aircall

Tech Quicky - 2022 Jan

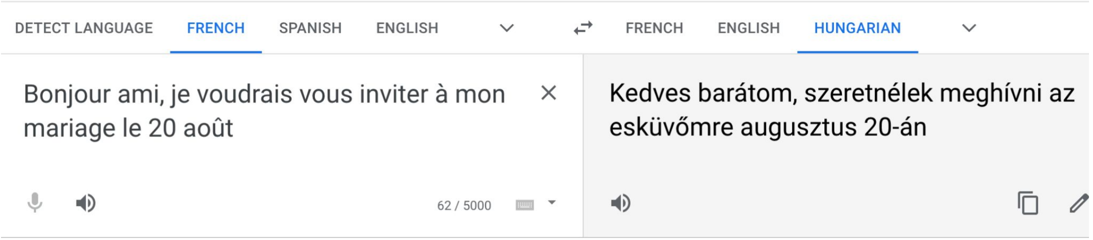
Introduction to NLP
Word Vectors

Natural language processing (NLP)

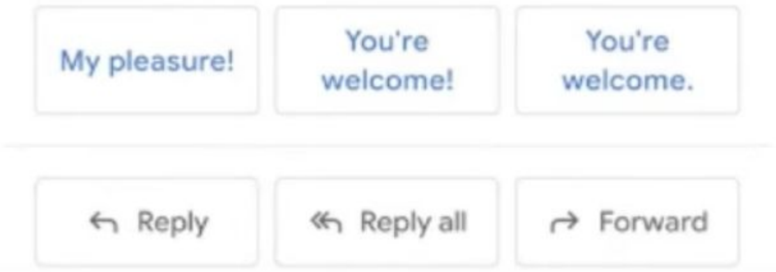
process/analyze natural language data (text/speech)
with algorithms and computer systems

Examples

Autocomplete / Autocorrect



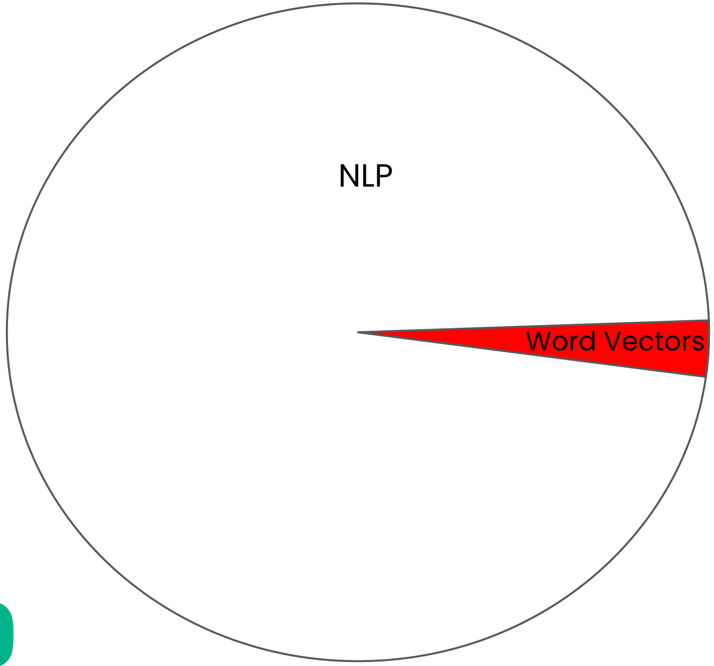
Machine Translation



Text / Response Generation

Huge and booming field

Only looking at a thin slice today



Word Vectors

Distributional Word Vectors

Word Embeddings

Neural Word Representation

Problem statement

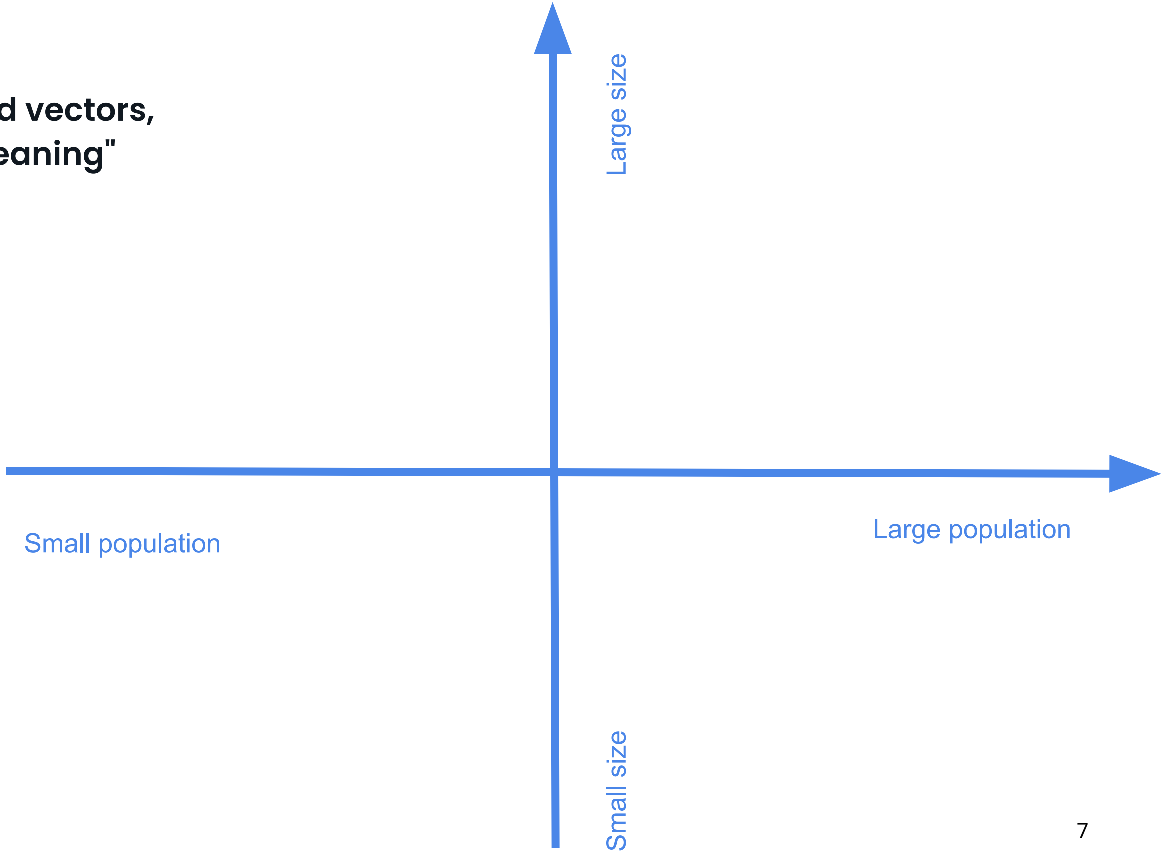
How can we

- get computers to understand the language?
- represent word meaning?
- encode similarity?

Word Vectors – intuition

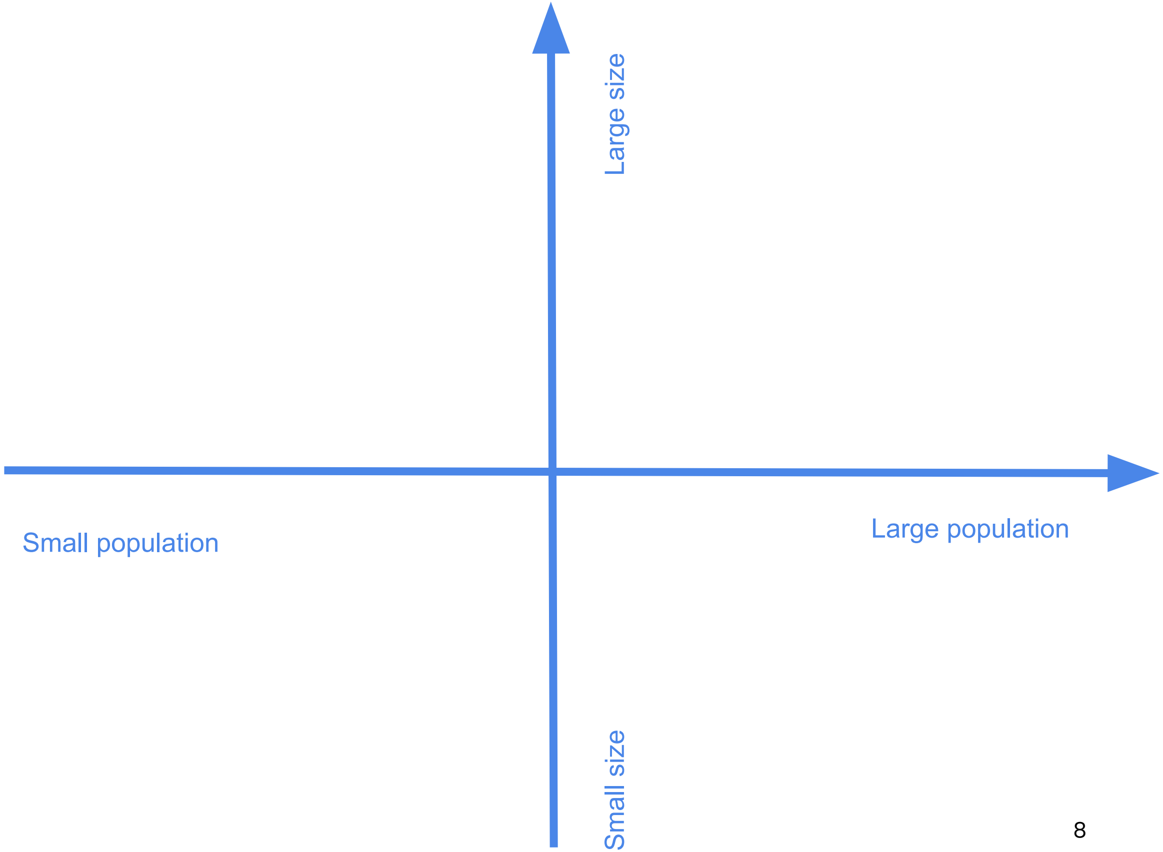
Word Vectors - Intuition

- let's represent words as fixed sized vectors,
- with dimensions along axis of "meaning"



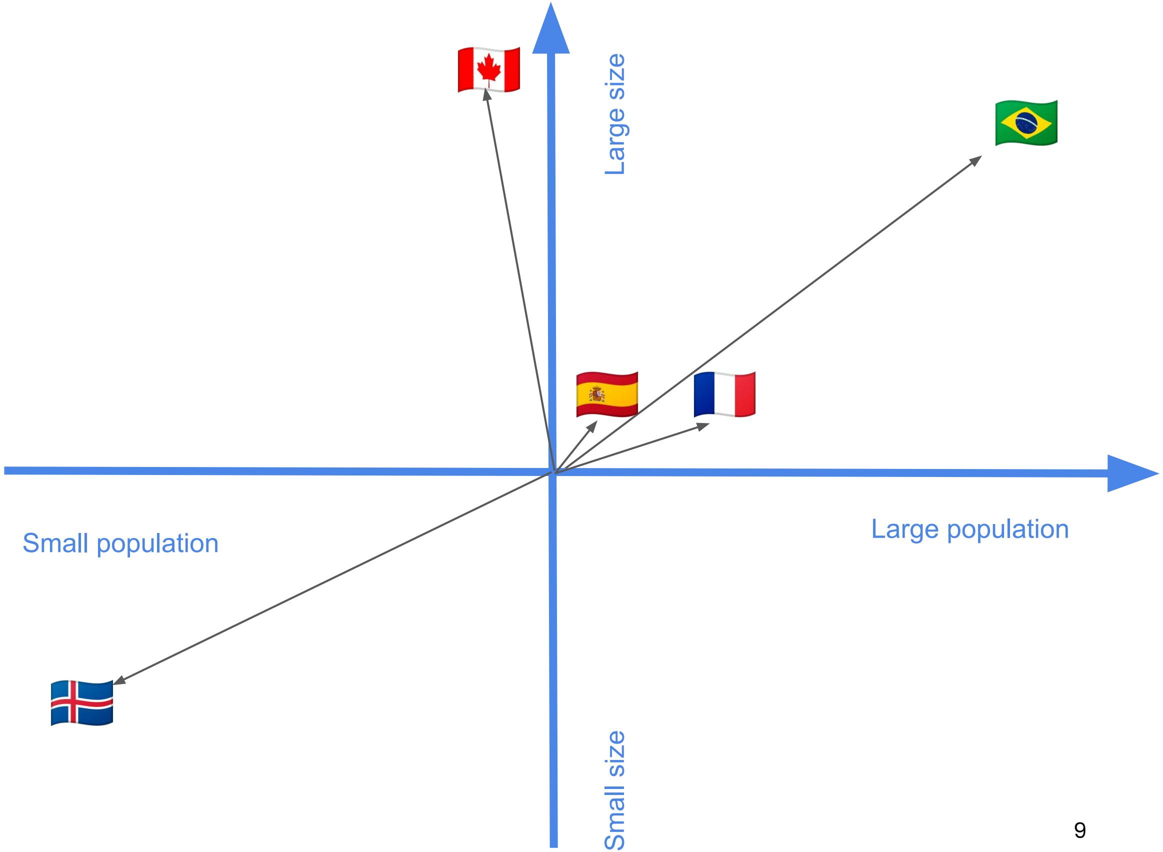
Word Vectors - Intuition

Country	Population (mil pp)	Size (mil km²)
🇫🇷 France	68	0.5
🇪🇸 Spain	47	0.5
🇧🇷 Brazil	214	8.5
🇮🇸 Iceland	0.4	0.1
🇨🇦 Canada	38	9



Word Vectors - Intuition

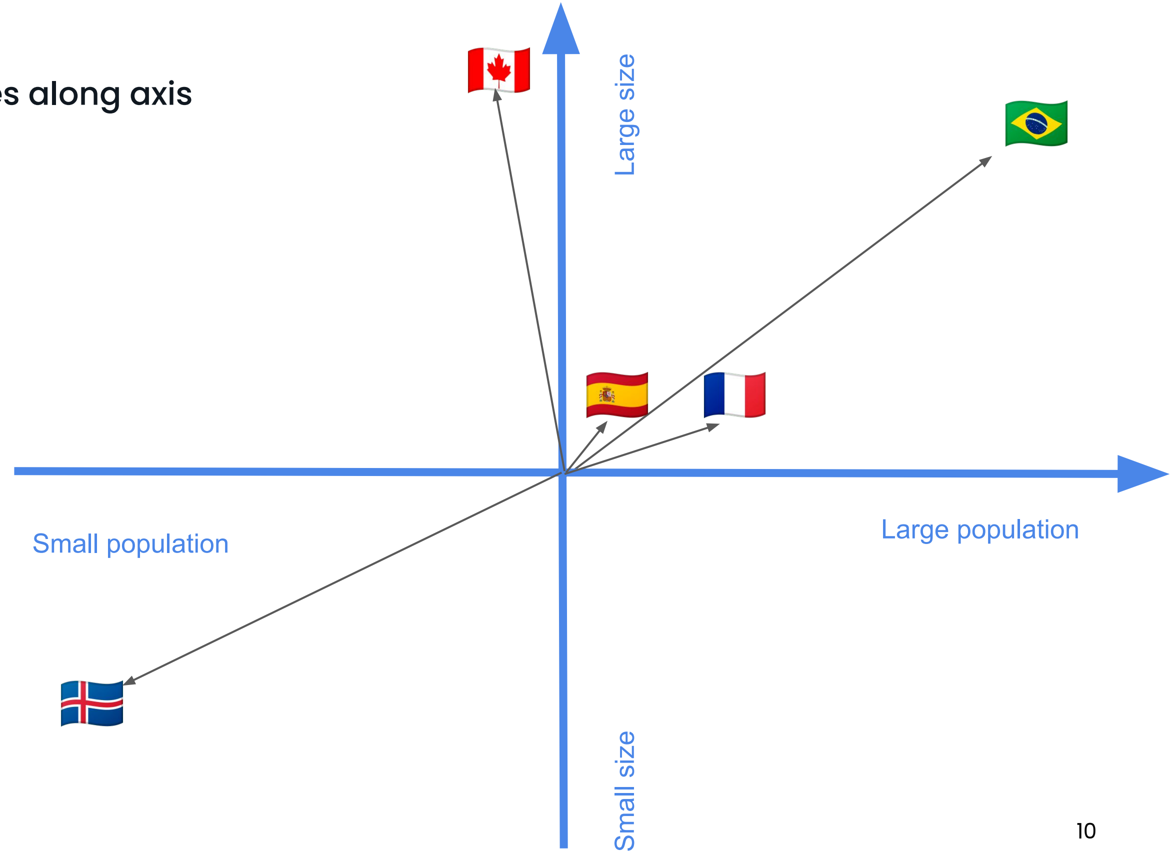
Country	Population (mil pp)	Size (mil km^2)
🇫🇷 France	68	0.5
🇪🇸 Spain	47	0.5
🇧🇷 Brazil	214	8.5
🇮🇸 Iceland	0.4	0.1
🇨🇦 Canada	38	9



Word Vectors - Intuition

Measured similarity by comparing values along axis

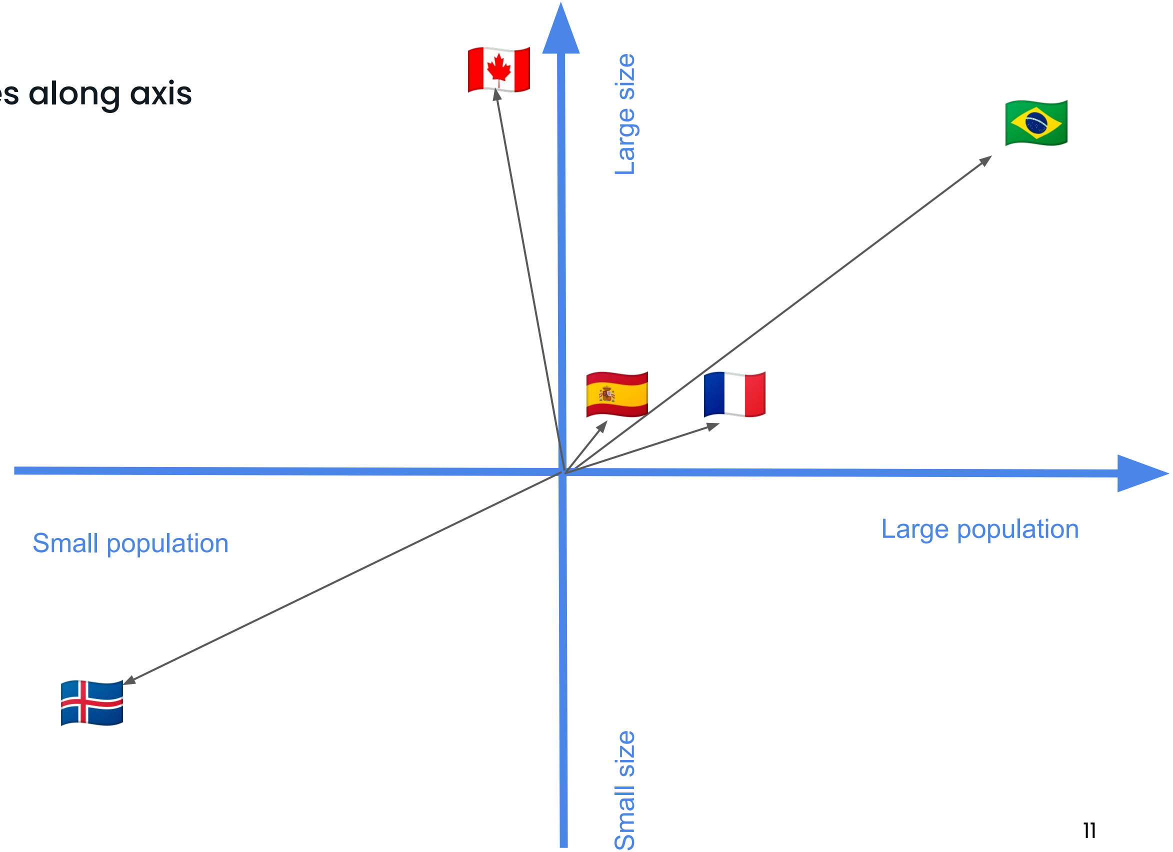
🇧🇷 [0.9, 0.8]
🇫🇷 [0.3, 0.2]
🇪🇸 [0.2, 0.2]
🇨🇦 [-.1, 0.9]
🇮🇸 [-.8, -.8]



Word Vectors - Intuition

Measured similarity by comparing values along axis

🇧🇷 [0.9, 0.8]
🇫🇷 [0.3, 0.2]
🇪🇸 [0.2, 0.2]
🇨🇦 [-.1, 0.9]
🇮🇸 [-.8, -.8]



Dot Product similarity

$$\mathrm{dot}(a,b) = a \cdot b = x_1 \times x_2 + y_1 \times y_2$$

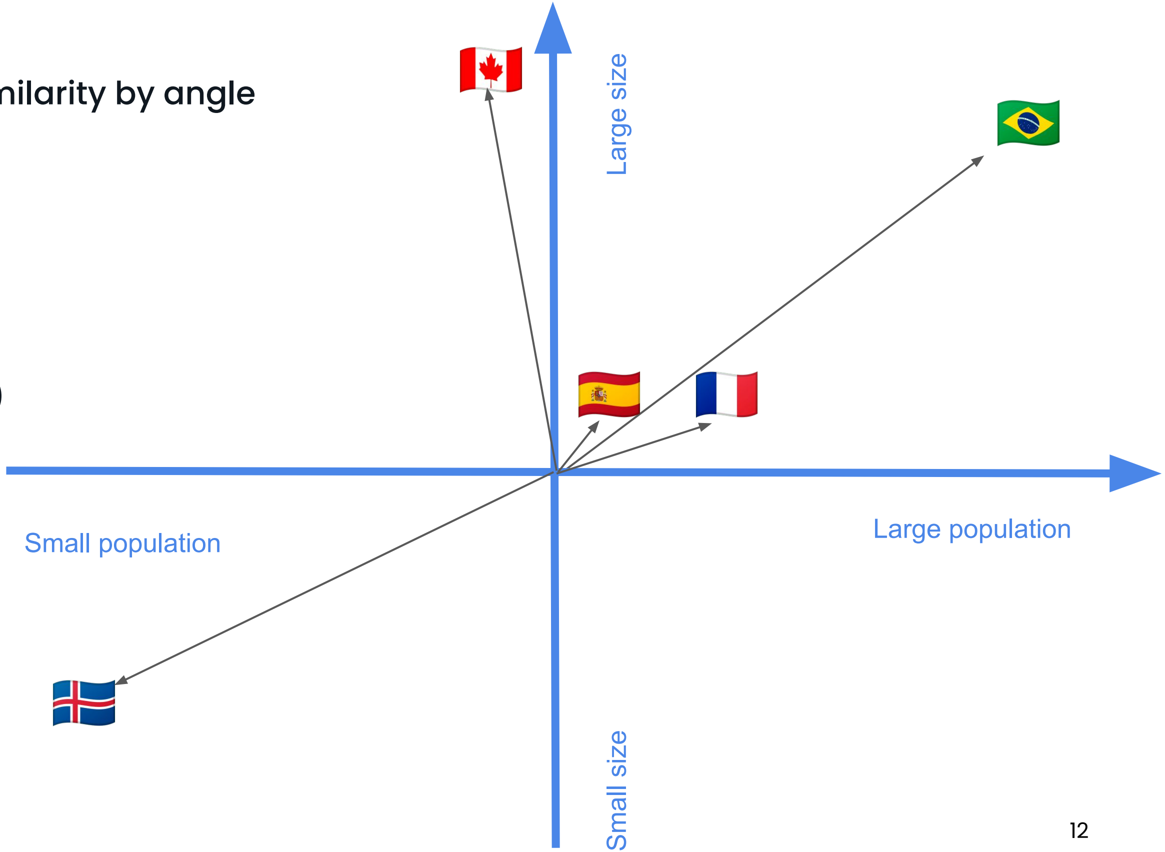
🇫🇷 · 🇪🇸 = 0.1
🇧🇷 · 🇨🇦 = 0.63
🇧🇷 · 🇮🇸 = -1.36

Word Vectors - Intuition

Normalize by size of vectors, measure similarity by angle

🇧🇷 [0.9, 0.8] - ||🇧🇷|| = 1.2
🇫🇷 [0.3, 0.2] - ||🇫🇷|| = 0.36
🇪🇸 [0.2, 0.2] - ||🇪🇸|| = 0.28
🇨🇦 [-.1, 0.9] - ||🇨🇦|| = 0.9
🇮🇸 [-.8, -.8] - ||🇮🇸|| = 1.13

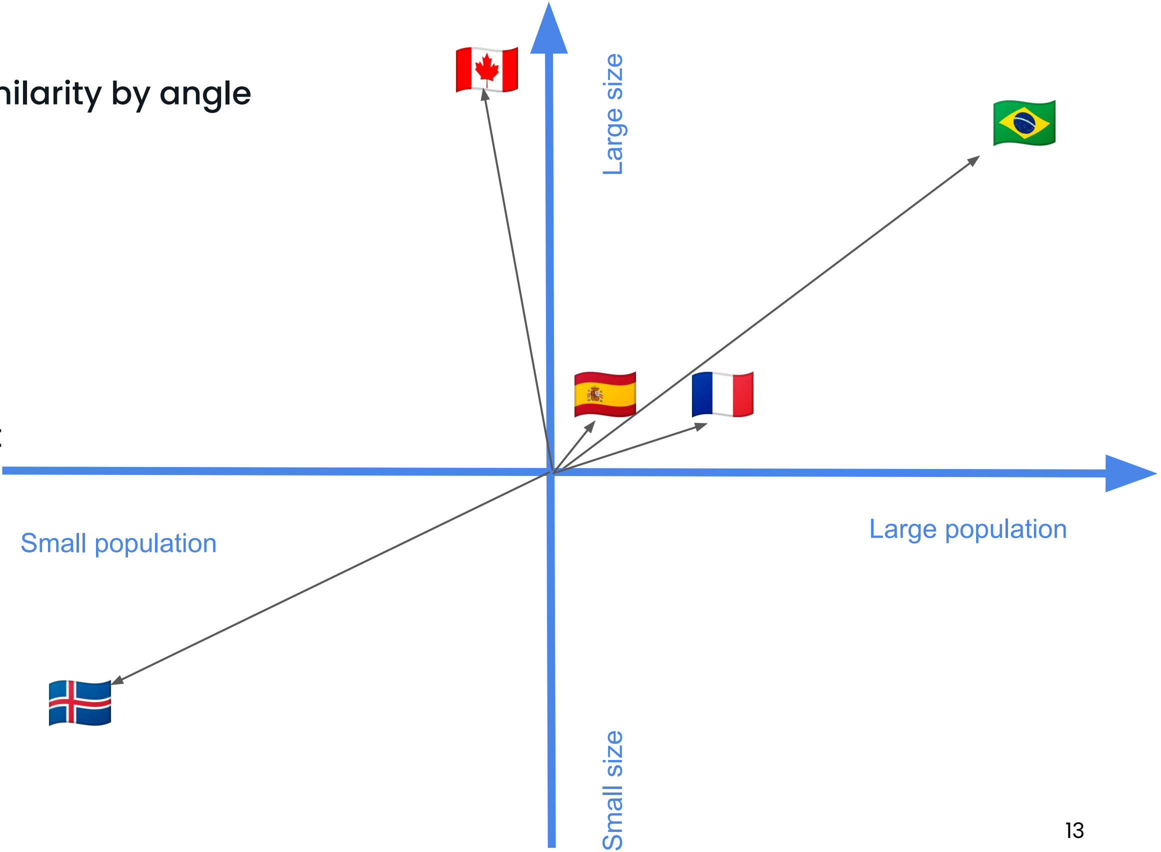
$||\mathbf{a}|| = \sqrt{\mathbf{a} \cdot \mathbf{b}}$ (Euclidean length of the vector)



Word Vectors - Intuition

Normalize by size of vectors, measure similarity by angle

🇧🇷 [0.9, 0.8] - ||🇧🇷|| = 1.2
🇫🇷 [0.3, 0.2] - ||🇫🇷|| = 0.36
🇪🇸 [0.2, 0.2] - ||🇪🇸|| = 0.28
🇨🇦 [-.1, 0.9] - ||🇨🇦|| = 0.9
🇮🇸 [-.8, -.8] - ||🇮🇸|| = 1.13



Cosine similarity - normalized dot product

cosine(a,b) = a · b / (||a|| × ||b||)

cosein(🇫🇷,🇪🇸) = 0.1 / 0.1008 =0.99
cosine(🇧🇷,🇨🇦) = 0.63 / 1.08 = 0.58
cosine(🇧🇷,🇮🇸) = -1.36 / 1.36 = -0.99

Values go from [-1, 1]

Word Vectors - calculation

How could we calculate meaningful word vectors automatically?

Idea 1 - Distributional Semantics

words occurring in the same contexts
tend to have similar meanings

"a word is characterized by the company it keeps"
J.R. Firth in 1957

Let's define a word’s meaning by

- **its context**
- **its neighboring words**

Example:

To start my day, I drink hot ***coffee*** *in the mornings.*

Sipping some warm ***tea*** *helps me wake up in the morning.*

Context:

coffee : drink, hot, morning, start my day
tea : sipping, warm, wake up, morning

coffee & tea

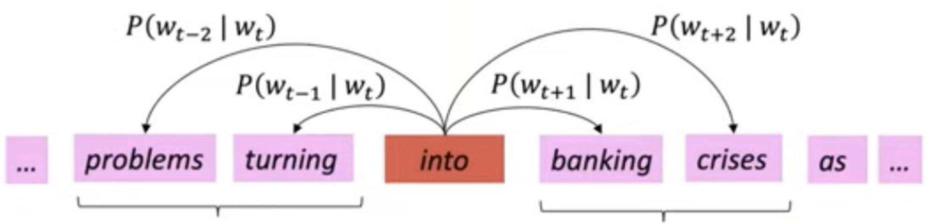
- both a liquid
- are consumed warm
- early in the day to activate people

Idea 2 - Self-supervised learning

We have lots of examples of language in use, available to us:

- Digitized books, newspapers
- Wikipedia, forums
- Social Media posts (reddit, twitter)

Let's train a neural network to calculate the word vectors - using these existing bodies of text as labelled data



Steps to calculate word vector:

1. Start with random word vectors assigned to each word
2. Go through each position in the training data (text corpus)
3. Take a center word and surrounding context words
4. Use word vector similarity to calculate probability of co-occurrence
5. Keep adjusting word vectors until you maximize the probability based on actual co-occurrences in the corpus

Timeline

2000s

- the idea of calculating word embedding using neural networks (Bengio 2003)
- word embedding are very useful for NLP tasks (Collobert 2007)

Early 2010s

- Efficient ways to calculate embeddings
  - Word2Vec paper - (Mikolov et al. 2013)
  - GloVe paper - (Pennington et al. 2014)

GloVe:
Global Vectors for Word Representation

Word Vectors - Demo

aka the “fun part”

Word Vectors - Usefulness

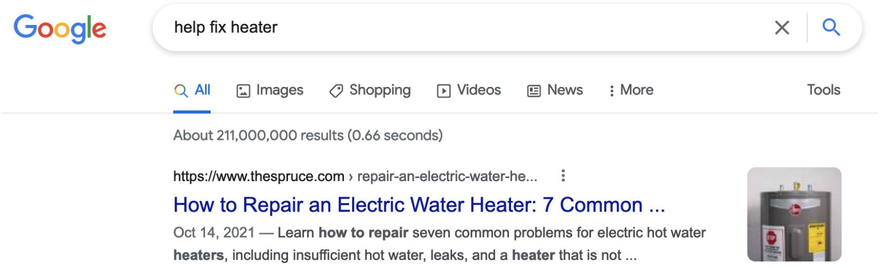
What are word vectors good at?
What are word vectors bad at?

What are word vectors good at?

Automated dictionary generation (🇬🇧 → 🇪🇸)

Semantic search

- Classical search is lexical/keyword based vs
- Understanding query context, user intent, similarities



They revolutionized NLP

Almost all research in NLP start with word vectors and use it as a baseline to compare results to

In many NLP tasks state-of-the-art results are obtained using word vectors

What are word vectors bad at?

Static mappings

- one vector mapped to all occurrences of a word
- different meanings are mixed together
- vectors for the different meanings are averaged

Notebook

do we mean 📔 or 💻?

Park

are we talking about 🚗 or 🌳?

Rare words

- if a word is only present a few times in the corpus, the algorithms will have problems learning good representations
- rare words, have bad representation and frequently appear close to other, unrelated rare words

They have their weaknesses

What’s next?

What comes after word vectors?

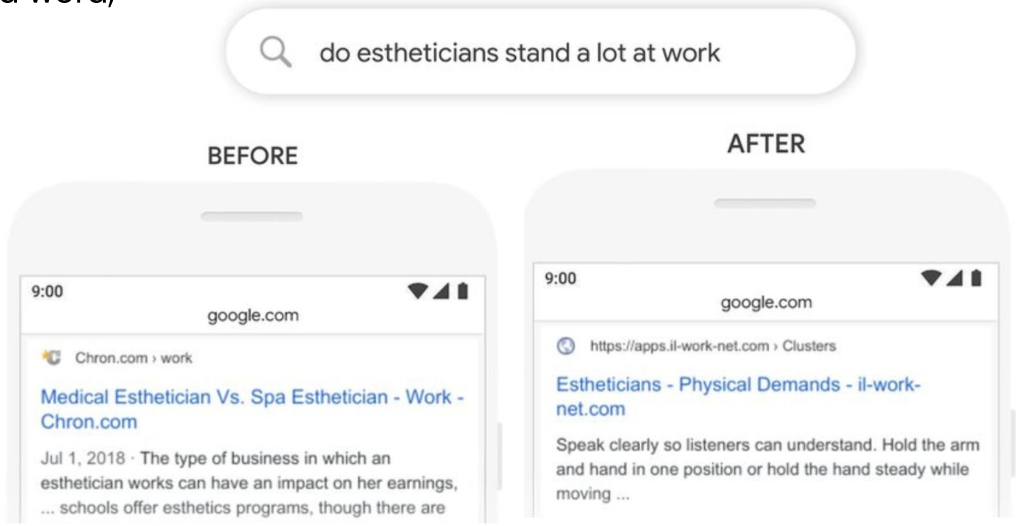
Dynamic, contextualized word vectors

Compute a different vector for each occurrence of a word, based on its immediate context

Many improvements:

- Attention mechanism (Vaswani et al. 2017)
- Transformer-based models
    - Recurrent neural networks (RNN)
    - Sequence-to-Sequence(Seq2Seq)
    - Encoder-decoder networks

In 2019 Google Search was updated with BERT model resulting in a massive improvement:



“the biggest leap forward in the past five years, and one of the biggest leaps forward in the history of Search”

Take away

- A surprising result - word meaning can be represented well by large vectors of numbers
- These vectors can be calculated using a "simple" task of calculating and updating distributional similarities

Thank you!

Any questions?