Neural Word Embeddings

CS-585

Natural Language Processing

Derrick Higgins

Word vectors: problems

- We saw last time how to build vectors to represent words:
  - One-hot encoding
    - Binary, count, tf*idf representations
  - Hashing trick

$$\text{cat} = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}, \text{dog} = \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \\ 0 \end{bmatrix}, \text{mouse} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \\ 0 \end{bmatrix}, \text{bird} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}$$

- Some problems
  - Large dimensionality of word vectors
  - Lack of meaningful relationships between words

Word vectors: problems

$$\text{cat} = \begin{bmatrix}1\\0\\\vdots\\0\\0\end{bmatrix}, \text{dog} = \begin{bmatrix}0\\1\\\vdots\\0\\0\end{bmatrix}, \text{mouse} = \begin{bmatrix}0\\0\\\vdots\\1\\0\end{bmatrix}, \text{bird} = \begin{bmatrix}0\\0\\\vdots\\0\\1\end{bmatrix}$$

$$CosineSimilarity(cat, dog) = \frac{v_{cat} \cdot v_{dog}}{\|v_{cat}\|\|v_{dog}\|} = 0$$

D_1="the **cat** and the **mouse**" $= \begin{bmatrix}1\\0\\\vdots\\1\\0\end{bmatrix}$, D_2="the **cat** and the **bird**" $= \begin{bmatrix}1\\0\\\vdots\\0\\1\end{bmatrix}$

$$CosineSimilarity(D_1, D_2) = \frac{D_1 \cdot D_2}{\|D_1\|\|D_2\|} = \frac{1}{\sqrt{2}\sqrt{2}} = \frac{1}{2}$$

How to learn word meanings

1. Look in a dictionary/thesaurus
   - "Dictionary-based learning"
2. Ask people
   - Word association and similarity judgement tasks
3. Distributional hypothesis
   - Words that occur in the same contexts tend to be similar in meaning

Distributional hypothesis

You shall know a word by the company it keeps

J.R. Firth, 1957

...if we consider words or morphemes A and B to be more different in meaning than A and C, then we will often find that the distributions of A and B are more different than the distributions of A and C. In other words, difference of meaning correlates with difference of distribution.

Zellig Harris, 1970

Contexts and word meanings

ed to this height .) **Apple** trees grew there also
referred to get their **apple** pies at the local bak
cally this : Wright's **apple** pie ; ; peel , core ,
ard for mom and mom's **apple** pie goes with : Af In
ned by drinking sweet **apple** juice in which pulver
ile as big as a small **apple** . The odor here was m
an upper bough of the **apple** tree . The primary qu
oo small to reach the **apple** tree bough , to twist

ly shaped like a large **pear** , and when properly ri
ncy of a ripe Bartlett **pear** , but oily . The avoca
neither sweet , like a **pear** , nor tart like an ora
side of the house to a **pear** tree , with crowds alr
d to that of `` a rich **pear** '' . Though she did no
C was at the DiGiorgio **pear** orchards in Yuba Count

e strode proudly into **Orange** Square , smiling like
e to a stretch of old **orange** groves , the tree dec
the juice out of the **orange** , now is it '' ? ? ``
robbing wall of fiery **orange** brown haze . Ben Prim
owers squirting great **orange** billows . A wave of f
up into one of those **orange** streetcars , rode awa
r the baby , milk and **orange** juice and vitamins an
n , nine mint , seven **orange** -- around the curve t
e thought . A shot of **orange** juice would make ever

Context may be a word in the neighborhood, a local syntactic relation, or document-level occurrence

USING THE DISTRIBUTIONAL HYPOTHESIS TO LEARN WORD REPRESENTATIONS

Latent Semantic Analysis

- One way to use the distributional hypothesis to learn word vectors is called *latent semantic analysis* (Landauer & Dumais, 1997)
- The idea is to construct a matrix that represents words and their context, and then create a reduced-dimensionality version of that matrix that preserves the most distinctive and important characteristics of words' contextual associations

Term-by-document matrix

- Recall that given a sparse encoding of our words as vectors in $\mathbb{R}^{|V|}$, where V is our vocabulary, we can create a vector for a document by aggregating over the vectors for all the words it contains
  - Binary, count, tf*idf vectorization
- So for $\mathrm{V} = \{the, fox, dog, lazy, jumped, over\}$, using a count vectorizer, we have

"the lazy fox jumped" $\rightarrow [1 \quad 1 \quad 0 \quad 1 \quad 1 \quad 0]$
"the fox jumped over the dog" $\rightarrow [2 \quad 1 \quad 1 \quad 0 \quad 1 \quad 1]$
"the lazy dog" $\rightarrow [1 \quad 0 \quad 1 \quad 1 \quad 0 \quad 0]$
"the dog jumped over the dog" $\rightarrow [2 \quad 0 \quad 2 \quad 0 \quad 1 \quad 1]$
"the fox jumped" $\rightarrow [1 \quad 1 \quad 0 \quad 0 \quad 1 \quad 0]$

Term-by-document matrix

"the lazy fox jumped" $\rightarrow [1 \quad 1 \quad 0 \quad 1 \quad 1 \quad 0]$

"the fox jumped over the dog" $\rightarrow [2 \quad 1 \quad 1 \quad 0 \quad 1 \quad 1]$

"the lazy dog" $\rightarrow [1 \quad 0 \quad 1 \quad 1 \quad 0 \quad 0]$

"the dog jumped over the dog" $\rightarrow [2 \quad 0 \quad 2 \quad 0 \quad 1 \quad 1]$

"the fox jumped" $\rightarrow [1 \quad 1 \quad 0 \quad 0 \quad 1 \quad 0]$

D_1	1	1	0	1	1	0
D_2	2	1	1	0	1	1
D_3	1	0	1	1	0	0
D_4	2	0	2	0	1	1
D_5	1	1	0	0	1	0

Term-by-document matrix

	V_1 = the	V_2 = fox	V_3 = dog	V_4 = lazy	V_5 = jumped	V_6 = over
D_1	1	1	0	1	1	0
D_2	2	1	1	0	1	1
D_3	1	0	1	1	0	0
D_4	2	0	2	0	1	1
D_5	1	1	0	0	1	0

Term-by-document matrix

$$v_2 = [1 \quad 1 \quad 0 \quad 0 \quad 1]$$
$$v_4 = [1 \quad 0 \quad 1 \quad 0 \quad 0]$$

	V_1 = the	V_2 = fox	V_3 = dog	V_4 = lazy	V_5 = jumped	V_6 = over
D_1	1	1	0	1	1	0
D_2	2	1	1	0	1	1
D_3	1	0	1	1	0	0
D_4	2	0	2	0	1	1
D_5	1	1	0	0	1	0

Word vectors in term-by-document matrix

$$\begin{aligned}
\text{the} &= v_1 = [1 \quad 2 \quad 1 \quad 2 \quad 1] \\
\text{fox} &= v_2 = [1 \quad 1 \quad 0 \quad 0 \quad 1] \\
\text{dog} &= v_3 = [0 \quad 1 \quad 1 \quad 2 \quad 0] \\
\text{lazy} &= v_4 = [1 \quad 0 \quad 1 \quad 0 \quad 0] \\
\text{jumped} &= v_5 = [1 \quad 1 \quad 0 \quad 1 \quad 1] \\
\text{over} &= v_6 = [0 \quad 1 \quad 0 \quad 1 \quad 0]
\end{aligned}$$

Each word is represented as a vector of values related to that word's occurrence in a document

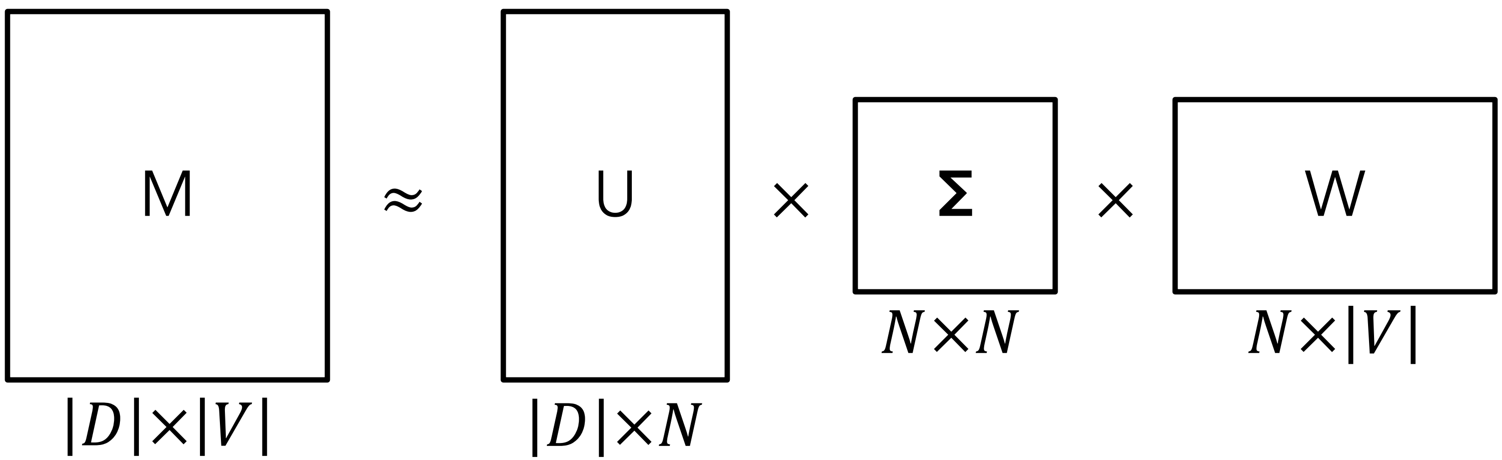
- Instead of $\mathbb{R}^{|V|}$, each vector is now in $\mathbb{R}^{|D|}$. (Vector length = number of documents.)
  - Still pretty high dimension
- Generally, $v_i \cdot v_j \neq 0$! (Vectors are not orthogonal.)

Latent semantic analysis

- The idea is to "compress" the representation of a word, using only $M \ll |D|$ dimensions for each vector
  - Compress for more efficient representation (smaller memory footprint)
  - Compress for *generalization*: retain only most important information, and allow distinctions between similar words to be obscured
- How to do this automatically?

Singular value decomposition

For a term-by-document matrix M, based on documents D and vocabulary V, we approximate



U is an orthogonal matrix with one row per document

W is an orthogonal matrix with one column per word

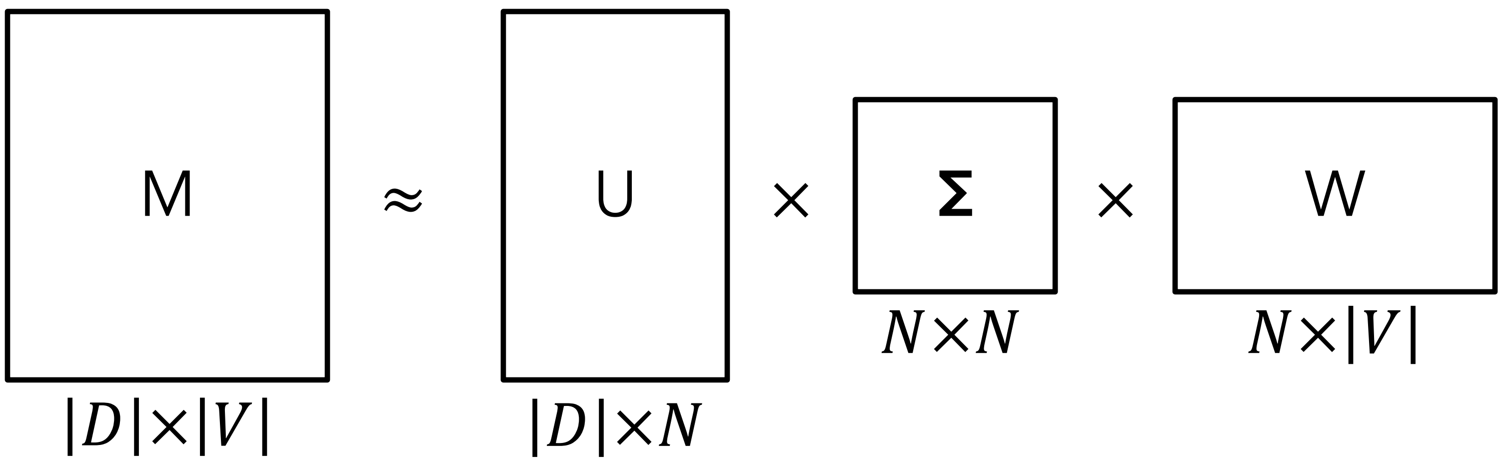
Σ is a diagonal matrix of "singular values"

Singular value decomposition

$$\underbrace{M}_{|D|\times|V|} \approx \underbrace{U}_{|D|\times N} \times \underbrace{\boldsymbol{\Sigma}}_{N\times N} \times \underbrace{W}_{N\times|V|}$$

Cannot be factored exactly, because $N \ll \min(|D|, |V|)$

Singular value decomposition



Approximate factorization can be done with the objective of minimizing $\sum_{i=1}^{|D|} \sum_{j=1}^{|V|} \left(M_{ij} - \widehat{M}_{ij}\right)^2$, where $\widehat{M} = U \times \Sigma \times W$

LSA: example

- [Notebook]

Related approaches

- pLSA: Word vectors based on document-based co-occurrence, but in graphical modeling framework
- Non-negative matrix factorization (NNMF): Constrain matrices in decomposition to include no negative values (for interpretability)

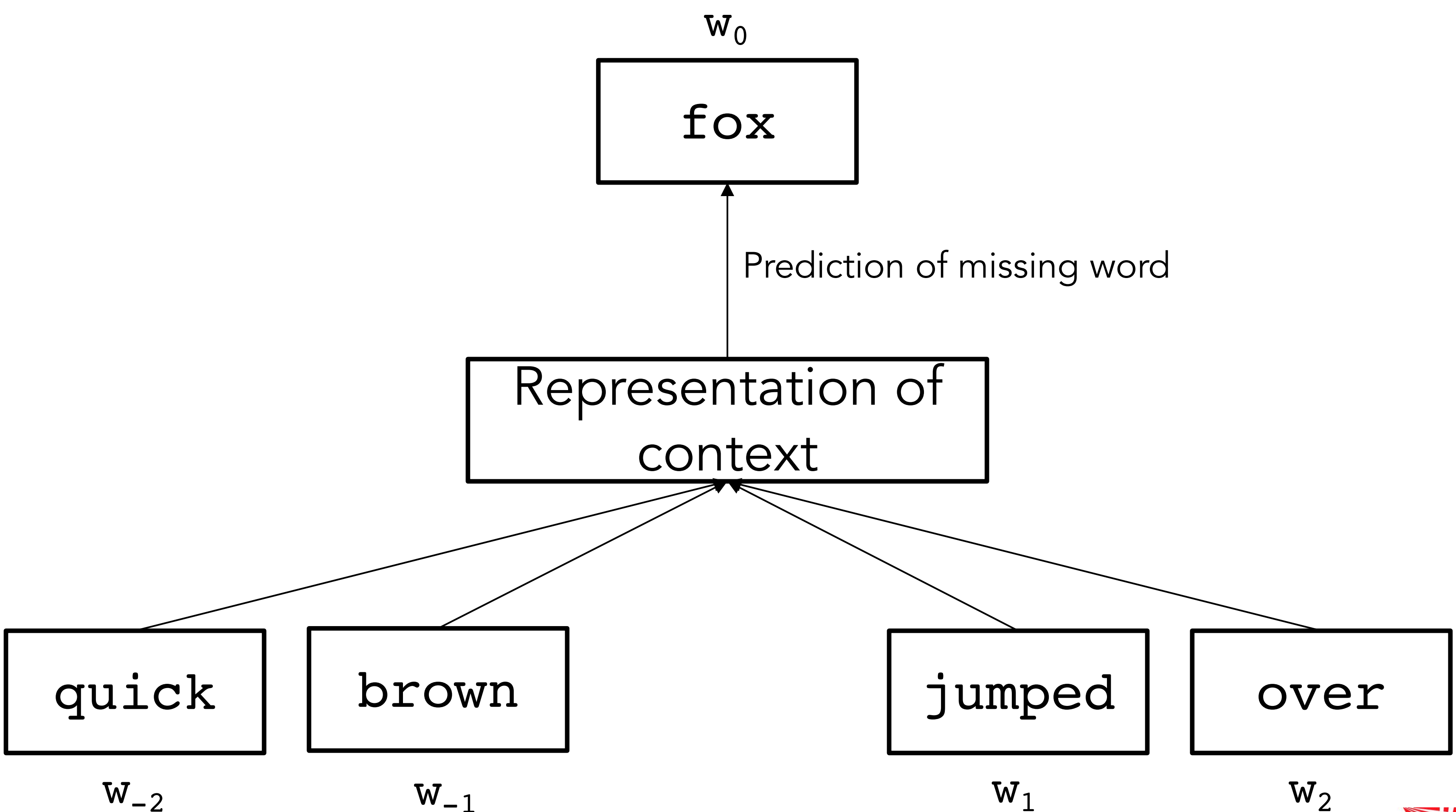
Neural Word Embeddings: `word2vec`

- Efficient and widely-used method for getting word embeddings (vectors) using a neural network framework
- What is a neural network?
  - Uses vector/matrix/tensor representations
  - Applies a sequence of algebraic operations (matrix multiplication, etc.)
  - Trained using some variant of gradient descent

`word2vec` model

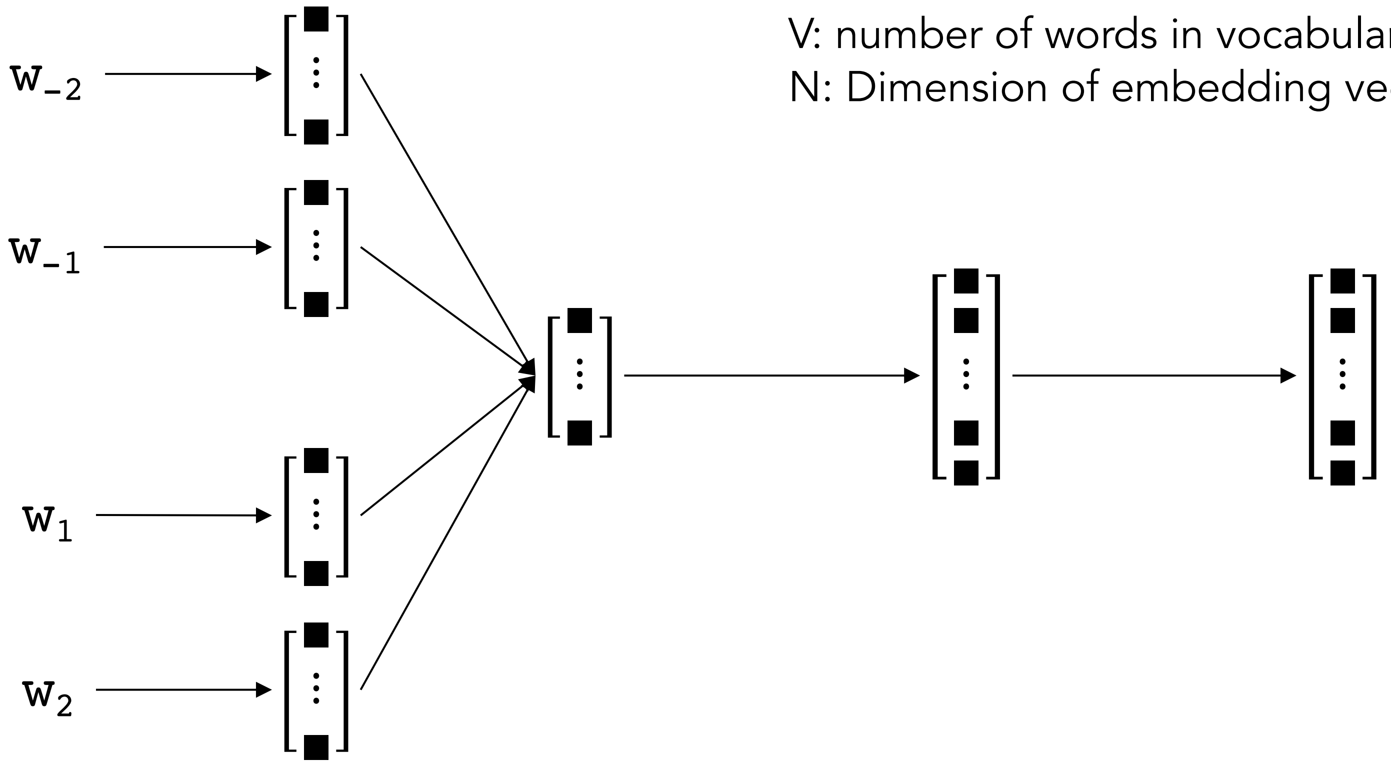
- The idea is that we will build a model that tries to predict the word that will show up in a given context
  - Alternatively, predict the context that is appropriate for a given word
- In the course of trying to do this task accurately, the model will be forced to select which information about a word/context is most important to represent
- *Representation learning* – automatically learning useful features for a task, rather than manually creating them
  - In this case, we want to learn a vector of useful attributes of words, rather than recording things like “is a noun”, “refers to a person”, “female gender”, etc.

word2vec
Contextual Bag of Words (CBOW)

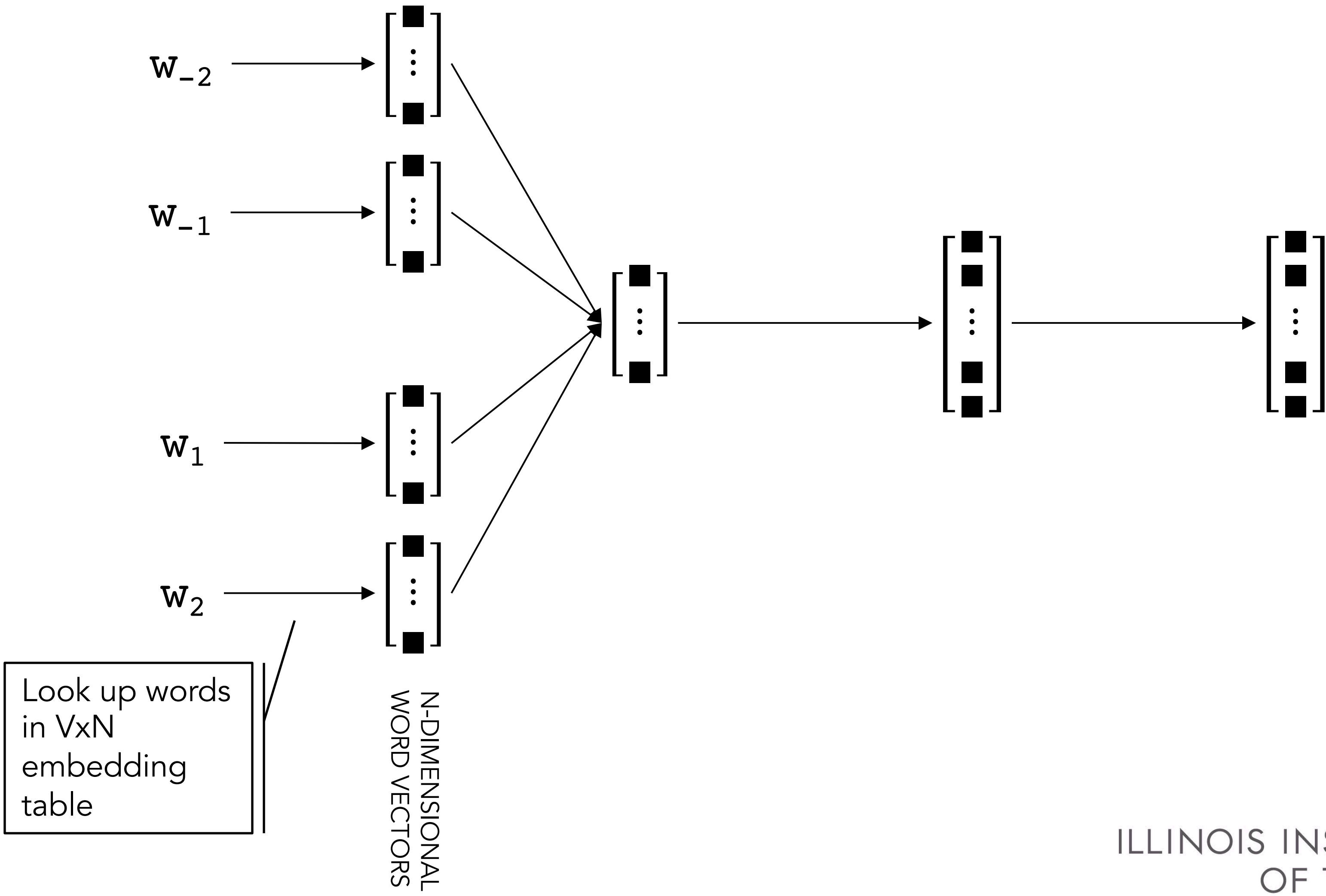


word2vec
Contextual Bag of Words (CBOW)

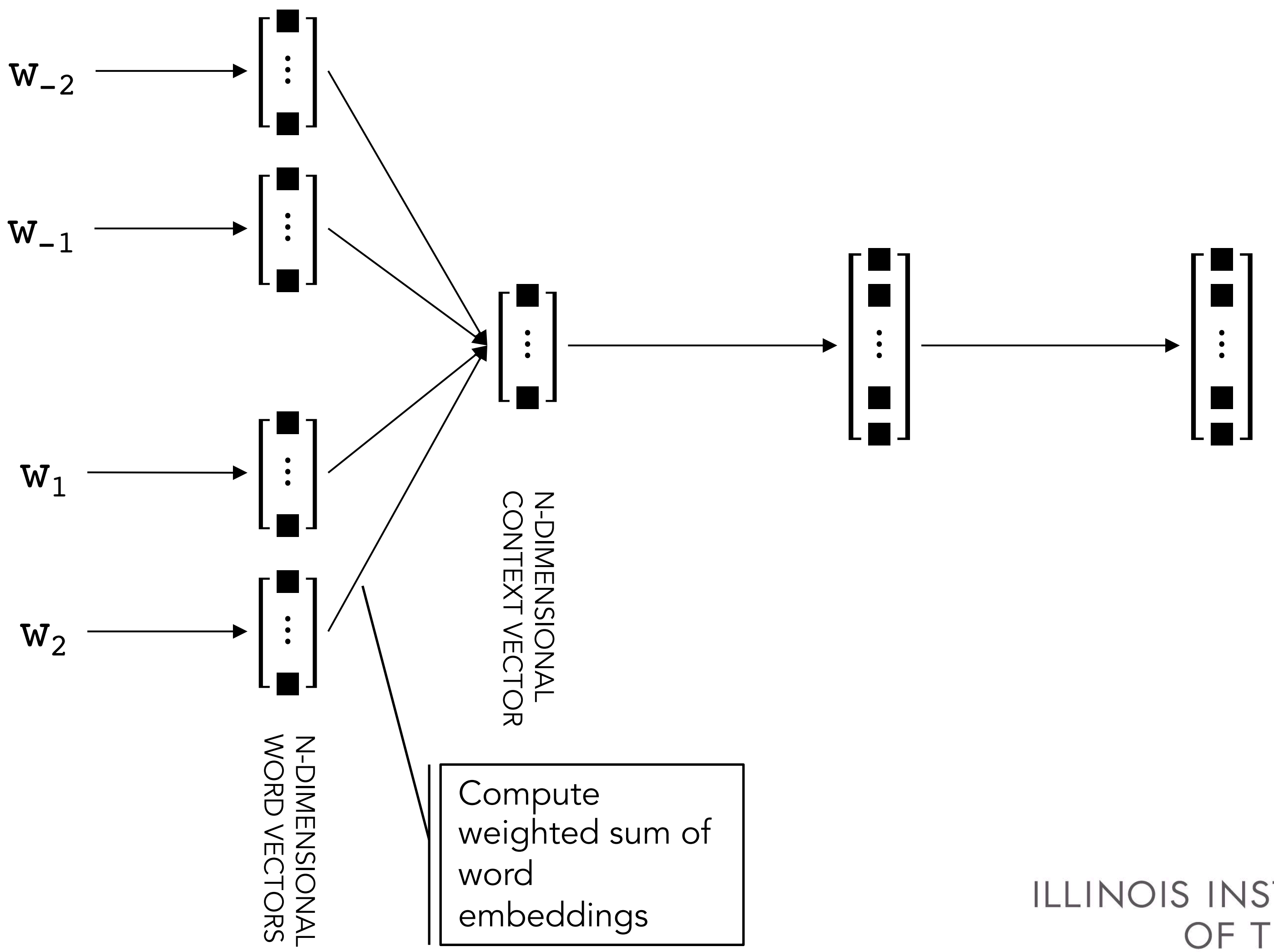
V: number of words in vocabulary
N: Dimension of embedding vectors



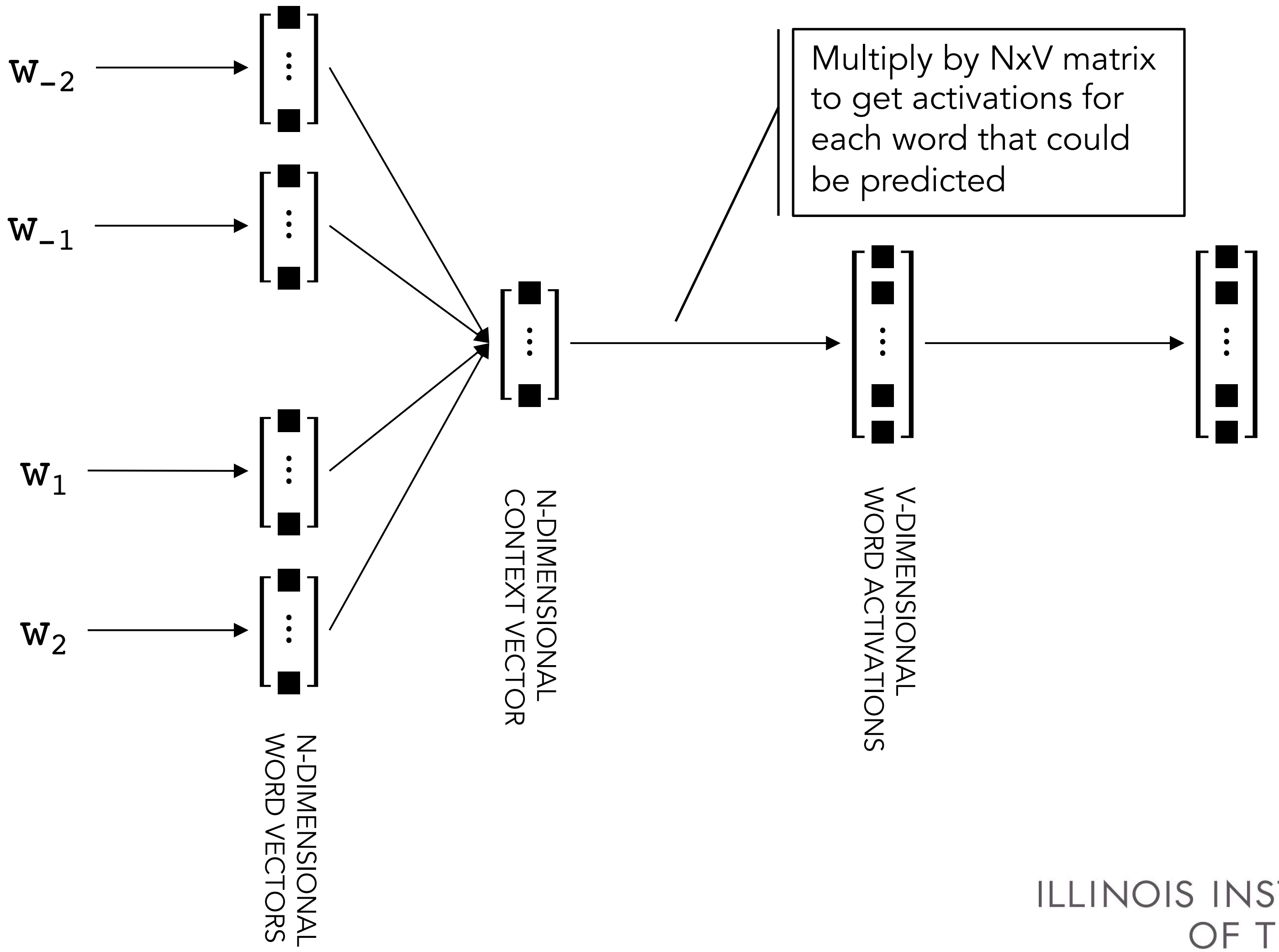
word2vec
Contextual Bag of Words (CBOW)



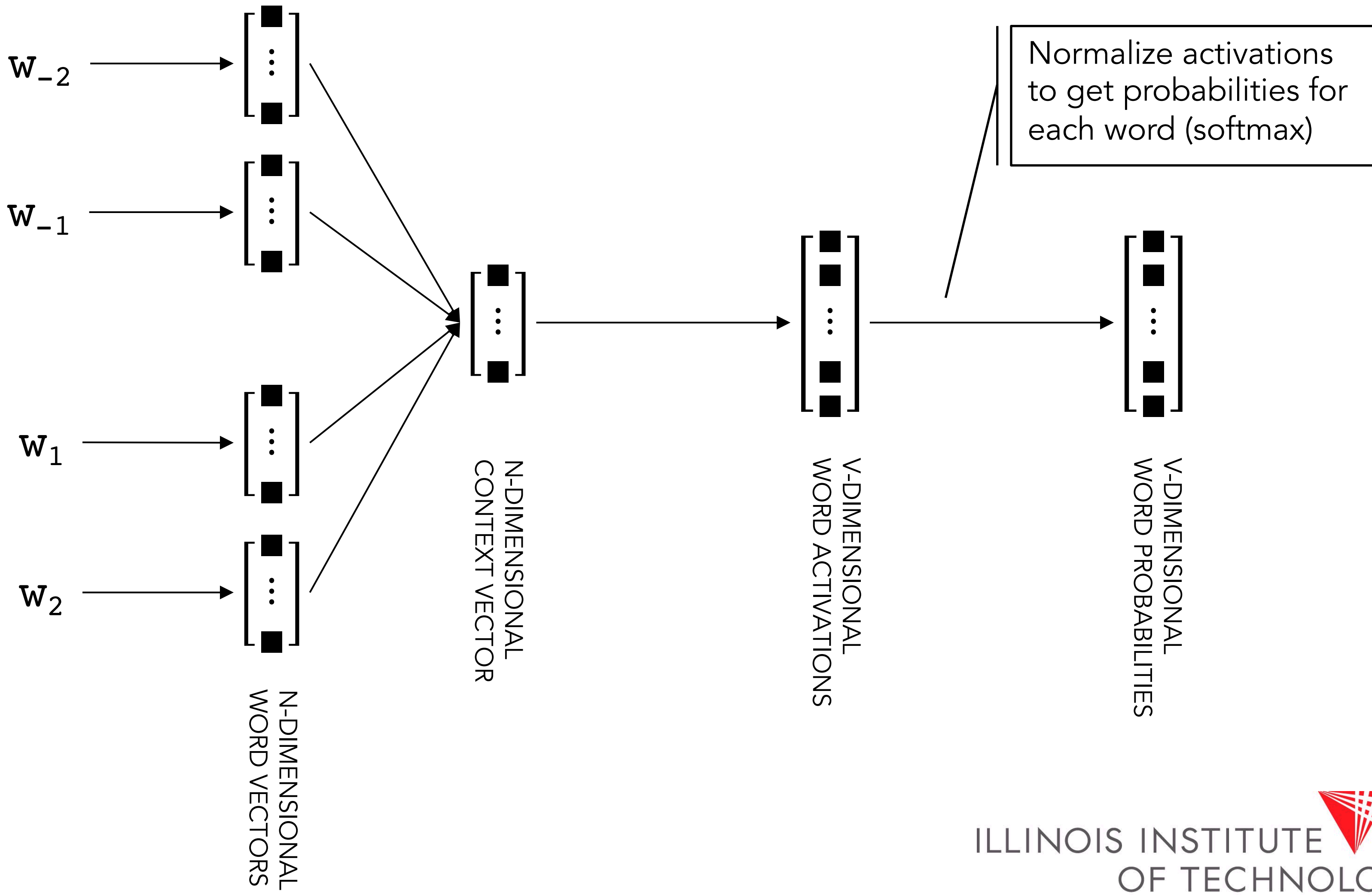
word2vec
Contextual Bag of Words (CBOW)



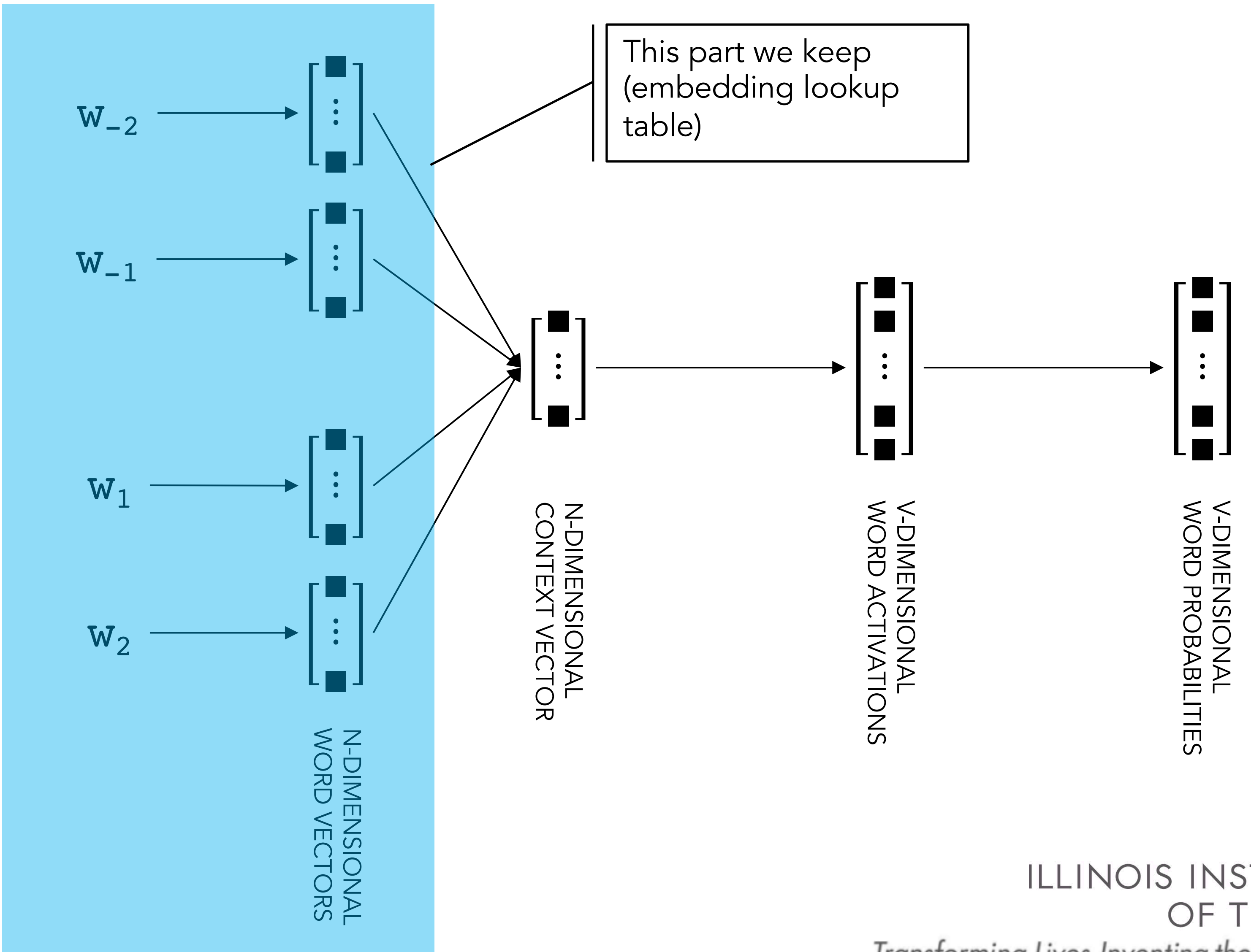
word2vec
Contextual Bag of Words (CBOW)



word2vec
Contextual Bag of Words (CBOW)



word2vec
Contextual Bag of Words (CBOW)



`word2vec` Contextual Bag of Words (CBOW)

$$P(w_0|w_{-2},w_{-1},w_1,w_2) = Softmax\left(\left(\sum_{i\in\{-2,-1,1,2\}} a_i E_{w_i}\right)\times P\right)$$

- E is the VxN matrix of word embeddings
- P is the NxV matrix mapping contextual representations to word predictions
- a_i is the weighting of a word at location i in the contextual representation (words closer to the target position are weighted more highly)

Softmax

- A convenient way to get a proper probability distribution out of an unconstrained vector of values
- Maps a vector with values in $[-\infty, \infty]$ to a vector with values in $[0,1]$ that sum to one
- Also conveniently differentiable (see *backpropagation*)

$$Softmax(\vec{x}) = \left[\frac{e^{\vec{x}_i}}{\sum_{\forall j} e^{\vec{x}_j}} \right]_{\forall i}$$

Softmax

$$Softmax\left(\begin{bmatrix}-1\\0\\1.5\end{bmatrix}\right) = Normalize\left(\begin{bmatrix}e^{-1}\\e^{0}\\e^{1.5}\end{bmatrix}\right)$$

$$= Normalize\left(\begin{bmatrix}0.37\\1\\4.48\end{bmatrix}\right)$$

$$= \begin{bmatrix}\dfrac{.37}{.37+1+4.48}\\\dfrac{1}{.37+1+4.48}\\\dfrac{4.48}{37+1+4.48}\end{bmatrix} = \begin{bmatrix}0.06\\0.17\\0.77\end{bmatrix}$$

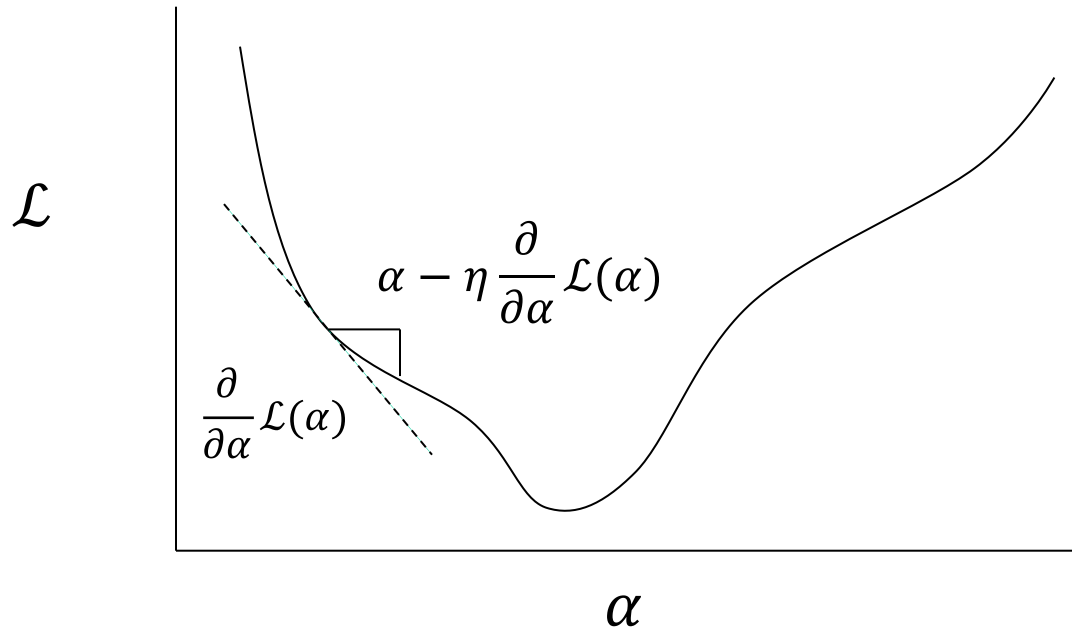
Backpropagation and gradient descent

Error backpropagation is a general framework for learning the parameters of a model (especially, neural network)

1. Define a loss function $\mathcal{L}$ to be minimized
   – Must be differentiable
2. Calculate the derivatives of the loss with respect to its inputs
   – Vector of derivatives derived from vector of inputs: gradient ($\nabla\mathcal{L}$)
3. Use the chain rule of differentiation to get the derivatives of the loss for all parameters in the model
   – $\frac{\partial}{\partial x}(u(v(x))) = \frac{\partial}{\partial v}(u(v))\,\frac{\partial}{\partial x}(v(x))$
4. Update all parameters in the direction of the negative gradient
   – "gradient descent"

Gradient descent

- Skiing downhill
  - start with a given parameter value
  - Find the slope of the loss function
  - Take a step in the downward direction



Gradient descent for CBOW

1. Loss function $\mathcal{L}$ is the cross-entropy:

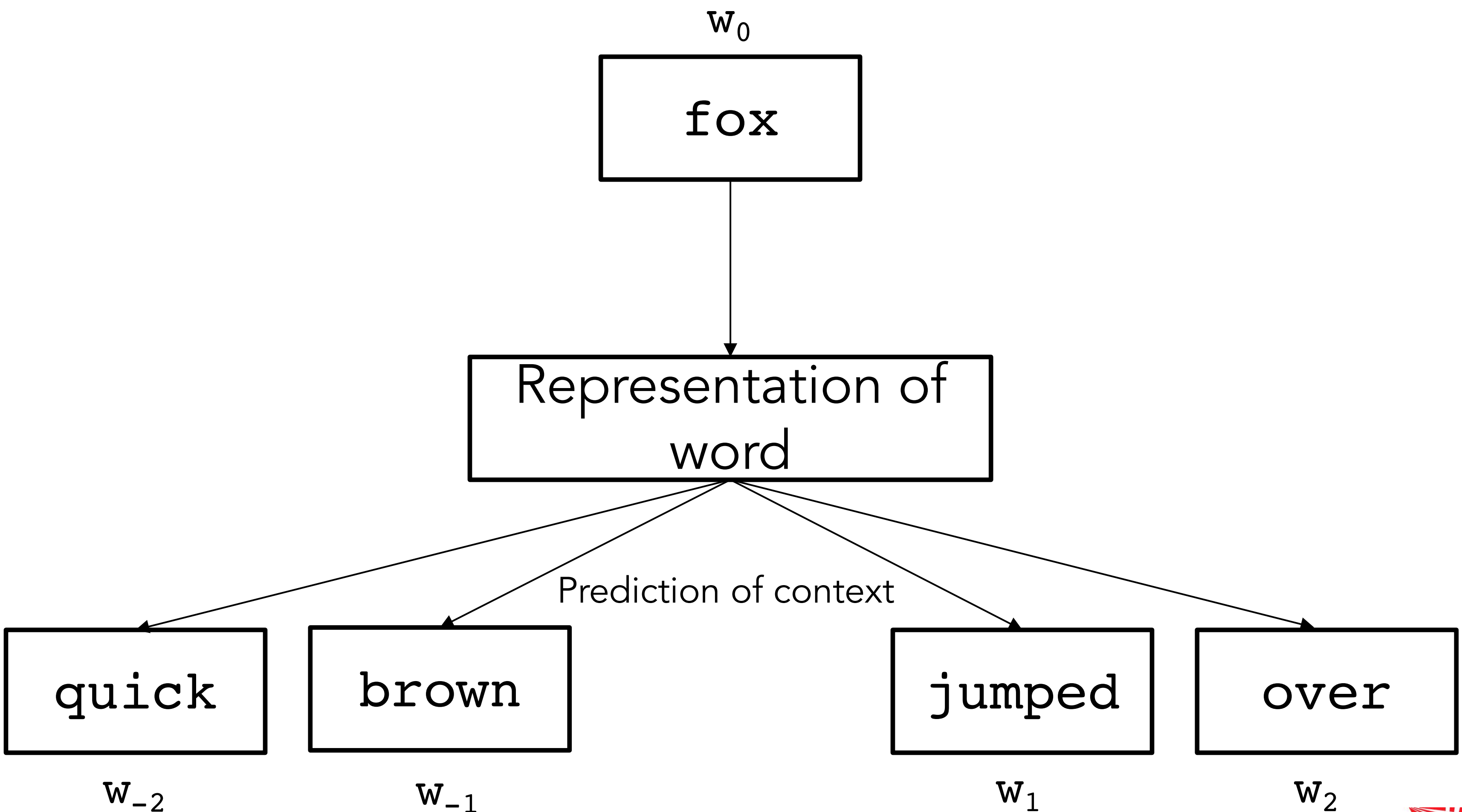
$$\mathcal{L} = -\sum_{w \in V} y_w \log \hat{p}(w)$$

   - y_w is an indicator variable with value 1 when w is the correct word w^*, and 0 otherwise
   - $\hat{p}(w)$ is the model's estimated probability for word w
   - So the loss function simplifies to $\mathcal{L} = -\log \hat{p}(w^*)$
2. Calculate the derivatives of the loss
   - In terms of the pre-softmax activations, $\nabla_{a_w}\mathcal{L} = \hat{p}(w) - y_w$
3. Use the chain rule of differentiation…
4. Update all parameters
   - For all word embeddings, model weights ϕ: $\phi \coloneqq \phi - \eta \nabla \mathcal{L}$

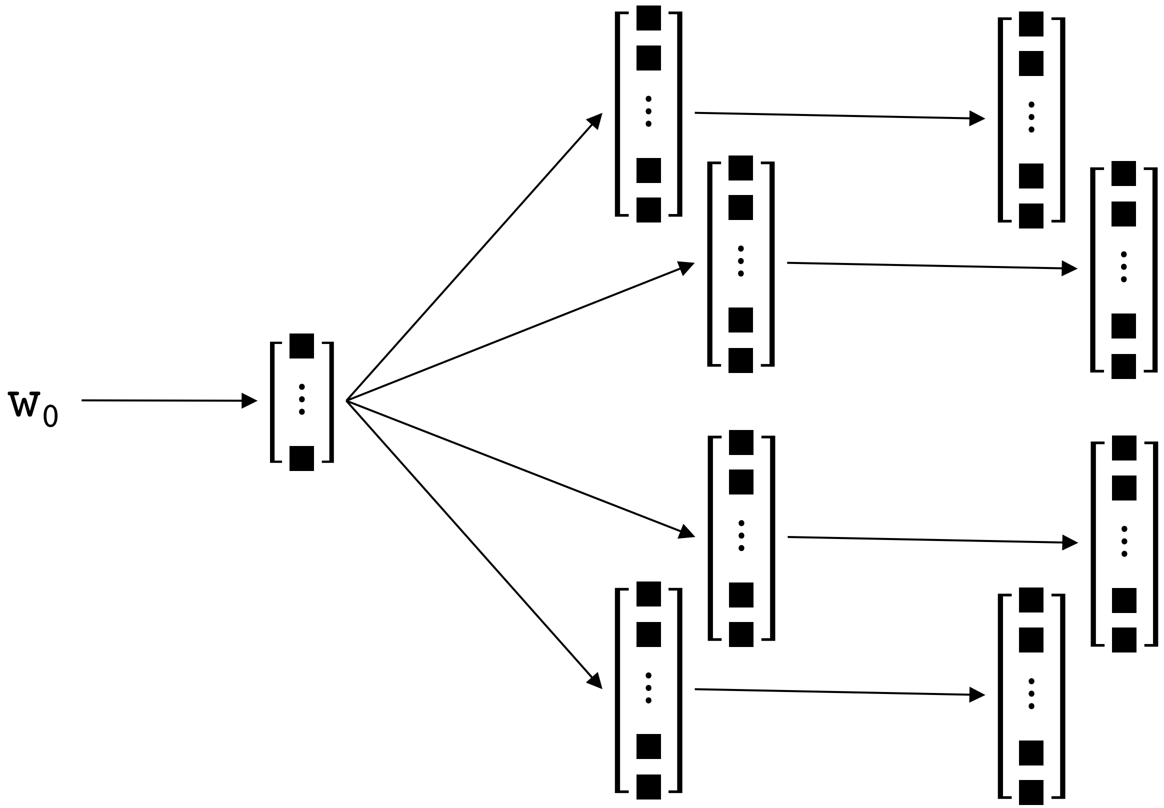
Computational considerations

- Conveniently, calculating the cross-entropy loss only requires $\hat{p}(w^*)$, not $\hat{p}(w)$ for all values of w
- Inconveniently, the gradients still require all values of $\hat{p}(w)$
  - Final matrix multiplication in model is $O(V \times D)$
- Solution 1: *hierarchical softmax*
  - Instead of a single $O(V \times D)$ matrix multiplication to get word activations from embedding layer, break operation down into sequential binary predictions. Complexity reduced to $O(\log V \times D)$
- Solution 2: *negative sampling*
  - Update weights for only a small sample of "negative words" $w \neq w^*$ per iteration

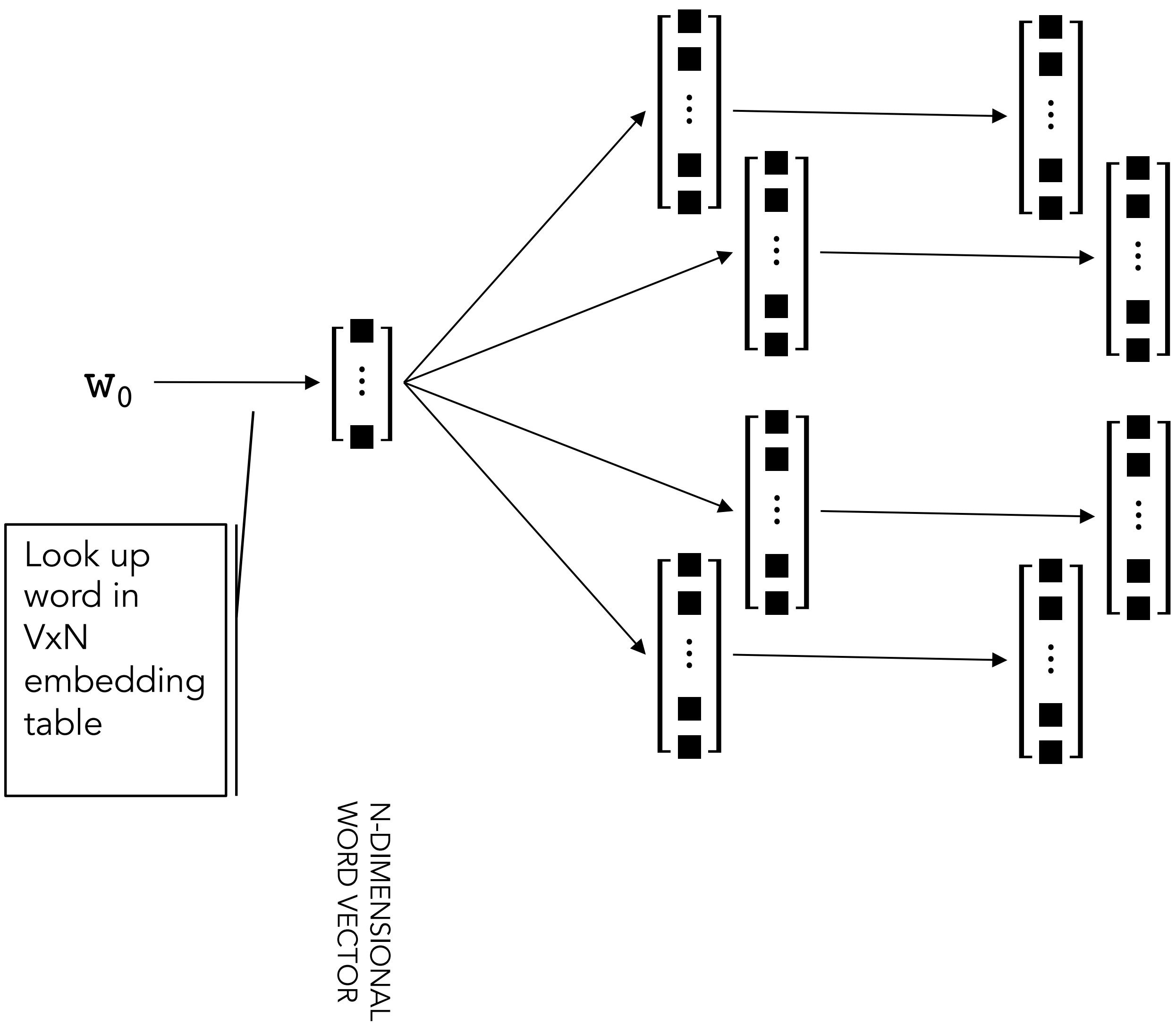
word2vec
Skip-gram



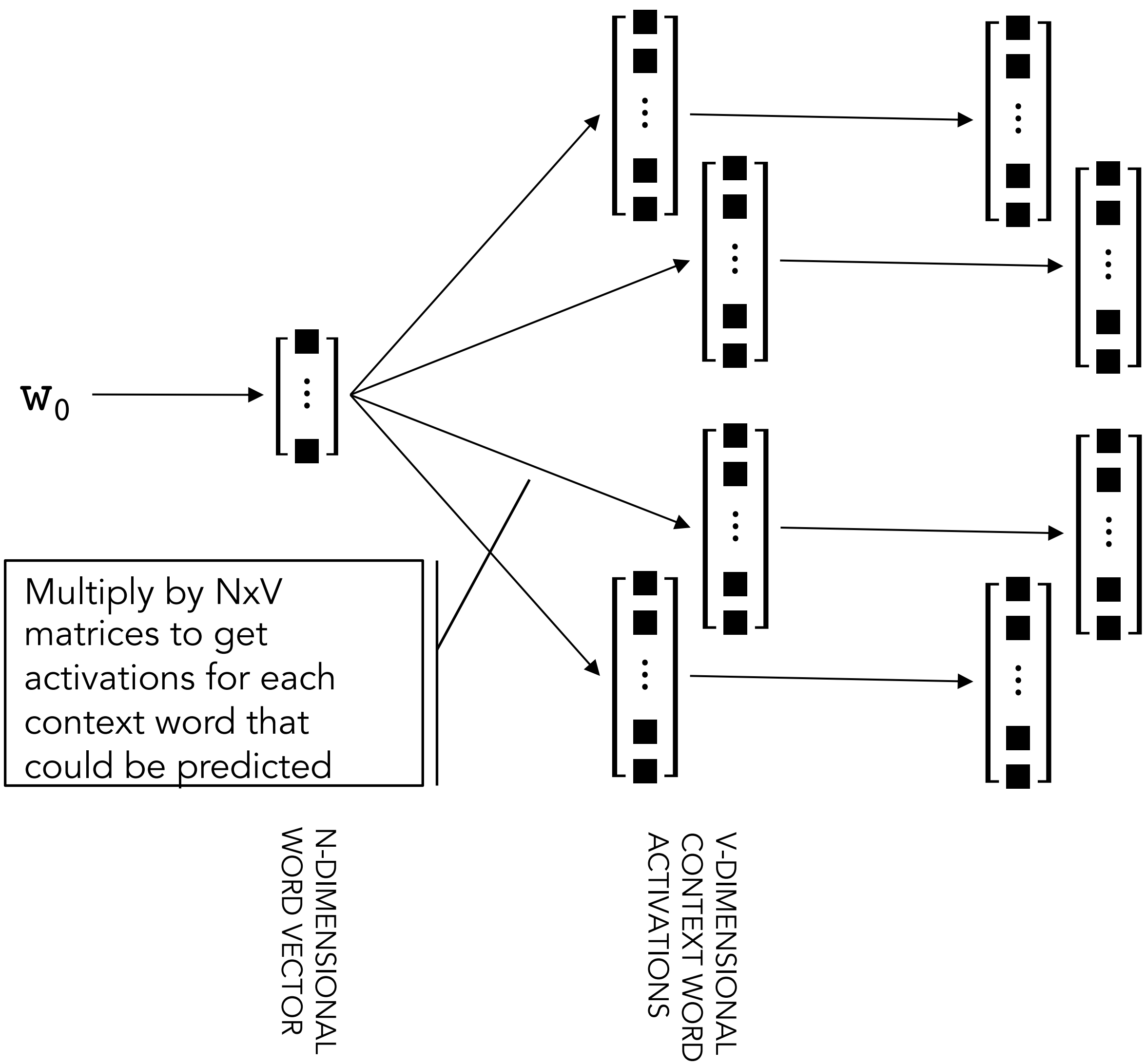
word2vec
Skip-gram



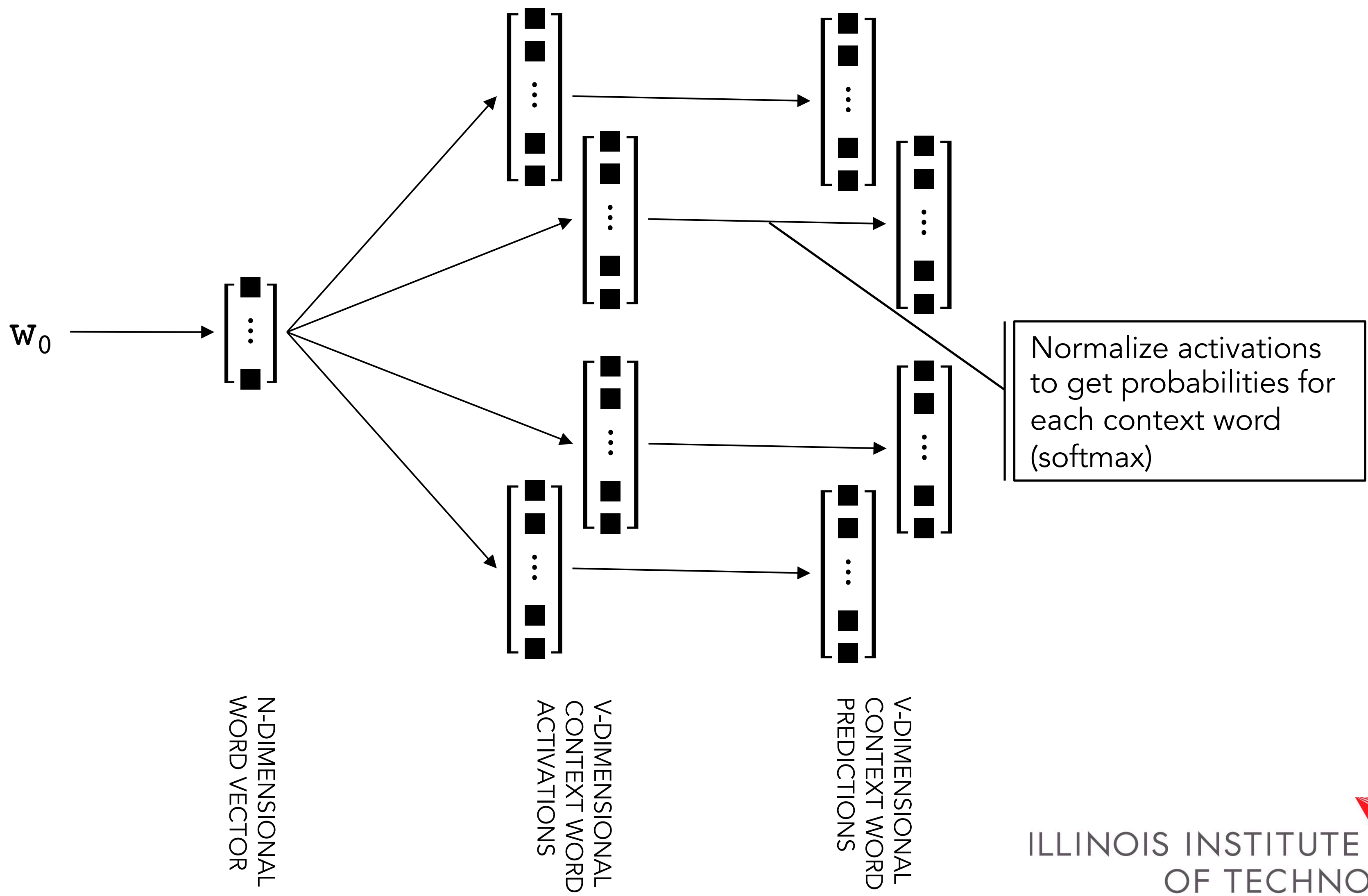
word2vec
Skip-gram



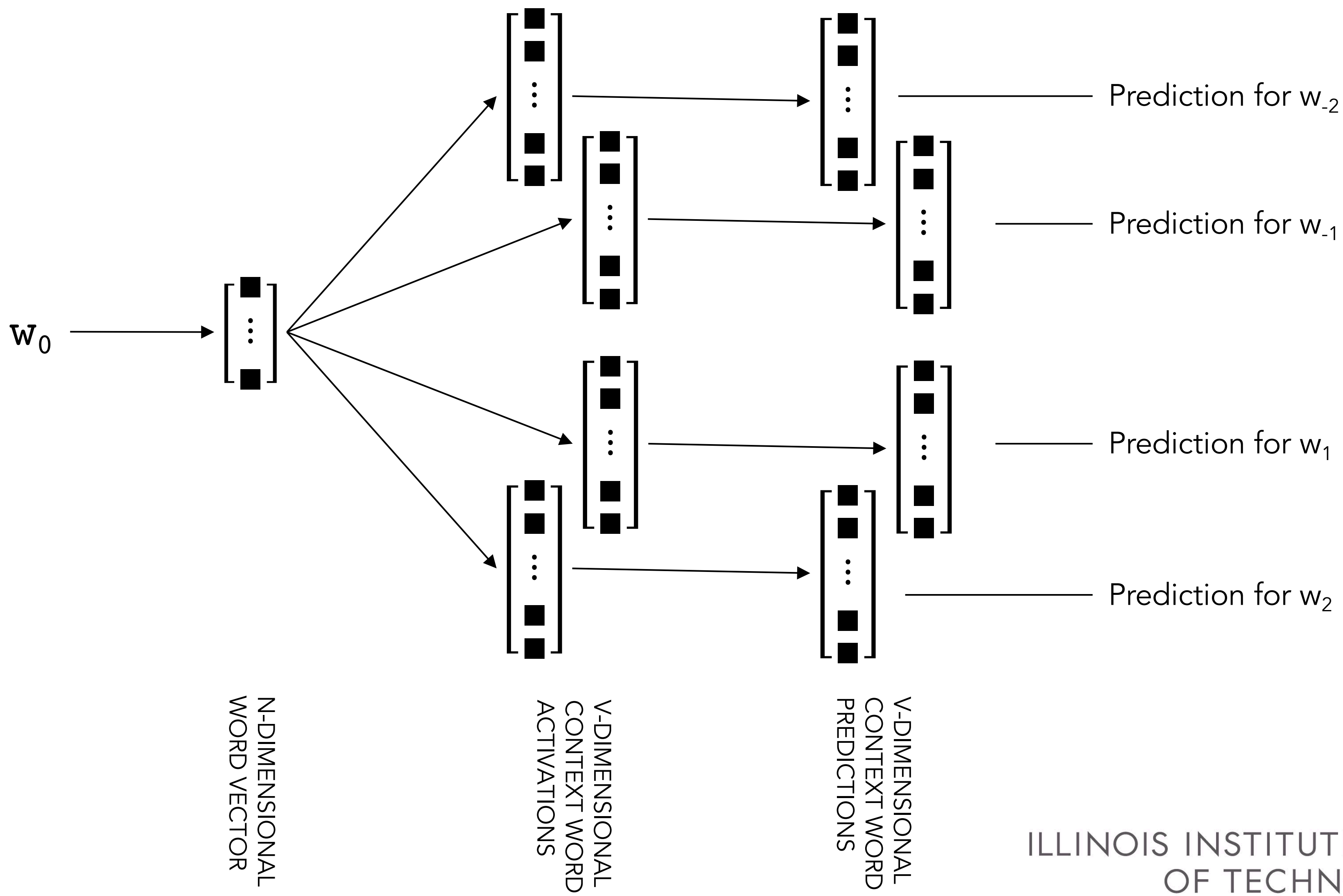
word2vec
Skip-gram



word2vec
Skip-gram

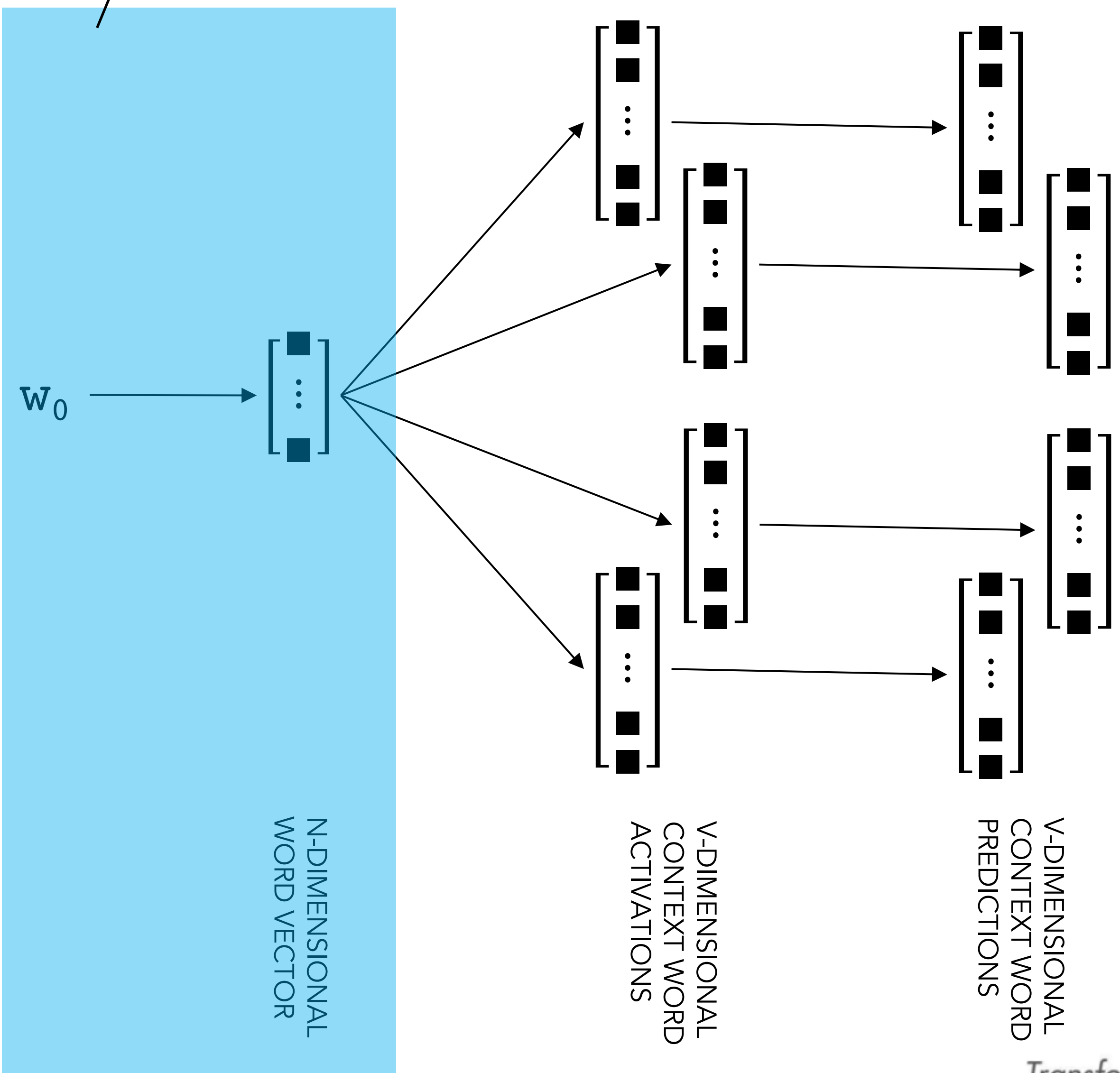


`word2vec`
Skip-gram



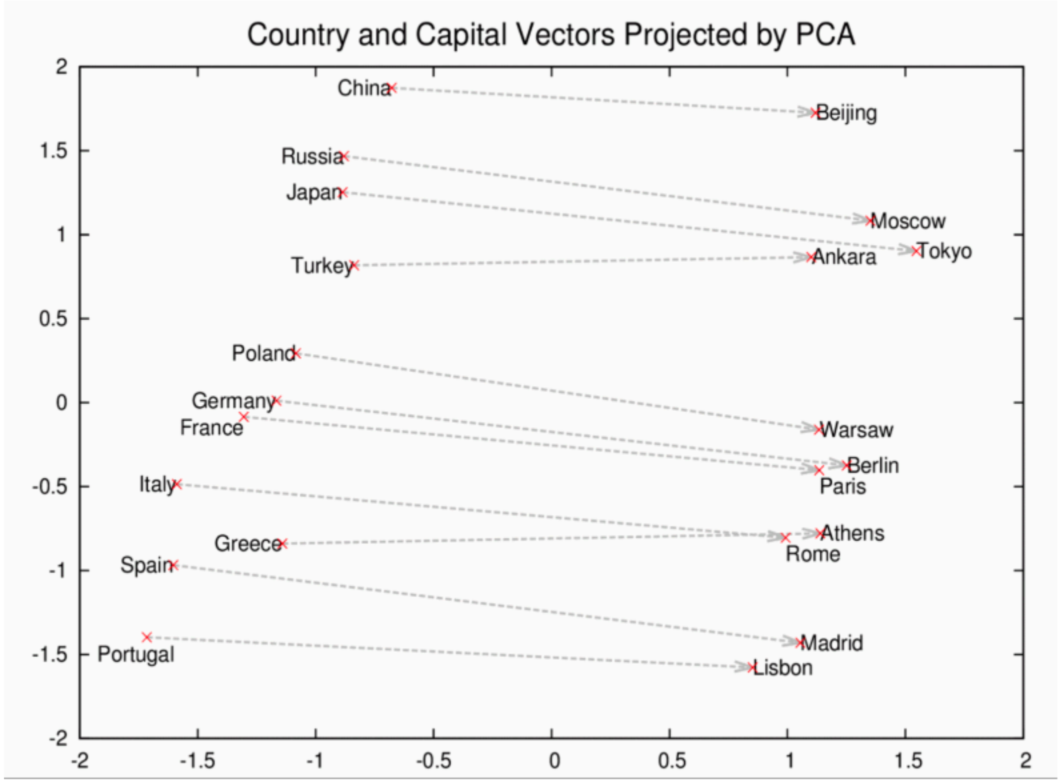
word2vec Skip-gram

This part we keep (embedding lookup table)



`word2vec` examples

- Vectors derived from word2vec are useful, but there is structure to the space, as well
- Semantic relations between words often correspond to translation operations (a consistent direction and distance in vector space)



Mikolov et al., *Distributed Representations of Words and Phrases and their Compositionality*

Analogies using `word2vec`

- Analogies represent consistent semantic relationships; we can rephrase them in terms of vector translation operations

king : man :: queen : woman

$$\overrightarrow{king} - \overrightarrow{man} \approx \overrightarrow{queen} - \overrightarrow{woman}$$

$$\overrightarrow{queen} \approx \overrightarrow{king} - \overrightarrow{man} + \overrightarrow{woman}$$

`man`	`:`	`king`	`::`	`woman`	`:`	`queen`
`China`	`:`	`Beijing`	`::`	`Russia`	`:`	`Moscow`
`knee`	`:`	`leg`	`::`	`elbow`	`:`	`arm`
`building`	`:`	`architect`	`::`	`software`	`:`	`programmer`

Representation learning and fairness/bias

- Learning from data is powerful
  - Eliminates need for manual knowledge engineering
  - Produces models with high accuracy
- But there may be statistical relationships in the data that we prefer the model *not* leverage
  - And it may not be easy to determine when the model is using it

man	:	king	::	woman	: queen
		doctor			nurse
		computer programmer			homemaker
		carpentry			sewing
		chuckle			giggle
		superstar			diva

Bolukbasi et al., *Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings*

Other Neural Word Embeddings

- GloVe (Stanford) – Trained on word-word co-occurrence statistics rather than CBOW/skip-gram
- Fasttext (Facebook AI) – Word vectors derived from vectors for smaller units (character ngrams)
- ELMO (Allen AI) – Word vector representation differs based on context of use
- Embeddings learned through end-to-end task-specific training

Advantages of word embeddings

- More efficient, concise representation → faster code (?)
- "Deeper" language features related to meaning, rather than specific words
- Transfer learning: build representations using large, general-purpose data set; fine-tune model based on smaller task-specific data