Representing human language and word meaning

- We, as humans, communicate efficiently through natural (human) language.
- ***How can we represent language in a computer such that the computer can understand and generate it?***
- Let's start by thinking about a word and its meaning!

Traditional/linguistic way to think about word meaning

cat 1 of 5 **noun**

'kat

often attributive

Synonyms of *cat* >

1 **a** : a carnivorous mammal (*Felis catus*) long domesticated as a pet and for catching rats and mice

b : any of a family (Felidae) of carnivorous usually solitary and nocturnal mammals (such as the domestic cat, lion, tiger, leopard, jaguar, cougar, wildcat, lynx, and cheetah)

https://www.merriam-webster.com/dictionary/cat
https://www.nationalgeographic.com/animals/mammals/facts/domestic-cat

Traditional/linguistic way to represent word meaning [Demo]

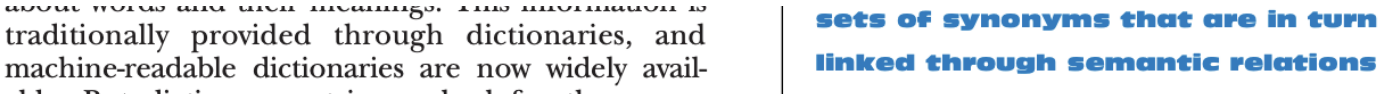
Previously common NLP solution: Use resources like dictionaries and thesauri

- **WordNet** (Miller, 1995): A thesaurus containing lists of synonyms, hypernyms, and other semantic relations

WordNet: A Lexical Database for English

George A. Miller

"WordNet contains more than 118,000 different word forms and more than 90,000 different word senses, or more than 166,000 (f[orm], s[ense]) pairs. Approximately 17% of the words in WordNet are polysemous; approximately 40% have one or more synonyms."


traditionally provided through dictionaries, and machine-readable dictionaries are now widely avail-

sets of synonyms that are in turn linked through semantic relations

Limitations with resources like WordNet

- A useful resource but…
- Requires human labor to create and maintain
- Subjective
- Missing context and nuance (e.g., connotations)
  - E.g., “proficient” is listed as a synonym for “good”
  - This is only correct in some contexts!
- Missing new meanings of words
  - E.g., bet, extra, cap, fire, busted, shook, salty
  - Impossible to keep up-to-date!
- Not a cost-efficient, scalable, and future-proof approach

[Guest lecture (slang)] Zhewei Sun (2/18)

Representing words as one-hot vectors

To use words as input to ML algorithms/neural networks, we need convert them to numbers.

In traditional NLP, we regard words as discrete symbols (e.g., “cat” and “dog”)

Means one 1, the rest 0s

Such symbols for words can be represented by one-hot vectors (simplest thing):

cat = [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]

dog = [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]

Vector dimension = number of words in vocabulary

Problem with representing words as one-hot vectors

Example: In web search, if a user searches for "**pet** adoption," we would like to match documents containing "**cat** adoption" and "**dog** adoption" as well, but…

cat = [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]

dog = [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]

pet = [0 0 0 0 0 0 0 0 0 0 0 0 0 1 0]

these vectors are orthogonal and there is no notion of **similarity** for one-hot vectors!

Solution:

- Represent them as a collection of features based on WordNet (and other human-annotated resources) to get similarity → Well-known to fail due to incompleteness
- **Instead, represent word meaning as well as encode similarity in the vectors themselves by learning from naturally occurring text source**

Word vectors

We will build a **dense vector** for each word and measure similarity with **dot product**

$$cat = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{pmatrix} \qquad dog = \begin{pmatrix} 0.413 \\ 0.582 \\ -0.007 \\ 0.247 \\ 0.216 \\ -0.718 \\ 0.147 \\ 0.051 \end{pmatrix}$$

Note: Word vectors are also called word embeddings or word representations.

Representing words by their context

- **A word's meaning is given by the words that frequently appear nearby**
  - *"You shall know a word by the company it keeps"* (Firth, 1957)
  - One of the most influential and successful ideas in modern NLP!

… made a cup of	***coffee***	*for breakfast …*
… made a cup of	***tea***	*for breakfast …*

… drank her black	***coffee***	*for a quick caffeine boost …*
… drank her green	***tea***	*in the afternoon …*

Representing words by their context

- **A word's meaning is given by the words that frequently appear nearby**
  - *"You shall know a word by the company it keeps"* (Firth, 1957)
  - One of the most influential and successful ideas in modern NLP!
- When a word *w* appears in a text, its **context** is the set of words that appear nearby (within a fixed-size window).
- We use many contexts of *w* to build up a representation of *w*

…government debt problems turning into ***banking*** *crises as happened in 2009…*
…saying that Europe needs unified ***banking*** *regulation to replace the hodgepodge…*
…India has just given its ***banking*** *system a shot in the arm…*

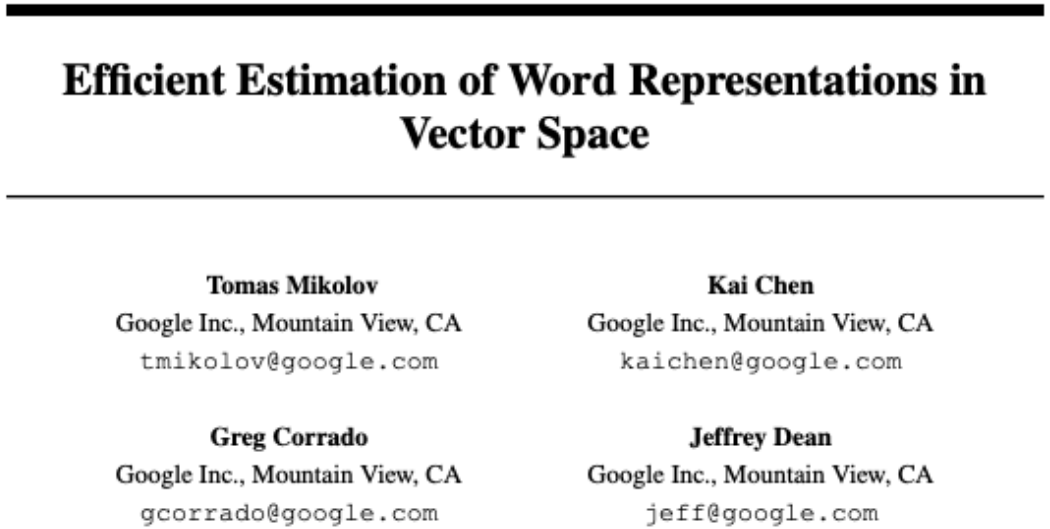
These context words will represent ***banking***

Lecture plan

1. The course
2. A few uses of NLP
3. Human language and word meaning
4. **Word2vec introduction**
5. Word2vec objective function

Word2vec: Overview

Word2vec (Mikolov et al., 2013) is a framework for learning word vectors



Efficient Estimation of Word Representations in Vector Space

Tomas Mikolov
Google Inc., Mountain View, CA
tmikolov@google.com

Kai Chen
Google Inc., Mountain View, CA
kaichen@google.com

Greg Corrado
Google Inc., Mountain View, CA
gcorrado@google.com

Jeffrey Dean
Google Inc., Mountain View, CA
jeff@google.com

1.1. Goals of the Paper

"The main goal of this paper is to introduce techniques that can be used for learning high-quality word vectors from huge data sets with billions of words, and with millions of words in the vocabulary."

Efficient Estimation of Word Representations in Vector Space (Mikolov et al., 2013)

Word2vec: Overview

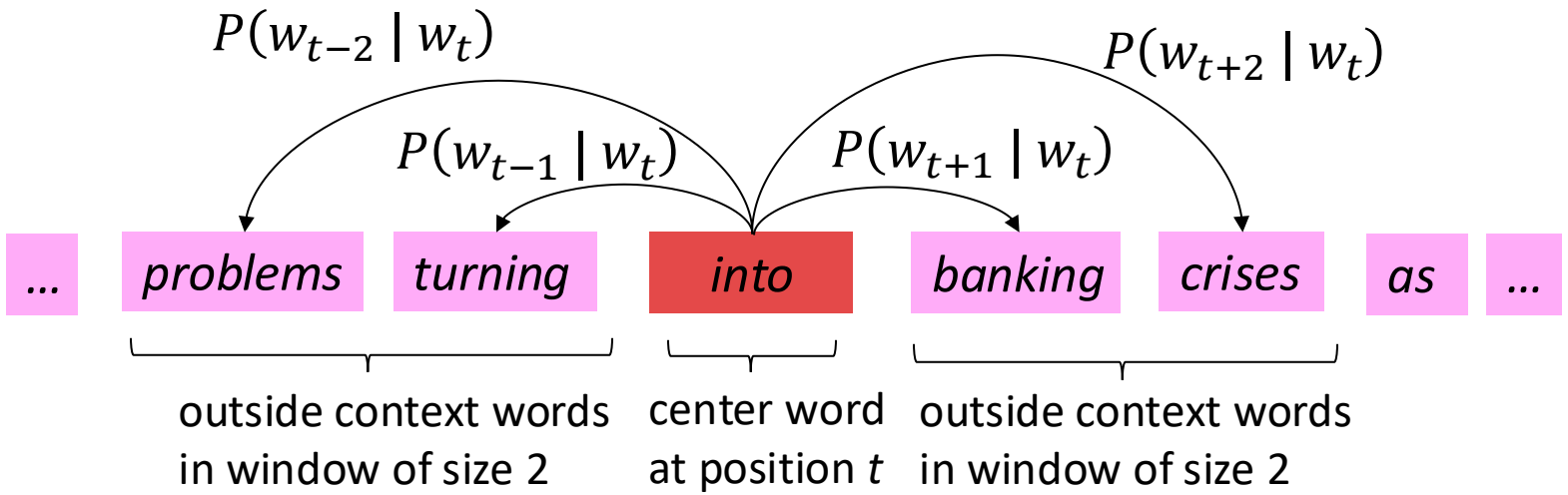
Word2vec (Mikolov et al., 2013) is a framework for learning word vectors

Idea:

- We have a large corpus ("body") of text (e.g., Google News dataset)
- Every word in a fixed vocabulary is represented by a vector
- Go through each position *t* in the text, which has a center word *c* and outer context words *o*
- Use the similarity of the word vectors for *c* and *o* to calculate the probability of *o* given *c* (or vice versa)
- Keep adjusting the word vectors to maximize this probability

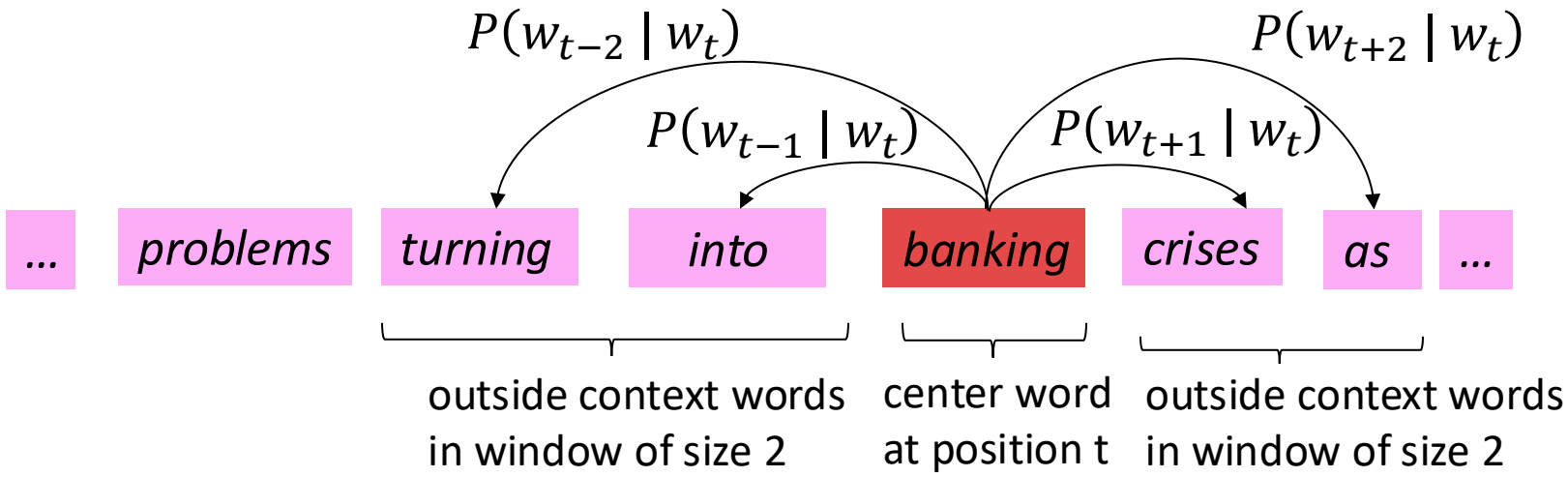
Word2vec: Overview

Example windows and process for computing $P(w_{t+j} \mid w_t)$



Word2vec: Overview

Example windows and process for computing $P(w_{t+j} \mid w_t)$



Word2vec: Objective function

For each position $t = 1, \ldots, T$, predict outer context words within a window of fixed size m, given center word w_t. Data likelihood:

$$\text{Likelihood} = L(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \le j \le m \\ j \neq 0}} P(w_{t+j} \mid w_t; \theta)$$

θ is all variables to be optimized

sometimes called a *cost* or *loss* function

The objective function $J(\theta)$ is the (average) negative log likelihood:

$$J(\theta) = -\frac{1}{T} \log L(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-m \le j \le m \\ j \neq 0}} \log P(w_{t+j} \mid w_t; \theta)$$

Minimizing objective function $\Leftrightarrow$ Maximizing predictive accuracy

Negative log likelihood explained

Word2vec: Prediction function

- Our objective function:

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-m\leq j\leq m \\ j\neq 0}} \log P\left(w_{t+j} \mid w_t; \theta\right)$$

- **Question:** How to model $P\left(w_{t+j} \mid w_t; \theta\right)$?
- **Notation:** We will *use two* vectors per word *w*:
  - v_w when *w* is a center word
  - u_w when *w* is an outer context word
- **Answer:** For a center word *c* and an outer context word *o*, we define $P(o|c)$ as follows:

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w\in V}\exp(u_w^T v_c)}$$

Word2vec: Prediction function

② Exponentiation makes anything positive

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

① Dot product compares similarity of *o* and *c*.
$u^T v = u \cdot v = \sum_{i=1}^{n} u_i \, v_i$
Larger dot product = larger probability

③ Normalize over entire vocabulary to give probability distribution

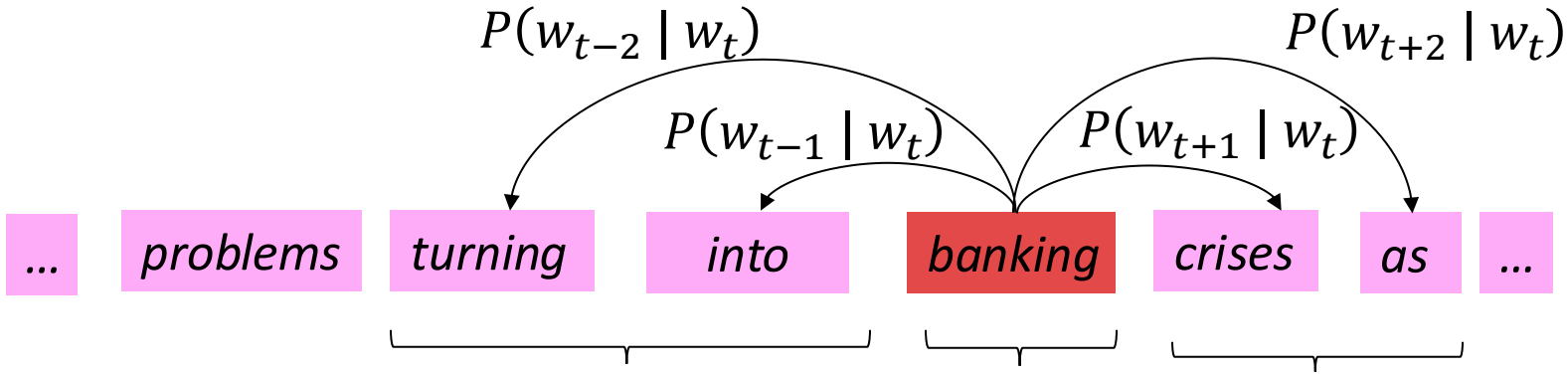
- This is an example of the **softmax function** $\mathbb{R}^n \to (0,1)^n$

$$\text{softmax}(x_i) = \frac{\exp(x_i)}{\sum_{j=1}^{n} \exp(x_j)} = p_i$$

- The softmax function maps arbitrary values x_i to a probability distribution p_i
  - "max" because amplifies probability of largest x_i
  - "soft" because still assigns some probability to smaller x_i
  - Frequently used in deep learning

Word2vec: Training word vectors with gradient descent

- Start with random word vectors
- Iterate through each position in the corpus
- Try to **predict** surrounding words using word vectors: $P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$



- Update vectors so they can predict surrounding words better using **gradient descent**
- Doing no more than this, this algorithm learns word vectors that capture well word similarity and meaningful directions in a word space!

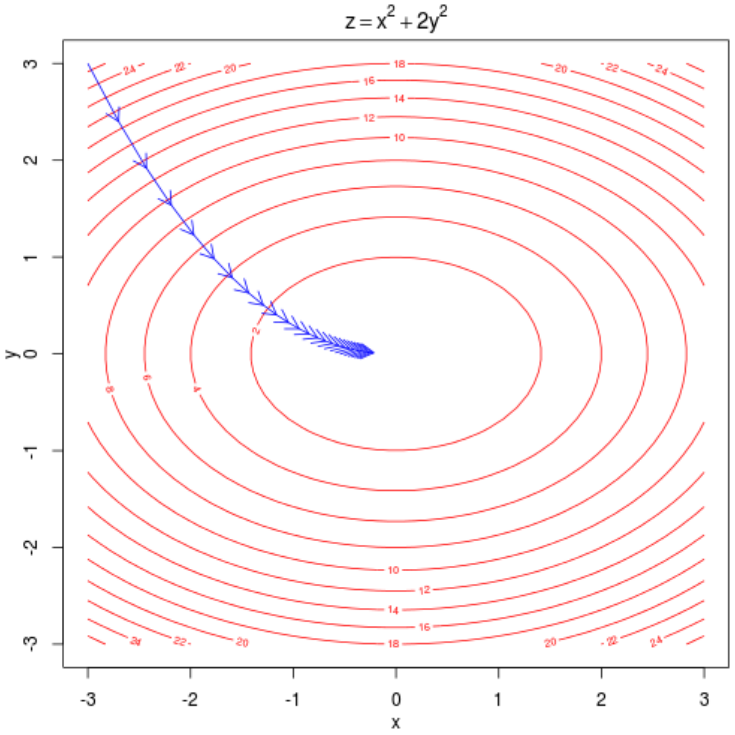
Word2vec: Training word vectors with gradient descent

To train a model, we gradually adjust parameters to minimize a loss

- Recall: θ represents **all** the model parameters
- In our case, with *d*-dimensional vectors and *V*-many words, we have →

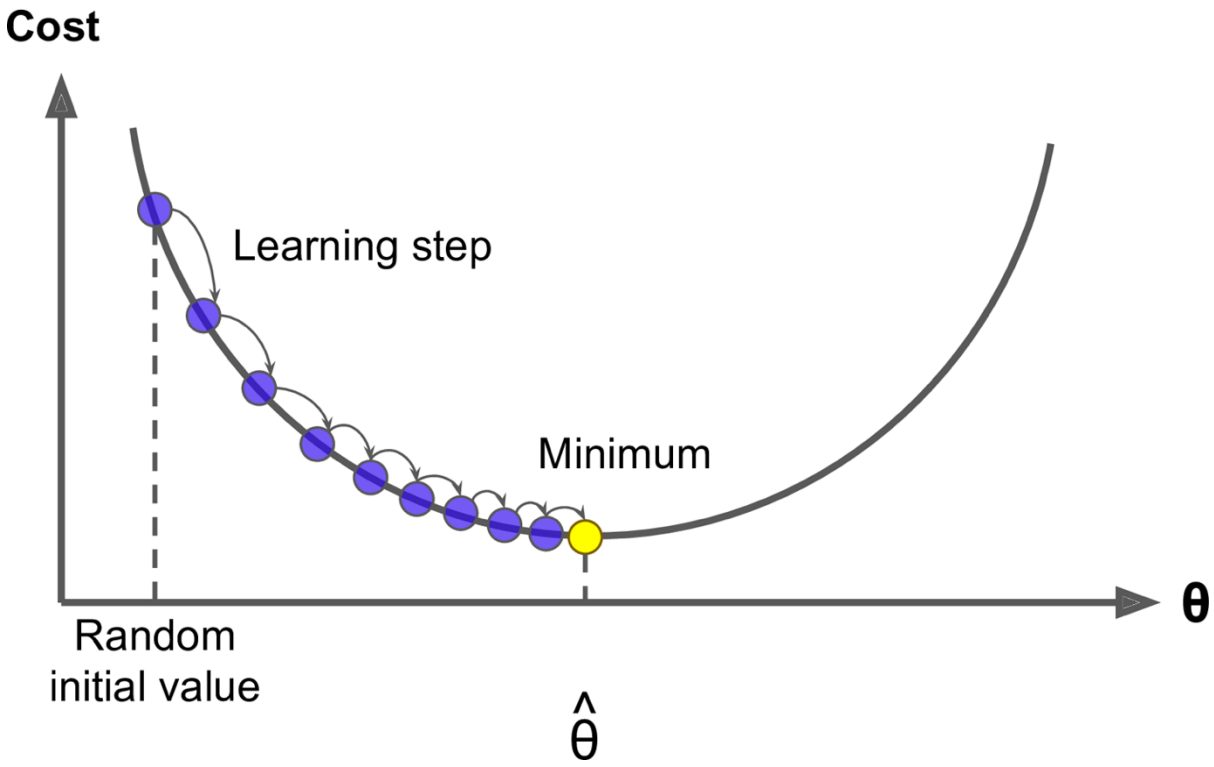
$$\theta = \begin{bmatrix} v_{aardvark} \\ v_a \\ \vdots \\ v_{zebra} \\ u_{aardvark} \\ u_a \\ \vdots \\ u_{zebra} \end{bmatrix} \in \mathbb{R}^{2dV}$$

- We optimize these parameters
- We compute all vector gradients!



[Review] Gradient descent

- To learn word vectors, we designed the objective function $J(\theta)$ we want to minimize
- **Gradient descent** is an algorithm to minimize $J(\theta)$
- **Idea:** For current value of θ, calculate gradient of $J(\theta)$, then take small step in direction of negative gradient. Repeat.



[Review] Gradient descent

- Update equation (in matrix notation):

$$\theta^{new} = \theta^{old} - \alpha \nabla_\theta J(\theta)$$

α = *step size* or *learning rate*

- Algorithm:

```
while True:
    theta_grad = evaluate_gradient(J,corpus,theta)
    theta = theta - alpha * theta_grad
```

[Review] Stochastic gradient descent

- **Problem**: $J(\theta)$ is a function of **all** positions in the corpus (potentially billions!)
  - So $\nabla_\theta J(\theta)$ is very expensive to compute
- You would wait a very long time before making a single update!
- Very bad idea for pretty much all neural nets!

- **Solution: Stochastic gradient descent (SGD)**
  - Repeatedly sample a small batch and update after each one
- Algorithm:

```
while True:
    window = sample_window(corpus)
    theta_grad = evaluate_gradient(J,window,theta)
    theta = theta - alpha * theta_grad
```

Objective Function

Maximize $J'(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \le j \le m \\ j \neq 0}} p(w'_{t+j} | w_t; \theta)$

Or minimize ave. neg. log likelihood $J(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-m \le j \le m \\ j \neq 0}} \log p(w'_{t+j} | w_t)$

[negate to minimize; log is monotone]

text length

window size

where $p(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w=1}^{V} \exp(u_w^T v_c)}$

word IDs

We now take derivatives to work out minimum

Each word type (vocab entry) has two word representations: as center word and context word

$$\frac{\partial}{\partial v_c} \log \frac{\exp(u_o^T v_c)}{\sum_{w=1}^{V} \exp(u_w v_c)}$$

$$= \underbrace{\frac{\partial}{\partial v_c} \log \exp(u_o^T v_c)}_{①} - \underbrace{\frac{\partial}{\partial v_c} \log \sum_{w=1}^{V} \exp(u_w^T v_c)}_{②}$$

① $\frac{\partial}{\partial v_c} \underbrace{\log \exp}_{\text{inverses}}(u_o^T v_c) = \frac{\partial}{\partial v_c} u_o^T v_c = u_o$

↑ Vector!

Not high school single variable calculus

You can do things one variable at a time, and this may be helpful when things get gnarly.

$$\forall j \quad \frac{\partial}{\partial (v_c)_j} u_o^T v_c = \frac{\partial}{\partial (v_c)_j} \sum_{i=1}^{d} (u_o)_i (v_c)_i$$

$$= (u_o)_j$$

Each term is zero except when $i = j$

② $\frac{\partial}{\partial v_c} \underbrace{\log}_{f} \underbrace{\sum_{w=1}^{V} \exp(u_w^T v_c)}_{z = g(v_c)}$

$= \frac{1}{\sum_{w=1}^{V} \exp(u_w^T v_c)} \cdot \frac{\partial}{\partial v_c} \sum_{x=1}^{V} \exp(u_x^T v_c)$ ← Important to change index

$\frac{\partial}{\partial v_c} f(\overbrace{g(v_c)}^{z}) = \frac{\partial f}{\partial z} \cdot \frac{\partial z}{\partial v_c}$ Use chain rule

$= \frac{1}{\sum_{w=1}^{V} \exp(u_w^T v_c)} \cdot \left(\sum_{x=1}^{V} \frac{\partial}{\partial v_c} \underbrace{\exp}_{f} \underbrace{(u_x^T v_c)}_{z = g(v_c)} \right)$ Move deriv inside sum

$\left(\sum_{x=1}^{V} \exp(u_x^T v_c) \frac{\partial}{\partial v_c} u_x^T v_c \right)$ Chain rule

$\left(\sum_{x=1}^{V} \exp(u_x^T v_c) u_x \right)$

CS257 NLP

2. Word vectors

Please suggest your favorite songs!

How can we improve and evaluate word vectors?

Mina Lee | Jan 9, 2025

(Based on slides from Chris Manning)

Lecture plan

1. **Course organization**
2. Finish looking at word2vec
3. Can we capture the essence of word meaning more effectively by counting?
4. Evaluating word vectors
5. Word senses

There are many variations of word embeddings! This can be your final project. 😳

Course organization

- The waitlist for this course will close **this Friday**
- **Memo 1** [github]
  - Writing is due **1/12 (Sunday) at 11:59 PM**
  - Voting is due **1/13 (Monday) at 11:59 PM**
    - Please vote on Monday!
    - On Monday, submission to Gradescope will be open (12:00 AM - 11:59 PM)
- **Assignment 1**
  - Shared part (A1-1) is due **1/14 (Tuesday) at 9:29 AM**
  - Creative part (A1-2) is due **1/21 (Tuesday) at 9:29 AM**
    - Please check out the updated version (addressed minor issues in A1-2) → Ed post
  - Author: Karen Zhou

Course organization

- **Office hours:** Come to discuss **final project ideas** as well as the assignments
  - Mina Lee (Thursdays 11-12 PM @ Searle 207 – starts on Week 2)
    - Research areas: Writing with AI
    - In charge of: Lectures, Memos, and Final project
  - Karen Zhou (Thursdays 1-2 PM @ JCL 205)
    - Research areas: Computational social science, Summarization, Evaluation of LLMs
    - In charge of: Assignment 1, Tutorial 2
  - Chacha Chen (Mondays 10-11 AM @ JCL 205 – starts on Week 2)
    - Research areas: Multimodal LLM evaluation, Post-training for healthcare, Human-centered explanations, Human-AI decision making
    - In charge of: Assignment 2, Tutorial 3
  - Dang Nguyen (Fridays 11-12 PM @ JCL 205)
    - Research areas: Interpretability, Model editing, and Bias and fairness
    - In charge of: Assignment 3, Tutorial 1

Lecture plan

1. Course organization
2. **Finish looking at word2vec**
3. Can we capture the essence of word meaning more effectively by counting?
4. Evaluating word vectors
5. Word senses

[Review of Lecture 1] Representing word meaning in a computer

In traditional NLP, we regarded words as **discrete symbols**

- Build and use resources like dictionaries and thesauri
- **WordNet (Miller, 1995):** Effort to hardcode word meaning and semantic relationship
- Limitation: Hard to create and maintain

We need to convert words to numbers to use them as input to ML algorithms/neural nets.

- Represent words with **one-hot vectors**
- Limitation: High dimensionality. No notion of similarity.

Let's learn **dense vectors** from data, that capture word meaning and encode similarity!

[Review of Lecture 1] Representing word meaning in a computer

Key idea: **Represent words by their context**

- Intuition: Words that appear in similar contexts have similar meanings!

… made a cup of	***coffee***	*for breakfast …*
… made a cup of	***tea***	*for breakfast …*

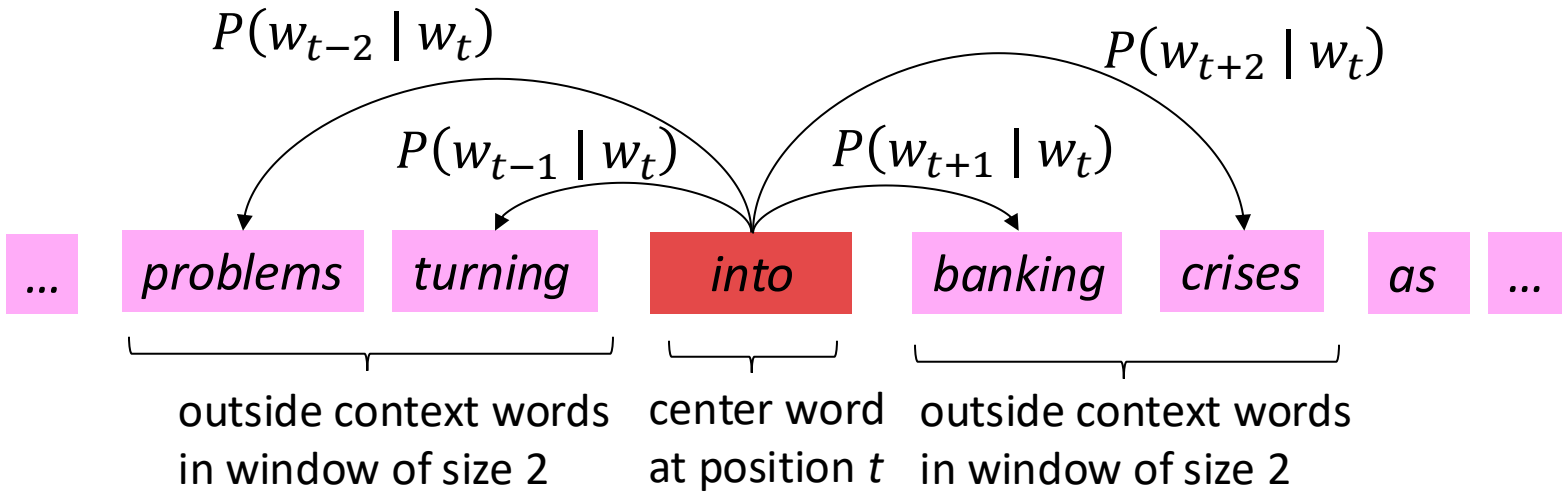
… drank her black	***coffee***	*for a quick caffeine boost …*
… drank her green	***tea***	*in the afternoon …*

Word2vec (Mikolov et al., 2013) is a framework for learning word vectors

[Review of Lecture 1] Word2vec: Main ideas

Word2vec (Mikolov et al., 2013) is a framework for learning word vectors

- Start with a large corpus of text
- Go through each position t in the text



[Review of Lecture 1] Word2vec: Main ideas

Word2vec (Mikolov et al., 2013) is a framework for learning word vectors

- Start with a large corpus of text
- Go through each position *t* in the text
- Try to **predict** outer context words using word vectors $P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$
  - Calculate the probability of *o* given *c*,
    using the similarity of the word vectors for *c* and *o*
- Keep adjusting word vectors to predict better using **stochastic gradient descent**
  - Minimize the following objective function:

$$J(\theta) = -\frac{1}{T}\log L(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-m \le j \le m \\ j \neq 0}} \log P\left(w_{t+j} \mid w_t; \theta\right)$$

[Self-guided review] CS224N Note 1 - Section 3. Working through a gradient [pdf]

Word2vec algorithm family: Variants

Two model variants:

1. **Skip-gram** (SG)

   Predict outer context words given a center word

2. Continuous Bag of Words (CBOW)

   Predict a center word given outer context words

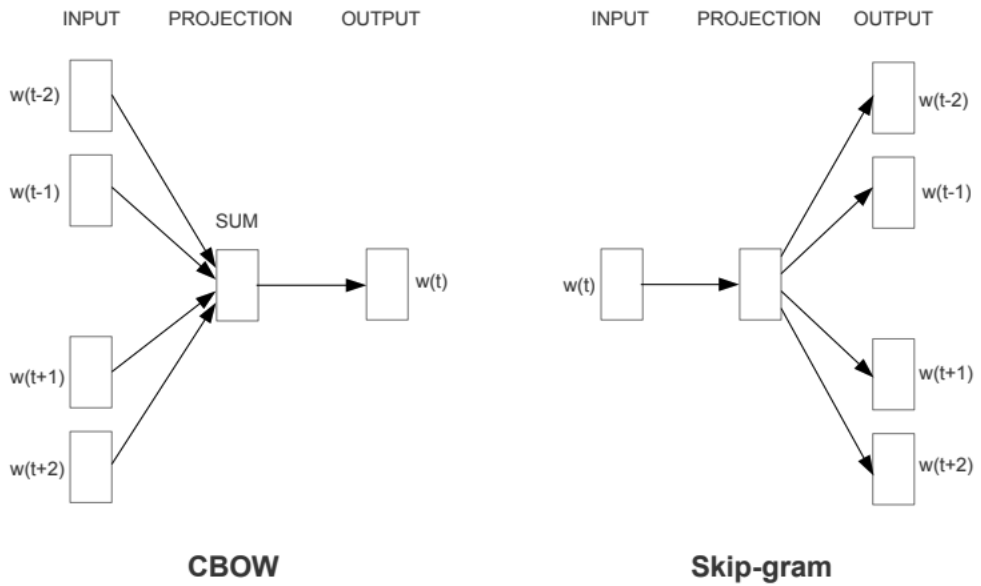


Figure 1: New model architectures. The CBOW architecture predicts the current word based on the context, and the Skip-gram predicts surrounding words given the current word.

Loss functions for training:

1. **Naïve softmax**
2. More optimized variants like hierarchical softmax
3. Negative sampling

Efficient Estimation of Word Representations in Vector Space (Mikolov et al., 2013)
Distributed Representations of Words and Phrases and their Compositionality (Mikolov et al., 2013)

Word2vec algorithm family: Skip-gram with negative sampling

- The normalization term in the prediction function is computationally expensive:

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

A big sum over words (expensive!)

- Hence, standard word2vec implements the skip-gram model with **negative sampling**

- Idea: A good model should be able to differentiate true data from noise by means of binary logistic regression

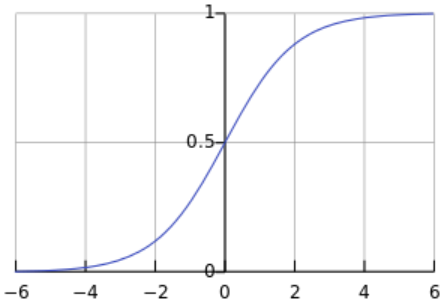
Word2vec algorithm family: Skip-gram with negative sampling

- With **negative sampling**, at each position *t*:
  - Randomly sample *K* words from vocabulary
  - Learn to differentiate a "true" pair (*c*, *o*) vs. several "noise" pairs (*c*, *k*)

$$\log P(o \mid c) + \sum_{k \in K} \log(1 - P(k \mid c))$$

$$= \log \sigma(u_o^T v_c) + \sum_{k \in K} \log(1 - \sigma(u_k^T v_c))$$

$$= \log \sigma(u_o^T v_c) + \sum_{k \in K} \log(\sigma(-u_k^T v_c))$$



$$\sigma(x) = \frac{1}{1+e^{-x}}$$

  - Maximize the following:

$$\log P(o \mid c) = \log \sigma(u_o^T v_c) + \sum_{k \in K} \log \sigma(-u_k^T v_c)$$

Word2vec algorithm family: Skip-gram with negative sampling

- With **naïve softmax**, at each position *t*, we tried to maximize the prediction accuracy

$$\log P(o \mid c) = \log \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

- With **negative sampling**, at each position *t*, maximize the probability of the true pair in the first log and minimize the probabilities of noise pairs in the second part

$$\log P(o \mid c) = \log \sigma(u_o^T\, v_c) + \sum_{k \in K} \log \sigma(-u_k^T\, v_c)$$

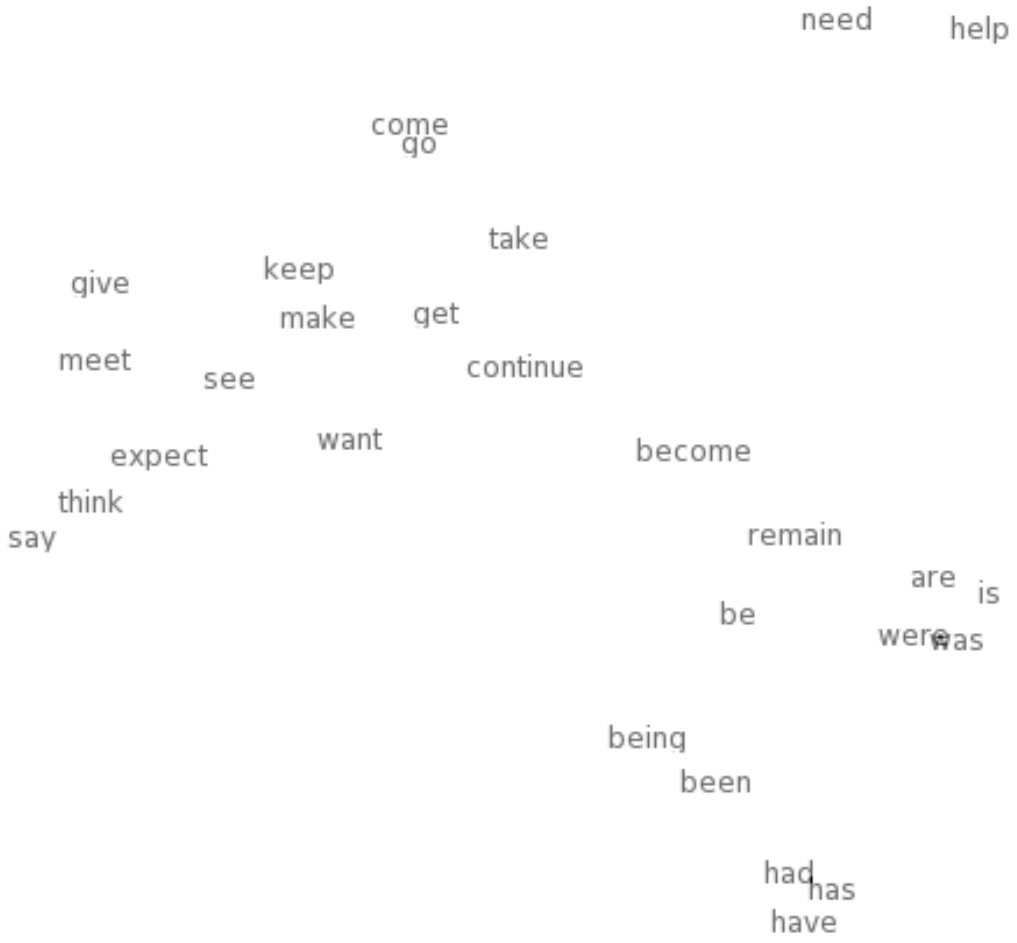
Word2vec algorithm family: Skip-gram with negative sampling

- Many tricks used on the word2vec paper to improve performance:
  - For k, sample negative samples based on word frequency raised to the 3/4rd power
  - For o, give less weight to the distant words by sampling less
  - For t = 1, … T, subsample frequent words (discard common words from data)
  - …

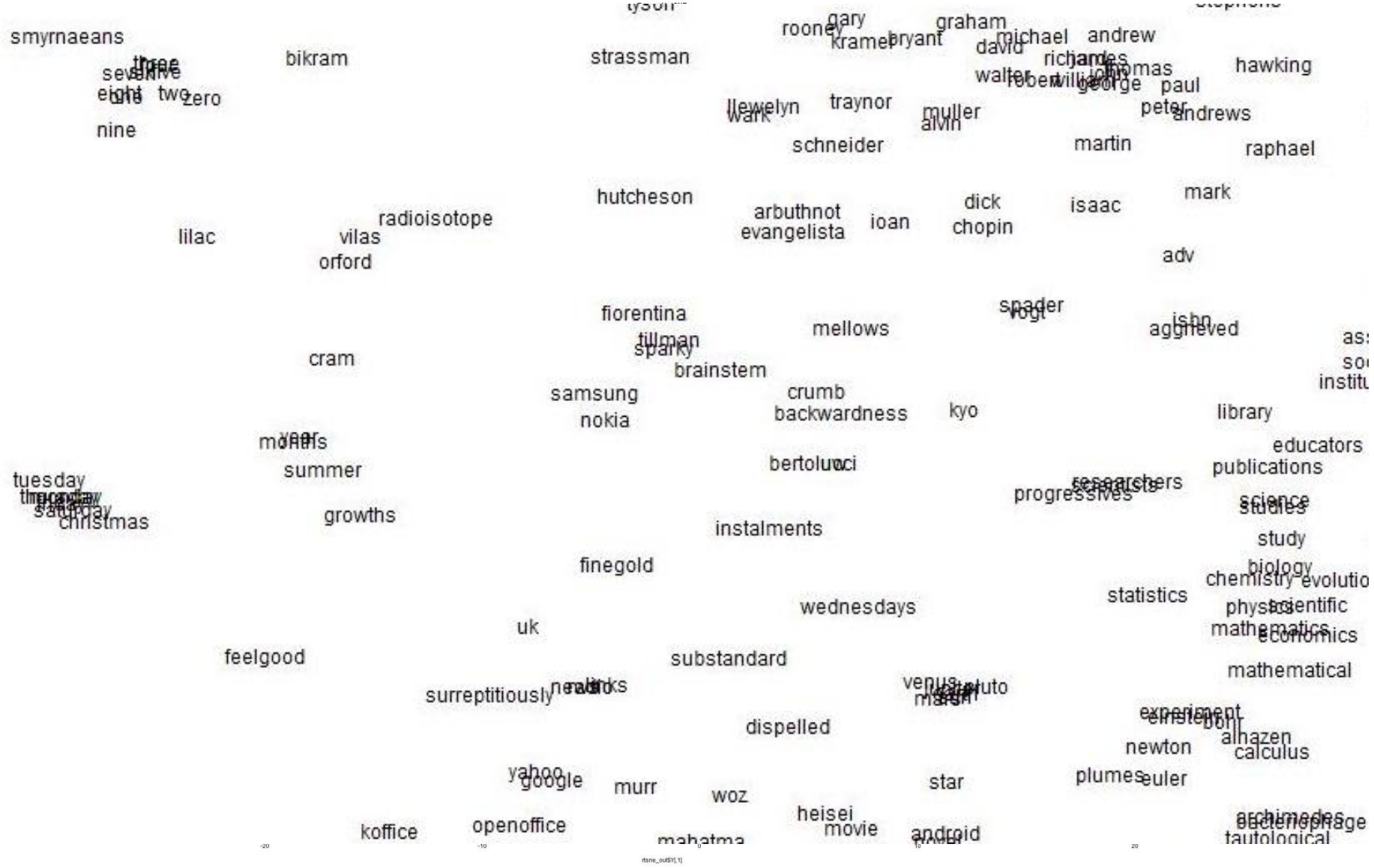
Efficient Estimation of Word Representations in Vector Space (Mikolov et al., 2013)
Distributed Representations of Words and Phrases and their Compositionality (Mikolov et al., 2013)

Word meaning as a word vector [Demo]

$$\textit{expect} = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \\ 0.487 \end{pmatrix}$$



Word2vec maximizes the objective function by putting similar words nearby in space



Lecture plan

1. Course organization
2. Finish looking at word vectors and word2vec
3. **Can we capture the essence of word meaning more effectively by counting?**
4. Evaluating word vectors
5. Word senses

Why not capture co-occurrence counts directly?

- We iterated through the whole corpus (perhaps many times) while doing a bunch of things (e.g., predicting outer context words, sampling random words, etc.).

- Why don't we simply count what words appear near each other?

- **Count-based** word vectors
  - Sometimes called "frequency-based" word vectors
  - Contrast with "prediction-based" word vectors (e.g., word2vec)

Example: Co-occurrence matrix

- Window length 1 (more common: 5~10)
- Symmetric (irrelevant whether left or right context)

- Example corpus:
  - I like deep learning
  - I like NLP
  - I enjoy flying

counts	I	like	enjoy	deep	learning	NLP	flying	.
I	0	2	1	0	0	0	0	0
like	2	0	0	1	0	1	0	0
enjoy	1	0	0	0	0	0	1	0
deep	0	1	0	0	1	0	0	0
learning	0	0	0	1	0	0	0	1
NLP	0	1	0	0	0	0	0	1
flying	0	0	1	0	0	0	0	1
.	0	0	0	0	1	1	1	0

Co-occurrence vectors

- Simple count co-occurrence vectors
  - Vectors increase in size with vocabulary
  - Very high dimensionality: Requires a lot of storage and data
    - Results could be less robust depending on what corpus you use.

- What we want: **Low-dimensional dense vectors**
  - Store “most” of the important information in a fixed, small number of dimensions
  - Usually 25~1,000 dimensions (similar to word2vec)
  - How can we reduce the dimensionality?
    - E.g., Singular Value Decomposition (SVD)

Efforts to improve co-occurrence vectors

- But… running an SVD on raw counts doesn't work well.
- Many efforts to improve co-occurrence vectors (e.g., Rohde et al., 2005)

- Scaling the counts can help ***a lot***
  - Problem: Function words are too frequent → Syntax has too much impact.
    - log the frequencies
    - Ignore the function words
    - …
- Using ramped windows that count closer words more than further away words
- Converting counts to word pair correlations

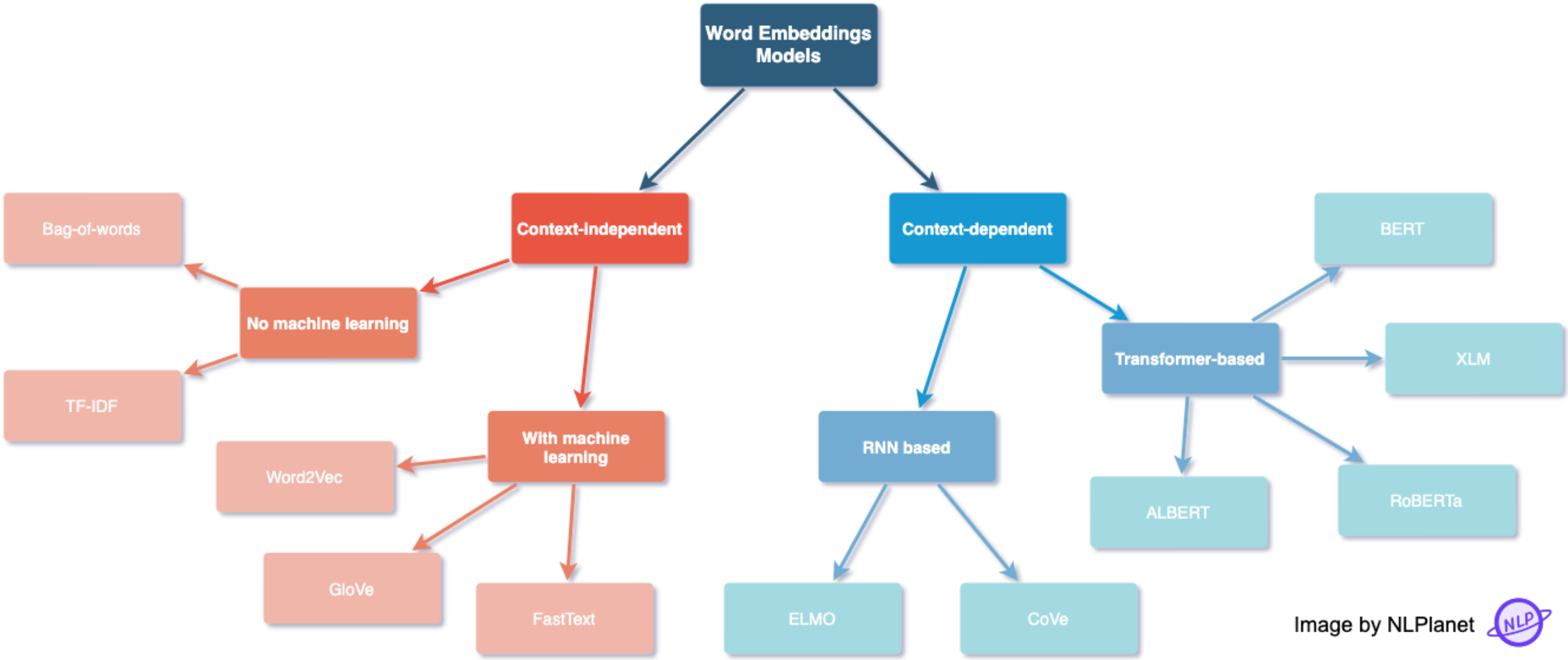
→ **Prediction-based embeddings (e.g., word2vec) perform better!**

An Improved Model of Semantic Similarity Based on Lexical Co-Occurrence (Rohde, Gonnerman, and Plaut, 2005)

Word2vec: Hmm… but still several question marks 🤔

- The model ignores the position of outer context words.
- The model is trained to give a reasonably high probability estimate to *all* words that occur in the context.
  - Strong frequency effect: If a word is common like function words (e.g., “in”, “the”, and “a”), then they are bound to have a reasonably high dot product with *all* words.
- Once trained, each word is assigned a fixed vector representation, regardless of their context (“**static embeddings**”).

Word embeddings in NLP



[Lecture] RNNs (Week 2-3)
[Lecture] Transformers (Week 5)

Two minutes NLP — 11 word embeddings models you should know

Lecture plan

1. Course organization
2. Finish looking at word vectors and word2vec
3. Can we capture the essence of word meaning more effectively by counting?
4. **Evaluating word vectors**
5. Word senses

How can we evaluate word vectors?

Evaluation in NLP: **Intrinsic** vs. **Extrinsic**

- **Intrinsic evaluation**
  - Evaluation on a specific, intermediate subtask
    - Directly test word vectors for syntactic or semantic relationships between words
  - + Fast to compute
  - - Not clear if it is actually helpful unless correlation to a real task is established

Evaluation methods for unsupervised word embeddings (Schnabel et al., 2015)

Intrinsic evaluation: Word vector analogies

- **Analogy questions**: "a is to b, as c is to ___"

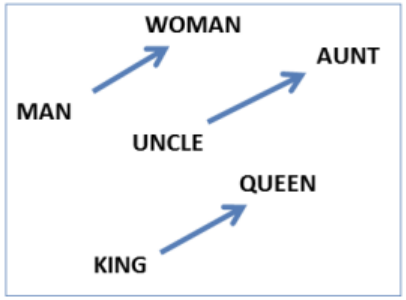
- Normalize word vectors (x_a, x_b, and x_c) to unit norm
- Compute $y = x_b - x_a + x_c$
- Search for the word whose vector has the greatest cosine similarity to y

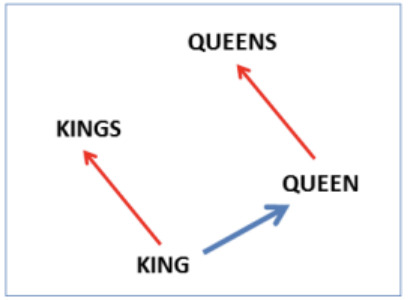
$$w^* = argmax_w \frac{x_w y}{||x_w|| \, ||y||}$$

- Limitation: Only capturing linear relation

a : b :: c : _

man : woman :: king : ?





Linguistic Regularities in Continuous Space Word Representations (Mikolov, Yih, and Zweig, 2013)

Intrinsic evaluation: Word meaning similarity

- Word vector distances and their **correlation with human judgments**
- Example: **WordSim353** **(Finkelstein et al., 2002)**

The WordSimilarity-353 Test Collection

Version: 1.0
Release date: February 10, 2002
Maintained by: Evgeniy Gabrilovich (gabr@cs.technion.ac.il)

Overview

The **WordSimilarity-353 Test Collection** contains two sets of English word pairs along with human-assigned similarity judgements. The collection can be used to train and/or test computer algorithms implementing semantic similarity measures (i.e., algorithms that numerically estimate similarity of natural language words).

Description

The first set (**set1**) contains 153 word pairs along with their similarity scores assigned by 13 subjects. The second set (**set2**) contains 200 word pairs, with their similarity assessed by 16 subjects. Subjects' names have been replaced by ordinal numbers (1..13, or 1..16) to protect their privacy; identical numbers in the two sets do not necessarily correspond to the same individual.

All the subjects in both experiments possessed near-native command of English. Their instructions were to estimate the *relatedness* of the words in pairs on a scale from 0 (totally unrelated words) to 10 (very much related or identical words). The precise instructions are available in file **instructions.txt** inside the ZIP archive (see section "Availability and usage" below).

Each set provides the raw scores assigned by each subject, as well as the mean score for each word pair. For convenience, a combined set (**combined**) is provided that contains a list of all 353 words, along with their mean similarity scores. The combined set is merely a concatenation of the two smaller sets.

Word 1	Word 2	Human (mean)
tiger	cat	7.35
tiger	tiger	10
book	paper	7.46
computer	internet	7.58
plane	car	5.77
professor	doctor	6.62
stock	phone	1.62
stock	CD	1.31
stock	jaguar	0.92

Placing Search in Context: The Concept Revisited (Finkelstein et al., 2002)

Intrinsic evaluation: Word meaning similarity

- Word vector distances and their **correlation with human judgments**

Model	Size	WS353	MC	RG	SCWS	RW
SVD	6B	35.3	35.1	42.5	38.3	25.6
SVD-S	6B	56.5	71.5	71.0	53.6	34.7
SVD-L	6B	65.7	72.7	75.1	56.5	37.0
CBOW†	6B	57.2	65.6	68.2	57.0	32.5
SG†	6B	62.8	65.2	69.7	58.1	37.2
GloVe	6B	65.8	72.7	77.8	53.9	38.1
SVD-L	42B	74.0	76.4	74.1	58.3	39.9
GloVe	42B	**75.9**	**83.6**	**82.9**	**59.6**	**47.8**
CBOW*	100B	68.4	79.6	75.4	59.4	45.5

GloVe: Global Vectors for Word Representation (Pennington, Socher, and Manning, 2014)

How can we evaluate word vectors?

Evaluation in NLP: **Intrinsic** vs. **Extrinsic**

- **Extrinsic evaluation**
  - Evaluation on a real task
    - Use word vectors as input features to a downstream task (e.g., information retrieval, document classification, question answering) and measure changes in performance metrics specific to that task
  - + Real task → direct evidence for improvement & usefulness

Evaluation methods for unsupervised word embeddings (Schnabel et al., 2015)

Extrinsic evaluation: Named entity recognition (NER)

- One example where good word vectors should help directly: **Named entity recognition**
  - Identify references to a person, organization, or location (e.g., "Mina Lee lives in Chicago.")

Model	Dev	Test	ACE	MUC7
Discrete	91.0	85.4	77.4	73.4
SVD	90.8	85.7	77.3	73.7
SVD-S	91.0	85.5	77.6	74.3
SVD-L	90.5	84.8	73.6	71.5
HPCA	92.6	**88.7**	81.7	80.7
HSMN	90.5	85.7	78.7	74.7
CW	92.2	87.4	81.7	80.2
CBOW	93.1	88.2	82.2	81.1
GloVe	**93.2**	88.3	**82.9**	**82.2**

- Subsequent NLP tasks in this class are other examples. So, more examples soon!

GloVe: Global Vectors for Word Representation (Pennington, Socher, and Manning, 2014)

How can we evaluate word vectors?

Evaluation in NLP: **Intrinsic** vs. **Extrinsic**

- **Extrinsic evaluation**
  - Evaluation on a real task
    - Use word vectors as input features to a downstream task (e.g., text classification, question answering) and measure changes in performance metrics specific to that task
  - + Real task → direct evidence for improvement & usefulness
  - - Can take a long time to evaluate

Evaluation methods for unsupervised word embeddings (Schnabel et al., 2015)

How can we evaluate word vectors?

Example: Suppose that you are building a **question answering** system

- Takes words as inputs and converts them to **word vectors**
- Uses the vectors as inputs for an elaborate machine learning system
  - Intent classification, document retrieval, span extraction, filtering, …
- Maps the output word vectors by this system back to words to produce answers

What would you do when:

- When tuning hyperparameters of word vectors (e.g., dimension, window size, corpus size/source, etc.)?
- When testing the question answering system before deployment?
- When you are unclear if the problem is this subsystem, another subsystem, or their interaction?

[Lecture] Benchmarking and evaluation (Week 6)

Lecture plan

1. Course organization
2. Finish looking at word vectors and word2vec
3. Can we capture the essence of word meaning more effectively by counting?
4. Evaluating word vectors
5. **Word senses**

Word senses and word sense ambiguity

- Most words have lots of meanings (i.e., word senses)!
  - Especially common words
  - Especially words that have existed for a long time

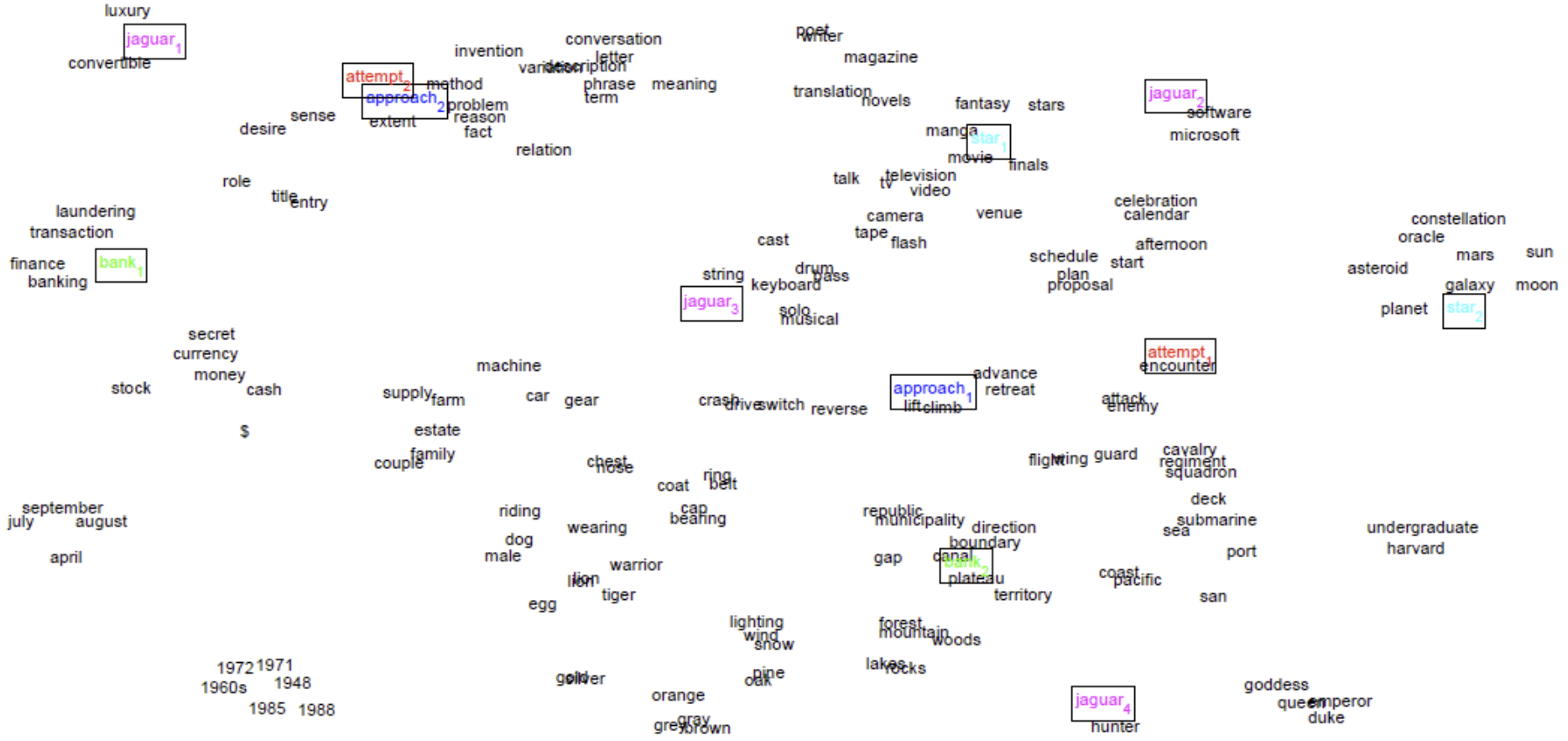
- Example: **close**

close

- to move so as to bar passage through something
- to bring to an end or period
- to bring or bind together the parts or edges of
- to contract, fold, swing, or slide so as to leave no opening
- to come together
- …
- being near in time, space, effect, or degree
- intimate, familiar
- strict, rigorous
- …
- a coming or bringing to a conclusion
- …

Do we need multiple vectors per word?

- Idea: Cluster word windows around words, retrain with each word assigned to multiple different clusters $bank_1$, $bank_2$, etc.



Improving Word Representations via Global Context and Multiple Word Prototypes (Huang et al., 2012)

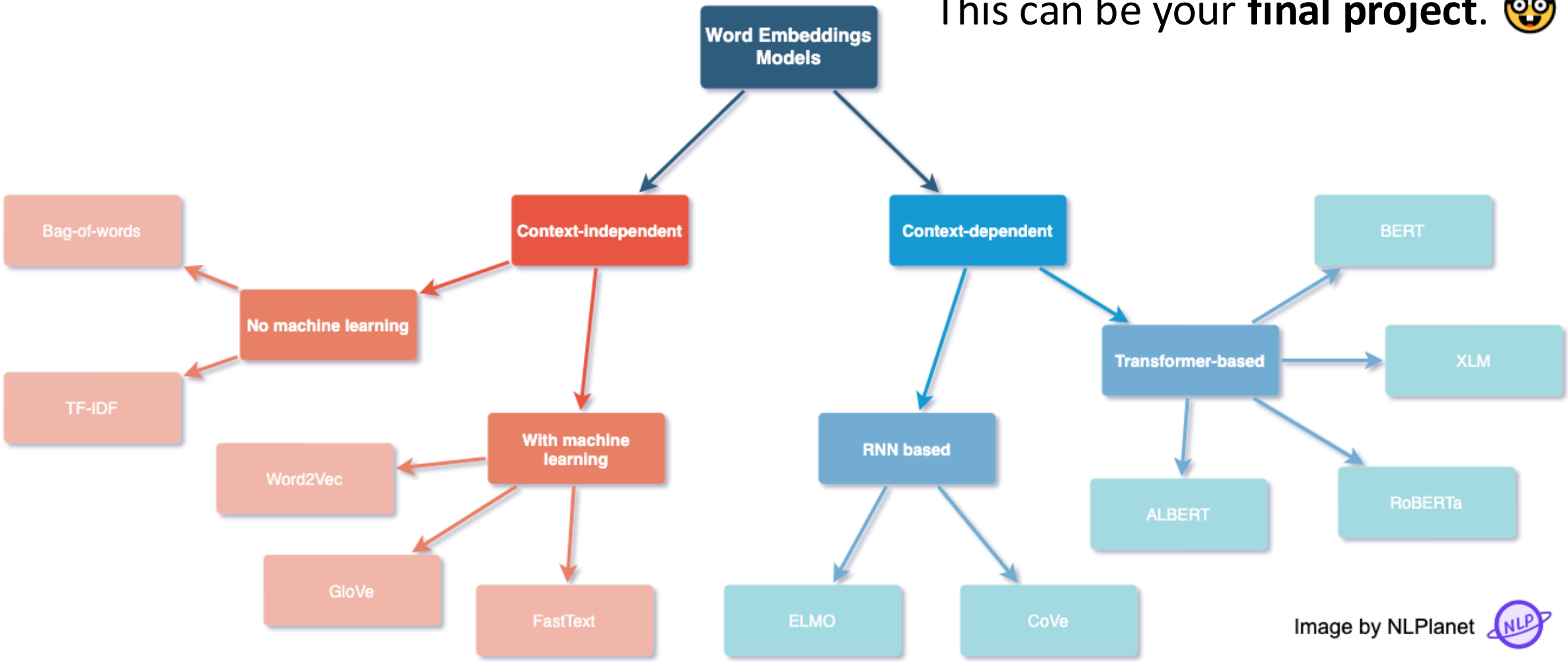
One vector can encode multiple senses!

- Multiple senses of a word reside in a **linear superposition** (weighted sum) in one vector
  - $v_{\text{close}} = \alpha_1 v_{\text{close}_1} + \alpha_2 v_{\text{close}_2} + \alpha_3 v_{\text{close}_3} + \ldots$
  - where $\alpha_1 = \frac{f_1}{f_1 + f_2 + f_3 + \ldots}$ for frequency f

Linear Algebraic Structure of Word Senses, with Applications to Polysemy (Arora et al., 2018)

Word embeddings in NLP

There are many variations of word embeddings!
This can be your **final project**.



[Lecture] RNNs (Week 2-3)
[Lecture] Transformers (Week 5)

Two minutes NLP — 11 word embeddings models you should know

$$\frac{\partial}{\partial v_c} \log\left(p(o|c)\right) = u_o - \frac{1}{\sum_{w=1}^{V} \exp(u_w^T v_c)} \cdot \left(\sum_{x=1}^{V} \exp(u_x^T v_c)\, u_x\right)$$

$$= u_o - \sum_{x=1}^{V} \frac{\exp(u_x^T v_c)}{\sum_{w=1}^{V} \exp(u_w^T v_c)}\, u_x$$

Distribute term across sum

$$= u_o - \sum_{x=1}^{V} p(x|c)\, u_x$$

This an expectation: average over all context vectors weighted by their probability

$$= \text{observed} - \text{expected}$$

This is just the derivatives for the center vector parameters
Also need derivatives for output vector parameters
(they're similar)
Then we have derivative w.r.t. all parameters and can minimize