# **Deep Learning Foundations: Natural Language Processing with TensorFlow**

### **Leveraging deep learning for natural language processing**

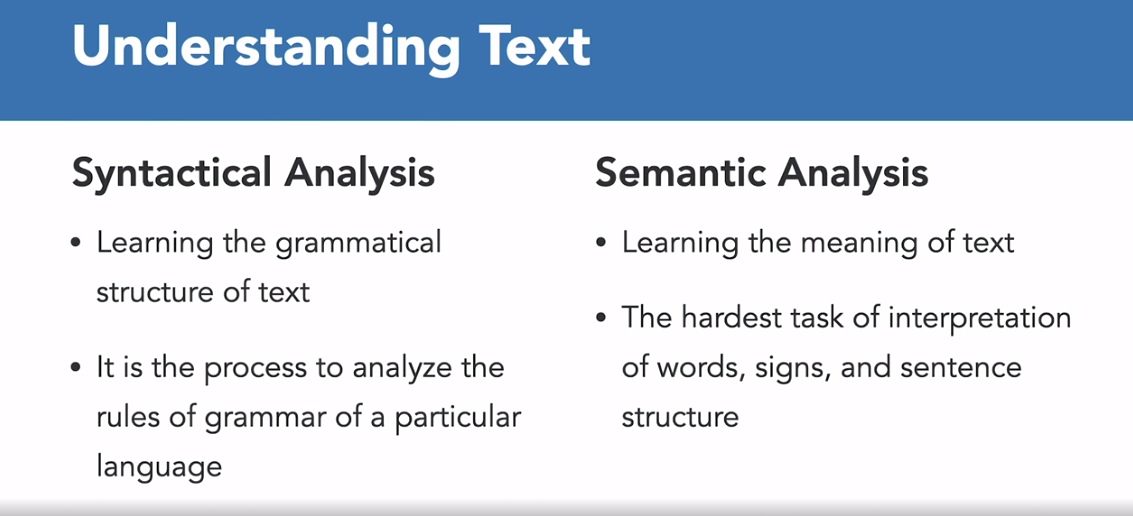
Selecting transcript lines in this section will navigate to timestamp in the video

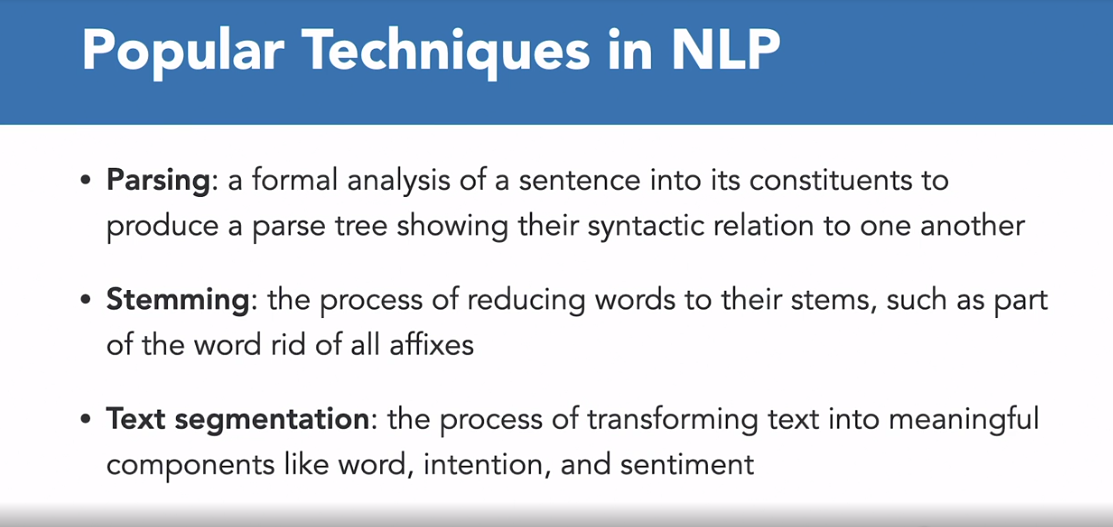
- [Harshit] Deep learning is a type of machine learning that tries to mimic the functioning of the human brain and can train over huge amounts of data to offer. Hi, I'm Harshit Tyagi. In this course, we are going to explore the foundations of deep learning and natural language processing. We'll use a technology called TensorFlow. Now, TensorFlow is an open source platform that is efficient and free. Within this course, we'll start by understanding the fundamentals of NLP, including the syntactic and semantic analysis, why deep learning performs better than classical machine learning algorithms, and we will learn to pre-process the data before feeding it into the deep learning model. Moving along, I'll show you how to apply word embeddings and visualize dense word vectors on TensorFlow Projector. Building on embeddings, we will learn to classify movie reviews, headlines using recurrent neural networks, and convolutional neural networks to further improve the model's performance. We'll learn how to generate poetry like Shakespeare using RNNs. So, let's get started.

### **Introduction to natural language processing**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] Let's start off by answering what is natural language processing. Now, by definition, i**t is an area of computer science and artificial intelligence concerned with the interaction between computers and humans through natural language.** The ultimate goal of NLP is to help computers understand language, as well as humans too. **Now, it is the driving force behind applications such as virtual assistants, speech recognition, sentiment analysis, automatic text summarization, machine translation, text generation, and many other applications as well.** Now, there are two primary techniques that lead to understanding of NLP, **syntactic analysis and semantic analysis.** **Now, language is a set of valid sentences, but what makes a sentence valid? Syntax and semantics. Now, syntax is the grammatical structure of the text, whereas semantics is the meaning that is being conveyed. Syntactic analysis is done to analyze the rules of grammar of a particular language, whereas semantic analysis is the process of interpreting words, signs, and sentence structure. A sentence that is syntactically correct, however, is not always semantically correct.** For example, dogs scream silently. It's grammatically valid because it has subject, verb, and adverb, but it doesn't make any sense. So, let's look at a few commonly used techniques in natural language processing. The first is parsing, which resolves a sentence into its component parts and describes their syntactic rules, and it does that using a parse tree. Now, a parse tree also provides us with information about the grammatical relationships of the words due to the structure of their representation. Now, the other technique is stemming. Now this is the process of reducing the words to their stems. It is the part of the word that remains after the removal of all the affixes. For example, the stem of the word learned is learn. Now learn is also the stem of the word learning, and so on. Now, stemming is important because we can encounter many variations of a word that actually have the same stem and meaning. Then we have text segmentation. It is the process of transforming text into meaningful units like words, sentences, different topics, the underlying intent, and more. Which again, is a difficult task because of the complexity of human language. **Lastly, we discuss deep learning, which has produced the best results in NLP as compared to all the other techniques. Now, deep learning uses vectors that embed syntactical and semantic information. With success of these embeddings, neural networks could achieve groundbreaking results in natural language processing.**





### **Introduction to word encodings**

Selecting transcript lines in this section will navigate to timestamp in the video

- Now that we are aware of the success of deep learning and understanding natural language. Let's understand the fundamentals of preprocessing textual data. So, computers understand numbers and not textual characters. So, we have to find ways to encode characters. We could take character encodings for each character in a set, for example, the ASCII values. But the question is, will that help us understand the meaning of a word? So for example, consider the word "Arms" as shown here and the ASCII values of each character in the word. So, you might think you could have a word like "Arms" encoded using these values. But the problem with this, of course, is that the semantics of the word are not encoded in the letters. This could be demonstrated using the word "Mars", which has a very different meaning but with exactly the same letters. So, it seems that training a Neural Network with just the letters could be a daunting task. So, let's try to consider words. We'll try to give words of value and have those values used for training a network. For example, consider the sentence, "It is a sunny day". We'll assign a value to each word, where, what that value is doesn't matter. This value is the same for the same word every time. So, a simple encoding for the sentence would be, for example, to give word "It" the value one, "is" the value two, "a" a value of three, "sunny" a value of four and so on. So, this is how we can start training a Neural Network based on words. Now fortunately, TensorFlow and keras, give us some APIs that make it very simple to do this. So, let's look at the code for this. Now, the first step is to import the required libraries the required APIs that will do all the heavy lifting. Now, there are many ways to tokenize or create these word encodings but we are going to use tokenizer. For the first step is to import the main library. The main API, which is TensorFlow. So, for that, we were write import TensorFlow as the IF, which is the convention. Abbreviated TensorFlow STF. Then the second step is we need to import keras API from the TensorFlow library and since we are using the tokenizer model, the tokenizer class to generate the word encodings. So, we need to import that class from keras dot preprocessing dot text module. Now run the cell. Now the next step is define your train sentences. So, we have just added one single sentence, "It is a sunny day" in a pattern list. So, let's quickly run that as well. The next step is to set up the tokenizer. Now here, what we need to do is first of all instantiate our tokenizer, the class that we have imported above and we have added a passive parameter called num\_words, which is equal to 100. Now we're using 100, which is way too big as there are only five distinct words in this data. So, if you're creating a training set based on a lot of text, so you usually don't know how many unique distinct words there are in that text. So, by setting this hyper parameter what the organizer will do is take up the top 100 words by volume and just encode those. So, it's just a handy shortcut when dealing with lots of text, a lots of data. Now the next step is to actually train the tokenizer on the training sentences. So, the method for that is, fit\_on\_text. So, this is the function and you past a training sentences that you have defined above. And the next step is to generate. So, you store all the word index for the words in the sentence and we have the attribute called word\_index, which is available in the tokenizer object. And I'll run this. And once you run this, the next step is look at the dictionary that the word index had generated. So, print word index, and you can see, we have one assigned to "it", two assign to "is", three assigned to "a". So, we have generated the entire dictionary of all the distinct words present in our trainings sentences list. Now, let's add another sentence to this list. "It is a cloudy day", and then run it again and set it up the tokenizer and train the tokenizer, based on the training sentences. Now, if you run the print word in the Excel, so you see, all of the words have been assigned the same value as earlier but the only new word was "cloudy" and it has been given the unique incremental value of six over here. So, you've just seen how we can use tokenizer class and TensorFlow to generate word encodings of the sentences of the textual data that we have. The next step is to convert these sentences into sequences, so that we can pass it to the Neural Network. And we'll do that in the next video.

### **Tokenization using TensorFlow**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] So we have seen how to tokenize the words and sentences, building up a dictionary of all the words to make a corpus. The next step will be to turn these sentences into list of values based on the tokens that we have generated from the tokenizer object, which are called sequences. So first of all, we're going to use the same tokenizer class so let's quickly import that from Kera's pre-processing text module. Let's quickly around this. So once that is done, the next step is to define the training sentences. And for that, we have defined a Python list called "twin sentences," and I've defined three sentences. So "it is a sunny day," "it is a cloudy day," and "will it rain today?" I've also added a question mark just to see how the tokenizer handles it. So let's quickly define these sentences. The next step is to train the tokenizer. So here for that, first of all, we need to instantiate the tokenizer class. And we have provided the non-words hyperparameter, which is set to 100. The next step is to train the tokenizer, which is again using the function "fit on text," as we saw earlier. And we are going to use the training sentences that we have defined above as the argument to this function. And the next step is to store word index, the word encoding dictionary, that the tokenizer is going to generate from the word index attribute. Store it, run the cell. Now, comes the main part, which is creating sequences of these trainings sentences. So for that, we are going to use text-to-sequences function and we are going to pass the training sentences on which we are going to run this function. And it will create the sequences for all the sentences that have been fine in this list. Let's run that. Now, what we're going to do is we are going to print both the word and coding dictionary, which is "wording\_index," that is the vocabulary. And we're going to print the sequences that we have generated. So let's quickly do that. So you can see in the output of the cell, we have word index, which is the dictionary, the word encoding dictionary. All of the words have been encoded. "It," one, "Is" is encoding as two, "a" is encoded as three, so on and so forth up till "today," which is encoded as nine. And we have sequence of words. So training sentences list had three sentences and we can see three sequences over here. One, two, three, five, four; one, two, three, six, four; and seven, one, eight, nine. So let's look at a sample sentence and its sequence. So we have, "it is a sunny day," which is encoded as "one, two, three, five, four," and you can match all of these encodings with the word index dictionary. All of the words have been encoding, referring to each of those keys defined in the dictionary. Now, the next step is to use this tokenizer on new sentences. So we have defined new sentences, which are containing new words as well. Will it be raining today? So "be" and "raining" are new here. "It is a pleasant day," where "pleasant" is a new word. So let's see how the tokenizer that we have trained on the training sentences operate on these new sentences. So to create these sequences, again, we use the textual sequences matter and we pass the new sentences. So we are creating the sequences of the new sentences that we have defined. Let's quickly run that. Now, if we print the new sentences and the new sequences, so you see, we have five words in the first sentence, whereas we have a sequence of only three values, which is, again... So basically it is not able to find out, the encoding for "raining" or the encoding for "be" from the tokenizer, which has been trained on the training sentences. And again, "it is a pleasant day." So that also contains five words, whereas the sequence of it contains only four. So that is a problem we see. Now, how do we handle that? So here, we can define the tokenizer. So while instantiating the tokenizer, we can pass this out of vocabulary parameter, which is called "oov\_token," and we can define the encoding for all the words that are not available inside our word encoding dictionary. So no matter how big your data is, how many words you are training on, there's always a chance that you will encounter a new word. And for that, we have this out of vocabulary token. So now, let's quickly train this new tokenizer that we have defined over here with the oov\_token on the training sentences. The training sentences as defined above these three sentences. And then we are going to create the word encoding dictionary. Now, if we use this same tokenizer to create a new text to sequence, basically using the text to sequence method on the new sentences, let's see what it generates. So you see, first, in the word encoding dictionary, now we have out of vocabulary token, which is given an encoding of one. And if you see the encoding of the new sentences where we had five words in both of the sentences, now we have encoded the "be" and the "raining" as one, and we can also actually look at it by printing the new sentences dictionary. So "will it be raining beginning today?" Now, it contains five encodings, basically five values, in its sequence. And the two values which are encoded as one are basically out of vocabulary token because "be" and "raining" are not available in our dictionary. And same goes for the next sentence. "Pleasant" is not available in the word encoding dictionary and thus, it is encoded as one, which is the code for out of vocabulary token. So this is how we can create sequences. Now, there are different manipulations that we are going to add so that all of the sequences are of the same length, so that the deep learning model, or the neural network, can actually process. And we are going to look at how we can manipulate those sequences, manipulate those lengths in the upcoming video.

### **Padding the sequences**

Selecting transcript lines in this section will navigate to timestamp in the video

- So far we have seen how to create sequences of sentences in the actual data. But these sequences can be of different length, which is a problem for deep learning models. We need to have a consistent input size for the model. And we can achieve that using padding. Now, TensorFlow, our you can say the Keras API actually offers padding functions that does all the heavy lifting for us. So again, here from Keras preprssessing sequence module we are going to import pad\_sequences function. So this is the one that we're going to be using. So we have imported it here in the fourth cell. All right. Now the next step is to define the training sentences on which we are going to train our tokenizer. So first of all, this is a python list, train sentences, I've defined four sentences of different lengths. So 'it will rain', the first sentence contains three words. 'The Weather is cloudy' contains four words, and we have 'Will it be raining today?', which has five words. And then last one ' It is a super hot day. This contains six words. All right, now the next step is again as we have seen previously we need to instantiate our tokenizer with our out-of-vocabulary encoding. So past the parameter 'oov\_token' So once that is defined, now the next step is to train this tokenizer using the fit on text method, passing the training sentences, and the last step is to store the word index. The word encoding dictionary that would be generated using the word and the score index attribute. So let's quickly run that as well. Now come the next step, which is creating the sequences of all the trainings sentences. So for that, we have seen, we can use the text to sequences method and pass the training sentences. Okay. We have, the sequences generated as well. Now comes the next very important step. So we are going to have sequences of different length, but we can make them of equal length using padding. And for that, we are using 'pad\_sequences'. The function that we have imported above, to this function we are going to pass the sequences that we have generated right above. All right. So once we run this, we would have the padded sequences stored in this variable pattern. And the (indistinct) sequences, let's quickly print all of these things. So first of all, let's print the word encoding dictionary. 'word\_index' The next step is to store trainings sentences. Let's print all the training sentences just for our convenience. Then we can print the sequences and we'll compare it with the padded sequences that we have generated using the pad sequences function. Let's quickly run the cell as well. So you can see, we have the dictionary at the top. In the first line, all of the words have been encoded. The second line contains all the sentences that have been defined in the training sentences. So 'it will rain.', 'The weather is cloudy.', all of those sentences are there. And then we have the sequences of all of those training sentences. 2, 3, 5. 6, 7, 4, 8. So we have 3, 4, 5, and then 6 values based on the length of each of those sentences. So when we have padded these sequences, you see we have zeros. If you look at the first sentences padded sequence we have 3 zeroes that have been placed before 2, 3, 5. So we are actually looking at pre-padding. So we have inserted zeros just to make the length of each of these sequences equal. And then accordingly in the second sentence we just had the four words. So we had to put 2 zeros. And then in the third sentence, we had five, so just one, zero, and the last, which was 'it is a super hot day.', last sentence, it had six values in its sequence. So that was all, okay, didn't need any pad sequences. But now all of the sequences are of equal length, which is six. Okay. Now, if, let's say we want to have a max length defined. We need to customize our padded sequence or let's say, instead of pre-padding we want to add these zeros at the end. We can actually achieve that as well. We can customize our padded sequences using some parameters. So here I have defined pad sequences function passing the sequences that we have generated about. Then I am passing this padding parameter which would basically contain the string, which will define do you want a pre-padding or a post-padding? So let's say, if you want a post-padding, you can type it like this. You can type the max length of your sequence. So here it has taken six as the max length without any max length parameter. But if we define, let's say, 5. So this is going to be our max length for our pattern sequence. So if there is any sentence that contains more than five words, it would be truncated. So do you want to truncate the word from the starting of the sentence or from the ending of the sentence that you can define here? So we want words to be truncated post, basically from the end. So you can define that as well using the truncating parameter. Now, if we run this function we have created the padded sequences using this function using these parameters. Or now, if we look at the padded sequence, you see all of the sequences are all of the length five. And you see the last sentence which had 2, 4 12, 13, 14, 15, as you can see over here. Now the 15 has been truncated from the end because we are using post truncating. And all the zeros have been added at the end because we were using post padding. So this is how we can customize the padded sequences. And now we are at a stage where we can actually start working or passing these trained data this pre-process data to our model. So we are going to work on a real world data set and perform all of these operations in the next video.

### **Challenge: Recognizing sarcasm in the text**

Selecting transcript lines in this section will navigate to timestamp in the video

(upbeat music) - So far we have seen all the pre-processing methods. We have seen how we can use tokenizer to create word encodings. We've seen how we can create sequences of the sentences. And we have seen how we can pad those sequences to manipulate their length. Now, in this particular challenge, you have to use all of these pre-processing methods to pre-process real-world dataset taken from Kaggle. So this dataset basically contains news headlines taken from different websites. So let's quickly download this dataset. I have hosted this file on Google Cloud Storage after correcting a few things inside the dataset so you can download it. And then I am reading it using pandas library using 3JSON function. So you see, you have a sercasting column, you have headline column where all the headlines are present and we are simply going to focus on this particular column for now and are also given the article link from where the headline comes. The next step is the important step where we are going to segregate all the headlines. So you see I've created a headlines variable and I've also stored labels as well but we are simply going to use the headlines list for now. So all of my headlines, all of the sentences in the headlines is actually stored in the headlines variable. Now, after that you have to import all the APIs. Once that is done. Now you have to follow what you have learned so far in the previous videos, define your tokenizer. Train your tokenizer, print the word index dictionary. So you have to fill up all of these blanks. Then you have to create padded sequences. So first grade the sequence then pad them and print those padded sequences sample. So this is a challenge for you and I'll see you in the solution video.

### **Solution: Recognizing sarcasm in the text**

Selecting transcript lines in this section will navigate to timestamp in the video

(upbeat music) - So here is how we are going to solve this challenge. First of all, you have to download the data, read the data using pandas, just as we did. And then we have segregated all of our training data in the variable call headlines. So all of the headlines are actually stored in this list called headlines. That's what we are concerned about. And now the next step is to import all the APIs. So, run that. After that we have to instantiate our tokenizer. So the out of vocabulary token, we have to add the token for that. So in coding, basically, so that is angle brackets and all rewritten. Now, in order to train the tokenizer, we are going to use the three-tone text method, and the training data is basically all the sentences. Basically headlines in this particular challenge, so that is there. Now in order to look at the word and coding dictionary or index. So you're going to use the word underscore index attribute. So print that and it's going to print the entire vocabulary, the dictionary for all the words in the headlines and then the next step is to create sequences first. So let's create the sequences, and the function for that is text to sequences and we have to pass all the headlines to this. So our sequences are created. The next step is to create padded sequences. So we have to pad those sequences that we have created. So for that, we're going to use the pad sequences function plus the sequences that we have just created. And since we are required to do post padding, so add padding type as post. I run this. And the last step is to print the padded sequence. So padded sequence, let's say for the first sentence. So you see, we have a post padded sequence of the first headline present in the data. You can also print the headline and look at how this is done, and you can also print the word index. So that's how you're going to pre-process a real world data set, which is then ready to go, into your deep learning model.

## **Question 1 of 4**

What is the purpose of padding the generated sequences?

* to add patterns to the sequences
* to create word encodings
* to make all sequences in a batch fit a given standard length  
  Correct  
  Correct, padding comes from the need to encode sequence data into contiguous batches.
* to add zeros for processing text

## **Question 2 of 4**

Choose the correct code snippet to train convert text(sentences) into a sequence of tokens using a tokenizer.

* tokenizer.word\_index(sentences)
* tokenizer.generate\_sequence(sentences)
* tokenizer.texts\_to\_sequences(sentences)  
  Correct  
  text\_to\_sequences() is the correct tokenizer method that converts text to sequences.
* tokenizer.fit\_on\_texts(sentences)

## **Question 3 of 4**

Consider the sentence below. Which of the following is the correct word encoding dictionary generated by the Tokenizer class?

"I am craving for ice cream!"

* {'I': 1, 'am': 2, 'craving': 3, 'for': 4, 'ice': 5, 'cream!': 6}  
  Incorrect  
  Correct encoding is a dictionary with key-value pairs after getting rid of all punctuations.
* {'I': 1, 'am': 2, 'craving': 3, 'for': 4, 'ice': 5, 'cream': 6}  
  Correct  
  Correct encoding is a dictionary with key-value pairs after getting rid of all punctuations.
* ['I', 'am', 'craving', 'for', 'ice', 'cream']
* {'I': 1, 'am': 2, 'for': 3, 'ice': 4, 'cream': 5}  
  Incorrect  
  Correct encoding is a dictionary with key-value pairs after getting rid of all punctuations.

## **Question 4 of 4**

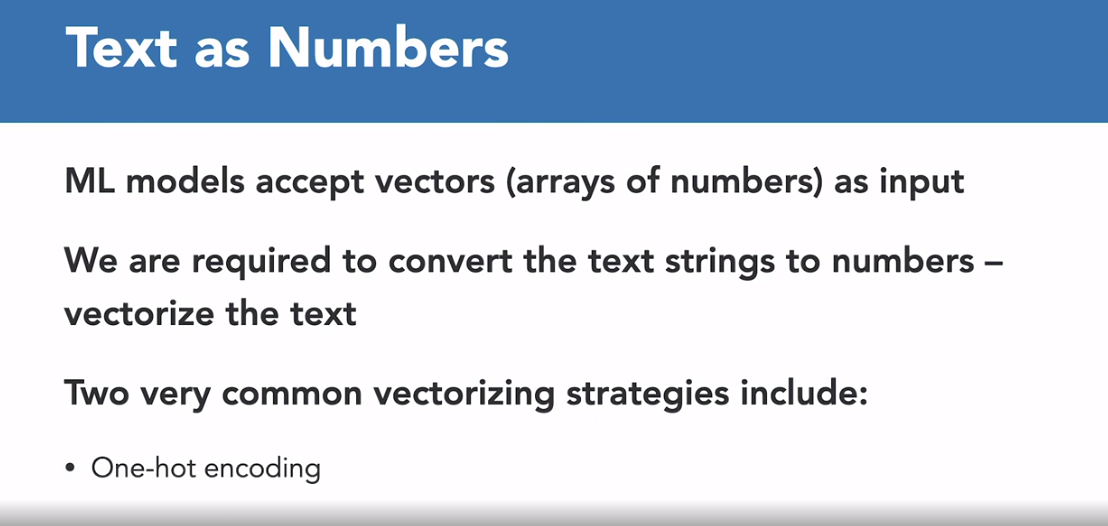
Which of the following is NOT a method of processing textual data?

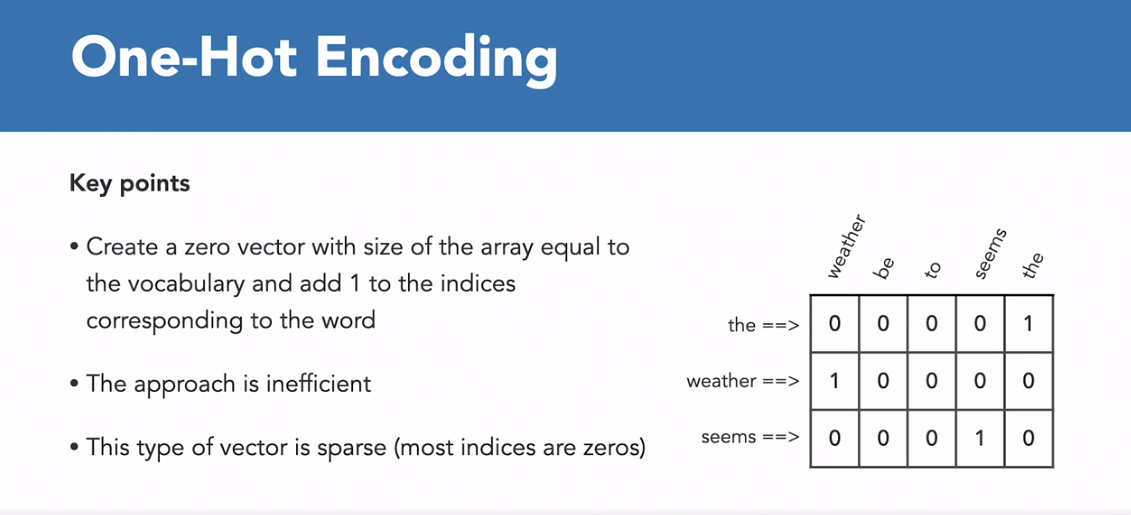
* text segmentation
* parsing
* edge detection  
  Correct  
  Yes, edge detection is an image processing algorithm and not involved in NLP.
* stemming

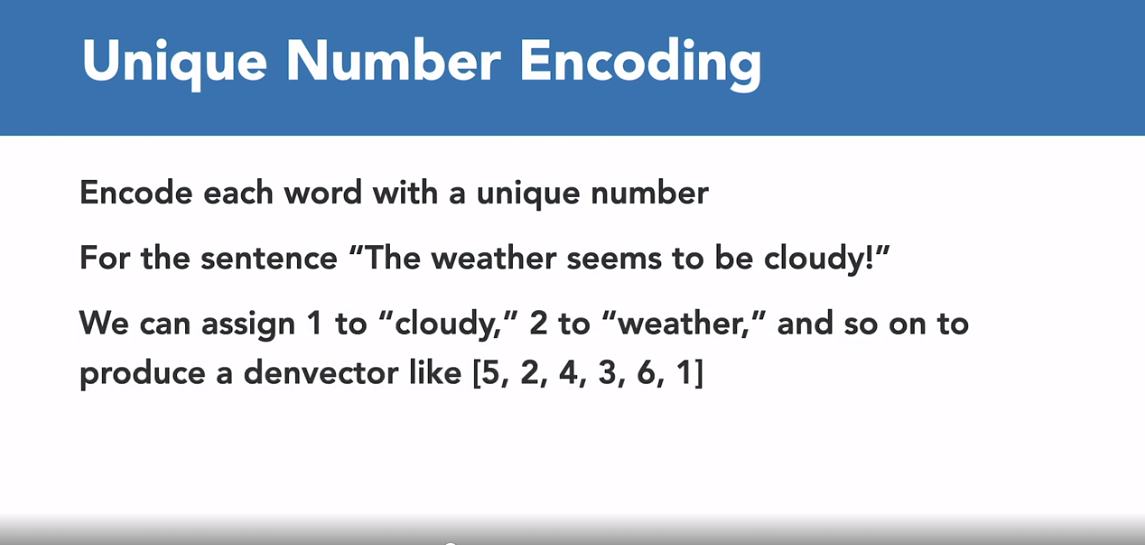
### **Introduction to word embeddings**

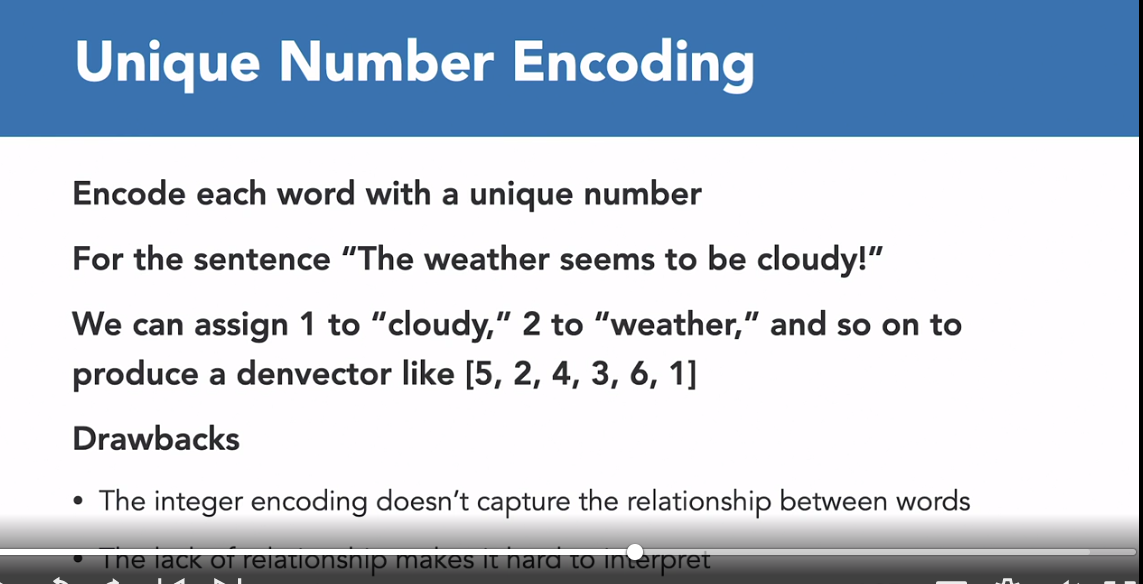
Selecting transcript lines in this section will navigate to timestamp in the video

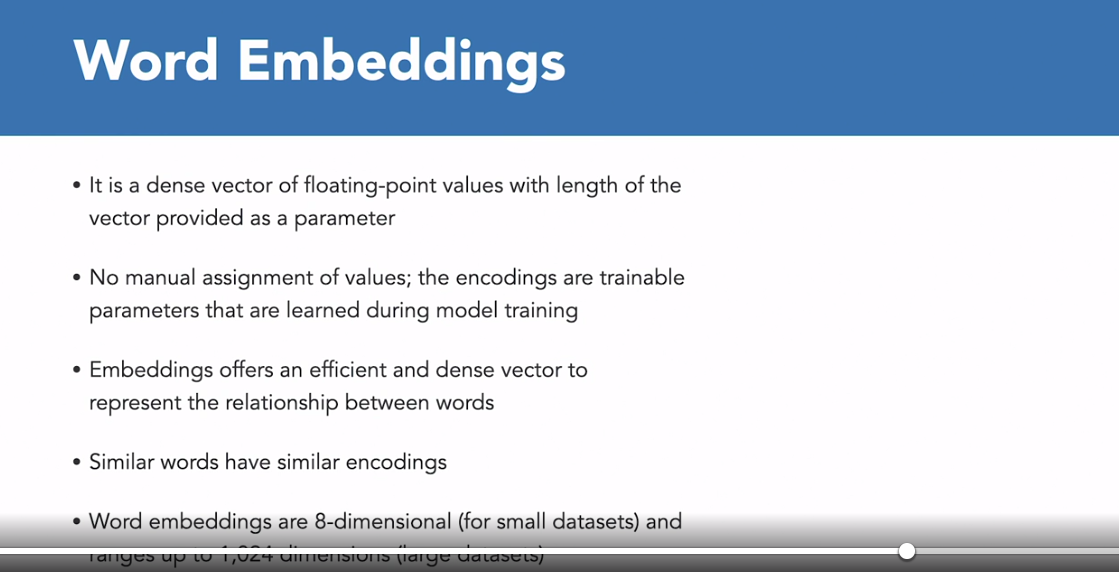
- [Instructor] Machine Learning models take vectors or you can say arrays of numbers as input. When you're working with text, the first thing that you must do is come up with a strategy to convert strings to numbers. Or to vectorize your text before you feed it to the model. There are two common vectorizing strategies: One-hot encoding and Unique number encoding. So as a first idea, you might one-hot encode each word in your vocabulary. So consider the sentence, "The weather seems to be cloudy." Now, the vocabulary or unique words in the sentence are weather, be, to, seem, cloudy, and the. To represent each word, you will create a zero vector with length equal to the vocabulary, and then place a one in the index that corresponds to the word, as you can see in this diagram. So wherever you see the in the sentence corresponding to that index you place one. Similarly for weather, similarly for seems, and so on. The approach is inefficient. A one-hot encoded vector is sparse, meaning that most of the indices are zero as you can see. Imagine you have 10,000 words in the vocabulary. To one-hot encode each word, you would create a vector where 99.99% of the elements are zero. Now a second approach you might try is to encode each word using a unique number. Now, continuing on the same example, you could assign 1 to cloudy, 2 to weather and so on. So you could then encode the entire sentence, "The weather seems to be cloudy," as a dense vector like [5,2,4,3,6,1]. Based on whatever values that you assigned to each word. Now this approach is efficient. Instead of a sparse vector, you now have a dense vector where all the elements are full. But there are two drawbacks, two downsides to this approach. First one is the integer encoding is totally arbitrary, it does not capture any relationship between the words. Second, an integer encoding can be challenging for the model to interpret. So a linear classifier, for example, learns a single way for each feature. Because there is no relationship between the similarity of any two words and the similarity of their encodings, this feature with combination is not meaningful. Now word embeddings give us a way to use an efficient dense representation in which similar words have a similar encoding. And importantly, you do not have to specify the encoding by hand. An embedding is a dense vector of floating-point values where the length of the vector is a parameter that you specify. Now, instead of specifying values for the embedding manually, they are trainable parameters, or you can say weights learned by the model during training in the same way a model learns weights for a dense layer. So it is common to see word embeddings that are 8-dimensional if you're working with small datasets up to 1,024 dimensions when you're working with large datasets. So a higher dimensional embedding can capture fine-grain relationship between words. But it takes more data to learn. Each word is represented as a 4-dimensional vector of floating point values as you can see on the right. For the example that we have, "The weather seems to be cloudy." So another way to think of it is as an embedding that it can be looked up, or you can say it is a lookup table. After these weights have been learned, you can encode each word by looking up this dense vector that it corresponds to in the table. Now to implement this, Geras API again provides an input embedding layer that we can add to our neural network. And we'll look at that in the next lesson.











### **Classifying movie reviews using TensorFlow**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] We understand that embeddings help us establish the relationship between words using vectors in multiple dimensions. Now, in this video we're going to learn how to add this embedding layer as an input layer to our neural network to learn about the sentiment of movie reviews. We're going to leverage TensorFlow dataset package to download a curated IMDb movie reviews dataset for this particular task. Now the same dataset will be used in the upcoming lessons to classify sentiment as well. So first of all, let's make sure that we install all of the required packages. So Google Colaboratory already has most of the packages installed, and if there is no package installed then, you can simply write pip install, and the name of the package to actually install it, so I've import a numpy here as np. TensorFlow as tf. TensorFlow dataset which is the package that we're going to use to download our dataset, and abbreviated it as tfds as you can see, then we are importing Tokenizer class, and the pad sequences as function. So one important thing to note here is that make sure that you are using TensorFlow 2.x, so some version of TensorFlow 2. So quickly run this to check the TensorFlow version. We're using 2.4.1. We are good to go. Now, the next step is to download the dataset, so the one that we are going to use is the IMDb underscore reviews, so make sure you first use the load function, so is the one that we are going to use to download our dataset. The identifier for our dataset is IMDb underscore reviews, so TensorFlow ds, our TensorFlow datasets package has a lot of datasets inside it, so the one that we're using is this one with info equals true means that I want to download the metadata as well, and as supervised equals true, that parameters is used to have labels downloaded as well. So I've stored the data, and the info, and the two variables let's quickly run this. So this takes a few seconds download all of the data, and the data is actually, we have both the training set, as well as, the testing set. Now, the next task is to actually segregate the training and the testing set, so that we can actually pass it to the model, and do both the training, as well as, the validation. So I have segregated it into two variables; train data, and test data. Now both of these dataset currently have both sentences, and the reviews basically, and the labels as well. So labels here would be some sort of sentiment, positive, negative, all right. Now create empty lists to store sentences and labels, so training sentences Python list, and then training and testing labels. So we are just defining the empty lists, and then we are going to iterate over the training data and the testing data to extract the sentences and the labeled. So here we are writing a loop to run over the train data, and simply we do that for our test data as well. So if you see here in this particular line I'm appending all of the training sentences, so I have written loop which will go with each sentence, convert it into a numpy array, decode it, and then convert it into a string and added to the list. So this would basically get my corpus ready, and then I am storing the labeled here, in a train underscore labels function, so this is both Python list. So let's quickly run this, doing the same process for both training data, as well as, testing data. This takes a few seconds. Now the next step is to convert these lists into numpy arrays. So simply write numpy, np.array function to convert this list into numpy array, run it, so we have the labels converted into numpy arrays as well. Now, a very important thing is to set up all of the parameters, define all the parameters that are going to be used in the Tokenizer and in padding, and while adding our embedding layer as well. So here vocab size would be, let's say 10,000. You can choose your own version, or your own vocabulary size. So embedding dimension, let's say 16. Max length, set it to maybe 150, and then truncation type, let's say we want to post, and similarly oov token, so that is angled brackets with oov. We define all of the parameters here, so that if we want to make or manipulate any of the parameters we can just come here in one place and do it from here itself. So the parameters are define, so let's quickly instantiate our Tokenizer as we did earlier. So num word is basically our vocub size, oov token our vocabulary token is oov underscore tok, and then tokenizer.fit, so we train the Tokenizer on the training sentences. So train underscores sentences. Similarly, we create the sequences on the train underscore sentences. All of this is following what we have learned in the previous chapter. Then once we have created the sequences, we pass it to the pad sequence function, and it will basically pad all of these sequences, pass the maximum and the truncating type, and then we do the same for our testing sequences as well. Quickly run it. Written small, very short function that'll actually help us understand what is actually happening. So we look at one sentence, how the padded sequence looks, and then we are going to learn what would the decoded sequence would actually turn into, so let's quickly print all of the three. So you see here's the original sentence, and then this is the padded sequence for it, and this is what the sentence actually looks like. So this is basically what's happening behind our testing sequence, and padded sequences that we have generated. Now, this is what we are going to pass on to our model as well. So let's define our neural network now which is the important part. So we are using the sequential API, then we're going to add an embedding input layer. So as the neural network trains, it can then learn these vectors, associating them with the labels to come up with what is called an embedding, so the vectors for each word with their associated sentiment. So the result of the embedding layer will be a 2D array with the length of the sentence, and the embedding dimension. So for example, we've used 16 here as its size, and then we flattened it out in much the same way as we required to flatten our images. So let's quickly first add our embedding layer in the sequential model. And we have passed the vocabulary size which is 10,000 we have set, and then the embedding dimension which is 16, and then the input length is the max length. Then we pass on to flatten font, flatten layer, and this will flatten it, and basically, convert it from in a ID tensile and then, tf.keras.layers, you pass this to the dense layers for classification, and these dense layers are basically added with activation towards a non-linearity to the model. And the in-depth analysis is basically covered in the intro to deep learning model, so it's beyond the scope of this particular course, so you can learn more about it in the deep learning course. So model.compile, then we have to compile our model by passing the loss function, the optimizer, and the metrics. Metric, we are using accuracy. Optimizer is using the Adam optimizer. You can learn more about it from the documentation, and loss is we are using binary cross entrepreneur since we are reclassifying. So modal.summary, let's look at the summary of this model that we have defined. We have total parameters, 174,413, and then we have embedding layer, flatten layer, and two dense layers. Great. Now comes the training part, so we're going to train it for 10 a box. So first of all, let's add the padded training sequences, the labels, and the validation data as well, so that would be our test underscore padded, and test labels. Let's quickly run it. This'll take a few seconds to train. So we see that our model is now trained. We have a training accuracy of one which is 100%, and then validation accuracy is close to 0.85, so 85%, so that's pretty good. But one thing that you can also do is you can replace this flatten layer with a global average pooling, a 1D layer, so that is what we use with most of the actual data. So once the model is trained, we can actually extract the weights that are learned in the embedding layer. So in the next video, we are also going to look at how we can visualize the vectors that the model has learned. So here isolating the first embedding layer, so moral.layers, so the zero index is basically the embedding layer, and then you can learn about get underscore weights function to derive the weights, and we are pending the weights, shape which is the vocabulary size and the embedding dimension, and then the weights that are learned by the model in the embedding layer as well. So you see this is how we can actually add an amending layer. And in the next video, we are going to use these and we'll try to generate some vectors, and some metadata, and see how those can actually help us understand how the words are related to each other using a projector.

### **Projecting vectors using TensorFlow**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] So after extracting the learn weights from them embedding layer, it's time to write the vectors and the metadata into files so that we can visualize these vectors in a 3D space on the TensorFlow projector. So the process is very simple, we are using the same collaboratory notebook that we used in the previous lesson. So first of all, we have imported all the libraries, APIs using the same dataset. IMDB reviews, segregated, the training and testing set prepared the data, then trained our model with the same configuration embedding layer, the first layer, the flatten, and then too dense layer. We've created our model, then the next step is to actually extract the weight. So we have first isolated our embedding layer. Then we extracted the weights from the gate\_weights function. And then we looked at the weight shape which is basically vocabulary size. That is we have 10,000 different words in 16 different dimensions. So embedding dimensions. So basically for each word we have schools in 16 different dimensions. So based on that, we can actually visualize it in the 3D space. And so let's quickly run that, extract all the weights. Now comes the important part. So we need to write all of these vectors into files so that we can actually upload those into the projector. So for that we're using the Import I/O module from Python. So first of all, we open up the text stream. So we are writing two different files both the vectors file and the metadata file. So first of all, open up the stream for vectors. Tab tsv which is a tab separated value file. And we open it up in the right mode with encoding 'utf-8.' Similarly, we do it for the text stream for metadata the metadata file meta.tsv in the same configuration, in the same mode. Then what we need to do is we need to write each word with its corresponding embedding. So the word goes into the metadata file and the embeddings or the vector which goes into the vectors file. So here for index and range so for all the words, basically for, let's say we need to run the slope 10,000 times which is the vocabulary size. Then we need to get the word corresponding to that particular index and we have created this reverse\_word\_index dictionary above. So we have basically flipped the word encoding's key value pairs. So earlier we had keys as word and values as encodings. So here we have flipped that so that we can actually get the word corresponding to that particular index. And similarly, we get the embedding score for that particular word, and let's quickly write the word in the metadata and in the vectors we need to write the embedding scores. So for all of the scores in the embeddings vector we need to write it give a creator tab, separated value, and then right into the vectors file, close the stream, quickly run it. Once you've run it, you have to download these files from collaboratory to your local machine. So this is for that. So they'll take a few seconds and download the vectors.tsv and meta.tsv once that is done, let's go to TensorFlow projector. So here, what we are going to do is we can now upload all of our data. So let's quickly click on load. So you have to first upload your tsv file, the vectors file. So vectors.tsv and then you have to upload the metadata, meta.tsv. So both of the files have been uploaded, click on this spherize data checkbox, and now you can actually visualize. So this is the entire Corpus that you have all of the words and the 3D space. And, so basically what you can do is you can actually look at all of these words and see how they have been clustered together. So if we had let's say, fair hair. And then we have Chad, we have A, so corresponding to a particular direction you'll see common words, words that have some close relationship you can also search for different words and see how the words have been clustered how the model has performed. So this is how you can project, now in the next video we are going to learn more about classifying the entire texts.

### **Building a text classifier**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] Now that we have learned all about pre-processing the textual data, how to add embedding layer, create a neural network, and how to visualize those embeddings, now it's time to actually get down to business and classify some textual data. Here in this particular video, we are going to use the same news headlines data that we talked about in the first chapter. First of all, again we're going to import all the required packages, the function, the classes. So, numpy, tensorflow, tokenizor, padding, pad sequences. Once that is done, we are going to again download the news headlines data from the cloud storage link that I provided. Simply the standards, and you'll have your data downloaded. And I am reading this downloaded data using pandas package. This would give us a data frame, which is stored in the data variable, as you can see over here. We'll simply print the first five rules. Let's quickly run this. You see, we have three columns in this data frame. It's kind of tabular structure. Is sarcastic, which is our label column. And then we have a headline column where all of the textual data that all of the headlines are actually present, which we are going to focus on and which we are going to actually classify as sarcastic or non-sarcastic. That's basically the ultimate goal of this particular video. And then we have article link, but we don't really need this particular column or the links. We're not using them at least in this particular video. Let's store the headlines and the labels. We are going to segregate the labels and the labels variable in the labels list. And we're going to create a separate headlines list. That is very simple. We can simply use the square brackets and use that notation. Se have headlines list now, and all of the labels are in the labels list. Now comes the next part. We are going to set the parameters. So heavy, I want to define the vocabulary size or max length as we did in the previous lesson. Let's say vocabulary size is 10,000 and then max length is, let's say, 120. Embedding dimension is 16. We are going use the post trunc type or sparing type. OOC token, out of vocabulary token is set. Training size. Now, training size is something that you can decide, mostly we have a 20% of the data reserved for testing, so you can define, let's say, out of, let's first quickly look at the number of rules that we have in this dataset. Data.info. You see we have 28,000 rows in this particular data. Let's say we have, we reserve 20,000 rules for our training purposes, and the rest 8,000 something for our testing purposes. This is how we are setting these parameters so that we can actually split the data. Here in the next segment, we'll see that we are actually splitting the training and testing set using slicing. First for the training purposes, we can go from zero to the training size, and for testing sentences, we can go from the training size to the end of the entity. That's how we're going to extract our sentences first and do use the same code, the same slicing, for the labels. That'll basically have everything in sync. Everything would be perfectly matched. Once that is done, we will have actually split the training and testing set now. Once that is done, we are going to pre-process the sentences. Again, we are going to use the tokenizer, train the tokenizer on the training sentences, and then create a word index. Then coding dictionary is ready. Create the sequences for both the training sentences and for the testing sentences using text to sequence matter, and then pad those sequences using the batch sequences function and the padding type max length, truncating all of the parameters are going to go from the definition that we have defined here. All right, so let's quickly pre-process those sentences, run it. And now we are working with Python lists. We have to convert those into non-binary so that we can make it work with TensorFlow 2.X. The model at stake now by areas rather than lists. Quickly convert all of the padded sequences and labels into nompy areas. Once that part is done, now comes the model definition. Here, what we're going to do is simply define our model. First of all, we need to add an embedding layer. Once that is done, the embedding layer accepts the vocabulary size, the embedding dimension input length. Input length is basically the max length that we have defined. And then we are going to add a global average pooling 1D, so you can also use flatten. You can also test it with different other layers. Feel free to play with the parameters, the hyper-parameters. Again, we're going to add dense layer to classify. The last output layer is going to give us the probability as in zero or one, which class is it sarcastic or is it non-sarcastic, any particular sentences? That's it. Activation relu. And the final one is the signoid, because we are using classification, and then we are going to compile where the loss function binary cross entropy and optimizer is Adam. And the metric that we have chosen for this whole model is accuracy for now. Let's quickly define this model, look at the summary of the model. You see, we have 160,433 parameters to train in this model. We have four layers, and let's quickly train the model. We have 30 epochs. This is going to take awhile. We have passed the padded training sequences, the labeled, the training labels, epochs is number of epochs, which we have set to 30, and then validation data, which contained testing padded sequences and testing labels. Let's quickly run this. This is going to take awhile. Here, our model has completed training. We have a training accuracy of 99.25%, and then we have a validation accuracy of 80.67%. that's decent. Right here, we have stored our training into this history variable so that we can actually plot whatever we have learned, all of the scores that we have actually gathered. Here, this is the code snippet to actually visualize the validation score, the training score for loss as well as accuracy. Here I have written this plot graphs function where I'm accepting the history of our model training and the string, which basically can be either accuracy, laws, or any other metric that you have chosen. Here, while training, we have chosen the accuracy metric and our model compilation, model configuration. You're going to use that and the loss function. Simply run this, and we'll basically be able to compare how the training accuracy has increased, and where the validation accuracy is. We see there's quite a bit of gap, so that kind of shows that there's a little bit of all fitting as the training accuracy is quite high, but the validation accuracy is not quite there yet. And then we can also compare the loss like this. Now, let's quickly classify a new sentence based on this particular model. I created a new list of sentences. I've added two sentences here. First one is the baby boy fears spiders in the garden might be real. And then second sentence is Game of Thrones season finale is showing this Sunday night. We need to find out if these are sarcastic or not. The first step is to create sequences. Use text to sequences, and then pass the sentences. And then you need to actually pad these sequences as well. for that, you're going to use the pad\_sequences function, and then pass the sequences that you have created above. Once that is done, we are going to simply call the model.predict and add these padded sequences to the model. Let's quickly run this. Here we see both of these are very close to zero. This is my 10 restrict minus five, so that is very close to 0.000 something. And both of these do not contain any sarcasm. That's close to zero. And that's how this model is actually classifying headlines based on if that particular sentence or if that particular headline is sarcastic or not. We've seen how we can do text classification using deep learning, using these embedding layers. Now we move on to a challenge in the next particular video.

### **Challenge: Text classification**

Selecting transcript lines in this section will navigate to timestamp in the video

(upbeat music) - [Instructor] So up until now, we have seen how to classify text using a deep learning model with embedding layer, dense layer, and the flatten layer. Well, here is a challenge for you where you are going to use IMDB movie reviews dataset again, and then classify the reviews as positive or negative. Positive is basically one and negative is zero. So here in this challenge notebook you have to fill up all of these blanks. First of all, download the data. How are you going to download the data? From TensorFlow dataset package. Then segregate the training and testing sets and the pre-process, prepare the data using Tokenizer, and then define your Neural Network with an embedding layer in it, and all the other layers that you would want to put in based on the configuration that you want to set for your model. You can play around with the hyper parameters, and then train your model, provide the training sequences, the validation set. And after that, you have to visualize the training and validation accuracy, and loss, so complete function over there. And at the end, you have to classify new reviews. So let's get down to completing this exercise, and put our knowledge to test.

### **Solution: Text classification**

Selecting transcript lines in this section will navigate to timestamp in the video

(upbeat music) - [Instructor] So let's quickly look at how we are going to solve this challenge. We're going to go through each line, each blank one by one. So firstly run the foresail import all of the packages, functions, and classes. Then we are using the IMDB reviews dataset from TensorFlow dataset package. So, download it, provide the identifier to download IMDB reviews dataset then with\_info is true and similarly as supervised is also true because we need the labels. So let's quickly download the dataset. Once that is done, you have to segregate the training and testing set. So the key is train and test as given in the documentation as well. So once that is done, I trade over the training set train data and the test data to get all of the training sentence and labels and testings sentences and labels. So segregate them. So these are patterned lists, convert them into number array. Then the next step is to define the parameters for tokenizing and padding. So vocabulary sizes, let's say 10,000, embedding dimension keep it to 16, max length keep it to 120. You can change the parameters if you want. And then while instantiating your tokenizer you have to pass the parameters number words would be equal to the vocabulary size and the token as we have defined above. And once that is done, tokenizer.fit on text. So you have to train your tokenizer on the training sentences, pass that, create the sequences of the trainings sentences and pad those sequences. So train\_sequences then do it for the testing sequences as well, run it. So while your tokenizer is creating those padding sequences Now let's create or define a model. So, first of all, in the sequential model we have to add an embedding layer. Now the embedding layer is going to take vocabulary size and the embedding dimension. Input\_length would be equal to max\_length that we have defined. Then we are going to add one globalAveragePooling1D The next layer would be the dense layer. You can change the number of nodes if you want six, 24, 32, whatever you want and then another output layer. And that will give you the final output. Find the probability. And then last function is binary\_crossentropy and optimizer as Adam. Let's look at the model summary, looks good. So before training, we have to pass the training sequences so that is training train\_padded pass the labels train\_labels And similarly do that for the testing dataset test\_padded test\_labels running it for 10 inbox. This is going to take a few seconds. So meanwhile, we'll complete our plot function. So here you see we are accepting history of the model training and the metric. So let's quickly add a metric overhead. So validation, accuracy, we'll run it for accuracy and loss. So that will be simply metric on one access. We'll run it for accuracy first, as you can see overhead and then we'll run it for loss. So the visualization function is also complete. So maybe waiting for the training to wrap up. So our training is complete. The training accuracy is 0.97% and the validation accuracy is 81.7%. So let's visualize the two. You can see there's slightly a gap between the training accuracy and the validation to the model is doing a decent job but it's a little offering as well. And then we have to classify the new reviews. So for that, we are going to first create the sequences. So it takes two sequences of the new sentence list that we have defined above. So once that is done pad those sequences, so that is done. And then these padded sequences should be first to the model.predict function. And for the two sentences you see for one we are getting 0.05. So that is close to zero. So that is a negative review the first part of the movie was dull and boring. Yes, it's negative. And for the next one we are getting 0.68 is close to 0.5 somewhere in the middle. But if you see this is a positive so our model is doing somewhat decent job. So that is basically how you are going to solve this. You can play around with a few parameters here and there and try to work on the accuracy of your model.

## **Question 1 of 4**

What is the drawback of one-hot encoding?

* It adds too many zeros.
* It oversimplifies the encodings.
* It is slow in encoding the categories.
* It is inefficient as it creates a huge sparse vector with a large feature space.  
  Correct  
  Curse of dimensionality is a drawback of one-hot encoding.

## **Question 2 of 4**

Which activation function would you use in the output layer of the network to classify a restaurant review as negative or positive?

* softmax
* softplus
* relu
* sigmoid  
  Correct  
  As sigmoid function returns value between 0 and 1. It is a suitable activation function for binary classification problems.

## **Question 3 of 4**

Choose the right mapping for the data that goes into vectors and metadata.

* Vectors - weights Metadata - words  
  Correct  
  Vectors contain the learned weights and metadata contains the words that are to be projected.
* Vectors - words Metadata - weights
* Vectors - weights Metadata - sequences
* Vectors - weights Metadata - vocabulary size

## **Question 4 of 4**

Which of the following is NOT a valid parameter to the Embedding layer?

* input\_length
* embedding\_dim
* input\_dim
* activation  
  Correct  
  Embedding layer doesn't accept any activation function.

### **Introduction to RNNs**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] So far we have done text classification by understanding the positive and negative sentiment in news headlines and movie reviews, used to pass the data and labels to a simple neural network to learn the function that classifies text as positive or negative. Now the model didn't do as well as one would have expected. The main reason for this was that the context of words was hard to follow when the words were broken down into sub words. So the sequence in which the tokens for the sub words appear become very important in understanding the meaning. In order to find the solution to this problem, let's first understand how sequential data influences the output. So here we have a factorial data. Where n factorial can be written as follows. You have n factorial which can be written as n into n minus one into n minus two, so on up till one. And similarly if you want to write n minus one factorial, you can write it as n minus one into n minus two, up till one. So you can say that n factorial can also be written as n into and minus one factorial. And similarly n minus one factorial can also be written as n minus one into n minus two factorial. So for example, we have one factorial over here, which is simply one. Then we have two factors which can be written as two into one factorial. Then we have three factorial which can be written as three into two factorial. Similarly for four factorial and five factorial. So you see all of the POS data points, the sequence is actually contributing to the output. So this is how basically a sequence, the POS sequence actually influences the whole output, the next prediction in RNN as well. So the neural network is like a function that when you feed in data and labels, it enforce the rule from these and then it gives you the output which is the rules. But this doesn't take any kind of sequence into account. This is similar to the basic idea of a recurrent neural network or RNN which is often drawn a little like this. You have your X as input and Y as output but there is also an element that is fed into the function from a previous function. So this becomes a little more clear when you chain them together like this. So have X0 which is fed into the function returning Y0 and output from the function is then fed into the next function which gets fed into the function along with X2 to get Y2 producing an output and continuing the sequence. As you can see, there is an element of X0 fed all the way through the network similar with X1 and X2 and so on. This forms the basis of the recurrent neural network or you can say RNN. So here's a sentence. We have a beautiful sunny dash. What do you think would come next? Probably day right. We have a beautiful sunny day. Now how did we know that? Well, there's a beautiful sunny in this sentence. In a context like this, it's quite likely that we are talking about a beautiful sunny something. We mean a beautiful sunny day. So the context word that helps us understand the next word is very close to the word that we are interested in. But what happens when the context word is away? So here's a sentence. I was born in China back in school. We only spoke one language and that was dash. Now how would you finish that sentence? Now you might say Chinese, but you'd be much more correct and accurate if you said I was born in China. Back in school, we only spoke one language and that was Mandarin. So first of course, it's the syntactical issue. Chinese describes the people, Mandarin describes the language. But in the ML context, the key word that gives us the details about the language, that is the word China which appears much earlier in the sentence. So if we are looking at a sequence of words, we might lose that context. So with that in mind and update to RNN is called LSTM or you can say Long Short-Term Memory has been created. So in addition to the context, being paused as it is an RNN, LSTMs have an additional pipeline of context called cell state. So this can pass through the network to impact it to actually influence it. This helps keep context from earlier tokens relevant in later ones. So issues like the one that we just discussed can be avoided. Cell states can also be bi-directional. So later context can impact earlier ones as well. And we'll see that when we look at the code. Now in this chapter, we will be focusing on improving the text classification with the implementation of LSTM.

### **Implementing LSTMs with TensorFlow**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] So we have understood the importance of sequence and providing context to classify texts as positive or negative, or you can say sarcastic or non-sarcastic. Now, in this particular video, we are going to implement the LSTM based model to classify our news headlines as sarcastic or not sarcastic. We're using the same dataset that we have used in the previous chapters. So here, first of all, let's import all of the packages that we are going to be using. So when NumPy, TensorFlow, tokenizer, or pad\_sequences make sure that you're using TensorFlow 2.x. So we are currently using 2.4.1. So quickly download your data and read it using the pandas package. So again, we have the same dataset headline column and is\_sarcastic column is the one that we are interested in, segregate create arrays for both your headlines and your labels. The next part, very important part is defining the parameters for tokenizing and padding. So vocab\_size would be, let's say, 10,000. Max\_length let's keep it to 120. Embedding dimension let's change it to 32. You can tweak this parameter as per your liking. Maybe make it 64 or whatever you want. Then training size, so we have 28,000 rows, so let's keep it to 20,000 for the training. Now, the next step is splitting the data set into training and testing set. So for training, let's go from zero to 20,000 which is our training size. So that is what we will add in the slicing. And similarly for the training labels again, we're basically going from zero to training size. So all of the rules from zero to 20,000 would be reserved for our training set. And similarly, from training size up until the end would be reserved for our testing purposes. So as agreed, we have split the dataset. Next step is to tokenize and create sequences for your training sequences. So let's quickly do that, pad those sequences. Once that is done, we have to make sure that all of our lists are converted into number arrays to make them work with TensorFlow 2.x. So let's do that as well. So once that is done, the data preparation step is done. Now comes the very important step. So we have to define a model. So the first layer would be the same that we have been using so far, embedding layer to give us the dense vector. Again, the parameters would remain the same, vocabulary size, the embedding dimension that we have said, and the input length, then comes the very important second layer. So here we are going to implement the LSTM layer. So using the same karas.layers module, and from here we are going to use LSTM with 64 as, I want 64 outputs from this LSTM layer. And since I want the output of this LSTM layer to go into another LSTM layer, I have to set this parameter called return\_sequences to "True." Well, this here makes sure that the output of one LSTM layer is compatible with the input size of the next LSTM layer. So the first layer, if within the two or within, if you're using, let's say four LSTM layers, so the first three should have return\_sequences to "True," and the last one does not need to have a return\_sequences parameter. So we are implementing two LSTMs layer. So, the second layer would have 32 output, and this does not need to have the return\_sequence parameter. And we want the cell state to go in both directions. So we are going to wrap this up with the bi-directional layer. So again, both of them, and then we are going to pass this output from these LSTM layers to the .Dense layer for classification, all right. So the compilation is the same. We're using binary\_crossentropy for loss, optimizer remains the same, metric as accuracy. Let's quickly define the model, look at the model summary. So here, if you see the model summary we have the embedding layer, which has given us 120 by 32, the output shape. The LSTM layer if you see, we have marked the 64 output but it is giving us 128. It is because we are wrapping it up with a bi-directional layer. So the cell state would go in both of the direction that's why output would be doubled. So that's where you see 128 and 64 in the next layer. So the total number of parameters to train 412,465. So that's there. We have set the number of epochs or 10. Model.fit, we're going to start training our model store it into history variable so that we can actually visualize them. So the model is going to take a few minutes. So hang in there. So we are done training and if you see we have a training accuracy of close to 100% and the validation accuracy if you see, it's close to 83% or 84%. Now let's quickly visualize the accuracy and the loss for training and validation. So here, if you see the accuracy curve, the training accuracy, the blue line is touching one whereas the orange line, the validation accuracy. So there's quite a bit of gap between the two which basically indicates that there's a bit of overfitting in this model. And whenever you using like many complex layers for a fairly simple problem, this kind of problems or another old fitting issues might come in. And similarly for the loss if you see, the training loss went down whereas the validation loss, it went up with some spikes as well. So let's see how the model performs when we are running it to classify new sentences. So we have defined two sentences. "The girl starting the first snakes "in the garden might be real," and "Game of Thrones season finale showing "this Sunday night." So if you see, we have both of them close to zero so that there's both of the sentences have been marked or classified as not sarcastic. Whereas the first one seems to be kind of sarcastic. So, that was how you can use bi-directional LSTM to a model to classify a news headlines. Now feel free to tweak all the hyper-parameters with different embedding dimensions or number of output, et cetera. And see if you can improve the validation accuracy. Now we'll look at how we can use convolution layer to improve the model in the next lesson.

### **Improving your movie review classifier**

Selecting transcript lines in this section will navigate to timestamp in the video

- So far, we have seen how the LSTM Layer or the LSTM Model has performed, in classifying text or classifying headlines as sarcastic and non-sarcastic. We've witnessed a bit of all fitting with the LSTM model. In this video we are going to try to improve the performance of the model using a Convolution Layer. So we've been using Convolution for image classification but we'll see how we can use the Convolution Layer for text classification purposes. Users using the same code, same packages. So using the same dataset as well, everything remains the same, embedding dimension, all of the parameters, sequences, number by areas. So let's get down to model definition. So here, we have a sequential model. So first of all, the first layer would remain the same, embedding layer, where the vocabulary size equals 100 embedding dimension and set 264 and input length max length that is also defined to be a hundred. Now we add a convolution layer here. So con one D so we specify the number of convolutions. So that is 64 and we specify the size of the convolution. So that is, let's say five and we set the activation function to realm. Now the effect of this will be that now the words will be grouped into the size of the filter which in this case is five and the convolution will be lower that can map the word classification to the desired output. So, let's add another layer. So after this we are going to add a global average pooling one D to down sample the feature map coming out of our convolution layer. And then we will pass the output to the dense layer for classification as we have been doing. So last function is the same, optimizer is the same. So let's look at them all at sampling and understand what's happening with the output shape. So embedding dimension, we had an input size the max land of hundred by 64. And then in the convolution, if you look at the output so if we have 96 and the number of convolutions is 64. So how did it go from a hundred to 96? So basically as the size of the input was a hundred words and we have a filter that is five words long. So that will basically print two word from the front and do word from the back, leaving us with only 96 words. So the 64 filters that we have specified will show up in the model. So total number of parameters to train, we have 662,000 parameters and we are going to train it for 10 a box providing all of the training sequences as well as the validation sequences. So this will take a few seconds make sure that you are connected to GPU runtime that you can check from the runtime task bar. So we are over training. So the model has done training and we have a training accuracy of again close to a hundred percent. And the validation accuracy is somewhere around 83%. So, it's exactly kind of similar to what we had with the LSTM model. Now again, you might want to try with different parameters, with different number of eBooks here and let's quickly visualize the accuracy, the training accuracy and the validation accuracy from the chart. So you see the loss. We do not have any spikes in the validation as we did in the LSTM. So here, the curves are a little smoother but there is still a gap between the accuracy and the loss. If you see the validation loss is higher. So that kind of indicates towards all fitting. And if we look at the classification of new sentences let's quickly run the model on the same new sentences. So here you see that we are getting close to zeros, the values. So this model is also classifying the two sentences as non sarcastic. So that's particularly how we can actually go on to implement different models to see which one performs better. And once you have actually shortlisted one model and then you can build a different hyper-parameters, do some hyper chronometer tuning and you can go ahead with the one that gives you the best validation accuracy. So this is all about text classification. And in the next video, we are going to pose a challenge, where you will be working on text classification problem and find out the best problem or the best model that would give you the best performance for that particular problem set.

### **Challenge: Yelp review classifier**

Selecting transcript lines in this section will navigate to timestamp in the video

(upbeat techno music) - [Instructor] So you've seen how an LSTM layer or a convolutional layer can be added to the model to classify text. Now, in this particular challenge, I want you to implement and explore LSTM and convolution model over this new Yelp review dataset from TensorFlow dataset package. Now, I have already provided the code snippet to download the Yelp popularity reviews. So accordingly, you will have both your training testing as well with the IMDb dataset from the TFDS package. So you have to fill up all of these blanks: vocabulary size, embedding dimension. Feel free to tweak your parameters, manipulate those numbers. And try a combination of them, different combinations. And then you have to define the model first. Try to work with an LSTM model and then do a CNN. So fill up all of these blanks and see how you get to classify those texts. And what sort of accuracy and loss you witness with the new models for the new dataset. And at the end, you have to classify some new reviews. So I've added these two sentences that you have to first create sequences. And then you have to pad those sequences and use the prediction function. So get on with the challenge. And I'll see you in the solution video.

### **Solution: Yelp review classifier**

Selecting transcript lines in this section will navigate to timestamp in the video

(upbeat music) - [Instructor] Let's look at the solution for the Yelp review classifier. One thing that you should keep in mind is that this data set, the Yelp polarity reviews from the TensorFlow dataset, this is a very large dataset and this might take a while for your model to train. Let's look at the solution for first of all we have downloaded the data to identify for this dataset as Yelp polarity reviews. As supervised=true, written for true, and then we have segregated the training and training sentences, training labels, test sentences, test labels, Then here is the configuration of the parameters, basically vocabulary size we have set to 10,000 and embedding dimension is 32. Max length, I've set it to 120, and I'm using padding type as post, and truncation type as well as that is also post. Now, moving on to the tokenize OP, everything is the same as we have been seeing. We have created the tokenizer, trained it, created the sequences, padded those sequences. Here comes the important part. Now, here, first of all, we have added an embedding layer. After that, we have added the bi-directional LSTM layer with 128 output, and then this LSTM output is going into another LSTM where the 64 output. You can see I've marked return sequences to true. Then we passing it to the dense layer where I have added 32 nodes, and then going at the output of that is going into the final output layer with the one output, which would be the classification, and the loss is binary crossentropy, optimizer, Adam, and here is the model summary. This is how basically you can define your model, and I have run it for 10 epochs, and it took me around 50 minutes to actually run this model because the data is so heavy. Make sure that you have enough time and you are currently running it on the GPU on Google collaboratory. After that, we can visualize these plots for accuracy and loss. The training accuracy, as you can see, was close to 100%, and the validation accuracy you see is still better. It is also close to 95%. That's a really good scale there, really good performance by the model. Now, the next step is to put our model to testing. For that, we have picked up these two random sentences. Now, the process is that we are required to create sequences out of these raw sentences, and then pad those sequences using the pad sequences function, pausing all the parameters that we have defined above, and then pass those padded sequences to the model.predict function. The predict method and the train model object is going to be invoked, and we are going to get the final scores. You see we have 0.64 and 0.57. So there's somewhere in the middle. The first sentence is on the positive side. That's how you could have solved the Yelp review challenge. And if we run the tool sentences, if you try to predict the classes of the new reviews. So let's say we have two sentences, the restaurant served a delicious pasta, and the restaurant didn't have a decent ambient. Here it is giving a score of 0.6 to the first sentence and a 0.5, so basically somewhere in the middle, it's keeping the two sentences. The first one is more on the positive side. So that is good.

## **Question 1 of 3**

Which of the following does NOT apply to an RNN?

* Encode internal states information about the timesteps it has seen so far.  
  Incorrect  
  RNNs maintains states in both directions b encoding the information about every timesteps it has seen so far.
* Reduce the size of the feature map.  
  Correct  
  A pooling layer reduced the size of the feature map and not an RNN or LSTM layer.
* Preserve states in both forward and backward direction.
* Can be used for time series problems.

## **Question 2 of 3**

What is right method of accessing the metrics after training a neural network?

* You can't access the training metrics
* Watch the training closely as it happens.
* Learn by running the model on new data.
* Use the history attribute of the model training history to access the metric.  
  Correct  
  Metrics can be accessed from the training history using the history attribute.

## **Question 3 of 3**

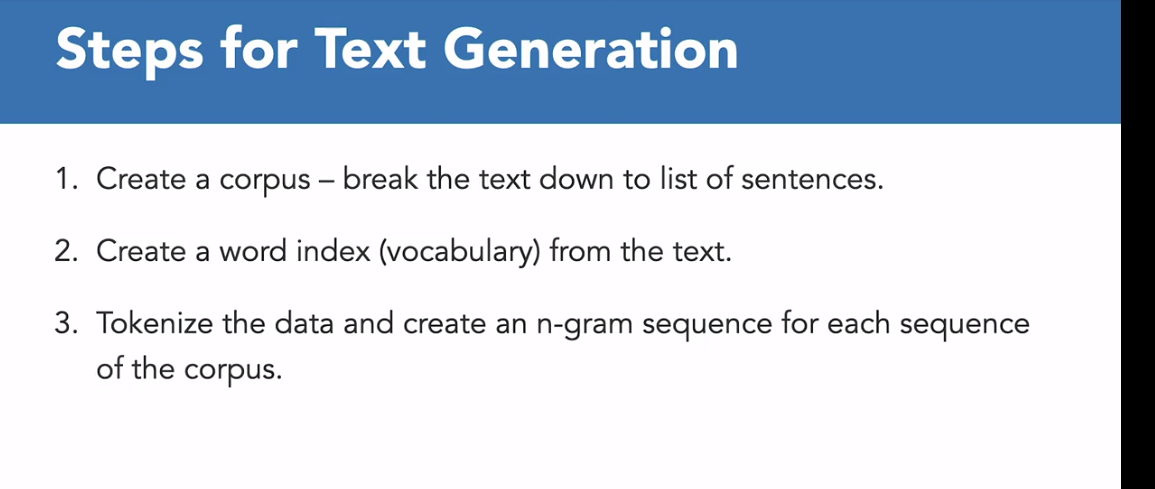
Which of the following arguments is required to add multiple bidirectional LSTM layers?

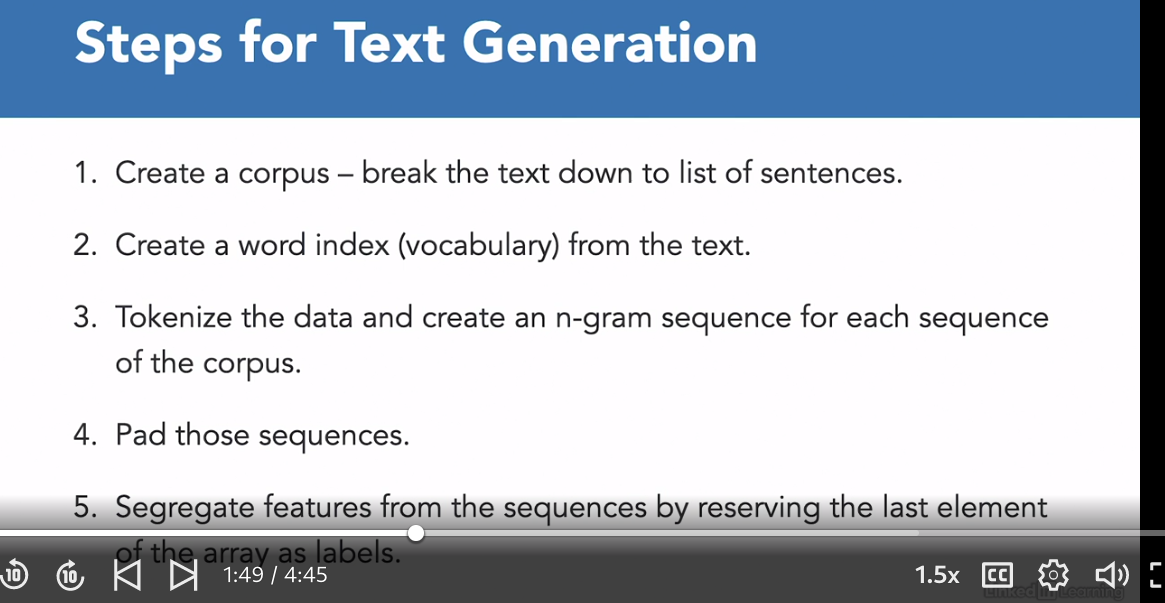
* input\_length
* activation
* return\_sequences  
  Correct  
  You need to provide return\_sequences=True to add multiple bidirectional layers in a neural network.
* embedding\_dim

### **Introduction to text generation**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] Do you know that we can get a body of text extract the full vocabulary from it and then create datasets from that. Where we make it phrase the training features which are called the X, and the next word in that phrase is called the Y, which is the label. Now up until now, we have learned about solving text classification problems with deep learning, but what about if you want to generate new text? Now this might sound like a complex and unknown directory but actually, we simply need to build on what we have learned so far. So in this chapter, we are going to focus on the text generation which is more of a prediction problem rather than a classification problem. So, how are we going to create the dataset, train the model, and how do we predict based on the C-Text that we provide and generate new texts? Let's look at the steps for that. So, here is a step-by-step guide to generate text. First of all, we create a new corpus, so whatever the sentence, or whatever phrase or huge body of text that you have taken, you have to break the text down to list of sentences. The second task is to create a word index that is the vocabulary, the word and coding dictionary from that particular text that we have been doing so far. The third step is Tokenize the data and create n-Gram sequences for each sequence of that particular corpus that you have created. The fourth step is to pad those n-Gram sequences. The fifth step, or the final step is to segregate features from the sequences by reserving the last element of the array as labels. Let's deep dive into each of these steps to look at how we are going to implement them. First of all, we have to take a corpus and create a word and coding dictionary. So here, let's say we have taken this corpus, a huge text, where we need to create and breakdown into different sentences. We are actually splitting them by next new line character. So, this is the kind of corpus that we have developed. So, we have converted into lower case and split the data to get it into a list of sentences. So, the corpus is ready. The next step is to create a word and coding dictionary, so this is how we can use a tokenizer, train it on this particular corpus and create the vocabulary or the word and coding dictionary. Now the next step is to create the n-Gram sequence. So, you must have created already the sequence of all of your corpus. All of the sentences in your corpus. So let's say, one of those sequence is this. So in order to create an n-Gram sequence. You'll write a loop and create an n-Gram sequence like this. So here, you see from 54 to 23. Starting off with two elements and then going into three, to four, to five, so on and so forth. Up until we have an entire sentence or an entire sequence of a sentence. So, this is called an n-Gram sequence of a sequence. So, the next step is to pre-pad all of these sequences. Let's say you have a text sequence like this. The pre-padded n-Gram sequences would look like this. So that, we make the length of all the n-Gram sequences to be equal because the model would be able to handle different input sizes. We need to be consistent with the input size, thus padding is essential in that particular case. Now in order to actually segregate the features and the labels. What we need to is simply extract the label, which would be the last element of your n-Gram sequence. And the rest of the elements of it would be reserved as the input X. Similarly, here for example we have up until 54th, which is the second last element. We have captured it, the entire sequence would be treated as input X and the last label. The last element of that pre-padded sequence would be treated as a label. And similarly for input for the second sequence, and for the third sequence. So now, we are going to train the model on all of these n-Gram sequences of all the sentences and then test it on a C-text. So, in the next lesson, we'll look at how we are going to implement all of these steps.





### **Predicting the next word**

Selecting transcript lines in this section will navigate to timestamp in the video

- So let's look at how we are going to implement all of these five steps in order to generate new text. So let's quickly import all of the required packages. So we are using the same set of packages same functions and classes. So if you run the cell let's make sure that we're using TensorFlow 2.x. Then the first step is to create a corpus. So here, I've taken this very long string from the book Harry Potter. So this is a text from there. Raw text, long string. And the first step is to tokenize it. So we have to instantiate the tokenizer and then we are required to create a corpus. So corpus is basically a list of sentences. So first of all, let's can word it into lowercase so that's not a problem. And then what we'll do is we will split this entire string on new line characters. So that'll give us sentences split by new lines. So let's give it a run and you see that we have a list of so many sentences now. So this is what we needed from the corpus. So we have a list of sentences ready. The next step is to train the tokenizer. So create a word encoding dictionary. So fit on texts, so we have to train the tokenizer on the corpus that we have created. And similarly we have to calculate the entire vocabulary size. So how many unique words do we have? So that can be calculated from the length of word index the word encoding dictionary that we have created. Plus one for the auto-vocabulary token You run it. So we have total of 116 words plus one for the auto vocabulary token. Now the next step is to create N-gram sequence. So for each of those sentences that we have in the corpus, so we'll run over each of those sentence, create a sequence and then for each of those sequences we will get the N-gram sequence and then pad them. So here is how we are doing it. So this is where we are going to basically store all of our sequences. So input sequence, create an empty list. Then you're going to iterate all of each sentence in the corpus. Create a text to sequences, create the sequence of that particular line. And then before all of the tokens that are present in that sequence, we are going to create the N-gram sequence as we saw in the previous video. So this particular code snippet, these two lines are doing the same work. We are creating the N-gram sequences of all of those tokens available in that particular line and we are doing it for the entire corpus. So once your N-gram sequences are done then the next step is to pad those sequences. And for that, we are going to use pre padding because we will have to extract the last element of all the sequences is going to be treated as the label. So we want all the zeros to be put in front. So you have to get the max sequence length. Now the max sequence length would be the largest sequence. So we have to first get the length of all the sequences and then get the maximum out of that. So this is what this line is doing. And then we have to pad those sequences pass those input sequences to the pad sequences function plus the max length. Padding is pre, padding type is pre. And let's quickly pad those sequences. Now comes the important part where we have to extract the features and the labels. So to extract the features, the features is basically the entire input sequence except the last element. So for that we can use slicing. We can basically skip the last element to exempt the last element the entire sequences would be treated as X, the feature variable. And the last element would be treated as the label. So we have basically extracted our features and label. Now the label is basically text, right now the number. So we need to actually one-hot encode all of these labels. So for that, we can use the TensorFlow keras to underscore categorical variable. So this categorical variable will basically what we'll do is it'll one-hot encode that particular label and it'll create a sparse vector as we discussed. And one would be placed where that word is actually present. So one would be placed in that particular zero vector that we would have. So let's quickly run this. And then let's print the encoding for let's say word mud and what do we have in our feature variable and the feature area for the first, let's say four sequence. And so you see the one-hot encoded label the first feature sequence, and the encoding for the word mud. Now comes the next part. So the important part is to define our model. So we're going to use the LSTM model. Now the first step would still remain the same. We create an embedding class. Vocabulary sizes already defined. And 64, we are using embedding dimension as 64 and input length equals max sequence length minus one. Minus one because the last element of the sequence is actually available or reserved for the label. So that's why the max sequence length minus one. Now the next layers would be the bidirectional layer and this would be an LSTM. And we want the cell state to go in both the direction. You can change the number of output so let's say we keep it to 32. You can change it to as per your wish. And the last layer is the dense layer with the vocabulary size as the number of classes that we have to predict. And since we have a multi-class problem so the activation that we're going to use is softmax. Then we're going to compile this model with the categorical crossentropy loss function and optimizer is still adam creating the model pass the X and Y that we have created epochs. We are training it for 500 epochs. You can change the number as you like. So this will take a few seconds now. So you see the model is done training. We have right for 500 epochs and we see an accuracy of 100%. Let's quickly plot the metrics. So we have written the same visualization function. The more training is stored in the history variable. So let's quickly call the right function, plot metric with the history. So you see this graph over here shows how the accuracy of the model has increased from epoch by epoch. So here in the range of let's say 290 or 300 epoch, we reached close to hundred percent accuracy and it took the model somewhere around 250, 200 epochs to reach that accuracy level. And then now that the model is trained, let's try to generate new texts. So we give the model a starting sentence, a seed text. So based on that, it will try and create new words. So let's define how many words we want. So let's say we want the next hundred words ahead of this seed text, which is, 'It was a cold night.' And then we're going to run the loop for the 100 times to predict a hundred words. And the first step is to actually create the sequence then pad those sequences and then use the train model to predict the class. So we're going to use the predict classes function and pass this token list to the model. Once you pass this token list then for that particular token, whatever the model predict gives the label you have to actually find out, reference that number with the word encoding dictionary, get the word and add it to your seed text and then print the seed text. So if we run this, so it'll take a few seconds to predict all of the hundred words. So you see, this is our text. 'It was a cold night, steam pouring from under her vivid hair gave the impression that her whole head was on fire fire fire was to gryffindor returning to and percy percy percy' So you see, there are a lot of repetitions of the words but still the model has done decently. Well, it's not the correct, and the sentences are not right but we are off to a great start. So in the next video, we're going to work over a more complex dataset.

### **Challenge: Generate poetry**

Selecting transcript lines in this section will navigate to timestamp in the video

(upbeat music) - So based on whatever we have learned so far about ex-generation, here is a challenge for you. Now in this challenge, you have to generate poetry. Now, how would you do that? So we are going to take a text from Shakespeare's sonnet and we're going to train the model on that particular text. So once your model is trained on Shakespeare's poetry, there are chances that when you provide the seat text you would be able to generate poetry like Shakespeare. So here in this particular challenge, you have to fill up all the blanks where you see you are required to pass on the function or the radio and define the layers that you have to add to the model. So for example, you have to follow all of the steps that we discussed in the previous lesson. One by one, create a Corpus set up the tokenizer. Then make sure that you define all the layers of your model. Then visualize the metrics and you have to pass on the seat takes to generate new text and generate the poetry. Since the model is learning on Shakespeare's data.

### **Solution: Generate poetry**

Selecting transcript lines in this section will navigate to timestamp in the video

(upbeat music) - [Instructor] So here's how we are going to solve the Poetry Generation Challenge. So first of all, let's import all of the packages. Make sure again, you're using TensorFlow to one X. Then we are going to use this sonnet text which I've hosted on my GitHub repo. So here's the link. So quickly download the data and then read the data. So we are printing the length of the Shakespeare text, which has turned out to be 28,000. This is a fairly good and large dataset to train on. So let's quickly create the Corpus, which is the first step. So shakespeare.text for the first of all, convert it into lower case. And then we have to split it on new line. So once that is done, let's see what the Corpus look like. So what does this look like? So we have quite a few sentences over here. As you can see, the Corpus is ready. The next step is to set up the tokenizer. So the tokenizer we have instantiated already. Let's quickly run that. Train the tokenizer on the Corpus. So that is done. Then to calculate the vocabulary size, we can use the word index dictionary and plus one for the outer vocabulary token. Once that is done, our word encoding dictionary is ready. We have 1,544 unique words in there. So that is the vocabulary size. Now in order to create sequences, basically the engram sequences. So here tokens equals tokenizer . text to sequences. So that is how you're going to solve it. And this will basically create all the engram sequences for it, and it will keep on appending it in the input sequences list. Then the next step is to pad those sequences. So for I N input\_sequences that will give us the max sequence length. And then max length while patting, we have to pass on the parameter. Max sequence length padding type is pre. Quickly pad all the sequences then create a feature area. So for that, we need all the elements except the last one. So that would be minus one because the ending index is excluded. And the label would be the last index. Now, since this is a classification problem, so we have all the labels and we need to convert those classes into one hot encoding variables. So let's quickly do that using the two categorical function in the kettle's utility library. And once that is done, we have to pass on the labels to this. And the number of classes is actually equal to the vocabulary size. So let's quickly run that. And here comes the important part, which is the model definition. So, first of all, very easy. We have the embedding layer. We have set vocabulary size as the input. And the input length is basically Mac sequence length minus one. Embedding dimension is 120 and then get us.layers the next layer is going to be bi-directional LSTM. So let's add that. Then the last layer is going to be the dense layer, which is going to have vocabulary size as there are multiple classes and multiple classes are equal to the vocabulary size. So since they're multiple classes, so the activation function to be used as softmax. Then learning rate is defined so that we have the step size that will actually tell the optimizer as in what the step size should be while learning. The last function is going to be categorical crossentropy. So let's quickly run the model, train it and store the model training into history. Now this will take a few minutes. So the model has done training and we have an accuracy of 88%. So let's quickly visualize these metrics. So we are using the broad metric function again and we are going to check the accuracy. So this is turning out to be quite interesting. We ran it for 208 box and the accuracy kept on fluctuating. So it stayed stable around 0.88 somewhere. So if you see came down a little bit with a few spikes. You see here with 0.61 and came down in a few a box , but yeah. Let's quickly generate the new text. So it was a cool night is the seed text that I've given. You can give it any other seed texts. I am then predicting the next a hundred words. So bad sequences all the token lists that we have got from those sequences and then model.predict. I need to predict the classes. So based on whatever predicted interior that I get, then I'll have to reference it back from the word encoding dictionary which is being done over here. And then I'll concatenate it in the seed text to get my entire seed. Or you can say the text with new hundred words. So let's quickly run this. We see, it was cool to see his skill are me like done hence me done thee hence Women's me done me, praise the whatever. So it's better than the one that we did in the previous lesson. So there are a number of ways that you can build over this. So a lot of hyper-parameters need to be doing. You can change the number of eight box, you can change the number of embedding dimension, the layers. You can add another LSTM layer to it. There are a bunch of ways that you can actually build over this and make it perform better. So we're just getting started with natural language processing over here and there is a lot to cover. So we have just covered the deep learning foundations for NLP in this particular course but feel free to dive in deep and learn more about encoders. Learn more about transformers, which do a better job as compared to the models that we have learned. This will definitely give you a kickstart in NLP using deep learning.

## **Question 1 of 2**

Which of the following is the appropriate loss function for text generation problems?

* MeanSquaredError
* hinge  
  Incorrect  
  Used for binary classification problems.
* categorical\_crossentropy  
  Correct  
  categorical\_crossentropy is the default loss function to use for multi-class classification problems using keras.
* binary\_crossentropy

## **Question 2 of 2**

How do you convert padded N-gram sequences to features and labels for a text generation problem?

* Reserve the last token in the sequence as the Label.  
  Correct  
  The last token in the sequence after a series of tokens is to be predicted and works as the label for training the model.
* Reserve the first token in the sequence as the Label.
* Reserve half of the tokens in the sequence as Label.
* Reserve the last two tokens as the label.

### **Learning more about NLP with TensorFlow**

Selecting transcript lines in this section will navigate to timestamp in the video

- You have just spent some time with a wide range of concepts used to process natural language. From encoding words to generating board embeddings, to classifying text using neural networks and learning advanced algorithms like RNNs and CNNs. With the skills learned here, you are well on your way to dive deeper into NLP research problems and use transformers like BERT or GPT-3. Feel free to connect with me on LinkedIn or you can subscribe to my YouTube channel where I discuss anything and everything about data. Thanks for watching and I'll catch you next time.