Natural Language Processing

WORKSHOP

24-28 August, 2022

TOPICS TO BE COVERED IN THIS WORKSHOP

GITHUB PAGE

This link provides all resources used in this workshop
https://github.com/Ramaseshanr/IITMDS

Ability to process and harness information from a large corpus of text with no human intervention

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

- **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 | more | happy | returns | of | the | day
- **Skip Gram Model**- Given the central word, identify the surrounding words
  - Wish you many more happy returns of the day

CONTINUOUS BAG OF WORDS (CBOW)

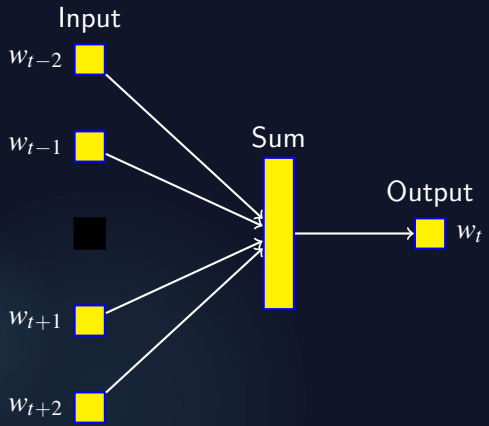


Figure: The CBOW architecture predicts the current word based on the context words of length n. Here the window size is 5

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

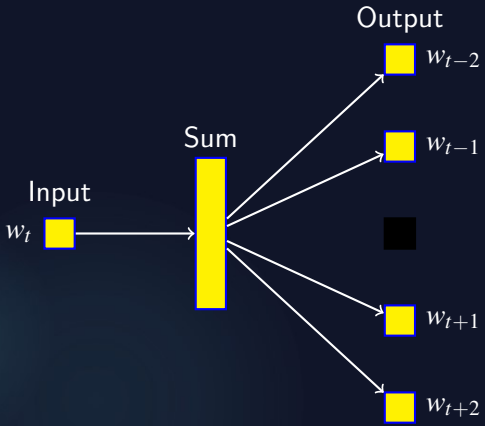


Figure: The SG architecture predicts the one context word at a time based on the center word. Here the window size is 5

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

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

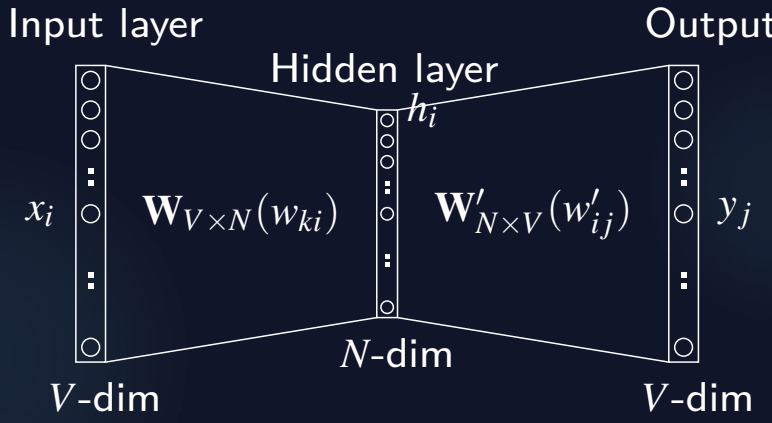


Figure: A CBOW model with only one word as input[**DBLP:journals/corr/Rong14**]. The layers are fully connected

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 [**US9037464B1**] [**Mikolov:2013:DRW:2999792.2999959**].

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

Now v_{wI} of the matrix (W) is the vector representation of the input one-hot vector w_I. From (0), h is a linear combination of input and weights.
In the same way. we get a score for u_j

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

where v_{w_I} is the vector representation of the input word w_I and v'_{w_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,

$$p(w_j|w_I) = y_j$$

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

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

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

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 (41) 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"

$$max\, p(w_o|w_I) = max(\log(y_{j*})) - - \text{Maximize}$$
$$-E = u_j - \log(y_{j*}) - - \text{minimize}$$
$$= u_{j*} - \log \sum_{j'=1}^{V} \exp(u'_j)$$

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

$$\frac{\partial E}{\partial u_j} = y_j - t_j = e_j \tag{2}$$

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

$$\frac{\partial E}{\partial w'_{ij}} = \frac{\partial E}{\partial u_j} \cdot \frac{\partial u_j}{\partial w'_{ij}} = e_j \cdot h_i \tag{2}$$

Using the above equation (43),

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

UPDATE INPUT TO HIDDEN WEIGHTS

Taking the derivative with respect to h_i, we get

$$\frac{\partial E}{\partial h_i} = \sum_{j=1}^{V} \frac{\partial E}{\partial u_j} \cdot \frac{\partial u_j}{\partial h_i} = \sum_{j=1}^{V} e_j \cdot w'_{ij} = \mathbf{EH_i} \tag{2}$$

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

$$\frac{\partial E}{\partial w_{ki}} = \frac{\partial E}{\partial h_i} \cdot \frac{\partial h_i}{\partial w_{ki}} = \mathbf{EH_i}.x_k \tag{2}$$

Now the weights are updated using

$$v_{w_i}{}^{(new)} = v_{wi}{}^{(old)} - \eta \mathbf{EH^T} \tag{2}$$

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

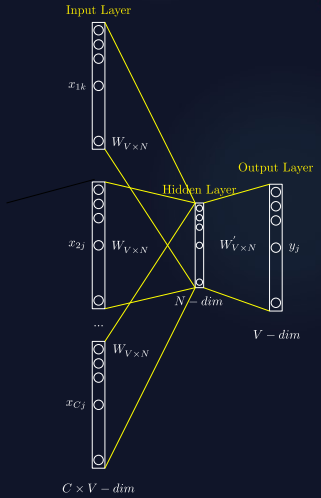
MATRIX OPERATIONS

MATRIX OPERATIONS

MATRIX OPERATIONS

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



The CBOW Model

INPUT-HIDDEN WEIGHT VECTORS AND LOSS FUNCTION

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

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

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

The equation for v'_j can be borrowed from (69) and E is

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

UPDATE INPUT AND OUTPUT VECTORS

There is no change in the hidden-output weights[2] (0)as the computations remain the same. The new $\mathbf{v}_{\mathbf{w_{I,c}}}^{(\mathbf{new})}$ is written as

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

where η is the learning rate.

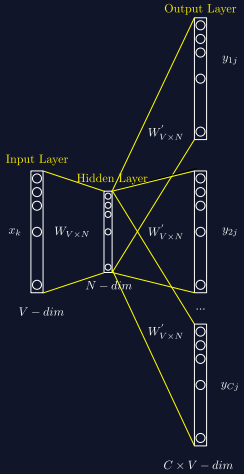
2

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

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



The Skip Gram Model

SKIP-GRAM MODEL - FORWARD AND BACK PROPAGATION UPDATE OF WEIGHTS

$$\text{Hidden layer } \mathbf{h} = W^T x$$

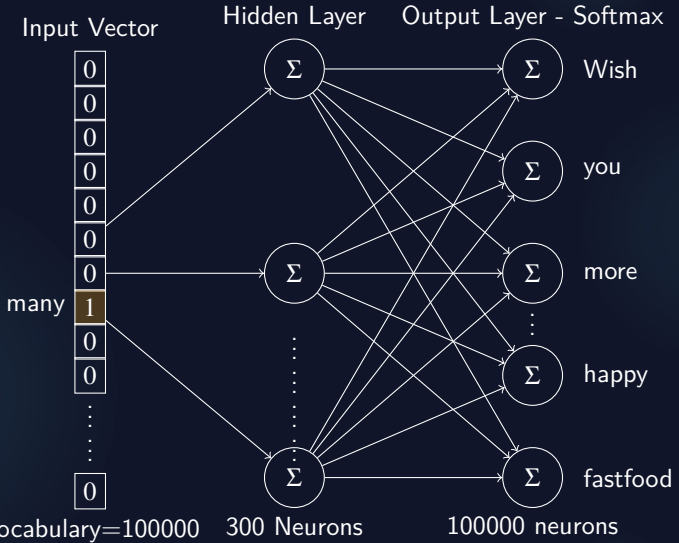
$$\text{Output neuron } u_{c,j} = \mathbf{v}' T_{w_j}.\mathbf{h}, \text{for } c = 1, 2, \ldots C$$

$$\text{cross entropy } E = -\sum_{c=1}^{C} u^*_{jc} + C.log \sum_{j'=1}^{|v|} exp(u_{j'})$$

$$\text{Context weights } w'^{new}_{ij} = w'^{old}_{ij} - \eta.EI_j.h_i$$

$$\text{input weights } v^{new}_{ij} = w^{old}_{ij} - \eta E_j.h_i$$

NEURAL NETWORK ARCHITECTURE - A SAMPLE



A PYTHON IMPLEMENTATION FOR WORD EMBEDDING - 1/5

Initialization

```
1
    def setup_corpus(self,corpus_dir='/home/ramaseshan/Dropbox/NLPClass/2019/
    SmallCorpus/'):
3       self.corpus = PlaintextCorpusReader(corpus_dir, '.*')

5   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
7       self.word_embedding_size = word_embedding_size
        self.epochs = epochs
9       self.eta = eta

11  def initialize_weights(self):
        self.embedding_weights = np.random.uniform(-0.9, 0.9, (self.
    vocabulary_size, self.word_embedding_size)) #input weights
13      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

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

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

A PYTHON IMPLEMENTATION FOR WORD EMBEDDING - 3/5

Back propagation

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

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

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

```
    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
```

A PYTHON IMPLEMENTATION FOR WORD EMBEDDING - 4/5

Training

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

```
    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

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

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→					
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

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

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;
        }
```

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

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

SELECTING A NEGATIVE SAMPLE

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

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

It is important to choose more frequent words.This equation decreases 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 $(output \rightarrow hidden)$ and $(hidden \rightarrow input)$ 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

SOFTMAX

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

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

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$\rightarrow$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

LIMITATIONS

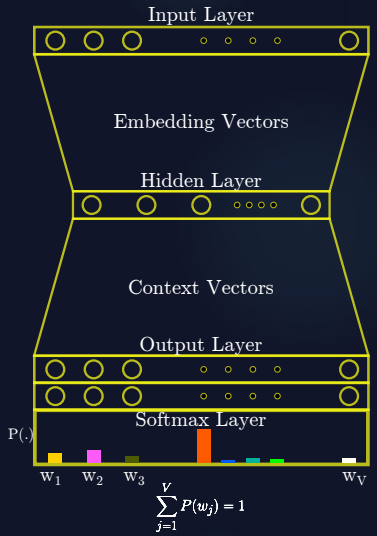
- Separate training is required for phrases
- Embeddings are learned based on a small local window surrounding words - good and bad share the almost the same embedding
- Does not address polysemy
- Does not use frequencies of term co-occurrences

SOFTMAX

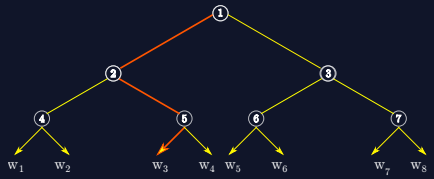
Softmax is a normalized exponential function

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

- 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

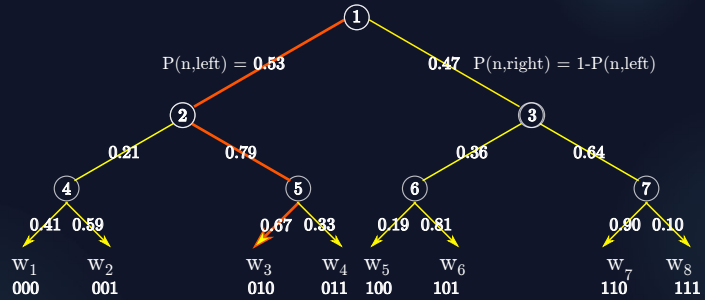


BALANCED BINARY TREE



- 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

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

	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

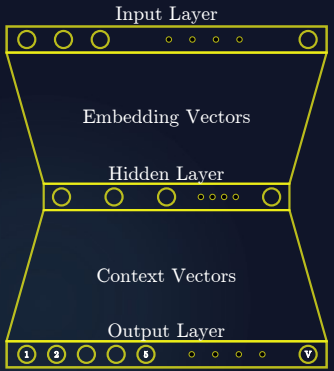
- Each node is indexed by a bit vector corresponding to the path from the root to the node
  - Append 1 or 0 according to whether the left or right branch of a decision node
- Normalized values for the words are calculated without finding the probability for every word
- The entire vocabulary is partitioned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

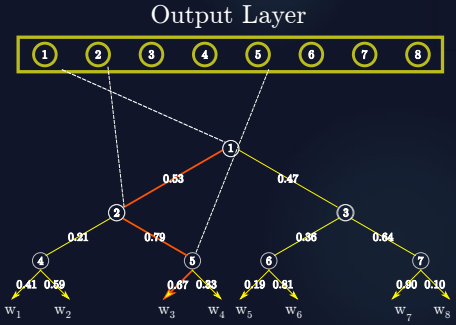
[3] Yoshua Bengio et al. "A Neural Probabilistic Model" In: J.MAch.Learn Res.3 (Mar.2003), pp.1137-1155. issn: 1532-4435

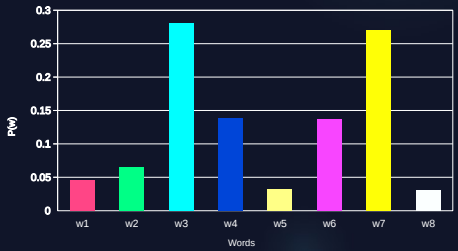
HIERARCHICAL SOFTMAX - ADVANTAGES

- 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 $|V| = 1 \quad million \quad words$, then the path length $= 19.9$ bits/word
- A balanced binary tree should provide an exponential speed-up, on the order of $\frac{|V|}{log_2(|V|)}$
- 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[**US9037464B1**][**Bengio:2003:NPL:944919.944966**][**Mikolov:2013:DRW:2999792.2**
Then the probability of word is

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

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

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

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

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

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 $-\log P(w|w_i)$

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

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

UPDATING WEIGHTS - 3/3

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

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

$$= \sigma(u_j) - t_j$$

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

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

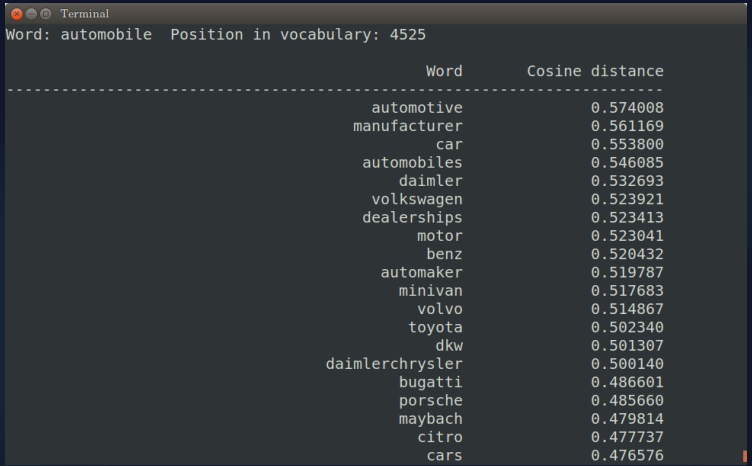
$$v'^{new}_j = v'^{old}_j - \eta(\sigma(v'_j.h) - t_j).h$$

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

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

WORD2VEC - RESULTS

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



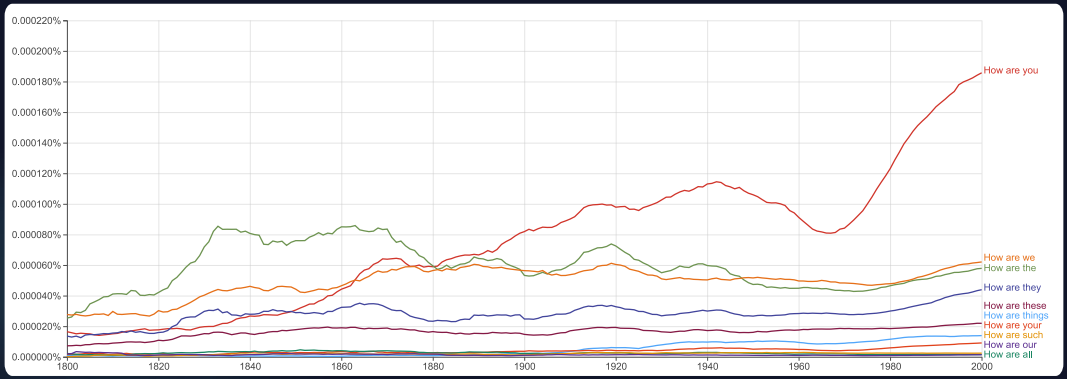
REFERENCES

INTRODUCTION

How are ____? Can you guess the missing word?

INTRODUCTION

How are ____? Can you guess the missing word?

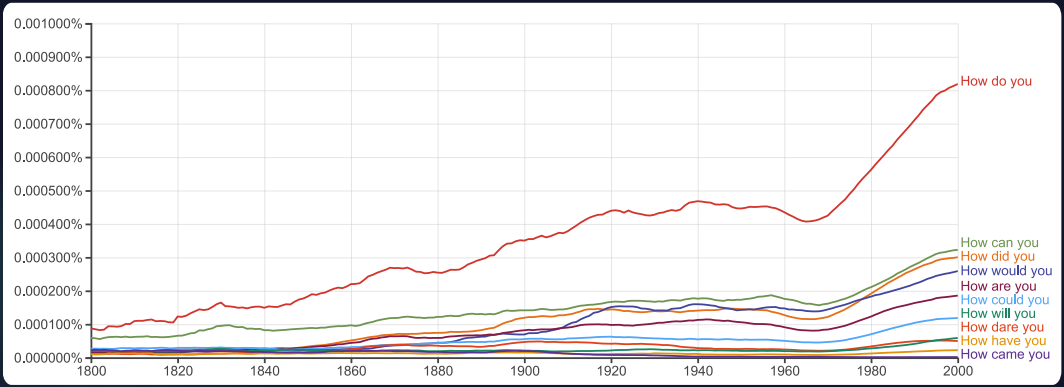


Source:Google NGram Viewer

INTRODUCTION

How ____ you? Can you guess the missing word?

How ____ you? Can you guess the missing word?



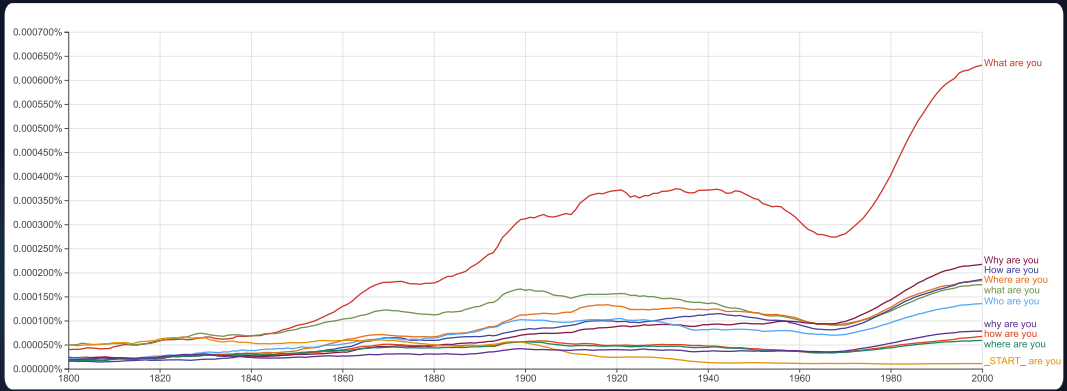
Source:Google NGram Viewer

INTRODUCTION

_____ are you?

INTRODUCTION

_____ are you?



Source:Google NGram Viewer

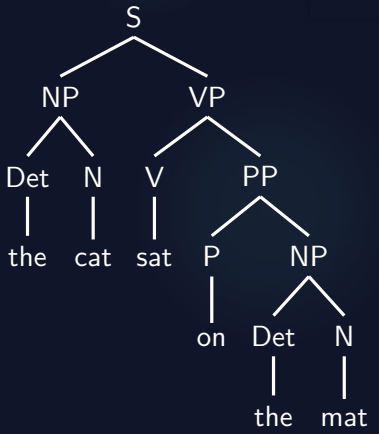
INTRODUCTION

How do humans predict the next word?

- Domain knowledge
- Syntactic knowledge
- Lexical knowledge
- Knowledge about the sentence structure
- Some words are hard to find. Why?
- Natural language is not deterministic in general
- Some sentences are familiar or had been heard/seen/used several times
- They are more likely to happen than others, hence we could guess

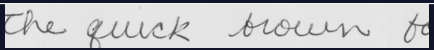
THE LANGUAGE MODEL

- Natural language sentences can be described by parse trees which use the morphology of words, syntax and semantics
- Probabilistic thinking - finding how likely a sentence occurs or formed, given the word sequence.
- In probabilistic world, the Language model is used to assign a probability $P(W)$ to every possible word sequence W.

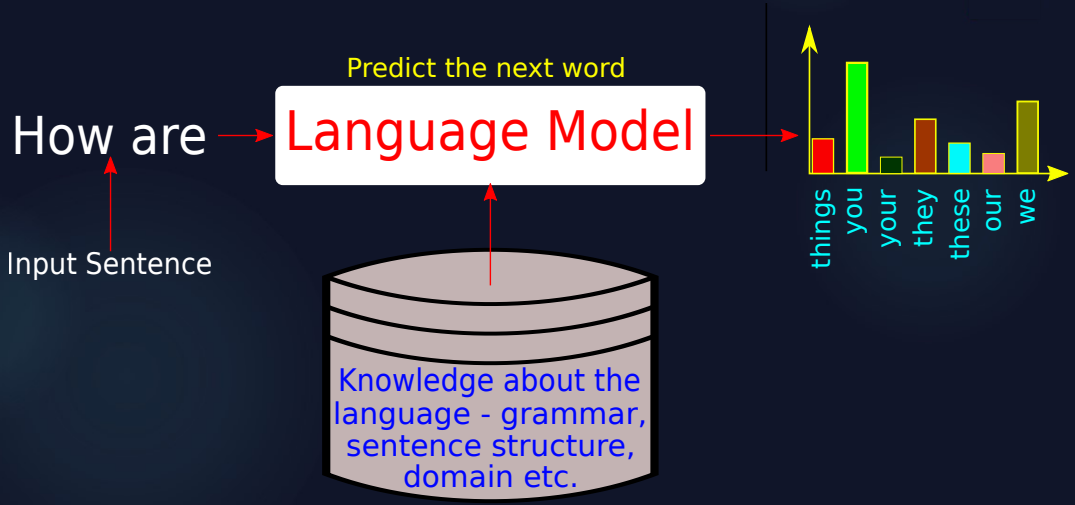


The current research in Language models focuses more on building the model from the huge corpus of text

APPLICATIONS

Application	Sample Sentences
Speech Recognition	Did you hear ***Recognize speech*** or Wreck a nice beach?
Context sensitive Spelling	One upon a ***tie***, ***Their*** lived aking
Machine translation	artwork is good $\rightarrow$ l'oeuvre est bonne
Sentence Completion	Complete a sentence as the previous word is given - GMail
OCR and Hand-written recognition	

A SIMPLE LANGUAGE MODEL IMPLEMENTATION



WHY PROBABILISTIC MODEL

- Speech recognition systems cannot depend on the processed speech signals. It may require the help of a language model and context recognizer to convert a speech to correct text format.
- As there are multiple combinations for a word to be in the next slot in a sentence, it is important for language modeling to be probabilistic in nature - judgment about the fluency of a sequence of words returns the probability of the sequence
- The probability of the next word in a sequence is real number $[0, 1]$
- The combination of words with high-probability in a sentence are more likely to occur than low-probability ones
- A probabilistic model continuously estimates the rank of the words in a sequence or phrase or sentence in terms of frequency of occurrence

FORMAL DEFINITION

Let $\mathscr{V}$ be the vocabulary, a finite set of symbols or words. Let us use $\triangleleft$ and $\triangleright$ as the start and stop symbols and let them be the part of $\mathscr{V}$. Let $|\mathscr{V}|$ denote the size of $\mathscr{V}$.

Let W be infinite sequences of words from the collection of $\mathscr{V}$. Every sequence in W starts with $\triangleleft$ and ends with $\triangleright$. Then a language model is a probability distribution of a random variable $\mathscr{X}$ which takes values from W. Or p: $W \to \mathbb{R}$ such that

$$\forall x \in W, p(x) \geq 0 \text{ and}$$
$$\sum_{x \in W} p(X = x) = 1$$

Goal: Compute the probability of a sequence of words

$$P(W) = P(w_1, w_2, w_3, ...w_n) \tag{-3}$$

Task: To predict the next word using probability. Given the context, find the next word using

$$P(w_n | w_1, w_2, w_3, \ldots, w_{n-1}) \tag{-3}$$

A model which computes the probability for (14) or predicting the next word (14) or complete the partial sentence is called as Probabilistic Language Model.

The goal is to learn the joint probability function of sequences of words in a language. The probability of P(The cat roars) is less likely to happen than P(The cat meows)

CHAIN RULE

Is it difficult to compute the probability of the entire sequence $P(w_1, w_2, w_3, \ldots, w_n)$? ***Chain rule*** is used to decompose the joint probability of a sequence into a product of conditional probability

$$\begin{aligned} P(W) &= P(w_1, w_2, w_3, \ldots, w_n) = P(w_1^n) \\ &= P(w_1)P(w2|w_1)P(w3|w_2, w_1)\ldots P(w_n|w_{n-1}, w_{n-2}, w_{n-3}, \ldots, w_1) \\ &= \prod_{k=1}^{n} P(w_k|w_1^{k-1}) \end{aligned}$$

- It is possible to $P(w|h)$, but it does not really help in reducing the computational complexity
- We use innovative ways to string words to form new sentences
- Finding the probability for a long sentence may not yield good outcome as the context may never occur in the corpus
- Short sequences may provide better results

MARKOV ASSUMPTION

Markov Assumption:The future behavior of a dynamic system depends on its recent history and not on the entire history

The product of the conditional probabilities can be written approximately for a bigram as

$$P(w_k|w_1^{k-1}) \approx P(w_k|w_{k-1})$$

Equation (16) can be generalized for an *n-gram* as

$$P(w_k|w_1^{k-1}) \approx P(w_k|w_{k-K+1}^{k-1})$$

Now, the joint probability of a sequence can be re-written as

$$\begin{aligned} P(W) &= P(w_1, w_2, w_3, \ldots, w_n) = P(w_1^n) \\ &= P(w_1)P(w2|w_1)P(w3|w_2, w_1)\ldots P(w_n|w_{n-1}, w_{n-2}, w_{n-3}, \ldots, w_1) \\ &= \prod_{k=1}^{n} P(w_k|w_1^{k-1}) \\ &\approx \prod_{k=1}^{n} P(w_k|w_{k-K+1}^{k-1}) \end{aligned}$$

TARGET AND CONTEXT WORDS

Next word in the sentence depends on its immediate past words, known as context words

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

n-grams

- unigram - $P(w_{k+1})$
- bigram - $P(w_{k+1}|w_k)$
- trigram - $P(w_{k+1}|w_{k-1}, w_k)$
- 4-gram - $P(w_{k+1}|w_{k-2}, w_{k-1}, w_k)$

LANGUAGE MODELING USING UNIGRAMS

- All words are generated independent of its history $W, W_2, W_3, \ldots . W_n$ and none of them depend on the other
- Not a good model for language generation
- It will have $|V|$ parameters
- $\theta_i = p(w_i) = \frac{c_{w_i}}{N}$, where c_{w_i} if the count of the word w_i and N is the total number of words in the vocabulary
- It may not be able to pick up regularies present tin the corpus
- It is more likely to generate ***the the the the*** as a sentence than a grammatically valid sentence

GENERATIVE MODEL

- Generates a document containing N words using n-gram
- A good model assigns higher probability to the word that actually occurs

$$P(\mathbf{W}) = P(N)\prod_{i=1}^{N} P(W_i) \tag{-4}$$

- The location of the word in the document is not important
- P(N) is the distribution over N and is same for all documents. Hence it may be ignored
- W_i, to be estimated in this model is $P(W_i)$ and it must satisfy $\sum_{i=1}^{N} P(w_i) = 1$

MAXIMUM LIKELIHOOD ESTIMATE

- One of the methods to find the unknown parameter(s) is the use of Maximum Likelihood Estimate
- Estimate the parameter value for which the observed data has the highest probability
- Training data may not have all the words in the vocabulary
- If a sentence with an unknown word is presented, then the MLE is zero.
- Add a smoothing parameter to the equation without affecting the overall probability requirements

$$P(\mathbf{W}) = \frac{C_{w_i} + \alpha}{C_W + \alpha|V|}$$

If $\alpha = 1$, then it is called as Laplace smoothing

$$P(\mathbf{W}) = \frac{C_{w_i} + 1}{C_W + |V|}$$

BIGRAM LANGUAGE MODEL

- This model generates a sequence one word at a time, starting with the first word and then generating each succeeding word conditioned on the previous one or its predecessor
- A bigram language model or the Markov model (first order)is defined as follows:

$$P(\mathbf{W}) = \prod_{i=1}^{n+1} P(w_i|w_{i-1})$$

where $\mathbf{W} = w_1, w_2, w_3, \ldots, w_n$

BIGRAM LANGUAGE MODEL

- Estimate the parameter $P(w_i|w_{i-1})$ for all bigrams
- The parameter estimation does not depend on the location of the word
- If we consider the sentence as a sequence in time, they are time-invariant MLE picks up the word that is $\frac{n_{w,w'}}{n_{w,o}}$ where nw, w' is the number of times the words w_1, w' occur together and $n_{w,o}$ is the number of times the word w appears in the bigram sequence with any other word
- The number of parameters to be estimated $= |V| \times (|V| + 1)$

PROBABILISTIC LANGUAGE MODEL - EXAMPLE

Peter Piper picked a peck of pickled peppers
A peck of pickled peppers Peter Piper picked
If Peter Piper picked a peck of pickled peppers
Where's the peck of pickled peppers Peter Piper picked?

—

The joint probability of a sentence formed with n words can be expressed as a product conditional probabilities - we use immediate context and not the entire history

$P(w_1|\langle\triangleleft\rangle) \times P(w_2|w_1) \times ...P(\langle E\rangle|w_n)$

and $P(w_{i+1}|w_i) = \frac{C(w_i.w_{i+1})}{C(w_i)}$

—

What is the probability of these sentences?
P(Peter Piper picked)
P(Peter Piper picked peppers)

Bigram	Frequency
◁Peter	1
Peter Piper	4
Piper picked	4
picked a	2
a peck	2
peck of	4
pickled peppers	4
peppers ▷	1
◁A peck	1
of pickled	4
peppers Peter	2
...	..
◁...	1

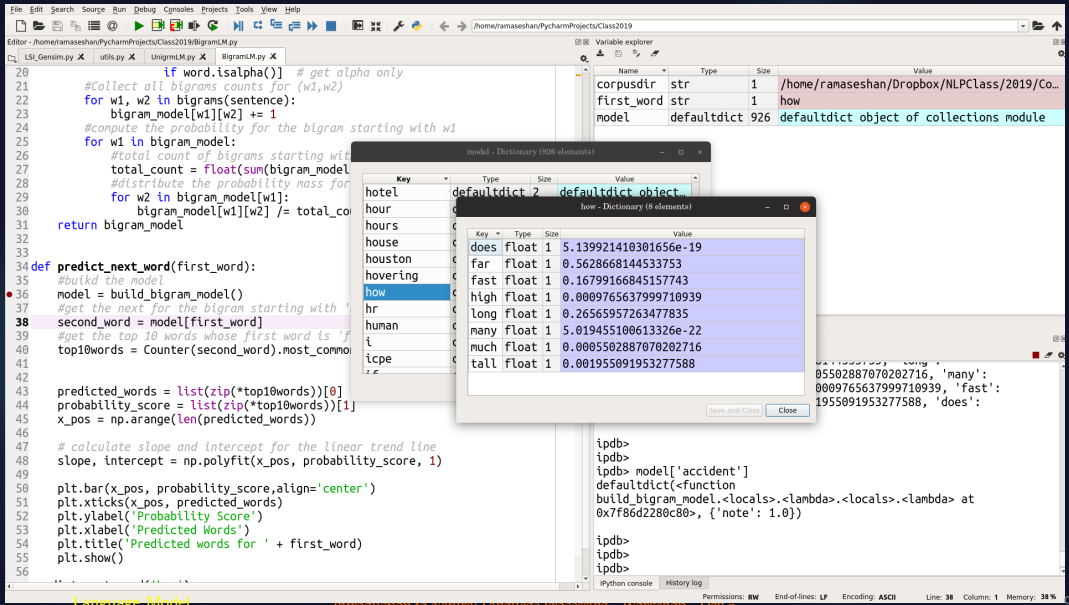
BUILDING A BIGRAM MODEL - CODE

```
#compute the bigram model
def build_bigram_model():
    bigram_model = collections.defaultdict(
        lambda: collections.defaultdict(lambda: 0))
    for sentence in kinematics_corpus.sents():
        sentence = [word.lower() for word in sentence
                    if word.isalpha()]  # get alpha only
        #Collect all bigrams counts for (w1,w2)
        for w1, w2 in bigrams(sentence):
            bigram_model[w1][w2] += 1
        #compute the probability for the bigram containing w1
        for w1 in bigram_model:
            #total count of bigrams conaining w1
            total_count = float(sum(bigram_model[w1].values()))
            #distribute the probability mass for all bigrams starting with w1
            for w2 in bigram_model[w1]:
                bigram_model[w1][w2] /= total_count
    return bigram_model
```

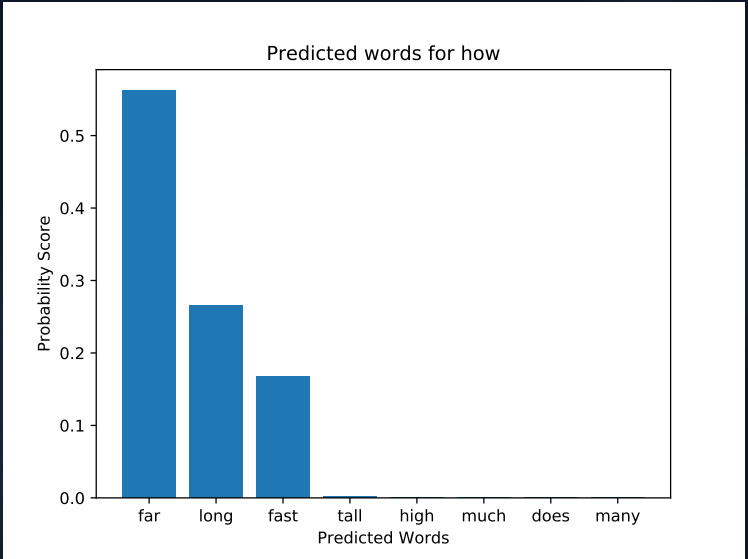
BUILDING A BIGRAM MODEL - CODE

```
def predict_next_word(first_word):
    #buikd the model
    model = build_bigram_model()
    #get the next for the bigram starting with 'word'
    second_word = model[first_word]
    #get the top 10 words whose first word is 'first_word'
    top10words = Counter(second_word).most_common(10)

    predicted_words = list(zip(*top10words))[0]
    probability_score = list(zip(*top10words))[1]
    x_pos = np.arange(len(predicted_words))

    plt.bar(x_pos, probability_score,align='center')
    plt.xticks(x_pos, predicted_words)
    plt.ylabel('Probability Score')
    plt.xlabel('Predicted Words')
    plt.title('Predicted words for ' + first_word)
    plt.show()

predict_next_word('how')
```

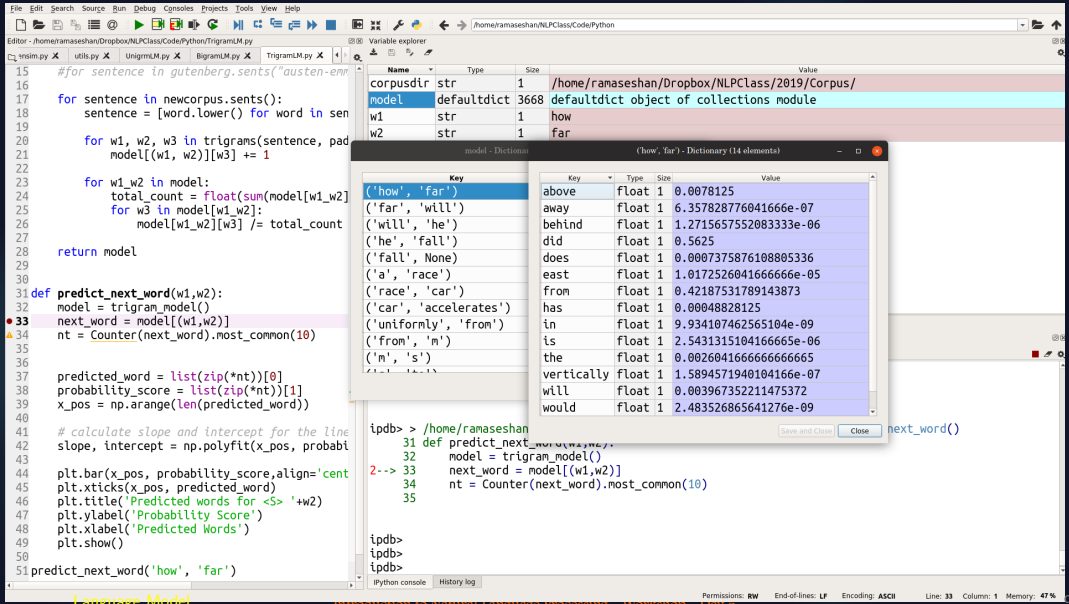
MODEL PARAMETERS - BIGRAM EXAMPLE



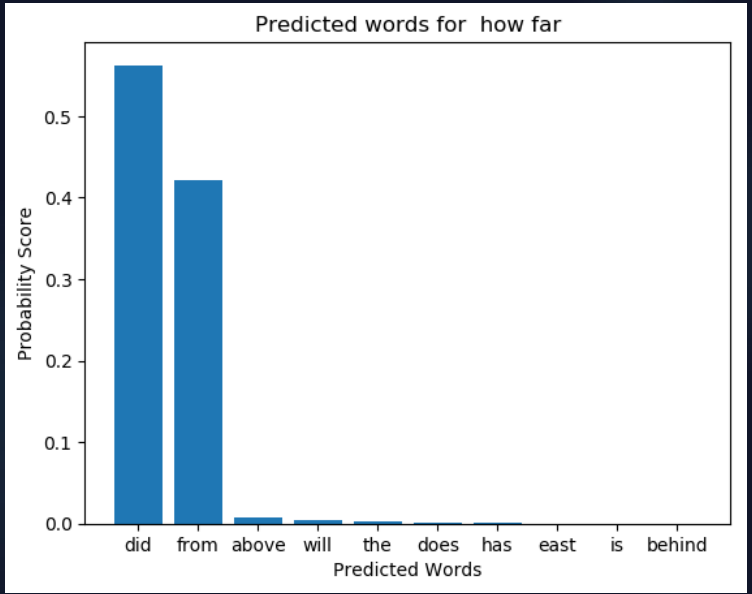
BIGRAM MODEL - NEXT WORD PREDICTION



MODEL PARAMETERS - TRIGRAM EXAMPLE



TRIGRAM MODEL - NEXT WORD PREDICTION



PERPLEXITY

Perplexity is a measurement of how well a probability model predicts a sample. Perplexity is defined as

$$\text{For bigram model, } PP(W_N) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_{i-1})}}$$

$$\text{For trigram model } PP(W_N) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_{i-1}w_{i-2})}}$$

A good model gives maximum probability to a sentence or minimum perplexity to a sentence

UNKNOWN WORDS

- In a closed vocabulary language model, there is no unknown words or ***out of vocabulary words (OOV)***
- In an open vocabulary system, you will find new words that are not present in the trained model
- Pick words below certain frequency and replace them as OOV.
- Treat every OOV as a regular word
- During testing, the new words would be treated as OOV and the corresponding frequency will be used for computation
- This eliminates zero probability for sentences containing OOV

CURSE OF DIMENSIONALITY

- A fundamental problem that makes language modeling and other learning problems difficult is the curse of dimensionality
- It is particularly obvious in the case when one wants to model the joint distribution between many discrete random variable
- If one wants to estimate the joint probability distribution of 10 words in a language with a million words as vocabulary, then we need to estimate $10000000^{10} - 1 = 10^{60} - 1$ free parameters

.

Thank You

for your insightful questions, comments and participation in the workshop in all five days made this a memorable one for me.
I wish you all the best and I want you to keep this high level of energy and enthusiasm throughout your life long learning journey

THANKS A TON

PROFESSOR ANAMIKA

Aishwarya and Anupama

THANK YOU

Ramaseshan.nlp@gmail.com