$\text{tinyml}_1 2_4$

hanna$_o ndrasek$

December 2025

1 Introduction

2 Why use a recurrent neural network in the first place?

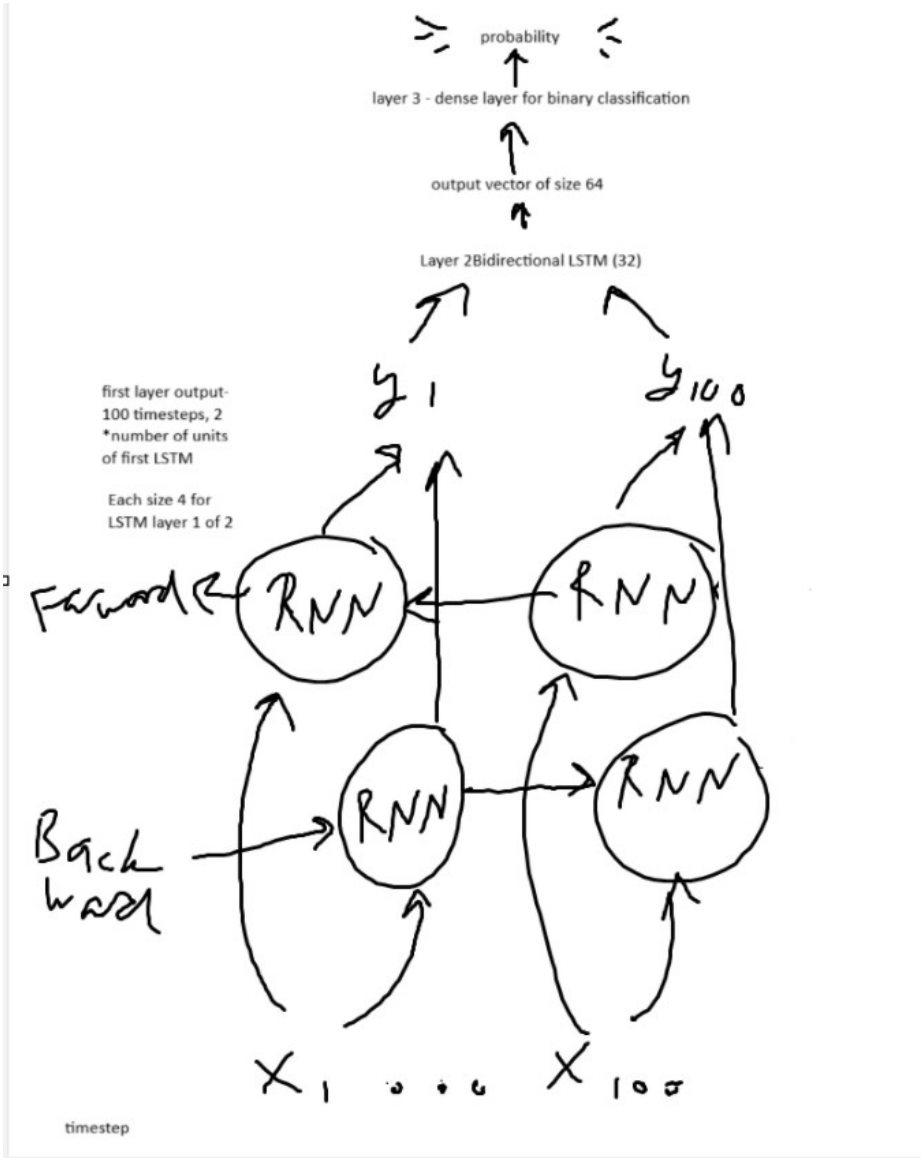
Typical neural networks pass independent data without memory, but RNNs feed information back into the network from one step to another. RNNs remember past information and use the output from one step as input to the next. In a forward-only RNN, each output depends on all earlier inputs. For classification tasks (like for this use case), the RNN uses its memory of the whole sequence to produce a single final prediction. This is achieved by using units (neurons) called Recurrent Units. Each contains a hidden state that contains information about previous inputs in a sequence, and they can use information from prior steps by feeding back their hidden state, allowing them to capture nuances across time.

3 Why I'm using a bidirectional RNN for MFCCs

Bidirectional RNN processes sequence from start to end, but also backwards, so it also gets the context around each coefficient, in simpler terms, if we were using text, it would get the context around the full text rather than just the context after (as most RNNs are forward feeding). This is good for our data since MFCCs are not really proper time series data (similarly to text)

3.1 Required shape

The input of lstm layer has a shape of (numtimesteps, numfeatures) One time step is one point of observation in the sample. One feature is one observation at a time step. https://machinelearningmastery.com/reshape-input-data-long-short-term-memory-networks-keras/ Our MFCC timestep would be 100 (for frames), the features would be 13 (13 coeffs per mfcc frame, I would consider this an observation)



4 How the model is doing

```
    Epoch 1/5
2/2  9s 1s/step - accuracy: 0.4643 - loss: 0.6974 - val_accuracy: 0.5714 - val_loss: 0.6913
Epoch 2/5
2/2  0s 138ms/step - accuracy: 0.5357 - loss: 0.6934 - val_accuracy: 0.7143 - val_loss: 0.68
Epoch 3/5
2/2  0s 127ms/step - accuracy: 0.7143 - loss: 0.6864 - val_accuracy: 0.8571 - val_loss: 0.68
```

```
Epoch 4/5
2/2  0s 206ms/step - accuracy: 0.7857 - loss: 0.6822 - val_accuracy: 0.8571 - val_loss: 0.67
Epoch 5/5
2/2  0s 175ms/step - accuracy: 0.8214 - loss: 0.6771 - val_accuracy: 0.8571 - val_loss: 0.67
```

5 Analysis

Since val loss is decreasing and loss is also decreasing, the model at least doesn't appear to be overfitting (model learns the training data too well, including random fluctuations, so it performs poorly on new, unseen data, e.g., if I were to just memorize homework problems for a test rather than understanding the concepts.)

5.1 However...

The model appears to perform poorly when I throw it a random audio file, for example, when I try to feed in a female voice or a male voice it predicts around [[0.48650196]] regardless of gender, which seems to just be random guessing. which is weird because the accuracy and validation accuracy (measures how well a model generalizes to unseen data with test set) are pretty high. also note that the accuracy varies wildly between runs of the code (fluctuates between .6 and .85)

5.2 Next Steps

Rather than using MFCCs, I may just try to do a fourier transform on each .wav audio file. I will not use the framing/windowing schema used for MFCCs. This will get the clip to the power spectrum. Get the peaks from the power spectrum, and compare using an RNN.