Connectionist Temporal Classification

The Connectionist Temporal Classification (CTc) is an algorithm used for training deep neural networks. The CTC algorithm is generally used for sequence modeling. Its applications include handwriting reconition, speech recognition, etc. In this project, the CTC algorithm will be use dto train a deep neural network to automatically convert speech to text.

Just like any machine learning task, in speech recognition, we seek a function that accurately maps the elements of the input space (audio) to the elements of the output space (transcripts). However, building a speech recognition machine differs from conventional supervised learning tasks because the inputs can vary in length. This is where the CTC algorithm comes in. The CTC algorithm is used when teh alignment pattern between the input and the output is unknown.

Specifically, for any given input (audio), the CTC algorithm gives an output distribution over all possible transcripts. This distribution can either be used to infer a likely output.

Performance evaluation

Each time we estimate the true outcome (Y) using a trained ML algorithm ($f(x)$), the discrepancy between the observed and predicted must be quantified. This quantification is done via the loss function. The loss function is a two-variable function that quantifies the loss (error) we sustain from predicting Y with f(x).

The CTC loss is a differentable function that computes the probability of the transcript given the audio; p(Y|X). The transcript that give the maximum probability is predicted. In this project, the CTC model used is similar to the DeepSpeech2 speech recognition model. The architecture consists of two convolutional layers, one recurrent neural network layer, and one dense layer. Note that as an improvement to the originally proposed model, we have used two dense layer instead of one dense layers. The breakdown of the neural network architecture is as follows;

Table 1: Deep Learning Architecture

S/N	Layer Name	Layer Description
1	Convolution Layer	32 filters of size 2 × 2 each, (ReLU), padding= same
	Batch Normalization	
2	Convolution Layer	32 filters of size 1 × 2 each, (ReLU), padding= same
	Batch Normalization	
3	Recurrent Neural Network	units=128, (tanh), recurrent_activation= sigmoid
	Bidirectional	
	Dropout	50%

4	Densely Connected	256 hidden neurons (ReLU)
	Dropout	50%
5	Densely Connected	128 hidden neurons (ReLU)
	Dropout	50%