개발자들을 위한 쉬운 자연어 처리

기계학습을 이용한 감성분석 및 기계 번역

한양대학교 류민호

오늘의 목표

1. 자연어처리 전반에 대한 원리적 이해 (수리적 X)

2. 실습을 통한 간단한 분석 방법 및 코드 작성 능력 함양

목차

자연어 처리 기초

자연어 처리?

감성 분석

기계 번역

음성 인식

문서 요약

개체명 인식

감성 분석

기계 번역

음성 인식

문서 요약

개체명 인식

[그림 2] “카카오”라는 발성에 대한 음성 파형

$\frac{1}{16000}$초 간격으로 음성의 크기를 측정한 뒤 각 샘플을 2byte short 로 표현

위 방법으로 샘플링 할 때, 1초의 음성이 가질 수 있는 경우의 수는 2^(16000*2*8)

이 값은 대략 10^100000정도로 1 뒤에 0이 10만 개가 있을 정도로 큰 수 이므로, 한 사람이 같은 단어를 여러 번 발음한다고 해도 동일한 값이 나올 확률은 거의 0!

그러면 사람은 어떻게 음성을 인식할 수 있을까?

그러면 사람은 어떻게 음성을 인식할 수 있을까?

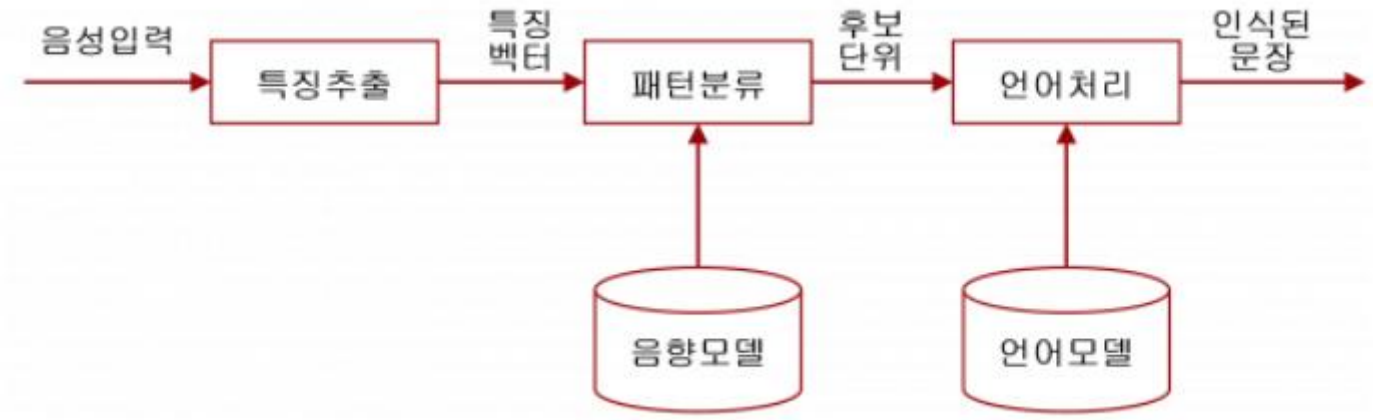
=> 음성이 가질 수 있는 다양한 변이를 훨씬 작은 차원으로 줄인 후 어떤 특징을 인지하여 말을 알아들을 수 있음!

그러면 사람은 어떻게 음성을 인식할 수 있을까?

=> 음성이 가질 수 있는 다양한 변이를 훨씬 작은 차원으로 줄인 후 어떤 특징을 인지하여 말을 알아들을 수 있음!

=> 말소리와 문장 구성을 동시에 고려!

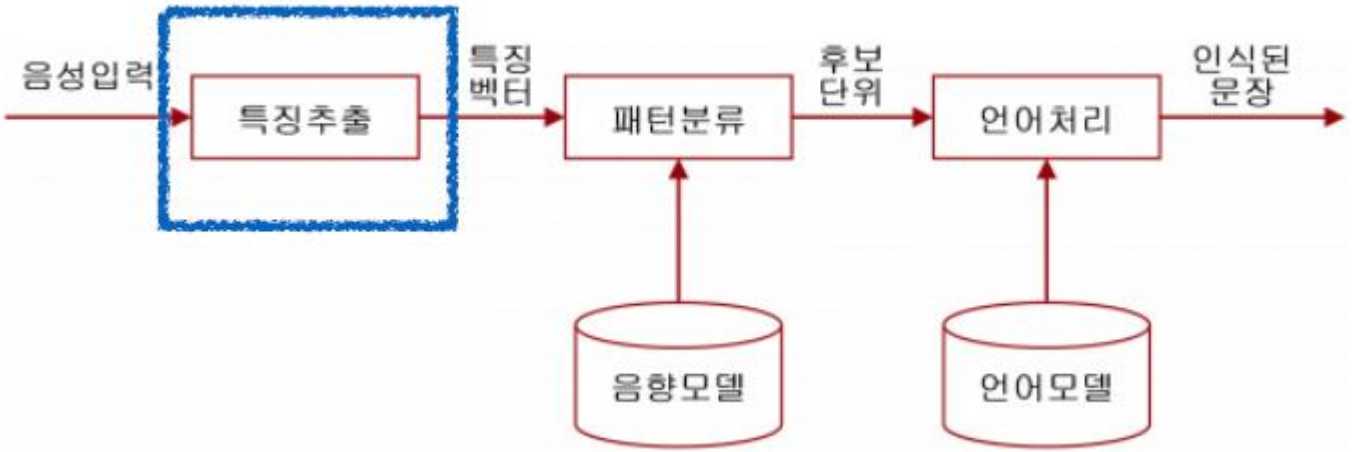
음성 인식(STT)



입력된 음성에 대해 여러 단계의 처리과정을 거친 후 단어 열로 변환해 출력해 주는 것.

음성인식은 입력된 음성이 어떤 단어들로 이루어져 있을 확률이 가장 높은가를 찾는 문제.

음성분석

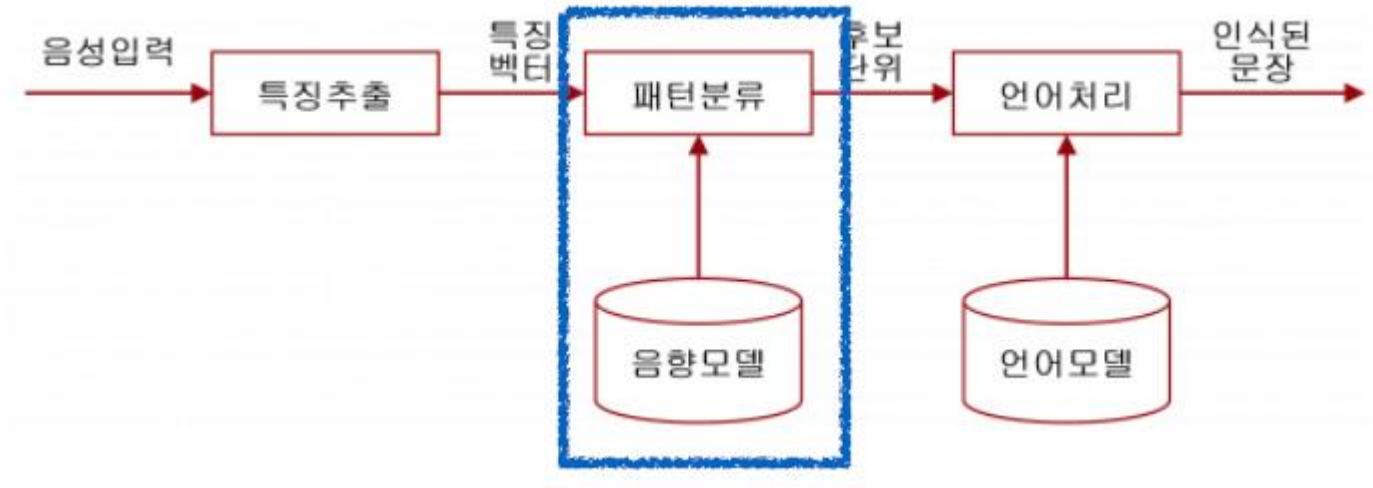


음성신호에서 주파수 분석을 통해 음성의 **특징되는 부분을 추출**하는 과정을 말함.

음성을 마이크로폰을 이용해 디지털 신호로 바꾸는데 이 과정을 **샘플링** 이라고 함.

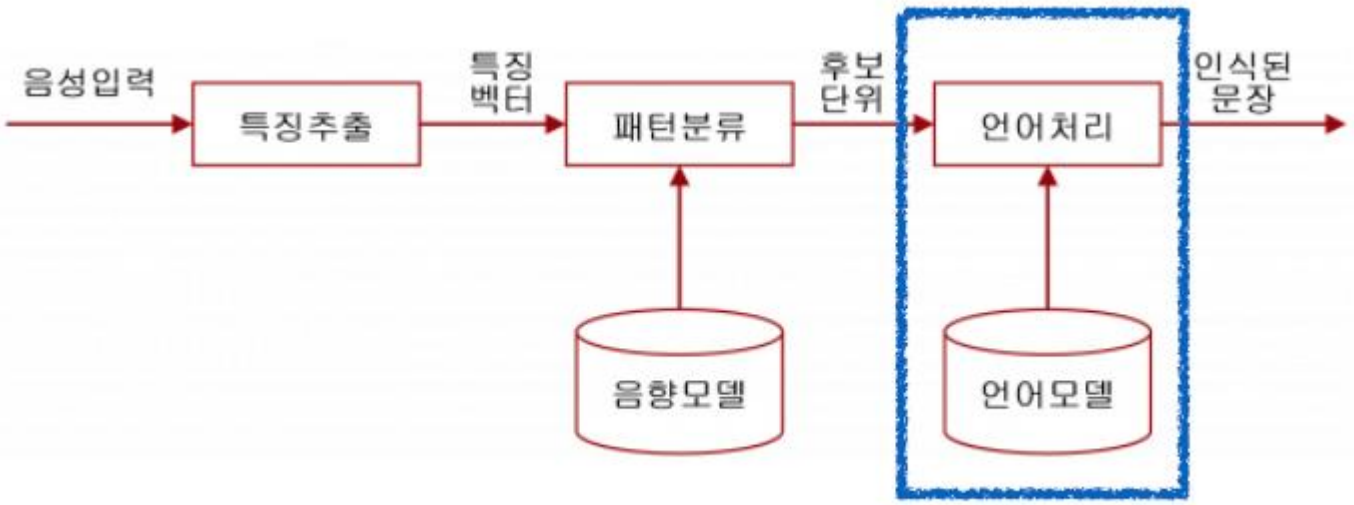
음성이 가질 수 있는 다양한 변이를 작은 차원으로 줄인 후 어떤 특징을 인지한다.

음향모델



음성 0.02초 구간을 0.01초씩 시가축에 따라 움직이며 만든 특징 벡터열과 어휘 셋에 대해 확률을 학습. 음성데이터와 원고를 통해 학습하며 음성신호의 특징을 모델링하는 것.

언어모델



현재 인식되고 있는 단어들 간의 결합 확률을 예측하는 과정.

특정 단어 다음에 나올 확률 추정이 이뤄진다.

감성 분석

기계 번역

음성 인식

문서 요약

개체명 인식

데이터셋 출처: https://github.com/e9t/nsmc

감성 분석 이론 및 실습

Dataset: Naver Movie Review Corpus
(training data size: 150k, test data size: 50k)

Prerequisite: jupyter, scikit-learn, konlpy, tensorflow, numpy, pandas, genism, scipy

Virtual Environment Setting

```
conda create –n tensor python=3.5
activate tensor
conda install tensorflow
conda install scipy
conda install pandas
conda install scikit-learn
pip install --upgrade pip
pip install JPype1-0.6.2-cp36-cp36m-win_amd64.whl
pip install konlpy
conda install ipykernel
python -m ipykernel install --user --name tensor --display-name "Python tensor"
```

1 번째: 자연어 인코딩

“영화 진짜 너무 감동적” → 스플릿 → ‘영화’, ‘진짜’, ‘너무’, ‘감동적’

“황정민 진짜 최고인듯” → 스플릿 → ‘황정민’, ‘진짜’, ‘최고인듯’

“내 인생 영화” → 스플릿 → ‘내’, ‘인생’, ‘영화’

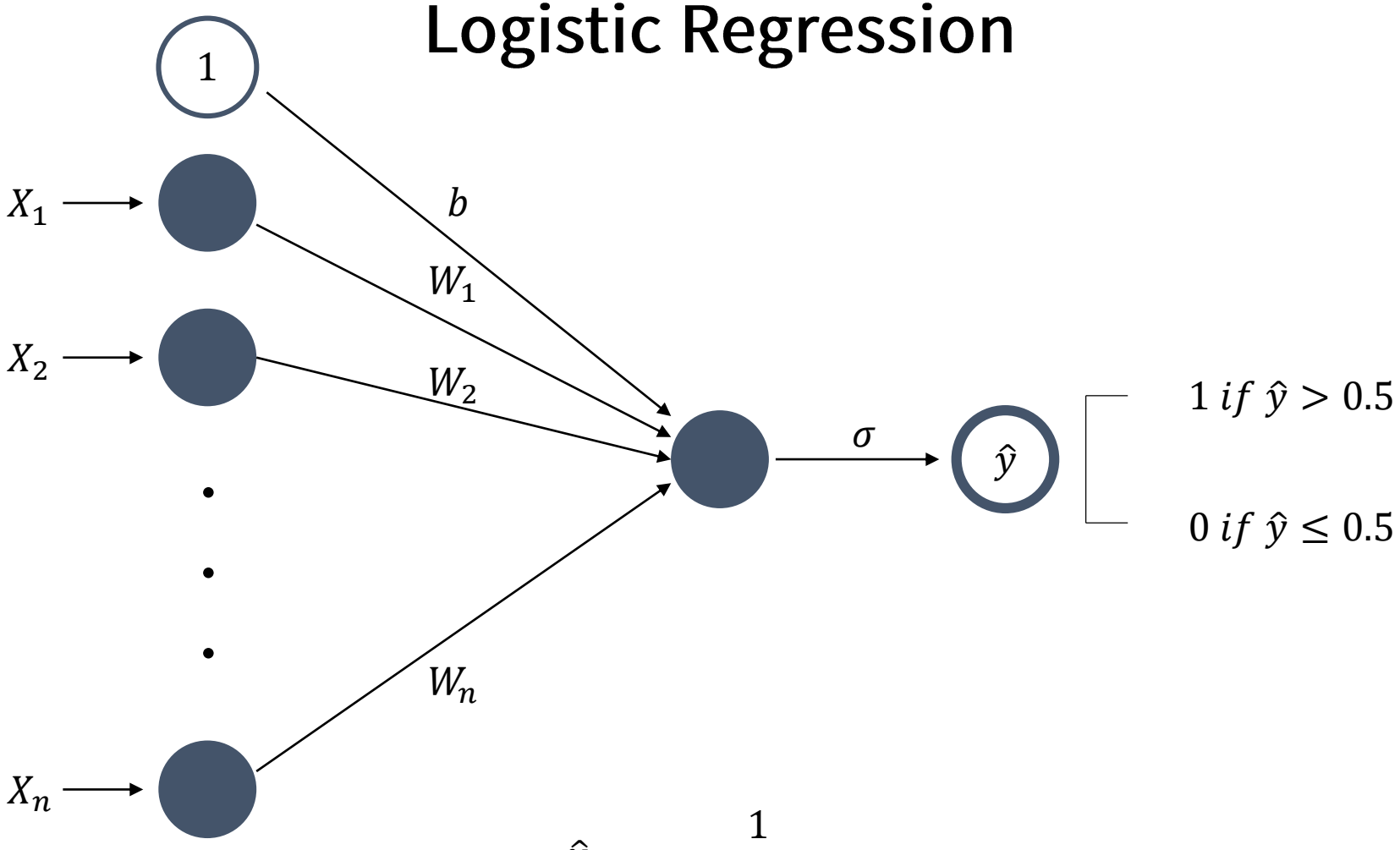
사전 = {영화: 0, 진짜:1, 너무:2, …, 인생: 8}

“영화 진짜 너무 감동적” → 벡터화 → [1,1,1,1,0,0,0,0]

“황정민 진짜 최고인듯” → 벡터화 → [0,1,0,0,1,1,0,0]

“내 인생 영화” → 벡터화 → [1,0,0,0,0,0,1,1]

Logistic Regression



$$\hat{y} = \frac{1}{1 + e^{-(XW+b)}}$$

감성 분석 실습1

보강: ngram

“영화 진짜 너무 감동적” —unigram→ ‘영화’, ‘진짜’, ‘너무’, ‘감동적’

“영화 진짜 너무 감동적” —bigram→ ‘영화 진짜’, ‘진짜 너무’, ‘너무 감동적’

사전 = {영화: 0, 진짜:1, 너무:2, 감동적:3, 영화 진짜:4,
진짜 너무:5, 너무 감동적:6}

장점: 여전히 부족하지만 단어의 순서를 일정부분 고려할 수 있다.

단점: 차원이 커진다.

보강: mindf, maxdf

*mindf: 단어 사전에 추가되기 위한 최소한의 등장 횟수를 설정

*maxdf: 일정 수준 이상으로 자주 발생하는 단어 사전에서 제외

보강: TF-IDF

*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

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

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

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

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

특정 문서 내에서 단어 빈도가 높을 수록, 그리고 전체 문서들 중 그 단어를 포함한 문서가 적을 수록 TF-IDF값이 높아지므로 이 값을 이용하면 모든 문서에 흔하게 나타나는 단어를 걸러내는 효과

감성 분석 실습2

0 번째: 자연어 전처리

실제 데이터는 지저분한 경우가 더 많다!

스플릿

“너무재밋었다그래서보는것을추천한닼ㅋ” ⟶ ‘너무재밋었다그래서보는것을추천한닼ㅋ’

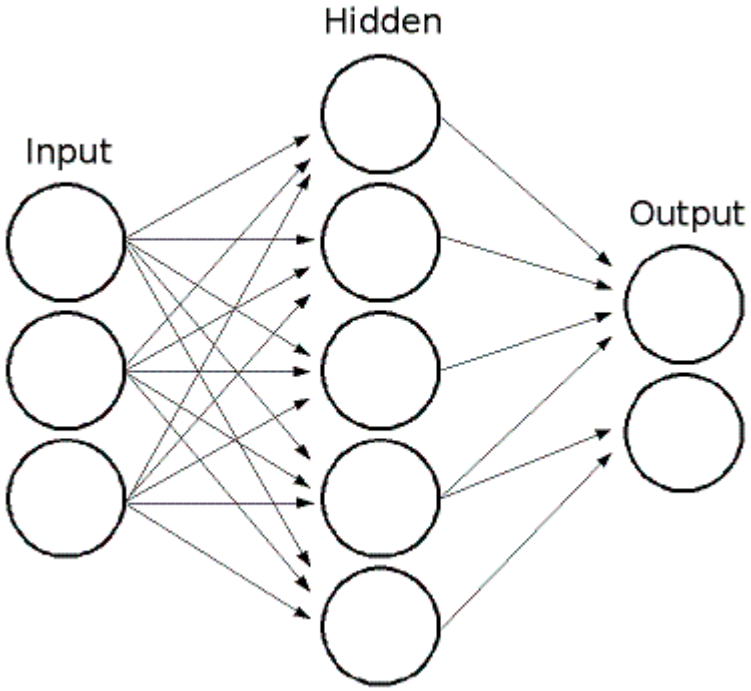
전처리

“너무재밋었다그래서보는것을추천한닼ㅋ” ⟶ ‘너무’, ‘재미있다’, ‘그래서’, ‘보다’, ‘것’, ‘을’, ‘추천하다’

정규화 및 어간 추출

감성 분석 실습3

Feed-forward neural network



$$h = \sigma(XW_h + b_h)$$

$$o = softmax(hW_o + b_o)$$

$$\hat{y} = argmax(o)$$

기계 학습 기초 구성

Data: 학습데이터로부터 패턴을 학습한다.

Model: 사용자가 지정한 모델구조에 맞춰서 학습을 진행

Loss function: 학습을 위한 가이드

Optimizer: 학습 도구 (gradient descent, Adam 등)

텐서플로우 기초 구성

tf.placeholder: 데이터를 넣는 공간

tf.Variable: 모델에 사용되는 학습할 파라미터

cost function: 학습을 어느 방향으로 진행할 지 알아내기 위한 지표

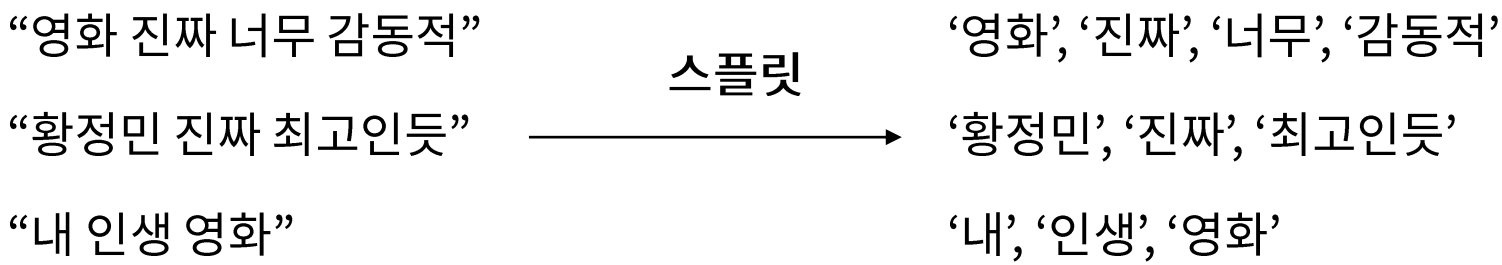
tf.train.GradientDescentOptimizer: 최적화 도구

감성 분석 실습4

워드 임베딩

(Word Embedding)

One-hot encoding



사전 = {영화: 0, 진짜:1, 너무:2, …, 인생: 8}

영화 ⟶ [1,0,0,0,0,0,0,0]

진짜 ⟶ [0,1,0,0,0,0,0,0]

⋮

인생 ⟶ [0,0,0,0,0,0,0,1]

One-hot encoding

Problem? → No correlation between words, too large dimension

dimension= $|V|$; 큰 데이터셋에서는 최대 천 만 개 이상

Computationally too expensive!

No contextual/semantic information embedded in one-hot vectors

Distributed Representation of words

영화 $\longrightarrow$ $[0.123, 0.643, \cdots, 0.212]$

진짜 $\longrightarrow$ $[0.533, 0.186, \cdots, 0.935]$

$\vdots$

인생 $\longrightarrow$ $[0.864, 0.111, \cdots, 0.486]$

단어를 특정 차원의 실수 값을 가지는 분산 표현으로 잘 나타낼 수 있으면, 단어 간의 유사도와 단어의 문맥적 의미를 파악할 수 있지 않을까?

https://shuuki4.wordpress.com/2016/01/27/word2vec-관련-이론-정리/

Distributed Representation of words

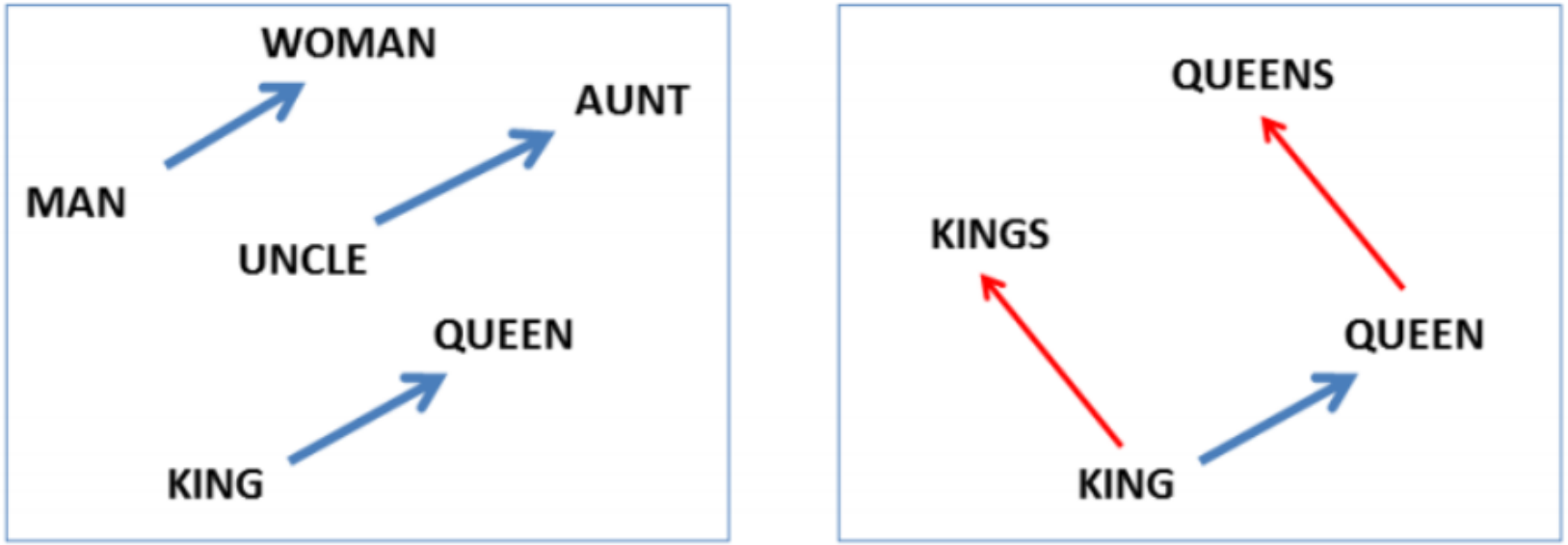
‘비슷한 분포를 가진 단어들은 비슷한 의미를 가진다’ 는
언어학의 distributional hypothesis 에 입각

비슷한 분포를 가진다는 것은 기본적으로 단어들이 같은
문맥에서 등장한다는 것을 의미

예를 들어, ‘사과’, ‘포도’, ‘딸기’라는 단어가 같이 등장하는 일이
빈번하게 일어난다면, 이 단어들이 유사한 의미를 가진 것으로
유추할 수 있다는 것

https://shuuki4.wordpress.com/2016/01/27/word2vec-관련-이론-정리/

Distributed Representation of words



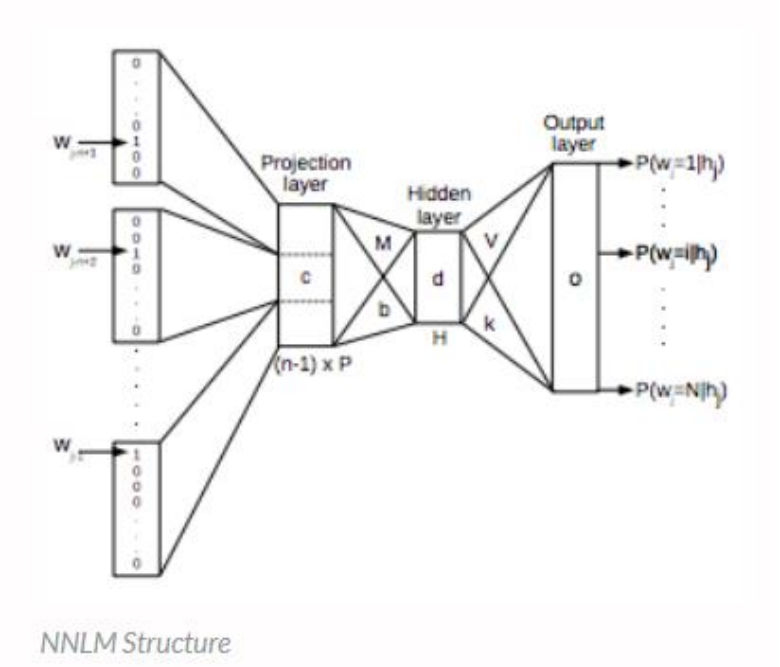
(Mikolov et al., NAACL HLT, 2013)

그렇다면 어떻게 단어를
벡터화 할 것인가?

NNLM – RNNLM – Word2Vec

https://shuuki4.wordpress.com/2016/01/27/word2vec-관련-이론-정리/

Feed-Forward Neural Net Language Model (NNLM)



NNLM Structure

현재 보고 있는 단어 이전의 단어들 N개를 one-hot encoding 으로 벡터화하여 인풋으로 넣어주고, Projection layer와 MLP를 거쳐 output layer에서 각 단어가 나올 확률을 계산

(사용하게 될 단어의 벡터들은 Projection Layer의 값)

https://shuuki4.wordpress.com/2016/01/27/word2vec-관련-이론-정리/

Feed-Forward Neural Net Language Model (NNLM)

Disadvantages

1. 몇 개의 단어를 볼 건지에 대한 파라미터 N이 고정되어 있고, 정해주어야 한다.
2. 이전의 단어들에 대해서만 신경쓸 수 있고, 현재 보고 있는 단어 앞에 있는 단어들을 고려하지 못한다.
3. 가장 치명적으로 **느리다**.

https://shuuki4.wordpress.com/2016/01/27/word2vec-관련-이론-정리/

Feed-Forward Neural Net Language Model (NNLM)

- 단어들을 Projection 시키는 데에 NxP
- Projection Layer에서 Hidden Layer로 넘어가는 데에 NxPxH
- Hidden Layer에서 Output Layer로 넘어가려면 모든 단어에 대한 확률을 계산해야 하므로 HxV

즉, NxP + NxPxH + HxV 만큼의 시간이 소요.

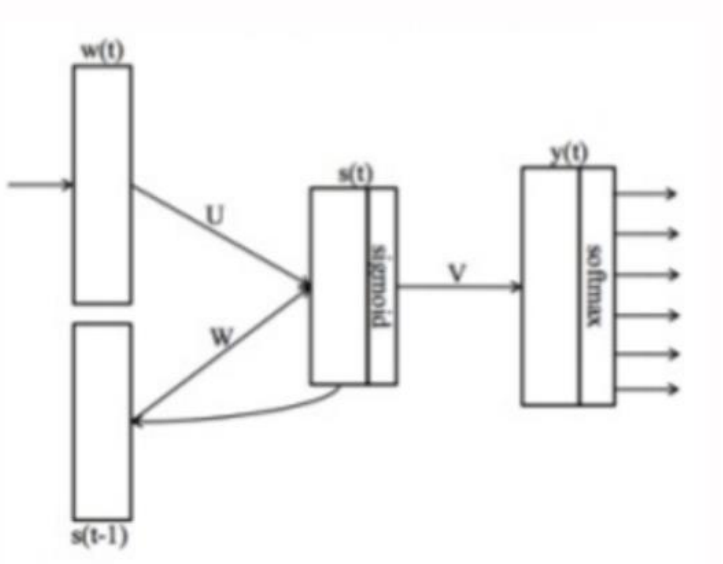
보통 사전은 가능한 모든 단어들을 가지고 있어야 하므로, 그 크기가 굉장히 큼 (최대 천 만개 정도).

즉, 이 경우 Dominating Term은 HxV가 될 것이다. 보통 N=10, P=500, H=500 정도의 값을 사용한다고 생각하면, 하나의 단어를 계산하는 데에 O(HxV) = O(50억) 정도의 계산이 필요.

Hierachical softmax를 이용하면 HxV 항을 Hx ln(V) 정도로 줄일 수 있고 이를 고려하면 Dominating Term은 NxPxH가 되지만 그래도 계산량은 O(NxPxH) = O(250만) 정도로 많은 연산이 소요

https://shuuki4.wordpress.com/2016/01/27/word2vec-관련-이론-정리/

Recurrent Neural Net Language Model (RNNLM)



RNNLM Structure

Projection Layer 없이 Input, Hidden, Output Layer로만 구성되는 대신, Hidden Layer에 Recurrent한 연결이 있어 이전 시간의 Hidden Layer의 입력이 다시 입력되는 형식. 이 네트워크의 경우 그림에서 U라고 나타나 있는 부분이 Word Embedding으로 사용이 되며, Hidden Layer의 크기를 H라고 할 때 각 단어는 길이 H의 벡터로 표현될 것

https://shuuki4.wordpress.com/2016/01/27/word2vec-관련-이론-정리/

Recurrent Neural Net Language Model (RNNLM)

Characteristics

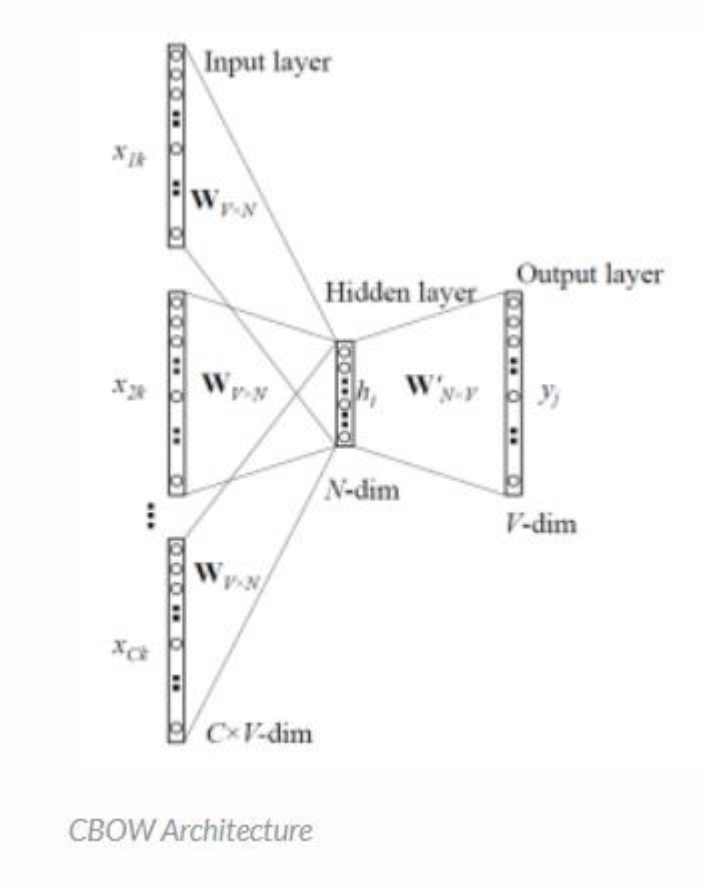
1. 기존 NNLM의 단점 1번, 2번을 해결
2. 하지만 여전히 **느리다.**

- Input Layer에서 Hidden Layer로 넘어가는 데에 H
- hidden(t-1)에서 hidden(t)로 넘어가는 벡터를 계산하는 데에 HxH
- Hidden Layer에서 Output 결과를 내기 위해 모든 단어에 대해 확률계산을 해야하므로 HxV

H를 500으로 잡는다면 O(25만)의 연산량이 필요

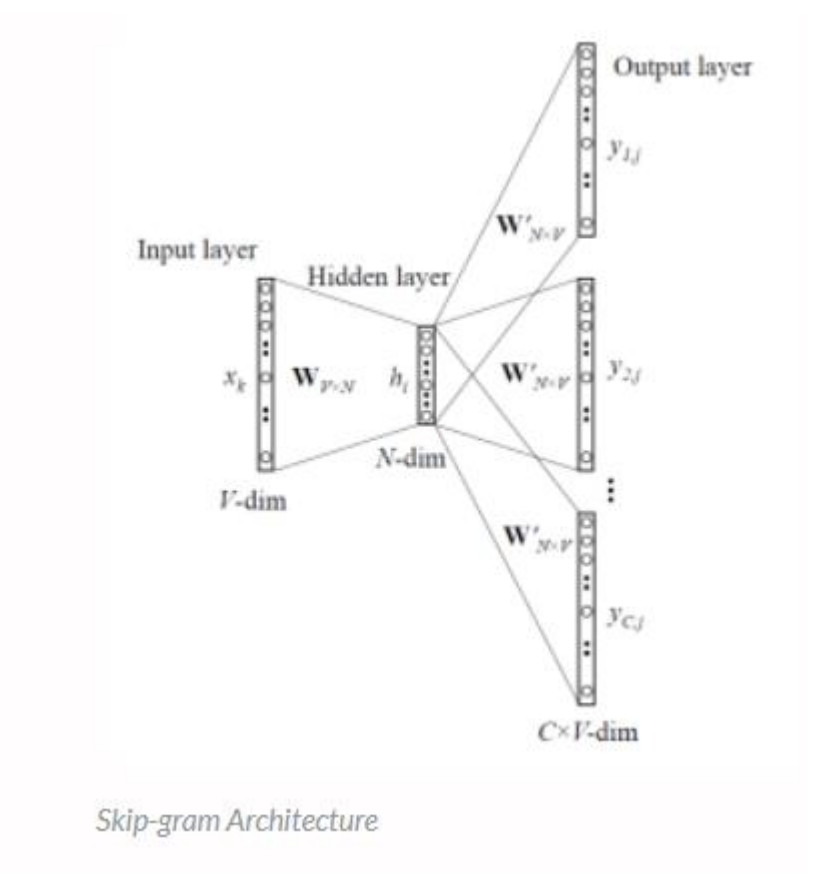
https://shuuki4.wordpress.com/2016/01/27/word2vec-관련-이론-정리/

Word2Vec



CBOW Architecture

주어진 단어 앞 뒤로 2/C개 씩 총 C개의 단어를 Input으로 이용하여 주어진 단어를 맞춤



Skip-gram Architecture

주어진 단어를 Input으로 이용하여 주변 단어를 예측

https://shuuki4.wordpress.com/2016/01/27/word2vec-관련-이론-정리/

Skip-gram model

- 현재 단어를 Projection 하는 데에 N (embedding dimension)
- Output을 계산하는 데에 N x V, 테크닉을 사용하면 N x ln V
- 총 C개의 단어에 대해 진행해야 하므로 총 C x N x ln V 의 연산이 필요 (약 O(C x10000)정도)

Negative sampling, Subsampling Frequent Words 등을 이용하여 학습속도 및 성능 추가 향상

Model Architecture	Semantic-Syntactic Word Relationship test set		MSR Word Relatedness Test Set [20]
	Semantic Accuracy [%]	Syntactic Accuracy [%]	
RNNLM	9	36	35
NNLM	23	53	47
CBOW	24	64	61
Skip-gram	55	59	56

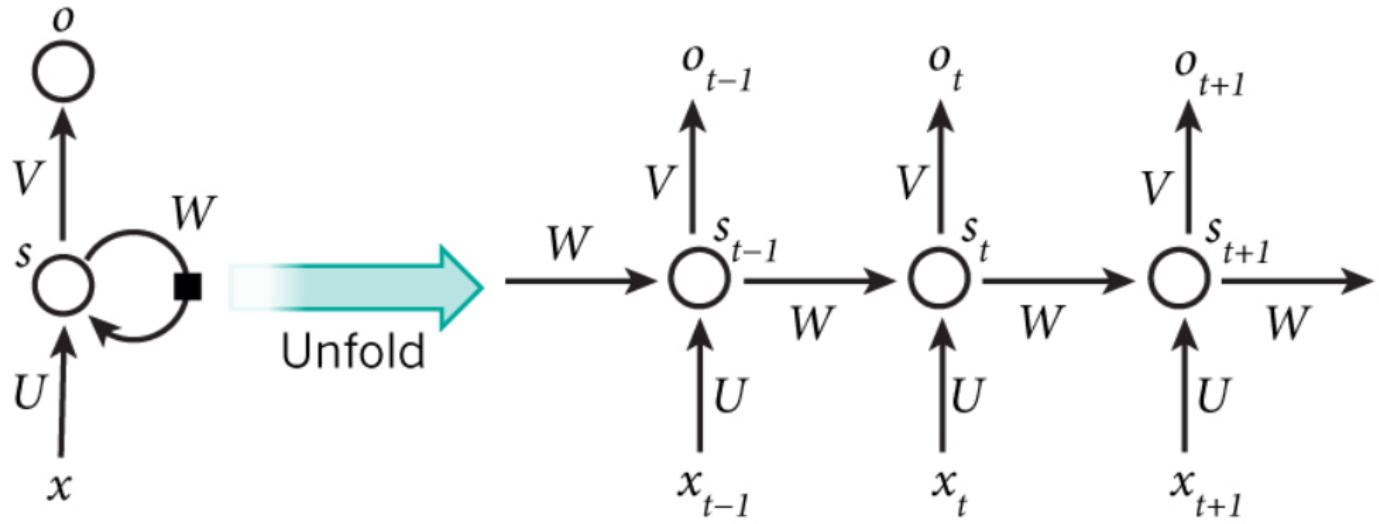
Analogy Reasoning Task Results for Various Models. Word Dimension = 640

Word2Vec 실습

딥러닝 자연어 처리 알고리즘

http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/

RNN



A recurrent neural network and the unfolding in time of the computation involved in its forward computation. Source: Nature

$$s_t = \tanh(U \cdot x_t + W \cdot s_{t-1} + b_s)$$

$$o_t = \text{softmax}(V \cdot s_t + b_o)$$

http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/

RNN

- s_t captures information about what happened in all the previous time steps. The output at each step o_t is solely calculated base on the memory at time t.

- Unlike a traditional deep neural network, which uses different parameters at each layer, a RNN shares the same parameters across all steps.

- The above diagram has outputs at each time step, but depending on the task this may not be necessary. For example, when predicting the sentiment of a sentence we may only care about the final output, not the sentiment after each word. Similarly, we may not need inputs at each time step.

RNN 실습

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

RNN

- In theory, RNNs are absolutely capable of handling such “long-term dependencies.” A human could carefully pick parameters for them to solve toy problems of this form. Sadly, in practice, RNNs don’t seem to be able to learn them.

  → **Vanishing gradient problem!!!**

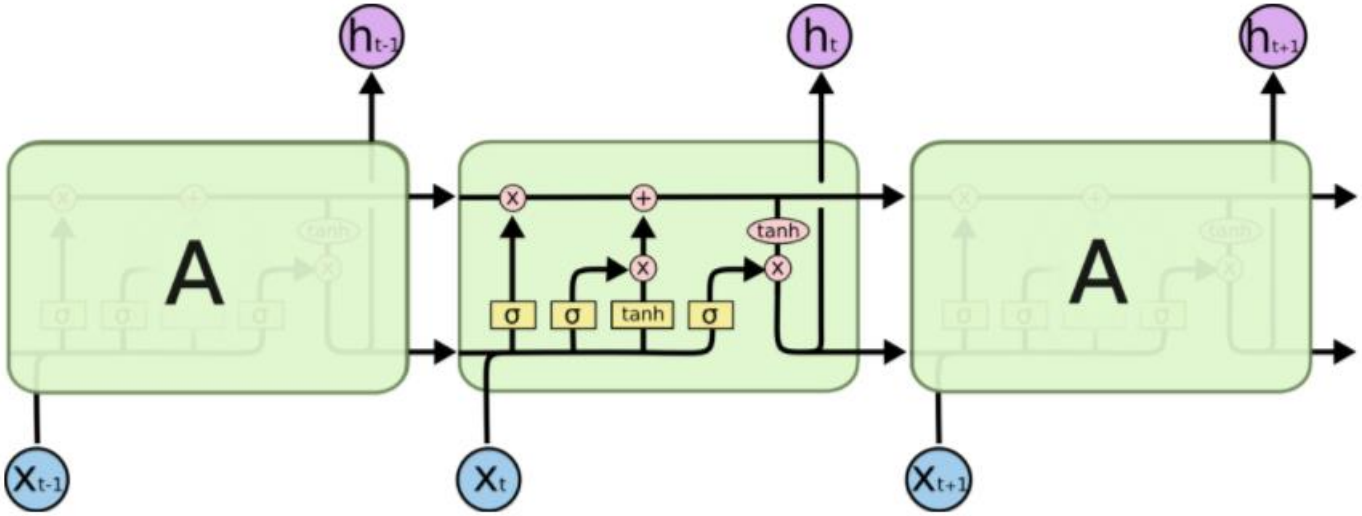
$$\frac{\partial L}{\partial W_h} = \sum_{t} \sum_{k=1}^{t+1} \frac{\partial L(t+1)}{\partial z_{t+1}} \cdot \frac{\partial z_{t+1}}{\partial h_{t+1}} \cdot \prod_{i=k}^{t} \left(\frac{\partial h_{i+1}}{\partial h_i}\right) \cdot \frac{\partial h_k}{\partial W_h}$$

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

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

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM (Long Short-Term Memory)



The repeating module in an LSTM contains four interacting layers.

$$\begin{bmatrix} f_t \\ i_t \\ o_t \end{bmatrix} = \sigma\left(\begin{bmatrix} W_f \\ W_i \\ W_o \end{bmatrix} \cdot [h_{t-1}, x_t] + \begin{bmatrix} b_f \\ b_i \\ b_o \end{bmatrix}\right)$$

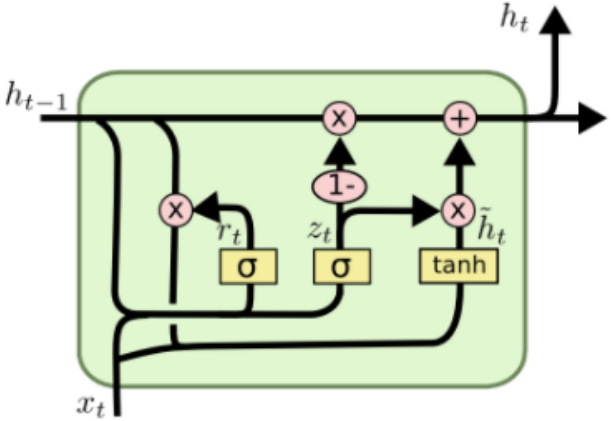
$$\tilde{C}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$$

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$

$$h_t = o_t \odot \tanh(C_t)$$

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

GRU (Gated Recurrent Unit)



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

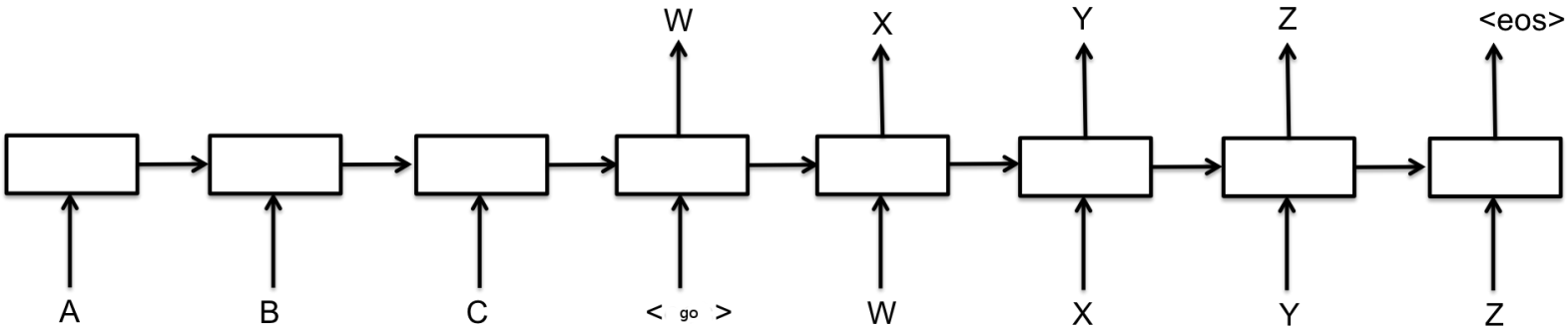
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

LSTM/GRU 실습

K.H. Cho, “Neural machine translation by jointly learning to align and translate”

Sequence to sequence model

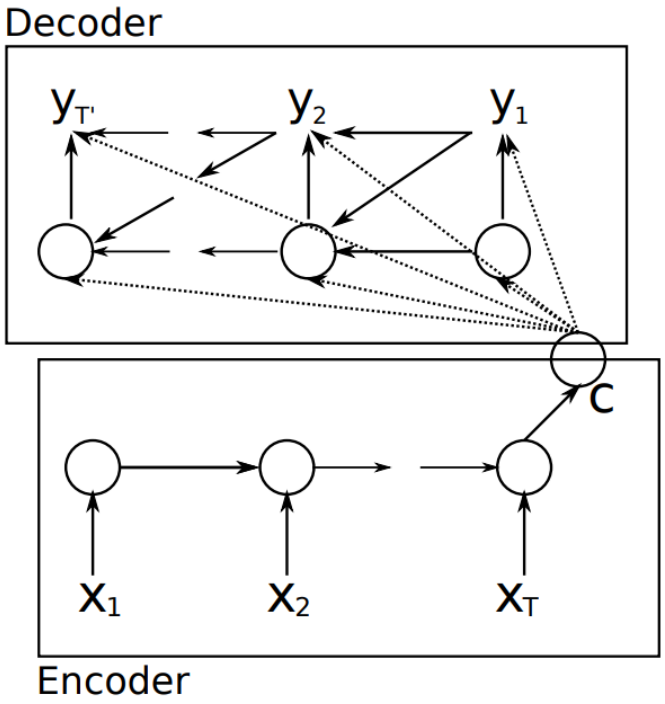


Encoder-decoder architecture

One RNN encodes a sequence of symbols into a fixed-length vector representation, and the other decodes the representation into another sequence of symbols.

K.H. Cho, “Neural machine translation by jointly learning to align and translate”

Sequence to sequence model



Encoder-decoder architecture

$$\boldsymbol{x} = (x_1, \cdots, x_{T_x}) \quad \boldsymbol{y} = (y_1, \cdots, y_{T_y})$$

$$h_t = f(x_t, h_{t-1})$$

$$c = q(\{h_1, \cdots, h_{T_x}\})$$

$$p(\boldsymbol{y}) = \prod_{t=1}^{T} p(y_t | \{y_1, \cdots, y_{t-1}\}, c)$$

$$p(y_t | \{y_1, \cdots, y_{t-1}\}, c) = g(y_{t-1}, s_t, c)$$

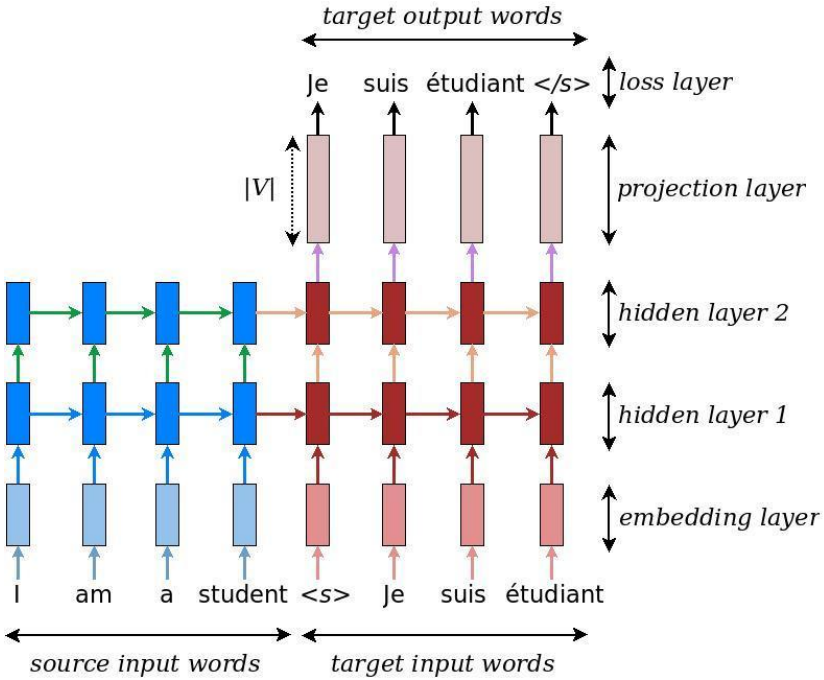
h_t: *encoder hidden state*

s_t: *decoder hidden state*

seq2seq 실습

K.H. Cho, “Neural machine translation by jointly learning to align and translate”

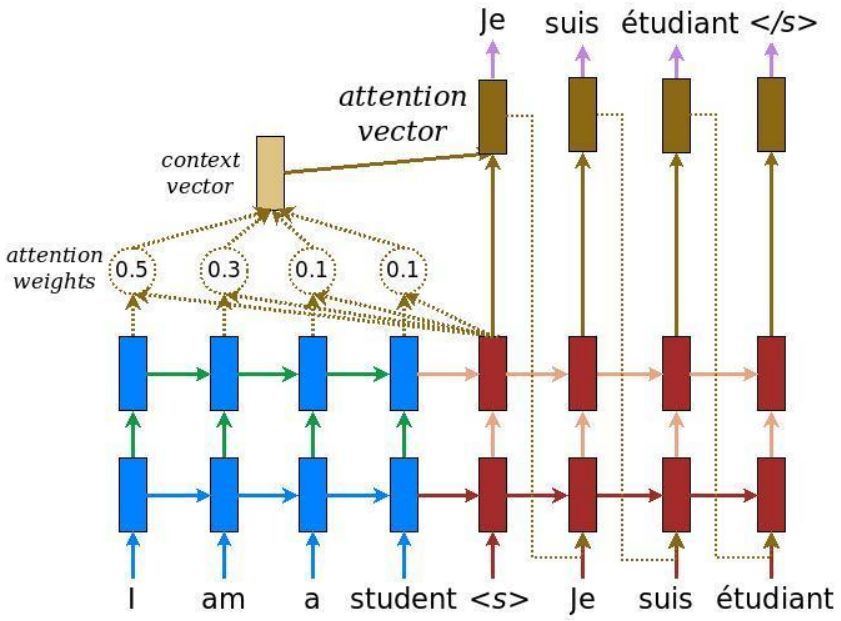
Seq2seq model Problem



Encoder-decoder architecture

“It seems somewhat unreasonable to assume that we can encode all information about a potentially very long sentence into a single vector and then have the decoder produce a good translation based on only that.”

Attention Mechanism



Attention mechanism

$$p(y_i|y_1,\cdots,y_{i-1},\boldsymbol{x}) = g(y_{i-1},s_i,c_i)$$

$$s_i = f(s_{i-1},y_{i-1},c_i)$$

Here the probability is conditioned on a distinct context vector c_i for each target y_i.

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j, \qquad \alpha_{ij} = \frac{exp(e_{ij})}{\sum_{k=1}^{T_x} exp(e_{ik})}$$

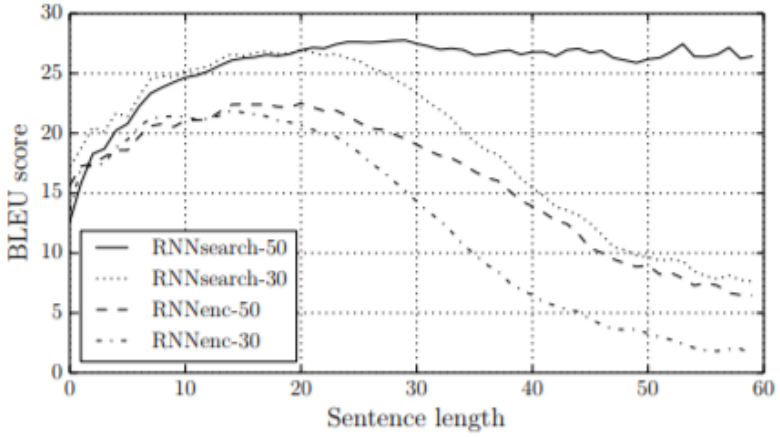
$$e_{ij} = a(s_{i-1},h_j)$$

The alignment a is parameterized as a feedforward neural network

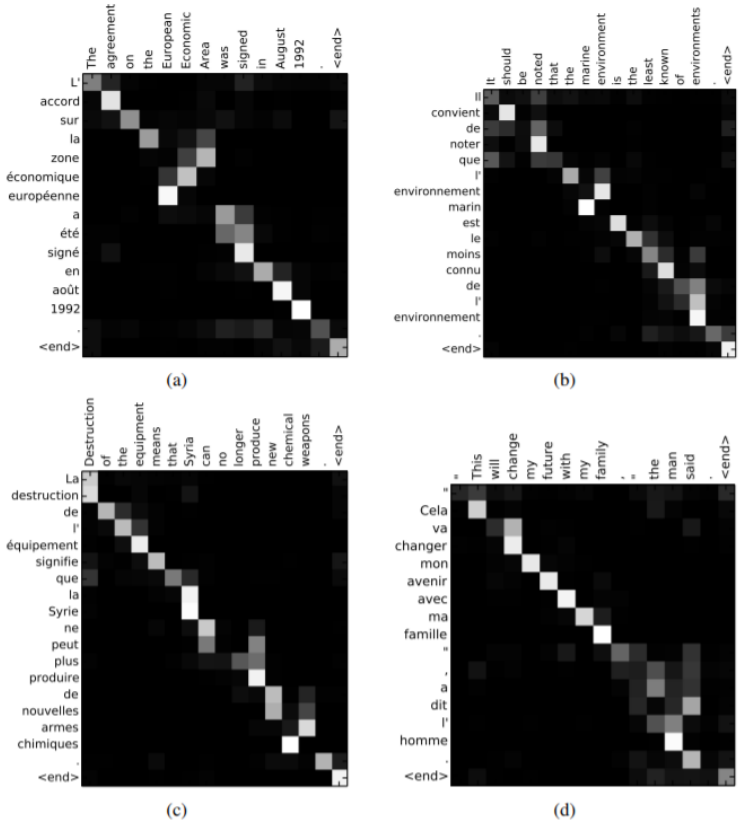
h_t: *encoder hidden state*, s_t: *decoder hidden state*

K.H. Cho, “Neural machine translation by jointly learning to align and translate”

Attention Mechanism



Attention mechanism application



Where to give attention

감사합니다