Text Mining

Week 5

Word2Vec, embeddings

- Resources:
  - Stanford CS224d: Deep Learning for NLP (Manning and Socher)
    - http://cs224d.stanford.edu/index.html
  - “word2vec Parameter Learning Explained”, Xin Rong
    - https://ronxin.github.io/wevi/
  - Word2Vec Tutorial - The Skip-Gram Model
    - http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/
  - https://github.com/tmikolov/word2vec
  - Softmax Regression Tutorial

Context

- Context-*assisted* techniques to Context-*centric* techniques
- Traditional context-*assisted*
  - Word Sense Disambiguation
  - Synonym detection
  - Relations extraction
  - Speech

- Why do we need context $\rightarrow$ meaning

How can we encode word meaning?

- Use a taxonomy like WordNet that has hypernyms (is - a) relationships as well as synonym sets (synsets)
- WordNet:
  - Great as a resource, but
    - Missing new words
    - Subjective
    - Requires human labor to create and adapt
    - Hard to computer accurate word similarity

One-hot vector representation

- In vector space terms, this is a vector with one 1 and a lot of zeroes:

  [0 0 0 0 0 0 0 1 0 0 0 0 0 0]

- Dimensionality:
  - 20K(speech)
  - 50K(PennTreeBank)
  - 500K(big vocab)
  - 13M(Google 1T)
- One-hot – localist representation

Distributional similarity based representation

- Distributional → the word meaning is distributed over a vector
- “You shall know a word by the company it keeps” (J.R.Firth, 1957:11)
- word2vec Approach to represent the meaning of word
    - Represent each word with a low-dimensional vector
    - Word similarity = vector similarity
    - Key idea: Predict surrounding words of every word
    - Fast and can easily incorporate a new sentence/document or add a word to the vocabulary

The Power of Word Vectors

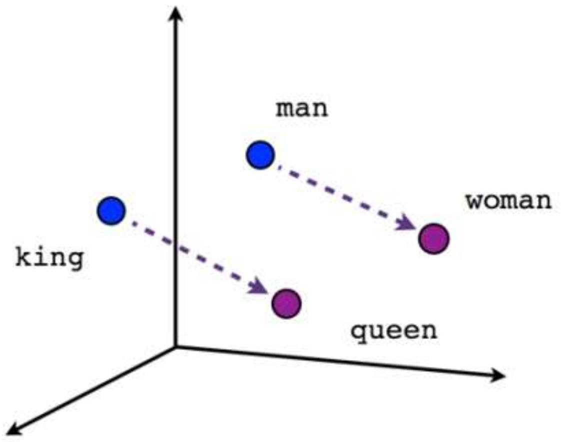
- They provide a fresh perspective to ***ALL*** problems in NLP, and not just solve one problem.

- Technological Improvement
  - Rise of deep learning since 2006 (Big Data + GPUs + Work done by Andrew Ng, Yoshua Bengio, Yann Lecun and Geoff Hinton)

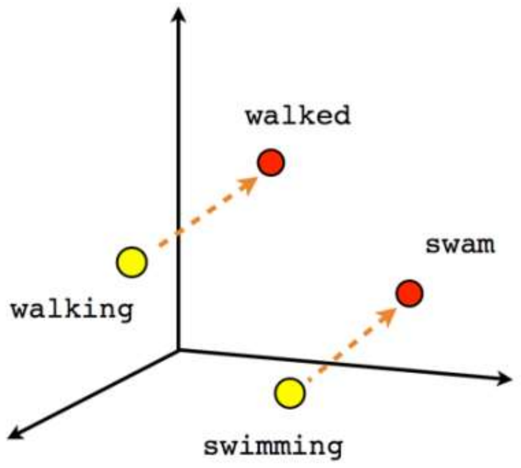
  - Application of Deep Learning to NLP – led by Yoshua Bengio, Christopher Manning, Richard Socher, Tomas Mikalov

- The need for unsupervised learning . (Supervised learning tends to be excessively dependant on hand-labelled data and often does not scale)

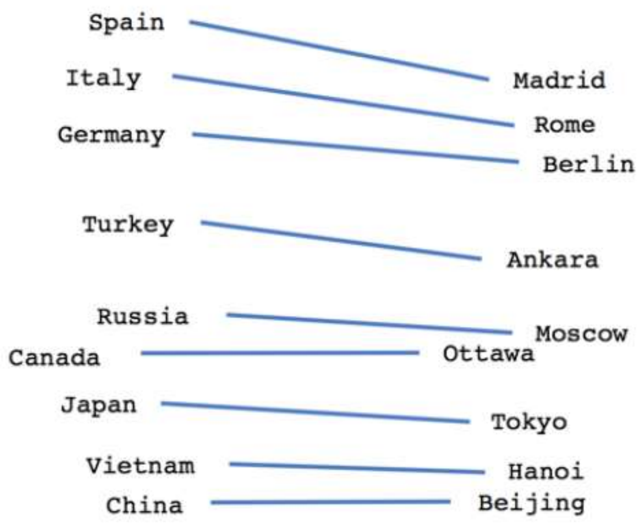
Examples



Male-Female



Verb tense



Country-Capital

vector[Queen] = vector[King] - vector[Man] + vector[Woman]

So, how exactly does Word Embedding ‘solve all problems in NLP’?

Building these magical vectors . . .

- How do we actually build these super-intelligent vectors, that seem to have such magical powers?

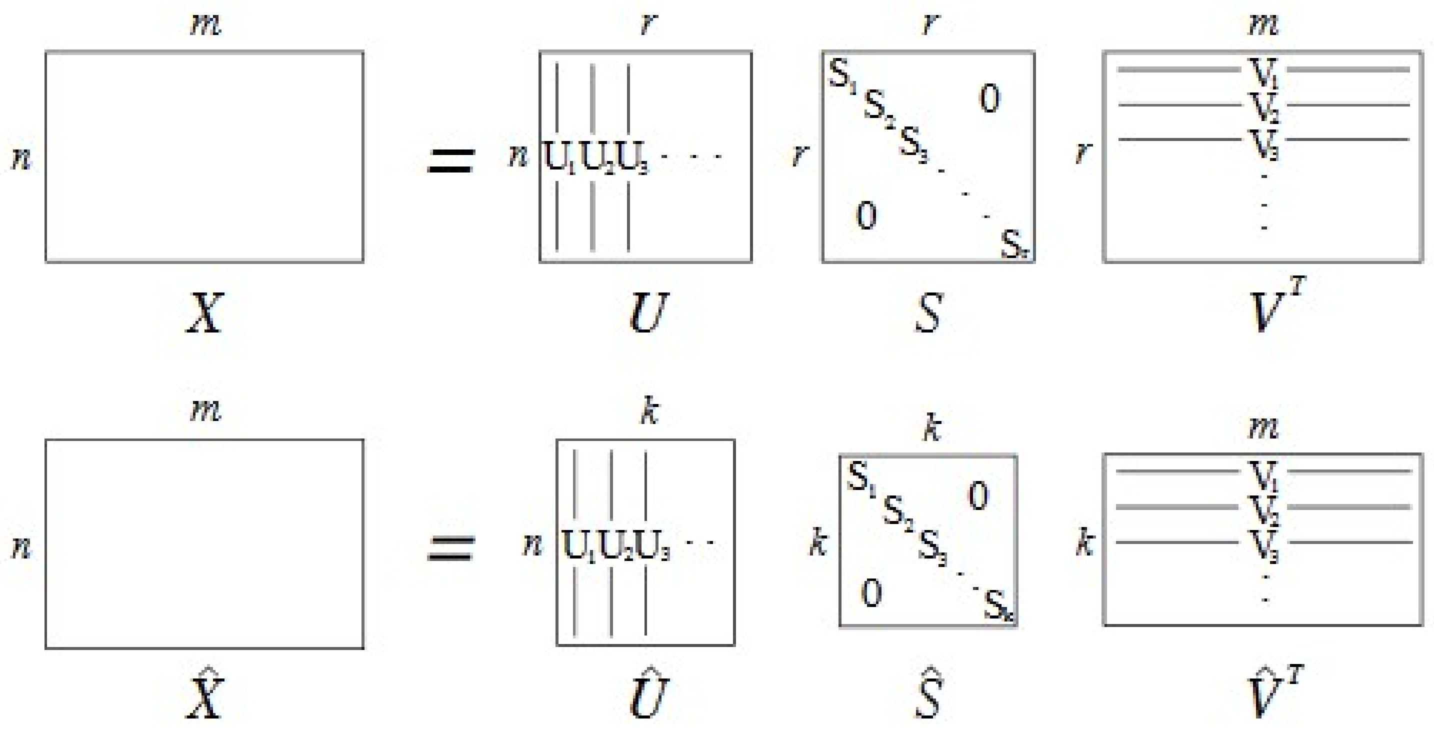
- How to find a word’s friends?

- We will discuss the most famous methods to build such lower-dimension vector representations for words based on their context
    1. Co-occurrence Matrix with SVD
    2. word2vec (*Google)*
    3. Global Vector Representations (GloVe) (*Stanford)*

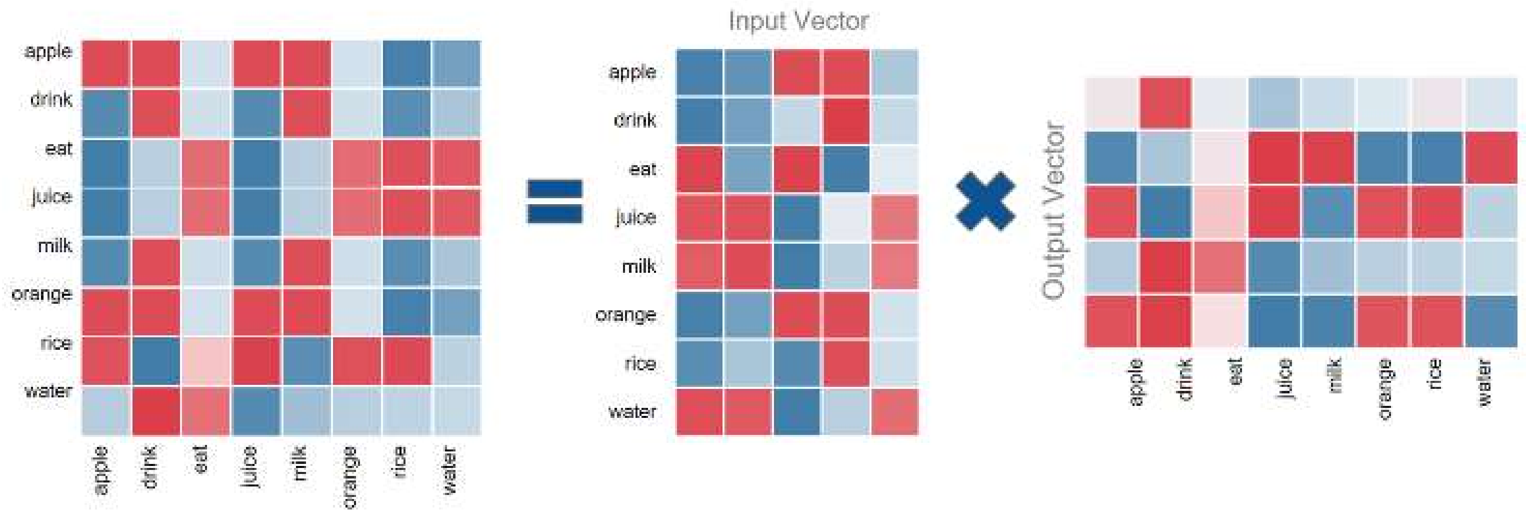
Word Representations

Traditional Method - Bag of Words Model	Word Embeddings
• Uses one **“hot encoding”** • Each word in the vocabulary is represented by one bit position in a HUGE vector. • For example, if we have a vocabulary of 10000 words, and “Hello” is the 4th word in the dictionary, it would be represented by: 0 0 0 1 0 0 . . . . 0 0 0 0 • Context information is not utilized	• Stores each word in as a point in space, where it is represented by a vector of fixed number of dimensions (generally 300) • Unsupervised, built just by reading huge corpus • For example, “Hello” might be represented as : [0.4, -0.11, 0.55, 0.3 . . . 0.1, 0.02] • Dimensions are basically projections along different axes, more of a mathematical concept.

Singular Value Decomposition



Singular Value Decomposition



The problem with this method, is that we may end up with matrices having billions of rows and columns, which makes SVD computationally restrictive.

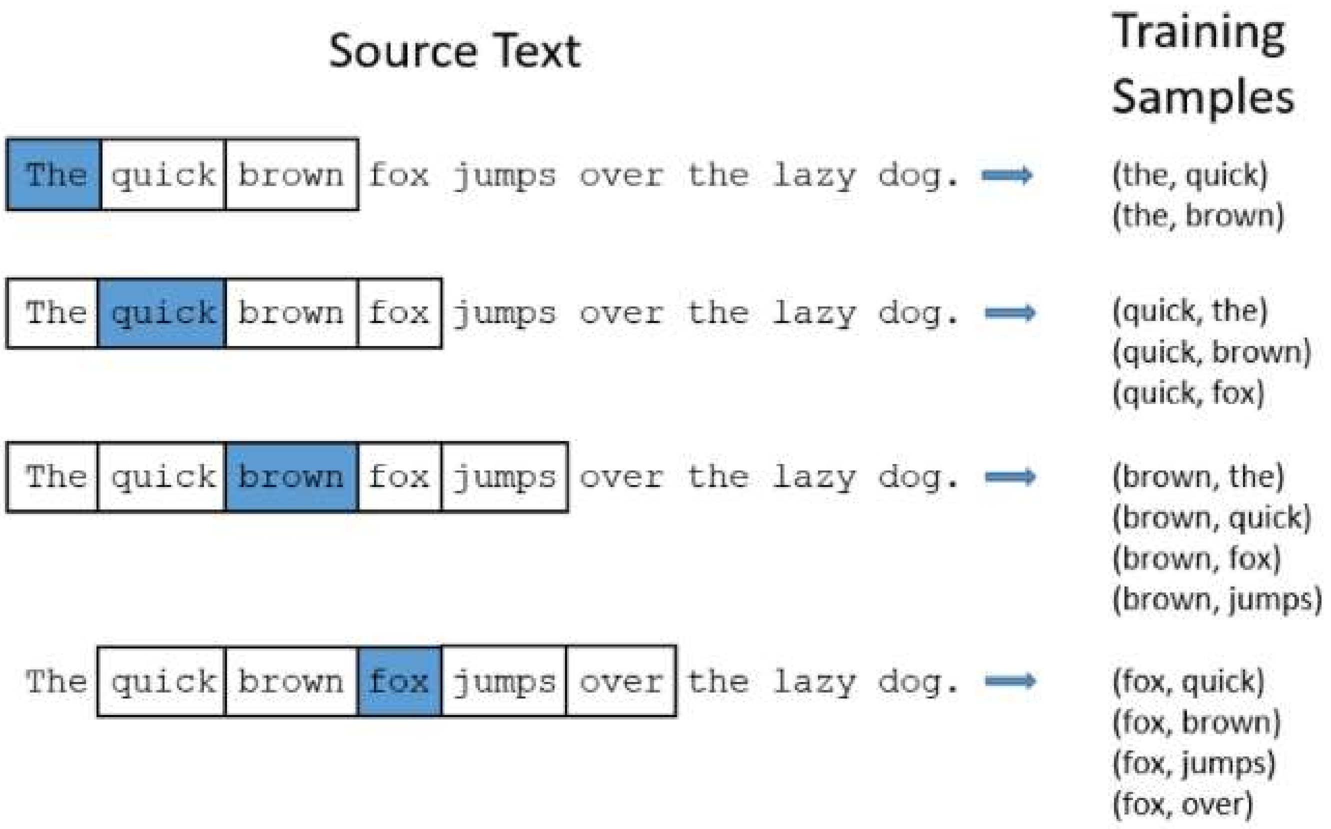
Word2Vec

- train a simple neural network with a single hidden layer to perform a certain task (word prediction)
- but then we're not actually going to use that neural network for the task we trained it on!
- Instead, the goal is actually just to **learn the weights** of the hidden layer–we'll see that these weights are actually the "word vectors" that we're trying to learn.

Fake Task

- We’re going to train the neural network to do the following.
  - Given a specific word in the middle of a sentence (the input word), look at the words nearby and pick one at random.
  - The network is going to tell us the probability for every word in our vocabulary of being the “nearby word” that we chose.
  - For "nearby", there is actually a "window size" parameter to the algorithm. A typical window size might be 5, meaning 5 words behind and 5 words ahead (10 in total).

- Order does not matter
- The network is going to learn the statistics from the number of times each pairing shows up.



Context windows

- Context can be anything – a surrounding n-gram, a randomly sampled set of words from a fixed size window around the word

For example, assume context is defined as the word following a word.

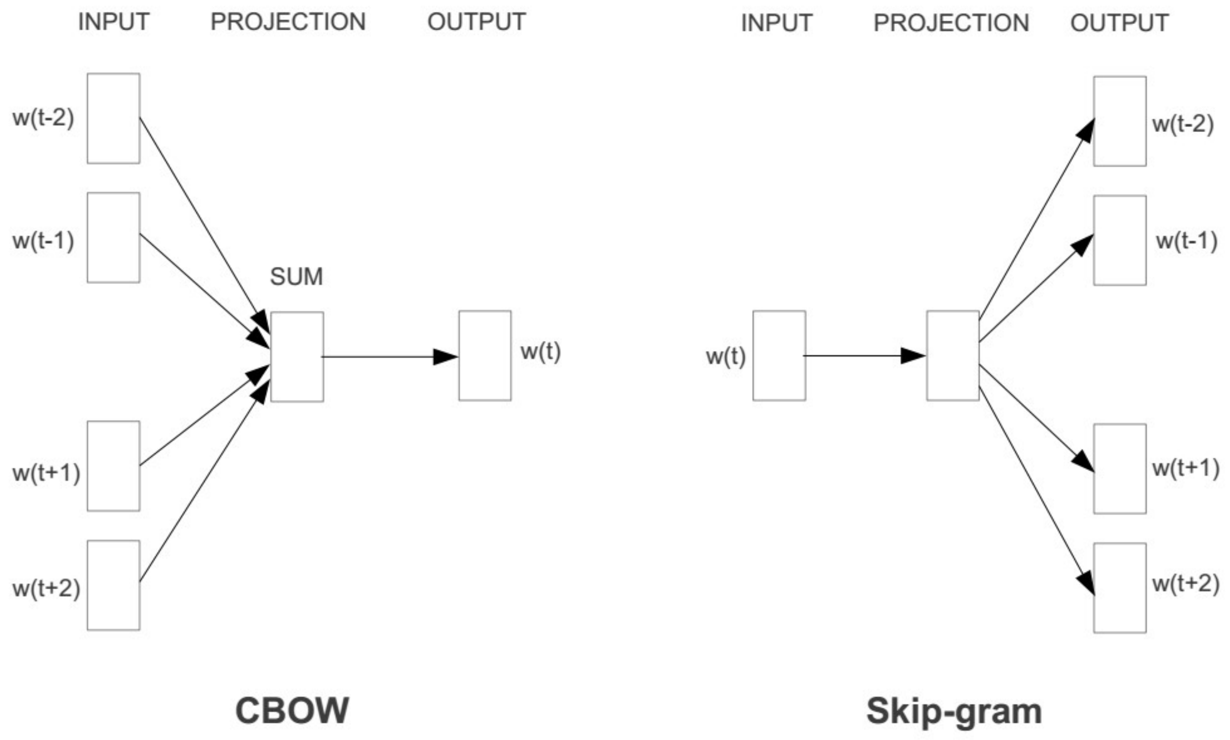
i.e. $\boldsymbol{context(w_i) = w_{i+1}}$

Corpus : I ate the cat

Training Set : I|ate, ate|the , the|cat, cat|.

Represent the meaning of word – word2vec

- 2 basic neural network models:
    - Continuous Bag of Word (CBOW): use a window of word to predict the middle word
    - Skip-gram (SG): use a word to predict the surrounding ones in window.

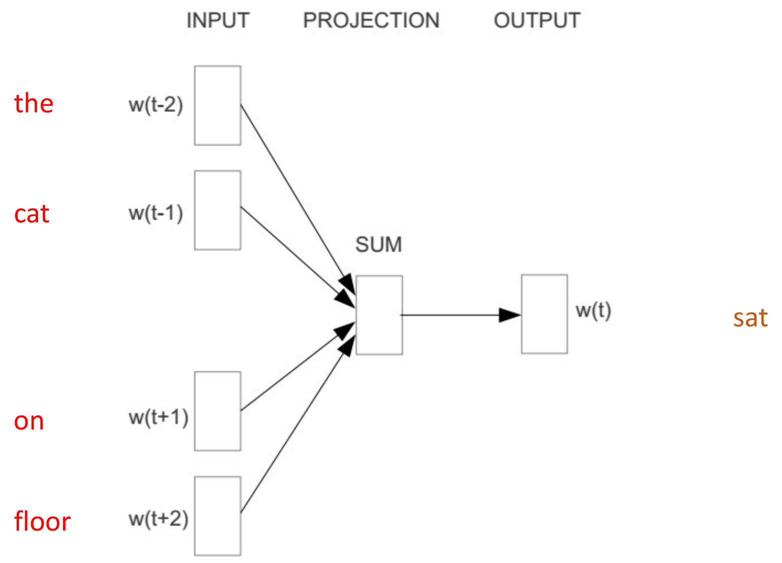


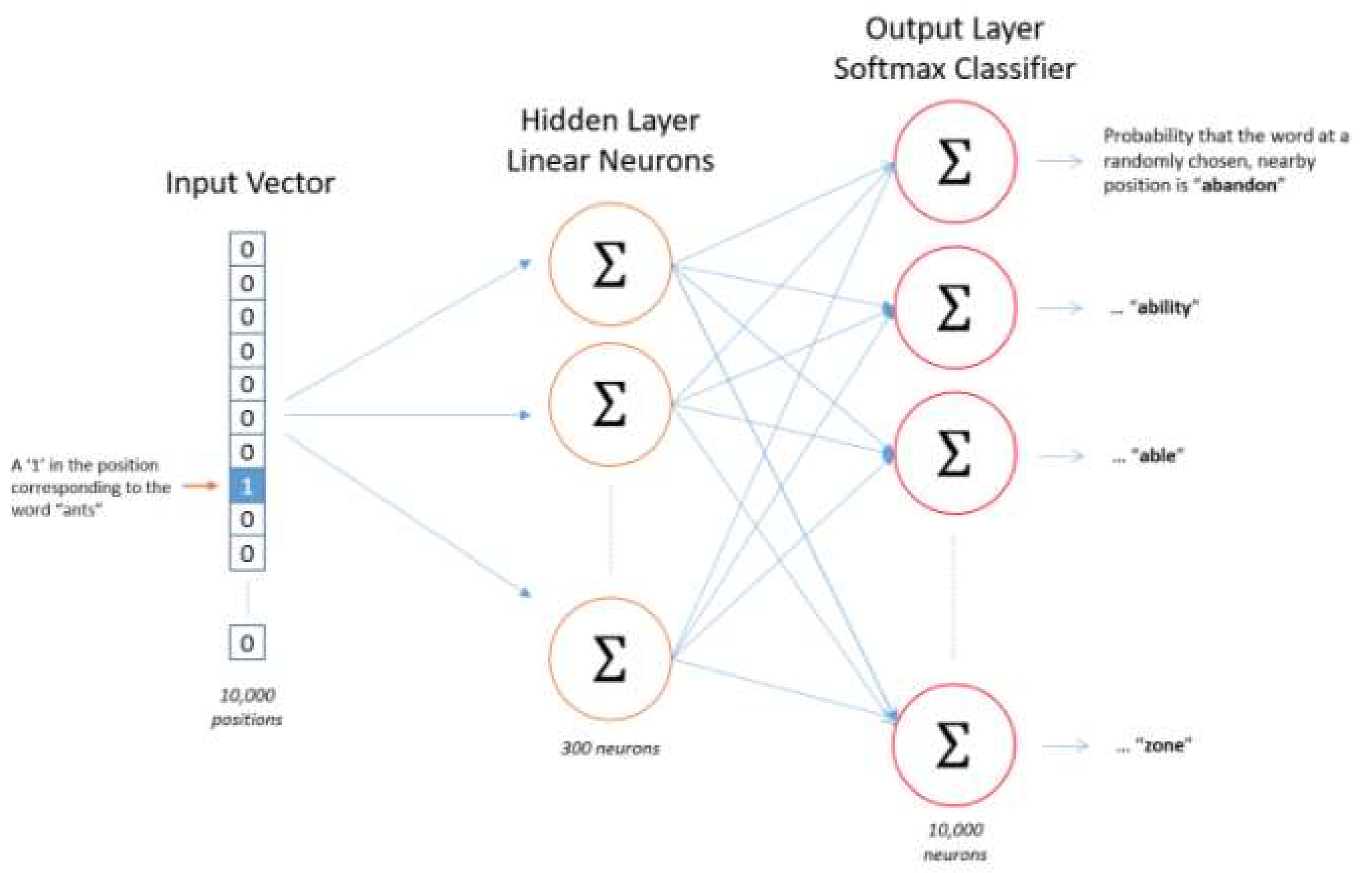
Model Details

- First, build vocabulary (let’s say 10,000 unique words)
- One-hot vector: 1 element of the 10,000-element vector is 1, the remaining 9,999 elements are 0s.

Word2vec – Continuous Bag of Word

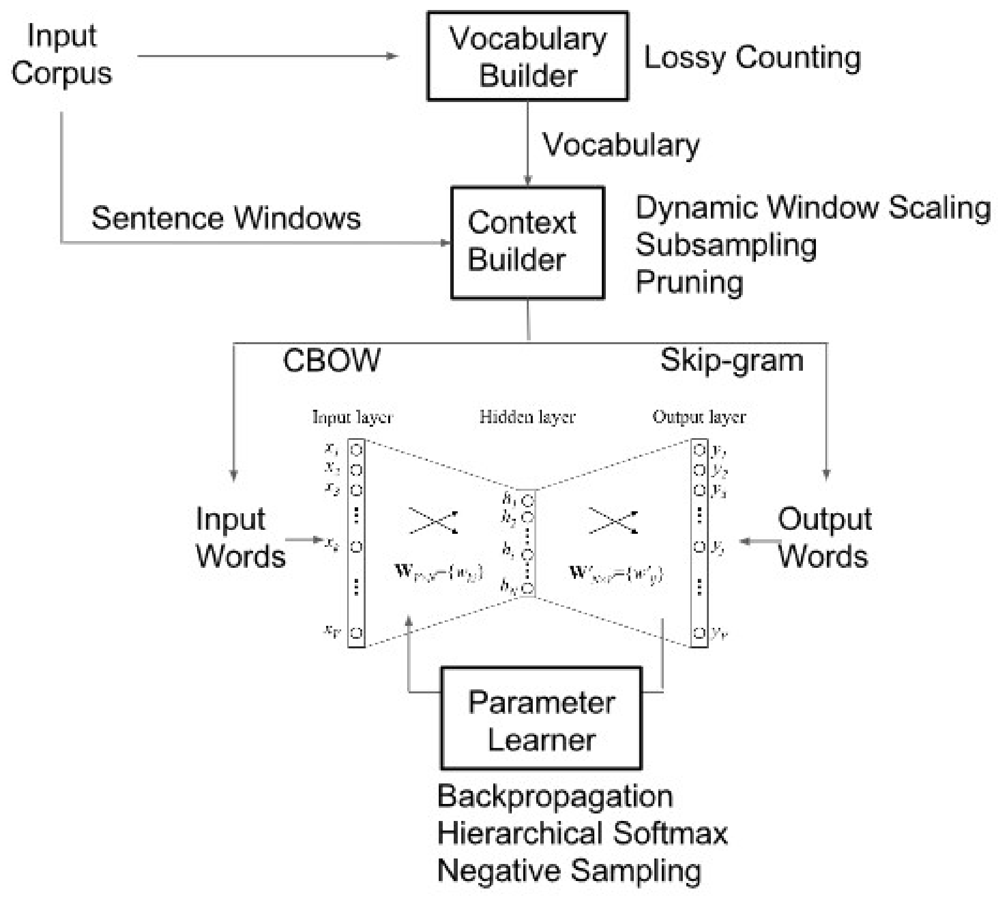
- E.g. “The cat sat on floor”
    - Window size = 2





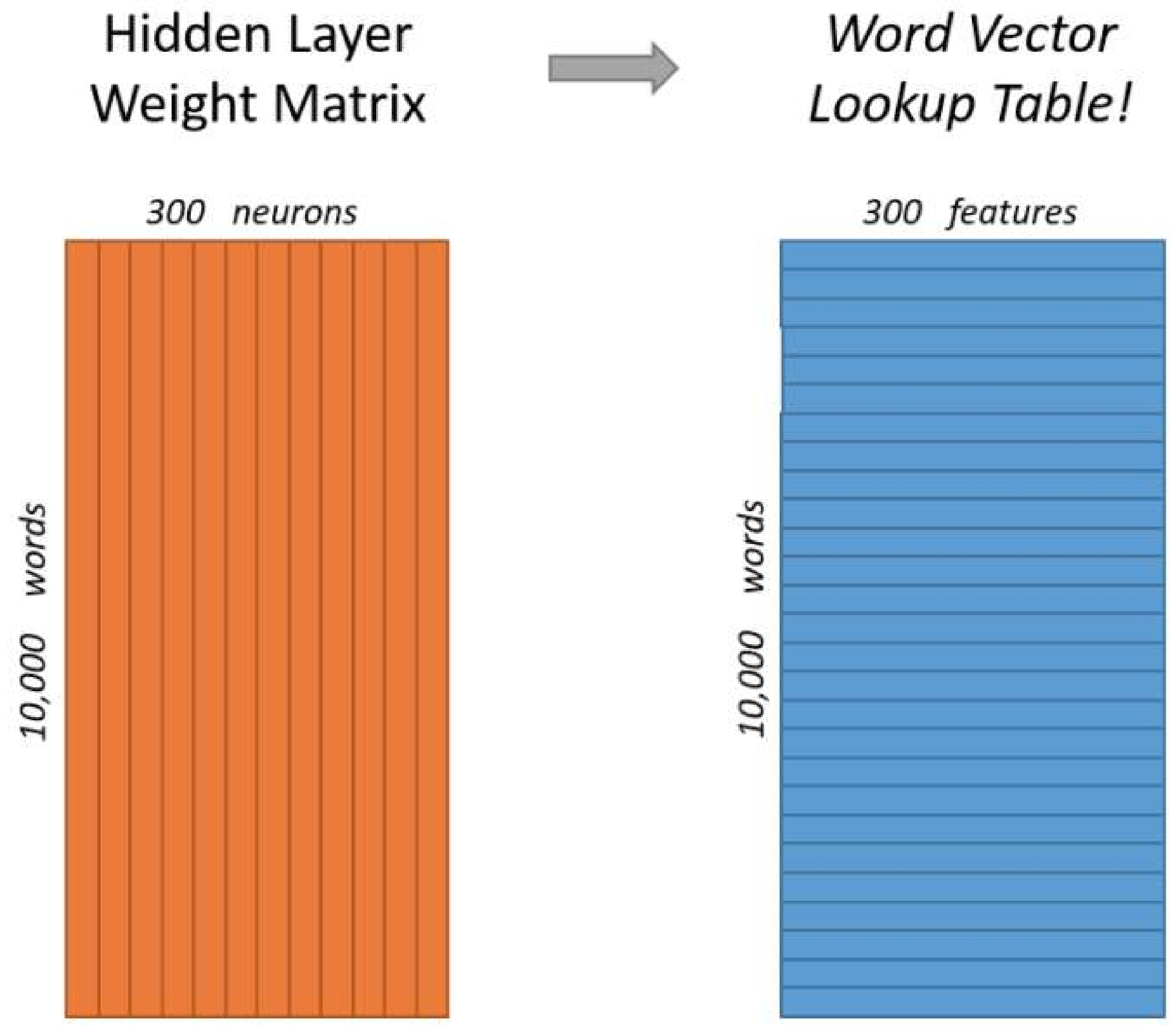
- There is no activation function on the hidden layer neurons, but the output neurons use softmax.
- When *training* this network on word pairs, the input is a one-hot vector representing the input word and the training output *is also a one-hot vector* representing the output word.
- But when you evaluate the trained network on an input word, the output vector will actually be a probability distribution (i.e., a bunch of floating point values, *not* a one-hot vector).

Architecture



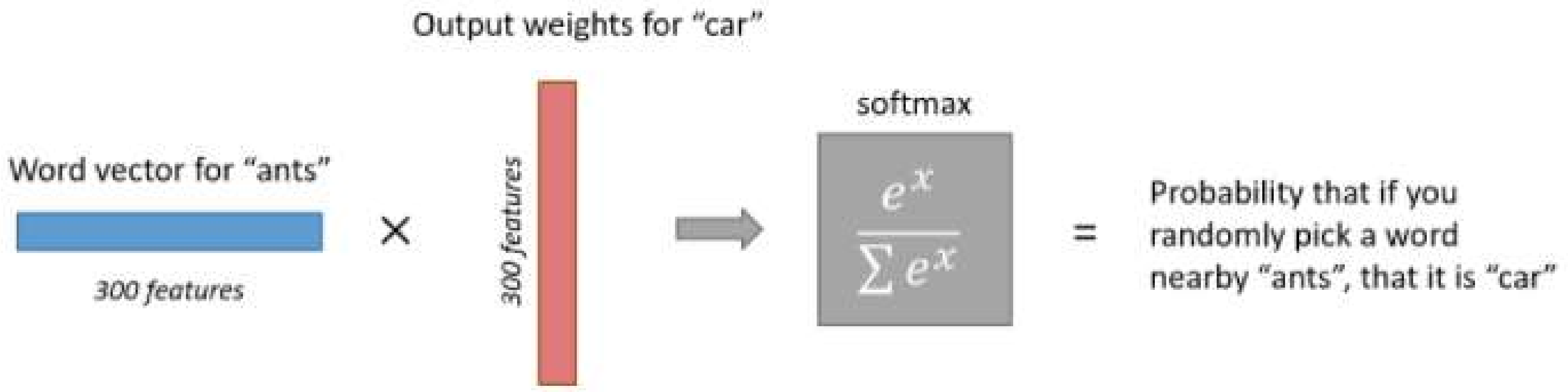
The Hidden Layer

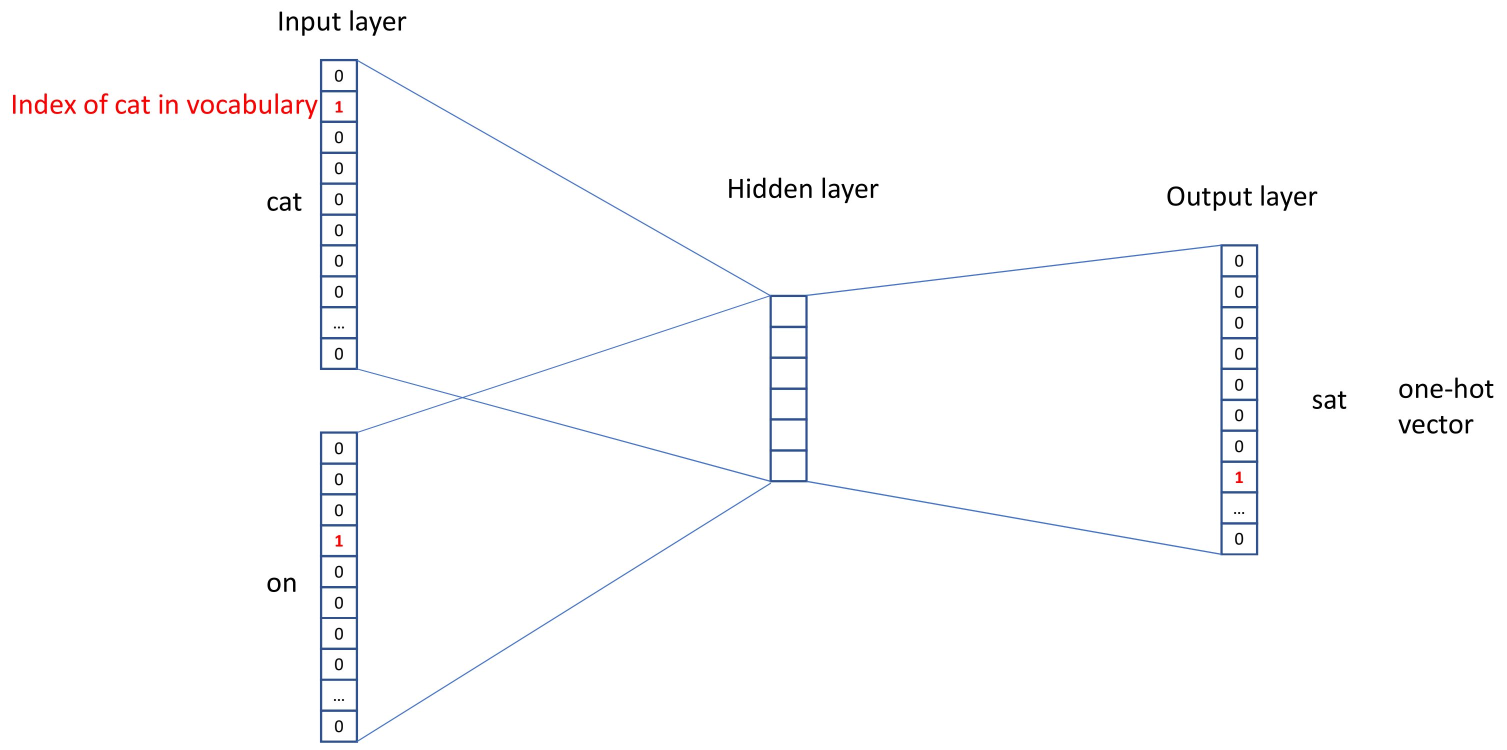
- Let us say we are learning vectors with 300 features.
- Hidden layer is a weighted 10,000 X 300 matrix
- The *rows* of this weight matrix, these are actually what will be our word vectors!
- Our goal is to learn the hidden layer weight matrix (lookup table)

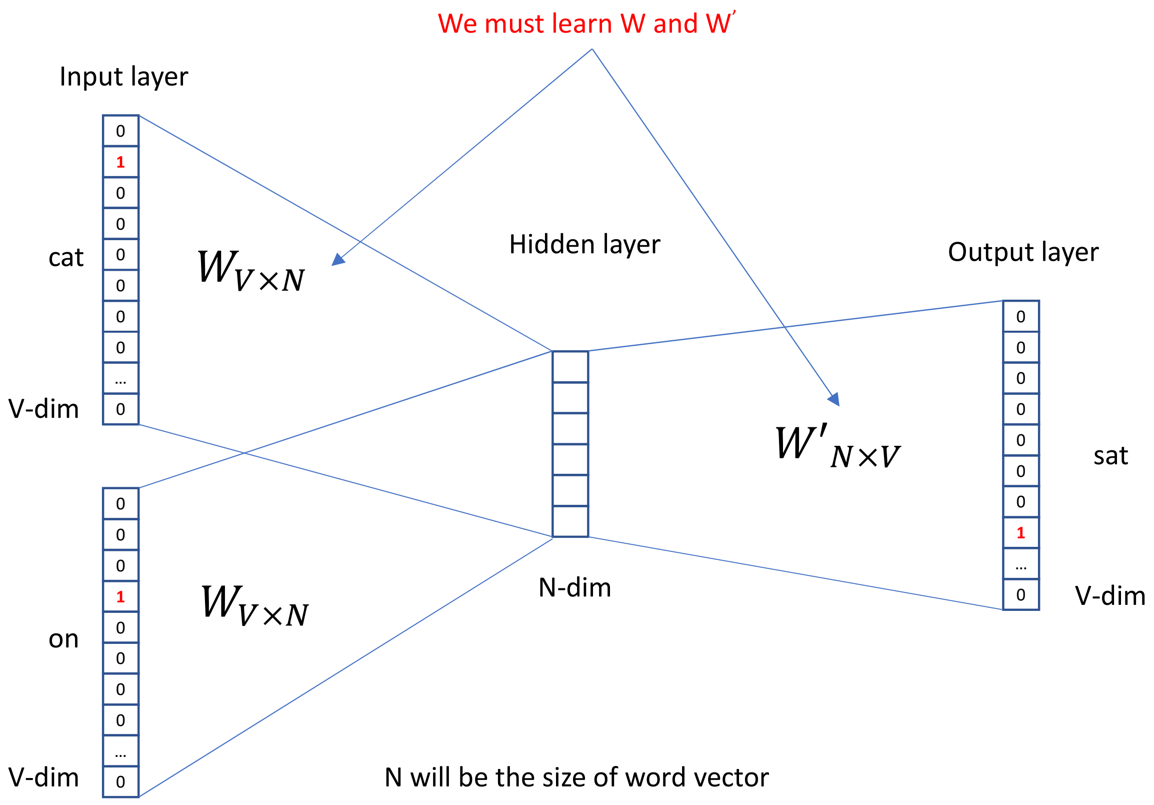


The Output Layer

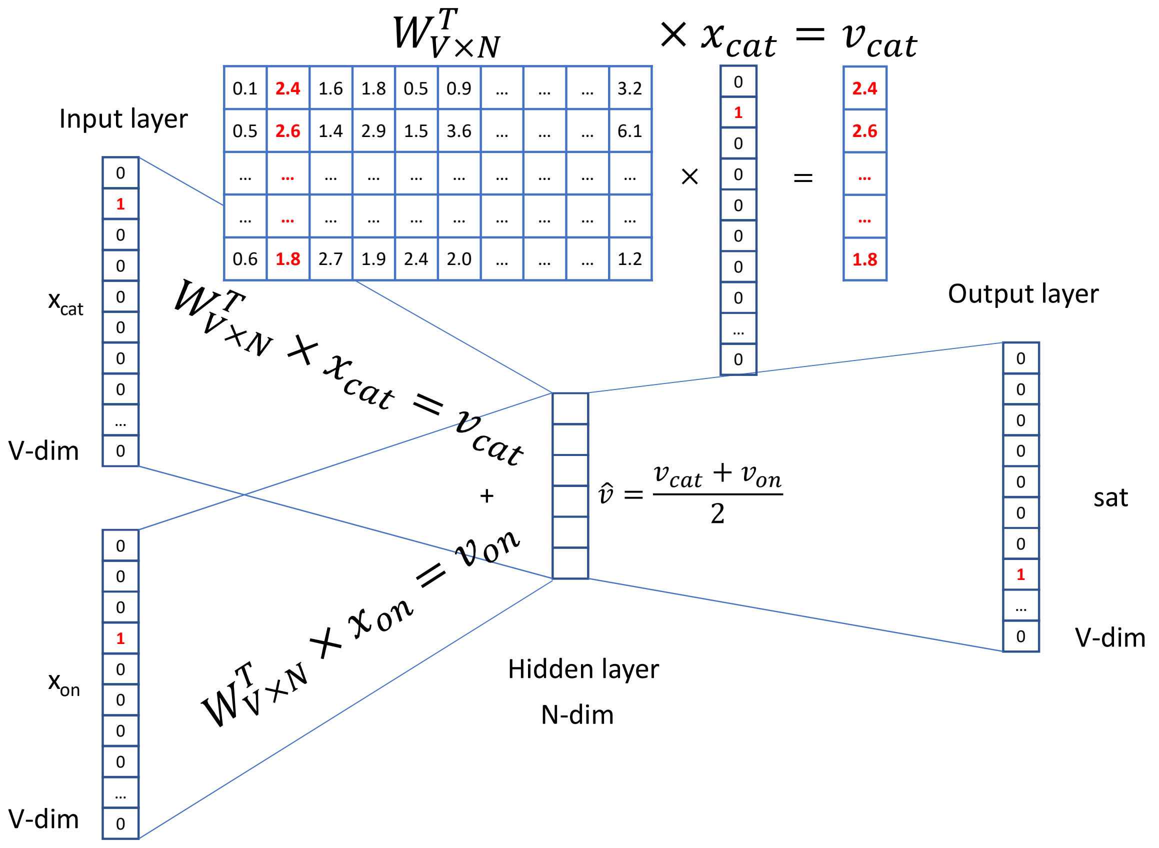
- Input: The 1x300 word vector for ‘ants’ then gets fed to the output layer. The output layer is a sotfmax regression classifier
  - Softmax regression (or multinomial logistic regression) is a generalization of logistic regression to the case where we want to handle multiple classes.
- Output: between 0 and 1, and the sum of all these output values will add up to 1.
  - Specifically, each output neuron has a weight vector which it multiplies against the word vector from the hidden layer, then it applies the function exp(x) to the result.
  - Finally, in order to get the outputs to sum up to 1, we divide this result by the sum of the results from all 10,000 output nodes.


Output weights for "car"
Word vector for "ants"
300 features
×
300 features
softmax
$\frac{e^x}{\sum e^x}$
=
Probability that if you randomly pick a word nearby "ants", that it is "car"

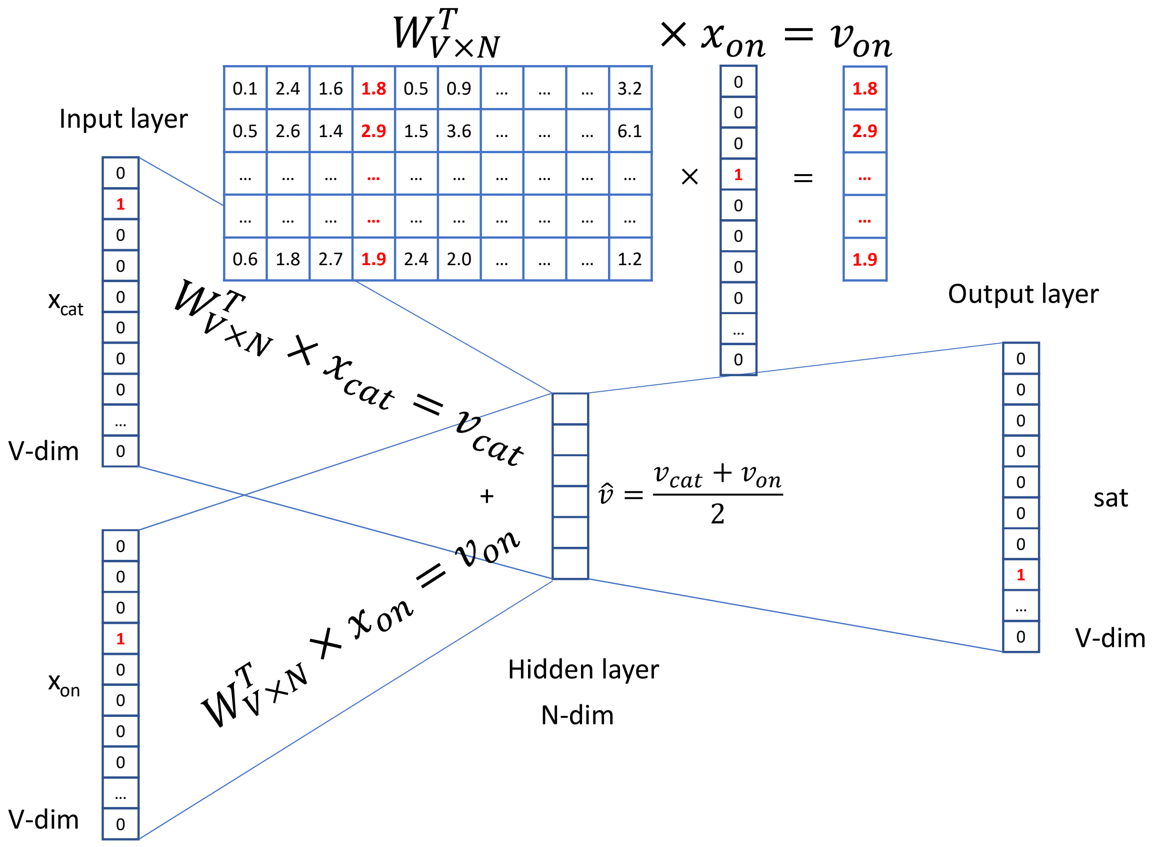

Input layer
Index of cat in vocabulary
cat
0
1
0
0
0
0
0
0
…
0
on
0
0
0
1
0
0
0
0
…
0
Hidden layer
Output layer
0
0
0
0
0
0
0
1
…
0
sat
one-hot
vector



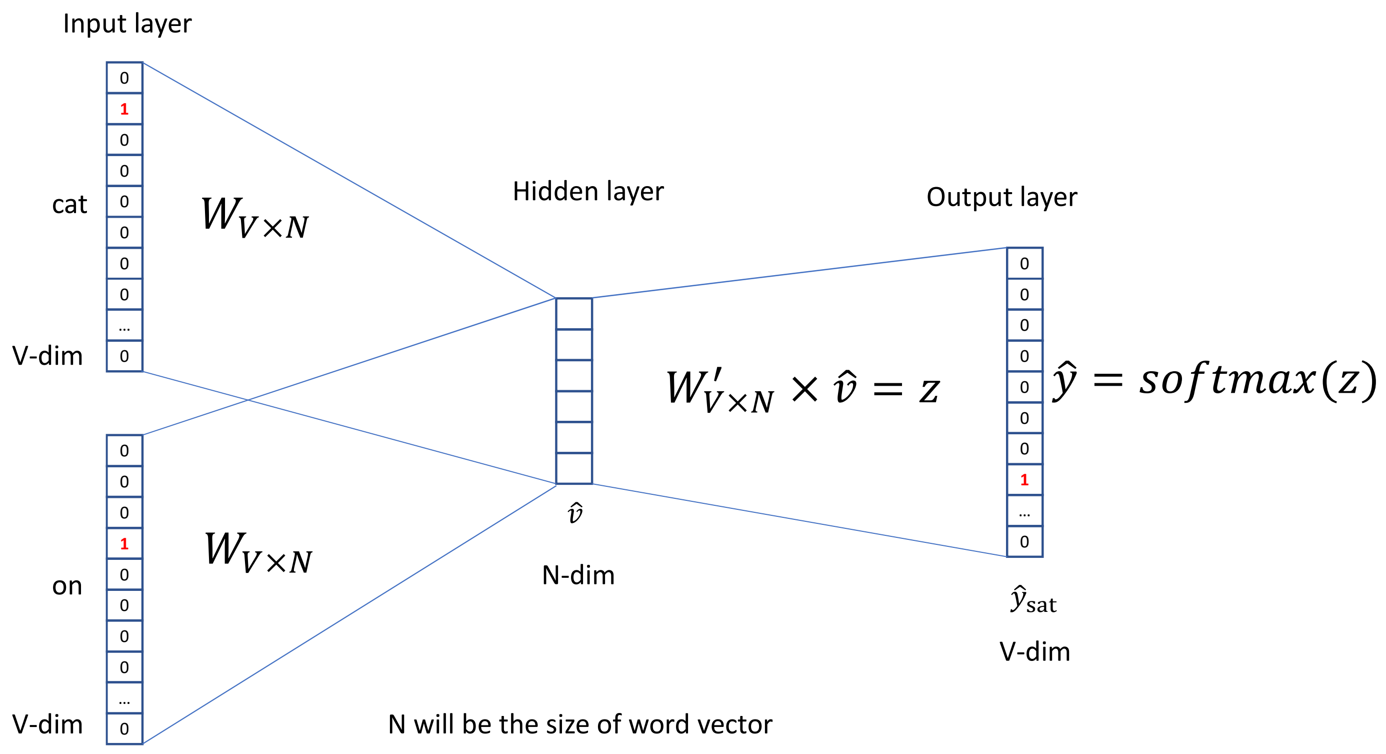
N will be the size of word vector



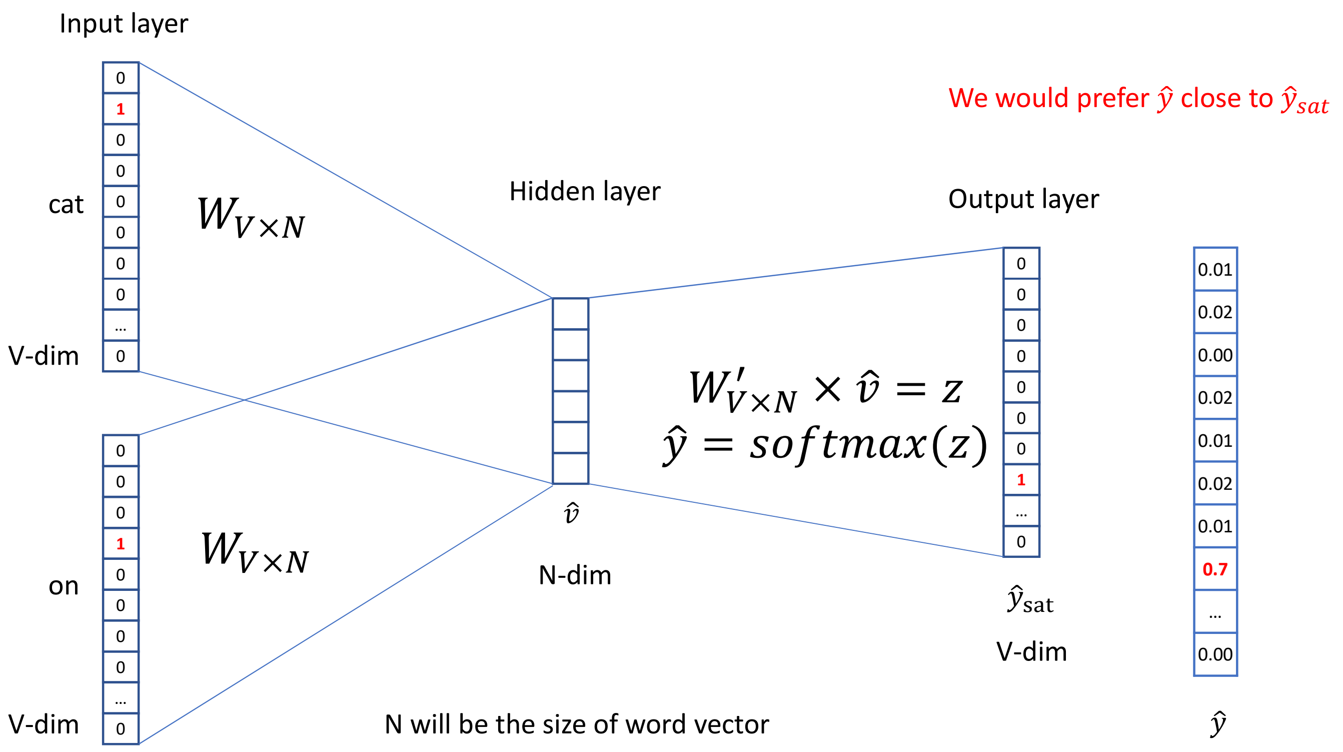
$W_{V\times N}^T \times x_{cat} = v_{cat}$
Input layer
0.1 2.4 1.6 1.8 0.5 0.9 … … … 3.2
0.5 2.6 1.4 2.9 1.5 3.6 … … … 6.1
0.6 1.8 2.7 1.9 2.4 2.0 … … … 1.2
2.4 2.6 … … 1.8
x_{cat}
V-dim
x_{on}
V-dim
$W_{V\times N}^T \times x_{cat} = v_{cat}$
+
$W_{V\times N}^T \times x_{on} = v_{on}$
$\hat{v} = \frac{v_{cat} + v_{on}}{2}$
Hidden layer
N-dim
Output layer
sat
V-dim



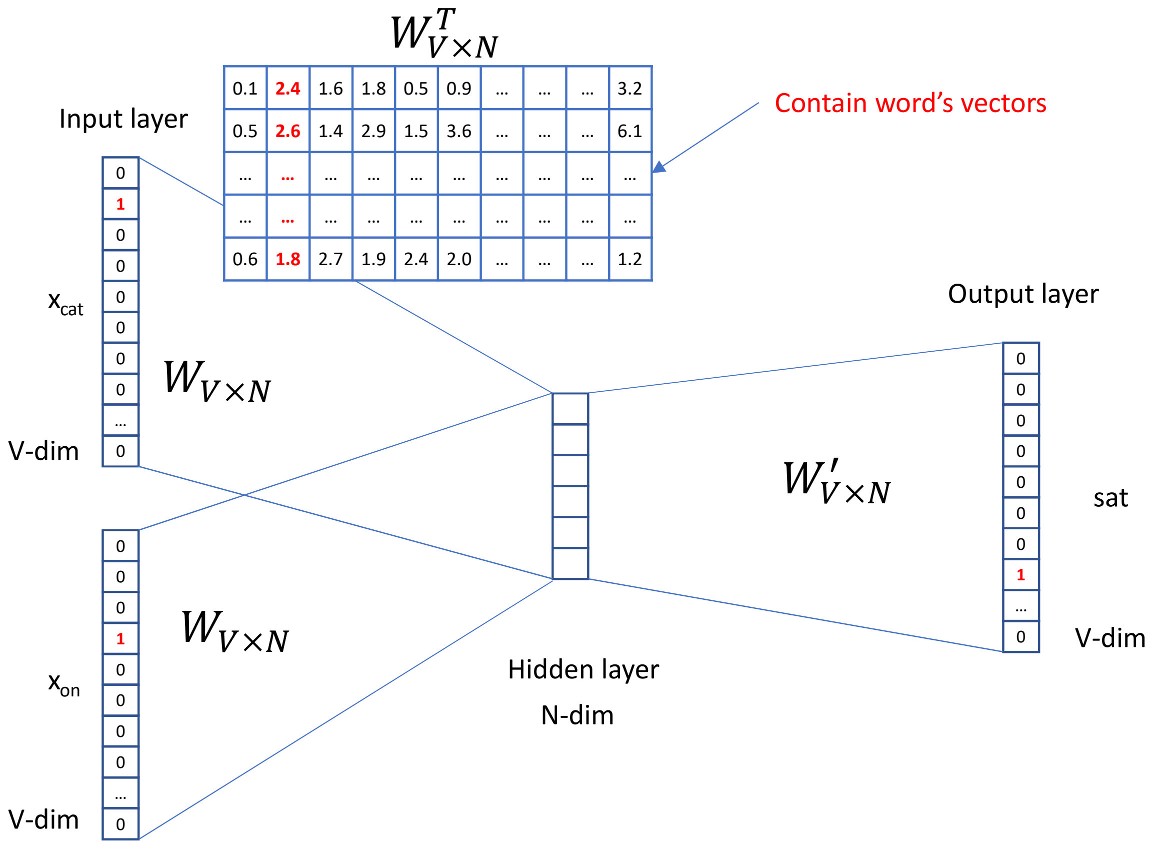
$W^T_{V\times N}$
$\times x_{on} = v_{on}$
0.1 2.4 1.6 1.8 0.5 0.9 … … … 3.2
0.5 2.6 1.4 2.9 1.5 3.6 … … … 6.1
0.6 1.8 2.7 1.9 2.4 2.0 … … … 1.2
1.8 2.9 … … 1.9
Input layer
x_{cat}
V-dim
x_{on}
V-dim
$W^T_{V\times N} \times x_{cat} = v_{cat}$
+
$W^T_{V\times N} \times x_{on} = v_{on}$
$\hat{v} = \frac{v_{cat} + v_{on}}{2}$
Hidden layer
N-dim
Output layer
sat
V-dim



N will be the size of word vector



Input layer
cat
V-dim
on
V-dim
0
1
0
0
0
0
0
0
...
0
0
0
0
1
0
0
0
0
...
0
$W_{V\times N}$
$W_{V\times N}$
Hidden layer
$\hat{v}$
N-dim
$W'_{V\times N} \times \hat{v} = z$
$\hat{y} = softmax(z)$
Output layer
0
0
0
0
0
0
0
1
...
0
$\hat{y}_{\text{sat}}$
V-dim
0.01
0.02
0.00
0.02
0.01
0.02
0.01
0.7
...
0.00
$\hat{y}$
We would prefer $\hat{y}$ close to $\hat{y}_{sat}$
N will be the size of word vector



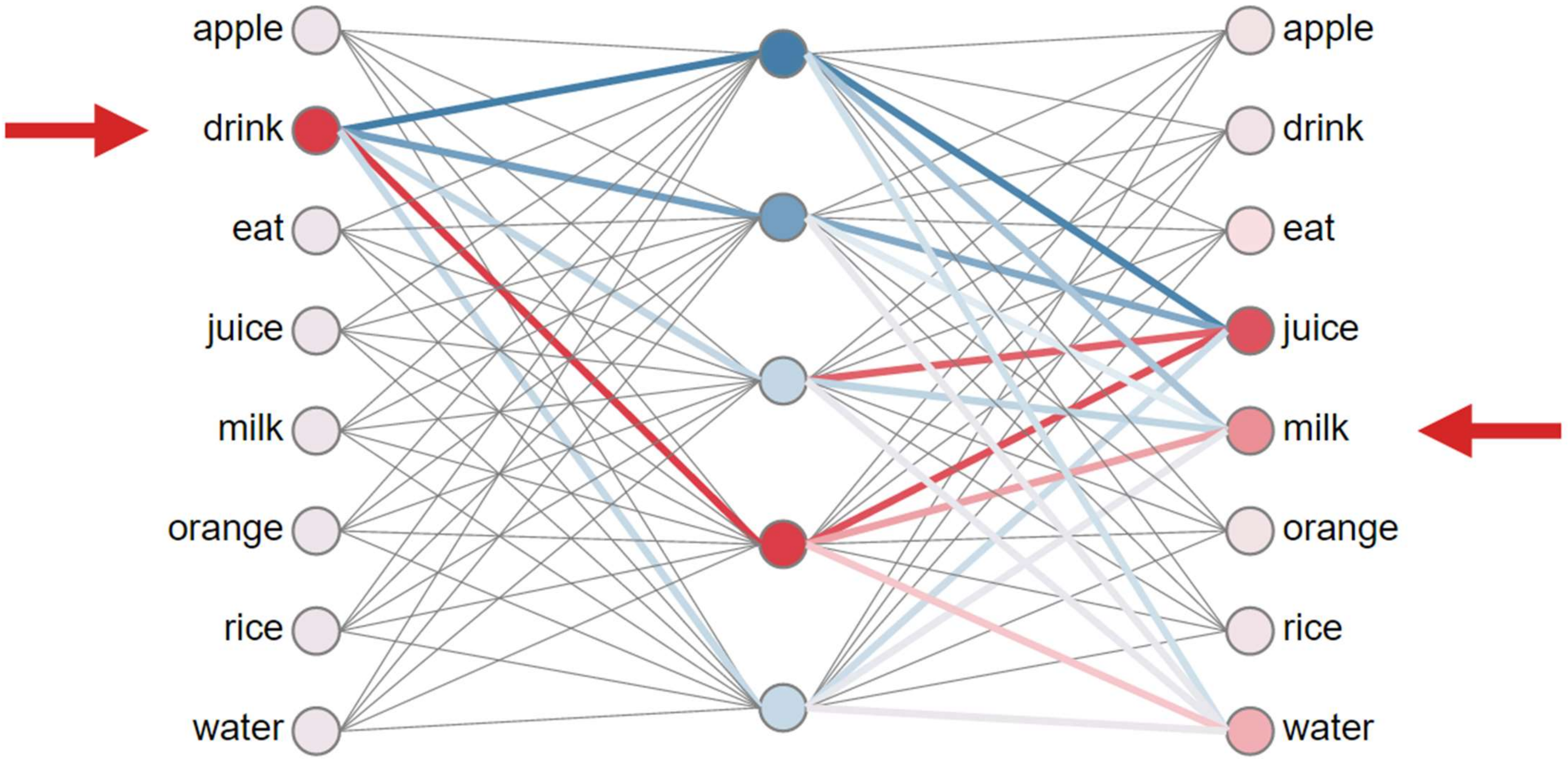
We can consider either W or W’ as the word’s representation. Or even take the average.

Training Data

1. eat|apple
2. eat|orange
3. eat|rice
4. drink|juice
5. drink|milk
6. drink|water
7. orange|juice
8. apple|juice
9. rice|milk
10. milk|drink
11. water|drink
12. juice|drink

Concept :

1. Milk and Juice are drinks
2. Apples, Oranges and Rice can be eaten
3. Apples and Orange are also juices
4. Rice milk is a actually a type of milk!


apple
drink
eat
juice
milk
orange
rice
water
apple
drink
eat
juice
milk
orange
rice
water

Word Embedding Visualization
http://ronxin.github.io/wevi/

- GloVe is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space.
  - https://nlp.stanford.edu/projects/glove/
- Doc2Vec:
  - Le and Mikolov, Distributed Representations of Sentences and Documents
  - https://radimrehurek.com/gensim/models/doc2vec.html

- Mikolov et al, Distributed Representations of Words and Phrases and their Compositionality
  - https://code.google.com/archive/p/word2vec/