Learning with Sound Data

Tom Södahl Bladsjö
LT2326

Abstract

While most natural language processing focuses on text data as its examples of natural language, spoken language is arguably a more "natural" setting for human interaction and, as such, extra interesting to me as a linguist. However, the continuous nature of a sound signal makes it, in some ways, more challenging to process than written language, which can be segmented more easily into discrete tokens that look the same regardless of who the speaker is. In this project, I attempt to get a more thorough understanding of audio data, including preprocessing and augmentation techniques, as well as model architectures used to learn from this type of data. While I do not manage to successfully perform speech recognition, I do experiment with different types of audio-related machine learning tasks and get a lot of valuable coding practice.

1 Introduction

Most natural language processing focuses primarily on text data. However, a lot, if not most, human interaction in real life happens face-to-face using spoken language. Face-to-face interaction gives us, as speakers, access to more cues about our interlocutors' mental states and intentions, such as, among other things, facial expression and tone of voice. While recordings of spoken language do not contain all of these cues, it is still considerably more expressive than written language.

The continuous nature of a sound signal makes it, in some ways, more challenging to process than written language, which can be segmented more easily into discrete tokens that look the same regardless of who the speaker is. Spoken language, in contrast, varies wildly depending on the speaker and their dialect, voice range and emotional state, among other things. The pronunciation of individual speech sounds also varies depending on the other speech sounds in the immediate context. These things together make automatic speech recognition (ASR) a challenging problem to tackle. My goal with this project was to learn more about how to preprocess sound data for machine learning, with the ultimate goal of training a speech recognition system and get it to generate text sequences based on audio data. I was not expecting to reach this goal, but my hope was that aiming high would let me learn a lot of useful things along the way. I did train an ASR system, but as expected I did not manage to perform inference. I did, however, learn a lot of useful things about sound data and about CNNs.

2 Background

2.1 Sound data

Sound signals are, when we get right down to it, variations in pressure (Doshi, 2021b) (when we talk about sound we normally mean pressure waves in air, although other materials like water transmit sound too). These variations in pressure usually come at regular intervals, in what we call sound waves. The intensity of the variations is what we call the amplitude of the sound, and a full cycle from one amplitude peak to the next is a period. The frequency of the sound – that is, how many periods the soundwave completes within a given time frame, such as a second – is what we humans percieve as pitch. A higher frequency is percieved as a higher pitch. The commonly used unit of frequency is Hertz (Hz), which is equivalent to periods per second.

This is a rather abstract and ideal description of soundwaves; in reality, most sound signals consist of complex soundwaves composed of multiple different pitches. In human speech, the different frequencies present in the speech signal, as well their different amplitudes, is what lets us distinguish different speech sounds, such as different vowels, even when they are spoken with the same fundamental frequency.

When processing sound digitally, we need to turn thecontinuous soundwave into a discrete representation. This is done by *sampling* – measuring the amplitude of the soundwave at fixed intervals of time (Doshi, 2021b).

2.1.1 Spectrograms

A *spectrum* is the set of frequencies, and their respective amplitudes, that make up a sound signal. If a sound signal is normally plotted as amplitude against time, a spectrum can be represented as frequency against time (frequency on the y-axis and time on the x-axis). This type of plot is called a *spectrogram*. We can generate a spectrogram by using *Fourier Transforms* to decompose a signal into its constituent frequencies (Doshi, 2021b). This is essentially a way of representing a sound signal as an image, where the value of each "pixel" is the amplitude of the signal at that frequency at that time. This is very convenient for deep learning, since it lets us use CNNs to extract features from the sound signal.

2.2 Speech processing

2.2.1 Older and newer approaches

Historically, speech recognition systems used to consist of two parts: an acoustic model of different possible phonemes, based on features chosen to be relatively robust across speakers and situations, and a statistical model for sequence prediction (Mael Fabien). Examples of features to distinguish phonemes are, for example, local amplitude maxima in the spectrum (so called formants). Choosing features for models of this kind requires a lot of expert knowledge in phonetics. Eventually, hybrid approaches appeared which used Hidden Markov Models for acoustic modelling and connected them with neural networks for sequence prediction, and approaches using CNNs for feature extraction (Abdel-Hamid et al., 2014). Nowadays it is more common to use an end-to-end architecture for speech recognition, where the feature extraction and the sequence prediction are connected in a single model and optimized simultaneously during training. Common architectures include CNN + RNN and Transformers. For this project I will focus on CNN + RNN architectures, since my goal is to get a more detailed understanding of how these architectures work, and I honestly do not think I am ready to understand Transformers in detail.

2.2.2 The alignment problem

In order to predict a sequence of characters from a continuous stream of sound, we need to somehow map each timestep or sample (in our discrete representation of the sound signal, see section 2.1) to the characters in our target transcript (Doshi, 2021a). To further complicate things, a single character in the target transcript could correspond to different durations of sound in the input. Additionally, there may be pauses in the speech, so that long sections of the input audio do not correspond to any character at all in the target transcript. In other words, we need a way to *align* these sequences. This is what makes ASR so challenging compared to simpler audio classification problems.
Connectionist temporal classification, or CTC, is a way to achieve this alignment automatically. The CTC algorithm introduces an additional character, ’blank’ (which is equivalent to the empty string, not to a blank space), into the vocabulary. At each timestep, the model will thus output probabilities for every character in the vocabulary, including ’blank’. During decoding the CTC algorithm merges any characters that are repeated and not separated by ’blank’. Thus, the sequence "h-h-e-l-’blank’-l-’blank’-o-o-o" will successfully be merged into "hello".
PyTorch provides an implementation of the CTC algorithm for loss calculation (*CTCLoss*), as well as for decoding (*CTCDecoder*). In the decoding step, rather than just outputting the most probable characters, a common practice is to connect the decoder to a langage model trained on the target language. That way we can influence the model towards predicting sequences that are more likely in the target language. In addition to this, it is common to use beam search rather than plain greedy search for decoding. The PyTorch CTCDecoder supports both these options.

3 Experiments

3.1 First experiment: Birds

In this initial experiment, my focus was to familiarize myself with the new type of data. For that reason, I used a simple binary classification task of detecting the presence of bird sounds in the audio signal, first presented as the Bird Audio Detection challenge (2016/2017) (Stowell et al., 2019).

3.1.1 Data

I used the Warblr dataset[1], which contains 10 000 ten-second smartphone audio recordings containing a wide range of sounds apart from bird sounds, such as weather and traffic noise, speech and even human bird imitations. I used 80% of the data for training and the remaining 20% for validation during training. Since this was not my main project, I did not set any data aside for more thorough evaluation.

3.1.2 Model and training

I used a two-layer CNN architecture combined with a linear classifier for this task. The CNN layers both consisted of a sequence of convolution – ReLU – maxpooling, and were followed by a fully connected layer with ReLU before the final linear classifier layer. For training I used a batch size of 16 and implemented early stopping with a patience of 3 epochs based on the accuracy on the validation set.

3.1.3 Results

At first I had trouble getting the model to converge because of dataset imbalance (over 70% of the data was labeled positive) which made the model always guess the most frequent label. I managed to counter this by weighting the loss function by the fraction of positive examples. This finally led to early stopping after 37 epochs, with a top accuracy of 80% on the validation set. Plots depicting the loss and accuracy per epoch on training and validation data can be found on GitHub[2].

3.2 Second experiment: Emotions

For the second experiment, I felt confident enough to tackle actual speech data, although I started out with a simpler classification task rather than sequence prediction (as in ASR), namely, emotion classification. My idea was that since, in my experience, prosody is a strong cue for a speaker's emotional state, it would be interesting to do a cross-lingual comparison to see if prosodic differences between the languages would affect the models' performance on different datasets. A similar experiment was carried out by the creators of the Urdu dataset (Latif et al., 2018). The emotion categories I am focusing on in this experiment are *anger*, *sadness*, *joy* and *neutral*.

[1]Available here: `https://machine-listening.eecs.qmul.ac.uk/bird-audio-detection-challenge/`

[2]`https://github.com/TomBladsjo/LT2326-Project`

3.2.1 Data

I used three datasets of labeled emotional speech: the EMOVO corpus (Costantini et al., 2014) in Italian, the Urdu dataset of emotional speech (Latif et al., 2018) and the Estonian Emotional Speech Corpus (Altrov and Pajupuu, 2012). All three datasets are quite small, containing 336[3], 400 and 1,234 samples respectively. The Italian and Urdu datasets both have unifom distribution of labels. While the estonian dataset does not, it is not wildly imbalanced. The Estonian and Italian datasets are both recorded by voice actors in controlled sound environments, while the Urdu dataset consists of naturally occurring speech from Urdu talkshows.
Because of the small amount of training data, I used augmentation to increase the size of the datasets. I used two different augmentation techniques: time shift, and time- and frequency masking.[4] Using both techniques on each audio clip lead to a dataset size increase by 200%. I used a 80/20 train/test split, stratified by label, on each dataset.

3.2.2 Models and training

I trained three identical models, one on each dataset. The models used a combined CNN/LSTM architecture, consisting on two convolutional layers with ReLU and average pooling (the second using adaptive average pooling to deal with varying input sizes), followed by a fully connected layer with Tanh, followed by a 2-layer bidirectional LSTM, and finally a linear classifier. For training, I used a batch size of 8 and early stopping with 5 epoch patience. This time, i based the stopping criterion on the evaluation loss rather than accuracy.

3.2.3 Results

As can be seen in 1, both the model trained on Italian and the one trained on Urdu perform well on their own language but very badly on other languages. The model trained on Estonian performs rather badly on its own language, but generalizes reasonably well (in the circumstances) to other languages.
One contributing factor to the worse performance of the Estonian model could be the slight label imbalance, which could possibly have introduced some biases based on label frequency, but it is hard

[3]The full EMOVO corpus contains more than 336 samples, but in order to be able to compare the datasets, I only extracted the samples that were labeled as one of my four target emotions.

[4]Thanks to `https://gist.github.com/ketanhdoshi` for code examples of augmentation!

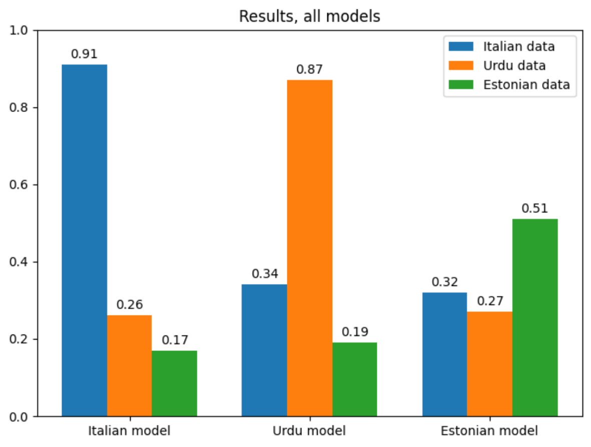

Figure 1: Emotion experiment: F1 score per model and dataset

to say for sure. Detailed training logs and plots of training and validation loss and accuracy per epoch, as well as plots of accuracy, precision, recall and F1 for each model on each dataset, can be found on GitHub[5].

3.3 Third experiment: Speech recognition

For this last experiment, I finally attempted speech recognition. I never got to actual inference and proper evaluation (which is why there is no Results section for this experiment), but I learned some valuable things, which was, after all, the point.

3.3.1 Data

I used the LibriSpeech dataset (Panayotov et al., 2015) from Open Speech and Language Resources (OpenSLR) which is a large scale corpus of read English speech. The data originally comes from audio books in the LibriVox project. I chose this dataset because it has been prepared specifically for ASR, and because OpenSLR provides suitable language models that can be used for decoding at inference time. For my project I used the smaller 100 hour train set, and the "clean" development set for validation during training.

3.3.2 Model and training

Like in the previous experiment I used a combination of CNN (with ReLU and average pooling) and LSTM. Since this experiment involved learning more complex patterns, and because I had more training data, I used a bigger model this time, consisting of four convolutional layers, followed by a fully connected layer with ReLU and a four layer biLSTM, and finally a linear classifier with LogSoftmax. I used a batch size of 4 (to avoid Cuda memory problems), CTC loss, learning rate 0.001, and early stopping with 5 epochs patience. However, with a dataset this size, running through an entire epoch takes a long time, and when I checked in on the training after a few days, it had completed 104 epochs and showed no signs of improvement after the very first epoch. With that in mind, I decided to interrupt the training. Unfortunately, I had not at that time realized that my script needed to account for the possibility that I might want to interrupt the training early, which meant that I did not write any logfile but only plotted the training and validation losses after training was completed. This means that my log data from that training process was lost, and I had no way to overview or visualize how the training had gone. I amended this in the script and trained again for 12 epochs just to make sure it worked, but I did, unsurprisingly, not obtain a better training outcome. For this reason, I did not proceed with the decoding step for this experiment.

4 Discussion

While I have definitely gained some general knowledge of how to preprocess and augment sound data for machine learning, as well as how to extract features with CNNs, I still do not have a sense of what makes a good model in terms of architecture and hyperparameters. It seems that the more complex the task, the less straightforward "quick-fixes" are available. For the first experiment, I managed to significantly improve the performance of my model by penalizing false positives in the loss function. In the second experiment, I had to tamper a lot with my model architectures before I finally arrived at one that lead to consistently decreasing loss during training. One thing that did help was to simplify the model, which initially had one additional CNN layer. I am still not entirely sure why this worked, although it seems like people generally do use smaller models for smaller datasets. On the other hand, the point of that seems to be to prevent overfitting, which was not at all the problem in my case (in fact, I do not think I have ever managed to overfit a model on the training data). My difficulties this far have mostly been about getting the model to fit the data at all.
The problem with my final experiment was similar: the loss jumped wildly up and down on both the training and the validation set, and there was

[5] https://github.com/TomBladsjo/LT2326-Project

no clear upward or downward trend for either set. From reading about CTC loss[6] I get the impression that it generally takes a long time to train a model with CTC loss, which makes sense because we are trying to teach the model complex patterns in unaligned data. If we compare this to more straightforward classification tasks like my second experiment, sequence prediction with CTC loss is clearly much more challenging - both in terms of the number of classes, and their relationship to specific time intervals in the input sequence. I did try out some different architectures for this experiment as well, but because of the size of the dataset and the complexity of the task, it was hard to "quick-test" different architectures; I had to run them for at least a few epochs to get a sense of how well they worked, which meant several hours per model architecture. I could possibly have tried my architectures out on smaller subsets of the training data, but I was worried that that would be misleading, because it might lead to the model generalizing in an oversimplified way which would not scale up to the actual training data (I may be wrong about this, and in hindsight it was a bit stupid of me to not even try). One thing I did try was to get rid of the adaptive average pooling after the last convolutional layer. My reasoning behind this was that it makes sense to retain some sense of the sequence length instead of squishing input sequences of any length into tensors of the same size (the reason I was not already doing this in the previous experiment was that it took me a while to figure out how to make this work without getting incompatible dimensions for matrix multiplications). However, when I trained the model with this architecture I got only NaN losses on both the training and validation data. It seems (from reading in discussion forums) that this may be because the loss was infinite. I did not have time to look into this further and find solutions, so instead I just kept to my old solution with adaptive average pooling, which was still not very good but at least did not lead to infinite loss.

5 Conclusion

Of the goals I initially set out to accomplish, the only thing I did not succeed in was to generate text sequences based on audio input. While I hav not given up on the idea entirely, it became clear that I would not manage to accomplish it within the time frame of this project. Apart from that, I have learned nearly everything I wanted to learn: I now have a detailed understanding of how sound is digitized and preprocessed for machine learning, how to use CNNs with audio data, and how to organize my preprocessing and training in a more practical way. One question which appeared during the course of the project, and which I did not manage to answer, was what makes a good model in terms of architecture and hyperparameters. I feel that I am also lacking an intuitive understanding of the relationship between dataset size, task complexity and model size. However, I suspect that at least the first of these is somewhat of an open question even in the machine learning community at large, so maybe I should not be too disappointed in not having managed to solve it on my own.

[6] E.g. here: `https://discuss.pytorch.org/t/ctcloss-predicts-blanks/66928`

References

Ossama Abdel-Hamid, Abdel-rahman Mohamed, Hui Jiang, Li Deng, Gerald Penn, and Dong Yu. 2014. Convolutional neural networks for speech recognition. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 22(10):1533–1545.

Rene Altrov and Hille Pajupuu. 2012. Estonian emotional speech corpus: theoretical base and implementation. In *4th International Workshop on Corpora for Research on Emotion Sentiment Social Signals (ES3)*, pages 50–53.

Giovanni Costantini, Iacopo Iaderola, Andrea Paoloni, and Massimiliano Todisco. 2014. EMOVO corpus: an Italian emotional speech database. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, pages 3501–3504, Reykjavik, Iceland. European Language Resources Association (ELRA).

Ketan Doshi. 2021a. Audio deep learning made simple: Automatic speech recognition (asr), how it works. *Towards Data Science*.

Ketan Doshi. 2021b. Audio deep learning made simple (part 1): State-of-the-art techniques. *Towards Data Science*.

Siddique Latif, Adnan Qayyum, Muhammad Usman, and Junaid Qadir. 2018. Cross lingual speech emotion recognition: Urdu vs. western languages. In *2018 International Conference on Frontiers of Information Technology (FIT)*, pages 88–93.

Mael Fabien. `https://maelfabien.github.io/machinelearning/speech_reco/#`. Accessed: November 3, 2023.

Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur. 2015. Librispeech: An asr corpus based on public domain audio books. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5206–5210.

Dan Stowell, Michael D. Wood, Hanna Pamuła, Yannis Stylianou, and Hervé Glotin. 2019. Automatic acoustic detection of birds through deep learning: The first bird audio detection challenge. *Methods in Ecology and Evolution*, 10(3):368–380.