TRAINING ON AI AND MACHINE LEARNING WITH PYTHON

ORGANIZED BY
CUET IT BUSINESS INCUBATOR

python
MACHINE
LEARNING

Word Embeddings

Word Embedding And Word2Vec

Representing words as discrete symbols

In traditional NLP, we regard words as discrete symbols:

hotel, motel – 2 words taken as example

Such symbols for words can be represented by one-hot vectors:

motel = [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]

hotel = [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]

Vector dimension = number of words in vocabulary

Problem with words as discrete symbols

In web search, if a user searches for "Seattle motel", we would like to match documents containing "Seattle hotel"

But:

motel = [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]

hotel = [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]

These two vectors are orthogonal

There is no natural notion of **similarity** for one-hot vectors!

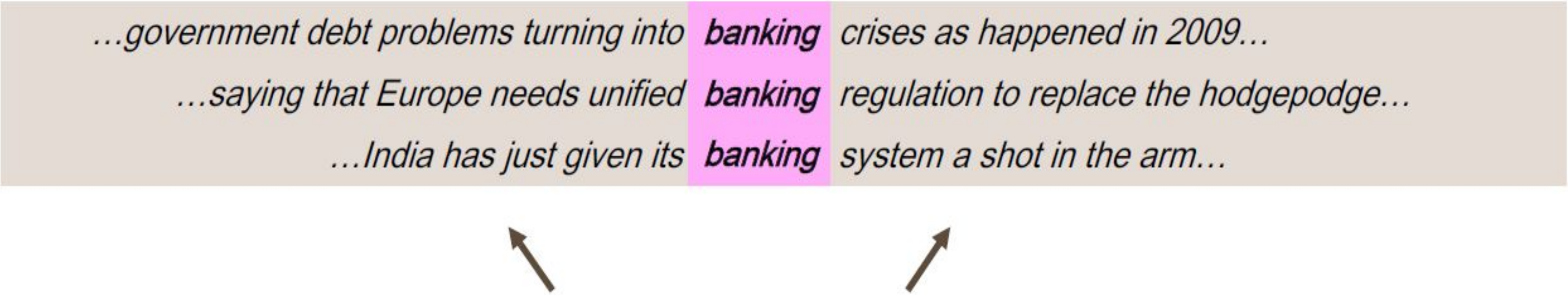
Solution: learn to encode similarity in the vectors themselves

Representing words by their context

- **Distributional semantics**: A word's meaning is given by the words that frequently appear close-by.

- When a word *w* appears in a text, its **context** is the set of words that appear nearby (within a fixed-size window).

- We use the many contexts of *w* to build up a representation of *w*



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

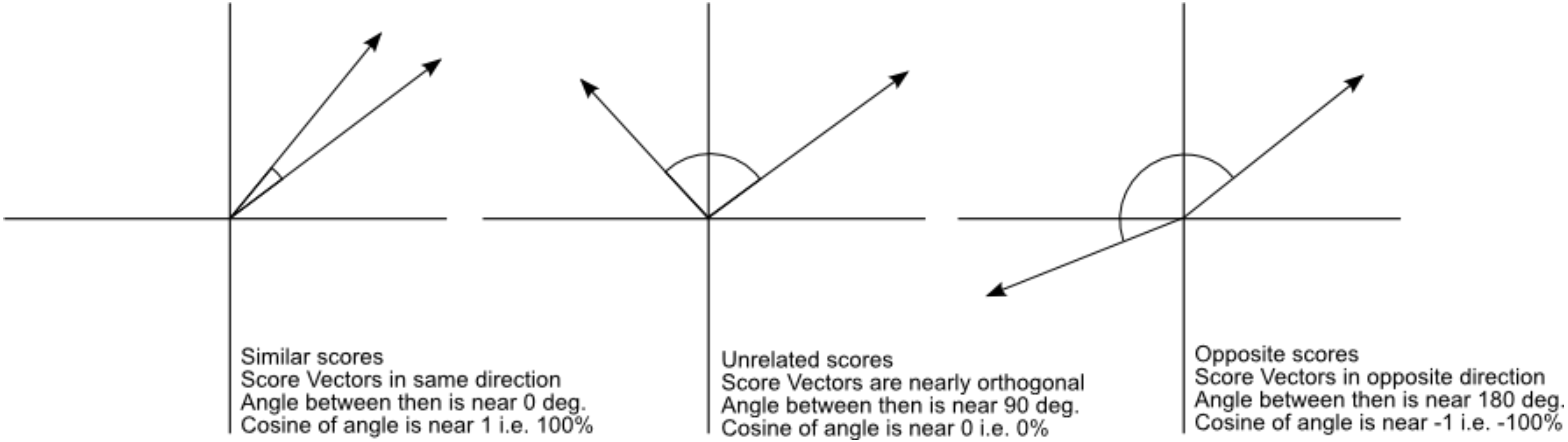
Word Embedding

We will build a dense vector for each word, chosen so that it is similar to vectors of words that appear in similar contexts, measuring similarity as the vector dot (scalar) product.

$$\textit{banking} = \begin{bmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{bmatrix}$$

Word embeddings are also called (word) vectors or (neural) word representations
They are a distributed representation.

Cosine Similarities

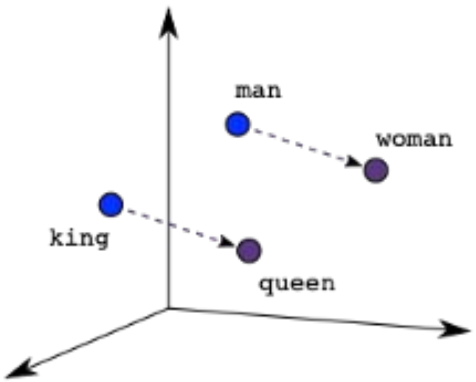


$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}},$$

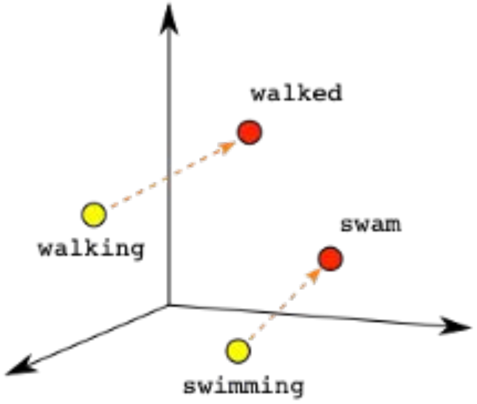
- ➔ When the similarity score is 1 (or close) then the two vectors are the similar,
- ➔ when 0 then two vectors are independent,
- ➔ when -1 then two vectors point in the opposite direction.

In case of word-vector representation, we would say that when the similarity score is -1 then the words are similar **but** have opposite meaning, for example words “hot” and “cold”.

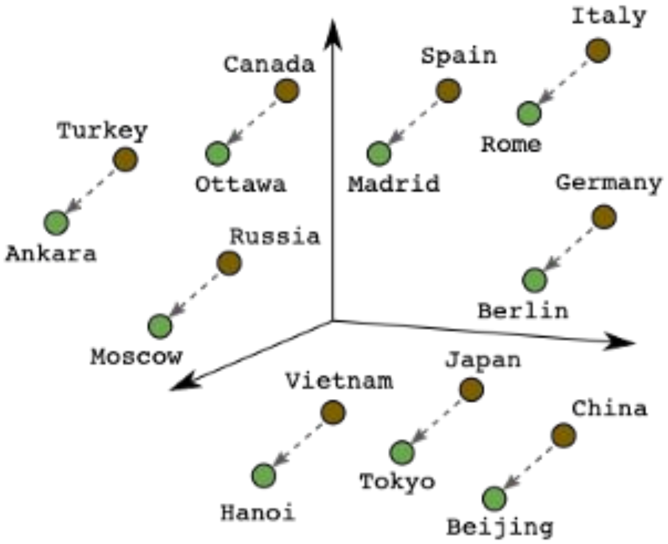
Word Analogies



Male-Female



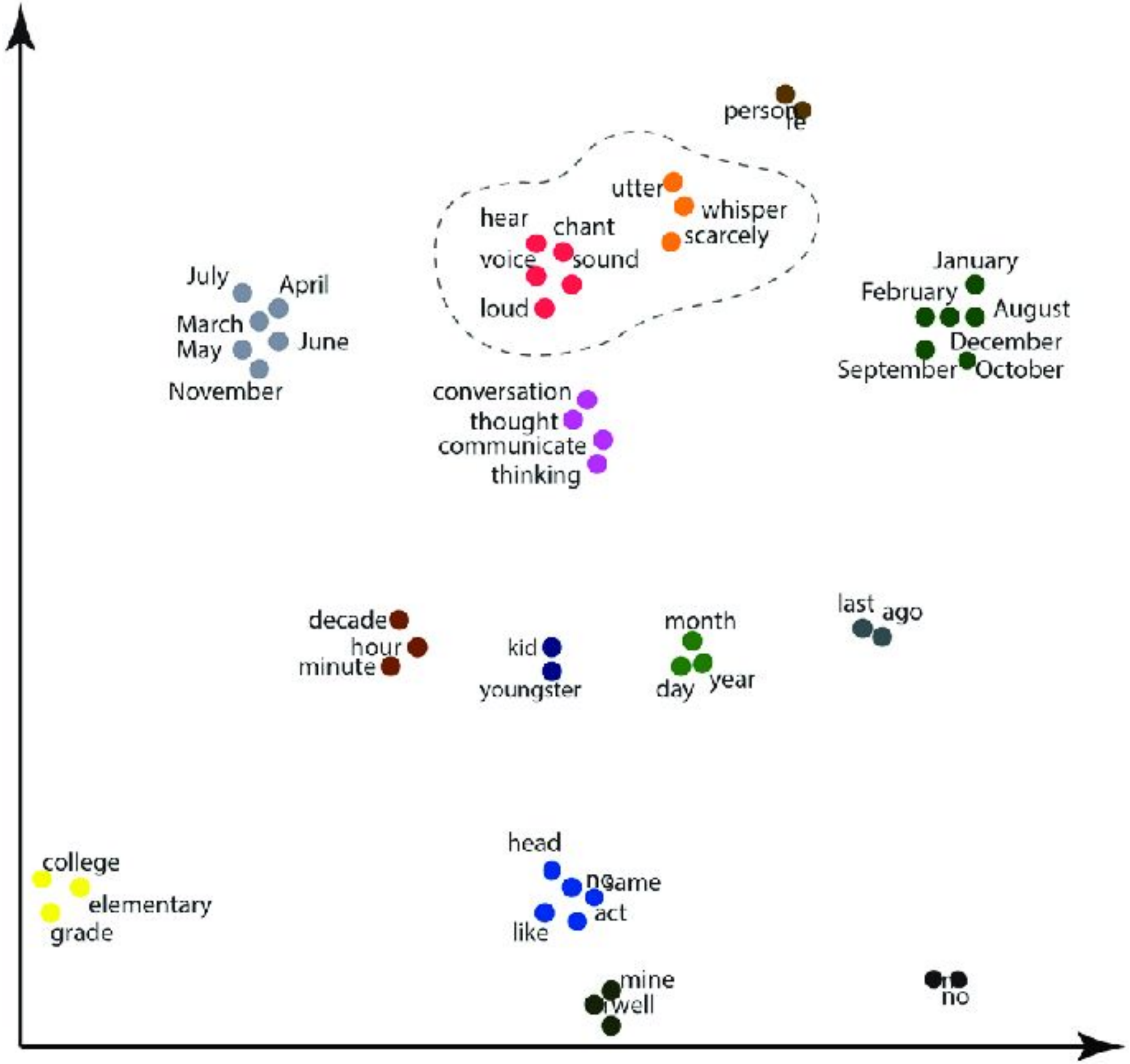
Verb Tense



Country-Capital

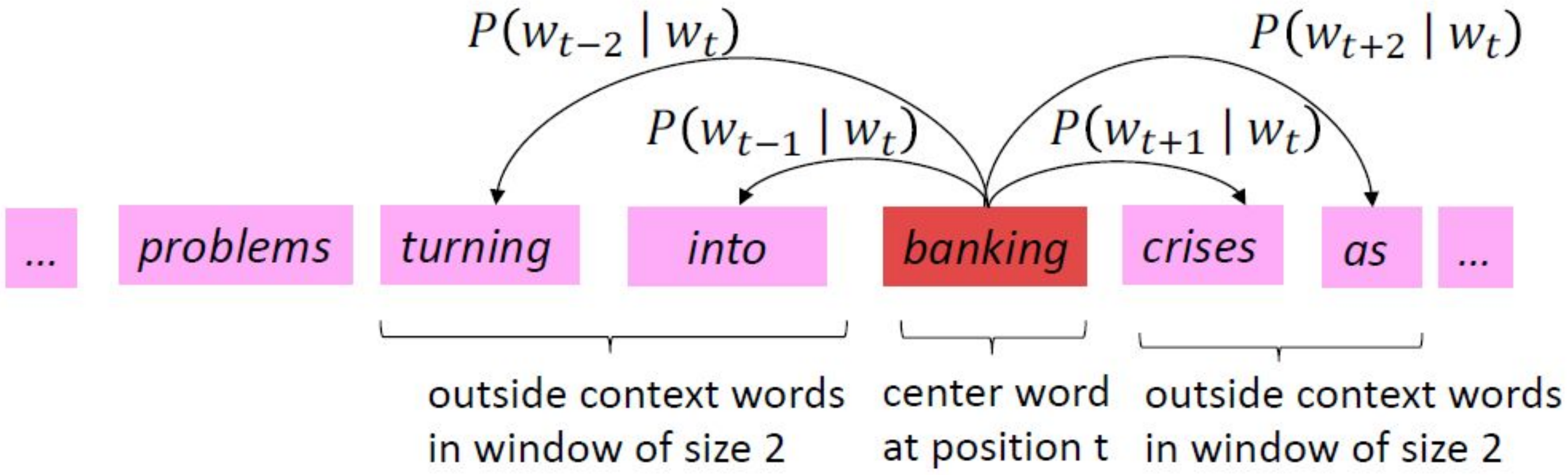
King — Man + Woman = Queen

Word Analogies



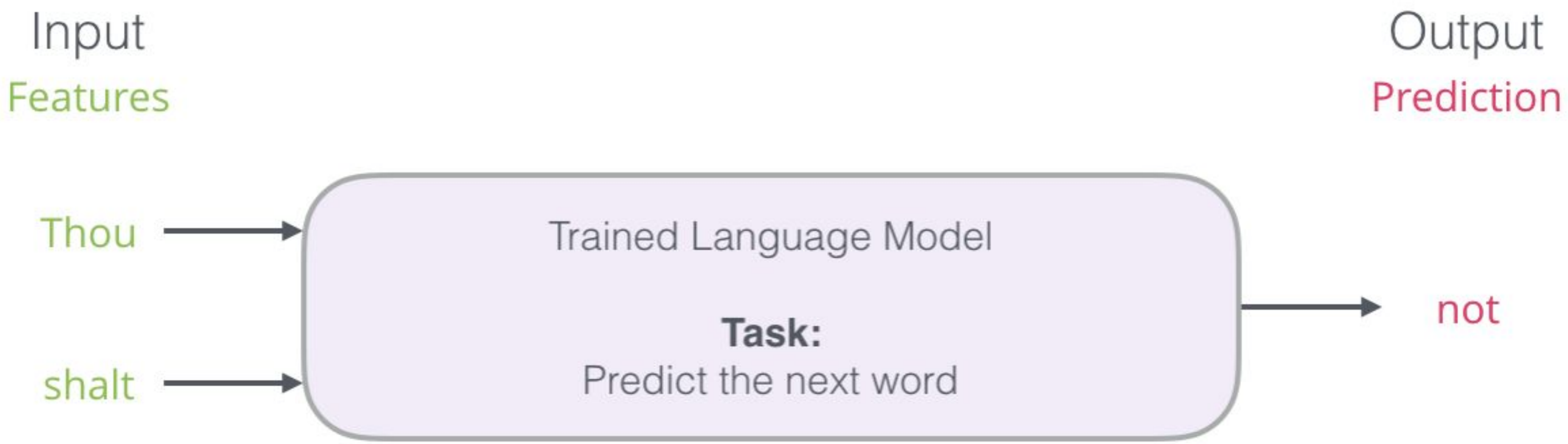
Word2vec: Overview

- ➔ Word2vec (Mikolov et al. 2013) is a framework for learning word vectors.
- ➔ The word2vec algorithm uses a neural network model to learn word associations from a large corpus of text.
- ➔ Once trained, such a model can detect synonymous words or suggest additional words for a partial sentence.
- ➔ As the name implies, word2vec represents each distinct word with a particular list of numbers called a vector.

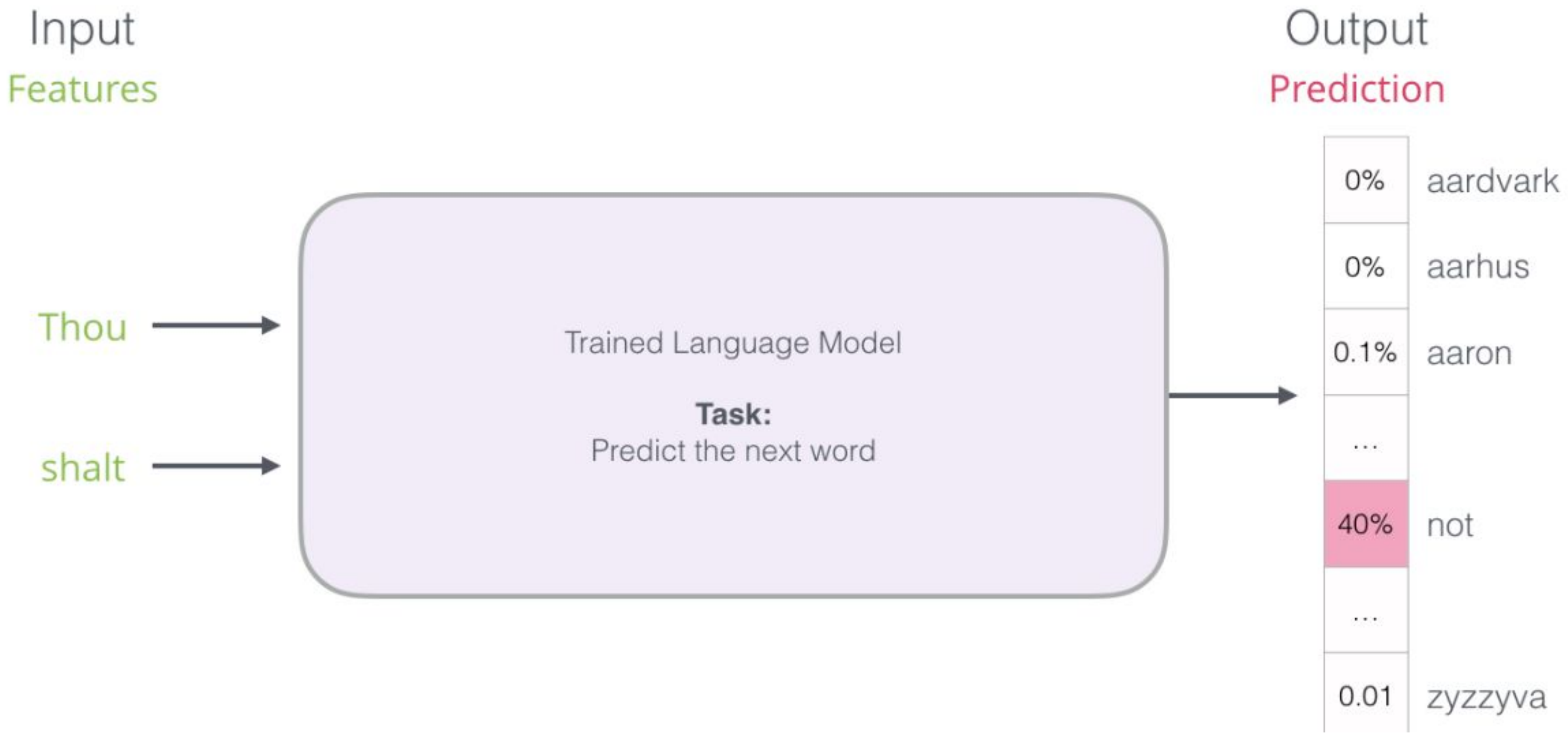

$P(w_{t-2} \mid w_t)$
$P(w_{t+2} \mid w_t)$
$P(w_{t-1} \mid w_t)$
$P(w_{t+1} \mid w_t)$
...
problems
turning
into
banking
crises
as
...
outside context words in window of size 2
center word at position t
outside context words in window of size 2

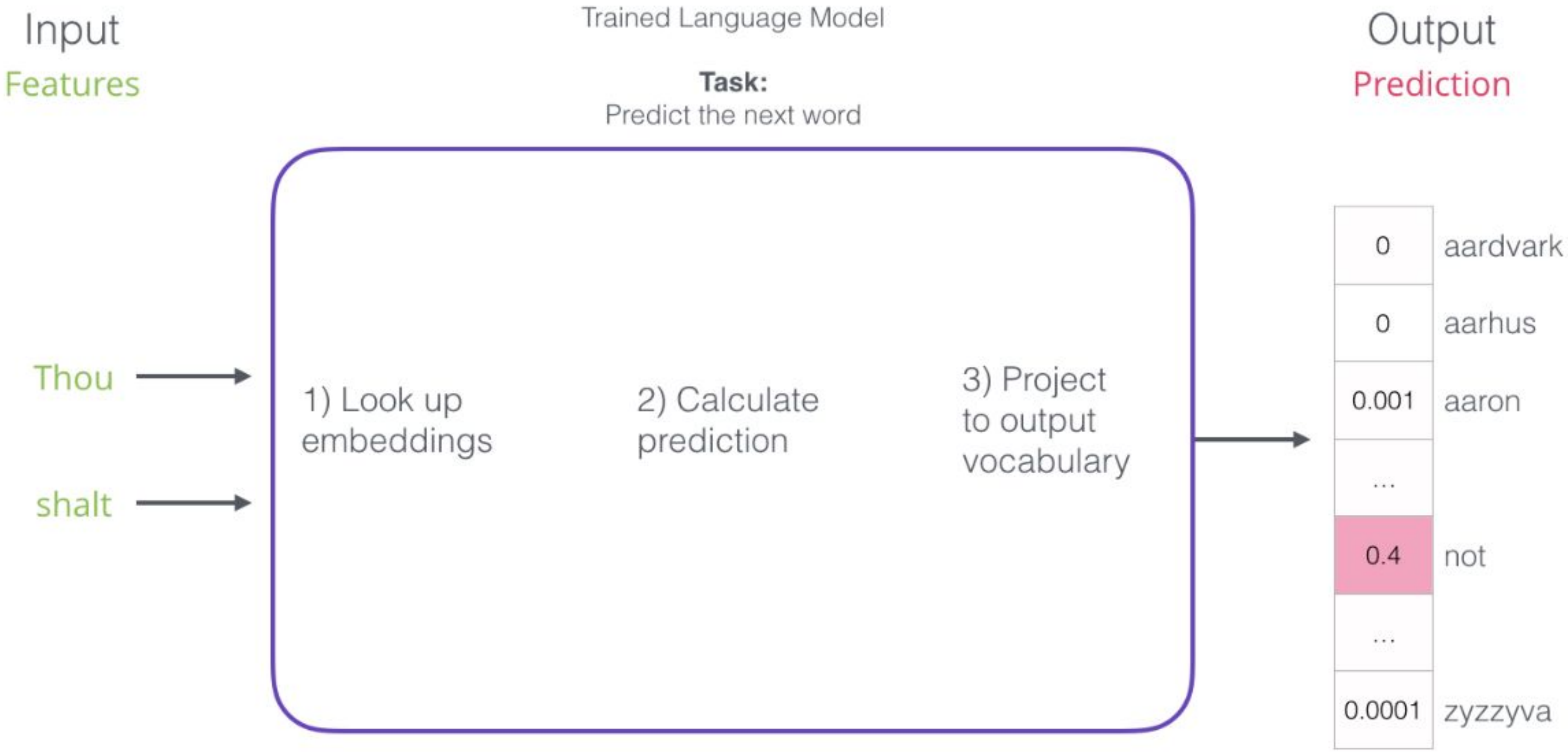
Language Modeling

Next-word prediction is a task that can be addressed by a *language model*. A language model can take a list of words (let’s say two words), and attempt to predict the word that follows them.

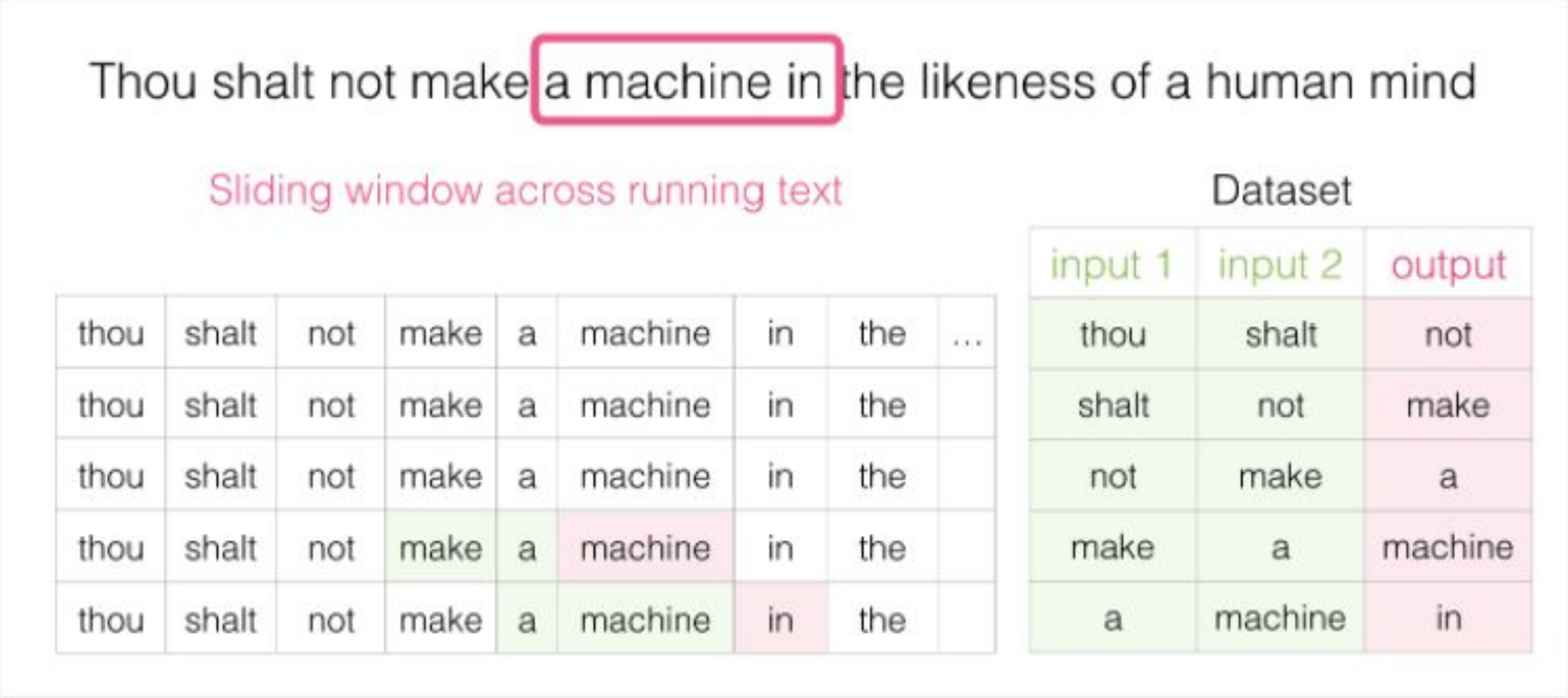


But in practice, the model doesn't output only one word. It actually outputs a probability score for all the words it knows (the model's "vocabulary", which can range from a few thousand to over a million words).




Input
Features
Trained Language Model
Task:
Predict the next word
Output
Prediction
Thou
shalt
1) Look up embeddings
2) Calculate prediction
3) Project to output vocabulary
0 aardvark
0 aarhus
0.001 aaron
...
0.4 not
...
0.0001 zyzzyva

Language Model Training



Word2Vec Model

Two model variants:

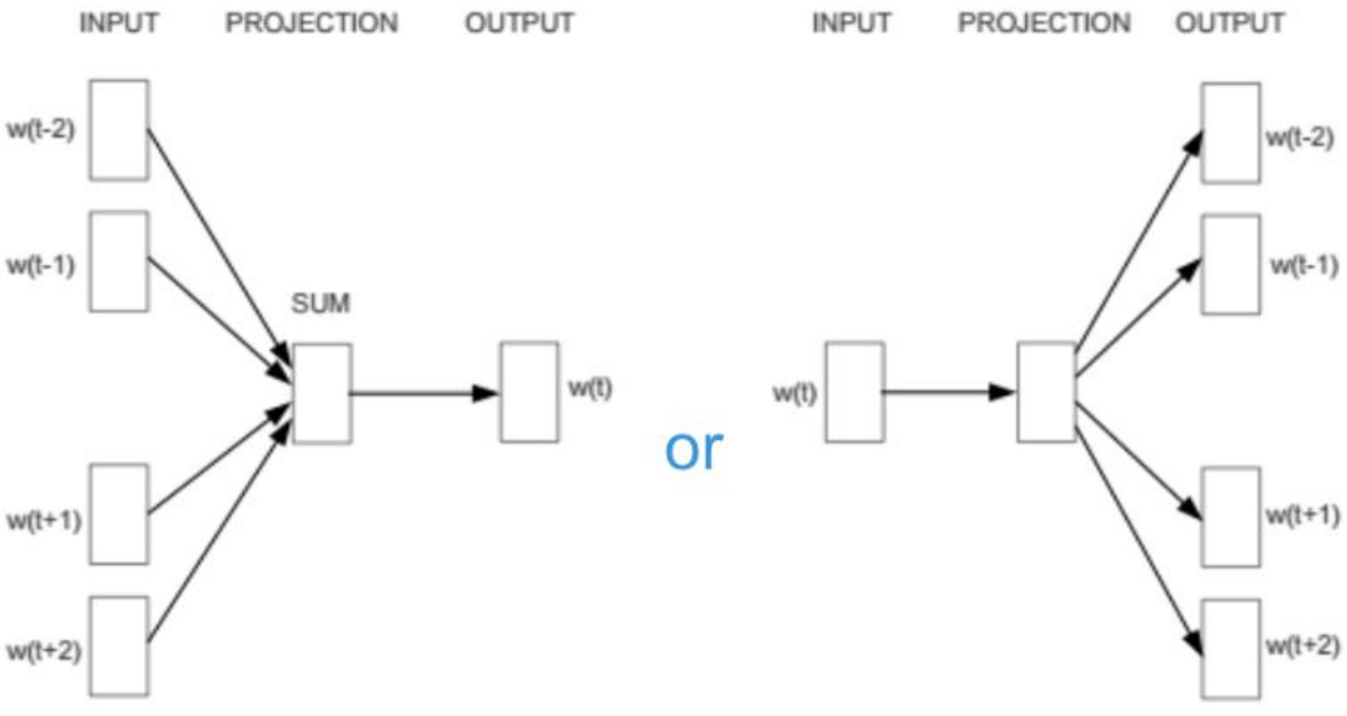
1. Skip-grams (SG)

   - Predict context ("outside") words (position independent) given center word

2. Continuous Bag of Words (CBOW)

   - Predict center word from (bag of) context words.

CBOW vs Skip Gram



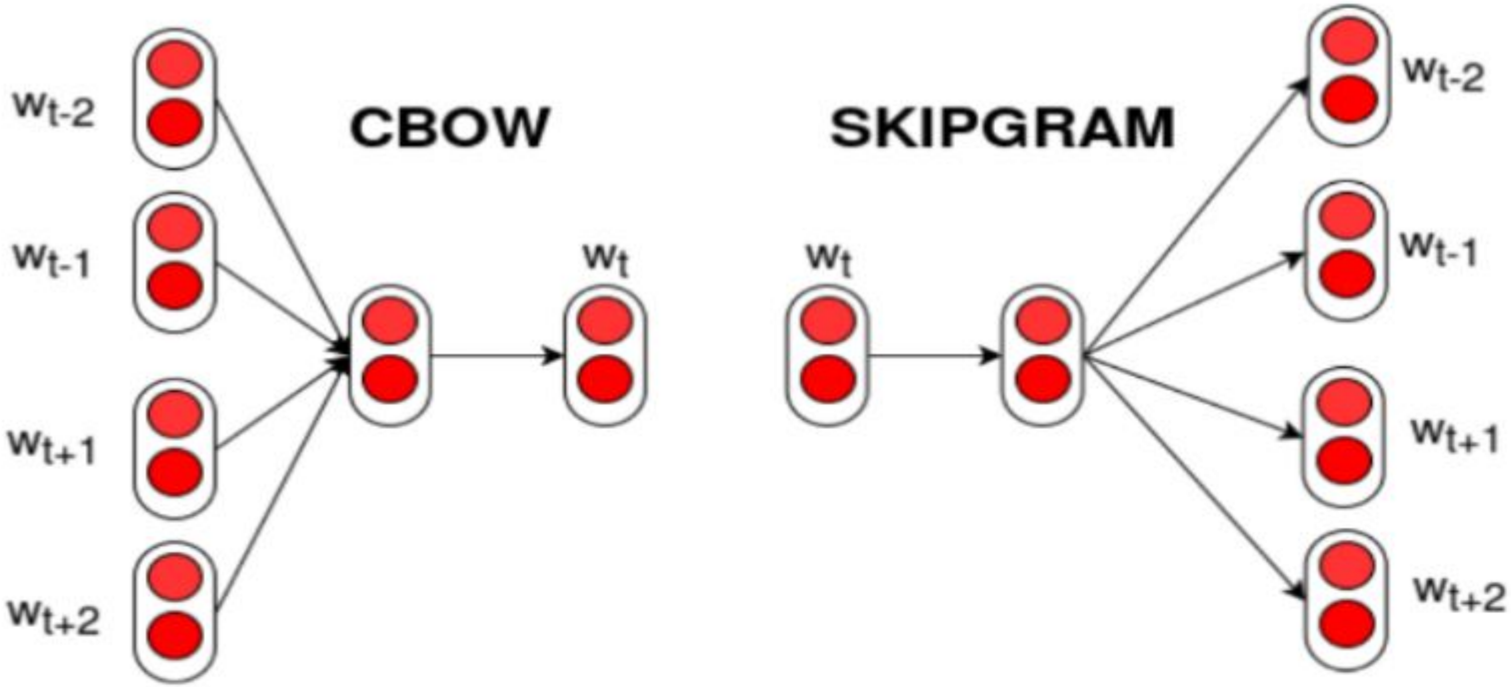
Given a set of (neighboring) words, **guess single words** that potentially occur along with this set of words.

CBOW
(Continuous Bag Of Words)

Skip-gram

Guess potential neighboring words based on the single word being analyzed.

CBOW vs Skip Gram

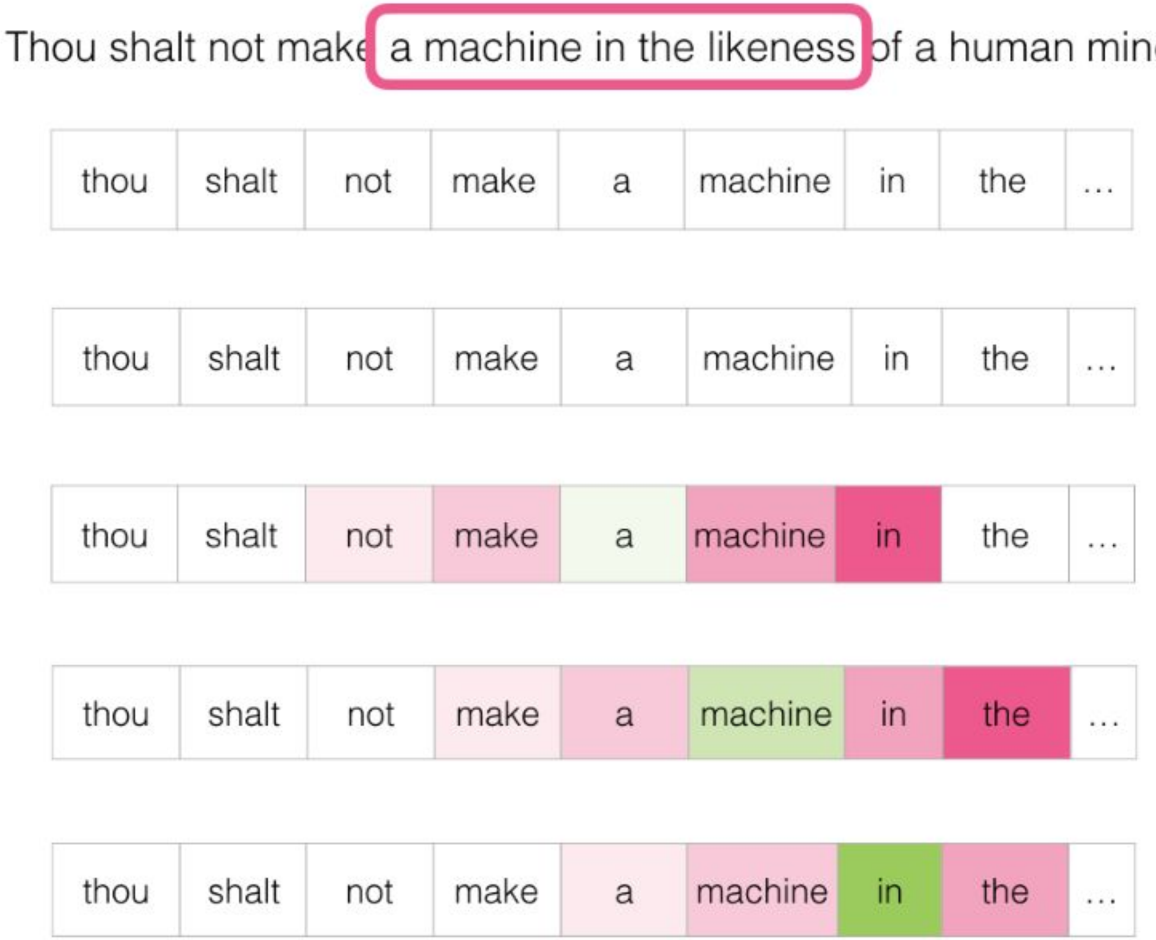


$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0} logp(w_t|w_{t+j})$$

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0} logp(w_{t+j}|w_t)$$

Skip-gram(SG)

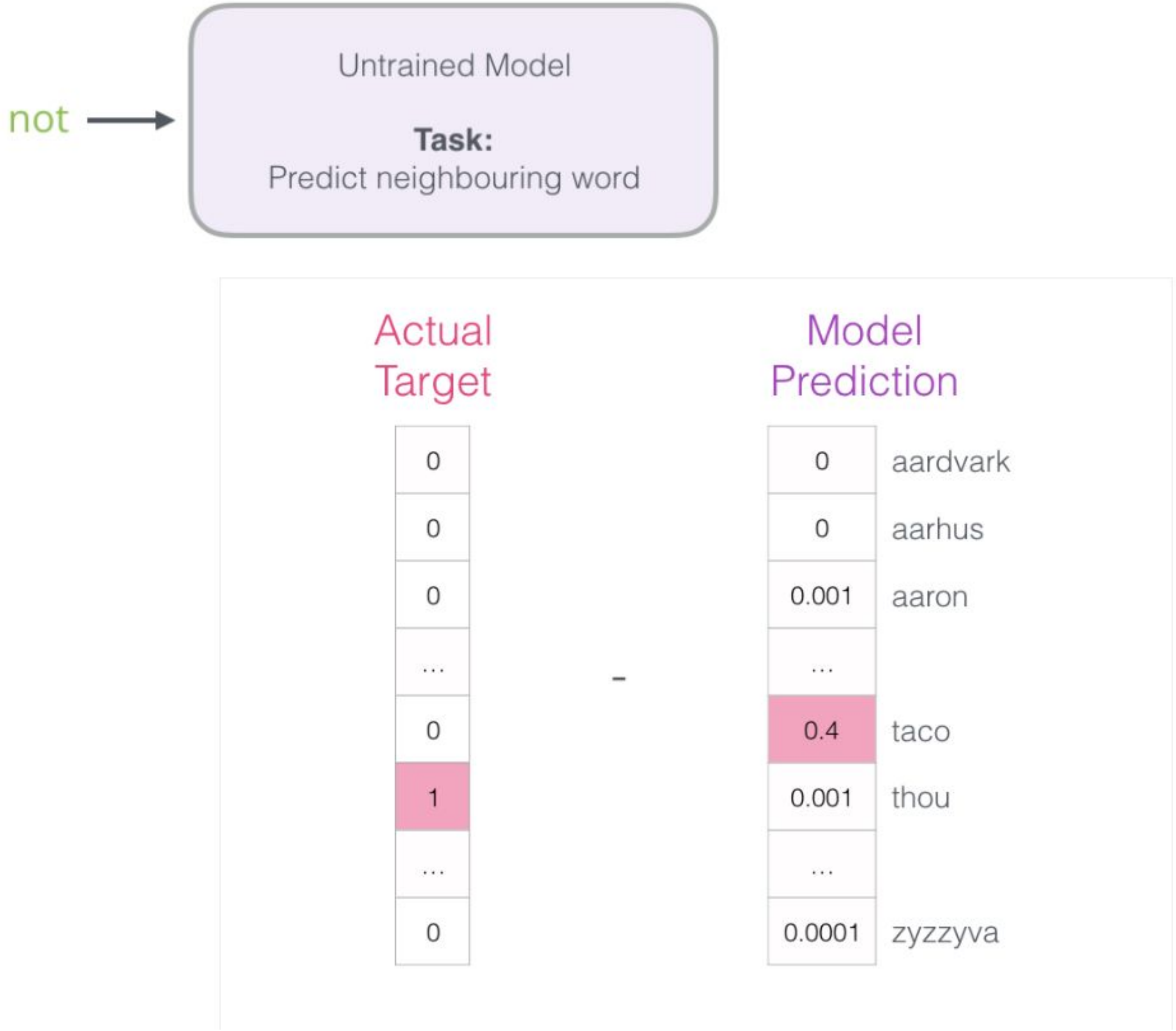
Instead of guessing a word based on its context (the words before and after it), skip-gram tries to guess context words using the center word.



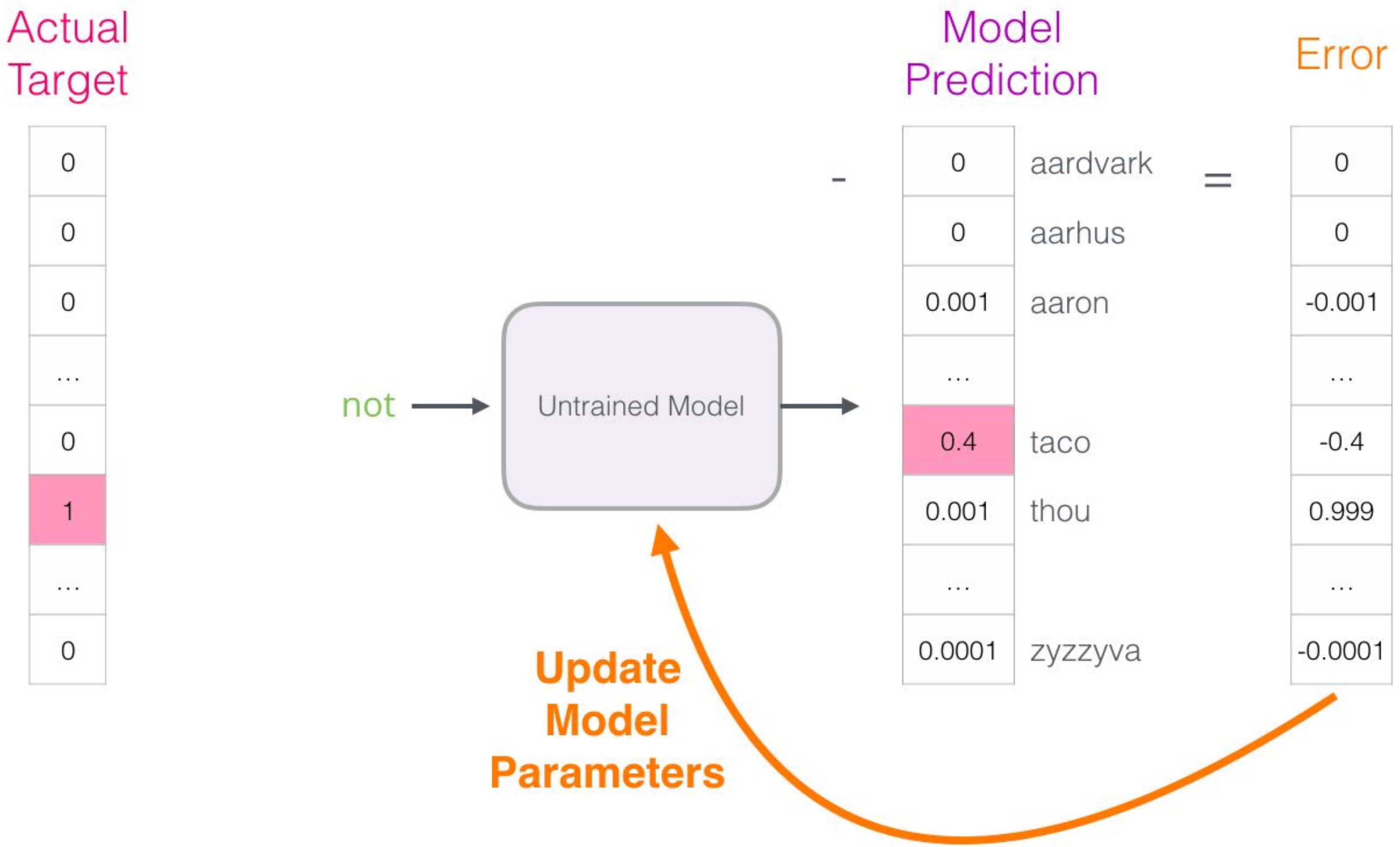
input word	target word
not	thou
not	shalt
not	make
not	a
make	shalt
make	not
make	a
make	machine
a	not
a	make
a	machine
a	in
machine	make
machine	a
machine	in
machine	the
in	a
in	machine
in	the
in	likeness

Train a basic neural language model

input word	target word
not	thou
not	shalt
not	make
not	a
make	shalt
make	not
make	a
make	machine
a	not
a	make
a	machine
a	in
machine	make
machine	a
machine	in
machine	the
in	a
in	machine
in	the
in	likeness



This error vector can now be used to update the model next time to get the right context word.



Skip-gram with Negative Sampling

input word	target word
not	thou
not	shalt
not	make
not	a
make	shalt
make	not
make	a
make	machine

input word	output word	target
not	thou	1
not	shalt	1
not	make	1
not	a	1
make	shalt	1
make	not	1
make	a	1
make	machine	1

But If all of our examples are positive (target: 1), we build a smartass model that always returns 1 – achieving 100% accuracy, but learning nothing and generating garbage embeddings.

So, we need to introduce *negative samples* to our dataset – samples of words that are not neighbors.

Skipgram

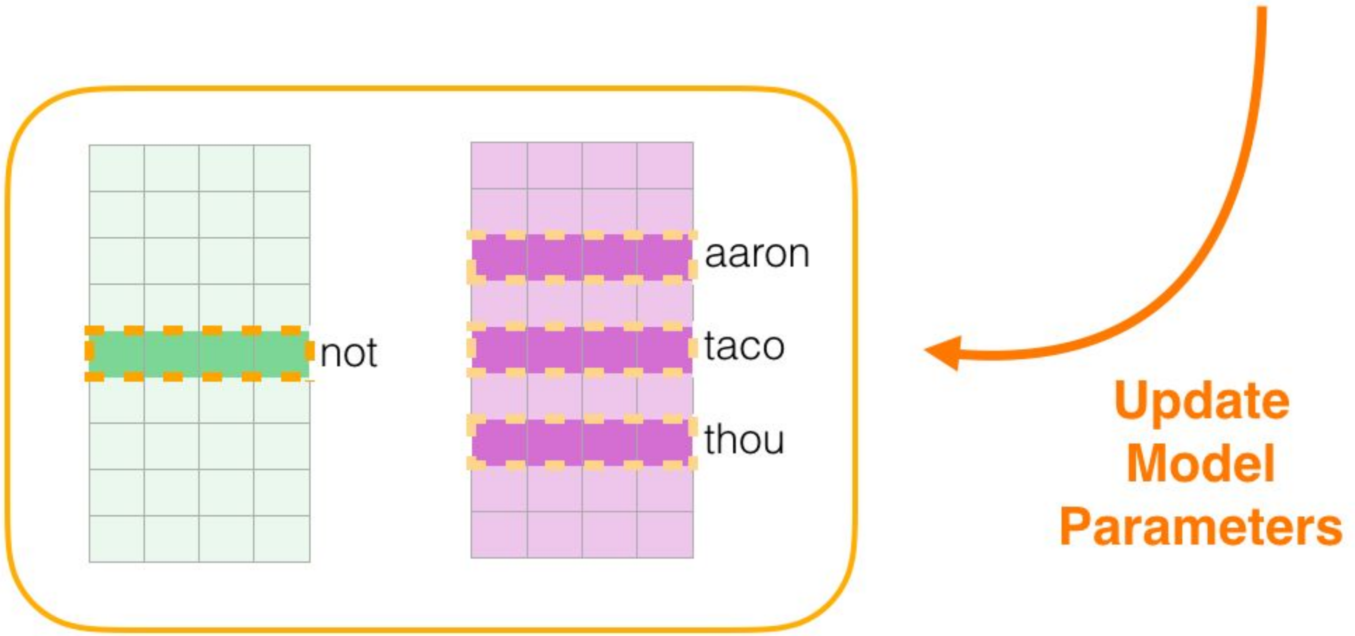
shalt	not	make	a	machine

input	output
make	shalt
make	not
make	a
make	machine

Negative Sampling

input word	output word	target
make	shalt	1
make	aaron	0
make	taco	0

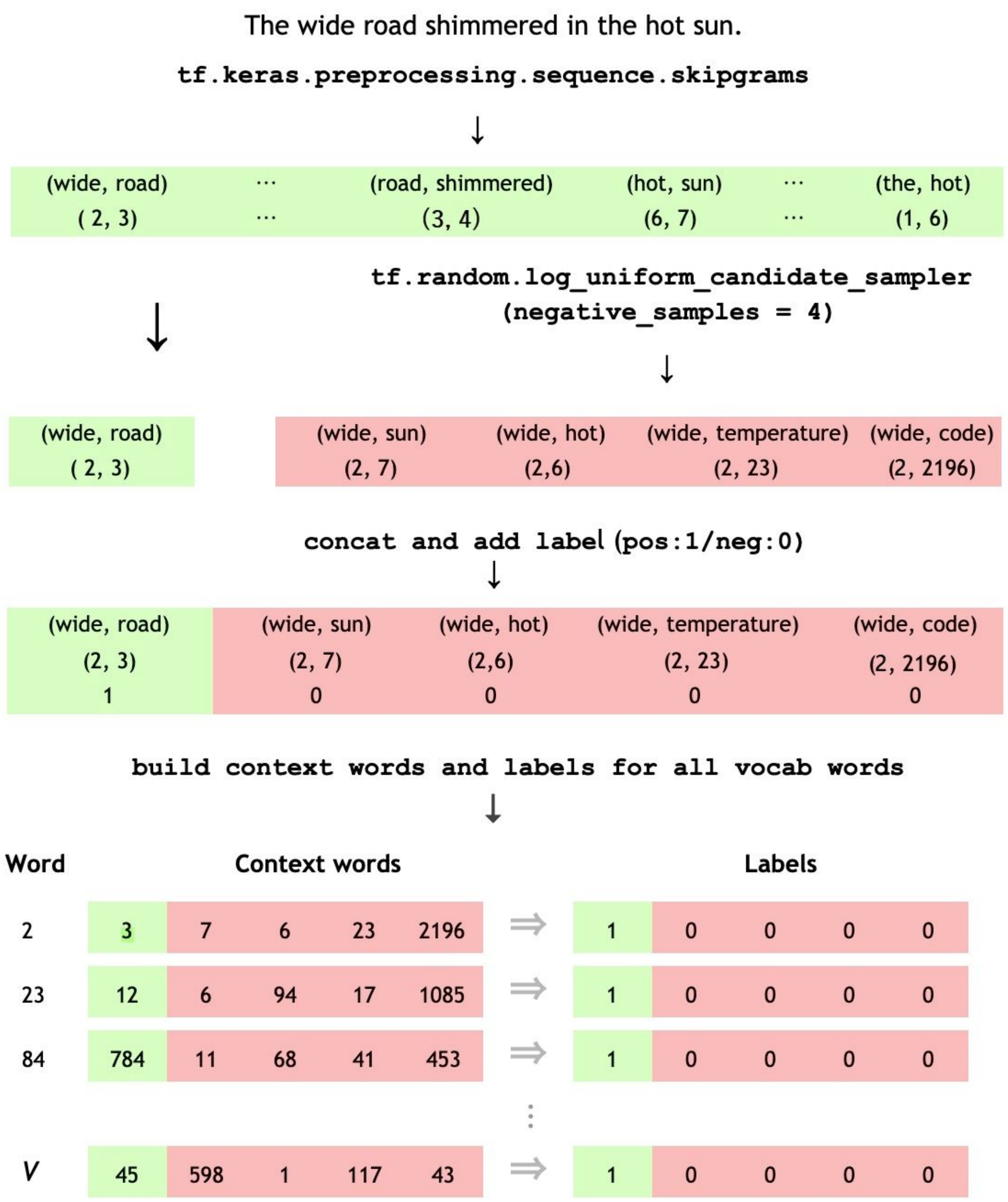
input word	output word	target	input • output	sigmoid()	Error
not	thou	1	0.2	0.55	0.45
not	aaron	0	-1.11	0.25	-0.25
not	taco	0	0.74	0.68	-0.68



How to Make training samples ?

Window Size	Text	Skip-grams
2	[The **wide** road shimmered] in the hot sun.	wide, the wide, road wide, shimmered
	The [wide road **shimmered** in the] hot sun.	shimmered, wide shimmered, road shimmered, in shimmered, the
	The wide road shimmered in [the hot **sun**].	sun, the sun, hot
3	[The **wide** road shimmered in] the hot sun.	wide, the wide, road wide, shimmered wide, in
	[The wide road **shimmered** in the hot] sun.	shimmered, the shimmered, wide shimmered, road shimmered, in shimmered, the shimmered, hot
	The wide road shimmered [in the hot **sun**].	sun, in sun, the sun, hot

With negative sampling



A wonderful Article on Skip Gram

[Skip Gram]

CBOW

Training Example	Context Word	Target Word
#1	(i, natural)	like
#2	(like, language)	natural
#3	(natural, processing)	language
#4	(language)	processing

CBOW

Training Example	Context Word	Target Word
#1	(i, natural)	like
#2	(like, language)	natural
#3	(natural, processing)	language
#4	(language)	processing

One hot encoding

	i	like	natural	language	processing
i	1	0	0	0	0
like	0	1	0	0	0
natural	0	0	1	0	0
language	0	0	0	1	0
processing	0	0	0	0	1

Training Example	Encoded Context Word	Encoded Target Word
#1	([1,0,0,0,0], [0,0,1,0,0])	[0,1,0,0,0]
#2	([0,1,0,0,0], [0,0,0,1,0])	[0,0,1,0,0]
#3	([0,0,1,0,0], [0,0,0,0,1])	[0,0,0,1,0]
#4	([0,0,0,1,0])	[0,0,0,0,1]

GloVe Model

- ➔ GloVe stands for Global Vectors for word representation.
- ➔ generate word embeddings by aggregating global word co-occurrence matrices from a given corpus.

- ➔ the co-occurrence matrix tells you how often a particular word pair occurs together.
- ➔ Each value in the co-occurrence matrix represents a pair of words occurring together.

GloVe : Differences with Word2Vec

- Word2Vec
  - Only local view of data
  - Local windows capture word similarities

- GloVe
  - Word co-occurrence matrix
    - captures global information
  - Simpler objective/cost function
    - no normalization required

Window based co-occurrence matrix

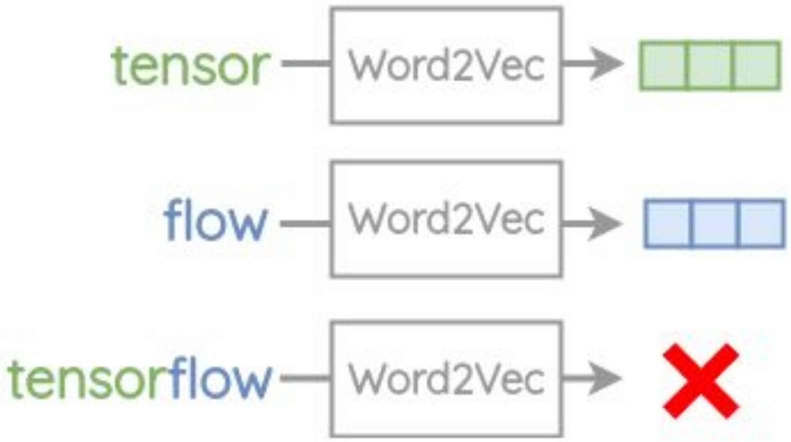
- Window length 1 (more common: 5–10)
- Symmetric (irrelevant whether left or right context)
- Example corpus:
  - I like deep learning
  - I like NLP
  - I enjoy flying

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

Limitations of Word2Vec

Out of Vocabulary(OOV) Words :

In Word2Vec, an embedding is created for each word. As such, it can’t handle any words it has not encountered during its training.



Morphology :

FastText

To solve the challenges of Word2Vec, Bojanowski et al. proposed a new embedding method called FastText.

Sub-word generation :

For a word, we generate character n-grams of length 3 to 6 present in it.

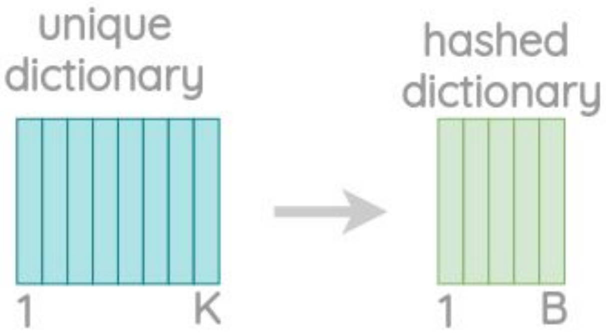
→ We take a word and add angular brackets to denote the beginning and end of a word.

→ Then, we generate character n-grams of length n. For example, for the word “eating”, character n-grams of length 3 can be generated by sliding a window of 3 characters from the start of the angular bracket till t

hed.

➔ Thus, we get a list of character n-grams for a word.

➔ Since there can be huge number of unique n-grams, we apply hashing to bound the memory requirements. Instead of learning an embedding for each unique n-gram, we learn total B embeddings where B denotes the bucket size,



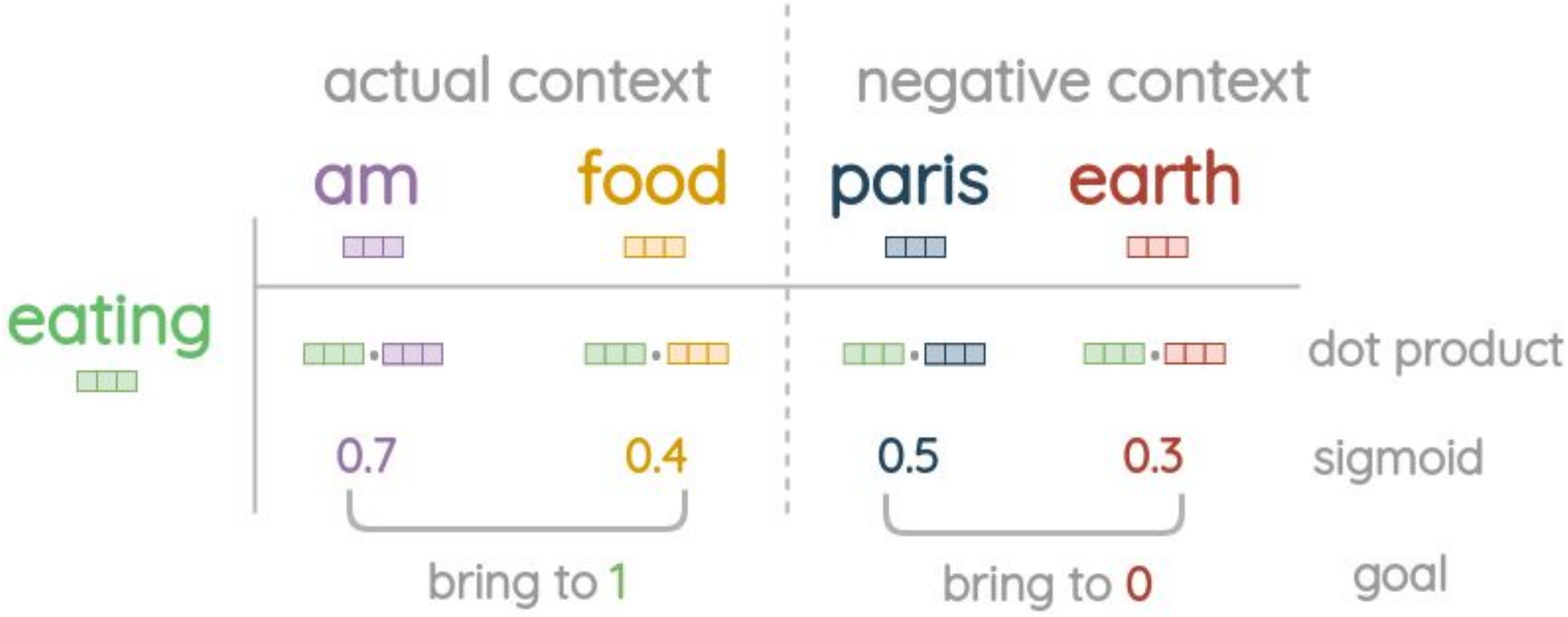
Each character n-gram is hashed to an integer between 1 to B.

→ First, the embedding for the center word is calculated by taking a sum of vectors for the character n-grams and the whole word itself.

→ For the actual context words, we directly take their word vector from the embedding table without adding the character n-grams.

→ Now, we collect negative samples randomly with probability proportional to the square root of the unigram frequency. For one actual context word, 5 random negative words are sampled.

➔ We take dot product between the center word and the actual context words and apply sigmoid function to get a match score between 0 and 1.



➔ Based on the loss, we update the embedding vectors with SGD optimizer to bring actual context words closer to the center word but increase distance to the negative samples.

References

i. Fundamentals of Bag Of Words and TF-IDF

ii. BoW Model and TF-IDF For Creating Feature From Text

iii. Stanford Online - CS224n: Natural Language Processing with Deep Learning – Lecture 1

iv. Stanford Online - CS224n: Natural Language Processing with Deep Learning – Lecture 2

v. The Illustrated Word2vec by Jay Alammar

vi. NLP — Word Embedding & GloVe by Jonathan Hui

vii. Mathematical Introduction to GloVe Word Embedding

viii. A Visual Guide to FastText Word Embeddings

Thanks to Ahatesham (ETE’17) for the slides content.