CSCE 638 Natural Language Processing Foundation and Techniques

Lecture 3: Word Representations

Kuan-Hao Huang

Spring 2025

(Some slides adapted from Chris Manning, Dan Jurafsky, Richard Socher, Karthik Narasimhan, and Danqi Chen)

Course Materials

- Available on the course website before the lecture
- Available on Canvas after the lecture

Assignment 0

- https://khhuang.me/CSCE638-S25/assignments/assignment0_0122.pdf
- Due: 1/29/2025 11:59pm
- Summit a .zip file to Canvas
    - submission.pdf for the writing section
    - submission.py and submission.ipynb for the coding section
- For questions
    - Discuss on Canvas
    - Send an email to csce638-ta-25s@list.tamu.edu

Course Staff

Instructor

Kuan-Hao Huang

- Email: khhuang@tamu.edu
- Office Hour: Wed. 2pm – 3pm
- Office: PETR 219

TA

Rahul Baid

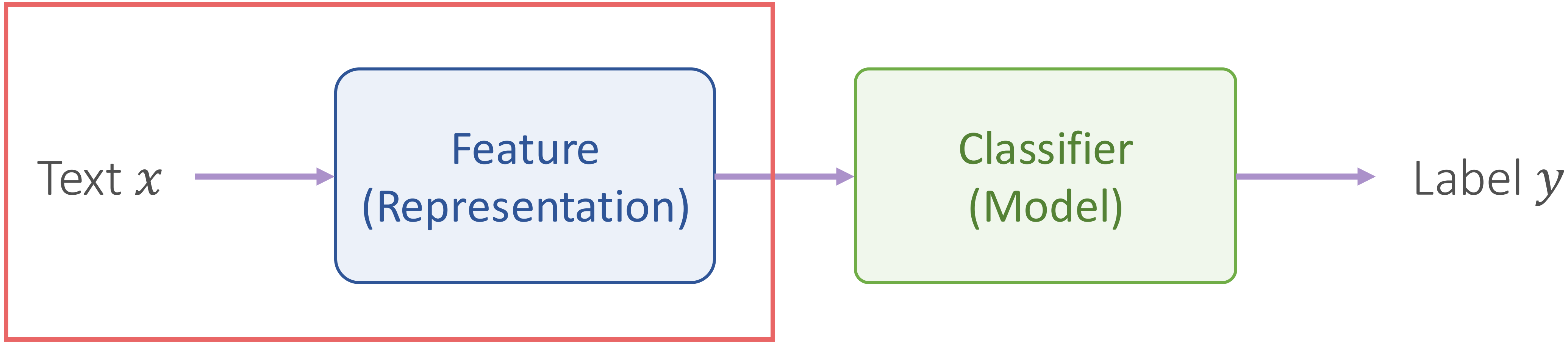
- Email: rahulbaid@tamu.edu
- Office Hour: Wed. 12pm – 1pm
- Office: PETR 359

For questions, send emails to csce638-ta-25s@lists.tamu.edu

Lecture Plan

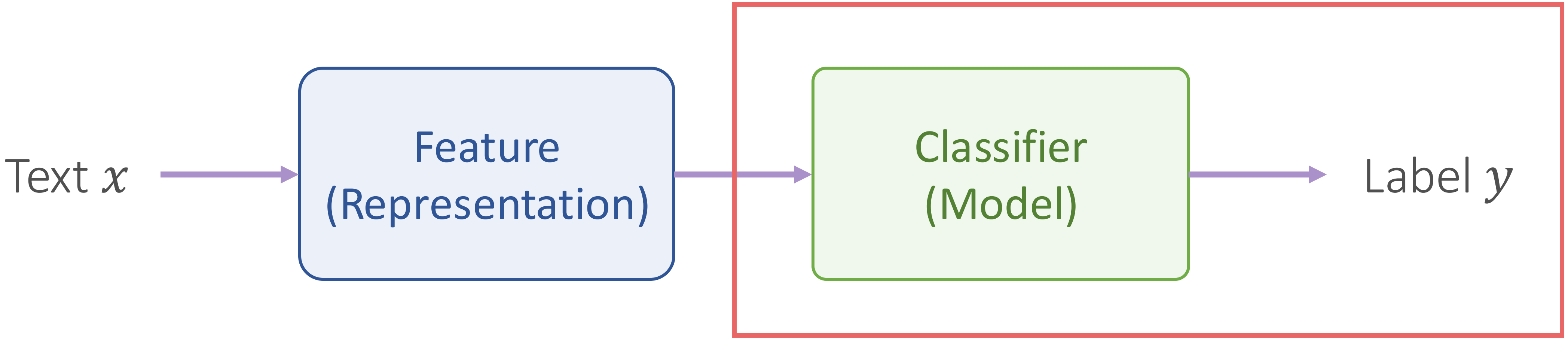
- Count-Based Word Vectors
- Prediction-Based Word Vectors
- Evaluation for Word Vectors

Recap: A General Framework for Text Classification



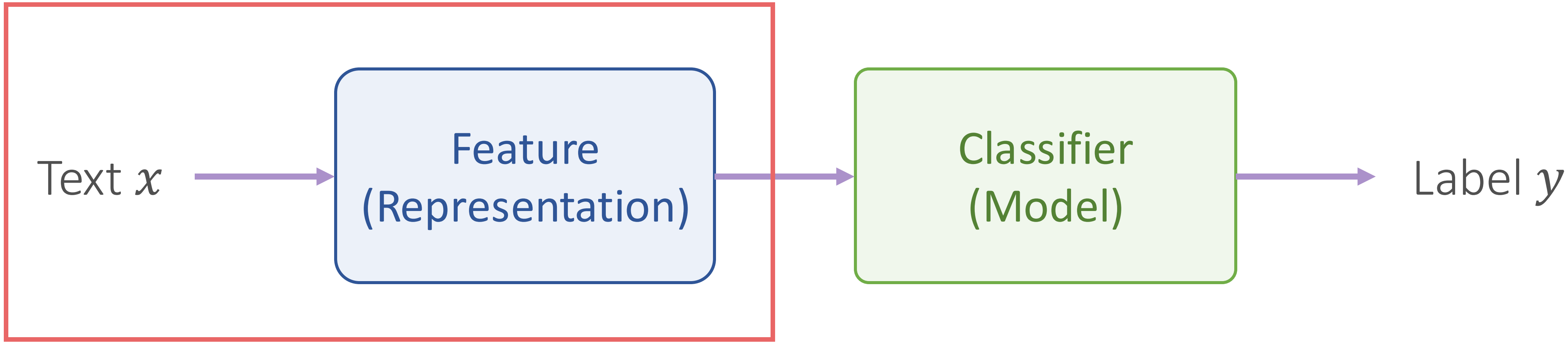
- Teach the model how to understand example x

Recap: A General Framework for Text Classification



- Teach the model how to make prediction y

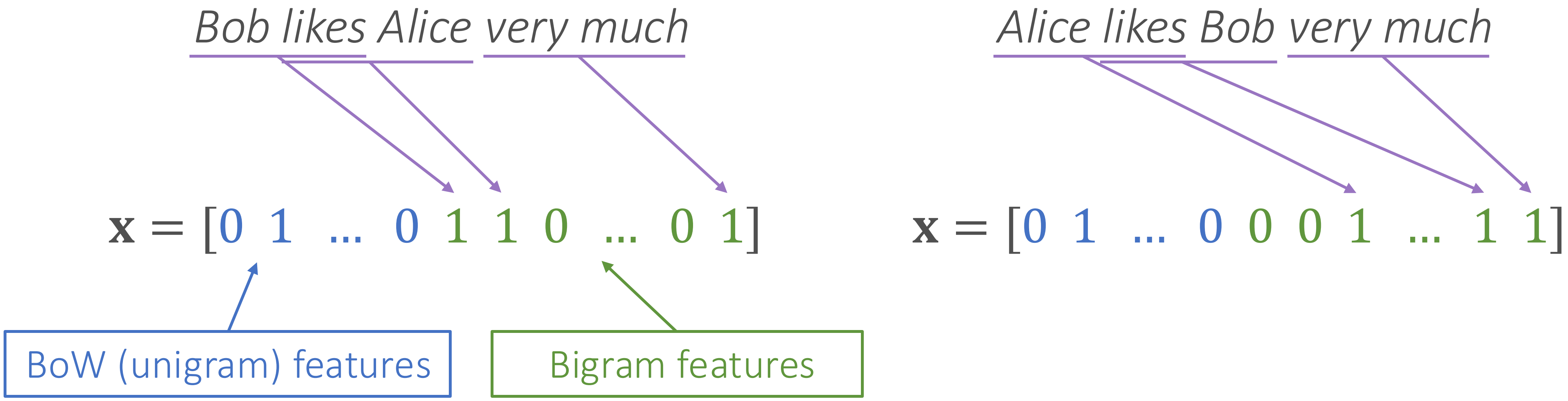
Recap: Bag-of-Words and N-Grams



- Teach the model how to understand example x
- Convert the text to a mathematical form
  - The mathematical form captures essential characteristics of the text
- Bag-of-words and n-grams

We will discuss “learnable” features today!

Bag-of-Words and N-Gram Features



Bob likes Alice very much

$$\mathbf{x} = [0\ \ 1\ \ \dots\ \ 0\ \ 1\ \ 1\ \ 0\ \ \dots\ \ 0\ \ 1]$$

BoW (unigram) features

Bigram features

Alice likes Bob very much

$$\mathbf{x} = [0\ \ 1\ \ \dots\ \ 0\ \ 0\ \ 0\ \ 1\ \ \dots\ \ 1\ \ 1]$$

We can consider trigrams, 4-grams, …

Encode a text to ***one vector***

Words as Vectors

$$\begin{array}{rc} & \begin{array}{ccccc} \mathit{Bob} & \mathit{likes} & \mathit{Alice} & \mathit{very} & \mathit{much} \end{array} \\ W = & \begin{bmatrix} | & | & | & | & | \\ w_{bob} & w_{likes} & w_{Alice} & w_{very} & w_{much} \\ | & | & | & | & | \end{bmatrix} \end{array}$$

Use ***one vector*** to represent ***each word***

Text = A list of vectors

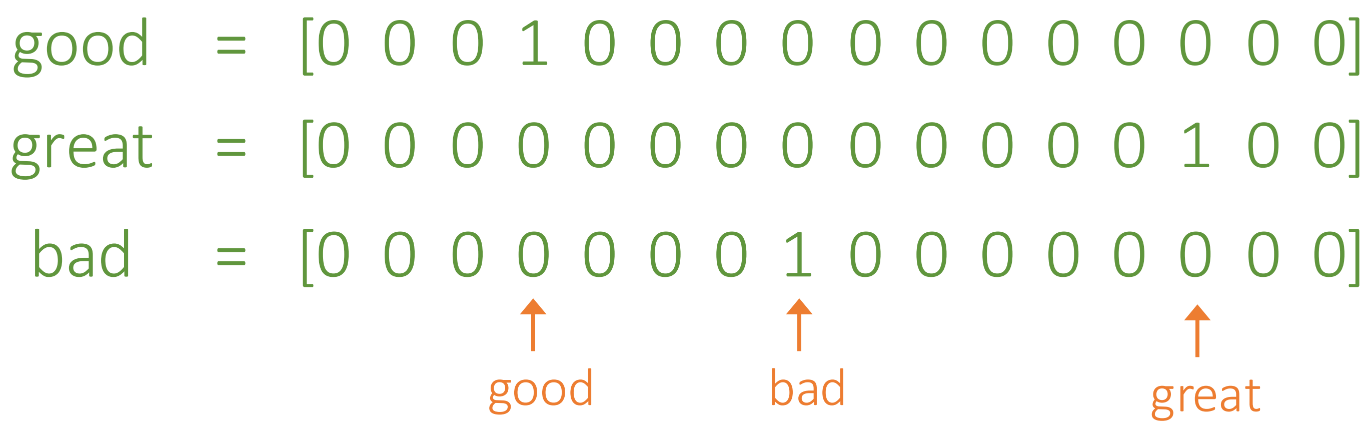
Advantages?

How to Represent Words?

A simple solution: discrete symbols

One 1, the rest 0s

Words can be represented by one-hot vectors:



good = [0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0]

great = [0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0]

bad = [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0]

Vector dimension = number of words in vocabulary (e.g., 500,000+)

Any disadvantages?

Problem with Words as Discrete Symbols

Example: in web search, if a user searches for “good restaurant”, we would like to match documents containing “great restaurant”

But

good = [0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0]

great = [0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0]

These two vectors are orthogonal

There is no way to encode similarity of words in these vectors!

Any solutions?

Previous Solution: Synonyms, Antonyms, and Hypernyms

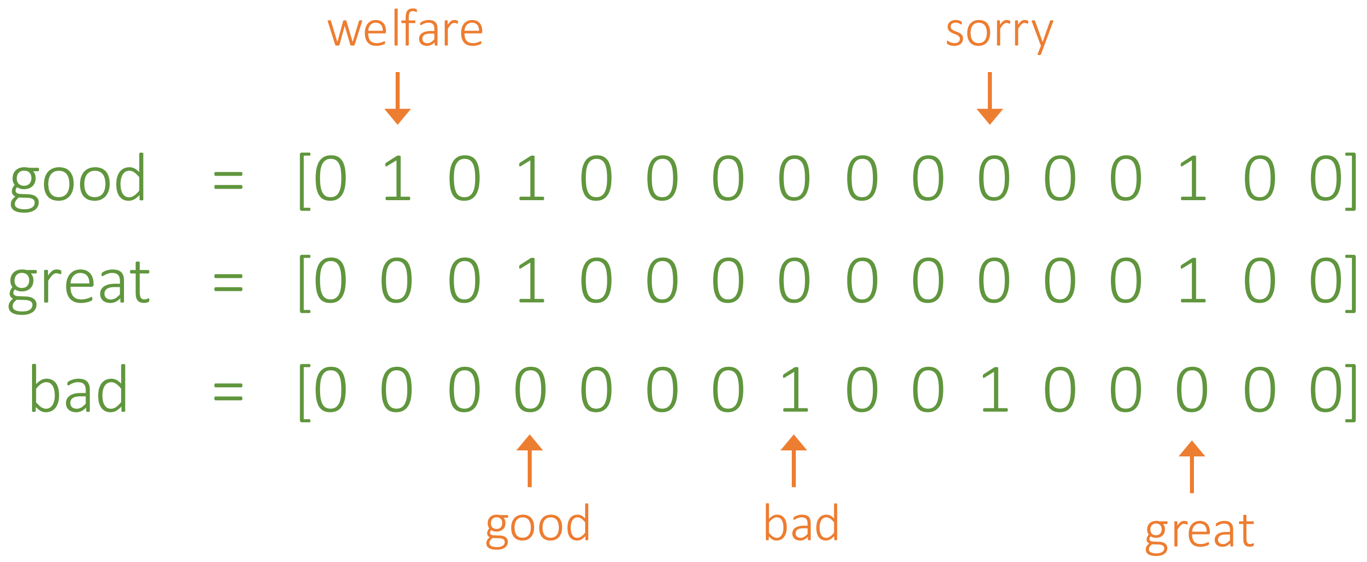
Consider external resources like WordNet, a thesaurus containing lists of Synonyms, antonyms, and hypernyms

```
from nltk.corpus import wordnet as wn
poses = { 'n’ : 'noun', 'v’ : 'verb', 's’ : 'adj (s)', 'a’ : 'adj', 'r’ : 'adv'}
for synset in wn.synsets(“bad”):
   print("{}: {}".format(poses[synset.pos()],
              ", ".join([l.name() for l in synset.lemmas()])))
```

```
noun: bad, badness
adj: bad
adj (s): bad, big
adj (s): bad, tough
adj (s): bad, spoiled, spoilt
adj: regretful, sorry, bad
adj (s): bad, uncollectible
…
adj (s): bad, risky, high-risk, speculative
adj (s): bad, unfit, unsound
adj (s): bad, forged
adj (s): bad, defective
adv: badly, bad
```

Previous Solution: Synonyms, Antonyms, and Hypernyms

Consider external resources like WordNet, a thesaurus containing lists of Synonyms, antonyms, and hypernyms



welfare

sorry

good = [0 1 0 1 0 0 0 0 0 0 0 0 0 1 0 0]

great = [0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0]

bad = [0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0]

good

bad

great

$$\cos(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{u}\|\|\mathbf{v}\|}$$

Similarity(good, great) > Similarity(good, bad)

Any disadvantages?

Problems with Resources Like WordNet

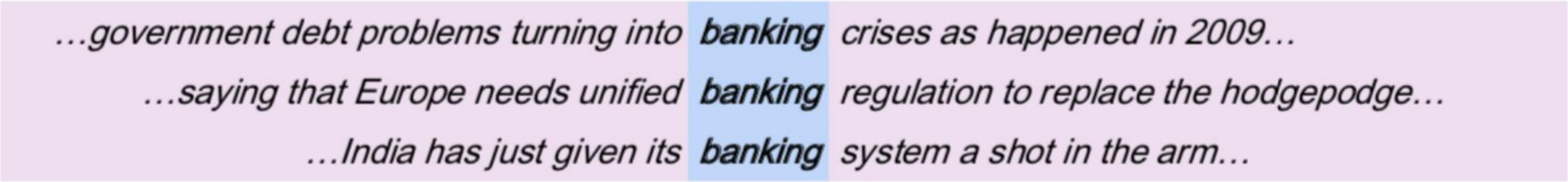
- Subjective
- A useful resource but missing nuance
  - e.g., “sorry” is listed as a synonym for “bad”
  - This is only correct in some contexts
- Requires human labor to create and adapt

Representing Words by Their Contexts

Distributional hypothesis: words that occur in similar contexts tend to have similar meanings

J.R.Firth 1957

- “You shall know a word by the company it keeps”
- One of the most successful ideas of modern statistical NLP!



…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

Distributional Hypothesis

C1: A bottle of ___ is on the table.

C2: Everybody likes ___.

C3: Don’t have ___ before you drive.

C4: I bought ___ yesterday.

	C1	C2	C3	C4
juice	1	1	0	1
loud	0	0	0	0
motor-oil	1	0	0	1
chips	0	1	0	1
choices	0	1	0	0
wine	1	1	1	1

Words that occur in similar contexts tend to have similar meanings

Word Vectors from Word-Word Co-Occurrence Matrix

- Main idea: Similar contexts → Similar word co-occurrence
- Collect a bunch of texts and compute co-occurrence matrix
- Words can be represented by row vectors

$$\cos(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{u}\| \|\mathbf{v}\|}$$

Word Vector

High cosine similarity!

	shark	computer	data	eat	result	sugar
apple	0	0	0	8	0	2
bread	0	0	0	9	0	1
digital	0	6	5	0	2	0
information	0	4	10	0	2	0

Low cosine similarity!

Most entries are 0s → sparse vectors

Issues with Word-Word Co-Occurrence Matrix

- Using raw frequency counts is not always very good (why?)
  - Some frequent words (e.g., the, it, or they) can have large counts

	the	computer	data	eat	result	sugar	the	it
apple	0	0	0	8	0	2	104	67
bread	0	0	0	9	0	1	95	76
digital	0	6	5	0	2	0	101	65

Similarity(apple, bread) ≈ 0.994710

Similarity(apple, digital) ≈ 0.995545

Similarity is dominated by frequent words

Solution: use a ***weighted function*** instead of raw counts

Pointwise Mutual Information

Pointwise Mutual Information (PMI)

Do events x and y co-occur more or less than if they were independent?

$$\mathrm{PMI}(x, y) = \log_2 \frac{P(x, y)}{P(x)P(y)}$$

- PMI = 0 → x and y occur independently → co-occurrence is as expected
- PMI > 0 → x and y co-occur more often than expected
- PMI < 0 → x and y co-occur less often than expected

Co-Occurrence Matrix with Positive PMI

Positive Pointwise Mutual Information (PPMI)

$$\mathrm{PPMI}(x, y) = \max\left(\log_2 \frac{P(x, y)}{P(x)P(y)}, 0\right)$$

	the	computer	data	eat	result	sugar	the	it
apple	0	0	0	1.80	0	0.35	0.08	0
bread	0	0	0	1.54	0	0.29	0	0.14
digital	0	1.47	1.22	0	0.61	0	0.10	0.06

Similarity(apple, bread) ≈ 0.995069

Similarity(apple, digital) ≈ 0.010795

Sparse Vectors vs. Dense Vectors

- The vectors in the word-word occurrence matrix are
  - **Long**: vocabulary size
  - **Sparse**: most are 0’s
- Can we have short short (50-300 dimensional) and dense (real-valued) vectors?
  - Short vectors are easier to use as features in ML systems
  - Dense vectors may generalize better than explicit counts
  - Sparse vectors can’t capture high-order co-occurrence
    - w_1 co-occurs with “car”, w_2 co-occurs with “automobile”
    - They should be similar, but they aren’t, because “car” and “automobile” are distinct dimensions
  - In practice, they work better!

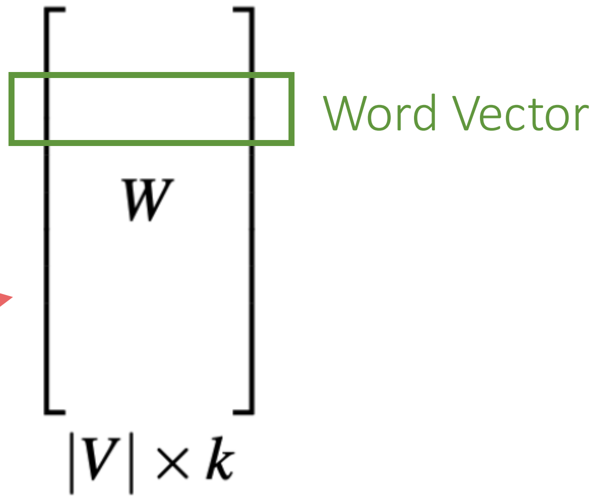
How to Get Dense Vectors?

- Singular value decomposition (SVD) of PPMI weighted co-occurrence matrix

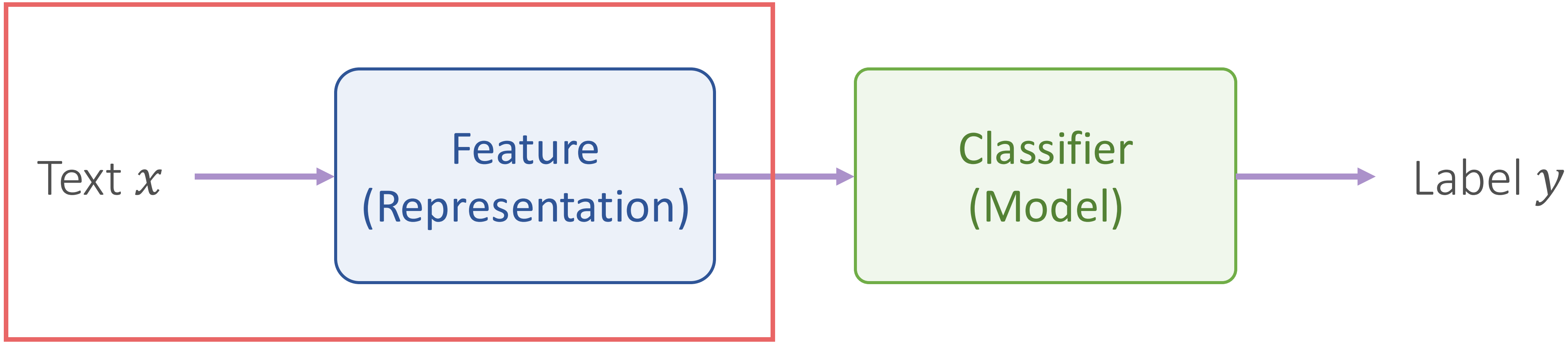
$$\underset{|V|\times|V|}{\begin{bmatrix} \quad \\ X \\ \quad \end{bmatrix}} = \underset{|V|\times|V|}{\begin{bmatrix} \quad \\ W \\ \quad \end{bmatrix}} \underset{|V|\times|V|}{\begin{bmatrix}\sigma_1 & 0 & 0 & \dots & 0\\ 0 & \sigma_2 & 0 & \dots & 0\\ 0 & 0 & \sigma_3 & \dots & 0\\ \vdots & \vdots & \vdots & \ddots & \vdots\\ 0 & 0 & 0 & \dots & \sigma_V\end{bmatrix}} \underset{|V|\times|V|}{\begin{bmatrix} \quad \\ C \\ \quad \end{bmatrix}}$$

Only keep the top k singular values

$$\underset{|V|\times|V|}{\begin{bmatrix} \quad \\ X \\ \quad \end{bmatrix}} = \underset{|V|\times k}{\begin{bmatrix} \quad \\ W \\ \quad \end{bmatrix}} \underset{k\times k}{\begin{bmatrix}\sigma_1 & 0 & 0 & \dots & 0\\ 0 & \sigma_2 & 0 & \dots & 0\\ 0 & 0 & \sigma_3 & \dots & 0\\ \vdots & \vdots & \vdots & \ddots & \vdots\\ 0 & 0 & 0 & \dots & \sigma_k\end{bmatrix}} \underset{k\times|V|}{\begin{bmatrix} C \end{bmatrix}}$$

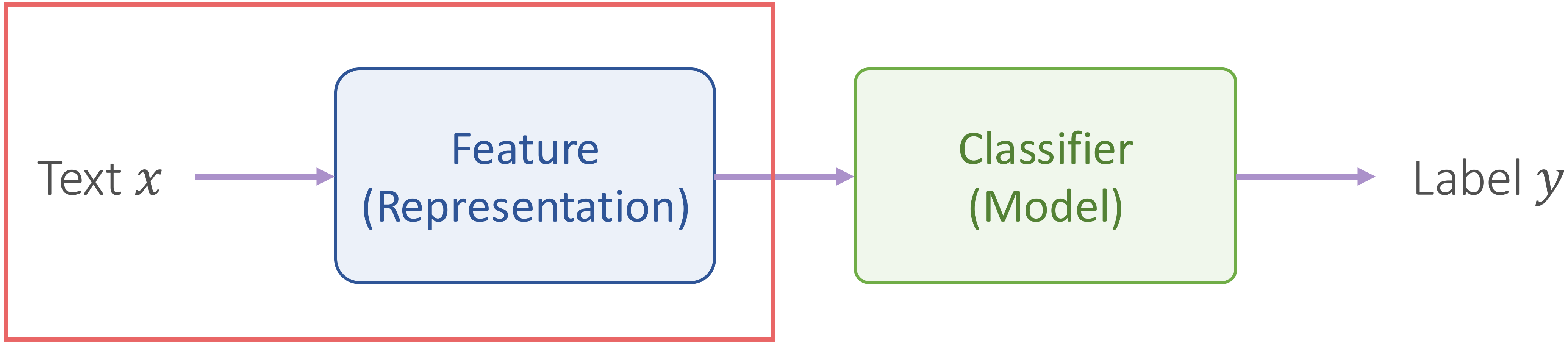


Count-Based Word Vectors



- Use one vector to represent each word
- Get word vectors by singular value decomposition (SVD) of PPMI weighted co-occurrence matrix

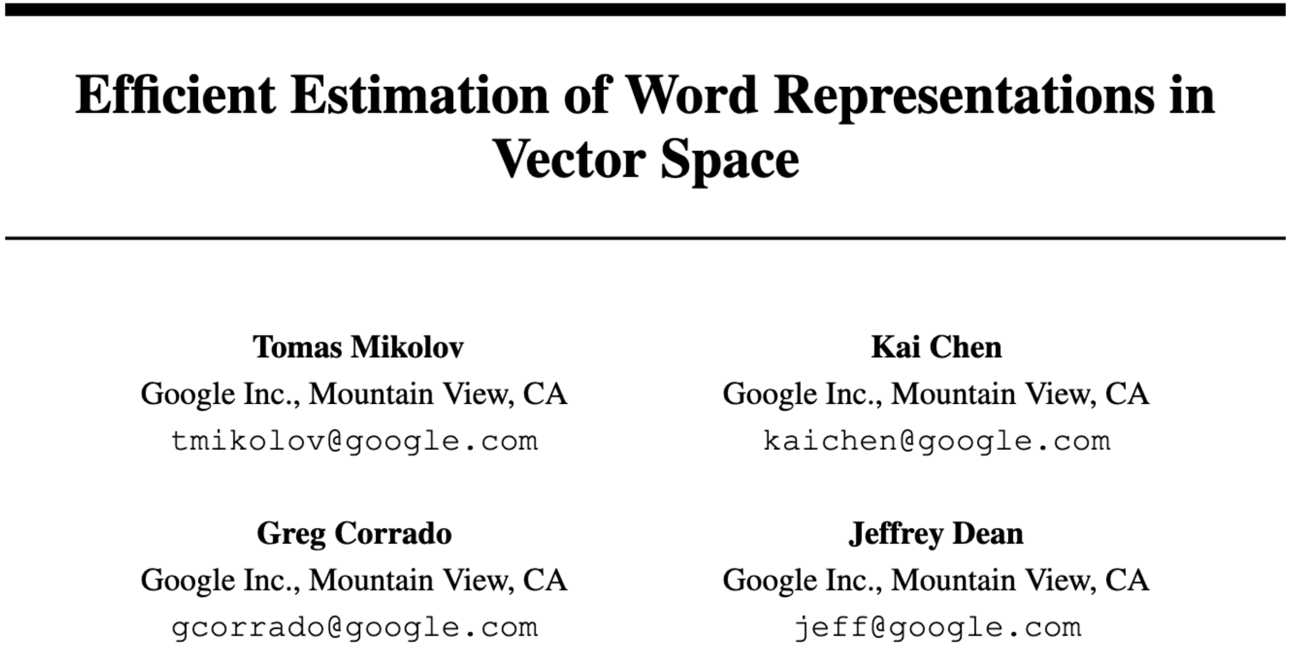
Prediction-Based Word Vectors



- Can we learn word vectors directly from text?

Word2Vec

- Efficient Estimation of Word Representations in Vector Space, 2013
  - 40000+ citations



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

Word Embeddings as Learning Problem

- Learning vectors (also called embeddings) from text for representing words
- Input:
  - A large text corpus
    - Wikipedia + Gigaword 5: 6B tokens
    - Twitter: 27B tokens
    - Common Crawl: 840B tokens
  - Vocabulary $\mathcal{V}$
  - Vector dimension d (e.g., 300)
- Output:
  - Mapping function $f\colon \mathcal{V} \rightarrow \mathbb{R}^d$

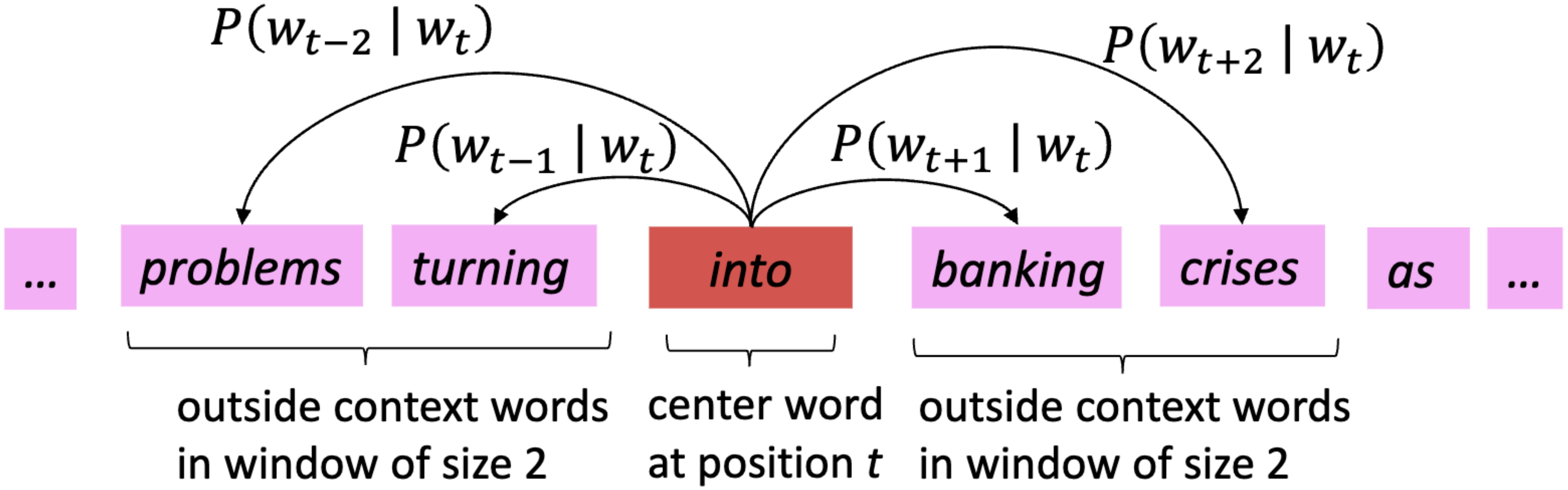
$$v_{apple} = \begin{pmatrix} -0.224 \\ 0.479 \\ 0.871 \\ -0.231 \\ 0.101 \end{pmatrix}$$

$$v_{digital} = \begin{pmatrix} 0.257 \\ 0.587 \\ -0.972 \\ -0.456 \\ -0.002 \end{pmatrix}$$

Word2Vec: Overview

- **Main idea:** we want to use words to predict their context words
- Context: a fixed window of size m

Use center word w_t to predict context words w_{t-m} to w_{t+m}



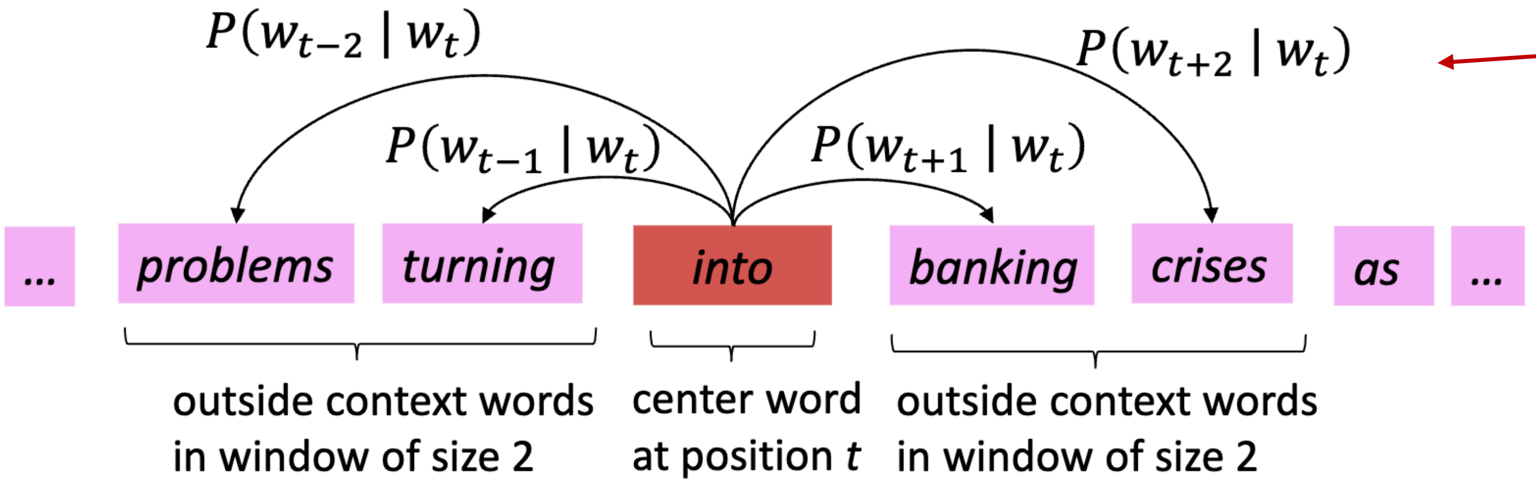
Words that occur in similar contexts tend to have similar meanings

Word2Vec: Overview

- **Main idea:** we want to use words to predict their context words
- Context: a fixed window of size m

Classification Problem

Use center word w_t to predict context words w_{t-m} to w_{t+m}



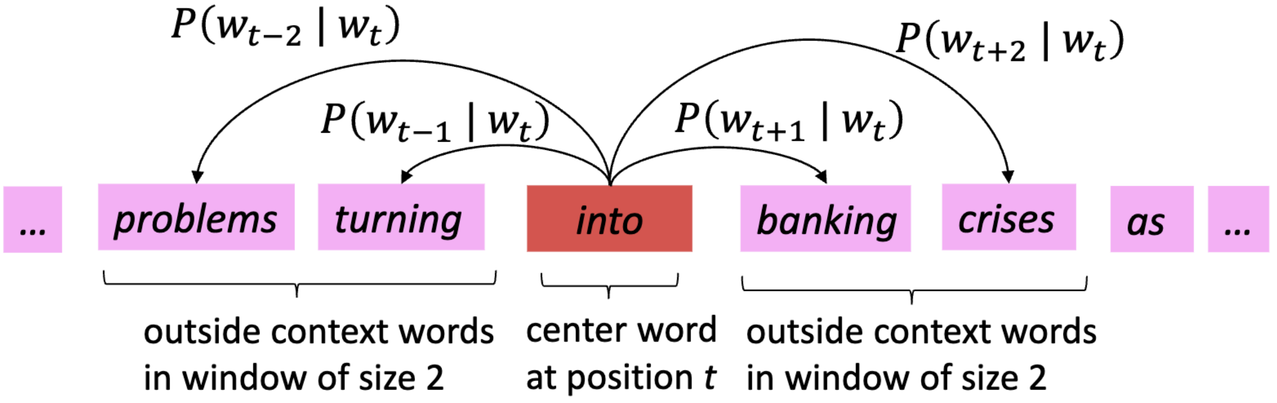
$P(b|a)$ = given the center word is a, what is the probability that b is a context word?

$P(\cdot\,|a)$ is a probability distribution defined over $\mathcal{V}$:

$$\sum_{w \in \mathcal{V}} P(w|a) = 1$$

We will define the distribution soon!

Word2Vec: Overview



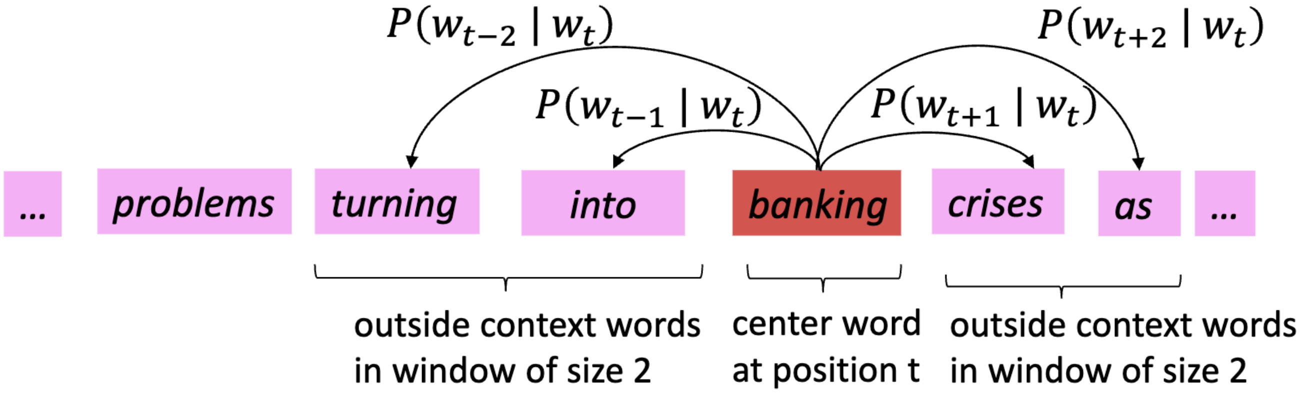
Collect into training data

(into, problems)

(into, turning)

(into, banking)

(into, crises)



Collect into training data

(banking, turning)

(banking, into)

(banking, crises)

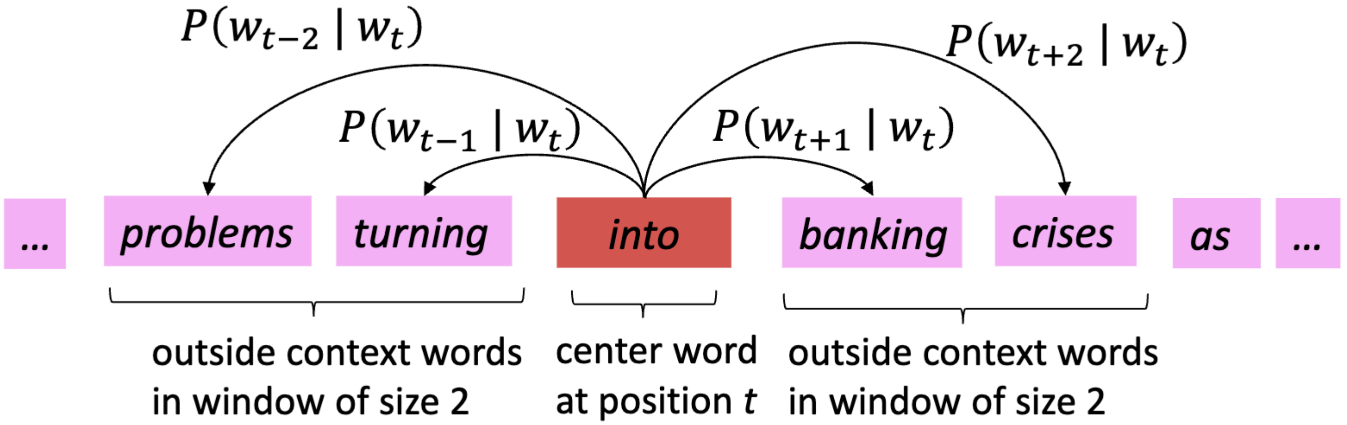
(banking, as)

Maximize the likelihood

$$P(\text{problems}\mid\text{into})\times P(\text{turning}\mid\text{into})\times P(\text{banking}\mid\text{into})\times P(\text{crises}\mid\text{into})$$

$$\times P(\text{turning}\mid\text{banking})\times P(\text{into}\mid\text{banking})\times P(\text{crises}\mid\text{banking})\times P(\text{as}\mid\text{banking})$$

Word2Vec: Likelihood



For each position $t = 1, \dots, T$, predict context words within a window of fixed size m, given center word w_t

θ all parameters to be optimized

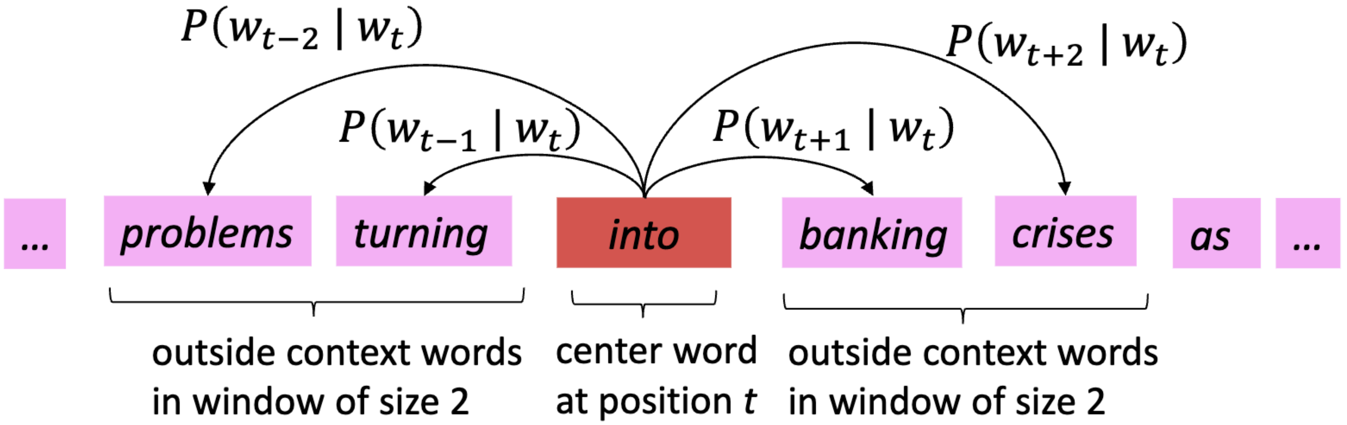
$$\text{Likelihood} = \mathcal{L}(\theta) = \prod_{t=1}^{T} \prod_{-m\leq j\leq m, j\neq 0} P\left(w_{t+j} \middle| w_t \,; \theta\right)$$

Probability over all vocabulary V

For each position $t = 1, \dots, T$

Likelihood for all context words given center word w_t

Word2Vec: Objective Function



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

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

We minimize the objective function (also called cost or loss function)

How to Define Probability?

Question: how to calculate $P\left(w_{t+j} \middle| w_t ; \theta\right)$?

Answer: we have two sets of vectors for each word in the vocabulary

$\mathbf{u}_w \in \mathbb{R}^d$: word vector when w is a center word

$\mathbf{v}_w \in \mathbb{R}^d$: word vector when w is a context word

We consider Inner product $\mathbf{u}_{w_t} \cdot \mathbf{v}_{w_{t+j}}$ as the score to measure how likely the context word w_{t+j} appears with the center word w_t, the larger the more likely!

$$P\left(w_{t+j} \middle| w_t ; \theta\right) = \frac{\exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_{w_{t+j}})}{\sum_{k \in V} \exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_k)} \qquad \theta = \left\{\{\mathbf{u}_k\}, \{\boldsymbol{v}_k\}\right\} \text{ all parameters}$$

How to Define Probability?

We have two sets of vectors for each word in the vocabulary

$\mathbf{u}_w \in \mathbb{R}^d$: word vector when w is a center word

$\mathbf{v}_w \in \mathbb{R}^d$: word vector when w is a context word

$$P\left(w_{t+j} \middle| \, w_t \, ; \theta\right) = \frac{\exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_{w_{t+j}})}{\sum_{k \in V} \exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_k)}$$

Normalize over entire vocabulary to give probability distribution

The score to indicate how likely the context word w_{t+j} appears with the center word w_t

Softmax function: mapping arbitrary values to a probability distribution

$$\mathrm{softmax}(t) = \frac{e^t}{\sum_c e^c}$$

Why Two Sets of Vectors?

We have two sets of vectors for each word in the vocabulary

$\mathbf{u}_w \in \mathbb{R}^d$: word vector when w is a center word

$\mathbf{v}_w \in \mathbb{R}^d$: word vector when w is a context word

$$P\left(w_{t+j} \middle| \, w_t \, ; \theta\right) = \frac{\exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_{w_{t+j}})}{\sum_{k \in V} \exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_k)}$$

- Scores can be asymmetric
- It is not likely that a word appears in its own context

How to Train Word Vectors?

Parameters: $\theta = \left\{\{\mathbf{u}_k\}, \{\boldsymbol{v}_k\}\right\}$

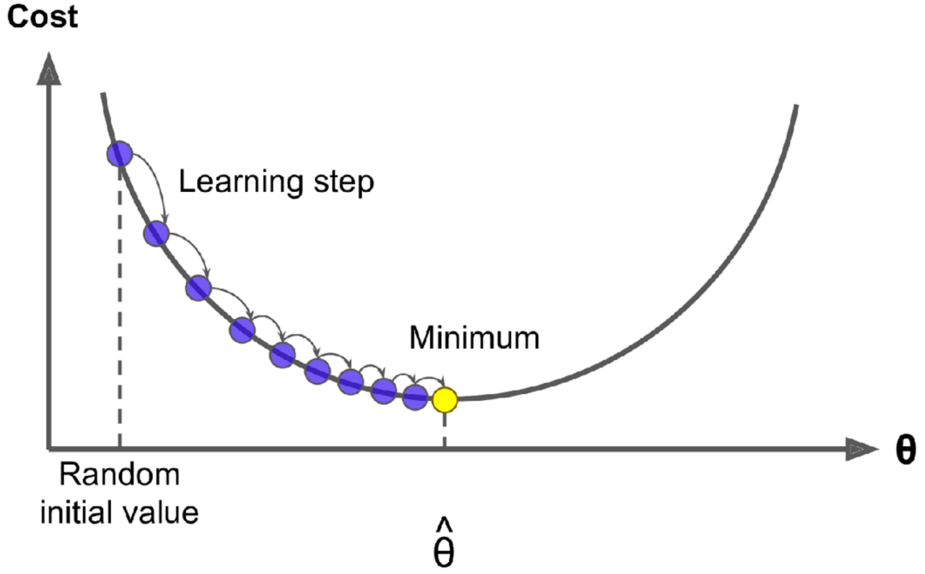
Objective function: $J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{-m\leq j\leq m, j\neq 0} \log P\left(w_{t+j} \middle| w_t\,;\theta\right)$

Our goal: find parameters θ that minimize the objective function $J(\theta)$

Solution: stochastic gradient descent (SGD)

- Randomly initialize parameters θ
- For each iteration $\theta \longleftarrow \theta - \eta\, \nabla_\theta\, J(\theta)$

Learning step   Gradient



https://insightfultscript.com/collections/programming/machine-learning/sgd/

Computing the Gradients

Objective function

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{-m\le j\le m, j\ne 0} \log P\left(w_{t+j} \middle| w_t\, ; \theta\right)$$

$$= \frac{1}{T}\sum_{t=1}^{T}\sum_{-m\le j\le m, j\ne 0} \boxed{-\log P\left(w_{t+j} \middle| w_t\, ; \theta\right)}$$

The gradients can be calculated separately!

For simplicity, we consider one pair of center/context words (o, c)

$$y = -\log P(c|o\, ; \theta) = -\log\left(\frac{\exp(\mathbf{u}_o \cdot \mathbf{v}_c)}{\sum_{k\in V}\exp(\mathbf{u}_o \cdot \mathbf{v}_k)}\right)$$

$$\boxed{\frac{\partial y}{\partial \mathbf{u}_o} \quad \frac{\partial y}{\partial \boldsymbol{v}_c}}$$

We need to compute this!

Computing the Gradients

$$y = -\log P(c|o) = -\log\left(\frac{\exp(\mathbf{u}_o\cdot\mathbf{v}_c)}{\sum_{k\in V}\exp(\mathbf{u}_o\cdot\mathbf{v}_k)}\right) = \boxed{-\log(\exp(\mathbf{u}_o\cdot\mathbf{v}_c))} + \log\left(\sum_{k\in V}\exp(\mathbf{u}_o\cdot\mathbf{v}_k)\right)$$

$$= -\mathbf{u}_o\cdot\mathbf{v}_c$$

$$\frac{\partial\log(x)}{\partial x} = \frac{1}{x} \qquad \frac{\partial\exp(x)}{\partial x} = \exp(x)$$

$$\frac{\partial y}{\partial \mathbf{u}_o} = \frac{\partial\left(-\mathbf{u}_o\cdot\mathbf{v}_c + \log\left(\sum_{k\in V}\exp(\mathbf{u}_o\cdot\mathbf{v}_k)\right)\right)}{\partial \mathbf{u}_o} = -\mathbf{v}_c + \frac{\sum_{k\in V}\frac{\partial\exp(\mathbf{u}_o\cdot\mathbf{v}_k)}{\partial\mathbf{u}_o}}{\sum_{k\in V}\exp(\mathbf{u}_o\cdot\mathbf{v}_k)}$$

$$= -\mathbf{v}_c + \frac{\sum_{k\in V}\exp(\mathbf{u}_o\cdot\mathbf{v}_k)\,\mathbf{v}_k}{\sum_{k\in V}\exp(\mathbf{u}_o\cdot\mathbf{v}_k)} = -\mathbf{v}_c + \sum_{k\in V}\frac{\exp(\mathbf{u}_o\cdot\mathbf{v}_k)\,\mathbf{v}_k}{\sum_{k\in V}\exp(\mathbf{u}_o\cdot\mathbf{v}_k)}$$

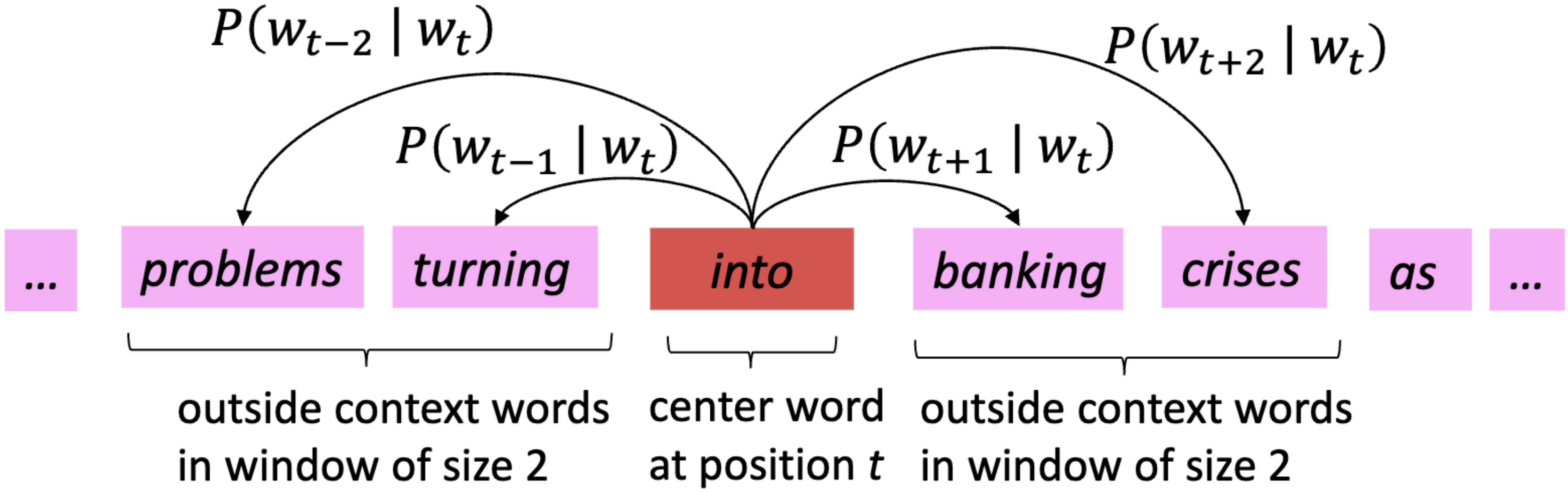
$$= -\mathbf{v}_c + \sum_{k\in V} P(k|o)\,\mathbf{v}_k$$

$$\boxed{\frac{\partial y}{\partial \mathbf{v}_k} = -1(k = c)\mathbf{u}_o + P(k|o)\mathbf{u}_o}$$

Similar calculation step

Training Process

- Randomly initialize parameters $\mathbf{u}_i, \mathbf{v}_i$
- Walk through the training corpus and collect training data (o, c)



$$\mathbf{u}_o \leftarrow \mathbf{u}_o - \eta \frac{\partial y}{\partial \mathbf{u}_o} \qquad \mathbf{v}_k \leftarrow \mathbf{v}_k - \eta \frac{\partial y}{\partial \mathbf{v}_k} \quad \forall k \in V$$

Negative Sampling

Issue: every time we get one pair of (o, c), we have to update $\mathbf{v}_k$ with all the words in the vocabulary.

$$\mathbf{u}_o \leftarrow \mathbf{u}_o - \eta \frac{\partial y}{\partial \mathbf{u}_o} \qquad \mathbf{v}_k \leftarrow \mathbf{v}_k - \eta \frac{\partial y}{\partial \mathbf{v}_k} \quad \forall k \in V$$

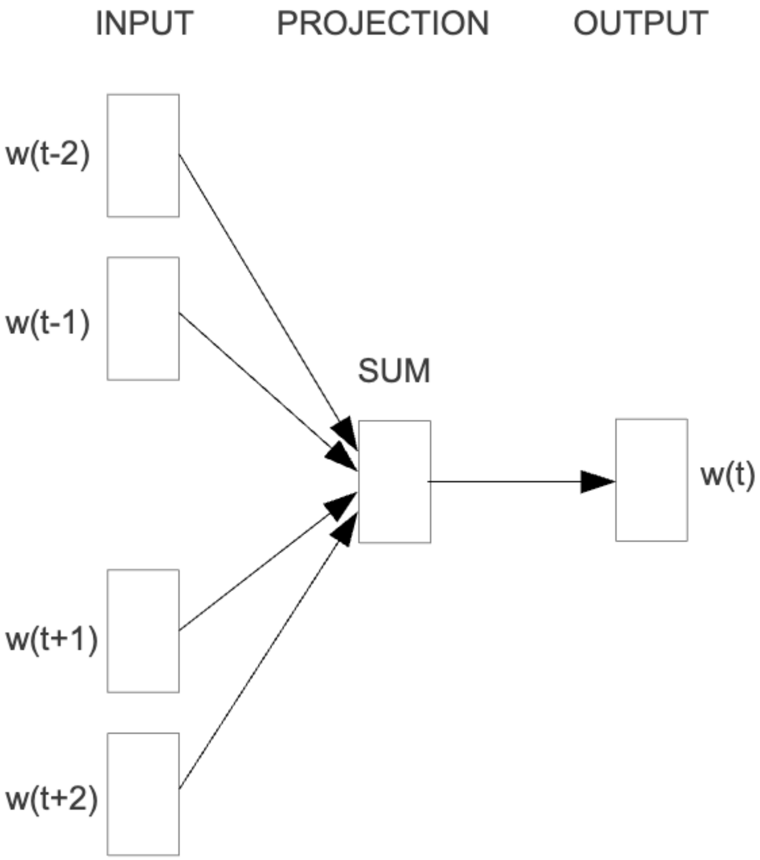
Negative sampling: instead of considering all the words in V, we randomly sample K(5-20) negative examples

Softmax $$y = -\log\left(\frac{\exp(\mathbf{u}_o \cdot \mathbf{v}_c)}{\sum_{k\in V} \exp(\mathbf{u}_o \cdot \mathbf{v}_k)}\right) = -\log(\exp(\mathbf{u}_o \cdot \mathbf{v}_c)) + \log\left(\sum_{k\in V} \exp(\mathbf{u}_o \cdot \mathbf{v}_k)\right)$$

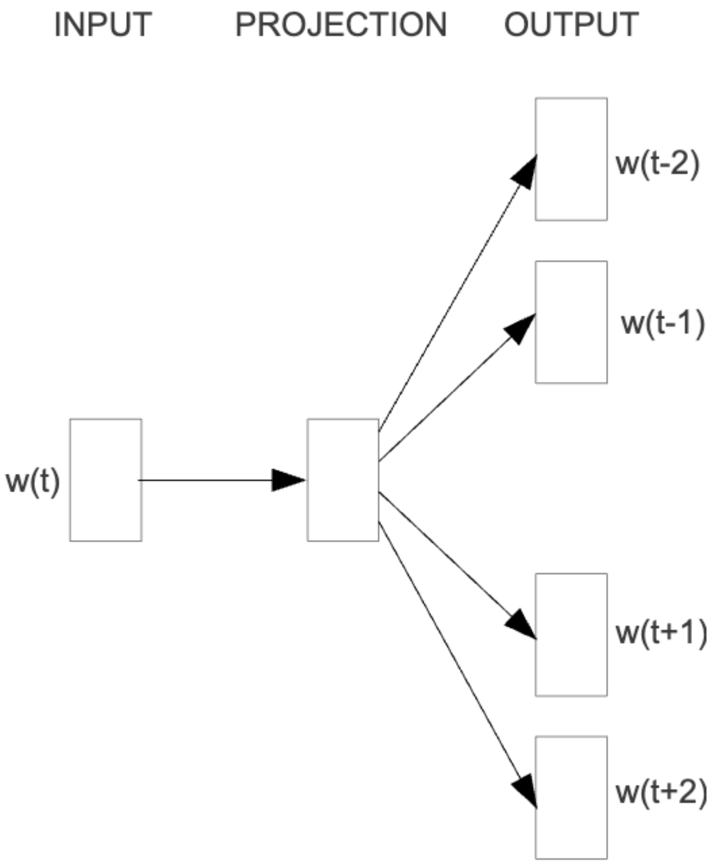
Negative sampling $$y = -\log(\sigma(\mathbf{u}_o \cdot \mathbf{v}_c)) - \sum_{i=1}^{K} \mathbb{E}_{j\sim P(w)} \log(\sigma(-\mathbf{u}_o \cdot \mathbf{v}_j))$$

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

Continuous Bag of Words (CBOW) vs Skip-Grams



CBOW



Skip-gram