RNN

with PYTORCH

CONTENTS

Before RNN

One hot encoding

RNN intro

WHY and HOW

RNN basics

RNN char sequence

RNN long sequence

ONE HOT ENCODING

Encoding categorical variables to number type

Difference between encoding by labeling and by one hot encoding : identical weight on each variables

Label Encoding

Food Name	Categorical #	Calories
Apple	1	95
Chicken	2	231
Broccoli	3	50

→

One Hot Encoding

Apple	Chicken	Broccoli	Calories
1	0	0	95
0	1	0	231
0	0	1	50

Source : https://medium.com/@john_analyst/%EB%8D%B0%EC%9D%B4%ED%84%B0-%EC%A0%84%EC%B2%98%EB%A6%AC-%EB%A0%88%EC%9D%B4%EB%B8%94-%EC%9D%B8%EC%BD%94%EB%94%A9%EA%B3%BC-%EC%9B%90%ED%95%AB-%EC%9D%B8%EC%BD%94%EB%94%A9-f0220df21df1

WHY rnn and HOW

Good on sequential data like words, sentence, time series…

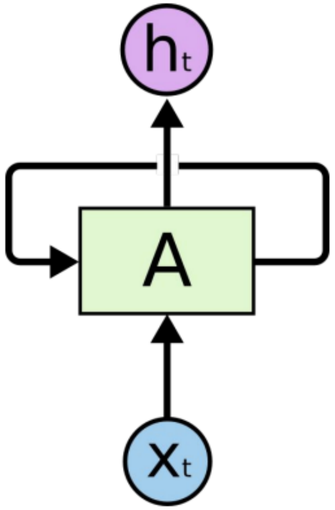
Are words and sentences sequential data?

Of course, think about the word “hello”. ‘e’ comes after ‘h’. To place ‘e’ at the right place, you should know which alphabet is in front of.

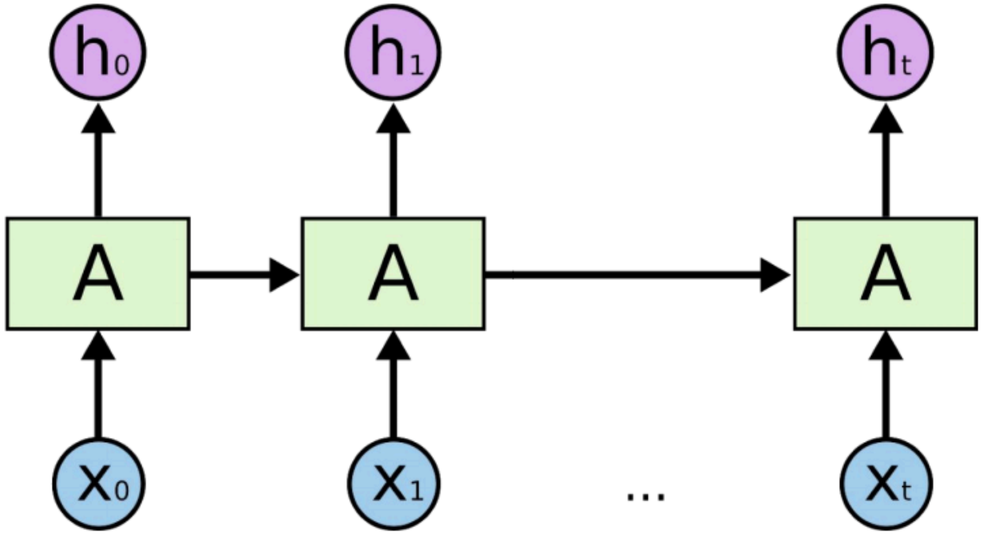
WHY rnn and HOW

Basically, RNN structure is like [picture A].

Unfolded? [picture B].

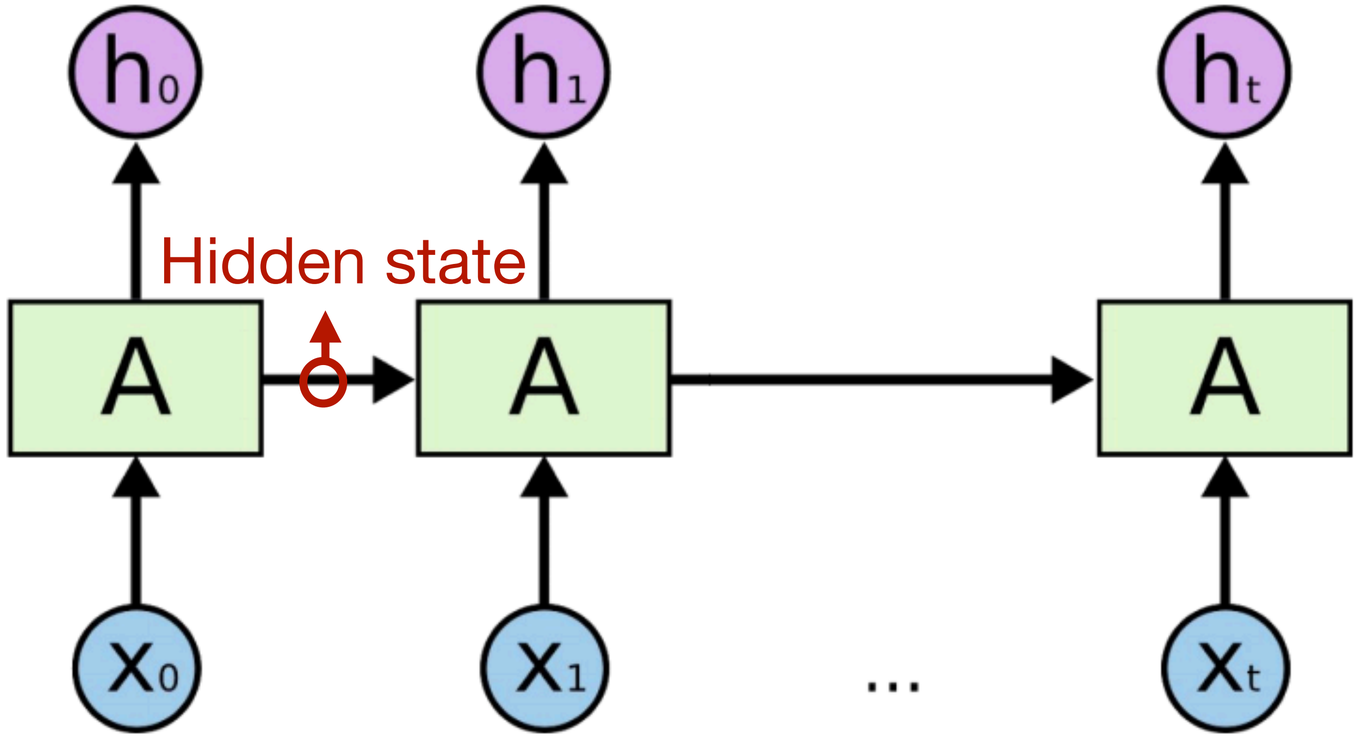


[picture A]



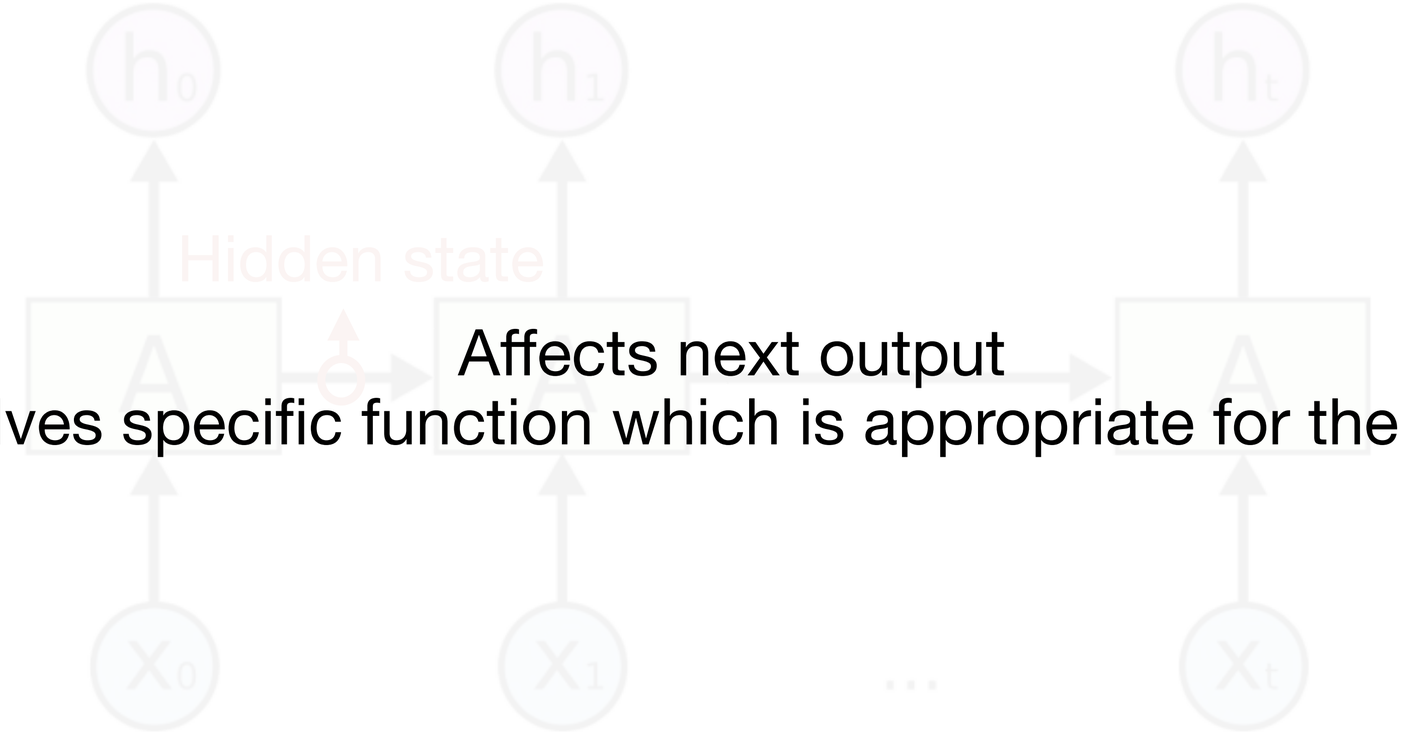
[picture B]

WHY rnn and HOW



[picture B]

WHY rnn and HOW



[picture B]

Affects next output
it solves specific function which is appropriate for the model

Basic RNN

Simple rnn

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

outputs, _status = rnn(input_data)
```

Basic RNN

Simple rnn

rnn = torch.nn.RNN(input_size, hidden_size) : declaring ‘A’, the function

outputs, _status = rnn(input_data) : results from solving input datas

Input datas’ shape should be 3-dimension

hidden_size is same as output_size

Solving char sequence in RNN

Represent char by index, and THEN one hot encode

```
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]]
```

Solving long sequence in RNN

How can we train long sentences?

By making chunks!

```
for i in range(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])
    y_data.append([char_dic[c] for c in y_str])
```

THANK Y  U