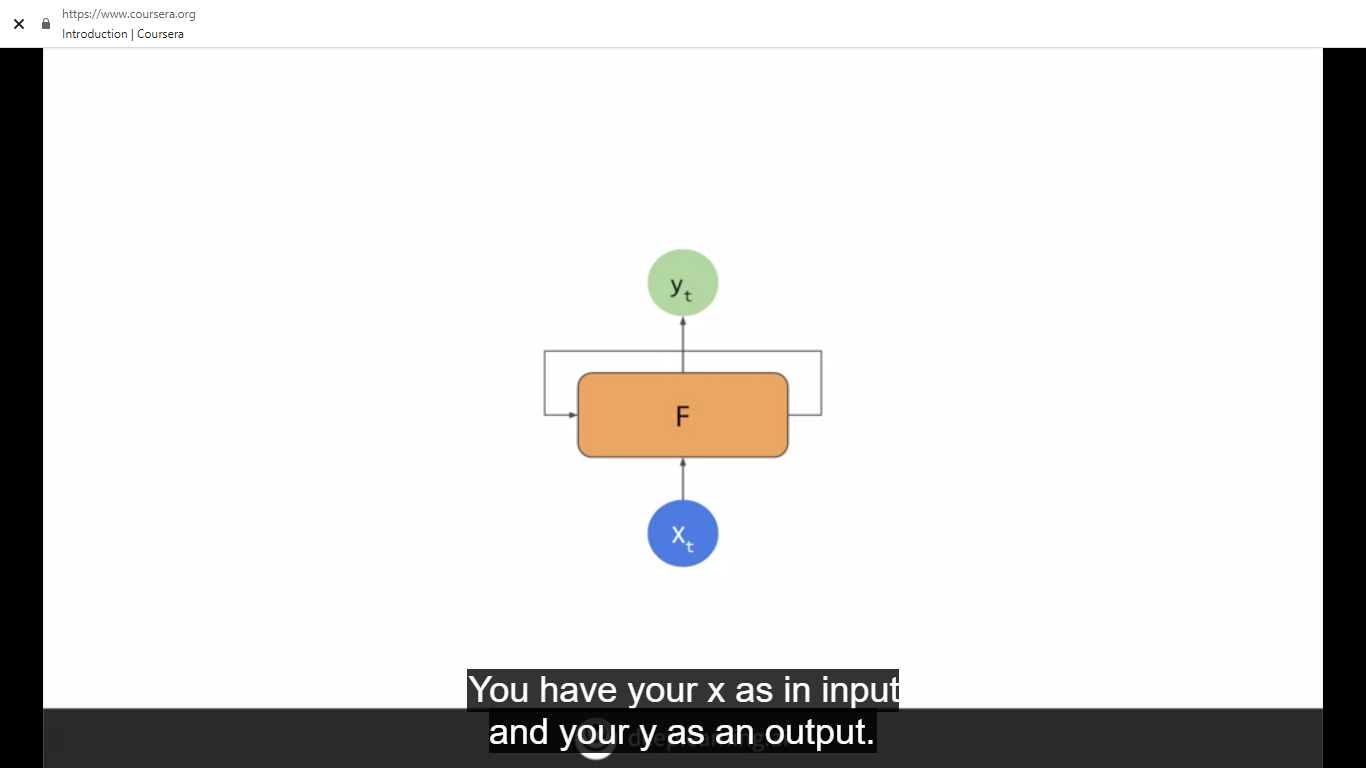
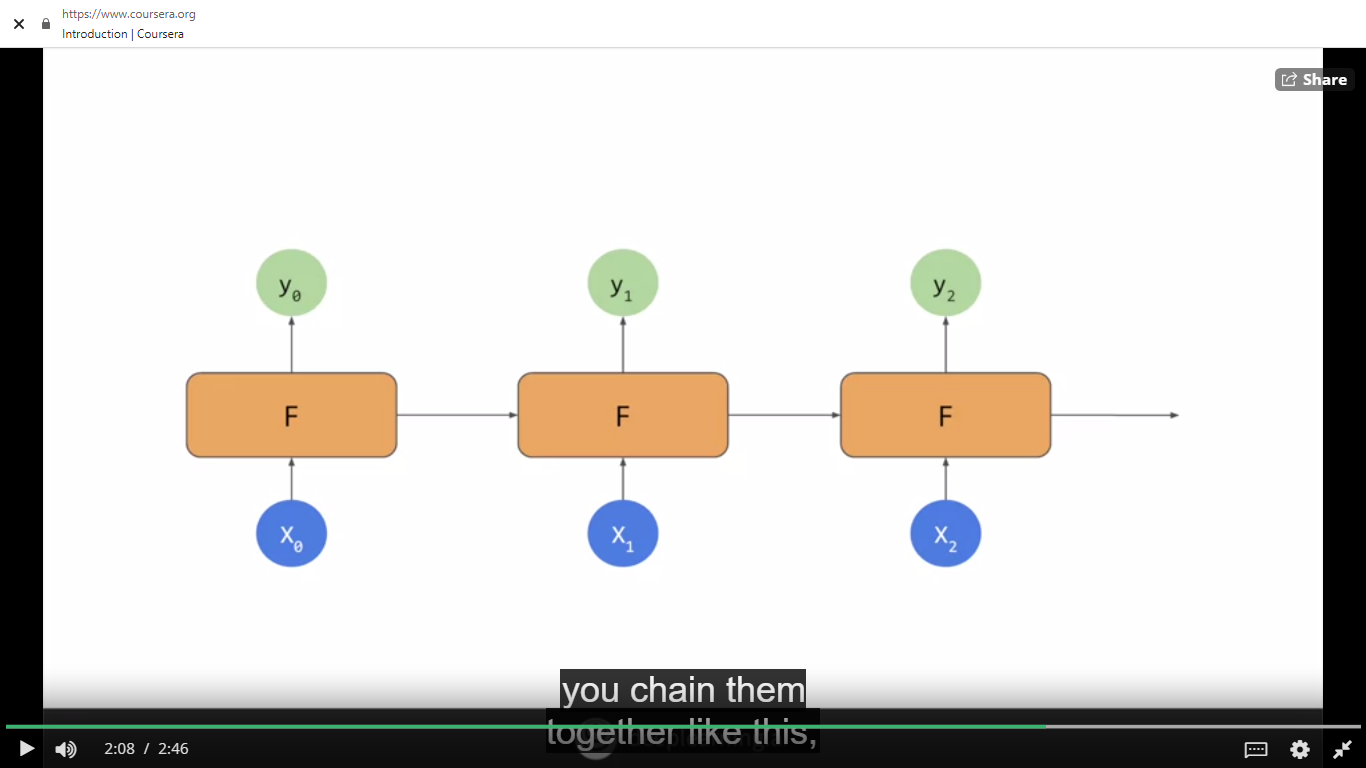
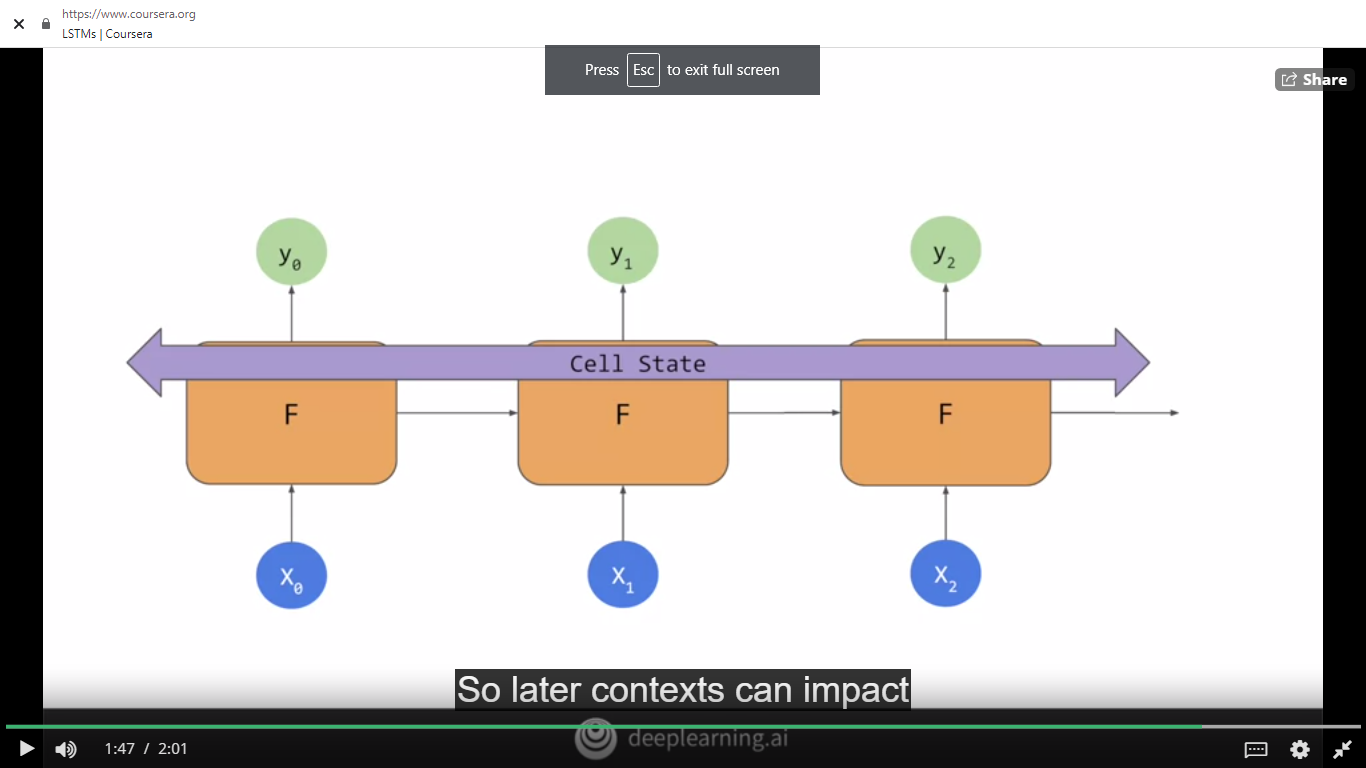
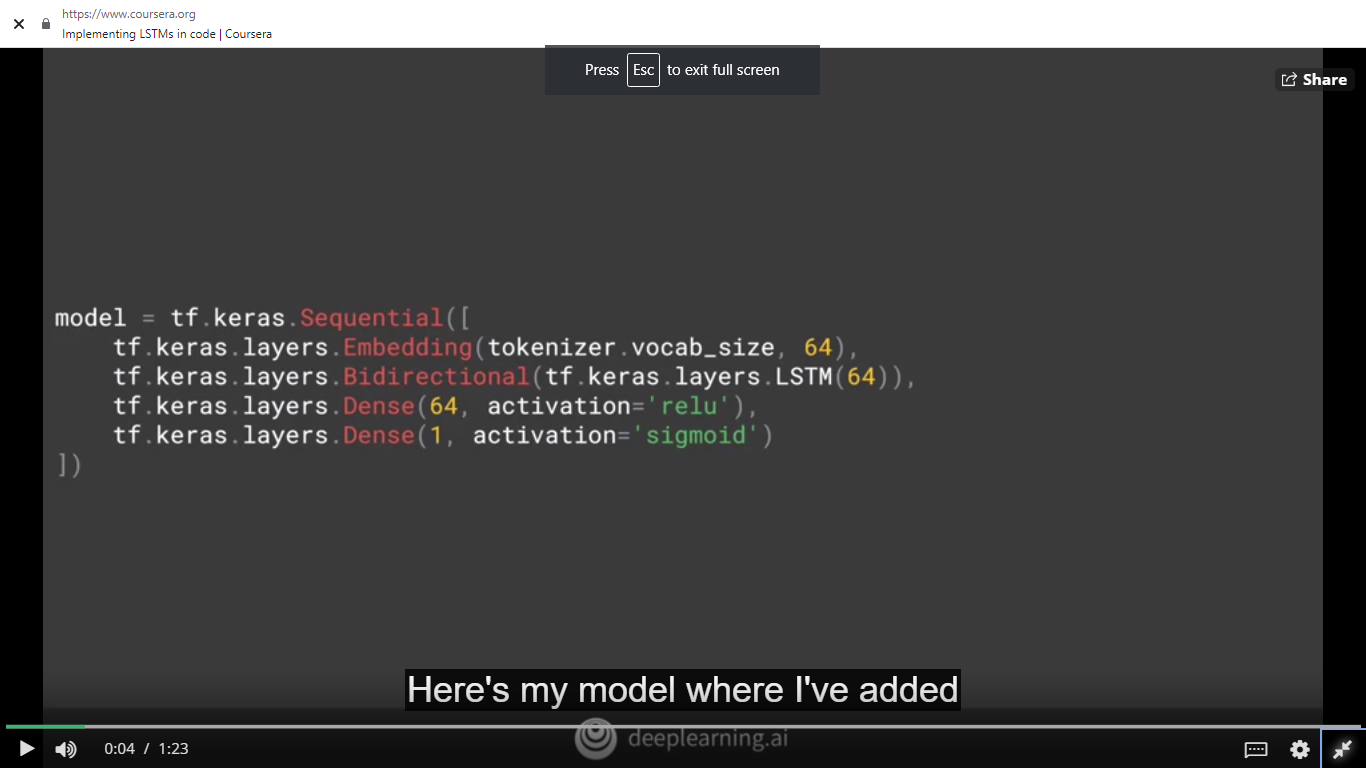
* **Last week,** we looked at doing classification using texts and trying to train and understand the positive and negative sentiment in movie reviews. We finished by looking at the effect of tokenizing words and saw that our classifier failed to get any meaningful results.
  + **Why is that?** The main reason for this was that the context of words was hard to follow when the words were broken down into sub-words and the sequence in which the tokens for the sub-words appear becomes very important in understanding their meaning.
  + **What is the solution?** We use specialized Neural Network Architectures, like an RNN, GRU, or LSTM
* The **basic idea of a recurrent neural network** or RNN is that you have your X as in input and your Y as an output, but there's also an element that's fed into the function from a previous function.



* That becomes a little more clear when you **chain them together**,X\_0 is fed into the function returning Y\_0. An output from the function is then fed into the next function, which gets fed into the function along with X\_2 to get Y\_2, producing an output and continuing the sequence. As you can see, there's an element of x\_0 fed all the way through the network, similar with x\_1 and x\_2 etc. 
* **What if we are looking at a long sequence of words?** We might lose context of the beginning of the sentence and are unable to understand the rest of the sentence.
* An **update** to RNNs is called LSTM, long short - term memory.
* In addition to the context being passed as it is in RNNs, LSTMs have an **additional pipeline of contexts called cell state**. This can pass through the network to impact it. This helps keep context from earlier tokens relevance in later ones so issues like the one just mentioned can be avoided. **Cell states can also be bidirectional.** So later contexts can impact earlier ones.



* How do we **implement** LSTMs in code?
  + **tf.keras.layers.LSTM**: To add the **second layer** as an LSTM.
  + Parameter passed is the **number of outputs** that we desire from that layer, in this case it's 64.
  + **tf.keras.layers.Bidirectional**: makes my cell state go in both directions. The output from the bidirectional will be 128, even though we told our LSTM that we wanted 64, the bidirectional doubles this up to a 128.

****

* + You can also **stack** LSTMs like any other keras layer. When you feed an LSTM into another one, you have to put the **return sequences equal true** parameter into the first one. This ensures that the outputs of the LSTM match the desired inputs of the next one.



* Notebooks to explore the **different types of sequence models,** see how they work and what the impact of different layer types have on training for classification.
  + [IMDB Subwords 8K with Single Layer LSTM](https://colab.research.google.com/github/lmoroney/dlaicourse/blob/master/TensorFlow%20In%20Practice/Course%203%20-%20NLP/Course%203%20-%20Week%203%20-%20Lesson%201a.ipynb)
  + [IMDB Subwords 8K with Multi Layer LSTM](https://colab.research.google.com/github/lmoroney/dlaicourse/blob/master/TensorFlow%20In%20Practice/Course%203%20-%20NLP/Course%203%20-%20Week%203%20-%20Lesson%201b.ipynb)
* **Comparison of accuracies** between the **one** layer LSTM and the **two** layer one over **10 epochs**:
  + Training curve is smoother in the two layer one. **Jaggedness can be an indication that your model needs improvement,** and the single LSTM is not the smoothest. If you look at loss, we can see similar results.
  + **What happens when we increase to 50 epochs training?** The one layer LSTM, while climbing in accuracy, is also prone to some pretty sharp dips. The two layer one looks much smoother.
* RNNs and LSTMs can help you with **text classification**. Their inherent sequencing is great for predicting unseen text if you want to generate some **(Text generation)**
* Some other **RNN types**:
  + [IMDB Subwords 8K with 1D Convolutional Layer](https://colab.research.google.com/github/lmoroney/dlaicourse/blob/master/TensorFlow%20In%20Practice/Course%203%20-%20NLP/Course%203%20-%20Week%203%20-%20Lesson%201c.ipynb)
    - The accuracy does even better than before with close to about 100% on training and around 80% on validation. But our loss increases in the validation set, indicating potential overfilling
  + [IMDB Reviews with GRU (and optional LSTM and Conv1D)](https://colab.research.google.com/github/lmoroney/dlaicourse/blob/master/TensorFlow%20In%20Practice/Course%203%20-%20NLP/Course%203%20-%20Week%203%20-%20Lesson%202d.ipynb#scrollTo=nHGYuU4jPYaj)
    - First, we use an **embedding** that is flattened before it goes into the dense. Model will have 171,533 parameters, and nice accuracy, but clear overfitting but it only takes about five seconds per epoch to train. If changed to us an **LSTM**, I'll now have only 30,129 parameters, but it will take about 43 seconds per epoch. The accuracy is better, but there's still some overfitting. If I try a **GRU** layer instead, with a GRU being a different type of RNN, and I make it bidirectional, my network will have a 169,997 parameters. My training time will fall to 20 seconds per epoch, and my accuracy is again very good on training, and not too bad on validation but again, showing some overfitting. With a **convolutional network**, I'll have a 171,149 parameters and it only takes about six seconds per epoch to get me close to 100 percent accuracy on training, and about 83 percent on validation, but again with overfitting
* With **text**, you'll probably get a bit more overfitting than you would have done with **images**. Because you'll almost always have **out of vocabulary words** in the validation data set. That is words in the validation dataset that weren't present in the training and can’t be classified, naturally leading to overfitting.
* Notebooks to explore the different types of sequence models, see what the impact of different layer types have on training for classification:
  + [Sarcasm with Bidirectional LSTM](https://colab.research.google.com/github/lmoroney/dlaicourse/blob/master/TensorFlow%20In%20Practice/Course%203%20-%20NLP/Course%203%20-%20Week%203%20-%20Lesson%202.ipynb#scrollTo=g9DC6dmLF8DC)
  + [Sarcasm with 1D Convolutional Layer](https://colab.research.google.com/github/lmoroney/dlaicourse/blob/master/TensorFlow%20In%20Practice/Course%203%20-%20NLP/Course%203%20-%20Week%203%20-%20Lesson%202c.ipynb#scrollTo=g9DC6dmLF8DC)
* Final exercise solution: https://colab.research.google.com/github/lmoroney/dlaicourse/blob/master/TensorFlow%20In%20Practice/Course%203%20-%20NLP/NLP%20Course%20-%20Week%203%20Exercise%20Answer.ipynb