AI VIETNAM
All-in-One Course

RNNs for Sequence and Time-series Data

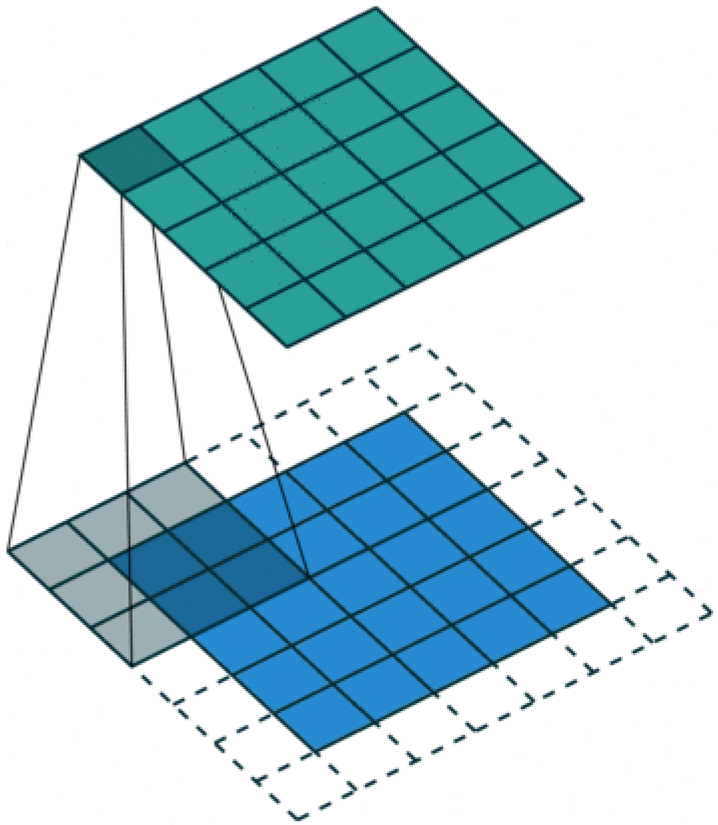
Quang-Vinh Dinh
Ph.D. in Computer Science

Year 2023

Outline

- Dealing with Text
- Constructing RNN
- RNN Examples for Text
- RNN Examples for Time-series Data
- PyTorch Implementation

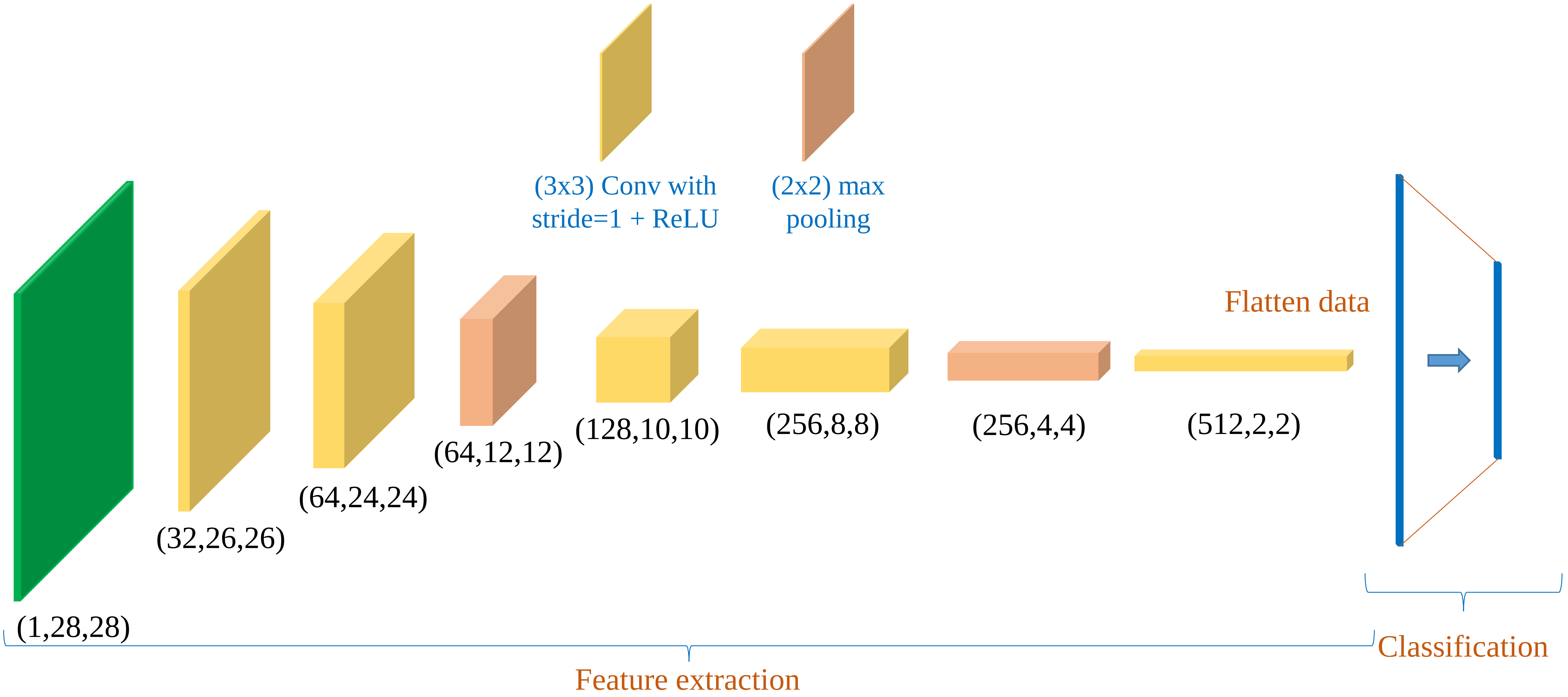
Image Data




airplane
automobile
bird
cat
deer
dog
frog
horse
ship
truck

Image Data

❖ Example



Text Classification

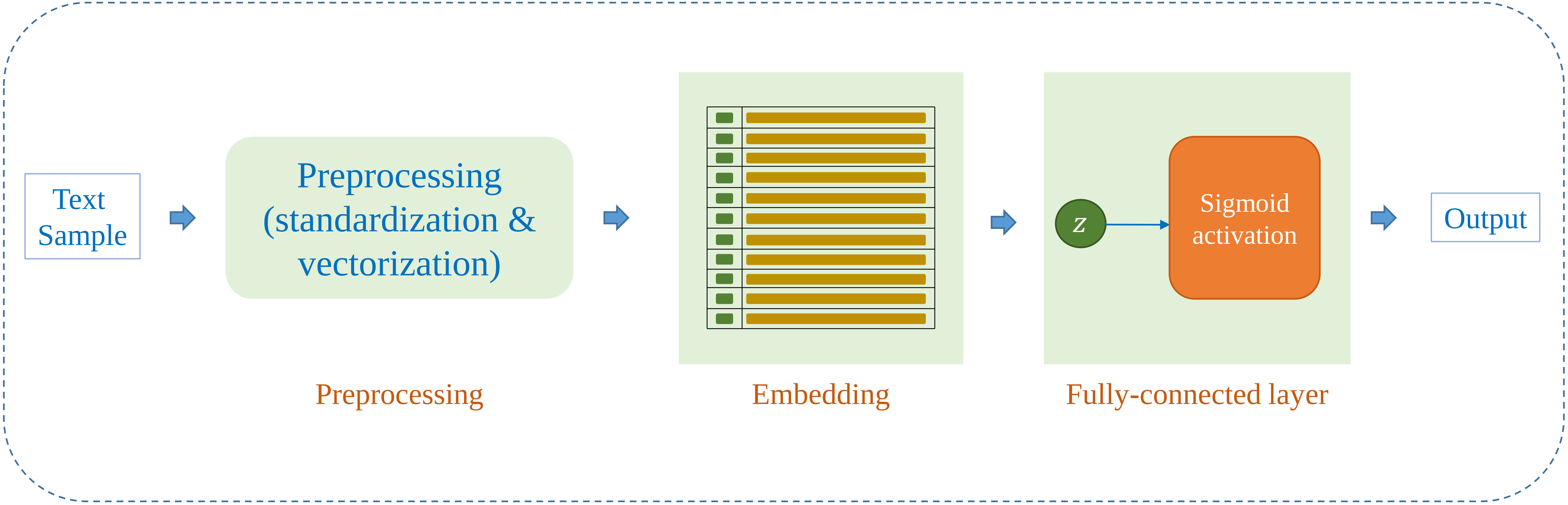
❖ IMDB dataset

- 50,000 movie review for sentiment analysis
- Consist of: + 25,000 movie review for training
  + 25,000 movie review for testing
- Label: positive – negative

“A wonderful little production. The filming technique is very unassuming- very old-time-BBC fashion and gives a comforting, and sometimes discomforting, sense of realism to the entire piece…..”	positive
“This show was an amazing, fresh & innovative idea in the 70's when it first aired. The first 7 or 8 years were brilliant, but things dropped off after that. By 1990, the show was not really funny anymore, and it's continued its decline further to the complete waste of time it is today….”	negative
“I thought this was a wonderful way to spend time on a too hot summer weekend, sitting in the air conditioned theater and watching a light-hearted comedy. The plot is simplistic, but the dialogue is witty and the characters are likable (even the well bread suspected serial killer)….”	positive
“BTW Carver gets a very annoying sidekick who makes you wanna shoot him the first three minutes he's on screen.”	negative

Text Classification

❖ Simple approach



Embedding

index	0	1	2	3	4	5	6	7
word	[UNK]	pad	ai	a	are	cs	is	learning

- Example corpus
  - sample1: ‘We are learning AI’
  - sample2: ‘AI is a CS topic’

(1) Build vocabulary from corpus

We are learning AI	AI is a CS topic

⬇ Standardize

we	are	learning	ai	ai	is	a	cs	topic

```python
from torchtext.data.utils import get_tokenizer

sample1 = 'We are learning AI'
sample2 = 'AI is a CS topic'

# Define tokenizer function
tokenizer = get_tokenizer('basic_english')
sample1_tokens = tokenizer(sample1)
sample2_tokens = tokenizer(sample2)

print(sample1_tokens)
print(sample2_tokens)
```

```
['we', 'are', 'learning', 'ai']
['ai', 'is', 'a', 'cs', 'topic']
```

Embedding

index	0	1	2	3	4	5	6	7
word	[UNK]	pad	ai	a	are	cs	is	learning

- Example corpus
  - sample1: ‘We are learning AI’
  - sample2: ‘AI is a CS topic’

(1) Build vocabulary from corpus

```python
# problem?
from torchtext.data.utils import get_tokenizer
from torchtext.vocab import build_vocab_from_iterator

sample1 = 'We are learning AI'
sample2 = 'AI is a CS topic'
data = [sample1, sample2, ...]

# Create vocabulary
vocab_size = 8
vocab = build_vocab_from_iterator(data)
```

(Large) Text Dataset

#different words are enormous

How to represent ‘text’ effectively?

Embedding

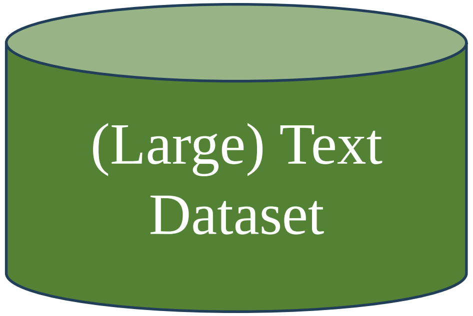
index	0	1	2	3	4	5	6	7
word	[UNK]	pad	ai	a	are	cs	is	learning

- Example corpus

  sample1: 'We are learning AI'

  sample2: 'AI is a CS topic'

(1) Build vocabulary from corpus




```python
from torchtext.data.utils import get_tokenizer
from torchtext.vocab import build_vocab_from_iterator

sample1 = 'We are learning AI'
sample2 = 'AI is a CS topic'
data = [sample1, sample2, ...]

# Create vocabulary
vocab_size = 8
vocab = build_vocab_from_iterator(data,
                                  max_tokens=vocab_size,
                                  specials=["<unk>",
                                            "<pad>"])
vocab.set_default_index(vocab["<unk>"])
```

#different words are enormous

How to represent 'text' effectively?

Use a limited number of words

Embedding

index	0	1	2	3	4	5	6	7
word	[UNK]	pad	ai	a	are	cs	is	learning

- Example corpus
  sample1: ‘We are learning AI’
  sample2: ‘AI is a CS topic’

(1) Build vocabulary from corpus

```
from torchtext.data.utils import get_tokenizer
from torchtext.vocab import build_vocab_from_iterator

sample1 = 'We are learning AI'
sample2 = 'AI is a CS topic'
data = [sample1, sample2]

# Create a function to yield list of tokens
tokenizer = get_tokenizer('basic_english')
def yield_tokens(examples):
    for text in examples:
        yield tokenizer(text)

# Create vocabulary
vocab_size = 8
vocab = build_vocab_from_iterator(yield_tokens(data),
                                  max_tokens=vocab_size,
                                  specials=["<unk>",
                                            "<pad>"])
vocab.set_default_index(vocab["<unk>"])
```

```
vocab.get_stoi()
```

```
{'<unk>': 0,
 '<pad>': 1,
 'ai': 2,
 'a': 3,
 'is': 6,
 'are': 4,
 'learning': 7,
 'cs': 5}
```

#different words are enormous

How to represent ‘text’ effectively?

➡ Use a limited number of words

➡ Get data sample-by-sample

Embedding

index	0	1	2	3	4	5	6	7
word	[UNK]	pad	ai	a	are	cs	is	learning

- Example corpus
  sample1: 'We are learning AI'
  sample2: 'AI is a CS topic'

(1) Build vocabulary from corpus

(2) Transform text into features

We are learning AI					AI is a CS topic				
we	are	learning	ai		ai	is	a	cs	topic
0	4	7	2	1	2	6	3	5	0

Standardize

Vectorization

'We' 'are' 'learning' 'AI'

Demo

```
tokens = tokenizer(sample1)
print(tokens)

sample1_tokens = [vocab[token] for token in tokens]
print(sample1_tokens)
```

```
['we', 'are', 'learning', 'ai']
[0, 4, 7, 2]
```

```
tokens = tokenizer(sample2)
print(tokens)

sample2_tokens = [vocab[token] for token in tokens]
print(sample2_tokens)
```

```
['ai', 'is', 'a', 'cs', 'topic']
[2, 6, 3, 5, 0]
```

Embedding

index	0	1	2	3	4	5	6	7
word	[UNK]	pad	ai	a	are	cs	is	learning

- Example corpus
  - sample1: ‘We are learning AI’
  - sample2: ‘AI is a CS topic’

(1) Build vocabulary from corpus

(2) Transform text into features

We are learning AI	AI is a CS topic

Standardize

we	are	learning	ai

ai	is	a	cs	topic

Vectorization

0	4	7	2	1

2	6	3	5	0

sample3 = AI topic in CS is difficult'

```
def vectorize(text, vocab, sequence_length):
    tokens = tokenizer(text)
    tokens = [vocab[token] for token in tokens]

    num_pads = sequence_length - len(tokens)
    tokens = tokens + [vocab["<pad>"]] * num_pads

    return torch.tensor(tokens, dtype=torch.long)


# Vectorize the samples
sequence_length = 5
vectorized_sample1 = vectorize(sample1,
                               vocab,
                               sequence_length)
vectorized_sample2 = vectorize(sample2,
                               vocab,
                               sequence_length)

print("Vectorized Sample 1:", vectorized_sample1)
print("Vectorized Sample 2:", vectorized_sample2)
```

```
Vectorized Sample 1: tensor([0, 4, 7, 2, 1])
Vectorized Sample 2: tensor([2, 6, 3, 5, 0])
```

Embedding

- Example corpus

  sample1: ‘We are learning AI’

  sample2: ‘AI is a CS topic’

(1) Build vocabulary from corpus

(2) Transform text into features

We are learning AI						AI is a CS topic				

↓ Standardize

we	are	learning	ai		ai	is	a	cs	topic

↓ Vectorization

0	4	7	2	1		2	6	3	5	0

```python
def vectorize(text, vocab, seq_len):
    tokens = tokenizer(text)
    tokens = [vocab[token] for token in tokens]

    num_pads = sequence_length - len(tokens)
    tokens = tokens[:sequence_length]
             + [vocab["<pad>"]]*num_pads

    return torch.tensor(tokens, dtype=torch.long)

# Vectorize the samples
sequence_length = 5
vectorized_sample1 = vectorize(sample1, vocab,
                               sequence_length)
vectorized_sample2 = vectorize(sample2, vocab,
                               sequence_length)

print("Vectorized Sample 1:", vectorized_sample1)
print("Vectorized Sample 2:", vectorized_sample2)
```

```
Vectorized Sample 1: tensor([0, 4, 7, 2, 1])
Vectorized Sample 2: tensor([2, 6, 3, 5, 0])
```

```python
sample3 = 'AI topic in CS is difficult'
vectorized_sample3 = vectorize(sample3, vocab,
                               sequence_length)
print(vectorized_sample3)
```

```
tensor([2, 0, 0, 5, 6])
```

Embedding Layer

(3) Embedding layer

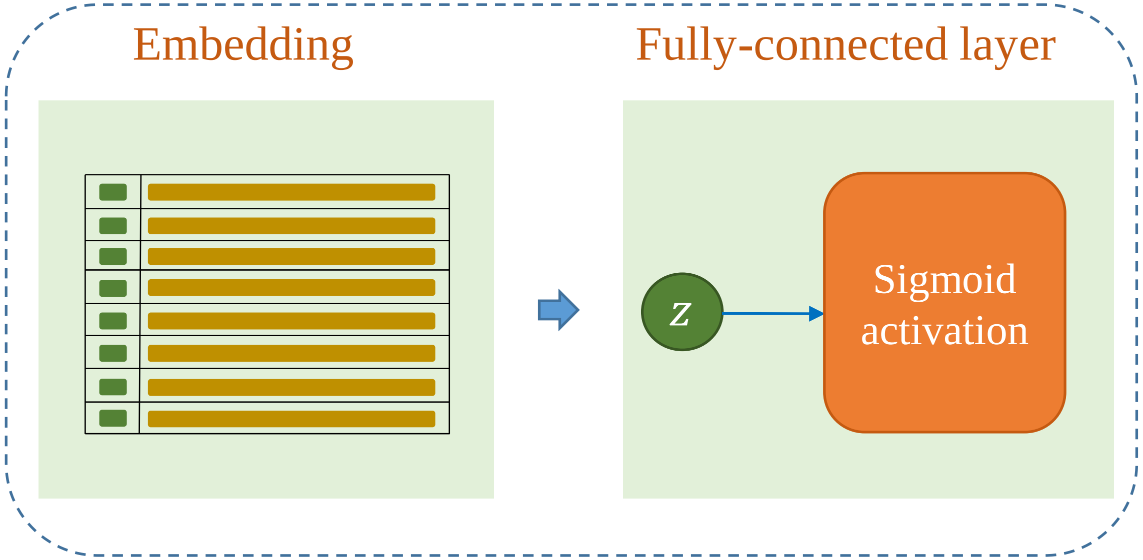
index	word
0	[UNK]
1	[pad]
2	ai
3	a
4	are
5	cs
6	is
7	learning

```
vocab_size = 8
embed_dim = 4
embedding = nn.Embedding(vocab_size,
                         embed_dim)
```

```
Parameter containing:
tensor([[-0.1882,  0.5530,  1.6267,  0.7013],
        [ 1.7840, -0.8278, -0.2701,  1.3586],
        [ 1.0281, -1.9094,  0.3182,  0.4211],
        [-1.3083, -0.0987,  0.7647, -0.3680],
        [ 0.2293,  1.3255,  0.1318,  2.0501],
        [ 0.4058, -0.6624, -0.8745,  0.7203],
        [ 0.5582,  0.0786, -0.6817,  0.6902],
        [ 0.4309, -1.3067, -0.8823,  1.5977]],
```

We are learning AI

0
4
7
2
1

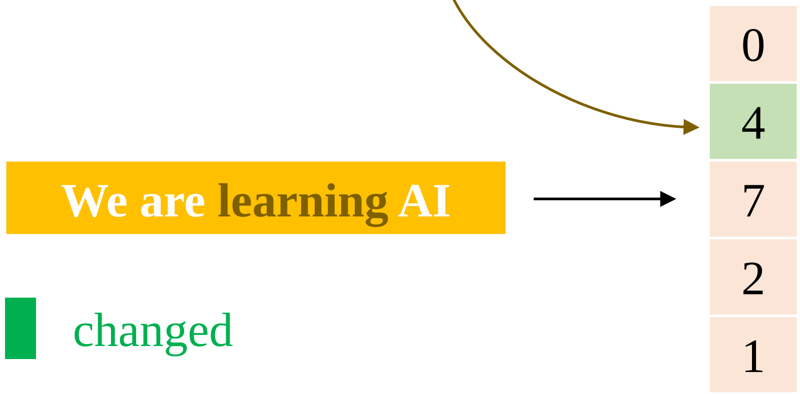


Output

Embedding Layer

(3) Embedding layer

index	word
0	[UNK]
1	[pad]
2	ai
3	a
4	are
5	cs
6	is
7	learning



changed

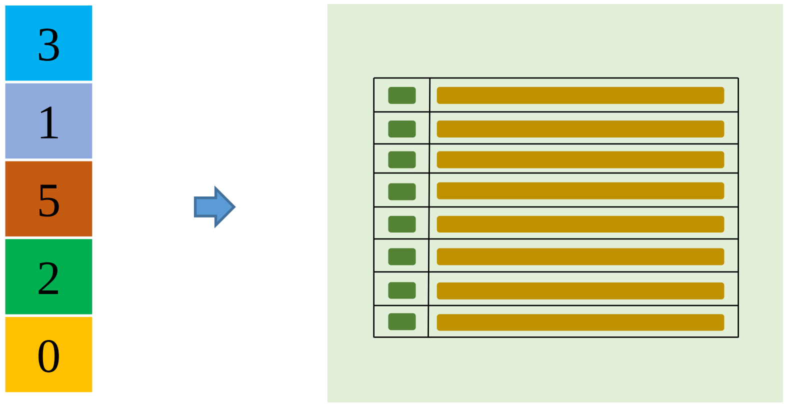
```
Parameter containing:
tensor([[-0.1882,  0.5530,  1.6267,  0.7013],
        [ 1.7840, -0.8278, -0.2701,  1.3586],
        [ 1.0281, -1.9094,  0.3182,  0.4211],
        [-1.3083, -0.0987,  0.7647, -0.3680],
        [ 0.2293,  1.3255,  0.1318,  2.0501],
        [ 0.4058, -0.6624, -0.8745,  0.7203],
        [ 0.5582,  0.0786, -0.6817,  0.6902],
        [ 0.4309, -1.3067, -0.8823,  1.5977]],
```

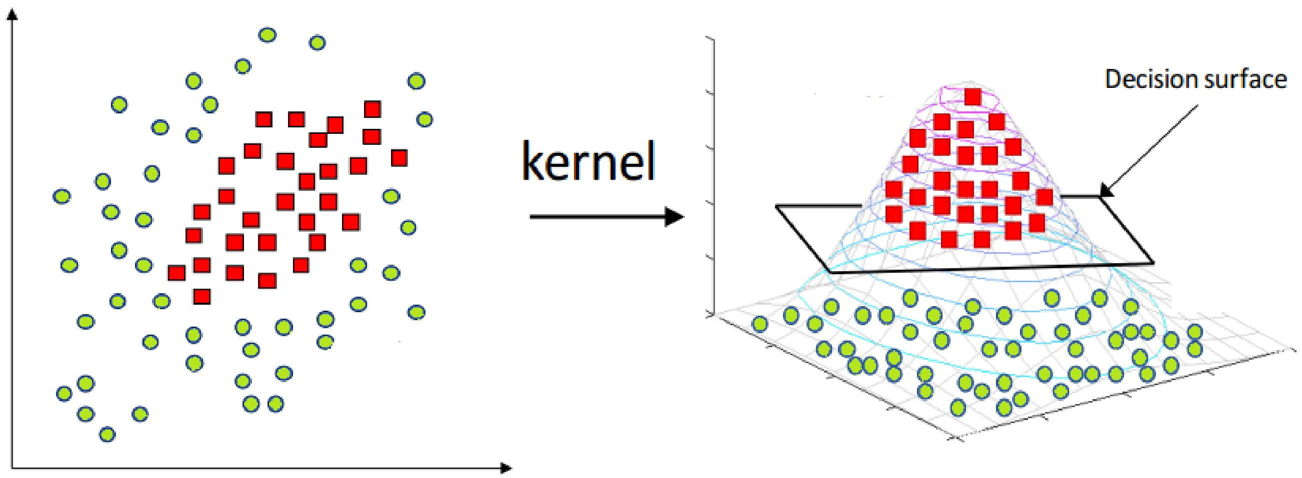
```
Parameter containing:
tensor([[-0.1872,  0.5540,  1.6277,  0.7023],
        [ 1.7830, -0.8268, -0.2711,  1.3576],
        [ 1.0291, -1.9084,  0.3192,  0.4201],
        [-1.3083, -0.0987,  0.7647, -0.3680],
        [ 0.2303,  1.3245,  0.1308,  2.0511],
        [ 0.4058, -0.6624, -0.8745,  0.7203],
        [ 0.5582,  0.0786, -0.6817,  0.6902],
        [ 0.4299, -1.3077, -0.8833,  1.5967]],
```

Word Embedding

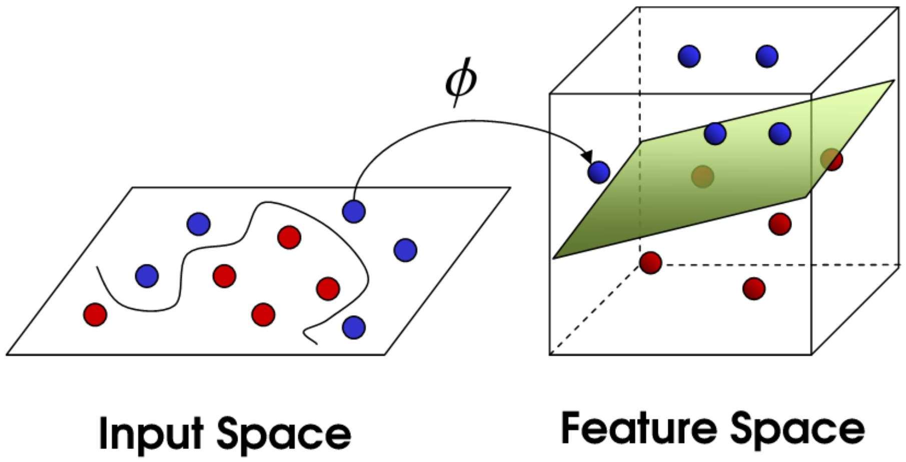
❖ **Why?**

Embedding





https://codatalicious.medium.com/kernels-ee967067aa9



https://towardsdatascience.com/the-kernel-trick-c98cdbcaeb3f

Outline

- Dealing with Text
- Constructing RNN
- RNN Examples for Text
- RNN Examples for Time-series Data
- PyTorch Implementation

Revisit input x

Convert from text to numbers

index	word
0	[UNK]
1	[pad]
2	ai
3	a
4	are
5	cs
6	is
7	learning

A sample X

We are learning AI → 0, 4, 7, 2, 1

```
Parameter containing:
tensor([[-0.1882,  0.5530,  1.6267,  0.7013],
        [ 1.7840, -0.8278, -0.2701,  1.3586],
        [ 1.0281, -1.9094,  0.3182,  0.4211],
        [-1.3083, -0.0987,  0.7647, -0.3680],
        [ 0.2293,  1.3255,  0.1318,  2.0501],
        [ 0.4058, -0.6624, -0.8745,  0.7203],
        [ 0.5582,  0.0786, -0.6817,  0.6902],
        [ 0.4309, -1.3067, -0.8823,  1.5977]],
```

A sample X

We	0	X_1: [-0.1882, 0.5530, 1.6267, 0.7013]
are	4	X_2: [0.2293, 1.3255, 0.1318, 2.0501]
learning	7	X_3: [0.4309, -1.3067, -0.8823, 1.5977]
AI	2	X_4: [1.0281, -1.9094, 0.3182, 0.4211]
<pad>	1	X_5: [1.7840, -0.8278, -0.2701, 1.3586]

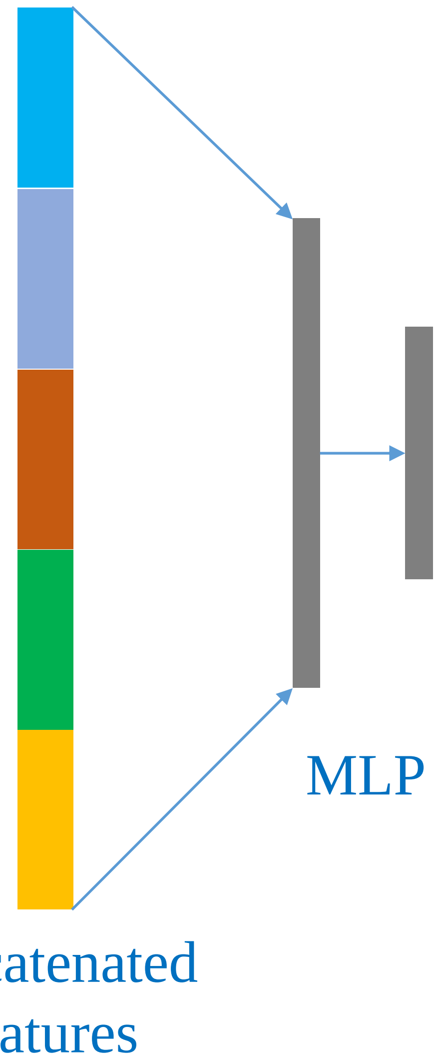
How to feed X to a network?
(Get ideas from how MLP and CNN work)

How to deal with this input?

- ❖ **Simplest idea: Based on MLP**
- ❖ **Concatenate all the features**

A sample X	Index	Feature
We	0	X_1: [-0.1882, 0.5530, 1.6267, 0.7013]
are	4	X_2: [0.2293, 1.3255, 0.1318, 2.0501]
learning	7	X_3: [0.4309, -1.3067, -0.8823, 1.5977]
AI	2	X_4: [1.0281, -1.9094, 0.3182, 0.4211]
<pad>	1	X_5: [1.7840, -0.8278, -0.2701, 1.3586]

A sample X

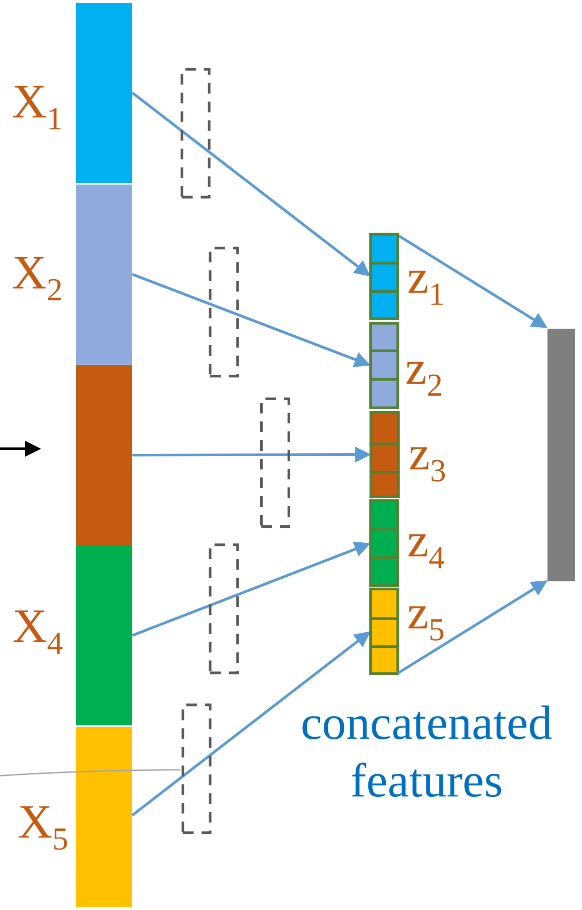


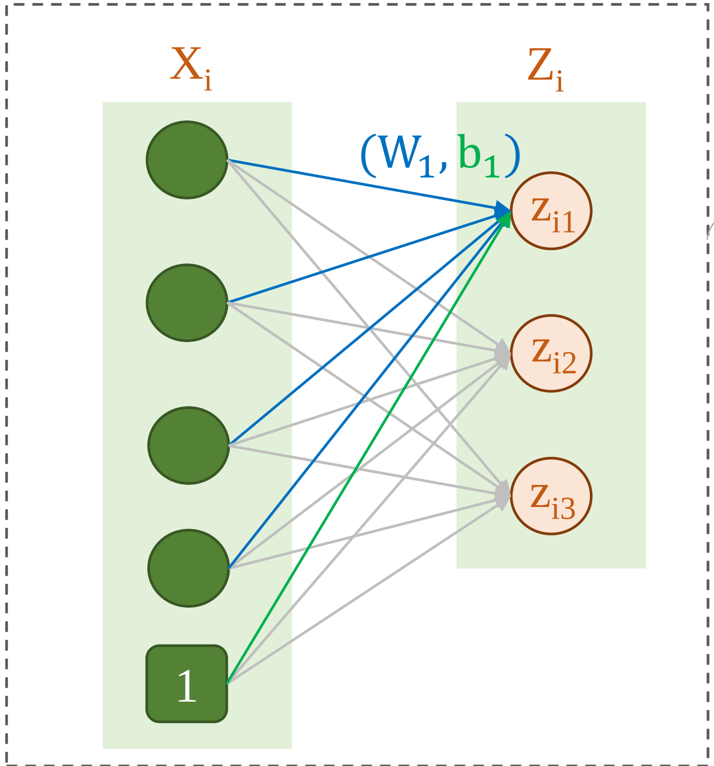
Pros and cons?

❖ Based on CNN: Shared weights

We	0	X_1: [-0.1882, 0.5530, 1.6267, 0.7013]
are	4	X_2: [0.2293, 1.3255, 0.1318, 2.0501]
learning	7	X_3: [0.4309, -1.3067, -0.8823, 1.5977]
AI	2	X_4: [1.0281, -1.9094, 0.3182, 0.4211]
<pad>	1	X_5: [1.7840, -0.8278, -0.2701, 1.3586]

A sample X





$W =$

$$b = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \quad z_i = \begin{bmatrix} z_{i1} \\ z_{i2} \\ z_{i3} \end{bmatrix}$$

$$z_{i1} = W_1{}^T X_i + b_1$$
$$z_{i2} = W_2{}^T X_i + b_2$$
$$z_{i3} = W_3{}^T X_i + b_3$$

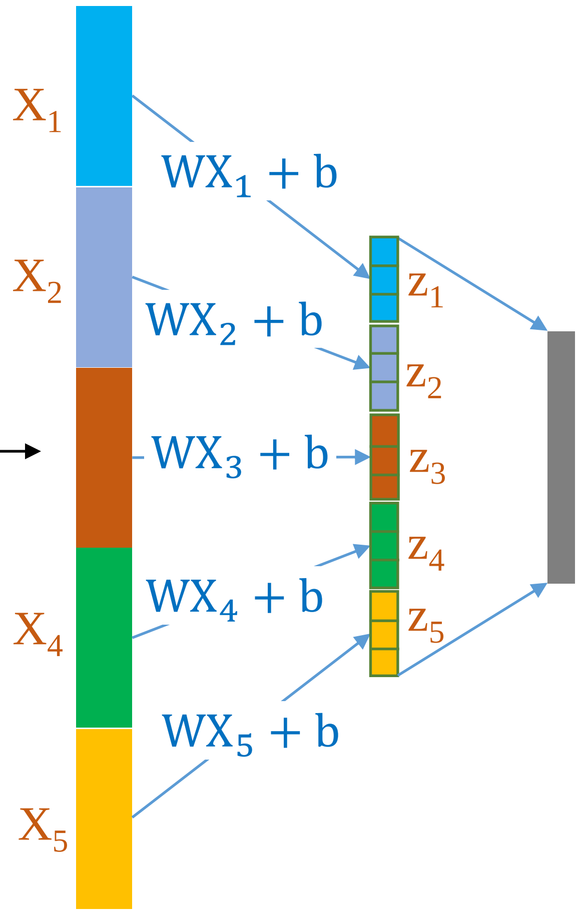
Fully-connected layer
Shared weights

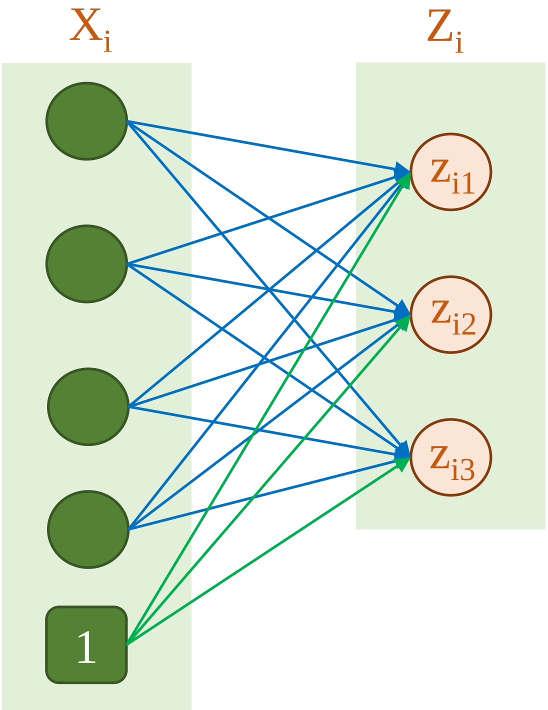
$$\Rightarrow z_i = WX_i + b$$

❖ Based on CNN: Shared weights

A sample X

We	0	X_1: [-0.1882, 0.5530, 1.6267, 0.7013]
are	4	X_2: [0.2293, 1.3255, 0.1318, 2.0501]
learning	7	X_3: [0.4309, -1.3067, -0.8823, 1.5977]
AI	2	X_4: [1.0281, -1.9094, 0.3182, 0.4211]
<pad>	1	X_5: [1.7840, -0.8278, -0.2701, 1.3586]





$$z_{i1} = W_1{}^T X_i + b_1$$
$$z_{i2} = W_2{}^T X_i + b_2$$
$$z_{i3} = W_3{}^T X_i + b_3$$

➧ $z_i = WX_i + b$

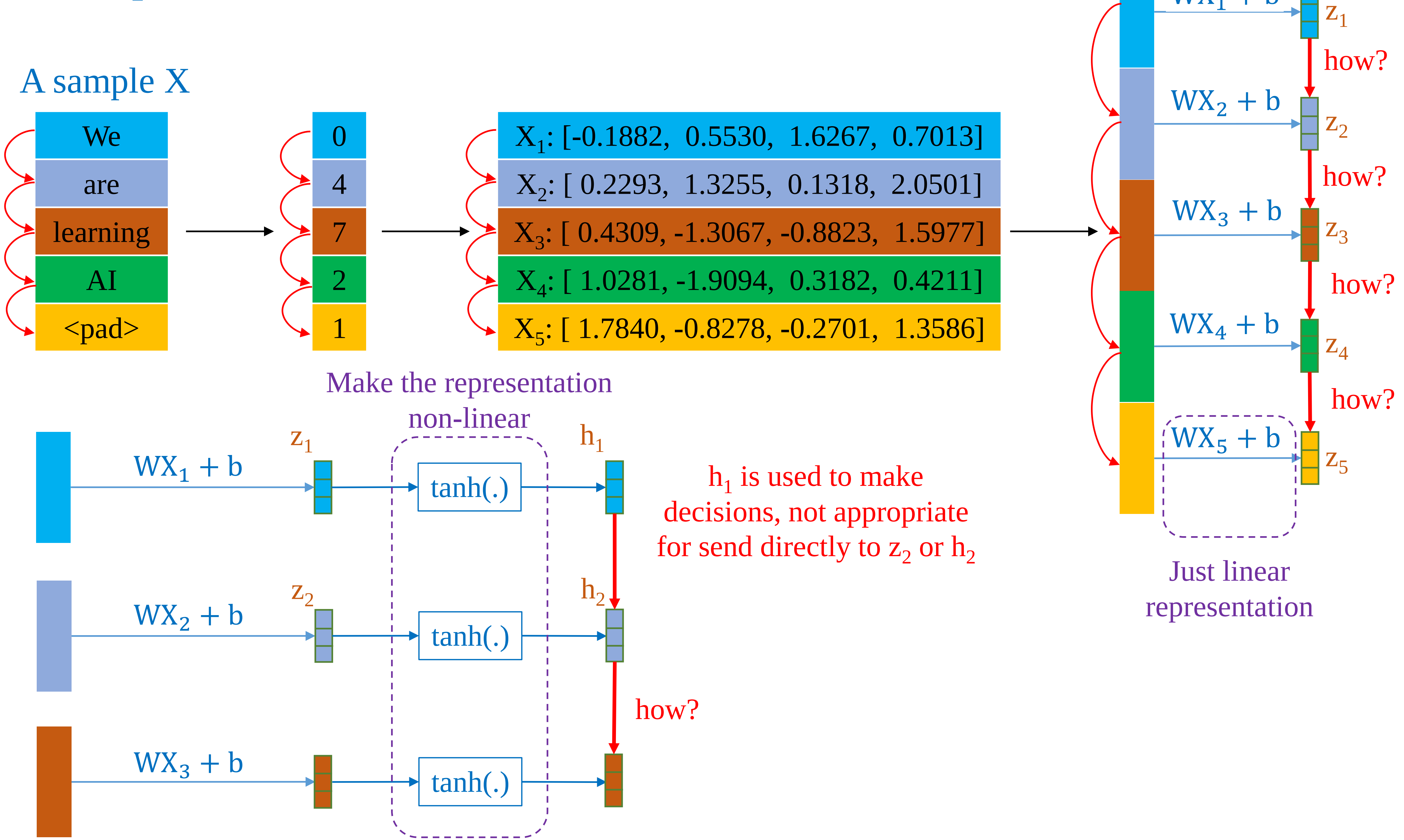
$W =$

$$b = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \qquad z_i = \begin{bmatrix} z_{i1} \\ z_{i2} \\ z_{i3} \end{bmatrix}$$

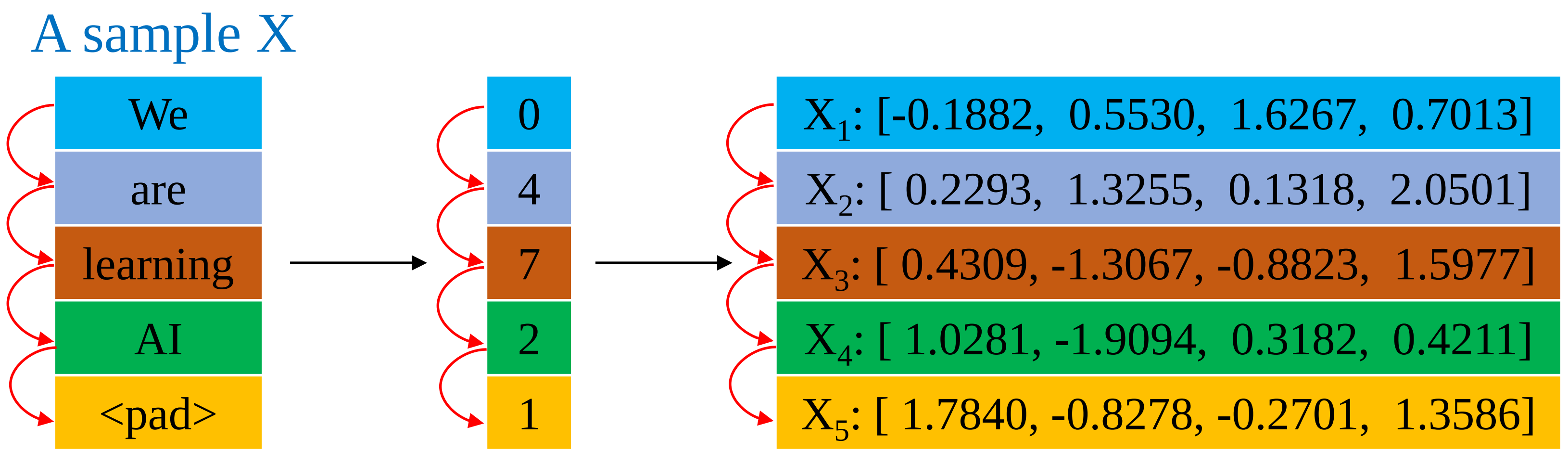
$$z_1 = WX_1 + b$$
$$z_2 = WX_2 + b$$
$$z_3 = WX_3 + b$$
$$z_4 = WX_4 + b$$
$$z_5 = WX_5 + b$$

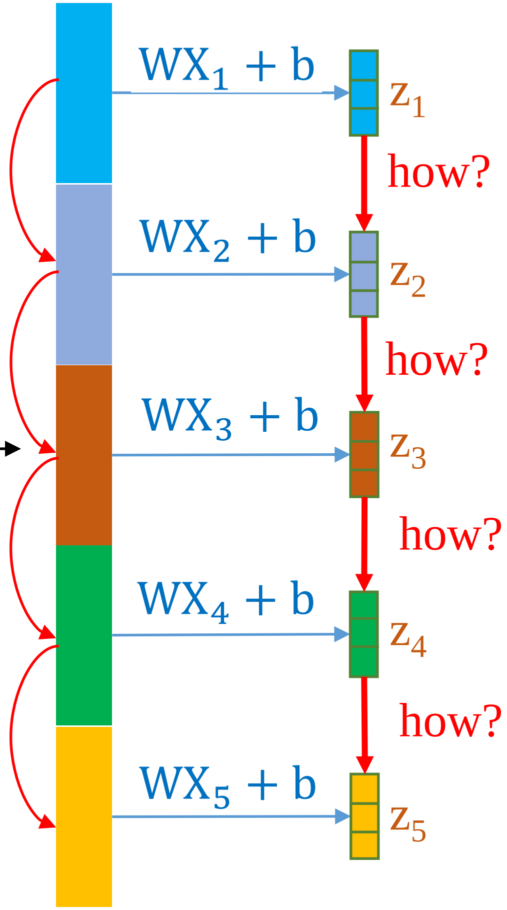
Any thing missing?

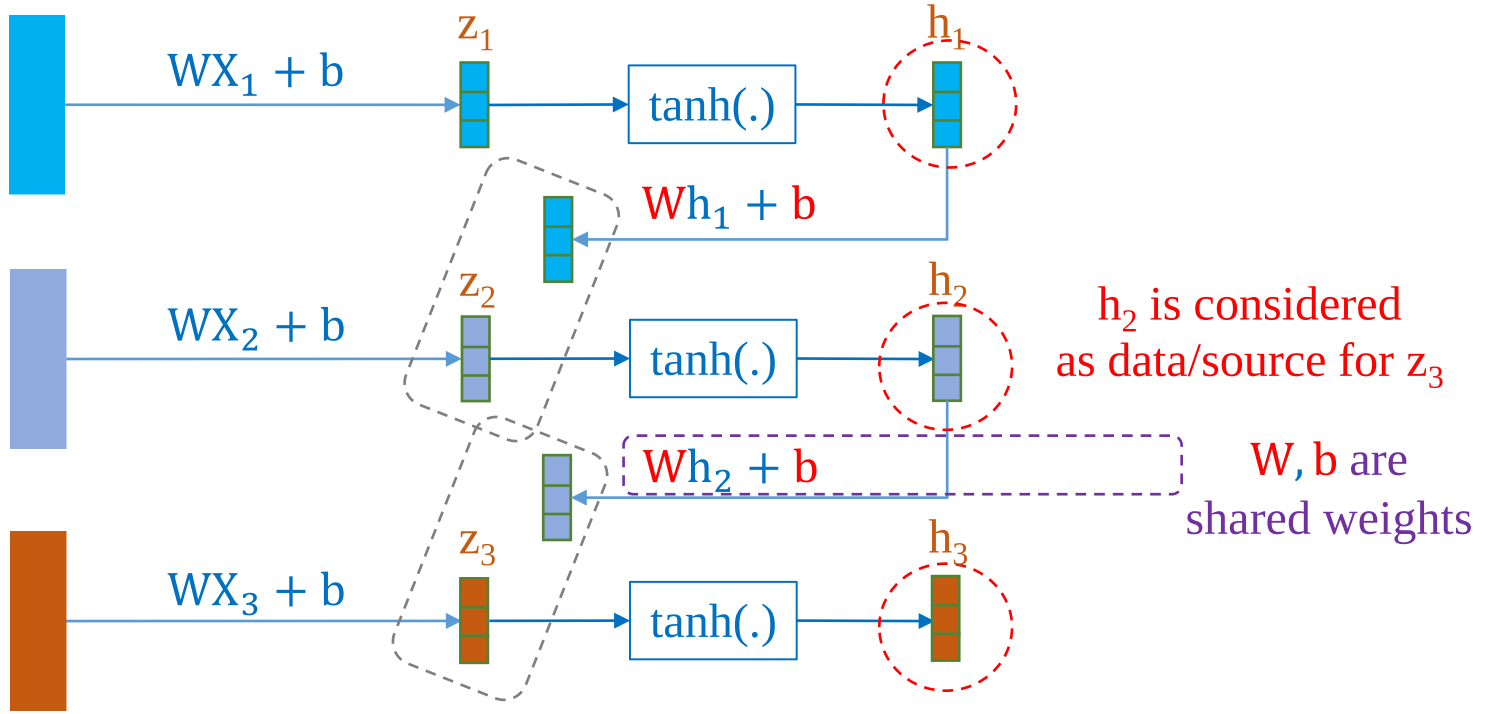
❖ How to exploit causal information?



❖ How to exploit causal information?







h_2 is considered as data/source for z_3

W, b are shared weights

How to combine the pairs of two tensors?

❖ How to exploit causal information?

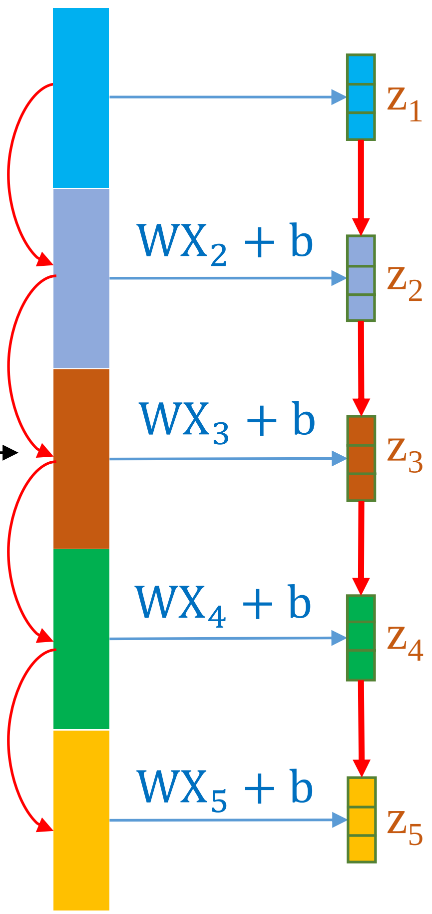
X_1: [-0.1882, 0.5530, 1.6267, 0.7013]

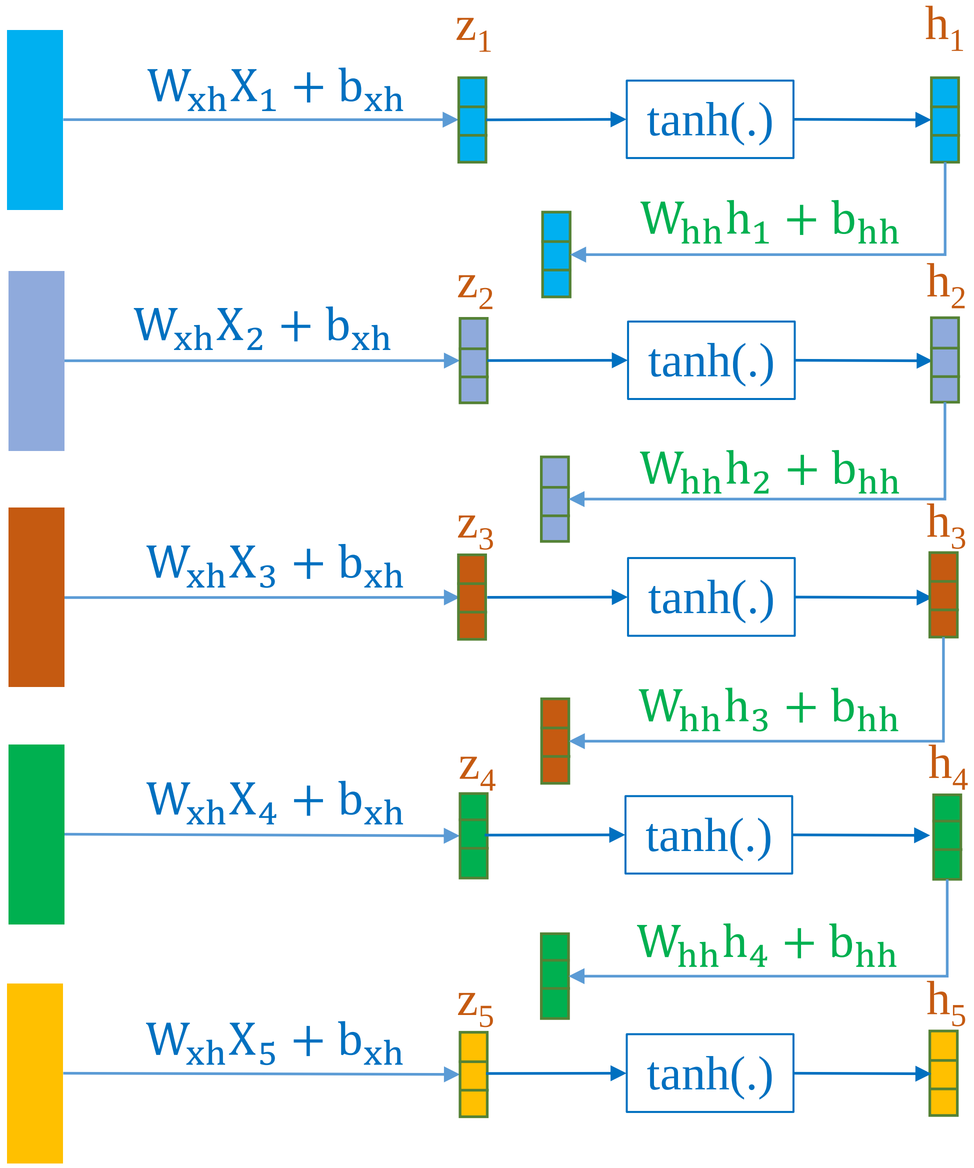
X_2: [0.2293, 1.3255, 0.1318, 2.0501]

X_3: [0.4309, -1.3067, -0.8823, 1.5977]

X_4: [1.0281, -1.9094, 0.3182, 0.4211]

X_5: [1.7840, -0.8278, -0.2701, 1.3586]

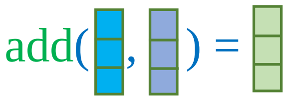


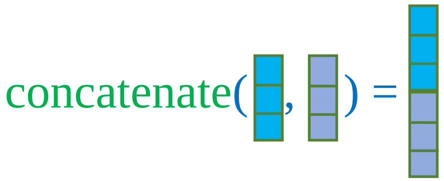


How to combine two tensors?

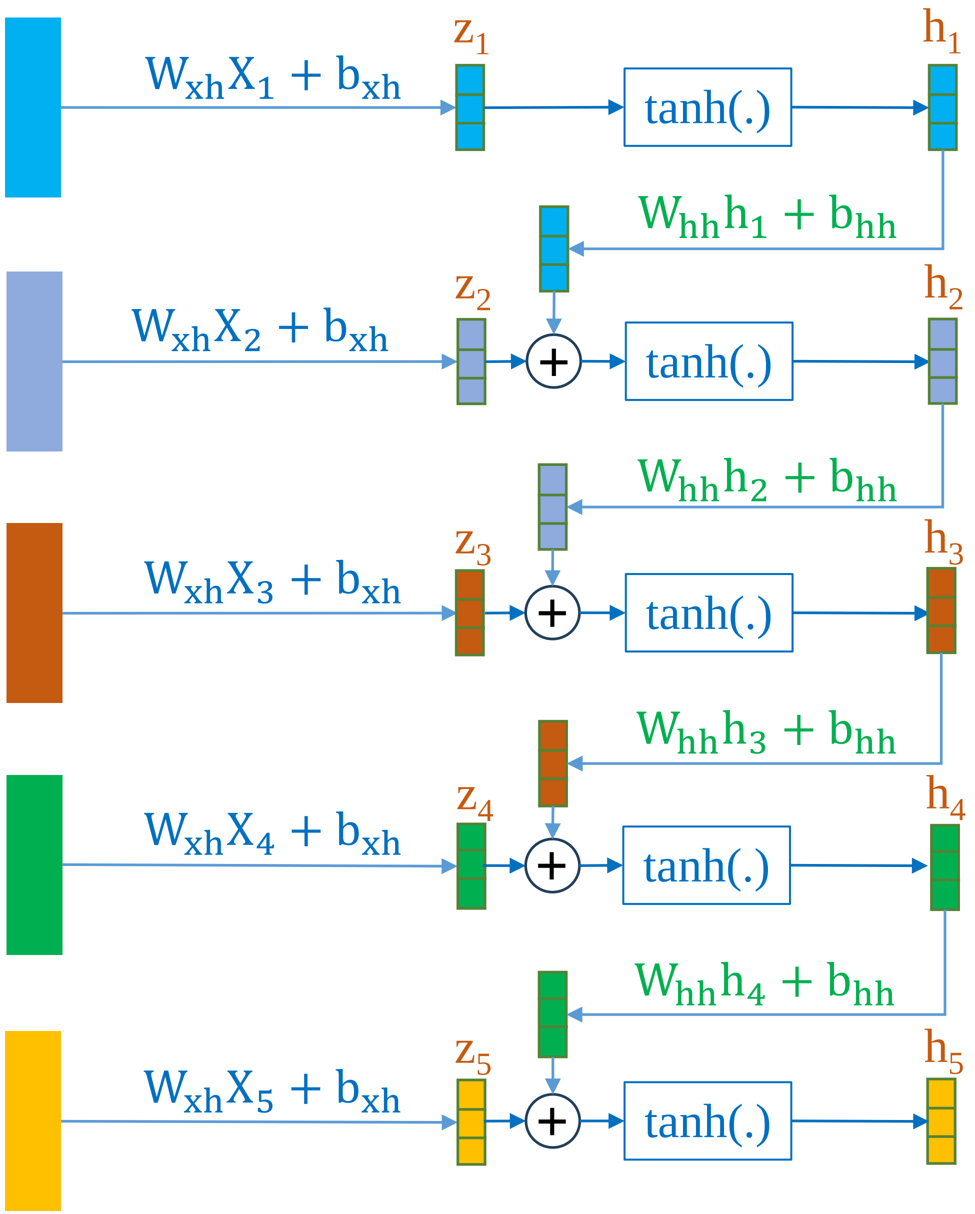
Idea from skip-connection

- or Concatenation
- or Addition





❖ How to exploit causal information?



$$h_1 = \tanh(W_{xh}X_1 + b_{xh})$$

$$h_2 = \tanh(W_{xh}X_2 + b_{xh} + W_{hh}h_1 + b_{hh})$$

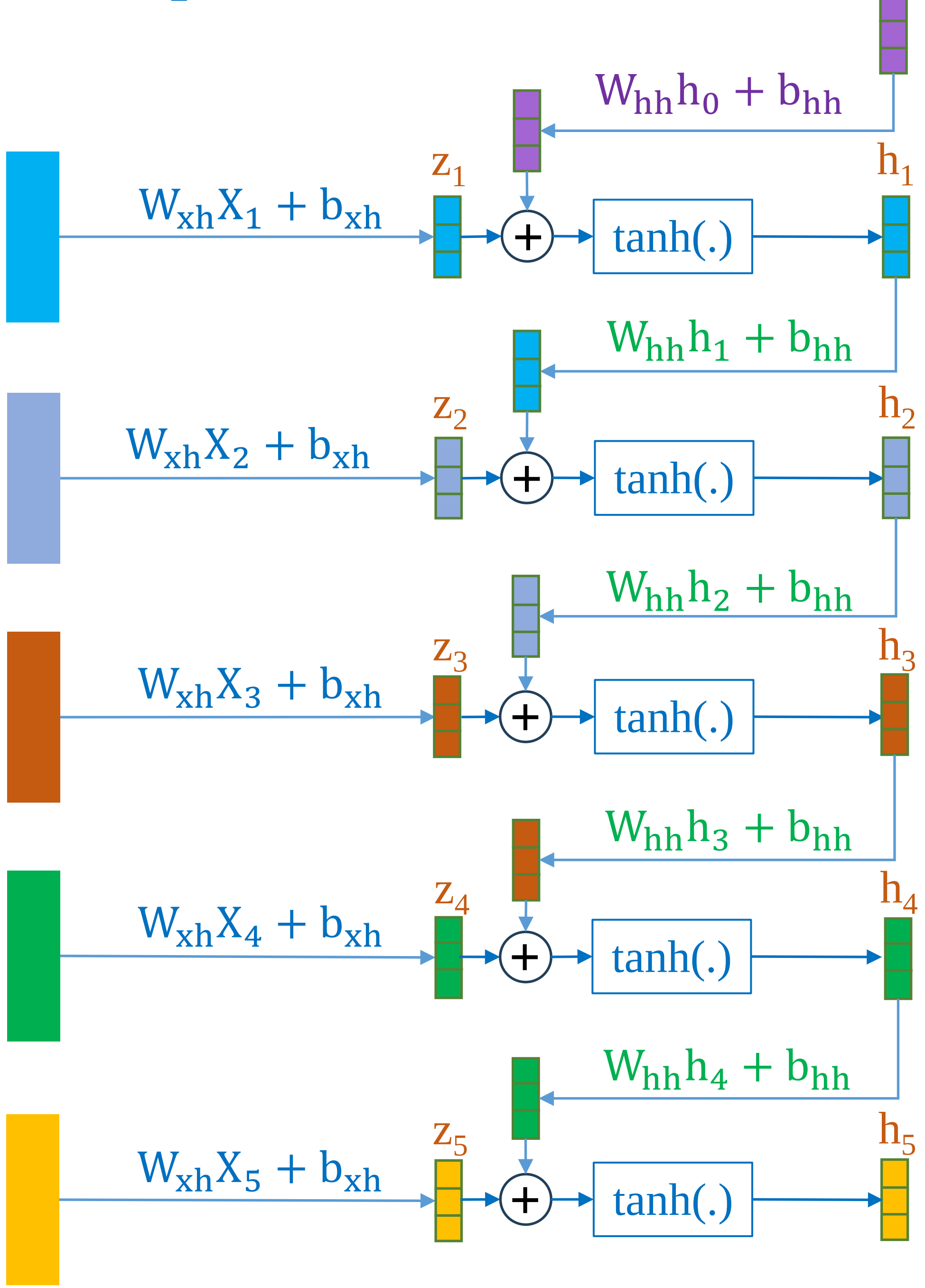
$$h_3 = \tanh(W_{xh}X_3 + b_{xh} + W_{hh}h_2 + b_{hh})$$

$$h_4 = \tanh(W_{xh}X_4 + b_{xh} + W_{hh}h_3 + b_{hh})$$

$$h_5 = \tanh(W_{xh}X_5 + b_{xh} + W_{hh}h_4 + b_{hh})$$

h_1 formula is different from the others. How to solve this issue?

❖ How to exploit causal information?



$$h_0 = \mathbf{0} \qquad b_{hh} = \mathbf{0}$$

$$h_1 = \tanh(W_{xh}X_1 + b_{xh} + W_{hh}h_0 + b_{hh})$$

$$h_2 = \tanh(W_{xh}X_2 + b_{xh} + W_{hh}h_1 + b_{hh})$$

$$h_3 = \tanh(W_{xh}X_3 + b_{xh} + W_{hh}h_2 + b_{hh})$$

$$h_4 = \tanh(W_{xh}X_4 + b_{xh} + W_{hh}h_3 + b_{hh})$$

$$h_5 = \tanh(W_{xh}X_5 + b_{xh} + W_{hh}h_4 + b_{hh})$$

➡ $h_t = \tanh\left(W_{xh}X_t + b_{xh} + W_{hh}h_{(t-1)} + b_{hh}\right)$

➡ RNN

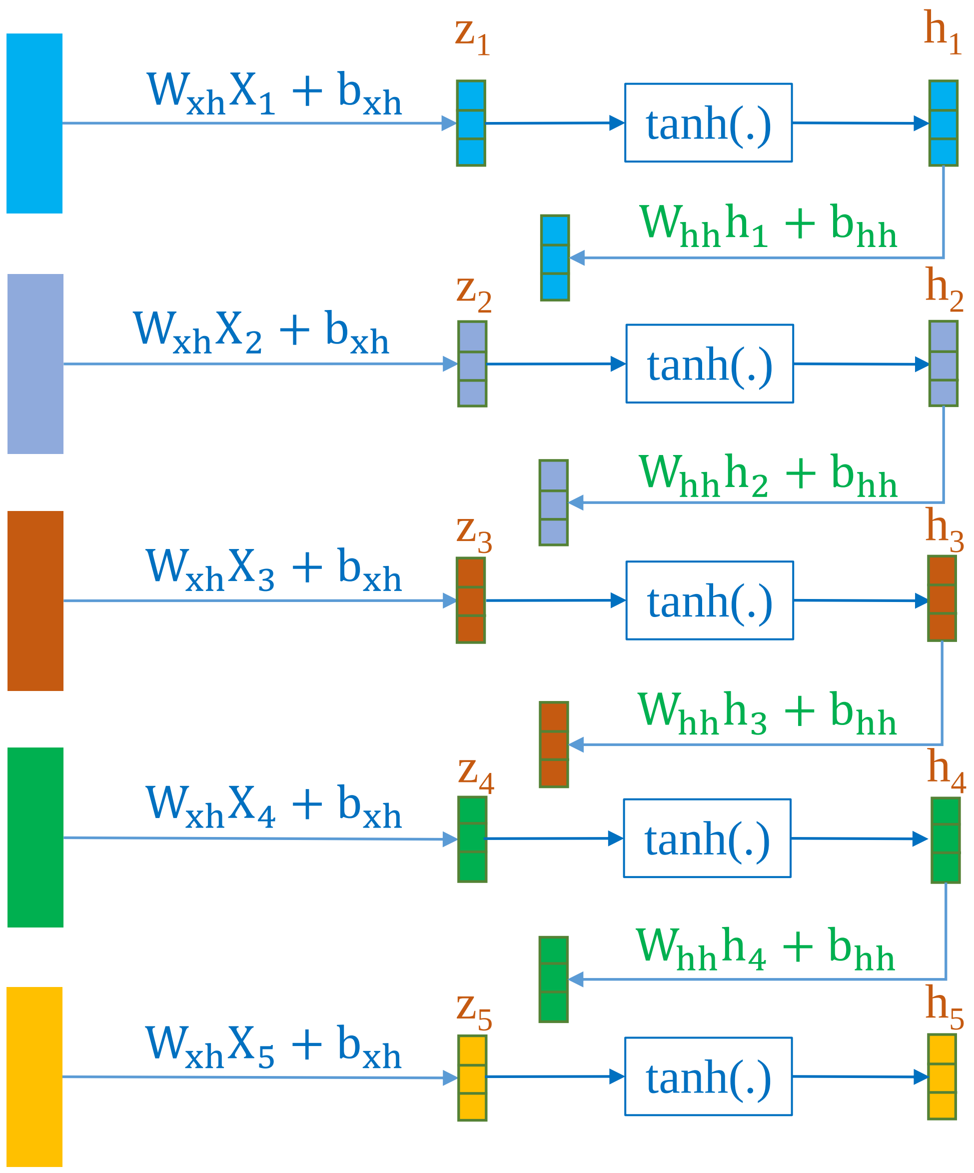
Example

X_1:	[-0.1882, 0.5530, 1.6267, 0.7013]
X_2:	[0.2293, 1.3255, 0.1318, 2.0501]
X_3:	[0.4309, -1.3067, -0.8823, 1.5977]
X_4:	[1.0281, -1.9094, 0.3182, 0.4211]
X_5:	[1.7840, -0.8278, -0.2701, 1.3586]

$$X_1 = \begin{bmatrix} -0.1882 \\ 0.5530 \\ 1.6267 \\ 0.7013 \end{bmatrix} \qquad X_2 = \begin{bmatrix} 0.2293 \\ 1.3255 \\ 0.1318 \\ 2.0501 \end{bmatrix}$$

$$X_3 = \begin{bmatrix} 0.4309 \\ -1.3067 \\ -0.8823 \\ 1.5977 \end{bmatrix} \qquad X_4 = \begin{bmatrix} 1.0281 \\ -1.9094 \\ 0.3182 \\ 0.4211 \end{bmatrix}$$

$$X_5 = \begin{bmatrix} 1.7840 \\ -0.8278 \\ -0.2701 \\ 1.3586 \end{bmatrix}$$



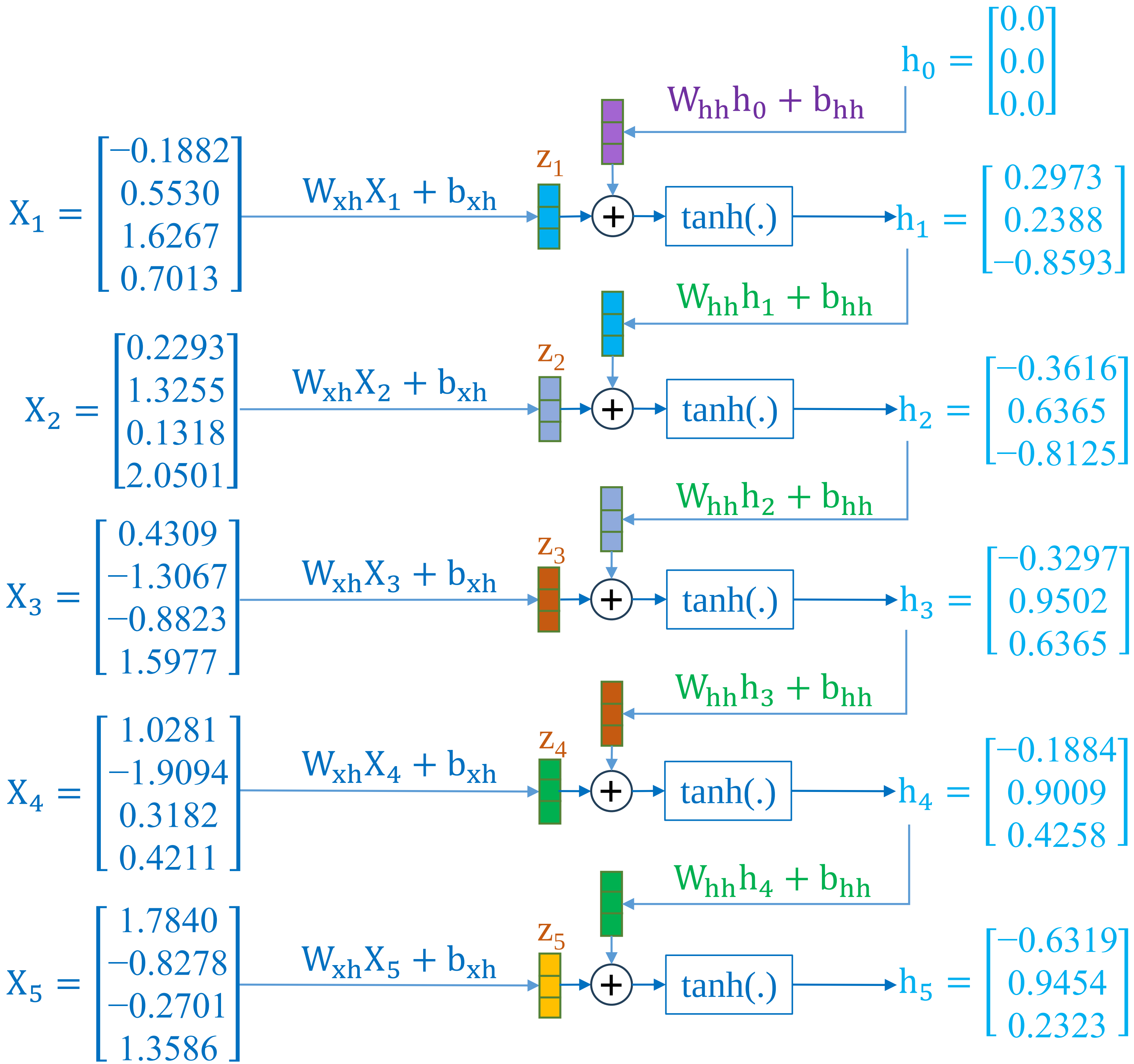
Example

$$W_{xh} = \begin{bmatrix} -0.4174 & 0.1953 & -0.0365 & -0.4025 \\ 0.4722 & -0.4085 & -0.0236 & 0.3763 \\ 0.0550 & -0.4921 & -0.4307 & 0.0855 \end{bmatrix}$$

$$b_{xh} = \begin{bmatrix} 0.4481 \\ 0.5537 \\ -0.5006 \end{bmatrix}$$

$$W_{hh} = \begin{bmatrix} -0.5511 & 0.1260 & 0.0463 \\ -0.2291 & -0.2084 & -0.2352 \\ -0.4341 & -0.2682 & 0.0612 \end{bmatrix}$$

$$b_{hh} = \begin{bmatrix} 0.0135 \\ -0.2209 \\ 0.1330 \end{bmatrix}$$



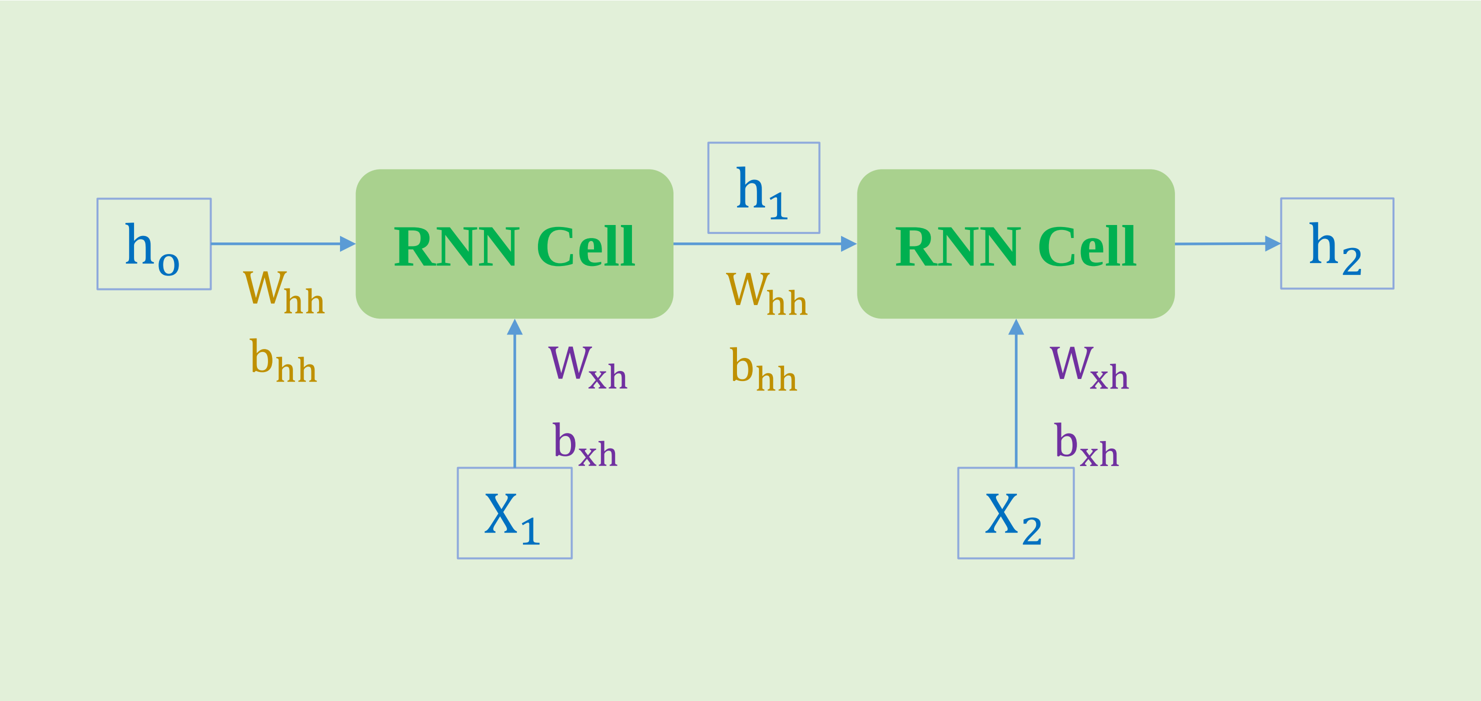
2.il_RNN_Layer.ipynb

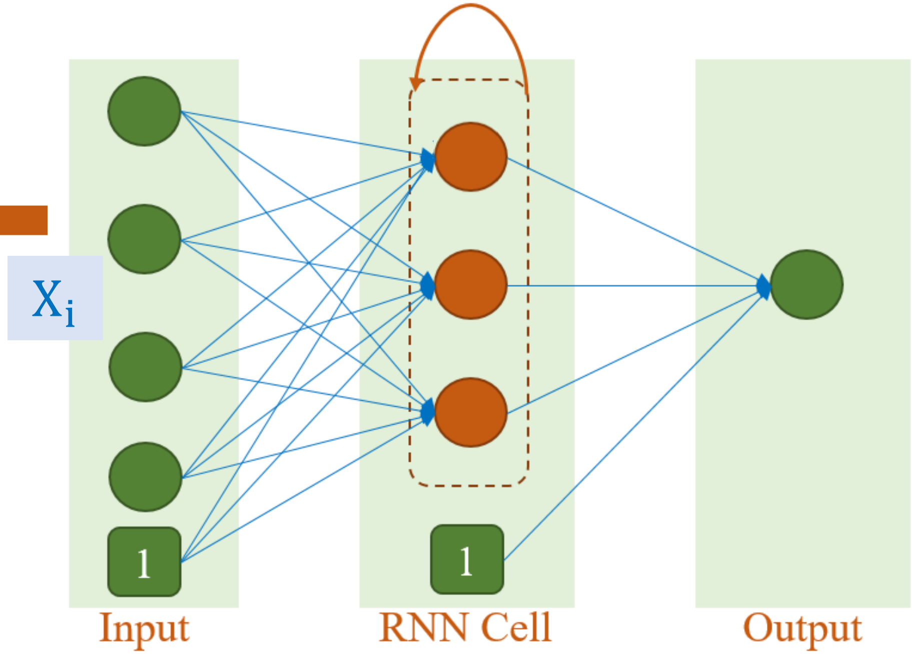
Text Deep Models

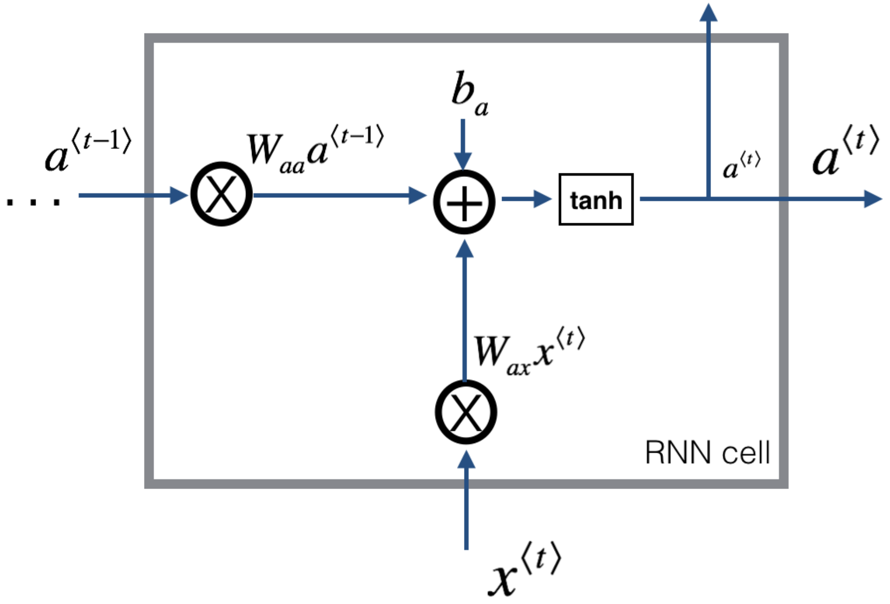
- Recurrent Neural Networks (RNNs)
  - Classification and time-step=2

$$h_1 = \tanh(W_{xh}X_1 + b_{xh} + W_{hh}h_0 + b_{hh})$$

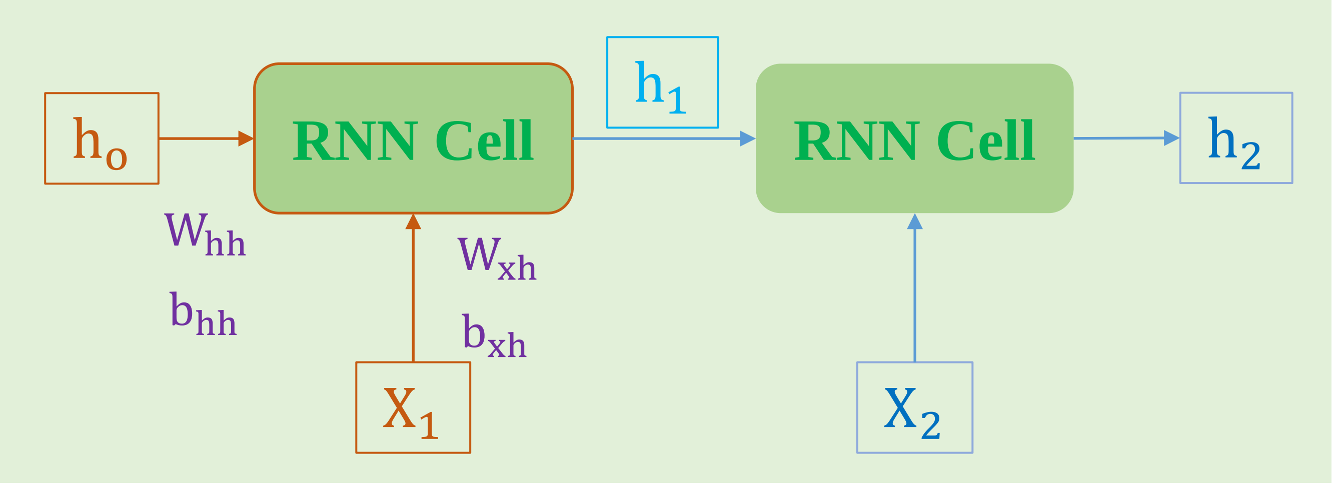
$$h_2 = \tanh(W_{xh}X_2 + b_{xh} + W_{hh}h_1 + b_{hh})$$







https://datascience-enthusiast.com/DL/Building_a_Recurrent_Neural_Network-Step_by_Step_v1.html



$$\mathbf{W_{hh}} = \begin{bmatrix} 0.9 & -0.1 & 0.1 \\ -0.1 & -0.9 & 0.3 \\ 0.1 & -0.3 & -0.9 \end{bmatrix} \qquad \mathbf{b_{hh}} = \begin{bmatrix} 0.0 \\ 0.0 \\ 0.0 \end{bmatrix}$$

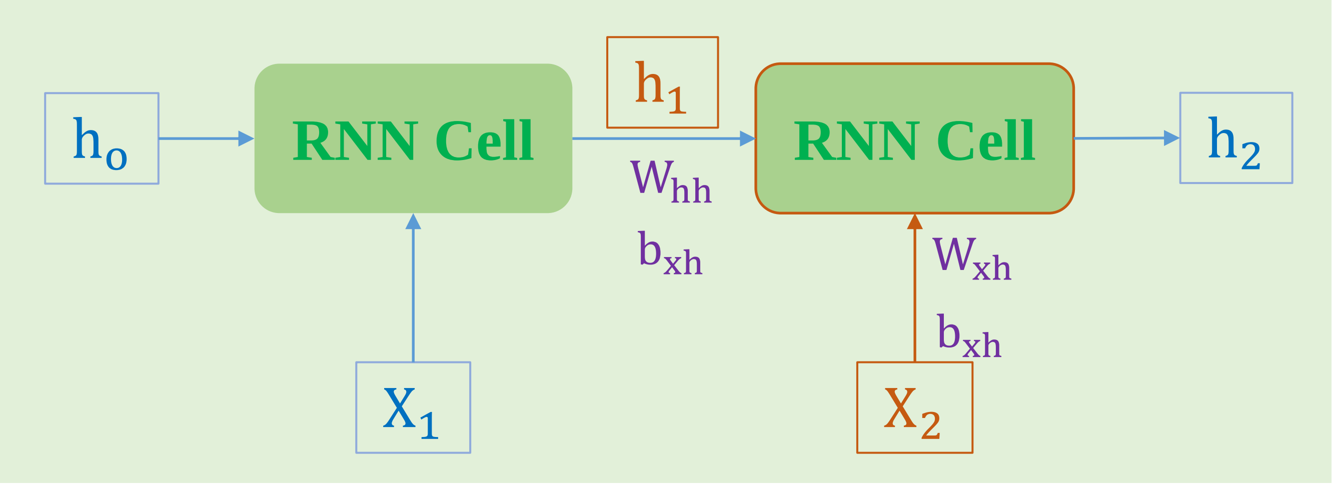
$$\mathbf{W_{xh}} = \begin{bmatrix} -0.2 & -0.4 & 0.3 & -0.4 \\ -0.6 & -0.8 & 0.5 & -0.3 \\ 0.1 & -0.1 & 0.6 & 0.1 \end{bmatrix} \qquad \mathbf{b_{xh}} = \begin{bmatrix} 0.0 \\ 0.0 \\ 0.0 \end{bmatrix}$$

$$\mathbf{X} = \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} 1. & 3. & 2. & 1. \\ 0. & 4. & 1. & 2. \end{bmatrix} \qquad \mathbf{h_0} = \begin{bmatrix} 0.0 \\ 0.0 \\ 0.0 \end{bmatrix}$$

$$h_1 = \tanh(W_{xh}X_1 + b_{xh} + W_{hh}h_0 + b_{hh})$$

$$= \tanh\left(\begin{bmatrix} -0.2 & -0.4 & 0.3 & -0.4 \\ -0.6 & -0.8 & 0.5 & -0.3 \\ 0.1 & -0.1 & 0.6 & 0.1 \end{bmatrix} \begin{bmatrix} 1.0 \\ 3.0 \\ 2.0 \\ 1.0 \end{bmatrix} + \begin{bmatrix} 0.9 & -0.1 & 0.1 \\ -0.1 & -0.9 & 0.3 \\ 0.1 & -0.3 & -0.9 \end{bmatrix} \begin{bmatrix} 0.0 \\ 0.0 \\ 0.0 \end{bmatrix} + \begin{bmatrix} 0.0 \\ 0.0 \\ 0.0 \end{bmatrix} \right)$$

$$= \tanh\left(\begin{bmatrix} -1.2 \\ -2.3 \\ 1.1 \end{bmatrix} \right) = \begin{bmatrix} -0.833 \\ -0.98 \\ 0.8 \end{bmatrix}$$



$$\mathbf{W_{hh}}=\begin{bmatrix} 0.9 & -0.1 & 0.1 \\ -0.1 & -0.9 & 0.3 \\ 0.1 & -0.3 & -0.9 \end{bmatrix} \qquad \mathbf{b_{hh}}=\begin{bmatrix} 0.0 \\ 0.0 \\ 0.0 \end{bmatrix}$$

$$\mathbf{W_{xh}}=\begin{bmatrix} -0.2 & -0.4 & 0.3 & -0.4 \\ -0.6 & -0.8 & 0.5 & -0.3 \\ 0.1 & -0.1 & 0.6 & 0.1 \end{bmatrix} \qquad \mathbf{b_{xh}}=\begin{bmatrix} 0.0 \\ 0.0 \\ 0.0 \end{bmatrix}$$

$$\mathbf{X}=\begin{bmatrix} X_1 \\ X_2 \end{bmatrix}=\begin{bmatrix} 1. & 3. & 2. & 1. \\ 0. & 4. & 1. & 2. \end{bmatrix} \qquad \mathbf{h_0}=\begin{bmatrix} 0.0 \\ 0.0 \\ 0.0 \end{bmatrix}$$

$$\boldsymbol{h}_1=\begin{bmatrix} -0.833 \\ -0.98 \\ 0.8 \end{bmatrix}$$

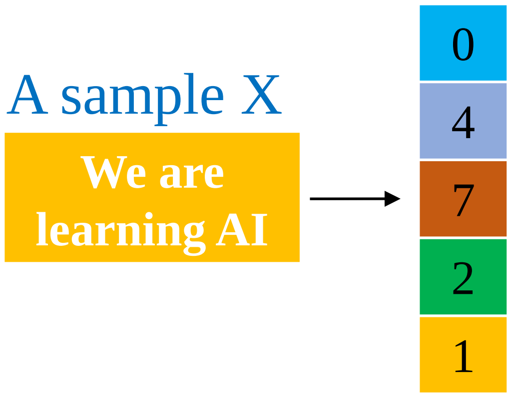
$$h_2 = \tanh(W_{xh}X_2 + b_{xh} + W_{hh}h_1 + b_{hh})$$

$$= \tanh\left(\begin{bmatrix} -0.2 & -0.4 & 0.3 & -0.4 \\ -0.6 & -0.8 & 0.5 & -0.3 \\ 0.1 & -0.1 & 0.6 & 0.1 \end{bmatrix}\begin{bmatrix} 0.0 \\ 4.0 \\ 1.0 \\ 2.0 \end{bmatrix} + \begin{bmatrix} 0.9 & -0.1 & 0.1 \\ -0.1 & -0.9 & 0.3 \\ 0.1 & -0.3 & -0.9 \end{bmatrix}\begin{bmatrix} -0.833 \\ -0.98 \\ 0.8 \end{bmatrix} + \begin{bmatrix} 0.0 \\ 0.0 \\ 0.0 \end{bmatrix}\right)$$

$$= \tanh\left(\begin{bmatrix} -2.67 \\ -2.09 \\ -0.109 \end{bmatrix}\right) = \begin{bmatrix} -0.99 \\ -0.97 \\ -0.109 \end{bmatrix}$$

RNN Models

❖ Implementation



(L, H_{out}) $\begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \end{bmatrix}$

Batch: (L, N, H_{out})

```
Parameter containing:
tensor([[-0.1882,  0.5530,  1.6267,  0.7013],
        [ 1.7840, -0.8278, -0.2701,  1.3586],
        [ 1.0281, -1.9094,  0.3182,  0.4211],
        [-1.3083, -0.0987,  0.7647, -0.3680],
        [ 0.2293,  1.3255,  0.1318,  2.0501],
        [ 0.4058, -0.6624, -0.8745,  0.7203],
        [ 0.5582,  0.0786, -0.6817,  0.6902],
        [ 0.4309, -1.3067, -0.8823,  1.5977]],
```

```python
data = torch.tensor([0, 4, 7, 2, 1],
                    dtype=torch.long)
data_embedding = embedding(data)
print(data_embedding)
```

```
tensor([[-0.1882,  0.5530,  1.6267,  0.7013],
        [ 0.2293,  1.3255,  0.1318,  2.0501],
        [ 0.4309, -1.3067, -0.8823,  1.5977],
        [ 1.0281, -1.9094,  0.3182,  0.4211],
        [ 1.7840, -0.8278, -0.2701,  1.3586]],
```

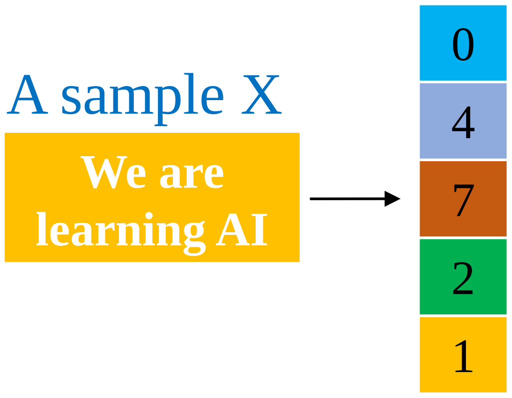
```python
embed_dim = 4
hidden_dim = 3
rnn = nn.RNN(embed_dim,
             hidden_dim,
             batch_first=True)

rnn_output, rnn_hidden = rnn(data_embedding)
rnn_output
```

```
tensor([[ 0.2973,  0.2388, -0.8593],
        [-0.3616,  0.6365, -0.8125],
        [-0.3297,  0.9502,  0.6365],
        [-0.1884,  0.9009,  0.4258],
        [-0.6319,  0.9454,  0.2323]], grad_fn=<
```

RNN Models

❖ Implementation



(L, H_{out}) $\begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \end{bmatrix}$

Batch: (N, L, H_{out})

$$\begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \end{bmatrix} = \begin{bmatrix} [\,0.2973 & 0.2388 & -0.8593] \\ [-0.3616 & 0.6365 & -0.8125] \\ [-0.3297 & 0.9502 & 0.6365] \\ [-0.1884 & 0.9009 & 0.4258] \\ [-0.6319 & 0.9454 & 0.2323] \end{bmatrix}$$

```python
test_data = data_embedding.reshape(1, 5, 4)
print(test_data.shape)
```

```
torch.Size([1, 5, 4])
```

```python
rnn_output, rnn_hidden = rnn(test_data)

# (N, L, H_out)
print(rnn_output.shape)
print(rnn_output)
print(rnn_output[:, -1, :])
```

```
torch.Size([1, 5, 3])
tensor([[[ 0.2973,  0.2388, -0.8593],
         [-0.3616,  0.6365, -0.8125],
         [-0.3297,  0.9502,  0.6365],
         [-0.1884,  0.9009,  0.4258],
         [-0.6319,  0.9454,  0.2323]]], grad_fn=<T
tensor([[-0.6319,  0.9454,  0.2323]], grad_fn=<Sli
```

```python
# (num_layers, N, H_out)
print(rnn_hidden.shape)
print(rnn_hidden)
print(rnn_hidden[-1, :, :])
```

```
torch.Size([1, 1, 3])
tensor([[[-0.6319,  0.9454,  0.2323]]], grad_fn=<S
tensor([[-0.6319,  0.9454,  0.2323]], grad_fn=<Sli
```

Stack of RNNs

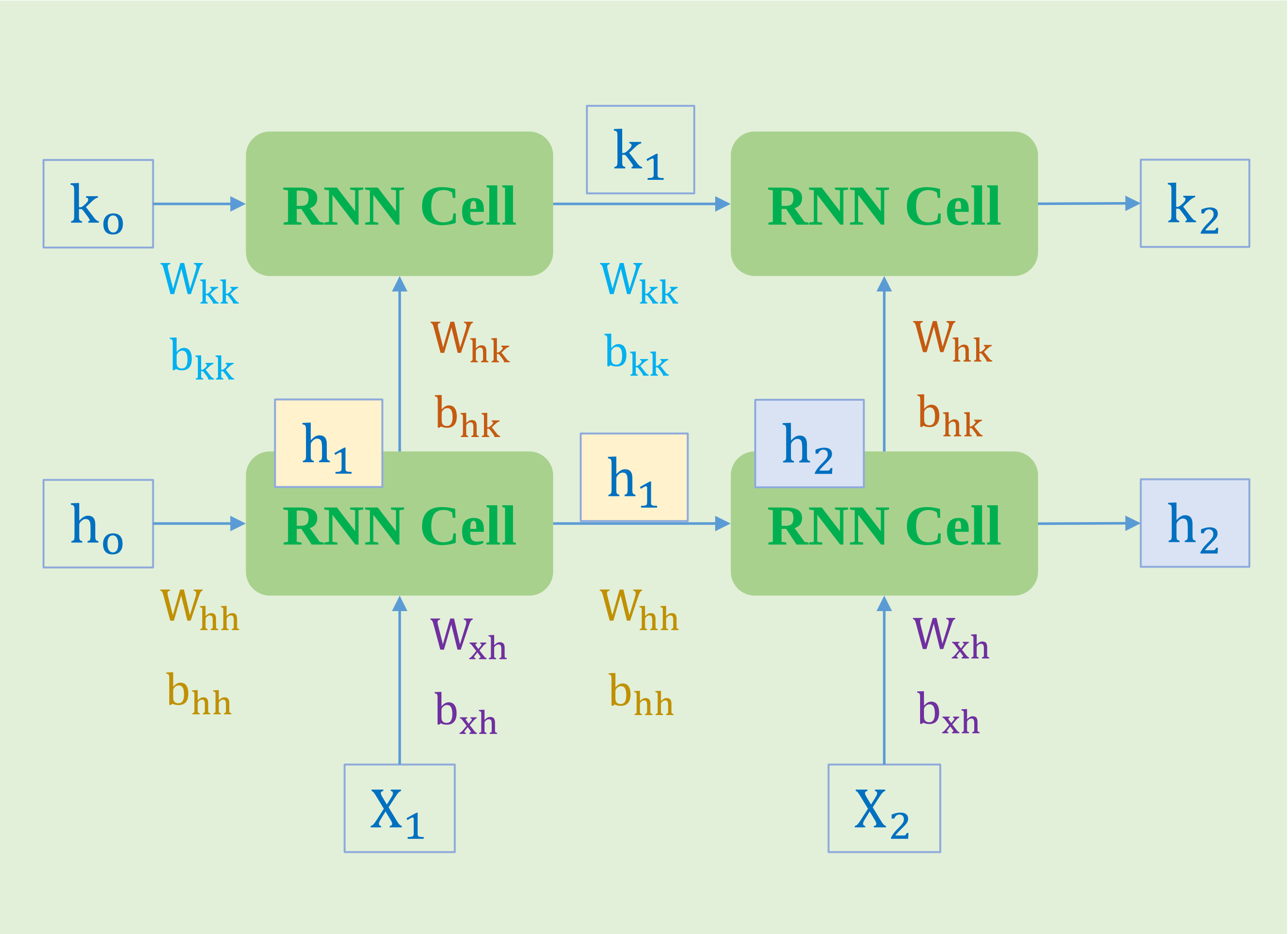
- **Recurrent Neural Networks (RNNs)**
  - **Two layers**

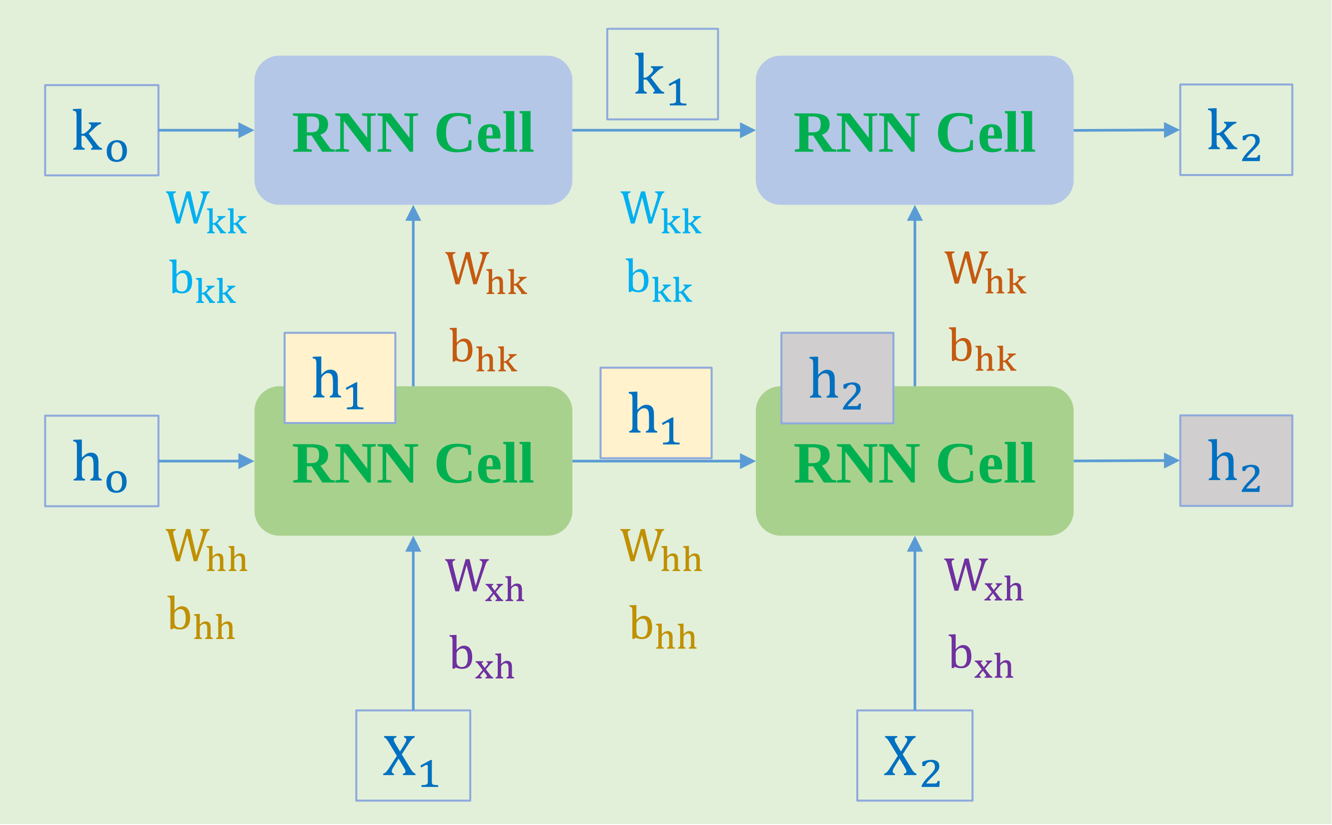
$$k_1 = \tanh(W_{hk}h_1 + b_{hk} + W_{kk}k_0 + b_{kk})$$

$$k_2 = \tanh(W_{hk}h_2 + b_{hk} + W_{kk}k_1 + b_{kk})$$

$$h_1 = \tanh(W_{xh}X_1 + b_{xh} + W_{hh}h_0 + b_{hh})$$

$$h_2 = \tanh(W_{xh}X_2 + b_{xh} + W_{hh}h_1 + b_{hh})$$





$$\mathbf{W_{hh}} = \begin{bmatrix} 0.9 & -0.1 & 0.1 \\ -0.1 & -0.9 & 0.3 \\ 0.1 & -0.3 & -0.9 \end{bmatrix} \qquad \mathbf{W_{kk}} = \begin{bmatrix} 0.1 & 0.9 \\ 0.9 & -0.1 \end{bmatrix}$$

$$\mathbf{W_{xh}} = \begin{bmatrix} -0.2 & -0.4 & 0.3 & -0.4 \\ -0.6 & -0.8 & 0.5 & -0.3 \\ 0.1 & -0.1 & 0.6 & 0.1 \end{bmatrix} \qquad \mathbf{W_{hk}} = \begin{bmatrix} -1.1 & 0.3 & -0.1 \\ 0.1 & -0.2 & 0.4 \end{bmatrix}$$

$$\mathbf{b_{xh}} = \mathbf{b_{hh}} = \begin{bmatrix} 0.0 \\ 0.0 \\ 0.0 \end{bmatrix} \qquad \mathbf{b_{hk}} = \mathbf{b_{kk}} = \begin{bmatrix} 0.0 \\ 0.0 \\ 0.0 \end{bmatrix}$$

$$k_1 = \tanh(W_{hk}h_1 + b_{hk} + W_{kk}k_0 + b_{kk}) = \begin{bmatrix} 0.49 \\ 0.407 \end{bmatrix}$$

$$k_2 = \tanh(W_{hk}h_2 + b_{hk} + W_{kk}k_1 + b_{kk}) = \begin{bmatrix} 0.84 \\ 0.42 \end{bmatrix}$$

$$h_1 = \tanh(W_{xh}X_1 + b_{xh} + W_{hh}h_0 + b_{hh}) = \begin{bmatrix} -0.833 \\ -0.98 \\ 0.8 \end{bmatrix}$$

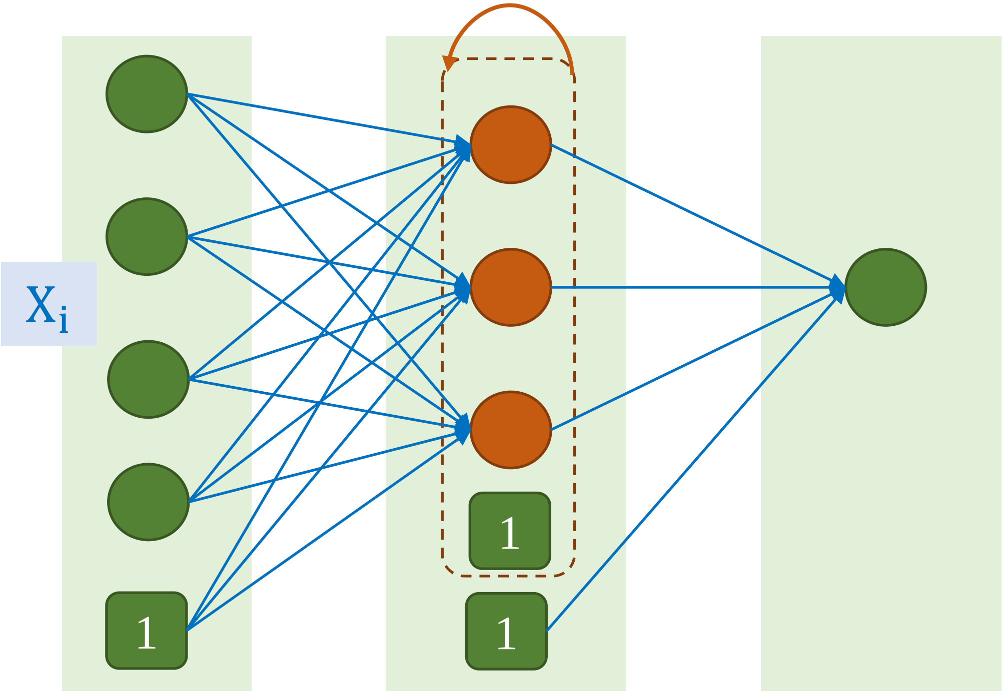
$$h_2 = \tanh(W_{xh}X_2 + b_{xh} + W_{hh}h_1 + b_{hh}) = \begin{bmatrix} -0.99 \\ -0.97 \\ -0.109 \end{bmatrix}$$

Text Deep Models

❖ Recurrent Neural Networks (RNN)

$$\mathbf{W_{hh}} = \begin{bmatrix} 0.226 & -0.068 & -0.971 \\ -0.973 & -0.014 & -0.226 \\ -0.001 & -0.997 & 0.070 \end{bmatrix} \qquad \mathbf{b}_{xh} = \mathbf{b}_{hh} = \begin{bmatrix} 0.0 \\ 0.0 \\ 0.0 \end{bmatrix}$$

$$\mathbf{W_{xh}} = \begin{bmatrix} -0.436 & 0.373 & -0.032 & 0.403 \\ -0.452 & 0.296 & -0.609 & -0.643 \\ -0.761 & 0.266 & -0.526 & -0.703 \end{bmatrix}$$

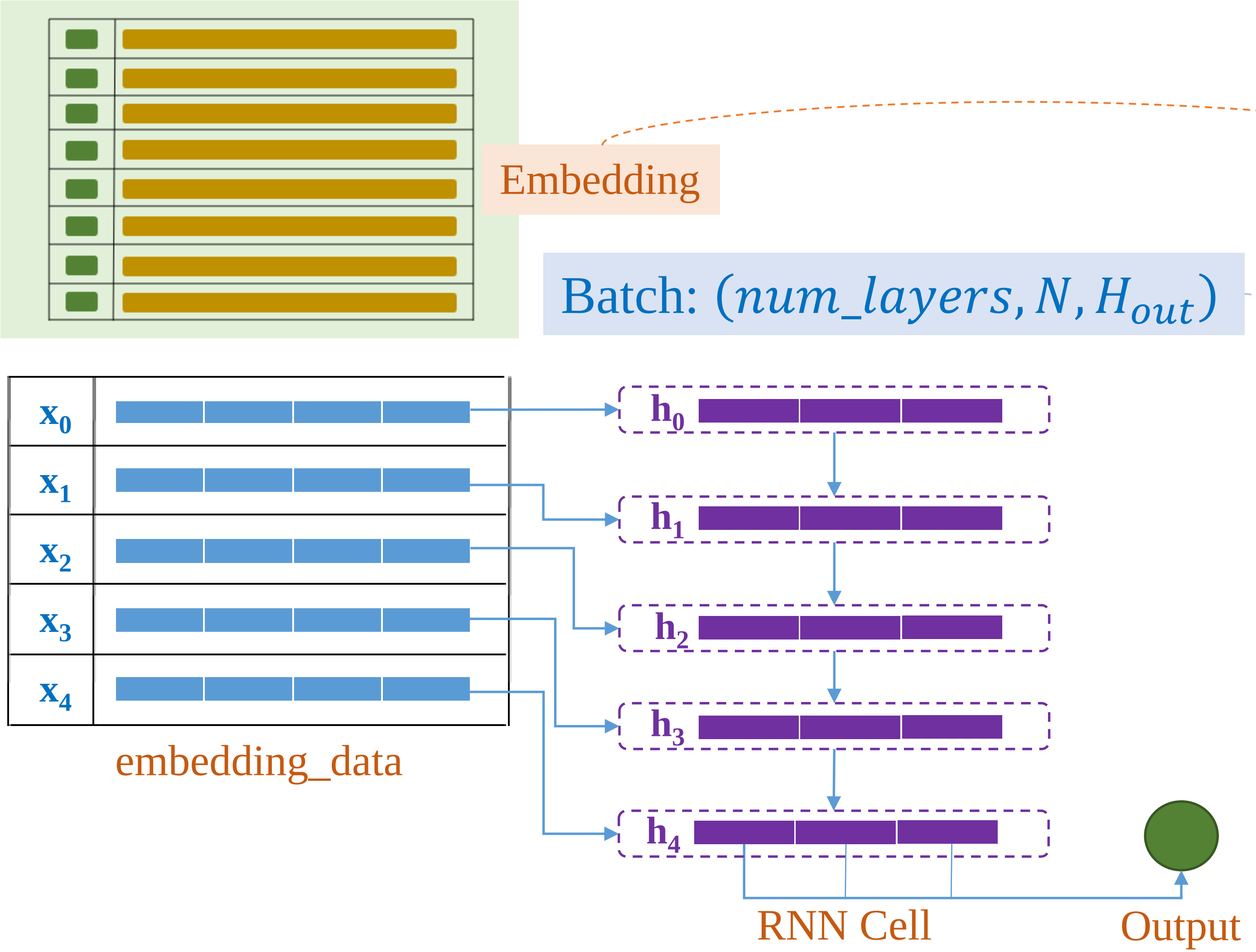



```python
class TextClsModel(nn.Module):
    def __init__(self, vocab_size=8,
                 emb_dim=4, hidden_dim=3):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size,
                                      emb_dim)
        self.rnn = nn.RNN(emb_dim, hidden_dim,
                          batch_first=True)
        self.fc = nn.Linear(hidden_dim, 1)

    def forward(self, x):
        x = self.embedding(x)
        _, h_rnn = self.rnn(x)
        last_h_rnn = h_rnn[-1,:,:]
        x = self.fc(last_h_rnn)
        return x
```

$$\mathbf{h}_t = \tanh(\mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{b}_{hh} + \mathbf{W}_{xh}\mathbf{x}_t + \mathbf{b}_{xh})$$

$$\hat{y} = \text{sigmoid}(\mathbf{W}_D\mathbf{h}_t + \mathbf{b}_D)$$




```python
class TextClsModel(nn.Module):
    def __init__(self, vocab_size=8,
                 emb_dim=4, hidden_dim=3):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size,
                                      emb_dim)
        self.rnn = nn.RNN(emb_dim, hidden_dim,
                          batch_first=True)
        self.fc = nn.Linear(hidden_dim, 1)

    def forward(self, x):
        x = self.embedding(x)
        _, h_rnn = self.rnn(x)
        last_h_rnn = h_rnn[-1,:,:]
        x = self.fc(last_h_rnn)
        return x

model = TextClsModel(vocab_size=8,
                     emb_dim=4,
                     hidden_dim=3)
```

```
==========================================================================
Layer (type:depth-idx)                   Output Shape              Param
==========================================================================
├─Embedding: 1-1                         [-1, 5, 4]                32
├─RNN: 1-2                               [-1, 5, 3]                27
├─Linear: 1-3                            [-1, 1]                   4
==========================================================================
```

```python
from torchsummary import summary
seq_length = 5
random_tensor = torch.randint(low=0, high=8,
                              size=(64, seq_length),
                              dtype=torch.long)
summary(model, random_tensor)
```