

# 50.039 Theory and Practice of Deep Learning

## W8-S1 The Embedding Problem

Matthieu De Mari



# About this week (Week 8)

1. Why are **embeddings** an essential component of Neural Networks (NNs)?
2. Why are **good embeddings** difficult to produce?
3. What are the **conventional approaches to embeddings in NLP**?  
What can we **learn from these approaches**?
4. What are the **typical issues with embeddings** and how do we address them?
5. **State-of-the-art** of current embedding problems, and **open questions** in research.

# About this week (Week 8)

6. How do we evaluate the **quality/performance** of an embedding?
7. Can embeddings be **biased**?
8. Can we help the neural networks **identify the important parts of the context** to focus on?
9. What is **attention** in Neural Networks? What are **transformers** in Neural Networks?
10. What are the typical **uses for attention** these days?
11. What are the **limits of attention** and the **current research directions** on this topic?

# The embedding problem

## Definition (the **embedding** problem):

Neural Networks are designed to operate with tensors of numerical values (vectors, matrices, arrays, etc.). Our training process relies on processing numerical values, giving “sense/logic” to the Neural Network.

In some simple scenarios (nicely formatted data, images, etc.), inputs can be passed directly to a Neural Network, without modifications.

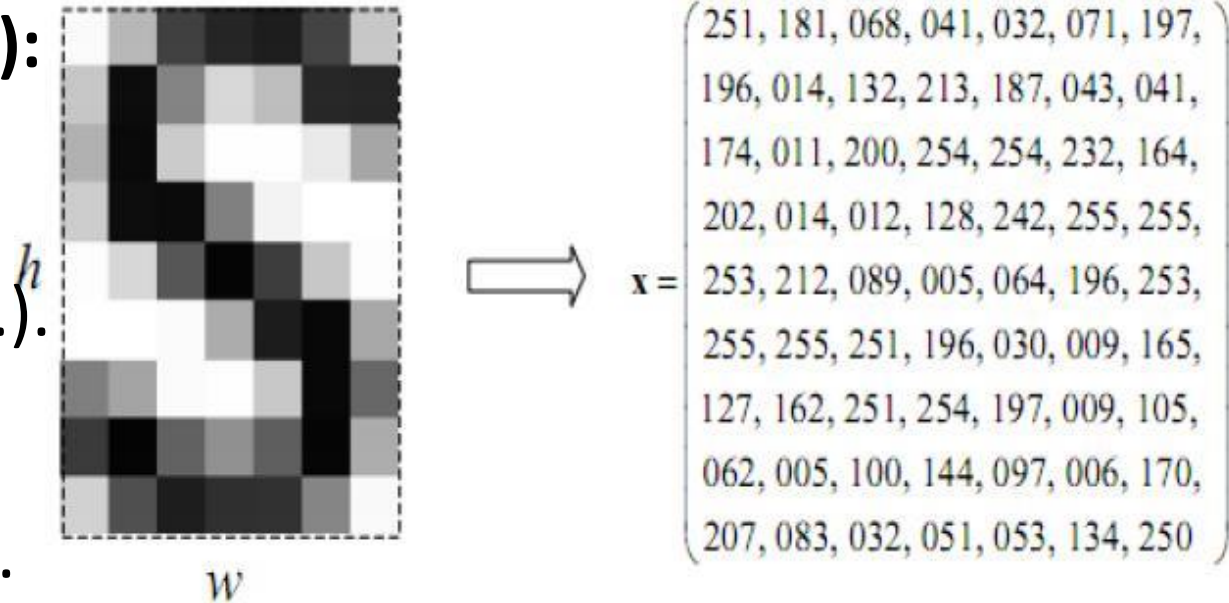
$y$ = Apartment Price (in \$)	500 000
$x_1$ = Age of the apartment (in years)	10
$x_2$ = Distance to closest MRT (in meters)	780
$x_3$ = Surface of the apartment (in sqm)	75
$x_4$ = Number of bedrooms	2
$x_5$ = Distance to closest Fairprice (in meters)	652
$x_6$ = Number of previous owners	0
$x_7$ = Number of balconies	1

# The embedding problem

## Definition (the **embedding** problem):

Neural Networks are designed to operate with tensors of numerical values (vectors, matrices, arrays, etc.). Our training process relies on processing numerical values, giving “sense/logic” to the Neural Network.

In some simple scenarios (nicely formatted data, images, etc.), inputs can be passed directly to a Neural Network, without modifications.



CNN of some sort

$y$  = index of the letter in the alphabet

# The embedding problem

## Definition (the **embedding** problem):

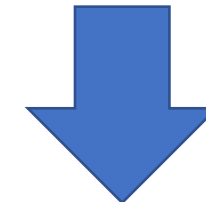
Neural Networks are designed to operate with tensors of numerical values (vectors, matrices, arrays, etc.). Our training process relies on processing numerical values, giving “sense/logic” to the Neural Network.

In some simple scenarios (nicely formatted data, images, etc.), inputs can be passed directly to a Neural Network, without modifications.



$x$  = list of successive values over days

$x = [8000, 7995, 8002, 8005, 8012, 8010, 8014, \dots]$



RNN/TCN of some sort

$y$  = next value for the list (i.e. market prediction)

$y = 8020$

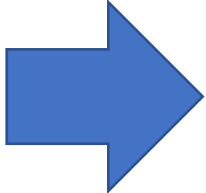
# The embedding problem

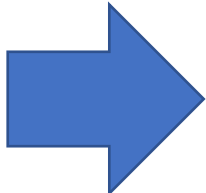
## Definition (the **embedding** problem):

Neural Networks are designed to operate with tensors of numerical values (vectors, matrices, arrays, etc.). Our training process relies on processing numerical values, giving “sense/logic” to the Neural Network.

In some simple scenarios (nicely formatted data, images, etc.), inputs can be passed directly to a Neural Network, without modifications.

But, what if the input is a string of text (a word or a sentence)? How do we convert text into numerical values?

“Hello”   $x = ?$

“The quick brown fox jumps over the lazy dog”   $x = ?$

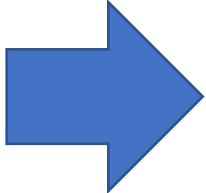
# The embedding problem

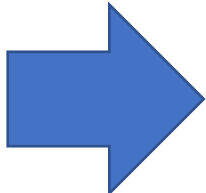
## Definition (the **embedding** problem):

Neural Networks are designed to operate with tensors of numerical values (vectors, matrices, arrays, etc.). Our training process relies on processing numerical values, giving “sense/logic” to the Neural Network.

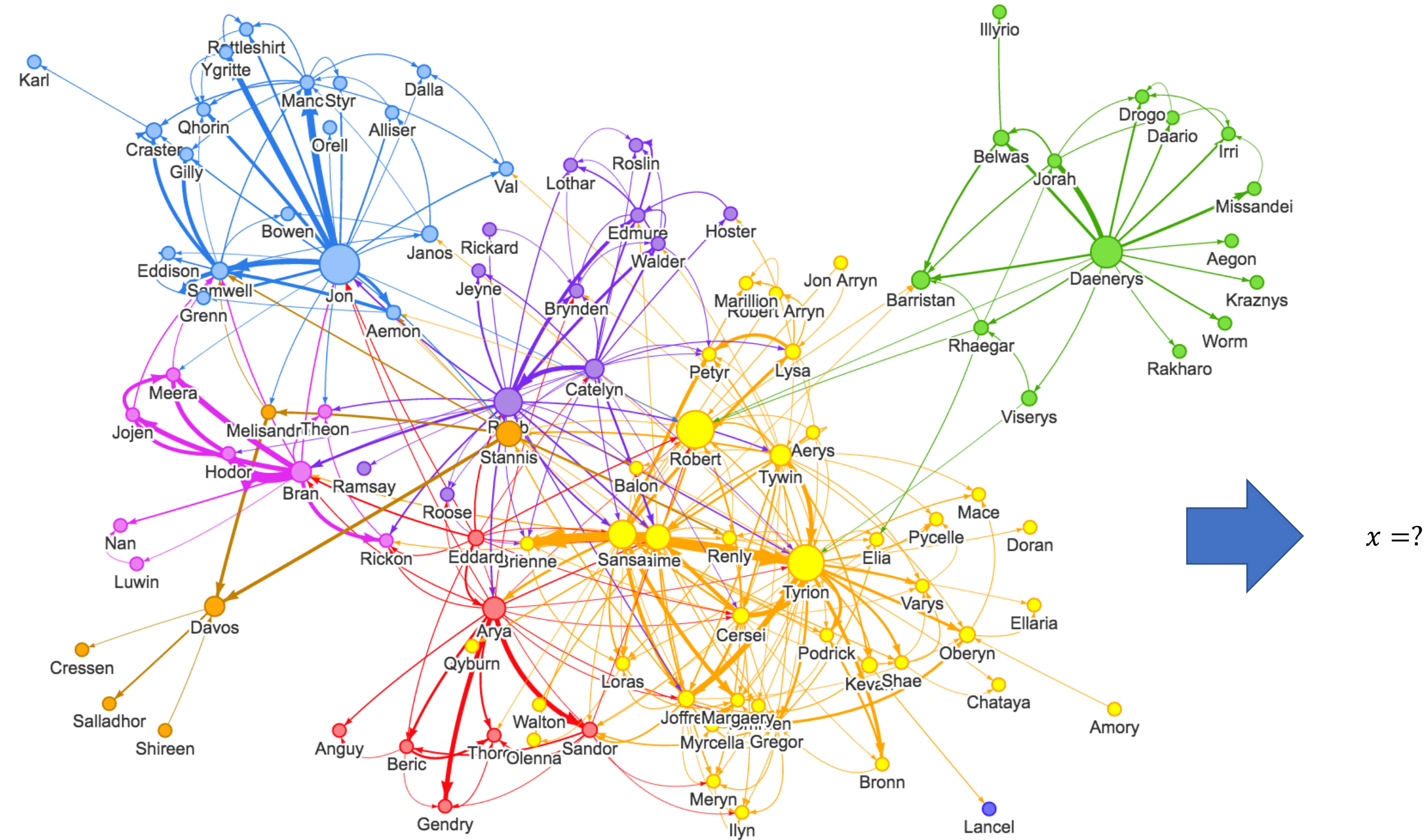
**Revision: unfortunately, it is often the case that the data cannot be simply formatted as a tensor. That is the **embedding** problem.**

But, what if the input is a string of text (a word or a sentence)? How do we convert text into numerical values?

“Hello”   $x = ?$

“The quick brown fox jumps over the lazy dog”   $x = ?$



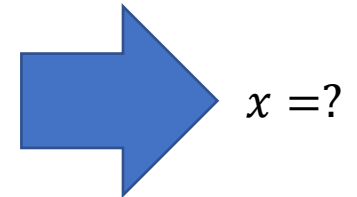


# The embedding problem

## Definition (the **embedding** problem):

Neural Networks are designed to operate with tensors (vectors, matrices, arrays, etc.) of numerical values. Our training process relies on processing numerical values, giving “sense/logic” to the Neural Network.

**Revision:** unfortunately, it is often the case that data cannot be simply formatted as a tensor. That is the **embedding** problem.



$x = ?$

# Let us start with a toy problem

Let us consider the following problem: **the recommendation system from Netflix.**

- Our dataset is a list of movies, which can be watched on Netflix and that the user may have seen and/or liked.
- Our objective is to suggest a recommendation of what our user would like to watch next.





# Let us start with a toy problem

Let us consider the following problem: **the recommendation system from Netflix.**

- Our dataset is a list of movies, which can be watched on Netflix and that the user may have seen and/or liked.
- Our objective is to suggest a recommendation of what our user would like to watch next.

**Question:** How do I represent a movie as a tensor  $x$ , carrying **meaningful values**?

- Values, which can later be used to describe some **similarity** between movies.
- Values, which can later be used to identify **preferences** of the user.
- Values, which later can be used to suggest **recommendations**.

# Embeddings, a mathematical definition

## Definition (**embedding**):

Mathematically speaking, an **embedding** refers to a **function**, which **transforms** an **object**  $x$ , into another **object**  $x'$ .

The result of the embedding,  $x'$ , is supposed to be mathematically more tractable and useful than  $x$ .

Embeddings are typically used in Machine Learning to **encode some non-mathematical data** into a meaningful tensor, which can later be fed to a mathematical model (often a NN).

$x =$



$x' = \text{tensor of some sort?}$

# Embeddings, a mathematical definition

## Definition (**embedding**):

Mathematically speaking, an **embedding** refers to a **function**, which **transforms** an **object**  $x$ , into another **object**  $x'$ .

The result of the embedding,  $x'$ , is supposed to be mathematically more tractable and useful than  $x$ .

In addition, it is often preferable for embeddings to be **injective**, or ideally, **bijective**.

Embeddings are typically used in Machine Learning to **encode some non-mathematical data** into a meaningful tensor, which can later be fed to a mathematical model (often a NN).

$x =$



$x' = \text{tensor of some sort?}$

# Embeddings, a mathematical definition

## Definition (**embedding**):

Mathematically speaking, an **embedding** refers to a **function**, which **transforms** an **object**  $x$ , into another **object**  $x'$ .

The result of the embedding,  $x'$ , is supposed to be mathematically more tractable and useful than  $x$ .

In addition, it is often preferable for embeddings to be **injective**, or ideally, **bijective**.

## Definition (**injective** function):

An **injective** function is a function  $f: X \rightarrow X'$ , that maps **distinct elements of its domain**  $X$  to **distinct elements of its codomain**  $X'$ .

This means that  $\forall x_1, x_2 \in X$ ,  
$$x_1 \neq x_2 \Rightarrow f(x_1) \neq f(x_2)$$

Or equivalently,  $\forall x_1, x_2 \in X$ ,

$$f(x_1) = f(x_2) \Rightarrow x_1 = x_2$$

# Embeddings, a mathematical definition

## Definition (**embedding**):

Mathematically speaking, an **embedding** refers to a **function**, which **transforms** an **object**  $x$ , into another **object**  $x'$ .

The result of the embedding,  $x'$ , is supposed to be mathematically more tractable and useful than  $x$ .

In addition, it is often preferable for embeddings to be **injective**, or ideally, **bijective**.

## Definition (**surjective** function):

A **surjective** function is a function  $f: X \rightarrow X'$ , that maps **all elements from the codomain  $X'$**  to **at least one value of the domain  $X$** .

This means that

$$\forall x' \in X', \exists x \in X \text{ s.t. } f(x) = x'.$$



# Embeddings, a mathematical definition

## Definition (**embedding**):

Mathematically speaking, an **embedding** refers to a **function**, which **transforms** an **object**  $x$ , into another **object**  $x'$ .

The result of the embedding,  $x'$ , is supposed to be mathematically more tractable and useful than  $x$ .

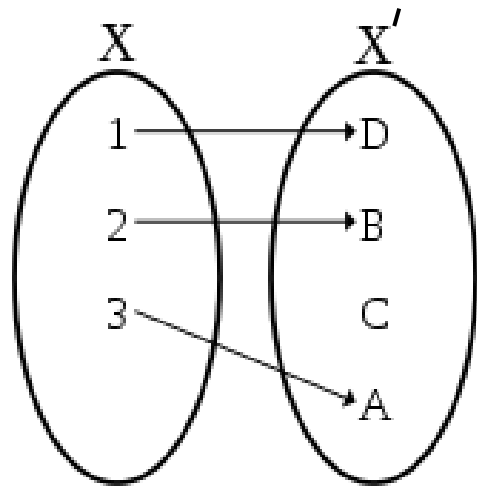
In addition, it is often preferable for embeddings to be **injective**, or ideally, **bijjective**.

## Definition (**bijjective** function):

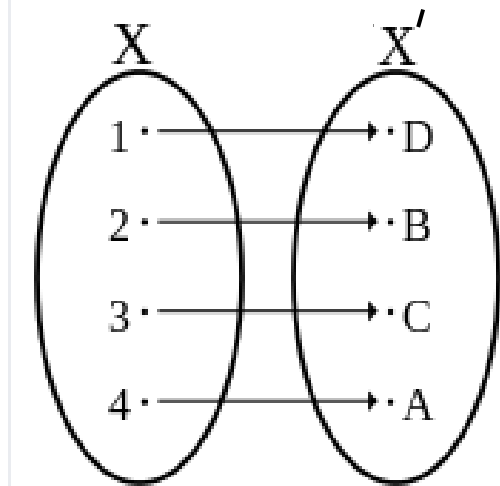
A **bijjective** function is a function  $f: X \rightarrow X'$ , that is **both injective** and **surjective**.

Bijjective functions define a **one-to-one mapping** between all elements in both the domain  $X$  and the codomain  $X'$ .

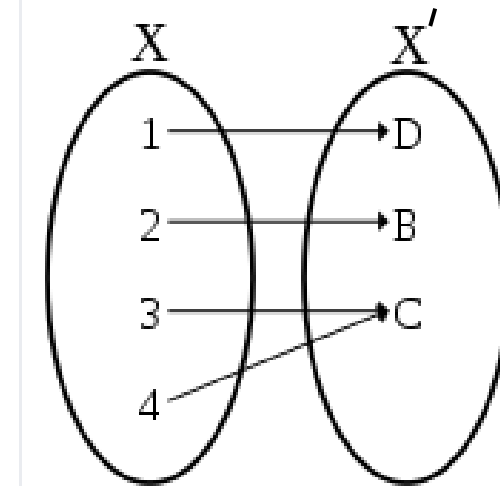
# Injective, surjective and bijective functions



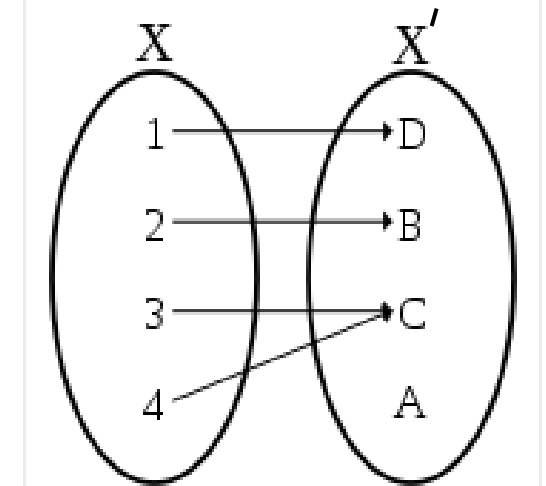
An injective non-surjective function  
(injection, not a bijection)



An injective surjective function  
(bijection)



A non-injective surjective function  
(surjection, not a bijection)



A non-injective non-surjective function  
(also not a bijection)

# Embeddings, a mathematical definition

**Theorem #1 (bijective functions and inverse functions):**

A function  $f: X \rightarrow X'$ , admits an inverse function  $f^{-1}: X' \rightarrow X$  if and only if said function  $f$  is **bijective**.

This means that our embedding converts any element  $x$  into a single and different element  $x'$  every time, and vice-versa.

# Embeddings, a mathematical definition

- **Non-surjective** would mean there is an element in  $X'$ , which cannot be mapped to an element in  $X$ . And vice-versa.
- This would be annoying for our recommendation system, as a **predicted vector  $x'$  could therefore have no meaning** (i.e. our recommendation AI could produce a vector  $x'$ , which corresponds to no movie of our Netflix catalogue).

# Embeddings, a mathematical definition

- **Non-injective** would mean that an element in  $X'$  could be mapped to multiple values in  $X$ .
- This would be annoying for our recommendation system, as a **predicted vector  $x'$  could therefore have multiple meanings** (i.e. our recommendation AI could produce a vector  $x'$  corresponding to more than one movie of our Netflix catalogue).

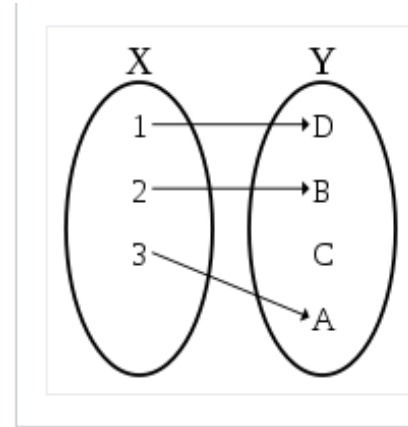
# Embeddings, a mathematical definition

## Theorem #2 (**bijjective** functions and **domain/codomain size**):

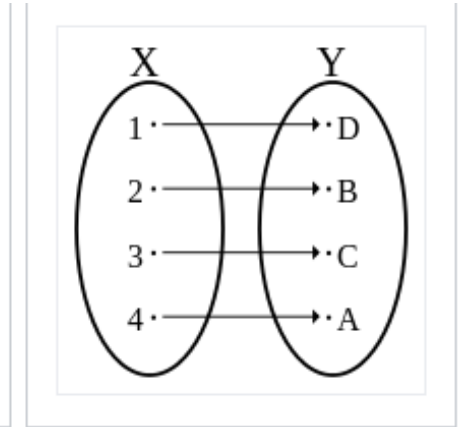
Consider two **countable** sets,  $X$  and  $X'$ .

(Countable sets are sets whose elements could be counted, or labeled as 1, 2, 3,...)

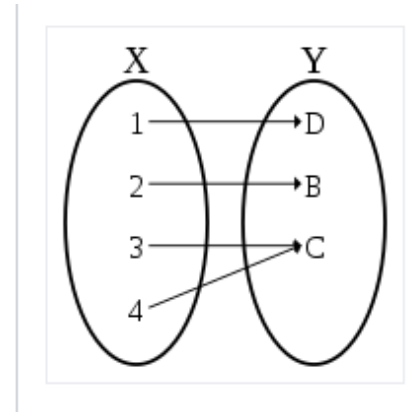
If there exists a **bijjective function**  $f: X \rightarrow X'$ , then  $X$  and  $X'$  have the **same cardinality**, i.e. the **same number of elements**.



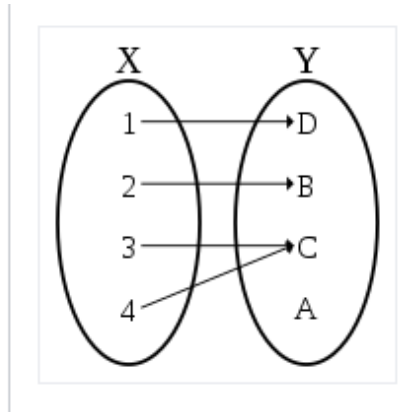
An injective non-surjective function (injection, not a bijection)



An injective surjective function (bijection)



A non-injective surjective function (surjection, not a bijection)



A non-injective non-surjective function (also not a bijection)

# Embeddings, a mathematical definition

## Definition (**embedding**):

Mathematically speaking, an **embedding** refers to a **function**, which **transforms** an **object**  $x$ , into another **object**  $x'$ .

The result of the embedding,  $x'$ , often is mathematically more tractable and useful than  $x$ .

In addition, it is often preferable for embeddings to be **injective**, or ideally, **bijective**.

## Definition (**bijective embedding**):

**Bijective embeddings** are often preferable as they allow for

- Any element  $x \in X$  to be **encoded** as a unique  $x' \in X'$ , with  $y = f(x)$
- Any element in  $x' \in X'$  to be **decoded** as a unique  $x \in X$ , with  $x = f^{-1}(x')$ .

(That is a great mathematical property to have! But not always possible unfortunately...)

# Manual embedding

## Definition (**Manual Embeddings**):

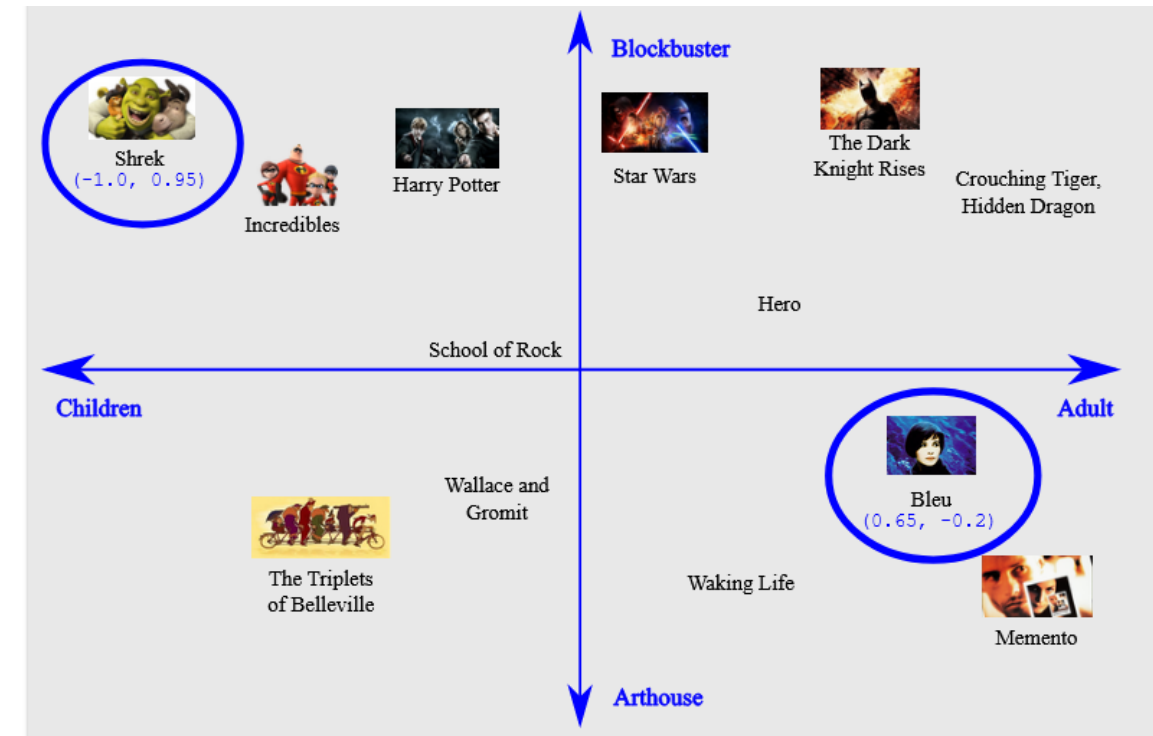
- In ML, **manual embedding** refers to manually extracting meaningful data  $y$  from an object  $x$ , whose format cannot be fed to a NN.



# Manual embedding

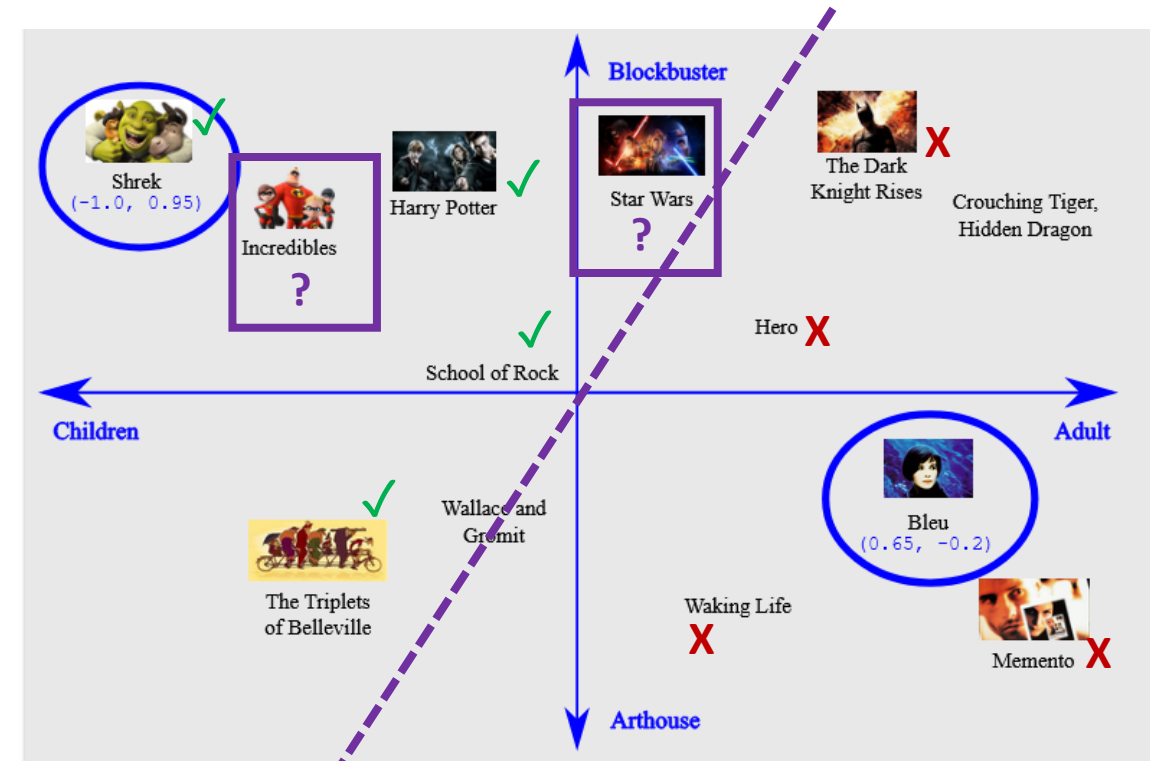
## Definition (Manual Embeddings):

- In ML, **manual embedding** refers to manually extracting meaningful data  $y$  from an object  $x$ , whose format cannot be fed to a NN.
- For instance, let us assume, we give the following scores to movies, with values in  $[-1, 1]$ .
- First score  $x_1$ : target audience (-1 if for kids only, 1 for adults only)
- Second score  $x_2$ : blockbuster vs. arthouse type of movie.



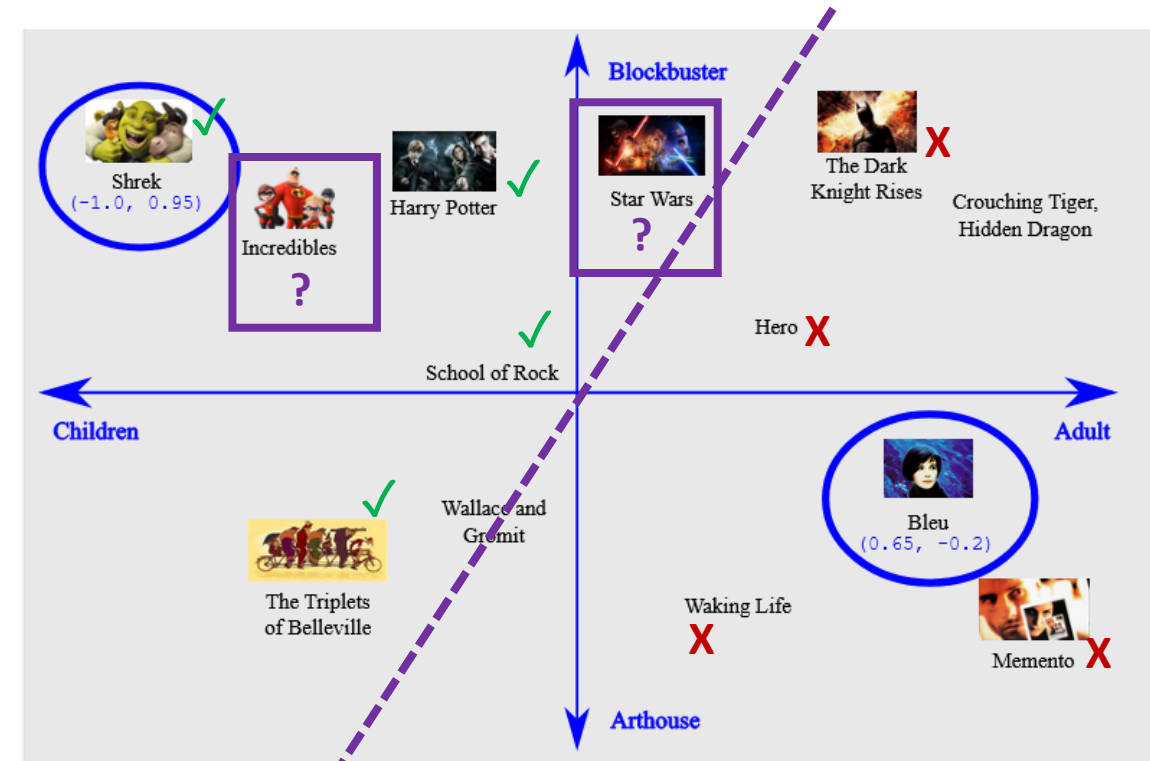
# Manual embedding

- Using this manual embedding, we can then train a binary classifier to classify **unseen movies**, based what the user has previously watched and **liked/not liked**.
- We could even **rank** these preferences by, e.g. computing **some distance** to the boundary.



# Manual embedding

- Using this manual embedding, we can then train a binary classifier to classify **unseen movies**, based what the user has previously watched and **liked/not liked**.
- We could even **rank** these preferences by, e.g. computing **some distance** to the boundary.



In general however, we would prefer an automated process → **How could we train a model to learn a proper embedding to use for a given task/data format?**

# The embedding problem, in language

- A typical scenario where embedding appears necessary, has to do with **language**, or as we call it in the DL community: **Natural Language Processing (NLP)**.
- Deep Learning Architectures are incapable of processing **strings or plain text in their raw form**.
- **And language is... a bit of a pain to embed, for multiple reasons...**
- **Let us see why through some examples.**



# A simple word embedding

**Definition (one-hot word embedding):**

Let us consider a dictionary of words  $V$ , with  $|V|$  distinct words.

The **one-hot word embedding** simply assigns a **one-hot vector of dimension  $|V|$**  to each word of the dictionary  $V$ .

# A simple word embedding

## Definition (**one-hot word embedding**):

Let us consider a dictionary of words  $V$ , with  $|V|$  distinct words.

The **one-hot word embedding** simply assigns a **one-hot vector of dimension  $|V|$**  to each word of the dictionary  $V$ .

- For instance, if
$$V = \{apple, durian, mango, orange, strawberry\}$$
- A possible one-hot embedding  $f$  would be
$$\begin{aligned} f(apple) &= e_1 = (1, 0, 0, 0, 0) \\ f(durian) &= e_2 = (0, 1, 0, 0, 0) \\ f(mango) &= e_3 = (0, 0, 1, 0, 0) \\ f(orange) &= e_4 = (0, 0, 0, 1, 0) \\ f(strawberry) &= e_5 = (0, 0, 0, 0, 1) \end{aligned}$$

# A simple word embedding

## Definition (**one-hot word embedding**):

Let us consider a dictionary of words  $V$ , with  $|V|$  distinct words.

The **one-hot word embedding** simply assigns a **one-hot vector of dimension  $|V|$**  to each word of the dictionary  $V$ .

Our one-hot word embedding is a function  $f: V \rightarrow OH_{\mathbb{R}^5}$ ,

With  $OH_{\mathbb{R}^5}$  being the set of all possible one-hot vectors of dimension 5.

# A simple word embedding

## Definition (**one-hot word embedding**):

Let us consider a dictionary of words  $V$ , with  $|V|$  distinct words.

The **one-hot word embedding** simply assigns a **one-hot vector of dimension  $|V|$**  to each word of the dictionary  $V$ .

Our one-hot word embedding is a function  $f: V \rightarrow OH_{\mathbb{R}^5}$ ,

With  $OH_{\mathbb{R}^5}$  being the set of all possible one-hot vectors of dimension 5.

- **Observation #1:**  $f$  is **bijective**.  
(mathematically nice to have!)



# A simple word embedding

## Definition (**one-hot word embedding**):

Let us consider a dictionary of words  $V$ , with  $|V|$  distinct words.

The **one-hot word embedding** simply assigns a **one-hot vector of dimension  $|V|$**  to each word of the dictionary  $V$ .

Our one-hot word embedding is a function  $f: V \rightarrow OH_{\mathbb{R}^5}$ ,

With  $OH_{\mathbb{R}^5}$  being the set of all possible one-hot vectors of dimension 5.

- **Observation #2:**  $f^{-1}$  is easy to find, as  $f$  is **easy to invert**. (To work your way back from a one-hot vector to a word, all you need is the index where the value 1 appears, and that is the index of the word in  $V$ ).

# A simple word embedding

## Definition (**one-hot word embedding**):

Let us consider a dictionary of words  $V$ , with  $|V|$  distinct words.

The **one-hot word embedding** simply assigns a **one-hot vector of dimension  $|V|$**  to each word of the dictionary  $V$ .

- **Observation #3:** The embeddings are nicely **orthogonal** in the **inner product space  $OH_{\mathbb{R}^5}$** .

$$\langle e_i, e_j \rangle = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{else} \end{cases}$$

- **Reminder:** the **inner product** in math is the same thing as the **dot product** in CS.

$$\langle x, y \rangle = \sum_i x_i y_i$$

# A simple word embedding

## Definition (**one-hot word embedding**):

Let us consider a dictionary of words  $V$ , with  $|V|$  distinct words.

The **one-hot word embedding** simply assigns a **one-hot vector of dimension  $|V|$**  to each word of the dictionary  $V$ .

- **Observation #3:** The embeddings are nicely **orthogonal** in the **inner product space  $OH_{\mathbb{R}^5}$** .

$$\langle e_i, e_j \rangle = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{else} \end{cases}$$

(This is also a nice property to have in mathematics, as this would mean that each embedding vector is as far as possible from each other in terms of vector distance).

# The embedding problem, in language

- A typical scenario where embedding appears necessary, has to do with **language**, or as we call it in the DL community: **Natural Language Processing (NLP)**.
- And language is... a (bit of a) pain to embed, for multiple reasons...
- **The one-hot word embedding, while simple and convenient, unfortunately suffers from many problems.**
- **Problems that have to do with the logic behind human language.**

# The embedding problem, in language

## Problem #1: Identical words can have multiple (and sometimes very different) meanings.

- In fact, this meaning is often decided by the surrounding words in the sentence, which provide context, not by the word on its own.

*I ate a **club** sandwich yesterday.  
I broke my golf **club** yesterday night.*

### Definition of *club* (Entry 1 of 2)

- 1
  - a : a heavy usually tapering staff especially of wood wielded as a weapon
  - b : a stick or bat used to hit a ball in any of various games
  - c : something resembling a club
- 2
  - a : a playing card marked with a stylized figure of a black clover
  - b **clubs** plural in form but singular or plural in construction : the suit comprising cards marked with clubs
- 3
  - a : an association of persons for some common object usually jointly supported and meeting periodically  
also : a group identified by some common characteristic  
// nations in the nuclear *club*
  - b : the meeting place of a club  
// lunch at the *club*
  - c : an association of persons participating in a plan by which they agree to make regular payments or purchases in order to secure some advantage
  - d : NIGHTCLUB
  - e : an athletic association or team
- 4 : CLUB SANDWICH

# The embedding problem, in language

**Problem #1: Identical words can have multiple (and sometimes very different) meanings.**

- Its syntactical meaning (is it a verb, a noun, an adjective?) also depends on surrounding context.

*I ate a **club** sandwich yesterday.  
I broke my golf **club** yesterday night.  
I went **clubbing** last night.*

## Definition of *club* (Entry 1 of 2)

- 1
  - a : a heavy usually tapering staff especially of wood wielded as a weapon
  - b : a stick or bat used to hit a ball in any of various games
  - c : something resembling a club
- 2
  - a : a playing card marked with a stylized figure of a black clover
  - b **clubs** *plural in form but singular or plural in construction* : the suit comprising cards marked with clubs
- 3
  - a : an association of persons for some common object usually jointly supported and meeting periodically  
*also* : a group identified by some common characteristic  
*// nations in the nuclear club*
  - b : the meeting place of a club  
*// lunch at the club*
  - c : an association of persons participating in a plan by which they agree to make regular payments or purchases in order to secure some advantage
  - d : NIGHTCLUB
  - e : an athletic association or team
- 4 : CLUB SANDWICH

# The embedding problem, in language

**Problem #1: Identical words can have multiple (and sometimes very different) meanings.**

- In fact, this meaning is often decided by the surrounding words in the sentence, which provide context, **not by the word on its own.**

*I ate a **club** sandwich yesterday.*

*I broke my golf **club** yesterday night.*

*I went **clubbing** last night.*

- And the embedding vector should intrinsically carry this difference in terms of meaning for the NN to use.
- This cannot happen if we use a naïve one-hot vectors embedding.
- Reason: two identical words with different meanings would have the same embedding vector.

# The embedding problem, in language

**Problem #1: Identical words can have multiple (and sometimes very different) meanings.**

- And do not get me started on words with different meanings based on the language they come from....!

*Coin (FR) = Corner (ENG)*

*Coin (ENG) = Piece (FR)*

*Piece (FR) = Room (ENG)*

- Or sarcasm, metaphors, etc.!

The image shows two screenshots of a French-English dictionary interface, likely from a web browser. The top screenshot shows the word 'coin' in French, with its definition as a triangular instrument or a piece of metal, and its translations in English as 'corner', 'wedge', 'place', 'spot', 'nook', and 'quoin'. The bottom screenshot shows the word 'coin' in English, with its definition as a flat piece of metal used as money, and its translations in French as 'pièce de monnaie', 'la pierre de coin', and 'la formulation'. The interface includes a search bar, a language selector (FRANÇAIS, ANGLAIS, ARABE), and a list of synonyms and translations for each word.



# The embedding problem, in language

**Problem #1: Identical words can have multiple (and sometimes very different) meanings.**

- Why is this a problem?
- This means an embedding for words **cannot be injective, let alone bijective...**
- Instead, it should be possible to embed a word  $x$  into different tensors  $x'$ , for each possible meanings of the word.

# The embedding problem, in language

**Problem #1: Identical words can have multiple (and sometimes very different) meanings.**

- And since the meaning of the word depends on **context**...
- This means that **our embedding function  $f$  should not just take a single word  $x$  as input to produce an embedding vector  $x'$  for said word  $x$ .**

# The embedding problem, in language

**Problem #1: Identical words can have multiple (and sometimes very different) meanings.**

- And since the meaning of the word depends on **context**...
- This means that **our embedding function  $f$  should not just take a single word  $x$  as input to produce an embedding vector  $x'$  for said word  $x$ .**

- This is what our one-hot embedding simply does with words.

$$f: V \rightarrow OH_{\mathbb{R}^m}$$
$$f(\textit{club}) = (0, 0, \dots, 0, 1, 0, \dots, 0) = e_k$$

*I ate a **club** sandwich yesterday.*

*I broke my golf **club** yesterday night.*

# The embedding problem, in language

**Problem #1: Identical words can have multiple (and sometimes very different) meanings.**

- And since the meaning of the word depends on **context**...
- This means that **our embedding function  $f$  should not just take a single word  $x$  as input to produce an embedding vector  $x'$  for said word  $x$ .**

- Maybe it would be preferable to have the word in question, plus some surrounding words for context instead as inputs for the embedding function.

$$g: V^n \rightarrow \mathbb{R}^m$$

$$g(\text{my, golf, club, yesterday, night}) \\ = (0, 0, \dots, 0, 1, 0, \dots, 0) = e_k$$

$$g(\text{ate, a, club, sandwich, yesterday}) \\ = (0, 0, 1, 0, \dots, 0) = e_{k'}$$

With  $k \neq k'$

*I ate a **club** sandwich yesterday.*

*I broke my golf **club** yesterday night.*

# The embedding problem, in language

## Problem #2: Two different words could have really close meanings.

- Our one-hot word embedding typically verifies

$$\begin{aligned}\langle e_{kitten}, e_{kitty} \rangle &= 0 \\ \langle e_{kitten}, e_{kitchen} \rangle &= 0\end{aligned}$$

- **Observation #3:** The embeddings are nicely **orthogonal** in the **inner product space**  $OH_{\mathbb{R}^{|V|}}$ .

$$\langle e_i, e_j \rangle = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{else} \end{cases}$$

- Reminder, the **inner product** in math is the same thing as the **dot product** in CS.

$$\langle x, y \rangle = \sum_i x_i y_i$$

# The embedding problem, in language

**Problem #2: Two different words could have really close meanings.**

- Our one-hot word embedding typically verifies

$$\begin{aligned}\langle e_{kitten}, e_{kitty} \rangle &= 0 \\ \langle e_{kitten}, e_{kitchen} \rangle &= 0\end{aligned}$$

- In practice, we would prefer to have something like this

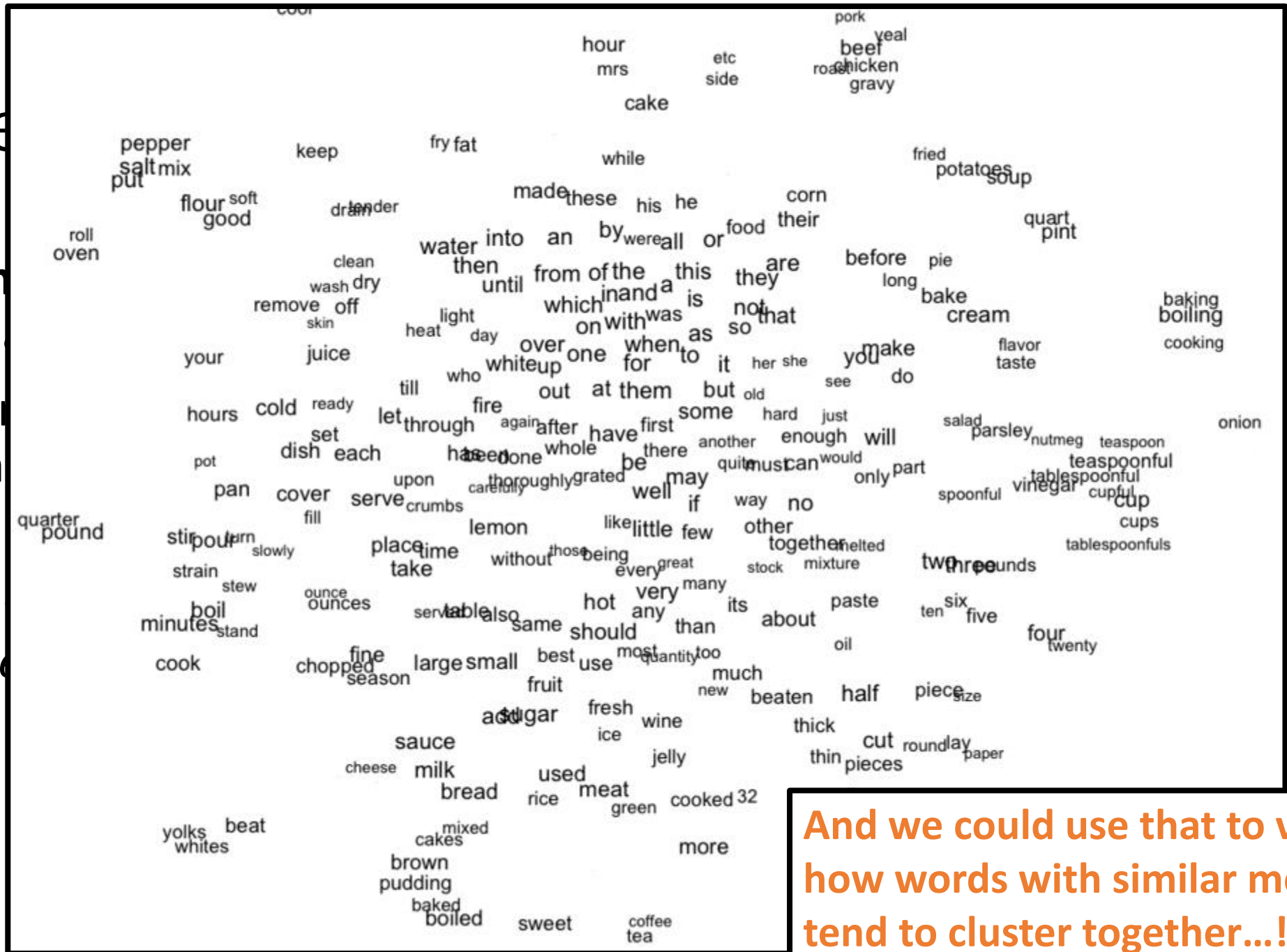
$$\begin{aligned}\langle e_{kitten}, e_{kitty} \rangle &\approx 1 \\ \langle e_{kitten}, e_{kitchen} \rangle &\approx 0\end{aligned}$$

As this would help understand that some words have some sort of **proximity** in terms of meaning.

Restricted

# Problem could h

- Our on typical



p have

that

And we could use that to visualize how words with similar meaning tend to cluster together...!

# The embedding problem, in language

**Problem #3: Two different words could have really close meanings, but their embeddings may or may not need to be... and this decision could be task-specific.**

- If the embedding is used for general language (non-medical), then we are probably fine with  $\langle e_{covid}, e_{cold} \rangle \approx 1$

As the only important information from these words is that they are both respiratory diseases.

- However, if the embedding will be used in a very specialized medical context, then it is probably better to have less similarity between the two words, i.e.

$$\langle e_{covid}, e_{cold} \rangle \in [0.5, 0.8],$$

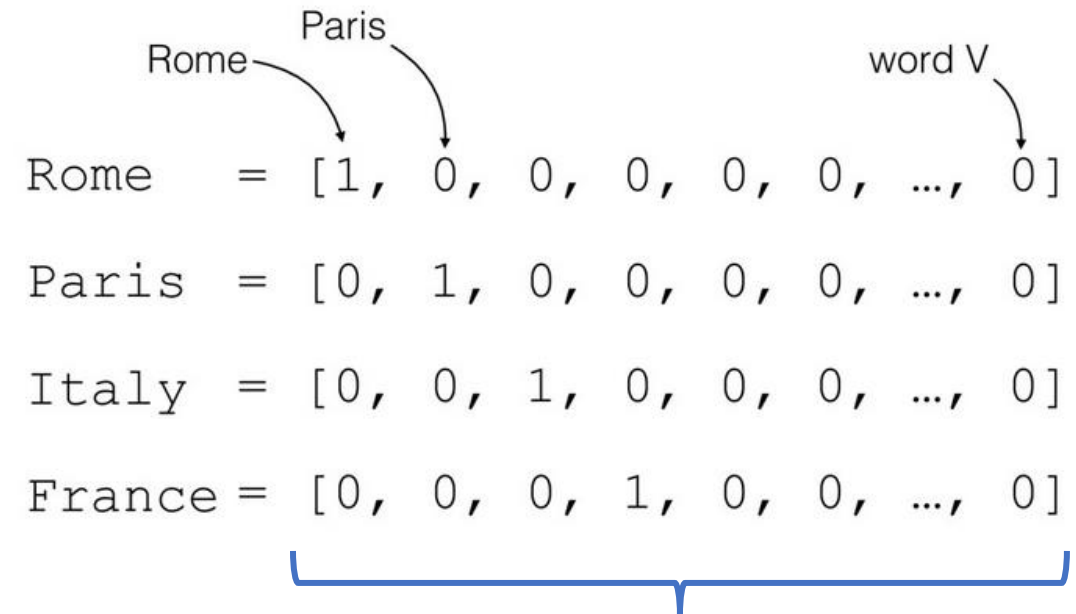
- That is because, covid and cold are two different diseases, maybe sharing some similar symptoms, but to be treated very differently.



# The embedding problem, in language

**Problem #4: Using  $O H_{\mathbb{R}^{|V|}}$  to represent embeddings is seriously problematic, if we consider the issue of memory space...**

- Languages contain millions (trillions if we include the typos, conjugations, acronyms, etc?) of possible words...
- We need a representation with lower dimensionality!



What is the size of this vector?!  
How many zeros in there?!

# The embedding problem, in language

- We cannot really decide the embedding manually... Neither can we come up with an expert system for word embedding.
- **And what do we do, when something like that happens?**

# The embedding problem, in language

- We cannot really decide the embedding manually... Neither can we come up with an expert system for word embedding.
- **And what do we do, when something like that happens?**
- **AS USUAL, LET AN AI FIGURE IT OUT FOR US!**

Two possible approaches:

- **Frequency-based embeddings**
- **Prediction-based embeddings**

# The embedding problem, in language

- We cannot really decide the embedding manually... Neither can we come up with an expert system for word embedding.
- **And what do we do, when something like that happens?**
- **AS USUAL, LET AN AI FIGURE IT OUT FOR US!**

Two possible approaches:

- Frequency-based embeddings (nowadays, we do not really use those anymore, but feel free to have a look at TF-IDF and consorts)
- **Prediction-based embeddings  
(Let us get started with Continuous Bag of Words and Skip-Gram)**

# How to generate an embedding with AI?

- Train a Neural Network for some meaningful (whatever actually!) task over a dataset of words and sentences.
- **Use one of the hidden layers for feature representation.**
  - Take a word  $w$  and map it onto its corresponding one-hot vector  $e_i(w)$ , as described earlier.
  - Input the corresponding one-hot vector  $e_i(w)$  into the trained NN and obtain the feature vector  $f(w)$  from the NN.
  - Then, usually,  $\langle f(w_i), f(w_j) \rangle \neq 0$ , even if  $w_i \neq w_j$ .
  - And similar words will have similar embeddings and a positive inner product, compared to unrelated pairs of words.

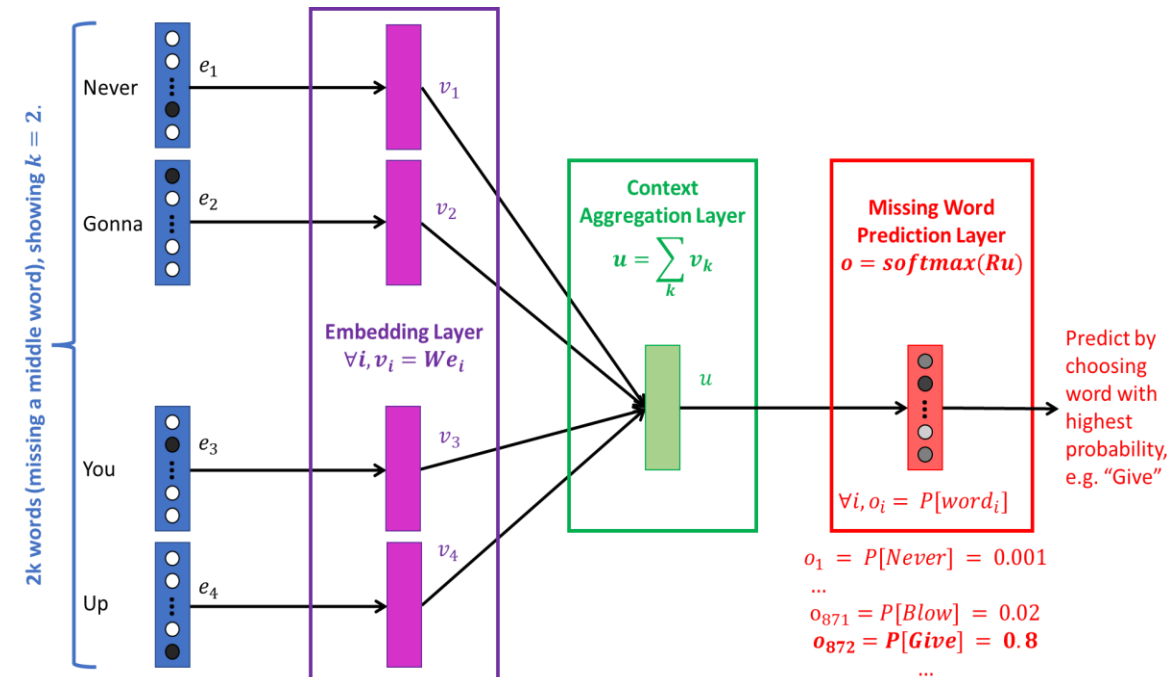
# Continuous Bag of Words (CBoW)

## Definition (CBoW):

**CBoW** (introduced in [Mikolov2013]) is a first feature representation model, which can be used for word embedding.

Using a large text corpus for training, it attempts to learn how to predict the word in the middle (with index  $k$ ) of a sequence of  $2k + 1$  words.

Here,  $k$  is called the **span** of the language model.



2k words (missing a middle word), showing  $k = 2$ .

Never



$e_1$

Gonna



$e_2$

You



$e_3$

Up



$e_4$

Embedding Layer  
 $\forall i, v_i = We_i$

$v_1$

$v_2$

$v_3$

$v_4$

Context  
Aggregation Layer

$$u = \sum_k v_k$$

$u$

Missing Word  
Prediction Layer  
 $o = \text{softmax}(Ru)$

$$\forall i, o_i = P[\text{word}_i]$$

Predict by  
choosing  
word with  
highest  
probability,  
e.g. "Give"

Restricted

# Continuous Bag of Words (CBoW)

- Consider the text: “**I have a dream that** one day this nation will rise up and live out the true meaning of its creed: We hold these truths to be self-evident, that all men are created equal. I have a dream that one day on the red hills of Georgia, the sons of former slaves and the sons of former slave owners will be able to sit down together at the table of brotherhood.”

0. We will use a sliding window, with e.g. size  $k = 2$ , to generate pairs of  $(x, y)$  values to train our CBoW on.

$$x_1 = (I, have, dream, that), \\ y_1 = a$$



# Continuous Bag of Words (CBoW)

- Consider the text: “I **have a dream that one** day this nation will rise up and live out the true meaning of its creed: We hold these truths to be self-evident, that all men are created equal. I have a dream that one day on the red hills of Georgia, the sons of former slaves and the sons of former slave owners will be able to sit down together at the table of brotherhood.”

0. We will use a sliding window, with e.g. size  $k = 2$ , to generate pairs of  $(x, y)$  values to train our CBoW on.

$$\begin{aligned}x_1 &= (I, have, dream, that), \\y_1 &= a \\x_2 &= (have, a, that, one), \\y_2 &= dream\end{aligned}$$

# Continuous Bag of Words (CBoW)

- Consider the text: “I have a dream that one day this nation will rise up and live out the true meaning of its creed: We hold these truths to be self-evident, that all men are created equal. I have a dream that one day on the red hills of Georgia, the sons of former slaves and **the sons of former slave** owners will be able to sit down together at the table of brotherhood.”

0. We will use a sliding window, with e.g. size  $k = 2$ , to generate pairs of  $(x, y)$  values to train our CBoW on.

$x_1 = (I, have, dream, that),$

$y_1 = a$

$x_2 = (have, a, that, one),$

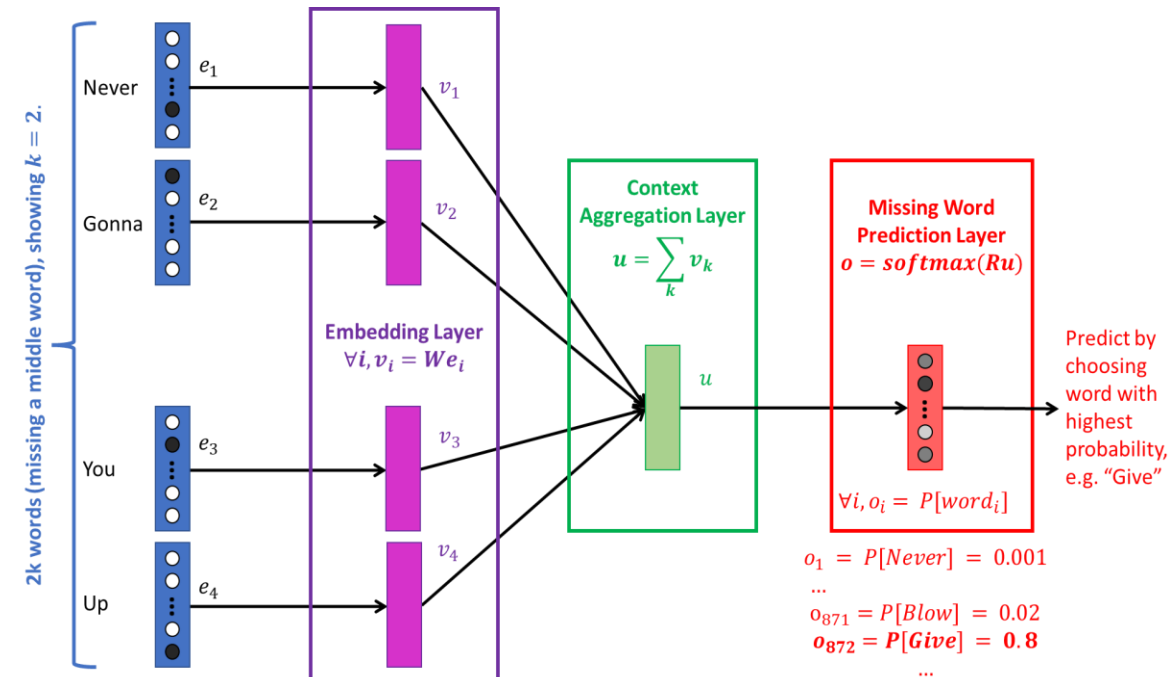
$y_2 = dream$

$x_{72} = (the, sons, former, slave),$

$y_{72} = of$

# Continuous Bag of Words (CBoW)

1. Then, build a simple NN, which takes the  $2k$  non-middle words, as one-hot embeddings  $e_i \in \mathbb{R}^{|V|}$ .



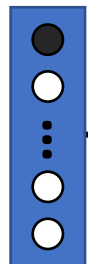
2k words (missing a middle word), showing  $k = 2$ .

Never



$e_1$

Gonna



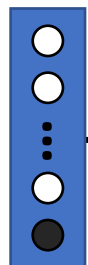
$e_2$

You



$e_3$

Up



$e_4$

Embedding Layer  
 $\forall i, v_i = W e_i$

$v_1$

$v_2$

$v_3$

$v_4$

Context  
Aggregation Layer

$$u = \sum_k v_k$$

$u$

Missing Word  
Prediction Layer  
 $o = \text{softmax}(Ru)$

$$\forall i, o_i = P[\text{word}_i]$$

$$o_1 = P[\text{Never}] = 0.001$$

...

$$o_{871} = P[\text{Blow}] = 0.02$$

$$o_{872} = P[\text{Give}] = 0.8$$

...

Predict by  
choosing  
word with  
highest  
probability,  
e.g. "Give"

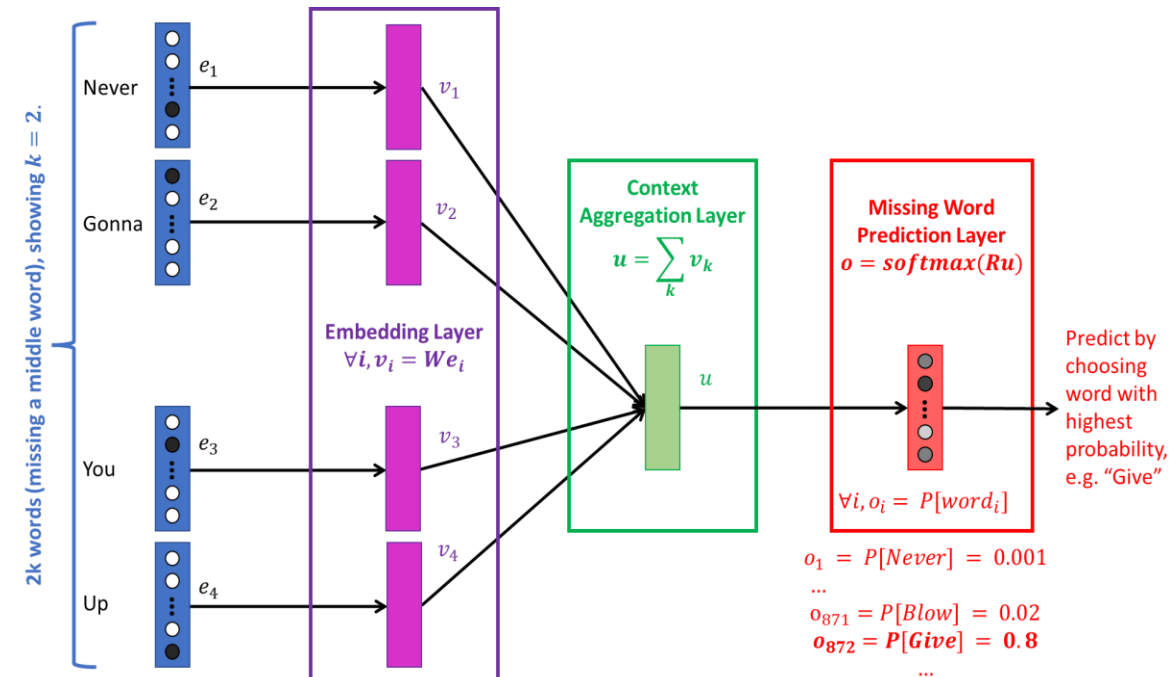
Restricted

1. Start by using one-hot encoding on your  $2k$  input words  
 $\forall i, e_i \in \mathbb{R}^{|V|}$ , with  $|V|$  being the dictionary size.

Restricted

# Continuous Bag of Words (CBoW)

1. Then, build a simple NN, which takes the  $2k$  non-middle words, as one-hot embeddings  $e_i \in \mathbb{R}^{|V|}$ .
2. Add one **embedding layer** with matrix  $W \in \mathbb{R}^{D \times |V|}$ . Here,  $D$  denotes the size of the new word embedding and is often chosen such that  $D \ll |V|$ .



2k words (missing a middle word), showing  $k = 2$ .

Never



$e_1$

Gonna



$e_2$

You



$e_3$

Up



$e_4$

Embedding Layer  
 $\forall i, v_i = We_i$

$v_1$

$v_2$

$v_3$

$v_4$

Context  
Aggregation Layer

$$u = \sum_k v_k$$

$u$

Missing Word  
Prediction Layer  
 $o = \text{softmax}(Ru)$

$$\forall i, o_i = P[\text{word}_i]$$

$$o_1 = P[\text{Never}] = 0.001$$

...

$$o_{871} = P[\text{Blow}] = 0.02$$

$$o_{872} = P[\text{Give}] = 0.8$$

...

Predict by  
choosing  
word with  
highest  
probability,  
e.g. "Give"

Restricted

Restricted  
2. Apply the same dense/linear transformation to each  $e_i$ .  
We have  $\forall i, v_i = We_i$ , which is implemented by an embedding  
layer in PyTorch, with trainable parameter  $W \in \mathbb{R}^{|V| \times D}$ .  
This produces lower dimensionality vector representations for  
words, i.e.  $\forall i, v_i \in \mathbb{R}^D$ , with  $D \ll |V|$ .

# Continuous Bag of Words (CBoW)

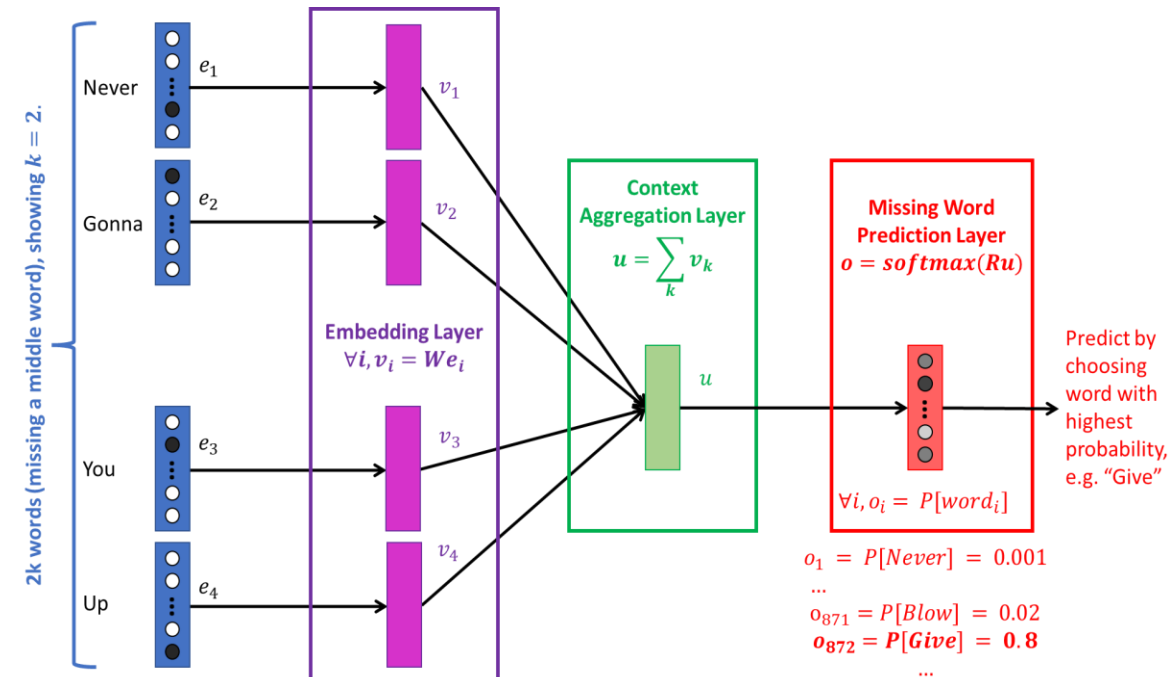
## Definition (the **Embedding Layer**):

On the first matrix multiplication, we use an **Embedding Layer**, not a **Linear one**.

In this layer, each one-hot vector of size  $|V|$  is multiplied by the same matrix  $W \in \mathbb{R}^{D \times |V|}$  to produce an embedding of size  $D$ .

This is important as this is our embedding function and we want it to be applied identically to each word in the inputs!

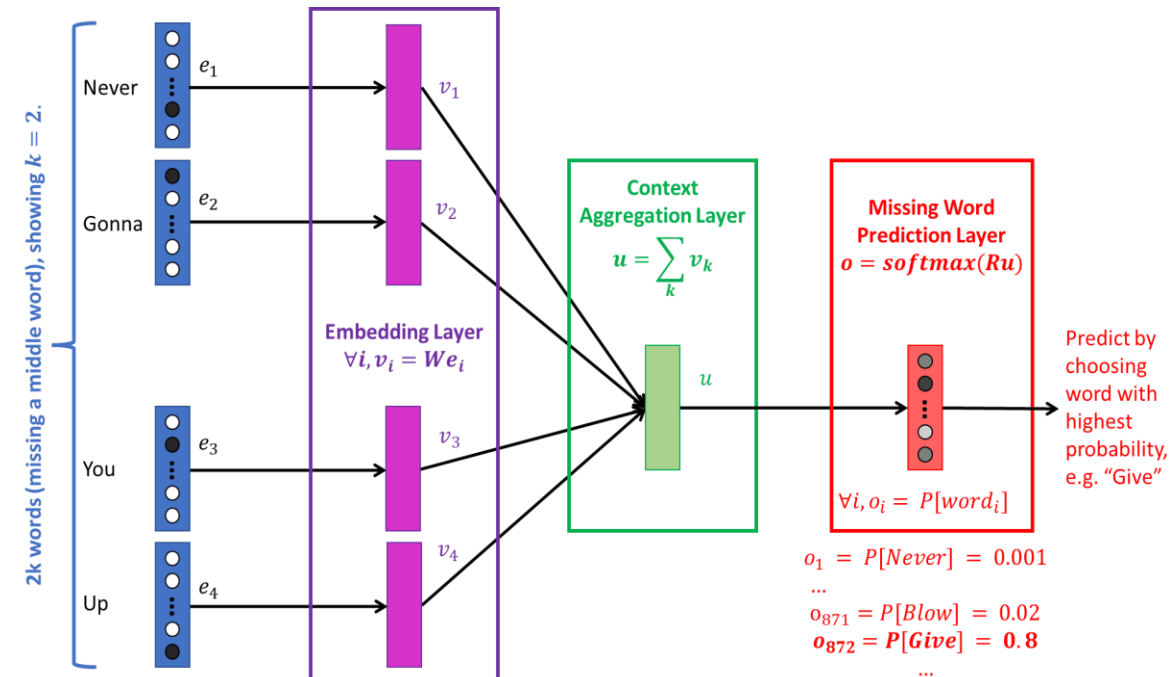
$W$  is  $D \times |V|$ , not  $kD \times 2k|V|$ !



# Continuous Bag of Words (CBoW)

1. Then, build a simple NN, which takes the  $2k$  non-middle words, as one-hot embeddings  $e_i \in \mathbb{R}^{|V|}$ .
2. Add one **embedding layer** with matrix  $W \in \mathbb{R}^{D \times |V|}$ . Here,  $D$  denotes the size of the new word embedding and is often chosen such that  $D \ll |V|$ .

**Note:** after training,  $v_k = W e_k$  will be used as the new word embedding to replace  $e_k$ !





# Continuous Bag of Words (CBoW)

1. Then, build a simple NN, which takes the 2k non-middle words, as one-hot embeddings  $e_i \in \mathbb{R}^{|V|}$ .
2. Add one **embedding layer** with matrix  $W \in \mathbb{R}^{D \times |V|}$ . Here,  $D$  denotes the size of the new word embedding and is often chosen such that  $D \ll |V|$ .

**Note:** after training,  $v_k = W e_k$  will be used as the new word embedding to replace  $e_k$ !

**Note:** having  $D \ll |V|$  is important, as it addresses the memory space issue (Problem #4) we discussed earlier.

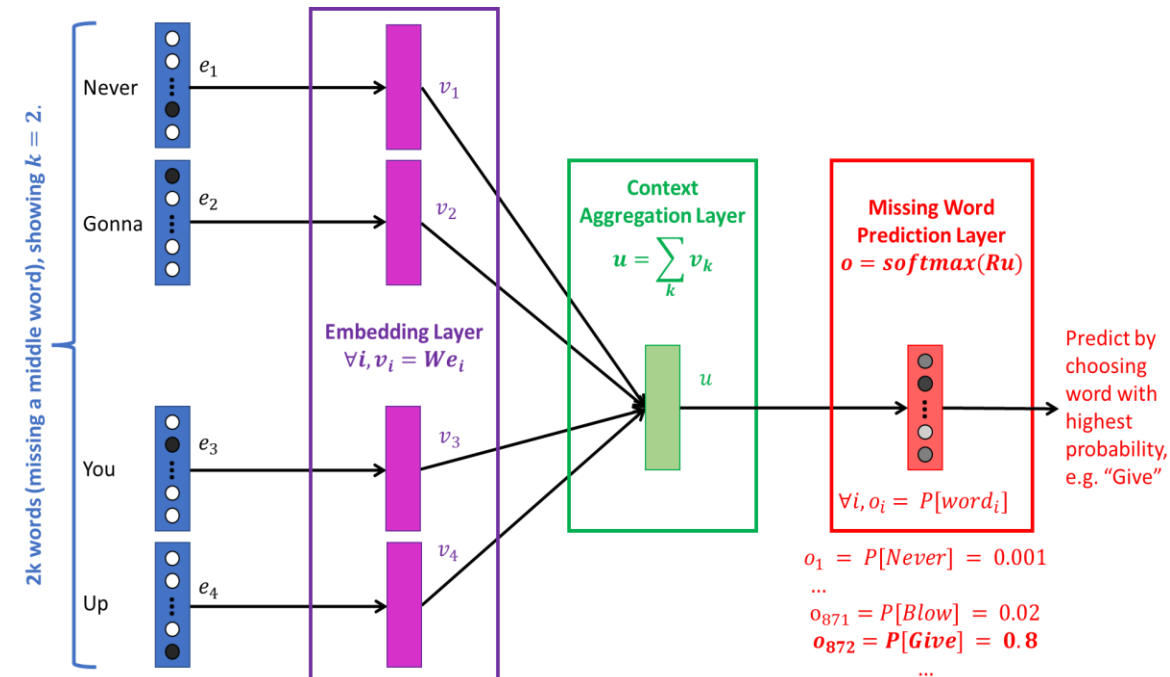
**(Problem #4: Using  $OH_{\mathbb{R}^{|V|}}$  to represent embeddings is seriously problematic, if we consider the issue of memory space...)**

# Continuous Bag of Words (CBoW)

3. The second to last layer is simply summing all the **context** new embedding together.

$$u = \sum_k v_k$$

This is not trainable and gives an **“average” context carried by the surrounding words**, which is used to predict the missing word in the final layer.



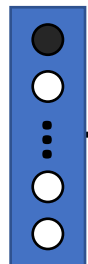
2k words (missing a middle word), showing  $k = 2$ .

Never



$e_1$

Gonna



$e_2$

You



$e_3$

Up



$e_4$

Embedding Layer  
 $\forall i, v_i = We_i$

$v_1$

$v_2$

$v_3$

$v_4$

Context  
Aggregation Layer

$$u = \sum_k v_k$$

$u$

Missing Word  
Prediction Layer  
 $o = \text{softmax}(Ru)$

$$\forall i, o_i = P[\text{word}_i]$$

$$o_1 = P[\text{Never}] = 0.001$$

...

$$o_{871} = P[\text{Blow}] = 0.02$$

$$o_{872} = P[\text{Give}] = 0.8$$

...

Predict by  
choosing  
word with  
highest  
probability,  
e.g. "Give"

Restricted

Restricted

3. The aggregation layer combines the contextual meaning vectors  $v_i$  together by simply summing them, i.e  $u = \sum_k v_k$ . This non-trainable layer produces a vector  $u \in \mathbb{R}^D$ , with  $D \ll |V|$ , which roughly encapsulates the meaning of the entire incomplete sentence of  $2k$  input words.

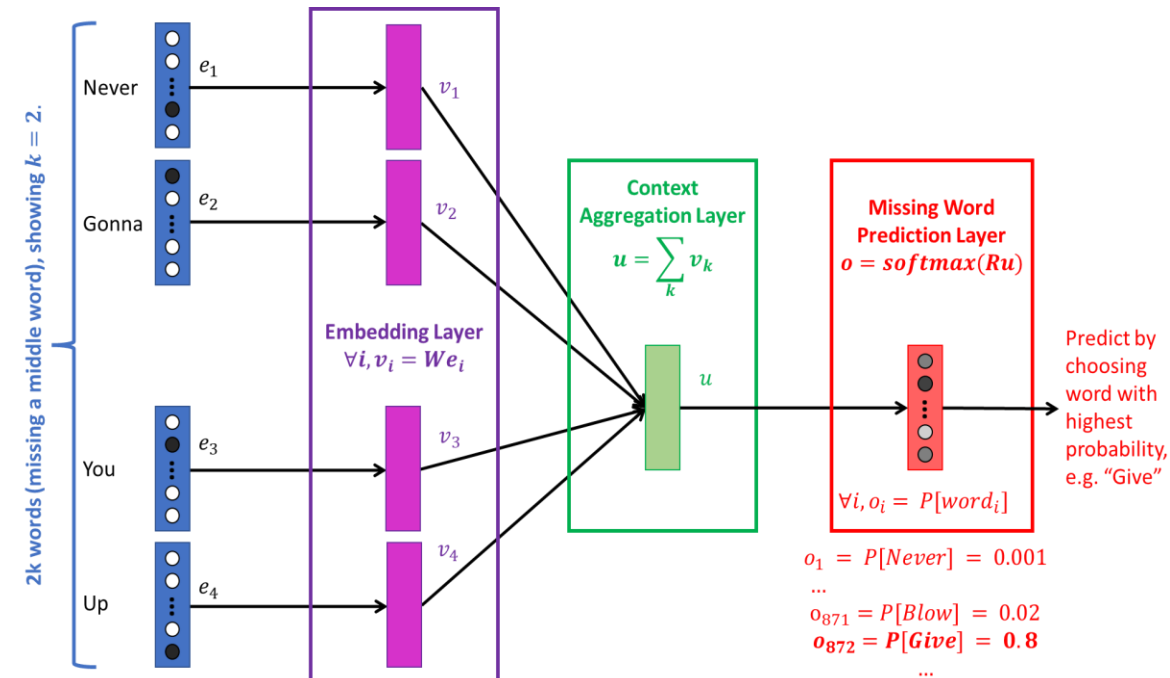
# Continuous Bag of Words (CBoW)

4. The final layer is trainable and produces an output  $o$ :

$$o = \text{softmax}(Ru)$$

With  $R \in \mathbb{R}^{|V| \times D}$  so that  $o \in \mathbb{R}^{|V|}$ .

The output  $o$  then gives probabilities over which word should be predicted.



2k words (missing a middle word), showing  $k = 2$ .

Never



$e_1$

Gonna



$e_2$

You



$e_3$

Up



$e_4$

Embedding Layer  
 $\forall i, v_i = We_i$

$v_1$

$v_2$

$v_3$

$v_4$

Context  
Aggregation Layer

$$u = \sum_k v_k$$

$u$

Missing Word  
Prediction Layer  
 $o = \text{softmax}(Ru)$

$$\forall i, o_i = P[\text{word}_i]$$

$$o_1 = P[\text{Never}] = 0.001$$

...

$$o_{871} = P[\text{Blow}] = 0.02$$

$$o_{872} = P[\text{Give}] = 0.8$$

...

Predict by  
choosing  
word with  
highest  
probability,  
e.g. "Give"

Restricted

4. Use a dense/linear layer, with trainable parameter  $R \in \mathbb{R}^{D \times |V|}$ , to predict the missing word, i.e.  $o = \text{softmax}(Ru)$ .

The output  $o \in \mathbb{R}^{|V|}$  contains the probability of each word  $i$  being the missing word in the incomplete input sentence, i.e.

$$\forall i, o_i = P[\text{word}_i]$$

Restricted

# Continuous Bag of Words (CBoW)

5. During the training, we browse through all the pairs  $(x, y)$  we generated on Step 0.

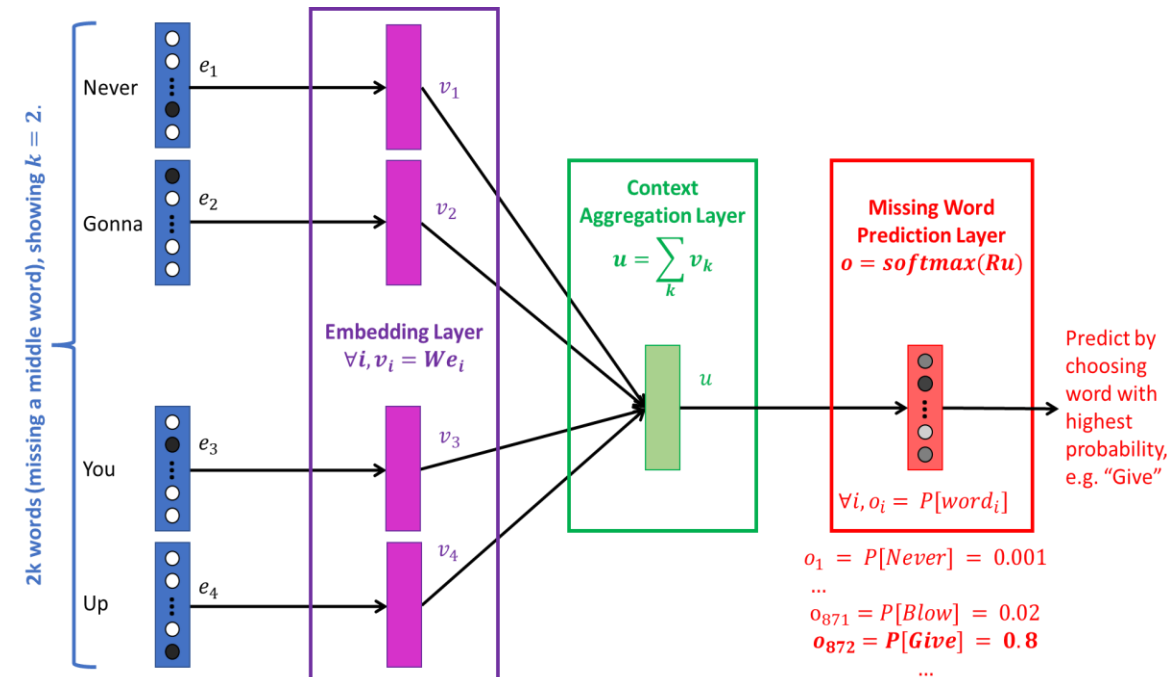
**Classification loss:** advisable to use a negative logarithm as the loss function, i.e.

$$L_t = -\log(o_t = e_t | e_{-t})$$

With

$$e_{-t} = (e_{t-k}, \dots, e_{t-1}, e_{t+1}, \dots, e_{t+k})$$

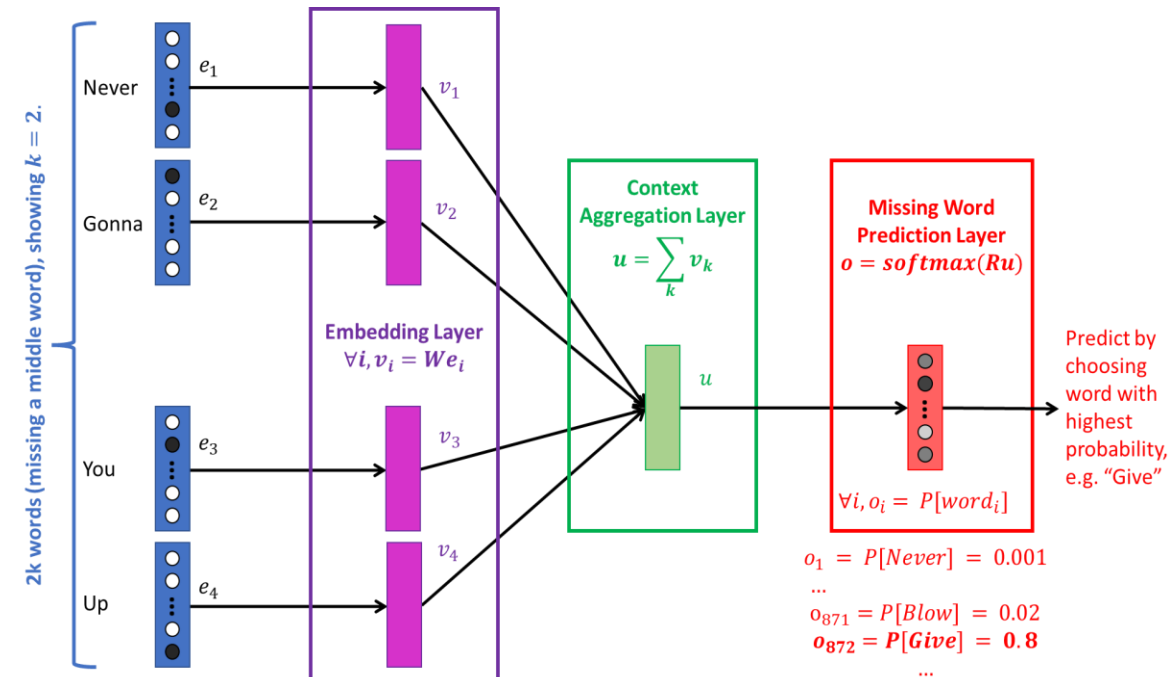
And sum over all the pairs  $(x, y)$ .



# Continuous Bag of Words (CBoW)

## Important Notes

- There are variations on this, as explained in [Mikilov2014].
- For instance, predict the word at the end of the context, instead of the middle.
- We could also remove non-meaningful words (“a”, “an”, “the”, etc.) during Step 1 to help the word embedding task not get lost in meaningless context.



# Time for a quick demo of CBoW

Notebooks 1. and 1bis.



# Why does that work?

## Definition (the **distributional hypothesis of linguistics**):

- We may rely on a fundamental linguistic assumption  
→ **Words often appearing in similar contexts/sentences tend to be related to each other.**
- This is a central hypothesis and is commonly referred to as the **distributional hypothesis of linguistics.**

A bottle of **tezgüino** is on the table.

Everyone likes **tezgüino**.

**Tezgüino** makes you drunk.

We make **tezgüino** out of corn.

Can you understand what **tezgüino** means ?

# Why does that work?

## Definition (the **distributional hypothesis of linguistics**):

- We may rely on a fundamental linguistic assumption  
→ **Words often appearing in similar contexts/sentences tend to be related to each other.**
- This is a central hypothesis and is commonly referred to as the **distributional hypothesis of linguistics**.

A bottle of **tezgüino** is on the table.

Everyone likes **tezgüino**.

**Tezgüino** makes you drunk.

We make **tezgüino** out of corn.

Can you understand what **tezgüino** means ?



# Why does that work?

## Definition (the **distributional hypothesis of linguistics – full version**):

- Similarity in linguistics does not simply mean having similar orthographic representations.
- Instead, we may rely on a fundamental linguistic assumption  
→ **Words appearing in similar contexts are related to each other.**
- This is a central hypothesis and is commonly referred to as the **distributional hypothesis of linguistics**.
- CBoW is therefore a technique, which aims at defining meaning for words (in the form of an embedding vector), by using this hypothesis, therefore establishing connections between words in texts.

# Conclusion (W8S1)

We have seen a few approaches to embeddings.

- Ideally, they should be bijective, but impossible for NLP.
- Manually embedding is often better but often impossible, typically in NLP.
- Train an AI to figure out embeddings?
  - CBoW
  - (SkipGram, next time)

A few more problems are still open at the moment for these word embeddings.

- Can we improve these embeddings and use more advanced techniques?
- How to deal with out of vocabulary words? (If we have never seen the word, how do we represent it as a one-hot vector to begin with?)

# Learn more about these topics

Out of class, for those of you who are curious

- [Mikolov2013] **Mikolov** et al., “Efficient Estimation of Word Representations in Vector Space”, 2013.  
<https://arxiv.org/abs/1301.3781>
- [Mikilov2014] **Mikolov** et al., “Distributed Representations of Words and Phrases and their Compositionality”, 2014  
<https://arxiv.org/abs/1310.4546>

# Learn more about these topics

Tracking important names (Track their works and follow them on Scholar, Twitter, or whatever works for you!)

- **Tomas Mikolov**: Research Scientist at Czech Institute of Informatics, Robotics and Cybernetics.  
**Former Researcher at Facebook and Google Brain.**  
<https://scholar.google.com.sg/citations?user=oBu8kMMAAAAJ&hl=en>  
<https://www.ciirc.cvut.cz/svetove-uznavany-expert-tomas-mikolov-prichazi-z-facebook-ai-do-ciirc-cvut-zameri-se-na-vyvoj-silne-umele-intelligence/>