GloVe

- Word2vec과 LSA의 단점을 극복하기 위해 등장

> While methods like LSA efficiently leverage statistical information, they do relatively poorly on the word analogy task, indicating a sub-optimal vector space structure. Methods like skip-gram may do better on the analogy ask, but they poorly utilize the statistics of the corpus since they train on separate local context windows instead of on global co-occurrence counts.

	LSA	Word2Vec
장점	말뭉치 전체의 통계량 모두 활용	단어 간 유사도 측정에 유리
단점	단어 간 유사도 측정이 어려움	지정한 윈도우 내의 로컬 문맥만 학습

- 목표: **임베딩된 단어 벡터 간 유사도 측정을 수월하게 하면서도 말뭉치 전체의 통계 정보를 더 잘 반영하자**
- 모델의 기본 구조
  - 동시등장확률(the words' probability of co-occurence)

Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

  - 특정 단어 k가 주어졌을 때 임베딩된 두 단어 벡터의 내적이 동시등장확률 간 비율이 되도록 임베딩 후 전체를 한 번에 반영하는 것이 GloVe의 아이디어
- 임베딩 과정 (p.146 그림으로 이해하는 GloVe 참고)

1. 학습 말뭉치를 대상으로 **단어-문맥 행렬** A를 만든다.
2. 목적 함수를 최소화하는 임베딩 벡터를 찾기 위해 행렬 분해를 수행한다.
3. 학습 손실이 줄지 않거나 정해진 스텝 수까지 학습한 경우 학습을 종료한다.
4. U나 $U + V^T$, $concatenate([U, V^T])$ 등을 단어 임베딩으로 사용한다.

  - 이 때 동시등장확률을 계산하기 위한 목적함수는 아래와 같음 (V는 어휘 집합 크기)

- $F(w_i, w_j, \tilde{w}_k) = \frac{P_{ik}}{P_{jk}}$ 인 F를 찾자!

$$F(w_{ice}, w_{steam}, w_{solid}) = \frac{P_{ice,solid}}{P_{steam,solid}} = \frac{P(solid|ice)}{P(solid|steam)} = \frac{1.9\times10^{-4}}{2.2\times10^{-5}} = 8.9$$

→ 임베딩 크기의 관계 비를 인코딩하기 위해 내적의 형태로 (양쪽을 스칼라 형태로)

$$F((w_i - w_j)^T \tilde{w}_k) = \frac{P_{ik}}{P_{jk}}$$

→ 이 때 F의 조건

① $\tilde{w}_k$는 다른 단어로 바뀔 수 있음. 따라서 w_i, w_k가 서로 바뀌어도 같은 결과. $w_i \leftrightarrow \tilde{w}_k$

② co-occurence matrix는 대칭 $X \leftrightarrow X^T$

③ homomorphism $F(X-Y) = \frac{F(X)}{F(Y)}$

조건③ → $F((w_i - w_j)^T \tilde{w}_k) = F(w_i^T \tilde{w}_k - w_j^T \tilde{w}_k) = \frac{F(w_i^T \tilde{w}_k)}{F(w_j^T \tilde{w}_k)}$ 를 만족하는 F: 지수함수

→ $\exp(w_i^T \tilde{w}_k) / \exp(w_j^T \tilde{w}_k) = P_{ik}/P_{jk}$

→ $\exp(w_i^T \tilde{w}_k) = P_{ik} = X_{ik}/X_i$ (동시등장 / $\sum_k X_{ik}$)

$w_i^T \tilde{w}_k = \log P_{ik} = \log X_{ik} - \log X_i$

$w_i^T \tilde{w}_k + \log X_i = \log X_{ik}$

조건① → 이때 $\log P_{ik} = \log X_{ik} - X_i \neq \log P_{ki} = \log X_{ki} - X_k$

∴ $\log X_i$를 상수 항으로 처리

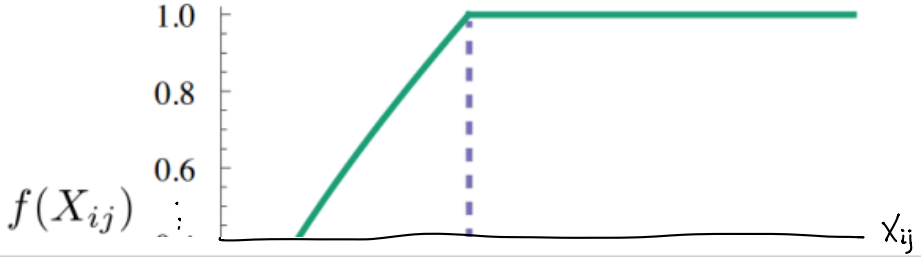
$w_i^T \tilde{w}_k = \log X_{ik} - b_i - \tilde{b}_k$

$w_i^T \tilde{w}_k + b_i + \tilde{b}_k = \log X_{ik}$

⇒ 우변과 차이를 최소로 하는 좌변의 값을 찾자!

∴ 목적함수 $J = \sum_{i,j=1}^{V} (w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij})^2$ 를 최소화하는 미지수 w_i, w_j, b_i, b_j

- X_{ij} (동시출현빈도)가 특정 값 이상일 경우 그 영향을 줄여 임베딩의 품질을 높이기 위한 함수 $f(X_{ij})$를 목적함수에 곱해줌



In [ ]:
```
cd /content/drive/My Drive/프로그래밍/NLP/한국어임베딩/embedding
```

In [2]:
```
# $'\r': command not found 오류 수정
!sed -i 's/\r$//' preprocess.sh
```

In [ ]:
```
# 데이터 다운로드
!bash preprocess.sh dump-tokenized
```

In [ ]:
```
cd /notebooks/embedding/data
```

In [14]:
```
# 데이터 합치기
!cat tokenized/wiki_ko_mecab.txt tokenized/ratings_mecab.txt tokenized/korquad_mecab.txt > tokeni
zed/corpus_mecab.txt
```

In [18]:
```
# GloVe 학습

!cd /notebooks/embedding
!mkdir -p /notebooks/embedding/data/word-embeddings/glove
```

In [ ]:
```
cd /content/drive/My Drive/프로그래밍/NLP/한국어임베딩/embedding/
```

In [19]:
```
##  자신이 가진 데이터(단 형태소 분석이 완료되어 있어야 함)로 임베딩하고 싶다면 아래 스크립트에서 /notebooks/embedding/
data/tokenized/corpus_mecab.txt를 해당 데이터 경로로 바꿔주면 됩니다.

!/notebooks/embedding/models/glove/build/vocab_count -min-count 5 -verbose 2 < /notebooks/embeddi
ng/data/tokenized/corpus_mecab.txt > /notebooks/embedding/data/word-embeddings/glove/glove.vocab
!/notebooks/embedding/models/glove/build/cooccur -memory 10.0 -vocab-file /notebooks/embedding/da
ta/word-embeddings/glove/glove.vocab -verbose 2 -window-size 15 < /notebooks/embedding/data/token
ized/corpus_mecab.txt > /notebooks/embedding/data/word-embeddings/glove/glove.cooc
!/notebooks/embedding/models/glove/build/shuffle -memory 10.0 -verbose 2 < /notebooks/embedding/d
ata/word-embeddings/glove/glove.cooc > /notebooks/embedding/data/word-embeddings/glove/glove.shuf
!/notebooks/embedding/models/glove/build/glove -save-file /notebooks/embedding/data/word-embeddin
gs/glove/glove.vecs -threads 4 -input-file /notebooks/embedding/data/word-embeddings/glove/glove.
shuf -x-max 10 -iter 15 -vector-size 100 -binary 2 -vocab-file /notebooks/embedding/data/word-emb
eddings/glove/glove.vocab -verbose 2
```

```
/bin/bash: /notebooks/embedding/models/glove/build/vocab_count: No such file or directory
/bin/bash: /notebooks/embedding/models/glove/build/cooccur: No such file or directory
/bin/bash: /notebooks/embedding/models/glove/build/shuffle: No such file or directory
/bin/bash: /notebooks/embedding/models/glove/build/glove: No such file or directory
```

Swivel (Submatrix-Wise Vector Embedding Learner)

- GloVe는 단어-문맥 행렬을 분해하여 사용한다면, Swivel은 PMI 행렬을 분해하여 임베딩을 진행한다.
- 모델 기본 구조
  - 목적함수

1. 타깃 단어 i와 문맥 단어 j가 윈도우 내에서 동시에 한 번이라도 등장한 적이 있는 경우

$$J = \frac{1}{2} f(x_{ij})(w_i^T \tilde{w}_j - \mathbf{pmi}(i;j))^2$$

   - w_i 와 $\tilde{w}_j$ 의 내적이 실제 PMI와 일치하도록 업데이트
   - 단어 i, j의 동시 등장 빈도 ($f(x_{ij})$)가 클수록(두 단어가 자주 같이 등장할수록) w_i 와 $\tilde{w}_j$ 의 내적이 실제 PMI와 비슷해야 학습 손실이 줄어듦

2. 말뭉치에 동시 등장한 케이스가 한 건도 없는 경우

$$J = log[1 + exp(w_i^T \tilde{w}_j - \mathbf{pmi}(i;j))]$$

   - 단어 i와 j가 각각 고빈도 단어이지만 동시 등장 빈도가 0이라면 의미상 관계가 없는 단어일 것이라고 가정
     → 두 단어의 내적 값이 PMI(한 번이라도 같이 등장했다고 가정했을 때)보다 조금 더 작게 되도록 학습
   - 저빈도 단어인데 동시 등장 빈도가 0이라면 의미상 관계가 어느정도 있을 수도 있다고 가정 → 두 단어의 내적 값이 PMI보다 조금 더 크게 되도록 학습

In [ ]:
```
# 모델 학습
!mkdir -p /notebooks/embedding/data/word-embeddings/swivel
```

In [28]:
```
cd /content/drive/My Drive/프로그래밍/NLP/한국어임베딩/embedding
```

```
/content/drive/My Drive/프로그래밍/NLP/한국어임베딩/embedding
```

In [ ]:
```
!models/swivel/fastprep --input /notebooks/embedding/data/tokenized/corpus_mecab.txt --output_dir
 /notebooks/embedding/data/word-embeddings/swivel/swivel.data
```

In [ ]:
```
!python /notebooks/embedding/models/swivel/swivel.py --input_base_path /notebooks/embedding/data/
word-embeddings/swivel/swivel.data --output_base_path /notebooks/embedding/data/word-embeddings/s
wivel --dim 100
```

단어 임베딩 평가

- 단어 간 통사적(syntactic), 의미론적(semantic) 관계가 얼마나 잘 녹아 있는지 정량적으로 평가
- 단어 유사도 평가 (word similarity test)
  - 단어 쌍을 구성한 후 사람이 평가한 점수와 단어 벡터 간 코사인 유사도 사이의 상관관계를 계산해 단어 임베딩의 품질을 평가
  - 상관관계는 스피어만, 피어슨을 사용
  - Wordsim 데이터를 사용하여 실험 한 결과 예측 기반 임베딩 기법(Word2Vec, FastText)이 행렬 분해 방법(GloVe, Swivel)보다 의미적 관계가 잘 녹아 있다.
- 단어 유추 평가 (word analogy test)
  - 의미론적 유추에서 단어 벡터 간 계산을 통해 특정 질의에 대해 원하는 단어를 도출할 수 있는지 평가
  - 이 역시 평가 데이터셋과 단어 임베딩 데이터가 필요
  - 아래의 실험에서 단어 유추 평가에 있어서는 Word2Vec과 GloVe가 좋은 성능
- 단어 임베딩 시각화
  - 의미가 유사한 단어를 사람이 쉽게 이해할 수 있는 그림으로 표현해 품질을 정성적으로 확인
  - 사람이 인식하는 2, 3차원으로 축소해 시각화 (t-Stochastic Neighbor Embedding; t-SNE)

In [ ]:
```
cd /content/drive/My Drive/프로그래밍/NLP/한국어임베딩/embedding
```

In [ ]:
```
# 학습된 단어 임베딩 다운로드
!bash preprocess.sh dump-word-embeddings
```

In [ ]:
```
!pip install fasttext
```

In [ ]:
```
!pip install soynlp
```

In [ ]:
```
!git clone https://github.com/kakao/khaiii.git
```

In [ ]:
```
!pip install cmake
```

In [3]:
```
!mkdir build
```

In [ ]:
```
!cd build && cmake /content/khaiii
```

In [ ]:
```
!cd /content/build/ && make all
```

In [ ]:
```
!cd /content/build/ && make resource
```

In [ ]:
```
!cd /content/build && make install
```

In [ ]:
```
!cd /content/build && make package_python
```

In [ ]:
```
!pip install /content/build/package_python
```

In [ ]:
```
# ! git clone https://github.com/SOMJANG/Mecab-ko-for-Google-Colab.git
```

In [ ]:
```
cd Mecab-ko-for-Google-Colab
```

In [ ]:
```
! bash install_mecab-ko_on_colab190912.sh
```

In [ ]:
```
import os

# mecab-ko-dic 설치
os.chdir('/tmp')
!curl -LO https://bitbucket.org/eunjeon/mecab-ko-dic/downloads/mecab-ko-dic-2.1.1-20180720.tar.gz
!tar -zxvf mecab-ko-dic-2.1.1-20180720.tar.gz
os.chdir('/tmp/mecab-ko-dic-2.1.1-20180720')
!./autogen.sh
!./configure
!make
# !sh -c 'echo "dicdir=/usr/local/lib/mecab/dic/mecab-ko-dic" > /usr/local/etc/mecabrc'
!make install
```

In [30]:
```
!pip install preprocess
```

```
Collecting preprocess
  Using cached https://files.pythonhosted.org/packages/05/f9/559841df6c91428a2024ce120d921928
44178e4b2ceec1da84ce18205380/preprocess-1.1.0.zip
ERROR: Command errored out with exit status 1: python setup.py egg_info Check the logs for fu
ll command output.
```

- 단어 유사도 평가

In [19]:
```
# 단어 유사도 평가 데이터셋 다운로드
!wget https://github.com/dongjun-Lee/kor2vec/raw/master/test_dataset/kor_ws353.csv
```

```
--2020-10-11 15:32:17--  https://github.com/dongjun-Lee/kor2vec/raw/master/test_dataset/kor_w
s353.csv
Resolving github.com (github.com)... 140.82.121.4
Connecting to github.com (github.com)|140.82.121.4|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://raw.githubusercontent.com/dongjun-Lee/kor2vec/master/test_dataset/kor_ws353
.csv [following]
--2020-10-11 15:32:17--  https://raw.githubusercontent.com/dongjun-Lee/kor2vec/master/test_da
taset/kor_ws353.csv
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 151.101.0.133, 151.101.64.
133, 151.101.128.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|151.101.0.133|:443... con
nected.
HTTP request sent, awaiting response... 200 OK
Length: 6753 (6.6K) [text/plain]
Saving to: ‘kor_ws353.csv’

kor_ws353.csv       100%[===================>]   6.59K  --.-KB/s    in 0s

2020-10-11 15:32:18 (64.2 MB/s) - ‘kor_ws353.csv’ saved [6753/6753]
```

In [20]:
```
!-P /notebooks/embedding/data/raw
```

```
/bin/bash: - : invalid option
Usage:  /bin/bash [GNU long option] [option] ...
        /bin/bash [GNU long option] [option] script-file ...
GNU long options:
        --debug
        --debugger
        --dump-po-strings
        --dump-strings
        --help
        --init-file
        --login
        --noediting
        --noprofile
        --norc
        --posix
        --rcfile
        --restricted
        --verbose
        --version
Shell options:
        -ilrsD or -c command or -O shopt_option         (invocation only)
        -abefhkmnptuvxBCHP or -o option
```

In [ ]:
```
# 평가 클래스 코드
from models.word_eval import WordEmbeddingEvaluator

model_name = 'word2vec'

if model_name == 'word2vec':
    model = WordEmbeddingEvaluator(
        vecs_txt_fname = '/notebooks/embedding/data/word-embeddings/word2vec/word2vec',
        method = 'word2vec', dim = 100, tokenizer_name = 'mecab'
    )

elif model_name == 'fasttext':
    model = WordEmbeddingEvaluator(
        vecs_txt_fname = '/notebooks/embedding/data/word-embeddings/fasttext/fasttext.vec',
        vecs_bin_fname = '/notebooks/embedding/data/word-embeddings/fasttext/fasttext.bin',
        method = 'fasttext', dim = 100, tokenizer_name = 'mecab'
    )

elif model_name == 'glove':
    model = WordEmbeddingEvaluator(
        vecs_txt_fname = '/notebooks/embedding/data/word-embeddings/glove/glove.txt',
        method = 'glove', dim = 100, tokenizer_name = 'mecab'
    )

elif model_name == 'swivel':
    model = WordEmbeddingEvaluator(
        vecs_txt_fname = '/notebooks/embedding/data/word-embeddings/swivel/row_embedding.tsv',
        method = 'swivel', dim = 100, tokenizer_name = 'mecab'
    )

else:
    print('model name error!')
```

In [ ]:
```
# 단어 유사도 평가 수행 코드
model.word_sim_test('/notebooks/embedding/data/raw/kor_ws353.csv')
```

- 단어 유추 평가

In [ ]:
```
# 단어 유추 평가 데이터셋 다운로드
!wget https://github.com/dongjun-Lee/kor2vec/raw/master/test_dataset/kor_analogy_semantic.txt -P
/notebooks/embedding/data/raw
```

In [ ]:
```
model.word_analogy_test('/notebooks/embedding/data/raw/kor_analogy_semantic.txt')
```

- 단어 임베딩 시각화

In [ ]:
```
model = WordEmbeddingEvaluator(
        vecs_txt_fname = '/notebooks/embedding/data/word-embeddings/word2vec/word2vec',
        method = 'word2vec', dim = 100, tokenizer_name = 'mecab'
    )

model.visualize_words('/notebooks/embedding/data/raw/kor_analogy_semantic.txt')
```

In [ ]:
```
# 단어 유사도 평가 데이터셋에 포함된 모든 단어 쌍 간 코사인 유사도
model.visualize_between_words('/notebooks/embedding/data/raw/kor_analogy_semantic.txt')
```

가중 임베딩

- 단어 수준 임베딩을 문장 수준 임베딩으로 확장
- 단어 등장 확률
  - 주제 벡터 c_s가 주어졌을 때 어떤 단어 w가 나타날 확률은 다음과 같음 (Z: 확률 값으로 만들어주는 노멀라이즈 팩터)

$$P(w|c_s) = \alpha P(w) + (1-\alpha)\frac{exp(\tilde{c_s} \cdot v_w)}{Z}$$

  - 주제와 상관없이 등장할 확률과 주제와 관련을 가질 확률(주제와 단어가 유사할 경우 내적 값이 큼)의 가중합
- 문장 등장 확률
  - 문장은 곧 단어들의 시퀀스이기 때문에 단어들이 동시에 등장할 확률을 문장 등장 확률으로 보고, 문장에 속한 모든 단어들이 등장할 확률의 누적 곱으로 표현

$$P(s|c_s) \propto \sum_{w \in s} logP(w|c_s) = \sum_{w \in s} f_w(\tilde{c_s})$$

  - 누적 곱이 지나치게 작아지는 문제를 보완하기 위해 로그를 취해 덧셈을 하는 방식으로 계산
  - 테일러 근사를 한 결과는 다음과 같음

$$f_w(\tilde{c_s}) \approx f_w(\mathbf{0}) + \bigtriangledown f_w(\mathbf{0})^T \tilde{c_s} = constant + \frac{(1-\alpha)/\alpha Z}{P(w) + (1-\alpha)/\alpha Z}\tilde{c_s} \cdot \mathbf{v}_w$$

    - w가 등장할 확률을 최대화하는 주제 벡터 $c_s / \tilde{c_s}$를 찾는 것이 목표
  - 문장의 등장 확률을 최대화하는 주제 벡터는 다음과 같음

$$\arg\max_{\tilde{c_s}} f_w(\tilde{c_s}) \approx \arg\max_{\tilde{c_s}} [constant + \frac{a}{P(w)+a}\tilde{c_s} \cdot \mathbf{v}_w] = \frac{a}{P(w)+a}\frac{\mathbf{v}_w}{||\mathbf{v}_w||}$$

  - 즉, 문장이 등장할 확률을 최대화하는 주제벡터는 문장에 속한 단어들에 해당하는 단어 벡터에 가중치를 곱해 만든 새로운 벡터들의 합에 비례
  - 새로운 단어벡터의 경우 희귀한 단어(P(w)가 작음)이면 높은 가중치를 곱하고 고빈도 단어라면 벡터의 크기를 줄이는 방식으로 가중치 생성

- 모델 구현
  - CBoWModel(Continuous Bag of Words Model): 문장을 토큰으로 나눈 뒤 해당 토큰들에 대응하는 벡터들의 합으로 문장의 임베딩을 계산
    - 학습: 학습 데이터 문장을 해당 문장에 속한 토큰에 해당하는 벡터들의 가중합으로 표현하고 레이블 정보와 함께 저장
    - 예측: 테스트 문장을 토큰 벡터의 가중합으로 만들고, 코사인 유사도가 가장 높은 학습 데이터 문장의 임베딩을 찾은 후 학습 데이터의 레이블 리턴

In [ ]: