



Módulo II: Modelos de Lenguaje y Etiquetado de Secuencia

.....

Dr. Gaddiel Desirena López

Procesamiento de Lenguaje Natural (NLP) 2021

I.- Vectores Palabra (Word Embeddings)

Vectores Palabra

- ▶ Un componente importante en las redes neuronales para el lenguaje es el uso de una capa embebida.
- ▶ Un mapeo de símbolos discretos a vectores continuos.
- ▶ Al embeber una palabra, se transforman de símbolos distintos aislados en símbolos matemáticos objetos sobre los que se puede operar.
- ▶ La distancia entre vectores se puede equiparar a la distancia entre palabras.
- ▶ Esto facilita la generalización del comportamiento de una palabra a otra.

Vectores de distribución

- ▶ **Hipótesis distributiva** [Harris, 1954]: las palabras que aparecen en el mismo contexto tienden a tener significados similares.
- ▶ **Representaciones distributivas**: las palabras están representadas por **vectores de alta dimensión** según el contexto donde ocurren.

Matrices de contexto de palabras

- ▶ Los vectores de distribución se construyen a partir de matrices de contexto de palabras M .
- ▶ Cada celda (i, j) es un valor de asociación basado en la co-ocurrencia entre una **palabra de destino** w_i y una **contexto** c_j calculada a partir de un corpus de documentos.
- ▶ Los contextos se definen comúnmente como ventanas de palabras que rodean w_i .
- ▶ La longitud de la ventana k es un parámetro (entre 1 y 8 palabras en los lados izquierdo y derecho de w_i).
- ▶ Si el vocabulario de las palabras de destino y las palabras de contexto es el mismo, M tiene dimensionalidad $|\mathcal{V}| \times |\mathcal{V}|$.
- ▶ Mientras que es probable que las ventanas más cortas capturen **información sintáctica** (por ejemplo, POS), es más probable que las ventanas más largas capturen similitudes temáticas [Goldberg, 2016, Jurafsky and Martin, 2008].

Distributional Vectors with context windows of size 1

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

⁰Ejemplo tomado de:

<http://cs224d.stanford.edu/lectures/CS224d-Lecture2.pdf>

Matrices de contexto de palabras

Las asociaciones entre palabras y contextos se pueden calcular utilizando diferentes enfoques:

1. Recuentos de co-ocurrencia.
2. Información mutua puntual positiva (PPMI).

El más común de todos según [Jurafsky and Martin, 2008] es PPMI.

Los métodos de distribución también se denominan métodos basados en recuento.

- ▶ PMI calcula el logaritmo de la probabilidad de que los pares palabra-contexto ocurran juntos sobre la probabilidad de que sean independientes.

$$\text{PMI}(w, c) = \log_2 \left(\frac{P(w, c)}{P(w)P(c)} \right) = \log_2 \left(\frac{\text{count}(w, c) \times |D|}{\text{count}(w) \times \text{count}(c)} \right) \quad (1)$$

- ▶ Los valores negativos del PMI sugieren que el par coexiste con menos frecuencia que el azar.
- ▶ Estas estimaciones no son fiables a menos que los recuentos se calculen a partir de un corpus muy grande [Jurafsky and Martin, 2008].
- ▶ PPMI corrige este problema reemplazando los valores negativos por cero:

$$\text{PPMI}(w, c) = \max(0, \text{PMI}(w, c)) \quad (2)$$

Vectores distribuidos o Wordembeddings

- ▶ Los vectores de distribución basados en conteo aumentan de tamaño con el vocabulario, es decir, pueden tener una dimensionalidad muy alta.
- ▶ El almacenamiento explícito de la matriz de co-ocurrencia puede consumir mucha memoria.
- ▶ Algunos modelos de clasificación no se adaptan bien a datos de gran dimensión.
- ▶ La comunidad de redes neuronales prefiere usar **representaciones distribuidas**¹ o **wordembeddings**.
- ▶ Los **Wordembeddings** son vectores de palabras densas continuas de baja dimensión entrenados a partir de corpus de documentos usando **redes neuronales**.
- ▶ Las dimensiones no se pueden interpretar directamente, es decir, representan características latentes de la palabra, “ con suerte capturando propiedades sintácticas y semánticas útiles ” cite turian2010word.
- ▶ Se han convertido en un componente crucial de las arquitecturas de redes neuronales para la PNL.

¹Idea: El significado de la palabra está “ distribuido ” sobre una combinación de dimensiones.

Vectores distribuidos o Wordembeddings (2)

- ▶ Hay dos enfoques principales para obtener incrustaciones de palabras:
 1. capas embebidas: uso de una capa embebida en una arquitectura de red neuronal específica de la tarea entrenada a partir de ejemplos etiquetados (por ejemplo, análisis de sentimientos).
 2. Pre-trained wordembeddings: crear una tarea predictiva auxiliar a partir de un corpus sin etiquetar (por ejemplo, predecir la siguiente palabra) en la que los wordembeddings surgirán naturalmente de la arquitectura de la red neuronal.
- ▶ Estos enfoques también se pueden combinar: se puede inicializar una capa embebida de una red neuronal específica de la tarea con los wordembeddings previamente entrenadas obtenidas con el segundo enfoque.

Vectores distribuidos o Wordembeddings (2)

- ▶ Los modelos más populares basados en el segundo enfoque son skip-gram [Mikolov et al., 2013], bolsa de palabras continua [Mikolov et al., 2013] y Glove [Pennington et al., 2014].
- ▶ Los Wordembeddings han demostrado ser más poderosas que los enfoques distributivos en muchas tareas de PNL [Baroni et al., 2014].
- ▶ En [Amir et al., 2015], se usaron como **características** en un modelo de regresión para determinar la asociación entre palabras de Twitter y **sentimientos positivos**.

Wordembeddings

Some basic applications of word embeddings

Wordembeddings

Some basic applications of word embeddings



Semantic analogies
and similarity

Some basic applications of word embeddings

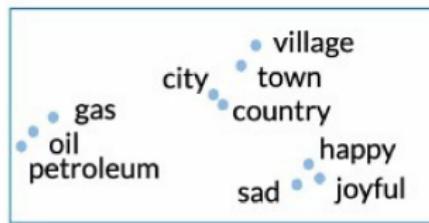


Semantic analogies
and similarity



Sentiment analysis

Some basic applications of word embeddings



Semantic analogies
and similarity



Sentiment analysis



Classification of
customer feedback

Wordembeddings

Advanced applications of word embeddings



Machine translation

Advanced applications of word embeddings



Machine translation



Information extraction

Wordembeddings

Advanced applications of word embeddings



Machine translation



Information extraction



Question answering

Wordembeddings

Integers

Word	Number
a	1
able	2
about	3
...	...
hand	615
...	...
happy	621
...	...
zebra	1000

Integers

- + Simple
- Ordering: little semantic sense

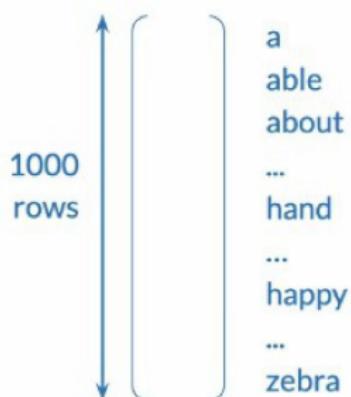
Integers

- + Simple
- Ordering: little semantic sense

hand < happy < zebra
615 621 1000
?! ?!

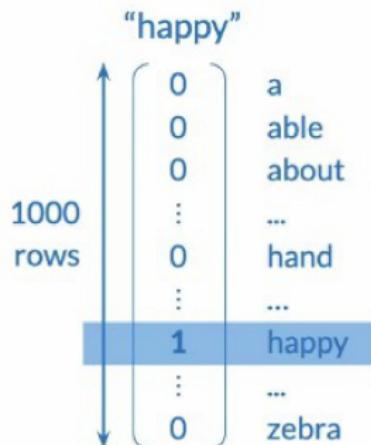
Wordembeddings

One-hot vectors



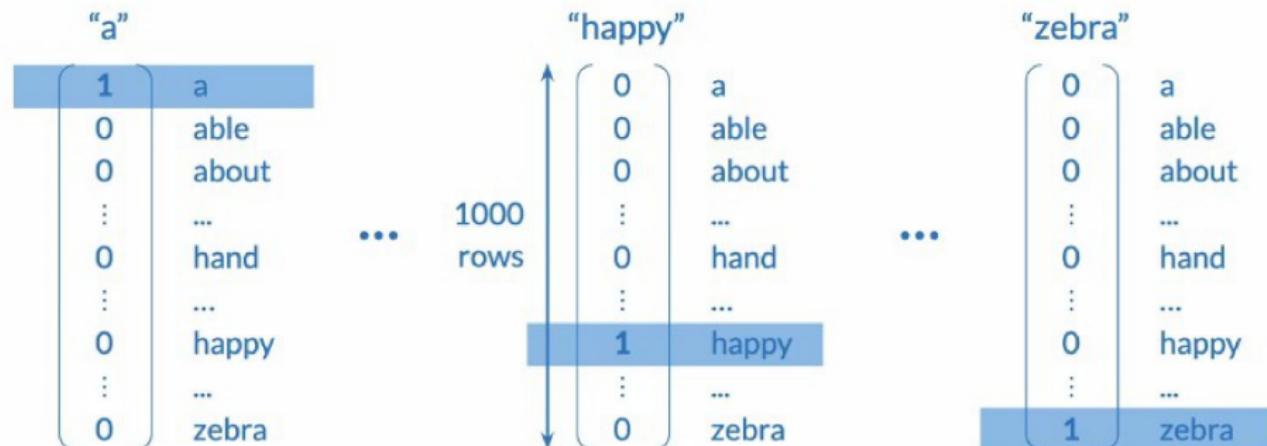
Wordembeddings

One-hot vectors



Wordembeddings

One-hot vectors



Wordembeddings

One-hot vectors

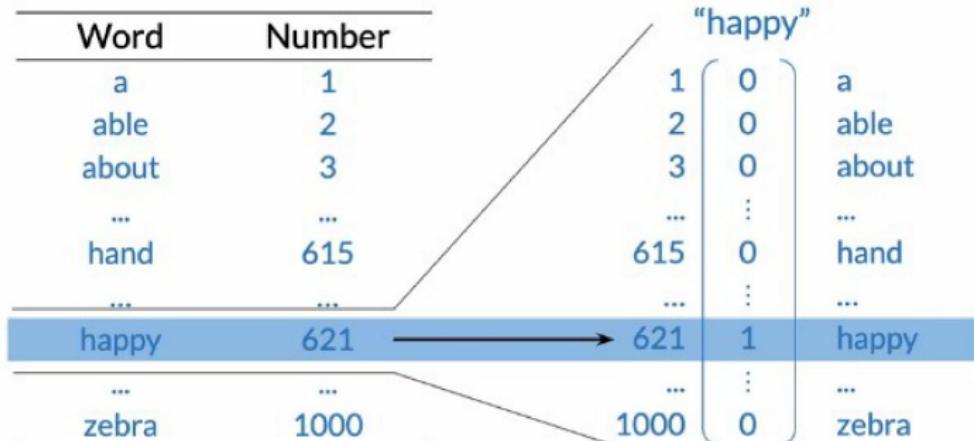
Word	Number
a	1
able	2
about	3
...	...
hand	615
...	...
happy	621
...	...
zebra	1000

"happy"

1 $\begin{pmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$ a
2 $\begin{pmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{pmatrix}$ able
3 $\begin{pmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$ about
...
615 $\begin{pmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$ hand
...
621 $\begin{pmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{pmatrix}$ happy
...
1000 $\begin{pmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$ zebra

Wordembeddings

One-hot vectors



One-hot vectors

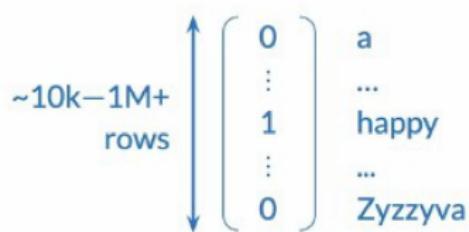
- + Simple
- + No implied ordering

One-hot vectors

- + Simple
- + No implied ordering
- Huge vectors

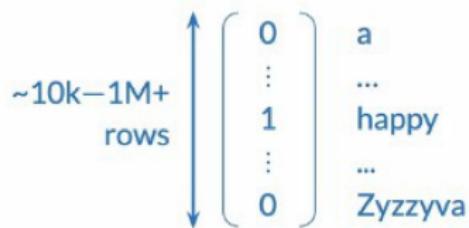
One-hot vectors

- + Simple
- + No implied ordering
- Huge vectors



One-hot vectors

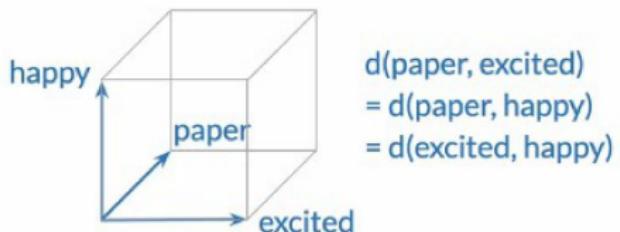
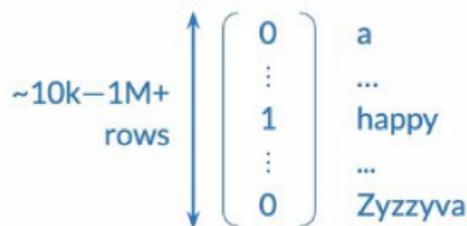
- + Simple
- + No implied ordering
- Huge vectors
- No embedded meaning



Wordembeddings

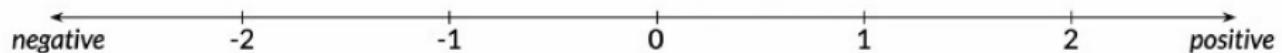
One-hot vectors

- + Simple
- + No implied ordering
- Huge vectors
- No embedded meaning



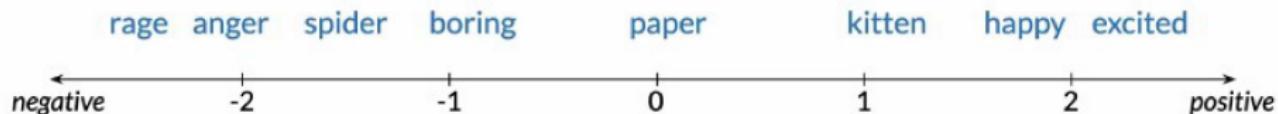
Wordembeddings

Meaning as vectors



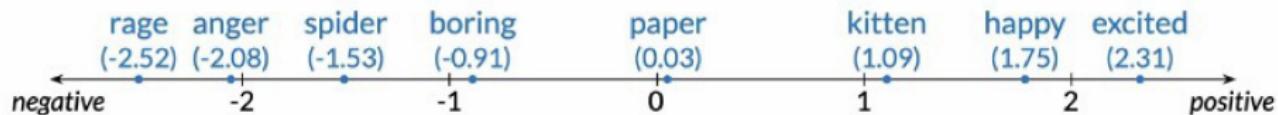
Wordembeddings

Meaning as vectors



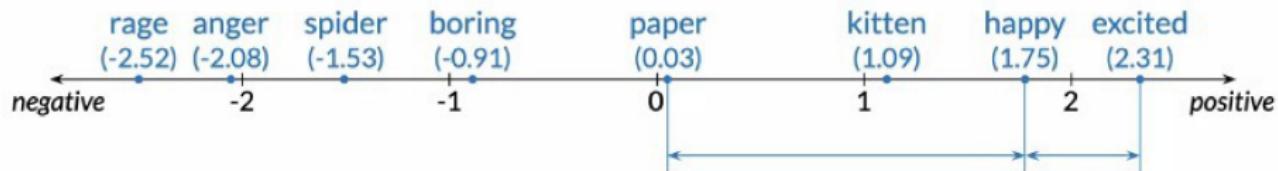
Wordembeddings

Meaning as vectors



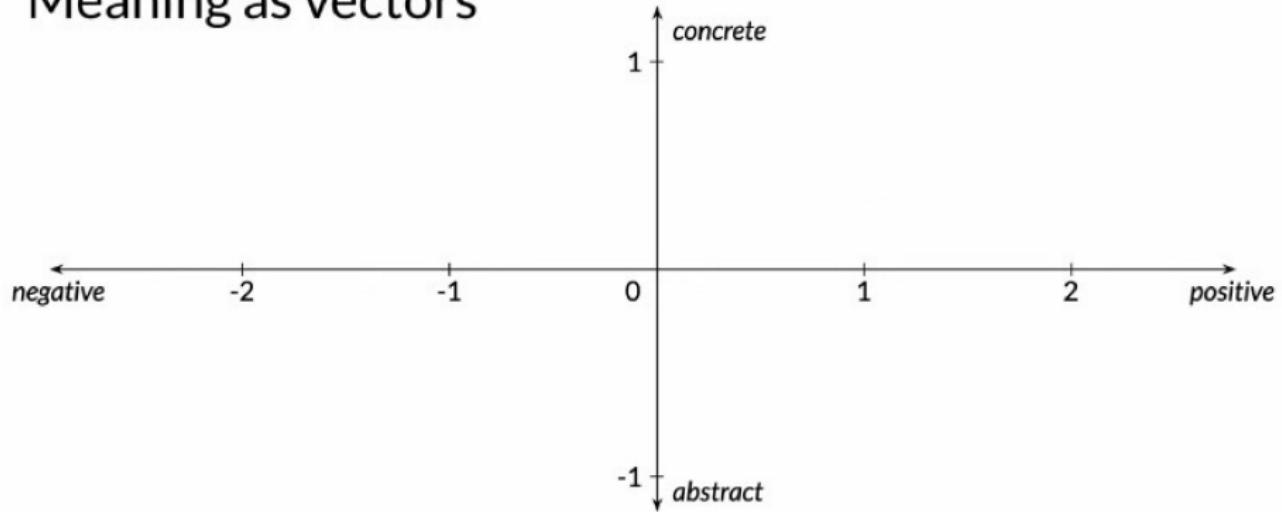
Wordembeddings

Meaning as vectors



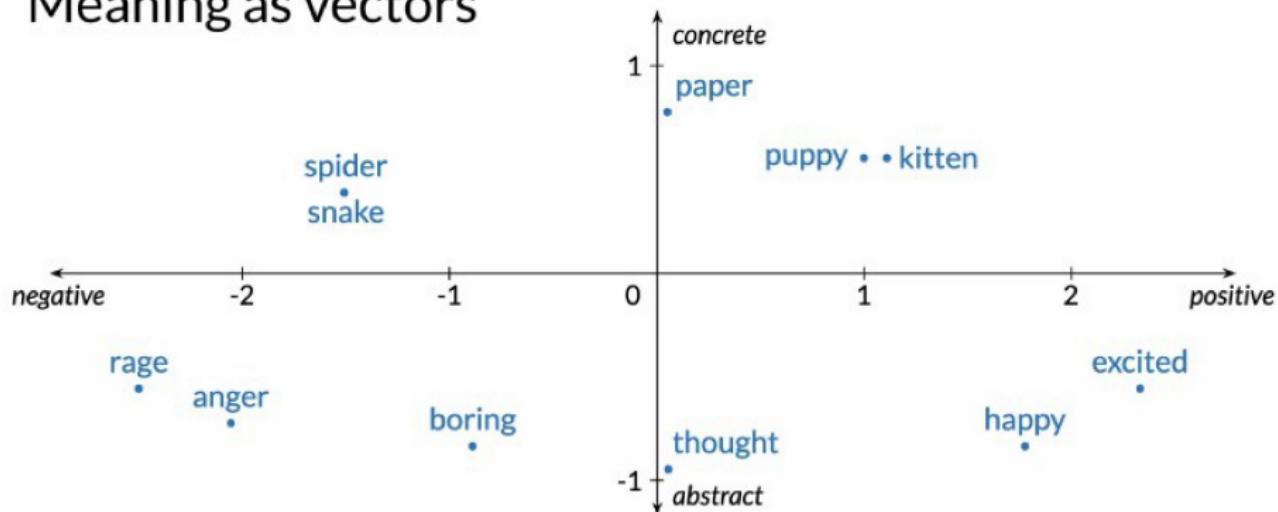
Wordembeddings

Meaning as vectors



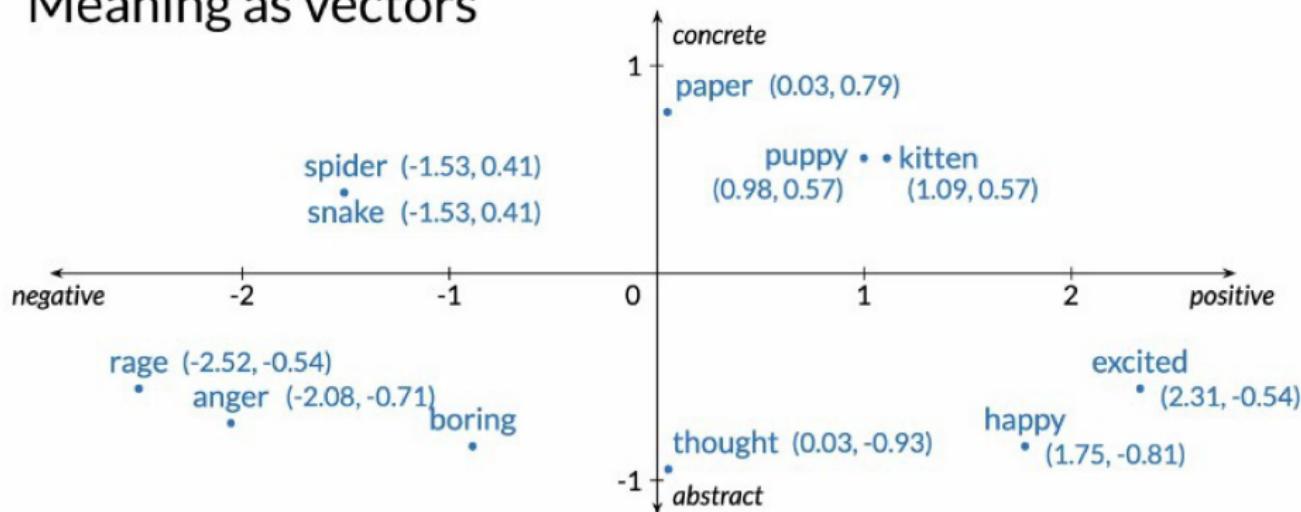
Wordembeddings

Meaning as vectors



Wordembeddings

Meaning as vectors



Word embedding vectors

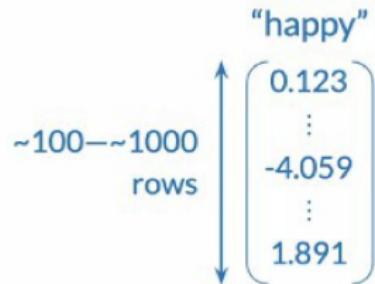
Word embedding vectors

- + Low dimension

Wordembeddings

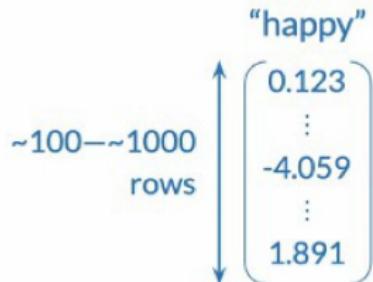
Word embedding vectors

- + Low dimension



Word embedding vectors

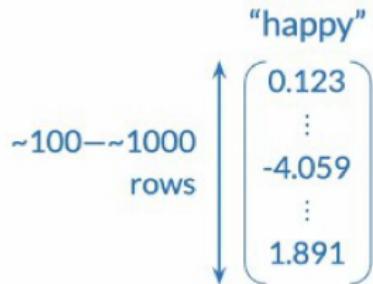
- + Low dimension
- + Embed meaning



Wordembeddings

Word embedding vectors

- + Low dimension
- + Embed meaning
 - o e.g. semantic distance

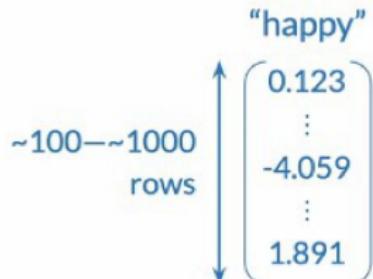


Wordembeddings

Word embedding vectors

- + Low dimension
- + Embed meaning
 - o e.g. semantic distance

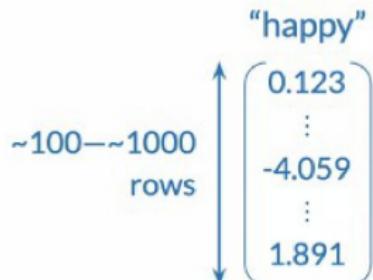
forest ≈ tree forest ≠ ticket



Wordembeddings

Word embedding vectors

- + Low dimension
- + Embed meaning
 - o e.g. semantic distance
 - forest = tree forest ≠ ticket
 - o e.g. analogies



Wordembeddings

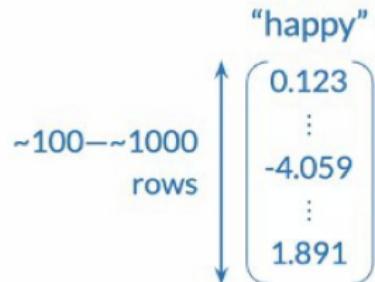
Word embedding vectors

- + Low dimension
- + Embed meaning
 - o e.g. semantic distance

$\text{forest} \approx \text{tree}$ $\text{forest} \neq \text{ticket}$

- o e.g. analogies

$\text{Paris:France} :: \text{Rome}:?$



Wordembeddings

Terminology

integers

one-hot vectors

word embedding vectors

Wordembeddings

Terminology

integers

word vectors

one-hot vectors

word embedding vectors

Wordembeddings

Terminology

integers

one-hot vectors

word vectors

word embedding vectors

“word vectors”

word embeddings

Word embedding process

Corpus

Embedding method

Wordembeddings

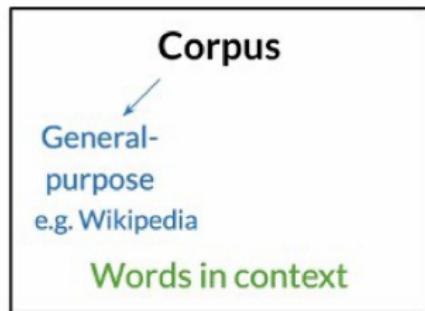
Word embedding process

Corpus

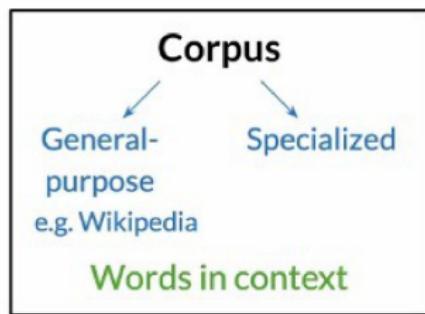
Embedding method

Words in context

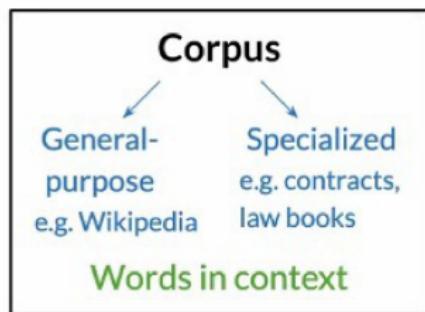
Word embedding process



Word embedding process

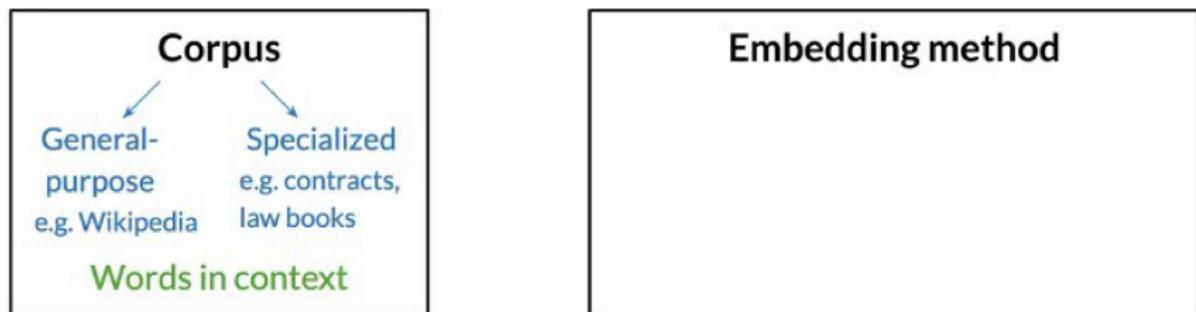


Word embedding process



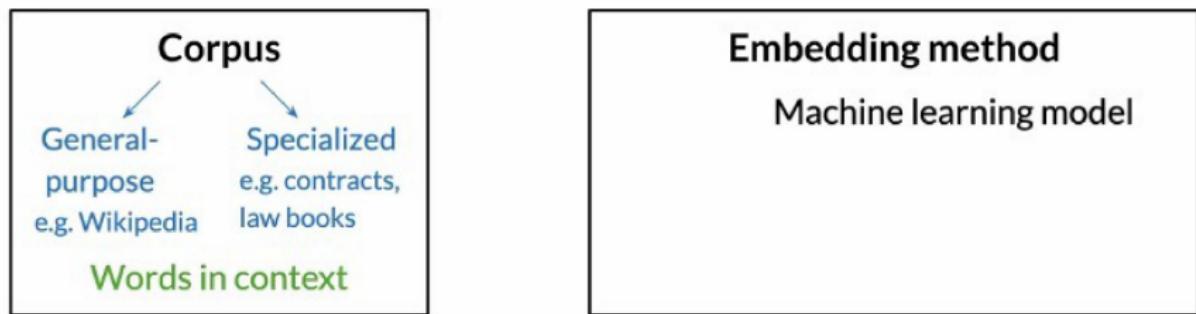
Wordembeddings

Word embedding process



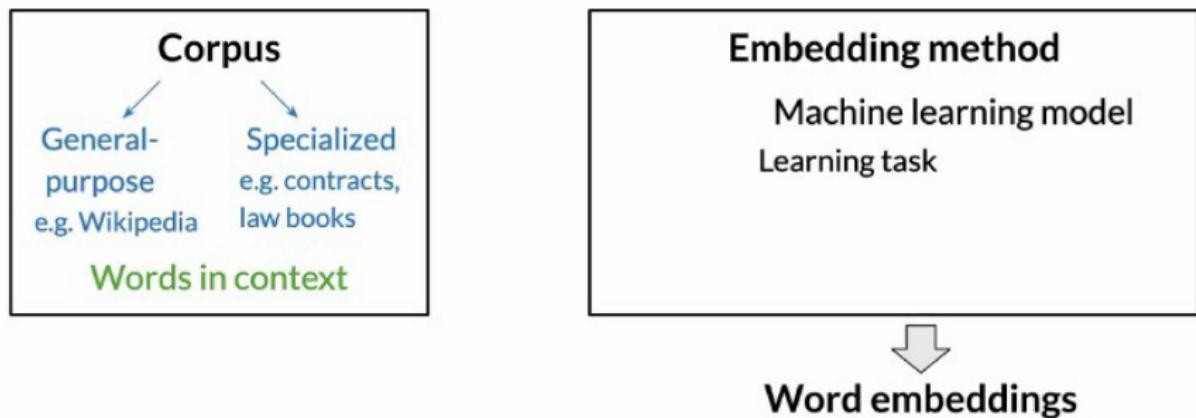
Wordembeddings

Word embedding process



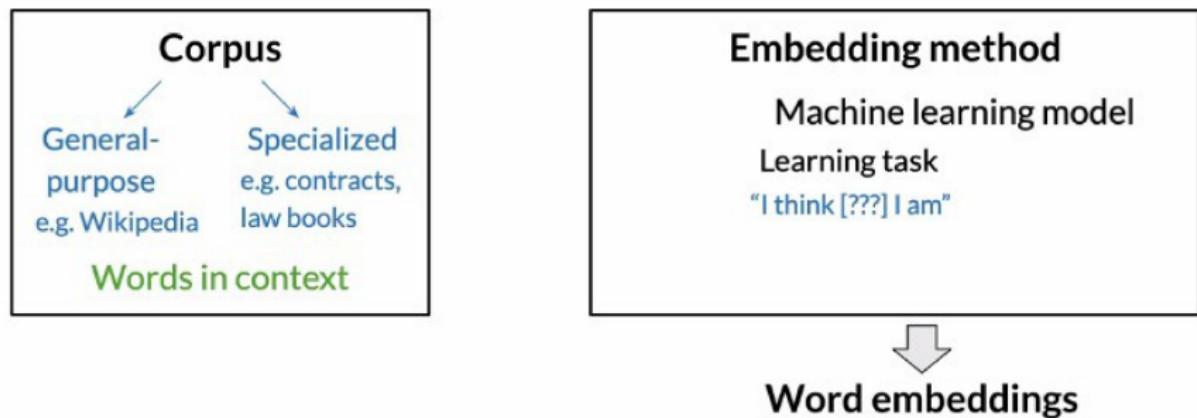
Wordembeddings

Word embedding process



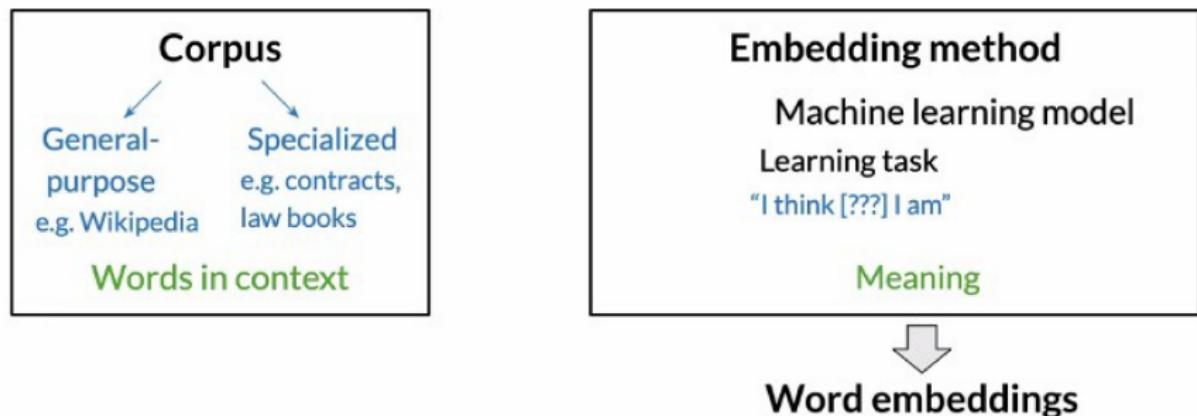
Wordembeddings

Word embedding process



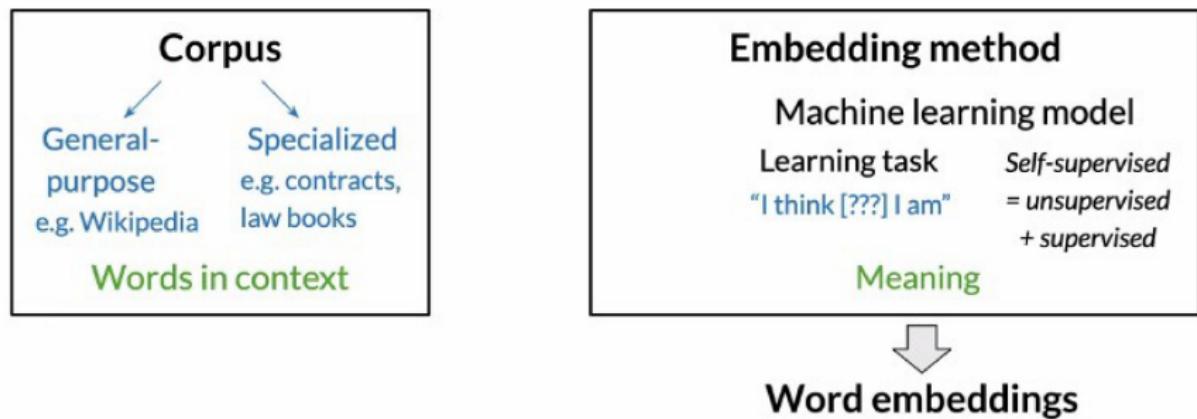
Wordembeddings

Word embedding process



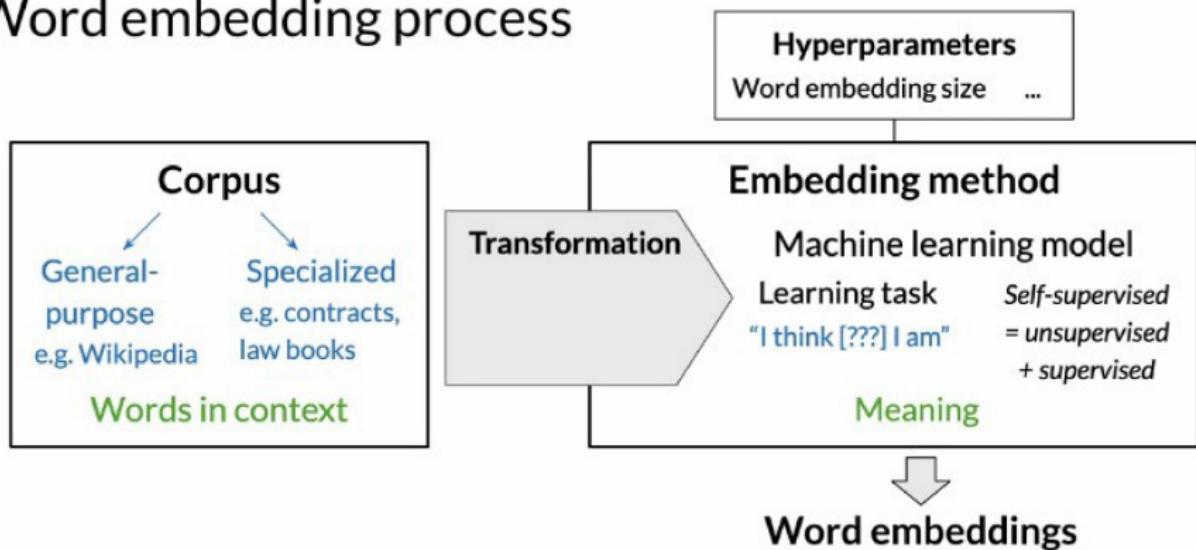
Wordembeddings

Word embedding process



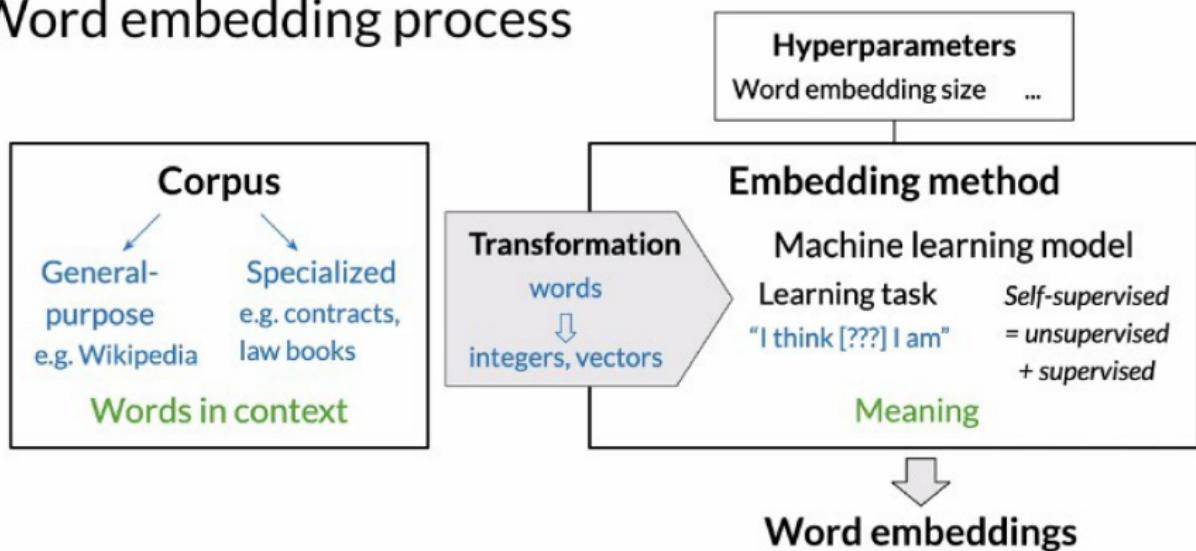
Wordembeddings

Word embedding process



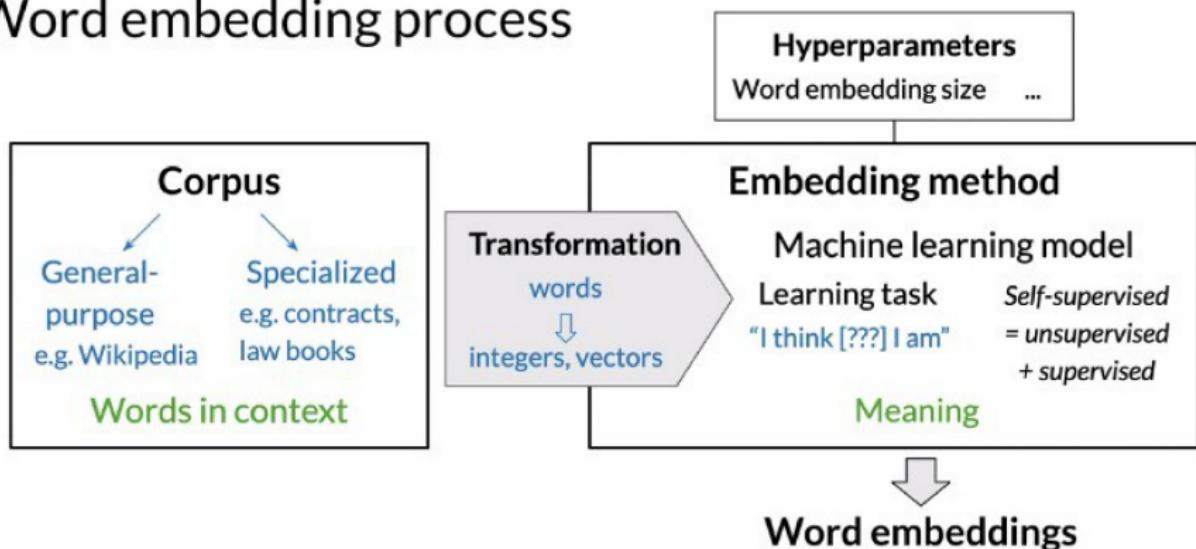
Wordembeddings

Word embedding process



Wordembeddings

Word embedding process



Basic word embedding methods

- word2vec (Google, 2013)
 - Continuous bag-of-words (CBOW)

Basic word embedding methods

- word2vec (Google, 2013)
 - Continuous bag-of-words (CBOW)
 - Continuous skip-gram / Skip-gram with negative sampling (SGNS)

Basic word embedding methods

- word2vec (Google, 2013)
 - Continuous bag-of-words (CBOW)
 - Continuous skip-gram / Skip-gram with negative sampling (SGNS)
- Global Vectors (GloVe) (Stanford, 2014)

Basic word embedding methods

- word2vec (Google, 2013)
 - Continuous bag-of-words (CBOW)
 - Continuous skip-gram / Skip-gram with negative sampling (SGNS)
- Global Vectors (GloVe) (Stanford, 2014)
- fastText (Facebook, 2016)
 - Supports out-of-vocabulary (OOV) words

Advanced word embedding methods

Deep learning, contextual embeddings

Advanced word embedding methods

Deep learning, contextual embeddings

- BERT (Google, 2018)

Advanced word embedding methods

Deep learning, contextual embeddings

- BERT (Google, 2018)
- ELMo (Allen Institute for AI, 2018)

Advanced word embedding methods

Deep learning, contextual embeddings

- BERT (Google, 2018)
- ELMo (Allen Institute for AI, 2018)
- GPT-2 (OpenAI, 2018)

Advanced word embedding methods

Deep learning, contextual embeddings

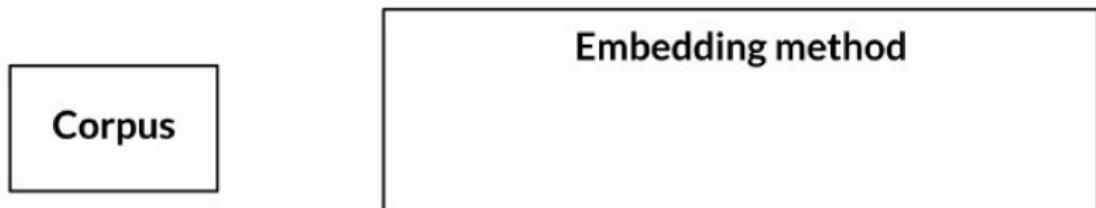
- BERT (Google, 2018)
 - ELMo (Allen Institute for AI, 2018)
 - GPT-2 (OpenAI, 2018)
- 
- Tunable pre-trained
models available

Wordembeddings CBOW

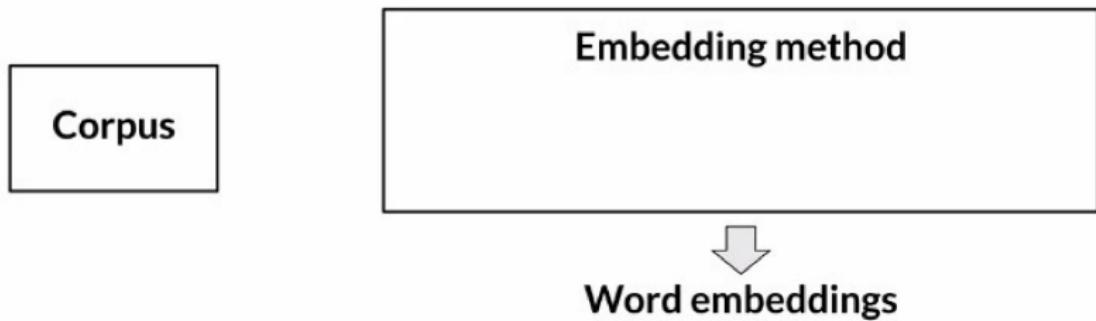
Continuous bag-of-words word embedding process

Wordembeddings

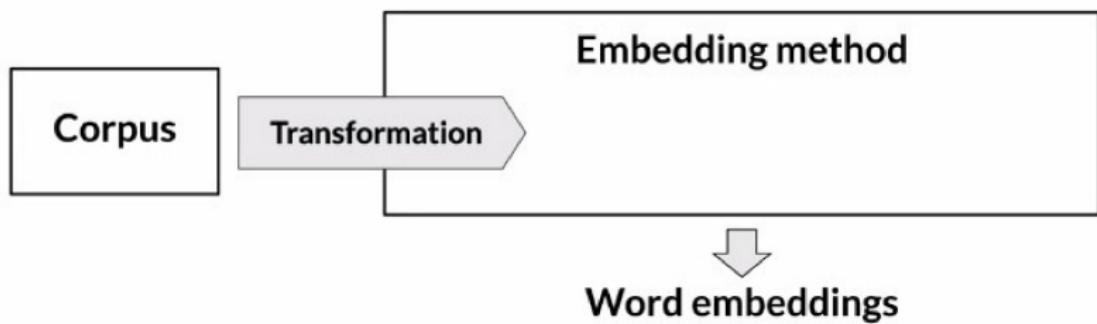
Continuous bag-of-words word embedding process



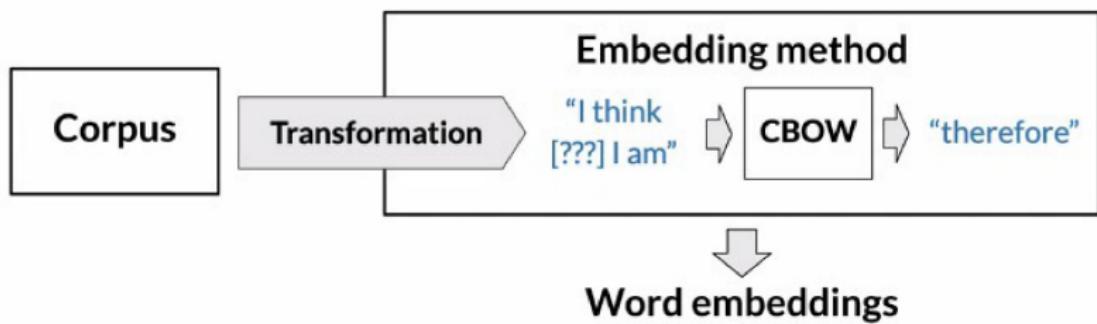
Continuous bag-of-words word embedding process



Continuous bag-of-words word embedding process



Continuous bag-of-words word embedding process



Wordembeddings



Center word prediction: rationale

Wordembeddings



Center word prediction: rationale

The little _____? _____ is barking

Wordembeddings



Center word prediction: rationale

The little _____ ? is barking



*dog
puppy
hound
terrier*

...

Wordembeddings



Creating a training example

I am happy because I am learning

Wordembeddings



Creating a training example

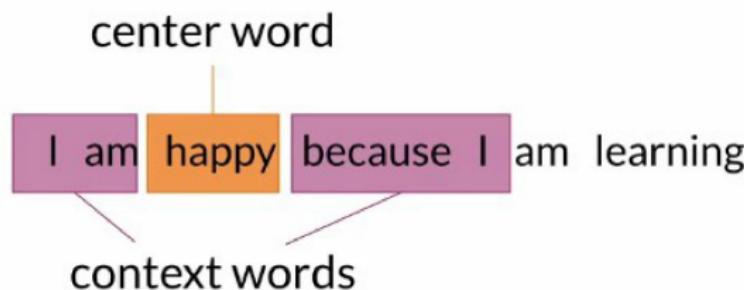
center word
I am **happy** because I am learning

The diagram shows a sentence: "I am happy because I am learning". The word "happy" is highlighted with a solid orange rectangle. Above this orange box, the text "center word" is written in black. A thin vertical yellow line connects the "center word" text to the center of the orange box.

Wordembeddings



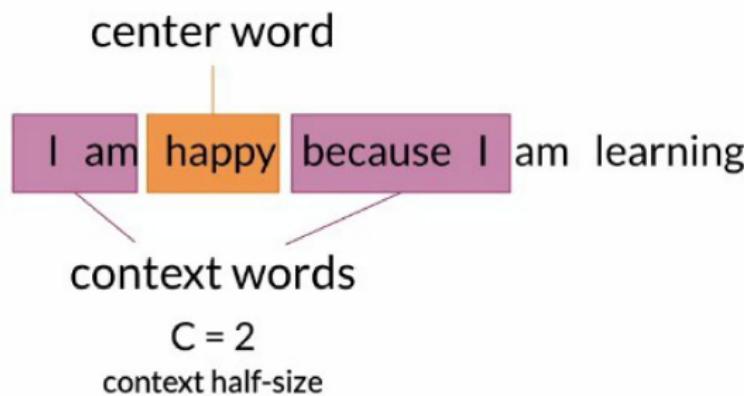
Creating a training example



Wordembeddings



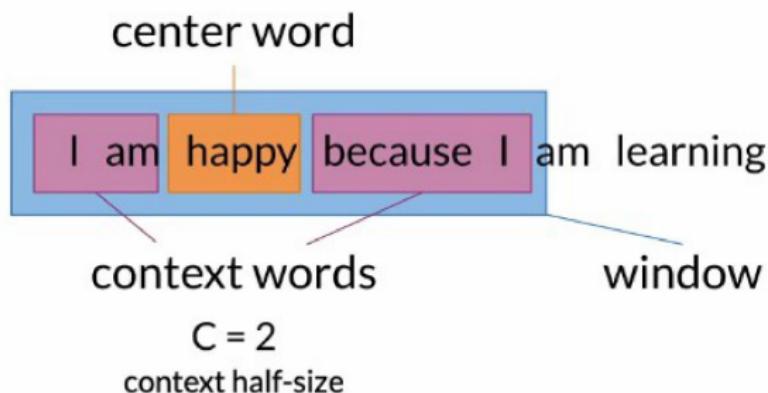
Creating a training example



Wordembeddings



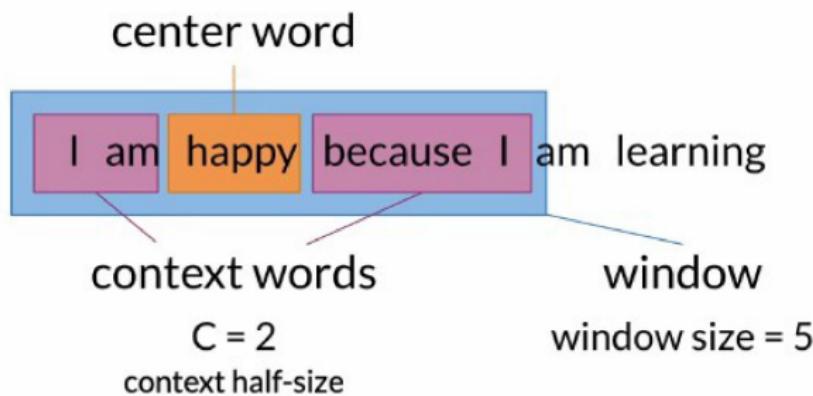
Creating a training example



Wordembeddings

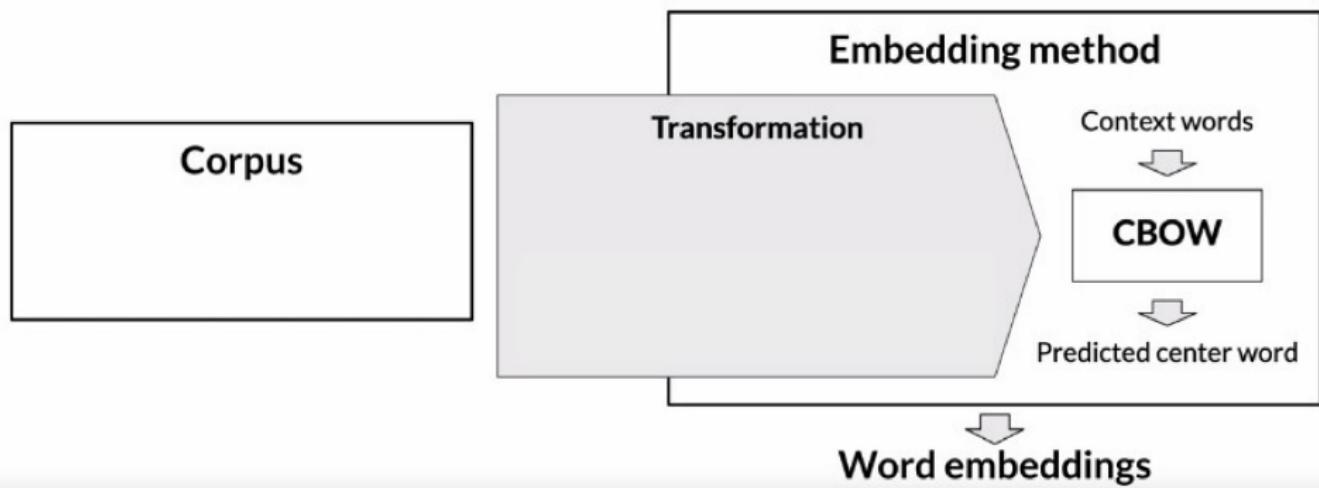


Creating a training example



Wordembeddings

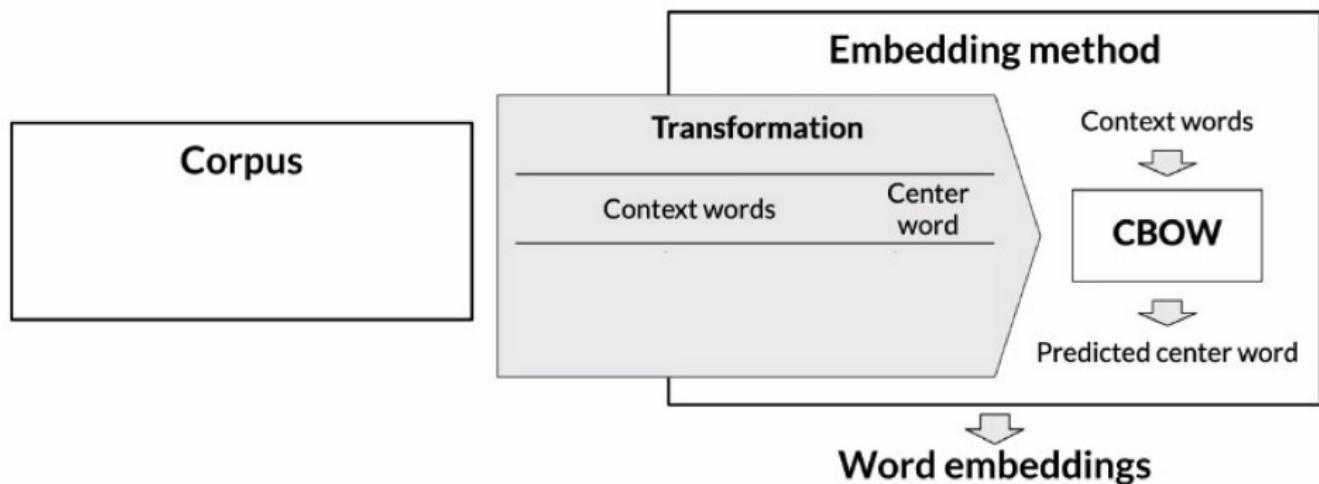
From corpus to training



Wordembeddings

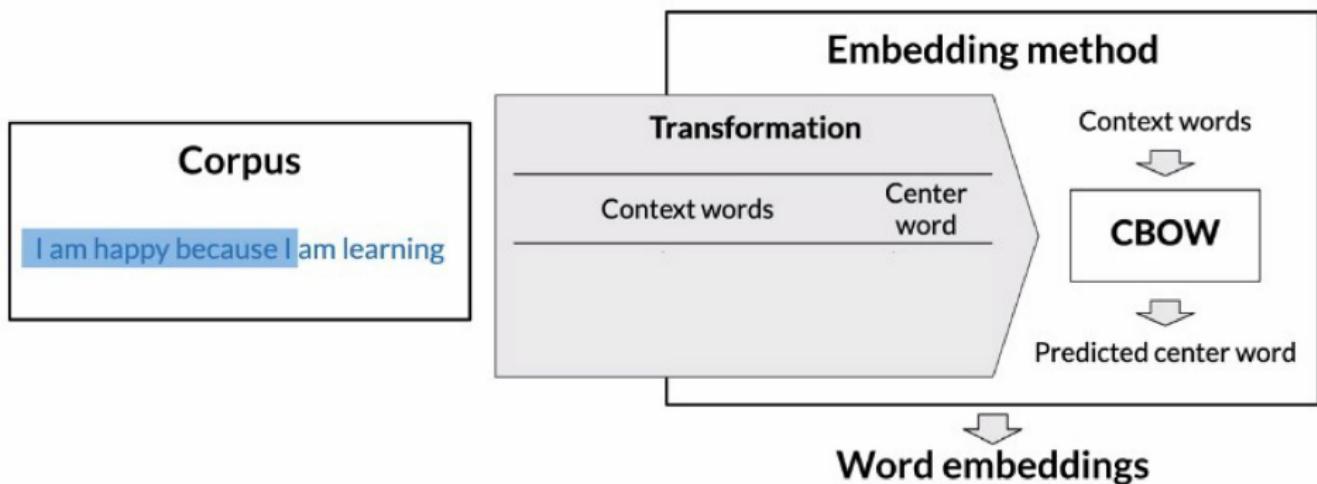


From corpus to training



Wordembeddings

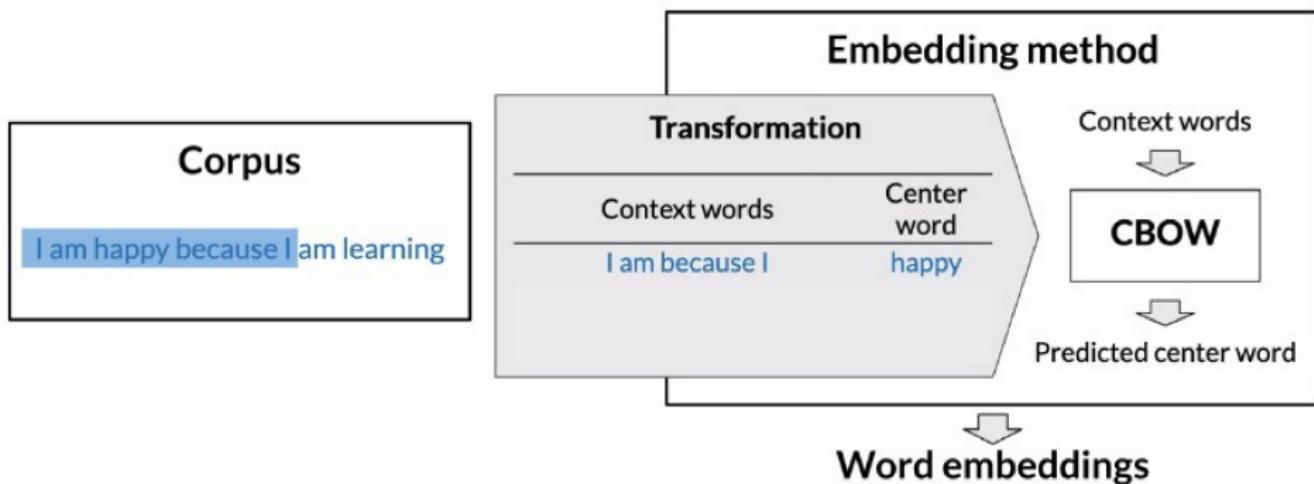
From corpus to training



Wordembeddings



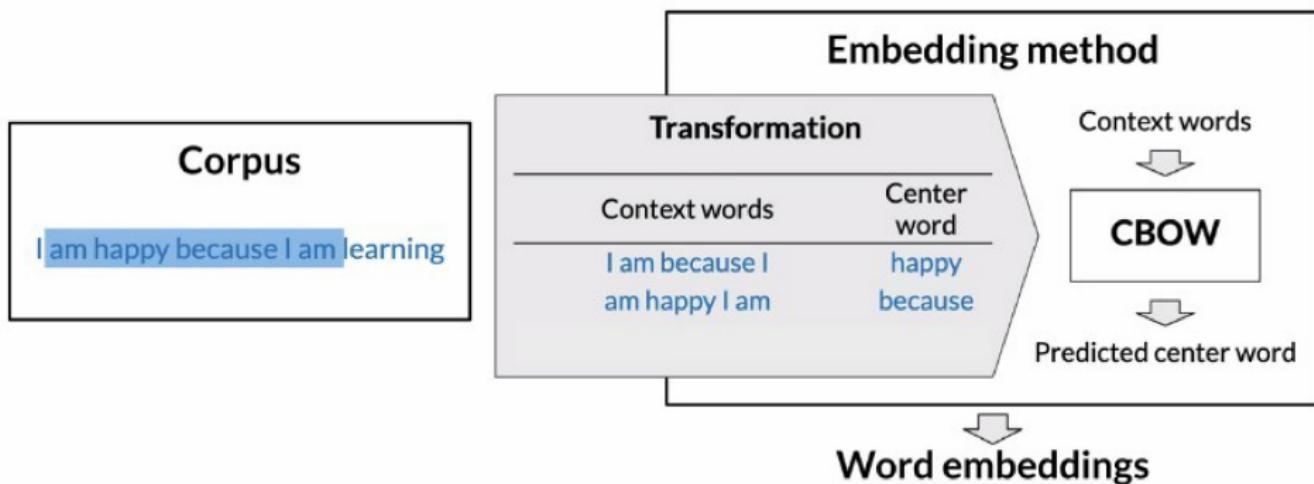
From corpus to training



Wordembeddings



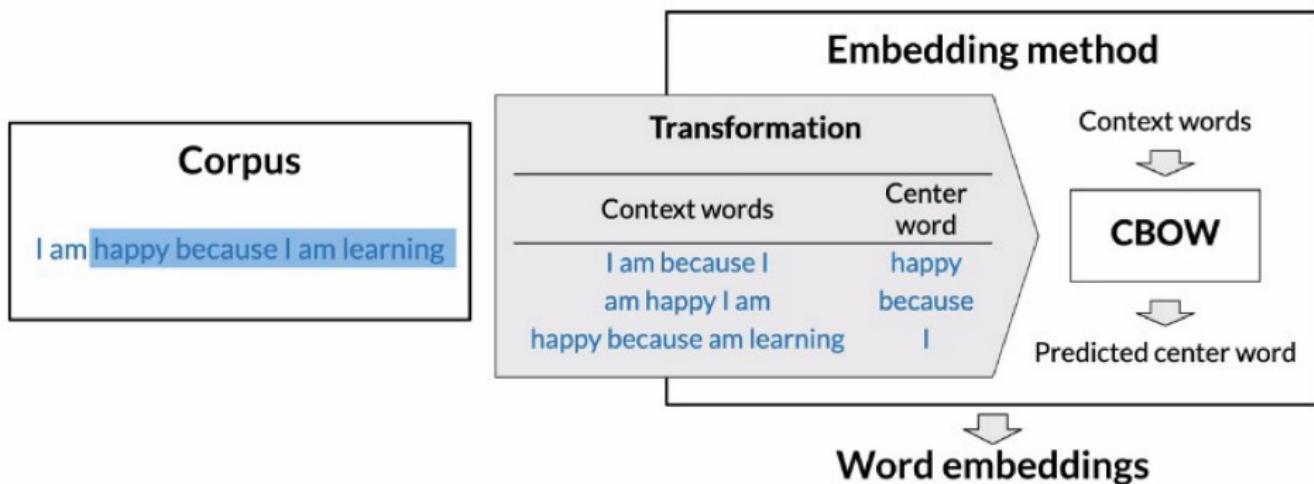
From corpus to training



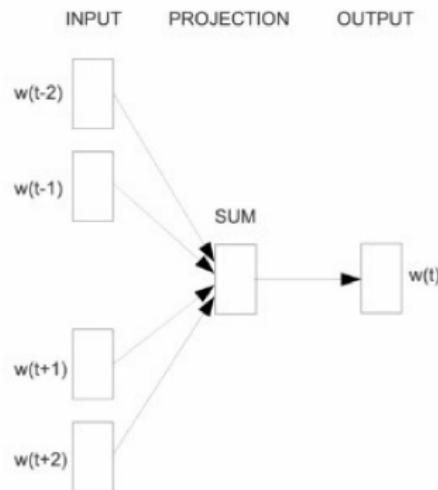
Wordembeddings



From corpus to training



CBOW in a nutshell



Source: Mikolov, T., Chen, K.,
Corrado, G.S., & Dean, J. (2013).
[Efficient Estimation of Word
Representations in Vector Space](#)

Cleaning and tokenization matters

Cleaning and tokenization matters

- Letter case

Cleaning and tokenization matters

- Letter case “The” == “the” == “THE”

Cleaning and tokenization matters

- Letter case "The" == "the" == "THE" → *lowercase / uppercase*

Cleaning and tokenization matters

- Letter case "The" == "the" == "THE" → *lowercase / uppercase*
- Punctuation

Cleaning and tokenization matters

- Letter case "The" == "the" == "THE" → lowercase / uppercase
- Punctuation , ! . ? → .

Wordembeddings

Cleaning and tokenization matters

- Letter case "The" == "the" == "THE" → lowercase / uppercase
- Punctuation , ! . ? → . " ‘ ‘ ‘ ’ ’ ” → Ø

Wordembeddings

Cleaning and tokenization matters

- Letter case "The" == "the" == "THE" → lowercase / uppercase
- Punctuation , ! . ? → . " ‘ ‘ ‘ ’ ’ ” → Ø ... !! ??? → .

Cleaning and tokenization matters

- Letter case "The" == "the" == "THE" → lowercase / uppercase
- Punctuation , ! . ? → . " ‘ ‘ ‘ ’ ’ ” → Ø ... !! ??? → .
- Numbers

Wordembeddings

Cleaning and tokenization matters

- Letter case "The" == "the" == "THE" → lowercase / uppercase
- Punctuation , ! . ? → . " ‘ ‘ ‘ ’ ’ ” → ∅ ... !! ??? → .
- Numbers 1 2 3 5 8 → ∅ 3.14159 90210

Wordembeddings

Cleaning and tokenization matters

- Letter case "The" == "the" == "THE" → lowercase / uppercase
- Punctuation , ! . ? → . " ‘ ‘ ‘ ’ ’ ” → ∅ ... !! ??? → .
- Numbers 1 2 3 5 8 → ∅ 3.14159 90210

Wordembeddings

Cleaning and tokenization matters

- Letter case "The" == "the" == "THE" → lowercase / uppercase
- Punctuation , ! . ? → . " ‘ ‘ ‘ ’ ’ ” → ∅ ... !! ??? → .
- Numbers 1 2 3 5 8 → ∅ 3.14159 90210 → as is / <NUMBER>

Cleaning and tokenization matters

- Letter case "The" == "the" == "THE" → lowercase / uppercase
- Punctuation , ! . ? → . " ‘ ‘ ‘ ’ ’ ” → Ø ... !! ??? → .
- Numbers 1 2 3 5 8 → Ø 3.14159 90210 → as is / <NUMBER>
- Special characters

Wordembeddings

Cleaning and tokenization matters

- Letter case "The" == "the" == "THE" → lowercase / uppercase
- Punctuation , ! . ? → . " ‘ ‘ ‘ ’ ’ ” → Ø ... !! ??? → .
- Numbers 1 2 3 5 8 → Ø 3.14159 90210 → as is / <NUMBER>
- Special characters ₣ \$ € § ¶ **

Wordembeddings

Cleaning and tokenization matters

- Letter case "The" == "the" == "THE" → lowercase / uppercase
- Punctuation , ! . ? → . " ‘ ‘ ‘ ’ ’ ” → Ø ... !! ??? → .
- Numbers 1 2 3 5 8 → Ø 3.14159 90210 → as is / <NUMBER>
- Special characters ⌧ \$ € § ¶ ** → Ø

Cleaning and tokenization matters

- Letter case "The" == "the" == "THE" → lowercase / uppercase
- Punctuation , ! . ? → . " ‘ ‘ ‘ ’ ’ ” → Ø ... !! ??? → .
- Numbers 1 2 3 5 8 → Ø 3.14159 90210 → as is / <NUMBER>
- Special characters ∇ \$ € § ¶ ** → Ø
- Special words

Cleaning and tokenization matters

- Letter case "The" == "the" == "THE" → lowercase / uppercase
- Punctuation , ! . ? → . " ‘ ‘ ‘ ’ ’ ” → Ø ... !! ??? → .
- Numbers 1 2 3 5 8 → Ø 3.14159 90210 → as is / <NUMBER>
- Special characters ∇ \$ € § ¶ ** → Ø
- Special words 😊 #nlp

Cleaning and tokenization matters

- Letter case "The" == "the" == "THE" → lowercase / uppercase
- Punctuation , ! . ? → . " ‘ ‘ ‘ ’ ’ ” → Ø ... !! ??? → .
- Numbers 1 2 3 5 8 → Ø 3.14159 90210 → as is / <NUMBER>
- Special characters ⌂ \$ € § ¶ ** → Ø
- Special words 😊 #nlp → :happy: #nlp

Wordembeddings

Example in Python: corpus

Who ❤️ "word embeddings" in 2020? I do!!!

Wordembeddings

Example in Python: corpus

Who ❤️ "word embeddings" in 2020? I do!!!

The sentence is divided into tokens by vertical orange bars. The tokens are: Who (purple box), ❤️ (orange box), "word embeddings" (blue box), in (orange box), 2020? (green box), I (orange box), do!!! (green box). Below the sentence, three categories are listed: emoji, punctuation, and number, each associated with a colored box.

emoji	punctuation	number
-------	-------------	--------

Example in Python: libraries

```
# pip install nltk
# pip install emoji

import nltk
from nltk.tokenize import word_tokenize
import emoji

nltk.download('punkt') # download pre-trained Punkt tokenizer for English
```

Wordembeddings

Example in Python: code

```
corpus = 'Who ❤ "word embeddings" in 2020? I do!!!'  
  
data = re.sub(r'[,!?;-]+', '.', corpus)
```

→ Who ❤ "word embeddings" in 2020. I do.

Wordembeddings

Example in Python: code

```
corpus = 'Who ❤️ "word embeddings" in 2020? I do!!!'

data = re.sub(r'[,!?;-]+', '.', corpus)
data = nltk.word_tokenize(data) # tokenize string to words

→ ['Who', '❤️', '', 'word', 'embeddings', "", 'in', '2020', '.', 'I',
'do', '.']
```

Wordembeddings

Example in Python: code

```
corpus = 'Who ❤️ "word embeddings" in 2020? I do!!!'

data = re.sub(r'[,!?;-]+', '.', corpus)
data = nltk.word_tokenize(data) # tokenize string to words
data = [ ch.lower() for ch in data
         if ch.isalpha()
         or ch == '.'
         or emoji.get_emoji_regexp().search(ch)
     ]
```

Wordembeddings

Example in Python: code

```
corpus = 'Who ❤️ "word embeddings" in 2020? I do!!!'

data = re.sub(r'[,!?;-]+', '.', corpus)
data = nltk.word_tokenize(data) # tokenize string to words
data = [ ch.lower() for ch in data
         if ch.isalpha()
         or ch == '.'
         or emoji.get_emoji_regexp().search(ch)
     ]
→ ['who', '❤️', 'word', 'embeddings', 'in', '.', 'i', 'do', '.']
```

Sliding window of words in Python

```
def get_windows(words, C):
    i = C
    while i < len(words) - C:
        center_word = words[i]
        context_words = words[(i - C):i] + words[(i+1):(i+C+1)]
        yield context_words, center_word
        i += 1
```

Wordembeddings

Sliding window of words in Python

```
def get_windows(words, C):
    i = C
    while i < len(words) - C:
        center_word = words[i]
        context_words = words[(i - C):i] + words[(i+1):(i+C+1)]
        yield context_words, center_word
        i += 1
```

Wordembeddings

Sliding window of words in Python

```
def get_windows(words, C):
    i = C
    while i < len(words) - C:
        center_word = words[i]
        context_words = words[(i - C):i] + words[(i+1):(i+C+1)]
        yield context_words, center_word
        i += 1
```

Wordembeddings

Sliding window of words in Python

```
def get_windows(words, C):
    i = C
    while i < len(words) - C:
        center_word = words[i]
        context_words = words[(i - C):i] + words[(i+1):(i+C+1)]
        yield context_words, center_word
        i += 1
```

I	am	happy	because	I	am	learning
---	----	-------	---------	---	----	----------

Wordembeddings

Sliding window of words in Python

```
def get_windows(words, C):
    i = C
    while i < len(words) - C:
        center_word = words[i]
        context_words = words[(i - C):i] + words[(i+1):(i+C+1)]
        yield context_words, center_word
        i += 1
```

I	am	happy	because	I	am	learning
0	1	2	3	4	5	6

Wordembeddings

Sliding window of words in Python

```
def get_windows(words, C):
    i = C
    while i < len(words) - C:
        center_word = words[i]
        context_words = words[(i - C):i] + words[(i+1):(i+C+1)]
        yield context_words, center_word
        i += 1
```

I	am	happy	because	I	am	learning
0	1	2	3	4	5	6

Wordembeddings

Sliding window of words in Python

```
def get_windows(words, C):
    i = C
    while i < len(words) - C:
        center_word = words[i]
        context_words = words[(i - C):i] + words[(i+1):(i+C+1)]
        yield context_words, center_word
        i += 1
```

I	am	happy	because	I	am	learning
0	1	2	3	4	5	6

Wordembeddings

Sliding window of words in Python

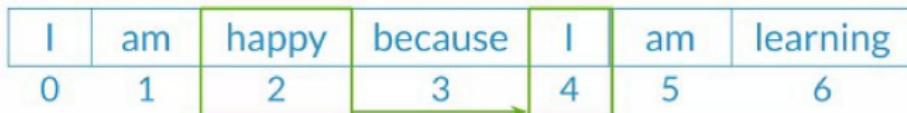
```
def get_windows(words, C):
    i = C
    while i < len(words) - C:
        center_word = words[i]
        context_words = words[(i - C):i] + words[(i+1):(i+C+1)]
        yield context_words, center_word
        i += 1
```



Wordembeddings

Sliding window of words in Python

```
def get_windows(words, C):
    i = C
    while i < len(words) - C:
        center_word = words[i]
        context_words = words[(i - C):i] + words[(i+1):(i+C+1)]
        yield context_words, center_word
        i += 1
```



Wordembeddings

Sliding window of words in Python

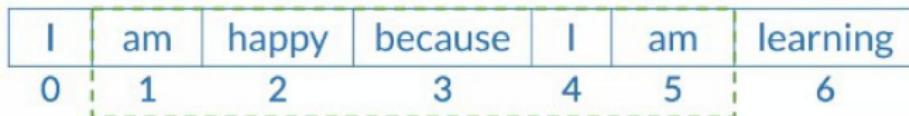
```
def get_windows(words, C):
    i = C
    while i < len(words) - C:
        center_word = words[i]
        context_words = words[(i - C):i] + words[(i+1):(i+C+1)]
        yield context_words, center_word
        i += 1
```

I	am	happy	because	I	am	learning
0	1	2	3	4	5	6

Wordembeddings

Sliding window of words in Python

```
def get_windows(words, C):
    i = C
    while i < len(words) - C:
        center_word = words[i]
        context_words = words[(i - C):i] + words[(i+1):(i+C+1)]
        yield context_words, center_word
        i += 1
```



Wordembeddings

Sliding window of words in Python

```
def get_windows(words, C):
    ...
    yield context_words, center_word
```

```
for x, y in get_windows(
        ['i', 'am', 'happy', 'because', 'i', 'am', 'learning'],
        2
    ):
    print(f'{x}\t{y}')
```

Wordembeddings

Sliding window of words in Python

```
def get_windows(words, C):
    ...
    yield context_words, center_word
```

```
for x, y in get_windows(
        ['i', 'am', 'happy', 'because', 'i', 'am', 'learning'],
        2
    ):
    print(f'{x}\t{y}')
```

Wordembeddings

Sliding window of words in Python

```
for x, y in get_windows(  
    ['i', 'am', 'happy', 'because', 'i', 'am', 'learning'],  
    2  
):  
    print(f'{x}\t{y}')
```

```
→ ['I', 'am', 'because', 'I']      happy  
  ['am', 'happy', 'I', 'am']      because  
  ['happy', 'because', 'am', 'learning'] I
```

Wordembeddings

Transforming center words into vectors

Corpus I am happy because I am learning

Wordembeddings

Transforming center words into vectors

Corpus I am happy because I am learning

Vocabulary am, because, happy, I, learning

Wordembeddings

Transforming center words into vectors

Corpus I am happy because I am learning

Vocabulary am, because, happy, I, learning

One-hot
vector

am
because
happy
I
learning



Wordembeddings

Transforming center words into vectors

Corpus I am happy because I am learning

Vocabulary am, because, happy, I, learning

One-hot
vector

	am
am	1
because	0
happy	0
I	0
learning	0

Wordembeddings

Transforming center words into vectors

Corpus I am happy because I am learning

Vocabulary am, because, happy, I, learning

One-hot vector	am	because
am	1	0
because	0	1
happy	0	0
I	0	0
learning	0	0

Wordembeddings

Transforming center words into vectors

Corpus I am happy because I am learning

Vocabulary am, because, happy, I, learning

One-hot vector	am	because	happy	I	learning
am	1	0	0	0	0
because	0	1	0	0	0
happy	0	0	1	0	0
I	0	0	0	1	0
learning	0	0	0	0	1

Transforming context words into vectors

Average of individual one-hot vectors

Transforming context words into vectors

Average of individual one-hot vectors

I am because I

Wordembeddings

Transforming context words into vectors

Average of individual one-hot vectors

$$\begin{array}{c|c|c|c} & | & am & because & | \\ \text{am} & 0 & & & \\ \text{because} & 0 & & & \\ \text{happy} & 0 & & & \\ I & 1 & & & \\ \text{learning} & 0 & & & \end{array}$$

Wordembeddings

Transforming context words into vectors

Average of individual one-hot vectors

$$\begin{matrix} & | & & & | \\ & am & & because & | \\ \begin{matrix} am \\ because \\ happy \\ I \\ learning \end{matrix} & \left(\begin{matrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{matrix} \right) & & & \end{matrix}$$

Wordembeddings

Transforming context words into vectors

Average of individual one-hot vectors

	I	am	because	I
am	$\begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}$	$\begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$		
because				
happy				
I				
learning				

Transforming context words into vectors

Average of individual one-hot vectors

	I	am	because	I
am	0	1	0	0
because	0	0	1	0
happy	0	0	0	0
I	1	0	0	1
learning	0	0	0	0

Wordembeddings

Transforming context words into vectors

Average of individual one-hot vectors

$$\left(\begin{array}{c} I \\ am \\ because \\ happy \\ I \\ learning \end{array} \right) + \left(\begin{array}{c} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{array} \right) + \left(\begin{array}{c} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{array} \right) + \left(\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{array} \right) \right) / 4 = \left(\begin{array}{c} 0.25 \\ 0.25 \\ 0 \\ 0.5 \\ 0 \end{array} \right)$$

I am because I

Wordembeddings

Transforming context words into vectors

Average of individual one-hot vectors

$$\left(\begin{array}{c} I \\ am \\ because \\ happy \\ I \\ learning \end{array} \right) + \left(\begin{array}{c} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{array} \right) + \left(\begin{array}{c} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{array} \right) + \left(\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{array} \right) \right) / 4 = \left(\begin{array}{c} 0.25 \\ 0.25 \\ 0 \\ 0.5 \\ 0 \end{array} \right)$$

I am because I

Wordembeddings

Final prepared training set

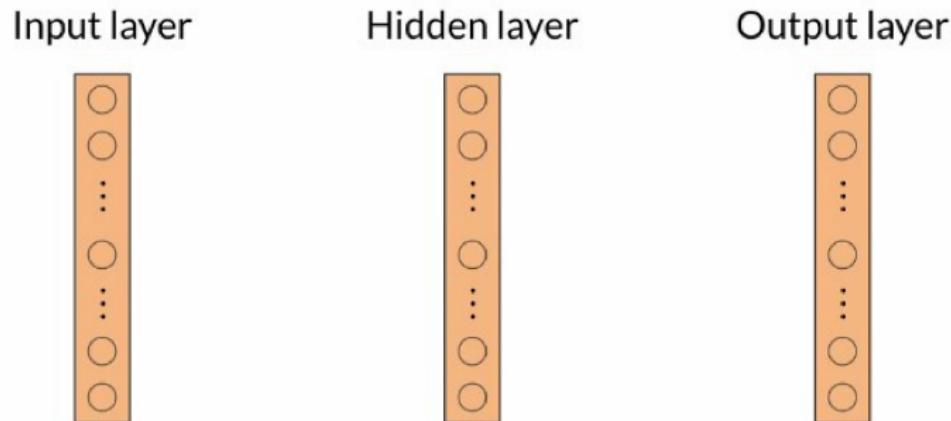
Context words	Context words vector	Center word	Center word vector
<i>I am because I</i>	[0.25; 0.25; 0; 0.5; 0]	<i>happy</i>	[0; 0; 1; 0; 0]

Final prepared training set

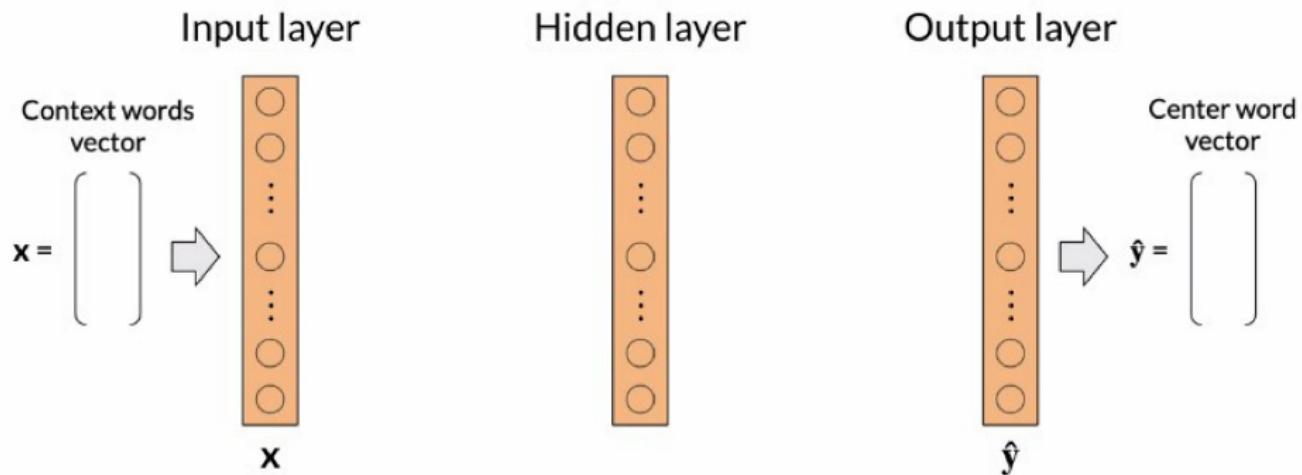
Context words	Context words vector	Center word	Center word vector
<i>I am because I</i>	[0.25; 0.25; 0; 0.5; 0]	<i>happy</i>	[0; 0; 1; 0; 0]
<i>am happy I am</i>	[0.5; 0; 0.25; 0.25; 0]	<i>because</i>	[0; 1; 0; 0; 0]
<i>happy because am learning</i>	[0.25; 0.25; 0.25; 0; 0.25]	<i>I</i>	[0; 0; 0; 1; 0]

Architecture of the CBOW model

Architecture of the CBOW model

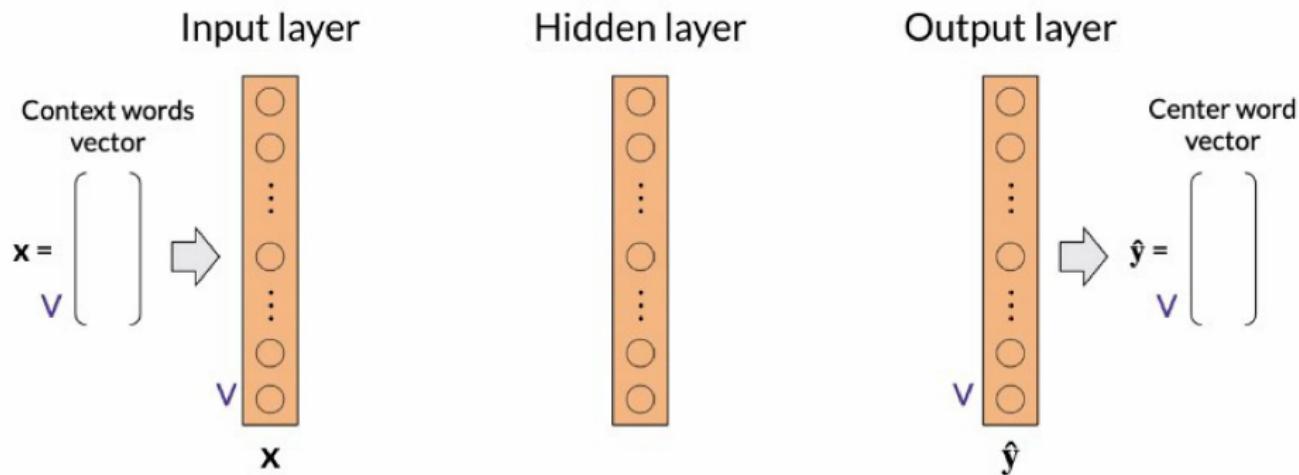


Architecture of the CBOW model

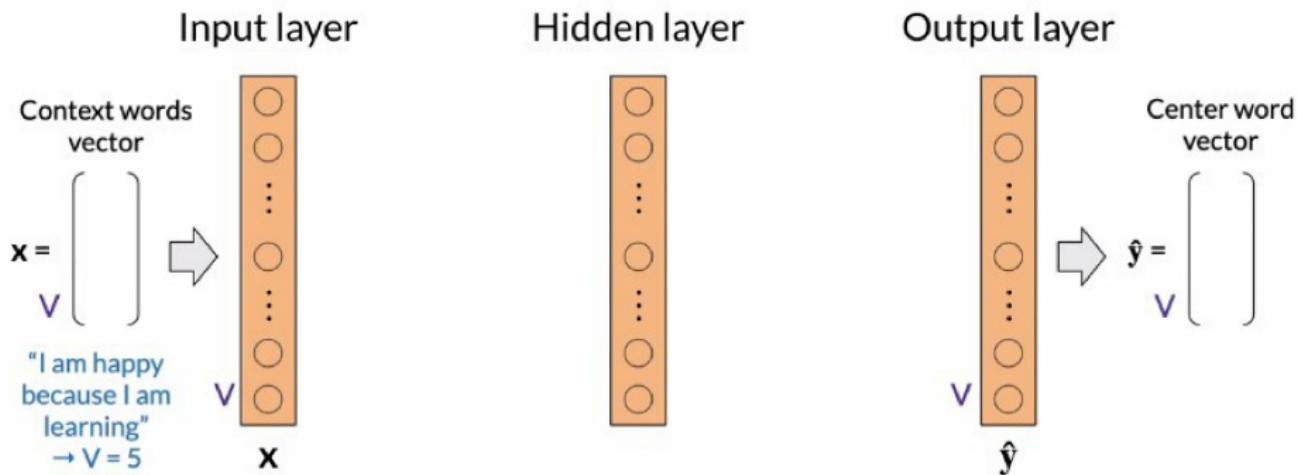


Wordembeddings

Architecture of the CBOW model



Architecture of the CBOW model

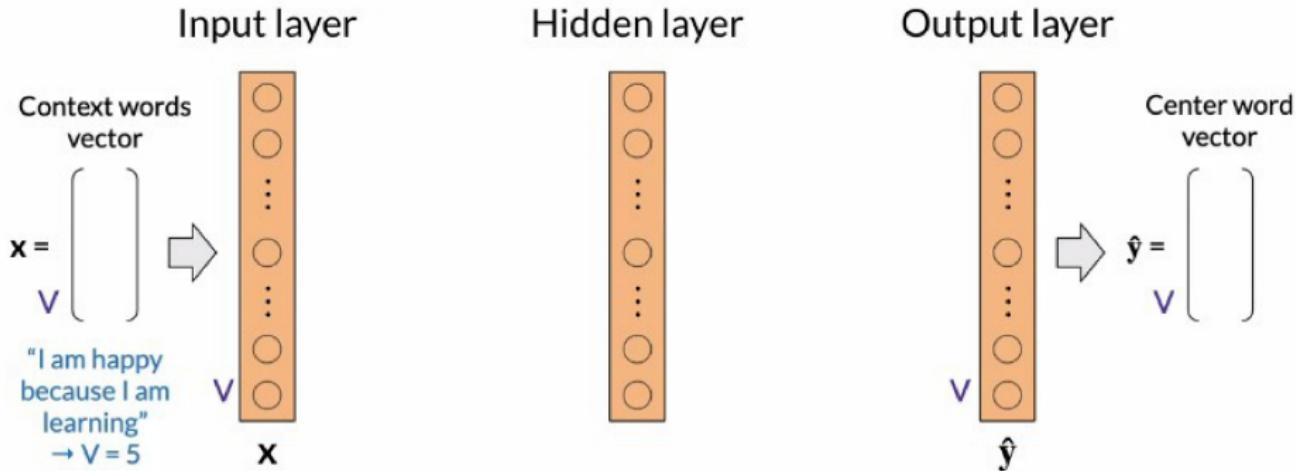


Architecture of the CBOW model

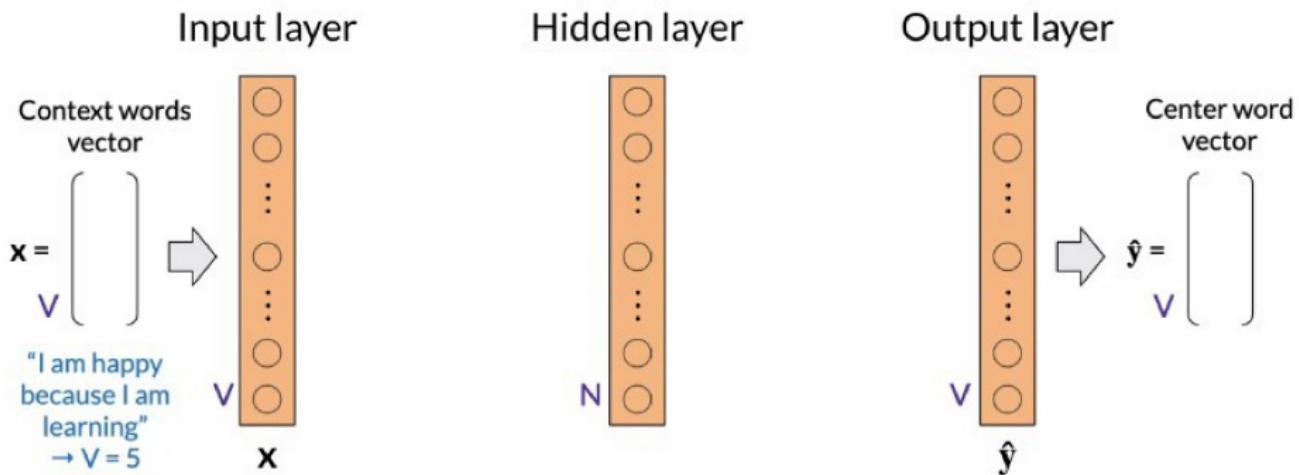
Hyperparameters

Word embedding size

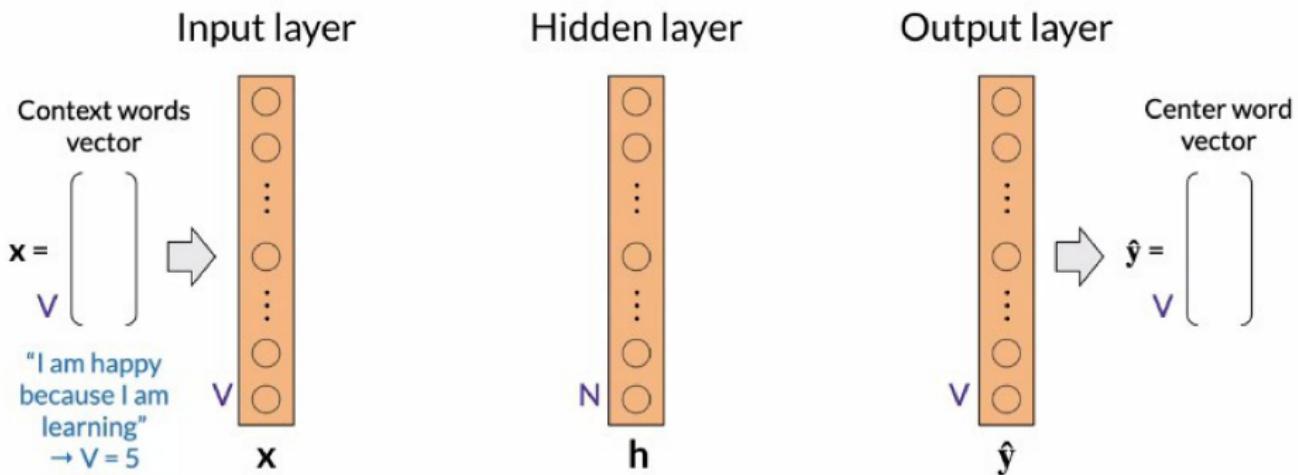
...



Architecture of the CBOW model



Architecture of the CBOW model

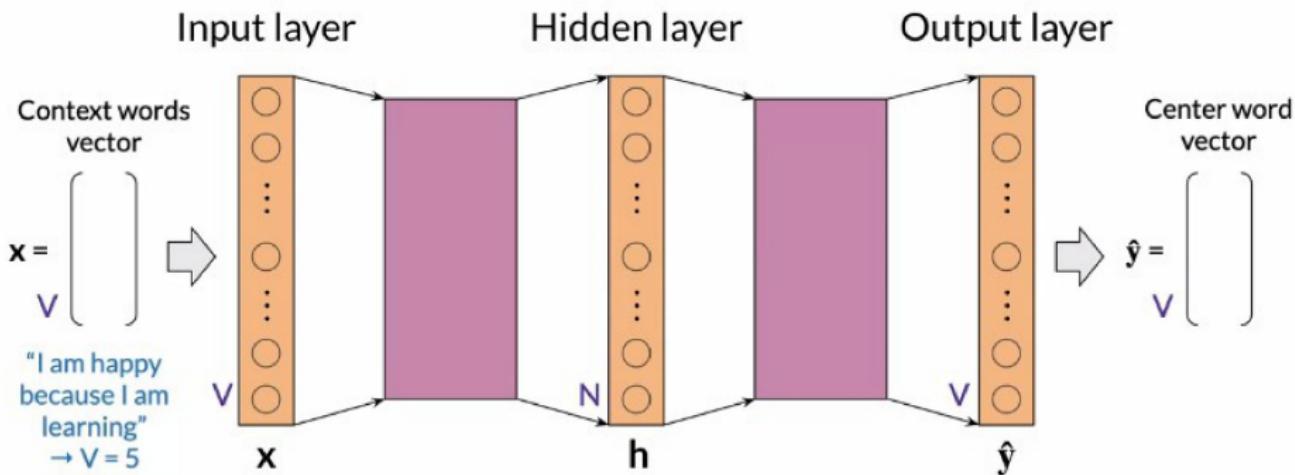


Architecture of the CBOW model

Hyperparameters

N: Word embedding size

...

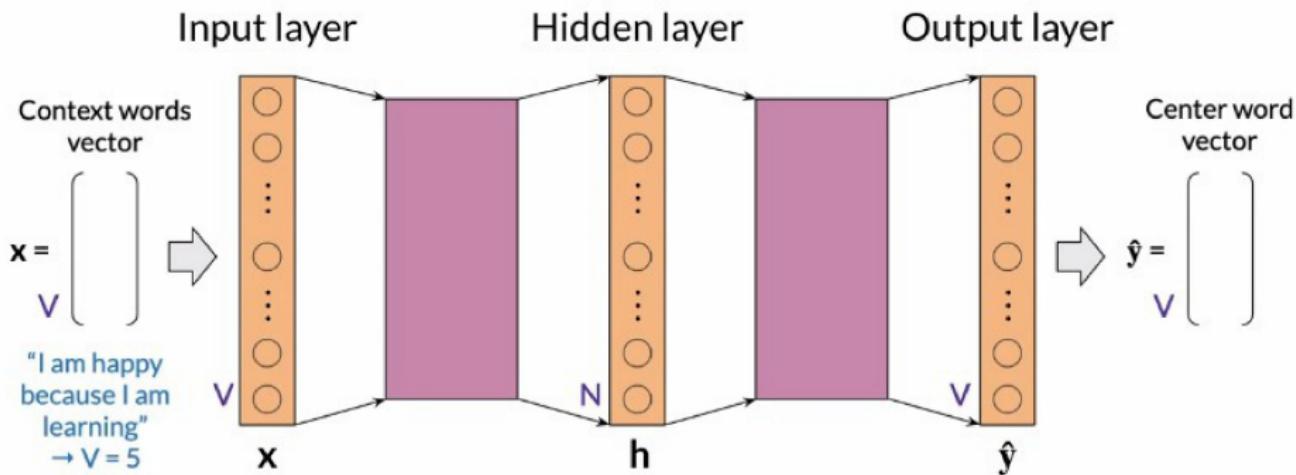


Architecture of the CBOW model

Hyperparameters

N: Word embedding size

...

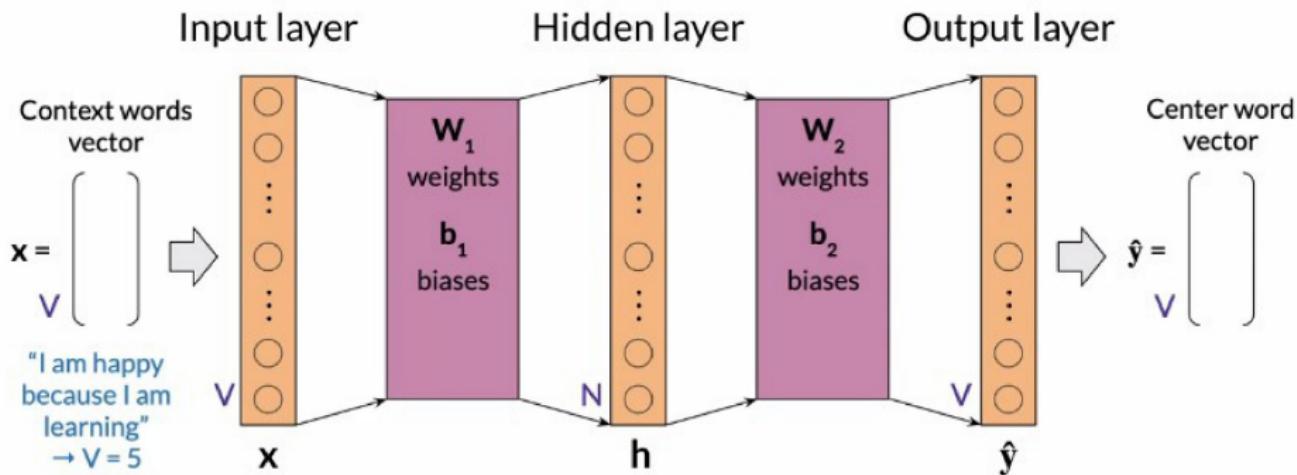


Wordembeddings

Architecture of the CBOW model

Hyperparameters

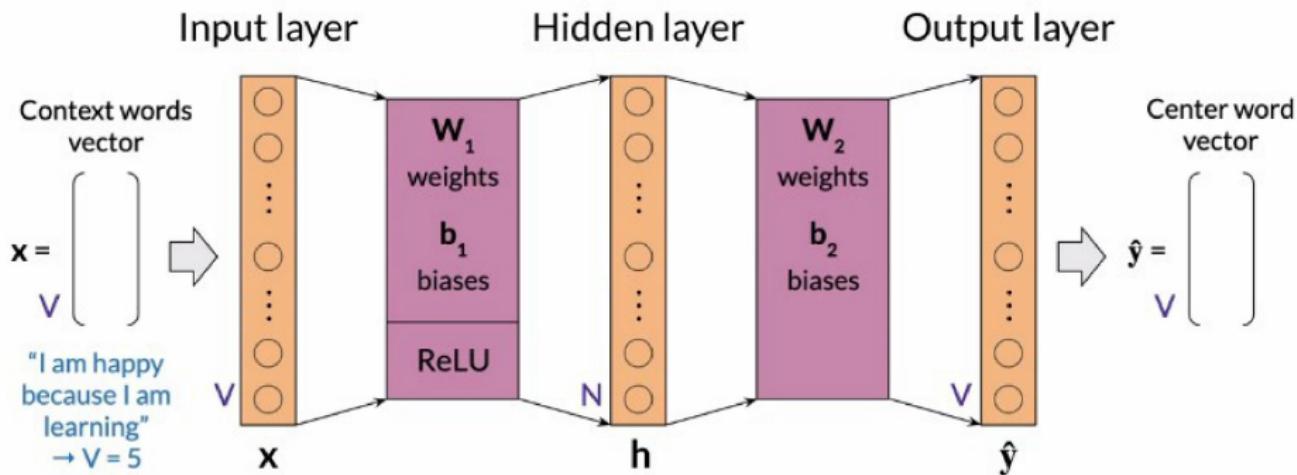
N: Word embedding size ...



Architecture of the CBOW model

Hyperparameters

N: Word embedding size ...

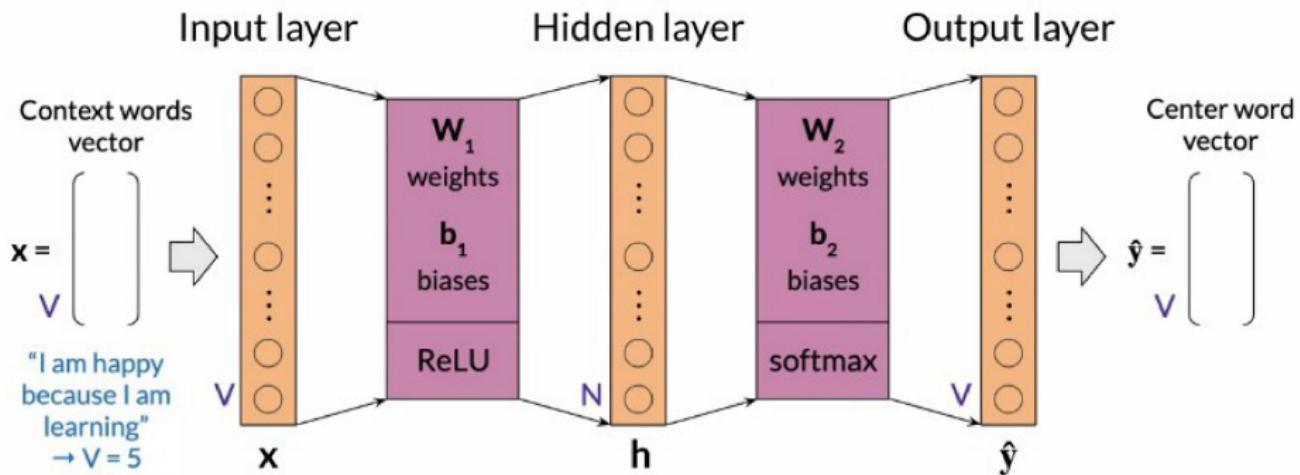


Architecture of the CBOW model

Hyperparameters

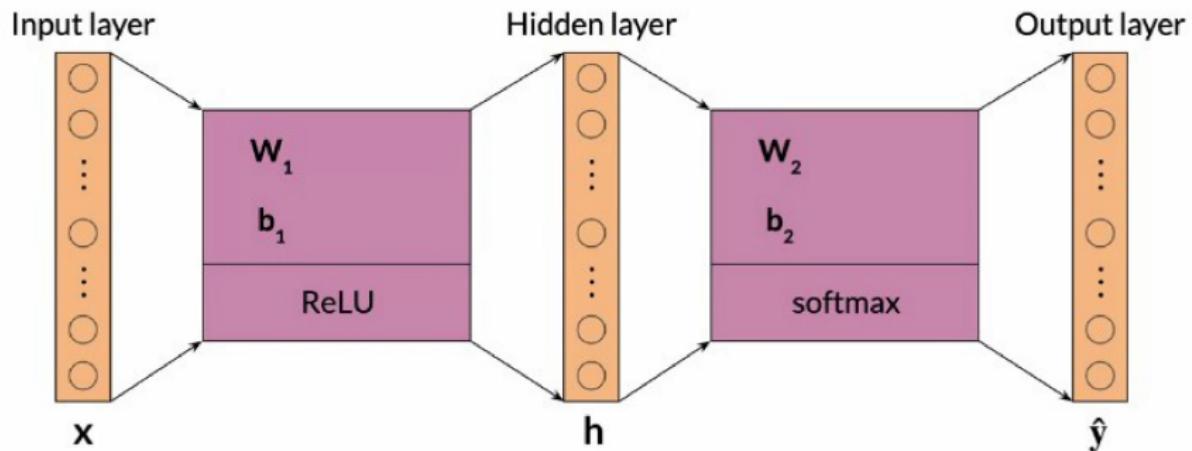
N : Word embedding size

...

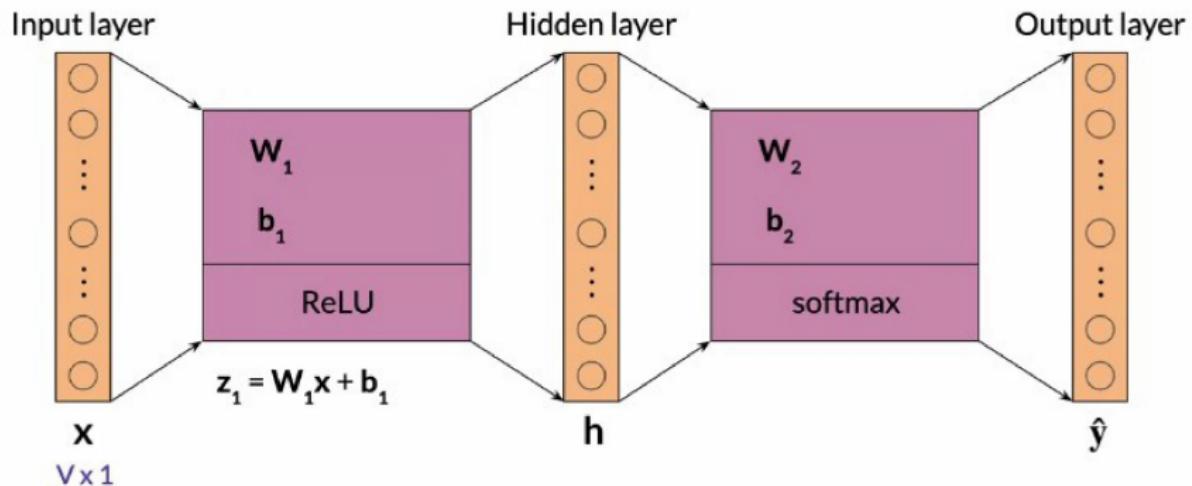


Wordembeddings

Dimensions (single input)

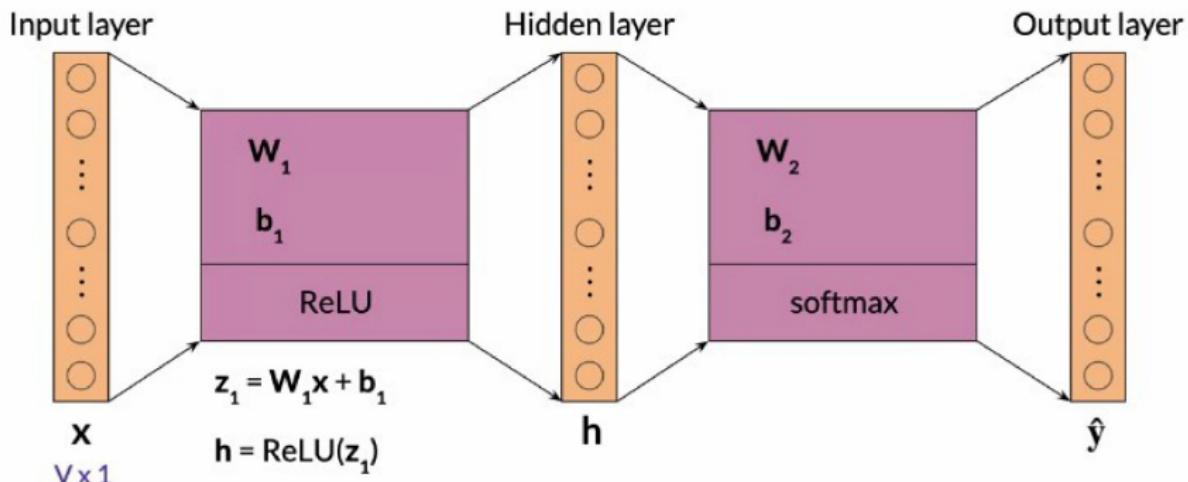


Dimensions (single input)



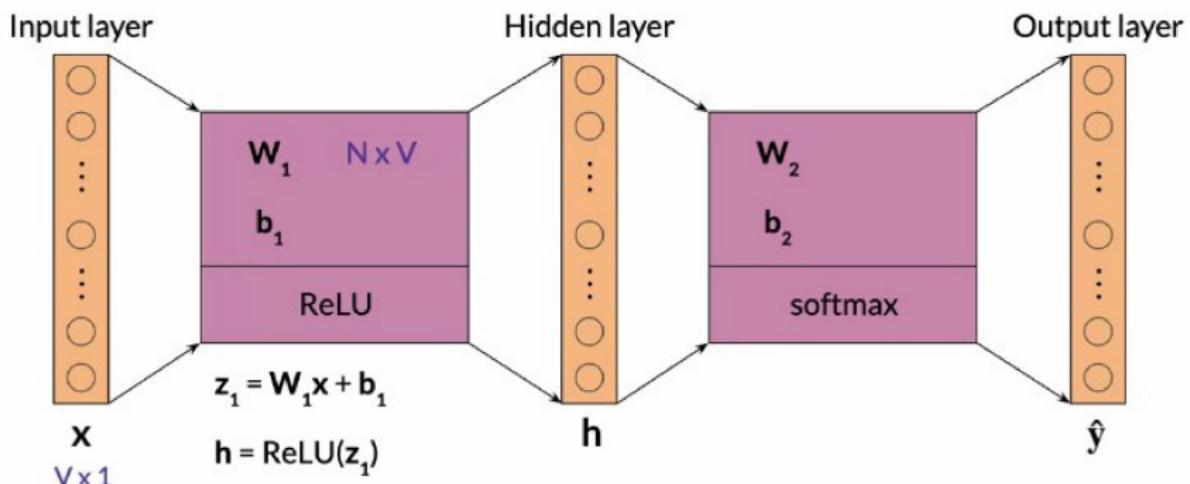
Wordembeddings

Dimensions (single input)



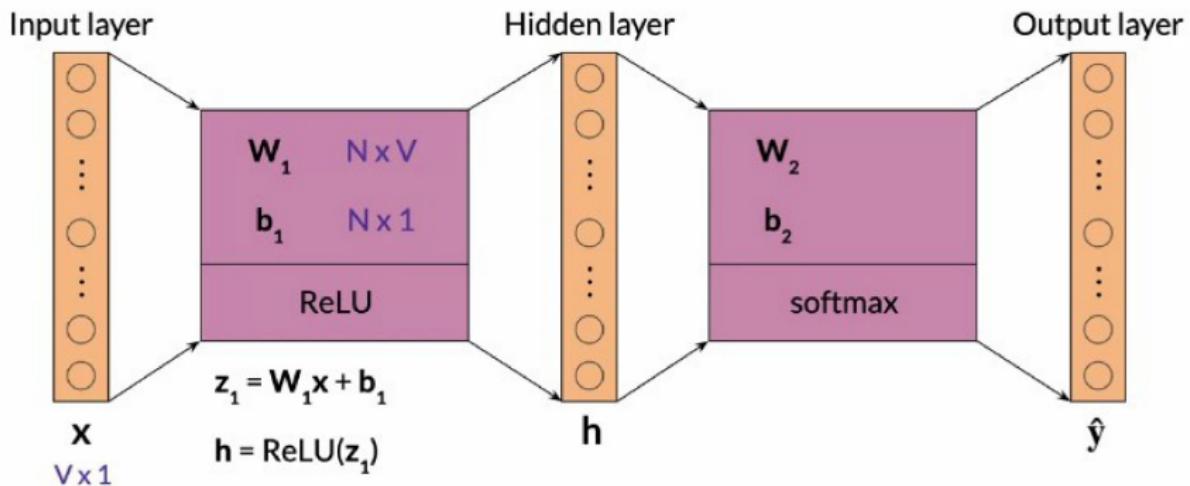
Wordembeddings

Dimensions (single input)



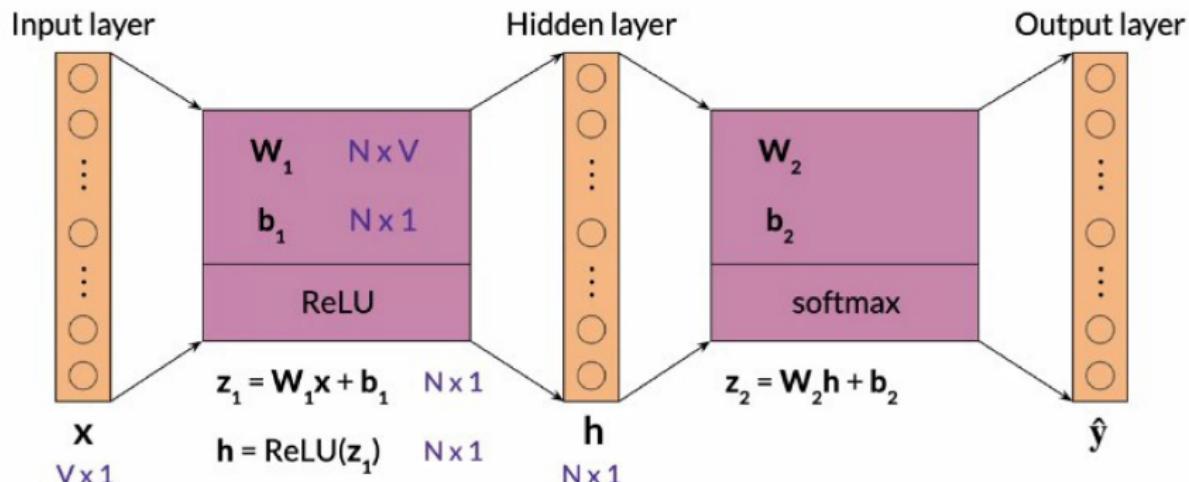
Wordembeddings

Dimensions (single input)



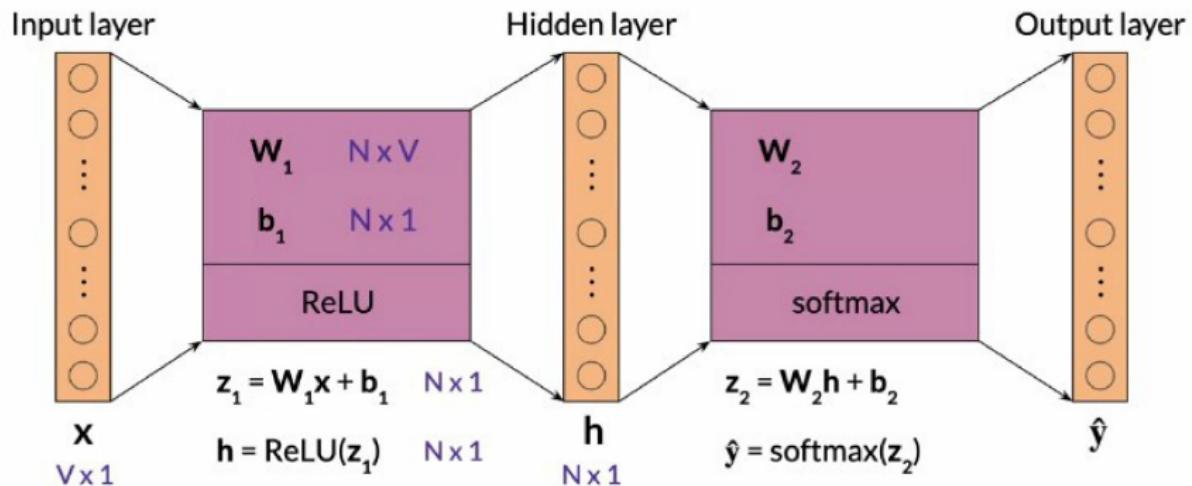
Wordembeddings

Dimensions (single input)



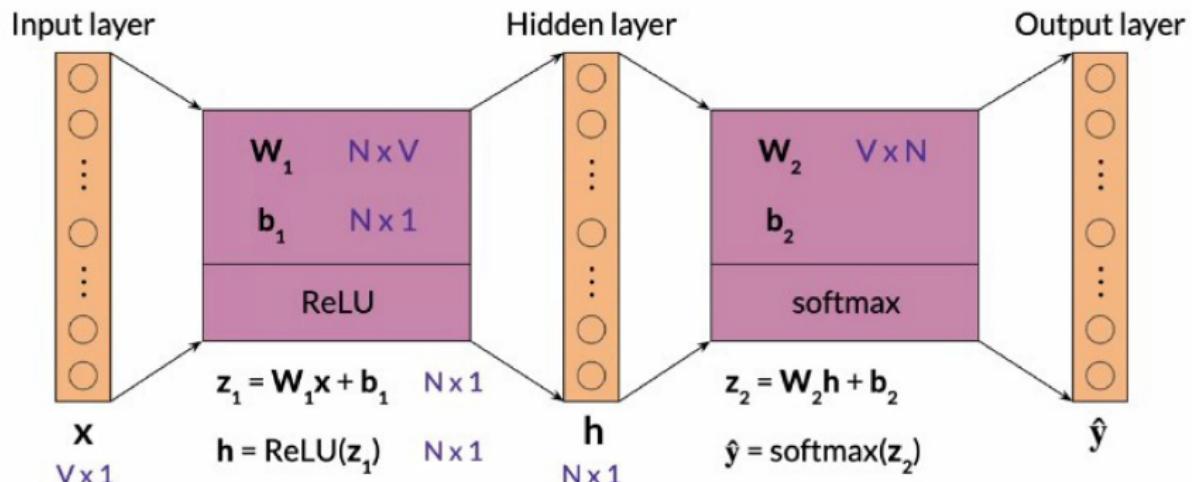
Wordembeddings

Dimensions (single input)



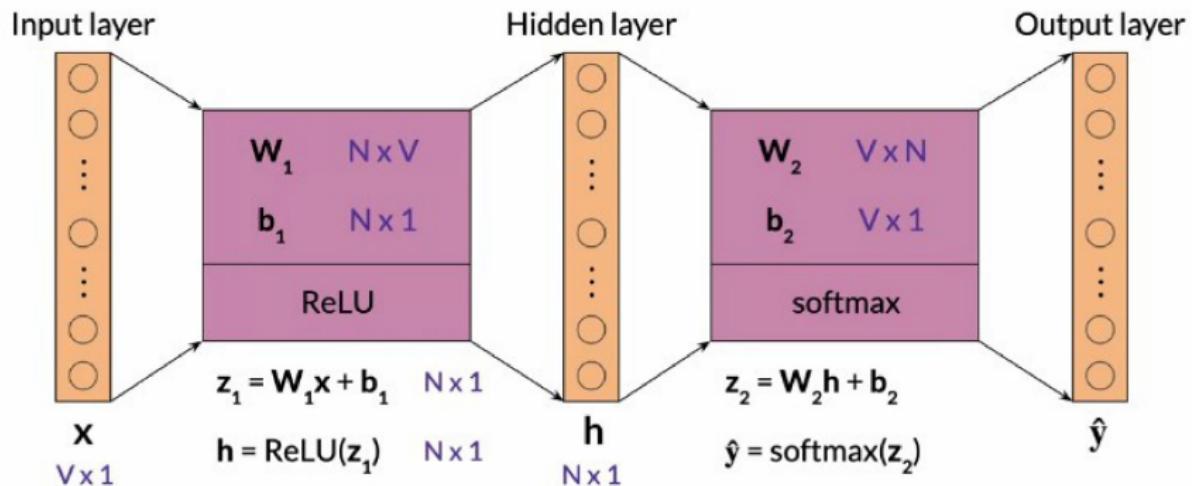
Wordembeddings

Dimensions (single input)



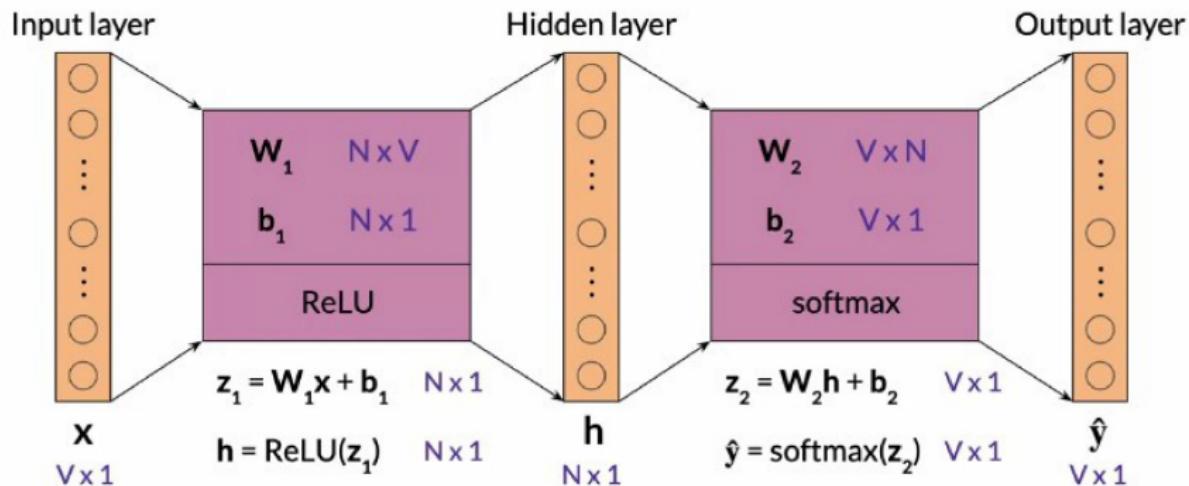
Wordembeddings

Dimensions (single input)



Wordembeddings

Dimensions (single input)



Dimensions (single input)

Column vectors

$$z_1 = \mathbf{W}_1 x + \mathbf{b}_1 \quad z_1 = \begin{pmatrix} \end{pmatrix} \quad \mathbf{W}_1 = \begin{pmatrix} & N \times V \end{pmatrix} \quad x = \begin{pmatrix} \end{pmatrix} \quad \mathbf{b}_1 = \begin{pmatrix} \end{pmatrix}$$

$N \times 1$ $V \times 1$ $N \times 1$

Dimensions (single input)

Column vectors

$$z_1 = W_1 x + b_1 \quad z_1 = \begin{pmatrix} \end{pmatrix} \quad W_1 = \begin{pmatrix} & N \times V \end{pmatrix} \quad x = \begin{pmatrix} \end{pmatrix} \quad b_1 = \begin{pmatrix} \end{pmatrix}$$

$N \times 1$ $V \times 1$ $N \times 1$

Wordembeddings

Dimensions (single input)

Column vectors

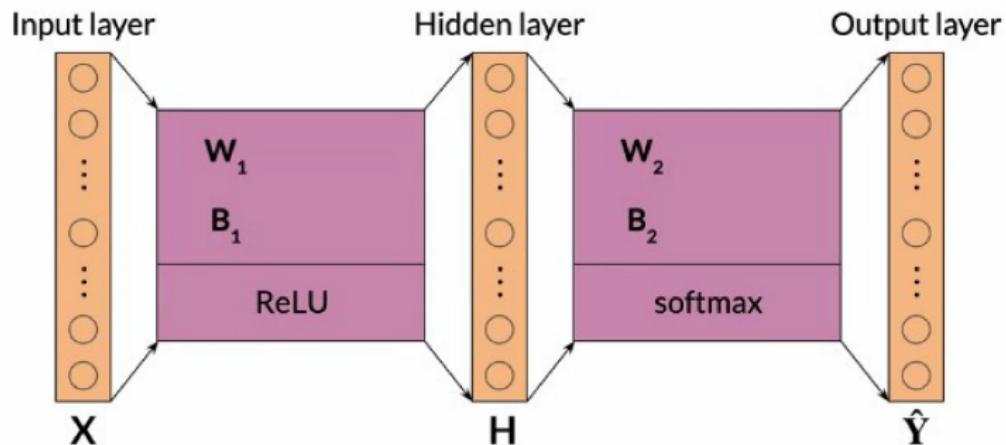
$$z_1 = \mathbf{W}_1 \mathbf{x} + \mathbf{b}_1 \quad z_1 = \begin{pmatrix} \end{pmatrix}_{N \times 1} \quad \mathbf{W}_1 = \begin{pmatrix} \end{pmatrix}_{N \times V} \quad \mathbf{x} = \begin{pmatrix} \end{pmatrix}_{V \times 1} \quad \mathbf{b}_1 = \begin{pmatrix} \end{pmatrix}_{N \times 1}$$

Row vectors

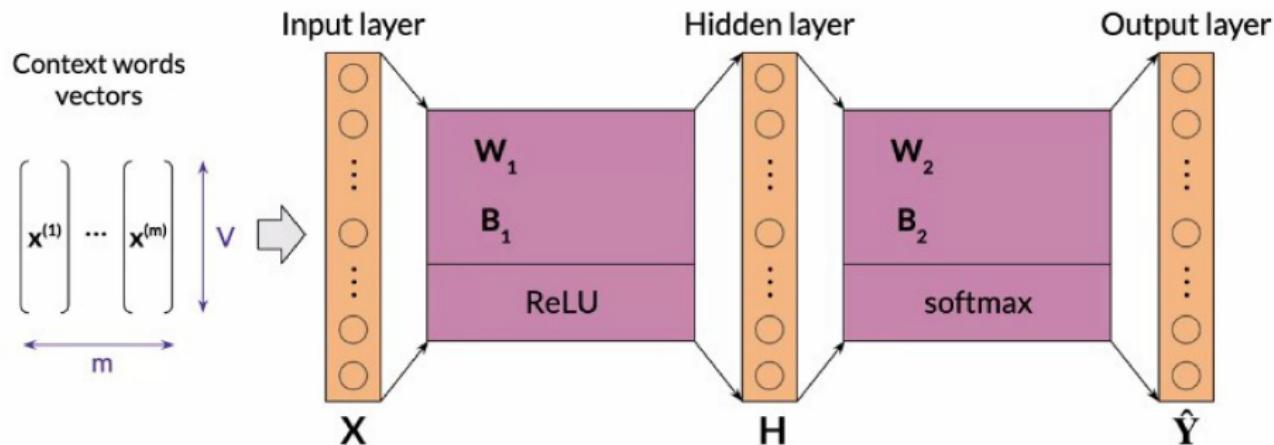
$$z_1 = \mathbf{x} \mathbf{W}_1^T + \mathbf{b}_1 \quad \mathbf{b}_1 = \begin{pmatrix} 1 \times N \end{pmatrix} \quad \mathbf{W}_1 = \begin{pmatrix} \end{pmatrix}_{N \times V} \quad \mathbf{b}_1 = \begin{pmatrix} 1 \times N \end{pmatrix}$$
$$\mathbf{x} = \begin{pmatrix} 1 \times V \end{pmatrix}$$

Arquitecture CBOW

Dimensions (batch input)

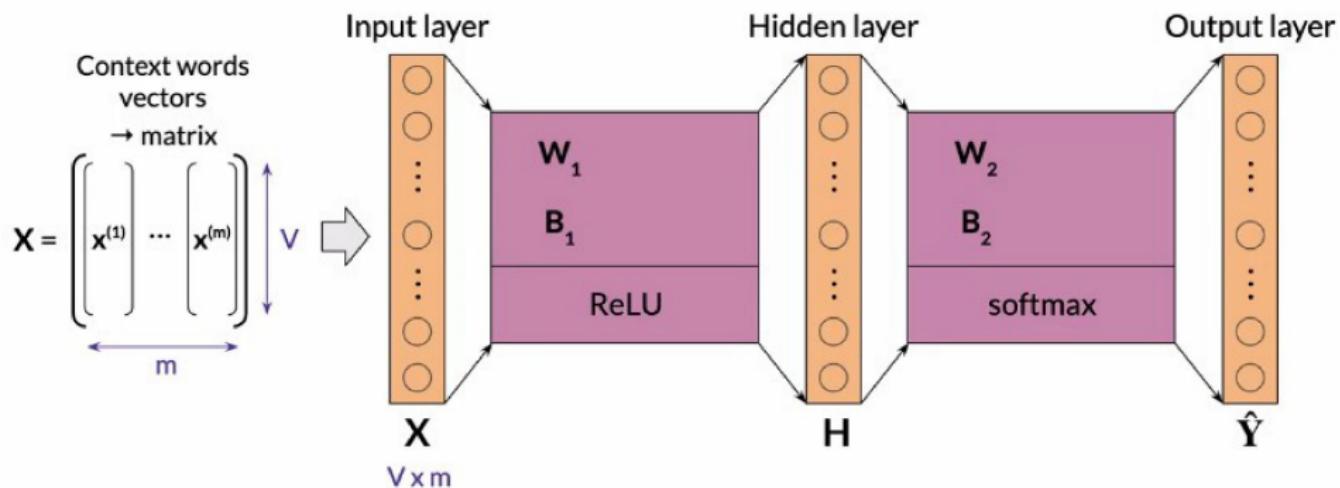


Dimensions (batch input)

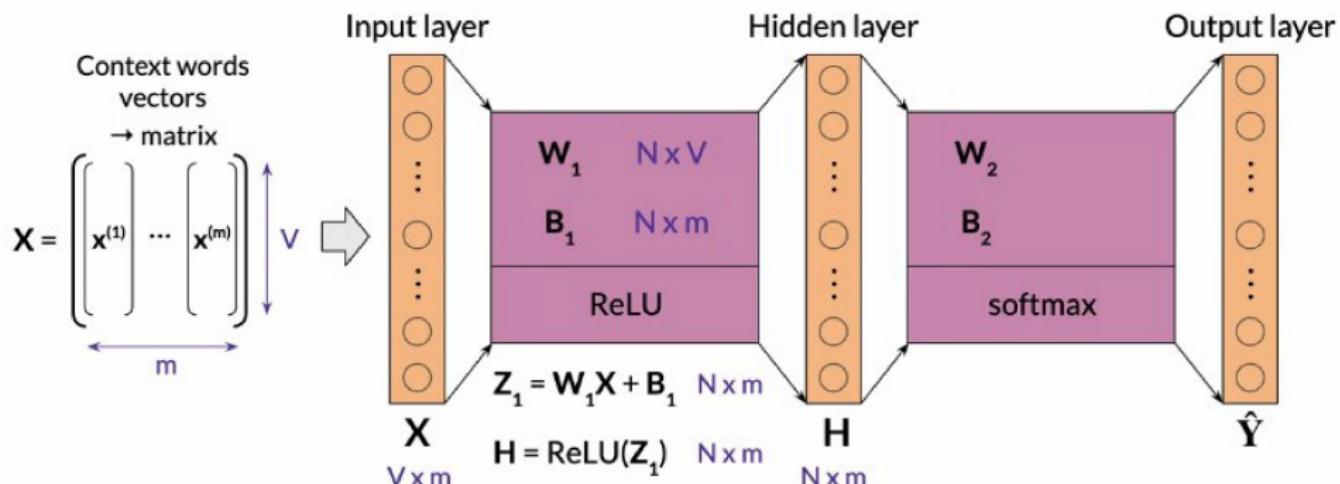


Wordembeddings

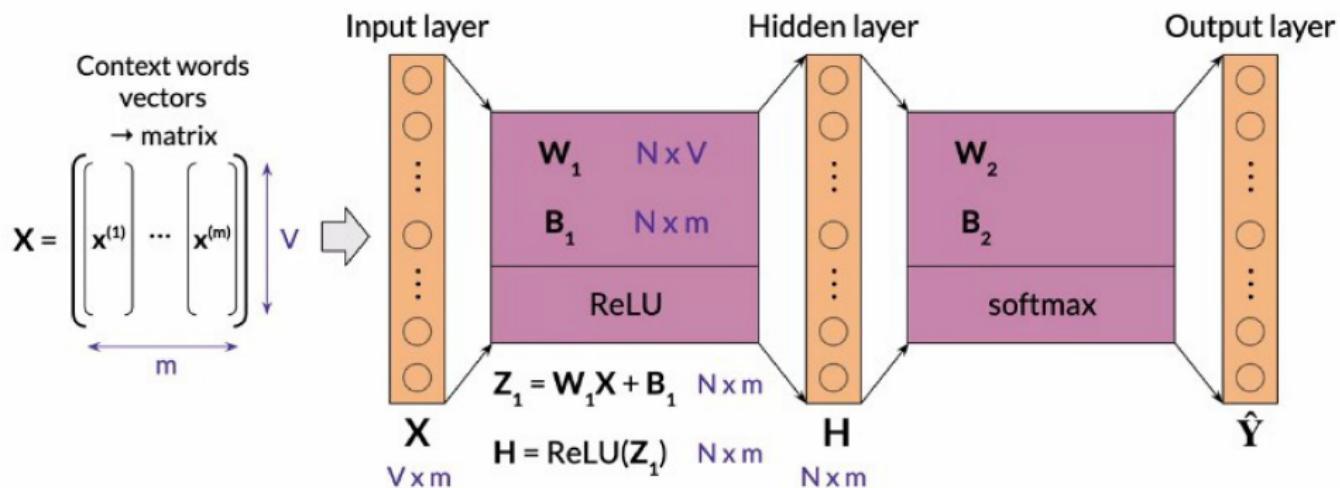
Dimensions (batch input)



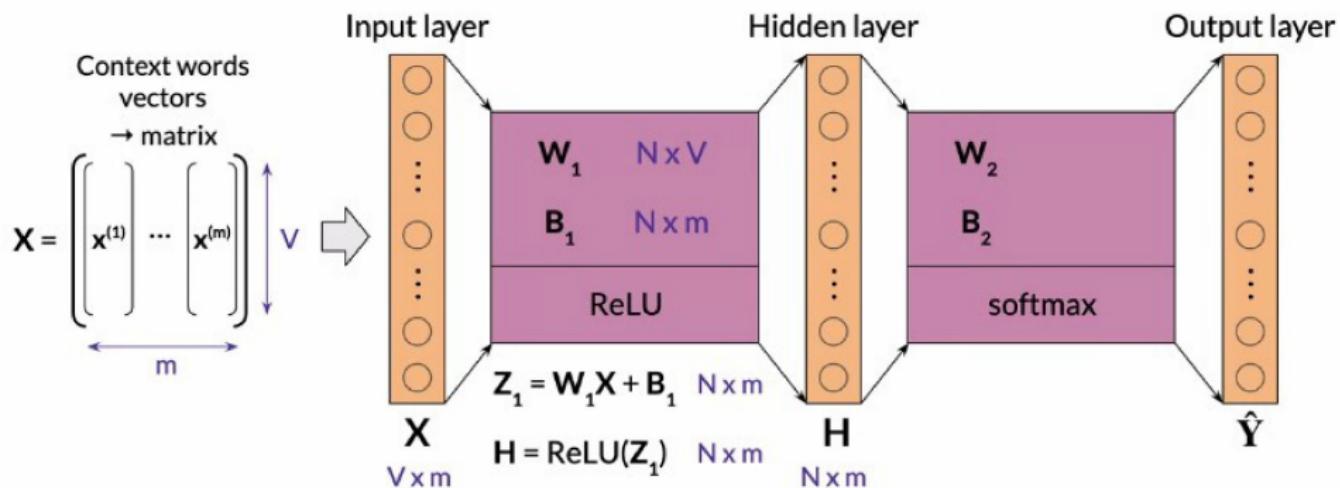
Dimensions (batch input)



Dimensions (batch input)



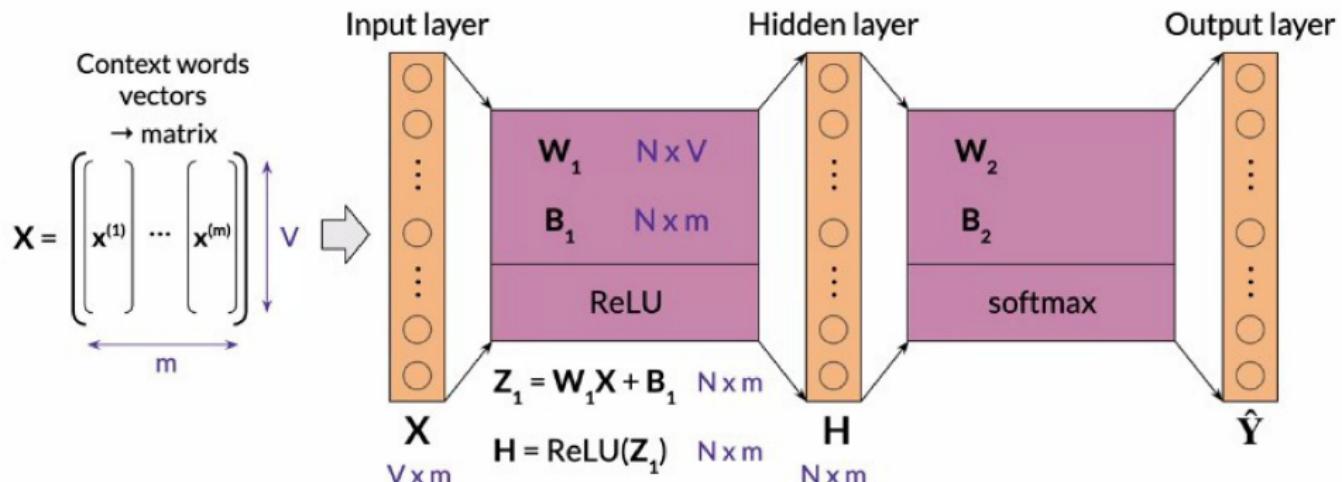
Dimensions (batch input)



Wordembeddings

Dimensions (batch input)

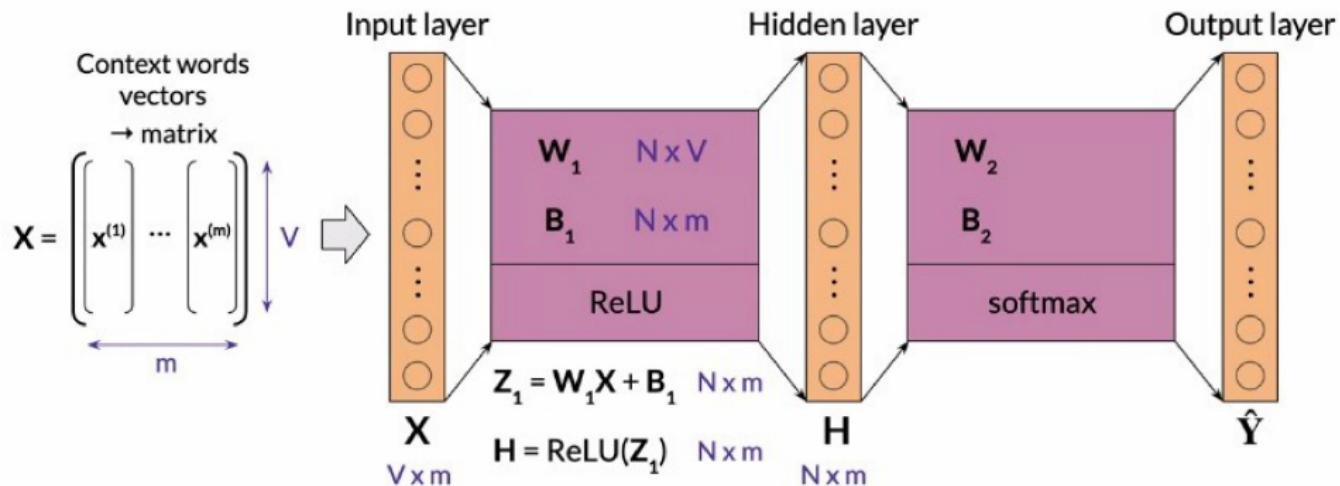
$$\begin{bmatrix} \mathbf{b}_1 \\ \vdots \\ \mathbf{b}_m \end{bmatrix} \rightarrow \mathbf{B}_1 = \begin{bmatrix} \mathbf{b}_1 \\ \vdots \\ \mathbf{b}_1 \end{bmatrix} \quad \text{N}$$



Wordembeddings

Dimensions (batch input)

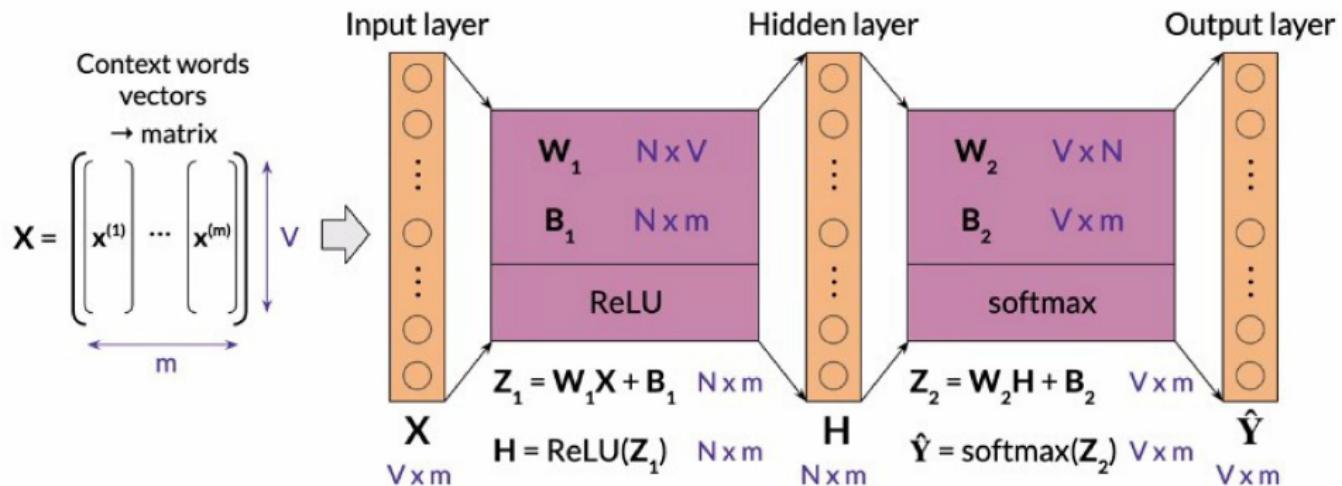
$$\begin{bmatrix} \mathbf{b}_1 \end{bmatrix} \rightarrow \mathbf{B}_1 = \begin{bmatrix} \mathbf{b}_1 \\ \vdots \\ \mathbf{b}_1 \end{bmatrix}_{m \times N} \quad \text{broadcasting}$$



Wordembeddings

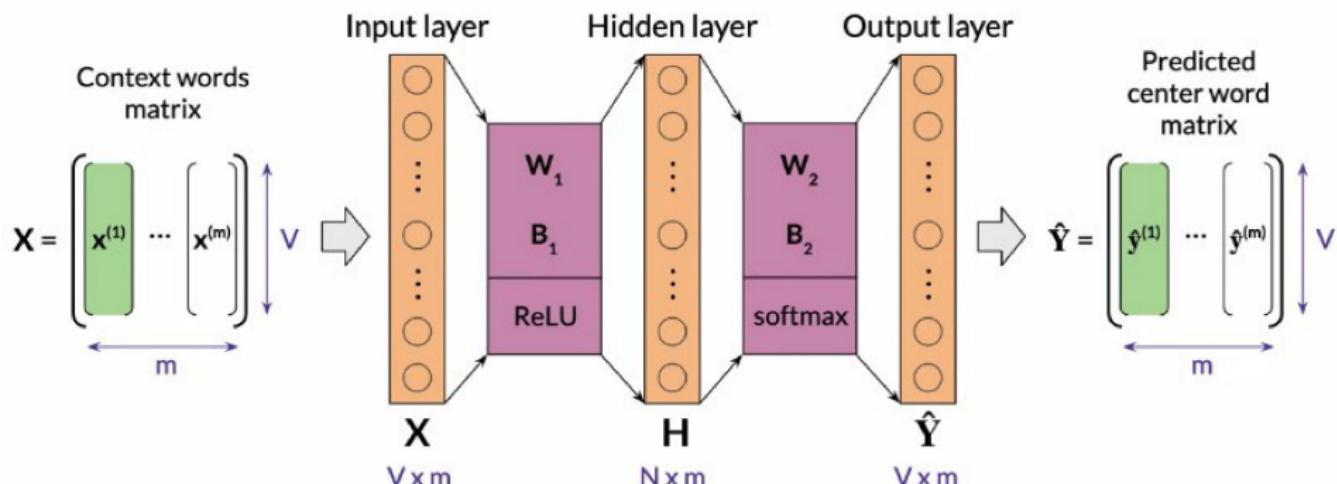
Dimensions (batch input)

$$\begin{bmatrix} \mathbf{b}_1 \end{bmatrix} \rightarrow \mathbf{B}_1 = \begin{bmatrix} \mathbf{b}_1 \\ \vdots \\ \mathbf{b}_1 \end{bmatrix}_{m \times N} \quad \text{broadcasting}$$

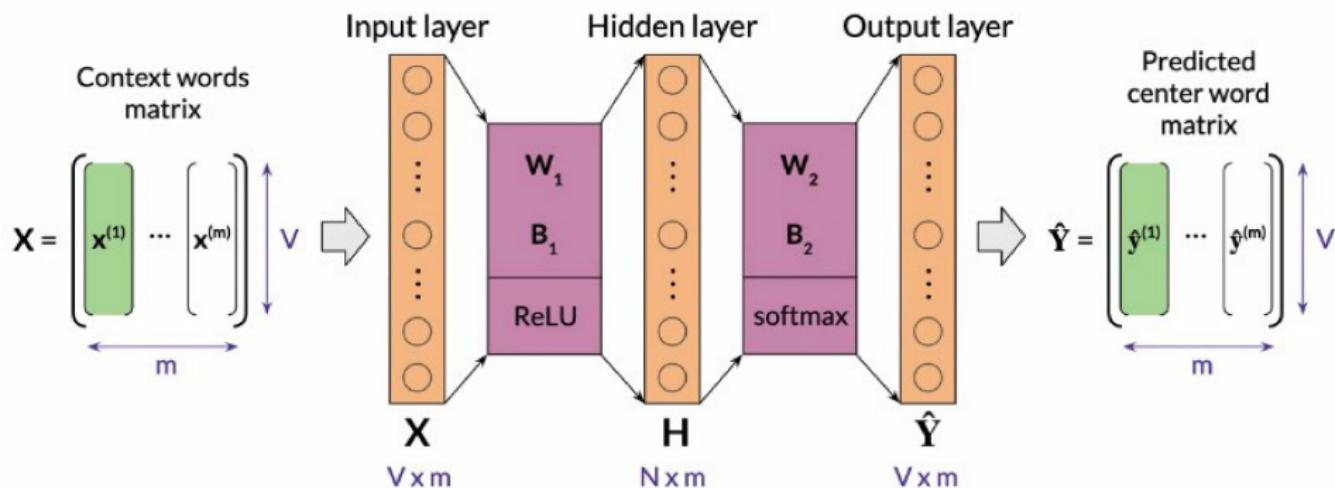


Wordembeddings

Dimensions (batch input)



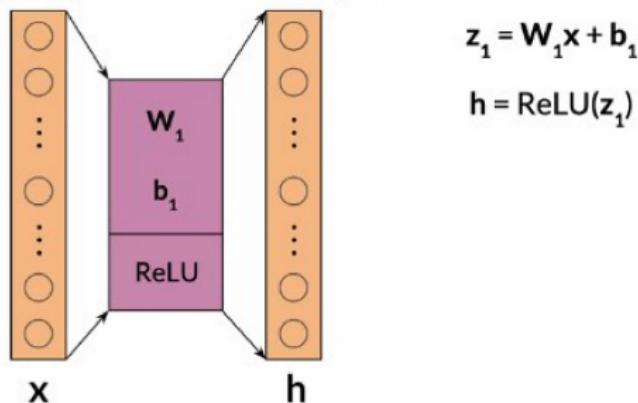
Dimensions (batch input)



Rectified Linear Unit (ReLU)

Rectified Linear Unit (ReLU)

Input layer Hidden layer



$$z_1 = \mathbf{W}_1 \mathbf{x} + \mathbf{b}_1$$

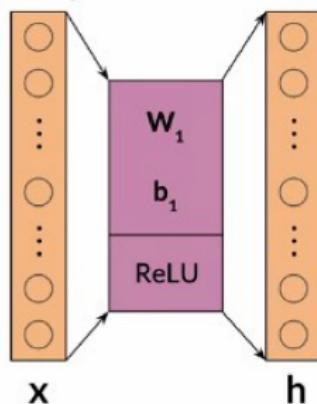
$$h = \text{ReLU}(z_1)$$

Rectified Linear Unit (ReLU)

Input layer

Hidden layer

$$\text{ReLU}(x) = \max(0, x)$$



$$z_1 = \mathbf{W}_1 \mathbf{x} + \mathbf{b}_1$$

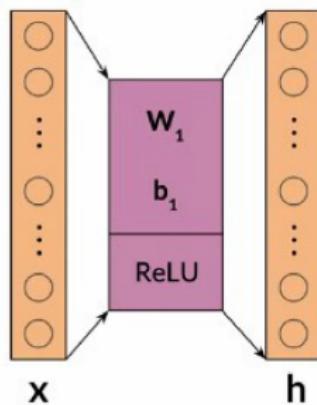
$$h = \text{ReLU}(z_1)$$

Rectified Linear Unit (ReLU)

Input layer

Hidden layer

$$\text{ReLU}(x) = \max(0, x)$$

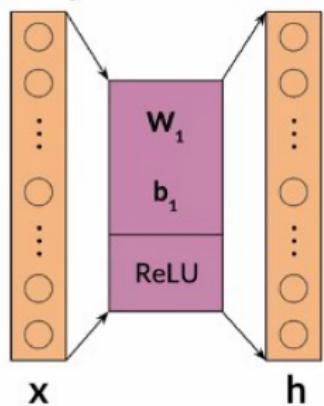


$$z_1 = \mathbf{W}_1 \mathbf{x} + \mathbf{b}_1$$

$$h = \text{ReLU}(z_1)$$

Rectified Linear Unit (ReLU)

Input layer Hidden layer

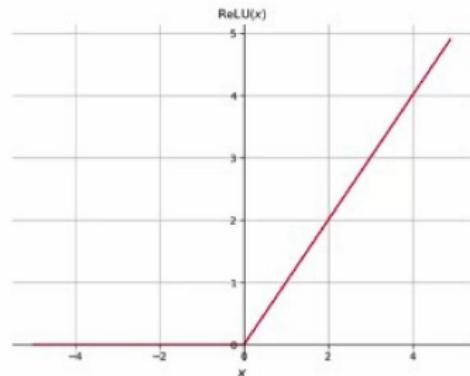


$$z_1 = \mathbf{W}_1 \mathbf{x} + \mathbf{b}_1$$

$$\mathbf{h} = \text{ReLU}(z_1)$$

This diagram shows the application of the ReLU function. On the left, a vertical vector z_1 is shown with entries: 5.1, -0.3, \vdots , -4.6, 0.2. An arrow labeled "ReLU" points to the right, where the resulting vector h is shown with entries: 0, \vdots , 0. The ReLU function is defined as $\text{ReLU}(x) = \max(0, x)$.

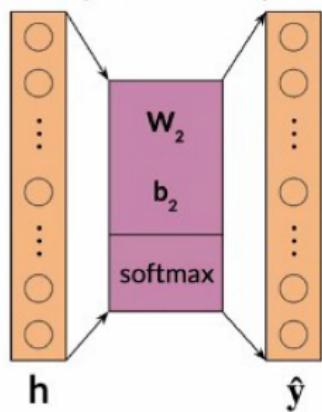
$$\text{ReLU}(x) = \max(0, x)$$



Wordembeddings

Softmax

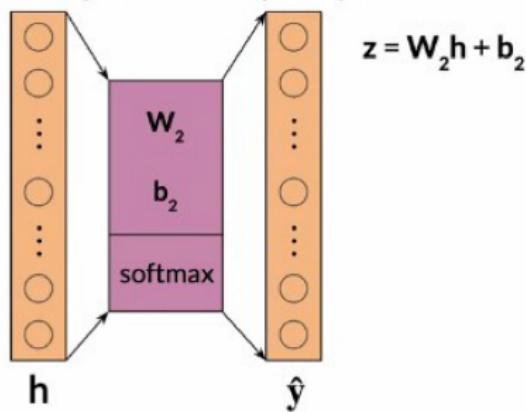
Hidden layer Output layer



Softmax

Hidden layer

Output layer

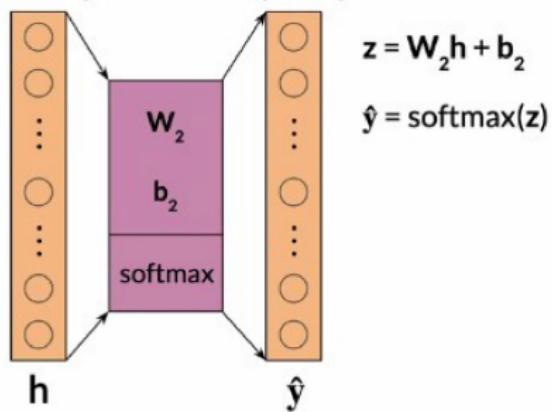


Wordembeddings

Softmax

Hidden layer

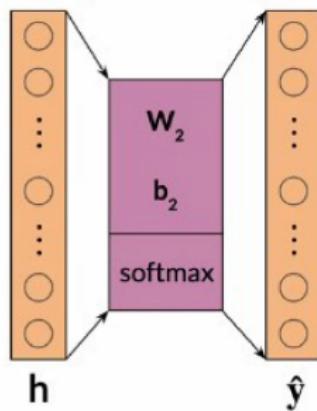
Output layer



Wordembeddings

Softmax

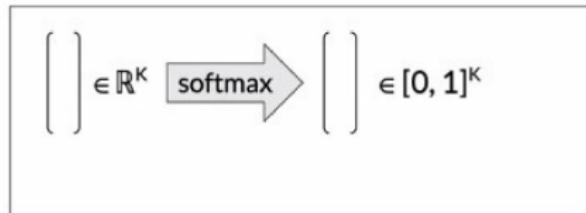
Hidden layer



Output layer

$$z = \mathbf{W}_2 h + \mathbf{b}_2$$

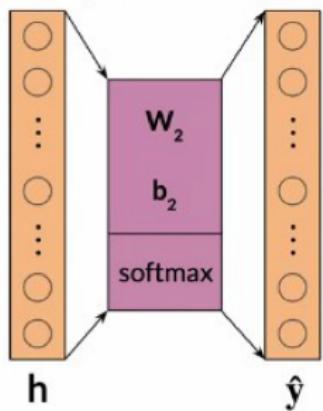
$$\hat{y} = \text{softmax}(z)$$



Wordembeddings

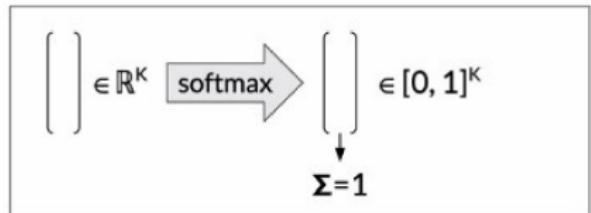
Softmax

Hidden layer



Output layer

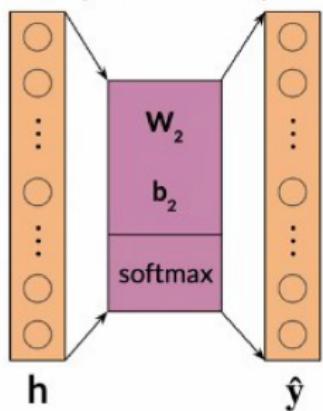
$$z = \mathbf{W}_2 \mathbf{h} + \mathbf{b}_2$$
$$\hat{\mathbf{y}} = \text{softmax}(z)$$



Wordembeddings

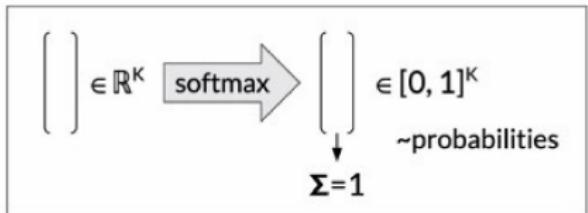
Softmax

Hidden layer



Output layer

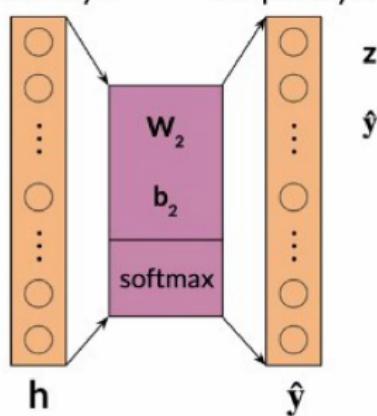
$$z = \mathbf{W}_2 \mathbf{h} + \mathbf{b}_2$$
$$\hat{\mathbf{y}} = \text{softmax}(z)$$



Wordembeddings

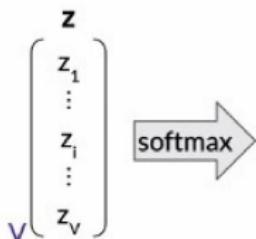
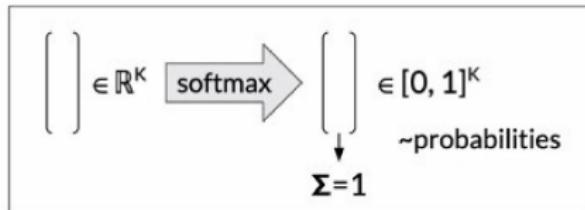
Softmax

Hidden layer



$$z = W_2 h + b_2$$

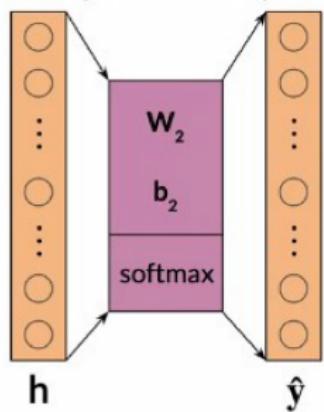
$$\hat{y} = \text{softmax}(z)$$



Wordembeddings

Softmax

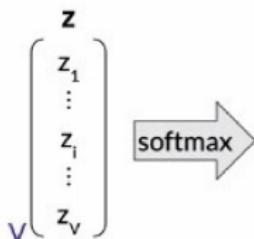
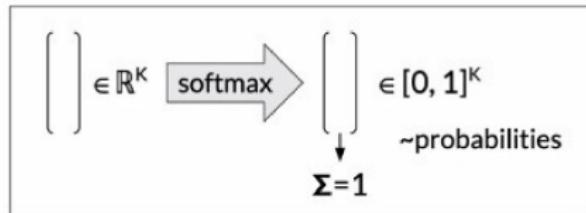
Hidden layer



Output layer

$$z = W_2 h + b_2$$

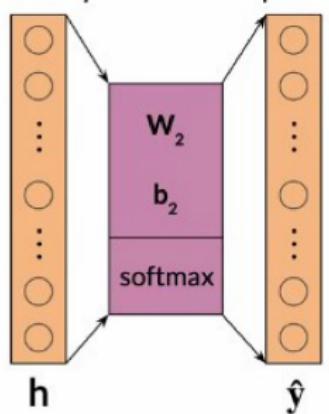
$$\hat{y} = \text{softmax}(z)$$



Wordembeddings

Softmax

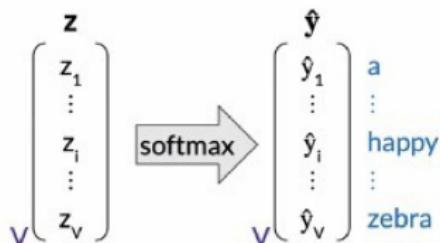
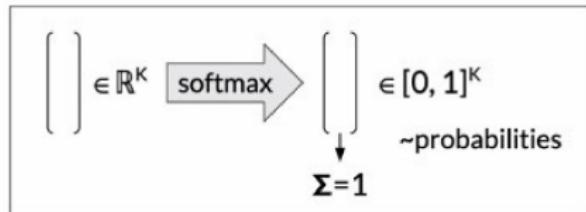
Hidden layer



Output layer

$$z = W_2 h + b_2$$

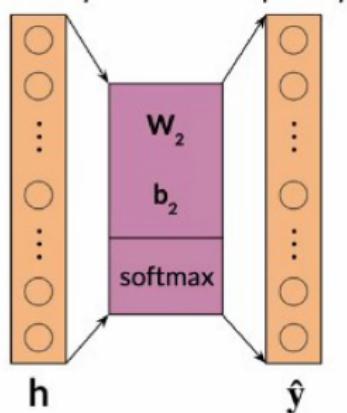
$$\hat{y} = \text{softmax}(z)$$



Wordembeddings

Softmax

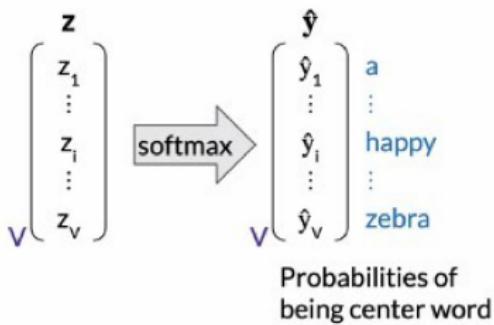
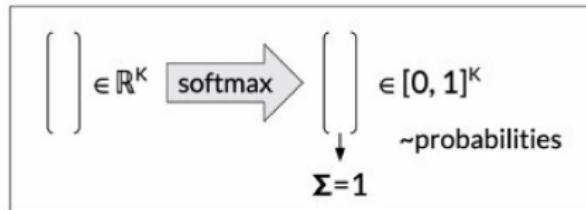
Hidden layer



Output layer

$$z = W_2 h + b_2$$

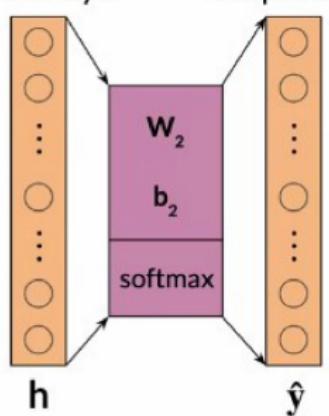
$$\hat{y} = \text{softmax}(z)$$



Wordembeddings

Softmax

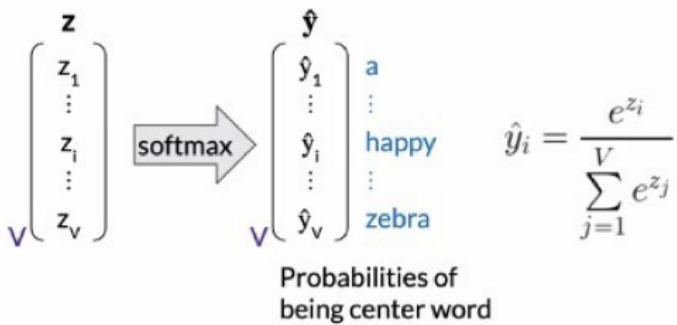
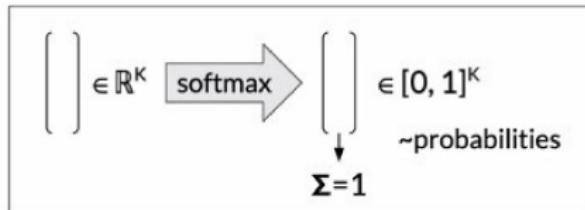
Hidden layer



Output layer

$$z = W_2 h + b_2$$

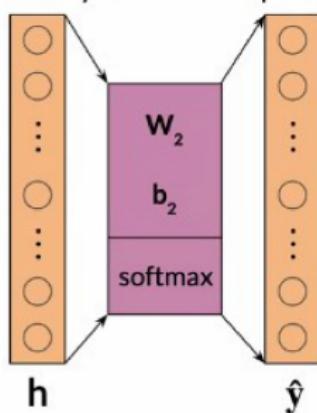
$$\hat{y} = \text{softmax}(z)$$



Wordembeddings

Softmax

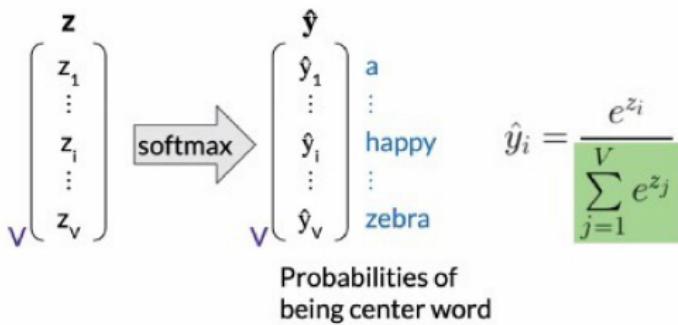
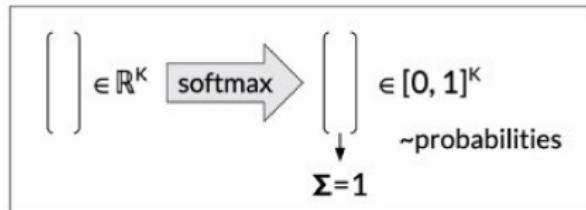
Hidden layer



Output layer

$$z = \mathbf{W}_2 \mathbf{h} + \mathbf{b}_2$$

$$\hat{y} = \text{softmax}(z)$$



Wordembeddings

Softmax: example

$$\hat{y}_i = \frac{e^{z_i}}{\sum_{j=1}^V e^{z_j}}$$

z

$$\begin{pmatrix} 9 \\ 8 \\ 11 \\ 10 \\ 8.5 \end{pmatrix}$$

Wordembeddings

Softmax: example

$$\hat{y}_i = \frac{e^{z_i}}{\sum_{j=1}^V e^{z_j}}$$

z	exp(z)
9	8103
8	2981
11	59874
10	22026
8.5	4915



Wordembeddings

Softmax: example

$$\hat{y}_i = \frac{e^{z_i}}{\sum_{j=1}^V e^{z_j}}$$

z

9
8
11
10
8.5

exp(z)

8103
2981
59874
22026
4915

exp

Σ=97899

The diagram illustrates the softmax function. On the left, a vertical vector labeled 'z' contains values 9, 8, 11, 10, and 8.5. An arrow labeled 'exp' points to the right, where another vertical vector labeled 'exp(z)' contains the corresponding exponential values: 8103, 2981, 59874, 22026, and 4915. Below this second vector is a downward-pointing arrow, indicating the summation of all these values. The result of this summation is given as Σ=97899.

Wordembeddings

Softmax: example

$$\hat{y}_i = \frac{e^{z_i}}{\sum_{j=1}^V e^{z_j}}$$

z

9
8
11
10
8.5

exp(z)

8103
2981
59874
22026
4915

exp

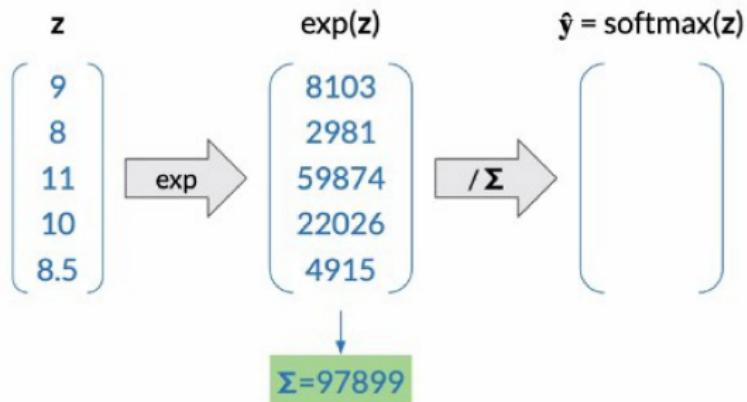
$\Sigma = 97899$

$\Sigma = 97899$

Wordembeddings

Softmax: example

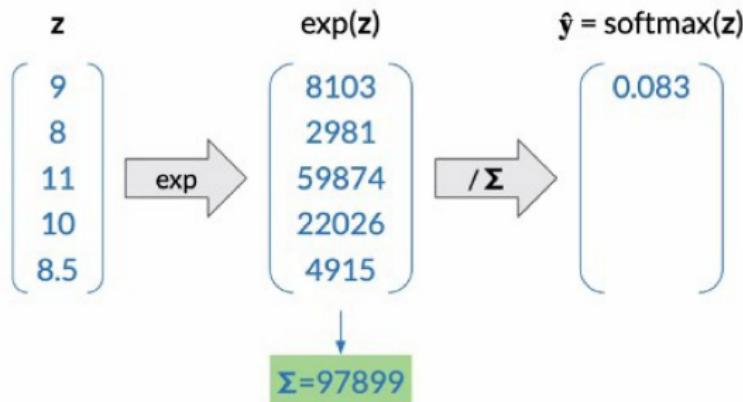
$$\hat{y}_i = \frac{e^{z_i}}{\sum_{j=1}^V e^{z_j}}$$



Wordembeddings

Softmax: example

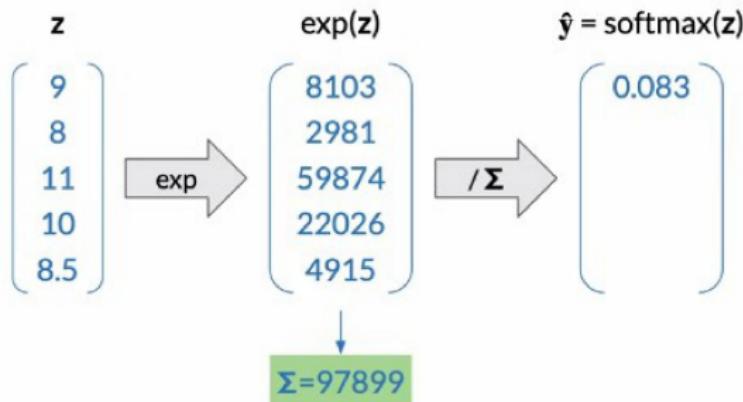
$$\hat{y}_i = \frac{e^{z_i}}{\sum_{j=1}^V e^{z_j}}$$



Wordembeddings

Softmax: example

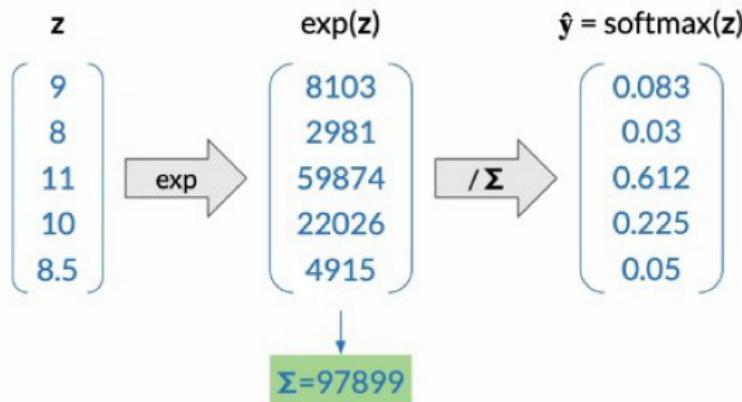
$$\hat{y}_i = \frac{e^{z_i}}{\sum_{j=1}^V e^{z_j}}$$



Wordembeddings

Softmax: example

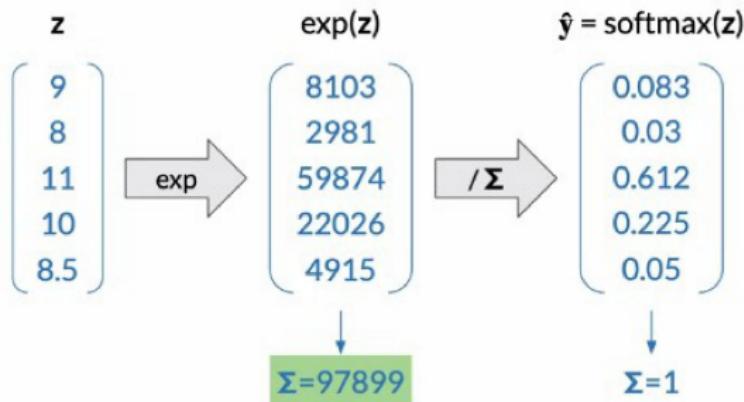
$$\hat{y}_i = \frac{e^{z_i}}{\sum_{j=1}^V e^{z_j}}$$



Wordembeddings

Softmax: example

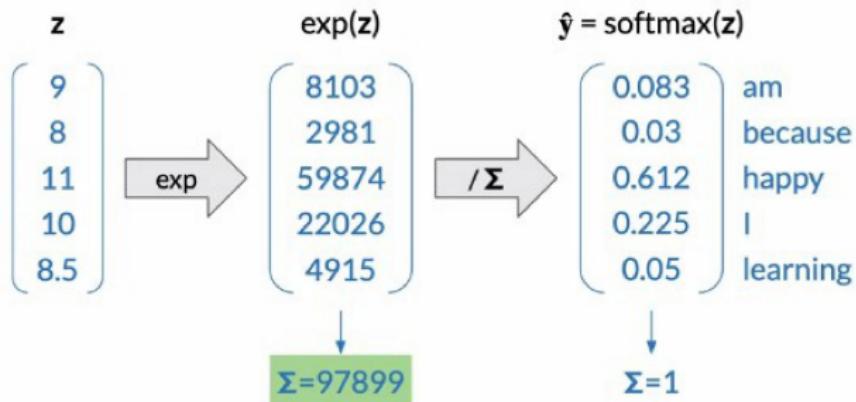
$$\hat{y}_i = \frac{e^{z_i}}{\sum_{j=1}^V e^{z_j}}$$



Wordembeddings

Softmax: example

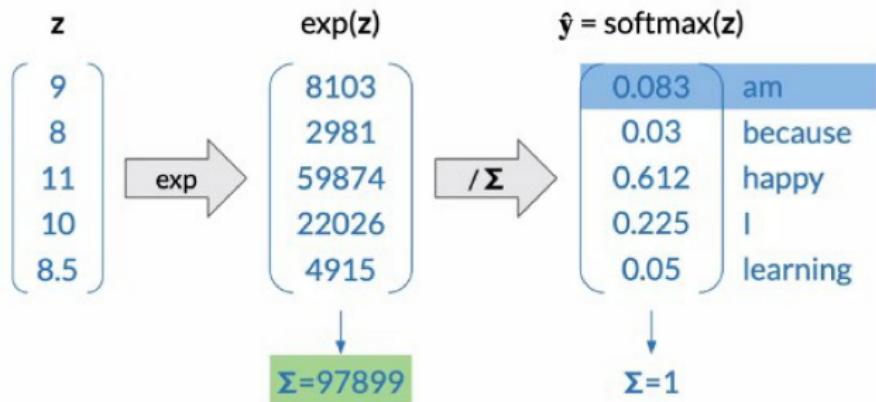
$$\hat{y}_i = \frac{e^{z_i}}{\sum_{j=1}^V e^{z_j}}$$



Wordembeddings

Softmax: example

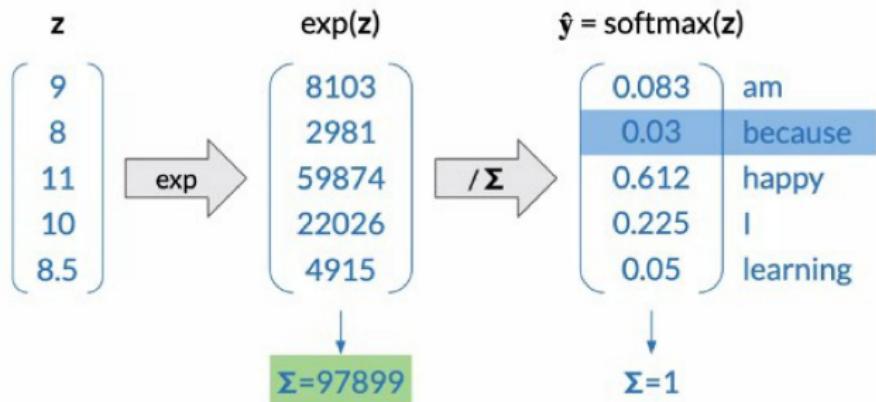
$$\hat{y}_i = \frac{e^{z_i}}{\sum_{j=1}^V e^{z_j}}$$



Wordembeddings

Softmax: example

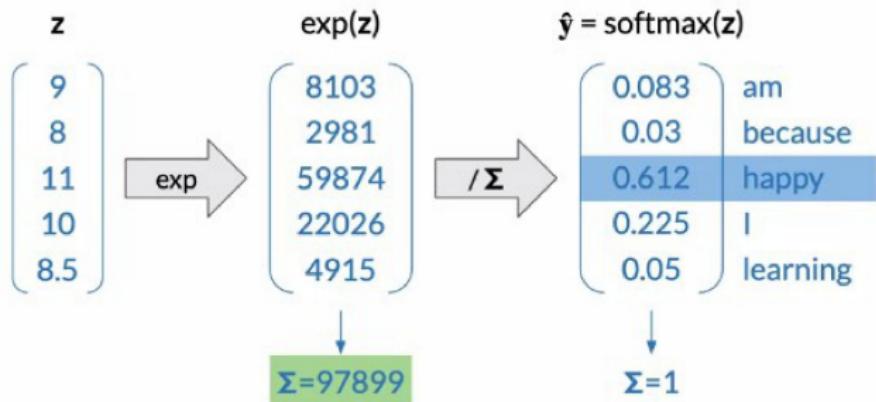
$$\hat{y}_i = \frac{e^{z_i}}{\sum_{j=1}^V e^{z_j}}$$



Wordembeddings

Softmax: example

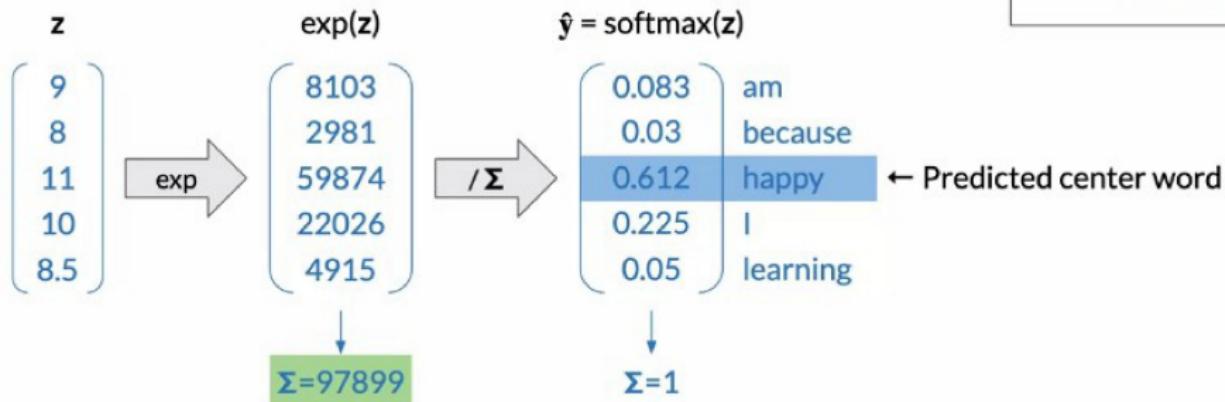
$$\hat{y}_i = \frac{e^{z_i}}{\sum_{j=1}^V e^{z_j}}$$



Wordembeddings

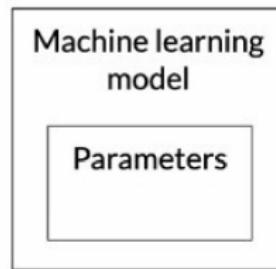
Softmax: example

$$\hat{y}_i = \frac{e^{z_i}}{\sum_{j=1}^V e^{z_j}}$$



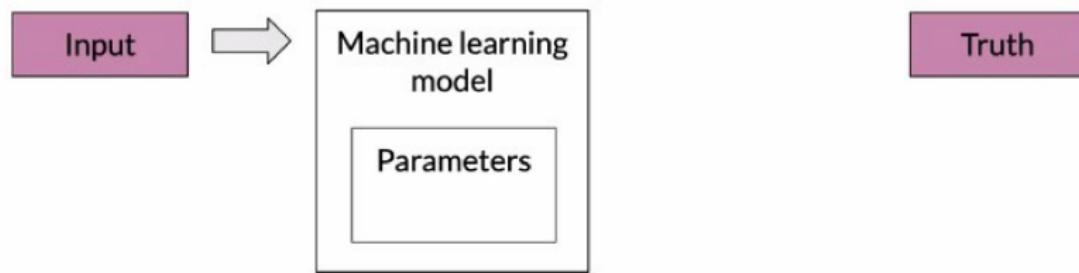
Wordembeddings

Loss



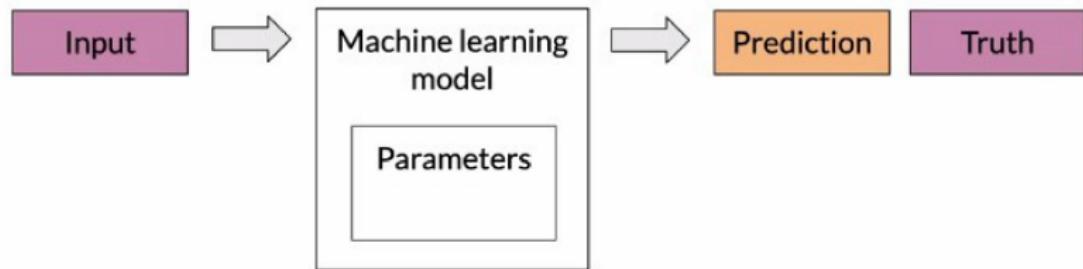
Wordembeddings

Loss



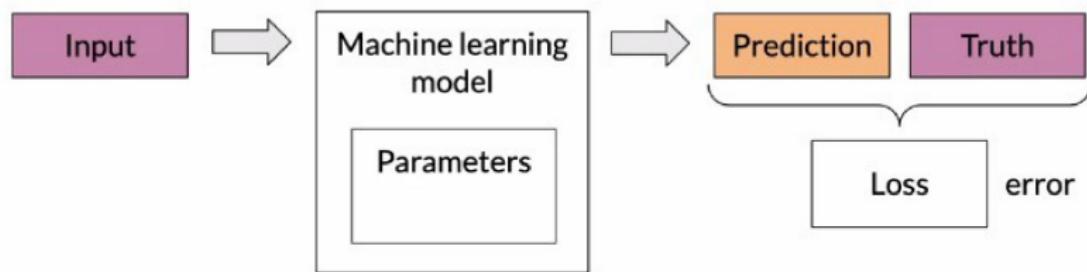
Wordembeddings

Loss



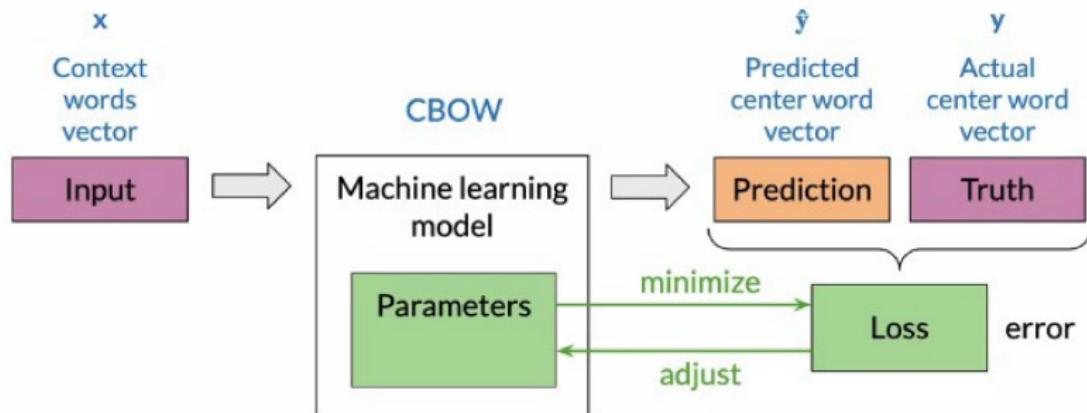
Wordembeddings

Loss



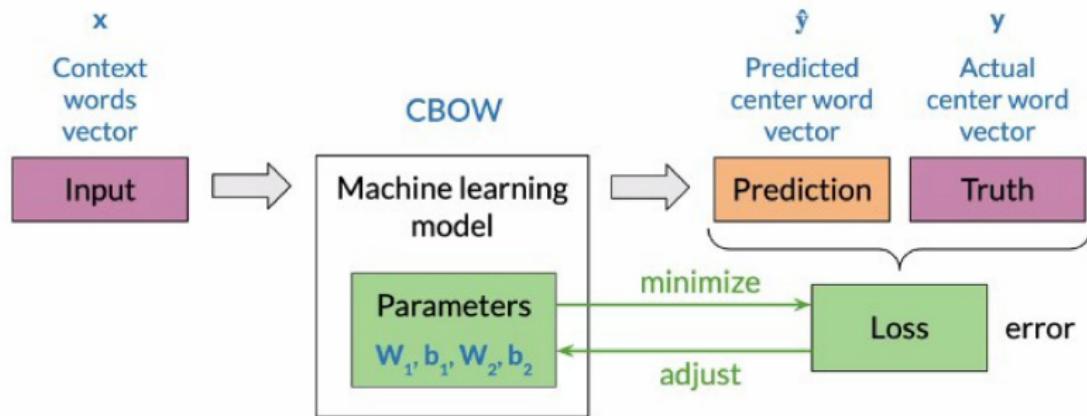
Wordembeddings

Loss



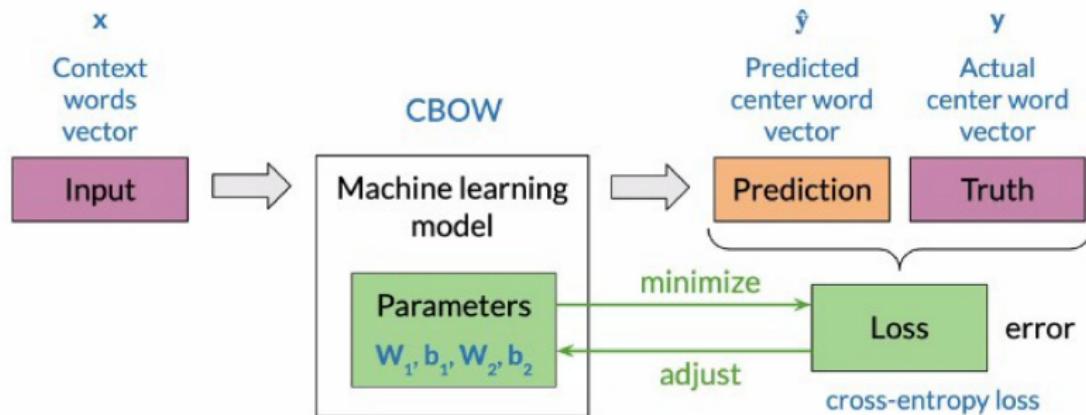
Wordembeddings

Loss



Wordembeddings

Loss



Cross-entropy loss

$$\begin{array}{ll} \text{Actual} & \text{Predicted} \\ \mathbf{y} = \begin{pmatrix} y_1 \\ \vdots \\ y_v \end{pmatrix} & \hat{\mathbf{y}} = \begin{pmatrix} \hat{y}_1 \\ \vdots \\ \hat{y}_v \end{pmatrix} \end{array}$$

Wordembeddings

Cross-entropy loss

$$J = - \sum_{k=1}^V y_k \log \hat{y}_k$$

Actual $\mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_v \end{bmatrix}$	Predicted $\hat{\mathbf{y}} = \begin{bmatrix} \hat{y}_1 \\ \vdots \\ \hat{y}_v \end{bmatrix}$
---	--

Wordembeddings

Cross-entropy loss

$$J = - \sum_{k=1}^V y_k \log \hat{y}_k$$

Actual $\mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_v \end{bmatrix}$	Predicted $\hat{\mathbf{y}} = \begin{bmatrix} \hat{y}_1 \\ \vdots \\ \hat{y}_v \end{bmatrix}$
---	--

I am happy because I am learning

Wordembeddings

Cross-entropy loss

$$J = - \sum_{k=1}^V y_k \log \hat{y}_k$$

$$\mathbf{y} = \begin{cases} 0 & \text{am} \\ 0 & \text{because} \\ 1 & \text{happy} \\ 0 & \text{I} \\ 0 & \text{learning} \end{cases}$$

Actual	Predicted
$\mathbf{y} = \begin{pmatrix} y_1 \\ \vdots \\ y_V \end{pmatrix}$	$\hat{\mathbf{y}} = \begin{pmatrix} \hat{y}_1 \\ \vdots \\ \hat{y}_V \end{pmatrix}$

I am happy because I am learning

Wordembeddings

Cross-entropy loss

$$J = - \sum_{k=1}^V y_k \log \hat{y}_k$$

Actual $\mathbf{y} = \begin{pmatrix} y_1 \\ \vdots \\ y_v \end{pmatrix}$	Predicted $\hat{\mathbf{y}} = \begin{pmatrix} \hat{y}_1 \\ \vdots \\ \hat{y}_v \end{pmatrix}$
---	--

I am happy because I am learning

\mathbf{y}	$\hat{\mathbf{y}}$	
0	am	0.083
0	because	0.03
1	happy	0.611
0	I	0.225
0	learning	0.05

Wordembeddings

Cross-entropy loss

$$J = - \sum_{k=1}^V y_k \log \hat{y}_k$$

Actual $\mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_v \end{bmatrix}$	Predicted $\hat{\mathbf{y}} = \begin{bmatrix} \hat{y}_1 \\ \vdots \\ \hat{y}_v \end{bmatrix}$
---	--

I am happy because I am learning

y		\hat{y}
0	am	0.083
0	because	0.03
1	happy	0.611
0	I	0.225
0	learning	0.05

Wordembeddings

Cross-entropy loss

$$J = - \sum_{k=1}^V y_k \log \hat{y}_k$$

Actual $\mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_v \end{bmatrix}$	Predicted $\hat{\mathbf{y}} = \begin{bmatrix} \hat{y}_1 \\ \vdots \\ \hat{y}_v \end{bmatrix}$
---	--

I am happy because I am learning

\mathbf{y}	$\hat{\mathbf{y}}$	$\log(\hat{\mathbf{y}})$	
0	am	0.083	-2.49
0	because	0.03	-3.49
1	happy	0.611	-0.49
0	I	0.225	-1.49
0	learning	0.05	-2.49

log

Wordembeddings

Cross-entropy loss

$$J = - \sum_{k=1}^V y_k \log \hat{y}_k$$

Actual $\mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_v \end{bmatrix}$	Predicted $\hat{\mathbf{y}} = \begin{bmatrix} \hat{y}_1 \\ \vdots \\ \hat{y}_v \end{bmatrix}$
---	--

I am happy because I am learning

\mathbf{y}	$\hat{\mathbf{y}}$	$\log(\hat{\mathbf{y}})$	$\mathbf{y} \odot \log(\hat{\mathbf{y}})$
$\begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$ am	$\begin{bmatrix} 0.083 \\ 0.03 \\ 0.611 \\ 0.225 \\ 0.05 \end{bmatrix}$	$\begin{bmatrix} -2.49 \\ -3.49 \\ -0.49 \\ -1.49 \\ -2.49 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \\ -0.49 \\ 0 \\ 0 \end{bmatrix}$
because			
happy			
I			
learning			

log

$\odot \mathbf{y}$

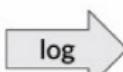
Wordembeddings

Cross-entropy loss

$$J = - \sum_{k=1}^V y_k \log \hat{y}_k$$

Actual $\mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_v \end{bmatrix}$	Predicted $\hat{\mathbf{y}} = \begin{bmatrix} \hat{y}_1 \\ \vdots \\ \hat{y}_v \end{bmatrix}$
---	--

I am happy because I am learning

\mathbf{y}					
$\begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$	am	$\begin{bmatrix} 0.083 \\ 0.03 \\ 0.611 \\ 0.225 \\ 0.05 \end{bmatrix}$		$\log(\hat{\mathbf{y}})$	$\mathbf{y} \circ \log(\hat{\mathbf{y}})$
	because			$\begin{bmatrix} -2.49 \\ -3.49 \\ -0.49 \\ -1.49 \\ -2.49 \end{bmatrix}$	
	happy				$\begin{bmatrix} 0 \\ 0 \\ -0.49 \\ 0 \\ 0 \end{bmatrix}$
	I				
	learning				

Wordembeddings

Cross-entropy loss

$$J = - \sum_{k=1}^V y_k \log \hat{y}_k$$

Actual $\mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_v \end{bmatrix}$	Predicted $\hat{\mathbf{y}} = \begin{bmatrix} \hat{y}_1 \\ \vdots \\ \hat{y}_v \end{bmatrix}$
---	--

I am happy because I am learning

The diagram illustrates the calculation of cross-entropy loss for the sentence "I am happy because I am learning". It shows the forward pass through the network, starting from the input vector \mathbf{y} and ending at the loss function $J = -\sum$.

Input: The input vector \mathbf{y} is given by:

$$\mathbf{y} = \begin{bmatrix} 0 & \text{am} \\ 0 & \text{because} \\ 1 & \text{happy} \\ 0 & \text{I} \\ 0 & \text{learning} \end{bmatrix}$$

Output: The predicted probabilities $\hat{\mathbf{y}}$ are given by:

$$\hat{\mathbf{y}} = \begin{bmatrix} 0.083 \\ 0.03 \\ 0.611 \\ 0.225 \\ 0.05 \end{bmatrix}$$

Log Step: The log of the predicted probabilities is calculated:

$$\log(\hat{\mathbf{y}}) = \begin{bmatrix} -2.49 \\ -3.49 \\ -0.49 \\ -1.49 \\ -2.49 \end{bmatrix}$$

Element-wise Product: The element-wise product of \mathbf{y} and $\log(\hat{\mathbf{y}})$ is calculated:

$$\mathbf{y} \odot \log(\hat{\mathbf{y}}) = \begin{bmatrix} 0 \\ 0 \\ -0.49 \\ 0 \\ 0 \end{bmatrix}$$

Summation: The summation step involves summing the negative values of the element-wise product:

$$-\sum = -(-0.49) = 0.49$$

Final Loss: The final cross-entropy loss is $J = 0.49$.

Wordembeddings

Cross-entropy loss

$$J = - \sum_{k=1}^V y_k \log \hat{y}_k$$

y
0
0
1
0
0

am
because
happy
I
learning

Wordembeddings

Cross-entropy loss

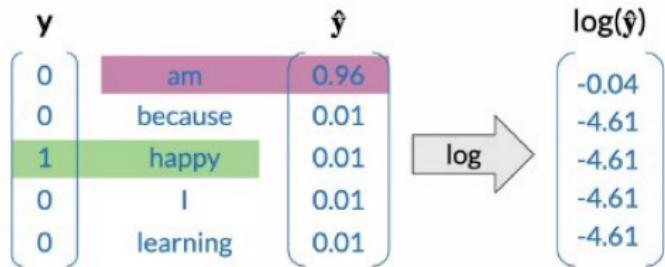
$$J = - \sum_{k=1}^V y_k \log \hat{y}_k$$

y		\hat{y}
0	am	0.96
0	because	0.01
1	happy	0.01
0	I	0.01
0	learning	0.01

Wordembeddings

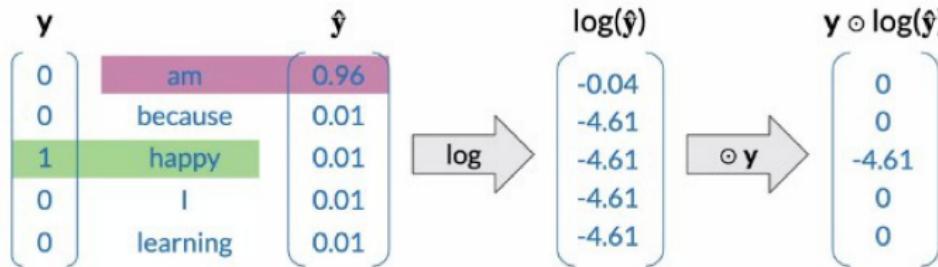
Cross-entropy loss

$$J = - \sum_{k=1}^V y_k \log \hat{y}_k$$



Cross-entropy loss

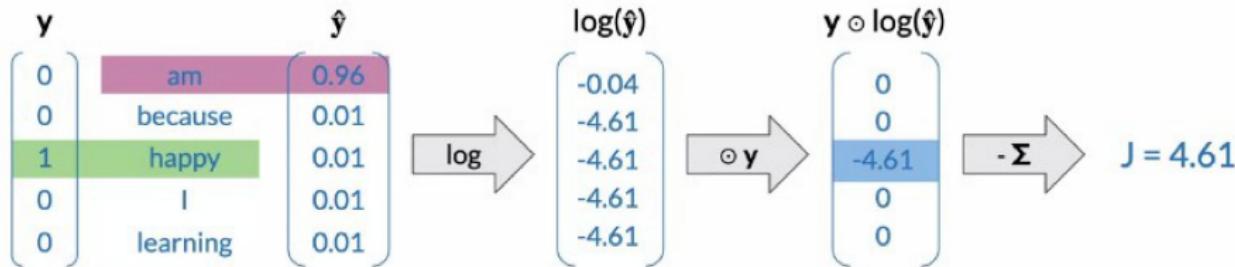
$$J = - \sum_{k=1}^V y_k \log \hat{y}_k$$



Wordembeddings

Cross-entropy loss

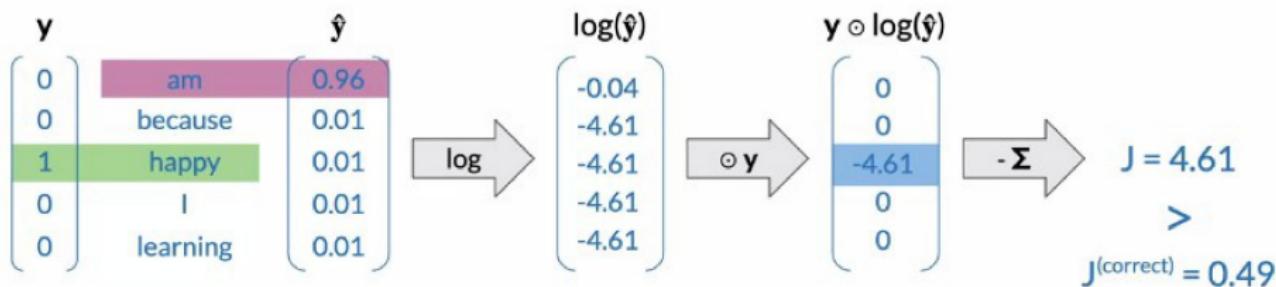
$$J = - \sum_{k=1}^V y_k \log \hat{y}_k$$



Wordembeddings

Cross-entropy loss

$$J = - \sum_{k=1}^V y_k \log \hat{y}_k$$



Wordembeddings

Cross-entropy loss

$$J = - \sum_{k=1}^V y_k \log \hat{y}_k$$

$$J = -\log \hat{y}_{\text{actual word}}$$

y		\hat{y}
0	am	0.96
0	because	0.01
1	happy	0.01
0	I	0.01
0	learning	0.01

$\rightarrow J = 4.61$

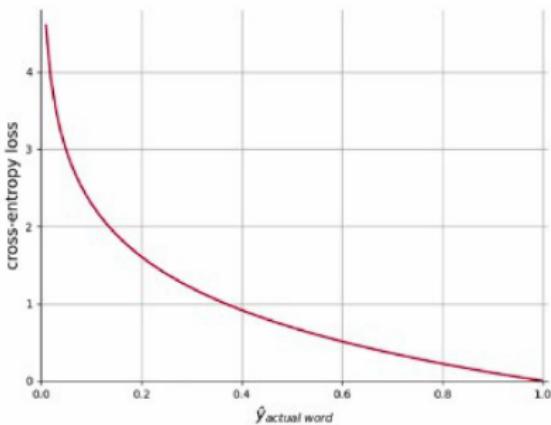
Wordembeddings

Cross-entropy loss

$$J = -\log \hat{y}_{\text{actual word}}$$

y		\hat{y}	
0	am	0.96	
0	because	0.01	
1	happy	0.01	
0	I	0.01	
0	learning	0.01	

$$\rightarrow J = 4.61$$



Wordembeddings

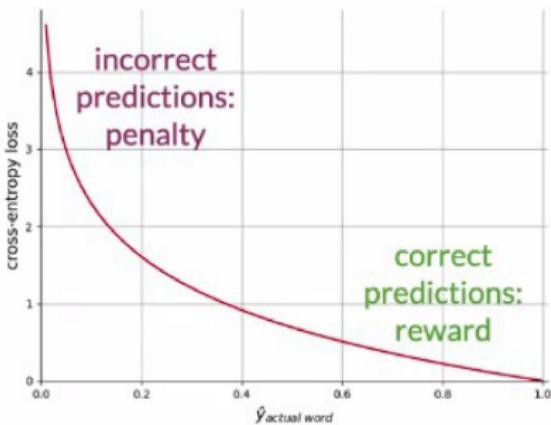
Cross-entropy loss

$$J = -\log \hat{y}_{\text{actual word}}$$

y		\hat{y}	
0	am	0.96	
0	because	0.01	
1	happy	0.01	
0	I	0.01	
0	learning	0.01	

$$\rightarrow J = 4.61$$

$$J = - \sum_{k=1}^V y_k \log \hat{y}_k$$



Training process

- Forward propagation

Training process

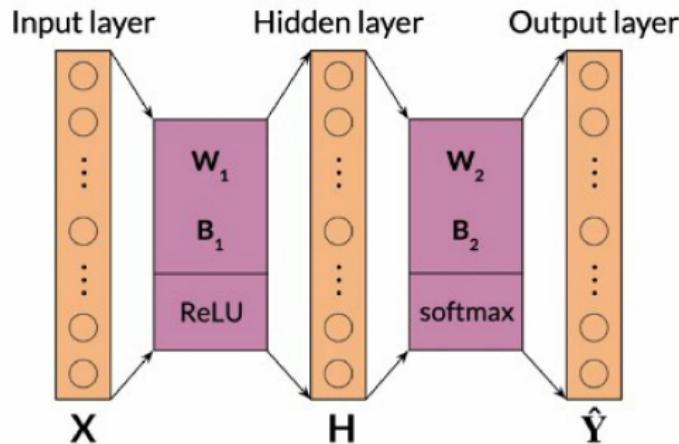
- Forward propagation
- Cost

Training process

- Forward propagation
- Cost
- Backpropagation and gradient descent

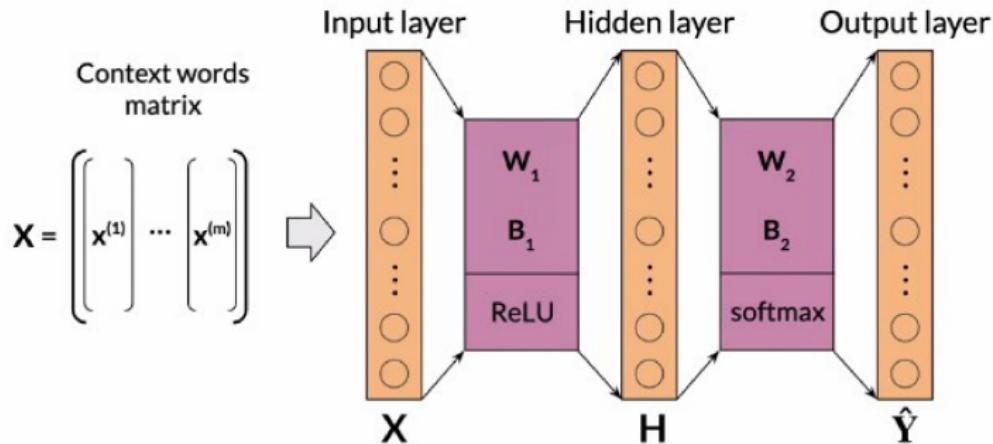
Wordembeddings

Forward propagation



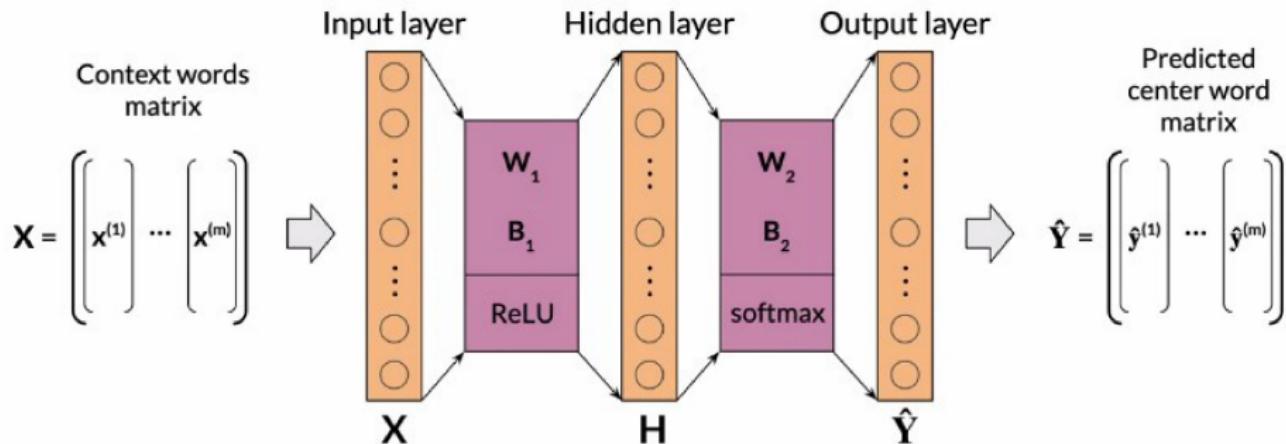
Wordembeddings

Forward propagation



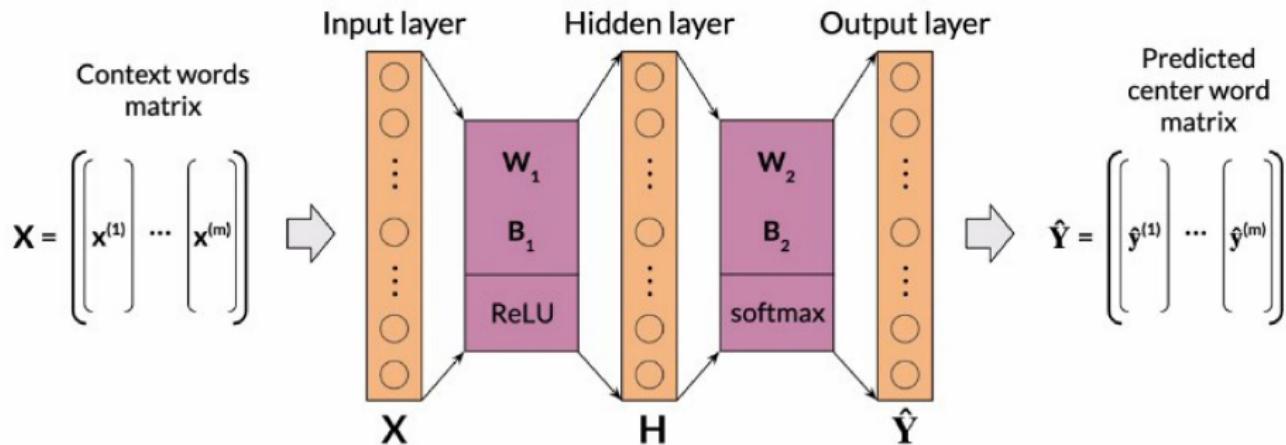
Wordembeddings

Forward propagation



Wordembeddings

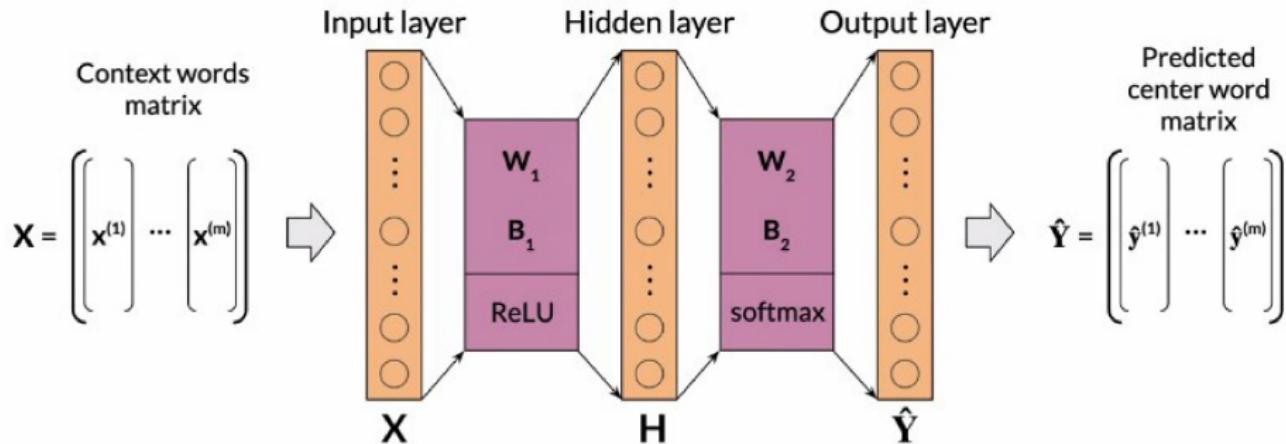
Forward propagation



Wordembeddings

Forward propagation

$$\begin{aligned} Z_1 &= W_1 X + B_1 & Z_2 &= W_2 H + B_2 \\ H &= \text{ReLU}(Z_1) & \hat{Y} &= \text{softmax}(Z_2) \end{aligned}$$



Wordembeddings

Cost

$$J = - \sum_{k=1}^V y_k \log \hat{y}_k$$

Wordembeddings

Cost

Cost: mean of losses

$$J = - \sum_{k=1}^V y_k \log \hat{y}_k$$

Wordembeddings

Cost

Cost: mean of losses

$$J_{batch} = -\frac{1}{m} \sum_{i=1}^m \sum_{j=1}^V y_j^{(i)} \log \hat{y}_j^{(i)}$$

Predicted
center word
matrix

$$\hat{\mathbf{Y}} = \begin{pmatrix} \hat{\mathbf{y}}^{(1)} & \cdots & \hat{\mathbf{y}}^{(m)} \end{pmatrix}$$

Actual center
word matrix

$$\mathbf{Y} = \begin{pmatrix} \mathbf{y}^{(1)} & \cdots & \mathbf{y}^{(m)} \end{pmatrix}$$

$$J = - \sum_{k=1}^V y_k \log \hat{y}_k$$

Wordembeddings

Cost

Cost: mean of losses

$$J_{batch} = -\frac{1}{m} \sum_{i=1}^m \sum_{j=1}^V y_j^{(i)} \log \hat{y}_j^{(i)}$$

Predicted
center word
matrix

$$\hat{\mathbf{Y}} = \begin{pmatrix} \hat{\mathbf{y}}^{(1)} & \cdots & \hat{\mathbf{y}}^{(m)} \end{pmatrix}$$

Actual center
word matrix

$$\mathbf{Y} = \begin{pmatrix} \mathbf{y}^{(1)} & \cdots & \mathbf{y}^{(m)} \end{pmatrix}$$

$$J_{batch} = -\frac{1}{m} \sum_{i=1}^m J^{(i)}$$

$$J = -\sum_{k=1}^V y_k \log \hat{y}_k$$

Minimizing the cost

Minimizing the cost

- Backpropagation: calculate partial derivatives of cost with respect to weights and biases

$$\frac{\partial J_{batch}}{\partial \mathbf{W}_1}, \frac{\partial J_{batch}}{\partial \mathbf{W}_2}, \frac{\partial J_{batch}}{\partial \mathbf{b}_1}, \frac{\partial J_{batch}}{\partial \mathbf{b}_2}$$

Minimizing the cost

$$J_{batch} = f(\mathbf{W}_1, \mathbf{W}_2, \mathbf{b}_1, \mathbf{b}_2)$$

- Backpropagation: calculate partial derivatives of cost with respect to weights and biases

$$\frac{\partial J_{batch}}{\partial \mathbf{W}_1}, \frac{\partial J_{batch}}{\partial \mathbf{W}_2}, \frac{\partial J_{batch}}{\partial \mathbf{b}_1}, \frac{\partial J_{batch}}{\partial \mathbf{b}_2}$$

Minimizing the cost

- Backpropagation: calculate partial derivatives of cost with respect to weights and biases

$$\frac{\partial J_{batch}}{\partial \mathbf{W}_1}, \frac{\partial J_{batch}}{\partial \mathbf{W}_2}, \frac{\partial J_{batch}}{\partial \mathbf{b}_1}, \frac{\partial J_{batch}}{\partial \mathbf{b}_2}$$

- Gradient descent: update weights and biases

Backpropagation

Backpropagation

$$\frac{\partial J_{batch}}{\partial \mathbf{W}_1}$$

$$\frac{\partial J_{batch}}{\partial \mathbf{W}_2}$$

$$\frac{\partial J_{batch}}{\partial \mathbf{b}_1}$$

$$\frac{\partial J_{batch}}{\partial \mathbf{b}_2}$$

Backpropagation

$$\frac{\partial J_{batch}}{\partial \mathbf{W}_1} = \frac{1}{m} \text{ReLU} \left(\mathbf{W}_2^\top (\hat{\mathbf{Y}} - \mathbf{Y}) \right) \mathbf{X}^\top$$

$$\frac{\partial J_{batch}}{\partial \mathbf{W}_2}$$

$$\frac{\partial J_{batch}}{\partial \mathbf{b}_1}$$

$$\frac{\partial J_{batch}}{\partial \mathbf{b}_2}$$

Backpropagation

$$\frac{\partial J_{batch}}{\partial \mathbf{W}_1} = \frac{1}{m} \text{ReLU} \left(\mathbf{W}_2^\top (\hat{\mathbf{Y}} - \mathbf{Y}) \right) \mathbf{X}^\top$$

$$\frac{\partial J_{batch}}{\partial \mathbf{W}_2} = \frac{1}{m} (\hat{\mathbf{Y}} - \mathbf{Y}) \mathbf{H}^\top$$

$$\frac{\partial J_{batch}}{\partial \mathbf{b}_1}$$

$$\frac{\partial J_{batch}}{\partial \mathbf{b}_2}$$

Backpropagation

$$\frac{\partial J_{batch}}{\partial \mathbf{W}_1} = \frac{1}{m} \text{ReLU} \left(\mathbf{W}_2^\top (\hat{\mathbf{Y}} - \mathbf{Y}) \right) \mathbf{X}^\top$$

$$\frac{\partial J_{batch}}{\partial \mathbf{W}_2} = \frac{1}{m} (\hat{\mathbf{Y}} - \mathbf{Y}) \mathbf{H}^\top$$

$$\frac{\partial J_{batch}}{\partial \mathbf{b}_1} = \frac{1}{m} \text{ReLU} \left(\mathbf{W}_2^\top (\hat{\mathbf{Y}} - \mathbf{Y}) \right) \mathbf{1}_m^\top$$

$$\frac{\partial J_{batch}}{\partial \mathbf{b}_2}$$

Backpropagation

$$\frac{\partial J_{batch}}{\partial \mathbf{W}_1} = \frac{1}{m} \text{ReLU} \left(\mathbf{W}_2^\top (\hat{\mathbf{Y}} - \mathbf{Y}) \right) \mathbf{X}^\top$$

$$\frac{\partial J_{batch}}{\partial \mathbf{W}_2} = \frac{1}{m} (\hat{\mathbf{Y}} - \mathbf{Y}) \mathbf{H}^\top$$

$$\frac{\partial J_{batch}}{\partial \mathbf{b}_1} = \frac{1}{m} \text{ReLU} \left(\mathbf{W}_2^\top (\hat{\mathbf{Y}} - \mathbf{Y}) \right) \mathbf{1}_m^\top$$

$$\frac{\partial J_{batch}}{\partial \mathbf{b}_2}$$

$$\mathbf{1}_m = \begin{bmatrix} 1, \dots, 1 \end{bmatrix}$$
$$\mathbf{A} \cdot \mathbf{1}_m^\top = \left[\begin{array}{c|c} \text{---} & \mathbf{1} \\ \hline \end{array} \right] \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} = \begin{bmatrix} \Sigma \end{bmatrix}$$

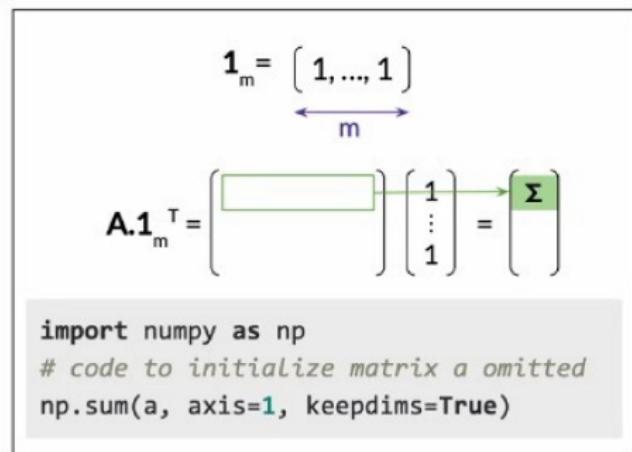
Backpropagation

$$\frac{\partial J_{batch}}{\partial \mathbf{W}_1} = \frac{1}{m} \text{ReLU} \left(\mathbf{W}_2^\top (\hat{\mathbf{Y}} - \mathbf{Y}) \right) \mathbf{X}^\top$$

$$\frac{\partial J_{batch}}{\partial \mathbf{W}_2} = \frac{1}{m} (\hat{\mathbf{Y}} - \mathbf{Y}) \mathbf{H}^\top$$

$$\frac{\partial J_{batch}}{\partial \mathbf{b}_1} = \frac{1}{m} \text{ReLU} \left(\mathbf{W}_2^\top (\hat{\mathbf{Y}} - \mathbf{Y}) \right) \mathbf{1}_m^\top$$

$$\frac{\partial J_{batch}}{\partial \mathbf{b}_2}$$



Wordembeddings

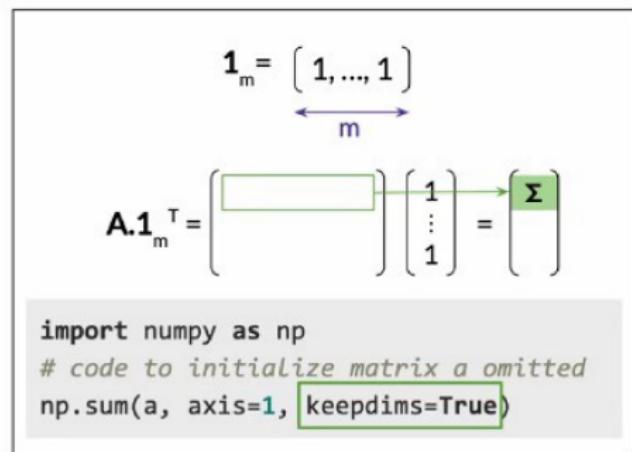
Backpropagation

$$\frac{\partial J_{batch}}{\partial \mathbf{W}_1} = \frac{1}{m} \text{ReLU} \left(\mathbf{W}_2^\top (\hat{\mathbf{Y}} - \mathbf{Y}) \right) \mathbf{X}^\top$$

$$\frac{\partial J_{batch}}{\partial \mathbf{W}_2} = \frac{1}{m} (\hat{\mathbf{Y}} - \mathbf{Y}) \mathbf{H}^\top$$

$$\frac{\partial J_{batch}}{\partial \mathbf{b}_1} = \frac{1}{m} \text{ReLU} \left(\mathbf{W}_2^\top (\hat{\mathbf{Y}} - \mathbf{Y}) \right) \mathbf{1}_m^\top$$

$$\frac{\partial J_{batch}}{\partial \mathbf{b}_2} = \frac{1}{m} (\hat{\mathbf{Y}} - \mathbf{Y}) \mathbf{1}_m^\top$$



Gradient descent

Hyperparameter: learning rate α

$$\mathbf{W}_1 := \mathbf{W}_1 - \alpha \frac{\partial J_{batch}}{\partial \mathbf{W}_1}$$

$$\mathbf{W}_2 := \mathbf{W}_2 - \alpha \frac{\partial J_{batch}}{\partial \mathbf{W}_2}$$

$$\mathbf{b}_1 := \mathbf{b}_1 - \alpha \frac{\partial J_{batch}}{\partial \mathbf{b}_1}$$

$$\mathbf{b}_2 := \mathbf{b}_2 - \alpha \frac{\partial J_{batch}}{\partial \mathbf{b}_2}$$

Gradient descent

Hyperparameter: learning rate α

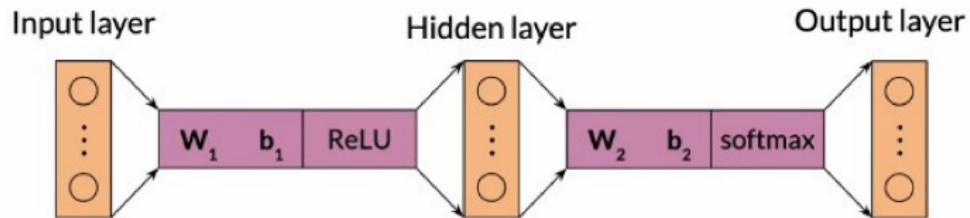
$$\mathbf{W}_1 := \mathbf{W}_1 - \alpha \frac{\partial J_{batch}}{\partial \mathbf{W}_1}$$

$$\mathbf{W}_2 := \mathbf{W}_2 - \alpha \frac{\partial J_{batch}}{\partial \mathbf{W}_2}$$

$$\mathbf{b}_1 := \mathbf{b}_1 - \alpha \frac{\partial J_{batch}}{\partial \mathbf{b}_1}$$

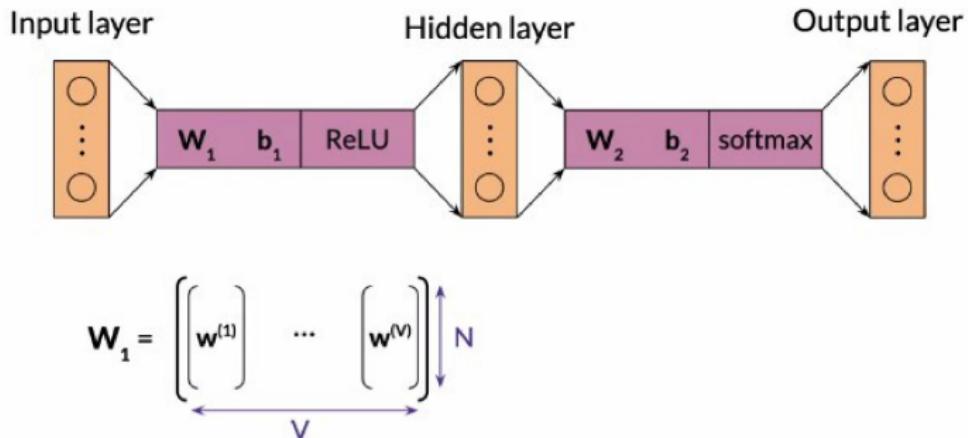
$$\mathbf{b}_2 := \mathbf{b}_2 - \alpha \frac{\partial J_{batch}}{\partial \mathbf{b}_2}$$

Extracting word embedding vectors: option 1



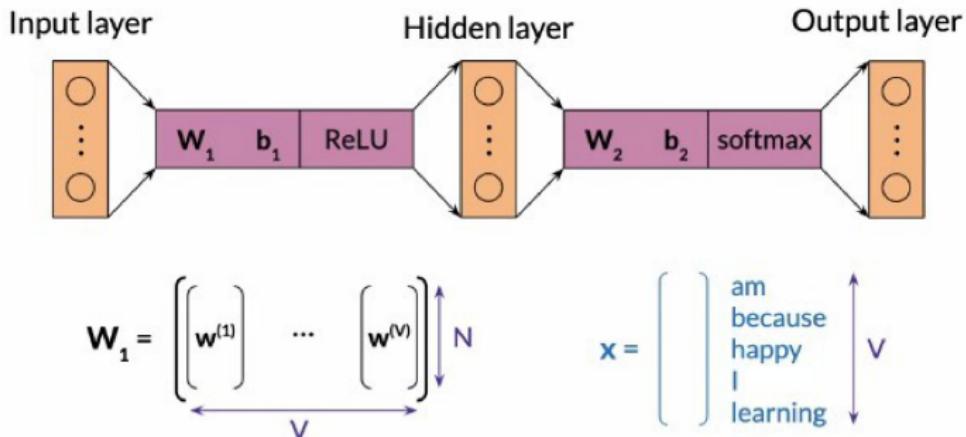
Wordembeddings

Extracting word embedding vectors: option 1



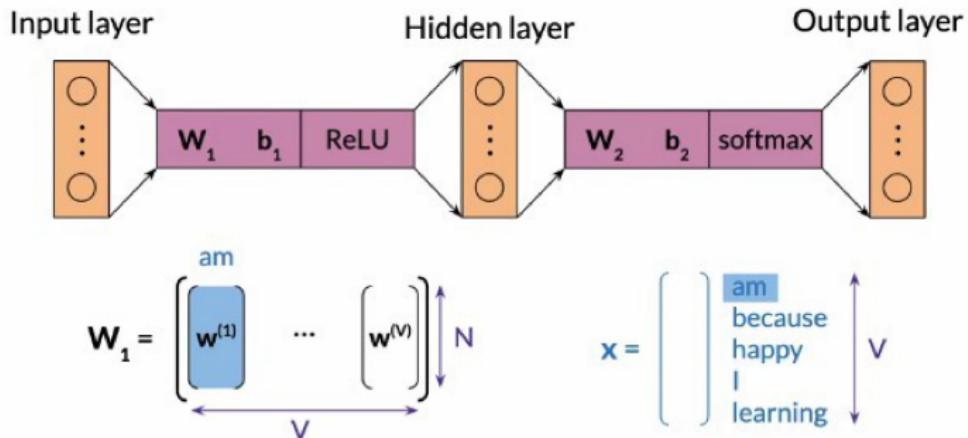
Wordembeddings

Extracting word embedding vectors: option 1

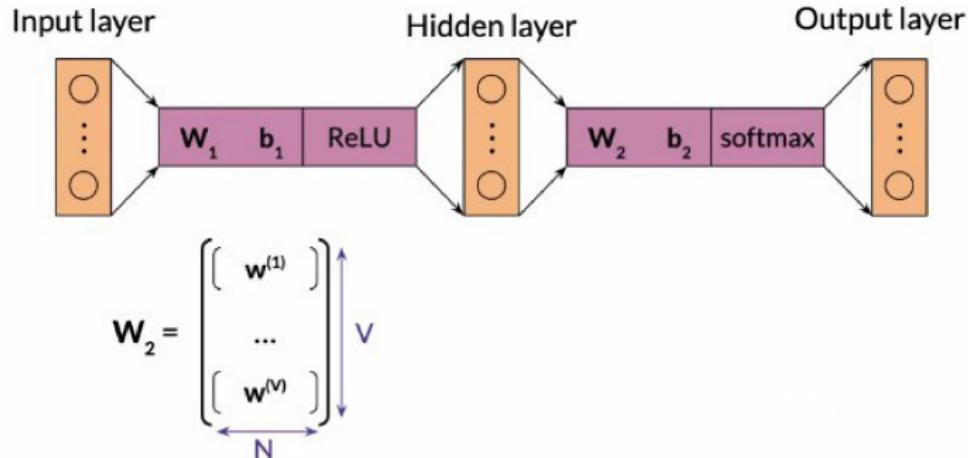


Wordembeddings

Extracting word embedding vectors: option 1

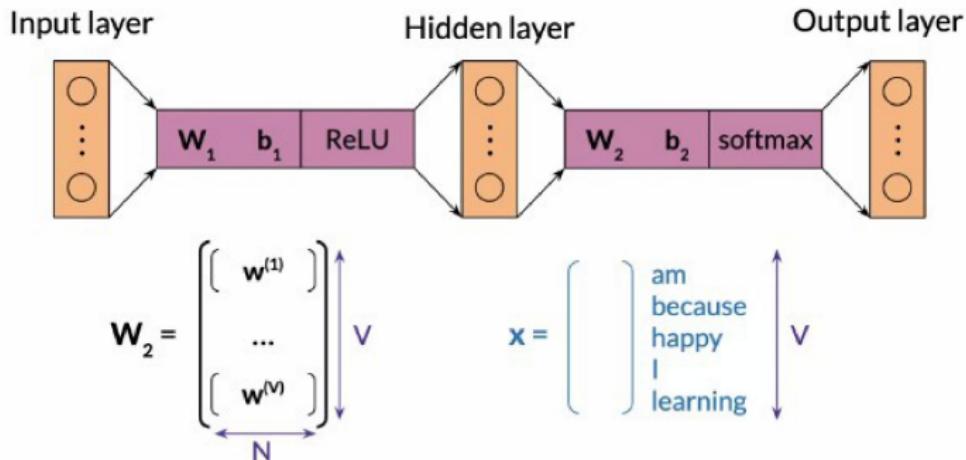


Extracting word embedding vectors: option 2



Wordembeddings

Extracting word embedding vectors: option 2



Extracting word embedding vectors: option 3

$$\mathbf{W}_1 = \begin{pmatrix} \left[\mathbf{w}_1^{(1)} \right] & \dots & \left[\mathbf{w}_1^{(V)} \right] \end{pmatrix}$$
$$\mathbf{W}_2 = \begin{pmatrix} \left[\mathbf{w}_2^{(1)} \right] \\ \dots \\ \left[\mathbf{w}_2^{(V)} \right] \end{pmatrix}$$

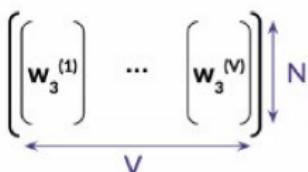
Wordembeddings

Extracting word embedding vectors: option 3

$$\mathbf{W}_1 = \begin{pmatrix} \left[\mathbf{w}_1^{(1)} \right] & \dots & \left[\mathbf{w}_1^{(V)} \right] \end{pmatrix}$$

$$\mathbf{W}_2 = \begin{pmatrix} \left[\mathbf{w}_2^{(1)} \right] \\ \dots \\ \left[\mathbf{w}_2^{(V)} \right] \end{pmatrix}$$

$$\mathbf{W}_3 = 0.5 (\mathbf{W}_1 + \mathbf{W}_2^T) = \begin{pmatrix} \left[\mathbf{w}_3^{(1)} \right] & \dots & \left[\mathbf{w}_3^{(V)} \right] \end{pmatrix}$$



Wordembeddings

Extracting word embedding vectors: option 3

$$\mathbf{W}_1 = \begin{pmatrix} \mathbf{w}_1^{(1)} & \cdots & \mathbf{w}_1^{(V)} \end{pmatrix}$$

$$\mathbf{W}_2 = \begin{pmatrix} \mathbf{w}_2^{(1)} \\ \cdots \\ \mathbf{w}_2^{(V)} \end{pmatrix}$$

$$\mathbf{W}_3 = 0.5 (\mathbf{W}_1 + \mathbf{W}_2^T) =$$

$$\begin{pmatrix} \mathbf{w}_3^{(1)} & \cdots & \mathbf{w}_3^{(V)} \end{pmatrix}$$

V

$$\mathbf{x} = \begin{pmatrix} \text{am} \\ \text{because} \\ \text{happy} \\ \text{I} \\ \text{learning} \end{pmatrix}$$

V

Intrinsic evaluation

Intrinsic evaluation

Test relationships between words

Intrinsic evaluation

Test relationships between words

- Analogies

Intrinsic evaluation

Test relationships between words

- Analogies

Semantic analogies

Intrinsic evaluation

Test relationships between words

- Analogies

Semantic analogies

“France” is to “Paris” as “Italy” is to <?>

Syntactic analogies

Intrinsic evaluation

Test relationships between words

- Analogies

Semantic analogies

“France” is to “Paris” as “Italy” is to <?>

Syntactic analogies

“seen” is to “saw” as “been” is to <?>

Intrinsic evaluation

Test relationships between words

- Analogies

Semantic analogies

“France” is to “Paris” as “Italy” is to <?>

Syntactic analogies

“seen” is to “saw” as “been” is to <?>

⚡ Ambiguity

Intrinsic evaluation

Test relationships between words

- Analogies

Semantic analogies

“France” is to “Paris” as “Italy” is to <?>

Syntactic analogies

“seen” is to “saw” as “been” is to <?>

⚡ Ambiguity

“wolf” is to “pack” as “bee” is to <?> → swarm? colony?

Wordembeddings

Intrinsic evaluation

Test relationships between words

- Analogies

Relationship	Example 1	Example 2	Example 3
France - Paris big - bigger	Italy: Rome small: larger	Japan: Tokyo cold: colder	Florida: Tallahassee quick: quicker
Miami - Florida	Baltimore: Maryland	Dallas: Texas	Kona: Hawaii
Einstein - scientist	Messi: midfielder	Mozart: violinist	Picasso: painter
Sarkozy - France copper - Cu	Berlusconi: Italy zinc: Zn	Merkel: Germany gold: Au	Koizumi: Japan uranium: plutonium
Berlusconi - Silvio	Sarkozy: Nicolas	Putin: Medvedev	Obama: Barack
Microsoft - Windows	Google: Android	IBM: Linux	Apple: iPhone
Microsoft - Ballmer	Google: Yahoo	IBM: McNealy	Apple: Jobs
Japan - sushi	Germany: bratwurst	France: tapas	USA: pizza

Wordembeddings

Intrinsic evaluation

Test relationships between words

- Analogies

Relationship	Example 1	Example 2	Example 3
France - Paris big - bigger	Italy: Rome small: larger	Japan: Tokyo cold: colder	Florida: Tallahassee quick: quicker
Miami - Florida	Baltimore: Maryland	Dallas: Texas	Kona: Hawaii
Einstein - scientist	Messi: midfielder	Mozart: violinist	Picasso: painter
Sarkozy - France	Berlusconi: Italy	Merkel: Germany	Koizumi: Japan
copper - Cu	zinc: Zn	gold: Au	uranium: plutonium
Berlusconi - Silvio	Sarkozy: Nicolas	Putin: Medvedev	Obama: Barack
Microsoft - Windows	Google: Android	IBM: Linux	Apple: iPhone
Microsoft - Ballmer	Google: Yahoo	IBM: McNealy	Apple: Jobs
Japan - sushi	Germany: bratwurst	France: tapas	USA: pizza

Intrinsic evaluation

Test relationships between words

- Analogies
- Clustering

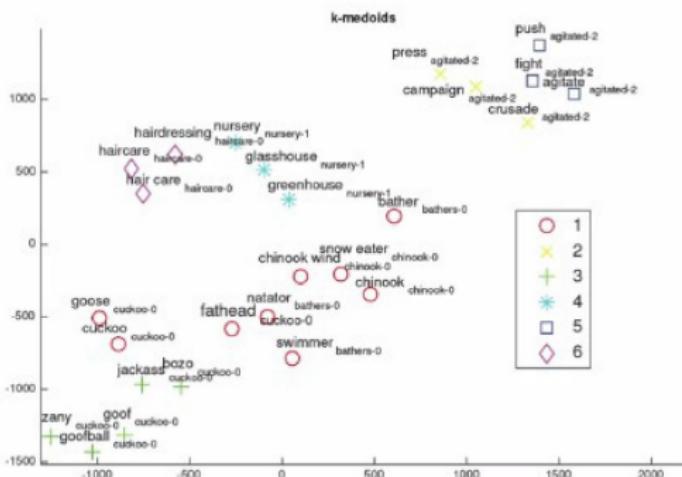
Wordembeddings

Intrinsic evaluation

Test relationships between words

- Analogies
 - Clustering

Source: Michael Zhai, Johnny Tan, and Jinho D. Choi. 2016. Intrinsic and extrinsic evaluations of word embeddings



Intrinsic evaluation

Test relationships between words

- Analogies
- Clustering
- Visualization



Word by Word Design

Number of times they *occur together within a certain distance k*

I like simple data

I prefer simple raw data

k=2

	simple	raw	like	I
data	2	1	1	0

Word by Word Design

Number of times they *occur together within a certain distance k*

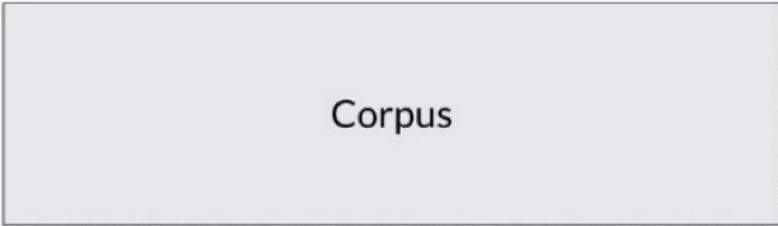
I like simple data
I prefer simple raw data

$k=2$

	simple	raw	like	I
data	2	1	1	0
n				

Word by Document Design

Number of times a word *occurs within a certain category*



Corpus

Word by Document Design

Number of times a word *occurs within a certain category*

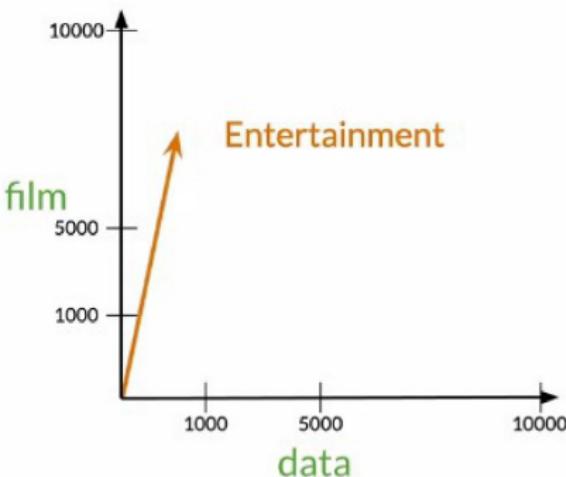


Word by Document Design

Number of times a word *occurs within a certain category*

	Entertainment	Economy	Machine Learning
Entertainment	500	6620	9320
Economy	7000	4000	1000

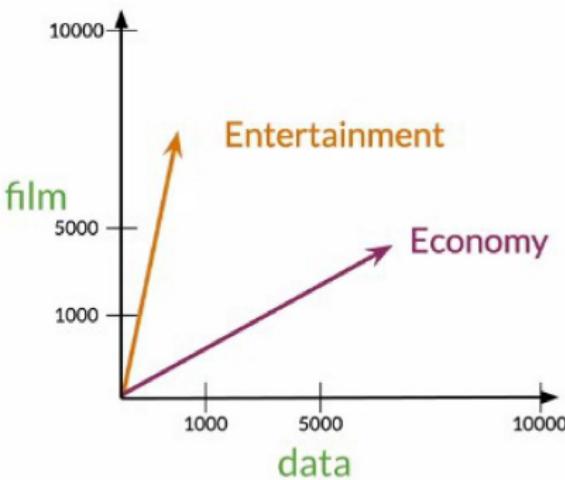
Vector Space



	Entertainment	Economy	ML
data	500	6620	9320
film	7000	4000	1000

vector spaces

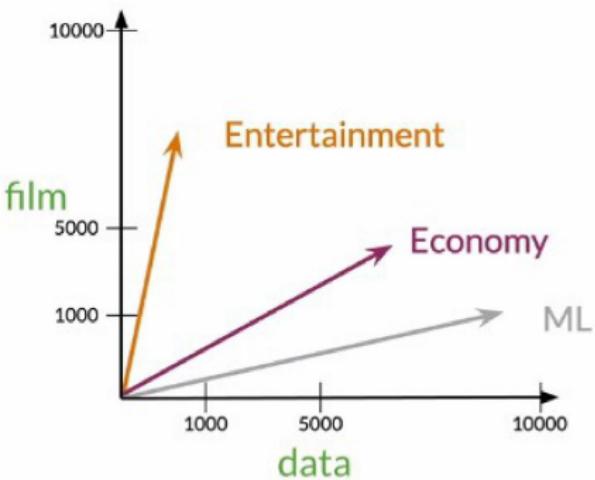
Vector Space



	Entertainment	Economy	ML
data	500	6620	9320
film	7000	4000	1000

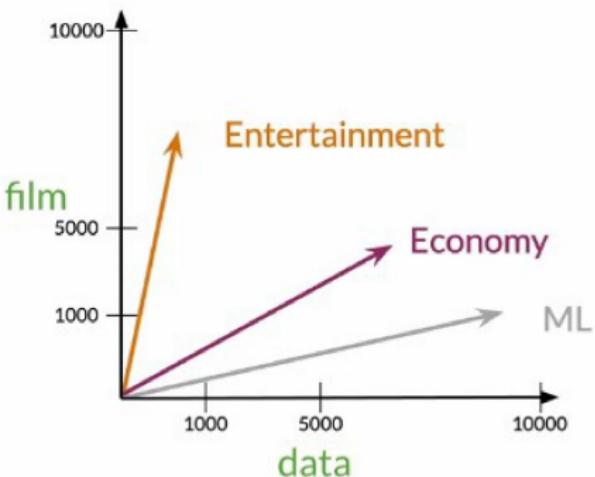
vector spaces

Vector Space



	Entertainment	Economy	ML
data	500	6620	9320
film	7000	4000	1000

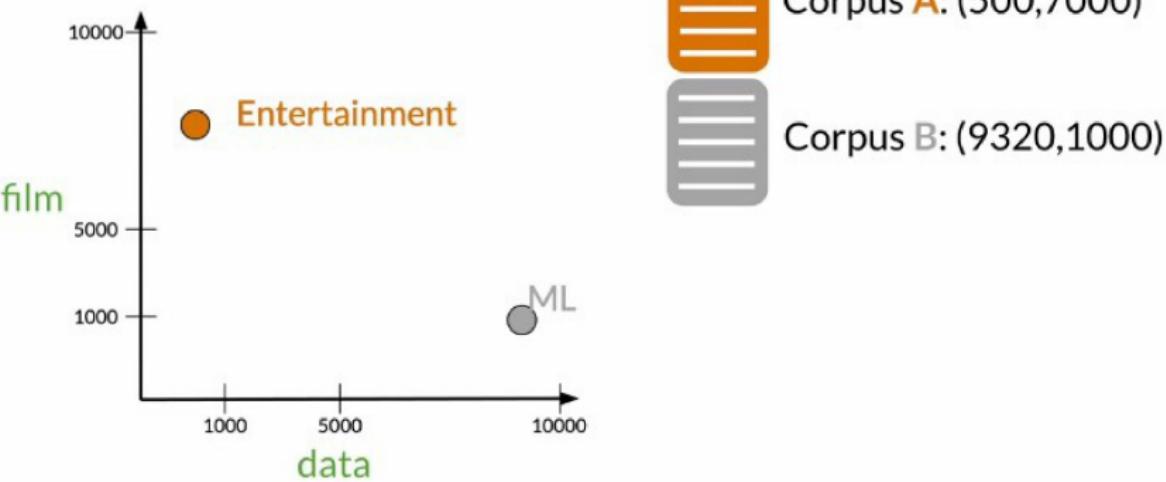
Vector Space



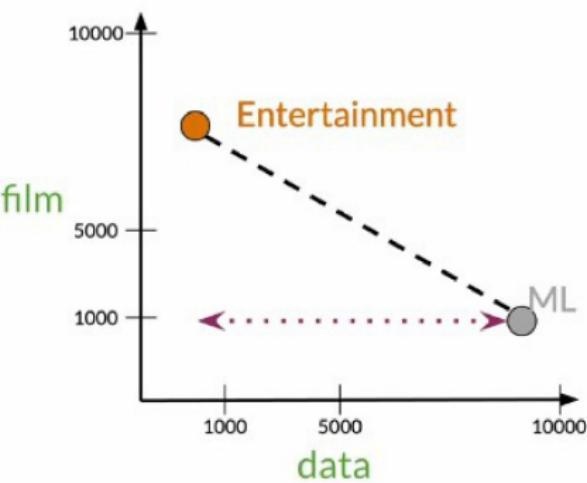
	Entertainment	Economy	ML
data	500	6620	9320
film	7000	4000	1000

Measures of “similarity:”
Angle
Distance

Euclidean distance



Euclidean distance



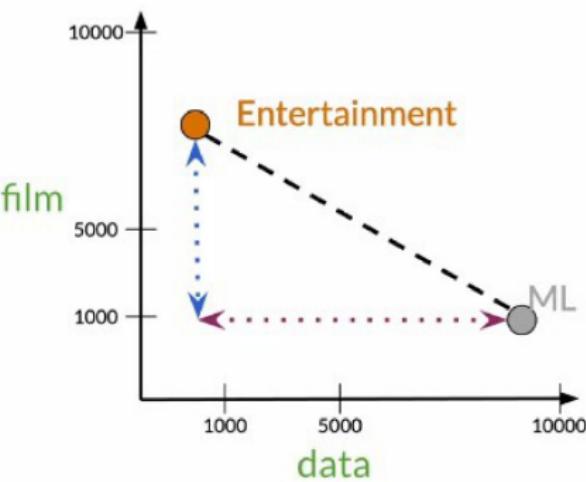
Corpus A: (500,7000)



Corpus B: (9320,1000)

$$d(B, A) = \sqrt{(B_1 - A_1)^2 + (B_2 - A_2)^2}$$

Euclidean distance



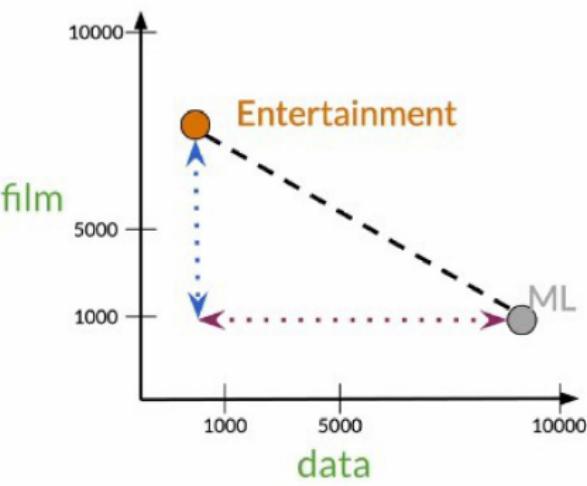
Corpus A: (500,7000)



Corpus B: (9320,1000)

$$d(B, A) = \sqrt{(B_1 - A_1)^2 + (B_2 - A_2)^2}$$

Euclidean distance



Corpus A: (500,7000)



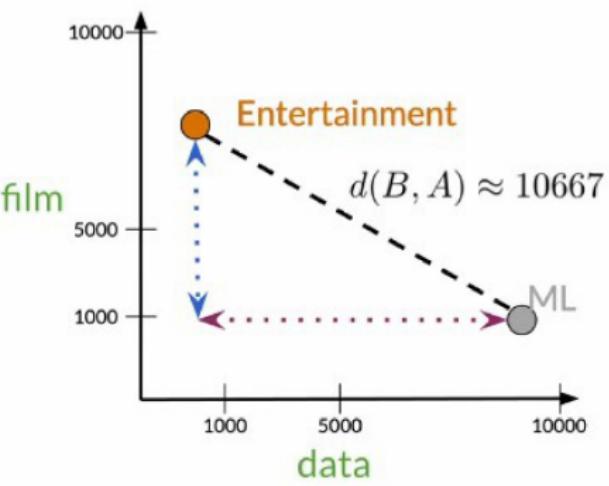
Corpus B: (9320,1000)

$$d(B, A) = \sqrt{(B_1 - A_1)^2 + (B_2 - A_2)^2}$$

$$c^2 = a^2 + b^2$$

$$d(B, A) = \sqrt{(-8820)^2 + (6000)^2}$$

Euclidean distance



Corpus A: (500,7000)



Corpus B: (9320,1000)

$$d(B, A) = \sqrt{(B_1 - A_1)^2 + (B_2 - A_2)^2}$$

$$c^2 = a^2 + b^2$$

$$d(B, A) = \sqrt{(-8820)^2 + (6000)^2}$$

Euclidean distance for n-dimensional vectors

$$\vec{v}$$

	data	boba	ice-cream
AI	6	0	1
drinks	0	4	6
food	0	6	8

Euclidean distance for n-dimensional vectors

	data	\vec{w}	\vec{v}
AI	6	0	1
drinks	0	4	6
food	0	6	8

Euclidean distance for n-dimensional vectors

	\vec{w}	\vec{v}
data	boba	ice-cream
AI	6	1
drinks	0	6
food	0	8

$$\begin{aligned} &= \sqrt{(1 - 0)^2 + (6 - 4)^2 + (8 - 6)^2} \\ &= \sqrt{1 + 4 + 4} = \sqrt{9} = 3 \end{aligned}$$

Euclidean distance for n-dimensional vectors

	data	\vec{w} boba	\vec{v} ice-cream
AI	6	0	1
drinks	0	4	6
food	0	6	8

$$\begin{aligned}
 &= \sqrt{(1 - 0)^2 + (6 - 4)^2 + (8 - 6)^2} \\
 &= \sqrt{1 + 4 + 4} = \sqrt{9} = 3
 \end{aligned}$$

$$d(\vec{v}, \vec{w}) = \sqrt{\sum_{i=1}^n (v_i - w_i)^2}$$

Euclidean distance for n-dimensional vectors

		\vec{w}	\vec{v}
	data	boba	ice-cream
AI	6	0	1
drinks	0	4	6
food	0	6	8

$$\begin{aligned}
 &= \sqrt{(1 - 0)^2 + (6 - 4)^2 + (8 - 6)^2} \\
 &= \sqrt{1 + 4 + 4} = \sqrt{9} = 3
 \end{aligned}$$

$$d(\vec{v}, \vec{w}) = \sqrt{\sum_{i=1}^n (v_i - w_i)^2} \longrightarrow \text{Norm of } (\vec{v} - \vec{w})$$

Euclidean distance in Python

```
# Create numpy vectors v and w
v = np.array([1, 6, 8])
w = np.array([0, 4, 6])

# Calculate the Euclidean distance d
d = np.linalg.norm(v-w)
# Print the result
print("The Euclidean distance between v and w is: ", d)
```

The Euclidean distance between v and w is: 3

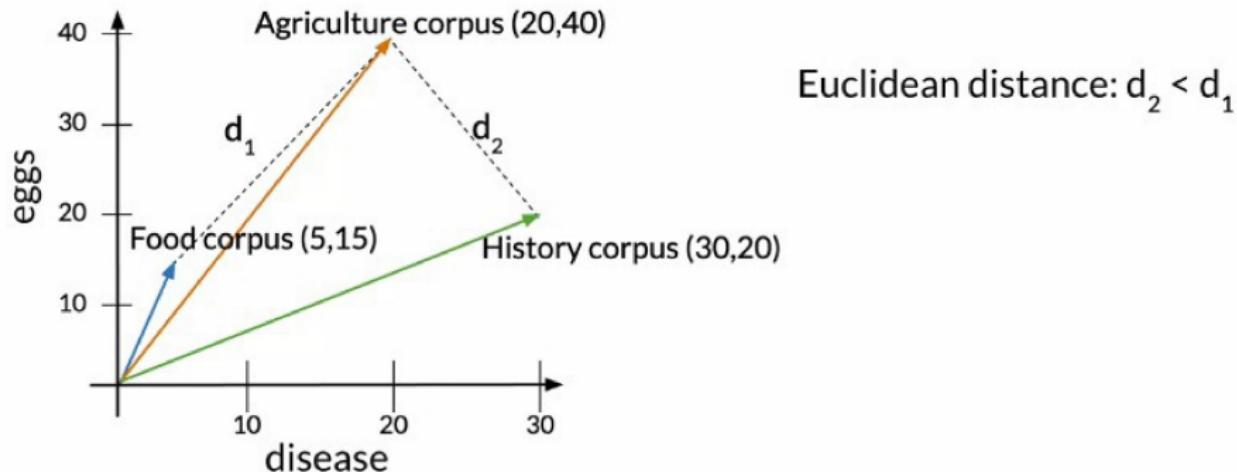
Summary

- Straight line between points
- Norm of the difference between vectors

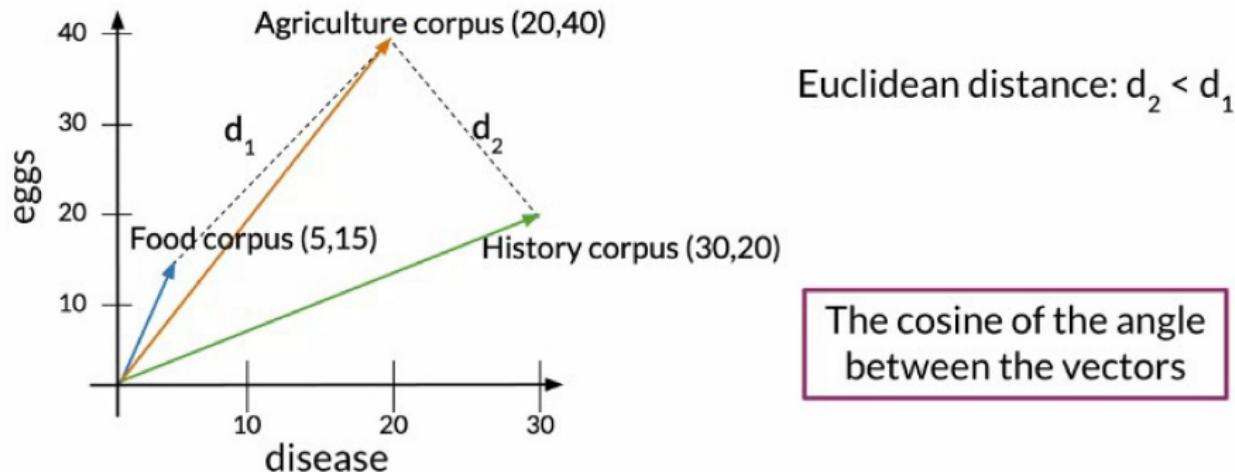
Outline

- Problems with Euclidean Distance
- Cosine similarity

Euclidean distance vs Cosine similarity



Euclidean distance vs Cosine similarity



Summary

- Cosine similarity when corpora are different sizes

Cosine Similarity

Outline

- How to get the cosine of the angle between two vectors
- Relation of this metric to similarity

Previous definitions

Vector norm

$$\|\vec{v}\| = \sqrt{\sum_{i=1}^n v_i^2}$$

Previous definitions

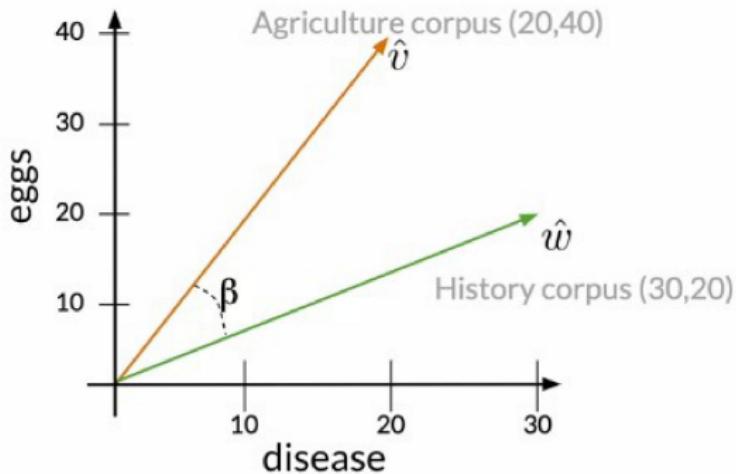
Vector norm

$$\|\vec{v}\| = \sqrt{\sum_{i=1}^n v_i^2}$$

Dot product

$$\vec{v} \cdot \vec{w} = \sum_{i=1}^n v_i \cdot w_i$$

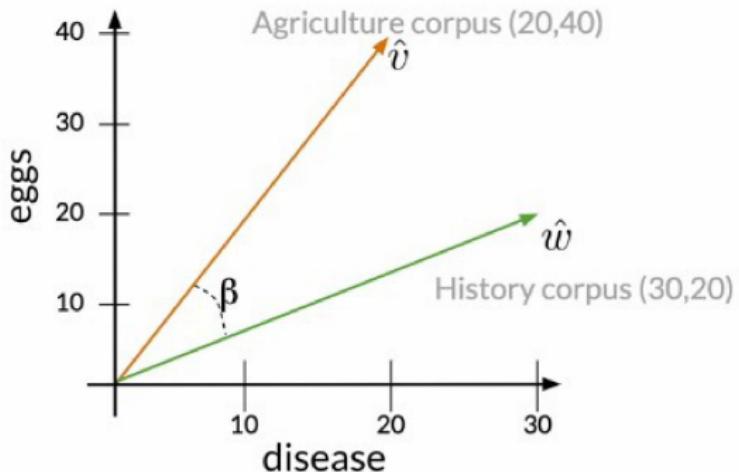
Cosine Similarity



$$\hat{v} \cdot \hat{w} = \|\hat{v}\| \|\hat{w}\| \cos(\beta)$$

$$\cos(\beta) = \frac{\hat{v} \cdot \hat{w}}{\|\hat{v}\| \|\hat{w}\|}$$

Cosine Similarity

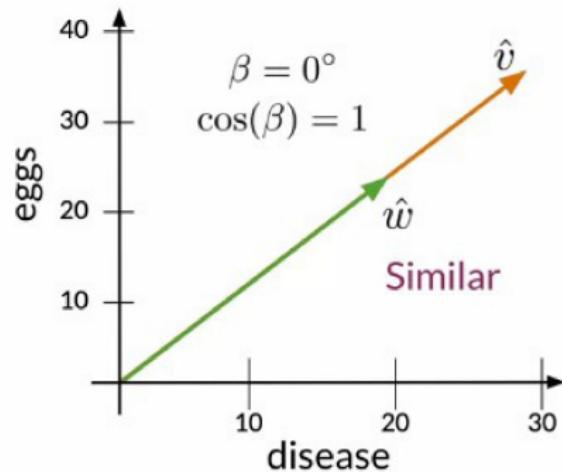
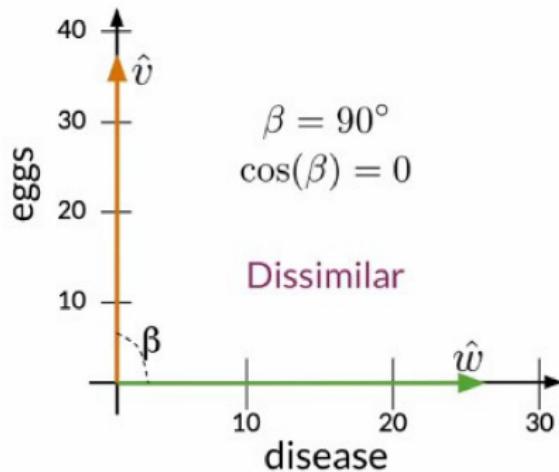


$$\hat{v} \cdot \hat{w} = \|\hat{v}\| \|\hat{w}\| \cos(\beta)$$

$$\cos(\beta) = \frac{\hat{v} \cdot \hat{w}}{\|\hat{v}\| \|\hat{w}\|}$$

$$\begin{aligned}
 &= \frac{(20 \times 30) + (40 \times 20)}{\sqrt{20^2 + 40^2} \times \sqrt{30^2 + 20^2}} \\
 &= 0.87
 \end{aligned}$$

Cosine Similarity



Summary

- Cosine \propto Similarity
- Cosine Similarity gives values between 0 and 1

Outline

- How to use vector representations

vector spaces

Manipulating word vectors



USA



Washington
DC



Russia



?

vector spaces

Manipulating word vectors



[Mikolov et al, 2013, Distributed Representations of Words and Phrases and their Compositionality]

vector spaces

Manipulating word vectors



[Mikolov et al, 2013, Distributed Representations of Words and Phrases and their Compositionality]

Summary

- Use known relationships to make predictions

Outline

- Some motivation for visualization

Outline

- Some motivation for visualization
- Principal Component Analysis

Visualization of word vectors

oil	0.20	...	0.10
gas	2.10	...	3.40
city	9.30	...	52.1
town	6.20	...	34.3

Visualization of word vectors

$d > 2$

oil	0.20	...	0.10
gas	2.10	...	3.40
city	9.30	...	52.1
town	6.20	...	34.3



oil & gas



town & city

Visualization of word vectors

$d > 2$

oil	0.20	...	0.10
gas	2.10	...	3.40
city	9.30	...	52.1
town	6.20	...	34.3



oil & gas



town & city

How can you visualize if your representation captures these relationships?

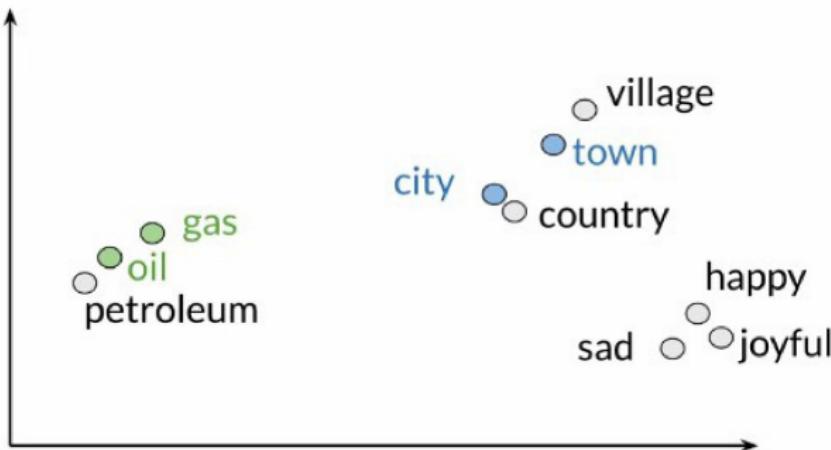
Visualization of word vectors

$d > 2$

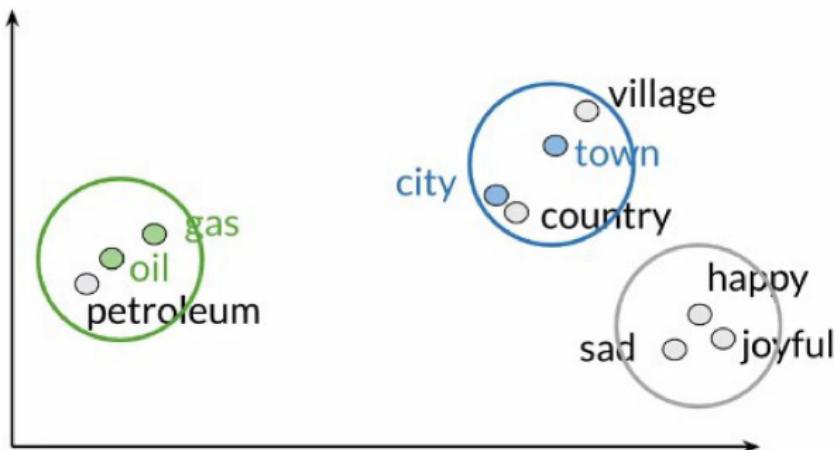
oil	0.20	...	0.10
gas	2.10	...	3.40
city	9.30	...	52.1
town	6.20	...	34.3

vector spaces

Visualization of word vectors



Visualization of word vectors



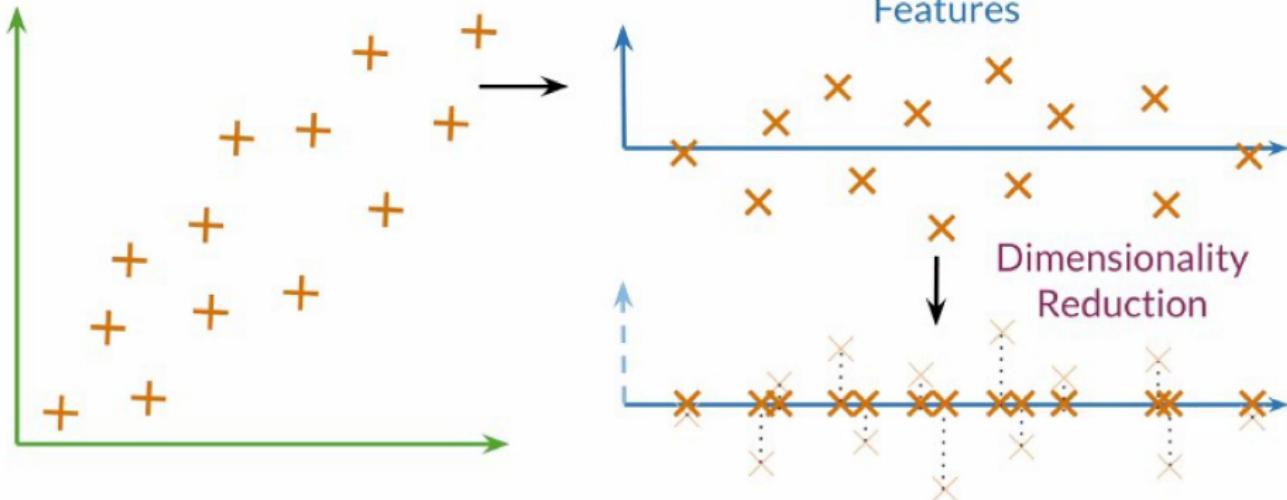
Summary

- Original Space → Uncorrelated features → Dimension reduction
- Visualization to see words relationships in the vector space

Outline

- How to get uncorrelated features
- How to reduce dimensions while retaining as much information as possible

Principal Component Analysis



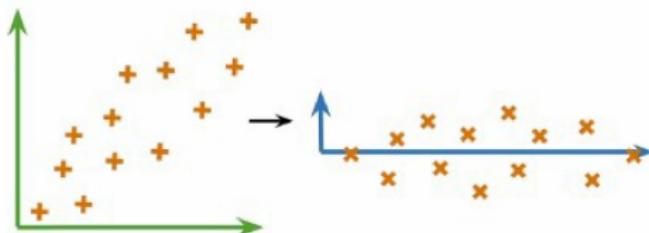
PCA algorithm

Eigenvector: Uncorrelated features for your data

Eigenvalue: the amount of information retained by each feature

vector spaces

PCA algorithm

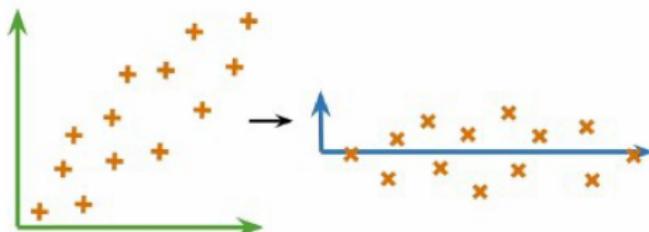


Mean Normalize Data $x_i = \frac{x_i - \mu_{x_i}}{\sigma_{x_i}}$

Get Covariance Matrix Σ

Perform SVD $SVD(\Sigma)$

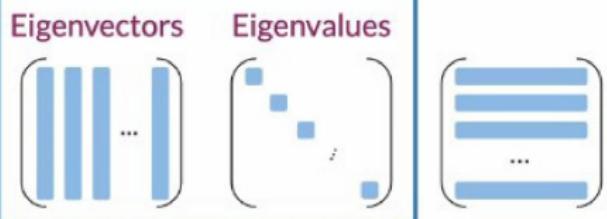
PCA algorithm



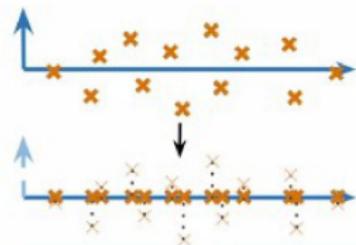
Mean Normalize Data $x_i = \frac{x_i - \mu_{x_i}}{\sigma_{x_i}}$

Get Covariance Matrix Σ

Perform SVD $SVD(\Sigma)$

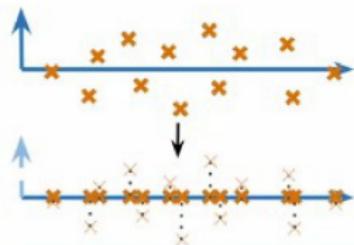


PCA algorithm



Eigenvectors	Eigenvalues
U	S

PCA algorithm



Eigenvectors Eigenvalues

$$\begin{pmatrix} U \\ \vdots \end{pmatrix} \quad \begin{pmatrix} S \\ \vdots \end{pmatrix}$$

Dot Product to
Project Data

$$X' = XU[:, 0 : 2]$$

Percentage of
Retained Variance

$$\frac{\sum_{i=0}^1 S_{ii}}{\sum_{j=0}^d S_{jj}}$$

Word2Vec

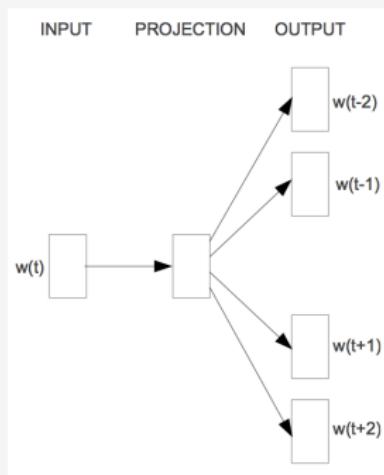
- ▶ Word2Vec es un paquete de software que implementa dos arquitecturas de redes neuronales para entrenar wordembeddings: Continuous Bag of Words (CBOW) y Skip-gram.
- ▶ Implementa dos modelos de optimización: Muestreo Negativo y Softmax Jerárquico.
- ▶ Estos modelos son redes neuronales superficiales que están capacitadas para predecir los contextos de las palabras.
- ▶ Un tutorial muy completo sobre los algoritmos de word2vec:
<https://arxiv.org/pdf/1411.2738.pdf>.

Modelo Skip-gram

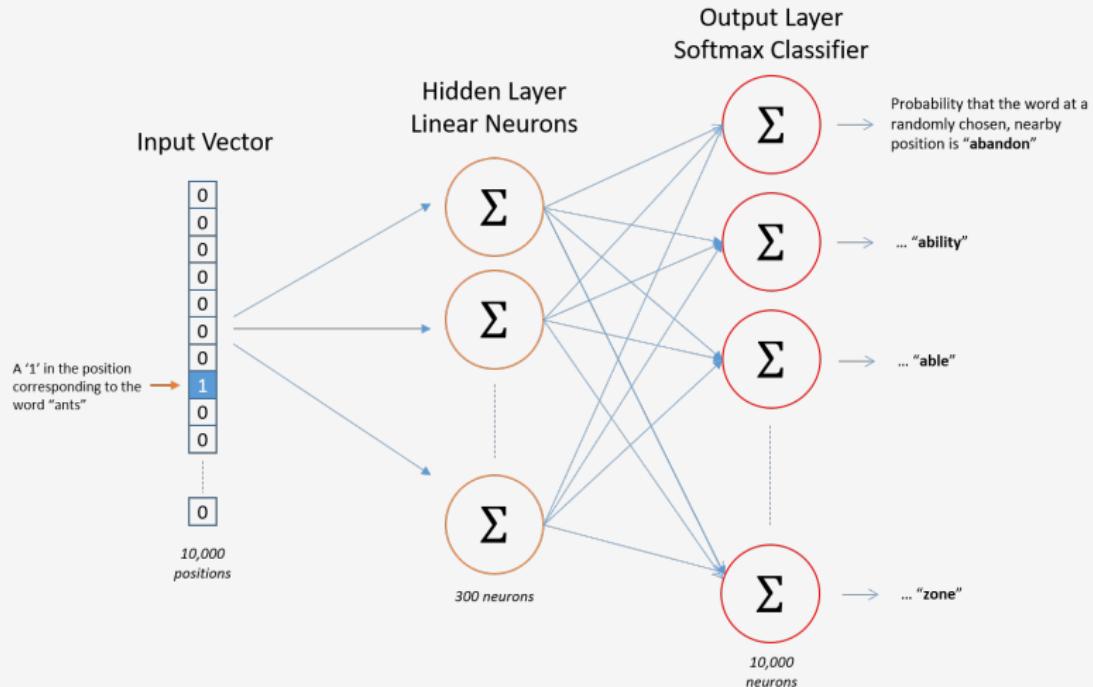
- ▶ Una red neuronal con una capa de “ proyección ” u “ oculta ” está entrenada para predecir las palabras que rodean una palabra central, dentro de una ventana de tamaño k que se desplaza a lo largo del corpus de entrada.
- ▶ El centro y las palabras k circundantes corresponden a las capas de entrada y salida de la red.
- ▶ Las palabras se representan inicialmente mediante vectores 1-hot: vectores del tamaño del vocabulario ($|V|$) con valores cero en todas las entradas excepto el índice de palabras correspondiente que recibe un valor de 1.

Modelo Skip-gram

- ▶ La capa de salida combina los vectores k 1-hot de las palabras circundantes.
- ▶ La capa oculta tiene una dimensionalidad d , que determina el tamaño de las incrustaciones (normalmente $d \ll |V|$).



Modelo Skip-gram



Parametrización del modelo Skip-gram

- ▶ Se nos da un corpus de entrada formado por una secuencia de palabras $w_1, w_2, w_3, \dots, w_T$ y un tamaño de ventana k .
- ▶ Denotamos las palabras objetivo o (centrales) con la letra w y las palabras de contexto circundantes con la letra c .
- ▶ La ventana de contexto $c_{1:k}$ de la palabra w_t corresponde a las palabras $w_{tk/2}, \dots, w_{t-1}, w_{t+1}, \dots, w_{t+k/2}$ (asumiendo que k es un número par).

Parametrización del modelo Skip-gram

- ▶ El objetivo del modelo Skip-gram es maximizar la probabilidad logarítmica promedio de las palabras de contexto dadas las palabras objetivo:

$$\frac{1}{T} \sum_{t=1}^T \sum_{c \in c_{1:k}} \log P(c|w_t)$$

- ▶ La probabilidad condicional de una palabra de contexto c dada una palabra central w se modela con un softmax (C es el conjunto de todas las palabras de contexto, que suele ser el mismo que el vocabulario):

$$P(c|w) = \frac{e^{\vec{c} \cdot \vec{w}}}{\sum_{c' \in C} e^{\vec{c}' \cdot \vec{w}}}$$

- ▶ Parámetros del modelo θ : \vec{c} y \vec{w} (representaciones vectoriales de contextos y palabras objetivo).

Parametrización del modelo Skip-gram

- ▶ Sea D el conjunto de pares correctos palabra-contexto (es decir, pares de palabras que se observan en el Corpus).
- ▶ El objetivo de la optimización es maximizar la probabilidad logarítmica condicional de los contextos c (esto es equivalente a minimizar la pérdida de entropía cruzada):

$$\arg \max_{\vec{c}, \vec{w}} \sum_{(w,c) \in D} \log P(c|w) = \sum_{(w,c) \in D} (\log e^{\vec{c} \cdot \vec{w}} - \log \sum_{c' \in C} e^{\vec{c}' \cdot \vec{w}}) \quad (3)$$

- ▶ : Suposición: maximizar esta función resultará en buenos wordembeddings \vec{w} es decir, palabras similares tendrán vectores similares.
- ▶ El término $P(c|w)$ es computacionalmente costoso debido a la suma $\sum_{c' \in C} e^{\vec{c}' \cdot \vec{w}}$ sobre todos los contextos c' .
- ▶ Fix: reemplace el softmax con un softmax jerárquico (el vocabulario se representa con un árbol binario de Huffman).
- ▶ Los árboles de Huffman asignan códigos binarios cortos a palabras frecuentes, lo que reduce el número de unidades de salida a evaluar.

Skip-gram con muestreo Negativo

- ▶ El muestreo negativo (NS) se presenta como un modelo más eficiente para calcular los skip-gram embeddings.
- ▶ Sin embargo, optimiza una función objetivo diferente [Goldberg and Levy, 2014].
- ▶ NS maximiza la probabilidad de que un par palabra-contexto (w, c) provenga del conjunto de pares correctos palabra-contexto D usando una función sigmoidea:

$$P(D = 1|w, c_i) = \frac{1}{1 + e^{-\vec{w} \cdot \vec{c}_i}}$$

- ▶ Supuesto: las palabras de contexto c_i son independientes entre sí:

$$P(D = 1|w, c_{1:k}) = \prod_{i=1}^k P(D = 1|w, c_i) = \prod_{i=1}^k \frac{1}{1 + e^{-\vec{w} \cdot \vec{c}_i}}$$

- ▶ Esto conduce a la siguiente función objetivo (log-likelihood):

$$\arg \max_{\vec{c}, \vec{w}} \log P(D = 1|w, c_{1:k}) = \sum_{i=1}^k \log \frac{1}{1 + e^{-\vec{w} \cdot \vec{c}_i}} \quad (4)$$

Skip-gram con muestreo Negativo (2)

- ▶ Este objetivo tiene una solución trivial si establecemos \vec{w} , \vec{c} tal que $P(D = 1|w, c) = 1$ para cada par (w, c) desde D .
- ▶ Esto se logra configurando $\vec{w} = \vec{c}$ y $\vec{w} \cdot \vec{c} = K$ para todos \vec{w}, \vec{c} , donde K es un número grande.
- ▶ Necesitamos un mecanismo que evite que todos los vectores tengan el mismo valor, al no permitir algunas combinaciones (w, c) .
- ▶ Una forma de hacerlo es presentar el modelo con algunos (w, c) pares para los cuales $P(D = 1|w, c)$ debe ser bajo, es decir pares que no están en los datos.
- ▶ Esto se logra tomando muestras negativas de \tilde{D} .

Skip-gram con muestreo Negativo (3)

- ▶ Ejemplo de m palabras para cada par palabra-contexto $(w, c) \in D$.
- ▶ Agrega cada palabra muestreada w_i junto con el contexto original c como un ejemplo negativo a \tilde{D} .
- ▶ Función objetivo final:

$$\arg \max_{\vec{c}, \vec{w}} \sum_{(w, c) \in D} \log P(D = 1 | w, c_{1:k}) + \sum_{(w, c) \in \tilde{D}} \log P(D = 0 | w, c_{1:k}) \quad (5)$$

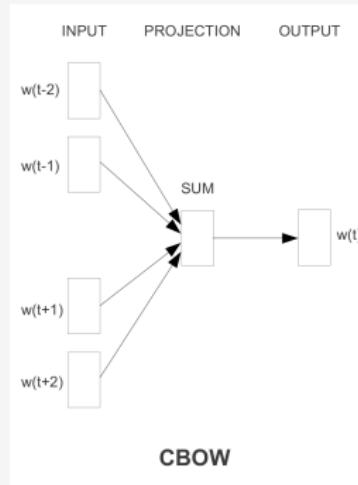
- ▶ Las palabras negativas se extraen de la versión suavizada de las frecuencias del corpus:

$$\frac{\#(w)^{0.75}}{\sum_{w'} \#(w')^{0.75}}$$

- ▶ Esto le da más peso relativo a las palabras menos frecuentes.

Continuous Bag of Words: CBOW

- ▶ Similar al modelo skip-gram , pero ahora la palabra central se predice a partir del contexto circundante.



- ▶ GloVe (de vectores globales) es otro método popular para entrenar incrustaciones de palabras [Pennington et al., 2014].
- ▶ Construye un contexto de palabras explícito matriz, y entrena los vectores de palabra y contexto \vec{w} y \vec{c} intentando satisfacer:

$$\vec{w} \cdot \vec{c} + b_{[w]} + b_{[c]} = \log \#(w, c) \quad \forall (w, c) \in D \quad (6)$$

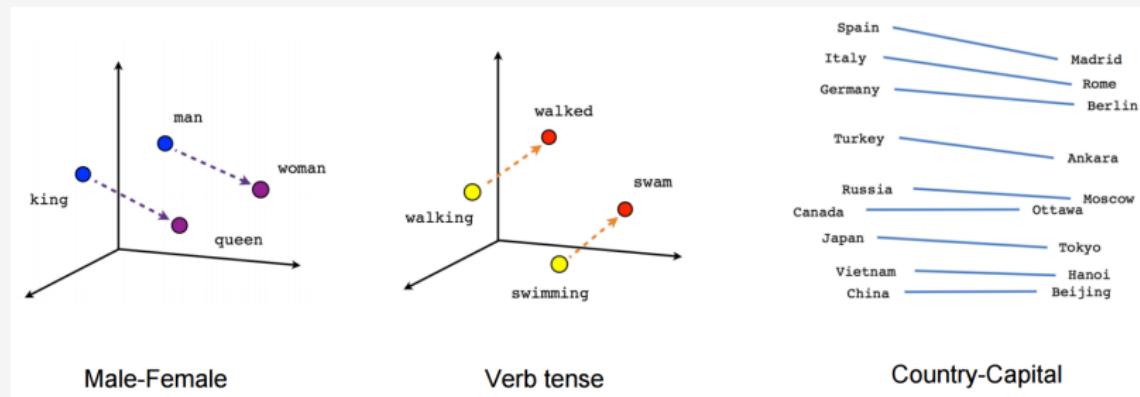
- ▶ donde $b_{[w]}$ y $b_{[c]}$ son sesgos entrenados específicos de la palabra y del contexto.

GloVe (2)

- ▶ En términos de factorización matricial, si arreglamos $b_{[w]} = \log \#(w)$ y $b_{[c]} = \log \#(c)$ obtendremos un objetivo que es muy similar a factorizar la matriz PMI de contexto de palabras, desplazada por $\log(|D|)$.
- ▶ En GloVe, los parámetros de sesgo se aprenden y no se fijan, lo que le da otro grado de libertad.
- ▶ El objetivo de optimización es la pérdida por mínimos cuadrados ponderados, asignando más peso a la correcta reconstrucción de los ítems frecuentes.
- ▶ Cuando se usa la misma palabra y vocabularios de contexto, el modelo sugiere representar cada palabra como la suma de sus correspondientes vectores de inserción de palabras y contextos.

Word Analogies

- ▶ Los wordembeddings pueden capturar ciertas relaciones semánticas, p. ej. hombre-mujer, tiempo verbal y relaciones país-capital entre palabras.
- ▶ Por ejemplo, la siguiente relación se encuentra para incrustaciones de palabras entrenado usando Word2Vec: $\vec{w}_{king} - \vec{w}_{man} + \vec{w}_{woman} \approx \vec{w}_{queen}$.



²Source: <https://www.tensorflow.org/tutorials/word2vec>

Evaluación

- ▶ Hay muchos conjuntos de datos con asociaciones anotadas humanas de pares de palabras o analogías que se pueden usar para evaluar algoritmos de wordembeddings.
- ▶ Estos enfoques se denominan *Enfoques de evaluación intrínseca*.
- ▶ La mayoría de ellos están implementados en: url
<https://github.com/kudkudak/word-embeddings-benchmarks>.
- ▶ Los wordembeddings también se pueden evaluar extrínsecamente usándolas en una tarea externa de NLP (por ejemplo, etiquetado de POS, análisis de sentimiento).

Gensim

Gensim is an open source Python library for natural language processing that implements many algorithms for training word embeddings.

- ▶ <https://radimrehurek.com/gensim/>
- ▶ <https://machinelearningmastery.com/develop-word-embeddings-python-gensim/>



References

-  Amir, S., Ling, W., Astudillo, R., Martins, B., Silva, M. J., and Trancoso, I. (2015). Inesc-id: A regression model for large scale twitter sentiment lexicon induction. In *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, pages 613–618, Denver, Colorado. Association for Computational Linguistics.
-  Baroni, M., Dinu, G., and Kruszewski, G. (2014). Don't count, predict! a systematic comparison of context-counting vs. context-predicting semantic vectors. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, pages 238–247. Association for Computational Linguistics.
-  Goldberg, Y. (2016). A primer on neural network models for natural language processing.