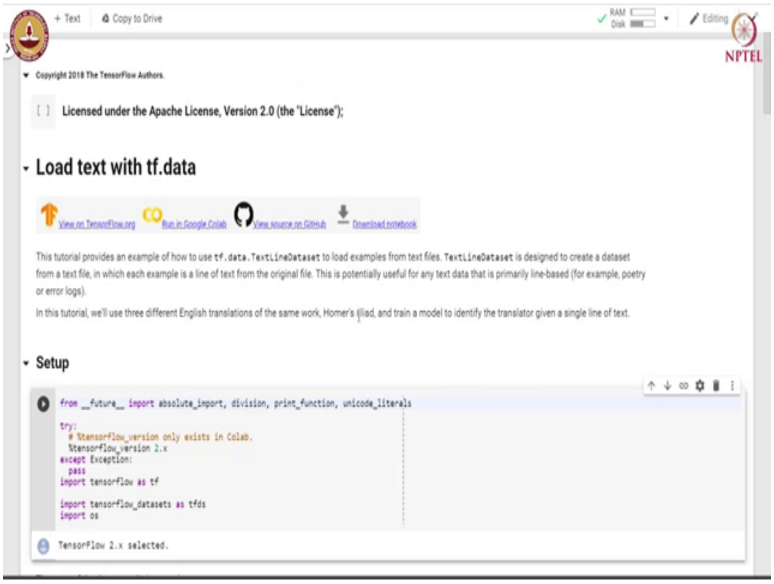
**Practical Machine Learning**
**Dr. Ashish Tendulkar**
**Department of Computer Science and Engineering**
**Indian Institute of Technology, Bombay**

**Lecture – 14**
**Building Data Pipelines for TensorFlow – Part 2**

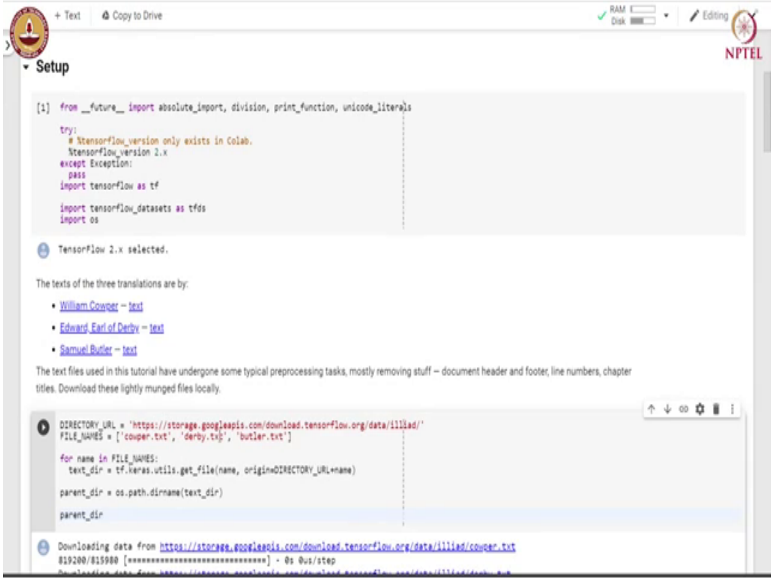
So, let us look at how to load the text data and create in pipelines based on the text data.

(Refer Slide Time: 00:22)




So, in this case we use tf.data.textLineDataset to load examples from text file into a dataset. The text line data set is designed to create a data set from a text file in which each example is a line of text from the original file. This is very important to note that each example is a line of text from the input file. This is potentially useful for any text data that is primarily line based. This kind of data sets could be a poetry, error logs, movie reviews etc.

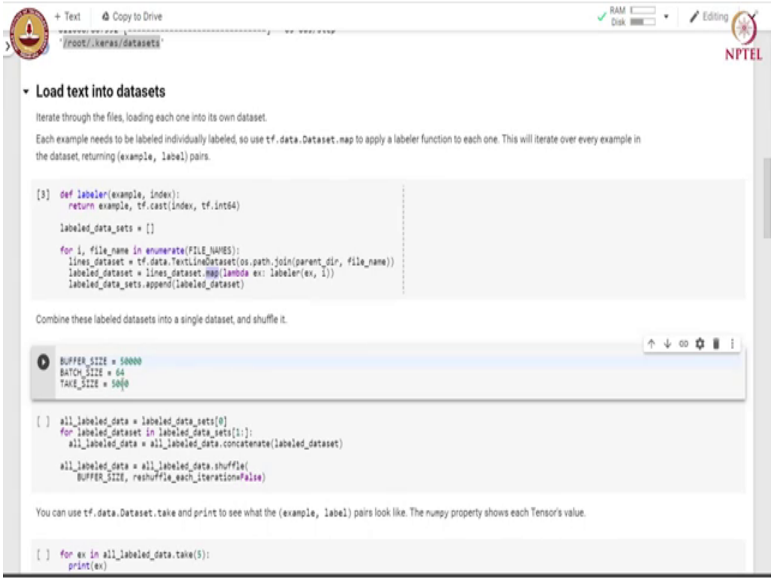
(Refer Slide Time: 01:14)




So, let us start by importing TensorFlow and TensorFlow data set. We will be using 3 different in English translations of the same work that is Horner's Iliad and see how to use text line data set and other pre processing on the text to create a data set that can be fade into the model for training. So, we have text of 3 translations as input. So, there are 3 files cowper.txt, derby.txt and butler.txt which is translation of Homer Iliad.

We first download these files and then print the parent directory. After getting the data the normal pre-processing tasks are about removing document header and footer, line numbers, chapter titles if those are present in the file. So, you can see that now the files are downloaded and they are in root/.keras/datasets directory.

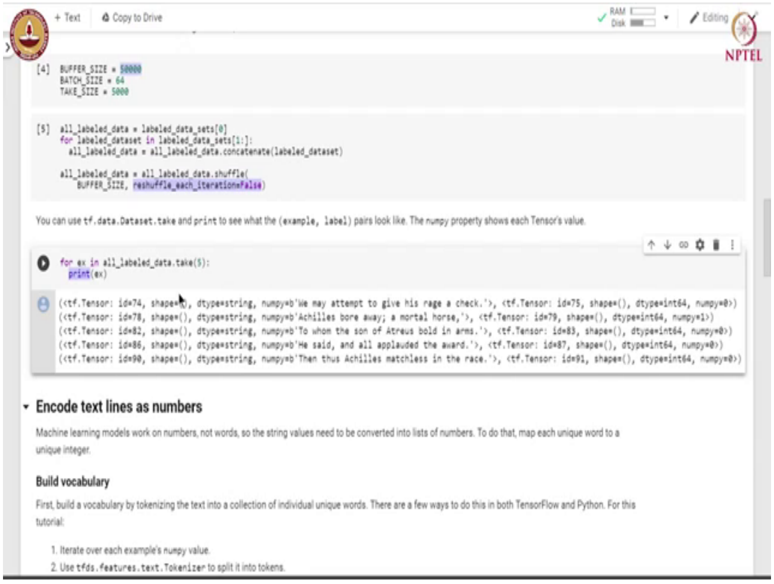
(Refer Slide Time: 02:45)




Now, that we have downloaded the files we will go through each of the file. We will load them into a dataset each file will be a single dataset, we need to label each example for that purpose we use dataset.map() function that applies a labeler() function on each example. So, labeler() function takes an example and assign a label to that particular example and we do that with a map() function.

So, map will iterate over every examples in the dataset and we will return example comma label pairs. So, let us apply the labels on each of the example by running this particular code cell. We will combine this label dataset into a single dataset and shuffle it for that we set the buffer size to 50,000. We are going to use batch size of 64 and we will look at 5,000 examples.

(Refer Slide Time: 04:05)




So, let us concatenate the individual datasets into a single dataset. You can see that first we copy the first dataset into all label data and then for the remaining datasets we combine we over remaining datasets and concatenate them to the all label data. So, at the end of this particular process all the datasets are concatenated into a single dataset which is all label data.

After concatenating all datasets next we shuffle the datasets using the buffer size of 50,000 and we set the *reshuffled_each_iteration* parameters to *false*. So, we do not want to reshuffle the dataset at each iteration then we can use take() and print() functions to see what example pair look like. So, what we do is we take 5 examples from all label data and we print each of the example.

We have 5 tensors for first 5 example, one per each example. We can see that each tensor is a scalar quantity or a 0 dimensional tensor. It is of type string and we see the string corresponding to this particular tensor. Corresponding to each of the sentence there is a label associated with it which is another scalar and a label for the first sentence is 0.

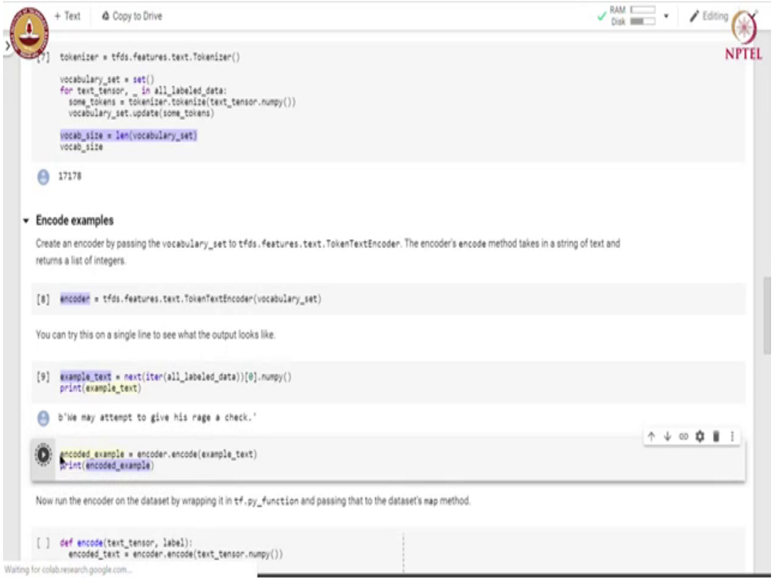
Second sentence has got label of one and so on. So, you can see the first 5 examples or first 5 examples from all_labeled_data. Now you know that machine learning models work on numbers and not words. So, naturally we need to convert these words into numbers.

(Refer Slide Time: 06:58)


We can map each unique word to unique integer. So, first you build a vocabulary by tokenizing the text into collection of individual words. There are few ways to do this both in TensorFlow and Python. Here we will iterate over each example and we will tokenize each example into tokens. We then collect each of the tokens and remove the duplicate.

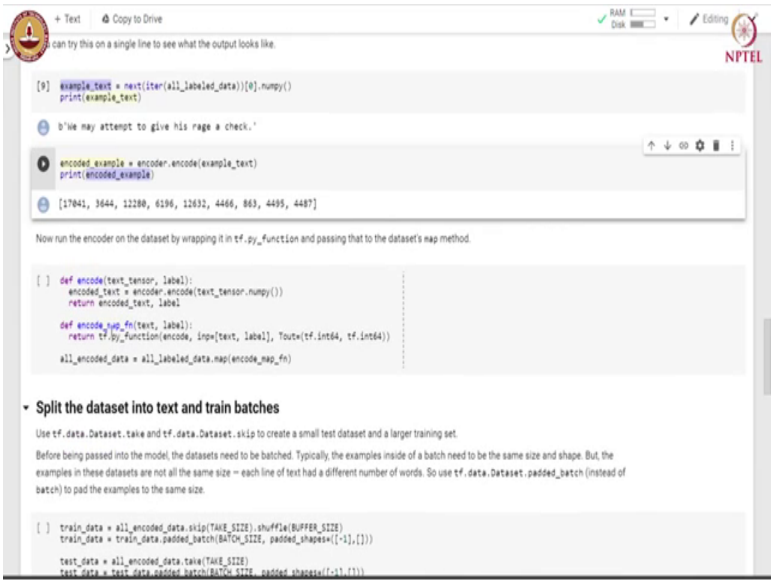
So, we can note that the vocabulary set is a set that we update with the tokens that we get after tokenization. So, the effect of that is the duplicate tokens are removed. Finally, after completing the process for each sentence we can find out how many words are there in the vocabulary just by taking the length of the vocabulary set.

(Refer Slide Time: 08:37)




So, we have 17,178 words in our vocabulary. We create an encoder by passing the vocabulary set to token text encoder which takes a string of text. The token text encoder has encode() method that takes a string of text and returns a list of integers. We take the example text and we encode that example text with the encoder defined over here and then we print the encoded example. So, we can see that the first text here is we may attempt to give his rage a check.

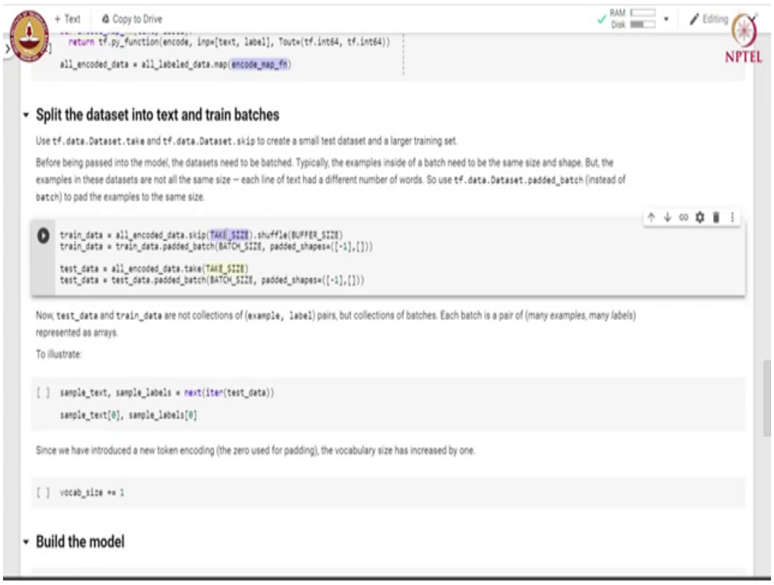
(Refer Slide Time: 09:36)




Let us see how this example is encoded. Each of the word over here is encoded with a unique integer. Now, that we have seen how the encoder works on a single example we will apply the encoder on each example in the dataset. For that we wrap the encode under tf.py_function().

The encode() function takes a takes and takes a text tensor and label and returns the encoded text along with its label. So, we use tf.py_function() to wrap encode and we give the text and labels as inputs. So, we use a map() function to apply the encode_function_map on each example. So, that each line gets converted into its numeric representation.

(Refer Slide Time: 11:03)




The next task is to split the dataset into test and training batches. We use *tf.data.dataset.take* and *tf.data.dataset.skip* to create a small test set and a large training set. Before being passed into the model we need to batch the dataset. Typically the examples inside a batch need to be of the same size and shape. But the examples in these datasets are not all of the same size. Remember each line of text had a different number of words. So, we use *padded_batch* instead of *batch* to pad each example to make it of the same size. So, let us see how to do that. So, let us first conceive how to construct the training data.

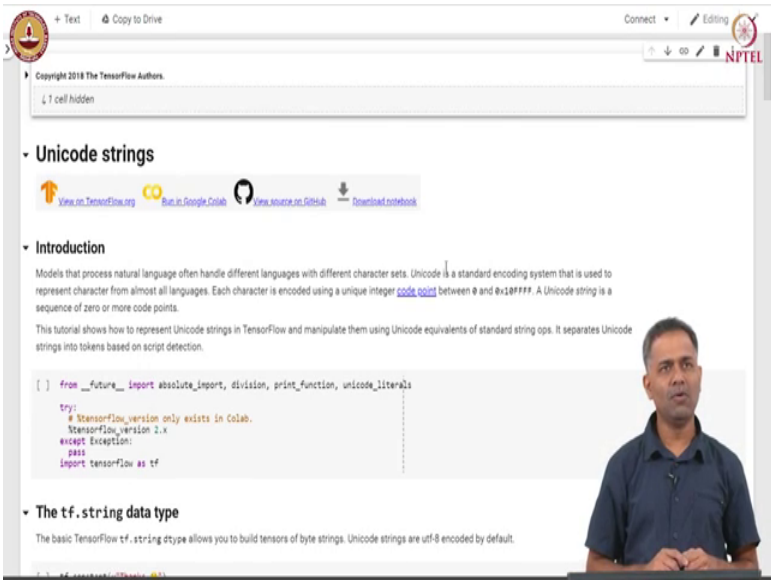
So, we take all encoded data and we skip first 5,000 examples because we have taken take_size to be 5,000. Then we shuffle these examples. So, by skipping the first 5,000 examples and shuffling we get the training data and we take these 5,000 examples that we skipped in the training data to form the test data. Then for training and test data we apply a padded batch so that each example is made of the same size by padding.

So, let us run this code cell now we have test data and training data ready. So, test_data and train_data are not collections of (examples, labels), but they are collection of batches and each batch is a pair of many (examples, labels) represented as arrays.

Let us look at the first batch. We use an iterate over the test data and we take the next. So, that this next returns the text and labels which we store in sample_text and sample_labels and then we examine them. So, you can see that the first example is already converted into numbers and it is padded with 0 so that its length is equal to the length of the longest sequence in the dataset and we also have a label which is 0 in this particular case.

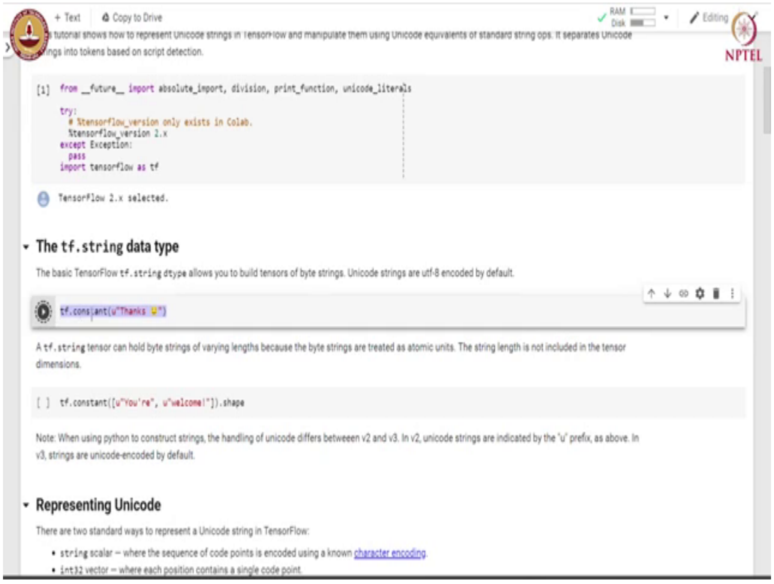
Since we have introduced a new token that is 0 for padding we increase the size of vocabulary by one. So, these are steps that we take in order to convert the text data into dataset. We use text line data set method to construct data set object from the text data. Now that we have constructed training and test data we can use that for building for training the model.

(Refer Slide Time: 15:38)



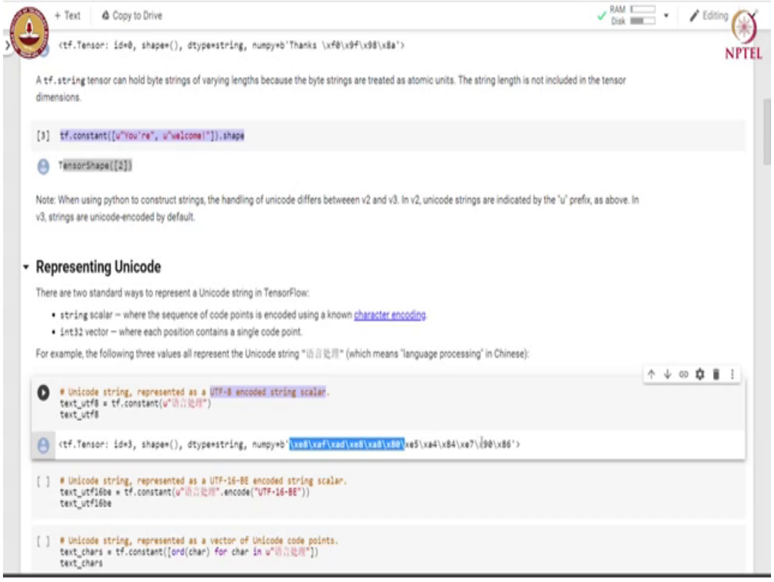
Let us understand how we can process unicode strings with TensorFlow and also understand how to construct data set objects with unicode strings. So, we can use TensorFlow to process different languages with different character sets. Unicode is a standard encoding system that is used to represent characters from almost all languages. Each character is encoded using a unique integer code point between 0 and this particular number. A unicode string is a sequence of 0 or more code points.

(Refer Slide Time: 16:46)




So, the *tf.stringdtype* allows us to build tensor of byte strings. Unicode strings are UTF-8 encoded by default. So, let us see how this thanks with emoji is represented. We can define that with *tf.constant*.

(Refer Slide Time: 17:19)

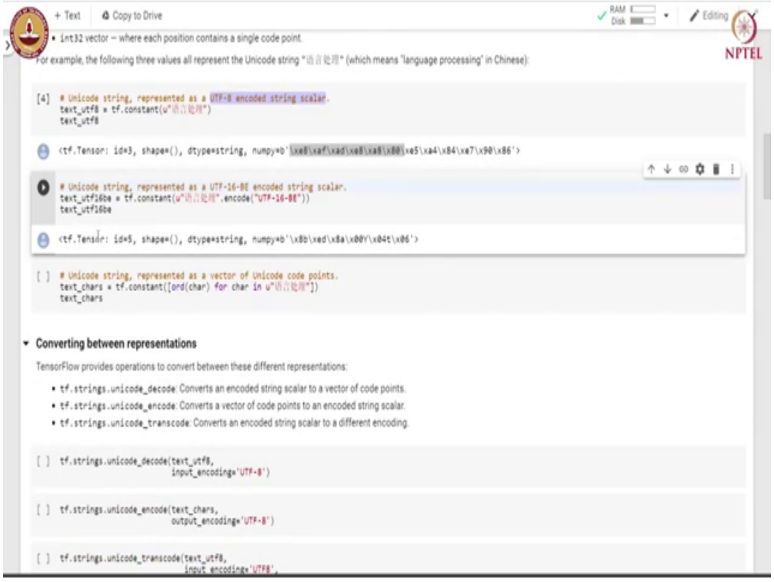



A *tf.string* tensor can hold byte strings of varying length because the byte strings are treated as atomic units. The string length is not included in the tensor dimension. Let us look at the shape of these 2 strings which has “You are” and “welcome” as two strings. So, you can see that this is a tensor which is a vector containing 2 elements or in other words this is a 1d tensor with shape 2 there are two ways to represent a unicode string in TensorFlow.

One is using a string scalar or integer 32 vector. The string scalar stores the sequence of code points with a known character encoding, whereas integer 32 vector stores each position containing a single code point.

So, as an example the following 3 values are represented in a unicode string. This is a Chinese character for language processing. Here, we are using unicode strings represented as a UTF-8 encoded string scalar. So, you can see that this is a scalar with d type string and we have UTF-8 encoded strings string scalars.

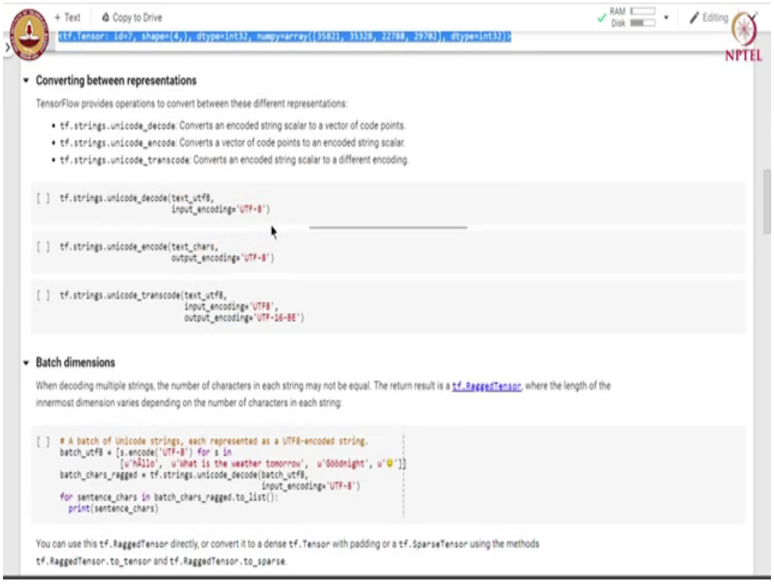
(Refer Slide Time: 19:21)




Here we have a unicode string represented as UTF-16-BE encoded string scalars and finally, we have the same unicode string represented as a vector of unicode code points. So, this text character is a vector containing 4 elements and each element is a unicode code point corresponding to the character.

So, this 35821 is a code point corresponding to this particular character 35328 is the code point corresponding to this character and so on. So, there are 4 characters and there are 4 code points corresponding to each one of them. So, we just saw 3 ways of representing strings one using UTF-8 encoded string scalar then UTF-16-BE encoded string scalars and as a vector of unicode code points.

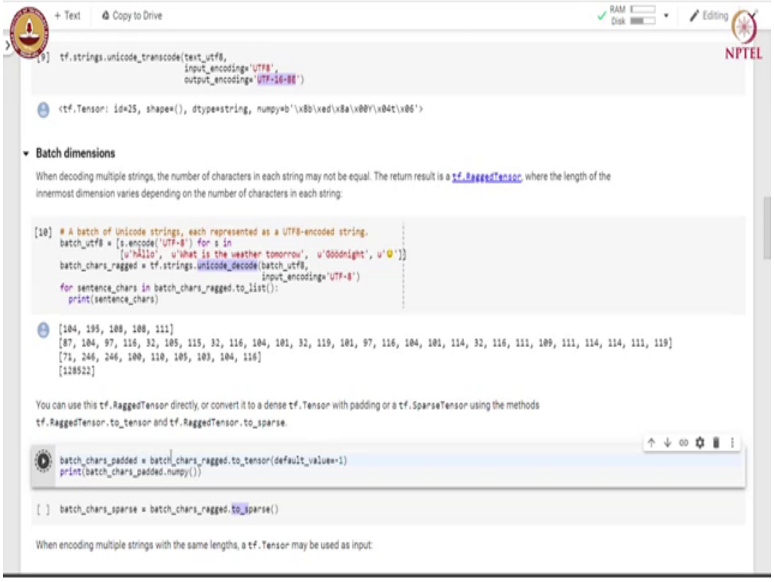
(Refer Slide Time: 20:38)




TensorFlow provides operations to convert between these different representations. We use tf.strings.unicode_decode to convert an encoded string scalar to a vector of code point. Unicode_encode converts a vector of code points to an encoded string scalar and unicode_transcode converts an encoded string scalar to a different encoding. Let us look at the examples. Here we use unicode_decode. This text_utf8 will get converted to a vector of code points.

Let us take the vector of code points and convert that to a string scalar using unicode_encode method. When we apply that the vector of code points get converted into a string scalar encoded in UTF-8 finally, we can use encode_transcode to convert between UTF-8 encoding and UTF-16-BE encoding. So, the input encoding is UTF-8 and the output encoding is UTF-16-BE.

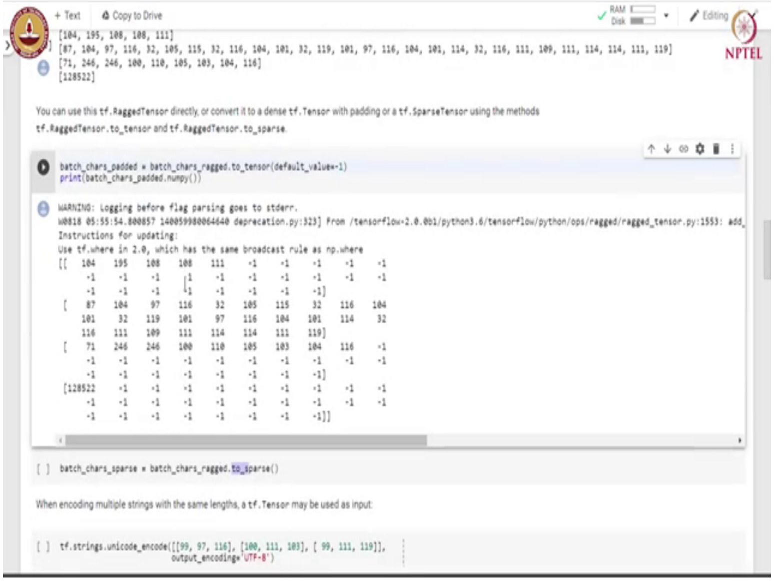
(Refer Slide Time: 22:34)




When decoding multiple strings the number of characters in each string may not be equal. So, it returns tf.RaggedTensor where the length of inner most dimension varies depending on the number of characters in each string. We take a batch of unicode strings each represented as a UTF-8 encoded string. So, we have hello what is the weather tomorrow then goodnight and an emoji for smiley.

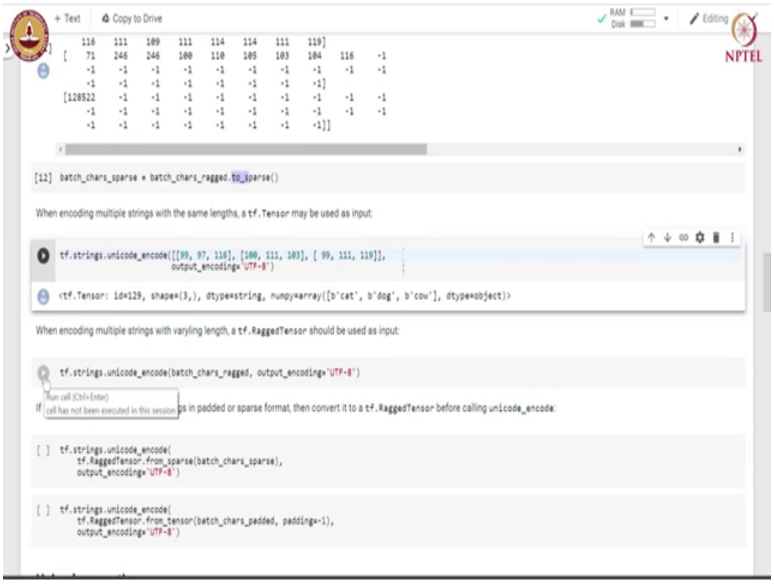
Use encode string() function that returns we use unicode_decode() function to convert the characters into a string of unicode code points. So, we get a RaggedTensor and you can see that the length of each vector resulting vector is different. We can use this RaggedTensor directly or convert that into tf.tensor with padding or to tf.sparse tensor using to_sparse() method.

(Refer Slide Time: 24:21)



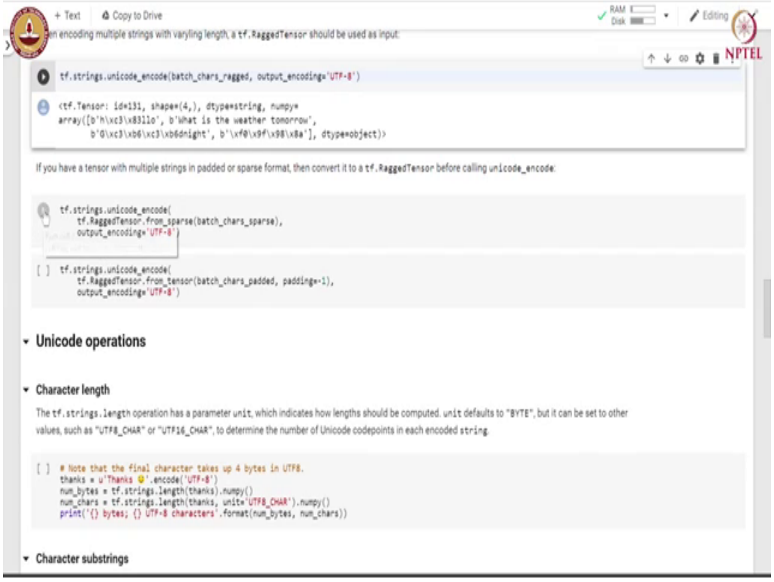

Now, here we converted the RaggedTensor to a dense tf.tensor. Here we pad each example with -1. So, that the lengths of all the examples are the same.

(Refer Slide Time: 24:47)




When encoding multiple strings with the same length a tf.tensor may be used as an input. When encoding multiple strings with varying length a tf.RaggedTensor should be used as an input.

(Refer Slide Time: 25:10)




If we have tensors with multiple strings in padded or sparse format then convert it to a RaggedTensor before calling unicode encode.

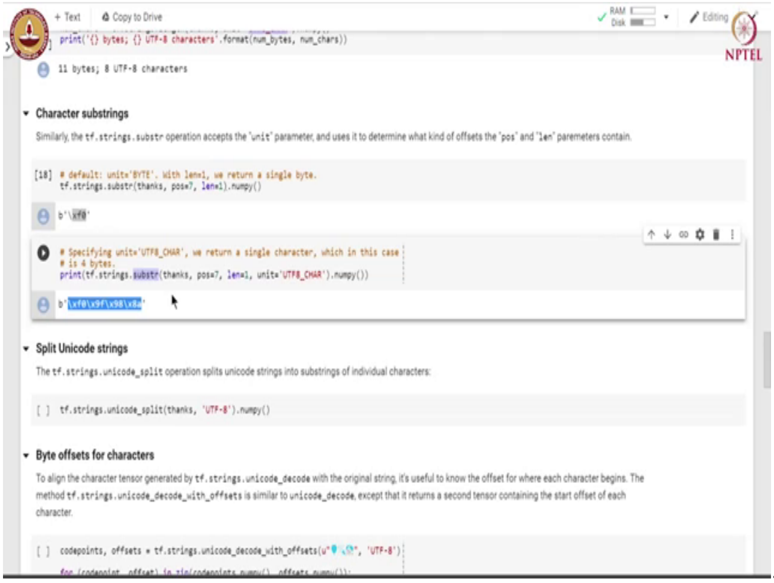
(Refer Slide Time: 25:37)


Let us look at some of the unicode operations. Let us see how to find the length of the unicode string. So, the tf.strings.length has a parameter unit which indicates how length should be computed unit defaults to bytes, but it can be set to other values such as

UTF8_CHAR or UTF16_CHAR to determine the number of unicode code points in each encoded strings.

So, let us take an example where we take thanks and smiley emoji and encode that using UTF-8 encoding store that in the thanks and we can use the length without any argument. So, we get number of bytes and if you want to find out number of characters we specify the unit as UTF8_CHAR and get the length. This particular string is encoded with 11 bytes and there are 8 UTF8_CHAR.

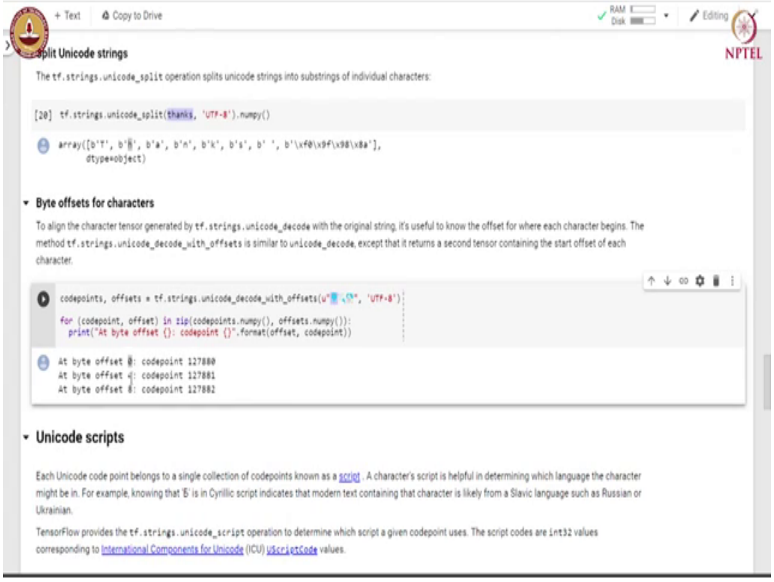
(Refer Slide Time: 27:06)




We can use tf.strings.substr() operation to get character sub strings. It also accepts the unit parameter and uses it to determine what kind of offsets the position and length parameter contain. Here the default unit is byte with length 1 we return a single byte. Here the unit is UTF8_CHAR and we return a single character which in this case is 4 bytes.

So, are you getting the difference between substrings here and substring here we are not specifying unit. So, by default byte is taken as the unit. Here we get a single byte and here we get single character which is 4 bytes. We can also split the unicode string using unicode_split() operation.

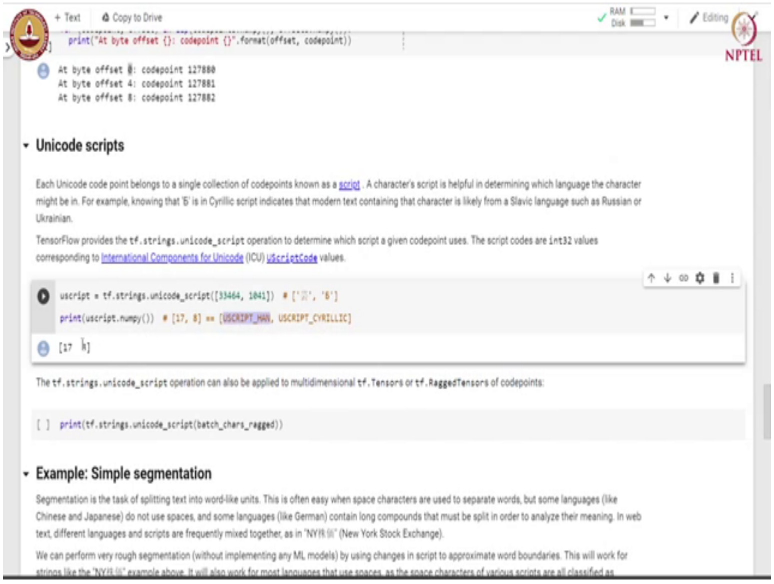
(Refer Slide Time: 28:31)




To align the character tensor generated by tf.strings.unicode_decode with offset to align character tensors generated by tf.strings unicode_decode with the original strings. It is useful to know the offsets for where each character begin for where each character begins the method tf.strings.unicode_decode_with_offsets is similar to unicode_decode except that it returns a second tensor containing the start offset of each character.

So, let us use unicode_decode_with_offset on a string of emoji which is encoded in UTF-8 it returns code points and the offsets. We print the offset and the code point. You can see that the first emoji is represented with code point 127880, the second emoji is encoded with this particular code point and third one is encoded with this particular point. We also get corresponding byte offsets of each of the emoji in the original string.

(Refer Slide Time: 30:37)




Each unicode code point belong to a single collection of code points known as a script. A character scripts is helpful in determine which language the character might be in for example, knowing that this particular character is a Cyrillic script indicates that modern text containing that character is likely from a Slavic language such as Russian or Ukrainian. TensorFlow provides unicode_script operation to determine which script a given code point uses. The script codes are so you can see that they have script 17 and 8 17 corresponds to Han Chinese and it corresponds to Cyrillic.