


Natural Language Processing With Deep Learning



Minho Ryu

MetaFact CTO/ IDS Lab

본 교안은 삼성전자 SR AI 입문교육을 위해 제작되었으며, 교육목적 외 사용을 금합니다.

Minho Ryu

Work Experience

- (現) Machine Learning Engineer in MetaFact
- (現) Graduate Researcher at Hanyang University
- IT Instructor at TRCSL in Sri-Lanka
- Programming Instructor for high school students

Education

- Bachelors in Industrial Engineering & Mathematics at Hanyang University
- Artificial Intelligence and Machine Learning (KMOOC - KCS470)
- Introduction to Big Data (KMOOC - CSED490k)
- Natural Language Processing with Deep Learning (Stanford - CS224N)



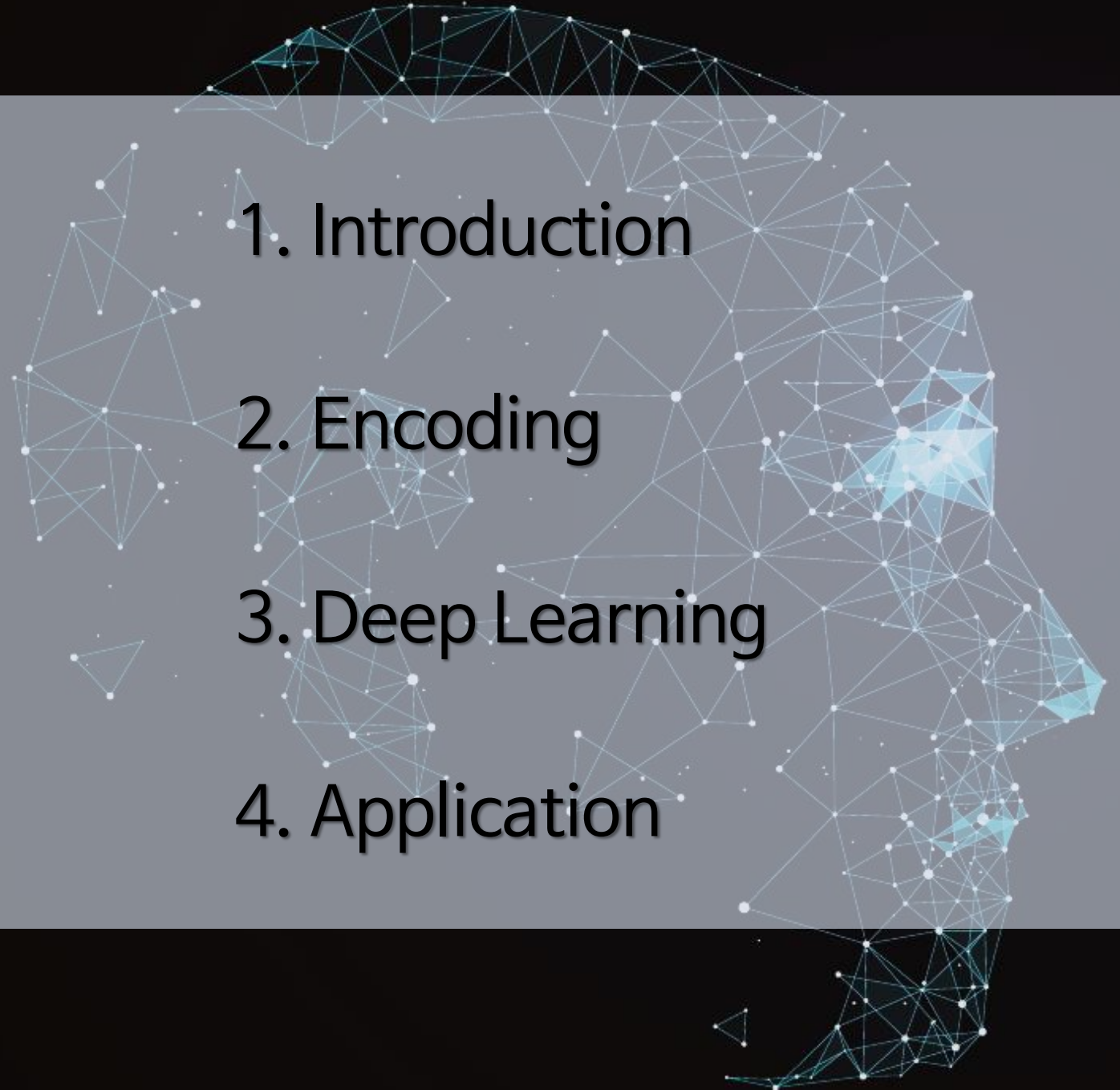
Index

1. Introduction

2. Encoding

3. Deep Learning

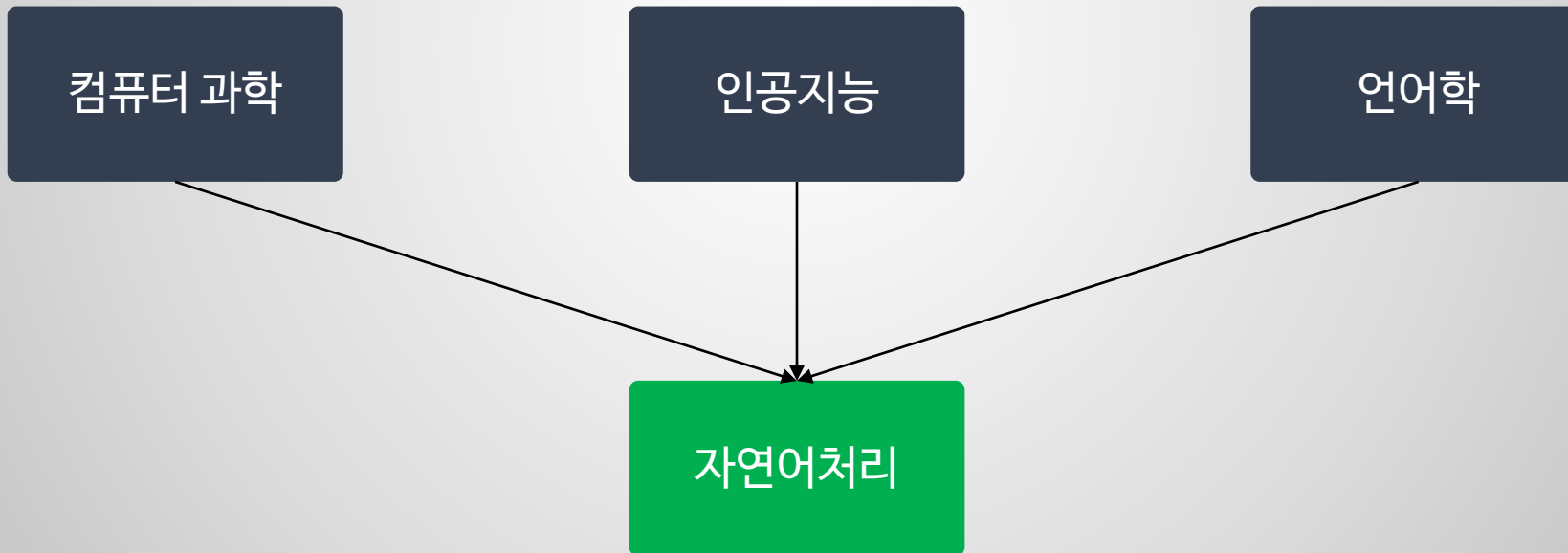
4. Application



1. Introduction

자연어처리 (Natural Language Processing)

사람의 말을 컴퓨터를 이용하여 처리하기 위해 컴퓨터과학, 인공지능 및 언어학이 합쳐진 분야



1. Introduction

“구글, 사람처럼 전화 통화하는 인공지능 공개”

- 구글 인공지능과 실제 헤어샵, 레스토랑 직원 사이 대화를 녹음
- 실제 사람과 구분이 가지 않을 정도로 매끄러운 목소리로 대화
- 예약 전화를 받고 있는 헤어샵 직원은 상대방이 인공지능인 것을 전혀 눈치 채지 못함



<https://www.youtube.com/watch?v=pci3IMxFk4M>

1. Introduction

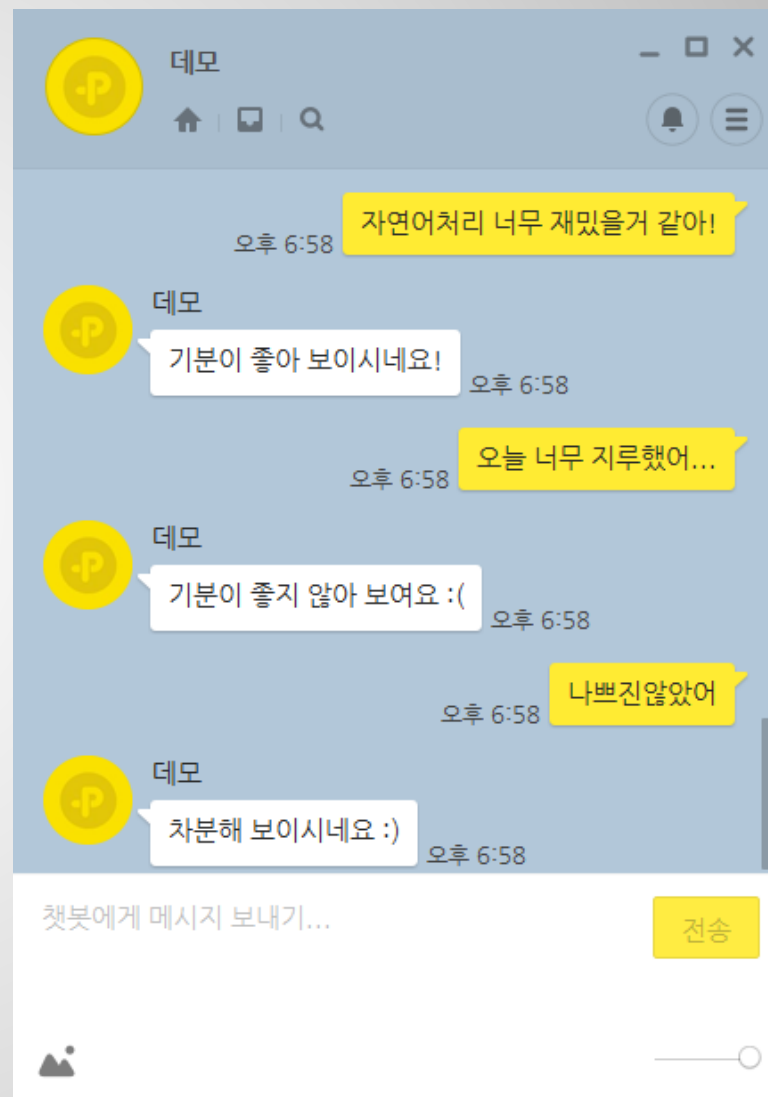


자연어처리의 대표적인 task

감성분석

주어진 글을 보고 감정을 파악하는 것

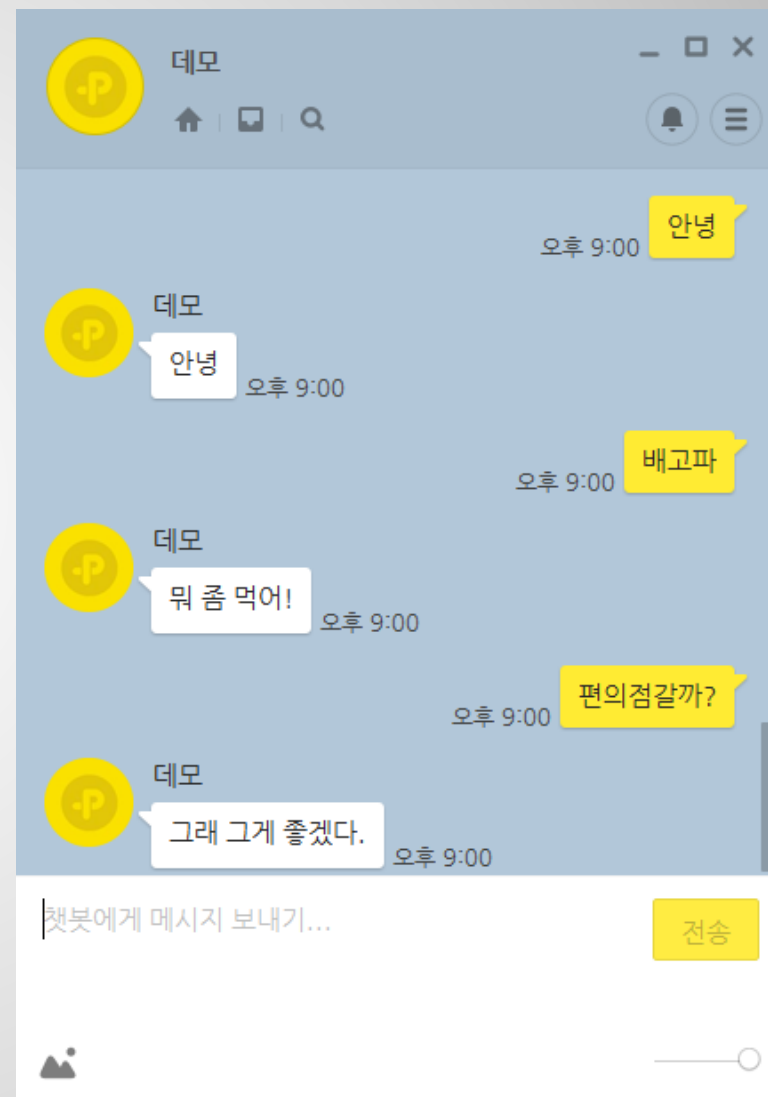
→ 긍정, 부정 (또는 중립)을 파악



자연어처리의 대표적인 task

대화

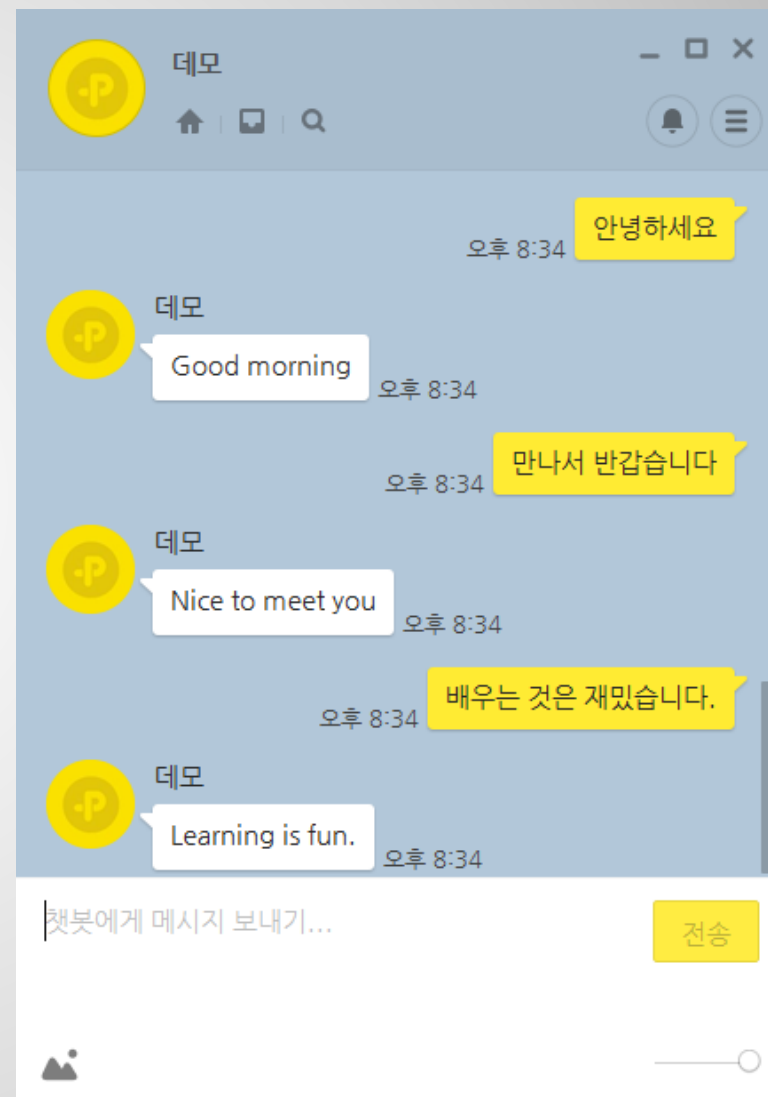
상대방의 글을 보고 적절한 대답을 하는 것



자연어처리의 대표적인 task

번역

주어진 문장을 다른 언어로 번역하는 것



구름 IDE 접속하기

<https://ide.goorm.io>

- FREE Plan 시작하기



가볍게 시작하세요

코딩/빌드/컴파일/공유가 가능한
나만의 컨테이너를 만들어보세요.

무료

시작하기

구름 IDE 컨테이너 생성



1. Introduction - jupyter setting

구름 IDE 컨테이너 생성

<https://github.com/Bricoler/nlp-tensorflow.git>

컨테이너 생성

소스	<input type="radio"/> 템플릿	<input type="radio"/> Github	<input type="radio"/> Bitbucket	<input checked="" type="radio"/> Git / SVN	<input type="radio"/> 압축파일
유형	<input checked="" type="radio"/> Git <input type="radio"/> SVN				
저장소 URL	<input type="text" value="https://github.com/Bricoler/nlp-tensorflow.git"/>				소스 트리 보기
리비전	<input checked="" type="radio"/> HEAD <input type="radio"/> <input type="text" value="e.g. 5c75fec"/>				
인증	<input checked="" type="radio"/> 익명 <input type="radio"/> 인증된 사용자				
<div>TEST</div>					



1. Introduction - jupyter setting

구름 IDE 컨테이너 생성

이름	<input type="text" value="nlp-tensorflow"/>
설명	<input type="text" value="natural language processing with deep learning"/>
소프트웨어 스택 선택	<div><div><div>Search</div><div><div> C/C++</div><div> Python</div><div> Django</div><div> Flask</div><div> Jupyter Notebook</div><div> TensorFlow</div><div> PyQt</div><div> JAVA</div><div> Maven</div><div> Spring</div></div></div><div><div>TensorFlow</div><div>프로젝트 유형</div><div>TensorFlow 프로젝트 (Jupyter Notebook)</div><div><div>OS</div><div>✓ Ubuntu 14.04 LTS</div><div>Python3</div><div>✓ 3.6.5</div><div>Python</div><div>✓ 2.7.6</div><div>pip3</div><div>✓ 9.0.1</div><div>pip</div><div>✓ 9.0.1</div><div>Jupyter</div><div>✓ 4.3.0</div><div>Django</div><div>✓ 1.11.12 LTS</div><div>Flask</div><div>✓ 0.12</div><div>TensorFlow</div><div>✓ 1.3.0</div></div></div></div>
<div><div>취소</div><div>생성하기</div></div>	

구름 IDE 컨테이너 실행

nlp-tensorflow

설정

No description

소프트웨어 스택

TensorFlow 프로젝트 (Jupyter N...

저장 공간

10GB

마지막 실행 시간

2018. 7. 2. 오전 10:32:35

SSH

컨테이너 실행 시 사용 가능

공유 링크

<https://goor.me/D1vR>

테마

☐ Light ☒ Dark

항상 켜두기

PREMIUM 상품 구매 시 사용가능

실행

터미널 실행

konlpy, gensim 설치하기

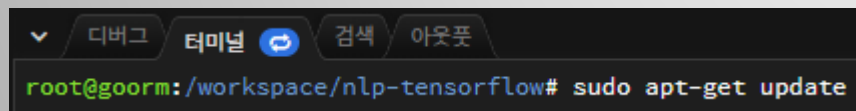
터미널에 아래 명령어 순서대로 입력

```
$ sudo apt-get update
```

```
$ sudo apt-get install g++ openjdk-7-jdk python-dev python3-dev
```

```
$ pip3 install JPy1-py3
```

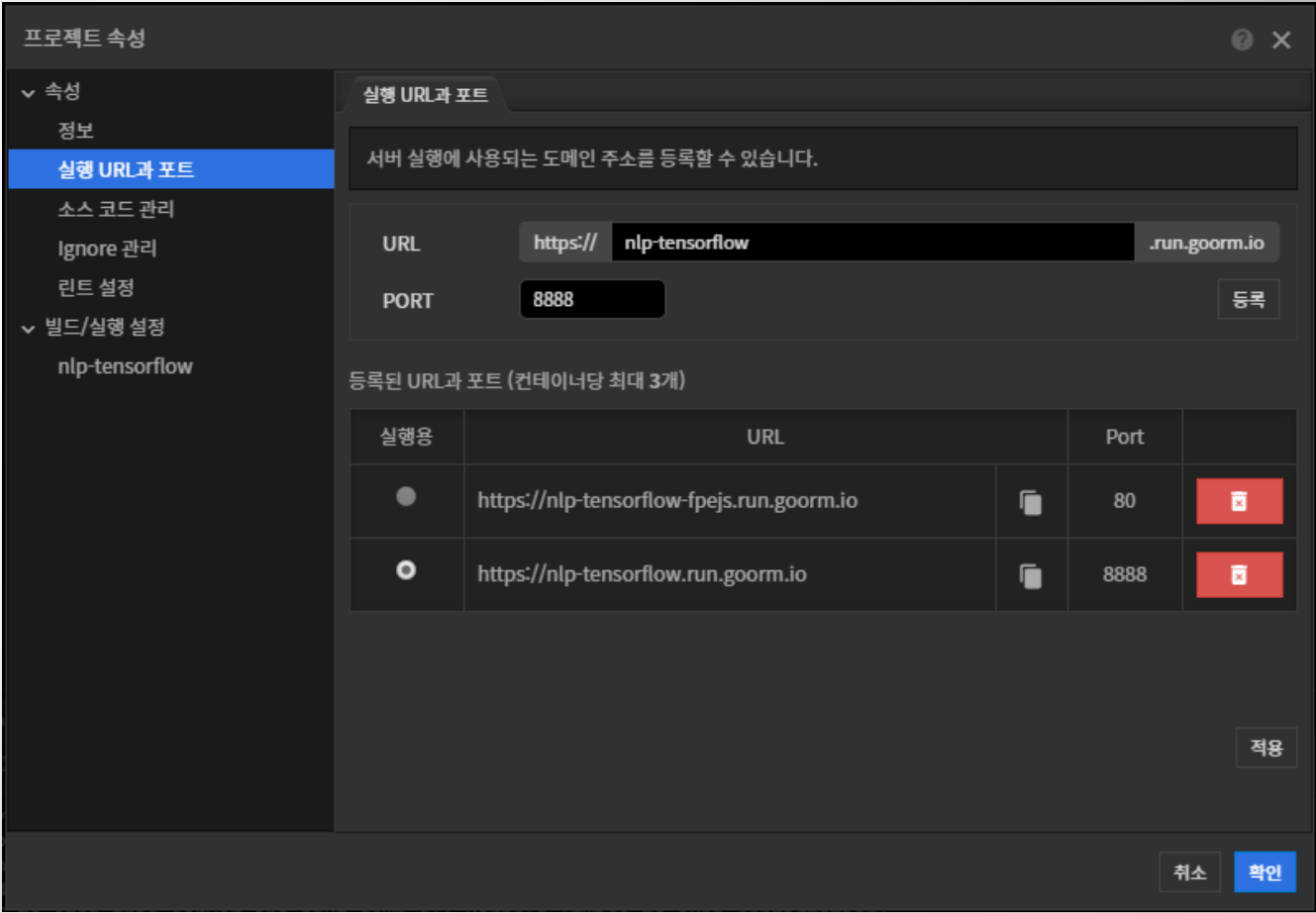
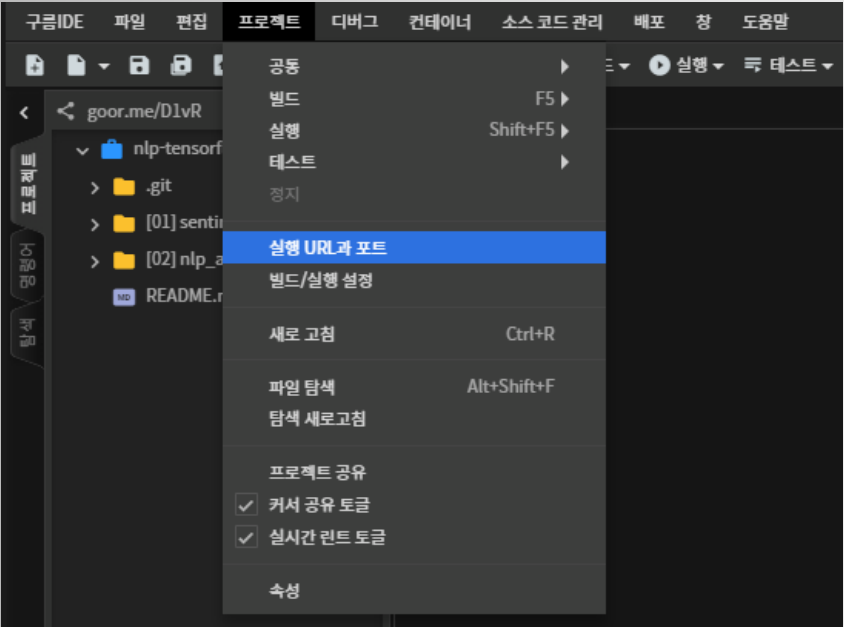
```
$ pip3 install konlpy gensim
```

A screenshot of a terminal window from the Goorm IDE. The terminal title bar shows '디버그' (Debug), '터미널' (Terminal), '검색' (Search), and '아웃풋' (Output). The terminal content shows the prompt 'root@goorm: /workspace/nlp-tensorflow#' followed by the command 'sudo apt-get update' which has been executed.

```
root@goorm: /workspace/nlp-tensorflow# sudo apt-get update
```

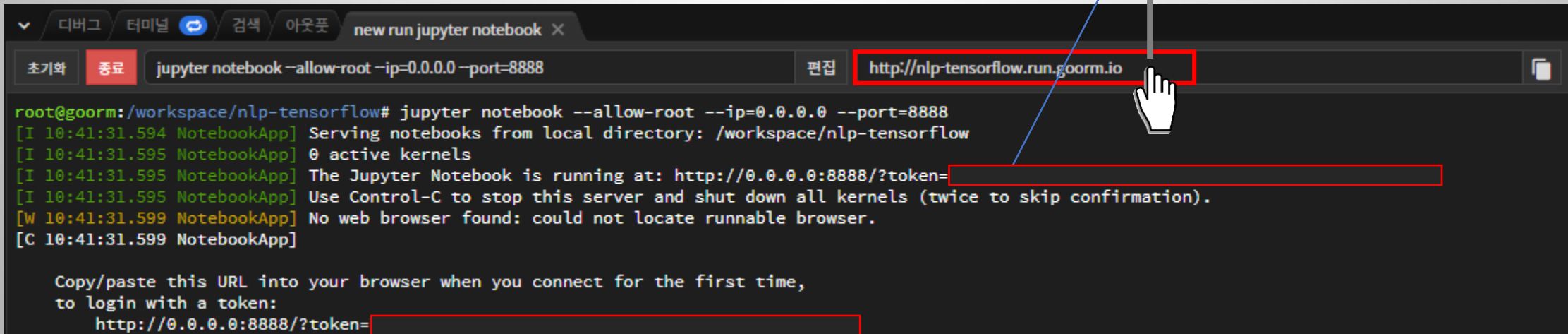
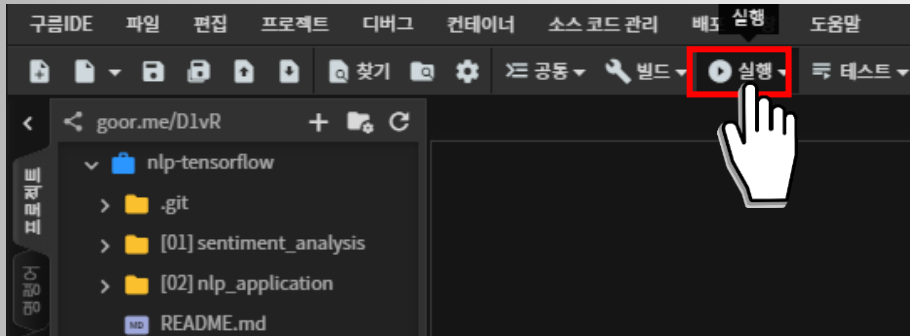
1. Introduction - jupyter setting

실행 URL과 포트에서 도메인 등록 및 확인



1. Introduction - jupyter setting

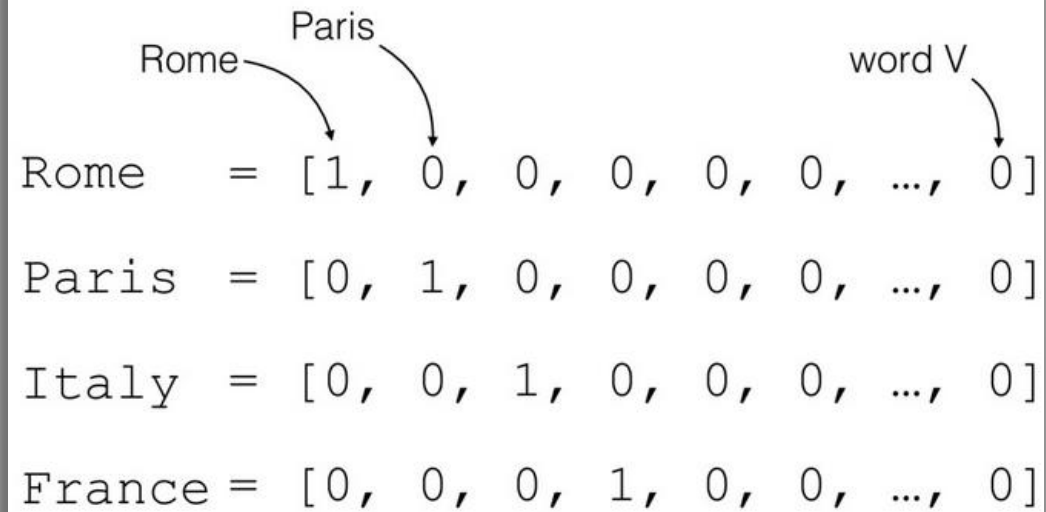
실행 및 URL 접속 후 token 입력



2. Encoding

2. Encoding - onehot encoding

one-hot encoding: 각 단어를 하나만 1이고 나머지가 모두 0인 벡터로 표현



The diagram illustrates one-hot encoding for a set of words. Each word is represented by a vector of length V , where only the index corresponding to the word is set to 1, and all other indices are 0. The words and their corresponding vectors are:

- Rome = $[1, 0, 0, 0, 0, 0, \dots, 0]$
- Paris = $[0, 1, 0, 0, 0, 0, \dots, 0]$
- Italy = $[0, 0, 1, 0, 0, 0, \dots, 0]$
- France = $[0, 0, 0, 1, 0, 0, \dots, 0]$

Arrows indicate the mapping: 'Rome' points to the first element (1) of its vector, 'Paris' points to the second element (1) of its vector, and 'word V' points to the last element (0) of the first vector.

2. Encoding - onehot encoding

one-hot encoding: 각 단어를 하나만 1이고 나머지가 모두 0인 벡터로 표현

```
import re
import numpy as np
```

```
data = ['나는 생각한다 고로 나는 존재한다.',
        '모든 국가는 그에 걸맞은 정부를 가진다.',
        '이것 또한 지나가리라',
        '죄는 미워하되 사람은 미워하지 말라']
```

```
def tokenizer(sentence):
    tokens = re.findall(r"[\w]+|^[^\s\w]", sentence)
    return tokens
```

```
tokenizer('모든% 특수기호를:) *분리해 준다.')
```

```
['모든', '%', '특수기호를', ':', ')', '*', '분리해', '준  
다', '.']
```

```
print(np.eye(vocab_size)[vocab['나는']])
```

```
[1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

```
vocab = dict()
index = 0
for line in data:
    tokens = tokenizer(line)
    for token in tokens:
        if token in vocab:
            continue
        else:
            vocab[token] = index
            index += 1
```

```
vocab_size = len(vocab)
```

```
print(vocab.items())
```

```
dict_items([('나는', 0), ('생각한다', 1), ('고로', 2),
            ('존재한다', 3), ('.', 4), ('모든', 5), ('국가는', 6),
            ('그에', 7), ('걸맞은', 8), ('정부를', 9), ('가진다', 10),
            ('이것', 11), ('또한', 12), ('지나가리라', 13), ('죄는', 14),
            ('미워하되', 15), ('사람은', 16), ('미워하지', 17),
            ('말라', 18)])
```

2. Encoding - onehot encoding

one-hot encoding 을 이용하여 문장을 벡터로 표현

```
print(tokenizer(data[0]))
```

```
['나는', '생각한다', '고로', '나는', '존재한다', '.']
```

```
print(tokenizer(data[1]))
```

```
['모든', '국가는', '그에', '걸맞은', '정부를', '가진다', '.']
```

```
print(tokenizer(data[2]))
```

```
['이것', '또한', '지나가리라']
```

```
print(tokenizer(data[3]))
```

```
['죄는', '미워하되', '사람은', '미워하지', '말라']
```

```
print(np.sum(np.eye(vocab_size)[vocab[token]]  
             for token in set(tokenizer(data[0]))))
```

```
[1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

```
print(np.sum(np.eye(vocab_size)[vocab[token]]  
             for token in set(tokenizer(data[1]))))
```

```
[0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0.]
```

```
print(np.sum(np.eye(vocab_size)[vocab[token]]  
             for token in set(tokenizer(data[2]))))
```

```
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 0. 0. 0. 0.]
```

```
print(np.sum(np.eye(vocab_size)[vocab[token]]  
             for token in set(tokenizer(data[3]))))
```

```
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1.]
```

2. Encoding - tfidf encoding

tf-idf encoding: 단어의 문장 내 빈도 및 문서 내 빈도를 고려하여 단어의 중요도를 반영

*tf(단어 빈도, term frequency): 특정 단어가 문서 내에 얼마나 자주 등장하는 지 나타내는 값

*df(문서 빈도, document frequency): 단어가 문서군 내에서 얼마나 자주 등장하는 지 나타내는 값

$$tf(t, d) = 0.5 + \frac{0.5 \times f(t, d)}{\max\{f(w, d): w \in d\}}$$

$$idf(t, D) = \log\left(\frac{|D|}{1 + |\{d \in D: t \in d\}|}\right)$$

$|\{d \in D: t \in d\}|$: the number of documents including word t

$$tfidf(t, d, D) = tf(t, d) \times idf(t, D)$$

특정 문서 내에서 단어 빈도가 높을 수록, 그리고

전체 문서들 중 그 단어를 포함한 문서가 적을수록

tf-idf 값이 높아지므로 이 값을 이용하면 모든 문

서에 흔하게 나타나는 단어를 걸러내는 효과

2. Encoding - tfidf encoding

tf-idf encoding: 단어의 문장 내 빈도 및 문서 내 빈도를 고려하여 단어의 중요도를 반영

```
def cal_idf(lines, vocab):  
    vocab_size = len(vocab)  
    doc_size = len(lines)  
    DF = np.zeros(vocab_size)  
    for line in lines:  
        tokens = tokenizer(line)  
        tokens = set(tokens)  
        for token in tokens:  
            if token in vocab.keys():  
                DF[vocab[token]] += 1  
    IDF = np.log(doc_size/(1 + DF))  
  
    return IDF
```

```
IDF = cal_idf(data, vocab)
```

```
print(IDF)
```

```
[0.69314718 0.69314718 0.69314718 0.69314718 0.28768207 0.  
69314718  
0.69314718 0.69314718 0.69314718 0.69314718 0.69314718 0.  
69314718  
0.69314718 0.69314718 0.69314718 0.69314718 0.69314718 0.  
69314718  
0.69314718]
```

2. Encoding - tfidf encoding

```
def sentence_to_tfidf(lines, vocab, IDF):
    vocab_size = len(vocab)
    doc_size = len(lines)

    tf_idfs = []
    for line in lines:
        tokens = tokenizer(line)
        freq = dict()
        TF = np.zeros(vocab_size, dtype=float)
        for token in tokens:
            if token in vocab.keys():
                if token in freq.keys():
                    freq[token] += 1
                else:
                    freq[token] = 1
        if len(freq) == 0:
            max_tf = 0
        else:
            max_tf = max(freq.values())
        tokens = set(tokens)
        for token in tokens:
            if token in vocab.keys():
                TF[vocab[token]] = 0.5 + 0.5 *
                    freq[token] / max_tf
        tf_idf = np.multiply(TF, IDF)
        tf_idfs.append(tf_idf)
    tf_idfs = np.asarray(tf_idfs)

    return tf_idfs
```

```
sentence_to_tfidf(data, vocab, IDF)
```

```
array([[0.69314718, 0.51986039, 0.51986039, 0.51986039, 0.
21576155,
        0.          , 0.          , 0.          , 0.          , 0.
        ,
        0.          , 0.          , 0.          , 0.          , 0.
        ,
        0.          , 0.          , 0.          , 0.          ],
       [0.          , 0.          , 0.          , 0.          , 0.
28768207,
        0.69314718, 0.69314718, 0.69314718, 0.69314718, 0.
69314718,
        0.69314718, 0.          , 0.          , 0.          , 0.
        ,
        0.          , 0.          , 0.          , 0.          ],
       [0.          , 0.          , 0.          , 0.          , 0.
        ,
        0.          , 0.          , 0.          , 0.          , 0.
        ,
        0.          , 0.69314718, 0.69314718, 0.69314718, 0.
        ,
        0.          , 0.          , 0.          , 0.          ],
       [0.          , 0.          , 0.          , 0.          , 0.
        ,
        0.          , 0.          , 0.          , 0.          , 0.
        ,
        0.          , 0.          , 0.          , 0.          ],
       [0.69314718,
        0.69314718, 0.69314718, 0.69314718]])
```

2. Encoding - vocabulary 구성

단어의 빈도 수를 고려한 vocab 구성

1. 각 문장을 토큰화
2. 각 토큰의 개수를 Count!
3. 자주 나오는 단어 순으로 vocab 구성

```
def build_vocab(lines, max_vocab=None):
    word_counter = Counter()
    vocab = dict()
    reverse_vocab = dict()

    for line in lines:
        tokens = tokenizer(line)
        word_counter.update(tokens)

    vocab['<PAD>'] = 0
    vocab['<UNK>'] = 1
    vocab_idx = 2

    if max_vocab > len(word_counter) or max_vocab == None:
        max_vocab = len(word_counter)

    for key, value in word_counter.most_common(max_vocab):
        vocab[key] = vocab_idx
        vocab_idx += 1

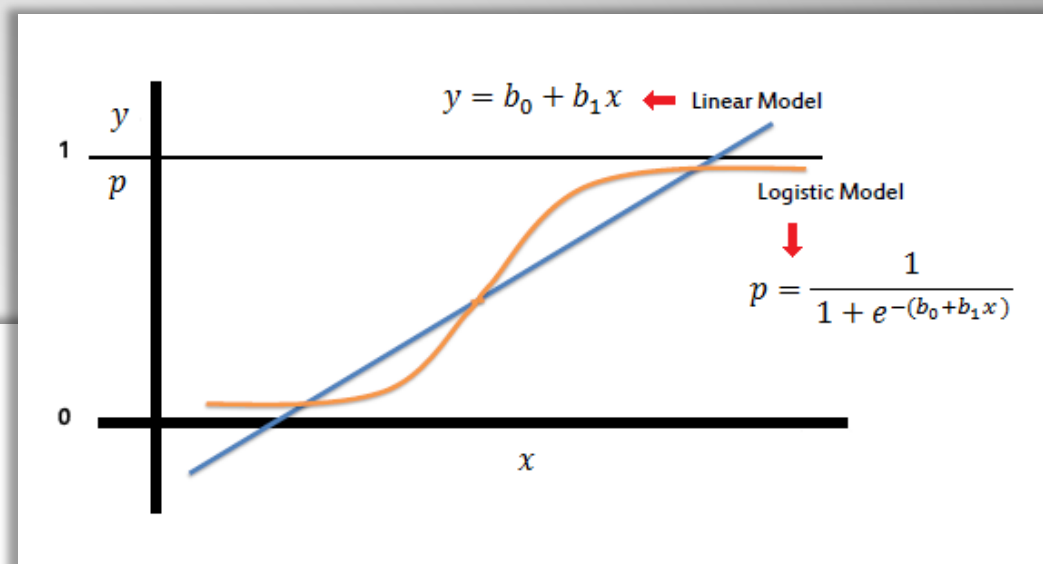
    for key, value in vocab.items():
        reverse_vocab[value] = key

    vocab_size = len(vocab.keys())

    return vocab, reverse_vocab, vocab_size
```


Logistic Regression

```
with tf.variable_scope("placeholder"):  
    self.input_x = tf.placeholder(tf.float32,  
                                  shape=(None, self.vocab_size))  
    self.input_y = tf.placeholder(tf.int32, shape=(None,))  
    Y_one_hot = tf.one_hot(self.input_y, self.n_class)  
  
with tf.variable_scope("output", reuse=tf.AUTO_REUSE):  
    W = tf.get_variable('W', dtype=tf.float32,  
                        initializer=tf.truncated_normal(  
                            (self.vocab_size, self.n_class)))  
    b = tf.get_variable('b', dtype=tf.float32,  
                        initializer=tf.constant(0.1, shape=(self.n_class,)))  
    logits = tf.nn.xw_plus_b(self.input_x, W, b)  
    self.prob = tf.reduce_max(tf.nn.softmax(logits), axis=1)  
    self.prediction = tf.cast(tf.argmax(logits, axis=1), tf.int32)
```

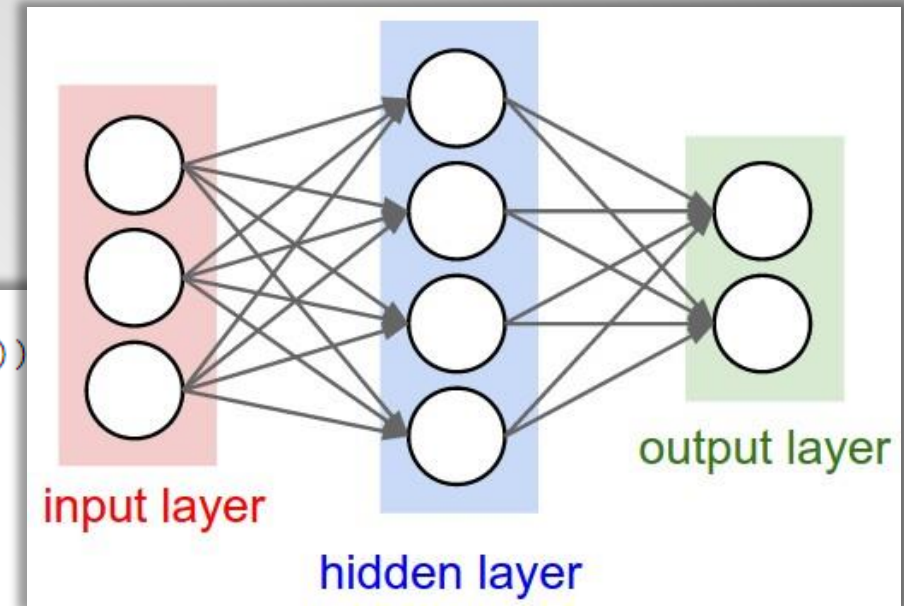


2. Encoding - Feed Forward Neural Network (FFNN)

Three Layer Neural Network

```
with tf.variable_scope("placeholder"):
    self.input_x = tf.placeholder(tf.float32, shape=(None, self.vocab_size))
    self.input_y = tf.placeholder(tf.int32, shape=(None,))

with tf.variable_scope("output", reuse=tf.AUTO_REUSE):
    W1 = tf.get_variable("W1", dtype=tf.float32,
                        initializer=tf.truncated_normal(
                            (self.vocab_size, self.hidden_size)))
    b1 = tf.get_variable("b1", dtype=tf.float32,
                        initializer=tf.constant(0.1,
                            shape=(self.hidden_size,)))
    W2 = tf.get_variable("W2", dtype=tf.float32,
                        initializer=tf.truncated_normal(
                            (self.hidden_size, self.n_class)))
    b2 = tf.get_variable("b2", dtype=tf.float32,
                        initializer=tf.constant(0.1, shape=(self.n_class,)))
    h = tf.nn.relu(tf.nn.xw_plus_b(self.input_x, W1, b1))
    logits = tf.nn.xw_plus_b(h, W2, b2)
    self.prob = tf.reduce_max(tf.nn.softmax(logits), axis=1)
    self.prediction = tf.cast(tf.argmax(logits, axis=1), tf.int32)
```



Let's Practice!

3. Deep Learning

3. Deep Learning - RNN

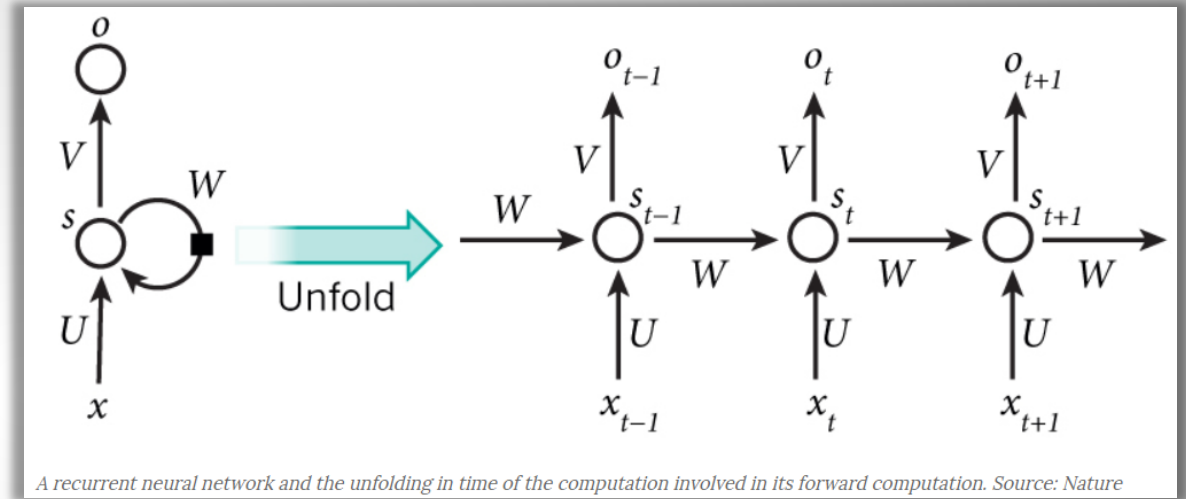
RNN

한 단어(글자)씩 순서대로 데이터를 받아 정보를 처리!

$$h_t = \tanh(U \cdot x_t + W \cdot h_{t-1} + b_h)$$

$$\hat{y}_t = o_t = \text{softmax}(V \cdot h_t + b_o)$$

문장을 한 번에 벡터화하여 입력받는 기존 방법 (로지스틱 회귀, FFNN 등)과 차이 → 문장 순서 고려 가능!



```
with tf.variable_scope("recurrent"):  
    cell = tf.nn.rnn_cell.BasicRNNCell(self.hidden_size)  
    _, states = tf.nn.dynamic_rnn(cell,  
                                  embedded_input_x,  
                                  input_length,  
                                  dtype=tf.float32)
```

3. Deep Learning - RNN

Bidirectional RNN

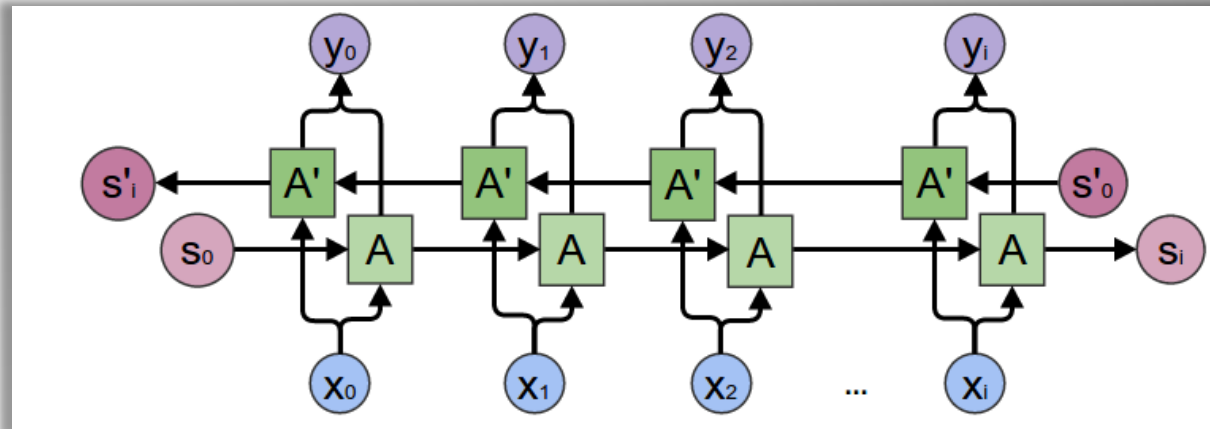
앞에서 뒤로 읽고, 뒤에서 앞으로도 읽기!

양 방향으로 RNN을 진행함으로써 앞, 뒤 맥락을 모두 고려하여 RNN의 capability 향상

$$\vec{h}_t = \tanh(\vec{U} \cdot x_t + \vec{W} \cdot \vec{h}_{t-1} + \vec{b})$$

$$\overleftarrow{h}_t = \tanh(\overleftarrow{U} \cdot x_t + \overleftarrow{W} \cdot \overleftarrow{h}_{t+1} + \overleftarrow{b})$$

$$\hat{y}_t = \text{softmax}(V \cdot [\vec{h}_t, \overleftarrow{h}_t] + b_o)$$

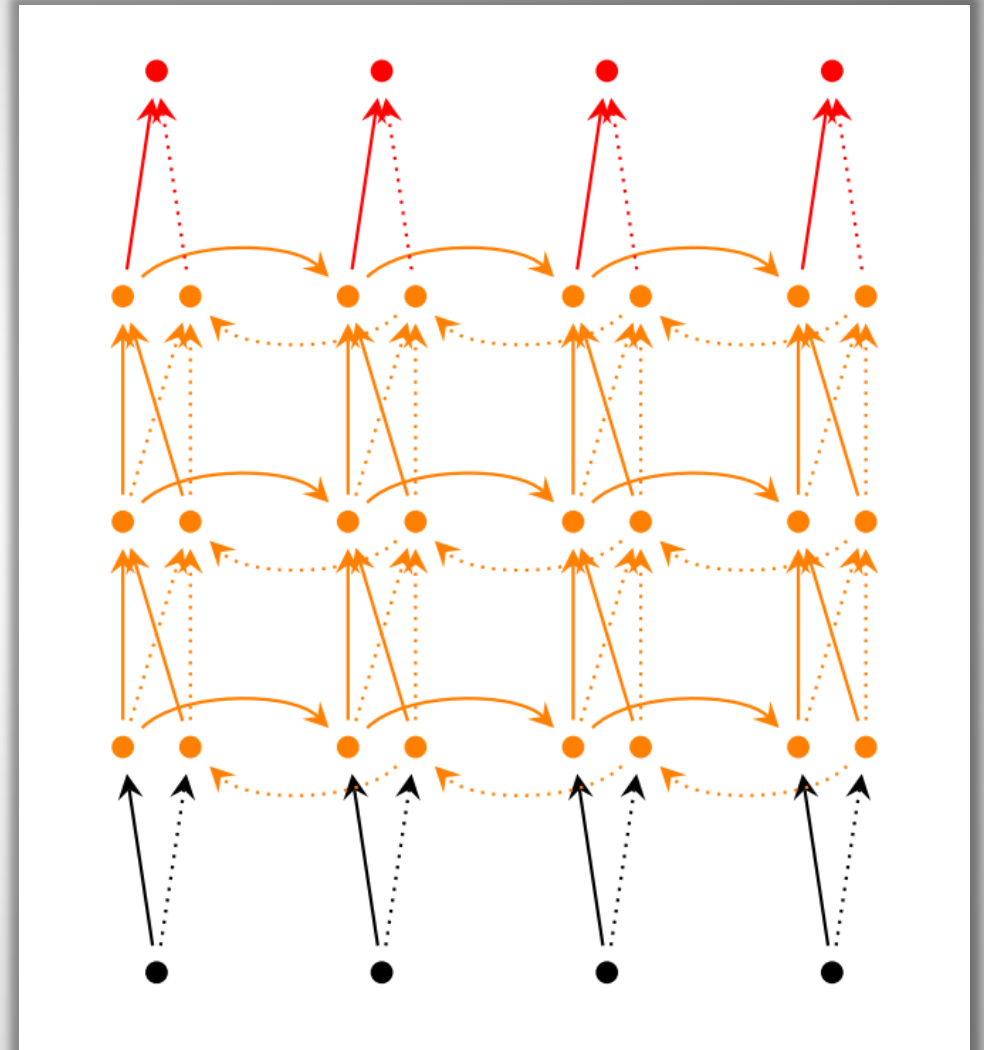


```
with tf.variable_scope("recurrent"):
    fw_cell = tf.nn.rnn_cell.BasicRNNCell(self.hidden_size)
    bw_cell = tf.nn.rnn_cell.BasicRNNCell(self.hidden_size)
    (_, _),
    (fw_states, bw_states)) =
    tf.nn.bidirectional_dynamic_rnn(fw_cell,
                                    bw_cell,
                                    embedded_input_x,
                                    input_length,
                                    dtype=tf.float32)
```


Deep bidirectional RNN

RNN에서도 hidden layer 층을 더 깊게 쌓음으로써 complexity가 더 높은 데이터에 대해서도 확률분포를 근사할 수 있다.

```
with tf.variable_scope("recurrent"):  
    fw_multi_cell = tf.nn.rnn_cell.MultiRNNCell([  
        tf.nn.rnn_cell.BasicRNNCell(  
            self.hidden_size) for _ in range(3)])  
    bw_multi_cell = tf.nn.rnn_cell.MultiRNNCell([  
        tf.nn.rnn_cell.BasicRNNCell(  
            self.hidden_size) for _ in range(3)])  
    ((_, _), (fw_states, bw_states)) =  
    tf.nn.bidirectional_dynamic_rnn(fw_multi_cell,  
                                    bw_multi_cell,  
                                    embedded_input_x,  
                                    input_length,  
                                    dtype=tf.float32)
```



RNN Limitation

이론적으로는 RNN으로 parameter를 잘 조정하면 긴 문장에 대해서도 표현할 수 있는 능력을 가지고 있지만, 실제로 그렇게 동작하도록 학습하는 것이 매우 어렵거나 불가능하다.

$$\frac{\partial h_{t+1}}{\partial h_{t-k+1}} = \prod_{j=1}^k \{(1 - h_{t-j+2}^2)\} \cdot W_h^k$$

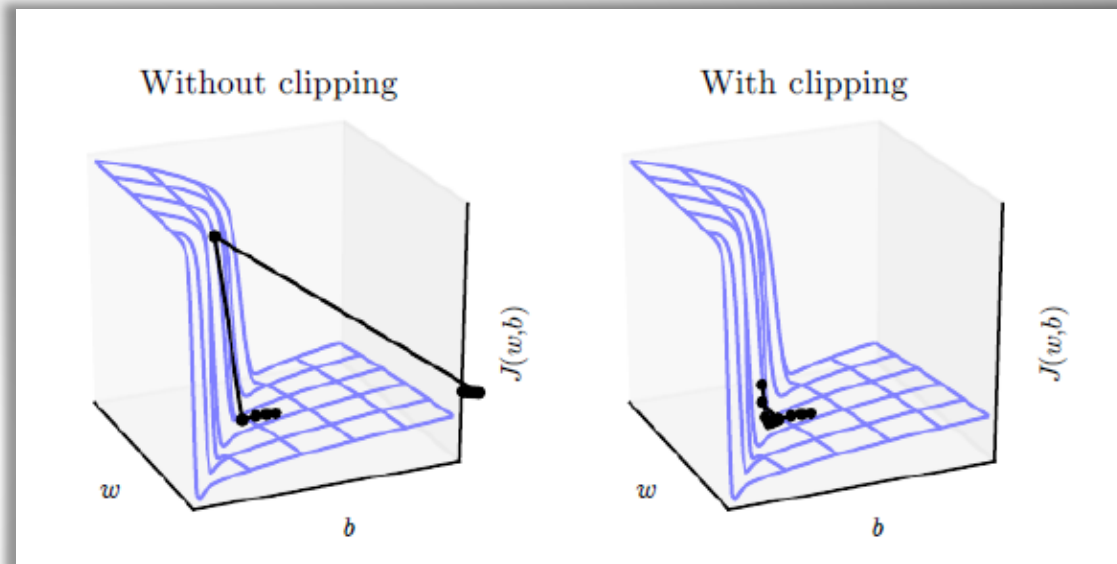
$W_h \uparrow \rightarrow$ Gradient Exploding!

$W_h \downarrow \rightarrow$ Gradient Vanishing!

Gradient Clipping

Gradient explosion 문제를 해결하는 방법 중 하나로 일정 크기 이상의 gradient 값을 가질 경우 값을 축소시킨다.

Algorithm. Pseudo-code for norm clipping

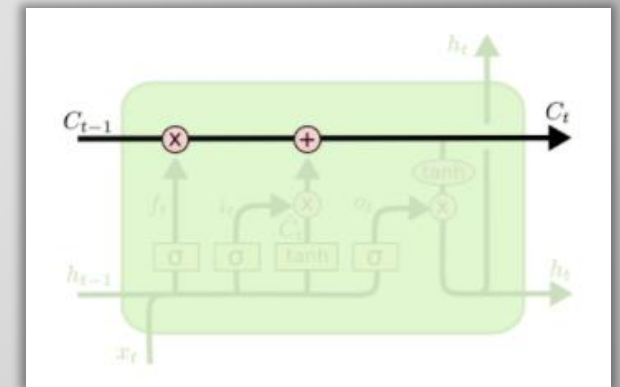
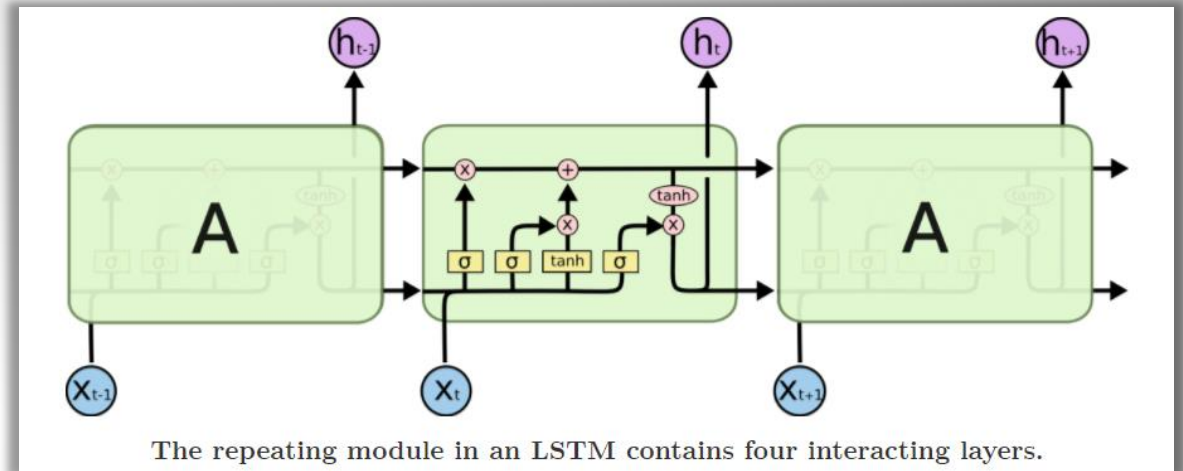
$$\begin{aligned} \hat{\mathbf{g}} &\leftarrow \frac{\partial \mathcal{L}}{\partial \theta} \\ \text{if } \|\hat{\mathbf{g}}\| &> \textit{threshold} \text{ then} \\ &\quad \hat{\mathbf{g}} \leftarrow \frac{\textit{threshold}}{\|\hat{\mathbf{g}}\|} \hat{\mathbf{g}} \\ \text{endif} \end{aligned}$$
[illegible]

3. Deep Learning - LSTM

LSTM

- RNN의 vanishing gradient 문제를 구조적으로 해결
- cell state 를 보면 단순한 선형 연산으로 이루어져 있다
- Gate 라는 구조를 통해 정보의 선별적 학습이 가능하게 된다
- 대부분 실제 task 에서는 LSTM 구조가 default!

```
with tf.variable_scope("recurrent"):  
    cell = tf.nn.rnn_cell.BasicLSTMCell(self.hidden_size)  
    _, states = tf.nn.dynamic_rnn(cell,  
                                  embedded_input_x,  
                                  input_length,  
                                  dtype=tf.float32)
```



3. Deep Learning - 시퀀스데이터 전처리

```
def sentence_to_index(lines, vocab, max_length=0):
    tokens = []
    indexes = []
    max_len = max_length

    if max_len == 0:
        for line in lines:
            token = tokenizer(line)
            tokens.append(token)
            length = len(token)
            if max_len < length:
                max_len = length
    else:
        for line in lines:
            token = tokenizer(line)
            tokens.append(token)

    for token in tokens:
        if len(token) < max_len:
            temp = token
            for _ in range(len(temp), max_len):
                temp.append('<PAD>')
        else:
            temp = token[:max_len]
        index = []
        for char in temp:
            if char in vocab.keys():
                index.append(vocab[char])
            else:
                index.append(vocab['<UNK>'])
        indexes.append(index)

    return indexes
```

```
data = ['나는 생각한다 고로 나는 존재한다.',
        '모든 국가는 그에 걸맞은 정부를 가진다.',
        '이것 또한 지나가리라',
        '죄는 미워하되 사람은 미워하지 말라']
```

```
sentence_to_index(data, vocab)
```

```
[[2, 4, 5, 2, 6, 3, 0],
 [7, 8, 9, 10, 11, 12, 3],
 [13, 14, 15, 0, 0, 0, 0],
 [16, 17, 18, 19, 20, 0, 0]]
```

- 순서정보를 반영할 수 없어서 문장을 한 개의 벡터로 변환했던 것과 달리 한 배치 내의 각 문장을 같은 길이의 인덱스 시퀀스로 변환!
- 각 문장의 길이가 다르므로 배치 내의 가장 긴 문장의 길이에 맞춰서 패딩(<PAD>)을 준다.
- 단어장에 없는 단어는 <UNK> 토큰으로 바꿔준다.

3. Deep Learning - LSTM implementation

```
class LSTM_onehot:
```

```
def __init__(self, sess, vocab_size, hidden_size=128,
              n_class=2, lr=1e-2, trainable=True):
    self.sess = sess
    self.vocab_size = vocab_size
    self.hidden_size = hidden_size
    self.n_class = n_class
    self.lr = lr
    self.trainable = trainable
    self._build_net()
```

```
def _build_net(self):
```

```
    with tf.variable_scope("placeholder"):
        self.input_x = tf.placeholder(tf.int32, (None, None))
        self.input_y = tf.placeholder(tf.int32, (None,))
        input_length = tf.reduce_sum(
            tf.sign(self.input_x), axis=1)
```

```
    with tf.variable_scope("onehot_encoding",
                           reuse=tf.AUTO_REUSE):
        onehot_input_x = tf.one_hot(self.input_x,
                                     self.vocab_size)
```

```
    with tf.variable_scope("recurrent"):
        cell = tf.nn.rnn_cell.BasicLSTMCell(self.hidden_size)
        _, states = tf.nn.dynamic_rnn(cell,
                                      onehot_input_x,
                                      input_length,
                                      dtype=tf.float32)
```

1. 하이퍼 파라미터 및 모델 설정

2. 시퀀스 데이터와 레이블을 입력할 placeholder

3. 각 문장의 길이 정보 추출 (패딩을 0 으로 줌)

4. 입력해준 각 문장의 길이만큼만 time step 진행

5. dynamic_rnn 의 return 은 outputs, states 두 가지

- outputs: 모든 time step의 hidden state
- states: 마지막 time step의 hidden state

3. Deep Learning - LSTM implementation

```
with tf.variable_scope("output", reuse=tf.AUTO_REUSE):  
    W = tf.get_variable('W', dtype=tf.float32,  
                        initializer=tf.truncated_normal(  
                            (self.hidden_size,  
                             self.n_class)))  
    b = tf.get_variable('b', dtype=tf.float32,  
                        initializer=tf.constant(  
                            0.1, shape=(self.n_class,)))  
    logits = tf.nn.xw_plus_b(states.h, W, b)  
    self.prob = tf.reduce_max(tf.nn.softmax(logits),  
                             axis=1)  
    self.prediction = tf.cast(tf.argmax(logits, axis=1),  
                             tf.int32)  
  
with tf.variable_scope("loss"):  
    self.loss = tf.reduce_mean(  
        tf.nn.sparse_softmax_cross_entropy_with_logits(  
            logits=logits, labels=self.input_y))
```

6. output 예측을 위한 계산 수행

$$\hat{y}_t = o_t = \text{softmax}(V \cdot h_t + b_o)$$

3. Deep Learning - LSTM implementation

onehot encoding 에서 **distributed embedding** 으로 변환

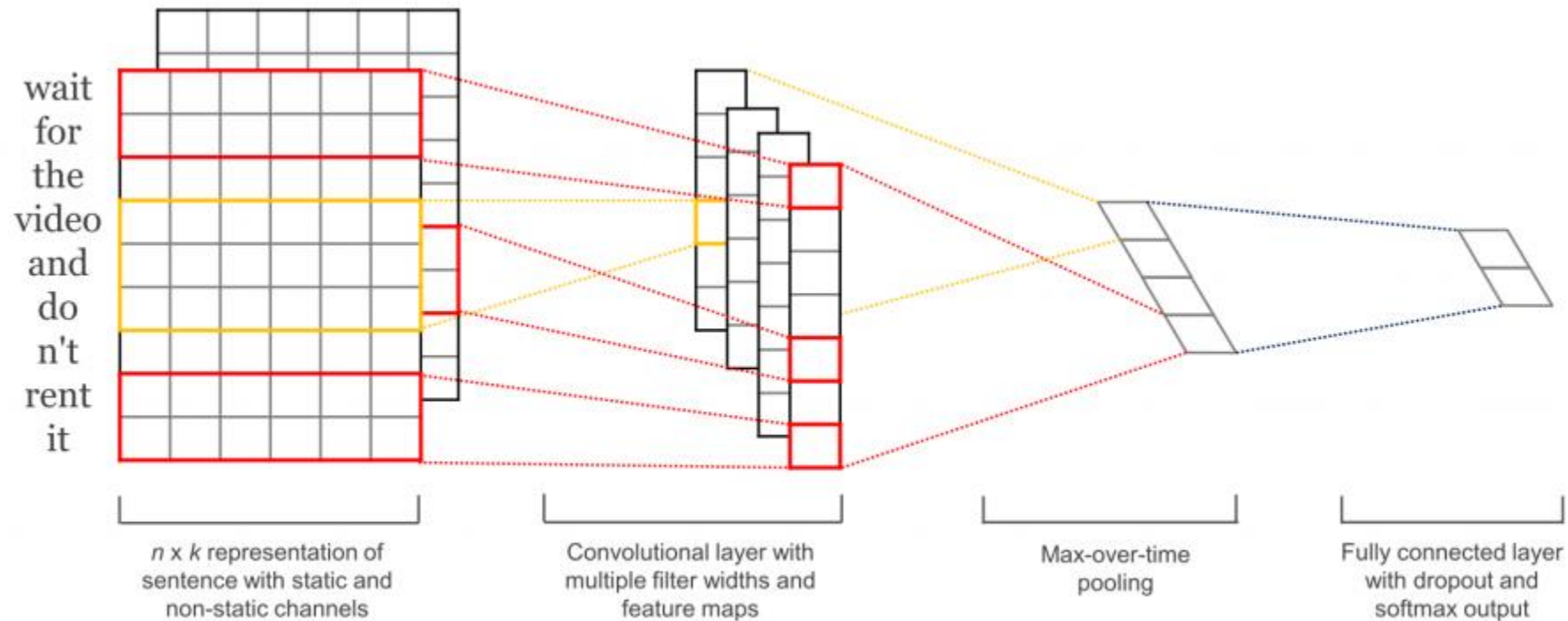
```
with tf.variable_scope("onehot_encoding",  
                        reuse=tf.AUTO_REUSE):  
    onehot_input_x = tf.one_hot(self.input_x,  
                                self.vocab_size)
```



```
with tf.variable_scope("embedding", reuse=tf.AUTO_REUSE):  
    W = tf.get_variable('W', dtype=tf.float32,  
                        initializer=tf.random_uniform(  
                            (self.vocab_size, self.embedding_size),  
                            minval=-1.0, maxval=1.0),  
                        trainable=self.trainable)  
    self.embedding_init = W.assign(self.embedding_placeholder)  
    embedded_input_x = tf.nn.embedding_lookup(W, self.input_x)
```

- onehot encoding은 벡터크기가 vocabulary dictionary 크기만큼 크므로 학습해야 할 모델의 파라미터 수가 큼
- Distributed embedding 을 사용함으로써 단어의 representation을 이용할 수 있다.

CNN for text classification



3. Deep Learning - CNN implementation

```
class CNN:
```

```
def __init__(
    self, sess, vocab_size, sequence_length=30,
    embedding_size=300, filter_sizes=(3,4,5),
    num_filters=128, n_class=2,
    lr=1e-2, trainable=True):
    self.sess = sess
    self.vocab_size = vocab_size
    self.sequence_length = sequence_length
    self.embedding_size = embedding_size
    self.filter_sizes = filter_sizes
    self.num_filters = num_filters
    self.n_class = n_class
    self.lr = lr
    self.trainable = trainable
    self._build_net()
```

1. 하이퍼 파라미터 및 모델 설정

```
def _build_net(self):
    with tf.variable_scope("placeholder"):
        self.input_x = tf.placeholder(tf.int32,
                                       (None, self.sequence_length))
        self.input_y = tf.placeholder(tf.int32, (None,))
        self.embedding_placeholder = tf.placeholder(
            tf.float32, (self.vocab_size,
                         self.embedding_size))
```

2. 시퀀스 데이터와 레이블을 입력할 placeholder

3. Deep Learning - CNN implementation

```
with tf.variable_scope("embedding",
                        reuse=tf.AUTO_REUSE):
    W = tf.get_variable("W", dtype=tf.float32,
                        initializer = tf.random_uniform(
                            [self.vocab_size, self.embedding_size],
                            -1.0, 1.0), trainable=self.trainable)
    self.embedding_init = W.assign(self.embedding_placeholder)
    embedded_chars = tf.nn.embedding_lookup(W, self.input_x)
    embedded_chars_expanded = tf.expand_dims(embedded_chars, -1)
```

3. Distributed embedding

4. CNN input을 위한 reshape

[batch_size, sequence_length, embedding_size, 1]

3. Deep Learning - CNN implementation

```
# Create a convolution + maxpool layer for each filter size
pooled_outputs = []
for filter_size in self.filter_sizes:
    with tf.variable_scope("conv-maxpool-%s" % filter_size,
                           reuse=tf.AUTO_REUSE):
        # Convolution Layer
        filter_shape = (filter_size,
                        self.embedding_size, 1,
                        self.num_filters)
        W = tf.get_variable("W", dtype=tf.float32,
                            initializer=tf.truncated_normal(
                                filter_shape, stddev=0.1))
        b = tf.get_variable("b", dtype=tf.float32,
                            initializer=tf.constant(
                                0.1, shape=(self.num_filters,)))
        conv = tf.nn.conv2d(
            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.sequence_length - filter_size + 1,
                  1, 1],
            strides=[1, 1, 1, 1],
            padding='VALID',
            name="pool")
        pooled_outputs.append(pooled)
```

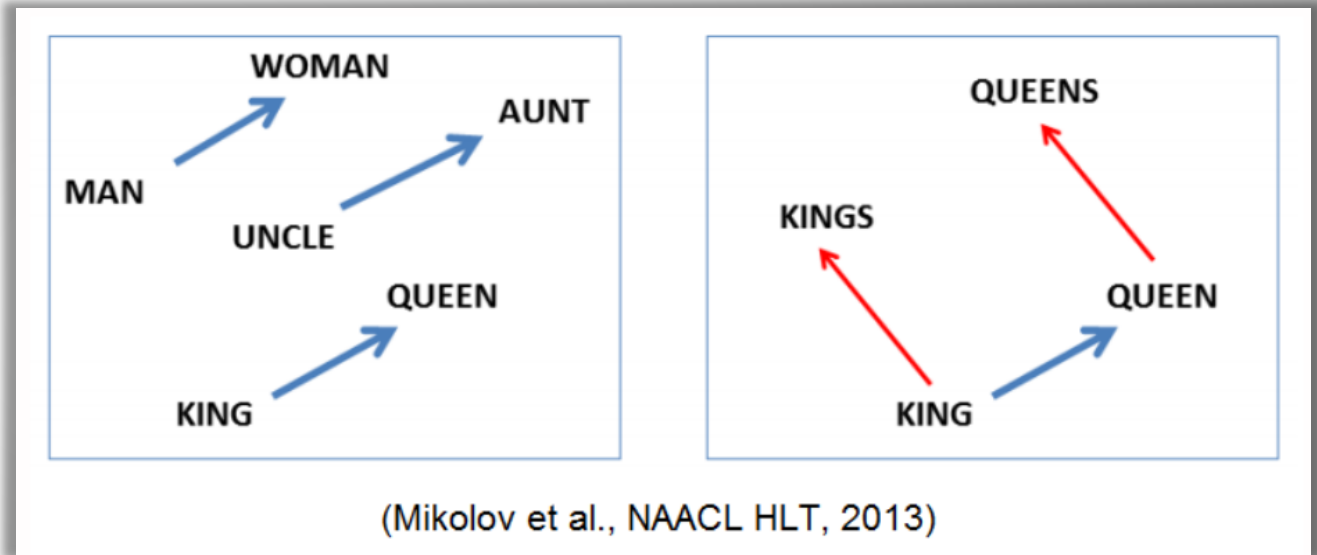
5. 필터 크기 별로 convolution & max pool 연산

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

6. 필터 별로 계산된 결과를 concatenate

word2vec

- 단어의 representation을 학습하는 비지도 학습 방법으로서 비슷한 맥락에서 자주 발생하는 단어들은 서로 비슷한 의미를 갖는다 가정을 기반
- 단어 간의 의미론적 그리고 문법적 관계도 학습
- 비지도 학습 방법인 word2vec을 통해 위 모델에서 사용했던 Distributed embedding에 적용하여 성능을 향상 시킬 수 있다.



<http://w.elnn.kr/search/>

3. Deep Learning - word2vec

CBoW

주어진 단어 앞 뒤로 m 개 씩 총 $2m$ 개의 단어를 Input
으로 이용하여 target 단어를 맞추

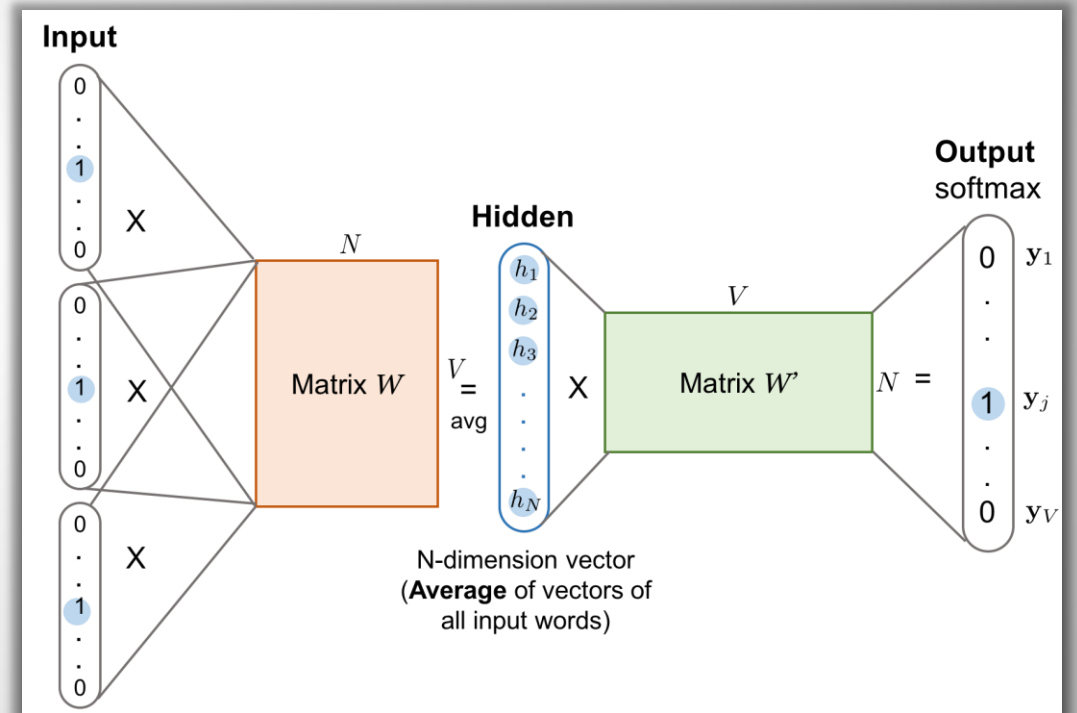
```
import gensim

tokens = [tokenizer(data[i]) for i in range(len(data))]
wv_model = gensim.models.Word2Vec(min_count=1, window=5,
                                   size=4, sg=0)

wv_model.build_vocab(tokens)
wv_model.train(tokens, total_examples=wv_model.corpus_count,
               epochs=wv_model.epochs)
word_vectors = wv_model.wv

print(word_vectors.get_vector('나는'))

[-0.02264038 -0.07700233 -0.05876465  0.0857119 ]
```



3. Deep Learning - word2vec

skip-gram

주어진 단어를 보고 주변 m 개 단어를 예측

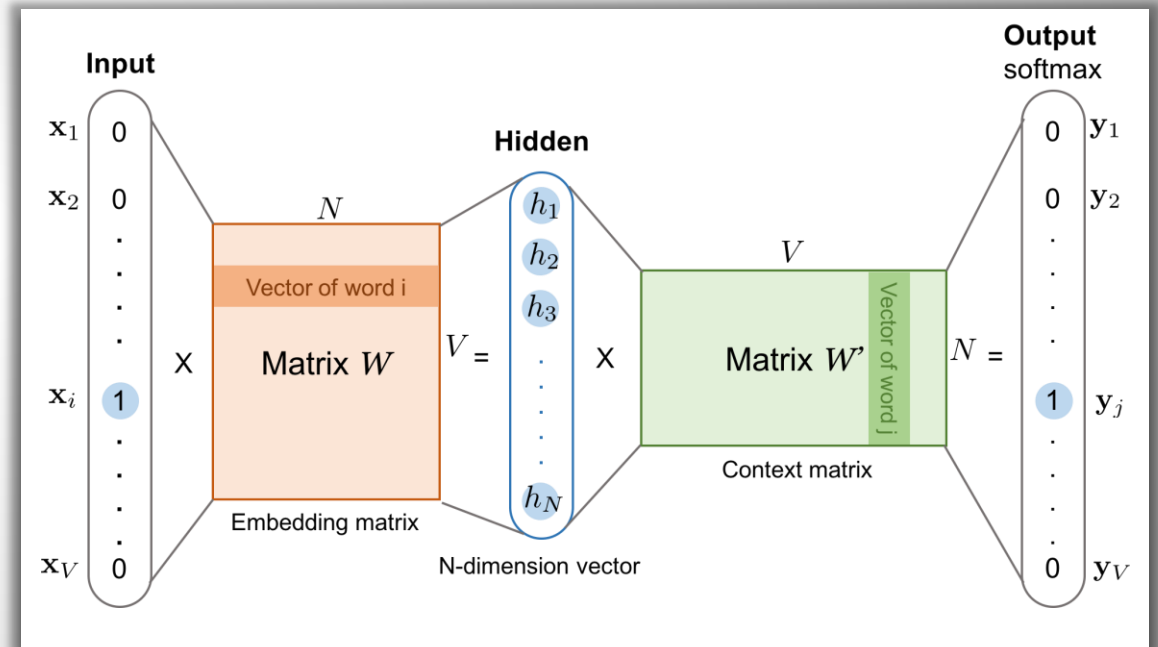
```
import gensim

tokens = [tokenizer(data[i]) for i in range(len(data))]
wv_model = gensim.models.Word2Vec(min_count=1, window=5,
                                   size=4, sg=1)

wv_model.build_vocab(tokens)
wv_model.train(tokens, total_examples=wv_model.corpus_count,
               epochs=wv_model.epochs)
word_vectors = wv_model.wv

print(word_vectors.get_vector('국가는'))

[-0.11459584 -0.07283041  0.05498222  0.11210392]
```



3. Deep Learning - word2vec

skip-gram

주어진 단어를 보고 주변 m 개 단어를 예측

```
import gensim
```

```
tokens = [tokenizer(data[i]) for i in range(len(data))]  
wv_model = gensim.models.Word2Vec(min_count=1, window=5,  
                                   size=4, sg=1)  
  
wv_model.build_vocab(tokens)  
wv_model.train(tokens, total_examples=wv_model.corpus_count,  
               epochs=wv_model.epochs)  
word_vectors = wv_model.wv
```

```
print(word_vectors.get_vector('국가는'))
```

```
[-0.11459584 -0.07283041  0.05498222  0.11210392]
```

Source Text	Training Samples			
<table><tr><td>The</td><td>quick</td><td>brown</td></tr></table> fox jumps over the lazy dog. ➡	The	quick	brown	(the, quick) (the, brown)
The	quick	brown		
The <table><tr><td>quick</td><td>brown</td><td>fox</td></tr></table> jumps over the lazy dog. ➡	quick	brown	fox	(quick, the) (quick, brown) (quick, fox)
quick	brown	fox		
The quick <table><tr><td>brown</td><td>fox</td><td>jumps</td></tr></table> over the lazy dog. ➡	brown	fox	jumps	(brown, the) (brown, quick) (brown, fox) (brown, jumps)
brown	fox	jumps		
The quick brown <table><tr><td>fox</td><td>jumps</td><td>over</td></tr></table> the lazy dog. ➡	fox	jumps	over	(fox, quick) (fox, brown) (fox, jumps) (fox, over)
fox	jumps	over		

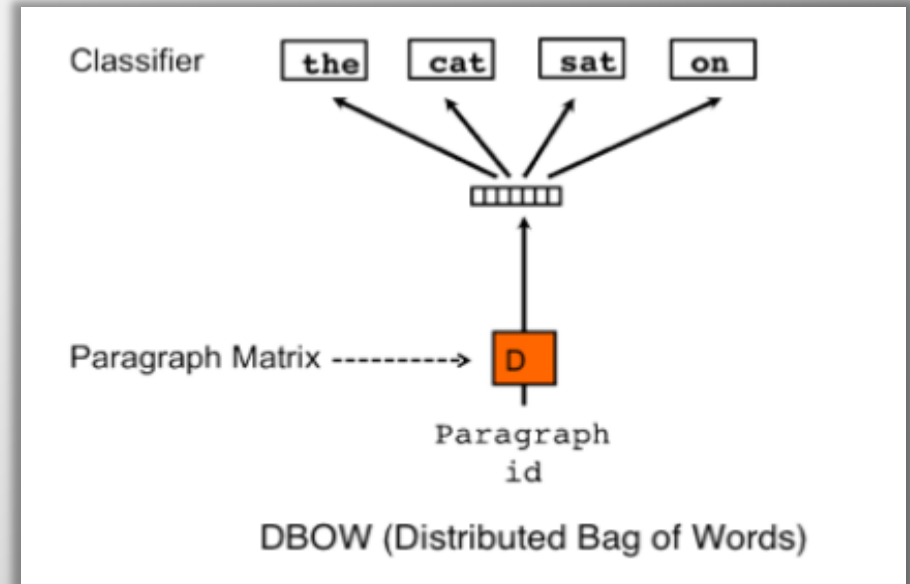
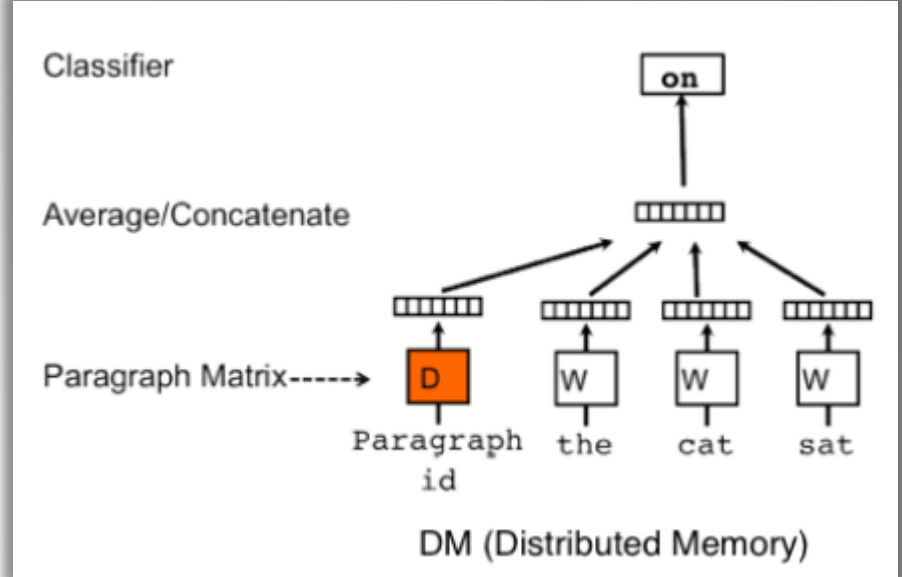
3. Deep Learning - doc2vec

doc2vec

document embedding vectors도 CBOW 또는 Skip-Gram 모델을 이용하여 word embedding vectors를 학습했던 것 처럼 document vector를 추가하여 학습 (문장의 분산표현 학습)

```
tagged_data = [TaggedDocument(words=_d, tags=[str(i)])
                for i, _d in enumerate(data)]

num_epochs_per_cycle = 5
num_cycles = 10
vec_size = 100
alpha = 1e-1
min_alpha = 1e-3
dv_model = Doc2Vec(vector_size=vec_size, alpha=alpha, min_count=2, dm=0)
dv_model.build_vocab(tagged_data)
for cycle in range(num_cycles):
    dv_model.train(tagged_data, total_examples=dv_model.corpus_count,
                   epochs=num_epochs_per_cycle, start_alpha=alpha,
                   end_alpha=min_alpha)
    print('cycle:', '%02d' % (cycle+1))
```



pos (part of speech)

한국어 형태소 분석기 패키지 인 konlpy 의
Twitter 분석기를 이용하여 품사를 추출

```
tw = Twitter()  
tw.pos(data[0])
```

```
[('나', 'Noun'),  
 ('는', 'Josa'),  
 ('생각한', 'Verb'),  
 ('다', 'Eomi'),  
 ('고로', 'Noun'),  
 ('나', 'Noun'),  
 ('는', 'Josa'),  
 ('존재한', 'Verb'),  
 ('다', 'Eomi'),  
 ('.', 'Punctuation')]
```

```
tw = Twitter()  
def pos_extractor(sentence):  
    """  
    extract Noun, Adjective, Verb only  
    """  
    tokens = []  
    pos = tw.pos(sentence, norm=True, stem=True)  
    for token, p in pos:  
        if p == 'Noun' or p == 'Adjective' or p == 'Verb':  
            tokens.append(token)  
    return tokens
```

```
pos_extractor(data[0])
```

```
['나', '생각', '하다', '고로', '나', '존재', '하다']
```

- 품사 중, 의미 전달에 주요한 역할을 하는 명사, 형용사, 동사만 추출
- 어근 추출을 통해 용법이 다르더라도 같이 토큰을 추출할 수 있음 (stem = True)

morphs

한국어 형태소 분석기 패키지 인 konlpy 의
Twitter 분석기를 이용하여 형태소 추출

```
tw = Twitter()  
tw.morphs(data[0])
```

```
['나', '는', '생각한', '다', '고로', '나', '는',  
'존재한', '다', '.']
```

```
tw = Twitter()  
def morphs_extractor(sentence):  
    """  
    extract morphs  
    """  
    tokens = tw.morphs(sentence, norm=True, stem=True)  
    return tokens
```

```
morphs_extractor(data[0])
```

```
['나', '는', '생각한', '다', '고로', '나', '는', '존재한',  
'다', '.']
```

- 주어진 문장을 형태소 토큰으로 분리
- pos 함수와 다르게 어근 추출이 되지 않는다.
(Twitter 분석기의 오류인 것 같다.)

Let's Practice!

4. Application

감성분석 응용

사용자의 입력을 받아 감성을 판단해보자

→ 같은 알고리즘으로 문서 분류 등 다른 task 수행 가능

```
User >> 완전노잼~~~~~  
Bot >> 기분이 좋지 않아 보여요 :(
```



```
User >> 진짜재밌더라!!  
Bot >> 기분이 좋아 보이시네요!
```



```
User >> 자연어처리를 배운다니 너무 재밌겠는걸!  
Bot >> 기분이 좋아 보이시네요!
```



```
User >> 오늘넘지루했엉...  
Bot >> 기분이 좋지 않아 보여요 :(
```



```
User >> 시간가는줄몰랐다ㅋㅋㅋ  
Bot >> 기분이 좋아 보이시네요!
```

삼행시 만들기

RNN을 이용하여 사용자의 입력을 받아 삼행시를 만들어 주는 인공지능을 만들어보자

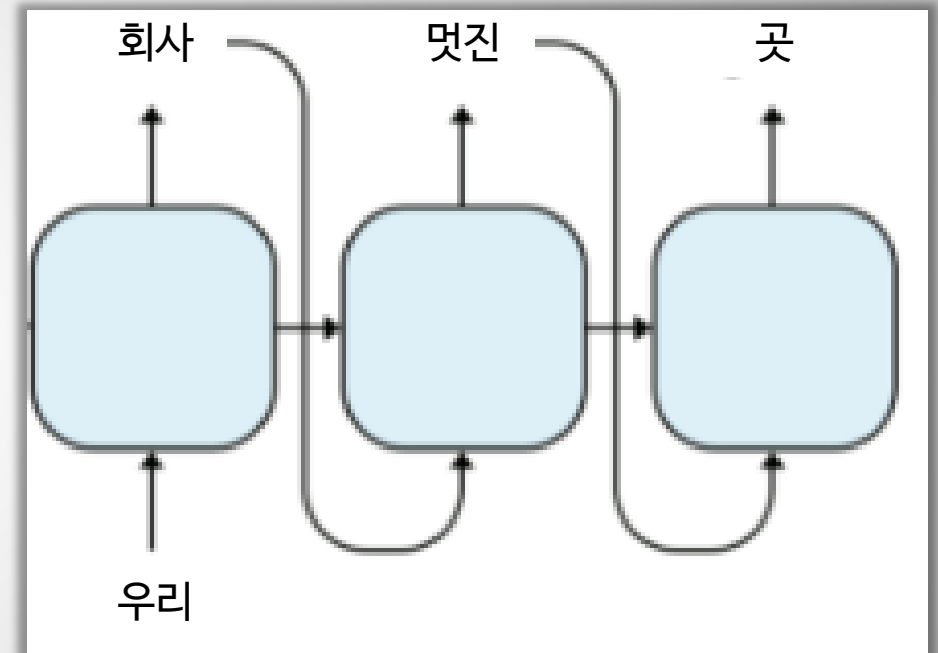
세 글자를 입력하세요: 자연어
자극이 그런 청을 들어주는 조건으로 게 아니었다
연인들의 웃음소리가 줄곧 귓속으로 빨려들어왔다
어서 가봐요

세 글자를 입력하세요: 갯삼성
갯 스무 살 차이였다
삼층짜리 저택 현관 앞으로 다가갔다 삼촌이 나를 내려다보며
성묘 생각을 뒤를 돌아보지 모두가 불쑥 멍하니 안개 속 풍경을
바라보다가, 얼마 후에 가야

세 글자를 입력하세요: 개발자
개지 않고 깊은 상념에 젖어드는 걸 알아챘다
발목, 속으로 하는 내내 ㅎㅎ 하고 콧노래를 불렀다
자국이나 속으로 쭈욱 빨려들어가는

Feed Previous Output as Input

RNN의 각 time step의 output을 다음 input 으로 사용



Feed Previous Output as Input

RNN의 각 time step의
output을 다음 input 으로 사용

```
with tf.variable_scope("inference"):  
    batch_size = tf.unstack(tf.shape(self.first_input))[0]  
    state = self.cell.zero_state(batch_size, dtype=tf.float32)  
    self.predictions = []  
    prediction = 0  
    for i in range(self.max_step):  
        if i == 0:  
            input_ = tf.nn.embedding_lookup(self.embedding, self.first_input)  
        else:  
            input_ = tf.nn.embedding_lookup(self.embedding, prediction)  
        output, state = self.cell(input_, state)  
        inf_logits = tf.add(tf.matmul(output, self.W), self.b)  
        values, indices = tf.nn.top_k(inf_logits, 2)  
        indices = tf.squeeze(indices, axis=0)  
        index = tf.squeeze(tf.multinomial(values, 1), axis=0)  
        prediction = tf.reshape(indices[index[0]], shape=(-1,))  
        self.predictions.append(prediction)  
    self.predictions = tf.stack(self.predictions, 1)
```

대화하기

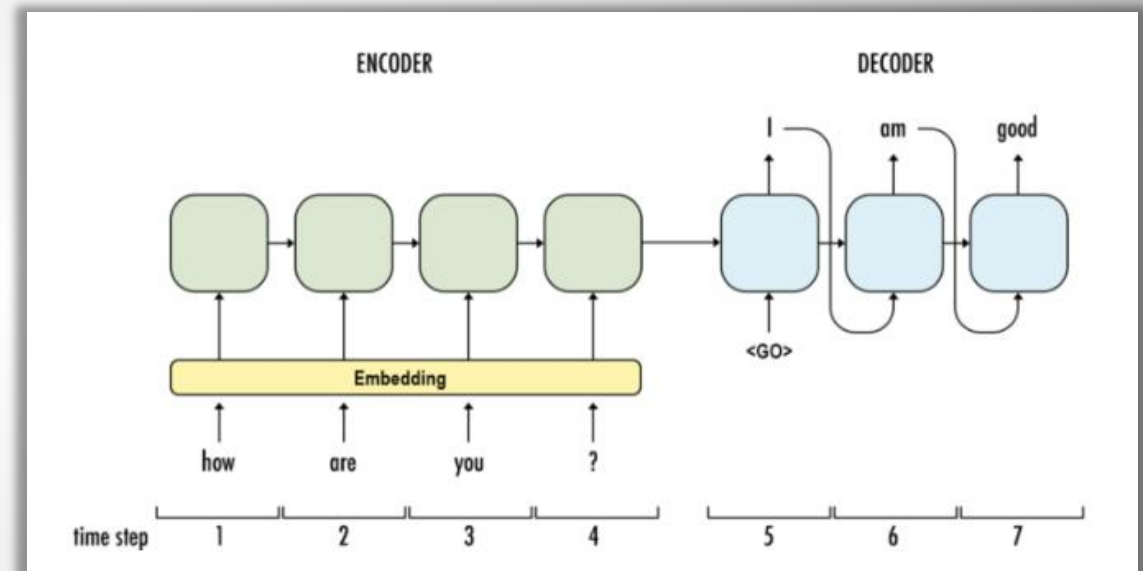
RNN을 이용하여 사용자의 입력을 받아 대답을
만들어내는 인공지능을 만들어보자

```
User >> 안녕!  
Bot >> 안녕  
  
User >> 배고프다  
Bot >> 뭐 좀 먹어!  
  
User >> 영화보러 갈래?  
Bot >> 무슨 영화?  
  
User >> 아 데드풀 보고싶다  
Bot >> 그러면 오늘은 어벤져스보고 내일은 데드풀 콜?
```

4. Application - conversation

seq2seq

RNN 두 개를 사용하여 sequence data를 입력 받아
sequence data를 출력하는 모델



Let's Practice!



Q & A



감사합니다

Email: ryumin93@naver.com

Github: <https://github.com/Bricoler>