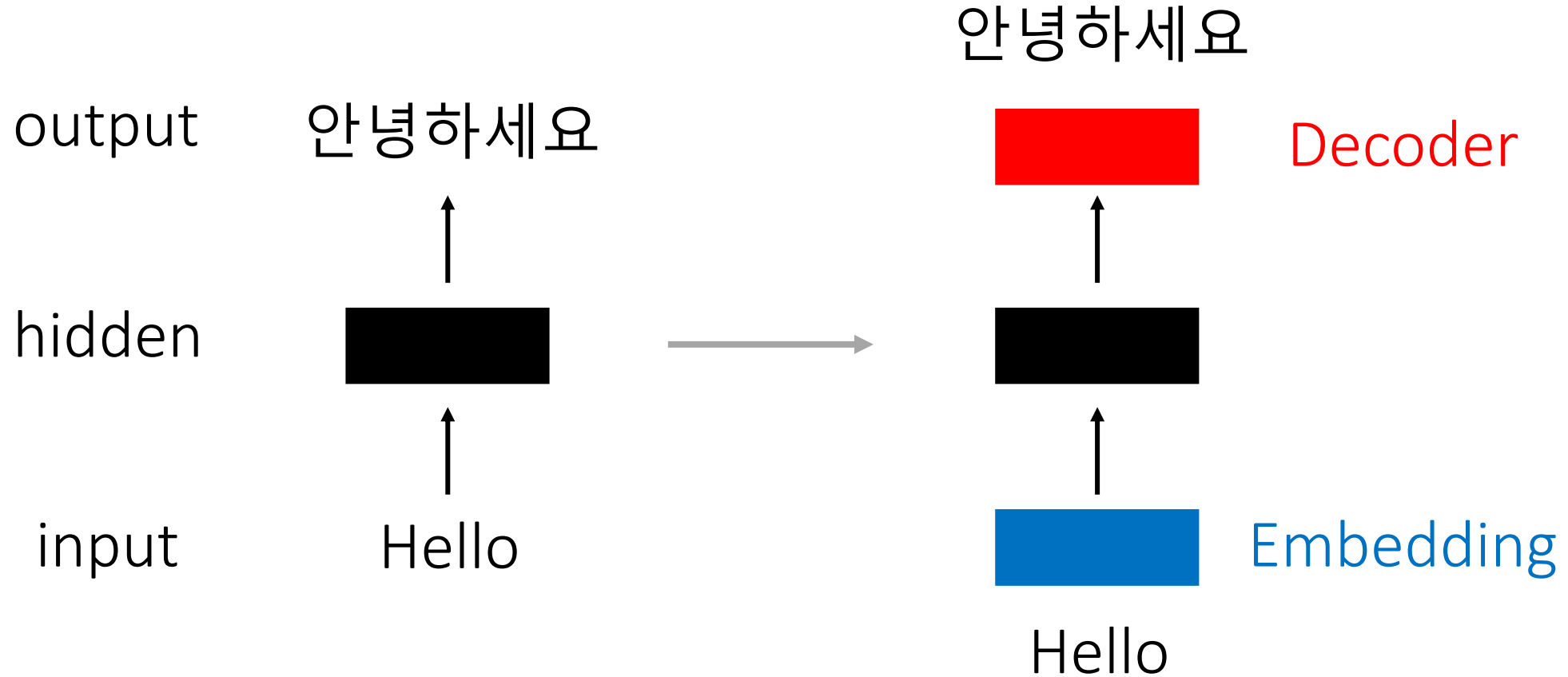
RNN in PyTorch

| <code>torch.nn.RNN(</code> | Parameter description | Data type |
|-----------------------------|-------------------------------------|---------------------------------|
| - <code>input_size</code> | # of expected features in the input | int |
| - <code>hidden_size</code> | # of features in the hidden state | int |
| - <code>num_layers</code> | # of recurrent layers | Default = 1 |
| - <code>Nonlinearity</code> | Non-linearity to use | int or tuple (default = 'tanh') |
| <code>)</code> | | |

Official documentation: <https://pytorch.org/docs/stable/generated/torch.nn.RNN.html>

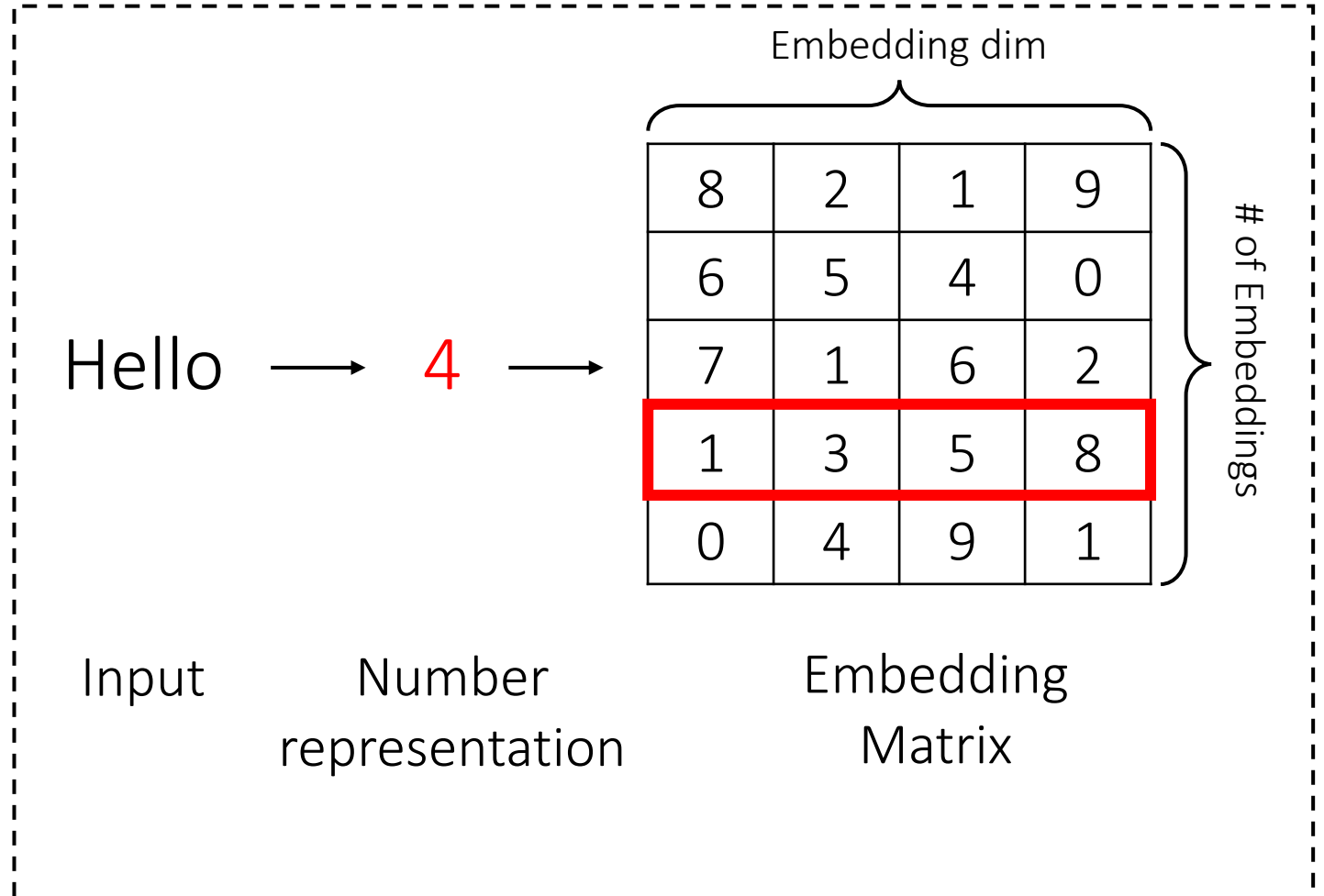
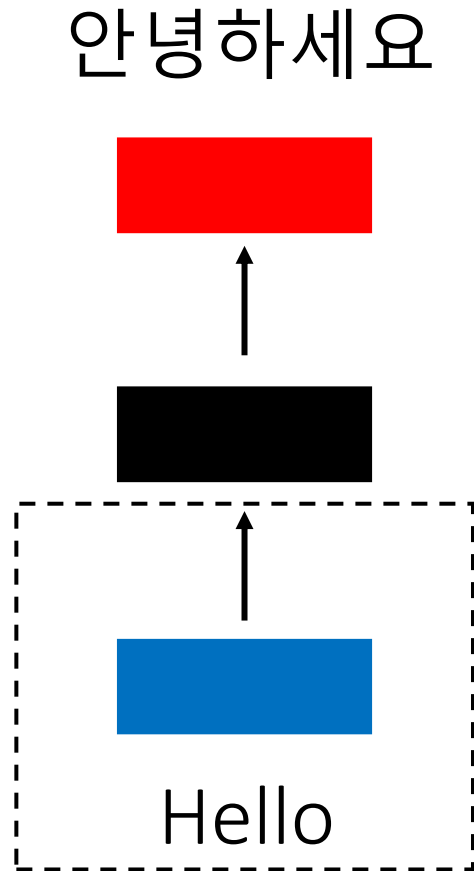


Embedding and Decoder



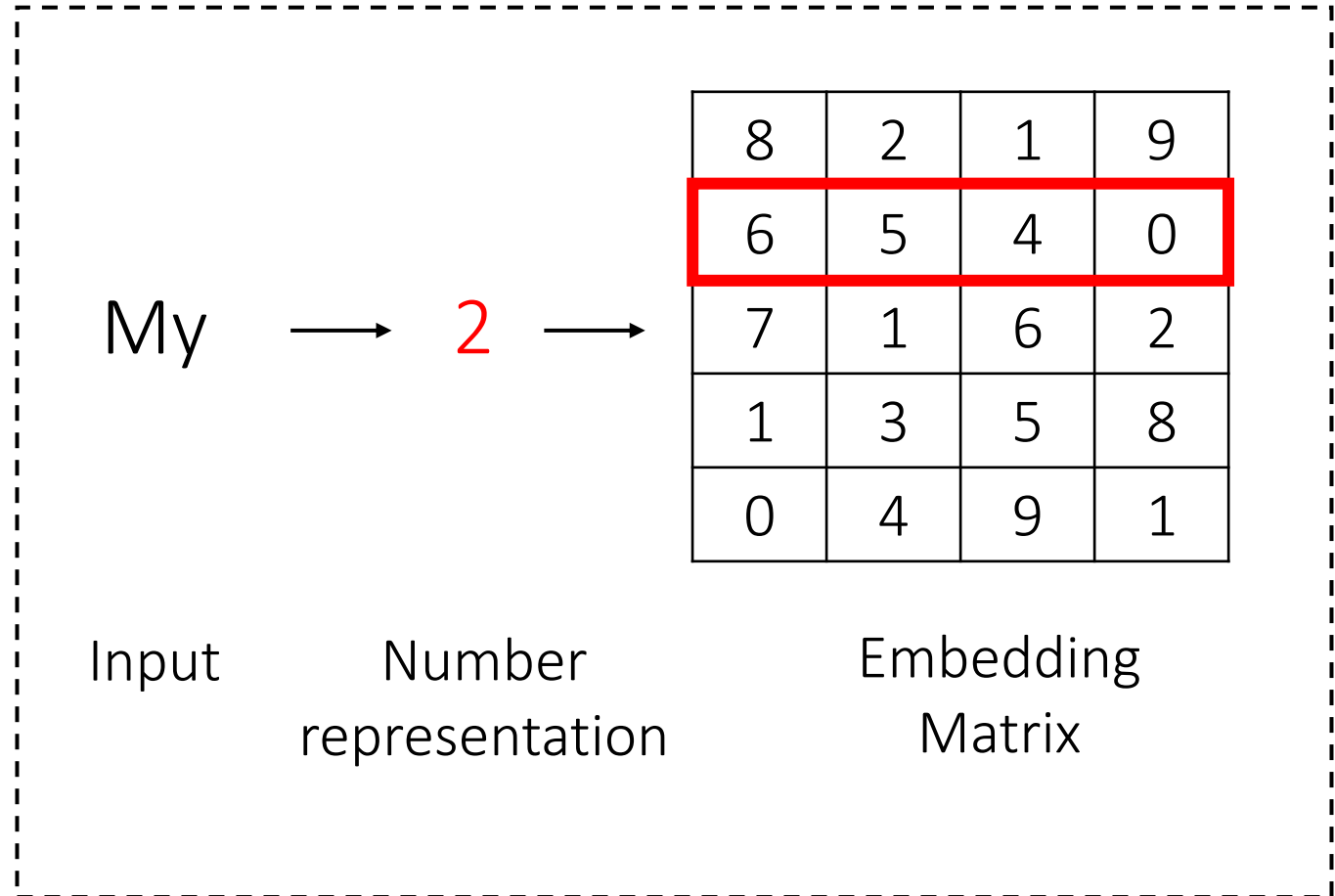
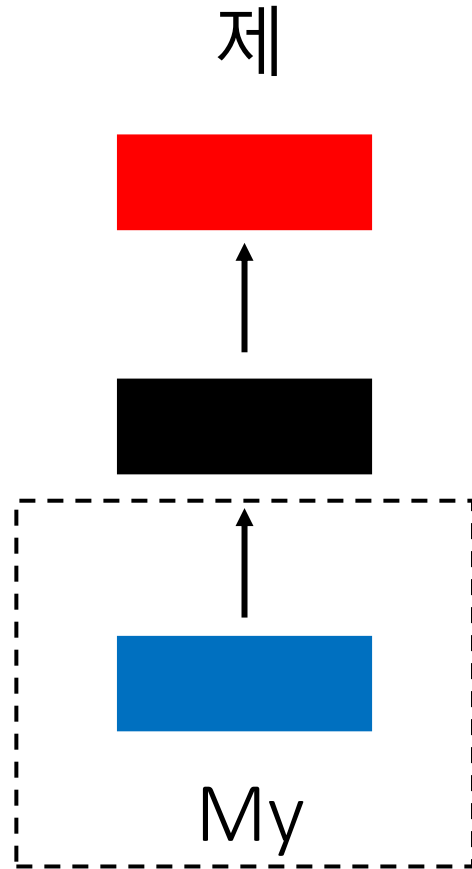


Embedding



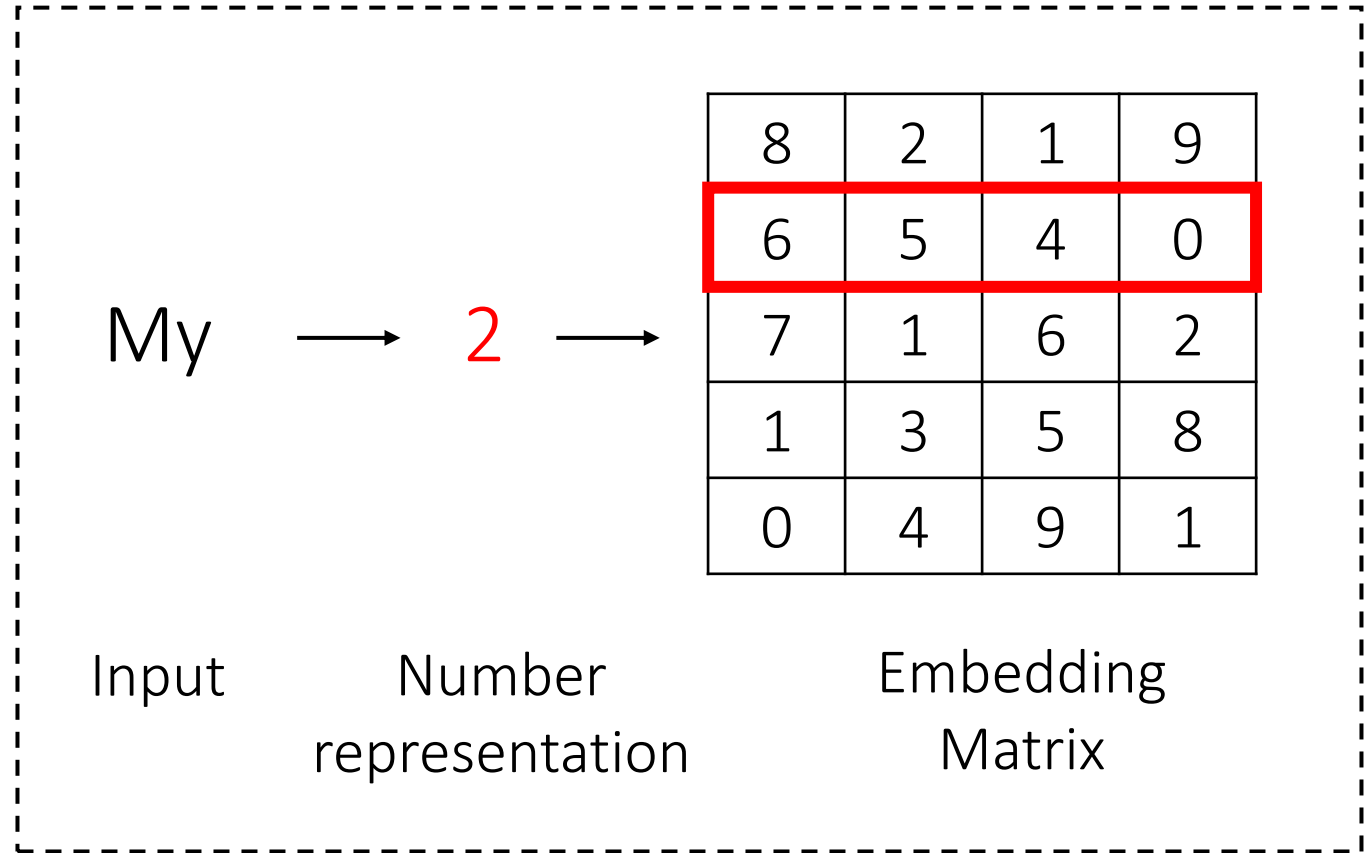
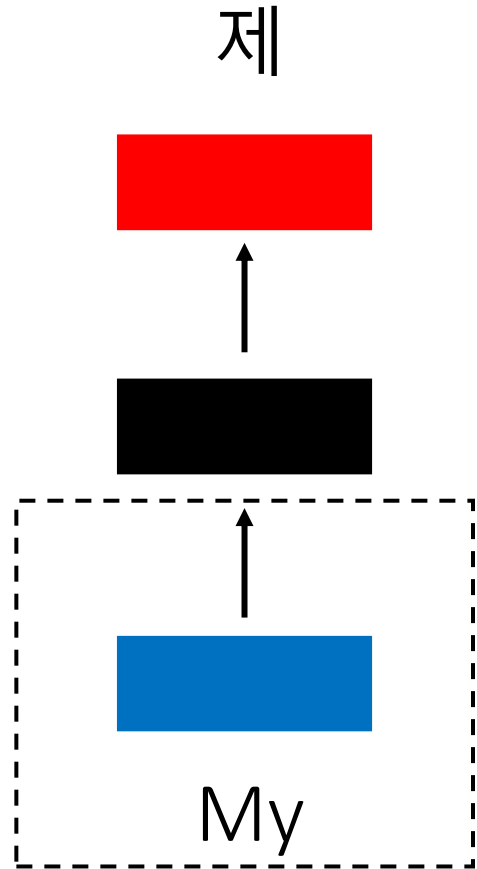


Embedding





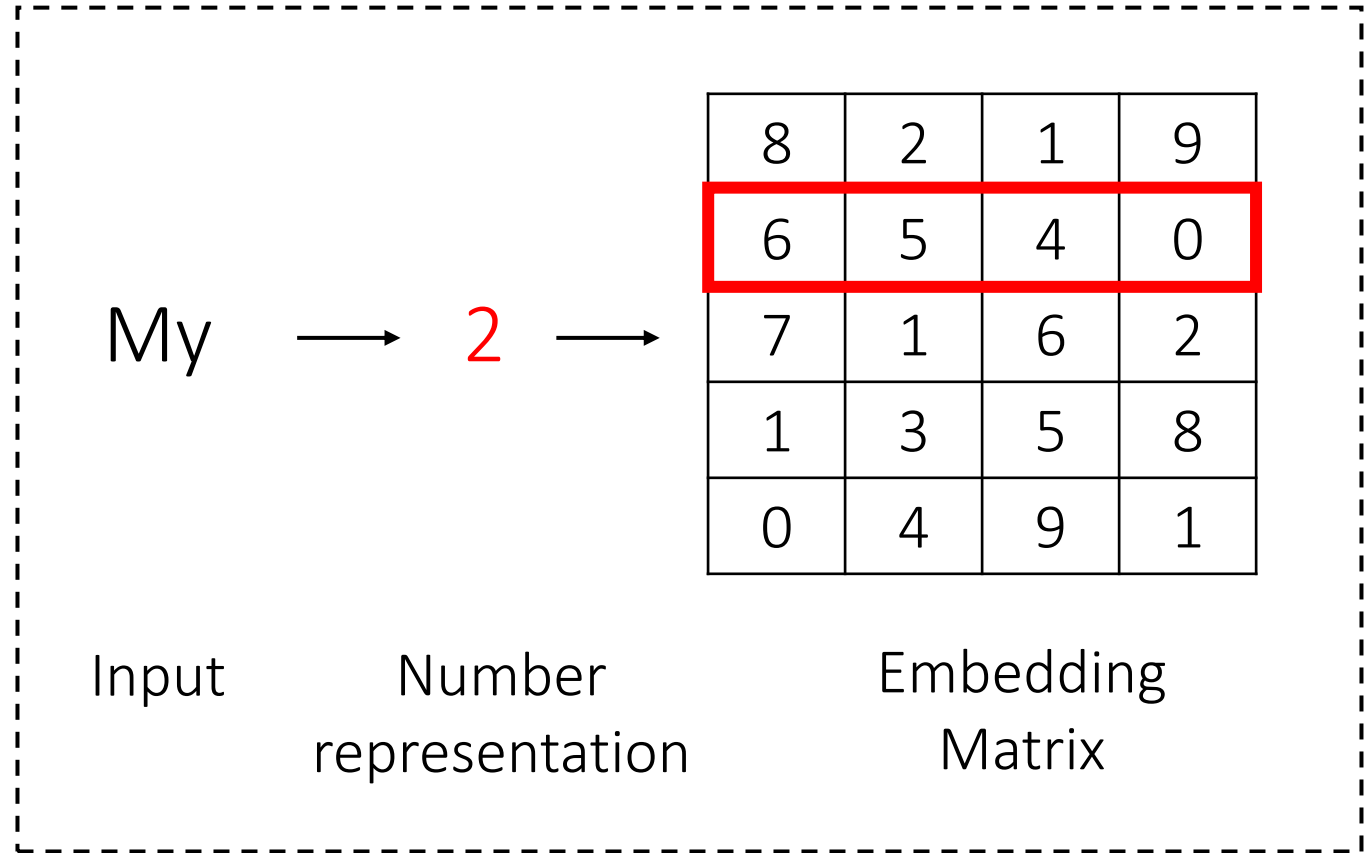
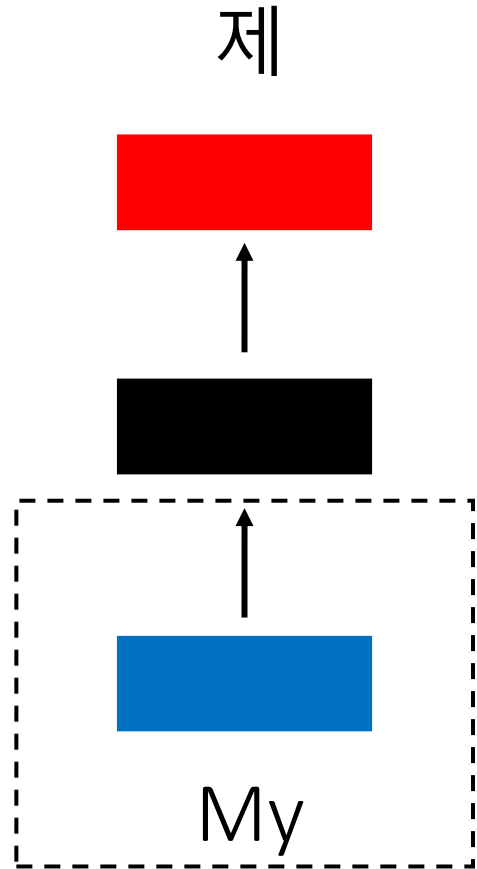
Embedding



Embedding matrix is trainable



Embedding

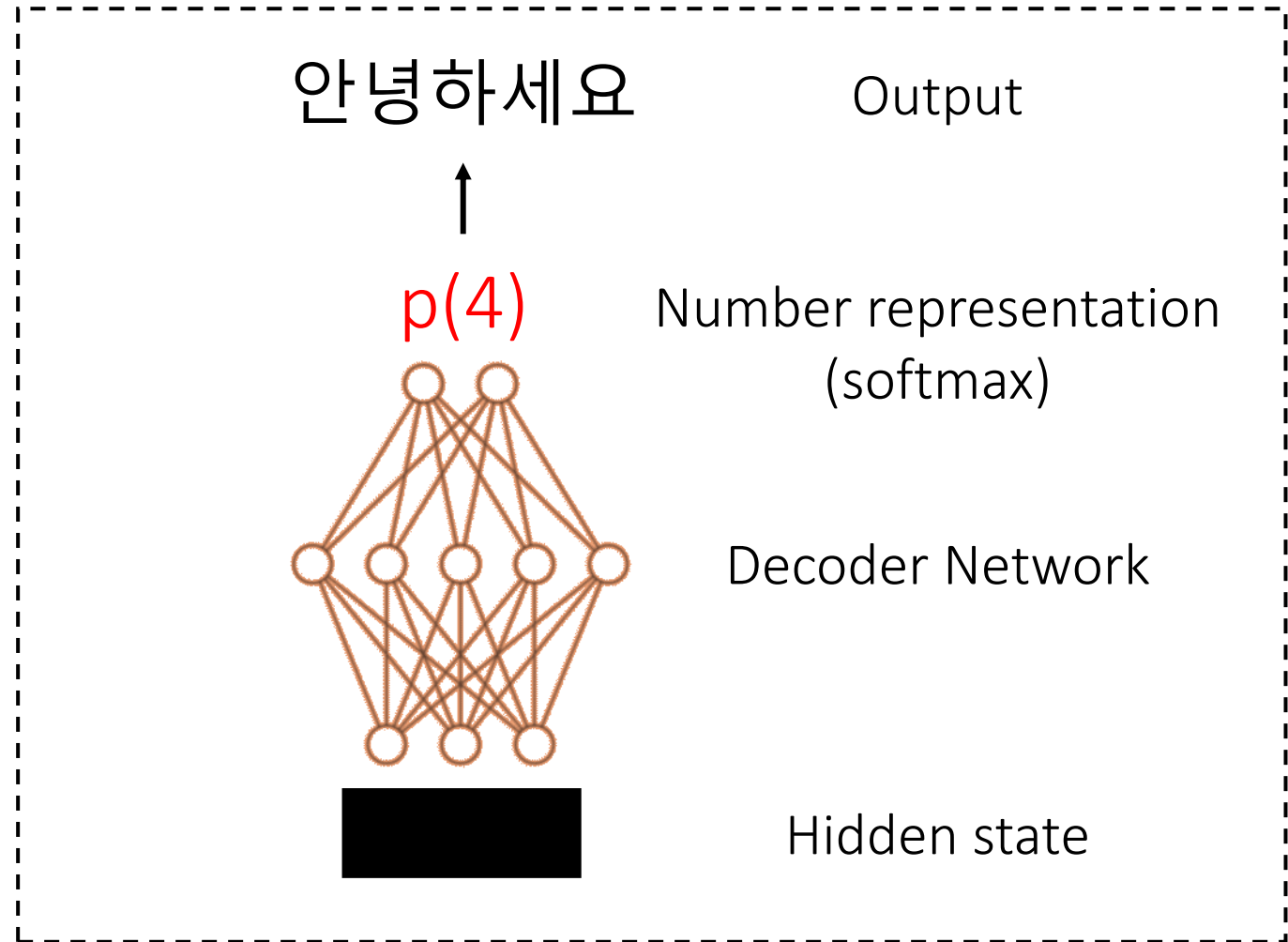
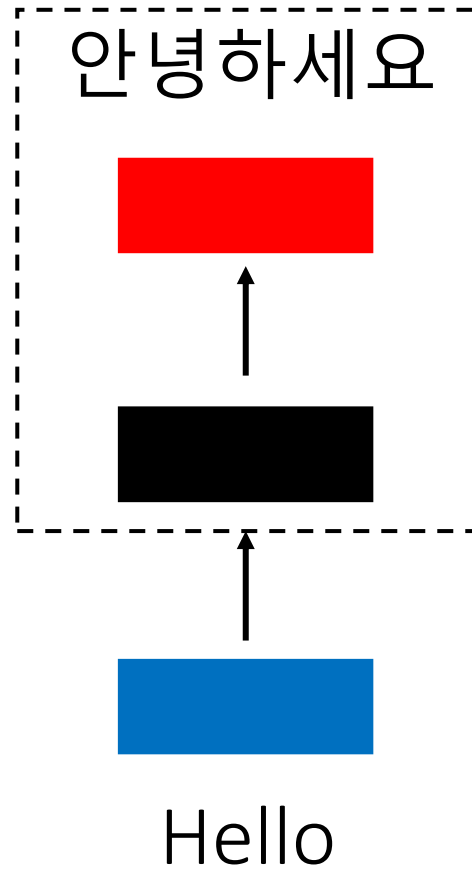


`torch.nn.embedding(num_embeddings, embedding dim)`

<https://pytorch.org/docs/stable/generated/torch.nn.Embedding.html>

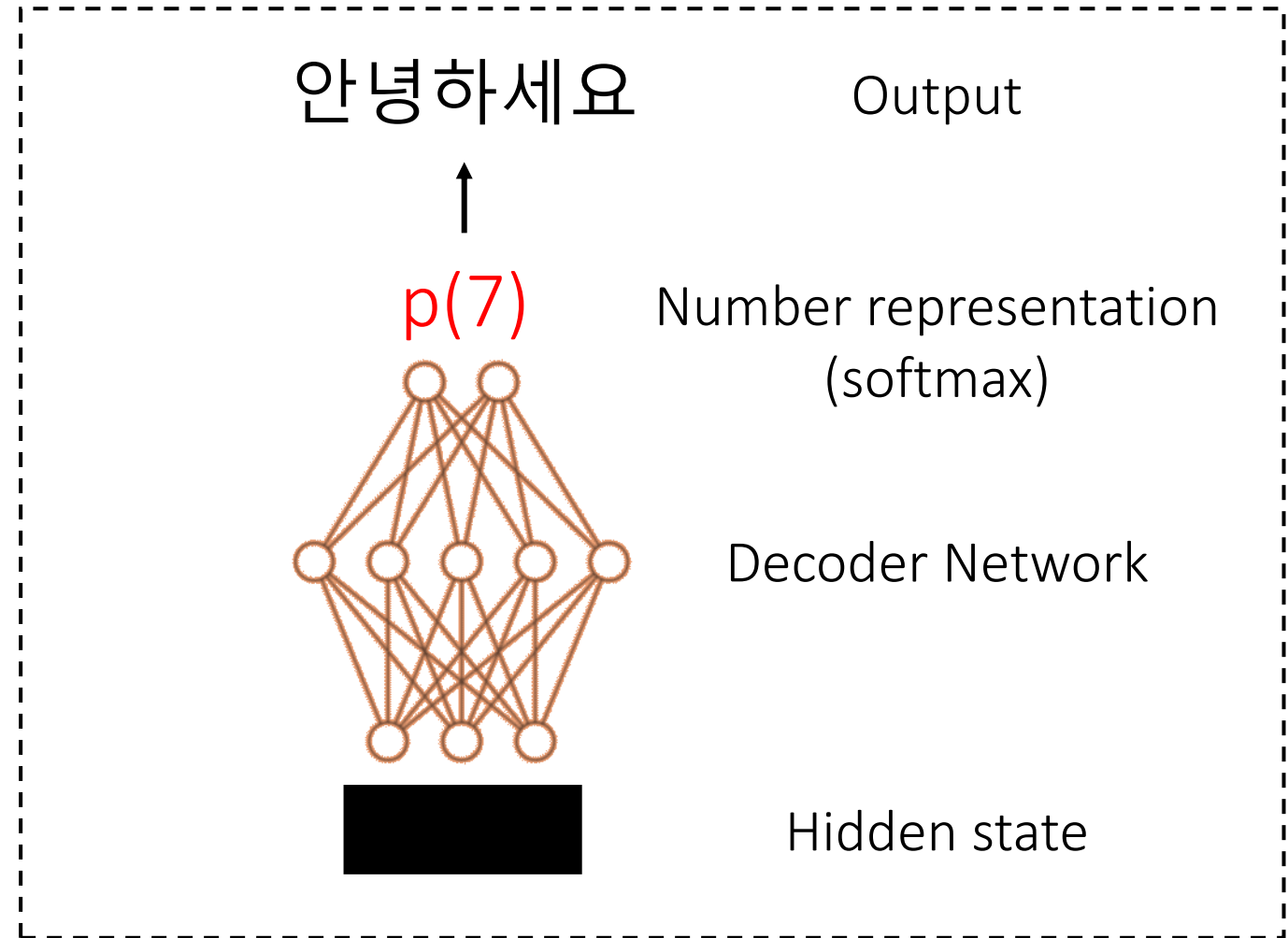
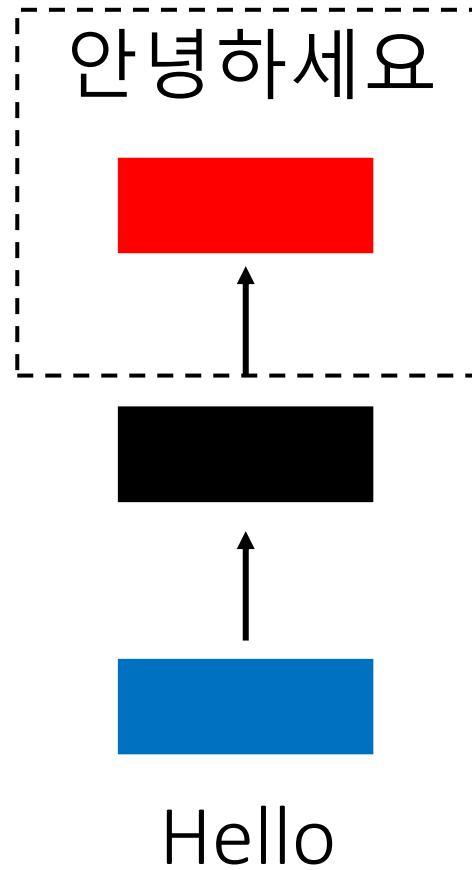


Decoder





Decoder



```
torch.nn.Linear(hidden_size, output_size)
```



TRAINING RNNs

Backpropagation in RNNs

Vanishing/Exploding Gradient Problems

Training RNNs with Teacher Forcing



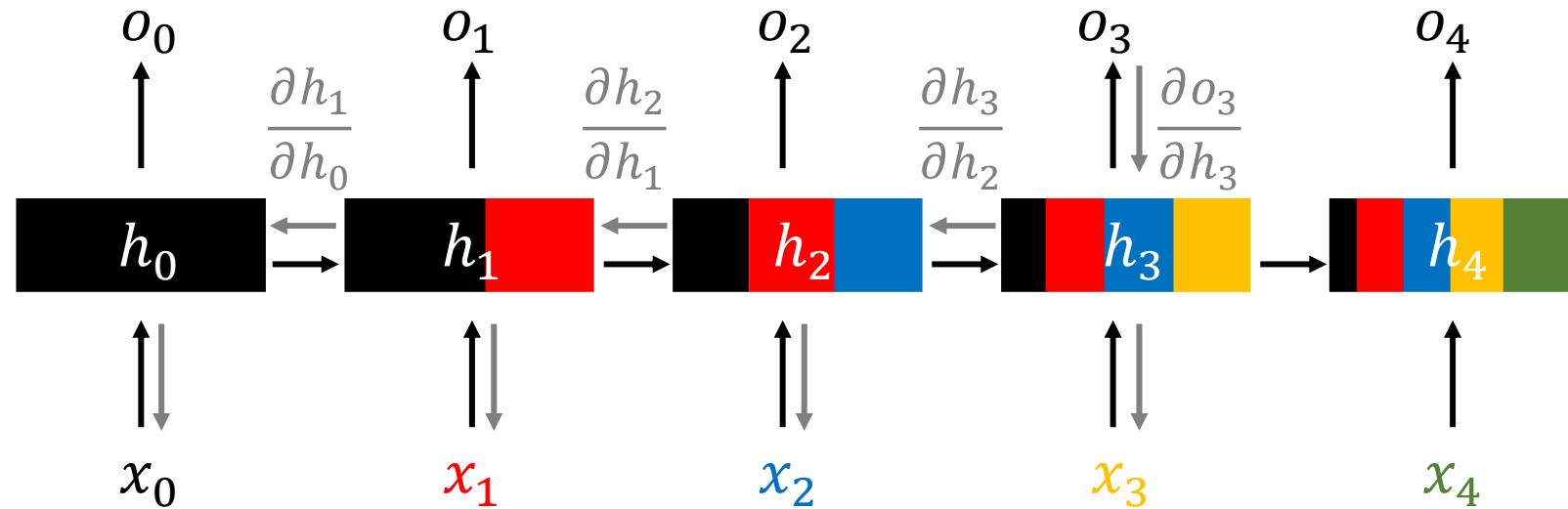
Backpropagation in RNNs

→ Forward
← Backward

output

hidden

input





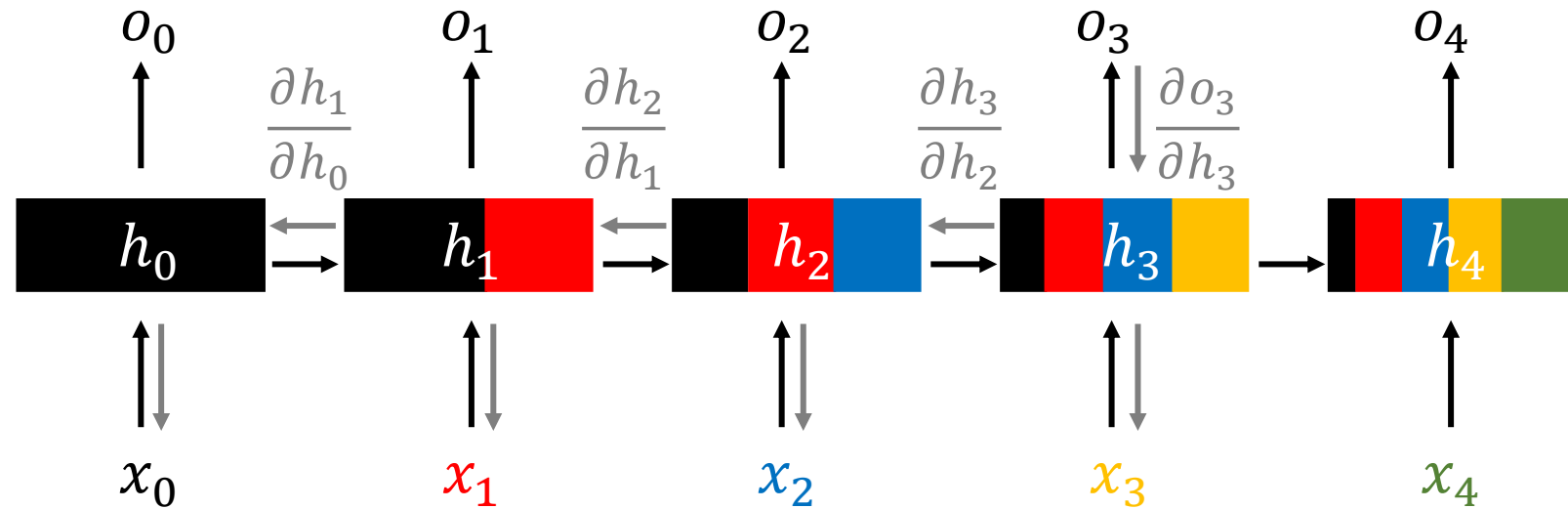
Backpropagation in RNNs

→ Forward
← Backward

output

hidden

input



Backpropagation is performed backward in time

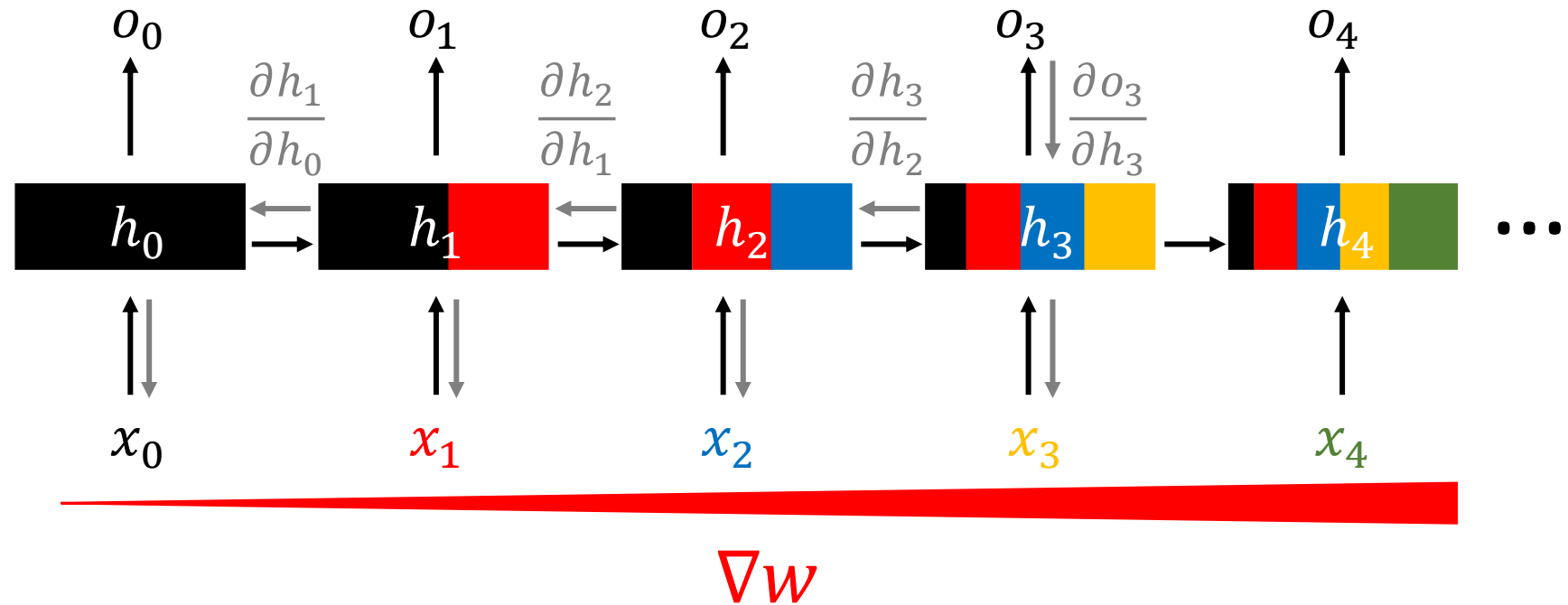
Vanishing and Exploding Gradients

→ Forward
← Backward

output

hidden

input



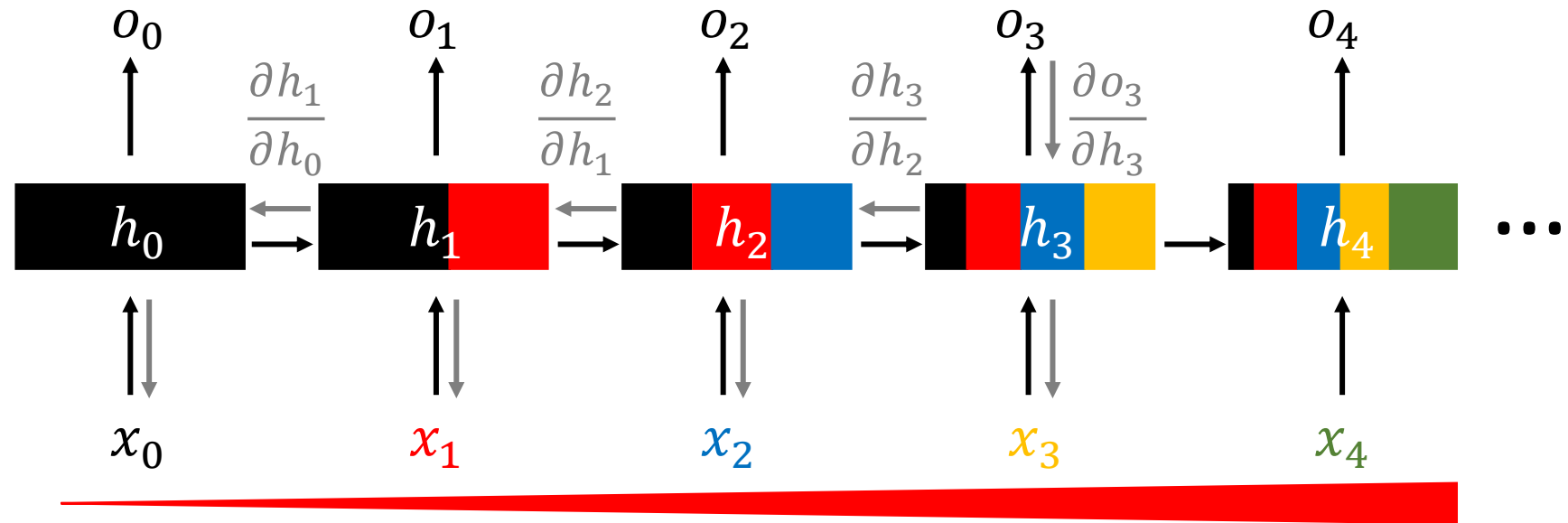
Vanishing and Exploding Gradients

→ Forward
← Backward

output

hidden

input



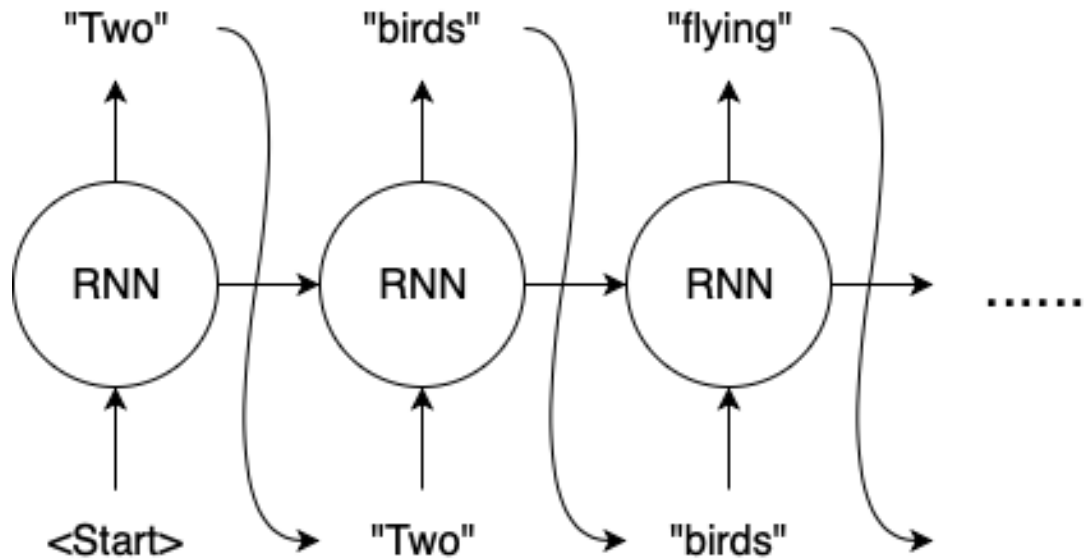
Longer input sequence →
higher risk of Vanishing/Exploding Gradients!



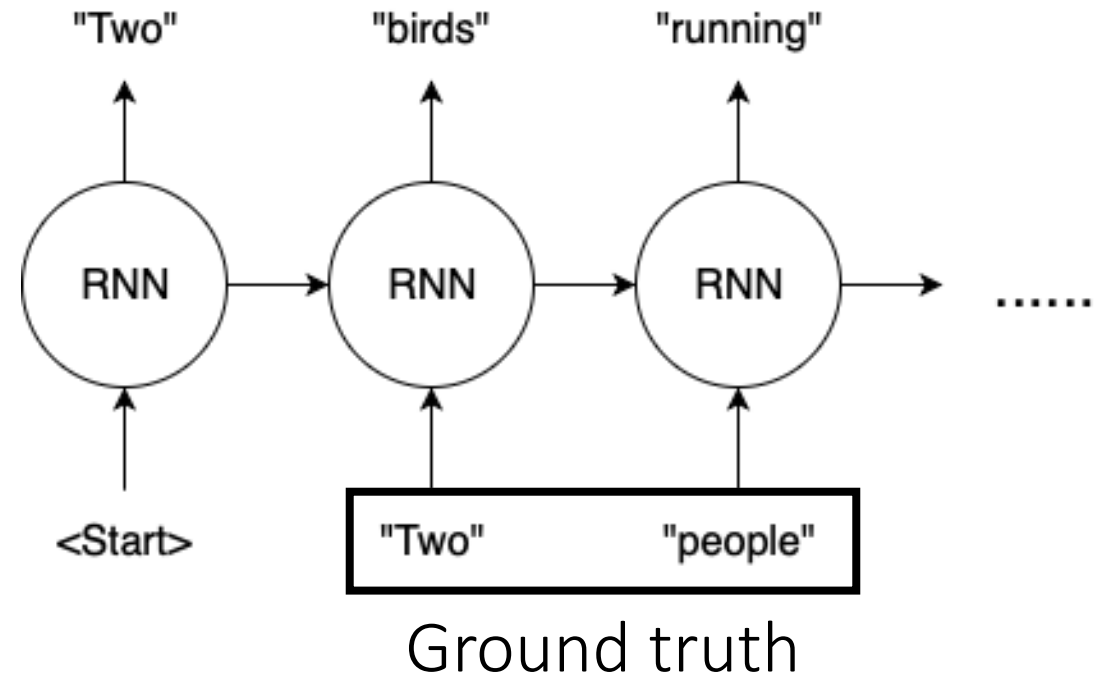
Vanishing and Exploding Gradients

- Use gated RNN architecture e.g., LSTM, GRU (Lab 6)
- ReLU activation as nonlinearity
- Smaller number of sequence
- Smaller learning rate

Training RNN with Teacher Forcing



Without Teacher Forcing



With Teacher Forcing



RNN PROBLEM TYPES

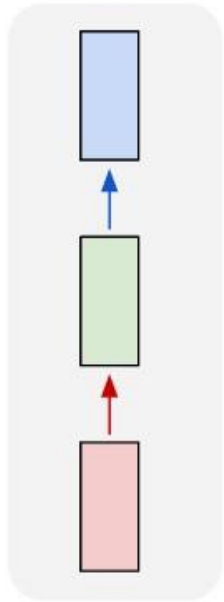
RNN Configurations

RNN Extensions

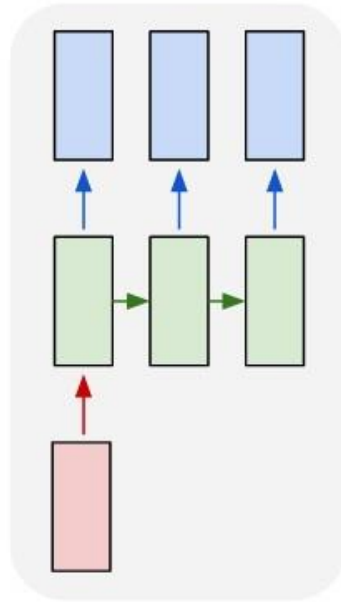


RNN Configurations

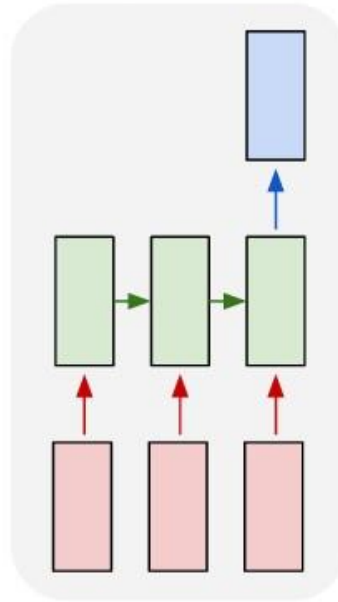
one to one



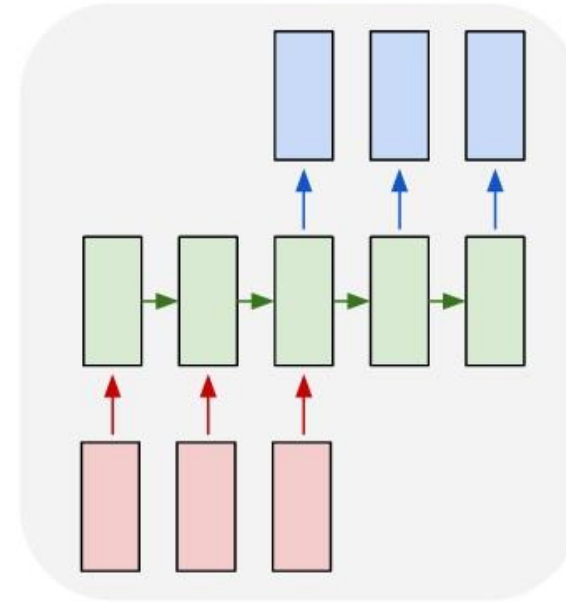
one to many



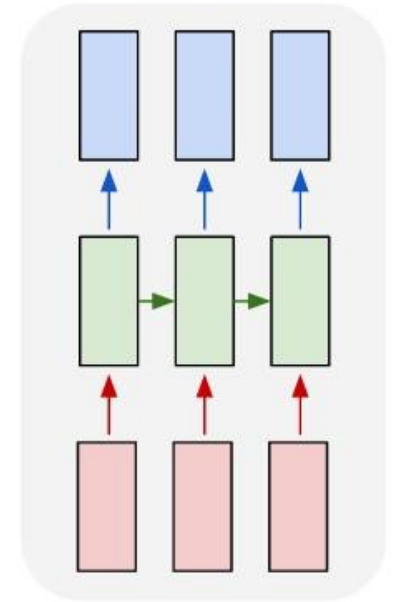
many to one



many to many



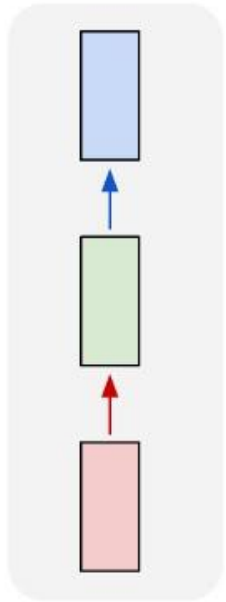
many to many





One to One

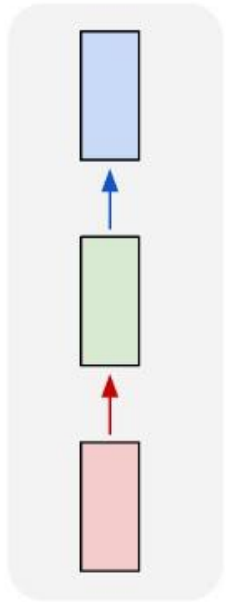
one to one





One to One

one to one

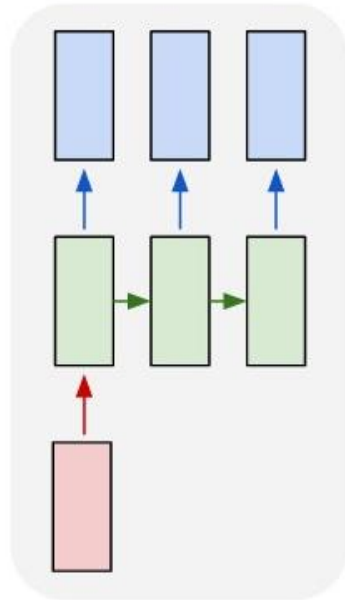


Identical to Feed Forward Network



One to Many

one to many





One to Many

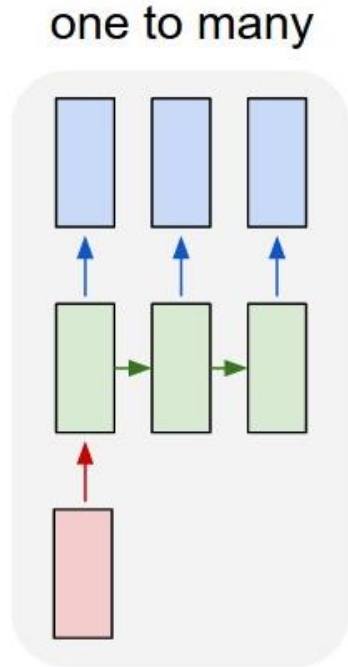
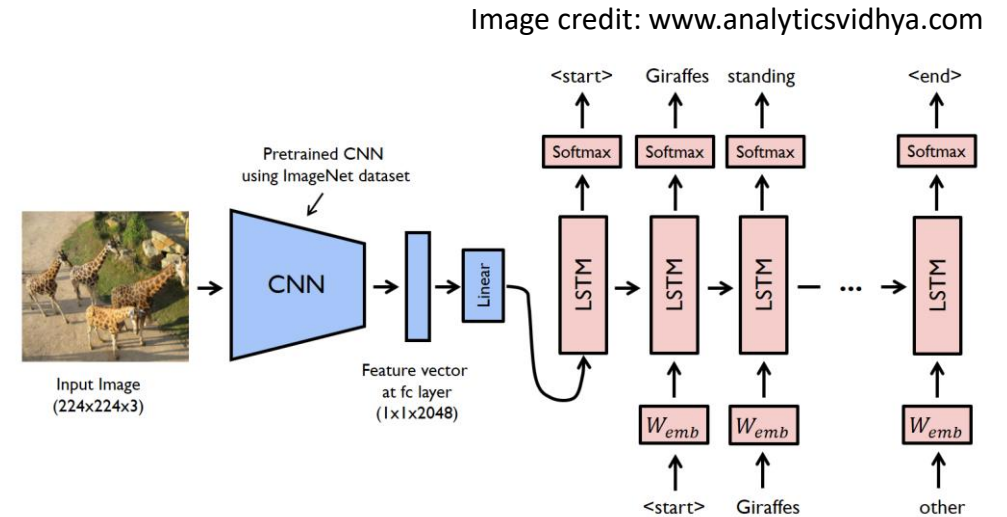
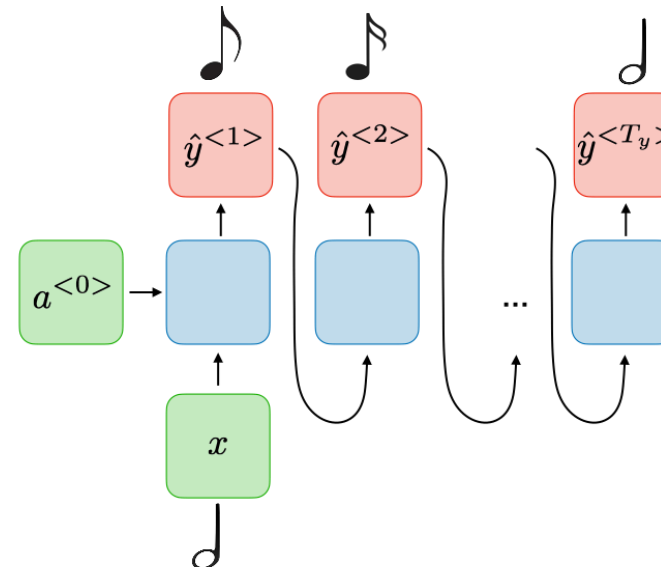


Image captioning

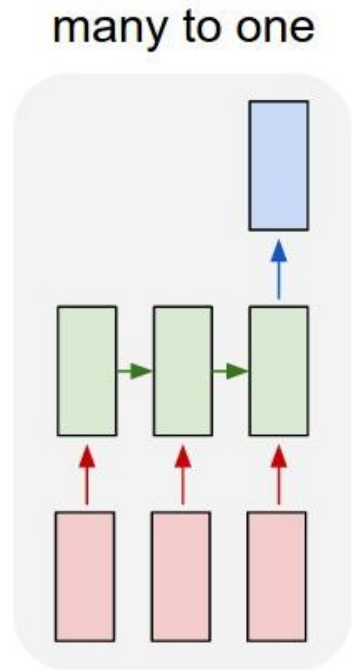


Music generation



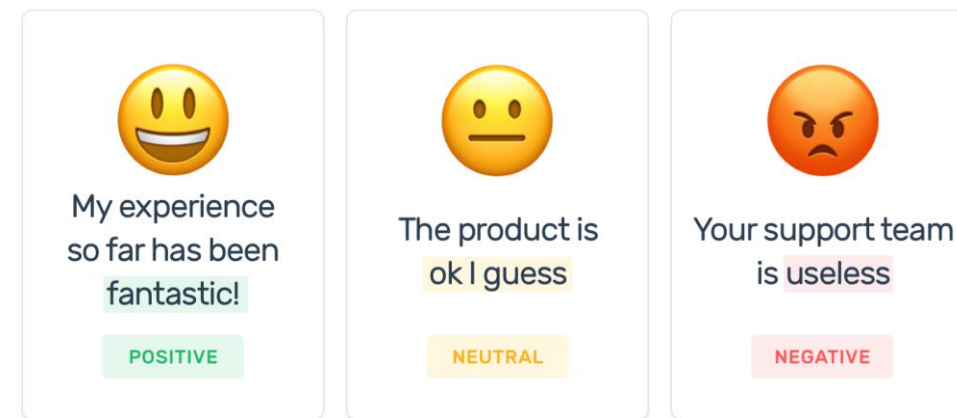
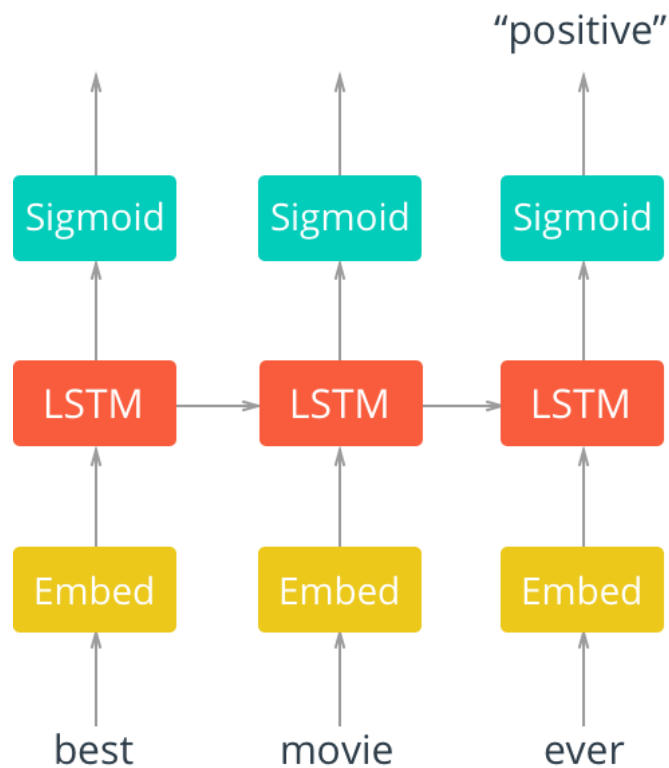
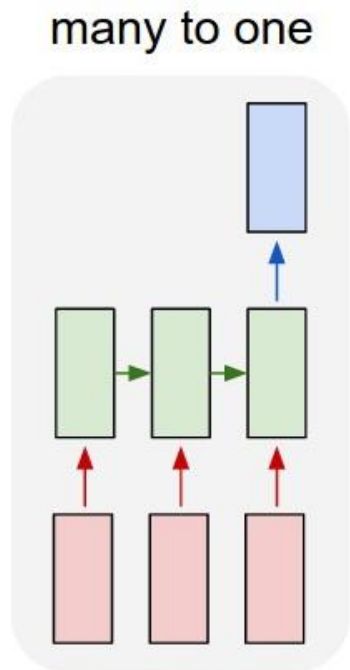


Many to One





Many to One

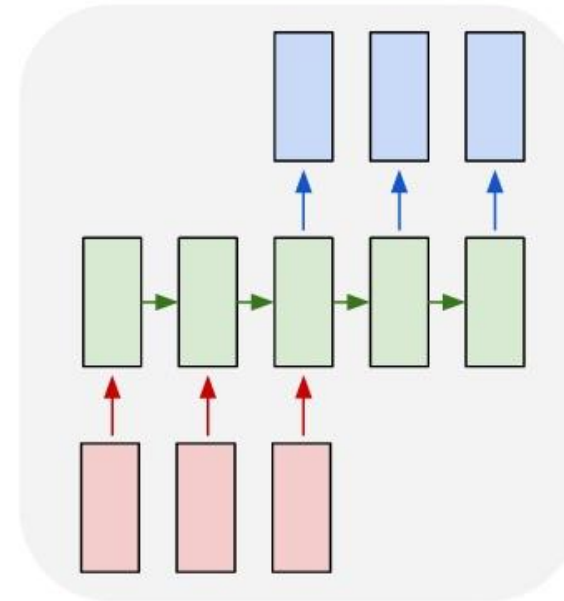


Sentiment Analysis

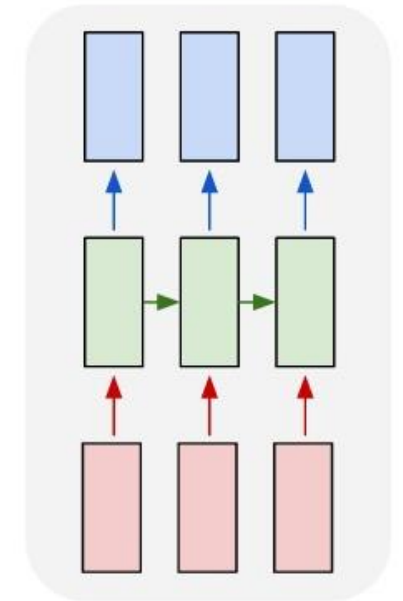


Many to Many

many to many



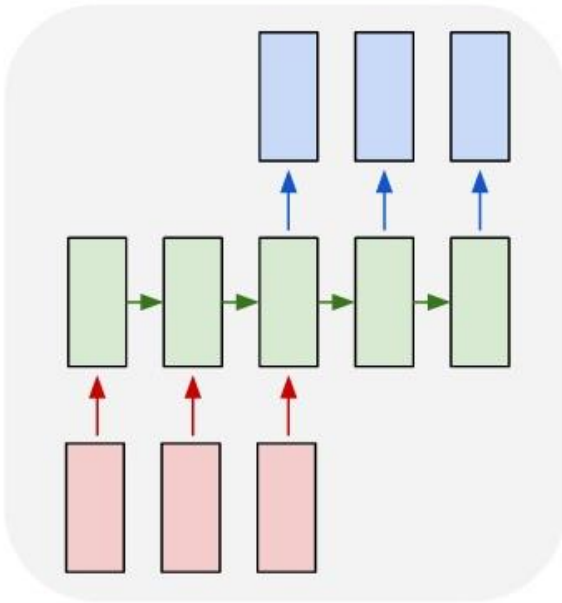
many to many



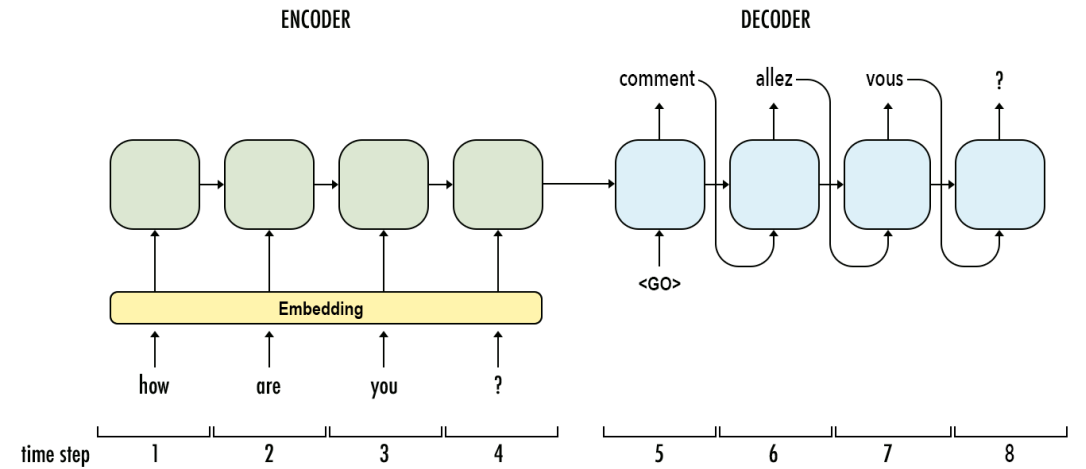
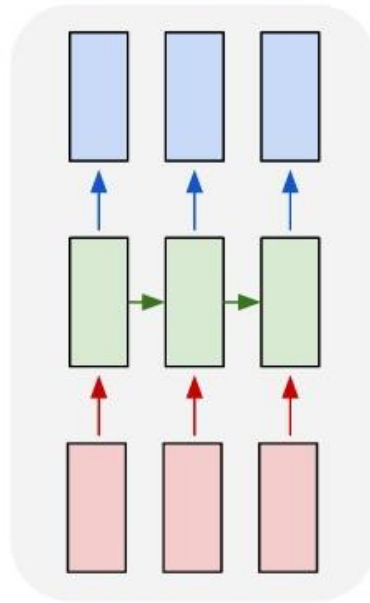


Many to Many

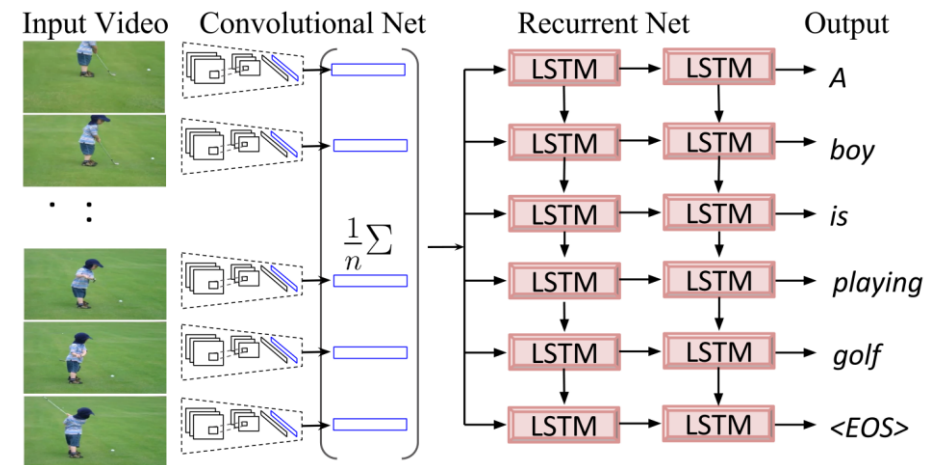
many to many



many to many



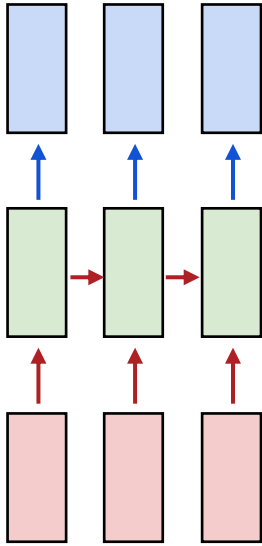
Machine Translation



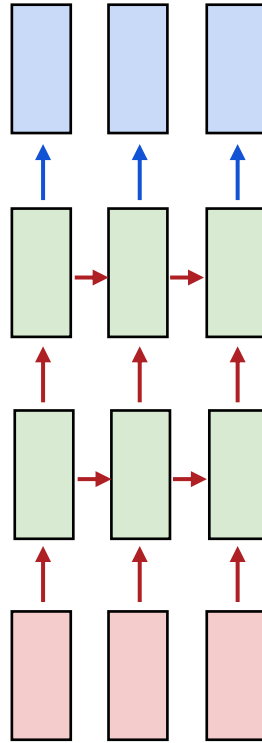
Video Captioning



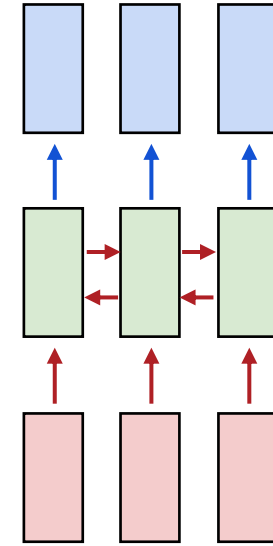
RNN Extensions



Regular RNN



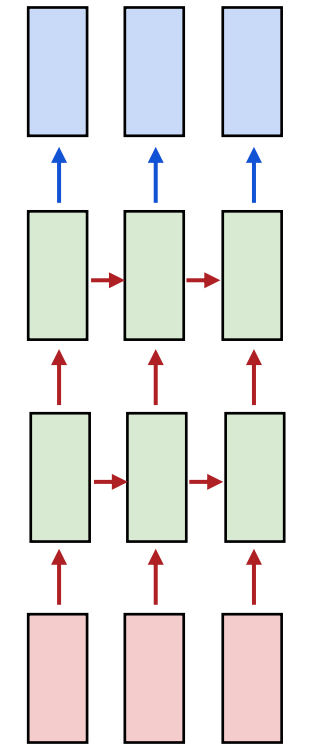
Deep RNN



Bi-directional RNN



Deep RNN



Deep RNN

(+)

Can provide better performance
Often used for complex problems

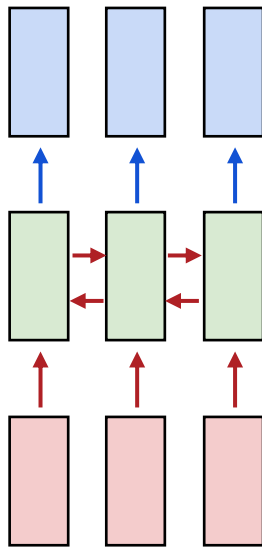
(-)

Potential for overfitting
Longer training time

Implemented as 'num_layers'
parameters in `torch.nn.RNN()`



Bi-directional RNNs



Bi-directional RNN

(+)

Higher performance in Natural Language Processing tasks

Suitable when both left and right contexts are used

(-)

Harder to train than Uni-directional RNN
Not suitable for real-time processing

Implemented as 'bidirectional' parameter in `torch.nn.RNN()`



RNN IMPLEMENTATION IN PYTORCH

Character Level Text Generation using
Shakespeare Dataset



Shakespeare Dataset

First Citizen:
Before we proceed any further, hear me speak.

All:
Speak, speak.

First Citizen:
You are all resolved rather to die than to famish?

All:
Resolved. resolved.

First Citizen:
First, you know Caius Marcius is chief enemy to the people.

All:
We know't, we know't.

First Citizen:
Let us kill him, and we'll have corn at our own price.
Is't a verdict?

All:
No more talking on't; let it be done: away, away!

Second Citizen:
One word, good citizens.

First Citizen:
We are accounted poor citizens, the patricians good.
What authority surfeits on would relieve us: if they
would yield us but the superfluity, while it were
wholesome, we might guess they relieved us humanely;

Full script of “Tragedy of Coriolanus” in .txt format

3801089 characters (including space)

66 unique characters



Prepare Data

```
1 data_size_to_train = 10000
2
3 data = open('shakespeare.txt', 'r').read()[:data_size_to_train]
4 characters = sorted(list(set(data)))
5 data_size, vocab_size = len(data), len(characters)
6
7 print("Data has {} characters, {} unique".format(data_size, vocab_size))
```

Define length of training data to be used for training

Load .txt file and sort unique characters using set()

Print # of total and unique characters

Data has 10000 characters, 57 unique

```
1 character_to_num = { ch:i for i,ch in enumerate(characters) }
2 num_to_character = { i:ch for i,ch in enumerate(characters) }
```

Create dictionaries that map each character to numbers and vice versa

```
1 print(character_to_num)
```

```
{'\n': 0, ' ': 1, '!': 2, '"': 3, ',': 4, '-': 5, '.': 6, ':': 7, ';': 8, '?': 9, 'A': 10, 'B': 11, 'C': 12, 'D': 13, 'E': 14, 'F': 15, 'H': 16, 'I': 17, 'J': 18, 'L': 19, 'M': 20, 'N': 21, 'O': 22, 'P': 23, 'R': 24, 'S': 25, 'T': 26, 'U': 27, 'V': 28, 'W': 29, 'Y': 30, 'a': 31, 'b': 32, 'c': 33, 'd': 34, 'e': 35, 'f': 36, 'g': 37, 'h': 38, 'i': 39, 'j': 40, 'k': 41, 'l': 42, 'm': 43, 'n': 44, 'o': 45, 'p': 46, 'q': 47, 'r': 48, 's': 49, 't': 50, 'u': 51, 'v': 52, 'w': 53, 'x': 54, 'y': 55, 'z': 56}
```



Prepare Data

```
1 data = list(data)
2
3 for i, ch in enumerate(data):
4     data[i] = character_to_num[ch]
```

Convert data into Python list

Map each character in the data to a number

```
1 print(data[:10])
```

First 10 characters of the data

```
[17, 48, 57, 58, 59, 1, 14, 48, 59, 48]
```



Define Model

```
1 class CharRNN(torch.nn.Module):
2
3     def __init__(self, num_embeddings, embedding_dim, input_size, hidden_size, num_layers, output_size):
4
5         super(CharRNN, self).__init__()
6
7         self.embedding = torch.nn.Embedding(num_embeddings, embedding_dim)
8
9         self.rnn = torch.nn.RNN(input_size=input_size, hidden_size=hidden_size,
10                                num_layers=num_layers,
11                                nonlinearity = 'relu')
12
13         self.decoder = torch.nn.Linear(hidden_size, output_size)
14
15     def forward(self, input_seq, hidden_state):
16
17         embedding = self.embedding(input_seq)
18
19         output, hidden_state = self.rnn(embedding, hidden_state)
20
21         output = self.decoder(output)
22
23         return output, hidden_state.detach()
```

Define embedding layer

Define RNN cell

Define decoder layer

Input_seq -> embedding layer

Embedding, hidden_state -> RNN cell

RNN output -> decoder layer -> output

Return both output & hidden state



Select Hyperparameters

```
1 torch.manual_seed(25)
2
3 rnn = CharRNN(num_embeddings = vocab_size, embedding_dim = 100,
4               input_size = 100, hidden_size = 512, num_layers = 3,
5               output_size = vocab_size)
6
7 lr = 0.001
8 epochs = 50
9 training_sequence_len = 50
10 validation_sequence_len = 200
11
12 loss_fn = torch.nn.CrossEntropyLoss()
13 optimizer = torch.optim.Adam(rnn.parameters(), lr=lr)
14
15 rnn
```

Define RNN specifics

- # of Embedding = vocab size
- Embedding dim = 100
- Input_size = 100
- Hidden_size = 512
- Num_layers = 3
- Output_size = vocab size

Define learning rate, epochs, length of training/validation text sequence

Define loss function and optimizer



Identify Tracked Values

```
1 train_loss_list = []
```

Python list to track training loss



Train Model

```
1 data = torch.unsqueeze(torch.tensor(data), dim = 1)
2
3 # Training Loop -----
4
5 for epoch in range(epochs):
6
7     character_loc = np.random.randint(100)
8     iteration = 0
9     hidden_state = None
10
11     while character_loc + training_sequence_len + 1 < data_size:
12
13         input_seq = data[character_loc : character_loc + training_sequence_len]
14         target_seq = data[character_loc + 1 : character_loc + training_sequence_
15
16         output, hidden_state = rnn(input_seq, hidden_state)
17
18         loss = loss_fn(torch.squeeze(output), torch.squeeze(target_seq))
19
20         train_loss_list.append(loss.item())
21
22         optimizer.zero_grad()
23         loss.backward()
24         optimizer.step()
25
26         character_loc += training_sequence_len
27
28         iteration += 1
29
30 print("Averaged Training Loss for Epoch ", epoch, ": ", np.mean(train_loss_list[-iteration:]))
31
```

Convert data into torch tensor in vertical orientation
(data_length, 1)

For each epoch, randomly select a starting character from
first 100 characters

input_seq =
character location -> training sequence size
target_seq =
character location + 1 -> training sequence size + 1

Retrieve output & hidden state from RNN cell and
compute loss

Save training loss

Backpropagation in time & update network

Update the character location

Update the iteration count

Print the averaged training loss throughout an epoch



Train Model

```
32 # Sample and generate a text sequence after every epoch -----
33
34 character_loc = 0
35 hidden_state = None
36
37 rand_index = np.random.randint(data_size-1)
38 input_seq = data[rand_index : rand_index+1]
39
40 print("-----")
41 with torch.no_grad():
42
43     while character_loc < validation_sequence_len:
44
45         output, hidden_state = rnn(input_seq, hidden_state)
46
47         output = torch.nn.functional.softmax(torch.squeeze(output), dim=0)
48         character_distribution = torch.distributions.Categorical(output)
49         character_num = character_distribution.sample()
50
51         print(num_to_character[character_num.item()], end='')
52
53         input_seq[0][0] = character_num.item()
54
55         character_loc += 1
56
57 print("\n-----")
```

Initialize character location and hidden state for validation

Pick a random character from the dataset as an initial input to RNN

Generate new output by using the previous RNN output as an input

Convert the output into character number via sampling from the decoder layer output (probability distribution of characters)

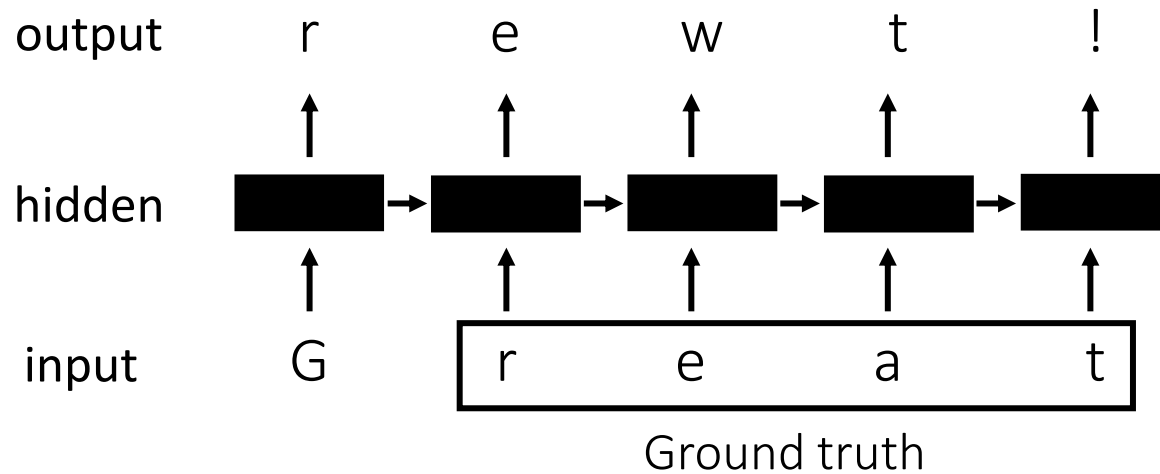
Print the actual character from the number

New input_seq is the output character we just generated

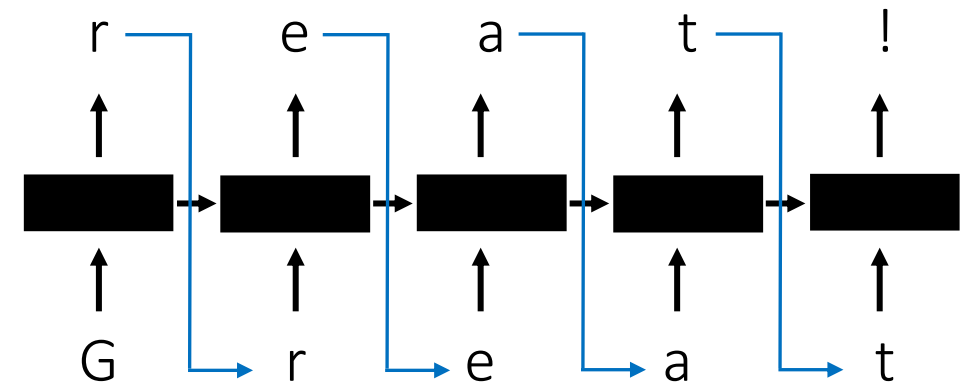
Update the character location



Train Model



During Training
(Teacher Forcing)



During Validation
(RNN output is used as next input)



Train Model

Averaged Training Loss for Epoch 0 : 2.7497982840345365

cous zend vous leaogkal.

MENUNUNENNNNNNENEUNNENNNNUENNNNNNUNNUNUNNNNENENENNNRNRSESNUNENNNUNENN
NNNANN

Suwirs touy to
Ther'krn;

MNENUNUNN
NNNNNNINNNNNNENYNNS:
NENUSNSNThat balt,
Theoud cogvrifing of

Generated text sequence after 1 epoch

Averaged Training Loss for Epoch 49 : 0.26021135710741405

the Capitol; who's fide our trumpetersvile in awe, which else
Would feed on one another? What's their abundance; our
seike.

it takes, cracking ten thus--
For, look o' the moon,
Shouting their emulatio

Generated text sequence after 50 epoch

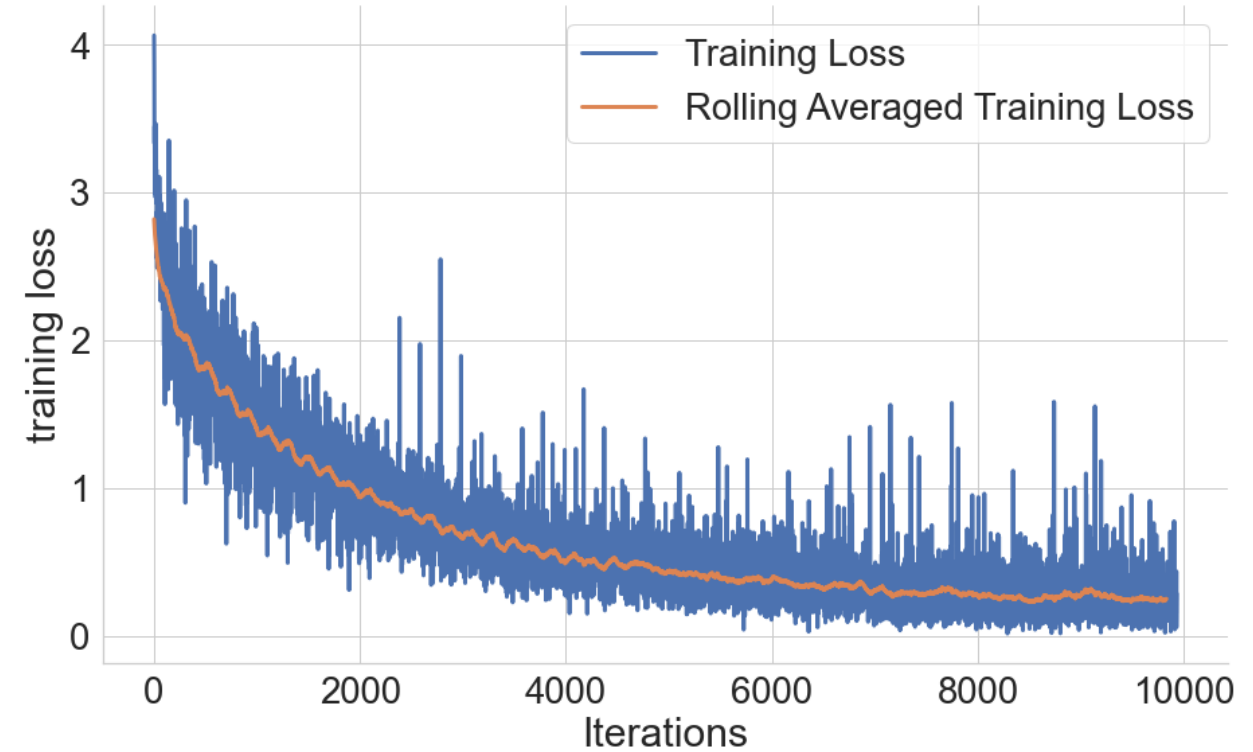


Validate & Evaluate Model

```
1 import seaborn as sns
2
3 sns.set(style = 'whitegrid', font_scale = 2.5)

1 plt.figure(figsize = (15, 9))
2
3 plt.plot(train_loss_list, linewidth = 3, label = 'Training Loss')
4 plt.plot(np.convolve(train_loss_list, np.ones(100), 'valid') / 100,
5          linewidth = 3, label = 'Rolling Averaged Training Loss')
6 plt.ylabel("training loss")
7 plt.xlabel("Iterations")
8 plt.legend()
9 sns.despine()
```

Plot the training loss + rolling average training loss after training





LAB 5 ASSIGNMENT:

Create Arthur Conan Doyle AI with RNN



Sherlock Holmes Dataset

PART I

(Being a reprint from the reminiscences of
John H. Watson, M.D.,
late of the Army Medical Department.)

CHAPTER I Mr. Sherlock Holmes

In the year 1878 I took my degree of Doctor of Medicine of the University of London, and proceeded to Netley to go through the course prescribed for surgeons in the army. Having completed my studies there, I was duly attached to the Fifth Northumberland Fusiliers as Assistant Surgeon. The regiment was stationed in India at the time, and before I could join it, the second Afghan war had broken out. On landing at Bombay, I learned that my corps had advanced through the passes, and was already deep in the enemy's country. I followed, however, with many other officers who were in the same situation as myself, and succeeded in reaching Candahar in safety, where I found my regiment, and at once entered upon my new duties.

The campaign brought honours and promotion to many, but for me it had nothing but misfortune and disaster. I was removed from my brigade and attached to the Berkshires, with whom I served at the fatal battle of Maiwand. There I was struck on the shoulder by a Jezail bullet, which shattered the bone and grazed the subclavian artery. I should have fallen into the hands of the murderous Ghazis had it not been for the devotion and courage shown by Murray, my orderly, who threw me across a pack-horse, and succeeded in bringing me safely to the British lines.

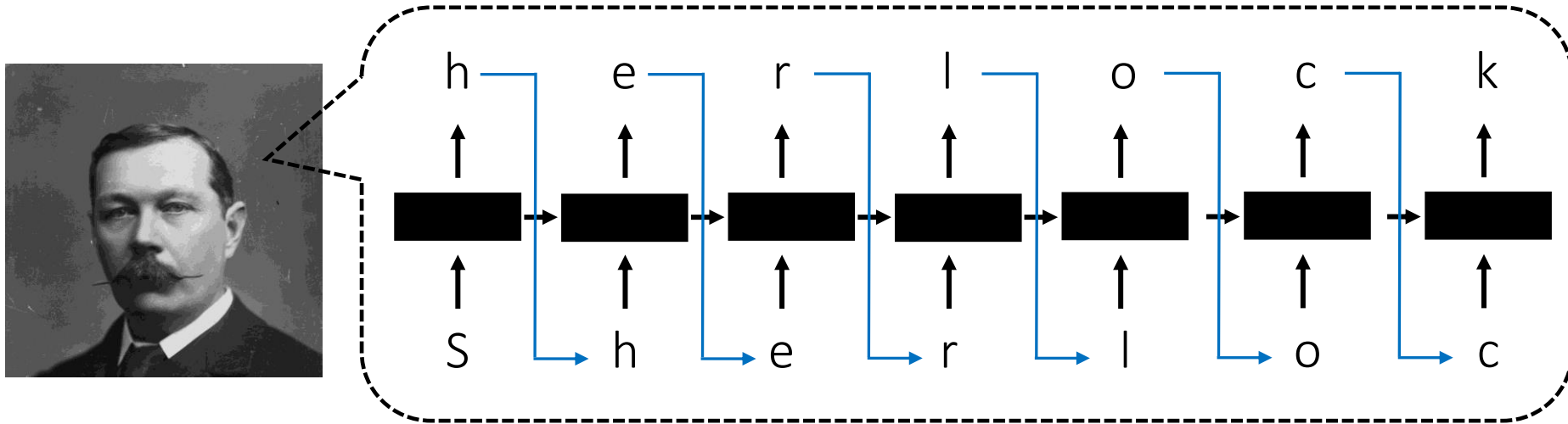
Full collection of Sherlock Holmes series

3011055 total characters (including space)

102 unique characters



Create Arthur Conan Doyle AI using RNN



In this exercise, you will use RNN to generate **Sherlock Holmes style sequence of texts**.

Prior to training, you can decide the **training size** you want to use for training.
(e.g., first 10k characters, 100k characters, etc)

Design your own RNN architecture with your choice of embedding dimension, hidden state size, number of RNN layers, nonlinearity (tanh or ReLU) and training sequence size.

After training your RNN, **print a validation text sequence** that most closely resembles Sherlock Holmes style in your opinion & plot the training curve to confirm the RNN successfully trained.



Tips for Training Your RNN

First things to decide

- Training data size (# of characters)
- Embedding dimension & RNN input
- RNN hidden size
- Activation function (ReLU, Tanh)
- Decoder output size
- Learning rate
- Optimizer
- Number of training epochs
- Training input sequence length

Additional tips

- If you get 'nan' errors while training -> your training is unstable -> decrease lr or training input sequence length
- With ReLU you might be able to process longer sequence
- Choose your training data size according to your machine spec
- Higher 'num_layers' might give you better performance but longer training.