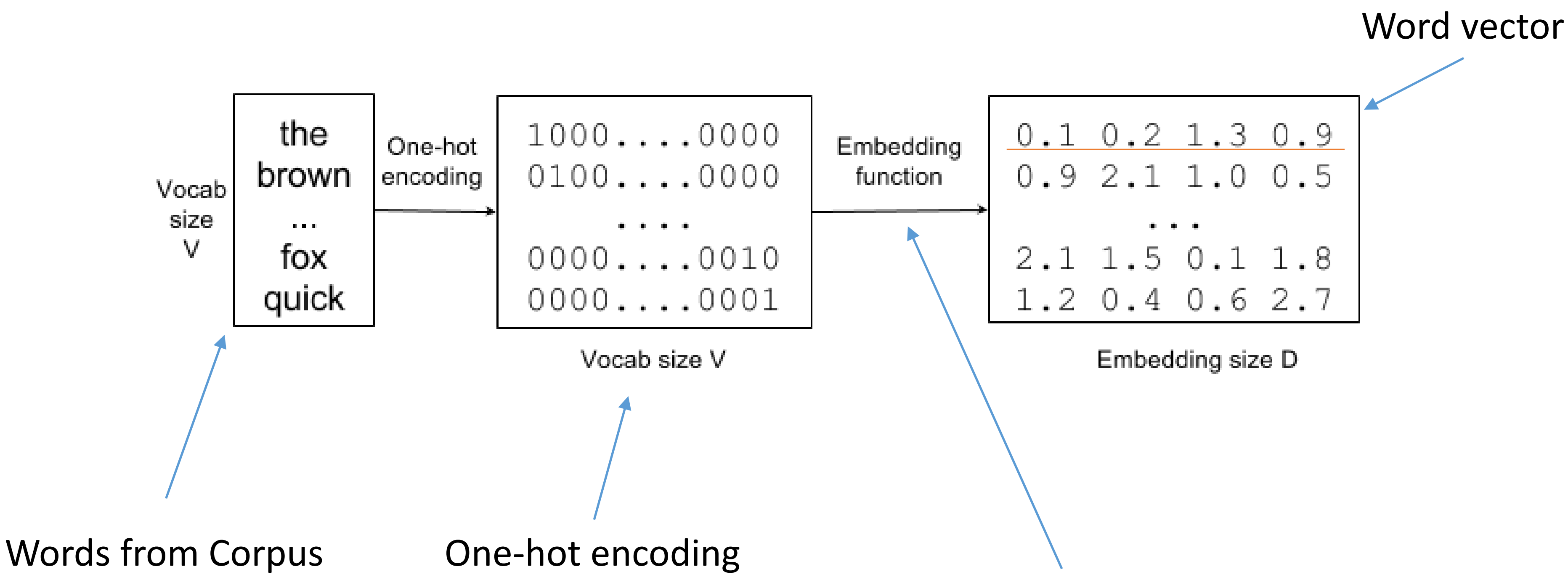
Word embedding

CBOW & SKIP-GRAM

Word2vec encoding



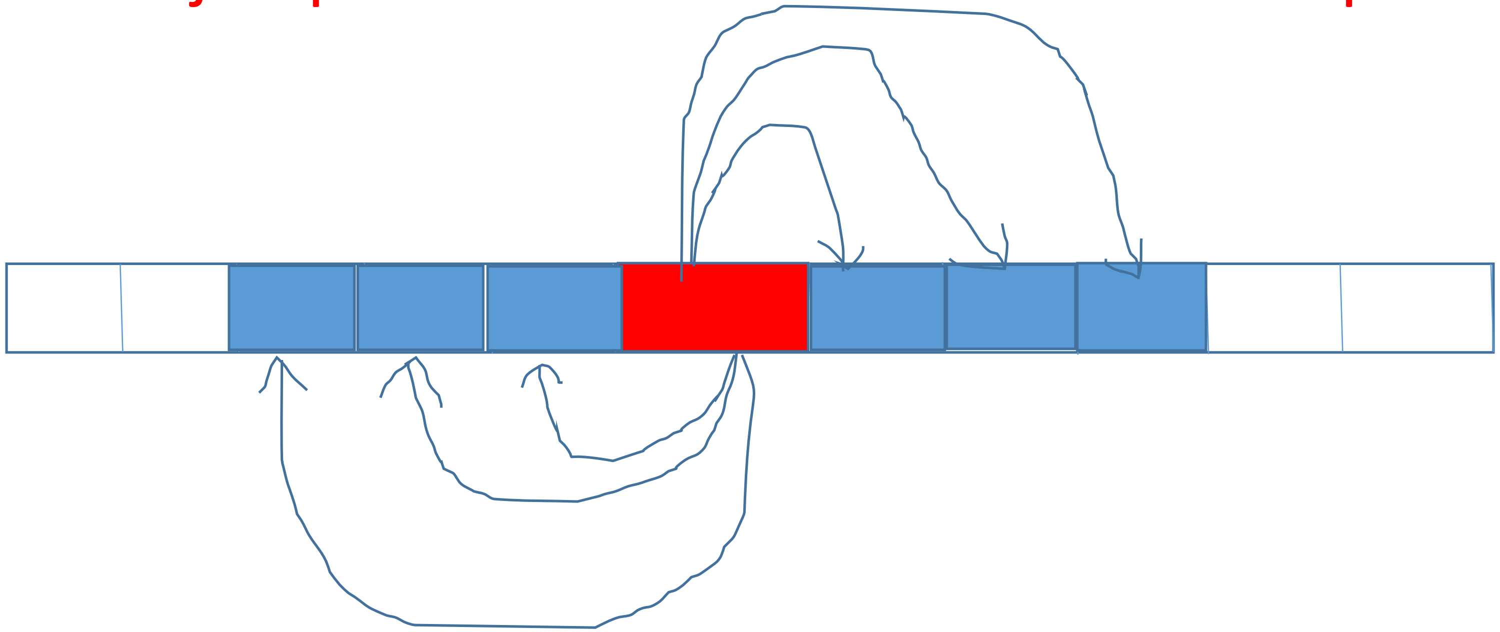
Use corpus, (word usage patterns), to train the Embedding function.

Need for word encoding

- Neural network can read only numeric inputs, not word input
- One-hot encoding wastes space, too many words
- One-hot encoding does not consider usage pattern, or similar usage patterns of of words

Jump and leap should have similar codes

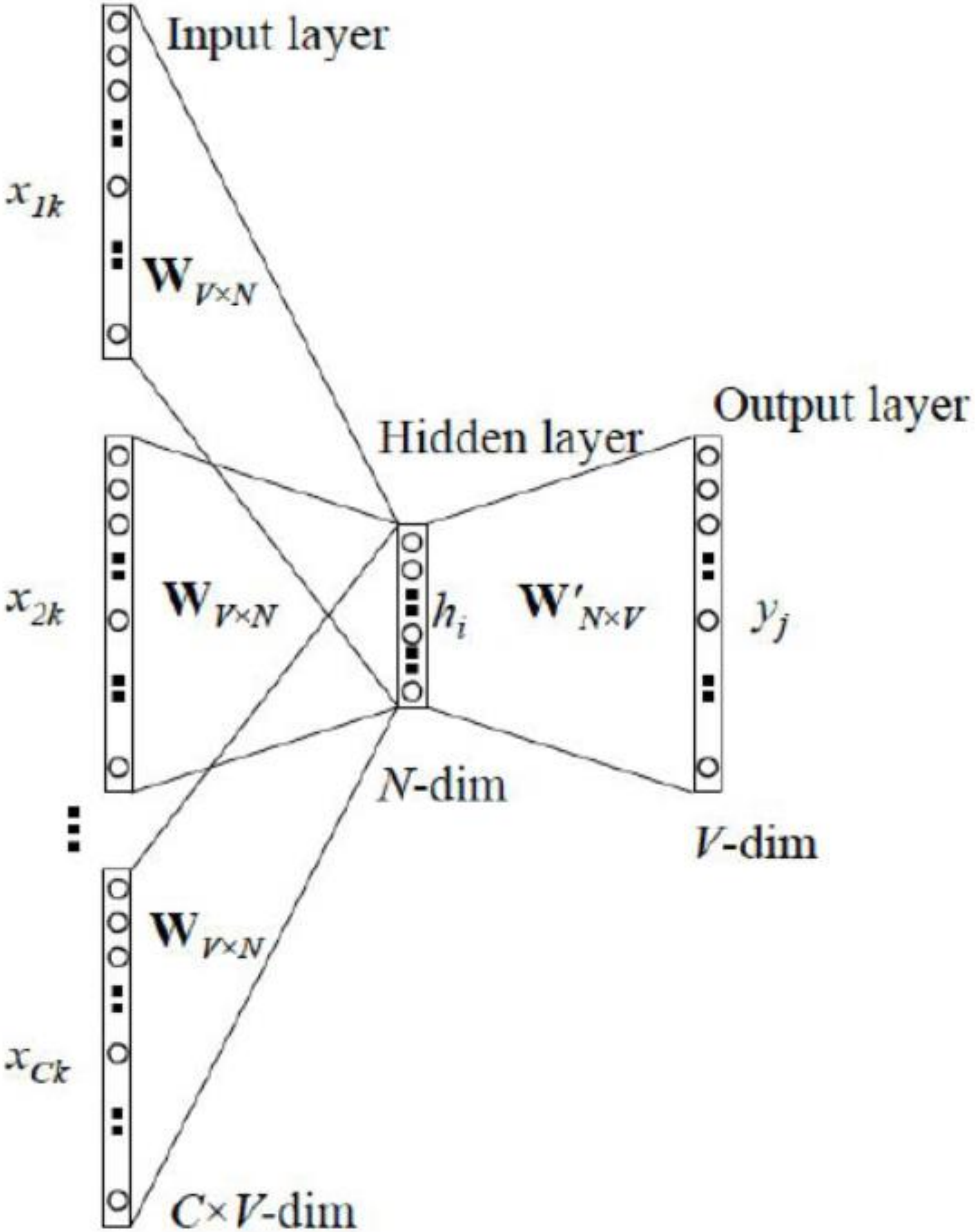
- Brown fox jumps over the dog   Brown fox leaps over the dog
- The boy jumps over the fence   The boy leaps over the fence
- The man jumps over the pothole the man leaps over the pothole
- The rabbit jumps over the tortoise the rabbit leaps over the tortoise



Continuous Bag of Words (CBOW)--Mikolov

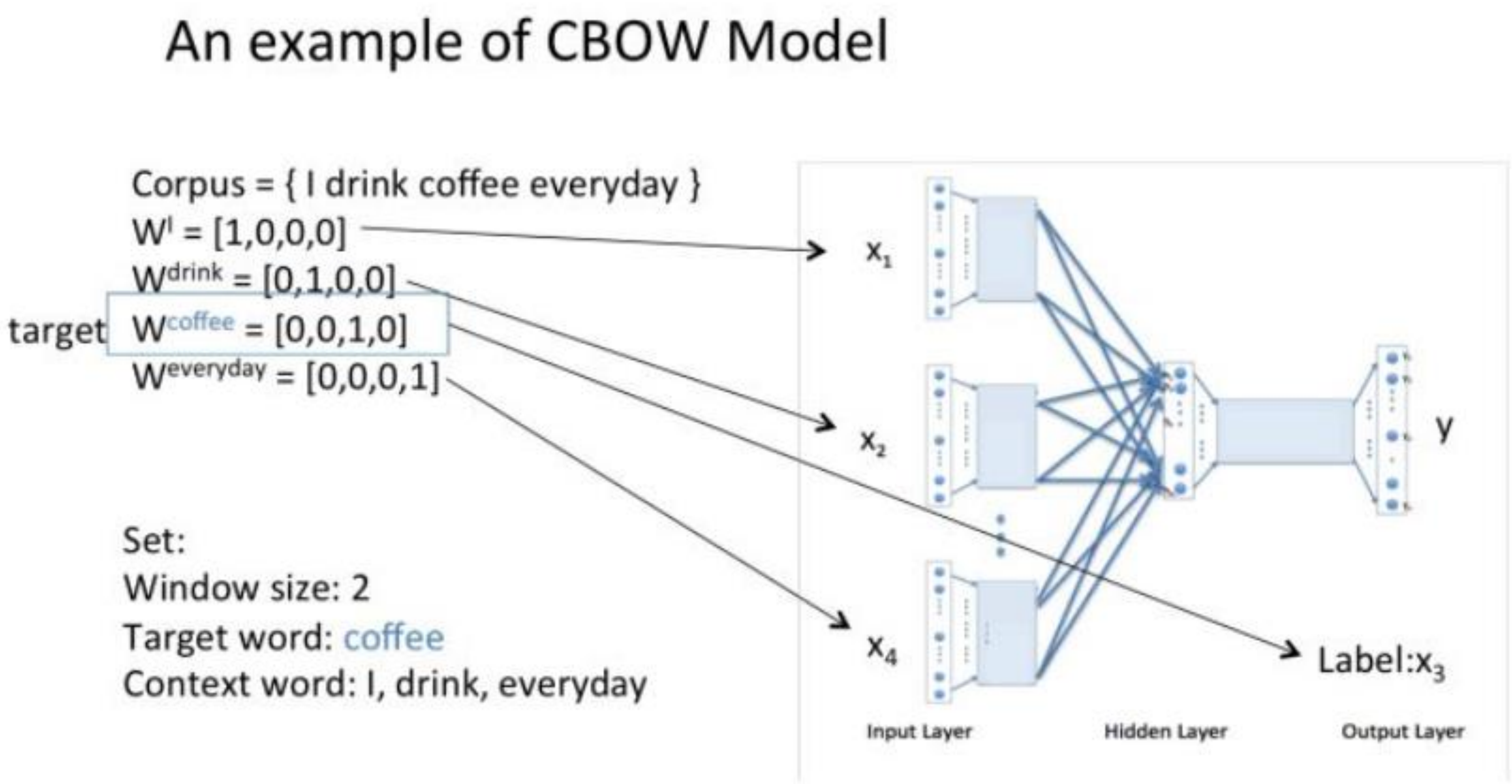
- Ref: Stanford “Deep learning for NLP”
- Encoding is based on “Predicting a center word from the surrounding context”
- Surrounding context–words in the window centered by the target word, note: it’s a bag not a set!
- It is best suited for a small vocabulary

CBOW



Input layer
x_{1k}
$\mathbf{W}_{V\times N}$
Hidden layer
Output layer
x_{2k}
$\mathbf{W}_{V\times N}$
h_i
$\mathbf{W}'_{N\times V}$
y_j
N-dim
V-dim
x_{Ck}
$\mathbf{W}_{V\times N}$
C×V-dim

Illustrating example (target word: “coffee”)



Corpus = { I drink coffee everyday }

Initialize:

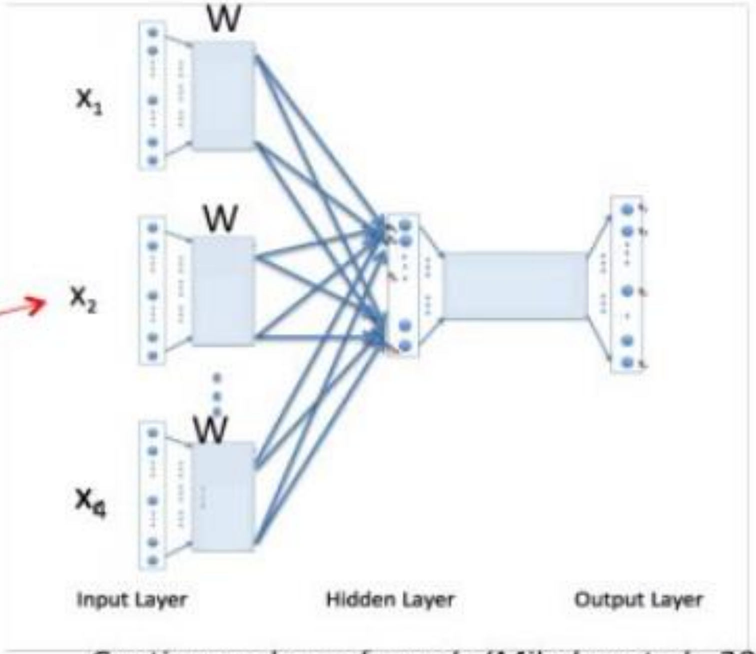
$$W = \begin{bmatrix} 1 & 2 & 3 & 0 \\ 1 & 2 & 1 & 2 \\ -1 & 1 & 1 & 1 \end{bmatrix}$$

Ex:

$W^{drink} = [0,1,0,0]$

$$Wx_2 = v_2$$

$$\begin{bmatrix} 1 & 2 & 3 & 0 \\ 1 & 2 & 1 & 2 \\ -1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 2 \\ 2 \\ 1 \end{bmatrix}$$



Continuous bag-of-words (Mikolov et al., 2013)

An example of CBOW Model

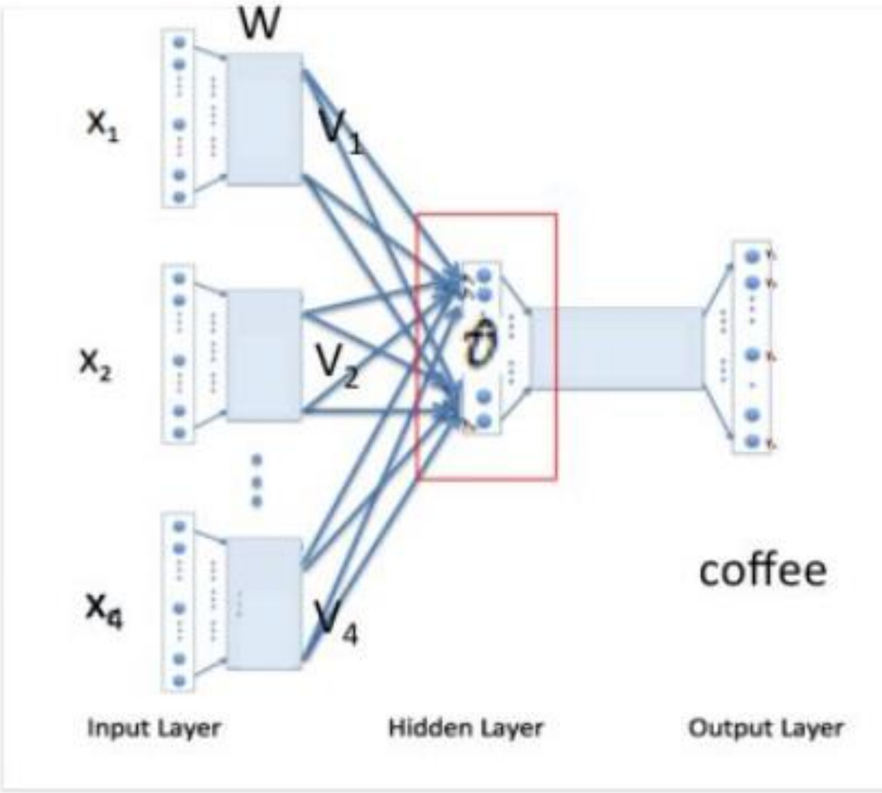
Corpus = { I drink coffee everyday }

Initialize:

$$W = \begin{bmatrix} \boxed{\begin{matrix}1\\1\\-1\end{matrix}} & \boxed{\begin{matrix}2\\2\\1\end{matrix}} & \begin{matrix}3\\1\\1\end{matrix} & \boxed{\begin{matrix}0\\2\\1\end{matrix}} \end{bmatrix}$$

$$\frac{V_1 + V_2 + V_4}{3} = \hat{v}$$

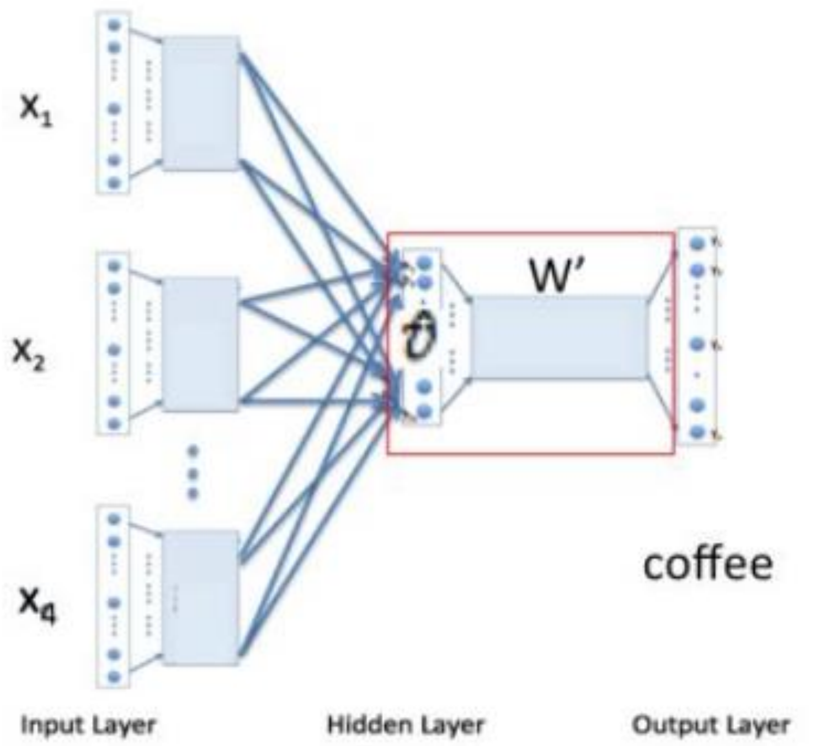
$$\frac{1}{3}\left(\begin{bmatrix}1\\1\\-1\end{bmatrix}+\begin{bmatrix}2\\2\\1\end{bmatrix}+\begin{bmatrix}0\\2\\1\end{bmatrix}\right)=\boxed{\begin{bmatrix}1\\1.67\\0.33\end{bmatrix}}$$



Corpus = { I drink coffee everyday }

Initialize:

$$W' = \begin{bmatrix} 1 & 2 & -1 \\ -1 & 2 & -1 \\ 1 & 2 & 2 \\ 0 & 2 & 0 \end{bmatrix}$$



$$W'\hat{v} = U_0$$

$$\begin{bmatrix} 1 & 2 & -1 \\ -1 & 2 & -1 \\ 1 & 2 & 2 \\ 0 & 2 & 0 \end{bmatrix} \begin{bmatrix} 1.00 \\ 1.67 \\ 0.33 \end{bmatrix} = \begin{array}{c c} \begin{bmatrix} 4.01 \\ 2.01 \\ 5.00 \\ 3.34 \end{bmatrix} & \begin{array}{c} u_1 \\ u_2 \\ u_c \\ u_4 \end{array} \end{array}$$

An example of CBOW Model

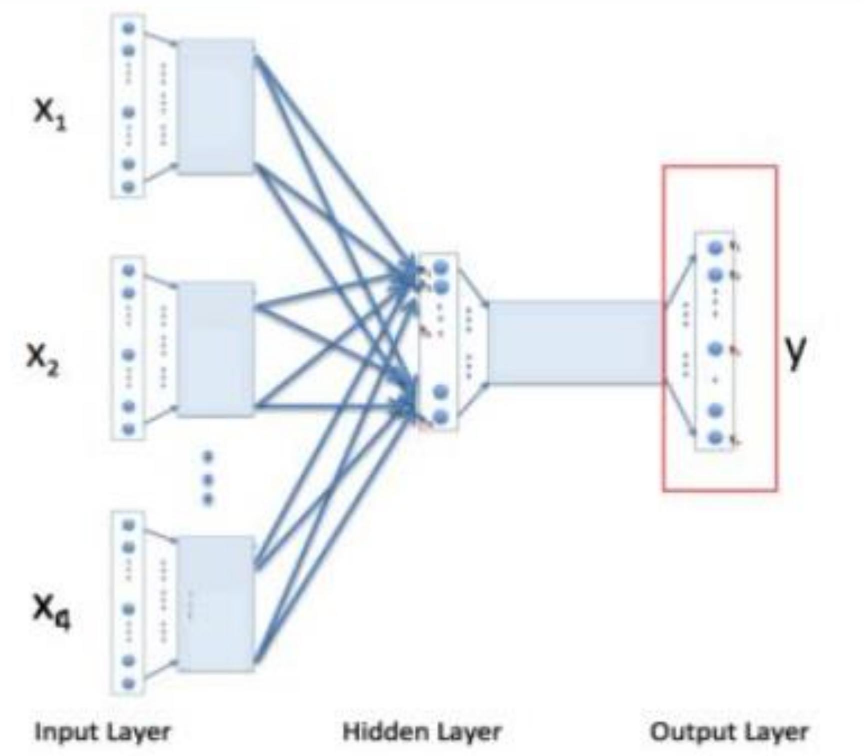
Output: Probability distribution

$$softmax(\mathbf{u}_o) = \mathbf{y}$$

$$softmax\left(\begin{bmatrix} 4.01 \\ 2.01 \\ 5.00 \\ 3.34 \end{bmatrix}\right) = \boxed{\begin{bmatrix} 0.23 \\ 0.03 \\ 0.62 \\ 0.12 \end{bmatrix}}$$

Probability of "coffee"

We desire probability generated to match the true probability(label) x_3 [0,0,1,0]
Use gradient descent to update W and W'



Notation for CBOW

- w_i: word i from vocabulary V
- $\mathcal{V} \in R^{n\times|V|}$: Input word matrix
- v_i:i-th column of $\mathcal{V}$, the input vector representation of word w_i
- $\mathcal{U} \in R^{|V|\times n}$: output word matrix
- u_i: i-th row of $\mathcal{U}$, the output vector representation of word w_i

- We generate our one hot word vectors ($x^{(c-m)}$, . . . , $x^{(c-1)}$, $x^{(c+1)}$, . . . , $x^{(c+m)}$) for the input context of size m.
- We get our embedded word vectors for the context($v_{c-m} = \mathcal{V}x^{(c-m)}, v_{c-m+1} = \mathcal{V}x^{(c-m+1)}, \ldots, v_{c+m} = \mathcal{V}x^{(c+m)}$)
- Average these vectors to get $\hat{v} = \frac{v_{c-m}+v_{c-m+1}+\cdots+v_{c+m}}{2m}$
- Generate a score vector $z = \mathcal{U}\hat{v}$
- Turn the scores into probabilities $\hat{y} = softmax(z)$
- We desire our probabilities generated, $\hat{y}$, to match the true probabilities,

y, which also happens to be the one hot vector of the actual word.

- Cross-entropy for cost function:

$$H(\hat{y}, y) = -\sum_{j=1}^{|V|} y_i \log(\hat{y}_i)$$

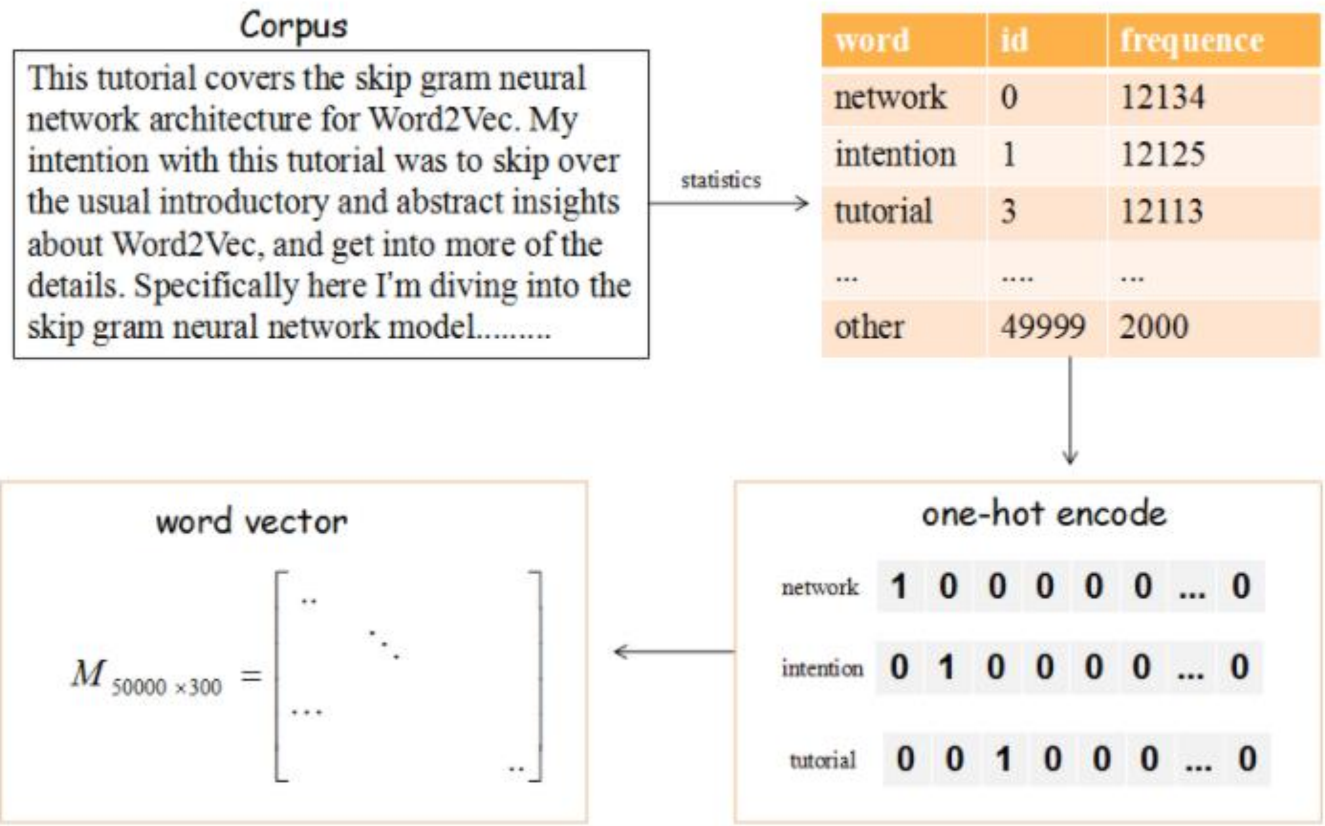
- Since one-hot encoding, let c is the index of the one:

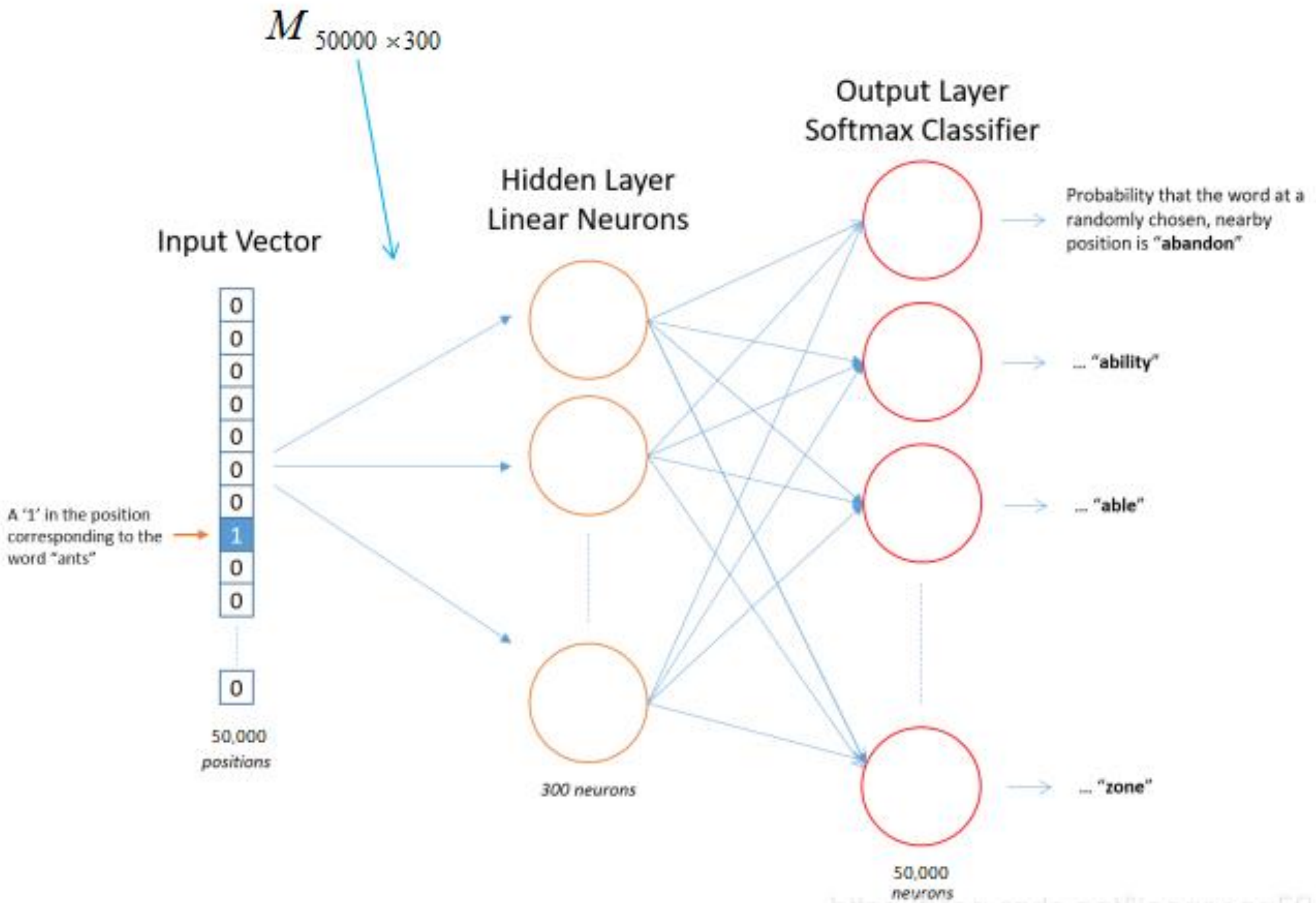
$$H(\hat{y}, y) = -y_c \log(\hat{y}_c)$$

- Objective function:

$$\begin{aligned}
\text{minimize } J &= -\log P(w_c | w_{c-m}, \ldots, w_{c-1}, w_{c+1}, \ldots, w_{c+m}) \\
&= -\log P(u_c | \hat{v}) \\
&= -\log \frac{\exp(u_c^T \hat{v})}{\sum_{j=1}^{|V|} \exp(u_j^T \hat{v})} \\
&= -u_c^T \hat{v} + \log \sum_{j=1}^{|V|} \exp(u_j^T \hat{v})
\end{aligned}$$

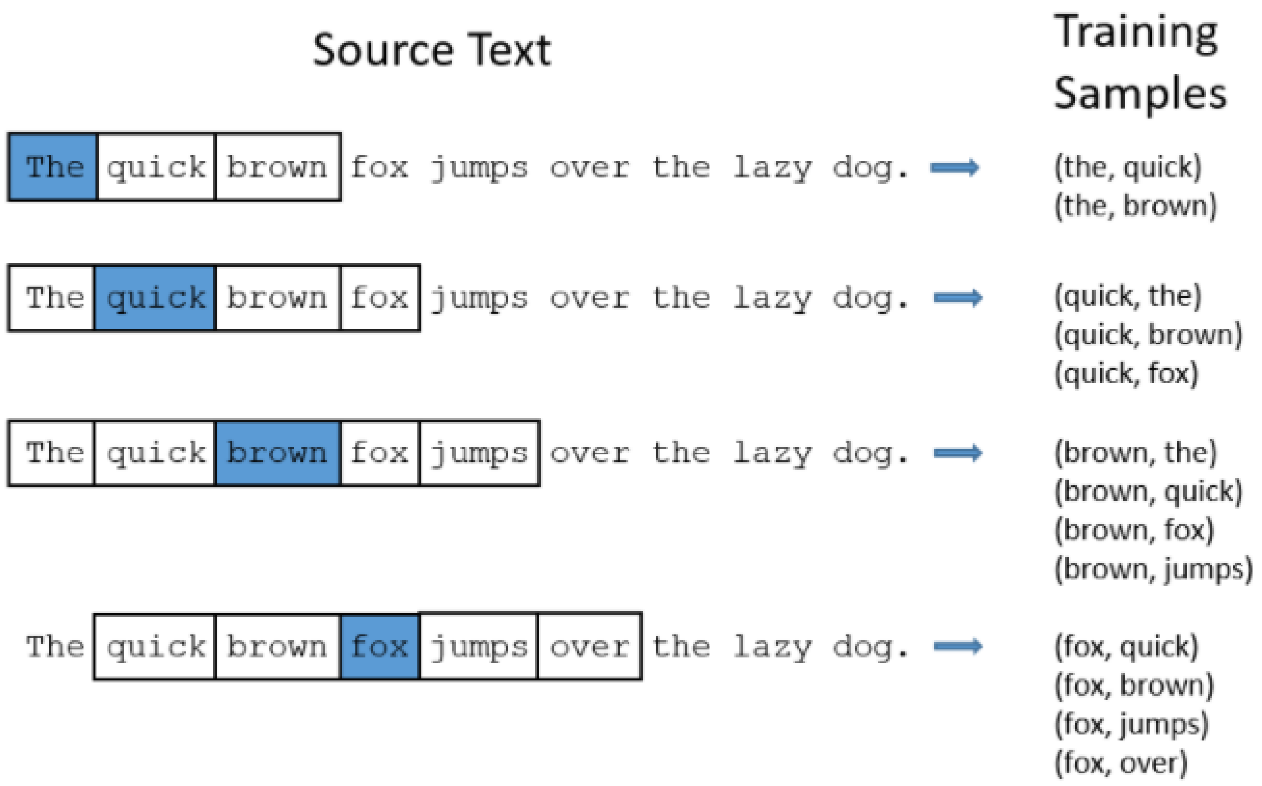
SKIP-GRAM


Corpus
This tutorial covers the skip gram neural network architecture for Word2Vec. My intention with this tutorial was to skip over the usual introductory and abstract insights about Word2Vec, and get into more of the details. Specifically here I'm diving into the skip gram neural network model.........
statistics
word id frequence
network 0 12134
intention 1 12125
tutorial 3 12113
... .
other 49999 2000
one-hot encode
network 1 0 0 0 0 0 ... 0
intention 0 1 0 0 0 0 ... 0
tutorial 0 0 1 0 0 0 ... 0
word vector
M 50000 ×300 =



$M_{50000 \times 300}$
Input Vector
Hidden Layer
Linear Neurons
Output Layer
Softmax Classifier
0
0
0
0
0
0
0
1
0
0
0
A '1' in the position corresponding to the word "ants"
50,000 positions
300 neurons
50,000 neurons
Probability that the word at a randomly chosen, nearby position is "abandon"
... "ability"
... "able"
... "zone"

If the window size equals 2, then take two words from the left and right of the target word.



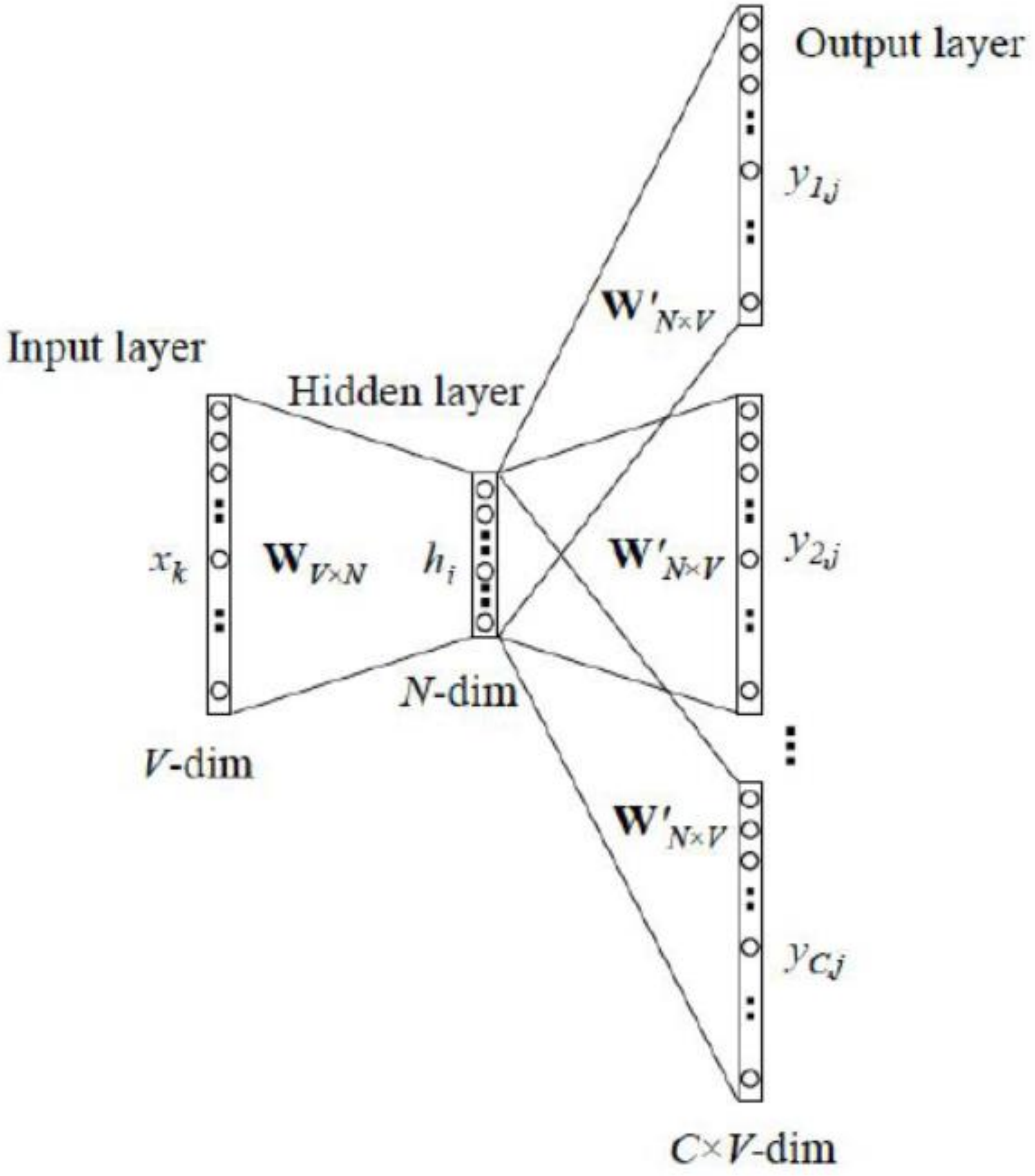
- For the above Word2Vec, the size of vocabulary is 50,000, the word vector is of dim 300.
- The embedding parameter to the keras is input_dim=50,000, output_dim=300
- For a simple example, output_dim=2, [4], [2] => [[0.25, 0.1], [0.6, 0.2]]

- If the id of “tom” is 4, then the word vector of “tom” is the column 4, and the value is [0.25, 0.1]

Skip-gram

Use the center word to predict the Context words, minimize the Prediction errors.

The training objective is to learn word vector representations that are good at predicting the nearby words. -- Mikolov



We breakdown the way this model works in these 6 steps:

1. We generate our one hot input vector x
2. We get our embedded word vectors for the context $v_c = \mathcal{V}x$
3. Since there is no averaging, just set $\hat{v} = v_c$?
4. Generate $2m$ score vectors, $u_{c-m}, \ldots, u_{c-1}, u_{c+1}, \ldots, u_{c+m}$ using $u = \mathcal{U}v_c$
5. Turn each of the scores into probabilities, $y = \text{softmax}(u)$
6. We desire our probability vector generated to match the true probabilities which is $y^{(c-m)}, \ldots, y^{(c-1)}, y^{(c+1)}, \ldots, y^{(c+m)}$, the one hot vectors of the actual output.

$$
\begin{aligned}
\text{minimize } J &= -\log P(w_{c-m}, \ldots, w_{c-1}, w_{c+1}, \ldots, w_{c+m} | w_c) \\
&= -\log \prod_{j=0, j\neq m}^{2m} P(w_{c-m+j} | w_c) \\
&= -\log \prod_{j=0, j\neq m}^{2m} P(u_{c-m+j} | v_c) \\
&= -\log \prod_{j=0, j\neq m}^{2m} \frac{\exp(u_{c-m+j}^T v_c)}{\sum_{k=1}^{|V|} \exp(u_k^T v_c)} \\
&= -\sum_{j=0, j\neq m}^{2m} u_{c-m+j}^T v_c + 2m \log \sum_{k=1}^{|V|} \exp(u_k^T v_c)
\end{aligned}
$$