**1. Recurrent neural networks for text classification**

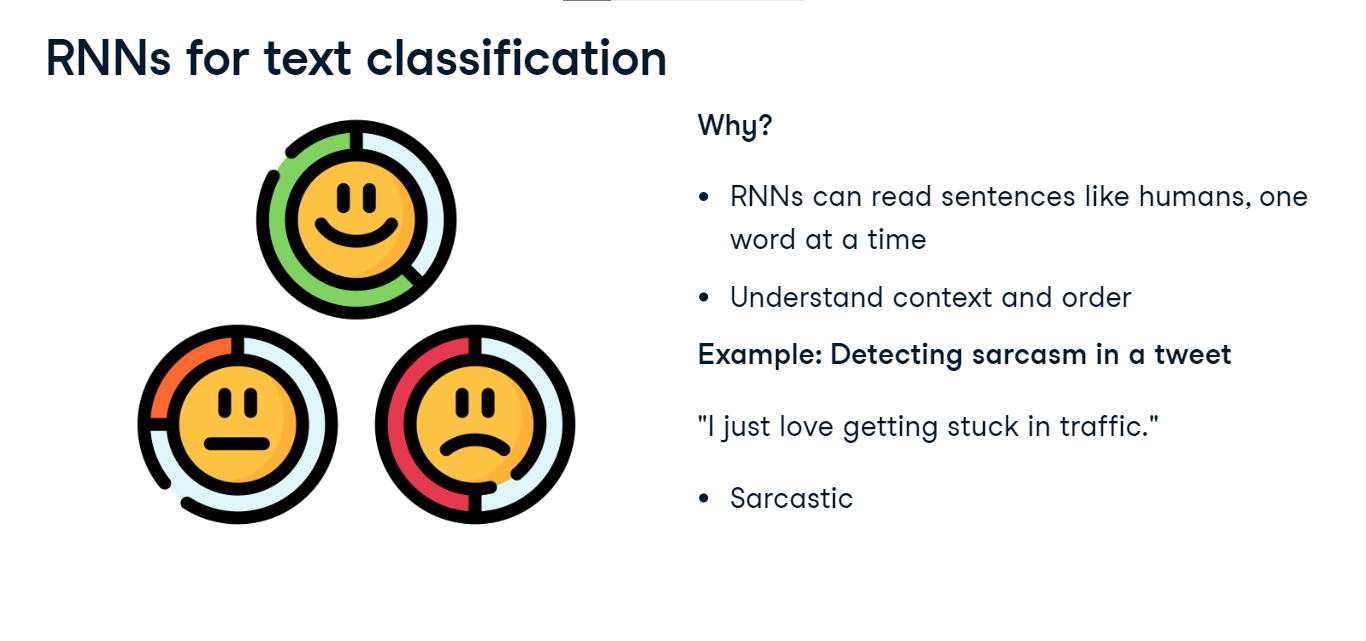
We've explored CNNs for text; now it's time to explore Recurrent Neural Networks, or RNNs, for text.

**2. RNNs for text**

Recurrent Neural Networks, or RNNs, are great at handling sequences of varying lengths. They maintain an internal short-term memory, enabling them to learn patterns across time. Unlike CNNs that spot patterns in chunks of text, RNNs remember past words to understand the whole sentence's meaning. Today, we will explore how to employ RNNs for text classification.

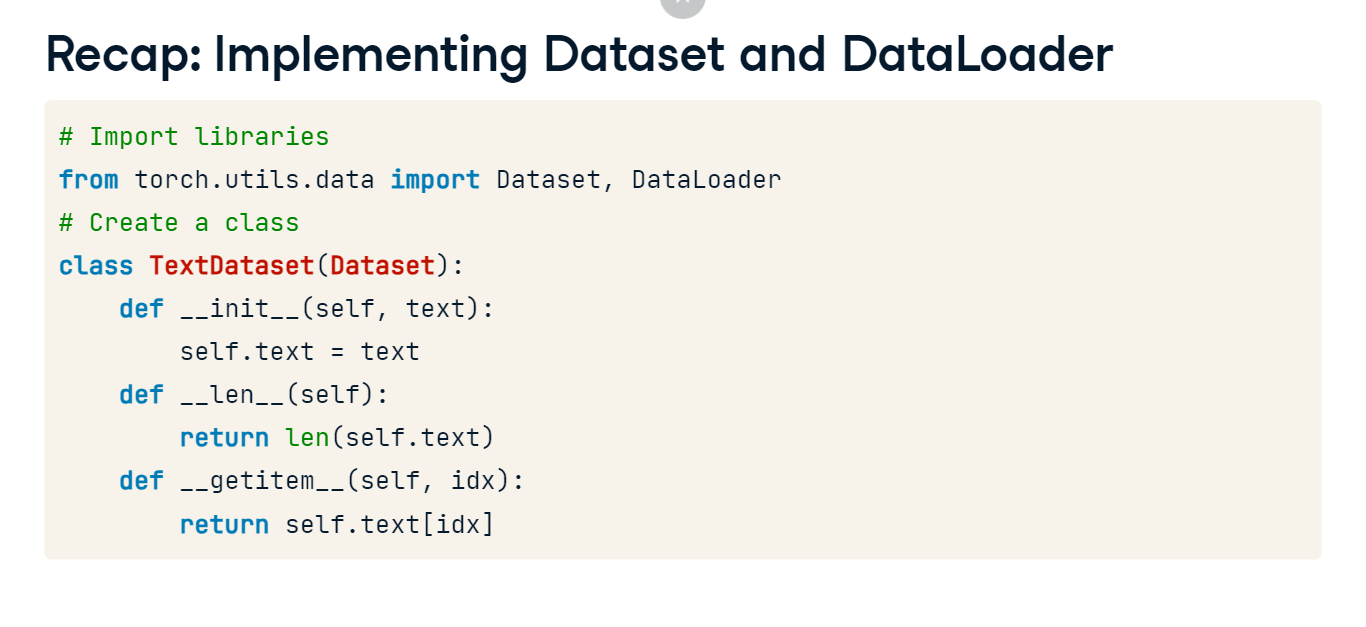
**3. RNNs for text classification**

RNNs are suitable for text classification because they process sequential data like humans read, one word at a time, allowing them to capture the context and order of words. Consider the tweet, "I just love getting stuck in traffic"; RNNs can accurately classify the tweet as sarcastic.



**4. Recap: Implementing Dataset and DataLoader**

Let's remind ourselves how to apply Dataset and DataLoader for text data in PyTorch. We create a custom class TextDataset, serving as our data container. The init method initializes the dataset with the input text data. The len method returns the total number of samples in the dataset, and the getitem method allows us to access a specific sample at a given index. This class, extending PyTorch's Dataset, allows us to organize and access our text data efficiently.



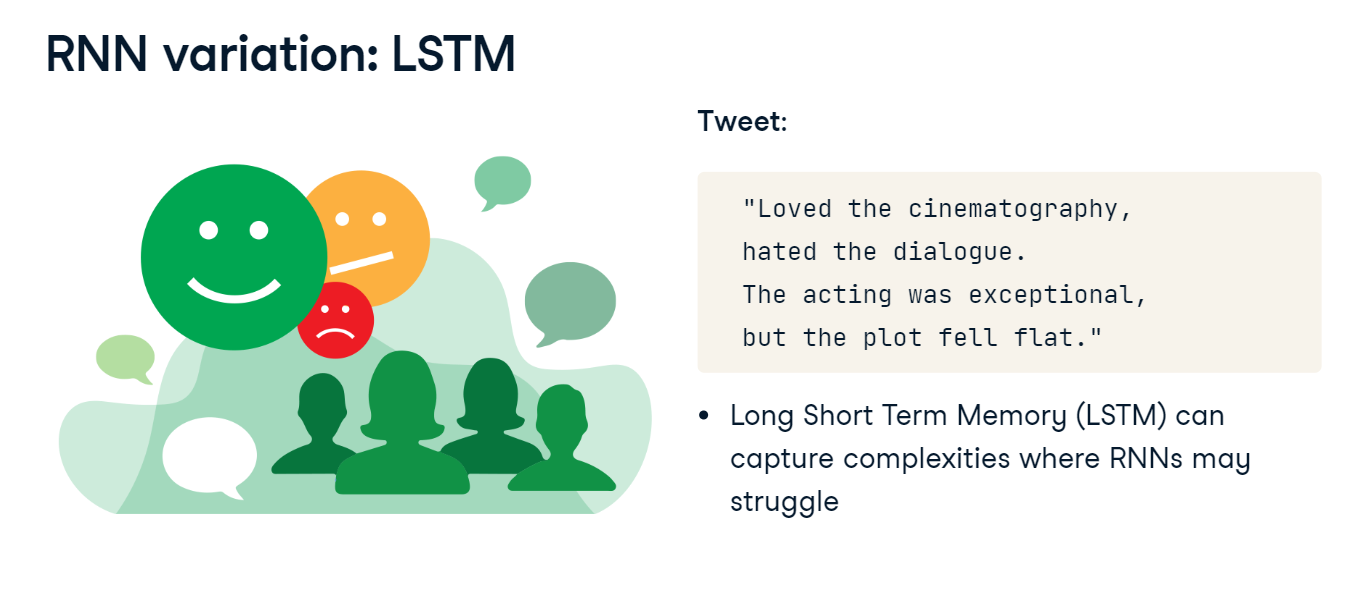
**5. RNN implementation**

Now let's take a look at an example of sentiment analysis for movie review from a tweet. We want to train an RNN model to classify movie reviews as either positive or negative. We can use our entire text processing pipeline here to feed to the model. This includes encoding or embedding. We preprocess the tweet and convert it to a tensor, which is not shown here for brevity. Then, we pass the preprocessed tensor through the model to make a sentiment prediction. In this case, the model predicts that the sentiment is "Positive."

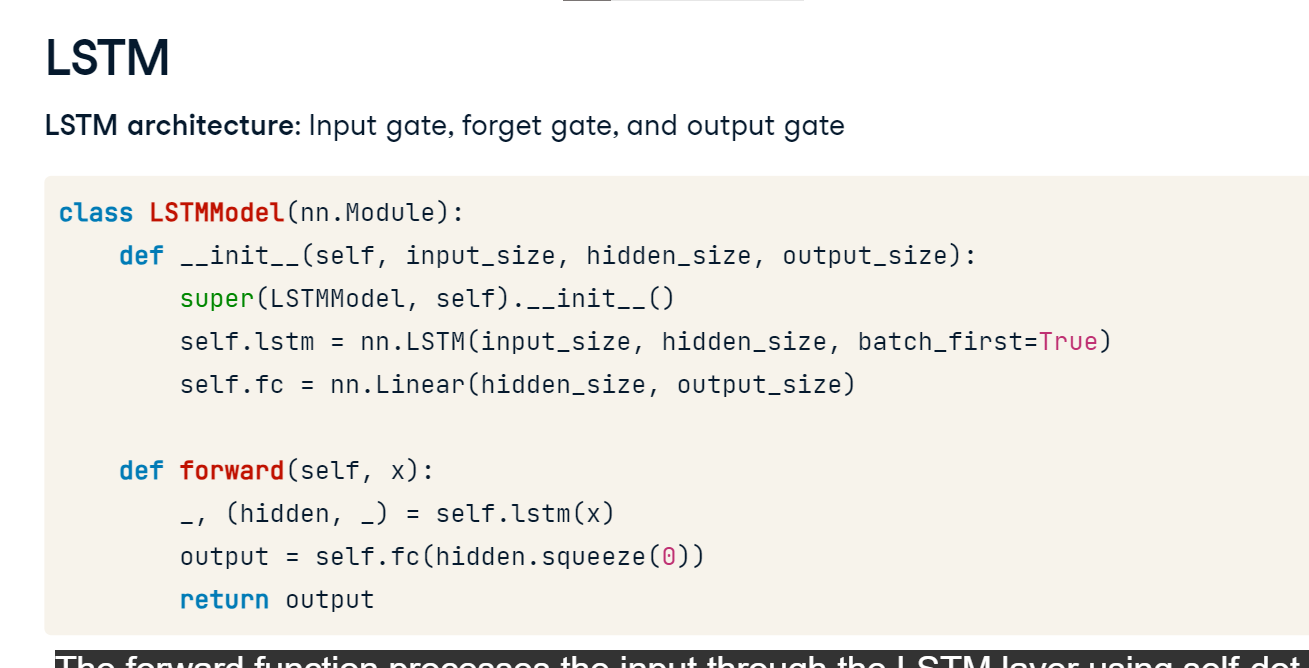


**6. RNN variation: LSTM**

But what if the tweet is not so straightforward to understand the sentiment. Take the tweet, "Loved the cinematography, hated the dialogue. The acting was exceptional, but the plot fell flat". These complex sentences contain subtle nuances and conflicting sentiments. While RNNs may struggle to capture the negative sentiment, Long Short Term Memory models or LSTMs excel at capturing such complexities. They can effectively understand the underlying emotions, making them a powerful tool for sentiment analysis.

**7. LSTM**

LSTMs have input, forget, and output gates that enable them to store and forget information as needed. This architecture is ideal for complex classification tasks. The code defines an LSTM model using nn-dot-LSTM, with an initialization function that sets the input size, hidden size, and batch-first parameter. The forward function processes the input through the LSTM layer using self-dot-lstm, and the rest is similar to RNN.



**8. RNN variation: GRU**

But, what if we wanted to detect spam emails without needing the full context. Given an email subject like "Congratulations! You've won a free trip to Hawaii!", a Gated Recurrent Unit or GRU, can quickly recognize spammy patterns without needing the full context. This makes them suitable for tasks like spam detection, sentiment analysis, text summarization, and more.

**9. GRU**

GRUs are a streamlined version of LSTMs that trade some complexity for faster training. The code defines a GRU model using nn-dot-GRU, with an initialization function that specifies the input size, hidden size, and batch-first parameter. The forward function remains the same, with the change of self-dot-lstm becoming self-dot-gru.

# 

# Evaluation metrics for text classification