



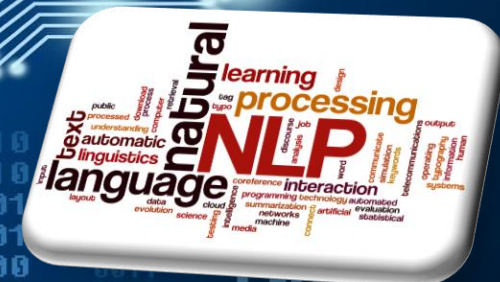
Cairo University

CAIRO UNIVERSITY

Cairo University
Faculty of Engineering
Computer Engineering Department

Natural Language Processing

Dr. Sandra Wahid



Word Embeddings

- Previously, we saw how to represent a word as a **sparse, long vector** with dimensions corresponding to words in the vocabulary or documents in a collection.
- We now introduce a more powerful word representation: **embeddings, short dense vectors**.
 - With number of dimensions d ranging from 50-1000.
 - The vectors are dense: instead of vector entries being sparse, mostly-zero counts or functions of counts, the values will be real-valued numbers that can be negative.
- It turns out that **dense vectors work better in every NLP task** than sparse vectors.
- One method for computing embeddings is **Word2vec** where embeddings are **static**: meaning that the method **learns one fixed embedding** for each word in the **vocabulary**.
- Other methods such as **BERT**, are for learning **dynamic contextual** embeddings in which the vector for each word is different in different contexts.

Word2vec

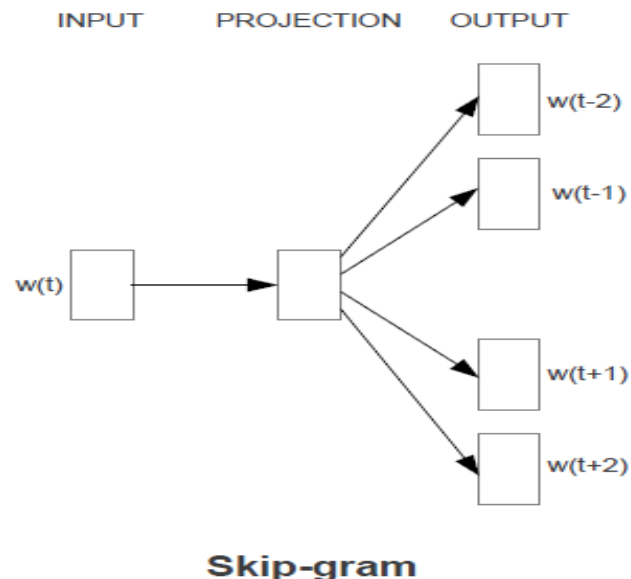
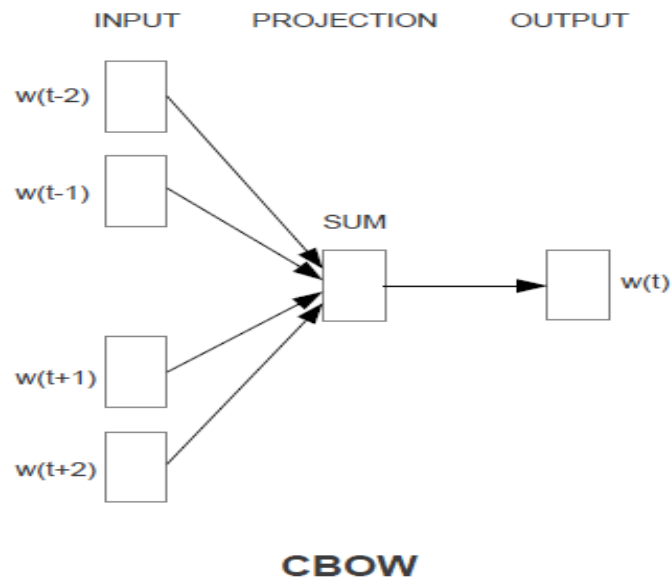
- Word2vec is developed by *Tomas Mikolov et al. at Google* where two models are proposed:

CBOW

- Continuous Bag-Of-Words
- where a word is predicted based on its context (the surrounding words).

SG

- Continuous Skip-Gram
- where the surrounding words are predicted given a current word.



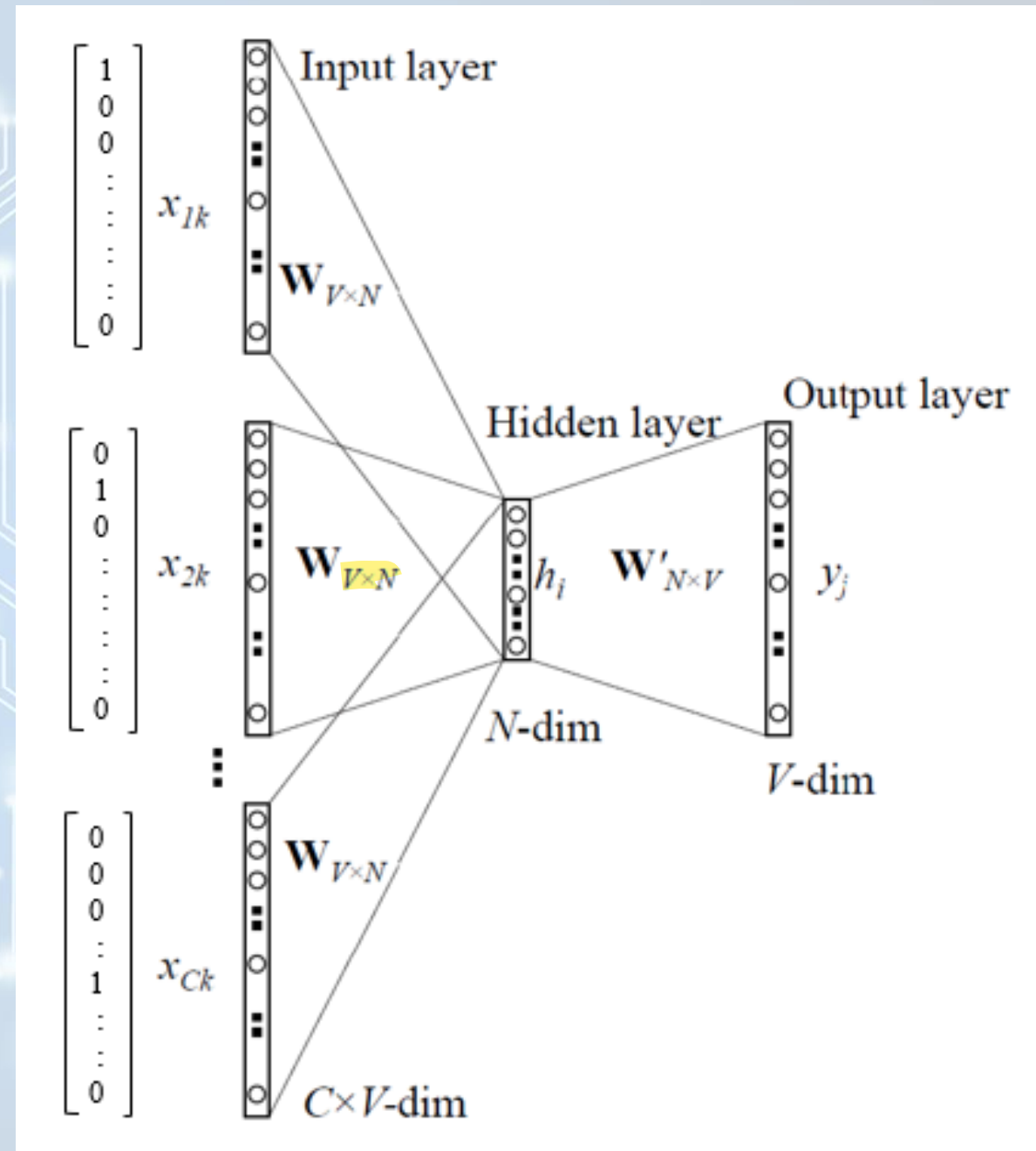
The models' architecture is a **neural network** with **three layers**: an input layer, a hidden layer and an output layer. The models are trained using **huge text corpora**.

Though, **we won't use the neural network after training!!!**

CBOW

Input Layer:

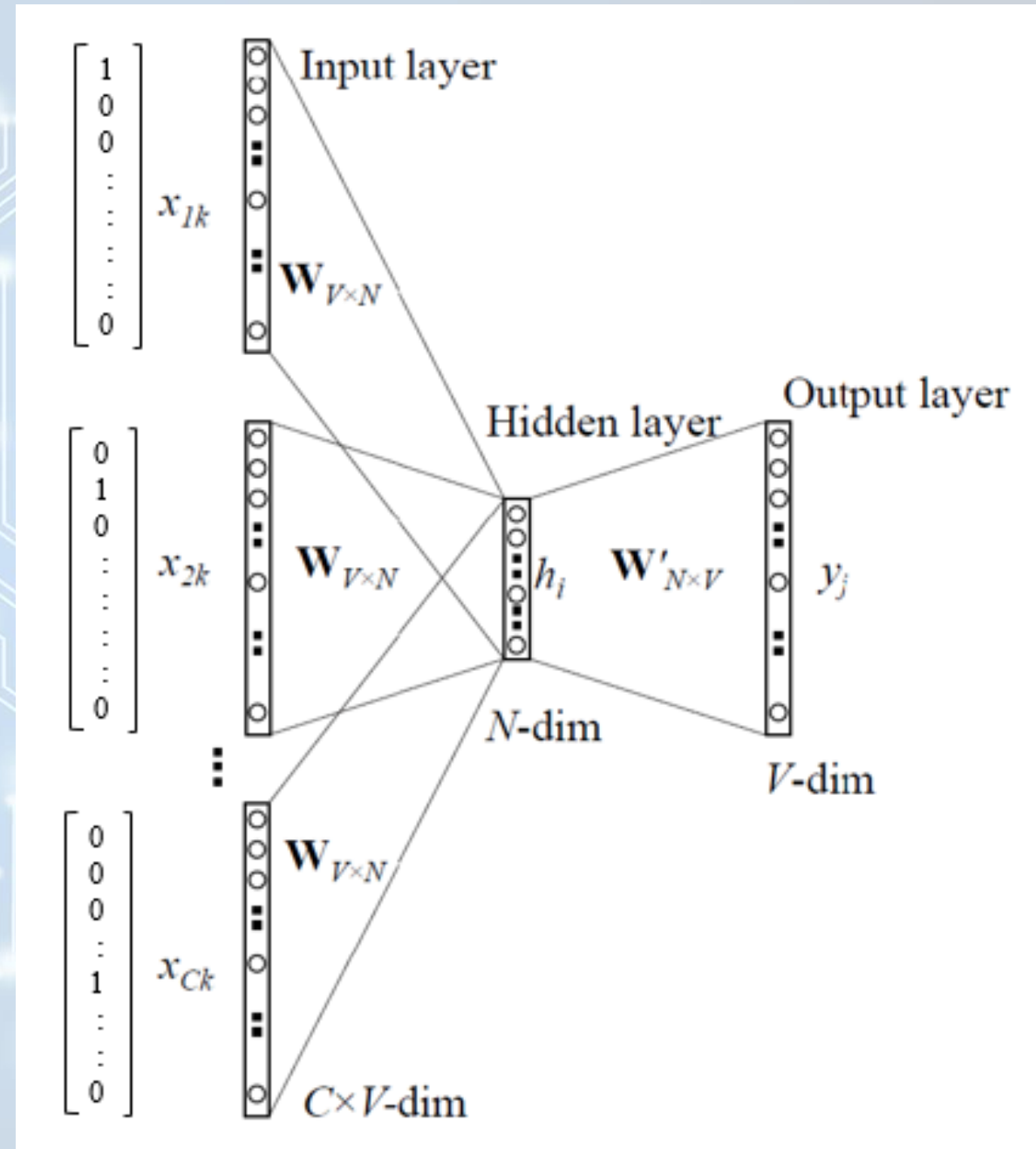
- Its size is CV
 - C =number of context words
 - V =number of words in the vocabulary
- The inputs are **C context vectors**:
 - Each context vector is one-hot encoded vector of size V (it is a vector having 1 in the position encoding the word, and 0 otherwise).



CBOW

Hidden Layer:

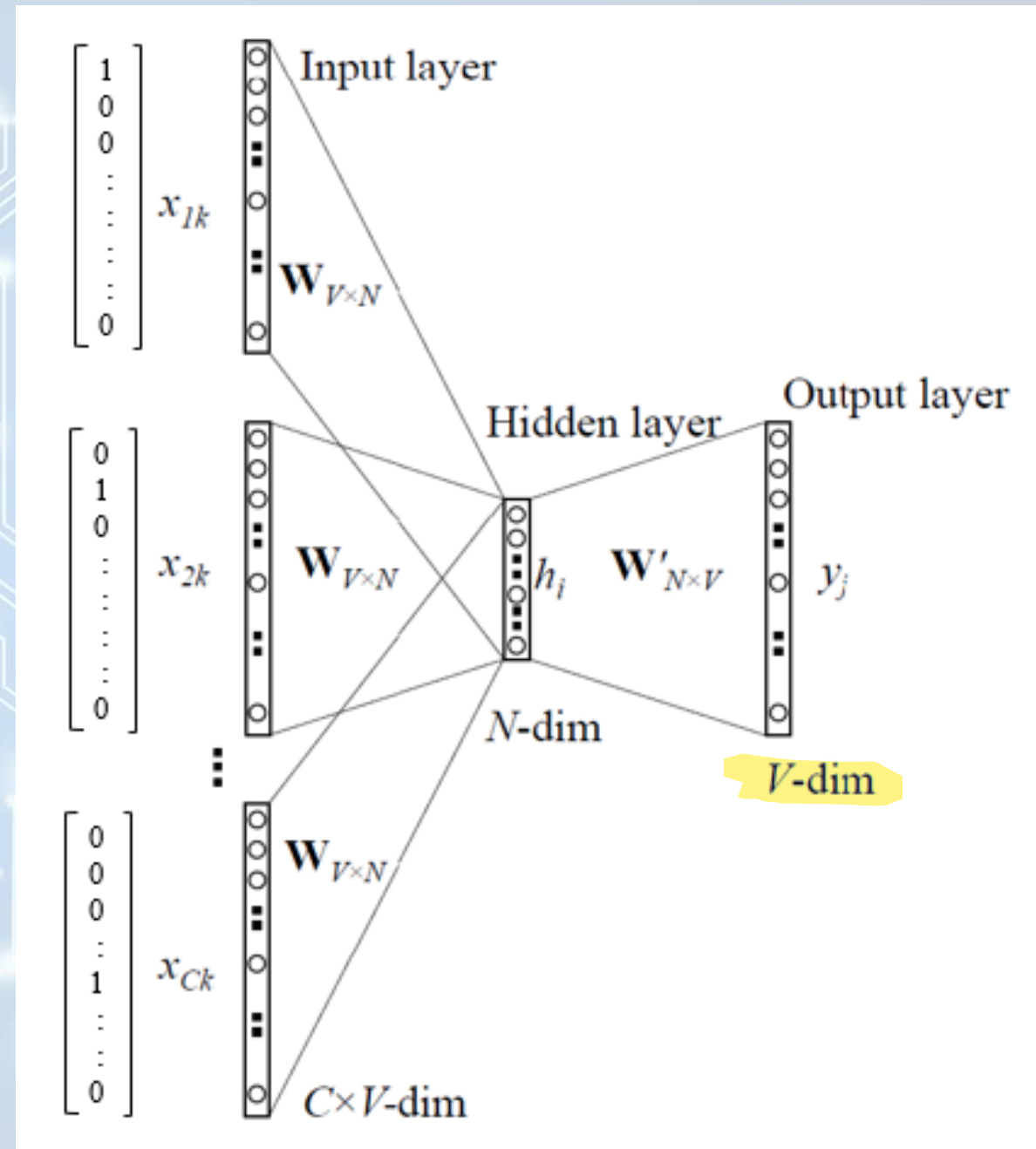
- Its size is N
 - N=number of neurons in the hidden layer
=the dimension of embeddings
(it is a hyper parameter)
- Fully-connected (dense) layer whose weights are the word embeddings.



CBOW

Output layer:

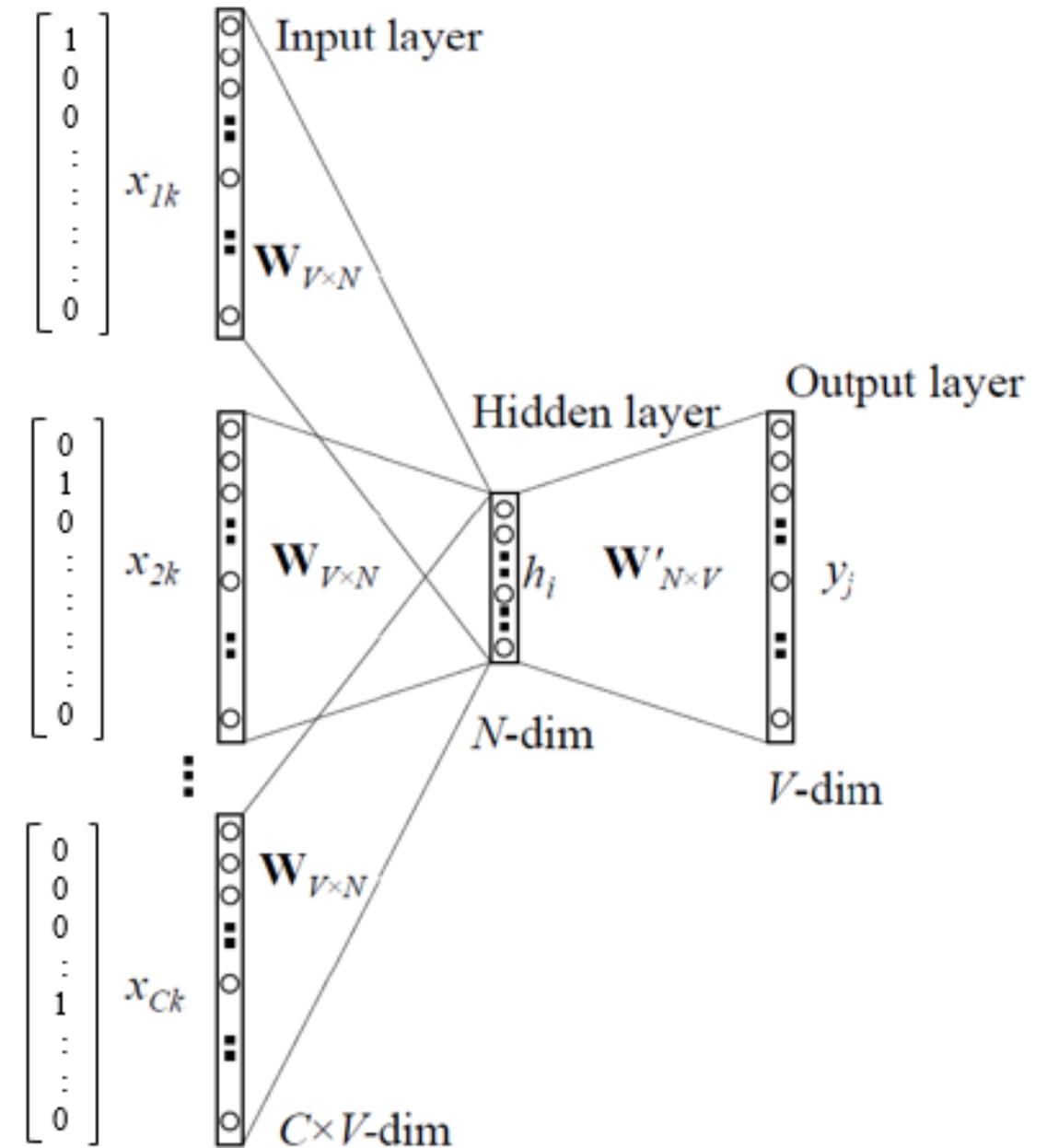
- Its size is V
 - Where the gold output is the one-hot vector encoding the “middle/target word”.
- It is a softmax layer which is designed so that the sum of the probabilities obtained in the output layer equals 1.
 - The network is going to tell us the probability for every word in our vocabulary of being the “target word” for the given input(s) “context word(s)”



CBOW

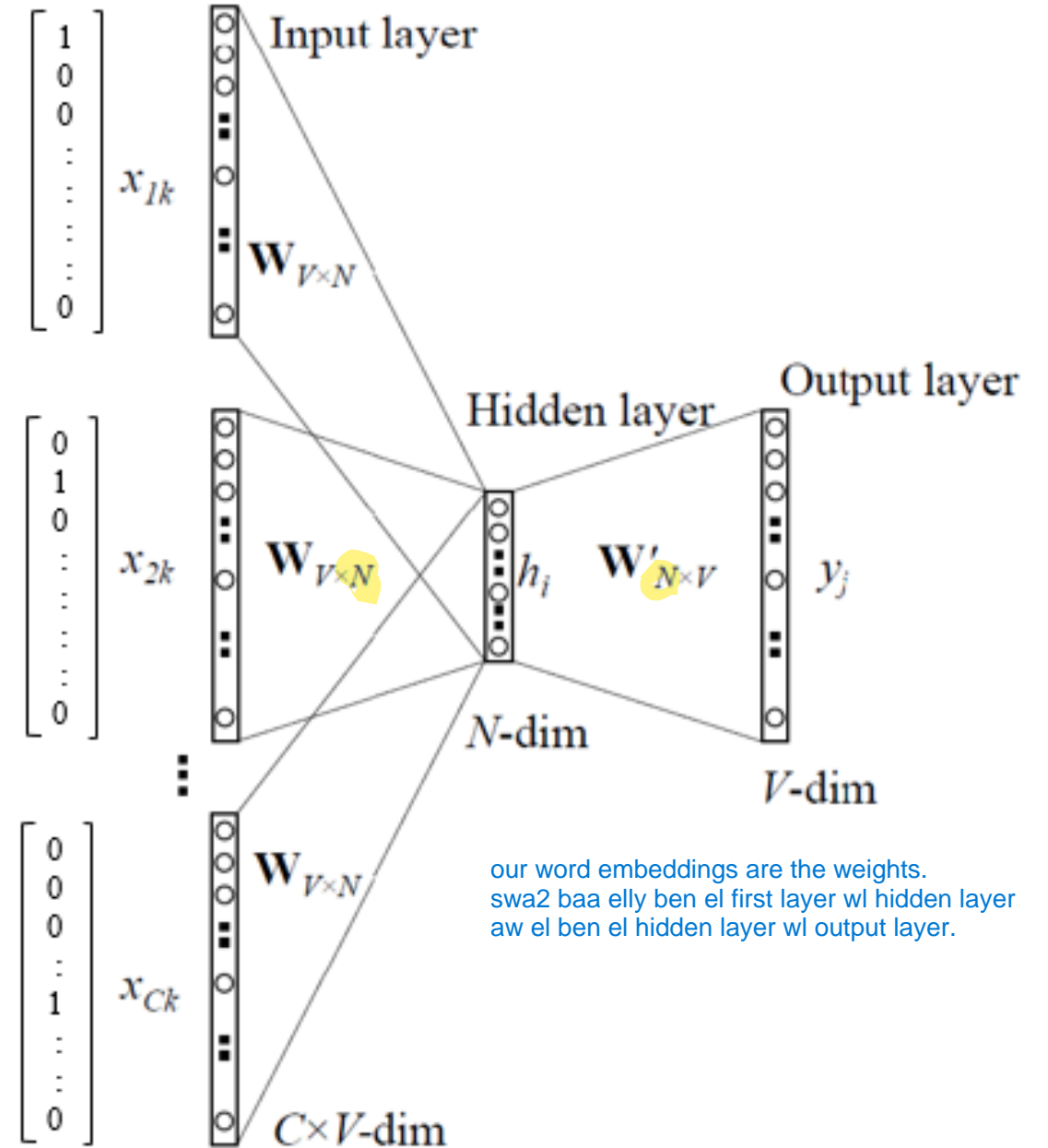
Weights:

- There are two sets of weights:
 - one is between the input and the hidden layer:
Input-Hidden layer matrix consisting of C $W[V \times N]$ matrices
 - second is between hidden and output layer:
Hidden-Output layer matrix size $= [N \times V]$



CBOW

- There is a **no activation function between any layers (linear activation):**
 - Linear network **except** for the **softmax** at the output layer.
- The input-to-hidden-layer weight matrix is multiplied with the input vector to produce the hidden layer output.
- The hidden input gets multiplied by hidden-output weights to produce the output then pass it to softmax.
- Error between output and target is calculated and back propagated to re-adjust the weights.
- The weights between the input layer and the hidden layer are taken as the word vectors after training.



CBOW

- Let the available corpus be this one sentence *“word vectors are widely used”*.
- The unique words in the vocabulary are 5 words ($V = 5$) as follows:
[“word”, “vectors”, “are”, “widely”, “used”].
- Training data for $C=1$ (context words)

training sample	context word	target word
1	word	vectors
2	vectors	are
3	are	widely
4	widely	used

- **Encoded** training data:

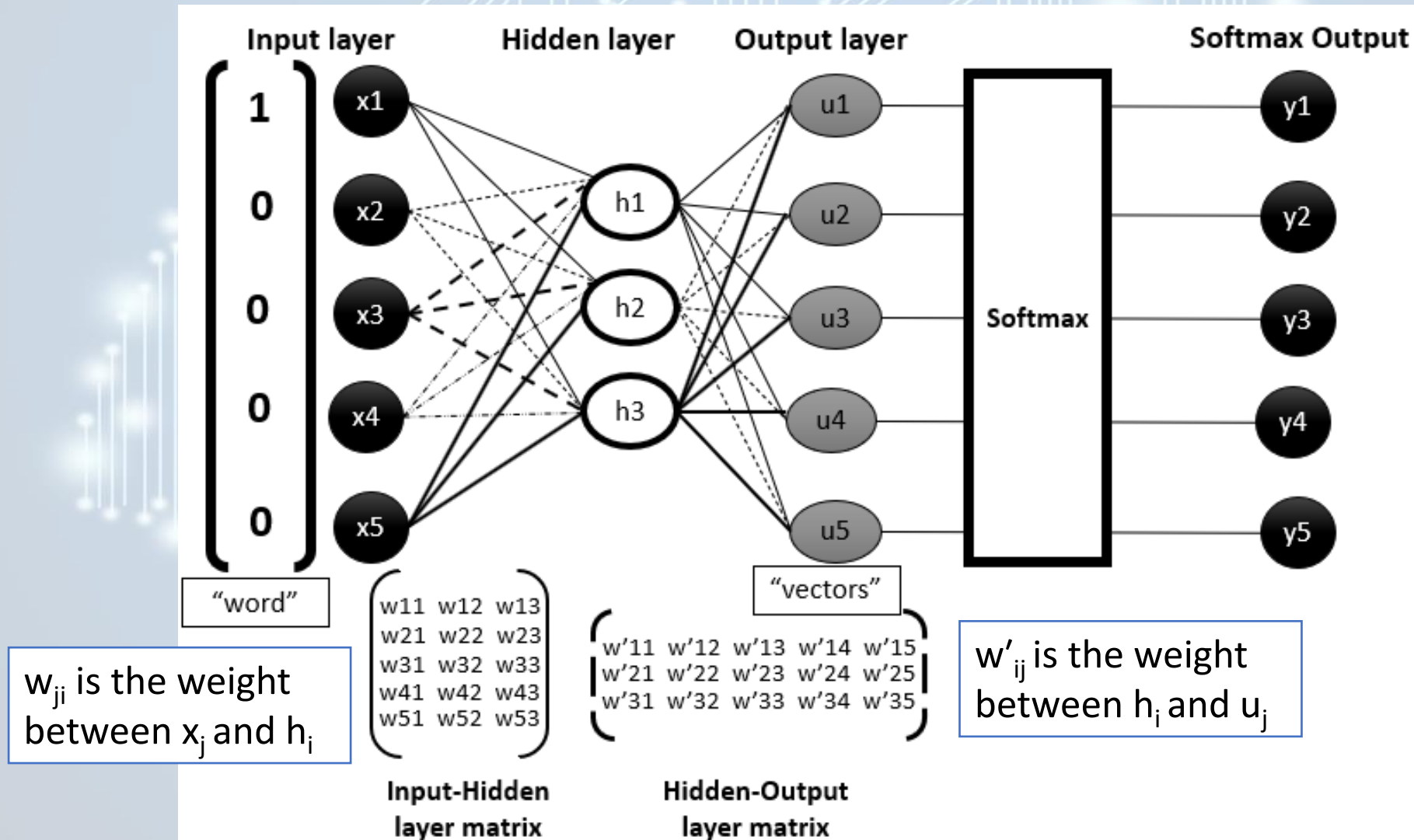
training sample	context word	target word
1	[1,0,0,0,0]	[0,1,0,0,0]
2	[0,1,0,0,0]	[0,0,1,0,0]
3	[0,0,1,0,0]	[0,0,0,1,0]
4	[0,0,0,1,0]	[0,0,0,0,1]

commonly used 300 & 100

CBOW

- Let $N = 3$, consider the first training sample:

training sample	context word	target word
1	[1,0,0,0,0]	[0,1,0,0,0]



CBOW

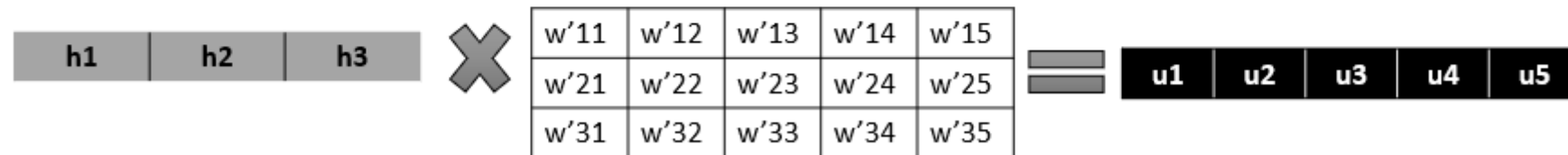
- Equations:

$$h_i = \sum_j w_{ji} * x_j \text{ where } i = 1, 2, 3 \text{ and } j = 1, 2, 3, 4, 5 \quad (1)$$

$$u_j = \sum_i w'_{ij} * h_i \text{ where } i = 1, 2, 3 \text{ and } j = 1, 2, 3, 4, 5 \quad (2)$$

$$y_j = \text{Softmax}(u_j) \text{ where } j = 1, 2, 3, 4, 5 \quad (3)$$

- Matrix Forms:



$\begin{bmatrix} 0 & 0 & 0 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} 17 & 24 & 1 \\ 23 & 5 & 7 \\ 4 & 6 & 13 \\ 10 & 12 & 19 \\ 11 & 18 & 25 \end{bmatrix} = \begin{bmatrix} 10 & 12 & 19 \end{bmatrix}$

The output of the hidden layer is just the “word vector” for the input word.

CBOW

- The error calculations are performed given the target vector: $[0,1,0,0,0]$ through **back propagation** where the weight values (w and w') are updated.
- The training continues for **many iterations** until the weights are **adjusted** for all the **training samples**.
- Finally, the vectors of the words in the vocabulary **are weights** in the Input-Hidden layer matrix such that each row in the matrix is the word vector representation of the equivalent word.

The word in the vocabulary	The equivalent word vector
word	$[w_{11} \ w_{12} \ w_{13}]$
vectors	$[w_{21} \ w_{22} \ w_{23}]$
are	$[w_{31} \ w_{32} \ w_{33}]$
widely	$[w_{41} \ w_{42} \ w_{43}]$
used	$[w_{51} \ w_{52} \ w_{53}]$

CBOW

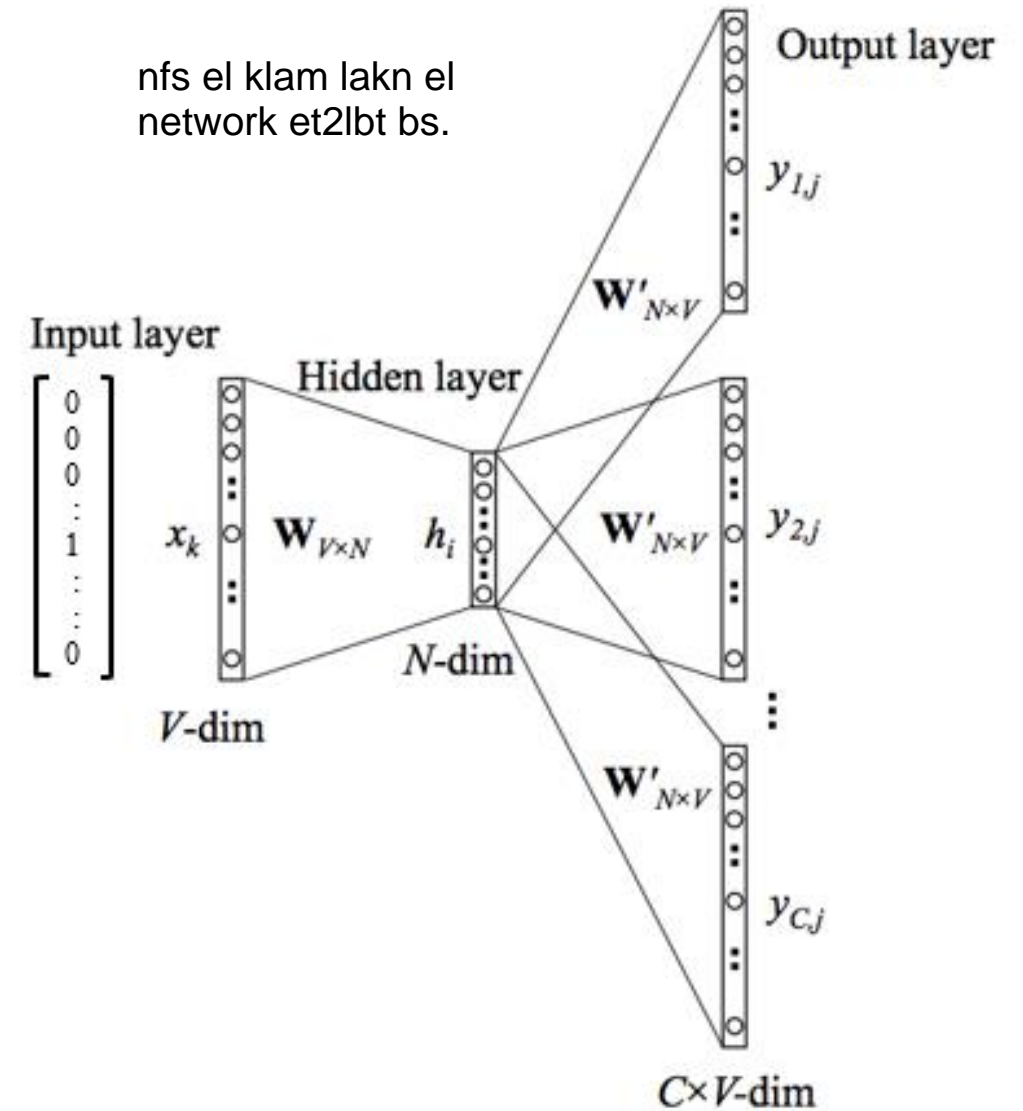
- For $C > 1$ (context words), the only difference will be in handling the C input vectors.
 - Each vector will be multiplied with the input-hidden layer matrix $W[V \times N]$ returning C $[1 \times N]$ vectors.
 - Finally, to obtain a single vector \rightarrow all C $[1 \times N]$ vectors will be averaged element-wise.
- This computation is equivalent to initially computing a **mean vector** for the C input **one-hot encoded vectors** then proceeding **as in the case of $C = 1$** .
- For $C=2 \rightarrow$ window size=1

training sample	context word	target word
1	(word,are)	vectors
2	(vectors,widely)	are
3	(are,used)	widely
4	widely	used

training sample	context words	mean context word	target word
1	$([1,0,0,0,0],[0,0,1,0,0])$	$[0.5,0,0.5,0,0]$	$[0,1,0,0,0]$
2	$([0,1,0,0,0],[0,0,0,1,0])$	$[0,0.5,0,0.5,0]$	$[0,0,1,0,0]$
3	$([0,0,1,0,0],[0,0,0,0,1])$	$[0,0,0.5,0,0.5]$	$[0,0,0,1,0]$
4	$[0,0,0,1,0]$	$[0,0,0,1,0]$	$[0,0,0,0,1]$

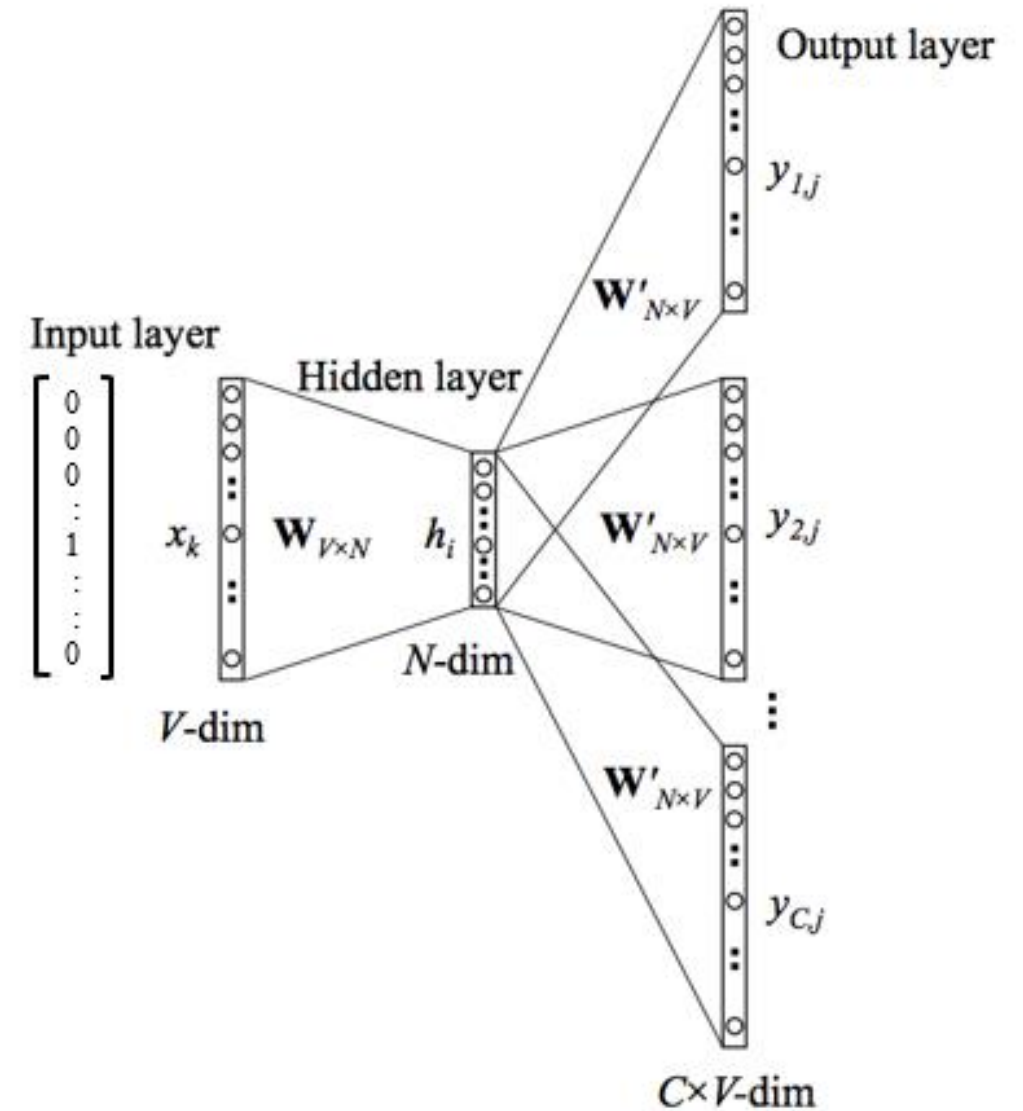
Input Layer:

- Its size is V
 - V =number of words in the vocabulary
- The input is **1 middle/target vector**:
 - the target vector is a **one-hot encoded** vector (a vector having 1 in the position encoding the word, and 0 otherwise).



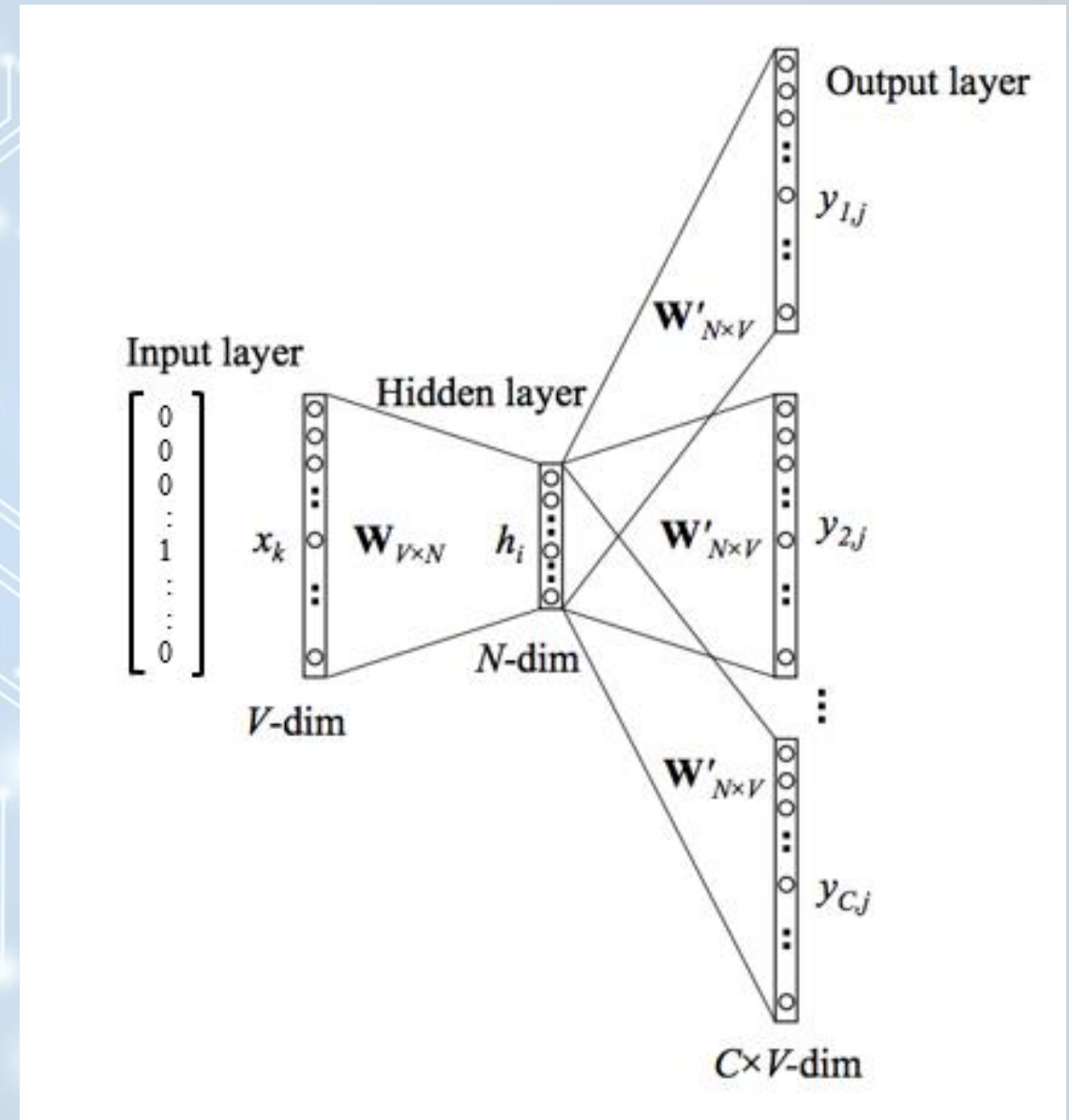
Hidden Layer:

- Its size is N
 - N =number of neurons in the hidden layer
=the dimension of embeddings
(it is a hyper parameter)
- Fully-connected (dense) layer whose weights are the word embeddings.



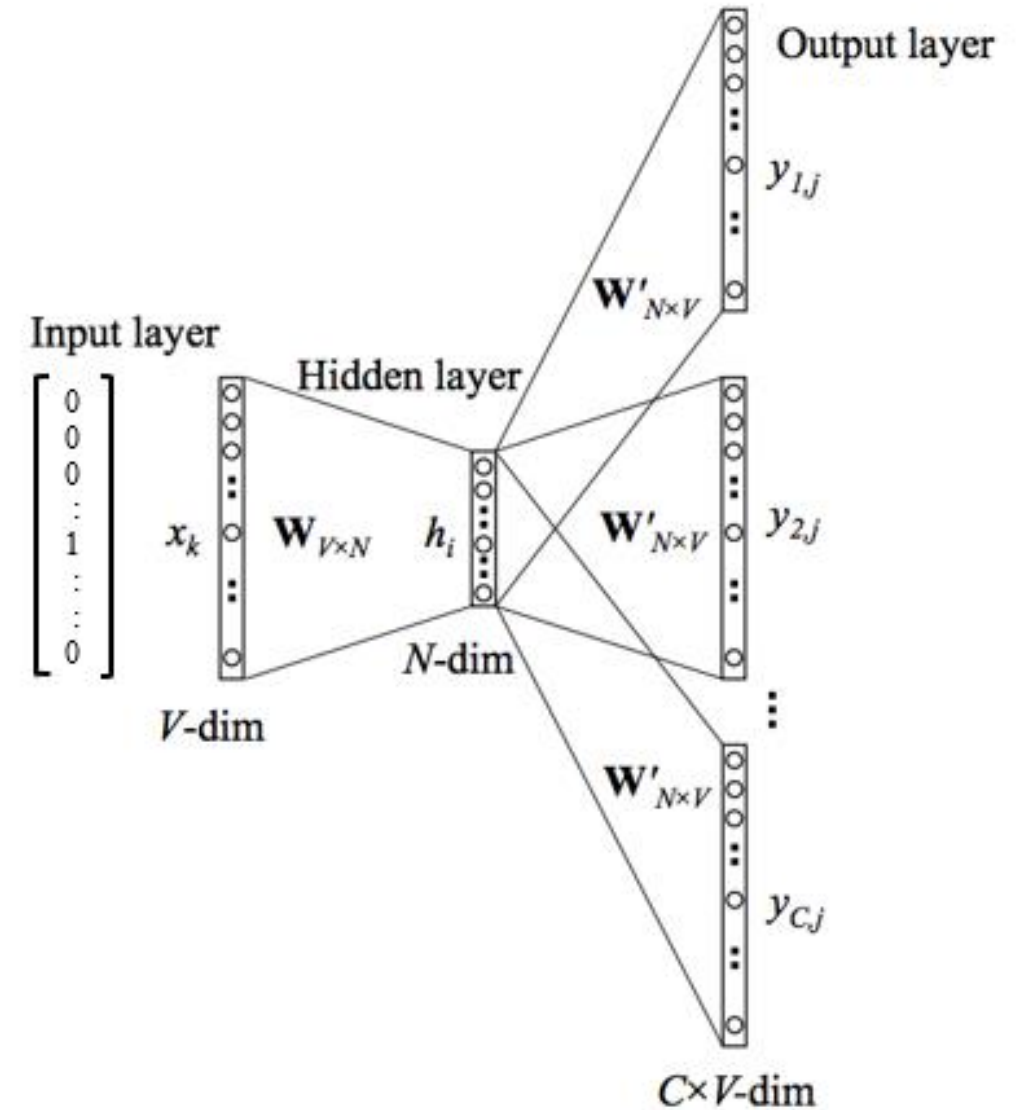
Output Layer:

- Its size is **CV**
 - C=number of context words
 - V=number of words in the vocabulary
 - Where the gold outputs are C one-hot vectors each encoding one of the “context words”.
- It is a softmax layer which is designed so that the sum of the probabilities obtained in the output layer equals 1.
 - The network is going to tell us the probability for every word in our vocabulary of being the “context word” for the given input “target word”
- **C separate errors** are calculated and are **added element-wise** to obtain a **final error vector** which is back propagated to **update the weights**.

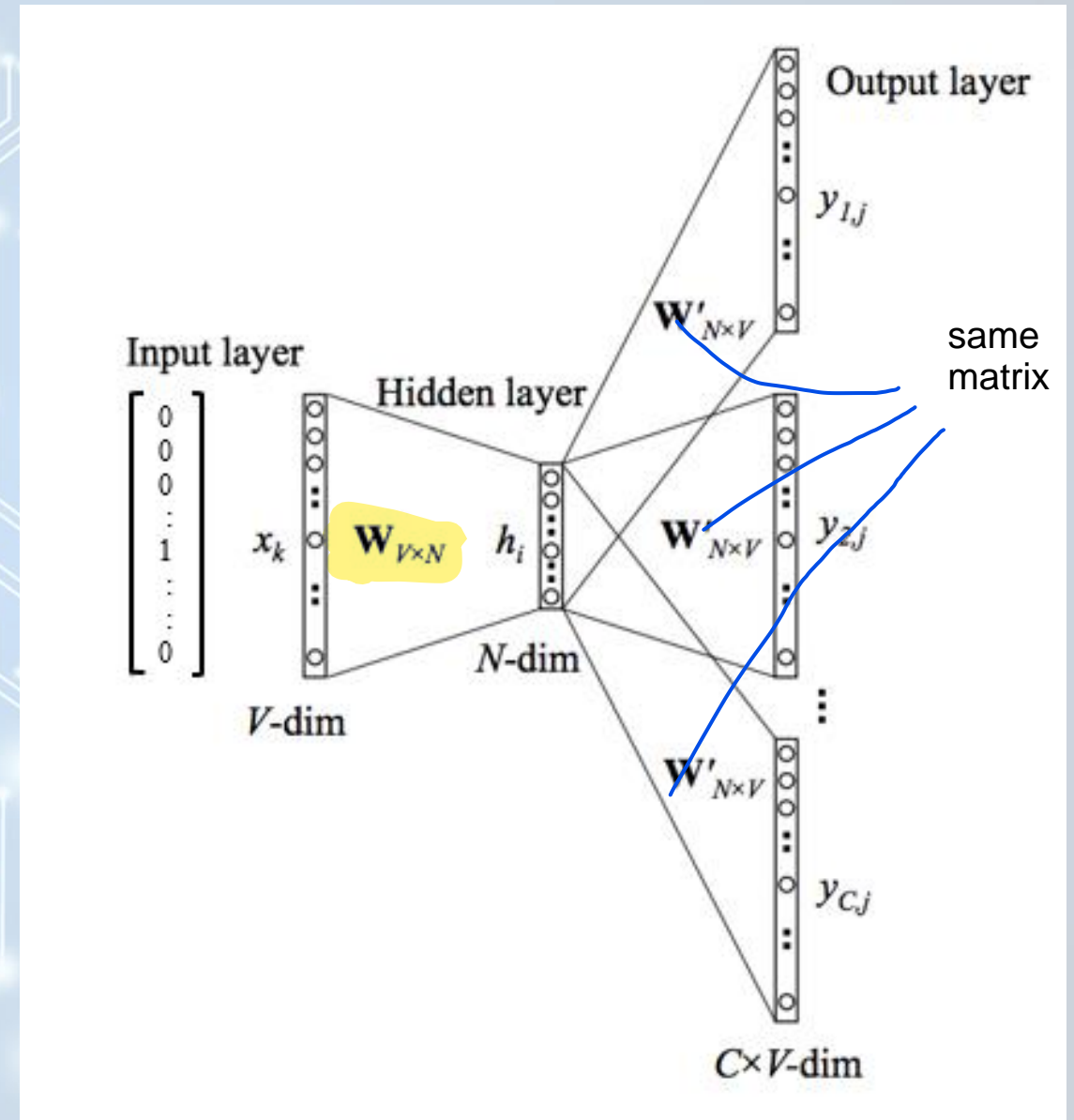


Weights:

- There are two sets of weights:
 - one is between the input and the hidden layer:
Input-Hidden layer matrix size $= [V \times N]$
 - second is between hidden and output layer:
Hidden-Output layer matrix consisting of C $W' [N \times V]$ matrices



- There is a no activation function between any layers (linear activation):
 - Linear network except for the softmax at the output layer.
- The weights between the input layer and the hidden layer are taken as the word vectors after training.



- Let the available corpus be this one sentence “*word vectors are widely used*”.
- The unique words in the vocabulary are 5 words ($V = 5$) as follows:
[“word”, “vectors”, “are”, “widely”, “used”].
- Training data for $C=2$ (context words) \rightarrow window size=1

training sample	context words	target word
1	(word,are)	vectors
2	(vectors,widely)	are
3	(are,used)	widely
4	(widely)	used

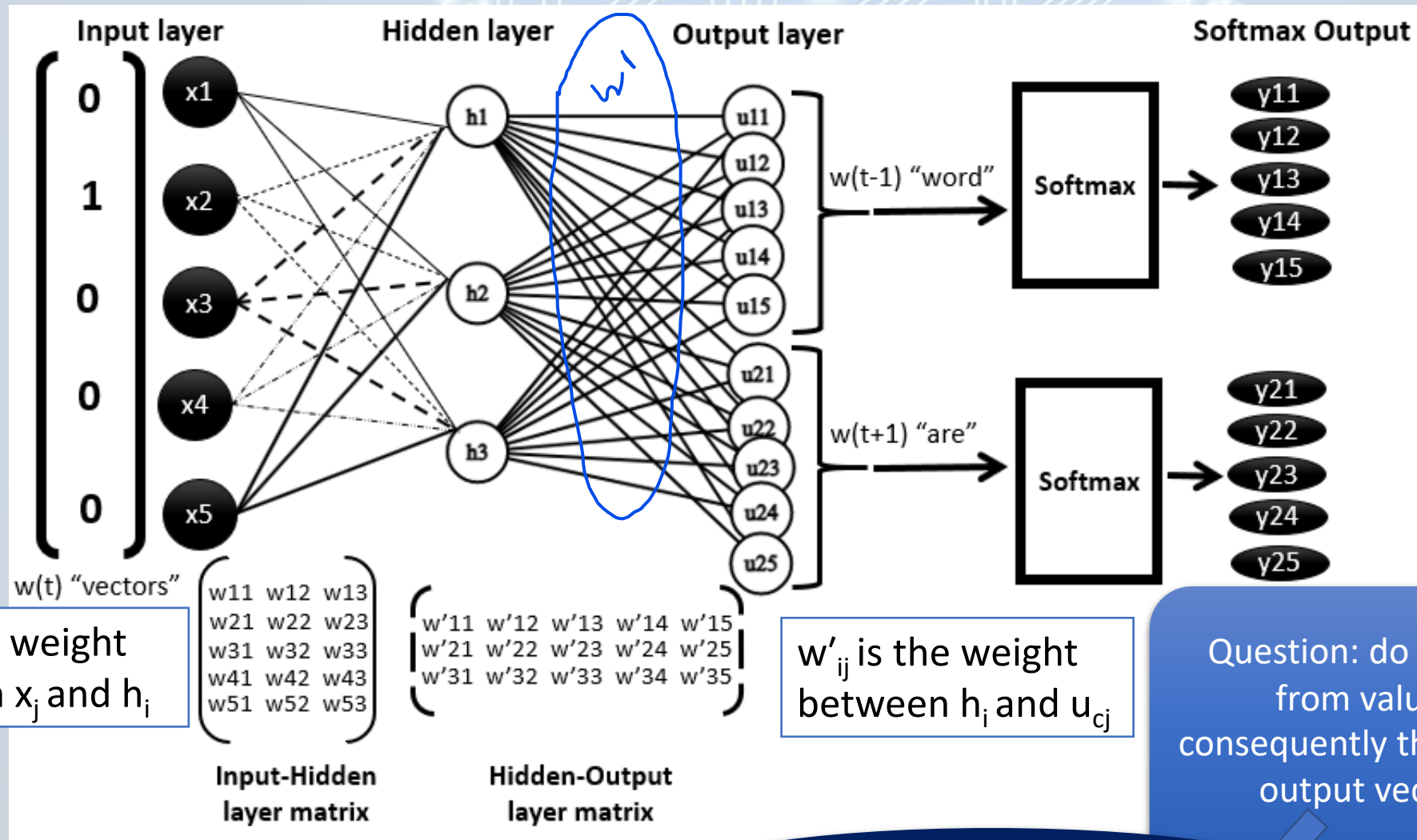
hena el 7aga mt3akstsh bs lazmt
teb2a fahem en el target 3ndna hena
howa el input, lakn el context word
howa el output.

- Encoded training data:

training sample	context words	target word
1	([1,0,0,0,0],[0,0,1,0,0])	[0,1,0,0,0]
2	([0,1,0,0,0],[0,0,0,1,0])	[0,0,1,0,0]
3	([0,0,1,0,0],[0,0,0,0,1])	[0,0,0,1,0]
4	([0,0,0,1,0])	[0,0,0,0,1]

- Let $N = 3$, consider the first training sample:

training sample	context words	target word
1	$([1,0,0,0,0],[0,0,1,0,0])$	$[0,1,0,0,0]$



kol el outputs henaar hayb2o
nfs el outputs, 34an enta el
hidden layers sabta, w el
matrix elly btdrb feha
wa7da, fa el output aked
msh hy5tlf, lakin el error
hwa elly hyt8yr.

Question: do values $u_{11} \rightarrow u_{15}$ differ
from values $u_{21} \rightarrow u_{25}$ (and
consequently the y) i.e. do we have one
output vector or C vectors???

Only one since W' matrix is
the same

- The training process is similar to CBOW except that the error calculation differs as there are C gold outputs. Therefore, the final error is computed as the summation of the error for each context word as follows:

$$E = \sum_{c=1}^C E_c$$

- Finally, the vectors of the words in the vocabulary are weights in the Input-Hidden layer matrix such that each row in the matrix is the word vector representation of the equivalent word.

The word in the vocabulary	The equivalent word vector
word	$[w_{11} \ w_{12} \ w_{13}]$
vectors	$[w_{21} \ w_{22} \ w_{23}]$
are	$[w_{31} \ w_{32} \ w_{33}]$
widely	$[w_{41} \ w_{42} \ w_{43}]$
used	$[w_{51} \ w_{52} \ w_{53}]$

SG Calculations

- For input x with $x_k=1$ and $x_{k'}=0$ for all $k' \neq k$, the outputs of the hidden layer will be row k in W : $\mathbf{h} = \mathbf{x}^T \mathbf{W} = \mathbf{W}_{(k, \cdot)} := \mathbf{v}_{w_I}$
- The input to the j th node of the c th output word is: $u_{c,j} = \mathbf{w}'_{w_j}{}^T \mathbf{h}$ → j th column in W' matrix
- Since the output layers for each output word share the same weights so $u_{c,j} = u_j$
- Computing the j th node of the c th output word via softmax function:

$$p(w_{c,j} = w_{O,c} | w_I) = y_{c,j} = \frac{\exp(u_{c,j})}{\sum_{j'=1}^V \exp(u_{j'})}$$

→ This is the probability that the output of the j th node of the c th output is equal to the actual value of the j th index of the c th output vector (one-hot encoded)

- This way the forward pass is done so next the weights W and W' need to be learned with backpropagation and stochastic gradient descent.

SG Calculations

- First, a loss function is defined as:

Negative log likelihood

→ objective is to minimize it

$$\begin{aligned}
 E &= -\log p(w_{O,1}, w_{O,2}, \dots, w_{O,C} | w_I) \\
 &= -\log \prod_{c=1}^C \frac{\exp(u_{c,j_c^*})}{\sum_{j'=1}^V \exp(u_{j'})} \\
 &= -\sum_{c=1}^C u_{j_c^*} + C \cdot \log \sum_{j'=1}^V \exp(u_{j'})
 \end{aligned}$$

de htl3 -1, 3nd el
j el s7, gher keda 0

The probability of the output words (context words) given the input (target/middle) word

j_c^* is the index of the c th output

- Take the derivative w.r.t. $u_{c,j}$:

$$\frac{\partial E}{\partial u_{c,j}} = y_{c,j} - t_{c,j}$$

$t_{c,j}=1$ if the j th node of the c th true output word is equal to 1 and 0 other wise.

How is this computed??

- Compute the derivative w.r.t. W' :

$$\begin{aligned}
 \frac{\partial E}{\partial w'_{ij}} &= \sum_{c=1}^C \frac{\partial E}{\partial u_{c,j}} \cdot \frac{\partial u_{c,j}}{\partial w'_{ij}} \\
 &= \sum_{c=1}^C (y_{c,j} - t_{c,j}) \cdot h_i
 \end{aligned}$$

$u = h \cdot w$

Using the chain rule

- The gradient descent update equation for W' :

$$w_{ij}^{(new)} = w_{ij}^{(old)} - \eta \cdot \sum_{c=1}^C (y_{c,j} - t_{c,j}) \cdot h_i$$

SG Calculations

- Computing for W, begin with taking the derivative of the error w.r.t. h_i :

$$\begin{aligned}\frac{\partial E}{\partial h_i} &= \sum_{j=1}^V \frac{\partial E}{\partial u_j} \cdot \frac{\partial u_j}{\partial h_i} \quad \text{Using the chain rule} \\ &= \sum_{j=1}^V \sum_{c=1}^C (y_{c,j} - t_{c,j}) \cdot w'_{ij}\end{aligned}$$

$u = hw$

- Compute the derivative w.r.t. W:

$$\begin{aligned}\frac{\partial E}{\partial w_{ki}} &= \frac{\partial E}{\partial h_i} \cdot \frac{\partial h_i}{\partial w_{ki}} \quad \text{Using the chain rule} \\ &= \sum_{j=1}^V \sum_{c=1}^C (y_{c,j} - t_{c,j}) \cdot w'_{ij} \cdot x_k\end{aligned}$$

- The gradient descent update equation for W:

$$w_{ji}^{(new)} = w_{ji}^{(old)} - \eta \cdot \sum_{j=1}^V \sum_{c=1}^C (y_{c,j} - t_{c,j}) \cdot w'_{ij} \cdot x_j$$

$$[V \times N] = [V \times N] - [V \times 1] ([N \times V] [V \times 1])^T$$

$$W_{input}^{(new)} = W_{input}^{(old)} - \eta \cdot x \cdot (W_{output} \sum_{c=1}^C e_c)^T$$

- Matrix forms:

$$[N \times V] = [N \times V] - [N \times 1] [1 \times V] \quad W_{output}^{(new)} = W_{output}^{(old)} - \eta \cdot h \cdot \sum_{c=1}^C e_c$$

Each gradient descent update requires a **summation over the entire vocabulary** → **Computationally expensive**. So techniques such as hierarchical softmax and negative sampling are used to make this computation more efficient.

Other Kinds of Static Embeddings



FastText

GloVe

Sphere

FastText

- An extension of word2vec developed by *Piotr Bojanowski et al. at Facebook*.
- Addresses a problem with word2vec “**unknown words**”—words that appear in a test corpus but were unseen in the training corpus.
- A related problem is **word sparsity**, such as in languages with rich morphology, where some of the many forms for each noun and verb may only occur rarely.
- FastText deals with these problems by using **subword models**, representing each word as **itself plus a bag of constituent n-grams**, with special boundary symbols < and > added to each word.
- For example, with $n = 3$ the word *where* would be represented by the sequence <where> plus the character n-grams: <wh, whe, her, ere, re>
 - Note that the sequence <her>, corresponding to the word *her* is different from the tri-gram *her* from the word *where*.
- Now, the vocabulary is the union of the subwords of all words (subwords refer to the words themselves + n-grams). Then a skip-gram/cbow embedding is learned for each subword.
- The vector of a word is the sum of the whole word vector in addition to the sum of its subwords' vectors.
- Unknown words can then be presented only by the sum of the constituent n-grams.

GloVe

- Short for Global Vectors, because the model is based on capturing global corpus statistics developed by *Jeffrey Pennington et al. at Stanford University*.
- Unlike word2vec that captures local statistics of a corpus, GloVe captures both local and global statistics of a corpus.
 - In word2vec, the vector learnt for a certain word is only affected by the surrounding context words.
 - In GloVe, the global statistics of the corpus are considered as it operates on **co-occurrence matrices** that are built across the entire corpus.
- GloVe is based on **ratios of probabilities** from the word-word co-occurrence matrix.
- GloVe is a global log-bilinear regression model.

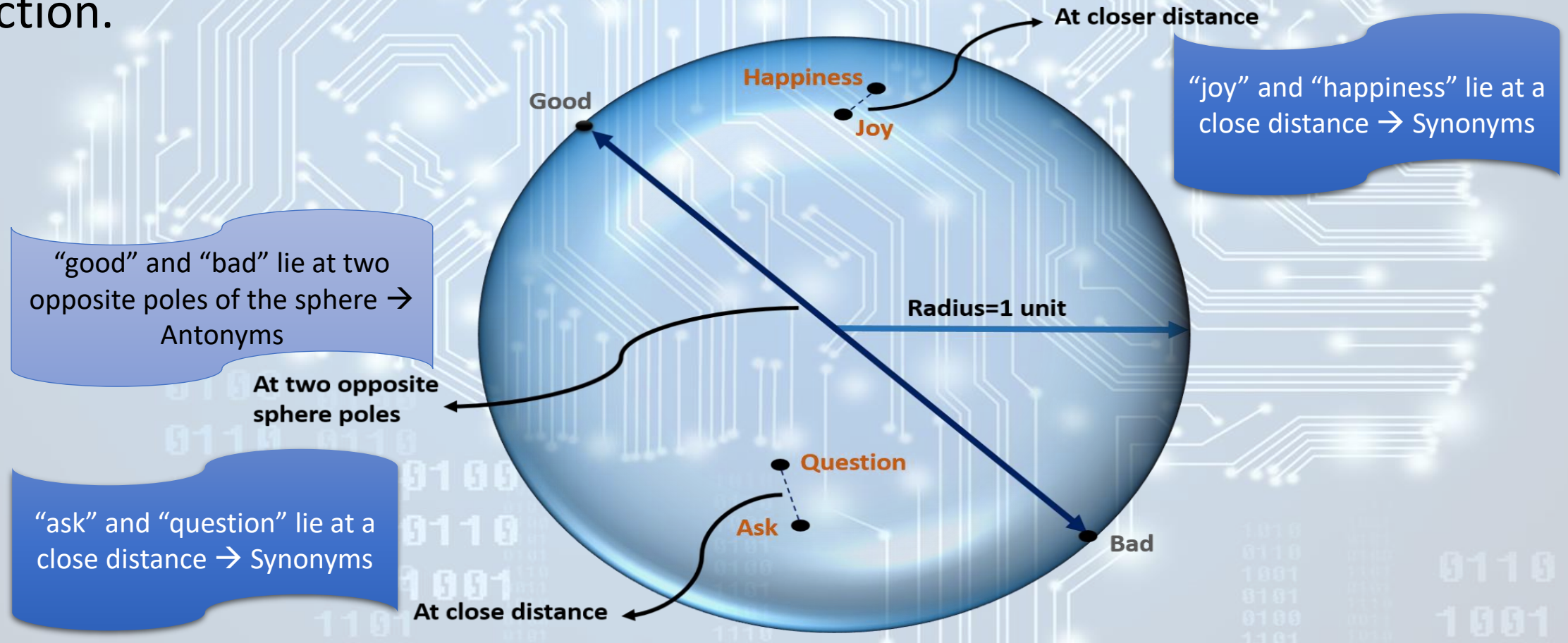
Sphere

- The approaches such as word2vec, fastText and GloVe are **distributional** approaches:
 - Antonyms (*good, bad*) can lie in close proximity (**cosine similarity close to 1**)
→ as antonyms can often occur in similar contexts.
 - Synonyms (*happiness, joy*) lie in close proximity (**cosine similarity close to 1**).
 - Unrelated words (*car, apple*) lie far away (**cosine similarity close to 0**).
 - Other antonyms can lie far away (**cosine similarity close to 0**).

There is **no clear distinction** between the different semantic relations among words such as synonyms, antonyms and unrelated words.

Sphere

- Sphere approach developed by *Sandra Rizkallah et al. at Cairo University* solves this problem by embedding the vectors of opposite or antonyms words at opposite poles of the sphere (**cosine similarity close to -1**).
- Sphere is a semi-supervised relaxation algorithm that optimizes a devised error function.



Evaluating Vector Models

- The most important evaluation metric for vector models is **extrinsic** evaluation on tasks.
- It is also useful to have **intrinsic** evaluations:
 1. **Similarity**: computing the correlation between an algorithm's word similarity scores and word similarity ratings assigned by humans.
 2. **Semantic Textual Similarity**: evaluates the performance of sentence-level similarity algorithms, consisting of a set of pairs of sentences, each pair with human-labeled similarity scores.
 3. **Analogy**: the system has to solve problems of the form "*a is to b as c is to ____*" e.g.: "*Cairo is to Egypt as Rome is to ____*" \rightarrow "*Italy*". Need to find *d* such that the associated word vectors *va*, *vb*, *vc*, *vd* are related to each other in the following relationship: " *$vb - va = vd - vc$* " where the *vd* is obtained to be the vector with highest cosine similarity to the vector (*vb*-*va*+*vc*).



Thank You