PES University, Bangalore

(Established under Karnataka Act No. 16 of 2013)

MAY 2020: IN SEMESTER ASSESSMENT (ISA) B.TECH. IV SEMESTER

UE18MA251- LINEAR ALGEBRA

MINI PROJECT REPORT

ON

Submitted by

1. Name L.Anvesh Reddy SRN PES1201801299
2. Name T.Sai Rahul SRN PES1201801232
3. Name Swarup Banik SRN PES1201801050

Branch &Section:CSE,C-section

PROJECT EVALUATION

(For Official Use Only)

Sl.No.	Parameter	Max Marks	Marks Awarded
1	Background & Framing of the problem	4	
2	Approach and Solution	4	
3	References	4	
4	Clarity of the concepts & Creativity	4	
5	Choice of examples and understanding of the topic	4	
6	Presentation of the work	5	
	Total	25	

Name of the Course Instructor : P RAMA DEVI

Signature of the Course Instructor :

Using Word2Vec model to Create Word Embeddings and Visualizing it using PCA

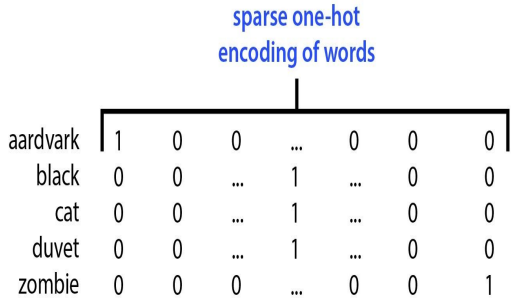
1 Introduction

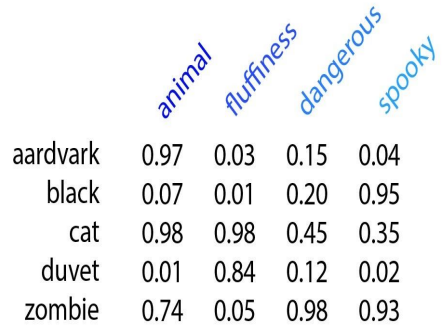
Classic sequence model's like Rnn,lstm which take a single word as an input in each time step and try to predict the next word or try to take a group of words(a sentence) as an input in each time step and predict the next word in the sentence.But is the network trying to learn what each words mean.

For example consider the job of rnn is to try to predict the next word in these sentences
1)He likes Orange juice
2)He likes Apple____
Every Human who saw the first example can interpret the second example as Apple juice,why can't an rnn do it,because one word in each sentence is not dependent on other words.This is where Embeddings come into the picture.Instead of using a one hot sparse vector as an input ,it creates a n-dimensional spaces for each input.

Say we create the embeddings for above which has inputs of different fruits,different vegetables etc,since Orange and apple are used in similar kind of context,they both are placed together, Even though in training set there,if there is no mentioning of apple juice,since orange and apple are similar kind of fruits and has embeddings close to each other,our model outputs it as,He likes Apple juice.

If we take 5 example words from our vocabulary (say… the words "aardvark", "black", "cat", "duvet" and "zombie") and examine their embedding vectors created by the one-hot encoding method.





Core problem that embeddings solve is generalisation

Above Images tell about different word representations,the left one is create using one-hot encoding of the vectors,the right one is a 4 dimensional embedding created using one-hot-encoding method,it learns four features to differentiate the words given,Word2vec model is used to create those embeddings.

Each words one hot vectors are mostly zeros,and many machine learning models won't work well with very high dimensional and sparse features,Neural networks struggle with this kind of data.With such a large vector,there is often a trouble of running into storage concerns and memory.

2 Word2vec

Word2vec is a group of related models that are used to produce the word embeddings.these models are shallow neural networks,that are trained to reconstruct linguistic context of words.word2vec takes input a large corpus of text and produces a vector space,typically of several hundred dimensions,with each unique word in corpus being assigned a individual vector space.word vectors are positioned in vector space such that words that share similar features are close to each other.

Word2vec was created and published in 2013 by a team of researchers led by Tomos mikolov at Google and patented. The algorithm has been subsequently analysed and explained by other researchers. Embedding vectors created using the Word2vec algorithm have some advantages compared to earlier algorithms such as latent semantic analysis.

Word2vec can utilize either of two model architectures to produce a representation of words: continuous bag of words (CBOW) or skip gram. In the continuous bag-of-words architecture, the model predicts the current word from a window of surrounding context words. The order of context words does not influence prediction (bag of words assumption). In the continuous skip-gram architecture, the model uses the current word to predict the surrounding window of context words. The skip-gram architecture weighs nearby context words more heavily than more distant context words.it is said that,CBOW is faster while skip-gram is slower but does a better job for infrequent words.

2.1 CBOW(continuous bag of words)

As mentioned,in cbow architecture the model predicts the word from a window of surrounding context words.

For example,

I want a glass of orange juice to go along with my cereal.

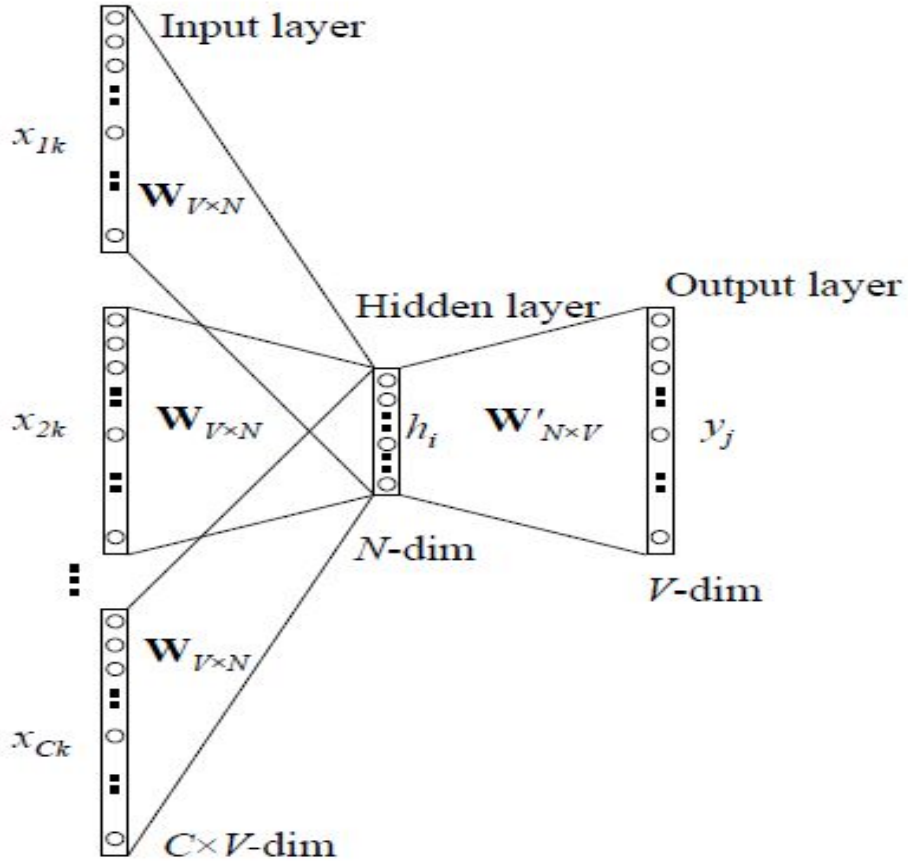
In the above examples we randomly pick a single target word and pick context words around the target word,with a window size,we take the window size as 2 for this example.

1)Target:juice Context:[orange,to]

2)Target:cereal Context:[my]

3)Target:glass Context:[a,of[

These context words are converted to one-hot-encoded vectors,and these are the input to a shallow neural network with a single hidden layer,which tries to predict the target word.



The input is a one-hot encoded vector consider we have corpus of 10,000 unique words,since there are three inputs we will compute hidden layer weight embedding matrix as

$$\begin{aligned}\mathbf{h} &= \frac{1}{C}\mathbf{W}^T(\mathbf{x}_1+\mathbf{x}_2+\cdots+\mathbf{x}_C)\\ &= \frac{1}{C}(\mathbf{v}_{w_1}+\mathbf{v}_{w_2}+\cdots+\mathbf{v}_{w_C})^T\end{aligned}$$

Input will have a dimension of 10,000xn(for n inputs)and hidden layer weights will have a dimension of 300x10,000,we chose 300 dimensional embedding.the output is a matrix of dimensions 300xn.So for each input a 300 dimensional embedding is created.we will update them using backpropagation.

From hidden to output layer,there is a different weight matrix for the output layer (Embedding),which is also of the shape 300xn,and is computed by the formula

$$\mathbf{h} = \mathbf{W}^T\mathbf{x} = \mathbf{W}^T_{(k,\cdot)} := \mathbf{v}^T_{w_I},$$

$$u_j = {\mathbf{v}'_{w_j}}^T\mathbf{h},$$

Where v'wj is j-th column of the matrix W'.Then we can use softmax,a log-linear classification model,to obtain posterior distribution of words,which is a multinomial distribution.

$$p(w_j|w_I) = y_j = \frac{\exp(u_j)}{\sum_{j'=1}^{V}\exp(u_{j'})},$$

Where yj is the output of the j-th unit in the output layer

$$p(w_j|w_I) = \frac{\exp\left({\mathbf{v}'_{w_j}}^T\mathbf{v}_{w_I}\right)}{\sum_{j'=1}^{V}\exp\left({\mathbf{v}'_{w_{j'}}}^T\mathbf{v}_{w_I}\right)}$$

Substituting all the equations we get the above equation.we call Vw as input vector and Vw' as the output vector of the word w.

Our goal is to maximize the probability p(wj/wi),For this we use a softmax loss function to find the loss b/w predicted and actual output.

$$\begin{aligned}\max p(w_O|w_I) &= \max y_{j^*} \\ &= \max \log y_{j^*} \\ &= u_{j^*} - \log \sum_{j'=1}^{V} \exp(u_{j'}) := -E,\end{aligned}$$

We try to minimize the loss function E.How can we minimize it?,we use gradient descent,we take the derivative of E with respect to uj(embedding(weight) matrix of the output layer).

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

Where tj=1 i.e tj will only be 1 when jth unit is the actual output word,otherwise tj=0.The derivative is simply the prediction error ej of the output layer.

Now we will take the derivative of w'ij to obtain gradient on the hidden->output weight

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

Therefore,using stochastic gradient descent,we obtain the weight updating equations for hidden->output weights

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

After,there is one more layer of weights to update, there is input->hidden layer weights(the weights for the first hidden matrix),for this we take derivative of loss on output of the hidden layer,obtaining

$$\frac{\partial E}{\partial h_i} = \sum_{j=1}^{V} \frac{\partial E}{\partial u_j} \cdot \frac{\partial u_j}{\partial h_i} = \sum_{j=1}^{V} e_j \cdot w'_{ij} := \mathrm{EH}_i$$

Since E with respect to h,we can get the derivative of E w.r.t wki,obtaining

$$\frac{\partial E}{\partial w_{ki}} = \frac{\partial E}{\partial h_i} \cdot \frac{\partial h_i}{\partial w_{ki}} = \mathrm{EH}_i \cdot x_k$$

This is equivalent to tensor product x and EH i.e.,

$$\frac{\partial E}{\partial \mathbf{W}} = \mathbf{x} \otimes \mathrm{EH} = \mathbf{x}\mathrm{EH}^T$$

By this we can obtain the weights of the first hidden matrix,using Stochastic gradient descent.

$$\mathbf{v}_{w_I}^{(\mathrm{new})} = \mathbf{v}_{w_I}^{(\mathrm{old})} - \eta \mathrm{EH}^T$$

As we iteratively update the model parameters by going through different context-target words,generated from a training corpus,all the Embeddings(Weights) of hidden layer and output layer are updated,our main goal is not to predict the context and target words,it is to update Embedding matrix correctly,all similar kind of context words(for eg:orange,mango) will have similar kind of target words surrounding them it means that,all the fruits(mango,orange) are used in similar kind of situations,so their Embedding matrices are updated in a similar way(need not be exactly similar),so all the similar contexts stay close to each other,that’s how the embedding matrices are updated and used.

2.2 Skip Gram model

Skip gram architecture is similar to the Cbow model except,the input and output patterns change.it is an opposite to the Cbow model.

Before we used to have multiple context words and a single target,now we have a single context word and multiple targets selected over a window of variable size.

For example,
Eg:The quick brown fox jumps over the lazy dog.

1)Target:[brown,jump] Context:fox

2)Target:[the,dog] Context:lazy

3)Target:[the,brown] Context:quick

As we can see in the above example,context word is randomly picked and we pick two words surrounding it and label them as target variables.

The above context word is converted to one-hot-encoded vector,and that is the input to a shallow neural network with a single hidden layer,which tries to predict the target words.

In the input layer of the Cbow model,we input n one-hot encoded vectors and average them,here there is only a single input vector,so we just copy it,whereas in the output layer instead of outputting one multinomial distribution we are outputting C multinomial distributions.Each output is computed using the same hidden->output matrix.

$$p(w_{c,j} = w_{O,c}|w_I) = y_{c,j} = \frac{\exp(u_{c,j})}{\sum_{j'=1}^{V} \exp(u_{j'})}$$

Wc,j is j-th word of the c-th panel of the output layer and Wo,c is the actual c-th word in the output word.Wi is the only input word;Ycj is the output of jth unit on the cth panel of the output layer.uc,j is the net input of the j-th unit on the c-th panel of the output layer.because the output panel shares the same weights.

$$\mathbf{h} = \mathbf{W}_{(k,\cdot)}^T := \mathbf{v}_{w_I}^T,$$

$$u_{c,j} = u_j = \mathbf{v}'_{w_j}{}^T \cdot \mathbf{h}, \text{ for } c = 1, 2, \cdots, C$$

Where V'wj is the output of the jth word in the vocabulary,wj.

The derivation of the parameter update equations(gradient descent) is not so different from the cbow model,the loss function changed to,

$$\begin{aligned} E &= -\log p(w_{O,1}, w_{O,2}, \cdots, w_{O,C}|w_I) \\ &= -\log \prod_{c=1}^{C} \frac{\exp(u_{c,j_c^*})}{\sum_{j'=1}^{V} \exp(u_{j'})} \\ &= -\sum_{c=1}^{C} u_{j_c^*} + C \cdot \log \sum_{j'=1}^{V} \exp(u_{j'}) \end{aligned}$$

jc* is the index of the actual c-th output context word in the vocabulary..

We take the derivative of E with regard to net input of every unit on every panel of the output layer,Uc,j and obtain

$$\frac{\partial E}{\partial u_{c,j}} = y_{c,j} - t_{c,j} := e_{c,j}$$

Next we take the derivative of E with regard to hidden->output matrix W',and obtain

$$\frac{\partial E}{\partial w'_{ij}} = \sum_{c=1}^{C} \frac{\partial E}{\partial u_{c,j}} \cdot \frac{\partial u_{c,j}}{\partial w'_{ij}} = \mathrm{EI}_j \cdot h_i$$

Thus we obtain the update equation for hidden->output matrix W',

$$w_{ij}'^{(\text{new})} = w_{ij}'^{(\text{old})} - \eta \cdot \text{EI}_j \cdot h_i$$

The above update equation is already explained in Cbow,After this we obtain hidden layer weight matrices(Vwi).

$$\mathbf{v}_{w_I}^{(\text{new})} = \mathbf{v}_{w_I}^{(\text{old})} - \eta \cdot \text{EH}^T$$

So after we update the embedding matrices for various pairs of context and target words,we can create an embedding matrix for each word,where as explained in Cbow,similar kind of context->target words are updated similarly and close to each other.

3 Visualization of Embeddings Using PCA

Now the embeddings are created,how do we visualize it?.We can't visualize it because many of the embeddings created have millions of unique words on it,in general case we create a 300 dimensional embeddings,visualizing 300 dimensional embeddings is impossible,so we project those high dimensional embeddings into lower dimensional embeddings without much loss of information,there are many dimensionality reduction techniques used such as t-Sne,pca etc.For this paper we chose PCA(principal component analysis) for dimensionality reduction.

What is PCA?,pca is an unsupervised linear transformation technique,which helps us to identify patterns in data based on correlation b/w features.in a nutshell,PCA aims to find the directions of maximum variance in high dimensional data and projects into a new subspace with equal or few dimensions than the original one.

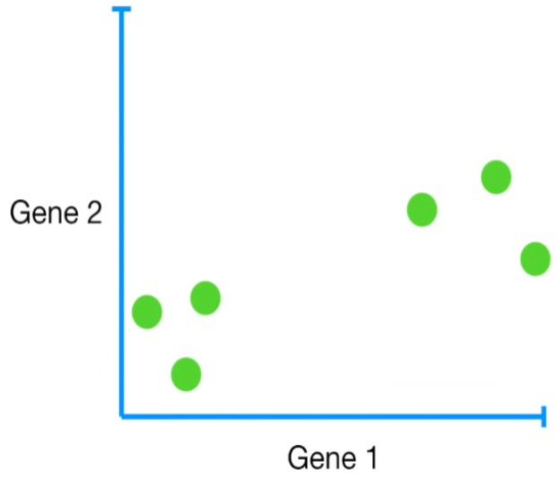
The principal components have now got nothing to do with original features,we will get 300 principal components with 300 features.The newly transformed feature set of principal components will have maximum variance explained in the first pc,the second pc will have the second highest variance and so on.

For example,if the first pc explains 70% of the whole data,the 2nd features explains 13% of the total data and the next 3 features explain around 12% of the whole data.so u have around 95% of the variance explained by just 5 principal components,lets say the next features explain 3% of the whole data.it makes less sense,now to include more than 5 dimensions here,by this we reduce the dimensionality from 300 to a mere 5.

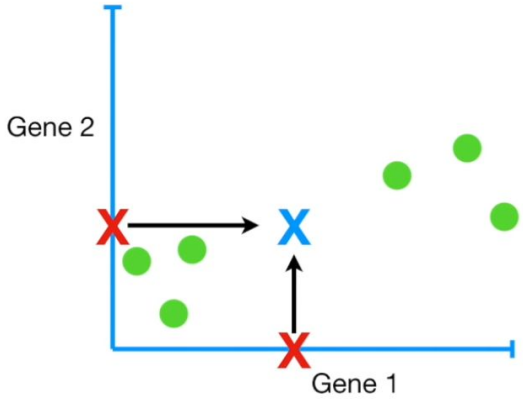
Let's consider a small example of a data of 2 dimensions(so it will be easy to visualize),and we will try to find principal components for that data.consider a data that has two genes Gene 1 and Gene 2.

	Mouse 1	Mouse 2	Mouse 3	Mouse 4	Mouse 5	Mouse 6
Gene 1	10	11	8	3	2	1
Gene 2	6	4	5	3	2.8	1

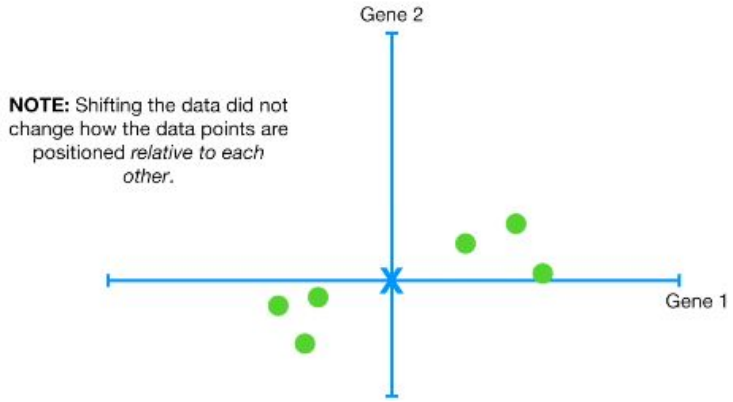
Below is the plotted representation,for the above data spread across Gene1 & 2



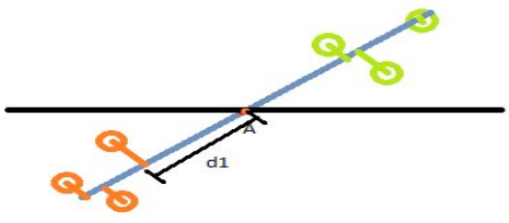
Then we will calculate the average measurement of Gene1 and the average measurement of Gene2,with those average values we can calculate the center of the data,let's call it c.



What we have to do now is take the point c,and shift it towards origin along with the other points in the data,we do that so it makes it easy to visualize the data.



After this we start by drawing a random line that goes through origin,we project all the points on to the created line and we try to find the best fitting line that maximizes the distance from the projected point to the origin,the distance d1 below is the distance of point 1 w.r.t origin,like this we calculate distance for all the 6 points and label them d2,d3,d4,d5,d6 respectively.

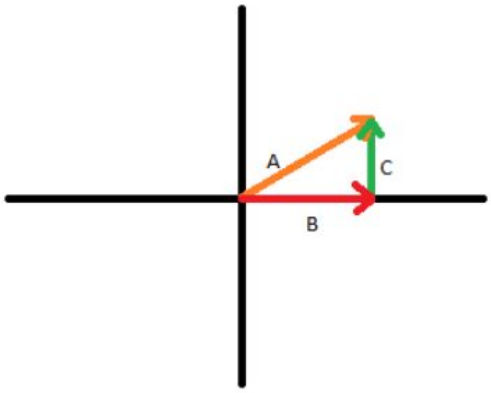


We will now calculate the sum of squared distances from d1-d6

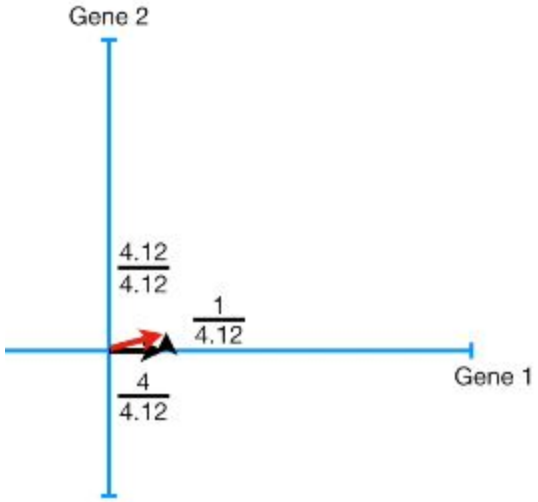
$$d_1^2 + d_2^2 + d_3^2 + d_4^2 + d_5^2 + d_6^2 = \text{sum of squared distances} = SS(\text{distances})$$

Like this we do for different random lines until we end up with the line with the largest sum of squared distances b/w projected points and the origin,that line is denoted as pc1(first principal component).

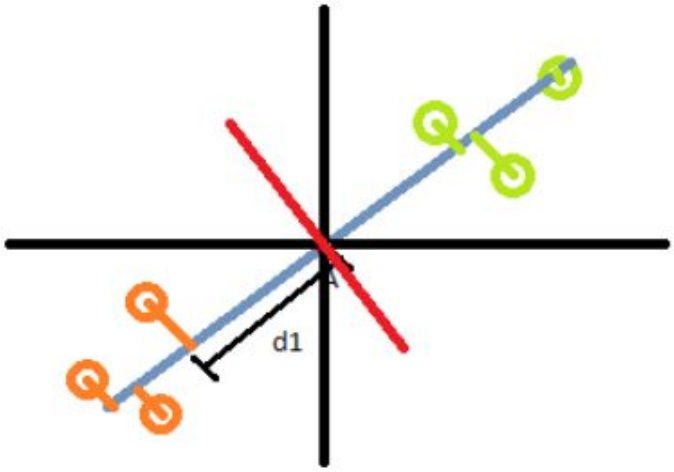
Suppose the slope of that line is 0.25,that means that line consists of 4 parts of gene1 and 1 part of gene2,it means in pc1,the data are mostly spread along the gene1 axis,so we mathematically call pc1 as a linear combination of 4 parts of gene1 and 1 part of gene2.



When B=4 and C=1,we can calculate A using pythagorean theorem,when you do pca with svd it scales the value A to be 1.Hence A=1,B=4/4.12=0.97,C=1/4.12=0.242.This unit vector A is eigenvector,sum of the squared distances(SS) is the eigenvalue.

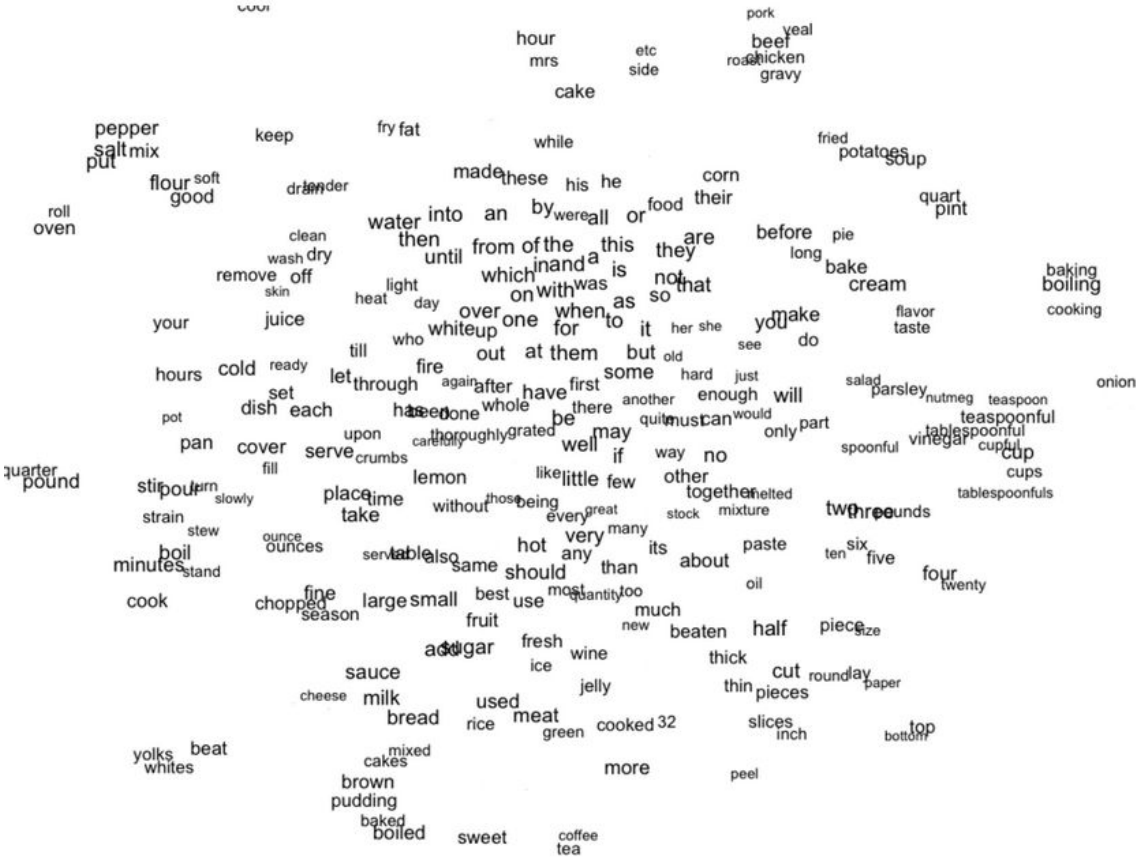


Now for the second principal component will be the vector orthogonal to the pc1 as principal components have 0 correlation b/w them,it is shown by the redline below.



As we know in pc1,pc2 consists of -1 parts of gene1 and 4 parts of gene 2,so for pc2 data are mostly spread out along the gene 2 axis,if we divide the sum of square distance for each of pc1 and pc2 by n-1 we will get variance for the respective principal component,let's suppose sum of square distance for pc1 is 13 and for pc2 is 5.so pc1 accounts for 13/18=72.22 percent of variance and pc2 accounts for 27.7 percent of the total variance,this tells that how much variance is explained by a particular component,principal components are ranked according to the variance percentage,so we select the top m components,by this we can say that pca is a feature extraction technique,it extracts the features with maximum variance.

This is how pca reduces the dimensions,lets do these for word embeddings,below picture is word embeddings for many different classes,created using word2vec and reduced the dimensionality from 300 to 2 using pca.



As we can see in the above image,as discussed the letters used in the same context,will have similar updates to their embedding vectors,so they stay near,in the above image(in the top-right side) we can see that baking,boiling,cooking are used in similar context so they stay close,and the pairs of (baked,boiled) and (coffee,tea) are close to each other.

By this we can say that the words which are used in similar contexts will represent each other,it means that even when a particular sentence doesn’t appear in the training data,

For example,
1)I need to cook a dish(appears in the training set)
Suppose we want to predict the next word in the following sentence
2)I need to bake___.
Since cook and bake have similar kinds of representation in the word embeddings,so if we using embeddings we can predict the next sentence as a dish,since bake and cook mean the same thing,this was not possible with classic rnn with one hot encoded input before,since one sentence is completely independent of other sentences in rnn.