Stony Brook University

Stony Brook University

Classification of Fetal Heart Rate Signals by Deep Learning

A Thesis presented

by

Xuan Chen

to

The Graduate School

in Partial Fulfillment of the

Requirements

for the Degree of

Master of Science

in

Electrical Engineering

Stony Brook University

December 2016

Stony Brook University

The Graduate School

Xuan Chen

We, the thesis committe for the above candidate for the

Master of Science degree, hereby recommend

acceptance of this thesis

Petar M. Djurić - Thesis Advisor
Professor, Department of Electrical and Computer Engineering

Mónica F. Bugallo - Second Reader
Associate Professor, Department of Electrical and Computer Engineering

This thesis is accepted by the Graduate School

Charles Taber
Dean of the Graduate School

Abstract of the Thesis

Classification of Fetal Heart Rate Signals by Deep Learning

by

Xuan Chen

Master of Science

in

Electrical Engineering

Stony Brook University

2016

In the course of delivery, a fetus may suffer from oxygen deficiency due to the intensive pressure changes. Electronic fetal monitoring (EFM) system has been widely used in obstetrics, to provide continuous information to clinicians for making decisions and in preparing for delivery. There has been many efforts to build automated systems to analyze fetal heart rates (FHRs) and offer clinical supports.

In this thesis, our goal is to introduce the most recent and popular machine learning method, deep learning, for FHR classification. We first introduce the preliminaries of FHR classification methods and the database used in our experiments. Then, the basics and unique characteristics of deep learning are discussed, in order to create foundation to understand our method. After that, we introduce 1-D convolutional layer to the models and select their parameters. Finally, we test the performance and generalization under three conditions.

We build two models, which take the raw FHR and features extracted from FHR, respectively. The comparison between two models confirms the capability of neural network to exploit nonlinear features. We also apply data augmentation to the FHR database, which eliminates the unbalance of data set and the lack of sample size. It shows good performance of cross validation on augmented data set. The generalization of the models is tested on the original data set used to generate augmented data. Finally, we propose conjectures on the low true positive rate happened in the validation on original data set that is not used in generation.

Acknowledgments

First of all, I would like to thank my advisor Prof. Petar M. Djurić for his help and advice throughout my Master's studies. His values, his extensive knowledge and enthusiasm for research, his enjoyment of teaching and incredible curiosity for and knowledge of the world around him, have been an inspiration to me. I am especially appreciative of his belief in me whenever I felt discouraged by my lack of progress in work.

I'm grateful to Prof. Mónica F. Bugallo as a of reader of my Master's thesis, taking her time to read, understand, and offer invaluable advice about my thesis. Her intelligence, enthusiasm for research and incredible ethic of teaching are worth for me to acquire. For her support and advice, I will always be grateful.

I offer sincere thanks to my friends and colleagues in the lab - Çagla Tasdemir, Iñigo Urteaga, Kezi Yu and Guanchao Feng. They have been working on topics related to my Master's thesis for a long time, and provided plenty of valuable suggestions and help during my research. Without their generosity, I would have spent much more time to find out correct methods. I have special acknowledgments for Kezi and Guanchao. This thesis is partially based on their prior works. Without their help, I would have had much more difficult time to complete my Master's thesis.

I offer special thanks to the other colleagues and friends in the lab - Hechuan Wang and Lingqing Gan. It has been nice to discuss with them research, food, travel and other wide ranging issues. They made my time in the COSINE lab very memorable and rewarding. I'd like to also mention my sincere appreciation to friends outside of the lab (there are too many of them to list individually) for making life interesting and for always being supportive and honest.

Finally, I would like to thank my parents and my family members for always being my strongest pillars of support. Their love and encouragement have been the most important things in my life.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

It has always been the most important and precious event for a mother to get pregnant, nourish and deliver a baby. During the process of pregnancy, the physiological status of mother is significantly different from its normal status. Maternal body provides extra nutritions in order to maintain the fetus in the uterine and drive normal metabolism of the fetus through the umbilical cord. The life of a fetus relies on exterior resources but as soon as the baby is delivered, the newborn individual has to support itself by its own physiological system. This crucial transition between the two statuses happens during the mother is laboring. Lots of changes of body structures occur to the fetal body within the fleeting hour and all procedures need to be ready for the first breath after labor. This process is so crucial that any unwanted outcome brings threat to the life of fetus.

A woman's uterine contracts when the fetus is about to be delivered, which is helpful during natural birth because of the pressure it creates for pushing the baby out of the vagina. The uterine contraction makes impact on the fetal body as well as the umbilical cord. The constriction gives pressure on the surface of the fetal body, squeezing the blood vessels under the skin. In addition, the constriction presses the umbilical cord, and reduces the blood flow into the fetus. The fetal cardiac activity decreases following the uterine activity, reflecting their intimate connections. When the uterine relaxes, the pressure and activities of the fetal body gradually return to normal. Since the fetus stays in a highly intensive environment during delivery, the

oxygen insufficiency caused by the blood flow reduction, which might be caused by excessive uterine activity or other reasons, is necessary to be taken care of. There are different stages for oxygen insufficiency. Hypoxemia is the initial stage of oxygen insufficiency. In this stage, the oxygen in the fetal arteries reduces. The reflex system increases the cardiac activity in order to raise more blood flow bringing enough oxygen and on the other hand, restrains fetal activity to reduce oxygen consumption.

If the fetus suffers from more serious oxygen insufficiency or stays in hypoxemia for a longer period without recovering to normal states, it enters the stage named hypoxia. In this stage, oxygen is insufficient to supply the whole body. Then, oxygen is provided to the central organs in order to sustain the baseline of staying alive. Consequently, the peripheral tissues are not able to obtain enough oxygen and they turn to anaerobic metabolism. Fetuses can last a few hours in this stage without irreversible damage. Then fetus may enter a third stage of oxygen insufficiency, asphyxia, where central organs lack oxygen and turn to anaerobic metabolism. The anaerobic metabolism consumes the glucose stored in the fetal body. Along with the glucose being consumed, the fetus will undergo unrecoverable damage of the brain and the heart, which is one of the sources of cerebral palsy, or die.

As for the intimate relation between fetal heart rate and oxygen insufficiency, electronic fetal monitoring system was introduced in order to continuously place fetus under supervision. The methods for monitoring can be divided into two categories: invasive and non-invasive. Invasive methods usually refer to the insertion of an electronic device with piezoelectric sensors. The piezoelectric sensors generate electric signals while being deformed by the fetal activities. The signal is transmitted through cables into the outside part and is displayed on a screen or analyzed by a computer and then displayed. There are similar devices for non-invasive measurements, which place piezoelectric sensors on the maternal body. Non-invasive methods are preferred nowadays due to the potential influence of invasive devices to neonates. However, non-invasive methods are less accurate than invasive ones due to the interference from maternal activities such as heartbeat, breath and peristalsis. Another non-invasive method is based on a device that uses Doppler. The device emits ul-

trasonic waves and receives the reflection and measures the difference between their frequencies to calculate the fetal activity.

The development of automatic monitoring system has been proposed for two major reasons. At first, clinical doctors who are able to diagnose fetal status are unable to supervise continuously because of either limited stamina or lack of staff. Therefore, it is extremely dangerous for a baby to enter the asphyxia stage when doctors are absent since there are few minutes before an unrecoverable damage occurs. Secondly, research reveals high intra- and inter observer variability in daily clinical practices. The diagnostic result sometimes differs between different doctors and between different states for a certain doctor regardless of their work place and experience. Thus, a reliable, stable and automatic monitoring system is expected, where the classification model is the most critical part.

1.1 Motivation

The aim of the this thesis is to contribute to the classification model of fetal heart rate by introducing state of the art technology, deep learning. Deep learning has thrived since 2006 when a theoretical breakthrough came into being [1]. It shows great advantage for some problems that were considered difficult. These problems involved images, audio signals and natural languages, which have common similarities in their features in that they are hard to describe and they have highly abstract information built hierarchically from bottom up. The information of fetal heart rate lies in time series, which can be described by a point existing in a high dimensional space whose dimension is the length of the time series. As long as the dimension increases, classification models suffer from the "curse of dimensionality", in which the number of samples exponentially increases with respect to the dimensions to keep sufficient sample density. Without sufficient sample density, models tend to overfit in the training stage and result in bad performance in generalization. Deep learning has shown the ability that overcomes overfitting to some extent when dealing with high dimensional problems. Intuitively, this property can be used for fetal heart rate

signals. Traditionally, the features of FHR signals were hand coded. Before the test result of a certain feature comes out, we do not know whether the feature provides additional information with respect to the information available in existent features. This leads to a trial and error process, which is inefficient and may fail if useful information hides in unknown features. The proposal of Multi layered perception (MLP) [2, 3] wanted to solve the problem. The predecessor of MLP is perceptron, which is a classification model for binary input and output with linear relations [4]. MLP extends the notion by stacking multiple perceptions and regards every layer except the last one as a nonlinear feature extractor. Since the stacking of linear functions is still linear, the output of each layer of MLP is followed by a nonlinear activation function. MLP is a type of artificial neural network. The work in this thesis will explore the available means and appropriate type of neural network for classification of FHR signals, test the performance, and provide directions further research.

1.2 Fetal Heart Rate Time series

The FHR signals are usually recorded by electrocardiograph (ECG), which takes consecutive time samples from sensors and outputs R-R intervals. FHR signals can be directly computed from the R-R intervals. Each sample of the signal is measured in beats per second. The data used in training and validation is from Physionet [5], an online resource of physiological data for biomedical researches. The database we use in this thesis is from the Czech Technical University in Prague and the University Hospital in Brno. The database contains 552 FHR recordings selected from 9164 recordings [6]. Each recording starts no more than 90 minutes before delivery and is at most 90 minutes long. The signal is sampled at 4Hz and the recording is stored as fetal heart rate in beats per minute. There are additional clinical variables included in each sample such as age, parity, sex, indices of umbilical artery blood sample and APGAR score. The blood samples are measured when the baby is delivered. In this work, we take the pH value of the umbilical blood as the golden standard to indicate

whether a fetus is undergoing oxygen insufficiency.

There are missing data during sampling due to imperfections of the recording system problem. This is resolved by preprocessing the acquired signals. The preprocessing strategy is described as follows:

1. Delete missing data (0s and NaNs) at the beginning and at the end of each recording.

2. For missing data in the middle, if the duration is less than 15 seconds, interpolate data by a piecewise cubic Hermite polynomial. Otherwise, delete the segment directly and connect the two adjacent segments.

1.3 Previous Studies

Monitoring FHR has been used in practice for a long time. At first stethoscope was used to inspect FHR intermittently. Then electronic fetal monitoring (EFM) system provided continuous monitoring, and overcame the potential danger when doctors were not in attendance. EFM usually includes cardiotocography (CTG) and uterine contractions (UC). Nowadays EFM has already been a common equipment and is deployed almost completely in obstetrical activities.

Though continuous monitoring sounds practical, the reading and interpretation of FHR signals have been a huge burden for clinical doctors. Furthermore, the interpretation suffers from significant inter- and intra-observer variability, which lead people to build fully automated FHR signal interpretation and decision system. However, there has been some progress. For example, the STAN system developed by Neoventa, Sweden, has full clinical use [7–9]. Also, there has been progress with new methods. These methods are based on different approaches, including time-frequency analysis [10], nonlinear feature classification [11], hierarchical Bayesian modeling [12] and use of neural networks [13, 14]. However, due to the complex biological environment during delivery, a lot of noise, artifacts and lack of continuity are introduced to the measured signal, bringing difficulty and challenges [15]. As a result, the search for an

accurate and stable method for supporting clinical decisions is still underway.

1.4 Structure of Thesis

This chapter discussed the motivation of this work, the previous studies of classification of FHR signals and preliminaries of fetal heart rate and deep learning, which are the basis of and closely related to the other chapters of this thesis.

In chapter 2, a detailed overview of deep learning is presented. We discuss the common properties of neural networks. We also discuss the characteristics of each type of neural network and determine the most suitable one to be used in our work.

In chapter 3, the model is built from a type of neural network to a well-defined model that could be about to work. We build different models for different inputs and later compare their performances.

Chapter 4 presents the training and validation results. The comparison between different models on augmented and real data sets is presented and analyzed.

Chapter 5 provides conclusions from the performance results and discusses the problems we encountered during our work. The discussion also involves the possible solutions of these problems. We also suggest directions for future research.

Chapter 2

Introduction Deep Learning

Deep learning has been extensively used in industry [16–24], due to its handiness and effectiveness without knowing domain specific knowledge. However, it is not so popular in academia. Researchers usually prefer models with concrete mathematical basis and rigorous logic, whereas deep learning still remains unexplainable to some extent. However, it is important to study its theory and properties, because we can improve it only if we know how it works. In recent years, attention of some researchers has been drawn to this field. Thus, a lot of variations of neural network with better performance such as Long Short Term Memory (LSTM) [25] and Neural Turing Machine (NTM) [26] have been published. On the other hand, we are still far away from the underlying theory that could unify all the frameworks and explain their properties.

This chapter will introduce some basics of deep learning and build the foundation for using neural networks in subsequent chapters.

2.1 Background of Deep Learning

Neural networks dominate the research on deep learning. However the predecessor of neural networks is the perceptron. The perceptron algorithm was invented in 1957 at the Cornell Aeronautical Laboratory by Frank Rosenblatt [4], and it is equivalent to a linear support vector machine (SVM). The perceptron is able to classify

input vectors into binary classes. However, the perceptron is only capable to solve linear separable problems. It cannot solve even a simple linear inseparable case, i.e. exclusive-or classification problem [27]. The successor of perceptron, the multilayer perceptron (MLP), introduces an activation function so that it allows perceptrons to stack together without reduction. The MLP can be learned by backpropagation and is also known as the basic type of feedforward artificial neural network (ANN) [2,3]. The neural network also known as recurrent neural network (RNN) was invented in 1993. The RNN contains feedbacks, which provide memory ability, enabling the neural network to process time series. The Long short-term memory (LSTM) is a type of RNN, which was proposed in 1997 [25]. The LSTM explicitly includes memory cells, and it resolved the gradient vanishing problem during the unfold backpropagation in RNN. In 2014, the Neural Turing Machines (NTMs) as an extension of RNN was proposed [26]. It couples the external memory resources and lets it be differentiable. In 2016, the differentiable neural computer (DNC) [28] was introduced. It combines NTMs with an attention mechanism.

2.2 Semantics of Activation Function

The activation function plays an important role in the neural network structure. It is obvious that a multi layered neural network with linear activation functions can be reduced to a single layer model as follows (2.1):

$$\begin{aligned} h^1 &= W^1 h^0 \\ h^2 &= W^2 h^1 \\ \Rightarrow h^2 &= W^2 W^1 h^0, \end{aligned} \tag{2.1}$$

where h^is are layer vectors and W^js are weights. Thus, adding more layers with linear activation functions doesn't help to improve performance because they are all equivalent to a single linear model.

Stacking more nonlinear layers is helpful because such model cannot degenerate.

It endows the neural network with the capability to become deep. There are a number of types of activation functions but most of them can be categorized into three classes: sigmoid, Rectifier Linear Unit (ReLU) and others. The sigmoid functions usually refer to a sigmoid or a hyperbolic tangent function (tanh) [29]. They are given by (2.2).

$$\begin{aligned} sigmoid(x) &= \frac{1}{1+e^{-x}} \\ tanh(x) &= \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} \end{aligned} \tag{2.2}$$

One of the unique properties of the sigmoid class is the S-shaped curve. The limits of the function at plus and minus infinity are constant values, and the only significant change happens near the origin. The output of the sigmoid function ranges from 0 to 1, which is often used to describe probability. Tanh ranges from -1 to 1, which is unable to have probabilistic semantics but is symmetric along the origin. The sigmoid class was preferred in the first few years after 2006, because of its concrete theoretical meaning. However, in practice, it was found that the sigmoid function causes severe degradations. When the neural network becomes deep, the gradient vanishes by exponential rate along with backward propagation [30,31]. This phenomenon prevents the building of deeper models. ReLU as an activation function solved the problems to some extent [32–34]. As described by (2.3).

$$ReLU(x) = \begin{cases} 0 & x < 0 \\ x & x \geq 0 \end{cases} \tag{2.3}$$

The ReLU is a piecewise linear function of which the gradient does not vanish along its positive half axis. The gradient of the negative part of ReLU is 0, so that as long as the input lies on the negative side, the gradient disappears and the corresponding parameters are fixed, or namely, the neuron is dead. In order to improve this, other alternatives such as Leaky ReLU (LReLU) [35] and Parametric ReLU (PReLU) [36] have been proposed. There are also other types of activation functions [37], but they are usually used in rare edge cases, so we will not discuss them in detail.

The effectiveness of the neural network is usually described in two ways. Some researchers expect to derive the model from basic mathematical assumptions and axioms, just like other models. This method tends to guarantee the logic and structure of the neural network rigorously following the existing mathematical system. Others prefer to develop the neural network from the principles of neuroscience, which in general studies the basics of neurons and neural system. Researchers from this area tend to view the neural network as an abstract mathematical description of a biological neural system, assuming that it somehow behaves as being intelligent [38].

2.2.1 Abstraction of Biological Neural Activation

The secret of intelligence is still under cover, but the mainstream notion of the source of intelligence is that it comes from neural activities. The firing characteristics of a single neuron has been broadly studied. It shows that the stimulation from receptors and synapses from other neural cells accumulates in the neuron. If there is enough stimulation received by a certain neural cell, it fires and generates action potential. The potential change moves along its axon and reaches the synapses, where it releases transmitters to cross the cell membrane, received by the receptors of the target cell. The two major type of activation functions represent two different perspectives in modeling the action characteristic of a neuron. The sigmoid activation function (2.4), taking input from the previous layer, accumulates input signals and outputs a probability of whether the neuron fires.

$$h^{i+1} = sigmoid(W^{i+1}h^i + b^{i+1}) \tag{2.4}$$

$$h^{i+1} = ReLU(W^{i+1}h^i + b^{i+1}) \tag{2.5}$$

The RuLU function (2.5), corresponding to a rate coded neuron, takes the input signals and produces a positive value from 0 to infinity, indicating the rate or frequency of the fired signal, i.e., these two types are used in the second generation of neural networks. There is a third generation of neural networks, aka spiking neural networks, which take the timing pattern into account and assume the timing pattern as a part

of representation, namely temporal coding. However, the mathematical background of spiking neural network is still rather weak, and it will not be covered in this thesis.

2.2.2 Probabilistic Semantics

The more acceptable perspective to view neural network is from probability theory. Probability theory has always been in the center of dispute in history. The dispute involves the multiple aspects of probability theory. For instance, there is divergence for the origin of randomness (or noise). One opinion is that the noise exists because there are effective factors that are not observed yet, but the underlying interaction is deterministic. The other opinion believes even if all factors are taken into account, the interactions between variables still have random part. One of the most successful usage of probability theory is the modern quantum theory, which heavily relies on probabilistic result. Therefore, the dispute of the aforementioned problem was naturally brought to the physics area. Though there is no solution so far, the problem itself somehow follows the rules of probability, and probability theory has been widely used and is regarded as one of the foundation of machine learning.

The sigmoid neural network can be derived independently from probabilistic graphical model (PGM). Consider a Restricted Boltzmann Machine (RBM) [39] with Bernoulli random variables for both the visible layer v and the hidden layer h. The joint probability if the RBM is

$$P(h, v) = e^{b'v + c'h + h'Wv}.$$

Since each element of h is conditionally independent given v, the conditional probability of h given v can be calculated as follows:

$$P(h|v) = \prod_i P(h_i|v)$$

$$
\begin{aligned}
P(h_i|v) &= \frac{e^{b'v+c_ih_i+h_iW_iv}}{\sum_{\widetilde{h}_i} e^{b'v+c_i\widetilde{h}_i+\widetilde{h}_iW_iv}} \\
&= \frac{e^{c_ih+h_i'W_iv}}{\sum_{\widetilde{h}_i\in\{0,1\}} e^{c_i\widetilde{h}_i+\widetilde{h}_iW_iv}} \\
&= \frac{e^{c_ih_i+h_iW_iv}}{1+e^{c_i+W_iv}}
\end{aligned}
$$

Thus, the conditional probability of a hidden variable $h_i = 1$ is

$$
\begin{aligned}
P(h_i = 1|v) &= \frac{1}{1+e^{-(c_i+W_iv)}} \\
&= sigmoid(W_iv + c_i).
\end{aligned}
$$

Similarly,

$$
P(v_i = 1|h) = sigmoid(W_i^T h + b_i).
$$

When the variables are assumed Gaussian, similar inference can be made and the result is still a sigmoid network. Deep belief network (DBN) can be derived by stacking RBMs [40]. The probability of an n-layered DBN can be formulated by

$$
P(v, h^1, ..., h^n) = P(v|h^1)\prod_{i=2}^{n-1} P(h^{i-1}|h^i)P(h^{n-1}, h^n) \tag{2.6}
$$

A deep rendering model (DRM) [41] explains the probabilistic semantics of convolutional neural network (CNN) with RuLU function. The model defines a rendering function $R(c, g)$ to render an image (or other type of signals) $I = R(c, g)+noise$ from a category c and a nuisance variables g. By applying max-sum marginalization on the maximum a posteriori (MAP) classifier, the estimated category of an observation is

$$
\begin{aligned}
\hat{c} &= arg\max_{c\in C} p(c|I) = arg\max_{c\in C} p(I|c)p(c) \\
&= arg\max_{c\in C}\max_{g\in G} p(I|c, g)p(c)p(g)
\end{aligned}
$$

$$= arg \max_{c \in C} \max_{g \in G} \langle \eta(\theta_{cg}) | T(I) \rangle$$
$$= arg \max_{c \in C} \max_{g \in G} \langle w_{cg} | I \rangle + b_{cg}$$

where

$$w_{cg} = \frac{1}{\sigma^2} \mu_{cg} = \frac{1}{\sigma^2} R(c, g)$$
$$b_{cg} = \frac{1}{2\sigma^2} \|\mu_{cg}\|_2^2.$$

where $\mu_{cg} = E(R(c, g))$. Since the input may contain multiple objects, a single point is possible to belong to different configurations for a random variable. To prevent this from happening, a switching variable a is introduced. When $a = 0$, the rendered patch is replaced by all zeros. Otherwise it is left unchanged. The switching variable of each point is assumed to be independent, which is one of the weaknesses of DRM, because in practice they can be correlated. Then, the estimate becomes

$$\hat{c} = arg \max_{c \in C} \max_{g \in G} \max_{a \in A} a(\langle w_{cg} | I \rangle + b_{cg}) + b_{cga}$$
$$= arg \max_{c \in C} \max_{g \in G} ReLU(\langle w_{cg} | I \rangle + b_{cg})$$

The marginalization of a forms the ReLU function.

2.3 Distributed Representation

The distributed representation refers to storing the memory of an object into all the memory elements in contrast to one-hot representation, which uses a unique memory element to represent a certain object. One of the advantages of distributed representation is its large capacity. Imagine that a memory includes n binary elements. If we store information by one-hot representation, maximally n objects can be represented. However, if we store information by distributed representation, by the power law, 2^n objects in total can be represented. In addition, distributed representation shows automatic generalization ability, which alleviates the necessity of regularization, al-

lowing models to be more integrated. [42]

The embodiment of distributed representation in neural networks is reflected by its multi-layered structure. With the notion of deep structure, one expects that the deep neural network describes the data in a hierarchical form. Each layer represents a level of features of the data and each layer is independent from each other. By this notion, the data can be stored in a distributed pattern, according the neural networks to process very complex data accurately without losing generalization.

2.4 Universal Approximation

There used to be few models that are capable for interdisciplinary usage. The key factor of the generality of a model is the capability of approximating functions. As shown in (2.1), a linear function cannot approximate a nonlinear function. Kernel methods could transform nonlinear relations to a linear relation, making it easier to solve difficult problems, but kernels need to be hand-coded, lacking flexibility. Polynomials and splines are used as kernels to provide universal approximation ability. However, the accuracy of the approximation of these methods depends on the number of parameters. When high accuracy is needed, the large number of parameters will lead to overfitting, which forms a dilemma between model accuracy and generalization. The standard multi-layered feedforward neural network has been proved to be a universal approximator. For arbitrary bounded and non-constant activation function, the network having at least one hidden layer can approximate any function by arbitrary precision, provided a sufficient number of neurons. [43–45]

2.5 Variations of Neural Networks

Until now there has been a number of different neural networks proposed in the literature. Some of them are intended to achieve better performance and others can be used where basic neural networks cannot fit. According to the topology, these models can be categorized into two classes: feedforward networks and recurrent networks.

Feedforward means the graph constructed by the network contains no cycles, that is, the graph is a directed acyclic graph (DAG). Originally, every neuron in a certain layer is connected by all neurons from its previous layers. However, when neural network is used in image processing problems, the number of parameters becomes excessively huge, resulting in their inpracticality. In order to reduce the parameters as well as being inspired by the concept of receptive field of the visual system [46–49], convolutional layer has been developed and is widely used in vision and graphical area [50, 51]. A convolutional layer can be expressed by

$$h_j^{i+1} = \sigma(\sum_{k=1}^{n_i} W_{jk}^{i+1} * h_k^i + b_j^{i+1}) \tag{2.7}$$

where σ is any activation function, h_j^i is the jth channel of the ith layer and n_i is the number of channels of the ith layer. The symbol (*) denotes 2-D convolution. A neural network with convolutional layers is also called convolutional neural network (CNN).

In contrast, a recurrent neural network (RNN) has feedback connections, which allows the output to be time correlated, or namely, the network has a memory ability. This characteristic is appropriate when processing time series such as audio signal and heart rate signals [18,21,52,53]. However, the RNN has its own limitations. There are several areas for improvement. Firstly, due to the feedback connections, in the training of the RNN, one has to unfold the whole network through time. When develop with long series in training, the gradient vanishes during the long term propagation. This problem can be ameliorated by choosing a proper activation function [54]. Secondly, a major reason for using RNN is the memory ability. However, it is shown in practice that RNN "forgets" too quickly, meaning that it cannot make inference upon events between long intervals [55]. Long short term memory (LSTM) introduces memory cells integrated into the neural network. The control signals: input, output and forget, of the cells are provided by neurons and can be trained as normal neurons. In practice, the LSTM demonstrates decent long term memory unlike the plain RNN [56].

There are plenty of variations of neural networks not mentioned above, but they

can be classified into either of the two classes. Another perspective is to categorize the neural networks according ti their ability to generate samples. If they can, they are generative networks; otherwise, they are a discriminative networks. Usually a generative network has the probabilistic semantics and sometimes can be used as a discriminative model, or vice versa.

2.6 Batch Normalization

The training methods of neural networks involve deterministic and probabilistic approaches. Contrastive divergence is an efficient method to train a neural network from probability perspective and is often used for training a probabilistic model without supervision as the initial value for supervised training. Backpropagation is a deterministic method, which requires labeled data so that it is only valid for supervised learning.

Backpropagation works with optimization techniques. The target loss function defines the distance between the estimated values from the network and the labels. Since the loss relates to all the samples in the training set, it is ideal to feed all the recordings into the network before the computing gradients. However, in most cases when the training set is too large to calculate together, the stochastic gradient decent (SGD) must be used. The notion of SGD is feeding one recording at a time to compute gradient according to the sample fed, allowing the loss to move to local minimal via a zig-zag curve. Batch normalization technique allows feeding multiple recordings at a time called mini-batch to neural network to compute the gradient. Technically, it reduces the variance of the gradient and gives a better estimation of it during each step. Batch normalization allows training with larger learning rate and simultaneously behaves as regularization [57].

2.7 Loss Function and Optimization

Loss function defines the distance between the estimates and the ground truth. It is also known as cost function, object function or error. Usually the aim of optimization is to minimize the loss function, so that the estimates can be as close to the expected output as possible. The Euclidean distance is often used in regression models. It defines the distance within a regular Euclidean space from the point of ground truth in estimation. For the classification problem, the ground truth can be considered as locating at either 0 or 1 at each dimension and the loss still refers to the Euclidean distance [58]. Recent studies proposed machine learning methods based on manifold, where one assumes the recordings to only exist on a manifold of the space, where the distance is defined accordingly [59].

The computer architectures so far are good at processing discrete values but can barely process continuous variables. Thus, optimization methods based on gradient must be discretized. The training of neural network is virtually optimizing a non-linear function. Since backpropagation only takes effect when the gradient exists, the gradient based optimization is the exclusive choice for deterministic training. By choosing a different optimization scheme, the behavior of training varies dramatically. The most intuitive method is SGD, whose iteration formula is

$$\begin{aligned} \Delta\theta_{t+1} &= -\alpha \nabla L(\theta_t) \\ \theta_{t+1} &= \theta_t + \Delta\theta_{t+1}. \end{aligned} \tag{2.8}$$

Plain SGD converges slowly if the loss function is flat. SGD with momentum is given by

$$\begin{aligned} \Delta\theta_{t+1} &= \eta\Delta\theta_t - \alpha \nabla L(\theta_t) \\ \theta_{t+1} &= \theta_t + \Delta\theta_{t+1}. \end{aligned} \tag{2.9}$$

The momentum behaves similarly to inertia, keeping a certain speed of convergence even if the gradient is small [60]. The parameter α refers to learning rate, designating

the speed of convergence. However, a large learning rate may cause unstable iterations and an absence of converge. The parameter η indicates momentum, which should be assigned a value between 0 and 1.

Adaptive gradient methods are welcomed because of their handiness and robustness. AdaGrad [61] attempts to compute updates according to the information from previous iterations. AdaDelta [62] not only realizes adaptive gradient, but also cancels out the learning rate, leading to a more robust training process and removing a hyperparameter. In this thesis, we chose AdaDelta in all our experiments.

2.8 Platform Selection

Deep learning is a compound technique, involving plenty of components that require specific algorithms. Building a feasible deep learning program takes tremendous work, not to mention making it efficient and general. Considering that the main aim of this work is not developing new algorithms, we decided to work with the deep learning framework. Fortunately, because of the popularity of deep learning, many academic groups and enterprises released their own frameworks such as CNTK [63], which is from Microsoft, TensorFlow [64], which is from Google, Theano [65–67], which is a product of Université de Montréal and Caffe [68], which is owned by Berkeley Artificial Intelligence Research (BAIR) Lab.

We chose Caffe for the following reasons.

1. Caffe provides Matlab interface, which is easier for accessing the FHR database and previous results.

2. Caffe implements GPU acceleration, allowing faster training than on CPU.

3. Caffe uses json format to describe the network structure, which explicitly separates the model construction and underlying implementation.

Caffe also has disadvantages. Its encapsulation is on a very high level, so that it is relatively difficult to modify low level principles. For those who want to study the theories of deep learning or develop new learning algorithms, Caffe is not recommended.

2.9 Summary

In this section, an overview picture of deep learning is introduced. Neural networks are interpreted from two semantics describing an identical structure. We also briefly introduced distributed representation and universal approximation theory, which are two popular research directions in the theoretical study of deep learning. Then we discussed several typical characteristics of neural networks, including activation function, batch normalization and optimization methods. At the end, we addressed practical ways for implementing deep learning.

Chapter 3

Model Configuration

In this chapter, we describe the details of the model we used in our experiments. Most statistical models require choice of some hyperparameters to determine their basic characteristics. The selection procedure is also called model selection. The results of a model vary a lot according to the hyperparameters. Usually the model user would try different sets of hyperparameters in order to achieve the best performance. Being different from ordinary parameters, the hyperparameters are not estimated from data. Generally speaking, they are a part of the model assumptions. They are also viewed as the prior knowledge of structures in contrast to the unknown part which corresponds to parameters. Not all hyperparameters cannot be estimated. Akaike information criterion (AIC) [69] and Bayesian information criterion (BIC) [70] can estimate the degree of freedom according to specific balance between the complexity of the model and entropy. But they still need to run the model multiple times.

Deep learning models usually involve enormous number of parameters and these parameters can be categorized into groups. The large degree of freedom and the interaction between these parameters bring difficulties in determining the hyperparameters. Furthermore, the size of each layer, the number of layers and the types of activation functions are also included in the set of hyperparameters. Specifically, every neuron has at least two hyperparameters. One is to determine whether it includes a bias and the other indicates the type of activation function. The weights between layers depend on the number of neurons in both layer and layer type. On the

other hand, the different types of data also influence the determination of number of weights. For example, the convolutional layer is meaningful for visual and audio data, and we can use it in these settings so that wedramatically decrease the number of weights. There is no golden standard for selecting the hyperparameters for a neural network, but thanks to its popularity some rules of thumb exist. In the following sections, we will determine neural network structures gradually according to these rules.

3.1 Selection of Neural Network

The topological structure should be the first thing to consider. If the data are not time-correlated, a feedforward structure is sufficient, otherwise a recurrent structure must be considered. Even if we choose feedforward structure, there are also variations such as ResNet [71], highway-net [72] and dense network [73]. When the model needs to be very deep (more than tens of layers), the choice of these variations is crucial to ensure convergence.

The FHR data represent time series, which are time-correlated. Intuitively, the recurrent structure is the first topological choice. However, it is not suitable to use RNN directly for our data set. The RNN takes input data and output data simultaneously, which means when the input of an RNN is a time series, the output of it is also a time series with the same length. In the data set, the input is a time series whereas the output is a scaler for each sample. Therefore, the training of RNN becomes problematic, and this leads us to select feedforward structure as the basis of a model.

The data set consists of 552 recordings in total, which is not a big number to train a complex model. In order to perform cross validation, the size of training samples further deceases. This limits the upper bound of number of parameters and bounds the number of layers. On the other hand, due to the selection of feedforward structure, the input signal must be fed into neural network at one time. Since the length of the FHR signal could be long (several thousands of samples), the size of

neurons at the input layer can be very large. Then the number of weights will be enormous and the model will be unable to train. Convolutional layer is one of the solutions to this limitation, because the weights of a convolutional layer are shared between neurons. The weights are structured as a set of kernel convolving with the input. The size of the kernels could be much smaller than input.

The size of the input and the output are highly unmatched. Therefore the intermediate layers should shrink their sizes from top to bottom. The convolutional layers are good at reducing the number of weights, but they are not able to decrease the size of the data, since the convolution keeps the input and the output roughly of the same size. Fully connected layers can be manually configured in such a way but the number of weights between the neurons will grow drastically. There are two major methods that could allow the convolutional layers to reduce the data size. One is based on adding a pooling layer [74, 75] and the other on using stride convolution [76]. Pooling layer follows a convolutional layer, and it involves a pooling function which aggregates multiple input values and outputs one scalar. For example, a pooling layer with a window of length 4 will take 4 elements from the output of the previous convolutional layer and outputs 1 element as output. Then the window moves by 4 elements to where it is adjacent to the previous position. The procedure is repeated and as a result, the output decreases by 4 times. Stride convolution adopts similar ideas. the plain convolution computes the inner product between a window of the input and a kernel, and moves the window by one element every time. Stride convolution accepts a hyperparameter called stride, indicating how many elements the window moves each time. Thus, the size of the output is the size of the input divided by the given stride. Obviously, the information loss occurs within the stride convolution, and the larger stride we use, the more information will lose.

The number of layers should be considered carefully. According to the power of distributed representation, more layers could provide better performance and generalization but will be difficult to train. This is because the gradients vanish gradually during the process of propagation. The speed of vanishing varies when choose different activation function [30, 31, 77]. The sigmoid function suffers from the rapid

vanishing of gradient because the maximum gradient of the sigmoid function is 0.25. That is, the gradient vanishes by a factor of at least 0.25 per layer and shrinks exponentially with respect to the number of layers. Additionally, in most of the domain of the sigmoid function where it is flat the gradient is close to 0, which exacerbates the vanishing of gradients. The gradient of the Tanh function locates between 0 and 1 despite of the original point, where the gradient is 1. However it has the same problem as the sigmoid function. ReLU does not have a gradient vanishing problem, thus, the network can be stacked for many layers.

3.2 Model with Raw Data Input

One of the extraordinary powers of deep neural networks is their ability to disentangle complex and nonlinear features in the data. Thus, our first model is intended to make use of these advantages and will be end-to-end model at first. The end-to end model takes raw time series as input and directly produces the desired output, a label that indicates whether the fetus is in danger. The input of the model are FHR time series. The length of each sample will not exceed 90 minutes, and the recording of each sample lasts until delivery. The active pushing phase is the second stage of labor, in which the fetus is pushed actively to prepare for delivery. The second stage lasts maximally 30 minutes. Thus the size of the input is chosen to be the last 30 minutes. The signal is sampled at 4Hz, so that there are 7200 elements in the input layer. The output label is a scalar binary value, indicating the fetal status. It is also possible to output multiple statuses to indicate different risk levels of the current fetal status. It is easy to do so by sightly modifying the model with binary output, as pointed out in the sequel.

The final structure is determined by considering all of the factors mentioned above, and compromising their pros and cons. The entire feedforward neural network consists of two portions. The first portion includes five convolutional layers followed by a pooling layer, which is followed by the second portion, consisting of two fully connected layers and a linear layer. The size of the parameters of a fully connected

layer depends on the size of its input and output. For the first layer, the input size is identical to the output of the output of the last pooling layer. Since we are keeping the number of weights as small as possible, the convolutional layers must compress the feature size as much as possible and preserve the useful information as much as possible. The number of the kernels for every convolutional layer is chosen to be the power of 2. We choose powers of 2 only in order to achieve the best performance for parallel computing.

The output size of the convolutional layer can be computed as follows:

$$len_{out} = n[(len_{in} - len_{kernel})/stride + 1]. \tag{3.1}$$

where n is the number of kernels of a certain layer.

In most cases, the length of the kernel is much smaller than the length of the input. Thus, the decreasing factor is mainly affected by stride. On the other hand, the kernel plays the role of representing features, where the richness of features depends on the size of the kernel. Since the output size is irrelevant to the kernel size, the relation between output size and the kernel size is decoupled. Therefore, we can determine them respectively.

The first layer is responsible for extracting morphological patterns directly from the time series. Thus, we assume that a longer kernel would be required. we choose the length of the kernel of the first layer to be 128, and the number of kernels to be 16. The window size of the polling following the first convolutional layer is 4, which decreases the data length from the first convolutional layer by 4. The output size of the pooling layer can be computed as follows:

$$len_{out} = \lceil len_{in}/stride \rceil . \tag{3.2}$$

Thus, the output size of the first layer is

$$\begin{aligned} len_{conv\_out} &= 16 \times [(7200 - 128)/1 + 1] \\ &= 16 \times 7073, \end{aligned}$$

$$\begin{aligned} len_{pool\_out} &= 16 \times \lceil 7073/4 \rceil \\ &= 16 \times 1769. \end{aligned}$$

Therefore, the output data of the first layer is 1769.

For hidden layers, since they no more extract morphological features, the length of the kernels are chosen shorter than the first layer. However, the scaling rates are still kept the same. The number of kernels of all the hidden layers are all set to 32, slightly higher than the first layer considering the flexibility provided by the decrease of kernel length. According to (3.1) and (3.2), the output of the fifth layer is $32 \times 6 = 192$.

The last convolutional layer is followed by fully connected layers. Fully connected layers, in which each neuron of its output is connected to every neuron of its input, can configure the size of the output independently from the other hyperparameters. We choose 32 and 16 as the size of the output of the first and second fully connected layer, respectively. The following linear layer aggregates the 16 output from the second fully connected layer to 2 neurons. The two neurons represent healthy and non-healthy status, respectively, and represent the final classification. The output layer is implemented by a softmax layer. The softmax layer normalizes the output of each neuron to a value between 0 and 1, and improves the difference between them at the same time [78]. The impact of the softmax layer on each neuron can be represented as

$$\begin{aligned} O_i &= \frac{e^{I_i}}{Z} \\ Z &= \sum_k e^{I_k} \end{aligned}$$

where I and O denote input and output of the softmax layer respectively. Since the two output neurons are normalized, they can be seen as probabilities of unhealthy fetus conditioned on the input FHR series. Then we can choose the neuron that has larger probability as the estimated fetal status. Table A.4 presents a detailed summary of the hyperparameters of each layer.

3.3 Model with Feature Input

Deep neural networks have shown ability to disentangle nonlinear features from complex data structure. The previous section discussed the detail of building a deep convolutional neural network which takes FHR time series as input, finds features and does classification in an integrated model. This section, in contrast with the previous one, provides a description of a design of another convolutional neural network which takes features already computed from raw FHR time series. The model itself does not find features but only classifies the FHR signals by corresponding given features. In the next chapter, we will compare the performance of the two models and examine the disentangling capability of deep neural networks.

Although it is immediately reflected by the appearance when fetus suddenly enters severely dangerous status, the biomarkers remain relatively stable. In order to gain enough sensitivity, we calculate features within a window of certain length. To ensure time resolution, the window is sightly shifted through time. Then, for each FHR sample, we get a feature matrix F as follows:

$$F_{ij} = feat_i(FHR[t + j * dt : t + \tau + j * dt])$$

where F_{ij} denotes the jth value of feature i, dt denotes the distance that the window is shifted in each step and τ is the length of the window. The index of FHR starts from the delivery moment and increases to earlier times.

Apparently, the matrix F forms a 2-D feature map. However, there are two ways to map F to the input of a convolutional neural network since the data of standard convolutional layers are 3 dimensional. The first choice is to regard the feature matrix as a 2-D image (Figure B-2). The image convolves with different kernels given by each channel of the first convolutional layer, and produces an output for each channel. Obviously, the output of each channel is a 2-D array, and this characteristic is inherited to all subsequent layers until the first fully connected layer. Secondly, we can divide features into channels respectively (Figure B-3). Each channel of the output of the first convolutional layer is the function of the sum of convolutions between each

feature channel and corresponding kernels. We use the second configuration in the experiment for the following reasons:

1. 1-D channels keep consistent with the model with raw data input (Figure B-1), making it easier to implant the remaining part of the model and reducing the additional differences in their performances.

2. The scale of the features are different. Some features have values greater than a thousand and others might be less than 1. If they convolve with the same kernel, the contribution of the features that have small values will vanish in the noise of large values. Convolving them separately could prevent the vanishing of small valued features and ensure less information loss, since their kernels are disjoint and can have different scales.

3. The 2-D feature maps introduce significant amount of weights in the concatenation of the last convolutional layer and the first fully connected layer. Using the second configuration reduces the weights that need to learn.

The windows size is finally chosen as 30 minutes. After each evaluation the window moves 30 seconds to the past. Since we use the last 60 minutes to evaluate feature values, the size of the feature matrix is 14 by 60. The complete configuration of the model with feature input is listed in Table A.3.

3.4 Summary

In this chapter, we discussed the model selection of neural network used in the experiments for this thesis. The two models have some common hyperparameters, which are selected in the first section. After that, the second section describes the methods to determine the remaining part of the model which takes raw FHR data as input, and gives a sample to show how to compute the size of each layer. The third section focuses on the model with feature input. We at first introduce the features we use and the way to compute and arrange the input matrix. Then we compare the pros and cons of the two mapping schemes and finally decide which will be used in the

model. The next chapter will discuss the procedure of the experiment and the result. The performance of both models are evaluated in different situations.

Chapter 4

Model Evaluation

Training a deep model is usually considered to be difficult due to the lack of knowledge of its convergence and theories. The first barrier of training is the convergence. There are a lot of reasons for lack of convergence. For instance, a very small learning rate results in a very slow convergence especially when the objective function is flat, so that the convergence is not observable in a short period or the update value in each iteration is less than the minimal number a computer could represent. By contrast, a too large learning rate may cause divergence. The choice of activation function also affects the convergence. The convergence becomes extremely slow if the activation function is sigmoid and the input is far away from the origin. Recall that the sigmoid function is flat when the absolute value of the input is large. The ReLU function keeps constant gradient in positive half axis (see 2.3). However, once the input of a ReLU neuron falls on negative axis, the neuron will "die" forever. This phenomenon usually leads to a minute accurate loss but sometimes causes inability to converge. There are many other reasons related to the convergence, and these reasons usually involve plenty of aspects such as algorithmic stability, quantitative approximation, software defect and unknown reasons, which increase the difficulty of training when they interweave with each other.

In chapter 3 we have build two deep models for FHR classification. In this chapter, we train the two models respectively in different situations. The models are trained directly on data set at first but the performance is poor due to the small sample size

and highly unbalanced data distribution. Then we use data augmentation to produce a synthesized data set with sufficient and balanced sample size. The models trained on this set show good generalization. The models trained on the synthesized data set are also tested on real samples. The results show that the performance is good when the test cases are used to generate synthesized data, whereas if the test cases are not included in the samples used to generate, the performance is as good as training directly on the data set.

This chapter is organized as follows: first, we introduce the training configurations for both models. Second, we describe the methods used to generate augmented data set. Then, the strategy of cross validation is introduced and the result of each situation is presented. Finally, a qualitative analysis of the result will be discussed at the end of this chapter.

4.1 Training Strategy

The training of a neural network involves configurations of several aspects. In this section, we will discuss the settings of each aspect and the rationality behind them. The detailed hardware and software description is listed in Table A.1.

4.1.1 Batch Size

Batch normalization is a common technique to facilitate the speed of convergence. A large batch size offers better estimation of gradient and furthermore reduces curvature of the path of optimization. Intuitively, it is better to assign larger as large as possible and can be ideally equal to the number of recordings. Thus, in the experiment that train the model by original data set, we choose a batch size of 497, where the rest 55 recordings are left for validation. However, for the experiment that train the model based on synthesized data, choosing the number of recordings as the batch size causes video memory overflow, since the training set is too large to entirely store into memory. Therefore, after several tests, we chose a batch size of 8192 for feature input model, in which more than half of the video memory is occupied, whereas the batch

size of raw input model is 512, since the size of raw FHR data is much greater than that of features.

4.1.2 Optimization Algorithm

We choose AdaDelta as the optimization algorithm in all experiments. Unlike SGD, AdaDelta is not intended to use the gradient as an update value. The principle of AdaDelta is to simulate the Newton's method, which gives optimal estimation by the inverse Hessian matrix. Since the computation of the matrix inverse is slow and the chain rule of the second derivative is not intuitive, backpropagation framework is designed based on the first derivative. AdaDelta uses the following equation to approximate the inverse Hessian:

$$\Delta\theta = \frac{\frac{\partial L(\theta)}{\partial \theta}}{\frac{\partial^2 L(\theta)}{\partial \theta^2}}$$
$$\Rightarrow \frac{1}{\frac{\partial^2 L(\theta)}{\partial \theta^2}} = \frac{\Delta x}{\frac{\partial L(\theta)}{\partial \theta}}$$

where θ denotes a parameter of the loss function L.

Then use the l_2-norm of previous gradients to approximate the gradient of the next point, and use the same technique to approximate update value, $\Delta\theta$, which so far is unknown. Therefore, the update value can be computed as:

$$\Delta\theta_{t+1} = -\frac{\sqrt{E(\nabla L(\theta_{t+1})^2) + \varepsilon}}{\sqrt{E(\Delta x_t) + \varepsilon}} \nabla L(\theta_t)$$

where

$$E(x_t) = \rho E(x_{t-1}) + (1 - \rho) x_t.$$

We use the same configuration for all the experiments. The decay of l_2-norm, ρ, is set to a default value of 0.99 and the numerical stability, ε, is set to 1e-6. The momentum of the update is set to 0.9.

4.1.3 Maximum Iteration

The training of neural networks requires a huge amount of computation. One needs to set an upper limit for the number of iterations when we conduct cross validation. Figure B-4 shows the learning curves of training and testing accuracies in the first 10,000 iterations performed on raw data before experiments in order to find an appropriate iteration limit. From the example shown in figure B-4, we could see that the training and validation accuracies do not increase significantly after 2500 iterations. Leaving some margin, we chose 3000 as the number of iterations in the experiments.

4.1.4 Regularization

Neural networks usually contain a number of parameters and our two models are not exceptions. In order to prevent overfitting, it is usually required to add a regularization term to constrain the number of parameters. However, practical use of neural network shows that even if without regularization, deep models possess good generalization properties [42]. After a few pre-experiments, we decided not to use regularization.

4.2 Data Augmentation

The FHR data set included 552 items. Depending on the selection of the pH threshold, the number of positive samples varied from around 20 to 60. Obviously, such data set is too small and biased for training a neural network. In fact, we did this experiment and according to the validation result, the model usually had accuracy of around 0.8, but true positive rate was low, somewhere between 0.1 and 0.2, whereas the the true negative rate was always close to 1. There are two possible reasons that could explain this consequence.

1. The model learns little from these samples due to lack of sample capacity. Consider the following equation used to describe the probability of class prediction

after the softmax layer:

$$P(c|s,w) = P(s|c,w)P(c)$$

where s denotes the input sample to the neural network, c denotes a particular class and w represents the parameters in the model. According to the validation result, the prediction accuracy of a particular class is close to the proportion of that class in the training set as well as the validation sets. Thus, the term $P(s|c,w)$ is roughly a uniform distribution, which means the model did not learn to distinguish two classes.

2. From the perspective of sample space, the 7200-point time series can be seen as a point in a 7200-D space. The model learns a border that can divide the space into two subspaces, which represent two classes respectively. The border can be either plane or curved surface. Since the neural network is a universal approximator, a curved surface should provide a more accurate description. We can imagine if there are enough recordings for both the positive and negative classes, the model is able to find an exact and clear border between them. However, since the data set is small and highly biased towards negative class, the positive recordings scatter sparsely across the ideal positive region. Thus, for the region belonging to the positive class where a positive recording in the validation set is located but there is no recording in training set, the behavior of model is unknown. A visualized explanation is given by Figure B-5.

Although we do not have complete understanding to determine which reason is the major influence, the second reason is the preferred, since it could also explain the good performance on augmented data.

Data augmentation is a technique that could improve the stability and generalization of models by generating large amount of data from the original data. There are a number of methods for generation depending on the requirement and data type. One of the most common methods is resampling, in which we repeatedly choose a random sample from the original data set with replacement. This method doesn't work for

our case since direct resampling does not generate new points in the sample space. Finally we decided to resample from the frequency domain of the FHR, because previous study shows that frequency information of FHR has larger correlation between different bands.

In order to correctly label the generated samples, we treated the negative and the positive traces separatively. The procedure of generating the fictitious data of positive class is presented as follows:

1. Transform every positive FHR trace to the frequency domain by FFT.

2. Divide the frequency domain into several bands. Each band has equivalent energy summed across all traces. Since we have 44 FHR samples, there will be 44 samples for each band.

3. In order to generate a fictitious sample, we randomly choose a sample from each band independently, integrate all bands into a complete frequency representation, and then transform it to time domain by the inverse FFT.

The generation of negative data is similar. In practice, we divide the frequency domain into 7 bands. The window of each band is listed in Table A.2. It is obvious that most energy is distributed in the low frequency part, so that the window is short in the low frequency bands but long in the high frequency bands.

Following combination rule, for positive data, we can generate at most 44^7 for positive class and 447^7 for negative class, which is large enough for training any model. By limiting the number of samples generated for each class, we can get balanced training set and validation set. In the following experiments, we generated 100,000 traces for each class.

4.3 Cross Validation

Data are never enough for any machine learning method. The more data we have, the more accuracy and generalization we can get from the model. In practice, there

does not exist an infinite data set. Therefore, there are many techniques that are proposed to ensure and prove the generalization of a certain model based on limited data. Cross validation is one of the most widely used method to do so. One cross validation method follows the leave-one-out test scheme. While using cross validation on a data set, we train the model with the whole data set except one chosen sample. Then we test the model on the selected sample. After that we train the model again by leaving out another sample. The procedure will continue until every sample in the data set has been chosen as the test sample. There are also n-fold cross validation. For this method, we first divide the whole data set into n partitions, train the model with $n-1$ partitions in it and test on the remaining one. We repeat until every partition has been chosen once [79–81]. Since the training of neural network takes long time, we choose 10-fold cross validation to balance between running time and effectiveness.

In order to keep the randomness of the selection of the validation set, each partition of the validation set is chosen randomly from the fictitious data set without replacement. We pick up 10,000 samples from positive and negative set respectively for each test, and train the model by the remaining 180,000 samples. The same procedure is applied to the model with raw input and feature input.

4.3.1 Validation on Original Data

Although the cross validation can show the stability and generalization on generated data set with confidence and concreteness, it is not proved if it performs well on real data. Thus, the second experiment performs validation of the 10 models trained in each fold of the cross validation in the first experiment on the original data set used to generate augmented data. This validation result is stronger than the previous one because it shows the generalization of the model from fictitious data to real data.

A more radical validation method is the follows. Before we generate fictitious data, leave half of each class from the original data as the validation set. Then generate 200,000 samples in total by the remaining data and use them in the same way as in the second experiment. After that, we test the model by the validation set that

comes from the original data. The performance of this experiment is stronger than the second experiment, in which it shows the generalization to independent and real samples.

4.4 Performance

This section presents the validation result for the aforementioned three experiments:

1. 10-fold cross validation on augmented data set (Figure B-6);
2. 10-fold cross validation on original data set used to generate augmented data (Figure B-7);
3. Validation on original data not used to generate augmented data (Figure B-8).

For each experiment, we train and validate two models by the same data. The first model directly takes FHR trace as input, and the second model takes the feature matrix described in Section 3.3. Despite of accuracy, we use four additional indices, true positive rate (sensitivity), true negative rate (specificity), false positive rate (false alarm or type I error) and false negative rate (type II error), to thoroughly describe the performance of each model in every situation.

4.4.1 Experiment 1

The validation of feature input model is presented in Table A.5. The average accuracy of 10-fold cross validation is 0.9819, and both TP and TN are around 0.98. The equality shows that there is no bias in the training stage, and the high value of the three indices shows that the model learns good difference between given features of positive and negative FHR traces. The validation of raw input model is presented in Table A.6. The three indices are all above 0.999, which are slightly better than those of the feature input model. This difference indicates that the neural network extracts better features than the manually chosen features. However, since there is no big difference, we can conclude that these 14 features have covered almost all necessary information to distinguish positive and negative traces.

4.4.2 Experiment 2

This experiment uses the model trained in Experiment 1 and validates in different data set. The validation of feature input model is presented in Table A.7. The average accuracy of 10 validations is 0.9481, and both TP and TN are around 0.95. The indices are slightly lower than the Experiment 1 but still are very good. The validation of raw input model is presented in Table A.8. The three indices are all above 0.97, which are slightly better than those of the feature input model and again lower than those in Experiment 1. The same trend as Experiment 1 supports the conclusion we made in Section 4.4.1. In addition, we find that all the indices in this experiment are worse than those in Experiment 1, but there are only slightly different. This indicates that there is information loss that occurs when generating data, in which we only use frequency information. On the other hand, it seems we can distinguish classes with high accuracy by only inspecting its frequencies. This phenomenon can also be interpreted as that neural network learns the transformation from the similarity among large amount of samples and the correspondence between frequency bands and classes defined on pH value.

4.4.3 Experiment 3

Experiment 3 trains the feature input model by the newly generated data set from half of the original data, and validates the models on the rest half of original data. In order to provide sufficient validation traces, we lightly increased the pH threshold. The traces that pH value is less than or equal to 7.1 are positive traces and the traces that pH value is greater than or equal to 7.2 are negative traces. Therefore, we obtained 61 positive recordings and 375 negative recordings in total. Thus the number of the positive and negative recordings in validation set are 30 and 187, respectively. The result accuracy is 0.8065. TP and TN are 0.2 and 0.9037 respectively. This result, which is much worse than the other two experiments, is similar to the result of training directly on the original data set. The performance we have obtained so far is still illustrating that the lack of negative traces makes it difficult to maintain

the correlation between them. Once we use some of them in the training, the lack of correlation prevents the model to infer the class of validation samples correctly.

4.5 Summary

In this chapter, we discussed the whole process of evaluating the performances of deep learning methods for analyzing FHR signals. The first section introduces configurations of training, including the selection of optimization methods, batch size, and maximum iteration. The reason of why not use regularization is also discussed. Then, the data augmentation technique, which is used in our experiments to overcome the shortage and unbalance of original data set, is described in detail in Section 4.2. The subsequent section talks about the validation scheme and the last section presents the results for each experiment individually.

Chapter 5

Conclusion and Future Work

This thesis intends to apply state-of-the-art deep learning technology to an important obstetric problem, the interpretation of fetal heart rate signals, in order to estimate the oxygen deficiency status of the fetus before delivery. Due to the scalar label of recordings in the database, we chose to work with a convolutional neural network instead of a recurrent neural network, which is more suitable to process time series. We also use data augmentation to overcome the lack of data and the severe unbalance in the database. This provided sufficient data for training the neural network for balancing the distance between positive and negative performance.

5.1 Conclusion

Deep learning has been proved to be effective in many areas by its capacity to represent complex nonlinear features by its intrinsic generalization. On the other hand, it has limitations. For example, it requires large amount of data to discover underlying structures whereas conventional models use hand-coded prior instead. This limitation often brings compromise for certain data sets, and allows us to select optimal model between deep learning and other models. The difference between training by original data set and by augmented data set presents this compromise. When the data size is not large enough, the stability and effectiveness may be significantly influenced.

We performed three experiments to evaluate detailed performance and generaliza-

tion of our models. From the result, we make following conclusions:

1. One needs to prepare enough data to train a neural network. The lack of data may cause the model to always output a single class which occupies the highest proportion in the training set.

2. Data augmentation is helpful to balance the data set and to provide sufficient training data to the neural network. The model trained on augmented data has good performance and generalization. However, it will not work if the training set is not general enough.

3. The neural network does excellent job in extracting unknown features. In our experiment, the accuracy is obtained with no prior knowledge provided to the raw data input.

4. According to the comparisons of results between the raw data input model and the feature input model, the one with carefully selected features performs equally well to that with raw data processed by the neural network. The good performances of the two models indicate that the manually selected features are able to provide most of the useful information.

5. We conjecture that the main reason of the low TP is because of the the sparse distribution of positive samples. The model cannot estimate the accurate border between the positive and the negative spaces. As long as the validation sample is far away from the training samples in the sample space, the behavior of the model is difficult to predict. It is intuitive that even if there are no large samples, as long as the training set includes corner cases, the estimated border will still be accurate. The bad thing is, unless we have already known the shape of the border, or it is unable to verify whether a sample is corner case.

5.2 Future Work

This thesis studied the behavior of deep learning methods in the classification of FHR signals, and found interesting phenomena which were described in this work. Some of them are good results, showing the power of deep learning and the correct way to apply it to certain data. Some of them point out the weak points in which current deep learning methods are not good at processing certain data. These limitations lead us to further study of the methods including how to apply deep learning to certain data, and in particular to FHR traces. New forms of deep learning methods that can perform better than the ones from this thesis also of great interest.

According to the problems we encountered, some of the following items could be important directions for study.

1. Collect enough data and reverify Experiment 3. Experiment 3 is important because it shows the generalization on independent practical data. If the model performs well in Experiment 3, it will provide a strong impetus for further study.

2. Collect time-varying labels, which indicate the status of the FHR signal over time. A series of outputs enables the usage of RNN, which is more meaningful for processing time series. Additionally, the variations of RNN are of the most interest in industry and academia and have the most cut-in-edge use cases. The most recent concepts such as attention and memory cell are coming from the investigation of it. RNN is Turing complete, and is theoretically being able to learn arbitrary sequential logic.

3. Improve the data augmentation method. Our data augmentation is based on individual frequency features. It is highly possible that some useful information is not included in frequencies. Furthermore, the distribution of the generated data has not been verified. Finding new augmentation methods could reduce information loss and increase generalization.

4. DRM (see Section 2.2.2) provides the method that train deep convolutional neural network by the EM algorithm. It is worth trying this and introduce

graphical model to FHR.

5. Change the output model to continuous variables, so that it can predict biomarkers, offer suggestions to clinical doctors and leave the decision out of the system. This is useful to circumvent potential legal responsibility and blames from morality. Similar to autopilot vehicles, FHR system connects directly to life security. Responsibility ownership should be seriously considered in case of any accident that world happen.

Appendix A

Tables

Table A.1: Evaluation Platform

Item	**Description**
OS	Windows 7 Enterprise Service Pack 1 64-bit
CPU	Intel(R) Core(TM) i5-2400 @ 3.10GHz
RAM	8GB
GPU	NVIDIA GeForce GTX 760
Caffe	Windows 1.0.0-rc3
MatLab	R2015b

Table A.2: Band Selection for Data Augmentaion

No. of Band	**Band Interval in FFT**	**Band Interval in Frequency (Hz)**
1	1	0
2	2 : 3	$5.56 \times 10^{-4} : 1.11 \times 10^{-3}$
3	4 : 19	$1.67 \times 10^{-3} : 1.00 \times 10^{-2}$
4	20 : 131	$1.06 \times 10^{-2} : 7.22 \times 10^{-2}$
5	132 : 700	$7.28 \times 10^{-2} : 3.88 \times 10^{-1}$
6	701 : 1800	$3.89 \times 10^{-1} : 9.99 \times 10^{-1}$
7	1801 : 3600	1 : 2

Table A.3: Network Setting for Feature Matrix Input

Layer	**Type**	**Weights**	**Input Dimension**	**Output Dimension**
1	Conv	14x16x16	14x60	16x45
2	Pool		16x45	16x23
3	Conv	16x32x8	16x23	32x16
4	Pool		32x16	32x8
5	Conv	32x64x4	32x8	64x5
6	Pool		64x5	64x3
7	FC	64x3x64	64x3	64
8	FC	64x16	64	16
9	Linear	16x2	16	2
10	SoftMax		2	2

Table A.4: Network Setting for Raw Data Input

Layer	Type	Weights	Input Dimension	Output Dimension
1	Conv	1x16x128	1x7200	16x7073
2	Pool		16x7073	16x1769
3	Conv	16x32x32	16x1769	32x1738
4	Pool		32x1738	32x435
5	Conv	32x32x16	32x435	32x420
6	Pool		32x420	32x105
7	Conv	32x32x8	32x105	32x98
8	Pool		32x98	32x25
9	Conv	32x32x4	32x25	32x22
10	Pool		32x22	32x6
11	FC	32x6x32	32x6	32
12	FC	32x16	32	16
13	Linear	16x2	16	2
14	SoftMax		2	2

Table A.5: Cross-validation Result on Augmented Data of Neural Network with Feature Input

Fold	Accuracy	True Pos.	True Neg.	False Pos.	False Neg.
1	0.9825	0.98	0.985	0.015	0.02
2	0.9745	0.978	0.971	0.029	0.022
3	0.982	0.99	0.974	0.026	0.01
4	0.981	0.983	0.979	0.021	0.017
5	0.9755	0.97	0.981	0.019	0.03
6	0.985	0.984	0.986	0.014	0.016
7	0.991	0.993	0.989	0.011	0.007
8	0.984	0.985	0.983	0.017	0.015
9	0.989	0.991	0.987	0.013	0.009
10	0.9745	0.976	0.973	0.027	0.024
Ave	0.9819	0.983	0.9808	0.019	0.017

Table A.6: Cross-validation Result on Augmented Data of Neural Network with Raw Input

Fold	**Accuracy**	**True Pos.**	**True Neg.**	**False Pos.**	**False Neg.**
1	0.9995	1	0.9990	0.0010	0
2	1	1	1	0	0
3	1	1	1	0	0
4	0.9990	0.9980	1	0	0.0020
5	1	1	1	0	0
6	1	1	1	0	0
7	0.9995	0.9990	1	0	00010
8	1	1	1	0	0
9	1	1	1	0	0
10	1	1	1	0	0
Ave	0.9998	0.9997	0.9999	0.0001	0.0003

Table A.7: Validation Result on Original Data of Neural Network with Feature Input

Fold	**Accuracy**	**True Pos.**	**True Neg.**	**False Pos.**	**False Neg.**
1	0.9613	1	0.9575	0.0425	0
2	0.9511	0.9773	0.9485	0.0514	0.0227
3	0.9430	0.9773	0.9396	0.0604	0.0227
4	0.9450	0.9545	0.9441	0.0559	0.0454
5	0.9491	0.9318	0.9508	0.0492	0.0682
6	0.9470	0.9545	0.9463	0.0537	0.0454
7	0.9348	0.9773	0.9306	0.0694	0.0227
8	0.9674	0.9318	0.9709	0.0291	0.0682
9	0.9593	0.9545	0.9597	0.0403	0.0454
10	0.9226	0.8409	0.9306	0.0694	0.1591
Ave	0.9481	0.9499	0.9479	0.0521	0.0500

Table A.8: Validation Result on Original Data of Neural Network with Raw Input

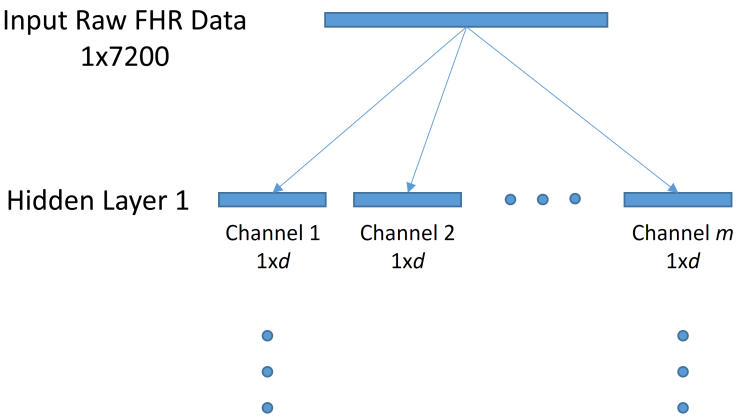
Fold	**Accuracy**	**True Pos.**	**True Neg.**	**False Pos.**	**False Neg.**
1	0.9919	1	0.9911	0.0089	0
2	0.9878	1	0.866	0.0134	0
3	0.9878	1	0.9866	0.0134	0
4	0.9878	0.9773	0.9888	0.0112	0.0227
5	0.9817	1	0.9799	0.0201	0
6	0.9919	1	0.9911	0.0089	0
7	0.9898	1	0.9888	0.0112	0
8	0.9796	1	0.9776	0.0224	0
9	0.9776	1	0.9754	0.0246	0
10	0.9939	1	0.9933	0.0067	0
Ave	0.9860	0.9977	0.9739	0.0141	0.0023

Table A.9: Validation Result on Original Data Not Used in Generation of Neural Network with Feature Input

Accuracy	**True Pos.**	**True Neg.**	**False Pos.**	**False Neg.**
0.8065	0.2000	0.9037	0.0963	0.8000

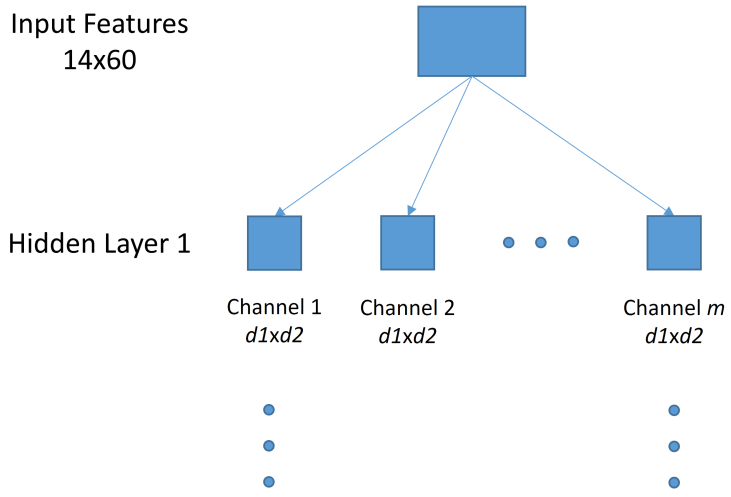
Appendix B

Figures



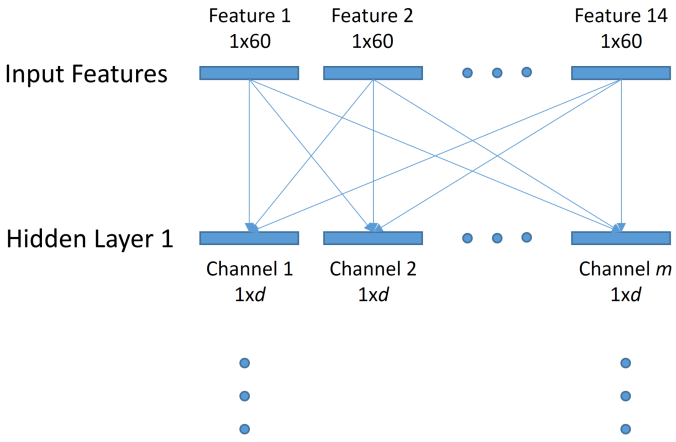
Every arrow denotes convolution with a certain kernel.

Figure B-1: Raw FHR Data as Input



Every arrow denotes convolution with a certain kernel.

Figure B-2: FHR Features as 2-D Input



Every arrow denotes convolution with a certain kernel.

Figure B-3: FHR Features as Saperatd 1-D Input

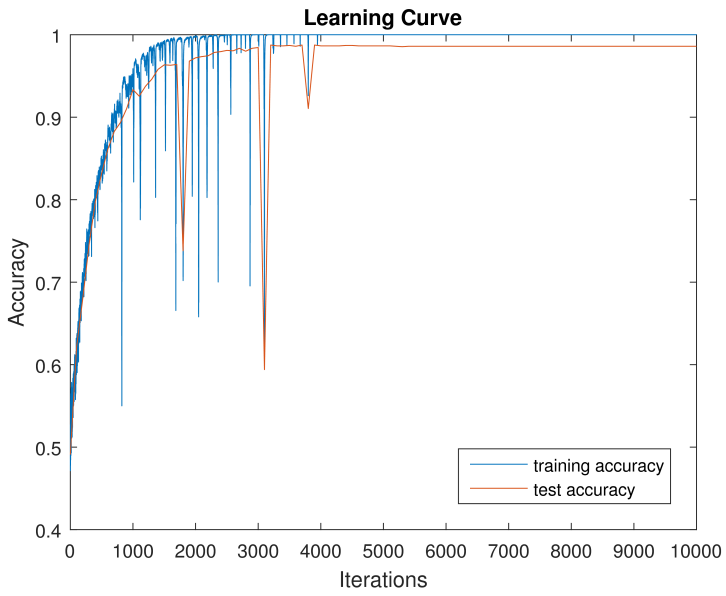


Figure B-4: Learning Curve of Training by Raw Data

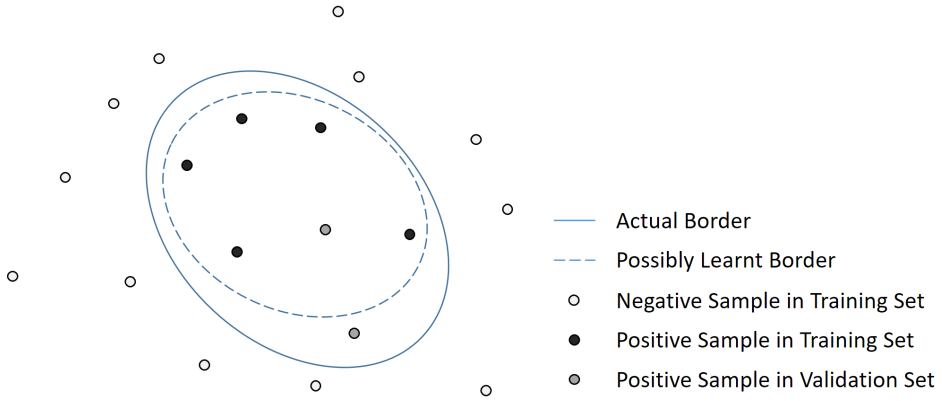


Figure B-5: An explaination of the bad performance on original data set

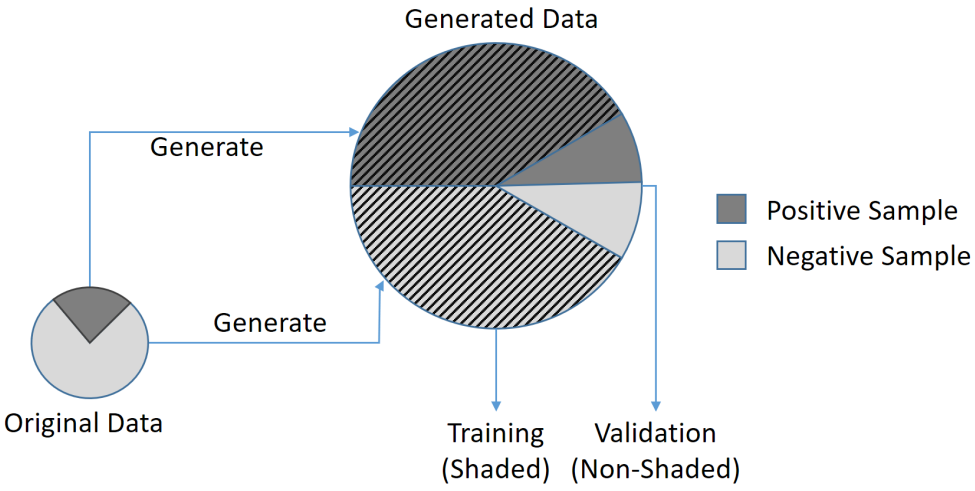


Figure B-6: Experiment Scheme of Cross Validation

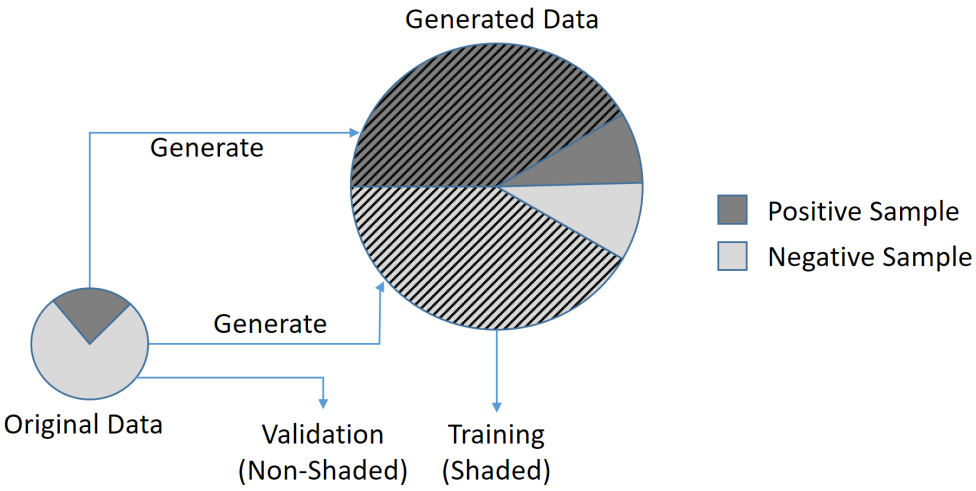


Figure B-7: Experiment Scheme of Validation on Original Data

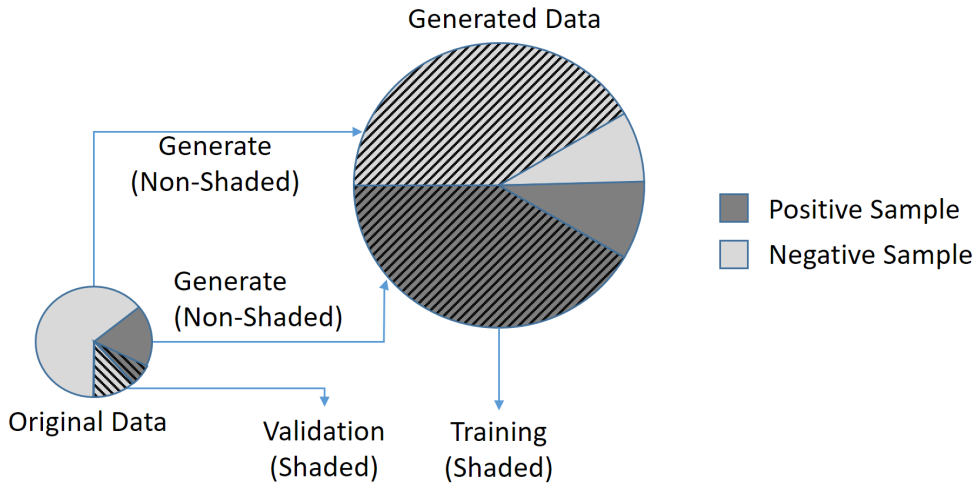


Figure B-8: Experiment Scheme of Validation on Left-out Original Data

Bibliography

[1] Geoffrey E. Hinton, Simon Osindero, and Yee Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–54, 2006.

[2] Frank Rosenblatt. Principles of neurodynamics. perceptrons and the theory of brain mechanisms. Technical report, DTIC Document, 1961.

[3] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, DTIC Document, 1985.

[4] F Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain., 1958.

[5] Ary L Goldberger, Luis A N Amaral, Leon Glass, Jeffrey M Hausdorff, Plamen Ch Ivanov, Roger G Mark, Joseph E Mietus, George B Moody, Chung-kang Peng, and H Eugene Stanley. PhysioBank, PhysioToolkit, and PhysioNet: Components of a New Research Resource for Complex Physiologic Signals. 2000.

[6] Václav Chudáček, Jiří Spilka, Miroslav Burša, Petr Janků, Lukáš Hruban, Michal Huptych, and Lenka Lhotská. Open access intrapartum CTG database. *BMC pregnancy and childbirth*, 14:16, 2014.

[7] Zarko Alfirevic, Declan Devane, G M Gyte, and Others. Continuous cardiotocography (CTG) as a form of electronic fetal monitoring (EFM) for fetal assessment during labour. *Cochrane Database Syst Rev*, 3(3):CD006066, 2006.

[8] Lawrence D Devoe, Michael Ross, Clayton Wilde, Maureen Beal, Andrej Lysikewicz, Jeffrey Maier, Victor Vines, Isis Amer-Wåhlin, Håkan Lilja, Håkan Norén, and Others. United States multicenter clinical usage study of the STAN 21 electronic fetal monitoring system. *American journal of obstetrics and gynecology*, 195(3):729–734, 2006.

[9] Roger K Freeman, Thomas J Garite, Michael P Nageotte, and Lisa A Miller. *Fetal heart rate monitoring*. Lippincott Williams & Wilkins, 2012.

[10] Shiying Dong, Boualem Boashash, Ghasem Azemi, Barbara E Lingwood, and Paul B Colditz. Detection of perinatal hypoxia using time-frequency analysis of heart rate variability signals. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 939–943. IEEE, 2013.

[11] Jirí Spilka, V Chudáček, Michal Koucký, Lenka Lhotská, Michal Huptych, Petr Janků, George Georgoulas, and Chrysostomos Stylios. Using nonlinear features for fetal heart rate classification. *Biomedical Signal Processing and Control*, 7(4):350–357, 2012.

[12] Shishir Dash. Bayesian methods for feature extraction and classification of fetal heart rate signals. (May), 2014.

[13] Antoniya Georgieva, Stephen J Payne, Mary Moulden, and Christopher W G Redman. Artificial neural networks applied to fetal monitoring in labour. *Neural Computing and Applications*, 22(1):85–93, 2013.

[14] Yasuaki Noguchi, Fujihiko Matsumoto, Kazuo Maeda, and Takashi Nagasawa. Neural network analysis and evaluation of the fetal heart rate. *Algorithms*, 2(1):19–30, 2009.

[15] Philip J Steer. Has electronic fetal heart rate monitoring made a difference? In *Seminars in Fetal and Neonatal Medicine*, volume 13, pages 2–7. Elsevier, 2008.

[16] J.˜R.˜R. Uijlings, K.˜E.˜A. van de Sande, T Gevers, and A.˜W.˜M. Smeulders. Selective Search for Object Recognition. 104(2):154–171, 2013.

[17] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *Iclr*, pages 1–14, 2015.

[18] Alex Graves, Santiago Fernandez, Faustino Gomez, and Jurgen Schmidhuber. Connectionist Temporal Classification : Labelling Unsegmented Sequence Data with Recurrent Neural Networks. *Proceedings of the 23rd international conference on Machine Learning*, pages 369–376, 2006.

[19] Ruhi Sarikaya, Geoffrey E Hinton, and Anoop Deoras. Natural Language Understanding. 22(4):778–784, 2014.

[20] Ossama Abdel-hamid, Abdel-rahman Mohamed, Hui Jiang, Li Deng, Gerald Penn, and Dong Yu. Convolutional Neural Networks for Speech Recognition. 22(10):1533–1545, 2014.

[21] Dario Amodei, Rishita Anubhai, Eric Battenberg, Case Carl, Jared Casper, Bryan Catanzaro, Jingdong Chen, Mike Chrzanowski, Adam Coates, Greg Diamos, Erich Elsen, Jesse Engel, Linxi Fan, Christopher Fougner, Tony Han, Awni Hannun, Billy Jun, Patrick LeGresley, Libby Lin, Sharan Narang, Andrew Ng, Sherjil Ozair, Ryan Prenger, Jonathan Raiman, Sanjeev Satheesh, David Seetapun, Shubho Sengupta, Yi Wang, Zhiqian Wang, Chong Wang, Bo Xiao, Dani Yogatama, Jun Zhan, and Zhenyao Zhu. Deep-speech 2: End-to-end speech recognition in English and Mandarin. *arXiv*, page 28, 2015.

[22] Theo Gevers and Arnold W M Smeulders. Segmentation as Selective Search for Object Recognition. (2):1879–1886, 2011.

[23] a J Holden, D J Robbins, W J Stewart, D R Smith, S Schultz, M Wegener, S Linden, C Hormann, C Enkrich, C M Soukoulis, D Schurig, a J Taylor, C Highstrete, M Lee, R D Averitt, P Markos, D Mcpeake, S a Ramakrishna, J B Pendry, V M Shalaev, M Maksimchuk, D Umstadter, W Chen, Y R Shen, and J V Moloney. Reducing the Dimensionality of Data with Neural Networks. *Science (New York, N.Y.)*, 313(July):504–507, 2006.

[24] Geoffrey Hinton, Li Deng, Dong Yu, George Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Vincent Vanhoucke, Patrick Nguyen, Tara Sainath, and Brian Kingsbury. Deep Neural Networks for Acoustic Modeling in Speech Recognition. pages 1–27.

[25] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[26] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

[27] Marvin L Minski and Seymour A Papert. Perceptrons: an introduction to computational geometry. *MA: MIT Press, Cambridge*, 1969.

[28] Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, and Others. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471–476, 2016.

[29] Yann A. LeCun, Léon Bottou, Genevieve B. Orr, and Klaus Robert Müller. Efficient backprop. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7700 LECTU:9–48, 2012.

[30] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, and Jürgen Schmidhuber. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, 2001.

[31] Sepp Hochreiter. Untersuchungen zu dynamischen neuronalen Netzen. *Diploma, Technische Universität München*, page 91, 1991.

[32] Richard H R Hahnloser, Rahul Sarpeshkar, Misha A Mahowald, Rodney J Douglas, and H Sebastian Seung. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature*, 405(6789):947–951, 2000.

[33] Richard H R Hahnloser, H Sebastian Seung, and Jean-Jacques Slotine. Permitted and forbidden sets in symmetric threshold-linear networks. *Neural computation*, 15(3):621–638, 2003.

[34] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep Sparse Rectifier Neural Networks. In *Aistats*, volume 15, page 275, 2011.

[35] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. ICML*, volume 30, 2013.

[36] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1026–1034, 2015.

[37] Luke B Godfrey and Michael S Gashler. A continuum among logarithmic, linear, and exponential functions, and its potential to improve generalization in neural networks. *arXiv preprint arXiv:1602.01321*, 2016.

[38] David Kriesel. *A Brief Introduction to Neural Networks*. 2013.

[39] Paul Smolensky. Information processing in dynamical systems: Foundations of harmony theory. Technical report, DTIC Document, 1986.

[40] Yoshua Bengio. *Learning Deep Architectures for AI*, volume 2. 2009.

[41] Ankit B Patel, Tan Nguyen, and Richard G Baraniuk. A Probabilistic Theory of Deep Learning. *arXiv*, pages 1–56, 2015.

[42] Paul Smolensky. Parallel distributed processing: explorations in the microstructure of cognition, vol. 1. chapter Information processing in dynamical systems: foundations of harmony theory. *MIT Press, Cambridge, MA, USA*, 15:18, 1986.

[43] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[44] Chen Debao. Degree of approximation by superpositions of a sigmoidal function. *Approximation Theory and its Applications*, 9(3):17–28, 1993.

[45] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, 1991.

[46] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[47] Dan Ciregan, Ueli Meier, and Jürgen Schmidhuber. Multi-column deep neural networks for image classification. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 3642–3649. IEEE, 2012.

[48] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202, 1980.

[49] David H Hubel and Torsten N Wiesel. Receptive fields and functional architecture of monkey striate cortex. *The Journal of physiology*, 195(1):215–243, 1968.

[50] Yann LeCun and Others. LeNet-5, convolutional neural networks. *URL: http://yann. lecun. com/exdb/lenet*, 2015.

[51] Masakazu Matsugu, Katsuhiko Mori, Yusuke Mitari, and Yuji Kaneda. Subject independent facial expression recognition with robust face detection using a convolutional neural network. *Neural Networks*, 16(5):555–559, 2003.

[52] Pankaj Malhotra, Lovekesh Vig, Gautam Shroff, and Puneet Agarwal. Long short term memory networks for anomaly detection in time series. In *Proceedings*, page 89. Presses universitaires de Louvain, 2015.

[53] Felix Gers. *Long short-term memory in recurrent neural networks*. PhD thesis, Universität Hannover, 2001.

[54] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. *ICML (3)*, 28:1310–1318, 2013.

[55] Y Bengio. Learning long-term dependencies with gradient descent is difficult, 1994.

[56] Hasim Sak, Andrew W Senior, and Françoise Beaufays. Long short-term memory recurrent neural network architectures for large scale acoustic modeling. In *INTERSPEECH*, pages 338–342, 2014.

[57] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[58] Christopher M Bishop. *Neural networks for pattern recognition*. Oxford university press, 1995.

[59] Junping Zhang, Hua Huang, and Jue Wang. Manifold learning for visualizing and analyzing high-dimensional data. *IEEE Intelligent Systems*, 25(4):54–61, 2010.

[60] Léon Bottou. Stochastic gradient descent tricks. In *Neural Networks: Tricks of the Trade*, pages 421–436. Springer, 2012.

[61] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[62] Matthew D Zeiler. ADADELTA: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[63] Dong Yu, Adam Eversole, Mike Seltzer, Kaisheng Yao, Zhiheng Huang, Brian Guenter, Oleksii Kuchaiev, Yu Zhang, Frank Seide, Huaming Wang, and Others. An introduction to computational networks and the computational network toolkit. Technical report, Technical report, Tech. Rep. MSR, Microsoft Research, 2014, 2014. research. microsoft. com/apps/pubs, 2014.

[64] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, and Others. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[65] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: A CPU and GPU math compiler in Python. In *Proc. 9th Python in Science Conf*, pages 1–7, 2010.

[66] Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, James Bergstra, Ian Goodfellow, Arnaud Bergeron, Nicolas Bouchard, David Warde-Farley, and Yoshua Bengio. Theano: new features and speed improvements. *arXiv preprint arXiv:1211.5590*, 2012.

[67] James Bergstra, Frédéric Bastien, Olivier Breuleux, Pascal Lamblin, Razvan Pascanu, Olivier Delalleau, Guillaume Desjardins, David Warde-Farley, Ian Goodfellow, Arnaud Bergeron, and Others. Theano: Deep learning on gpus with python. In *NIPS 2011, BigLearning Workshop, Granada, Spain*. Citeseer, 2011.

[68] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678. ACM, 2014.

[69] Hirotugu Akaike. A new look at the statistical model identification. *IEEE transactions on automatic control*, 19(6):716–723, 1974.

[70] Gideon Schwarz and Others. Estimating the dimension of a model. *The annals of statistics*, 6(2):461–464, 1978.

[71] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *Arxiv.Org*, 7(3):171–180, 2015.

[72] Gao Huang, Zhuang Liu, Kilian Q Weinberger, and Laurens van der Maaten. Densely connected convolutional networks. *arXiv preprint arXiv:1608.06993*, 2016.

[73] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015.

[74] Dominik Scherer, Andreas Müller, and Sven Behnke. Evaluation of pooling operations in convolutional architectures for object recognition. In *International Conference on Artificial Neural Networks*, pages 92–101. Springer, 2010.

[75] Dan C Ciresan, Ueli Meier, Jonathan Masci, Luca Maria Gambardella, and Jürgen Schmidhuber. Flexible, high performance convolutional neural networks for image classification. In *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, volume 22, page 1237, 2011.

[76] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

[77] Ronen Eldan and Ohad Shamir. The Power of Depth for Feedforward Neural Networks. *arXiv preprint arXiv:1512.03965*, 2015.

[78] Christopher M Bishop. Pattern recognition. *Machine Learning*, 128, 2006.

[79] Pierre A Devijver and Josef Kittler. *Pattern recognition: A statistical approach*. Prentice hall, 1982.

[80] Ron Kohavi and Others. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Ijcai*, volume 14, pages 1137–1145, 1995.

[81] Seymour Geisser. *Predictive inference*, volume 55. CRC press, 1993.