Word Embedding using Word2Vec

Ramaseshan Ramachandran

Ramaseshan

GOAL

- Process each word in a Vocabulary of words to obtain a respective numeric representation of each word in the Vocabulary
- Reflect semantic similarities, Syntactic similarities, or both, between words they represent
- Map each of the plurality of words to a respective vector and output a single merged vector that is a combination of the respective vectors

CONTEXT WORDS AND CENTRAL WORD

$$P(w_{k+1}|\underbrace{w_{i-k}, w_{i-k+1}, \ldots, w_k}_{\text{Context words}})$$

- **Continuous Bag of Words (CBOW)** Models – A central word is surrounded by context words. Given the context words identify the central word
  - Wish you many $\boxed{\text{more}}\boxed{\text{happy}}\boxed{\text{returns}}\boxed{\text{of}}\boxed{\text{the}}$ day
- **Skip Gram Model**- Given the central word, identify the surrounding words
  - Wish you many $\boxed{\text{more}}\boxed{\text{happy}}\boxed{\text{returns}}\boxed{\text{of}}\boxed{\text{the}}$ day

CONTINUOUS BAG OF WORDS (CBOW)

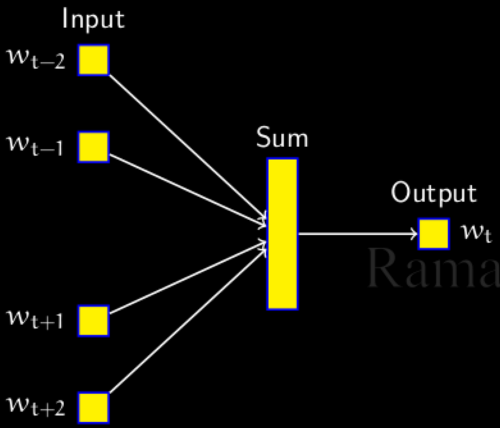


Figure: The CBOW architecture predicts the current word based on the context words of length n. Here the window size is 5

CBOW uses the sequence of words "Wish", "you", "a", "happy", "year" as a context and predicts or generates the central word "new"

- CBOW is used for learning the central word
- Maximize probability of word based on the word co-occurrences within a distance of n

SKIP GRAM MODEL

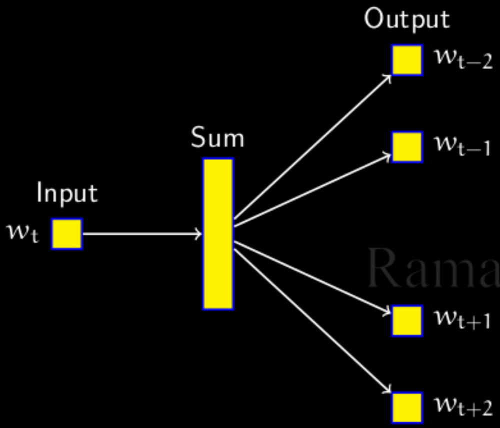


Figure: The SG architecture predicts the one context word at a time based on the center word. Here the window size is 5

SG uses the central word "new" and predicts the context words "Wish", "you", "a", "happy", "year"

- SG is used to learn the context words given the central word
- Maximize probability of word based on the word co-occurrences within a distance of $[-n, +n]$ from the center word

SOURCE PREPARATION FOR TRAINING

Source Text

Wish you many more happy returns of the day →

Wish you more happy returns of the day →

Wish you many more happy returns of the day →

Wish you many more happy returns of the day →

Wish you many more happy returns of the day →

Wish you many more happy returns of the day →

Wish you many more happy returns of the day →

Training Samples

(wish, you)
(wish, many)
(you, Wish)
(you, more), (you, happy)
(many, Wish), (many, you)
(many, more), (many, happy)
(more, many), (more, you)
(more, happy), (more, returns
(happy, many), (happy, more
(happy, returns), (happy, of)
(returns, more), (returns, happ
(returns, of), (returns, the)
(of, happy), (of, returns)
(of, the), (of, day)

ONE-WORD LEARNING

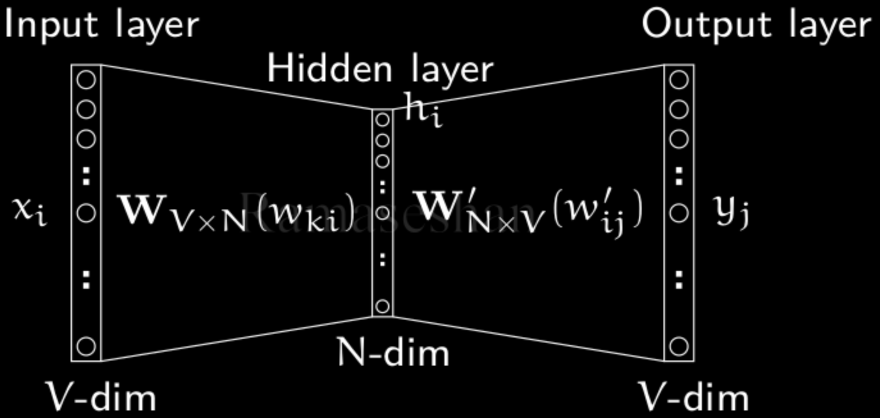


Figure: A CBOW model with only one word as input[1]. The layers are fully connected

Here the W and W' are learned

INPUT LAYER

$$t^{aback} = \begin{pmatrix} 0 \\ 1 \\ \dots \\ 0 \\ \dots \\ 0 \\ 0 \end{pmatrix} \dots t^{zoom} = \begin{pmatrix} 0 \\ 0 \\ \dots \\ 0 \\ \dots \\ 1 \\ 0 \end{pmatrix} t^{zucchini} = \begin{pmatrix} 0 \\ 0 \\ \dots \\ 0 \\ \dots \\ 0 \\ 1 \end{pmatrix}$$

$$x_k = 1 \text{ and } x'_k = 0, \forall k' \neq k$$

HIDDEN LAYER

This neural network is fully connected. Input to the network is a one-hot vector. W is the N-dimensional vector representation of the word, v_w^{T}, presented as input [2] [3].

$$\mathbf{h} = \mathbf{W}^{\mathbf{T}}\mathbf{X} \tag{1}$$

Now $v_{w\mathrm{I}}$ of the matrix (W) is the vector representation of the input one-hot vector w_{I}. From (1), h is a linear combination of input and weights.

In the same way. we get a score for $\mathrm{u_j}$

$$\mathrm{u_j} = \mathbf{v}'^{\mathbf{T}}_{\mathbf{w_j}}\mathbf{h} = \mathbf{v}'^{\mathbf{T}}_{\mathbf{w_j}}\mathbf{v}_{\mathbf{w_I}} \tag{2}$$

where $v_{w_{\mathrm{I}}}$ is the vector representation of the input word w_{I} and $v'_{w_{\mathrm{j}}}$ is the j^{th} column of (W')

OUTPUT LAYER

At the output layer, we apply the softmax to get the posterior distribution of the word(s). It is obtained by,

$$p(w_j|w_I) = y_j \tag{3}$$

where y_j is the output of the j^{th} unit in the output layer

$$y_j = \frac{\exp(u_j)}{\sum_{j'=1}^{V} \exp(u_j')} \tag{4}$$

$$= \frac{\exp(\mathbf{v}'^{\mathbf{T}}_{\mathbf{wj}} v_{wI})}{\sum_{j'=1}^{V} \exp((\mathbf{v}'^{\mathbf{T}}_{\mathbf{w_{j'}}}) v_{w_I})} \tag{5}$$

where $\mathbf{v}_w$, $\mathbf{v}'_w$ are the input vector (word vector) and output vector (feature vector) representations, of w_j and $w_{j'}$, respectively

UPDATE WEIGHTS - HIDDEN-OUTPUT LAYERS

The learning/training objective is to maximize (5) or minimize the error between the target and the computed value of the target which is $y_j^* - t$ and t is same as the input vector, in this case. We use cross-entropy as it provides us with a good measure of "error distance"

$$\max p(w_o|w_I) = \max(\log(y_j*)) \text{--Maximize} \quad (6)$$

$$-E = u_j - \log(y_{j*}) \text{--minimize} \quad (7)$$

$$= u_j * - \log \sum_{j'=1}^{V} \exp(u_j') \quad (8)$$

where

w_O is the output word and E is the loss function.

It is the special case of cross-entropy measurement between two probabilistic distributions u_{j*} and $u_{j'}$

- $\log p(x)$ is well scaled
- Selection of step size is easier
- With $p(x)$ multiplication may yield to near zero causing *underflow*
- For better optimization, $\log p(x)$ is considered (multiplication $\rightarrow$ addition)

UPDATE WEIGHTS (HO) - MINIMIZATION OF E

To minimize E,take the partial derivative of E with respect to j^{th} unit of u_j

$$\frac{\partial E}{\partial u_j} = y_j - t_j = e_j \tag{9}$$

where e_j is the prediction error. Taking partial derivative with respect to the hidden-output weights, we get,

$$\frac{\partial E}{\partial w'_{ij}} = \frac{\partial E}{\partial u_j} \cdot \frac{\partial u_j}{\partial w'_{ij}} = e_j \cdot h_i \tag{10}$$

Using the above equation (10),

$$w'^{\,new}_{ij} = w'^{\,old}_{ij} - \eta e_j \cdot h_i \text{ or} \tag{11}$$

$$\mathbf{v}^{(\mathbf{new})}_{\mathbf{w_j}} = \mathbf{v}'^{(\mathbf{old})}_{\mathbf{w_j}} - \eta e_j \cdot \mathbf{h} \quad \text{for } j = 1, 2, 3, \ldots, V \tag{12}$$

UPDATE INPUT TO HIDDEN WEIGHTS

Taking the derivative with respect to h_i, we get

$$\frac{\partial E}{\partial h_i} = \sum_{j=1}^{V} \frac{\partial E}{\partial u_j} \cdot \frac{\partial u_j}{\partial h_i} = \sum_{j=1}^{V} e_j \cdot w'_{ij} = \mathbf{EH_i} \tag{13}$$

Taking the derivative with respect to w_{ki}, we get

$$\frac{\partial E}{\partial w_{ki}} = \frac{\partial E}{\partial h_i} \cdot \frac{\partial h_i}{\partial w_{ki}} = \mathbf{EH_i}.x_k \tag{14}$$

Now the weights are updated using

$${v_{wI}}^{(new)} = {v_{wi}}^{(old)} - \eta \mathbf{EH^T} \tag{15}$$

SOME INSIGHTS ON OUTPUT-HIDDEN-INPUT LAYER WEIGHT UPDATES

- The prediction error E propagates the weighted sum of all words in the vocabulary to every output vector v'_j
- The change in the input vector is defined by the output vector which in turn is updated due to the prediction error
- The model parameters accumulate the changes until the system reaches a state of equilibrium
- Ideally the $v_j \cdot v'_j$ will result in an identity
- The rows in the Input-Hidden layer (v_j) stores the features of the words in the vocabulary V

MATRIX OPERATIONS

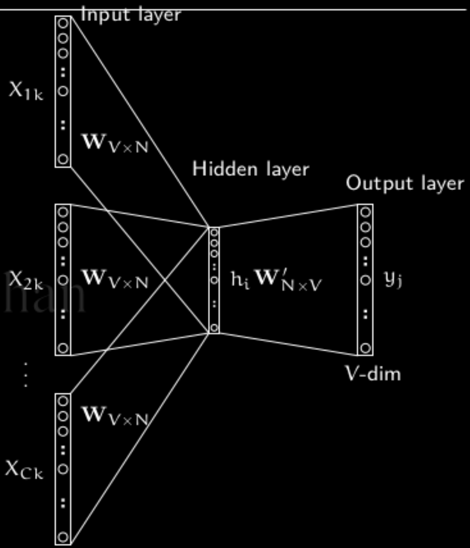
MATRIX OPERATIONS

MATRIX OPERATIONS

MATRIX OPERATIONS

CBOW MODEL FOR MULTIPLE WORDS

- C is the number of context words
- V is the size of the vocabulary
- h_i receives average of the vectors of the input context words
- Output vector v'_{wj} is the column vector in the $\mathbf{W}'$ representing relationship between the context words and the target word
- Softmax is used for the output layer probability distribution for the target word



1

[1] Reference:Xin Rong, word2vec Parameter Learning Explained

INPUT-HIDDEN WEIGHT VECTORS AND LOSS FUNCTION

The hidden units receive values from the linear combination of the context vectors and the weights

$$u_j = \mathbf{v}'^{\mathbf{T}}_{\mathbf{w_j}}\mathbf{h} = \mathbf{v}'^{\mathbf{T}}_{\mathbf{w_j}}\mathbf{v}_{\mathbf{w_I}} \tag{16}$$

$$\begin{aligned} \mathbf{h} &= \frac{1}{C}\mathbf{W}^{\mathsf{T}}(x_1 + x_2 + x_3 + \cdots + x_C) \\ &= \frac{1}{C}(\mathbf{v}_{w1} + \mathbf{v}_{w2} + \mathbf{v}_{w3} + \cdots + \mathbf{v}_{wC}) \end{aligned} \tag{17}$$

The equation for v'_j can be borrowed from (16) and E is

$$\begin{aligned} E &= -\log p(w_O | w_{I,1}, w_{I,2}, w_{I,3}, \ldots, w_{I,C}) \\ &= -v'_{wO} \cdot \mathbf{h} + \log \sum_{j'=1}^{V} \exp\left({\mathbf{v}'_{\mathbf{w_j}}}^{\mathsf{T}} \cdot \mathbf{h}\right) \end{aligned} \tag{18}$$

UPDATE INPUT AND OUTPUT VECTORS

There is no change in the hidden-output weights[2] (19)as the computations remain the same. The new $\mathbf{v}_{\mathbf{w_{I,c}}}^{(\mathbf{new})}$ is written as

$$\mathbf{v}_{\mathbf{w_{I,c}}}^{(\mathbf{new})} = \mathbf{v}_{\mathbf{w_{I,c}}}^{(\mathbf{old})} - \frac{1}{C} \cdot \eta \mathbf{EH}^{T}, \text{for} \quad j = 1, 2, 3, \dots C \tag{20}$$

where η is the learning rate.

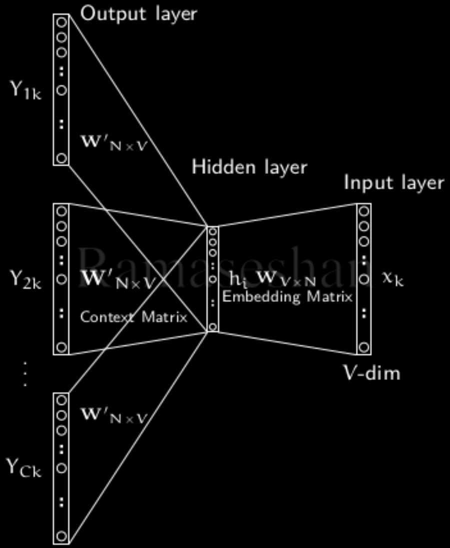
2

$$\mathbf{v}_{\mathbf{w_j}}^{(\mathbf{new})} = \mathbf{v'}_{\mathbf{w_j}}^{(\mathbf{old})} - \eta e_j \cdot \mathbf{h} \quad \text{for } j = 1, 2, 3, \dots, V \tag{19}$$

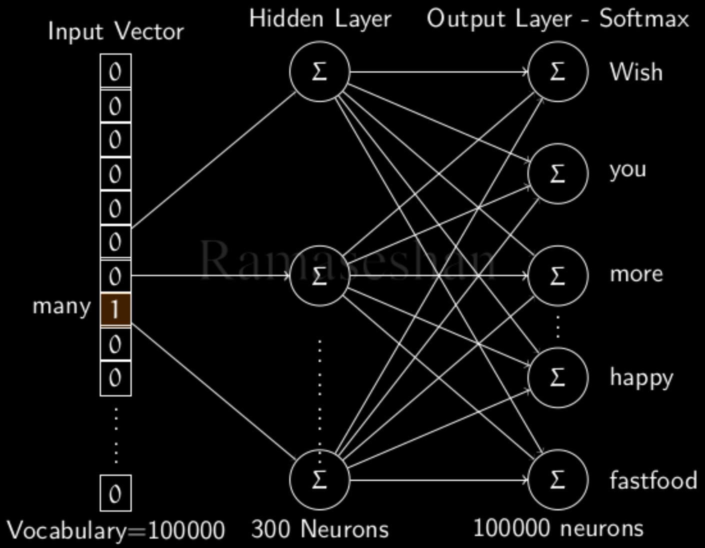
WHAT DOES IT LEARN?

- Distributed representation of words as vectors
- The learned vectors explicitly encode many linguistic regularities and patterns
- The learning should produce a similar word vectors for those words that appeared in similar context. How do we find out?
- Comparing the word vectors for similarity? Cosine similarity?
- Has the learned word vectors address stemming? run, running, ran as similar?
  - He runs half-marathon
  - He ran half-marathon
  - He is running half-marathon
- How about car, cars, automobile?
- How about awesome, fantastic, great?

SKIP-GRAM MODEL



NEURAL NETWORK ARCHITECTURE - A SAMPLE



A PYTHON IMPLEMENTATION FOR WORD EMBEDDING - 1/5

Initialization

```
def setup_corpus(self,corpus_dir='/home/ramaseshan/Dropbox/NLPClass/2019/SmallCorpus/'):
    self.corpus = PlaintextCorpusReader(corpus_dir, '.*')

def init_model_parameters(self,context_window_size=5,word_embedding_size=70,epochs=400,eta=0.01):
    self.context_window_size = context_window_size
    self.word_embedding_size = word_embedding_size
    self.epochs = epochs
    self.eta = eta

def initialize_weights(self):
    self.embedding_weights = np.random.uniform(-0.9, 0.9, (self.vocabulary_size, self.word_embedding_size)) #input weights
    self.context_weights = np.random.uniform(-0.9, 0.9, (self.word_embedding_size, self.vocabulary_size)) #input weights
```

A PYTHON IMPLEMENTATION FOR WORD EMBEDDING - 2/5

Forward pass

$$\mathbf{H} = \mathbf{W}^{\mathbf{T}}\mathbf{X}$$

$$\mathbf{U} = \mathbf{W}'^{\mathbf{T}}\mathbf{H} = \mathbf{W}'^{\mathbf{T}}.\mathbf{W}^{\mathbf{T}}\mathbf{X}$$


```python
def forward_pass(self,X):
    H = np.dot(self.embedding_weights.T, X)
    U = np.dot(self.context_weights.T, H)
    y_hat = self.softmax(U)
    return y_hat, H, U
```

A PYTHON IMPLEMENTATION FOR WORD EMBEDDING - 3/5

Back propagation

$$w'^{\,new}_{ij} = w'^{\,old}_{ij} - \eta e_j \cdot h_i \text{ or}$$

$$\mathbf{v}'^{(\mathbf{new})}_{\mathbf{w_j}} = \mathbf{v}'^{(\mathbf{old})}_{\mathbf{w_j}} - \eta e_j \cdot \mathbf{h} \quad \text{for } j = 1, 2, 3, \ldots, V$$

$$\frac{\partial E}{\partial w_{ki}} = \frac{\partial E}{\partial h_i} \cdot \frac{\partial h_i}{\partial w_{ki}} = \mathbf{EH_i}.x_k$$

```
def back_propagation(self,X,H,E):
    delta_context_weights = np.outer(H, E)
    delta_embedding_weights = np.outer(X, np.dot(self.context_weights, E.T
))

    # Change the weights using the back propagation values
    self.context_weights = self.context_weights - (self.eta *
delta_context_weights)
    self.embedding_weights = self.embedding_weights - (self.eta *
delta_embedding_weights)
    pass
```

Training

$$E = -v'_{wO} \cdot \mathbf{h} + \log \sum_{j'=1}^{V} \exp\left(\mathbf{v}'_{\mathbf{w}_j}{}^{T} \cdot \mathbf{h}\right) \tag{21}$$

```
    def train(self):
        for i in range(0, self.epochs):
            for target_word, context_words in np.array(self.training_samples):
                #for all the words
                y_hat, H, U = self.forward_pass(target_word)
                # compute error for all context words
                EI = np.sum([np.subtract(y_hat, word) for word in
                         context_words],axis=0)
                # back propagation to adjust weights
                self.back_propagation(target_word, H,EI)
                #Compute the error
                self.error[i] = -np.sum([U[word.index(1)]
                                     for word in context_words]) + \
                                       len(context_words) * \
                                       np.log(np.sum(np.exp(U)))
```

A PYTHON IMPLEMENTATION FOR WORD EMBEDDING - 5/5

Word vector for ***deep*** and similar words

$$\begin{bmatrix} -2.01970447 & 0.68963328 & 0.35593417 & 0.64125108 & \ldots & 0.91503001 \end{bmatrix}$$

Word	Similarity
deep	1.0
heard	0.767841548247
depth	0.706466540662
well	0.684150968491
sound	0.662830677002
peso	0.507131975602
hit	0.464345901325
after	0.458074275823
water	0.424398813383

SOURCE PREPARATION FOR TRAINING

Source Text

Wish you many more happy returns of the day $\longrightarrow$

Wish you more happy returns of the day $\longrightarrow$

Wish you many more happy returns of the day $\longrightarrow$

Wish you many more happy returns of the day $\longrightarrow$

Wish you many more happy returns of the day $\longrightarrow$

Wish you many more happy returns of the day $\longrightarrow$

Wish you many more happy returns of the day $\longrightarrow$

Training Samples

(wish, you)
(wish, many)
(you, Wish)
(you, more), (you, happy)
(many, Wish), (many, you)
(many, more), (many, happy)
(more, many), (more, you)
(more, happy), (more, returns
(happy, many), (happy, more
(happy, returns), (happy, of)
(returns, more), (returns, happ
(returns, of), (returns, the)
(of, happy), (of, returns)
(of, the), (of, day)

SUB-SAMPLING

- The words (of, the) in the pairs $(of, happy), (returns, the)$ do not give much information about the words happy and returns, respectively. Similarly, some pairs reappear with the order of the words switched.
- Some words could also be randomly removed from the based on the frequencies
- Words with less frequency or infrequent words appearing as context words could be discarded as they may not provide contextual information to the central word

SUB-SAMPLING IN WORD2VEC.C-GOOGLE

Here is the code for sub-sampling used by word2vec.c that randomly removes a word from the sample

```
if (word == 0) break;
// The subsampling randomly discards frequent words while keeping the ranking same
if (sample > 0) {
  real ran = (sqrt(vocab[word].cn / (sample * train_words)) + 1) * (sample * train_words) / vocab[word].cn;
  next_random = next_random * (unsigned long long)25214903917 + 11;
  if (ran < (next_random & 0xFFFF) / (real)65536) continue;
}
```

$$\text{let} \quad f(x) = \frac{vocab[word].cn}{train\_words} \quad \text{and} \quad ran = \left(\sqrt{f(x)} + 1\right) \times \frac{1}{f(x)} \tag{22}$$

where $vocab[word].cn$ is the count of the word $word$ and train_words represents all all the training words.Then, the probability of keeping the word is decided based on the generated random value $random$. If $ran < random$ keep it, else discard the word

NEGATIVE SAMPLING

- The size of the network is proportional to the size of the vocabulary V. For every training cycle of input, the every weight in the network needs to be updated
- For every training cycle, Softmax function computes the sum of the output neuron values
- Cost of updating all the weights in the fully connected network is very high
- Is it possible to change only a small percentage of the weights?

NEGATIVE SAMPLING

- The size of the network is proportional to the size of the vocabulary V. For every training cycle of input, the every weight in the network needs to be updated
- For every training cycle, Softmax function computes the sum of the output neuron values
- Cost of updating all the weights in the fully connected network is very high
- Is it possible to change only a small percentage of the weights?
- Select a small number of *negative* words
- While updating the weights, these samples output zero while the positive sample(s) will retain its value
- During the backpropagation, the weights related to the negative and positive words are changed and the rest will remain untouched for the current update
- This reduces drastically the computation $\smile$

SELECTING A NEGATIVE SAMPLE

$$P(w_i) = \frac{f(w_i)}{\sum_{j=0}^{n} f(w_j)} \tag{23}$$

$$P(w_i) = \frac{f(w_i)^{\frac{3}{4}}}{\sum_{j=0}^{n} f(w_j)^{\frac{3}{4}}} \tag{24}$$

It is important to chose more frequent words.This equation increases the probability of choosing the less frequent words. One way to implement is to create a unigram table filled with the words according to the probability. The frequently occurring words would be repeated several times according to their frequency thereby increasing the probability of choosing the frequent words for the ***negative*** samples

TROUBLE WITH THE SIZE OF THE NETWORK

- All weights $(\mathrm{output} \rightarrow \mathrm{hidden})$ and $(\mathrm{hidden} \rightarrow \mathrm{input})$ are adjusted by taking a training sample so that the prediction cycle minimizes the loss function
- This amounts to updating all the weights in the neural network - amounts to several million weights for a network which has input neurons, $|V| = 1M$, and hidden unit size as 300
- In addition, we should consider the several million training samples pairs

To deal with classification with multiple classes, softmax is very useful.If there are k classes in the data set, this activation function fits the classes in the range [0,1] by calculating the probability. This is best suited for the finding the activation value of the neurons in the output layer.It is a normalized exponential function

$$P(C_k|x_j) = \frac{e^{a_j}}{\Sigma_k e^{a_k}}, where \quad k = 1, K \tag{25}$$

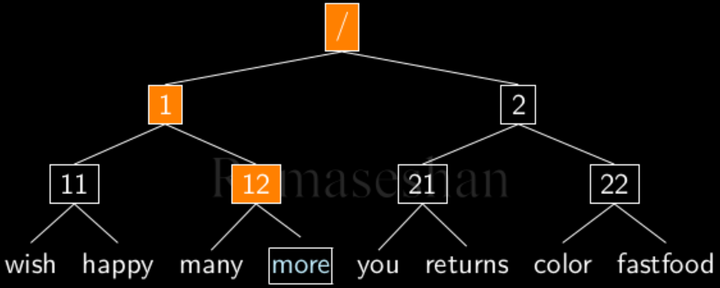
HIERARCHICAL SOFTMAX

- Has a flat hierarchy with a probability value for every output node of depth $= 1$
- Normalized over the probabilities of all $|V|$ words
- Error correction happens for every output$\longrightarrow$hidden units
- Huge costs if the vocabulary size $|V|$ is of the order of several thousands
- Decompose the flat hierarchy into a binary tree
- Form a hierarchical description of a word as a sequence of $O(\log_2|V|)$ decisions and there by reducing the computing complexity of Softmax - $O(|V|) \rightarrow O(\log_2(|V|))$
- Lay the words in a tree-based hierarchy - words as leaves
- Binary tree with $|V|-1$ nodes for left (0) and right(1) traversal
- Every leave represents the probability of the word

HIERARCHICAL SOFTMAX - 2

- Path length of a balanced Tree is $\log_2(|V|)$. If the $|V| = 1 \quad million \quad words$, then the path length $= 19.9$ bits/word
- Constructing an Huffman encoded-tree would help frequent words to have short unique binary codes
- Learn to take these probabilistic decisions instead of directly predicting each word's probability [**Bengio:2003:NPL:944919.944966**]
- Every intermediate node denotes the relative probabilities of its child nodes
- The path to reach every leaf (word) is unique
- H-Softmax in many cases increases the prediction speed by more than 50X times

HIERARCHICAL TREE STRUCTURE



Here the leaves represent the words and the numbered nodes represent the ***probability mass***

LIMITATIONS

- Separate training is required for phrases
- Embeddings are learned based on a small local window surrounding words - good and bad share the almost the same embedding
- Does not address polysemy
- Does not use frequencies of term co-occurrences

WORD2VEC - RESULTS

Word similarity for the word ***automobile***

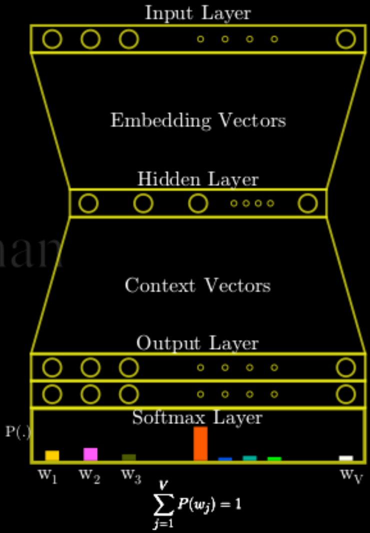
```
Terminal
Word: automobile  Position in vocabulary: 4525

                                              Word       Cosine distance
------------------------------------------------------------------------
                                        automotive              0.574008
                                      manufacturer              0.561169
                                               car              0.553800
                                       automobiles              0.546085
                                           daimler              0.532693
                                        volkswagen              0.523921
                                       dealerships              0.523413
                                             motor              0.523041
                                              benz              0.520432
                                         automaker              0.519787
                                           minivan              0.517683
                                             volvo              0.514867
                                            toyota              0.502340
                                               dkw              0.501307
                                   daimlerchrysler              0.500140
                                           bugatti              0.486601
                                           porsche              0.485660
                                           maybach              0.479814
                                             citro              0.477737
                                              cars              0.476576
```

SOFTMAX

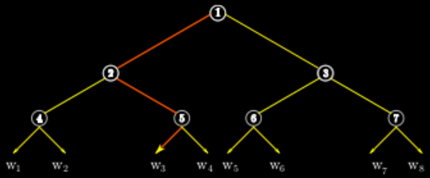
Softmax is a normalized exponential function

$$P(C_k|x_j) = \frac{e^{a_j}}{\Sigma_k e^{a_k}}, where \quad k = 1, K \quad (26)$$

- Has a flat hierarchy with a probability value for every output node of depth = 1
- Normalized over the probabilities of all $|V|$ words
- Error correction happens for every output→hidden units
- Huge costs if the vocabulary size $|V|$ is of the order of several thousands

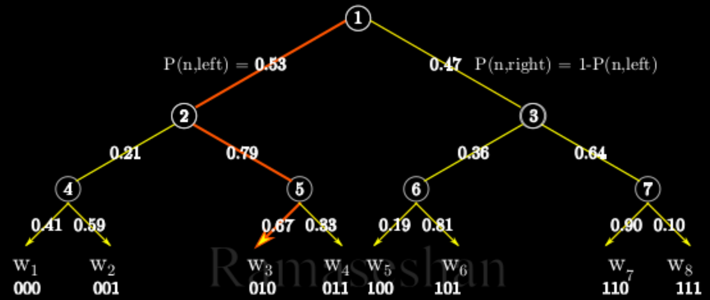


BALANCED BINARY TREE



- Move the words into a binary tree - depth depends on the vocabulary
- The path to any word in the vocabulary is known
- Traverse through the binary tree to reach any word
- At every step, make a binary decision to reach the word
- The length to reach any word in a balanced tree is $\log_2(|V|)$
- Words could be arranged using
  - random order
  - IS-A relationship
  - TF-IDF frequency

BALANCED BINARY TREE WITH 8 WORDS



$P(w_i) = \prod_{j \in N_L} P(n(w,j))$,where N_L is the list of nodes to reach the word and

$\mathbf{P}(\mathbf{W}) = \sum_{i=1}^{V} P(w_i) = 1$

	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8
$P(w_i)$	0.0456	0.0657	0.2805	0.1382	0.0321	0.1371	0.2707	0.0301

Thus the hierarchical Softmax is a well defined multinomial distribution among all words

HIERARCHICAL SOFTMAX - ADVANTAGES

- Decomposes the flat hierarchy into a binary tree
- The path to reach every leaf (word) is unique
- Lays the words in a tree-based hierarchy - words as leaves
- Binary tree with $|V| - 1$ nodes for left and right traversal
- Every intermediate node denotes the relative probabilities of its child nodes
- Every leaf represents the probability of the word

HIERARCHICAL SOFTMAX - ADVANTAGES

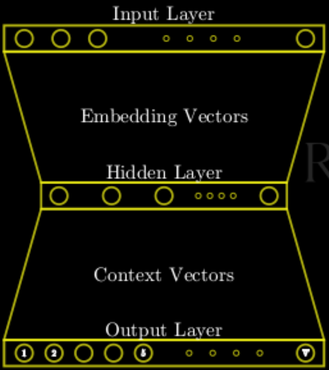
- Each node is indexed by a bit vector corresponding to the path from the root to the node
  - Append 1 or 0 according to whether the left or right branch of a decision node
- Normalized values for the words are calculated without finding the probability for every word
- The entire vocabulary is partitioned into classes
- ANN learns to take these probabilistic decisions instead of directly predicting each word's probability[3]

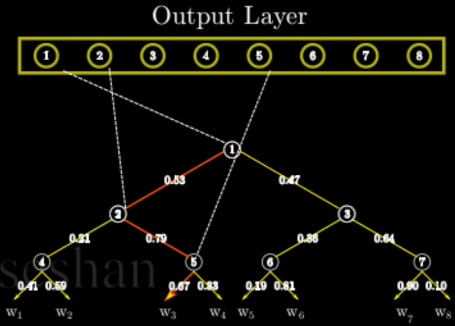
[3] Yoshua Bengio et al. "A Neural Probabilistic Model" In: J.MAch.Learn Res.3 (Mar.2003), pp.1137-1155. issn: 1532-4435

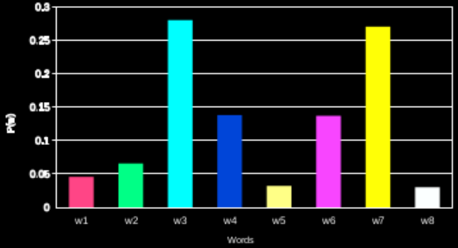
HIERARCHICAL SOFTMAX - ADVANTAGES

- Forms a hierarchical description of a word as a sequence of $O(\log_2|V|)$ decisions
- Reduces the computing complexity of Softmax - $O(|V|) \rightarrow O(\log_2(|V|))$
- Path length of a balanced Tree is $\log_2(|V|)$. If the $|V| = 1 \quad \text{million} \quad \text{words}$, then the path length $= 19.9$ bits/word
- A balanced binary tree should provide an exponential speed-up, on the order of $\dfrac{|V|}{\log_2(|V|)}$
- Constructing an Huffman encoded-tree would help frequent words to have short unique binary codes
- H-Softmax in many cases increases the prediction speed by more than 50X times

WORD2VEC WITH HIERARCHICAL SOFTMAX







UPDATING WEIGHTS - 1/3

Let $L(w)$ be the number of nodes to traverse to the word from the root and $n(w,i)$ is the i^{th} node

on this path and the associated vector in the context matrix is $v_{n(w,i)}$. $ch(n)$ is the child node[2][**Bengio:2003:NPL:944919.944966**][3][**Mnih:2008:SHD:2981780.2981915**].

Then the probability of word is

$$P(w|w_i) = \prod_{j=1}^{L(w)-1} \sigma([n(w,j+1) = ch(n(w,j))].v^T_{n(w,j)}h)$$

$$= \prod_{j=1}^{L(w)-1} \sigma([n(w,j+1) = ch(n(w,j))].v^T_{n(w,j)}v_{w_i}) \tag{27}$$

$$\text{where}[x] = \begin{cases} 1, & \text{if } x \text{ is true} \\ -1, & \text{otherwise} \end{cases} \quad \text{and } \sigma(.) \text{is the sigmoid function}$$

if the child node $ch(n(w,j)$ is left of the parent node, then the term $[n(w,j+1) = ch(n(w,j))]$ is 1, and equal to -1 if the path goes to the right.

UPDATING WEIGHTS - 2/3

Since the sum of the probabilities of at the node is 1, we can prove that

$$\sigma(v_n^T v_{w_i}) + \sigma(-v_n^T v_{w_i}) = 1 \tag{28}$$

Example

$$P(w|w_i) = \sigma(v_{n(w,j)}^T v_{w_i}).\sigma(-v_{n(w,j)}^T v_{w_i}).\sigma(v_{n(w,j)}^T v_{w_i})$$

To train the model, we need to minimize the negative log likelihood $-\log P(w|w_i)$

$$E = -\sum_{j=1}^{L(w)-1} \log \sigma([.]u_j'), \text{ where } u_j = v_j'.h \tag{29}$$

$$\tag{30}$$

where $t_j = 1$, if $[n(w,j+1) = ch(n(w,j))] = 1$ and $t_j = 0$ otherwise

UPDATING WEIGHTS - 3/3

$$\frac{\partial E}{\partial u_j} = \sigma(u_j - 1)[.] \tag{31}$$

$$= \begin{cases} \sigma(u_j) - 1 & \text{for } [.] = 1 \\ \sigma(u_j) & \text{for } [.] = -1 \end{cases} \tag{32}$$

$$= \sigma(u_j) - t_j \tag{33}$$

where $t_j = 1$, if $[n(w, j+1) = ch(n(w,j))] = 1$, else $t_j = 0$

$$\frac{\partial E}{\partial v'_j} = \sigma(v'_j.h) - t_j \tag{34}$$

$$v'^{new}_j = v'^{old}_j - \eta(\sigma(v'_j.h) - t_j).h \tag{35}$$

$$\frac{\partial E}{\partial h} = \sum_{j=1}^{L(W)-1} \frac{\partial E}{\partial v'_j h} . \frac{\partial v'_j h}{\partial h} = EH \tag{36}$$

$$v^{new}_{w_i} = v^{old}_{w_i} - \eta.EH^T \tag{37}$$

WORD2VEC - RESULTS

The source code for word2vec is available at https://github.com/dav/word2vec. Word similarity for the word ***automobile***

```
Terminal
Word: automobile  Position in vocabulary: 4525

                                              Word       Cosine distance
------------------------------------------------------------------------
                                        automotive              0.574008
                                      manufacturer              0.561169
                                               car              0.553800
                                       automobiles              0.546085
                                           daimler              0.532693
                                        volkswagen              0.523921
                                       dealerships              0.523413
                                             motor              0.523041
                                              benz              0.520432
                                         automaker              0.519787
                                           minivan              0.517683
                                             volvo              0.514867
                                            toyota              0.502340
                                               dkw              0.501307
                                   daimlerchrysler              0.500140
                                           bugatti              0.486601
                                          porsche               0.485660
                                           maybach              0.479814
                                             citro              0.477737
                                              cars              0.476576
```

REFERENCES

- Xin Rong. "word2vec Parameter Learning Explained". In: *CoRR* abs/1411.2738 (2014). arXiv: 1411.2738. URL: http://arxiv.org/abs/1411.2738.
- Jeffrey A. Dean Tomas Mikolov Kai ChenGregory S. Corrado. U.S. pat. US9037464B1. May 2015.
- Tomas Mikolov et al. "Distributed Representations of Words and Phrases and Their Compositionality". In: *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*. NIPS'13. Lake Tahoe, Nevada: Curran Associates Inc., 2013, pp. 3111–3119. URL: http://dl.acm.org/citation.cfm?id=2999792.2999959.