الجامعة السورية الخاصة
SYRIAN PRIVATE UNIVERSITY

Topic 1

الذكاء الصنعي العملي

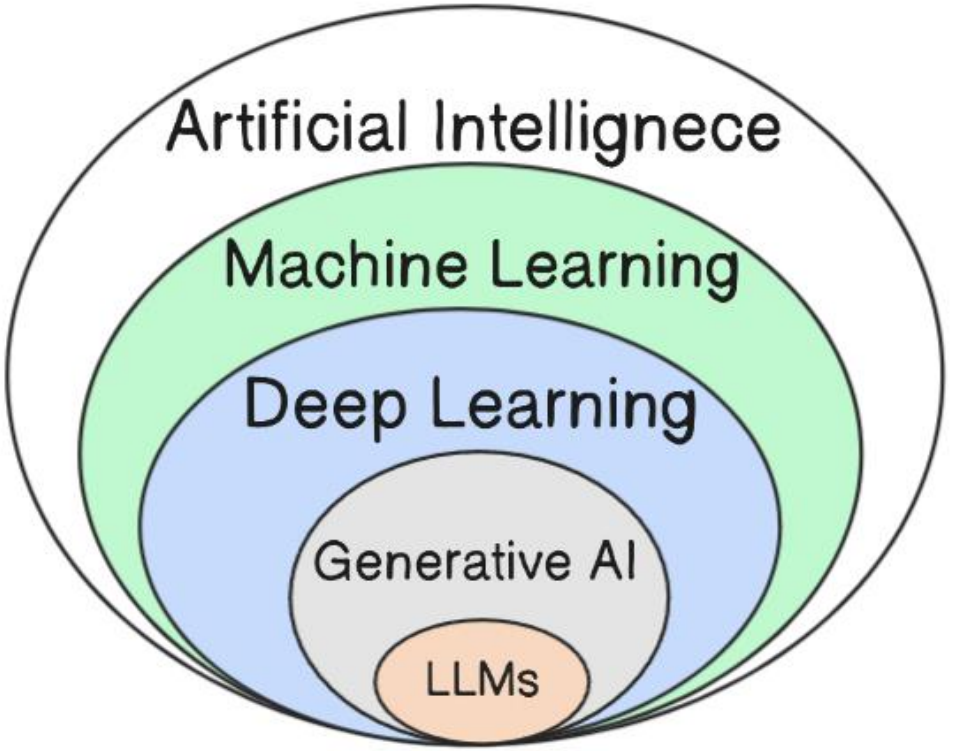
Introduction to Generative AI & LLMs
Word Embeddings

د. رياض سنبل

What is Generative AI?

- Generative AI is a branch of artificial intelligence focused on **creating new content** from learned patterns in data. Generative AI describes a category of capabilities within AI that create **original content**.

- It includes models that generate:
  - **Text** (ChatGPT, Claude, Bard)
  - **Images** (DALL-E, Stable Diffusion)
  - **Music & Audio** (Jukebox, VALL-E)
  - **Code** (GitHub Copilot, Code Llama)



Evolution of AI & NLP

- Early AI models (rule-based systems) → Statistical NLP → Deep Learning-based NLP
- **Major breakthroughs:**
  - 2013: Word Embeddings (Word2Vec, GloVe, FastText) <= Today
  - 2017: Transformers (Attention Is All You Need)
  - 2018: BERT (Bidirectional Transformers)
  - 2020–Present: GPT-3, GPT-4, LLaMA, Claude, Mistral

Language Models

- LMs model the probability distribution of token sequences in the language.
  - Word sequences, if words are the tokens
- Can be used to:
  - Compute the probability of a given token sequence
  - Generate sequences from the distribution of the language

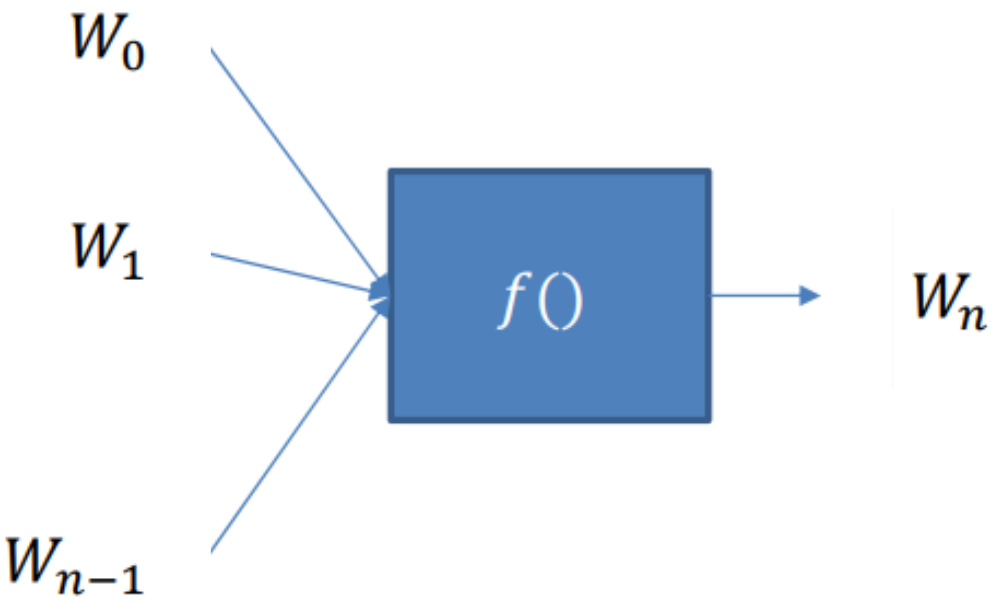
$$P(w_1\ w_2\ w_3\ w_4\ ....) = P(w_1)\ P(w_2|w_1)\ P(w_3|w_1\ w_2)\ P(w_4|w_1\ w_2\ w_3)...$$

- The actual target is to model the probabilities of entire word sequences
- However, we typically use Bayes rule to compute this incrementally
  - Language models generally perform next symbol prediction

Language modelling - Next Symbol Prediction

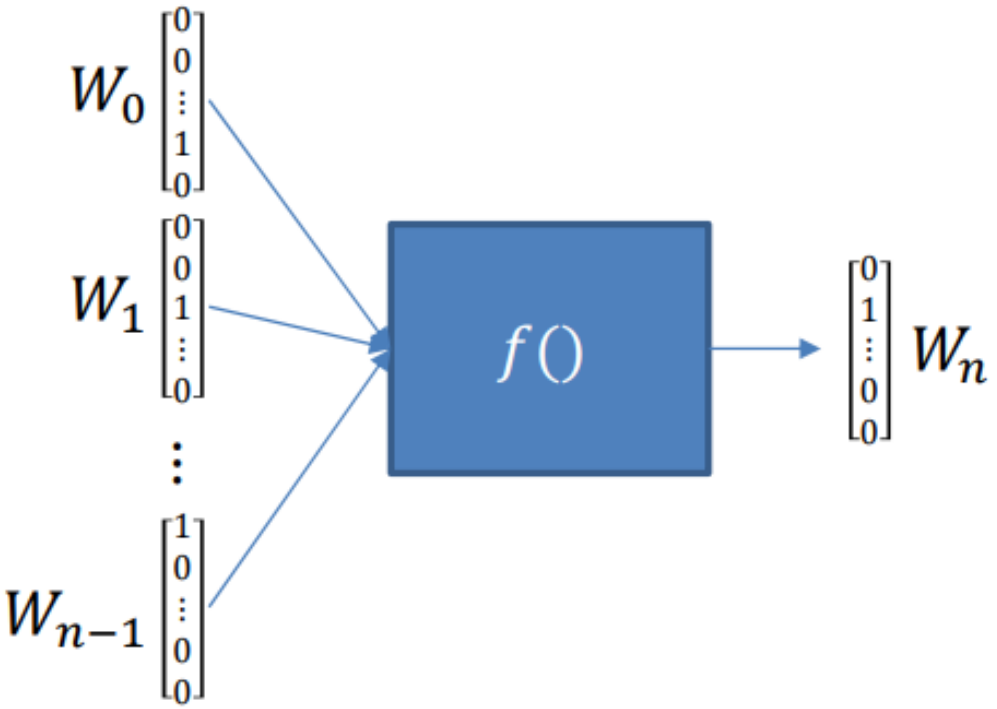
Problem: Given a sequence of words (or characters) predict the next one

But How to Represent words?



Language modelling - Next Symbol Prediction

Problem: Given a sequence of words (or characters) predict the next one



Represent words as one-hot vectors

- One challenge is the **size of the representation**.
- Possible Solutions:
  - Its common to **discard infrequent words** occurring less than e.g. 5 times.
  - one can also use the **feature selection methods**: mutual information and chi-squared tests.
  - **Preprocessing** steps: stemming.
- What should we **include** in this dictionary:
  - Bag-of-words
  - OR: Bag-of-sences, or bag-of-characters, etc.

Represent words as one-hot vectors

- **Word order is lost**, the sentence meaning is weakened:
  - The mat sat on the cat vs The cat sat on the mat.
  - word order is important, especially the order of nearby words.
- Possible Solutions:
  - **N-grams capture this**, by modeling tuples of consecutive words.
  - Typically, its advantages to use multiple n-gram features in machine learning models with text, e.g. unigrams + bigrams (2-grams) + trigrams (3-grams).

Notice how even these short n-grams "make sense" as linguistic units. For the other sentence we would have different features:

Sentence: The mat sat on the cat

2-grams: the-mat, mat-sat, sat-on, on-the, the-cat

We can go still further and construct 3-grams, Which capture still more of the meaning:

Sentence: The mat sat on the cat

3-grams: the-mat-sat, mat-sat-on, sat-on-the, on-the-cat

Represent words as one-hot vectors

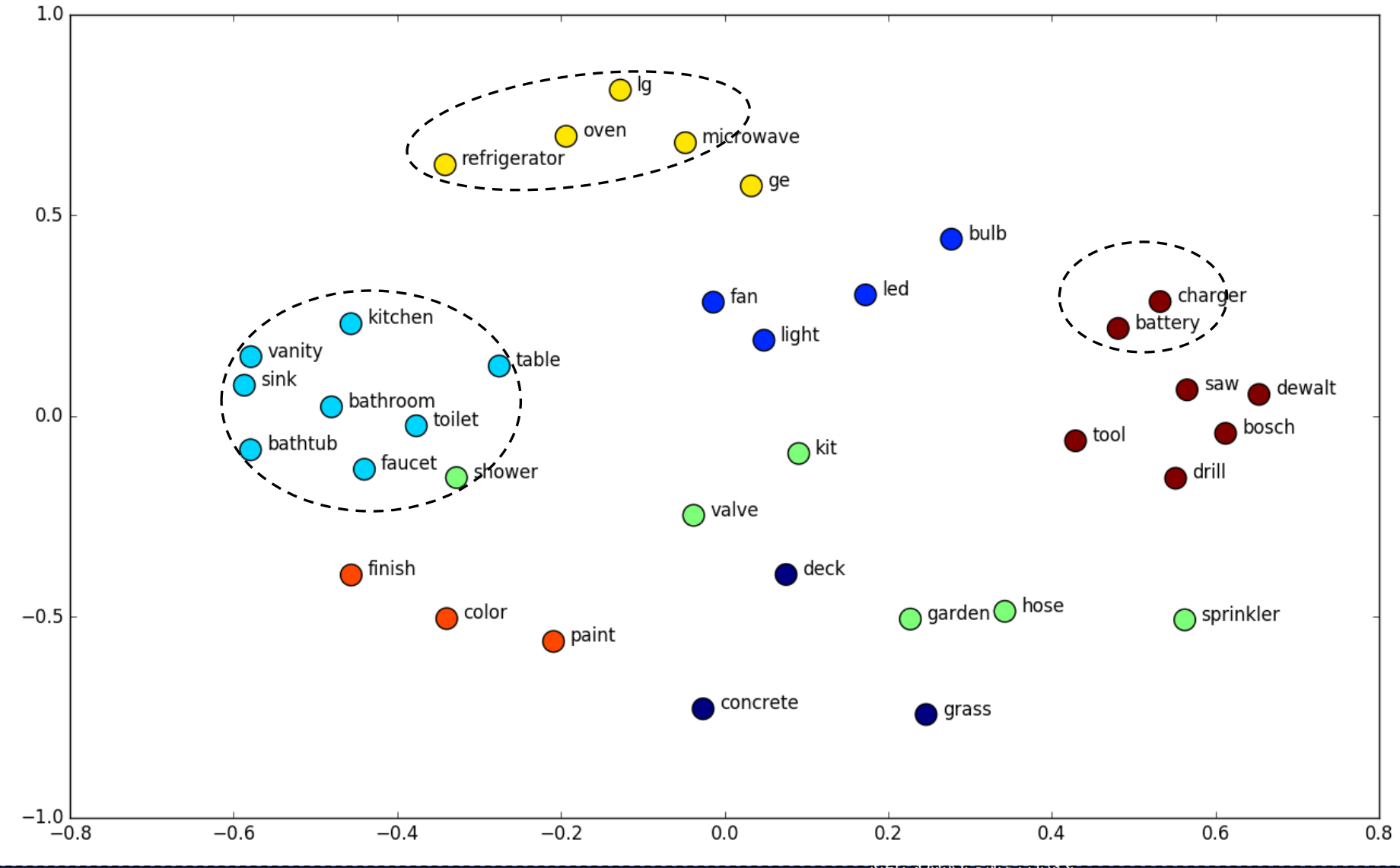
OR… Representing words as **low-dimensional dense vectors**

Word Embedding

Vector Embedding of Words

- A word is represented as a vector.
- Word embeddings depend on a notion of ***word similarity***.
- A very useful definition is paradigmatic similarity:
  - ***Similar words*** occur in ***similar contexts.*** They are ***exchangeable.***
  - Yesterday { POTUS / The President / Trump } called a press conference.
  - "POTUS: President of the United States."

The “Good” Embedding?



Good Embedding

≈

Word similarity

$$\cos(w_x, w_y) = \frac{w_x \cdot w_y}{||w_x||\ ||w_y||}$$

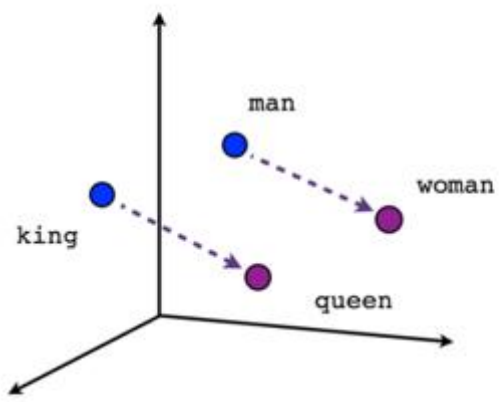
The “Good” Embedding?

Good Embedding

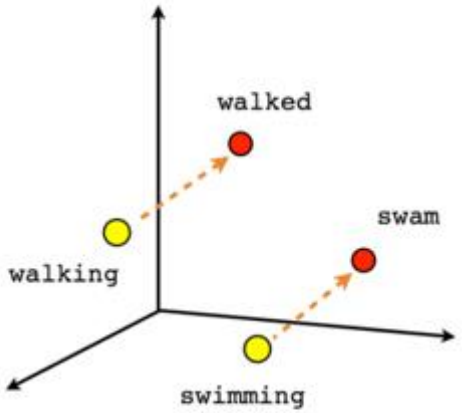
≈

Word analogy

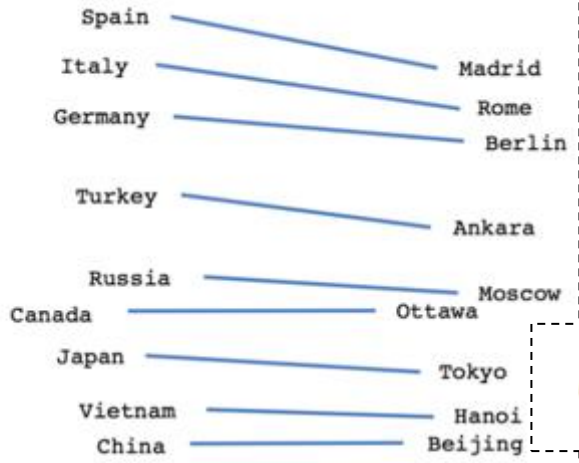
$$b^* = \underset{b'}{\text{argmax}}(\cos(b', a^* - a + b)),$$



Male-Female



Verb tense



Country-Capital

- vector[Queen] ≈ vector[King] - vector[Man] + vector[Woman]
- vector[Paris] ≈ vector[France] - vector[Italy] + vector[Rome]
  - This can be interpreted as “France is to Paris as Italy is to Rome”.

Word Embedding

- An embedding maps each word to a point in a much smaller m-dim space (e.g. 300 values)
- Two approaches:
  - **Learn the embedding jointly with your main task:**
    - An embedding layer with m hidden nodes to map word IDs to an m-dim vector.
    - Add your hidden layer and output layer, learn weights end-to-end with SGD.
  - **Use pre-trained embedding:**
    - Usually trained on another, much bigger dataset.
    - Freeze embedding weights to produce simple word embedding.

Training Embedding Layer

- Input layer use fixed length document (e.g. 100 nodes for 100 word IDs).
  - Pad with 0's is doc is shorter.
- Add an embedding layer to learn embeddings:
  - (1) Represent every word as an n-dim one hot encoding BOW.
  - (2) Learn an m-dim embedding using m-hidden nodes.
    - Learn weight matrix W(n*m) to map one hot encoded word to embedding.
  - (3) Use Linear activation function => $X_{embed} = W. X_{orig}$
- Add a layer to map word embedding to desired output.
- Learn all weights from labeled data.

Pre-trained embeddings

- More data => better embeddings BUT also **more labels**!
- **Solution**: self-supervised learning
  - Given a word, predict the surrounding words.
- Most common approaches:
  - **Word2vec:** learn neural embedding for word based on surrounding words.
  - **GloVe (Global Vector):** Count co-occurences of words in matrix.
  - **FastText:** learns embedding for character n-gram
  - Language models: learn context-dependenct embedding.
    - **BERT**, **ELMO**, **GPT3**

Skip-grams

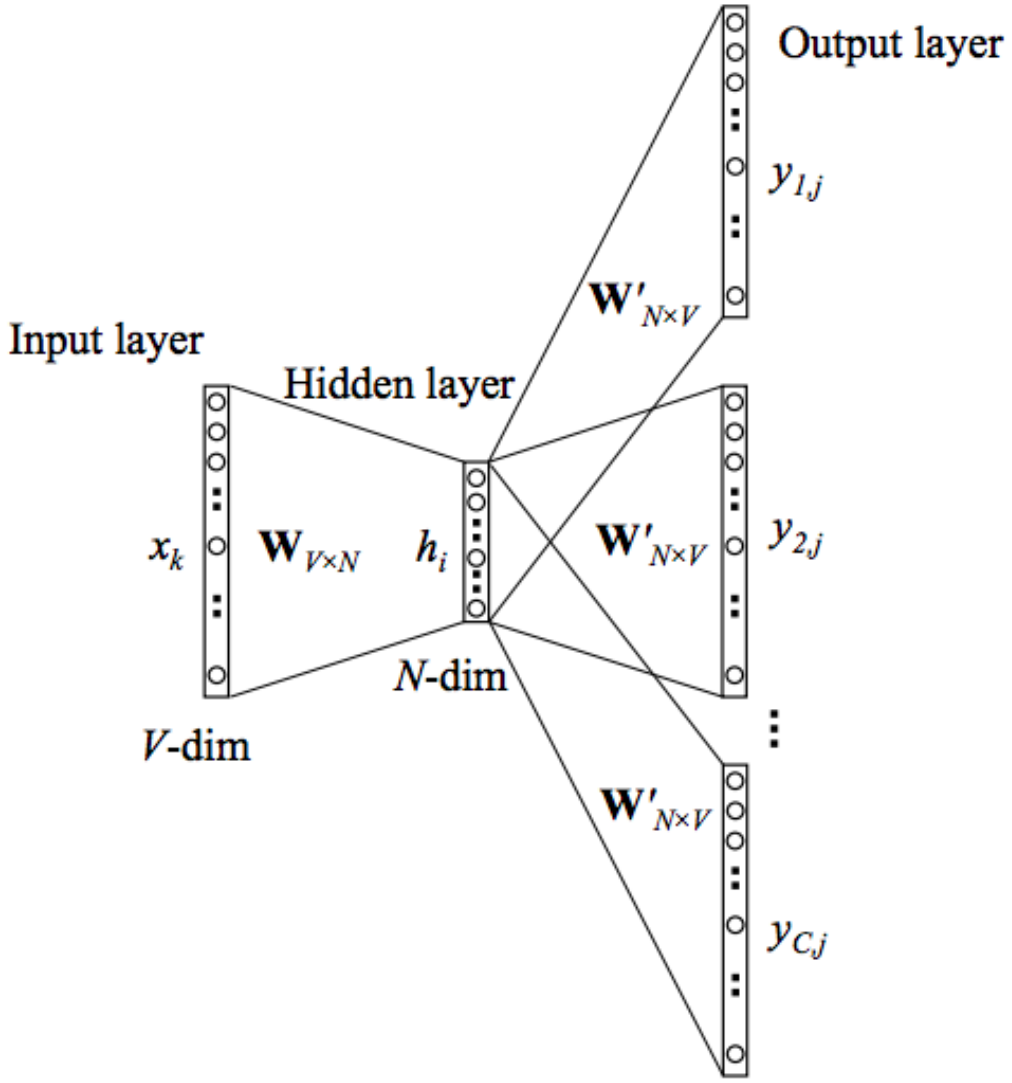
- We can also analyze the meaning of a particular word by looking at the contexts in which it occurs.
- The context is the set of words that occur near the word, i.e. at displacements of …,-3,-2,-1,+1,+2,+3,… in each sentence where the word occurs.
- A skip-gram is a set of non-consecutive words (with specified offset), that occur in some sentence.
- We can construct a BoSG (bag of skip-gram) representation for each word from the skip-gram table.
  - Example?

word2Vec: Local contexts

- Instead of entire documents, ***Word2Vec*** uses words ***k*** positions away from each center word.
  - These words are called **context words**.
- Example for ***k=3***:
  - "It was a bright cold day in April, and the clocks were striking".
  - Center word: red (also called focus word).
  - Context words: blue (also called target words).
- Word2Vec considers all words as center words, and all their context words.

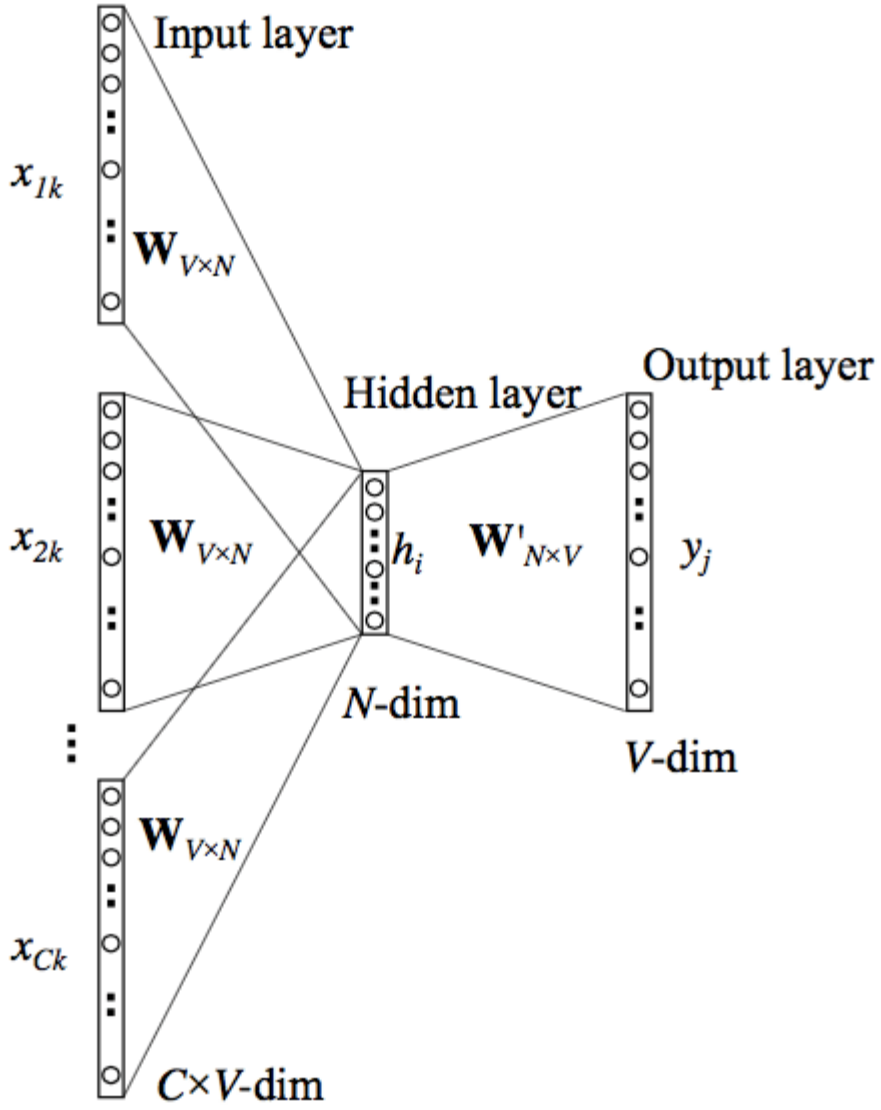
The skip-gram model

- Vocabulary size: V
- Input layer: center word in 1-hot form.
- k-th row of W_{VxN} is **center** vector of k-th word.
- k-th column of W'_{NXV} is **context** vector of the k-th word in V.
- **Note, each word has 2 vectors, both randomly initialized.**
- The output column y_{ij}, i=1..C, has 3 steps
  1) Use the context word 1-hot vector to choose its column in W'_{NxV}
  2) dot product with h_i the center word
  3) compute the softmax

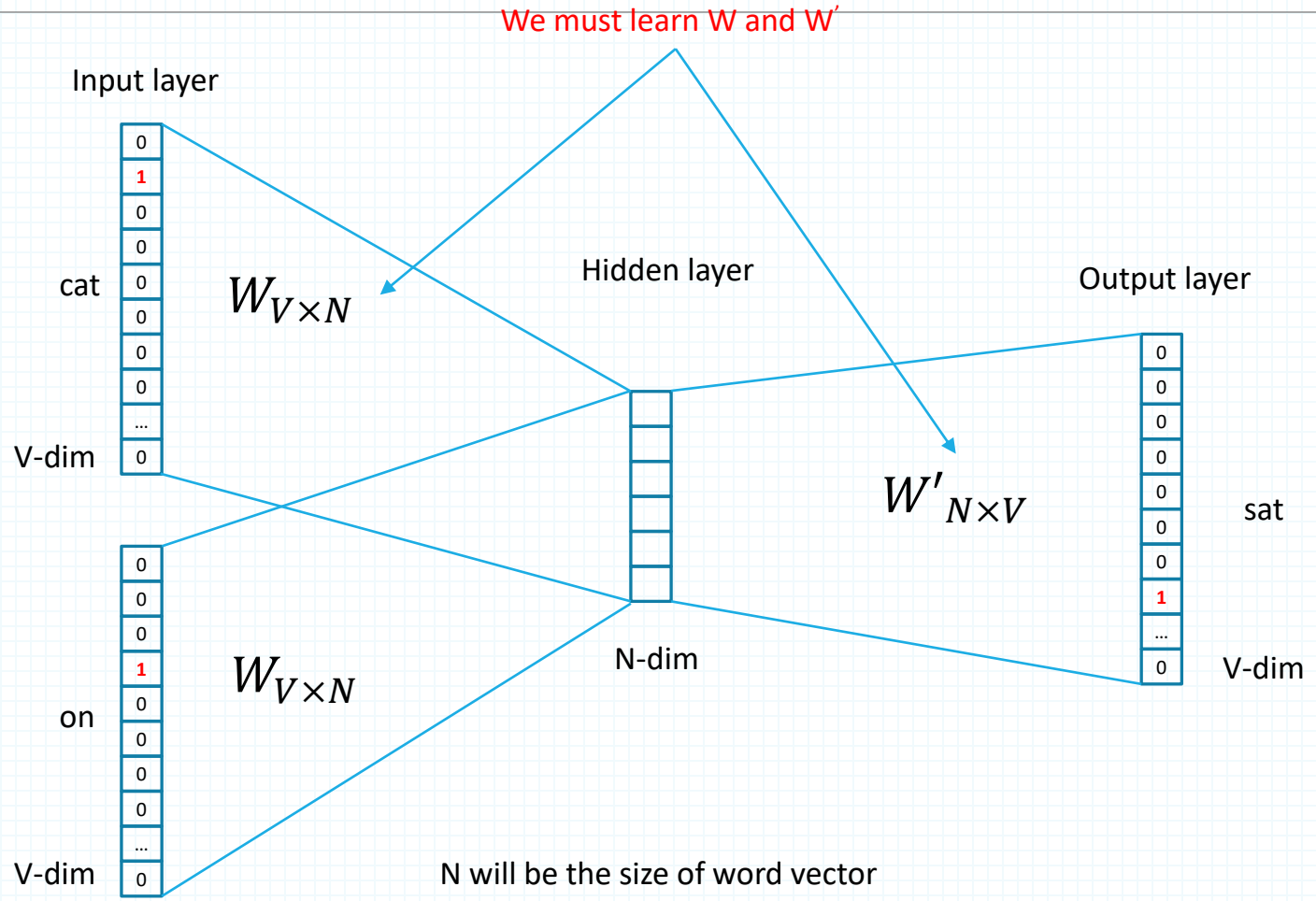


CBOW

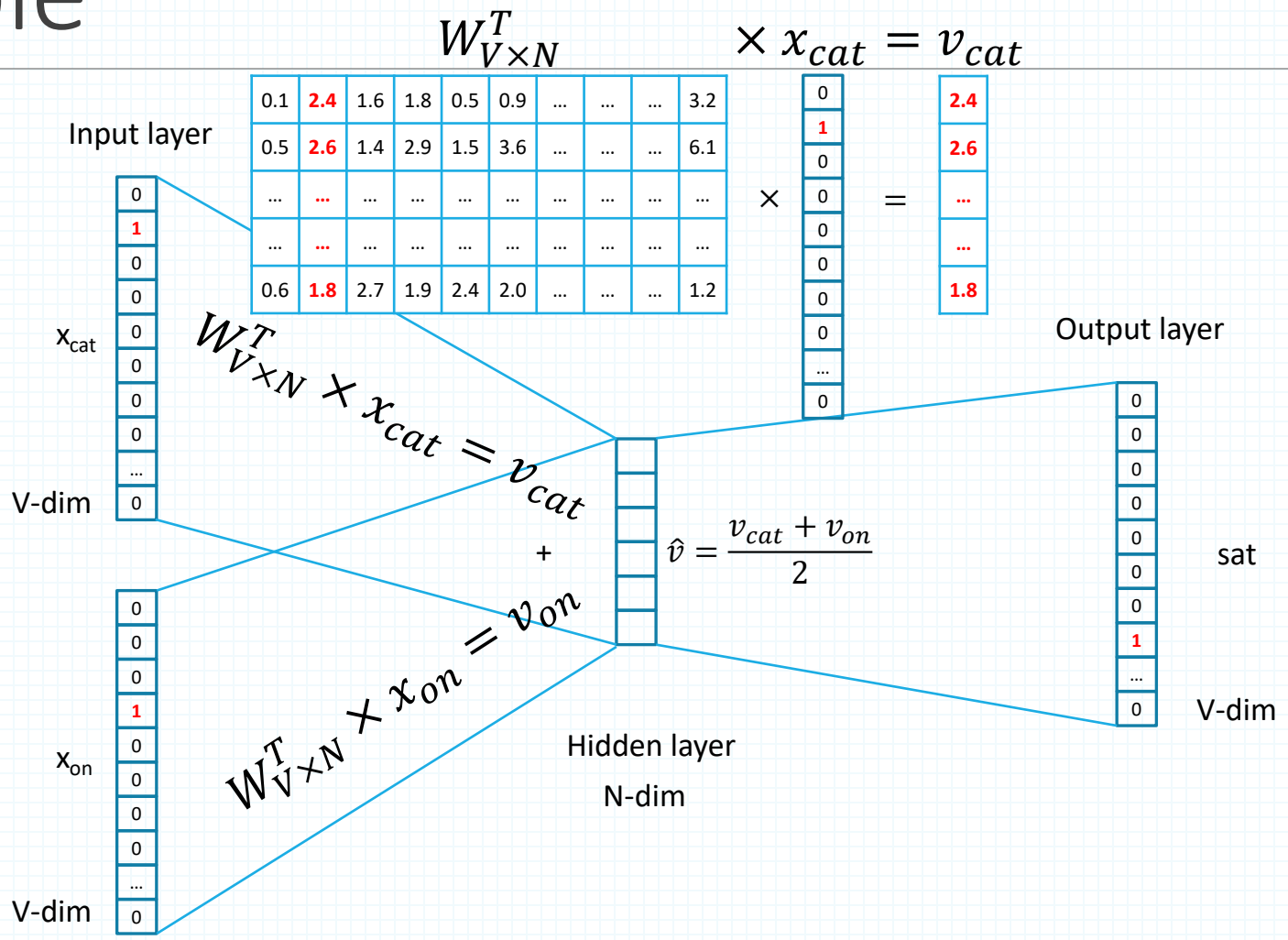
- What about we predict center word, given context word, opposite to the skip-gram model?
- Yes, this is called **Continuous Bag Of Words model** in the original Word2Vec paper.



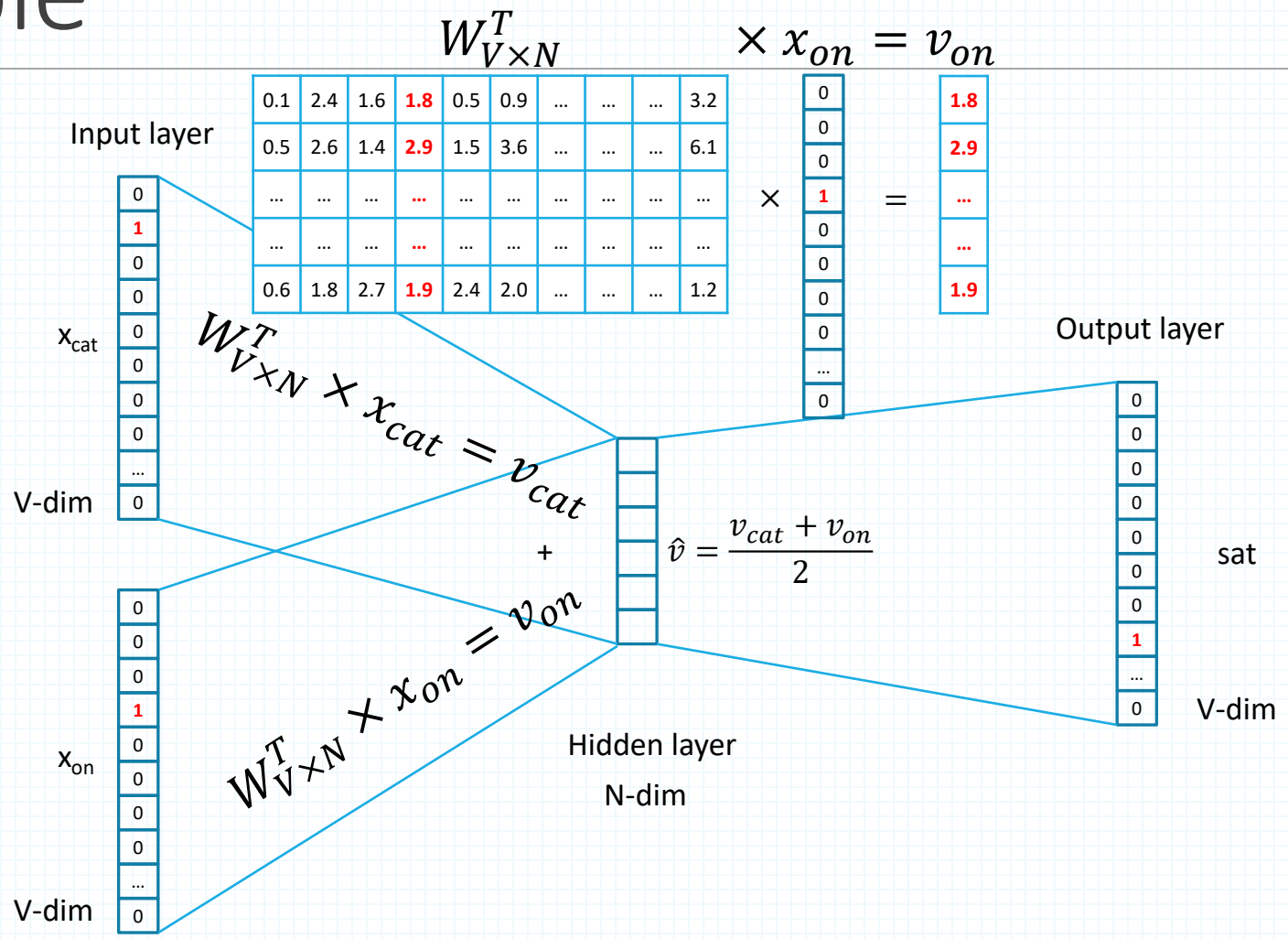
Example


We must learn W and W'
Input layer
cat
V-dim
on
V-dim
$W_{V\times N}$
$W_{V\times N}$
Hidden layer
N-dim
$W'_{N\times V}$
Output layer
sat
V-dim
N will be the size of word vector

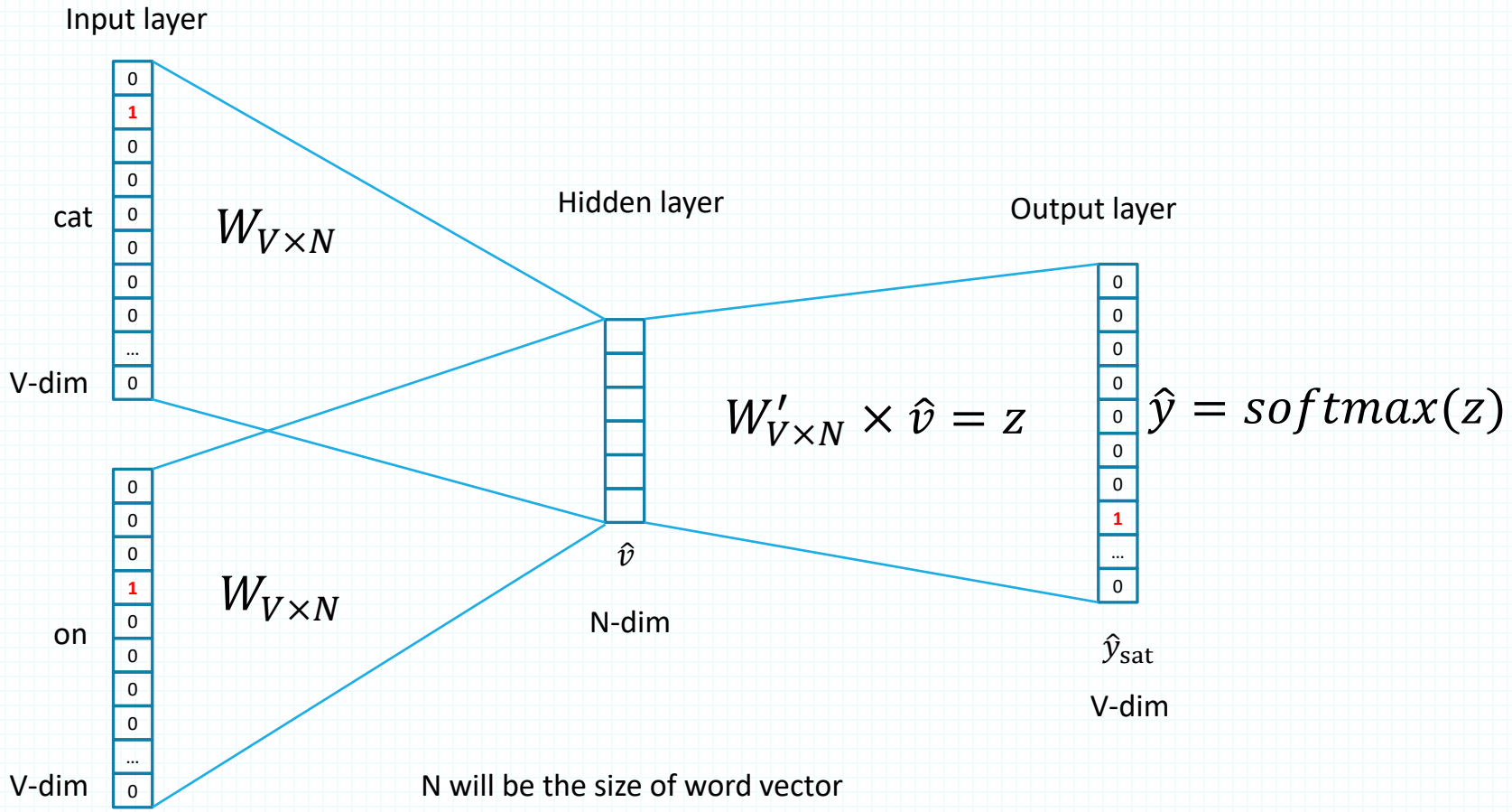
Example


$W^T_{V\times N}$
$\times x_{cat} = v_{cat}$
Input layer
x_{cat}
V-dim
$W^T_{V\times N} \times x_{cat} = v_{cat}$
+
$W^T_{V\times N} \times x_{on} = v_{on}$
x_{on}
V-dim
$\hat{v} = \frac{v_{cat} + v_{on}}{2}$
Hidden layer
N-dim
Output layer
sat
V-dim

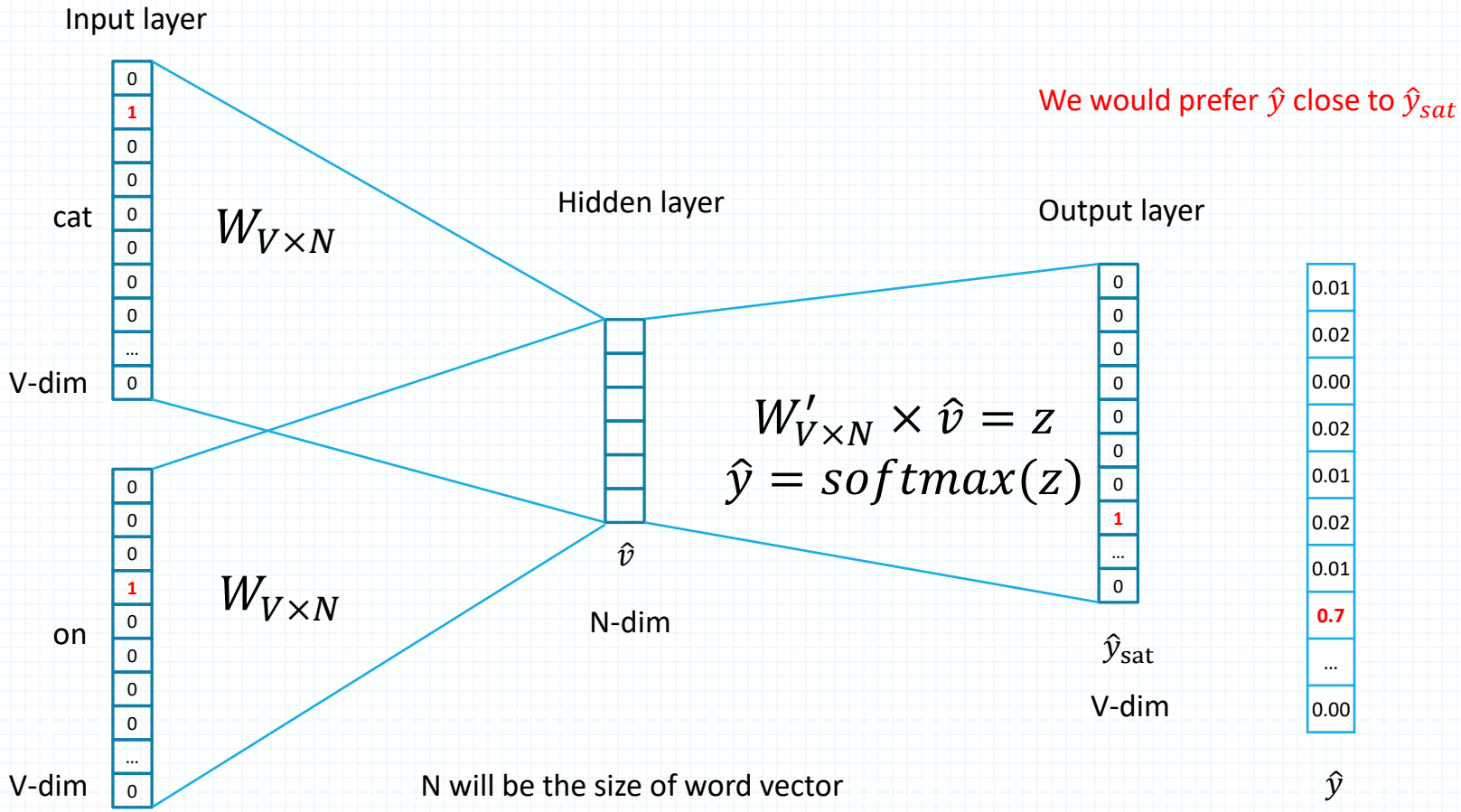
Example


$W^T_{V\times N}$
$\times x_{on} = v_{on}$
Input layer
x_{cat}
V-dim
x_{on}
V-dim
$W^T_{V\times N} \times x_{cat} = v_{cat}$
+
$W^T_{V\times N} \times x_{on} = v_{on}$
$\hat{v} = \frac{v_{cat} + v_{on}}{2}$
Hidden layer
N-dim
Output layer
sat
V-dim

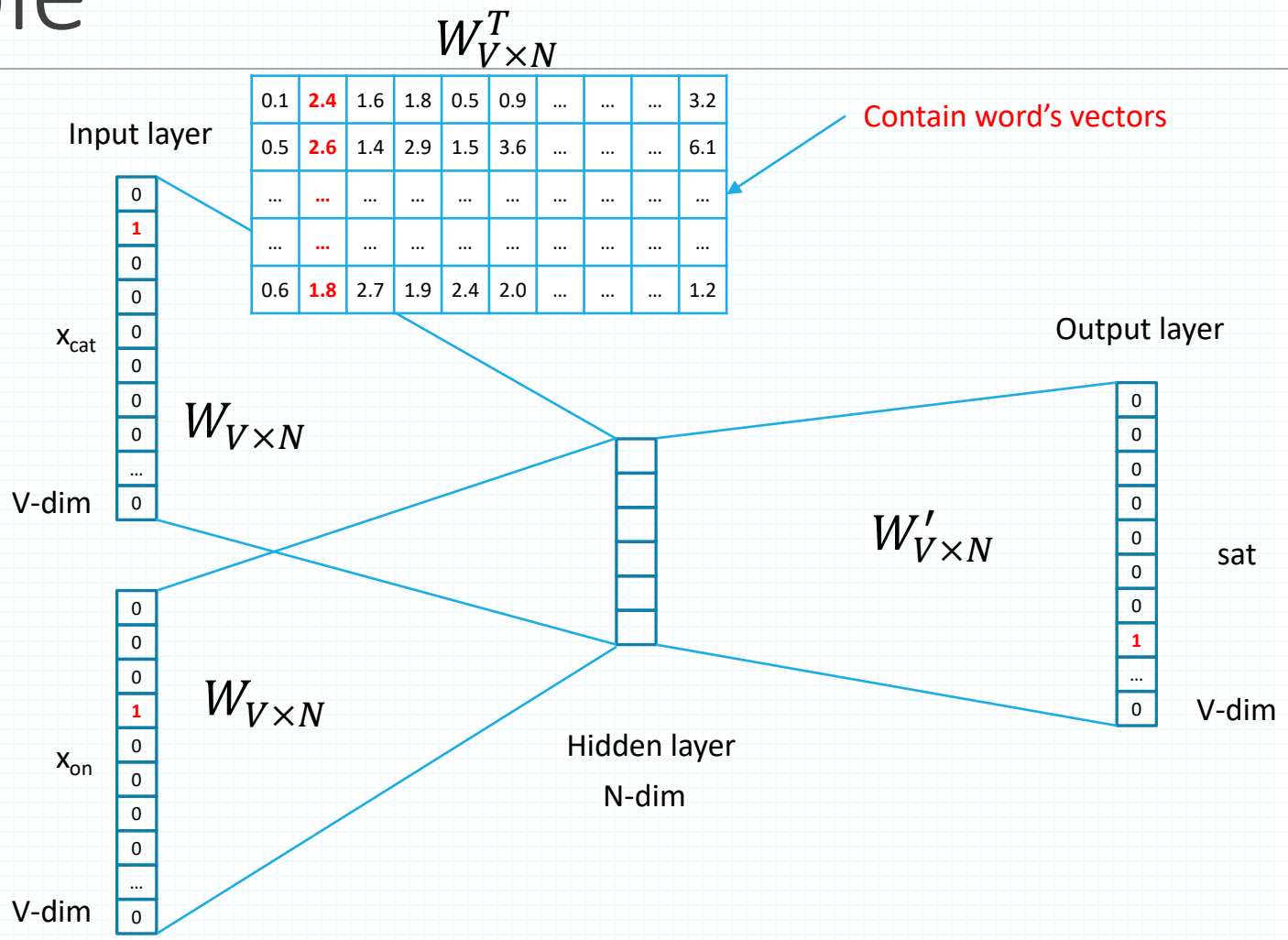
Example


Input layer
cat
V-dim
on
V-dim
$W_{V\times N}$
$W_{V\times N}$
Hidden layer
$\hat{v}$
N-dim
$W'_{V\times N} \times \hat{v} = z$
Output layer
$\hat{y} = softmax(z)$
$\hat{y}_{sat}$
V-dim
N will be the size of word vector

Example


Input layer
cat
V-dim
on
V-dim
$W_{V\times N}$
$W_{V\times N}$
Hidden layer
$\hat{v}$
N-dim
$W'_{V\times N} \times \hat{v} = z$
$\hat{y} = softmax(z)$
Output layer
$\hat{y}_{sat}$
V-dim
We would prefer $\hat{y}$ close to $\hat{y}_{sat}$
$\hat{y}$
N will be the size of word vector

Example



We can consider either W or W' as the word's representation. Or even take the average.

Using Pretrained Model

Do I need to Train Word2Vec?

- Answer: NO!
- You can download pre-trained Word2Vec models trained on massive corpora of data.
- Common example: Google News Vectors, 300 dimensional vecto for 3 million words, trained on Google News articles.
- File containing vectors (1.5 GB) can be downloaded for free and easily loaded into gensim.

```
from gensim.models import Word2Vec
import gensim.downloader as api
v2w_model = v2w_model = api.load('word2vec-google-news-300')
sample_word2vec_embedding=v2w_model['computer'];
```

FastText

- Limitations of Word2Vec:
  - OOV problem.
  - Words like “meet” and “meeting” are learned independently (no param sharing)
- FastText use character n-gram (word hashing)
  - Basic model can be similar to word2vec model.
  - Words are represented by all char n-grams of length 3 to 6.
  - Represent words based on its n-gram embeddings.

- short n-grams (n = 4) are good to capture syntactic information
- longer n-grams (n = 6) are good to capture semantic information