HARNESSING ADVERSARIAL SAMPLES AGAINST VOICE ASSISTANTS

By

YI WU

A thesis submitted to the

School of Graduate Studies

Rutgers, The State University of New Jersey

In partial fulfillment of the requirements

For the degree of

Master of Science

Graduate Program in Electrical and Computer Engineering

Written under the direction of

Prof. Yingying Chen

And approved by

New Brunswick, New Jersey

May, 2019

ABSTRACT OF THE THESIS

Harnessing Adversarial Samples Against Voice Assistants

by YI WU

Thesis Director:

Prof. Yingying Chen

With the widespread use of machine learning techniques in many areas of our life (e.g., recognizing images, videos, and voice commands), their security vulnerabilities have become an increasing concern of the public. For instance, a deep neural network (DNN), as a popular class of machine learning techniques, has been proved defenseless against deliberately designed adversarial attacks. Existing studies mainly focus on feeding adversarial images to fool an image classification system. They can make the system misclassify the images as any target object by adding carefully-crafted perturbations. In this thesis, we propose an adversarial attack against automatic speech recognition (ASR) systems (e.g., Siri and Google Assistant). We demonstrate that an adversary can embed malicious voice commands into regular songs, and these embedded commands can be recognized by the ASR system but go unnoticed by humans. Particularly, we use a genetic-based algorithm to craft the original song with the probability density function (PDF) identifier of the malicious comments, allowing the crafted song to be recognized as the embedded comments. Evaluation demonstrates that the commands in the crafted songs can be successfully recognized by the ASR system (e.g., Kaldi) with an average success rate of over 58%. By calculating the covariance between the crafted song and the original one, we show that the similarity between them is over

95%, making it hard to be noticed by humans.

Acknowledgements

First and foremost, I would like to show my deepest gratitude to my supervisor, Prof. Yingying Chen, for her careful guidance and great patience for my master thesis in the past semesters.

I would also like to show my greatest thanks to my mentor Jian Liu and Chen Wang. They helped me a lot when I face troubles din the research process and gave me much useful advice on analyzing problems. They also helped me modify this thesis; without their help, I would never finish this thesis.

Last but not least, I'd like to thank all my friends for their encouragement and support.

Table of Contents

Chapter 1

Introduction

A voice assistant is a system which can listen to a human's command and assist with various tasks. It can perform a variety kinds of actions (e.g., turning on/off lights, answering questions, playing music and broadcasting the weather reports, etc) after hearing a specific wake word. Many smart devices we use in our daily lives have been integrated with voice assistants. They are on our smartphones (e.g., Siri, Google Assistant), standalone smart speakers (e.g., HomePod, Amazon Alexa), and various IoT devices in our smart homes. Moreover, specific functions/operations in vehicles, as well as in retail, education, healthcare, and telecommunication environments, can be operated by voices. A survey [1] conducted in May 2017 indicates that 46% of Americans have used voice assistants, with 42% having used them on smartphones, 14% on computers or tablets, 8% on stand-alone devices and 3% on other devices as shown in Figure 1.1.

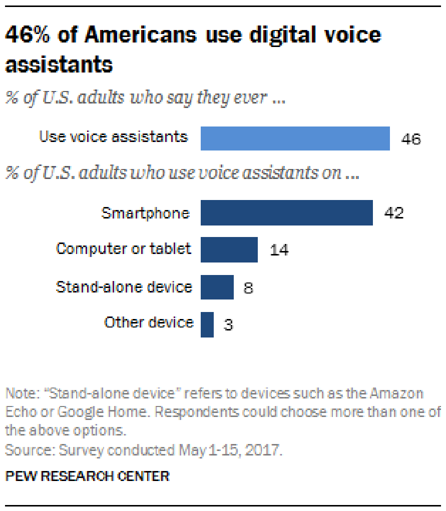


Figure 1.1: Voice Assistant Widely Used [1].

Meanwhile, the number of people using voice assistants on smart speakers is still growing. According to the Voicebot Smart Speaker Consumer Adoption Report 2018 [3], almost 10% of people who do not own a smart speaker plan to purchase one. If this holds, the user base of smart speaker users will grow 50%, meaning a quarter of adults in the United States will own a smart speaker.

Since voice assistants have already played an essential role in our daily lives, it becomes imperative to resolve their security problems. Can we ensure that they are always working accurately, and will there be potential dangers hidden inside its prosperity and convenience? What if the voice assistants misunderstand what users are saying, e.g. when the owner wants it to dim the light, but it triggers the fire alarm instead?

Particularly, a typical voice assistant system consists of three main parts: voice capture, speech recognition, and command execution. In this thesis, we will provide a proof-of-concept attack against speech recognition system. Speech recognition is basic and essential to automatic speech recognition (ASR) systems, and we demonstrate that an adversary can embed malicious voice commands into regular songs, and these embedded commands can be recognized by the ASR system but are unnoticeable to humans. This is a serious security concern for voice assistants. In Chapter 4 and 5, we will explain our attack approach in detail and in Chapter 6 we will provide with evaluations. Specifically, we use the gradient-independent genetic algorithm to do the attack approach and reach an overall success rate of 58%. By calculating the Pearson Correlation Coefficient (PCC) between the crafted song and the original one, we show that the similarity between them is over 95%, making it hard to be noticed by humans.

Chapter 2
Related Work

There are some existing attacks against ASR systems, but most of them are focused on attacking DeepSpeech, which is a state-of-the-art speech recognition system developed using end-to-end deep learning [13]. End-to-end allows it to directly convert the complex input data (e.g. the raw audio) to the simple output (e.g. the readable text). It skips composing words with phonemes and does not use Hidden Markov Model (HMM) at all. Instead, it uses Connectionist Temporal Classification (CTC) loss function to score the transcripts produced from the Deep Neueral Network (DNN). So the target for attacks against DeepSpeech is clear and easy to understand; just minimize the loss function between the adversarial sample and the malicious command. Indeed, many attacks successfully attacked DeepSpeech [9] [5] [28], but after manually testing all available samples generated by these attacks, we found that none of them can be recognized as the embedded command by Kaldi. Kaldi is a speech recognition toolkit which is widely used in research area. Besides, Kaldi has a better accuracy compared with DeepSpeech - Kaldi provides Word Error Rate (WER) of 4.28% whereas DeepSpeech gives 5.83% on Librispeech clean data [21]. Moreover, DeepSpeech has been implemented using TensorFlow [4], which is highly popular among deep learning areas, making attacks against DeepSpeech easier and much more convenient.

Hidden Voice Command [8] is another attack approach, but it is based on GMM-based acoustic models and is harmless against modern DNN-based systems. Dolphin attack [33] successfully embedded commands into ultrasonic, which is inaudible for humans, but the device is too expensive and can be eliminated by enhanced hardware such as a microphone.

CommanderSong [32] and Adversarial Attacks Against ASR Systems via Psychoacoustic Hiding [26] were the only two attack approaches against Kaldi until now. The latter mainly focused on using psychoacoustic hiding to make the attack more stealthy but did not concentrate much on how to fool the DNN system. CommanderSong raised the idea of embedding malicious commands in songs and this idea highly inspired our research. By using a gradient algorithm (genetic algorithm), we successfully crafted a song which can be recognized as the malicious command.

Chapter 3

Background

3.1 Speech Recognition

In this section, we will describe the overall process of speech recognition and an important model which is closely related to the attack: Hidden Markov Model (HMM) [24].

3.1.1 The Overall Process of Speech Recognition

Automatic speech recognition (ASR) is a technique which aims to let a machine understand what a human is saying, more specifically, given an audio input (e.g. a recording) and transfer it to text. Figure 3.1 shows the overall process of speech recognition which can be regarded as two parts: extract the features from the raw model then use acoustic modeling and language modeling to decode it to text.

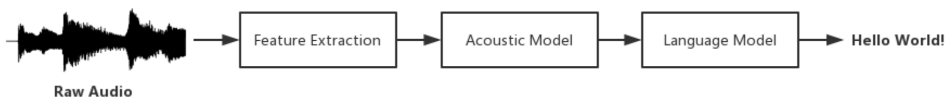


Figure 3.1: The Overall Process of Speech Recognition.

The feature extraction is also called "audio preprocessing" in the research area. It can convert the raw audio data per frame (e.g. 12ms) to a multi-dimension vector. These vectors are independent with each other while being salient, thus they must contain essential information which is useful for decoding. There are many existing ways of preprocessing, such as Mel Frequency Cepstral Coefficient (MFCC) [20], Linear Predictive Coefficients (LPC) [16] and Perceptual linear predictive (PLP) [15]. In this thesis, we choose MFCC because it is most widely used in open source speech recognition tool kits.

The acoustic model is used during the acoustic feature extraction. The outputs are acoustic features, phonemes or tri-phones, something that can be connected to the atomic parts of a speech. By linking them together we can get a small unit composing the speech, like a single word. Acoustic model is based on deep neural networks.

The language model, simply referred to as "grammar", defines how words can be connected with each other. In some cases, it can be defined by a set of rules, e.g. in a helpdesk menu navigation system or when you want to understand simple commands. In more generic cases however, such as speech transcription, a statistic approach is more often used. In the latter case, the language model is a large list of word tuples (n-grams) with assigned probabilities. For instance “good” has a higher possibility of aligned with “morning” or “afternoon”, instead of “breakfast” or “lunch”.

As for how to use the two kinds of models, we will discuss in Chapter 4.

3.1.2 Hidden Markov Model (HMM)

Hidden Markov Model (HMM) is a statistical Markov model in which the system being modelled is assumed to be a Markov process with unobserved (i.e. hidden) states. It is a fundamental and essential concept in speech recognition. A HMM has several states ranging from 1,2,...N and there exists a unique transition probability between each pair (e.g. $p_{i,j}$ is the probability of transition between state i and j).

In speech recognition problems, we stipulate each word is composed of several phonemes, and phoneme is the smallest unit. For each phoneme, there exists several HMM-states, and a series of the transitions between these states can define a phoneme. Given a sequence of features extracted from the raw audio (e.g. MFCC feature), the acoustic model and language model can produce the most likely sequence of HMM-states, which then can be transferred to words and sentences. By crafting the original song, we can generate a HMM-state sequence which can be assigned to the malicious command and successfully fool the ASR system but remain inconspicuous to humans.

We will explain the HMM topology in Kaldi and the way to craft the original song in Chapter 4 and Chapter 5.

3.2 Threat to Deep Neural Network

Deep neural networks (DNN) [14] are highly useful and reliable in image recognition and speech recognition. They can reach an assuring result by implementing a complex computation where all parameters have been optimized after guaranteed training and iteration, while their flexibility and variety can make them distinguished in many areas: from image recognition and colorization to the prediction of time-series problems.

But researchers have found that they are vulnerable to carefully-designed adversarial samples. An adversarial sample is an input which is designed in order to fool the DNN system. Although the convincing learning ability of DNNs can make them achieve an overall high-level performance, it also causes them to learn unreasonable examples which can lead to a confusing and unsatisfying result [27]. When handling with image classification problems, they misclassify samples which differ only slightly from the original image and different DNNs with different structures trained on different datasets can misclassify the same adversarial sample. This has been demonstrated [12] and the example is shown in Figure 3.2.

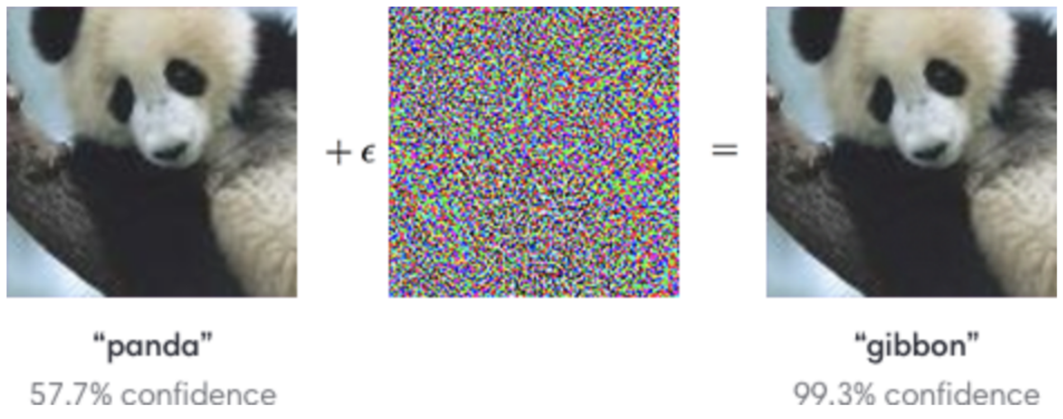


Figure 3.2: Adversarial Samples Against Image Recognition [12].

These samples can be generated by adding small perturbations to the original input, and these unnoticeable noises can mislead the network to do a wrong classification but will not capture human's awareness. As we can see, the differences between two pandas are impossible to be noticed by a human's eyes, but the DNN makes a mistake with a high percentage of confidence. This brings in serious security concerns, e.g. a vicious attacker can mislead an automatic driving system to misdeem a stop sign as other denotes, which may cause serious traffic accidents [22].

Much work has been down on image inputs, but few have focused in other domains, such as audio. We investigate adversarial attacks that can also be done on speech recognition: we aim to embed malicious commands in a song in the form of adversarial samples that a human interprets as an ordinary song while the automatic speech recognition (ASR) system (e.g. Apple Siri) can decode it as the malicious command such as "Hey Siri unlock the phone" which is shown in Figure 3.3.

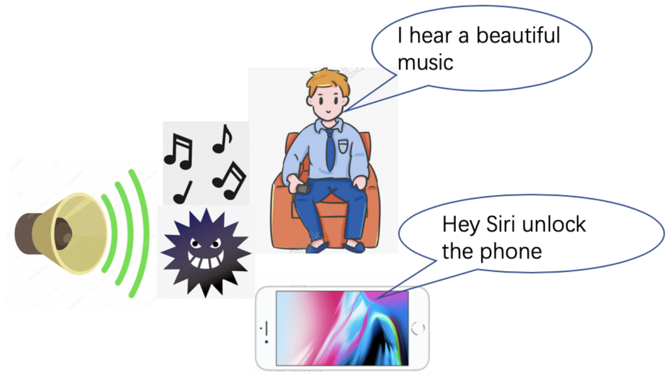


Figure 3.3: Adversarial Samples Fool the ASR but Remain Unnoticable to Human.

The main reason is that image itself has less background distribution, but a small perturbation to voice commands may be easily be recognized as just simple noise and then filtered by well-trained ASR systems, which makes the attack powerless. So our attack focuses on two points: first is to ensure the adversarial sample can successfully mislead the DNN system without being filtered out; second is to disguise the sample so that it has little difference with the original input, making it hard for a human to notice. We aim to attack Kaldi, which is a popular open source speech recognition toolkit in research area.

Chapter 4

Kaldi Speech Recognition Toolkit

In this chapter, we will give a brief introduction of Kaldi and explain its principle.

4.1 Introduction to Kaldi

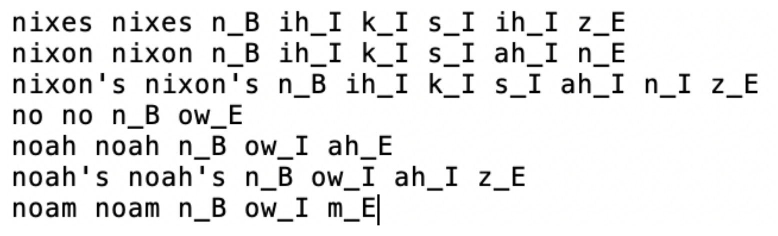
Kaldi is an open-source toolkit used for speech recognition based on DNN-HMM. It's written in C++ and licensed under the Apache License v2.0, developed by the researchers at John Hopkins University [23].

Kaldi is one of the most popular ASR systems among researchers, and it's used by major companies like IBM [6] and Microsoft [31]. Recent studies also have proved that Kaldi is also used in commercial products such as Amazon Echo. Compared with other open-source speech recognition toolkits like Sphinx [19], developed by CMU and HTK, not only does Kaldi have an overall better performance, it is also more popular in research areas (it has a citation of 2792 according to Google Scholar and 5539 stars on GitHub whereas Sphinx has a citation of 1073 and 1099 stars on Github). As mentioned above, there are few attacks that focus on Kaldi, and other attacks based on other toolkits all proved powerless against it.

4.2 HMM Topology and Transition Model

In this section we will discuss the HMM topology and the transition model in Kaldi.

From Figure 4.1 we can see that phoneme is the smallest unit composing a word.



```
nixes nixes n_B ih_I k_I s_I ih_I z_E
nixon nixon n_B ih_I k_I s_I ah_I n_E
nixon's nixon's n_B ih_I k_I s_I ah_I n_I z_E
no no n_B ow_E
noah noah n_B ow_I ah_E
noah's noah's n_B ow_I ah_I z_E
noam noam n_B ow_I m_E
```

Figure 4.1: Phonemes Composing Words.

For each phoneme, there are three HMM states and a series of transitions among those states can confirm a phoneme. Figure 4.2 shows the HMM topology for each phoneme in Kaldi:

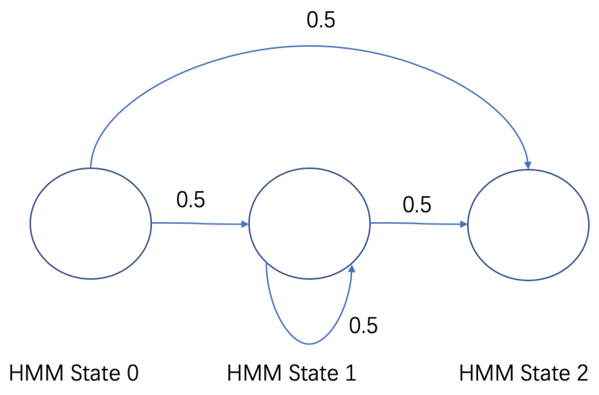


Figure 4.2: The HMM Topology of Kaldi.

The three HMM states can be referred to as HMM state 0,1 and 2. At HMM state 0, it has an equal possibility of transfer to state 1 or 2, and at HMM state 1, it has a fair probability of transfer to state 2 or self-loop.

The transition model contains three major elements: HMM state, pdf-id and transition-id. Now we will state the concept of pdf-id and transition-id.

A transition-id is used to index a specific transition between two HMM states of a same phoneme, and a sequence of them directly determines which phoneme is the final decoding result. From a sequence of transition-ids, we can get the information about the transition between HMM states and further identify a specific phoneme.

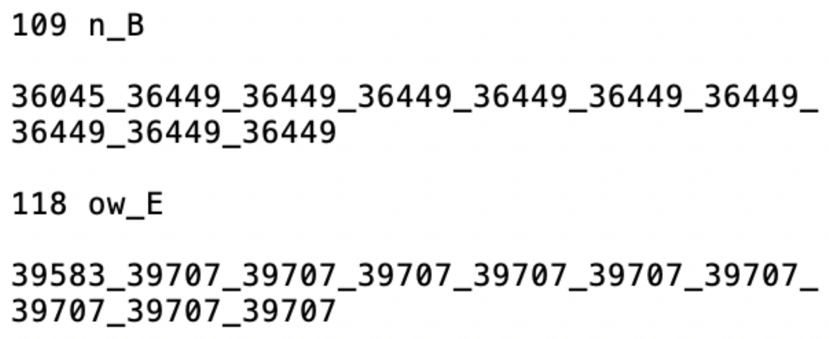
To confirm the most-likely sequence of HMM states, we need to use the pre-trained probability density function (p.d.f) of the DNN-based acoustic model to calculate the probability of all HMM states of the acoustic input signal. The p.d.f is indexed using pdf-id, and for each pdf-id, they are mapped with a specific HMM state while also connected with a transition-id. Therefore, the output of the DNN are probabilities of each HMM state at each frame. It's a $m*n$ matrix where m represents the total number of frames of the audio input while n is the total number of pdf-ids of the acoustic model. The number at *ith* line and *jth* column means the probability of pdf-id j at frame i.

Table 4.1 states the relationship between HMM state, pdf-id and transition-id clearly.

phoneme	**HMM state**	**Pdf-id**	**Transition-id**	**Transition**
n_B	0	307	36045	0 to 1
			36046	0 to 2
n_B	1	1692	36449	self-loop
			36450	1 to 2
ow_E	0	2725	39583	0 to 1
			39584	0 to 2
ow_E	1	89	39707	self-loop
			39708	1 to 2

Table 4.1: Transition Model in Kaldi.

Figure 4.3 is the decoding result of decoding the word "no".



```
109 n_B

36045_36449_36449_36449_36449_36449_36449_
36449_36449_36449

118 ow_E

39583_39707_39707_39707_39707_39707_39707_
39707_39707_39707
```

Figure 4.3: The Decode Result of "No".

Although the transition-id is calculated from the likelihood of pdf-id, after we obtained a specific sequence of transition-id that can determine the phoneme, we can retrodict the sequence of pdf-id referring to the transition model in Table 4.1. So for the phoneme “n_B”, the pdf-id sequence should be

$$[307, 1692, 1692, 1692, 1692, 1692, 1692, 1692, 1692, 1692],$$

and the pdf-id sequence for phoneme “ow_E” should be

$$[2725, 89, 89, 89, 89, 89, 89, 89, 89, 89].$$

Therefore, if the output posterior probability matrix of an audio input has a high possibility generating the pdf-id sequence above, then it will also have a high possibility of decoding result of “n_B” and “ow_E”, which then leads to the word “no”. Thus the target of the attack is to add small perturbations to the original song and crafting a song that can generate the target pdf-id sequence.

In this thesis, we use the pre-trained Aspire Chain Model as the decoding model. It’s trained on Fisher English and has been augmented with impulse responses and noises to create multi-condition training. It has the best overall performance comparing with other models provided by Kaldi.

Chapter 5

Attack Approach

In this chapter, we will describe the way we generate the crafted song that can reach the same decoding result with the original song while maintaining a high similarity. The overall process of the attack approach is shown in Figure 5.1.

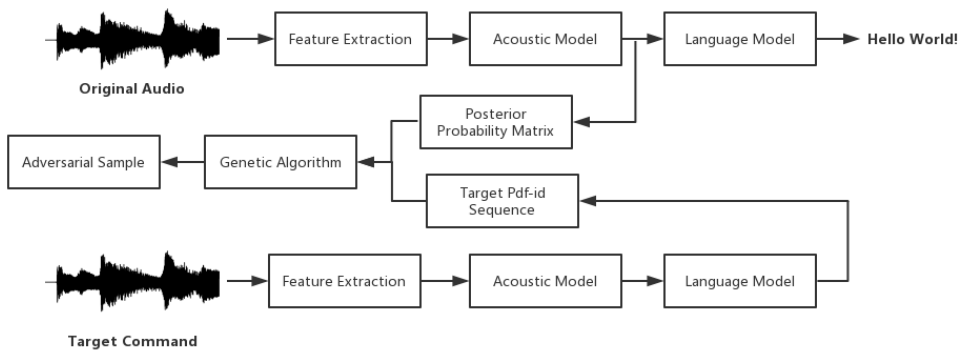


Figure 5.1: Overall Attack Process.

5.1 Making Original Song Recognized as Malicious Command

As we discussed in Chapter 4, in order to decode the crafted song as the malicious command, we need to ensure the posterior probability matrix generated by the song has a high probability of producing a specific pdf-id sequence. This pdf-id sequence can then transfer to a transition-id sequence that can identify a particular phoneme. After getting a clear comprehension of the decoding process of Kaldi, before we begin to generate the sample, we need to use Kaldi to achieve the following goals, described in the sections below.

5.1.1 Derivation of Malicious Command's Pdf-id Sequences

First, we get the decoding result of the malicious command, which is a sequence of transition-ids. Then according to the relationship between the transition-id and the pdf-id provided by the transition model, we get the pdf-id sequence of the malicious command. Before generating the crafted song, we first get the target pdf-id sequence p.

By digging deeper into the transition model, we found that even if the phoneme, the HMM state and the transition are all the same, there exists different transition-ids and pdf-ids at the same state. The results are shown in Table 5.1:

phoneme	**HMM state**	**Pdf-id**	**Transition-id**	**Transition**
ow_E	1	57	39705	self-loop
			39706	1 to 2
ow_E	1	89	39707	self-loop
			39708	1 to 2
ow_E	1	129	39709	self-loop
			39710	1 to 2
ow_E	1	134	39711	self-loop
			39712	1 to 2

Table 5.1: Different Transition-ids Represent Same Stage.

Given the transition-id sequence

$$p = [39583, 39707, 39707, 39707, 39707, 39707, 39707, 39707, 39707, 39707],$$

if we replace 39707 with 39705,39709 or 39711, the final decoding result will still be the same. We can replace a transition-id with any other transition-id, as long as they have the same phoneme, HMM state and transition. By replacing same-stage transition-id, we can get a total different pdf-id sequence. For example, transition-id sequence

$$[39583, 39707, 39707, 39707, 39707, 39707, 39707, 39707, 39707, 39707]$$

refers to pdf-id sequence

$$[2725, 89, 89, 89, 89, 89, 89, 89, 89, 89],$$

but the transition-id sequence

$$[39583, 39707, 39705, 39705, 39707, 39709, 39711, 39709, 39707, 39711]$$

will also be decoded as phoneme "ow_E" while the pdf-id sequence related to it is

$$[2725, 89, 57, 57, 89, 129, 134, 129, 89, 134].$$

Different pdf-id sequences may differ significantly from each other, but they will lead to the same final decoding result. With a specific transition-id sequence of the malicious command and the transition model, we can derive thousands of different pdf-id sequences that can be regarded as "target pdf-id sequence". We store all these qualified sequences in a list, which will be used for the *scoring* segment in the genetic algorithm.

5.1.2 Most Likely pdf-id Sequence Derivation

Second, we use Kaldi to decode the original song and get the posterior probability matrix calculated by DNN. From the matrix, we can get the highest-probability pdf-id sequence of the original song. As we mentioned in Chapter 4, the matrix has m lines, and n columns, where m represents the number of frames and n represents the total number of pdf-id. Suppose there is a matrix A, and we let $a_{i,j}$ be the element at the ith row and jth column. Then $a_{i,j}$ means the probability of pdf-id j at the frame i. For frame n, we can get the highest probability pdf-id p_n which equals to

$$p_n = \underset{j}{\operatorname{argmax}}\, a_{i,j}.$$

Then for each frame, we can get the pdf-id with the highest probability. By connecting them together, we can also obtain a sequence of pdf-ids, which means the most likely pdf-id sequence of the original song based on the calculation of DNN.

We set function f to represent the relationship between an original song and the most-likely pdf-id sequence. That is

$$f(x(t)) = p,$$

where $x(t)$ means the original song and p is the pdf-id sequence.

5.1.3 Calculating the Distance Between Two Sequences

So now we have two pdf-id sequences: one is the most-likely sequence of the original song, while the other is a certain sequence of the malicious command. In order to recognize the original song as the malicious command, we come up with an idea: by adding noises to the original song, we let the most likely sequence of pdf-id generated by it be close, or even equal to the pdf-id sequence of the command. We minimize the Euclidean Distance between the two sequences. The distance L is calculated using the formula below:

$$L(a, b) = \sqrt{\sum_{i=1}^{n} (a_i - b_i)^2},$$

where a is the most likely pdf-id sequence of the original song and b is one of the pdf-id sequences of the target command.

We aim to minimise the distance between the two sequences. We try to find the $x(t)$ which equals to

$$\underset{x(t)}{\operatorname{argmin}}\, L(f(x(t)), b).$$

Thus, the crafted song $x(t)$ can be recognized as the malicious command. We use the gradient-independent algorithm, genetic algorithm to generate the crafted song.

5.2 Generating Adversarial Sample based on Genetic Algorithm

In this section, we will give a brief introduction of Genetic Algorithm and explain every step of it in detail.

5.2.1 Background of Genetic Algorithm

The genetic algorithm is a method used for solving optimization problems based on the principle of natural selection. It's commonly used to generate high-quality solutions by relying on bio-inspired operators such as mutation, crossover and selection [25]. At each step, the genetic algorithm selects individuals which have a better performance from the current population to be parents and uses them to produce children for the next generation. For each generation, it only keeps individuals who have the best score and pairs them with other children to produce the next generation. Over successive generations, the population "evolves" toward an optimal solution. It is instrumental in solving a variety of optimization problems that are not well suited for standard optimization algorithms, including problems in which the objective function is discontinuous, non-differentiable, stochastic, or highly nonlinear. Figure 5.2 shows the process of genetic algorithm, and the pseudocode of the genetic algorithm is shown in Algorithm 1.

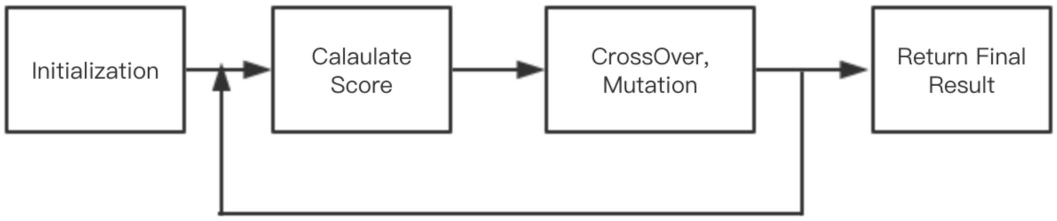


Figure 5.2: Process of Genetic Algorithm.

Algorithm 1 Genetic Algorithm

```
Input: Audio Input $X$, Target Sequence $Y$
Output: Adversarial Sample $X^{'}$
1: function GENETIC($X, Y$)
2:     $poulation \leftarrow addNoise(X) * populationSize$
3:     while $iter < maxIter$ and $L(f(x), y)$ != 0 do
4:         for each individual X in population do
5:             $scores \leftarrow getScore(X, y)$
6:         end for
7:         $bestSample \leftarrow population[argmax(scores)]$
8:         $children \leftarrow Mutation(Crossover(population))$
9:         $newPopulation \leftarrow best + children$
10:        $MutationP \leftarrow newMuatationP$
11:        $iter \leftarrow iter + 1$
12:    end while
13:    return $bestSample$
14: end function
```

Now we will explain the process of the algorithm in detail.

- Given an original input, we generate a number of *populationSize* samples by adding random noise in it. *populationSize* is the size of the population for each generation.

- When the iteration time is smaller than the $maxNumberOfIteration$, we calculate the *score* of each individual in the current population. We only keep the individual who has the best *score*.

- Using *crossover* and *mutation* based on natural selection, we generate the amount of $populationSize - 1$ children and make them and the best-score-individual as the next generation.

- Keep iterating until one of the individuals reaches the best solution or reaches the max iteration time.

We will now describe the following function: *score*, *crossover* and *mutation.*

5.2.2 Crafted Song Initialization

When implementing the genetic algorithm, we need to generate the initial population first. By adding random noise to the original input separately, we get the initial population.

We use the third-python library scipy to convert the original song into an array which can be easily crafted [18]. Using one of its functions, we can get the sample rate and the data in a .wav file. The data is returned as a numpy array [29] with a data-type determined for the file. It's usually an array that contains integers range from -32768 to 32768. We name this array as *audioArray.*

For each integer i in the audio array, we randomly generate another integer in the range (*-i/10,i/10*). This integer is considered as the random noise, and we name it *noiseArray.* By adding the two arrays together, we get the audio-array of the crafted song, and write it back to a .wav file to test its performance.

The original audio array:

$$[1630, 456, 34, 675, 23, \ldots, 612, 2340, 87, 100].$$

The noise array:

$$[-105, -16, 0, 33, 1, \ldots, 40, 2, -8, -6].$$

The crafted audio array:

$$[1525, -440, 34, 708, 24, \ldots, 652, 2342, 79, 94].$$

We define this as the function *addNoise.*

5.2.3 Scoring the Similarity Between the Crafted Song and Malicious Command

To calculate the score for each individual in the population, we first calculate the distances of the pdf-id sequence generated by it with all of the pdf-id sequences which can lead to the malicious command. We assume *x(t)* is the audio-array input, and P is the list which contains all possible pdf-id sequences. And L_i is the distance between $x(t)$ and the *ith* element in the list which is p_i. So that

$$L_i = L(x(t), p_i),$$

and for all of the L_i we find the minimum value L_{min}. The lower the L_{min} is, the closer the crafted song is to the malicious command, and the better performance it has. As a result, we set the score function to be

$$score = e^{-L_{min}}.$$

Thus we calculate the score of all individuals and they are used to choose the best performing individual and calculate the probability of being chosen at the crossover step.

5.2.4 Iterative Refinement Based on Crossover and Mutation

Crossover is the step to generate children for the next generation. Two of the individuals in the current generation will be chosen as the parents and generate a child for the next generation. The parents are selected randomly but the higher score they have, the higher probability they will be chosen. The likelihood of an individual to be chosen as a parent is calculated using the *softmax* function. It's a function that takes as input a vector of K real numbers (in our task K equals to the population size, and the numbers are scores), and normalizes it into a probability distribution consisting of K probabilities. The formula is described as follows:

$$P(s_i) = \frac{e^{s_i}}{\sum_{i=1}^{K} e^{s_i}}.$$

After selecting the two audio arrays a and b as the parents, we use this formula to generate the child audio array c:

$$c_i = p * a_i + (1 - p) * b_i,$$

where p is a float randomly generated between 0 and 1. In this way the child carries both parents' characteristics. Figure 5.3 shows the process of crossover.

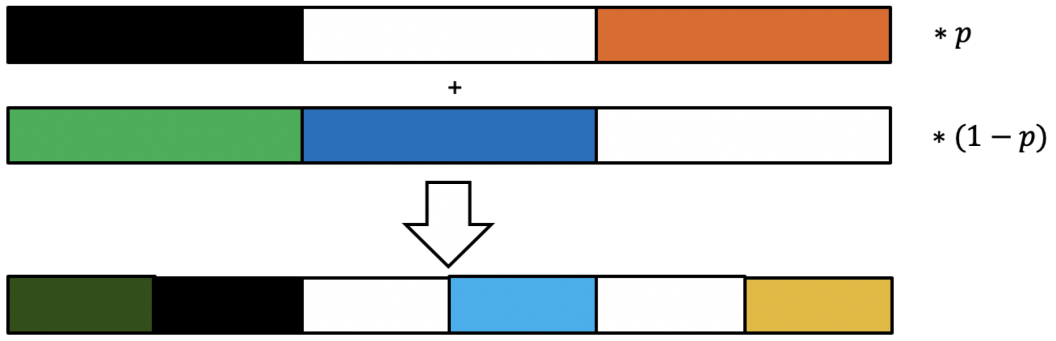


Figure 5.3: The Process of Crossover.

Mutation is an essential step because if we only generate the children using crossover, after many times of iteration, all of the children will tend to be the same and will not produce better samples.

We set a mutation probability *mutation_p*, and for each child, it has the probability of mutation after crossover. If after one iteration, the best score of the population didn't increase(no better sample is generated), we increase the *mutation_p* slightly. Once the best score increases, we set the *mutation_p* to its original value. Mutation is similar to the first step of the algorithm: generate a noise array and add it to the original audio-array. But for mutation, we set the range to be *(-i/5,i/5)* while adding a limitation that each element in the array only has a possibility of 50% to mutate.

Chapter 6

Evaluation

In this chapter, we will present the experimental results of our attack approach. We evaluate our attack on two aspects: the success rate of the crafted song correctly recognized as the malicious command, and the similarity between them.

6.1 Environmental Setup

We use three audio clips as the original input. They vary from a cut from a popular song “Good Time” by Owl City & Carly Rae Jepsen [17], or just pure music “Take me hands” by Daishi Dance [10], or a voice command generated by the text speech engine - a man speaking a single word. We set the target command to be single words “yes” or “no”.

We use TTSREADER [2] as the text-to-speech engine to generate malicious commands. It can read out loud any text with natural sounding speech synthesizers, and it’s free and simple, directly usable from a web browser.

We directly feed the crafted song into Kaldi to get the recognition result. We use Google Cloud Compute Engine to generate these songs. The virtual machine has a memory of 16GB. If we set the max iteration time to be 2000, it will cost around 12 hours to generate one sample, while a max iteration time of 3000 costs nearly 18 hours. The reason why it takes so long is that for each time we calculate the score for each individual, we need to use Kaldi to do the calculation, and this costs much time and memory. Existing attacks fully restored the function of Kaldi in python using TensorFlow, and this implementation makes the calculation mainly rely on GPU, which will significantly decrease the calculation time. The huge size of the DNN model imposes a practical limit on our experiments, and we consider larger-scale experiments as our future work which will discuss in Chapter 7.

6.2 Recognition Success Rate

We define success as the recognition of the crafted song exactly as the malicious command. Otherwise, we claim it's a failure. For instance, we set the malicious command to be "yes" or "no", then even if the crafted song is recognized as "yeah" or "nope", it's also a failure. The result of our attack is shown in Table 6.1. We observe a overall success rate of 58%.

Audio Type	Sample Number	Iteration	Success Rate
Pure Music	10	2000	80%
	10	3000	90%
Pop Music	10	2000	40%
	10	3000	70%
Voice Command	5	2000	0%
	5	3000	20%

Table 6.1: Recognition Success Rate.

We can see that pure music has the highest success rate. We think this is because there are no human voices in it, and it's easy to embed commands into pure music. As for popular songs, since the human voice is not that clear (people won't even be able to hear lyrics clearly some times), it's also suitable as a carrier for malicious commands. But when it comes to the pure voice command generated by the text-to-speech engine, the success rate decreases largely: it nearly falls to 0, only one sample luckily fools Kaldi. Meanwhile, the noise added to it can be heard much more clearly than those in music.

The pdf-likelihood is much higher in the posterior probability matrix computed from a raw voice command compare to popular music. Because the command is clear to hear, the model is well trained so the probability of correct pdf-ids are much higher than others, making it hard to make the malicious-pdf-ids have a higher possibility.

6.3 Similarity Between the Original and Crafted Song

We use the Pearson Correlation Coefficient (PCC) to calculate the similarity between the adversarial sample and original song. We first calculate the covariance between them. Covariance is a measurement of the relationship between two variables. The covariance can be calculated using the forumla below:

$$Cov(x,y) = \frac{1}{m}\sum_{i=1}^{m}(x_i - \overline{x})(y_i - \overline{y}).$$

Then we calculate the PCC. PCC is a measurement of the linear correlation between two variables X and Y. It has a value between +1 and 1, where 1 is total positive linear correlation, 0 is no linear correlation, and 1 is total negative linear correlation. It is widely used in the sciences.

PCC is calculated using the forumla below:

$$p_{x,y} = \frac{cov(x,y)}{D(x)D(y)}.$$

while $D(x)$ is the standard deviation of x.

The results are shown in Table 6.2:

Audio Type	Sample Number	Iteration	Similarity
Pure Music	10	2000	99%
	10	3000	97%
Pop Music	10	2000	98%
	10	3000	96%
Voice Command	5	2000	96%
	5	3000	94%

Table 6.2: The Similarity Between the Original and Crafted Song.

The similarity between the original audio input and the adversarial sample is relatively high, which is over 95%. This makes the noise hard for human to notice.

From Figure 6.1, we can see that the original audio and adversarial samples have little difference when using the pure music as the audio input, and the perturbation add to it is relatively small. The similarity between them is 99%.

Figure 6.2 is the waveform of the pure voice command. We can see that the noise is much more obvious than the previous one. The similarity is 96%.

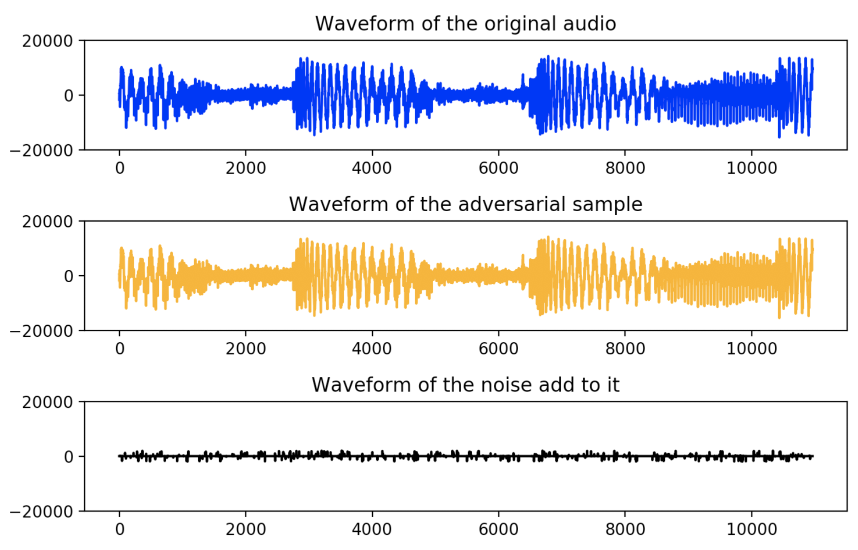


Figure 6.1: Similarity Between the Original Song and Crafted Song - Pure Music.

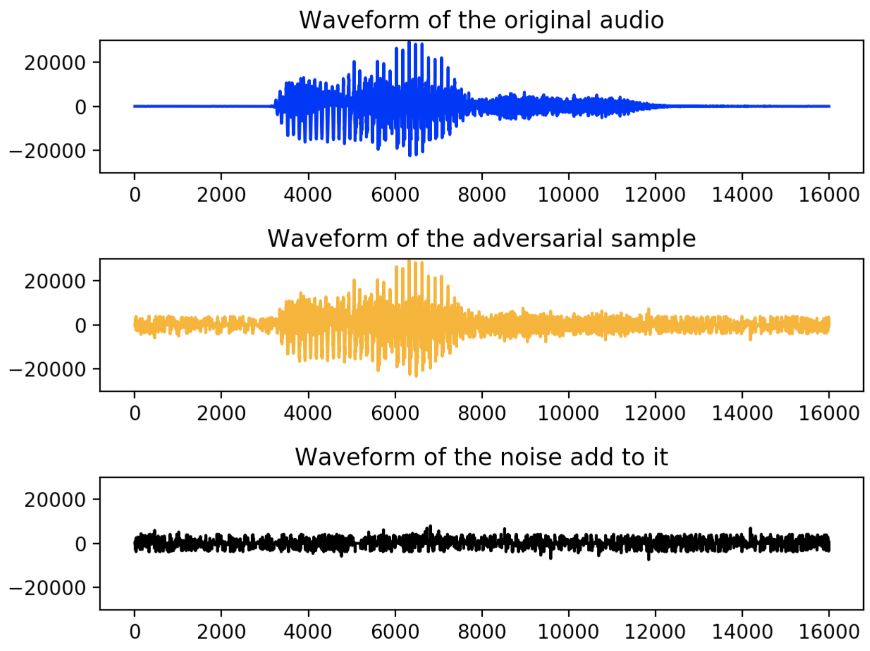


Figure 6.2: Similarity Between the Original Song and Crafted Song - Voice Command.

Chapter 7

Future Work

In this chapter, we discuss the existing problems and the future work of the experiment.

7.1 Generating Adversarial Sample Using Gradient Descent

Although the genetic algorithm can produce successful adversarial samples that can cheat the ASR system, the success rate of it is not that high, and it's way too time-consuming. We thought of introducing a new way to craft the song: gradient descent [7].

Gradient descent is an optimization algorithm used to minimize some function by iteratively moving in the direction of the steepest descent as defined by the negative sign of the gradient. It is usually used to update the parameters of a machine learning model - like DNN. Given an initial input (e.g. an audio-array), the algorithm will follow the direction which reduces the distance between the two pdf-id sequences most quickly.

But to make this algorithm perform better, we need to introduce the back propagation [11]. Back propagation is a method to improve the performance of gradient descent - it will calculate the gradient based on the weights of the model, and give feedback to the loss function to update weights to minimize it. In a neural network, it requires all the layer's activate functions to be differentiable.

In the genetic algorithm, we set the distance between the two pdf-id sequences as the loss function and try to minimize it. But we use Kaldi to calculate the output of the DNN - and since it's highly-integrated, if we call Kaldi every time to do the calculation, it's impossible to calculate the gradient of the function. Furthermore, we also can not use the back propagation method since it's non-differentiable.

Therefore we have an initial thought of applying the gradient-method by this way: rebuild the DNN model in tensorflow, restore all functions of the original DNN model in Kaldi. In order to rebuild it, we need to know the parameter of it first. Thus, we analyze the source code of Kaldi again and find out the structure and parameters of the DNN: it's a time-delay neural network (TDNN) [30].

The hidden layer of time-delay neural networks is not only related to the current input, but also connected with future inputs. The structure of a TDNN is shown in Figure 7.1.

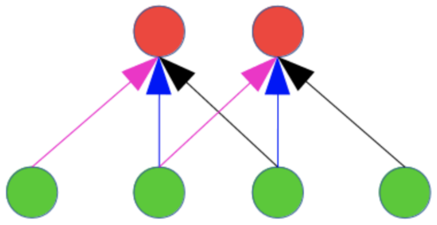


Figure 7.1: The Structure of Time-Delay Neural Network.

The green nodes are input data which can be regarded as MFCC features at each frame. We can see in this figure the time-delay is set to 2, while the red nodes are calculated by three continuous time data points, which are represented by green nodes. It also shares the same weights: same colour means same weights. Since each input data shares the same weight to the next-level nodes, it's easy to train, and it's time-invariant.

We are using TensorFlow to restore the DNN model and will use gradient descent to craft the song to acquire a more accurate result and also higher efficiency. Since the model is huge, we still require some time to finish it.

7.2 Speaker Verification

Our work focuses on speech recognition, but we also think that our work can be extended to speaker verification. Speaker verification is also widely used in commercial products, such as in voice assistants like Apple's Siri, since it can only be activated if the owner's voice is detected. So does there exist a way to fool the verification system by adding small perturbations to it like we've done to the ASR system?

The answer is probably "yes" since Kaldi has the function to implement speaker verification: it also uses DNN to produce a vector which further leads to the confirmation of one specific speaker. But instead of posterior probability matrix, this kind of DNN generate a vector which is named as “xvector” [34]. This model has been raised recently (in 2018), and there are no existing attacks targeting this model. Since it’s also based on DNN, we think it’s worth a try to extend our attack to speaker verification area.

7.3 Attack over Air

In our experiment, we directly feed the crafted song to Kaldi, but when playing them over the air as recordings, the recording can not be recognized as the malicious command again - the success rate is 0%.

The recording device caused differences between audio signals, and noises in the air also contribute to foiling the attack. The perturbation we add to the original song makes the attack successful, but noises in the air disturb its effect. To solve this problem, we plan to build a random-noise model and add it to the original song as the new audio input in the future study.

Chapter 8

Conclusion

In this thesis, we demonstrate that an adversary sample can embed malicious voice commands into regular songs, and these embedded commands can be recognized by the ASR system but are unnoticeable to humans. This is a serious security concern for voice assistants. We describled the principle of Kaldi and how we came up with an idea to use its decoding theory to create the adversarial sample. We use the gradient-independent genetic algorithm to do the attack approach and reach an overall success rate of 58%. Furthermore, we raise the idea that our attack can be improved by implementing gradient-descent algorithms and can be extended to speaker verification and attacks over the air.

References

[1] Nearly half of americans use digital voice assistants, mostly on their smartphones. Accessed 12 December 2017. https://www.pewresearch.org/fact-tank/2017/12/12/nearly-half-of-americans-use-digital-voice-assistants-mostly-on-their-smartphones/.

[2] Text to speech online. https://ttsreader.com/.

[3] Voice assistant consumer adoption report 2018. Accessed 12 December 2018. https://voicebot.ai/voice-assistant-consumer-adoption-report-2018/.

[4] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283, 2016.

[5] Moustafa Alzantot, Bharathan Balaji, and Mani Srivastava. Did you hear that? adversarial examples against automatic speech recognition. *arXiv preprint arXiv:1801.00554*, 2018.

[6] K. Audhkhasi, B. Kingsbury, B. Ramabhadran, G. Saon, and M. Picheny. Building competitive direct acoustics-to-word models for english conversational speech recognition. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4759–4763, April 2018.

[7] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[8] Nicholas Carlini, Pratyush Mishra, Tavish Vaidya, Yuankai Zhang, Micah Sherr, Clay Shields, David Wagner, and Wenchao Zhou. Hidden voice commands. In *25th {USENIX} Security Symposium ({USENIX} Security 16)*, pages 513–530, 2016.

[9] Nicholas Carlini and David Wagner. Audio adversarial examples: Targeted attacks on speech-to-text. In *2018 IEEE Security and Privacy Workshops (SPW)*, pages 1–7. IEEE, 2018.

[10] Daishi Dance. Take me hand. In *WONDER Tourism*, 2012.

[11] Anthony TC Goh. Back-propagation neural networks for modeling complex systems. *Artificial Intelligence in Engineering*, 9(3):143–151, 1995.

[12] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[13] Awni Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, et al. Deep speech: Scaling up end-to-end speech recognition. *arXiv preprint arXiv:1412.5567*, 2014.

[14] Simon Haykin. *Neural networks*, volume 2. Prentice hall New York, 1994.

[15] Hynek Hermansky. Perceptual linear predictive (plp) analysis of speech. *the Journal of the Acoustical Society of America*, 87(4):1738–1752, 1990.

[16] Fumitada Itakura. Line spectrum representation of linear predictor coefficients of speech signals. *The Journal of the Acoustical Society of America*, 57(S1):S35–S35, 1975.

[17] Owl City & Carly Rae Jepsen. Good time. In *Good Time*, 2012.

[18] Eric Jones, Travis Oliphant, and Pearu Peterson. {SciPy}: Open source scientific tools for {Python}. 2014.

[19] Paul Lamere, Philip Kwok, Evandro Gouvea, Bhiksha Raj, Rita Singh, William Walker, Manfred Warmuth, and Peter Wolf. The cmu sphinx-4 speech recognition system. In *IEEE Intl. Conf. on Acoustics, Speech and Signal Processing (ICASSP 2003), Hong Kong*, volume 1, pages 2–5, 2003.

[20] Lindasalwa Muda, Mumtaj Begam, and Irraivan Elamvazuthi. Voice recognition algorithms using mel frequency cepstral coefficient (mfcc) and dynamic time warping (dtw) techniques. *arXiv preprint arXiv:1003.4083*, 2010.

[21] Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur. 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. IEEE, 2015.

[22] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. Practical black-box attacks against deep learning systems using adversarial examples. *arXiv preprint arXiv:1602.02697*, 1(2):3, 2016.

[23] Daniel Povey, Arnab Ghoshal, Gilles Boulianne, Lukas Burget, Ondrej Glembek, Nagendra Goel, Mirko Hannemann, Petr Motlicek, Yanmin Qian, Petr Schwarz, Jan Silovsky, Georg Stemmer, and Karel Vesely. The kaldi speech recognition toolkit. 2011. IEEE Catalog No.: CFP11SRW-USB.

[24] Lawrence R Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[25] Javad Sadeghia, Saeid Sadeghib, and Seyed Taghi Akhavan Niakic. Optimizing a hybrid vendor-managed inventory and transportation problem with fuzzy demand: An improved particle swarm optimization algorithm.

[26] Lea Schönherr, Katharina Kohls, Steffen Zeiler, Thorsten Holz, and Dorothea Kolossa. Adversarial attacks against automatic speech recognition systems via psychoacoustic hiding. *arXiv preprint arXiv:1808.05665*, 2018.

[27] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[28] Rohan Taori, Amog Kamsetty, Brenton Chu, and Nikita Vemuri. Targeted adversarial examples for black box audio systems. *arXiv preprint arXiv:1805.07820*, 2018.

[29] Stefan Van Der Walt, S Chris Colbert, and Gael Varoquaux. The numpy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22, 2011.

[30] Alexander Waibel, Toshiyuki Hanazawa, Geoffrey Hinton, Kiyohiro Shikano, and Kevin J Lang. Phoneme recognition using time-delay neural networks. *Backpropagation: Theory, Architectures and Applications*, pages 35–61, 1995.

[31] W. Xiong, L. Wu, F. Alleva, J. Droppo, X. Huang, and A. Stolcke. The microsoft 2017 conversational speech recognition system. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5934–5938, April 2018.

[32] Xuejing Yuan, Yuxuan Chen, Yue Zhao, Yunhui Long, Xiaokang Liu, Kai Chen, Shengzhi Zhang, Heqing Huang, Xiaofeng Wang, and Carl A Gunter. Commandersong: A systematic approach for practical adversarial voice recognition. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pages 49–64, 2018.

[33] Guoming Zhang, Chen Yan, Xiaoyu Ji, Tianchen Zhang, Taimin Zhang, and Wenyuan Xu. Dolphinattack: Inaudible voice commands. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 103–117. ACM, 2017.

[34] Yingke Zhu, Tom Ko, David Snyder, Brian Mak, and Daniel Povey. Self-attentive speaker embeddings for text-independent speaker verification. In *Proc. Interspeech*, pages 3573–3577, 2018.