Jan. 6, 2020

# CS 886 Deep Learning and NLP

Ming Li

# Prelude: the importance of language

Do we have better neural networks than them?



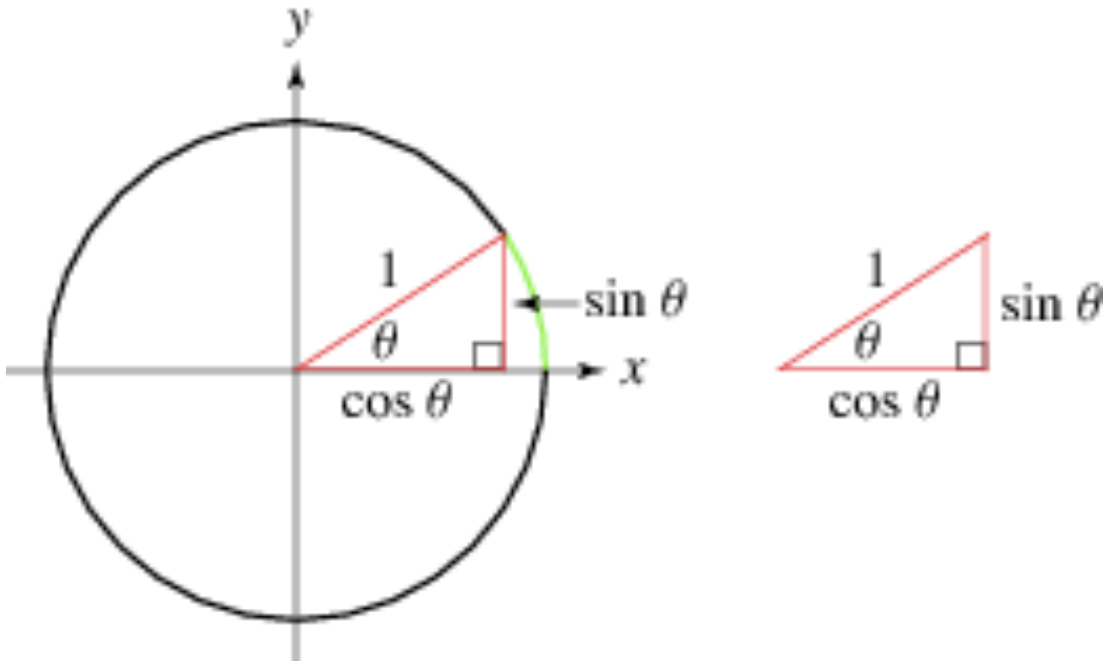


# CONTENT


# Word2Vec

LECTURE ONE

## Things you need to know:




### Dot product:

$a \bullet b = ||a||||b||\cos(\theta_{ab})$

$= a_1b_1+a_2b_2+ \ldots +a_nb_n$

One can derive Cosine similarity

$\cos \theta_{ab} = a \bullet b \ / \ ||a||||b||$

### Softmax Function:

If we take an input of [1,2,3,4,1,2,3], the softmax of that is [0.024, 0.064, 0.175, 0.475, 0.024, 0.064, 0.175].
The softmax function highlights the largest values and suppress other values, so that they are all positive and sum to 1.

$$\sigma(x_j) = \frac{e^{x_j}}{\sum_i e^{x_i}}$$


# Word2Vec

Two cases of transforming from discrete to continuous space

1. Calculus (for computing the space covered by a curve)
2. Word2Vec (for computing the “space” covered by the meaning of a word)

# Traditional representation of a word's meaning

1. Dictionary, not too useful in computational linguistic research.
2. WordNet

   It is a graph of words, with relationships like "is-a", synonym sets.

   Problems: Depend on human labeling hence missing a lot, hard to automate this process.
3. These are all using atomic symbols: hotel, motel, equivalent to 1-hot vector:

   Hotel: [0,0,0,0,0,0,0,0,1,0,0,0,0,0]

   Motel: [0,0,0,0,1,0,0,0,0,0,0,0,0,0]

   These are called one-hot representations. Very long: 13M (google crawl).

   Example: inverted index.

# Problems.

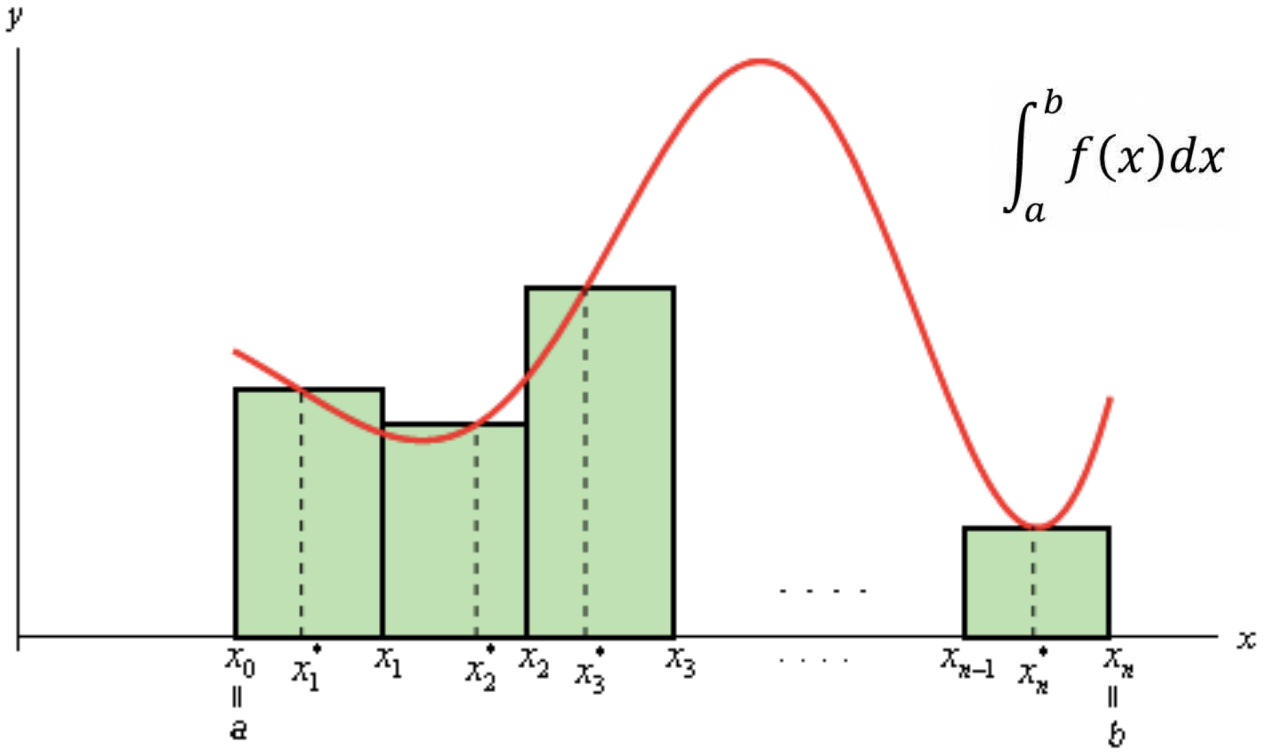
There is no natural meaning of similarity,

$$\text{hotel} \cdot \text{motel}^T = 0$$

No inherent notion of similarity with 1-hot vectors.

They are very long too. For daily speech, 20 thousand words, 50k for machine translation. ½ million for material science. Google web crawl, 13M words.

How do we solve the problem? This is what Newton and Leibniz did for calculus (the name f(x) is like 1-hot vector):




$$\int_a^b f(x)dx$$ when #rectangles → ∞

# Let's do something similar:

1. Use a lot of "rectangles", a vector of numbers, to represent to approximate the meaning of a word.
2. What represents the meanings of a word?

   "You shall know a word by the company it keeps" – J.R. Firth, 1957

Thus, our goal is to assign each word a vector such that similar words have similar vectors (by dot-product).

We will believe JR Firth and use a neural network to train (low dimension) vectors such that if two words appear together in a text each time, they should get slightly closer. This allows us to use a massive corpus without annotation (core theme of this course)! Thus, we will scan thru training data by looking at a window 2d+1 at a time, given a center word, trying to predict d words on the left and d words on the right.

## To design a neural network for this:

More specifically, for a center word $w_t$, and "context words" $w_{t'}$, within a window of some fixed size say 5 (t'=t-1, … t-5, t+1, … , t+5) we use a neural network to predict all $w_{t'}$ to maximize:

$$p(w_{t'}|w_t) = \ldots$$

This has a loss function

$$L = 1 - p(w_{t'}|w_t)$$

Thus by looking at many positions in a big corpus, we keep on adjusting these vectors to minimize this loss, we arrive at a (low dimensional) vector approximation of the meaning of each word, in the sense that if two words occur in close proximity often then we consider them similar.

## To design a neural network for this:

Thus the objective function is: maximize the probability of any context word given the current center word:

$$L'(\theta) = \Pi_{t=1..T} \Pi_{d=-1..-5,1..5} P(w_{t+d} \mid w_t, \theta)$$

Where θ is all the variables we optimize (I,e, the vectors).

Taking negative logarithm (and average per word) so that we can minimize

$$L(\theta) = -1/T \Sigma_{t=1..T} \Sigma_{d=-1 .. -5, 1,..,5} \log P(w_{t+d} \mid w_t)$$

Then what is $P(w_{t+d} \mid w_t)$? We can just take their vector dot products, and then take softmax, to approximate it, letting v be the vector for word w:

$$L(\theta) \approx -1/T \Sigma_{t=1..T} \Sigma_{d=-1 .. -5, 1,..,5} \log \text{Softmax}(v_{i+d} \bullet v_t)$$

# To design a neural network for this:

Last slide:

$$L(\theta) \approx -\frac{1}{T} \sum_{t=1..T} \sum_{d=-1 .. -5, 1,..,5} \log \text{Softmax}(v_{i+d} \bullet v_t)$$

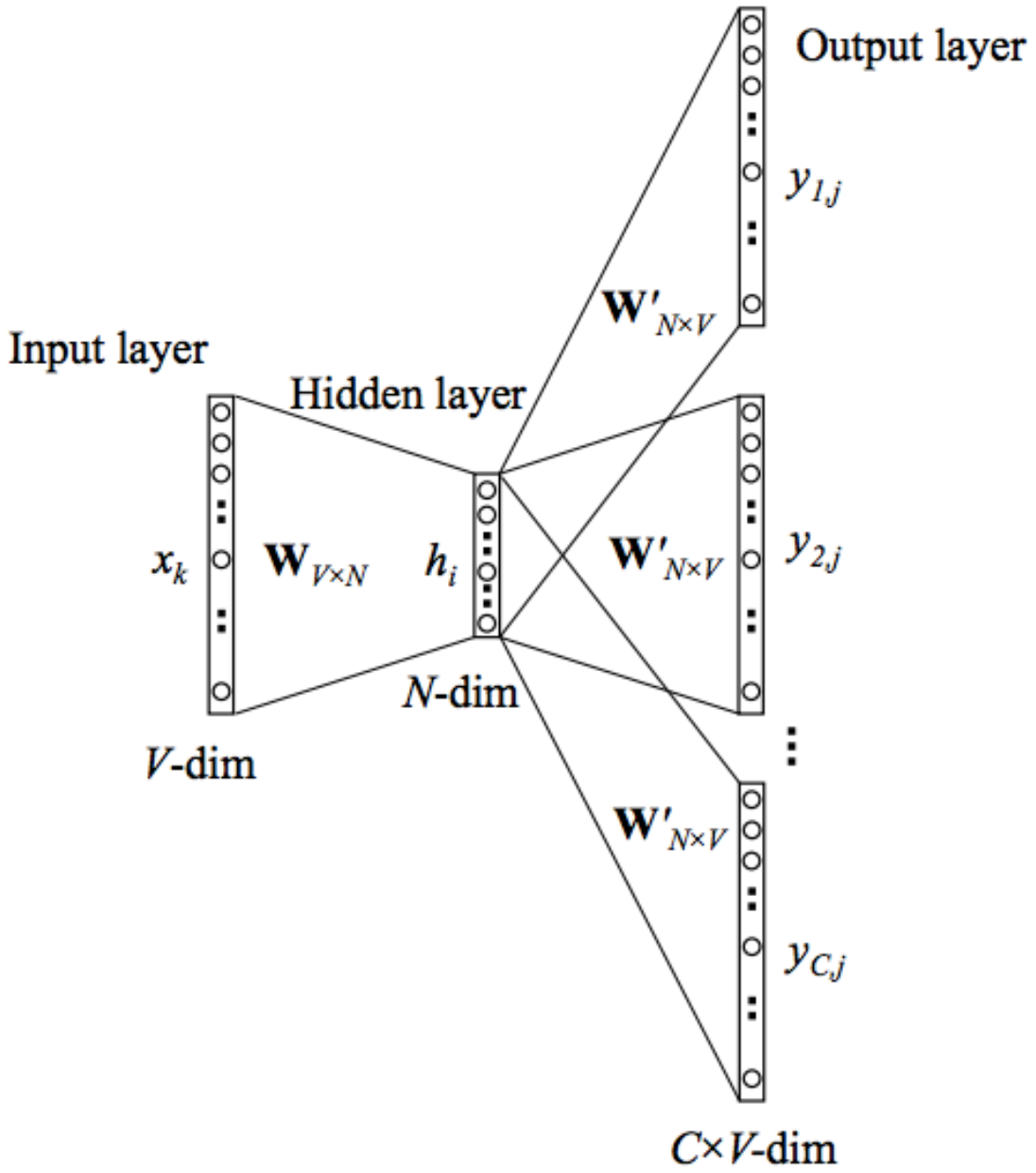
The softmax of a center word c, and a context/outside word o

$$\text{Softmax}(v_o \bullet v_c) = e^{(v_o \bullet v_c)} / \sum_{k=1..V} e^{(v_k \bullet v_c)}$$

Note, the index runs over the dictionary of size V, not the whole text T.

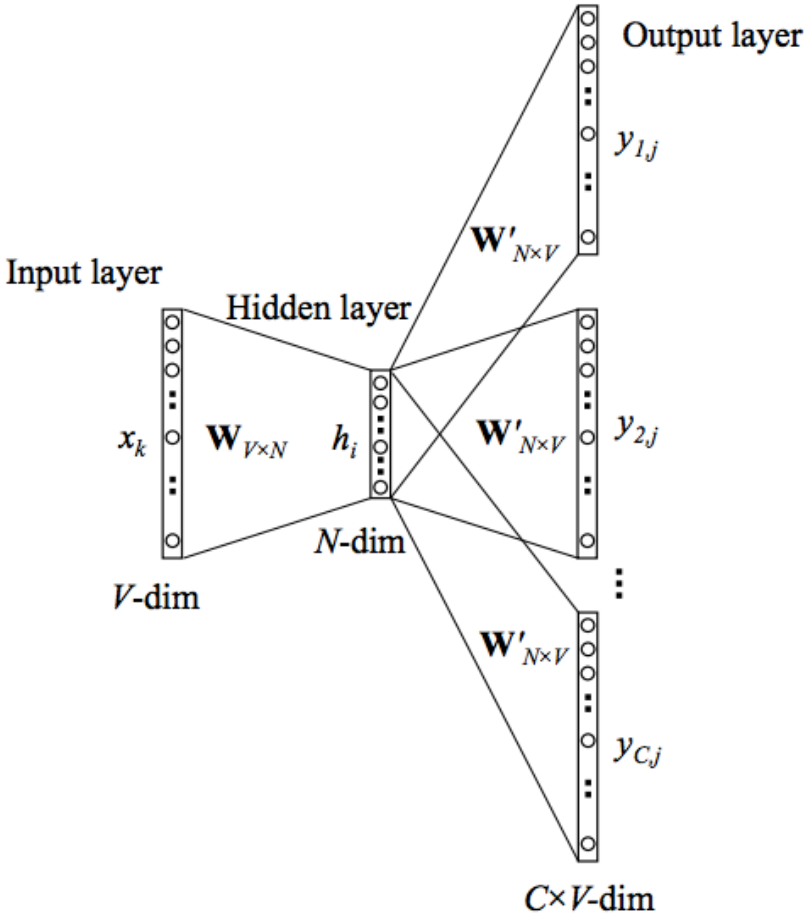
# The skip-gram model

- Vocabulary size: V
- Input layer: center word in 1-hot form.
- k-th row of $W_{VxN}$ is center vector of k-th word.
- k-th column of $W'_{NXV}$ is context vector of the k-th word in V. Note, each word has 2 vectors, both randomly initialized.
- The output column $y_{ij}$, i=1..C, has 3 steps
  1) Use the context word 1 hot vector to choose its column in $W'_{NxV}$
  2) dot product with $h_i$ the center word
  3) compute the softmax



# The Training of θ

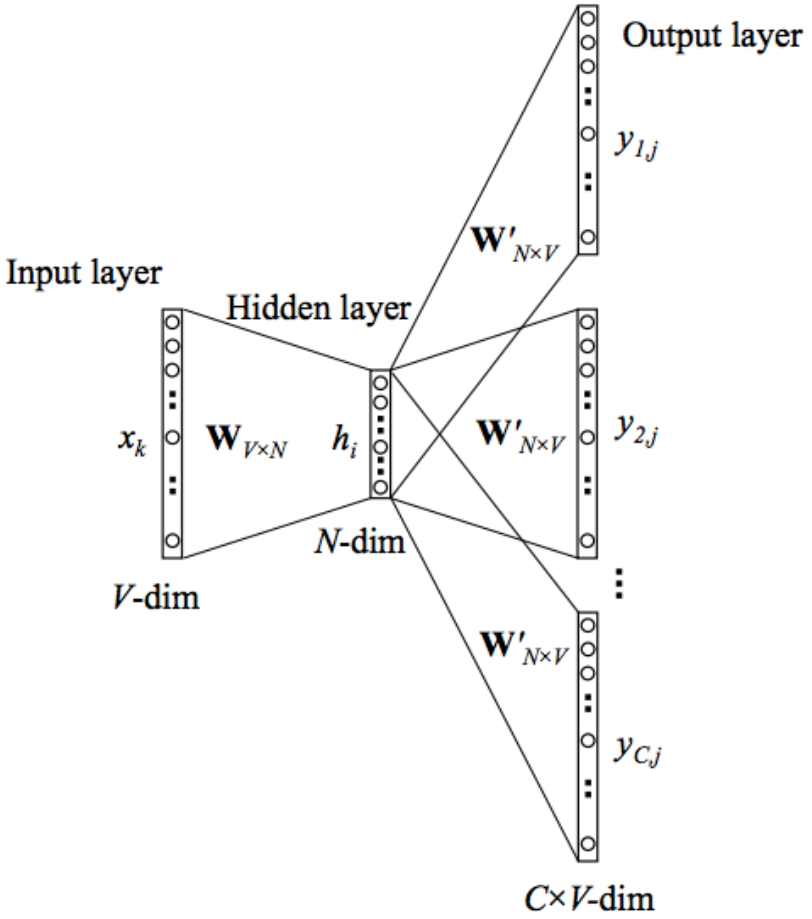
- We will train both $W_{VxN}$ and $W'_{NxV}$
- I.e. compute all vector gradients.
- Thus θ is in space $R^{2NV}$, N is vector size, V is number of words.
- $\partial L(\theta) / \partial v$ , for all vectors in θ.




$$\theta = \begin{bmatrix} v_{aardvark} \\ v_a \\ . \\ . \\ v_{zebra} \\ u_{aardvark} \\ u_a \\ . \\ . \\ u_{zebra} \end{bmatrix} \text{ in } R^{2NV}$$

# Gradient Descent

- $\theta^{new} = \theta^{old} - \alpha \, \partial L(\theta^{old}) / \partial \, \theta^{old}$
- Stochastic gradient descent (SGD): Just do one position (one center word and its context words) at a time.




$$\theta = \begin{bmatrix} v_{aardvark} \\ v_a \\ . \\ . \\ v_{zebra} \\ u_{aardvark} \\ u_a \\ . \\ . \\ u_{zebra} \end{bmatrix} \text{ in } R^{2NV}$$

# Negative Sampling in Original Word2Vec

- In our $L(\theta) \approx -1/T \sum_{t=1..T} \sum_{d=-1 .. -5, 1,..,5} \log \text{Softmax}(v_{i+d} \bullet v_t)$ where

  $\text{Softmax}(v_o \bullet v_c) = e^{(v_o \bullet v_c)} / \sum_{k=1..V} e^{(v_k \bullet v_c)}$

  Each time we have to calculate $\sum_{k=1..V} e^{(v_k \bullet v_c)}$, this is too expensive.
- To overcome this, use negative sampling. Overall objective function: $L(\theta) = 1/T \sum_{t=1..T} L_t(\theta)$

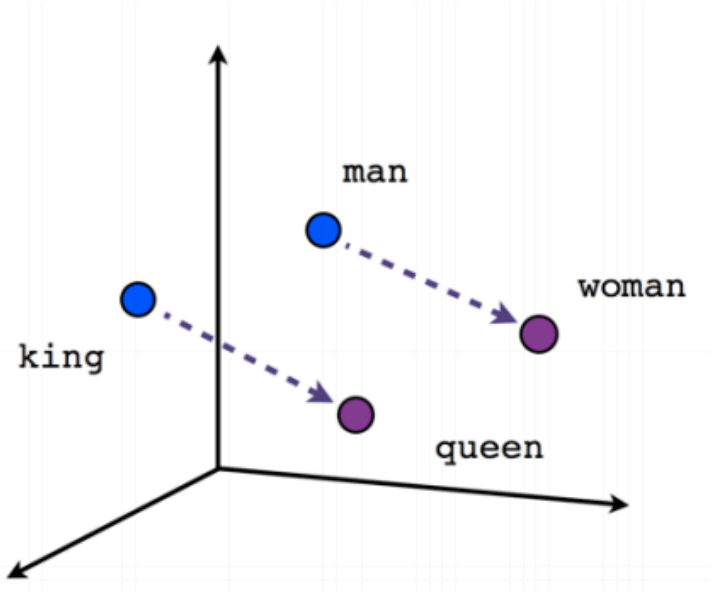
  $L_t(\theta) = \log \sigma(u_o^T v_c) + \sum_{t=1..k} E_{j \sim P(w)} [\log \sigma(-u_j^T v_c)]$

  $= \log \sigma(u_o^T v_c) + \sum_{j \sim P(w)} [\log \sigma(-u_j^T v_c)]$
- Where the sigmoid function $\sigma(x) = 1/1+e^{-x}$, treated as probability for ML people. I.e. maximize the first term, taking k=10 random samples in the second term.
- For sampling, we can use unigram distribution $U(w)$ or $U(w)^{3/4}$ for rare words.

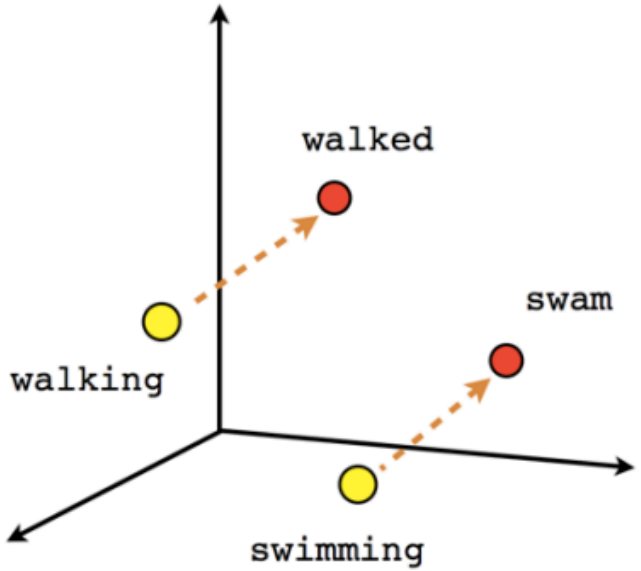
# CBOW

- What about we predict center word, given context word, opposite to the skip-gram model?
- Yes, this is called Continuous Bag Of Words model in the original Word2Vec paper.

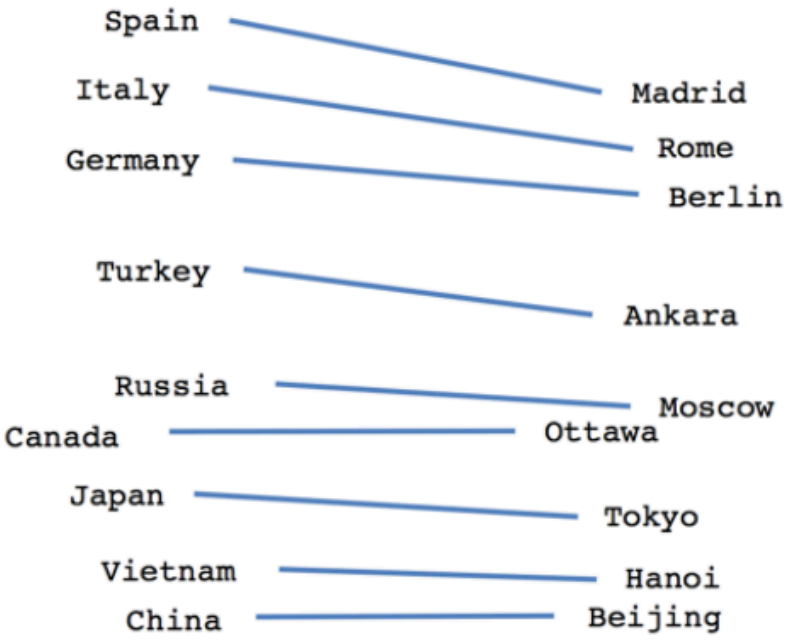
# Results




Male-Female

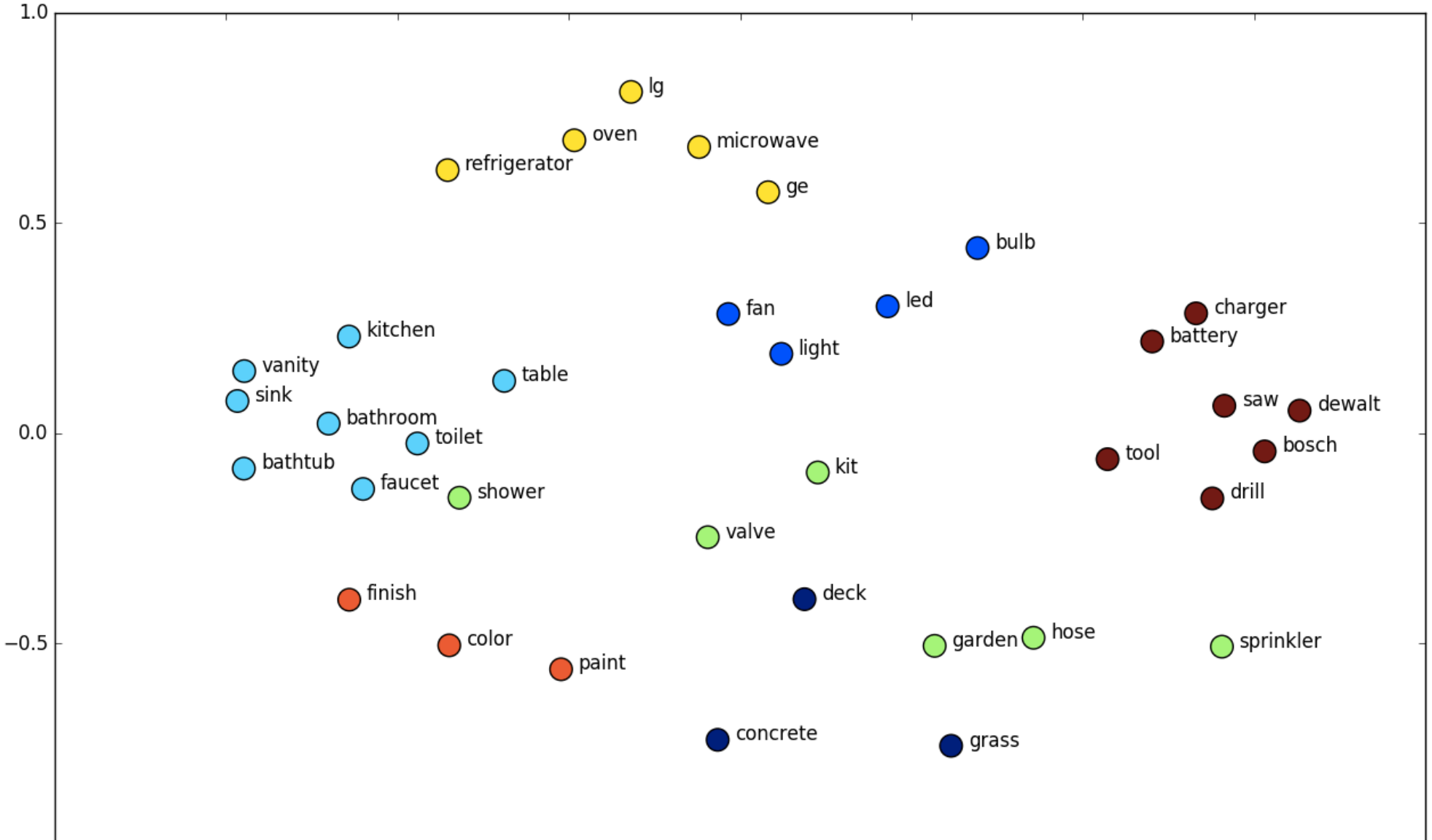



Verb tense

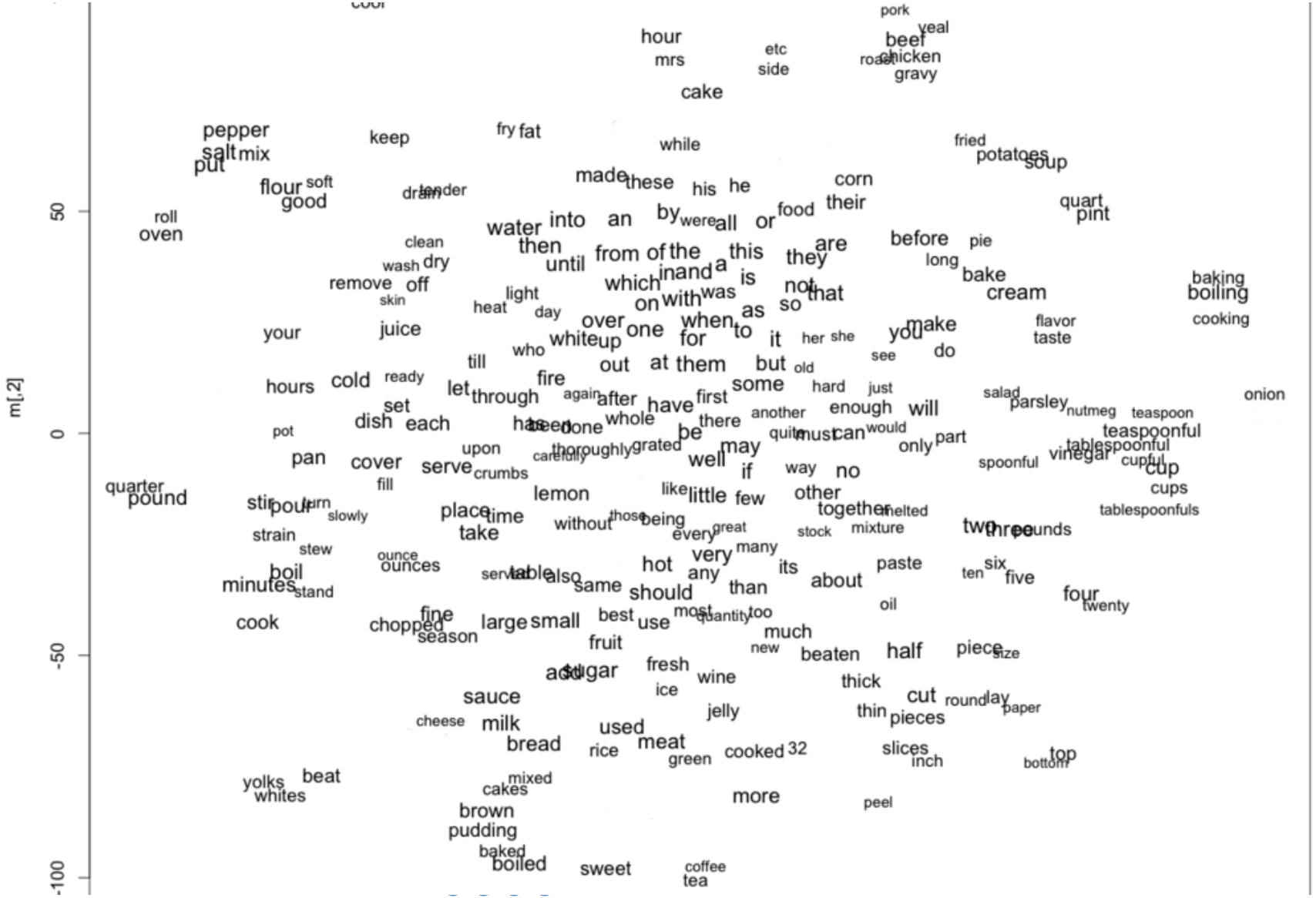



Country-Capital

# Results


1.0
0.5
0.0
−0.5
lg
oven
microwave
refrigerator
ge
bulb
led
fan
charger
battery
kitchen
light
vanity
table
sink
saw
dewalt
bathroom
toilet
tool
bosch
bathtub
kit
faucet
shower
drill
valve
finish
deck
color
garden
hose
sprinkler
paint
concrete
grass

# More realistic data – not everything is perfect



# An interesting application outside CS

- Nature, July 2019 V. Tshitonya et al, “Unsupervised word embeddings capture latent knowledge from materials science literature”.
- Lawrence Berkeley lab material scientists applied word embedding to 3.3 million scientific abstracts published between 1922-2018. V=500k.  Vector size: 200 dimension, used skip-gram model
- With no explicit insertion of chemical knowledge
- Captured things like periodic table and structure-property relationship in materials:

  ferromagnetic − NiFe + IrMn ≈ antiferromagnetic
- Discovered new thermoelectric materials: “would be years before their discovery”.

# Beyond Word2Vec

- Co-occurrence matrix

  Window based co-occurrence

  Document based co-occurrence

  Ignore the, he, has … frequent words.

  Close proximity weigh more …
- In word2vec, if a center word w appear again, we have to repeat this process. Here they are processed together. Also consider documents.
- Symmetric.
- SVD decomposition, this was before Word2Vec. But it is $O(nm^2)$, too slow for large data.

# GloVe (Global vectors model)

- Combining Word2Vec and Co-occurrence matrix approaches. Optimize

$$L(\theta) = \tfrac{1}{2} \Sigma_{i,j=1..W} f(P_{i,j}) (u_i^T v_j - \log P_{i,j})^2$$

Where, u, v vectors are still the same, $P_{i,j}$ is the count that $u_i$ and $v_j$ co-occur. Essentially This says, the more $u_i$,$v_j$ co-occur, the larger their dot product should be. f gets rid of too frequent occurrences.

- What about these two vectors? X=U+V works.
- Polysemy? Somebody please present S. Arora et al and related papers.

# Literature & Resources for Word2Vec

Bengio et al, 2003, A neural probabilistic language model.

Collobert & Weston 2008, NLP (almost) from scratch

Mikolov et al 2013, word2vec paper

Pennington Socher, Manning, GloVe paper, 2014

Rohde et al 2005 (SVD paper) An improved model of semantics similarity based on lexical co-occurrence.

Plus thousands more.

Resources:

https://mccormickml.com/2016/04/27/word2vec-resources/

https://github.com/clulab/nlp-reading-group/wiki/Word2Vec-Resources

# Project Ideas

1. Current word2vec or GloVe approaches are good for high frequency words. The lower frequency words are overwhelmed by the higher frequency ones. Can you experiment on some stratified strategy (for example, by removing higher frequency words, but not their vectors) so that we could gradually also train the relationship for lower frequency words.
2. Can you try to investigate some sort of “conditional word2vec” so that it solves polysemy problems? (Note, do literature search first.) For example, if a word of technical flavour appears in the same page as “apple”, then “apple” is more likely to be a piece of electronic device. But one needs to do this automatically, without human intervention.


# Attention and Transformers

LECTURE TWO