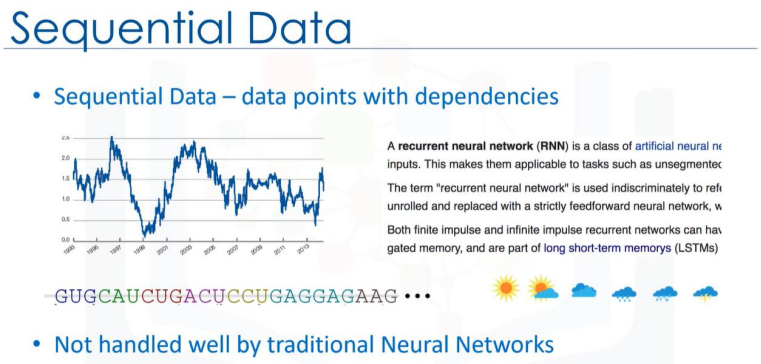
Module 3 - Recurrent Neural Networks (RNNs)

Learning Objectives

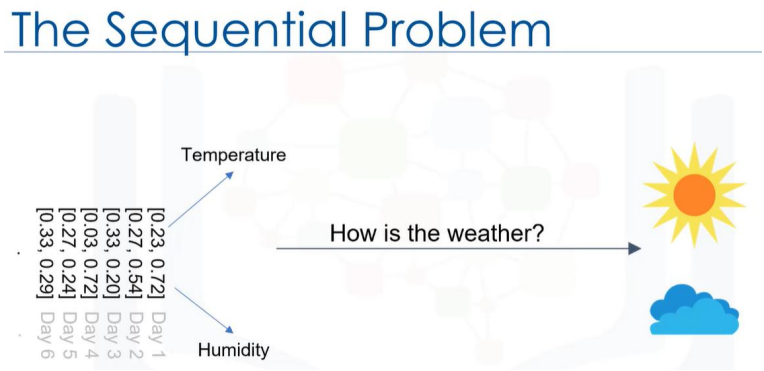
In this lesson you will learn about:

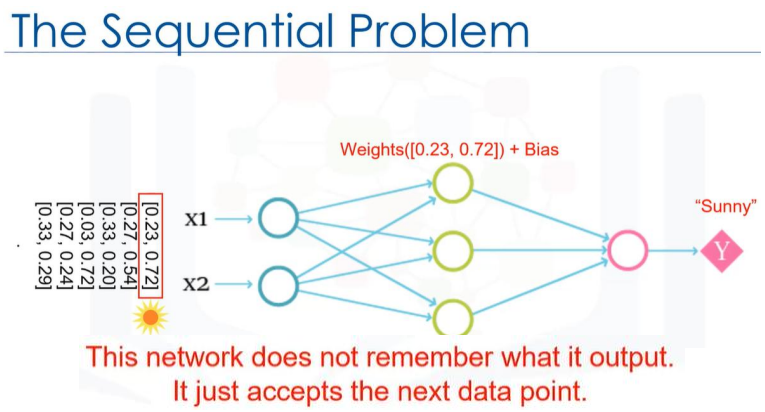
- The Recurrent Neural Network Model
- Long Short-Term Memory
- Recursive Neural Tensor Network Theory
- Applying Recurrent Networks to Language Modelling

3.1 The Sequential Problem



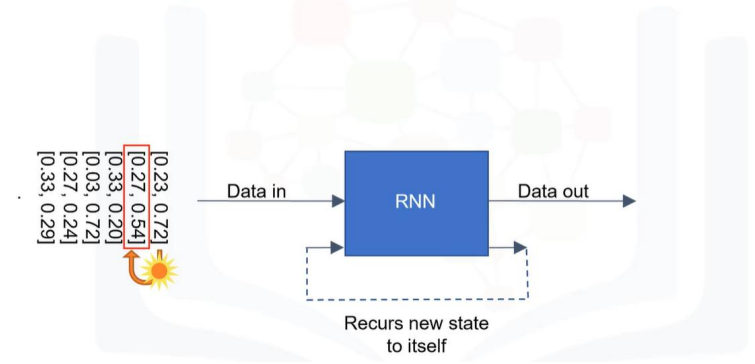
Hello, and welcome! In this video, we'll provide an overview of **sequential data**, and explain why it poses a problem for traditional neural networks. Whenever the points in a dataset are dependent on the other points, the data is said to be sequential. A common example of this is a time series, such as a stock price or sensor data, where each data point represents an observation at a certain point in time. There are other examples of sequential data, like sentences, gene sequences, and weather data.
But traditional neural networks typically can't handle this type of data. So, let's see why we can't use feedforward neural networks to analyze sequential data.





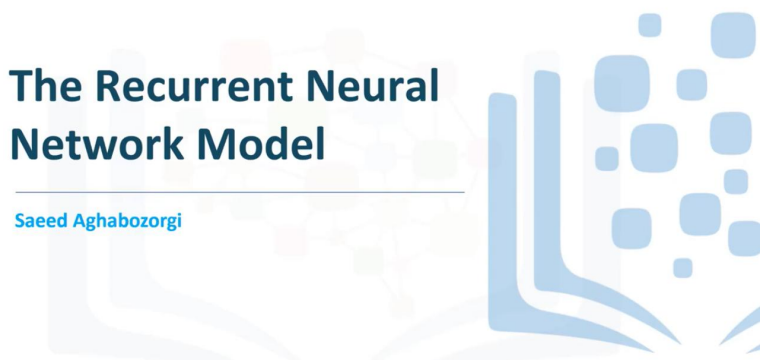
Let's consider a sequential problem to see how well-suited a basic neural network might be. Suppose we have a sequence of data that contains temperature and humidity values for every day. Our goal is to build a neural network that imports the temperature and humidity values of a given day as input and output. For instance, to predict if the weather for that day is sunny or rainy. This is a straightforward task for traditional feedforward neural networks. Using our dataset, we first feed a data point into the input layer. The data then flows to the hidden layer or layers, where the weights and biases are applied. Then, the output layer classifies the results from the hidden layer, which ultimately produces the output of sunny or cloudy. Of course, we can repeat this for the second day, and get the result. However, it's important to note that the model does not remember the data that it just analyzed. All it does is accept input after input, and produces individual classifications for every day. In fact, a traditional neural network assumes that the data is non-sequential, and that each data point is independent of the others. As a result, the inputs are analyzed in isolation, which can cause problems if there are dependencies in the data.

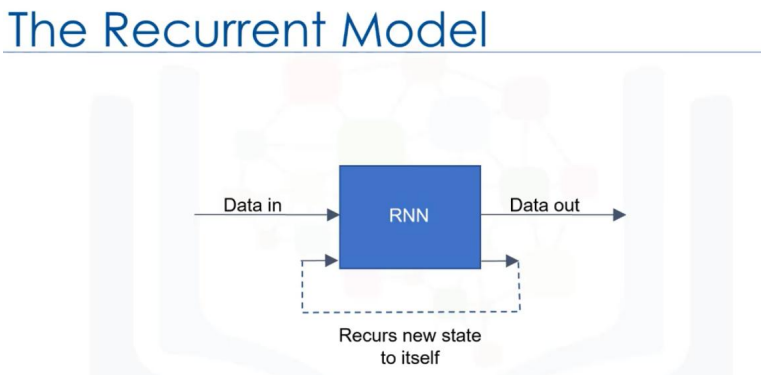
总结:



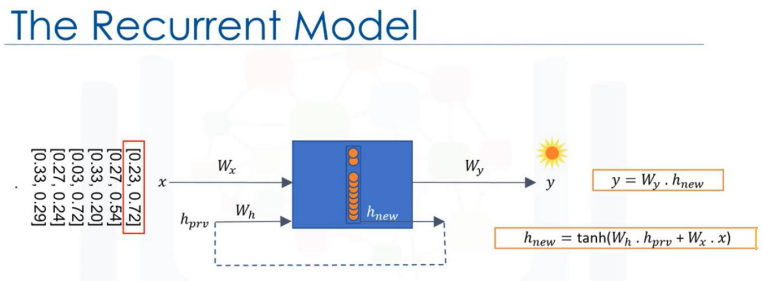
To see how this can be a limitation, let's go back to the weather example again.As you can imagine when examining weather, there is often a strong correlation of the weather on one day having some influence on the weather in subsequent days. That is, if it was sunny on one day in the middle of summer, it's not unreasonable to presume that it'll also be sunny on the following day. A traditional neural network model does not use this information, however, so we'd have to turn to a different type of model, like a recurrent neural networks model. A **Recurrent Neural Network** has a mechanism that can handle a sequential dataset. By now, you should have a good understanding of the problem that the recurrent neural network model is trying to address.

3.2 The RNN Model



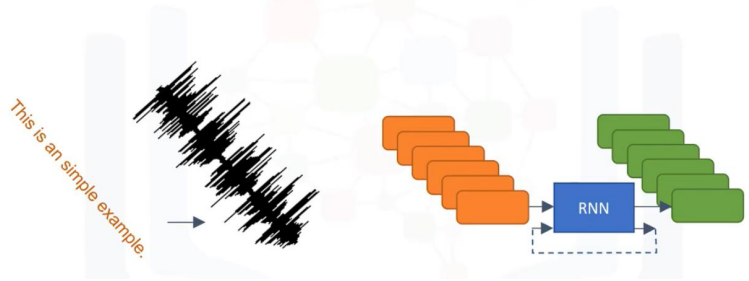


Hello, and welcome! In this video, we'll provide an overviewof Recurrent Neural Networks, and explain their architecture. A Recurrent neural network, or RNN for short, is a great tool for modeling sequential data. The RNN is able to remember the analysis that was done up to a given point by maintaining a state, or a context, so to speak. You can think of the "state" as the "memory" of RNN, which captures information about what's been previously calculated. This "state" recurs back into the net with each new input, which is where the network gets its name.



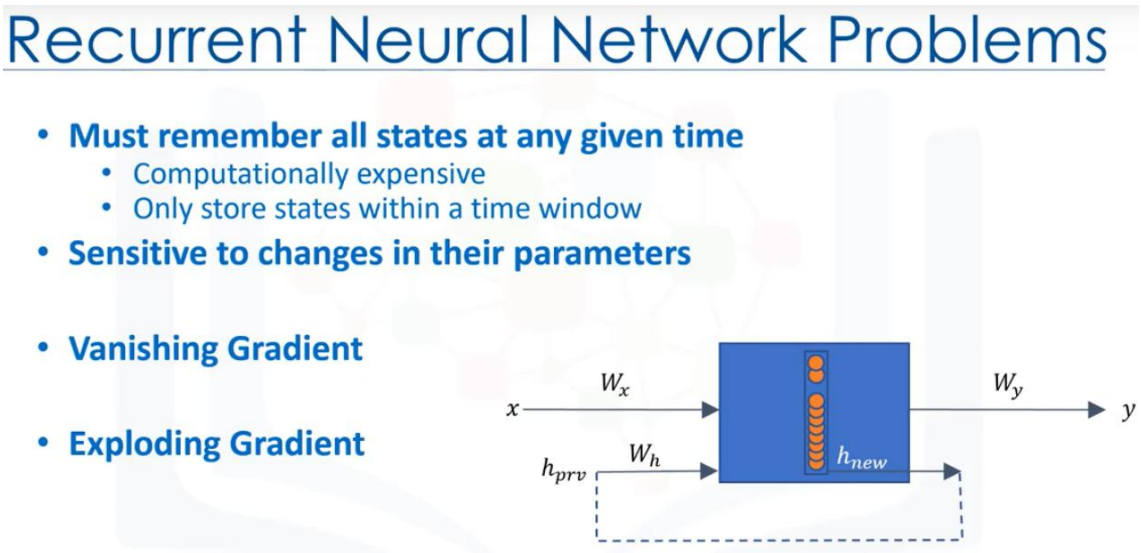
Let's take a closer look at how this works. Imagine that the RNN we're using has onlyone hidden layer. The first data point flows into the network as input data, denoted as x. As we mentioned before, the hidden units, also receive the previous state or the context, denoted as h_prv, along with the input. Then, in the hidden layer, two values will be calculated: First, the new or updated state, denoted as h_new, is to be used for the next data point in the sequence. And second, the output of the network will be computed, which is denoted as y. The new state is a function of the previous state and the input data, as shown here. If this is the first data point, then some form of "initial state" is used, which will differ, depending on the type of data being analyzed. But, typically it is initialized to all zeroes. Please notice that W_x, in this equation, is the weight matrix between the input and the hidden unit. And W_h are the weights that are multiplied by the previously hidden state in the equation. The output of the hidden unit is simply calculated by multiplication of the new hidden state and the output weight matrix. So, after processing the first data point, in addition to the output, a new context is generated that represents the most recent point. Then, this context is fed back into the net with the next data point, and we repeat these steps until all the data is processed.

总结:



Recurrent neural networks are extremely versatile and are used in a wide range of applicationsthat deal with sequential data.

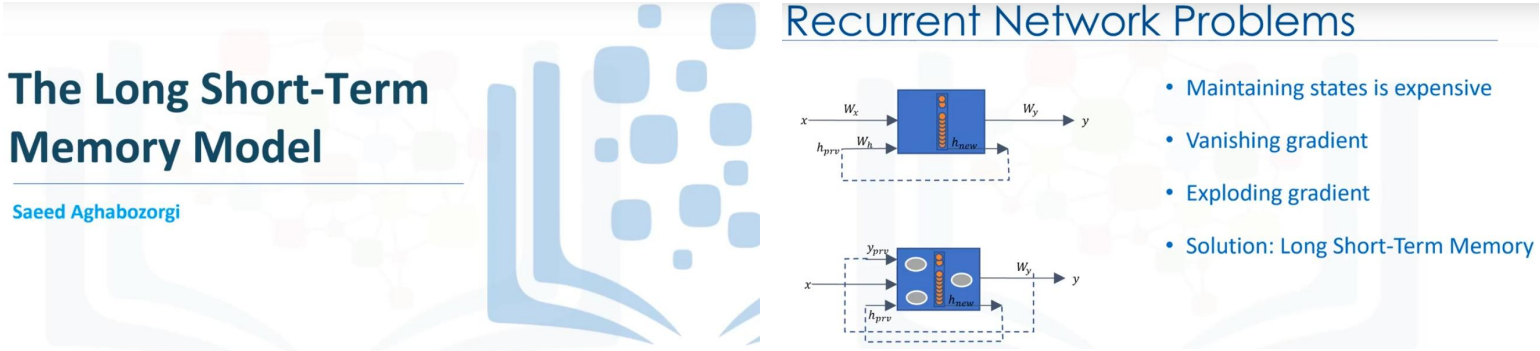
- One of these applications is speech recognition. As you can see, it is a type of a many-to-many network. That is, the goal is to consume a sequence of data and then produce another sequence.
- Another application of RNN is image captioning. Although it's not purely recurrent, youcan create a model that's capable of understanding the elements in an image. Then, using the RNN, you can string the elements as words together to form a caption that describes the scene. Typically, RNN has outputs at each time step, but it depends on the problem that RNN is addressing. For example, in this type of RNN for captioning, there is one input as image, and the output is a sequence of words. So, it is sometimes called one-to-many.
- RNN can also be Many-to-one, that is, it consumes a sequence of data and produces just one output.For example, to predict the stock market price, where we might only be interested in the price of a particular stock in tomorrow's market. Or, for sentiment analysis, where we may only care about the final output, not the sentiment after each word.



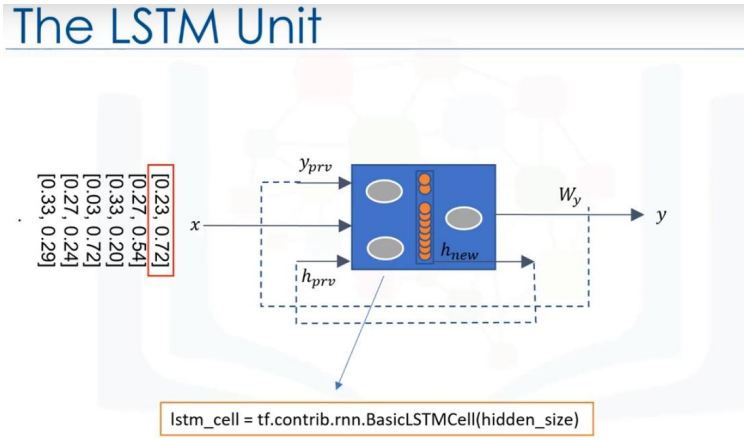
We've only covered a few applications, but variants of the recurrent models are continuingto solve increasingly complex problems, which are beyond the scope of this video. Despite all of its strengths, the recurrent neural network is not a perfect model. One issue is that the network needs to keep track of the states at any given time. There could be many units of data, or many time steps, so this becomes computationally expensive. One compromise is to only store a portion of the recent states in a time window. Another issue is that Recurrent Neural Networks are extremely sensitive to changes in their parameters. As a result, gradient descent optimizers may struggle to train the net. Also, the net may suffer from the "Vanishing Gradient" problem, where the gradient drops to nearly zero and training slows to a halt. Finally, it may also suffer from the "Exploding Gradient", where the gradient grows exponentially off to infinity. In either case, the model's capacity to learn will be diminished. By now, you should have a good understanding of the main ideas behind the recurrent neural network model.

总结:

3.3 The LSTM Model



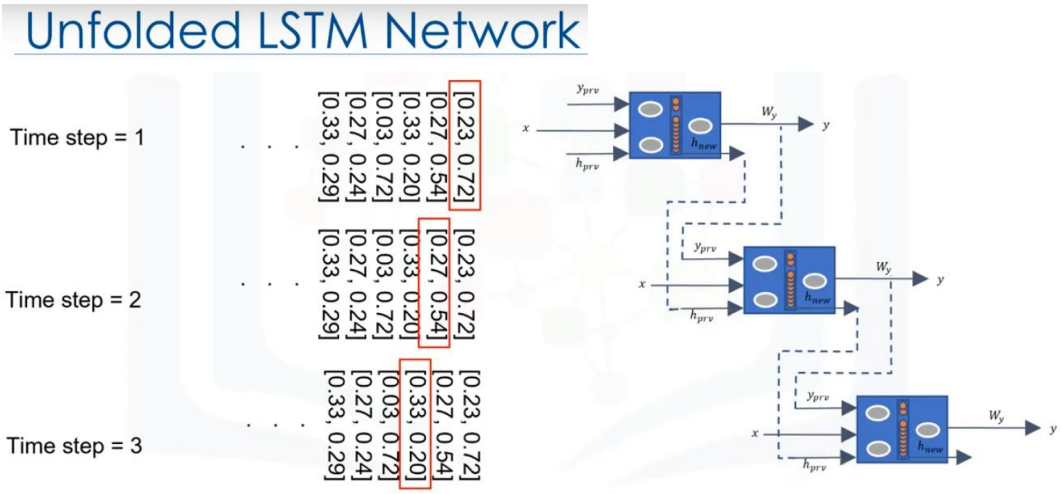
Hello, and welcome. In this video, we'll be reviewing the **LongShort-Term Memory Model**. The recurrent neural network is a great tool for modeling sequential data, but there are a few issues that need to be addressed in order to use the model on a large scale. For example, recurrent nets are needed to keep track of states, which is computationally expensive. Also, there are issues with training, like the vanishing gradient, and the exploding gradient. As a result, the RNN, or to be precise, the "**Vanilla RNN**" cannot learn long sequences very well. A popular method to solve these problems is a specific type of RNN, which is called the Long Short-Term memory, or LSTM for short. LSTM maintains a strong gradient over many time steps. This means you can train an LSTM with relatively long sequences.



An LSTM unit in Recurrent Neural Networks is composed of four main elements. The "**memory cell**", and **three logistic "gates"**. The "memory cell" is responsible for holding data. The write, read, and forget gates, define the flow of data inside the LSTM.

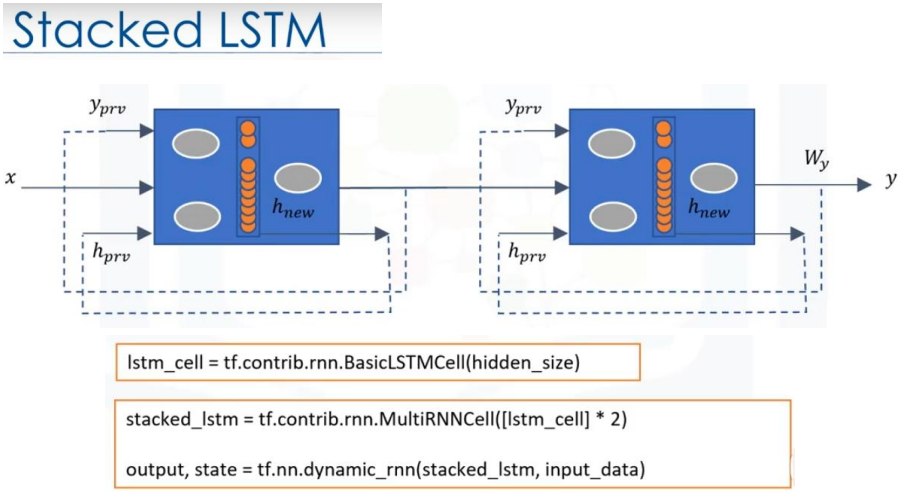
- The Write Gate is responsible for writing data into the memory cell.
- The Read Gate reads data from the memory cell and sends that data back to the recurrent network.
- And the Forget Gate maintains or deletes data from the information cell, or in other words, determines how much old information to forget.

In fact, these gates are the operations in the LSTM that execute some function on a linear combination of the inputs to the network, the network's previous hidden state and previous output. I don't want you to dive into the details of how these gates interact with each other, but what is important here, is that, by manipulating these gates, a Recurrent Network is able to remember what it needs in a sequence and simply forget what is no longer useful. Now, let's look at the data flow in LSTM Recurrent Networks.

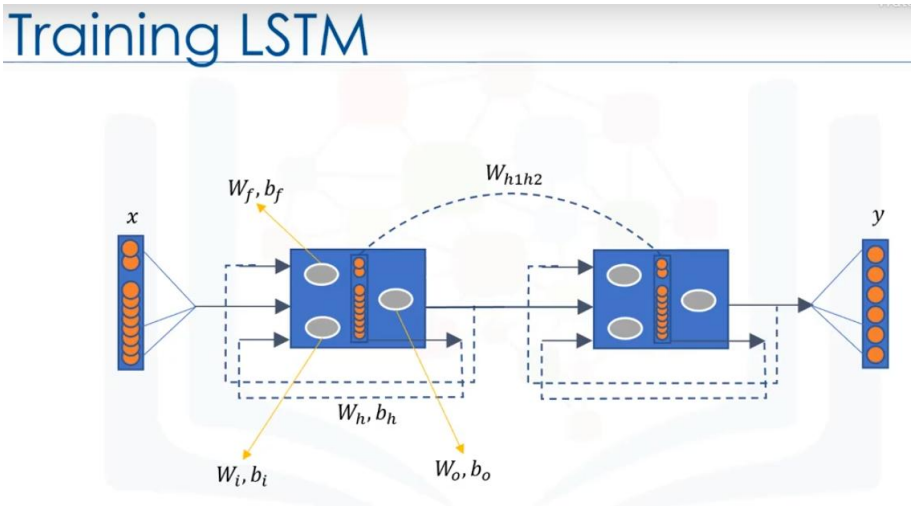


So, let's see how data is passed through the network.In the first time step, the first element of the sequence is passed to the network. The LSTM unit uses the random initialized hidden state and output to produce the new hidden state and also the first step output. The LSTM then sends its output and hidden state to the net in the next time step. And the process continues for the next time steps. So, as you can see, the LSTM unit keeps two pieces of information as it propagates through time. First, a hidden state, which is in fact, the memory the LSTM accumulates using its gates through time, and second, the previous time-step output.

总结:



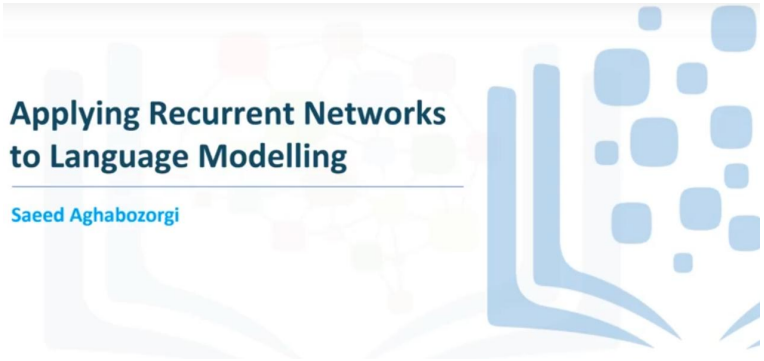
The original LSTM model has only one single hidden LSTM layer.But, as you know, in cases of simple feedforward neural network, we usually stack layers to create hierarchical feature representation of the input data. So, does this also apply for LSTM's? What about if we want to have a RNN with stacked LSTM, for example, a 2-layer LSTM? In this case, the output of the first layer will feed as the input into the second layer. Then, the second LSTM blends it with its own internal state to produce an output. Stacking an LSTM allows for greater model complexity. So, the second LSTM can create a more complex feature representation of the current input. That is, stacking LSTM hidden layers makes the model deeper, and most probably leads to more accurate results.

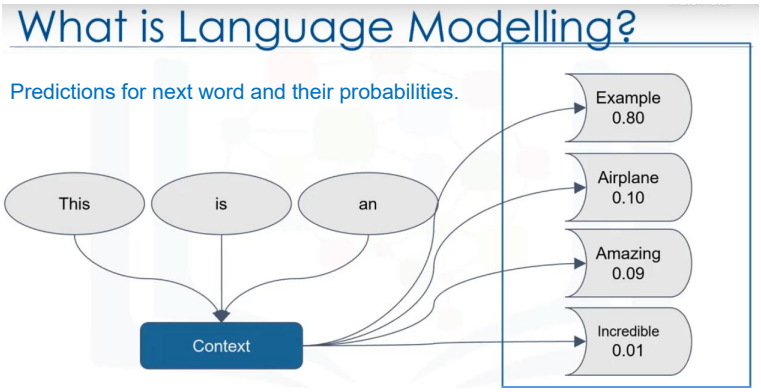


Now, let’s see what happens during the training process.The network learns to determine how much old information to forget through the forget gate. So, the weights, denoted as Wf, and biases denoted as bf will be learned through the training procedure. We also determine how much new information, called x, to incorporate through the input gate and its weights. We also calculate the new cell state based on the current and the previous internal state, so the network has to learn its corresponding weights and biases. Finally, we determine how much of our cell state we want to output an output gate. Basically, the network is learning the weights and biases used in each gate, for each layer. By now, you should have a good understanding of the main ideas behind the LSTM.

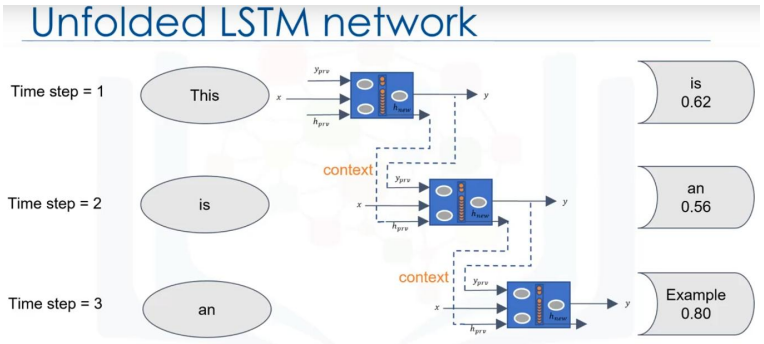
总结：

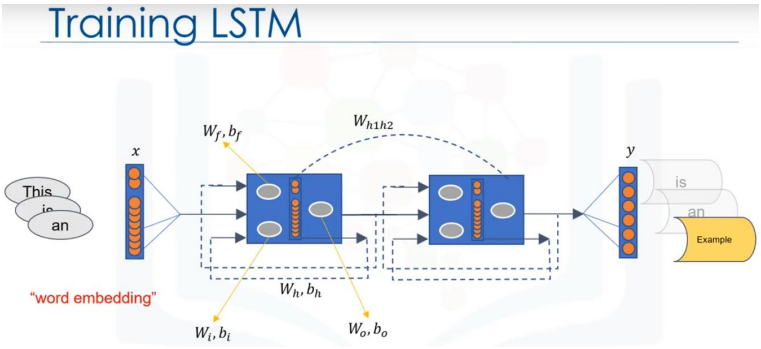
3.4 Applying RNNs to Language Modelling





Hello, and welcome. In this video, we'll be reviewing how toapply recurrent neural networks to language modelling. Language modelling is a gateway into many exciting deep learning applications like speech recognition, machine translation, and image captioning. At its simplest, language modelling is the process of assigning probabilities to sequences of words. So for example, a language model could analyze a sequence of words, and predict which word is most likely to follow. So with the sequence, "This is an", which you see here, a language model might predict what the next word might be. Clearly, there are many options for what word could be used as the next one in the string. But a trained model might predict, with an 80 percent probability of being correct, that the word "example" is most likely to follow. This boils down to a sequential data analysis problem. The sequence of words forms the context, and the most recent word is the input data. Using these two pieces of information, you need to output both a predicted word, and a new context that contains the input word.

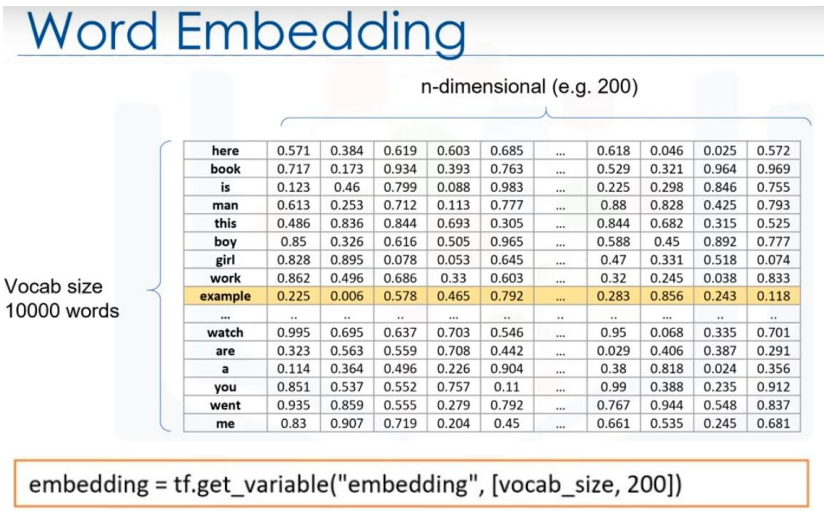


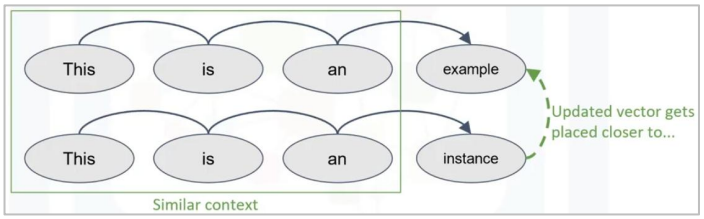


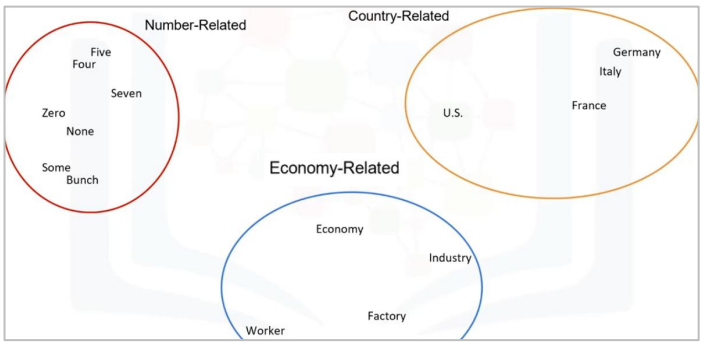
Recurrent neural networks are a great fit for this type of problem.At the first time step, a recurrent net can receive a word as input along the initial context. It generates an output. The output word with the current sequence of words as the context will then be re-fed into the network in the second time step. A new word would be predicted. And, these steps are repeated until the sentence is complete.

Now, let's take closer look at an LSTM network for modeling the language.In this network, we will use an RNN network with two stacked LSTM units. For training such a network, we have to pass each word of the sentence to the network, and let the network generate an output. For example, after passing the words "this" and "is", if we pass the word "an" in the third time step, we expect the network to generate the word, "example," as output. But notice that we cannot easily pass a word to the network. We have to convert it into a vector of numbers somehow. We can use "word embedding" for this purpose.

总结:

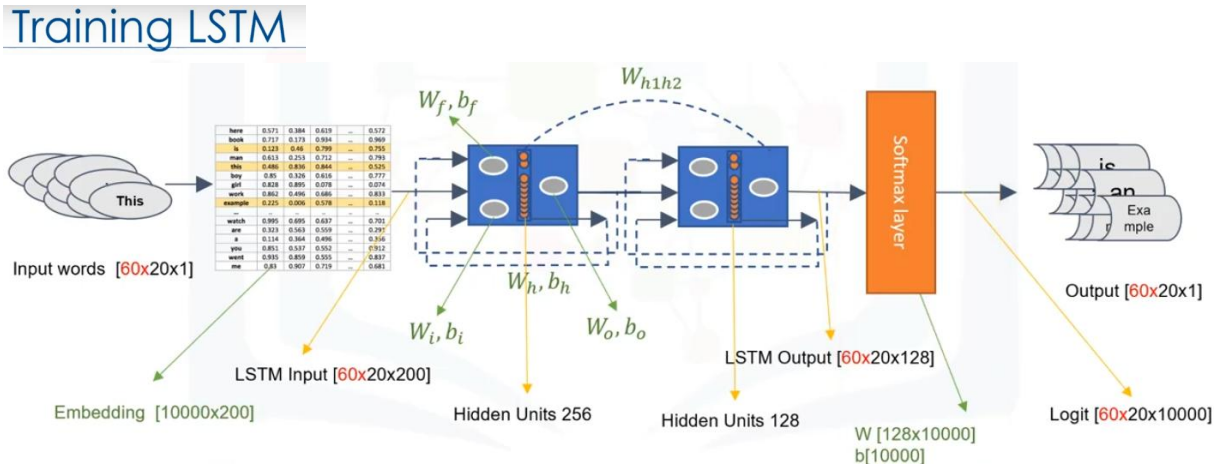






Let's quickly examine what happens in word embedding.An interesting way to process words is through a structure known as a Word Embedding. A word embedding is an n-dimensional vector of real numbers for each word. The vector is typically large, for example, 200 length. You can see what that might look like with the word "example" here. You think of word embedding as a type of encoding for text-to-numbers. Now the question is, how do we find the proper values for these vectors? In our RNN model, the vectors (also known as the matrix for the vocabulary) are initialized randomly for all the words that we are going to use for training.

Then, during the recurrent network's training, the vector values are updated based on thecontext into which the word is being inserted. So, words that are used in similar contexts end up with similar positions in the vector space. This can be visualized by utilizing a dimensionality-reduction algorithm.
Take a look at the example shown here. After training the RNN, if we visualize thewords based on their embedding vectors, the words are grouped together either because they're synonyms, or they're used in similar places within a sentence. For example, the words "zero" and "none" are close semantically, so it's naturalfor them to be placed close together. And while "Italy" and "Germany" aren't synonyms, they can be interchanged in several sentences without distorting the grammar.



Ok, now let's look back at the RNN that we've been using.Imagine that the input data is a batch with only one sequence of words. Think of it as a batch that includes one sentence only - one that includes 20 words. Assume that the vocabulary size of the words is 10,000 words, and the length of each embedding vector is 200. We have to look up those 20 words in the randomly initialized embedding matrix, and then feed them into the first LSTM unit. Please notice that only one word in each time step is fed into the network, and one word would be the output. But, during 20 time steps, the output would be 20 words. In our network, we have 2 LSTM units, with arbitrary hidden sizes of 256 and 128. So, the output of the second LSTM unit would be a matrix of size 20-by-128. Now, we need a softmax layer to calculate the probability of the output words. It "squashes" the 128-dimensional vector of real values to a 10,000-dimensional vector, which is our vocabulary size. This means that the output of the network at each time step is a probability vector of length 10,000. So, the output word is the one with maximum probability value in the vector. Now, we can compare the sequence of 20 output words with the ground truth words. And finally, calculate the discrepancy as a quantitative value, so called loss value, and back-propagate the errors into the network. And of course, we will not train the model using only one sequence. We will use a batch of sequences to train it and calculate the error. So, instead of feeding one sequence, we can feed the network in many iterations - perhaps even a batch of 60 sentences, for example. Now, the key question to be asked is: "What does the network learn when the error is propagated back, in each iteration?" Well, as previously noted, the weights keep updating based on the error of the network-in-training. First, the embedding matrix will be updated in each iteration. Second, there are a bunch of weight matrices related to the gates in the LSTM units which will be changed. And finally, the weights related to the Softmax layer, which somehow plays the decoding rolefor encoded words in the embedding layer. By now, you should have a good understanding of how to use LSTM for language modeling.

总结:

What is NOT TRUE about RNNs?

- ○ RNNs are VERY suitable for sequential data.
- ○ RNNs need to keep track of states, which is computationally expensive.
- ◉ RNNs are very robust against vanishing gradient problem.

✔

What is NOT TRUE about RNNs?

- ○ RNNs are VERY suitable for sequential data.
- ○ RNNs need to keep track of states, which is computationally expensive.
- ◉ RNNs are very robust against vanishing gradient problem.

✔

What application(s) is(are) suitable for RNNs?

- ○ Estimating temperatures from weather data
- ○ Natural Language Processing
- ○ Video context retriever
- ○ Speech Recognition
- ◉ All of the above

✔

Why are RNNs susceptible to issues with their gradients?

- ○ Numerical computation of gradients can drive into instabilities
- ○ Gradients can quickly drop and stabilize at near zero
- ○ Propagation of errors due to the recurrent characteristic
- ○ Gradients can grow exponentially
- ◉ All of the above

✔

总结：