UNIVERSIDAD EAFIT

Escuela de Ciencias Aplicadas e Ingeniería

ST7003 Procesamiento Natural del Lenguaje

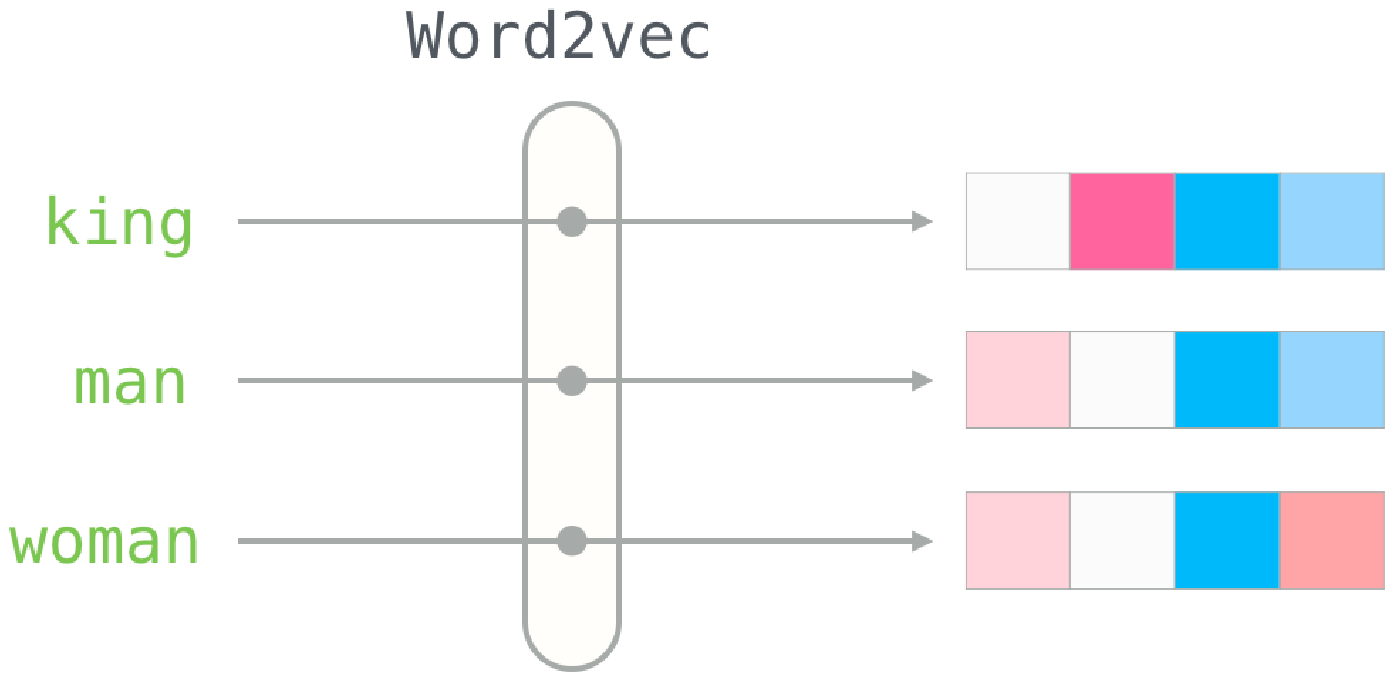
Lecture03 – Word2vec

Contenido

UNIVERSIDAD EAFIT
Escuela de Ciencias Aplicadas e Ingeniería

1. Word embeddings
2. Language modeling
3. Next Word prediction
4. Language model training
5. Continuous bag of words
6. Skipgrams
7. Negative sampling
8. Training

Word Embeddings



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

Distributed Representations of Words and Phrases and their Compositionality

Tomas Mikolov
Google Inc.
Mountain View
mikolov@google.com

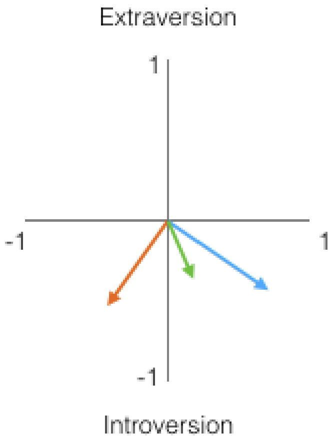
Ilya Sutskever
Google Inc.
Mountain View
ilyasu@google.com

Kai Chen
Google Inc.
Mountain View
kai@google.com

Greg Corrado
Google Inc.
Mountain View
gcorrado@google.com

Jeffrey Dean
Google Inc.
Mountain View
jeff@google.com

Word Embeddings



	Trait #1	Trait #2			
Jay	-0.4	0.8			
Person #1	-0.3	0.2			
Person #2	-0.5	-0.4			

Openness to experience	79 out of 100
Agreeableness	75 out of 100
Conscientiousness	42 out of 100
Negative emotionality	50 out of 100
Extraversion	58 out of 100

	Trait #1	Trait #2	Trait #3	Trait #4	Trait #5
Jay	-0.4	0.8	0.5	-0.2	0.3
Person #1	-0.3	0.2	0.3	-0.4	0.9
Person #2	-0.5	-0.4	-0.2	0.7	-0.1

$$\text{cosine similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|}$$

cosine_similarity(Jay [-0.4, 0.8] , Person #1 [-0.3, 0.2]) = 0.87

cosine_similarity(Jay [-0.4, 0.8] , Person #2 [-0.5, -0.4]) = -0.20

cosine_similarity(Jay [-0.4, 0.8, 0.5, -0.2, 0.3] , Person #1 [-0.3, 0.2, 0.3, -0.4, 0.9]) = 0.66

cosine_similarity(Jay [-0.4, 0.8, 0.5, -0.2, 0.3] , Person #2 [-0.5, -0.4, -0.2, 0.7, -0.1]) = -0.37

Word Embeddings

Two central ideas:

1. We can represent things as vectors

Jay	-0.4	0.8	0.5	-0.2	0.3

2. We can easily calculate how similar vectors are to each other:

The people most similar to Jay are:

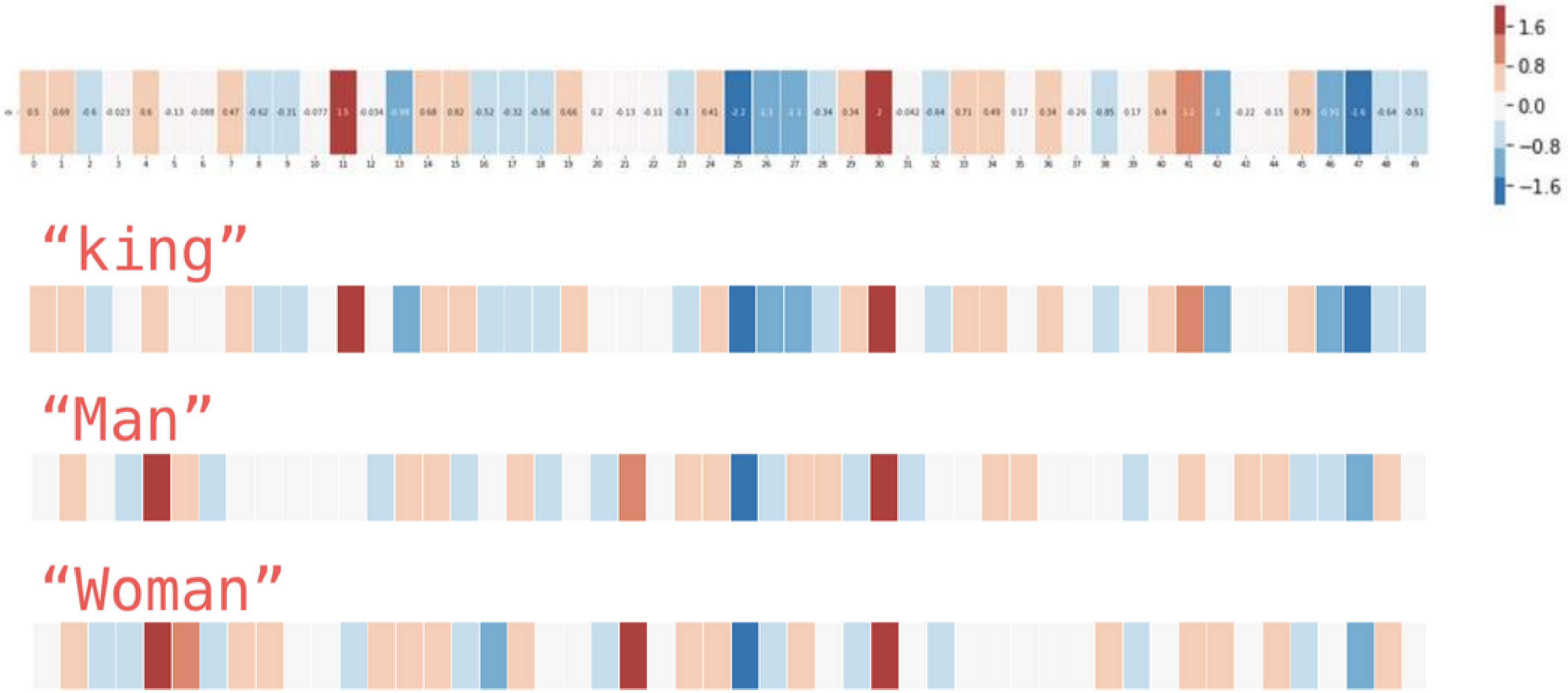
	cosine_similarity ▼
Person #1	0.86
Person #2	0.5
Person #3	-0.20

Word Embeddings

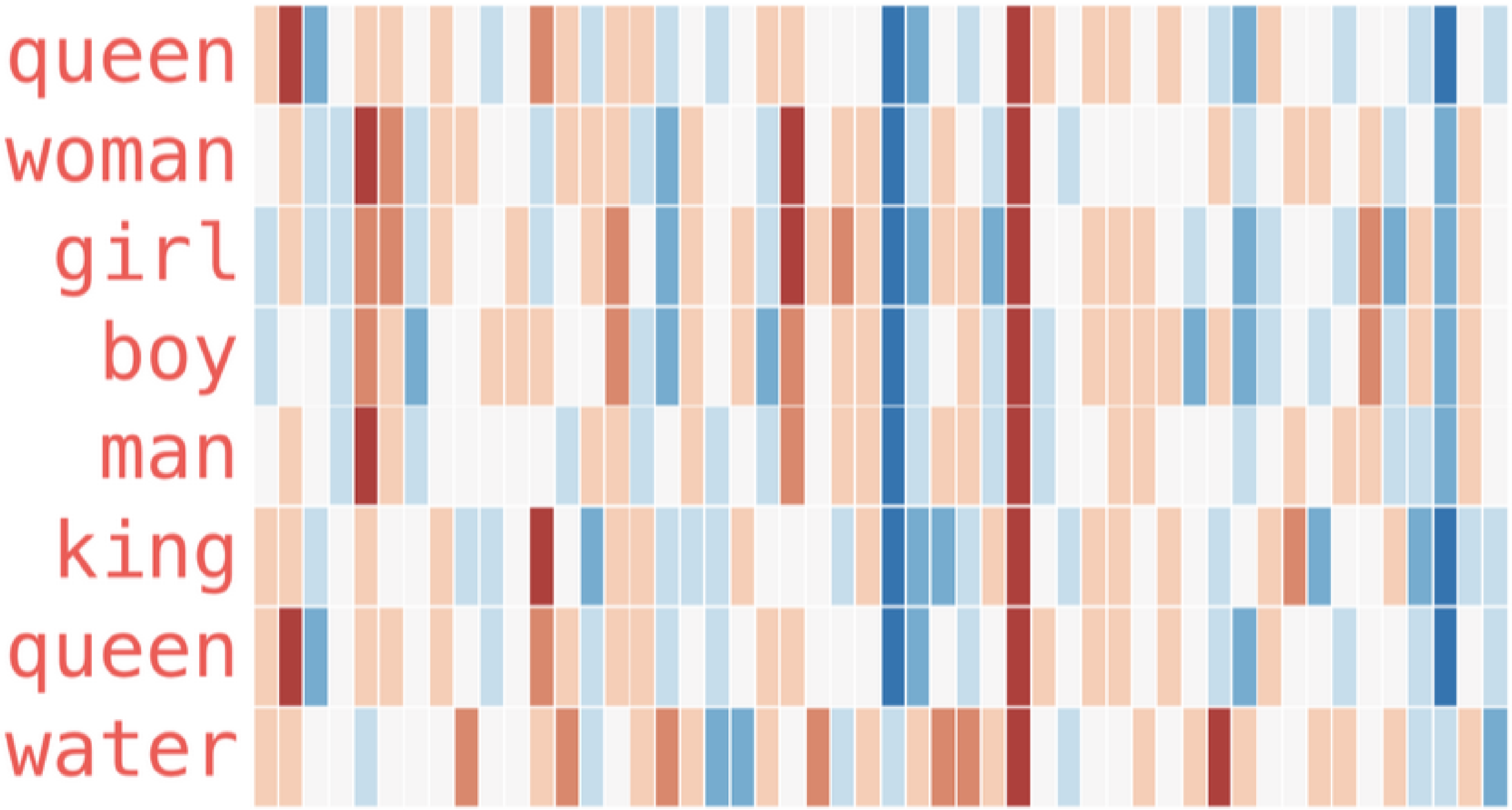
UNIVERSIDAD EAFIT
Escuela de Ciencias Aplicadas e Ingeniería

King (GloVe vector trained on Wikipedia):

```
[ 0.50451 , 0.68607 , -0.59517 , -0.022801, 0.60046 , -0.13498 , -0.08813 , 0.47377 , -0.61798 , -0.31012 ,
-0.076666, 1.493 , -0.034189, -0.98173 , 0.68229 , 0.81722 , -0.51874 , -0.31503 , -0.55809 , 0.66421 , 0.1961
, -0.13495 , -0.11476 , -0.30344 , 0.41177 , -2.223 , -1.0756 , -1.0783 , -0.34354 , 0.33505 , 1.9927 ,
-0.04234 , -0.64319 , 0.71125 , 0.49159 , 0.16754 , 0.34344 , -0.25663 , -0.8523 , 0.1661 , 0.40102 , 1.1685 ,
```



Word Embeddings



Word Embeddings

•**Word Embedding Analogies**: Demonstrates how word embeddings capture semantic relationships.

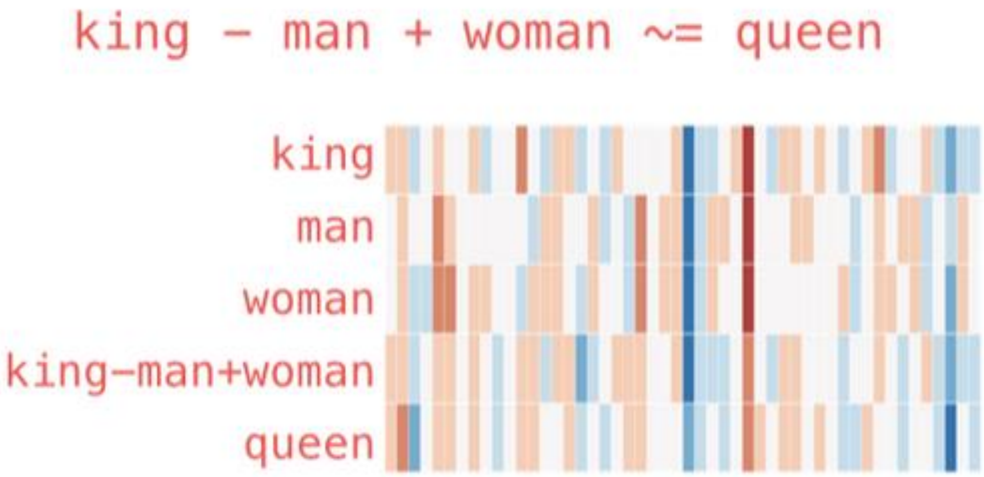
•**Famous Example**: "king" - "man" + "woman" ≈ "queen".

•**Key Insight**: Embeddings encode meaning in a way that allows vector arithmetic to reflect real-world relationships.

•**Implication**: Shows how models learn contextual and relational information beyond individual words.

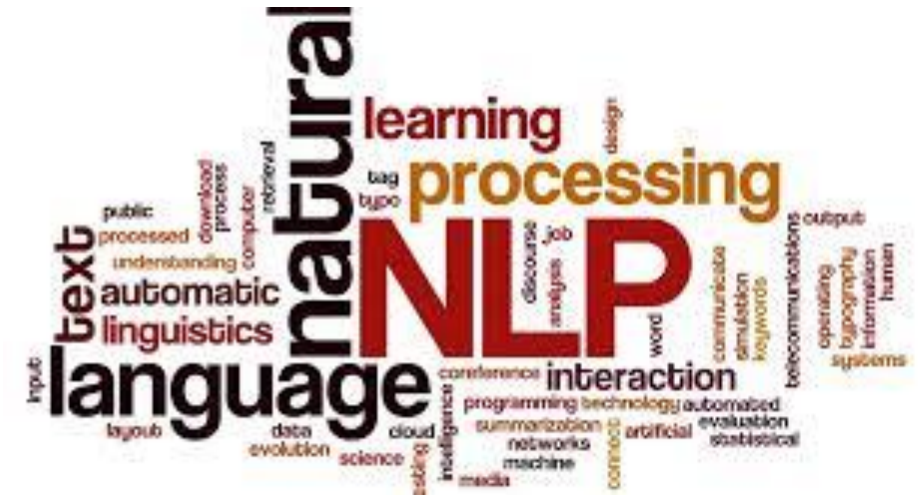
```
model.most_similar(positive=["king","woman"], negative=["man"])

[('queen', 0.8523603677749634),
 ('throne', 0.7664333581924438),
 ('prince', 0.7592144012451172),
 ('daughter', 0.7473883032798767),
 ('elizabeth', 0.7460219860076904),
 ('princess', 0.7424570322036743),
 ('kingdom', 0.7337411642074585),
 ('monarch', 0.721449077129364),
 ('eldest', 0.7184862494468689),
 ('widow', 0.7099430561065674)]
```



Language Modeling

A **language model** is a statistical or probabilistic model that learns the likelihood of word sequences. It is used to predict the next word in a sentence or generate text based on learned patterns.

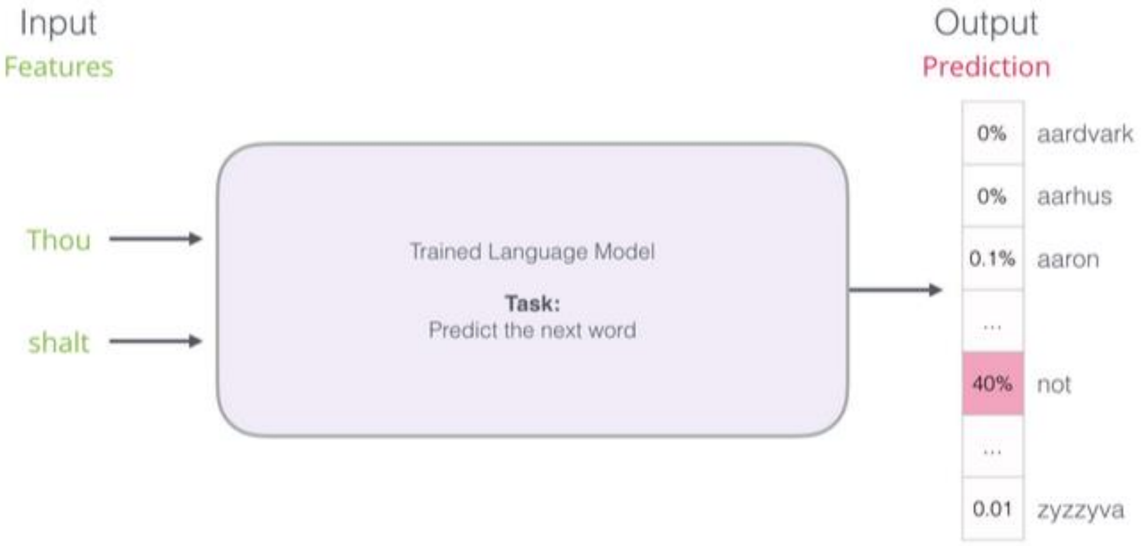


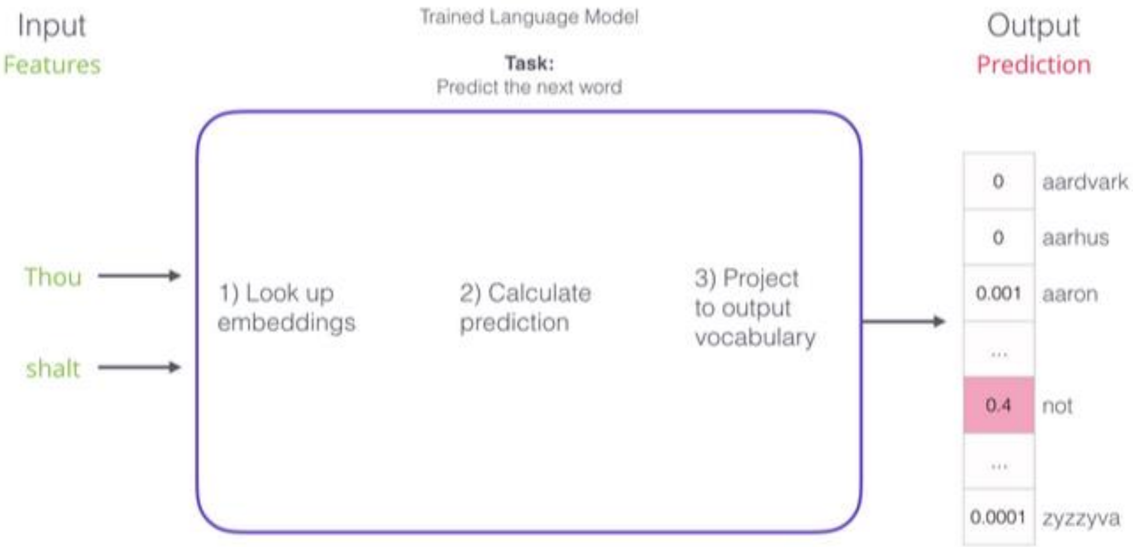
Limitations of Traditional Language Models

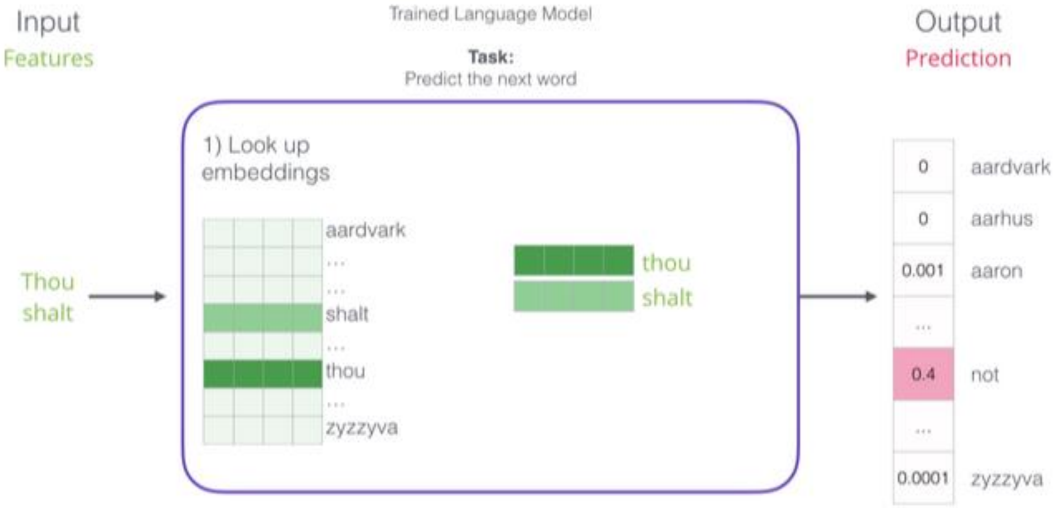
- Traditional models process text **linearly**, which may fail to capture deeper, multi-dimensional word relationships.
- Example from *God Emperor of Dune*: Language imposes a **linear structure**, but meaning can be **non-linear and contextual**.

This is not a mechanical universe. The linear progression of events is imposed by the observer. Cause and effect? That's not it at all. ***The prophet utters fateful words.*** *You glimpse a thing "destined to occur." But the prophetic instant releases something of infinite portent and power. The universe undergoes a ghostly shift." ~God Emperor of Dune*

Next word prediction

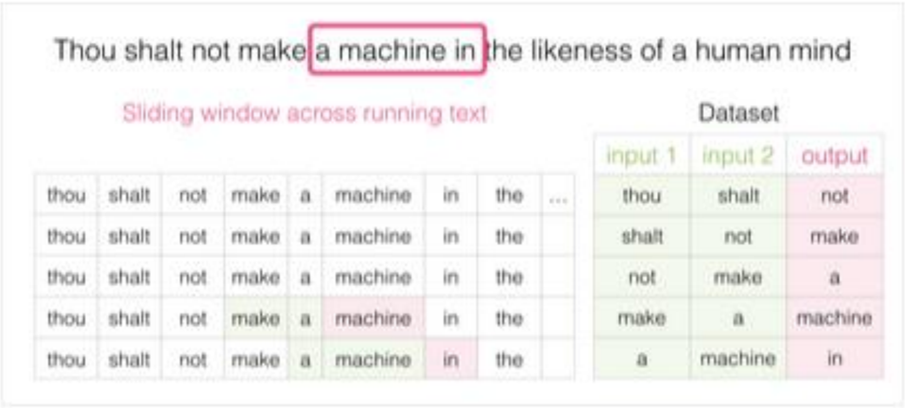


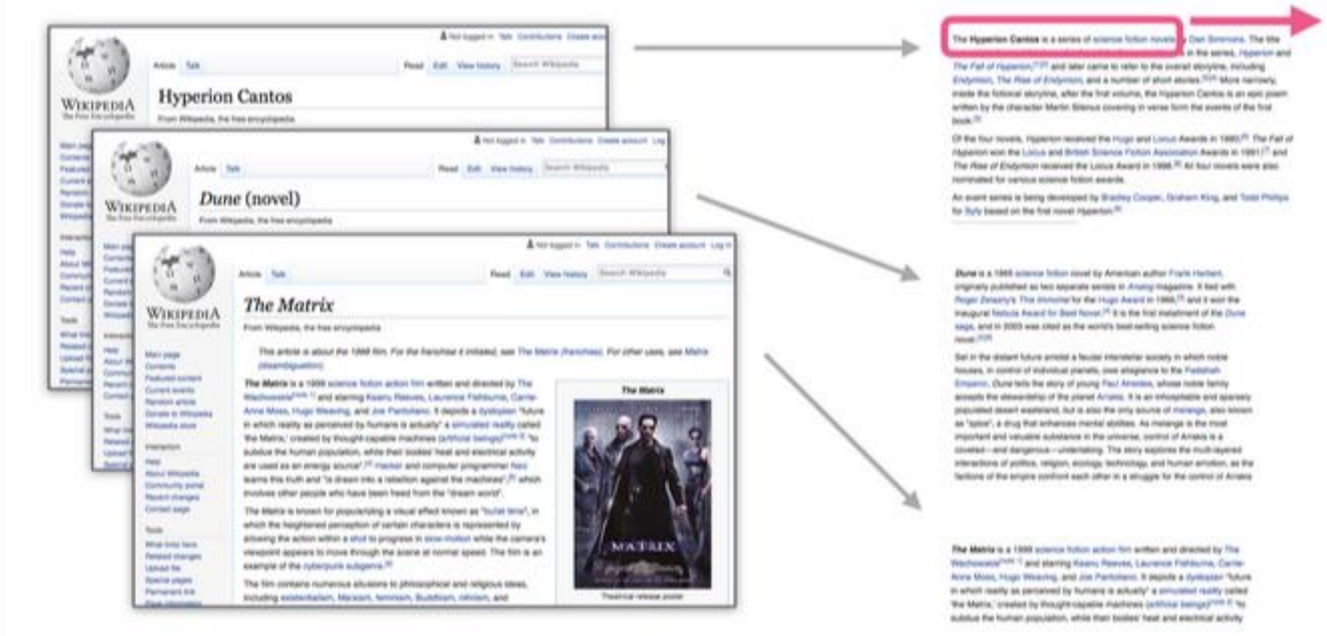




Language Model Training

- Learn the context in which a particular word appears
- Words that appear in similar contexts have similar embeddings
- Unlike other applications, in Language Models we have a lot of text to train





Continuos Bag of Words

Jay was hit by a _____   Jay was hit by a _____ bus

- Instead of only looking two words before the target word, we can also look at two words after it.

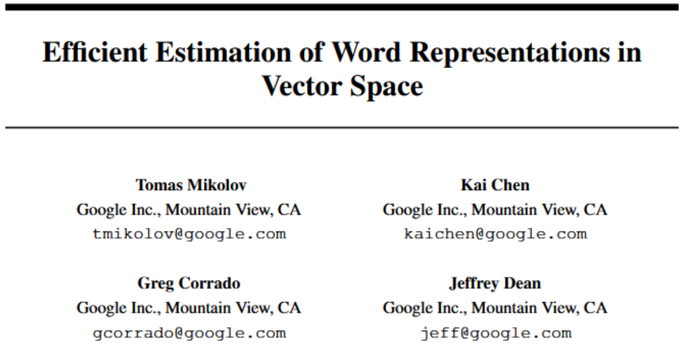
Jay was hit by a _____ bus in...

by	a	red	bus	in

- If we do this, the dataset we're virtually building and training the model against would look like this:

input 1	input 2	input 3	input 4	output
by	a	bus	in	red

Continuos Bag of Words - CBoW


Efficient Estimation of Word Representations in Vector Space

Tomas Mikolov
Google Inc., Mountain View, CA
tmikolov@google.com

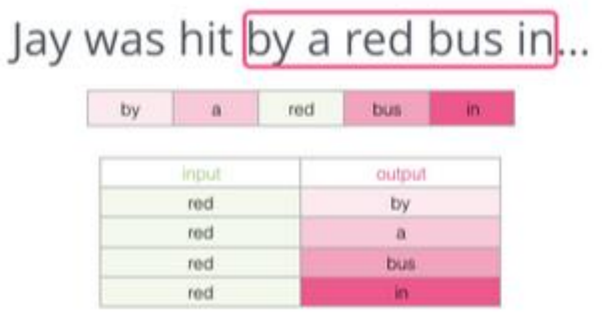
Kai Chen
Google Inc., Mountain View, CA
kaichen@google.com

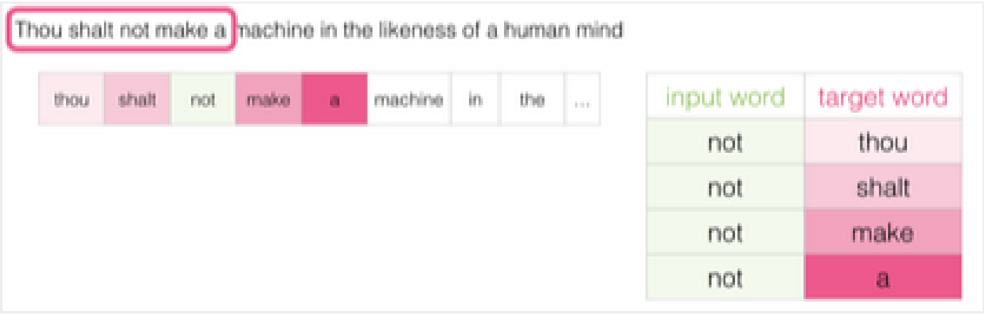
Greg Corrado
Google Inc., Mountain View, CA
gcorrado@google.com

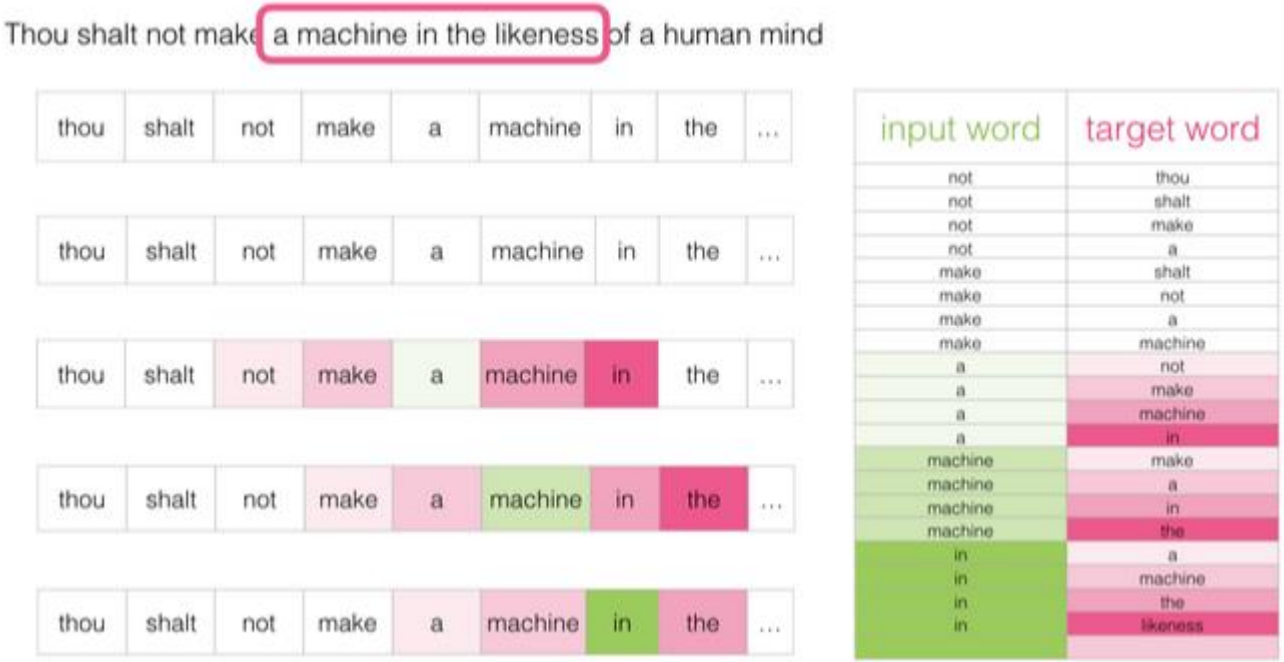
Jeffrey Dean
Google Inc., Mountain View, CA
jeff@google.com

Skipgram

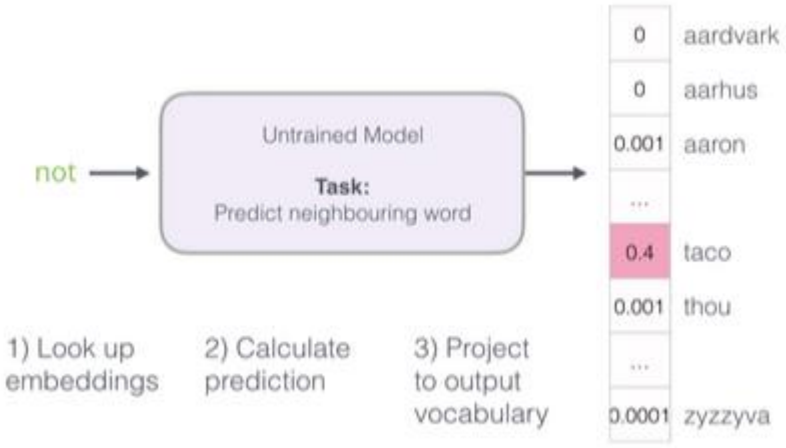
- **Skip-Gram Architecture**: Instead of predicting a word based on its context, Skip-Gram predicts **neighboring words** given a **target word**.
- **Sliding Window Approach**: The model moves through the text, learning which words frequently appear around others.
- **Key Benefit**: Works well with **smaller datasets** and captures rare word relationships.

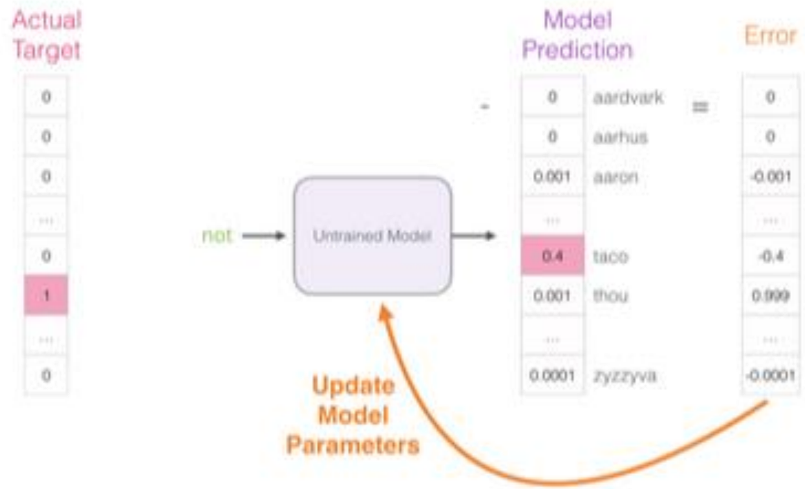






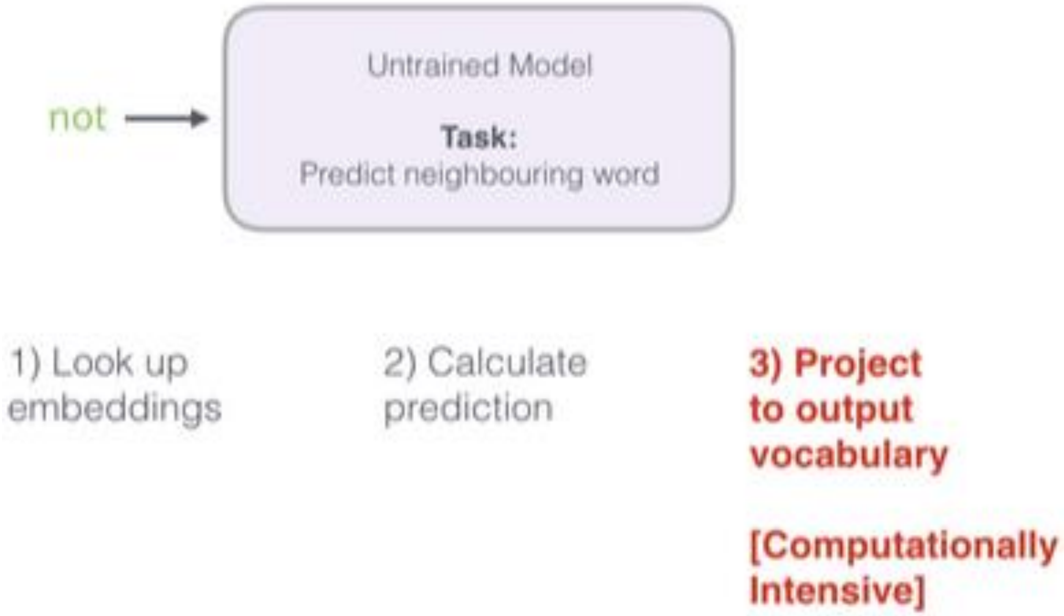
Trainig process





Negative sampling

- Recall the three steps of how this neural language model calculates its prediction:



- The third step is computationally expensive due to the size of the vocabulary

1. Generate high-quality word embeddings (Don't worry about next-word prediction).
2. Use these high-quality embeddings to train a language model (to do next-word prediction).

We'll focus on step 1.

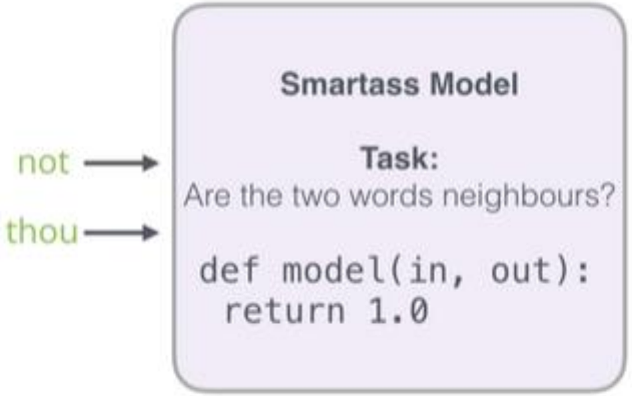
From:

To:

Negative sampling

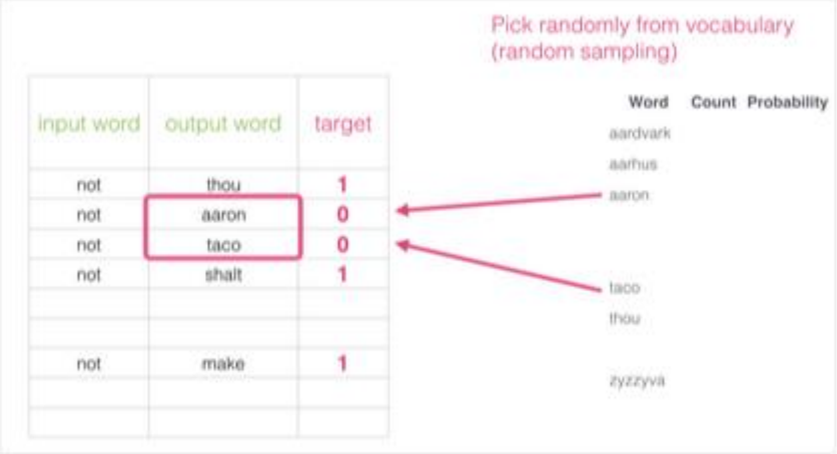
UNIVERSIDAD EAFIT
Escuela de Ciencias Aplicadas e Ingeniería

- We need to introduce *negative samples* to our dataset – samples of words that are not neighbors.
- We use random words from our vocabulary



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



Skipgram with negative sampling

UNIVERSIDAD EAFIT | Escuela de Ciencias Aplicadas e Ingeniería

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

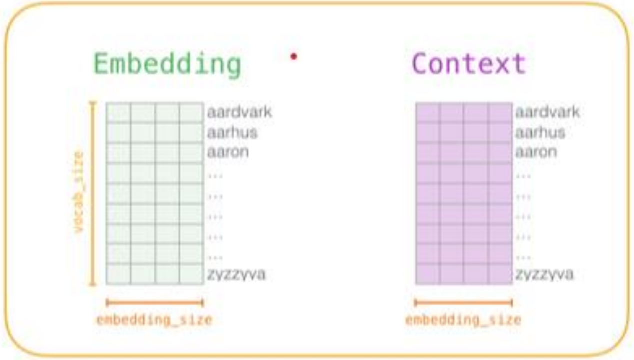
Noise-contrastive estimation: A new estimation principle for unnormalized statistical models

Michael Gutmann
Dept of Computer Science
and HIIT, University of Helsinki
michael.gutmann@helsinki.fi

Aapo Hyvärinen
Dept of Mathematics & Statistics, Dept of Computer
Science and HIIT, University of Helsinki
aapo.hyvarinen@helsinki.fi

Word2vec training process

- We create two matrices – an Embedding matrix and a Context matrix. These two matrices have an embedding for each word in our vocabulary. At the start of the training process we initialize both matrices randomly.



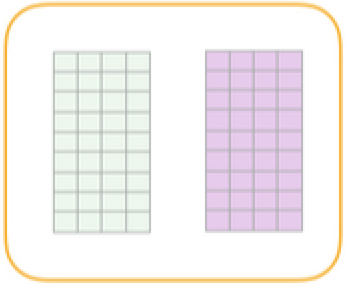
vocab_size = 10.000
embedding_size = 300

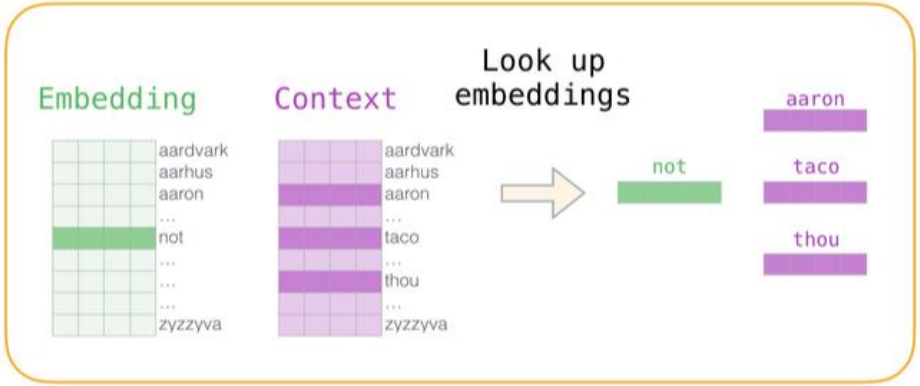
- In each training step, we take one positive example and its associated negative examples.

dataset

input word	output word	target
not	thou	1
not	aaron	0
not	taco	0
not	shalt	1
not	mango	0
not	finglonger	0
not	make	1
not	plumbus	0
...	...	...

model

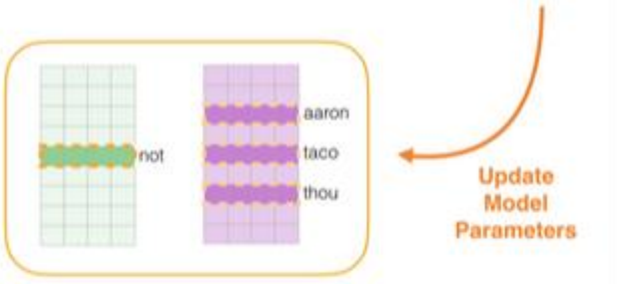




Word2vec training process

- **Compute Similarity**: Take the **dot product** between the input word embedding and each context word embedding.
- **Apply Sigmoid Function**: Convert similarity scores into **probabilities** between **0 and 1**.
- **Compare with Target**: Measure how well the model’s predictions match the **actual context words**.
- **Compute Error**: Calculate **error = target - sigmoid_scores** for each word.
- **Update Embeddings**: Adjust the embeddings using the error to improve **word representation**.
- **Repeat Process**: Move to the **next training sample** and perform the same steps.

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



Word2vec cost function

- **For each positive (correct) word pair**, the model should output a probability **close to 1.**
- **For each negative (randomly sampled) word pair**, the model should output a probability **close to 0.**

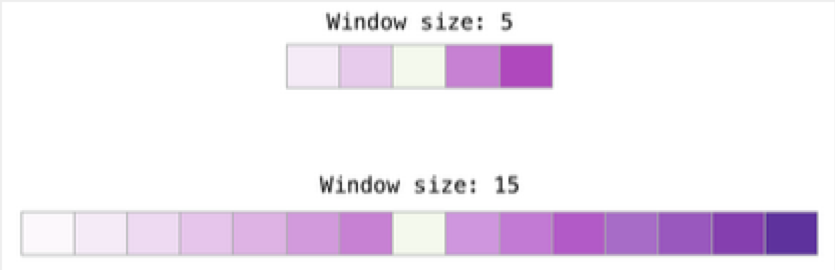
The loss function for a single word pair is:

$$L = -\sum_{(c,w)\in D} \left[y \log \sigma(\mathbf{v}_c \cdot \mathbf{v}_w) + (1-y) \log(1 - \sigma(\mathbf{v}_c \cdot \mathbf{v}_w)) \right]$$

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

- y is **1** for positive (real) word-context pairs and **0** for negative (random) pairs.
- σ(x) is the **sigmoid function**, ensuring outputs are between **0 and 1**.

Parameters



Negative samples: 2

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

Negative samples: 5

input word	output word	target
make	shalt	1
make	aaron	0
make	taco	0
make	finglonger	0
make	plumbus	0
make	mango	0

- **Window Size in Word2Vec**: Determines how many words before and after the target word are considered.
- **Small Window (2-15 words)**: Produces embeddings where **high similarity means interchangeability** (e.g., "good" and "bad" may appear in similar contexts).
- **Large Window (15-50+ words)**: Captures **relatedness** rather than interchangeability (e.g., "car" and "road" are related but not interchangeable).
- **Annotation Matters**: The choice of window size depends on the task and may require manual tuning.
- **Gensim Default**: Uses a window size of **5** (five words before and after the target word).