# Computational Social Science

## Word embeddings I

Dr. Thomas Davidson

Rutgers University

March 9, 2022

# Plan

1. Course updates
2. Word embeddings
3. Contextualized embeddings

# Course updates

- Project proposals due 5pm Friday
  - Submit form on Canvas
- Mid-semester evaluation
  - Please complete by Friday

# Recap

- Language models are probabilistic models for language understanding and generation
  - Auto-complete, search suggestions, etc.
- N-gram language models
  - Probabilistic models predicting words based on previous N words used

# Language models

## Neural language models

- Recent advances in both the availability of large corpora of text *and* the development of neural network models have resulted in new ways of computing language models.
- By using machine-learning techniques, particularly neural networks, to train a language model, we can construct better vector representations

# Word embeddings

## Intuition

- We use the context in which a word occurs to train a language model
  - The model learns by viewing millions of short snippets of text (e.g 5-grams)
- This model outputs a vector representation of each word in $k$-dimensional space, where $k << |V|$.
  - Like LSA, these vectors are *dense*
    - Each element contains a real number and can be positive or negative

# Word embeddings

## Word2vec: Skip-gram and continuous bag-of-words (CBOW)

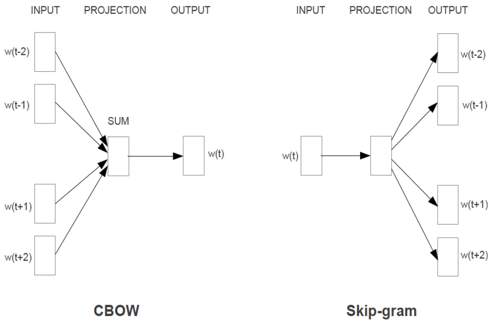



Figure 1: New model architectures. The CBOW architecture predicts the current word based on the context, and the Skip-gram predicts surrounding words given the current word.

# Word embeddings

## Word2vec: CBOW intuition

- We start with a string where the focal word is known, but hidden from the model, but we know the context within a window, in this case two words on either side of the focal word
  - e.g. "The cat ? on the", where ? = "sat"
- The model is trained using a process called *negative sampling*, where it must distinguish between the true sentence and "fake" sentences where ? is replaced with another token.
  - Each "guess" allows the model to begin to learn the correct answer
- By repeating this for millions of text snippets the model is able to "learn" which words go with which contexts

# Word embeddings

## Word2vec: Skip-gram intuition

- We start with a string where the focal is known, but the context within the window is hidden
  - e.g. "$?_1$ $?_2$ sat $?_3$ $?_4$"
- The model tests different words in the vocabulary to predict the missing context words
  - Each "guess" allows the model to begin to learn the correct answer
- By repeating this for millions of text snippets the model is able to "learn" which contexts go with which words

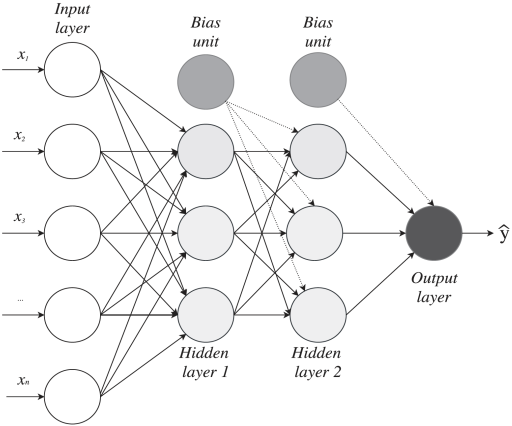
# Word embeddings

### Word2vec: Model

- Word2vec uses a shallow neural-network to predict a word given a context (CBOW) or a context given a word (skip-gram)
  - But we do not care about the prediction itself, only the *weights* the model learns
- It is a self-supervised method since the model is able to update using the correct answers
  - e.g. In CBOW the model knows when the prediction is wrong and updates the weights accordingly

# Word embeddings

## Word2vec: Feed-forward neural network




This example shows a two-layer feed-forward neural network.

# Word embeddings

## Word2vec: Estimation procedure

- Batches of strings are passed through the network
  - After each batch, weights are updated using *back-propagation*
    - The model updates its weights in the direction of the correct answer (the objective is to improve predictive accuracy)
    - Optimization via *stochastic gradient descent*

# Word embeddings

### Vector representations of words

- Each word is represented as a vector of weights learned by the neural network
  - Each element of this vector represents how strongly the word activates a neuron in the hidden layer of the network
  - This represents the association between the word and a given dimension in semantic space

# Word embeddings

## Distributional semantics

- The word vectors in the embedding space capture information about the context in which words are used
  - Words with similar meanings are situated close together in the embedding space
- This is consistent with Ludwig Wittgenstein's *use theory of meaning*
  - "the meaning of a word is its use in the language", *Philosophical Investigations* (1953)
- *Distributional semantics* is the theory that the meaning of a word is derived from its context in language use
  - "You shall know a word by the company it keeps", J.R. Firth (1957)

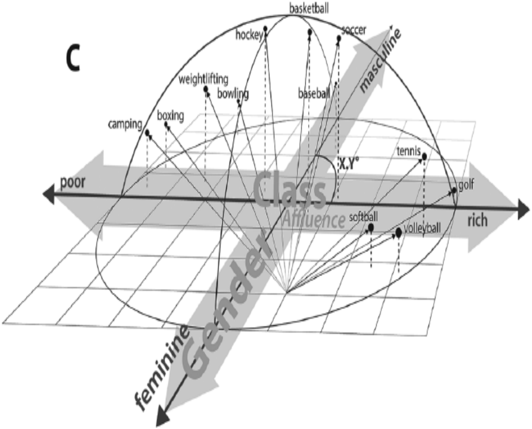
# Word embeddings

### Analogies

- The most famous result from the initial word embedding paper is the ability of these vectors to capture analogies:
  - $king - man + woman \approx queen$
  - $Madrid - Spain + France \approx Paris$

# Word embeddings

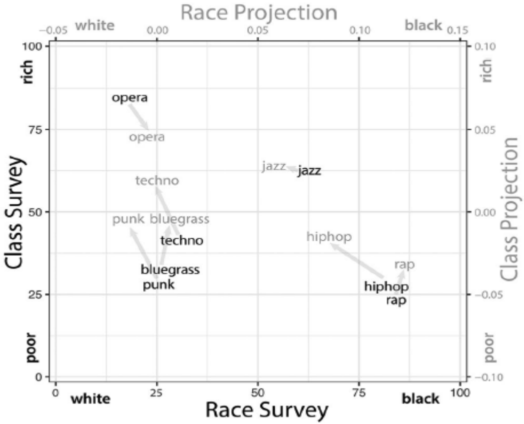
## Applications: Understanding social class




Kozlowski, Austin C., Matt Taddy, and James A. Evans. 2019. "The Geometry of Culture: Analyzing the Meanings of Class through Word Embeddings." American Sociological Review, September, 000312241987713. https://doi.org/10.1177/0003122419877135.

# Word embeddings

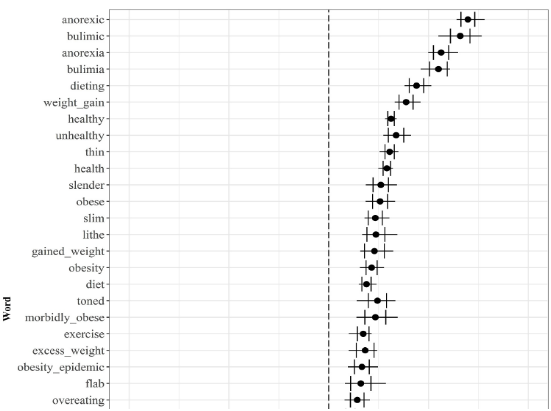
## Applications: Understanding social class



# Word embeddings

## Applications: Understanding cultural schematas

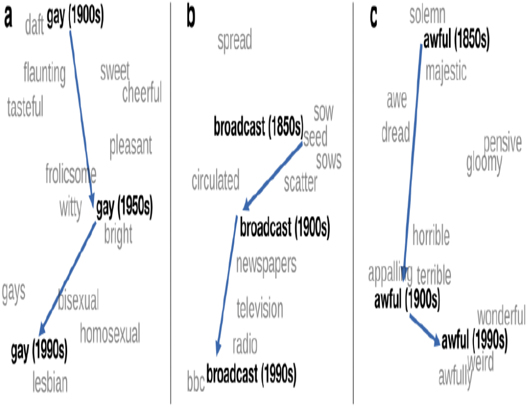
Figure 4: Gendering of Obesity-Related Words




Arseniev-Koehler, Alina, and Jacob G. Foster. 2020. "Machine Learning as a Model for Cultural Learning: Teaching an Algorithm What It Means to Be Fat." Preprint. *SocArXiv*. https://doi.org/10.31235/osf.io/c9yj3.

# Word embeddings

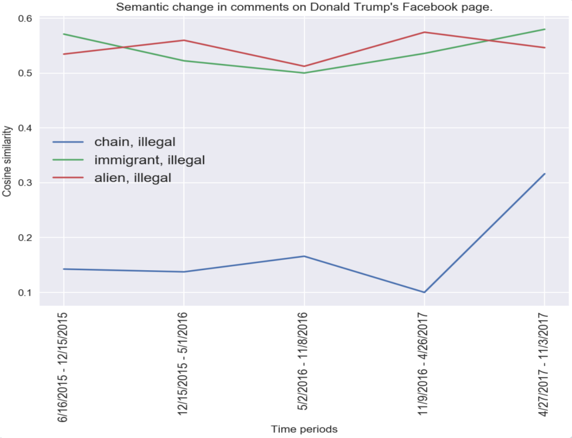
## Applications: Semantic change




Hamilton, William L., Jure Leskovec, and Dan Jurafsky. 2016. "Diachronic Word Embeddings Reveal Statistical Laws of Semantic Change." In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, 1489–1501.

# Word embeddings

## Applications: Semantic change




Davidson 2017, unpublished.

# Word embeddings

## Pre-trained word embeddings

- In addition to `word2vec` there are several other popular variants including `GloVe` and `Fasttext`
  - Pre-trained embeddings are available to download so you don't necessarily need to train your own
- When to train your own embeddings?
  - You have a large corpus of text ($>$ tens of thousands of documents)
  - You think the underlying language model / data generating process may differ from that represented by existing corpora
    - e.g. A word embedding trained on newspapers may not be very useful for studying Twitter since online language use differs substantially from written news media

# Word embeddings

## Loading a corpus: Politicians' tweets

```
library(stringr)
library(tidyverse)

df <- read_csv("data/politics_twitter.csv")
unique(df$screen_name)
```

```
##  [1] "JoeBiden"         "KamalaHarris"     "SpeakerPelosi"    "BernieSa
##  [5] "AOC"              "SenSchumer"       "LeaderMcConnell"  "LindseyG
##  [9] "tedcruz"          "Mike_Pence"       "MarshaBlackburn"  "lisamurk
```

# Word embeddings

## Word embeddings in R

We're going to use the library `word2vec`. The library is a R wrapper around a `C++` library. The the original library can be found here and the R version wrapper here.

```
#install.packages("word2vec")
library(word2vec)
set.seed(08901) # random seed
```

# Word embeddings

### Word embeddings in R

Let's train a model on the Jane Austen texts.

```
model <- word2vec::word2vec(x = tolower(df$text),
                  type="cbow",# continuous bag of words
                  dim=100, # 100-dim output vectors
                  window = 3L, # window size 3
                  iter = 10L, # train over 100 iterations
                  negative= 10L,# 10 negative samples for each positive
                  min_count = 10L) # Drop words less than 3 times
```

# Word embeddings

### Getting embeddings for words

We can use the `predict` function to find the nearest words to a given term.

```r
predict(model, c("nation"), type = "nearest", top_n = 10)
```

```
## $nation
##     term1        term2 similarity rank
## 1  nation      country  0.8992831    1
## 2  nation      america  0.7705463    2
## 3  nation      society  0.7266870    3
## 4  nation commonwealth  0.7032403    4
## 5  nation      economy  0.7011756    5
## 6  nation     nation's  0.6952459    6
## 7  nation        globe  0.6904517    7
## 8  nation    democracy  0.6868933    8
## 9  nation     founders  0.6858405    9
## 10 nation     progress  0.6839540   10
```

# Word embeddings

## Getting embeddings for words

We can also get the embedding matrix and try to do reasoning by analogy. We can see the model doesn't perform very well. This is because it has only been trained on a small corpus of text.

```
emb <- as.matrix(model)
vector <- emb["king", ] - emb["man", ] + emb["woman", ]
predict(model, vector, type = "nearest", top_n = 10)
```

```
##       term similarity rank
## 1     king  0.9651592    1
## 2     born  0.8487223    2
## 3       jr  0.8469628    3
## 4   luther  0.8390354    4
## 5   martin  0.8351078    5
## 6      tom  0.8141624    6
## 7   ahmaud  0.8131991    7
## 8   taylor  0.8062732    8
## 9  breonna  0.8055453    9
## 10     lee  0.8000150   10
```

# Word embeddings

### Getting embeddings for words

Let's try another example.

```r
vector <- emb["chicago", ] - emb["illinois", ] + emb["california", ]
predict(model, vector, type = "nearest", top_n = 10)
```

```
##          term similarity rank
## 1  california  0.9420524    1
## 2     chicago  0.9406440    2
## 3    colorado  0.8025264    3
## 4          ca  0.7691315    4
## 5   locations  0.7656763    5
## 6      storms  0.7627865    6
## 7        2014  0.7626160    7
## 8  highlights  0.7601315    8
## 9          37  0.7560194    9
## 10       2016  0.7541974   10
```

# Word embeddings

### Loading a pre-trained embedding

Let's try another example. I downloaded a pre-trained word embedding model trained on a much larger corpus of English texts. The file is 833MB in size. Following the documentation we can load this model into R.

```
model.pt <- read.word2vec(file = "data/sg_ns_500_10.w2v", normalize = T
```

# Word embeddings

**Similarities**

Find the top 5 most similar terms to “love” in the embedding space.

```
predict(model.pt, c("nation"), type = "nearest", top_n = 5)
```

# Word embeddings

**Similarities**

Find the top 5 most similar terms to "hamlet" in the embedding space.

```
predict(model.pt, c("hamlet"), type = "nearest", top_n = 5)
```

# Word embeddings

### Re-trying the analogy test

Let's re-try the analogy test. We still don't go great but now queen is in the top 5 results.

```
emb <- as.matrix(model.pt)
vector <- emb["king", ] - emb["man", ] + emb["woman", ]
predict(model.pt, vector, type = "nearest", top_n = 10)
```

# Word embeddings

### Re-trying the analogy test

Let's try another analogy. The correct answer is second. Not bad.

```
vector <- emb["madrid", ] - emb["spain", ] + emb["france", ]
predict(model.pt, vector, type = "nearest", top_n = 10)
```

# Word embeddings

### Re-trying the analogy test

Let's try another slightly more complex analogy. Not bad overall.

```
vector <- (emb["new", ] + emb["jersey", ])/2 - emb["trenton", ] + emb["
predict(model.pt, vector, type = "nearest", top_n = 10)
```

# Word embeddings

## Representing documents

Last week we focused on how we could represent documents using the rows in the DTM. So far we have just considered how words are represented in the embedding space. We can represent a document by averaging over its composite words.

```
descartes <- (emb["i", ] +
                emb["think", ] +
                emb["therefore", ] +
                emb["i", ] +
                emb["am", ])/5
predict(model.pt, descartes, type = "nearest", top_n = 10)
```

# Word embeddings

### Representing documents

The package has a function called `doc2vec` to do this automatically. This function includes an additional scaling factor (see documentation) so the results are slightly different.

```
descartes <- doc2vec(model.pt, "i think therefore i am")
predict(model.pt, descartes, type = "nearest", top_n = 10)
```

# Word embeddings

### Visualizing high-dimensional embeddings in low-dimensional space

- There are various algorithms available for visualizing word-embeddings in low-dimensional space
  - `PCA`, `t-SNE`, `UMAP`
- There are also browser-based interactive embedding explorers
  - See this example on the Tensorflow website

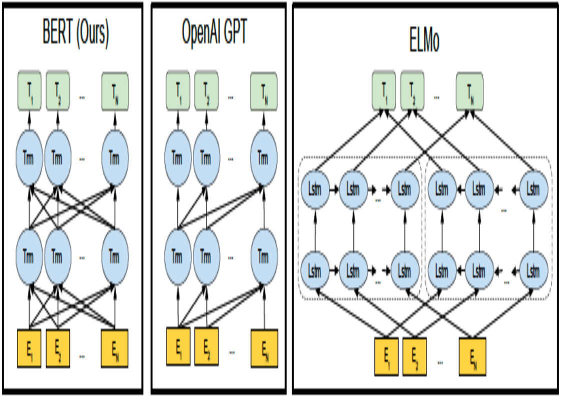
# Contextualized embeddings

## Limitations of existing approaches

- Word2vec and other embedding methods run into issues when dealing with *polysemy*
  - e.g. The vector for "crane" will be learned by averaging across different uses of the term
    - A bird
    - A type of construction equipment
    - Moving one's neck
  - "She had to crane her neck to see the crane perched on top of the crane".
- New methods have been developed to allow the vector for "crane" to vary according to different contexts
- The intuition here is that we want to take more context into account when constructing vectors

# Contextualized embeddings

## Architectures



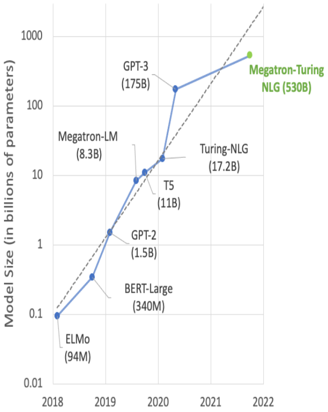

Source: Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. “BERT: Pre-Training of Deep Bidirectional Transformers for Language Understanding.” In Proceedings of NAACL-HLT 2019, 4171–86. ACL.

# Contextualized embeddings

## Methodological innovations

- More complex, deeper neural networks
  - *Attention* mechanisms, *LSTM* architecture, *bidirectional transformers*
- Optimization over multiple tasks (not just a simple prediction problem like Word2vec)
- Character-level tokenization and embeddings
- Much more data and enormous compute power required
  - e.g. BERT trained on a 3.3 billion word corpus over 40 epochs, taking over 4 days to train on 64 TPU chips (each chip costs ~$10k).

# Very large language models




See Nvidia blog on Megatron-Turing NLG.

# Contextualized embeddings

## On the Dangers of Stochastic Parrots: Can Language Models Be Too Big?

Emily M. Bender*
ebender@uw.edu
University of Washington
Seattle, WA, USA

Timnit Gebru*
timnit@blackinai.org
Black in AI
Palo Alto, CA, USA

Angelina McMillan-Major
aymm@uw.edu
University of Washington
Seattle, WA, USA

Shmargaret Shmitchell
shmargaret.shmitchell@gmail.com
The Aether

**ABSTRACT**

The past 3 years of work in NLP have been characterized by the development and deployment of ever larger language models, especially for English. BERT, its variants, GPT-2/3, and others, most recently Switch-C, have pushed the boundaries of the possible both through architectural innovations and through sheer size. Using these pretrained models and the methodology of fine-tuning them for specific tasks, researchers have extended the state of the art

alone, we have seen the emergence of BERT and its variants [39, 70, 74, 113, 146], GPT-2 [106], T-NLG [112], GPT-3 [25], and most recently Switch-C [43], with institutions seemingly competing to produce ever larger LMs. While investigating properties of LMs and how they change with size holds scientific interest, and large LMs have shown improvements on various tasks (§2), we ask whether enough thought has been put into the potential risks associated with developing them and strategies to mitigate these risks.

We first consider environmental risks. Echoing a line of recent

Bender, Emily M, Timnit Gebru, Angelina McMillan-Major, and Shmargaret Shmitchell. 2021. "On the Dangers of Stochastic Parrots: Can Language Models Be Too Big?" In Conference on Fairness, Accountability, and Transparency (*FAccT '21*).

# Contextualized embeddings

## Fine-tuning

- One of the major advantages of BERT and other approaches is the ability to "fine-tune" a model
  - We can train the model to accomplish a new task or learn the intricacies of a new corpus without retraining the mode
    - Although this can still take time and require quite a lot of compute power
- This means we could take an off-the-shelf, pre-trained BERT model and fine-tune it to an existing corpus
  - See this notebook for a Python example of fine-tuning BERT

# Contextualized embeddings

### Using contextualized embeddings in R

- Most contextualized embeddings require specialized programming languages optimized for large matrix computations like `PyTorch` and `Tensorflow`
- Once installed, I recommend using `keras`, a high-level package that can be used to implement various neural network methods without directly writing `Tensorflow` code.
- It is possible to work with these models in R, but you might be better off learning Python!

# Summary

- Word embeddings use a neural language model to better represent texts as dense vectors
  - Distributional semantics
  - Analogical reasoning
  - Sociological analysis of meaning and representations
- Recent methodological advances better incorporate context
  - Better semantic representations but huge financial and environmental costs