4-30-2021

Understanding Model Reasoning in Automated Speech Systems: Implementing a Prototype Explanation System Using the LIME Method

Vadim Kudlay
University of Richmond

UNDERSTANDING MODEL REASONING IN AUTOMATED SPEECH SYSTEMS: IMPLEMENTING A PROTOTYPE EXPLANATION SYSTEM USING THE LIME METHOD

Vadim Kudlay

University of Richmond
April 30, 2021

An honors thesis submitted to
The University of Richmond
Department of Math and Computer Science
in partial fulfillment of the requirements
for the undergraduate honors program

Advisor: Dr. Douglas Szajda

_____ Date: _____

Dr. Douglas Szajda (Thesis Advisor)
Associate Professor of Computer Science
Department of Math and Computer Science

_____ Date: _____

Dr. Jon Park (Secondary Reader)
Assistant Professor of Computer Science
Department of Math and Computer Science

Abstract

The field of voice processing has seen great advancements thanks in part to the rise of deep learning. However, the application of these deep learning techniques with an audio input space leads to an interesting result not commonly found when dealing with other input domains. Namely, common techniques for generating auditory adversarial samples using gradient-based optimization have been observed to have extremely low transferability among even the same model structure. This implies an inherent difference in the latent representations of audio samples that may be worth investigating in the pursuit of a more resilient and interpretable voice processing framework.

Our core contribution is an investigation of the decision-making processes of modern voice processing implementations. Specifically, we are interested in explaining the impacts of audio input features on the alphabetic character outputs of a modern speech-to-text system such as DeepSpeech2. We investigate this with the aid of the Local Interpretable Model-agnostic Explanations (LIME) explanation technique as applied to an appropriate and contextually-aware representation of the problem space. For every alphabetic character, we select samples of audio that center on the value and use them as inputs for the voice processing system. The model predictions of these inputs are explained via LIME and the collection of all letter-use clusters are aggregated for analysis. With an understanding of the reasoning behind the classification of characters, we will be able to better understand why attacks succeed or fail, develop novel new attacks, and better defend voice processing systems against adversarial attacks in general.

Math Notations

Some convenience notation that will be used through the paper:

- ΣA : The set of all sequences constructable using the set (or alphabet) A.
- $\mathbb{N}_{\geq a}$, $\mathbb{Z}_{<b}$: Sets bounded below (like a) and above (like b). (i.e., $\{n \in \mathbb{N} \mid a \leq n\}$).
- $X[a : b]$: Convenient slice notation a la Python. Sequence of elements from the a'th element to (but not including) the b'th element, where $a, b \in \mathbb{Z}$.

Contents

1. Introduction

In our constant pursuit to automate and scale human capacity, the ability to interpret vocal inputs is a clear point of interest for many applications. A skill integral to the human experience, voice processing is a computationally hard problem that has been chipped away at over decades of innovation. Among the selection of *voice processing systems* (VPS's), *automatic speech recognition* (ASR) and *speaker identification* (SI) are two specific examples that promise to make certain laborious processes, like transcription and authentication, unsupervised and near-instantaneous [1].

These processes depends heavily on the model's ability to be accurate and reliable, which can be problematic when techniques exist that can fool a model into illogically misinterpreting an input scenario. Adversarial samples - inputs built to make the model intentionally misclassify - can be derived to take advantage of the model's learned feature relationships [2]. Many techniques for generating such examples work in the audio space, but a selection of these are difficult to explain and understand the intuition for and are thereby difficult to investigate [3]. This may suggest some inherent differences in the audio recognition models that may be worth investigating. Finding some underlying relationships between the sample generation and its impact on decisions will lead to insights that can further the pursuit of a more resilient and understood framework for general voice processing.

Our approach involves applying techniques for explaining a model's decisions on a localized scope - that is, experimenting with various audio sample types and identifying which features are impacting a given ASR model's predictions. We specifically identify existing techniques that have been successful in explaining complex models (such as those reasoning with tables, text, or images) and apply them to our auditory problem formulation. By clustering data into letter-phoneme categories and analyzing them in groups, we work to explain a model's response to different phonetic structures with the hope of finding an association between different model instances. Questions including why a model classifies a phonetic structure as it does (for example, a "c"-like element) are ones that we would like to answer by the application of this research.

2. Background

Prior to the advent of deep learning, speech recognition schemes were constructed with "hand-engineered" components that were incorporated into a unified pipeline [4]. These components focused on deriving specialized features that mimic human auditory perception. For example, a pipeline would likely have a component that modifies the frequency representation of a speech signal so that the relative weighting of frequencies matches the human auditory system response (as with, for example, the transform that computes the mel-frequency cepstrum (MFC) [5]). The output of this layer would then be used as input to another processing layer and so on. In this manner, the unified pipeline converts, in stages, portions (*frames*) of the original raw audio into character probabilities per frame, and finally to probabilistically-optimized textual representation of the entire audio sample. This hand-engineered pipeline incorporated a lot of assumptions about the nature of auditory data that seemed practical at the time. However, as machine learning techniques became more practical, models trained using deep learning techniques were discovered to be more accurate than the hand-made models and have since become the domain go-to.

2.1 Machine Learning

For this thesis, we do not assume prerequisite knowledge of machine or deep learning, so we introduce it. Since voice processing models are primarily supervised machine/deep learning (ML/DL) models, we focus our attention on these.

Supervised machine learning is the study of building models to approximate input-to-output mappings when the function in question is unknown. Assuming that there exists some unknown function $f : X \to Y$ that maps an input space X to an output space Y, machine learning can be used to parameterize a function $\hat{f} : X \to \hat{Y}$ that estimates outputs based on an observable ground truth - known (x, y) pairs of the *training data*. One such model type is a *linear model* which estimates output features as linear summations of weighted input features. This approach boils the problem down to an exercise of hyperspace curve-fitting and takes advantage of surface-level feature interactions to generate predictions [6].

Linear models can be used to tackle many hard problems, but their structure limits the overall learning ability and so they cannot be expected to capture many high-level relationships. In that case, we could incorporate two or more simplistic models in an end-to-end fashion with the hope that something of use will be captured in the intermediate data. This extension is the logical backbone behind *deep learning*, which uses multiple layers of simplistic trainable neurons connected by activation functions that introduce non-linearity to the systems. These layers are trained to create hidden intermediate representations which are useful for mapping between the input and output space. [7]. Given a well-defined neuron structure, a deep learning model is trained to a corpus of data by adjusting the neuron weights per an optimization scheme. The objective of the training is to make the network prediction - the resulting output values after the input data goes through the neuron layers - resemble the appropriate output data and generalize well for unseen data. The similarity of the prediction and the true output is quantified via a *loss function* (i.e., euclidean distance), and the method for optimizing the neuron weights is specified by an *optimizer* (i.e., gradient descent) [6].

Two types of deep learning structures will be of especial note; *convolutional neural networks* (CNNs) and *recurrent neural networks* (RNNs). CNNs are networks that model the vision processes of living organisms by preserving the location relationships of data entries. Simply put, they are the current go-to networks for trying to re-encode or interpret data when the relative location of the features is important to their interpretations. RNNs are networks with varying input and/or output lengths that are useful for temporal data representations such as audio. They flow from the intuition that data entries need context to be properly interpreted, and allow earlier data points to contribute to the decisions of later entries when useful for the decision-making process. The details of their inner workings are not relevant here, but it is useful to note that many RNNs are implemented as specific variants: LSTMs (*long-short-term memory networks*) or GRUs (*Gated Recurrent Units*). These address various technical issues, such as the so called *vanishing gradient* problem of RNNs [8]. LSTMs and GRUs are structural variations of the RNN that allow for a long-term memory component to be trained and contribute to later-stage feature predictions. For both the CNN and RNN structures, it is worthy to note that a network can have a combination of layers from either kind. Such a *deep neural network* (DNN) can have convolution layers before and after the recurrent components to take advantage of various latent properties of the dataset.

2.2 Modern VPS Models

Modern voice processing systems incorporate deep learning structures at various levels of the pipeline. Specialized feature engineering efforts like the MFCC layer have been largely replaced with convolutional neural layers that work under the assumption that the features will be properly perturbed for the system to maximize learning performance. Many uses of specialized acoustic and hidden Markov models that were integral components of piece-by-piece voice processing systems have been substituted with temporal deep learning units such as RNNs and LSTMs. Using these elements, the entire pipeline can be structured into a single end-to-end model that can be trained to efficiently capture more complex input-output relationships.

Considering the DeepSpeech 2 end-to-end pipeline [9], the following process can be used for creating an end-to-end VPS pipeline:

- **The input audio is fed in and pre-processed.** The audio sample x corresponds to a time series of amplitude vectors $\vec{x}_t$ sampled at a fixed rate. The time series typically undergoes noise reduction to better separate the signal from background noise, and low pass filtering to remove frequencies beyond human auditory perception. Depending on the model, the resulting time series of amplitudes is converted into an alternative form to take advantage of some biases and/or relational properties. For example, some models may choose to use the MFCC form to bias the representation to resemble human hearing, while others may choose a spectrogram representation to convert the amplitudes into a 2D image of the frequency spectrum [10]. In the case of DeepSpeech2, it converts the amplitude vectors into power-normalized spectrograms to take advantage of the 2D locality while limiting the data loss of the translation.
- **The time series is fed through a DNN system.** A model with a recurrent core is selected to calibrate weights to fit to the pre-processed training audio set. In the case of DeepSpeech2, the first layers are convolutional layers intended to aggregate the representation to a most suitable form for the recurrent core to reason with. At this stage, features of the input data may be biased based on the network's optimizer; for example, mutations like MFCC's human-biasing may or may not be made depending on what the network determines as most useful in minimizing the loss function in tandem with the rest of the network.

The convolutional layers then feed into a series of recurrent layers such as GRUs (as is the case for DeepSpeech2). As recurrent structures can take a variable number of inputs to generate a variable number of outputs, they are generally designated to take in as many input vectors as are present in the time series. Their output can then be designated to be a series of alphabetic characters (a-z) as well as a null tokens ϵ which will be explained shortly. More specifically, the prediction is a sequence of probabilities corresponding to the likelihood of an input component representing an instance of each output category, the result of which is selected based on some selection criteria (i.e. max probability, contextual reasoning, etc).

- **The DNN's prediction sequence is mapped into regular text.** As current techniques position the DNN to make per-time-frame predictions (also known as a predicted alignment), a decision has to be made about how this can be evaluated against a correct answer in the human-readable text space. For this reason, the RNNs used for speech processing are trained using the Connectionist Temporal Classification (CTC) loss function [11]. The CTC function relies on the idea of a compression where consecutive instances of the same character are compressed down to a single instance and the ϵ blanks are removed to allow for consecutive characters. For example, the alignment "hheeeellϵlllloo" could be collapsed to "helϵlo" and finally to "hello". Working in reverse, the CTC algorithm uses a dynamic programming approach to compute the difference between the RNN's predicted alignment and the set of valid alignments that could map to the desired output on compression.

By training the above pipeline with a sufficiently large dataset, the resulting model will allow for a very flexible solution to VPS that can be adopted to a wide range of audio types and languages. For example, DeepSpeech2 was unveiled as a structure that could be trained to operate on either English or Chinese depending on which language the training corpus was in [9]. For completeness, it should be noted that an ASR model will follow up by incorporating a language model that converts an alignment into a valid word or phrase based on contextual information. However, the impacts of this component are outside of the scope of our work, as we would like to limit our discussion to the end-to-end model decisions.

2.3 Adversarial Examples

By alleviating the need for explicitly-defined specialized components, a deep learning model can be trained to the data in ways that may not make intuitive sense to humans, despite demonstrating a high level of predictive power. The weights of a model are being optimized given a constant structure of the network; for example, there will be a well-defined number of layers with a specific number of nodes per layer for a given machine learning model. Each of these nodes will have associated parameters that effectively capture the "knowledge" of the model. When training on such a network, any semblance of the intuitive intermediate representations that are learned through training will be embedded to fit within the given structure. This can lead to interesting and unrepresentative component relationships when the model is trained to a given selection of ground truth data points. Because of this, the network can make unexpected and unreasonable predictions in certain scenarios. This disadvantage can be exploited using *adversarial samples*, or maliciously-crafted inputs that cause the network to misclassify despite adding only minor alterations to a correctly classified example [12].

Adversarial attacks are a known problem in deep learning systems, and the techniques used to generate them have been well researched across many domains. In doing so, it has been found that many adversarial techniques lead to *transferable* adversarial samples, which are effective across a range of similar models. Surprisingly, many techniques for creating transferable adversarial samples have been shown to be ineffective in the sound domain, even when training and validated on the same group of data. Special techniques like the one discussed in *"Hear "No Evil", see "Kenansville""* have been created to make transferable audio samples, but more mainstream techniques that rely on gradient-based optimization have been shown to be ineffective in the audio space [3]. This runs counter to the results found in other deep learning domains, which makes ASR models very interesting and may suggest an inherent difference in their fundamental operations.

2.4 Explainable AI

We are interested in investigating the interesting properties of adversarial techniques in the audio, and so begin by trying to explain the ASR model's decisions. By wanting to

explain the predictions of a model from a human-intuitive perspective, we want the model to become *interpretable* and *explainable* by means of either the network's simplistic structure or a specialized explanation technique [13]. This is trivial for a class of *interpretable models* such as linear models. In such models, there is an intrinsic property that allows the internal structure of the network to be interpreted to derive insight; for example, the linear model contains coefficients which directly specify what input feature contributes to a given output feature. However, it can become less straight-forward to explain more complex models as the inputs are propagated through many hidden layers. Manually investigating the weights and reasoning about the relationships can quickly become intractable for many neural networks, and trying to identify clusters of human-intuitive characteristics can be very difficult due to the mechanical nature of the optimization. This also assumes ready access to the model's internal weights and structures, which is typically not available for real-world systems.

Outside of manual investigation, *explanation techniques* can be used to reason about how input features impact a model's prediction. One such explanation technique, Local Interpretable Model-agnostic Explanations (LIME), is a model-agnostic technique for explaining predictions of arbitrary classifiers or regressors [14] on specific inputs. LIME explains a complex model's prediction of a specific data point by creating an interpretable model that mimics the classifier's decisions around the point in question. The *local* interpretable model can be trained by generating a large number of neighboring points and aggregating the complex model's predictions of them. The neighbor-prediction pairs can then be fed as training data for the interpretable model and the result will be a simple model that should mimic the complex network for inputs within a small area of the domain. (This idea is akin to approximating high degree polynomials with lines in a sufficiently small neighborhood of a given point.) Given this simplified model, we can identify which features are important for the original point's classification by identifying which coefficients have the most weight.

An important feature of LIME is that the approximating model generally acts on a specialized space that is different from the domain of the complex model. In particular, the interpretable model acts on an *interpretable space*, which is a version of the input space where the features are known and easily interpreted by humans. The purpose of the interpretable space is to add meaning to the features being explained, since quantifying the impact of arbitrary-seeming inputs does not greatly improve our understanding of the model logic. Thus, a significant consideration when employing the LIME method is determining a suitable interpretable representation for the problem's natural input space.

This technique utilizes the complex model as a *black-box* for which the weights and structures are not known beyond a surface level. In the general sense, the LIME algorithm is model-agnostic, meaning that it can be parameterized to an arbitrary model and data space. As such, we will be using the algorithm to gain insight into an ASR model's predictions on various audio components.

3. Methodology

Our starting objective is to build an understanding of a VPS's decision-making process, so we must first formalize what relationships can be investigated on a common model. We will then propose a method for identifying and investigating the relationships between the input and prediction space. This will be done using black-box explanation techniques and is intended to be tractable for a variety of modern VPS models. For our main focus, we will consider speech recognition models, as they are a rather complex example that is of great use and already implemented on a large scale.

3.1 Black-Box Assumptions of ASR Models

We will be observing the models from a purely black-box perspective, meaning that we will consider the input/prediction relationships at the end-points of the DNN system. By investigating the model at a black-box level, we hope to avoid narrowing our window of applicability to a certain class of model structures and to capture more general relationships that are not specific to various model structures and training runs. A modern ASR model:

- Takes as input either a time-series of amplitudes (raw audio) or a time-series of spectral vectors (spectrogram). Both are valid representations, but we will start off by assuming a raw audio input space and building up to the spectral representation later.
- Predicts either a character alignment (from the DNN) or a phrase in a given language (from an end-stage language model). We will avoid considering the latter, as it would needlessly introduce a language model's decision-making complexity into our analysis.

Of note, it would be beneficial in our analysis to be able to reason with phoneme predictions. This would allow for a more natural selection of output categories given the basic structure of audio. However, since the CTC loss function is the standard-use solution that works well in practice, modern models generally do not produce phoneme classifications by default. The several models that can offer phoneme predictions (such as CMU Sphinx) do so with the understanding that phoneme prediction is slower and less accurate by being a problem

with far fewer inherent constraints [15]. As it stands, reasoning with the letter outputs of the predicted alignment is our most direct option given the problem space.

3.2 Aggregating Sound Clusters

We would like to explain how deviations in the input space affect the model output, so we will investigate clusters of associated samples and see which input features have the most effect on their predictions. We previously suggested that investigating clusters of phoneme data would be preferred due to the phonemes' more distinct appearance in raw audio representations. Despite noting that this is not a natural prediction of modern ASR models, we can still take advantage of phonemic structures if labeled phoneme data is available. Given both the letters and phonemes associated with a particular sound utterance, we can segregate the sample into a bucket of letter-phoneme joint class and take advantage of both letter and phoneme categorizations. For example, a sound sample with focusing letter "a" can be further classified to contain an "a" component of the "ae" phoneme, or "a_ae". This is in contrast to sounds like "a_ao" and "e_ae", which have different phoneme and/or letter classes associated with them. In the event of a letter existing with no associated phoneme, such as a silent "e," a blank phoneme class could be used.

With this premise in mind, we can aggregate audio samples with desired letter-phoneme classes if ground-truth transcription, phoneme sequence, and incidence time is known:

- Let $Wd = \{w_0, w_1, \cdots\}$ be the set of all valid raw audio frames.
- Let $Al = \{$'a', 'b', $\cdots$, 'z', ' '$\}$ be the set of valid transcription characters.
- Let $Ph = \{$'aa', 'ae', $\cdots$, 'zh'$\}$ be the set of valid phonemes.
- Let $APh = \{a + p \mid a \in Al,\ p \in Ph\}$ be the set of possible letter-phoneme pairs.
- Let $x \in \Sigma(Wd)$ be a ground-truth audio samples bound on the discrete time interval $[t_0(x), t_f(x)]$ with known transcript $text(x) \in \Sigma(Al)$, phoneme sequence $phon(x) \in \Sigma(Ph)$, and phoneme incidence time $time(x) = \{(t_i, t_{i+1}) \mid 0 \leq i < f\}$.
- Let X be the set of all ground-truth audio samples.

Then, we can create a set of data clusters based on their expected letter-phoneme classes using Algorithm 1. This algorithm returns a dictionary of raw audio samples separated based on their known letter-phoneme designations.

Algorithm 1: Letter-phoneme group aggregator

Input : The set of ground-truth audios X
Output : Dictionary associating APh-space categories with raw audio samples
C = Dictionary: $APh \rightarrow$ [List: X]
for $x \in X$ **do**
  $sa \leftarrow$ `align`$(text(x),\ phon(x))$ `// Subset alignment per phoneme`
  for $1 \leq i < f - 1$ **do**
    $ts_0 \leftarrow time(x)_{i-1,\ 0}$ `// Select enlarged range of audio`
    $ts_1 \leftarrow time(x)_{i+1,\ 1}$ `//  to pull in temporal context`
    for $c \in sa_i$ **do**
      $ap \leftarrow c + phon(x)_i$ `// Create APh from aligned chars`
      $C[ap] \leftarrow C[ap] \bigcup x[ts_0 : ts_1]$ `// Add padded audio segment to D`
    end
  end
end
return C

At a high level, the algorithm separates out the manually-labeled phoneme components associated with a raw audio file, identifies the location within the phoneme for which each letter corresponds to, and records letter-phoneme pairs as they are identified. We then drop them in the appropriate dictionary bucket along with a raw audio snippet that contains an area around the time of the letter's utterance. Note that the algorithm considers the phonemes that surround a given choice. This is to pull in temporally-contextual features that could be used by the DNN's recurrent components. The padding is applied to both sides of the phoneme utterance as we will not be assuming a preference between a uni-directional and bi-directional recurrent system [16]. The algorithm also takes advantage of an undiscussed `align` algorithm. This algorithm identifies the time instances of the alignment characters within the phoneme being considered. This can be done by working backwards through various CNN layers to see which audio frames contributed to a specific position and when they occurred.

3.2.1 Extending the Method to Multi-Letter Graphemes

The output centric approach we propose arises from practical considerations, but does provide advantages in terms of flexibility. We have already mentioned the possibility of grouping different instances of a specific character into distinct classes, depending on their use (e.g., hard consonant versus silent). The approach also allows us to derive explanations for multi-letter grapheme outputs such as "ch", "sh", and "eigh". Extending our method to handle such output is straightforward. Suppose, for example, that the characters in the grapheme "eigh" consist of the slice $A_x[i:j]$ for indices i, j of alignment A_x. Then, rather than determining the raw audio frames that directly contribute to a single character's classification, we would instead determine the frames that contributed to the entire grapheme. This would become our s_{char} (or at that point $s_{grapheme}$) segment.

3.3 Explaining ASR Decisions with LIME

Now that we have derived a dictionary of segmented audio samples, we can proceed by explaining the audio sample's predictions and grouping the results by the letter-phoneme. We have specified how to aggregate letter-phoneme clusters with the goal of explaining their predictions when fed through an ASR model. One such explanation is the quantification of feature importance to help identify which input components contribute the most to a given decision. Here, we will discuss in detail how LIME can be used to generate such predictions.

3.3.1 Assumptions of LIME

In our case, we seek to explain a model's prediction for some audio sample x. The original ASR model C is a complex end-to-end pipeline with no clear explainable component such as a set of simple coefficients. For this, LIME can be used to generate an interpretable (linear) model g that provides an approximation for a decision boundary derived from C. In general, x exists in an original ("raw") feature space D_R that is well-defined for a given problem. The algorithm requires an analogous *interpretable space* D_I whose elements are binary vectors (e.g., elements of $\{0,1\}^N$ for some fixed length N) that is associated with D_R by some function $\gamma : D_R \to D_I$. This function must be at least pseudo-invertible such that $\gamma^{-1} : D_I \to D_R$ exists and is well-defined on those vectors in the image of γ.

Given this premise, the LIME method requires training a linear model g that provides an approximation for the decision boundary near a modified $x' = \gamma(x)$ in D_I.

In order to determine the parameters of the linear model g, we need to provide training vectors, whose inputs are in D_I, which are derived from the behavior of the original model near the base input x. We start by determining the interpretable analogue $x' \in D_I$ of x. We then choose a specific character, denoted c, at a specific position, j, in the output alignment resulting from the original model applied to x. A single training datum is then obtained as follows:

1. Pseudo-randomly generate a perturbation $z'_i \in D_I$ such that b_i bits are flipped. z'_i is the input part of one training datum.
2. Apply the mapping γ^{-1} to z'_i to obtain the vector $z_i \in D_R$.
3. Apply the original model to z_i.
4. The output assigned to input z'_i is the probability the original model specifies for character c at alignment position j when given input z_i.

This process is repeated to create a large set of training vectors $z' \in D_I$ (each associated with some $z = \gamma^{-1}(z') \in D_R$) which will be used to train the coefficients of g. To give priority for the training vectors, the impacts of z' on the model is weighted based on the distance from z to the original x. Specifically, if z is far from x in D_R, it will have less impact on the model weights. This can then be achieved by some proximity function $\pi_x : D_R \to \mathbb{R}$ that is parameterized to compute proximity relative to base x and takes advantage of a raw distance metric $d : D_R^2 \to \mathbb{R}$. For our implementation, we will allow g to specifically be a linear model, such that every input feature contributes to every output feature with some weight. As such, g has the form $g(z') = \beta z'$ where we treat z' as a column vector and assume β is the coefficient vector $\beta = [\beta_0, \beta_1, \ldots, \beta_{N-1}]$. Then, we seek to determine the values of β_i that minimize a loss function $\mathcal{L}$ that we will have to specify. Once the parameters (β_i coefficients) for g are determined, the few with the largest magnitude tell us which frequencies are driving the classification of x.

The variations of the proximity function π_x and the loss function $\mathcal{L}$ will take the form of the defaults specified by the LIME authors for our usage; specifically, $\mathcal{L}$ will be taken as

a weighted linear loss which can optionally be regularized or modified as needed:

$$\mathcal{L}(f, g, \pi_x) = \sum_{z,z'} \pi_x(z)(f(z) - g(z'))^2.$$

Similarly, the proximity metric π_x can use a standard exponential kernel as a default, and we can define it relative to x for convenience:

$$\pi_x(z, d_x) = \exp(-d_x(z)^2/\sigma^2).$$

For the distance metric over D_R, There are actually more options that can be considered, but we can safely use any normalized distance metric, such as the normalized euclidean L^2 norm or cosine similarity. Since we will only need the distance relative to the original sample point x, we can simplify the defining d relative to x:

$$d_x(y) = \frac{1}{L} \sum_{n=0}^{L-1} |x_n - y_n|^2.$$

With all of this, the overall LIME algorithm can be specified by Algorithm 2. Note that to apply this method to an ASR model, we still need to implement the details of D_R, D_I, γ, γ^{-1}, and δ in order to apply the method to our chosen space. Note that in practice, the process should be largely vectorized for performance reasons; however, the algorithm is presented iteratively for simplicity.

Algorithm 2: LIME Algorithm

Input : s base sample to explain
γ pseudo-invertible mapping between input and interpretable space
δ perturbation function (in interpretable space)
n perturbation count
d_x distance metric in input space
π_x proximity metric for weighing perturbations
C queriable complex model
g trainable simple model

Output : An interpretable model trained from C

$x' \leftarrow \gamma(s)$ `// Maps sample to interpretable space`
for $i \in \mathbb{N}_{\leq n}$ **do**
  $z'_i \leftarrow \delta(x')$ `// Generates perturbations for` x'
  $z_i \leftarrow \gamma^{-1}(z'_i)$ `// Maps perturbation to input space`
  $\ell_i \leftarrow C[z_i]$ `// Makes prediction for perturbation`
  $w_i \leftarrow \pi_x(z_i, d_x)$ `// Computes weight based on proximity`
end
g.`fit`(z', ℓ, w) `// Predict` $z' \rightarrow \ell$ `weighted by` w
return g

3.3.2 Defining Input and Interpretable Space

In order to use LIME on our ASR model, we will need to parameterize the algorithm for our problem spaces. We first need to specify the original feature space D_R, the interpretable feature space analogue D_I, and the (pseudo)invertible function $\gamma : D_R \rightarrow D_I$ that maps between them. So far, we have used Algorithm 1 to aggregate a set of inputs X with elements $x \in \Sigma(Wd)$ such that $||phon(x)|| = 3$ and a clear character of focus for the center phoneme is known. We will modify the representation away from the windowed view in favor of a 3-segment view. Specifically, we will subdivide the raw audio sample into three consecutive intervales, $x = \{s_{prefix},\ s_{char},\ s_{suffix}\}$, where s_{char} contains the amplitude samples corresponding to the character of interest, and s_{prefix} and s_{suffix} contain all prior and later amplitude samples, respectively.

For the subsequent stage, we will need to start considering the raw audio space's conversion into a spectral representation. Spectra are derived via the discrete Fourier transform (DFT), so define $DFT : \mathbb{R}^N \to \mathbb{C}^N$ such that for a sequence $a \in \mathbb{R}^N$:

$$DFT(a)_k = \sum_{n=0}^{N-1} a(n) e^{-\frac{i2\pi}{N}kn}, \ 0 \leq k \leq N-1.$$

Computing the DFT as defined can be costly, so in practive implementations leverage the Fast Fourier Transform[17]. The FFT requires N to be a power of 2, so for a sequence a of length N, let $\texttt{Pad}(a)$ be the padding of a to the smallest length $[\widetilde{N} \in 2^n] \geq N$ for $n \in \mathbb{N}$. Finally, for notational simplicity, we denote time-domain sequences with lowercase letters and their DFT with the corresponding uppercase letter, i.e., $DFT(s_{char}) = S_{char}$. With all of this, we have established a $D_R \in \mathbb{R}^{||x||}$ that corresponding to the spectral representation of a raw audio sample.

3.3.3 Defining Gamma

Let us define the mapping γ as follows. Let $T > 0$ be a pre-configured threshold value (more on this below), and consider j such that $s_{char} = x[j : j + N]$. Then for arbitrary $z \in D_R$, we define $\gamma(z) \in \{0, 1\}^{\widetilde{N}/2}$ as follows:

$$\gamma(z)_k = \begin{cases} 1 & \text{if } \left|\texttt{DFT}(\texttt{Pad}(z[j : j + N]))_k\right|^2 > T \\ 0 & \text{otherwise} \end{cases}, \ 0 \leq k < \widetilde{N}/2$$

Put another way, to compute $\gamma(z)$, we first take the power spectrum of the $\widetilde{N}$-element sequence resulting from padding the elements of z corresponding to s_{char} out to length $\widetilde{N}$. Then we map the resulting $\widetilde{N}$ element power spectrum to $\{0, 1\}^{\widetilde{N}/2}$ by converting spectra greater than the cutoff threshold T to the value 1, and all others to the value 0. In effect, $\gamma(z)_k$ is a boolean indicating whether the power present at the k-th frequency component is higher than threshold T. In turn, that is the definition of a point in D_I. Note that though the DFT of $\texttt{Pad}(z[j \ : \ j + N])$ is a sequence of length $\widetilde{N}$, the output of γ is only the first half of that sequence. This is both sufficient and necessary (as will be seen below) because of the symmetry properties of the DFT when applied to real-valued sequences.

For our method, we will use an assumption that the input can be explained by considering existing and non-marginal frequencies (i.e., that are above a threshold T). With that, we would like to see how the model classification will be affected when the frequencies are marginalized and how that affects the letter-phoneme classification. For this, the perturbation function δ can interpreted as perturbing the original sample x by marginalizing a small randomly selected set of frequencies.

To define the inverse mapping, we need to reverse D_I values back into spectral space. The mapping $\gamma^{-1} : D_I \to D_R$ is defined as follows. First, let S_{char} be the discrete Fourier transform of $\texttt{Pad}(s_{char})$ and let $S_{diff}(k)$ be a frequency that deviates from the k-th component of S_{char}. Recall that the purpose of a D_I entry is to indicate, per perturbation, whether or not each feature should be kept in a similar class as that of the original x sample. Recall also that to achieve a D_I-space feature with value 0, the frequency must be below some threshold T. Then, it may seem logical to let $S_{diff} = 0$ and thereby eliminate frequencies to see how that impacts the ASR model's decisions. The inclusion of 0 frequencies has an interesting property in models that was noted in [18] where a method for creating adversarial examples by zero-ing out select frequencies was introduced. However, on experimentation it was found that this effect is due to the model not having being trained on 0-value features; specifically, using the same technique with light noise effectively prevented the attack. With that, we will defer to using a light frequency profile of random noise that is far below the desired T threshold. One way of achieving something similar is to allow $S_{diff} = UT/2$ for a normalized uniform random variable U, but we will leave it as S_{diff} for generality. As such, for $z' \in D_I$, define the vector $(Z')^{-1}$ of length $\widetilde{N}$ as follows:

$$(Z')_k^{-1} = \begin{cases} \begin{cases} S_{char}(k) & \text{if } z'(k) = 1 \\ S_{diff}(k) & \text{if } z'(k) = 0 \end{cases} & \text{if } 0 \le k < \widetilde{N}/2 \\ \overline{(Z')^{-1}_{\widetilde{N}-k}} & \text{if } \widetilde{N}/2 \le k \le \widetilde{N} \end{cases}$$

where the bar represents the complex conjugate.

We then compute the inverse DFT of $(Z')^{-1}$, where as usual, the inverse DFT applied to an arbitrary sequence length N sequence H in frequency space is defined by

$$DFT^{-1}(H)_k = \frac{1}{N} \sum_{k=0}^{N-1} H_k e^{-\frac{i2\pi}{N} kn}.$$

Applying the the inverse DFT to $(Z')^{-1}$ results in a real-valued sequence of length $\widetilde{N}$. This sequence is then truncated down to the first N elements, leaving $\mathrm{DFT}^{-1}((Z')^{-1})[0:N]$. Finally, this slice is inserted into the original time sequence in place of the segment s_{char}, resulting in a sequence of raw audio samples such that $x \approx \gamma^{-1}(\gamma(x))$:

$$\gamma^{-1}(z) = s_{prefix} \ \cup \ \mathrm{DFT}^{-1}((Z')^{-1})[0:N] \ \cup \ s_{suffix}.$$

3.3.4 Redefining for a Smaller Sample Space

The previous premise works well in theory, but presents a practical difficulty: the interpretable space is high dimensional. As a result, attempting to achieve good coverage of a neighborhood of a point in D_I requires an unreasonable number of permutations. Modifications to the algorithm can be made, for example, by grouping small clusters of adjacent frequencies together, but such an approach risks losing frequency information that may have a bearing on our generated explanation. We are currently considering this and other potentially more effective approaches.

One of these alternate approaches involves using an autoencoder to derive a set of smaller features with mappings to and from the original D_R [19]. Simply put, an autoencoder uses a single hidden layer structure and trains it to map an input to itself. By using fewer nodes in the hidden layer than the features of the data entry, the hidden layer values represent a more compact form of the data that can be disassociated to reform the original entry with some retained accuracy. With this, the coefficients between the input and hidden layer create the forward mapping, and the coefficients between the hidden layer and the output form the backward mapping. Using a new D'_R composed of the encoded features, a D'_I space can be defined using the same definitions. This has the potential to reduce the dimension of the intepretable space to a level that is reasonable compact, but that retains important information, and can be performed on a per letter-phoneme cluster. In future work, this may also help to identify components that have shared affect as part of the explanation.

4. Experiment

To test our methodology, we are running experiments on a modern ASR model. For our initial experiments we use a readily-available pre-trained DeepSpeech2 engine [9], specifically the open-source *DeepSpeech on PaddlePaddle* implementation (GitHub: PaddlePaddle/DeepSpeech). The explanations are applied to a set of moderate subset of TIMIT entries [20] which have been sliced in accordance with the methodology.

For the LIME method, we have implemented a bare-bones and flexible wrapper algorithm in Python that allowed for heavy customization towards an arbitrary model. The library is designed to minimize the number of assumptions on the system and concentrate complexity at the user-specifiable endpoints; in that sense, it is less tailored for general LIME users and more tailored for those wanting to experiment with more novel applications of LIME. This diverges from mainstream implementations like the Python `lime` library (**cite**) by allowing for customizations that not definable using even its most generic base class.

The following protocols must be specified and are available in the code:

- **InterpretableMapping:** Mapping between real space and interpretable space.
- **PerturbationFunction:** Perturbs input data in the interpretable space.
- **DistanceFunction:** Computes distance between permuted entries.
- **KernelFunction:** Uses distance to assign weights per permuted entries.
- **LIMEEstimator:** Linear model; can be fit to data and used to make predictions.
- **ModelPredictFunction:** Black-box complex model which takes input and predicts.

This fully parameterizes the **Explainer** which implements the LIME algorithm as described above in a vectorized way.

5. Results and Conclusion

At the time of this writing, the discussed LIME implementation and a surrounding framework for feeding in auditory data and evaluating the system has been constructed and is currently being fine-tuned. The implementation of and experimentation with the system has been met with roadblocks that have allowed us to progress in the right direction and flesh out a methodology scheme to guide further developments. The discussed in 3.3.4 is actively being incorporated into the framework at the time of writing, and we expect other special considerations to come up as the system reaches a testable state. At the time of writing, we have not been able to produce an analysis of the clustered results but will be generating results from the developed framework after some more tuning. Still, the theoretical grounding and resolution of pitfalls that have been discussed will serve as useful finding for those interested in progressing towards an interpretation voice model framework.

As we stated previously, the goal of investigating the discussed clusters is to find associations and progress towards a method of interpreting a speech recognition model's decisions. Further development of the theory intends to progress the state of knowledge towards realizing the successes (and likely the limitations) of existing interpretations techniques found in other domains. Once this is achieved, we will be able to analyze adversarial attacks from an interpretive lens and better reason about what kinds of auditory properties are being exploited by them. Furthermore, this would allow us to connect the audio space to the existing pool of interpretable model research. Such a development would hopefully allow us to take advantage of existing findings in adversarial attack/defence (such as those in the image space [2]) to deal with the audio problems. From our current findings, we do theorise that the LIME interpretation technique may have some success limitations in the time sequence domain due to the technique's linear interpretation tendencies [14]. As a result, we intend to consider the application of LEMNA in a subsequent work due to its non-linear interpretation properties and compare the two in the audio space [21]. This application will involve some more considerations but should have significant overlap with the input-output constructions discussed in the paper.

Bibliography

[1] T. Kinnunen and H. Li, "An overview of text-independent speaker recognition: From features to supervectors," *Speech Communication*, vol. 52, no. 1, pp. 12–40, 2010.

[2] N. Akhtar and A. Mian, "Threat of adversarial attacks on deep learning in computer vision: A survey," 2018.

[3] H. Abdullah, K. Warren, V. Bindschaedler, N. Papernot, and P. Traynor, "Sok: The faults in our asrs: An overview of attacks against automatic speech recognition and speaker identification systems," 2020.

[4] A. Graves and N. Jaitly, "Towards end-to-end speech recognition with recurrent neural networks," in *Proceedings of the 31st International Conference on Machine Learning* (E. P. Xing and T. Jebara, eds.), vol. 32 of *Proceedings of Machine Learning Research*, (Bejing, China), pp. 1764–1772, PMLR, 22–24 Jun 2014.

[5] B. Logan, "Mel frequency cepstral coefficients for music modeling," in *In International Symposium on Music Information Retrieval*, 2000.

[6] T. Hastie, J. Friedman, and R. Tisbshirani, *The Elements of statistical learning: data mining, inference, and prediction*. Springer, 2017.

[7] Y. Lecun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, p. 436–444, 2015.

[8] S. Hochreiter, "The vanishing gradient problem during learning recurrent neural nets and problem solutions," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 06, no. 02, pp. 107–116, 1998.

[9] D. Amodei, R. Anubhai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, J. Chen, M. Chrzanowski, A. Coates, G. Diamos, E. Elsen, J. H. Engel, L. Fan, C. Fougner, T. Han, A. Y. Hannun, B. Jun, P. LeGresley, L. Lin, S. Narang, A. Y. Ng, S. Ozair, R. Prenger, J. Raiman, S. Satheesh, D. Seetapun, S. Sengupta, Y. Wang, Z. Wang,

C. Wang, B. Xiao, D. Yogatama, J. Zhan, and Z. Zhu, "Deep speech 2: End-to-end speech recognition in english and mandarin," *CoRR*, vol. abs/1512.02595, 2015.

[10] L. Wyse, "Audio spectrogram representations for processing with convolutional neural networks," *CoRR*, vol. abs/1706.09559, 2017.

[11] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, "Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks," in *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, (New York, NY, USA), p. 369–376, Association for Computing Machinery, 2006.

[12] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," 2015.

[13] R. Guidotti, A. Monreale, S. Ruggieri, F. Turini, F. Giannotti, and D. Pedreschi, "A survey of methods for explaining black box models," *ACM Comput. Surv.*, vol. 51, Aug. 2018.

[14] M. T. Ribeiro, S. Singh, and C. Guestrin, ""why should i trust you?": Explaining the predictions of any classifier," 2016.

[15] N. Shmyrev, "Phoneme recognition (caveat emptor)."

[16] M. Schuster and K. Paliwal, "Bidirectional recurrent neural networks," *Signal Processing, IEEE Transactions on*, vol. 45, pp. 2673 – 2681, 12 1997.

[17] E. W. Weisstein, "Fast fourier transform. From MathWorld–A Wolfram Web Resource.." Last visited on 17/5/2021.

[18] H. Abdullah, M. S. Rahman, W. Garcia, L. Blue, K. Warren, A. S. Yadav, T. Shrimpton, and P. Traynor, "Hear "no evil", see "kenansville": Efficient and transferable black-box attacks on speech recognition and voice identification systems," 2019.

[19] D. Bank, N. Koenigstein, and R. Giryes, "Autoencoders," 2020.

[20] J. S. Garofolo, L. F. Lamel, W. M. Fisher, J. G. Fiscus, and D. S. Pallett, "DARPA TIMIT acoustic-phonetic continous speech corpus CD-ROM. NIST speech disc 1-1.1." NASA STI/Recon Technical Report N, Feb. 1993.

[21] W. Guo, D. Mu, J. Xu, P. Su, G. Wang, and X. Xing, "Lemna: Explaining deep learning based security applications," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, (New York, NY, USA), p. 364–379, Association for Computing Machinery, 2018.