Word Embedding

Word Embedding : method of vector representation of textual data.

Word2vec, Glove, Fasttext – word embedding techniques.

- **T**akes as its input a **large corpus of text** and produces a **vector space**, typically of several hundred dimensions, with **each unique word** in the corpus being **assigned a corresponding vector** in the space.
- **Word vectors** are positioned in the **vector space** such that **words that share common contexts** in the corpus are **located in close proximity** to one another **in the space.**
- This technique is based on the idea that **words that occur** in the **same contexts** tend to have **similar meanings**.
- Following is an intuitive way to understand the word2vec embedding (numerical representation of contextual similarity)

Word2Vec (CBOW)

CBOW - Predicting a center **word** form the **surrounding context**
(context could be single or group of words)
(input & output both are one-hot coded, output layer is softmax layer to to total sum of probabilities to 1. weights from hidden to output layer would be used as word vector representation)

Example:
One approach is to treat **{“the", "cat", ’sits", “on’, “the"}** as a context and from these words, CBOW be able to **predict** the word **“mat"**

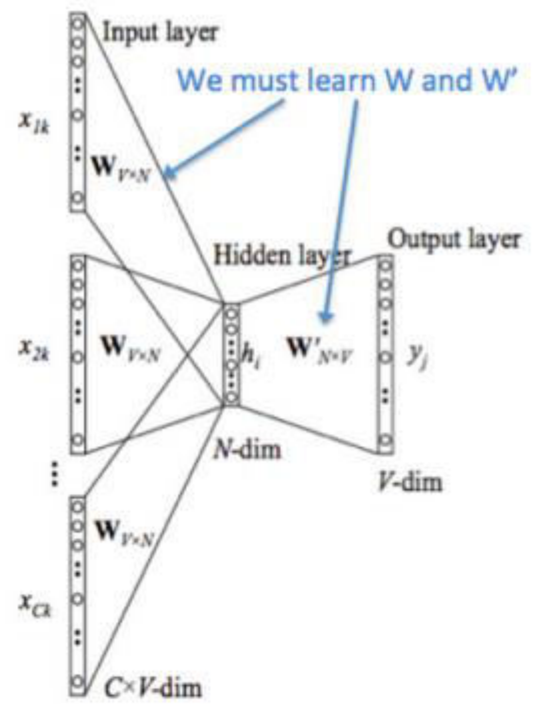


Figure 1: This image demonstrates how CBOW works and how we must learn the transfer matrices

We breakdown the way this model works in these steps:

1. We generate our one hot word vectors $(x^{(c-m)}, \ldots, x^{(c-1)}, x^{(c+1)}, \ldots, x^{(c+m)})$ for the input context of size m.
2. We get our embedded word vectors for the context ($v_{c-m} = \mathcal{V}x^{(c-m)}, v_{c-m+1} = \mathcal{V}x^{(c-m+1)}, \ldots, v_{c+m} = \mathcal{V}x^{(c+m)}$)
3. Average these vectors to get $\hat{v} = \frac{v_{c-m}+v_{c-m+1}+\ldots+v_{c+m}}{2m}$
4. Generate a score vector $z = \mathcal{U}\hat{v}$
5. Turn the scores into probabilities $\hat{y} = \text{softmax}(z)$
6. 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.

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

Word2Vec (CBOW)

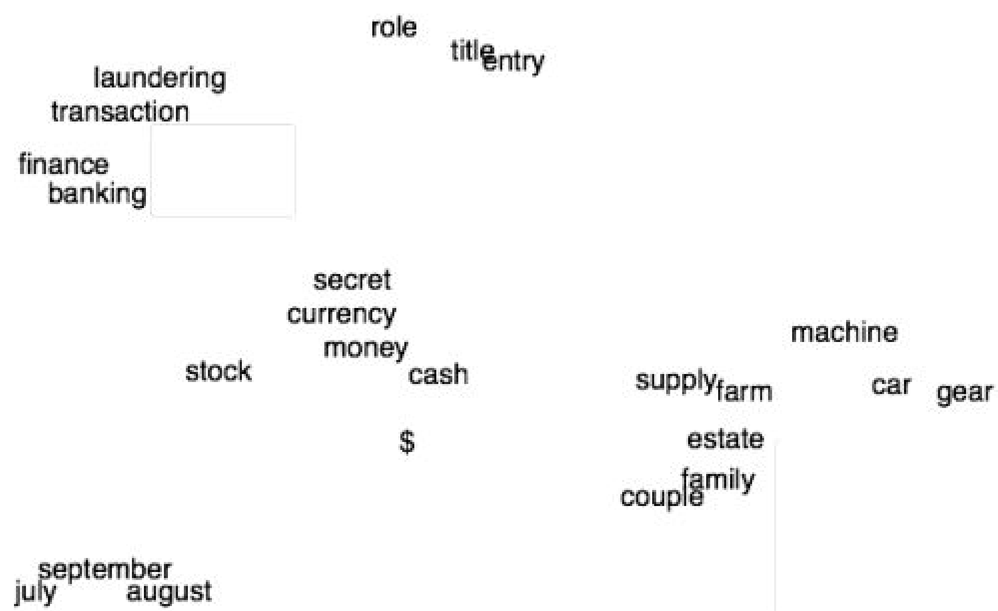
Final **Word Vectors** is a **dense vector** for each word type are numerical **representations of contextual similarities** between words.

Word meaning as a vector

The result is a dense vector for each word type, chosen so that it is good at predicting other words appearing in its context

… those other words also being represented by vectors

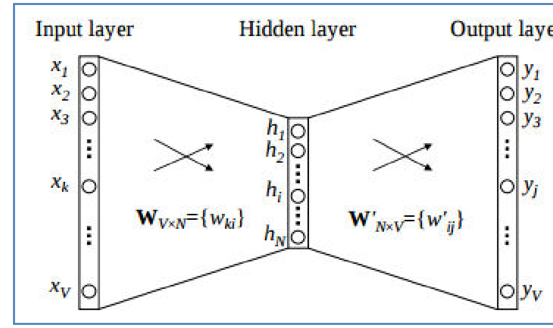
$$\textit{currency} = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{pmatrix}$$



Word2Vec – CBOW

Example: Corpus = "the cat sits on the mat"
Single context word (context window = 1)
Pair of (Context word, Target word)

Input	Output	:	'the'	'cat'	'sits'	'on'	'the'	'mat'
'the'	'cat'		1	0	0	0	0	0
'cat'	'the'		0	1	0	0	0	0
'sits'	'cat'		0	0	1	0	0	0
'sits'	'on'		0	0	1	0	0	0

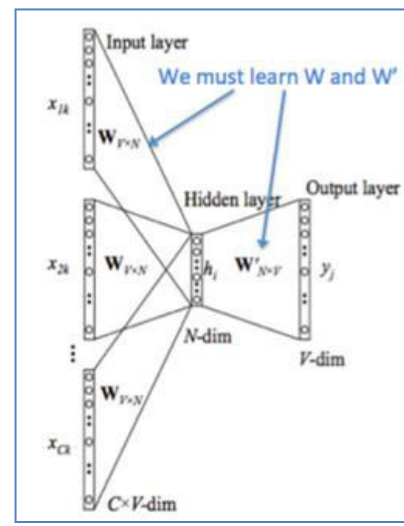


Single context CBOW

(input layer Nodes = total number of unique word)
(hidden layer size = Vector dimensions that word need to be converted
(output layer is the softmax function, size = total number of unique word)
(weights between hidden layer to output layer will be used as word vectors)

Corpus = "the cat sits on the mat"
Multi context word (context window = 2)

Input	Output	:	'the'	'cat'	'sits'	'on'	'the'	'mat'
['the', 'sits']	'cat '		1	0	1	0	0	0
['cat', 'on']	'sits'		0	1	0	1	0	0
['sits' , 'the']	'on'		0	0	1	0	1	0
['on' , 'mat']	'the'		0	0	0	1	0	1

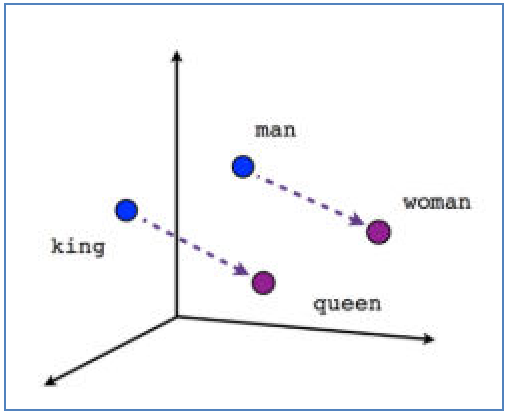


Word2Vec – CBOW

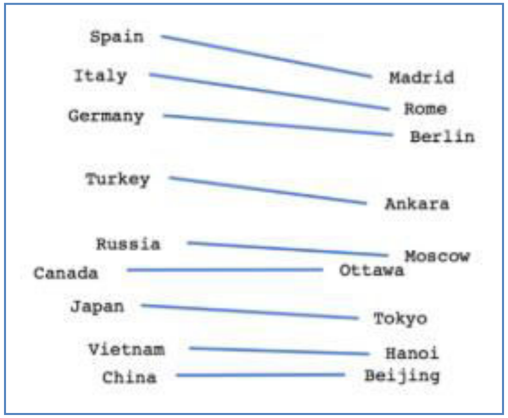
In **Word2vec**, gradient descent optimization will keep updating the word embedding until the model is successfully discriminating the real words from noise words.

Learned word vectors capture the **semantic information about words** and their **relationship with one another.**

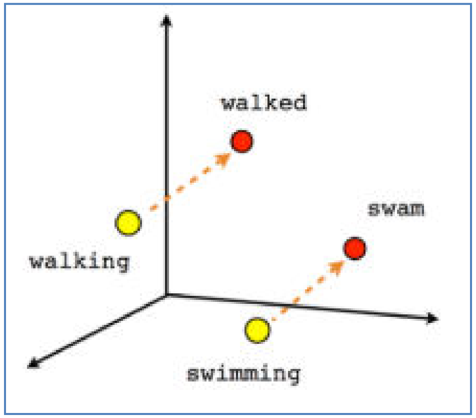
Followings are the word vectors visualization in lower dimensional space showing **certain semantic relationship between words.**



Male-Female



Country-Capital



Verb-Tense

Word2Vec(SKIP-Gram)

Skip-Gram - Predicting **surrounding context words** given a **center word**

Example:
Given the **center word** “mat", the model will be able to predict or generate the surrounding words “**the", "cat", “sits", “on", “the**".
Here we call the word **“mat"** the context.

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.

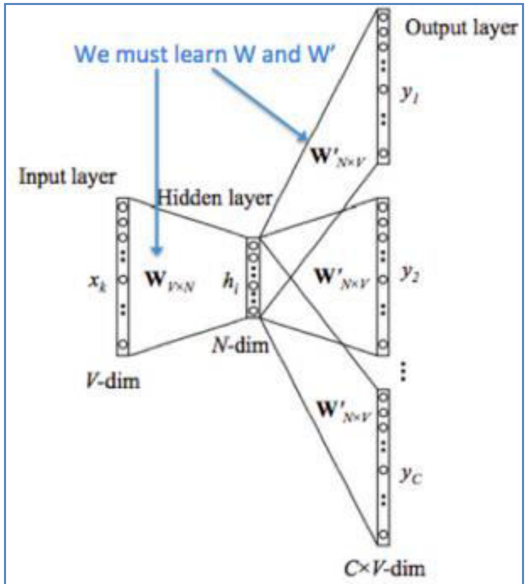


Figure 2: This image demonstrates how Skip-Gram works and how we must learn the transfer matrices

Final Word Vectors are numerical representations of contextual similarities between words.