Speech Recognition Using Transformers

Adam Aslanturk Mohammad Khoshkholgh Shaun Nothard

Jack Hawkins Kaan Kilinchan

Abstract

This report details a transformer model that has been developed and adapted through the removal of a decoder to make predictions on what number between 0 and 9 has been spoken in the English language. The model takes audio files processed using Fast Fourier Transforms and Windowing as an input to the transformer model which uses the multi-head attention architecture to recognise the speech command. The yielded results show an approximate accuracy rate of 93% during the latter epochs.

Introduction

Exploring speech recognition technology to provide a hands-free method of communicating requests while in an elevator will help solve many issues regarding. Those who are blind will not be required to see and press any button to get to the floor they desire, the spread of bacteria and viruses can be slowed down due to the lack of physical interaction with the buttons` surface, people physical impairments, and even those who simply have their hands occupied holding their phone or shopping bags will be at an advantage with an elevator that recognises speech commands.

Literature Review

As part of Machine Learning, manipulation of unstructured data has been vital towards applications that benefit from speech analysis. The majority of the world's generated data is known to be classified as unstructured data[1], increasingly making it an important area of research towards Artificial Intelligence. As voice is a primary form of communication between humans, speech recognition has an appeal that is inherent, hence, its input is preferred over other types of data. The application will make use of automatic speech recognition (ASR). The purpose of ASR is to recognise and convert audio files into other mediums. In the case of this application audio files will be converted to a numpy array to act as the input for the model. ASR systems can be used to alter noisy and fuzzy, unstructured sound frequencies generated by humans, into more meaningful text in the form of structured data[2].

Although it has high potential for being an input device, it is not simple like communicating with another person is. The choice of limited and concise vocabulary that is to be used in the application of speech as an input device demands much awareness and consideration[3].

The process of speech recognition converts sound waves of the audio into an electrical signal which is then converted into a digital signal. The signals are then presented in the form of a sequence of feature vectors. The pipeline of this system involves feature extraction, acoustic modelling, pronunciation modeling and the encoder. The application will be an example of a speaker independent platform, i.e. the application should be able to understand what a user is saying without requiring any previous training data to train off of. This requires the training data to consist of multiple voices instead of one.

Previous studies have made use of speech recognition using an encoder-decoder based sequence-to-sequence model for carrying out ASR requirements, [4] and have implemented attention mechanisms for fast and efficient training[5]

The architecture of a transformer uses self-attention for the modelling of the temporal context information to achieve much lower error rates than other models such as recurrent neural networks. A transformer consists of encoder and decoder systems, outputs of varying lengths are produced. The transformer model was developed and released by Google in 2017 [6]

The model is based on complex recurrent and convolutional neural networks with encoder and decoders. The mechanism is based on attention mechanisms, instead of recurrent layers. This is used to ensure that the model can focus on relevant details of the input while ignoring less important aspects such as noise.

Transformers are a relatively new type of neural network architecture. They were initially developed to solve the sequence transduction problem. This involves any task that may need to transform an input sequence into an output sequence [7]. This includes speech recognition.

Transformers work by using convolutional neural networks together with attention models. These increase the speed at which a model can translate from one sequence to another [6]. A transformer contains six encoders and six decoders. Each encoder will have the same architecture and will consist of two layers (self-attention and a feed forward neural network). While decoders share similar properties [8].

The encoder’s input will flow through a self-attention layer. The encoder will look at different objects in the input sequence as it encodes a specific object. The decoder also has both layers but will also have an additional attention layer that helps with focusing on relevant areas of the input sequence. Transformers go through the process of inferring which is different from training the model.[7]

The model is fed the full encoder sequence, and an empty sequence with a single filled first position is used for the decoder. This outputs a sequence where only the first element is taken. The element is filled into the second position of the decoder’s input sequence. After this both the encoder and decoder sequence are fed into the model. Take the second element of the decoder and input it into the decoder’s input sequence. This procedure is repeated until the end of the sequence. This means that the model is re-run multiple times. [6]

Dataset

The system will be trained and tested with a public dataset from Google AI. The dataset consists of a collection of 65,000 one second long audio clips of 30 short words, by thousands of people. The dataset contains appropriate commands for use in a lift, each of which is appropriately labelled. These words are one of: "Zero", "One", "Two", "Three", "Four", "Five", "Six", "Seven", "Eight", and "Nine" [9].

The audio files were collected in several locations from around the world. These audio files represent examples of speech data where there may be no say in what recording equipment is used or where the speech was recorded. As such there is a range of different qualities of audio. The audio was captured in several formats including Ogg Vorbis encoding for the web application, after which it was converted to a 16 bit little endian PCM-encoded WAV file at a 16000 sample rate [10]. All audio files were trimmed to be one second in length with the use of the extract loudest section tool. All files were then screened in order to remove files that may contain silence or incorrect words. The dataset comes with a pre-defined training, testing and validation list which details which samples will be used for training the model and which will be used for validation [11].

Design Implementation

The chosen dataset was required to undergo pre-processing to convert the audio files into a format that's appropriate for use within the model.

Pre-Processing

The preprocessing of the system required the conversion of the raw audio samples into a format that is appropriate for use within the transformer model, illustrated in figure 1.

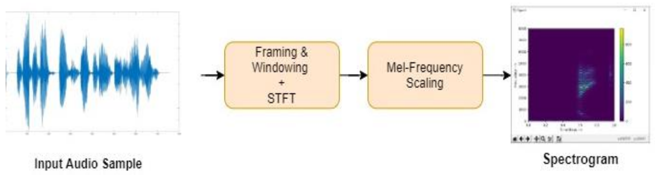

Figure 1: Audio Processing

The initial step of the process is to convert each audio sample, provided as a waveform, to a spectrogram. Further processing is required to make the data appropriate for the transformer. The first step is to make use of a technique called windowing which reduces the amplitude of discontinuities at the end of each boundary. (REF) The process of windowing consists of multiplying the waveform by a finite-length window with an amplitude that graduates towards zero at the edges. The Hanning window touches zero at both ends of the waveform, which eliminates any discontinuities. This is appropriate for our data since the peak generally consists around the middle mark of the waveform. Originally we applied Fast Fourier Transforms [12], but this approach produced INF/Nan values due to divide by zeros on the dataset which our algorithm implementation did not handle well. To overcome this, we applied Short Time Fourier Transforms, which evaluates the Fourier Transforms over a short window providing information regarding the fluctuation of frequencies over time. Not only did this fix our error with INF values, but it removes a large amount of noise in the data, which allows the critical features to be prominent, leading to higher accuracy in the model. Figure 2 shows a comparison between using these two transform functions.

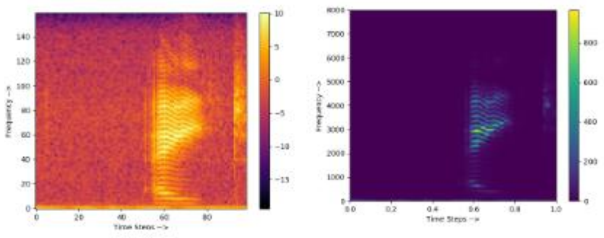

Figure 2: Noise Reduction

A few other issues were encountered during this phase. The first being that not all files had a duration of 1 second as expected. To overcome this, zero-padding was introduced to ensure all clips were uniform when inputted into the encoder. The signal.stft() function is used to obtain the stft for each sample using the hanning window. The results of this process are all normalised. Finally, Mel-Frequency scaling is applied to this data which is transposed to get frequency values by time. Once Mel scaling has been applied the absolute values are extracted to get the raw magnitude of each frequency bin, discarding the imaginary values generated as they negatively affect the accuracy of the model. The Spectrogram is now ready to be processed.

The dataset is then split into training, testing and validation sets using a pre-computed list to ensure consistent training in each run. Each of these sets is then converted to a dictionary item, with the key of the dictionary representing the label and the values representing the values.

These dictionary values are then split into two further numpy arrays. One array for the data values of one for their respective labels. This is done for both the training and testing sets.

Each label in the training and testing sets will then go through further encoding. Label encoding is used to encode the target values to each sample. One-Hot Encoding is used to create a binary column of each category in the sample. Returning a matrix/density array. The dataset is now able to be used within the model.

Multi-Head Attention

Multi-head attention was implemented which consisted of taking three inputs, a query, a key and a value that is derived from a simple attention function. Each input is then parsed through a linearity layer separately in parallel before being fed into the scaled dot product attention layer for further computations (as shown in figure x). This enables the model to obtain information at multiple positions from various representational subspaces ultimately further reducing the overall dimensionality (curse of dimensionality).

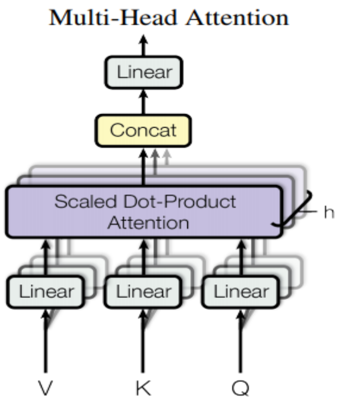

Figure 3: Multi-head attention architecture with inputs: Value (V), Key (K) and Query (Q)

The multi-head attention's architecture consists of four core parts: the linear layers and splitting into heads, the scaled dot product attention, concatenation of heads and the final linear layer.

The scaled dot product is applied to each head. An appropriate mask is applied to each attention step. The attention output for each head is then concatenated with tf.transpose and tf.reshape which is then parsed through a final dense layer.

Q, K and V are split into multiple heads as it allows the model to attend information at different positions from different spaces. After each split each head will have a reduced dimensionality, which has a total computation cost which is the same as a single head with full dimensionality.

Scaled Dot Product

Scaled Dot Product, which is a core aspect of multi-head attention, was utilised as part of the system rather than using standard additive attention, as it offers a more cost efficient approach to computations for the output of matrices using the formula:

$$\mathrm{Attention}(Q, K, V) = \mathrm{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

Equation 1: Attention Function [6]

The dot product attention is scaled by the square root of the depth. This is due to the large values of depth, which results in the dot product growing large which pushes the softmax function in areas where small gradients are present. This results in a hard softmax.

The mask is multiplied by a value that is close to negative infinity. This is done to ensure that the mask is summed with the scaled matrix multiplication of Q and K. This is done before the softmax. The purpose of this is to zero out these cells and to have large negative inputs to softmax which are all near to zero in the output.

Positional Encoding

As the model contains no recurrence or convolution, due to the use of stacked attention layers instead of CNN/RNNs, a separate class is required to perform positional encoding. This provides the model with information about the relative position of the input values. This is needed as the model contains no built in information of the positions of items in a sequence.

The model makes use of linear projection which is a linear transformation such that whenever the transformation is applied to a value twice, the same result is always achieved as if it was only applied once. Adding positional encoding will ensure that values will be closer to each other, based on their similarities and their position in the d-dimensional space.

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{model}})$$

Equation 2: Formula for calculating positional encoding [6]

The formulas for calculating the positional encoding. The encoding is a vector of sines and cosines at each position, where each pair will rotate at a different frequency. Similar locations will have similar position-encoding vectors.

The calculated angle rates occur in the 1rads/step range to the minimum rate rads/step. The resulting exponent will range between 0 and 1, which will cause the angle rates to drop from 1 to the minimum rate.

Broadcast a multiply over angle rates and positions in order to obtain a map of the position of all encoding angles as a function of the depth. The raw angles aren't a good model input as they are unbounded or discontinuous so only the sine and cosine are used.

Masking

The STFT and Mel function produce fully zero time steps at the start and end of each value. This allows for the model to not recognise the padding. The use of masking reduces the chance that the model will recognise it as valid data during the training phase.

Transformer

The implemented model follows a similar architecture to a traditional model with the main difference being the exclusion of the decoder. A decoder was not implemented as the system involves sequence to labelling transformations rather than sequence to sequence format frequently seen with this model architecture.

Furthermore, the output of the top encoder layer is usually transformed into a collection of attention vectors, which are used by the decoder during the encoder-decoder layer to help focus on segments of the input sequences which are relevant.

Since we are passing in labels as the other input into this encoder-decoder layer, our input data into the decoder was not the same shape. This meant that it was not possible to include the decoder segment of a transformer. Thus we needed to look into other options for improving the accuracy of the model.

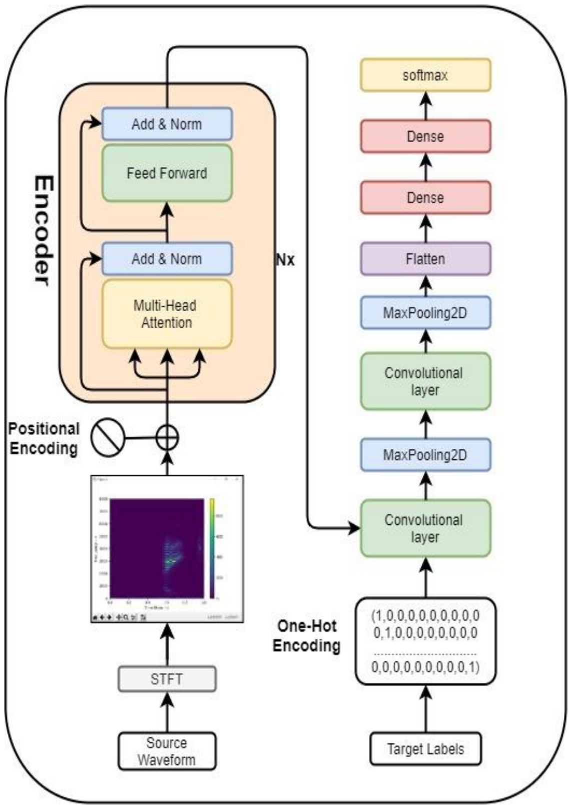

Figure 4: Model Architecture

Figure 4 illustrates the final model architecture used for this project. It consists of an encoder and a fully connected convolutional network. Each encoder layer consists of the multi head attention with added padding mask and point wise feed forward networks.

A point wise feed forward network consists of two fully connected layers each of which has a ReLU activation in between them.

Each sublayer contains a residual connection followed by normalisation of the layer. The residual connections help with the avoidance of the vanishing gradient problem that is present in deep networks.

The output of each sublayer is the normalisation of the model's axis plus the sublayer of the same model axis. The transformer has a total of four encoder layers.

The encoder contains the linear projection, positional encoding and the 4 encoder layers. The input of the encoder goes through projection which is summed with the positional encoding. The output of the preparatory functions acts as the input for the encoder layers. The output from the encoder layer is passed into a series of convolutional layers alongside the one hot encoded labels. Using convolution layers allows the model to build up a series of activations which result in a feature map. This feature map indicates the location and strength of a feature that was detected. By applying a flattening layer, we remove all dimensions except for one so that it can be processed by the dense layer, which requires a feature vector. These Dense layers then act as classifiers before passing the final softmax layer which converts the output to a probability distribution.

With the use of additional libraries we can visualise the layers of the model to better understand it, which can be seen in Figure (5). This allowed us to confirm that the model is connected the way we intended with sequential layers working as expected. It also ensures that the shapes of our input data are correct.

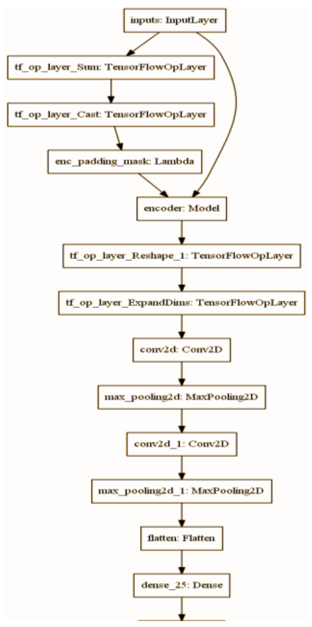
inputs: InputLayer
tf_op_layer_Sum: TensorFlowOpLayer
tf_op_layer_Cast: TensorFlowOpLayer
enc_padding_mask: Lambda
encoder: Model
tf_op_layer_Reshape_1: TensorFlowOpLayer
tf_op_layer_ExpandDims: TensorFlowOpLayer
conv2d: Conv2D
max_pooling2d: MaxPooling2D
conv2d_1: Conv2D
max_pooling2d_1: MaxPooling2D
flatten: Flatten
dense_25: Dense

Figure 5: Model summary

Training

Hyperparameters	**Value**
Num_Layers	4
D_Model	128
Num_Heads	8
Units	2048
Dropout	0.1
Time_Steps	100
Output_size	10

Table 1: Hyperparameters of the model

The hyperparameters of the model were set to the values depicted in table 1. These include the number of layers, model dimension (d_model), units and the number of heads. The d_model and time_steps are set to fit the shape of our input data. The units parameter defines the size of the feedforward neural network.We found that a dropout rate any higher than 0.1 negatively impacted the model, decreasing its accuracy. This is likely due to dropout potentially removing all audio from the sample on higher settings. The output parameter is defined to match the number of labels that we are passing into the next layer of the model. The Adam optimiser is used to optimise the model.

$$lrate = d_{model}^{-0.5} * min(step\_num^{-0.5}, step\_num * warmup\_steps^{-1.5})$$

Equation 3: Formula to calculate learning rate [6]

Different values of the learning rate were tested, and it was concluded that 0.0001 was best for the model. Anything higher causes the model to not converge, anything lower causes the model to overfit.

Results and Evaluation

Two metrics were used to evaluate the generated model. These were the model accuracy (ratio of correct predictions against the total input) and the model loss (summation of the errors made for the calculation of a prediction for a sample).

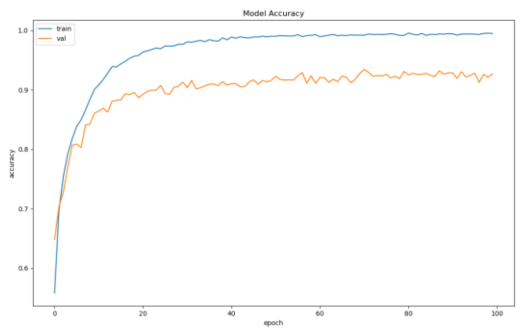

Figure 6: Model Accuracy for training and validation sets

The accuracy of the model is shown to be relatively high. During the training the model the total accuracy was shown to near 1.0 towards the later stages of the training phase. The accuracy of the validation phase was shown to be less than that of the training phase. The accuracy of the validation phase was shown to somewhat fluctuate during the process. The maximum value of the accuracy that was gained during this process was approximately 0.93.

One aim of the fine tuning of the model was to get the model loss as close to zero as possible. As a loss of zero indicates a perfect prediction.
The model loss for the training set was shown to continuously decrease with a final minimum value of 0.04, indicating that while not perfect the predictions made with the training set were accurate.

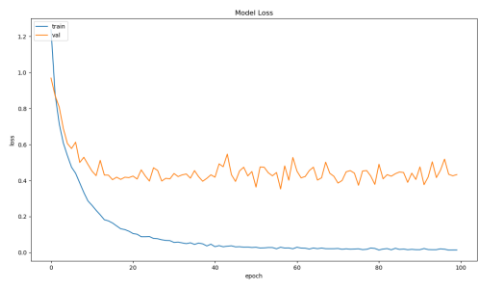

Figure 7: Model loss for training and validation sets

The validation set was shown to have a higher loss on average than that of the training set. The loss for the validation data was shown to somewhat fluctuate during the multiple phases of the model. With the final model loss at the end of the 100th epoch being around 0.45. The overall minimum model loss across the whole model was around 0.39 for the 70th epoch. This indicates that although not perfect the model does seem to be able make predictions on unseen data relatively accurately with few mistakes being made.

Overall the model was shown to be relatively accurate as shown by the high accuracy and the minimal model loss, showing that a majority of the unseen data fed into the model was labelled correctly. However the model is shown to not be perfect and there is a possibility for unseen data to be labelled incorrectly.

Conclusion

In this project, a transformer was adapted from the traditional model through the exclusion of a decoder. This was used to develop a system that was capable of predicting what number, between 0 and 9, has been said in an audio sample. The audio samples were initially pre-processed with the use of FFT and Windowing, as well as masking, and positional encoding to appropriate the data for use in the transformer model.

The transformer also utilises several different architectures that are present within it. This pertains to the use of multi-head attention, Scaled Dot Product and feed-forward networks.

The model was routinely tuned to gain the hyperparameters which resulted in the final implementation having its highest possible model accuracy and lowest possible model loss.

The completed model was shown to have a validation accuracy of approximately 90-93% and a minimal model loss after the initial epochs were computed.

Acknowledgments

[1] W. Li and B. Lang, ‘A tetrahedral data model for unstructured data management’, *Sci. China Inf. Sci.*, vol. 53, no. 8, pp. 1497–1510, Aug. 2010, doi: 10.1007/s11432-010-4030-9.

[2] K. Adnan and R. Akbar, ‘An analytical study of information extraction from unstructured and multidimensional big data’, *J Big Data*, vol. 6, no. 1, p. 91, Oct. 2019, doi: 10.1186/s40537-019-0254-8.

[3] A. Caranica, H. Cucu, A. Buzo, and C. Burileanu, ‘On the Design of an Automatic Speech Recognition System for Romanian Language’, *Control Engineering and Applied Informatics*, vol. 18, pp. 65–76, Jun. 2016.

[4] N. Moritz, T. Hori, and J. L. Roux, ‘Streaming automatic speech recognition with the transformer model’, *arXiv:2001.02674 [cs, eess, stat]*, Mar. 2020, Accessed: May 25, 2020. [Online]. Available: https://arxiv.org/pdf/2001.02674.pdf

[5] L. Dong, S. Xu, and B. Xu, ‘Speech-Transformer: A No-Recurrence Sequence-to-Sequence Model for Speech Recognition’, in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Calgary, AB, Apr. 2018, pp. 5884–5888, doi: 10.1109/ICASSP.2018.8462506.

[6] A. Vaswani *et al.*, ‘Attention Is All You Need’, *arXiv:1706.03762 [cs]*, Dec. 2017, Accessed: May 25, 2020. [Online]. Available: http://arxiv.org/abs/1706.03762.

[7] ‘What is a Transformer? - Inside Machine learning - Medium’. https://medium.com/inside-machine-learning/what-is-a-transformer-d07dd1fbec04 (accessed May 25, 2020).

[8] ‘How Transformers Work - Towards Data Science’. https://towardsdatascience.com/transformers-141e32e69591 (accessed May 25, 2020).

[9] ‘Launching the Speech Commands Dataset’, *Google AI Blog*. http://ai.googleblog.com/2017/08/launching-speech-commands-dataset.html (accessed May 25, 2020).

[10] ‘TensorFlow Speech Recognition Challenge’. https://kaggle.com/c/tensorflow-speech-recognition-challenge (accessed May 25, 2020).

[11] P. Warden, ‘Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition’, *arXiv:1804.03209 [cs]*, Apr. 2018, Accessed: May 25, 2020. [Online]. Available: http://arxiv.org/abs/1804.03209.

[12] ‘Understanding FFTs and Windowing.pdf’. Accessed: May 25, 2020. [Online]. Available: https://download.ni.com/evaluation/pxi/Understanding%20FFTs%20and%20Windowing.pdf.

Group Contributions

As a group, every team member was involved to some degree with every aspect of the project. Some members were more invested in the coding side of the project while others focused on the write-up, but everyone was aware of the changes being made. A quick summary of team roles:

- **Shaun**: Project Leader, Primary Role: Implementation, Secondary Role: Report Writer.
- **Jack**: Primary Role: Implementation, Secondary Role: Report Writer.
- **Adam**: Primary Role Report Writer & Research, Secondary Role: Implementation.
- **Kaan**: Primary Role Report Writer & Research, Secondary Role: Implementation.
- **Mohammad**: Primary Role Report Writer & Research, Secondary Role: Implementation.

This structure worked well as us for a team. Throughout the lifecycle of this project, we had regular calls to discuss the code changes and progress made within the report.