Appendix

4

Vector Space Model

LEARNING OBJECTIVES

After studying this appendix, you will be able to:

- **LO 1** Understand the basic concept of vectors and related concepts like: (1) vector space, (2) components of vectors, (3) dimensions and 'basis' of vectors, and (4) representation of vectors as a 'single column matrix'. Understanding of vectors in n-dimension, i.e., R^n space
- **LO 2** Understand the word2vec algorithm. (word2vec algorithm implements what is called the neural net language model)
- **LO 3** Grasp the concept of 'words of a language' as a 'one-hot vector'
- **LO 4** Understand distributed representation or Word embedding
- **LO 5** Understand term frequency (tf) and Inverse document frequency (idf)
- **LO 6** Understand how a typical word embedding algorithm works
- **LO 7** Understand the basics of 'Topic modelling'

This appendix discusses the Vector space models in natural language processing.

A4.1 INTRODUCTION

This appendix is not on actual programming but rather on the concept of vectors as used in language models.

A4.2 VECTORS IN 2D AND 3D

Vectors are generally introduced in high school physics and explained as a contrast to scalars. So a vector in physics has 'direction' while a scalar only has magnitude. In physics one deals with variables like 'distance', 'velocity', 'force', etc., so one generally uses 2D or 3D vectors. Here one talks of vectors in terms of unit vectors $\hat{i}$, i.e., i(cap), $\hat{j}$, i.e., j(cap) and $\hat{k}$, i.e., k(cap).

Further you may represent each point in Euclidean space as a sequence of numbers, i.e., (x,y,z). Note the word 'sequence' which means that the 'order' of x, y and z are important. So a number like say (1,2,3) will represent a different point than say (3,2,1) or (1,3,2).

There are a few other concepts which may be discussed here. (***Note:*** The aim here is not to give 'formal definition' of the terms but just to give an 'insight' into the concepts. The discussion is in context of 3D vectors and then it is extended to higher dimensions.)

- **Vector space:** Now a combination of the 3 unit vectors multiplied by scalars will generate various points in 3D. For example if a point say A = (2,3,4), then it could be represented in vector space by $\vec{A} = 2i + 3j + 4k$, where $\vec{A}$ can be thought of as a line drawn from the origin to point A. So in general one can write $\vec{A} = xi + yj + zk$, where by varying the values for x, y and z you can generate an infinite number of points in 3D. And these generated points create a 3-D 'vector space'.
- **Orthogonality of vectors, dot product and cosine similarity:** Two vectors are said to be 'orthogonal' or 'perpendicular' to each other if in 2D or 3D, the angle between them is 90°. Connected to this is the concept of 'dot product' of two vectors. Dot product of 2 vectors say $\vec{A}$ and $\vec{B}$ can be represented as: $\vec{A} \cdot \vec{B} = A \cdot B \cdot \cos\theta$, where θ is the 'angle' between them. Note that then the dot product of the unit vectors will be 0 because the 3 unit vectors are perpendicular to each other so $\theta = \pi/2$. The dot product of 2 vectors also gives what is called the 'cosine similarity' between two vectors. So if two vectors are parallel, then they have same 'orientation', i.e., same 'direction', so the angle between them will be 0, so $\cos\theta = 1$.
- **Components of vectors:** A vector can be thought of as having 'components'.
- **Cylindrical coordinates and spherical coordinates:** 3D space can be represented by other coordinate systems like cylindrical and spherical coordinate systems.
- **Dimension and basis:** Here the focus is not on the definitions of the concepts but rather on 'understanding' them. In the 3D vector space discussed above, the 3 unit vectors i, j and k can be thought of as 3 basis of this vector space and this vector space also has a dimension of 3. In fact, these may be called the 'standard basis' of the vector space. One could have more than one standard basis also. For example, if you chose to represent the same vector space in cylindrical or spherical coordinates, the 'basis' would change.

  Note that you could chose 'non-standard' basis also to create a vector space. Detailed discussion is beyond the scope of this book.
- **Representing a vector as a single column matrix:** A vector can be represented as a single column matrix also. So you could write $\vec{A} = xi + yj + zk$ and unit vectors in column form as shown Figure A4.1.

$\vec{A} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$	$\vec{i} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$	$\vec{j} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$	$\vec{k} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$
$\vec{A} = xi + yj + zk$	$\vec{i} = 1*i + 0*j + 0*k$	$\vec{j} = 0*i + 0*j + 1*k$	$\vec{k} = 0*i + 0*j + 1*k$

FIGURE A4.1 A 3D vector and also unit vectors i, j and k in matrix form

A4.3 VECTORS IN N-DIMENSION AND R^N SPACE

In general, have a n-dimension vector. Of course it is difficult to imagine a vector space of more than 3 dimensions. A vector of dimension n is a sequence of n numbers where the order of the numbers is important.

One good way to conceptualize an n-dimension vector is to think of it as a 'table'. Consider a table representing 4 hypothetical vehicles V1 to V4. The table also has columns (or features): Max speed, Fuel, Mileage, Wheels, Year and Model.

TABLE A4.1 Four hypothetical cars V1 to V4 as 4 rows and their 6 'characteristics' as columns. The rows represent the number of vectors and the columns represent the number of dimensions of each vector

	Max speed	Fuel	Mileage	Wheels	Year	Model
V1 →	100	petrol	12	4	1990	M111
V2 →	75	diesel	8	6	2000	M112
V3 →	60	diesel	6	12	2005	M113
V4 →	70	petrol	14	2	2008	M114

Now each row of the table can be thought of as a 'vector' with a name. So $\overrightarrow{V1}$ = is [100 petrol 12 4 1990 M111]^T.

Further each column of the table represents a dimension or a 'feature' or 'basis' of the vector. You can think of each column as representing a dimension in an n-dimension space. So here the vector space has 6 dimensions. Another thing to note is that each column represents what may be called a 'basis' for that dimension of the vector. For example one of the 'basis' of the above table is 'Milage'. Now if you chose to represent the 'Milage' in km/litre then you have one basis for Milage. However should you chose to use a different unit say miles/gallon, then you would have a different 'basis' for the dimension 'Milage'. Further note that there may be a situation where a 'dimension' depends upon some combination of other dimensions. Should this be the case, you can say that such a dimension (or column) would not be considered as a 'basis' of the vector. So for example if a vector has 6 dimensions but one of the dimensions is some linear combination of some or all of the other dimensions, then the vector will have 5 and not 6 basis.

Further if you chose to remove some of the columns you are doing what is called 'Dimension reduction'.

A4.4 Word to Vector (word2vec)

The word2vec algorithm implements what is called the neural net language model.

A good introduction of the Word2vec model is available here[1].

A4.5 Words as 'one-hot' Vector

Suppose you had a vocabulary (i.e., all unique words) of n words, then if you have a vector of dimension n, then it could be used to represent each word in the vocabulary.

Consider a sentence say 'The car is red in color'. Further consider a hypothetical dictionary of 6 words, i.e., [The], [car], [is], [red], [in], [color]. Then these words could be encoded as vectors as shown in Table A4.2.

Table A4.2 The sentence 'The car is red in color' has been converted into 6 one-hot vectors.

Word	The	car	is	red	in	color	Vector
The	1	0	0	0	0	0	The $[1\ 0\ 0\ 0\ 0\ 0]^T$
car	0	1	0	0	0	0	car $[0\ 1\ 0\ 0\ 0\ 0]^T$
is	0	0	1	0	0	0	is $[0\ 0\ 1\ 0\ 0\ 0]^T$
red	0	0	0	0	0	0	red $[0\ 0\ 0\ 1\ 0\ 0]^T$
in	0	0	0	0	1	0	in $[1\ 0\ 0\ 0\ 1\ 0]^T$
color	0	0	0	0	0	1	color $[1\ 0\ 0\ 0\ 0\ 1]^T$

Note the superscript T which indicates that it is a transverse, i.e., a column and not a row vector. The above table is an example of one-hot vector coding, where each word in the vocabulary has a dimension to it. Note that in each vector, the values of all dimensions except one are 0. So 'one-hot' encoding is also called sparse encoding or sparse vector.

A4.6 Distributed Representation or Word Embedding

Now consider the distributed encoding (also called word embedding). You need to understand the idea first. A word in a vocabulary has certain 'features' which makes it related to other words. For example, this relation could be of following types:

- One relation could be the 'context' of the world, meaning thereby that a 'word is known by the company it keeps'.
- Another relation of a word to other words in the vocabulary could be say the 'synonyms' of the word. Similarly a word is also related to its 'antonyms'.
- Similarly a word also has some relation with its 'lemma', 'stem', 'inflections', etc.

This scheme of 'distributed representation' has many advantages. For example, there is a greater chance that a word like 'swim' would have a distributed vector which is closer to vector for say 'float'.

[1]See: https://arxiv.org/pdf/1301.3781

Another advantage is that by 'training' of the model, it is possible to bring closer two word vectors which are semantically related but vectors have less 'cosine similarity', i.e., the angle between them is large.

In Figure A4.2, you have 2 vectors $\vec{A}$ and $\vec{B}$. Also vector $\vec{C} = (\vec{B} - \vec{A})$.

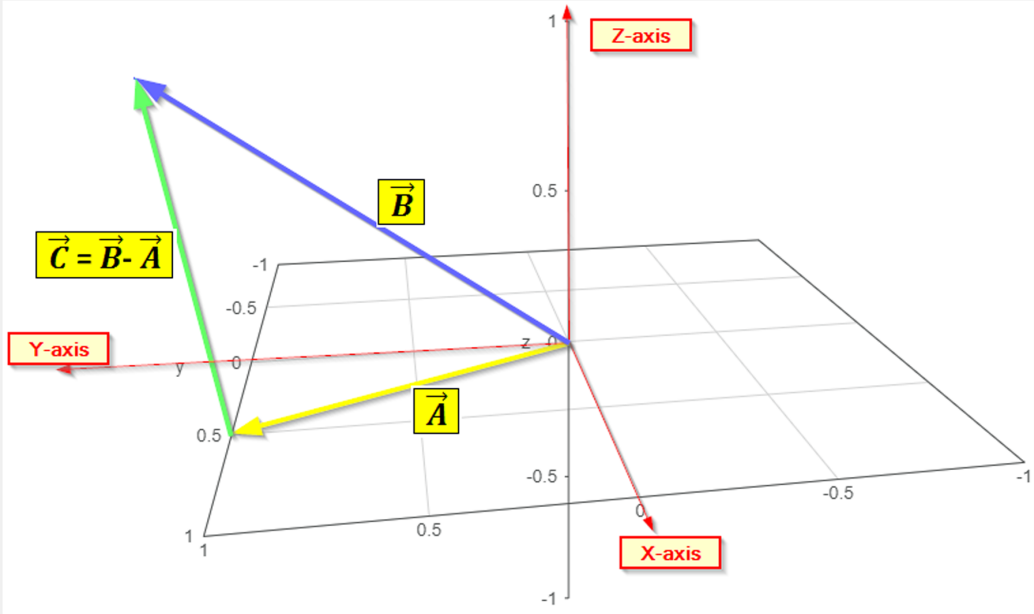

FIGURE A4.2 Two vectors $\vec{A}$ and $\vec{B}$. It also shows a third vector $\vec{C} = \vec{B} - \vec{A}$.

How do you bring together the 2 vectors? This can be explained with help of an example. Suppose you have 2 vectors say $\vec{A}$ and $\vec{B}$. Suppose by training you want to bring $\vec{A}$ closer to $\vec{B}$. If you add a fraction of $(\vec{B} - \vec{A})$ say $\delta(\vec{B} - \vec{A})$ to $\vec{A}$, then it can be shown that the new vector, i.e., $\vec{A} + \delta(\vec{B} - \vec{A})$ will be closer to $\vec{B}$. This can be shown in Figure A4.3.

Another example could be suppose you have $\vec{R}$ for Russia, i.e., $\vec{R} = \overrightarrow{\text{Russia}}$, $\vec{M}$ for Moscow, i.e., $\vec{M} = \overrightarrow{\text{Moscow}}$. Also you have $\vec{C} = \overrightarrow{\text{Capital}}$ for capital (meaning capital of a country). Then in the vector space you could have:

$\overrightarrow{\text{Russia}} + \overrightarrow{\text{Capital}} = \overrightarrow{\text{Moscow}}$ or $\overrightarrow{\text{Capital}} = \overrightarrow{\text{Moscow}} - \overrightarrow{\text{Russia}}$. So knowing $\vec{R}$ for $\overrightarrow{\text{Russia}}$, $\vec{M}$ for $\overrightarrow{\text{Moscow}}$, you could perhaps get $\vec{C}$ for $\overrightarrow{\text{Capital}}$. Now suppose you have to find the capital of another country say France and vector for France is say $\vec{F} = \overrightarrow{\text{France}}$. Then it can be argued that the capital of France can be represented by vector $\vec{F} - \vec{C}$ or one can say vector for capital of France, i.e., Paris is $\vec{P} = \overrightarrow{\text{Paris}}$. So $\vec{P} = \vec{F} - \vec{C} = \vec{F} + \vec{R} - \vec{M}$. This is shown in Figure A4.4.

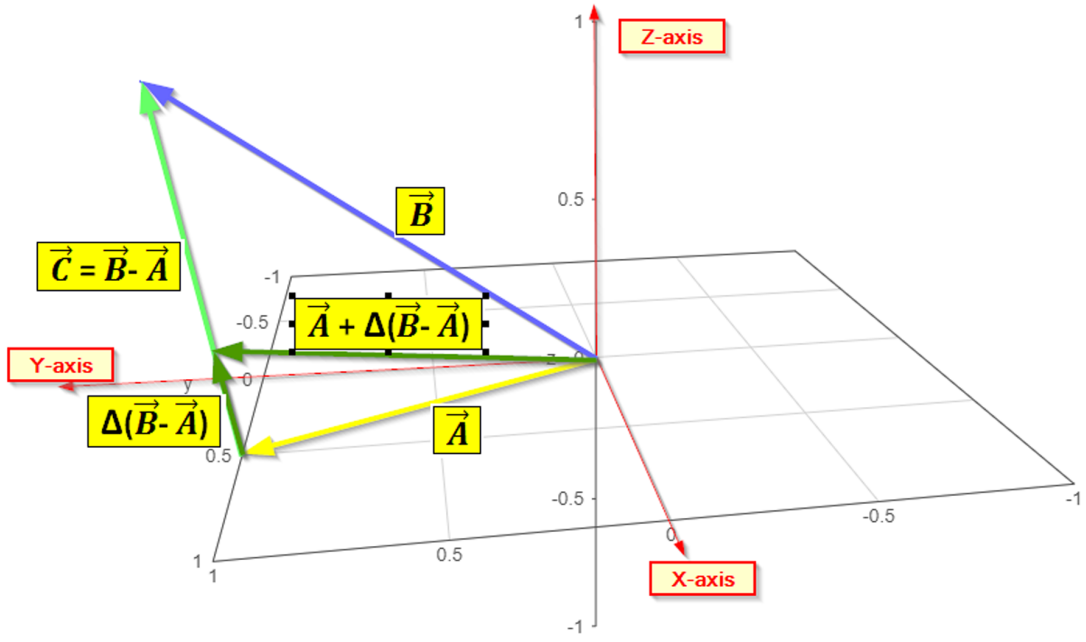

FIGURE A4.3 $\vec{A} + \delta(\vec{B} - \vec{A})$ will be closer to $\vec{B}$ than the original vector $\vec{A}$.

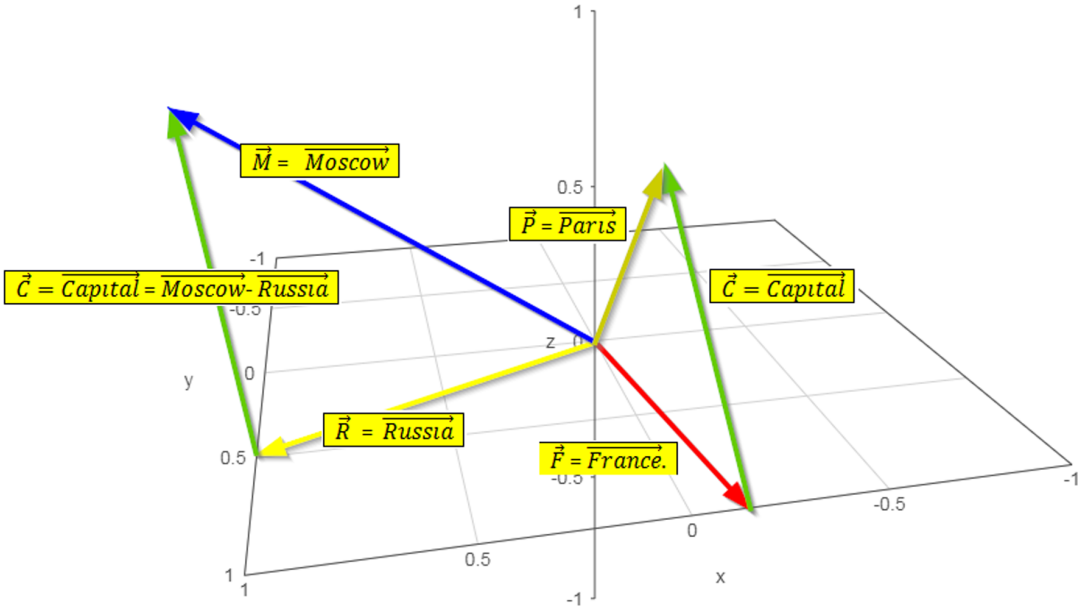

FIGURE A4.4 If you have vectors for a country and its capital, then the vector distance between them could be used to find the capital of another country given the vector for that country name.

This argument can be extended to other situations. Suppose you know vectors for say $\overrightarrow{\text{MAN}}$ and $\overrightarrow{\text{WOMAN}}$. Then you can say $\overrightarrow{\text{MAN}} + \overrightarrow{\text{OPPOSITE}} = \overrightarrow{\text{WOMAN}}$ or you have:

$$\overrightarrow{\text{OPPOSITE}} = \overrightarrow{\text{WOMAN}} - \overrightarrow{\text{MAN}}$$

Now suppose you know $\overrightarrow{\text{UNCLE}}$ and want to find $\overrightarrow{\text{AUNT}}$. Then you could get

$$\overrightarrow{\text{AUNT}} = \overrightarrow{\text{UNCLE}} + \overrightarrow{\text{OPPOSITE}} = \overrightarrow{\text{UNCLE}} + \overrightarrow{\text{WOMAN}} - \overrightarrow{\text{MAN}}$$

A4.7 Vector Space Models

Vector space model or term vector model is an algebraic model for representing text documents (and any objects, in general) as vectors of identifiers, such as, index terms.

A4.8 Term Frequency (tf), Inverse Document Frequency (idf)

Term frequency (tf) is the number of times a term (say t) appears in the text/document. However, it may not always be a good representation of the text because it could lead to abnormally high values for common words. Many times the term frequency may be 'normalized' by dividing the term frequency by the total number of terms in the document.

$$\text{TF(t)} = \left(\frac{\text{(Frequency count of the term t in given text)}}{\text{Total number of terms in the given text}} \right)$$

Inverse document frequency measures how important a term is. The presumption is that if the term is appearing in a large number of documents then it is less important (since it is more likely to be a common word). So here one may consider a term which is appearing in fewer documents in a collection of documents to be more important.

$$\text{IDF(t)} = \log\left(\frac{\text{(Total number of documents)}}{\text{Number of documents with the term t in it}} \right)$$

Example: Suppose a document has 50 terms in it and a particular term t has a frequency of say 2 in it. Then the tf for t, i.e., tf(t) = 2/50 = 0.04. Further suppose that the collection has say 1000 documents and the term t appears in 10 out of them, then the IDF(t) = log(1000/10) = 2 (assuming log to base 10). Then tf-idf for the term t = $0.04 \times 2 = 0.08$

A4.9 Understanding How a Typical Word Embedding Algorithm Works

A typical vector model has what is called a 'bag of words'. In NLP, a bag of words can be thought of as a collection of all unique words along with their frequency in the text (text could be a sentence, a document or a collection of documents). However, in most applications common words (also called stop words) are removed from the collection.

There are two parts in the working of a typical word embedding algorithm. They are:

- The actual working phase
- The training phase

The actual working of the algorithm is discussed first.

A4.10 Working of the Algorithm with Only 1 Context Word (bigram)

Consider the scenario, where given the context word (also called history word) one tries to guess the next word. As an example: given the word 'brave', there should be higher probability of getting a word like 'men' rather than getting an seemingly unrelated word like say 'stone'.

Note here you have a two word context, i.e., a bigram.

One can lay down the terminology as follows:

- Let V = vocabulary size (i.e., all unique words in the vocabulary). So V could be of order of 10^4 to 10^5.
- X = One hot encoded vector for the input word. So X is the vector for the word whose context words are to be found. So the vector X (being hot encoded) will have one entry 1 and all others 0. So vector $\vec{X}$ would be something like $\vec{X} = (0, 0, 0, ..., 1, 0, 0, ...)^T$. Again note the T which indicates that this is actually a column vector, but shown as transpose of a row vector to save space. Also note that $\vec{X}$ is a one hot encoded vector and it has dimension V.
- Let N = Number of dimensions chosen to represent a word in the vocabulary. Note N < V. Note that N also represents the number of neurons in the hidden layer.
- $W_{V \times N}$ = This is a matrix on the input side. It has V rows (one row for each word in the dictionary) and N columns (where you have reduced the original V to N). N is of order say 300 to 500. So this means that you are reducing your vocabulary to around 300 to 500 'features'.

So this matrix would look something like this:

$\begin{bmatrix} w11 & \cdots & w1N \\ \vdots & \ddots & \vdots \\ wV1 & \cdots & wVN \end{bmatrix}$	Note this matrix has V rows. So each row corresponds to each word in the vocabulary. Further this matrix has N columns. Here N is the number of 'features' or dimensions that you have chosen to represent the word.

This conversion or transformation of a one-hot encoded vector to a word (taking the word 'brave') embedded vector can be shown as follows:

If you look at Figure A4.5 closely then you will notice that the input matrix has acted like a lookup table. Each row of this lookup table represents an N dimension word embedded code for each word in the vocabulary.

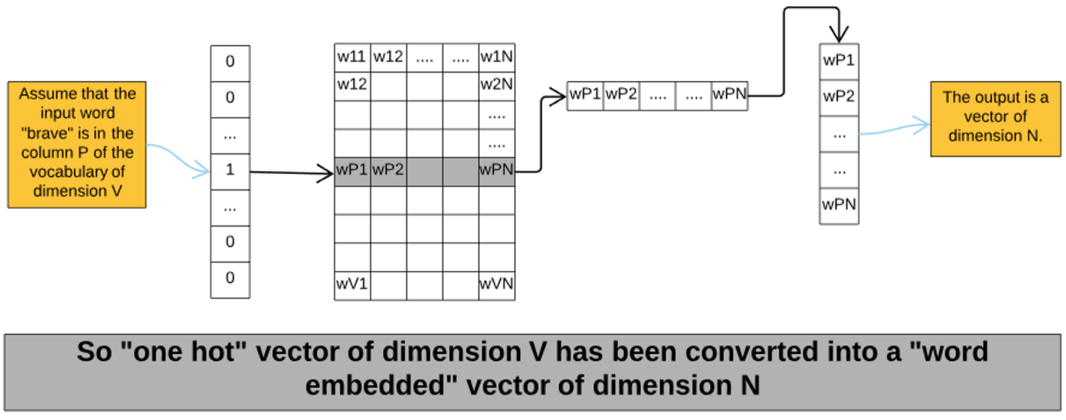

Figure A4.5 'One-hot embedded vector' converted into a vector of dimension N.

- Y = One hot encoded vector for the word likely in context of the input word vector X. Y also will have one 1 and all other entries 0. So $\vec{Y}$ is also something like $\vec{Y}$ = (0, 0, 0, ..., 1, 0, 0, ...)^T.
- $W'_{N \times V}$ = This is a matrix on the output side. What it does is that it takes a 'word embedded' vector representation of the input word (created by the input matrix W) and converts it into a hot-embedded vector.

  So what the matrix W' does it that it converts a word embedded vector say U back to a one-hot embedded vector. (The actual process is more complex, but for the present discussion, this is a sufficiently good scenario).

  This conversion from a word embedded vector to a hot embedded vector is shown in Figure A4.6.

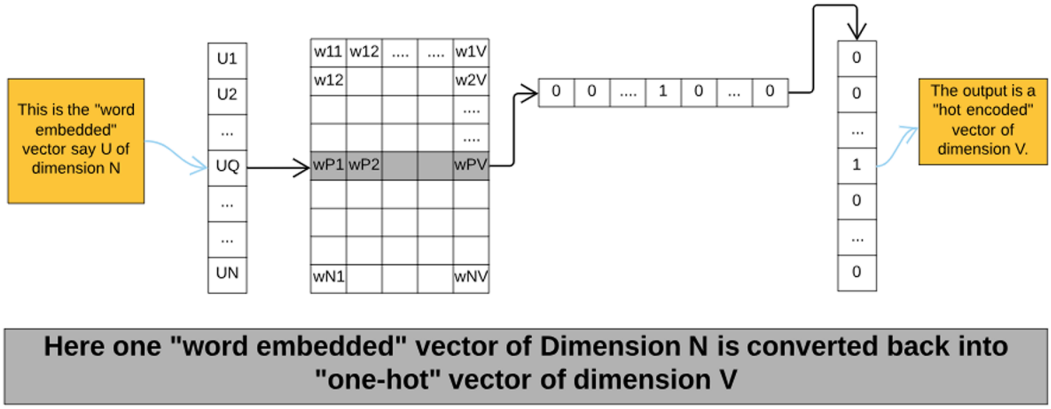

FIGURE A4.6 A vector represented by an 'embedded vector of dimension N' is converted back into a 'one hot embedded vector'.

- Activation functions: For the current purpose you can think of an activation function as some kind of a classifier. Some common examples of activation functions are sigmoid function and softmax function.

A4.11 WORKING OF THE ALGORITHM WITH MORE THAN 1 CONTEXT WORDS

In the discussion above, there was an example of only 1 context word. So given a word like 'brave', one tried to guess the next word. But what if you have more than 1 context word. Suppose you have 3 context words say 'Writing on the' and you have to guess the next word (which should be wall, but could be paper, scroll, book, etc.). Here you will have 3 'one-hot' vectors, i.e., 1 for each of the words 'Writing', 'on' and 'the'. Then you will get a N dimension word embedded vector for each of these three words as you did earlier. Then you can 'average' out the vectors for the three words using some function. Finally you take this 'averaged out' word vector and find its output one-hot vector.

A4.12 TOPIC MODELLING

The concept of topic modelling can be best understood by an example.

Consider the text shown in Figure A4.7.

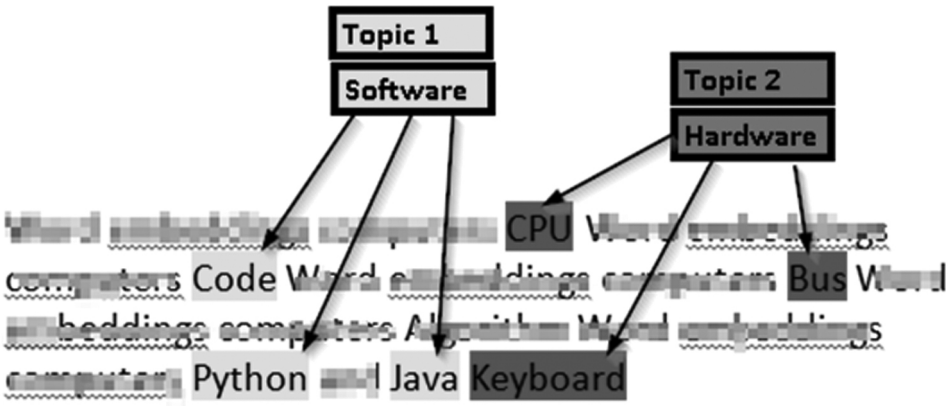

FIGURE A4.7 Words in a piece of text which belong either to Topic1, i.e., Software or Topic2, i.e., Hardware. The rest of the words which don't fall in either category have been made 'hazy' and are treated as 'noise'.

Some text has been taken and its words have been categorized into two topics namely Topic1→ Software and Topic2→ Hardware. This categorization can be called 'Topic analysis'. If this task was to be done by a human, then he would:

- Read the text and identify certain key words (Leaving out common words/stop words)
- Think of or create a list of 'topics' and then
- 'Assign' each of the key words to one or maybe more of the topics. This process of 'assigning' can also be called **annotation** or **labelling**.

But suppose the same task was to be done by a machine. Then what is available to the machine is only the input text. The machine has to 'find or discover' on its own, the possible topics. Once the topics are identified, the machine may assign each key word to one or more topics.

So 'topic analysis' is a kind of 'tagging'. Note that the key words in the text are known, but the topics are 'hidden' in the sense that they are not available to the machine. Rather the machine has to create the list of topics on its own. So topics can be called 'hidden structure'. The process of 'discovering' the 'hidden or latent' topics falls in the domain of 'probabilistic modelling'. One common algorithm for this is Latent Dirichlet Allocation (LDA). LDA is in turn a development from another model called Latent semantic Analysis (LSA)[2].

Some further points to be noted about 'Topic modelling' are as follows:

- Topic modelling is basically finding a 'bunch of words' in a text document, or corpus which are likely to occur together. For example, there could be a topic called 'Warfare' and one then expect words like 'army', 'tanks', 'warplanes' to 'cluster' around this topic. So instead of having separate 'dimensions' like 'army', 'tanks', 'warplanes', you could have a single dimension 'warfare'.
- So topic modelling is also a form of 'Dimensionality reduction'. Suppose you have a corpus of N terms and you extract T topics out of it (where T < N). Then you have reduced the dimension of the vector space from N to T.

[2]Note that LSA was developed first. LSA did what is called 'Singular Value Decomposition' of the document-term matrix. From LSA came probabilistic latent semantic analysis (pLSI). From pLSI came LDA

- Topic modelling can also be thought of as a type of 'tagging'. For example, when you put a word say 'army' in a topic 'warfare', you have actually tagged the word 'army' with the word 'warfare'.
- Do note that there can be a 'hierarchy' of topics such as: Animal→ Mammal→ Apes→ Gorilla.
- The topics need not be mutually exclusive in the sense that a word with same spelling but different meaning could go to different topics. For example, the word 'bark' could refer to either a 'dog's bark' or the 'bark of a tree'. So if you had two topics say one for pets and other for trees, then the word 'bark' could be present in both.

One can summarize topic modelling as follows:

- Topic modelling is a form of dimension reduction. Suppose a vocabulary has V words. So instead of representing a word in dimension V, you may take T topics where T < V and will represent the word as T.
- Topic modelling can be thought as a form of clustering. When you do topic modelling, you are actually building clusters of words.
- Topic modelling is a form of tagging.

Some common algorithms for topic modelling are:

LDA – Latent Dirichlet Allocation

LSA – Latent Semantic Analysis (LSA is based on Singular Value Decomposition (SVD))

NMF – Non-Negative Matrix Factorization

A typical topic modelling algorithm has 3 matrices. They are:

- A Document Term Matrix (Also called DTM). In this matrix, the rows are the documents and the columns are the terms (or words) in the vocabulary. So if you have say 10 documents and a vocabulary of say 10,000, then this matrix will have dimensions 10 x 10,000. (i.e., 10 rows and 10,000 columns)
- WTM, i.e., Word Topic Matrix. Suppose you chose to have say 15 topics. Then the 10,000 words of the vocabulary will be 'clustered' into 15 groups or clusters. So you have significantly reduced the 'dimension' of the DTM, i.e., reduced the 'columns' of the matrix.
- TDM, i.e., Topic Document matrix. This matrix represents the percentage of each topic represented by each word. In an ideal condition where the clustering is perfect, each document will represent only 1 topic.

Notes/References

In this book of programming, it is not possible to give all mathematical concepts required for 'understanding' and 'using' the various algorithms in vector space models and in libraries like gensim and scikit-learn.

A number of books on linear algebra are available which can be used for better understanding of the maths involved.

However in keeping with the open source tradition of Python programming language, you can follow a free/open source book, i.e., 'Interactive Linear Algebra'[3] by professors of Georgia Institute of Technology.

[3]See: https://textbooks.math.gatech.edu/ila/ila.pdf