Word Embeddings
with Python and Apache Spark

Mark Roepke
SWFL Coders Meetup — April 9, 2020

Agenda

1. Who I am
2. What we're going to learn
3. What you need to know before we talk about word embeddings
4. Word embeddings
    a. What word embeddings are
    b. How word embeddings are created
    c. Why word embeddings are useful
    d. How to use word embeddings
5. What to do next...

Who I am

I’m Mark Roepke

- Moved to Fort Myers in July 2019
- Curriculum Engineer at Databricks
- Design and develop machine learning education content and certification exams
- Previously:
  - Senior Data Scientist @ 84.51 (Kroger)
  - Python Instructor at University of Cincinnati
- I like pizza and soccer
- I have a puppy named Millie

What we’re going to learn

Objectives

1. Build a general, high-level understanding of machine learning

Objectives

1. Build a general, high-level understanding of machine learning
2. Understand what word embeddings are and why they're important

Objectives

1. Build a general, high-level understanding of machine learning
2. Understand what word embeddings are and why they're important
3. Know what to use to create word embeddings

What you need to know *before* we talk about word embeddings

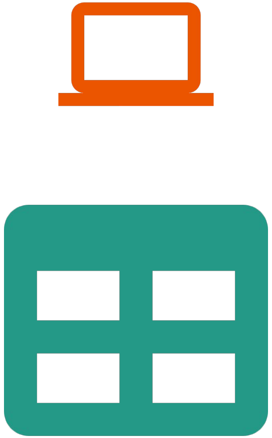
Apache Spark is a distributed data engine

- An open-source, general-purpose, distributed cluster-computing framework
- APIs in Python, Scala, Java, R, and a distributed SQL engine
- Started in 2009 by Matei Zaharia at University of California, Berkeley.
- Creators of Apache Spark went on to form Databricks in 2013, a big data and artificial intelligence company

Sometimes data is too big for one computer

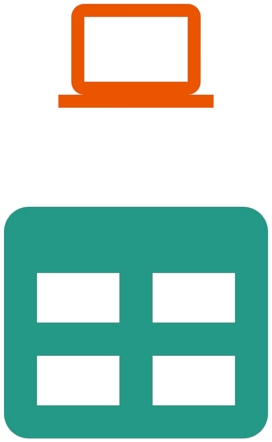
Sometimes data is too big for one computer

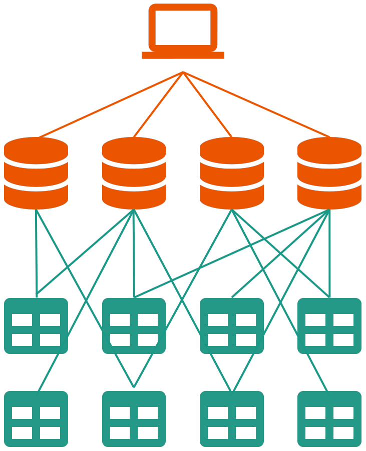
Big data does not fit on **disk** or in **memory** of a single computer



Sometimes data is too big for one computer

Big data does not fit on disk or in memory of a single computer

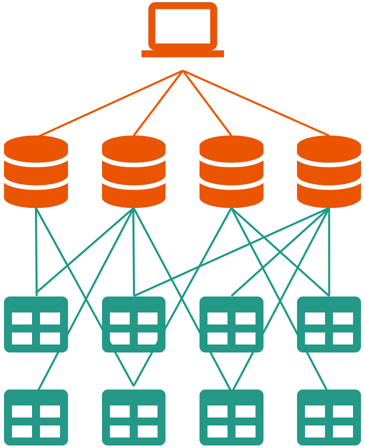




But we can partition data into smaller pieces to help it fit disk or in memory of a cluster of computers

Sometimes data is too big for one computer

Big data does not fit on disk or in memory of a single computer



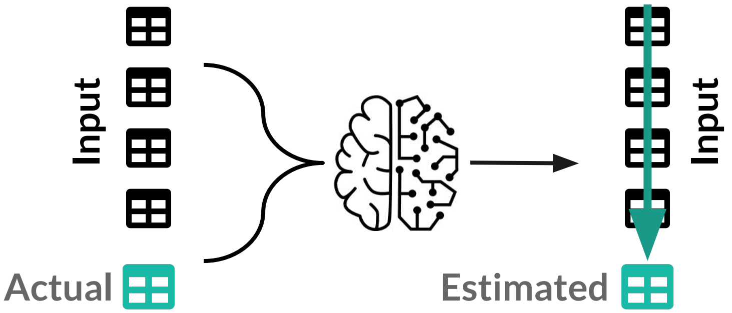
But we can partition data into smaller pieces to help it fit disk or in memory of a cluster of computers

Most of machine learning fits into two categories

Most of machine learning fits into two categories

Supervised Machine Learning

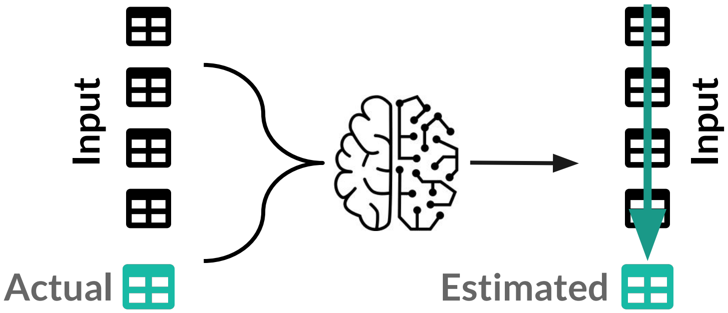
Learning to predict or estimate a target based on input data



Most of machine learning fits into two categories

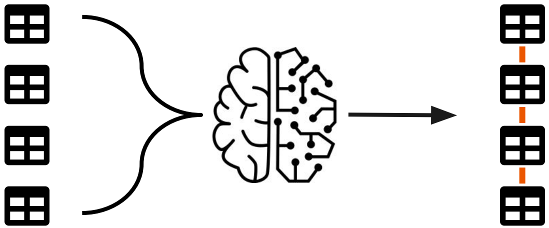
Supervised Machine Learning

Learning to predict or estimate a target based on input data



Unsupervised Machine Learning

Learning relationships between data without trying to predict or estimate anything



Supervised and unsupervised machine learning can be used to solve real-world problems

Supervised and unsupervised machine learning can be used to solve real-world problems

Predicting **book sales** based on the author's **previous sales, book price, release date**

Supervised and unsupervised machine learning can be used to solve real-world problems

Predicting **book sales** based on the author's **previous sales, book price, release date**

Clustering customers into actionable groups based on their **spend behavior, visit frequency, common modality**

Supervised and unsupervised machine learning can be used to solve real-world problems

Predicting **book sales** based on the author's **previous sales, book price, release date**

Clustering customers into actionable groups based on their **spend behavior, visit frequency, common modality**

Identifying **what calls are fraudulent** based on **who is making the call, length-of-call, whether or not somebody answered**

Supervised and unsupervised machine learning can be used to solve real-world problems

Predicting **book sales** based on the author's **previous sales, book price, release date**

Clustering customers into actionable groups based on their **spend behavior, visit frequency, common modality**

Identifying **what calls are fraudulent** based on **who is making the call, length-of-call, whether or not somebody answered**

What do all of these problems have in common?

But what if we don’t have numeric data?

But what if we don’t have numeric data?

Word Clouds :)

Counting how often words appear in a document

But what if we don’t have numeric data?

Word Clouds :)

Counting how often words appear in a document

Sentiment Analysis

Comparing the amount of positive words and negative words in a document

But what if we don’t have numeric data?

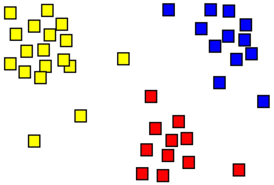
Word Clouds :)

Counting how often words appear in a document

Sentiment Analysis

Comparing the amount of positive words and negative words in a document

Topic Modeling



Identifying which words occur in the same documents most frequently

But what if we don't have numeric data?

Word Clouds :)

Counting how often words appear in a document

Sentiment Analysis

Comparing the amount of positive words and negative words in a document

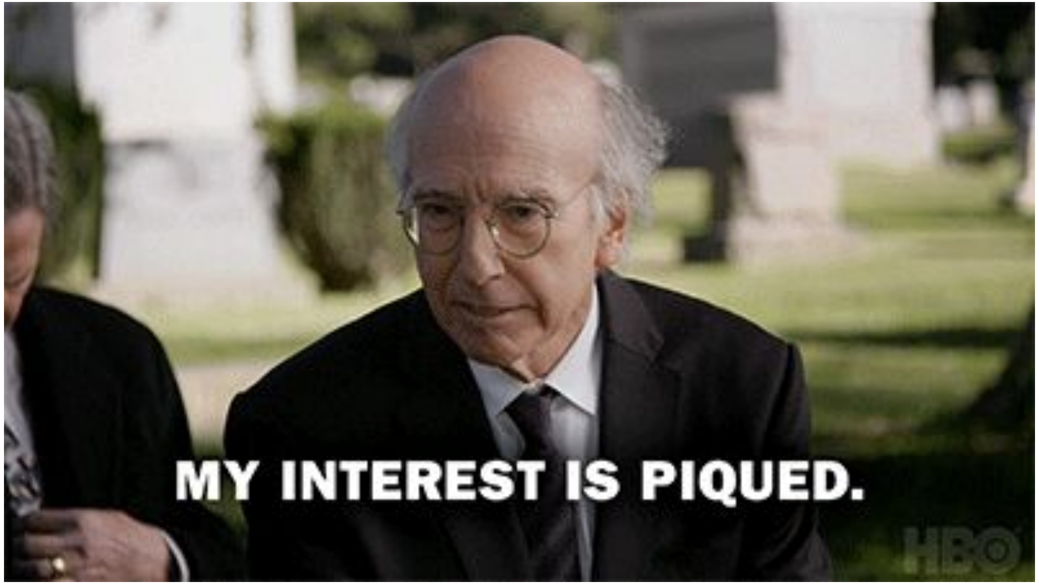
Topic Modeling

Identifying which words occur in the same documents most frequently

LIMITING

What if we could *turn text into numbers*?

What if we could *turn text into numbers*?



Word embeddings

What word embeddings are

Word embeddings are vectors that represent words

For this reason, they're also called word vectors...

```
        add = [0.12 -1.45 … 2.54]
   subtract = [0.14 -1.61 … 0.11]
grandmother = [3.56  2.01 … -1.67]
grandfather = [3.55  0.41 … -1.67]
 margherita = [2.55  1.89 …  0.56]
  margarita = [1.42  4.10 … -0.04]
```

Word embeddings are vectors that represent words

For this reason, they're also called word vectors...

```
        add = [0.12 -1.45 … 2.54]
   subtract = [0.14 -1.61 … 0.11]
grandmother = [3.56  2.01 … -1.67]
grandfather = [3.55  0.41 … -1.67]
 margherita = [2.55  1.89 …  0.56]
  margarita = [1.42  4.10 … -0.04]
```

Note that each word only has a single vector.

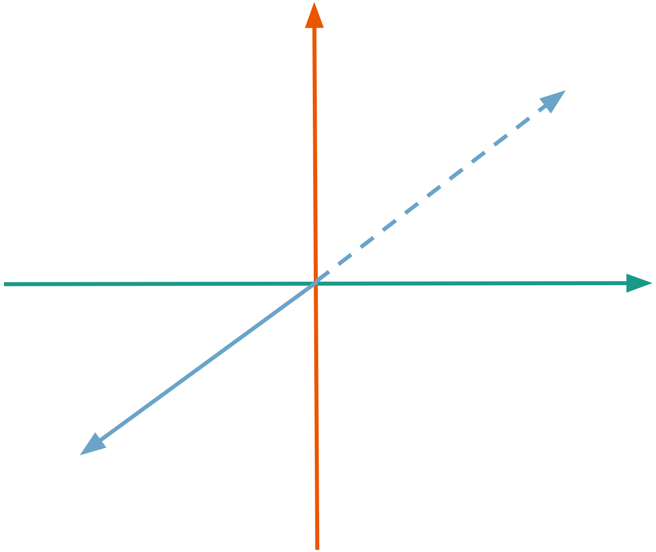
Each word embedding is unique to that word

But all word embeddings are in the same linear space...

$$
\begin{aligned}
\textbf{add} &= [add_1\ add_2\ \dots\ add_n] \\
\textbf{subtract} &= [subtract_1\ subtract_2\ \dots\ subtract_n] \\
\textbf{grandmother} &= [grandmother_1\ grandmother_2\ \dots\ grandmother_n] \\
\textbf{grandfather} &= [grandfather_1\ grandfather_2\ \dots\ grandfather_n] \\
\textbf{margherita} &= [margherita_1\ margherita_2\ \dots\ margherita_n] \\
\textbf{margarita} &= [margarita_1\ margarita_2\ \dots\ margarita_n]
\end{aligned}
$$

These word vectors can be graphed in linear space

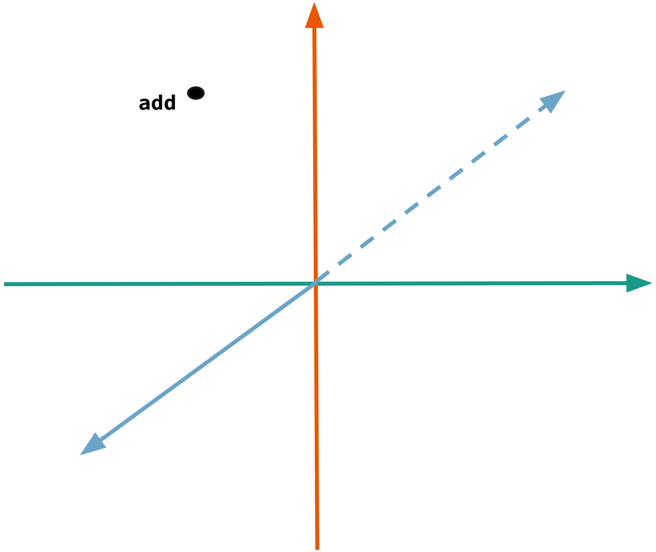
Just like graphing points on a plot, but there usually are a lot more dimensions



These word vectors can be graphed in linear space

Just like graphing points on a plot, but there usually are a lot more dimensions

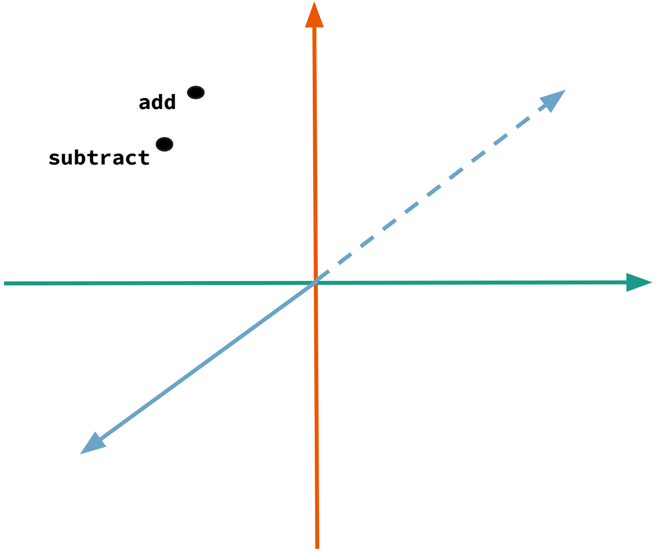
```
add = [0.12 -1.45 …  2.54]
```



These word vectors can be graphed in linear space

Just like graphing points on a plot, but there usually are a lot more dimensions

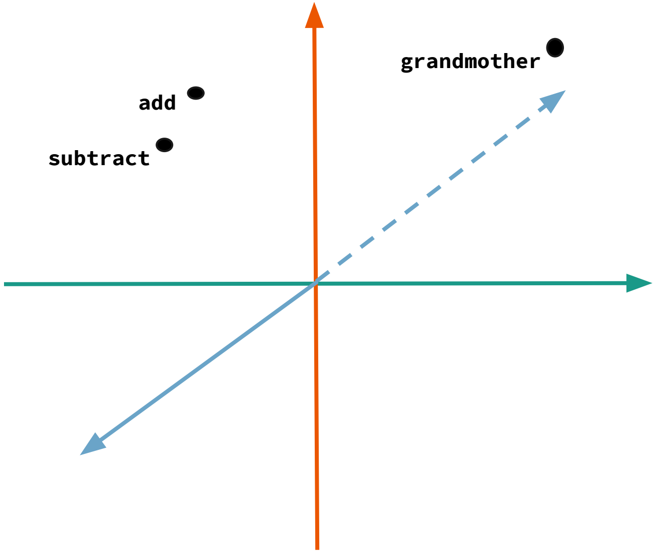
```
     add = [0.12 -1.45 …  2.54]
subtract = [0.14 -1.61 …  0.11]
```



These word vectors can be graphed in linear space

Just like graphing points on a plot, but there usually are a lot more dimensions

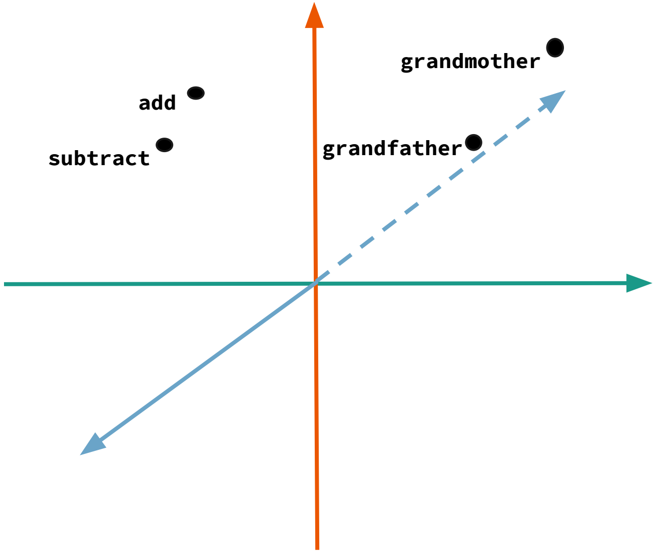
```
       add = [0.12 -1.45 …  2.54]
  subtract = [0.14 -1.61 …  0.11]
grandmother = [3.56  2.01 … -1.67]
```



These word vectors can be graphed in linear space

Just like graphing points on a plot, but there usually are a lot more dimensions

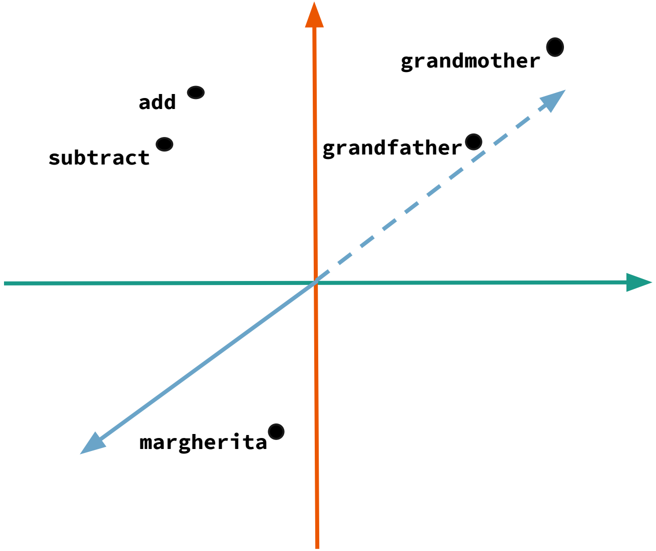
```
       add = [0.12 -1.45 …  2.54]
  subtract = [0.14 -1.61 …  0.11]
grandmother = [3.56  2.01 … -1.67]
grandfather = [3.55  0.41 … -1.67]
```



These word vectors can be graphed in linear space

Just like graphing points on a plot, but there usually are a lot more dimensions

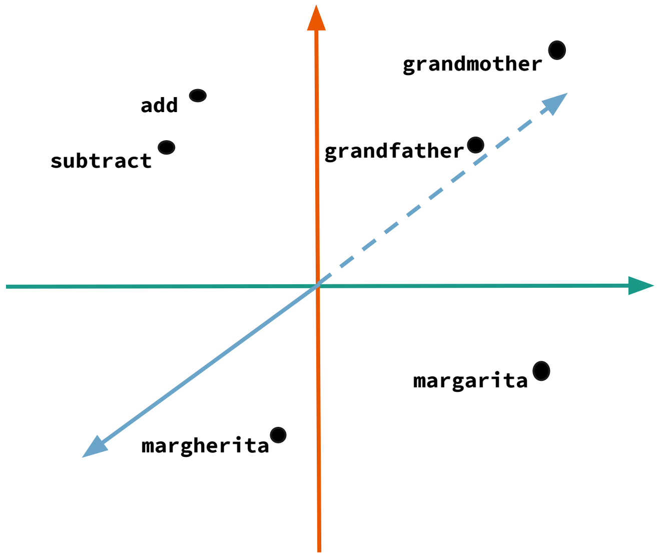
```
        add = [0.12 -1.45 …  2.54]
   subtract = [0.14 -1.61 …  0.11]
grandmother = [3.56  2.01 … -1.67]
grandfather = [3.55  0.41 … -1.67]
 margherita = [2.55  1.89 …  0.56]
```



These word vectors can be graphed in linear space

Just like graphing points on a plot, but there usually are a lot more dimensions

```
       add = [0.12 -1.45 …  2.54]
  subtract = [0.14 -1.61 …  0.11]
grandmother = [3.56  2.01 … -1.67]
grandfather = [3.55  0.41 … -1.67]
 margherita = [2.55  1.89 …  0.56]
  margarita = [1.42  4.10 … -0.04]
```



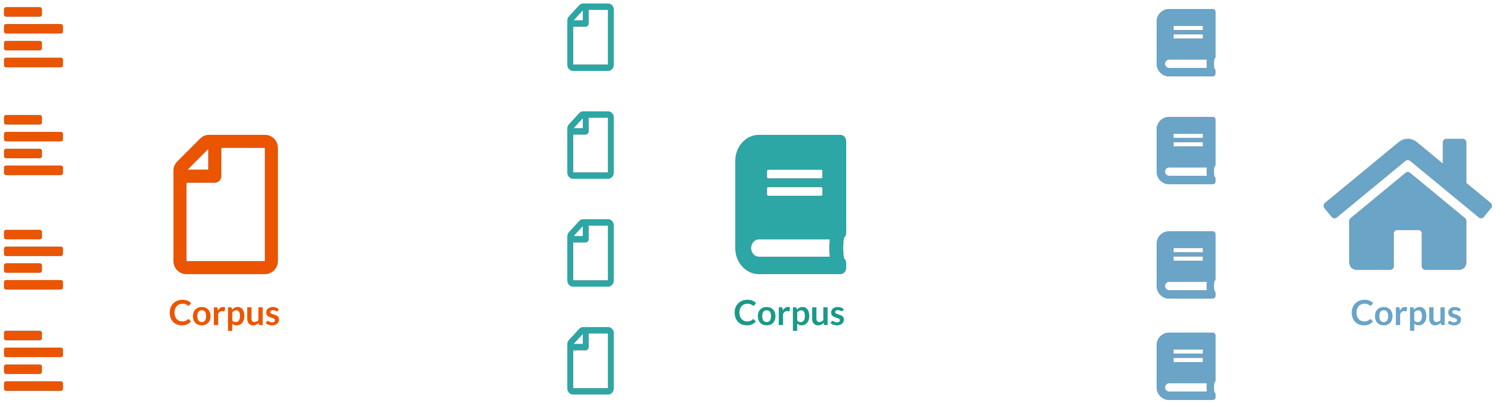
How word embeddings are created

First, we need words...

Word embeddings are created from something called a text corpus

First, we need words...

Word embeddings are created from something called a text corpus



A text corpus is a collection of text documents...

We then organize our data based on how words appear in each document within the corpus

We then organize our data based on how words appear in each document within the corpus

Document	add	subtract	grandmother	...
1	1	1	0	...
2	0	0	0	...
3	0	0	1	...
4	0	0	1	...
5	1	1	0	...
6	0	1	0	...
...	...	...	...	...

We then organize our data based on how words appear in each document within the corpus

Document	add	subtract	grandmother	...
1	1	1	0	...
2	0	0	0	...
3	0	0	1	...
4	0	0	1	...
5	1	1	0	...
6	0	1	0	...
...	...	...	...	...

This is called a document-term matrix, and it can get **REALLY** big

Word embeddings are mapped from that large space

In other words, they're reduced from a large/sparse corpus-level representation to a small/dense embedding-level representation

Word embeddings are mapped from that large space

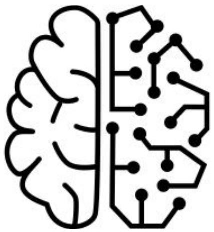
In other words, they're reduced from a large/sparse corpus-level representation to a small/dense embedding-level representation

Document	add	subtract	grandmother	...
1	1	1	0	...
2	0	0	0	...
3	0	0	1	...
4	0	0	1	...
5	1	1	0	...
6	0	1	0	...
...	...	...	...	...

Word embeddings are mapped from that large space

In other words, they're reduced from a large/sparse corpus-level representation to a small/dense embedding-level representation

Document	add	subtract	grandmother	...
1	1	1	0	...
2	0	0	0	...
3	0	0	1	...
4	0	0	1	...
5	1	1	0	...
6	0	1	0	...
...	...	...	...	...

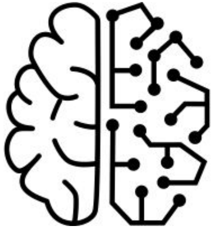


```
        add = [0.12 -1.45 … 2.54]
   subtract = [0.14 -1.61 … 0.11]
grandmother = [3.56  2.01 … -1.67]
```

Word embeddings are mapped from that large space

In other words, they're reduced from a large/sparse corpus-level representation to a small/dense embedding-level representation

Document	add	subtract	grandmother	...
1	1	1	0	...
2	0	0	0	...
3	0	0	1	...
4	0	0	1	...
5	1	1	0	...
6	0	1	0	...
...	...	...	...	...



```
       add = [0.12 -1.45 … 2.54]
  subtract = [0.14 -1.61 … 0.11]
grandmother = [3.56  2.01 … -1.67]
```

THIS IS A FORM OF UNSUPERVISED LEARNING

This reduction is the machine learning…

There are two primary families of algorithms to create word embeddings

This reduction is the machine learning...

There are two primary families of algorithms to create word embeddings

Global matrix factorization

- Factors a term-feature matrix from a document-term matrix of a text corpus
- Based on the **"Distributional Hypothesis of Linguistics"**
- Uses **singular value decomposition** to create embeddings for each term
- An example is **latent semantic analysis**

This reduction is the machine learning...

There are two primary families of algorithms to create word embeddings

Global matrix factorization

- Factors a term-feature matrix from a document-term matrix of a text corpus
- Based on the **"Distributional Hypothesis of Linguistics"**
- Uses **singular value decomposition** to create embeddings for each term
- An example is **latent semantic analysis**

TIRED

This reduction is the machine learning...

There are two primary families of algorithms to create word embeddings

Global matrix factorization

- Factors a term-feature matrix from a document-term matrix of a text corpus
- Based on the **"Distributional Hypothesis of Linguistics"**
- Uses **singular value decomposition** to create embeddings for each term
- An example is **latent semantic analysis**

TIRED

Local context window

- Predicts meaning and context of words inside a rolling window of words in each document
- Based on the **"Distributional Hypothesis of Linguistics"**
- Uses **neural networks** to create embeddings for each term
- An example is **word2vec**

This reduction is the machine learning...

There are two primary families of algorithms to create word embeddings

Global matrix factorization

- Factors a term-feature matrix from a document-term matrix of a text corpus
- Based on the **"Distributional Hypothesis of Linguistics"**
- Uses **singular value decomposition** to create embeddings for each term
- An example is **latent semantic analysis**

TIRED

Local context window

- Predicts meaning and context of words inside a rolling window of words in each document
- Based on the **"Distributional Hypothesis of Linguistics"**
- Uses **neural networks** to create embeddings for each term
- An example is **word2vec**

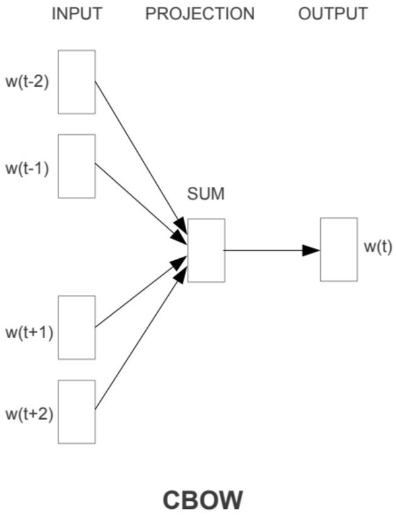
WIRED

There are two approaches to word2vec...

They each have their own strengths and optimal use cases

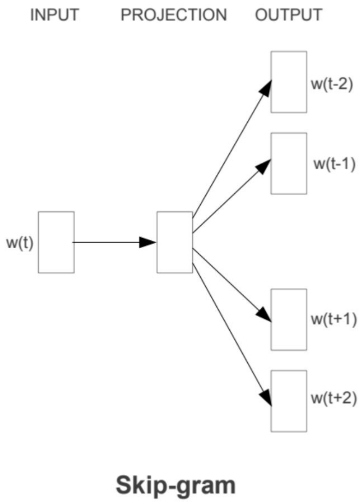
Continuous bag-of-words

- Learns embeddings by learning to predict the focus word using the words around the focus word
- Word order and word proximity are irrelevant
- Really fast, decent results with common words



There are two approaches to word2vec...

They each have their own strengths and optimal use cases



Skip-gram

- Learns embeddings by learning to predict the words around a focus word using the focus word
- More heavily weights predictions of context words that are closer to the focus word
- Slower, great results with most words

DEMO 1

Why word embeddings are useful

Word embeddings have revolutionized text-based machine learning...

They provide a mathematically meaningful representation of text

There are two high-level important points on why word embeddings are useful:

Word embeddings have revolutionized text-based machine learning...

They provide a mathematically meaningful representation of text

There are two high-level important points on why word embeddings are useful:

1. **The information they contain:** word embeddings learn from how the word is used, so they contain both **semantic** and **syntactic** information

Word embeddings have revolutionized text-based machine learning...

They provide a mathematically meaningful representation of text

There are two high-level important points on why word embeddings are useful:

1. **The information they contain:** word embeddings learn from how the word is used, so they contain both **semantic** and **syntactic** information
2. **A numerical representation of that information:** numerically based **analytical tools** and methodologies **can now be used on words and language**

Word embeddings have revolutionized text-based machine learning...

They provide a mathematically meaningful representation of text

There are two high-level important points on why word embeddings are useful:

1. **The information they contain:** word embeddings learn from how the word is used, so they contain both **semantic** and **syntactic** information
2. **A numerical representation of that information:** numerically based **analytical tools** and methodologies **can now be used on words and language**

The real power of word embeddings is in the **combination of these two points**

Word embeddings contain semantic information...

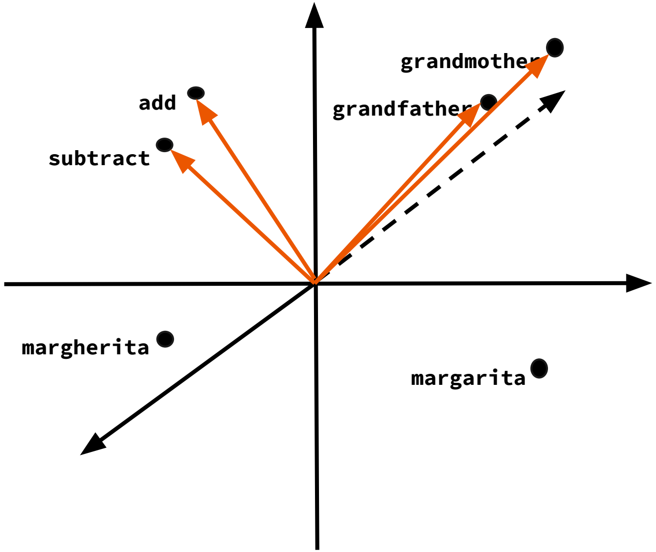
So we can compare word meaning using common numerical algebra on embeddings

Word embeddings contain semantic information...

So we can compare word meaning using common numerical algebra on embeddings

Similarity

- Recall that each word embedding is within the same vector space
- And word embeddings contain semantic information
- So logic says that **similar words will have similar vectors**



DEMO 2

Word embeddings contain semantic information...

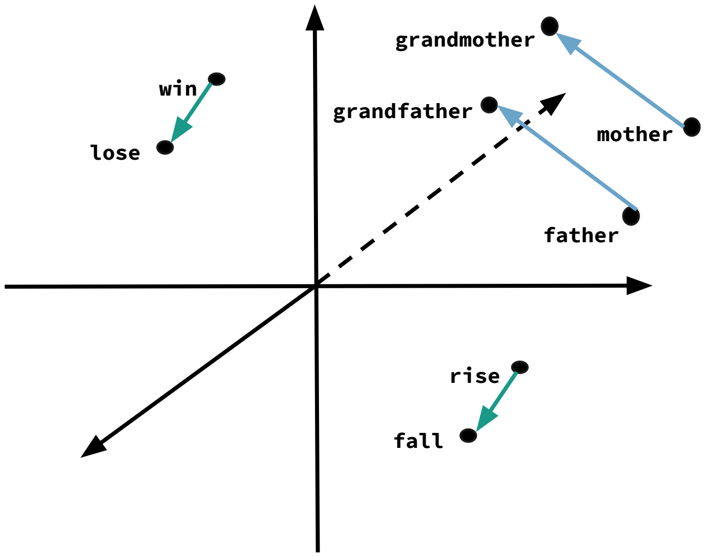
So we can compare word meaning using common numerical algebra on embeddings

Word embeddings contain semantic information...

So we can compare word meaning using common numerical algebra on embeddings

Analogy

- Analogies are comparisons between things for the purpose of explanation
- Analogous relationships can be represented algebraically:
  - *Mother - Grandmother = Father - Grandfather*
  - *Win - Lose = Rise - Fall*
- The differences between these words can be represented by their **difference vectors**



DEMO 3

Word embeddings contain syntactic information...

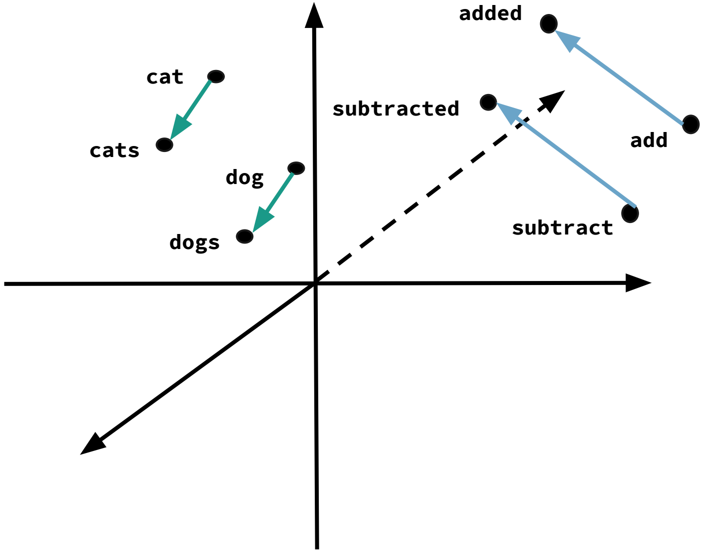
So we can compare word grammar using common numerical algebra on embeddings

Word embeddings contain syntactic information...

So we can compare word grammar using common numerical algebra on embeddings

Analogy

- Analogous comparisons also work for grammatical explanations like verb tense and plurality
- For example :
  - *Add - Added = Subtract - Subtracted*
  - *Cat - Cats = Dog - Dogs*
- The average of the resulting difference vectors represent **ideas** like **present → past tense**, and **singular → plural**



DEMO 4

How to use word embeddings

So drawing lines and learning plurals is great, but...

*The power of word embeddings is **not** in linear space similarities and analogies*

- These exercises exemplify the intelligence of word embeddings, but **not the practical power**

So drawing lines and learning plurals is great, but...

*The power of word embeddings is **not** in linear space similarities and analogies*

- These exercises exemplify the intelligence of word embeddings, but **not the practical power**
- Machine learning is useful when **it can be applied to solve problems at scale**

So drawing lines and learning plurals is great, but...

*The power of word embeddings is **not** in linear space similarities and analogies*

- These exercises exemplify the intelligence of word embeddings, but **not the practical power**
- Machine learning is useful when **it can be applied to solve problems at scale**
- Think back to our example machine learning problems:

Predicting book sales based on the author's previous sales, book price, release date

Classifying customers into actionable groups based on their spend behavior, visit frequency, common modality

Identifying what calls are fraudulent based on who is making the call, length-of-call, whether or not somebody answered

Have word embeddings improved our ability to do any of these things?

Remember that word embeddings are vectors

This means they can be fed to standard numeric machine learning algorithms

Remember that word embeddings are vectors

This means they can be fed to standard numeric machine learning algorithms

- When we use word embeddings as features for machine learning, **we expand our data set to be far larger and far more rich**:

Predicting book sales based on the author's previous sales, book price, release date

Classifying customers into actionable groups based on their spend behavior, visit frequency, common modality

Identifying what calls are fraudulent based on who is making the call, length-of-call, whether or not somebody answered

Remember that word embeddings are vectors

This means they can be fed to standard numeric machine learning algorithms

- When we use word embeddings as features for machine learning, **we expand our data set to be far larger and far more rich**:

Predicting book sales based on the author's previous sales, book price, release date, and **the actual words of the book (story, topics, etc.)**

Classifying customers into actionable groups based on their spend behavior, visit frequency, common modality, and **their reviews and comments**

Identifying what calls may be fraudulent based on who is making the call, length-of-call, whether or not somebody answered, and **the actual words said on the call**

This opens up a world of possibilities in what can be solved using machine learning.

Things to be aware of

But word embeddings aren’t perfect…

There a couple key limitations with word embeddings

But word embeddings aren't perfect...

There a couple key limitations with word embeddings

There can only be one vector for each word, but many words have multiple meanings.

DEMO 5

But word embeddings aren't perfect...

There a couple key limitations with word embeddings

There can only be one vector for each word, but many words have multiple meanings.

Most embedding algorithms require a LARGE amount of data to successfully learn embeddings

DEMO 6

Word embeddings are the foundation of modern natural language processing

More advanced NLP tasks embrace the principles of word embeddings

Word embeddings are the foundation of modern natural language processing

More advanced NLP tasks embrace the principles of word embeddings

Doc2Vec

- Uses word embeddings to create document-level vectors
- Can be used to summarize and compare text documents -- even when they have different lengths
- Extremely useful in applied settings

Word embeddings are the foundation of modern natural language processing

More advanced NLP tasks embrace the principles of word embeddings

Doc2Vec

- Uses word embeddings to create document-level vectors
- Can be used to summarize and compare text documents -- even when they have different lengths
- Extremely useful in applied settings

Seq2Seq

- Accepts a sequence of words as input to a neural network and returns a sequence of words
- Uses an “encoder” similar to create embeddings for the sequence of words and a “decoder” to return words based on their embeddings
- Built for machine translation, but also useful for summarization, conversational modeling, and even image captioning

What to do next...

If you want to learn more about general machine learning...

- Introductory
  - *Introduction to Statistical Learning (James, Witten, Hastie, Tibshirani)*
  - *Introduction to Machine Learning with Python (Muller, Guido)*
- Intermediate
  - *Hands-on Machine Learning with R (Boehmke, Greenwell)*
  - *Hands-on Machine learning with Scikit-learn and TensorFlow (Geron)*
- Deep Learning
  - *Deep Learning with Python (Chollet)*

If you want to learn more about natural language processing...

- Introductory
  - *Text Mining with R (Silge, Robinson)*
  - KDNuggets Preprocessing Tutorial
- Word Embeddings
  - “A Survey of Word Embedding Evaluation Methods” (Bakarov)
  - “Efficient Estimation of Word Representations in Vector Space” (Mikolov)
  - “Distributed Representations of Sentences and Documents” (Le, Mikolov)
  - “GloVe: Global Vectors for Word Representation” (Pennington, Socher, Manning)
- Modern NLP
  - Stanford CS224N: Natural Language Processing with Deep Learning
- Curious
  - *The Distributional Hypothesis of Linguistics* (Firth)

Questions

Resources

1. Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. "Efficient estimation of word representations in vector space." ICLR Workshop, 2013.
2. Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. "Distributed representations of words and phrases and their compositionality." NIPS, 2013.
3. Scott Deerwester, Susan Dumais, George Furnas, Thomas Landauer and Richard Harshman. "Indexing by latent semantic analysis." Journal of the American Society For Information Science, 1990. 41, 391-407.
4. Thomas Landauer and Susan Dumais. "A solution to Plato's problem: the latent semantic analysis theory of acquisition, induction, and representation of knowledge." Psychological Review, 1997. Vol. 1M. No. 2, 211-240
5. John Rupert Firth. "A Synopsis of linguistic theory, 1930-55." 1968.
6. Quoc Le and Tomas Mikolov. "Distributed representations of sentences and documents." ICML, 2014.
7. Jeffrey Pennington, Richard Socher, and Christopher Manning. "GloVe: Global Vectors for Word Representation." 2014.
8. Amir Bakarov. "A Survey of Word Embeddings Evaluation Methods." 2018.