

Representation of Word/Document

Hyunjoong Kim

soy.lovit@gmail.com

github.com/lovit

1. Word2Vec
2. Doc2Vec
3. GloVe
4. FastText (subwords embedding)
5. FastText (supervised embedding)

Embedding

- 임베딩은 데이터를 공간 x 에서 새로운 공간 y 로 보내는 것입니다.
 - 원하는 정보를 잘 저장하며 공간을 변환하는 것으로,
 - 어떤 정보를 보존할 것이냐에 따라서 다양한 임베딩 방법이 존재합니다.
- 각 임베딩 방법을 “어떤 정보를 보존”하려 하는지의 관점에서 살펴본다면, 각 알고리즘이 무엇을/어떻게 학습하는지 이해하기 쉽습니다.

Distributed representation

- 단어/문서를 **d차원 공간의 벡터**로 표현합니다.
 - Word2Vec 이 대표적이며, 각 차원이 특별한 의미를 지니지는 않습니다.
- 벡터 공간은 단어의 "**의미적 유사성**"을 반영합니다.
 - 벡터가 비슷한 단어/문서는 의미가 비슷합니다.
 - 비슷함의 정의는 알고리즘마다 다릅니다.

'dog' = [0.31, -0.21, 2.01, 0.58, ...]

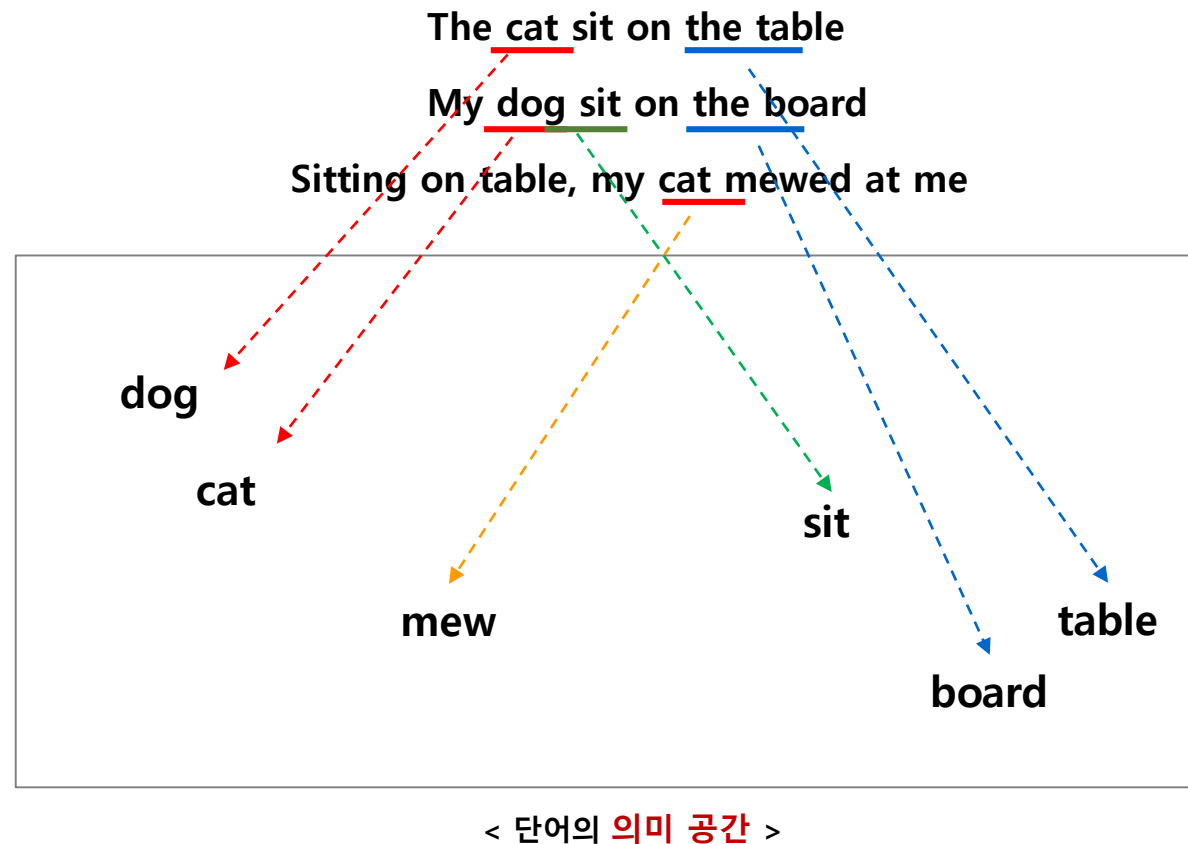
'cat' = [0.45, -0.17, 1.79, 0.61, ...]

'topic modeling' = [-2.01, 0.03, 0.22, 0.54, ...]

'dim. reduction' = [-1.88, 0.11, 0.19, 0.45, ...]

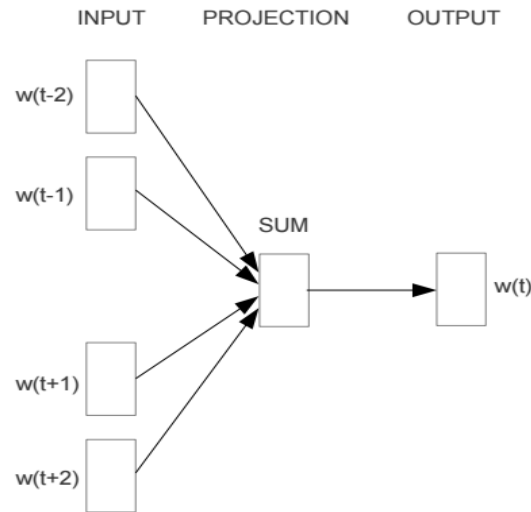
Distributed representation

- 각 벡터는 "의미 공간"에서의 좌표값 역할을 합니다.

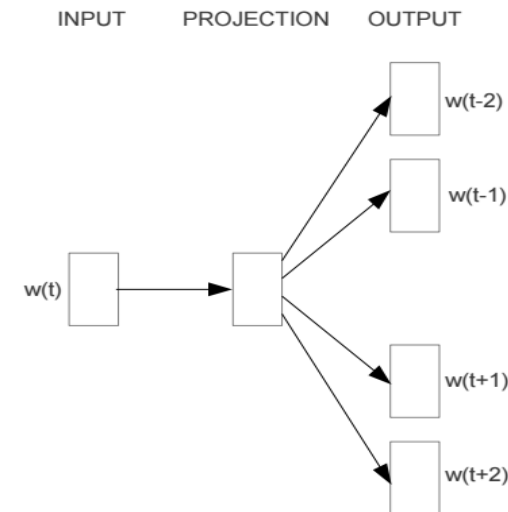


Word2Vec

- Word2Vec은 CBOW, Skip-gram 두가지 형태로 제안되었습니다.
 - CBOW는 주위 단어로 현재 단어 $w(t)$ 를 예측하는 모델
 - Skipgram은 현재 단어 $w(t)$ 로 주위 단어 모두를 예측하는 모델



CBOW



Skip-gram

Word2Vec

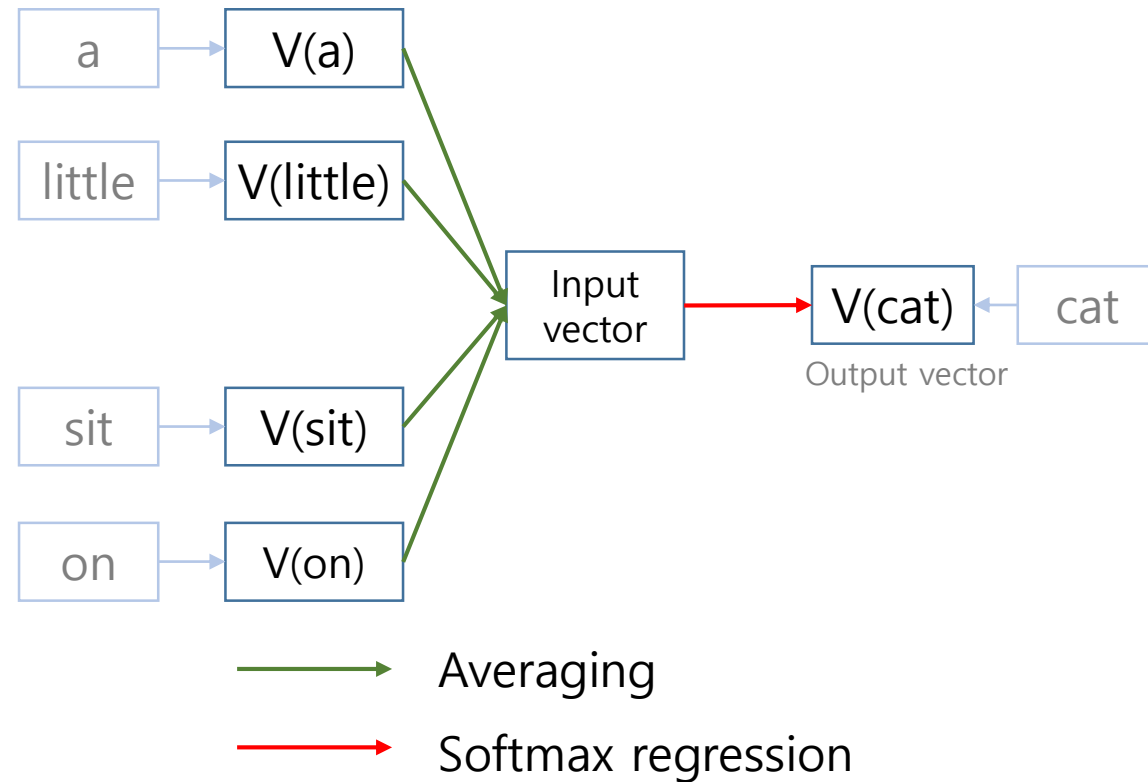
- Word2Vec은 $|V|$ 개의 classes 를 예측하는 Softmax regression 입니다.

Lookup table

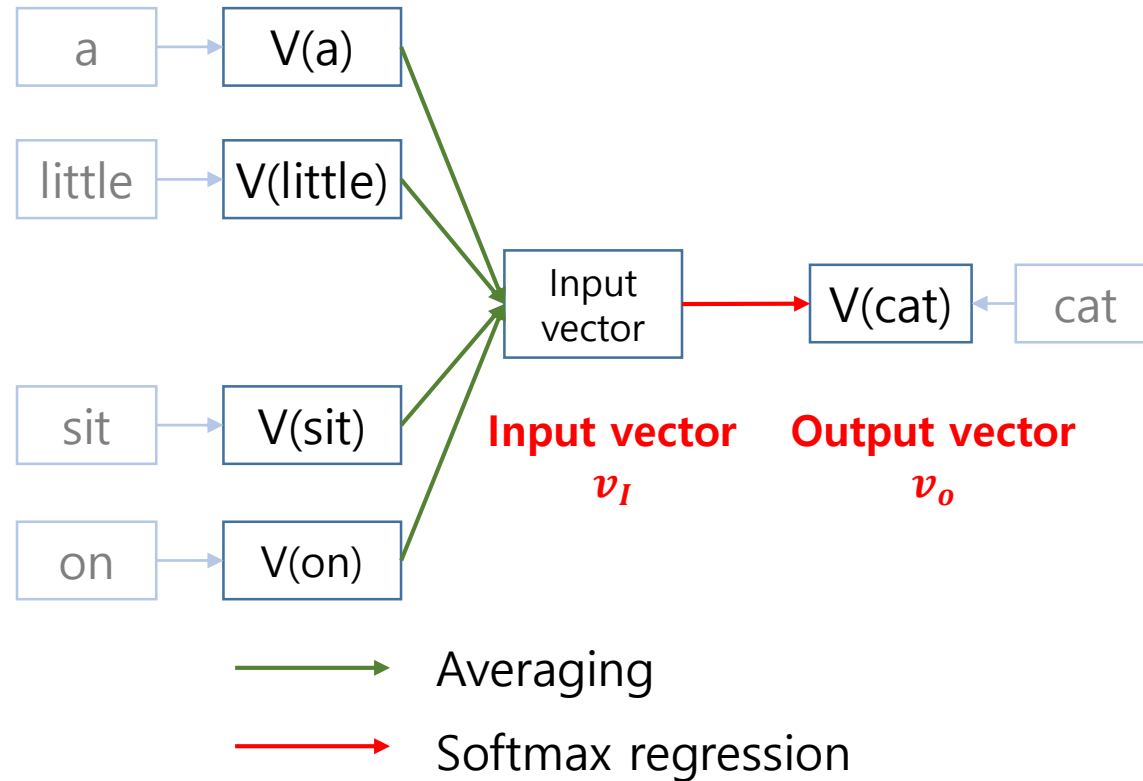
```
{  
  cat: [.31, -.22, .54]  
  dog: [.22, -0.3, -0.52]  
  on: ...  
  the: ...  
  little: ...  
}
```

Lookup word vector

$V(\text{'cat'}) = [.31, -.22, .54]$



Word2Vec



$$y_j = \frac{\exp(v_I^T v_{o_j})}{\sum_j \exp(v_I^T v_{o_j})}$$

Logistic Regression

- Logistic Regression (LR)은 대표적인 binary classification 알고리즘
 - positive class에 속할 점수를 $[0, 1]$ 사이로 표현하기 때문에 확률 모형처럼 이용

$$y_{\theta}(x) = \frac{1}{1 + \exp(-\theta^T x)}$$

- 학습 데이터 $\{(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})\}$ 가 주어졌을 때, loss function은

$$J(\theta) = - \left[\sum_{i=1}^m y^{(i)} \log h_{\theta}(x^{(i)}) + (1 - y^{(i)}) \log (1 - h_{\theta}(x^{(i)})) \right]$$

Logistic Regression

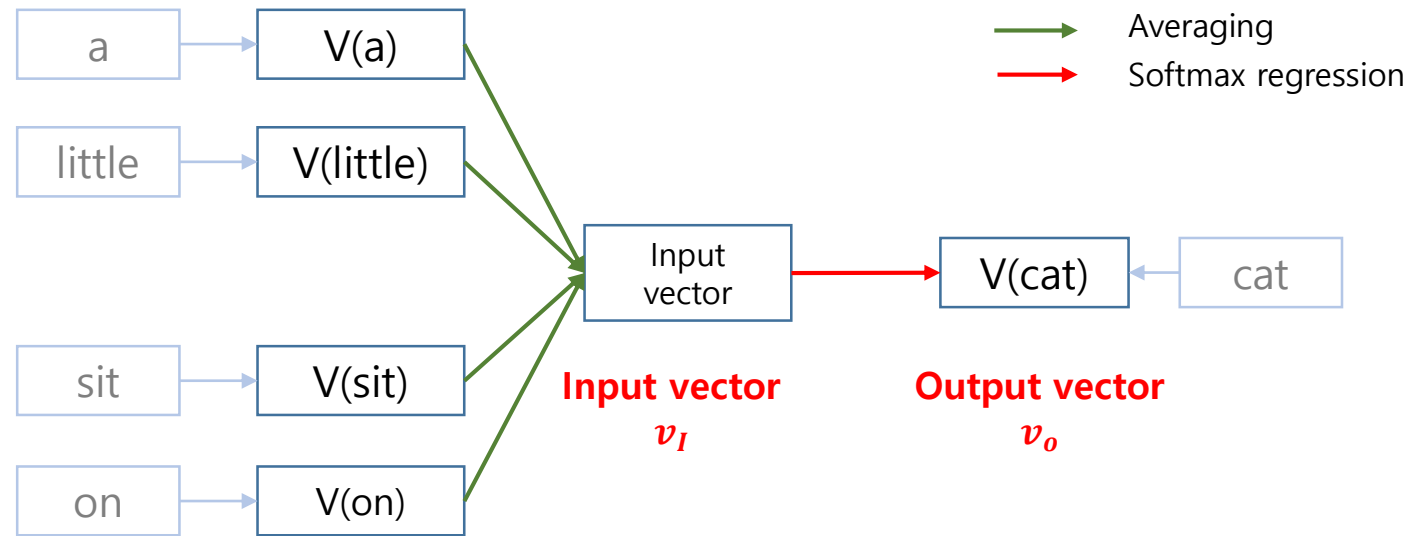
- Softmax regression은 Logistic regression의 multi class classification 버전

$$h_{\theta}(x) = \begin{bmatrix} P(y = 1|x; \theta) \\ \vdots \\ P(y = K|x; \theta) \end{bmatrix} = \frac{1}{\sum_{j=1}^K \exp(\theta^{(j)T} x)} \exp \left(\begin{bmatrix} \theta^{(1)T} x \\ \vdots \\ \theta^{(K)T} x \end{bmatrix} \right)$$

- 학습 데이터 $\{(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})\}$ 가 주어졌을 때, loss function은

$$J(\theta) = - \left[\sum_{i=1}^m \sum_{k=1}^K 1\{y^{(i)} = k\} \log \frac{\exp(\theta^{(j)T} x^{(i)})}{\sum_{j=1}^K \exp(\theta^{(j)T} x^{(i)})} \right]$$

Word2Vec

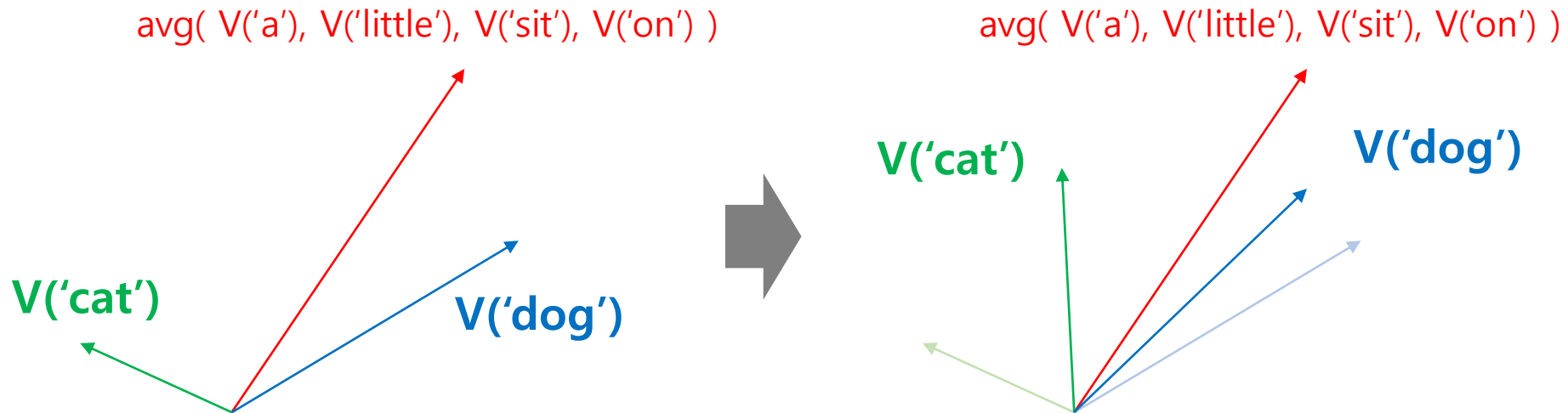


$$h_{\theta}(x) = \begin{bmatrix} P(w_{(t)} = \text{cat}) \\ P(w_{(t)} = \text{dog}) \\ P(w_{(t)} = \text{table}) \\ \dots \\ P(w_{(t)} = \text{Vocab}) \end{bmatrix} = \frac{1}{\sum_{j=1}^{|V|} \exp(\theta^{(j)T} x)} \begin{bmatrix} \exp(v(\text{cat})^T v_I) \\ \exp(v(\text{dog})^T v_I) \\ \exp(v(\text{table})^T v_I) \\ \dots \\ \exp(v(\text{Vocab})^T v_I) \end{bmatrix}$$

x : Input vector (average of contextual word vector)

Word2Vec

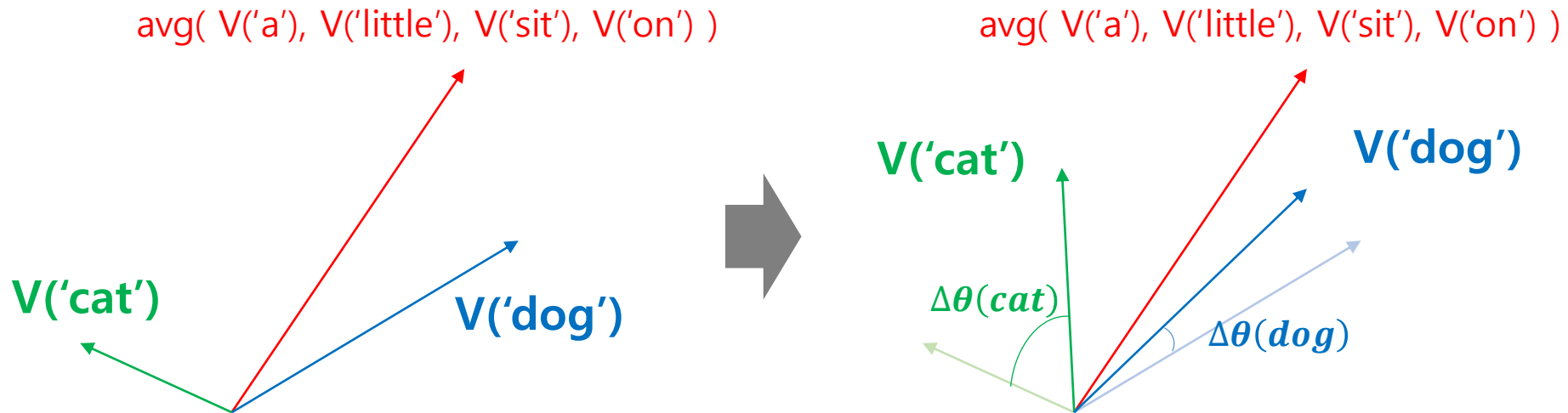
- 'cat'과 'dog' 은 비슷한 문맥 단어 분포(contextual word distribution) 를 지니기 때문에 비슷한 word embedding vector 로 학습됩니다.
 - 'cat', 'dog' 은 모두 context vector 에 가깝게 이동합니다



Word2Vec

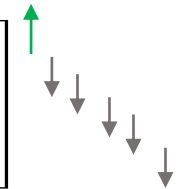
- 'cat', 'dog'의 두 단어 벡터 모두 context vector에 가깝게 이동
- Context vector 와 차이가 많이나는 단어 'cat'은 **학습량 (벡터의 변화량)**도 큼

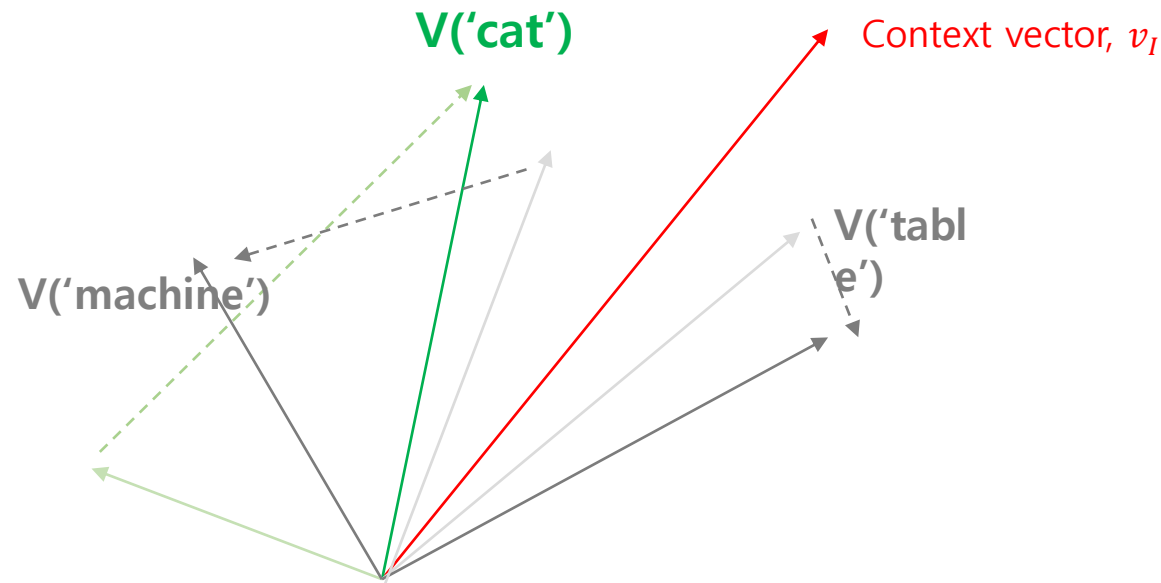
Softmax regression loss



Word2Vec

- Softmax regression 은 알맞은 단어는 context vector 방향으로 당기고, 틀린 단어는 다른 방향으로 밀어냅니다.

$$h_{\theta}(x) = \begin{bmatrix} P(w_{(t)} = cat) \\ P(w_{(t)} = dog) \\ P(w_{(t)} = table) \\ \dots \\ P(w_{(t)} = Vocab) \end{bmatrix} = \frac{1}{\sum_{j=1}^{|V|} \exp(\theta^{(j)T} x)} \begin{bmatrix} \exp(v(cat)^T v_I) \\ \exp(v(dog)^T v_I) \\ \exp(v(table)^T v_I) \\ \dots \\ \exp(v(Vocab)^T v_I) \end{bmatrix}$$




Word2Vec

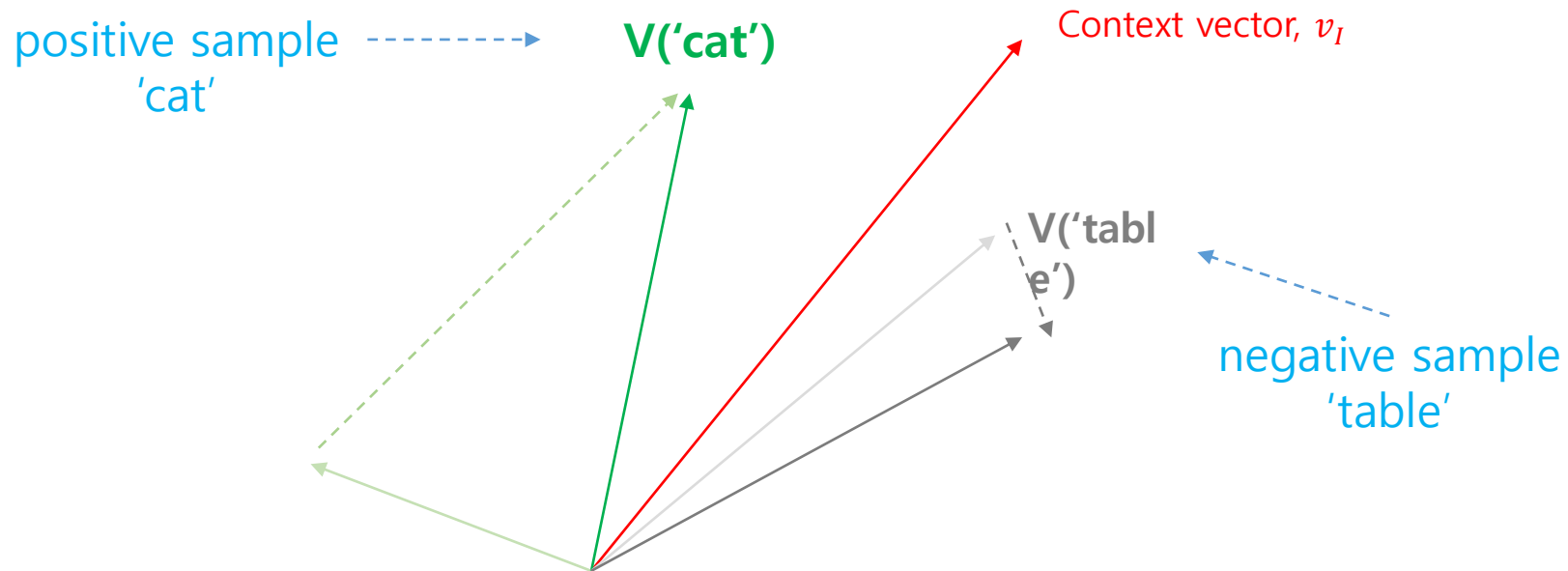
- Softmax regression 은 한 단어의 벡터를 학습하기 위해 V 개의 모든 단어 벡터를 수정하기 때문에 계산량이 큽니다.
- 학습 데이터 $\{(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})\}$ 가 주어졌을 때, loss function은

$$J(\theta) = - \left[\sum_{i=1}^m \sum_{k=1}^V 1\{w_{(t)} = cat\} \log \frac{\exp(\theta^{(j)T} x^{(i)})}{\sum_{j=1}^K \exp(\theta^{(j)T} x^{(i)})} \right]$$

Word2Vec

- Negative sampling 은 cat 이 아닌 단어 일부를 샘플링하여 밀어냅니다.
 - A positive sample 'cat'과, 수십개 수준의 negative samples

$$h_{\theta}(x) = \begin{bmatrix} P(w_{(t)} = cat) \\ P(w_{(t)} = dog) \\ P(w_{(t)} = table) \\ \dots \\ P(w_{(t)} = Vocab) \end{bmatrix} = \frac{1}{\sum_{j=1}^{|V|} \exp(\theta^{(j)T} x)} \begin{bmatrix} \exp(v(cat)^T v_I) \\ \exp(v(dog)^T v_I) \\ \exp(v(table)^T v_I) \\ \dots \\ \exp(v(Vocab)^T v_I) \end{bmatrix}$$



Word2Vec

• ~~(조금 더 디테일하게, 그래서 minor한 문제),~~

negative sampling을 할 때, 각 단어를 선택하는 확률 모델을 이용합니다

1. $U(w)^{3/4} / Z$: unigram 기준 단어 w 의 빈도수의 $3/4$ 승에 비례하여 샘플링

2. $P(w) = 1 - \sqrt{\frac{t}{f(w)}}$: negative samples 인 단어 w 를 $P(w)$ 확률로 제거

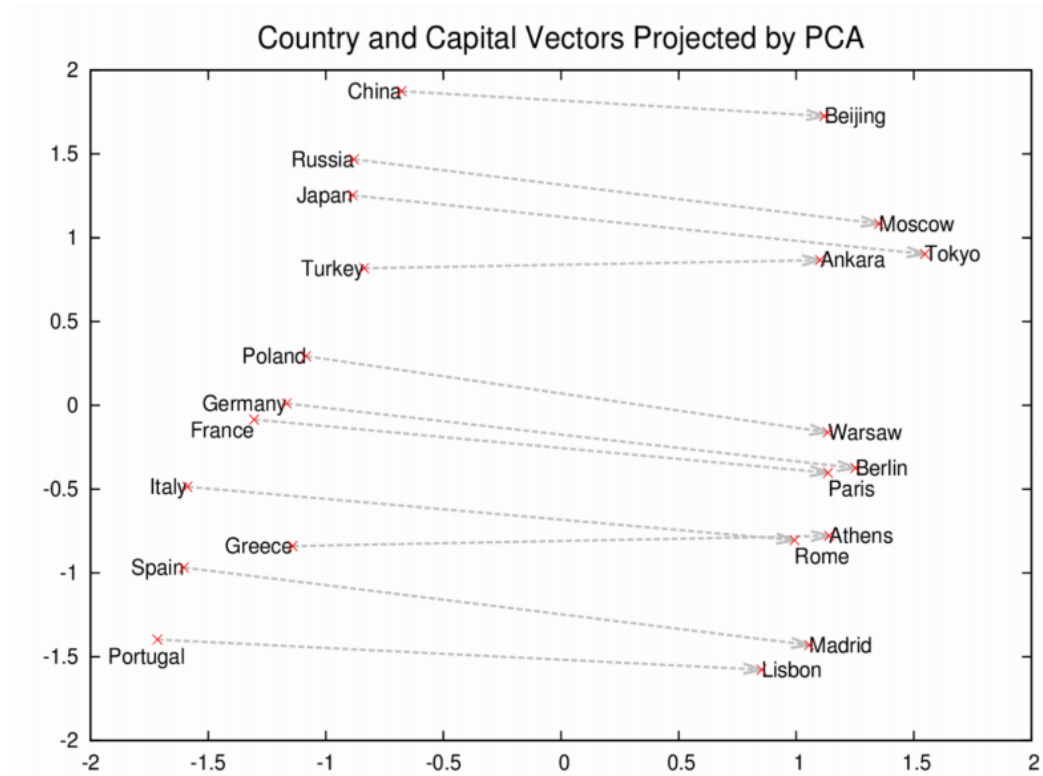
- frequent / infrequent words 의 불균형을 맞추기 위함

Word2Vec

- 'cat' 이 등장한 많은 문장이 있기 때문에, 한 번에 조금씩 $V('cat')$ 을 학습하면 여러 문맥들을 모두 고려할 수 있습니다.
 - [a, little, cat, sit, on, table],
[my, pretty, cat, ran, out],
....
 - 조금씩 학습 = 작은 learning rate

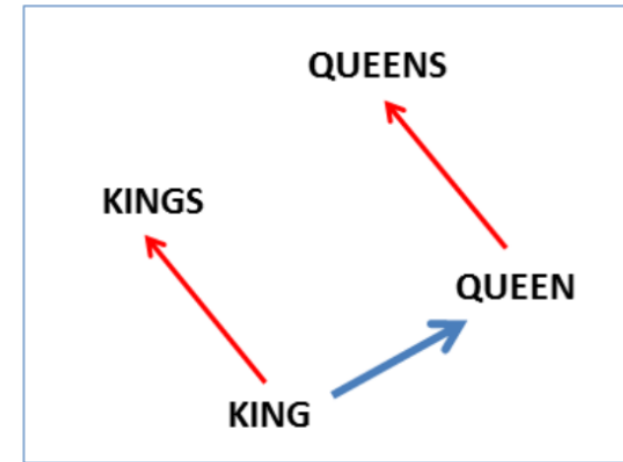
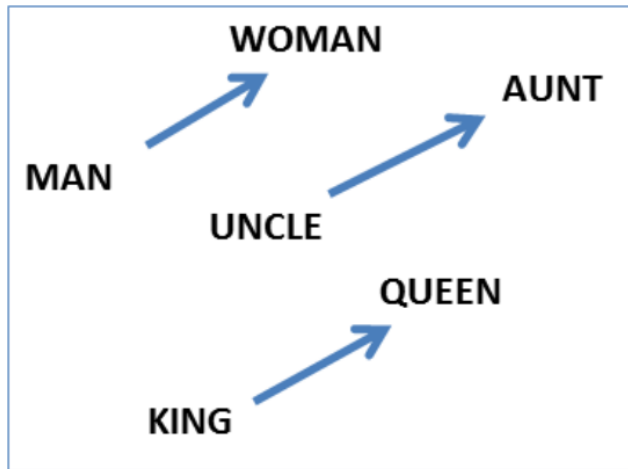
Word2Vec

- Word2Vec은 단어간의 관계성이 추출되는 효과가 있습니다.
 - “ $V(\text{나라}) - V(\text{수도})$ ” 벡터가 (나라, 수도) 에 공유됩니다.



Word2Vec

- Word2Vec은 단어간의 관계성이 추출되는 효과가 있습니다
 - $V(\text{kings}) - V(\text{king}) = V(\text{queens}) - V(\text{queen})$

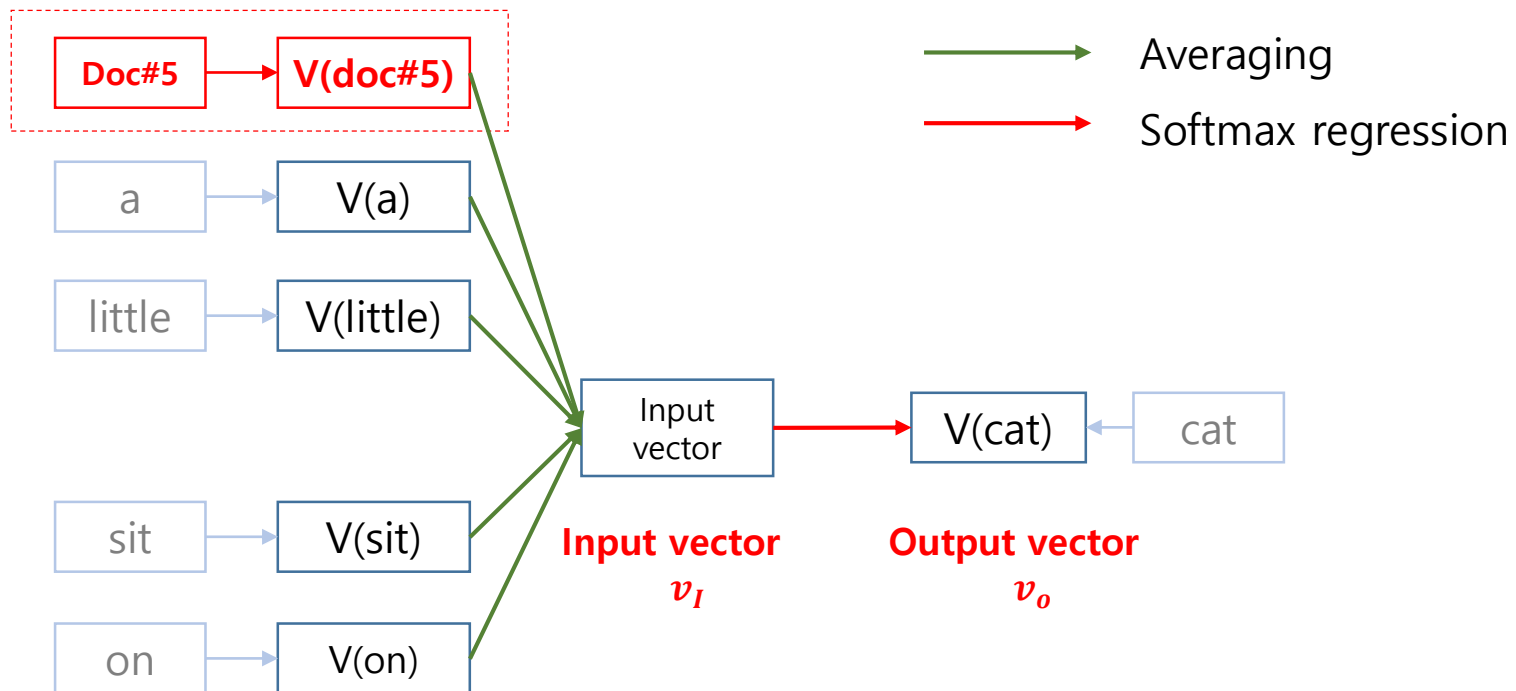


Word2Vec

- Word analogy 라 하여, 유사어 탐색 문제와 더불어 word embedding 의 정량적 평가 기준으로 널리 쓰입니다.
 - “kings : king = queens : queen” 와 같은 정답 데이터가 마련되어 있습니다.

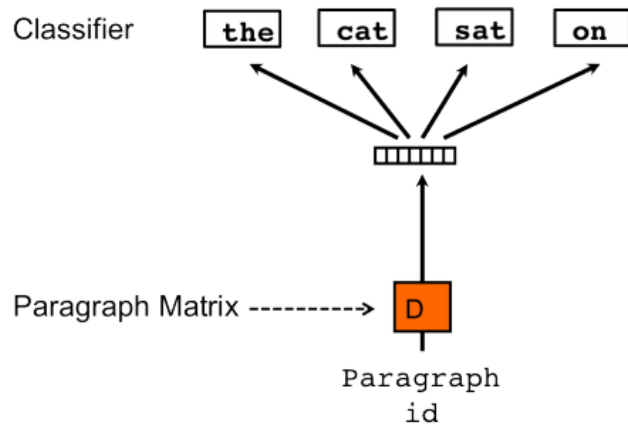
Doc2Vec

- Doc2Vec 은 Document Id 를 가상의 단어로 생각한 Word2Vec 입니다.
- Document id와 word가 같은 embedding 공간에서 학습됩니다.

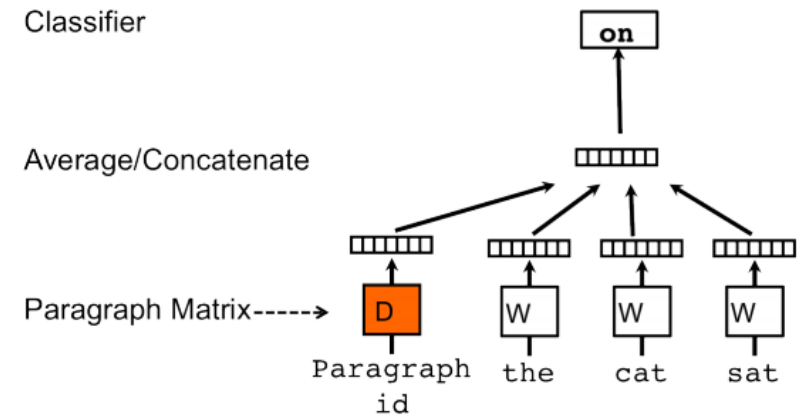


Doc2Vec

- Doc2Vec 은 Document Id 를 가상의 단어로 생각한 Word2Vec 입니다.
- Word2Vec처럼 2가지 모델 구조가 제안되었습니다.



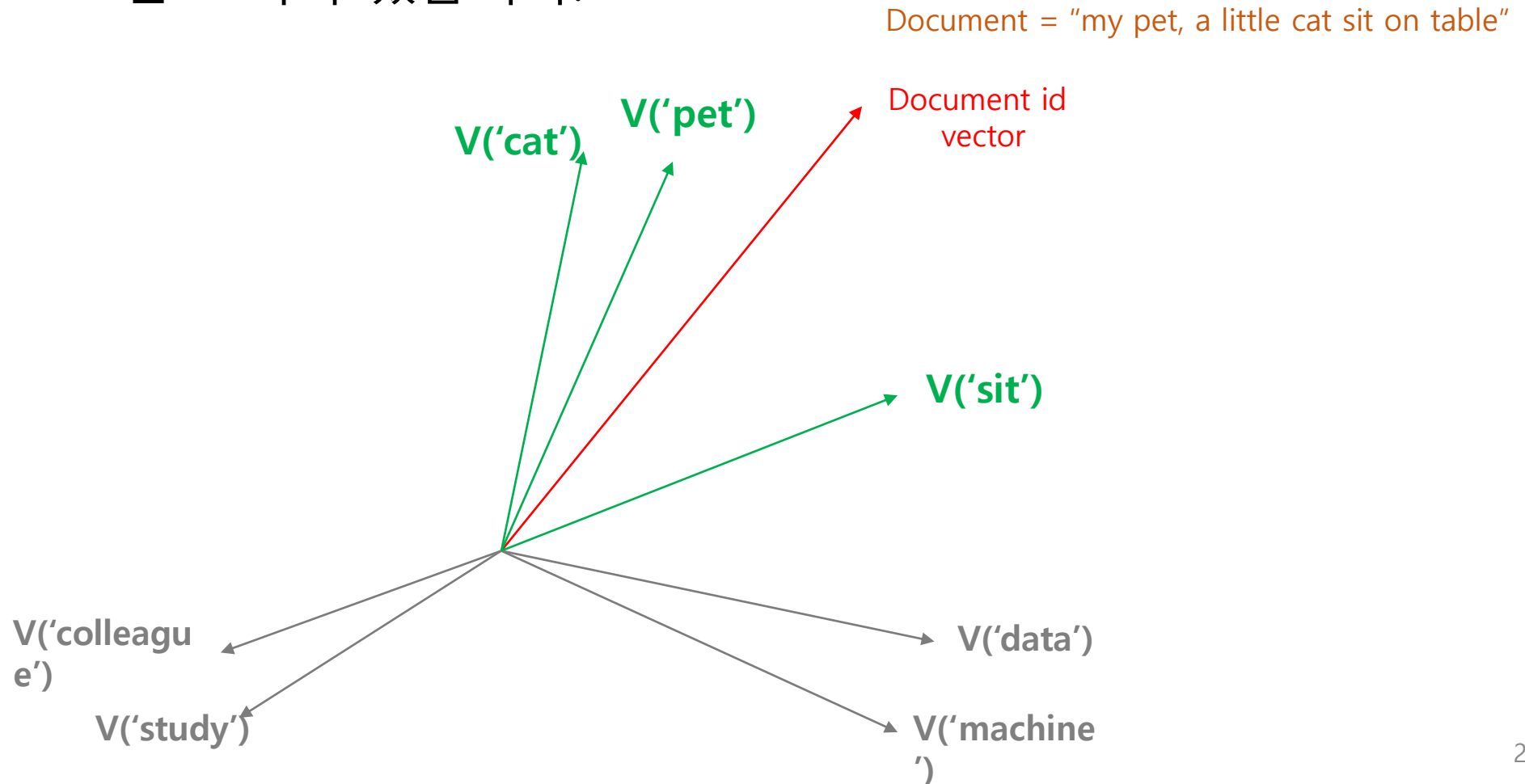
- Document id 를 단어처럼 취급
- (doc id) 로 문서 내 모든 단어를 예측하는 Softmax regression



- Document id 를 단어처럼 취급
- (Doc id, the, cat, sat) 를 이용하여 sat 다음의 단어를 예측하는 Softmax regression

Doc2Vec

- Doc2vec 은 한 문서에 등장한 단어 벡터들과 document id vector 를 한 곳에 모으는 효과가 있습니다.



Paragraph2Vec

- Document vector 간의 관계도 학습됩니다.
 - 위키피디아의 각 문서 (doc)를 이용하여 Doc2Vec 을 학습하였습니다.

Article	Cosine Similarity
Christina Aguilera	0.674
Beyonce	0.645
Madonna (entertainer)	0.643
Artpop	0.640
Britney Spears	0.640
Cyndi Lauper	0.632
Rihanna	0.631
Pink (singer)	0.628
Born This Way	0.627
The Monster Ball Tour	0.620

(a) Nearest neighbor of "Lady Gaga"

Article	Cosine Similarity
Ayumi Hamasaki	0.539
Shoko Nakagawa	0.531
Izumi Sakai	0.512
Urbangarde	0.505
Ringo Sheena	0.503
Toshiaki Kasuga	0.492
Chihiro Onitsuka	0.487
Namie Amuro	0.485
Yakuza (video game)	0.485
Nozomi Sasaki (model)	0.485

(b) "Lady Gaga" – "American" + "Japanese"

Paragraph2Vec

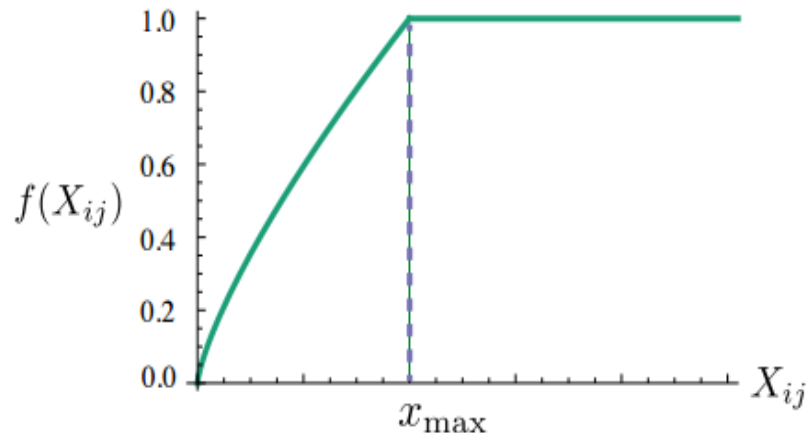
- “Lady Gaga” – “American” + “Japanese”
 - “Lady Gaga” – “American” : 미국과 관련된 단어를 전반적으로 제거합니다
 - “American” 에 관계된 단어를 “Japanese”의 단어로 치환한 것과 같습니다.

Paragraph2Vec

- Document vector 는 문서 내의 단어 벡터 (Bag of words)를 압축합니다.
 - 단어 분포의 압축이기 때문에, 문서의 주제를 표현하기에 적합합니다.
 - 소수의 단어에 큰 영향을 받지 않습니다.
 - 특정한 단어의 유무가 중요한 작업에는 적합하지 않습니다.

GloVe

- Word2Vec 은 $P(W_t|W_{[t-2:t+2]})$ 처럼 단어의 등장 확률을 보존합니다.
- GloVe 는 두 단어 w_i, w_j 의 co-occurrence frequency X_{ij} 를 보존합니다.



$$J = \sum_{i,j} f(X_{ij}) * \left(w_i^T w_j + b_i + b_j - \log(X_{ij}) \right)^2$$

- $f(X_{ij})$ 는 co-occurrence 에 따른 중요도입니다.
- 빈도수가 높은 두 단어 w_i, w_j 에 집중합니다.

FastText

- Out of vocabulary 문제를 해결하기 위하여 제안되었습니다.
- Word Piece Model 도 미등록단어 문제를 해결하기 위하여 단어를 subwords 로 표현합니다.
 - 그러나 WPM 은 겹치지않는 subwords 를 이용합니다.
 - 'appear' → 'app' + 'ear'

FastText

- FastText 는 bag of character n-gram으로 단어를 표현합니다.
 - 단어의 시작과 끝을 구분하기 위해 <, > 추가
 - Character 3-gram:
 - 'where' → '<wh, whe, her, ere, re>'
 - 논문에서는 3 ~ 6 gram 의 모든 subwords 를 합쳐서 사용

FastText

- 단어 w 의 벡터 $\mathbf{W}_{(t)}$ 를 subwords 벡터의 합으로 표현합니다
 - $\mathbf{W} = \text{where}$
 - $\mathbf{W}_{(t)} = \sum_{g \in G_w} \mathbf{z}_g$
 - 3 – 6 grams 을 이용한다면,
 - $\mathbf{z}_g = [<\text{wh}, \text{whe}, \text{ere}, \text{re}>, <\text{whe}, \text{wher}, \dots, <\text{where}, \text{where}>]$

FastText

- Fasttext 는 노이즈가 있는 단어도 비슷한 벡터로 표현할 수 있습니다.

- where → <wh, whe, her, ere, re>

서로 다른 subwords

- wherre → <wh, whe, her, err, rre, re>

- 대부분의 subwords 가 공통으로 존재하기 때문에 'where', 'wherre' 의 벡터는 비슷합니다.

- 각 단어의 벡터는 subword vectors 의 합으로 표현됩니다.

FastText

Word2Vec Loss

$$\log(1 + \exp(-w_{(t)}^T w_{[t-w:t+w]})) + \sum_{n \in N_{t,c}} \log(1 + \exp(w_{(t)}^T w_n))$$



FastText Loss

$$\underbrace{\log\left(1 + \exp\left(-\sum_{g \in G_w} w_g^T w_{[t-w:t+w]}\right)\right)}_{\text{loss from positive samples}} + \sum_{n \in N_{t,c}} \underbrace{\log\left(1 + \exp\left(\sum_{g \in G_w} w_g^T w_{[t-w:t+w]}\right)\right)}_{\text{loss from negative samples}}$$

FastText (text classification)

- Facebook Research github* 에는 세 개의 논문이 참조되어 있습니다.

1. Enriching Word Vectors with Subword Information

2. Bag of Tricks for Efficient Text Classification

3. FastText.zip: Compressing text classification models

(효율적인 문서 분류를 위한 모델 압축)

- 앞서 말한 unsupervised word embedding 은 1번 논문입니다.

FastText (text classification)

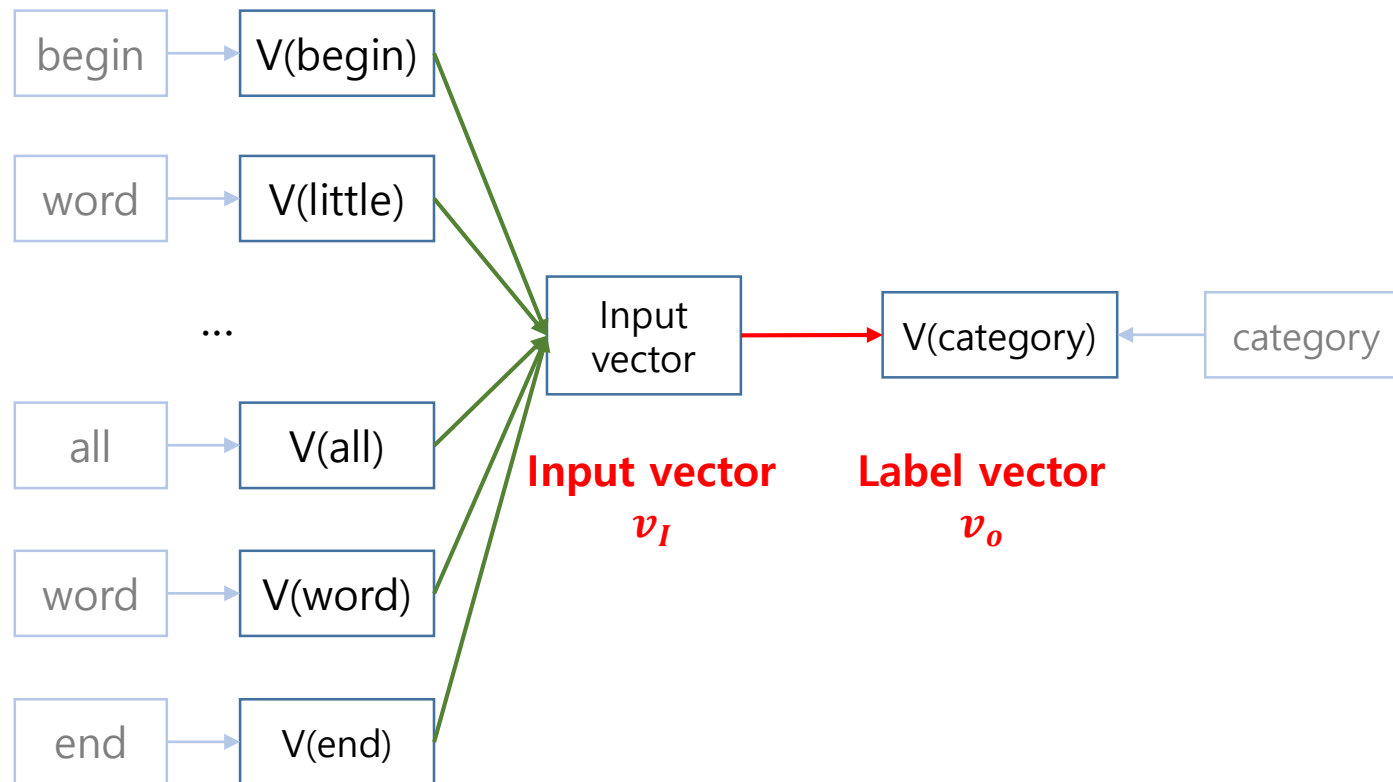
- 2 번 논문은 문서 분류를 위한 단어 임베딩 방법입니다.
 - Word2Vec, Doc2Vec, GloVe, FastText (1 번 논문) 은 다른 목적을 위한 word embedding 이 아니었습니다.
 - 이들은 단어의 문맥이나 (Word2Vec), 문서 내 단어 분포를 보존합니다 (Doc2Vec).
 - Document classification 이 목적이기 때문에 이를 위한 단어 벡터를 학습합니다.

FastText (text classification)

- Document classification 이 목적입니다.
 - 문서 분류의 정답 label 이 있으니 이 정보를 적극적으로 활용하여 이에 적합한 단어의 임베딩 벡터를 학습해야 합니다.
 - 문맥의 유사성 기준에서는 'good', 'bad' 은 비슷합니다.
 - 하지만 감성 분류에서는 'good' 과 'bad' 는 서로 다르게 표현되어야 합니다.

FastText (text classification)

- 문서 내 모든 단어가 average 됩니다.
- 단어가 아닌 label 을 예측합니다.



$$y_j = \frac{\exp(v_I^T v_{L_j})}{\sum_j \exp(v_I^T v_{L_j})}$$

—→ Averaging
—→ Softmax regression

FastText (text classification)

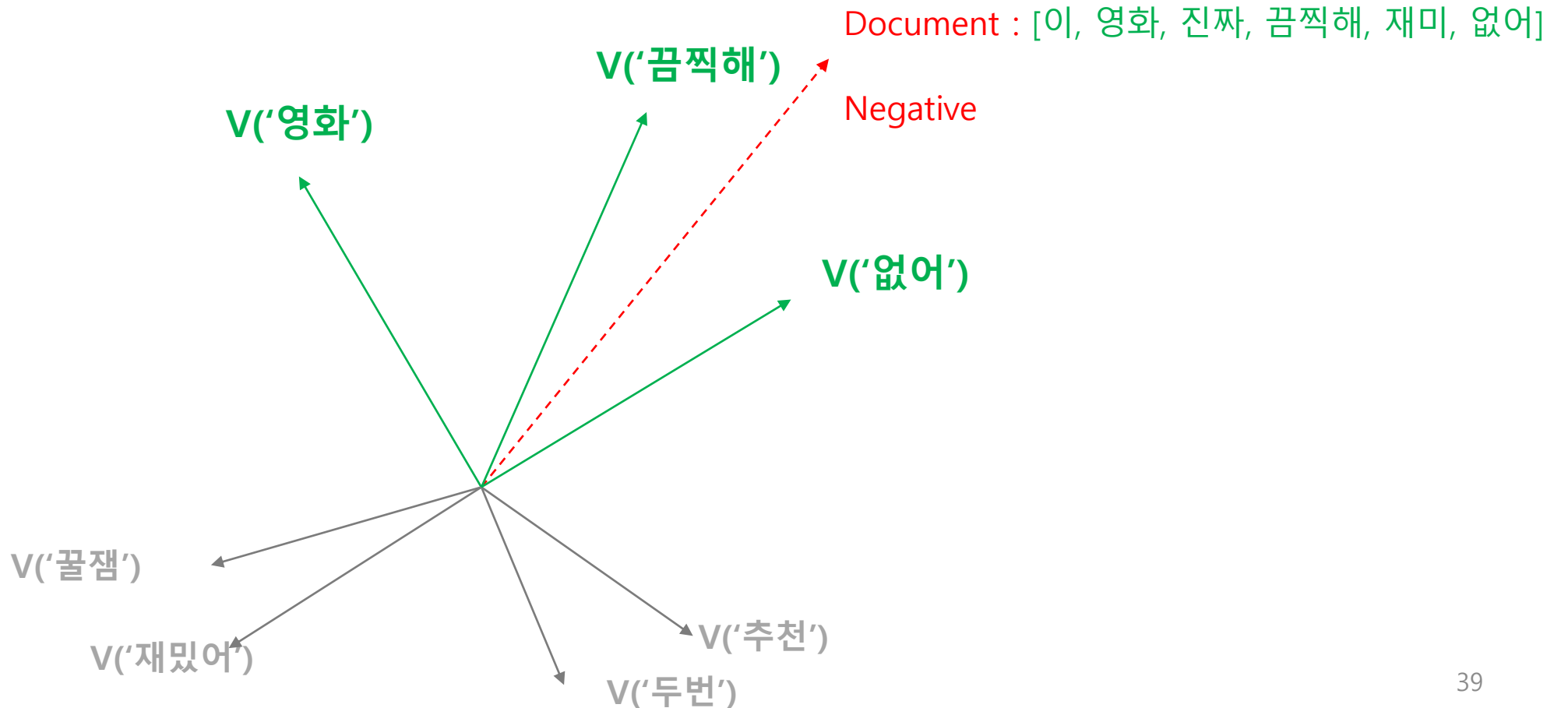
- 다양한 데이터의 실험에서 좋은 성능을 보여줍니다.
- 복잡한 딥러닝 모델들과도 비슷한 성능을 보입니다.
- 애초에 Bag of Words 모델이 기본적으로 좋은 성능을 보입니다.

Model	AG	Sogou	DBP	Yelp P.	Yelp F.	Yah. A.	Amz. F.	Amz. P.
BoW (Zhang et al., 2015)	88.8	92.9	96.6	92.2	58.0	68.9	54.6	90.4
ngrams (Zhang et al., 2015)	92.0	97.1	98.6	95.6	56.3	68.5	54.3	92.0
ngrams TFIDF (Zhang et al., 2015)	92.4	97.2	98.7	95.4	54.8	68.5	52.4	91.5
char-CNN (Zhang and LeCun, 2015)	87.2	95.1	98.3	94.7	62.0	71.2	59.5	94.5
char-CRNN (Xiao and Cho, 2016)	91.4	95.2	98.6	94.5	61.8	71.7	59.2	94.1
VDCNN (Conneau et al., 2016)	91.3	96.8	98.7	95.7	64.7	73.4	63.0	95.7
fastText, $h = 10$	91.5	93.9	98.1	93.8	60.4	72.0	55.8	91.2
fastText, $h = 10$, bigram	92.5	96.8	98.6	95.7	63.9	72.3	60.2	94.6

Table 1: Test accuracy [%] on sentiment datasets. FastText has been run with the same parameters for all the datasets. It has 10 hidden units and we evaluate it with and without bigrams. For char-CNN, we show the best reported numbers without data augmentation.

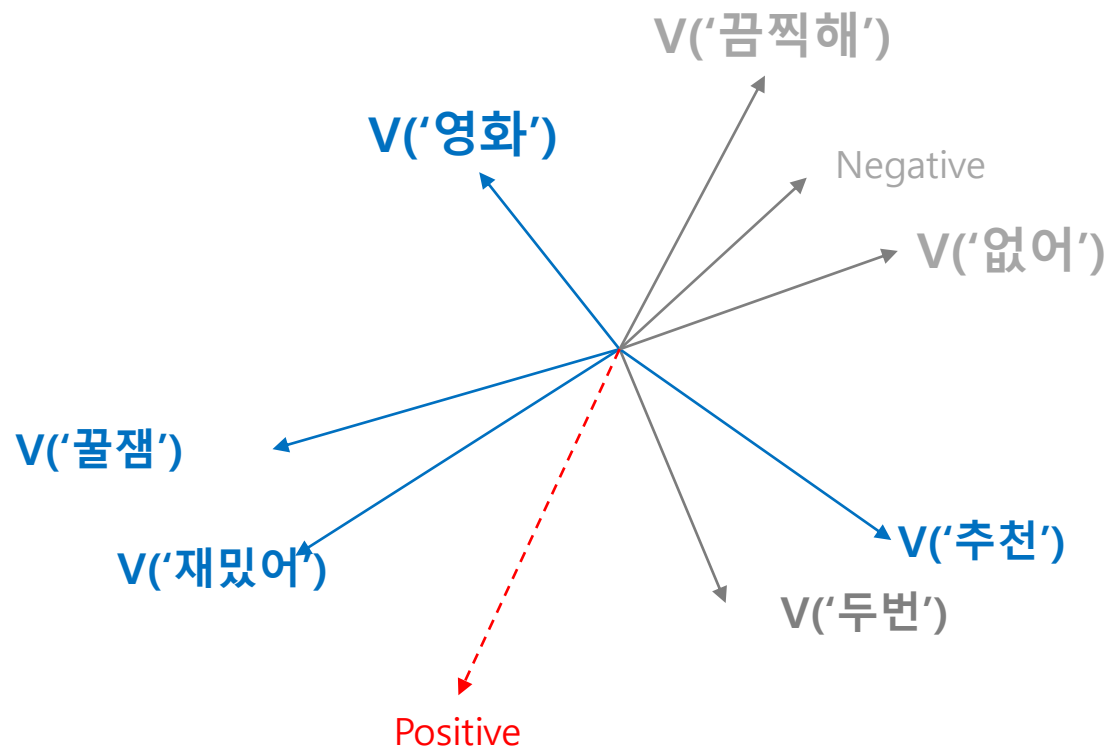
FastText (text classification)

- FastText 은 특정 label 에 자주 등장한 단어를 label 방향으로 당겨옵니다.



FastText (text classification)

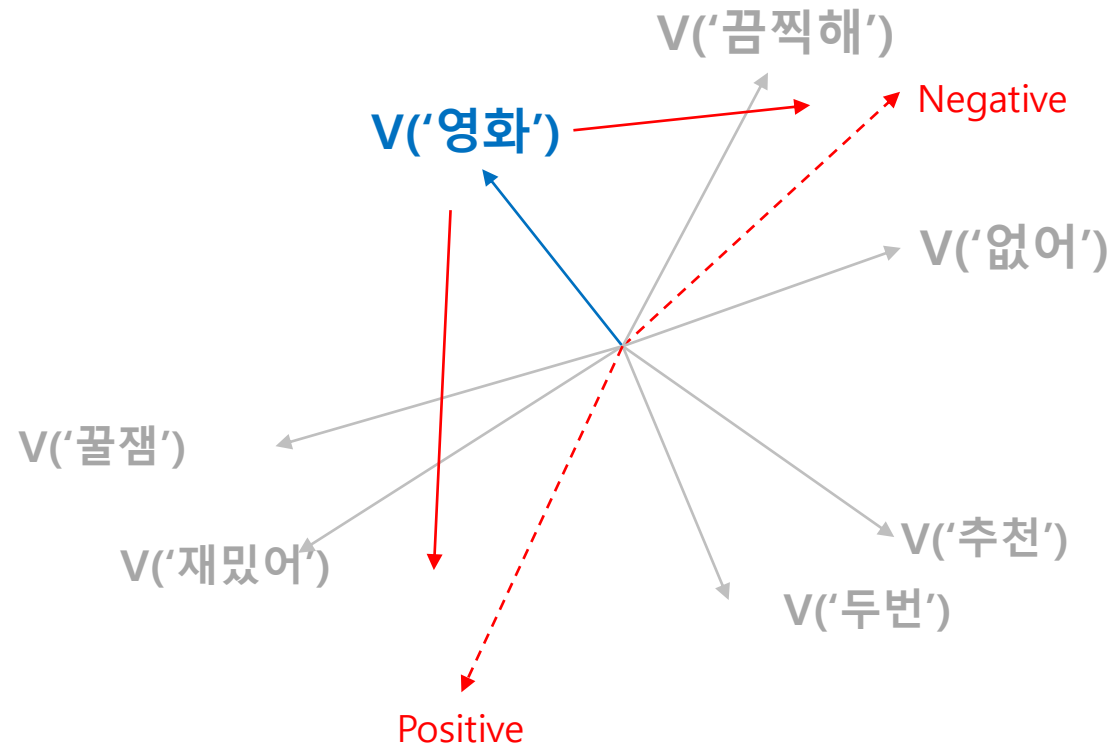
- FastText 은 특정 label 에 자주 등장한 단어를 label 방향으로 당겨옵니다.



Document : [이, 영화, 진짜, 꿀잼, 완전, 재밌어, 추천, 해]

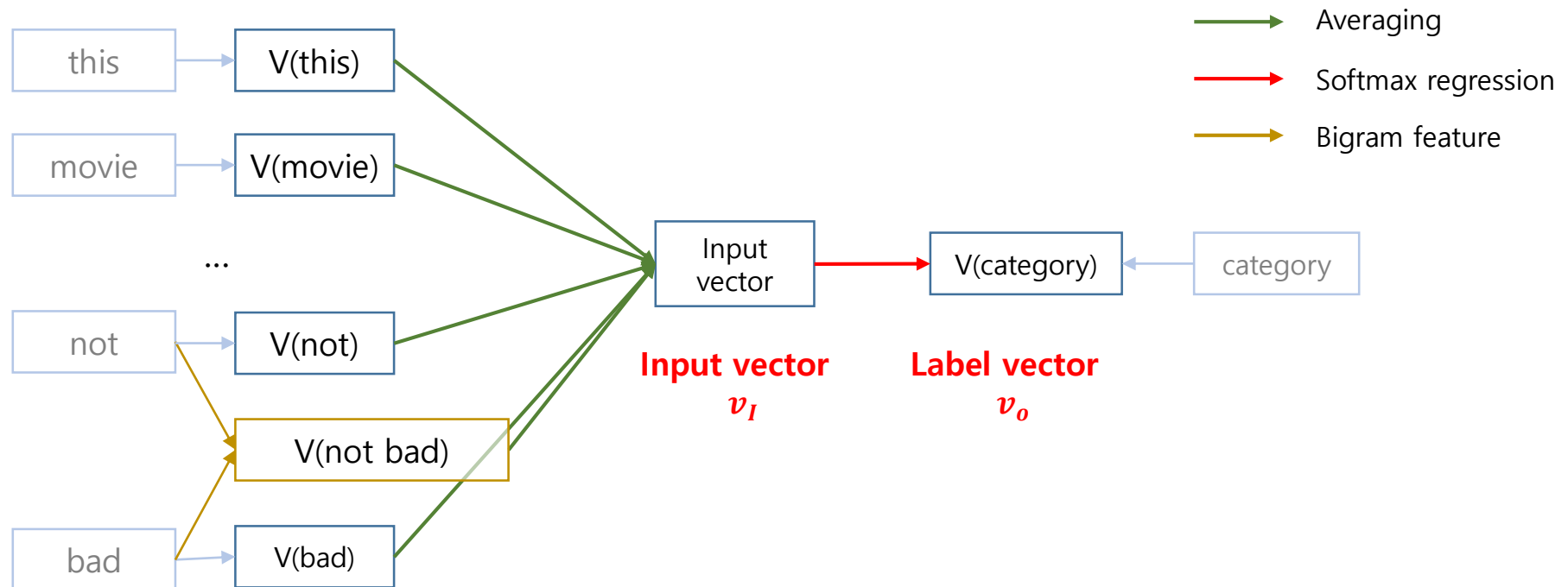
FastText (text classification)

- 여러 labels 에 모두 등장하는 단어는 어떤 label 과도 가까워지지 않습니다.



FastText (text classification)

- Bigram + Linear model 은 document classification 의 base model 입니다.
 - 'not bad' 는 negative 가 아니지만, unigram 은 이를 반영하지 못합니다.
 - Bigram 까지만 이용해도 이 문제는 충분히 해결됩니다.



Notes of embedding methods

Skip-gram vs. explicit representation

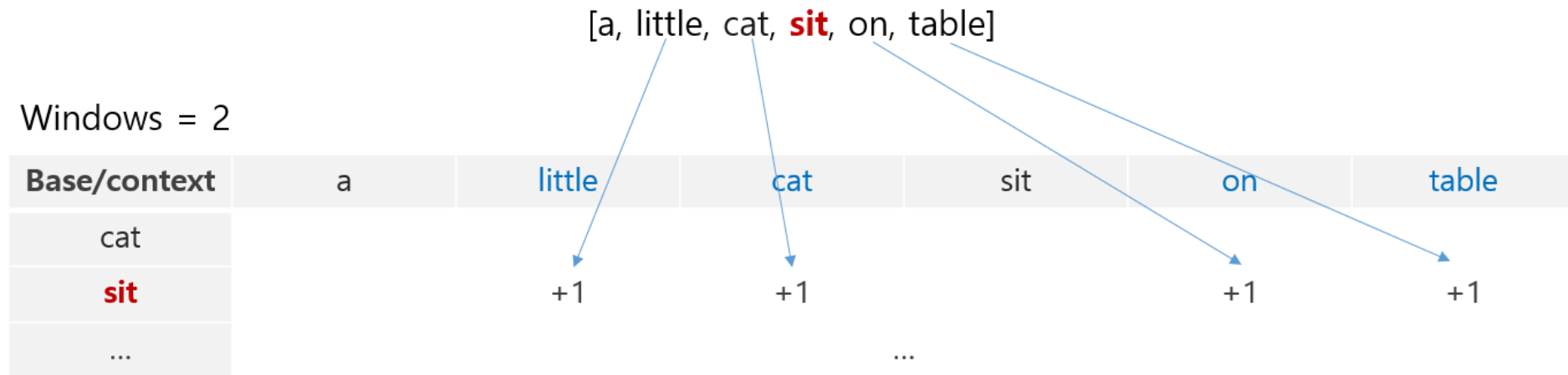
- Levy and Goldberg (2014) 는 Word2Vec 의 skip-gram 은 (word, context) co-occurrence 에 Shifted PPMI 를 적용한 것과 같음을 증명하였습니다.

$$loss = \sum_{w \in V_W} \sum_{c \in V_C} \#(w, c) \cdot (\log \sigma(\vec{w} \cdot \vec{c}) + k \cdot E_{C_N P_D}[\log \sigma(-\vec{w} \cdot \vec{c}_N)])$$

$$\vec{w} \cdot \vec{c} = \log \left(\frac{\#(w, c) \cdot |D|}{\#(w) \cdot \#(c)} \cdot \frac{1}{k} \right) = \log \left(\frac{\#(w, c) \cdot |D|}{\#(w) \cdot \#(c)} \right) - \log(k)$$

Skip-gram vs. explicit representation

- (word – context) pair 의 context 는 word 와 앞/뒤로 windows 안에 함께 등장한 단어입니다.



Skip-gram vs. explicit representation

- Shifted PPMI 는 k 보다 큰 PMI 만 값을 보존하며, 이보다 작은 값은 0 으로 변환합니다.

$$SPPMI_k(x, y) = \max(0, PMI(x, y) - \log(k))$$

Skip-gram vs. explicit representation

- Co-occurrence + PMI 를 이용하는 방법을 explicit representation 이라 합니다. 문맥의 단어를 해석할 수 있는 장점이 있습니다.
- Word2Vec 처럼 dense vector representation 을 얻기 위해서, co-occurrence matrix 에 SVD 를 적용할 수 있습니다.

Skip-gram vs. GloVe

- Skip-gram 은 (Shifted) PMI 정보를 학습합니다.

$$\vec{w} \cdot \vec{c} = \log \left(\frac{\#(w, c) \cdot |D|}{\#(w) \cdot \#(c)} \cdot \frac{1}{k} \right) = \log \left(\frac{\#(w, c) \cdot |D|}{\#(w) \cdot \#(c)} \right) - \log(k)$$

Skip-gram vs. GloVe

- GloVe 에서 bias 를 학습하지 않는다면, co-occurrence 값을 학습합니다.

$$J = \sum_{i,j} f(X_{ij}) * \left(w_i^T w_j - \log(X_{ij}) \right)^2$$

Skip-gram vs. GloVe

- 각 단어의 bias 를 각 단어의 빈도수로 지정하면 GloVe 의 embedding vector 도 co-occurrence 의 PMI 를 학습합니다.

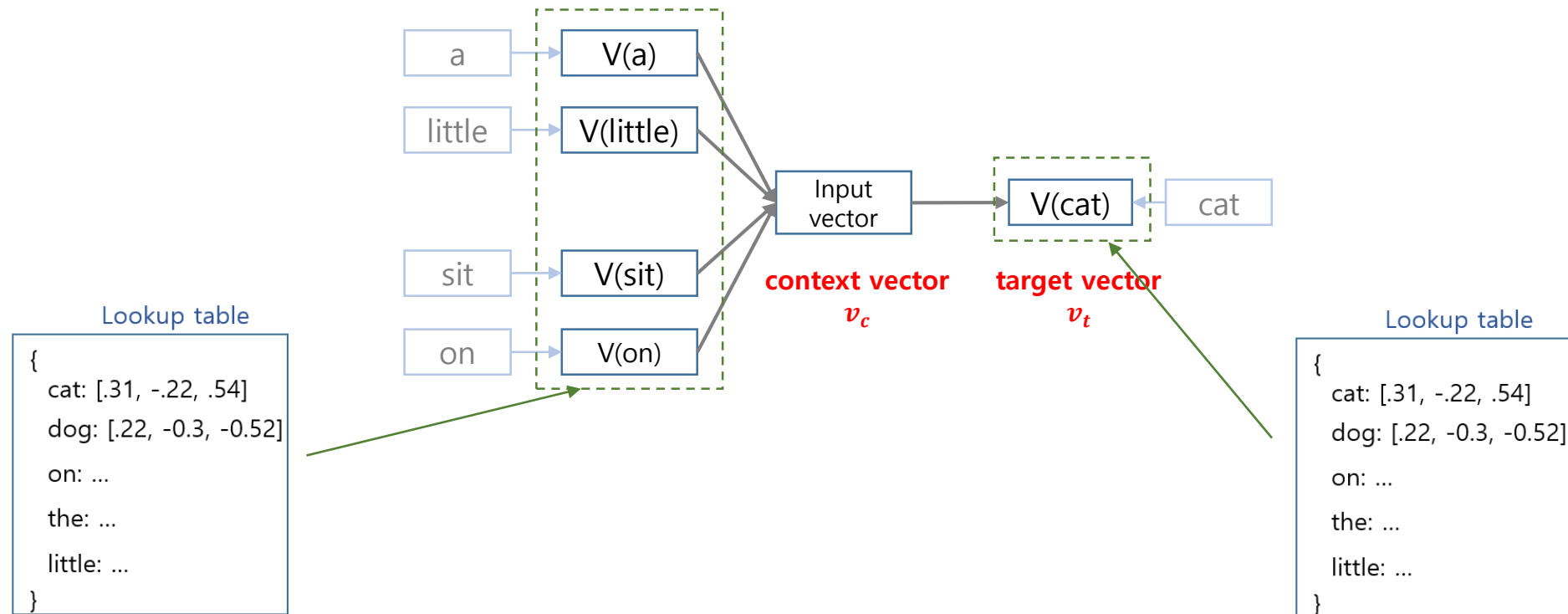
$$J = \sum_{i,j} f(X_{ij}) * \left(w_i^T w_j + b_i + b_j - \log(X_{ij}) \right)^2$$

- 우리는 bias 를 단어의 빈도수로 고정하지는 않습니다.
하지만 GloVe 는 b_i 가 w_i 의 빈도수와 비슷한 값이 되도록 학습합니다.

-
- Word2Vec, GloVe, explicit representation 모두 (word, context) 의 co-occurrence 정보를 이용합니다.

Two embedding space in Word2Vec

- 앞의 Word2Vec 의 설명은 개념적인 설명이며, 실제 구현체에서는 두 개의 lookup tables 이 존재하기도 합니다
 - Target / context words 가 서로 다른 lookup tables 을 이용합니다



Two embedding space in Word2Vec

- 앞의 Word2Vec 의 설명은 개념적인 설명이며, 실제 구현체에서는 두 개의 lookup tables 이 존재하기도 합니다
 - Context vector 주위로 target words 가 움직이는 현상은 동일하며,
 - 우리는 target words embedding vector 를 이용합니다.
- Context vector 를 만들 때 concatenation 을 한다면 target words 의 lookup tables 의 각 벡터의 크기는 context vector 보다 $2 \times w$ 배 큼니다.

Highlighting contexts

- GloVe 는 harmonic weight function 을 이용합니다.
 - Co-occurrence 를 계산할 때, 기준 단어와 가까울수록 유의미한 문맥일 가능성이 높습니다.
 - window = 3 일 때, 거리에 반비례한 가중치, $[\frac{1}{3}, \frac{2}{3}, \frac{3}{3}]$ 을 곱한 co-occurrence 를 이용한다면 문맥이 더 잘 강조될 수 있습니다.

Highlighting contexts

- Negative sampling 구현 시, context words 는 negative samples 이 될 수 없도록 강제하기도 합니다.

Handling rare words

- Word2Vec 의 negative sampling 은 아래의 분포를 이용합니다.

- $U(w)^{3/4} / Z$

- 이는 PMI 를 다음처럼 변형하는 것과 같습니다.

- $PMI_{\alpha}(w, c) = \log \frac{\hat{P}(w, c)}{\hat{P}(w)\hat{P}(c)}$,

where $\hat{P}_{\alpha}(c) = \frac{\#(c)^{\alpha}}{\sum \#(c)^{\alpha}}$, $0 < \alpha < 1$, normally $\alpha = 0.75$

- Infrequent prob. 에 의한 왜곡을 방지합니다.