Sound event detection with recurrent neural networks

Lu, C.

Award date:
2019

Awarding institution:
Technische Universität Berlin

Technische Universität Berlin

Institute of Software Engineering and Theoretical Computer Science
Neural Information Processing

Fakultät IV Marchstr.23
10587 Berlin
http://www.ni.tu-berlin.de

Master Thesis

Sound event detection with recurrent neural networks

Changbin Lu

Matriculation Number: 0395571
Email:lzxqcc@gmail.com
Berlin, 25th October 2018

Supervised by
Prof. Dr. Klaus Obermayer

Assistant Supervisors
Drs.Moritz Augustin, Ivo Trowitzsch

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und eigenhändig sowie ohne unerlaubte fremde Hilfe und ausschließlich unter Verwendung der aufgeführten Quellen und Hilfsmittel angefertigt habe

Berlin, 25th October 2018

. . . . . .

Abstract

Auditory is a sense of human to perceive the surrounding world in which sound event often occurs with multiple sound sources. The human hearing can easily segregate and recognize which sound source is present at the moment. The sound event is distinguished and located by machine identification, which highly depends on the algorithms and models.

In the past decades, particular interest was placed on how to improve the simulation system for sound event detection. Classical classification methods have been widely used in Environmental Sound Recognition (ESR) like Least Absolute Shrinkage and Selection Operator (Lasso), Gaussian Mix Models (GMM) and Hidden Markov Model (HMM) which have achieved good performance. However, these methods do not consider or only a small part of recent information of time series which is due to the limited computing power in the past. With the development of GPU computing, now it is possible to train complex and deep neural network in an efficient way. Recurrent neural network (RNN) is one of the most learnable methods to simulate natural listening behaviour to process a continuous digital signal. In this work, our experiments based on an RNN architecture for sound event detection in real-world binaural auditory scenes. In order to process the continuously income audio signal while utilizing part of recent received information as a teaching reference to the current input signal. The Long Short-term memory (LSTM) is used which can capture long-term dependency and are evolved to various architectures for different usage. Stacked LSTM is an extension of simple LSTM model in the deep neural network which has multiple hidden states to capture the more sophisticated temporal structure of the long sequence. The main component of our targeted architecture is a stacked LSTM followed with one DNN network.

In the present study, we first describe the advantage of neural networks in simulating the human hearing compared to the classical classification model and then compare the performance between block interprete and instant interprete label. To fulfill the comparison between different types of label, each label has its optimal architecture respectively, so we need to optimize all related hyperparameters on complete training data twice. Tuning deep neural network is a huge time and computing resources consumption, especially when the objective space is constructed from multiple dimensions. Consider the comparability and stability during the optimization phase, we choose random search as a reliable and efficient method and adapt it more flexible by incorporating the idea of successive halve searching.

Zusammenfassung

Auditory ist ein Menschenempfinden, um die umgebende Welt wahrzunehmen, in der Schallereignisse oft mit mehreren Schallquellen auftreten. Das menschliche Gehör kann leicht trennen und erkennen, welche Schallquelle momentan vorhanden ist. Das Klangereignis wird durch eine Maschinenidentifikation unterschieden und lokalisiert, die stark von den Algorithmen und Modellen abhängt.

In den vergangenen Jahrzehnten wurde besonderes Interesse auf die Verbesserung des Simulationssystems für die Erkennung von Schallereignissen gerichtet. Klassische Klassifizierungsmethoden wurden in der Environmental Sound Recognition (ESR), wie zum Beispiel der Minimalschrumpf- und Selektionsoperator (Lasso), den Gaussian Mix-Modellen (GMM) und dem Hidden-Markov-Modell (HMM) weit verbreitet, die eine gute Leistung erreicht haben. Diese Verfahren berücksichtigen jedoch nicht oder nur einen kleinen Teil der neueren Zeitreiheninformationen, die auf die begrenzte Rechenleistung in der Vergangenheit zurückzuführen sind. Mit der Entwicklung von GPU-Computing ist es nun möglich, komplexe und tiefe neuronale Netze effizient zu trainieren. Das rekurrente neuronale Netzwerk (RNN) ist eine der am meisten erlernbaren Methoden, um natürliches Hörverhalten zu simulieren, um ein kontinuierliches digitales Signal zu verarbeiten. In dieser Arbeit basieren unsere Experimente auf einer RNN-Architektur zur Erkennung von Schallereignissen in realen binauralen Hörszenen. Um das kontinuierlich ankommende Audiosignal zu verarbeiten, während ein Teil der kürzlich empfangenen Information als Lehrreferenz auf das aktuelle Eingangssignal verwendet wird. Es wird der Long-Short-Term-Memory (LSTM) verwendet, der Langzeitabhängigkeiten erfassen kann und zu verschiedenen Architekturen für unterschiedliche Zwecke entwickelt wird. Gestapeltes LSTM ist eine Erweiterung des einfachen LSTM-Modells im tiefen neuronalen Netzwerk, das mehrere versteckte Zustände aufweist, um die komplexere zeitliche Struktur der langen Sequenz zu erfassen. Der Hauptbestandteil unserer gezielten Architektur ist ein gestapeltes LSTM gefolgt von einem DNN-Netzwerk.

In der vorliegenden Studie beschreiben wir zunächst den Vorteil neuronaler Netze bei der Simulation des menschlichen Gehörs im Vergleich zum klassischen Klassifikationsmodell und vergleichen dann die Leistung zwischen Blockinterpret und Instant-Interpreter-Label. Um den Vergleich zwischen verschiedenen Etikettentypen zu erfüllen, hat jedes Etikett seine optimale Architektur. Daher müssen wir alle zugehörigen Hyperparameter zweimal auf vollständige Trainingsdaten optimieren. Das Tuning eines tiefen neuronalen Netzwerks ist ein enormer Zeit- und Rechenressourcenverbrauch, insbesondere wenn der Zielraum aus mehreren Dimensionen aufgebaut ist. Berücksichtigen Sie die Vergleichbarkeit und Stabilität während der Optimierungsphase, wählen wir die Zufallssuche als zuverlässige und effiziente Methode und passen Sie sie flexibler an, indem Sie die Idee der sukzessiven Halbwertsuche integrieren.

Contents

1 Introduction

The sound is one of the principal sources of human perception of the environment, and it is also an important feature that reflects human behaviour. The sound here refers not only to speech but also to other environmental sound events. A large variety of environmental sound events are usually seen in our daily lives, such as speech, dog bark, baby crying, the presence of fire, etc. Moreover, human usually utilizes hearing perception to aware what happened around them by capturing acoustic wave. The acoustic wave acts on the auditory organ to make it feel excited and send the incoming information through the impulse of the auditory nerve. Human perception of speech signals is a typical and complicated process of signal and information processing, and biologically it has visible multi-level or deep-level processing structures.

Deep learning is essentially the use of deep neural networks for multi-layer nonlinear feature extraction and transformation. Many vision-based recognition systems heavily relies on the quality of the visual signal which is unstable since the lightness, location, and angle changed over time. Although there still exist interference factors for the audio signal, like distractor sources and location, audio is less dependent on the environmental factors. For example, a perceptual challenge in the cocktail-party-like social environment can be solved by training on manually design mixture audio data. Since the rapid development of big data and high-performance computing, deep learning has achieved great success in the field of machine audition. The reason is that the bare input signal has continuity in the perceptual problems such as speech. For example, a sentence that is slightly louder, a bit sharper, a little bit more, a bit more noise, and a change in tone, it is still this sentence for humans. That is, the input signal can be continuously changed in its small neighbourhood without changing its meaning, or the input point can make small perturbations without changing its meaning. Therefore, the auditory system can be used in robotic navigation, assistive robotics, and other automatic sound alarm system.

Recognizing environmental sounds is an application of acoustic event classification in specific environments. The achievements in this research area can be used to analyse acoustic mixture environments, such as monitoring people who is screaming for help, the fire hazard of warehouse, the sound of closing door or the alarm is going on. Sound event recognition includes two tasks: sound event detection and sound event classification. Sound event detection refers to identifying and locating events in real time in a continuous stream of sound signal. Sound events classification refers to identifying events that have been separated from the sound stream. It converts it into meaningful information symbol based on an understanding of the different sounds in the sound field.

In real-world aural environments, ESR faces more difficulties than other tasks due to the various possible background sounds. For example, the sounds appearing in different environments are complex and diverse, with a full frequency band, multiple sounds superimposed, reflections, significant differences in sound length, a wide dynamic range, and a wide variety of vocal sound sources.

1.1 Motivation

Regarding the previous works on sound recognition, the majority of work has focused on automatic speech recognition (ASR). ESR still has many challenges, especially in noisy and reverberant environments. The environment of ASR is relatively pure since the Signal-to-noise ratio (SNR) is high. It indicates there is more useful information than unwanted noise. Conversely, the low SNR has worse specification. Although many classic classifiers are widely used in ESR problem, its performance is still far from the ideal level. Due to the superposition of multiple sound sources, the background environment is quite complex and unstructured. When the SNR is low, the wide variety of noises causes the distribution of each phoneme to be shifted in all directions, and the data distribution becomes complicated, which requiring a more sophisticated model. Sometimes it is not even possible to distinguish between them.

The reliable detection of environmental sound event affected by following challenges. 1) The multiple sound events occurs currently which make recognition difficult, especially when the task is about categorized the type of sound event in acoustic mixture scene. The co-occurrence sound events are overlapping themselves which disturb the state of dominated sound. 2) The unstructured environments include not only multiple targeted sound classes but also some unknown sound sources which require the model insensitive to the outlier and equip discriminate ability. 3) Poor adaptability, mainly reflect in the strong dependence on the environment quality. Unlike the speech, which is captured from a specific position, environmental sound can happen anywhere around the sound collector. Under this circumstance, to detect multiple sound occurrences from different directions, the capability of the model representation should be as close to human hearing as possible.

Human hearing is capable of recognizing specific sound source from an unstructured and mixture audio stream, such as the piano sound mixed with dog bark, the conversation mixed with a baby crying. Moreover, it also relatively robust to the mixture sound event in which the targeted source in one direction while the distractor source in another direction. Although sometimes human hearing also make mistake, the sound collector is much more sensitive to this situation when the distractor source is overwhelming the targeted one. All of the challenges above require the model should be sufficiently representative to simulate human hearing.

1.2 Objective

The classical classification method for any audio related recognition problem requires only a small amount of computing power to learn the distribution of each phoneme. However, it discards much information of time series because the model incapable of capturing the correlation between each timeframe and the potential influence along feature depth. Theoretical investigation shows the internal feature representations learned by the deep neural networks (DNNs) that is capable of extracting the intrinsic pattern from spectral and temporal dimensions. This pattern is much more distinguished on the hyperplane of the projection space in which only need a simple softmax layer can discriminate well. For modelling sequence to sequence pattern, RNN has a hidden internal state to process sequence from the previous input which is a natural connected function to bridge following events along the time series. In deep learning, RNN is specialized for sequence modelling. However, when choosing RNN as our primary network, training the model for sound event detection is generally more cumbersome since the vanishing or exploding gradient. This drawback of the recurrent neural network is eliminated by the proposal of LSTM. LSTM is most commonly used to model temporal variation and was extended to N-dimensions LSTM recently. We implement LDNN (stacked-LSTM connected with DNN) to build the recognition model which is composed by a stacked LSTM and followed with a multi-layer perceptron network (MLP) as output.

In a conclusion, the main objective of this thesis is to evaluate and compare the performance of block interprete and instant interprete labels upon the same LDNN architecture. Sound event in the auditory scene is labelled by two different ways, one is block interprete label which consider the latency and another one is instant interprete labels which is more close to actual scenario. To investigate the modelling capacity of LDNN for different types of label, exploring the optimal hyperparameters respectively in parameter space is essential for the fair of final model comparison. Apart from that, the variants of sound scenes requires the model is capable of generalizing in different kinds of testing scenes. So using all kinds of scene as training data is mandatory but at meanwhile it also maximizes the quantity of training time. Therefore, it requires an efficient and adaptable method to do hypeparameter optimization, such as random search or successive halve search. Although the later method can search more space at the same time, random search promise the fair of comparison when the learning rate is one of the hyperparameters. Consequently, our method for hyperparameter optimization consists both advantages from these two ideas and an estimate of the difficulty of evaluating different validation set.

1.3 Structure of this thesis

The rest of the thesis is organized as follows:

Chapter 2 Introduce the origins of neural networks and the concept of RNNs, LSTM,

and its variants. Then discuss recent work on exploring RNNs architecture in environmental sound recognition. It also includes currently used methods for utilizing the network and hyper-parameter optimization.

Chapter 3 Introduces the concept and theory of each component of LDNN, such as RNN, LSTM, stacked LSTM and grid LSTM.

Chapter 4 Illustrate the process of how we exploring and running different combinations of hyperparameter. And we obtain some insight by plotting these different combinations' result. And then using both best combination hyperparameter and averaged epoch number to get the final result on testing set..

Chapter 5 Discuss about the final results of two types of label and further discussion.

2 Fundamentals and Related Work

The study of sound event recognition, which aims to separate and recognize the mixture of sound sources present in an auditory scene, is broadly known in the literature as Computational Auditory Scene Analysis (CASA) [15]. CASA aims to enable computers to hear and understand audio content much as humans do. This chapter describes relevant technologies in the research area of environmental sound recognition, also includes the related work on the classification of acoustic scenes and events, the development of deep neural network and the mainstream hyperparameters search method.

2.1 The origin of Environmental Sound Detection

Several feature extraction methods have been used in the past to obtain a sufficient representation of audio signal. State-of-the-art audio recognition use various features in parallel, in order to obtain different aspects and applying them in a complementary way to achieve more accurate recognition. The commonly used feature is Mel frequency cepstral coefficients (MFCC) [15] which was introduced in the early 1980s which was inspired by human auditory perception. It better represents the perceptually relevant aspects of the short-term spectrum and also performs better in high-level frequency. In order to model the spectro-temporal patterns of time-frequency representations, 2D Gabor filters has been used to exploit spectro-temporal information [38]. Besides, some conventional feature extraction methods are limited in temporal integration such as the delta and double delta features. It [39] shown that the human auditory system decomposes an audio signal not only into its acoustic frequency but also into its amplitude modulation frequency. The human hearing is rely on the low frequency amplitude modulations. Therefore, the Amplitude Modulation Filter Bank is also an optional feature. Additionally, due to the recent success of self-learned feature of CNNs in computer vision, researchers have also made good progress in the extraction of audio features on spectrograms.

Environmental sounds contain many contextual cues, such as time, place, and level/-type of activity, that allow us to perceive the events that occur around us. How to use it on the machine has become a researcher's interest. The earliest research on the study of environmental sound recognition appeared in 1997 and had 68% overall classification accuracy, which was proposed by Sawhney and Maes [16]. This study used RNNs with 80 hidden layers to classify five different acoustic sources, including traffic, people, subway trains, general indoor/outdoor noise and other sounds. In 1998, Clarkson [17] simulated a real-world acoustic environment by using a wearable microphone to

collect audio in an office, supermarket or busy street and chose an HMM framework to identify specific auditory events such as the speakers, cars and shutting the door. By collecting surrounded audio data and predicting a sound event by predefined model, the contextual cues are given by a wearable computing application for the user. Additionally, researchers also conducted deep exploration and comparison of different classifiers for environmental sound recognition. Nishiura [18] proposed a new HMM composition method that composes a conventional speech HMM with the categorized environmental sound HMM for a selected category which outperformed the conventional HMM. Couvreur and laniary [19] use HMMs and multi-layer perceptrons to detect sounds and whistle squealing through scooters in an urban environment, achieving a 95% accuracy.

In the past, there were some conventional solutions to the ESR problem, such as the GMMs, HMMs, and KNN described above. These are commonly used in machine learning methods, but they are all classifiers with shallow structures. It has a good effect on simple problems or complete constraints, but when dealing with complex natural signals such as environmental sounds and natural languages, shallow models often fail to meet the requirements for recognition performance due to the lack of expressive capabilities and modeling capabilities for complex signals. A recent advance in DNNs was found to outperform conventional combinations of GMMs and HMMs. As the less sensitive to the input dimensionality of DNNs, it is easier and popular to expand the dimension of feature representation.

2.2 Neural Network

Since 2013, Institute of Electrical and Electronics Engineers Audio and Acoustic Signal Process (IEEE AASP) held the competition about Detection and Classification of Acoustic Scenes and Events (DCASE), aims to evaluate existing acoustic scenes detection methods. Recent submissions to the challenges (Sound event detection in real life audio, DCASE 2017) have suggested that the classical methods are being replaced by deep learning was not accidental, that they are different representations of different computing power on the same issue. The foundation of the theory of neural networks today originated in 1890, the psychologist William James published a monograph on the structure and function of human brain “Principles of Psychology.” He believes that a nerve cell can be stimulated to transmit to another nerve cell after being stimulated, and the activation of nerve cells is the result of all the input of the cell. The development of artificial neural networks experienced three periods: cybernetics from the 1940s to the 1960s, connectionism from the 1980s to the mid-1990s, and deep learning since 2006.

In 1943, the neuroscientist and control expert McCulloch and logician Pitts presented the mathematical description and structure of neurons. And it turns out that as long as there are enough simple neurons, any calculation function (M-P model) can be simulated if these neurons are interconnected and run synchronously. In the late 1940s, psychologist Donald Hebb proposed Hebbian learning based on the mechanism of neural

plasticity, which is considered as a typical unsupervised learning rule. In 1957, American psychologist Frank Rosenblatt proposes a neural network with a single-layer computing unit called the Perceptron, inspired by biological neural networks that process information from the human brain[20]. It is a simple two-layer associative networks in which the input pattern arriving at input layer are mapped directly to output layer. Such two layers network only contains input and output units but not hidden unit. Therefore, there is no internal representation between the input and output mapping and it only capable of learning linear problems. Standard multi-layer perceptron (MLP) is known as a neural network that has unlimited number of neurons in each hidden layer to approximate an arbitrary continuous (nonlinear) function to any desired degree of accuracy. At this moment, there is no way to train the hidden units. In 1969, Minisky and Papert discuss the bottleneck of the MLP and pointed out that theoretically it cannot be proved that it is meaningful to extend the perceptron model to a multi-layer network. Because for the nodes of each hidden layer, they do not have expected output, so it is impossible to train multi-layer perceptrons through learning rules. Consequently, research on artificial neural networks had stagnated, and other linear machine learning methods such as Support Vector Machine (SVM) was gradually exceeds that of neural networks. But the study of shallow networks and related theories has accumulated a good theoretical and empirical foundation for the further development of deep learning.

In 1974, Dr. Paul Werbos first proposed using Error Back Propagation (BP) which is essentially a chain rule of complex functions. In 1986, Rumelhart and Hinton used BP effectively trained some shallow networks[22], which solved the challenge of Minisky and Papert and rekindle the hope of neural networks. Its basic idea is that the learning process consists of two processes: positive propagation of signals and backward propagation of errors. BP aims to find a powerful synaptic weight adjustments rule to develop an internal structure that is appropriate for a particular task domain. Specifically, the hidden state should be capable of learning when being active to simulate the input-output pattern. It may be trapped in local minima when looking for the optimum value of gradient descent. In 1989, the MLP with as few as one hidden layer had been proved in [21] is indeed capable of universal approximation in a satisfactory scene. Hanki and Stinchcombe established a mapping function based on this MLP which implies any unsuccessful mapping must arise from insufficient learning, numbers of hidden units or the lack of a deterministic relationship between input and target.

Since 2000, with the rapid development of the Internet, although shallow learning networks have achieved great success, the disadvantages of shallow networks have become more obvious. For example, the independence assumptions of Hidden Markov Models (HMMs) is unrealistic and their representational capacity is limited by hidden states. Until 2006, deep learning was indeed applied to many pattern recognition fields such as speech processing and image recognition. Hinton's research [10] shows that it is entirely feasible to train a deep densely connected deep network such as Depp Belief Network. In the fine-tuning process, the BP algorithm is used to find the optimum global value as

accurately as possible over a good initial value range. The deep learning model is used by many scholars in the field of speech recognition because of its good representation of features and the accuracy of solving complex speech problems. Mohamed et al. [23] used DBN to identify phonemes on the TIMIT database and achieved a 23% error rate. It also include the exploration of effect of layer size in which the performance starts to plateau when adding more layers than three. Dahl et al. [24] proposed a method for constructing a new acoustic model combining pre-trained DNNs and HMMs, replacing the traditional GMM-HMM model, achieving an accuracy of 71.8%. However, almost all conditional generative models face a problem. If the current model's prediction is based only on the most recent inputs, and the inputs themselves are predicted by the model, then it is highly unlikely that it will recover from past mistakes. Having a longer memory has a stable effect because even if the network misunderstand its recent history, it can also review earlier rules. Consequently, saving longer-term memories is a more profound and effective solution.

Recurrent neural networks (RNNs) is a kind of neural networks (Rumelhart et al. 1986), which is dedicated to mapping sequential data, such as text, genomes, handwriting, the spoken word, or numerical times series data emanating from sensors and stock markets. It is a dynamic model that can be trained for time series generation by processing sequence one step at a time and predicting what the next most likely is. Unlike other neural networks, RNNs use their internal representation to perform a high dimensional interpolation between consecutive samples. It implements a mechanism similar to that of the human brain that retains a certain memory of processed information, unlike other types of neural networks that do not retain the memory of processed information. Goodfellow et al. [2] explain RNNs originated from an idea in machine learning and statistical models of 1980s: sharing parameters across different parts of a model, which makes it possible to extend the model to examples of different lengths of a sequence. It is a deep version of time-delay neural network which use convolution filter span over 1-D temporal sequence. It is a shallow representation than RNN since the output of convolution filter is an operation of a small pieces of neighbouring members of the input. The critical component of RNNs is their recurrent connections that it can store previous information and use it as part of the input for next step. This structure allows it to keep previous information through a very deep computational graph, but it also brings a problem when the interval of the dependency to be captured become longer. This is likely due to the insufficient use of these potentially compelling and suitable time-series models. Bengio [25] claimed when the time span of correlation increases, the highly unstable relationship between the parameters and the hidden layer causes the gradient descent fails to train the RNN properly. It is called vanishing or exploding [26] gradients where the weights and activation function affects the magnitude of the gradient to blows up or decays exponentially as it cycles around the recurrent connections. It is also related to a trade-off for long term or short term cell states.

Before Hochreiter and Schmidhuber [27] proposed a modified architecture call the Long

Short-term Memory (LSTM), the latest research results for the RNN model was often the same as random guessing. At the meantime, there were two main ways of incorporating previous information into sequence processing tasks: decompose inputs into overlapping time-windows and treat it as spatial, or use recurrent connections to build the relation directly. However, they both have drawbacks: time-windows size is highly dependent on task requirement, too small will underfitting or too large will overfitting; standard RNNs is known to have difficulty learning long-term information. The standard LSTM constructed by an RNN architecture with an internal memory cell which specializes in transmitting long-term information, along with four gates which allow the memory cell to store and control information flow over long periods of time. The internal memory cell is designed to have fixed linear dynamics with a self-connection of value one; then the gradient will not vanish or explode as it is back propagated through them. LSTM has been given state-of-the-art results in a variety of sequential problem. For example, Graves [28] successfully applied bidirectional LSTM to speech recognition and outperformed unidirectional LSTM which suggests both directions are equally important. LSTM is efficient to converge and also more accurate than both standard RNNs and time-windowed MLPs. Besides, they found out LSTM benefits more from longer target delays than RNNs since LSTM can make use of the extra context without suffering as much from the expense of maintaining previous inputs. Based on standard LSTM, the stacked LSTM [29] [33] is a deeper RNNs composed of multiple LSTM layers on top of each other which inspired by standard RNNs benefits from depth in space. This approach potentially allows each level of hidden states to operate information at a different timescale.

For deep RNNs, although increase the size of the recurrent layer increases memory capacity with a quadratic slow down, deepening networks in multiple dimensions can improve their representational efficiency and memory capacity with a linear complexity cost. However, all these architecture above only add RNN or LSTM cell along the temporal dimension. Since RNN can only be applied in one dimension in the past, it means that those 2D or multidimensional problems need to be compressed to 1D in advance, such as image recognition need to be processed line by line. It is intuitively to make the network have access to their surrounding context in all directions. Therefore, [30] proposed multi-dimensional RNN (MDRNNs) which adds hidden to hidden connection to every dimension of data. It is an extension of the bidirectional recurrent neural network (BRNNs) that includes two hidden layers connected to a single output layer. Therefore, it has access to both past and future context. Similar to BRNNs, MDRNNs contains 2^n separate hidden layers. In the feed-forward pass, each hidden layer at the network receives specific input and its activation from one time step back along with all dimensions of data. One-dimensional LSTM can be applied in MDRNNs by using multiple memory which corresponds to each dimension in the data with individual forget gate. Although the LSTM cell in MDRNNs receives multiple inputs and memory cells, it only produces one memory cell and one hidden state by linear accumulation which brings some numerical problems. As the number of the path in the grid grows with

the combination of the size of each dimension, the accumulation of memory cell will be unstable.

MDRNNs has multiple memory cells but not distinct for each dimension so that only applied to the temporal dimension. Various neural network architecture has been proposed in past five years to model correlations of the input signal in both frequency and time dimensions. For example, convolutional neural networks (CNNs), Frequency LSTMs [35], Time-Frequency LSTMs [30] and grid LSTMs [10]. Frequency and Time-Frequency LSTMs are alternatives to CNNs to model correlations in frequency which are more adaptive and robust to the input signal with noise conditions. Google Deep Mind [10] proposed a simple alternative that can solve the computational instability in MDRNNs by producing distinct memory cells for each dimension in the grid. Grid Long Short-term Memory (Gird-LSTM) is a network of LSTM cells can expand depth in any dimension of the network. One dimensional Grid-LSTM is very close to high way neural network where it is a feed-forward neural network with LSTM cell instead of transferring function such as tanh or relu. Two-dimensional Grid-LSTM is similar to stacked LSTMs with an extra memory cell in the vertical dimension. In this thesis, two-dimensional Grid-LSTM will be used as a part of the architecture.

Each of the above neural networks has its unique characteristics and has achieved tremendous success for large vocabulary continuous speech recognition tasks compared to conventional machine learning methods. Recently, research realized the advantage of the complementarity of CNNS, LSTMs, and DNNs can make up for their modeling limitations by combining variants of different neural networks into one unified architecture, like LSTMs followed by DNNs (LDNN), convolutional LDNN (CLDNN) and Grid LSTM followed by LDNN (GLDNN). These proposed architectures are designed to improve the capabilities of the RNNs. Pascanus and Bengio [40] proposed a deep transition RNNs and proved that the temporal modeling of RNNs could be improved if convert original input feature from lower level to higher level. Because the potential factors of variation within the input feature can be refined and make it easier to modeling. Although some people believe stacked RNNs is already deep, there are some other aspects of the model might be regarded as shallow. For instance, the transition between two consecutive hidden states at a same layer is shallow due to affine transformation followed by an element-wise nonlinearity. The implementations have already been applied like a few fully connected CNNs or Grid LSTMs.

2.3 Hyperparameter Optimization

As [4] stated, "In practice, one can usually do much better with a correct application of a commonplace algorithm than by sloppily applying an obscure algorithm. Correct application of an algorithm depends on mastering some fairly simple methodology". Successfully evaluating model capacity requires more than just knowing the advantage of algorithm. A good set of parameters determines the degree to which the represen-

tative capabilities of the architecture are reflected. When we compare the capacity of the architecture, we should find the best set of parameters of each architecture. The parameters defining the model attributes or governing the training process are called hyperparameters. For example, part of hyperparameters for a deep neural network is configuring how many hidden layers and neurons between input layer and output layer. According to the principle of No Free Lunch, there is no perfect set of hyperparameters for all models. So each model require running multiple trials in a single training job. Usually, the hyperparameter optimization of machine learning model is generally considered as a black box optimization problem. The choice of hyperparameters has a great influence on the final effect of the model. For example, a complex model may have better expressive ability to process different types of data, but it may also cause the gradient to disappear because of too many layers, and if the learning rate is too large, the convergence effect may be poor, and if the learning rate is too small, the convergence speed may be slow. The so-called black box problem is that we only see the input and output of the model in the process of tuning, we cannot obtain the gradient information of the model training process, nor can we assume that the model hyperparameters conforms to the convex optimization condition. Otherwise, we can find the optimal solution through the derivative or convex optimization method. Since the training process of the model is relatively expensive, we need a more accurate and efficient method to tune the hyperparameters.

Grid search has been widely used in empirical machine learning problem which can find a set of 'good' parameter values for a function. The grid search method divides the parameters to be searched into a grid in a particular spatial range and finds the optimal parameters by traversing all the points in the grid. This brute force method is sufficient when the optimization interval is large enough, and the step size is small enough. However, when the number of parameters is increasing, traversing all parameter groups in the grid would be expensive. What's more, cross-validation is used to assess the predictive performance of the models in each hyperparameter. In particular, the performance of trained models on new data can reduce overfitting to some extent. The motivation for using cross-validation techniques is that when we fit a model, we are applying it to a training data set. Without cross-validation, we only have information about how our model performs as part of training data. Ideally, we would like to see how the model behaves when it comes to new data about the predictive accuracy. Therefore, it is too expensive to apply in deep learning problem since exhaustive search include too many combinations when too many parameters are used in the neural network.

Yoshua Bengio [6] proved random search finds better hyperparameters at the same computational budget and has all practical advantages of grid search (conceptual simplicity, ease of implementation, trivial parallelism), especially in the case of a large number of parameters. Because not all hyper-parameters are equally essential to tune while grid search allocates too many trials to cover the subspace inefficiently. In contrast, random search sample far more evenly distributed in the subspace. When the number of trials

is the same as Grid search, the maximum value is generally greater. Of course, the variance is also greater but this does not affect the final result. Since the experiment can be stopped any time and the trials form a complete experiment. Moreover, new trials can be added or abandoned when computing resources become available. However, it is still a problem that how to measure test error of an experiment when many trials claim to be best. When using relative small validation set, there are two sources of uncertainty need to take into account, one is selecting best model by cross validation, another one is measuring the test set performance. It is important to all experiments in which many sets of hyperparameters achieve approximately the best validation set performance. Additionally, the sequential combination of manual and grid search from an expert is outperformed random search (Larochelle et al., 2007).

In recent years, Bayesian optimization attempt to improve the efficiency upon grid/random search. According to [7], a benchmark of over a hundred hyperparameter optimization datasets suggests that although Bayesian Optimization perhaps consistently outperform random sampling, they are negligible. Therefore, they proposed an alternative approach called hyperband to exploit the iterative algorithms of machine learning which applied for recent advances in pure-exploration algorithms for multi-armed bandits (Gittins, J. C. 1989). Multi-armed bandits is a problem in which a fixed, limited set of resources must be allocated between competing (alternative) choices in a way that maximizes their expected gain, when each choice's properties are only partially known at the time of allocation and may become better understood as time passes or by allocating resources to the choice. In contrast to treating the problem as a configuration selection problem, hyperband treat the problem as a configuration evaluation problem and randomly select the configuration. By computing more efficiently, hyperband look at more hyperparameter configurations not only makes up for the inefficiencies of the random search used for selection but also exceeds state-of-the-art Bayesian Optimization procedures.

Overall, tuning model is a necessary part in deep learning, a good set of hyperparameters plays the vital role for the efficiency and quality of performance. The grid search is inapplicable in deep learning since the demand for computing resources increases exponentially as the number of parameters increases. Compared to random search, the successive halves which also called Hyperband to broaden the search space while using the same amount of computing resources. After predefine the search space for each parameter, Hyperband will iteratively train each set of parameters from small epochs and pick up top sets with high performance to run vast epochs.

2.4 Training methods

There are some commonly used techniques when training RNN model:
Stochastic optimization: Adaptive moment estimation (Adam) was presented by Diederik Kingma [31] in 2015 which consists both advantages from AdaGrad and RMSProp. It

can update alpha weight during the learning process. It needs less memory and calculation efficiently.

Gradient clipping: Goodfellow [2] indicates the cliffs commonly occur in RNN. By limiting the magnitude of gradient, it can perform better in the vicinity of cliffs.

Regularization: Dropout is ineffective when applied to recurrent connections, as repeated random masks zero all hidden units in the limit. The most common solution is only to apply dropout to non-recurrent connections. Gal et al[4] proposed a variational inference based dropout method. From the Bayesian view, the RNN model treated as probability model and using same dropout mask at each time step for both inputs, outputs, and recurrent layers. Actually, it uses recurrent dropout via dropping weights.

L2 regularization: L2 is the sum of the square of the weights which used to prevent the coefficients to fit so perfectly to overfitting. λ_{norm} is a method to compute the L2 norm of a vector or tensor.

Early stopping: It is a regularization method to avoid overfitting when training a model with an iterative method such as gradient descent.

Cell initialization: The earliest deep learning began with repeated matrix multiplications. The most important feature of the orthogonal matrix for us is that all the eigenvalues of the orthogonal matrix have an absolute value of 1. This means that no matter how many repeated matrix multiplications are performed, the resulting matrix will not explode or disappear.

LSTM forget bias: Rafal [32]recommended using 1.0 as forget gate bias to retain more information.

Learning rate: [32] also states When the learning rate is low, then training is more reliable, but optimization will take much time because steps towards the minimum of the loss function are tiny. When the learning rate is high, then training may not converge or even diverge. Weight changes can be so big that the optimizer overshoots the minimum and makes the loss worse. It is better to start from a relatively large learning rate like 0.1, then try exponentially lower values like 0.01, 0.001, etc.

3 Methodology

In this chapter, the components and the architecture of LDNN were the drive of this thesis and are described in this chapter.

3.1 Recurrent neural network

RNN is a prevalent model for deep learning and have achieved good results in many problems of dealing with serialized input of variable length. The traditional feedforward network assumes that the input (output) sequences are independent of each other. So it usually targeted to fixed length input and learn separate parameter to be able to find all the rule of the input data separately at each position. However, this assumption does not hold in many practical scenarios, such as continuous text and speech recognition. There are two main difference between RNN and traditional feedforward network. One is that the RNN share parameters over time frame, it is consistent with a situation where a specific piece of information or event can occur at multiple sub interval within the input sequence. The other one is the feedback loop connect to their past decisions, taking their output moments as input. This sequence information is preserved in the hidden state of the RNN. As it progresses step by step to influence the processing of each new instance, the hidden state management spans many time steps. It finds the correlation between events that are separated by many moments, and these correlations are called "long-term dependencies" because the events downstream of the time depend on and are functions of one or more events that occur upstream in time.

Theoretically, RNN can handle any length of sequence data. But consider the fixed size vector that used to store the hidden state, it is intend to selective keep part of past data with more useful information. The general formula to represent the state$S^{(t)}$ is Equ 3.1.

$$S^{(t)} = f(S^{(t-1)}, X^{(t)}; \theta) \tag{3.1}$$

The following figure is a classical structure of RNN and it is the fundamental structure of LSTM network. The unfolded graph of RNN, shown in the right part of Fig 3.1, where each node is associated with one paricular input instance.

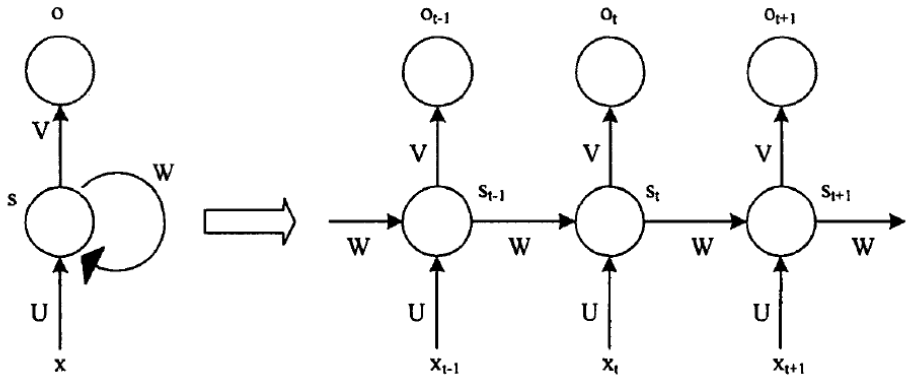

Figure 3.1: A Unfolded computational graph of Recurrent Neural Networks

(1) The input: X_t is the input of the network at time step t. For example, X_t can be a sample at the t time step in the corresponding input audio stream.

$$\alpha^t = W \cdot S_{t-1} + U \cdot X_t + b^{'} \tag{3.2}$$

(2) The hidden layer:S_t is the state of step t of the hidden layer which is the memory cell of the network. S_t is calculated based on the output of the current input layer and the state of the hidden layer in the previous step. The formula is below, where f is generally a nonlinear activation function such as tanh or ReLU.

$$S_t = f(\alpha^t) \tag{3.3}$$

(3) The output: O_t is the output of the network at time step t. For example, if the goal of the model is to predict the occurrence of sound events, the output is the probability for all sound time categories.

$$O_t = V \cdot S_t + b^{''} \tag{3.4}$$

(4) U, V, W are weight matrices respectively for input-to-hidden, hidden-to-output and hidden-to-hidden connections. It indicates the learned model always receive same size of input data.

The conventional learning algorithm for RNNs is Backpropagation through time (BPTT) which moves backward from the final error through outputs, weights, activation function and inputs of each layer to calculate partial derivatives by using chain rules. Those derivatives are then used by the objective function to adjust the weights. The phenomenon of exponential growth of errors is relatively small, and the vanishing gradient is very common in BPTT.

3.2 Long short-term memory

In 1997, a variation of the recurrent net with so-called Long Short-Term Memory units was proposed by Sepp Hochreiter and Juergen Schmidhuber[27] as a solution to the vanishing gradient problem. The LSTM is essentially no different from the general RNNs structure but contains extra cell and gates to make the error more constant during the backpropagation. These cell and gates act like connectionist neurons, they control information flow block or pass based on the weight of gate. Those weights, like the weights that modulate input and hidden states, are adjusted via the learning process. That is, the cells learn when to allow data to enter, leave or be deleted through the backpropagating error, and to adjust weights via gradient descent.

Generally, the only difference between standard LSTM and RNN is the hidden layer; the structure is called cells. Cells can be regarded as black box to save the saved state h_{t-1} before the current input x_t. These cells plus certain conditions determine which cells suppress and which cells are excited. They combine the previous state, current memory, and current input. It has been proved that this network structure is very effective in the problem of long sequence dependency. An overview of LSTM structure is shown in Fig 3.2 ,which is referenced [43].

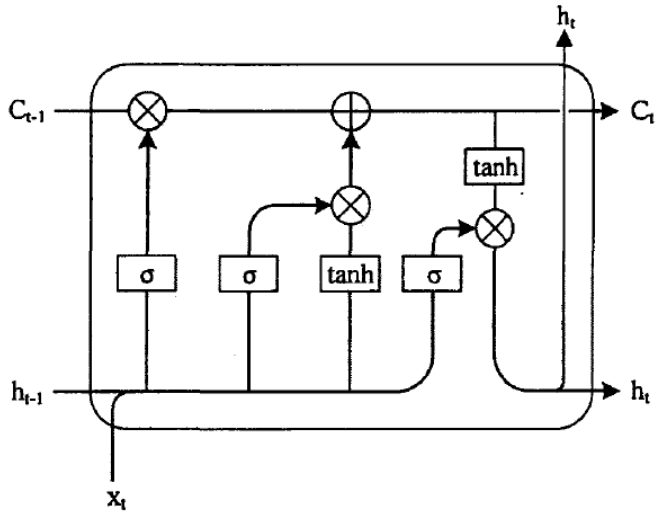

Figure 3.2: Long Short-term Memory Cell:

Cell state is the key component of LSTM. As shown in Fig 3.3 below, the cell state is updated and transmitted along a horizontal line and only involves linearly related operations. Therefore, the transmitted information will not be easily lost or changed

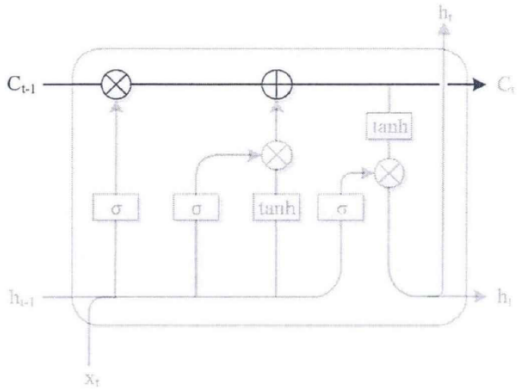

Figure 3.3: Update flow of cell state

The LSTM controls the discarding or addition of certain information from the cellular state through a structure called "gate" consisting of a Sigmoid neuron and an element-wise multiplication operation. The output of the Sigmoid layer is a value between 0 and 1, describing how much of information should be let through. When Sigmoid's output is 0, it means that "door" is closed and no information is passed; when it is 1, it means that "door" is open, allowing all information to pass. There are a total of three "doors" in the LSTM network element to control the updating and discarding of cell states, known as "forget gates," "input gates" and "output gates," respectively.

"Forget Gate" is used to control what information in the cell state should be discarded and the gate is oblivious. A value between 0 and 1 is calculated based on Eq.3.5 in which the output is depends on the hidden state of the previous time step h_{t-1}, the current input X_t and bias.

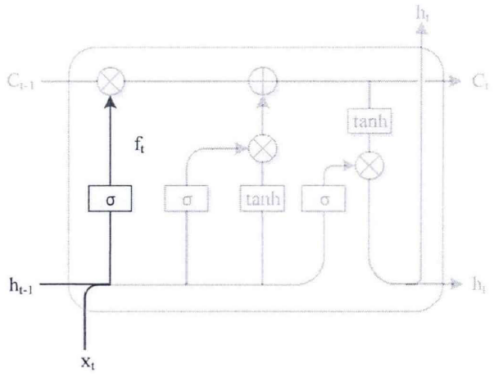

Figure 3.4: Forget Gate

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f) \tag{3.5}$$

For the input at time step t, there are two steps to decide what information will be added to the cell state. In the left part of Fig.3.5, a sigmoid layer called the "input gate layer" decides which values we will update in eq.3.6. Then a temporary cell state comes from a non-linear neural layer (tanh) calculated by the current input X_t and the hidden state counter h_{t-1} at the previous moment in eq.3.7.

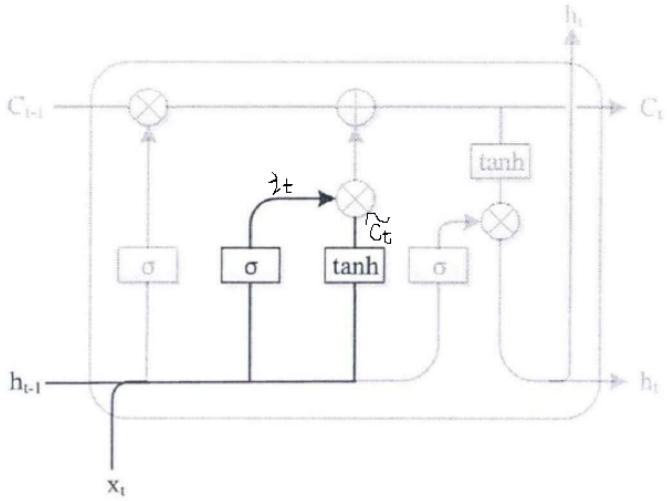

Figure 3.5: Input Gate

$$i_t = (W_i[h_{t-1}, x_t] + b_t) \tag{3.6}$$

$$\widetilde{C}_t = tanh(W_c[h_{t-1}, x_t] + b_c) \tag{3.7}$$

Then, the cell state of the LSTM network unit at time t is updated by two following steps in eq.3.8: In the first step, C_{t-1} is multiplied with f_t to represent the part of information forgotten by gate forgetting; the second step it that the temporary cell state $\widetilde{C}_t$ is multiplied with i_t to indicate that part of the information in the provisional cell state is updated into the cell state C_t.

$$C_t = f_t \odot c_t - 1 + i_t \odot \widetilde{C}_t \tag{3.8}$$

Finally, the "output gate" is used to control what information is output from the cell state. Its value is calculated by eq.3.9.

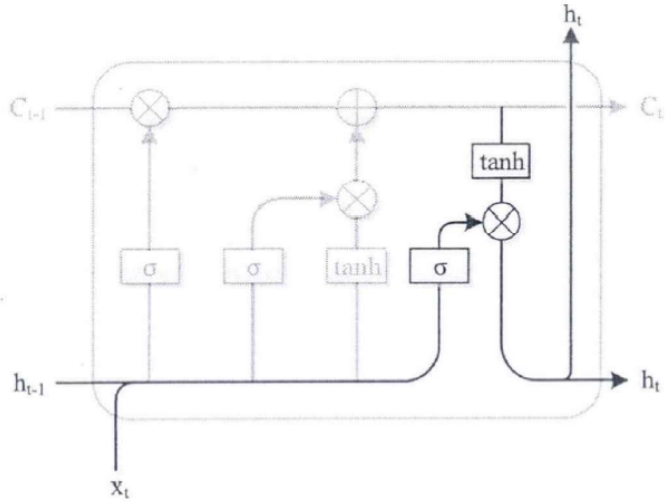

Figure 3.6: Output Gate

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \tag{3.9}$$

$$h_t = o_t \odot tanh(C_t) \tag{3.10}$$

3.2.1 Stacked Long Short-term Memory

Although there are a variety of LSTM variants proposed in recent years, like Gated Recurrent Unit (GRU), a large-scale analysis [14] of LSTM variant shows that none of the variants can improve upon the standard LSTM significantly. Thus, the standard LSTM is adopted in this comparison as an essential component of the network structure and introduced in this section. LSTM is inherently deep in the time since their memory cell is a function of all previously hidden states. Although there are not theoretically proof what is the additional power gained by the deeper networks, it was observed empirically that deep RNNs work better than shallower ones. Graves [33] has shown that the depth of the recurrent neural network was more important than the number of memory cells in a given layer to model representations. One way to enhance the neural network is to increase its depth, like vertically stacked LSTM. By stacking multiple LSTM layers on top of each other, hopefully, each layer can process part of the task we wish to solve and pass it on to the next. It is a process of learning a hierarchical representation of time-series data. This hierarchy of hidden layers enables more complex representation of the time-series data, capturing information at different scales.

The architecture of stacked LSTM is illustrated in Fig 3.7 below, in which the output of a LSTM hidden layer will be fed as the input into the subsequent LSTM hidden layer.

Output S_t from the lower LSTM layer is the input X_t of the upper LSTM layer. Thus it combines representative context in different levels.

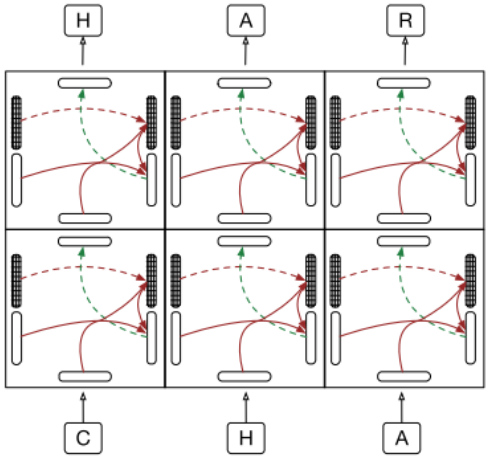

Figure 3.7: Stacked LSTM: the input and output present different classes; the white box is the hidden vector and the black box is the memory vector; the green dashed line bridge the output of lower LSTM layer as the input to the upper layer; the red solid line transport hidden state and the red dashed line transport memory state.

3.3 LDNN

The complementarity of LSTMs and DNNs in their modelling capabilities can be combined into a unified architecture. Because LSTM is good at modelling and DNN is suitable for converting LSTM output into target space. Usually the output of the last LSTM layer is connected to a fully connected DNN layers to make it deeper and also transform features into a more discriminative space that makes the output easier to classify. The additional layers between units and outputs are used to improve output prediction by reconstructing the learned representation from prior layers and create new representations at high levels of abstraction. Increasing the depth of the DNNs provides an alternate solution that requires fewer neurons and train faster. Ultimately, adding depth it is a type of representational optimization.

The LDNN architecture shown in Fig 3.8. In time step t, the input X_t is a 160-dimensional feature that consists of ratemaps and amplitude modulation spectrogram. By passing the input to network, the temporal correlation between each frame can be modelled. After that, the output of stacked LSTM layers is passed to a few fully connected DNN layers. These higher layers project hidden units' output into our targeted space in which the feature representations is more easily to classify. Since there will be more than one targeted class occur at the same time, the targeted classes are mutually

exclusive and softmax cannot be used to transfer them to probability. In the top of DNN, each class will be applied a sigmoid function.

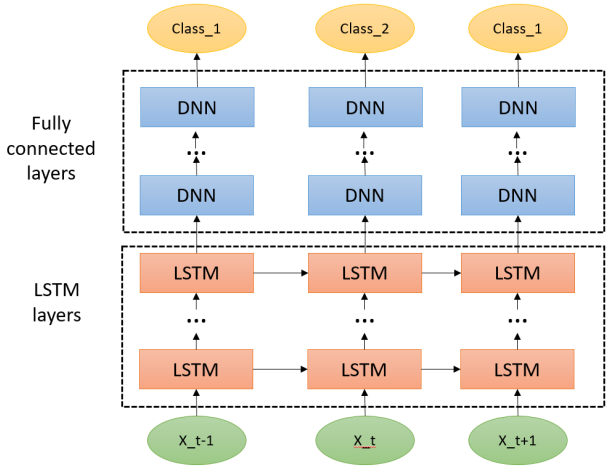

Figure 3.8: The framework of LDNN

3.4 Extension:Grid Long Short-term Memory

The state-of-the-art architecture of LSTM is Grid LSTM. 2D Grid LSTM differs from traditional stacked LSTM by adding LSTM cells along the depth dimension of the network as well as the temporal dimension. That is, each LSTM layer uses both a hidden state and a memory cell to communicate the subsequent layer. It gives the depth dimension the same gradient channel characteristics available along the temporal dimension, helping to mitigate the vanishing gradient problem in networks with many layers and allowing layers more flexible to dynamically select or ignore their inputs.

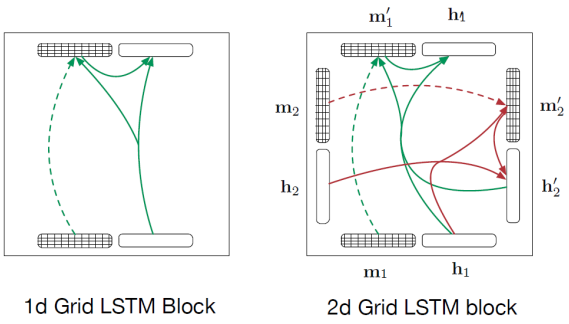

Figure 3.9: Grid LSTM block:the shaded block is the memory vector and the white block is hidden vectore; the green line presents the transformation along feature dimension and the red line presents the transformation along temporal dimension.

The structure of Grid-LSTM has an extra flow from lower layer to upper layer. For each dimension, it applies the standard LSTM mechanism across the respective dimension in which the weight matrices and states are distinct.

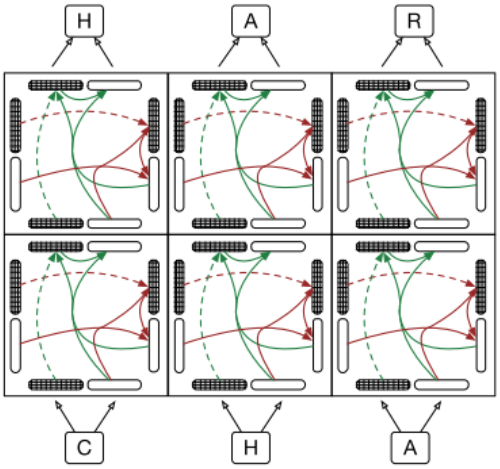

Figure 3.10: 2D Grid LSTM:the shaded block is the memory vector and the white block is hidden vectore; the green line presents the transformation along feature dimension and the red line presents the transformation along temporal dimension.

Stacked LSTM and 2D Grid LSTM have different ways to remember informations. In stacked LSTM, the information in the lower cell must pass through an output gate, a hyperbolic nonlinearity, an input gate and another hyperbolic nonlinearity to reach the upper layer. On the other hand, the grid LSTM network can write information to a lower cell, close the forget gate on it, carry it across multiple layers and then expose the information directly to higher layers. This replaces the prior path through multiple nonlinearities with a linear identity transformation, modulated by a series of forget gates. In a word, grid LSTM makes linear information flow along the depth of the network might be so beneficial.

4 Experiments

This chapter first describe how to generate the dataset and the features we have chosen, then introduces the algorithm we used for batch generation and hyperparameters optimization. At the end, we discuss the evaluation of optimal hyperparameter combination on testing set.

4.1 Data description

The dataset used in this thesis comes from the NIGENS database [36]. It was created by Ivo Trowitzsch [1] which provides audio files of fourteen different event classes: running engine, crash, footsteps, piano, barking dog, phone, knocking, burning fire, crying baby, alarm, female speech, male speech, screams, and a "general" class. There are two kinds of sounds in "general" class, one of which is nature sounds such as wind, rain or animals and the other one is artificial sounds such as honks and coughs. They both served as negative example for the classifiers and as distractor signals. In order to make our result be comparable to multi-conditional training in[1], here we use the same strategy to generate data which includes thirteen classes as detection target: 'alarm', 'baby', 'femaleSpeech', 'fire', 'crash', 'dog', 'engine', 'footsteps', 'knock', 'phone', 'piano', 'maleSpeech', 'femaleScreammaleScream'. All of these audio files are convolved into a two-channel ear signals by a binaural simulator of the TWO!EARS system[37].

4.1.1 Auditory scene generation

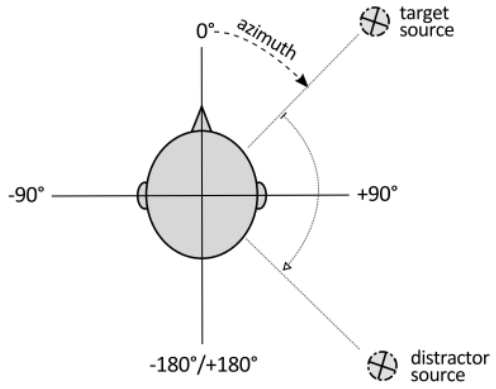

Figure 4.1: Binaural TWO!EARS system

In this work, the system renders four sets of binaural auditory scenes which composed of a single target source emitting sounds from all classes and multiple distractor sources emitting sounds only from general class. Scenes containing multiple diffuse ambient sound sources presented simultaneously as the setting in Fig 4.1:

1. Target source is located at five azimuth angles 0, 22.5, 45, 67.5, 90 (azimuths are given with respect to the direction of the nose of the binaural head. If the scene only contains single target source, it is regarded as clean sounds.

2. Distractor sound sources are located at the following combinations of azimuth angles:
   a) One distractor sound source: 135, 90, -135, 157.5, 0, 67.5 45, 180, -90, -112.5.
   b) Two distractor sound sources: [-90 67.5], [-67.5 -112.5].
   c) Three distractor sound sources: [157.5 22.5 90], [0 67.5 -22.5].
   d) Four distractor sound sources: [-180 112.5 22.5 90], [135 45 0 -90].

3. The binaural simulation was conducted separately for each distractor sound source with four different value of SNR 10 dB, 0dB, -10dB, -20dB. The target sound source is designated as SNR of inf dB.

4. Each target and distractor sound sources are mixed into the two-channel signal and then were used to generate mean-channel features. This feature is the auditory front-end representation were averaged over the left and right channels.

5. All scenes were clipped into 500ms overlapping blocks with a shift length of 167ms.

4.1.2 Feature description

The feature representation of the final input signal is 160-dimensions which are composed by ratemaps and amplitude modulation spectrogram: a) ratemaps is an auditory spectrograms that similar to auditory nerve firing rate across auditory filters for individual time frames; b) amplitude modulation spectrogram is extracted based on a bank of logarithmically-scaled modulation filters. Each time frame was assigned 16 frequency channels * 9 modulation filters value. All the features has to be transformed to arithmetic mean of zero, and a variance of one. The main reason to standardize the input is that the nature of the easy-saturation of the nonlinear activation function in the LSTM block leads to difficult and slow learning.

There are two types of labels: instant interprete label is whether the sound event active at this exact moment; block interprete label is whether the sound event been active in the last 500ms. There are four different values of label:

1. **+1**: if the respective targeted sound event has been active only on the master source during the last 500ms to an extent of at least 75%

2. **0**: if it has been active only on the master source during the last 500ms to an extent below 75%

3. **-1**: if the respective targeted sound event has not been active on any source during last 500ms

4. **NaN**: the respective sound event has been active on at least one source different than the master source during the last 500ms.

4.1.3 Training batch generation

In order to simulate a real-world scenario, the audio stream is only considered as one-way with two types of label, one is the instant interprete label with a few milliseconds delay, and the other one is the block interprete label with smooth labelling strategy. To cope with the need of mini batch learning while the length of different scene instances is various, we propose a kind of random insert algorithm (1) to create the rectangle dataset, eliminating the waste of computational calculate padding batch and randomly preserving the primitive data during each iteration. The stateful LSTM is a mechanism after processing the training data of a batch, its internal state (memory) will be used as the initial state of the training data of the next batch. It is a proper way to implement the streaming processing for real-time system.

Algorithm 1 Construct rectangle

Require: $TotalEpochs$,$SceneInstance_1$, $SceneInstance_2$, ..., $SceneInstance_{N-1}$
Ensure: *Rectangele*

```
SceneInstance[k] ← training data scene instance K
Rectangle ← batchsize rows of variable length(initially empty)
for m ← 1 to TotalEpochs do
    Permutation ←getRandomOrder(0,1,..,N-1)
    for j ← 0, ..., N − 1 do
        shortestRow ← row of Rectangele with minimal length
        shortestRow ← shortestRow + SceneInstance[Permutation[j]]
for i ← 0, ..., TotalBatches − 1 : do
    batch[i] ← Rectangle[:;i*BatchTimeLength:(i+1)*BatchTimeLength]
```

In the above algorithm, the 'scene instance' represents a series of acoustic signal occur in specific scene. And the dimension of final rectangle is [Total epochs (batch size), Total length (the sum of all scene instances' length].

4.1.4 Evaluation strategy

Our dataset was divided into a training set for learning model and a test set for evaluating the generalization performance of the model. There are six separately training subsets to conduct cross validation while promise the training and validation sets independently. Each sets was randomly constructed from all scene instances. Only five training subsets were used for building the classification models and the rest set for evaluating the performance. For each combination of training subset, there is an individual weight of the targeted classes for calculating the loss function based on weighted cross entropy. Since the ground true label of 0 frame is unclear, here we assign zero cost in training and testing. We also treat NaN frame as +1 in training and zero cost in testing.

$$\begin{aligned} Sensitivity &= \frac{TP}{TP + FN} \\ Specificity &= \frac{TN}{TN + FP} \end{aligned} \tag{4.1}$$

$$BAC1 = \frac{1}{2} \cdot (Sensitivity + Specificity) \tag{4.2}$$

$$BAC2 = 1 - \sqrt{(1 - Sensitivity)^2 + (1 - Specificity)^2} \tag{4.3}$$

The evaluation of model predictive capacity is balanced accuracy (BAC1 and BAC2) in Eq. 4.2,4.3. BAC1 is the arithmetic mean of sensitivity (positive class accuracy) and specificity (negative class accuracy) which calculated by true positive (TP), true negative (TN), false positive (FP) and false negative (FN) in Eq.4.1.Instead of using common performance meauses like F1 socre and error rate, BAC is better to handle the real-world scenario where the propotions of positive and negatives classes are imbalanced.

When we evaluate the test set, samples of different lengths should have the same weight for the overall accuracy and each scene should also be weighted. Therefore, we use the following procedure (2) to calculate the final accuracy. Taking into account the various length of each scene instance, the evaluation of the test set first calculate TP, TN, FP, FN of each scene instance and decorrelate them by dividing the sum of the number of sound event of all classes N (13 * the length of each scene instance). Then averaging over each scene instance of their TP/N, TN/N, FP/N, FN/N. Finally, calculate the BAC of each class and calculate the mean of all classes for each scene.

Algorithm 2 Evaluation procedure

Require: TP,TN,FP,FN of each scene instance, N

1: **for** Scene instance$_i$ ← 1,2,... **do**
2: N$_i$ ← TP+TN+FP+FN
3: matrice$_i$ ← $[\frac{TP}{N}, \frac{TN}{N}, \frac{FP}{N}, \frac{FN}{N}]$ (decorrelate from length)
4: **for** Scene$_i$ ← 1,2,... **do**
5: Scene$_i$← mean(TPs,TNs,FPs,FNs),$\forall$ scene instance $\in Scene_i$
6: calculate sensitivity and specificity for each scene and each class
7: average sensitivities over scenes (nSrc-weighted); specificities similarly
8: for each class: calculate BAC1 and BAC2 from theirs per-class sensitivities and specificities
9: average BAC1 and BAC2 over classes

4.2 Hyperparameter optimization

Model configuration, generally referred as a combination of hyperparameters used to build the model. It is a parameter sets that define a value before starting the learning process, not a parameter that is obtained through training. Since the space of the hyperparameter is infinite, the combined configuration of the hyperparameters can only be a "better" solution, and there is no optimal solution. We can only approximate the global optimal solution by finding the local optimal solution while increasing the number of experiments. Tab 4.1 is the hyperparameter space in this work. All of these hyperparameters are randomly selected except for the batch length and size. Because the memory limitation of one GPU, we need to prioritize the structure of the neural network such as the number of layers and neurons. So the batch size will be decreased when the network is complex and require big memory.

Hyperparameters	Range
Number of LSTM layers	1, 2, 3, 4
Number of LSTM cells	50 - 1024
Number of MLP layers	0, 1, 2, 3
Number of MLP cells	50 - 1024
Weightdrop rate	0.25-0.9
Initial learning rate	0.0001-0.01
Batch length	1000, 2000, 3000,4000
Batch size	8,16,32,64

Table 4.1: An overview of the hyperparameter space we searched in this work.The number of LSTM and MLP cells are sampled in logarithmic scale.The dropout rate is not the usual one but the output keep probability. If it is equals to 1, no output dropout will be added. The batch length is truncated backprop length and batch size is minibatch size for gradient descent.

In the early stage of the hyperparameter optimization work, we found one hyperparameter combination takes per cross validation split approximately 50 hours training time. The reason behind is that the size of training data is larger than 100GB and the batch size is limited by the relatively longer batch length and GPU memory. What's more, it is already based on the condition of using five epochs early stopping. Therefore, the full cross validation (taking each of the 6 folds once as validation set and the respectively remaining 5 folds as training set) requires 6X the above training time, which is too expensive and undersampling of the hyperparameter space. In addition, after completing the first ten combination, the generalization capacity of each combination has almost the same rank under different cross validation setting.

Consequently, we design a strategy to speed up this optimization process.

1. The training data was divided into six folds. Firstly, we use no.3 data fold as validation set to train all generated combinations;

2. Then we select these combinations with the accuracy higher than 80% and train it on no.4 data fold;

3. Finally, we pick the top five combinations and train it on no.2 data fold.

4. At the end, we have finally used 3 out of 6 splits, representing a partial cross validation. And the averaged validation BAC1 is used as the final performance. And the best combination and its averaged bestepochs is used for further evaluation of testing data.

4.2.1 Optimization of instant interprete based model

Tab.4.2 is the result of first level optimization.There are eight row missing training and validation result in the bottom of table. Because when the training is doing a gradient descent on the objective loss function, sometimes the value of loss underflow and makes the logarithm of this value turn to NaN. Training can only be terminated because it is impossible to minimize NaN. In another case, as the training progresses, the neural network seems to fall into a dilemma: although it continues to learn, it always predicts that all validation samples are negative. These two cases is the reason to explain why these rows are blank.

Tab.4.3 includes whose BAC1 value higher than 80% in 4.2. And the BAC1 column here is the averaged result of first and second level optimization.

ID	LearningRate	λ_{norm}	Keep probability	LSTM Layer	LSTM Cells per layer	MLP Layer	MLP Cells per layer	BatchSize	BatchLength	Train BAC1	Validation BAC1
52	0.000193061	4.59E-05	0.868368616	4	582	1	210	16	3000	0.917	0.828
54	0.000102086	0.000359033	0.556215399	3	323	2	609	16	3000	0.899	0.822
8	0.000147535	0.000113739	0.842929106	4	280	2	335	16	2000	0.907	0.817
49	0.000406474	0.000310605	0.672600762	3	388	0	752	16	2500	0.89	0.815
55	0.000133801	0.000317152	0.368802962	2	251	1	457	16	2500	0.882	0.815
36	0.000180087	0.000051358	0.638239844	2	744	0	573	8	4000	0.903	0.813
47	0.000123646	5.01E-05	0.6570035	2	353	1	164	16	2500	0.908	0.813
42	0.000453885	6.11E-05	0.897078665	1	1014	1	90	16	2000	0.88	0.812
12	0.000456774	0.000269956	0.867914643	3	195	0	424	16	3000	0.889	0.811
2	0.000560727	0.000330904	0.592211535	2	418	1	470	16	2000	0.858	0.807
9	0.00015429	0.000698932	0.476082038	2	184	2	753	16	3000	0.876	0.807
45	0.001018068	0.000292025	0.295242227	3	197	0	57	16	2500	0.872	0.807
38	0.00015138	0.000509952	0.768722298	3	101	3	150	16	3000	0.833	0.8
1	0.000762759	0.00037326	0.583579225	3	546	2	400	8	4000	0.857	0.798
22	0.002540271	8.35E-05	0.645276036	2	77	1	143	16	2000	0.839	0.794
15	0.000115141	0.00019282	0.494174204	1	60	1	94	16	2000	0.846	0.788
26	0.00045379	0.000623145	0.330122412	1	158	1	785	8	4000	0.851	0.788
4	0.000101208	0.000790961	0.254627726	3	146	2	280	16	2500	0.86	0.787
11	0.000699443	0.000633778	0.643842796	2	350	2	382	16	2000	0.832	0.783
23	0.000487585	0.000812129	0.468174008	1	93	2	319	16	2000	0.826	0.781
6	0.000333547	0.000323973	0.258102147	1	558	1	126	16	3000	0.835	0.779
10	0.000412369	0.000954182	0.507340906	4	145	1	75	16	2500	0.824	0.772
30	0.00062414	0.000785571	0.788696581	1	1011	2	889	8	2000	0.819	0.772
25	0.002229056	0.000568781	0.78403681	3	143	1	76	16	2000	0.808	0.77
53	0.001321715	0.000992345	0.829313495	3	627	1	146	16	2500	0.806	0.768
32	0.003192106	0.000208522	0.495317245	3	90	1	64	16	2000	0.8	0.765
19	0.004640242	0.000291662	0.633866993	2	69	1	179	16	2500	0.797	0.764
44	0.006045392	8.75E-05	0.394771837	3	102	1	216	8	4000	0.799	0.758
5	0.000832375	0.000753247	0.469908901	3	589	3	150	16	2000	0.79	0.752
51	0.003989551	0.000207214	0.859240775	1	284	0	64	16	2500	0.788	0.752
27	0.003480072	0.000571822	0.840686188	2	617	2	182	16	3000	0.775	0.751
7	0.004379102	0.000976133	0.381640046	4	254	0	301	16	4000	0.761	0.747
28	0.001828038	0.000991292	0.785394247	3	280	2	697	8	4000	0.764	0.745
31	0.004193867	0.000717891	0.86876256	3	153	2	299	16	2500	0.765	0.744
34	0.004021766	0.000996427	0.847558153	2	337	1	333	16	2500	0.768	0.743
39	0.006247124	1.30E-05	0.30781122	1	149	2	392	16	2500	0.785	0.74
46	0.002900694	0.000699353	0.355517673	3	632	2	661	16	2500	0.75	0.739
18	0.009787386	1.38E-05	0.513549901	1	809	1	731	16	2000	0.774	0.729
29	0.006766428	0.000512191	0.764980571	3	150	1	644	16	2000	0.752	0.72
14	0.000175367	0.000684897	0.397842091	2	286	3	72	16	2500	0.741	0.719
56	0.000522418	0.000982491	0.319620778	1	307	2	109	16	2500	0.754	0.719
41	0.001677512	0.000867171	0.278615701	2	197	1	56	16	2500	0.738	0.716
16	0.005965893	0.000594471	0.639374343	3	599	2	400	16	2500	0.729	0.715
40	0.004882104	0.000958252	0.58505406	4	139	2	520	16	3000	0.712	0.709
13	0.005138169	0.000713306	0.515954491	1	413	2	598	16	2500	0.73	0.706
50	0.004733806	0.000586347	0.628986864	1	779	2	638	16	2000	0.729	0.706
43	0.010372016	0.000488253	0.521426135	1	98	1	420	16	2000	0.72	0.705
3	0.007406627	0.000702475	0.661246612	4	320	3	761	8	3000		
17	0.005663059	0.000517704	0.300829814	3	1089	1	191	16	2000		
21	0.009022413	0.000557344	0.647441971	3	302	1	105	16	3000		
24	0.002102772	0.000999083	0.504835878	3	710	1	93	16	2000		
33	0.001323074	0.000478849	0.652779747	4	1066	1	957	8	2000		
35	0.003253835	0.00095951	0.668697054	4	948	3	291	16	3000		
37	0.000683381	0.000992557	0.347995921	3	1038	3	86	16	3000		
48	0.00545837	0.000922601	0.701933365	3	783	3	123	8	4000		

Table 4.2: Instant interprete label, first level optimization with validation fold = 3.This result consists 60 combinations and sorted by the value of BAC1, which you can see in the rightest column. The top red part is selected for the second level optimization in Tab.4.3.

ID	LearningRate	λ_{norm}	Keep probability	LSTM Layer	LSTM Cells per layer	MLP Layer	MLP Cells per layer	BatchSize	BatchLength	Train BAC1	Validation BAC1
52	0.000193061	4.59E-05	0.868368616	4	582	1	210	16	3000	0.917	0.837
8	0.000147535	0.000113739	0.842929106	4	280	2	335	16	2000	0.907	0.833
36	0.000180087	0.000051358	0.638239844	2	744	0	573	8	4000	0.903	0.8315
55	0.000133801	0.000317152	0.368802962	2	251	1	457	16	2500	0.882	0.829
54	0.000102086	0.000359033	0.556215399	3	323	2	609	16	3000	0.899	0.8285
12	0.000456774	0.000269956	0.867914643	3	195	0	424	16	3000	0.889	0.828
42	0.000453885	6.11E-05	0.897078665	1	1014	1	90	16	2000	0.88	0.8275
47	0.000123646	5.01E-05	0.6570035	2	353	1	164	16	2500	0.908	0.8275
49	0.000406474	0.000310605	0.672600762	3	388	0	752	16	2500	0.89	0.825
2	0.000560727	0.000330904	0.592211535	2	418	1	470	16	2000	0.858	0.8225
9	0.00015429	0.000698932	0.476082038	2	184	2	753	16	3000	0.876	0.820
45	0.001018068	0.000292025	0.295242227	3	197	0	57	16	2500	0.872	0.819
38	0.00015138	0.000509952	0.768722298	3	101	3	150	16	3000	0.833	0.816

Table 4.3: Instant interprete label, second level optimization with validation fold = 4. The value of BAC1 is the averaged result of validation fold 3 and 4. The top five rows are used for the third level optimization in Tab.4.4.

Tab.4.4 selects top five combinations from 4.3. And the BAC1 column here is the averaged result of first and second and third level optimization.

ID	LearningRate	λ_{norm}	Keep probability	LSTM Layer	LSTM Cells per layer	MLP Layer	MLP Cells per layer	BatchSize	BatchLength	Train BAC1	Validation BAC1
52	0.000193061	4.59E-05	0.868368616	4	582	1	210	16	3000	9	0.84
8	0.000147535	0.000113739	0.842929106	4	280	2	335	16	2000	18	0.835
36	0.000180087	0.000051358	0.638239844	2	744	0	573	8	4000	7	0.833
55	0.000133801	0.000317152	0.368802962	2	251	1	457	16	2500	13	0.829
54	0.000102086	0.000359033	0.556215399	3	323	2	609	16	3000	16	0.829

Table 4.4: Instant interprete label, third level optimization with validation fold = 2. The value of BAC1 is the averaged result of validation fold 2, 3 and 4

From the above tables, We make some explanation for the influence of each parameter:

Learning rate and λ_{norm}

1. It is hard to explain that what is the positive influence of these two hyperparameters. But appropriate values of them is a prerequisite to ensure that the model can be effectively trained. According to the comparison of the upper and lower parts of the table, if either of these two hyperparameters is too large, regardless of whether the network structure is good or bad, the final performance is poor or blank (untrainable). The top red part of Tab. 4.3 contains all combinations with BAC1 higher than 80%. Their learning rate and λ_2 both are smaller an order of magnitude than those combinations in the bottom of table. It is also supports above argument.

2. From the observation of Tab.4.4, the range of learning rate is [0.0001, 0.001]. It is the "balanced interval" of learning rate in this work that neither overshoot nor converge too slowly. Compare the averaged epochs of combination 52, 8 and 36, too small learning rate even takes more than double epochs to converge. Consider the required epoch of training and the model learnability, the optimal learning rate probably lies in [0.00018, 0.00019].

3. Gradient descent of loss is easily fall into overshoot especially when the large λ_2 is added. It is a much stronger regularization term than dropout. Generally, the performance drop as the increasing of its value.

Dropout:

1. Tab.4.4 tells large keep probability (small dropout rate) is better when λ_{norm} is close. The main difference between combination 52 and 36 is dropout rate, high dropout rate leads to lower training and validation accuracy. This relation reflects the overfiting of training is effectively controlled.

Network:

1. **LSTM:**Looking down from the table, the number of layers and neurons are decreasing. We prefer to take the multiplication of the number of layers and neurons as the indicator for the learnability of model. The higher value of this indicator tend to show better learnability.

2. **MLP:**It's unlike the analysis in LSTM, one simple mlp layer with a followed output layer also can achieve good performance.

Batch size and length:

1. We didn't find the influence of different batch length. From the Keras documentation, cudnnLSTM is a fast LSTM implementation backed by CuDNN. And stateful LSTM can help resume last cell state for next batch training, so we prefer to use longer batch length in training.

Since the limitation of computation budget, all the above suggestions were drawn from a undersampled hyperparameter space. And the most straight way to obtain the effect of specific parameter is to control variable by comparing similar combinations. When the network is not very complex, we found the learnability of LSTM improved by the increasing number of layer and neurons. And a simple MLP layer is enough to fit with LSTM. And the larger complexity of networks would overfit such as combination 30 in Tab.4.2.

4.2.2 Optimization of block interprete based model

In order to avoid repeatedly training low rank combinations, here we take the rank in Tab. 4.3 as our training order. For each combination, we did full three level optimization. After completing 11 combinations and sort the averaged BAC1 in Tab.4.5, we can conclude that block interprete label has almost the same rank as instant interprete label.

ID	LearningRate	λ_{norm}	Keep probability	LSTM Layer	LSTM Cells per layer	MLP Layer	MLP Cells per layer	BatchSize	BatchLength	Train BAC1	Validation BAC1
36	0.000180087	0.000051358	0.638239844	2	744	0	573	8	4000	12	0.851
55	0.000133801	0.000317152	0.368802962	2	251	1	457	16	2500	20	0.840
52	0.000193061	4.59E-05	0.868368616	4	582	1	210	16	3000	10	0.844
8	0.000147535	0.000113739	0.842929106	4	280	2	335	16	2000	14	0.844
54	0.000102086	0.000359033	0.556215399	3	323	2	609	16	3000	14	0.839
42	0.000453885	6.11E-05	0.897078665	1	1014	1	90	16	2000	6	0.844
2	0.000560727	0.000330904	0.592211535	2	418	1	470	16	2000	7	0.840
47	0.000123646	5.01E-05	0.6570035	2	353	1	164	16	2500	16	0.838
49	0.000406474	0.000310605	0.672600762	3	388	0	752	16	2500	9	0.844
12	0.000456774	0.000269956	0.867914643	3	195	0	424	16	3000	10	0.839
9	0.00015429	0.000698932	0.476082038	2	184	2	753	16	3000	22	0.836

Table 4.5: Block interprete label, averaged result of three level optimization with validation fold =2,3,4

4.2.3 Optimization comparison

Compare the Tab. 4.4,4.5, almost each combination has around 0.5-2% improvement. We are also interested in how many epochs it should take for different labels. So we combine these two table to Tab. 4.6. It is obvious that each combination achieved higher BAC1 on block interprete label with similar epochs.

ID	Instant: Averaged Epochs	Instant: BAC1	Rank		Block: Averaged Epochs	Block: BAC1	Rank
52	9	0.84	1		10	0.844	2
8	18	0.835	2		14	0.844	2
36	7	0.833	3		12	0.851	1
55	13	0.829	4		20	0.840	3
54	16	0.829	4		14	0.839	4

Table 4.6: Instant interprete vs Block interprete label, averaged result of three level optimization with validation fold =2,3,4

There is one change of rank in different label, combination 36 and 52. We can found combination 52 has relatively small improvement while 36 has huge improvement when the model migrates from instant interprete to block interprete label. By investigating the hyperparameters of each combinations, combination 36 has relatively simple neural networks with stronger regularization.It suggests the block interprete label is easier to overfitting especially when the neural networks is complex enough. And a proper regularization is necessary for detect the learnability of model.

4.3 Results and discussion

In this analysis, it was investigated how well the multi-conditional model generalize to simulated practical acoustic environment where distractor source was superimposed on the target source at signal-tonoise ratios (SNR) of 10 dB, 0 dB, -10 dB and -20 dB. Target and distractor sound sources were placed at different azimuth configurations as Section 4.1.1) states.

[1] states that training enough specialized models to cover all the combination of conditions that may exist in the dynamic environment is extremely demanding. And specialized model is particularly sensitive to the deviation of the SNR from the training situation. It is interesting to see how the performance of neural network in multi-conditional problem. In order to ensure the dissimilarity between training and testing set, they never contained parts of the same sound file. So the training set we used to build model does not intersect with testing set. The following test set model performance values was pooled over thirteen classes, all angular configurations.

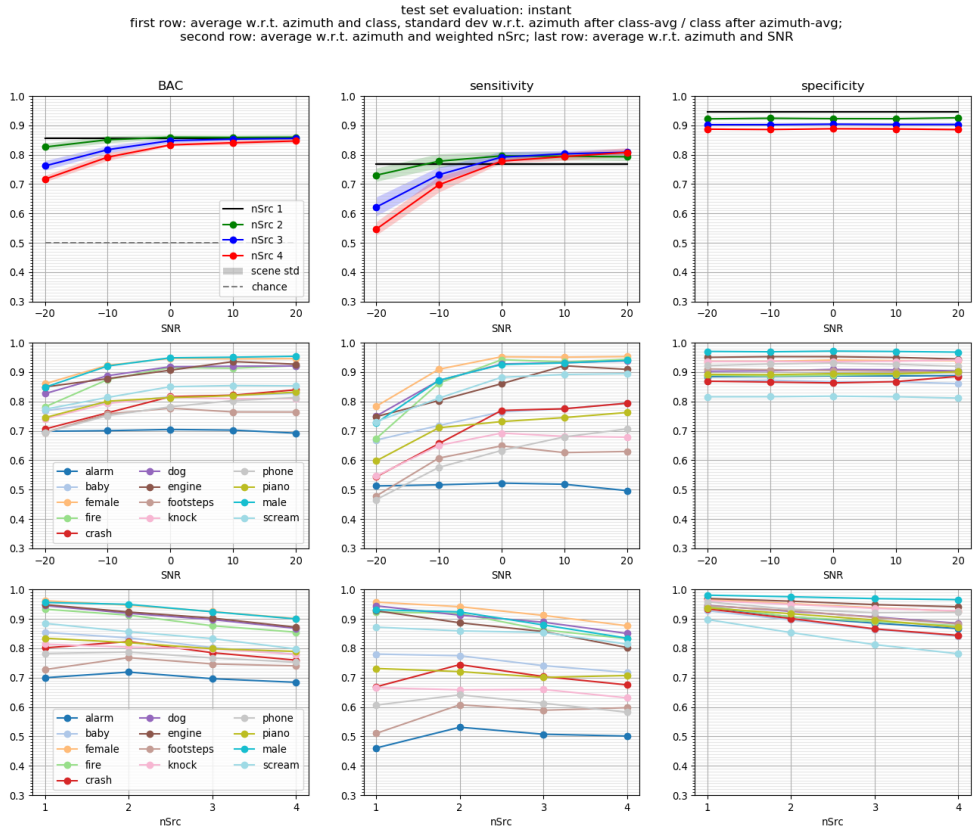

Figure 4.2: Test set evaluation of instant-interpret based model: it presents different levels of performance from three perspectives: balanced accuracy, sensitivity and specificity

Fig.4.2 is the evaluation of instant interprete based model with optimal combination 36 from Tab.4.5. The first row shows the neural network has quite good generalization ability to any numbers of sources when the SNR larger than 0. No matter in left(balanced accuracy), middle(sensitivity) or right(specificity), there is a obvious rank of performance for different number of ambient sources (nSrc2 > nSrc3 > nSrc4). This is to be expected that the scene with one source is easier to distinguish. It was found that nSrc1 is not associated with SNR and super stable for any SNR condtion. The middle plot in first row shows nSrc1 get a lower performance of sensitivity when model trained at SNR larger than -10dB. In Fig.4.3, block interprete based model has the similar situation.

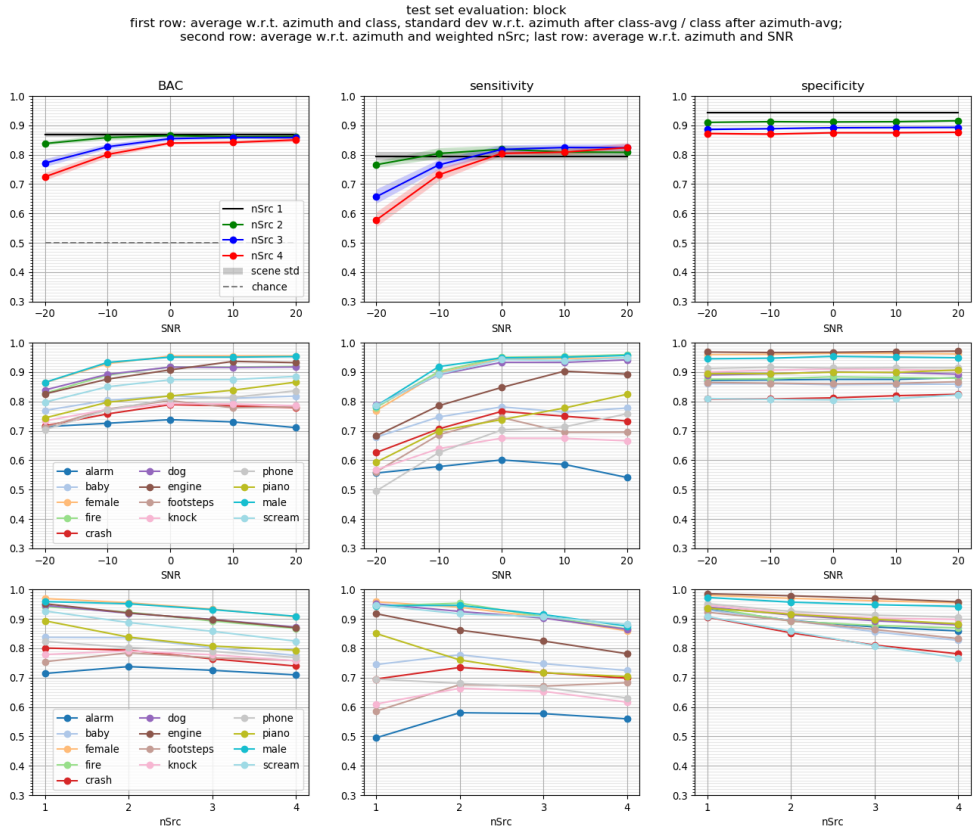

Figure 4.3: Test set evaluation of block interprete based model: it presents different levels of performance from three perspectives: balanced accuracy, sensitivity and specificity

Although there is no significant difference between Fig.4.2 and Fig.4.3, Fig.4.4 presents the comparison of these two results which obviously indicates the block interprete label exceeds instant one in sensitivity but fails in specificity. It is consistent with the idea of smoothing label over time. Block interprete label contains more active frame than instant one so that the model has higher possibility to captures positive frames. There is a roughly 4% improvements of sensitivity in any conditions of SNR and a 1.5% drop of specificity.

From the above graph, the comparison showed the sensitivity of model is very sensitive to the deviations of the environments from training conditions. The best performance is obtained with the model trained at 20dB. This condition mainly affects the sensitivity and almost not associated with specificity. It is also applys to the performance of each sound class. The majority contribution of difference is comes from sensitivity. The sensi-

tivity of male sound outperform other sounds, then the balanced accuracy is dominated by this effects.

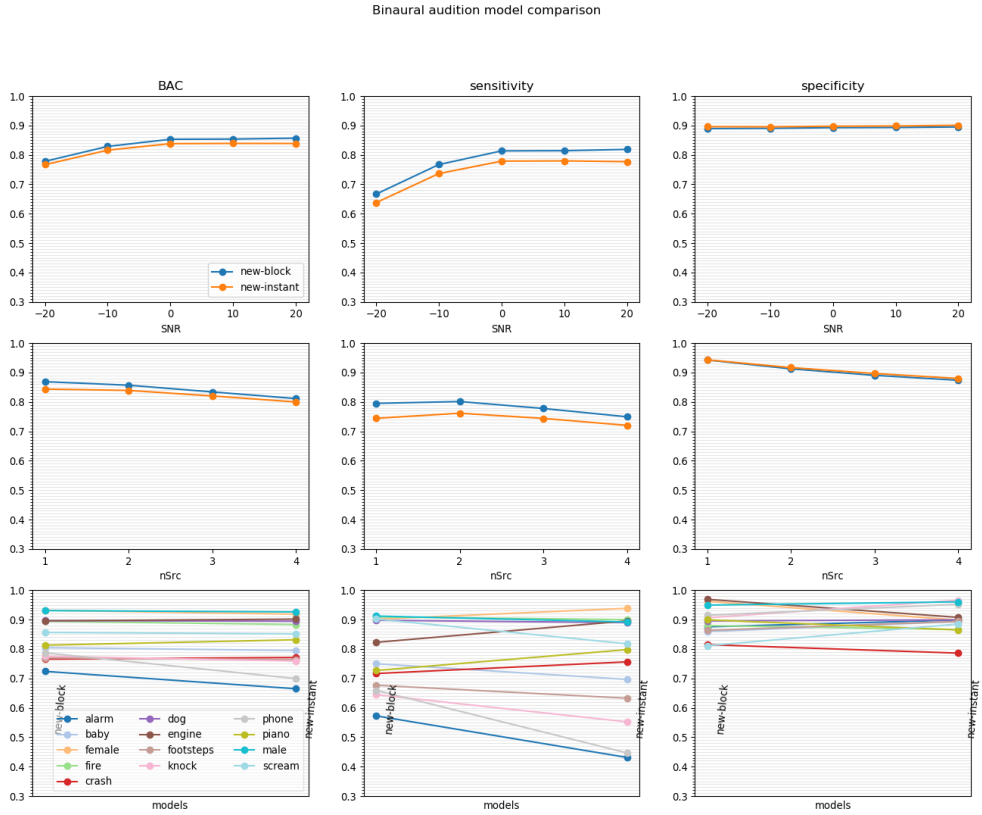

Figure 4.4: The comparison between block and instant label. Performance groups at different SNRs and nSrc, averaged over all azimuth configurations, sound classes

Tab. 4.5 contains the class-wise performance of the evaluation for our optimal block interprete based model and the baseline result presented in [1]. The general class-averaged performance of neural network has a 2.9% improvements. It shows that neural network overwhelmingly beat common classification model in each targeted class except alarm, footsteps, knock, piano. It is likely to explain that the common classification model is enough to distinguish these four types sounds.

model	[1] baseline	block-interpret LDNN
alarm	0.67	0.67
baby	0.77	0.80
femaleSpeech	0.85	0.94
fire	0.82	0.91
crash	0.74	0.77
dog	0.82	0.91
engine	0.79	0.89
footsteps	0.79	0.74
knock	0.84	0.74
phone	0.74	0.76
piano	0.83	0.82
maleSpeech	0.87	0.94
femaleScreammaleScream	0.81	0.87
class-averaged	0.795	0.826

Figure 4.5: Compare the evaluation of block interprete model to [1]'s baseline

5 Conclusions

In this work, inspired by the great advanced of neural network models that do multi-channel audio recognition, the optimal performance of LDNN model on block interprete and instant interprete labels are investigated. It showed that the block-interpret label got higher performance as expected, just as it is essentially smoother and easier to distinguish than instant interprete label.

Although the general improvement brought by the neural network is not as significant as imagined when we compare the performance to the baseline in [1].In the Fig.4.5, the separately performance is worth to have a look. It was found out the neural network can improve 10 out of 13 sound classes while only lose a small part of performance for the other 3 sound classes. The evaluation presents testing performances is ordered by the number of added distractor sources . Despite the final improvement did not meet our expectations, the neural network shows the increased robustness of universal generalization which is the most valuable property in real-world applications to handle dynamically changed environment. We believe there still a lot benefit of neural network remain to be explored, not only limited in the pattern described in this work, but also for other similar problem.

In a conclusion, using neural network to build multi-conditional model is a better solution for practical binaural application than classical linear model: only a single model needs to be trained on full training set with the optimal hyperparameter combination, the completely tuned model is robust with respect to multiple distractor sources superimposition, arbitrary SNRs and various angular configurations.

There are several state-of-the-art architecture such as GLDNN which could be taken to further investigation of model generalization ability in this binaural sound recognition problem. As we explained in section3.4 it has extra information gains from feature dimension over time. Although it takes more computational resources and time, it would be worth to explore its hyperparameter space and test its generalization capacity.

Bibliography

[1] Trowitzsch, I., Mohr, J, Kashef, Y. and Obermayer, K. 2017. *Robust Detection of Environmental Sounds in Binaural Auditory Scenes.*

[2] Ian Goodfellow, Yoshua Bengio, Aaron Courville. *Deep Learning*

[3] Bo Li, Tara N. Sainath. *Acoustic modeling for Google home*

[4] Yarin Gal,Zoubin Ghahramani. *A Theoretically Grounded Application of Dropout in Recurrent Neural Networks*

[5] Hasim Sak, Andrew Senior, Francoise Beaufays. *Long Short-Term Memory Recurrent Neural Network Architectures for Large Scale Acoustic Modeling*

[6] James Bergstra, Yoshua Bengio. *Random Search for Hyper-Parameter Optimization*

[7] Lisha Li,Kevin Jamieson,etc. *Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization*

[8] Elad Hazan, Adam Klivans,etc. *Hyperparameter Optimization: A Spectral Approach*

[9] Jimmy Lei Ba, Jamie Ryan Kiros,Geoffrey E. Hinton. *Layer Normalization*

[10] Nal Kalchbrenner, Ivo Danihelka, Alex Graves. *Grid Long Short-Term Memory*

[11] Tara N. Sainath, Bo Li. *Modeling Time-Frequency Patterns with LSTM vs. Convolutional Architectures for LVCSR Tasks*

[12] Bo Li, Tara N. Sainath. *Acoustic modeling for Google home*

[13] Gal and Ghahramani. *A theoretically grounded application of dropout in recurrent neural network*

[14] Zaremba et al. *Recurrent Neural Network Regularization*

[15] Davis, S and Mermelstein, P. *Comparison of Parametric representations for monosyllabic word recognition in continuously spoken sentences.*

[16] Sawhney N, Maes P *Situational Awareness from Environmental Sounds[J].1997*

[17] Clarkson B, Sawhney N, Pentl A. *Auditory Context Awareness via Wearable Computing[J]. In proceedings of the 1998 workshop on preceptual user interfaces,1998*

[18] Nishiura T, Nakamura S. *Study of environmental sound source identification based on hidden Markov model for robust speech recognition[J]. Journal of the Acoustical Society of America, 2003*

[19] Couvreur L, Laniray M. *Automatic Noise Recognition in Urban Environments Based on Artificial Neural Networks and Hidden Markov Models[J]. Internoise, 2004*

[20] Rosenblatt F. *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms[J]. Archives of General Psychiatry, 1962(3):218-219.*

[21] Hornik K, Stinchcombe M, White H. *Multilayer feedforward networks are universal approximators[J]. Neural Networks, 1989, 2(5):359-366*

[22] Rumelhart D, Mcclelland J. *Learning Internal Representations by Error Propagation[C]. Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. MIT Press, 1986:318-362.*

[23] Mohamed A, Sainath T N, Dahl G, et al. *Deep Belief Networks using discriminative features for phone recognition[C]. IEEE International Conference on Acoustics, Speech, Signal Processing. 2011:5060-5063.*

[24] Dahl G E, Yu D, Deng L, et al. *Context-Dependent Pre-trained Deep Neural Networks for Large Vocabulary Speech Recognition[J]. IEEE Transactions on Audio Speech Language Processing, 2012, 20(1):30-42.*

[25] Yoshua Bengio, Patrice Simard and Paolo Frasconi. *Learning Long-Term Dependencies with Gradient Descent is Difficult.*

[26] Sepp Hochreiter. *The vanishing gradient problem during learning recurrent neural nets and problem solutions.*

[27] S. Hochreiter and J. Schmidhuber. *Long short-term memory. Neural Computation, (8):1735–1780, 1997*

[28] A. Graves and J.Schmidhuber. *Framewise phoneme classification with bidirectional LSTM and other neural network architectures. Neural Networks, 18:602–610, 2005*

[29] A. Graves. *Generating sequences with recurrent neural networks.*

[30] A. Graves, S. Fernandez, and J. Schmidhuber *Multi-Dimensional Recurrent Neural Networks," in Proc. ICANN, 2007*

[31] Diederik P. Kingma, Jimmy Ba.Adam *A Method for Stochastic Optimization. 2015 ICLR*

[32] Rafal Jozefowicz, et al. *An empirical exploration of recurrent network architectures*

[33] Graves, et al. *Speech Recognition With Deep Recurrent Neural Networks, 2013*

[34] K. Greff, R. K. Srivastava, J. Koutnık, B. R. Steunebrink, and J. Schmidhuber, *Lstm: A search space odyssey," IEEE transactions on neural networks and learning systems, 2017.*

[35] J. Li, A. Mohamed, G. Zweig, and Y. Gong *LSTM Time and Frequency Recurrence for Automatic Speech Recognition," in Proc. ASRU, 2015*

[36] I. Trowitzsch, J. Taghia, Y. Kashef, and K. Obermayer. *NIGENS anechoic earsignals," Dec. 2016. [Online]*

[37] N. Yamakawa, T. Kitahara, T. Takahashi, K. Komatani, T. Ogata, and H. G. Okuno. *"Effects of modelling within- and between-frame temporal variations in power spectra on non-verbal sound recognition," in INTERSPEECH 2010, 11th Annual Conference of the International Speech*

[38] M. Kleinschmidt and D. and Gelbart. *Improving word accuracy with Gabor feature extraction,," in Proc. Interspeech, 2002*

[39] N. Mesgarani, D. Stephen and S. Shamma. *Representation of phonemes in primary auditory cortex: how the brain analyses speech," in Proc. ICASSP, 2007*

[40] R. Pascanu, C. Gulcehre, K. Cho, and Y. Bengio. *How to Construct Deep Recurrent Neural Networks," in Proc. ICLR, 2014*

[41] Stephen Merity , Nitish Shirish Keskar , Richard Socher. *Regularizing and Optimizing LSTM Language Models*

[42] Williams, R.J. and D. Zipser (1995). *"Gradient-based learning al- gorithms for recurrent networks and their computational complexity". In: Backpropagation: theory, architectures, and applications. Ed. by D.E. Rumelhart and Y. Chauvin. L. Erlbaum Associates Inc., pp. 433–486. isbn: 0805812598*

[43] LSTM introducton: *https://colah.github.io/posts/2015-08-Understanding-LSTMs/*