In [1]:
```
%load_ext do_not_print_href
```

Word2Vec (Skipgram) in Tensorflow

참고자료

- Google Word2Vec Publications
  - "Efficient Estimation of Word Representations in Vector Space"
    - https://arxiv.org/abs/1301.3781
  - "Distributed Representations of Words and Phrases and their Compositionality", Mikolov, et. al.
    - https://arxiv.org/abs/1310.4546

- word2vec 관련 이론 정리, 김범석, 블로그 포스트
  - https://shuuki4.wordpress.com/2016/01/27/word2vec-관련-이론-정리/

- 한국어와 NLTK, Gensim의 만남, 박은정, PyCon Korea 2015
  - https://www.lucypark.kr/slides/2015-pyconkr/

- Stanford Univ CS20SI 강의 예제
  - https://github.com/chiphuyen/stanford-tensorflow-tutorials/blob/master/examples/04_word2vec_visualize.py

- 텐서플로우 튜토리얼
  - https://www.tensorflow.org/tutorials/word2vec

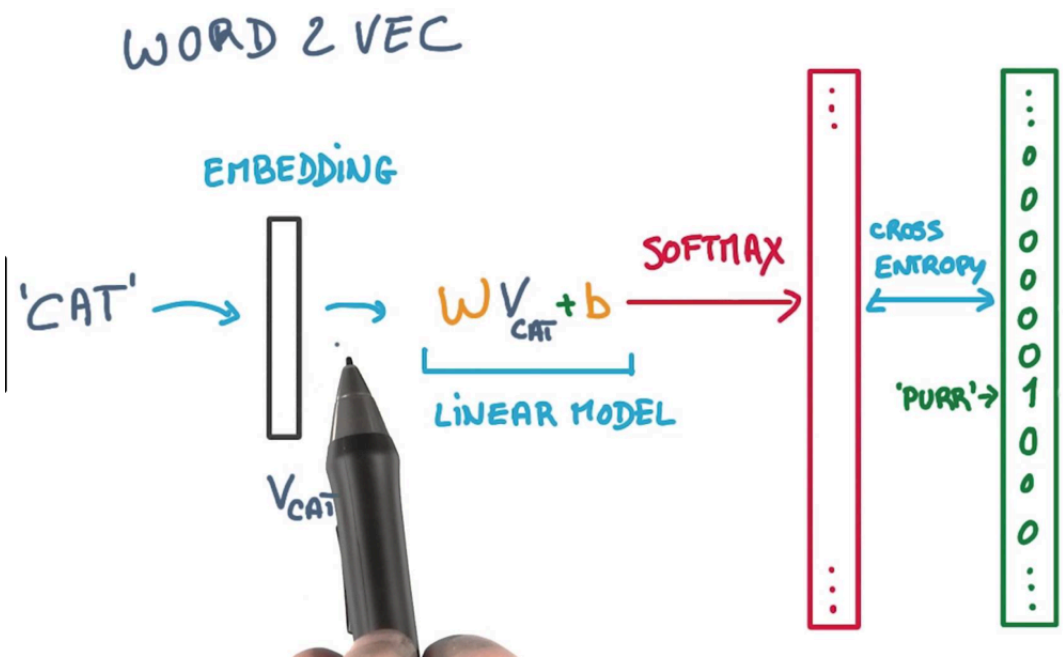
- 구글 그룹스 word2vec-toolkit
  - https://groups.google.com/forum/#!forum/word2vec-toolkit

개념 : 단어를 실수의 다차원 벡터로 표현

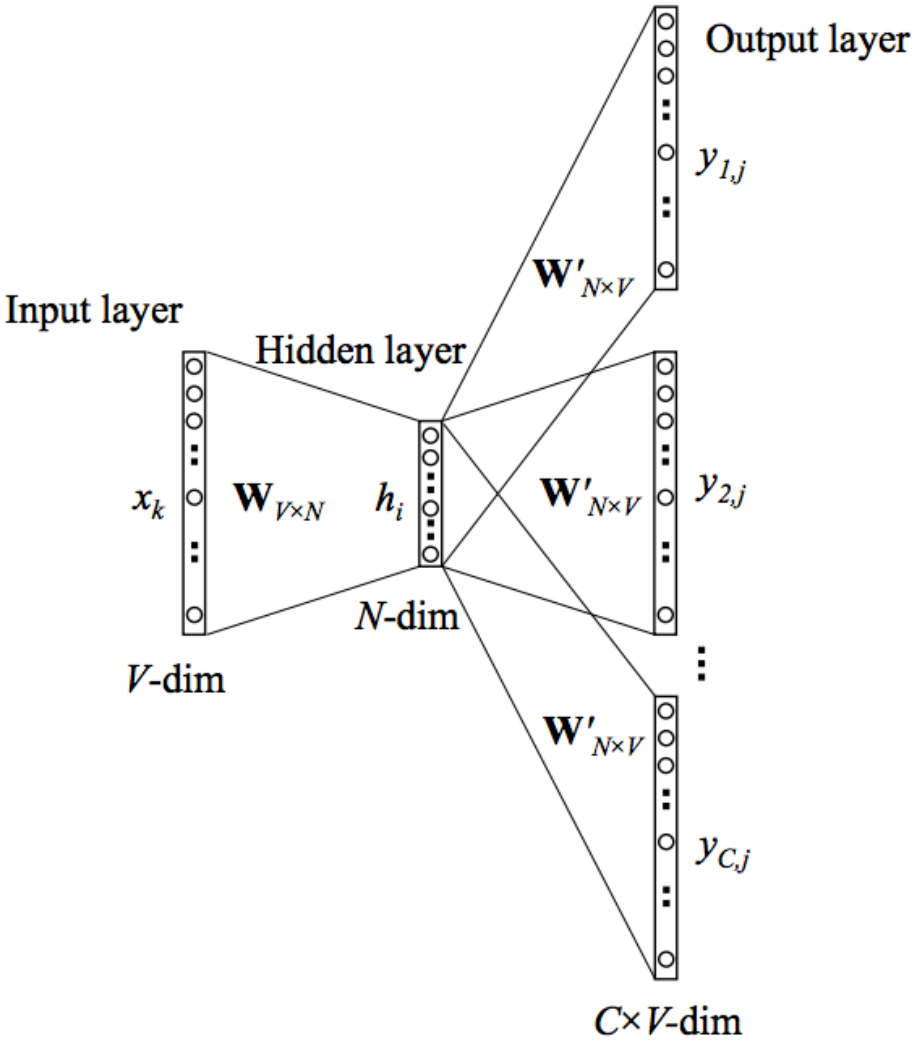
- 단어의 갯수 (V) 는 고정, 예를 들어 50000 ~ 100000 정도의 크기
- 단어를 표현하는 차원수 (N) 는 100 ~ 1000 차원

학습 : Auto-Encoder 와 유사한 구조의 모델을 이용

- input layer
  - 단어의 인덱스값의 one-hot encoding
- hidden layer
  - 단어의 임베딩을 출력 (N개 유니트)
- output layer
  - 단어의 one-hot encoding 을 출력
  - 목적하는 단어의 one-hot vector 와 cross-entropy loss 로 학습
- 원하는 결과는 hidden layer 출력값을 학습하는 것



이미지출처: https://youtu.be/BD8wPsr_DAI



이미지출처: https://www.tensorflow.org/tutorials/word2vec

input layer to hidden layer weight parameter

- $V \times N$: 5,000,000 ~ 100,000,000

```
# embed_matrix : VOCAB_SIZE x EMBED_SIZE
self.embed_matrix = tf.Variable(
    tf.random_uniform(
        [self.vocab_size,
         self.embed_size],
        -init_range,
        init_range),
    name='embed_matrix')
```

hidden to output layer weight parameter

- $N \times V$: 5,000,000 ~ 100,000,000

```
# hidden_to_output weight : EMBED_SIZE x VOCAB_SIZE
nce_weight = tf.Variable(
    tf.truncated_normal(
        [self.vocab_size, self.embed_size],
        stddev=1.0 / (self.embed_size ** 0.5)),
    name='nce_weight')
nce_bias = tf.Variable(
    tf.zeros([VOCAB_SIZE]),
    name='nce_bias')
```

임베딩에 최적화된 연산을 이용

- tf.nn.embedding_lookup()
  - 주의: `embedding_lookup()` 을 GPU 에 놓는 것은 텐서플로우에서 지원하지 않습니다.
  - `with tf.device('/cpu:0'):` 과 같은 코드를 이용해서 CPU 에서 실행되도록 할 필요가 있음
- tf.nn.nce_loss()

tf.nn.embedding_lookup()

```
embedding_lookup(
    params,
    ids,
    partition_strategy='mod',
    name=None,
    validate_indices=True,
    max_norm=None
)
```

tf.nn.nce_loss()

```
nce_loss(
    weights,
    biases,
    labels,
    inputs,
    num_sampled,
    num_classes,
    num_true=1,
    sampled_values=None,
    remove_accidental_hits=False,
    partition_strategy='mod',
    name='nce_loss'
)
```

그러면 오류 역전파 (back propagation) 는 어떻게 하나

- embedding_lookup() 연산의 back propagation 시의 gradients 는 다음 연산들에 대한 gradients 계산을 이용해서 구현되어 있다
  - tf.gather()
  - tf.sparse_segment_sum()

네트워크 모형 구성

placeholders

```
self.center_words = tf.placeholder(
    tf.int32,
    shape=[self.batch_size],
    name='center_words')
self.target_words = tf.placeholder(
    tf.int32,
    shape=[self.batch_size, 1],
    name='target_words')
```

variables

```
init_range = 0.5 / self.embed_size
self.embed_matrix = tf.Variable(
    tf.random_uniform(
        [self.vocab_size,
         self.embed_size],
        -init_range,
        init_range),
    name='embed_matrix')

nce_weight = tf.Variable(
    tf.truncated_normal(
        [self.vocab_size, self.embed_size],
        stddev=1.0 / (self.embed_size ** 0.5)),
    name='nce_weight')
nce_bias = tf.Variable(
    tf.zeros([VOCAB_SIZE]),
    name='nce_bias')
```

operations

```
embed = tf.nn.embedding_lookup(
    self.embed_matrix,
    self.center_words,
    name='embed')

self.loss = tf.reduce_mean(
    tf.nn.nce_loss(
        weights=nce_weight,
        biases=nce_bias,
        labels=self.target_words,
        inputs=embed,
        num_sampled=self.num_sampled,
        num_classes=self.vocab_size),
    name='loss')
```

사용예: word analogies (italy - rome + france)

- analogy_a = rome
- analogy_b = italy
- analogy_c = france

```
analogy_a = tf.placeholder(dtype=tf.int32)  # [N]
analogy_b = tf.placeholder(dtype=tf.int32)  # [N]
analogy_c = tf.placeholder(dtype=tf.int32)  # [N]

# n_emb = tf.nn.l2_normalize(embed_matrix, 1)

a_emb = tf.gather(embed_matrix, analogy_a)
b_emb = tf.gather(embed_matrix, analogy_b)
c_emb = tf.gather(embed_matrix, analogy_c)

target = c_emb + (b_emb - a_emb)

dist = tf.matmul(target, embed_matrix, transpose_b=True)

# For each question (row in dist), find the top 4 words.
_, pred_idx = tf.nn.top_k(dist, 4)
```

사용예: similar words

- nearby_word = rome

```
n_emb = tf.nn.l2_normalize(embed_matrix, 1)

nearby_word = tf.placeholder(dtype=tf.int32)
nearby_emb = tf.gather(n_emb, nearby_word)
nearby_dist = tf.matmul(nearby_emb, n_emb, transpose_b=True)
nearby_val, nearby_idx = tf.nn.top_k(
    nearby_dist,
    min(1000, vocab_size))
```

학습데이터의 준비

- 예시: 나무위키 백업 파일 (json 포맷)

```
$ du -sh namuwiki_170327.json
7.1G    namuwiki_170327.json
```

1단계: 텍스트로 변환

- json 원본


```
[{"namespace":"0","title":"!","text":"#redirect
\ub290\ub08c\ud45c\n","contributors":["namubot","R:hoon12560"]},
{"namespace":"0","title":"!!\uc544\uc557!!","text":"
[[\ud30c\uc77c:3444050440.jpg]]\n([[\uc2e0 \uc138\uacc4\uc218\uc758
\ubbf8\uad81 2]]\uc5d0\uc11c \ub72c !!\uc544\uc557!!)\n{{{+1 ...
```


- namuwiki_170327.json 파일은 항목들의 리스트로 구성되어 있는데, 각 항목은 파이썬 dictionary 같은 구조를 가짐
  - "namespace"
  - "title"
  - "text" : 필요한 것은 text 필드
  - "contributors"
  - etc?
- 항목 갯수는 namuwiki_170327.json 에서는 930,000 개 가량
- json 파일을 파이썬에서 읽어오기 위해서는 `json.load()` 함수 사용
  - 하지만 파일 크기가 7.1G 넘는 크기라는 점을 감안해서 메모리 관리에 유의
  - 읽어온 데이터는 dictionary 의 list 객체
  - 리스트 각 항목의 "text" 값만 읽어서 사용

2 단계: 메타 텍스트 처리

- 나무위키의 경우는 "위키문법" 이라는 것에 따른 메타표기를 사용
  - https://namu.wiki/w/%EB%82%98%EB%AC%B4%EC%9C%84%ED%82%A4:%EB%AC%B8%

- 예시:

```
[include(틀:다른 뜻1, other1=말줄임표 등으로 사용하는 용어, rd1=(...))]

 * 상위 문서: [[개그 콘서트/종영 코너]]

||<table align=right>|| http://www.mftp.info/20150901/1443910706x-576894271.jpg?width=320 ||
|||| “……” ||
|| 참여 프로그램 || [[개그 콘서트]] ||
|| 시작 || [[2013년]] [[5월 26일]] ||
|| 종료 || 2013년 [[10월 13일]] ||
|| 출연진 || [[유민상#s-1|유민상]], [[김희원]], [[송필근]], 남궁경호 ||
|| 유행어 || xx는 더 이상합니다[* 어색한 상황을 수습해보려는 변명이 더 안 좋은 결과를 낼 때 나오는 말.] ||

--[[나는 킬러다|2년 뒤에는 사위가 장인을 죽이려한다]]--

'''진짜로 코너 [[이름]]이 ([[말풍선]])“……”[* 큰 따옴표도 포함된다.](말풍선)이다'''. 언론에서는 이 코너를 언급할 때 '점점점' 또는 '점점점점점점'이라는 표현을 사용한다. 코너명의 유래는 흔히 쓰이는 [[...|말줄임표]]. 2013년 5월 26일부터 방송되었고 [[김희원]], 남궁경호, [[유민상#s-1|유민상]], [[송필근]]이 출연한다.

남자친구(남궁경호)와 여자친구(김희원)의 아버지(유민상) 간의 어색함을 다룬 코미디. 코너 초창기에는 남자친구 역인 남궁경호의 비중이 높았고 송필근은 배달부나 이웃주민 등 조연 역할이었으나, 송필근이 남궁경호의 아버지 역할(즉 유민상과 사돈관계) 컨셉으로 고정되면서 남궁경호의 비중이 줄어들고[* 심지어 아예 등장하지
```

- 참고: "namu_wiki_db_preprocess", 김인식
  - https://github.com/insikk/namu_wiki_db_preprocess
  - 정식의 위키문법 파서(?) 를 사용하지 않고 regular expression 을 이용해서 메타문자를 제거하는 코드 제공

3단계: 형태소 분석을 통한 단어 추출

- 형태소 분석을 통해서 불필요한 형태소를 제거
- 파이썬용 형태소 분석기로 konlpy.tag 모듈의 분석기를 사용할 수 있음
  - http://konlpy.org/
- 사용 예시:

```
    >>> from konlpy.tag import Kkma
    >>> from konlpy.utils import pprint
    >>> kkma = Kkma()
    >>> pprint(kkma.pos(u'오류보고는 실행환경, 에러메세지와함께 설명을 최대한상세히!^^
'))
    [(오류, NNG),
     (보고, NNG),
     (는, JX),
     (실행, NNG),
     (환경, NNG),
     (,, SP),
    ...
```

- kkma.pos() 호출 결과는 (단어스트링, 품사태그) 튜플의 목록
- 단어스트링은 어미변화등이 정규화 되어 있는 상태
- 품사태그는 단어의 형태소 유형을 서술
  - konlpy.tag 에서 제공하는 분석기들 마다 품사태그이름을 자체 정의해서 통일되어 있지 않다는 점에 유의
  - N으로 시작하는 태그 이름은 명사, V로 시작하면 동사, ... 라는 식의 대략의 규칙은 일치하는 경우가 많음
- 품사태그의 예시:
  - http://openuiz.blogspot.kr/2016/07/mecab-ko-dic.html

mecab-ko-dic 품사 태그 설명

7월 22, 2016

mecab-ko-dic 품사 태그 설명

대분류	세종 품사 태그		mecab-ko-dic 품사 태그	
	태그	설명	태그	설명
체언	NNG	일반 명사	NNG	일반 명사
	NNP	고유 명사	NNP	고유 명사
	NNB	의존 명사	NNB	의존 명사
			NNBC	단위를 나타내는 명사
	NR	수사	NR	수사
	NP	대명사	NP	대명사
	VV	동사	VV	동사
	VA	형용사	VA	형용사

- 형태소 분석 결과를 이용한 단어 추출
  - 사용 목적에 따라 달라지겠지만, 단어 관계를 유추하는 목적으로 다음과 같은 규칙을 임의 설정해서 시도

```
if  e[1].startswith(u'J') or \
    e[1].startswith(u'E') or \
    e[1].startswith(u'M') or \
    e[1].startswith(u'I') or \
    e[1].startswith(u'X') or \
    e[1].startswith(u'S') or \
    False
    continue
```

- 여기서는 두 가지 결과물이 나와야 함
  - 전체 문서에 등장하는 정규화된 단어의 빈도순 목록
    - 일정 빈도 이하로 나타나는 단어들은 생략
    - 단어 목록에 해당 단어가 없음을 나타내는 UNK 심볼 추가
    - 가장 많이 등장하는 단어가 1번이 되도록 빈도수 역순으로 인덱스 부여
  - 전체 문서 중에서 정규화된 단어들만 차례로 나열하여 다음 단계에서 사용할 입력 텍스트 파일 작성
    - 2-pass 스캔으로 단어 인덱스 시퀀스를 만들어 두면 좋음 (.npy 포맷 등으로 저장)

프로젝트 소스

- namuwiki_170327.json : 나무위키 덤프 원본
- nb-namu-db-parse.ipynb : 나무위키 텍스트 추출 스크립트
  - https://github.com/insikk/namu_wiki_db_preprocess 코드 기반으로 작성되었음
- scan_count_words.py : 형태소 처리및 동사 명사 추출 스크립트
- word2vec-demo.py : word2vec 학습/테스트 스크립트
  - stanford tensorflow tutorial 중의 04_word2vec_visualize.py 코드 기반으로 작성되었음
  - https://github.com/chiphuyen/stanford-tensorflow-tutorials/tree/master/examples

Time for Source Code Walk-through

문제점

- CS20SI 예제코드를 기반으로 해서 지나치게 단순화한 모형 사용
- 형태소 분석을 하였지만, POS 태그 정보는 활용하지 않고 단어를 정규화 하는 용도로만 사용
- 나무위키 문서의 특성상 일반 문서와 단어 분포가 다른게 아닌가 의심됨
- Tensorflow 에서 Word2Vec 에 필요한 `tf.nn.embedding_lookup()` 연산에 대해서 GPU 가속 기능을 사용할 수 없음

Word2Vec Optimization

- "Distributed Representations of Words and Phrases and their Compositionality", Mikolov, et. al.
  - https://arxiv.org/pdf/1310.4546.pdf
- 공통적으로 같이 붙여서 사용하는 단어 쌍을 찾아서 하나의 단어처럼 취급

> For example, “Boston Globe” is a newspaper, and so it is not a natural combination of the meanings of “Boston” and “Globe”

- 빈도수 높은 단어의 subsampling
- “Negative Sampling” : 기존의 nce_loss() 를 간략화(?) 한 cost function

> each training sample to update only a small percentage of the model’s weights

빈도수 높은 단어의 subsampling

- 자주 등장하는 단어의 중요성을 떨어뜨려 주는 역활
  - 영어의 "a", "the", 등
- 특정한 threshold 보다 빈도수가 높은 단어를 확률 p 로 랜덤하게 skip 하는 방법
  - $P(w_i) = 1 - \sqrt{\frac{t}{f(w_i)}}$
  - $f(w_i)$ 는 $\frac{\text{단어수카운트}}{\text{전체문서의단어수}}$
  - t 는 단어수 빈도의 threshold 값으로, 10^{-5} 정도의 값을 추천

중심단어와의 거리에 따른 weight 부여

제안

- Tensorflow 외의 방법으로 구현된 word2vec 프로그램의 사용을 추천
- Word2Vec, google
  - https://code.google.com/archive/p/word2vec/
- Gensim (Python)
  - "한국어 Word2Vec", 개인 블로그, theeluwin
    - http://blog.theeluwin.kr/post/146591096133/한국어-word2vec
- 기타 github 검색:
  - https://github.com/search?utf8=✓&q=word2vec&type=