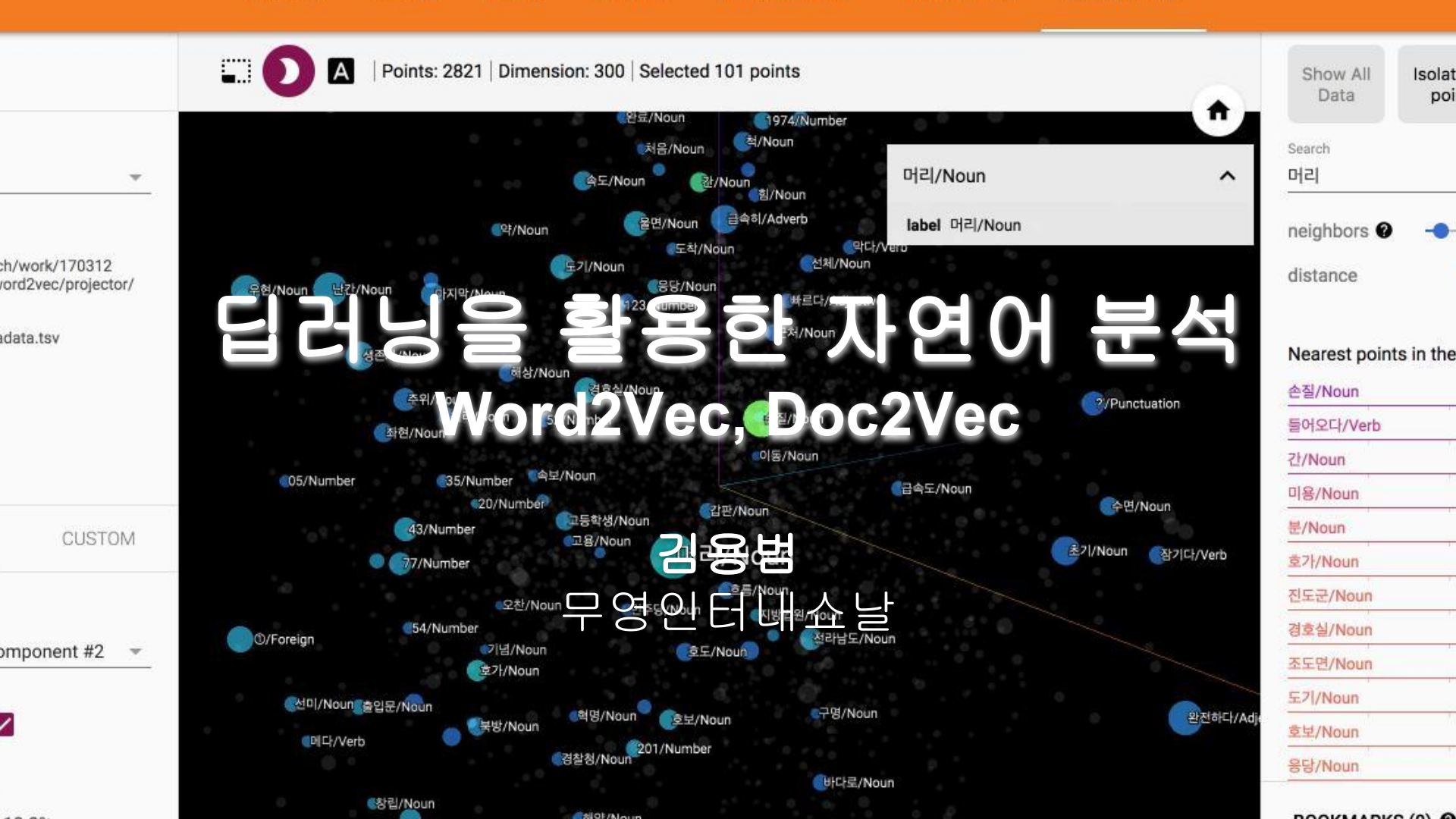
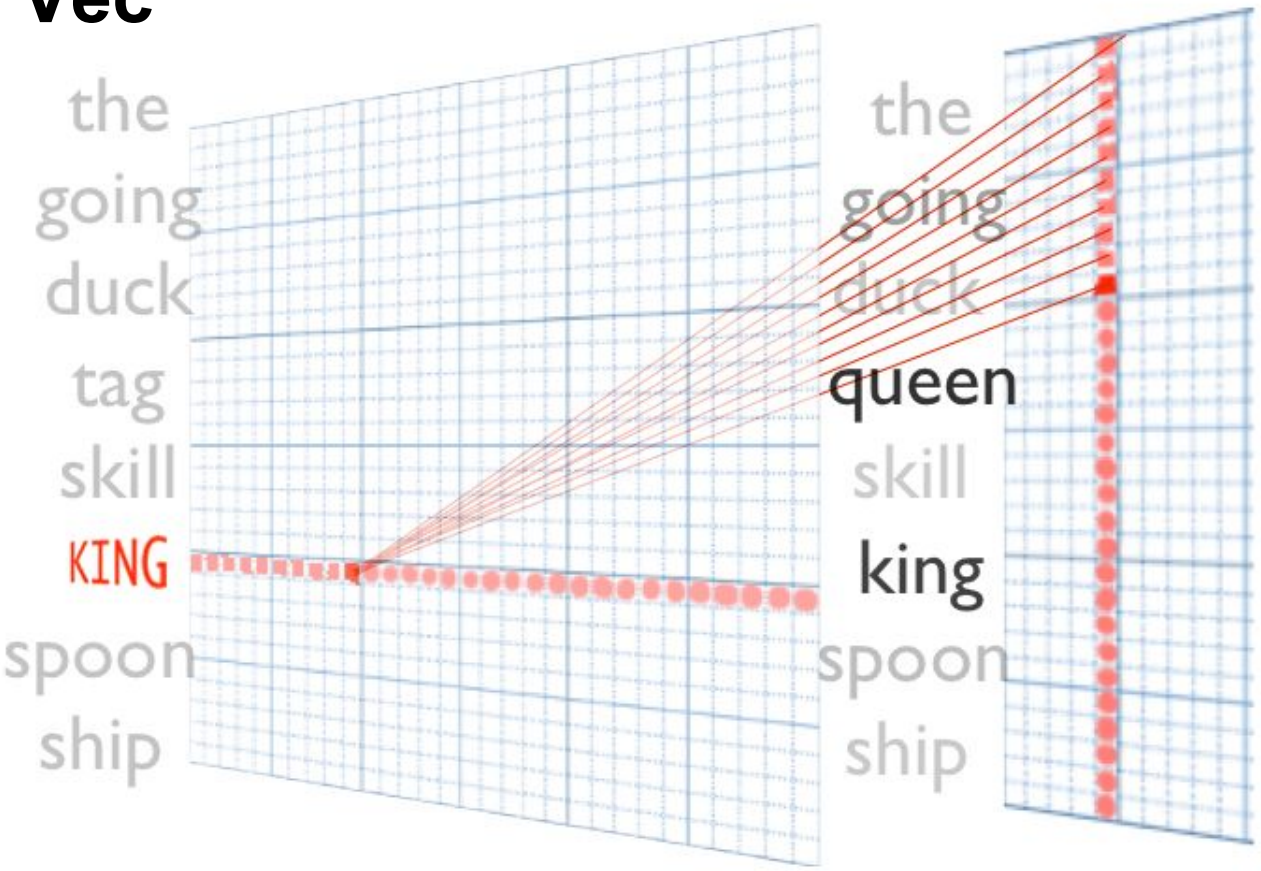

Points: 2821 | Dimension: 300 | Selected 101 points
머리/Noun
label 머리/Noun
Show All Data
Search
머리
neighbors
distance
Nearest points in the
손질/Noun
들어오다/Verb
간/Noun
미용/Noun
분/Noun
호가/Noun
진도군/Noun
경호실/Noun
조도면/Noun
도기/Noun
호보/Noun
응당/Noun
CUSTOM
딥러닝을 활용한 자연어 분석
Word2Vec, Doc2Vec
김용범
무영인터내쇼날

# Word 2 Vec

# 임베딩 - **원시 데이터**(raw data)를 학습 후 **축소된 숫자 목록**으로 변환

1. **Tf-idf** : 벡터화에 바탕을 둔 **용어빈도/ 역 문서 빈도** 를 활용
2. **One-hot Encoding** : 단어간 유사도는 알기 어렵다
3. **Word2Vec** : Mikolov가 고안한 방법으로 **"주변 단어를 보면 그 단어를 알 수 있다"** (John Firth) 에서 착안

# Word 2 Vec


the
going
duck
tag
skill
KING
spoon
ship
the
going
duck
queen
skill
king
spoon
ship

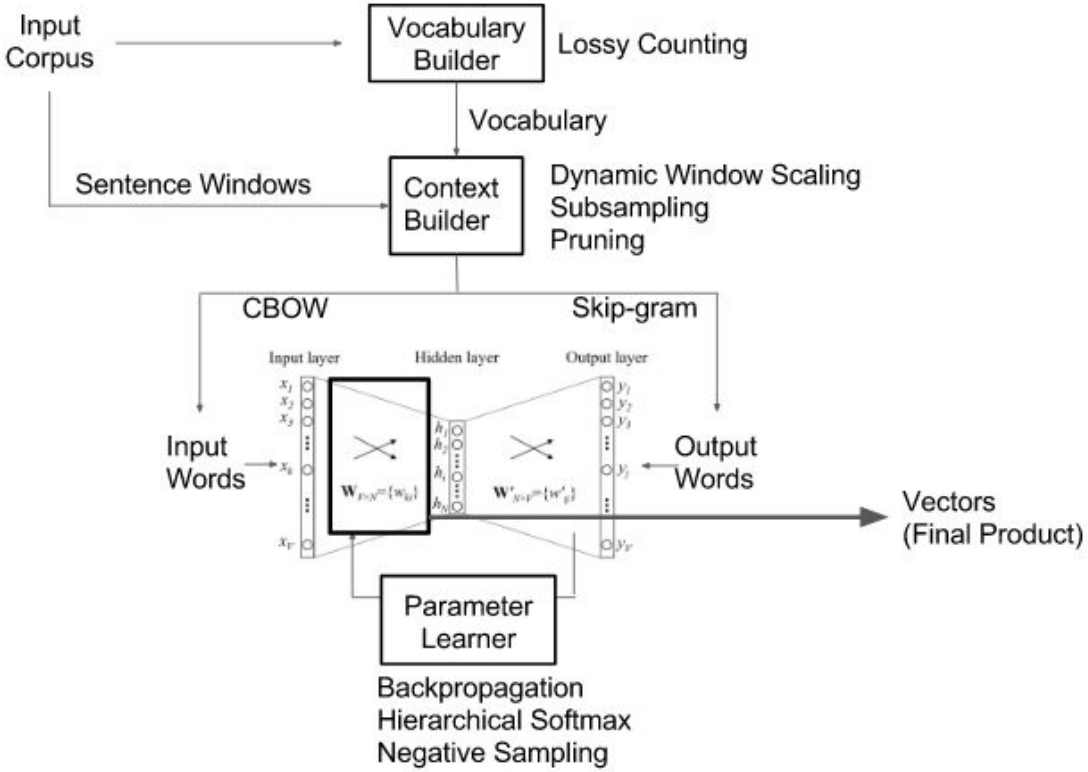
# Word 2 Vec

CBOW skip-gram

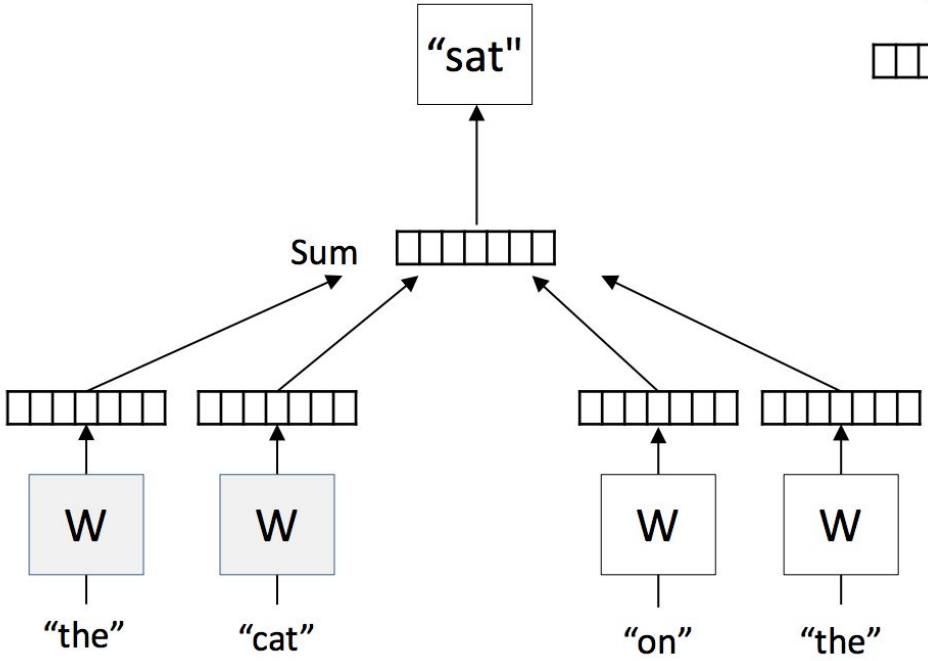
# Word 2 Vec

1. 2013년 구글에서 개발/ 공개한 기법
2. 고밀도 **단어벡터**공간에 **단어간 유사도(코싸인유사도)**를 표현
3. **(Continuous Bag of Word) - 문맥 속 어휘들**로 모델을 만들고 **Target 단어를 예측**한다
4. **(Skip Gram) - Target 단어를 중심**으로 모델을 만들고 **문맥 요소들을 예측**한다

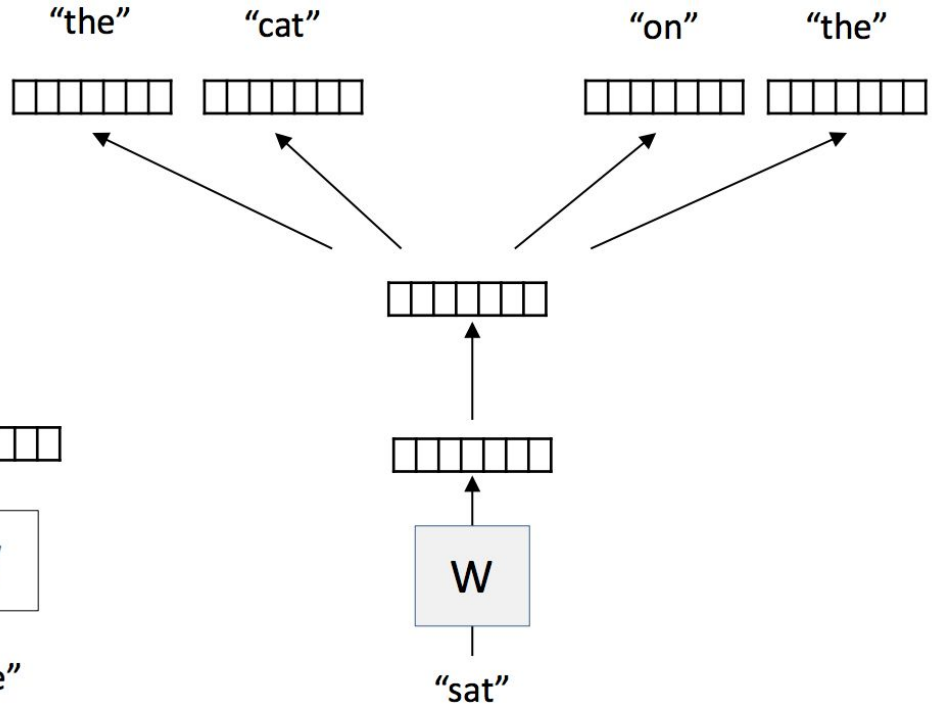
# Word 2 Vec


Input
Corpus
Vocabulary
Builder
Lossy Counting
Vocabulary
Sentence Windows
Context
Builder
Dynamic Window Scaling
Subsampling
Pruning
CBOW
Skip-gram
Input layer
Hidden layer
Output layer
Input
Words
Output
Words
Vectors
(Final Product)
Parameter
Learner
Backpropagation
Hierarchical Softmax
Negative Sampling

# Word 2 Vec




CBOW




Skip-gram (sg)

# **CBOW** - Continuous Bag-of-Words

1. 문장의 **여러 단어들** 가운데, **빈 단어를 채운다**
2. 단어 사이에 **적합한 내용을 유추하는 Network**를 생성

# Skip Gram

1. 주어진 1개의 **token** 을 갖고서 **주변 단어들을 유추**한다
2. **샘플링 기준 단어**를 몇개로 정하는지에 따라 연산이 차이 (다양한 기법이 가능)
3. **CBOW**와 비교하여 더 좋은 결과를 도출

# Word 2 Vec

gensim

# pip install --upgrade gensim

1. Why is **Gensim Word2Vec** so much **faster** than **Keras GPU**? [link]
2. **데이터**와 **모델** 을 저장하고, 호출하는 방식을 잘 익히자
3. **주요한 기능**을 메소드 함수로 제공


https://radimrehurek.com/gensim/install.html

# 데이터 불러오기

In [1]:
```
# 독일 퀘르버 재단 연설문 : 베를린 선언
f     = open('./data/베를린선언.txt', 'r')
texts = f.read()
f.close()

texts = texts.replace('\n\n', '\n')
texts = texts.replace('\n\n', '\n')
texts[:500]
```

In [2]:
```
# 텍스트를 한 줄씩 문법 tag를 추가한다
from konlpy.tag import Okt
twitter    = Okt()

token_sent = texts.split('\n')
token_sent[:5]
```

Out[2]:
```
['존경하는 독일 국민 여러분,',
 '고국에 계신 국민 여러분,',
 '하울젠 쾨르버재단 이사님과 모드로 전 동독 총리님을 비롯한 내외 귀빈 여러분,',
 '먼저, 냉전과 분단을 넘어 통일을 이루고,',
 '그 힘으로 유럽통합과 국제평화를 선도하고 있는']
```

# Twitter 한글 Tag 전처리

In [3]:
```
%%time
results_sent = []
for token in token_sent:
    twitter_token = twitter.pos(token, norm=True, stem=True)
    results = [ word[0]
                for word in twitter_token    # 어미/조사/구두점 제외
                if not word[1] in ["Eomi", "Josa", "Punctuation"] ]
    rl = (" ".join(results)).strip()
    results_sent.append(rl)
print(results_sent[:5])
```

```
['존경 하다 독일 국민 여러분', '고국 계시다 국민 여러분', '하울 젠 쾨르버 재단 이사 님 모드 전 동독 총리 님 비롯 내외 귀빈 여러분', '먼저 냉전 분단 넘다 통일 이루다', '그 힘 유럽 통합 국제 평화 선도 있다']
CPU times: user 12.9 s, sys: 240 ms, total: 13.2 s
Wall time: 4.79 s
```

In [4]:
```
texts_file = './data/Berlin.tagged'
with open(texts_file, 'w', encoding='utf-8') as file:
    file.write("\n".join(results_sent))
```

In [5]:
```
! cat ./data/Berlin.tagged | head -n 5
```

```
존경 하다 독일 국민 여러분
고국 계시다 국민 여러분
```

# Word2Vec 학습 후 모델 저장

In [7]:

```
%%time
texts_file = './data/Berlin.tagged'

from gensim.models import word2vec
data  = word2vec.LineSentence(texts_file)
model = word2vec.Word2Vec(data, size=200, window=2, hs=1, min_count=2, sg=1)
model.save("./data/Berlin.model")
print("model saved.")
```

```
model saved.
CPU times: user 2.81 s, sys: 345 ms, total: 3.16 s
Wall time: 2.9 s
```

```
from gensim.models import Word2Vec
Word2Vec(data, size=100, window = 2, min_count=50, workers=4, iter=100, sg=1)
```

1. **size = 100** : 100차원 벡터를 사용 (크면 **차원의 저주**)
2. **window = 2** : 주변 단어(window)는 앞 뒤 **두개**
3. **min_count = 50** : 출현 빈도가 50번 미만인 단어는 제외
4. **iter = 100** : 멀티코어를 활용 100번 반복 (Multi Thread)
5. **sg = 1** : CBOW, Skip-Gram 중 **Skip-Gram**를 사용

# 모델의 활용 - 모델 생성후에는 이것만 실행하면 된다

In [14]:
```
from gensim.models import word2vec
model = word2vec.Word2Vec.load('./data/Berlin.model')
```

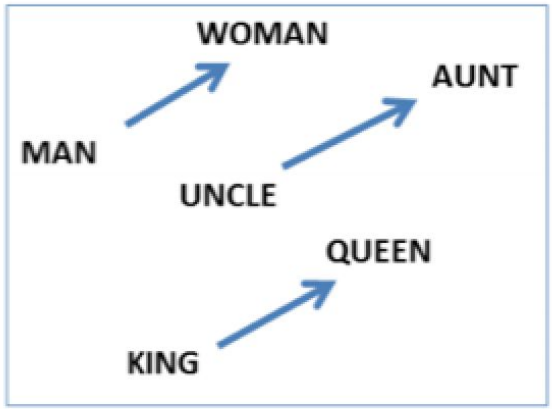
In [15]:
```
model.wv.most_similar(positive=['한반도'])
```

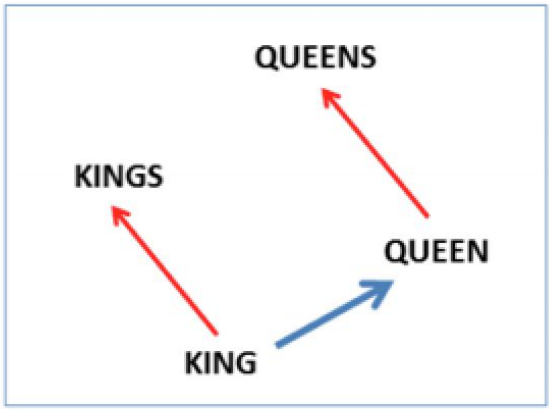
```
/home/markbaum/Python/python/lib/python3.6/site-packages/gensim/matutils.py:737: FutureWarni
ng: Conversion of the second argument of issubdtype from `int` to `np.signedinteger` is depr
ecated. In future, it will be treated as `np.int64 == np.dtype(int).type`.
  if np.issubdtype(vec.dtype, np.int):
```

Out[15]:
```
[('적', 0.9921074509620667),
 ('하다', 0.9911549091339111),
 ('있다', 0.9908924698829651),
 ('정치', 0.9889532327651978),
 ('경제', 0.9877470135688782),
 ('군사', 0.9866889715194702),
 ('이다', 0.9864642024040222),
 ('것', 0.9863808155059814),
 ('수', 0.9858399629592896),
 ('세계', 0.9850960969924927)]
```

# 단어들의 벡터 연산

1. 긍/부정 (**벡터의 방향성**) 관계망도 분석 가능하다
2. 하지만 이는 연산결과일 뿐, **구체적 내용분석**은 **분야의 전문지식**을 갖고서 별도 작업을 해야한다




(Mikolov et al., NAACL HLT, 2013)

# Word 2 Vec - 단어간의 벡터관계 활용

In [19]:
```
model.wv.most_similar(positive=['북한', '한반도'],
                      negative=['전쟁'])
```

```
/home/markbaum/Python/python/lib/python3.6/site-packages/gensim/matutils.py:737: FutureWarning: Conversion of the second argument of issubdtype from `int` to `np.signedinteger` is deprecated. In future, it will be treated as `np.int64 == np.dtype(int).type`.
  if np.issubdtype(vec.dtype, np.int):
```

Out[19]:
```
[('하다', 0.859728217124939),
 ('것', 0.8535114526748657),
 ('있다', 0.8520656824111938),
 ('이다', 0.8445901274681091),
 ('적', 0.8365031480789185),
 ('되다', 0.8355264067649841),
 ('협력', 0.818851888179779),
 ('냉전', 0.8153940439224243),
 ('들', 0.8132228851318359),
 ('정치', 0.809958279132843)]
```

# 시각화 - 2차원 데이터로 차원축소

In [18]:
```
# model.wv.vocab : { word: object of numeric vector }
vocab = list(model.wv.vocab)
X = model[vocab]
```

```
/home/markbaum/Python/python/lib/python3.6/site-packages/ipykernel_launcher.py:3: Deprecatio
nWarning: Call to deprecated `__getitem__` (Method will be removed in 4.0.0, use self.wv.__g
etitem__() instead).
  This is separate from the ipykernel package so we can avoid doing imports until
```

In [19]:
```
from sklearn.manifold import TSNE
tsne = TSNE(n_components=2)
X_tsne = tsne.fit_transform(X)
```

In [20]:
```
import pandas as pd
df = pd.DataFrame(X_tsne, index=vocab, columns=['x', 'y'])
df.head()
```

Out[20]:

| | x | y |
|---|---|---|
| 존경 | -4.606161 | 20.196222 |
| 하다 | 10.832066 | -31.455429 |
| 독일 | 8.204652 | -25.016357 |

# **TSNE** - t-distributed Stochastic Neighbor Embedding

고차원 공간에서의 **유클리디안 거리측정방법**을 활용하여

데이터 포인트의 **유사성**을 표현하는

**조건부 확률로 변환**하는 방법

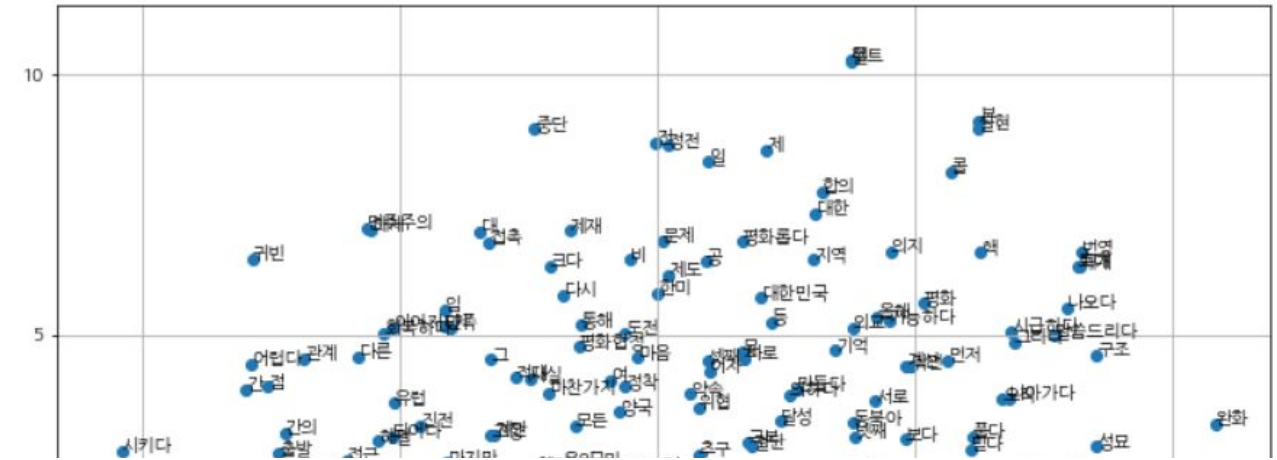
단점으로는 **조건부 확률의 기준**이 정해져 있지 않아서

생성시 마다 모양이 다르다

# **matplotlib** - 차원축소 데이터 시각화

In [15]:
```
%matplotlib inline
from matplotlib import rc
rc('font', family='NanumGothic')

import matplotlib.pyplot as plt
fig = plt.figure(figsize=(12,12))
ax  = fig.add_subplot(1, 1, 1)
ax.scatter(df['x'], df['y'])
for word, pos in df.iterrows():
    ax.annotate(word, pos)
plt.grid(True)
```



# Doc 2 Vec

# Doc 2 Vec

비지도 학습

## Doc 2 Vec

1. **Word2Vec** 는 개별 **단어 Token**의 관계를 학습
2. **Doc2Vec는 문장, 단락, 문서**와 같은 **더 큰 블록에 대한 연속표현**을 **비지도 학습**으로 모델을 생성
3. **학습 데이터의 성격이 유사**할수록 관계망이 잘 생성된다
4. **GloVe** 알고리즘(2014) / embedding 결과에 **tf/idf 가중치**를 곱한 평균을 활용방법 등 다양한 대안들이 모색

https://ratsgo.github.io/from%20frequency%20to%20semantics/2017/04/09/glove/

# Doc 2 Vec - 데이터 호출 및 전처리

In [1]:
```
from konlpy.tag import Okt
twitter = Okt()

def read_data(filename):
    with open(filename, 'r') as f:
        data = [line.split('\t') for line in f.read().splitlines()]
    from random import randint
    random_data = [data[randint(1, len(data))]  for no in range(int(len(data)/10)) ]
    return random_data

def tokenize(doc):
    return ['/'.join(t) for t in twitter.pos(doc, norm=True, stem=True)]
```

In [2]:
```
%%time
from collections import namedtuple
train_data        = read_data('data/ratings_train.txt')
train_docs        = [(tokenize(row[1]), row[2]) for row in train_data[1:]]
TaggedDocument    = namedtuple('TaggedDocument', 'words tags')
tagged_train_docs = [TaggedDocument(d, [c]) for d, c in train_docs]
```

```
CPU times: user 1min, sys: 390 ms, total: 1min 1s
Wall time: 46.7 s
```

In [3]:
```
from pprint import pprint
pprint(tagged_train_docs[0])
```

```
TaggedDocument(words=['아따/Noun', ',/Punctuation', '좋다/Adjective', ':-)/Punctuation'], tags=['1'])
```

# Doc 2 Vec - 모델 파라미터 설정 및 학습

In [26]:
```
%%time
from gensim.models import doc2vec
doc_vectorizer = doc2vec.Doc2Vec(vector_size=300, alpha=0.025, min_alpha=0.025, seed=1234)
doc_vectorizer.build_vocab(tagged_train_docs)

for epoch in range(10):
    doc_vectorizer.train(tagged_train_docs,
                         total_examples = doc_vectorizer.corpus_count,
                         epochs = doc_vectorizer.epochs)
    doc_vectorizer.alpha -= 0.002
    doc_vectorizer.min_alpha = doc_vectorizer.alpha

# 학습이 완료된 모델의 데이터를 저장한다
doc_vectorizer.save('data/doc2vec.model')
```

```
CPU times: user 53.4 s, sys: 3.87 s, total: 57.3 s
Wall time: 26.8 s
```

# Doc 2 Vec - 저장된 모델 활용하기

In [10]:
```
# 저장된 모델을 호출하여 활용한다
from gensim.models import doc2vec
from pprint import pprint
doc_vectorizer = doc2vec.Doc2Vec.load('data/doc2vec.model')
pprint(doc_vectorizer.wv.most_similar('공포/Noun'))
```

```
[('시리즈/Noun', 0.7168623805046082),
 ('에겐/Josa', 0.7058377265930176),
 ('에서/Noun', 0.67695552110672),
 ('중/Suffix', 0.6729834079742432),
 ('SF/Alpha', 0.6674826145172119),
 ('초등학교/Noun', 0.6674530506134033),
 ('메다/Verb', 0.6669254899024963),
 ('공포영화/Noun', 0.664033055305481),
 ('역대/Noun', 0.6585553288459778),
 ('칼/Noun', 0.6533156037330627)]
```

# Doc 2 Vec - 단어간 벡터연산 활용

In [38]:
```
pprint(doc_vectorizer.wv.most_similar(positive=['여자/Noun', '공포/Noun'],
                                      negative=['남자/Noun']))
```

```
[('에로영화/Noun', 0.35033077001571655),
 ('만난/Noun', 0.34775984287261963),
 ('..?/Punctuation', 0.31526681780815125),
 ('인지/Noun', 0.31278467178344727),
 ('c/Alpha', 0.30435502529144287),
 ('결과/Noun', 0.3010618984699249),
 ('장르/Noun', 0.299261212348938),
 ('이탈리아/Noun', 0.29889777302742004),
 ('밀려오다/Verb', 0.29832980036735535),
 ('B/Alpha', 0.291999876499176)]
```

# Doc 2 Vec 모델의 내적벡터 계산

단어 묶음을 활용하여 **벡터간 Cosin 유사**도 측정
Word 2 Vec 에 비해 유의미한 벡터를 찾기 힘들다.

```
In [37]: doc_vectorizer.infer_vector(['호러/Noun', '여자/Noun', '공포/Noun'])[:10]

Out[37]: array([-0.01689258,  0.00375193,  0.00620129,  0.01125604,  0.0064879 ,
                -0.00527091,  0.0069678 , -0.00344358, -0.01045213,  0.00610426],
               dtype=float32)

In [35]: doc_vectorizer.infer_vector(['공포/Noun', '연설/Noun', '역작/Noun']).sum()

Out[35]: 0.028070673
```

# Doc 2 Vec - 내적벡터 (Web)

`infer_vector(doc_words, alpha=None, min_alpha=None, epochs=None, steps=None)`

Infer a vector for given post-bulk training document.

Notes

Subsequent calls to this function may infer different representations for the same document. For a more stable representation, increase the number of steps to assert a stricket convergence.

**Parameters:**

- **doc_words** (*list of str*) – A document for which the vector representation will be inferred.
- **alpha** (*float, optional*) – The initial learning rate. If unspecified, value from model initialization will be reused.
- **min_alpha** (*float, optional*) – Learning rate will linearly drop to *min_alpha* over all inference epochs. If unspecified, value from model initialization will be reused.
- **epochs** (*int, optional*) – Number of times to train the new document. Larger values take more time, but may improve quality and run-to-run stability of inferred vectors. If unspecified, the *epochs* value from model initialization will be reused.
- **steps** (*int, optional, **deprecated***) – Previous name for *epochs*, still available for now for backward compatibility: if *epochs* is unspecified but *steps* is, the *steps* value will be used.

**Returns:** The inferred paragraph vector for the new document.

**Return** np.ndarray