CS 481

Artificial Intelligence Language Understanding

March 28, 2023

Announcements / Reminders

- Please follow the Week 20 To Do List instructions
- Programming Assignment #02 is due on ~~Sunday 04/02/23~~ Thursday 04/06/23 11:59 PM CST

- Final Exam date:

  Thursday 04/27/2023 (last week of classes!)

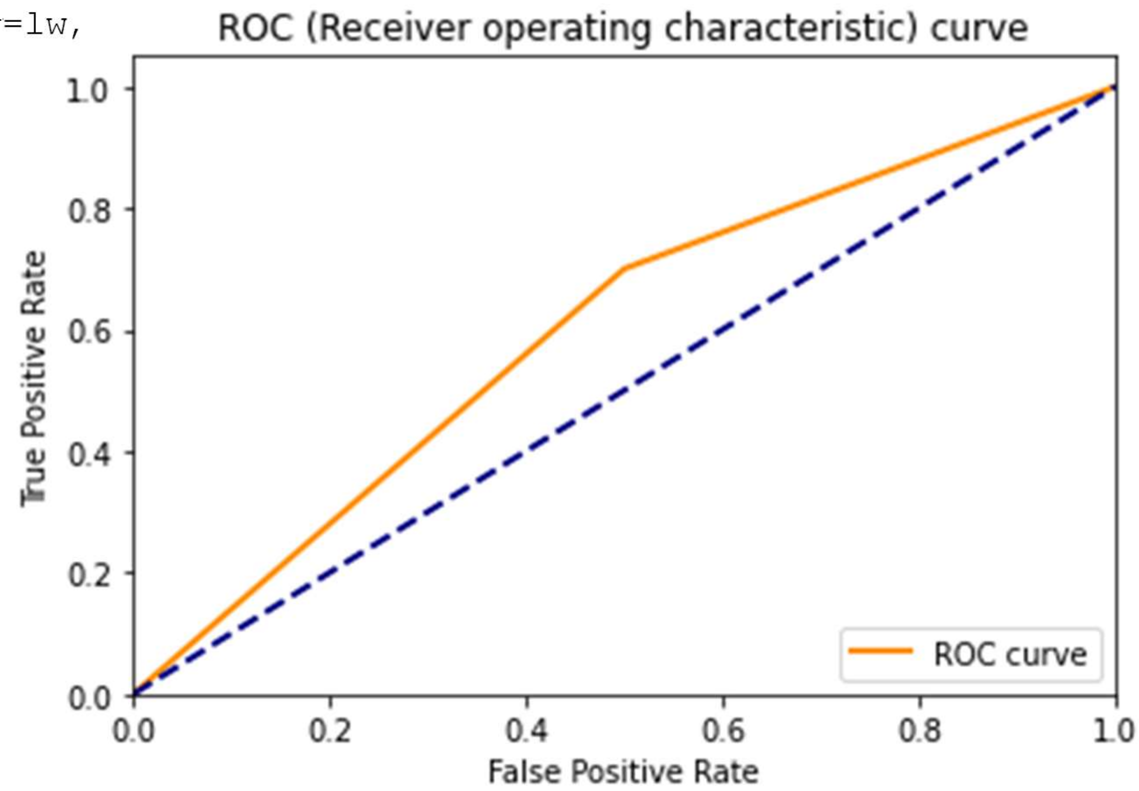
  - Ignore the date provided by the Registrar
  - Section 02 [Online]: contact Mr. Charles Scott (scott@iit.edu) to arrange your exam

Plan for Today

- **Word Embeddings**
- **Word2Vec**

Programming Assignment #02

```
plt.figure()
lw = 2
false_positive_rate = 0.5 # get actual number
from your results
true_positive_rate = 0.7 # get actual number
from your results
plt.plot([0, false_positive_rate, 1], [0,
true_positive_rate, 1], color='darkorange',
lw=lw, label='ROC curve')
plt.plot([0, 1], [0, 1], color='navy', lw=lw,
linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC (Receiver operating
characteristic) curve')
plt.legend(loc="lower right")
plt.show()
```



Computational Models of Meaning

"*a word is characterized by the company it keeps*"

- John Rupert Firth (English linguist)

"*In most cases, the meaning of a word is its use*"

- Ludwig Wittgenstein (Austrian philosopher)

"*If A and B have almost identical environments we say that they are synonyms.*"

- Zellig Harris (American linguist)

Word Embedding: Definition

Word Embedding:

a term used for the ***representation of words*** *for text analysis,* ***typically in the form of a real-valued vector*** *that* ***encodes the meaning of the word such*** *that the* ***words that are closer in the vector space are expected to be similar*** *in meaning*

from Wikipedia

Word Embedding: Key Benefits

- **don't require expensive annotation** → can be derived from large unannotated corpora that are readily available

- **pre-trained embeddings** can be used in to assist NLP tasks that use small amounts of labeled data.

Vector Embeddings: Methods

- `tf-idf`
  - popular in Information Retrieval
  - sparse vectors
  - word represented by (a simple function of) the counts of nearby words

- `Word2vec`
  - dense vectors
  - representation is created by training a classifier to predict whether a word is likely to appear nearby

Sparse vs. Dense Vectors

- **Sparse vectors have a lot of values set to zero.**

  [0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 2]

- **Dense vector: most of the values are non-zero.**
  - **better use of storage**
  - **carries more information**

  [3, 1, 5, 0, 1, 4, 9, 8, 7, 1, 1, 2, 2, 2, 2]

Sparse vs. Dense Vectors

- `tf-idf` **vectors are typically:**
  - **long (length 20,000 to 50,000)**
  - **sparse (most elements are zero)**

- **What if we could learn vectors that are**
  - **short (length 50-1000)**
  - **dense (most elements are non-zero)**

Short / Dense Vectors: Benefits

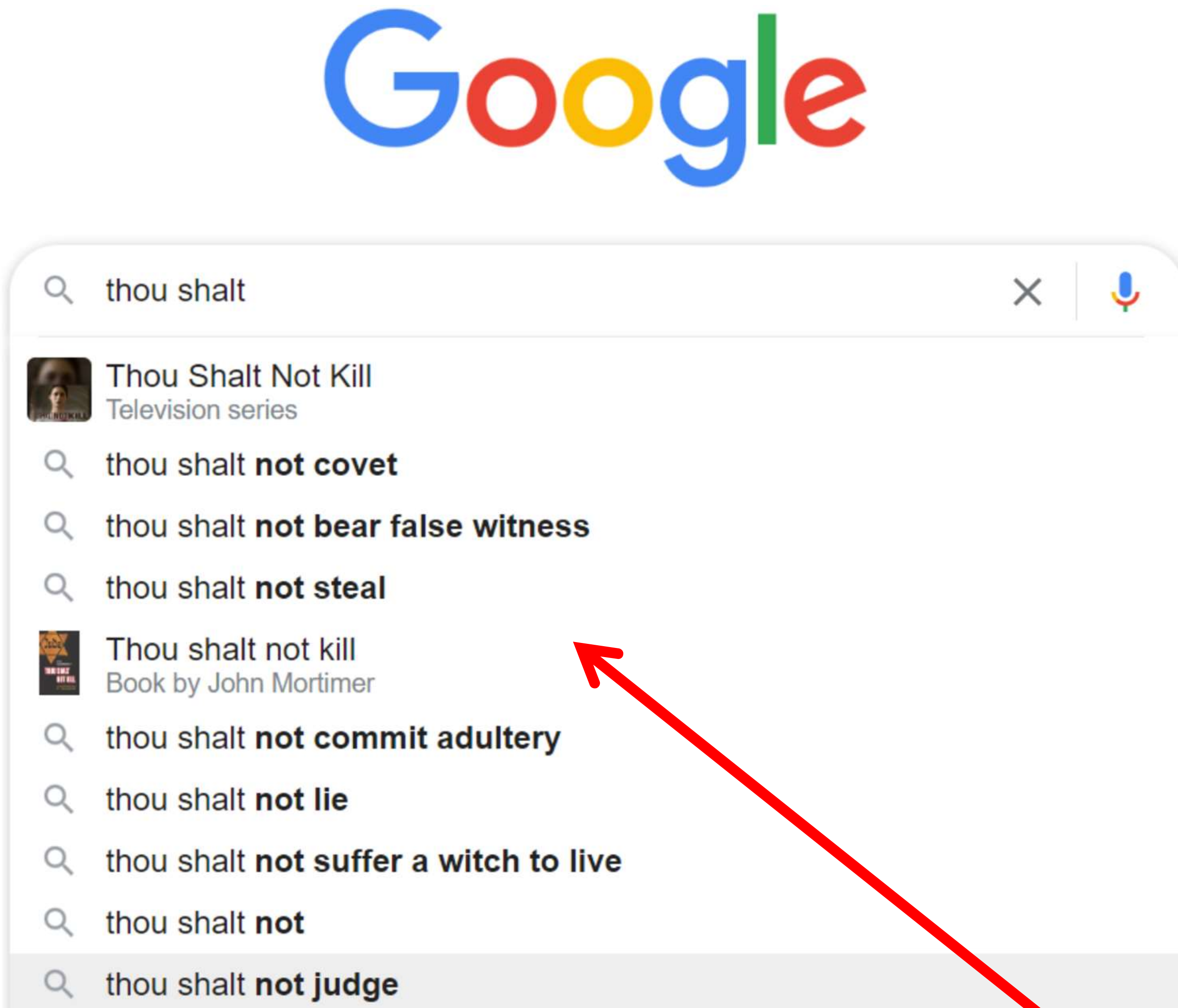
- Why short/dense vectors?
  - short vectors may be easier to use as **features** in machine learning (fewer weights to tune)
  - dense vectors may **generalize** better than explicit counts
  - dense vectors may do better at capturing synonymy:
    - *car* and *automobile* are synonyms; but are distinct dimensions
      - a word with *car* as a neighbor and a word with *automobile* as a neighbor should be similar, but aren't
- **In practice, they work better**

Short/Dense Vectors: Methods

- "Neural Language Model"-inspired models
  - Word2vec, GloVe

- Singular Value Decomposition (SVD)
  - A special case of this is called LSA – Latent Semantic Analysis
- Alternative to these "static embeddings":
  - Contextual Embeddings (ELMo, BERT)
  - Compute distinct embeddings for a word in its context
  - Separate embeddings for each token of a word

Language Models: Application



we want to find predict the "rest" of the query

N-gram Language Models

General Maximum Likelihood Estimation (MLE) of an N-gram:

$$P(w_N \mid w_{N-K+1}, w_{N-K+2}, \ldots, w_{N-1}) = \frac{count(w_{N-K+1}, w_{N-K+2}, \ldots, w_{N-1}, w_N)}{count(w_{N-K+1}, w_{N-K+2}, \ldots, w_{N-1})}$$

where:

w_i **- ith word / token**

In MLE, the resulting parameter set maximizes the likelihood of the training set T given the model M (i.e., P(T | M)).

N-gram Language Models

General Maximum Likelihood Estimation (MLE) of an N-gram:

<------------thou shalt----------->

$$P(w_N \mid w_{N-K+1}, w_{N-K+2}, \ldots, w_{N-1}) = \frac{count(w_{N-K+1}, w_{N-K+2}, \ldots, w_{N-1}, w_N)}{count(w_{N-K+1}, w_{N-K+2}, \ldots, w_{N-1})}$$

"rest" | next query word

where:

w_i - ith word / token

In MLE, the resulting parameter set maximizes the likelihood of the training set T given the model M (i.e., P(T | M)).

N-gram Language Models

General Maximum Likelihood Estimation (MLE) of an N-gram:

<------------thou shalt----------->

$$P(w_N \mid w_{N-K+1}, w_{N-K+2}, \ldots, w_{N-1}) = \frac{count(w_{N-K+1}, w_{N-K+2}, \ldots, w_{N-1}, w_N)}{count(w_{N-K+1}, w_{N-K+2}, \ldots, w_{N-1})}$$

"rest" | next query word

where:

w_i - ith word / token

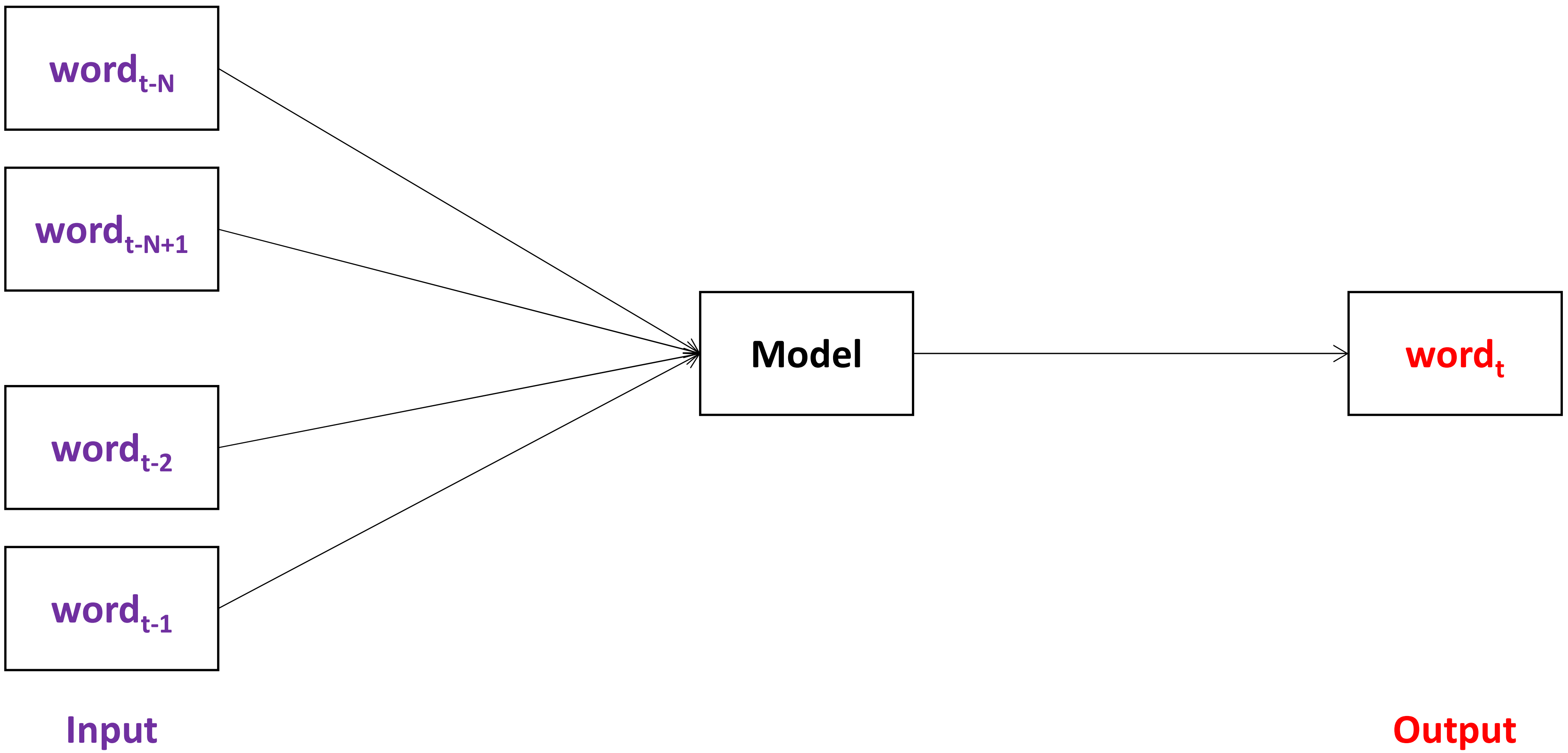
Looks at PAST words to predict the NEXT word! (highest P() NEXT word)

In MLE, the resulting parameter set maximizes the likelihood of the training set T given the model M (i.e., P(T | M)).

N-gram Language Models: Prediction

Given:

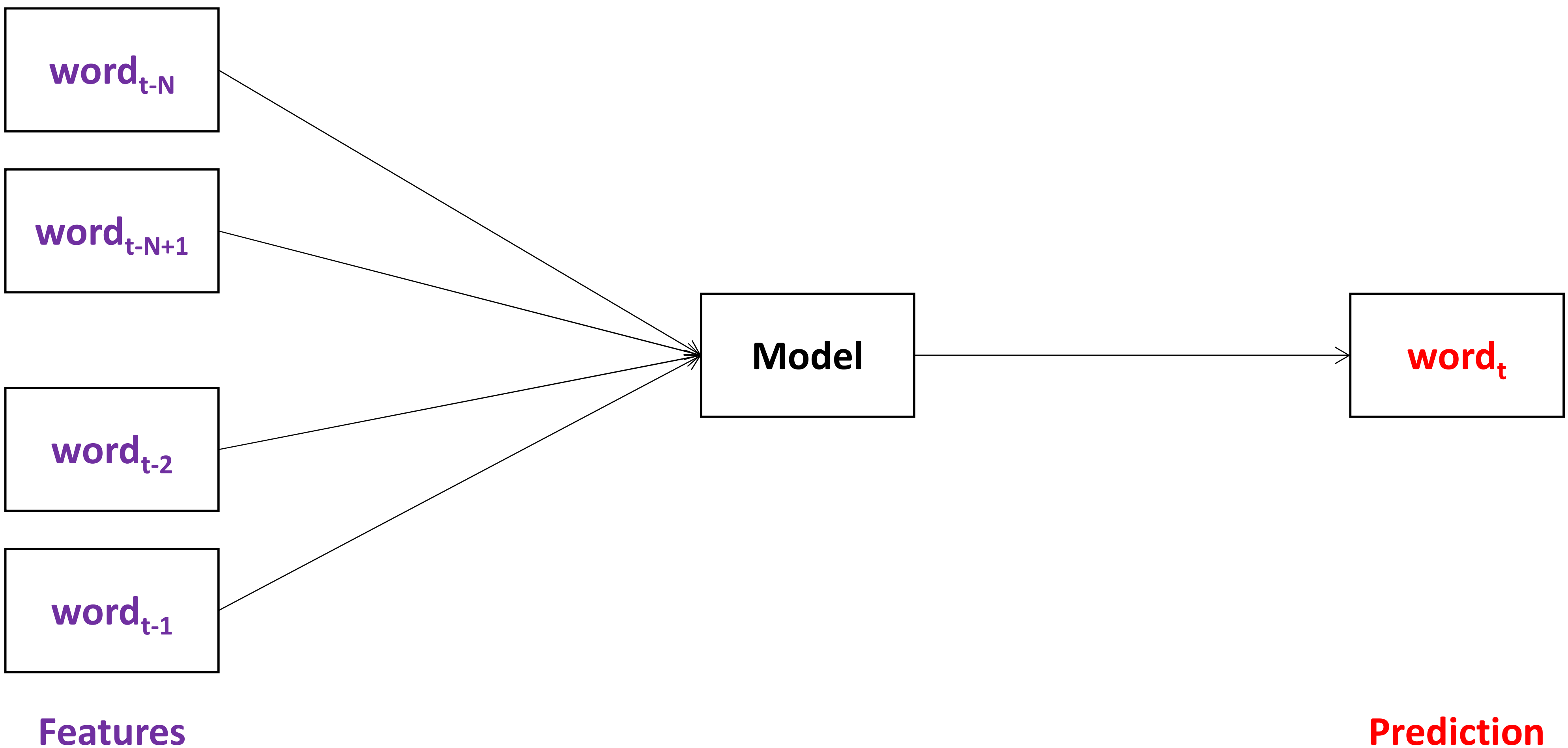
S: word$_{t-N}$ ... word$_{t-2}$ word$_{t-1}$ ______



N-gram Language Models: Prediction

Given:

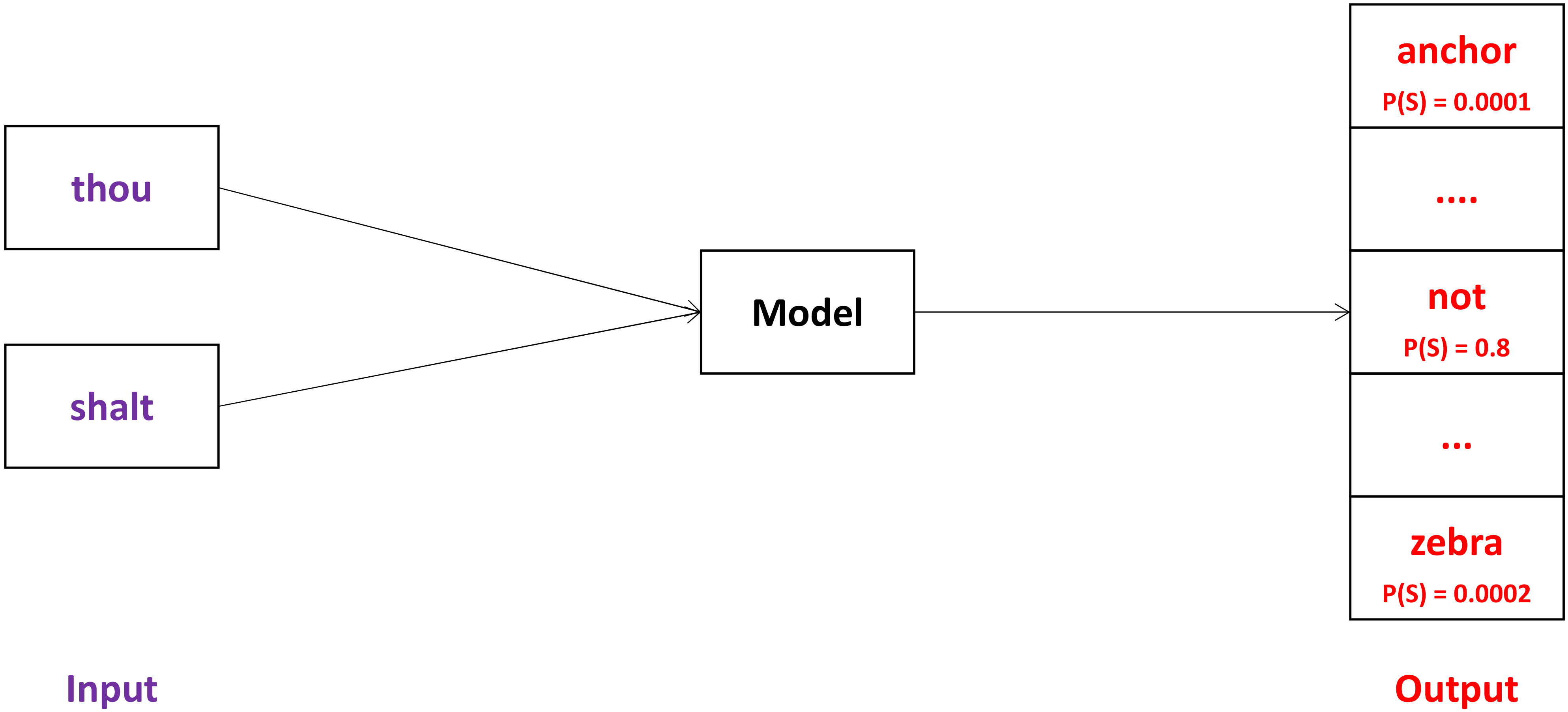
S: $word_{t-N}$ … $word_{t-2}$ $word_{t-1}$ _____



N-gram Language Models: Prediction

Given (N = 2):

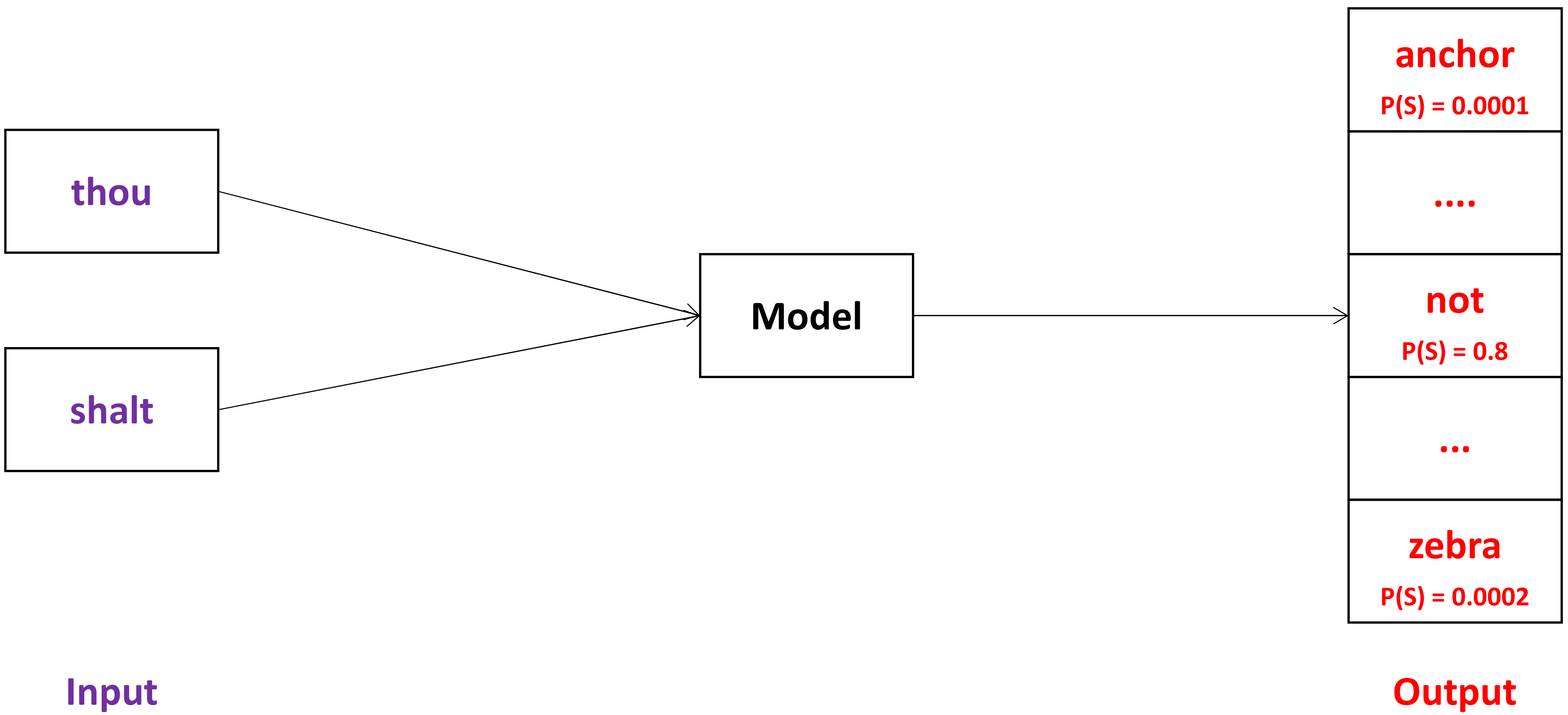
S: thou shalt _____



N-gram Language Models: Prediction

Given (N = 2):

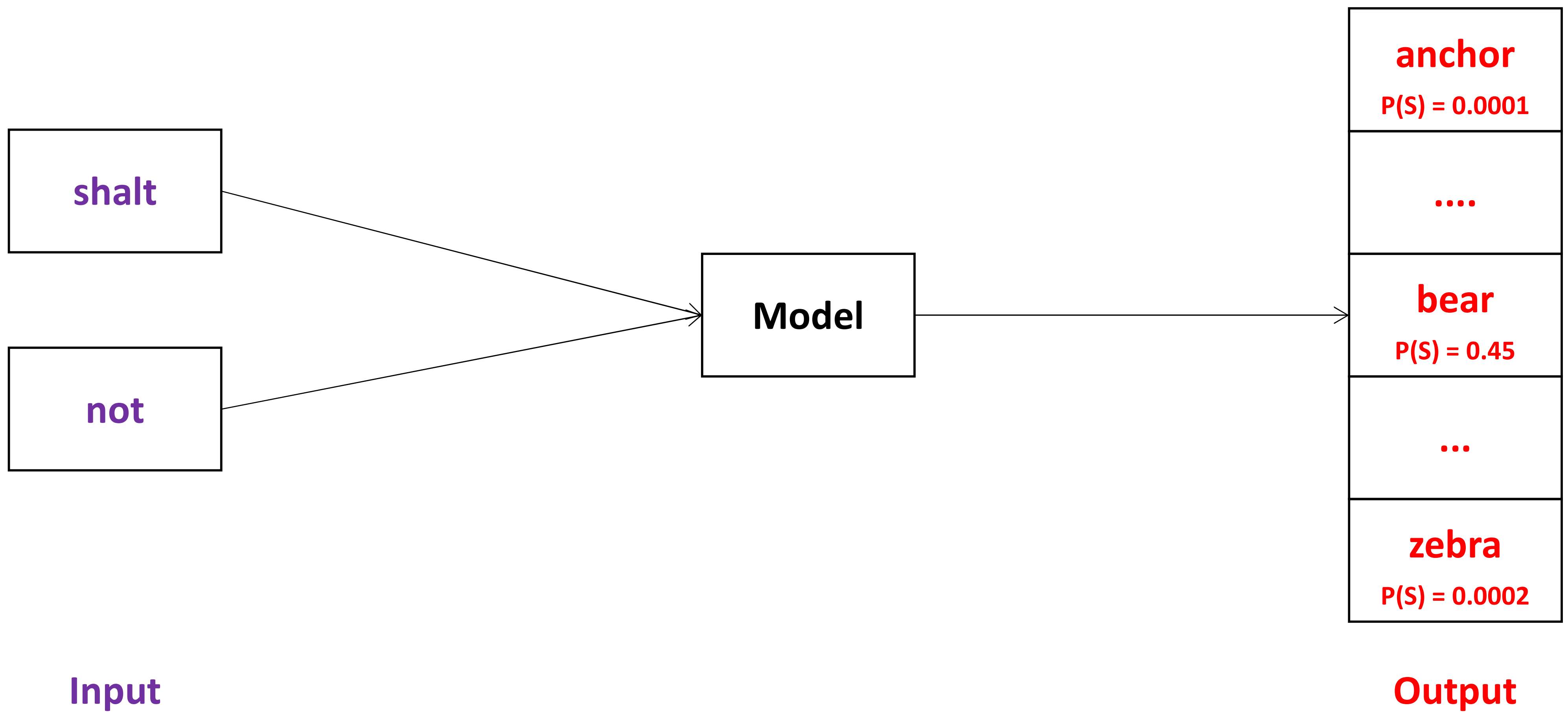
S: thou shalt _____



N-gram Language Models: Prediction

Given (N = 2):

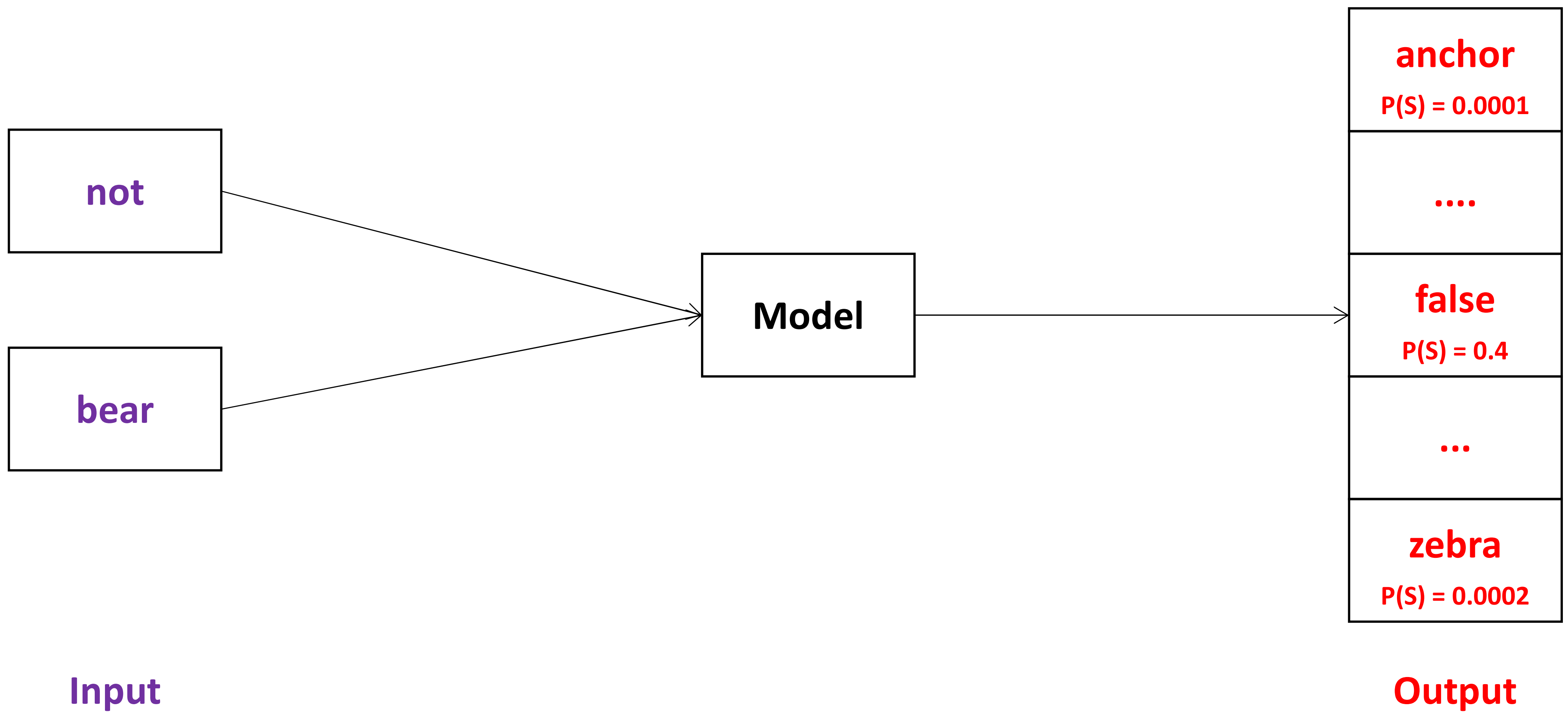
S: thou shalt not _____



N-gram Language Models: Prediction

Given (N = 2):

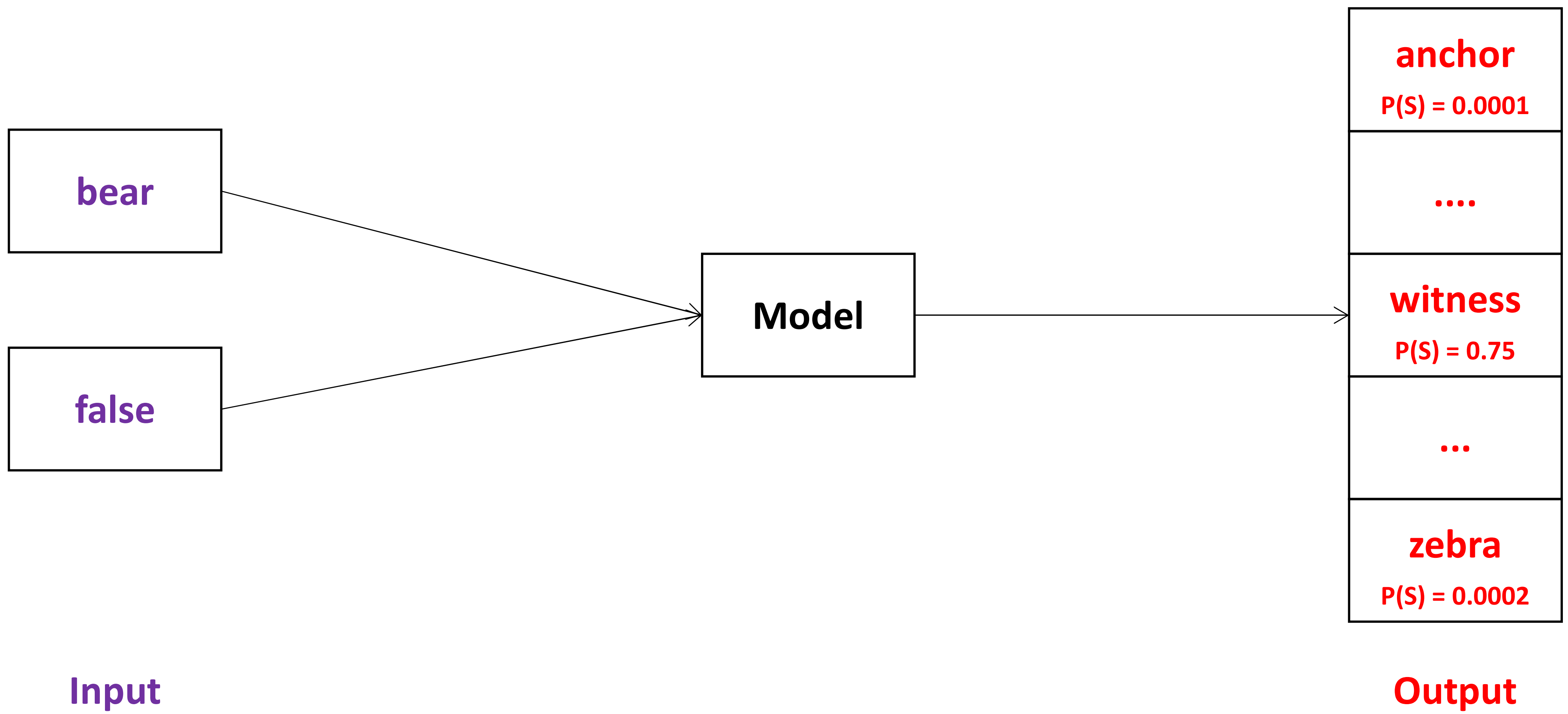
S: thou shalt not bear _____



N-gram Language Models: Prediction

Given (N = 2):

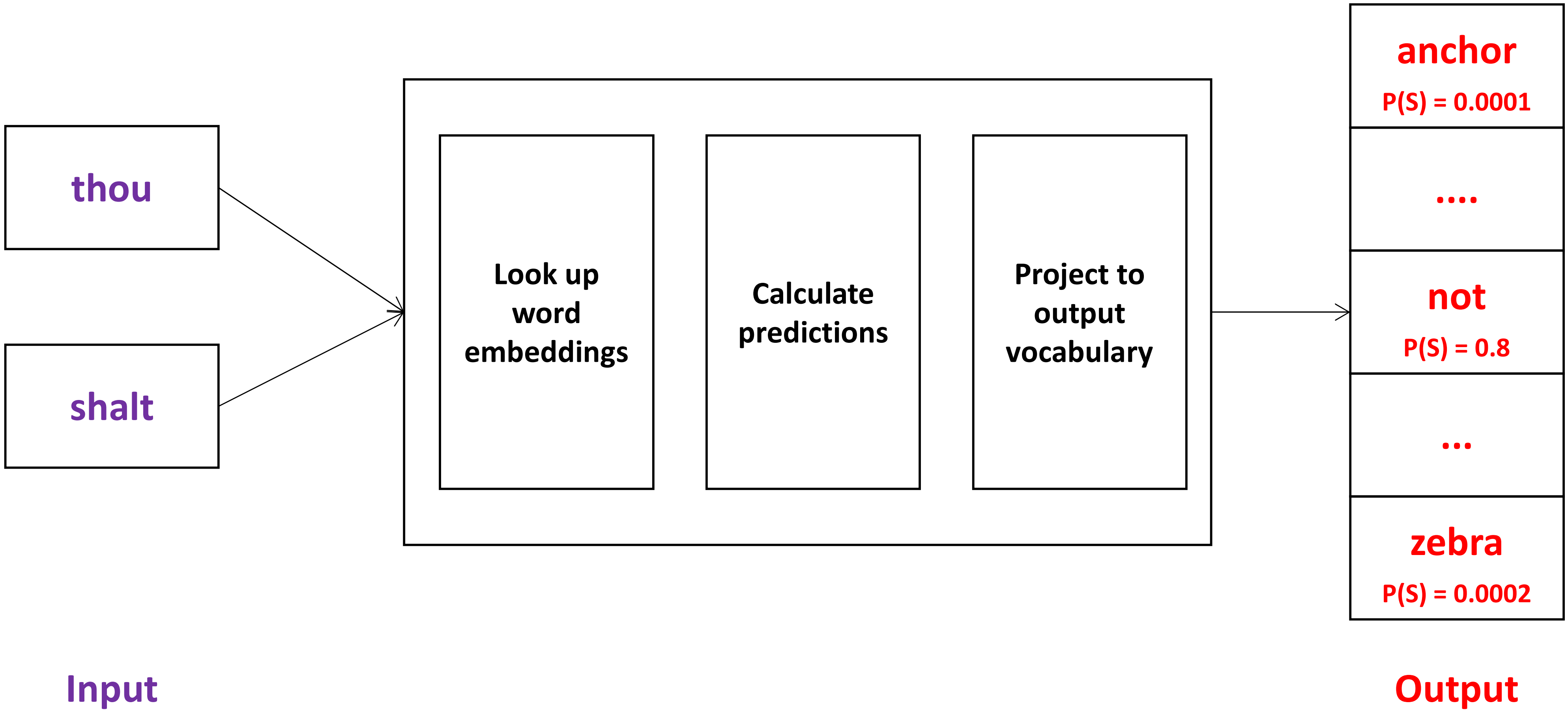
S: thou shalt not bear false _____



Trained Language Models: Prediction

Given input and a model (word embeddings):

S: thou shalt _____



N-gram Language Models

N-gram language model will handle cases such as:

Tomorrow is _______

but <u>not</u>:

Tomorrow is _______ to be

where:

- **context words**
- **a word to be predicted: target word**

N-gram Language Models

N-gram language model will handle cases such as:

Tomorrow is _______

but <u>not</u>:

target word

Tomorrow is _______ to be

context words

Predicting the Missing Word

Say, we want to predict the missing word _____ in:

Tomorrow is _______ to be

How would you go about it?

Predicting the Missing Word

Say, we want to predict the missing word _____ in:

Tomorrow is _______ to be

Two approaches possible:

- **use the context words to predict the target word**
- **use the target word to predict context words**

Sliding Window

Say, we want to predict the missing word _____ in:

Tomorrow is _______ to be

Let's generalize it a bit:

$\mathbf{word_{t-2}\ word_{t-1}\ word_{t}\ word_{t+1}\ word_{t+2}}$

Sliding Window

Say, we want to predict the missing word _____ in:

Tomorrow is _______ to be

Let's generalize it even further:

word_{t-N} ... word_{t-2} word_{t-1} word_{t} word_{t+1} word_{t+2} ... word_{t+N}

Sliding Window

Say, we want to predict the missing word _____ in:

Tomorrow is _______ to be

But we don't need to look at ALL words in:

word_{t-N} … word_{t-2} word_{t-1} word_{t} word_{t+1} word_{t+2} … word_{t+N}

We can reduce the size of the context:

word_{t-N} … $\boxed{\text{word}_{t-2}\ \text{word}_{t-1}\ \text{word}_{t}\ \text{word}_{t+1}\ \text{word}_{t+2}}$ … word_{t+N}

sliding window +/- 2

Predicting the Missing Word

Say, we want to predict the missing word _____ in:

Tomorrow is _______ to be

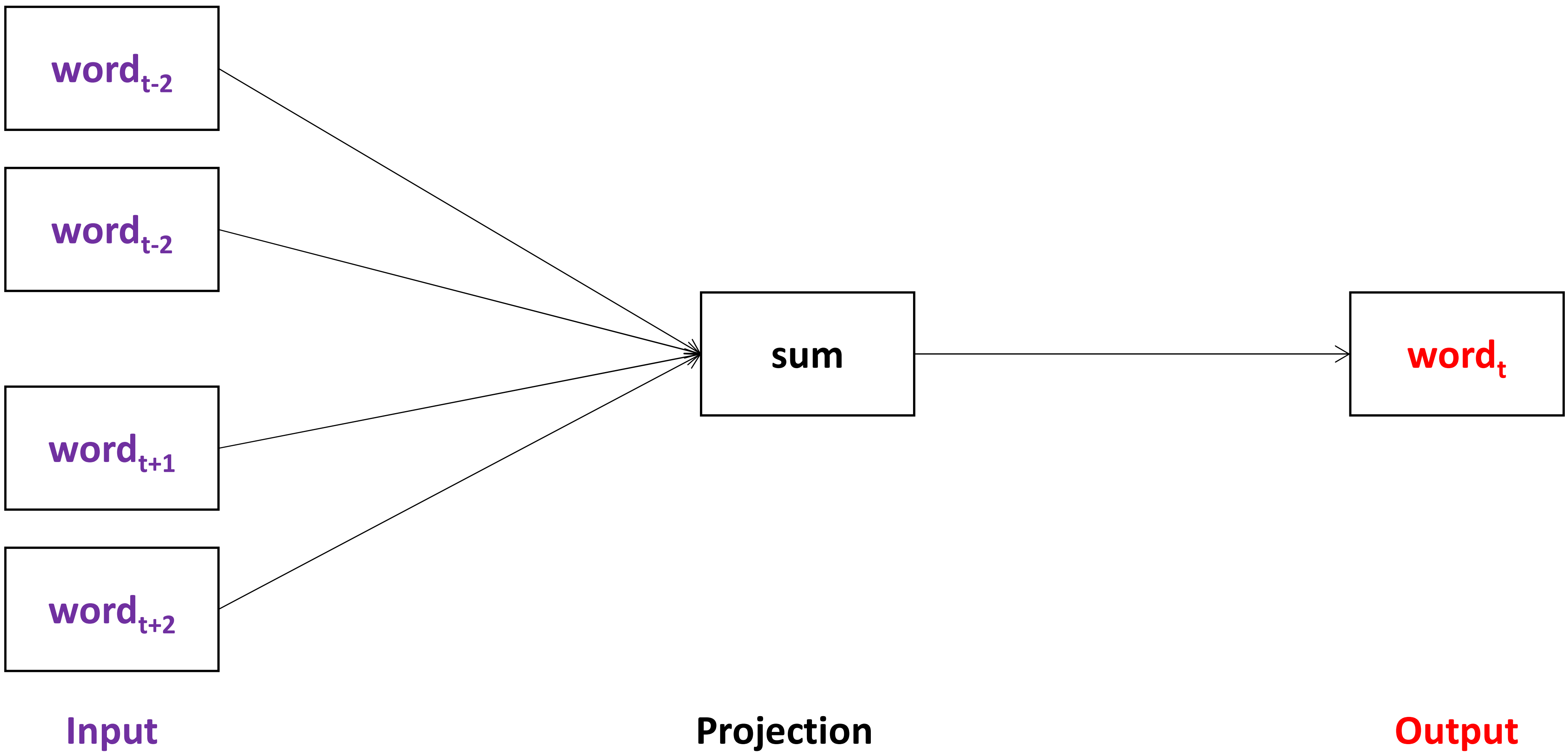
Two approaches possible:

- **use the context words to predict the target word: Continuous Bag of Words model (CBOW)**
- **use the target word to predict context words: Skip Gram model**

CBOW Word2Vec

Given (window size 2):

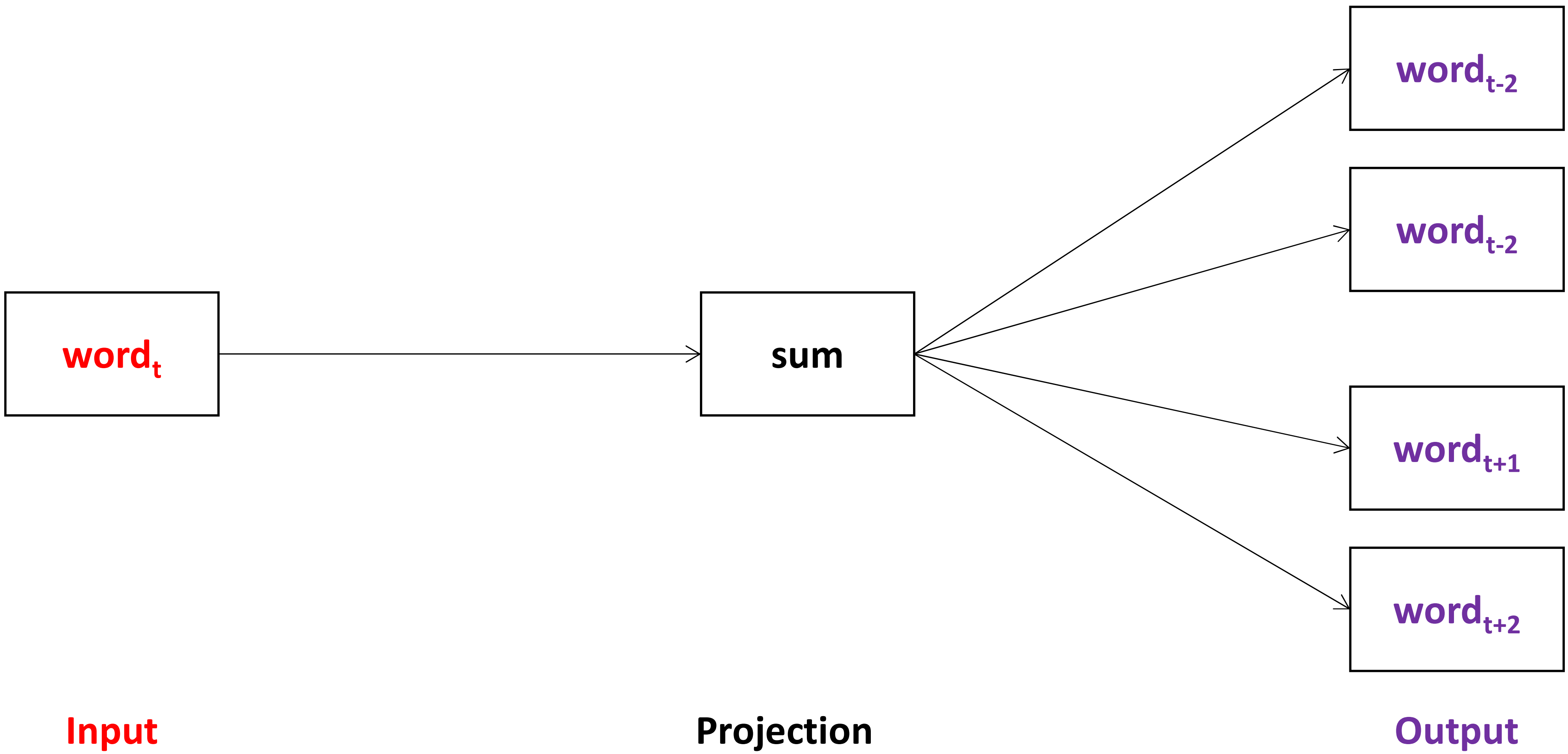
$word_{t-N}$ … $word_{t-2}$ $word_{t-1}$ ______ $word_{t+1}$ $word_{t+2}$ … $word_{t+N}$



Skip Gram Word2Vec

Given (window size 2):

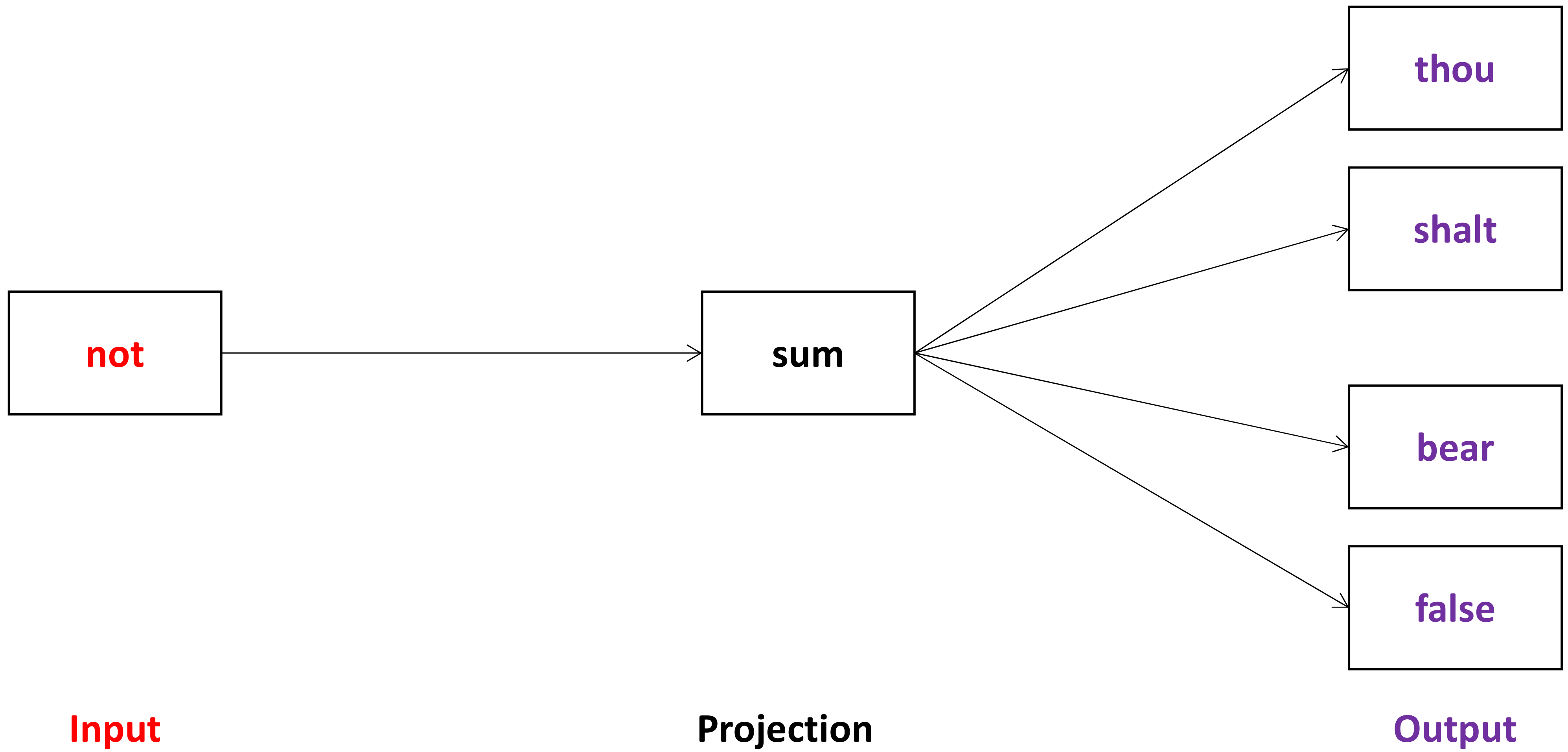
word_{t-N} … word_{t-2} word_{t-1} _____ word_{t+1} word_{t+2} … word_{t+N}



Skip Gram Word2Vec

Predict context given target word:

thou shalt _not_ bear false witness



Word2Vec: Idea

DON’T count - Predict!

Word2Vec: Idea

- Instead of **counting** how often each word *w* occurs near "*apricot*"
  - Train a classifier on a binary **prediction** task:
    - Is *w* likely to show up near "*apricot*"?
- We don't actually care about this task
  - but we'll take the learned classifier weights as the word embeddings
- Use **self-supervision**:
  - A word c that occurs near "*apricot*" in the corpus acts as the gold "correct answer" for supervised learning
  - **No need for human labels**

Available Tools

- Word2vec (Mikolov et al)

https://code.google.com/archive/p/word2vec/

- GloVe (Pennington, Socher, Manning)

http://nlp.stanford.edu/projects/glove/

Word2Vec: the Approach

1. Treat the target word *t* and a neighboring context word *c* as **positive examples**.
2. Randomly sample other words in the lexicon to get **negative examples**
3. Use **logistic regression** to train a classifier to distinguish those two cases
4. Use the **learned classifier weights** as the **embeddings**

Word2Vec: the Approach

Given the set of **positive** and **negative** training instances, and an **initial set of embedding vectors**

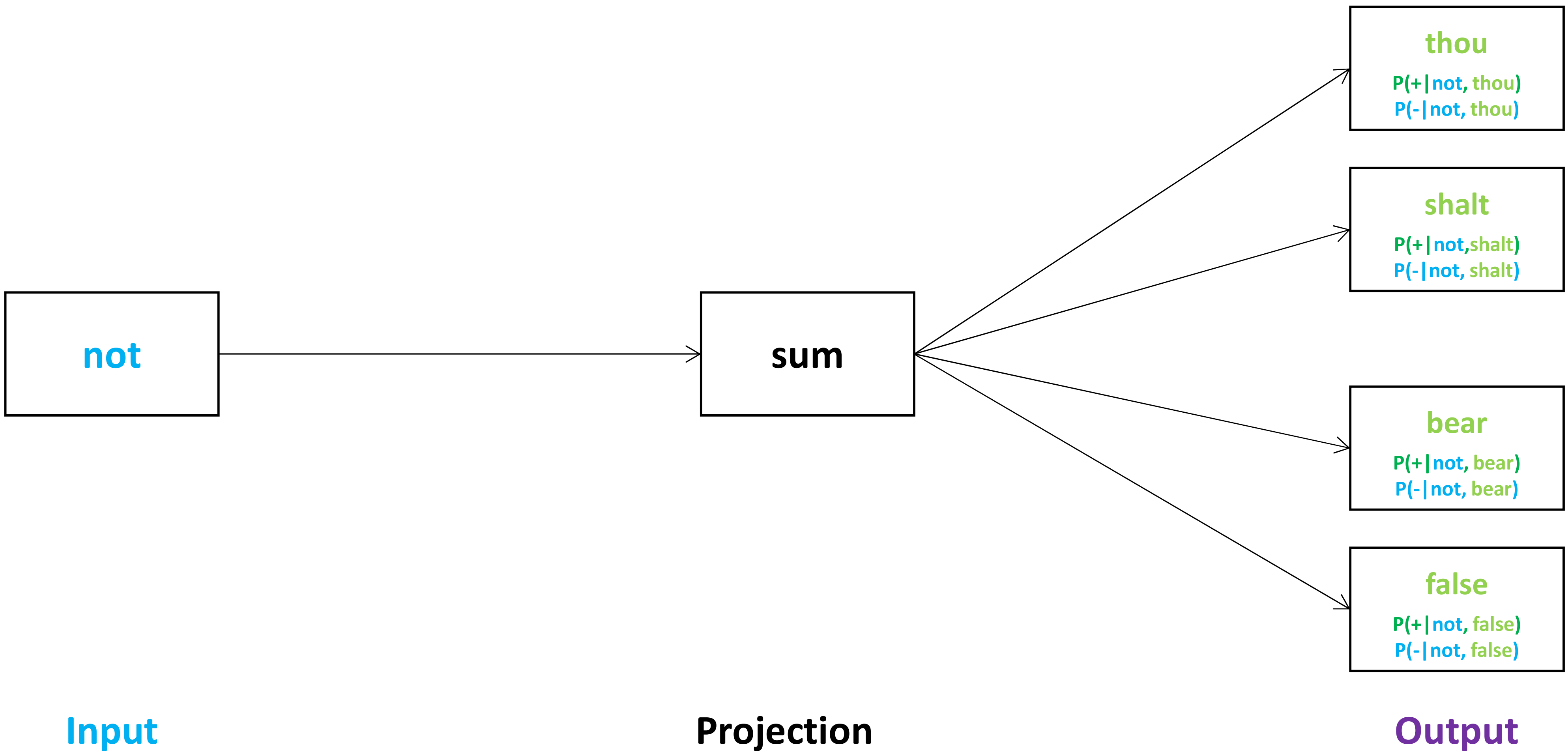
The goal of learning is to adjust those word vectors such that we:

- **maximize** the similarity of the target word, context word pairs (w, c_{pos}) drawn from the **positive** data
- **minimize** the similarity of the (w, c_{neg}) pairs drawn from the **negative** data.

Skip Gram Word2Vec

Predict context given target word:

thou shalt _not_ bear false witness



Skip Gram Classifier

Assume a +/- 2 (L = 4) word window, given training sentence:

target word

…lemon, a [tablespoon of apricot jam, a] pinch…

context words

Skip Gram Classifier

Assume a +/- 2 (L = 4) word window, given training sentence:

w

…lemon, a [tablespoon of apricot jam, a] pinch…

c_1 c_2 c_3 c_4

Skip Gram Classifier

Assume a +/- 2 (L = 4) word window, given training sentence:

w

…lemon, a [**tablespoon of** **apricot** **jam**, **a**] pinch…

c_1 c_2 c_3 c_4

Goal 1: **train a classifier** that is given a candidate (**word**, **context word**) pair: (**apricot**, **jam**), (**apricot**, **aardvark**), etc.

Skip Gram Classifier

Assume a +/- 2 (L = 4) word window, given training sentence:

w

…lemon, a [tablespoon of apricot jam, a] pinch…

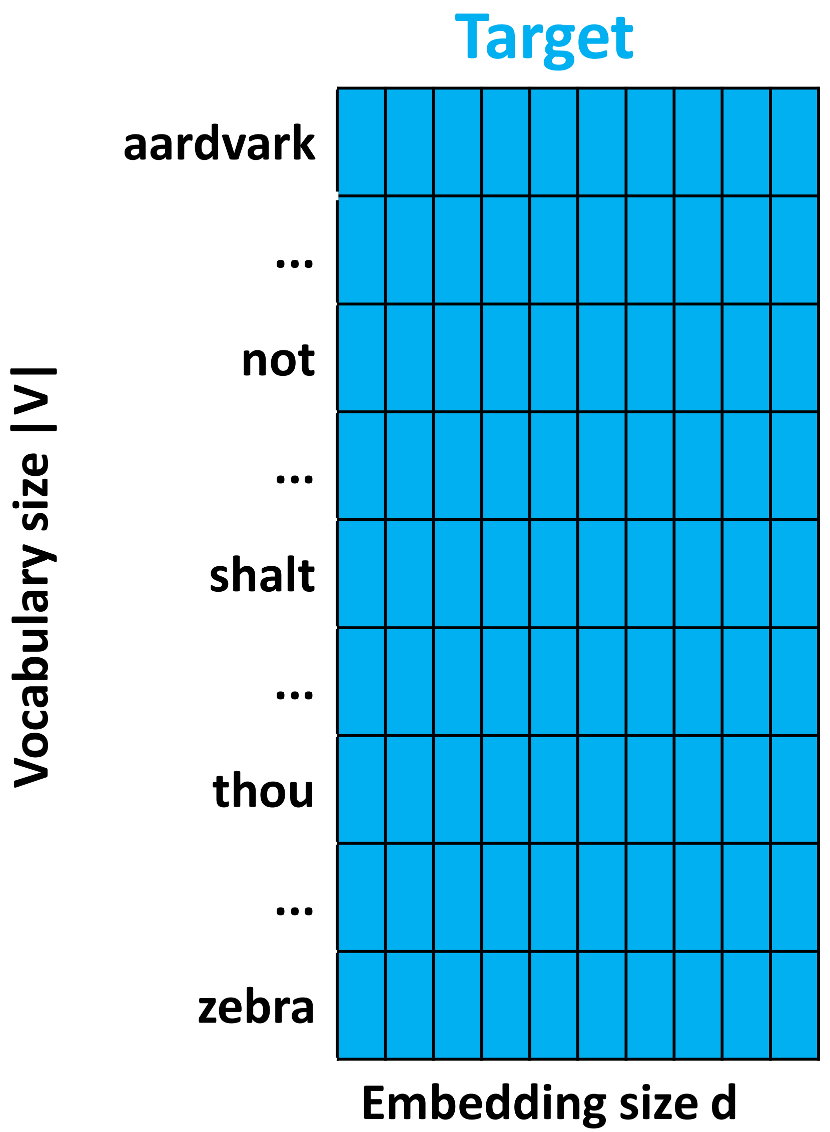
c_1 c_2 c_3 c_4

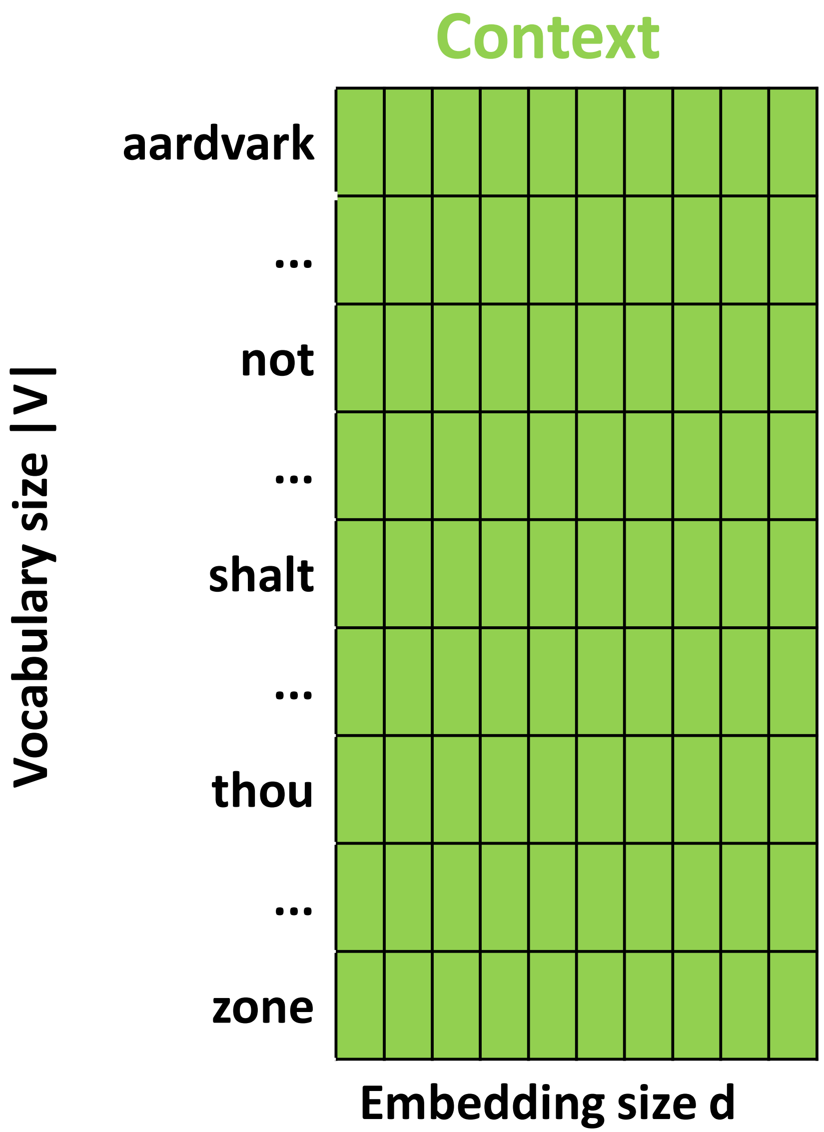
Goal 2: **assign probabilities** to every (**word**, **context word**) pair:

$$P(+ \mid w, c_i) \text{ and}$$

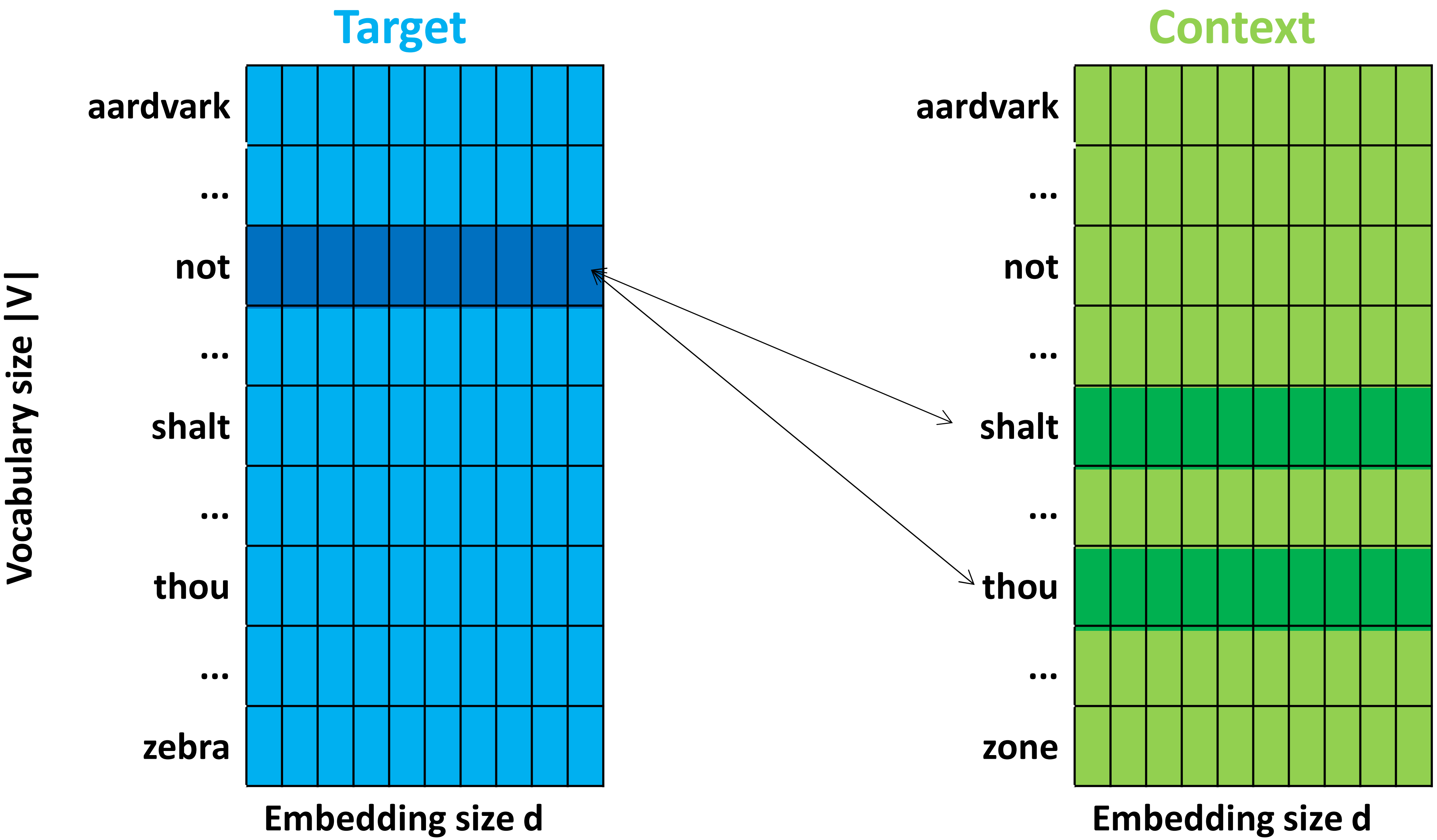
$$P(- \mid w, c_i) = 1 - P(+ \mid w, c_i)$$

Target and Context Embeddings

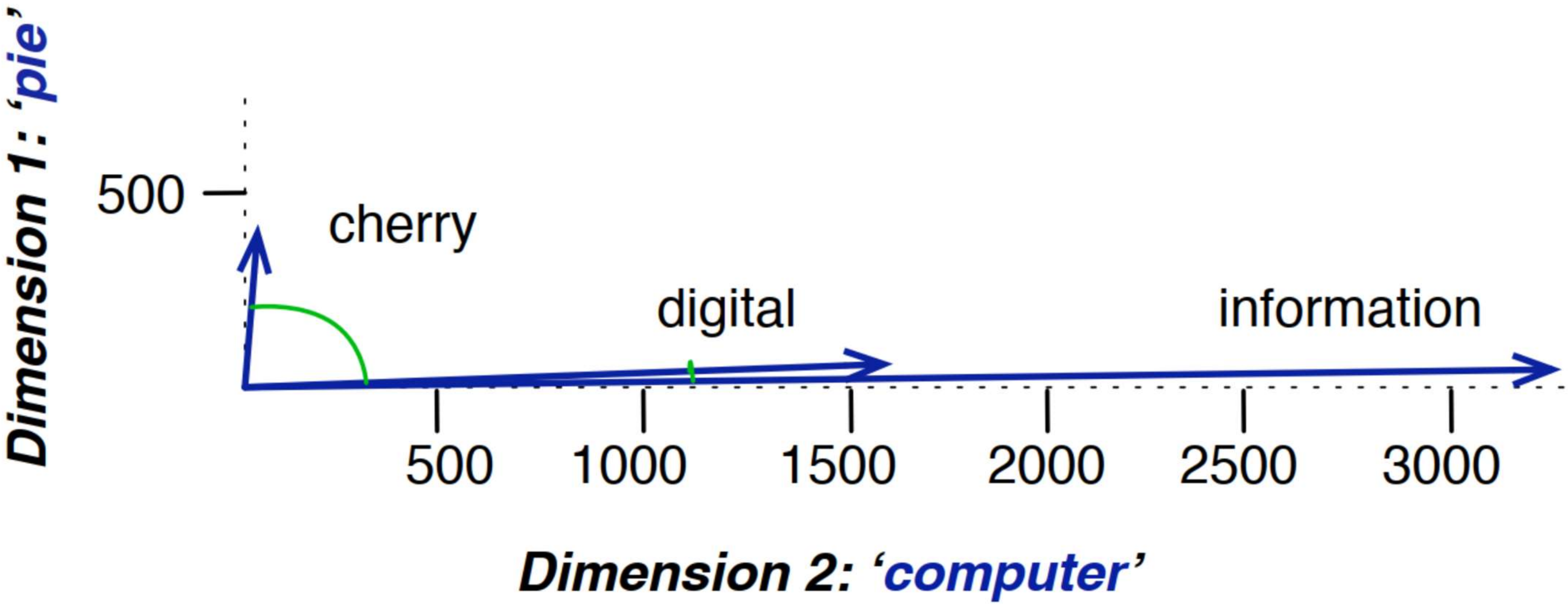




Intuition: Target & Context Similar

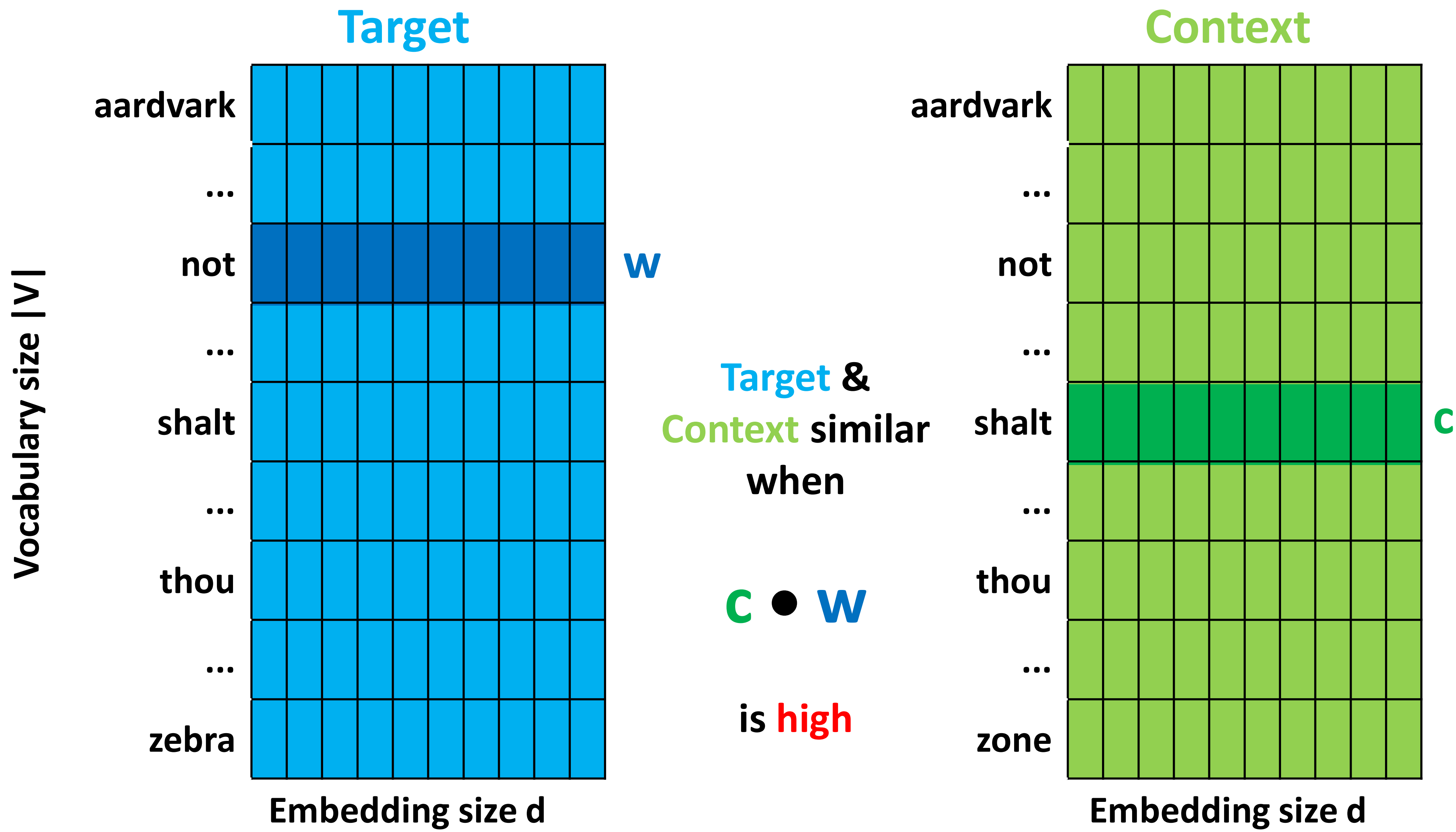


Cosine Similarity Visualization



Two vectors are similar if they have a high dot product | cosine similarity

Intuition: Target & Context Similar

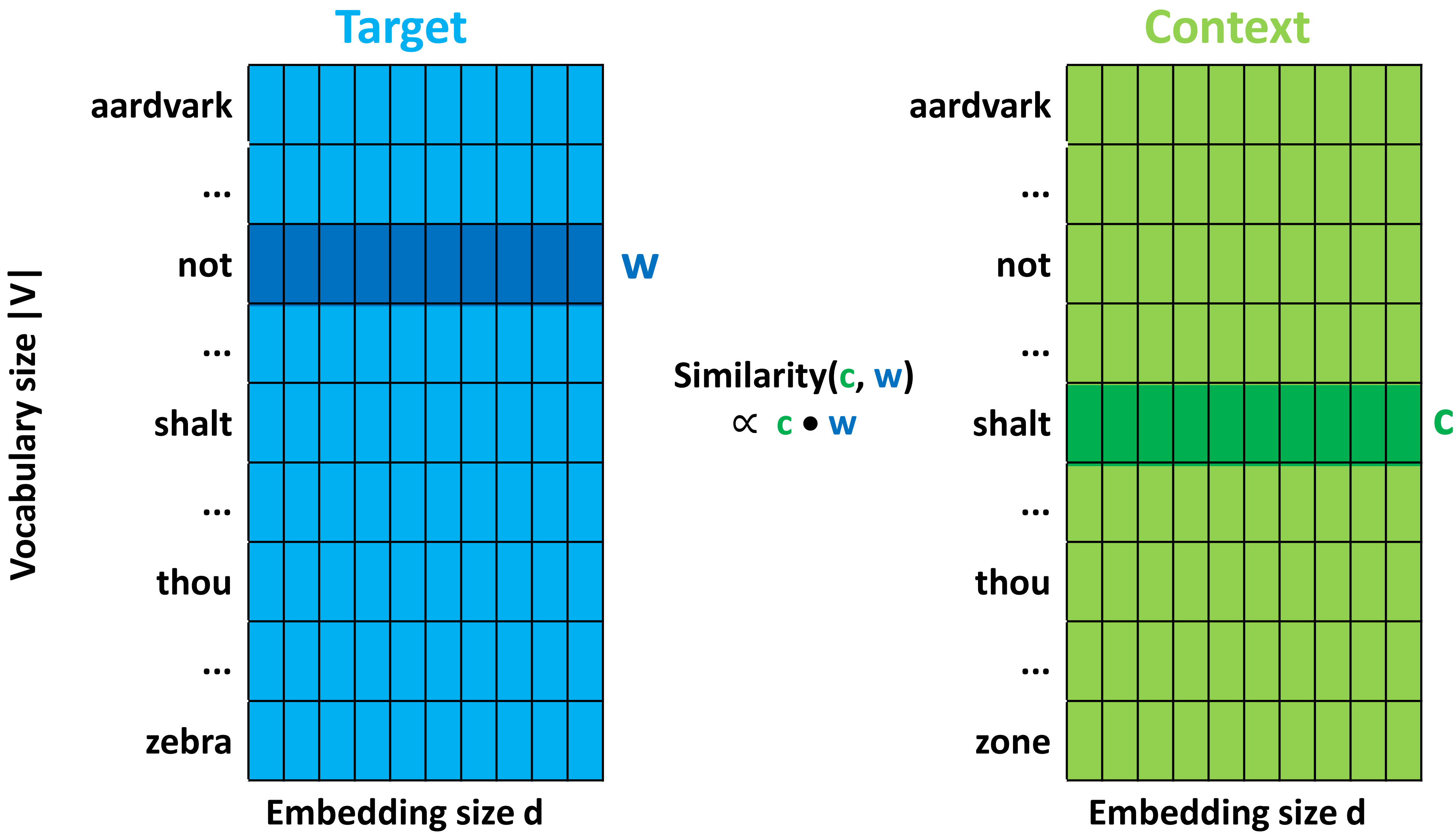


Target & Context similar when

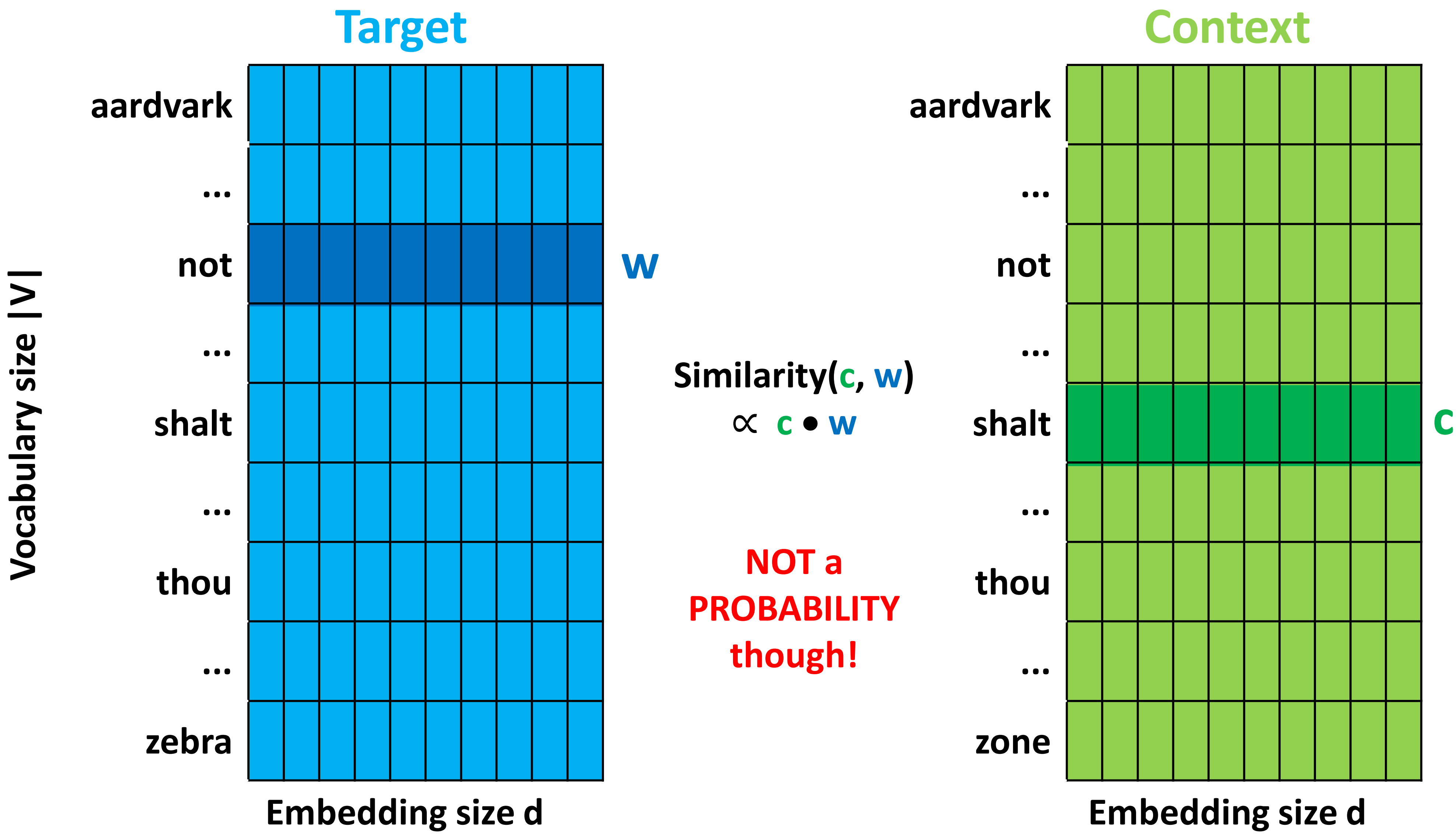
c • w

is high

Intuition: Target & Context Similar



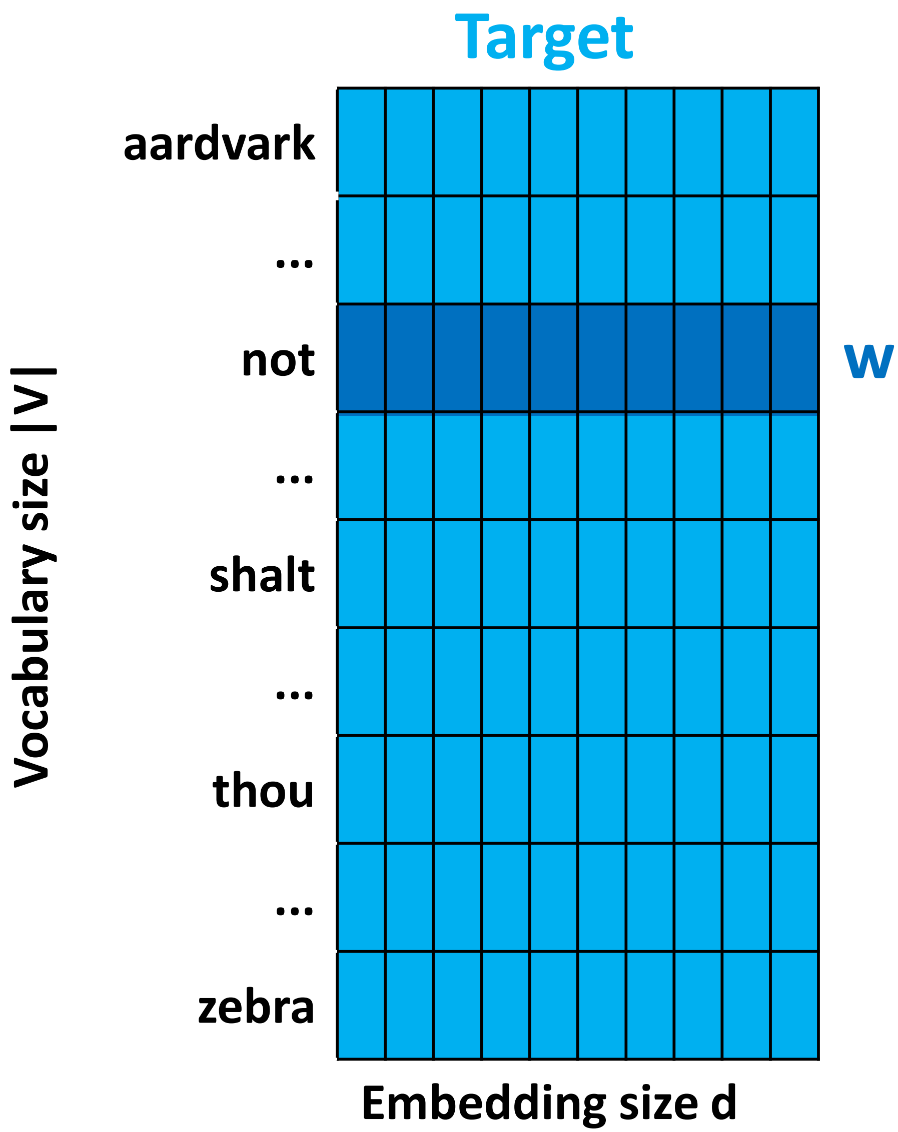
Intuition: Target & Context Similar



Similarity(c, w)
$\propto$ c • w

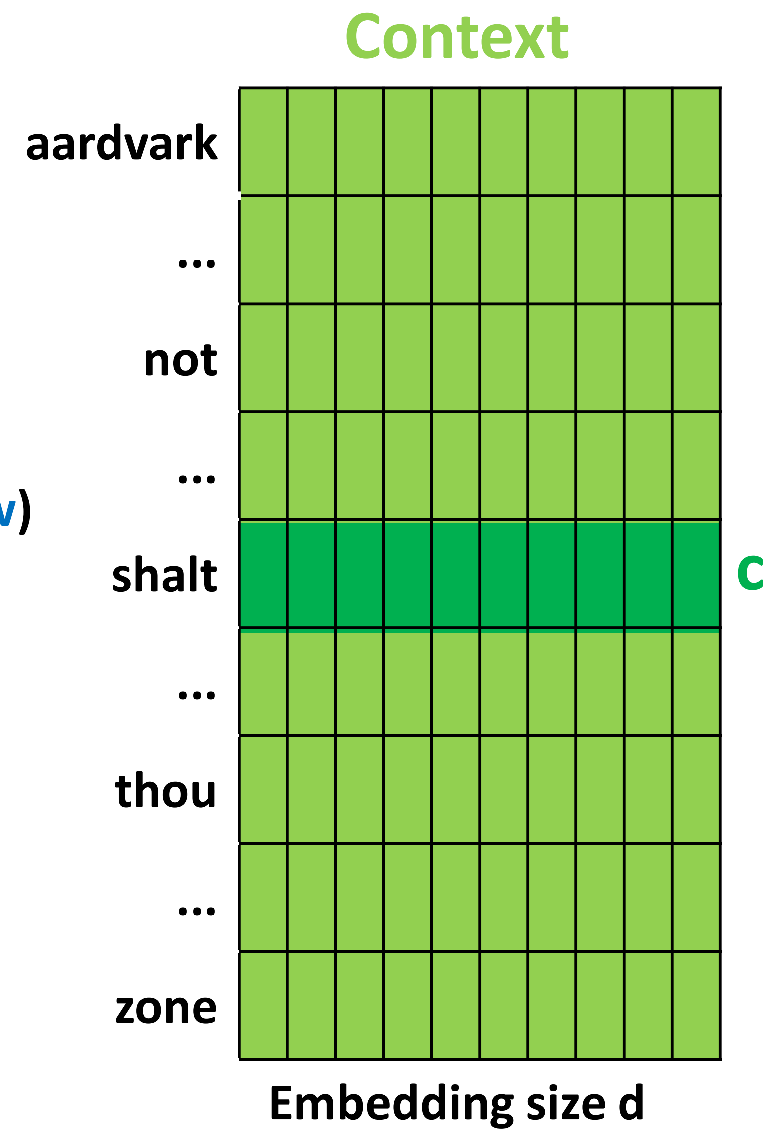
NOT a PROBABILITY though!

Intuition: Target & Context Similar



Similarity(c, w)
$\propto$ c • w

Use sigmoid function!



Similarity → Probability

$$P(+ \mid \mathbf{w}, \mathbf{c}) = \sigma(\mathbf{c} \bullet \mathbf{w}) = \frac{1}{1+exp(-\mathbf{c} \bullet \mathbf{w})}$$

$$P(- \mid \mathbf{w}, \mathbf{c}) = 1 - P(+ \mid \mathbf{w}, \mathbf{c}) =$$

$$= \sigma(-\mathbf{c} \bullet \mathbf{w}) = \frac{1}{1+exp(\mathbf{c} \bullet \mathbf{w})}$$

Skip Gram Classifier

Assume a +/- 2 (L = 4) word window, given training sentence:

w

…lemon, a [tablespoon of apricot jam, a] pinch…

c_1 c_2 c_3 c_4

$P(+ \mid w, c_1)$ $P(+ \mid w, c_2)$ $P(+ \mid w, c_3)$ $P(+ \mid w, c_4)$

$P(- \mid w, c_1)$ $P(- \mid w, c_2)$ $P(- \mid w, c_3)$ $P(- \mid w, c_4)$

OK, but we have lots of possible context words!

Skip Gram Classifier

Assume a +/- 2 (L = 4) word window, given training sentence:

w

…lemon, a [tablespoon of apricot jam, a] pinch…

c_1 c_2 c_3 c_4

$P(+ \mid \mathbf{w}, \mathbf{c}_1)$ $P(+ \mid \mathbf{w}, \mathbf{c}_2)$ $P(+ \mid \mathbf{w}, \mathbf{c}_3)$ $P(+ \mid \mathbf{w}, \mathbf{c}_4)$

$P(- \mid \mathbf{w}, \mathbf{c}_1)$ $P(- \mid \mathbf{w}, \mathbf{c}_2)$ $P(- \mid \mathbf{w}, \mathbf{c}_3)$ $P(- \mid \mathbf{w}, \mathbf{c}_4)$

Assuming word independence, calculate:

$$P(+ \mid \mathbf{w}, \mathbf{c}_1, \mathbf{c}_2, \mathbf{c}_3, \mathbf{c}_4) = \prod_{i=1}^{4} \sigma(\mathbf{c}_\mathrm{i} \bullet w)$$

Skip Gram Classifier

Assume a +/- 2 (L = 4) word window, given training sentence:

w

…lemon, a [tablespoon of apricot jam, a] pinch…

c_1 c_2 c_3 c_4

P(+ | w, c_1)	P(+ | w, c_2)	P(+ | w, c_3)	P(+ | w, c_4)
P(- | w, c_1)	P(- | w, c_2)	P(- | w, c_3)	P(- | w, c_4)

In general:

$$P(+ \mid \mathbf{w}, \mathbf{c}_{1:L}) = \prod_{i=1}^{L} \sigma(\mathbf{c}_i \bullet w)$$

Skip Gram Classifier

Assume a +/- 2 (**L = 4**) word window, given training sentence:

w

…lemon, a [**tablespoon of apricot jam**, **a**] pinch…

c_1 c_2 c_3 c_4

$P(+ \mid \mathbf{w}, c_1)$ $P(+ \mid \mathbf{w}, c_2)$ $P(+ \mid \mathbf{w}, c_3)$ $P(+ \mid \mathbf{w}, c_4)$

$P(- \mid \mathbf{w}, c_1)$ $P(- \mid \mathbf{w}, c_2)$ $P(- \mid \mathbf{w}, c_3)$ $P(- \mid \mathbf{w}, c_4)$

In general [with sums instead of products]:

$$\log P(+ \mid \mathbf{w}, \mathbf{c}_{1:L}) = \sum_{i=1}^{L} log\, \sigma(\mathbf{c}_i \bullet w)$$

Skip Gram Classifier: Summary

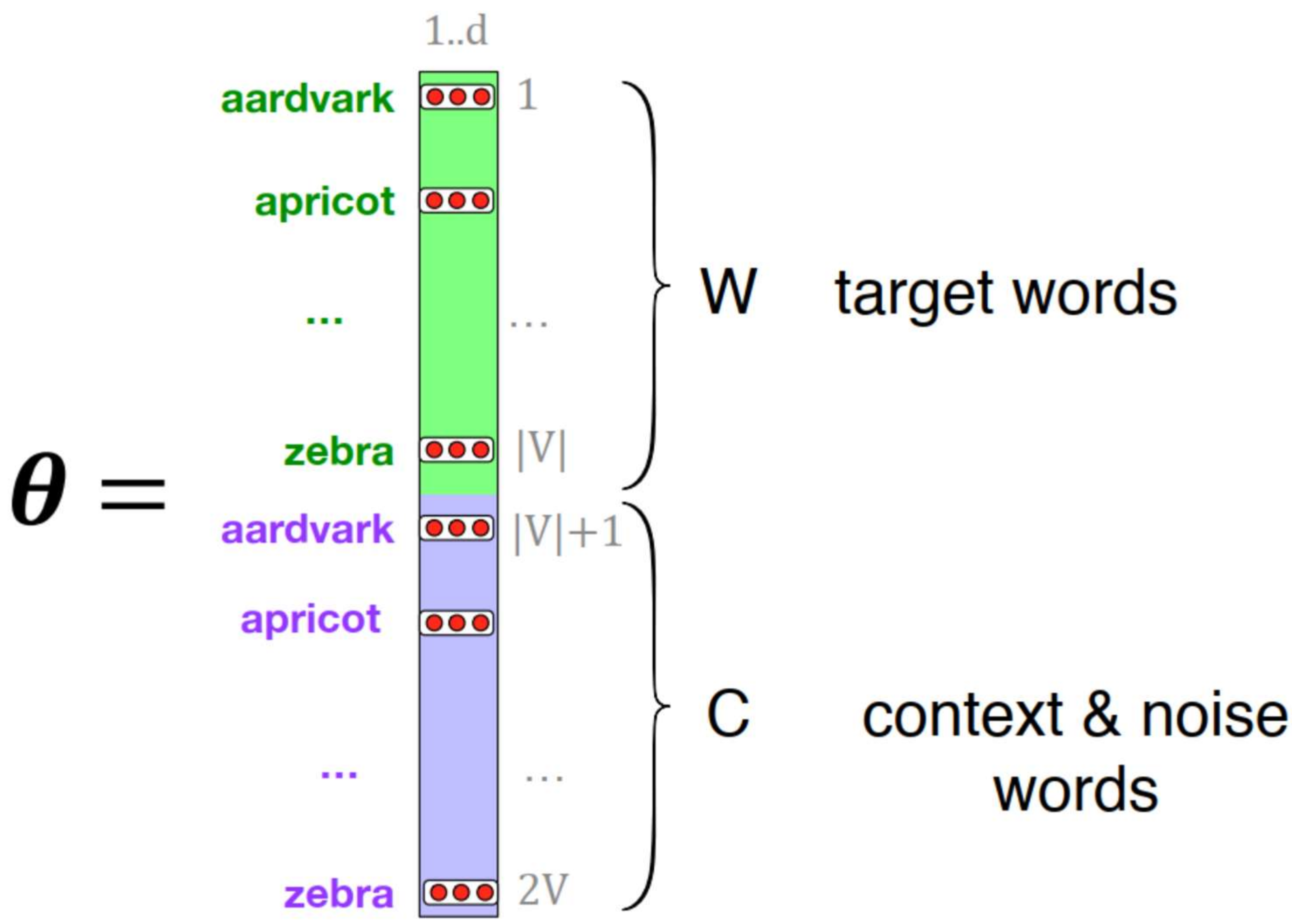
A probabilistic classifier, given

- a test target word w
- its context window of L words $c_{1:L}$

Estimates probability that w occurs in this window based on similarity of w (embeddings) to $c_{1:L}$ (embeddings).

To compute this, we just need embeddings for all the words.

Parameters:Target (W) and Context (C)



Word2Vec: the Approach

1. Treat the target word *t* and a neighboring context word *c* as **positive examples**.
2. Randomly sample other words in the lexicon to get **negative examples**
3. Use **logistic regression** to train a classifier to distinguish those two cases
4. Use the **learned classifier weights** as the **embeddings**

Word2Vec: Training

Assume a +/- 2 (**L = 4**) word window, given training sentence:

…lemon, a [**tablespoon of apricot jam**, **a**] pinch…

Positive (+) examples:
(**apricot**, **tablespoon**),(**apricot**, **of**),(**apricot**, **jam**),(**apricot**, **a**)

Negative (-) K (typically double (+)) examples:
(**apricot**, **aardvark**),(**apricot**, **my**),(**apricot**, **where**),(**apricot**, **coaxial**)
(**apricot**, **seven**),(**apricot**, **forever**),(**apricot**, **dear**),(**apricot**, **if**)

Word2Vec: the Approach

Given the set of **positive** and **negative** training instances, and an **initial set of embedding vectors**

The goal of learning is to adjust those word vectors such that we:

- **maximize** the similarity of the target word, context word pairs (w, c_{pos}) drawn from the **positive** data
- **minimize** the similarity of the (w, c_{neg}) pairs drawn from the **negative** data.

Loss Function

Loss function for one w with c_{pos} , c_{neg1} ...c_{negk}

Maximize the **similarity of the target with the actual context words (+)**, and minimize **the similarity of the target with the k negative sampled non-neighbor words (-)**.

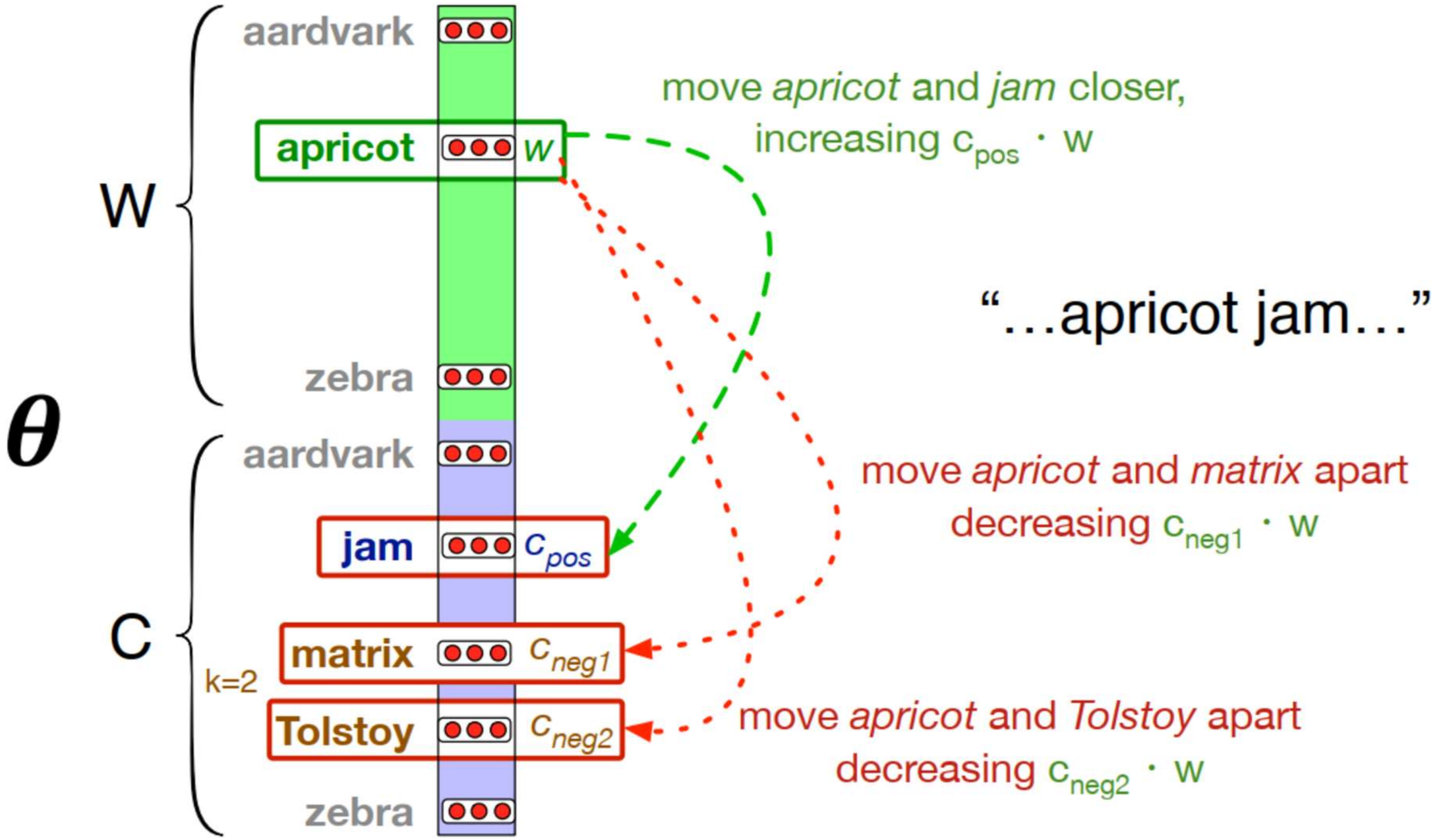
$$
\begin{aligned}
L_{CE} &= -\log\left[P(+|w,c_{pos})\prod_{i=1}^{k}P(-|w,c_{neg_i})\right] \\
&= -\left[\log P(+|w,c_{pos})+\sum_{i=1}^{k}\log P(-|w,c_{neg_i})\right] \\
&= -\left[\log P(+|w,c_{pos})+\sum_{i=1}^{k}\log\left(1-P(+|w,c_{neg_i})\right)\right] \\
&= -\left[\log\sigma(c_{pos}\cdot w)+\sum_{i=1}^{k}\log\sigma(-c_{neg_i}\cdot w)\right]
\end{aligned}
$$

Classifier: Learning Process

- **How to learn?**
  - **use stochastic gradient descent**

- **Adjust the word weights to:**
  - **make the positive pairs more likely**
  - **and the negative pairs less likely,**
  - **... for the entire training set.**

Gradient Descent: Single Step



Loss Function Derivatives

$$L_{CE} = -\left[\log \sigma(c_{pos} \cdot w) + \sum_{i=1}^{k} \log \sigma(-c_{neg_i} \cdot w)\right]$$

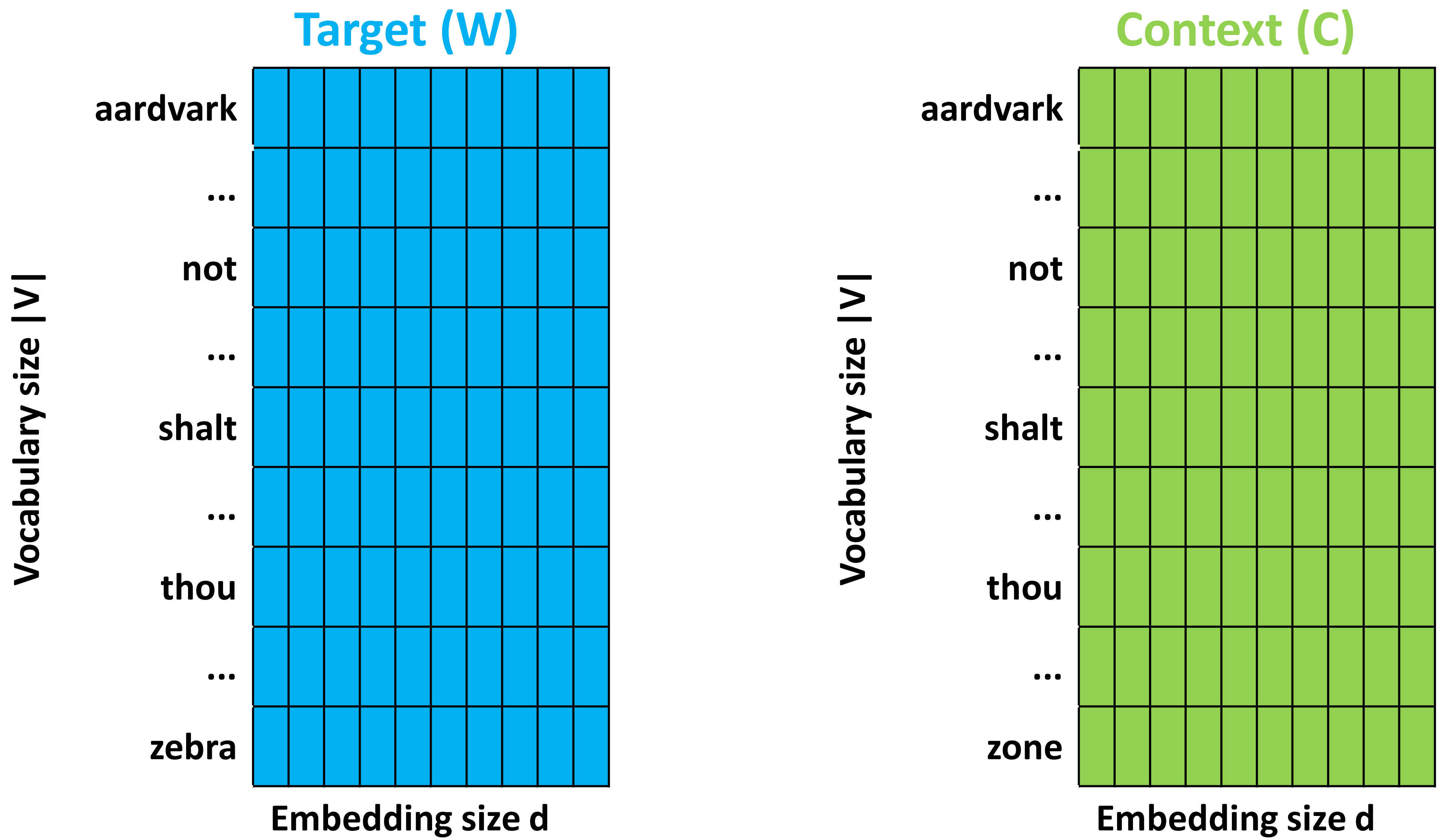
$$\frac{\partial L_{CE}}{\partial c_{pos}} = [\sigma(c_{pos} \cdot w) - 1]w$$

$$\frac{\partial L_{CE}}{\partial c_{neg}} = [\sigma(c_{neg} \cdot w)]w$$

$$\frac{\partial L_{CE}}{\partial w} = [\sigma(c_{pos} \cdot w) - 1]c_{pos} + \sum_{i=1}^{k} [\sigma(c_{\mathrm{neg}_i} \cdot w)]c_{neg_i}$$

Gradient Descent: Updates

Start with randomly initialized W and C matrices



Gradient Descent: Updates

... then incrementally do updates using.

$$
\begin{aligned}
c_{pos}^{t+1} &= c_{pos}^{t} - \eta[\sigma(c_{pos}^{t} \cdot w^{t}) - 1]w^{t} \\
c_{neg}^{t+1} &= c_{neg}^{t} - \eta[\sigma(c_{neg}^{t} \cdot w^{t})]w^{t} \\
w^{t+1} &= w^{t} - \underset{\text{learning rate}}{\eta} \left[[\sigma(c_{pos} \cdot w^{t}) - 1]c_{pos} + \sum_{i=1}^{k} [\sigma(c_{\mathrm{neg}_i} \cdot w^{t})]c_{neg_i} \right]
\end{aligned}
$$

learning rate

Skip Gram Word2Vec: Summary

- **Start with |V| random d-dimensional vectors as initial embeddings**
  - **Train a classifier based on embedding similarity**
  - **Take a corpus and take pairs of words that co-occur as positive examples**
  - **Take pairs of words that don't co-occur as negative examples**
  - **Train the classifier to distinguish these by slowly adjusting all the embeddings to improve the classifier performance**
  - **Throw away the classifier code and keep the embeddings.**

Sliding Window Size

- **Small windows (+/- 2) : nearest words are syntactically similar words in same taxonomy**
  - *Hogwarts* **nearest neighbors are other fictional schools**
    - *Sunnydale*, *Evernight*, *Blandings*

- **Large windows (+/- 5) : nearest words are related words in same semantic field**
  - *Hogwarts* **nearest neighbors are Harry Potter world:**
    - *Dumbledore*, *half-blood*, *Malfoy*