Procesamiento de Lenguaje Natural

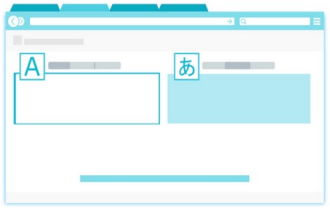
Word Embeddings – Vectores de palabras

Dra. Helena Gómez Adorno
helena.gomez@iimas.unam.mx

Correo del curso:
pln.cienciadedatos@gmail.com

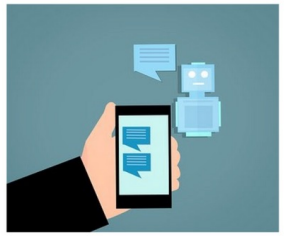
Contenido

- Las incrustaciones de palabras se utilizan en la mayoría de las aplicaciones de PLN. Siempre que se trate de texto, primero debe encontrar una forma de codificar las palabras como números. La incrustación de palabras es una técnica muy común que le permite hacerlo. Aquí hay algunas aplicaciones de incrustaciones de palabras:

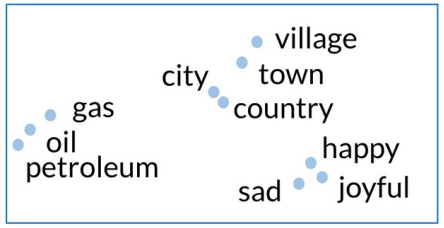


Machine translation

Information extraction



Question answering



Semantic analogies and similarity

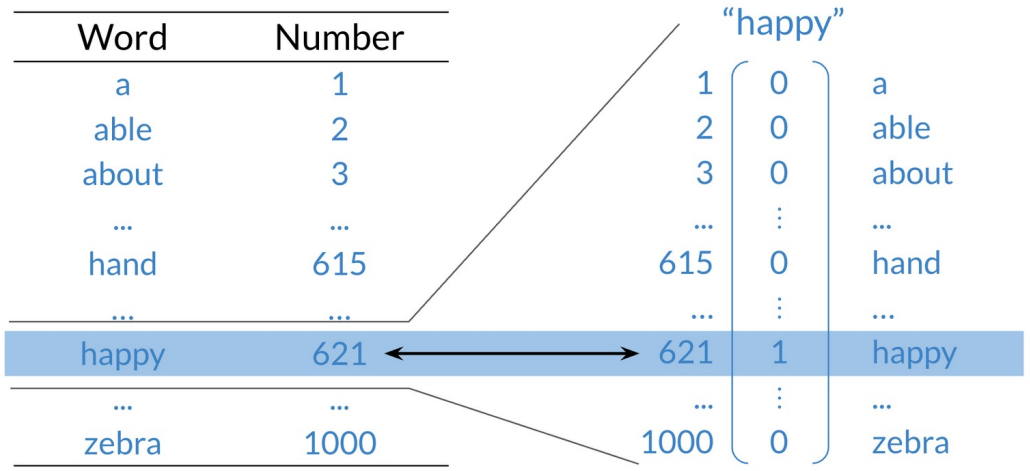
Sentiment analysis

Classification of customer feedback

Representaciones básicas de palabras

Las representaciones básicas de palabras se pueden dividir en:

- Enteros
- Vectores One-hot
- Incrustaciones de palabras(Word embeddings)



- A la izquierda, tienen un ejemplo en el que usas números enteros para representar una palabra. El problema es que no hay ninguna razón por la que una palabra corresponda a un número mayor que otra. Para solucionar este problema introducimos un vector *one-hot* (diagrama a la derecha). Para implementar un vector *one-hot*, tienen que inicializar un vector de ceros de dimensión *V* y luego poner un 1 en el índice correspondiente a la palabra que están representando.

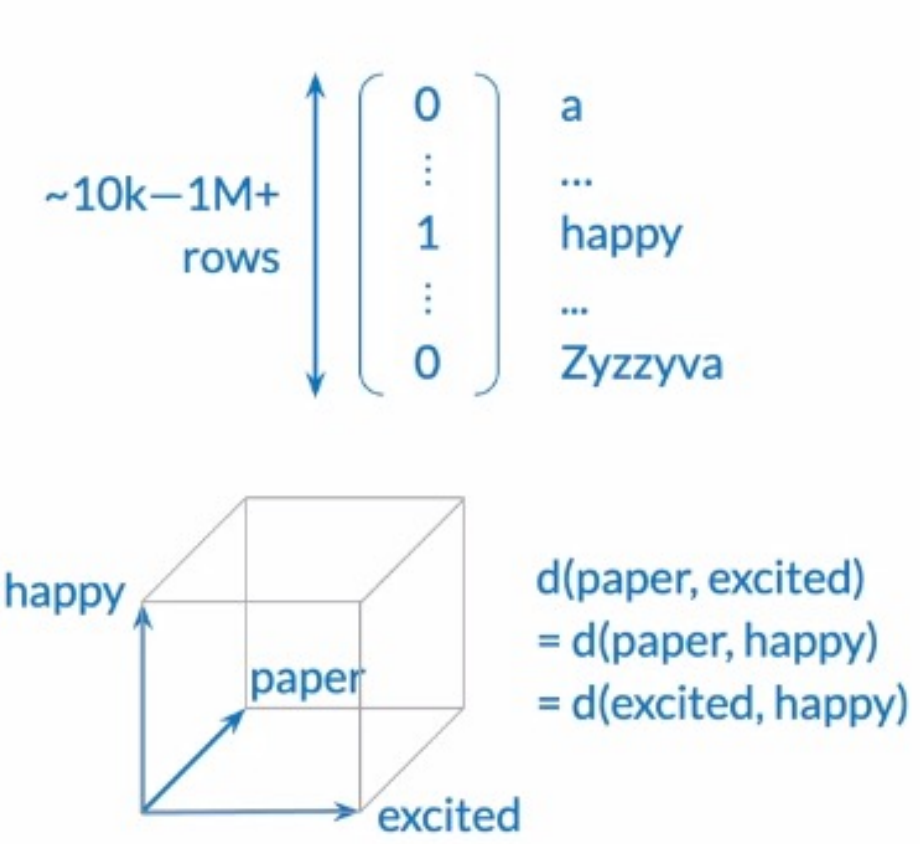
Vectores one-hot

Los pros de los vectores one-hot:

- Simples y
- No requieren ordenamiento implícito.

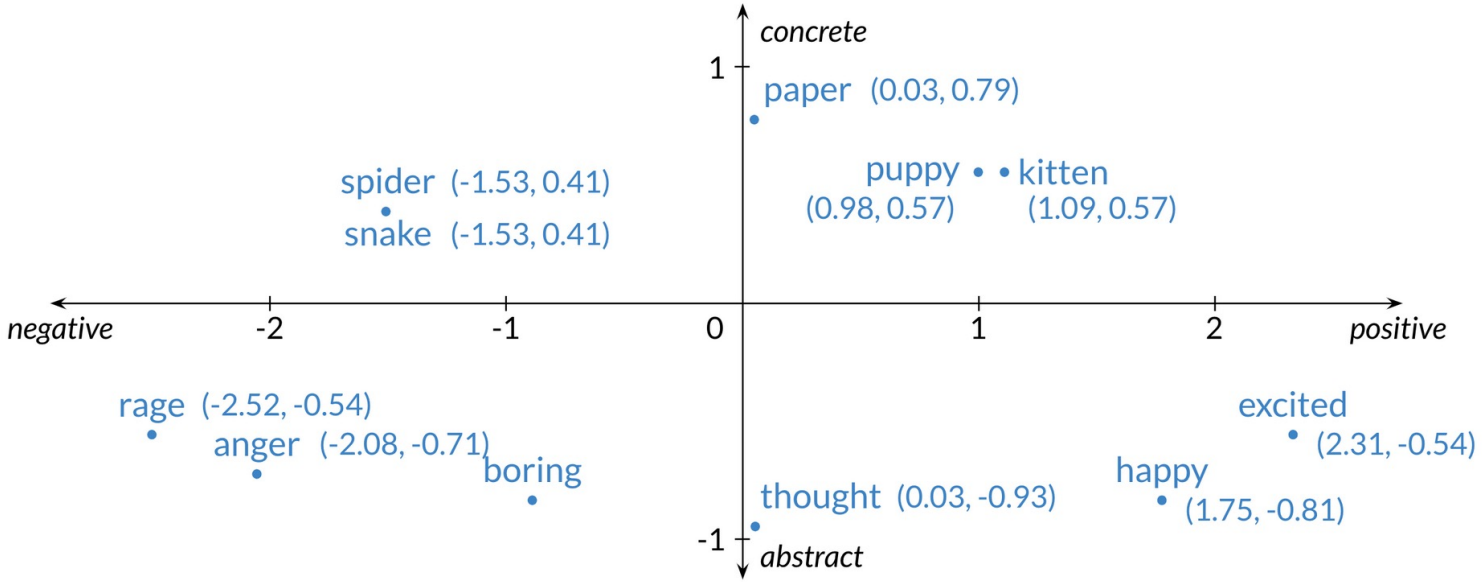
Los contras de los vectores one-hot:

- Enormes y
- Sin codificar significado.



Incrustaciones de palabras: Word Embeddings

- En el gráfico, pueden ver que al codificar una palabra en 2D, las palabras similares tienden a encontrarse una al lado de la otra. Quizás la primera coordenada representa si una palabra es positiva o negativa. La segunda coordenada te dice si la palabra es abstracta o concreta. Este es solo un ejemplo, en el mundo real encontrarán incrustaciones con cientos de dimensiones. Puedes pensar en cada coordenada como un número que te dice algo sobre la palabra.



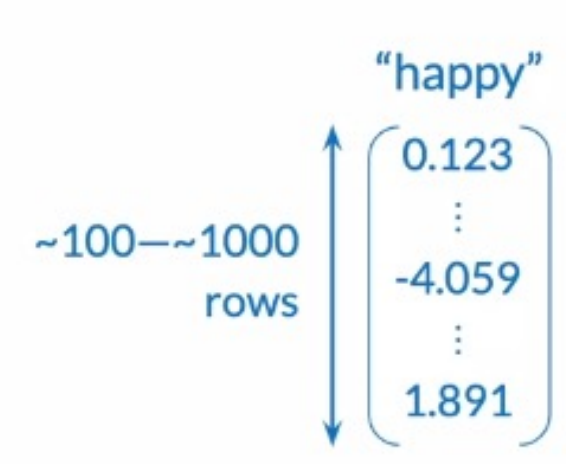
Incrustaciones de palabras: Word Embeddings

Los pros:

- Dimensiones bajas (menos de *V*)
- Permiten codificar el significado
  - Ejemplo: distancia semántica

  Bosque = árbol bosque <> serpiente
  - Ejemplo: analogías

  Paris:Francia :: Roma:?



Terminología

integers

word vectors

one-hot vectors

word embedding vectors

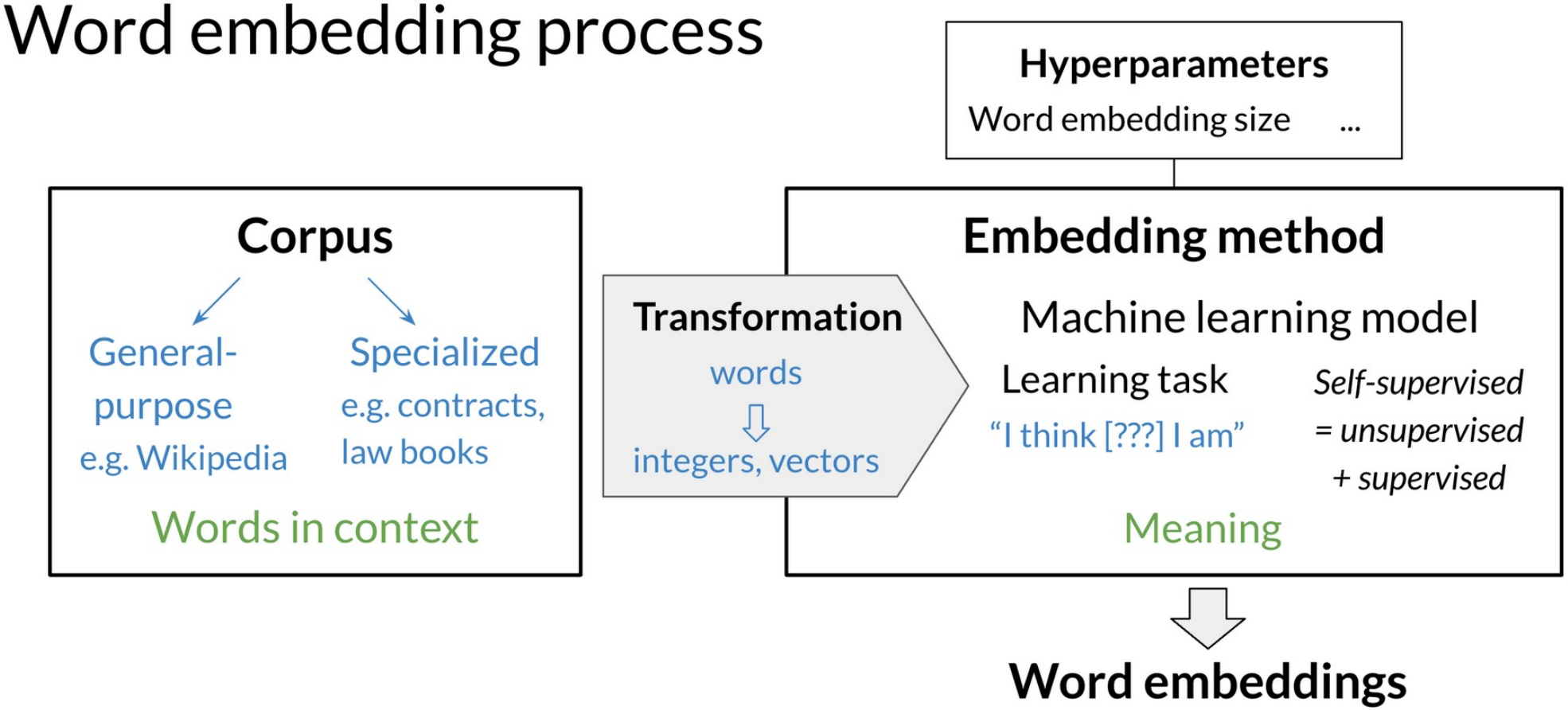
"word vectors"

word embeddings

Cómo crear *word embeddings*?

- Para crear incrustaciones de palabras, siempre se **necesita** un **corpus** de texto y un **método** de incrustación.
- El contexto de una palabra te dice qué tipo de palabras tienden a aparecer cerca de esa palabra específica. El contexto es importante ya que este es el que le dará sentido a cada palabra incrustada.
- Hay muchos tipos de **métodos** posibles que le permiten aprender ***word embeddings***. El modelo de aprendizaje automático realiza una tarea de aprendizaje y los principales subproductos de esta tarea son las incrustaciones de palabras. La tarea podría ser aprender a predecir una palabra en función de las palabras que la rodean en una oración del corpus, como en el caso de ***continuous bag-of-words***.
- La tarea es **autosupervisada**: no es supervisada en el sentido de que los datos de entrada (el corpus) no está etiquetado, y es supervisada en el sentido de que los datos mismos proporcionan el contexto necesario que normalmente formaría las etiquetas.
- Al entrenar vectores de palabras, hay algunos parámetros que debe ajustar. (por ejemplo, la dimensión del vector de palabras)

Cómo crear *word embeddings*?



Métodos clásicos

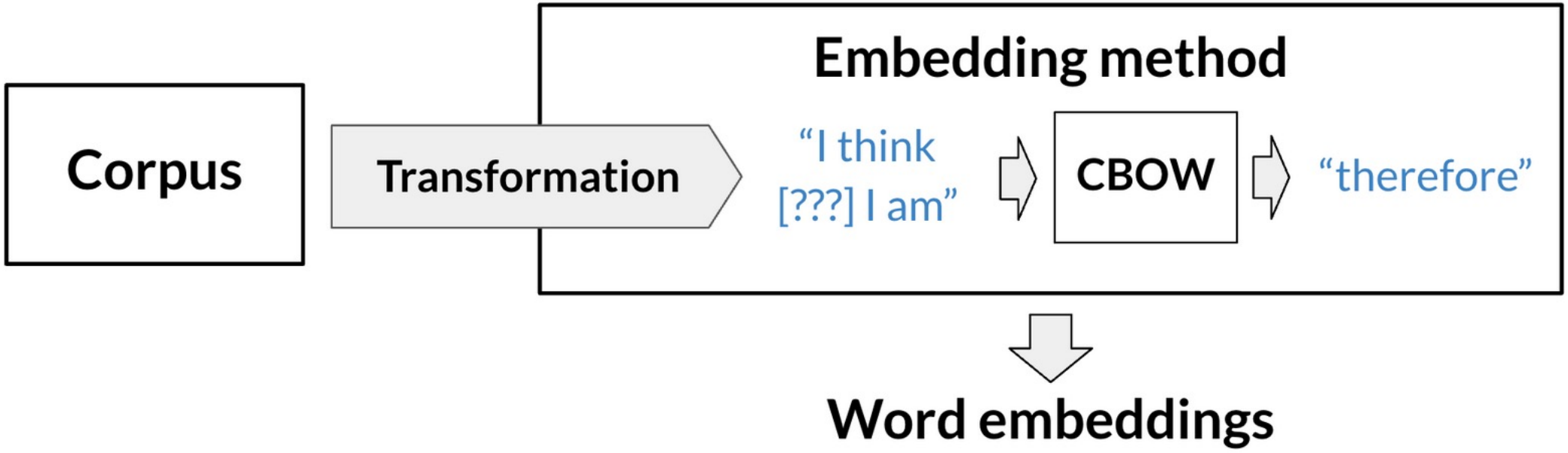
- **word2vec** (Google, 2013)
  - *Continuous bag-of-words (CBOW):* el modelo aprende a predecir la palabra central dadas algunas palabras de contexto.
  - *Continuous skip-gram / Skip-gram with negative sampling (SGNS)*: el modelo aprende a predecir las palabras que rodean una palabra de entrada dada.
- ***Global Vectors** (GloVe) (Stanford, 2014)*: factoriza el logaritmo de la matriz de co-ocurrencia de palabras del corpus, similar a la matriz de conteo que hemos usado antes.
- ***fastText** (Facebook, 2016):* basado en el modelo skip-gram y tiene en cuenta la estructura de las palabras al representar las palabras como un n-grama de caracteres. Admite palabras fuera de vocabulario (OOV).

Métodos basados en *deep learning*

- **Embeddings contextuales**
- En estos modelos más avanzados, las palabras tienen diferentes incrustaciones según su contexto. Pueden descargar incrustaciones previamente entrenadas (modelos pre-entrenados) para los siguientes modelos:
- BERT (Google, 2018):
- ELMo (Allen Institute for AI, 2018)
- GPT-2 (OpenAI, 2018)

Modelo de bolsa de palabras continua

- Para crear incrustaciones de palabras, necesita un corpus y un algoritmo de aprendizaje. El subproducto de esta tarea sería un conjunto de incrustaciones de palabras. En el caso del modelo de bolsa de palabras continua, el objetivo de la tarea es predecir una palabra que falta en función de las palabras que la rodean.



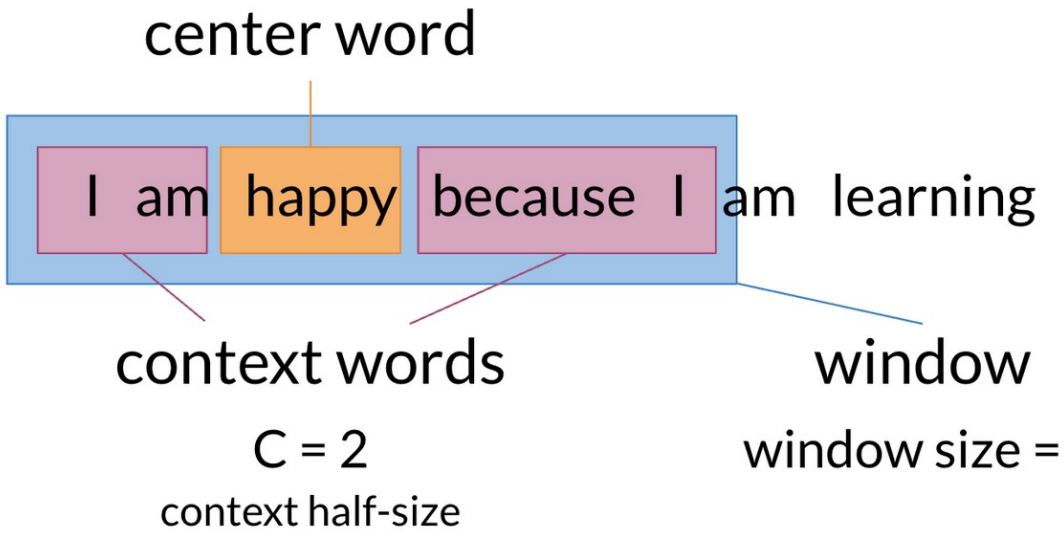
Modelo de bolsa de palabras continua

The little ? is barking

dog
puppy
hound
terrier
...

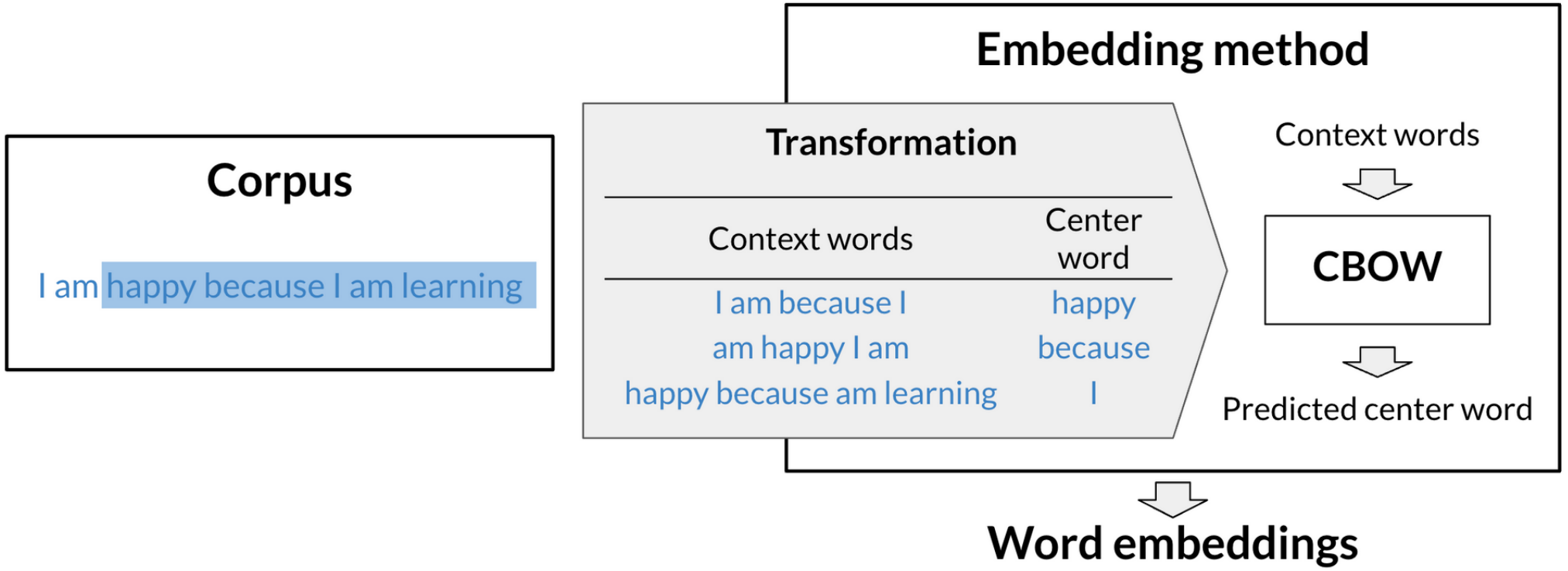
Modelo de bolsa de palabras continua

- Aquí hay una visualización que le muestra cómo funcionan los modelos.



- Como puede ver, el tamaño de la **ventana** en la imagen de arriba es 5. El tamaño del contexto, C, es 2. C generalmente le dice cuántas palabras antes o después de la palabra central usará el modelo para hacer la predicción.

Continuous Bag of Words Model



Visualización que muestra una descripción general del modelo.

Limpieza y tokenizacion

- Antes de implementar cualquier algoritmo de procesamiento de lenguaje natural, es posible que desdeseemos limpiar los datos y tokenizarlos. Aquí hay algunas cosas para realizar un seguimiento al manejar sus datos.
    - Letter case “The” == “the” == “THE” → *lowercase / upper case*
    - Punctuation , ! . ? → . “ ‘ « » ’ ” → ∅ … !! ??? → .
    - Numbers 1 2 3 5 8 → ∅ 3.14159 90210 → *as is/<NUMBER>*
    - Special characters ∇ $ € § ¶ ** → ∅
    - Special words 😊 #nlp → :happy: #nlp

Cleaning and Tokenization

- Puedes limpiar datos usando python de la siguiente manera :

```
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', '.']
```

- Pueden agregar tantas condiciones como desee en las líneas correspondientes al rectángulo verde de arriba.

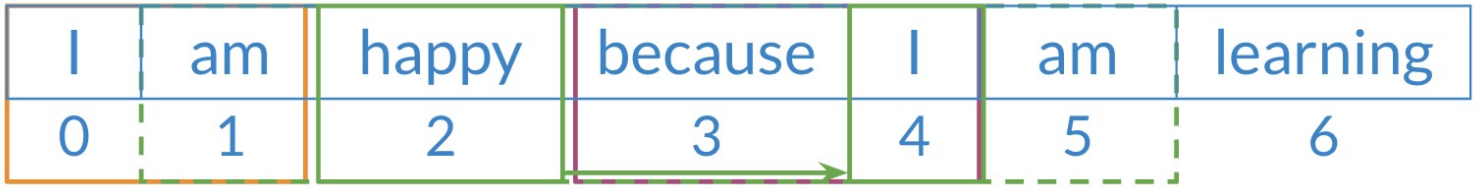
Ventana deslizante de palabras en Python

```
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
```

- **Ejemplo:** I am happy because I am learning
- El código muestra una función que toma dos parámetros.
  - **words**: una lista de palabras.
  - **C**: el tamaño del contexto.
- Primero comenzamos configurando *i* en C. Luego seleccionamos center_word y context_words. Luego obtenemos esos e incrementamos *i*.

Ventana deslizante de palabras en 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
```



- El código muestra una función que toma dos parámetros.
  - **words**: una lista de palabras.
  - **C**: el tamaño del contexto.
- Primero comenzamos configurando *i* en C. Luego seleccionamos center_word y context_words. Luego obtenemos esos e incrementamos *i*.

Ventana deslizante de palabras en Python

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

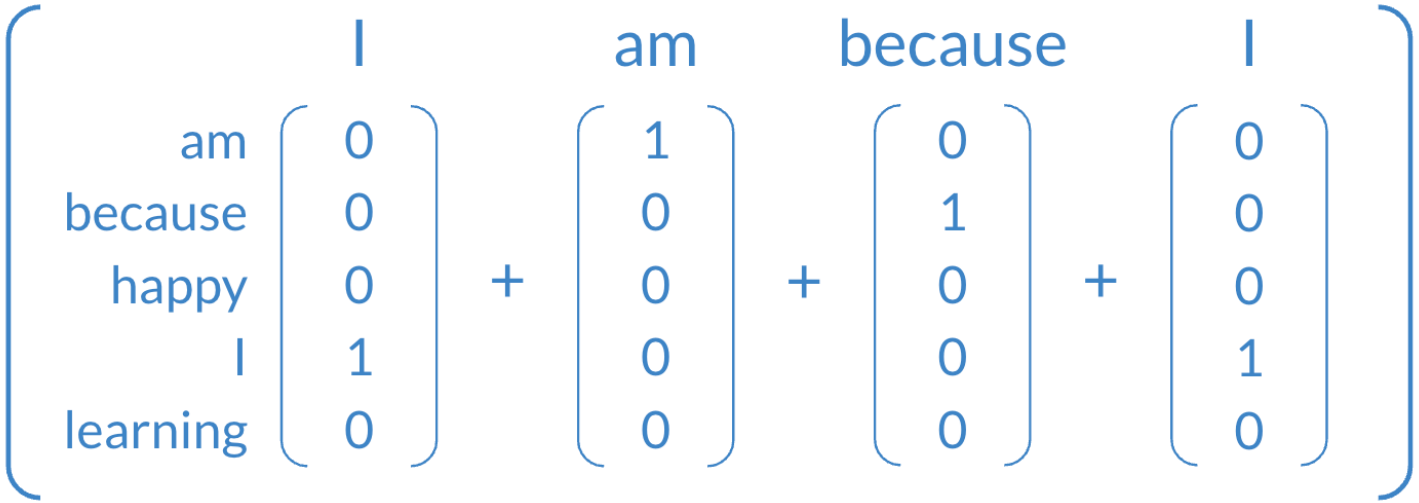
```
→ ['I', 'am', 'because', 'I']    happy
  ['am', 'happy', 'I', 'am']     because
  ['happy', 'because', 'am', 'learning']  I
```

Transformar palabras en vectores

- Corpus: I am happy because I am learning

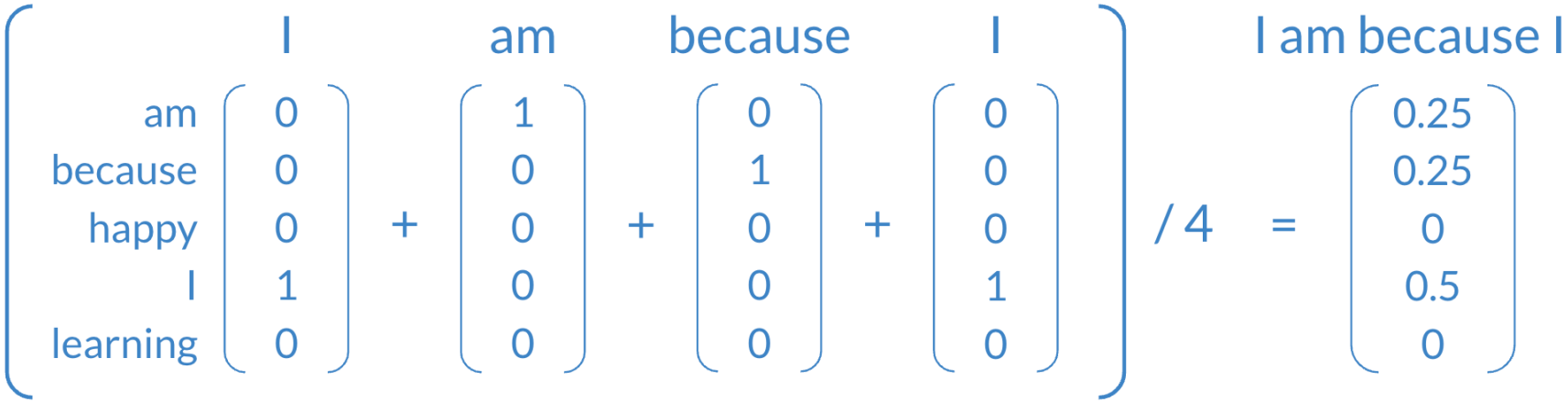
- Vocabulario: am, because, happy, I, learning

- Vectores One-hot



Transformar palabras en vectores

- Para transformar los vectores de contexto en un solo vector, puede usar lo siguiente.



- Como puede ver, comenzamos con vectores one-hot para las palabras de contexto y los transformamos en un solo vector tomando un promedio. Como resultado, terminas teniendo los siguientes vectores que puedes usar para tu entrenamiento.

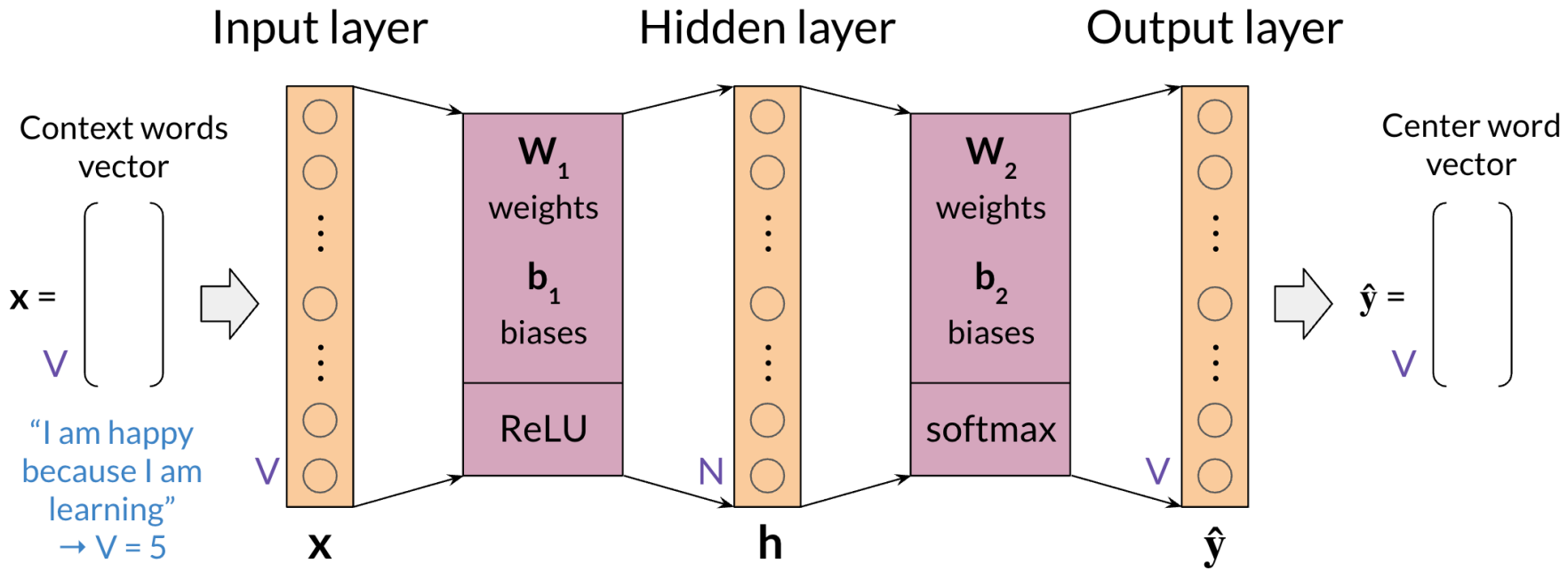
Context words	Context words vector	*Center word*	Center word vector
I am because I	[0.25; 0.25; 0; 0.5; 0]	*happy*	[0; 0; 1; 0; 0]

Conjunto de entrenamiento final

Context words	Context words vector	*Center word*	Center word vector
I am because I	[0.25; 0.25; 0; 0.5; 0]	*happy*	[0; 0; 1; 0; 0]
am happy I am	[0.5; 0; 0.25; 0.25; 0]	*because*	[0; 1; 0; 0; 0]
happy because am learning	[0.25; 0.25; 0.25; 0; 0.25]	*I*	[0; 0; 0; 1; 0]

Arquitectura para el modelo CBOW

Hiperparámetro:
N: tamaño del embedding



- Tiene una entrada, X, que es el promedio de todos los vectores de contexto. Luego lo multiplicas por W_1 y sumas b_1. El resultado pasa por una función ReLU para darle su capa oculta. Luego, esa capa se multiplica por W_2 y se suma b_2. El resultado pasa por un *softmax* que te da una distribución sobre *V*, palabras de vocabulario. Elige la palabra de vocabulario que corresponde al arg-max de la salida.

Dimensiones (entrada única)

- Las ecuaciones del modelo previo son:

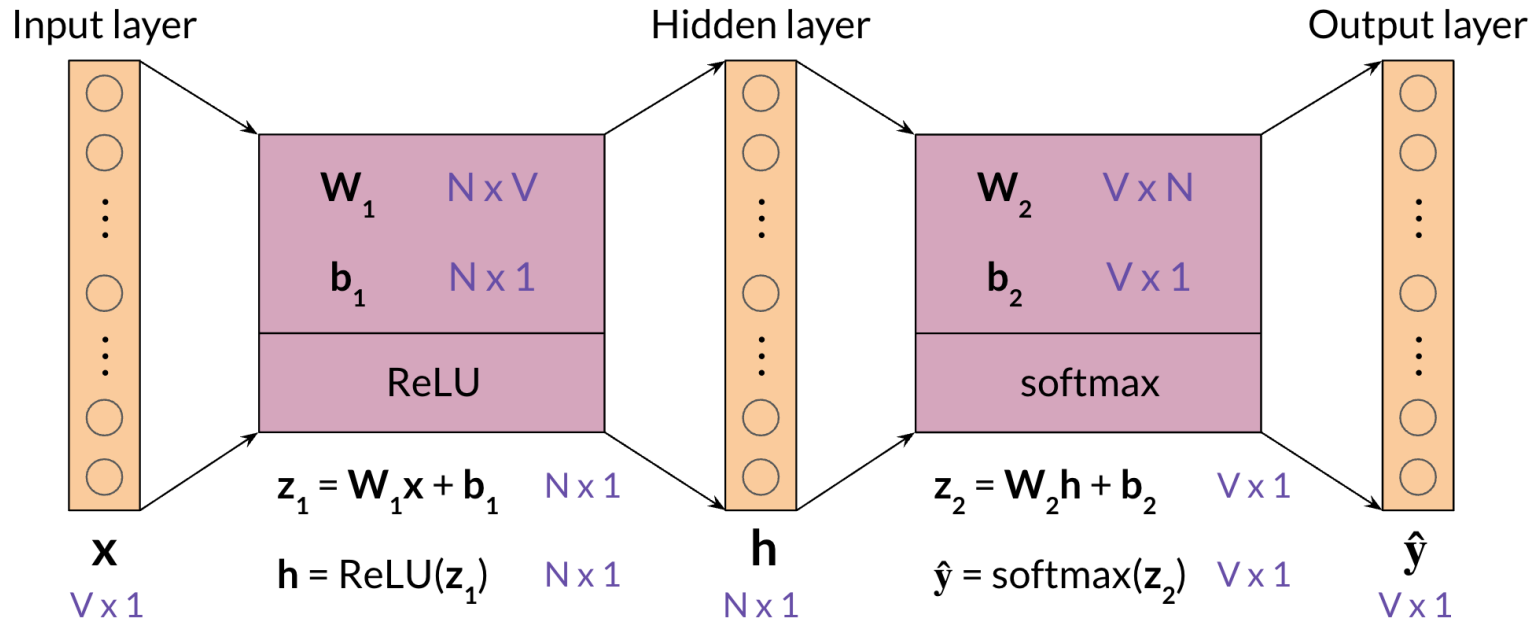
$$z_1 = W_1 x + b_1$$

$$h = ReLU(z_1)$$

$$z_2 = W_2 h + b_2$$

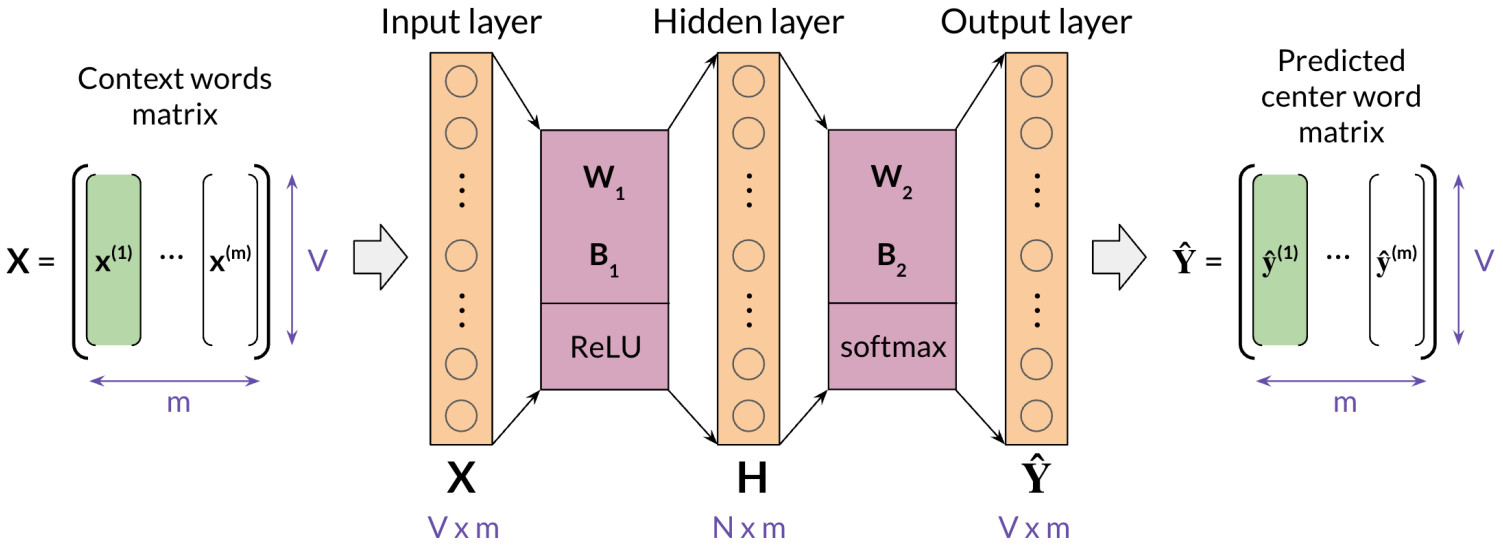
$$\hat{y} = softmax(z_2)$$

- Aquí pueden ver las dimensiones:



Dimensiones (lotes)

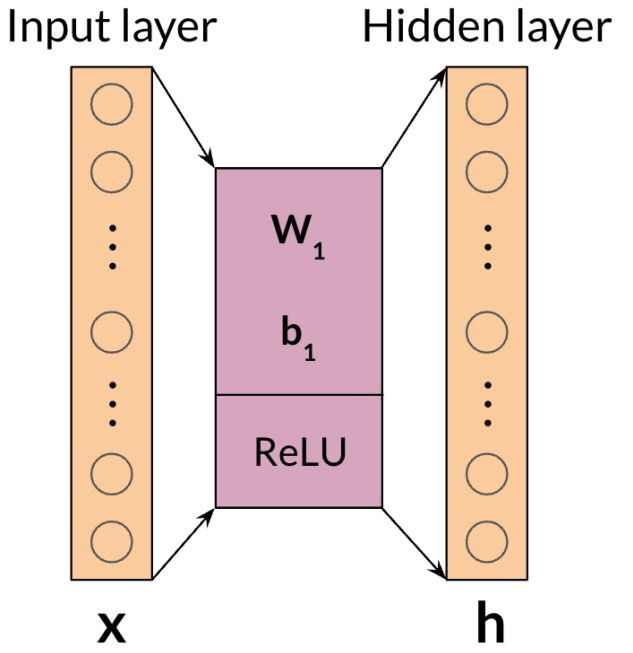
- Cuando se trata de entrada por lotes, puede apilar los ejemplos como columnas. A continuación, puede proceder a multiplicar las matrices de la siguiente manera:



- En el diagrama de arriba, puedes ver las dimensiones de cada matriz. Tenga en cuenta que su Y^ es de dimensión V x m. Cada columna es la predicción de la columna correspondiente a las palabras de contexto. Entonces, la primera columna de Y^ es la predicción correspondiente a la primera columna de X.

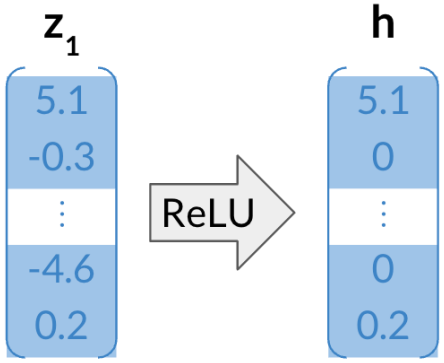
Funciones de activación: ReLU

- La unidad lineal rectificada (ReLU), es una de las funciones de activación más populares. Cuando alimenta un vector, a saber, x, en una función ReLU. Terminas tomando x=max(0,x). Este es un dibujo que muestra ReLU.

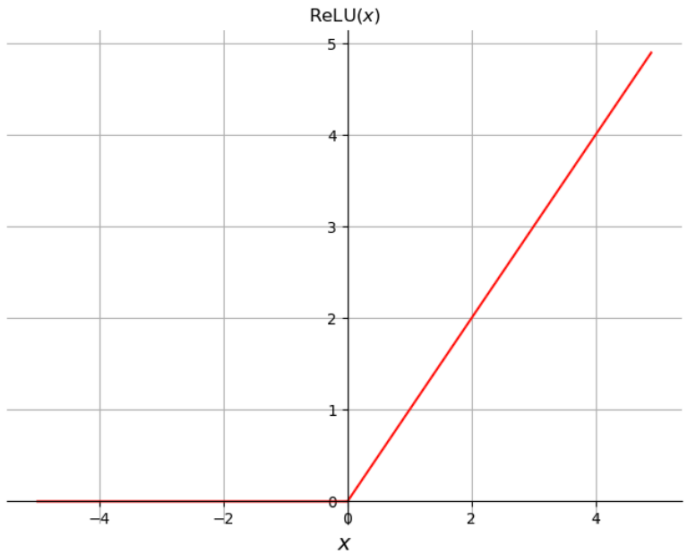


$$z_1 = W_1 x + b_1$$

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

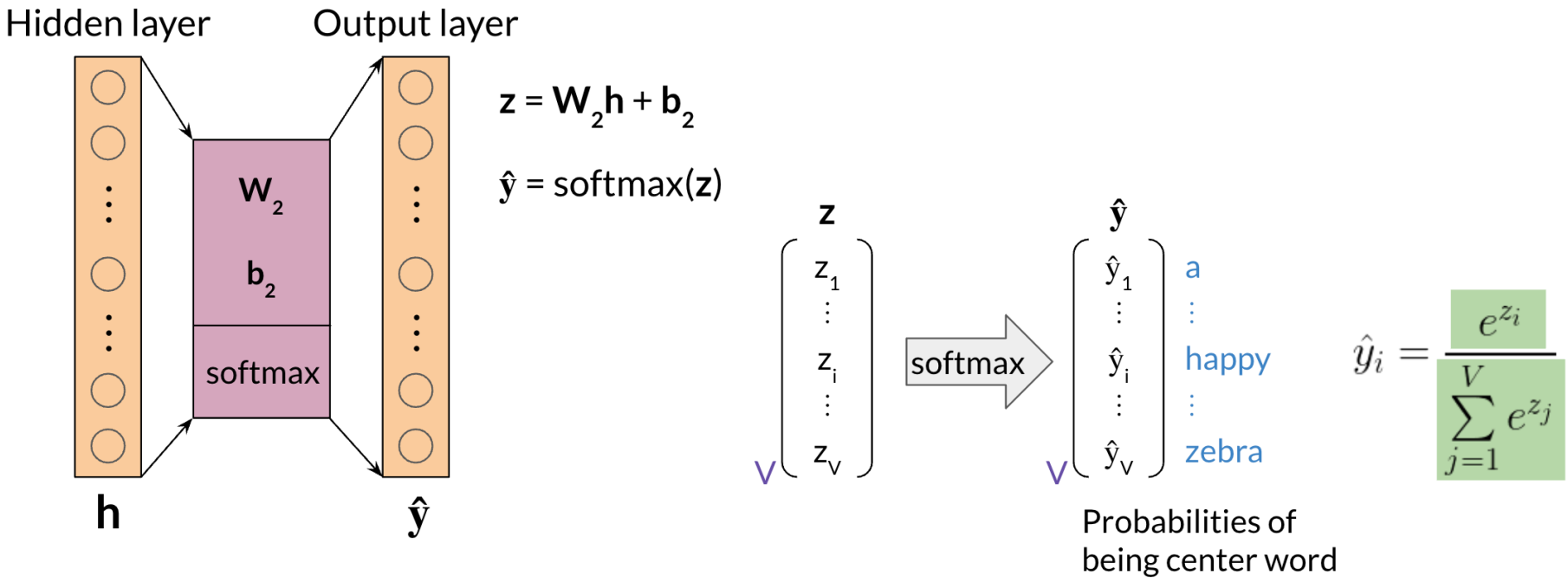


ReLU(x) = max(0, x)



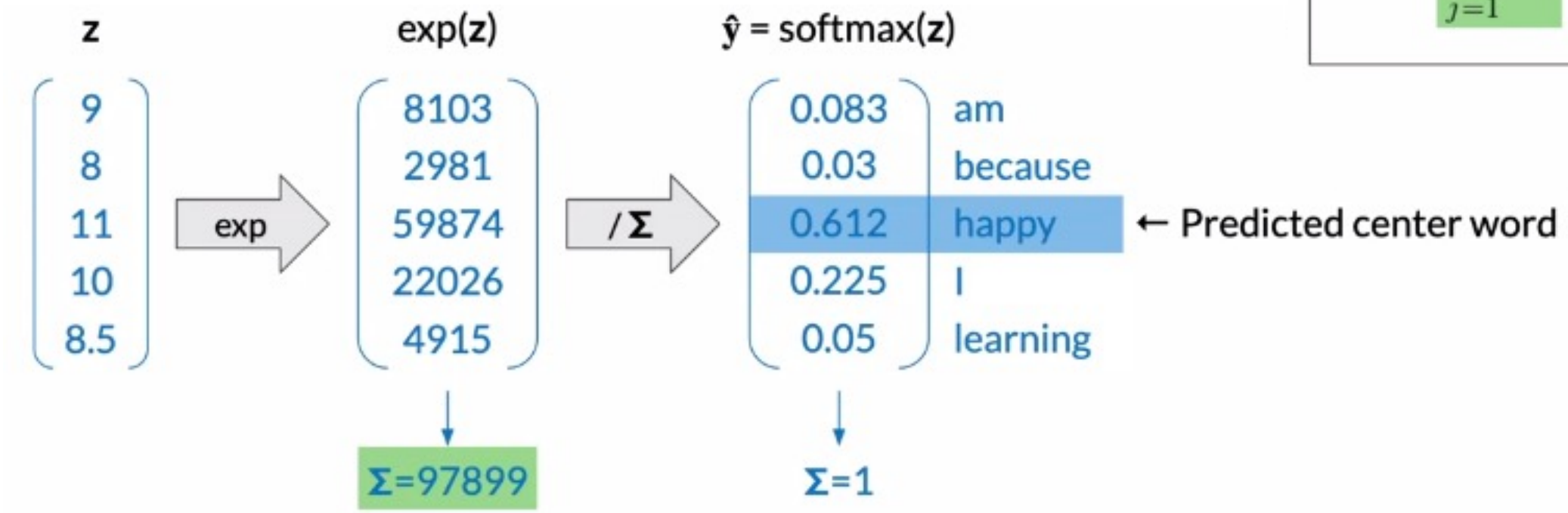
Funciones de activación: Softmax

- La función softmax toma un vector y lo transforma en una distribución de probabilidad. Por ejemplo, dado el siguiente vector z, puede transformarlo en una distribución de probabilidad de la siguiente manera.

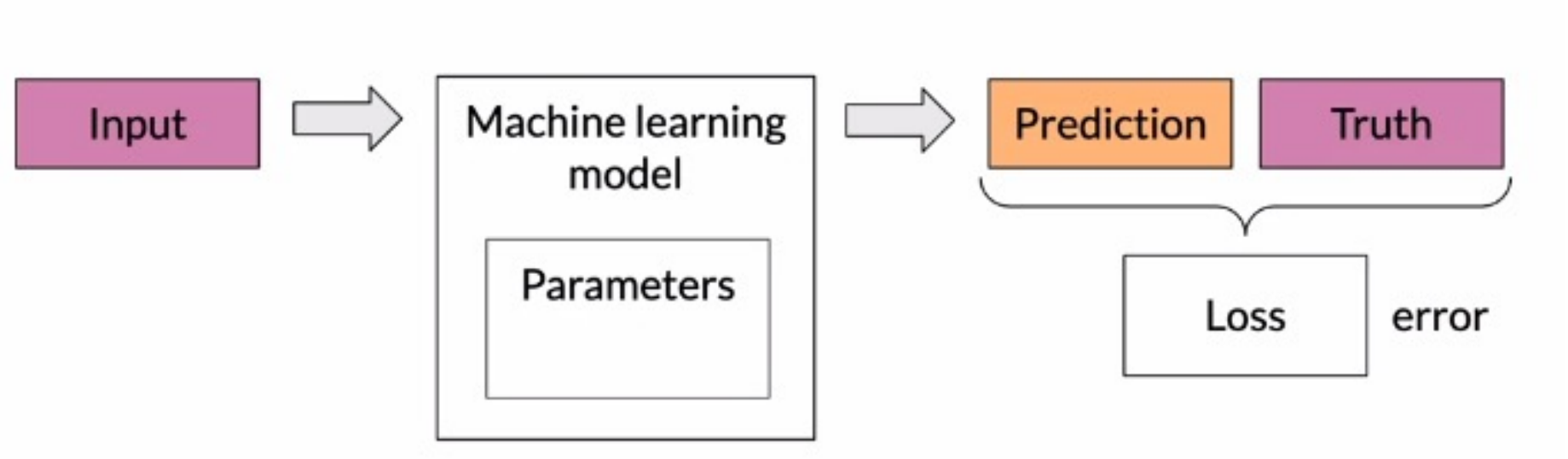


Funciones de activación: Softmax

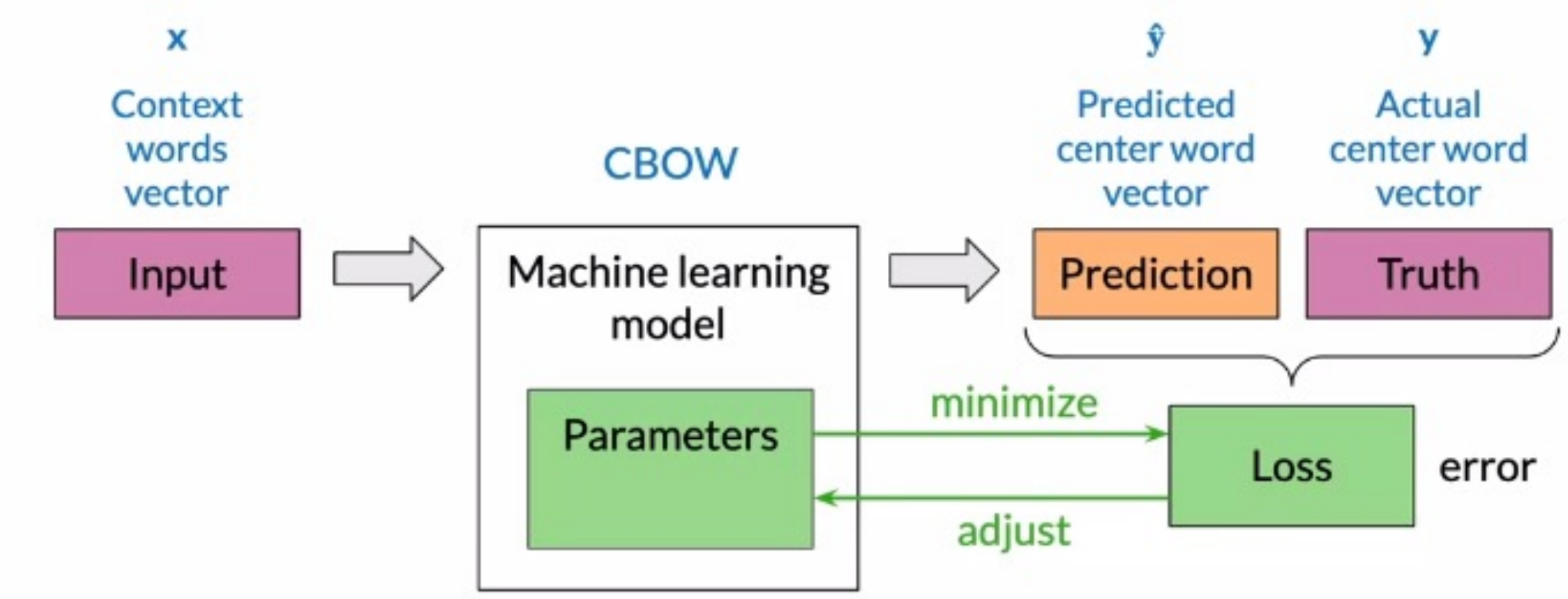
$$\hat{y}_i = \frac{e^{z_i}}{\sum_{j=1}^{V} e^{z_j}}$$



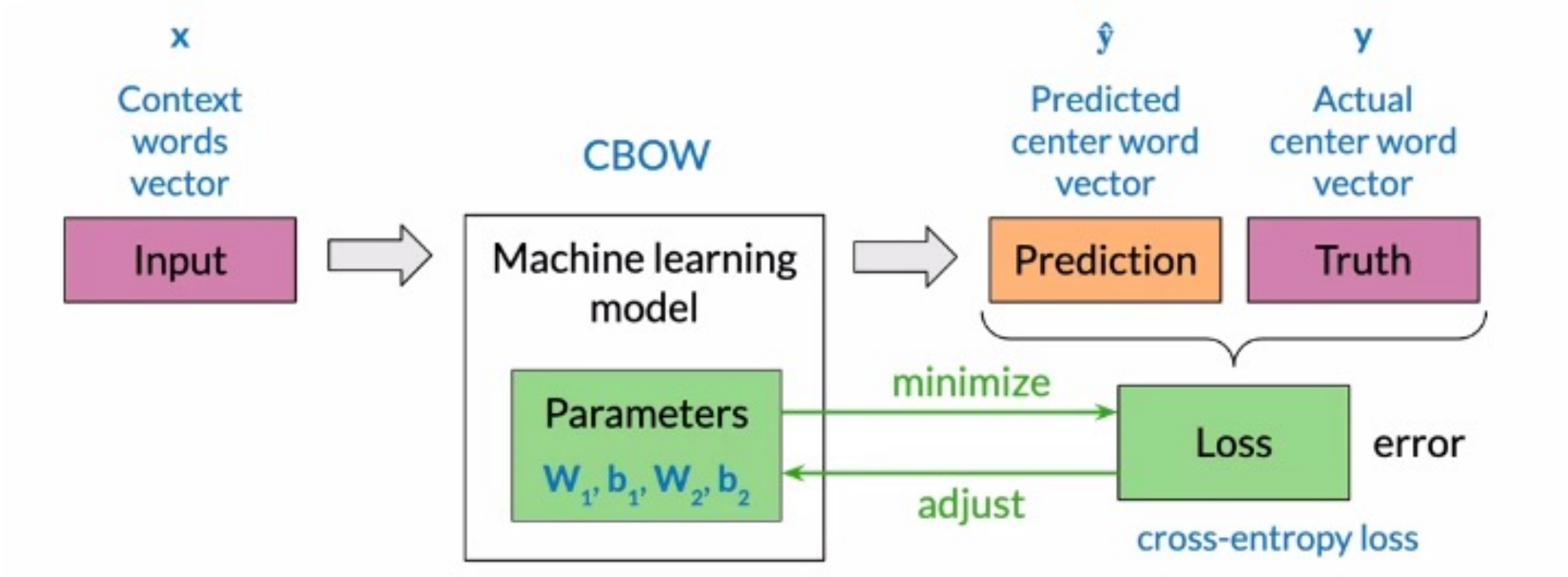
Pérdida


Input
Machine learning model
Parameters
Prediction
Truth
Loss
error

Pérdida


x
Context words vector
Input
CBOW
Machine learning model
Parameters
ŷ
Predicted center word vector
Prediction
y
Actual center word vector
Truth
minimize
adjust
Loss
error

Pérdida

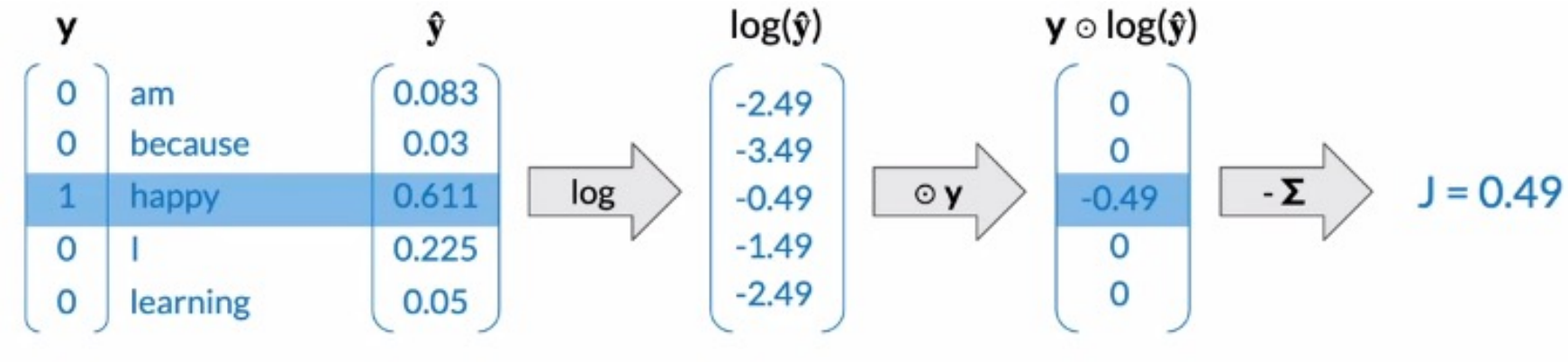

x
Context words vector
Input
CBOW
Machine learning model
Parameters
W_1, b_1, W_2, b_2
ŷ
Predicted center word vector
Prediction
y
Actual center word vector
Truth
minimize
adjust
Loss
error
cross-entropy loss

Pérdida de entropía cruzada

- La función de costo para el modelo CBOW es una pérdida de entropía cruzada definida como:
- Aquí hay un ejemplo donde usas la ecuación anterior.

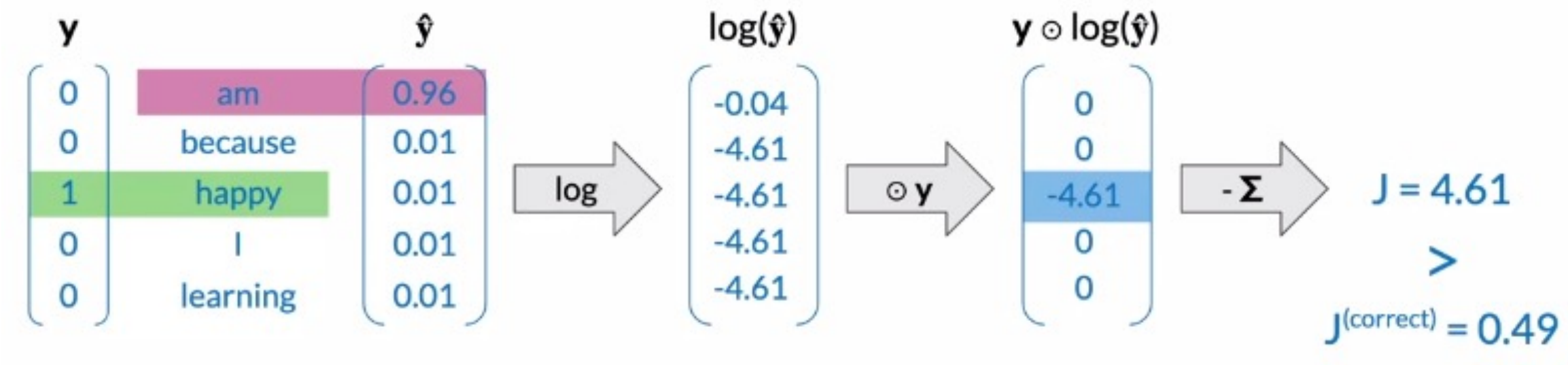
$$J = -\sum_{k=1}^{V} y_k \log \hat{y}_k$$

I am happy because I am learning



Pérdida de entropía cruzada

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

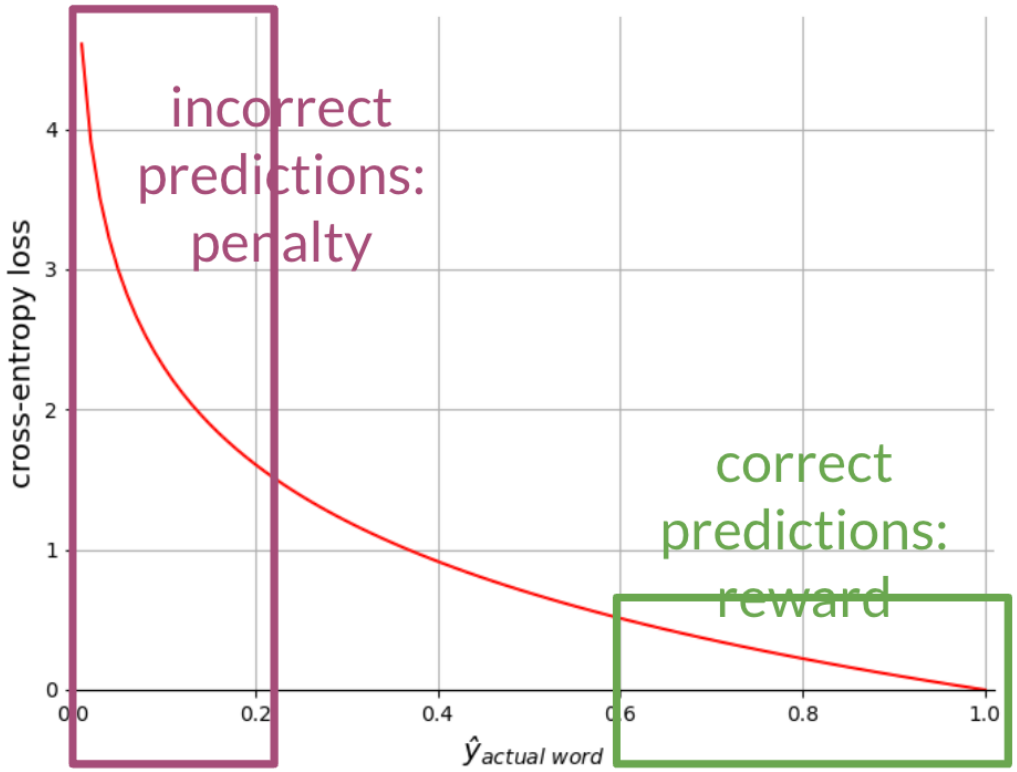


Función de costo

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

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

→ J = 4.61



Entrenamiento del modelo

- Forward propagation
- Cost
- Backpropagation and gradient descent

Entrenamiento del modelo: Forward Propagation

- Se define como

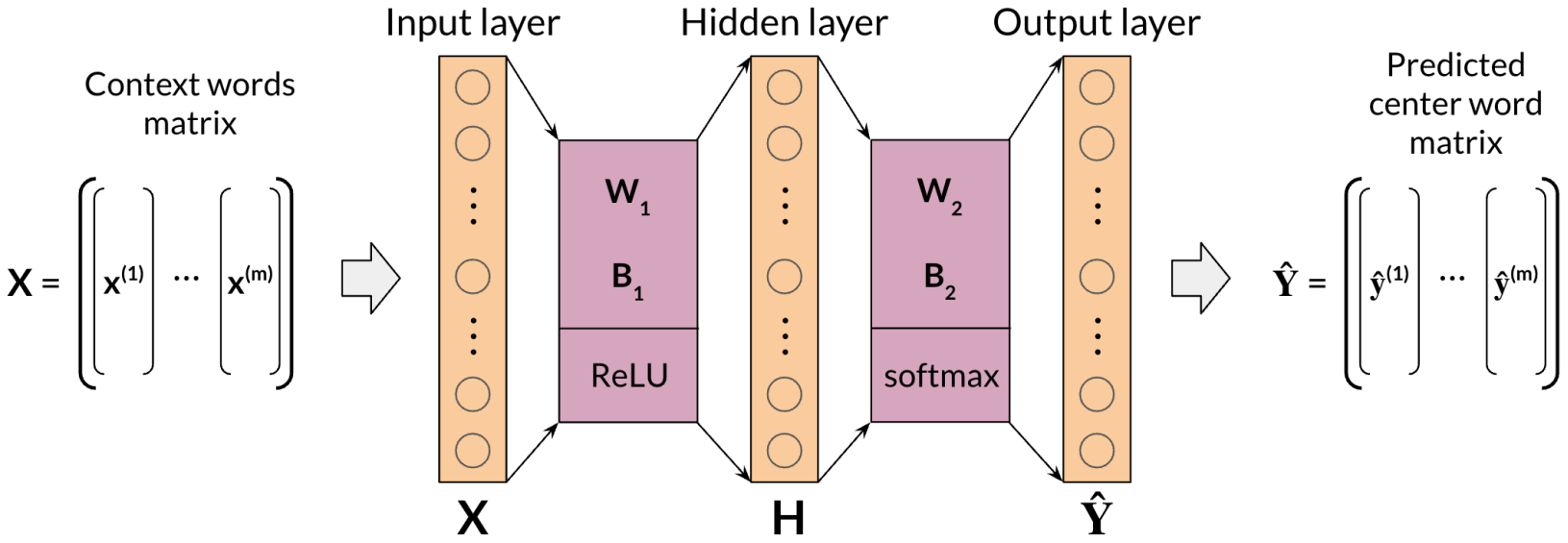
$$Z_1 = W_1 X + B_1$$

$$H = ReLU(Z_1)$$

$$Z_2 = W_2 H + B_2$$

$$\hat{Y} = softmax(Z_2)$$

- En la imagen a continuación, comienza desde la izquierda y se propaga hacia adelante hasta la derecha.



Entrenamiento del modelo: Costo

- Para calcular la pérdida de un lote, debe calcular lo siguiente:

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

- Dada su matriz de palabra central predicha y la matriz de palabra central real, se puede calcular la pérdida.

Predicted center word matrix

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

Actual center word matrix

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

Entrenamiento del modelo: Minimizar el costo

- Brackpropagation: calcula el costo parcial de la derivada con respecto a los pesos y al bias.

- Descenso del gradiente: actualiza los pesos y los bias

$$\frac{\partial J_{batch}}{\partial \mathbf{W_1}} = \frac{1}{m}\text{ReLU}\left(\mathbf{W_2}^\top(\hat{\mathbf{Y}} - \mathbf{Y})\right)\mathbf{X}^\top$$

$$\frac{\partial J_{batch}}{\partial \mathbf{W_2}} = \frac{1}{m}(\hat{\mathbf{Y}} - \mathbf{Y})\mathbf{H}^\top$$

$$\frac{\partial J_{batch}}{\partial \mathbf{b_1}} = \frac{1}{m}\text{ReLU}\left(\mathbf{W_2}^\top(\hat{\mathbf{Y}} - \mathbf{Y})\right)\mathbf{1}_m^\top$$

$$\frac{\partial J_{batch}}{\partial \mathbf{b_2}} = \frac{1}{m}(\hat{\mathbf{Y}} - \mathbf{Y})\mathbf{1}_m^\top$$

$$\mathbf{W_1} := \mathbf{W_1} - \alpha\frac{\partial J_{batch}}{\partial \mathbf{W_1}}$$

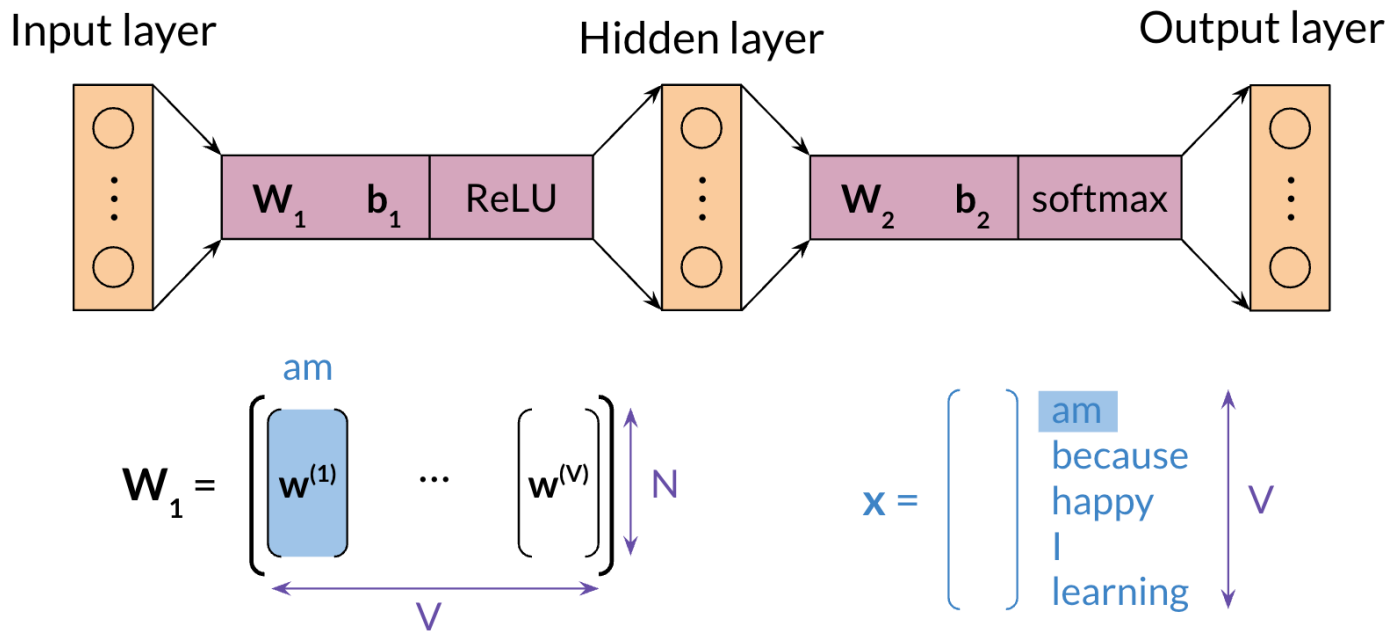
$$\mathbf{W_2} := \mathbf{W_2} - \alpha\frac{\partial J_{batch}}{\partial \mathbf{W_2}}$$

$$\mathbf{b_1} := \mathbf{b_1} - \alpha\frac{\partial J_{batch}}{\partial \mathbf{b_1}}$$

$$\mathbf{b_2} := \mathbf{b_2} - \alpha\frac{\partial J_{batch}}{\partial \mathbf{b_2}}$$

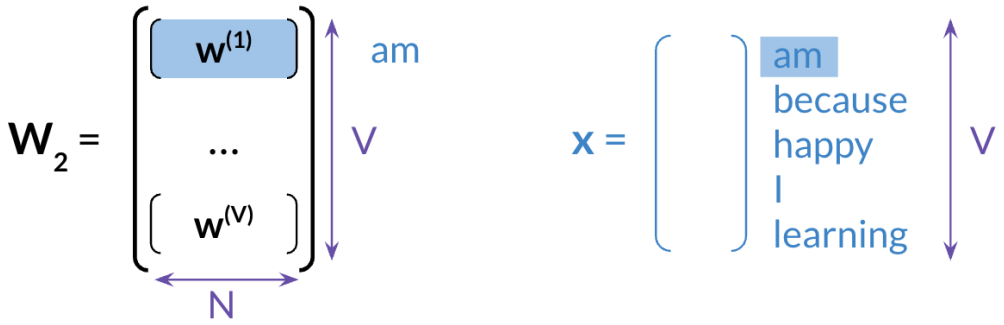
Extracción de incrustación de palabras

- Hay dos opciones para extraer incrustaciones de palabras después de entrenar el modelo de bolsa continua de palabras. Puede usar w_1 de la siguiente manera:

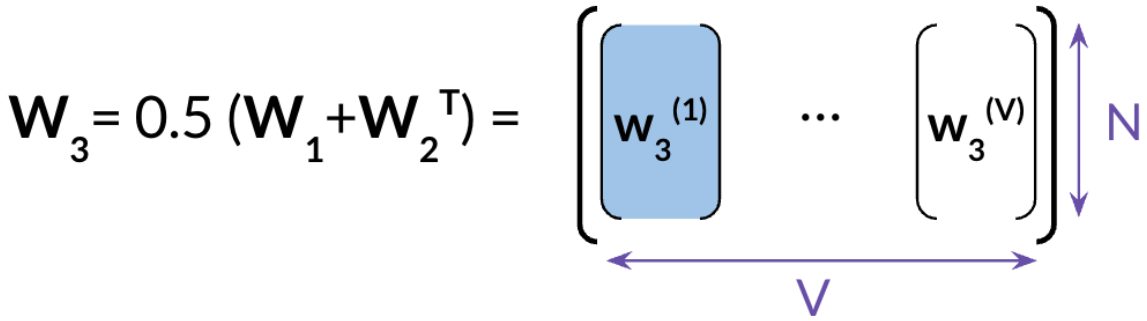


Extracción de incrustación de palabras

- Si tuviera que usar w_1, cada columna corresponderá a las incrustaciones de una palabra específica. También puede usar w_2 de la siguiente manera:



- La opción final es tomar un promedio de ambas matrices de la siguiente manera:



Evaluación de incrustaciones de palabras: evaluación intrínseca

- La evaluación **intrínseca** le permite probar las relaciones entre las palabras. Permite capturar analogías semánticas como "Francia" es a "París" como "Italia" es a <?> y también analogías sintácticas como "visto" es a "vió" como "es" es a <?>.
- Los casos ambiguos podrían ser mucho más difíciles de rastrear:

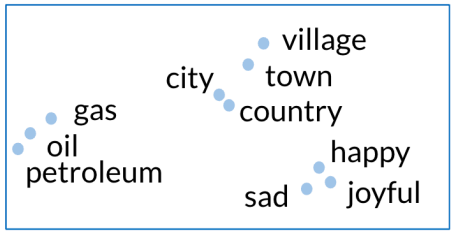
⚡ Ambiguity

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

- Aquí hay algunas formas que permiten usar la evaluación intrínseca.

Test relationships between words

- Analogies
- Clustering
- Visualization



Evaluación de incrustaciones de palabras: evaluación extrínseca

- La evaluación **extrínseca** prueba las incrustaciones de palabras en tareas externas como el reconocimiento de entidades nombradas, el etiquetado de partes del discurso, etc.
- + Evalúa la utilidad real de las incrustaciones
- - Pérdida de tiempo
- - Más difícil de solucionar problemas
- Así que ahora conoces la evaluación **intrínseca** y **extrínseca**.