NUS | Computing
National University of Singapore

CS4248: Natural Language Processing

Lecture 6 — Word Embeddings

Recap of Week 05

Text Classification — Probabilistic Classifiers

- Common goal: Learn $P(y|x)$
  - Learn $P(y|x)$ from the data
- Two basic approaches

(1) Generative Classifiers
- Learn joint probability $P(x, y)$
- Apply Bayes Rule to get $P(y|x)$

$$\rightarrow \hat{y} = \underset{y \in Y}{\operatorname{argmax}} \overbrace{P(x|y)P(y)}^{= P(x,y) \propto P(y|x)}$$

(2) Discriminative Classifiers
- Learn $P(y|x)$ directly

$$\rightarrow \hat{y} = \underset{y \in Y}{\operatorname{argmax}} P(y|x)$$

6

Overfitting — Intuition (Naive Bayes Classifier)

- Scenario — movie reviews
  - (Very) low number of reviews
  - NB classifier based on 4-grams

This movie drew me in, and it'll do the same to you.	positive
I can't tell you how much I hated this movie. It sucked.	negative
...	...

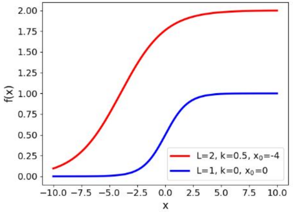
→ Effect of Naive Bayes classifier
- Each 4-gram most likely unique and associated with only 1 class
  (e.g., *"tell you how much"* only found in a negative review)
- Unseen positive review x containing "tell you how much" → $P(positive|x) = 0$

54

Logistic Regression

- Logistic Regression → Real-valued predictions interpreted as probability
  - Function *f* is the standard **Logistic Function** (Sigmoid function)

$$f(x) = \frac{L}{1 + e^{-k(x-x_0)}} \xrightarrow{L=1,\ k=1,\ x_0=0} f(x) = \frac{1}{1+e^{-x}}$$

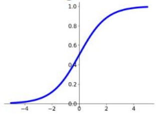


15

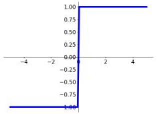
Neural Network — Activation Functions

- Wide range of activation functions
- Activations functions for hidden layers
  - Do not need to have a probabilistic interpretation
  - Only requirement: non-linear function!
  - Examples:

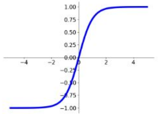
Sigmoid



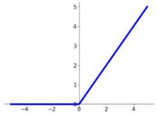
Sign



Tanh

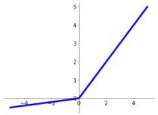


ReLU



ReLU = Rectified Linear Unit

Leaky ReLU



81

Announcements

- Project
  - Intermediate Update — Deadline: 7 Mar, 23:59
  - Template available here: https://bit.ly/cs4248-2320-iu-template
    (you don't have to use the template, but please use a 16:9 aspect ratio for your slides)

- Assignment 2
  - Reminder — Deadline: 9 Mar, 23:59
  - Goal: practice manual feature engineering
  - To be fair, only certain technologies already covered are allowed

Outline

- **Motivation**
- Sparse Word Embeddings
  - Co-occurrence Vectors
  - Discussion & Limitations
- Dense Word Embeddings
  - Basic Idea
  - Word2Vec (CBOW & Skipgram)
  - Negative Sampling
  - Basic Properties
  - Practical Considerations & Limitations
- NLP Ethics

Embeddings in One Slide

- We want good language (word) representations
  - Language modeling a good start. This used purely statistics.
  - Can we use Supervised ML (NNs) to do this task?

- Yes! Define word prediction as a task that LMs do already
  - Although a supervised task, we don’t need to provide labeled data for this problem
  - Learn and tune a good representation

- Leverage advantages of NNs to enhance this representation
  - Make dense vectors instead of sparse ones
  - Use good approximations instead of exact solutions

Motivation

- Recall from Lecture 4: Most NLP algorithms require
  - Numerical input
  - Standardized input

  ➜ Most common representation: **vectors** (a.k.a. **embeddings**)

- So far: Vector Space Model (VSM)
  - Vector representation of <u>documents</u>
  - Document vector for document d_i = column document–term matrix (typically using weighting schemes, e.g., tf-idf)

Document-Term Matrix

	d_1	d_2	d_3	d_4	d_5
car	0	0	0.4	0	0.4
cat	0.22	0.29	0	0	0.22
chase	0.22	0.22	0.22	0	0
dog	0.29	0	0	0.29	0.22
sit	0	0	0	0	0.7
tv	0	0	0.4	0.4	0
watch	0	0	0	0.7	0

🤔 How to represent <u>words</u> as vectors?

Representing Words — Traditional NLP

- Words as discrete symbols: **One-Hot Encoding**
  - Length of vector = size of vocabulary V (number of unique words)
  - Vector values: 1 if dimension reflects word, 0 otherwise

Note: VSM document vector = aggregation over word vectors (with some weighting, e.g., sum, tf-idf)

- Toy example
  - V = {dog (w_1), cat (w_2), lion (w_3), bear (w_4), cobra (w_5), cow (w_6), frog (w_7), ...}

	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	...	$w_{\|V\|}$
dog	**1**	0	0	0	0	0	0	0	0	...	0
cat	0	**1**	0	0	0	0	0	0	0	...	0
lion	0	0	**1**	0	0	0	0	0	0	...	0
bear	0	0	0	**1**	0	0	0	0	0	...	0
...	...	...	...	...	...	...	...	...	...	...	...

one-hot vector of *"cat"*

Symbolic Representation of Words — Limitation

*"I saw a **cat**."*

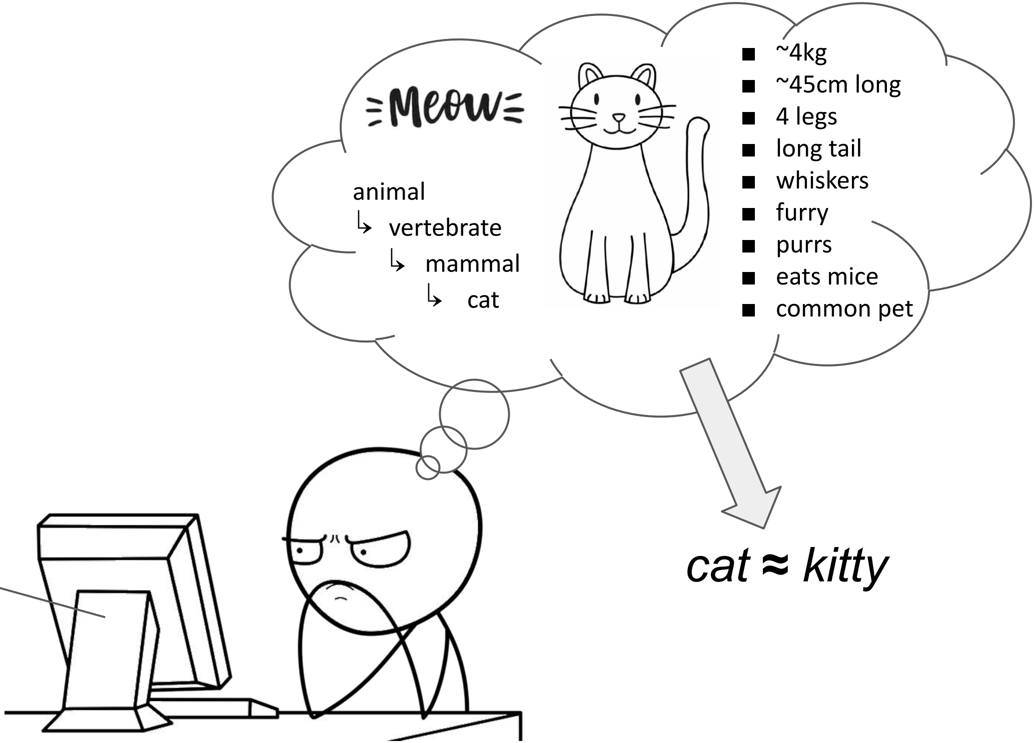
vs.

*"I saw a **kitty**."*

cat	=	[0 0 0 0 … 0 0 0 0 1 0 0 0 0 0 0]
kitty	=	[0 0 1 0 0 0 0 0 0 0 0 0 … 0 0 0]

orthogonal
word vectors

cat ≠ kitty



cat ≈ kitty

Symbolic Representation of Words — Limitation

- Problem: **No notion of similarity**
  - Words are just labels without meaning
  - Different words (syntax) ➜ orthogonal word vectors
    (even for words with the same/similar meaning)

cat ≠ kitty
fast ≠ quick
lift ≠ elevator
funny ≠ amusing
hotel ≠ lodging

- Goal: Similar words (meaning) ➜ similar word vectors
  - Word vectors no longer just labels but also encode "some" meaning
  - Improve basically all NLP tasks!

🤔 To think about: What are good embeddings, and how can we find them?

Distributional Hypothesis

"The meaning of a word is its use in the language."

(Wittgenstein, 1953)

"If A and B have almost identical environments [...], we say they are synonyms"

(Harris, 1954)

"You shall know a word by the company it keeps."

(Firth, 1957)

🏃‍♀️🏃🏃‍♂️ Quick Quiz

Outline

- Motivation
- **Sparse Word Embeddings**
  - **Co-occurrence Vectors**
  - Discussion & Limitations
- Dense Word Embeddings
  - Basic Idea
  - Word2Vec (CBOW & Skipgram)
  - Negative Sampling
  - Basic Properties
  - Practical Considerations & Limitations
- NLP Ethics

Pre-Lecture Activity from Last Week

- Assigned Task
  - Post a 1–2 sentence answer to the following question into your Tutorial Group’s discussions
    (you will find the thread on Canvas > Discussions)

“What do we mean by sparse or dense vectors?
Are documents characterised by tf-idf sparse or dense?”

Side notes:
- This task is meant as a warm-up to provide some context for the next lecture
- No worries if you get lost; we will talk about this in the next lecture
- You can just copy-&-paste others' answers, but his won't help you learn better


CONFIDENTIAL
TOP SECRET
CONFIDENTIAL

Revisiting the Document–Term Matrix (DTM)

- Word vectors derived from DTM
  - Assumption: context of word *w* = set of documents containing *w*
  - In principle, valid word vectors

- Problem
  - Assumption does not capture distributional hypothesis
  - Not used in practice

	d_1	d_2	d_3	d_4	d_5
car	0	0	0.4	0	0.4
cat	0.22	0.29	0	0	0.22
chase	0.22	0.22	0.22	0	0
dog	0.29	0	0	0.29	0.22
sit	0	0	0	0	0.7
tv	0	0	0.4	0.4	0
watch	0	0	0	0.7	0

word vector of *"cat"*

🏃‍♀️🏃🏃‍♂️ Document–Term Matrix

Let's focus on these four words in our previous Shakespeare example.

What do we observe?

	As You Like It	***Twelfth Night***	***Julius Caesar***	***Henry V***
battle	1	1	8	15
soldier	2	2	12	36
fool	37	58	1	9
clown	6	117	0	0

Co-Occurrence Vectors

- Basic idea
  - Context of a word *w* = (small) window of words surrounding *w*
  - Count how often a word *w* occurs with another (w.r.t. the context of *w*)

context

	w_1	w_2	w_3	…	$w_{\|V\|}$
w_1					
w_2					
w_3					
…				c_{ij}	
$w_{\|V\|}$					

➜ **Term–Context Matrix**

The number of times w_j was in the context of w_i

Word vector of w_i

Term–Context Matrix — Toy Example

...has shown that the movie rating reflects to overall quality...

...the cast of the show turned in a great performance and...

...is to get nlp data for ai algorithms on a large scale...

...only with enough data can ai find reliable patterns to be effective…

…

	aardvark	***rating***	***story***	***data***	***cast***	***result***	**…**
movie	0	2	4	0	1	0	
show	0	6	3	0	2	1	
nlp	0	0	1	3	0	4	
ai	0	1	0	5	0	2	

movie ≈ show

nlp ≈ ai

Term–Context Matrix

- Problems with raw counts: Often very skewed
  - e.g., *"the"* and *"of"* are very frequent, but typically not very discriminative

➜ Alternative: **Pointwise Mutual Information** (PMI)

- Do words w_i and w_j co-occur more than if they were independent?

$$PMI(w_i, w_j) = \log_2 \frac{P(w_i, w_j)}{P(w_i)P(w_j)}$$

Oops, PMI can be < 0, but no good intuition for negative values for word vectors

➜ **Positive Pointwise Mutual Information** (PPMI)

$$PPMI(w_i, w_j) = max\left(\log_2 \frac{P(w_i, w_j)}{P(w_i)P(w_j)}, 0\right)$$

PPMI Matrix — Toy Example

Assume this is the complete term–context matrix

	rating	story	data	cast	result
movie	2	4	0	1	0
show	6	3	0	2	1
nlp	0	1	3	0	4
ai	1	0	5	0	2

$$P(w{=}movie, c{=}cast) = 1/35 = 0.03$$

$$P(w{=}movie) = 7/35 = 0.2$$

$$P(c{=}cast) = 3/35 = 0.09$$

	rating	story	data	cast	result	P(w)
movie	0.06	0.11	0	0.03	0	0.20
show	0.17	0.09	0	0.06	0.03	0.34
nlp	0	0.03	0.09	0	0.11	0.23
ai	0.03	0	0.14	0	0.06	0.23
P(context)	0.26	0.23	0.23	0.09	0.20	

$$PPMI(w{=}movie, c{=}cast) = \log_2 \frac{0.03}{0.09 \cdot 0.2} = 0.74$$

	rating	story	data	cast	result
movie	0.15	1.32	0	0.74	0
show	0.96	0.13	0	0.96	0
nlp	0	0	0.71	0	1.32
ai	0	0	1.45	0	0.32

PPMI Word Vectors — Discussion

- Various refinements to handle (very) rare words
  - Raise context probabilities
  - Use Add-1 Smoothing

similar effects

- Consideration: Sparsity
  - Matrix is of size $|V| \times |V|$ (|*V*| typically between 20k and 50k)
  - PPMI word vectors are very sparse (most vector entries are 0)

Outline

- Motivation
- Sparse Word Embeddings
  - Co-occurrence Vectors
  - Discussion & Limitations
- **Dense Word Embeddings**
  - **Basic Idea**
  - Word2Vec (CBOW & Skipgram)
  - Negative Sampling
  - Basic Properties
  - Practical Considerations & Limitations
- NLP Ethics

Why Dense Word Vectors?

- Important practical benefits of dense vectors
  - More convenient features: less weights to tune, lower risk of overfitting
  - Tend to generalize better than features derived from counts
  - Tend to better capture synonymy than sparse vectors

Example sparse vector: [0 0 0 0 0.8 0 0 … 0 0 0 0 1.2 0 … 0 0]

(e.g., for the word "actor")

"movie" **"film"**

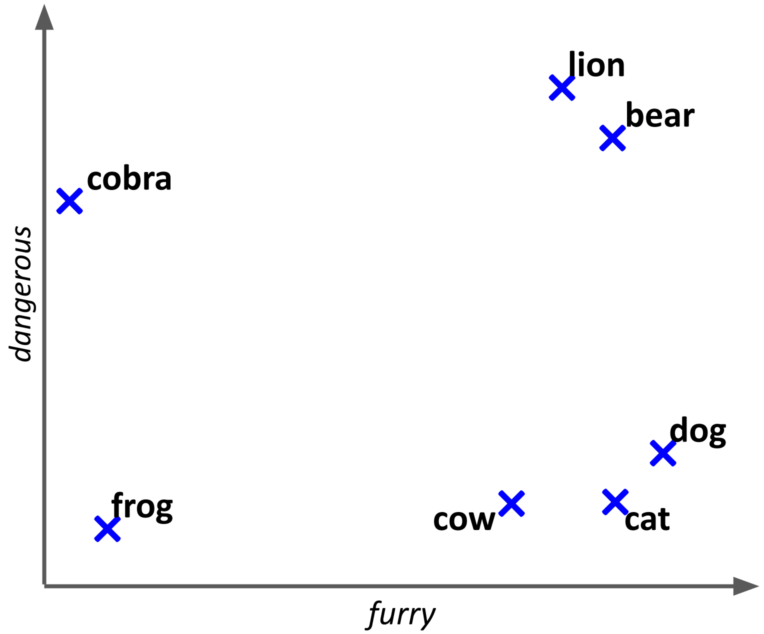
Each word represents a distinct dimension; fails to capture similarity between words

- Dense vector in practice
  - Common dimensions: 100 to 1,000 entries
  - Most to all vector elements are non-zero

Dense Word Vectors — Intuition

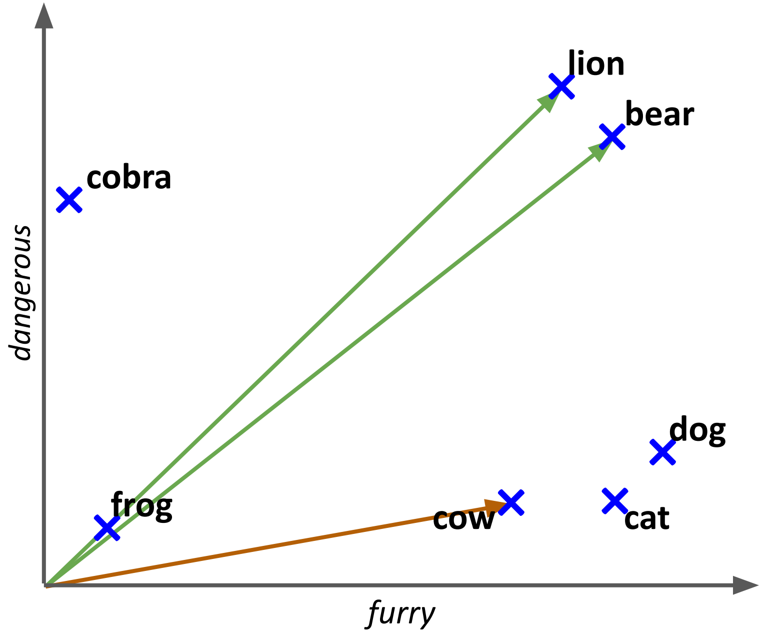
- Toy example: custom encoding with 2 dimensions
  - Each dimension represent a property shared between words

	furry	dangerous
dog	0.90	0.15
cat	0.85	0.10
lion	0.80	0.95
bear	0.85	0.90
cobra	0.0	0.80
cow	0.75	0.10
frog	0.05	0.05
...	...	...



What about “movie”, “dignity”, “cake”, …?

Dense Word Vectors — Intuition



Using suitable similarity metric

- $\text{sim}(w_{lion}, w_{bear}) = 1.54$
- $\text{sim}(w_{lion}, w_{cow}) = 0.70$

This notion of similarity between words is what we are after!

- Problems with custom encoding
  - How to decide on the dimensions?
  - How to decide on the values?

Manual assignment simply impractical/impossible!

➜ Need for automated methods

Outline

- Motivation
- Sparse Word Embeddings
  - Co-occurrence Vectors
  - Discussion & Limitations
- **Dense Word Embeddings**
  - Basic Idea
  - **Word2Vec (CBOW & Skipgram)**
  - Negative Sampling
  - Basic Properties
  - Practical Considerations & Limitations
- NLP Ethics

Basic Approaches

- Popular alternatives (but not covered here)
  - Singular Value Decomposition (SVD; matrix factorization)
  - Brown Clustering

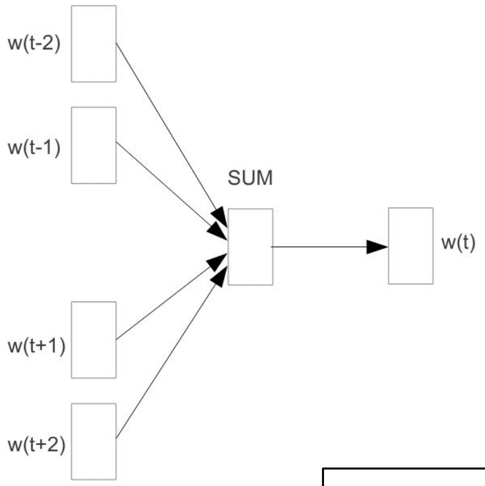
- Neural Network-based Methods
  - Inspired by (Neural) Language Models
  - Learn embeddings as part of the process of word prediction
  - Typically fast & easy to train
  - In the following: **Word2Vec**

Word2Vec encompasses 2 network architectures: **CBOW** & **Skip-gram**

Word2Vec: CBOW & Skip-Gram

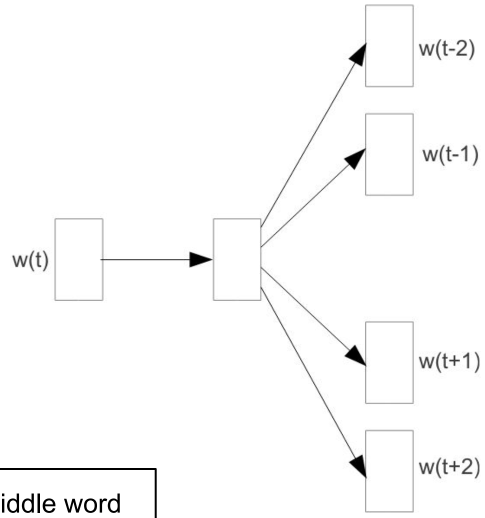
Continuous Bag of Words (CBOW)

Given context ➜ Predict word



Skip-gram

Given word ➜ Predict context



Context = window of words surrounding the middle word

Source: [Exploiting Similarities among Languages for Machine Translation]

🏃‍♀️🏃🏃‍♂️ CBOW — Predicting a Word from Context

“watching funny ______ on netflix”

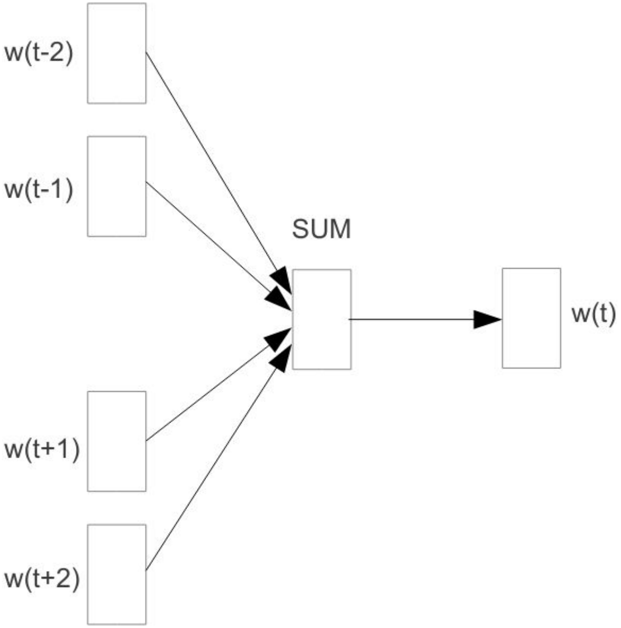
watching

funny

______

on

netflix



What seem like good predictions?

across

movies

quickly

lectures

old

clowns

shows

of

stairs

cars

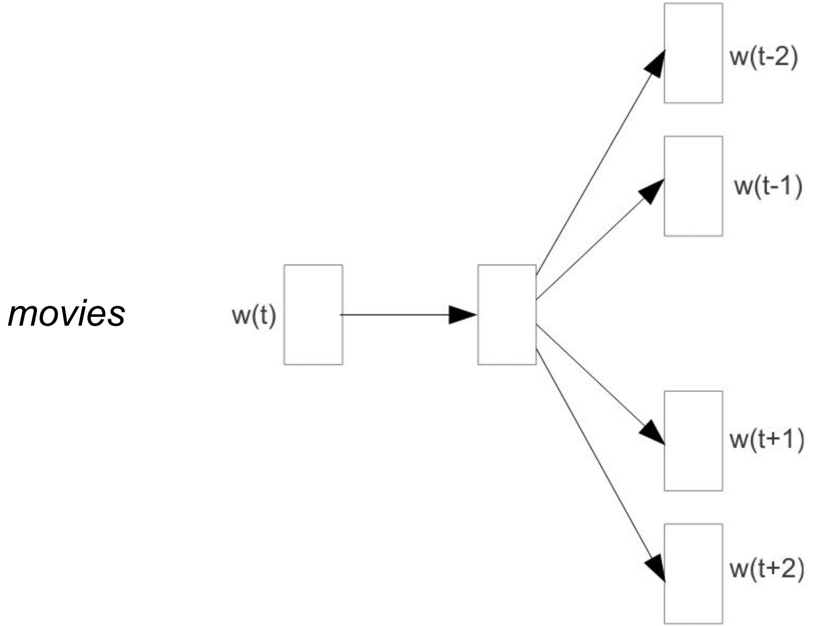
apples

cakes

CBOW — Predicting a Word from Context

🏃‍♀️🏃🏃‍♂️ Skip-Gram — Predicting Context from a Word

“______ ______ movies ______ ______”



What seem like good predictions?

funny
tomato
cinema
in
street
road
ticket
boring
lectures
lectures

Skip-Gram — Predicting a Context from a Word

Outline

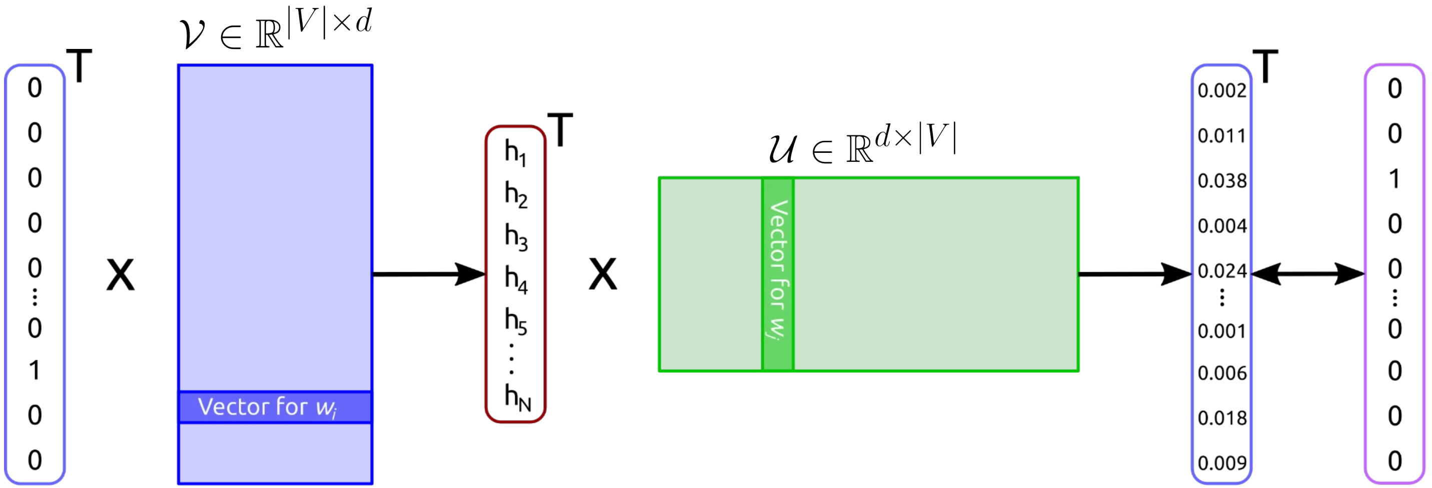
- Motivation
- Sparse Word Embeddings
  - Co-occurrence Vectors
  - Discussion & Limitations
- **Dense Word Embeddings**
  - Basic Idea
  - **Word2Vec (CBOW & Skipgram)**
  - Negative Sampling
  - Basic Properties
  - Practical Considerations & Limitations
- NLP Ethics

🏃‍♀️🏃🏃‍♂️ Neural Word Prediction

Word2Vec — Basic Setup (CBOW & Skip-gram)

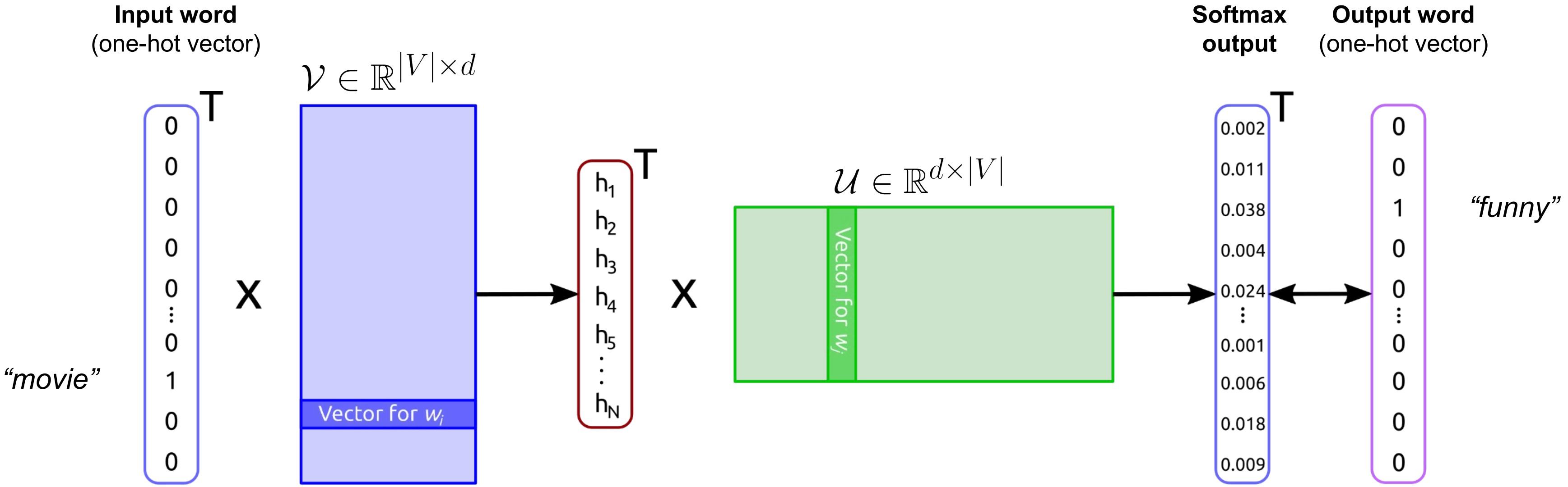
- Define two matrices
  - $\mathcal{V} \in \mathbb{R}^{|V| \times d}$ input embedding matrix
  - $\mathcal{U} \in \mathbb{R}^{d \times |V|}$ output embedding matrix
  - Given a word w_i, let $v_i \in \mathcal{V}$ and $u_i \in \mathcal{U}$ be the input and output embedding of w_i

Note that Word2Vec learns <u>2 embeddings</u> for each word!

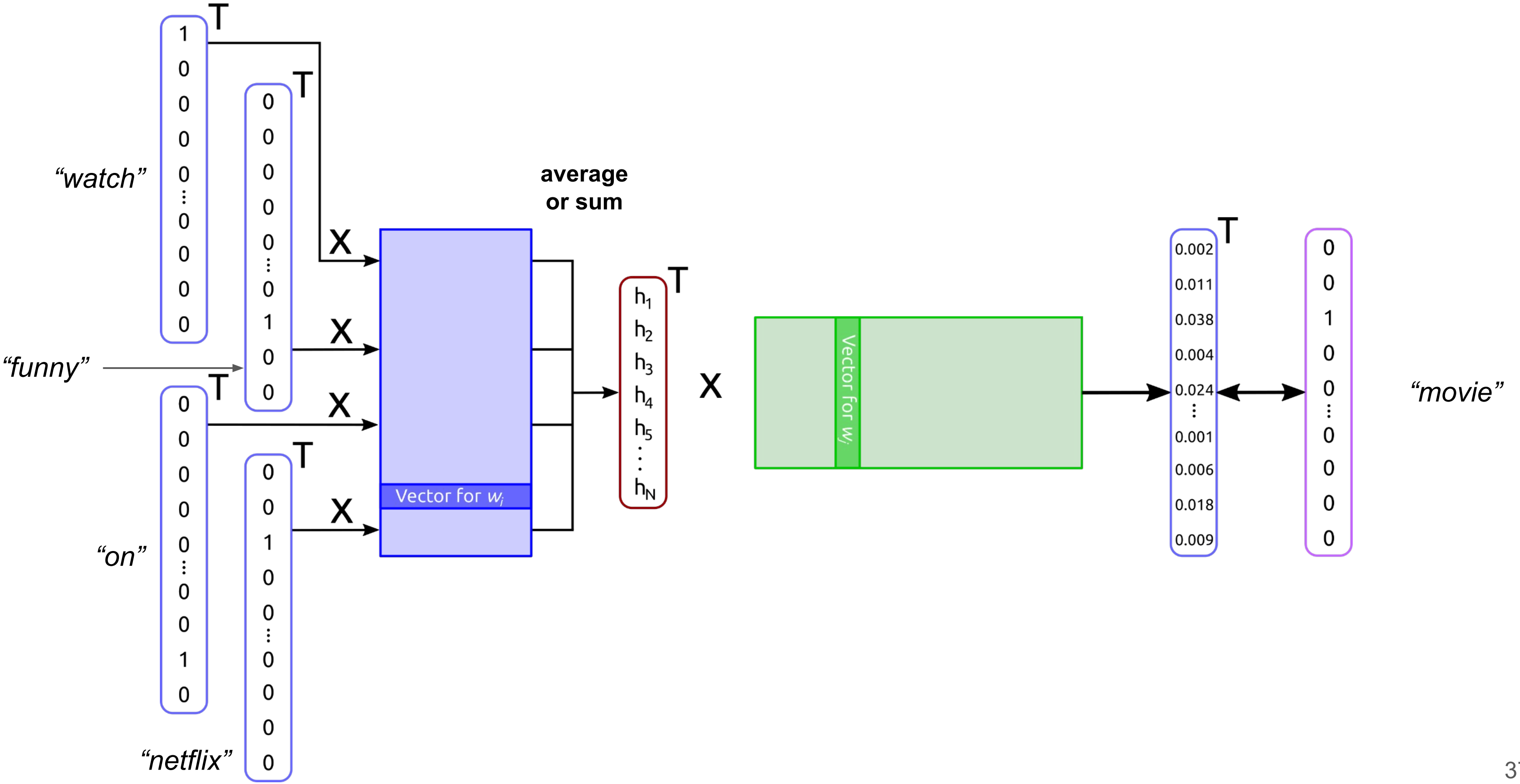


Word2Vec — Basic Setup (CBOW & Skip-gram)

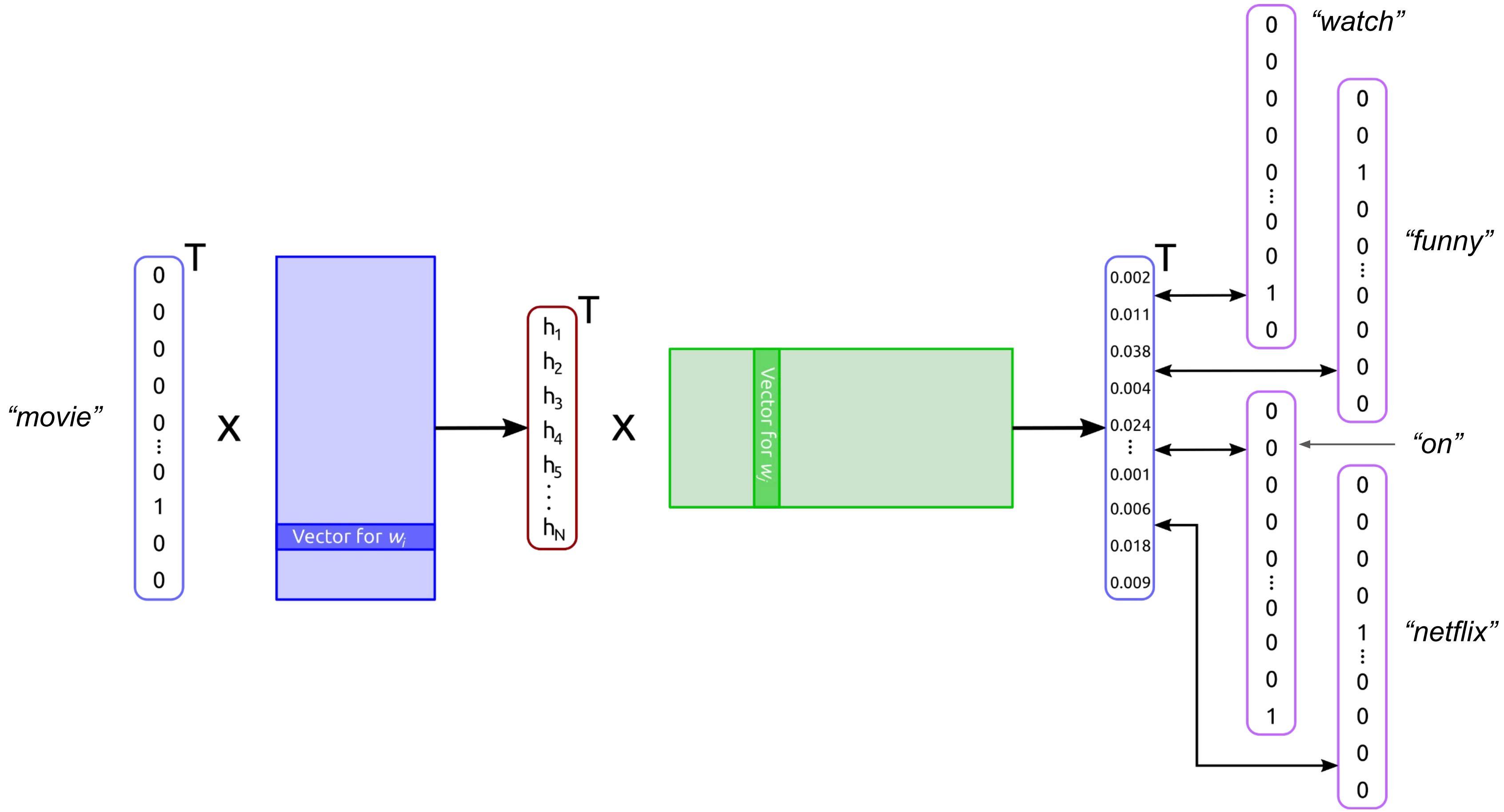
- Prediction task: 1 input word w_i, 1 output word w_o (both as one-hot vectors)
  - $w_i^\top \cdot V$ ➜ v_i (note: one-hot vector multiplied with a matrix is just a row "lookup")
  - $\mathrm{softmax}(v_i^T \cdot U)$ ➜ Probability $P(w|w_i)$ for all $w \in V$



Word2Vec — CBOW (window size m=2)



Word2Vec — Skip-Gram (window size m=2)



Training Objective — Loss Function

Continuous Bag of Words (CBOW)

$$L = -\log P(w_c \mid w_{c-m}, \ldots, w_{c-1}, w_{c+1}, \ldots, w_{c+m})$$

Sum up vectors of context words (BoW!)

$$= -\log P(u_c \mid \tilde{v}) \text{ , with } \tilde{v} = \sum_{\substack{-m \leq j \leq m \\ j \neq 0}} v_{c+j}$$

$$P(u_c \mid \tilde{v}) = \frac{\exp(u_c^\top \cdot \tilde{v})}{\sum_{j=1}^{|V|} \exp(u_j^\top \cdot \tilde{v})}$$

Skip-gram

$$L = -\log P(w_{c-m}, \ldots, w_{c-1}, w_{c+1}, \ldots, w_{c+m} \mid w_c)$$

Utilize conditional independence (BoW!)
+ law of logarithms

$$= -\sum_{\substack{-m \leq j \leq m \\ j \neq 0}} \log P(u_{c+j} \mid v_c)$$

$$P(u_{c+j} \mid v_c) = \frac{\exp(u_{c+j}^\top \cdot v_c)}{\sum_{j=1}^{|V|} \exp(u_j^\top \cdot v_c)}$$

Training Objective — Intuition

- Main objective for Skip-gram (for CBOW, it's just mirrored")

$$P(u_{c+j} \mid v_c) = \frac{\exp(u_{c+j}^\top \cdot v_c)}{\sum_{j=1}^{|V|} \exp(u_j^\top \cdot v_c)}$$

$P(u_{c+j}|v_c)$ larger

⇧

Dot product $u_{c+j}^\top \cdot v_c$ is higher

⇧

Vectors u_{c+j} and v_c are more similar

- Goal of training
  - Make vectors of center words close to vectors of their context words

➜ Vectors of words with similar contexts will be close

Intermediate goal

Main goal

Getting the Word Embeddings

- Learning $\mathcal{U}$ and $\mathcal{V}$
  - Minimize loss using Gradient Descent (or similar optimization technique)
  - All trainable / learnable parameters are in $\mathcal{U}$ and $\mathcal{V}$

- Which are the final embeddings? (recall, both matrices contain embeddings for each word)
  - Use only $\mathcal{U}$
  - Use only $\mathcal{V}$
  - Use average of $\mathcal{U}$ and $\mathcal{V}$

Outline

- Motivation
- Sparse Word Embeddings
  - Co-occurrence Vectors
  - Discussion & Limitations
- **Dense Word Embeddings**
  - Basic Idea
  - **Word2Vec (CBOW & Skipgram)**
  - Negative Sampling
  - Basic Properties
  - Practical Considerations & Limitations
- NLP Ethics

Word2Vec — Real-World Example (but on a very small scale)

- Setup & training
  - 50k movie reviews from IMDB
    (Source: https://ai.stanford.edu/~amaas/data/sentiment/)
  - Dataset preparation (window size *m*=2)

    Now-word removal, lowercase, lemmatization, consider only 20k most frequent words

    Treat all whole dataset as a single string
    (i.e., context windows cross sentence boundaries)

“watching funny movie on netflix”

x	y
watch funny on netflix	movie

➜ **1 CBOW sample**

x	y
movie	watch
movie	funny
movie	on
movie	netflix

➜ **2m Skip-gram samples**

PyTorch implementation of CBOW and Skip-gram

```
class CBOW(nn.Module):

    def __init__(self, vocab_size, embed_dim):
        super(CBOW, self).__init__()
        self.embeddings = nn.Embedding(vocab_size, embed_dim)
        self.linear = nn.Linear(embed_dim, vocab_size)

    def forward(self, contexts):
        x = self.embeddings(contexts)
        x = x.mean(axis=1)
        x = self.linear(x)
        x = F.log_softmax(x, dim=1)
        return x
```

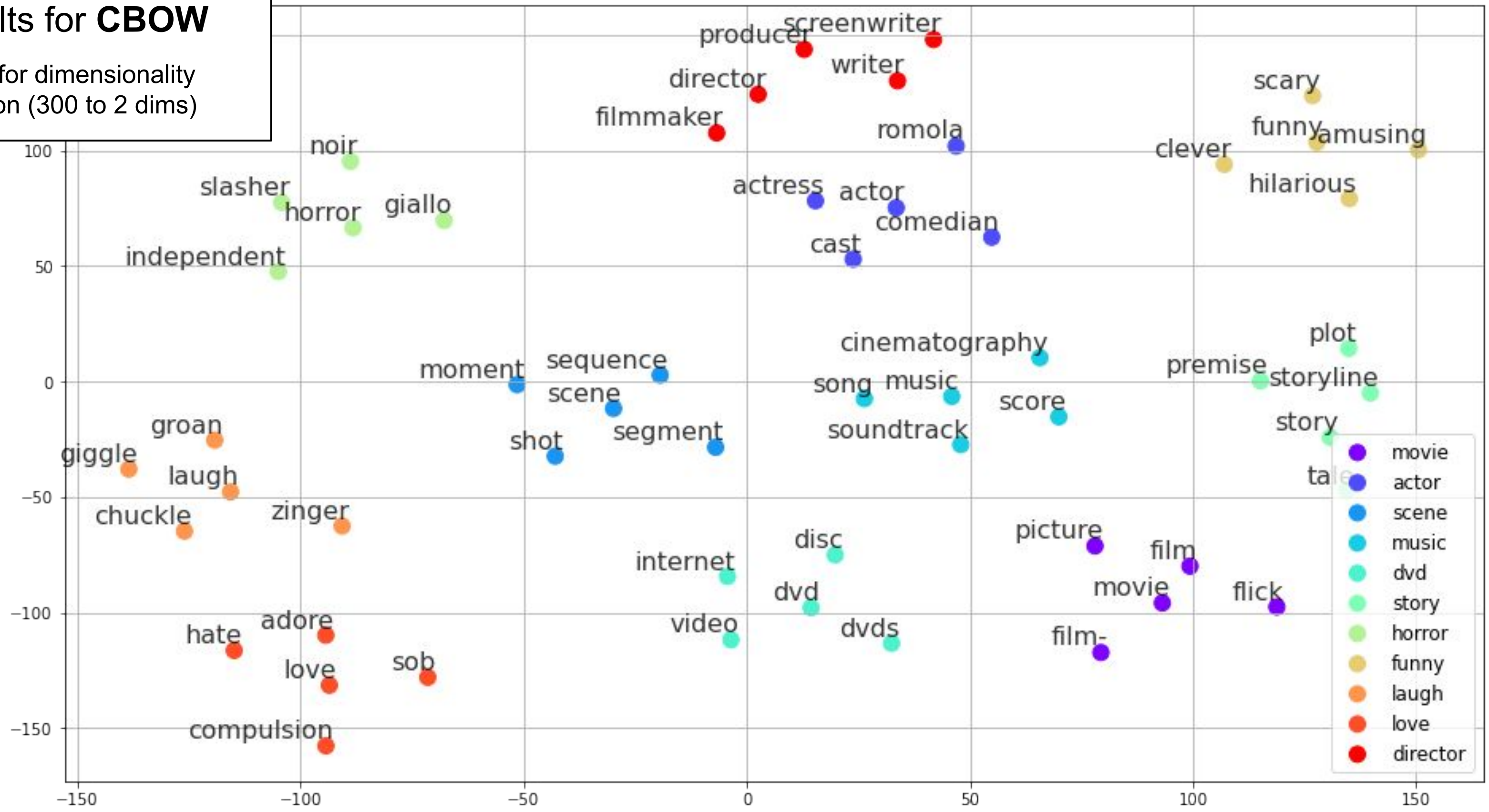
```
class Skipgram(nn.Module):

    def __init__(self, vocab_size, embed_dim):
        super(Skipgram, self).__init__()
        self.embeddings = nn.Embedding(vocab_size, embed_dim)
        self.linear = nn.Linear(embed_dim, vocab_size)

    def forward(self, inputs):
        x = self.embeddings(inputs)
        x = self.linear(x)
        x = F.log_softmax(x, dim=1)
        return x
```

Word2Vec — Real-World Example

Quick Quiz: Can you already spot some issues here?

Results for **CBOW**

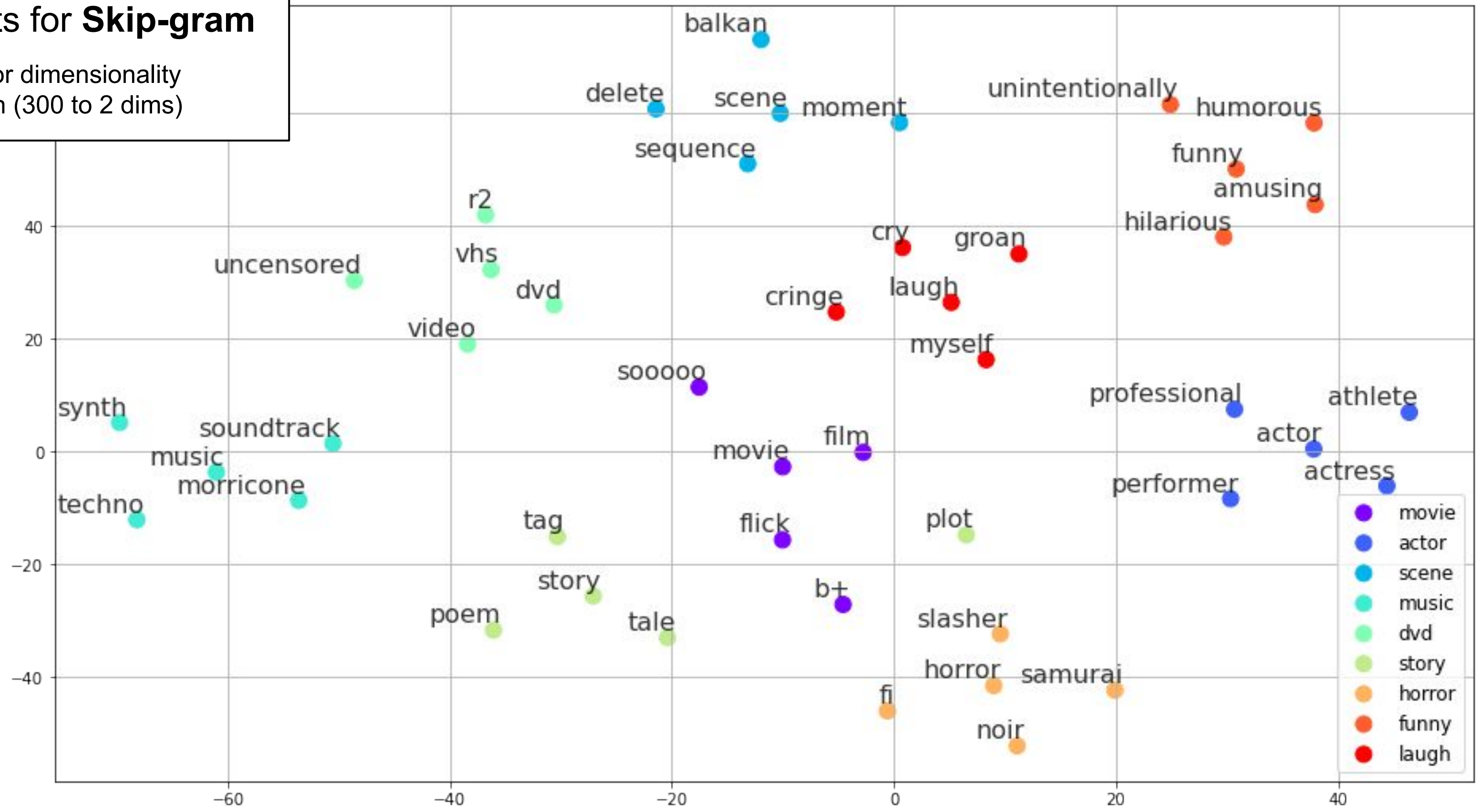
T-SNE for dimensionality reduction (300 to 2 dims)



Word2Vec — Real-World Example

Results for **Skip-gram**

T-SNE for dimensionality reduction (300 to 2 dims)



Break

Outline

- Motivation
- Sparse Word Embeddings
  - Co-occurrence Vectors
  - Discussion & Limitations
- **Dense Word Embeddings**
  - Basic Idea
  - Word2Vec (CBOW & Skipgram)
  - **Negative Sampling**
  - Basic Properties
  - Practical Considerations & Limitations
- NLP Ethics

Word2Vec — Tweaks for Word2Vec

Observation regarding training performance

1. Efficiency
   - Basic training objective includes a Softmax
   - Normalization over entire(!) vocabulary
     (to ensure a valid probability distribution of outputs)
   - Each sample potentially tweaks all(!) weights
     (all elements in embedding matrices $\mathcal{V}$ and $\mathcal{U}$)

$$P(u_{c+j} \mid v_c) = \frac{\exp(u_{c+j}^\top \cdot v_c)}{\sum_{j=1}^{|V|} \exp(u_j^\top \cdot v_c)}$$

2. Effectiveness
   - We use both positive and negative samples
   - Some samples help more than others

Word2Vec — Negative Sampling

1. Efficiency
   - Subsample a smaller batch of weights to update

$$P(u_{c+j} \mid v_c) = \frac{\exp(u_{c+j}^\top \cdot v_c)}{\sum_{j=1}^{|V|} \exp(u_j^\top \cdot v_c)}$$

2. Effectiveness
   - Pick informative samples more often than uninformative ones

➜ **Negative Sampling** (in the following: **SGNS** — Skip-Gram with Negative Sampling)

- Convert training from a word prediction task to a binary classification task

Word2Vec — Negative Sampling

Does this look familiar?

- Negative sampling — illustration
  - Window size *m*=2

"watching funny movie on netflix"

Data samples for (basic) Skip-gram

x	y
movie	watch
movie	funny
movie	on
movie	netflix

$2m$ samples

Data samples for SGNS

x	y
(movie, watch)	1
(movie, funny)	1
(movie, on)	1
(movie, netflix)	1
(movie, cake)	0
(movie, nlp)	0
(movie, soccer)	0
(movie, barely)	0
(movie, cluster)	0
(movie, morpheme)	0
(movie, traffic)	0
(movie, nimble)	0

$2m$ positive samples

$2mk$ negative samples

🏃‍♀️🏃🏃‍♂️ Double Negative

Word2Vec — Negative Sampling

- Selection of negative samples
  - Essentially at random (error of picking a "wrong" negative sample is negligible)
  - To increase probability of rare words: Sampling using (α-)weighted unigram frequency

occurrences of word w_i

$$P_\alpha(w_i) = \frac{Count(w_i)^\alpha}{\sum_{w \in V} Count(w)^\alpha} \qquad 0 < \alpha \leq 1$$

Probability of word w_i to be selected as a negative sample

SGNS — Training Objective

$$P(+|c,m) = \frac{1}{1+\exp(u_m^\top v_c)}$$

Let's assume this a given mini batch *B*

(movie, watch)	1
(movie, funny)	1
(movie, on)	1
(movie, netflix)	1
(movie, cake)	0
(movie, nlp)	0
(movie, soccer)	0
(movie, barely)	0
(movie, cluster)	0
(movie, morpheme)	0
(movie, traffic)	0
(movie, nimble)	0

B_{pos} (rows 1–4)

B_{neg} (rows 5–12)

$$L = -\log\left[\prod_{(c,m)\in B_{pos}} P(+|c,m) \cdot \prod_{(c,m)\in B_{neg}} P(-|c,m)\right]$$

$$= -\log\left[\prod_{(c,m)\in B_{pos}} P(+|c,m) \cdot \prod_{(c,m)\in B_{neg}} (1-P(+|c,m))\right]$$

$$= -\left[\sum_{(c,m)\in B_{pos}} \log P(+|c,m) + \sum_{(c,m)\in B_{neg}} \log\left(1-P(+|c,m)\right)\right]$$

SGNS — Training Objective

$$1 - \frac{1}{1+e^{-a}} = \frac{1+e^{-a}}{1+e^{-a}} - \frac{1}{1+e^{-a}} = \frac{e^{-a}}{1+e^{-a}} = \frac{1}{1+e^{a}}$$

Let's assume this a given mini batch *B*

(movie, watch)	1
(movie, funny)	1
(movie, on)	1
(movie, netflix)	1
(movie, cake)	0
(movie, nlp)	0
(movie, soccer)	0
(movie, barely)	0
(movie, cluster)	0
(movie, morpheme)	0
(movie, traffic)	0
(movie, nimble)	0

B_{pos} (first four rows)

B_{neg} (last eight rows)

$$L = -\left[\sum_{(c,m)\in B_{pos}} \log \frac{1}{1+\exp(-u_m^\top v_c)} + \sum_{(c,m)\in B_{neg}} \log\left(1 - \frac{1}{1+\exp(-u_m^\top v_c)}\right)\right]$$

$$= -\left[\sum_{(c,m)\in B_{pos}} \log \frac{1}{1+\exp(-u_m^\top v_c)} + \sum_{(c,m)\in B_{neg}} \log \frac{1}{1+\exp(u_m^\top v_c)}\right]$$

$$= -\left[\sum_{(c,m)\in B_{pos}} \log \sigma(-u_m^\top v_c) + \sum_{(c,m)\in B_{neg}} \log \sigma(u_m^\top v_c)\right]$$

SGNS — Parameters

- Sampling method to generate negative samples
  - e.g., subsampling to ignore very frequent words

- Number *k* of negative samples (per positive sample)
  - $2 \leq k \leq 5$ for large text
  - $5 \leq k \leq 20$ for small text.

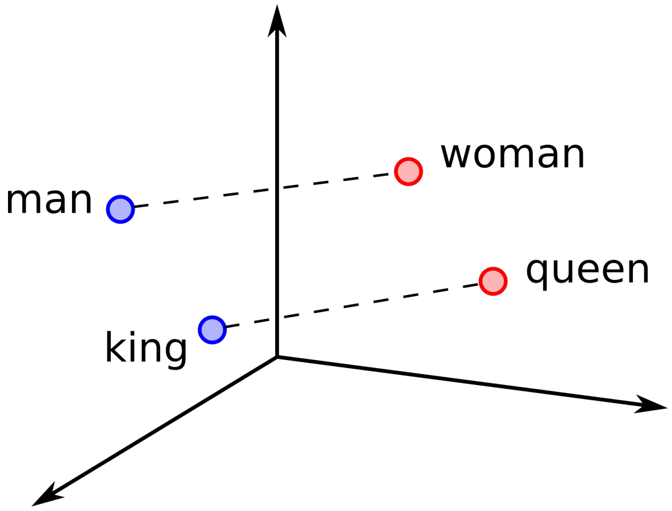
Outline

- Motivation
- Sparse Word Embeddings
  - Co-occurrence Vectors
  - Discussion & Limitations
- **Dense Word Embeddings**
  - Basic Idea
  - Word2Vec (CBOW & Skipgram)
  - Negative Sampling
  - **Basic Properties**
  - Practical Considerations & Limitations
- NLP Ethics

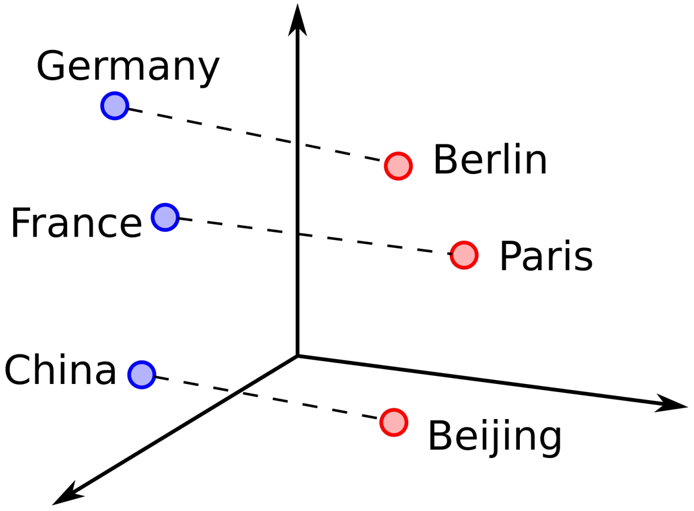
Word Embeddings — (Desired) Properties

- Vector differences yield semantic relationships ➜ linear substructures

$$v(king) - v(man) + v(woman) \approx v(queen)$$

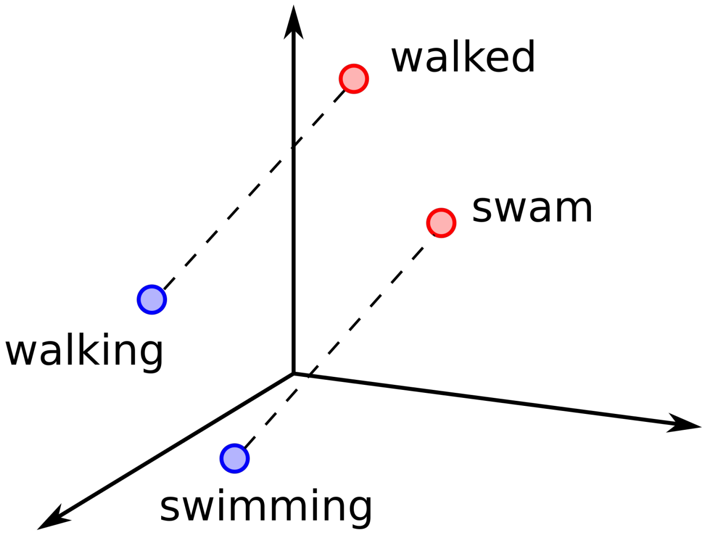


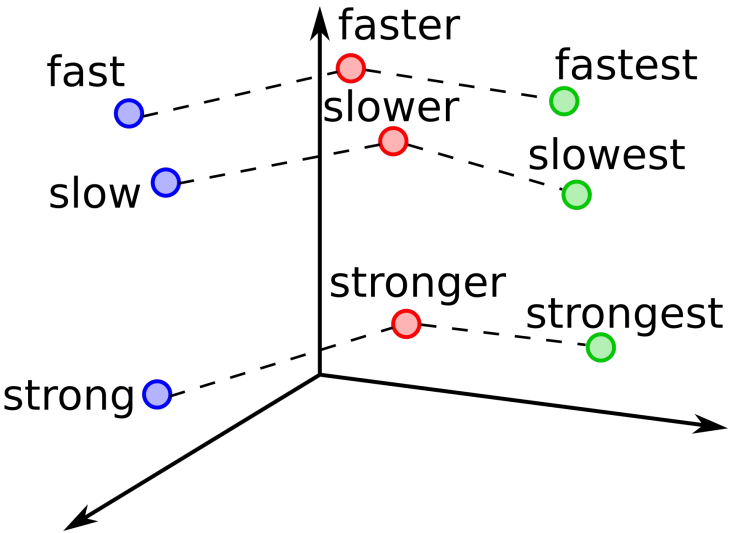
$$v(France) - v(Paris) + v(Berlin) \approx v(Germany)$$



Word Embeddings — (Desired) Properties

- Other meaningful linear substructures





Note: Getting these semantic relationships prohibit the use of stemming to lemmatization!

Outline

- Motivation
- Sparse Word Embeddings
  - Co-occurrence Vectors
  - Discussion & Limitations
- **Dense Word Embeddings**
  - Basic Idea
  - Word2Vec (CBOW & Skipgram)
  - Negative Sampling
  - Basic Properties
  - **Practical Considerations & Limitations**
- NLP Ethics

Word2Vec — Practical Considerations & Limitations

- Data preprocessing steps
  - Choice of tokenizer
  - Case-folding (yes/no)
  - Stemming/lemmatization (yes/no)
  - Stopword removal (yes/no)
  - Cross-sentence contexts (yes/no)

- Parameters
  - Window size m
  - Number of negative samples (e.g., $2mk$ for Skip-gram)

Word2Vec — Practical Considerations & Limitations

- Unable to represent phrases
  - *“New York”*, *“snow cat”*, *“ice cream”*, *“land mine”*, *“hot dog”*, *“disc drive”*, etc.

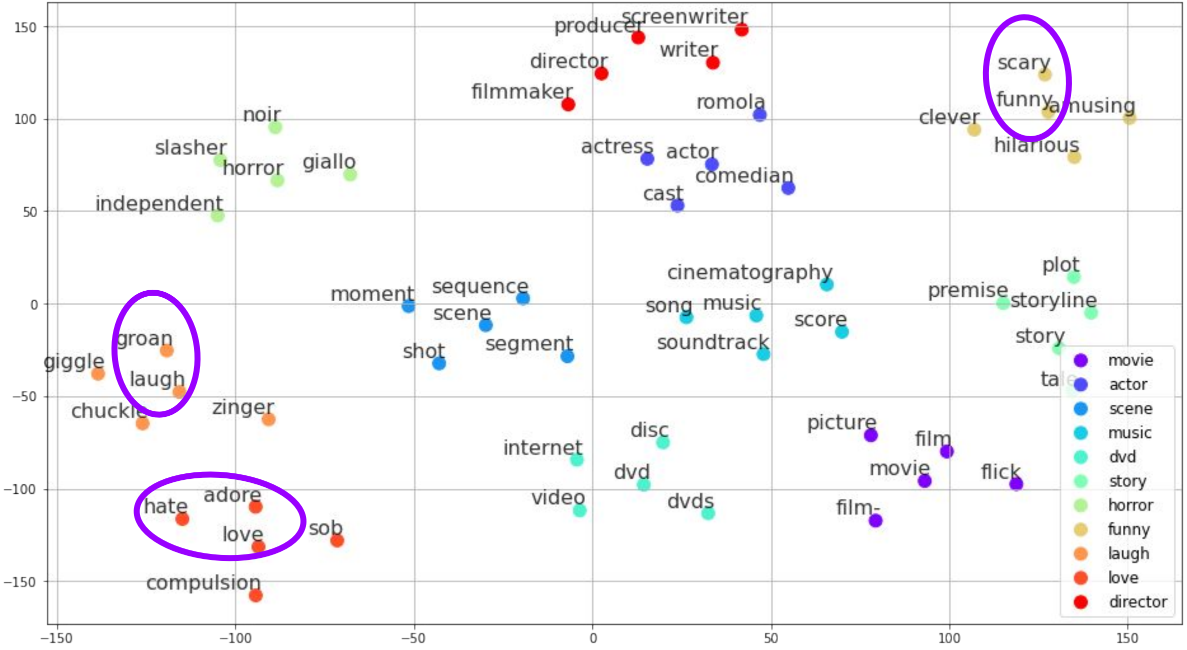
- Unable to handle polysemy and part of speech
  - Polysemy: multiple meanings for the same word
  - Part of speech: the same word used as noun, verb, or adjective

```
word2vec_wikipedia.wv.most_similar("light", topn=10)
```

```
[('lights', 0.5668156743049622),
 ('illumination', 0.5530915260314941),
 ('glow', 0.5415263175964355),
 ('sunlight', 0.5396571159362793),
 ('lamp', 0.5024341344833374),
 ('flame', 0.4877270166397095),
 ('lamps', 0.4784994721412658 7),
 ('dark', 0.4764614701271057),
 ('luminous', 0.4740492105484009),
 ('lighting', 0.4717761576175689 7)]
```

Word2Vec — Practical Considerations & Limitations

- Distributional representation does not capture all semantics
  - Common case: words with opposite polarity (sentiment) ➜ **Why?**



Word2Vec — Practical Considerations & Limitations

- Embeddings dependent on application / dataset

Dataset: Wikipedia

```
word2vec_wikipedia.wv.most_similar("house")
```

```
[('mansion', 0.7079392075538635),
 ('cottage', 0.6541333198547363),
 ('farmhouse', 0.6259987950325012),
 ('barn', 0.5747625827789307),
 ('bungalow', 0.5724436044692993),
 ('townhouse', 0.567018449306488),
 ('houses', 0.5506472587585449),
 ('parsonage', 0.5426527857780457),
 ('tavern', 0.5370140671730042),
 ('summerhouse', 0.5307810306549072)]
```

Dataset: Google News

```
word2vec_googlenews.most_similar("house")
```

```
[('houses', 0.7072390913963318),
 ('bungalow', 0.6878559589385986),
 ('apartment', 0.6628996729850769),
 ('bedroom', 0.6496936678886414),
 ('townhouse', 0.6384080052375793),
 ('residence', 0.6198420524597168),
 ('mansion', 0.6058192253112793),
 ('farmhouse', 0.5857570171356201),
 ('duplex', 0.5757936239242554),
 ('appartment', 0.569032549858933)]
```

Word2Vec in Practice (credits to Google https://code.google.com/archive/p/word2vec/)

- Architecture:
  - Skip-gram: slower, better for infrequent words
  - CBOW (fast)
- Training:
  - Hierarchical softmax: better for infrequent words
  - Negative sampling: better for frequent words, better with low dimensional vectors
- Sub-sampling of frequent words
  - can improve both accuracy and speed for large data sets (useful values are in range 1e-3 to 1e-5)
- Dimensionality of the word vectors
  - usually more is better, but not always
- Context (window) size
  - For skip-gram usually around 10, for CBOW around 5

Outline

- Motivation
- Sparse Word Embeddings
  - Co-occurrence Vectors
  - Discussion & Limitations
- Dense Word Embeddings
  - Basic Idea
  - Word2Vec (CBOW & Skipgram)
  - Negative Sampling
  - Basic Properties
  - Practical Considerations & Limitations
- **NLP Ethics**

NLP Ethics — Why this Topic?

- Real-world NLP applications have real-world impacts
  - Wide range of very common and popular services based on NLP we all use
    (online search / information retrieval, machine translation, chatbots, text summarization, etc.)
  - Many NLP applications making decisions affecting people's lives
    (e.g., what content we see — or don't see — on social media ➜ Think about it: What is needed for that?)

- Language does not exist in isolation
  - Natural language ➜ humans gave, give, and will give meaning to written and spoken word
  - Humans have different knowledge, beliefs, biases, preconceptions, etc.

"The common misconception is that language has to do with words and what they mean.
It doesn't. It has to do with people and what they mean."

(Clark & Schober, 1982)

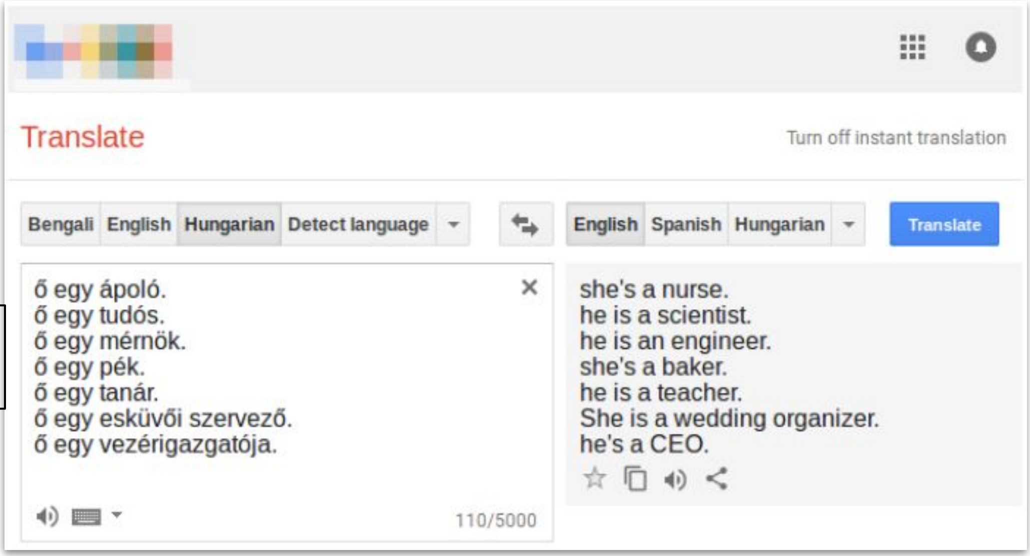
Dual Use and Adversarial NLP (a.k.a. Malicious NLP)

	(Arguably) Useful	(Potentially) Harmful
Authorship attribution (NLP/AI meets Linguistic Forensics)	Historical documents, ransom notes, plagiarism	Authors of political dissent
Text Generation	Fake content <u>detection</u>	Fake content <u>generation & misrepresentation</u>
User content analysis	Personalized content and recommendations	Privacy intrusion, “over-personalized” content (e.g., echo chambers / filter bubbles)
Censorship	Censorship evasion	More robust censorship

(Unintentionally) Harmful NLP

- Biased humans/society ➜ biased data ➜ biased model
  - NLP models are very likely to pick up such biases
  - Example: machine translation

Hungarian is an example of a gender-neutral language



Source: [Assessing Gender Bias in Machine Translation – A Case Study with Google Translate]

Biased NLP Technologies

- Biases identified in many NLP tasks/technologies
  - Bias in Word Embeddings (Bolukbasi et al. 2017; Caliskan et al. 2017; Garg et al. 2018)
  - Bias in Language identification (Blodgett & O'Connor. 2017; Jurgens et al. 2017)
  - Bias in Visual Semantic Role Labeling (Zhao et al. 2017)
  - Bias in Natural Language Inference (Rudinger et al. 2017)
  - Bias in Coreference Resolution (Rudinger et al. 2018; Zhao et al. 2018)
  - Bias in Automated Essay Scoring (Amorim et al. 2018)
  - Bias in Machine Translation (Prates et al. 2019)

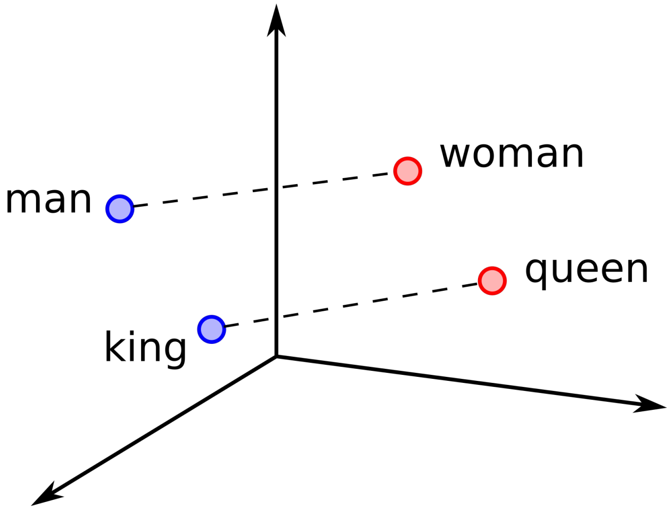
Bias in Word Embeddings

- Recall: Desired properties of word embeddings

$$v(king) - v(man) + v(woman) \approx v(queen)$$

$$v(France) - v(Paris) + v(Berlin) \approx v(Germany)$$

$$v(programmer) - v(man) + v(woman) \approx v(homemaker)$$



Source: [Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings]()

Bias in Sentiment Analysis

- Simple sentiment analysis
  - Sentiment lexicon + word embeddings (replace pos/neg words with their pretrained embedding)
  - Train a model to predict word sentiments (input: word vectors; target: sentiment label)

Looks alright

```
text_to_sentiment("this example is pretty cool")
```

3.889968926086298

```
text_to_sentiment("this example is okay")
```

2.7997773492425186

```
text_to_sentiment("meh, this example sucks")
```

-1.1774475917460698

Yeah, well…

```
text_to_sentiment("Let's go get Italian food")
```

2.0429166109408983

```
text_to_sentiment("Let's go get Chinese food")
```

1.4094033658140972

```
text_to_sentiment("Let's go get Mexican food")
```

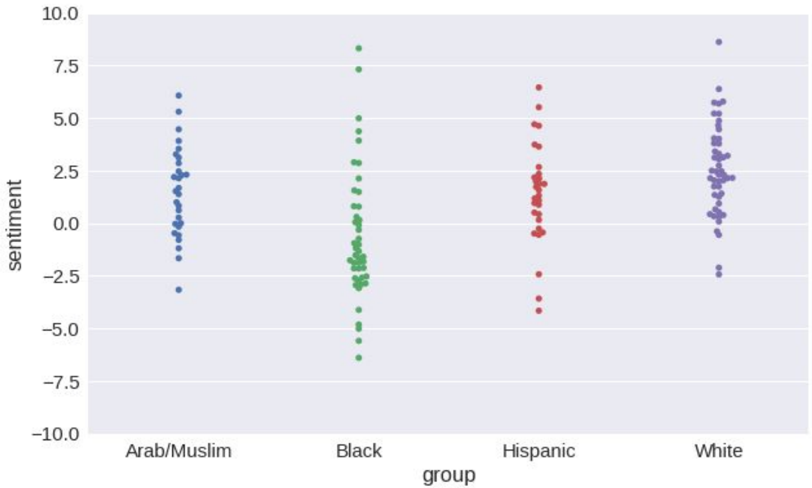
0.38801985560121732

Source: [How to make a racist AI without really trying]

Bias in Sentiment Analysis

- Here: Looking at common first names
  - more specifically: word vectors of names

	sentiment	group
mohammed	0.834974	Arab/Muslim
alya	3.916803	Arab/Muslim
terryl	-2.858010	Black
josé	0.432956	Hispanic
luciana	1.086073	Hispanic
hank	0.391858	White
megan	2.158679	White



Source: [How to make a racist AI without really trying]

🏃‍♀️🏃🏃‍♂️ De-biasing Embeddings

Towards Debiasing — Measuring Bias

- How to check if your word embeddings contain biases?
  - Example: gender biases
  - Approach: find nearest occupations to *"he"* and *"she"*
    (e.g. the word vector for "homemaker" is very close to the word vector of "she")

common female occupations

vs.

common male occupations

Extreme *she*	**Extreme *he***
1. homemaker	1. maestro
2. nurse	2. skipper
3. receptionist	3. protege
4. librarian	4. philosopher
5. socialite	5. captain
6. hairdresser	6. architect
7. nanny	7. financier
8. bookkeeper	8. warrior
9. stylist	9. broadcaster
10. housekeeper	10. magician

Source: [Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings]

Towards Debiasing — Measuring Bias

- Identifying biases using analogies
  - Approach: Find pairs of words (x, w) that minimize:

$$cosine(v(she) - v(he), v(x) - v(y)) \quad \text{if } \overbrace{\|v(x) - v(y)\| \leq \delta}^{\text{to ensure that } x \text{ and } y \text{ are semantically similar "enough"}}$$

Gender stereotype *she*-*he* analogies

sewing-carpentry	registered nurse-physician	housewife-shopkeeper
nurse-surgeon	interior designer-architect	softball-baseball
blond-burly	feminism-conservatism	cosmetics-pharmaceuticals
giggle-chuckle	vocalist-guitarist	petite-lanky
sassy-snappy	diva-superstar	charming-affable
volleyball-football	cupcakes-pizzas	lovely-brilliant

Gender appropriate *she*-*he* analogies

queen-king	sister-brother	mother-father
waitress-waiter	ovarian cancer-prostate cancer	convent-monastery

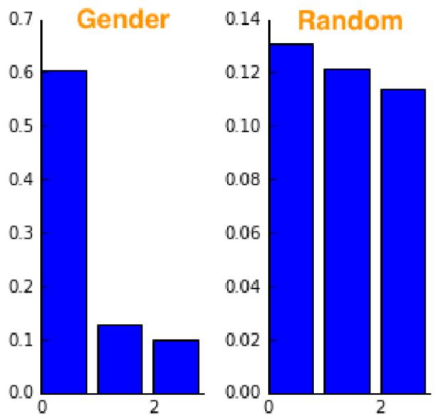
Source: Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings

Towards Debiasing — Identify Gender Subspace

- Which “direction” of the 300-dim embedding space encodes gender?
  - Approach: Pick top pairs of gender words ➜ interpret difference as direction(s) of gender
  - Problem: Language is “messy” ➜ difference point exactly in the same direction(s)

$$\begin{aligned}
\overrightarrow{\text{she}} &- \overrightarrow{\text{he}} \\
\overrightarrow{\text{her}} &- \overrightarrow{\text{his}} \\
\overrightarrow{\text{woman}} &- \overrightarrow{\text{man}} \\
\overrightarrow{\text{Mary}} &- \overrightarrow{\text{John}} \\
\overrightarrow{\text{herself}} &- \overrightarrow{\text{himself}} \\
\overrightarrow{\text{daughter}} &- \overrightarrow{\text{son}} \\
\overrightarrow{\text{mother}} &- \overrightarrow{\text{father}} \\
\overrightarrow{\text{gal}} &- \overrightarrow{\text{guy}} \\
\overrightarrow{\text{girl}} &- \overrightarrow{\text{boy}} \\
\overrightarrow{\text{female}} &- \overrightarrow{\text{male}}
\end{aligned}$$

Principal Component Analysis (PCA)



Use top PCs (principal components, vectors in the embedding space) as gender subspace

Source: [Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings]

Towards Debiasing — Identify Gender-Neutral Words

- Split vocabulary into gender-neutral words (*N*) and gender-specific words (*S*)
    - Manually identify a set of gender-specific words ➜ training data
      (we are interesting in N, but there are much more gender-neutral words, so that's easier)
    - Train a binary classifier to predict if a word (vector) is gender-neutral or gender-specific
    - Generate *N* and *S* by predicting the class for each word (vector)

Source: [Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings]()

Towards Debiasing — Embedding Transformation

Frobenius norm:

$$\|A\|_F^2 = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}|a_{ij}|^2}$$

- Goal: Transform embeddings to remove gender bias
  - Idea: Find a transformation matrix T the transforms the embedding matrix W
  - Approach: Find T by minimizing

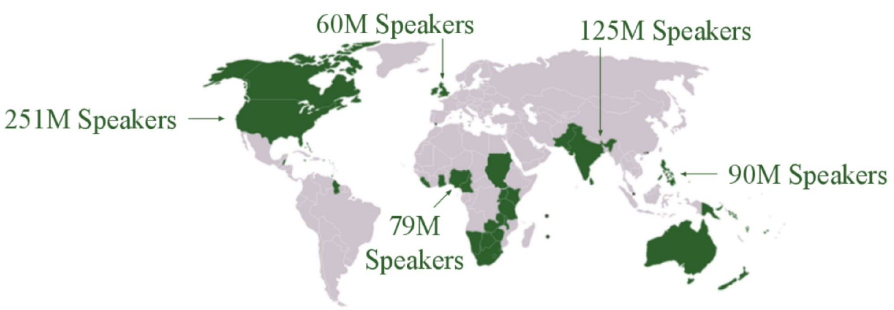
$$\min_T \underbrace{\underbrace{\|(TW)^\top(TW)}_{\text{inner products of embeddings **after** transformation}} - \underbrace{(W^\top W)}_{\text{inner products of embeddings **before** transformation}}\|_F^2}_{\text{Keep difference (here: Frobenius norm) small — preserve the original embeddings as much as possible!}} + \underbrace{\lambda\|\underbrace{(TN)^\top}_{\text{transformed gender-neutral word vectors}}\underbrace{(TB)}_{\text{transformed gender subspace}}\|_F^2}_{\text{Minimal if gender subspace removed from vectors of gender-neutral words}}$$

Source: [Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings]()

Locally Relevant: Code Switching

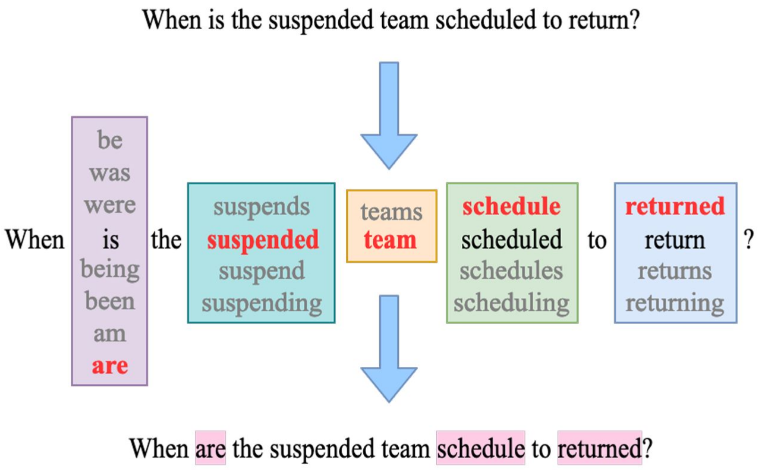
- Code Switching: changing languages in the middle of discourse

- Related: Language Change: [Pidgins, Creoles and Patois]



Augment the Training Data: Morpheus

- Generate morphological variations in data to desensitize the model to morphological variations that L2 speakers might use.



Algorithm 1 MORPHEUS

Require: Original instance x, Label y, Model f
Ensure: Adversarial example x'
  $T \leftarrow \textsc{Tokenize}(x)$
  for all $i = 1, \ldots, |T|$ **do**
    if $\text{POS}(T_i) \in \{\text{NOUN}, \text{VERB}, \text{ADJ}\}$ **then**
      $I \leftarrow \textsc{GetInflections}(T_i)$
      $T_i \leftarrow \textsc{MaxInflected}(I, T, y, f)$
    end if
  end for
  $x' \leftarrow \textsc{Detokenize}(T)$
  return x'

Source: [It's Morphin' Time! Combating Linguistic Discrimination with Inflectional Perturbations]

Summary

- (Dense) word embeddings
  - Core component of many to most NLP applications
    (particularly applications based on neural network solutions)
  - Dense vectors automatically learned from data
  - Support a intuitive notion of similarity between words
    (words with similar meanings ➜ words have similar word vectors)

- NLP Ethics
  - Like with all technologies: **use** and **misuse** (accidentally or maliciously)
  - Focus here: biases in word embeddings (due to biases in the data, due to biases in society)

Outlook for Next Week: Sequences

From Karolina Grabowska via *Pexels*

Pre-Lecture Activity for Next Week

- Assigned Task (due before Mar 8)
  - Post a 1-2 paragraph answer to the following question in the [Pre-Lecture] discussion
  - Watch the panel discussion of the recent **Generative AI, Free Speech, & Public Discourse**
    https://www.youtube.com/live/BBhewsinQwU?si=ODkIpYjqOCLZh8xD&t=8659

Relate an opinion by one of the panelists to the lecture materials presented today. Why did you pick this opinion to highlight and what is your own opinion on it?

Side notes:
- We will talk about this in the next lecture
- You can just copy-&-paste others' answers or use AI Tools, but please consider your original stance and opinion too.