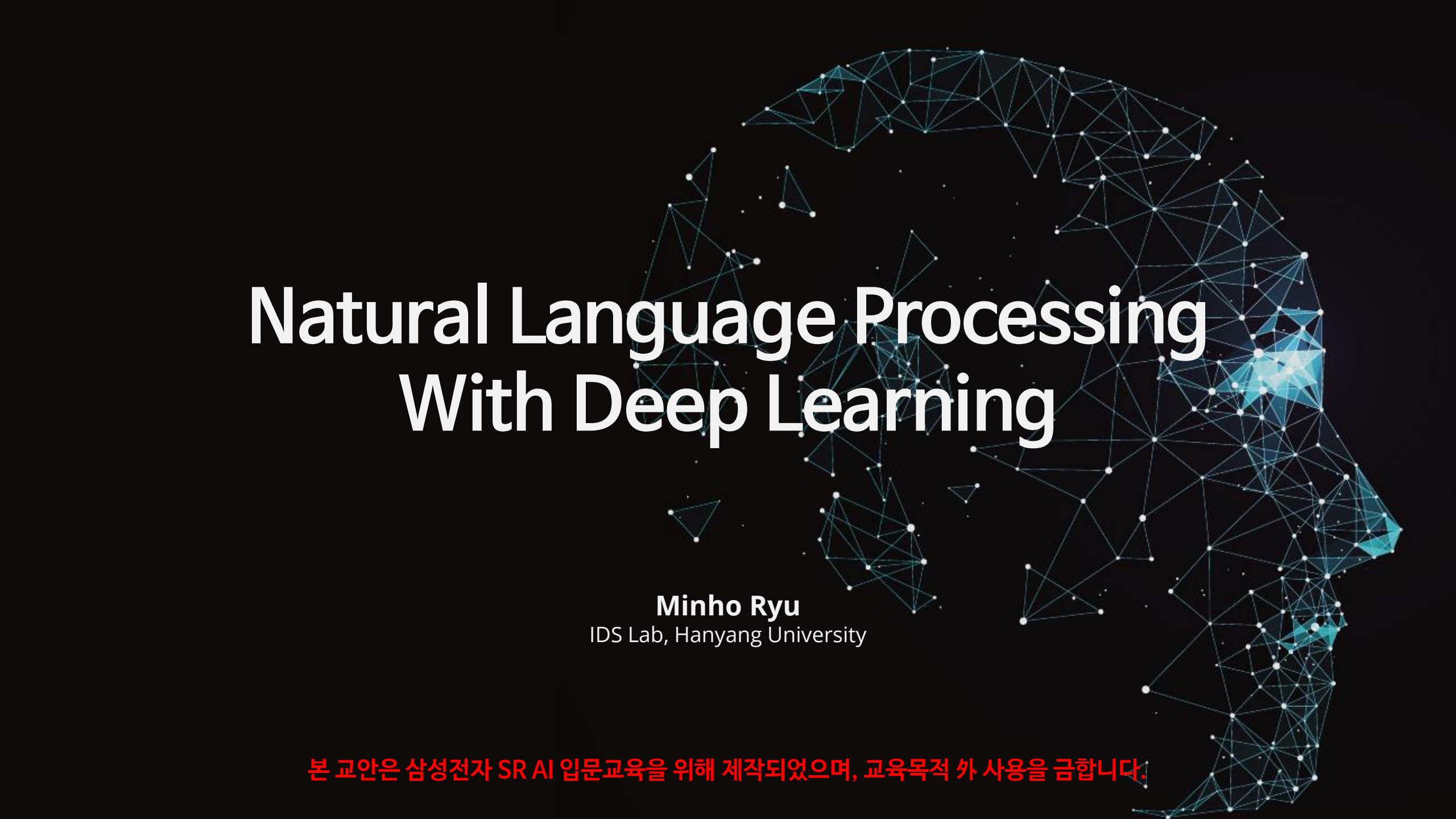


# Natural Language Processing With Deep Learning

**Minho Ryu**
IDS Lab, Hanyang University

# Minho Ryu

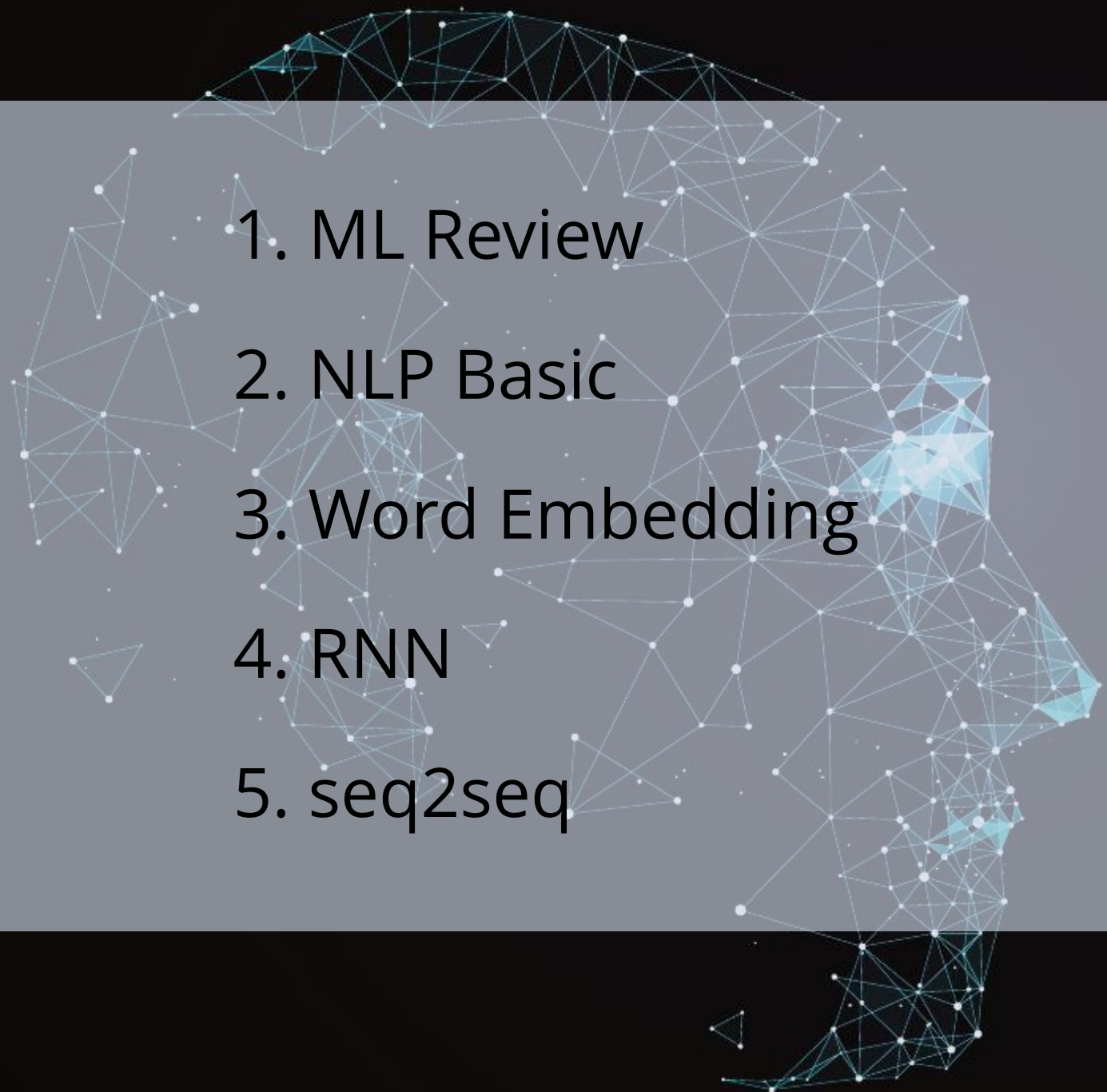
## Work Experience

- (現) Graduate Researcher at Hanyang University
- (現) Machine Learning Engineer in MetaFact
- Delegate for Youth Exchange Program in ADB
- IT Instructor at TRCSL in Sri-Lanka

## 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)

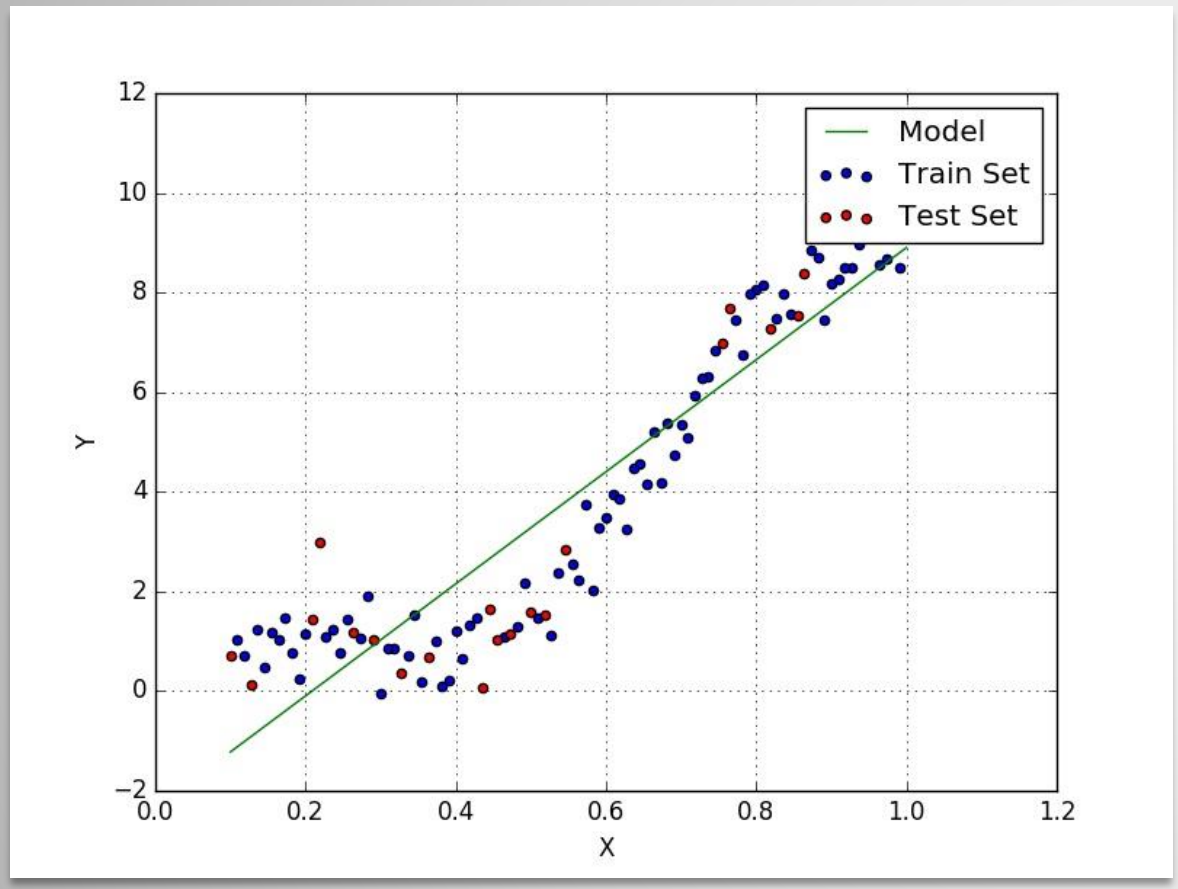
# Index



1. ML Review
2. NLP Basic
3. Word Embedding
4. RNN
5. seq2seq

# 1. ML Review

# 1. ML Review – **Linear Regression**




선형회귀는 데이터 X와 라벨 Y의 상관관계를 선형적 관계로 가정하여 푸는 것

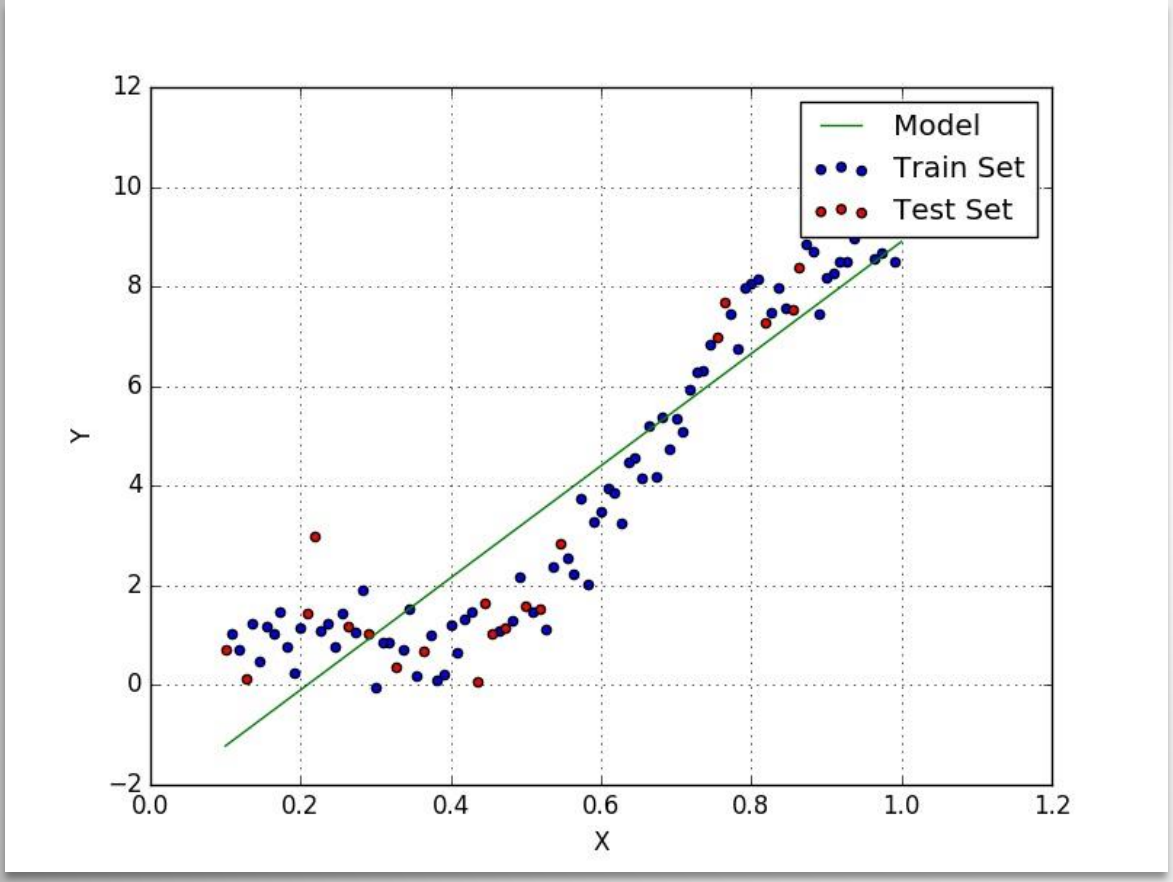
$$\hat{Y} = f(\boldsymbol{x}) = \beta_0 + \beta_1 x_1 + \cdots + \beta_n x_n$$

여기서 푼다는 것은 관계성을 나타내는 parameter 인 $\beta$를 구하는 것을 나타냄

선형회귀식에서는 보통 MSE를 사용!

$$\mathrm{MSE}_{\mathrm{train}} = \frac{1}{m}\sum_{i=1}^{m} \|y_i - \hat{y}_i\|^2$$

# 1. ML Review – **Linear Regression**



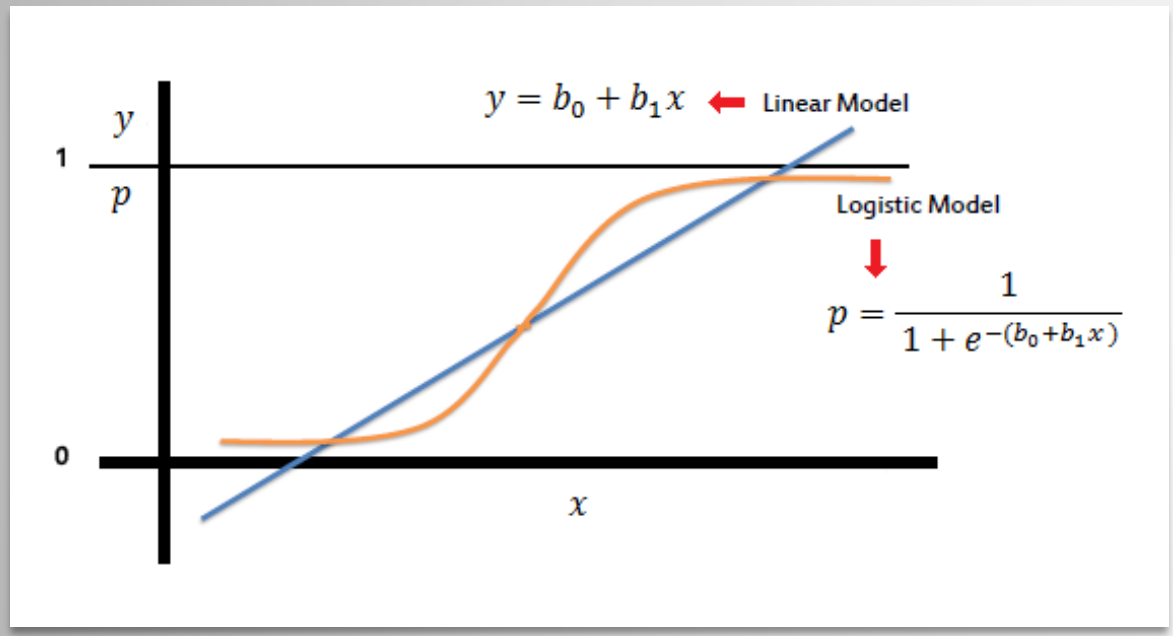

$$\mathrm{MSE}_{\mathrm{train}} = \frac{1}{m}\sum_{i=1}^{m}\|y_i - \hat{y}_i\|^2$$

위 식을 풀면,

$\beta = (X^T X)^{-1} X^T y$와 같이 closed form 이 존재!

$$X = \begin{pmatrix} 1 & x_{11} & \cdots & x_{1n} \\ 1 & x_{21} & \cdots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & \cdots & x_{nn} \end{pmatrix}, y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$

# 1. ML Review – **Logistic Regression**




로지스틱회귀는 선형회귀식에 로지스틱 함수를 사용하여 예측 값으로 0~1사이의 값을 가짐

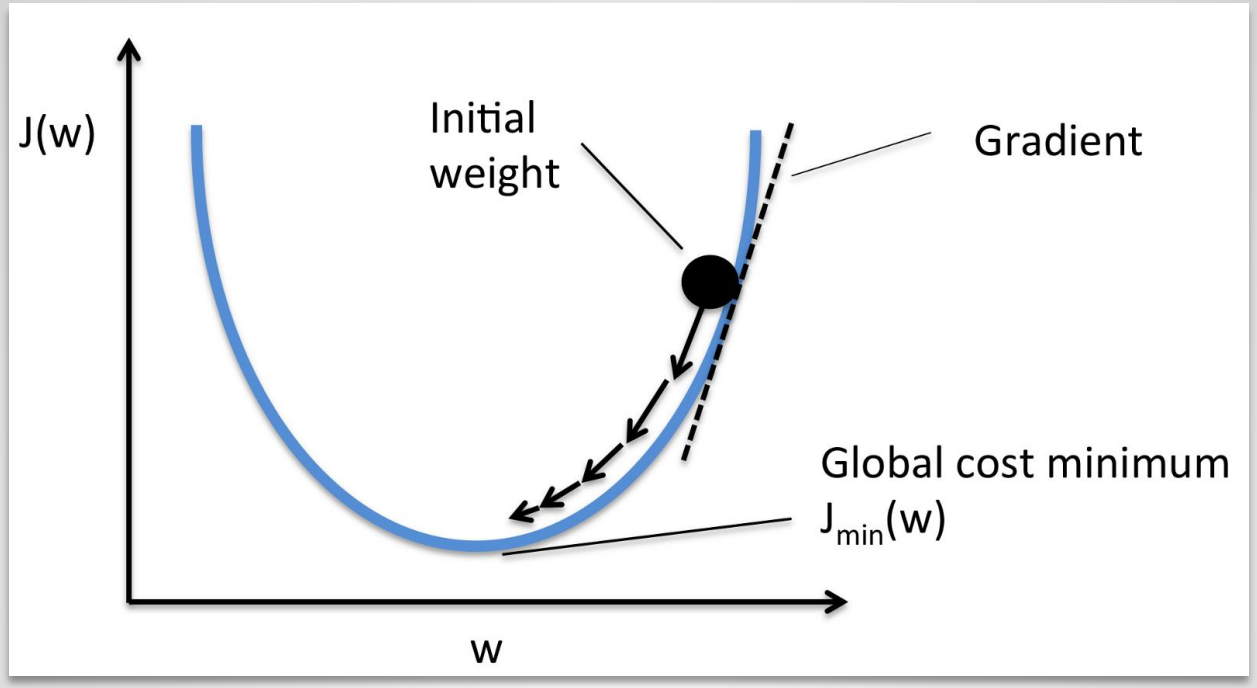
$$f(\boldsymbol{x}) = \beta_0 x_0 + \beta_1 x_1 + \cdots + \beta_n x_n$$

$$\hat{y} = \frac{1}{1 + e^{-f(\boldsymbol{x})}} = \frac{1}{1 + e^{-(\beta_0 x_0 + \beta_1 x_1 + \cdots + \beta_n x_n)}}$$

$$\mathcal{L}(y, \hat{y}) = -\frac{1}{m}\left(\sum_{i=1}^{m} (y_i log \hat{y}_i + (1 - y_i) log(1 - \hat{y}_i))\right)$$

→ Negative log likelihood 이용 (logistic cross entropy)

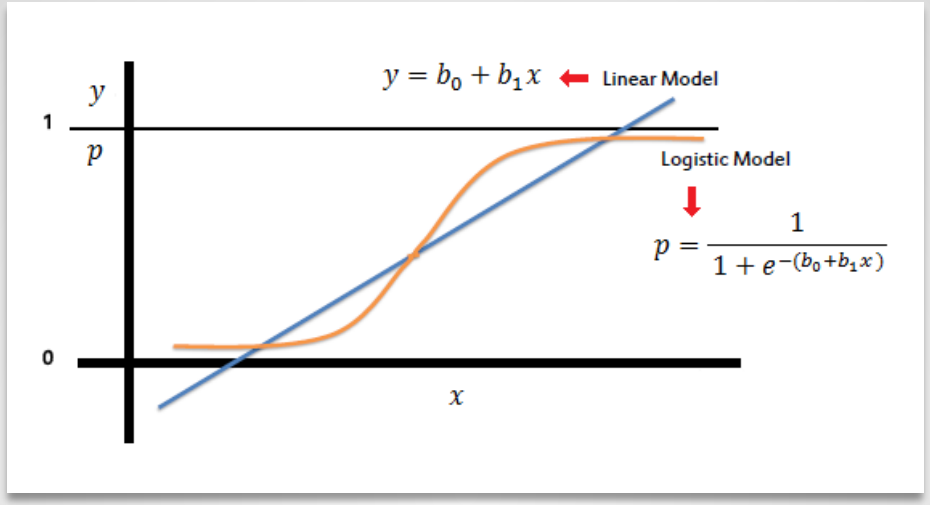
# 1. ML Review – **Gradient Descent Algorithm**




로지스틱회귀의 손실함수는 선형회귀의 손실함수와 다르게 closed form을 갖지 않기 때문에 numerical한 방법으로 구해야한다! → 대표적인 방법으로 **Gradient Descent Algorithm**

$$\theta^{(k+1)} = \theta^{(k)} - \alpha_k \frac{\partial L}{\partial \theta}$$

# 1. ML Review – **Chain Rule**



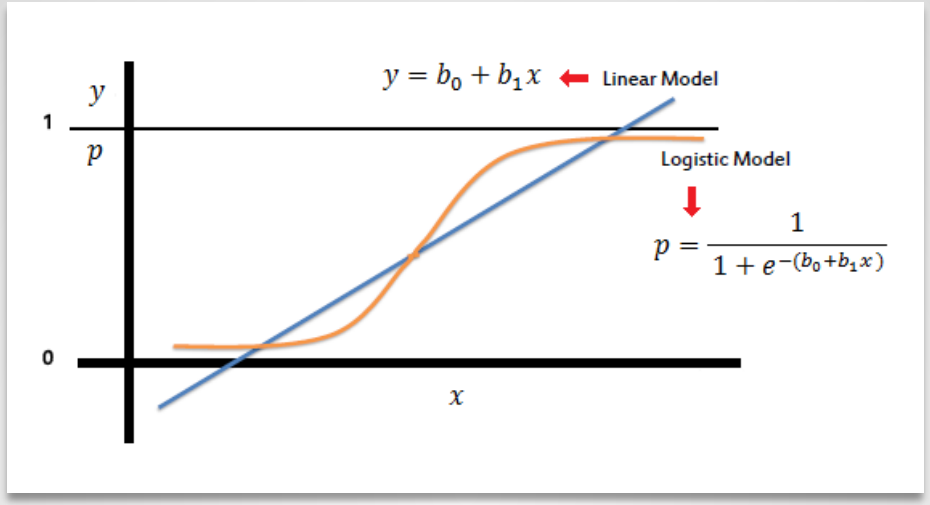

$$\mathcal{L}(y, \hat{y}) = -\frac{1}{m}\left(\sum_{i=1}^{m}(y_i log\hat{y}_i + (1 - y_i)log(1 - \hat{y}_i))\right)$$

$$\hat{y} = \frac{1}{1 + e^{-f(\boldsymbol{x})}}, \quad f(\boldsymbol{x}) = \beta_0 x_0 + \beta_1 x_1 + \cdots + \beta_n x_n$$

**Chain rule** 을 이용하면 복잡한 편미분식도 쉽게 구할 수 있다!

$$\frac{\partial \mathcal{L}}{\partial \beta_j} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial f} \times \frac{\partial f}{\partial \beta_j}$$

# 1. ML Review – **Chain Rule**




$$\frac{\partial \mathcal{L}}{\partial \hat{y}} = -\frac{1}{m}\sum_{i=1}^{m}\left(\frac{y_i}{\hat{y}_i} - \frac{(1-y_i)}{(1-\hat{y}_i)}\right), \qquad \frac{\partial \hat{y}}{\partial f} = \hat{y}(1-\hat{y}), \qquad \frac{\partial f}{\partial \beta_j} = x_j$$

$$\frac{\partial \mathcal{L}}{\partial \beta_j} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial f} \times \frac{\partial f(\boldsymbol{x})}{\partial \beta_j} = -\frac{1}{m}\sum_{i=1}^{m}\left(\frac{y_i}{\hat{y}_i} - \frac{(1-y_i)}{(1-\hat{y}_i)}\right) \times \hat{y}(1-\hat{y}) \times x_j = -\frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)x_j$$

$$\beta_j^{(k+1)} = \beta_j^{(k)} - \alpha_k \frac{\partial \mathcal{L}}{\partial \beta_j}, j = 0, \dots, n$$

# 1. ML Review – **Stochastic Gradient Algorithm**

$$J(\theta) = \mathbb{E}_{\boldsymbol{x},y\sim\hat{p}_{data}} L(\boldsymbol{x}, y, \boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m} \subset(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta})$$

$$\nabla_\theta J(\theta) = \mathbb{E}_{\boldsymbol{x},y\sim\hat{p}_{data}} L(\boldsymbol{x}, y, \boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m} \nabla_\theta \subset(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta})$$

만약에 데이터의 수가 너무 많다면? → gradient 를 계산할 때 비싼 컴퓨팅 비용이 필요 $(O(m))$

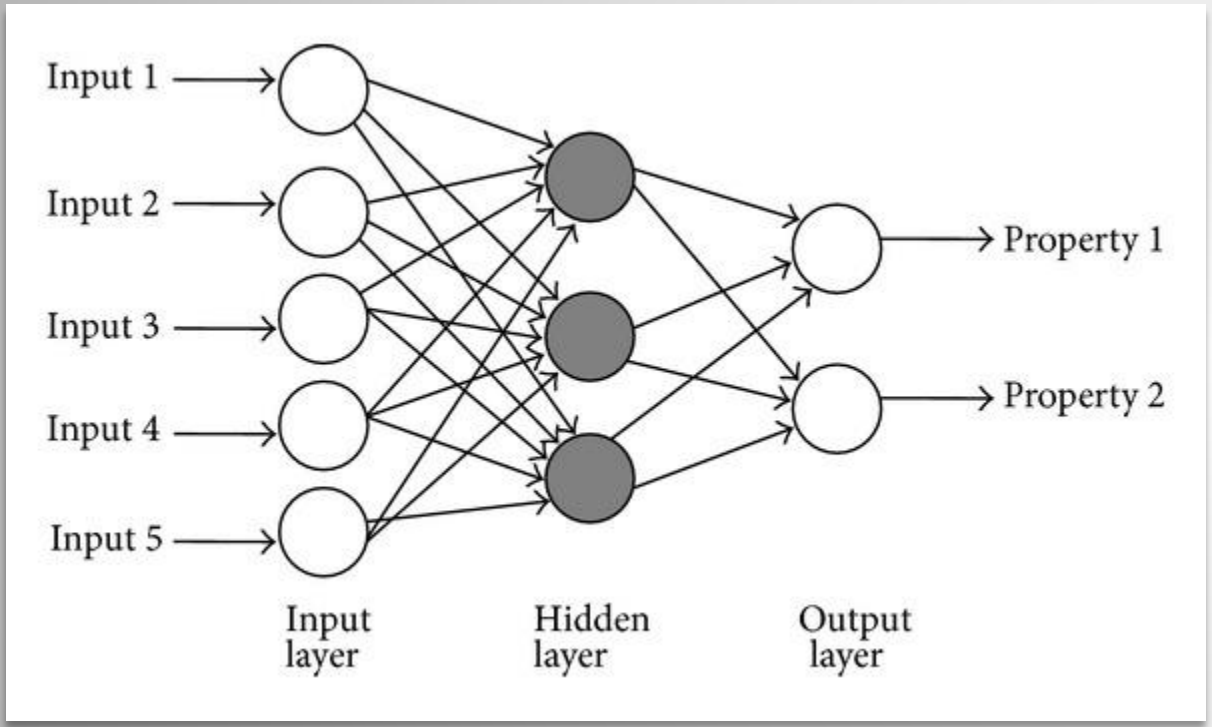
**Stochastic Gradient Descent** 는 위 gradient 식이 평균이라는 점에 주목!

$$\boldsymbol{g} = \frac{1}{m'}\sum_{i=1}^{m'} \nabla_\theta \subset(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta})$$

$$\theta^{(k+1)} = \theta^{(k)} - \alpha_k \boldsymbol{g}$$

**minibatch 만큼 sampling 하여 gradient 를 근사!**

# 1. ML Review – **Multi Layer Perceptron**




$$h = \sigma(XW_{xh} + b_h)$$

$$\hat{y} = softmax(hW_{ho} + b_o)$$

$$\mathcal{L}(y, \hat{y}) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{j} y_{ij} log \hat{y}_{ij}$$

Cost function으로 **Cross Entropy** 사용

Cross Entropy 는 두 확률 분포의 거리를 계산하는 방법

# 2. NLP Basic

# One-hot encoding

“영화 진짜 너무 감동적”
“황정민 진짜 최고인듯”
“내 인생 영화 오브 영화”

**스플릿** →

‘영화’, ‘진짜’, ‘너무’, ‘감동적’
‘황정민’, ‘진짜’, ‘최고인듯’
‘내’, ‘인생’, ‘영화’, ‘오브’, ‘영화’

사전 = {‘영화’: 0, ‘진짜’:1, ‘너무’:2, ‘감동적’:3, ‘황정민’:4, ‘최고인듯’:5, ‘내’:6, ‘인생’:7, ‘오브’:8}

**벡터화**

“영화” → [1,0,0,0,0,0,0,0,0]
“황정민” → [0,0,0,0,1,0,0,0,0]
“인생” → [0,0,0,0,0,0,0,1,0]

# One-hot encoding

"영화 진짜 너무 감동적" → '영화', '진짜', '너무', '감동적'

"황정민 진짜 최고인듯" → '황정민', '진짜', '최고인듯'

"내 인생 영화 오브 영화" → '내', '인생', '영화', '오브', '영화'

스플릿

사전 = {'영화': 0, '진짜': 1, '너무': 2, '감동적': 3, '황정민': 4, '최고인듯': 5, '내': 6, '인생': 7, '오브': 8}

벡터화

"영화 진짜 너무 감동적" → [1,1,1,1,0,0,0,0,0]

"황정민 진짜 최고인듯" → [0,1,0,0,1,1,0,0,0]

"내 인생 영화 오브 영화" → [2,0,0,0,0,0,1,1,1]

# ngram

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

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

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

사전 = {‘영화’: 0, ‘진짜’:1, ‘너무’:2, ‘감동적’:3, ‘영화 진짜’:4, ‘진짜 너무’:5,
‘너무 감동적’:6, ‘영화 진짜 너무’:7, ‘진짜 너무 감동적’:8}

# Minimum degree of freedom, Maximum degree of freedom

**min_df**: 단어 사전에 추가되기 위한 최소한의 등장 횟수를 설정

ex) 이영화진쨔뱔루, 겁나졸잼, suchagreatmovie

**max_df**: 일정 수준 이상으로 자주 발생하는 단어 사전에서 제외

ex) 가, 는, a, the

# Term Frequency-Inverse Document Frequency

*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-IDF 값이 높아지므로 이 값을 이용하면 모든 문서에 흔하게 나타나는 단어를 걸러내는 효과

## 실제 데이터는 지저분한 경우가 더 많다!

**스플릿**

“너무재밌었다그래서보는것을추천한닼ㅋ” → ‘너무재밌었다그래서보는것을추천한닼ㅋ’

**전처리**

“너무재밌었다그래서보는것을추천한닼ㅋ” → ‘너무’, ‘재미있다’, ‘그래서’, ‘보다’, ‘것’, ‘을’, ‘추천하다’

# NLP Basic 실습

## CountVectorizer

```
from sklearn.feature_extraction.text import CountVectorizer
vect = CountVectorizer(ngram_range=(1,1), binary=True).fit(train.document.values.astype('U'))
X_train = vect.transform(train.document.values.astype('U'))
X_test = vect.transform(test.document.values.astype('U'))
```

### Argument

- binary: True 이면, 문장 표현 시 0 또는 1로만 사용
- ngram_range=(1,1): unigram만 사용

### method

- CounterVectorizer.fit: 주어진 데이터 셋을 이용한 사전 구축
- CounterVectorizer.transform: 주어진 문장들을 0-1 벡터표현으로 변환

## TfidfVectorizer

```
from sklearn.feature_extraction.text import TfidfVectorizer
vect = TfidfVectorizer(ngram_range=(1,2), max_df=0.8, min_df=2).fit(train.document.values.astype('U'))
X_train_v2 = vect.transform(train.document.values.astype('U'))
X_test_v2 = vect.transform(test.document.values.astype('U'))
```

### Argument

- max_df/min_df: maximum/minimum degree of freedom
- ngram_range=(1,2): unigram에서 bigram까지 사용

### method

- TfidfVectorizer.fit: 주어진 데이터 셋을 이용한 사전 구축
- TfidfVectorizer.transform: 주어진 문장들을 tf-idf 벡터표현으로 변환

## konlpy.Twitter

```
from konlpy.tag import Twitter
twitter = Twitter()
```

```
print(train.loc[0, 'document'])
```

```
아 더빙.. 진짜 짜증나네요 목소리
```

```
print(twitter.morphs(train.loc[0, 'document'], stem=True, norm=True))
```

```
아 더빙.. 진짜 짜증나네요 목소리
['아', '더빙', '..', '진짜', '짜증', '나네', '요', '목소리']
```

```
print(twitter.pos(train.loc[0, 'document']))
```

```
[('아', 'Exclamation'), ('더빙', 'Noun'), ('..', 'Punctuation'), ('진짜', 'Noun'), ('짜증', 'Noun'), ('나네', 'Verb'), ('요', 'Eomi'), ('목소리', 'Noun')]
```

```
print(twitter.nouns(train.loc[0, 'document']))
```

```
['더빙', '진짜', '짜증', '목소리']
```

```
print(twitter.phrases(train.loc[0, 'document']))
```

```
['더빙', '진짜 짜증', '목소리', '진짜', '짜증']
```

**Twitter.morphs:** 형태소 분석

- norm: if True, 정규표현추출
- Stem: if True, 어근추출

**Twitter.pos:** part of speech tagging

- norm: if True, 정규표현추출
- Stem: if True, 어근추출

**Twitter.nouns:** 명사 추출

**Twitter.phrases:** phrases 추출

# 3. Word Embedding

# One-hot encoding

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

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

“내 인생 영화 오브 영화” → 스플릿 → ‘내’, ‘인생’, ‘영화’, ‘오브’, ‘영화’

사전 = {‘영화’: 0, ‘진짜’:1, ‘너무’:2, ‘감동적’:3, ‘황정민’:4, ‘최고인듯’:5, ‘내’:6, ‘인생’:7, ‘오브’:8}

영화 → [1,0,0,0,0,0,0,0,0]

진짜 → [0,1,0,0,0,0,0,0,0]

⋮

인생 → [0,0,0,0,0,0,0,1,0]

# 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  및 공간 비용이 너무 비싸진다.

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

# Distributed Representation of words

영화 ⟶ $[0.123, 0.643, \cdots, 0.212]$

진짜 ⟶ $[0.533, 0.186, \cdots, 0.935]$

⋮

인생 ⟶ $[0.864, 0.111, \cdots, 0.486]$

단어를 특정 차원의 실수 값을 가지는 분산 표현으로 잘 나타낼 수 있으면,

단어 간의 유사도와 단어의 문맥적 의미를 파악할 수 있지 않을까?

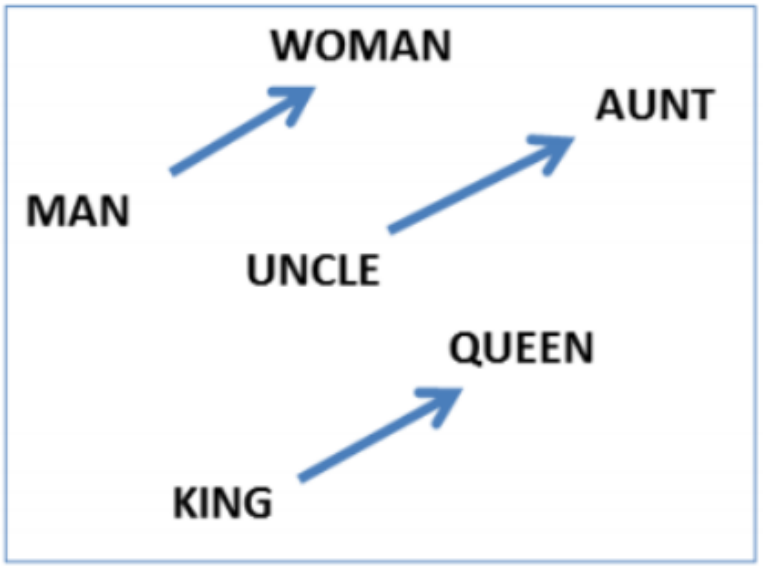
# Distributed Representation of words

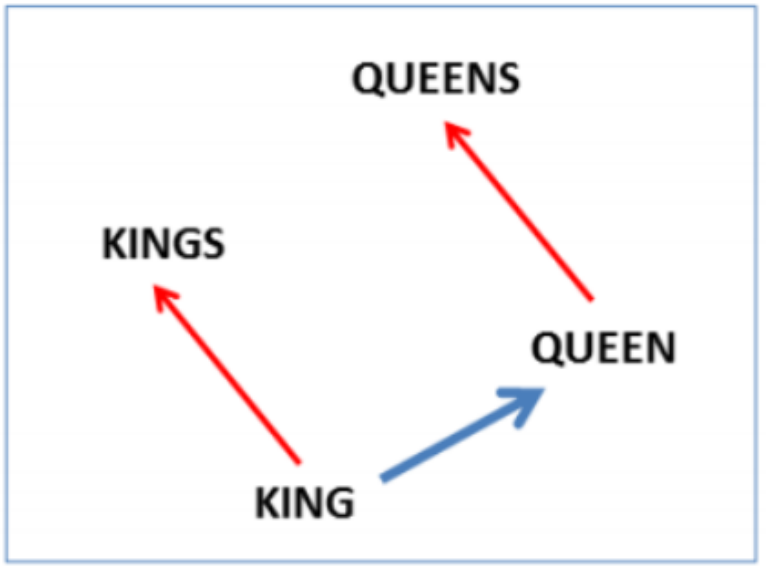
'비슷한 분포를 가진 단어들은 비슷한 의미를 가진다' 는 언어학의 distributional hypothesis 에 입각

비슷한 분포를 가진다는 것은 기본적으로 단어들이 같은 문맥에서 등장한다는 것을 의미

예를 들어, '사과', '포도', '딸기'라는 단어가 같은 맥락에서 등장하는 일이 빈번하게 일어난다면, 이 단어들이 유사한 의미를 가진 것으로 유추할 수 있다는 것

# Distributed Representation of words

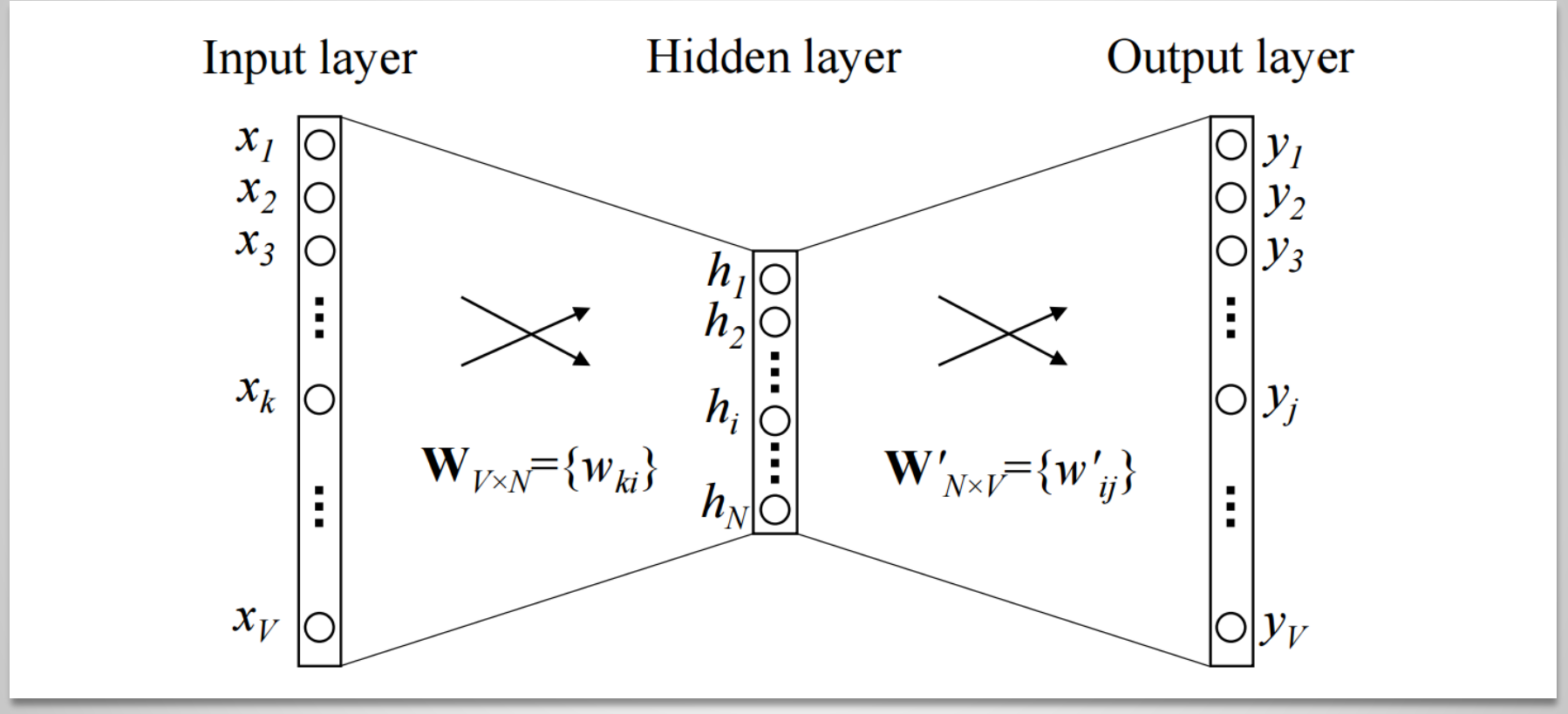



(Mikolov et al., NAACL HLT, 2013)

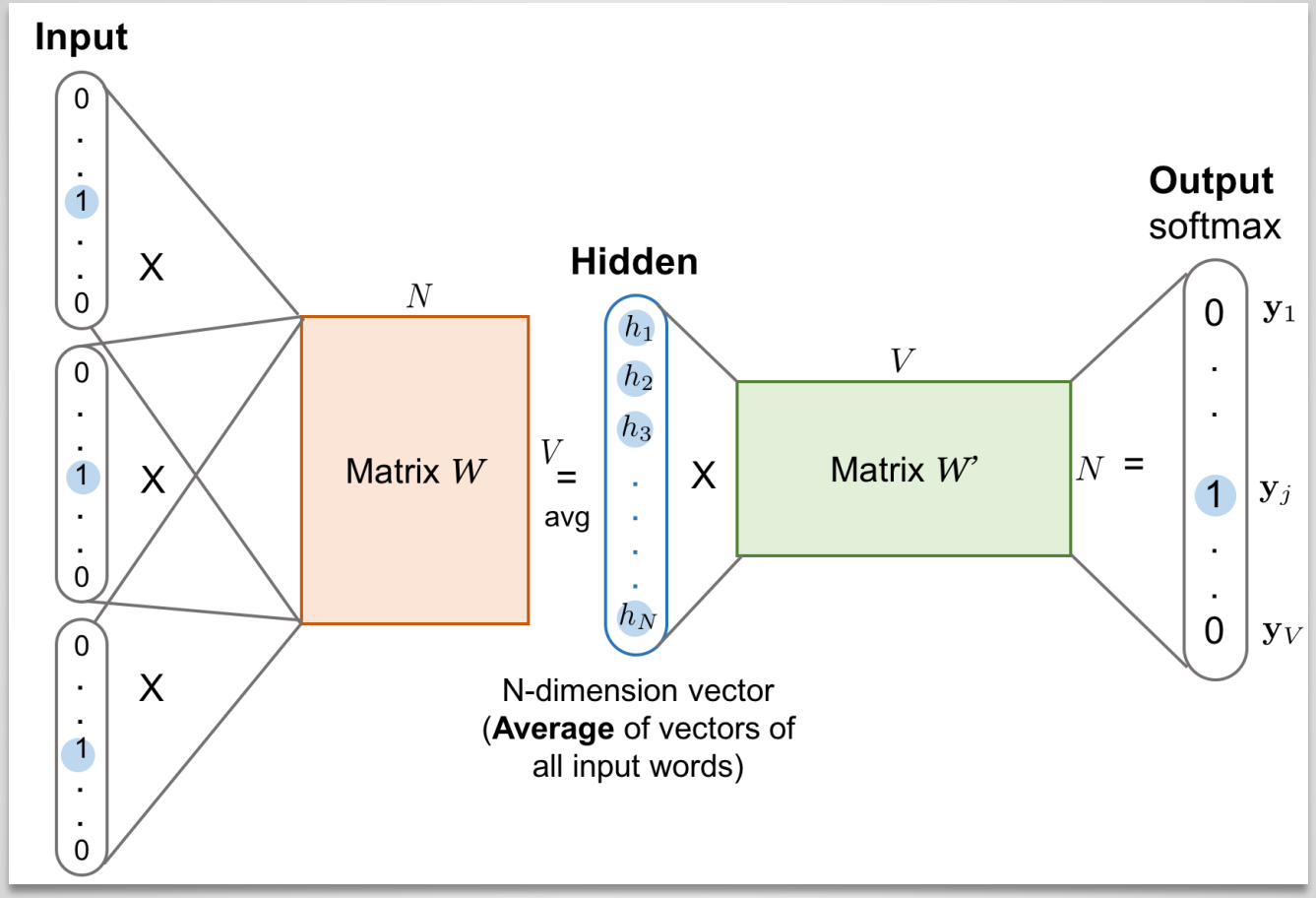
http://w.elnn.kr/search/

# 3. Word Embedding – **Word2Vec**




**Word2Vec:** Continuous Bag of Word/ Skip Gram Model

# 3. Word Embedding – **Word2Vec(CBOW)**

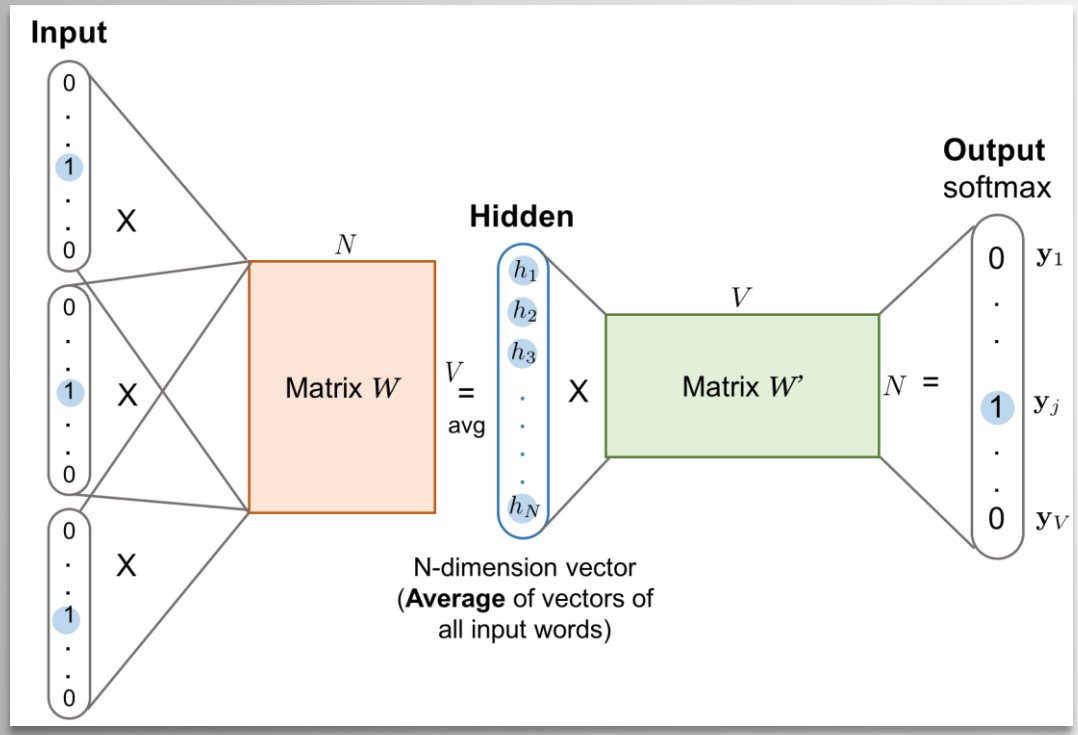



## Continuous Bag of Word

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

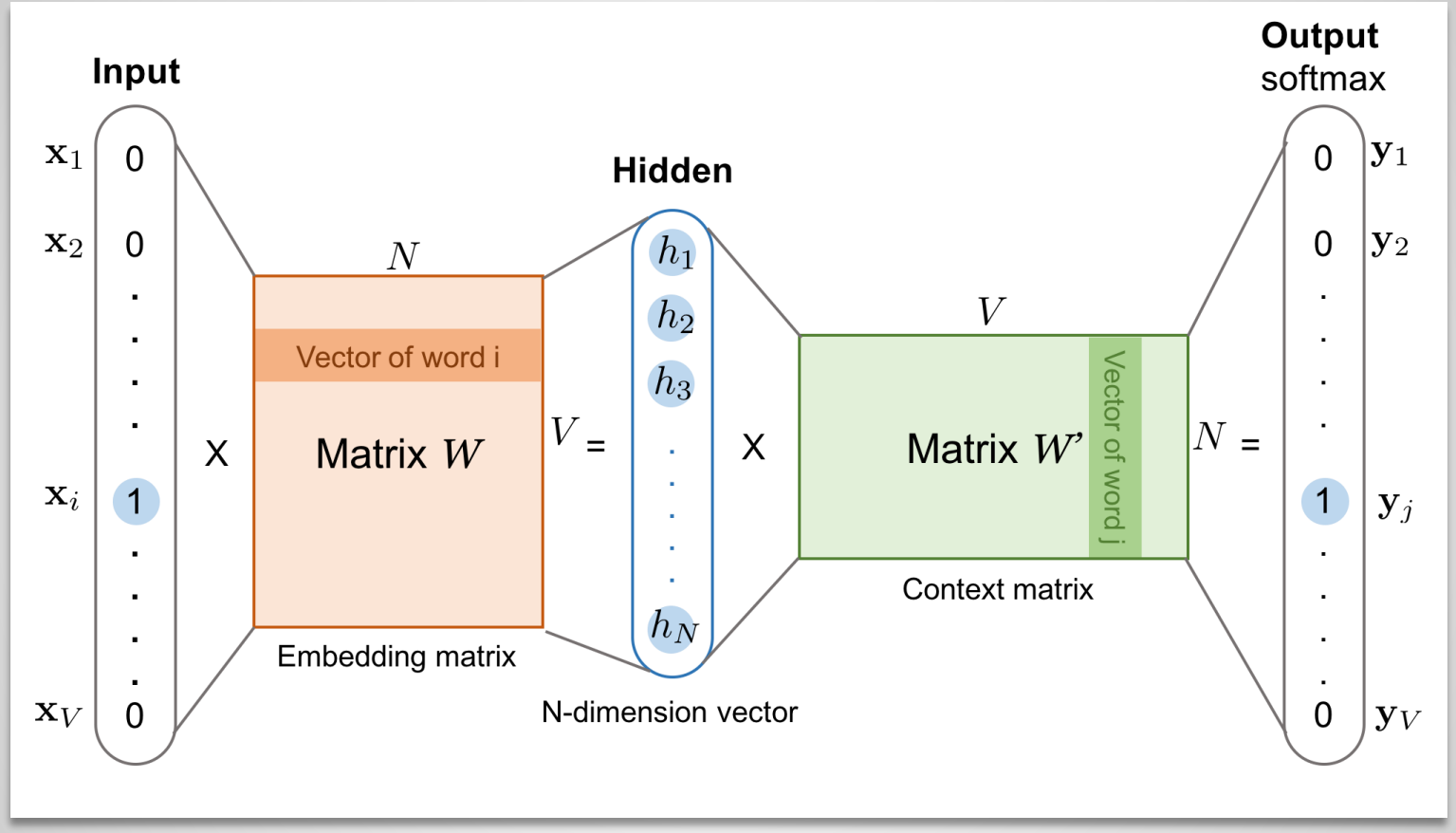
# 3. Word Embedding – **Word2Vec(CBOW)**

- **Continuous Bag of Word**




1. Generate one hot word vectors $(x_{c-m}, \dots, x_{c-1}, \dots, x_{c+1}, \dots, x_{c+m})$ for the input context of size $2m$
2. Get our embedded word vectors for the context $(v_{c-m} = Wx_{c-m}, \dots, v_{c+m} = Wx_{c+m})$
3. Average these vectors to get $\hat{v} = \frac{v_{c-m} + v_{c-m+1} + \cdots v_{c+m-1} + v_{c+m}}{2m}$
4. Generate a score vector $z = W'\hat{v}$
5. Turn the scores into probabilities $\hat{y} = softmax(z)$
6. Train embedded vectors using $H(\hat{y}, y) = -y_i \log(\hat{y}_i) = -\log(\hat{y}_i)$

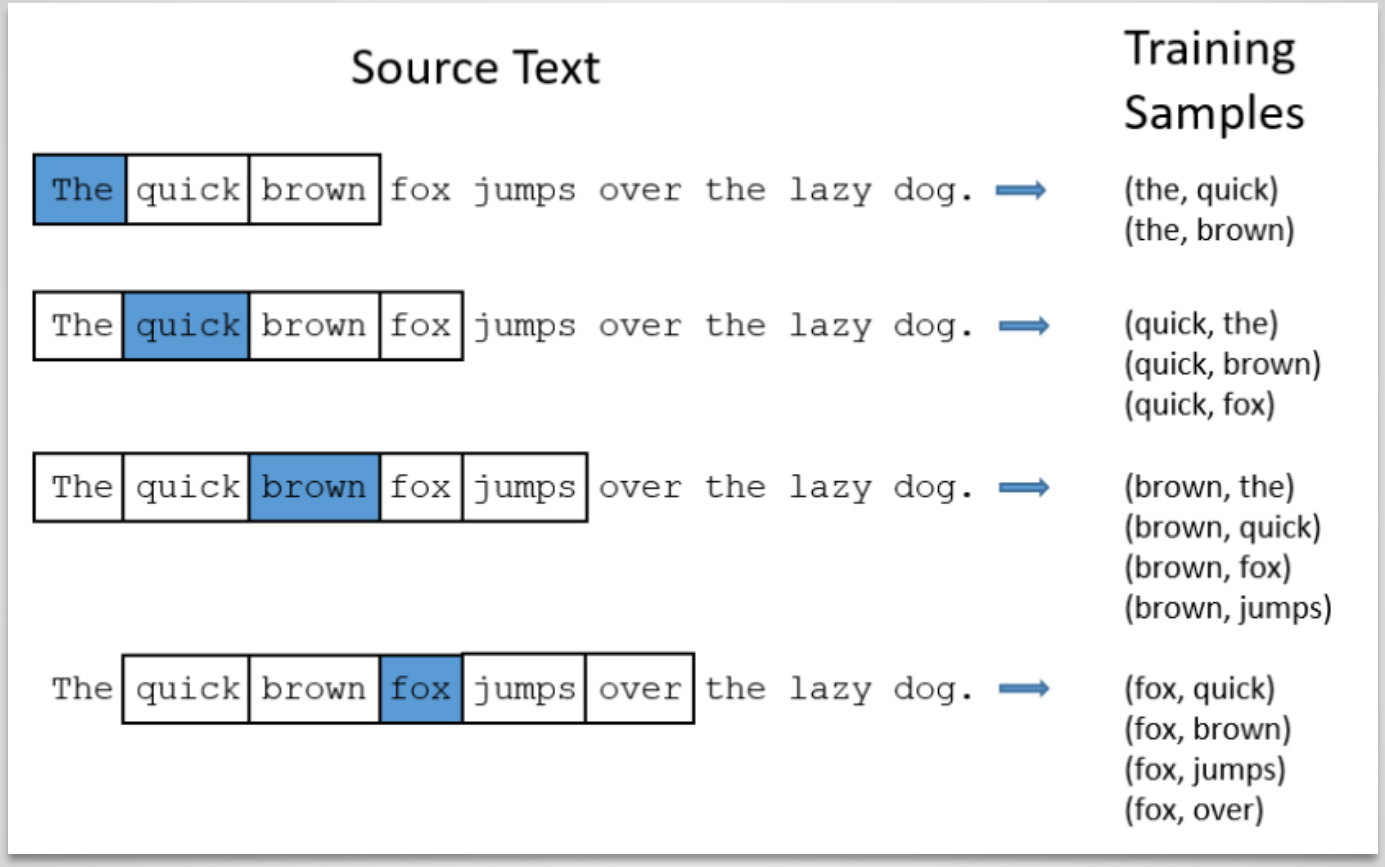
# 3. Word Embedding – **Word2Vec(Skip-Gram)**




## Skip-Gram Model

주어진 단어를 Input word 로 이용하여 주변 $m$개 단어를 예측

# 3. Word Embedding – **Word2Vec(Skip-Gram)**

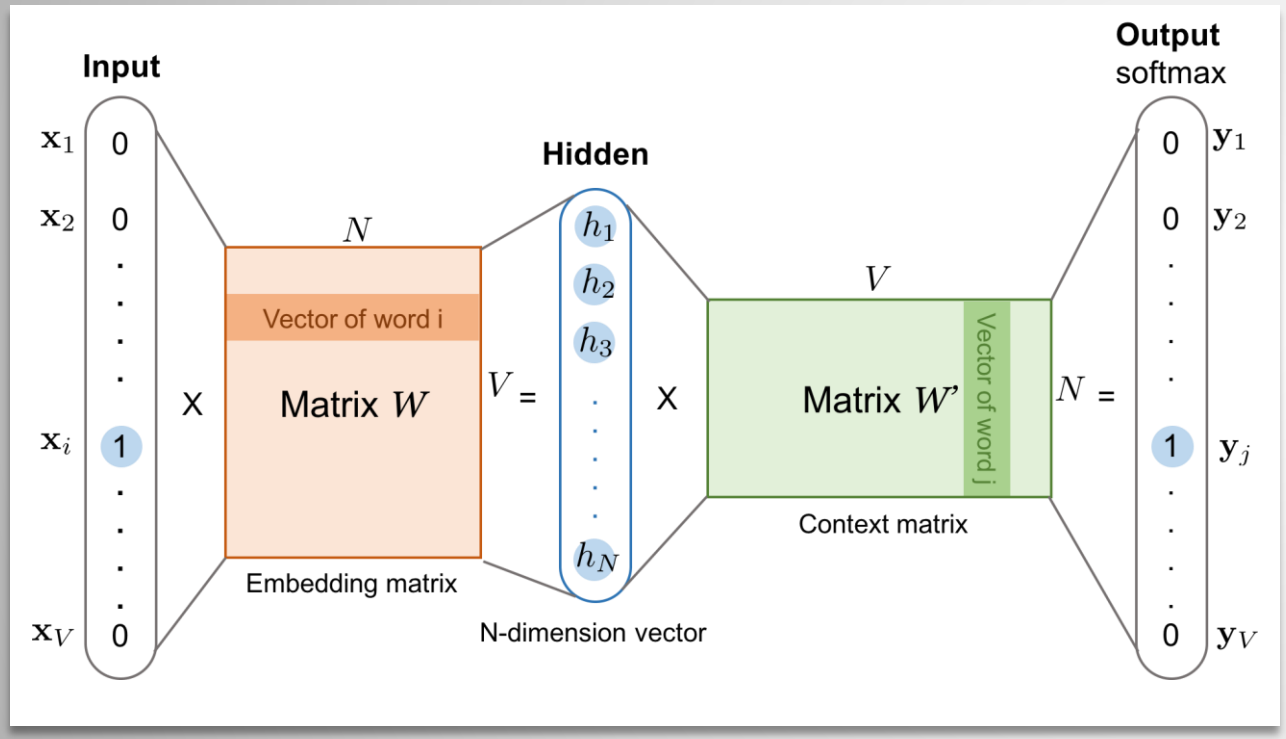



## Skip-Gram Model

주어진 단어를 Input word 로 이용하여 주변 $m$개 단어를 예측

# 3. Word Embedding – **Word2Vec(Skip-Gram)**

- **Skip-Gram**



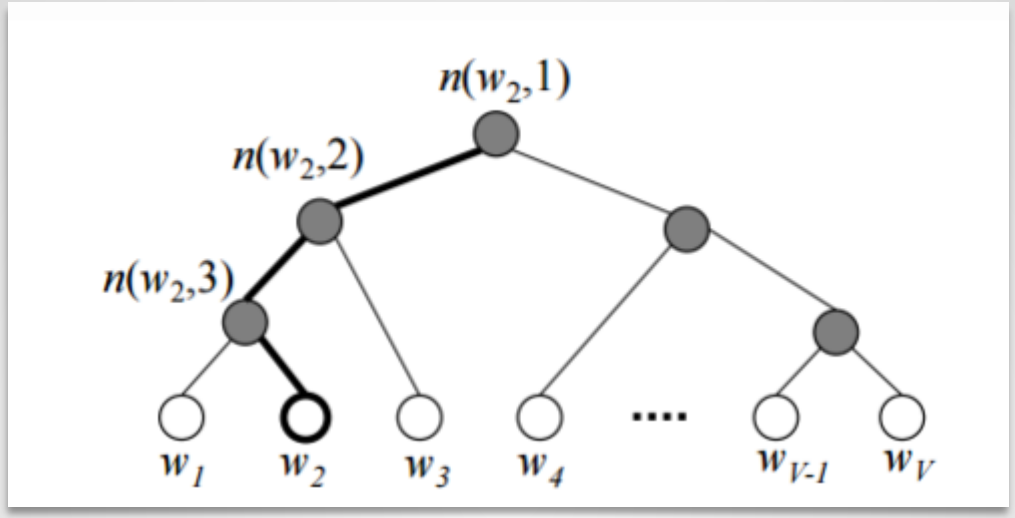

$$p(w_o|w_I) = \frac{exp(v'_{w_o} \cdot v_{w_o})}{\sum_{i=1}^{V} exp(v'_{w_i} \cdot v_{w_o})}$$

$$\zeta_\theta = -\log \frac{\exp(v'_{w_o} \cdot v_{w_o})}{\sum_{i=1}^{V} \exp(v'_{w_i} \cdot v_{w_o})}$$

$$= -v'_{w_o} \cdot v_{w_o} + \log \sum_{i=1}^{V} \exp(v'_{w_i} \cdot v_{w_o})$$

Calculating $\sum_{i=1}^{V} \exp(v'_{w_i} \cdot v_{w_o})$ is too expensive!

# 3. Word Embedding – **Hierarchical Softmax**



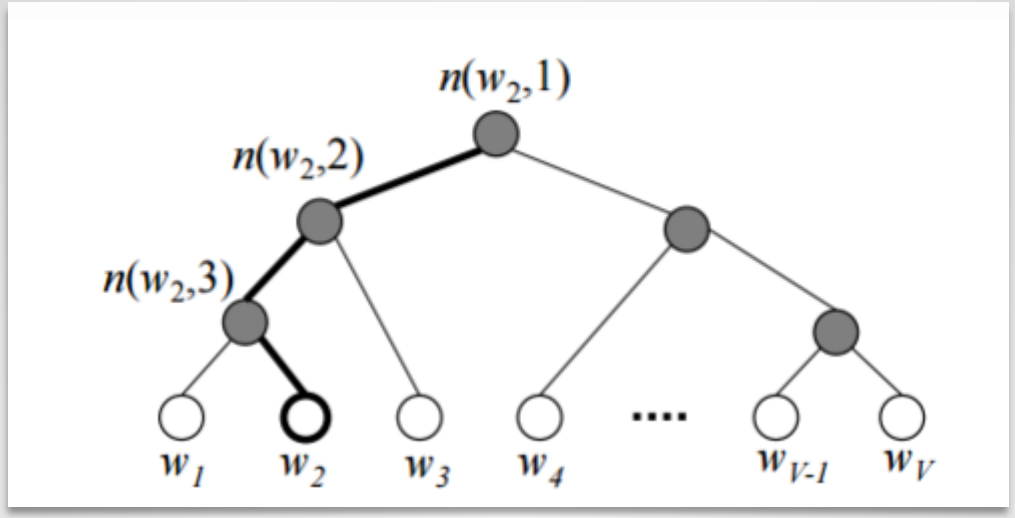

$$p(turn\ right\ \rightarrow\ ...w_I|n) = \sigma(v'_n \cdot v_{w_I})$$

$$p(turn\ left\ \rightarrow\ ...w_I|n) = 1 - p(turn\ right\ \rightarrow\ ...w_I|n) = \sigma(-v'_n \cdot v_{w_I})$$

$$p(w_o|w_I) = \prod_{k=1}^{L(w_o)} \sigma(\mathbb{I}_{turn}(n(w_o,k), n(w_o,k+1)) \cdot v'_{n(w_o,k)} \cdot v_{w_I})$$

Where $L(w_o)$ is the depth of the path leading to the word $w_o$ and $\mathbb{I}_{turn}$ is a specially indicator function which returns 1 if $n(w_o, k+1)$ is the left child of $n(w_o, k)$ otherwise -1

# 3. Word Embedding – **Hierarchical Softmax**




$$p(turn\ right\ \rightarrow\ \ldots w_I|n) = \sigma(v'_n \cdot v_{w_I})$$

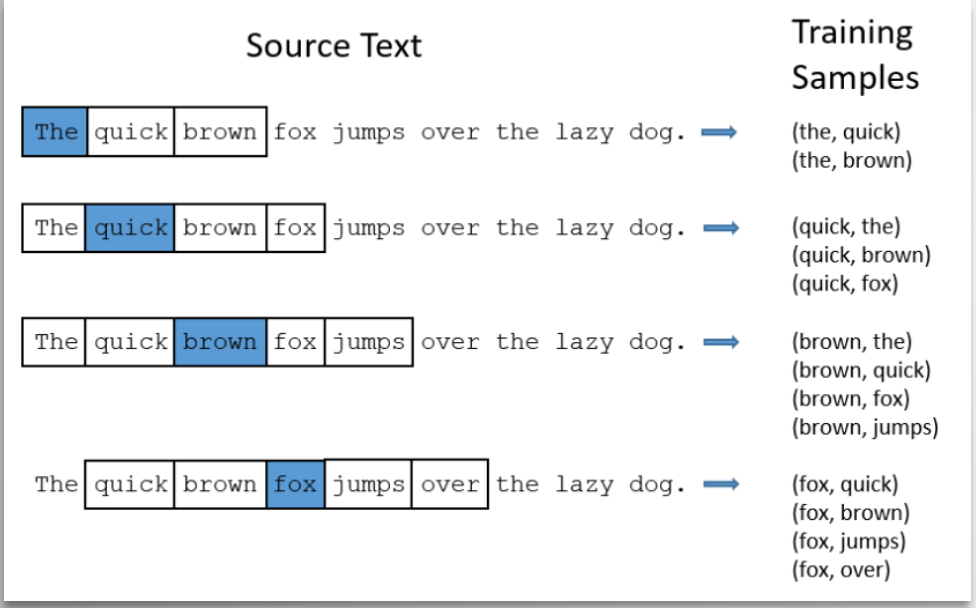
$$p(turn\ left\ \rightarrow\ \ldots w_I|n) = 1 - p(turn\ right\ \rightarrow\ \ldots w_I|n) = \sigma(-v'_n \cdot v_{w_I})$$

$$p(w_o|w_I) = \prod_{k=1}^{L(w_o)} \sigma(\mathbb{I}_{turn}(n(w_o,k), n(w_o,k+1)) \cdot v'_{n(w_o,k)} \cdot v_{w_I})$$

$O(V) \rightarrow O(\ln(V))$ $\sum_{i=1}^{V} \exp(v'_{w_i} \cdot v_{w_o})$계산을 하지 않고 확률을 계산

# 3. Word Embedding – **Negative Sampling**

- **Negative Sampling**




Negative sampling 은 다중 분류 문제를 logistic regression 을 이용한 실제 pair 데이터를 가짜 pair로부터 분류하는 이진 분류 문제로 변환하여 근사하는 방법

$$p(d = 1|w, w_I) = \sigma(v'_w \cdot v_{w_I})$$

$$p(d = 0|w, w_I) = 1 - \sigma(v'_w \cdot v_{w_I}) = \sigma(-v'_n \cdot v_{w_I})$$

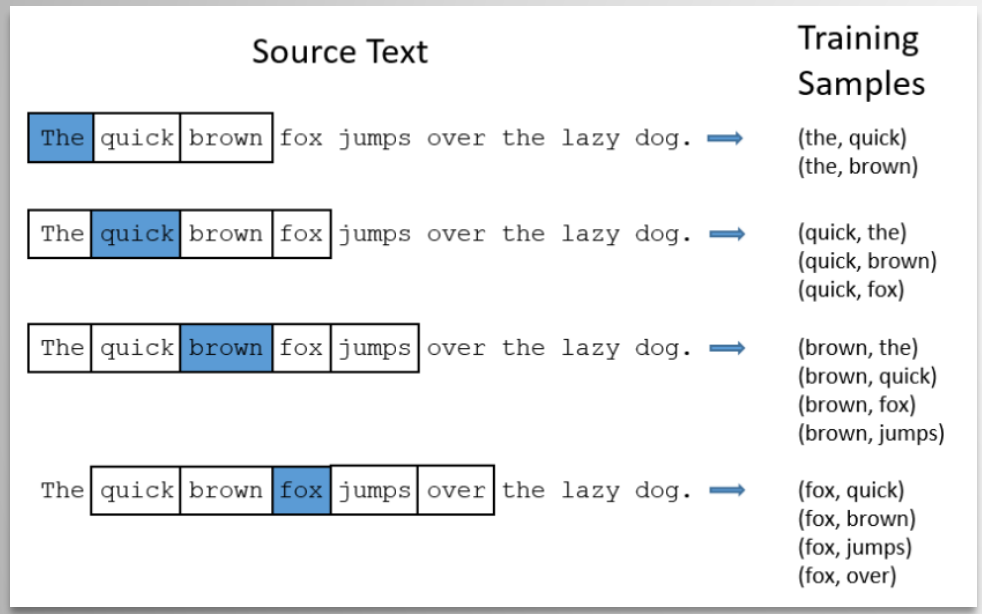
$$\varsigma_\theta = -[\log \sigma(v'_w \cdot v_{w_I}) + \sum_{i=1}^{k} \mathbb{E}_{w_i \sim P_n(w)} [\log \sigma(-v'_{w_i} \cdot v_{w_I})]]$$

실제 target word (positive sample) 와 k개의 negative sample (가짜 pair) 들을 뽑아 최적화를 진행한다.

$$P(w_i) = \frac{f(w_j)}{\sum_j (f(w_j))^{3/4}}, \text{ where } f(w_j) \text{ is the frequency of word.}$$

# 3. Word Embedding – **Subsampling Frequent Words**

- **Subsampling Frequent Words**

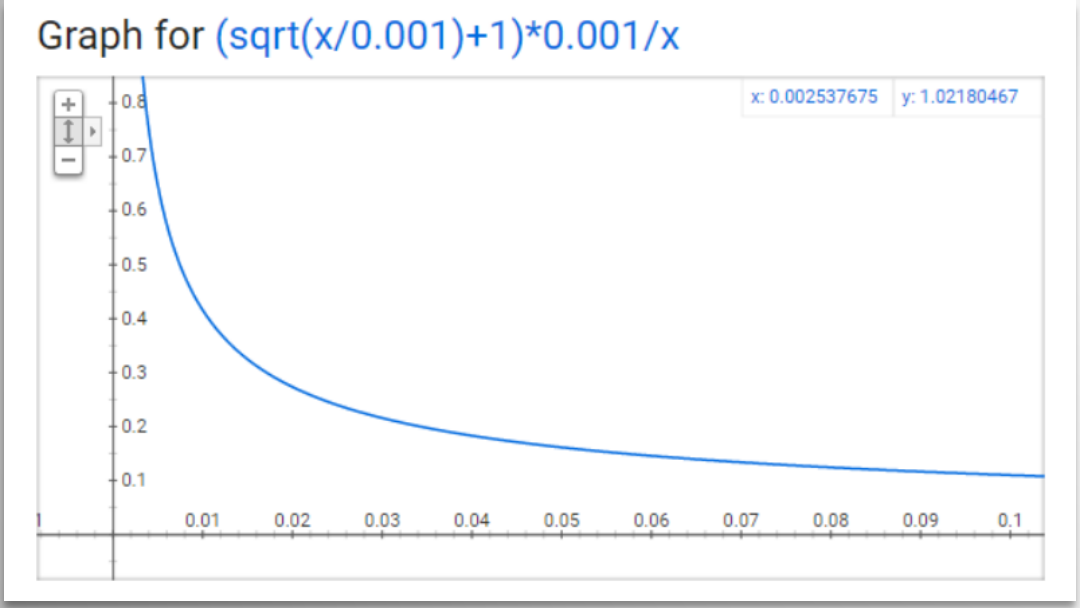



- (quick, the), (brown, the)와 같은 sample은 quick/brown의 의미에 대한 정보를 담고 있지 않다.
- (the, …), (a, …)와 같이 굉장히 자주 나오는 단어들은 pair가 무수히 많이 존재하므로 적은 비율로 sampling 되더라도 충분히 많다.
- Subsampling을 통해 자주 나오는 단어는 덜 뽑고, 적게 나오는 단어는 더 많이 뽑아 보다 적은 훈련으로 더 좋은 distributed representation을 학습한다.
- $$P(w_i) = \left(\sqrt{\frac{z(w_i)}{0.001}} + 1\right) \times \frac{0.001}{z(w_i)}$$

*$z(w_i)$ 는 전체 단어 중 차지하는 비율

# 3. Word Embedding – **Subsampling Frequent Words**

- **Subsampling Frequent Words**

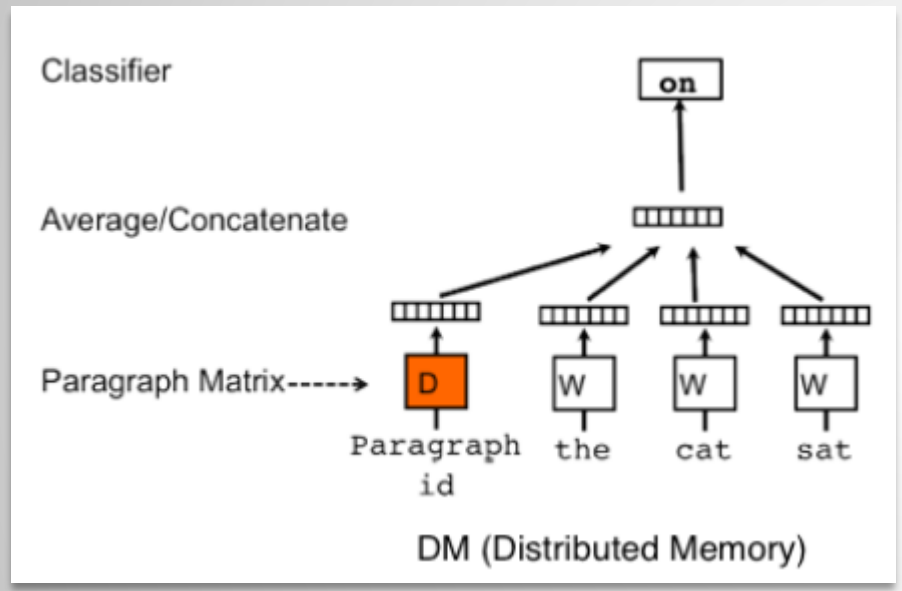


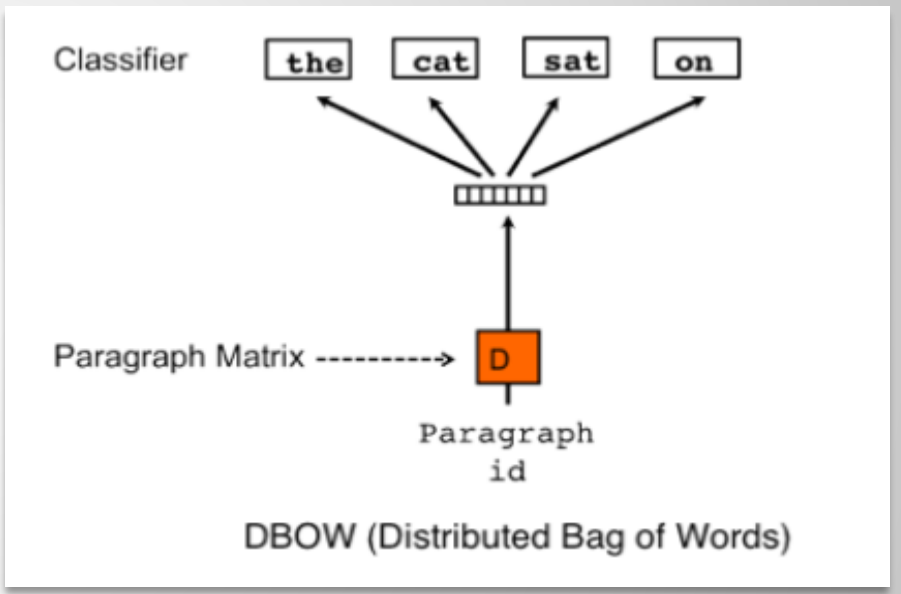

- (quick, the), (brown, the)와 같은 sample은 quick/brown의 의미에 대한 정보를 담고 있지 않다.
- (the, …), (a, …)와 같이 굉장히 자주 나오는 단어들은 pair가 무수히 많이 존재하므로 적은 비율로 sampling 되더라도 충분히 많다.
- Subsampling을 통해 자주 나오는 단어는 덜 뽑고, 적게 나오는 단어는 더 많이 뽑아 보다 적은 훈련으로 더 좋은 distributed representation을 학습한다.
- $$P(w_i) = \left(\sqrt{\frac{z(w_i)}{0.001}} + 1\right) \times \frac{0.001}{z(w_i)}$$

*$z(w_i)$ 는 전체 단어 중 차지하는 비율

# 3. Word Embedding – **Doc2Vec**

## Distributed Representation of Sentences and Documents



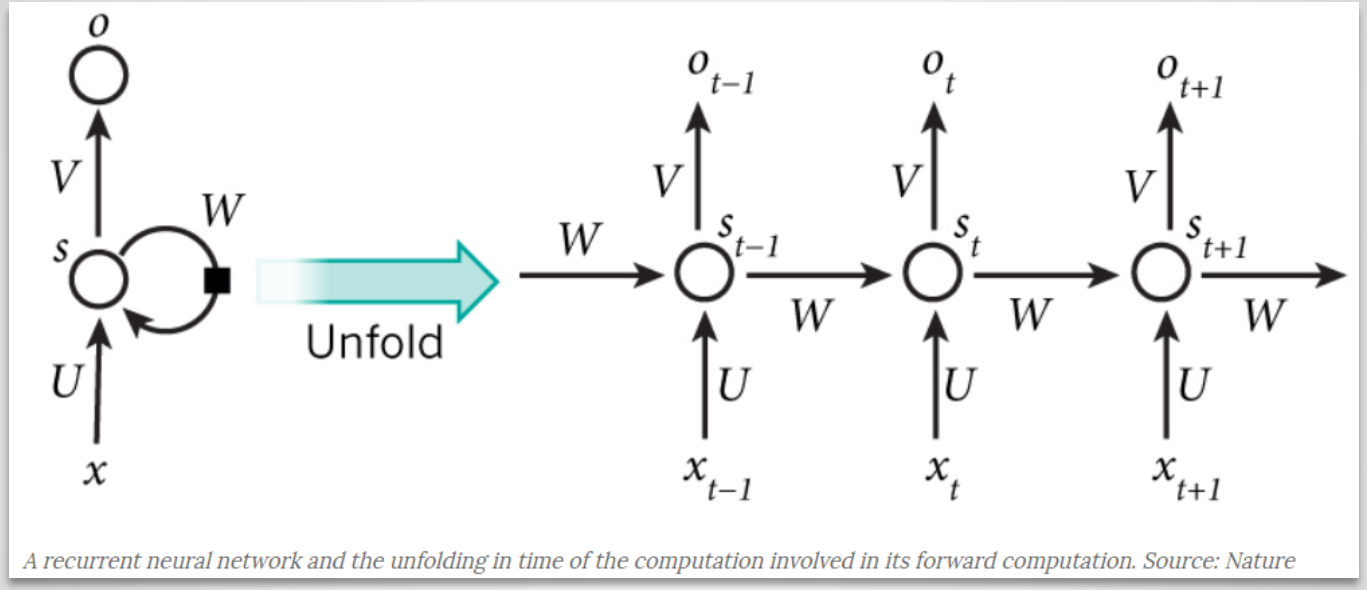




document embedding vectors도 CBOW 또는 Skip-Gram 모델을 이용하여
word embedding vectors를 학습했던 것 처럼 document vector를 추가하여 학습

# Word Embedding 실습

# 4. Recurrent Neural Network

# RNN (Recurrent Neural Network)




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

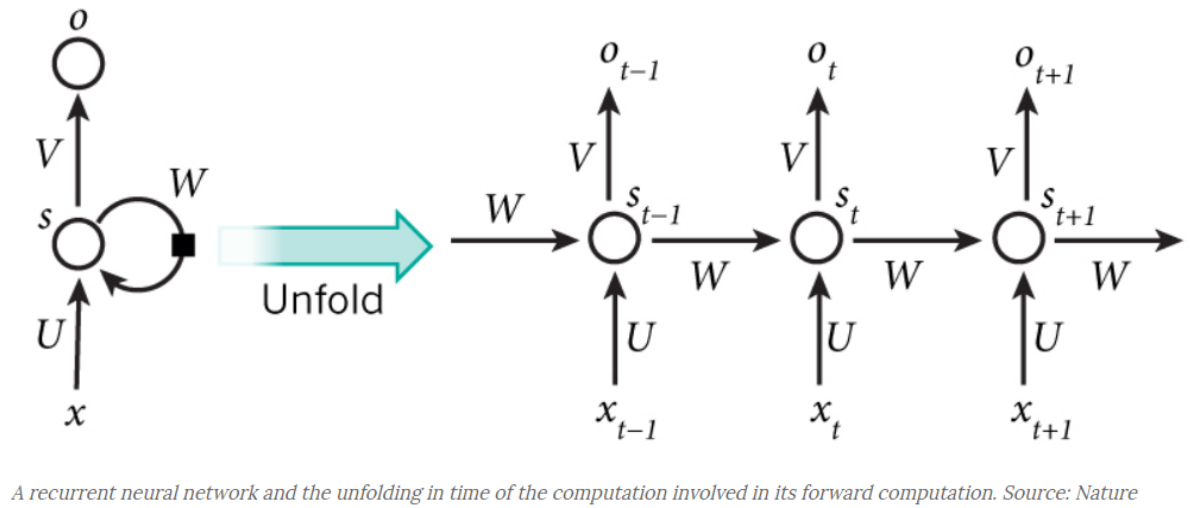
$$s_t = \tanh(U_s \cdot x_t + W_s \cdot s_{t-1} + b_s) \quad x_t \in \mathbf{R}^n, s_t \in \mathbf{R}^m, U_s \in \mathbf{R}^{n \times m}, W_s \in \mathbf{R}^{m \times m}$$

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

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

# 4. RNN – **Recurrent Neural Network**

- **RNN (Recurrent Neural Network)**




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

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

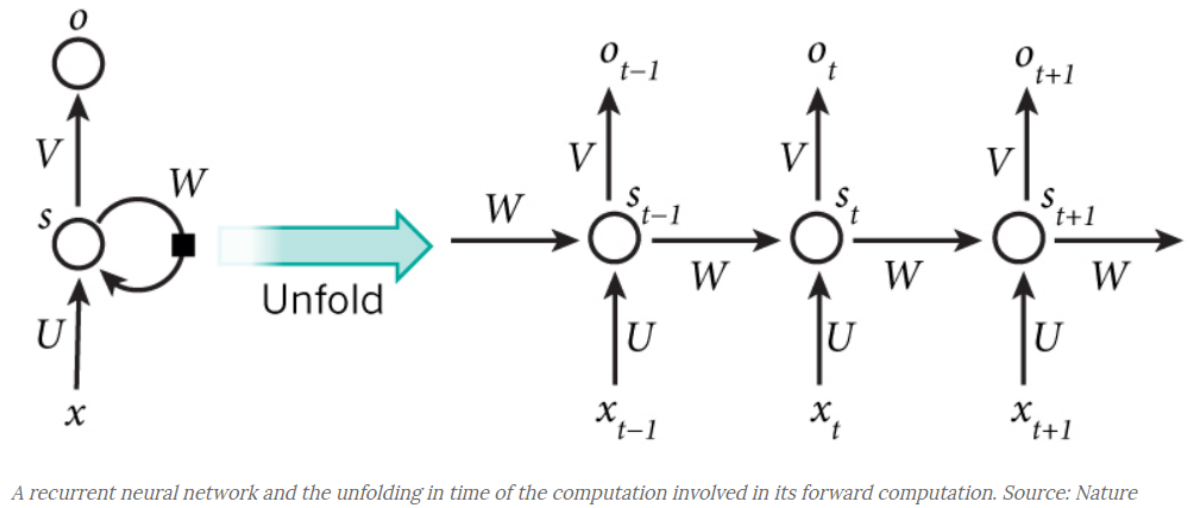
$$\hat{y}_t = o_t = \text{softmax}(V \cdot s_t + b_o)$$

$$J^{(t)}(\theta) = -\sum_{j=1}^{j=|V|} y_{t,j} \times \log(\hat{y}_{t,j})$$

$$J = -\frac{1}{T}\sum_{t=1}^{T} J^{(t)}(\theta) = - -\frac{1}{T}\sum_{t=1}^{T}\sum_{j=1}^{j=|V|} y_{t,j} \times \log(\hat{y}_{t,j})$$

# 4. RNN – **Recurrent Neural Network**

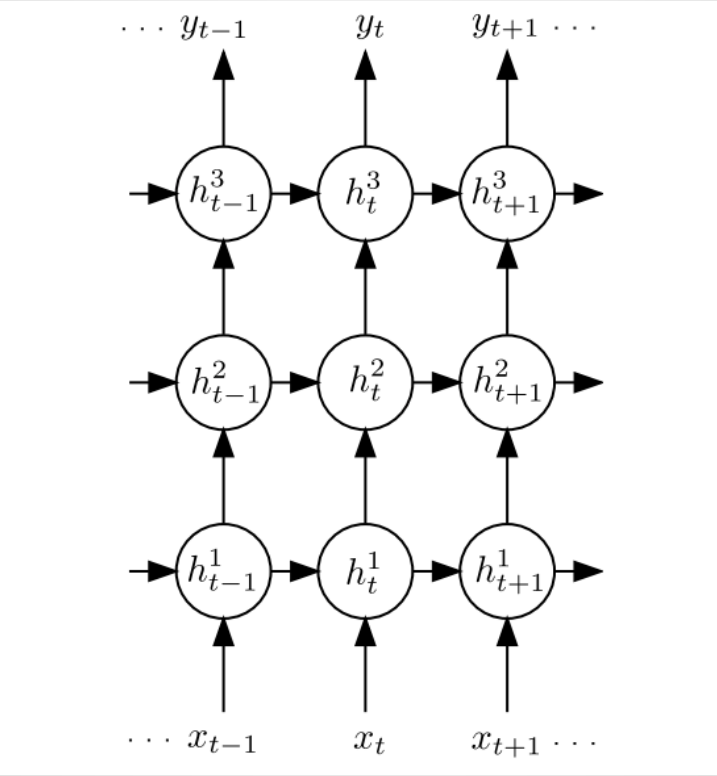
- **RNN (Recurrent Neural Network)**




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

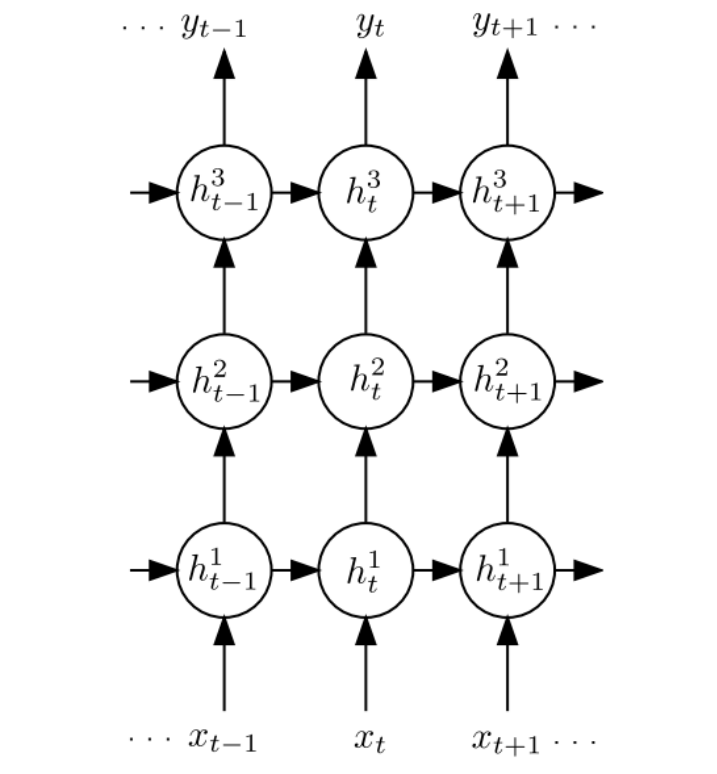
- Hidden state $s_t$ 는 previous step에서 발생한 정보 또한 다룰 수 있으며 output state인 $o_t$ 는 오직 $s_t$를 이용하여 계산이 이루어진다.
- 기존 전통 딥러닝과는 다르게, RNN은 각 time step이 같은 parameter를 공유한다.
- 위 그림에서는 output state를 각 time step마다 계산하여 도출하지만 task에 따라서는 모든 output state가 꼭 필요로 되지 않는다. 예를 들어 감성분석의 경우, 우리는 각 단어 이후의 감성이 아닌 모든 단어를 본 뒤의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

# Deep RNN


$\cdots y_{t-1}$
$y_t$
$y_{t+1} \cdots$
$h^3_{t-1}$
$h^3_t$
$h^3_{t+1}$
$h^2_{t-1}$
$h^2_t$
$h^2_{t+1}$
$h^1_{t-1}$
$h^1_t$
$h^1_{t+1}$
$\cdots x_{t-1}$
$x_t$
$x_{t+1} \cdots$

# 4. RNN – **Deep RNN**

- **Deep RNN**



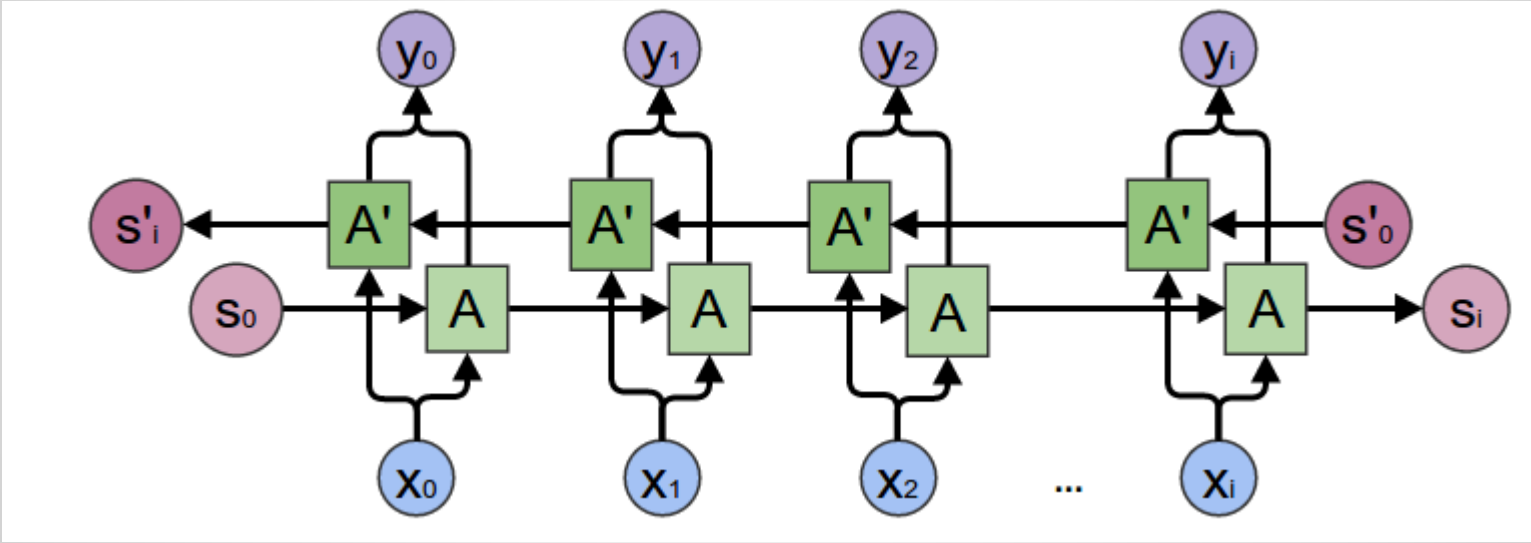

RNN에서도 hidden layer 층을 더 깊게 쌓음으로써 complexity가 더 높은 데이터에 대해서도 확률분포를 근사할 수 있다.

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

$$h_t^{(i)} = \tanh\left(U^{(i)} \cdot h_t^{(i-1)} + W^{(i)} \cdot h_{t-1}^{(i)} + b^{(i)}\right), i \geq 2$$

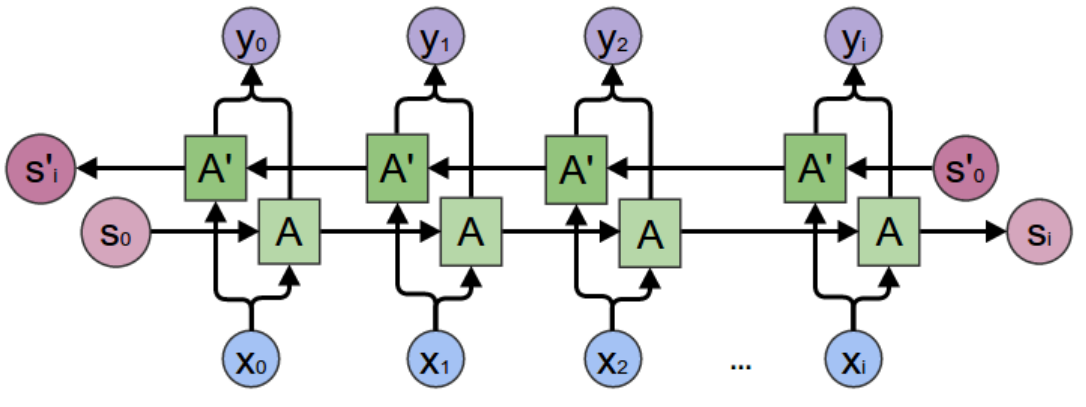
$$\hat{y}_t = \text{softmax}(V \cdot h_t^{(n)} + b_o)$$

# Bidirectional RNN



# 4. RNN – **Bidirectional RNN**

- **Bidirectional RNN**



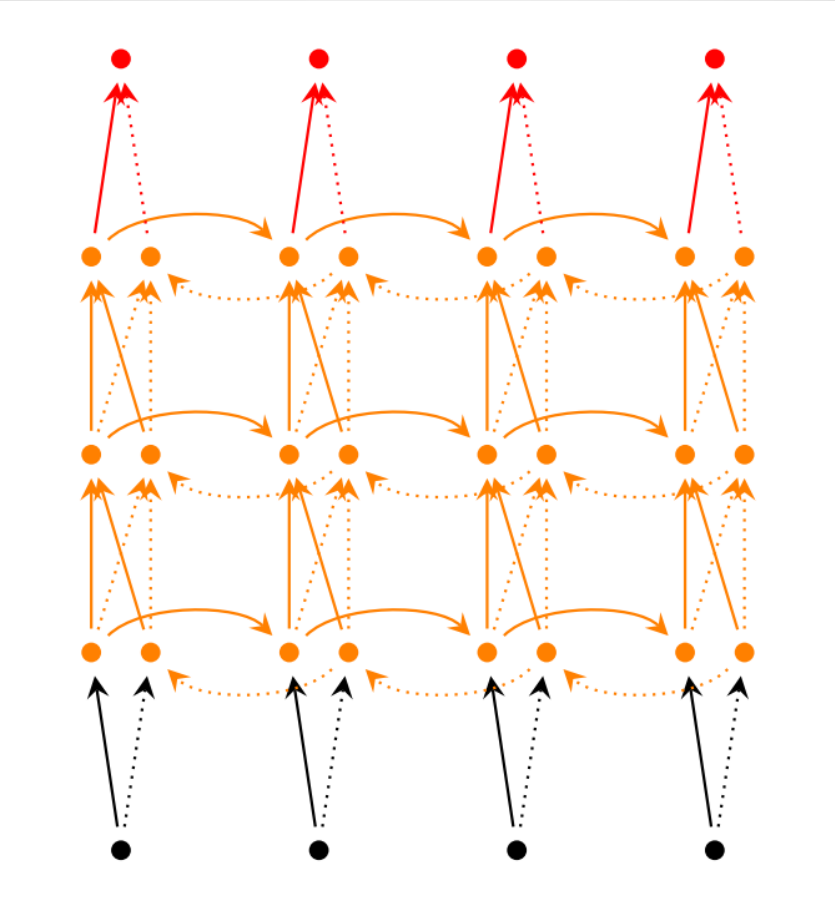

Unidirectional RNN의 경우 time step이 길어질수록 앞쪽 time step에 대한 정보가 희석되기 때문에 양 방향으로 RNN을 진행함으로써 RNN을 향상시킨다.

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

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

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

# Deep bidirectional RNN



# RNN 실습1

# 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.

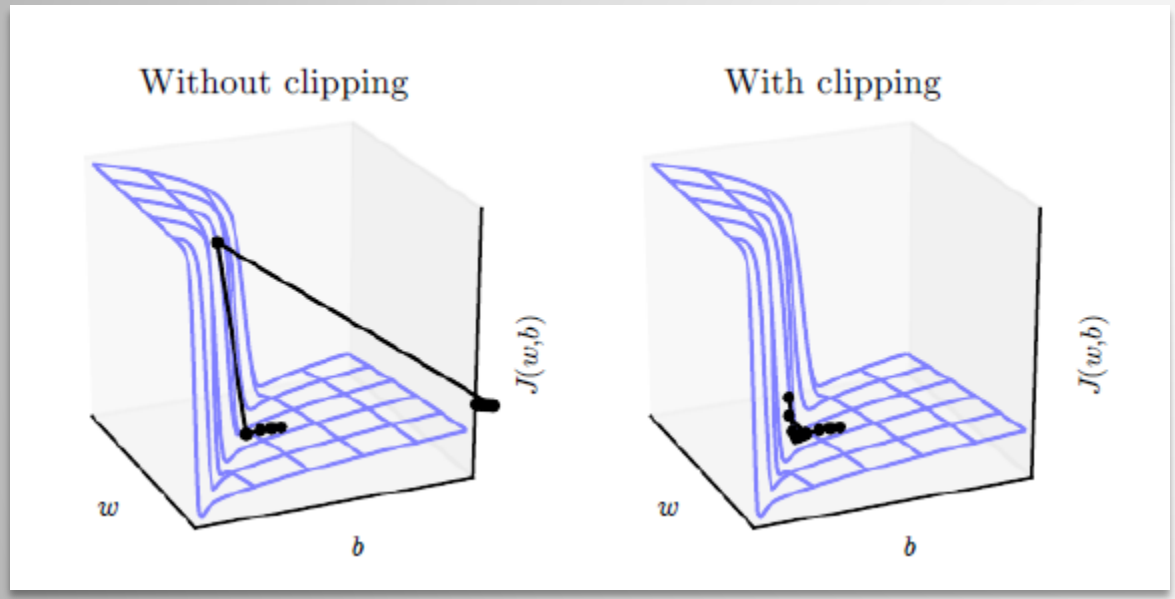
**→ Gradient 소실(vanishing) 또는 폭발(explosion) 문제 발생 때문**

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

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

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

- **Gradient Clipping**

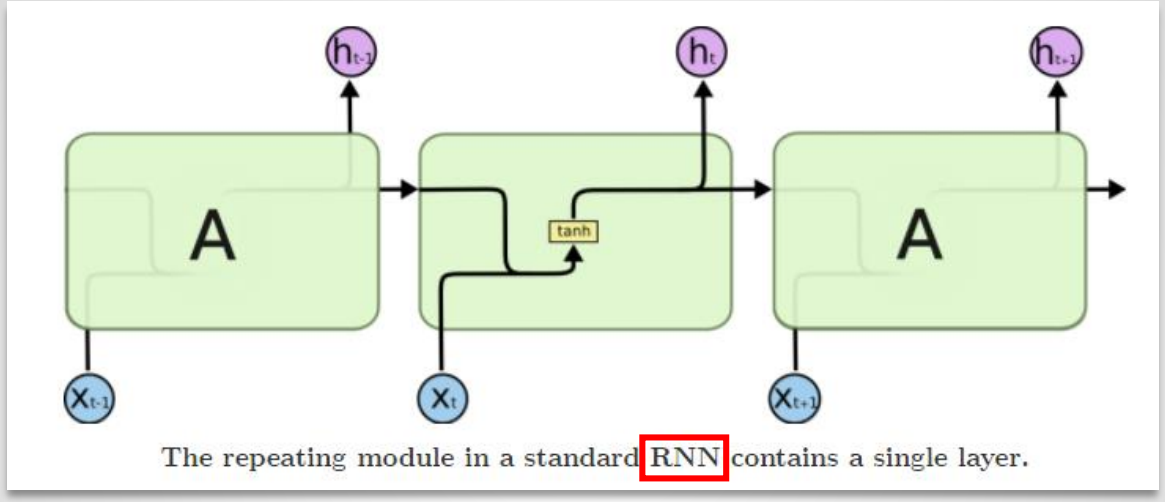



Gradient explosion 문제를 해결하는 방법 중 하나로 일정 크기 이상의 gradient 값을 가질 경우 값을 축소시킨다.

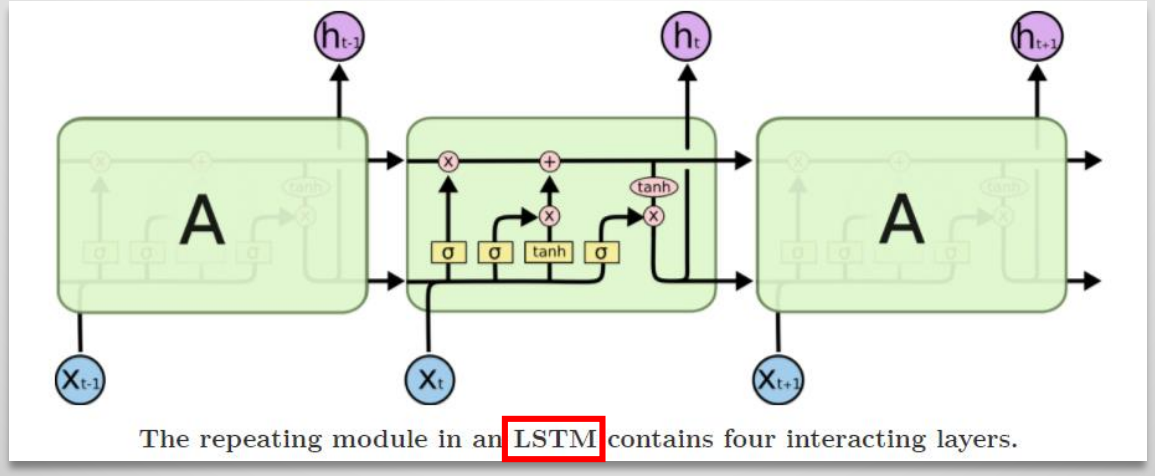
**Algorithm.** Pseudo-code for norm clipping

$\hat{\boldsymbol{g}} \leftarrow \frac{\partial \mathcal{L}}{\partial \theta}$
**if** $\|\hat{\boldsymbol{g}}\| > threshold$ **then**
$\hat{\boldsymbol{g}} \leftarrow \frac{threshold}{\|\hat{\boldsymbol{g}}\|}\hat{\boldsymbol{g}}$
**endif**

# LSTM (Long Short-Term Memory)

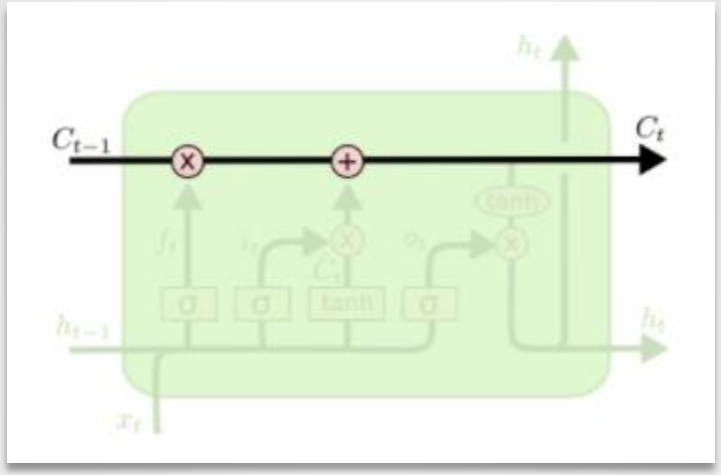



The repeating module in a standard RNN contains a single layer.




The repeating module in an LSTM contains four interacting layers.

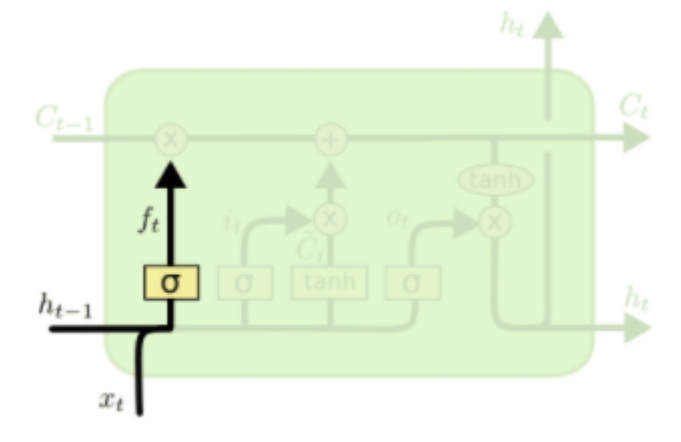
# LSTM (Long Short-Term Memory)




- 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.

# LSTM (Long Short-Term Memory)

## 1. forget gate

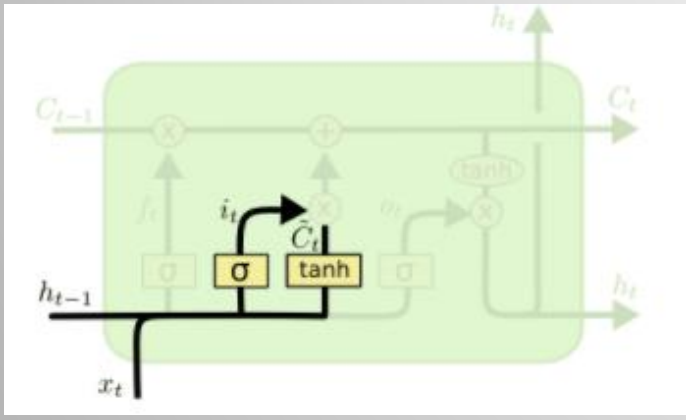



- previous hidden state 에서 어떤 정보를 잊을 것인지 결정

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

- (0, 1) 사이의 값으로 이루어진 vector

## 2. input gate




- new candidate hidden state에서 어떤 정보를 취할 것인지 결정

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

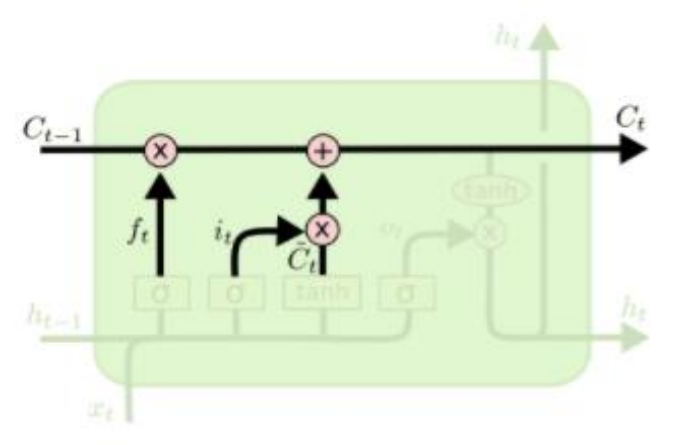
- (0, 1) 사이의 값으로 이루어진 vector

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

- new candidate state

# LSTM (Long Short-Term Memory)

### 3. New hidden state

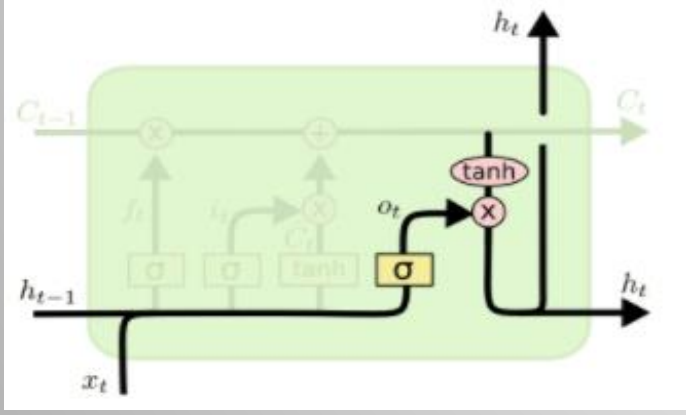



- forget gate 및 input gate를 통한 new hidden state 계산

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

- $*$ 는 point-wise multiplication

### 4. output gate




- output gate 를 통한 output state (hidden state 2) 결정

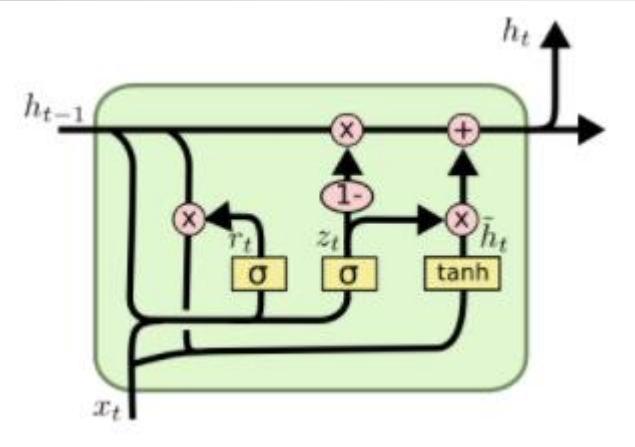
$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

- (0, 1) 사이의 값으로 이루어진 vector

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

- output state at time step t

# GRU (Gated Recurrent Unit)



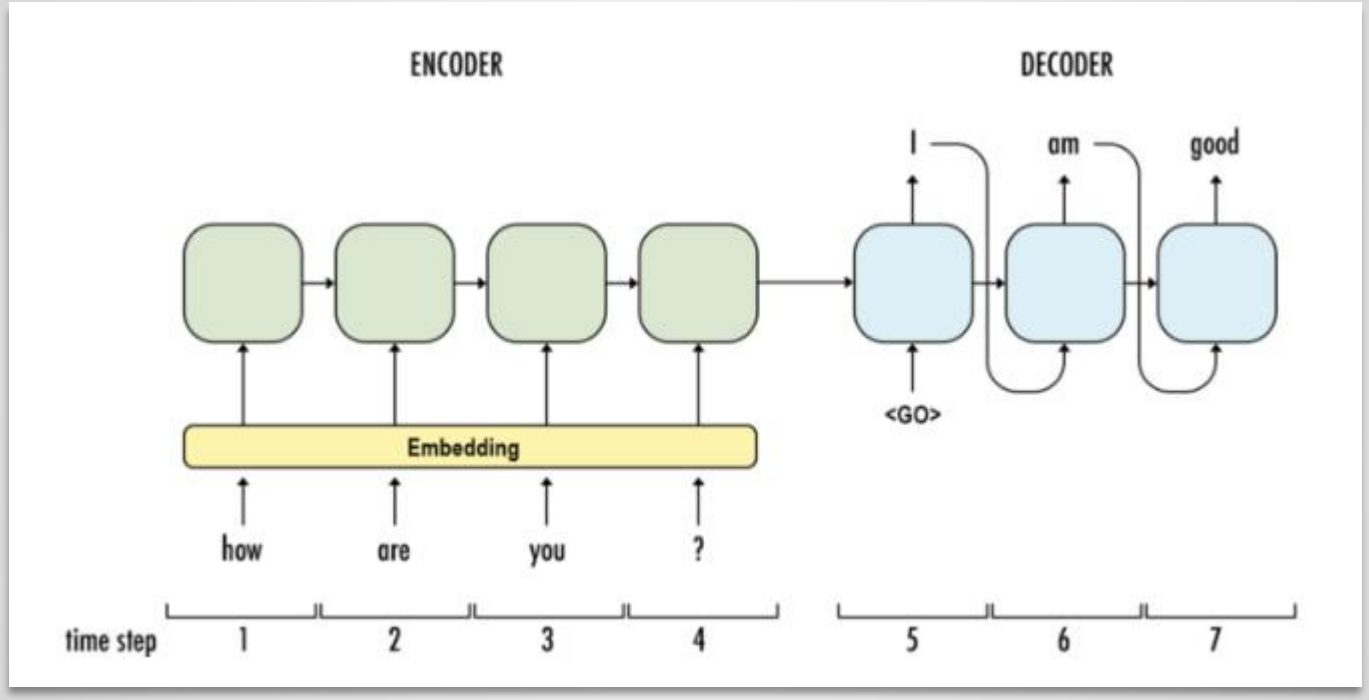

$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t])$$
$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t])$$
$$\tilde{h}_t = tanh(W_h \cdot [r_t * h_{t-1}, x_t])$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

- $z_t$: 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.

# RNN 실습2

# 5. Sequence to sequence

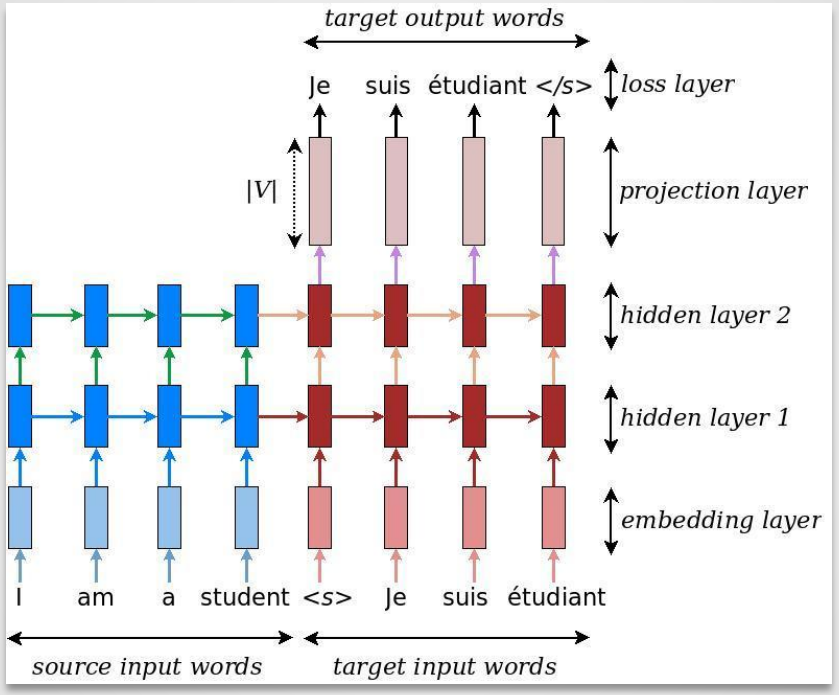
# Sequence to sequence model




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

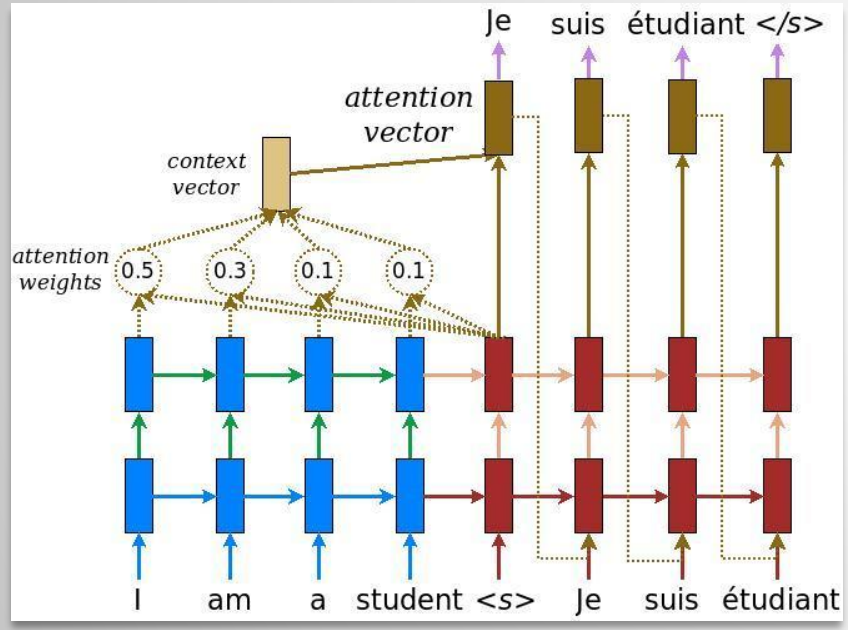
# seq2seq 실습1

# 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.

# 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.

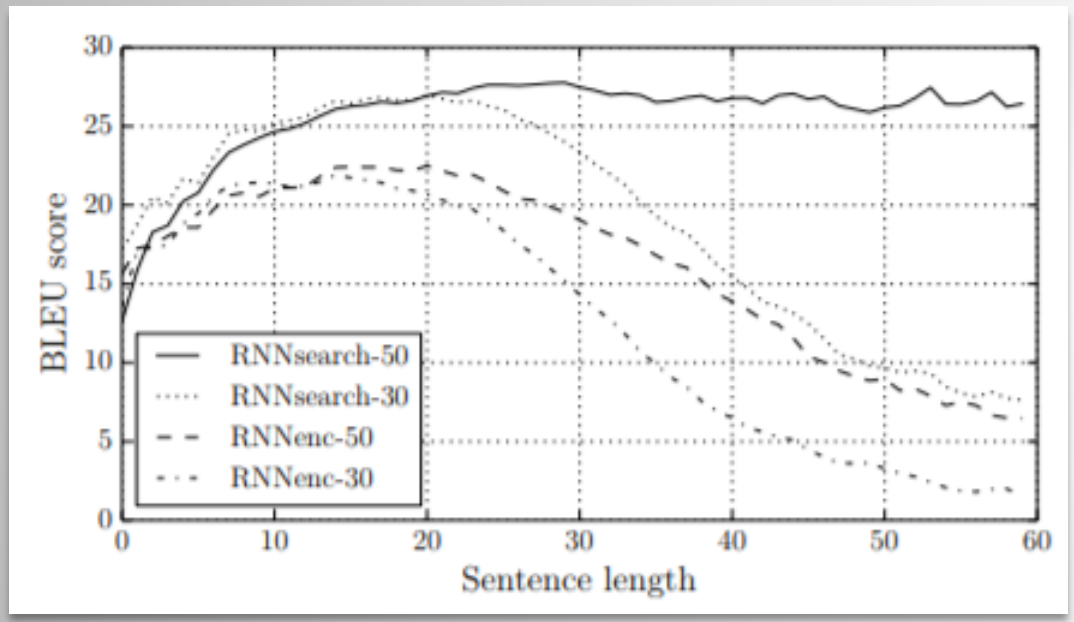
$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j, \qquad \alpha_{ij} = \frac{exp(e_{ij})}{\sum_{k=1}^{T_x} exp(e_{ik})}$$

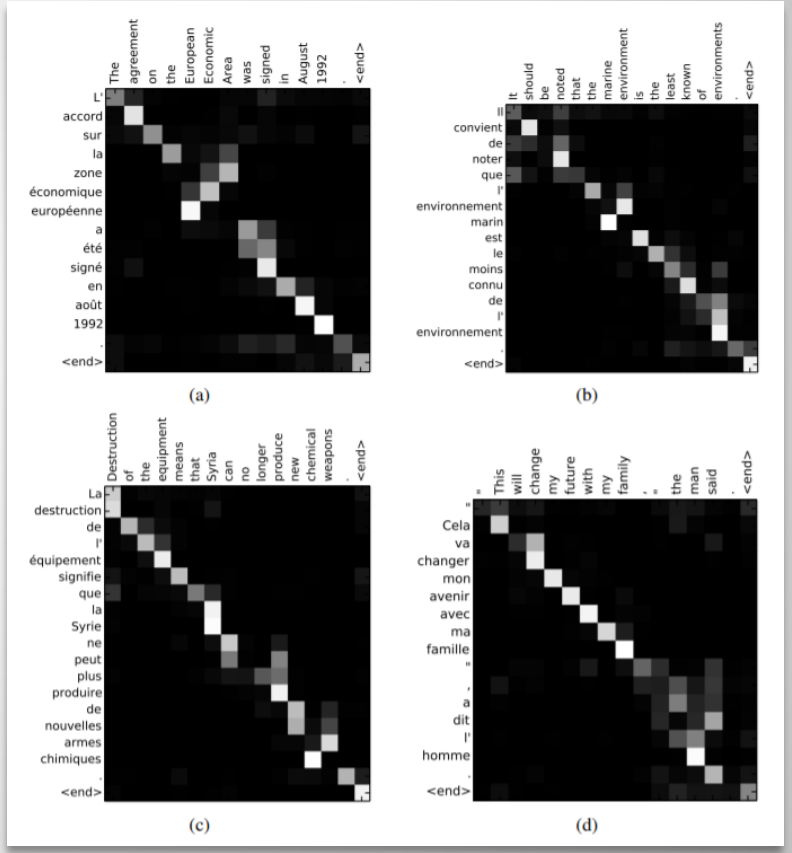
$$e_{ij} = a(s_{i-1},h_j)$$

$$\hat{y}_t = \text{softmax}(V \cdot [h_t,c_i] + b_o)$$

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

# Attention Mechanism



# seq2seq 실습2

# Any Question?

# 감사합니다

Email: minho@metafact.org

Github: https://github.com/Bricoler