Distributional Semantics Part 2: word2vec

LIN 313 Language and Computers
UT AustinFall 2025
Instructor: Gabriella Chronis

Admin

- Test grades posted this afternoon
- Reading "Man is to programmer as woman is to homemaker? Debiasing Word Embeddings" for Monday 11/3
    - focus on sections 1-4 ; skim the rest / don't worry about the math

Overview for today

- Go over co-occurrence based semantic spaces
- How do we compare vectors? Cosine similarity
- building a semantic space with neural network: word2vec

Review: Word Vectors

The idea of vector semantics is to represent a word as a point in a multidimensional semantic space that is derived from the distributions of word neighbors

- each value in the vector is a coordinate along a dimension in Euclidean space
    - euclidean spaces:
        - 0 dimensions (no variation)
        - 1 dimension: x-axis (the number line - variation along one dimension or variable)
        - 2 dimensions: x and y axes
        - 3 dimensions:
- the coordinates

Count-Based Semantic Space

The goal is to represent words in terms of their shared contexts (their distributions). Words that are similarly distributed across contexts are similar in meaning.

1. choose a corpus
2. process the corpus
    a. make a list of words (the vocabulary)
        i. e.g. 'pet'
    b. make a list of contexts
        i. could be documents
        ii. could be words on either side of the target word
            1. e.g. 'my ___ salamander' , 'she ____ the'
3. make a matrix (a table) with word types as rows and contexts as columns

Term x Document Matrix for Shakespeare

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Figure 6.2 The term-document matrix for four words in four Shakespeare plays. Each cell contains the number of times the (row) word occurs in the (column) document.

J&M Speech and Language Processing

Term x Document Matrix for Shakespeare

columns = 1 for each doc

rows

= 1 for each word in vocab

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Figure 6.2 The term-document matrix for four words in four Shakespeare plays. Each cell contains the number of times the (row) word occurs in the (column) document.

[J&M Speech and Language Processing]()

Document Vectors

Looking at the columns of our co-occurrence matrix gives us vector representations of each document

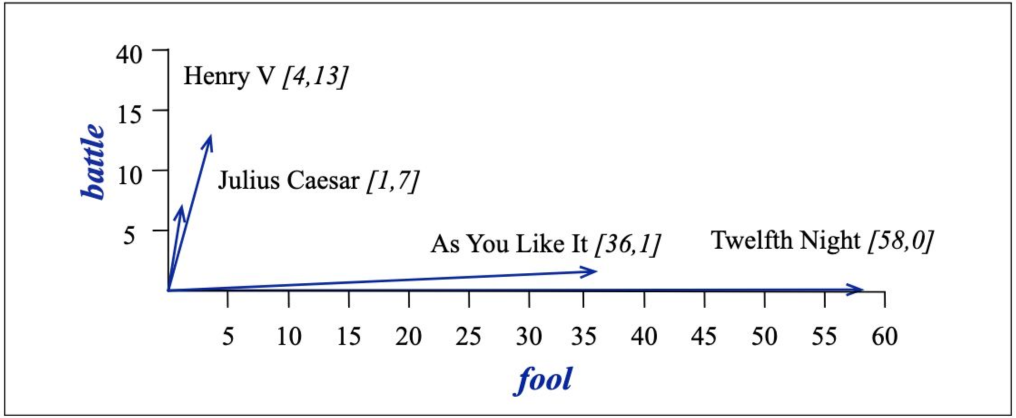


Figure 6.4 A spatial visualization of the document vectors for the four Shakespeare play documents, showing just two of the dimensions, corresponding to the words *battle* and *fool*. The comedies have high values for the *fool* dimension and low values for the *battle* dimension.

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Figure 6.3 The term-document matrix for four words in four Shakespeare plays. The red boxes show that each document is represented as a column vector of length four.

[J&M Speech and Language Processing]

Word Vectors

word x document vectors

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Figure 6.5 The term-document matrix for four words in four Shakespeare plays. The red boxes show that each word is represented as a row vector of length four.

Looking at the **rows** of our co-occurrence matrix gives us vector representations of each word in the vocabulary

[J&M Speech and Language Processing]

Word Vectors

word x document vectors

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Figure 6.5 The term-document matrix for four words in four Shakespeare plays. The red boxes show that each word is represented as a row vector of length four.

Looking at the **rows** of our co-occurrence matrix gives us vector representations of each word in the vocabulary

word x word vectors

is traditionally followed by	**cherry**	pie, a traditional dessert
often mixed, such as	**strawberry**	rhubarb pie. Apple pie
computer peripherals and personal	**digital**	assistants. These devices usually
a computer. This includes	**information**	available on the internet

	aardvark	...	computer	data	result	pie	sugar	...
cherry	0	...	2	8	9	442	25	...
strawberry	0	...	0	0	1	60	19	...
digital	0	...	1670	1683	85	5	4	...
information	0	...	3325	3982	378	5	13	...

Figure 6.6 Co-occurrence vectors for four words in the Wikipedia corpus, showing six of the dimensions (hand-picked for pedagogical purposes). The vector for *digital* is outlined in red. Note that a real vector would have vastly more dimensions and thus be much sparser.

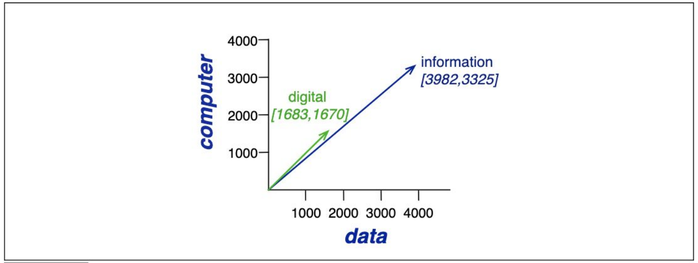


Figure 6.7 A spatial visualization of word vectors for *digital* and *information*, showing just two of the dimensions, corresponding to the words *data* and *computer*.

[J&M Speech and Language Processing]

Semantic Similarity

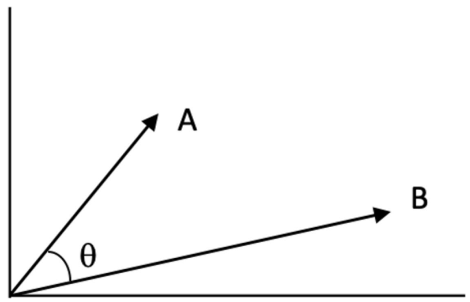
Two words are similar if:

- they have a lot of *features* in common
- they are used the same way
- they have a lot of the same relationships to other words

The simplifying assumption of **distributional semantics**: two words are similar if they are used in similar contexts.

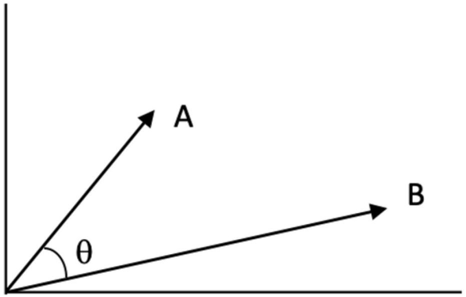
Quantifying Semantic Similarity

Word vectors are more "similar" the smaller the angle is between them.



Quantifying Semantic Similarity

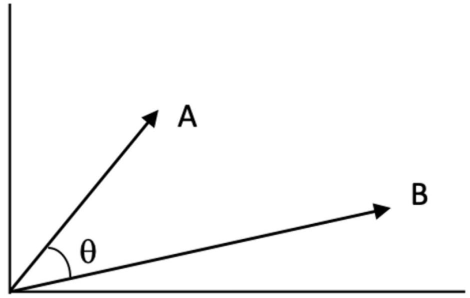
Word vectors are more "similar" the smaller the angle is between them.



$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum\limits_{i=1}^{n} A_i B_i}{\sqrt{\sum\limits_{i=1}^{n} A_i^2} \sqrt{\sum\limits_{i=1}^{n} B_i^2}}$$

Quantifying Semantic Similarity

Word vectors are more "similar" the smaller the angle is between them.

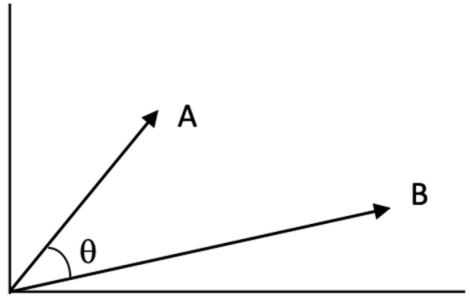


The dot product!

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

Quantifying Semantic Similarity

Word vectors are more "similar" the smaller the angle is between them.



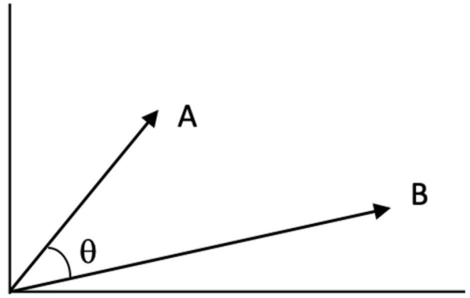
The dot product!

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

||A|| = the length of A (distance from the origin)

Quantifying Semantic Similarity

Word vectors are more "similar" the smaller the angle is between them.



Q: If A = (3, 4), what is the ||A||

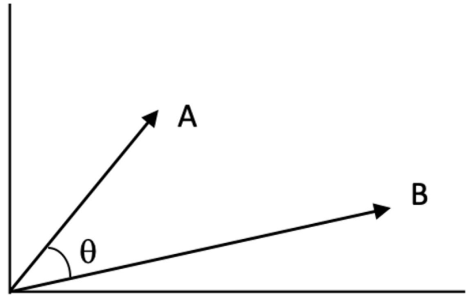
The dot product!

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

||A|| = the length of A (distance from the origin)

Quantifying Semantic Similarity

Word vectors are more "similar" the smaller the angle is between them.



The dot product!

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

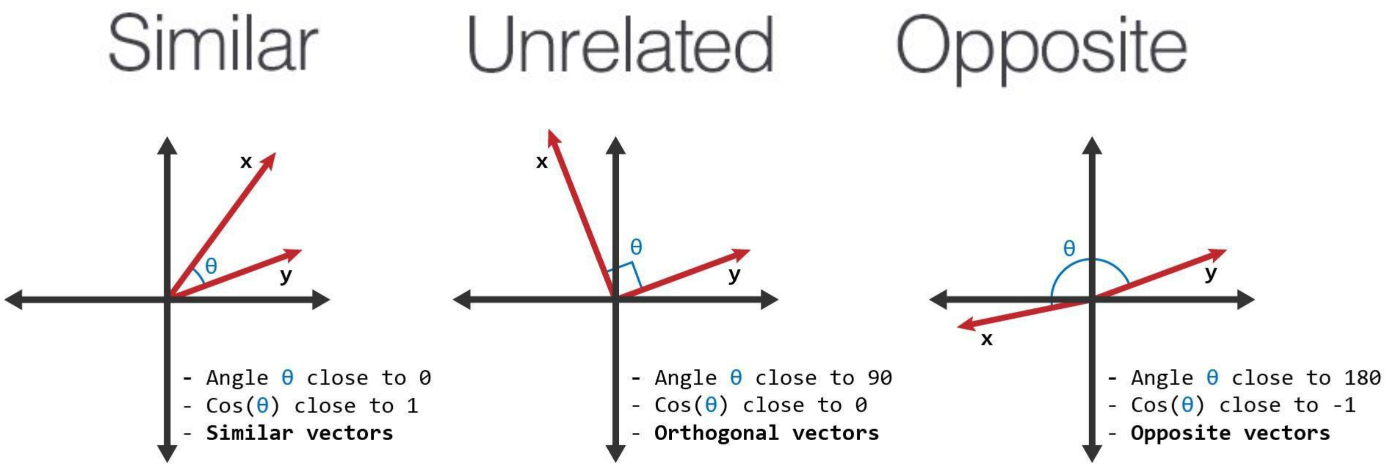
||A|| = the length of A (distance from the origin)

Q: If A = (3, 4), what is the ||A||

A: 5!

sqrt(3^2 + 4^2) = sqrt(9+16) = sqrt(25)

Quantifying Semantic Similarity



The Neural Turn in Vector Semantics (circa 2010s)

arXiv
https://arxiv.org › cs

[Efficient Estimation of Word Representations in Vector Space]

by T Mikolov · 2013 · Cited by 50129 — Abstract:We propose two novel model architectures for computing continuous vector representations of words from very large data sets.

Don't count, predict! A systematic comparison of context-counting vs. context-predicting semantic vectors

Marco Baroni and **Georgiana Dinu** and **Germán Kruszewski**
Center for Mind/Brain Sciences (University of Trento, Italy)
(marco.baroni|georgiana.dinu|german.kruszewski)@unitn.it

Great. How do we build Prediction-based Semantic Spaces?

Start out with random vectors for each word.

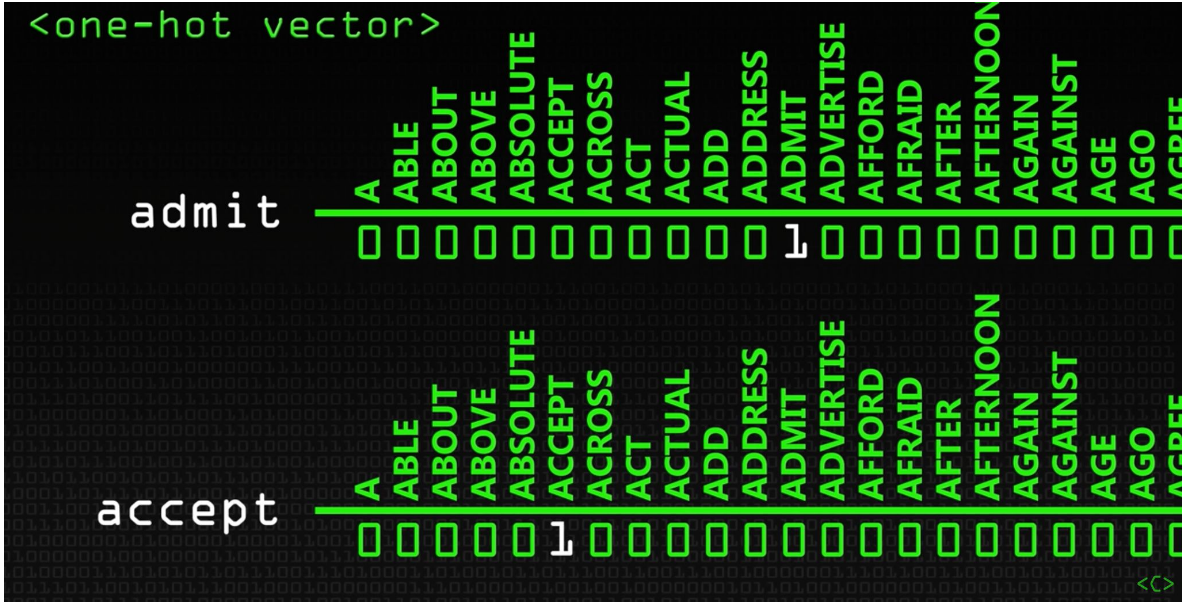
Train a machine learning model to do a task that requires knowledge of word meaning (e.g., 'guess the missing word')

Put the embedding layer between your input layer and your output layer

One-Hot Vectors

The absolute simplest way to represent words as vectors.

- each vector is as long as the vocabulary
- vector for *admit* = 1 at the index for *admit* and 0 everywhere else



problem:

- this encoding does not represent semantic relationships. every word is maximally different from every other word

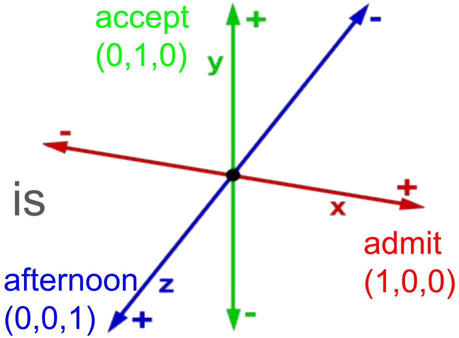


Image from video: [Computerphile on Word Embeddings]

Word2Vec

Basic idea (not exact):

- inputs: one-hot encoded words
- output: predict the next word
- "hidden layer" in between

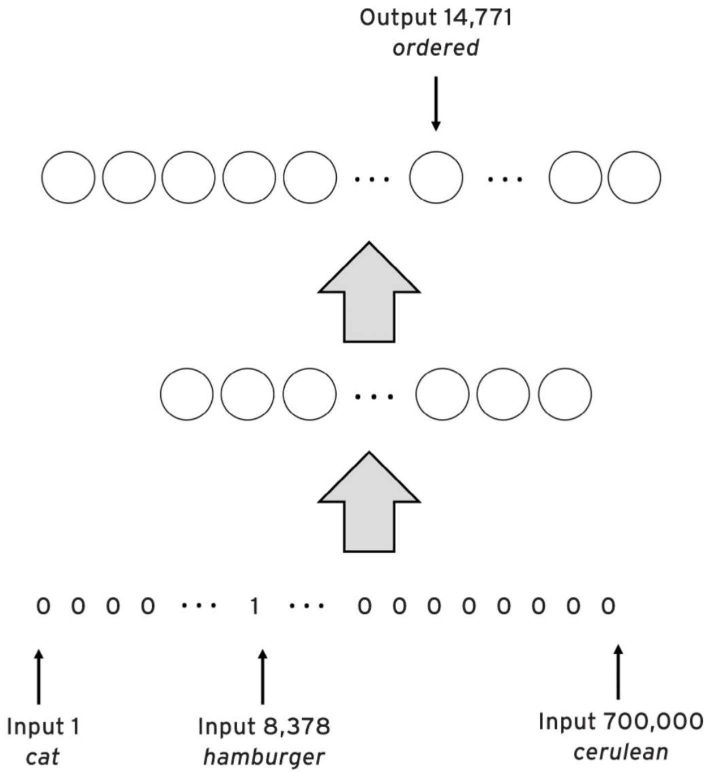


FIGURE 35: Illustration of the word2vec neural network, given the word pair (hamburger, ordered)

Image from: *AI: A Guide for Thinking Humans* (Mitchell 2019:191)

Word2Vec

Basic idea (not exact):

- inputs: one-hot encoded words
- output: predict the next word
- "hidden layer" in between

Where are the perceptrons in here?

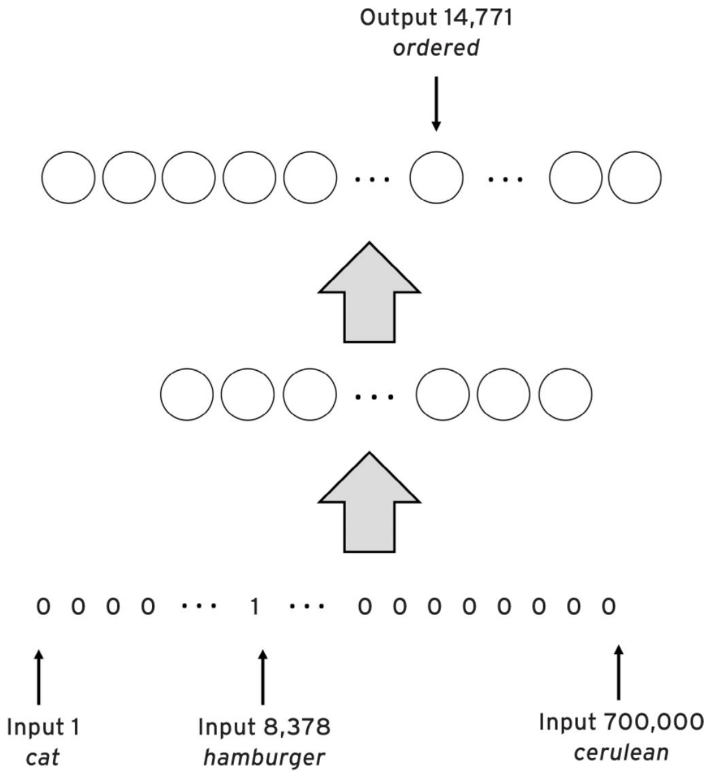


FIGURE 35: Illustration of the word2vec neural network, given the word pair (hamburger, ordered)

Image from: *AI: A Guide for Thinking Humans* (Mitchell 2019:191)

Word2Vec

Basic idea (not exact):

- inputs: one-hot encoded words
- output: predict the next word
- "hidden layer" in between

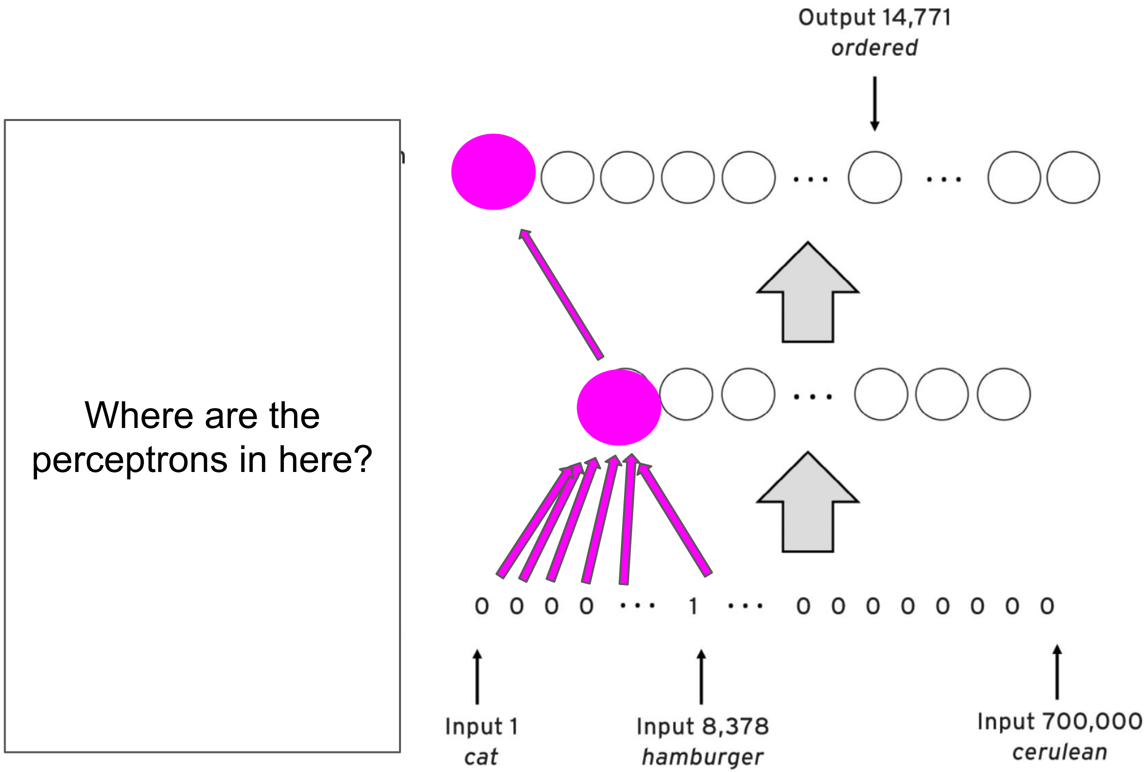


FIGURE 35: Illustration of the word2vec neural network, given the word pair (hamburger, ordered)

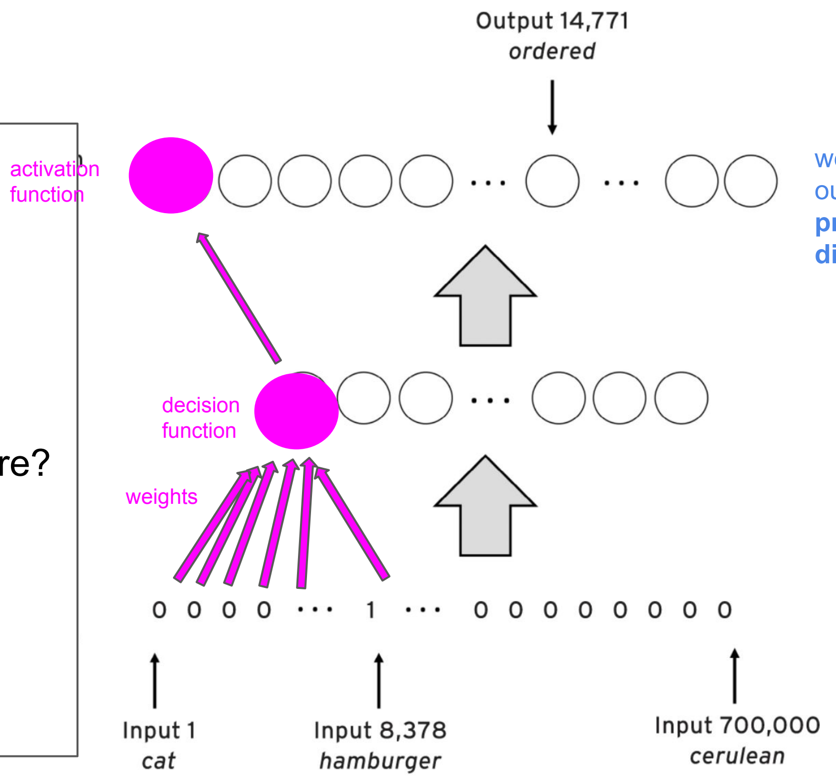
Image from: *AI: A Guide for Thinking Humans* (Mitchell 2019:191)

Word2Vec

Basic idea (not exact):

- inputs: one-hot encoded words
- output: predict the next word
- "hidden layer" in between

Where are the perceptrons in here?



we can turn the outputs into a **probability distribution**

FIGURE 35: Illustration of the word2vec neural network, given the word pair (hamburger, ordered)

Image from: *AI: A Guide for Thinking Humans* (Mitchell 2019:191)

Word2Vec

Basic idea (not exact):

- inputs: one-hot encoded words
- output: predict the next word

Size of each layer?

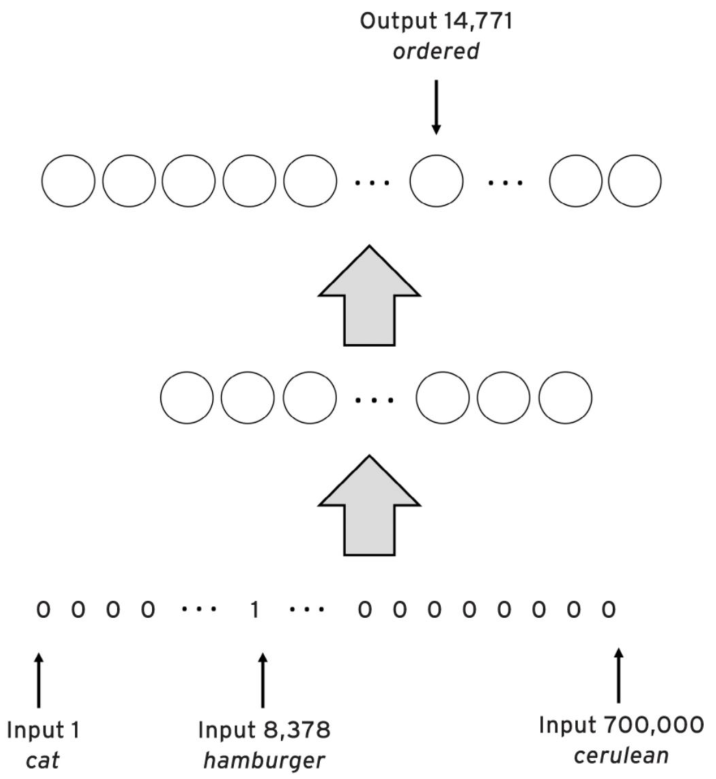


FIGURE 35: Illustration of the word2vec neural network, given the word pair (hamburger, ordered)

Image from: *AI: A Guide for Thinking Humans* (Mitchell 2019:191)

Word2Vec

Basic idea (not exact):

- inputs: one-hot encoded words
- output: predict the next word

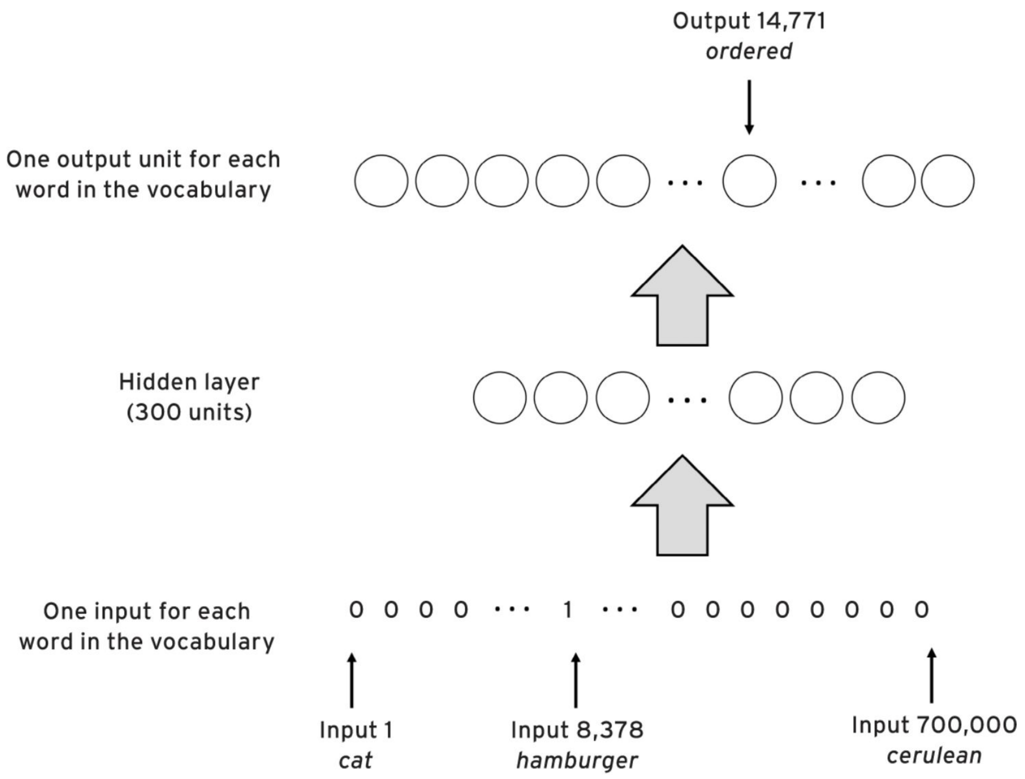


FIGURE 35: Illustration of the word2vec neural network, given the word pair (hamburger, ordered)

Image from: *AI: A Guide for Thinking Humans* (Mitchell 2019:191)

Predicting the next word: pet

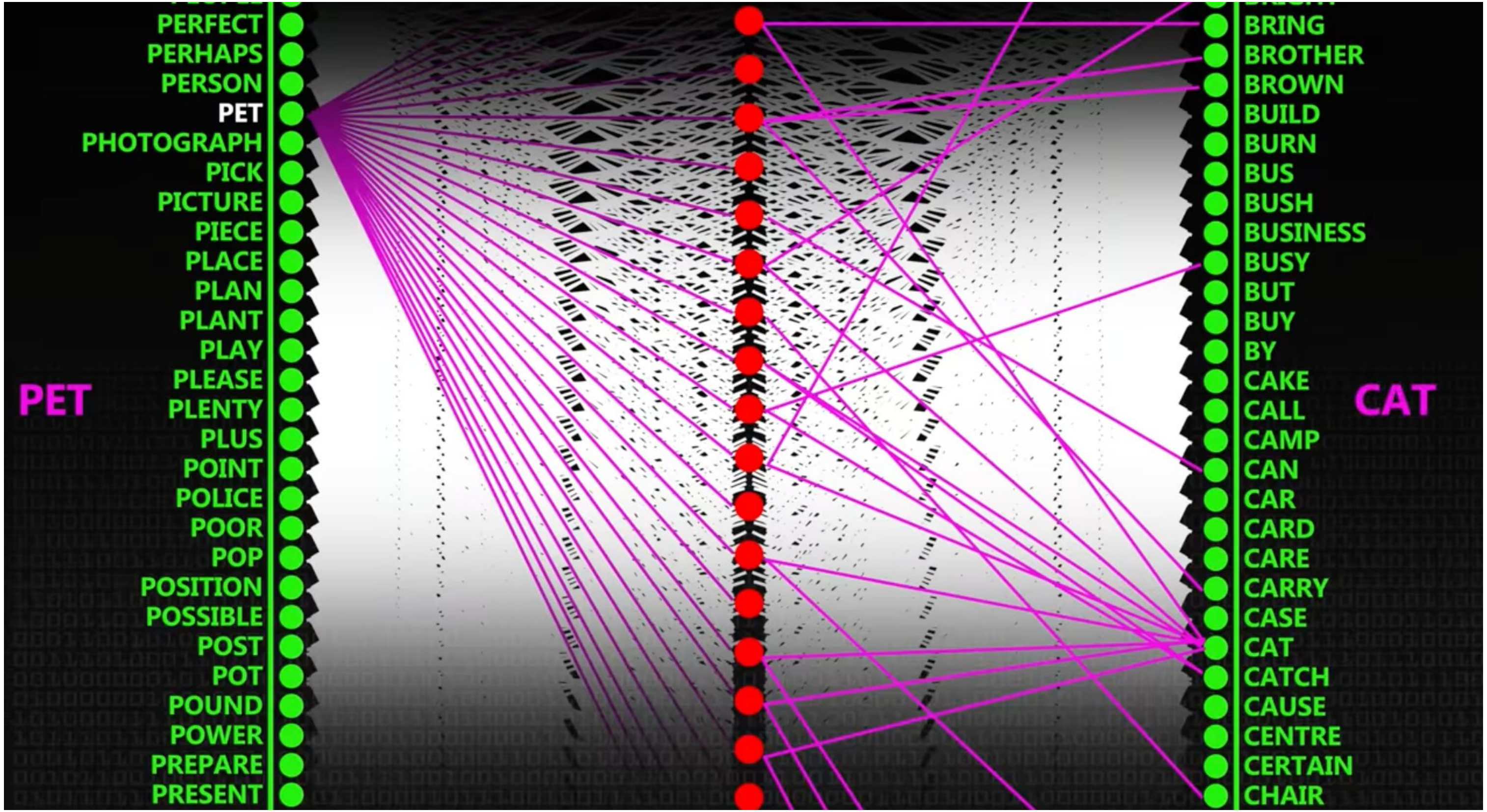


Image from video: Computerphile on Word Embeddings

Word2Vec Training Details

You will often hear word2vec described as 'skip-gram with negative sampling'. WHAT!?

The actual setup is a BINARY classification task.

Inputs: word vector, context vector

Output: 1 or 0, depending on whether the word is found in that context

Skipgram: instead of just using adjacent words as context, we use non-adjacent words (3-5 to the left and right)

Negative Sampling: our training data are all positive examples (hello class imbalance!). Negative sampling is a technique to procedurally generate false training examples.

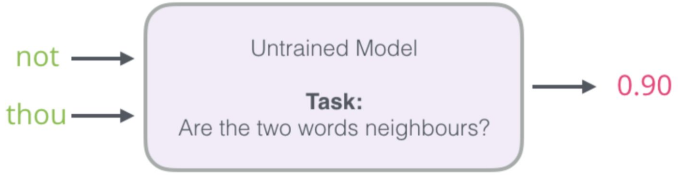
these terms describe how we create our training data

The clever trick for super-efficient training.

Change Task from

And switch it to a model that takes the input and output word, and outputs a score indicating if they're neighbors or not (0 for "not neighbors", 1 for "neighbors").

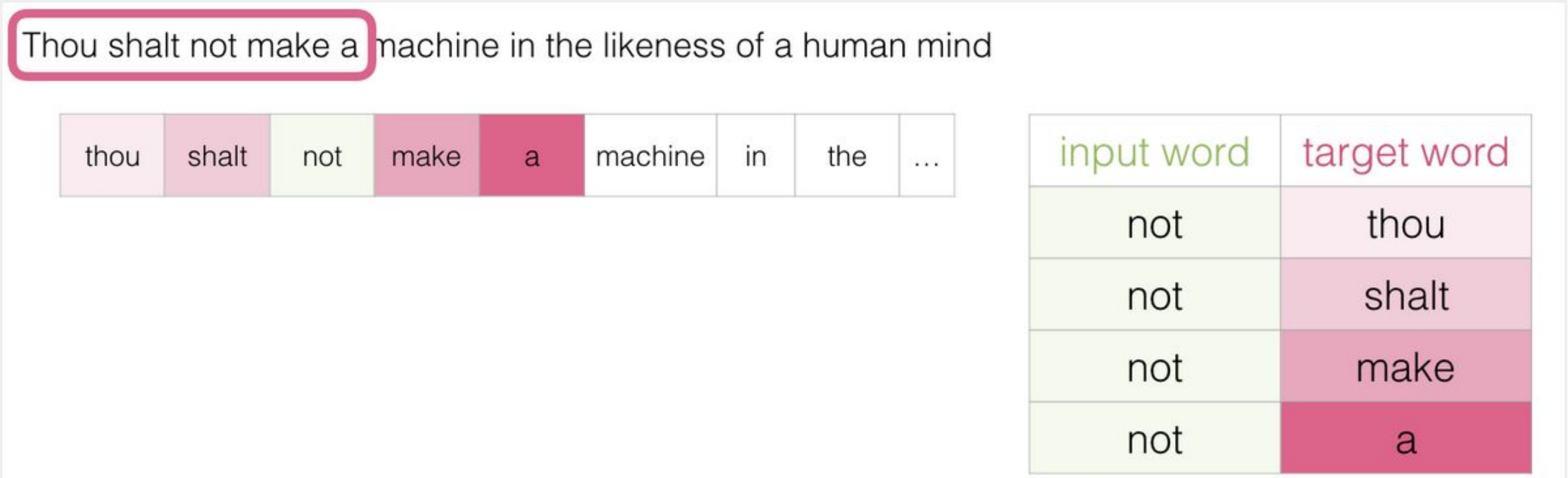
To:



Images taken from: https://jalammar.github.io/illustrated-word2vec/ (a fantastic resource)

Making the Training Data: Skipgrams

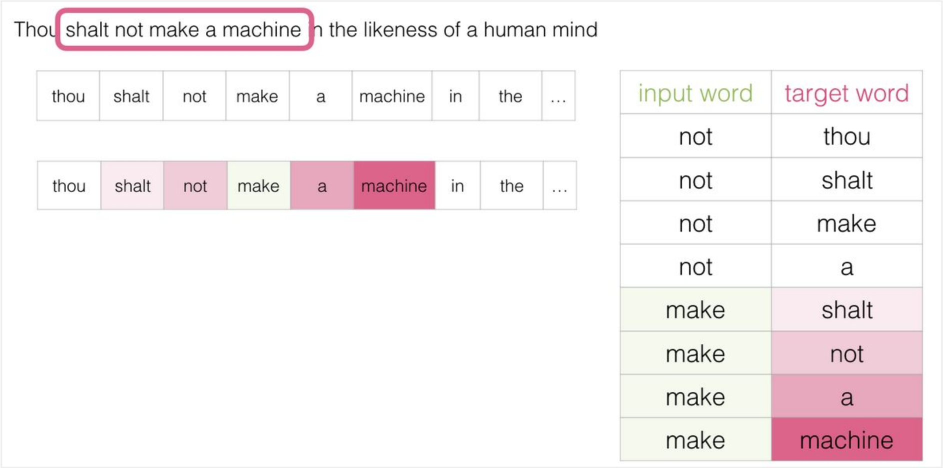
To generate training examples, we slide a context window across the corpus. The target word is in the middle, and the context words are on the sides. We add a training datapoint for each target-word context-word pair.



Images taken from: https://jalammar.github.io/illustrated-word2vec/ (a fantastic resource)

Making the Training Data: Skipgrams

We slide the window over by one and use the new target and context words to make more training examples

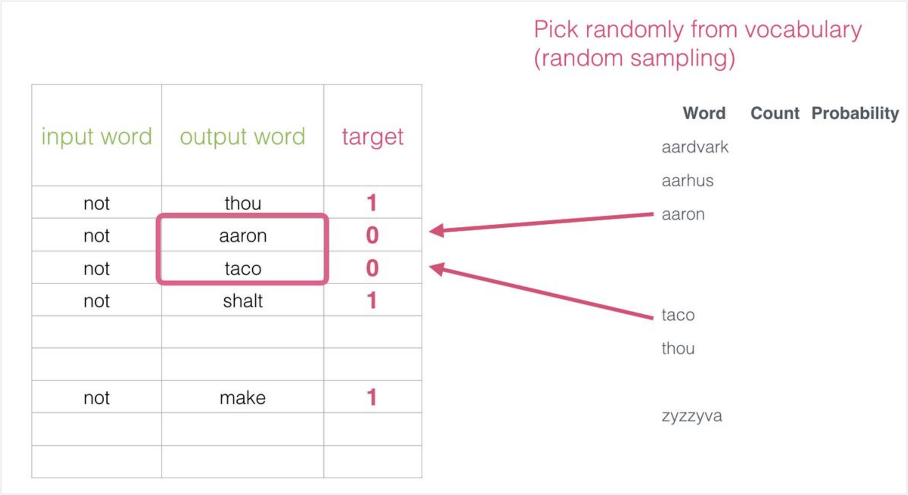


Images taken from: https://jalammar.github.io/illustrated-word2vec/ (a fantastic resource)

Making the Training Data: Negative Sampling

If we used the training dataset as is, we would only have positive examples! Talk about class imbalance.

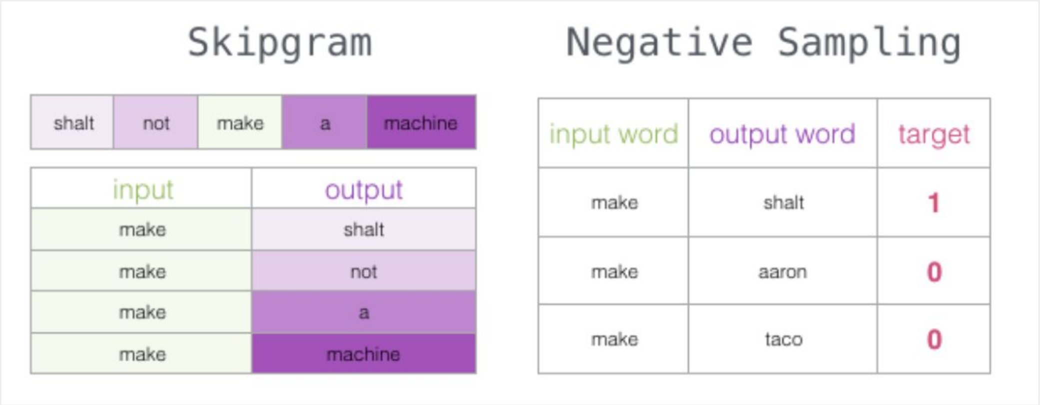
So we generate negative examples by picking random context words for each target word.



Images taken from: https://jalammar.github.io/illustrated-word2vec/ (a fantastic resource)

That's the big secret!

The thing that sounds complicated is actually just a design trick that makes word2vec look even MORE like the perceptron classification model we are already familiar with, where there are only two possible outputs, 1 and 0


Skipgram

shalt	not	make	a	machine

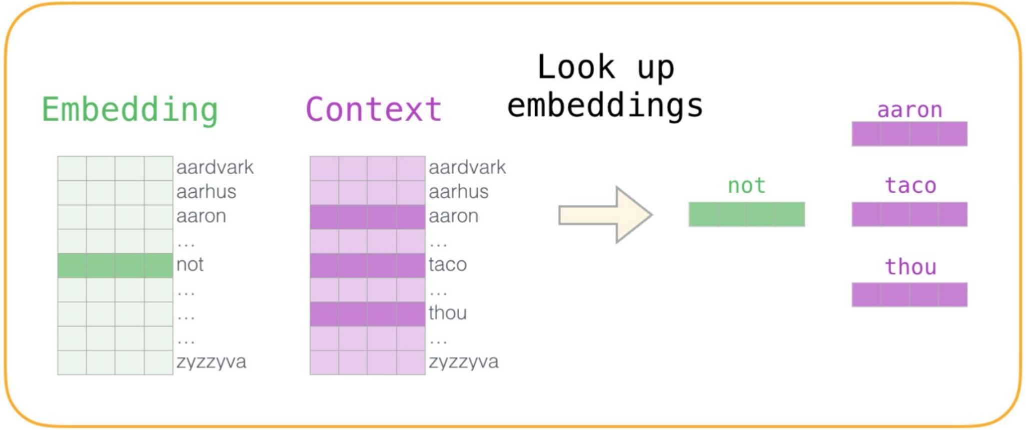
input	output
make	shalt
make	not
make	a
make	machine

Negative Sampling

input word	output word	target
make	shalt	1
make	aaron	0
make	taco	0

Images taken from: https://jalammar.github.io/illustrated-word2vec/ (a fantastic resource)

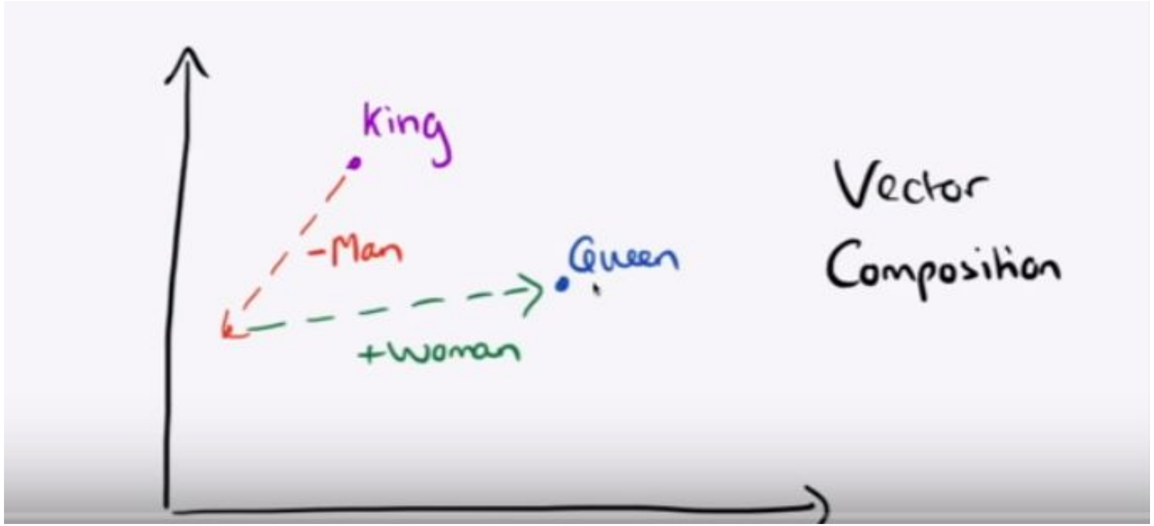
Final Word2vec Architecture



Images taken from: https://jalammar.github.io/illustrated-word2vec/ (a fantastic resource)

Analogy Solving with Vectors

- If you take the vector for **king**, subtract the vector for **man,** add the vector for **queen**, you end up at a new point in space.
- When you look around, you find that the closest neighbor in that space is the vector for **queen**



Word2Vec learns geographic relationships

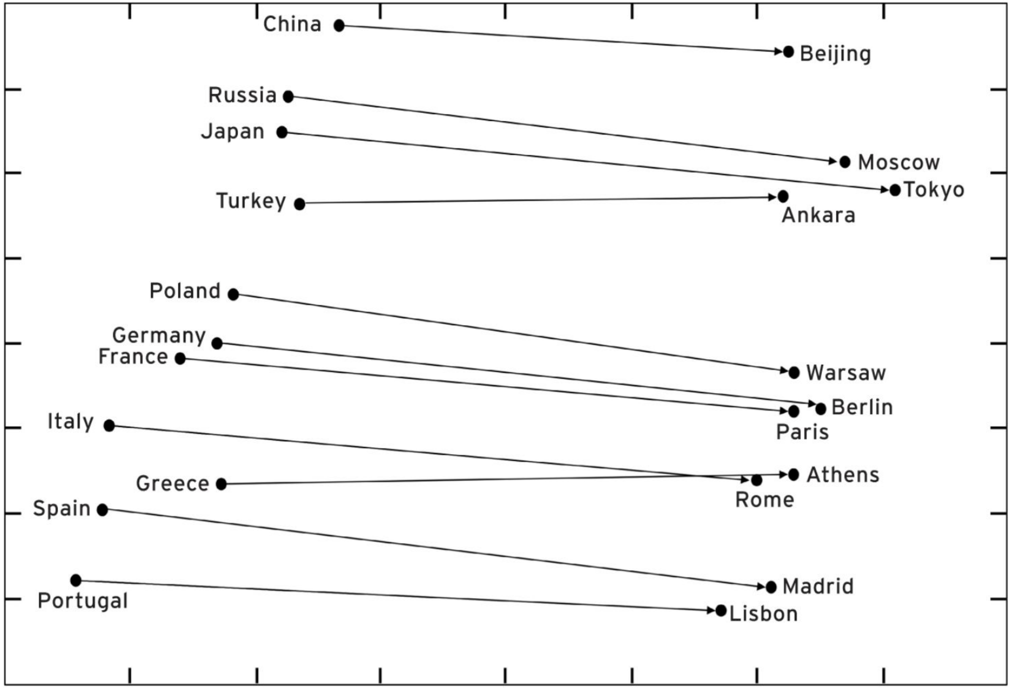


FIGURE 37: Two-dimensional representation of distances between word vectors for countries and word vectors for their capital cities

Equivalence of Deep-learning embeddings and Count-based Models

- neural networks start with random embeddings and *tune* them during training.
- Q: Why do we bother training neural networks when if can just count and factorize?
- A: with neural networks we never need to hold all of our training data in memory at once (impossible with realistic large datasets!)

Neural Word Embedding as Implicit Matrix Factorization

Omer Levy
Department of Computer Science
Bar-Ilan University
omerlevy@gmail.com

Yoav Goldberg
Department of Computer Science
Bar-Ilan University
yoav.goldberg@gmail.com

Abstract

We analyze skip-gram with negative-sampling (SGNS), a word embedding method introduced by Mikolov et al., and show that it is implicitly factorizing a word-context matrix, whose cells are the pointwise mutual information (PMI) of the respective word and context pairs, shifted by a global constant. We find that another embedding method, NCE, is implicitly factorizing a similar matrix, where each cell is the (shifted) log conditional probability of a word given its context. We show that using a sparse *Shifted Positive PMI* word-context matrix to represent