AI School 6기 9주차

파이썬 알고리즘

RNN 기초

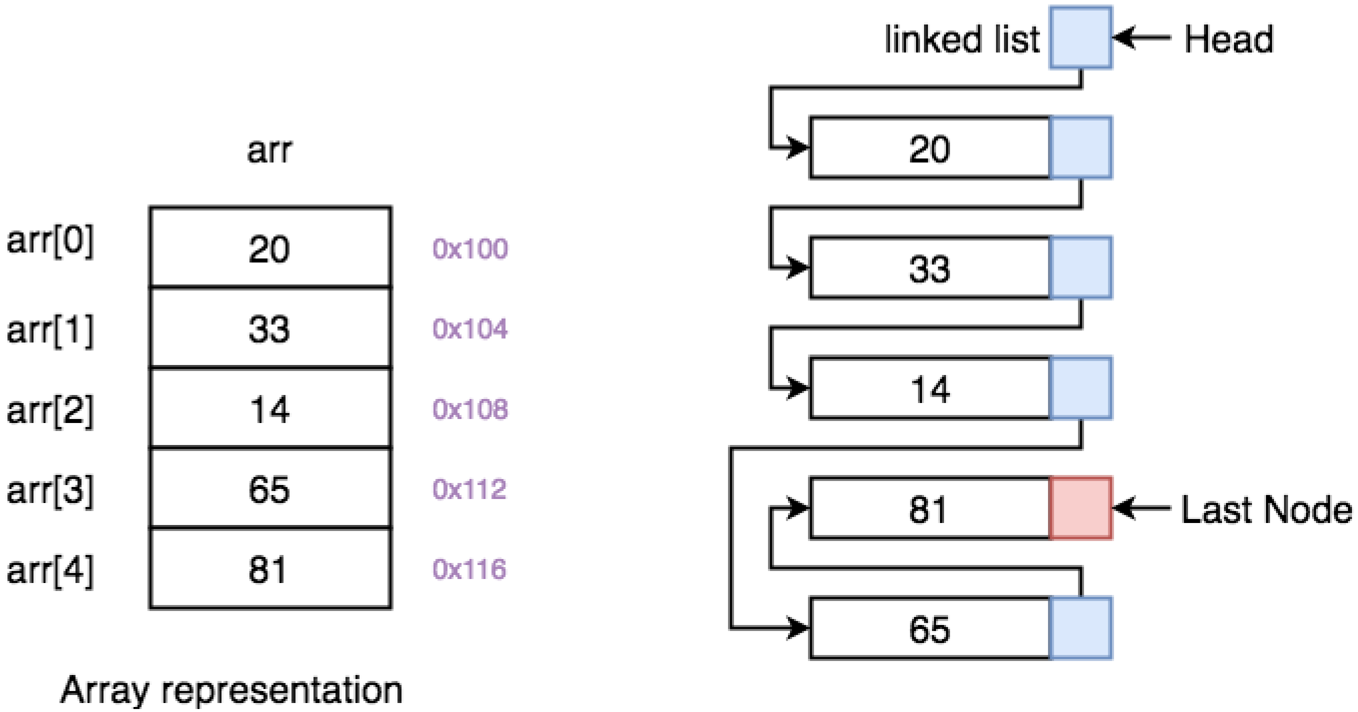
RNN 기반 텍스트 분류기

AI School 6기 9주차

파이썬 알고리즘

자료구조 구현

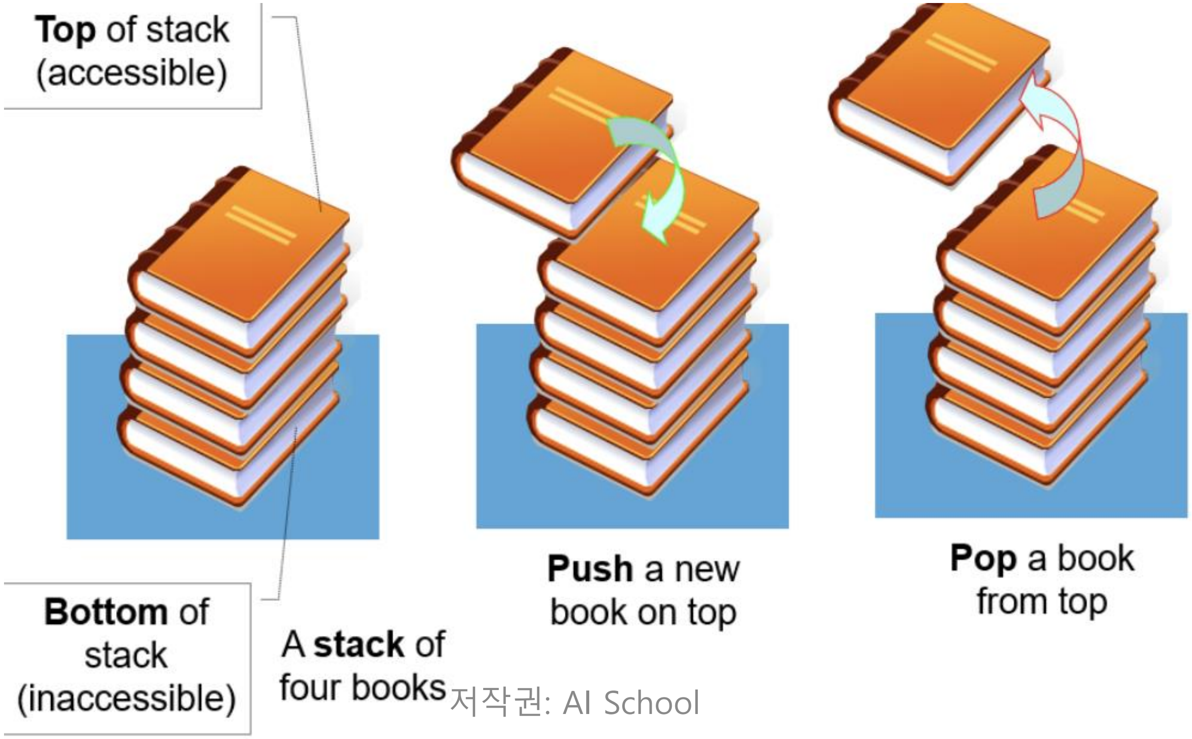
- 자료구조는 크게 배열 기반의 연속 (continuation) 방식과 포인터 기반의 연결 (link) 방식으로 분류



저작권: AI School

스택 (Stack)

- 스택은 배열의 끝에서만 데이터를 접근할 수 있는 선형 자료구조
- 후입선출 (Last In, First Out: LIFO)
- push: 스택 맨 끝(맨 위)에 항목을 삽입
- pop: 스택 맨 끝 항목을 반환하는 동시에 제거
- top/peek: 스택 맨 끝 항목을 조회
- empty: 스택이 비어 있는지 확인
- size: 스택 크기를 확인



배열기반 스택

stack.py

```
class Stack(object):
    def __init__(self):
        self.items = [ ]

    def isEmpty(self):
        return not bool(self.items)

    def push(self, value):
        self.items.append(value)

    def size(self):
        return len(self.items)

    def __repr__(self):
        return '{ }'.format(self.items)
```

배열기반 스택

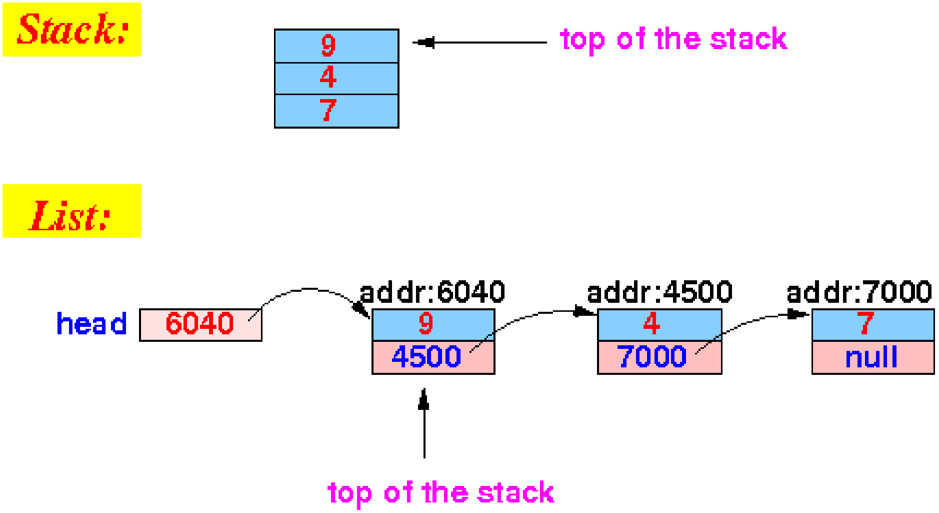
stack.py

```
    def peek(self):
        if self.items:
            return self.items[-1]
        else:
            print('Stack is empty.')

    def pop(self):
        value = self.items.pop()
        if value is not None:
            return value
        else:
            return 'Stack is empty'
if __name__ == '__main__':
    stack = Stack()
    print(stack.isEmpty())
    ...
    print(stack)
```

포인터기반 스택

linked_stack.py

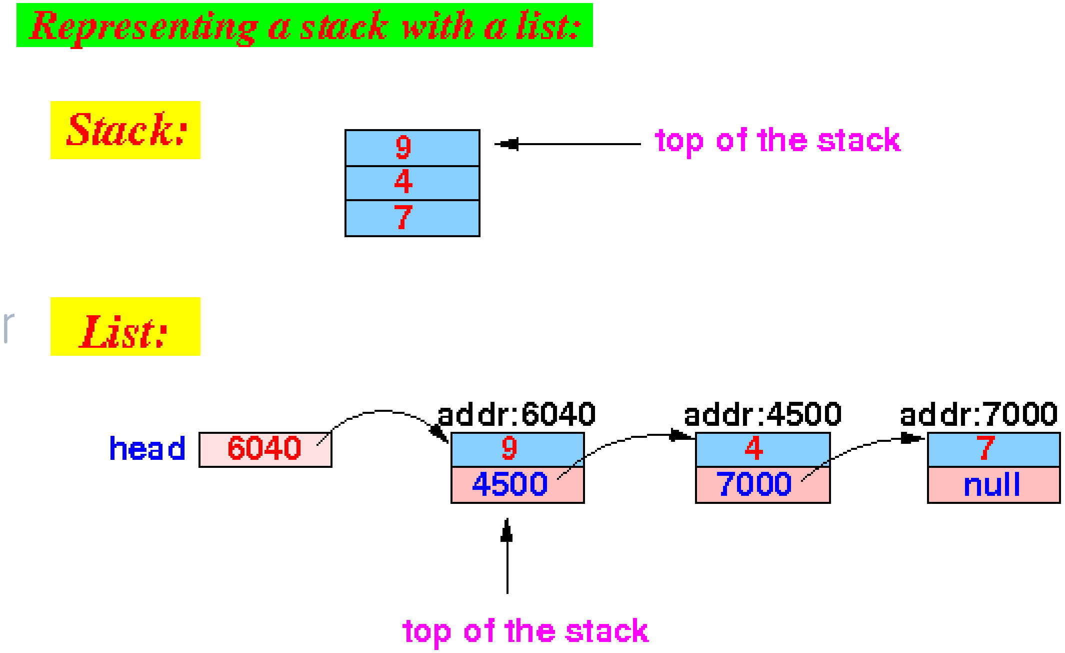
```python
class Node(object):
    def __init__(self, value=None, pointer=None):
        self.value = value
        self.pointer = pointer

class Stack(object):
    def __init__(self):
        self.head = None

    def isEmpty(self):
        return not bool(self.head)

    def push(self, item):
        self.head = Node(item, self.head)
```



포인터기반 스택

linked_stack.py

```
def size(self):
    node = self.head
    count = 0
    while node:
        count +=1
        node = node.pointer
    return count

def pop(self):
    if self.head:
        node = self.head
        self.head = node.pointer
        return node.value
    else:
        print('Stack is empty.')
```



Representing a stack with a list:

Stack:

9
4
7

top of the stack

List:

head 6040

addr:6040
9
4500

addr:4500
4
7000

addr:7000
7
null

top of the stack

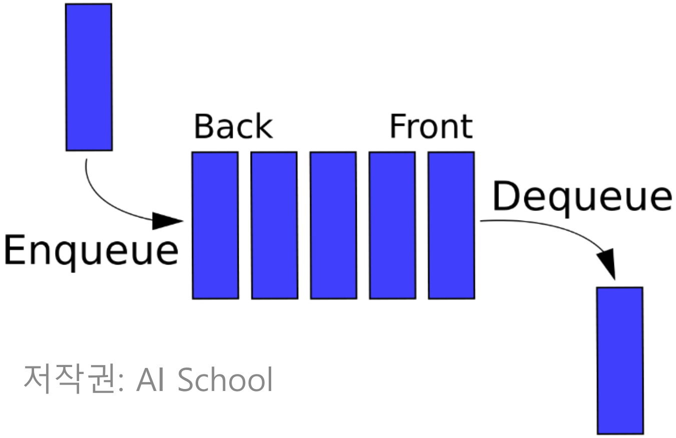
포인터기반 스택

linked_stack.py

```
    def peek(self):
        if self.head:
            return self.head.value
        else:
            print('Stack is empty.')

    def __repr__(self):
        items = [ ]
        node = self.head
        while node:
            items.append(node.value)
            node = node.pointer
        items.reverse()
        return '{}'.format(items)
if __name__ == '__main__':
    stack = Stack()
    …
```

큐 (Queue)

- 큐는 스택과 달리 항목이 들어온 순서대로 접근 가능
- 선입선출 (First In, First Out: FIFO)
- enqueue: 큐 뒤쪽에 항목을 삽입
- dequeue: 큐 앞쪽의 항목을 반환하고, 제거
- peek/front: 큐 앞쪽의 항목을 조회
- empty: 큐가 비어 있는지 확인
- size: 큐의 크기를 확인



배열기반 큐

queue.py

```python
class Queue(object):
    def __init__(self):
        self.items = []

    def isEmpty(self):
        return not bool(self.items)

    def enqueue(self, value):
        self.items.insert(0, value)

    def size(self):
        return len(self.items)

    def __repr__(self):
        return '{}'.format(self.items)
```

배열기반 큐

queue.py

```
    def peek(self):
        if self.items:
            return self.items[-1]
        else:
            print('Queue is empty.')

    def dequeue(self):
        value = self.items.pop()
        if value is not None:
            return value
        else:
            return 'Queue is empty'
if __name__ == '__main__':
    queue = Queue()
    print(queue.isEmpty())
    queue.enqueue(23)
...
```

배열기반 큐 (두개의 스택을 이용한 큐)

two_stacks_queue.py

```
class Queue(object):
    def __init__(self):
        self.in_stack = [ ]
        self.out_stack = [ ]

    def isEmpty(self):
        return not (bool(self.in_stack) or bool(self.out_stack))

    def _transfer(self):
        while self.in_stack:
            self.out_stack.append(self.in_stack.pop())

    def enqueue(self, item):
        return self.in_stack.append(item)

    def size(self):
        return len(self.in_stack) + len(self.out_stack)
```

배열기반 큐 (두개의 스택을 이용한 큐)

two_stacks_queue.py

```python
def peek(self):
    if not self.out_stack:
        self._transfer()
    if self.out_stack:
        return self.out_stack[-1]
    else:
        return "Queue empty!"

def __repr__(self):
    if not self.out_stack:
        self._transfer()
    if self.out_stack:
        return '{}'.format(self.out_stack)
    else:
        return "Queue is empty"
```

배열기반 큐 (두개의 스택을 이용한 큐)

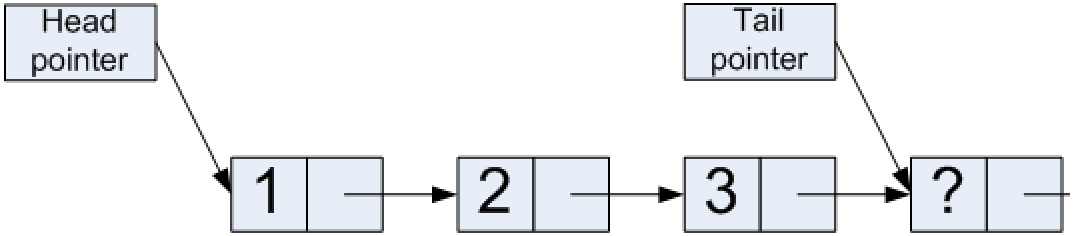
two_stacks_queue.py

```
    def dequeue(self):
        if not self.out_stack:
            self._transfer()
        if self.out_stack:
            return self.out_stack.pop()
        else:
            return "Queue is empty"
if __name__ == '__main__':
    queue = Queue()
    print(queue.isEmpty())
    queue.enqueue(23)
    queue.enqueue(4)
    queue.enqueue(8)
    print("Size: ", queue.size())
    print(queue)
    print("Peek: ", queue.peek())
    print("Dequeue!  ", queue.dequeue())
    print(queue)
```

포인터기반 큐

linked_queue.py

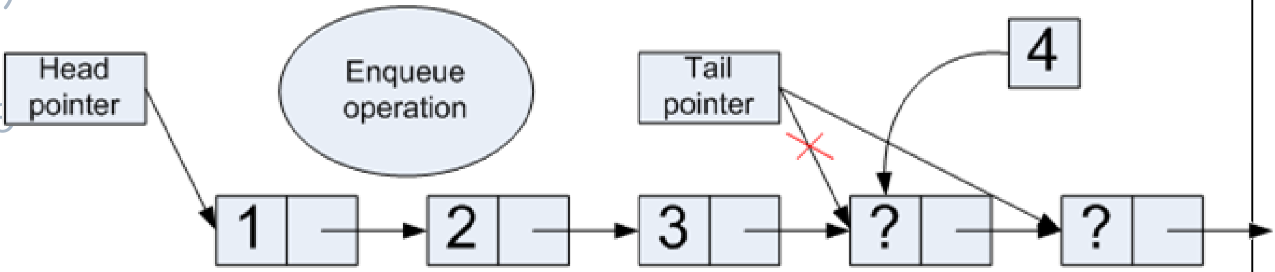
```
class Node(object):
    def __init__(self, value=None, pointer=None):
        self.value = value
        self.pointer = None

class Queue(object):
    def __init__(self):
        self.head = None
        self.tail = None

    def isEmpty(self):
        return not bool(self.head)
```



포인터기반 큐

linked_queue.py

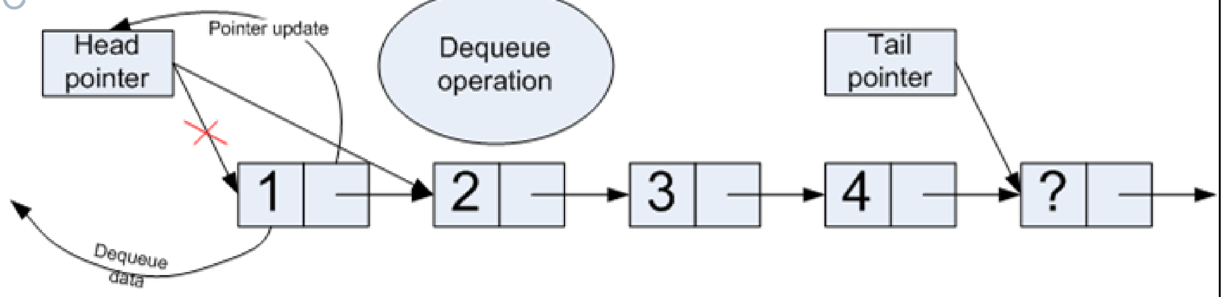
```
def enqueue(self, value):
    node = Node(value)
    if not self.head:
        self.head = node
        self.tail = node
    else:
        if self.tail:
            self.tail.pointer = node
        self.tail = node

def size(self):
    node = self.head
    count = 0
    while node:
            count += 1
            node = node.pointer
    return count
```



포인터기반 큐

linked_queue.py

```
    def peek(self):
        return self.head.value

    def __repr__(self):
        items = []
        node = self.head
        while node:
            items.append(node.value)
            node = node.pointer
        items.reverse()
        return '{}'.format(items)
    def dequeue(self):
        if self.head:
            value = self.head.value
            self.head = self.head.pointer
            return value
        else:
            print('Queue is empty')
```



포인터기반 큐

linked_queue.py

```python
if __name__ == '__main__':
    queue = Queue()
    print(queue.isEmpty())
    queue.enqueue(23)
    queue.enqueue(4)
    queue.enqueue(8)
    print("Size: ", queue.size())
    print(queue)
    print("Peek: ", queue.peek())
    print("Dequeue!  ", queue.dequeue())
    print(queue)
```

Homework

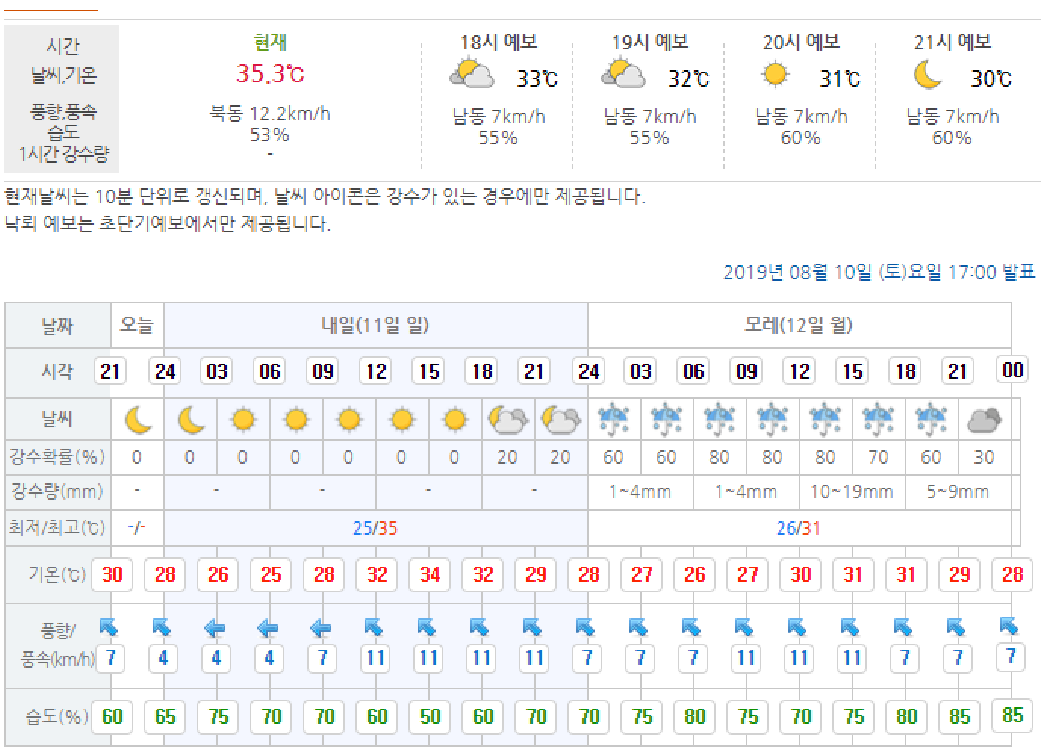
- 스택을 활용해서 문자열을 반전해보세요.

```
def reverse_with_stack(input)
    s = Stack()
    …

print(reverse_with_stack(“AI School”))
→loohcS IA
```

AI School 6기 9주차

RNN 기초

Sequence data



시간 날씨,기온 풍향,풍속 습도 1시간 강수량	현재	18시 예보	19시 예보	20시 예보	21시 예보
	35.3℃	⛅ 33℃	⛅ 32℃	☀ 31℃	☾ 30℃
	북동 12.2km/h 53% -	남동 7km/h 55%	남동 7km/h 55%	남동 7km/h 60%	남동 7km/h 60%

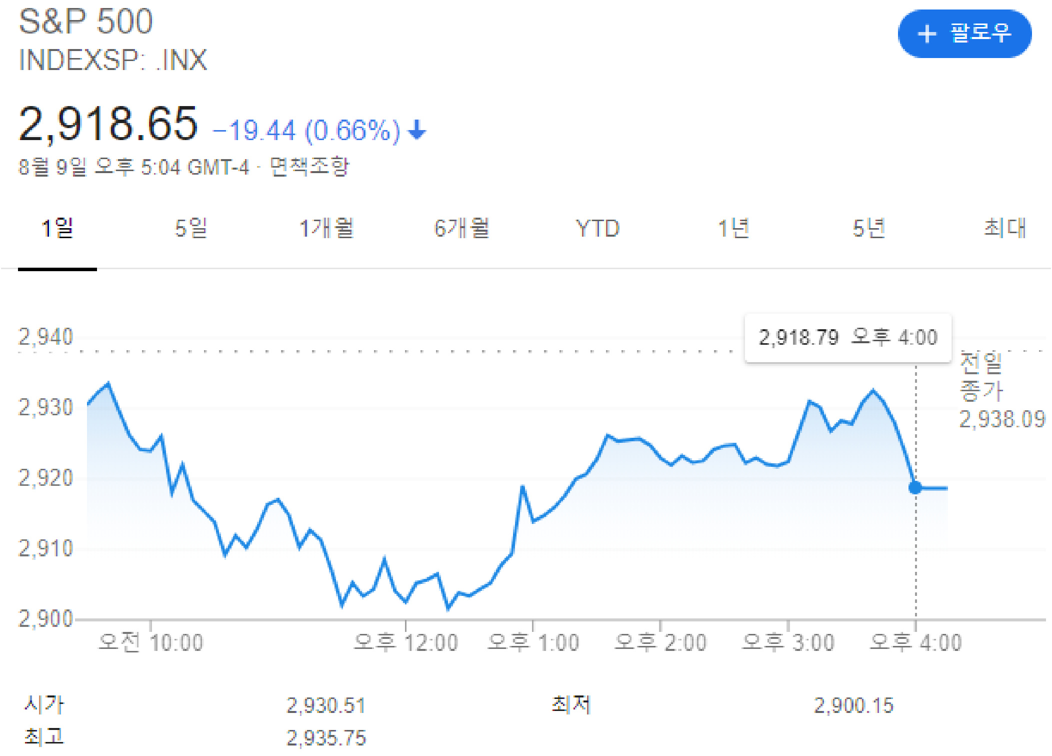
현재날씨는 10분 단위로 갱신되며, 날씨 아이콘은 강수가 있는 경우에만 제공됩니다.
낙뢰 예보는 초단기예보에서만 제공됩니다.

2019년 08월 10일 (토)요일 17:00 발표

날짜	오늘	내일(11일 일)								모레(12일 월)								
시각	21	24	03	06	09	12	15	18	21	24	03	06	09	12	15	18	21	00
날씨	☾	☾	☀	☀	☀	☀	☀	⛅	⛅	☔	☔	☔	☔	☔	☔	☔	☁	
강수확률(%)	0	0	0	0	0	0	0	20	20	60	60	80	80	80	70	60	30	
강수량(mm)	-	-		-		-		-		1~4mm		1~4mm		10~19mm		5~9mm		
최저/최고(℃)	-/-	25/35								26/31								
기온(℃)	30	28	26	25	28	32	34	32	29	28	27	26	27	30	31	31	29	28
풍향/ 풍속(km/h)	↖ 7	↖ 4	← 4	← 4	← 7	↖ 11	↖ 11	↖ 11	↖ 11	↖ 7	↖ 7	↖ 7	↖ 11	↖ 11	↖ 11	↖ 7	↖ 7	↖ 7
습도(%)	60	65	75	70	70	60	50	60	70	70	75	80	75	70	75	80	85	85

일기 예보

Sequence data



S&P 500
INDEXSP: .INX

2,918.65 −19.44 (0.66%)

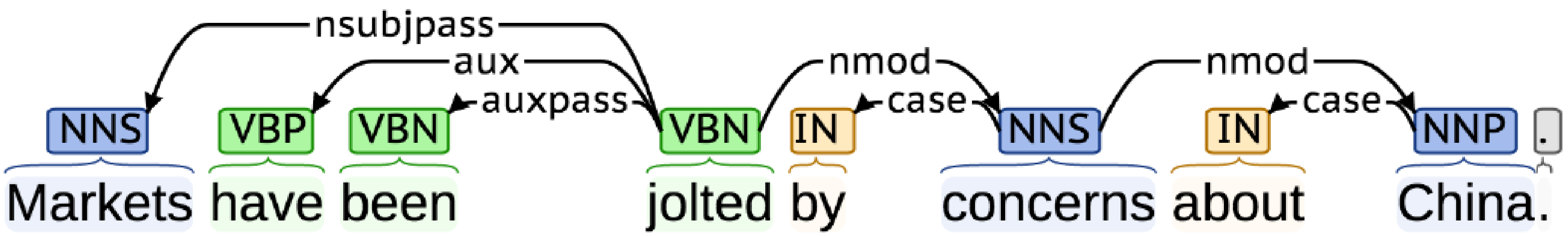
8월 9일 오후 5:04 GMT-4 · 면책조항

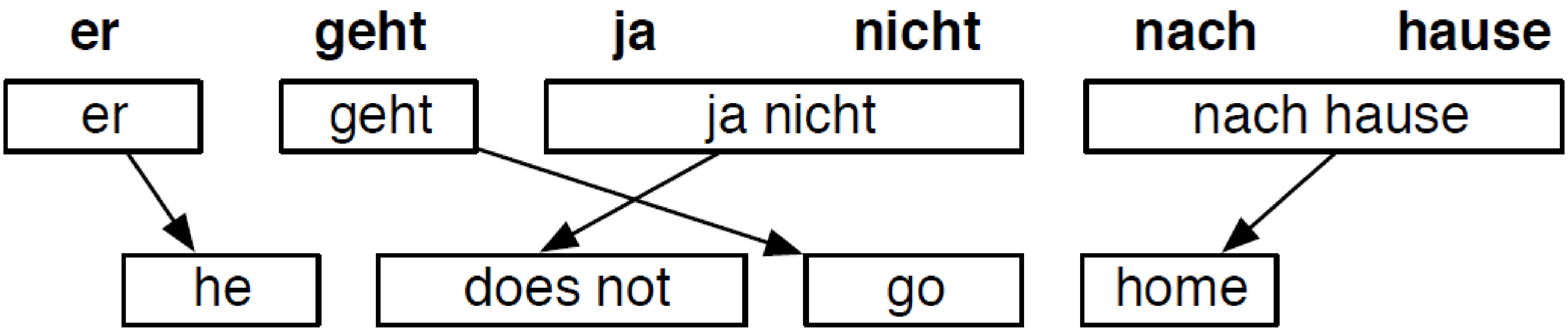
시가	2,930.51	최저	2,900.15
최고	2,935.75		

시장 분석

저작권: AI School

Sequence data



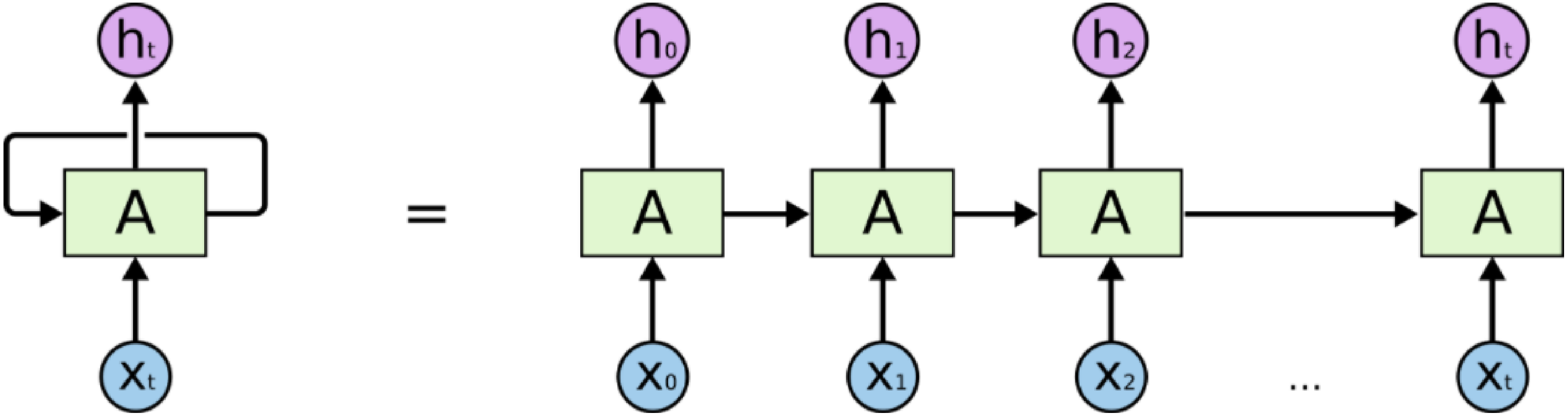


자연어 처리

저작권: AI School

Recurrent neural network (RNN)

- Deep learning 구조 중 하나로, 연속적인 input을 처리하기 위해 고안된 모델



저작권: AI School

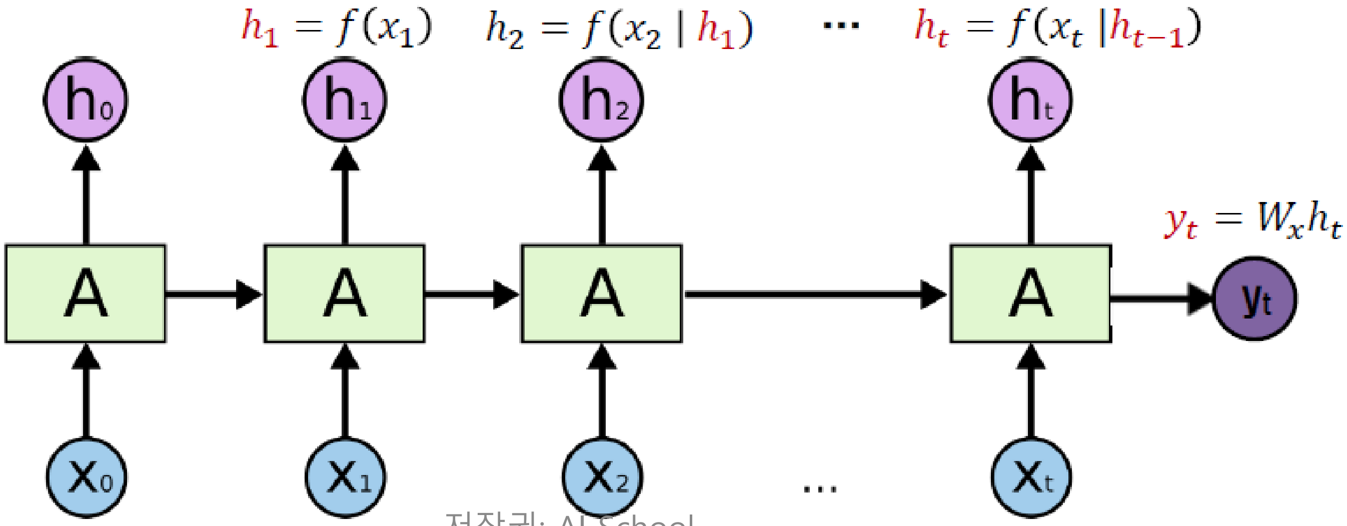
Recurrent neural network (RNN)

- RNN은 추가된 input과 과거의 정보를 조합해, 새로운 정보를 생성
- 최종적으로 생성된 정보를 통해 task를 수행

h_t : Current hidden state

h_{t-1} : Previous hidden state

x_t : Current input



$h_1 = f(x_1)$ $\quad h_2 = f(x_2 \mid h_1)$ $\quad \cdots \quad$ $h_t = f(x_t \mid h_{t-1})$

$y_t = W_x h_t$

Recurrent neural network (RNN)

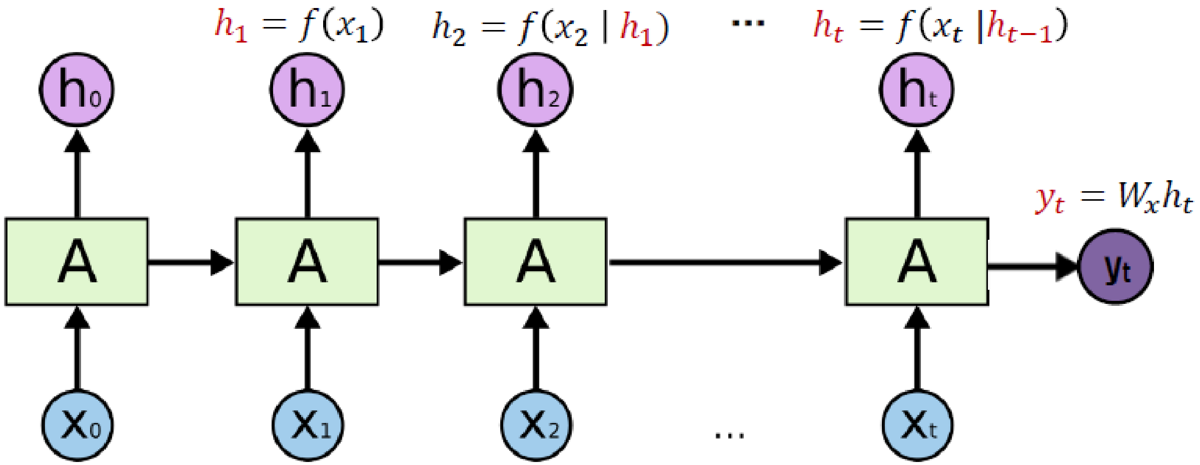
- Input과 이전state을 선형 변환하여 결합한 후, activation 적용

h_t : Current hidden state

h_{t-1} : Previous hidden state

x_t : Current input

$$h_t = f(x_t \mid h_{t-1}) = \tanh(Wx_t + Uh_{t-1})$$

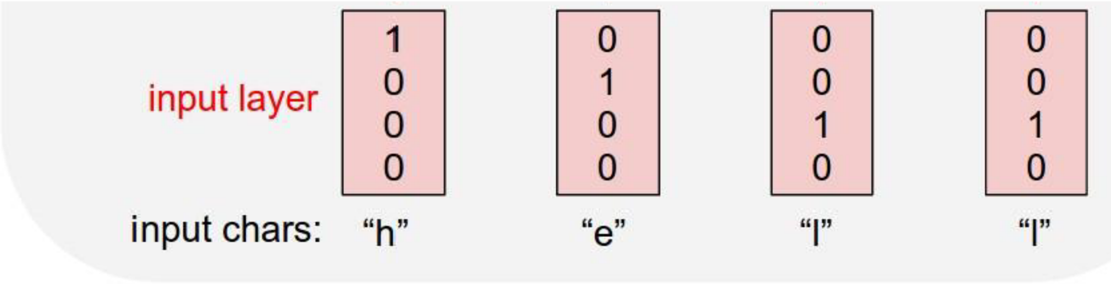


Recurrent neural network (RNN)

Example:
Character-level
Language Model

Vocabulary:
[h,e,l,o]

Example training
sequence:
“hello”



input layer	1	0	0	0
	0	1	0	0
	0	0	1	1
	0	0	0	0
input chars:	“h”	“e”	“l”	“l”

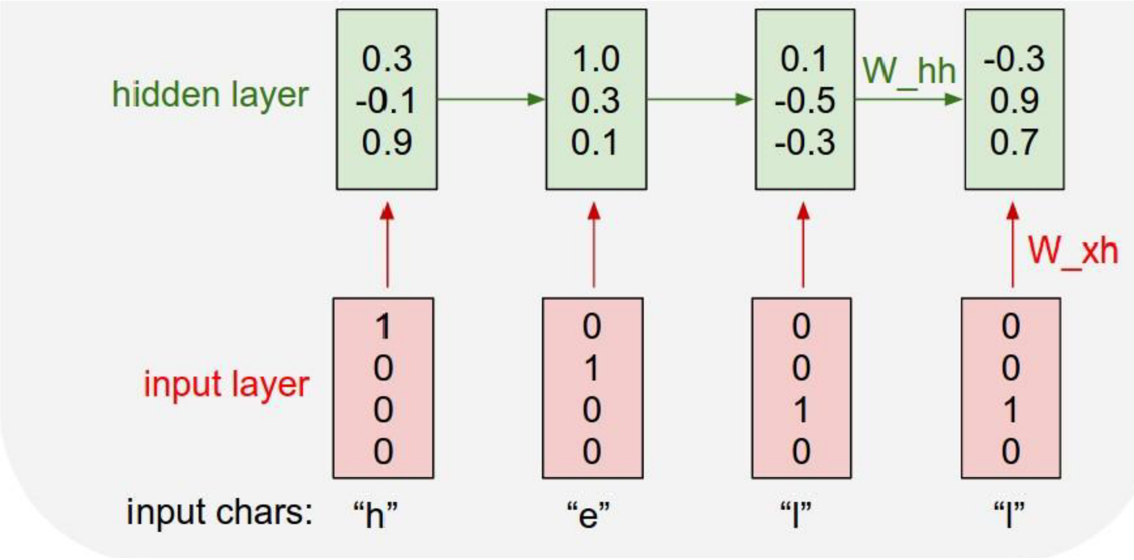
Recurrent neural network (RNN)

**Example:
Character-level
Language Model**

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

Vocabulary:
[h,e,l,o]

Example training
sequence:
"hello"

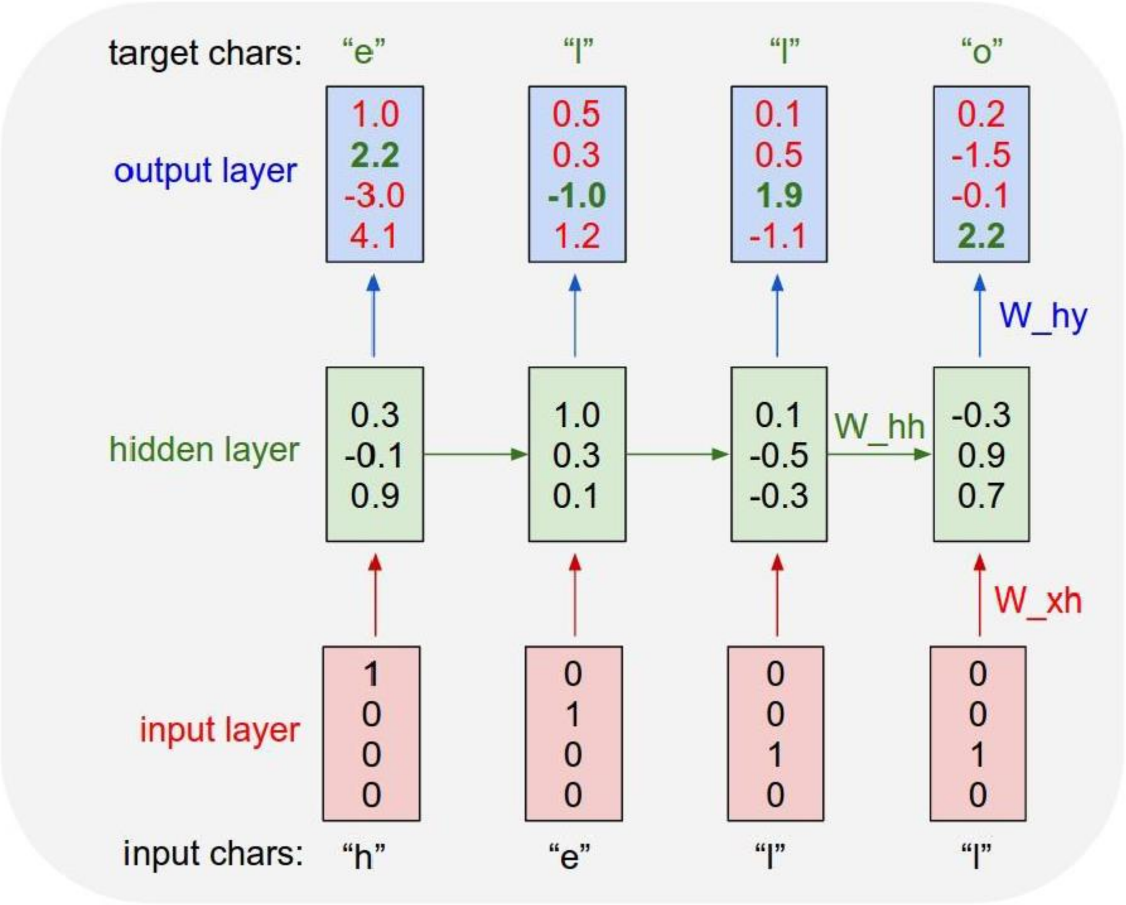


Recurrent neural network (RNN)

Example: Character-level Language Model

Vocabulary: [h,e,l,o]

Example training sequence: **"hello"**

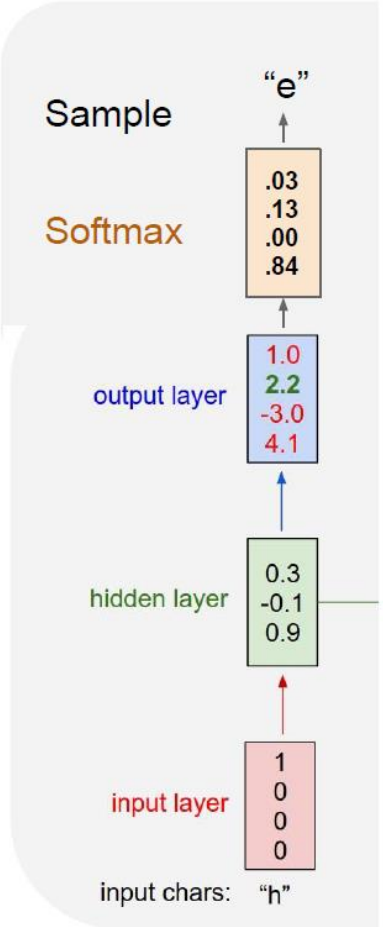


Recurrent neural network (RNN)

Example: Character-level Language Model Sampling

Vocabulary: [h,e,l,o]

At test-time sample characters one at a time, feed back to model

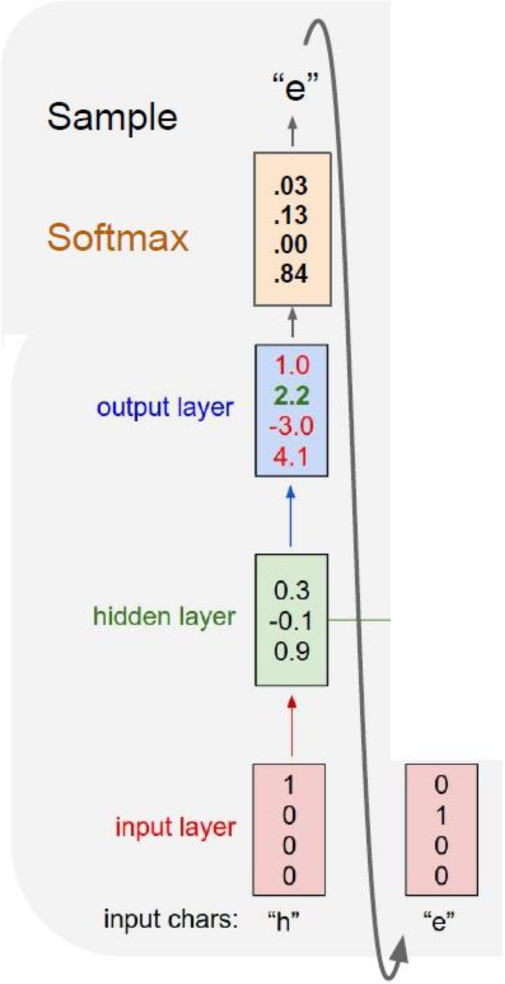


Recurrent neural network (RNN)

Example: Character-level Language Model Sampling

Vocabulary: [h,e,l,o]

At test-time sample characters one at a time, feed back to model

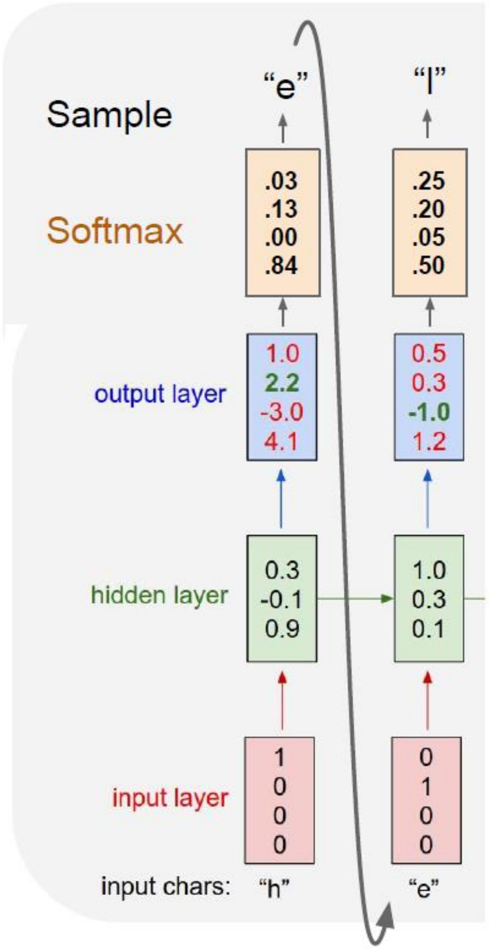


Recurrent neural network (RNN)

Example:
Character-level
Language Model
Sampling

Vocabulary:
[h,e,l,o]

At test-time sample characters one at a time, feed back to model

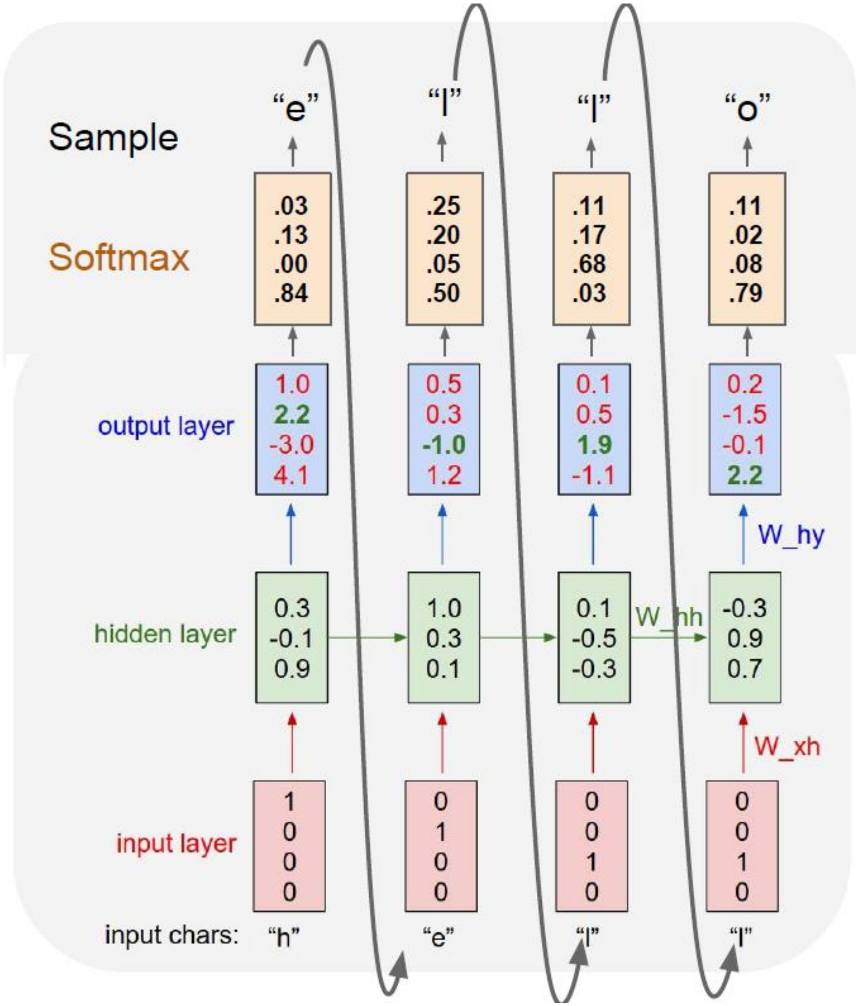


Recurrent neural network (RNN)

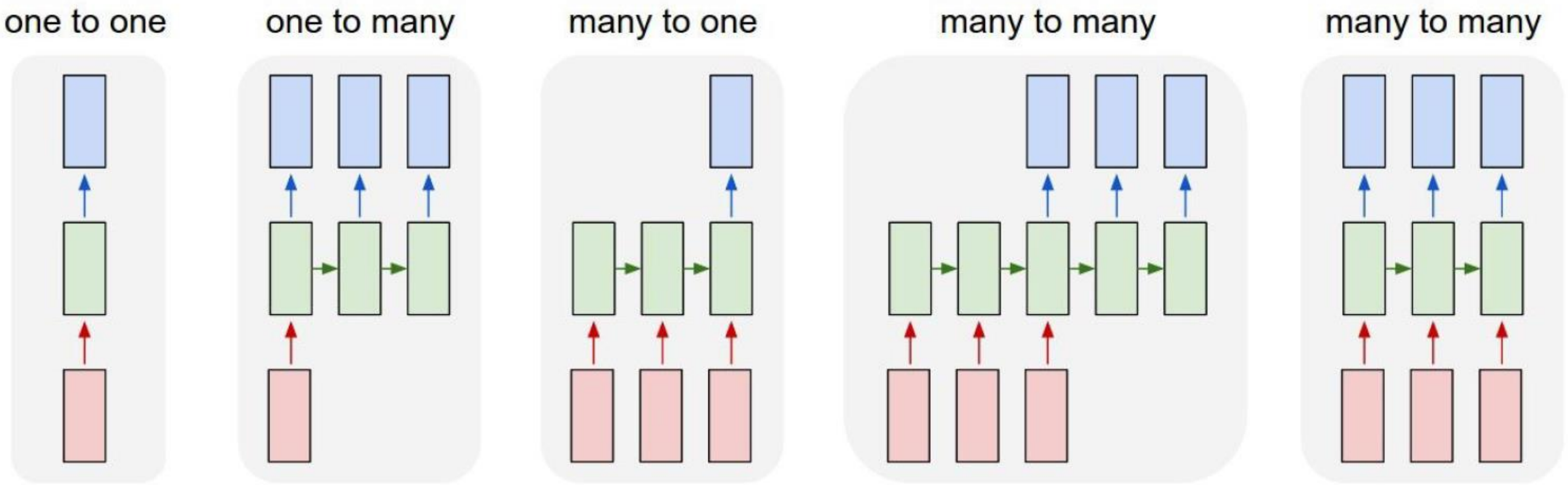
Example:
Character-level
Language Model
Sampling

Vocabulary:
[h,e,l,o]

At test-time sample characters one at a time, feed back to model

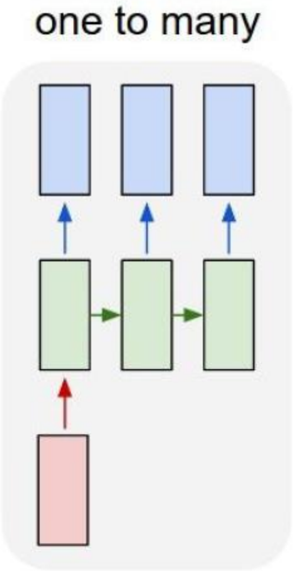


RNN applications



저작권: AI School

RNN applications

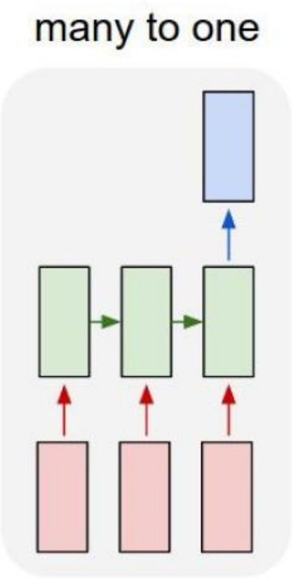


The man in grey swings a bat
while the man in black looks on.

A big bus sitting next to a person.

e.g., **Image captioning**
Image -> sequence of words

RNN applications



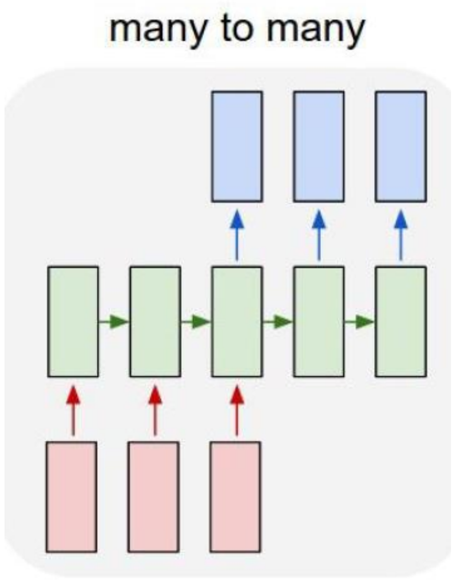
many to one

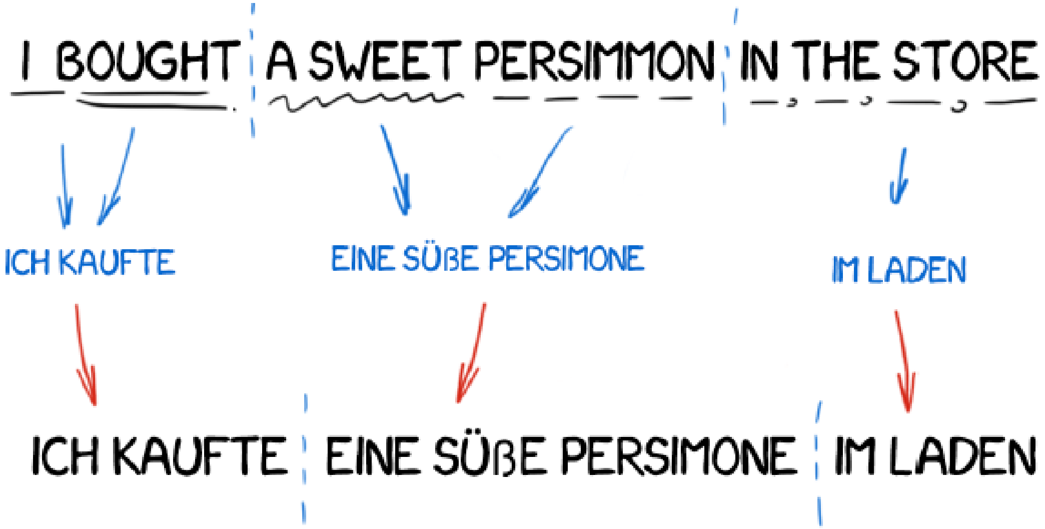
"I love this movie.
I've seen it many times
and it's still awesome." → 👍

"This movie is bad.
I don't like it it all.
It's terrible." → 👎

e.g., **Sentiment classification**
sequence of words -> sentiment

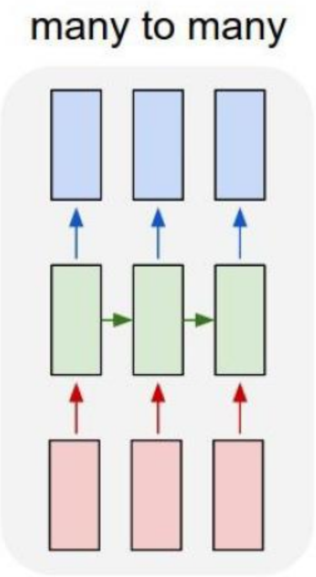
RNN applications

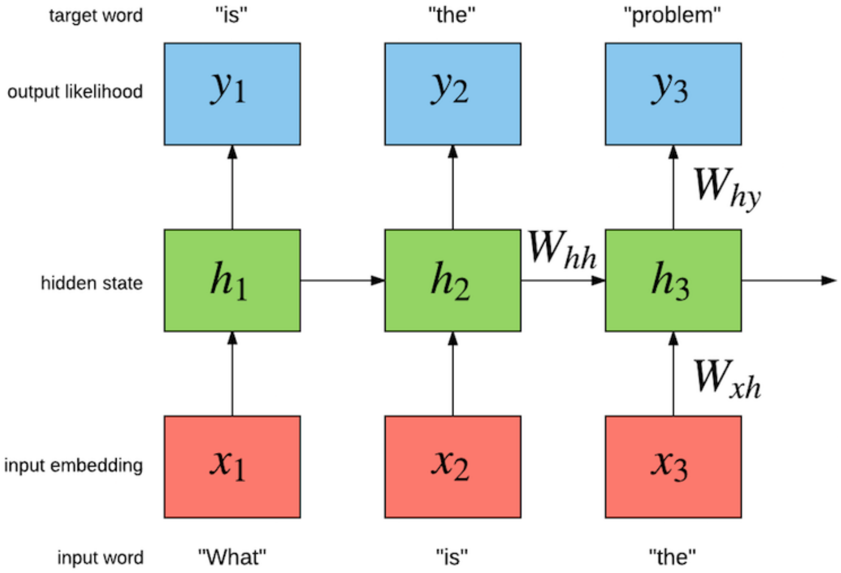




e.g., **Machine translation, Question answering**
sequence of words -> sequence of words

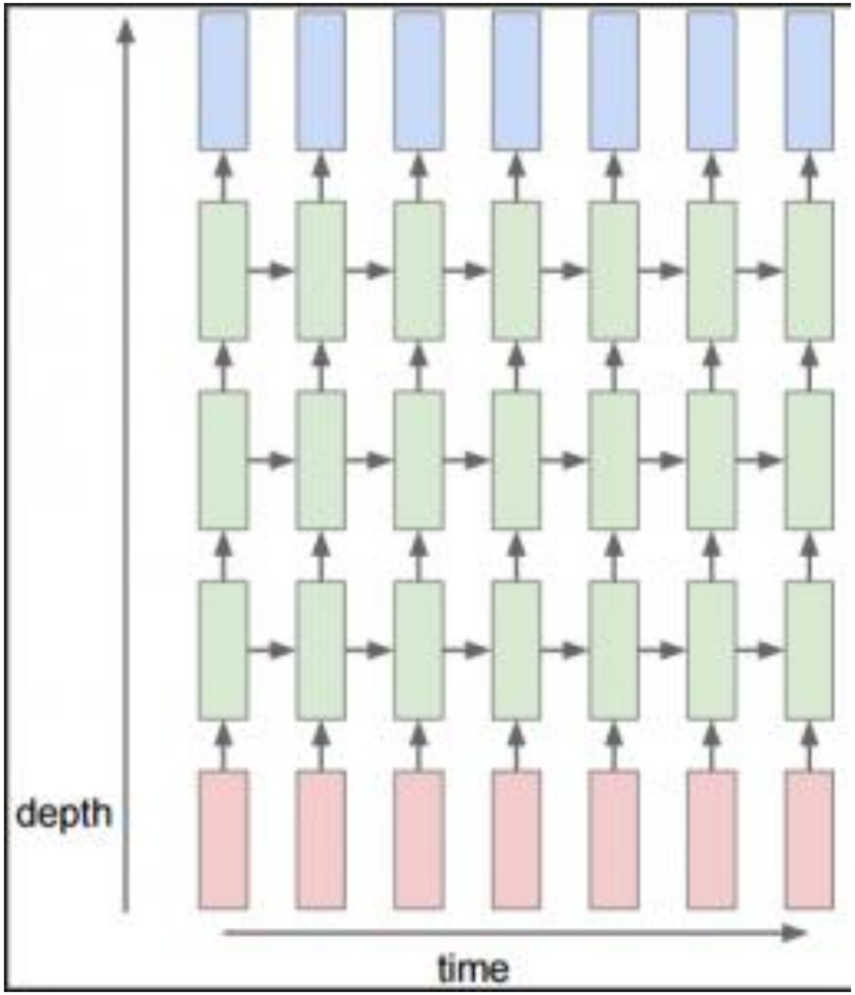
RNN applications



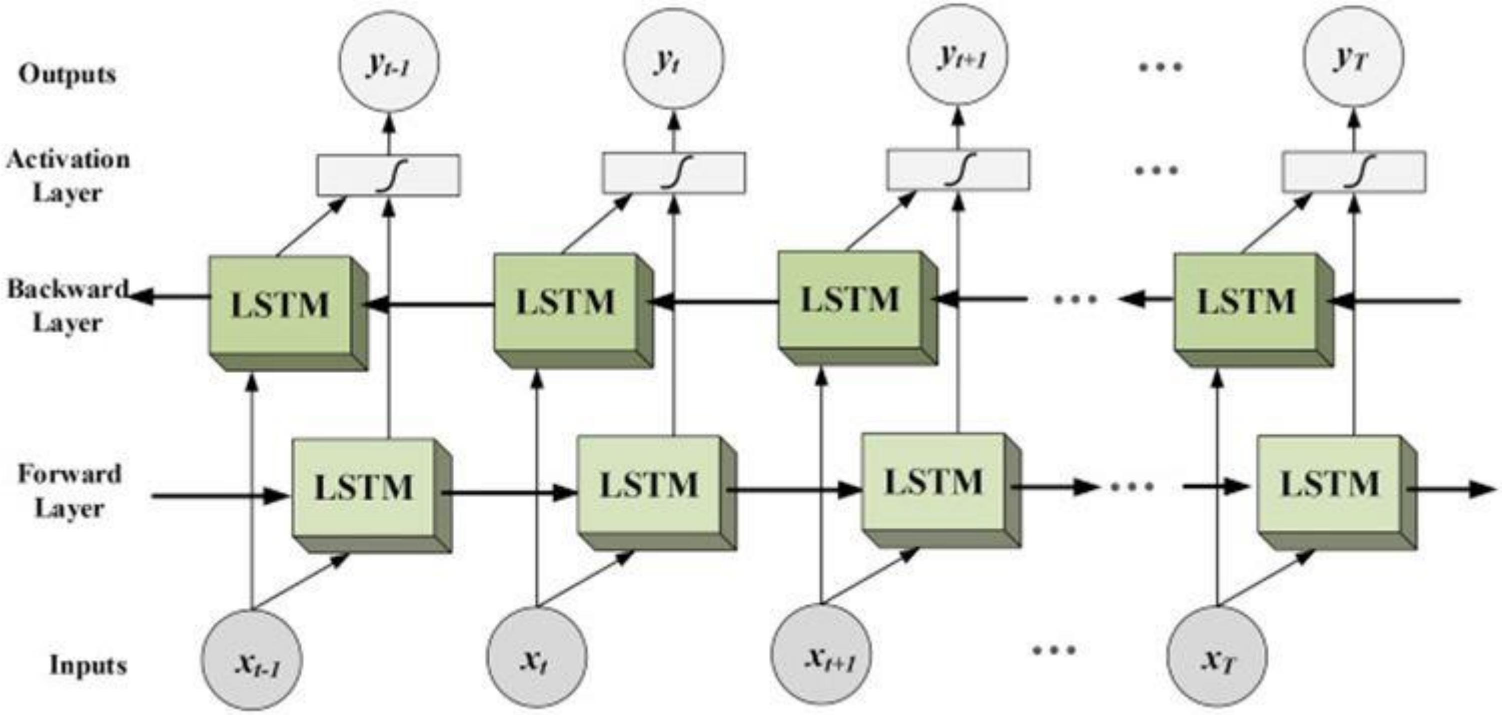


e.g., **Language modeling**

Multi-layer RNN

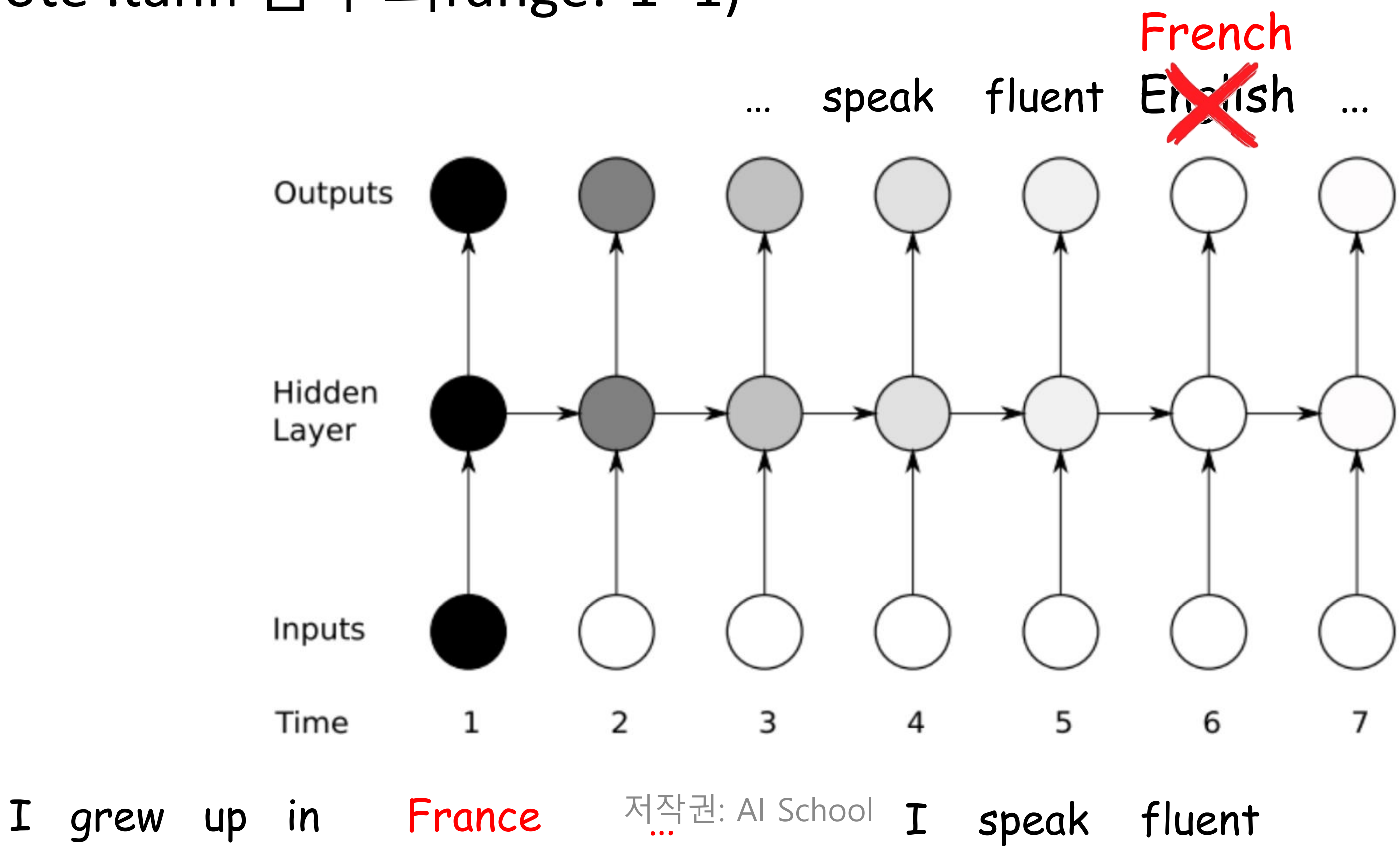


Bidirectional RNN



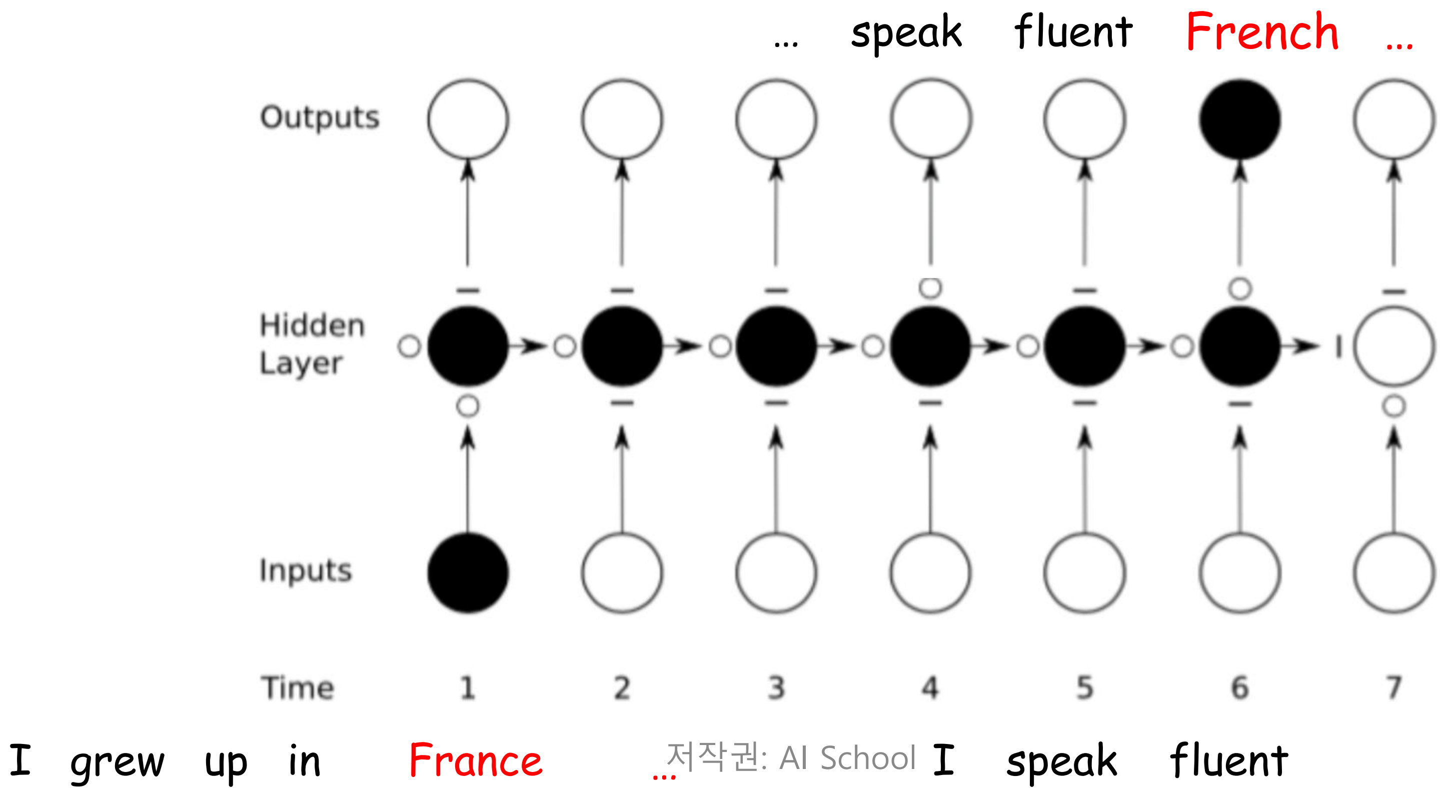
RNN의 한계

- 중요한 정보가 recurrent step이 계속됨에 따라 희석되는 문제 (long-term dependency)
- France라는중요한정보가점차희석됨
- (note :tanh 함수의range:-1~1)

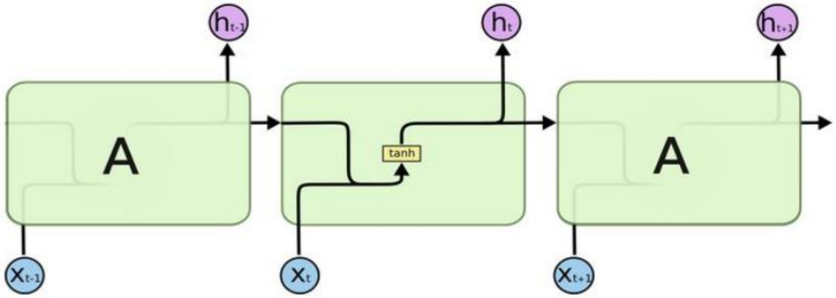

French
… speak fluent ~~English~~ …
Outputs
Hidden Layer
Inputs
Time
1
2
3
4
5
6
7
I grew up in France … I speak fluent

Long short-term memory (LSTM)

- 중요한 정보만 선택하여 이를 다음 state에 전달함으로써, long term dependency를 해결
- Cell에 정보를 저장하며, 정보들은 gate에 의해서 선택됨

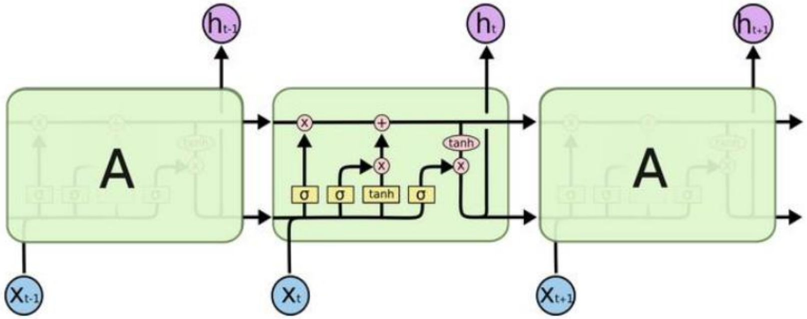


Long short-term memory (LSTM)



The repeating module in a standard RNN contains a single layer.

Basic RNN



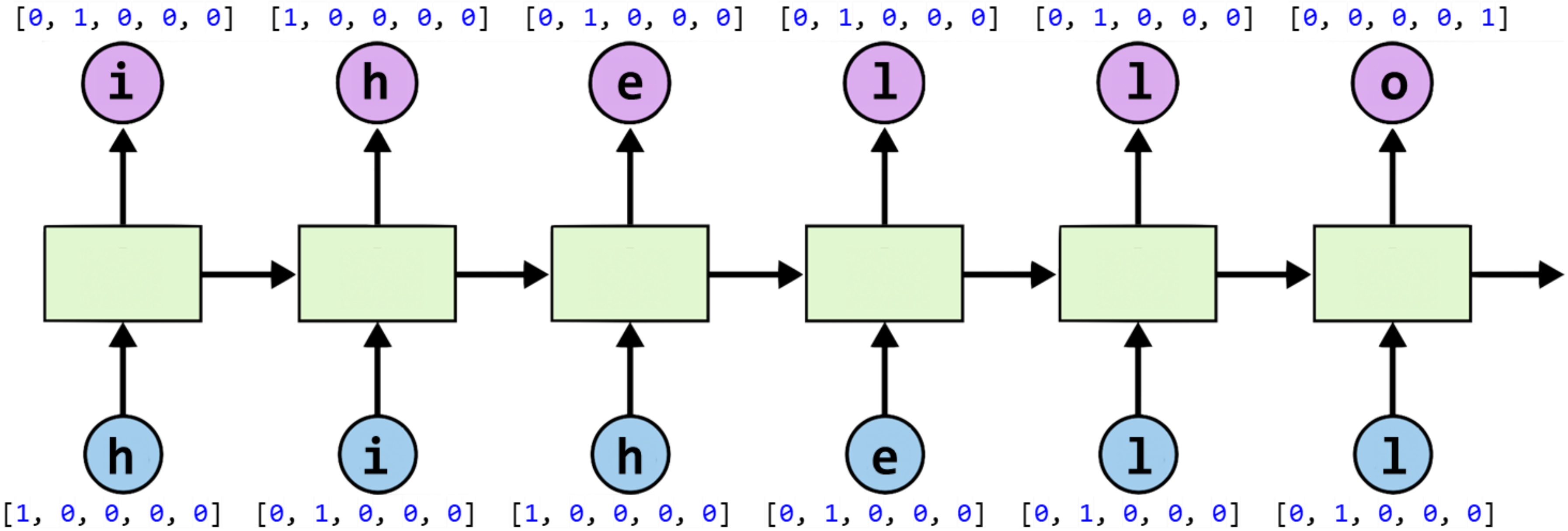
The repeating module in an LSTM contains four interacting layers.

LSTM

https://dgkim5360.tistory.com/entry/understanding-long-short-term-memory-lstm-kr

Character-LM

```
[1, 0, 0, 0, 0],    # h 0
[0, 1, 0, 0, 0],    # i 1
[0, 0, 1, 0, 0],    # e 2
[0, 0, 0, 1, 0],    # l 3
[0, 0, 0, 0, 1],    # o 4
```



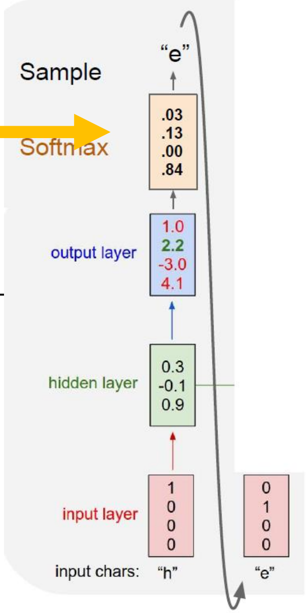
저작권: AI School

AI School 6기 9주차

RNN 기반 언어모델링 실습

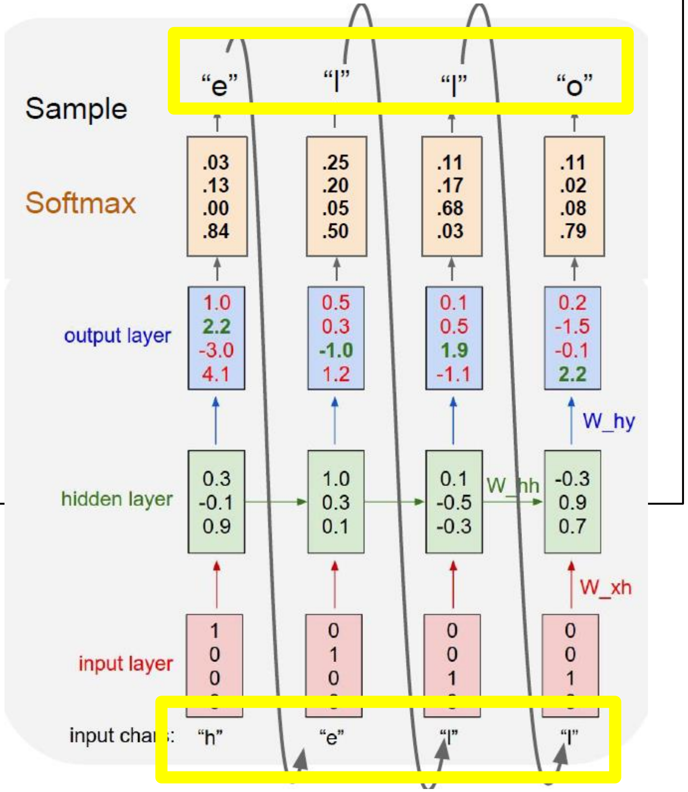
Character-LM

```
import tensorflow as tf
import numpy as np
from tensorflow.contrib import rnn

sentence = ("if you want to build a ship, don't drum up people together to "
            "collect wood and don't assign them tasks and work, but rather "
            "teach them to long for the endless immensity of the sea.")

char_set = list(set(sentence))
char_dic = {w: i for i, w in enumerate(char_set)}
# {'b': 0, ',': 1, ' ': 2, 'h': 3, 'w': 4, 'l': 5, "'": 6, 'e': 7, 'f': 8, 'n': 9, 'y': 10, ...}

hidden_size = 50
num_classes = len(char_set)
sequence_length = 10  # Any arbitrary number
learning_rate = 0.1
```

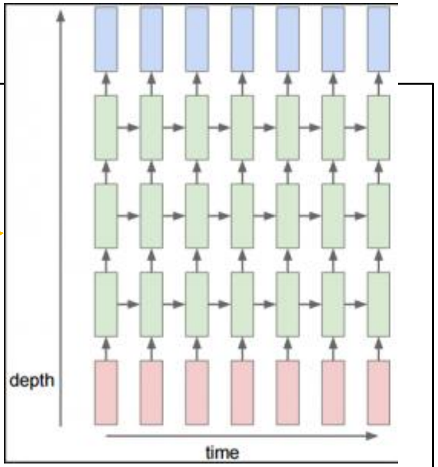


Character-LM

```
dataX = []
dataY = []
for i in range(0, len(sentence) - sequence_length):
    x_str = sentence[i:i + sequence_length] # h,e,l,l
    y_str = sentence[i + 1: i + sequence_length + 1] # e,l,l,o
    print(i, x_str, '->', y_str)

    x = [char_dic[c] for c in x_str]  # x str to index
    y = [char_dic[c] for c in y_str]  # y str to index

    dataX.append(x)
    dataY.append(y)

batch_size = len(dataX)
```



Character-LM



RNN layer 수

```
X = tf.placeholder(tf.int32, [None, sequence_length])
Y = tf.placeholder(tf.int32, [None, sequence_length])

# One-hot encoding
X_one_hot = tf.one_hot(X, num_classes)
print(X_one_hot)  # check out the shape

# Make a lstm cell with hidden_size (each unit output vector size)
def lstm_cell():
    cell = rnn.BasicLSTMCell(hidden_size, state_is_tuple=True)
    return cell

multi_cells = rnn.MultiRNNCell([lstm_cell() for _ in range(2)], state_is_tuple=True)

# outputs: unfolding size x hidden size, state = hidden size
outputs, _states = tf.nn.dynamic_rnn(multi_cells, X_one_hot, dtype=tf.float32)
```

Character-LM

```
# FC layer
X_for_fc = tf.reshape(outputs, [-1, hidden_size])
outputs = tf.contrib.layers.fully_connected(X_for_fc, num_classes,
activation_fn=None)
# softmax_w = tf.get_variable("softmax_w",[hidden_size, num_classes])
# softmax_b = tf.get_variable("softmax_b",[num_classes])
# outputs = tf.matmul(X_for_fc, softmax_w) + softmax_b

# reshape out for sequence_loss
outputs = tf.reshape(outputs, [batch_size, sequence_length, num_classes])

# All weights are 1 (equal weights)
weights = tf.ones([batch_size, sequence_length])

sequence_loss = tf.contrib.seq2seq.sequence_loss(
    logits=outputs, targets=Y, weights=weights)
mean_loss = tf.reduce_mean(sequence_loss)
train_op =
tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(mean_loss)
```

Character-LM

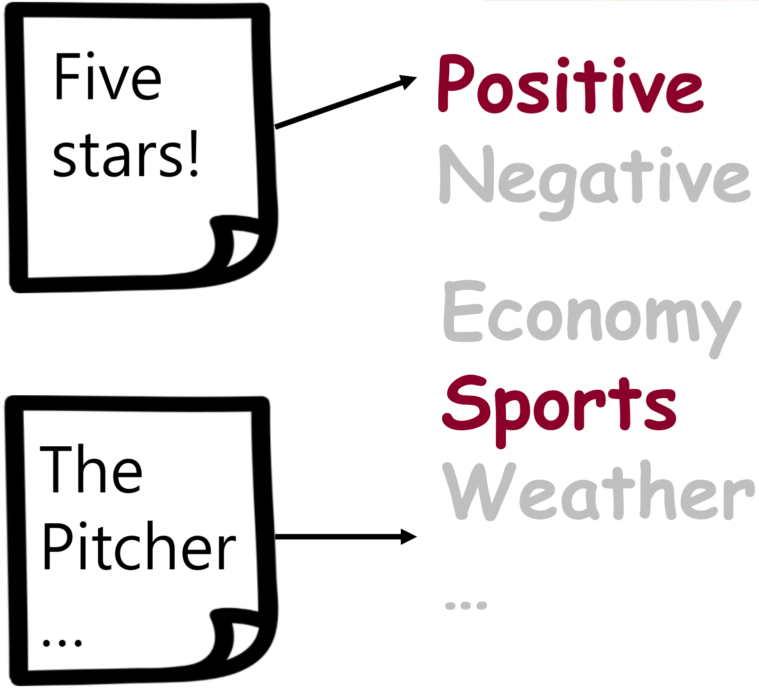
```
sess = tf.Session()
sess.run(tf.global_variables_initializer())

for i in range(500):
    _, l, results = sess.run(
        [train_op, mean_loss, outputs], feed_dict={X: dataX, Y: dataY})
    for j, result in enumerate(results):
        index = np.argmax(result, axis=1)
        print(i, j, ''.join([char_set[t] for t in index]), l)

# Let's print the last char of each result to check it works
results = sess.run(outputs, feed_dict={X: dataX})
for j, result in enumerate(results):
    index = np.argmax(result, axis=1)
    if j is 0:  # print all for the first result to make a sentence
        print(''.join([char_set[t] for t in index]), end='')
    else:
        print(char_set[index[-1]], end='')
```

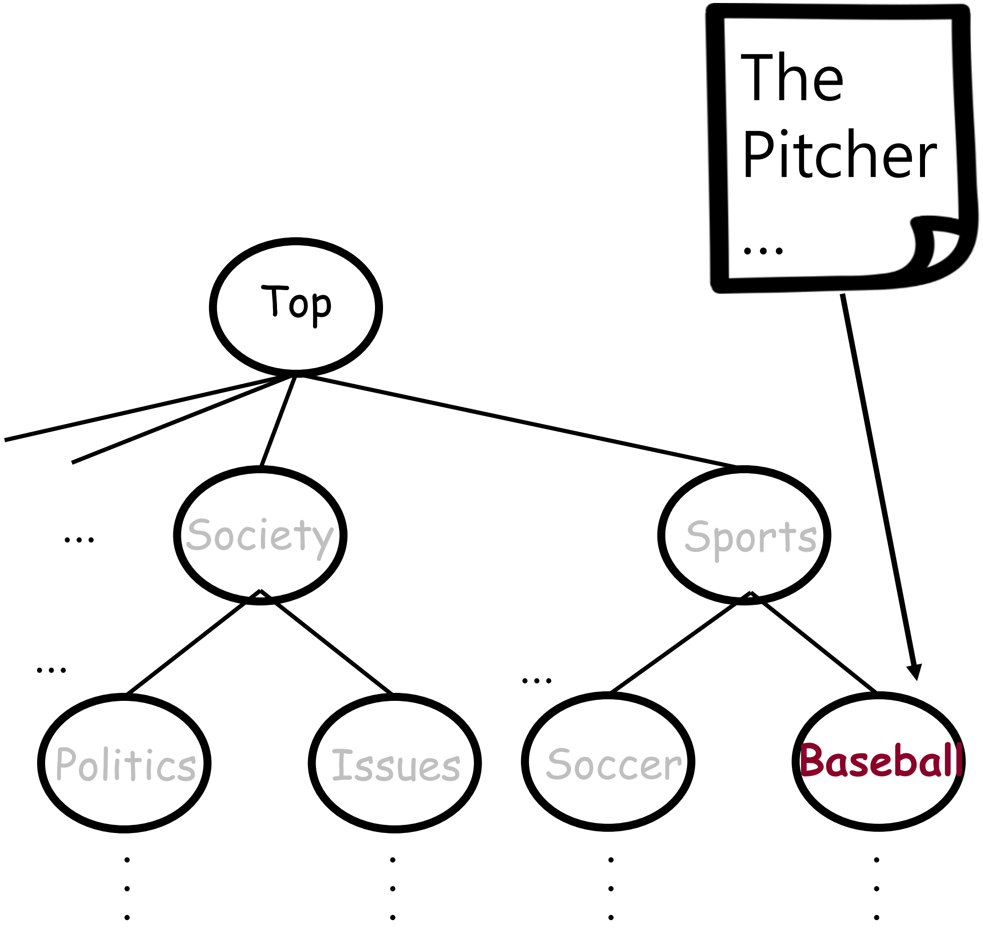
AI School 6기 9주차

CNN 기반 텍스트 분류기

CNN for text classification

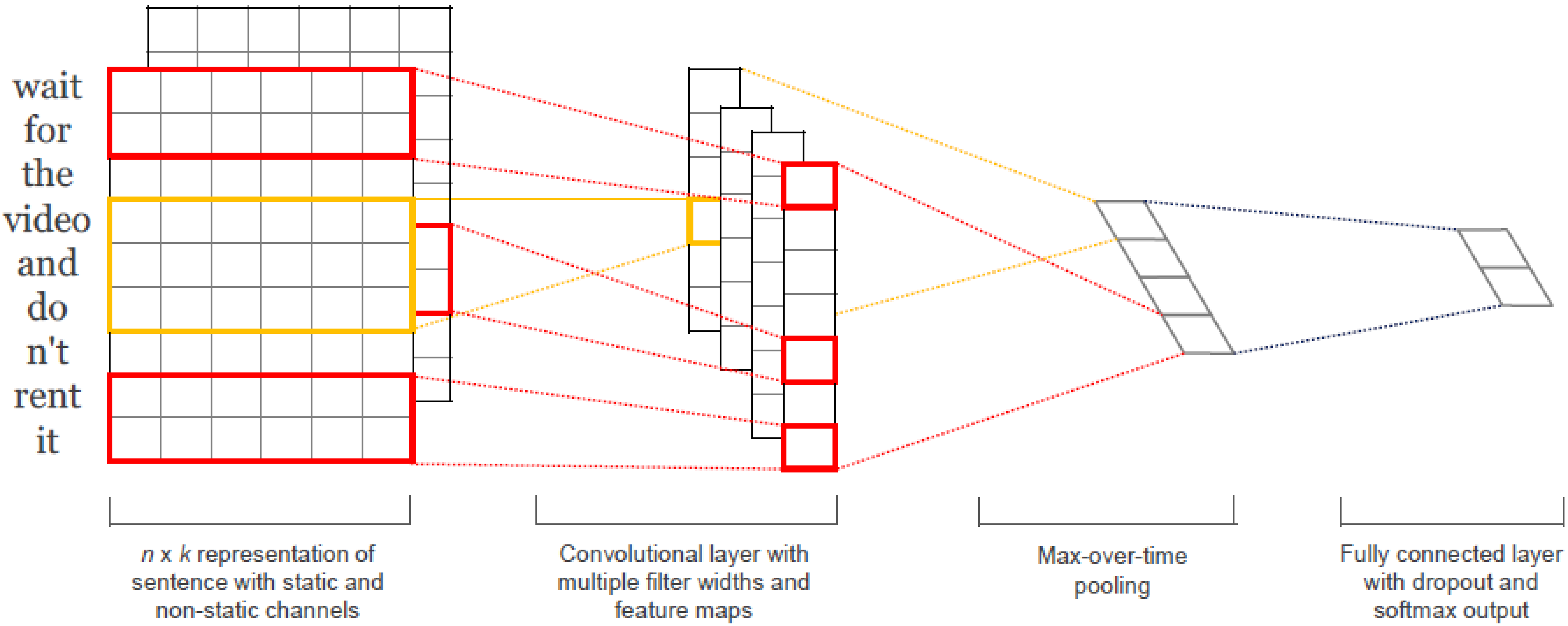
- CNN for sentence classification [Kim et al., 2014]





CNN for text classification

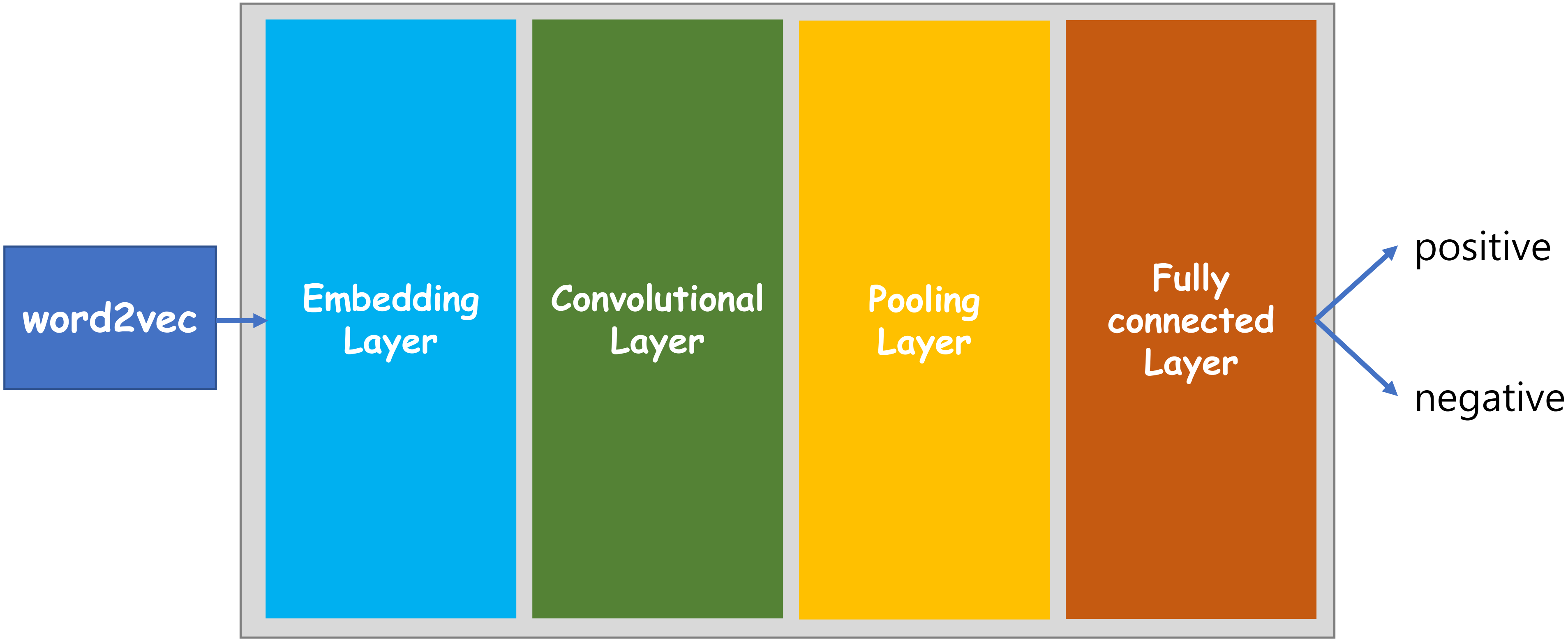
- Model overview


wait
for
the
video
and
do
n't
rent
it
n x *k* representation of sentence with static and non-static channels
Convolutional layer with multiple filter widths and feature maps
Max-over-time pooling
Fully connected layer with dropout and softmax output

저작권: AI School

CNN for text classification

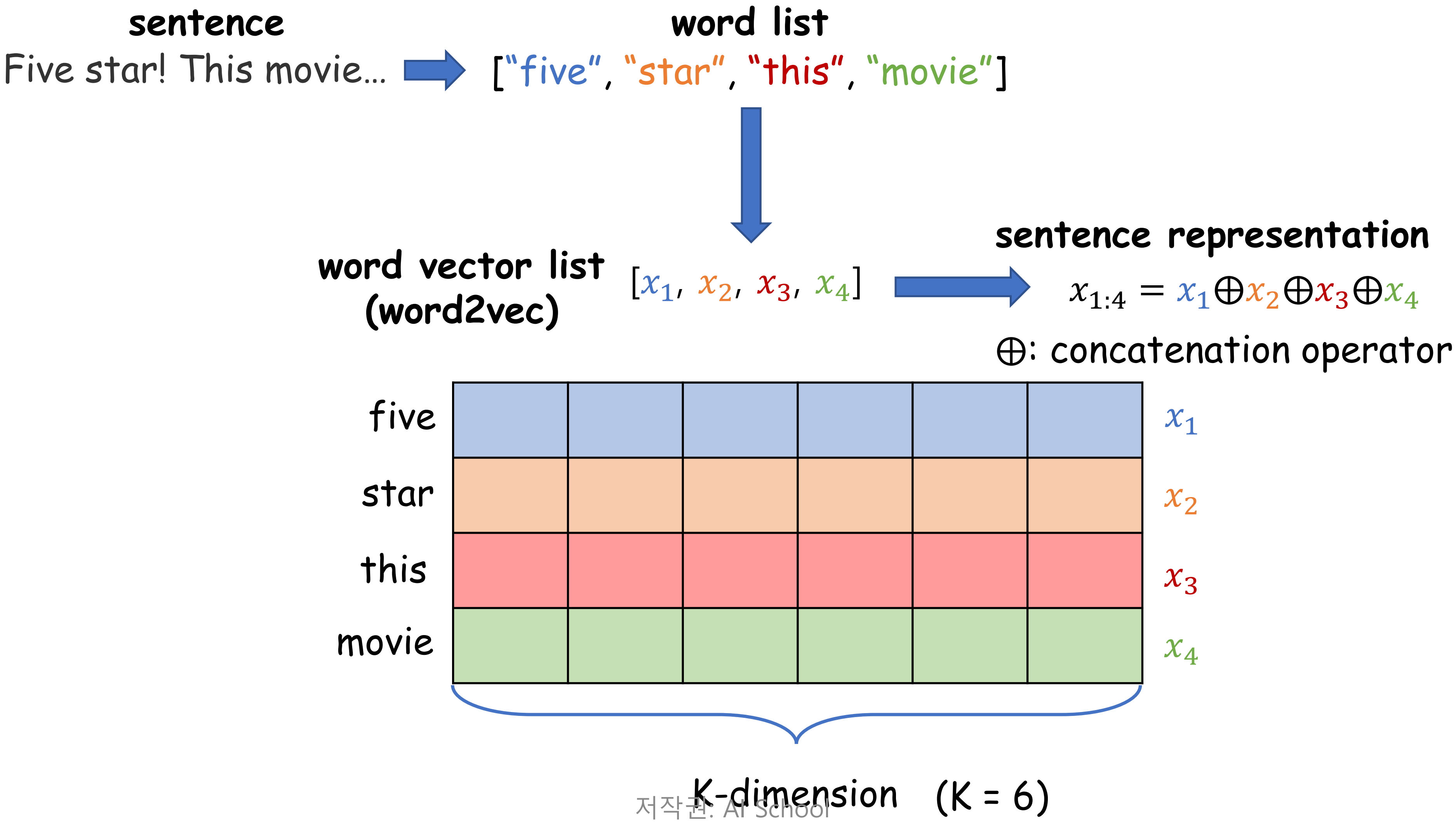
- Model overview



저작권: AI School

CNN for text classification

- Word2vec & Embedding layer



sentence

Five star! This movie...

word list

["five", "star", "this", "movie"]

word vector list (word2vec)

$[x_1, x_2, x_3, x_4]$

sentence representation

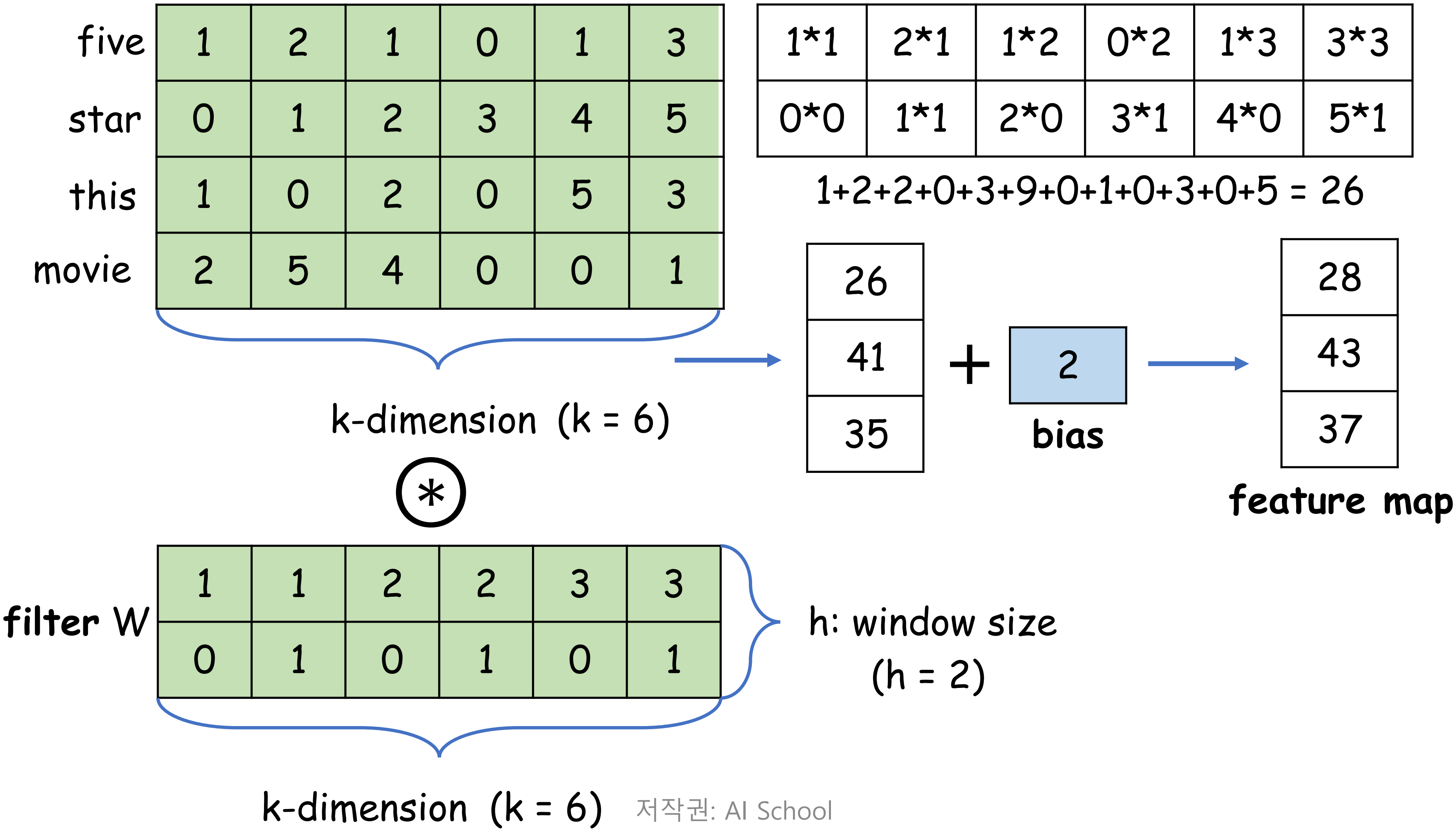
$$x_{1:4} = x_1 \oplus x_2 \oplus x_3 \oplus x_4$$

$\oplus$: concatenation operator

K-dimension (K = 6)

CNN for text classification

- Convolutional layer



five	1	2	1	0	1	3
star	0	1	2	3	4	5
this	1	0	2	0	5	3
movie	2	5	4	0	0	1

k-dimension (k = 6)

∗

filter W	1	1	2	2	3	3
	0	1	0	1	0	1

h: window size (h = 2)

k-dimension (k = 6)

1*1	2*1	1*2	0*2	1*3	3*3
0*0	1*1	2*0	3*1	4*0	5*1

1+2+2+0+3+9+0+1+0+3+0+5 = 26

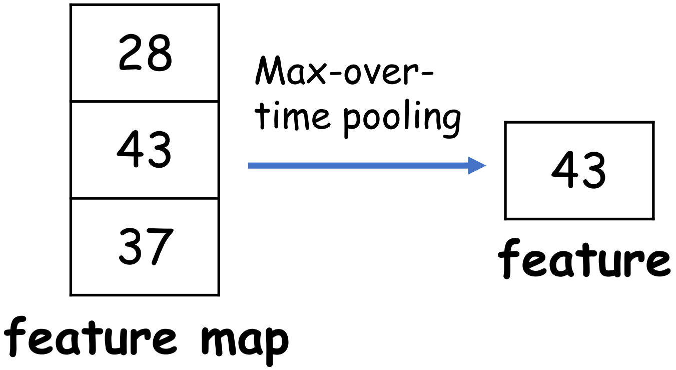
$$\begin{bmatrix}26\\41\\35\end{bmatrix} + \underbrace{\boxed{2}}_{\text{bias}} \rightarrow \underbrace{\begin{bmatrix}28\\43\\37\end{bmatrix}}_{\text{feature map}}$$

CNN for text classification

- Pooling layer

five	1	2	1	0	1	3
star	0	1	2	3	4	5
this	1	0	2	0	5	3
movie	2	5	4	0	0	1

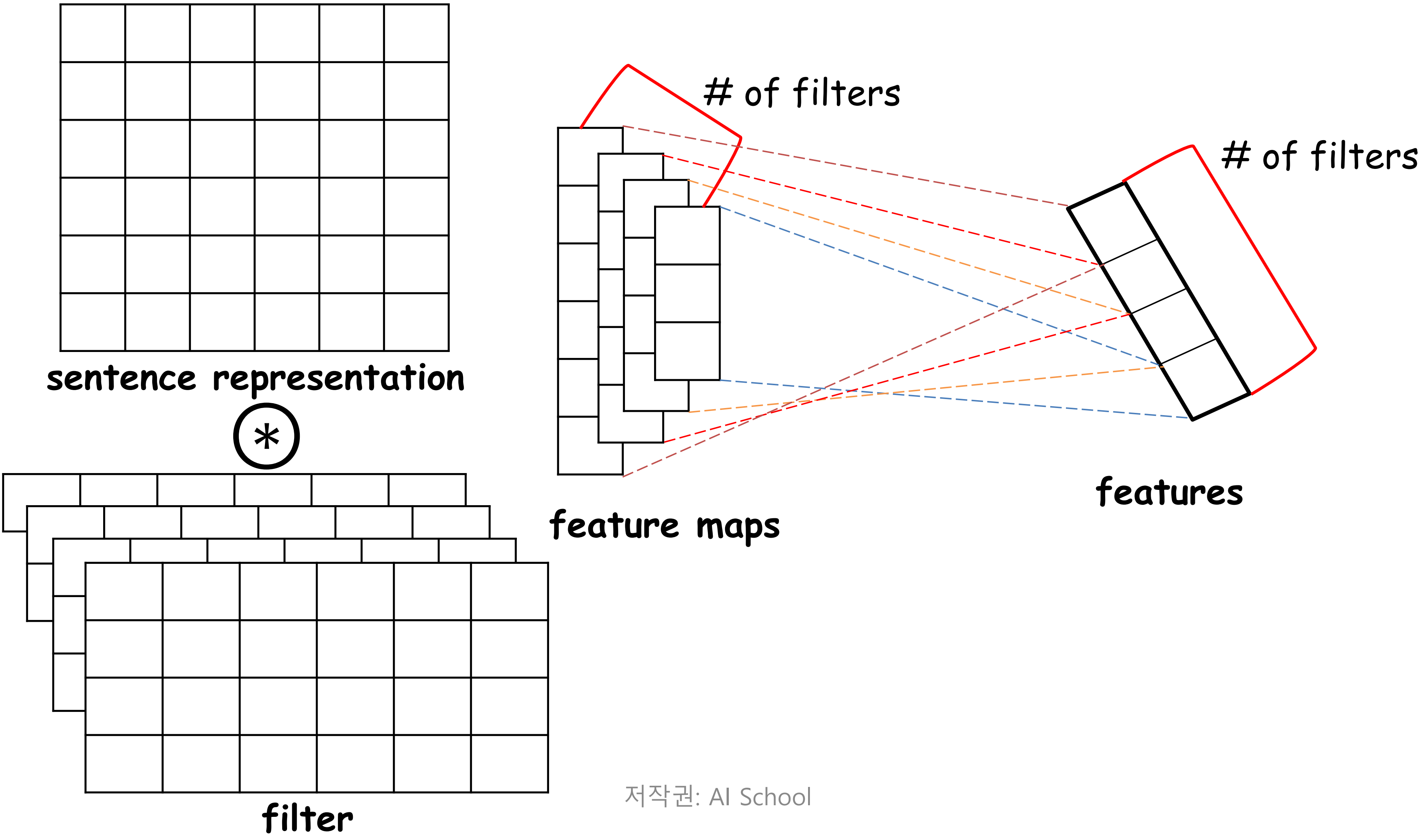
filter W	1	1	2	2	3	3
	0	1	0	1	0	1



One filter -> **One** feature
Multiple filter -> **Multiple** feature

CNN for text classification

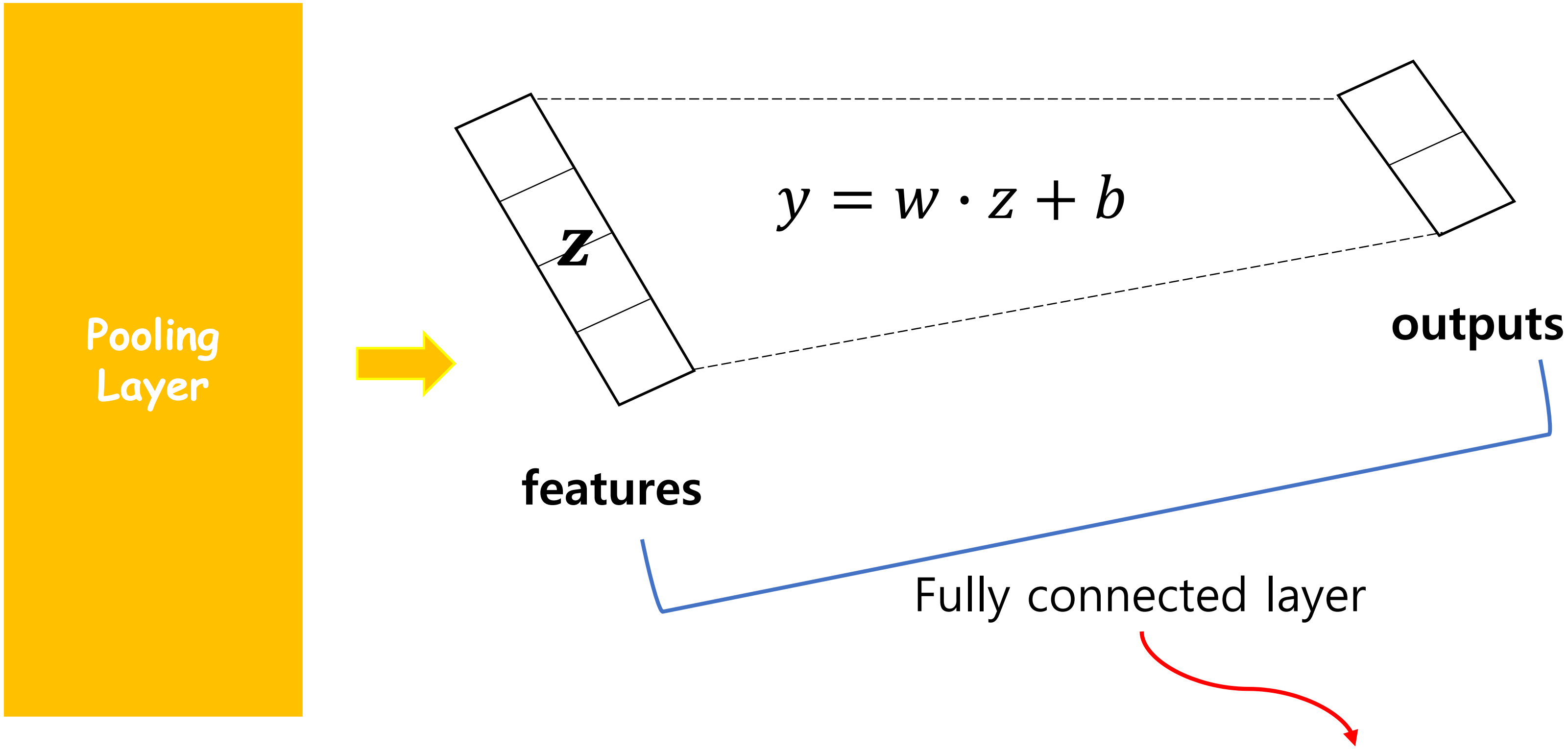
- Pooling layer



저작권: AI School

CNN for text classification

- Fully connected layer



Classification

- softmax output

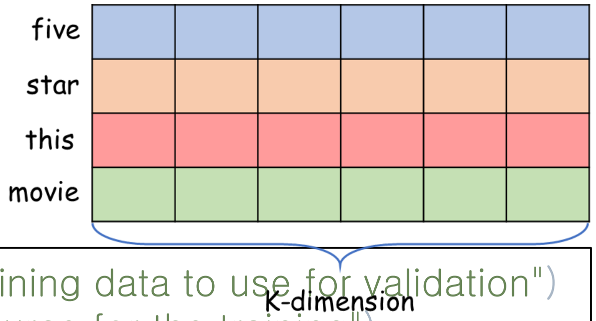
CNN 분류기 학습

- Import

```
import tensorflow as tf
import numpy as np
import os
import time
import datetime
import re
import smart_open
import pickle
import text_classification_master.data_helpers as dh
from text_classification_master.text_cnn import TextCNN
from gensim.models.keyedvectors import KeyedVectors
```

Hyperparameters

- train.py

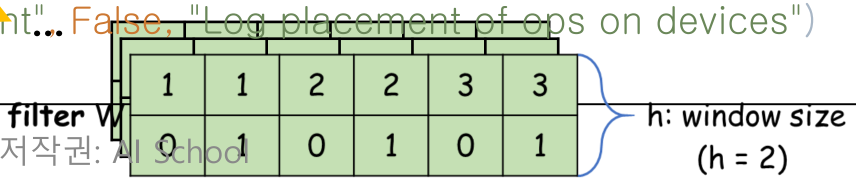


```
tf.flags.DEFINE_float("dev_sample_percentage", .1, "Percentage of the training data to use for validation")
tf.flags.DEFINE_string("x_train_file", "./data/train/x_TrecTrain.txt", "Data source for the training")
tf.flags.DEFINE_string("t_train_file", "./data/train/t_TrecTrain.txt", "Data source for the training")
tf.flags.DEFINE_string("word2vec", "./data/GoogleNews-vectors-negative300.bin", "Word2vec file")
tf.flags.DEFINE_integer("vocab_size", 30000, "Vocabulary size (defuault: 0)")
tf.flags.DEFINE_integer("num_classes", 0, "The number of labels (defualt: 0)")
tf.flags.DEFINE_integer("max_length", 0, "max sequence length (defualt: 0)")
tf.flags.DEFINE_integer("embedding_dim", 300, "Dimensionality of character embedding (default: 128)")
tf.flags.DEFINE_string("filter_sizes", "3,4,5", "Comma-separated filter sizes (default: '3,4,5')")
tf.flags.DEFINE_integer("num_filters", 100, "Number of filters per filter size (default: 128)")
tf.flags.DEFINE_float("dropout_keep_prob", 0.5, "Dropout keep probability (default: 0.5)")
tf.flags.DEFINE_float("l2_reg_lambda", 0.001, "L2 regularization lambda (default: 0.0)")
tf.flags.DEFINE_float("lr_decay", 0.9, "Learning rate decay rate (default: 0.98)")
tf.flags.DEFINE_float("lr", 1e-3, "Learning rate(default: 0.01)")
tf.flags.DEFINE_integer("batch_size", 50, "Batch Size (default: 64)")
tf.flags.DEFINE_integer("num_epochs", 200, "Number of training epochs (default: 200)")
tf.flags.DEFINE_integer("evaluate_every", 100, "Evaluate model on dev set after this many steps (default: 100)")
tf.flags.DEFINE_integer("checkpoint_every", 100, "Save model after this many steps (default: 100)")
tf.flags.DEFINE_integer("num_checkpoints", 3, "Number of checkpoints to store (default: 5)")
tf.flags.DEFINE_boolean("allow_soft_placement", True, "Allow device soft device placement")
tf.flags.DEFINE_boolean("log_device_placement", False, "Log placement of ops on devices")

FLAGS = tf.flags.FLAGS
```

단어를 표현하는 벡터의 크기

Filter의 높이
or
Window size
or
N-gram

Filter 종류별 개수



Data loading & preprocessing

- train.py

```
print("Loading data...")
x_text, y, _ = dh.load_data(FLAGS.x_train_file, FLAGS.t_train_file)

# Build vocabulary
word_id_dict, _ = dh.buildVocab(x_text, FLAGS.vocab_size)
print(word_id_dict)
FLAGS.vocab_size = len(word_id_dict) + 4
print("vocabulary size: ", FLAGS.vocab_size)

for word_id in word_id_dict.keys():
    word_id_dict[word_id] += 4  # 0: <pad>, 1: <unk>, 2: <s>, 3: </s>
word_id_dict['<pad>'] = 0
word_id_dict['<unk>'] = 1
word_id_dict['<s>'] = 2
word_id_dict['</s>'] = 3
```

data loading

- data_helpers.py

```
def load_data(x_file, t_file):
    # Load data from files

    t_large = []

    x_text = list(open(x_file, "r", encoding='UTF8').readlines())
    x_text = [s.strip() for s in x_text]
    x_text = np.array([clean_str(sent) for sent in x_text])

    lengths = np.array(list(map(len, [sent.split(" ") for sent in x_text])))

    t_text_temp = np.array(list(open(t_file, "r", encoding='UTF8').readlines()))

    maxLabel = t_text_temp.astype(np.int)
    print(maxLabel)
    maxLabel = np.max(maxLabel) + 1
    print("max label: "+str(maxLabel))
    for i, s in enumerate(t_text_temp):
        t = np.zeros(maxLabel)
        t[int(s)] = 1.0
        t_large.append(t)

    t_large = np.array(t_large)

    return [x_text, t_large, lengths]
```

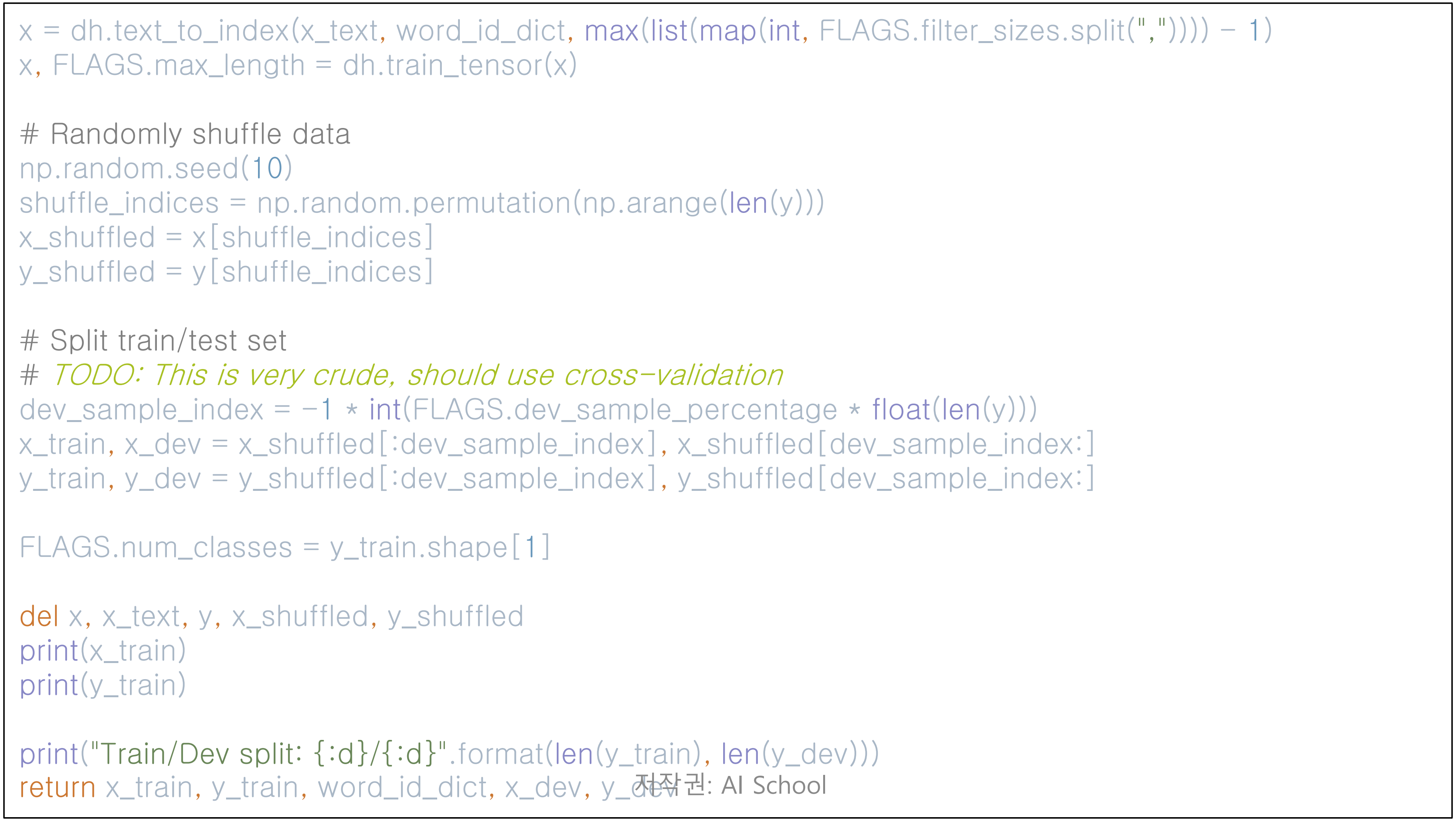
Build Vocabulary

- train.py

```
def buildVocab(sentences, vocab_size):
    # Build vocabulary
    words = []
    for sentence in sentences: words.extend(sentence.split())
    print("The number of words: ", len(words))
    word_counts = collections.Counter(words)
    # Mapping from index to word
    vocabulary_inv = [x[0] for x in word_counts.most_common(vocab_size)]
    # Mapping from word to index
    vocabulary = {x: i for i, x in enumerate(vocabulary_inv)}
    return [vocabulary, vocabulary_inv]
```

Text to indices, indices to tensor

- train.py



```
x = dh.text_to_index(x_text, word_id_dict, max(list(map(int, FLAGS.filter_sizes.split(",")))) - 1)
x, FLAGS.max_length = dh.train_tensor(x)

# Randomly shuffle data
np.random.seed(10)
shuffle_indices = np.random.permutation(np.arange(len(y)))
x_shuffled = x[shuffle_indices]
y_shuffled = y[shuffle_indices]

# Split train/test set
# TODO: This is very crude, should use cross-validation
dev_sample_index = -1 * int(FLAGS.dev_sample_percentage * float(len(y)))
x_train, x_dev = x_shuffled[:dev_sample_index], x_shuffled[dev_sample_index:]
y_train, y_dev = y_shuffled[:dev_sample_index], y_shuffled[dev_sample_index:]

FLAGS.num_classes = y_train.shape[1]

del x, x_text, y, x_shuffled, y_shuffled
print(x_train)
print(y_train)

print("Train/Dev split: {:d}/{:d}".format(len(y_train), len(y_dev)))
return x_train, y_train, word_id_dict, x_dev, y_dev
```

Text to indices, indices to tensor

- train.py

```
x = dh.text_to_index(x_text, word_id_dict, max(list(map(int, FLAGS.filter_sizes.split(",")))) - 1)
x, FLAGS.max_length = dh.train_tensor(x)

# Randomly shuffle data
np.random.seed(10)
shuffle_indices = np.random.permutation(np.arange(len(y)))
x_shuffled = x[shuffle_indices]
y_shuffled = y[shuffle_indices]

# Split train/test set
# TODO: This is very crude, should use cross-validation
dev_sample_index = -1 * int(FLAGS.dev_sample_percentage * float(len(y)))
x_train, x_dev = x_shuffled[:dev_sample_index], x_shuffled[dev_sample_index:]
y_train, y_dev = y_shuffled[:dev_sample_index], y_shuffled[dev_sample_index:]

FLAGS.num_classes = y_train.shape[1]

del x, x_text, y, x_shuffled, y_shuffled
print(x_train)
print(y_train)

print("Train/Dev split: {:d}/{:d}".format(len(y_train), len(y_dev)))
return x_train, y_train, word_id_dict, x_dev, y_dev
```

Text to indices, indices to tensor

- data_helpers.py

```
def text_to_index(text_list, word_to_id, nb_pad):
    text_indices = []
    for text in text_list:
        words = text.split(" ")
        pad = [0 for _ in range(nb_pad) ]
        ids = []
        for word in words:
            if word in word_to_id:
                word_id = word_to_id[word]
            else:
                word_id = 1
            ids.append(word_id)
        ids = pad + ids
        text_indices.append(ids)
    return text_indices

def train_tensor(batches):
    max_length = max([len(batch) for batch in batches])
    tensor = np.zeros((len(batches), max_length), dtype=np.int64)
    for i, indices in enumerate(batches):
        tensor[i, :len(indices)] = np.asarray(indices, dtype=np.int64)

    return tensor, max_length
```

TextCNN class & input

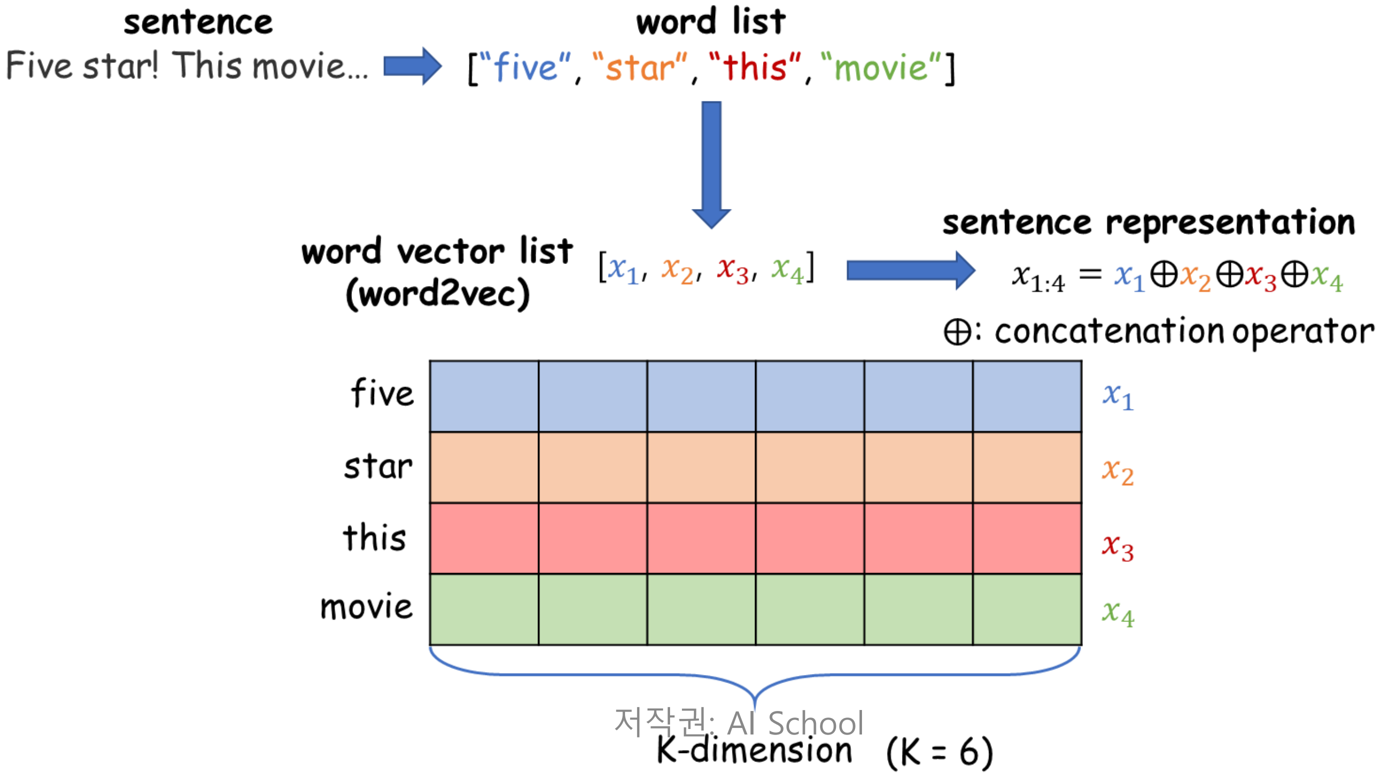
- text_cnn.py

```
import tensorflow as tf
import numpy as np


class TextCNN(object):

    def __init__(self, config):
        self.num_classes = config["num_classes"]
        self.vocab_size = config["vocab_size"]
        self.embedding_size = config["embedding_dim"]
        self.filter_sizes = list(map(int, config["filter_sizes"].split(",")))
        self.num_filters = config["num_filters"]
        self.l2_reg_lambda = config["l2_reg_lambda"]
        self.max_length = config["max_length"]

        # Placeholders for input, output and dropout
        self.input_x = tf.placeholder(tf.int32, [None, self.max_length], name="input_x")
        self.input_y = tf.placeholder(tf.float32, [None, self.num_classes], name="input_y")
        self.dropout_keep_prob = tf.placeholder(tf.float32, name="dropout_keep_prob")
```

Embedding layer

- text_cnn.py

```
# Embedding layer
with tf.device('/gpu:0'), tf.name_scope("embedding"):
    self.W = tf.Variable(
        tf.random_uniform([self.vocab_size, self.embedding_size], -1.0, 1.0), trainable=True,
        name="W")
    self.embedded_chars = tf.nn.embedding_lookup(self.W, self.input_x)
    self.embedded_chars_expanded = tf.expand_dims(self.embedded_chars, -1)
```



sentence

Five star! This movie...

word list

["five", "star", "this", "movie"]

word vector list
(word2vec)

$[x_1, x_2, x_3, x_4]$

sentence representation

$x_{1:4} = x_1 \oplus x_2 \oplus x_3 \oplus x_4$

$\oplus$: concatenation operator

five x_1

star x_2

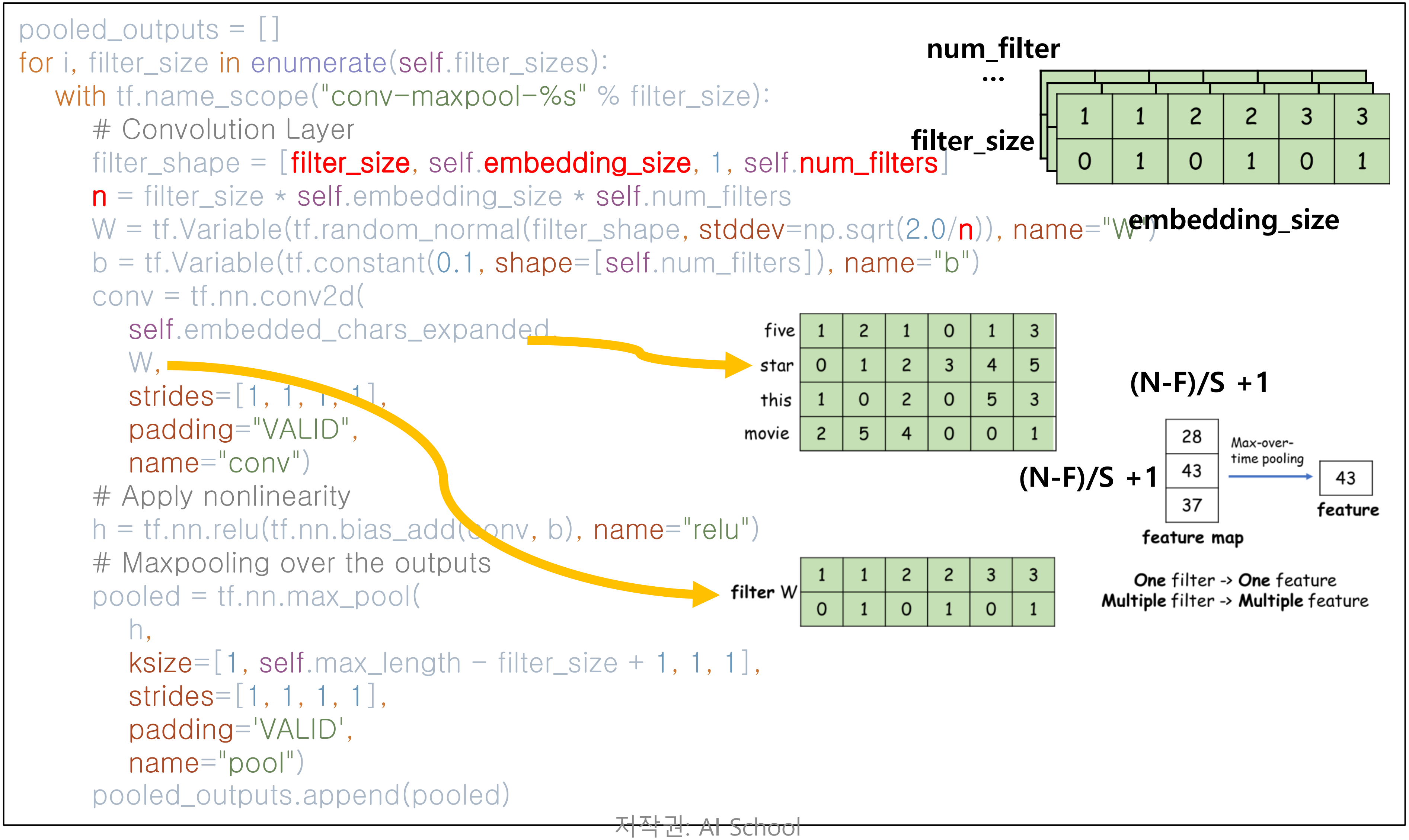
this x_3

movie x_4

K-dimension (K = 6)

Convolutional layer

- text_cnn.py



```
pooled_outputs = []
for i, filter_size in enumerate(self.filter_sizes):
    with tf.name_scope("conv-maxpool-%s" % filter_size):
        # Convolution Layer
        filter_shape = [filter_size, self.embedding_size, 1, self.num_filters]
        n = filter_size * self.embedding_size * self.num_filters
        W = tf.Variable(tf.random_normal(filter_shape, stddev=np.sqrt(2.0/n)), name="W")
        b = tf.Variable(tf.constant(0.1, shape=[self.num_filters]), name="b")
        conv = tf.nn.conv2d(
            self.embedded_chars_expanded,
            W,
            strides=[1, 1, 1, 1],
            padding="VALID",
            name="conv")
        # Apply nonlinearity
        h = tf.nn.relu(tf.nn.bias_add(conv, b), name="relu")
        # Maxpooling over the outputs
        pooled = tf.nn.max_pool(
            h,
            ksize=[1, self.max_length - filter_size + 1, 1, 1],
            strides=[1, 1, 1, 1],
            padding='VALID',
            name="pool")
        pooled_outputs.append(pooled)
```

1	1	2	2	3	3
0	1	0	1	0	1

five	1	2	1	0	1	3
star	0	1	2	3	4	5
this	1	0	2	0	5	3
movie	2	5	4	0	0	1

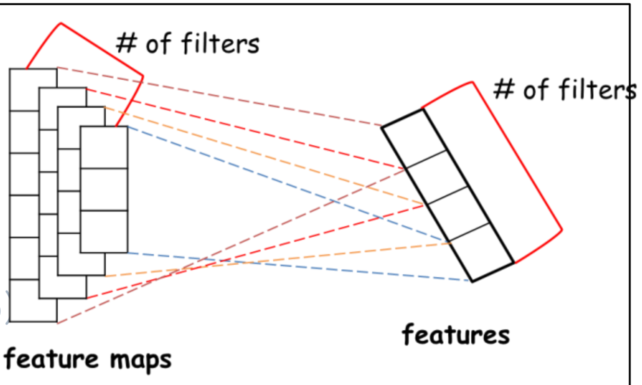
filter W	1	1	2	2	3	3
	0	1	0	1	0	1

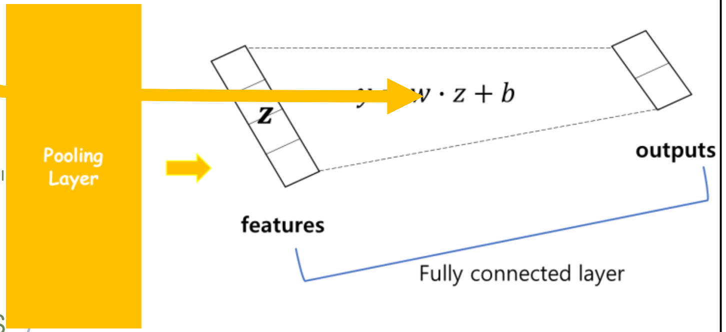
One filter -> **One** feature
Multiple filter -> **Multiple** feature

Fully connected layer

- text_cnn.py

```
# Combine all the pooled features
num_filters_total = num_filters * len(filter_sizes)
self.h_pool = tf.concat(pooled_outputs, 3)
self.h_pool_flat = tf.reshape(self.h_pool, [-1, num_filters_total])

with tf.name_scope("dropout"):
    self.h_drop = tf.nn.dropout(self.h_pool_flat, self.dropout_keep_prob)

with tf.name_scope("output"):
    W = tf.get_variable(
        "W",
        shape=[num_filters_total, num_classes],
        initializer=tf.contrib.layers.xavier_initializer())
    b = tf.Variable(tf.constant(0.1, shape=[num_classes]), name='
    l2_loss += tf.nn.l2_loss(W)
    l2_loss += tf.nn.l2_loss(b)
    self.scores = tf.nn.xw_plus_b(self.h_drop, W, b, name="scores")
    self.predictions = tf.argmax(self.scores, 1, name="predictions")
```





Fully connected layer

- text_cnn.py

```
costs = [ ]
for var in tf.trainable_variables():
    costs.append(tf.nn.l2_loss(var))
l2_loss = tf.add_n(costs)

# Calculate mean cross-entropy loss
with tf.name_scope("loss"):
    losses = tf.nn.softmax_cross_entropy_with_logits(logits=self.scores, labels=self.input_y)
    self.loss = tf.reduce_mean(losses) + self.l2_reg_lambda * l2_loss

# Accuracy
with tf.name_scope("accuracy"):
    correct_predictions = tf.equal(self.predictions, tf.argmax(self.input_y, 1))
    self.accuracy = tf.reduce_mean(tf.cast(correct_predictions, "float"), name="accuracy")
```

Optimizer

- train.py

```
with tf.Graph().as_default():
    session_conf = tf.ConfigProto(
      allow_soft_placement=FLAGS.allow_soft_placement,
      log_device_placement=FLAGS.log_device_placement)
    sess = tf.Session(config=session_conf)
    with sess.as_default():
        cnn = TextCNN(FLAGS.flag_values_dict())

        # Define Training procedure
        global_step = tf.Variable(0, name="global_step", trainable=False)
        decayed_lr = tf.train.exponential_decay(FLAGS.lr, global_step, 1000, FLAGS.lr_decay,
staircase=True)
        optimizer = tf.train.AdamOptimizer(decayed_lr)
        grads_and_vars = optimizer.compute_gradients(cnn.loss)
        train_op = optimizer.apply_gradients(grads_and_vars, global_step=global_step)
```

Save vocabulary and FLAGS

- train.py

```
# Write vocabulary
with smart_open.smart_open(os.path.join(out_dir, "vocab"), 'wb') as f:
    pickle.dump(word_id_dict, f)
with smart_open.smart_open(os.path.join(out_dir, "config"), 'wb') as f:
    pickle.dump(FLAGS.flag_values_dict(), f)
```

CNN 분류기 평가

- cnn_eval.py

```
tf.flags.DEFINE_string("x_test_file", "./data/test/x_Trec_test.txt", "Data source for the ODP training")
tf.flags.DEFINE_string("t_test_file", "./data/test/t_Trec_test.txt", "Data source for the ODP training")
# Eval Parameters
tf.flags.DEFINE_string("dir", "./runs/1585383108", "Checkpoint directory from training run")
# Misc Parameters
tf.flags.DEFINE_boolean("allow_soft_placement", True, "Allow device soft device placement")
tf.flags.DEFINE_boolean("log_device_placement", False, "Log placement of ops on devices")
FLAGS = tf.flags.FLAGS

x_raw, y_test, _ = dh.load_data(FLAGS.x_test_file, FLAGS.t_test_file)
y_test = np.argmax(y_test, axis=1)

# Map data into vocabulary
with smart_open.smart_open(os.path.join(FLAGS.dir, "vocab"), 'rb') as f:
    word_id_dict = pickle.load(f)
with smart_open.smart_open(os.path.join(FLAGS.dir, "config"), 'rb') as f:
    config = pickle.load(f)

    x_test = dh.text_to_index(x_raw, word_id_dict, max(list(map(int, config["filter_sizes"].split(",")))) - 1)
    x_test = dh.test_tensor(x_test, config["max_length"])

print("₩nEvaluating...₩n")
```

CNN 분류기 평가

- cnn_eval.py

```
checkpoint_file = tf.train.latest_checkpoint(os.path.join(FLAGS.dir, "checkpoints"))
graph = tf.Graph()
with graph.as_default():
    session_conf = tf.ConfigProto(
      allow_soft_placement=FLAGS.allow_soft_placement,
      log_device_placement=FLAGS.log_device_placement)
    sess = tf.Session(config=session_conf)
    with sess.as_default():
        cnn = TextCNN(config)
        sess.run(tf.global_variables_initializer())
        saver = tf.train.Saver(tf.global_variables())
        saver.restore(sess, checkpoint_file)

        # Generate batches for one epoch
        batches = dh.batch_iter(list(x_test), config["batch_size"], 1, shuffle=False)

        all_predictions = []
        for x_test_batch in batches:
            batch_predictions = sess.run(cnn.predictions, {cnn.input_x: x_test_batch,
cnn.dropout_keep_prob: 1.0})
            all_predictions = np.concatenate([all_predictions, batch_predictions])
```

AI School 6기 9주차

RNN 기반 텍스트 분류기

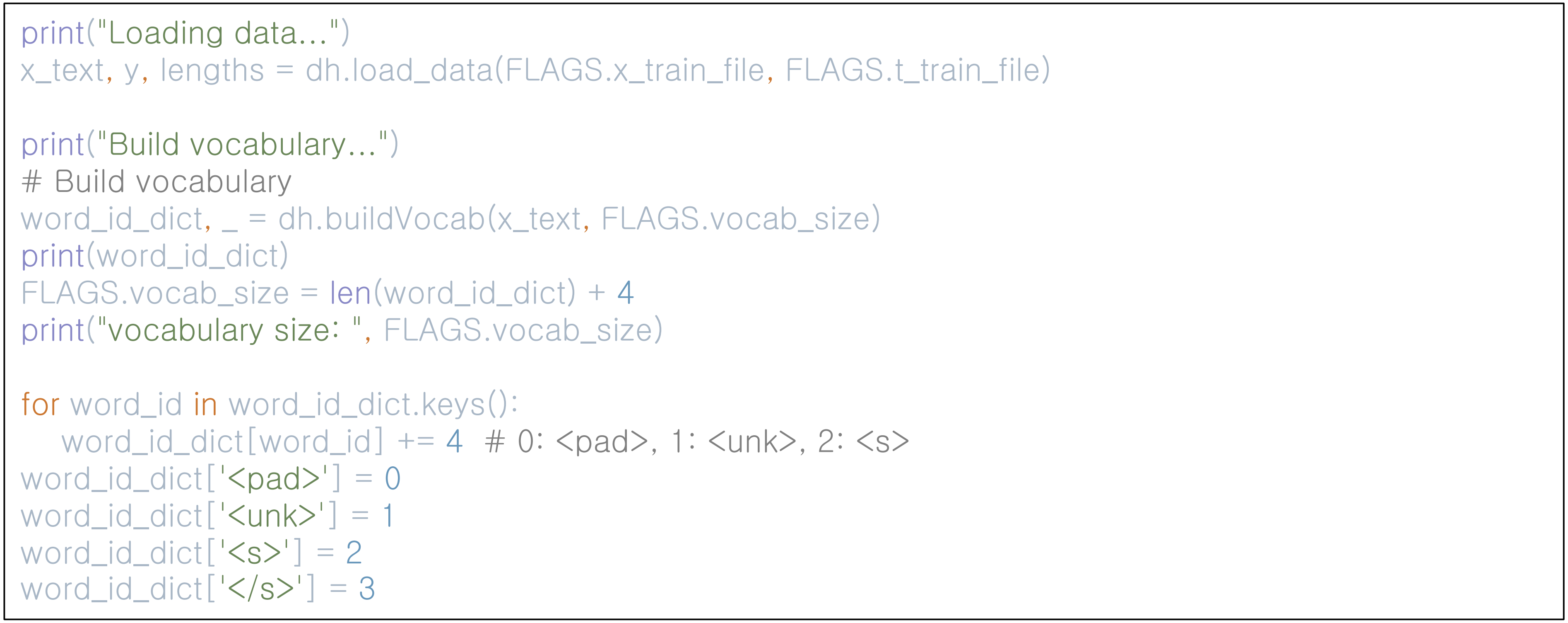
Hyperparameters

- train_rnn.py

```
tf.flags.DEFINE_float("val_sample_percentage", .1, "Percentage of the training data to use for validation")
tf.flags.DEFINE_string("x_train_file", "./data/train/x_agnewsTrain.txt", "Data source for the training")
tf.flags.DEFINE_string("t_train_file", "./data/train/t_agnewsTrain.txt", "Data source for the training")
tf.flags.DEFINE_string("word2vec", None, "Word2vec file with pre-trained embeddings (default: None)")

tf.flags.DEFINE_integer("embedding_dim", 100, "Dimensionality of word embedding (default: 128)")
tf.flags.DEFINE_string("model", "LSTM-pool", "Type of classifiers. You have three choices: [LSTM,
BiLSTM, LSTM-pool, BiLSTM-pool, ATT-LSTM, ATT-BiLSTM] (default: LSTM)")
tf.flags.DEFINE_integer("hidden_layer_num", 1, "LSTM hidden layer num (default: 1)")
tf.flags.DEFINE_integer("hidden_neural_size", 100, "LSTM hidden neural size (default: 128)")
tf.flags.DEFINE_integer("attention_size", 200, "LSTM hidden neural size (default: 128)")

tf.flags.DEFINE_float("lr", 0.001, "learning rate (default=0.001)")
tf.flags.DEFINE_float("lr_decay", 0.9, "Learning rate decay rate (default: 0.98)")
tf.flags.DEFINE_float("dropout_keep_prob", 0.5, "Dropout keep probability (default: 0.5)") #살리는 확률
tf.flags.DEFINE_float("l2_reg_lambda", 1.0e-4, "L2 regularization lambda (default: 0.0)")
tf.flags.DEFINE_integer("vocab_size", 30000, "Vocabulary size (defualt: 0)")
tf.flags.DEFINE_integer("num_classes", 0, "Number of classes (defualt: 0)")

tf.flags.DEFINE_integer("batch_size", 50, "Batch Size (default: 64)")
tf.flags.DEFINE_integer("num_epochs", 200, "Number of training epochs (default: 200)")
tf.flags.DEFINE_integer("evaluate_every", 100, "Evaluate model on dev set after this many
steps(iterations) (default: 100)")
tf.flags.DEFINE_integer("checkpoint_every", 100, "Save model after this many steps (default: 100)")
tf.flags.DEFINE_integer("num_checkpoints", 3, "Number of checkpoints to store (default: 5)")
```

Data loading & preprocessing

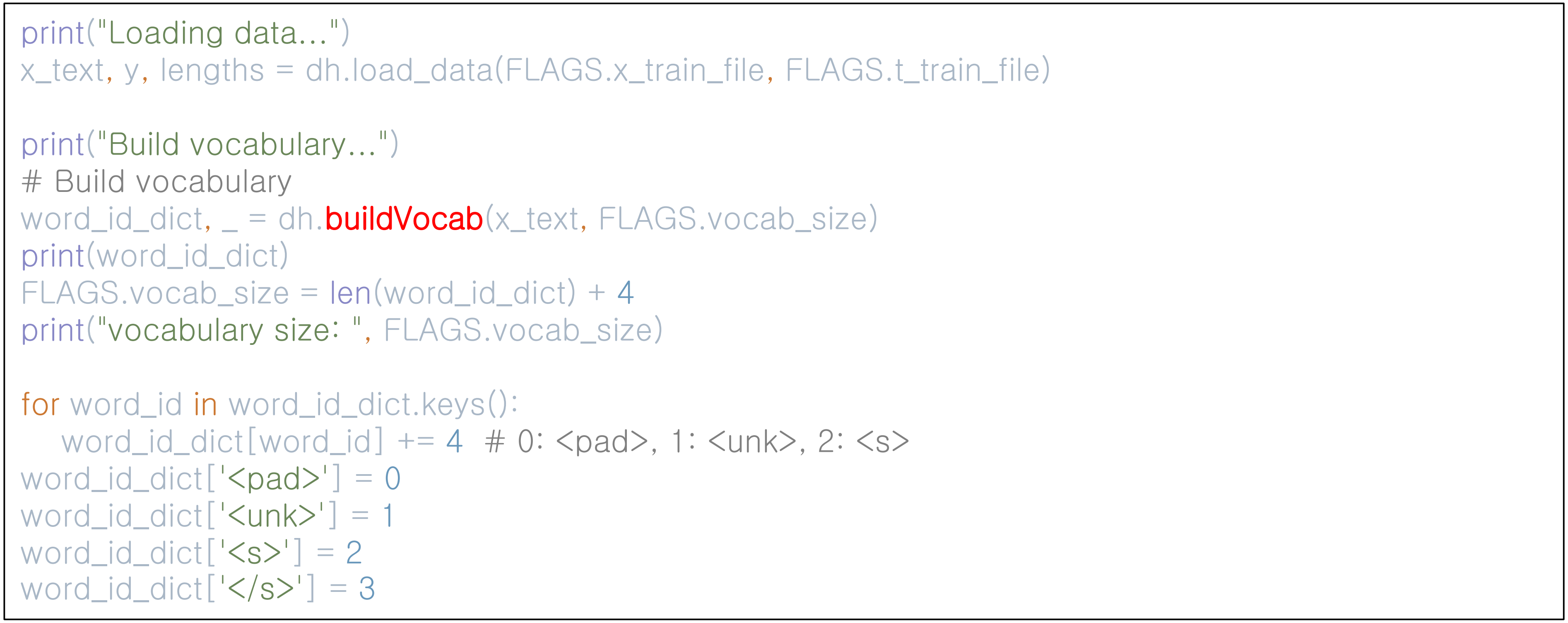
- train.py



```
print("Loading data...")
x_text, y, lengths = dh.load_data(FLAGS.x_train_file, FLAGS.t_train_file)

print("Build vocabulary...")
# Build vocabulary
word_id_dict, _ = dh.buildVocab(x_text, FLAGS.vocab_size)
print(word_id_dict)
FLAGS.vocab_size = len(word_id_dict) + 4
print("vocabulary size: ", FLAGS.vocab_size)

for word_id in word_id_dict.keys():
    word_id_dict[word_id] += 4  # 0: <pad>, 1: <unk>, 2: <s>
word_id_dict['<pad>'] = 0
word_id_dict['<unk>'] = 1
word_id_dict['<s>'] = 2
word_id_dict['</s>'] = 3
```

Data loading & preprocessing

- train.py



```
print("Loading data...")
x_text, y, lengths = dh.load_data(FLAGS.x_train_file, FLAGS.t_train_file)

print("Build vocabulary...")
# Build vocabulary
word_id_dict, _ = dh.buildVocab(x_text, FLAGS.vocab_size)
print(word_id_dict)
FLAGS.vocab_size = len(word_id_dict) + 4
print("vocabulary size: ", FLAGS.vocab_size)

for word_id in word_id_dict.keys():
    word_id_dict[word_id] += 4  # 0: <pad>, 1: <unk>, 2: <s>
word_id_dict['<pad>'] = 0
word_id_dict['<unk>'] = 1
word_id_dict['<s>'] = 2
word_id_dict['</s>'] = 3
```

Data loading & preprocessing

- train.py

```
np.random.seed(10)
shuffle_indices = np.random.permutation(np.arange(len(y)))
x_text = x_text[shuffle_indices]
print("Split train/validation set...")
val_sample_index = -1 * int(FLAGS.val_sample_percentage * float(len(y)))
x_train, x_val = x_text[:val_sample_index], x_text[val_sample_index:]

x_train = dh.text_to_index(x_train, word_id_dict, 0)
x_val = dh.text_to_index(x_val, word_id_dict, 0)

FLAGS.num_classes = y.shape[1]

y = y[shuffle_indices]
lengths = lengths[shuffle_indices]

y_train, y_val = y[:val_sample_index], y[val_sample_index:]
lengths, lengths_val = lengths[:val_sample_index], lengths[val_sample_index:]

print("Vocabulary Size: {:d}".format(FLAGS.vocab_size))
print("Train/Val split: {:d}/{:d}".format(len(y_train), len(y_val)))
return x_train, y_train, lengths, word_id_dict, x_val, y_val, lengths_val
```

TextRNN class & input

- text_rnn.py

```
class TextRNN(object):

    def __init__(self, config):
        self.num_classes = config["num_classes"] # e.g., positive, negatie - 2
        self.vocab_size = config["vocab_size"]
        self.hidden_size = config["hidden_neural_size"]
        self.attention_size = config["attention_size"]
        self.embedding_dim = config["embedding_dim"] # word vector size
        self.num_layers = config["hidden_layer_num"] #
        self.l2_reg_lambda = config["l2_reg_lambda"]

        self.batch_size = tf.placeholder(tf.int32, shape=(), name="batch_size")
        self.input_x = tf.placeholder(tf.int32, [None, None], name="input_x")
        self.input_y = tf.placeholder(tf.float32, [None, self.num_classes], name="input_y")
        self.dropout_keep_prob = tf.placeholder(tf.float32, name="dropout_keep_prob")
        self.sequence_length = tf.placeholder(tf.int32, [None], name="sequence_length")

        self.l2_loss = tf.constant(0.0)
```

Embedding layer

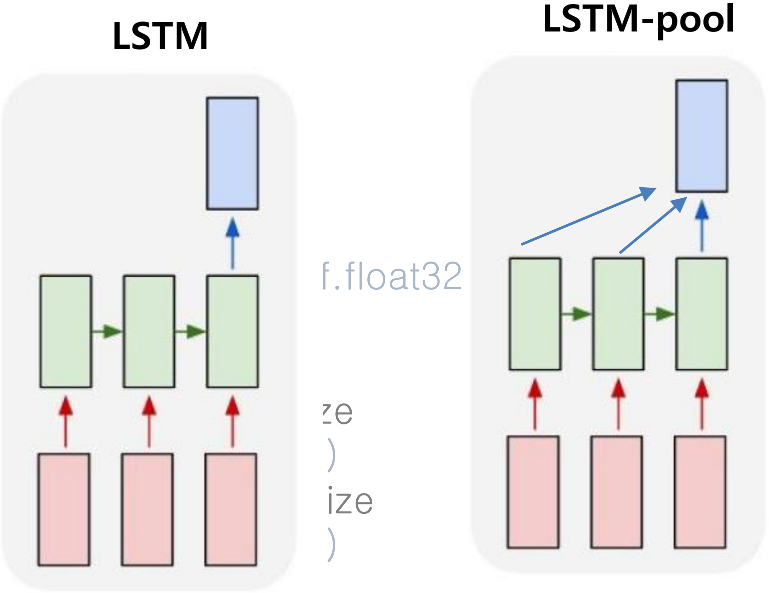
- text_rnn.py

```
# Embedding layer
with tf.device('/gpu:0'), tf.name_scope("embedding"):
    self.W = tf.Variable(tf.random_uniform([self.vocab_size, self.embedding_dim], -1.0, 1.0),
trainable=True, name="W")
    self.inputs = tf.nn.embedding_lookup(self.W, self.input_x)
```

Embedding layer

- text_rnn.py

```
if config["model"] == "LSTM":
    _, self.final_state = self.normal_lstm()
elif config["model"] == "LSTM-pool":
    output, _ = self.normal_lstm()
    masks = tf.sequence_mask(lengths=self.sequence_length,
                             maxlen=tf.reduce_max(self.sequence_length), dtype=tf.float32, name='masks')
    output = output * tf.expand_dims(masks, -1)
    self.final_state = tf.div(tf.reduce_sum(output, 1), tf.expand_dims(tf.cast(self.sequence_length,
tf.float32), 1))
elif config["model"] == "BiLSTM":
    _, self.final_state = self.bi_lstm()
elif config["model"] == "BiLSTM-pool":
    output, _ = self.bi_lstm()
    masks = tf.sequence_mask(lengths=self.sequence_le
                             maxlen=tf.reduce_max(self.sequenc                f.float32
    output_fw = output[0] * tf.expand_dims(masks, -1)
    output_bw = output[1] * tf.expand_dims(masks, -1)
    output_fw = tf.div(tf.reduce_sum(output_fw, 1), # bat                     ze
                       tf.expand_dims(tf.cast(self.sequence_le                )
    output_bw = tf.div(tf.reduce_sum(output_bw, 1), # ba                      ize
                       tf.expand_dims(tf.cast(self.sequence_le                )
    self.final_state = tf.concat([output_fw, output_bw], 1)
```



Q&A