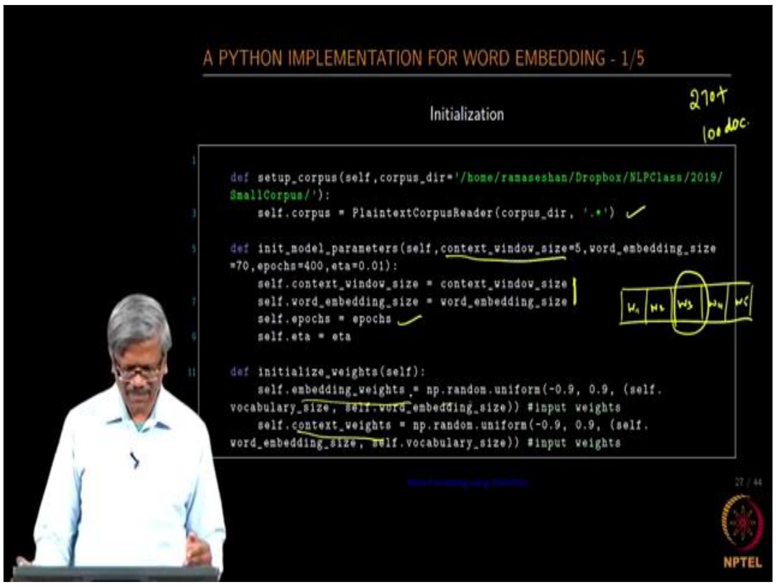
Applied Natural Language Processing

Prof. Ramaseshan Ramachandran
Department of Computer Science and Engineering
Chennai Mathematical Institute

Lecture – 45
Building Skip-gram model using Python

(Refer Slide Time: 00:15)



A PYTHON IMPLEMENTATION FOR WORD EMBEDDING - 1/5

Initialization

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

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

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

27 / 44

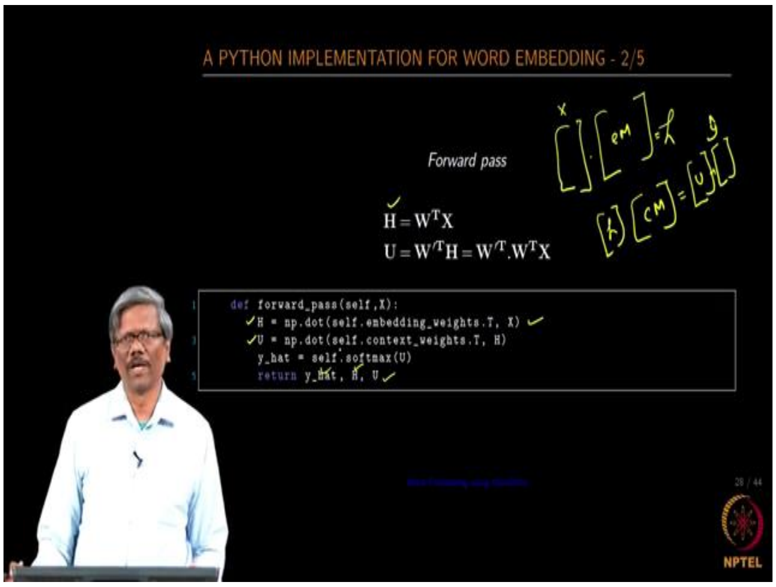
So, I am going to be taking this Skip-gram Model as an example and I am going to be showing some small pieces of code. It is not complete code and I am going to be providing only the pieces of code for you as well your job is to really take this code, integrate and make the application run. So, I will not be giving all the details, every line of code, and so on, I will be giving only pieces of code for you this time not like last time where I gave you the full code ok.

So, for any network to work so, you have to do some kind of initialization. We need to first set up our corpus. I am taking a very small corpus of about a hundred documents. It is very small it is in good enough to really identify all the word vectors in the right context this is just to show how it works ok. So, I will be initializing some parameters initial initially I set up the corpus, so that I can read all the documents and then I initialize the model parameters. In this case I am taking the context window I remember we will be having the context window where we will be looking at in this skip-gram model I will be inputting this and these are the context word the application is supposed to predict.

So, I have the number of epochs and then the context window size and these are the initialization that you can provide when you start the program, and then you initialize the weights. As I mentioned earlier the initially to start with you have to have some weights associated with that. So, I am just using a random number using a uniform distribution between the values of minus 0.9 to 0.9. I initialize both embedding rates as well as context weights here using this ok. So, this is the second step that you have to do.

So, you initialize all the parameters and then initialize your weights, and then we start the forward process.

(Refer Slide Time: 02:47)



A PYTHON IMPLEMENTATION FOR WORD EMBEDDING - 2/5

Forward pass

$$H = W^T X$$
$$U = W'^T H = W'^T . W^T X$$

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

28 / 44

So, what is the forward process? We know that now you need to identify the H. So, we have our X here and then we have our embedding matrix and then we need to find the hidden values ok.

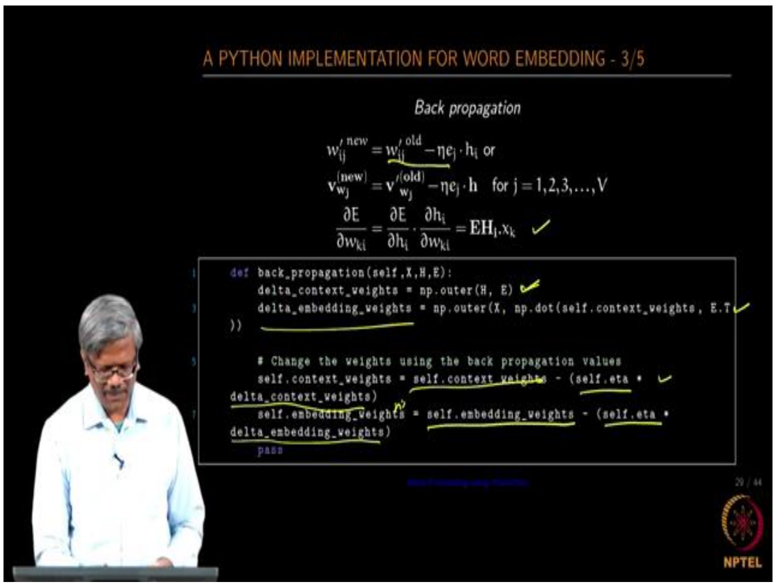
$$H = w^T X$$

$$u = w^T H = w^T . w^T$$

So, in the forward pass was what we do is we just you take the transpose of the weight matrix and then take a dot product, took it H and then what is the U? U use you the final value that is calculated using the hidden values and the context matrix, correct? This is one step before softmax all right. So, in one go I compute the forward pass ok. So, you know what is the input that you are providing and then once it is done H is calculated and now u can be calculated using H as well; initially these weights as I mentioned earlier are random weights.

So, now, you compute y hat; y hat is your softmax values ok. So, using softmax you find they hat. So, we have H here and then you have the context matrix, and now we will give you U and then this results in y hat and then I return all the three values in one go ok. So, we have computed H, U and y hat in the forward pass, how easy it is rights.

(Refer Slide Time: 04:44)



A PYTHON IMPLEMENTATION FOR WORD EMBEDDING - 3/5

Back propagation

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

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

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

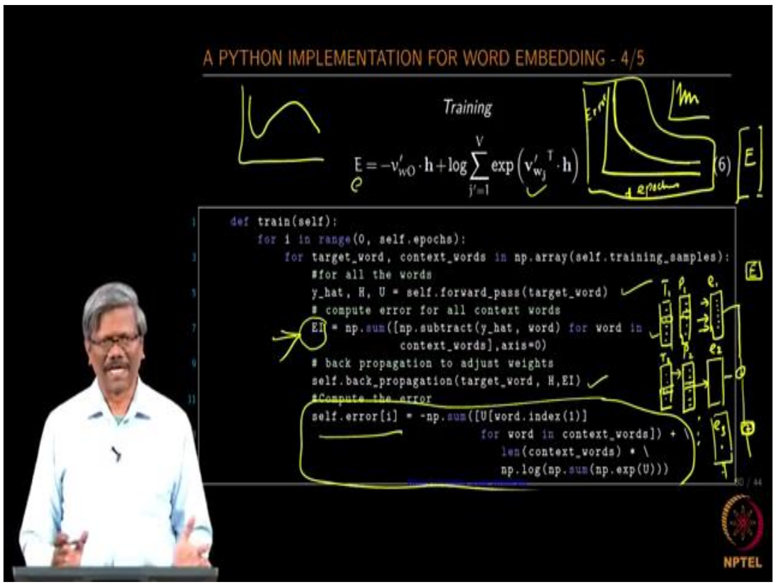
```
def back_propagation(self,X,H,E):
    delta_context_weights = np.outer(H, E)
    delta_embedding_weights = np.outer(X, np.dot(self.context_weights, E.T
))

    # 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
```

So, then now you have to do the backpropagation. So, U is computed softmax is computed so, find the error. Now, we have the equations for your convenience I have provided these equations. So, we need to update the weights in the context matrix using this form right. So, for that you need to find the difference right. So, what is the difference that you want to compute and then later use the difference to find the new values? So, this is what we are doing here.

So, we do it for the context matrix and then do it for the embedding matter, and then update the context weights. So, the new weights are updated using the old context term weights and then use a learning parameter, and then use the delta context weights which we computed earlier here, and get the new context weight. So, once the context weights are obtained we now go back to the embedding weights and then update it as well.

Again, using the formula we are going to be updating the embedding weights. So, we have the old embedding weights here, and then use this same inner parameter, and then use the delta embedding weights that we computed earlier to find the new embedding weights here. So, this is your new value right. So, in about 5 lines we have done the backpropagation part right.

(Refer Slide Time: 06:42)



A PYTHON IMPLEMENTATION FOR WORD EMBEDDING - 4/5

Training

$$\underset{e}{E} = -v'_{wO} \cdot \mathrm{h} + \log \sum_{j'=1}^{V} \exp\left(\mathrm{v}'_{\mathrm{w_j}}{}^{T} \cdot \mathrm{h}\right) \quad (6)$$

```
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)))
```

And, then how do you keep doing this right. So, we need to have the error parameter that needs to be minimized as given here and that will be incorporated as part of the training process and then every time when those weights are computed when the errors are computed, we want to find out whether the value of e minimizes ok. So, when it keeps going down and down so, usually you get something of this type ok. This is your error and this is the number of epochs.

So, we need to keep looking at this. You know initially, it will be random you keep a tab on this right when you start doing the training part and then see how it goes sometimes if you have not done your implementations right it will be very erratic in this fashion or if the learning parameter is not appropriately chosen it will also not be in this fashion. So, sometimes it will calm down and then go up and then try to come down like this ok. So, make sure that you get something ideally in this fashion ok. So, this is something that you want to get at the end ok.

So, here what we do is we just do the forward pass as I had shown earlier and then we keep computing the EI part for every word and then compute the backpropagation ok. So, this particulars instruction what it does is, it actually computes the error for every context word. You remember in the first slide when I was talking about we have the target right and then this is the predicted value and then we found the error all right for a bigram where we have only one target word and the context word.

So, now, we have more than one context word so, that means, we will have more error values correct. So, let us say this is 1 this is for the first context word, target 2 predicted value 2, so, error 2. So, when you compute the error you do it for all elements or all the words ok. So, the size would be the same as T_1, T_2 and so on. So, in the same fashion for the second word you compute for example, initially we this is our target and this is our prediction even though we are only finding the difference for this, but we also find the difference for all the elements right and then they go here.

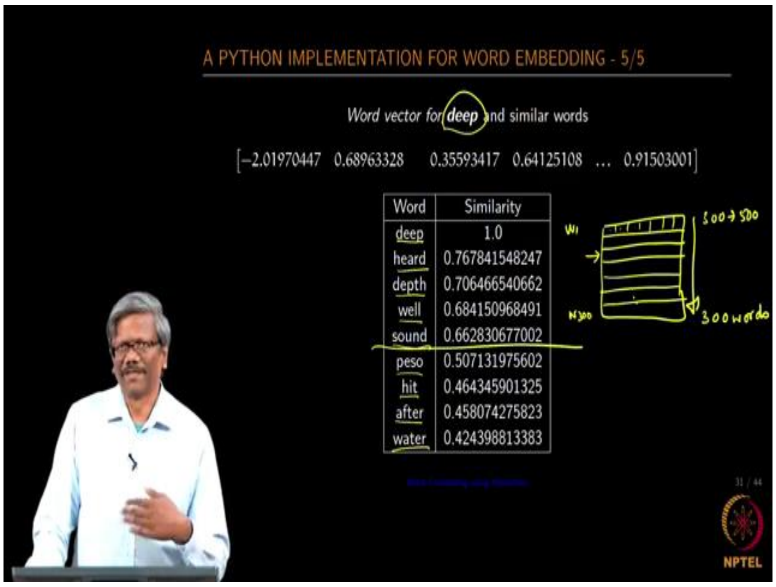
So, in the same fashion for this word we calculate error 2 and so on. So, in this fashion if you have got 5 or 4 we need to compute the sum of all this. So, in this case what we do is we and then finally, we get one E from all this right. So, the E size also is the same as the size of our vocabulary. So, all differences are summed at the end and this is what is happening in this step ok. So, once it is done use the backpropagation to update all the weights.

So, you remember in the backpropagation we are updating both the embedding weights as well as the context weights, I am sorry in the order of context weight and embedding weights and then compute the error for that particular epoch so, using this formula. So, we have given here and this is the error that you have to keep looking at. If this error goes down in this fashion smoothly you are doing good ok. So, if you have something in this erratic fashion something is wrong.

So, do not start with a thousand epochs immediately, do not start with a one million vocabulary immediately; start small. Take about 10 words as vocabulary or 5 epochs and then start looking at this ok. So, make sure that your program is right, make sure that the error is slowly coming down then you can really take a bigger corpus, bigger vocabulary, and then increase the number of epochs and so on. So, while doing that keep playing with the learning parameter to see what is the right learning parameter for you as well ok. So, you got it?

So, so when you look at it through the program you know sometimes it looks so simple. So, that is why I thought I will also take this route rather than just explaining the equations, rather than just giving them in the matrix form I thought I will also give the program so that you can follow that ok.

(Refer Slide Time: 12:49)



A PYTHON IMPLEMENTATION FOR WORD EMBEDDING - 5/5

Word vector for **deep** and similar words

$[-2.01970447 \quad 0.68963328 \quad 0.35593417 \quad 0.64125108 \quad \ldots \quad 0.91503001]$

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

So, after completing the exercise of we now have embedding vector ok. So, each row represents a word vector for you supposing you have about 300 words ok; this is a word 1 to 300. So, each vector will have let us say 300 to 500 elements depending on what you have chosen as the hidden value size.

So, what I have done in this case is I have trained that very small corpus which contains a problem statement related to kinematics and then I trained the vector and each problem is about 1 or 2 lines consisting of about 20 to 30 words max I have taken one word and then I am just trying to find out what are these similar words in this.

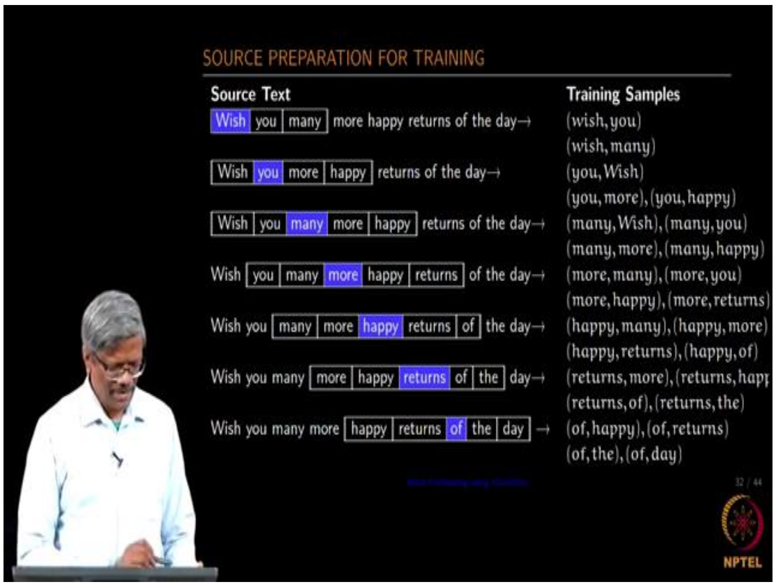
So, the kinematics problem there are problems where you are dropping a stone into a well and then the sound was heard after a few seconds how deep is well ok. So, like that there are several problems of that type. So, I just wanted to find out how this word deep is related to the words that are surrounding it or I just wanted to find out the words that really define deep in this given context ok. So, once the program reaches an equilibrium state we do not need the context weight part, we do not need that softmax, we do not need any of those, what we require is the embedding matrix ok.

So, now I take one word deep as a word, and then I take the vector out of that and then run it through all the vectors, word vectors here to find the similarities. So, I use a cosine similarity, I am sure you remember that and then find out the values. And, then listed the top 10 or 15 of them to find out how close those words are in the given embedded matrix ok.

So, you will see that heard, death, well, sound, peso that is a coin used and then hit, after how many seconds are you hurt, water. So, all of those words are related to deep in this case ok. So, it varies from 0.77, rather 0.8 to 0.4 ok. So, if you want still closer once you know you do not want beyond 5 or 4 this is how you got it. So, in this case what I have done is I ran through the word vectors through the entire set of words or the vectors that were found through the model and then found the similarity and then listed them in the descending order ok.

So, you get this right now this is the most crucial part in the natural language processing, where you need to really create word vectors for the given context and the corpus ok. There could be a general-purpose of word vectors that you can also create by looking at the content available on the internet by taking close to 2 billion or 3 billion words. Say for example, extract close to 3 billion words from Wikipedia and then start training using this skip-gram model and so on. So, as I mentioned earlier the implementation that I have shown uses a skip-gram model all right.

(Refer Slide Time: 17:35)



SOURCE PREPARATION FOR TRAINING

Source Text	Training Samples
Wish | you | many | more happy returns of the day→	(wish,you) (wish,many)
Wish | you | more | happy | returns of the day→	(you,Wish) (you,more),(you,happy)
Wish | you | many | more | happy | returns of the day→	(many,Wish),(many,you) (many,more),(many,happy)
Wish | you | many | more | happy | returns | of the day→	(more,many),(more,you) (more,happy),(more,returns)
Wish you | many | more | happy | returns | of | the day→	(happy,many),(happy,more) (happy,returns),(happy,of)
Wish you many | more | happy | returns | of | the | day→	(returns,more),(returns,hap (returns,of),(returns,the)
Wish you many more | happy | returns | of | the | day | →	(of,happy),(of,returns) (of,the),(of,day)

32 / 44

NPTEL

So, so far what we have seen is how this model works right. We also have mentioned earlier that the size of the network really matters, correct. If you start looking at the larger corpus where you have more than a million-word as your vocabulary I am going to be having 1 million 1 hat vectors right and then every 1 hat vector will have 1 million elements and then correspondingly you have a 1 million into let us say if you restrict the hidden layer size to 300, 1 million into 300.

And then if you go to the context side again you have the metric of matrix operation that you have to perform for the matrix of size 300 into 1 million and then we have U that is 1 million elements, we have a softmax that is 1 million elements and then the difference that will be 1 million elements and so on right. So, look at the size as the number of words in the vocabulary grows.