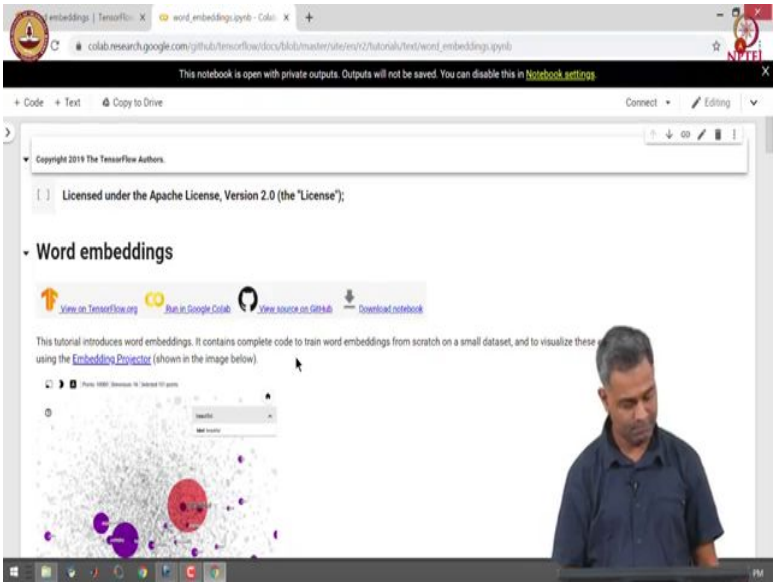
Practical Machine Learning
Dr. Ashish Tendulkar
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

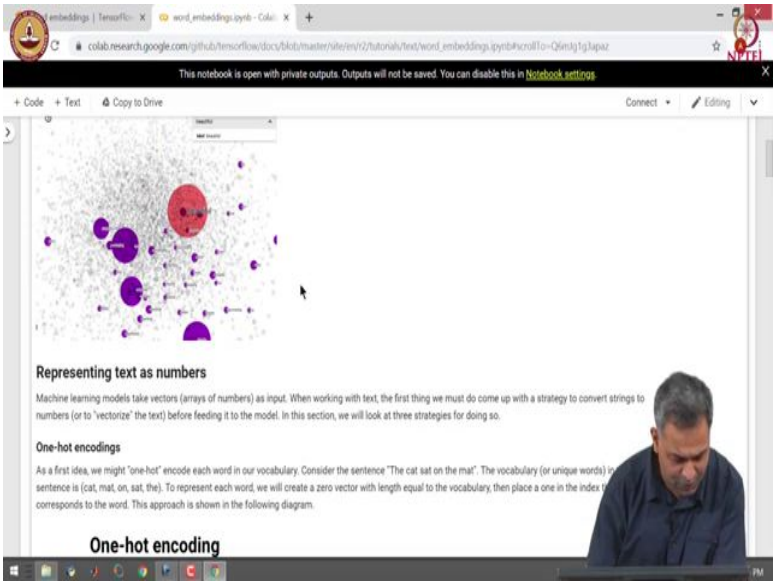
Lecture – 30
Introduction to word embedding

In this session, we will study word embeddings. We will train these embeddings from scratch and use them for training the model. You can also visualize these embeddings with embedding projector.

(Refer Slide Time: 00:33)



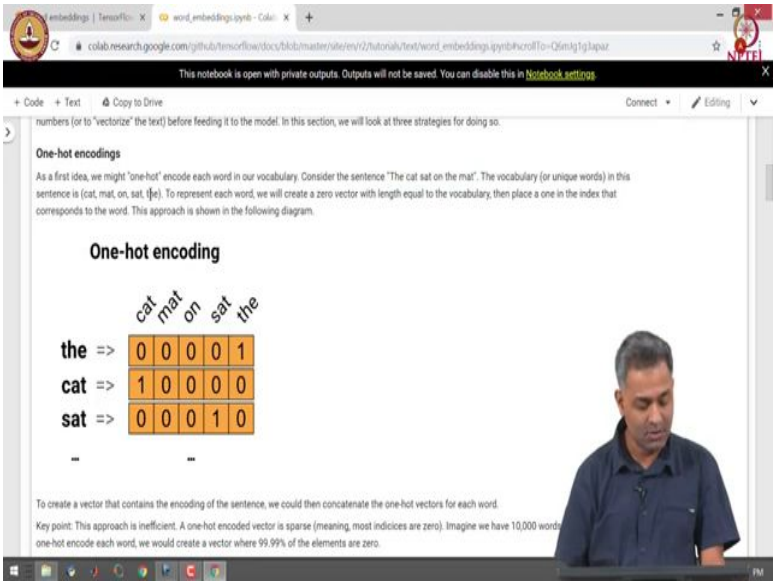
(Refer Slide Time: 00:35)



This is an example of an embedded projector where you are able to visualize the embeddings, where we can see different words embedded in the embedding space.

As we discussed earlier, we cannot use text content as it is and we need to convert the text into numbers. There are different schemes that are available for converting text into numbers.

(Refer Slide Time: 01:12)

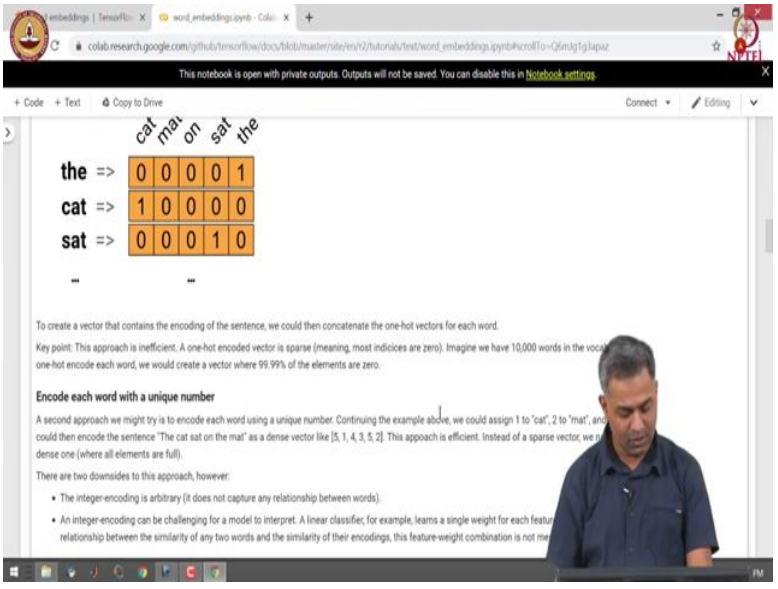


So, one of the schemes is one-hot encoding where what we do is we take the sentences and extract vocabulary from these sentences by tokenizing them and selecting unique tokens.

For example, for the sentence the cat sat on the mat, the vocabulary or unique words are cat, mat, on, sat and the. Notice that, the is included only once in the vocabulary list, though it is repeated in the sentence. To represent each word what we do is we create a zero vector of length equal to the vocabulary. So, here in our vocabulary there are five words. So, we create zero vectors for each of the tokens and then you place 1 in the index that corresponds to words.

So, for example, for the word "the", we have a zero vector and then we convert the 0 corresponding to the word "the" and replace that with 1.

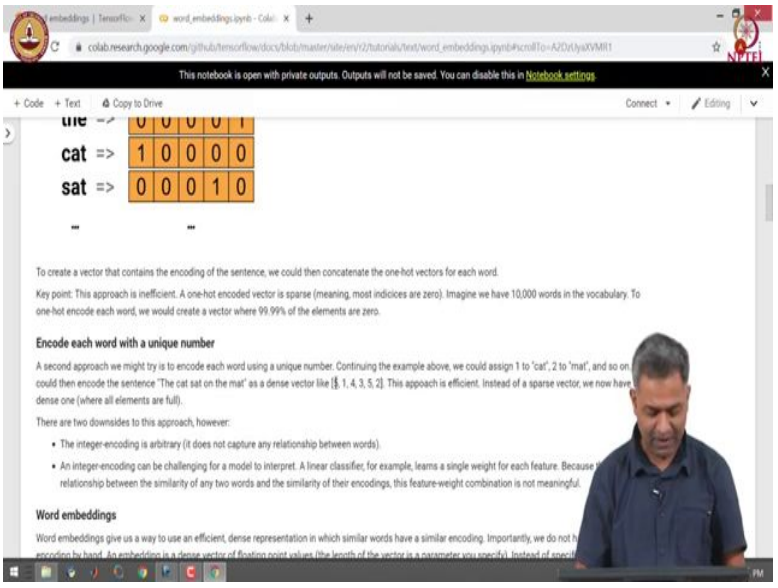
(Refer Slide Time: 02:51)



To create a vector that contains encoding of a sentence what we do is we concatenate this one-hot vectors for each word. But, this is terribly inefficient because a large percentage of positions will have zero. Imagine a 10,000 word vocabulary and if you use one-hot encoding for each word a very large percentage of positions will be 0.

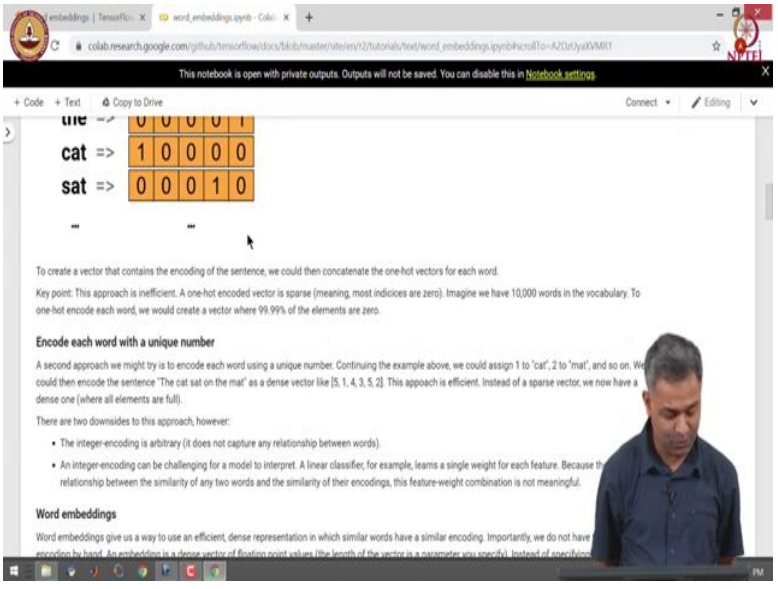
The second approach of converting these words into numbers can be achieved through encoding of word with a unique number. So, what we do is as soon as we get the vocabulary we map each word to unique integer.

(Refer Slide Time: 03:50)



So, for example, for the word the cat sat on the mat, we can have a representation which is some numbers.

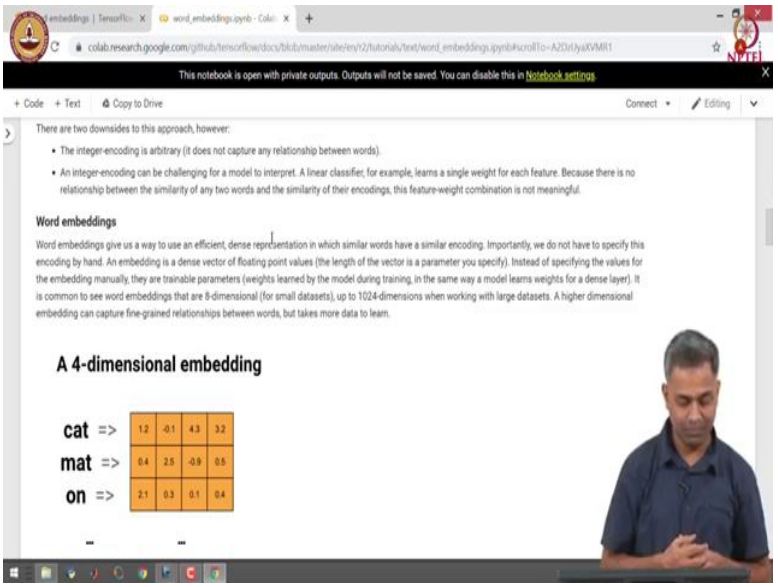
(Refer Slide Time: 04:06)



So, here for the word “the” we have mapped it to number 5. So, we see 5 at positions corresponding to “the”. So, this is an efficient approach where we have a dense tensor rather than a sparse tensor as we experienced that in one-hot encoding.

However, there are two downsides to this approach. The integer encoding is arbitrary, does not capture any relationship between words. And, an integer encoding can be challenging for the model to interpret. A linear classifier for example, learns a single weight for each feature because there is no relationship between similarity of any two words and the similarity of their encoding the feature-weight combination is not meaningful.

(Refer Slide Time: 05:30)

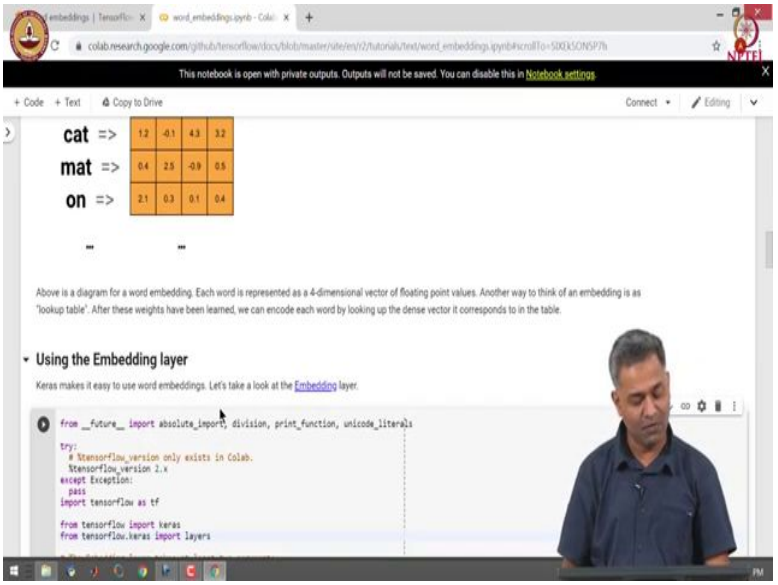


So in order to overcome the limitations of first two approaches, we have an approach of word embedding. Embedding gives us an efficient and dense representation, where similar words have similar encodings. Importantly we do not have to specify the encoding by hand, and these encodings can be learnt during the training process.

What is embedding really, what does embedding really mean? An embedding is a dense vector of floating point values. The length of a vector is a parameter that we specify. Instead of specifying the embedding manually, these are trainable parameters which are learned through the training process. It is common to see word embeddings that are 8-dimensional for small datasets and up to 1024 dimensions when we are working with really large data sets.

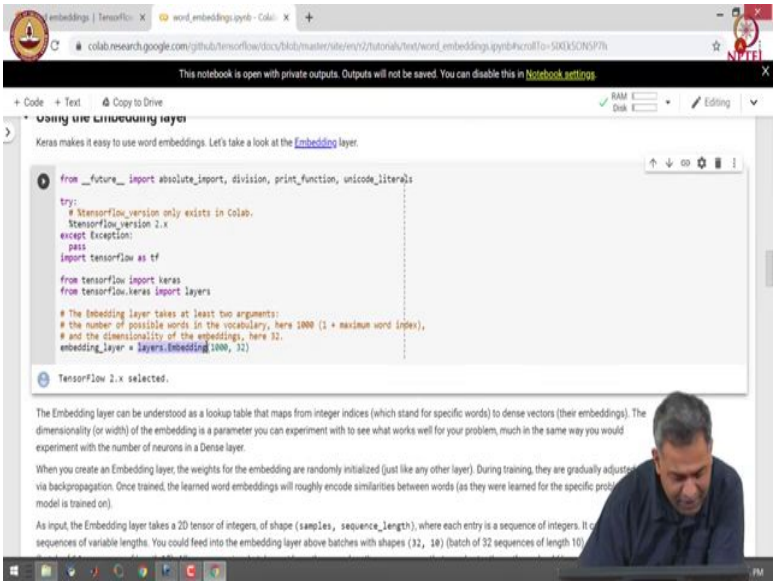
A higher dimension embedding captures more fine-grained relationships between words, but they take more time to learn. Let us look at some of the embeddings these are 4-dimensional embeddings for our example of the cat sat on the mat. So, here the word cat is embedded in 4-dimension where you can see that each of the dimension has the real number. So, this is a dense representation and it is quite efficient.

(Refer Slide Time: 07:16)



We can also think of an embedding as a lookup table. After the weights are learned we can encode each word by looking up the dense vector corresponding to it from the embedding table. So, keras has embedding layer that makes it easy for us to use the word embedding.

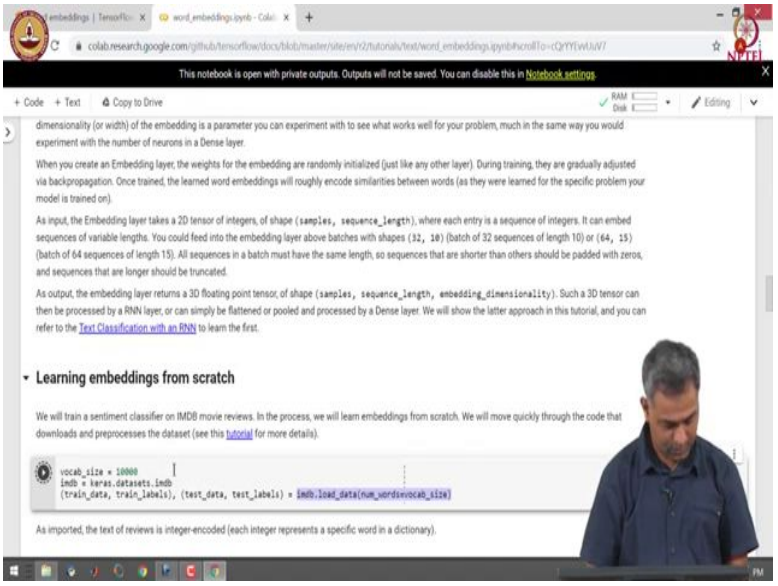
(Refer Slide Time: 07:49)



Let us see how to construct these word embeddings in tf.keras. So, here we use an embedding layer which is specified by layers.embeddings where we pass the number of possible words in the vocabulary which is total number of words plus 1 for padding and you also specify the embedding of and we also specify the dimensionality of the embedding which is 32 in this example. So, with this we can define an embedding layer.

The embedding layer can be understood as a lookup table that maps from integer indices which stand for specific word to dense vectors. The dimensionality of embedding is a parameter we experiment with. This is the same as any other hyper parameters that we set or tuning the neural network as an example the number of units in each dense layer.

(Refer Slide Time: 09:05)



When we create an embedding layer the weights of the embeddings are randomly initialized just like any other layer. During training they are gradually adjusted via back propagation. Once trained, the learned word embeddings will roughly encode similarity between words.

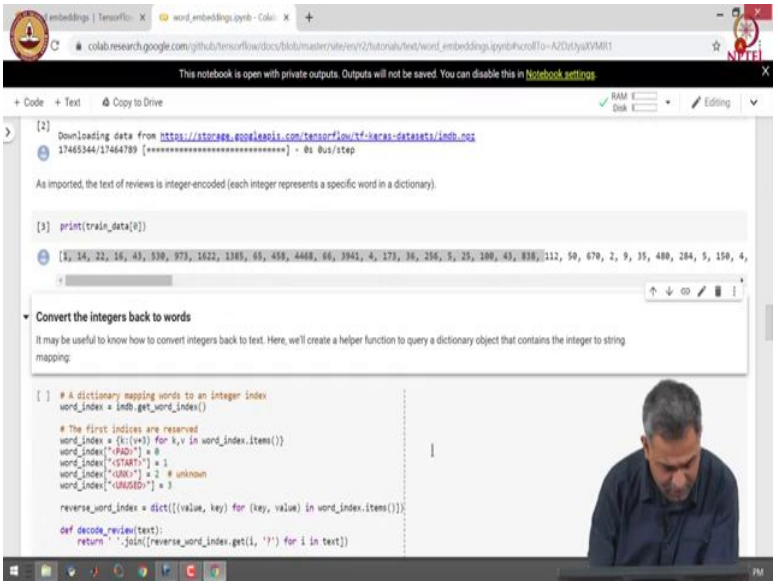
As input, the embedding layer takes 2D tensors of shape (sample, sequence length) where each entry is a sequence of integers. It can embed sequences of variable length. All sequences in a batch must have the same length. So, sequences that are shorter than the others are padded with 0 and the sequences that are longer are truncated. Here you have two batches of the shape (32, 10) where we have a batch of 32 sequences each of length 10. If we have a batch of shape say (64, 15), we have a batch of 64 sequences each with length 15.

As output embedding layer returns a 3D floating point tensor of shape (samples, sequence_length, embedding_dimensionality). The embedding layer tensor such a 3D tensor can be processed by a RNN layer or it can be flattened or pooled and processed by a dense layer.

Let us learn embeddings from scratch. So, here you will train a sentiment classifier on IMDB reviews. In the process we learn the embeddings from scratch. So, let us see find a vocabulary of size 10000 and we have IMDB data set from keras.datasets. IMDB data sets have movie reviews and we will train our embeddings from the movie reviews.

We use a load_data function on IMDB to get training and test data along with their labels.

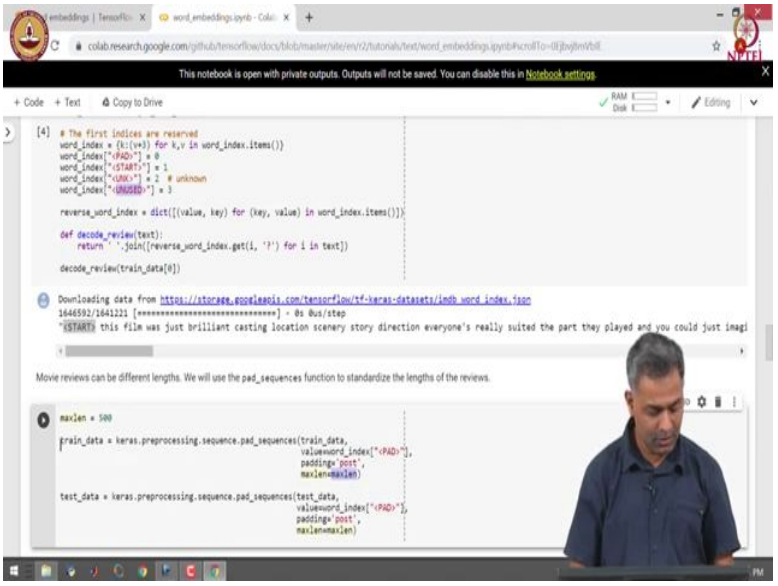
(Refer Slide Time: 12:01)



When we import this data, it is already integer coded where each integer represents a specific word in the dictionary. We can see that the first training example is represented with a list of integers where each integer corresponds to a specific word in the dictionary.

We can write a small decoder function to convert each integer back into the text.

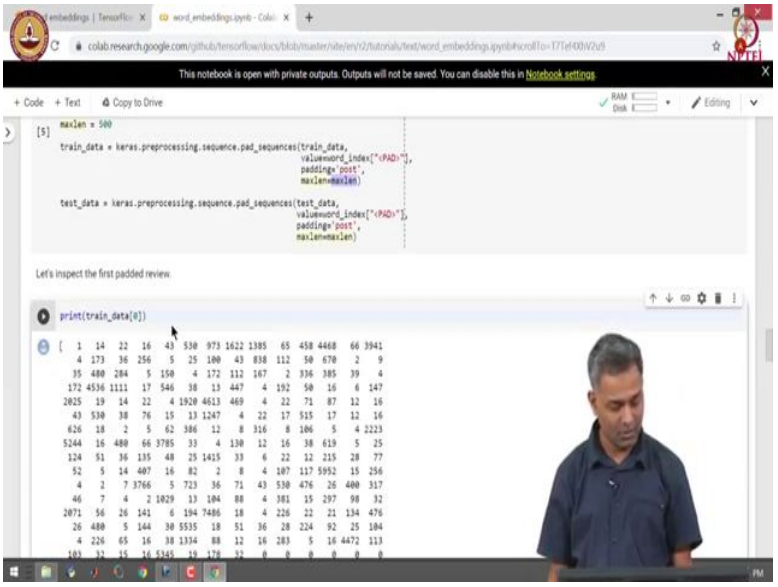
(Refer Slide Time: 12:38)



So, after applying the decoder function here we get the words corresponding to the first review. We define four special tokens which is PAD, START, UNKNOWN and UNUSED. The START token is inserted at the start of every review. So, movie reviews can be of different lengths and hence we use pad_sequence function to standardize the length of the reviews.

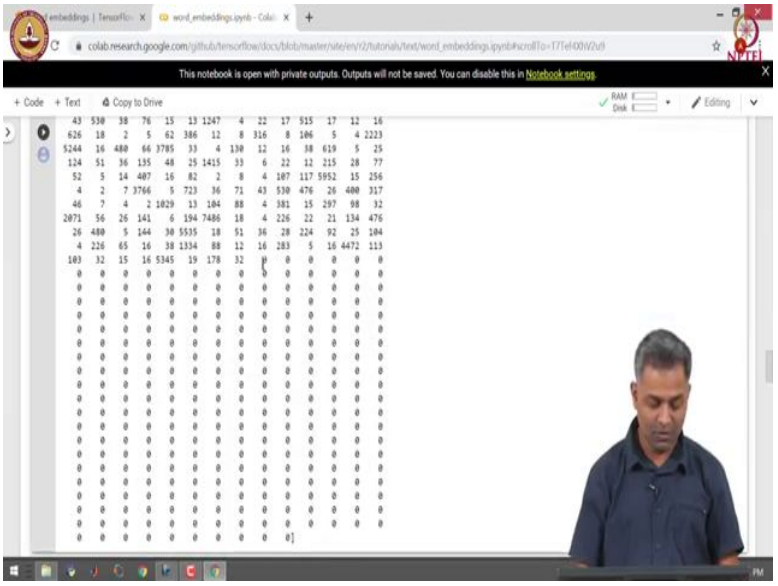
So we pad each of the sequence with a special token for PAD and we do padding at the end and we specify the maximum length for each of the sequence. So, if the sequence has lesser length than the max length we add those many pads. If the review is more than the max length we truncate the review. So, we apply the pad_sequences function on both training and test data.

(Refer Slide Time: 14:06)



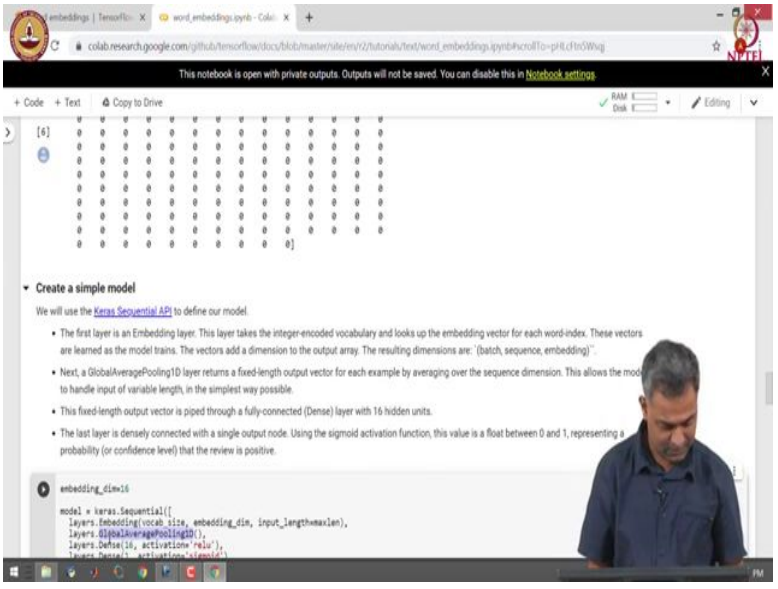
Let us look at the first example again.

(Refer Slide Time: 14:13)



So, you can see that the first example was less than 500 in length. So, you have padded that with number 0, which is the number corresponding to the PAD token.

(Refer Slide Time: 14:28)

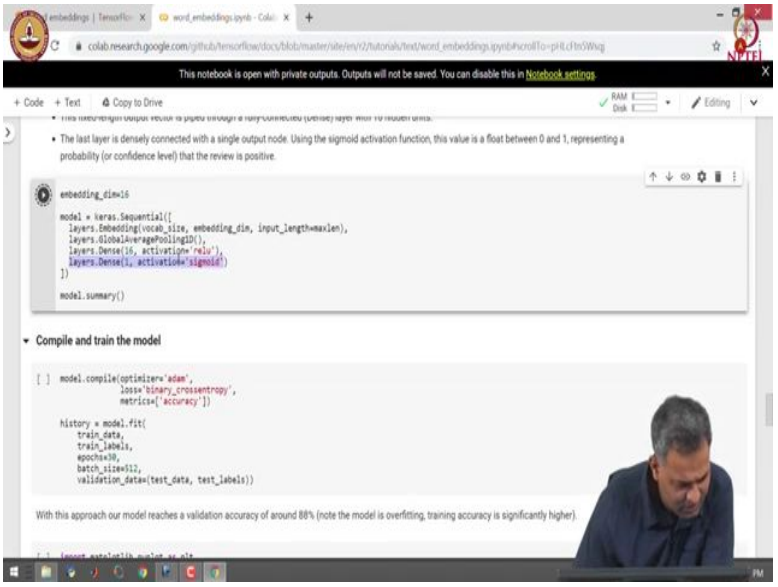


Now, what we will do is we have the integer representation for every sequence and each sequence also has the same length. So, let us define a model for classifying movie reviews and in the process we learn the embeddings. So, here we have embedding layer as the first

layer that takes the vocabulary size, the embedding dimensions and the input length which is 500 in this particular case.

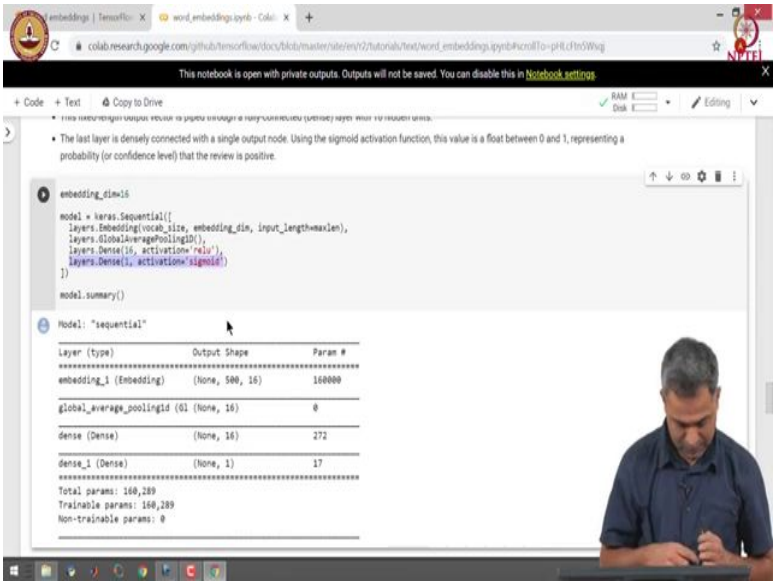
Next, we use global average pooling 1D that returns a fixed length output vector for each example by averaging over the sequence dimension. This allows the model to handle input of variable length in the simplest way possible.

(Refer Slide Time: 15:40)



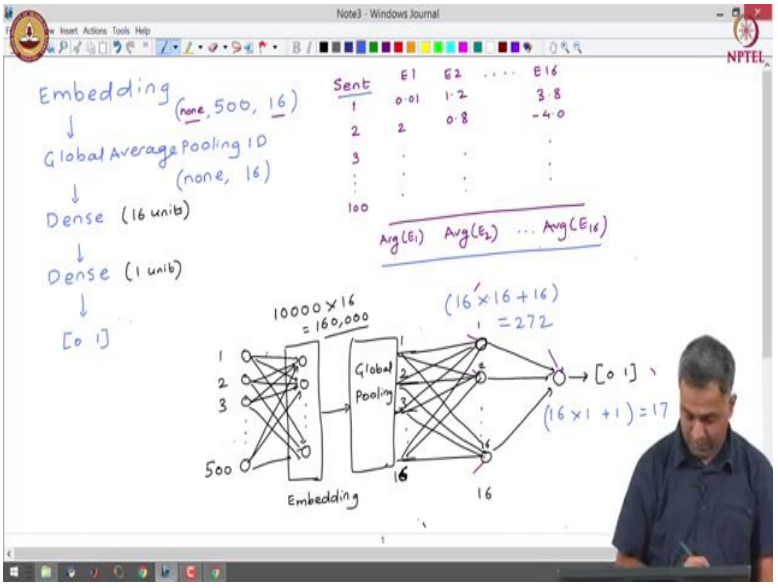
The fixed length vector is piped through a dense layer with 16 hidden units and we use ‘relu’ as an activation over here. Finally, we have a last layer which is a dense layer with a single output unit with ‘sigmoid’ activation function. With sigmoid activation function it returns a float value between 0 and 1 representing a probability that the review is positive. Let us look at the model summary.

(Refer Slide Time: 16:20)



So, let us draw this particular model in the note.

(Refer Slide Time: 16:39)



So, we have a model the first layer is an embedding layer followed by global average pooling 1D. The output of that is piped to a dense layer and we have another dense layer which is an output layer. This returns number between 0 and 1, and the input to the embedding layer is integer encoded strings; each string is of the same length.

So, we see that the input shape is 500 and you have defined embedding dimensions to be 16. We have defined embedding dimensions to be 16. So, what happens is we get a 500 length sequence and for each of the 500 words, you are going to return a 16th length embedding and we do it for multiple such kind of sequences. So, this is really a batch dimension.

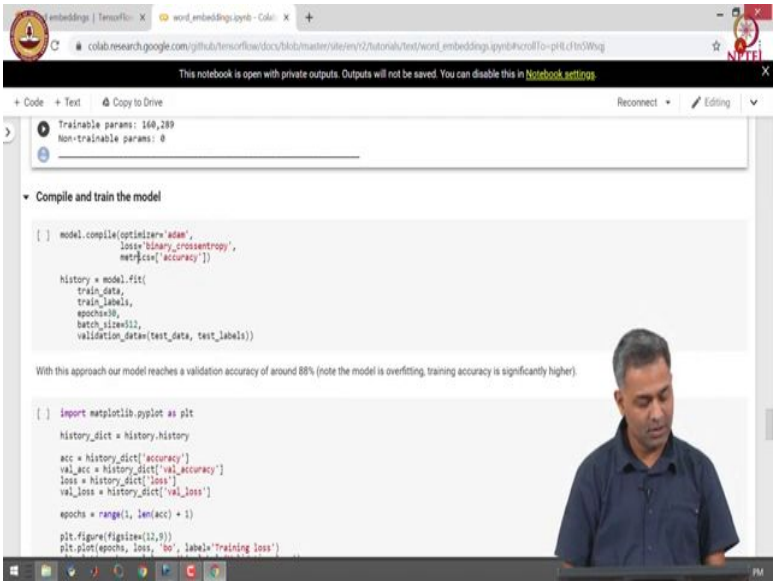
In global average 1D pooling what we do is we essentially average each embedding we average across each of the embedding. So, we take average of E 1, average of E 2 and so on average of E 16, correct? So, average pooling returns for every sentence we essentially get 16 numbers. So, we have average pooling returning 16 numbers. So, that is the output shape for average pooling.

So, global pooling use 16 numbers and these 16 numbers are fed into hidden layer with 16 units. Then, we have an output layer with a single unit which gives us which gives us the probability of reviewing positive.

So, you can see that in embedding, we have the vocabulary size is 10000 and we need to get embedding in 16-dimensional space. So, we have in all 16 cross 10000 which is 160,000 parameters to learn in the embedding layer. Global average pooling does not have any parameters. Now, each of the hidden unit has 16 inputs. So, there is a weight corresponding to each of the inputs and so, there are 16 parameters per unit and there is a bias parameter.

So, there are 16 parameters and there are 16 units plus 16 bias parameters that makes it to 272 parameters. And, you can similarly work out number of parameters for the output layer. Output layer has 16 incoming connections. So, there is a weight for each of the connection. So, there are 16 units, there are 16 weights per unit there is a single unit and then there is a single bias term. So, we have in all 17 parameters. So, the total number of parameters are 17 plus 272 plus 160k. So, it is 160289 parameters is what you see over here.

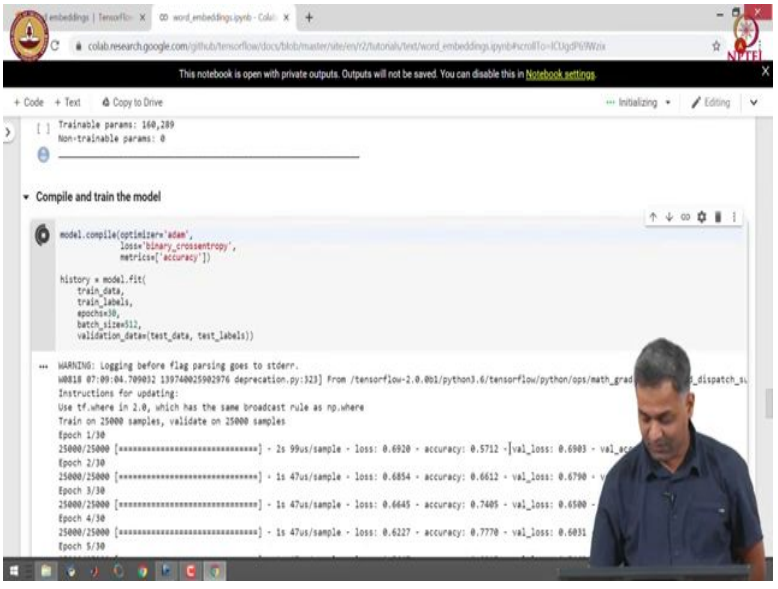
(Refer Slide Time: 25:21)



So, let us specify the optimizer the loss function and the metrics to track in the model compilation phase and fit the model with the training data and training labels.

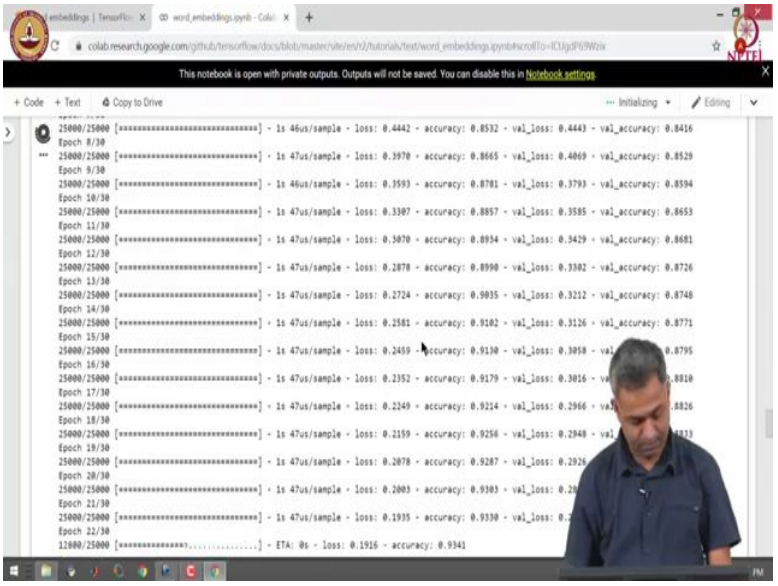
Let us train the model for 30 epochs with batch size of 512. We also use the validation data here to obtain the accuracy numbers for the validation data along with the training data. And, we capture the progress of training in the history object. Let us train the model.

(Refer Slide Time: 26:05)



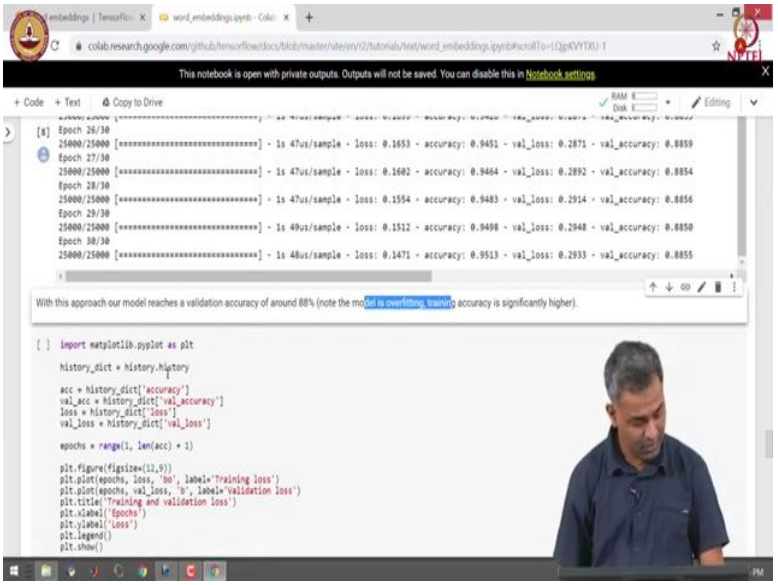
So, you can see that for the initial epoch we have loss of 0.69 for the training and loss of 0.69 for the validation.

(Refer Slide Time: 26:23)



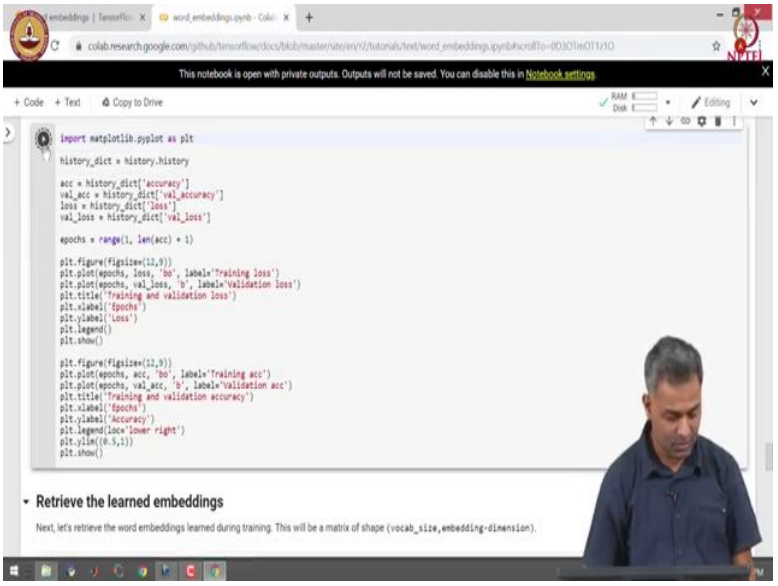
As the training progresses the loss is coming down and the accuracy is going up.

(Refer Slide Time: 26:33)



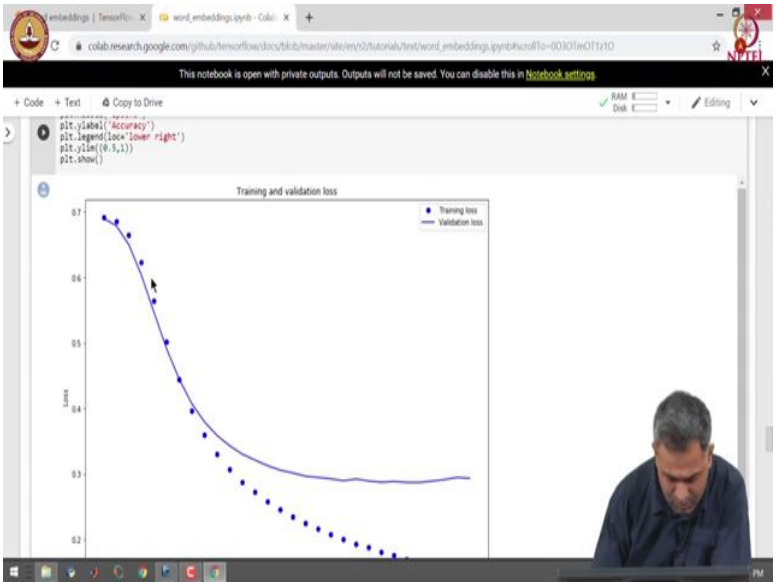
At the end of 30th epoch we achieved accuracy of 95 percent on the training data and about 88.5 percent on the validation data.

(Refer Slide Time: 27:18)

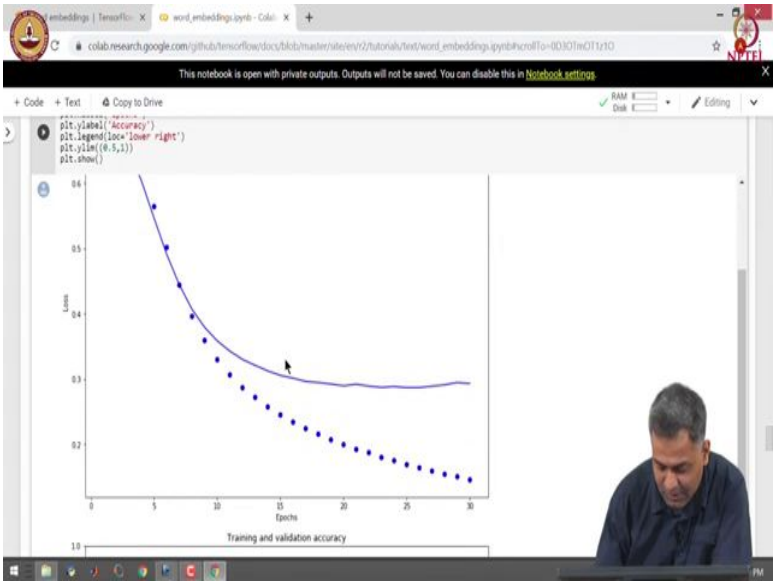


Let us plot the information that we have captured in the history object and see how the training progressed.

(Refer Slide Time: 27:26)

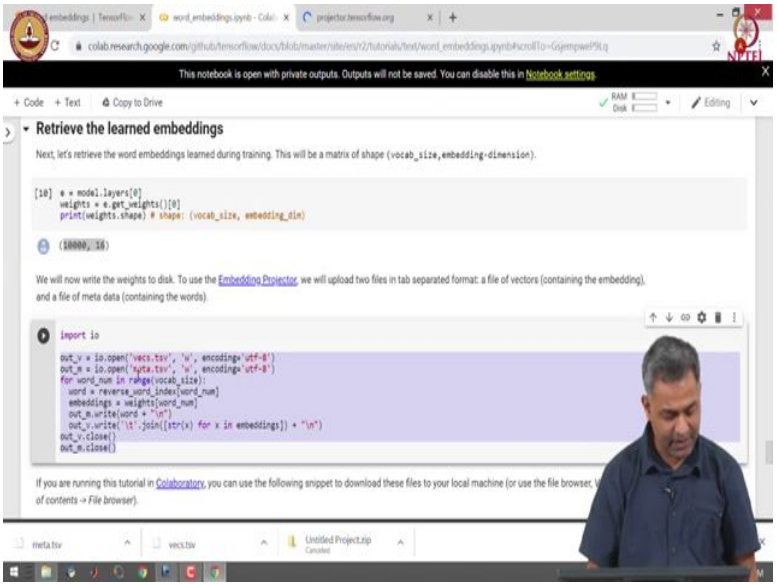


(Refer Slide Time: 27:33)



So, you can see that the training loss continues to go down as we train for longer, but the validation loss has plateaued after about 15 epochs.

(Refer Slide Time: 27:51)

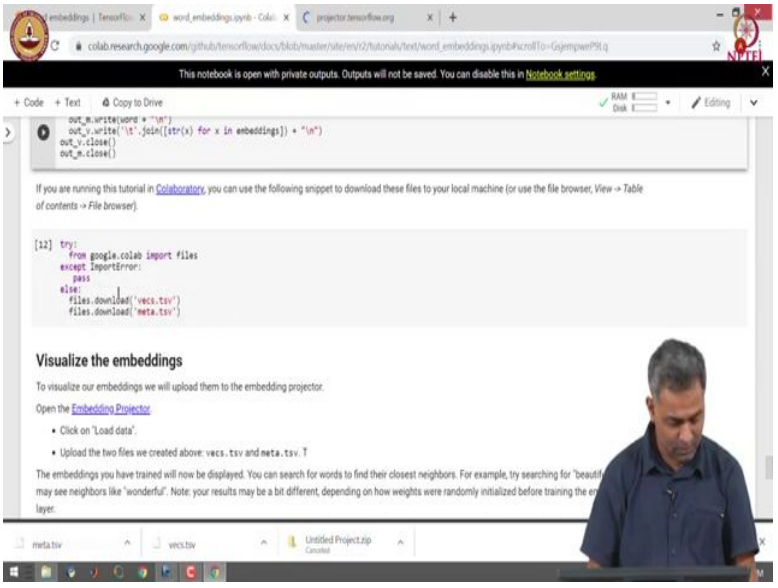


Let us retrieve the learned embeddings for the words. Note that we have trained the word embeddings during the training. So, we will have the embeddings with shape

(vocabulary_size, embedding_dimension). So, the vocabulary size was 10000 and embedding dimensions were 16. So, we get a tensor with shape (10000, 16).

We can write these weights to the disk and visualize them with embedding projector. Let us go to embedding projector and upload both these files. So, by executing this particular code we have returned the weights to the disk. We use embedding projector we need two files one containing embedding and then the second containing the word. So, the meta file contains the words and vecs file contains the embeddings.

(Refer Slide Time: 29:29)



We download both the files on the disk and we will go to embedding projector and load the data.

As an exercise, what you should do is you should upload both these files in embedding projector and visualize the embeddings. When you upload these two files you see embeddings like this and when you try searching for “beautiful” you will see neighbours like “wonderful”.