Classifying a finite number of human intents from voice with slight negative latency, and solving the use mention distinction

Isaac Leonard ifleonar@us.ibm.com, Chris Dye dyec@us.ibm.com

Abstract

We present Aural2, a data collection and labeling infrastructure capable of training an LSTM model to use voice to classify the action which a user wished to be performed. It is usually capable of correctly classifying an intent before the user has finished speaking; assuming latency to be measured from end of utterance, Aural2 has slight negative latency. Furthermore, Aural2 will automatically learn to integrate past context into its classification, allowing it to learn to accurately solve the use-mention distinction[1], negating the need for wake-words.

We describe the architecture of Aural2, its advantages over existing systems, its failings, and future directions for development.

Introduction

It is useful for users to be able to control machines via voice. To do this, the machine must be able to turn sound into one of the finite number of actions which the machine is capable of performing. Most voice command systems use word level natural language parsing (NLP) systems operating on top of speech to text (STT)[2][3]. As a series of words contains less information then the audio it was transcribed from, purely from an information theory perspective a system which transforms sound directly into intents can be more accurate than word level NLP on top of speech to text (STT).

[1]Use-mention distinction: https://en.wikipedia.org/wiki/Use-mention_distinction http://shomir.net/pdf/publications/swilson_cicling11.pdf

[2]https://developer.amazon.com/docs/custom-skills/define-the-interaction-model-in-json-and-text.html

[3]https://people.mpi-inf.mpg.de/~smukherjee/Intent-Classification-WWW-2013.pdf

Technologies used

TensorFlow Compute Graph (TF graph)

TensorFlow (TF) Compute Graph is a purely functional language for defining graphs of transformations on tensors, which the TensorFlow runtime lazily evaluates using the best hardware available at runtime. Each node in the graph takes zero or more tensors as input, and returns one or more tensors as output. As recursion and loops are forbidden, TF compute graphs are provably halting, execute in approximately fixed time, and are not Turing complete. A TF graph can be stored as a GraphDef protobuf file. This GraphDef is cross platform, able to be evaluated by the TensorFlow runtime on any supported hardware, whether that be an x86 or ARM CPU, or NVIDIA GPU.

A tensor in this context is an n dimensional array of numbers. The shape of a tensor is denoted with a list of its dimensions. For example, [1, 2, 3] denotes a three dimensional tensor; a list containing one list of two lists of three numbers. All numbers are float32 unless otherwise noted.

Although it is possible to write a text encoded GraphDef by hand, it is far more common to construct the graph using some more general purpose language such as Python[4] or Go[5], either for export as a GraphDef for later use, or for immediate evaluation.

Aural2 does both: constructing the main training graph in python at build time, and the numerous other supporting graphs in Go at initialization time.

Aural2's extensive use of TF compute graphs allow it to take advantage of dedicated hardware accelerators such as NVIDIA GPUs, while still running with full capabilities, albeit somewhat slower, on a generic CPU.

Mel-frequency cepstral coefficient (MFCC)

It is computationally expensive to train a neural net directly on the waveform of audio[6]. Therefore, it is common practice to train the NN on fingerprints of windows of the waveform[7]. A Fourier transform of a window of audio reduces it to the a list of amplitudes in the frequency domain. Mel-frequency cepstral coefficient (MFCC) remaps the frequency domain information produced by a Fourier transform to a scale optimized for human speech. In the configuration used in Aural2, MFCC uses 13 frequency bins each represented as a float32.

[4]Python, an open source programming language maintained by the Python Community https://www.python.org/

[5]Golang, an open source programming language developed by Google: https://golang.org/

[6]https://github.com/buriburisuri/speech-to-text-wavenet

[7]https://arxiv.org/pdf/1305.1145.pdf

Long short-term memory neural nets (LSTM)

An LSTM can be thought of as a Recurrent Neural Net (RNN) augmented with persistent memory[8].

Each cell of a RNN takes as input the output of the previous cell concatenated with the current state of the world and returns an output which is sent to the next cell. This allows them to recognise patterns in time series data of arbitrary length. However, for reasons[9], RNNs have difficulty remembering long term state. LSTMs solve this problem by augmenting an RNN with persistent memory which it can read from and write to, thereby allowing it to persist information for arbitrarily longer periods.

Aural2 currently uses a stack of two LSTMs, the first taking as input the series of MFCCs of audio, and the second taking the output of the first and producing an embedding of the user's intent. Both LSTMs have a state of size 64. As each LSTM is passing both the RNNs information and the state of its memory foreword, a total of 256 float32s are being passed forward in each iteration.

Architecture

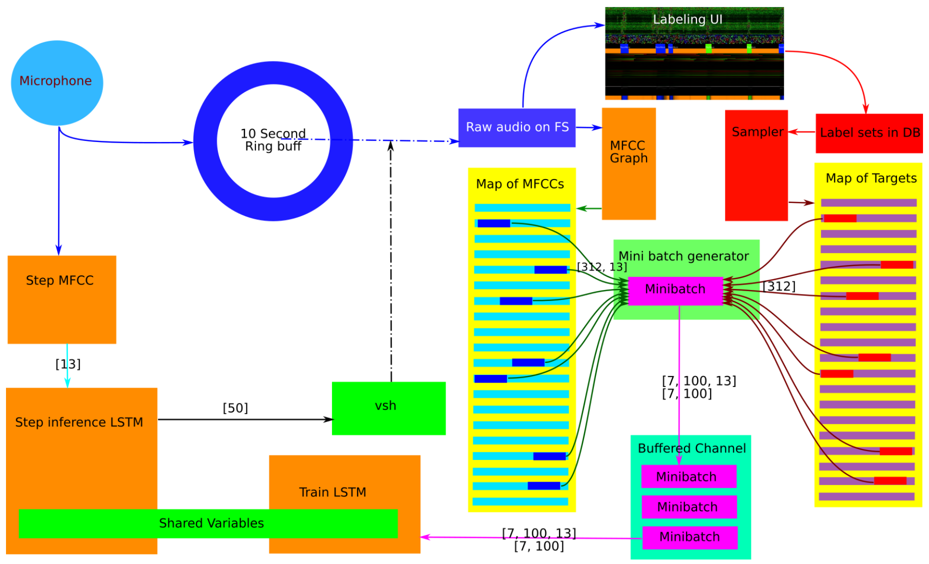

Figure 1: Architecture of Aural2

The primary TF compute graphs used by Aural2 are as follows.

[8]Recurrent Neural Networks: https://en.wikipedia.org/wiki/Recurrent_neural_network https://karpathy.github.io/2015/05/21/rnn-effectiveness/

[9]https://en.wikipedia.org/wiki/Vanishing_gradient_problem

- Step MFCC: Takes 1024 bytes of int16 PCM. Returns a [13] tensor.
- Clip MFCC: Takes 160,000 bytes of int16 PCM. Returns a [312, 13] tensor.
- Step inference LSTM: Takes a [256] state, and a [13] input tensor. Returns a [50] one-hot[10] output, and a final state of [256].
- Train LSTM: Takes a [7, 100, 13] input tensor and a [7, 100] int32 target tensor. Updates the weights and biases when evaluated.

Note that the step inference and training LSTM graphs share weight and bias variables.

There are also various TF graphs for generating visualizations of data. These graphs will not be discussed in detail here.

Sound is recorded at a sample rate of 16,000Hz with 16 bit depth. 512 sample windows are read and, both written to a ring buffer and fed into a TF graph to compute the MFCC, producing a tensor of shape [13]. This tensor is used as the input to the one or more inference LSTM graphs. The output of the LSTM, once `matmul`ed, and `softmax`ed is a list of 50 floats between 0 and 1. The nth element of the output is the probability that the world is in state n. As the world can be in one and only one state, the probabilities of the various states always sum to 1.

Training

Data collection

As mentioned before, Aural2 maintains a ring buffer of the past 10 seconds of audio. At any time, the past 10 seconds can be written to disk. This may be triggered by a REST API, or by the user saying "upload", "mistake", or otherwise expressing their intent that the audio should be saved. Clips of raw audio are stored as files in a directory on the local storage. Metadata about the raw audio clip is stored in a local boltDB[11] or other key/value database.

Labeling

Neural nets transform one dataset into another dataset. To use supervised learning to train the neural net to turn the input data into the *correct* output data, we must give Aural2 many examples of the correct outputs for the various inputs. To help the user create this information, Aural2 provides a web based labeling UI.

Aural2 serves an index page listing the audio clips which have been captured, each clip name linking to the labeling UI for that clip. The labeling UI for a given clip contains various visualizations of the clip and the labels made by the current

[10] One-hot embedding: https://en.wikipedia.org/wiki/One-hot
[11] BoltDB: https://github.com/boltdb/bolt

model when processing it. The visualizations of the labels are automatically reloaded every ~second, allowing users to watch the output of the model change in real time. The user may listen to the audio, see visualizations of it, and create labels defining the beginning and end of each period of any given state. Once the user has labeled all periods in the clip during which a non nil intent was being expressed, the label set is submitted back to the server which both writes it to the local DB, and adds both the label set and the corresponding audio clip to the training data object.

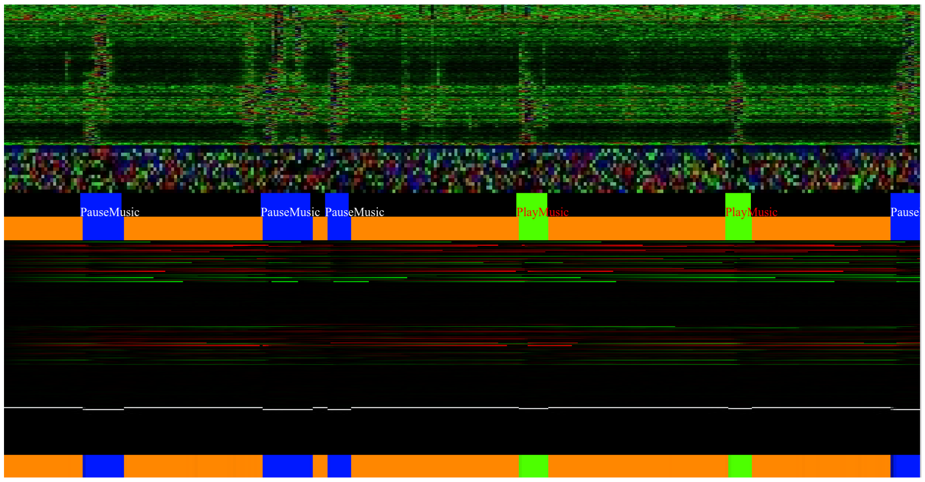

Figure 2: Labeling UI and audio visualizations

Training

The training data object contains two maps, one of inputs and one of targets, where each input is of shape [312, 13] and type float32 and each target is of shape [312] and type int32.

When Aural2 starts, it reads the list of label sets from the boltDB and the audio clips to which they refer, and adds them to the training data object.

When a label set and clip are added to the training data object, the clip is fed into the clip MFCC TF graph to transform it to a tensor of shape [312, 13], which is added to the inputs map, and the label sets are transformed into a list of the integer state ID at each of its 312 time steps, which is added to the targets map.

In this way, a set of preprocessed inputs and targets is created from existing training data on startup, and added to when new labels are submitted.

However, these are sequences with a length of 312 steps. To train an LSTM on sequences of n time steps, one must unroll the cells, creating a training graph containing n copies of the LSTM cell. It is computationally expensive and unnecessary to train on 312 sample sequences. Aural2 currently trains on 100 sample long sequences.

The data preparation loop is as follows:

- From the set of labeled audio clips, randomly select 7.
- For each clip:
  - At random, select two positive integers such that the second is less than 312, and exactly 100 more than the first.
  - From both the inputs and the targets for this clip, take the slice of 100 time steps defined by the two numbers previously selected.
- We now have 7 inputs of 100 time slices, and 7 corresponding targets of 100 time slices.
- Convert these two sets to tensors, a float32 input tensor of shape [7, 100, 13] and an int32 target tensor of shape [7, 100].
- Write this input-target pair to the minibatch channel, blocking until there are fewer than three mini batches in the channel.

The training loop reads a mini batch from the mini batch channel and evaluates the training LSTM graph on the inputs and targets, thereby updating the weights and biases.

In this way, the training loop is always supplied with a buffer of mini batches randomly drawn from a recent state of the training data, and training data preparation is free to hog CPU resources while the train loop is blocked by the GPU doing training.

It should be reiterated that the graph used for performing inference on incoming audio, the graph for performing batch inference on whole audio clips for visualizations, and the training graph, while distinct graphs, share a single set of variables. The weight and bias variables are updated by the training graph, and the accuracy with which aural2 classifies the state of the world increases. The variables stay on the GPU, or whatever compute device TensorFlow has decided to use, and need never leave. TensorFlow transparently handles locking to ensure that the various graphs can read and write to the shared memory safely.

Results

The neural net used by Aural2 is continuously trained in real time from a dynamically collected training set. The distribution of data added to the training set changes depending on the environment and user, which depend on the state of the Aural2 model. In particular, the data are heavily biased towards the sounds before and during sounds which the model thinks are the user telling it to save audio, although because of this bias in the training data, false positives for the “saveAudio” intent drop to near zero quite quickly.

There is no standard training or test set for Aural2; each user is encouraged to generate their own training set, and to keep saving and labeling clips until Aural2 stops making too many mistakes. It is therefore difficult to provide numbers describing Aural2s accuracy in a reproducible manner.

The closest systems to which we can compare Aural2 are the Google Home and the Amazon Echo. Like Aural2, the Google Home and Echo respond to voice commands to perform such actions as playing or pausing music. However, unlike Aural2, they are unable to perform use-mention differentiation and therefore require the use of a wake word not common in everyday speech. Additionally, the only command which they are capable of detection locally is the wake word; for all other speech processing, they must send audio to the cloud and receive the results. This is a privacy issue as well as incurring high latency. Contrast this behavior to Aural2 which processes all audio locally, and can therefore respond in time limited only by local compute speeds. As the LSTM used by Aural2 is trained to label the whole duration of the utterance, it will usually begin outputting the intent before the utterance is finished; a whole utterance is unneeded to correctly classify an intent.

As Aural2 is trained locally, its model is trained on the user’s voice recorded from actual usage. There is no need to collect a diverse training set; what would in any other model be overfitting to a single user or small group of users is Aural2’s correct and non-problematic behavior.

An advantage of sending audio to the cloud for conversion to an intent is that it allows such a system to use models running on servers in the cloud, which can be far larger than any model suitable for running on cheap edge devices.

Additionally, requiring a wake word has various advantages. As good at Aural2’s use-mention distinction is, it will eventually make a mistake. However, if additional safely is desired, it is simple to train Aural2 to require that some prefix, “sudo” for example, be said before particularly dangerous intents. Aural2 need only be taught that when the user says “format disk”, the user does not intend that the disk be formatted, whereas when they user says “sudo format disk”, the user does want the disk to be formated. Aural2 will usually learn the distinction with a few examples. For less dangerous intents, no prefix need be required. In this way, the benefits of requiring a wake word may be easily gained when desired, while still preserving the ease of non-prefixed commands where very occasional false positives are not so harmful.

We invite readers to download Aural2 at <insert link> so as to evaluate it for themselves.

Shortcomings

Although perhaps superior to existing technology in latency, simplicity, and speed of training, Aural2 currently uses one-hot embedding which scales linearly with number of outputs. While the currently used embedding size of 50 intents is fully sufficient for many tasks such as controlling music, interacting with simple toys, or as a safety stop for industrial equipment, it would likely be impractical for many thousands of outputs, and as such, can have no ambition for use in full vocabulary natural language parsing.

Additionally, Aural2 leaves much to be desired with regard to the labeling of training data. While it can save audio on command, this merely helps to collect unlabeled audio rich in states which the user thought a past state of the model had misclassified; it does nothing to label the audio with the true state.

A significant improvement to Aural2 would be to make use of user feedback to directly train via reinforcement learning. This would require an additional model to classify user voice, facial expressions, etc, into an emotional state. This hybrid system of training an emotion classifier via supervised learning, which can then be used to train an intent model via reinforcement learning is however somewhat inelegant.

Another potential improvement would be to use a pair of one-hot embedded outputs, one for verbs and another for nouns. Whereas the current Aural2 outputs the action which the user wishes the machine to take, a dual embedding would allow Aural2 to classify arbitrary pairs of action and thing to which the action is to be applied. This would allow a significantly larger portion of the space of commands to be represented by the output.