Text Vectorization

Bal Krishna Nyaupane
Assistant Professor
Department of Electronics and Computer Engineering
Paschimanchal Campus, IOE
bkn@wrc.edu.np

Introduction

- Machine learning algorithms operate on a numeric feature space. In order to perform machine learning on text, transform documents into vector representations. This process is called feature extraction or more simply, vectorization, and is an essential first step toward language-aware analysis.

- In Machine Learning, vectorization is a step in feature extraction. The idea is to get some distinct features out of the text for the model to train on, by converting text to numerical vectors.
- The procedure of ***converting raw text data into machine understandable format*** (numbers) is called feature engineering of text data.
- Machine learning and deep learning algorithms' performance and accuracy is fundamentally dependent on the type of feature engineering technique used.

One Hot Encoding

- The traditional method used for feature engineering is One Hot encoding.
- It is a process of converting categorical variables into features or columns and coding one or zero for the presence of that particular category.
- In this technique, we represent each unique word in vocabulary by setting a unique token with value 1 and rest 0 at other positions in the vector.
- In simple words, a vector representation of a one-hot encoded vector represents in the form of 1, and 0 where 1 stands for the position where the word exists and 0 everywhere else.

One Hot Encoding

- ***Disadvantages of One-hot Encoding***

1) The Size of the vector is equal to the count of unique words in the vocabulary.
2) It does not capture the relationships between different words. Therefore, it does not convey information about the context.

```
Text = "I am teaching NLP in Python"
pd.get_dummies(Text.split())
```

	I	NLP	Python	am	in	teaching
0	1	0	0	0	0	0
1	0	0	0	1	0	0
2	0	0	0	0	0	1
3	0	1	0	0	0	0
4	0	0	0	0	1	0
5	0	0	1	0	0	0

Label Encoding vs. One Hot Encoding

- ***We apply One-Hot Encoding when:***
  1) *The categorical feature is not ordinal*
  2) *The number of categorical features is less so one-hot encoding can be effectively applied.*

- ***We apply Label Encoding when:***
  1) *The categorical feature is ordinal (like Jr. kg, Sr. kg, Primary school, high school)*
  2) *The number of categories is quite large as one-hot encoding can lead to high memory consumption.*

Bag-of-Words(BoW)

- It is basic model used in natural language processing. ***BoW*** is a vectorization technique that converts the text content to numerical feature vectors (P. D. Turney. 2002).
- Bag of Words takes a document from a corpus ***and converts it into a numeric vector*** by mapping each document word to a feature vector for the machine learning model.
- A bag of words is a representation of text that ***tells weather word is present in the document or not***. It just ***keep track of word counts*** and ***disregard the grammatical details*** and the ***word order***.
- The disadvantage of ***BoW*** is that it doesn't preserve the word order and does not allow to draw useful inferences for downstream NLP tasks. Bag of word models doesn't respect the semantics of the word.

Bag-of-Words(BoW)

- It involves two operations:

  1) ***Tokenization***
     - Preprocess the data - ***convert text to lower case***, ***remove all punctuations***, ***remove all stop words*** etc.
     - Each word or symbol is called a token. After tokenization we will take unique words from the corpus. Corpus represents the tokens from all the documents and used for the bag of words creation.
  2) ***Vectors Creation:*** In this step construct a vector, which tell whether a word in each sentence is a frequent word or not. If a word in a sentence is a frequent word, we set it as 1, else we set it as 0.

BoW - CountVectorizer

- CountVectorizer is used to ***transform a given text into a vector on the basis of the frequency*** (count) of each word that occurs in the entire text.
- This is helpful when we have multiple such texts, and we wish to convert each word in each text into vectors (for using in further text analysis).
- CountVectorizer ***creates a matrix in which each unique word is represented by a column of the matrix***, and each text sample from the document is a row in the matrix. The value of each cell is nothing but the count of the word in that particular text sample.

BoW - CountVectorizer

```
from sklearn.feature_extraction.text import CountVectorizer
```

```
documemts = ['This is the first document.','This document is the second document.',
             'And this is the third one.','In this way there are so many documents.']
```

```
vectorizer = CountVectorizer()
features = vectorizer.fit_transform(documemts)
```

```
# Printing the identified Unique words along with their indices
print("Vocabulary with index : ", vectorizer.vocabulary_)
print("Sorted Vocabulary: ",sorted(vectorizer.vocabulary_.keys()))
```

```
Vocabulary with index :  {'first': 2, 'document': 0, 'second': 5, 'third': 6, 'one': 4, 'way': 7,
'many': 3, 'documents': 1}
Sorted Vocabulary:  ['document', 'documents', 'first', 'many', 'one', 'second', 'third', 'way']
```

BoW - CountVectorizer

```
# Summarizing the Encoded Texts
print("Encoded Document is:")
print(features.toarray())
```

```
Encoded Document is:
[[0 0 1 0 1 0 1 0 0 0 0 1 0 0 1 0]
 [0 0 2 0 0 0 1 0 0 1 0 1 0 0 1 0]
 [1 0 0 0 0 0 1 0 1 0 0 1 0 1 1 0]
 [0 1 0 1 0 1 0 1 0 0 1 0 1 0 1 1]]
```

N-gram

- n-grams considers sequence of n words in the text; where n is (1,2,3..)
- Example 1-gram, 2-gram for token pair. Unlike BoW, n-gram maintains word order.
- **Example:** A swimmer is swimming in the swimming pool.
- ***Unigram(1-gram):*** A , swimmer , is , swimming , in , the , swimming , pool

- ***Bigram (2-gram):*** A swimmer , swimmer is , is swimming , swimming in

- ***Trigram(3-grams):*** A swimmer is, swimmer is swimming, is swimming in…

TF-IDF

- TF-IDF, ***Term Frequency–Inverse Document Frequency***, is a numerical statistic that's intended to reflect how important a word is to a document. Although it's another frequency-based method, it's not as naive as Bag of Words.
- ***Term Frequency*** is simply the ratio of the count of a word present in a sentence/document, to the length of the sentence/document. Term frequency can be defined as:

$$TF = \frac{Frequency\ of\ word\ in\ a\ document}{Total\ number\ of\ words\ in\ that\ document}$$

TF-IDF

- ***Inverse Document Frequency***
  - It is based on the fact that less frequent words are more informative and important.
  - IDF will measure the rareness of a term. Words like “a,” and “the” show up in all the documents of the corpus, but rare words will not be there in all the documents. So, if a word is appearing in almost all documents, then that word is of no use to us since it is not helping to classify or in information retrieval. IDF will nullify this problem.
  - IDF of each word is the log of the ratio of the total number of rows to the number of rows in a particular document in which that word is present.

TF-IDF

- ***IDF is represented by formula:***

$$IDF = \log(\frac{Total\ number\ of\ documents}{Documents\ containing\ word\ W})$$

- ***The final TF-IDF score:***

$$TF - IDF = TF * IDF$$

TF-IDF

Example sentences :-

- A: This pasta is very tasty and affordable.
- B: This pasta is not tasty and is affordable.
- C: This pasta is very very delicious.

Let's consider each sentence as a document. Here also our first task is tokenization (dividing sentences into words or tokens) and then taking unique words.

Word	TF				TF * IDF		
	A	B	C		A	B	C
this	1/7	1/8	1/6	log(3/3)=0	0	0	0
pasta	1/7	1/8	1/6	log(3/3)=0	0	0	0
is	1/7	2/8	1/6	log(3/3)=0	0	0	0
Very	1/7	0	2/6	log(3/2)=0.176	0.025	0	0.058
tasty	1/7	1/8	0	log(3/2)=0.176	0.025	0.022	0
and	1/7	1/8	0	log(3/2)=0.176	0.025	0.022	0
affordable	1/7	1/8	0	log(3/2)=0.176	0.025	0.022	0
not	0	1/8	0	log(3/1)=0.477	0	0.0596	0
delicious	0	0	1/6	log(3/1)=0.477	0	0	0.079

Word Embeddings

- ***From Wikipedia:*** In natural language processing (NLP), word embedding is a term used for the ***representation of words*** for text analysis, typically ***in the form of a real-valued vector that encodes the meaning of the word*** such that the ***words that are closer in the vector space*** are expected to be ***similar in meaning***.

- Word embedding is the collective name for a set of language modeling and feature learning techniques in natural language processing (NLP) ***where words or phrases from the vocabulary are mapped to vectors of real numbers.***

Word Embeddings

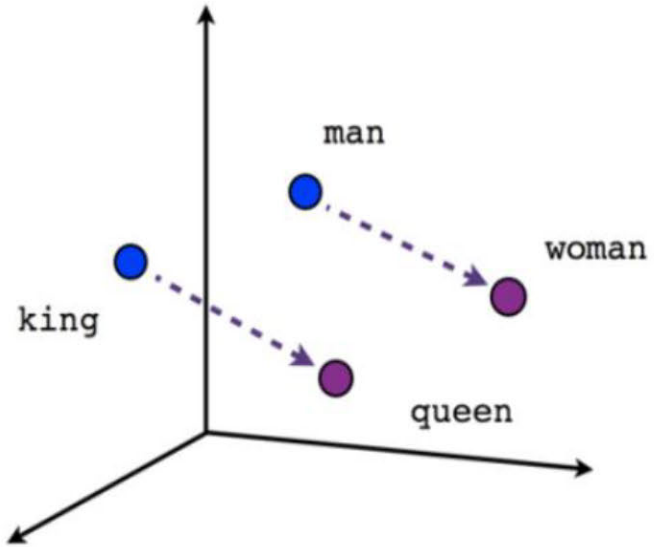
- Word embeddings are ***vectorized, fixed-length, distributed, dense representations of words*** that interpret a word's textual meaning by mapping it to a vector of real values.
- ***Fixed-length vectors*** - all the words in vocabulary would be represented by a vector(of real numbers) of a fixed predefined size that we decide on.
- Word embeddings of size 3 would look like: ***'cold' — [0.2, 0.5, 0.08], 'house' — [0.05, 0.1, 0.8]***
- **Dense representations:** If vocabulary size is 5 with the words — {cat, food, house, milk, water}, the cat will be encoded as a binary vector of [1, 0, 0, 0, 0] and milk would be [0, 0, 0, 1, 0] and so on.

Word Embeddings

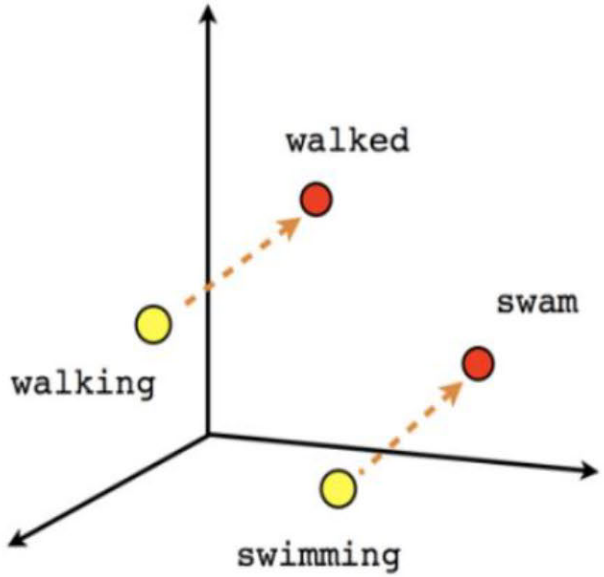
- Word embeddings have a capability of ***capturing semantic and syntactic relationships between words*** and also the context of words in a document.
- ***Word embedding-*** mapping of words in vector space.
- ***Word2vec is the technique*** to implement word embeddings.

- Every word in a sentence is dependent on another word or other words. If you want to ***find similarities and relations between words*** ,we have to capture word dependencies. ***By using Bag-of-words and TF-IDF techniques we can not capture the meaning or relation of the words from vectors***. ***Word2vec constructs such vectors called embeddings.***

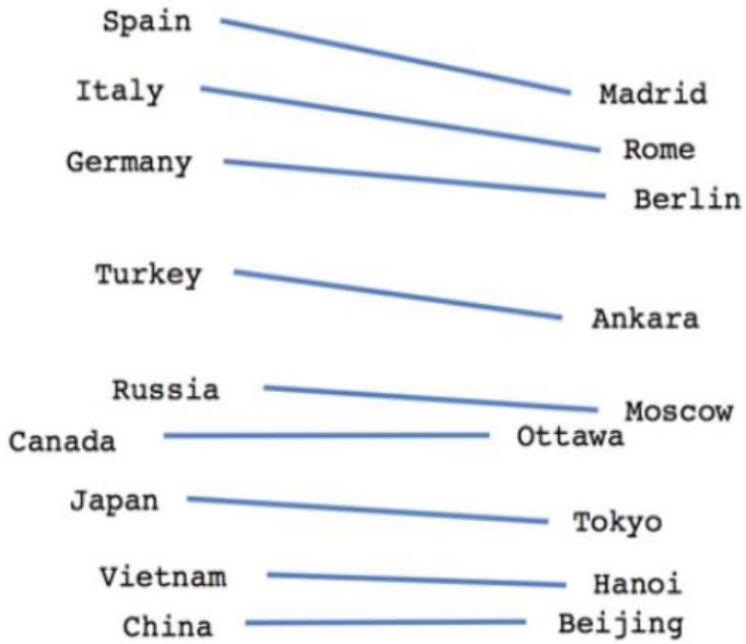
Word Embeddings



Male-Female



Verb tense



Country-Capital

Word2vec

- The word2vec objective function causes the words that occur in similar contexts to have similar embeddings.

Example: The **kid** said he would grow up to be superman.
The **child** said he would grow up to be superman.

The words kid and child will have similar word vectors due to a similar context.

Word2vec

- Instead of entire documents, ***Word2Vec*** uses words ***k*** positions away from each center word.
  - These words are called **context words**.
- Example for ***k=3***:
  - "It was a bright cold day in April, and the clocks were striking".
  - Center word: red (also called focus word).
  - Context words: blue (also called target words).
- Word2Vec considers all words as center words, and all their context words.
- There are actually two ways to implement word2vec: ***CBOW (Continuous Bag-Of-Words) and Skip-gram.***

Word2vec

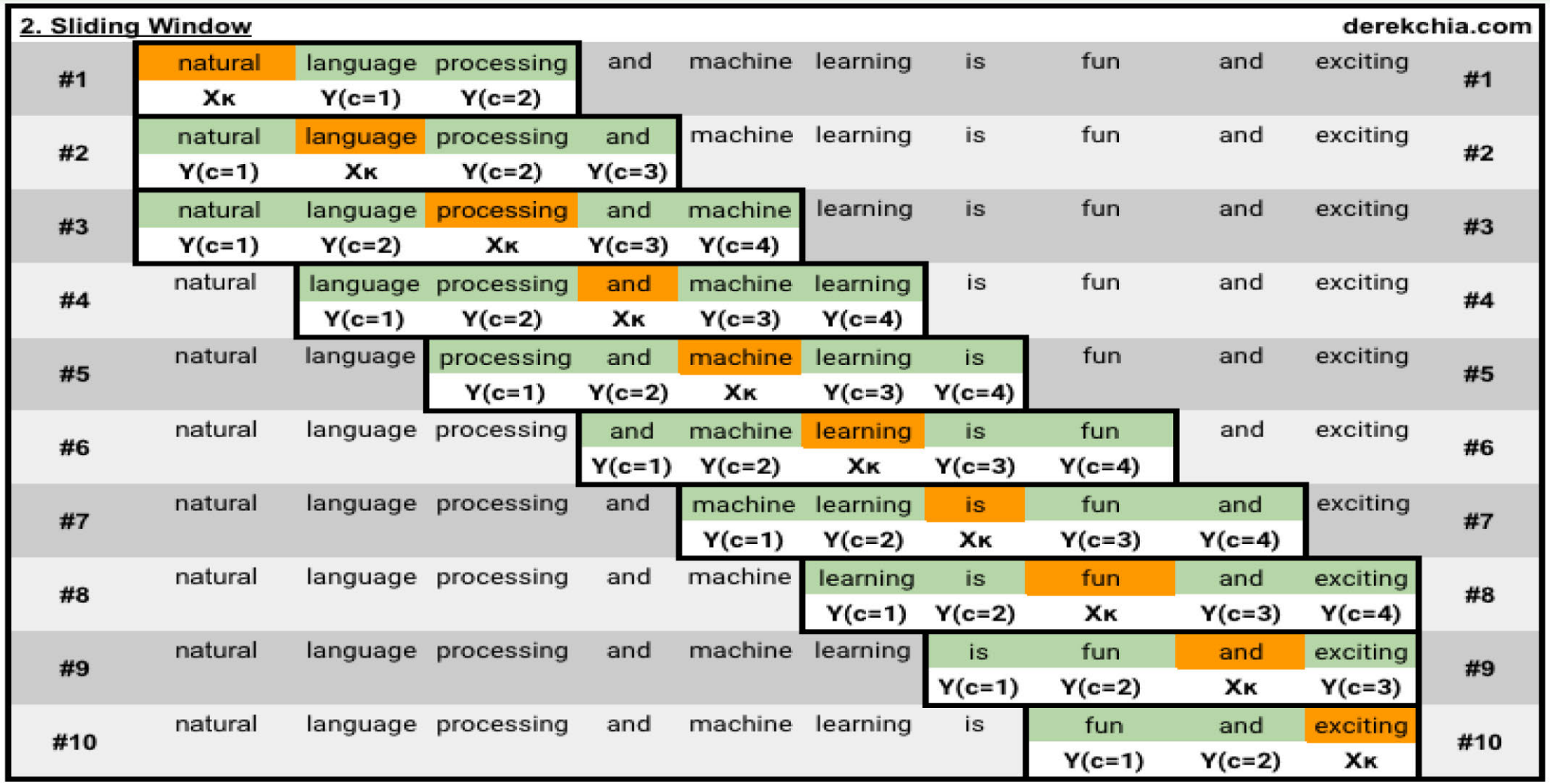
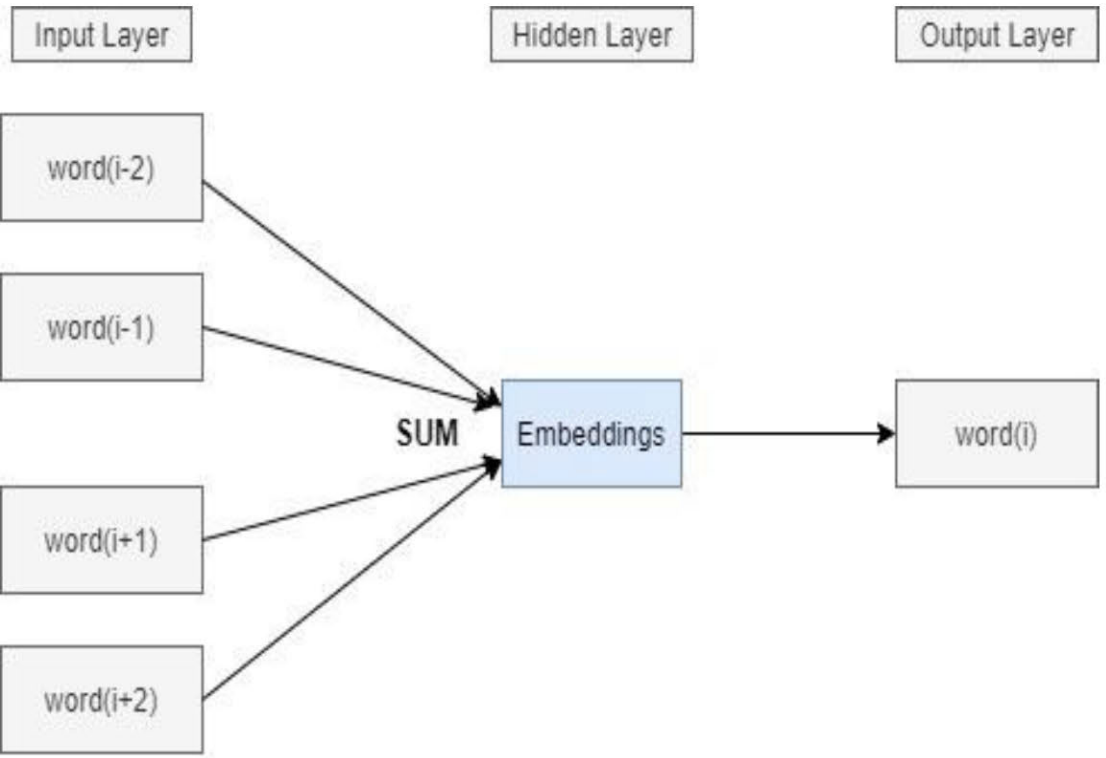


Figure: With a window_size of 2, the target word in orange and context words in green

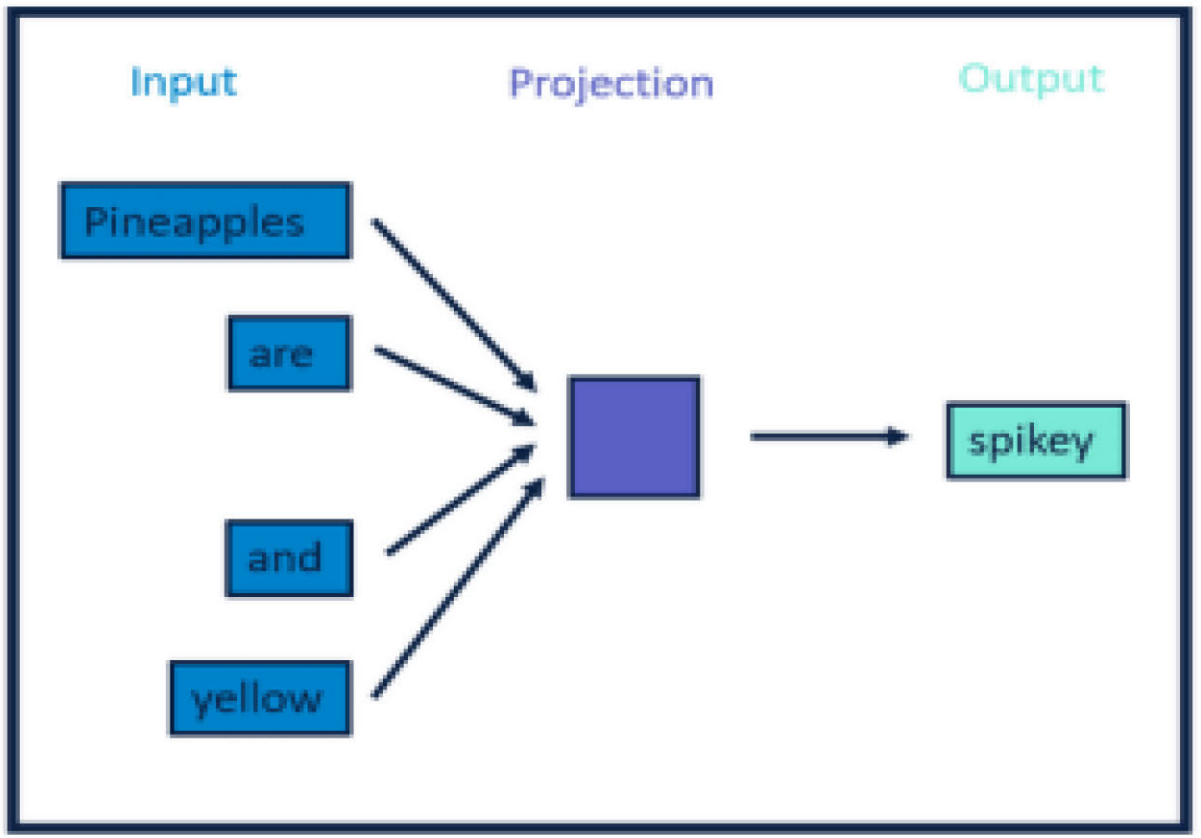
Word2vec: CBOW

- The CBOW architecture comprises a deep learning classification model in which we take in context words as input, X, and try to predict our target word, Y.
- In the CBOW model, try to predict the distributed representation of the target word (middle word) from the context words (surrounding words) which lie on either side of the target word within the context window.
- For example, if we consider the sentence – "***Word2Vec has a deep learning model working in the backend***.", there can be pairs of context words and target (center) words. A context window of size 2 would have the following pairs - (***context_window***, ***target_word***) , then we will have pairs like ***([deep, model], learning)***, ***([model, in], working), ([a, learning), deep)*** etc.
- The deep learning model would try to predict these target words based on the context words.

Word2vec: CBOW



CBOW



CBOW

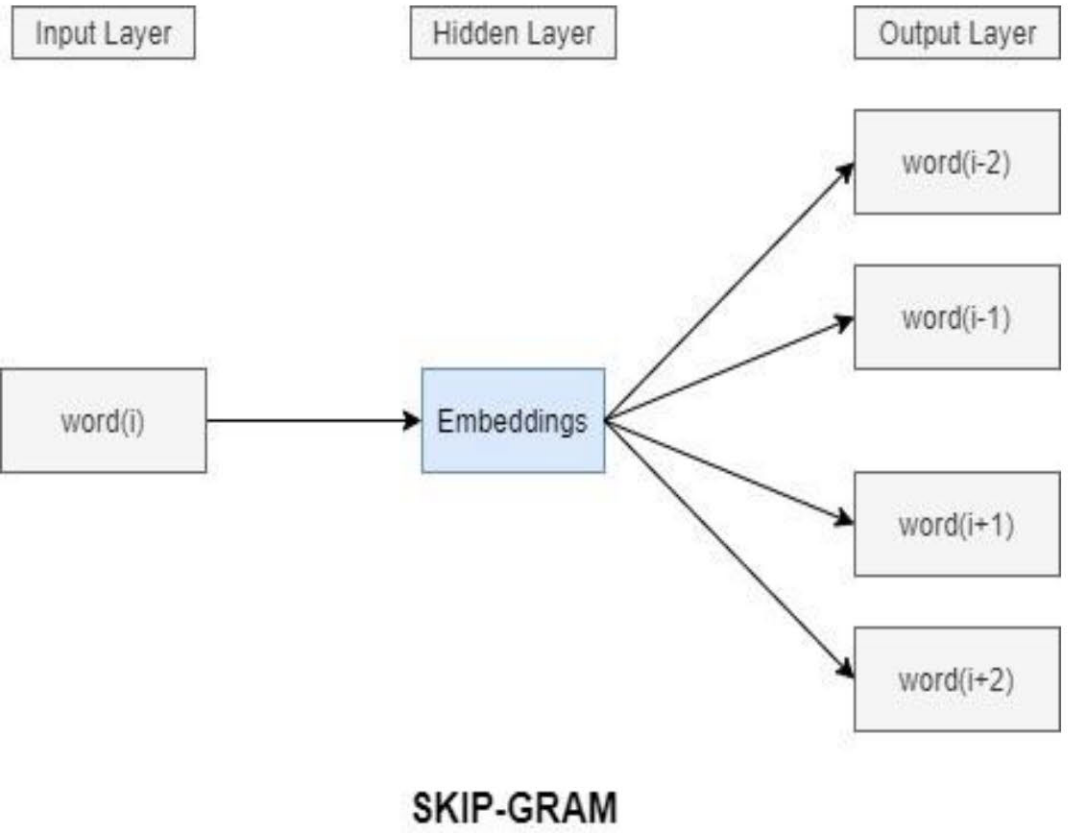
Word2vec: Skip-Gram

- The Skip-gram model is similar to the CBOW model, but instead of predicting the current word given the context, take the center word from the window size words as an input and context words (neighbor words) as outputs.

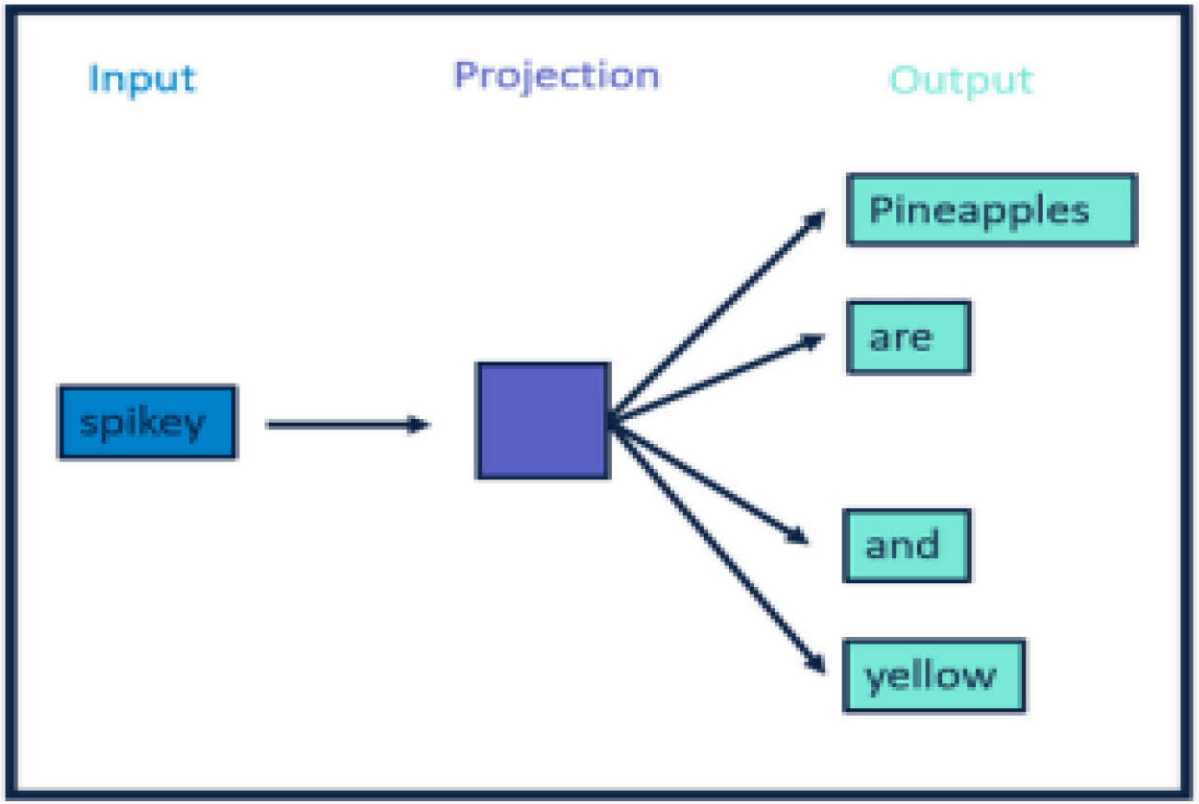
- Word2vec models predict the context words of a center word using skip-gram method.

- For example, ***"Pack my box with five dozen liquor jugs"***, a context window of size 2 would have the following pairs - ***(current_word, context_window) -*** (dozen, [five, liquor]), (liquor, [dozen, jugs]) and so on.

Word2vec: Skip-Gram

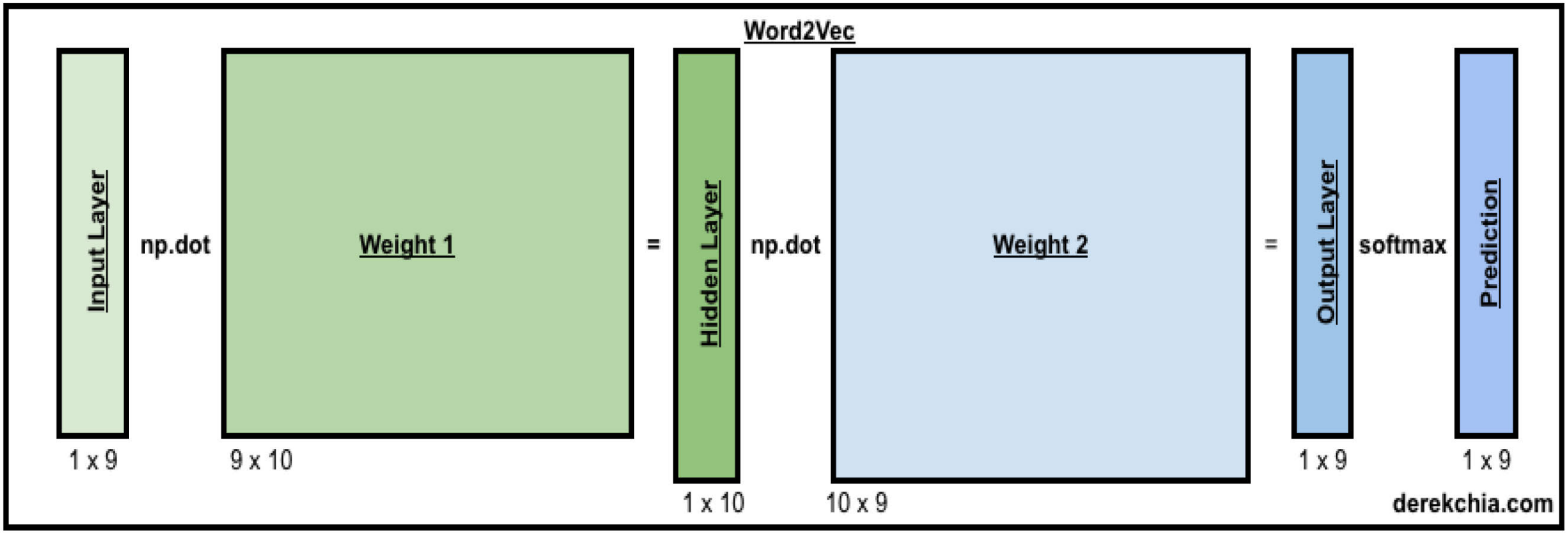


SKIP-GRAM



Skip-gram

Skip-Gram network architecture



Calculate hidden layer, output later and SoftMax

4. Skip-gram Model Architecture derekchia.com

Forward Pass for #1

Parameters - Embedding size 10

Random initialisation (Range) -1 to 1

Calculate Hidden Layer

#	Token	Input - w_t
0	natural	1
1	language	0
2	processing	0
3	and	0
4	machine	0
5	learning	0
6	is	0
7	fun	0
8	exciting	0

1 x 9

np.dot

Weight 1 - W1

0.236	**-0.962**	**0.686**	**0.785**	**-0.454**	**-0.833**	**-0.744**	**0.677**	**-0.427**	**-0.066**
-0.907	0.894	0.225	0.673	-0.579	-0.428	0.685	0.973	-0.070	-0.811
-0.576	0.658	-0.582	-0.112	0.662	0.051	-0.401	-0.921	-0.158	0.529
0.517	0.436	0.092	-0.835	-0.444	-0.905	0.879	0.303	0.332	-0.275
0.859	-0.890	0.651	0.185	-0.511	-0.456	0.377	-0.274	0.182	-0.237
0.368	-0.867	-0.301	-0.222	0.630	0.808	0.088	-0.902	-0.450	-0.408
0.728	0.277	0.439	0.138	-0.943	-0.409	0.687	-0.215	-0.807	0.612
0.593	-0.699	0.020	0.142	-0.638	-0.633	0.344	0.868	0.913	0.429
0.447	-0.810	-0.061	-0.495	0.794	-0.064	-0.817	-0.408	-0.286	0.149

9 x 10

=

Hidden Layer - h

0.236
-0.962
0.686
0.785
-0.454
-0.833
-0.744
0.677
-0.427
-0.066

1 x 10

Calculate y_pred

Hidden Layer - h

0.236
-0.962
0.686
0.785
-0.454
-0.833
-0.744
0.677
-0.427
-0.066

1 x 10

np.dot

Weight 2 - W2

-0.868	-0.406	-0.288	-0.016	-0.560	0.179	0.099	0.438	-0.551
-0.395	0.890	0.685	-0.329	0.218	-0.852	-0.919	0.665	0.968
-0.128	0.685	-0.828	0.709	-0.420	0.057	-0.212	0.728	-0.690
0.881	0.238	0.018	0.622	0.936	-0.442	0.936	0.586	-0.020
-0.478	0.240	0.820	-0.731	0.260	-0.989	-0.626	0.796	-0.599
0.679	0.721	-0.111	0.083	-0.738	0.227	0.560	0.929	0.017
-0.690	0.907	0.464	-0.022	-0.005	-0.004	-0.425	0.299	0.757
-0.054	0.397	-0.017	-0.563	-0.551	0.465	-0.596	-0.413	-0.395
-0.838	0.053	-0.160	-0.164	-0.671	0.140	-0.149	0.708	0.425
0.096	-0.995	-0.313	0.881	-0.402	-0.631	-0.660	0.184	0.487

10 x 9

=

Output Layer

1.258
-1.369
-1.828
1.196
0.545
1.113
1.333
-1.528
-2.335

1 x 9

Softmax - y_pred

0.218
0.016
0.010
0.205
0.107
0.189
0.235
0.013
0.006

1 x 9

Word2vec

- ***Skip gram***
  - *In this input is center word and output is context words (neighbor words).*
  - *Works well with small datasets.*
  - *Skip-gram identifies rarer words better.*
- ***CBow***
  - *In this context or neighbor words are input and output is the center word.*
  - *Works good with large datasets.*
  - *Better representation for frequent words than rarer.*

Thank You
???

- ***References:***

1) Foundations of Statistical Natural Language Processing - Christopher D. Manning.
2) Text Data Management and Analysis - A Practical Introduction to Information Retrieval and Text Mining
3) Natural Language Processing using Python- Apress (2019)
4) Applied Text Analysis with Python O'Reilly Media (2018)
5) Natural Language Processing Python and NLTK-Packt Publishing (2016)
6) https://towardsdatascience.com
7) https://www.analyticsvidhya.com