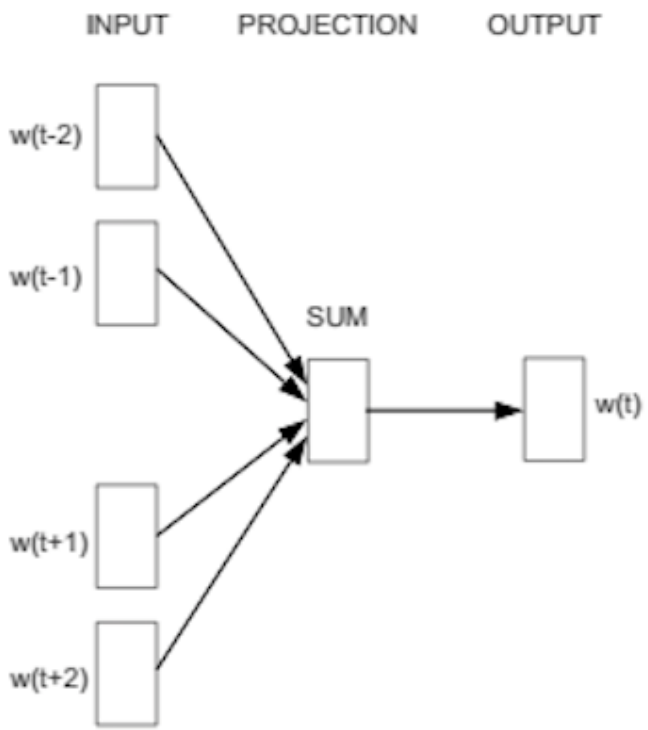

language
one
mainly
standard
extraction
specific
people
tagger
segmentation
based
morphology
disambiguation
resolution
etc
gold
parsers
compared
words
especially
considered
real-world
Note
make
entity
word
typically
named
criteria
sentences
example
written
English
information
ambiguity
generation
section
However
systems
text
input
generally
Automatic
meaning
translation
edit
often
processing
result
developed
rules
sources
corpora
type
task
intelligence
general
semantic
conducted
project
different
large
tasks
data
automatic
human
languages
semantics
vs
problem
evaluation
algorithms
machine
names
common
recognition
Given
results
refer
possible
analysis
i.e
NLP
world
although
serve
sometimes
multiple
made
door
IR
separate
process
use
complexity
speech
statistical
system
learning
larger
identify
may
procedures
including
difficult
chunk
machine-learning
quality
number
hand-written
compare
French
determine
1980s
concerned
focused
knowledge
many
also
referred
output
within
natural
set
sentence
research
Natural
however
automatically
grammar
corresponding
understanding
e.g
given
probabilistic
problems
campaign
Many
related
models
complex
usually
entities
decisions
reliable
devoted
termed
using
much
identifying
POS

Word Vector

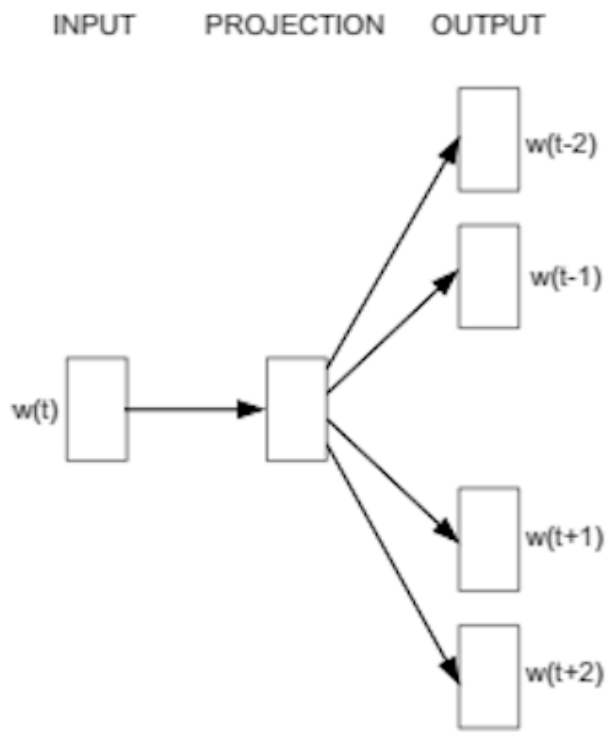
- it’s simply a vector of weights. In a simple 1-of-N (or ‘one-hot’) encoding every element in the vector is associated with a word in the vocabulary. The encoding of a given word is simply the vector in which the corresponding element is set to one, and all other elements are zero.
- Suppose our vocabulary has only five words: King, Queen, Man, Woman, and Child. We could encode the word ‘Queen’ as: [0,1,0,0,0] for [King, Queen, Man, Woman, and Child]

Word2vec

- Word2vec is a two-layer neural net that processes text. Its input is a text corpus and its output is a set of vectors: feature vectors for words in that corpus. It was introduced in 2013 by team of researchers led by Tomas Mikolov at Google.
- Word2vec is a prediction based model rather than frequency. It uses predictive analysis to make a weighted guess of a word co-occurring with respect to it’s neighbouring words. There are 2 variants to this model:



CBOW



Skip-gram

```
from gensim.models import Word2Vec
from nltk.corpus import brown, movie_reviews, treebank

b = Word2Vec(brown.sents())
mr = Word2Vec(movie_reviews.sents())
t = Word2Vec(treebank.sents())

b.most_similar('money', topn=5)
```

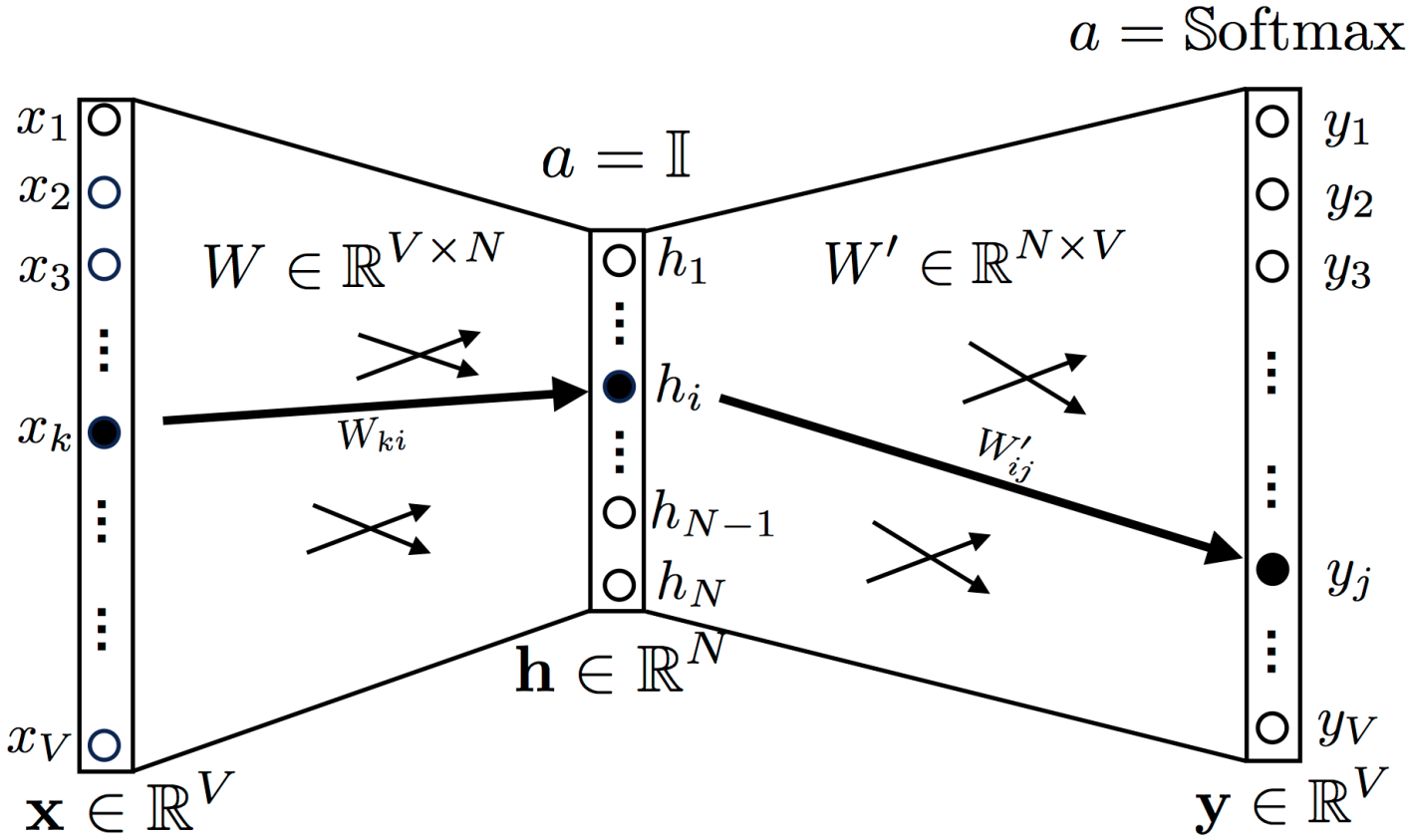
```
Outpt:
[('job', 0.9242240786552429),
('care', 0.9002009630203247),
('work', 0.8949295282363892),
('trouble', 0.8837088346481323),
('faith', 0.8803966641426086)]
```

```
from __future__ import print_function, division
from future.utils import iteritems
from builtins import range

from gensim.models import KeyedVectors

word_vectors = KeyedVectors.load_word2vec_format(
  'GoogleNews-vectors-negative300.bin',
  binary=True
)

def find_analogies(words):
  r = word_vectors.most_similar(positive=[words[0], words[2]], negative=[words[1]])
  print("%s - %s = %s - %s" % (words[0], words[1], r[0][0], words[2]))

def nearest_neighbors(w):
  r = word_vectors.most_similar(positive=[w])
  print("neighbors of: %s" % w)
  for word, score in r:
    print("\t%s" % word)


def read_dataset(fname):
    file = open(fname,"r",encoding="utf8")
    for line in file:
        line = line.strip("\n").lower()
        words = line.split()
#         print(words)
        find_analogies(words[:3])
    file.close()

read_dataset("analogy_small.txt")
```

CBOW (Continuous Bag of Words)

This model tries to predict a word on bases of it's neighbours.

The input to the model could be $w_{i-2}, w_{i-1}, w_{i+1}, w_{i+2}$, the preceding and following words of the current word we are at. The output of the neural network will be w_i. Hence you can think of the task as "predicting the word given its context"



$$h = W^T x$$

$$u = W'^T h = W'^T W^T x$$

$$y = softmax(u)$$

Loss Function

We are training the model against the target-context word pair (w_t, w_c). The target word represents the ideal prediction from our neural network model, given the context word w_c. It is represented by a one-hot encoded vector. Say that it has value 1 at the position j^* (and the value 0 for any other position).

The loss function will need to evaluate the output layer at the position j^*. Since the values in the softmax can be interpreted as conditional probabilities of the target word, given the context word w_c, we write the loss function as:

$$L = -logP(w_t|w_c) = -\log y_j = -log[Softmax(u_j)] = -log(\frac{\exp u_j}{\sum_i \exp u_i})$$

$$L = -u_j + \log \sum_i \exp(u_i)$$

- **Example** Let's go back to our previous example of the sentence "I like playing football". Suppose we are training the model against the first training data point: the context word is "I" and the target word is "like".

$$W = \begin{pmatrix} -1.38118728 & 0.54849373 \\ 0.39389902 & -1.1501331 \\ -1.16967628 & 0.36078022 \\ 0.06676289 & -0.14292845 \end{pmatrix}$$

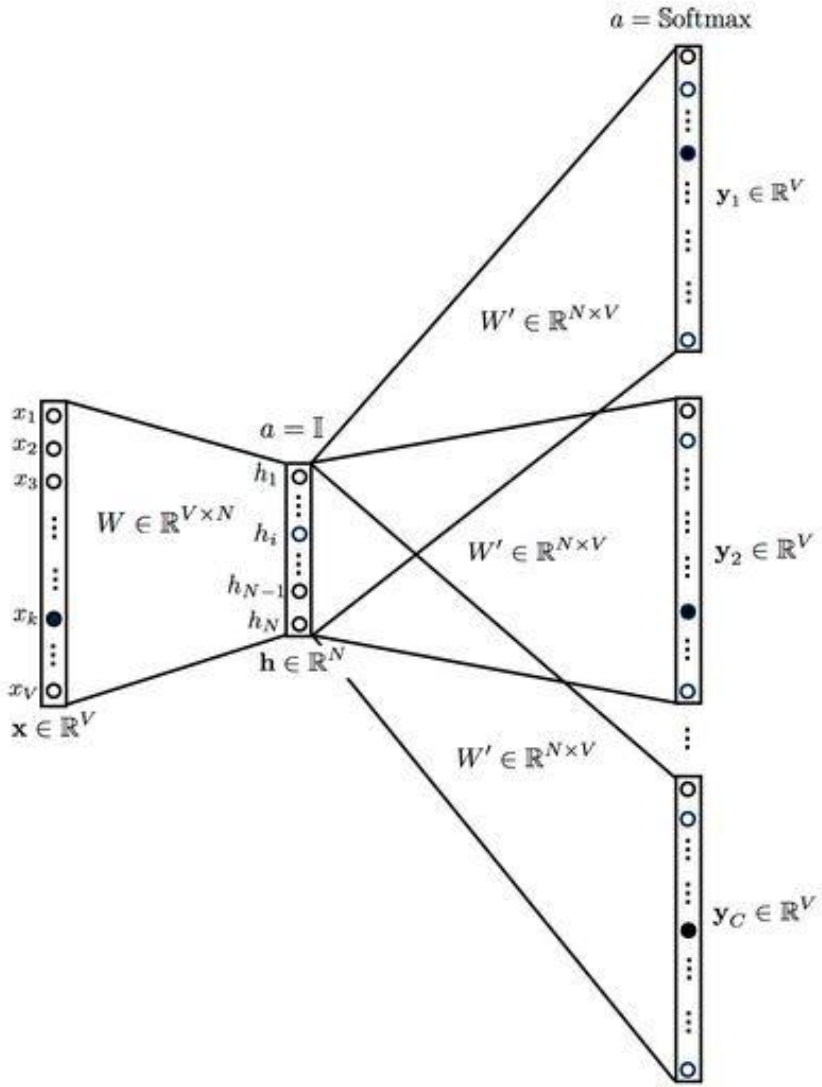
$$W' = \begin{pmatrix} 1.39420129 & -0.89441757 & 0.99869667 & 0.44447037 \\ 0.69671796 & -0.23364341 & 0.21975196 & -0.0022673 \end{pmatrix}$$

SkipGram

This models tries to predict the neighbours of a word.

The input to the model is w_i, and the output could be $w_{i-2}, w_{i-1}, w_{i+1}, w_{i+2}$. So the task here is "predicting the context given a word". The Skip-gram model is essentially the inverse of the CBOW model. The input is a center word and the output predicts the context words, given the center word. The resulting neural network looks like in the figure below,

Since SkipGram had been shown to perform better on analogy-related tasks than the CBOW model, it's better to use SkipGram model for the most part.



$$h = W^T x$$

$$u_c = W'^T h = W'^T W^T x \quad c = 1, \cdots, C$$

$$y_c = softmax(u) \quad c = 1, \cdots, C$$

Loss Function:

$$L = -\log P(w_{c,1}, w_{c,2}, \ldots, w_{c,C} | w_o) = -\log \prod_{c=1}^{C} P(w_{c,i} | w_o)$$

$$= -\log \prod_{c=1}^{C} \frac{\exp(u_{c,j^*})}{\sum_{j=1}^{V} \exp(u_{c,j})} = -\sum_{c=1}^{C} u_{c,j^*} + \sum_{c=1}^{C} \log \sum_{j=1}^{V} \exp(u_{c,j})$$

Note You can find detail explanation at this link : http://www.1-4-5.net/~dmm/ml/how_does_word2vec_work.pdf

Word Vector Evaluation

Analogy

man is to king is like woman is to ___?(queen).

Cosine similarity: we have the vectors. The first question:
Which word is closest in the vector space to a given word? How do we compute the distance between two word vectors a, b?

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|_2 \|\mathbf{B}\|_2} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}}$$

Note see Ben Schmidt's blog for some great examples : http://bookworm.benschmidt.org/posts/2015-10-25-Word-Embeddings.html