AI VIETNAM
All-in-One Course

Steps to Sequence and Time-series Data

Quang-Vinh Dinh
Ph.D. in Computer Science

Year 2022

(Module 5 – Sequence Data - Friday)

AI VIETNAM
All-in-One Course

Schedule

❖ What's next?

25/12/2022	TA-Exercise
28/12/2022	Project for image data (OCR)
30/12/2022	Project for image data (OCR)
4/1/2023	Project for text data
7/1/2023	Project for time-series data

MODULE 6 (Deep Learning for Images)
CÔNG VIỆC
CNN Generalization - Noise Adding, Dropout, …
CNN Generalization - BN, Skip Connection, …
TA-Exercise
Domain Conversion - Denoising and Segmentation
Domain Conversion - Colorization and Super-resolution
TA-Exercise
Object Detection
Object Detection
TA-Exercise
Object Detection
Object Detection
TA-Exercise
Project: Multi-Tasking CNN
Exam

MODULE 7 (Deep Learning for Text)
CÔNG VIỆC
RNN/LSTM/GRU
Text Classification and POS Tagging
TA-Exercise
Transformer
Text Classification and POS Tagging
TA-Exercise
Transformer - Machine Translation
Transformer - Machine Translation
TA-Exercise
Transformer - Text Generation
Transformer - Image Captioning
TA-Exercise
Project: English Writing Evaluation
Exam

MODULE 8 (Data Science)
CÔNG VIỆC
Data Visualization (including PCA)
Data Visualization
TA-Exercise
Data Analysis Using Pandas
Data Analysis Using Pandas
TA-Exercise
Statistics-based Models for Time-series data
Statistics-based Models for Time-series data
TA-Exercise
SOTA Models for Time-series data
SOTA Models for Time-series data
TA-Exercise
Deep Learning for Audio Classification
Deep Learning for Audio Classification
TA-Exercise
Time-series Project: Air Pollution Prediction
Audio Project: Speech Recognition
Exam

Outline

- Dealing with Text
- Text Classification
- RNN and LSTM
- Bidirectional RNN/LSTM
- RNN/LSTM for Time-series data
- Introduction to NLP (Optional)

Image Data

Cifar-10 dataset

- Color images
- Resolution=32x32
- Training set: 50000 samples
- Testing set: 10000 samples

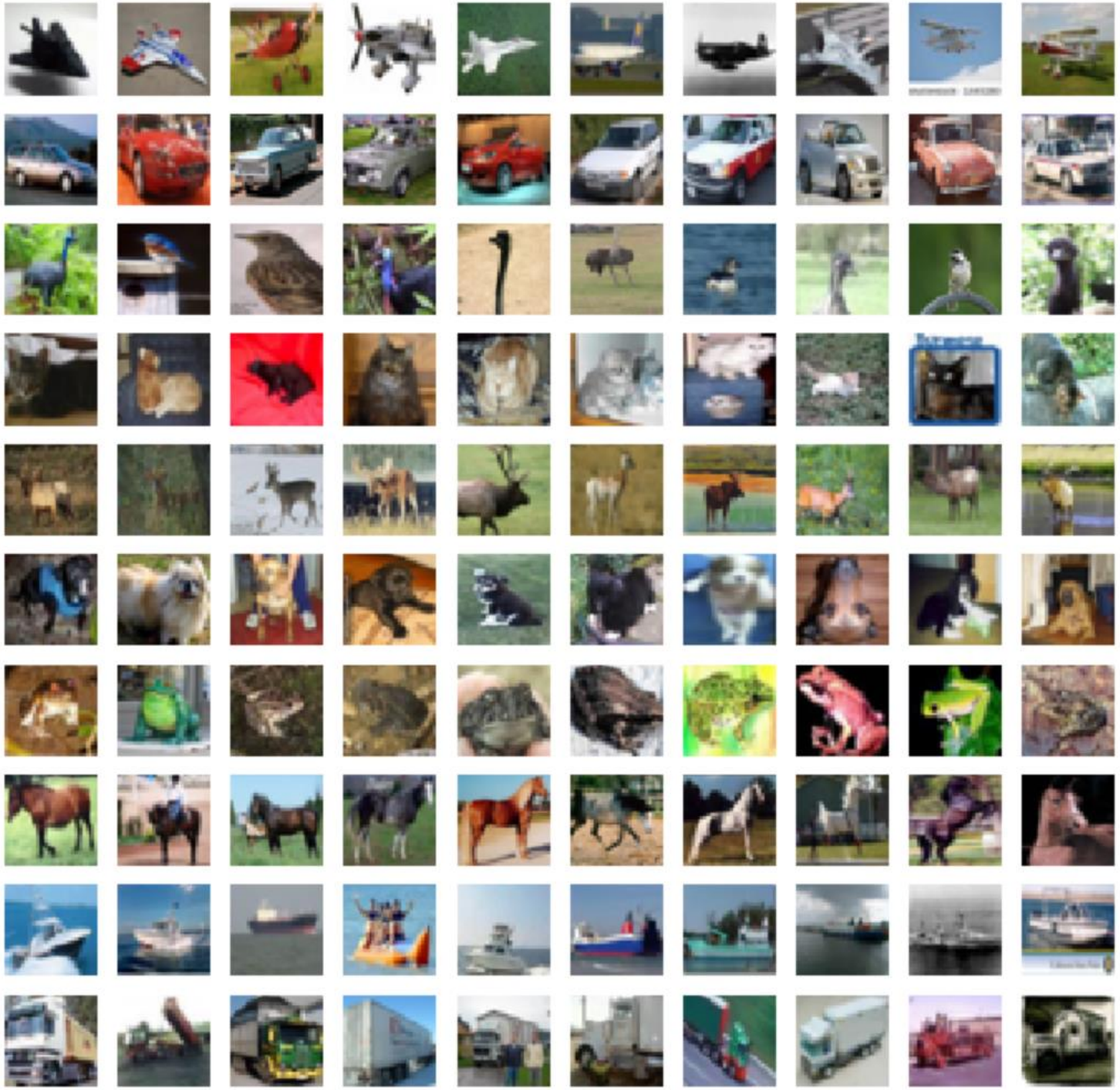
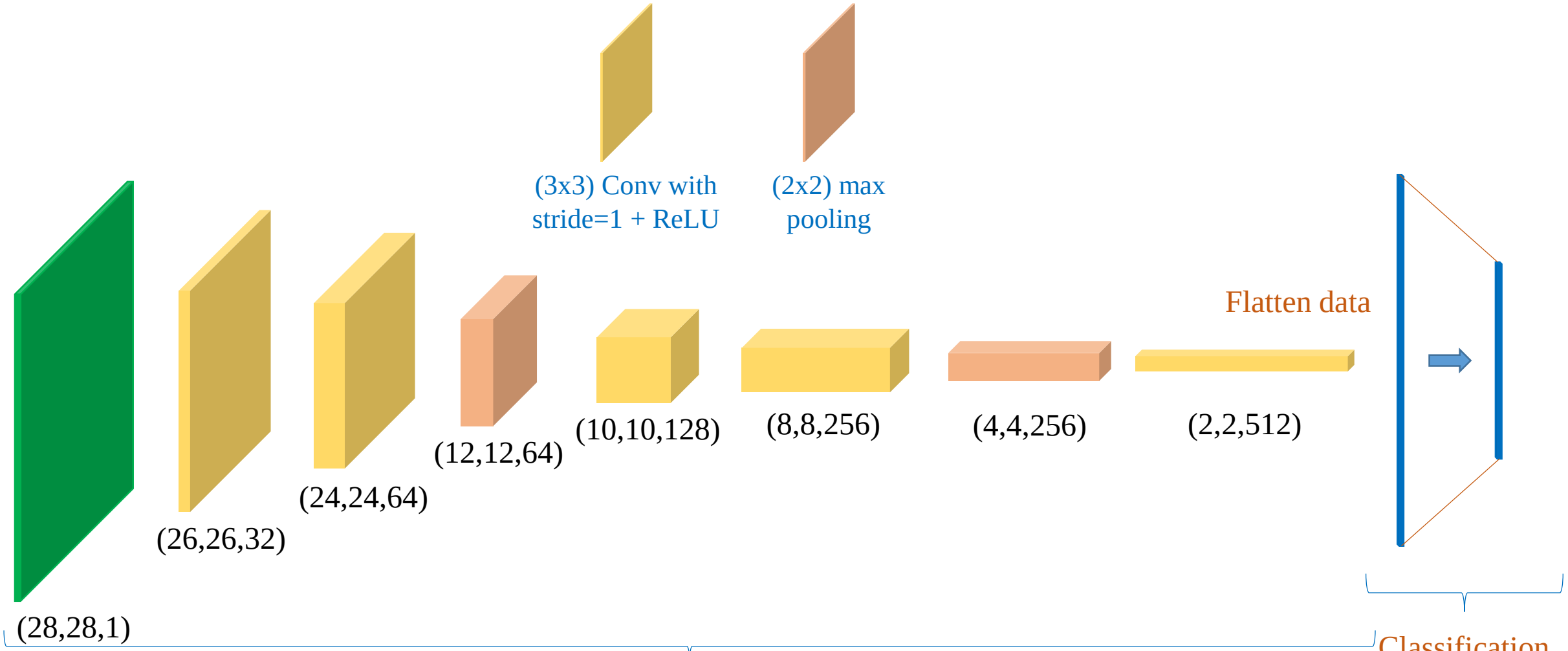


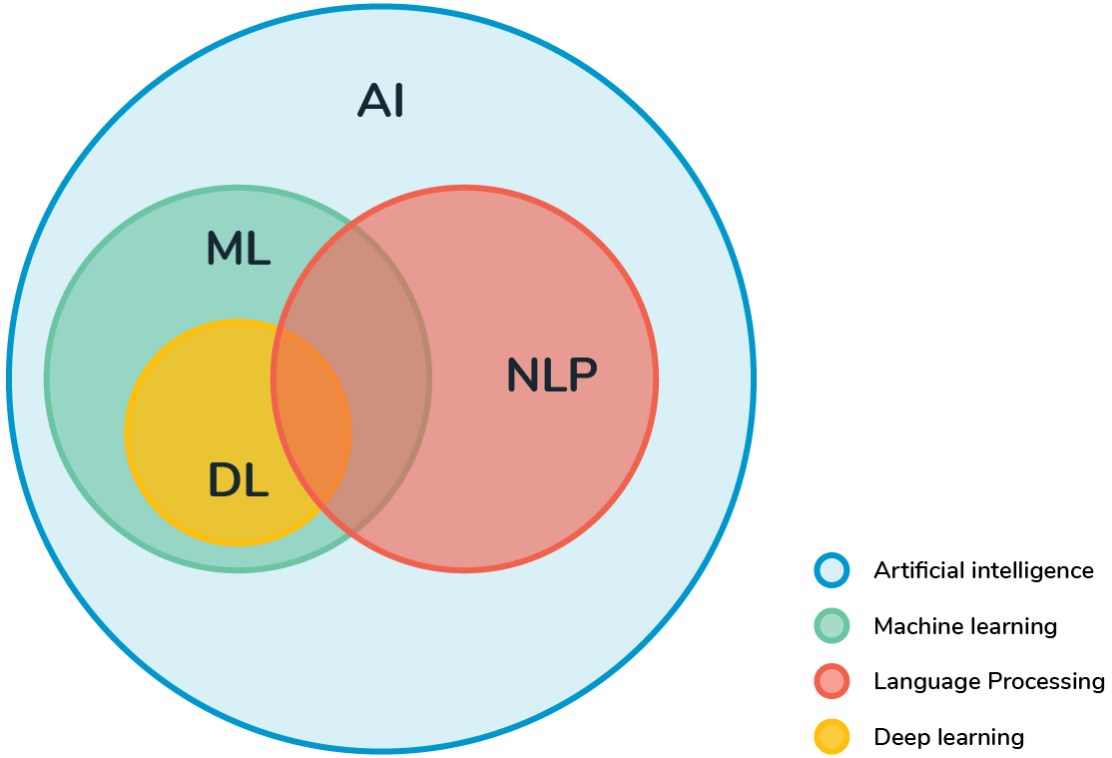
Image Data

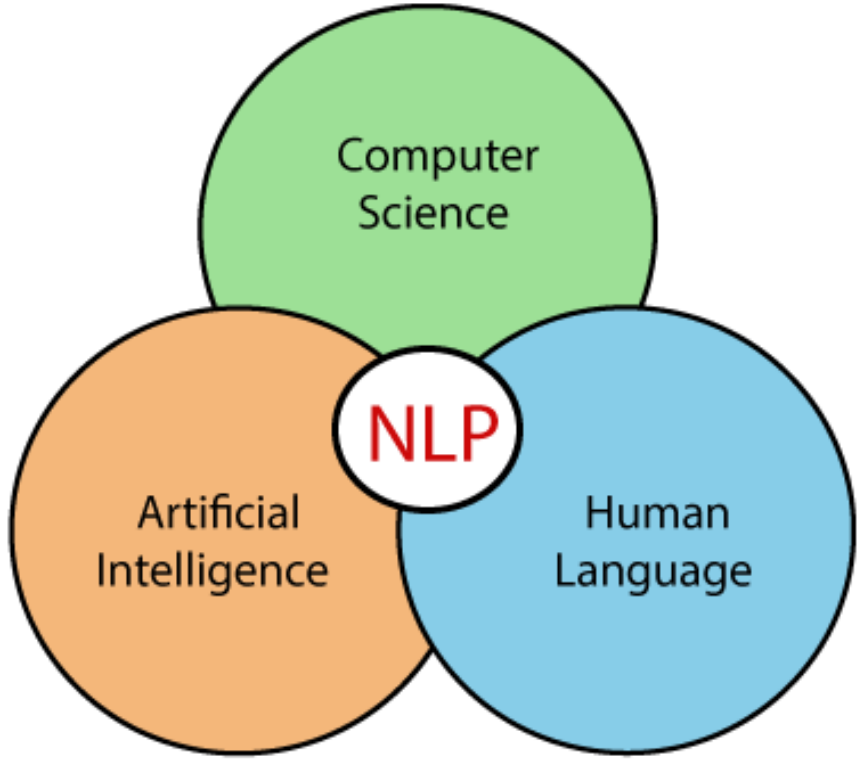
❖ Example



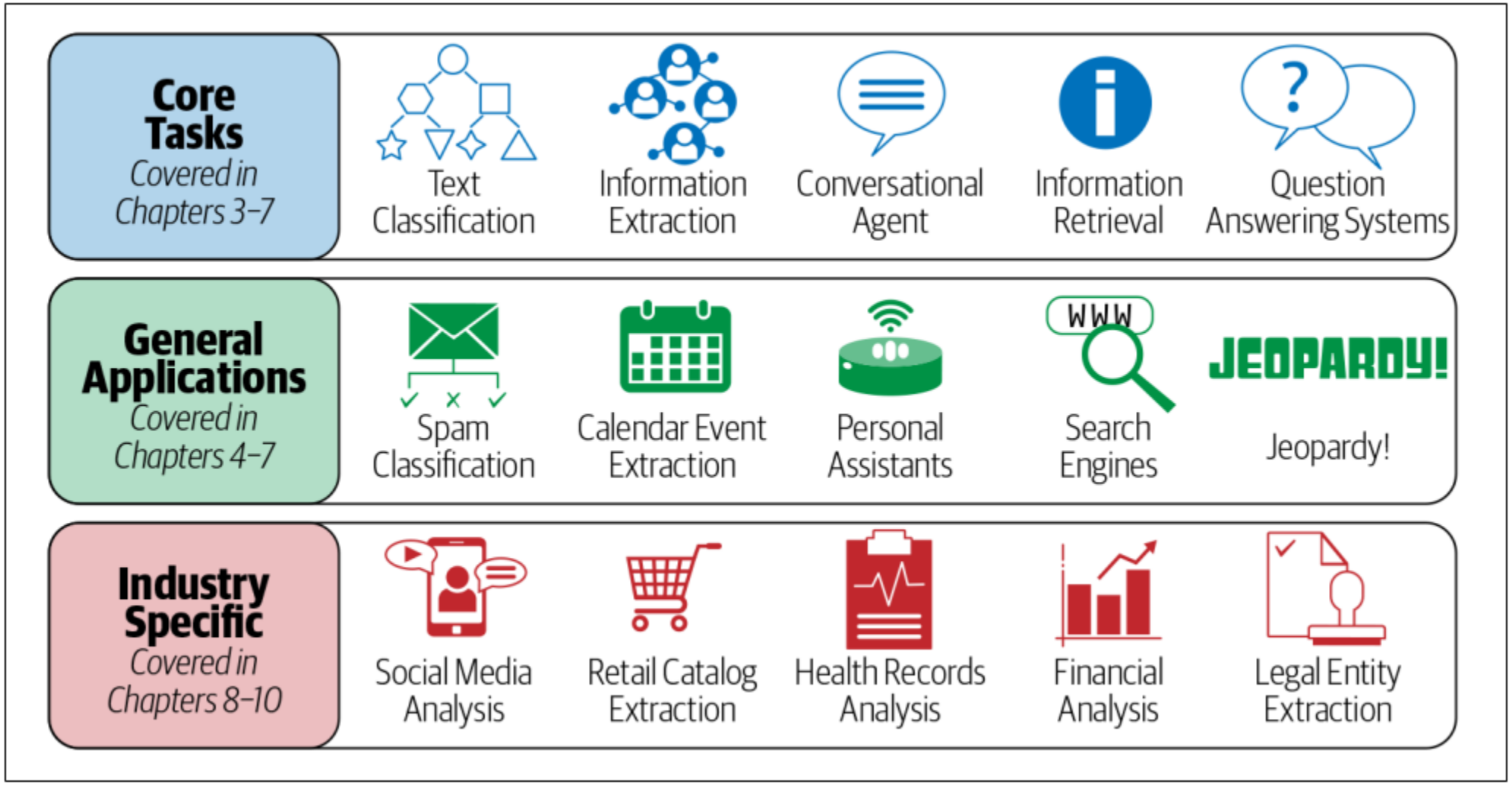
Natural Language Processing

❖ Introduction





NLP Applications



NLP Applications

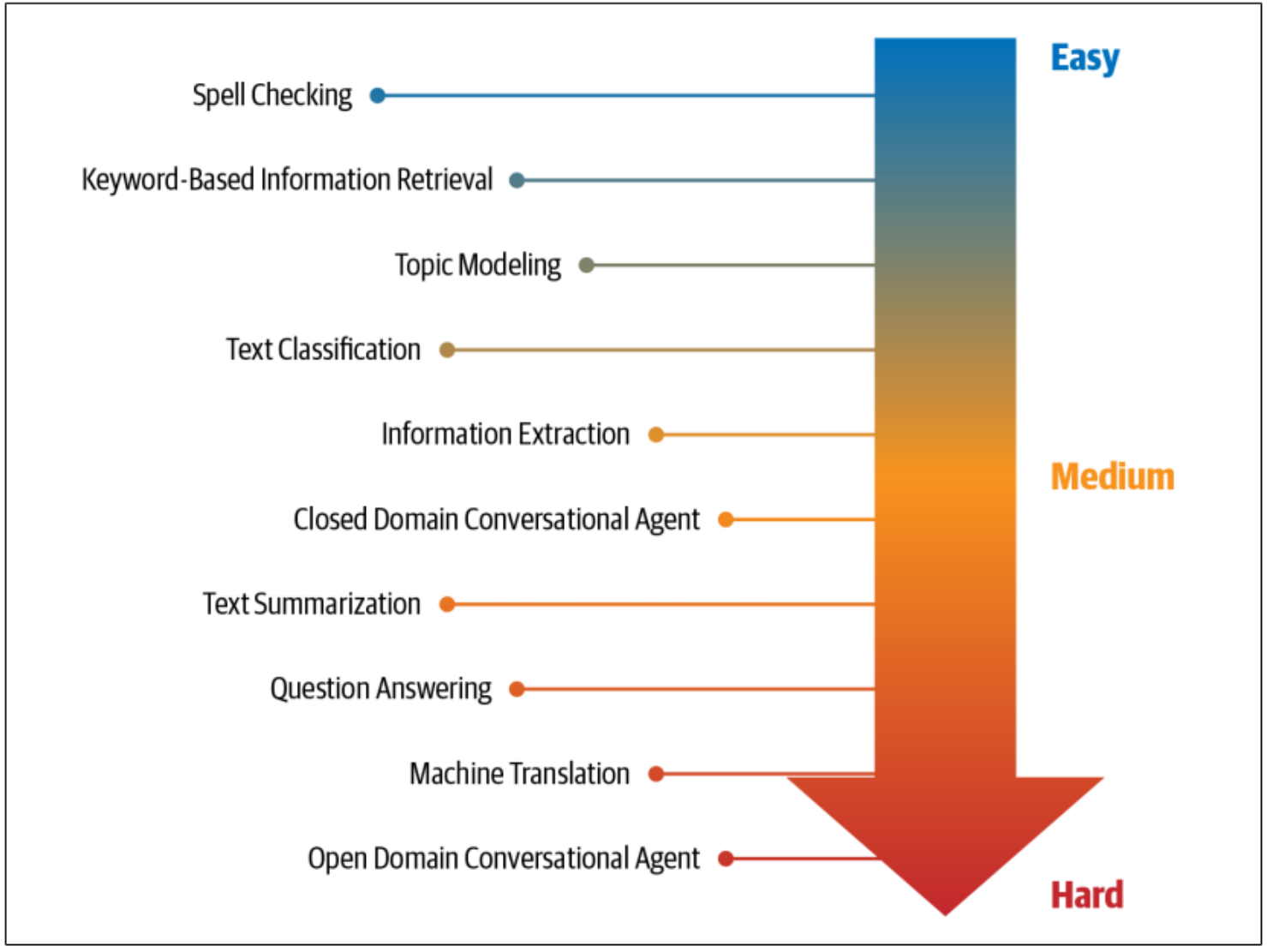
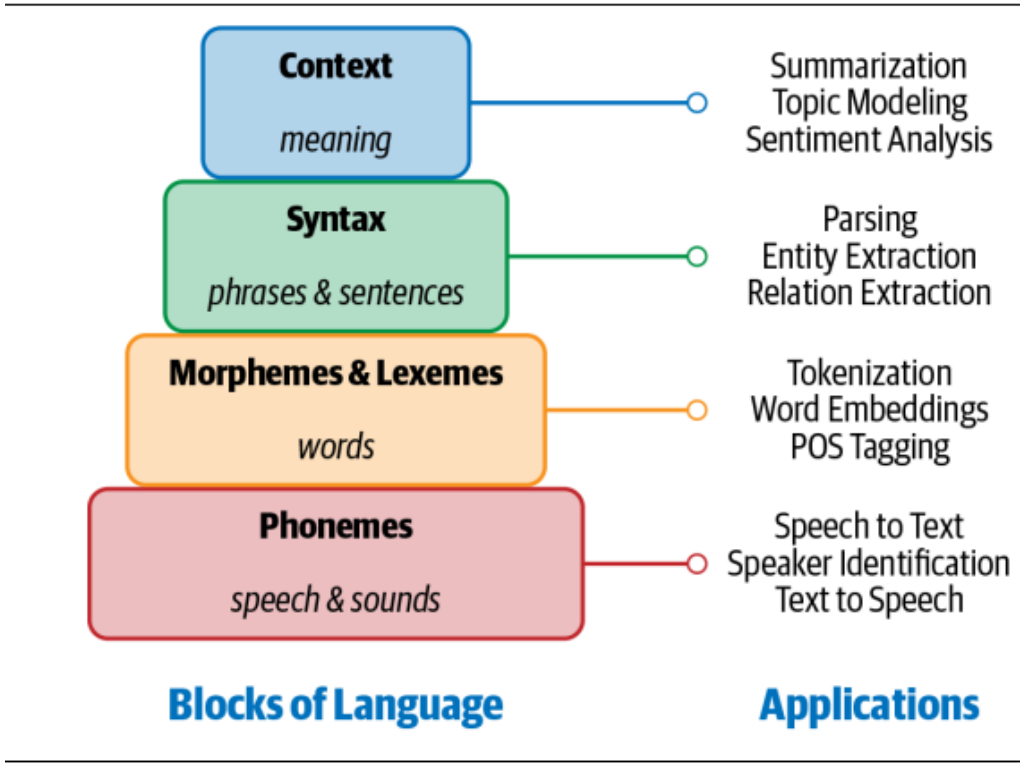


Figure 1-2. NLP tasks organized according to their relative difficulty



Building blocks of language and their applications

Practical Natural Language Processing by Sowmya Vajjala, Bodhisattwa Majumder, Anuj Gupta, Harshit Surana

Applications of Text Analysis

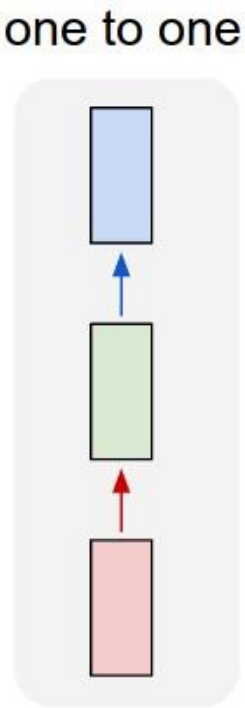


Image classification

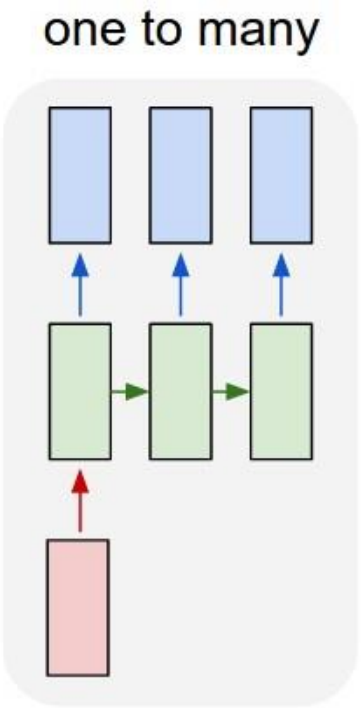
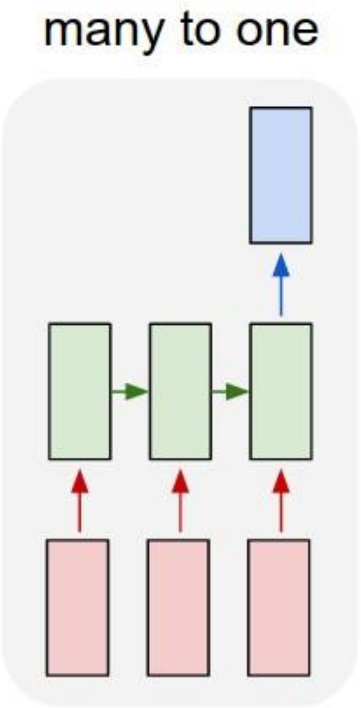
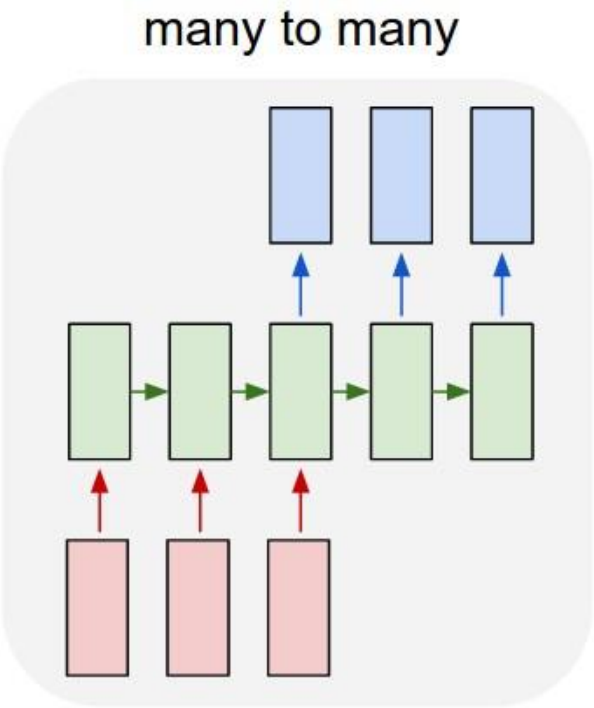


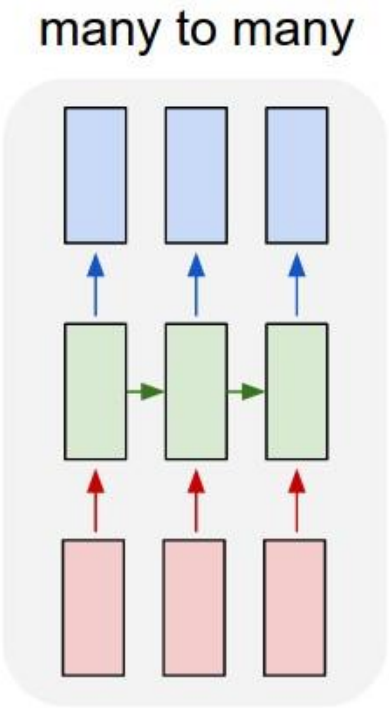
Image Captioning



Text classification



Machine Translation
Recognition



POS Tagging

Source: The Unreasonable Effectiveness of Recurrent Neural Networks

illustration

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_dataset_from_directory

```
main_directory/
...class_a/
......a_text_1.txt
......a_text_2.txt
...class_b/
......b_text_1.txt
......b_text_2.txt
```

```
keras.utils.text_dataset_from_directory(
                            directory,
                            labels='inferred',
                            label_mode='int',
                            class_names=None,
                            batch_size=32,
                            max_length=None,
                            shuffle=True,
                            seed=None,
                            validation_split=None,
                            subset=None,
                            follow_links=False
                          )
```

- 'int': means that the labels are encoded as integers (e.g. for `sparse_categorical_crossentropy` loss).

'categorical' means that the labels are encoded as a categorical vector (e.g. for categorical_crossentropy loss).

'binary' means that the labels (there can be only 2) are encoded as float32 scalars with values 0 or 1 (e.g. for binary_crossentropy).

None (no labels).

Only valid if "labels" is "inferred". This is the explicit list of class names (must match names of subdirectories)

1.Imdb_dataset

Text Representation

Binary BoW: 1 for word appearing
0 for word non-appearing

BoW: frequency of each word occurring

❖ Bag-of-Words (BoW)

vocabulary_size = 9

Corpus

doc1 = "deep learning book"

doc2 = "machine learning algorithm"

doc3 = "learning ai from scratch"

doc4 = "ai vietnam"

Tokenization →

['deep', 'learning', 'book']

['machine', 'learning', 'algorithm']

['learning', 'ai', 'from', 'scratch']

['ai', 'vietnam']

Vocabulary =	deep	learning	book	machine	algorithm	ai	from	scratch	vietnam

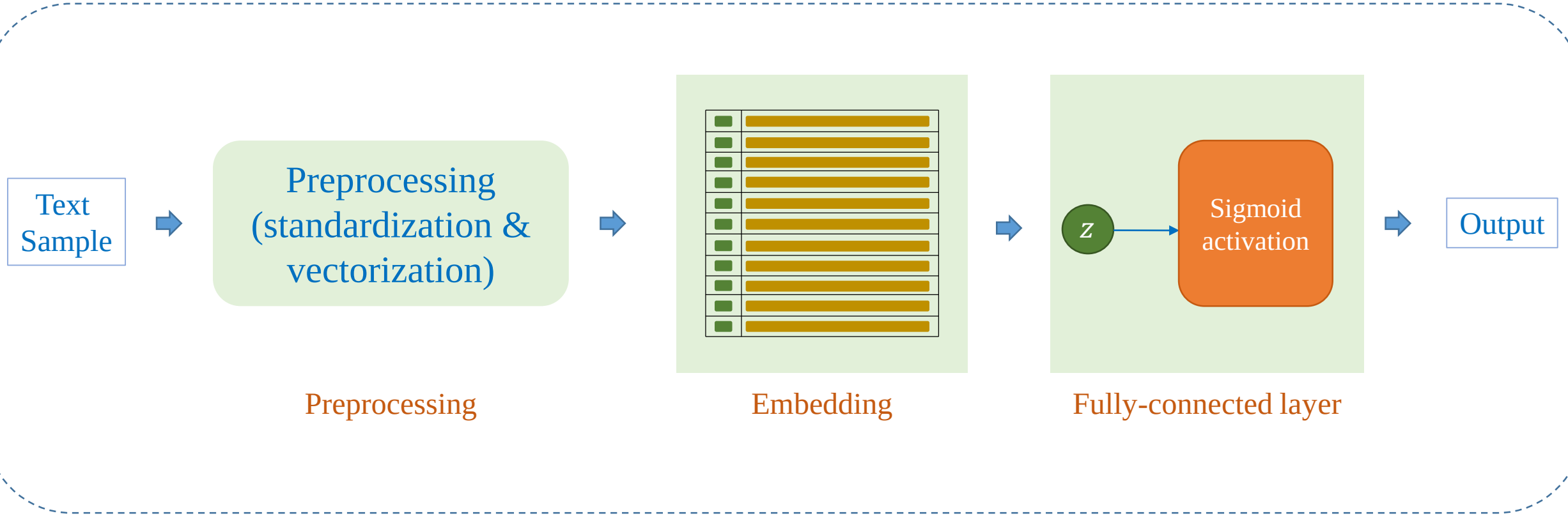
same size

☞ **Given a string =** "vietnam machine learning deep learning book"

	deep	learning	book	machine	algorithm	ai	from	scratch	vietnam
BoW	1	2	1	1	0	0	0	0	1
Binary BoW	1	1	1	1	0	0	0	0	1

Text Classification

❖ Simple approach



Text Vectorization

A preprocessing layer which maps text features to integer sequences

Example corpus: ["dog bites man", "man bites dog",
"dog eats meat", "dog and dog are friends"]

Build vocabulary from corpus (vocab size = 7)

+ Build vocabulary:

word	dog	man	bites	meat	friends	eats	are	and
freq.	5	2	2	1	1	1	1	1

+ Select 5 words with the most frequency

dog	man	bites	meat	friends

+ Append padding and unknown word

<pad>	<UNK>	dog	man	bites	meat	friends

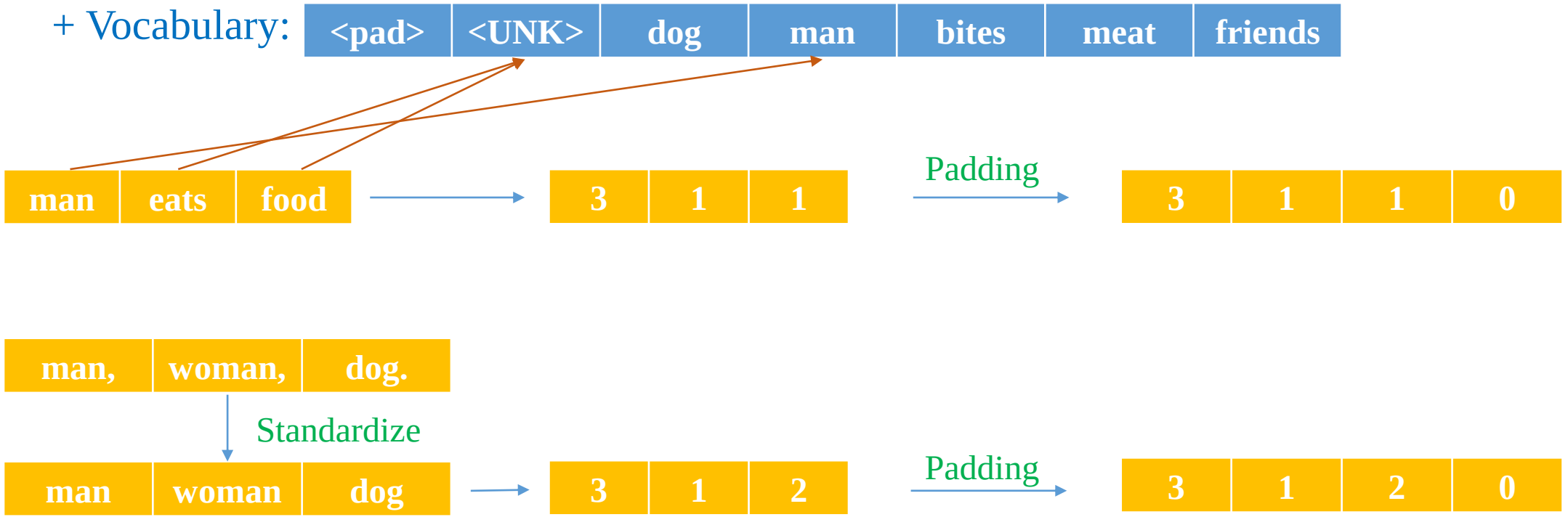
Text Vectorization

❖ Example

Corpus: ["dog bites man", "man bites dog",
"dog eats meat", "dog and dog are friends"]

Transform text into sequence

+ Vocabulary:



Text Vectorization

A preprocessing layer which maps text features to integer sequences

Example corpus: [“dog bites man”, “man bites dog”,
“dog eats meat”, “dog and dog are friends”]

(1) Define parameters and vectorization layer

```
max_tokens = 7
seq_length = 4

vec_layer = TextVectorization(max_tokens=max_tokens,
                              standardize='lower_and_strip_punctuation',
                              output_mode="int",
                              output_sequence_length=seq_length)
```

(2) Build vocabulary from corpus (vocab size = 7)

```
vectorize_layer.adapt([sample1, sample2, ...])
```

```
vec_layer(['man eats food', 'man, woman, dog.'])
```

```
<tf.Tensor: shape=(2, 4), dtype=int64, numpy=
array([[3, 1, 1, 0],
       [3, 1, 2, 0]])>
```

Embedding

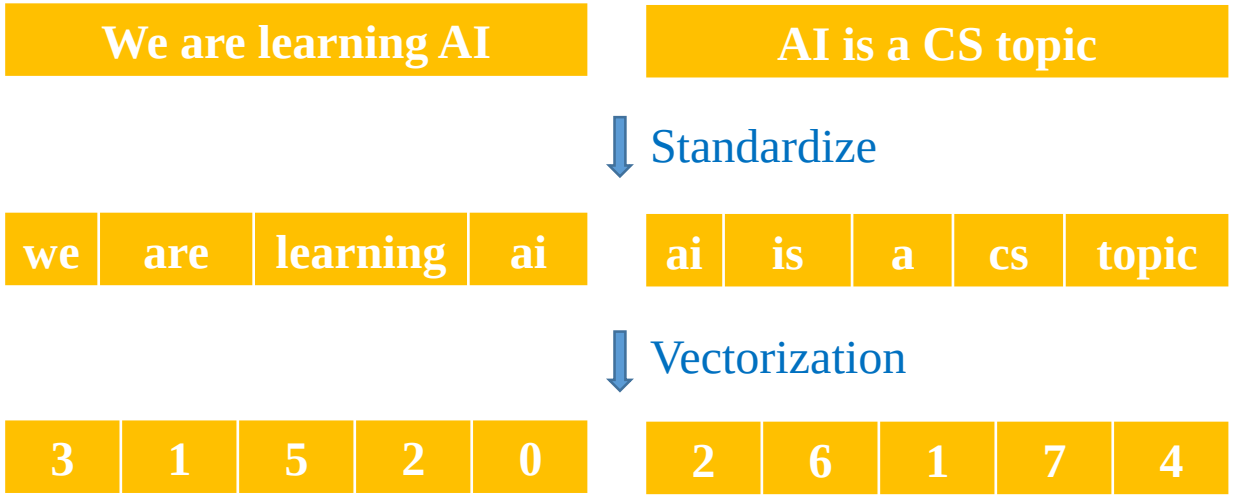
- Example corpus
  - sample1: ‘We are learning AI’
  - sample2: ‘AI is a CS topic’

(1) Build vocabulary from corpus

```
max_features = 8
embedding_dim = 4
sequence_length = 5

vectorize_layer = TextVectorization(
    max_tokens=max_features,
    output_mode="int",
    output_sequence_length=sequence_length,
)

vectorize_layer.adapt([sample1, sample2])
```

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

(2) Transform text into features




```
vectorize_layer.get_vocabulary()
```

```
['', '[UNK]', 'ai', 'we', 'topic', 'learning', 'is', 'cs']
```

```
sample1_vector = vectorize_layer(sample1)
print(sample1_vector)
```

```
tf.Tensor([3 1 5 2 0], shape=(5,), dtype=int64)
```

```
sample2_vector = vectorize_layer(sample2)
print(sample2_vector)
```

```
tf.Tensor([2 6 1 7 4], shape=(5,), dtype=int64)
```

Embedding Layer

(3) Embedding layer

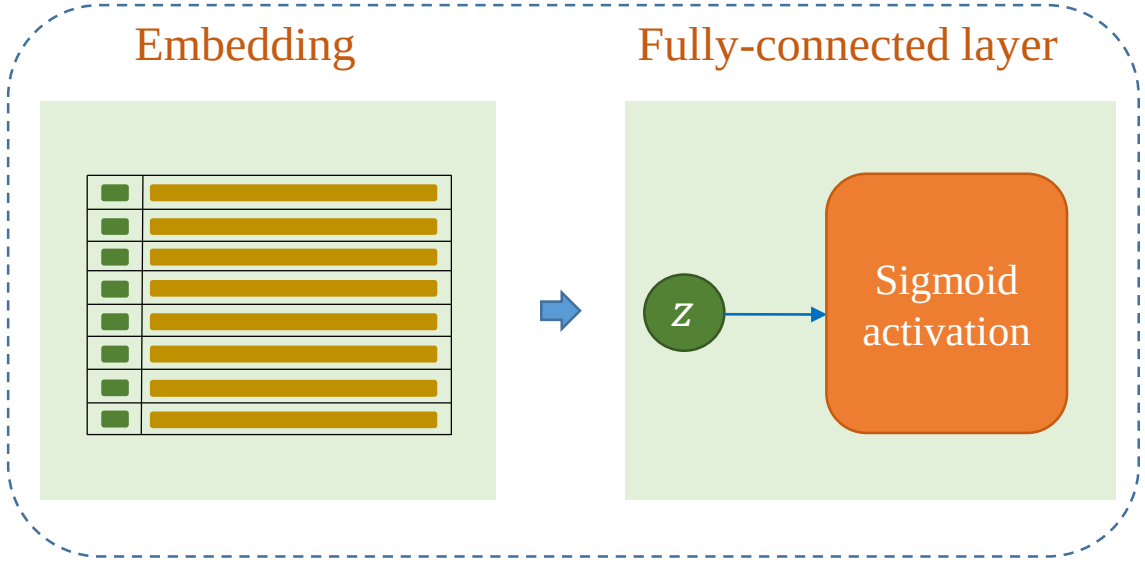
index	word
0	pad
1	[UNK]
2	AI
3	We
4	Topic
5	Learning
6	is
7	cs

```
1  print(model.layers[1].weights[0])
```

```
<tf.Variable 'embedding_3/embeddings:0' shape=(8, 4) dtype=float32,
array([[-0.03296757, -0.03054714,  0.01625914, -0.02955737],
       [ 0.01487434,  0.0038103 , -0.02253381,  0.04637194],
       [-0.00092559, -0.00625734,  0.03492227, -0.01756487],
       [-0.03613882, -0.0436982 , -0.04447322, -0.02143981],
       [-0.01163145, -0.0085723 ,  0.01227676, -0.0089262 ],
       [ 0.02521226,  0.04333058, -0.02373846,  0.02480527],
       [-0.02779102, -0.03064094, -0.04173347, -0.02026683],
       [ 0.04235573,  0.00204159, -0.04987942,  0.04008349]],
      dtype=float32)>
```

We are learning AI

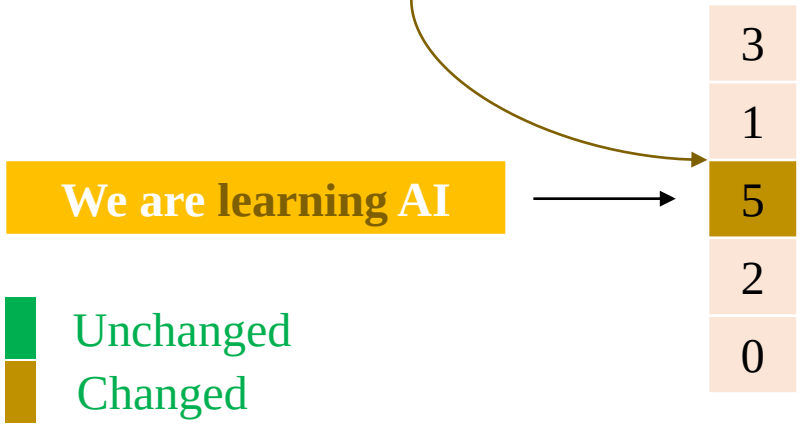
3
1
5
2
0



Embedding Layer

(3) Embedding layer

index	word
0	pad
1	[UNK]
2	AI
3	We
4	Topic
5	Learning
6	is
7	cs




```
1 print(model.layers[1].weights[0])
```

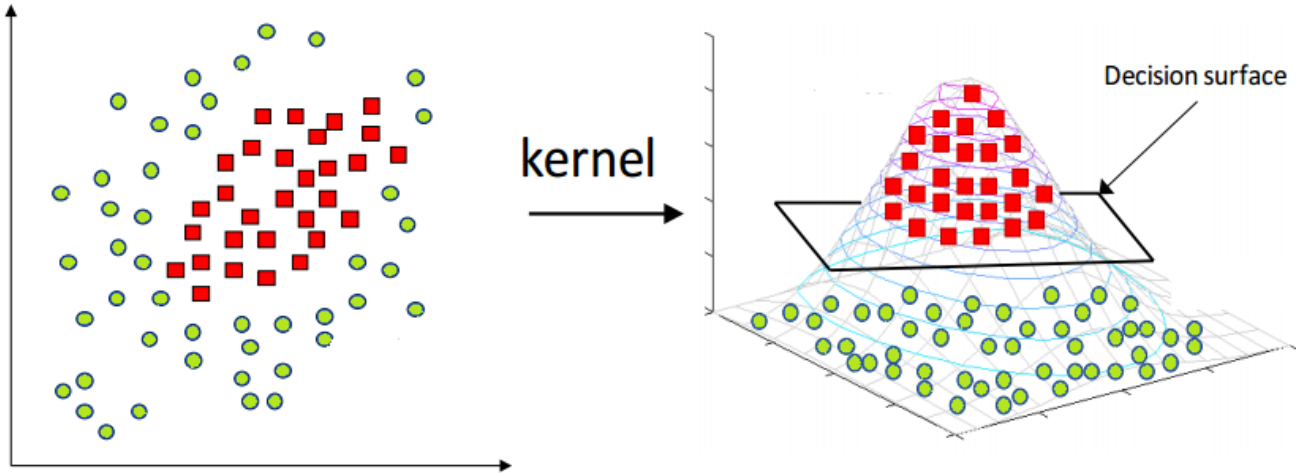
```
<tf.Variable 'embedding_3/embeddings:0' shape=(8, 4) dtype=float32,
array([[-0.03296757, -0.03054714,  0.01625914, -0.02955737],
       [ 0.01487434,  0.0038103 , -0.02253381,  0.04637194],
       [-0.00092559, -0.00625734,  0.03492227, -0.01756487],
       [-0.03613882, -0.0436982 , -0.04447322, -0.02143981],
       [-0.01163145, -0.0085723 ,  0.01227676, -0.0089262 ],
       [ 0.02521226,  0.04333058, -0.02373846,  0.02480527],
       [-0.02779102, -0.03064094, -0.04173347, -0.02026683],
       [ 0.04235573,  0.00204159, -0.04987942,  0.04008349]],
      dtype=float32)>
```

```
1 print(model.layers[1].weights[0])
```

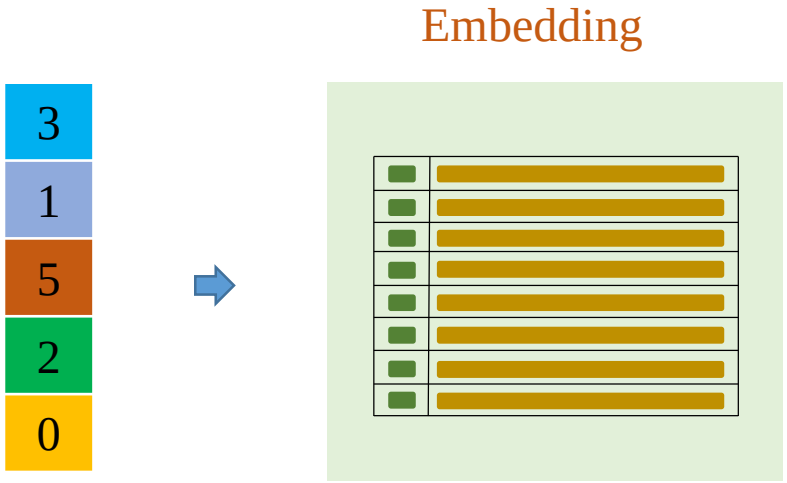
```
<tf.Variable 'embedding_3/embeddings:0' shape=(8, 4) dtype=float32,
array([[-0.03196755, -0.03154716,  0.01525914, -0.03055738],
       [ 0.01587436,  0.00281029, -0.0215338 ,  0.04737195],
       [-0.00192559, -0.00725714,  0.03592228, -0.01856488],
       [-0.03713876, -0.04469814, -0.04347321, -0.02043984],
       [-0.01163145, -0.0085723 ,  0.01227676, -0.0089262 ],
       [ 0.02621228,  0.04433057, -0.02473847,  0.02380528],
       [-0.02779102, -0.03064094, -0.04173347, -0.02026683],
       [ 0.04235573,  0.00204159, -0.04987942,  0.04008349]],
      dtype=float32)>
```

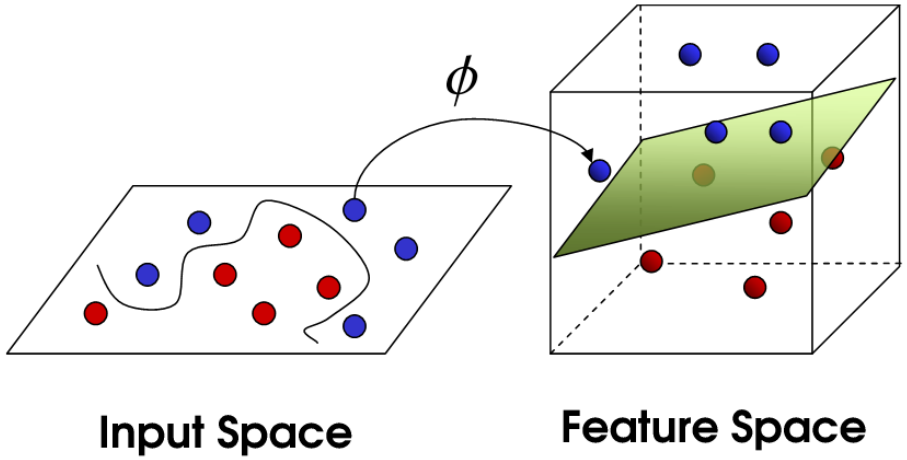
Word Embedding

❖ **Why?**



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





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

Embedding

Input shape
2D tensor with shape: (batch_size, input_length)

Output shape
3D tensor with shape: (batch_size, input_length, output_dim)

```
tf.keras.layers.Embedding(
                    input_dim,
                    output_dim,
                    embeddings_initializer="uniform",
                    embeddings_regularizer=None,
                    activity_regularizer=None,
                    embeddings_constraint=None,
                    mask_zero=False,
                    input_length=None
                )
```

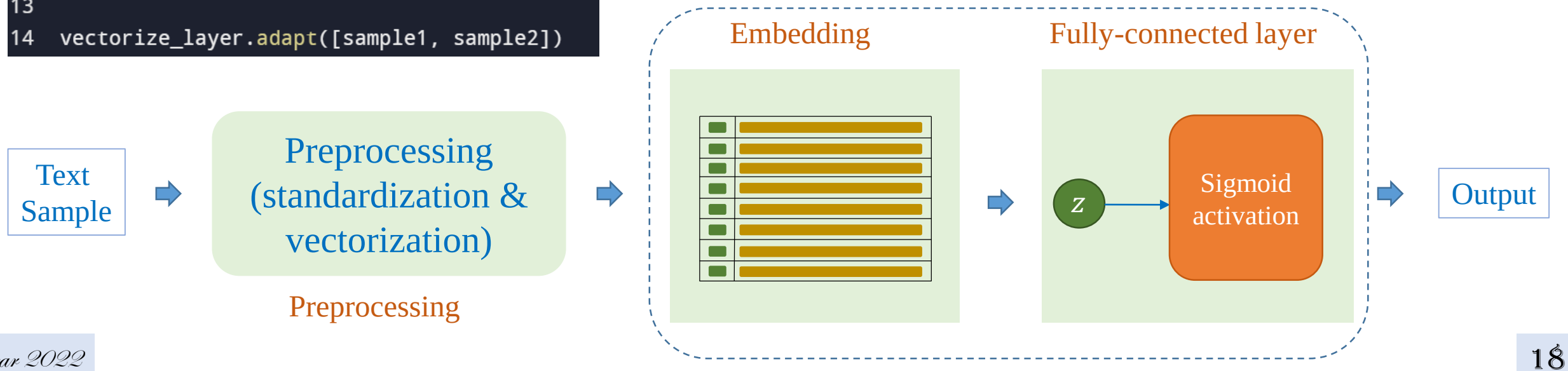
```
max_features = 1000
output_dims  = 128
batch_size   = 32

model = tf.keras.Sequential()
model.add(layers.Embedding(max_features, output_dims, input_length=10))

input_array = np.random.randint(max_features, size=(batch_size, 10))
output_array = model.predict(input_array)
```

(32, 10, 128)

Text Classification

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

max_features    = 8
embedding_dim   = 4
sequence_length = 5

vectorize_layer = TextVectorization(
    max_tokens=max_features,
    output_mode="int",
    output_sequence_length=sequence_length,
)

vectorize_layer.adapt([sample1, sample2])
```

```python
# model
inputs = tf.keras.Input(shape=(sequence_length,),
                        dtype="int64")
x = layers.Embedding(max_features, embedding_dim)(inputs)
x = layers.Flatten()(x)
x = layers.Dense(1, activation="sigmoid")(x)
model = tf.keras.Model(inputs, x)

model.compile(loss="binary_crossentropy",
              optimizer="adam",
              metrics=["accuracy"])
```



Text Classification

❖ IMDB dataset

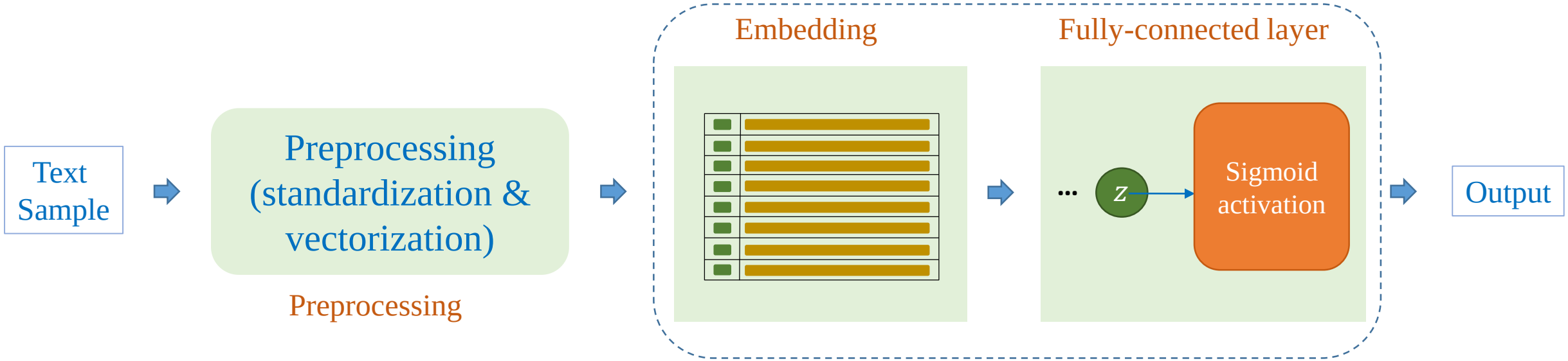
- 50,000 movie review for sentiment analysis
- Consist of: + 25,000 movie review for training
  + 25,000 movie review for testing
- Label: positive – negative

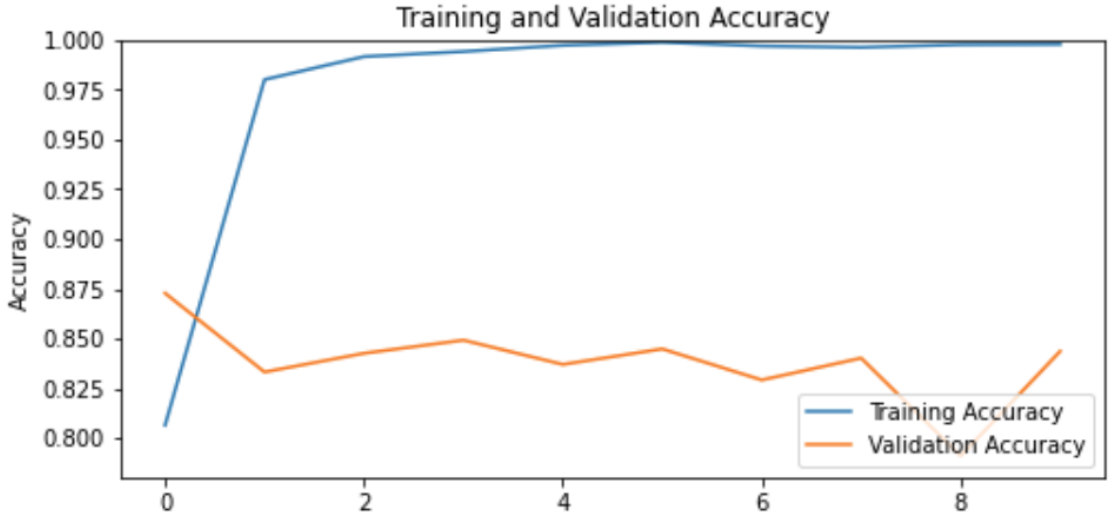
“A wonderful little production. The filming technique is very unassuming- very old-time-BBC fashion and gives a comforting, and sometimes discomforting, sense of realism to the entire piece…..”	positive
“This show was an amazing, fresh & innovative idea in the 70's when it first aired. The first 7 or 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 Deep Models

❖ Multilayer perceptron

2.TextClassification_MLP.ipynb

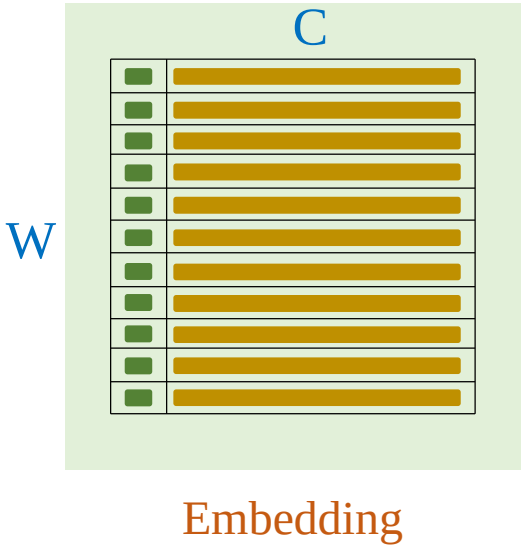


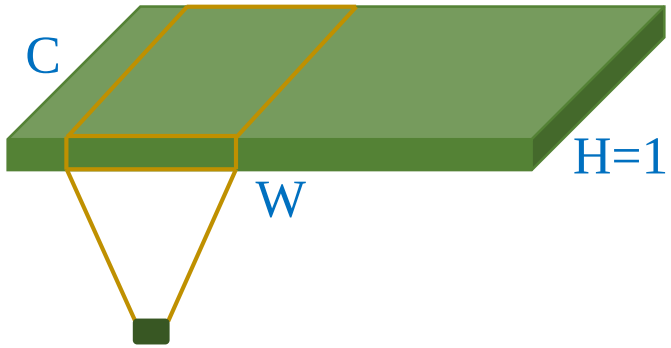



```
inputs = tf.keras.Input(shape=(500,), dtype="int64")
x = layers.Embedding(max_features, embedding_dim)(inputs)
x = layers.Flatten()(x)
x = layers.Dense(1024, activation="relu")(x)
x = layers.Dense(128, activation="relu")(x)
x = layers.Dense(1, activation="sigmoid")(x)
model = tf.keras.Model(inputs, x)
```

Text Deep Models

❖ Conv1D






```python
tf.keras.layers.Conv1D(
    filters,
    kernel_size,
    strides=1,
    padding="valid",
    activation=None,
    use_bias=True,
    kernel_initializer="glorot_uniform",
    bias_initializer="zeros"
)
```

AI VIETNAM
All-in-One Course

Text Deep Models

❖ Conv1D

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 500)]             0

 embedding (Embedding)       (None, 500, 128)          2560000

 conv1d (Conv1D)             (None, 165, 128)          114816

 conv1d_1 (Conv1D)           (None, 53, 128)           114816

 flatten (Flatten)           (None, 6784)              0

 dense (Dense)               (None, 128)               868480

 dense_1 (Dense)             (None, 1)                 129

=================================================================
Total params: 3,658,241
Trainable params: 3,658,241
Non-trainable params: 0
_________________________________________________________________
```

```python
# Model params
max_features = 20000
embedding_dim = 128
sequence_length = 500

inputs = tf.keras.Input(shape=(sequence_length,),
                        dtype="int64")
x = layers.Embedding(max_features,
                     embedding_dim)(inputs)

x = layers.Conv1D(128, 7,
                  padding="valid",
                  activation="relu",
                  strides=3)(x)
x = layers.Conv1D(128, 7,
                  padding="valid",
                  activation="relu",
                  strides=3)(x)

# Flatten and Dense
x = layers.Flatten()(x)
x = layers.Dense(128, activation="relu")(x)
x = layers.Dense(1, activation="sigmoid")(x)

model = tf.keras.Model(inputs, x)
```

AI VIETNAM
All-in-One Course

Global Pooling

Max pooling

v_1	v_2	v_3	v_4
v_5	v_6	v_7	v_8
v_9	v_{10}	v_{11}	v_{12}
v_{13}	v_{14}	v_{15}	v_{16}

Data

2x2 max pooling (| v_1 | v_2 | v_3 | v_4 | | v_5 | v_6 | v_7 | v_8 | | v_9 | v_{10} | v_{11} | v_{12} | | v_{13} | v_{14} | v_{15} | v_{16} |) = | m_1 | m_2 | | m_3 | m_4 |

$$m_1 = \max(v_1, v_2, v_5, v_6)$$
$$m_2 = \max(v_3, v_4, v_7, v_8)$$
$$m_3 = \max(v_9, v_{10}, v_{13}, v_{14})$$
$$m_4 = \max(v_{11}, v_{12}, v_{15}, v_{16})$$

Global max pooling

v_1	v_2	v_3	v_4
v_5	v_6	v_7	v_8
v_9	v_{10}	v_{11}	v_{12}
v_{13}	v_{14}	v_{15}	v_{16}

Data

global max pooling (| v_1 | v_2 | v_3 | v_4 | | v_5 | v_6 | v_7 | v_8 | | v_9 | v_{10} | v_{11} | v_{12} | | v_{13} | v_{14} | v_{15} | v_{16} |) = | m |

$$m = \max(v_1, v_2, \ldots, v_{16})$$

keras.layers.GlobalMaxPool2D()

AI VIETNAM
All-in-One Course

Text Deep Models

❖ Conv1D + GlobalMaxPooling1D

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 500)]             0

 embedding (Embedding)       (None, 500, 128)          2560000

 conv1d (Conv1D)             (None, 165, 128)          114816

 conv1d_1 (Conv1D)           (None, 53, 128)           114816

 global_max_pooling1d (Globa  (None, 128)              0
 lMaxPooling1D)

 dense (Dense)               (None, 128)               16512

 dense_1 (Dense)             (None, 1)                 129

=================================================================
Total params: 2,806,273
Trainable params: 2,806,273
Non-trainable params: 0
_________________________________________________________________
```

```python
# Model params
max_features = 20000
embedding_dim = 128
sequence_length = 500

inputs = tf.keras.Input(shape=(sequence_length,),
                        dtype="int64")
x = layers.Embedding(max_features,
                     embedding_dim)(inputs)

x = layers.Conv1D(128, 7,
                  padding="valid",
                  activation="relu",
                  strides=3)(x)
x = layers.Conv1D(128, 7,
                  padding="valid",
                  activation="relu",
                  strides=3)(x)

# GlobalMaxPooling1D
x = layers.GlobalMaxPooling1D()(x)

x = layers.Dense(128, activation="relu")(x)
x = layers.Dense(1, activation="sigmoid")(x)

model = tf.keras.Model(inputs, x)
```

Outline

- Dealing with Text
- Text Classification
- RNN and LSTM
- Bidirectional RNN/LSTM
- RNN/LSTM for Time-series data
- Introduction to NLP

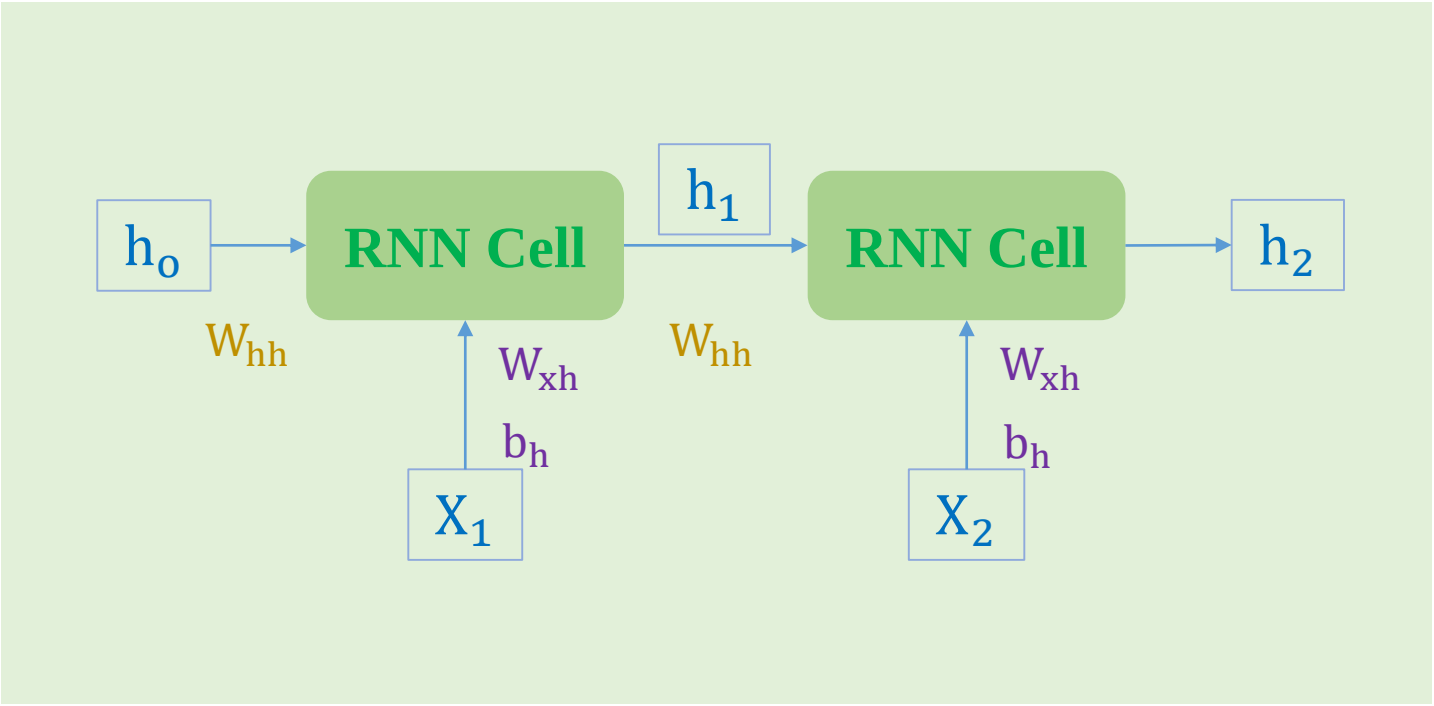
Text Deep Models

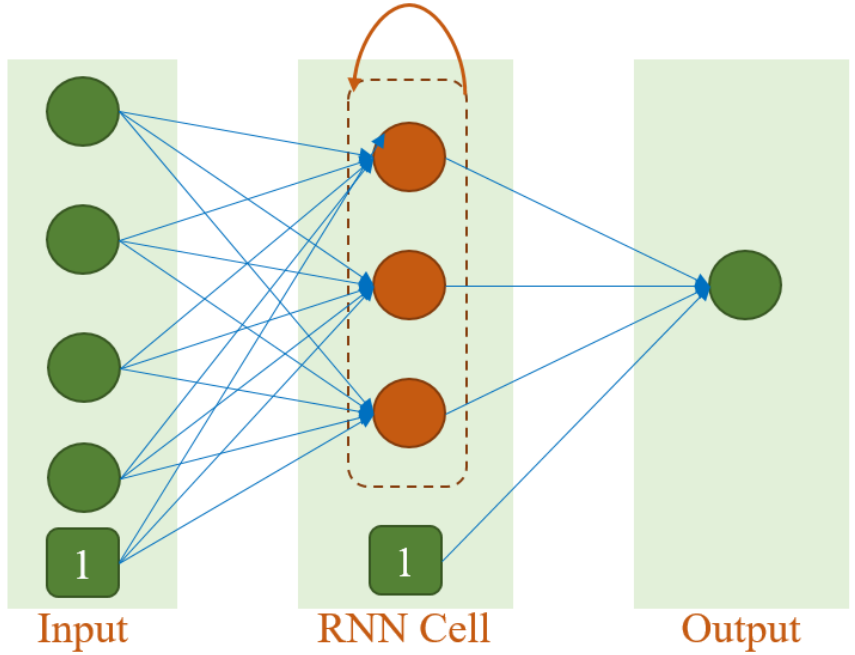
❖ Recurrent Neural Networks (RNNs)

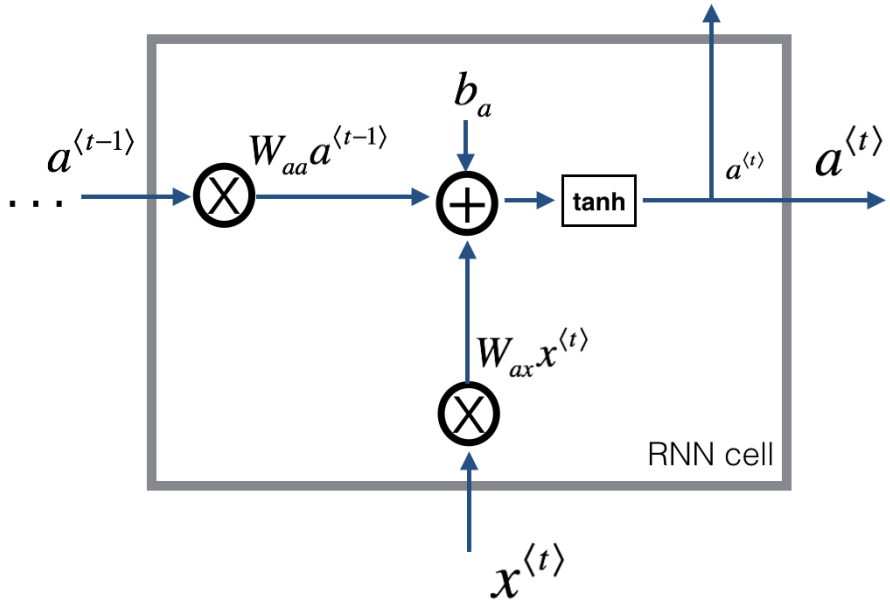
❖ Classification and time-steps=2

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

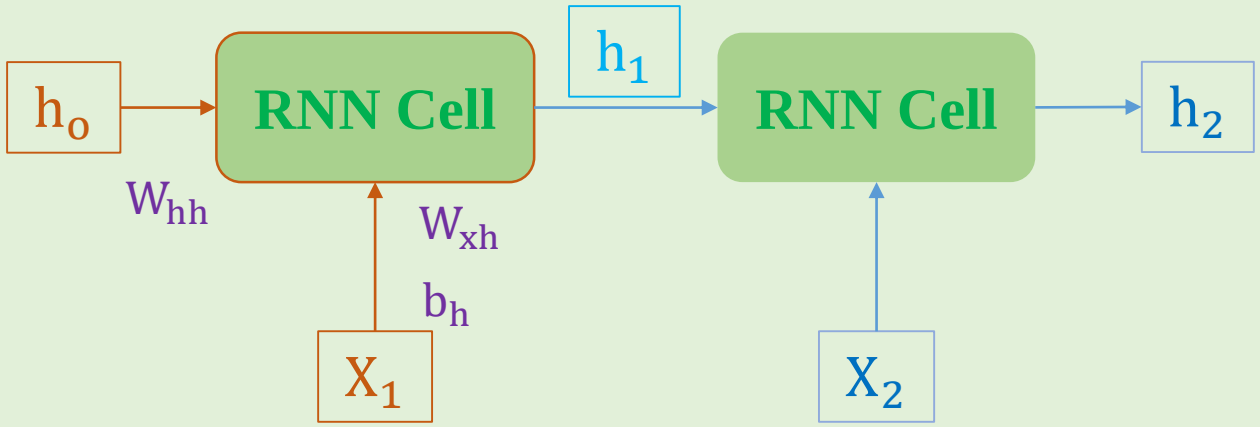
$$h_2 = \tanh(X_2 W_{xh} + h_1 W_{hh} + b_h)$$







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



transpose

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

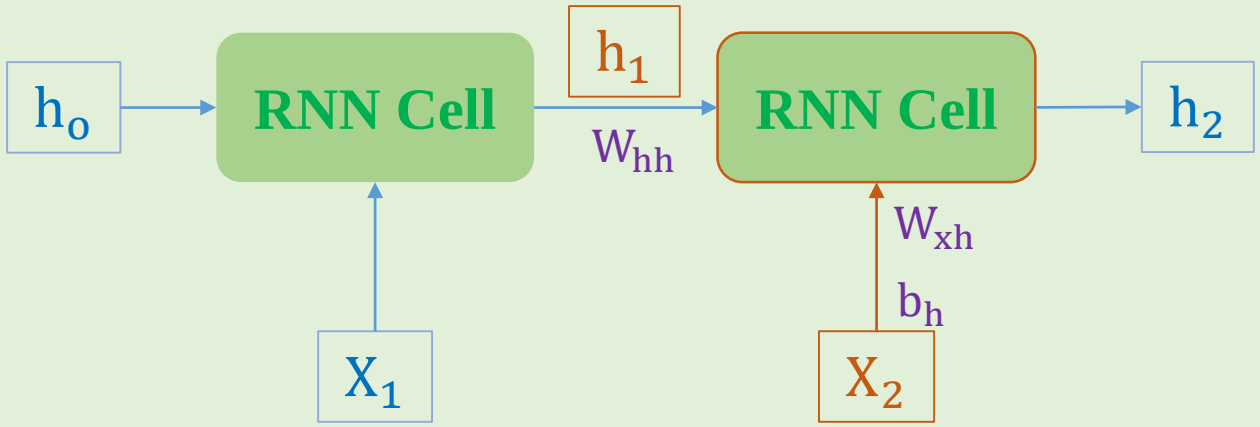
$$\mathbf{W_{xh}} = \begin{bmatrix} -0.2 & -0.6 & 0.1 \\ -0.4 & -0.8 & -0.1 \\ 0.3 & 0.5 & 0.6 \\ -0.4 & -0.3 & 0.1 \end{bmatrix}$$

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

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

$$= \tanh\left([1. \quad 3. \quad 2. \quad 1.] \begin{bmatrix} -0.2 & -0.6 & 0.1 \\ -0.4 & -0.8 & -0.1 \\ 0.3 & 0.5 & 0.6 \\ -0.4 & -0.3 & 0.1 \end{bmatrix} + [0. \quad 0. \quad 0.] \begin{bmatrix} 0.9 & -0.1 & 0.1 \\ -0.1 & -0.9 & -0.3 \\ 0.1 & 0.3 & -0.9 \end{bmatrix} + \begin{bmatrix} 0.0 \\ 0.0 \\ 0.0 \end{bmatrix} \right)$$

$$= \tanh([-1.2 \quad -2.3 \quad 1.1]) = [-0.833 \quad -0.98 \quad 0.8]$$



transpose

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

$$\mathbf{W_{xh}} = \begin{bmatrix} -0.2 & -0.6 & 0.1 \\ -0.4 & -0.8 & -0.1 \\ 0.3 & 0.5 & 0.6 \\ -0.4 & -0.3 & 0.1 \end{bmatrix} \qquad \boldsymbol{h}_0 = [0. \quad 0. \quad 0.]$$

$$\boldsymbol{h}_1 = [-0.833 \quad -0.98 \quad 0.8]$$

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

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

$$= \tanh\left([0. \quad 4. \quad 1. \quad 2.] \begin{bmatrix} -0.2 & -0.6 & 0.1 \\ -0.4 & -0.8 & -0.1 \\ 0.3 & 0.5 & 0.6 \\ -0.4 & -0.3 & 0.1 \end{bmatrix} + [-0.833 \quad -0.98 \quad 0.8] \begin{bmatrix} 0.9 & -0.1 & 0.1 \\ -0.1 & -0.9 & -0.3 \\ 0.1 & 0.3 & -0.9 \end{bmatrix} + \begin{bmatrix} 0.0 \\ 0.0 \\ 0.0 \end{bmatrix} \right)$$

$$= \tanh([-2.67 \quad -2.09 \quad -0.109]) = [-0.99 \quad -0.97 \quad -0.109]$$

Text Deep Models

❖ Implementation

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

```python
import tensorflow as tf
import tensorflow.keras as keras

# inputs: [batch, timesteps, feature]
inputs = tf.convert_to_tensor([[[1., 3., 2., 1.],
                                [0., 4., 1., 2.]]])

# create SimpleRNN
simple_rnn = keras.layers.SimpleRNN(3)

# feed data
output = simple_rnn(inputs)
```

$$\boldsymbol{h}_2 = [-0.99 \quad -0.97 \quad -0.109]$$

```python
import tensorflow as tf
import tensorflow.keras as keras

# inputs: [batch, timesteps, feature]
inputs = tf.convert_to_tensor([[[1., 3., 2., 1.],
                                [0., 4., 1., 2.]]])

# create SimpleRNN
simple_rnn = tf.keras.layers.SimpleRNN(3,
                                       return_sequences=True,
                                       return_state=True)

# feed data
sequence_output, final_state = simple_rnn(inputs)
```

$$\boldsymbol{h}_1 = [-0.833 \quad -0.98 \quad 0.8]$$

$$\boldsymbol{h}_2 = [-0.99 \quad -0.97 \quad -0.109]$$

Text Deep Models

❖ Recurrent Neural Networks (RNNs)

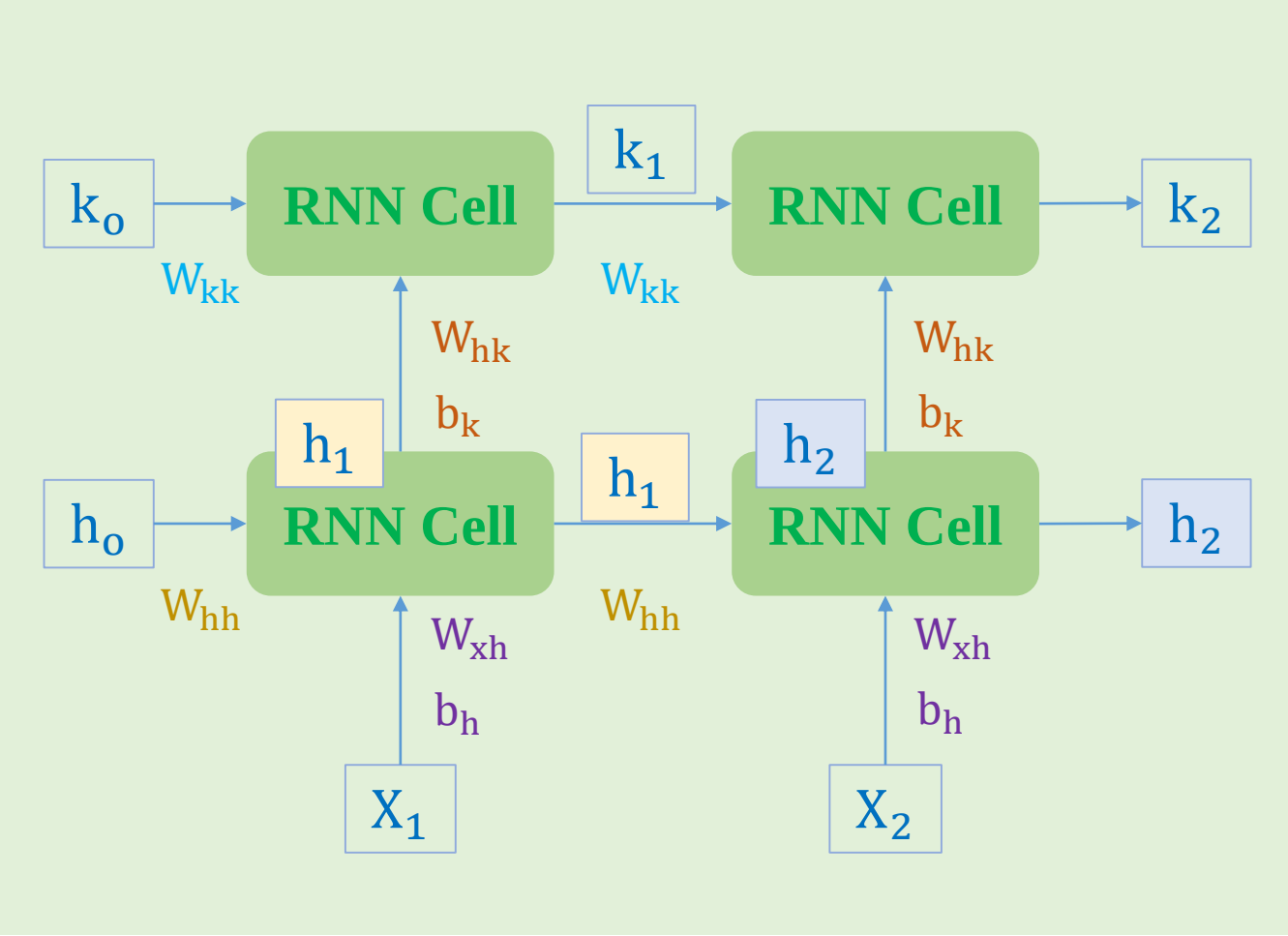
❖ Two layers

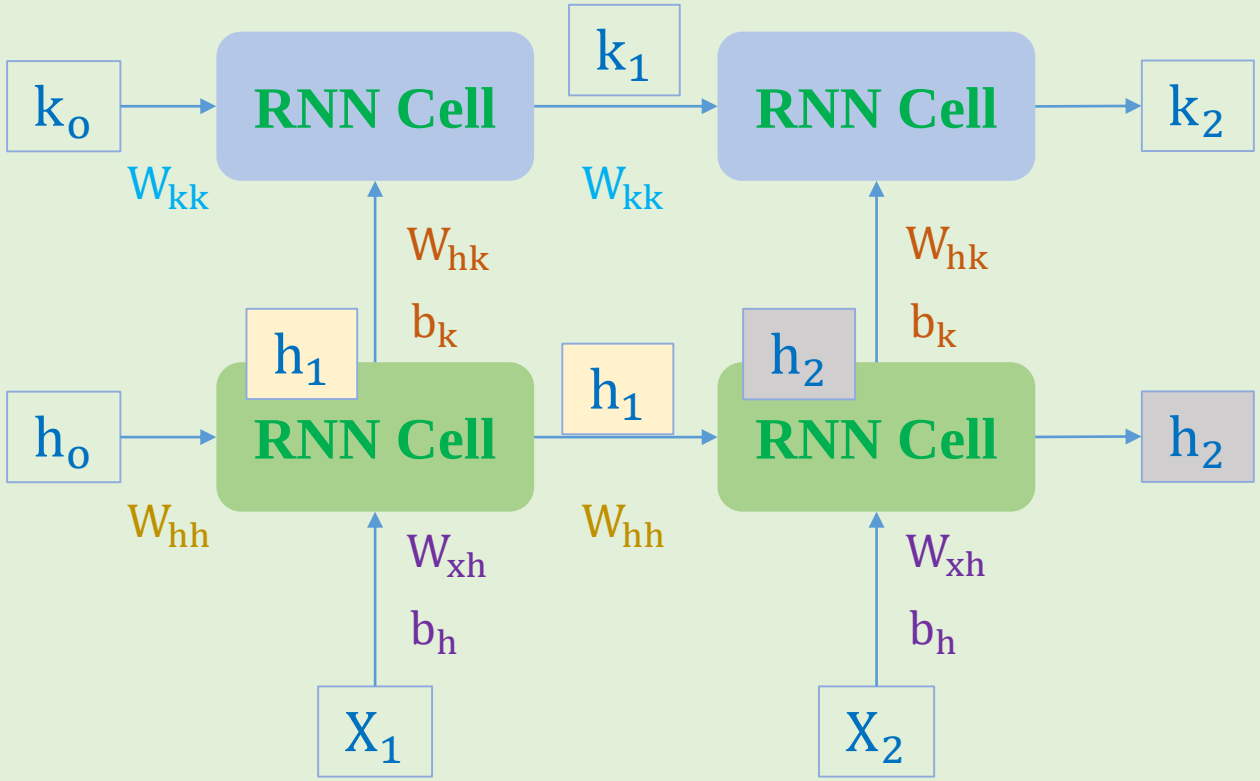
$$k_1 = \tanh(h_1 W_{hk} + h_0 W_{kk} + b_k)$$

$$k_2 = \tanh(h_2 W_{hk} + h_1 W_{kk} + b_k)$$

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

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





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

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

$$\mathbf{b_h} = \begin{bmatrix} 0.0 \\ 0.0 \\ 0.0 \end{bmatrix} \qquad \mathbf{b_k} = \begin{bmatrix} 0.0 \\ 0.0 \\ 0.0 \end{bmatrix}$$

$$k_1 = \tanh(h_1 W_{hk} + h_0 W_{kk} + b_k)$$

$$= [0.49 \quad 0.407]$$

$$k_2 = \tanh(h_2 W_{hk} + h_1 W_{kk} + b_k)$$

$$= [0.84 \quad 0.42]$$

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

$$= [-0.833 \quad -0.98 \quad 0.8]$$

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

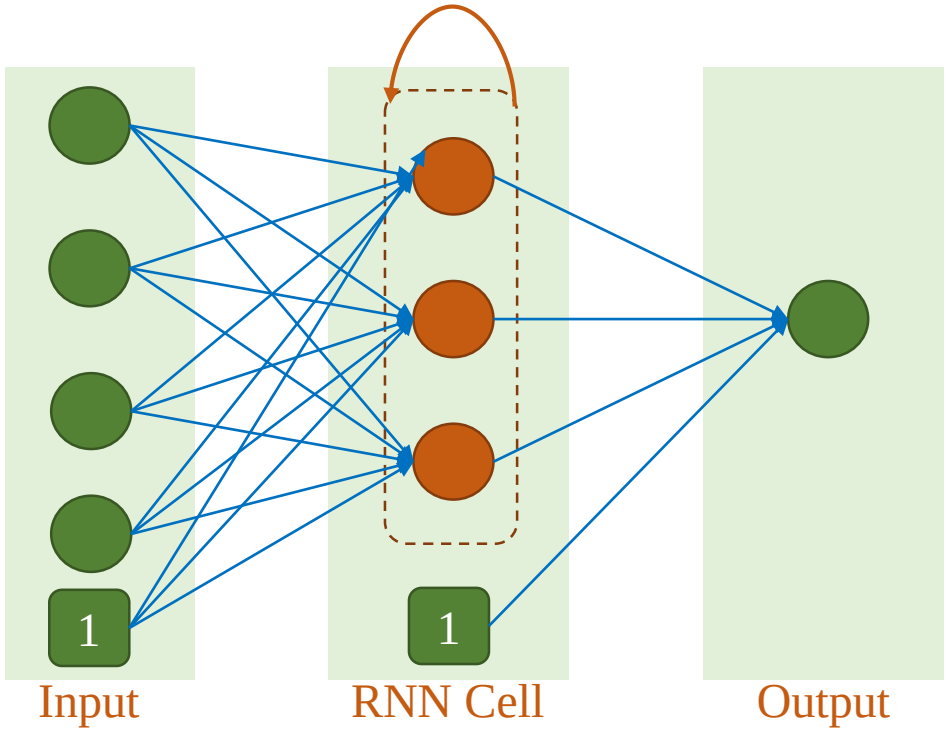
$$= [-0.99 \quad -0.97 \quad -0.109]$$

Text Deep Models

❖ Recurrent Neural Networks (RNN)

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

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




```python
inputs = keras.Input(shape=(5,), dtype="int32")
x = layers.Embedding(100, 4)(inputs)

# Add SimpleRNN
x = layers.SimpleRNN(3)(x)

# Add a classifier
outputs = layers.Dense(1, activation="sigmoid")(x)

# model
model = keras.Model(inputs, outputs)
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 5)]               0

 embedding (Embedding)       (None, 5, 4)              400

 simple_rnn (SimpleRNN)      (None, 3)                 24

 dense (Dense)               (None, 1)                 4

=================================================================
Total params: 428
Trainable params: 428
Non-trainable params: 0
_________________________________________________________________
```

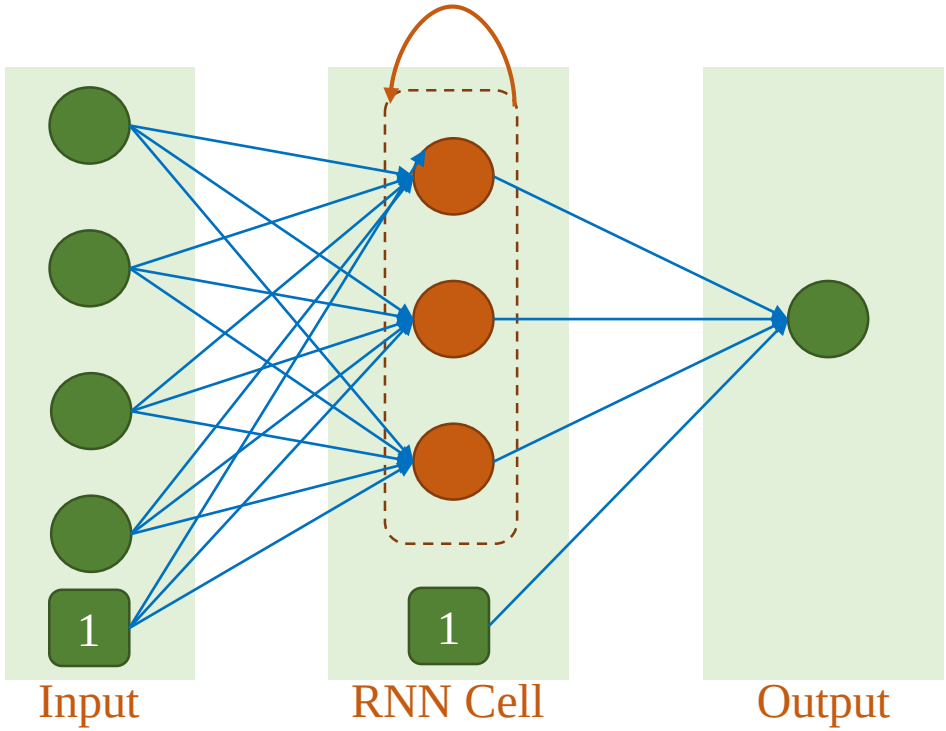
Text Deep Models

```
1  inputs = keras.Input(shape=(5,), dtype="int32")
2  x = layers.Embedding(100, 4)(inputs)
3
4  # Add SimpleRNN
5  x = layers.SimpleRNN(3)(x)
6
7  # Add a classifier
8  outputs = layers.Dense(1, activation="sigmoid")(x)
9
10 # model
11 model = keras.Model(inputs, outputs)
```

❖ Recurrent Neural Networks (RNN)

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

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



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

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

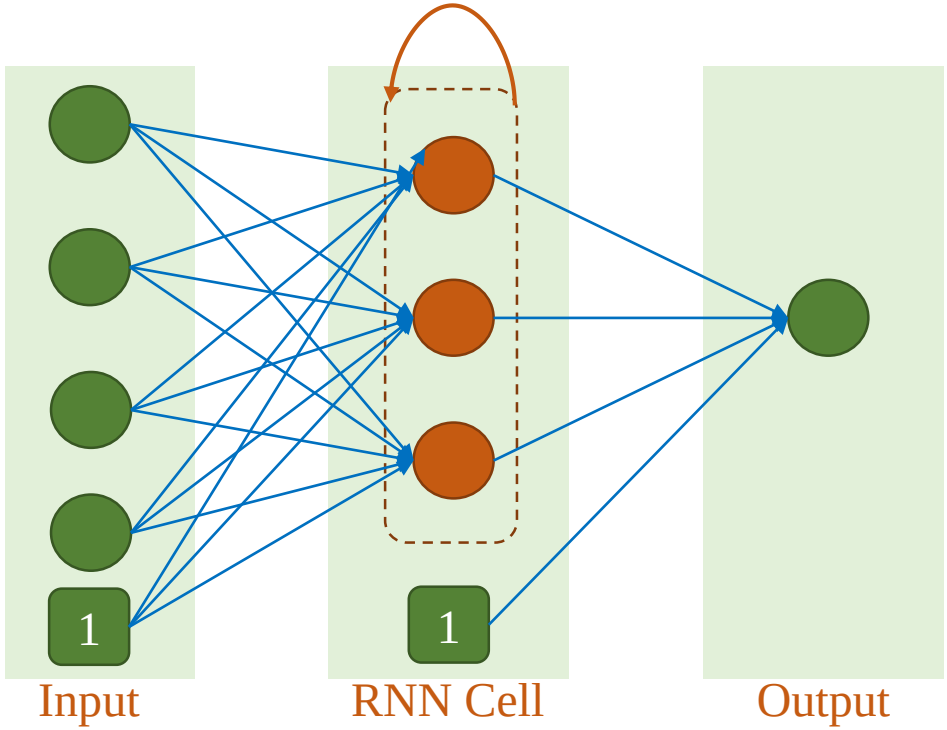
Text Deep Models

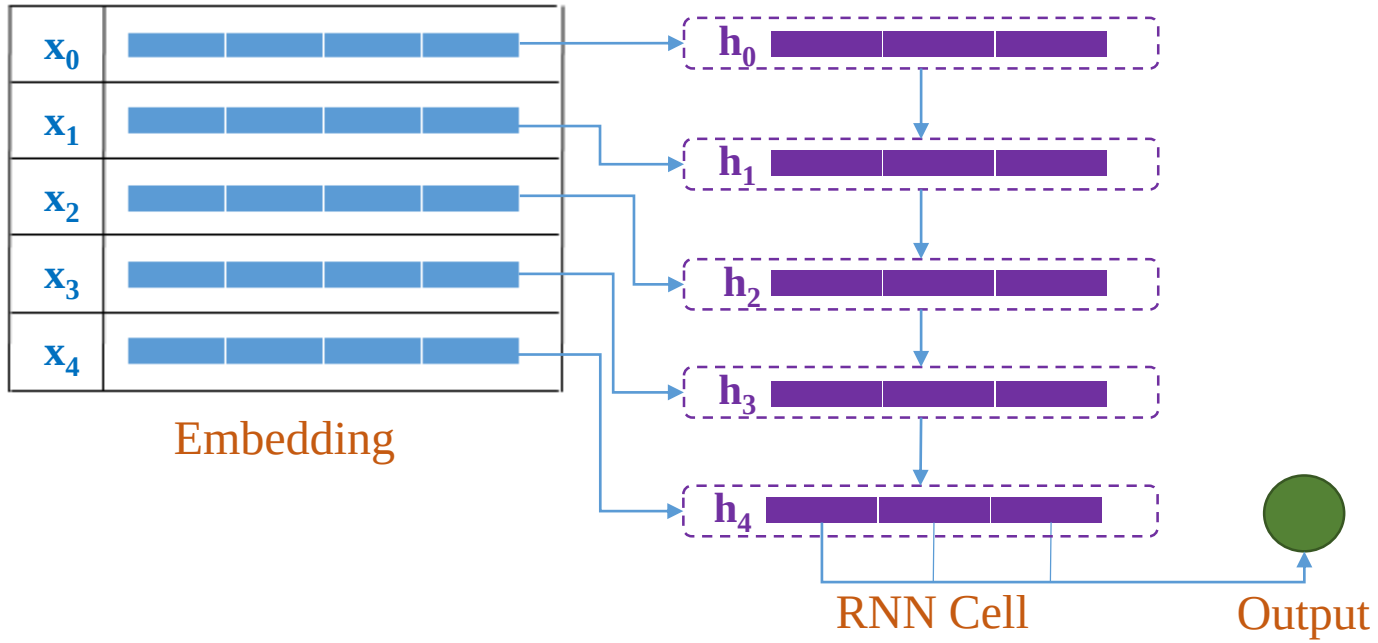
```
inputs = keras.Input(shape=(5,), dtype="int32")
x = layers.Embedding(100, 4)(inputs)

# Add SimpleRNN
x = layers.SimpleRNN(3)(x)

# Add a classifier
outputs = layers.Dense(1, activation="sigmoid")(x)

# model
model = keras.Model(inputs, outputs)
```

❖ Recurrent Neural Networks (RNN)

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

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





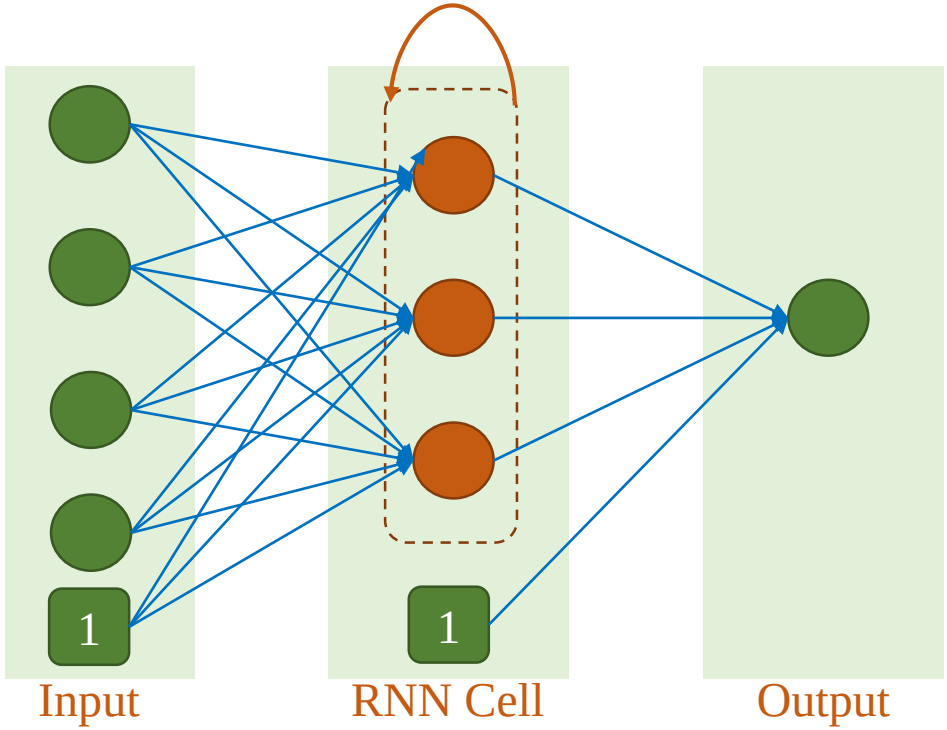
Text Deep Models

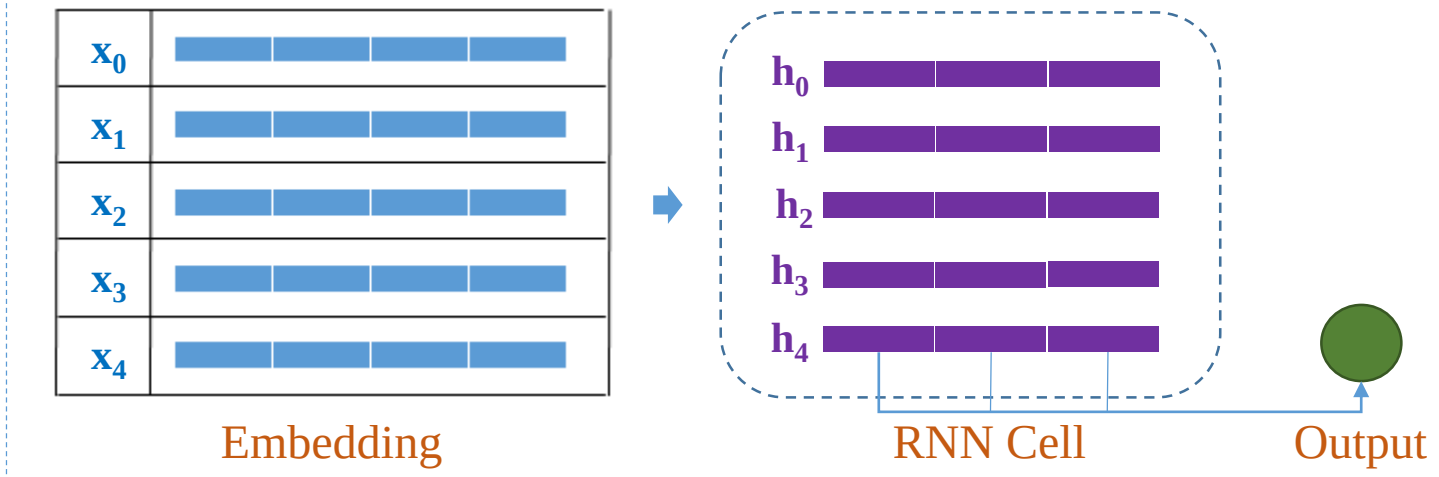
```
1  inputs = keras.Input(shape=(5,), dtype="int32")
2  x = layers.Embedding(100, 4)(inputs)
3
4  # Add SimpleRNN
5  x = layers.SimpleRNN(3)(x)
6
7  # Add a classifier
8  outputs = layers.Dense(1, activation="sigmoid")(x)
9
10 # model
11 model = keras.Model(inputs, outputs)
```

❖ Recurrent Neural Networks (RNN)

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

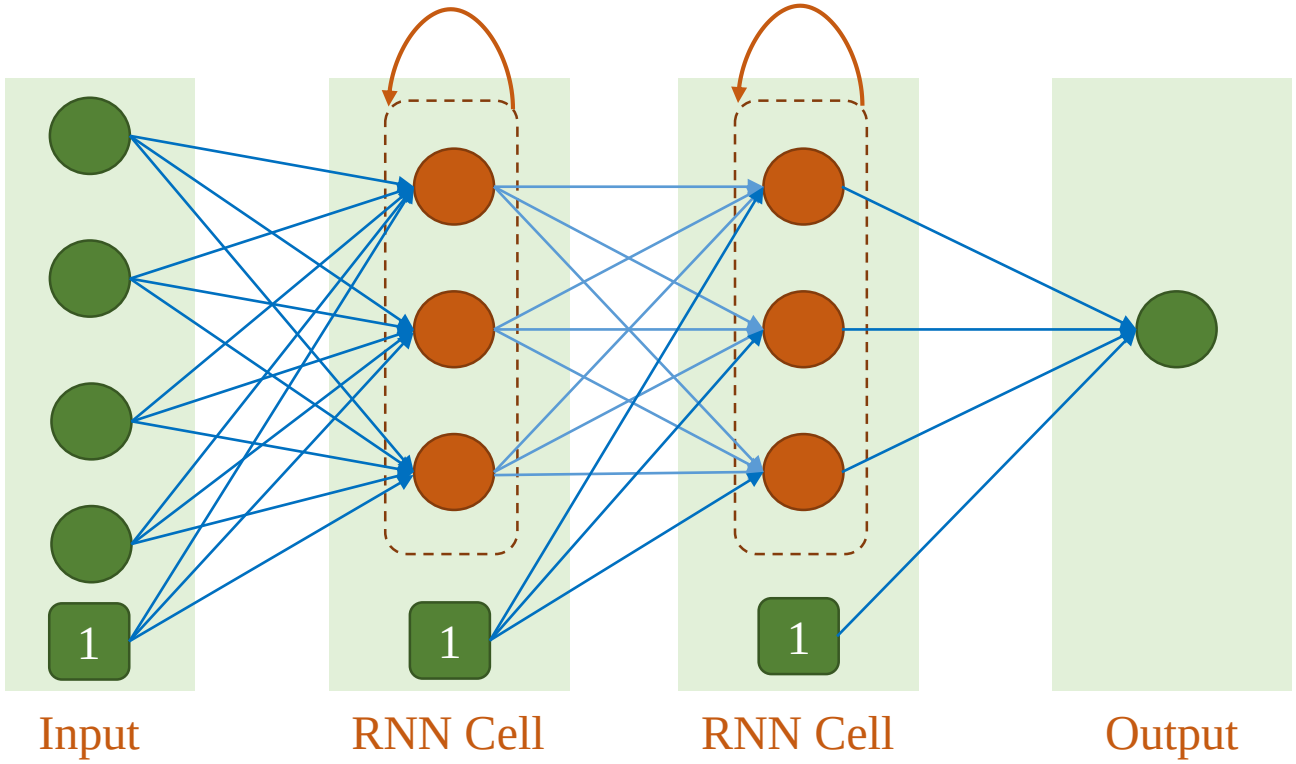
$$W_{xh}=\begin{bmatrix}-0.436 & 0.373 & -0.032 & 0.403\\ -0.452 & 0.296 & -0.609 & -0.643\\ -0.761 & 0.266 & -0.526 & -0.703\end{bmatrix}$$





Text Deep Models

❖ **Recurrent Neural Networks (RNN)**



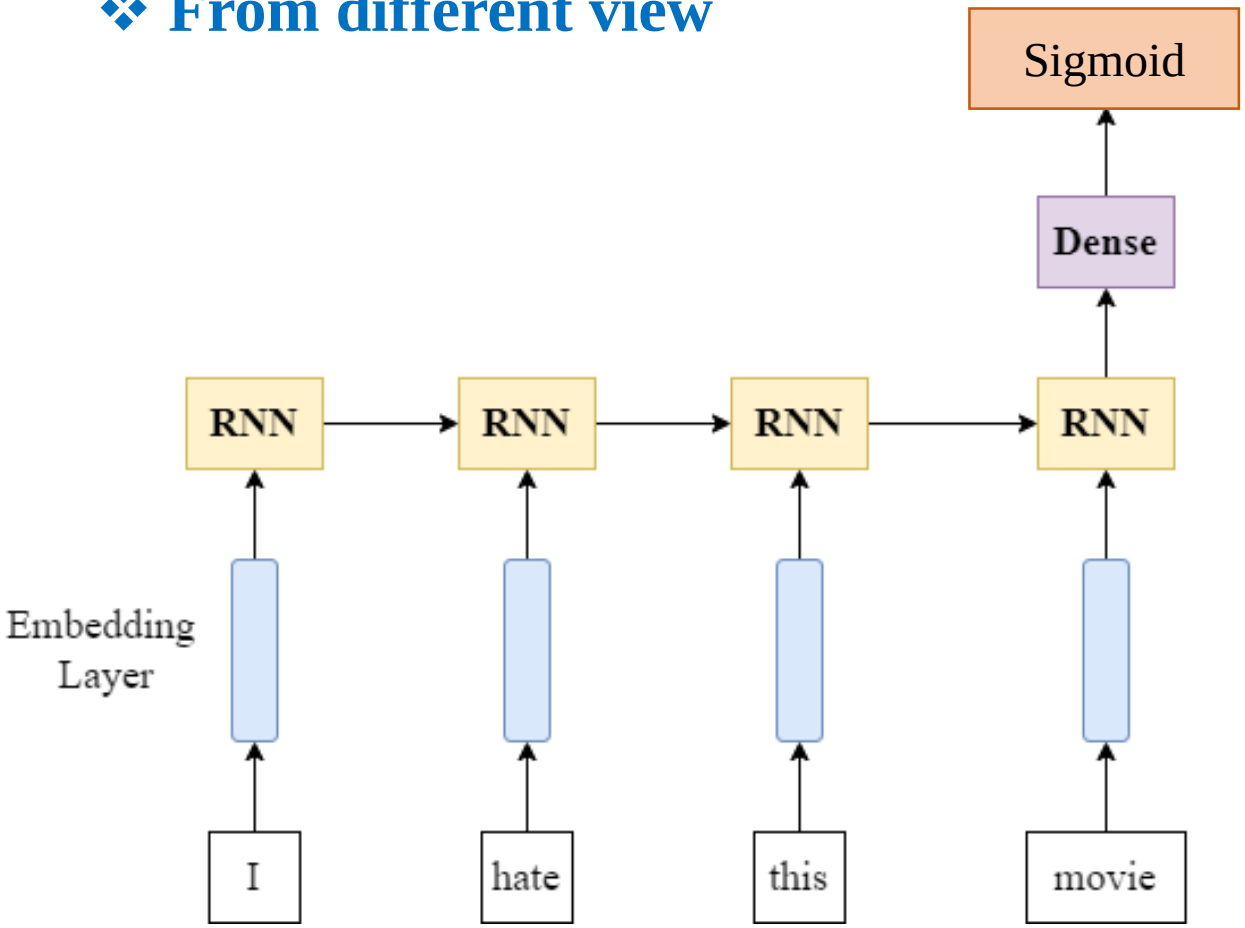

```python
inputs = keras.Input(shape=(5,), dtype="int32")
x = layers.Embedding(100, 4)(inputs)

# Add SimpleRNN
x = layers.SimpleRNN(3, return_sequences=True)(x)
x = layers.SimpleRNN(3)(x)

# Add a classifier
x = layers.Dense(1, activation="sigmoid")(x)

# model
model = keras.Model(inputs, x)
```

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 5)]	0
embedding (Embedding)	(None, 5, 4)	400
simple_rnn (SimpleRNN)	(None, 5, 3)	24
simple_rnn_1 (SimpleRNN)	(None, 3)	21
dense (Dense)	(None, 1)	4

```
Total params: 449
Trainable params: 449
Non-trainable params: 0
```

Applications of Text Analysis

- Text classification
- From different view




```python
inputs = keras.Input(shape=(5,), dtype="int32")
x = layers.Embedding(100, 4)(inputs)

# Add SimpleRNN
x = layers.SimpleRNN(3)(x)

# Add a classifier
outputs = layers.Dense(1, activation="sigmoid")(x)

# model
model = keras.Model(inputs, outputs)
```

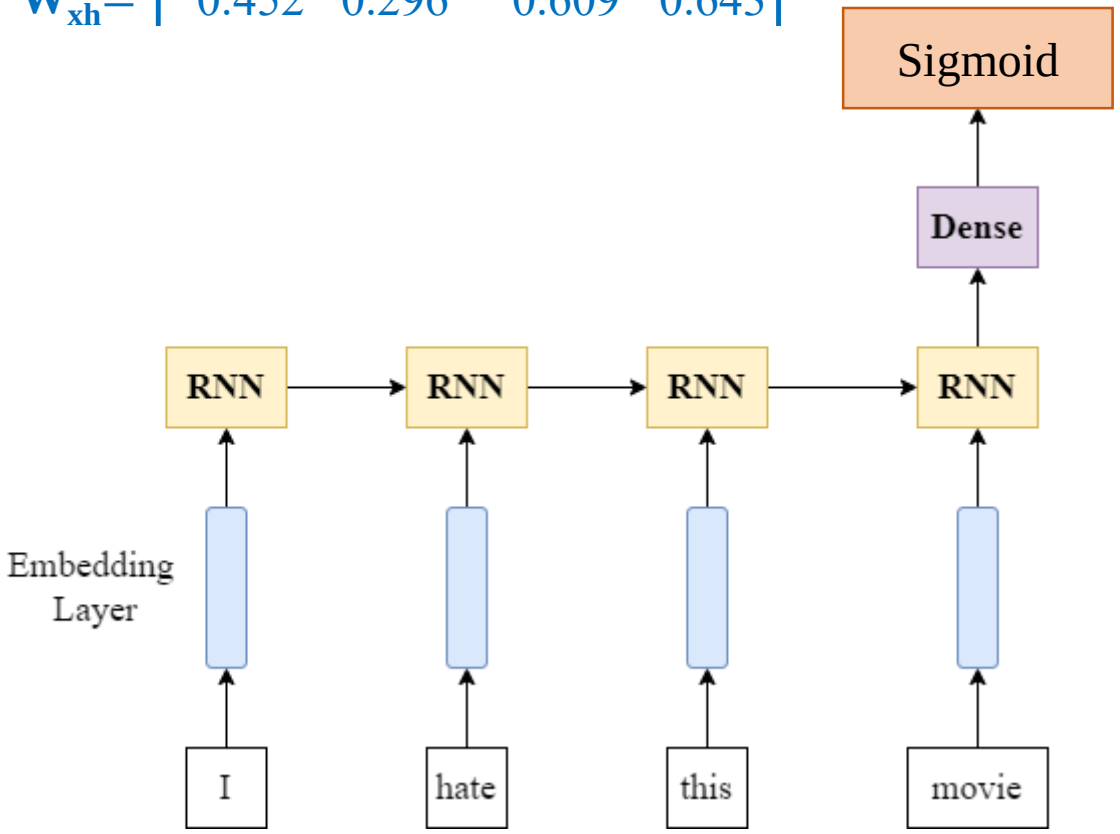
```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 5)]               0

 embedding (Embedding)       (None, 5, 4)              400

 simple_rnn (SimpleRNN)      (None, 3)                 24

 dense (Dense)               (None, 1)                 4

=================================================================
```

Text Deep Models

❖ Recurrent Neural Networks (RNN)

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

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




```python
inputs = keras.Input(shape=(5,), dtype="int32")
x = layers.Embedding(100, 4)(inputs)

# Add SimpleRNN
x = layers.SimpleRNN(3)(x)

# Add a classifier
outputs = layers.Dense(1, activation="sigmoid")(x)

# model
model = keras.Model(inputs, outputs)
```

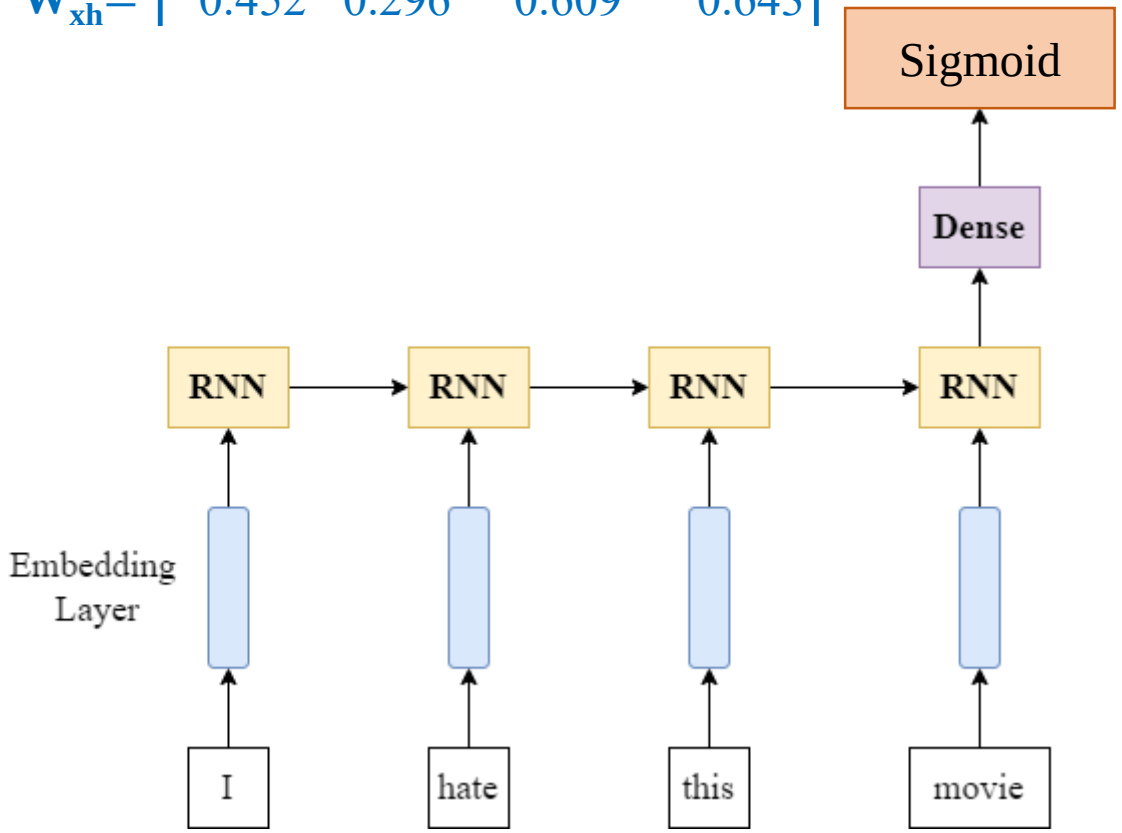
```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 5)]               0

 embedding (Embedding)       (None, 5, 4)              400

 simple_rnn (SimpleRNN)      (None, 3)                 24

 dense (Dense)               (None, 1)                 4

=================================================================
Total params: 428
Trainable params: 428
Non-trainable params: 0
_________________________________________________________________
```

Text Deep Models

❖ Recurrent Neural Networks (RNN)

$$W_{hh} = \begin{bmatrix} 0.226 & -0.068 & -0.971 \\ -0.973 & -0.014 & -0.226 \\ -0.001 & -0.997 & 0.070 \end{bmatrix} \quad b_h = \begin{bmatrix} 0.0 \\ 0.0 \\ 0.0 \end{bmatrix}$$

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




```python
inputs = keras.Input(shape=(5,), dtype="int32")
x = layers.Embedding(100, 4)(inputs)

# Add SimpleRNN
x = layers.SimpleRNN(3)(x)

# Add a classifier
outputs = layers.Dense(1, activation="sigmoid")(x)

# model
model = keras.Model(inputs, outputs)
```

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t + b_h)$$

$$o_t = \text{sigmoid}\,(W_{hy}h_t + b_y)$$

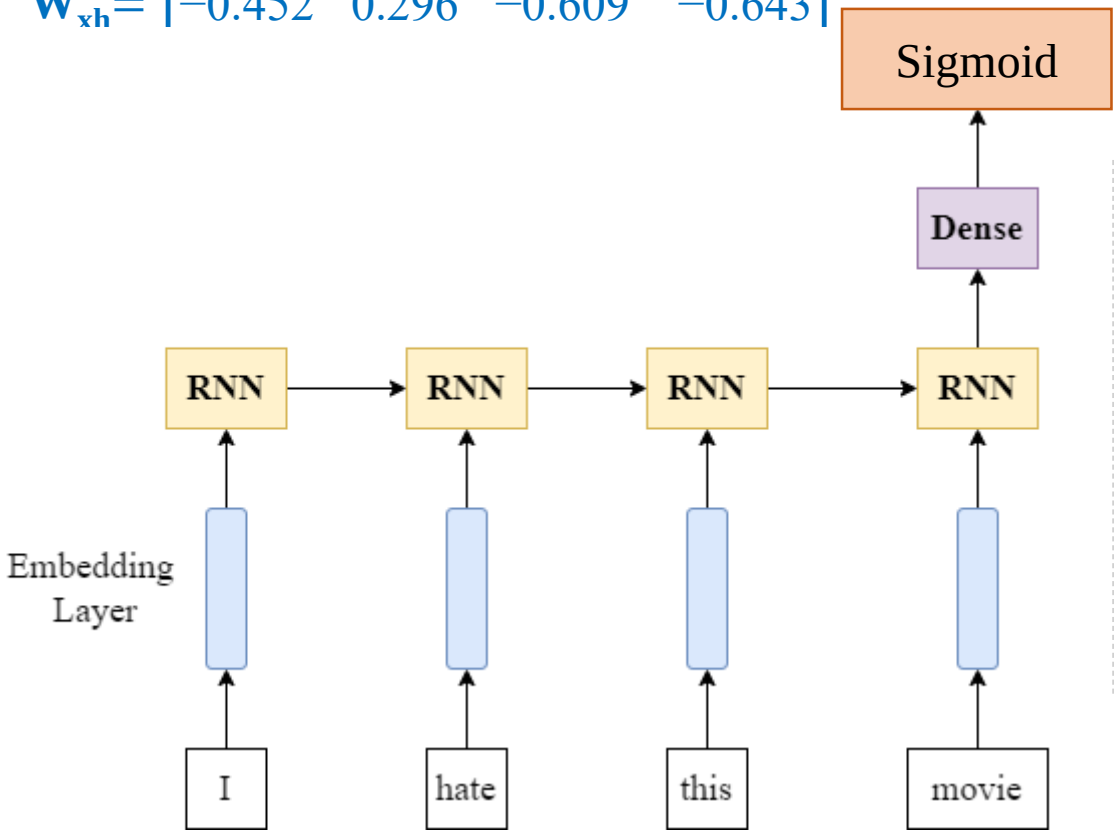
Text Deep Models

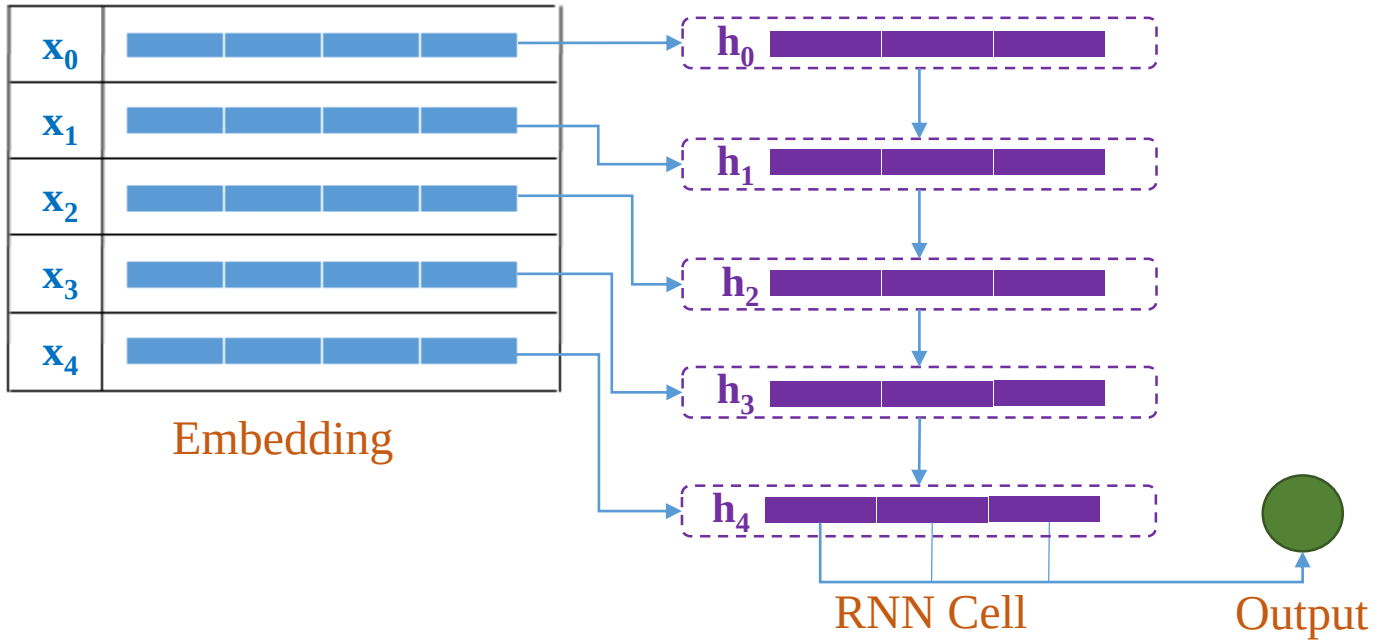
❖ Recurrent Neural Networks (RNN)

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

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

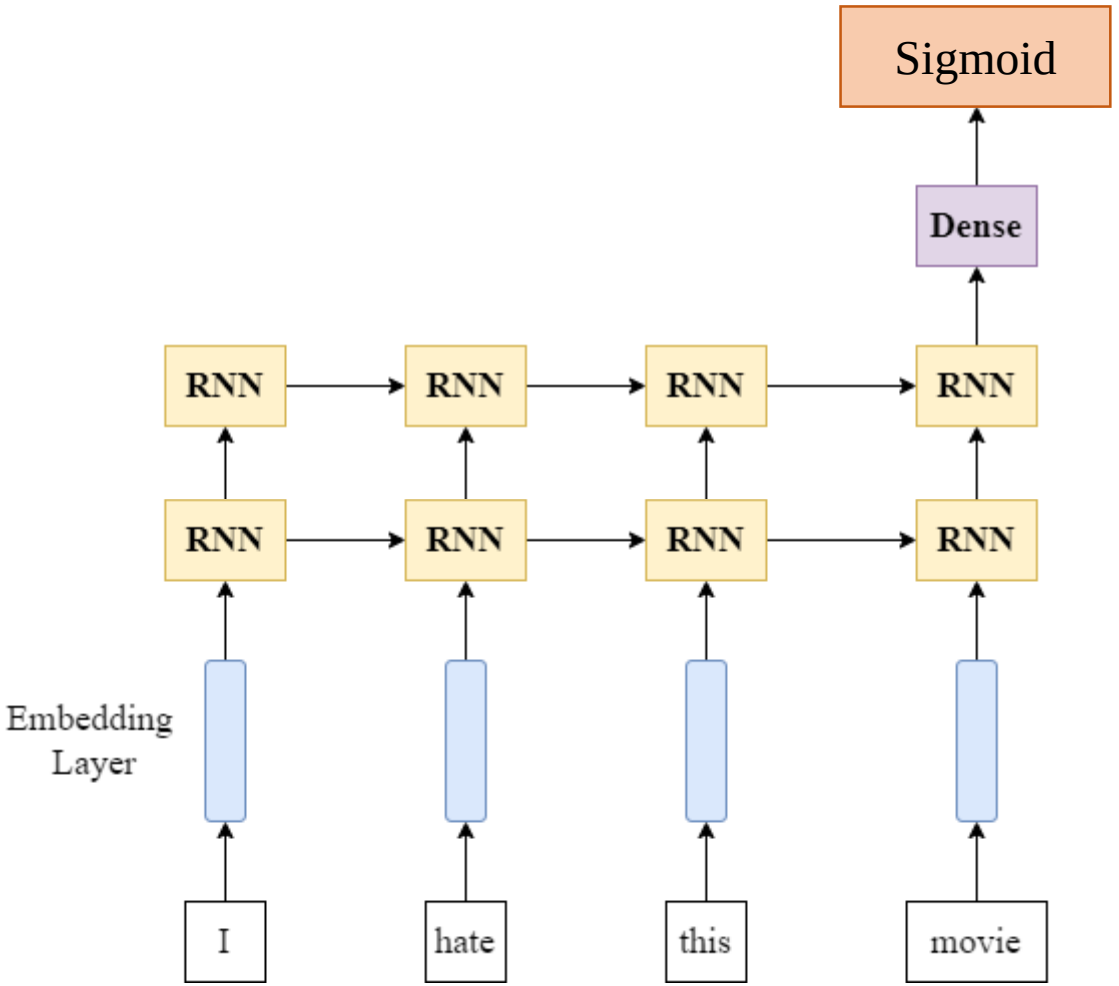
```python
inputs = keras.Input(shape=(5,), dtype="int32")
x = layers.Embedding(100, 4)(inputs)

# Add SimpleRNN
x = layers.SimpleRNN(3)(x)

# Add a classifier
outputs = layers.Dense(1, activation="sigmoid")(x)

# model
model = keras.Model(inputs, outputs)
```





Text Deep Models

❖ Recurrent Neural Networks (RNN)

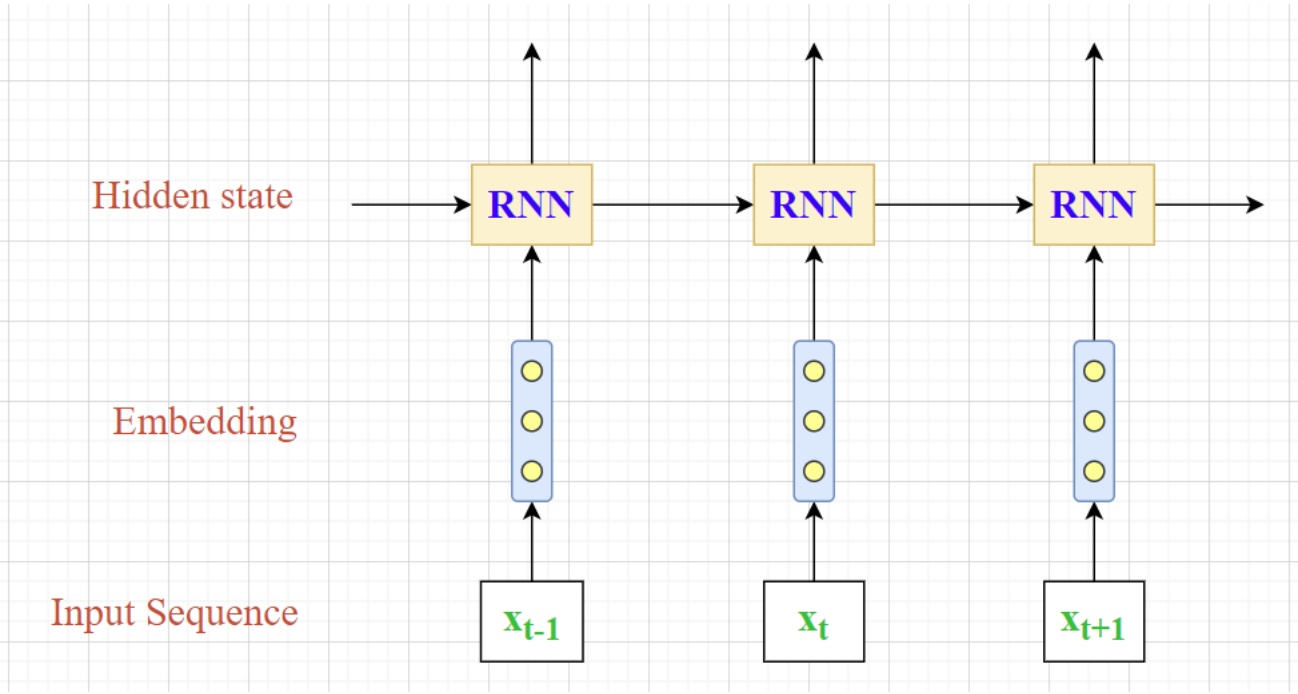


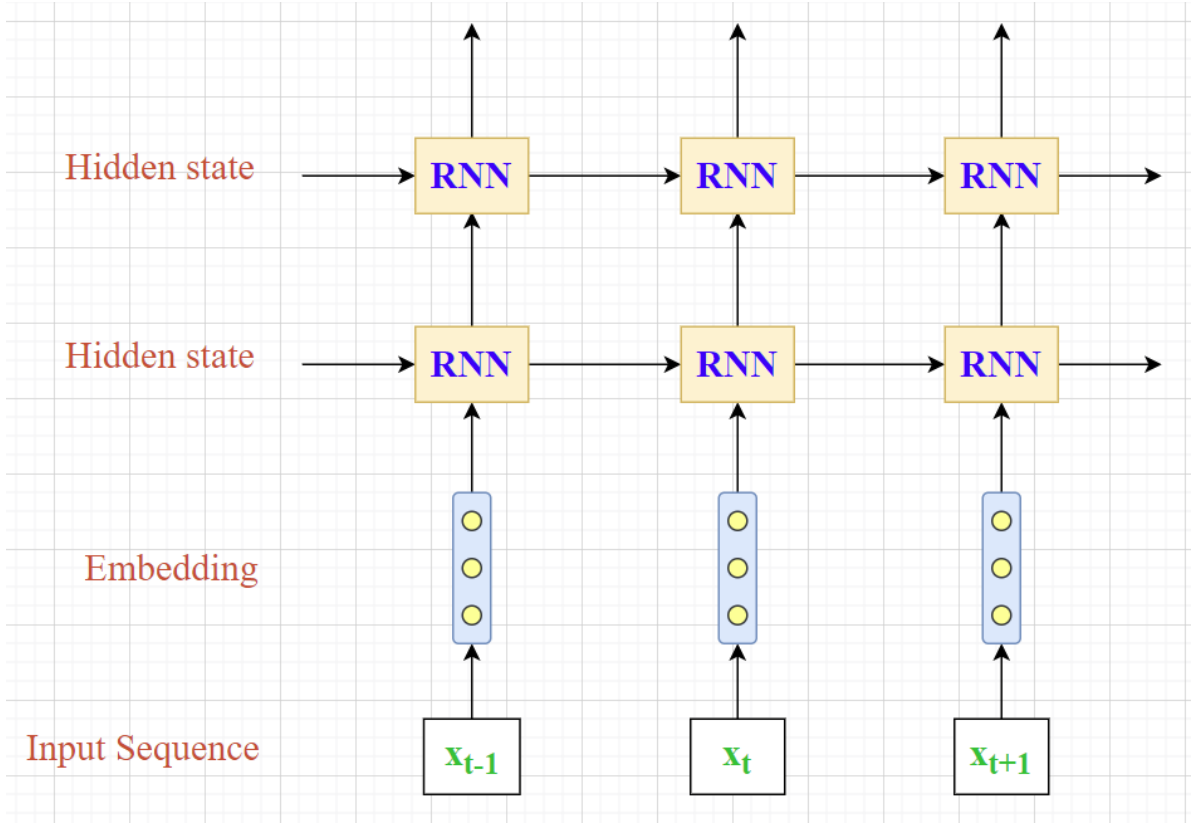

```python
inputs = keras.Input(shape=(5,), dtype="int32")
x = layers.Embedding(100, 4)(inputs)

# Add SimpleRNN
x = layers.SimpleRNN(3, return_sequences=True)(x)
x = layers.SimpleRNN(3)(x)

# Add a classifier
x = layers.Dense(1, activation="sigmoid")(x)

# model
model = keras.Model(inputs, x)
```

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 5)]               0

 embedding (Embedding)       (None, 5, 4)              400

 simple_rnn (SimpleRNN)      (None, 5, 3)              24

 simple_rnn_1 (SimpleRNN)    (None, 3)                 21

 dense (Dense)               (None, 1)                 4

=================================================================
Total params: 449
Trainable params: 449
Non-trainable params: 0
```

Text Deep Models

❖ Recurrent Neural Networks (RNN)

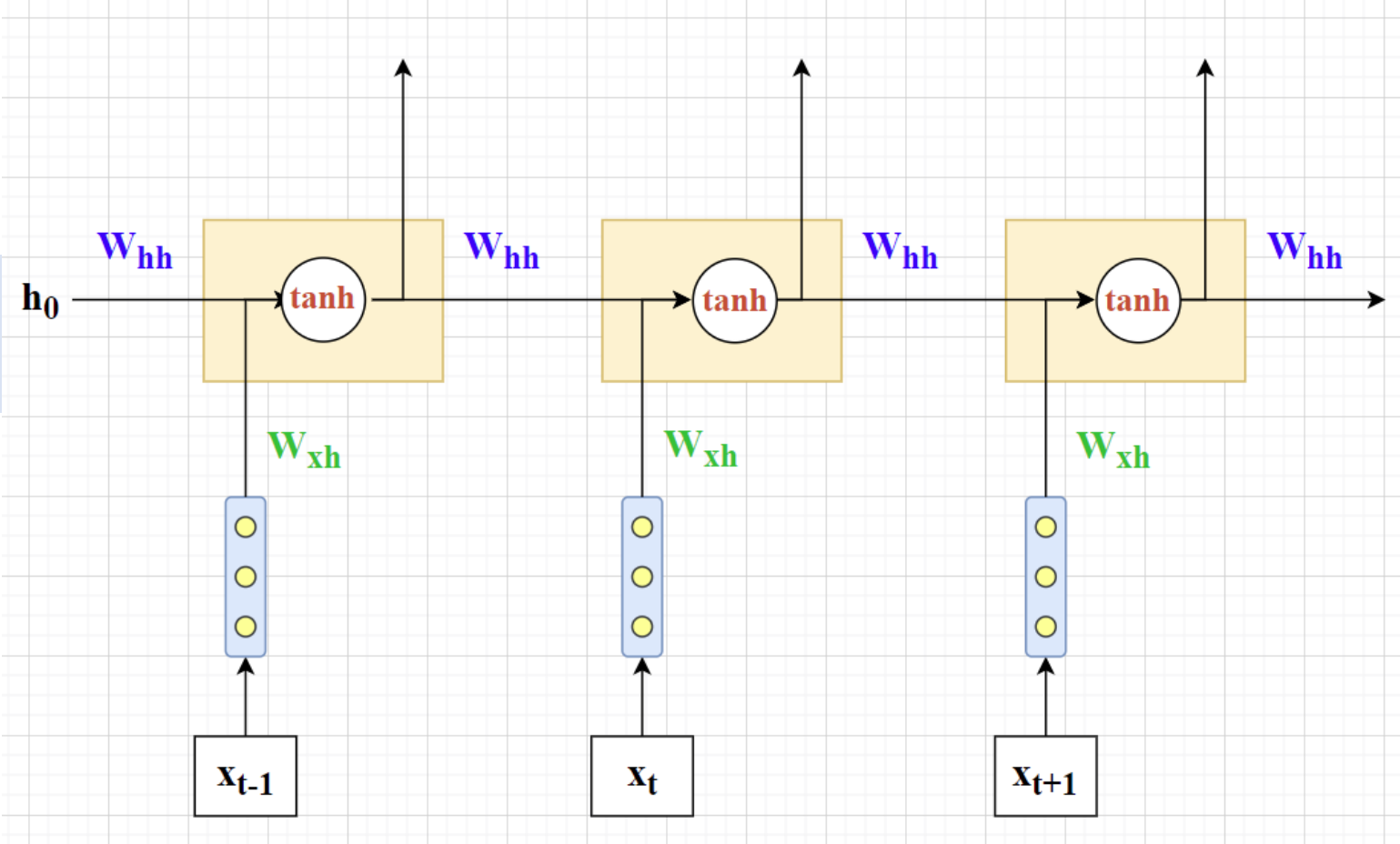




Text Deep Models

❖ Recurrent Neural Networks (RNN)

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t + b_h)$$



Text Classification

❖ IMDB dataset

- 50,000 movie review for sentiment analysis (data)
- Consist of: + 25,000 movie review for training
  + 25,000 movie review for testing
- Label: positive – negative = 1 – 1

“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

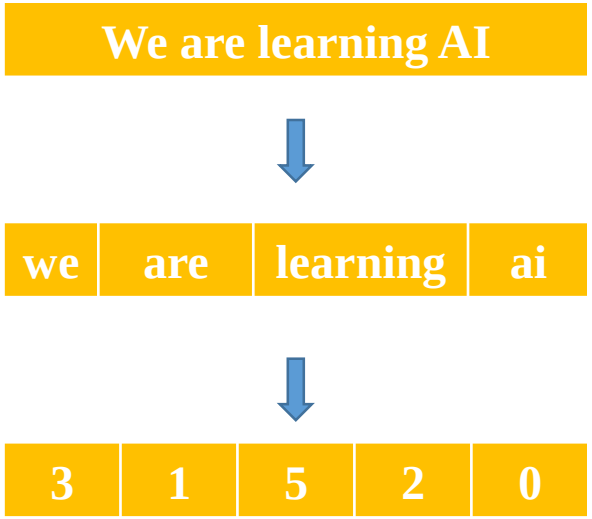
❖ IMDB dataset

❖ Data preparation

```
tokenizer = Tokenizer(num_words=vocab_size,
                      oov_token="<OOV>")
tokenizer.fit_on_texts(train_sentences)

# convert text to features
train_sequences = tokenizer.texts_to_sequences(train_sentences)
test_sequences = tokenizer.texts_to_sequences(test_sentences)

# padding or truncating
train_padded_sequences = pad_sequences(train_sequences,
                                       maxlen=max_len,
                                       truncating='post',
                                       padding="post")
test_padded_sequences  = pad_sequences(test_sequences,
                                       maxlen=max_len,
                                       truncating='post',
                                       padding="post")
```

sample1: ‘We are learning AI’

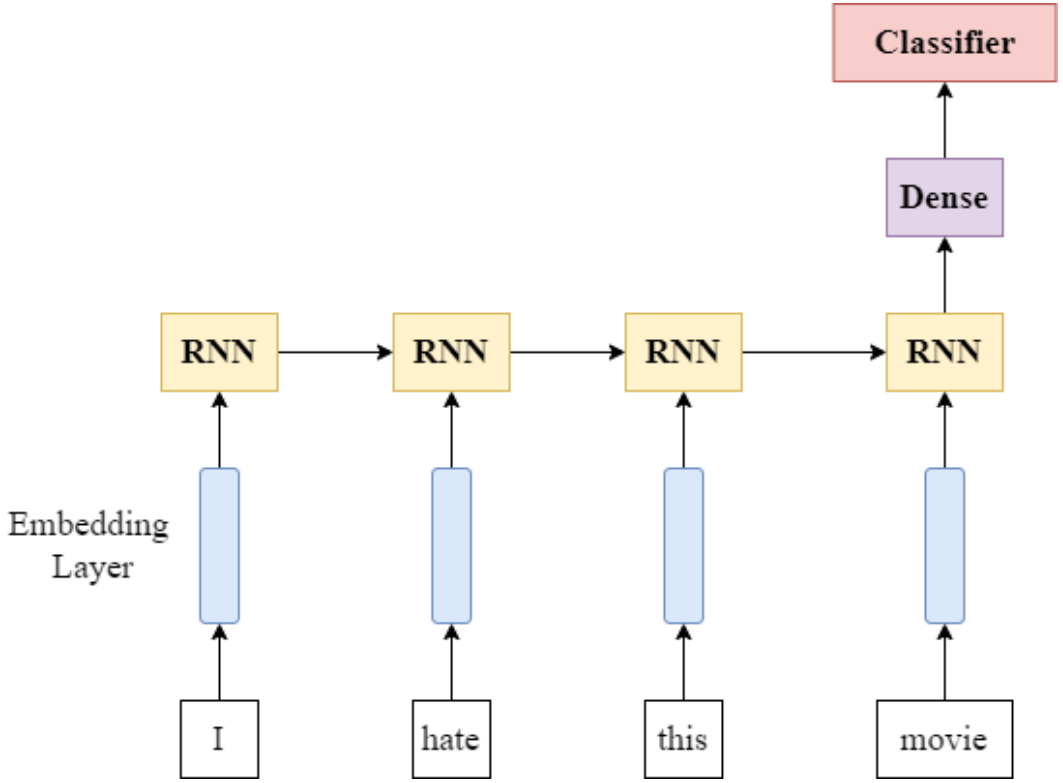
sample2: ‘AI is a CS topic’

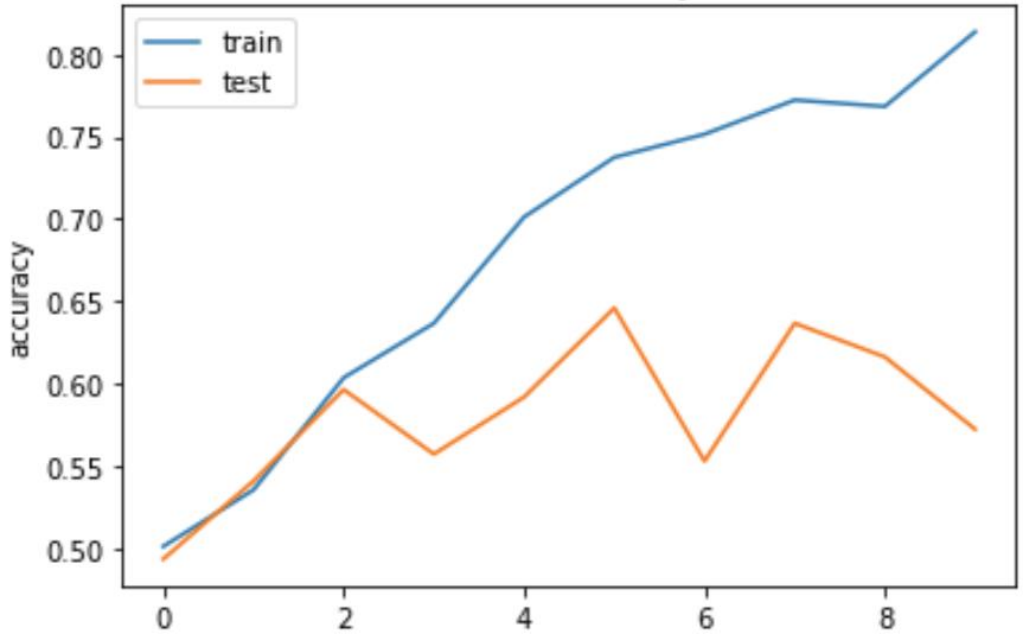


Text Classification

- IMDB dataset
  - Model

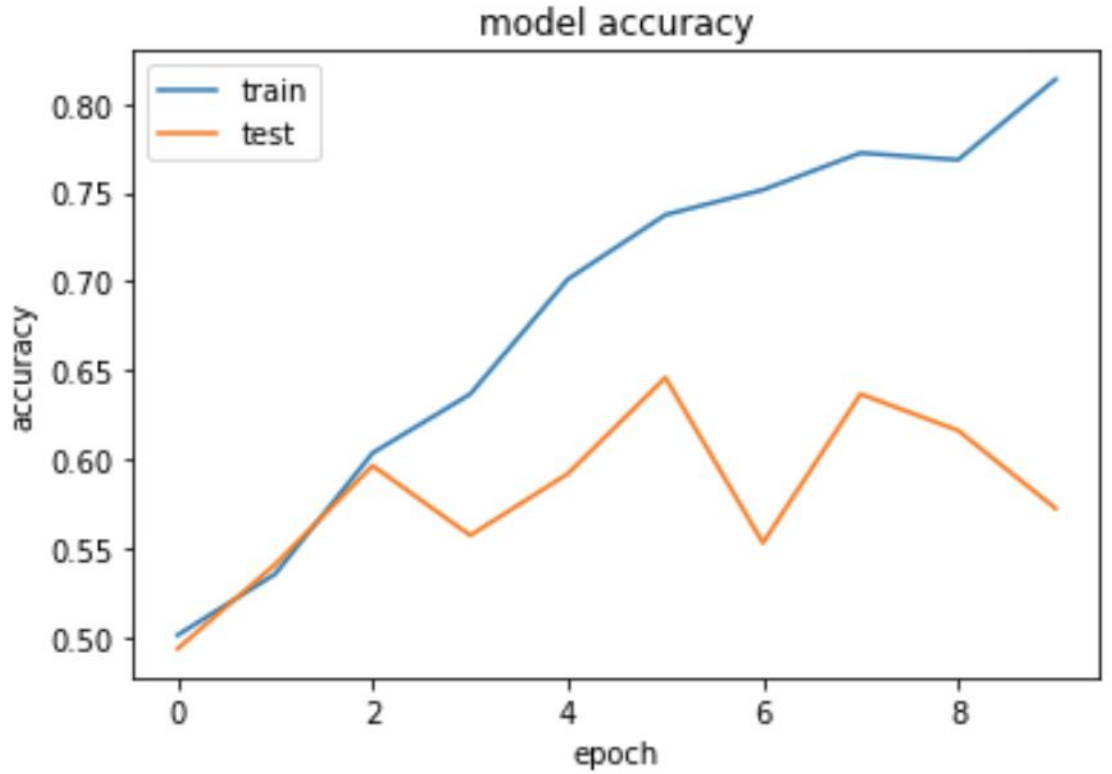
```
vocab_size    = 10000
embedding_dim = 300
max_len       = 100
hidden_size = 256

classifier_model = Sequential([
    Embedding(vocab_size, embedding_dim,
              input_length=max_len),
    SimpleRNN(hidden_size),
    Dense(1, activation='sigmoid')
])

classifier_model.compile(loss='binary_crossentropy',
                         optimizer='adam',
                         metrics=['acc'])
```





Text Classification

❖ **IMDB dataset: Results**




```
1  test_sentence = ["I hate this movie"]
2  test_sequence = tokenizer.texts_to_sequences(test_sentence)
3  test_padded_seqence = pad_sequences(test_sequence,
4                                      maxlen=max_len,
5                                      truncating="post",
6                                      padding="post")
7
8  classifier_model.predict(test_padded_seqence)
```

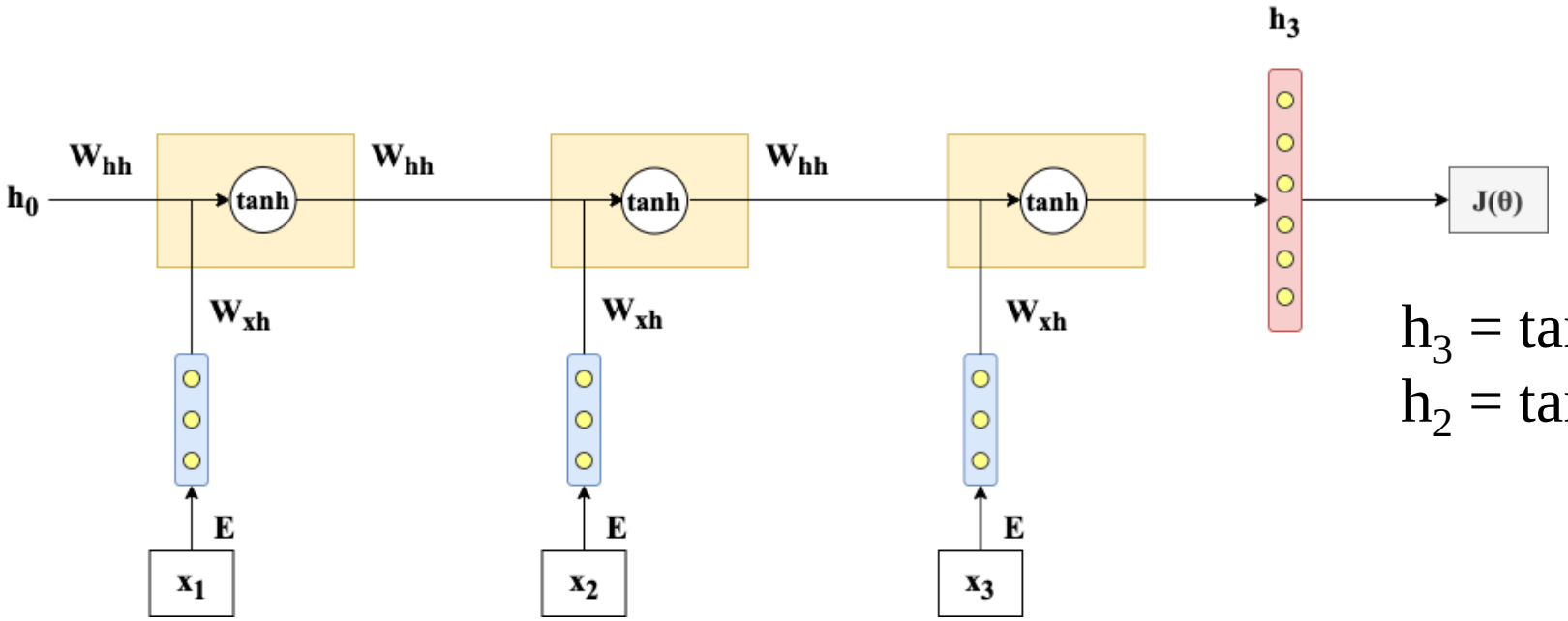
```
array([[0.1827561]], dtype=float32)
```

1.2.[colab]-rnn_TextClassification

Text Deep Models

❖ Recurrent Neural Networks (RNN) - Classification

Backpropagation



Loss: $J(\theta)$

Compute: $\frac{\partial J}{\partial W_{xh}}$

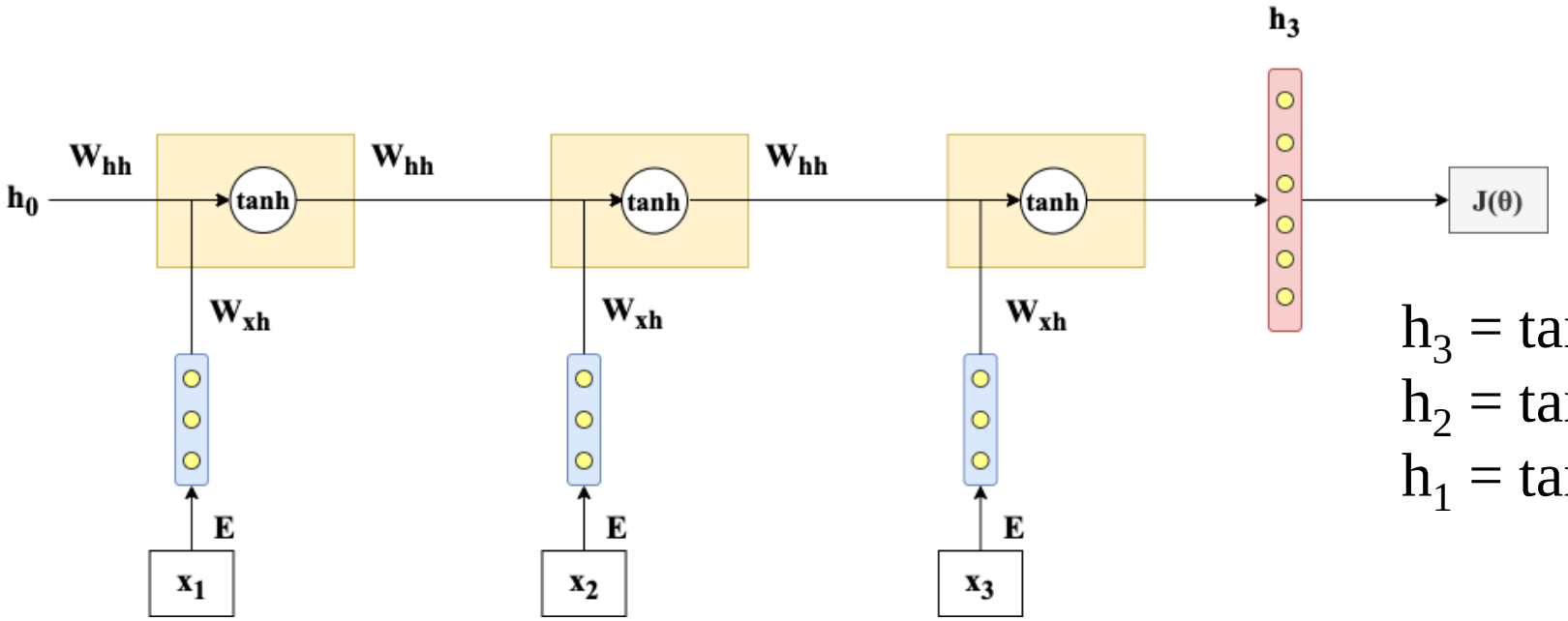
$$h_3 = \tanh(W_{hh}h_2 + W_{xh}x_3 + b_h)$$
$$h_2 = \tanh(W_{hh}h_1 + W_{xh}x_2 + b_h)$$

$$\frac{\partial J}{\partial W_{xh}} = \frac{\partial J}{\partial h_3}\frac{\partial h_3}{\partial W_{xh}} + \frac{\partial J}{\partial h_3}\frac{\partial h_3}{\partial h_2}$$

Text Deep Models

❖ Recurrent Neural Networks (RNN) - Classification

Backpropagation



Loss: $J(\theta)$

Compute: $\frac{\partial J}{\partial W_{xh}}$

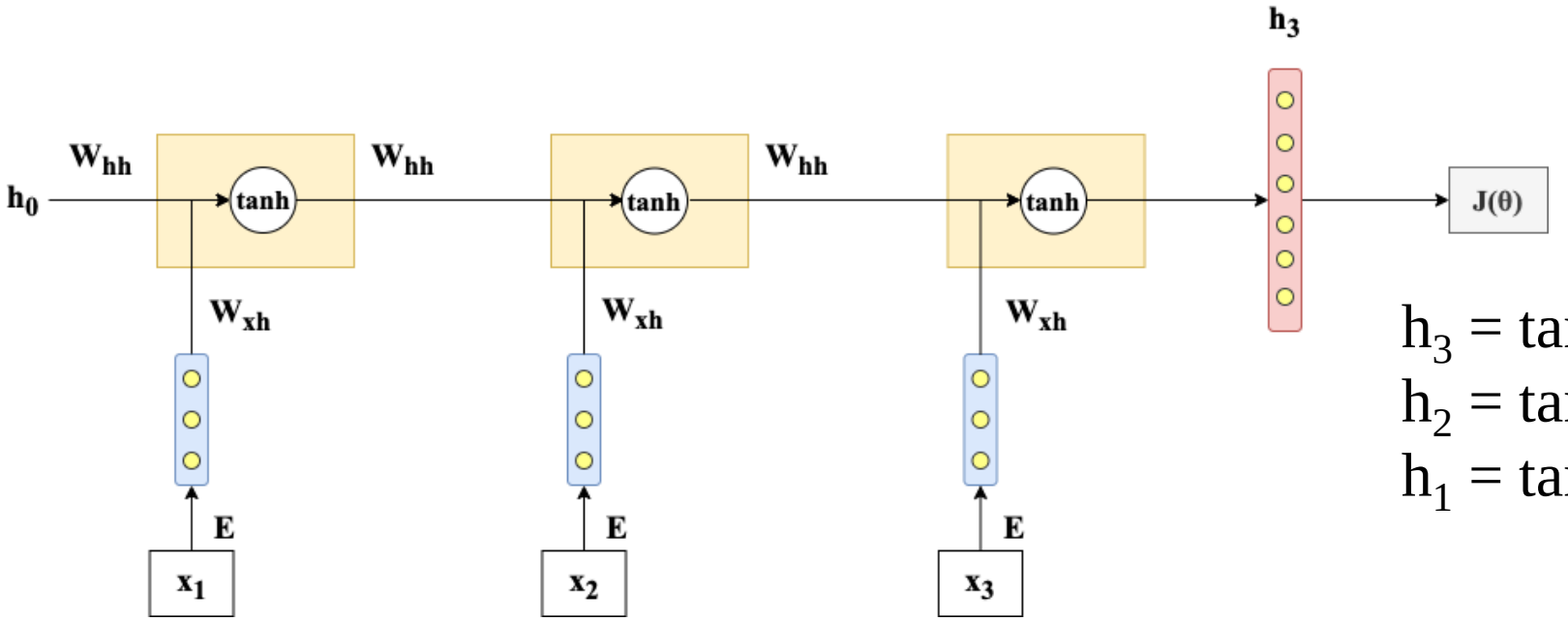
$$h_3 = \tanh(W_{hh}h_2 + W_{xh}x_3 + b_h)$$
$$h_2 = \tanh(W_{hh}h_1 + W_{xh}x_2 + b_h)$$
$$h_1 = \tanh(W_{hh}h_0 + W_{xh}x_1 + b_h)$$

$$\frac{\partial J}{\partial W_{xh}} = \frac{\partial J}{\partial h_3}\frac{\partial h_3}{\partial W_{xh}} + \frac{\partial J}{\partial h_3}\frac{\partial h_3}{\partial h_2}\left(\frac{\partial h_2}{\partial W_{xh}} + \frac{\partial h_2}{\partial h_1}\right) = \frac{\partial J}{\partial h_3}\frac{\partial h_3}{\partial W_{xh}} + \frac{\partial J}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial W_{xh}} + \frac{\partial J}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial h_1}$$

Text Deep Models

❖ Recurrent Neural Networks (RNN) - Classification

Backpropagation



Loss: $J(\theta)$

Compute: $\frac{\partial J}{\partial W_{xh}}$

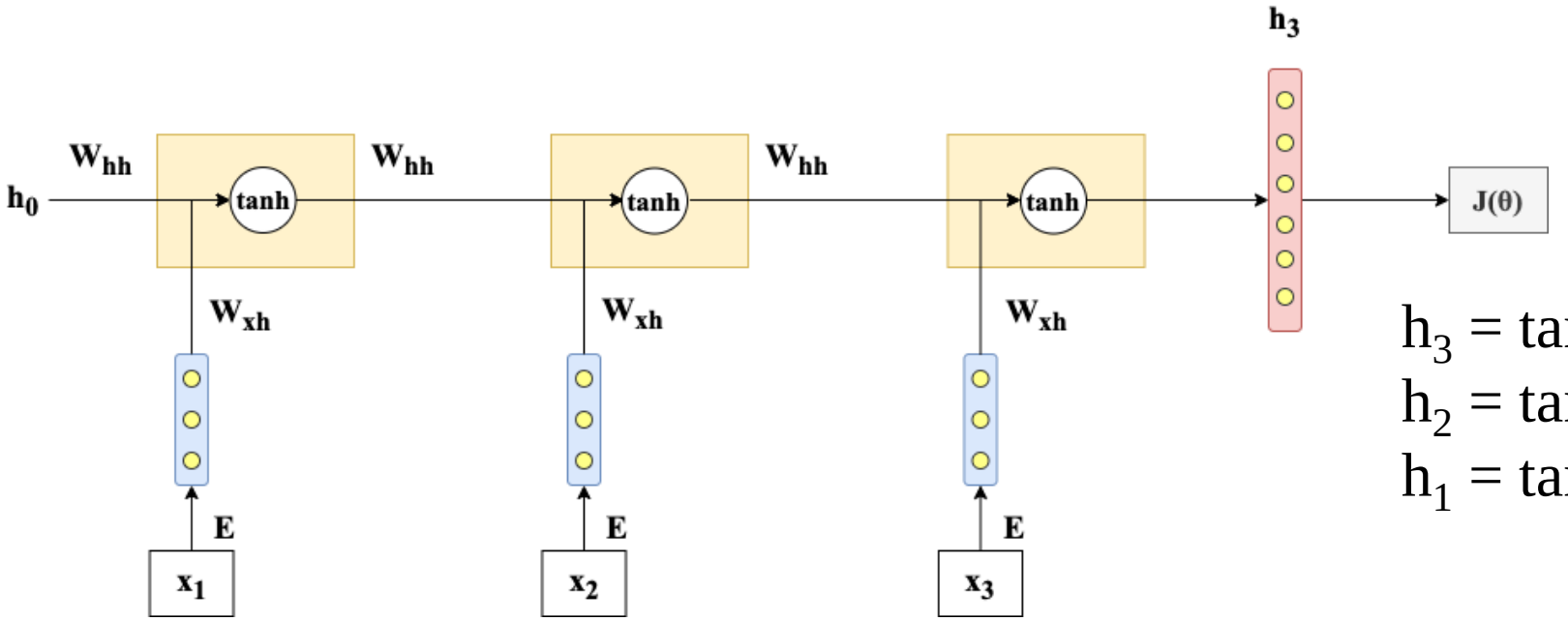
$$h_3 = \tanh(W_{hh}h_2 + W_{xh}x_3 + b_h)$$
$$h_2 = \tanh(W_{hh}h_1 + W_{xh}x_2 + b_h)$$
$$h_1 = \tanh(W_{hh}h_0 + W_{xh}x_1 + b_h)$$

$$\frac{\partial J}{\partial W_{xh}} = \frac{\partial J}{\partial h_3}\frac{\partial h_3}{\partial W_{xh}} + \frac{\partial J}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial W_{xh}} + \frac{\partial J}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial h_1}\frac{\partial h_1}{\partial W_{xh}}$$

Text Deep Models

❖ Recurrent Neural Networks (RNN) - Classification

Backpropagation



Loss: $J(\theta)$

Compute: $\frac{\partial J}{\partial W_{xh}}$

$$h_3 = \tanh(W_{hh}h_2 + W_{xh}x_3 + b_h)$$
$$h_2 = \tanh(W_{hh}h_1 + W_{xh}x_2 + b_h)$$
$$h_1 = \tanh(W_{hh}h_0 + W_{xh}x_1 + b_h)$$

$$\frac{\partial J}{\partial W_{xh}} = \sum_{k=1}^{T} \frac{\partial J}{\partial h_T} \underbrace{\frac{\partial h_T}{\partial h_k}} \frac{\partial h_k}{\partial W_{xh}}$$

$$\underbrace{\frac{\partial h_T}{\partial h_{T-1}} \frac{\partial h_{T-1}}{\partial h_{T-2}} \cdots \frac{\partial h_{k+2}}{\partial h_{k+1}} \frac{\partial h_{k+1}}{\partial h_k}}$$

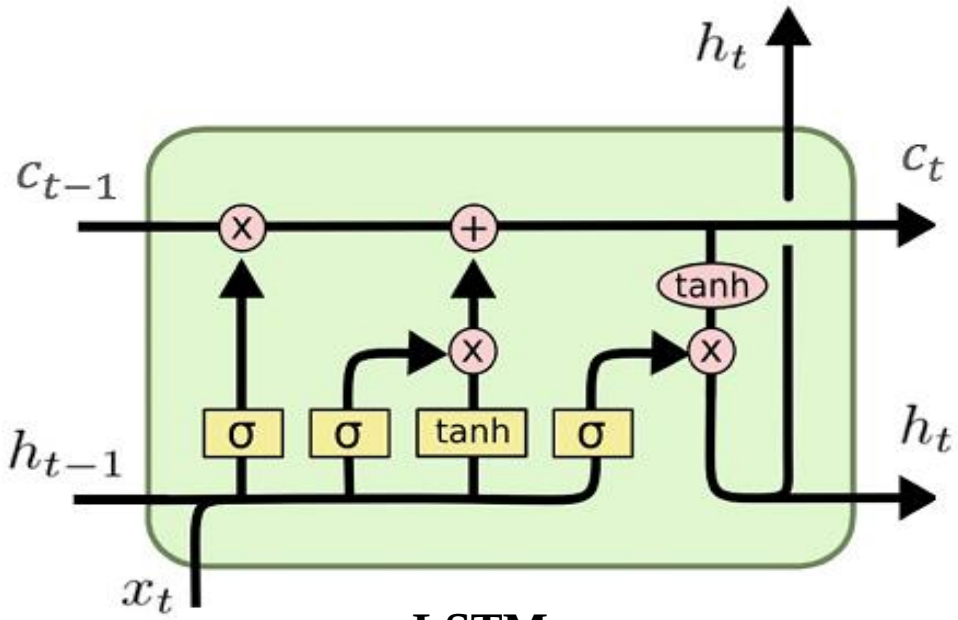
Outline

- Dealing with Text
- Text Classification
- RNN and LSTM
- Bidirectional RNN/LSTM
- RNN/LSTM for Time-series data
- Introduction to NLP (optional)

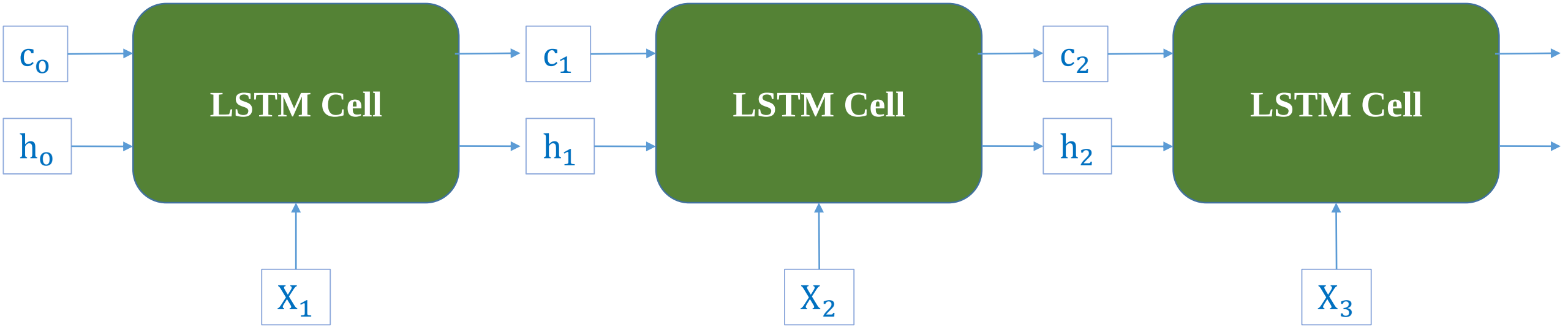
Text Deep Models

❖ Long short term memory

```
tf.keras.layers.LSTM(
    units,
    activation='tanh',
    recurrent_activation='sigmoid',
    return_sequences=False)
```



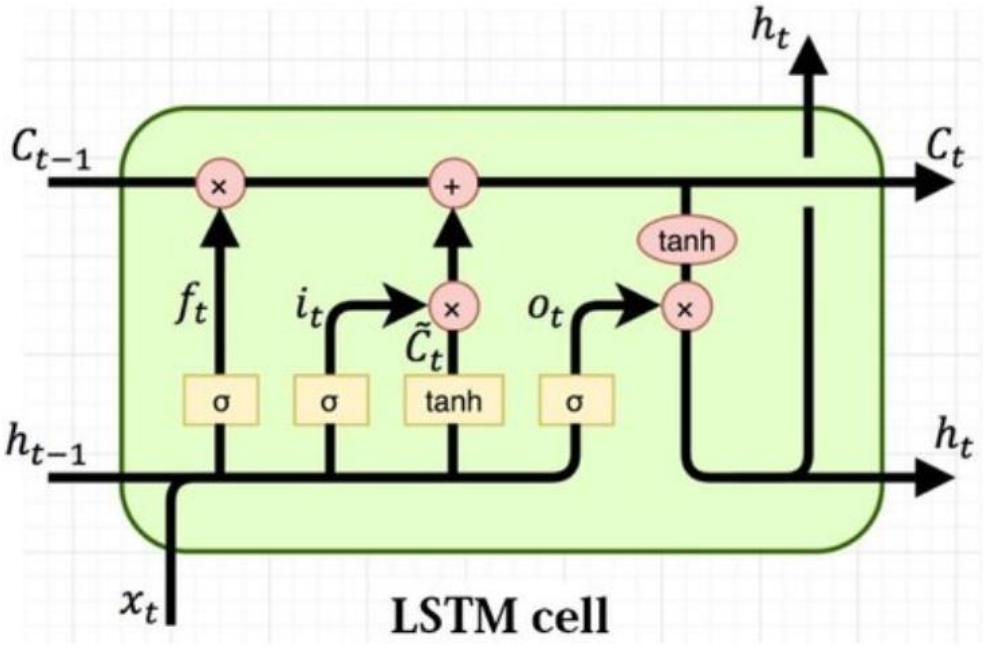
LSTM



Text Deep Models

Long short term memory

$$
\begin{aligned}
i_t &= \sigma\left(x_t U^i + h_{t-1} W^i\right) \\
f_t &= \sigma\left(x_t U^f + h_{t-1} W^f\right) \\
o_t &= \sigma\left(x_t U^o + h_{t-1} W^o\right) \\
\tilde{C}_t &= \tanh\left(x_t U^g + h_{t-1} W^g\right) \\
C_t &= \sigma\left(f_t * C_{t-1} + i_t * \tilde{C}_t\right) \\
h_t &= \tanh(C_t) * o_t
\end{aligned}
$$



LSTM cell

```
tf.keras.layers.LSTM(
    units,
    activation='tanh',
    recurrent_activation='sigmoid',
    return_sequences=False)
```

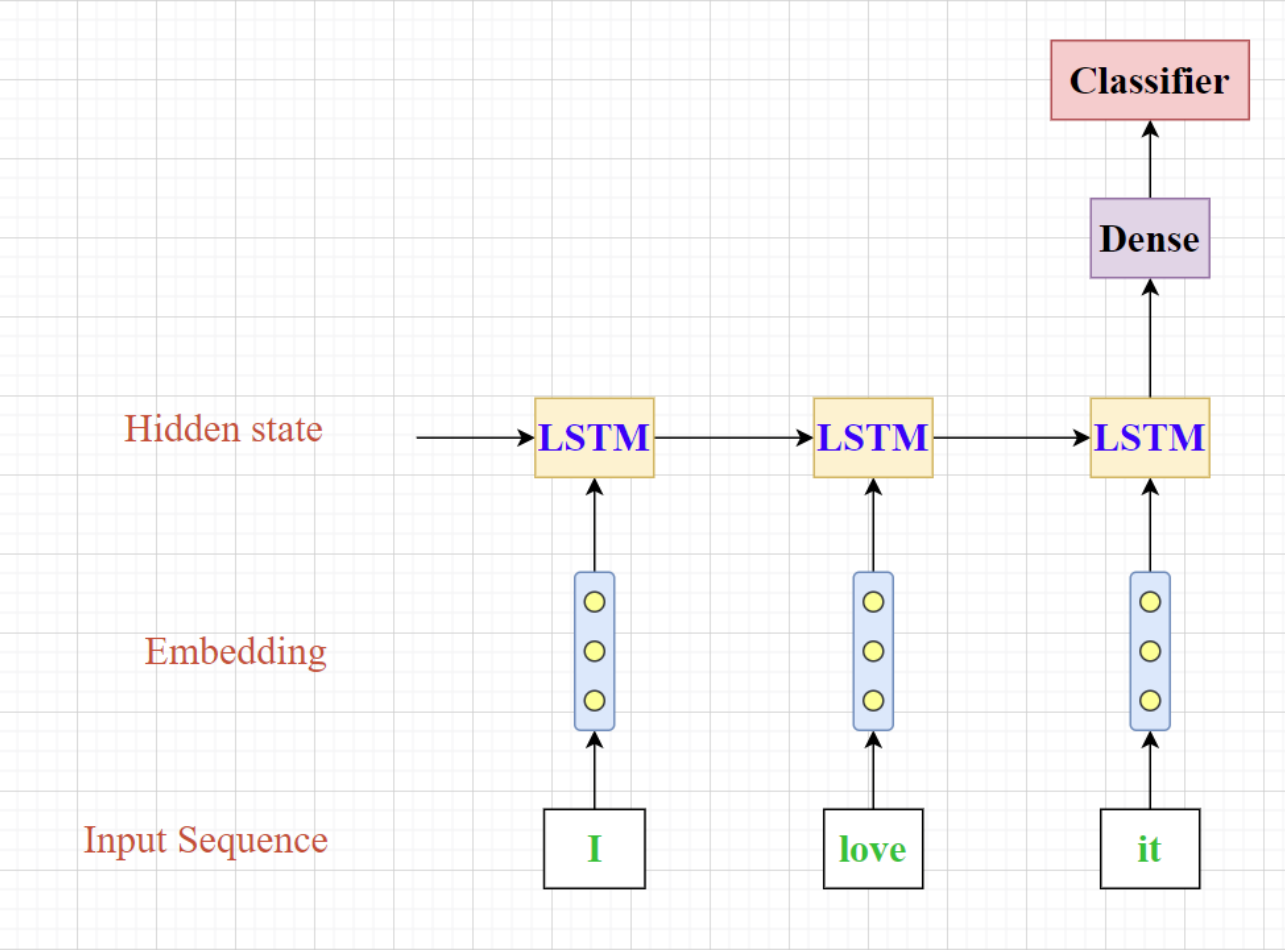
```
inputs = keras.Input(shape=(5,), dtype="int32")
x = layers.Embedding(100, 4)(inputs)

# Add SimpleRNN
x = layers.LSTM(3)(x)

# Add a classifier
x = layers.Dense(1, activation="sigmoid")(x)

# model
model = keras.Model(inputs, x)
```

```
=================================================================
 input_1 (InputLayer)        [(None, 5)]               0

 embedding (Embedding)       (None, 5, 4)              400

 lstm (LSTM)                 (None, 3)                 96

 dense (Dense)               (None, 1)                 4

=================================================================
Total params: 500
Trainable params: 500
Non-trainable params: 0
```

Text Deep Models

❖ LSTM

```
tf.keras.layers.LSTM(
    units,
    activation='tanh',
    recurrent_activation='sigmoid',
    return_sequences=False)
```



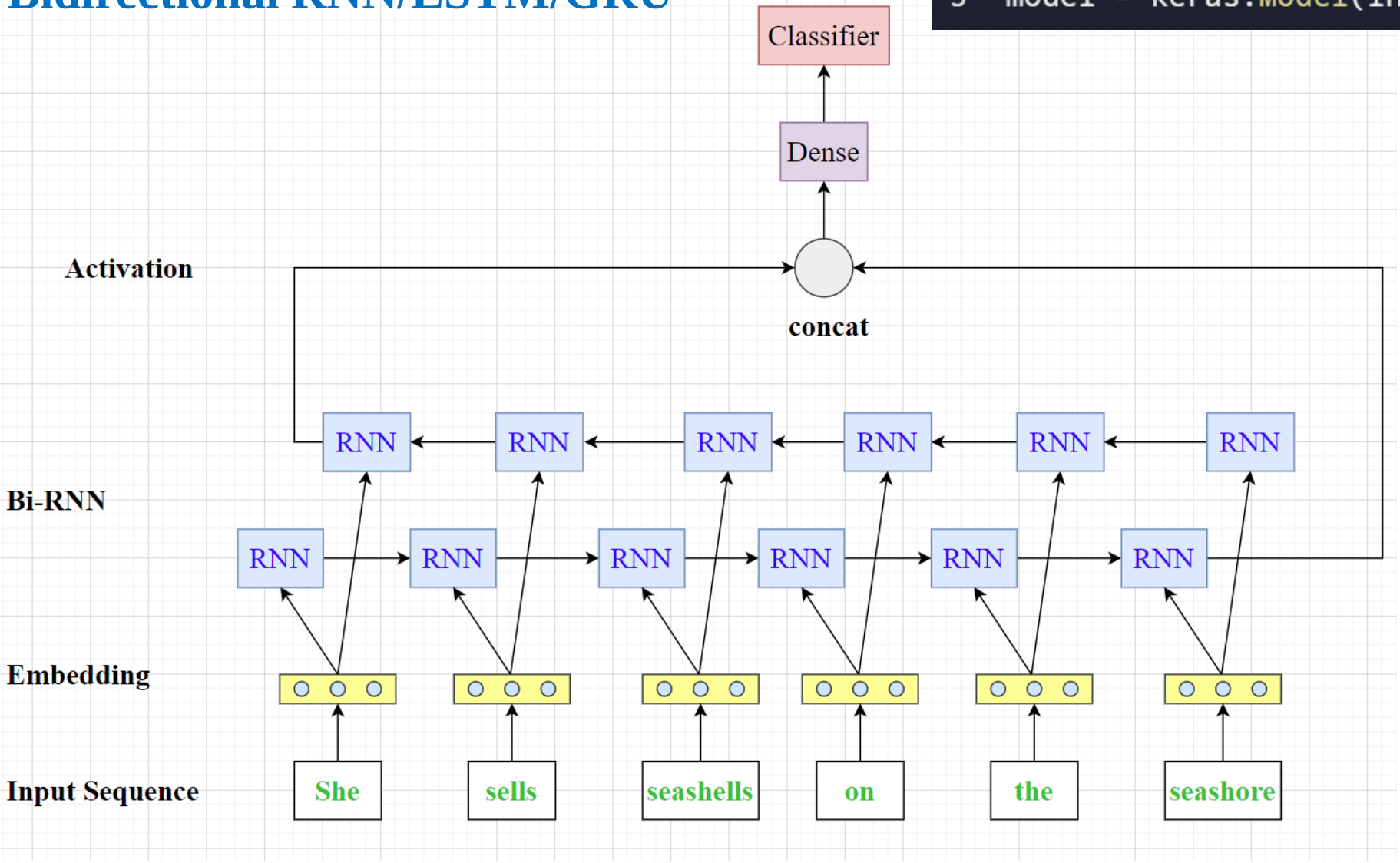

```
inputs = keras.Input(shape=(5,), dtype="int32")
x = layers.Embedding(100, 4)(inputs)

# Add SimpleRNN
x = layers.LSTM(3)(x)

# Add a classifier
x = layers.Dense(1, activation="sigmoid")(x)

# model
model = keras.Model(inputs, x)
```

```
=================================================================
 input_1 (InputLayer)        [(None, 5)]               0

 embedding (Embedding)       (None, 5, 4)              400

 lstm (LSTM)                 (None, 3)                 96

 dense (Dense)               (None, 1)                 4

=================================================================
Total params: 500
Trainable params: 500
Non-trainable params: 0
```

Text Deep Models

```
1  inputs = keras.Input(shape=(5,), dtype="int32")
2  x = layers.Embedding(100, 4)(inputs)
3  x = layers.Bidirectional(layers.SimpleRNN(3))(x)
4  x = layers.Dense(1, activation="sigmoid")(x)
5  model = keras.Model(inputs, x)
```

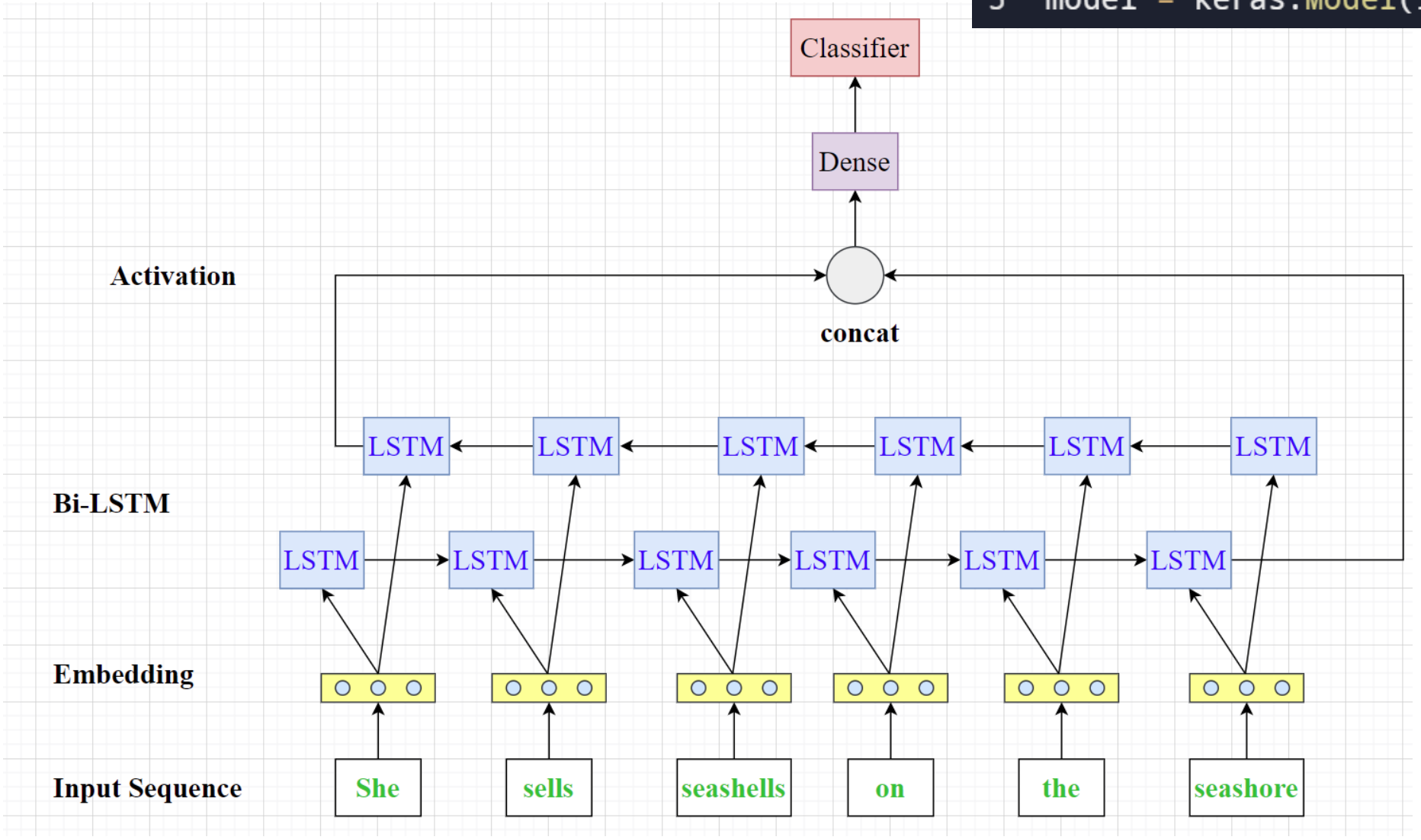
❖ Bidirectional RNN/LSTM/GRU



Text Deep Models

```
1  inputs = keras.Input(shape=(5,), dtype="int32")
2  x = layers.Embedding(100, 4)(inputs)
3  x = layers.Bidirectional(layers.LSTM(3))(x)
4  x = layers.Dense(1, activation="sigmoid")(x)
5  model = keras.Model(inputs, x)
```

❖ Bidirectional RNN/LSTM



AI VIETNAM
All-in-One Course

Text Deep Models

```python
inputs = keras.Input(shape=(5,), dtype="int32")
x = layers.Embedding(100, 4)(inputs)

# Add 2 bidirectional LSTMs
x = layers.Bidirectional(layers.SimpleRNN(3,
                                          return_sequences=True))(x)
x = layers.Bidirectional(layers.SimpleRNN(3))(x)

# Add a classifier
x = layers.Dense(1, activation="sigmoid")(x)

# model
model = keras.Model(inputs, x)
```

```
input_3 (InputLayer)         [(None, 5)]               0

embedding_2 (Embedding)      (None, 5, 4)              400

bidirectional_2 (Bidirectio  (None, 5, 6)              192
nal)

bidirectional_3 (Bidirectio  (None, 6)                 240
nal)

dense_2 (Dense)              (None, 1)                 7
```

```python
inputs = keras.Input(shape=(5,), dtype="int32")
x = layers.Embedding(100, 4)(inputs)

# Add 2 bidirectional LSTMs
x = layers.Bidirectional(layers.LSTM(3,
                                     return_sequences=True))(x)
x = layers.Bidirectional(layers.LSTM(3))(x)

# Add a classifier
x = layers.Dense(1, activation="sigmoid")(x)

# model
model = keras.Model(inputs, x)
```

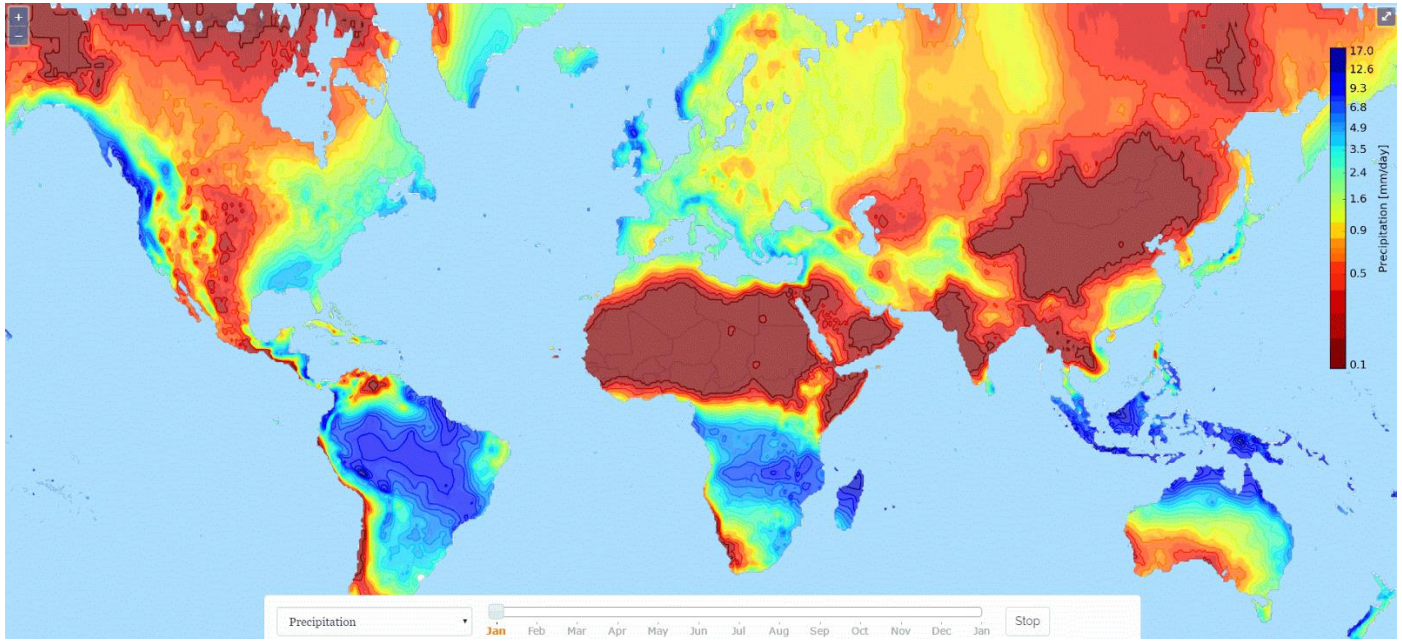
```
input_6 (InputLayer)         [(None, 5)]               0

embedding_5 (Embedding)      (None, 5, 4)              400

bidirectional_6 (Bidirectio  (None, 5, 6)              48
nal)

bidirectional_7 (Bidirectio  (None, 6)                 60
nal)

dense_5 (Dense)              (None, 1)                 7
```

Outline

- Dealing with Text
- Text Classification
- RNN and LSTM
- Bidirectional RNN/LSTM
- RNN/LSTM for Time-series data
- Introduction to NLP (optional)

Weather Forecasting

❖ Introduction



Predict future temperature in weather forecasting

Weather Forecasting

❖ Introduction

Problem Statement: Given temperature from the previous 6 hours (including the current one), predict temperature of the next 1 hour.

Hour	13:00	14:00	15:00	16:00	17:00	18:00	19:00	20:00
Condition								
Temperature	32	31	31	30	29	26	25	

Time-series Data

Temperature forecasting

Date	Summary	Precip Type	Temperature (C)	Apparent Temperature (C)	Humidity	Wind Speed (km/h)	Wind Bearing (degrees)	Visibility (km)
2006-04-01 00	Partly Cloudy	rain	9.472222222	7.388888889	0.89	14.1197	251	15.8263
2006-04-01 01	Partly Cloudy	rain	9.355555556	7.227777778	0.86	14.2646	259	15.8263
2006-04-01 02	Mostly Cloudy	rain	9.377777778	9.377777778	0.89	3.9284	204	14.9569
2006-04-01 03	Partly Cloudy	rain	8.288888889	5.944444444	0.83	14.1036	269	15.8263
2006-04-01 04	Mostly Cloudy	rain	8.755555556	6.977777778	0.83	11.0446	259	15.8263
2006-04-01 05	Partly Cloudy	rain	9.222222222	7.111111111	0.85	13.9587	258	14.9569
2006-04-01 06	Partly Cloudy	rain	7.733333333	5.522222222	0.95	12.3648	259	9.982
2006-04-01 07	Partly Cloudy	rain	8.772222222	6.527777778	0.89	14.1519	260	9.982
2006-04-01 08	Partly Cloudy	rain	10.82222222	10.82222222	0.82	11.3183	259	9.982
2006-04-01 09	Partly Cloudy	rain	13.77222222	13.77222222	0.72	12.5258	279	9.982
2006-04-01 10	Partly Cloudy	rain	16.01666667	16.01666667	0.67	17.5651	290	11.2056
2006-04-01 11	Partly Cloudy	rain	17.14444444	17.14444444	0.54	19.7869	316	11.4471
2006-04-01 12	Partly Cloudy	rain	17.8	17.8	0.55	21.9443	281	11.27
2006-04-01 13	Partly Cloudy	rain	17.33333333	17.33333333	0.51	20.6885	289	11.27
2006-04-01 14	Partly Cloudy	rain	18.87777778	18.87777778	0.47	15.3755	262	11.4471
2006-04-01 15	Partly Cloudy	rain	18.91111111	18.91111111	0.46	10.4006	288	11.27
2006-04-01 16	Partly Cloudy	rain	15.38888889	15.38888889	0.6	14.4095	251	11.27
2006-04-01 17	Mostly Cloudy	rain	15.55	15.55	0.63	11.1573	230	11.4471
2006-04-01 18	Mostly Cloudy	rain	14.25555556	14.25555556	0.69	8.5169	163	11.2056
2006-04-01 19	Mostly Cloudy	rain	13.14444444	13.14444444	0.7	7.6314	139	11.2056
2006-04-01 20	Mostly Cloudy	rain	11.55	11.55	0.77	7.3899	147	11.0285
2006-04-01 21	Mostly Cloudy	rain	11.18333333	11.18333333	0.76	4.9266	160	9.982
2006-04-01 22	Partly Cloudy	rain	10.11666667	10.11666667	0.79	6.6493	163	15.8263
2006-04-01 23	Mostly Cloudy	rain	10.2	10.2	0.77	3.9284	152	14.9569
2006-04-10 00	Partly Cloudy	rain	10.42222222	10.42222222	0.62	16.9855	150	15.8263
2006-04-10 01	Partly Cloudy	rain	9.911111111	7.566666667	0.66	17.2109	149	15.8263
2006-04-10 02	Mostly Cloudy	rain	11.18333333	11.18333333	0.8	10.8192	163	14.9569
2006-04-10 03	Partly Cloudy	rain	7.155555556	5.044444444	0.79	11.0768	180	15.8263
2006-04-10 04	Partly Cloudy	rain	6.111111111	4.816666667	0.82	6.6493	161	15.8263

Weather Forecasting

❖ Introduction

Time	Temperature (C)
2006-04-01 00:00:00.000 +0200	9.472222
2006-04-01 01:00:00.000 +0200	9.355556
2006-04-01 02:00:00.000 +0200	9.377778
2006-04-01 03:00:00.000 +0200	8.288889
2006-04-01 04:00:00.000 +0200	8.755556
2006-04-01 05:00:00.000 +0200	9.222222
2006-04-01 06:00:00.000 +0200	7.733333

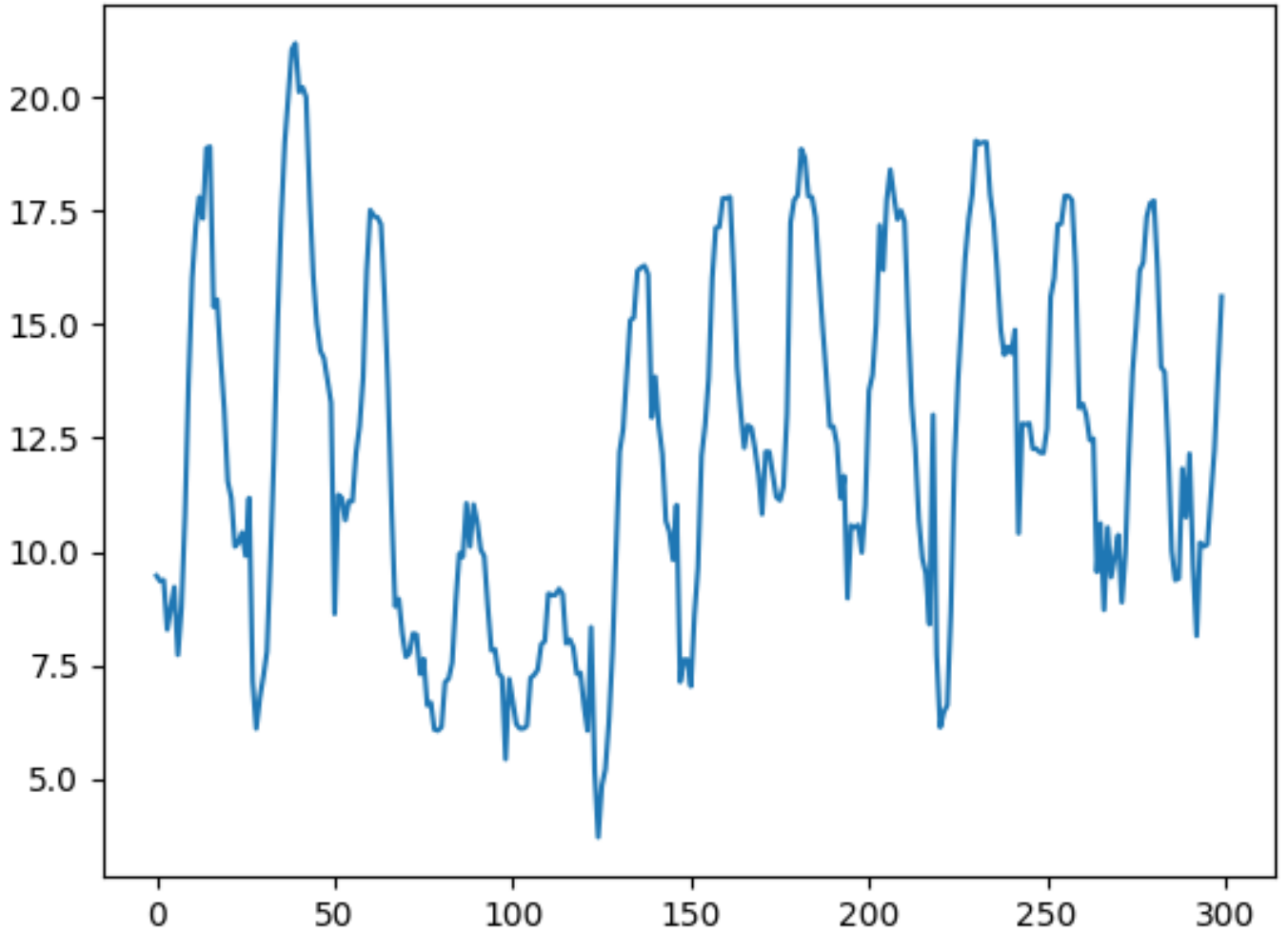
x

y

Temperature forecasting datatable

Time-series Data

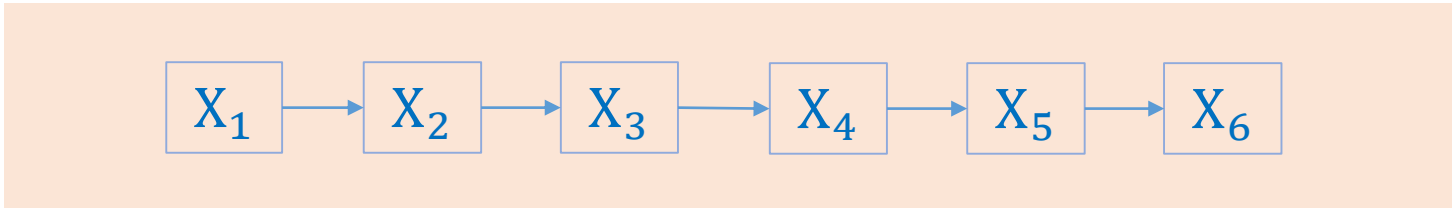
Temperature forecasting

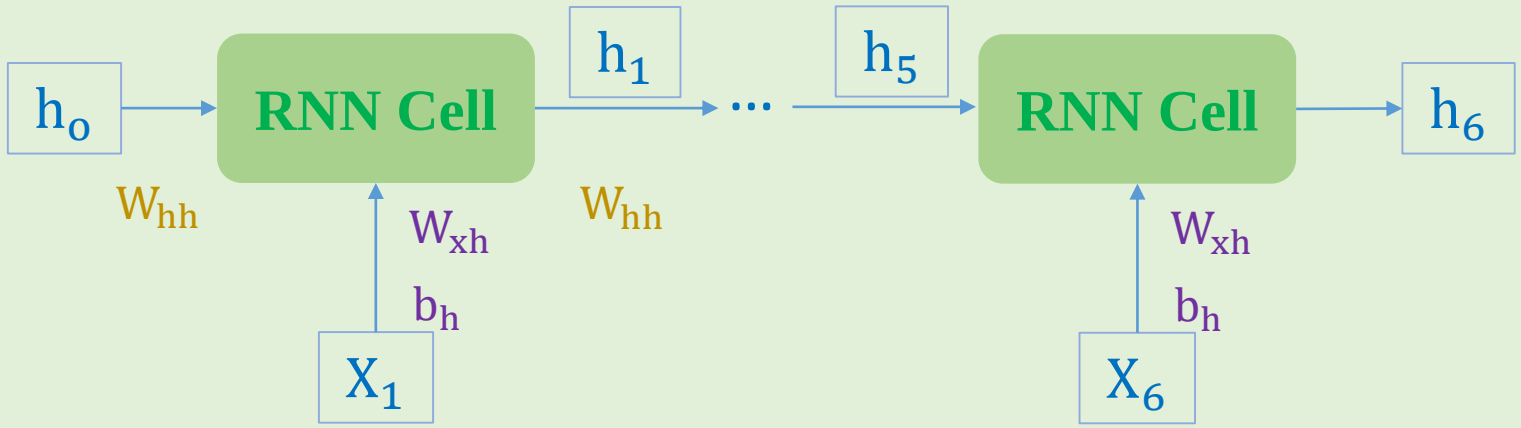


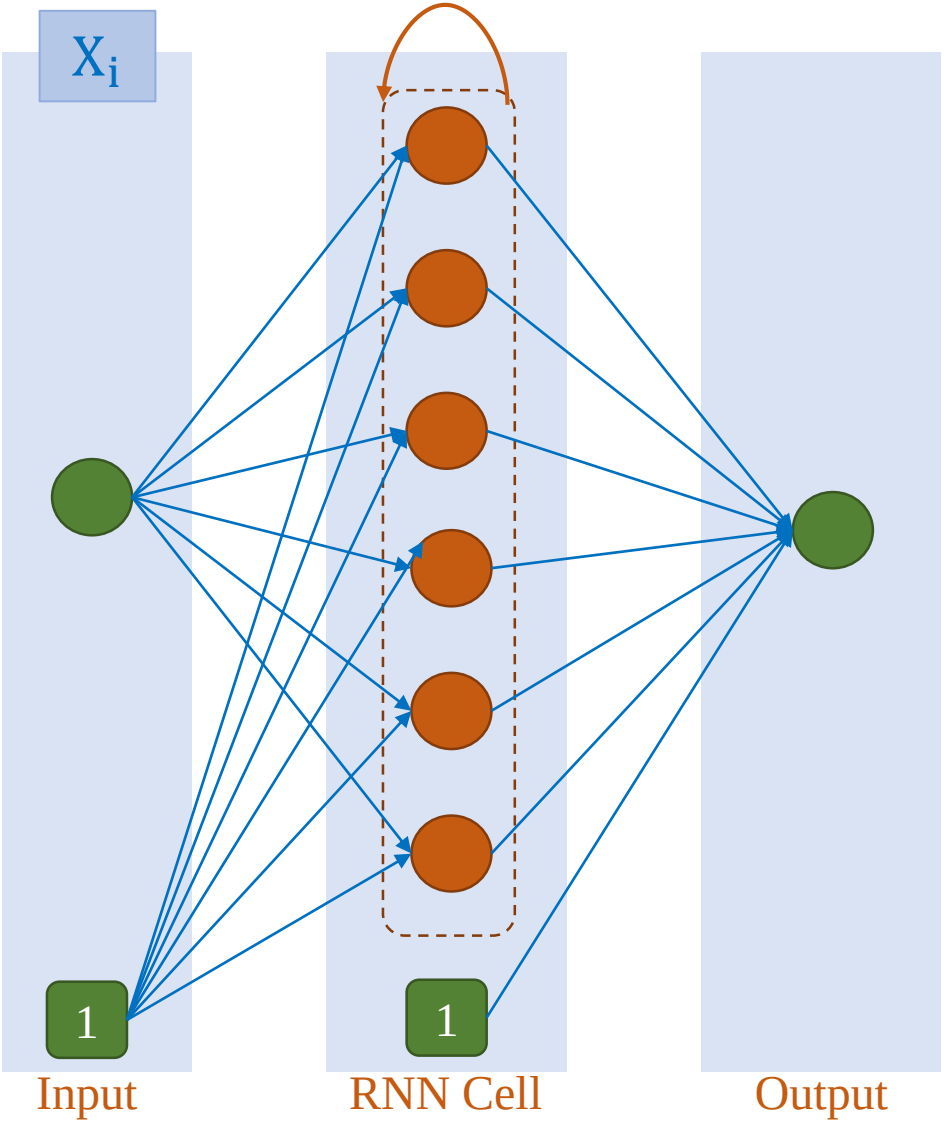
Date	Temperature (C)
2006-04-01 00	9.472222222
2006-04-01 01	9.355555556
2006-04-01 02	9.377777778
2006-04-01 03	8.288888889
2006-04-01 04	8.755555556
2006-04-01 05	9.222222222
2006-04-01 06	7.733333333
2006-04-01 07	8.772222222
2006-04-01 08	10.82222222
2006-04-01 09	13.77222222
2006-04-01 10	16.01666667
2006-04-01 11	17.14444444
2006-04-01 12	17.8
2006-04-01 13	17.33333333
2006-04-01 14	18.87777778
2006-04-01 15	18.91111111
2006-04-01 16	15.38888889
2006-04-01 17	15.55
2006-04-01 18	14.25555556
2006-04-01 19	13.14444444
2006-04-01 20	11.55
2006-04-01 21	11.18333333
2006-04-01 22	10.11666667
2006-04-01 23	10.2
2006-04-10 00	10.42222222
2006-04-10 01	9.911111111
2006-04-10 02	11.18333333
2006-04-10 03	7.155555556
2006-04-10 04	6.111111111

Time-series Data

Temperature forecasting

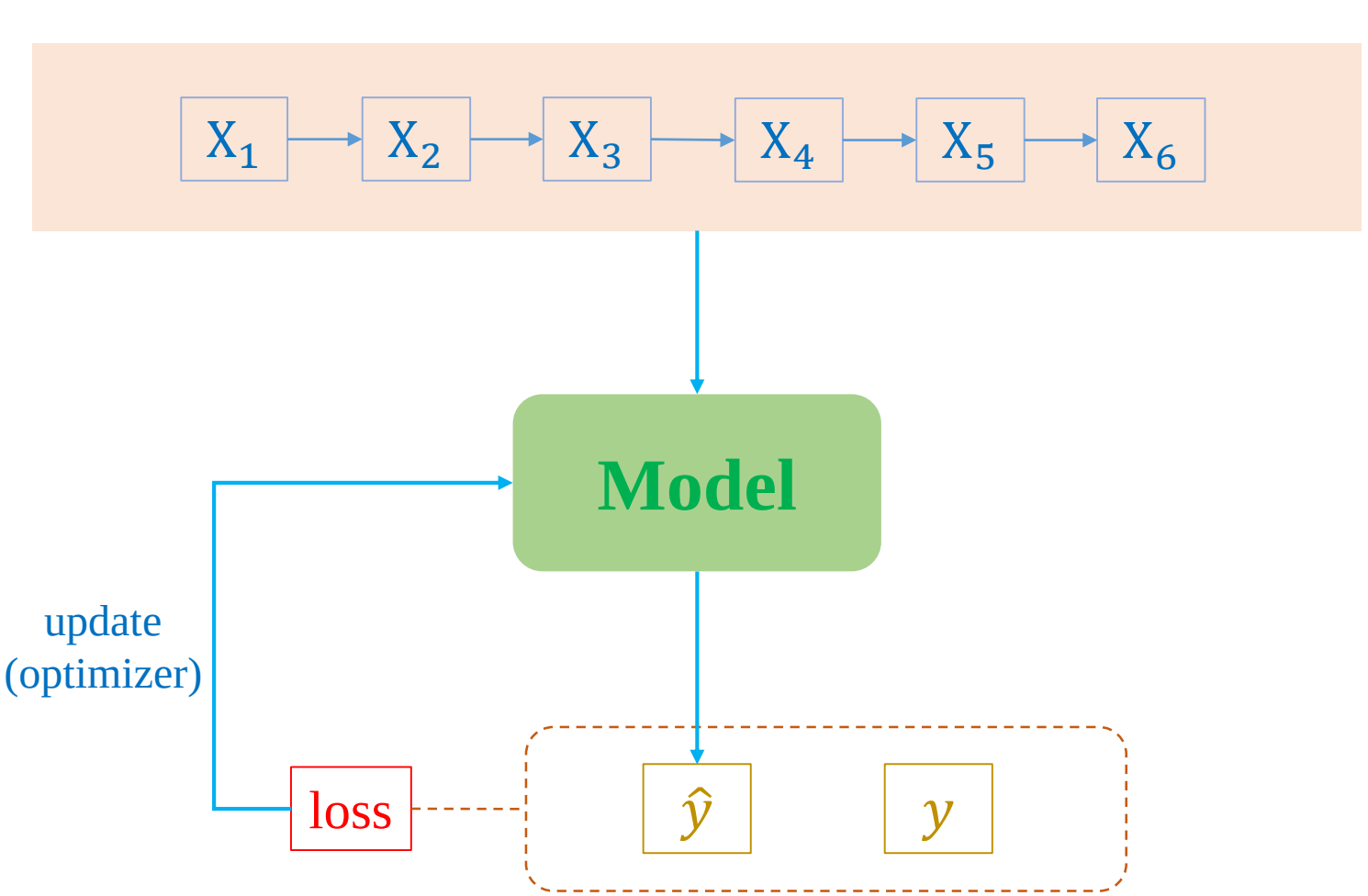


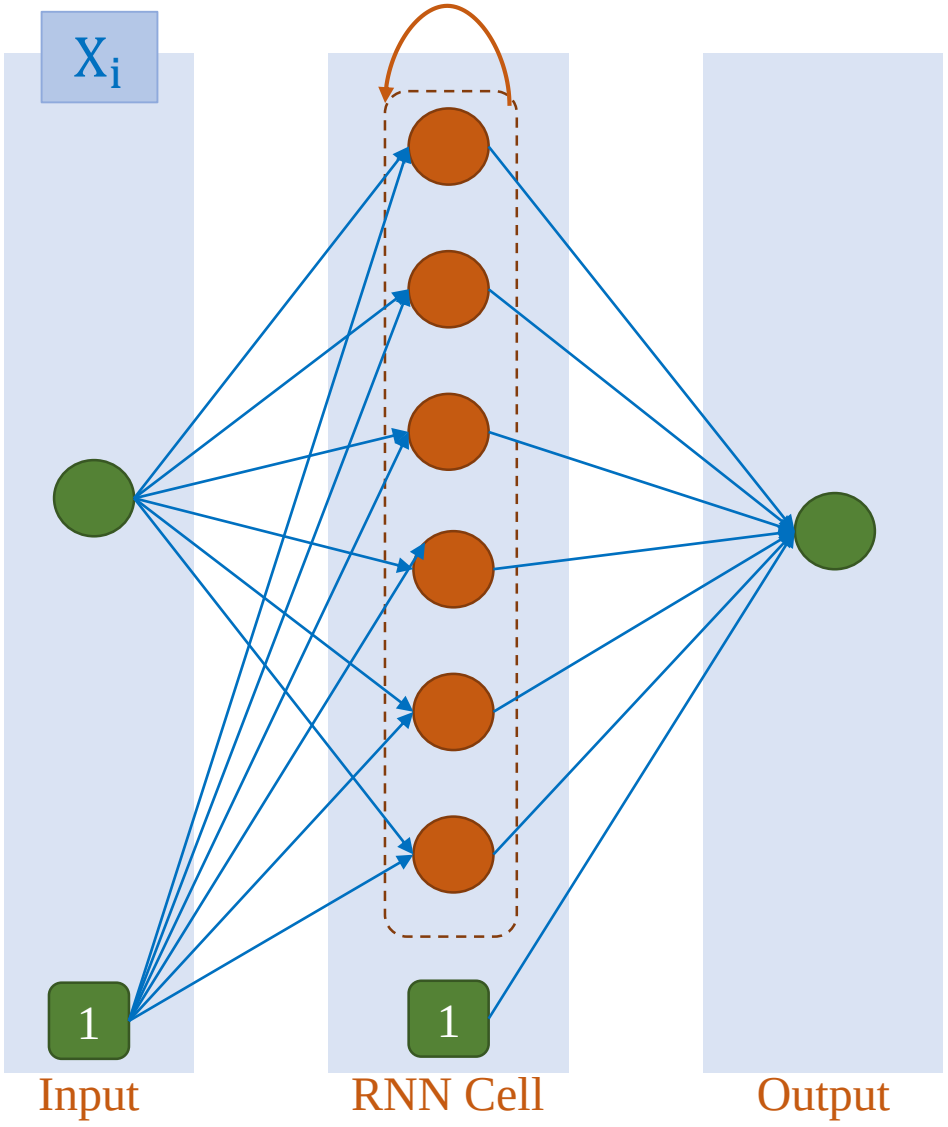




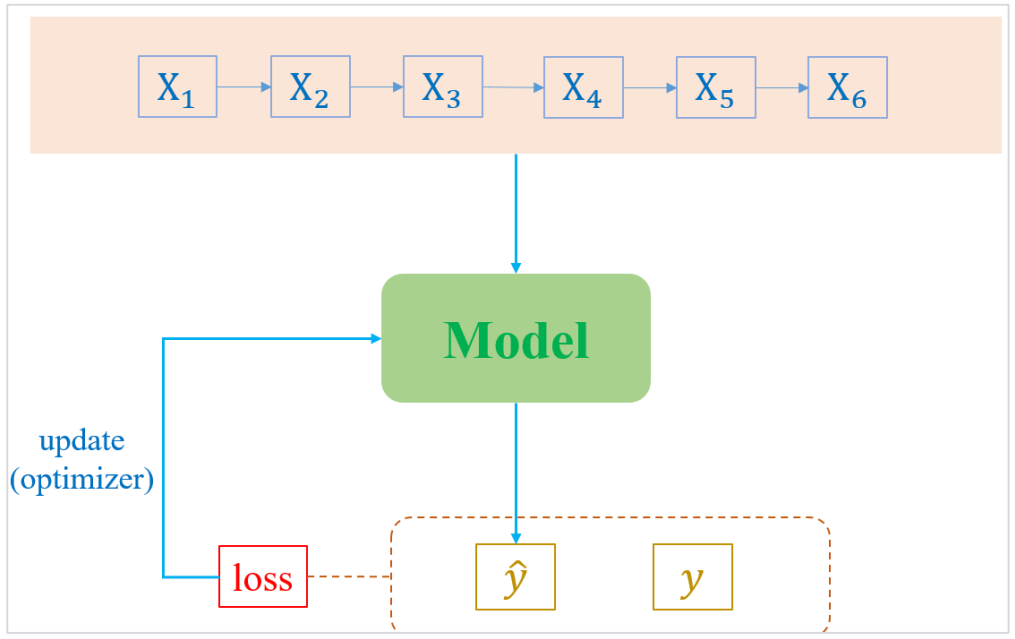
Time-series Data

Temperature forecasting

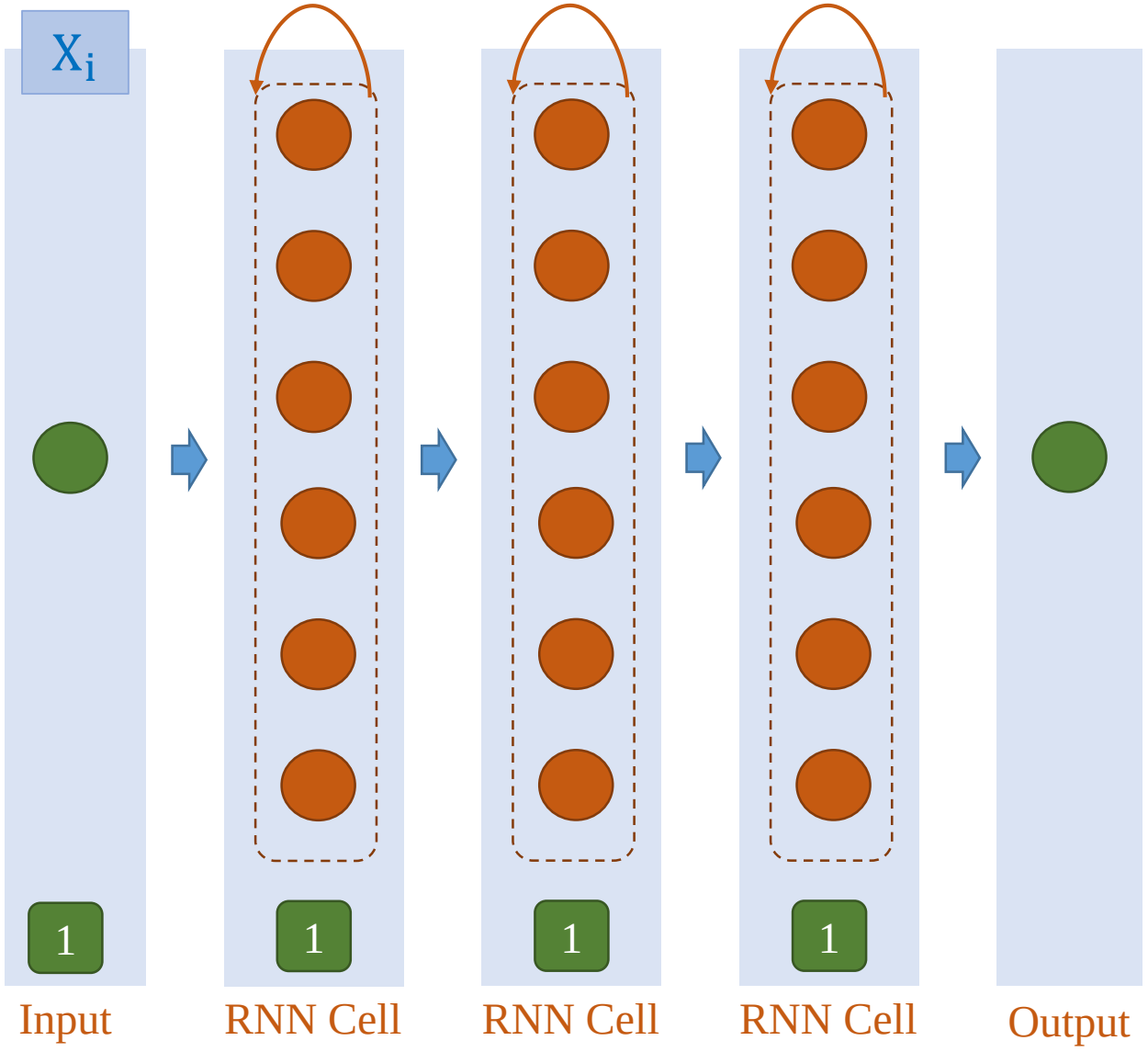




Time-series Data




```python
model = tf.keras.Sequential([
        # input
        tf.keras.Input(shape=(6,1)),

        tf.keras.layers.SimpleRNN(6, return_sequences=True),
        tf.keras.layers.SimpleRNN(6, return_sequences=True),
        tf.keras.layers.SimpleRNN(6),

        # Output Layer
        tf.keras.layers.Dense(1)])
```



Time-series Data

```
model = tf.keras.Sequential([
        tf.keras.Input(shape=(6,1)),
        tf.keras.layers.Flatten(),

        # Dense
        tf.keras.layers.Dense(32, activation='relu'),
        tf.keras.layers.Dense(32, activation='relu'),

        # Output Layer
        tf.keras.layers.Dense(1)])
```

val_loss: 1.425

```
model = tf.keras.Sequential([
        tf.keras.Input(shape=(6,1)),

        tf.keras.layers.SimpleRNN(16, return_sequences=True),
        tf.keras.layers.SimpleRNN(16, return_sequences=True),
        tf.keras.layers.SimpleRNN(16),

        # Output Layer
        tf.keras.layers.Dense(1)])
```

val_loss: 1.629

```
model = tf.keras.Sequential([
        tf.keras.Input(shape=(6,1)),

        tf.keras.layers.LSTM(16, return_sequences=True),
        tf.keras.layers.LSTM(16, return_sequences=True),
        tf.keras.layers.LSTM(16),

        # Output Layer
        tf.keras.layers.Dense(1)])
```

val_loss: 1.394

```
model = tf.keras.Sequential([
        tf.keras.Input(shape=(6,1)),

        layers.Bidirectional(layers.LSTM(16,
                                         return_sequences=True)),
        layers.Bidirectional(layers.LSTM(16,
                                         return_sequences=True)),
        layers.Bidirectional(layers.LSTM(16)),

        tf.keras.layers.Dense(1)])
```

val_loss: 1.396

Time-series Data

```python
model = tf.keras.Sequential([
        tf.keras.Input(shape=(20,1)),
        tf.keras.layers.Flatten(),

        # Dense
        tf.keras.layers.Dense(32, activation='relu'),
        tf.keras.layers.Dense(32, activation='relu'),

        # Output Layer
        tf.keras.layers.Dense(1)])
```

val_loss: 1.271

```python
model = tf.keras.Sequential([
        tf.keras.Input(shape=(20,1)),

        tf.keras.layers.SimpleRNN(16, return_sequences=True),
        tf.keras.layers.SimpleRNN(16, return_sequences=True),
        tf.keras.layers.SimpleRNN(16),

        # Output Layer
        tf.keras.layers.Dense(1)])
```

val_loss: 1.419

```python
model = tf.keras.Sequential([
        tf.keras.Input(shape=(20,1)),

        tf.keras.layers.LSTM(16, return_sequences=True),
        tf.keras.layers.LSTM(16, return_sequences=True),
        tf.keras.layers.LSTM(16),

        # Output Layer
        tf.keras.layers.Dense(1)])
```

val_loss: 1.206

```python
model = tf.keras.Sequential([
        tf.keras.Input(shape=(20,1)),

        layers.Bidirectional(layers.LSTM(16,
                                         return_sequences=True)),
        layers.Bidirectional(layers.LSTM(16,
                                         return_sequences=True)),
        layers.Bidirectional(layers.LSTM(16)),

        tf.keras.layers.Dense(1)])
```

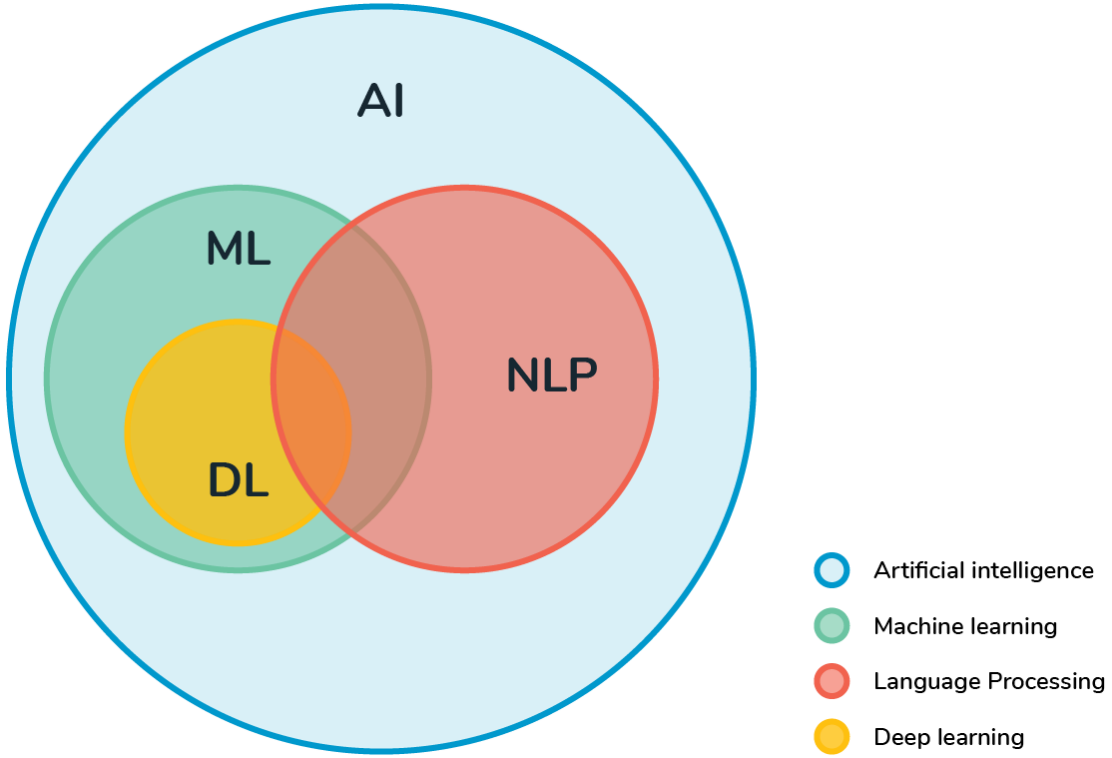
val_loss: 1.212

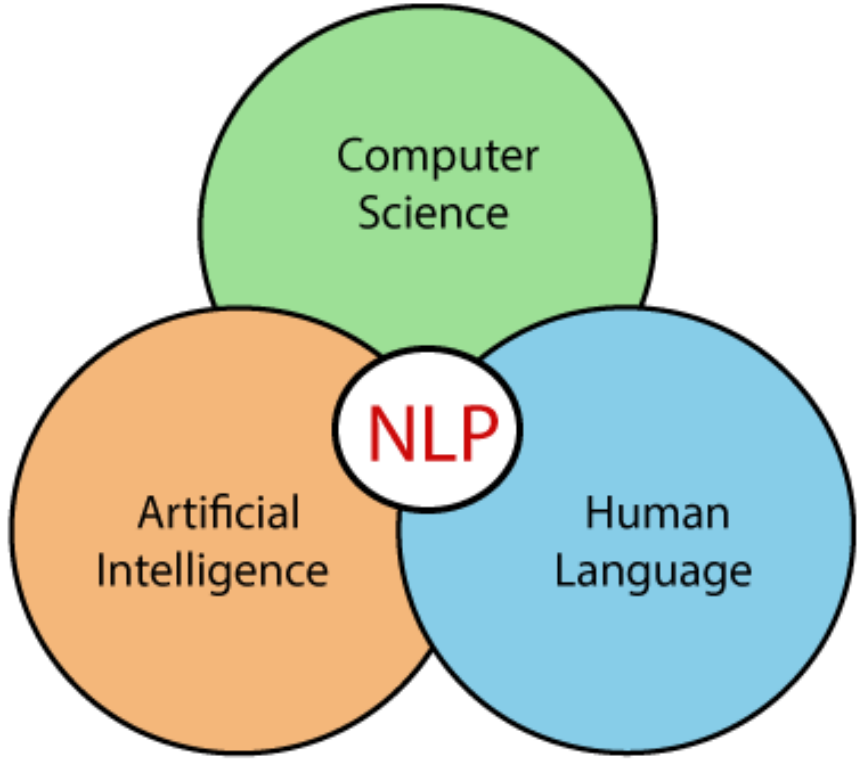
Outline

- Dealing with Text
- Text Classification
- RNN and LSTM
- Bidirectional RNN/LSTM
- RNN/LSTM for Time-series data
- Introduction to NLP (optional)

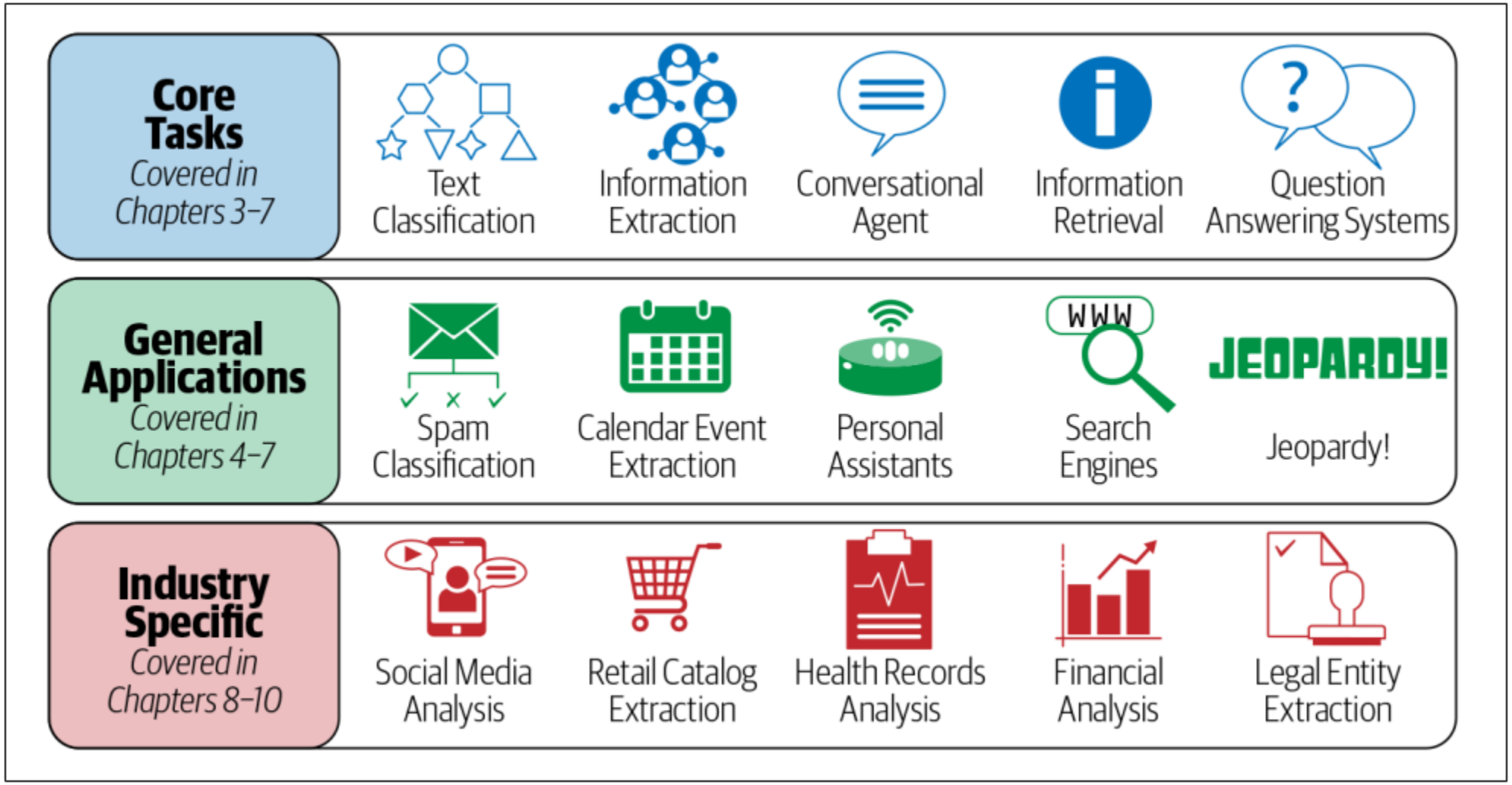
Natural Language Processing

❖ Introduction





NLP Applications



Practical Natural Language Processing by Sowmya Vajjala, Bodhisattwa Majumder, Anuj Gupta, Harshit Surana

NLP Applications

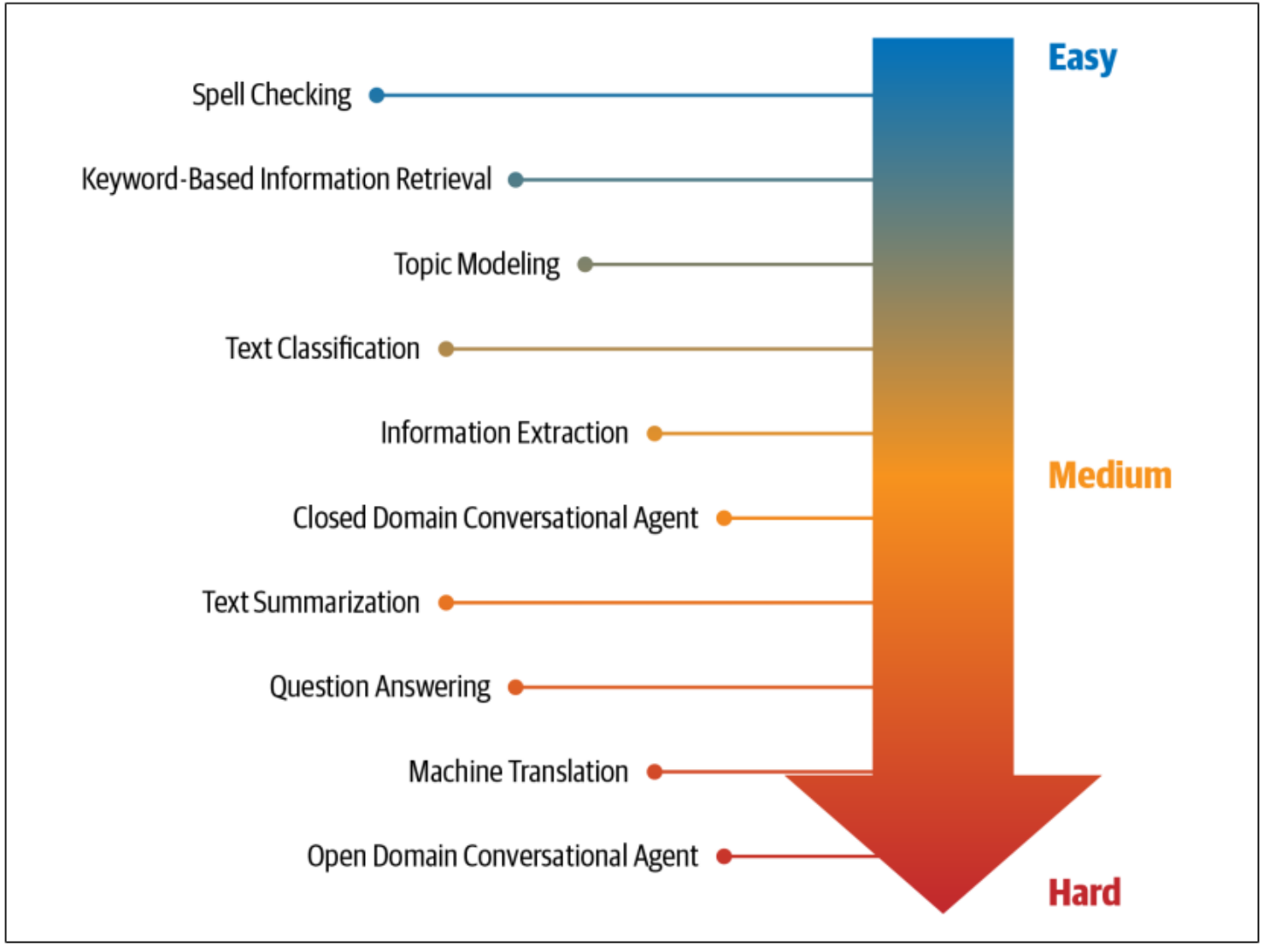
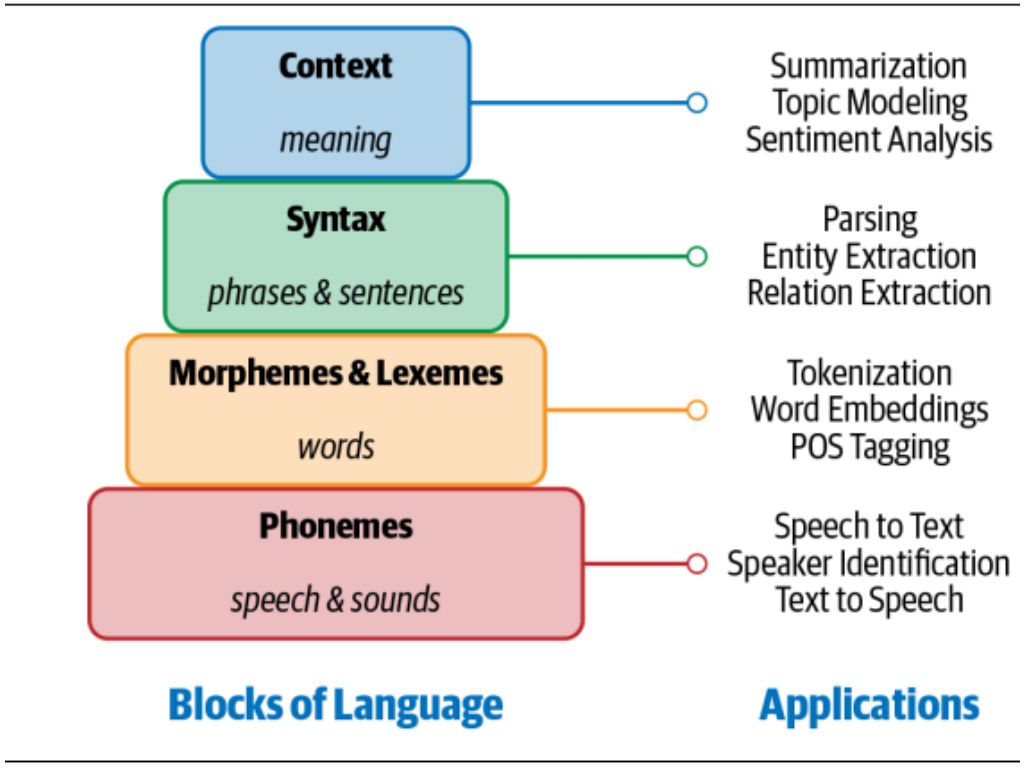


Figure 1-2. NLP tasks organized according to their relative difficulty



Building blocks of language and their applications

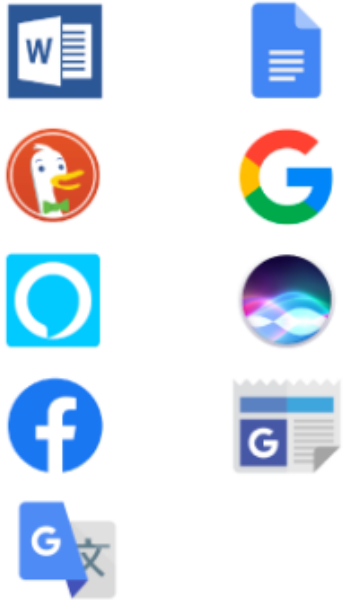
Practical Natural Language Processing by Sowmya Vajjala, Bodhisattwa Majumder, Anuj Gupta, Harshit Surana

NLP Applications

Natural Language Processing, abbreviated as NLP helps machines understand and analyse natural languages. It is a field of Artificial Intelligence that gives machines the ability to read, understand and derive meaning from human languages.

Applications of NLP

- Grammarly, Microsoft Word, Google Docs
- Search engines like DuckDuckGo, Google
- Voice assistants – Alexa, Siri
- News Feeds- Facebook, Google News
- Translation systems – Google translate



https://www.mygreatlearning.com/blog/trending-natural-language-processing-applications/

NLP Applications

❖ Text Classification

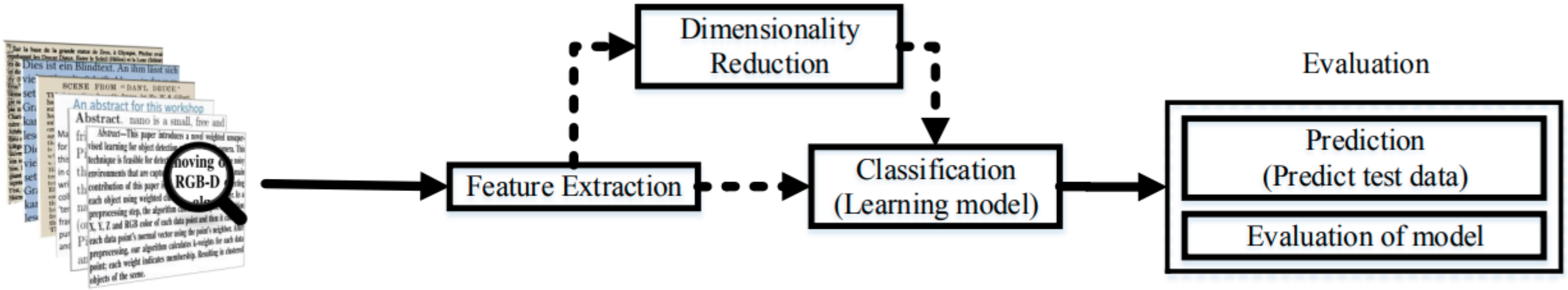


Figure 1. Overview of text classification pipeline.

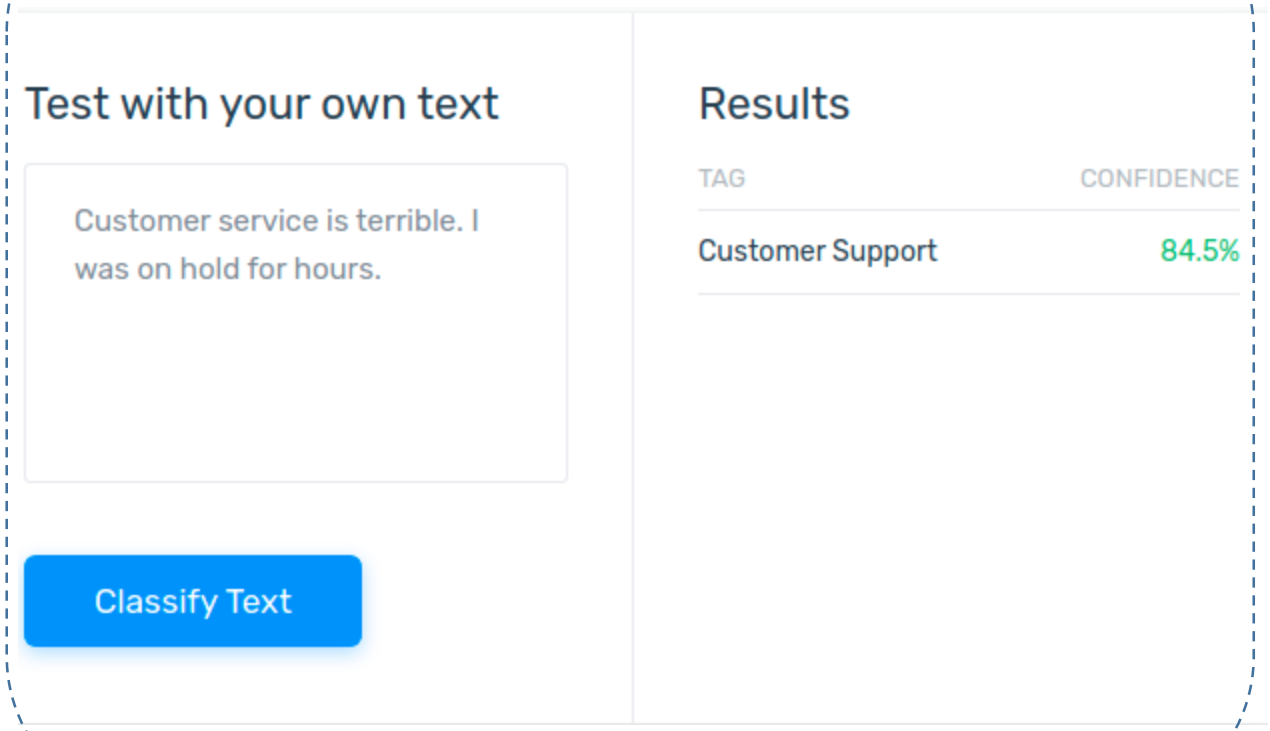
https://codeit.us/blog/nlp-applications-in-business

NLP Applications

❖ Text Classification

Sentiment Analysis

Topic Labeling

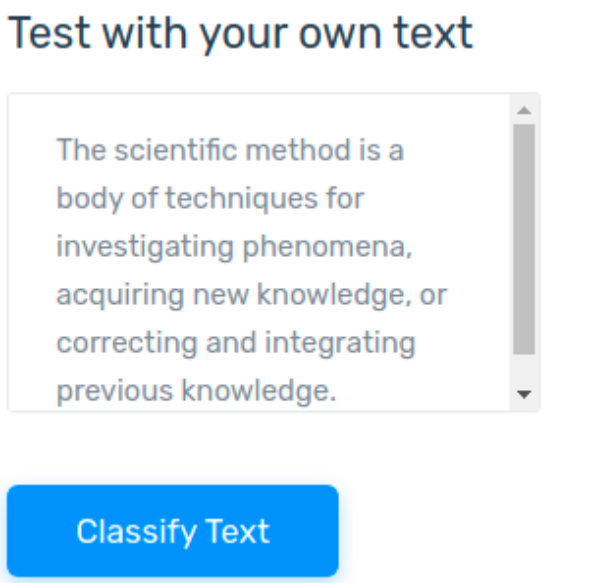


https://codeit.us/blog/nlp-applications-in-business
https://monkeylearn.com/text-classification/

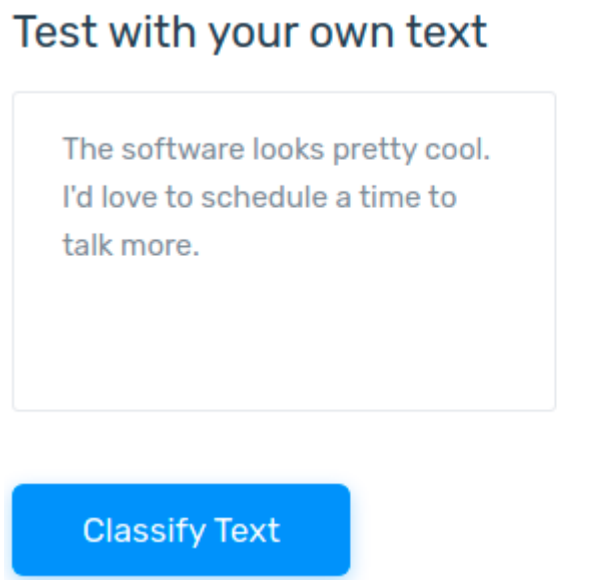
NLP Applications

❖ Text Classification

Language Detection



Intent Detection



https://monkeylearn.com/text-classification/

NLP Applications

❖ Information Extraction

What is "Information Extraction"

As a task: Filling slots in a database from sub-segments of text.

October 14, 2002, 4:00 a.m. PT

For years, Microsoft Corporation CEO Bill Gates railed against the economic philosophy of open-source software with Orwellian fervor, denouncing its communal licensing as a "cancer" that stifled technological innovation.

Today, Microsoft claims to "love" the open-source concept, by which software code is made public to encourage improvement and development by outside programmers. Gates himself says Microsoft will gladly disclose its crown jewels--the coveted code behind the Windows operating system--to select customers.

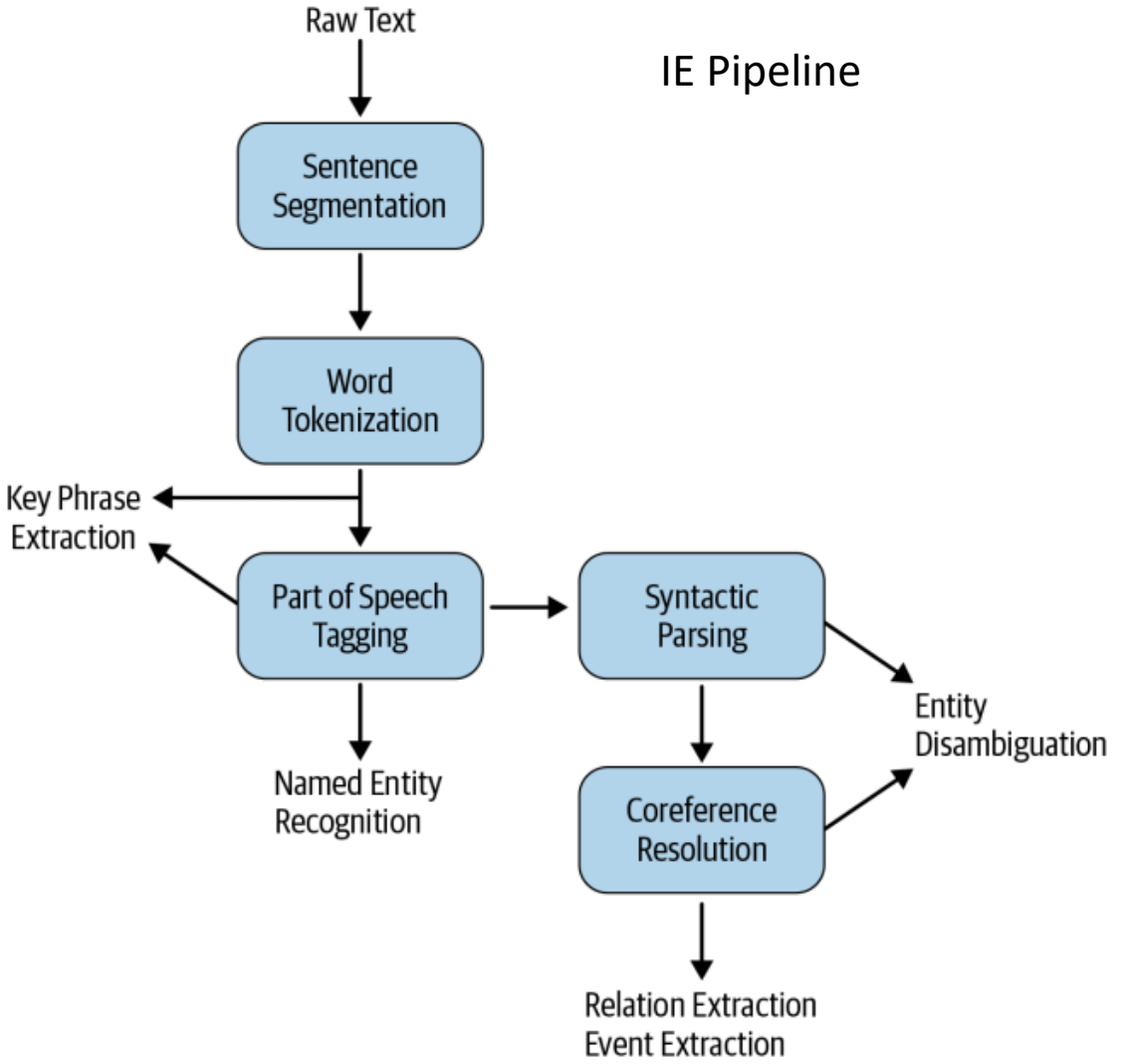
"We can be open source. We love the concept of shared source," said Bill Veghte, a Microsoft VP. "That's a super-important shift for us in terms of code access."

Richard Stallman, founder of the Free Software Foundation, countered saying...

IE

NAME	TITLE	ORGANIZATION
Bill Gates	CEO	Microsoft
Bill Veghte	VP	Microsoft
Richard Stallman	founder	Free Soft..

Slides from Cohen & McCallum

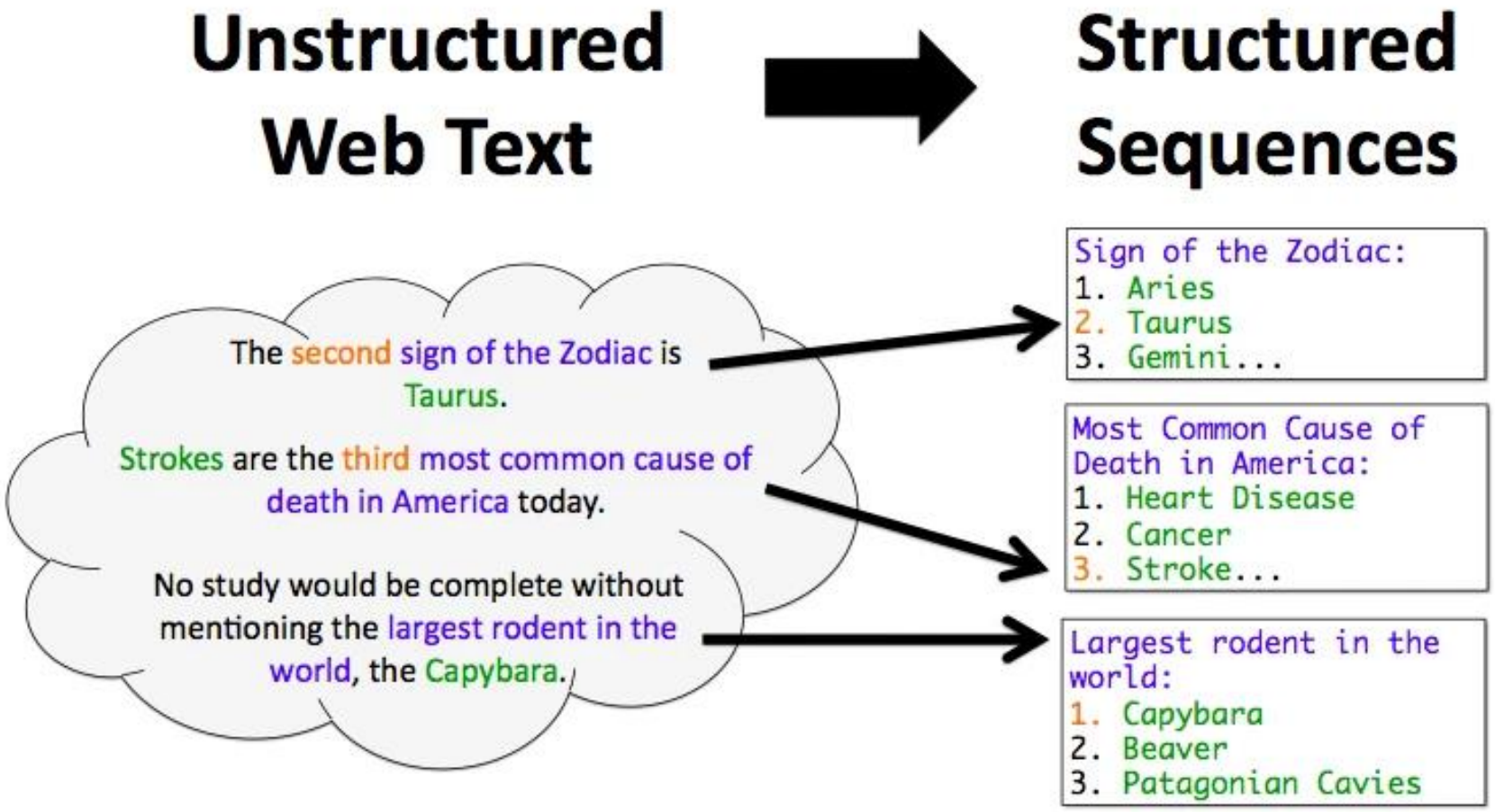


3. *IE pipeline illustrating NLP processing needed for some IE tasks*

https://www.slideserve.com/london/natural-language-processing-information-extraction-cse-573-dan-weld

NLP Applications

❖ Information Extraction



NLP Applications

❖ Information Extraction

Named Entity Recognition

In the 19th century, there was something called the "cult of domesticity" for many American women. This meant that most married women were expected to stay in the home and raise children. As in other countries, American wives were very much under the control of their husband, and had almost no rights. Women who were not married had only a few jobs open to them, such as working in clothing factories and serving as maids. By the 19th century, women such as Lucretia Mott and Elizabeth Cady Stanton thought that women should have more rights. In 1848, many of these women met and agreed to fight for more rights for women, including voting. Many of the women involved in the movement for women's rights were also involved in the movement to end slavery.

WIKIPEDIA

Tag colors:

LOCATION PERSON TERM DATE CONDITION PROCESS PEOPLE

https://mobidev.biz/blog/natural-language-processing-nlp-use-cases-business

NLP Applications

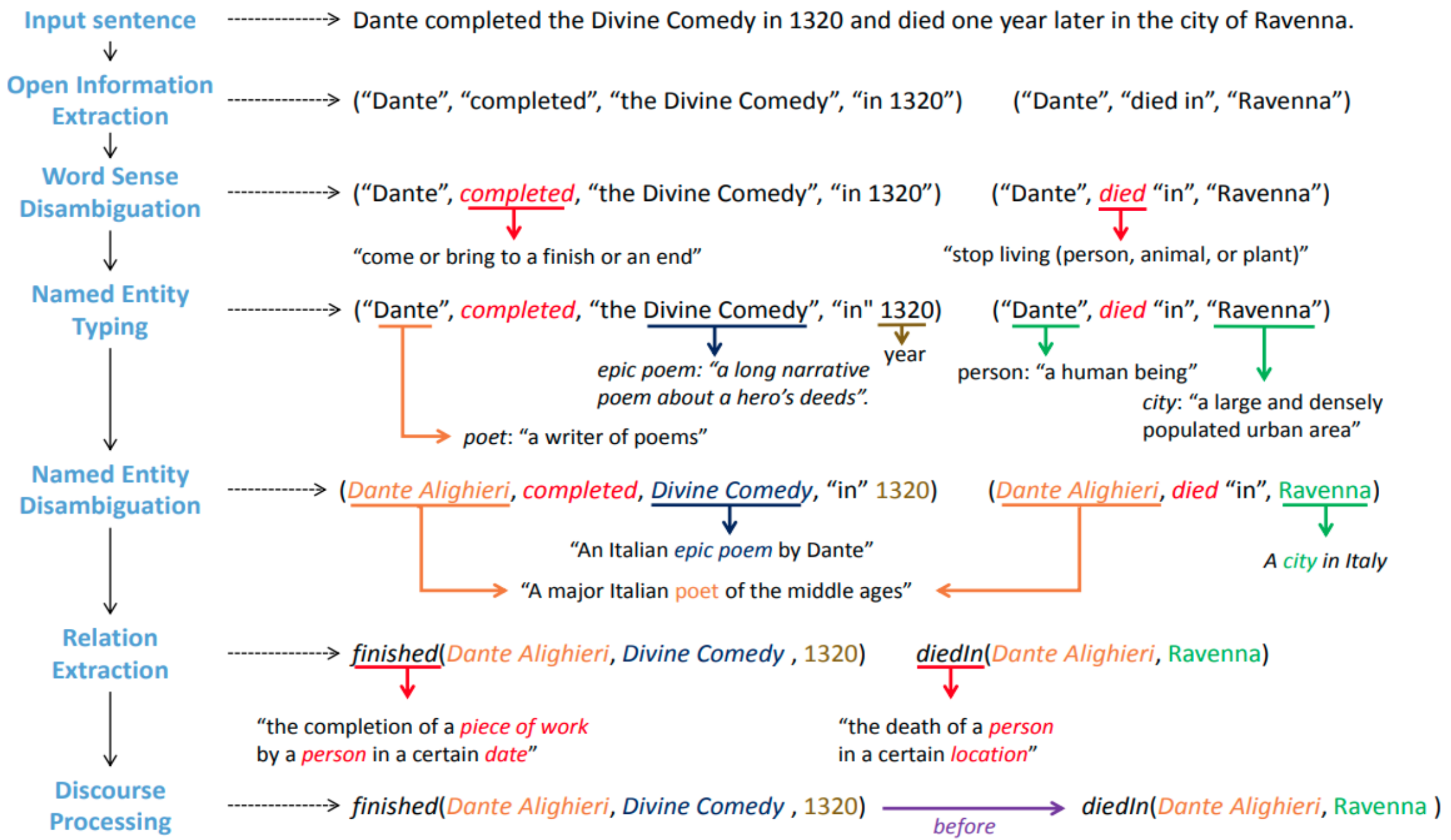
❖ Information Extraction

Part-Of-Speech Tagging

Larry went to the office by bus.

Larry - PROPN
went - VERB
to - PREP
the - ART
office - NOUN
by - PREP
bus - NOUN
. - PUNCT

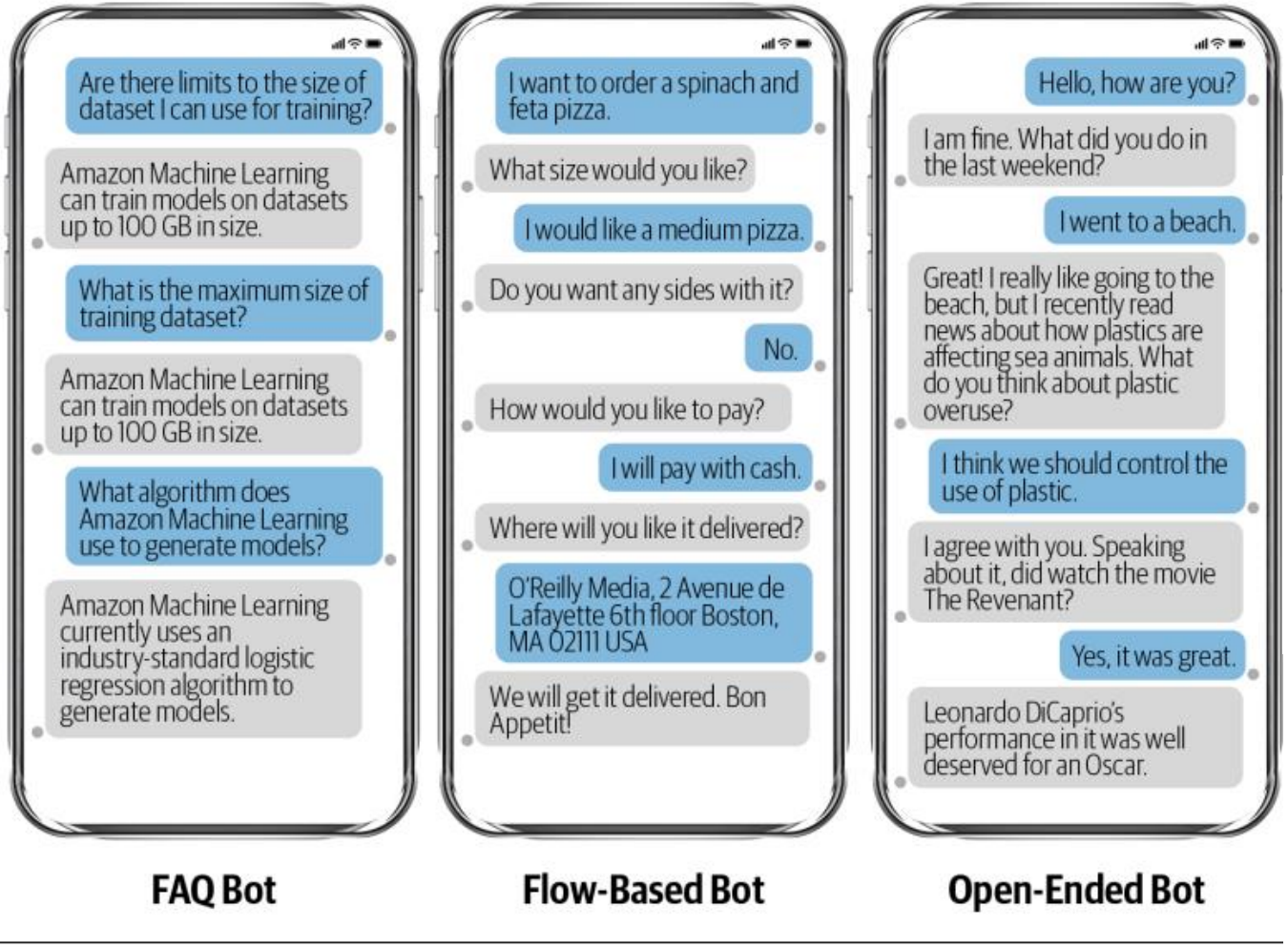
https://mobidev.biz/blog/natural-language-processing-nlp-use-cases-business



Corro, L.D. (2015). Methods for open information extraction and sense disambiguation on natural language text.

NLP Applications

❖ Conversational Agents (Chatbots)



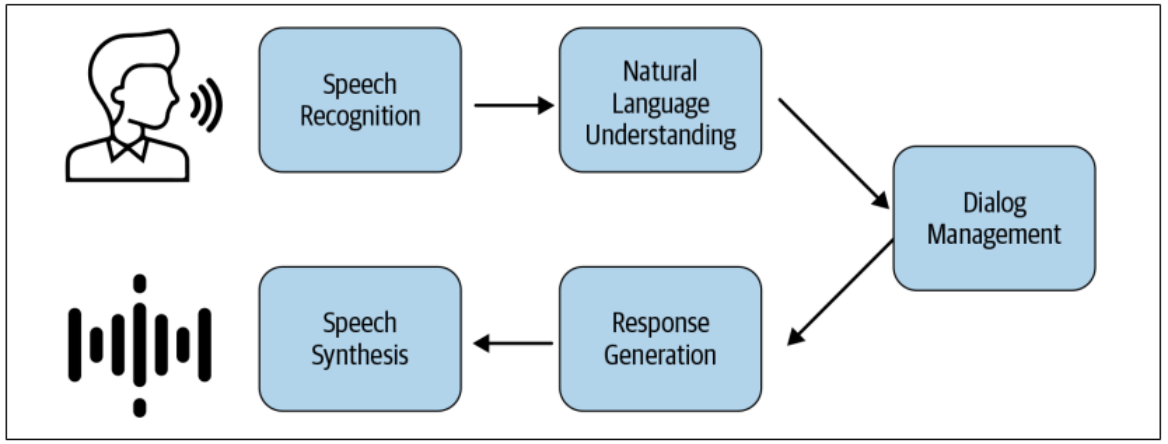


Figure 1-19. Flow of conversation agents

Practical Natural Language Processing by Sowmya Vajjala, Bodhisattwa Majumder, Anuj Gupta, Harshit Surana

NLP Applications

❖ Conversational Agents (Chatbots)

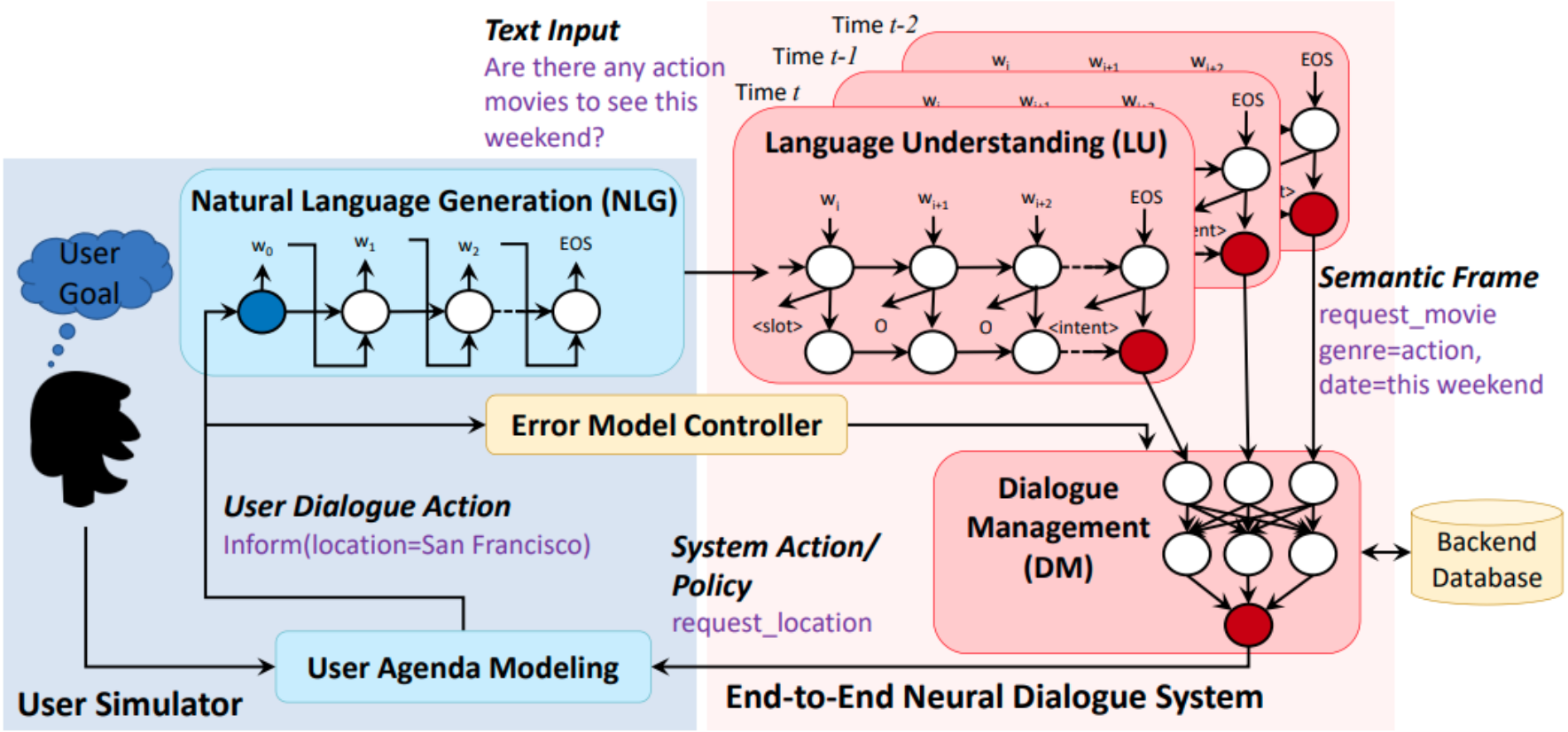


Figure 1: Illustration of the end-to-end neural dialogue system: given user utterances, reinforcement learning is used to train all components in an end-to-end fashion.

Li, X., Chen, Y.(., Li, L., Gao, J., & Celikyilmaz, A. (2017). End-to-End Task-Completion Neural Dialogue Systems. IJCNLP.

NLP Applications

❖ Conversational Agents (Chatbots)

Intent: orderPizza

"get me a medium pizza with extra cheese"

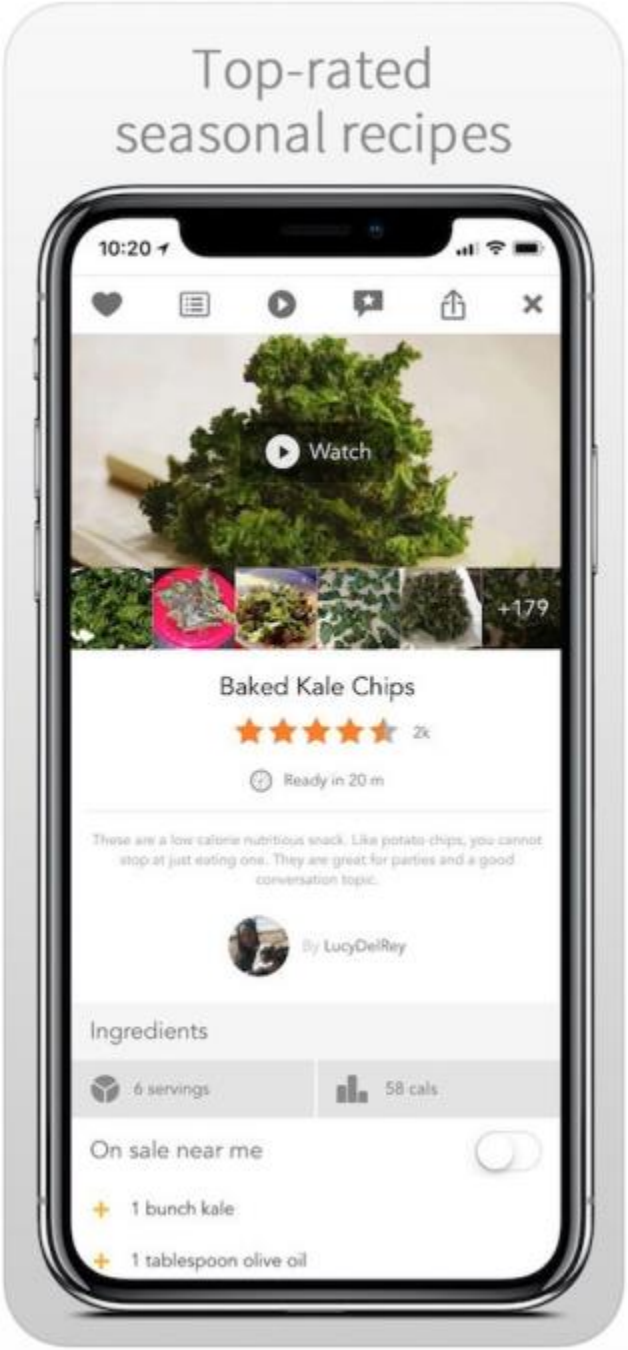
Entity Entity

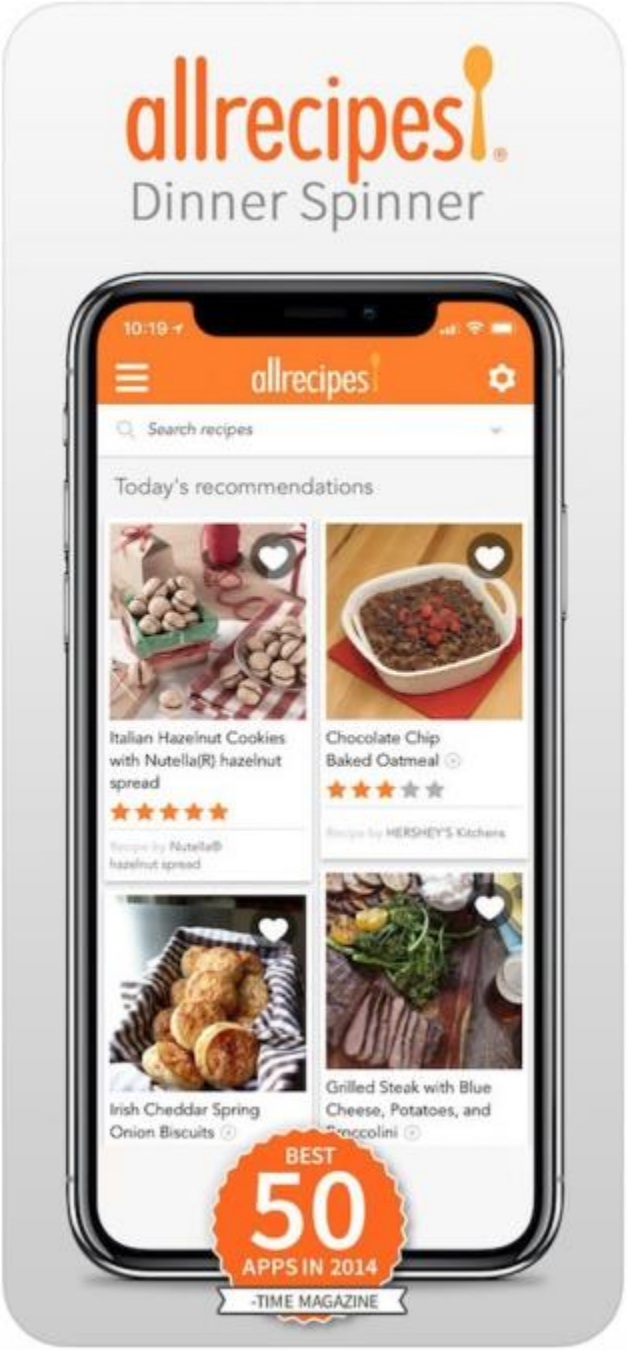
Intent: getStockQuote

"how is dow jones doing today?"

Entity Entity

Dialog Systems in Detail





Recipe Recommendations

Practical Natural Language Processing by Sowmya Vajjala, Bodhisattwa Majumder, Anuj Gupta, Harshit Surana

NLP Applications

❖ Topics in Brief

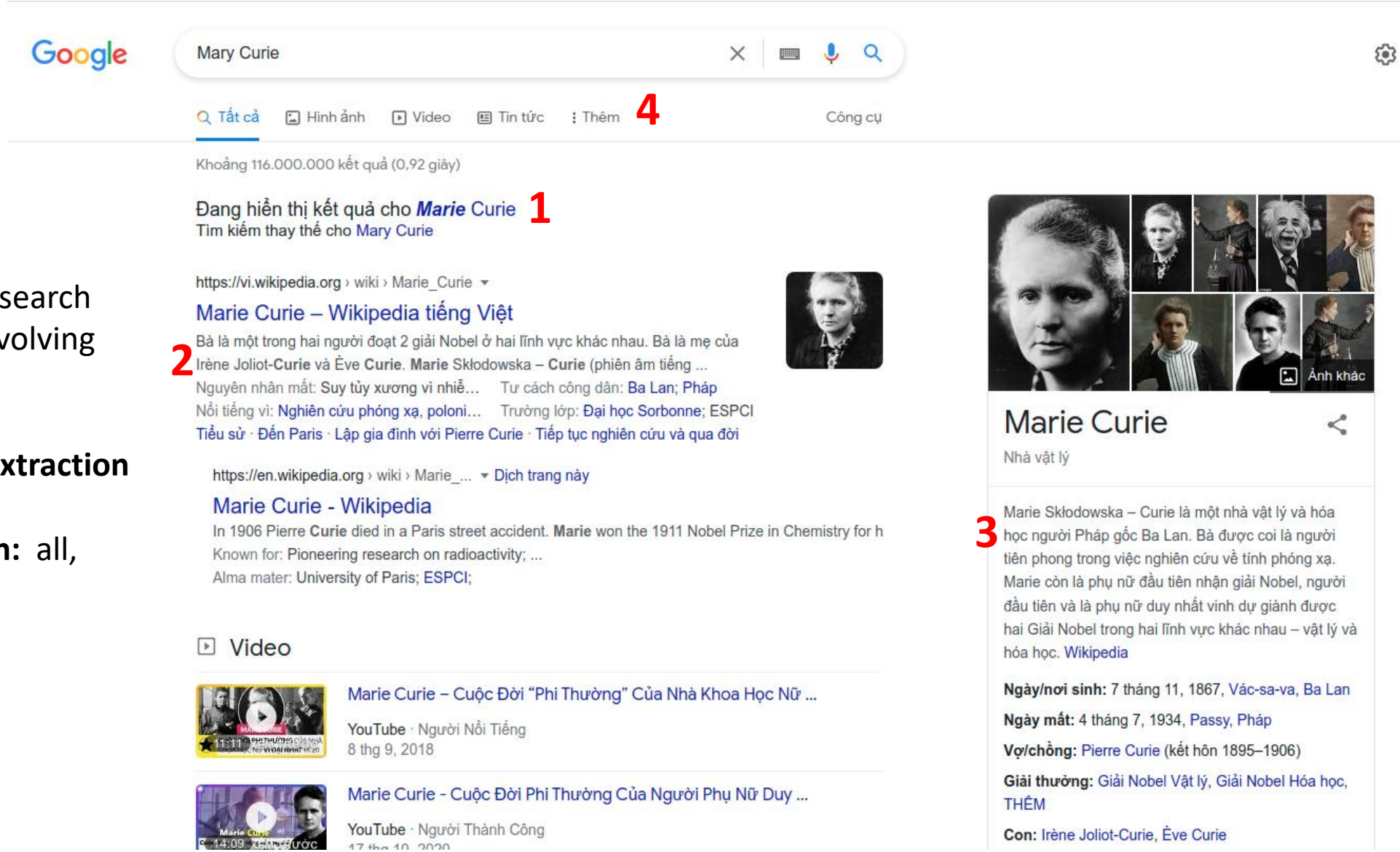
Search and Information Retrieval

1. **Spelling correction**

2. **Snippet extraction:** All the search results show a text snippet involving the query.

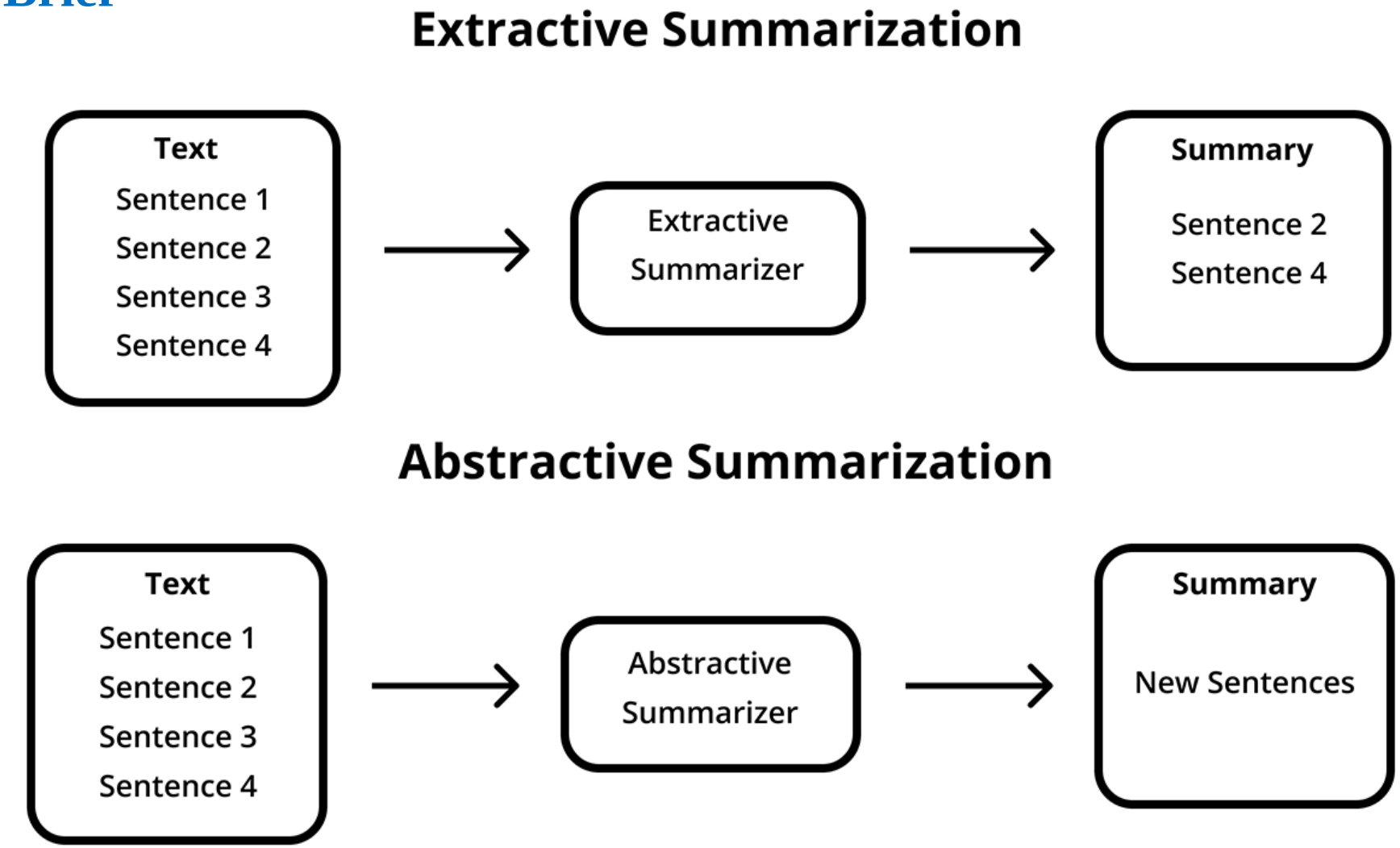
3. **Biographical information extraction**

4. **Search results classification:** all, news, images, videos



NLP Applications

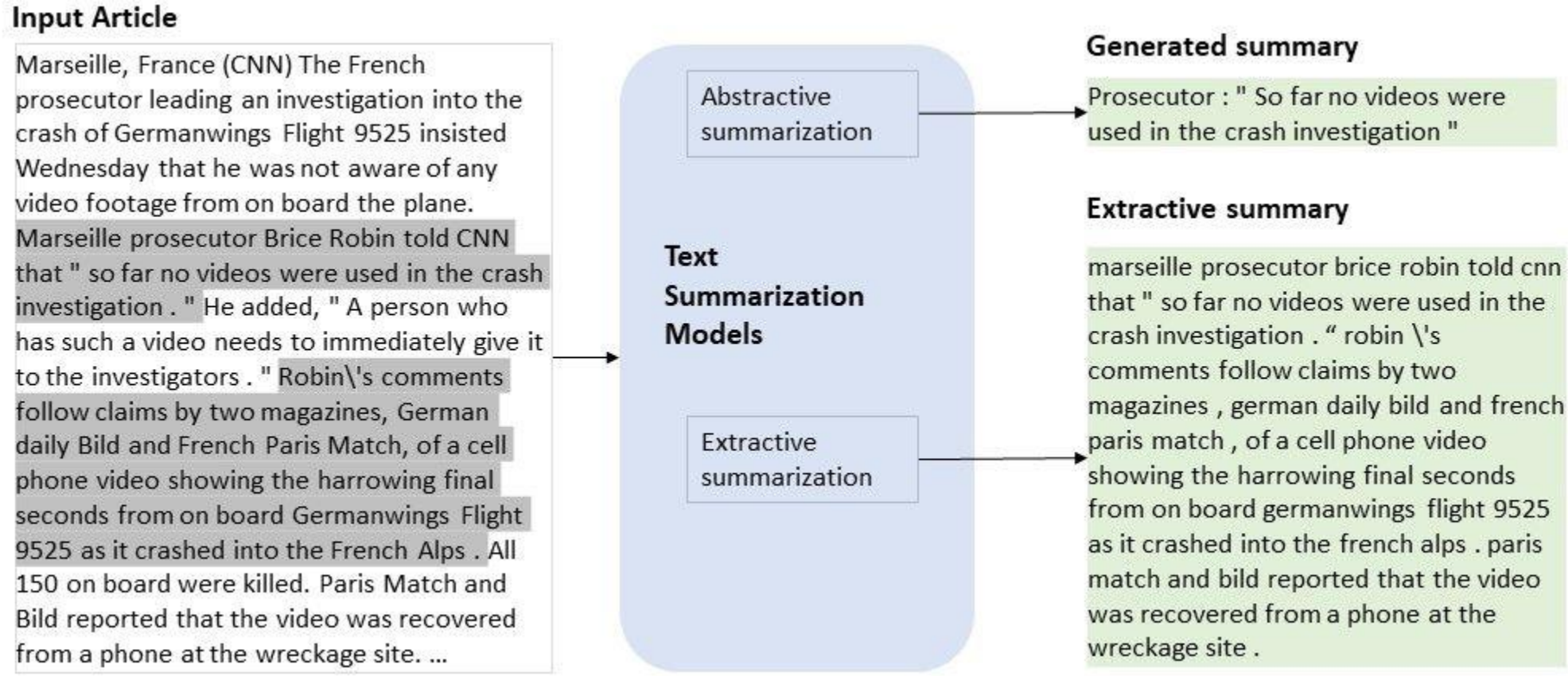
❖ Topics in Brief



https://mobidev.biz/blog/natural-language-processing-nlp-use-cases-business

NLP Applications

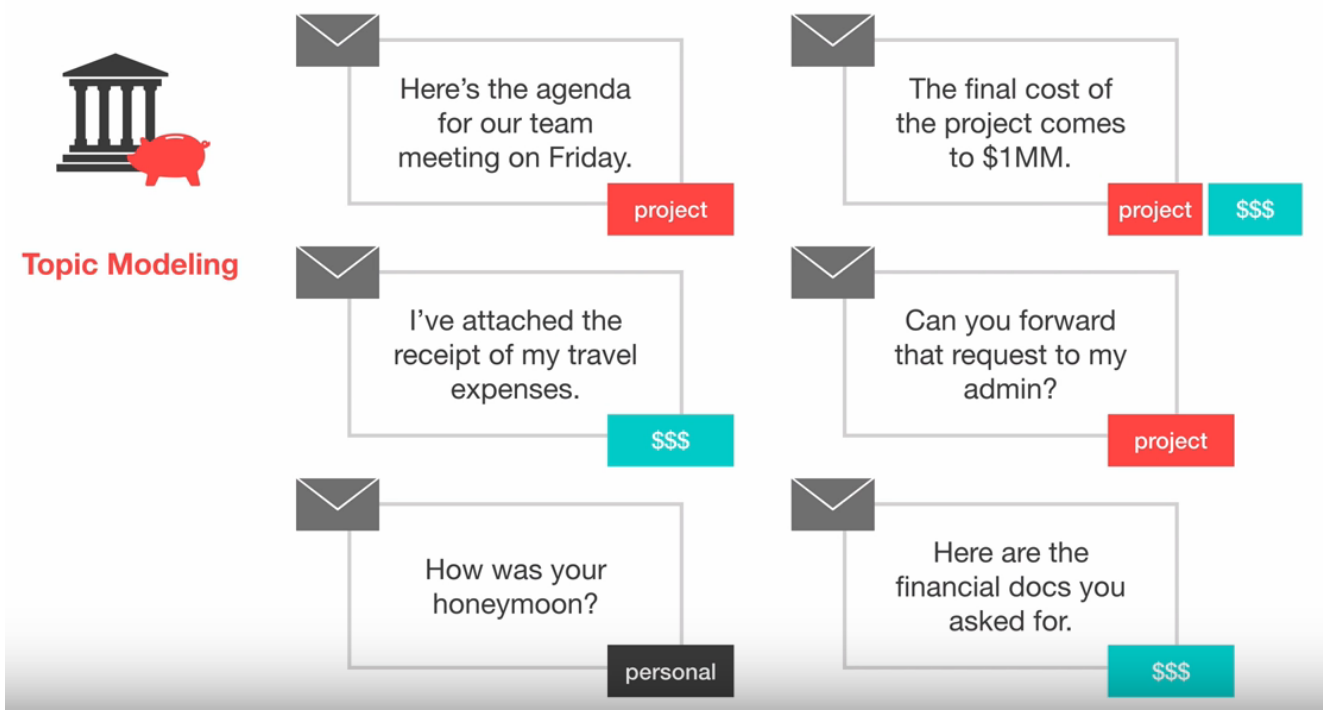
❖ Topics in Brief

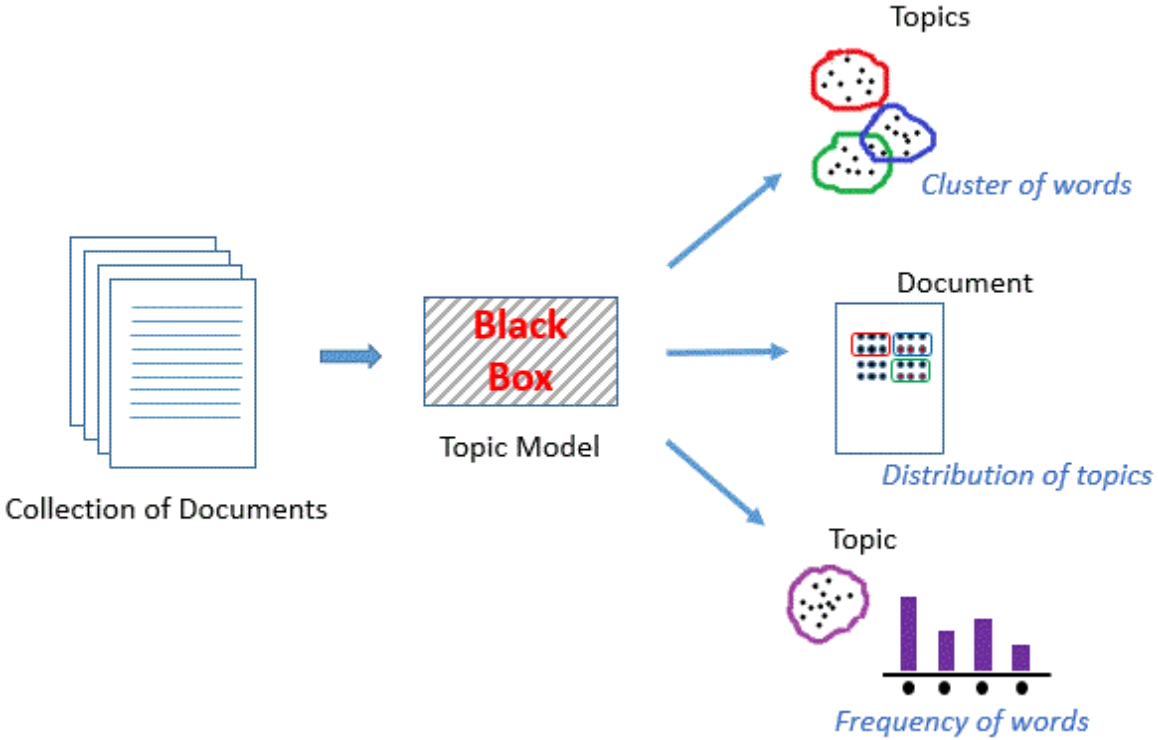


https://techcommunity.microsoft.com/t5/ai-customer-engineering-team/bootstrap-your-text-summarization-solution-with-the-latest/ba-p/1268809

NLP Applications

❖ Topics in Brief



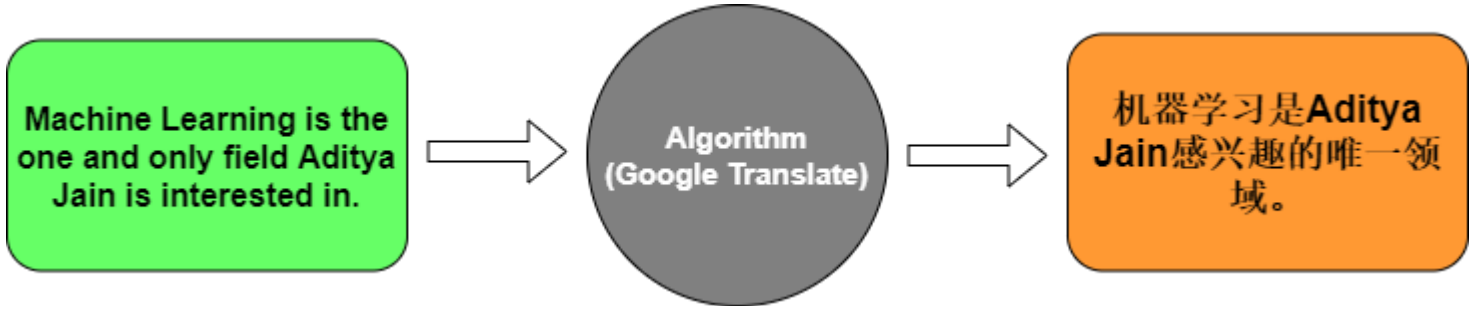


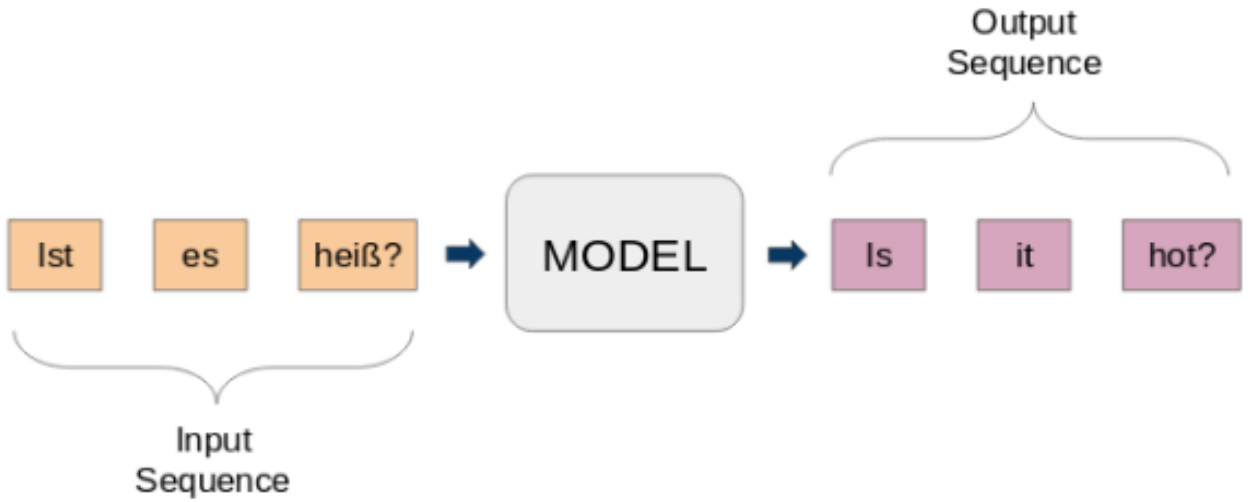
https://devopedia.org/topic-modelling

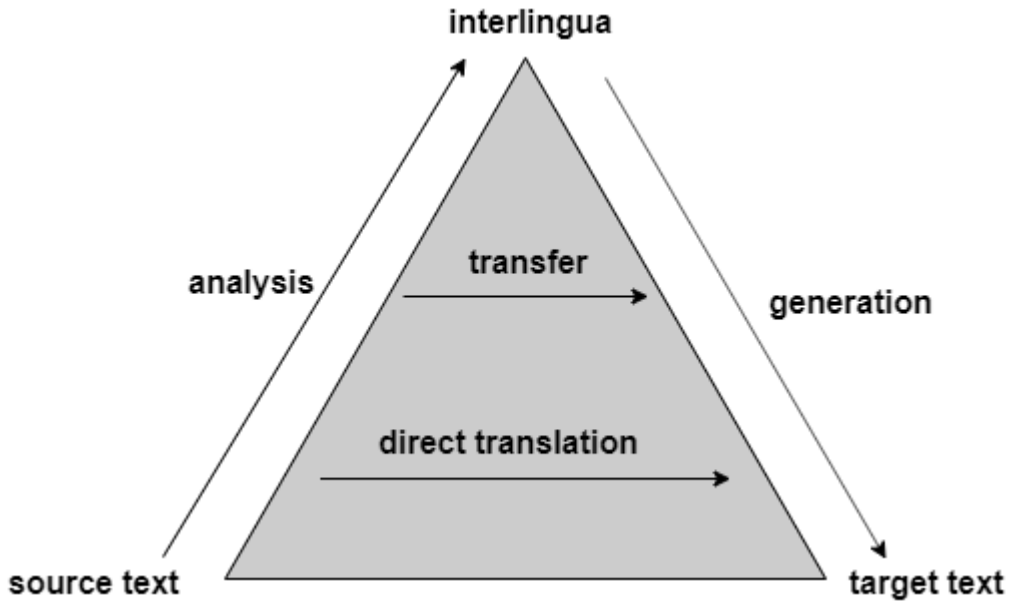
NLP Applications

❖ Topics in Brief

Machine Translation







https://www.studytonight.com/post/different-machine-translation-models-in-nlp
https://www.analyticsvidhya.com/blog/2019/01/neural-machine-translation-keras/