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

1부. 자연어 처리 기초

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

Konlpy Setting

```
pip install --upgrade pip

pip install JPype1-0.6.2-cp36-cp36m-win_amd64.whl

(jdk installed & path setting)

pip install konlpy
```

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’

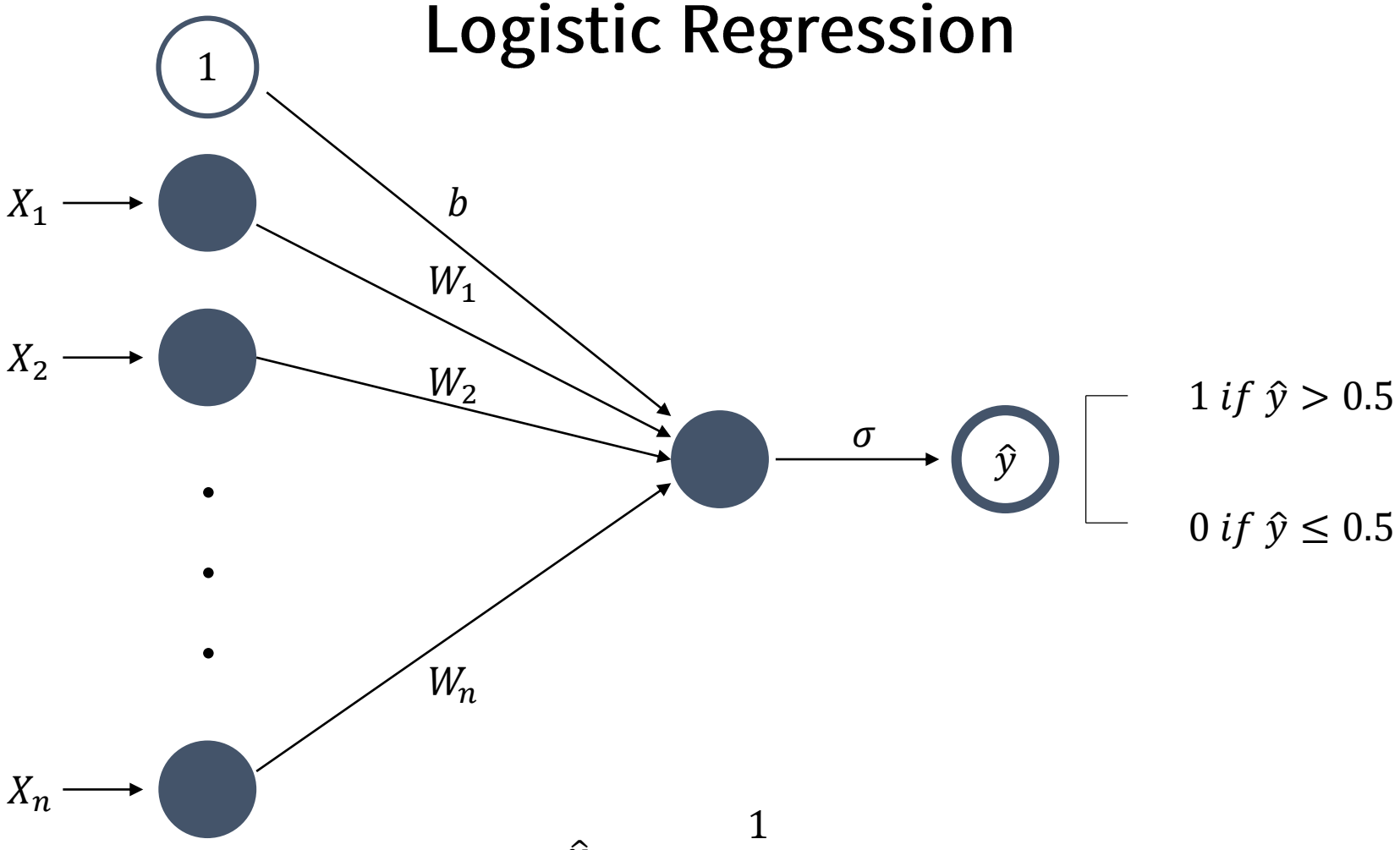
사전 = {영화: 0, 진짜:1, 너무:2, …, 인생: 8}

“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’

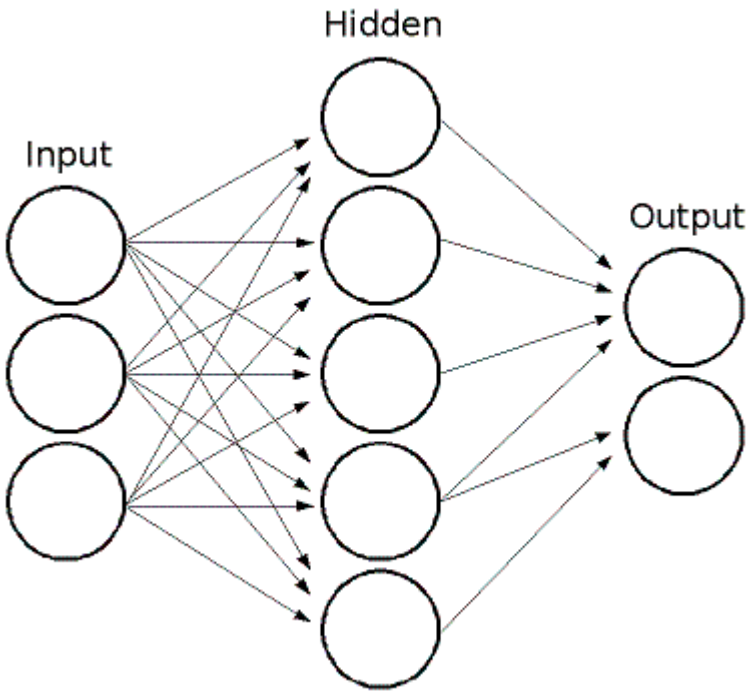
전처리

“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

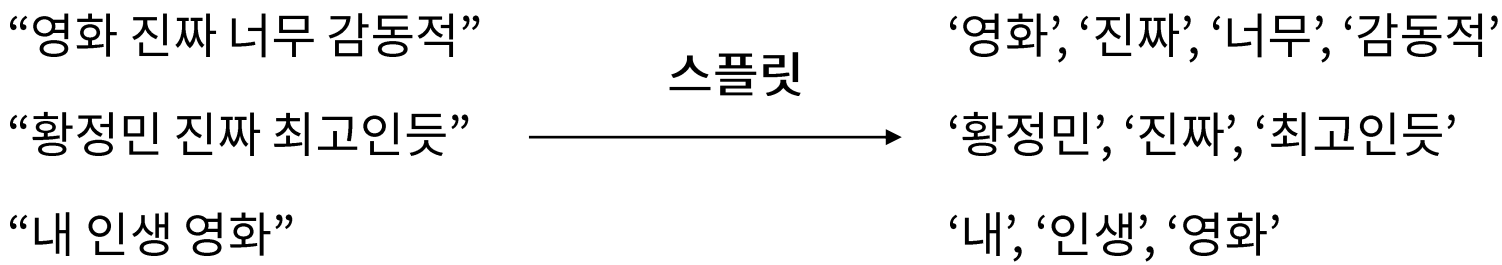
tf.train.GradientDescentOptimizer: 최적화 도구

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? → 단어 간 상관관계를 구할 수 없고, 차원이 너무 커짐!

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

차원이 커짐에 따라 계산 비용이 너무 비싸진다.

one-hot encoding 으로는 문맥적, 의미론적 정보를 담을 수 없다.

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

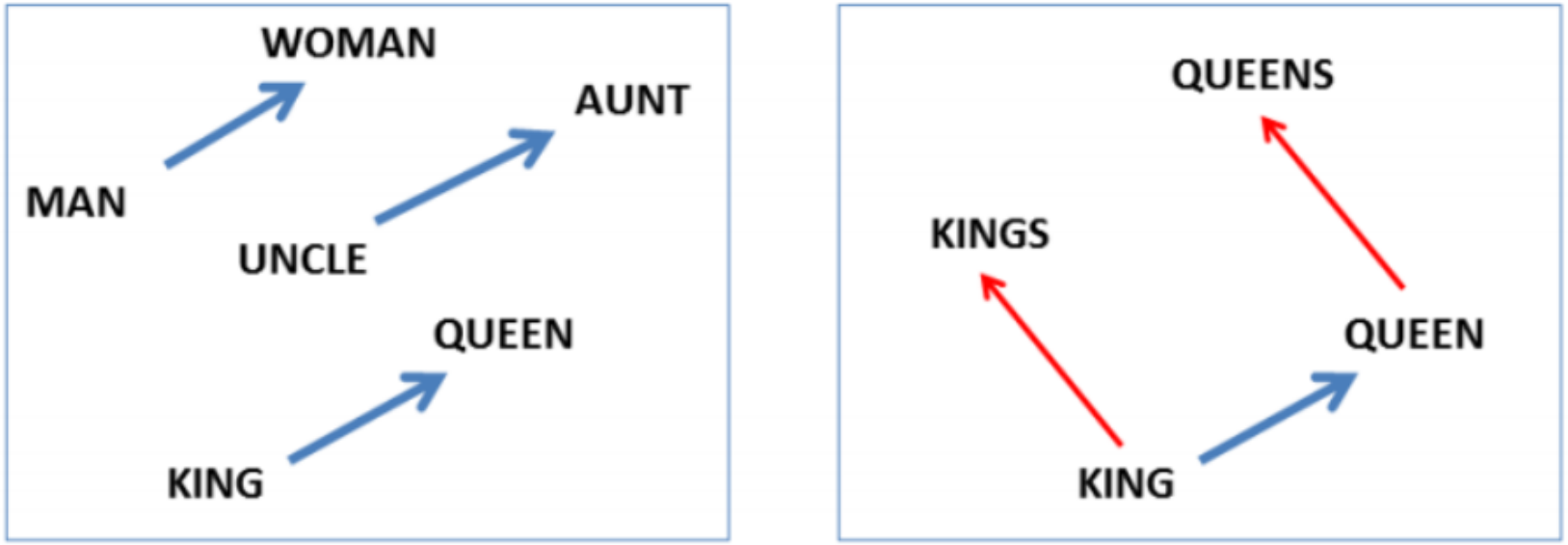
‘비슷한 분포를 가진 단어들은 비슷한 의미를 가진다’ 는
언어학의 distributional hypothesis 에 입각

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)

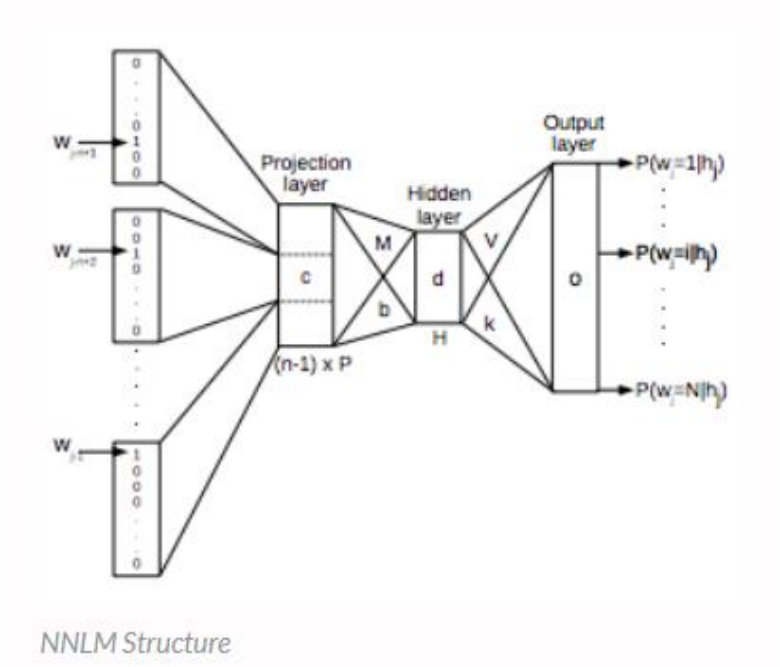
http://w.elnn.kr/search/

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.

Word Projection: NxP

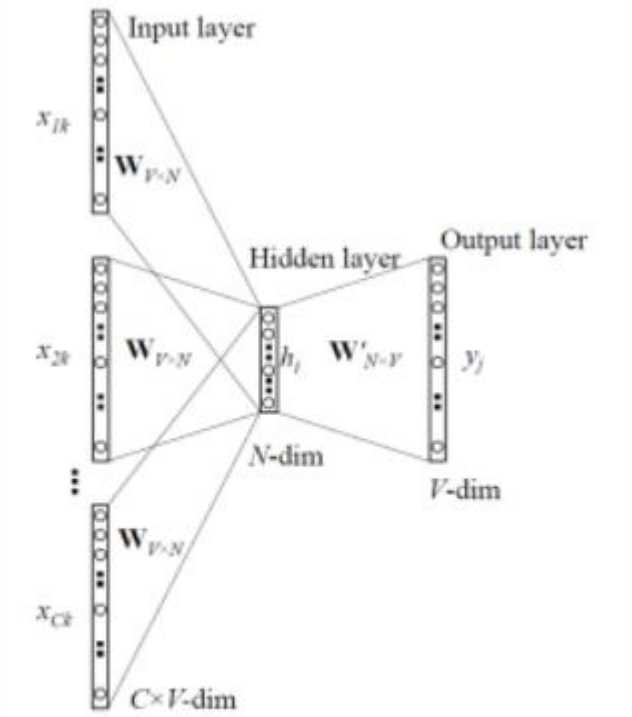
Projection Layer -> Hidden Layer: NxPxH

Hidden Layer -> Output Layer: HxV hierarchical softmax 를 사용하면 Hxln(V)

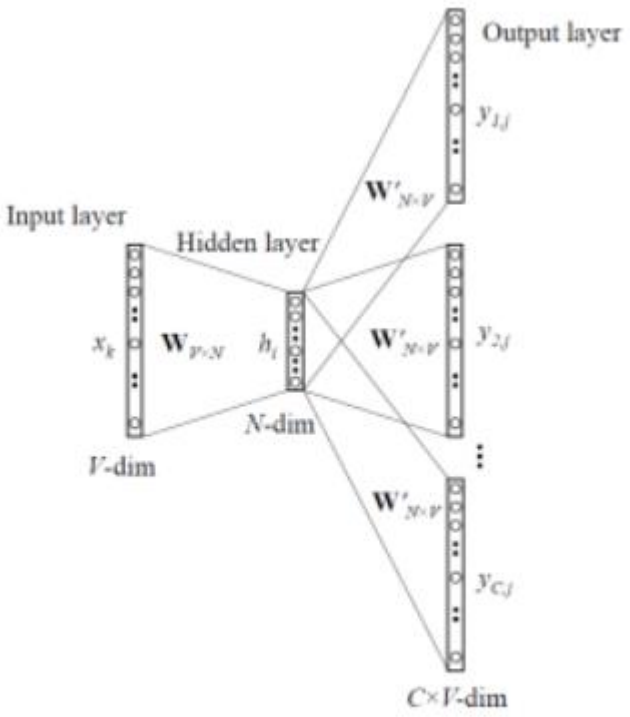
즉, NxP + NxPxH + Hxln(V) 만큼의 시간이 소요

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

Word2Vec



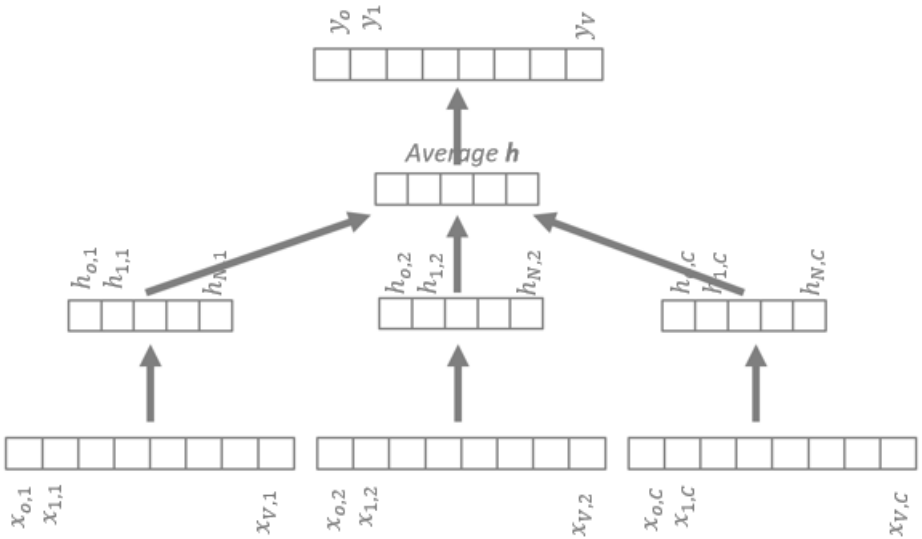
CBOW Architecture



Skip-gram Architecture

https://www.knime.com/blog/word-embedding-word2vec-explained

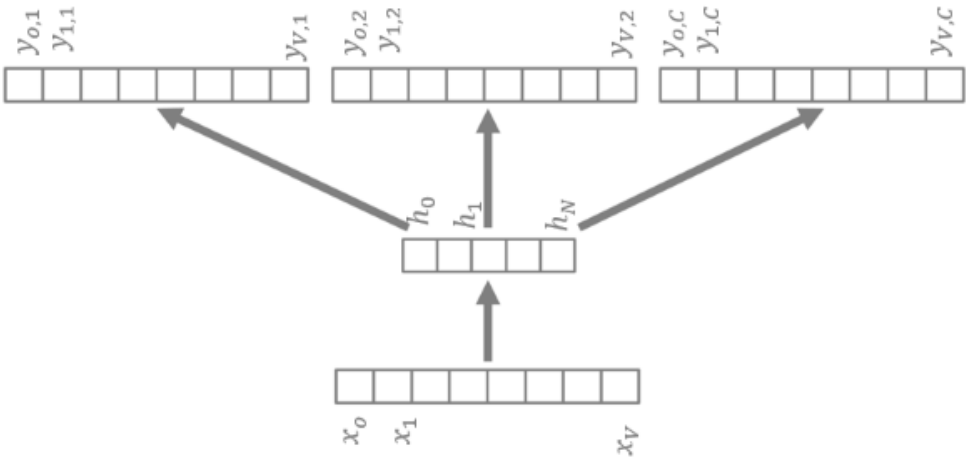
CBOW model



주어진 단어 앞 뒤로 2/C개 씩 총 C개의 단어를 Input 으로 이용하여 target 단어를 맞춤

https://www.knime.com/blog/word-embedding-word2vec-explained

Skip-gram model



주어진 단어를 Input word 로 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, hierarchical softmax 를 사용하면 N x ln V
- 총 C개의 단어에 대해 진행해야 하므로 총 C x N x ln V 의 연산이 필요

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

Skip-gram model

Negative sampling

Noise Constrastive Estimation (NCE)를 간소화 시킨 목적 함수를 가짐
NCE 는 언어 모델 예측 문제를 확률 이진 분류기의 parameter 를 학습하는 문제로 전환
1개의 positive sample 과 k개의 negative samples 을 통해 학습

Subsampling Frequent Words

크기가 엄청 큰 corpus 의 경우에, 가장 빈도가 높은 단어들은 수 억 번 이상 반복되어
나오기 때문에 희박하게 나오는 단어와 자주 나오는 단어의 불균형을 해소하기 위해 도입

$$P(w_i) = 1 - \sqrt{\frac{t}{f(w_i)}}, where\ f(w_i)\ is\ frequency\ and\ t\ is\ threshold\ (typically\ 10^{-5})$$

$f(w_i) > t$ 인 경우에 대해서만 적용하여 빈도 순위는 유지하되 공격적으로 subsampling

자세한 내용은 다음 논문들을 참조하세요.

Distributed Representations of Sentences and Documents

https://arxiv.org/pdf/1405.4053.pdf

word2vec Explained: Deriving Mikolov et al.'s Negative-Sampling Word-Embedding Method

https://arxiv.org/pdf/1402.3722.pdf

Distributed Representations of Words and Phrases and their Compositionality

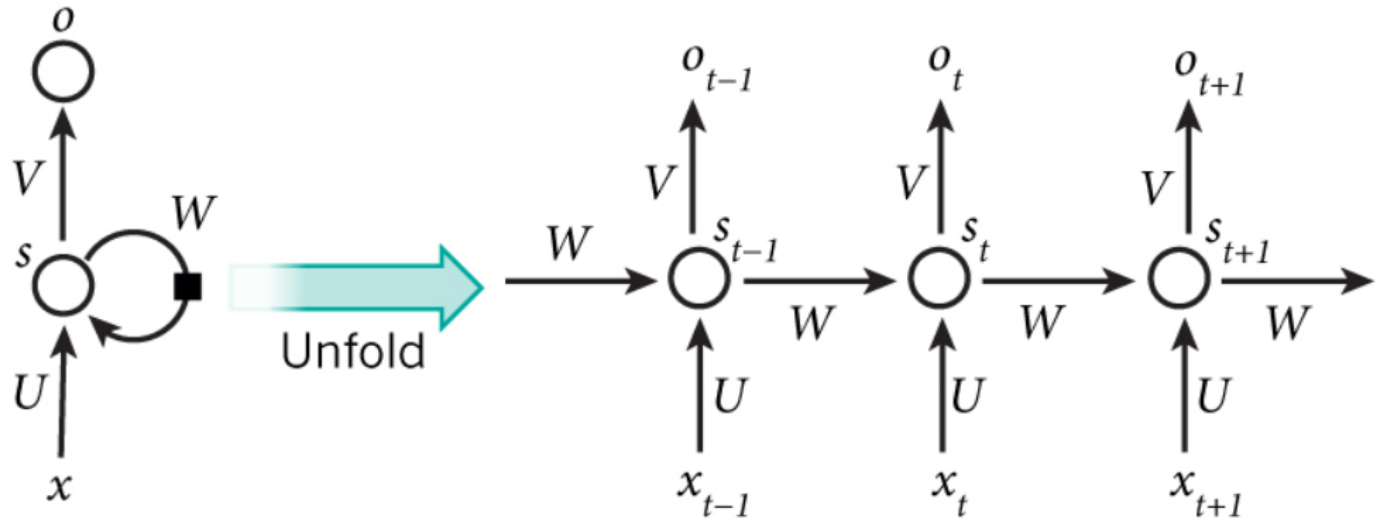
https://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf

Word2Vec 실습

2부. 딥러닝 자연어 처리 알고리즘

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

RNN (Recurrent Neural Network)



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

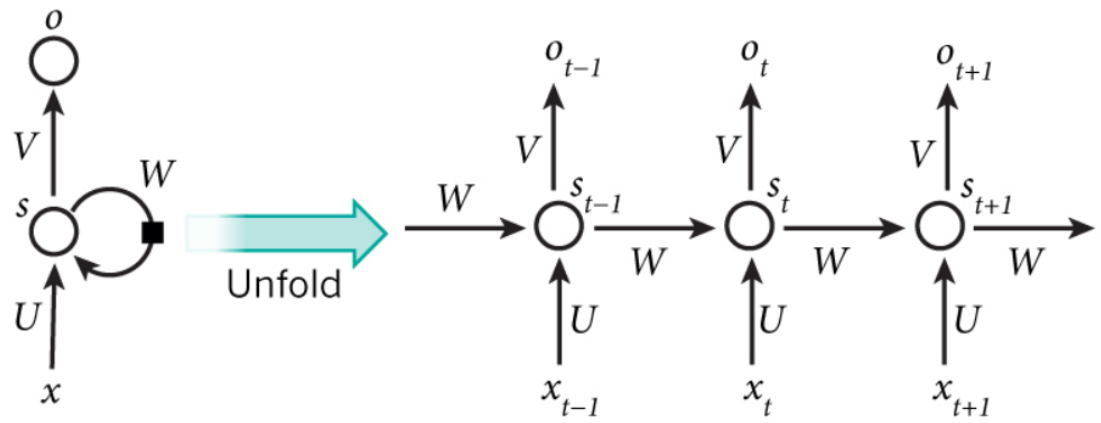
$$s_t = \tanh(U_s \cdot x_t + W_s \cdot s_{t-1} + b_s) \quad x_t \in \mathbf{R}^n, s_t \in \mathbf{R}^m, U_s \in \mathbf{R}^{n \times m}, W_s \in \mathbf{R}^{m \times m}$$

$$s_t = \tanh(W \cdot [s_{t-1}, x_t] + b_s) \qquad W \in \mathbf{R}^{(m+n) \times m}, b_s \in \mathbf{R}^m$$

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

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

RNN (Recurrent Neural Network)



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

- Hidden state s_t 는 previous step에서 발생한 정보 또한 다룰 수 있으며 output state인 o_t 는 오직 s_t를 이용하여 계산이 이루어진다.
- 기존 전통 딥러닝과는 다르게, RNN은 각 time step이 같은 parameter를 공유한다.
- 위 그림에서는 output state를 각 time step마다 계산하여 도출하지만 task에 따라서는 모든 output state가 꼭 필요로 되지 않는다. 예를 들어 감성분석의 경우, 우리는 각 단어 이후의 감성이 아닌 모든 단어를 본 뒤의 마지막 감성 즉, 마지막 state에 대해서만 관심이 있다.

RNN 실습

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

RNN (Recurrent Neural Network)

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

→ Gradient 소실 또는 폭발 문제 발생 때문

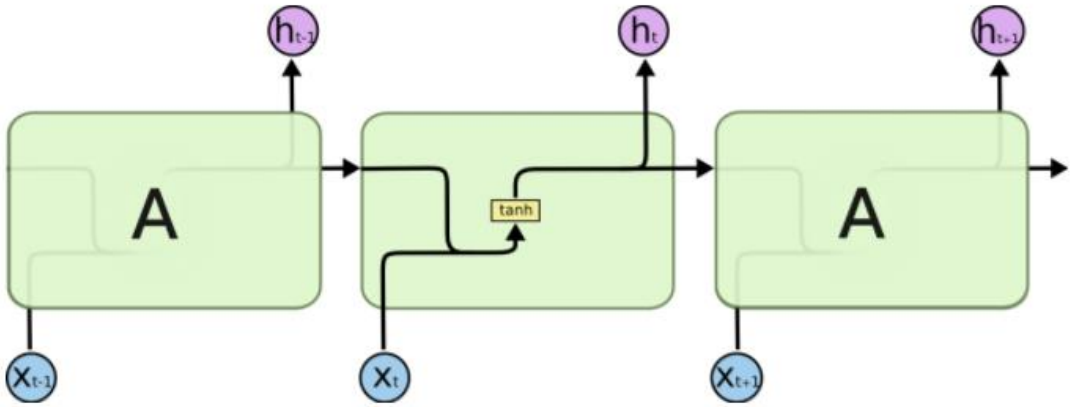
$$\frac{\partial L}{\partial W_h} = \sum_{t} \sum_{k=1}^{t+1} \frac{\partial L(t+1)}{\partial z_{t+1}} \cdot \frac{\partial z_{t+1}}{\partial h_{t+1}} \cdot \prod_{i=k}^{t} \left(\frac{\partial h_{i+1}}{\partial h_i}\right) \cdot \frac{\partial h_k}{\partial W_h}$$

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

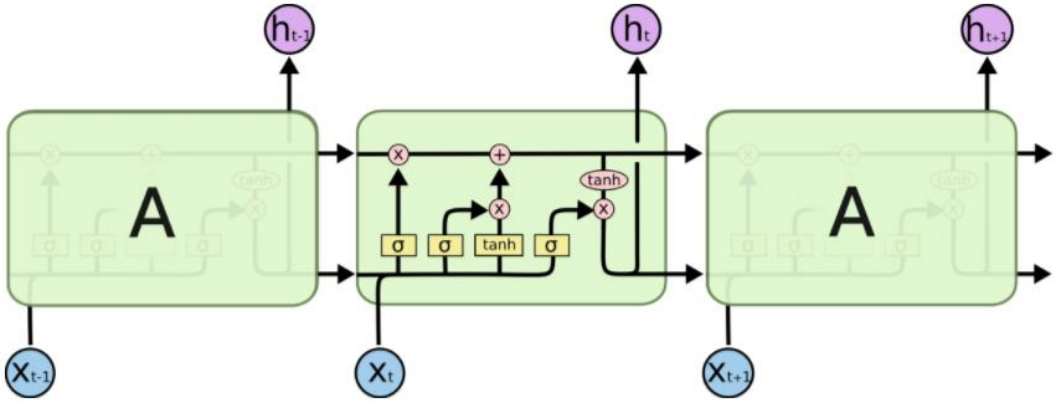
$$\frac{\partial h_{t+1}}{\partial h_{t-k+1}} = \prod_{j=1}^{k} \{(1 - h_{t-j+2}^2)\} \cdot W_h^k$$

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

LSTM (Long Short-Term Memory)



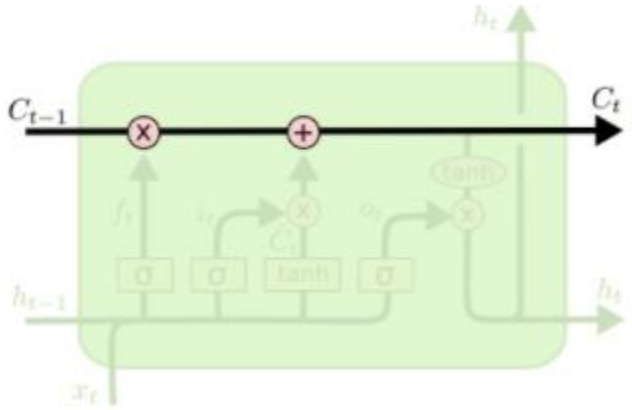
The repeating module in a standard RNN contains a single layer.



The repeating module in an LSTM contains four interacting layers.

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

LSTM (Long Short-Term Memory)

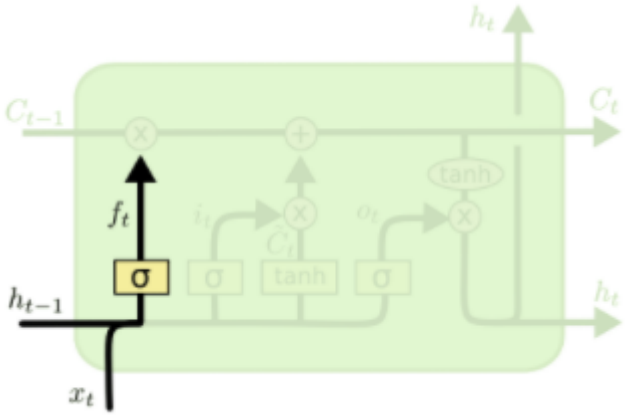


- LSTM의 핵심은 위 그림에 나와있는 cell state 에 있다!
- 위 cell state를 보면 비선형 구조가 없이 단순한 선형 연산으로 이루어져 있는 것을 볼 수 있다.
- Gate 라는 구조를 통해 정보의 선별적 학습이 가능하게 된다.

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

LSTM (Long Short-Term Memory)

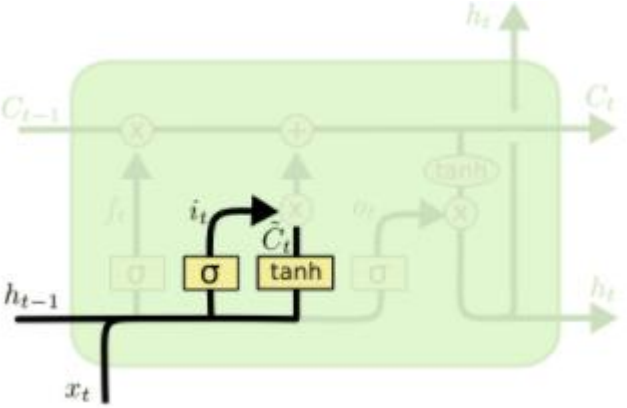
1. forget gate: previous hidden state 에서 어떤 정보를 잊을 것인지 결정



$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

- (0, 1) 사이의 값으로 이루어진 vector

2. input gate: new candidate hidden state에서 어떤 정보를 취할 것인지 결정



$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

- (0, 1) 사이의 값으로 이루어진 vector

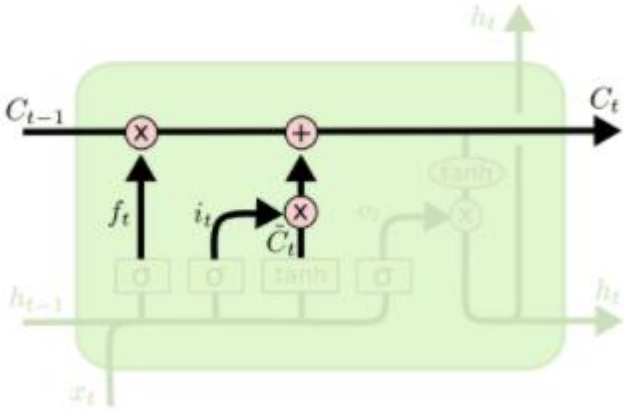
$$\tilde{C}_t = tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

- new candidate state

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

LSTM (Long Short-Term Memory)

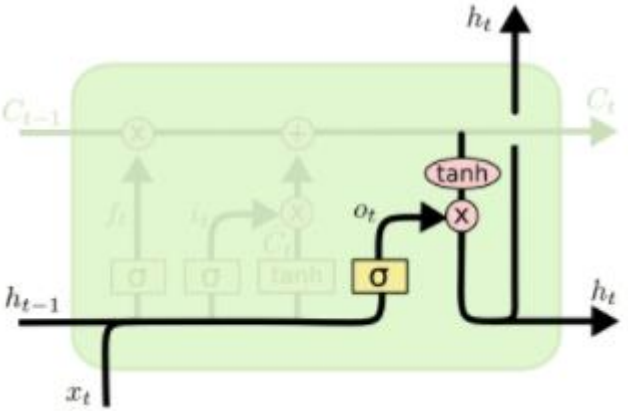
3. forget gate 및 input gate를 통한 new hidden state 계산



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

- $*$ 는 point-wise multiplication

4. output gate 를 통한 output state (hidden state 2) 결정



$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

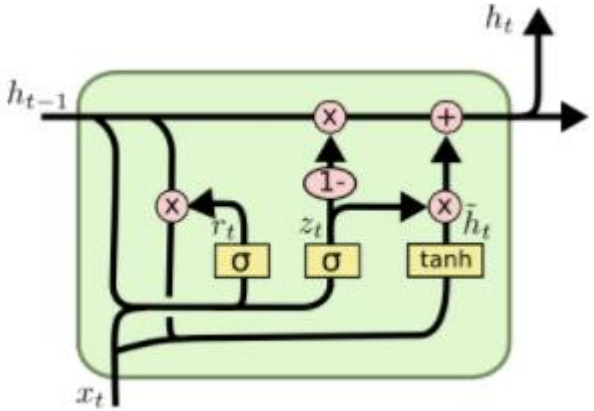
- (0, 1) 사이의 값으로 이루어진 vector

$$h_t = o_t * tanh(C_t)$$

- output state at time step t

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

GRU (Gated Recurrent Unit)



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

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

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

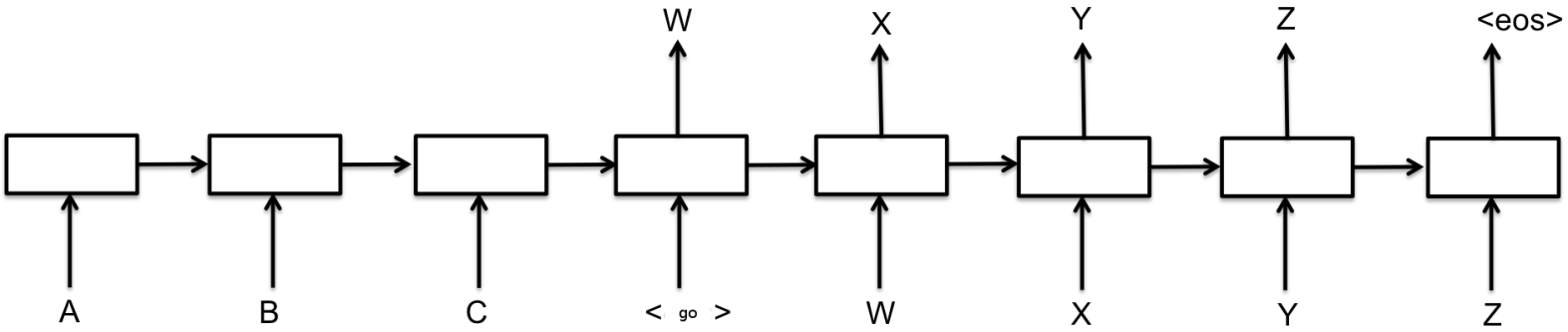
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

- z_t: LSTM에서의 forget gate 와 input gate 를 합친 update gate.
  두 연산을 하나로 합침으로써 연산을 줄일 수 있다.
- r_t: reset gate 로서 previous hidden state 에서 얼마나 정보를
  잊을 것인지 결정한다.
- 위 두 vector 는 모두 sigmoid 연산을 통해 (0, 1) 사이의 값을 가진다.

LSTM/GRU 실습

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

Sequence to sequence model



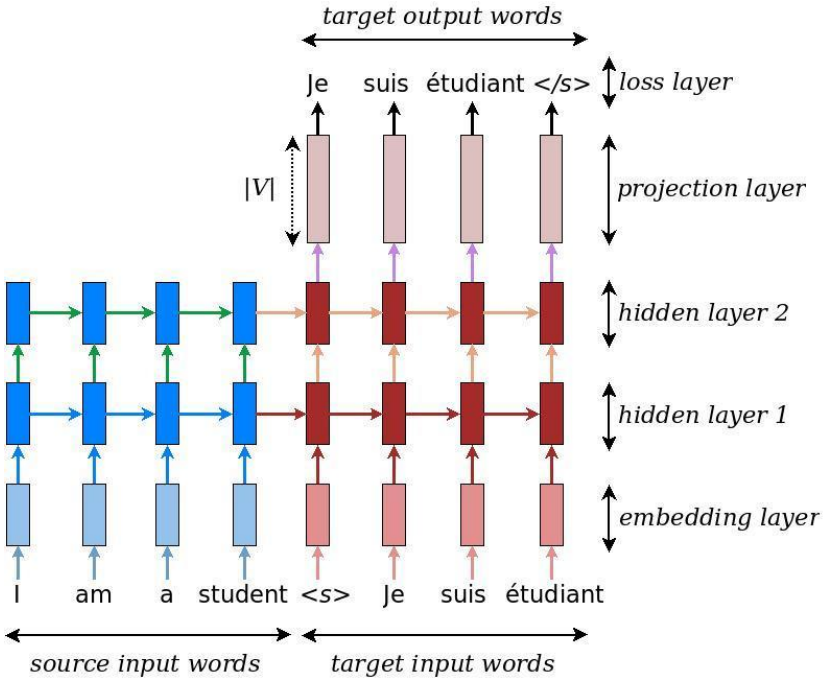
Encoder-decoder architecture

한 RNN 이 한 sequence of symbol 들을 고정 길이 벡터에 인코딩하고,
다른 RNN으로 다른 sequence of symbol 들로 디코딩한다.

seq2seq 실습

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

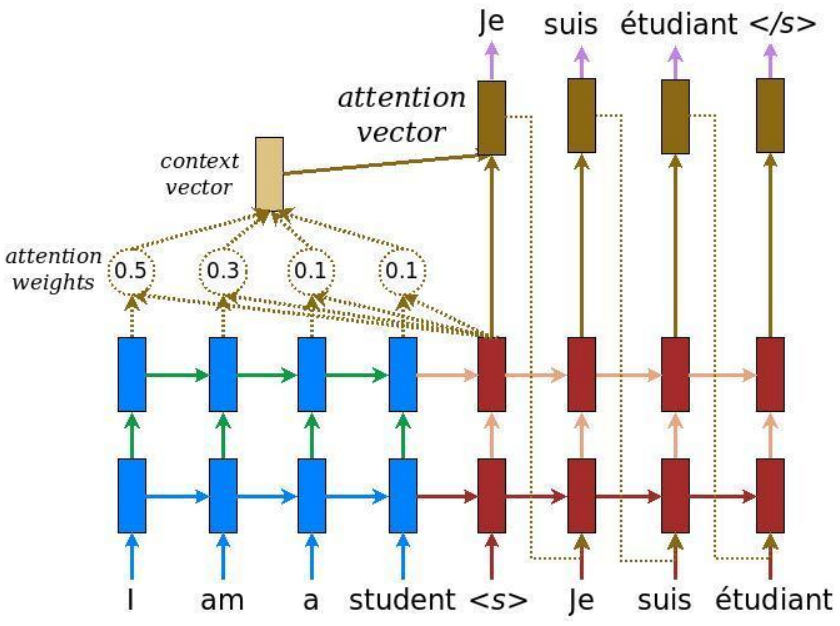
Seq2seq model Problem



Seq2seq model 에서는 encoder의 마지막 hidden state (a single fixed length vector) 에 모든 단어의 정보를 함축한다고 가정하지만, 문장의 길이가 길어지는 경우에, 그 많은 정보가 한 벡터에 모두 저장이 될 것이라고 생각하기 어렵다.

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

Attention Mechanism



<Attention mechanism>

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

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

이제 y_i 은 각각 distinct context vector c_i 에 의해 계산되므로 seq2seq 모델의 한계를 극복할 수 있다.

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

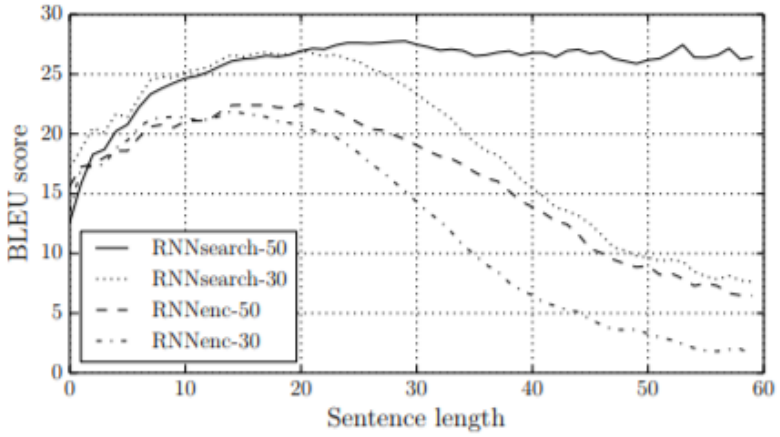
$$e_{ij} = a(s_{i-1},h_j)$$

*위 논문에서는 a 는 feedforward neural network 로 설정되어있음.

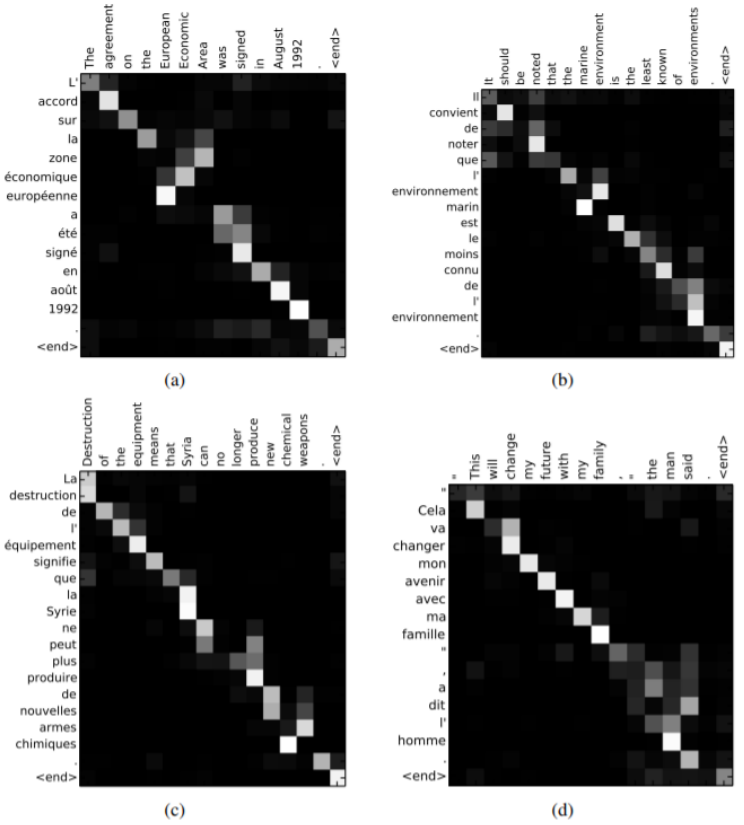
h_t: encoder hidden state, s_t: decoder hidden state

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

Attention Mechanism



<Attention mechanism application>



<Where to give attention>

Attention 실습

Any Question?

감사합니다

Email: minho@metafact.org
Github: https://github.com/Bricoler