Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

A New Topology of Wavelet Neural Network for Fast Deep Learning of Big Data Applications

Ph.D. candidate:
Haitao Cao

Supervisor:
Prof. Alessandro Paccagnella

Co-Supervisor:
Prof. Gaoyong Luo

Coordinator:
Prof. Andrea Neviani

Ph.D. School in
Information Engineering

Department of Information Engineering
University of Padova
2020

UNIVERSITÀ DEGLI STUDI DI PADOVA

Università degli Studi di Padova/ University of Padova

Department of Information Engineering

Ph.D. course in Information Engineering

Information and Communication Technologies

XXXII

A New Topology of Wavelet Neural Network for Fast Deep Learning of Big Data Applications

Una Nuova Topologia della Rete Neurale Wavelet per un Apprendimento Rapido e Approfondito di Applicazioni per Big Data

Coordinator: Prof. Andrea Neviani

Supervisor: Prof. Alessandro Paccagnella

Co-Supervisor: Prof. Gaoyong Luo

Ph.D. student: Haitao Cao

Abstract

With technological innovations progressing rapidly, big data is now produced from various applications. Due to its capability of handling complex problems, Deep Neural Network (DNN) has become one of the fastest-growing and most exciting areas of Machine Learning (ML) in the data-intensive field. However, it remains a challenge to train such a deep structure, facing the gradient problems and slow convergence speed. In big data analysis, the original dataset is usually characterized as high dimensionality and the key features are very often buried in the noise, which increases computational complexity and slows down DNN training. Through research, it is noted that the mathematical properties of activation functions may be attributed to the gradient exploding or vanishing problem, resulting in training failure or with big errors. As the networks go deeper and deeper to deal with more and more complex real-world problems, many more trainable parameters of DNN model need to be trained with higher computational efficiency.

In this thesis, we propose a new topology of Wavelet Neural Network (WNN) termed as FDIDWT-MEXHACT-NN or simply FDMHNN for fast Deep Learning (DL) of big data applications. With preprocessing it would have two concatenation components.

Since Fast Fourier Transform (FFT) performed at the beginning of hidden layers for feature pre-extraction and noise reduction in the training process is demonstrated to improve computational efficiency as the preliminary work, based on the Correlation Fractal Dimension (CFD) theory and Inverse Discrete Wavelet Transform (IDWT), the first component of FDMHNN implements a new type of preprocessing method, i.e., FDIDWT, to transform the original dataset to a low-dimensional and feature-extracted one for low computational complexity and fast DNN training.

The second component of FDMHNN is a Multilayer Perceptron (MLP) model equipped with the derived Activation Function and Derivative Function pair (ADF), i.e., Tight Frame Wavelet Activation Function (TFWAF) and Tight Frame Wavelet Derivative Function (TFWDF) of Mexican hat wavelet. These two functions, termed as TFMH, are designed to normalize and constrain the training data of hidden layers to constant energy that stabilizes the training and speeds up convergence further. The nonlinearity of wavelet functions can strengthen the

learning capacity of DNN model, while the sparse property of wavelets derived can reduce the computational complexity of training process and enhance the robustness of model. In addition, TFMH can also alleviate the overfitting problem.

The proposed FDMHNN model is characterized as stable, fast construct and fast convergent, and is evaluated by various experiments. Through big data analysis for nonlinear system modelling and speech signal processing, and particularly feature extraction and classification for the astronomical data collected from a nano-satellite simulation environment and the real *Fermi*-LAT source catalog (3FGL), experiments performed demonstrate that the deep FDMHNN model achieves more stable training and faster convergence than the traditional deep MLPs in the binary classification tasks as well as complicated applications.

Contents

List of Figures

List of Tables

Symbols

$\boldsymbol{x}, \boldsymbol{y}$	vector
$\mathbf{A}, \mathbf{B}$	matrix
$\mathbb{A}, \mathbb{B}$	set
N	the number of samples in datasets
E	the number of attributes in datasets
S	the number of classes in datasets
$\|\|\cdot\|\|_2, \|\|\cdot\|\|$	$L2$-norm
$I(\cdot)$	indication function, $I = 1$ if $\cdot$ is true, otherwise $I = 0$
$L^2(\mathbb{R})$	one-dimensional measurable, square-integrable function space
J	wavelet decomposition level
$\downarrow 2$	downsampling: removing one for every two points
$\uparrow 2$	upsampling: inserting one zero between every two points
η	learning rate
$f(\cdot)$	the activation function of hidden layers
$f'(\cdot)$	the derivative function of hidden layers
$\sigma(\cdot)$	the activation function of output layers
$\sigma'(\cdot)$	the derivative function of output layers
$\boldsymbol{\delta}$	the errors of hidden layers
$I_a(\cdot)$	the information content of attributes
$I_c(\cdot)$	the information content of wavelet coefficients
$\otimes$	Kronecker Products
$\mathbf{I}_n$	n-dimensional identity matrix

Abbreviations

3FGL	The third *Fermi*-LAT source catalog
5G	The 5th generation mobile networks
ADF	Activation Function and Derivative Function pair of hidden layers
AGILE	Astro-Rivelatore Gamma a Immagini Leggero
AGN	Active Galactic Nuclei
AS	Attribute Selection
ASC	Attribute Set Core
AUC	Area Under the Curve
B-FlaP	Blazar Flaring Patterns
BCU	Blazar Candidate of Uncertain type
BDT	Boosted Decision Tree
BICS	Backward Internal Covariate Shift
BL Lac	BL Lacertae object
BP	Backward Propagation
CART	Classification And Regression Tree
CE	Cross Entropy
CFD	Correlation Fractal Dimension
CGRO	Compton Gamma Ray Observatory
CNN	Convolutional Neural Network
ConvL	Convolutional Layer
COMPTEL	Imaging Compton Telescope
CV	Cross Validation
CWNN	Compact Wavelet Neural Network
CWT	Continuous Wavelet Transform
DFT	Discrete Fourier Transform
DIT	Decimation In Time
DL	Deep Learning

DNN	Deep Neural Network
DSSDs	Double-Sided Strip Detectors
DT	Decision Tree
DWT	Discrete Wavelet Transform
ED	Embedding Dimension
EGRET	Energetic Gamma-Ray Experiment Telescope
EL	Ensemble Learning
EM	Ensemble Method
FcL	Fully-Connected Layer
Fermi-LAT	Large Area Telescope on-board *Fermi* Gamma-Ray Space Telescope
FDAF	Frequency-Domain Adaptive Filter
FDIDWT	proposed method based on Fractal Dimension theory and IDWT
FDMHNN	proposed model combing FDIDWT with TFMH
FFT	Fast Fourier Transform
FFTNN	proposed Fast Fourier Transform Neural Network activated by ELU
FFTMHNN	proposed Fast Fourier Transform Neural Network activated by TFMH
FP	Forward Propagation
FPR	False Positive Rate
FSRQ	Flat Spectrum Radio Quasar
FSSC	*Fermi* Science Support Centre
FT	Fourier Transform
GAN	Generative Adversarial Nets
GCN	Graph Convolutional Network
GD	Gradient Descent
GPU	Graphics Processing Unit
HoM	Hold-out Method
HEM	Hard Ensemble Method
IoT	Internet of Things
IC	Individual Contribution
ICS	Internal Covariate Shift
ID	Intrinsic Dimension
IDWT	Inverse Discrete Wavelet Transform
INTEGRAL	International Gamma-ray Astrophysics Laboratory
IWT	Inverse Wavelet Transform
kNN	*k* Nearest Neighbor

LeReLU	Leaky ReLU
LMS	Least Mean Square
LR	Logistic Regression
LWNN	Loose Wavelet Neural Network
MEGALib	Medium Energy Gamma-ray Astronomy Library
MGD	Mini-batch Gradient Descent
ML	Machine Learning
MLP	Multilayer Perceptron
MSE	Mean Square Error
NuSTAR	Nuclear Spectroscopic Telescope Array
NN	Neural Network
PoolL	Pooling Layer
PCA	Principal Component Analysis
PCC	Pearson's Correlation Coefficient
PID	Partial Intrinsic Dimension
PRC	Precision-Recall Curve
ReLU	Rectified Linear Unit
RF	Random Forest
RNN	Recurrent Neural Network
ROC	Receiver Operating Characteristic
SEM	Soft Ensemble Method
SGD	Stochastic Gradient Descent
SMO	Sequential Minimal Optimization
SNR	Signal-to-Noise Ratio
SRCC	Spearman's Rank Correlation Coefficient
SSDC	Space Science Data Center
SVD	Singular Value Decomposition
SVM	Support Vector Machine
TFMH	TFWAF and TFWDF of Mexican hat wavelet
TFWAF	Tight Frame Wavelet Activation Function
TFWDF	Tight Frame Wavelet Derivative Function
TPR	True Positive Rate
WD	Wavelet Decomposition
WNN	Wavelet Neural Network
WT	Wavelet Transform

1 Introduction

There is no doubt that scientific researches and applications in the real life are generating massive data. Various applications and the commercial values behind the so-called *big data* require us to explore how we can better benefit from the huge amount of data. Big data processing has shown great popularity in different fields. Digging the significant information out from big data collected from different applications, namely *feature extraction*, is meaningful to serve a better human life (Bahga and Madisetti, 2016).

Machine Learning (ML) plays an important role in data analysis due to its ability to learn from data and provide data-driven insights, decisions, and predictions. However, traditional ML models are inefficient to deal with the increasing amount of data. Algorithms with strong capability of feature extraction are desirable in the big data era.

In recent years, Deep Neural Network (DNN) is becoming a research focus in the ML field. The high volume of data available today brings opportunities and challenges to the development of DNN. How to cope with the massive-scale data technically with DNN models becomes a hot topic in the industry area and scientific community nowadays. While there exist many difficulties in designing and training this type of deep network structures. Extensive researches have dedicated to overcoming these difficulties.

1.1 Research Background

Every day we are dealing with data generated from many devices and instruments, such as personal computers, mobile phones, servers, and industrial systems. "We are living in the information age" is a popular saying; however, we are actually living in the data age (Han et al., 2011). Data processing is a big topic that covers many research fields. It has incorporated techniques from many domains, such as statistics, machine learning, pattern recognition, database, information retrieval, high-performance computing, and so on. Discovery of the knowledge and values behind data is the final goal of data processing.

Recently, with the development of technologies such as the Internet of Things (IoT), the 5th generation mobile networks (5G), and cloud computing, the data generated by the Internet, commerce, industries, healthcare, smart cities, and other fields is explosively growing. Usually, big data can be characterized from five aspects (Qiu et al., 2016):

1) *Volume*: big data has high volume due to many data sources;

2) *Variety*: the different sources usually generate big data characterized as heterogeneous, high-dimensional, and nonlinear;

3) *Velocity*: some time-sensitive tasks may require real-time big data processing;

4) *Veracity*: the real-world big data often contains noise and missing values;

5) *Value*: big data is often rich in valuable information.

Due to the aforementioned "5V" properties, the design of efficient algorithms for big data processing is not an easy work. Over the past decades, ML models have been widely applied to many fields, such as industry, medicine, astronomy, biology, and so on. These models provide satisfactory solutions to mining the information of data. ML is a highly interdisciplinary field built on the ideas from many different domains, such as artificial intelligence, optimization theory, information theory, statistics, cognitive science, optimal control, and many other disciplines. It has also been used for a variety of issues, including recommendation engines, identification systems, data mining, and autonomous control systems (Qiu et al., 2016).

Training ML models is to automatically learn from experience concerning some tasks and performance measures (Jordan and Mitchell, 2015). ML models enable users to uncover the underlying structure and make predictions from large datasets (Zhou et al., 2017). Neural Network (NN) is an important aspect of ML models, which usually uses multilayer structure to automatically learn hierarchical representations (Yu and Deng, 2010). It is inspired by the biological nervous system to extract features from data. While feature extraction is the key to many applications (Liu et al., 2017a). NN nonlinearly projects the input data onto a hyper-plane, so that more features hidden in data can be detected. The hyper-plane is constructed by the trainable model parameters whose values are adjusted in the NN training process.

Taking advantage of the learning capability, ML models have achieved great success in many big data applications, such as astronomical data processing. An interesting telescope spanning over the gamma-ray band is the *Fermi*-LAT (Large Area Telescope on-board *Fermi* Gamma-Ray Space Telescope, see Atwood et al. 2009), and it was launched in 2008. The third *Fermi*-LAT source catalog (3FGL) covers the four-year observation data of the *Fermi*-LAT. It contains 3034 sources, among which 1744 belong to Active Galactic Nuclei (AGN) class. Within the AGN sources, there are 573 Blazars, a subclass of AGN, haven't been tagged as BL Lacertae objects (BL Lacs) or Flat Spectrum Radio Quasars (FSRQs). These uncertain Blazars are also named as Blazar Candidate of Uncertain types (BCUs). Besides, the 3FGL remains 1010 unassociated sources (Acero et al., 2015). Therefore, these BCUs and unassociated sources result in an incomplete catalog, which decreases the significance of observations.

Parkinson et al. (2016) used several traditional ML models, including Random Forest (RF), Logistic Regression (LR), and Support Vector Machine (SVM), to identify AGNs from the unassociated sources. Based on the discriminative Blazar Flaring Patterns (B-FlaP), these predicted AGN-like sources were further classified into BL Lacs and FSRQs with a shallow Multilayer Perceptron (MLP) (Salvetti et al., 2017). This method was also applied to classify the BCUs of 3FGL to BL Lacs and FSRQs by Chiaro et al. (2016). While Lefaucheur and Pita (2017) identified BCUs from the unassociated sources of 3FGL, and then classified these BCUs together with the associated BCUs of 3FGL into BL Lacs and FSRQs with a multivariate classifier. In their work, the prediction of the multivariate classifier came from the agreement of Boosted Decision Tree (BDT) with a single hidden layer MLP model.

The previously discussed results indicate the effectiveness of traditional ML models in the classification of 3FGL sources. But these models are too simple to deal with the increasing number of sources detected by many of current and next-generation telescopes. In particular, the low-cost and fast-launch nano-scale satellites (Lucchetta et al., 2017; Racusin et al., 2017; Fiore et al., 2018; Chattopadhyay et al., 2018; Rando et al., 2019) will increase the volume of data in the few coming years dramatically. The bigger and bigger data brings great challenges to the traditional ML models. Under this circumstance, the computational efficiency and learning performance will decrease. Learning and extracting features from big data becomes a very complex problem, and more powerful and efficient algorithms are required.

While among the ML models, NN attracts many researchers coming from different fields, such as science, engineering, and applied mathematics, due to its potentiality in big data applications. Historically, there are three waves of NN research (Goodfellow et al., 2016).

The first wave appeared in the 1940-1960s, with the development of theories of biological learning (McCulloch and Pitts, 1943) and the implementation of the perceptron model (Rosenblatt, 1958). The second wave appeared during the 1980-1995 period, with the backward propagation (Rumelhart et al., 1986) used to train a NN model with one or two hidden layers. The current and third wave, i.e., Deep Learning (DL), started around 2006 (Hinton and Salakhutdinov, 2006) and became explosively popular from 2012 when Graphics Processing Unit (GPU) starts to be used for training (Krizhevsky et al., 2012), until now.

DL enables the computer to build complex concepts out of simpler concepts (Goodfellow et al., 2016). The quintessential example of a DL model is the Deep Neural Network (DNN), which typically has more than three hidden layers. Researches have indicated that in the real world, most complex problems are expected to be handled by very deep network structures (Bengio et al., 2009; He et al., 2016). DNN has become one of the fastest growing and most exciting areas of ML with big data. It is able to capture more complicated, hierarchically launched statistical patterns of inputs for achieving the adaptivity to new areas than traditional learning models. It hence often outperforms state-of-the-art achieved by hand-made features (Arel et al., 2010).

DNN is playing an important role in big data solutions to many applications which involve feature extraction. Apart from MLP, different DNNs have been proposed and successfully applied for big data processing. For instance, Convolutional Neural Network (CNN) is good at processing images (Dong et al., 2018; Jifara et al., 2019; Li et al., 2019), while Recurrent Neural Network (RNN) is widely used to deal with prediction problems (Zyner et al., 2019; Alemany et al., 2019) or speech recognition (Lokesh et al., 2019; Lam et al., 2019). Besides the data with Euclidean structure, such as images and speech, there have also some researches dedicated to more complex data with non-Euclidean structure for example chemical molecules and social networks. Recently, Graph Convolutional Network (GCN) becomes a hot topic for the complex data. It is the extension of CNNs from regular grid to irregular graph and it has shown great advantages on complex problems (Yan et al., 2018; Li et al., 2018; Chen et al., 2018).

However, regardless of the remarkable achievements of DNN in many applications, training such type of deep structure with big data is challenging. The gradient problems and the slow convergence speed are two of the most troublesome problems in DNN training. The gradient vanishing problem leads to big errors, while the gradient exploding problem causes a numerical overflow. The slow convergence of DNN training costs a lot of computing time and requires many resources, which may not be available in the real-world applications. Besides, DNN model always has many trainable parameters, which increases the computing burden. As a result, this bulky model cannot be efficiently implemented on the resource-limited embedding systems. Thus, nowadays more and more attentions have been focused on alleviating the gradient problems and accelerating DNN training (Bengio et al., 2007; Erhan et al., 2010; Dauphin and Bengio, 2013; Sainath et al., 2013; Povey et al., 2014).

1.2 Research Contents

The gradient problems refer to the abnormal change of data values during training. Specially, gradient vanishing problem refers to the exponential decrease in the norm of the gradients. In this case, the gradients will tend to zeros in a deep structure and the parameters of final layers cannot be updated, which makes the training easily stick to a local minimum. While gradient exploding problem, on the other hand, refers to the opposite behavior of gradients vanishing, where the norm of gradients dramatically increases, thus causing numerical overflow and leading to failure of training. To reduce the gradient problems and to speed up DNN training, our research concentrates on the data preprocessing and the improvements of activation function in DNN.

1.2.1 Data Preprocessing

Data collected from the real world is often described by many attributes. In other words, the samples in a dataset can be seen as laying in a high-dimensional *attribute space*. Meanwhile, the number of samples is also dramatically growing as many more sensors are used to sample or instruments are used to measure. The large attribute space and the huge number of samples lead to the so-called *dimensionality curse* (Korn et al., 2001). As a result, learning features from such big data is difficult and requires many computing resources and time.

Because of the dimensionality curse, many problems that do not exist in low-dimensional space will arise in high-dimensional space. While researches have shown that the real data is actually highly redundant due to some unimportant attributes (Bakshi and Stephanopoulos, 1993; Bengio et al., 2009; Glorot et al., 2011). The redundancy of data indicates the risk of being very sparse in high-dimensional space, which means that most samples in the dataset are likely to be far away from each other. For instance, in classification tasks, the sparsity makes predictions much less reliable than in low dimensions, since it will be based on much larger extrapolations (Géron, 2017).

Moreover, real data always contains noise, which will reduce the quality of data. If there exist too much noisy data in datasets, the key features of data will be very difficult to extract and the training performance will dramatically decrease. Therefore, if fed the raw data, the DNN algorithm may lead to slow training and not provide good results.

Indeed, data preprocessing is required and is important in many applications. Through preprocessing, the unimportant attributes and the undesirable noise of real-world data are detected and removed for low computational complexity and fast training. Moreover, in practice, it is also possible to combine the preprocessing step with the DNN structure to form a new topology with high computational efficiency.

1.2.2 Activation Functions of Deep Neural Network

Even though DNN kindled the popularity of DL in many big data applications, there are few theories to guide the design of a DNN structure. During DNN training, the activation function plays a fundamental role since it provides the nonlinear fitting capability to DNN. DNN models may suffer weak learning capability resulting from the linearity of activation functions, which makes learning slow when facing complex problems. Many numerical problems arising in DNN training can also be attributed to the inappropriately chosen activation functions. For example, saturation activation functions easily cause gradient vanishing problem, while unsaturation functions, such as the Rectified Linear Unit (ReLU) (Nair and Hinton, 2010; Le et al., 2015), may lead to gradient exploding problem if not carefully tune the learning rate or the initial values of weights.

Furthermore, Internal Covariate Shift (ICS) is considered as another reason making DNN extract features slowly (Shimodaira, 2000; Ioffe and Szegedy, 2015). It refers to the changes of the distributions of hidden layers' inputs that are also affected by activation functions. Due to ICS, the data of the next hidden layer needs to continuously adapt to the distribution of the previous hidden layer, which will be time-consuming and slows down training.

Therefore, the activation function of DNN determines the change of data within models. Optimization of activation functions can be one of the most attractive research fields for overcoming the gradient problems and to accelerating DNN training.

1.3 Research Objectives

1.3.1 Dimensionality Reduction and Feature Pre-extraction

As previously discussed, the dimensionality curse is troublesome when dealing with big data. Hence, reducing the dimensionality of original dataset before further data analysis should be a preprocessing step for low computational complexity (Kuang et al., 2015; He et al., 2018; Sellami and Farah, 2018). Moreover, the redundancy of real-world data also leaves room for the better results through dimensionality reduction and denoising (Bengio et al., 2009).

One of the widely used techniques for dimensionality reduction is Attribute Selection (AS). It picks out the subset of attributes that are most relevant to the task, and as a result, improves computational efficiency and reduces the generalization error of models (Raschka, 2015). However, if the data has many missing values or is disturbed by noise from the real world, the result of AS method will not be reliable.

Another strategy to reduce the dimensionality is to project the data into another space. In the new space, the dimensions that contribute few to the dataset are removed without losing too much information. Principal Component Analysis (PCA) is one of the most widely used projection tools (Jolliffe and Cadima, 2016). It linearly projects the original data into a new space by finding the directions where the variance is maximal, and then ignores the directions with small variances for dimensionality reduction (Calvo et al., 1998). While PCA aims at discovering linear correlations and is therefore unsuitable for nonlinear applications. Even though there exists kernel PCA (Schölkopf et al., 1997) proposed for nonlinear generalizations, its nonlinear projection will increase the computational complexity. In addition, with the PCA method, the representations of data in the new space, i.e., the components considered principal, are difficult to comprehend.

In fact, both the AS and PCA methods can be seen as projecting the original data from the original attribute space to another one. Specially, AS projects the data into a low-dimensional attribute subspace, while PCA projects the data into a new space and then removes unimportant components there. These two dimensionality reduction methods can be referred to as a kind of "hard" projection since they just remove some attributes or components directly and keep others as they are. Based on it, a "soft" projection can thus be inferred. It should not only reduce the dimensionality of dataset, but also extract the key features for the next learning process. This projection can also be seen as an operation of *feature pre-extraction* before training, which consequently remove the noise and improve learning performance (Piramuthu and Sikora, 2009; Outrata, 2010; Kaul et al., 2017; Maheshwary et al., 2017; Alexandropoulos et al., 2019).

As an example of feature pre-extraction, the correlations among attributes give further insight into the dataset (Verain et al., 2016; Bu et al., 2017). These correlations can be taken as the additional features of data. Pearson's correlation coefficient is used to measure the correlation between two attributes, but it assumes that data is normally distributed and it cannot discover the nonlinear relationship. Spearman's correlation coefficient needn't the assumption of a normal distribution, but it cannot give any information for a set of attributes. In addition, these two traditional correlation coefficient techniques cannot identify the attributes that best characterize the dataset, which is important in dimensionality reduction.

Another method to measure the correlations among attributes is the fractal dimension theory applied to estimate the potential contribution of each attribute to the dataset (Belussi and Faloutsos, 1995, 1998). This method is proposed from the fact that the independent attributes contribute more to the dataset while the correlated attributes contribute less. Thus, it not only gives more information about the attributes, but also provides a global view on the whole dataset or sub-dataset.

It is also worthy to mention that Fourier Transform (FT) has been widely used in signal process field to transform the data to be the frequency representations. This can be seen as feature pre-extraction when used in big data applications, since the new representations in frequency domain may provide more features than the original data. Even though FT doesn't reduce the dimensionality and thus remains problematic in high-dimensional case, it can be a pathfinder to study the effectiveness of feature pre-extraction.

From the above analysis, it can be seen that the preprocessed data, which is characterized as low-dimensional and feature-extracted, can be of help to decrease computational complexity and accelerate DNN training. Moreover, the correlations among attributes are expected to give more information for better learning performance.

1.3.2 Stable Training and Fast Convergence

In DNN training, slow convergence speed will require a lot of resources and computing time, while gradient problems will lead to training failure or with big errors. There have been some researches to interpret and deal with these issues from the perspective of training process (Bengio et al., 1994; Erhan et al., 2009; LeCun et al., 2012).

Decrease of the computing of DNN training is one of the most efficient strategies to accelerate training. Singular Value Decomposition (SVD) is a good way to simplify and compress the 2D convolution operations within CNN training (Zhang et al., 2015). It belongs to the so-called low-rank based methods that can also be extended to the high-dimensional tensor decomposition, such as CP decomposition (Lebedev et al., 2014), Tucker decomposition (Kim et al., 2015), Tensor train decomposition (Novikov et al., 2015), and Block term decomposition (Wang and Cheng, 2016). Wang et al. (2018b) and Ye et al. (2018) applied the idea of low-rank decomposition to RNNs and achieved satisfactory results. But the low-rank approximation is not efficient for small convolutions which are already quite fast, such as 1×1 or 3×3 convolutions widely-used in many CNNs nowadays.

Introducing sparsity into the training process will also help to train DNNs, where the pruning method represents an active research topic. Yoon and Hwang (2017) attempted to perform analysis and pruning on the weights of model. In particular, the authors imposed sparsity regularization on grouped features to prune columns of weight matrix, and then promoted competition for features between different weights to learn effective filters. Another perspective of pruning focuses on the output data of hidden layers. Liu et al. (2017b) associated a scaling factor with each channel of CNNs and imposed sparsity regularization on these scaling factors during training. Those channels with small scaling factors were pruned for slim networks and as a result, the training process was accelerated. Besides the pruning performed in Forward Propagation (FP), with meProp method (Sun et al., 2017), the gradients with small values were pruned in Backward Propagation (BP). Even though this method improved training performance, extra analysis of gradients was required.

Quantization is another direction to speed up DNN training. Recently Leng et al. (2018) proposed to train low-bit networks via ADMM (Boyd et al., 2011). Besides the quantization of weights, the output data of hidden layers can also be quantized, as proposed by Wang et al. (2018a). Moreover, in distributed deep learning, with gradient quantization (Wen et al., 2017), the cost of gradient communication between servers and workers was demonstrated dramatically reduce. On the other hand, there are also some researches dedicated to design the so-called compact networks, such as MobileNets (Howard et al., 2017; Sandler et al., 2018) and ShuffleNet (Zhang et al., 2018), for fast deep learning.

However, most of the solutions aforementioned have fine-tuning process, where some labeled samples required may not available in practice. Furthermore, these solutions usually involve a number of hyper-parameters that heavily depend on the researcher's experience.

For the gradient exploding problem, in the work Pascanu et al. (2013), authors tried to directly clip the high values of some gradients into a valid range. A regularization term was also added as a penalty on the gradients, demonstrating to be a useful solution to suppressing the exploding gradients during training. In other work Glorot and Bengio (2010) and He et al. (2015), authors proposed some strategies to better initialize the weights for a stable start of training.

While the gradient vanishing problem can be alleviated if normalizing the layer data for the purpose of keeping them flow well in the layer-to-layer transformations, which also leads to a fast convergence (Glorot and Bengio, 2010; Raiko et al., 2012). Moreover, by normalizing the output data of hidden layer to the distribution at near-zero mean, the so-called zero-mean activation functions can partly alleviate the negative effects of ICS (Maas et al., 2013; He et al., 2015). While a small ICS brings the standard gradient closer to the natural gradient and can speed up training (Clevert et al., 2015). Based on the above analysis, Batch Normalization (BN) on each training batch was proposed to accelerate the training process (Ioffe and Szegedy, 2015), but as a drawback it increased the number of operations.

In fact, for DNN the value of inputs to the other layers changes dynamically during training (Wiesler et al., 2014). From the signal processing point of view, these drastic changes are referred to the high frequency components. Therefore, normalizing the layer output data results in a low-pass filtering, so that the drastic changes can be suppressed and the ICS between hidden layers may decrease, and that there is no need to carefully tune the learning rate, meanwhile the weight initialization is not so crucial, thanks to the low-pass filtering that can ensure a stable training process. The numerical problem can also be avoided with this method.

Hence, the design of a suitable activation function provides a promising direction for fast deep learning. The objectives of the new activation function are to reduce computational complexity during training and constrain the layer data in a reasonable range, which are the keys to stable DNN training and fast convergence.

1.4 Contributions and Outline

Based on the discussions aforementioned, efforts on data preprocessing and the design of activation function can alleviate the difficulties of DNN training. Thus this thesis focuses on these two research aspects and its contributions are summarized as follows:

1) A pathfinder topology of DNN model is proposed to perform training in frequency domain. The fast Fourier transform is factorized and implemented as the matrix operations of hidden layers. The model has low computational complexity and it demonstrates the effectiveness of feature pre-extraction for fast DNN training.

2) A preprocessing method on big data is proposed based on the fractal dimension theory and inverse discrete wavelet transform. The processed dataset is characterized as low-dimensional and feature-extracted. It is demonstrated to be of help to improve the efficiency of DNN training.

3) The properties of the ideal activation function and its derivative, also called derivative function, for fast DNN training are summarized. These can be the baseline to design activation functions and derivative functions in the deep learning field.

4) Based on the wavelet frame theory, a tight frame wavelet activation function and a tight frame wavelet derivative function are proposed to speed up DNN training. The proposed functions can be the alternative activation function and derivative function on various applications.

5) A new topology of WNN model is constructed by combing the proposed preprocessing method and the tight frame wavelet functions. Systematic experiments on the astronomical data processing validate its advantages on the improvements of fast deep learning.

The structure of the thesis is illustrated in Fig. 1.1.

Chapter 2 - Literatures Review

This chapter gives the basic concepts and theories used in the thesis, including the wavelet theory and the techniques applied at different steps of data processing.

Chapter 3 - Fast Deep Learning in Frequency Domain

This chapter proposes a new topology of deep neural network by implementing the fast Fourier transform in the form of matrix operations. This model can be seen as performing feature pre-extraction, i.e., Fourier transform, at the beginning of hidden layers, while training the parameters in the latter hidden layers. It achieved better performance in the speech recognition tasks and howl noise reduction, since the frequency representations of speech signal provide more features for training.

Chapter 4 - Feature Pre-extraction of Big Data Based on Multiresolution Representations

As the data preprocessing step, a new feature pre-extraction method is described. The inverse discrete wavelet transform is performed on the arranged data obtained by estimating the information content of attributes with fractal dimension, and finally a new low-dimensional and feature-extracted dataset is generated for training. This method gives good results on a real dataset coming from the UCI machine learning repository (Dua and Graff, 2017).

Chapter 5 - Tight Frame Wavelet Functions for Fast Deep Learning

Based on the wavelet frame theory, a tight frame wavelet activation function and a tight frame wavelet derivative function are proposed for the forward and backward propagation processes, respectively. They are designed according to the summarized properties of the ideal activation function and derivative function. These proposed functions are demonstrated to be of help for fast convergence of DNN training in four experiments.

Chapter 6 - Systematic Experiments

This chapter first builds the extension experiments of the speech recognition tasks performed in chapter 3 to investigate the characteristics of the proposed tight frame wavelet activation function and tight frame wavelet derivative function through training the pathfinder DNN model in frequency domain. Then the proposed preprocessing method and the tight frame wavelet functions are combined to construct a new topology of wavelet neural network. The performance of the proposed model is evaluated on the astronomical data collected from a nano-satellite simulation environment and the real *Fermi*-LAT sources catalog (3FGL).

Chapter 7 - Experimental Results and Discussions

The results of the systematic experiments performed in chapter 6 are reported and discussed in this chapter.

Chapter 8 - Conclusions and Future Work

According to the research work with big data analysis and applications, some conclusions are summarized. Several ideas are also listed for the future work.

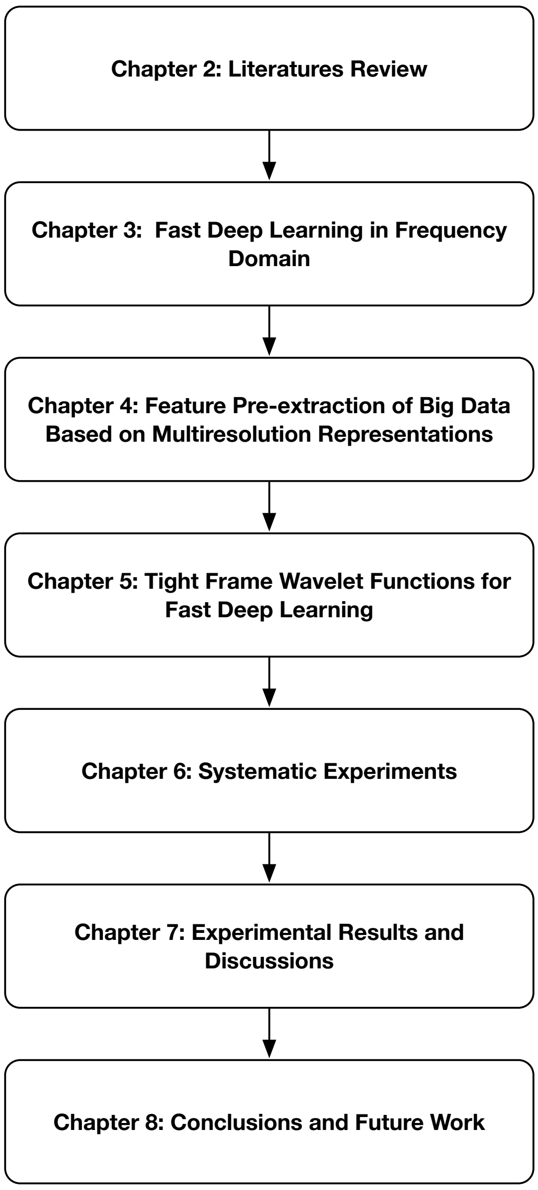

Chapter 2: Literatures Review
Chapter 3: Fast Deep Learning in Frequency Domain
Chapter 4: Feature Pre-extraction of Big Data Based on Multiresolution Representations
Chapter 5: Tight Frame Wavelet Functions for Fast Deep Learning
Chapter 6: Systematic Experiments
Chapter 7: Experimental Results and Discussions
Chapter 8: Conclusions and Future Work

Figure 1.1: Outline of the thesis.

2
Literatures Review

The procedure of data processing in Machine Learning (ML) field can be divided into three steps: data preprocessing, learning process, and performance evaluation. The preprocessing step aims to prepare the data before training. It's usually time-consuming due to the requirement of human labor (Zhou et al., 2017). At the second step, the developed ML models or methods are used to learn features from data. While their learning performances are finally evaluated at the last step based on the predefined measure.

From this perspective, the data preprocessing and learning process cost a lot of time when dealing with big data. Besides, the development of ML models or methods affects the learning performance dramatically. Therefore, our research for fast Deep Learning (DL) will concentrate on the first two steps.

As discussed before, the dimensionality reduction and feature pre-extraction are the objectives at the data preprocessing step, where the Wavelet Transform (WT) can play due to the sparsity and localization of wavelet. For wavelet analysis, the sparsity can be used to remove the redundancy of data, and the localization is of help to extract more key features from data. These properties have been widely used in many data processing applications, such as compressed sensing (Qureshi and Deriche, 2016; Huang et al., 2017), audio coding (Luo, 2010), and noise reduction (Lu et al., 2015; Cao and Luo, 2016). Moreover, WT has fast algorithm, which leads to its popularity in real applications. While the key to stabilizing and accelerating Deep Neural Network (DNN) training is to constrain the layer data in a reasonable range, where the wavelet frame may give some hints since it has the ability to restrict the energy of wavelet signal similarly. Thus, applying the wavelet theory in ML field is promising to provide better solutions for big data applications.

2.1 Data Preprocessing

Real-world datasets are highly susceptible to missing, noisy, and inconsistent data due to their huge size and heterogeneous sources (Han et al., 2011). For example, in astrophysics field, study the nature of different sources relies on the data (flux or flux density) collected by various ground observatories and space telescopes. The ground-based Cherenkov telescope (Hinton, 2009; Garrido Beltrán et al., 2011) covers the very high energies ($>$ 100GeV). AGILE (Astro-Rivelatore Gamma a Immagini Leggero, see Tavani et al. 2009) observes the lower energies which range from 20 MeV to 300 GeV. A variety of X-ray observatories cover the lower energies as well, for instance: NuSTAR (Nuclear Spectroscopic Telescope Array, see Harrison et al. 2013) in 3$\sim$79 keV, Swift (Gehrels et al., 2004) in 0.2$\sim$10 keV and 15$\sim$150keV, and INTEGRAL (International Gamma-ray Astrophysics Laboratory, see Winkler et al. 2003) in 15keV$\sim$10MeV. Therefore, each instrument observes different sources in a particular energy band and produces a lot of data, which brings a big challenge to design an efficient data analysis algorithm. While each source generates radiations detected by various instruments in different energy bands, which means that the study of a particular source needs to deal with the heterogeneous data coming from many instruments. Thus preparing such type of data with complex structure for next step is difficult.

Also, in the actual situation, the data often contains noise and missing values. Therefore, data cleaning is necessary to get a high-quality dataset featured as complete, accurate, and consistent. Moreover, how to efficiently reduce the dimensionality and extract more key features should be considered in the data preprocessing step for fast training.

2.1.1 Data Cleaning

Missing Data

Missing data is common under real conditions due to different reasons. Typically, there are four methods used to deal with the missing data:

1) The simplest solution is to discard the samples which contain missing values under some attributes. But in the case of a dataset has few samples or the new samples are not easy to collect, this method is not reasonable.

2) The missing values can be filled manually; but it is time-consuming and may not be feasible when given a large dataset with many missing values.

3) Another solution is to fill the mean value of known data under the attribute that contains missing values. While the filled-in mean value may be incorrect.

4) The filled-in values can also be determined with inference-based tools, such as Bayesian formalism (Oba et al., 2003) or nearest neighbor methods (Tutz and Ramzan, 2015), which is a popular strategy.

Noisy Data

Noise is unavoidable when collecting data from the real world. It can be seen as the outlier after cluster analysis (Gan and Ng, 2017). Noisy data contains abnormal values which will mislead the result and decrease the performance. The regularized Neural Network (NN) is robust to noise due to its small weight values (Nielsen, 2015). In this case, the noisy data becomes a minor issue. However, too much noisy data may slow down training since they will cover up the key features. Therefore, noise should be reduced for better feature extraction.

Inconsistent Data

Samples in a dataset are always described by many attributes, which results from the different data sources or different measurements. The values under the attributes may have different ranges, even different units. However, the majority of ML and optimization algorithms behave much better if attribute values are on the same scale (Raschka, 2015). Thus, attribute scaling, also known as *normalization*, is often performed on attribute values for consistency. There are four commonly used normalization methods:

1) "Standardize": standardizing attribute values by removing the mean and scaling to unit variance. The operation for a particular attribute is given by:

$$\hat{x}_i = \frac{x_i - \mu_x}{\sigma_x}, \tag{2.1}$$

where, x_i is the value of the ith sample under a particular attribute. μ_x and σ_x are the corresponding mean and standard deviation, respectively. With this transform, each attribute value has the distribution of zero-mean and one-variance. Standardization maintains useful information about outliers and makes the algorithm less sensitive to them (Raschka, 2015).

2) "Minmax": scaling the attribute values to range $[0,1]$. This method is written as:

$$\hat{x}_i = \frac{x_i - x_{min}}{x_{max} - x_{min}}, \tag{2.2}$$

where, x_{max} and x_{min} are the maximum and minimum values under a particular attribute, respectively.

3) "MaxAbs": the attribute values are scaled to the range $[-1,1]$ after divided by the maximum absolute value under a particular attribute:

$$\hat{x}_i = \frac{x_i}{max\left(|x|\right)}. \tag{2.3}$$

4) "Normalize": scaling one sample to unit norm. Different norms ($L1$-norm, $L2$-norm, and $Maximum$-norm) used to normalize the nonzero samples can be considered.

2.1.2 Dimensionality Reduction

Attribute Selection

Big datasets usually have hundreds or thousands of attributes, but most of these attributes are task-irrelevant and redundant. Ignoring some attributes randomly may lose key information, while retaining all the attributes will lead to a high computational complexity. Attribute Selection (AS) method reduces data volume by removing the "unimportant" attributes. The goal of AS is to find a minimum set of attributes such that the resulting probability distribution of the data classes is as close as possible to the original distribution obtained using all attributes (Han et al., 2011).

Decision Tree (DT) is often used to select the "important" attributes. A non-leaf node in DT denotes a test on an attribute, each branch indicates an outcome of the test, and each leaf node denotes a class prediction. Then the attributes that appear in the tree are taken as important. More times an attribute appears, more important it is assumed. The number of times an attribute appears is considered to be its *attribute importance*. While the attributes that do not appear in the tree are considered to be unimportant and can be removed. Random Forest (RF) is composed of many DTs and it reduces the bias of DTs on the estimation of attribute importance (Breiman, 2001). The attribute importance calculated with the RF model is aggregated from all the DTs. Details of DT and RF models will be discussed in section 2.2.1.

However, the AS method is sensitive to noisy data. In addition, this method doesn't consider the correlations among attributes, which may lead to redundancy among the important attributes. For instance, if two most important attributes are strongly correlated, then the second one contains similar information to the first one and it is supposed to be removed. Additionally, there is not any rule to decide the number of important attributes.

Principal Component Analysis

The idea of Principal Component Analysis (PCA) is to maximize the variance of the projected samples in an orthogonal space. The goal of dimensionality reduction is achieved by remaining the dimensions that contain most variances of the projected samples.

Assume the original dataset $\mathbf{X}$ has N samples and E attributes (dimensions). $\mathbf{X}$ has also assumed to be centralized, which means $\sum_i \boldsymbol{x}_i = \mathbf{0}$, $\boldsymbol{x}_i$ indicates the ith sample of $\mathbf{X}$. $\{\boldsymbol{w}_1, \boldsymbol{w}_2, \dots, \boldsymbol{w}_D\}$ construct the coordinate system of the new space and $D \leq E$, where $\boldsymbol{w}_i$ is an orthonormal base, i.e., $||\boldsymbol{w}_i||_2 = 1$, $\boldsymbol{w}_i^{\mathrm{T}} \boldsymbol{w}_i = 0$. Then the projection of $\boldsymbol{x}_i$ will be $\boldsymbol{z}_i = \mathbf{W}^{\mathrm{T}} \boldsymbol{x}_i$, where $\mathbf{W}^{\mathrm{T}} = (\boldsymbol{w}_1, \boldsymbol{w}_2, \dots, \boldsymbol{w}_D)$. The variance of the projected samples is $\sum_i \mathbf{W}^{\mathrm{T}} \boldsymbol{x}_i \boldsymbol{x}_i^{\mathrm{T}} \mathbf{W}$, hence the optimization will be:

$$\begin{aligned} &\max_{\mathbf{W}} \ \mathrm{tr}\left(\mathbf{W}^{\mathrm{T}}\mathbf{X}\mathbf{X}^{\mathrm{T}}\mathbf{W}\right) \\ &\text{s.t. } \mathbf{W}^{\mathrm{T}}\mathbf{W} = \mathbf{I}. \end{aligned} \tag{2.4}$$

Introducing the Lagrangian multiplier, then:

$$\mathbf{X}\mathbf{X}^{\mathrm{T}} \boldsymbol{w}_i = \lambda_i \boldsymbol{w}_i, \tag{2.5}$$

where, λ_i is the eigenvalue of the covariance matrix $\mathbf{X}\mathbf{X}^{\mathrm{T}}$; it is also the variance of the projected samples in the ith dimension. In order to reduce the dimensionality from D to D', firstly it is needed to sort these eigenvalues in descending order, e.g., $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_D$. Then the D' eigenvectors corresponding to the largest D' eigenvalues $\boldsymbol{u} = (\lambda_1, \lambda_2, \cdots, \lambda'_D)$ are considered as the principal components $\mathbf{V} = (\boldsymbol{v}_1, \boldsymbol{v}_2, \dots, \boldsymbol{v}'_D)$. The projected (transformed) dataset will be $\mathbf{Y} = \mathbf{V}^{\mathrm{T}}\mathbf{X}$.

Thus, the number of the principal components can be decided from the number of large eigenvalues. The PCA procedure for dimensionality reduction is summarized in Tab. 2.1. Note that, in practice, the eigenvalue decomposition of the covariance matrix is usually implemented as Singular Value Decomposition (SVD) for high computational efficiency.

PCA is a kind of linear transform. It gathers most information into the transformed dataset while loses some for obtaining lower dimension. But with this method, the nonlinear features of data may be lost. Similar to the AS method, deciding the number of principal components also remains problematic.

Table 2.1: Principal Component Analysis algorithm.

```
Input:  dataset $\mathbf{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}$, transformed dimension $D'$
Output: variance vector $\boldsymbol{u}$, principal components $\mathbf{V}$, transformed dataset $\mathbf{Y}$

1: centralize the samples: $\boldsymbol{x}_i \leftarrow \boldsymbol{x}_i - \frac{1}{N}\sum_{i=1}^{N}\boldsymbol{x}_i$;
2: obtain the covariance matrix $\mathbf{X}\mathbf{X}^{\mathrm{T}}$;
3: calculate the eigenvalues of $\mathbf{X}\mathbf{X}^{\mathrm{T}}$ with SVD, and sort them in descending order;
4: construct variance vector $\boldsymbol{u}$ with $D'$ largest eigenvalues;
5: take the $D'$ corresponding eigenvectors to be the principal components $\mathbf{V}$;
6: get the transformed dataset $\mathbf{Y} = \mathbf{V}^{\mathrm{T}}\mathbf{X}$;
```

2.1.3 Analysis of Attribute Correlations

Correlations among attributes can be taken as the additional features pre-extracted at the beginning of data processing and they can be measured by correlation coefficients and correlation fractal dimension. While feature pre-extraction has been demonstrated to be of help for a fast DNN training (Andrearczyk and Whelan, 2016).

Correlation Coefficients

Two commonly used correlation coefficients are Pearson's Correlation Coefficient (PCC) and Spearman's Rank Correlation Coefficient (SRCC) (Mukaka, 2012). PCC is a measure of the linear correlation between two variables $\boldsymbol{x}$ and $\boldsymbol{y}$; it is computed by

$$r = \frac{\sum_{i=1}^{N}(x_i - \mu_x)(y_i - \mu_y)}{\sqrt{\left(\sum_{i=1}^{N}(x_i - \mu_x)^2\right)\left(\sum_{i=1}^{N}(y_i - \mu_y)^2\right)}}, \tag{2.6}$$

where, x_i and y_i are the ith elements of $\boldsymbol{x}$ and $\boldsymbol{y}$ respectively, and μ_x and μ_y are the corresponding mean values. N is the number of elements for each variable.

PCC has a value between +1 and -1, where +1 is a total positive linear correlation, 0 means no linear correlation, and -1 is a total negative linear correlation. While PCC is sensitive to extreme values, which may exaggerate or dampen the strength of the relationship. Hence, it is inappropriate when the assumption of a normal distribution for the variable is not satisfied (Jiang et al., 2004).

SRCC do not need the assumption of normal distribution and it is a nonparametric measure of rank correlation between two variables. Differs from PCC which assesses linear relationships, SRCC can be used to assess monotonic relationships. The formula for calculating

SRCC is given by

$$\rho = 1 - \frac{6 \sum_{i=1}^{N} d_i^2}{N\left(N^2 - 1\right)}, \tag{2.7}$$

where, d_i is the difference between the ranks of $\boldsymbol{x}$ and $\boldsymbol{y}$.

The values of SRCC will be high when samples have a similar (or identical for a correlation of +1) rank between variables, and low when samples have a dissimilar (or fully opposed for a correlation of -1) rank between variables. SRCC is appropriate when variables are skewed or ordinal and is robust to extreme values (Mukaka, 2012). But it is only used to estimate the correlation between two variables; it cannot give the relationship among multivariate, i.e., a set of attributes in the big dataset.

Correlation Fractal Dimension

Another technique used to estimate the correlations among attributes comes from the application of fractal dimension theory. Let $\mathbb{A} = \{a_1, a_2, ..., a_E\}$ denotes the E numerical, real attribute space for a dataset, then **A** defined by $\mathbb{A}$ is a dataset in $\mathbb{R}^E$. The number of attributes E is called *Embedding Dimension* (ED) of the dataset, and $E \in \mathbb{N}$. Suppose each sample of dataset **A** is a point in E-dimensional space, then the potential existence of correlated attributes leads to suppose that the set of points describes one spatial object in a dimension which lower than or equal to E. The dimension of the object represented by the dataset is called *Intrinsic Dimension* (ID), denoted by D, $D \in \mathbb{R}^+$. It is the minimum number of attributes necessary to represent a dataset.

The projected ID, named as *Partial Intrinsic Dimension* (PID) $pD(\mathbb{C})$, can be obtained by estimating the ID of a sub-dataset $\mathbf{C} \subset \mathbf{A}$, where $\mathbf{C}$ is defined by $\mathbb{C} \subset \mathbb{A}$. Therefore, there also exist: the ID of **A** equals to $pD(\mathbb{A})$. While an attribute $a_i \in (\mathbb{A} - \mathbb{C})$ increases $pD(\mathbb{C})$ by at most its *Individual Contribution* (IC) according to the degree of the correlation between a_i and the attributes in $\mathbb{C}$. The IC of attribute $a_i \in (\mathbb{A} - \mathbb{C})$, i.e., $iC(a_i)$, is the maximum potential contribution of the attribute a_i to $pD(\mathbb{C})$. The greater the correlation between a_i and the attributes in $\mathbb{C}$, the lower its contribution to $pD(\mathbb{C})$.

Also, the IC of attribute a_i can be measured by $iC(a_i) = pD(\{a_i\})$ and it ranges in $[0, 1]$. More independent distribution of the values of a_i leads to $iC(a_i)$ closer to one, while a more structured distribution brings $iC(a_i)$ closer to zero (de Sousa et al., 2007). Thus, the E-dimensional dataset can be seen as formed by adding the attributes with different contributions to the D-dimensional sub-dataset.

The Correlation Fractal Dimension (CFD) D_2 gives a close estimation of the ID of a dataset (Belussi and Faloutsos, 1995, 1998). It is built on the assumption that the dataset analyzed is fractal and self-similar. This means that the dataset has roughly the same properties for a wide variation in scale or size, i.e., parts of any size of the fractal are similar (exactly or statistically) to the whole fractal (Traina Jr et al., 2010). Fortunately, most of the real-world datasets are fractals, thus their CFDs can be taken as their IDs (Traina Jr et al., 2000; Schroeder, 2009). As a result, CFD can be used to estimate the ID of a dataset, or the PID of a sub-dataset, or the IC of an attribute. Therefore, CFD not only gives the information of a single attribute, but also provides the estimation of the correlations among a set of attributes. It can be used to additionally extract the correlation features of data.

Given a dataset defined by $\mathbb{A}$ and is self-similar in the range of scales $[r_1, r_2]$, its CFD for this range is measured by

$$D_2 = \frac{\partial \log\left(\sum_i C_{r,i}^2\right)}{\partial \log(r)}, \tag{2.8}$$

where, $D_2 \in \mathbb{R}^+$ and r is the size of the side of the cells in a (hyper-)cubic grid divided from the E-dimensional space. It is usually selected in geometric progression, e.g., 0.5, $(0.5)^2$, ..., $(0.5)^n$, n is called *scale range*. $C_{r,i}$ is the number of points lying in the ith cell in scale r. Eqn. (2.8) is called the *box-counting* measure of D_2. If define $S(r) = \sum_i C_{r,i}^2$, then the CFD is computed as the slope of the resulting *box-count plot*, i.e., the plot of logS(r) versus log(r).

An example of calculating the CFD of a simple dataset with three attributes is shown in Fig. 2.1. The dataset contains five samples and they construct a line embedded in a three-dimensional space, thus $E = 3$. The ceiling of the ID, $\lceil D \rceil = 1$, indicates that just one attribute can sufficiently characterize the whole dataset, which is true for a line embedded in a three-dimensional attribute space.

The degree of correlations among attributes is measured by a threshold ξ: a sub-dataset $\mathbf{B} \subset \mathbf{A}$ is said to be ξ-correlated to another sub-dataset $\mathbf{C} \subset \mathbf{A}$, and attribute spaces $\mathbb{B} \cap \mathbb{C} = \emptyset$, if each attribute a_i in $\mathbb{B}$ does not contribute more than $\xi * iC(a_i)$ to $pD(\mathbb{C})$. The threshold $\xi \in [0, 1)$ tunes how strong the correlation between attributes in $\mathbb{B}$ and attributes in $\mathbb{C}$ should be in order to be detected.

Next, some concepts of attribute sets are intuitively introduced; more mathematical definitions can be found in de Sousa et al. (2007):

- *Attribute set core* ξC: a subset of attributes whose PID approaches the ID of the whole dataset. It is the union set of all correlation bases, together with the individual attributes that not ξ-correlated to any other attributes.

- *Correlation base* ξB_p: a minimal subset of attributes required to define a mapping $M_k(\xi B_p) \twoheadrightarrow a_k$, $a_k \in (\mathbb{A} - \xi C)$. The attributes in a correlation base are not ξ-correlated to each other.

- *Correlation group* ξG_p: it includes the correlation base ξB_p and every attribute ξ-correlated to all attributes in ξB_p, but excludes the attributes not ξ-correlated to the full correlation base ξB_p.

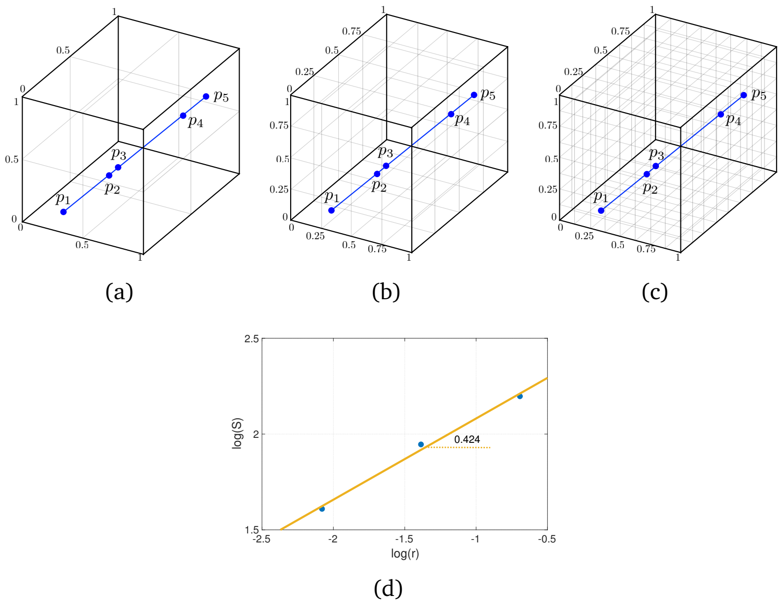


Figure 2.1: Illustration of a simple (normalized) dataset lying in different scales. (a) $r = 0.5$, $S(r) = 9$; (b) $r = (0.5)^2$, $S(r) = 7$; (c) $r = (0.5)^3$, $S(r) = 5$; (d) Box-count plot, $\lceil D \rceil = 1$.

Fig. 2.2 illustrates a simple example of a dataset with five attributes. It has two correlation groups and two corresponding correlation bases. There are not individual attributes, then the attribute set core is the union set of two correlation bases. These relationships are shown in Tab. 2.2.

A greedy algorithm, named as *FD-ASE*, is proposed by de Sousa et al. (2007) to identify the correlated attributes in a dataset. The algorithm shows a low computational complexity $O(N)$ (N is the number of samples in the dataset), which is extremely important when dealing with very large datasets.

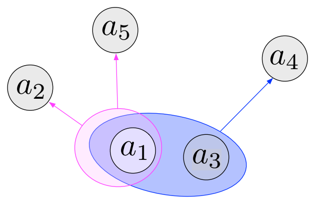


Figure 2.2: Illustration of correlation groups in a simple dataset with five attributes.

Table 2.2: Correlations in a simple dataset with five attributes.

Attributes	$\mathbb{A} = \{a_1, a_2, a_3, a_4, a_5\}$	
Mappings	$M_2(\{a_1\}) \twoheadrightarrow a_2$, $M_5(\{a_1\}) \twoheadrightarrow a_5$, $M_4(\{a_1, a_3\}) \twoheadrightarrow a_4$	
Correlation groups	$\xi G_1 = \{a_1, a_2, a_5\}$	$\xi G_2 = \{a_1, a_3, a_4\}$
Correlation bases	$\xi B_1 = \{a_1\}$	$\xi B_2 = \{a_1, a_3\}$
Attribute set core	$\xi C = \xi B_1 \cup \xi B_2 = \{a_1, a_3\}$	

2.2 Learning Process

Classification problem is common in big data fields. This section takes the classification task as a case to study the learning process. The classification problem is often handled with an inference rule learnt by extracting features from data. Based on the features extracted, samples coming from different classes are separated, which also called *Pattern Recognition*. The inference rule is usually given by the trained ML models.

ML model mainly involves two types of parameter. The first one is called *hyper-parameter* that must be defined manually before learning. But tuning the hyper-parameters is a tedious and tricky work, it largely depends on human's experience. *Grid search* is a simple solution to evaluating many possible combinations of given hyper-parameter values and taking the one that performs best as the choice. But obviously, it is time-consuming. The second type of parameter refers to the *model parameter* learnt automatically from the data; it usually has a big amount. In the learning process, the model parameters can be seen as constructing a very complex basis used to transform the original, sparse, low-level data to be the dense, high-level abstractions, namely, features.

2.2.1 Traditional Machine Learning Models or Methods

k Nearest Neighbor

k Nearest Neighbor (kNN) is a basic classification method (Cover et al., 1967). Given a dataset with known classes, the new sample is predicted as the class corresponding to the output of the majority voting system of its k nearest neighbors. Hence, there is no explicit learning process in kNN method.

Specially, kNN method used to perform classification mainly has two steps. First, based on the predefined distance measurement such as *Euclidean distance* or *Manhattan distance*, k neighbors of the new sample are collected to form a set. Second, the major class of the neighbors inside the set is predicted as the class of the new sample.

k is the hyper-parameter of kNN method. If set a small k, then a small set of known samples are used to predict the new sample. The prediction will be accurate because only the samples that similar to the new sample affect the result. However, if the neighbor samples are polluted by noise, or they have missing data but not handled appropriately, the prediction performance will decrease. So the prediction is sensitive to its neighbor samples (Hastie et al., 2009). On the contrary, if k is a big value, then the dissimilar samples are also taken as the neighbor samples, which will lead to an incorrect prediction. In this case, the model becomes simple since it ignores the discriminative information of data.

Naïve Bayes

Naïve Bayes is a classification method based on the Bayes' theorem and the conditional independence assumption. Actually, this method attempts to learn the mechanism of data generation; it hence belongs to *generative models*.

Naïve Bayes can also be summarized as two steps to classify a new sample. First, the prior probability of each known class and the conditional probability under each attribute for each known class are calculated. Second, based on the probabilities, the class that maximizes the posterior probability is predicted as the class of the new sample.

The conditional independence of Naïve Bayes method is a strong assumption that makes the method highly efficient, but it may lead to bad classification performance. Additionally, this method only works on the dataset containing nominal data, which limits its applications.

Decision Tree and Random Forest

Based on the attributes in a dataset, Decision Tree (DT) model tries to learn a series of questions to infer the class of a sample. One of the advantages of DT is that it requires very little data preparation, which means that this method doesn't need normalization. Each node in the tree counts how many samples work on one particular attribute. Specially, for each node, the impurity decrease of subsets split by each particular attribute is computed and the attribute which produces the purest subsets is taken as the criterion. It is the basic idea of Classification And Regression Tree (CART) algorithm (Breiman, 2017), which extensively searches among the pairs of the attribute for the largest impurity decrease.

Obviously, CART is a greedy algorithm, since it searches for an optimum split at the top level, and then repeats the process at each level. It does not check whether or not the split will lead to the lowest possible impurity at several levels down. So it may produce a suboptimal result. In addition, this algorithm compares all attributes for every sample at each node, which lead to a high computational complexity when dealing with large datasets. Thus, restricting the grown tree in a reasonable depth will help to speed up learning partly. The depth is an important hyper-parameter of the tree model.

Another hyper-parameter for DT models is the metric used to measure the impurity. There are two commonly used metrics to calculate the impurity, i.e., *Gini*:

$$G_i = 1 - \sum_{s=1}^{S} p_{i,s}^2, \tag{2.9}$$

and *Entropy*:

$$H_i = -\sum_{\substack{s=1 \\ p_{i,s}\neq 0}}^{S} p_{i,s} \log\left(p_{i,s}\right) \tag{2.10}$$

where, $p_{i,s}$ is the ratio of class s samples among the dataset in the ith node.

Random Forest (RF) model consists of many DTs. It was firstly proposed by Kam (1995) to reduce both the bias and variance of DT models, which is the idea of Ensemble Learning (EL) (Dietterich, 2002). RF model has become a powerful ML method for classification and prediction tasks despite its simple mechanism (Breiman, 2001; Parkinson et al., 2016).

Logistic Regression

Logistic Regression (LR) model is widely used for classification in industry. It is originally designed for binary classification problems, but it can be extended to deal with multi-class classification problems. The parameters of LR model are usually estimated by maximizing the likelihood function of the conditional probability distribution. In practice, the maximization is transformed to be the minimization of a cost function, which often implemented with *Gradient Descent* (GD) method.

GD method is an iterative process described in Tab. 2.3. In each step, the partial differentials of the cost function $C(\boldsymbol{w})$ are calculated on the parameters $\boldsymbol{w}$ and they are denoted as gradients $\boldsymbol{g}$. Then $\boldsymbol{g}$ are used to update $\boldsymbol{w}$ with a learning rate η. The process is repeated until meeting some conditions. When the cost function is convex, the output of the GD method is a global optimal solution. While in many cases, the global optimal solution cannot be guaranteed.

Support Vector Machines

Another powerful and widely used learning algorithm is Support Vector Machines (SVM) that formally published in Cortes and Vapnik (1995). The training strategy of SVM is to maximize the margin, and it can be mathematically transformed to solve a convex quadratic programming problem. The margin is denoted as the distance between the separating hyperplane (decision boundary) and the samples that are closest to this hyperplane. These samples are the so-called *support vectors*.

Table 2.3: Gradient Descent method.

```
Input:  cost function $C(\boldsymbol{w})$, gradient function $g(\boldsymbol{w}) = \nabla C(\boldsymbol{w})$
        learning rate $\eta$, precision $\varepsilon$, iterations $K$
Output: minimal point $\hat{\boldsymbol{w}}$
1:  initialize $\boldsymbol{w}^{(0)}$, set $k = 0$;
2:  calculate gradient $\boldsymbol{g}_k = g(\boldsymbol{w}^{(k)})$,
    if $\left\|\boldsymbol{g}_k\right\| < \varepsilon$: stop, output $\hat{\boldsymbol{w}} = \boldsymbol{w}^{(k)}$;
3:  set $\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} - \eta \boldsymbol{g}_k$, compute $C(\boldsymbol{w}^{(k+1)})$;
    if $\left\|C(\boldsymbol{w}^{(k+1)}) - C(\boldsymbol{w}^{(k)})\right\| < \varepsilon$ or $\left\|\boldsymbol{w}^{(k+1)} - \boldsymbol{w}^{(k)}\right\| < \varepsilon$: stop, output $\hat{\boldsymbol{w}} = \boldsymbol{w}^{(k+1)}$;
4:  set $k = k + 1$,
    if $k = K$: stop, output $\hat{\boldsymbol{w}} = \boldsymbol{w}^{(k)}$;
    else: turn to step 2.
```

Considering a binary classification problem on a linearly separable dataset, as shown in Fig. 2.3(a). The training objective is to find a separating hyperplane in the attribute space to separate the samples into "positive" (+1) and "negative" (-1), and then to maximize the margin, which is called *hard margin maximization*. However, this solution cannot be directly used on the approximate-linearly separable dataset that contains outliers, as shown in Fig. 2.3(b). Under this circumstance, a penalty parameter that works on the misclassified results is usually introduced to balance the predicted error and computational complexity. Hence, the convex quadratic programming problem in the approximate-linearly separable case is called *soft margin maximization*, and its solution can be obtained with the similar optimization procedure of the hard margin maximization.

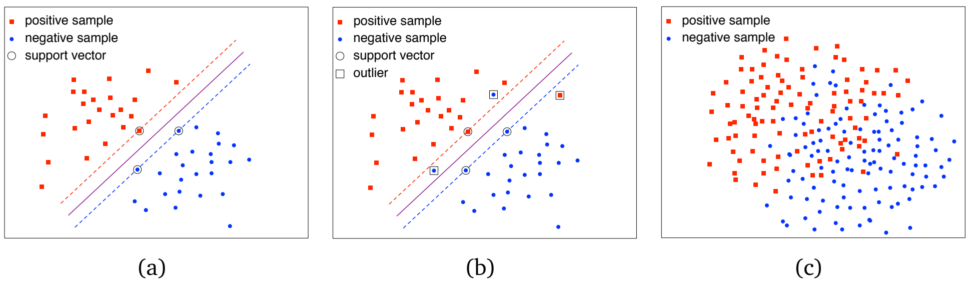


Figure 2.3: Binary classification problems for the (a) linearly separable dataset; (b) approximate-linearly separable dataset; (c) linearly inseparable dataset.

Nevertheless, in real-world scenarios, datasets are often more complex and cannot be classified by the linear SVM aforementioned, as shown in Fig. 2.3(c). In such case, the nonlinear classification problem can be handled with the nonlinear SVM. The idea is that the original dataset is nonlinearly transformed into a higher space with a *kernel function*. The linearly inseparable features in the original space will become linearly separable in the new space. Then the classification becomes a linearly separable problem in the high space, which can be solved with the linear SVM. More details about the nonlinear SVM can be found in Scholkopf and Smola (2001) and Hofmann et al. (2008).

Sequential Minimal Optimization (SMO) (Platt, 1998) is popularly used to train SVM. It has been efficiently implemented in many packages, such as SVM Light (Joachims, 1999) and LIBSVM (Chang and Lin, 2011). Additionally, the binary classification SVM can also be extended to address multi-classification problems (Weston et al., 1999; Crammer and Singer, 2001).

Ensemble Methods

Ensemble Method (EM), proposed by Freund and Schapire (1995), aggregates independent learning models into an ensemble predictor and leads to better performance. This strategy reduces both the bias and the variance of individual models, which always achieves good results on unseen samples. EM usually works best when the individual models are as independent as possible.

According to the method used to aggregate models, EM is classified into Hard EM (HEM) and Soft EM (SEM). HEM counts the prediction of each individual model and outputs the class that gets the most votes (Zhang et al., 2014). It is similar to the majority voting system. The drawback of HEM is that it cannot estimate the probability of predicted class. In other words, it cannot tell how certain the class of the predicted result is.

While SEM predicts a sample based on the weighted probability of the predictions coming from individual models (Jiménez, 1998). The weights indicate the contributions of models, and they are decided by the performance of individual models. Therefore, SEM often achieves better performance than HEM, since SEM gives more weight to highly confident votes (Géron, 2017), i.e., the model that performs better will hold higher weight, while in HEM, all the models have the same weight. Obviously, the soft ensemble strategy requires every individual model to have the ability to estimate the probability of prediction.

According to the relationship of individual models, EM can also be cataloged into two types. The first one is a sequence method, where there is a strong dependency between successive models and these models must be constructed serially. A famous example of sequence EM is AdaBoost (Freund and Schapire, 1997). While another type of EM is parallel, where the individual models can be implemented simultaneously because they don't have strong dependencies. Bagging (Breiman, 1996) and RF are the common examples.

2.2.2 Neural Network Models

Actually, the early NN can be cataloged into the traditional ML models due to its shallow structure and straightforward mechanism (Rosenblatt, 1958; Novikoff, 1963; Widrow and Lehr, 1990). However, in recent years, the structures of the NN model become deeper and deeper, and they show an enormous potentiality in many complex problems. It is of value to particularly introduce and analyze the characteristics of different NN structures for improvements.

Multilayer Perceptron

Multilayer Perceptron (MLP) is a basic structure of NN. Each layer of the MLP model is fully connected by the trainable parameters called *weights* and *bias*. These layers are also known as Fully-Connected Layers (FcLs). One simple example of MLP is illustrated in Fig. 2.4. In the figure, $\boldsymbol{a}$ contains the outputs of one hidden layer and they are called *activations*. Specifically, the activations of a hidden layer constitute a vector whose length equals the number of hidden units. For example, vector $\boldsymbol{a}^l$ consists of m elements corresponding to m units of the lth hidden layer. $\boldsymbol{z}$ is called *logits* which is the summation of the bias b with the inner product between the activations $\boldsymbol{a}$ of the previous layer and the weights $\mathbf{W}$ of the local layer. $\mathbf{W}$ is a matrix with a shape of (m,n), i.e., $\mathbf{W} \in \mathbb{R}^{m \times n}$. m and n are the number of hidden units of the previous layer and local layer, respectively. $f(\cdot)$ is the activation function of hidden layers.

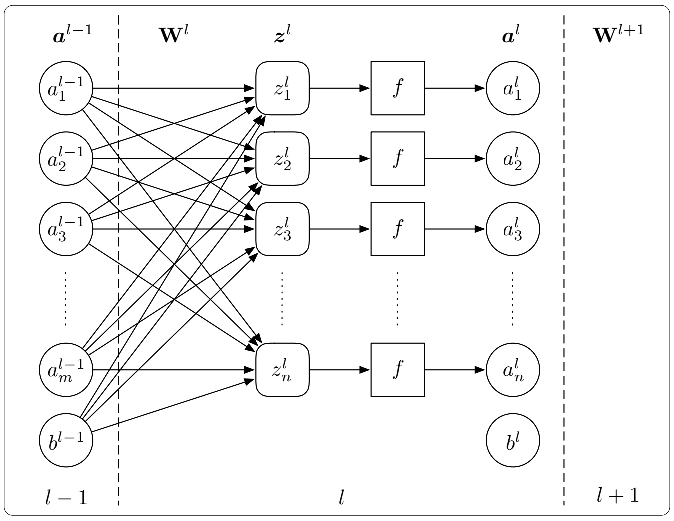


Figure 2.4: The structure of a MLP model. The superscript and subscript are the indices of layers and hidden units, respectively.

One-time MLP training process includes Forward Propagation (FP) and Backward Propagation (BP). According to the symbols shown in Fig. 2.4, the FP process of the lth hidden layer can be described in the matrix form:

$$\begin{cases} \boldsymbol{z}^l = \boldsymbol{a}^{l-1}\mathbf{W}^l + b^{l-1} \\ \boldsymbol{a}^l = f\left(\boldsymbol{z}^l\right) \end{cases}. \tag{2.11}$$

The purpose of FP process is to calculate the *error* (also called *loss*) between the model outputs and the ground-truth, based on a predefined loss function. Cross Entropy (CE) usually acts as the loss function in classification tasks, while Mean Square Error (MSE) function is often used in regression problems. These two loss functions are written as:

$$\begin{cases} C_{\mathrm{CE}} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{s=1}^{\mathrm{S}} I\left(y_i = s\right)\log a_{i,s} & \text{(2.12a)} \\ a_{i,s} = \frac{\exp\left(z_{i,s}\right)}{\sum_{k=1}^{\mathrm{S}}\exp\left(z_{i,k}\right)} & \text{(2.12b)} \end{cases}$$

$$C_{\mathrm{MSE}} = \frac{1}{2N}\sum_{i=1}^{N}\left(\boldsymbol{z}_i - \boldsymbol{r}_i\right)^2, \tag{2.13}$$

In Eqn. (2.12a), N is the number of samples and S is the number of classes. $I(\cdot)$ is the indication function, i.e., $I = 1$ if the target class for the ith sample equals to s, otherwise, $I = 0$. Eqn. (2.12b) is called softmax function, where $z_{i,s}$ and $a_{i,s}$ are the logit and activation of the sth output unit for the ith sample, respectively. When using CE function as the loss function in classification tasks, the logits of the output layer are better to be activated by softmax function and the one-hot labels should be used at the same time. Glorot and Bengio (2010) has found that the plateaus in the training criterion will barely present in classification problems when using this kind of log-likelihood loss function. In Eqn. (2.13), $\boldsymbol{z}_i$ and $\boldsymbol{r}_i$ are the logits of the output layer and the ground-truth for the ith sample, respectively. In addition to CE and MSE functions, some other non-classical loss functions can also be considered, as listed in Janocha and Czarnecki (2016).

The BP algorithm was proposed by Rumelhart et al. (1986) to update the weights and bias. Their idea was to construct internal representations with gradient descent procedure. There is an opposite computing direction in comparison with the FP process: in FP process the activation data flows from the input to the output layer, while in BP process, the error data derived from the loss function propagates from the output to the input layer. Then the errors of each hidden layer are calculated successively and the gradients of trainable parameters are computed with the respective errors. Finally, the parameters are updated with different gradient-based *optimizers*, such as Stochastic Gradient Descent (SGD) (Koushik and Hayashi, 2016), Mini-batch Gradient Descent (MGD) (Li et al., 2014), AdaGrad (Duchi et al., 2011), and Adaptive Moment Estimation (Adam) (Kingma and Ba, 2014). The BP algorithm can be summarized as:

$$\begin{cases} \boldsymbol{\delta}^L = \nabla_{\boldsymbol{a}} C \odot \sigma'\left(\boldsymbol{z}^L\right) & (2.14a) \\ \boldsymbol{\delta}^l = \left(\left(\mathbf{W}^{l+1}\right)^{\mathrm{T}} \boldsymbol{\delta}^{l+1}\right) \odot f'\left(\boldsymbol{z}^l\right) & (2.14b) \\ \mathbf{W}^l \leftarrow \mathbf{W}^l - \eta \left(\boldsymbol{a}^{l-1}\right)^{\mathrm{T}} \boldsymbol{\delta}^l & (2.14c) \\ b^l \leftarrow b^l - \eta \sum \boldsymbol{\delta}^l & (2.14d) \end{cases}$$

where, C is the error calculated with the loss function, and $\nabla_{\boldsymbol{a}} C$ are the partial differentials of the predicted error on the output $\boldsymbol{a}$. $\boldsymbol{\delta}^l$ contains the errors of the lth layer, $l = 1, 2, ..., L-1, L$, where L indicates the output layer. $\sigma'(\cdot)$ is the derivative of activation function of the output layer. As it often occurs in nonlinear fitting tasks, $\sigma(\boldsymbol{z}^L) = \boldsymbol{z}^L$, i.e., there is no nonlinear mapping in the output layer. But in classification tasks, softmax function typically acts as the $\sigma(\cdot)$ function. $f'(\cdot)$ is the derivative function of the hidden layers. The Activation Function and Derivative Function pair (ADF) of hidden layers are often discussed simultaneously in this thesis. Eqn. (2.14c) and (2.14d) are used to update the weights and bias of the lth hidden layer, respectively. η is called *learning rate* that determines the learning step in training.

Convolutional Neural Network

Besides FcLs, there are other types of layers in DNN structure. For example, the Convolutional Layer (ConvL) and Pooling Layer (PoolL) in Convolutional Neural Network (CNN) (LeCun et al., 1998) which is a specialized kind of NN for processing data that has a known grid-like topology (Goodfellow et al., 2016), such as images.

ConvL was inspired by neuroscience that shows many neurons in the visual cortex system have a small local *receptive field* (Hubel, 1959; Hubel and Wiesel, 1959, 1968). The abstracts of higher-level neurons are based on the outputs of neighboring lower-level neurons (Fukushima, 1980), which leads to the multilayer structure of CNN. The receptive field is often implemented as the convolutional window in CNN, and it is also named as *kernel*. The values of the kernel are the trainable parameters of CNN and they can be updated with BP algorithm as well, where ADF is required. The convolution operation is illustrated in Fig. 2.5.

ConvL involves the convolution operation[1] which differs from the general matrix multi-

[1] In practice, many machine learning libraries implement "cross-correlation" instead of the "convolution" operation in CNN. But they achieve the same performance during training. More discussions about them can be found in section 9.1 of the deep learning book Goodfellow et al. (2016).

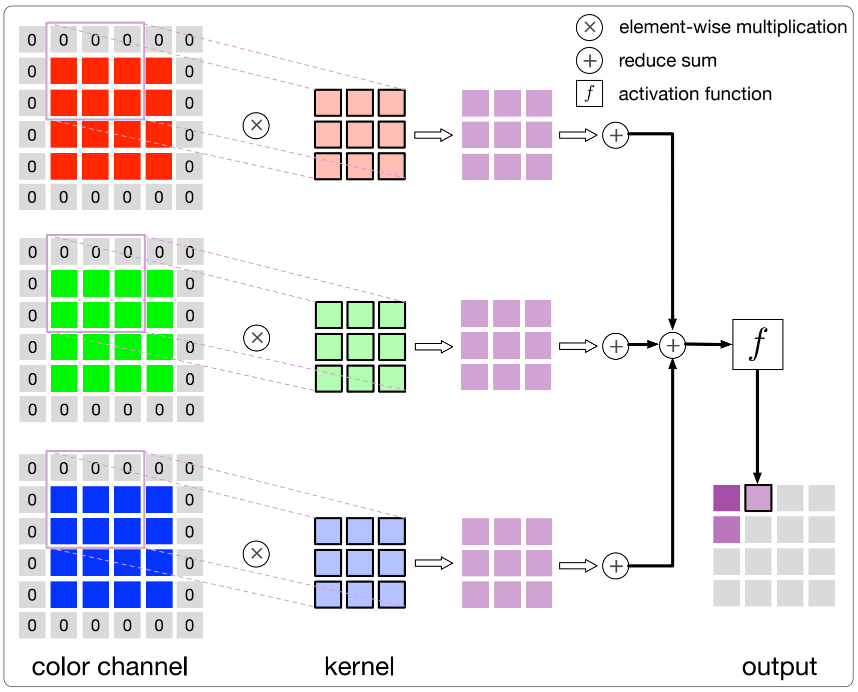


Figure 2.5: Convolution for a RGB color image.

plication in FcL. The convolutional windows result in the *sparse connectivity* in CNN, which will reduce the memory requirements of the model and thus improve the computational efficiency. The sparsity will also enhance the robustness of model to noise input. In addition, the parameters of the kernel are shared by every position of the input, which speeds up the training further.

The pooling operation of PoolL is used to replace the output of the layer at a certain location with a summary of the nearby outputs. The target of it is to transform the joint feature representations into a new, more usable one that preserves important information while discard or weaken irrelevant details (Boureau et al., 2010; Sermanet et al., 2012). Given a rectangular window in an image, after pooling the output of a window is typically either the maximum value (*max pooling*) or the average value (*average pooling*) of the rectangular neighborhood. In fact, from the signal processing point of view, the PoolL of CNN can be seen as a *downsampling* operation. Therefore, it doesn't have trainable parameters and needs no ADF.

But the simple max pooling or average pooling may not respect the structure of the previous layer, which leads some structures in the features to be absent or lost (Chaabane et al., 2017). Several improvements have been proposed to deal with the disadvantages of the traditional pooling operations (Zeiler and Fergus, 2013; Yu et al., 2014; Williams and Li, 2018).

Generative Adversarial Nets

Another promising NN model, Generative Adversarial Nets (GAN) firstly proposed by Goodfellow et al. (2014), can also be constructed by FcLs. It includes a *discriminator* and a *generator*, between which an adversarial game takes place: the discriminator is trained to tell the samples coming from the generator, while the generator is trained to produce samples which cannot be recognized by the discriminator. The discriminator and generator are trying to find Nash equilibria in high-dimensional, continuous, non-convex games.

GAN model can also be trained with iterative FP and BP processes. It has been quite popular and successful to generate intuitive images, which benefits from the perfect trained generator (Denton et al., 2015; Sønderby et al., 2017). In recent years, researchers are focusing on the combination of semi-supervised learning with the discriminator to build a semi-supervised classifier. This idea has been demonstrated successful in image classification and it will be an attractive field. For example, in Odena et al. (2017) and Salimans et al. (2016), the authors labeled the samples produced from the generator as one class, namely the fake class, while the samples coming from the dataset were labeled as true. As the generator was trained to approximate the true samples, the discriminator was trained to identify the fake samples. Moreover, because the true samples had distinct known classes as well, the discriminator was additionally trained to predict the known classes. Their work obtained good results on some public image datasets.

2.3 Performance Evaluation

Following the learning phase, there should be some methods to evaluate the performance of the trained models on unseen samples. In particular, monitoring the loss values is significant to observe the status of training process.

2.3.1 Split Dataset

Usually, a dataset is divided into two parts: training dataset for learning, and test dataset for evaluation. The samples in test dataset are considered to be unseen samples, thus they shouldn't simultaneously appear in the training dataset or ever used for training. For a dataset $\mathbf{X} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_N, y_N)\}$, there are two commonly used split methods to obtain the training dataset $\mathbf{S}$ and test dataset $\mathbf{T}$: *Hold-out Method* (HoM) and *Cross Validation* (CV).

Hold-out Method

With the HoM, two mutually-exclusive sub-datasets are generated, i.e., $\mathbf{X} = \mathbf{S} \cup \mathbf{T}$, $\mathbf{S} \cap \mathbf{T} = \emptyset$. The number of samples in training dataset and test dataset usually follows a ratio, for which, 7:3 is a common choice. Note that the split of training and test datasets should ensure the consistency of data distribution as far as possible so that not introduce the additional bias. For example, if dataset $\mathbf{X}$ has 500 positive samples and 500 negative samples, and the number of samples in the sub-datasets follows the ratio 7:3, then the training dataset should contain 350 positive samples and 350 negative samples, while the test dataset contains 150 positive samples and 150 negative samples.

However, how to select the samples into the sub-datasets remains problematic. For instance, in the aforementioned example, there exist many solutions to deciding which 350 positive samples should be included in the training dataset. Therefore, the evaluation result coming from a single partition of a dataset is usually not reliable. A commonly used method is to split the dataset several times and return the average result.

Cross Validation

CV method divides the dataset $\mathbf{X}$ into k mutually-exclusive sub-datasets, i.e., $\mathbf{X} = \mathbf{X}_1 \cup \mathbf{X}_2 \cup \cdots \cup \mathbf{X}_k$, and $\mathbf{X}_i \cap \mathbf{X}_j = \emptyset$, $(i \neq j)$. In each process, the union of $k-1$ sub-datasets is taken as the training dataset by turn, and the remaining one is the test dataset. Then k results are averaged to be the result. The CV method is hence also called *k-fold cross validation*, where k is usually chosen as 10.

CV method should also respect the consistency of data distribution. Besides, the k sub-datasets can be generated with different partitioning strategies as well. Therefore, similar to HoM, the training and evaluation processes of CV method should be repeated on different partitioning datasets several times, and the average result is returned.

2.3.2 Intuitive Performance

It is expected that the ML models are able to learn the general patterns in training dataset and have the ability to identify unseen samples in test dataset correctly. However, if a model is trained too complex, then it tends not to generalize well to unseen data, which is called *overfitting* (high variance). On the contrary, if a trained model is not complex enough to capture the key patterns of data, it will lead to bad performance on unseen data, this is called *underfitting* (high bias).

Underfitting indicates a weak learning capability of the model. It can usually be tackled by designing a more complex model, such as, adding the depth of DT, increasing the number of DTs in RF, and adding the hidden layers of NN models. However, overfitting is troublesome and is a common issue in ML field. It indicates a weak *generalization capability* of the model.

The overfitting problem can be alleviated in two folds. On one side, training on a larger dataset will produce a model with stronger generalization capability, which is also one of the reasons leading to the success of DNN nowadays. But collecting more samples is always not feasible in practice. Thus, data augmentation is a common choice, such as, adding noise to the samples (Salimans et al., 2016; Sønderby et al., 2017) or distorting the images as new samples (Wong et al., 2016; Perez and Wang, 2017).

On the other side, decreasing the complexity of models is another solution to the overfitting problem. The improvements for the models include pruning branches in DT, and randomly discarding some neurons during the NN training process (Srivastava et al., 2014). Furthermore, the weights sharing mechanism of CNN inherently introduces sparsity and gives the ability to alleviate overfitting. Therefore, designing a sparse activation function to keep the sparsity of propagating data during training may be a promising direction to reduce overfitting problem of DNN.

Other solutions may come from the optimization of NN training process, such as, adding a regularization term to the loss function (Goodfellow et al., 2016; Kukačka et al., 2017) or normalizing the data of hidden layers (Ioffe and Szegedy, 2015). Ensemble method is also a useful strategy to strengthen the generalization capability of models.

2.3.3 Performance Measure

The mathematical evaluation of the model performance can be given by comparing the model outputs with the ground-truth. A simple method is to use the measurement function to calculate the distance between the model outputs and the ground-truth, such as using the MSE function. While in classification tasks, there exist better measurements for evaluation: *accuracy*, *precision*, and *recall*. For a binary classification task, the three measurements can be defined by the form of a *confusion matrix* that consists of four predicted cases:

- true positive (tp): the number of positive samples correctly predicted to be positive;
- false positive (fp): the number of negative samples wrongly predicted to be positive;
- false negative (fn): the number of positive samples wrongly predicted to be negative;
- true negative (tn): the number of negative samples correctly predicted to be negative.

The confusion matrix is described in Tab. 2.4, and the aforementioned three measurements can be written as:

$$\begin{aligned} \text{accuracy} &= \frac{\text{tp}+\text{tn}}{\text{tp}+\text{fp}+\text{tn}+\text{fn}}, \\ \text{precision} &= \frac{\text{tp}}{\text{tp}+\text{fp}}, \\ \text{recall} &= \frac{\text{tp}}{\text{tp}+\text{fn}}. \end{aligned} \tag{2.15}$$

Table 2.4: Confusion matrix of the binary classification problem.

Labels		Predicted positive	Predicted negative
Truth	positive	tp	fn
	negative	fp	tn

Inspecting the results with accuracy score is common in many classification tasks, while precision or recall score is also suitable under the circumstance where the positive samples are more concerned. Precision indicates the ability of the classifier not to label as positive a sample that is negative, and recall shows the ability of the classifier to find all the positive samples.

In order to get a high precision score, for example, the case of the positive samples misclassified to be negative can be acceptable, while the negative samples are not desirable to be misclassified as positive, as shown in Fig. 2.6. The case will be opposite if a high recall score is expected.

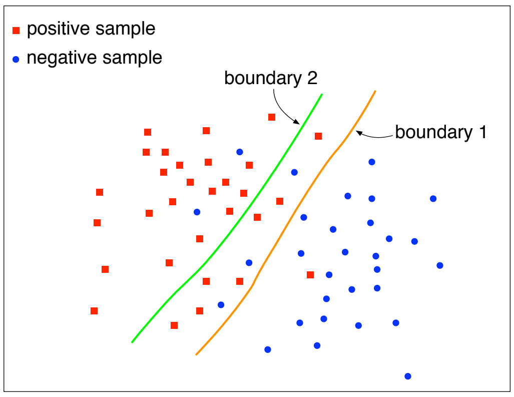


Figure 2.6: Different boundaries in a binary classification problem. In the case of boundary 1, the precision score is lower while the recall score is higher, than in the case of boundary 2.

2.3.4 Performance Curve

To evaluate the trained models on the test dataset, the three measurements introduced before can be considered. However, in most cases, high precision score and high recall score cannot be achieved at the same time. There should be some tools to perform precision/recall tradeoff. In ML field, *Precision-Recall Curve* (PRC) and *Receiver Operating Characteristic* (ROC) are commonly used.

In binary classification tasks, the trained models usually output real values or probabilities for the test samples. These outputs are compared with a threshold, where the sample with an output higher than the threshold is considered as positive, otherwise as negative. Therefore, as the threshold varies from a high value to a low value, the precision score is decreasing while the recall score is increasing. Given different thresholds, a PRC can be plotted as shown in Fig. 2.7(a). A perfect classifier would fall into the top-right corner of the graph with both precision and recall equal to 1. In particular, for accuracy measurement, the threshold is fixed to 0.5.

However, based on different thresholds, ROC uses another two values to plot the curve, i.e., *True Positive Rate* (TPR, another name for recall) and *False Positive Rate* (FPR), as shown in Fig. 2.7(b). A perfect classifier would fall into the top-left corner of the graph with both a TPR of 1 and a FPR of 0. FPR is defined as:

$$\mathrm{FPR} = \frac{\mathrm{fp}}{\mathrm{tn} + \mathrm{fp}}. \tag{2.16}$$

Based on the ROC curve, the so-called *Area Under the Curve* (AUC) is often used to further characterize the performance of a classifier. The classifier with larger AUC performs better.

When comparing PRC with ROC, as a rule of thumb, the PRC is preferable whenever the positive class is rare or when the false positives are more concerned than the false negatives, and the ROC otherwise (Géron, 2017).

In particular, when training NN models, monitoring the training and test loss values is of great help to study the status of training. The convergence can also be observed if the loss values are plotted during training, as illustrated in Fig. 2.7(c).

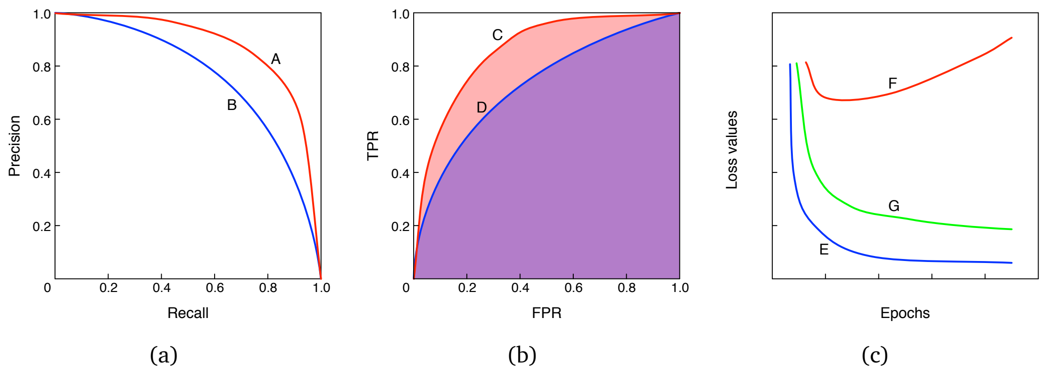


Figure 2.7: The intuitive curve examples of (a) PRC, where model A performs better than model B; (b) ROC, where model C performs better than model D; (c) loss values, where model E converges normally, model F diverges, while model G converges to a suboptimal level.

2.4 Wavelet Theory

As discussed before, the key to many big data applications is feature extraction. Another powerful tool to extract features from data is wavelet analysis. Having the property of being localized in time as well as in frequency domain, wavelet analysis has been widely used in the last decades. Many mathematical conclusions in wavelet theory have been validated useful in many applications, especially in speech and image data processing (Tanmoy et al., 2018; Wang et al., 2016; Luo et al., 2006; Vyas and Paik, 2018).

In big data era, the real data is expected to be transformed into a low-dimensional space where the key features are also extracted. The possibility of dimensionality reduction shows the sparsity of real data, which has been demonstrated to be of help to save storage (Langr and Tvrdik, 2015; Bylina et al., 2018), improve computational efficiency (Dalton et al., 2015; Elafrou et al., 2018; Yang et al., 2018), and remove noise (Zhuang and Bioucas-Dias, 2018). Meanwhile, Donoho (1993) pointed out that wavelets are unconditional bases of various function spaces, indicating that they typically compress the energy of signal into a smaller number of coefficients. It is the sparsity and energy compaction of wavelet. Therefore, wavelet analysis may provide solutions to the problems arisen in DNN training.

2.4.1 Wavelet Transform

Fourier Transform (FT) projects the one-dimensional signal in the time domain to the one-dimensional representations in the frequency domain, i.e., transforming a pure "time description" into a pure "frequency description" (Strang and Nguyen, 1996). While the two-dimensional coefficients, namely the time and frequency representations, can be obtained after Wavelet Transform (WT) (Sidney Burrus et al., 1998). Besides, the original signal can be reconstructed from the wavelet coefficients with Inverse Wavelet Transform (IWT).

In practice, WT is usually performed as a Discrete Wavelet Transform (DWT) on computers for high efficiency, which can be implemented with filtering operations. The filters are designed based on the standpoint of multiresolution: the difference of information between the approximation of a signal at the resolutions 2^{m+1} and 2^m (where m is an integer) can be extracted by decomposing this signal on a wavelet orthonormal basis of $L^2(\mathbb{R}^n)$ (Mallat, 1989). The pyramidal structure of the wavelet filter bank makes it possible that the information at a low resolution can be inferred from the information at a high resolution.

Function Space

A *function space* is a linear vector space where the vectors are functions; the scalars are real numbers (sometimes complex numbers). If function $f(t) \in L^2(\mathbb{R})$ and $g(t) \in L^2(\mathbb{R})$, where $L^2(\mathbb{R})$ is denoted as the one-dimensional measurable, square-integrable function space, then the *inner product* of $f(t)$ with $g(t)$ is a scalar a obtained by an integral:

$$a = \langle f(t), g(t) \rangle = \int f(t)g(t)dt. \tag{2.17}$$

The *norm* of $f(t)$ in $L^2(\mathbb{R})$ is given by:

$$||f||^2 = \int |f(t)|^2 dt \tag{2.18}$$

The *convolution* of two functions $f(t)$ and $g(t)$ is written as:

$$f(t) * g(t) = \int f(u)g(t-u)du. \tag{2.19}$$

The *Fourier transform* of $f(t)$ is written as $\hat{f}(\omega)$ defined by:

$$\hat{f}(w) = \int e^{-j\omega t} f(t)dt. \tag{2.20}$$

Multiresolution Analysis

A set of scaling functions refers to the integer translates of the basic *scaling function* $\phi(t)$:

$$\phi_k(t) = \phi(t-k),\ k \in \mathbb{Z},\ \phi \in L^2. \tag{2.21}$$

A subspace of $L^2(\mathbb{R})$, i.e., $V_0 \subset L^2(\mathbb{R})$, can be assumed to be spanned by these functions. Moreover, the size of the subspace will change with different time scales of $\phi(t)$, which means:

$$\phi_m(t) = 2^m \phi(2^m t), \ \forall m \in \mathbb{Z} \tag{2.22}$$

whose span is $V_m \subset L^2(\mathbb{R})$ and the *resolution* is 2^m. According to the natural scaling condition:

$$f(t) \in V_m \iff f(2t) \in V_{m+1}, \ \forall m \in \mathbb{Z}, \tag{2.23}$$

a nesting of the spanned spaces exists:

$$0 \cdots \subset V_{-2} \subset V_{-1} \subset V_0 \subset V_1 \subset V_2 \cdots \subset L^2, \tag{2.24}$$

or

$$V_m \subset V_{m+1}, \ \forall m \in \mathbb{Z}. \tag{2.25}$$

Thus, based on Eqn. (2.23) and (2.24), there exist:

$$\phi(t) = \sum_n h(n)\sqrt{2}\phi(2t-n), \ n \in \mathbb{Z}, \tag{2.26}$$

where, $h(n)$ is called the coefficient sequence of scaling function.

Therefore, a two-dimensional family of functions can be generated from $\phi(t)$ by

$$\begin{aligned} \phi_{m,k}(t) &= 2^{-m/2}\phi_m(t-2^{-m}k) \\ &= 2^{m/2}\phi(2^m t - k), \ k \in \mathbb{Z} \end{aligned} \tag{2.27}$$

whose span over k is $V_m \subset L^2(\mathbb{R})$ for all integers $k \in \mathbb{Z}$. Eqn. (2.27) shows that an orthonormal basis[2] $\phi_{m,k}$(t) of any space V_m can be built, by dilating a function $\phi(t)$ with a coefficient 2^m and then translating the resulting function on a grid whose interval is proportional to 2^{-m}.

However, only the subspaces V_m spanned by the orthonormal basis $\phi_{m,k}$(t) are not sufficient to represent a function. The *detail information* at the resolution 2^m is needed. This information is given by the orthogonal projection of the original signal in the *orthogonal complement* of V_m in V_{m+1}. Let W_m be this orthogonal complement, i.e.,

$$\begin{aligned} &W_m \text{ is orthogonal to } V_m, \\ &W_m \oplus V_m = V_{m+1}, \end{aligned} \tag{2.28}$$

[2]The concept of orthonormal basis will be introduced in section 2.4.2.

then

$$L^2 = V_0 \oplus W_0 \oplus W_1 \oplus \cdots \tag{2.29}$$

The orthonormal basis of W_m is built by dilating and translating an *orthogonal wavelet* $\psi(t)$:

$$\psi_{m,k}(t) = 2^{m/2}\psi(2^m t - k), \ k \in \mathbb{Z}. \tag{2.30}$$

Similarly, based on the relationship $W_m \subset V_{m+1}$, there also exist

$$\psi(t) = \sum_n g(n)\sqrt{2}\phi(2t - n), \ n \in \mathbb{Z}, \tag{2.31}$$

where, $g(n)$ is called the coefficient sequence of wavelet. Due to the orthogonal and complementary requirements of subspaces W_m and V_m, there should be

$$g(n) = (-1)^n h(1 - n). \tag{2.32}$$

In signal processing, $g(n)$ and $h(n)$ are called *Quadrature Mirror Filters*. In practice, these filters are usually designed with the orthogonal wavelets, e.g., Daubechies' wavelets, in order to obtain the orthogonal representations of a signal.

Implementation of DWT and IDWT

Based on the structure of the L^2 space shown in Eqn. (2.29), combing Eqn. (2.27) and (2.30), then any function $f(t) \in L^2(\mathbb{R})$ can be written as:

$$f(t) = \sum_k c^{(m=0)}(k)\phi_{m=0,k}(k) + \sum_k \sum_{m=0}^{M-1} d^{(m)}(k)\psi_{m,k}(k), \tag{2.33}$$

where, $c^{(m)}(k)$ and $d^{(m)}(k)$ are the approximation information and detail information of the original signal $f(t)$ at the resolution 2^m, respectively. The process to get them is called Discrete Wavelet Transform (DWT), or Wavelet Decomposition (WD). So information $c^{(m)}(k)$ and $d^{(m)}(k)$ are also called *wavelet coefficients*. They are the representations of signal $f(t)$ at the resolution 2^m in wavelet domain. For example, $c^{(0)}(k)$ is the approximation coefficient at the coarsest resolution, while $d^{(M-1)}(k)$ is the detail coefficient at the finest resolution. Conversely, the reconstruction of original signal $f(t)$ based on these wavelet coefficients is called Inverse Discrete Wavelet Transform (IDWT).

Both DWT and IDWT can be efficiently implemented with *scaling filter* and *wavelet filter* whose coefficients are given by $h(n)$ and $g(n)$, respectively. Specially, to calculate the wavelet coefficients with DWT, there are:

$$c^{(m)}(k) = \sum_{n} h(n-2k)c^{(m+1)}(n), \tag{2.34}$$

and

$$d^{(m)}(k) = \sum_{n} g(n-2k)c^{(m+1)}(n). \tag{2.35}$$

In case of the wavelet coefficients at the low resolution 2^m are known, the approximation information at the high resolution 2^{m+1} is computed by

$$c^{(m+1)}(k) = \sum_{n} c^{(m)}(n)h(k-2n) + \sum_{n} d^{(m)}(n)g(k-2n). \tag{2.36}$$

Eqn. (2.36) indicates the IDWT process. Eqn. (2.34-2.36) are called *pyramid transform* or *Mallat's algorithm* (Mallat, 1989), as intuitively shown in Fig. 2.8. Note that, the computational complexity of Mallat's algorithm is $O(N)$, where N is the length of the original signal. Therefore, the algorithm is really efficient and as a result widely used in many applications.

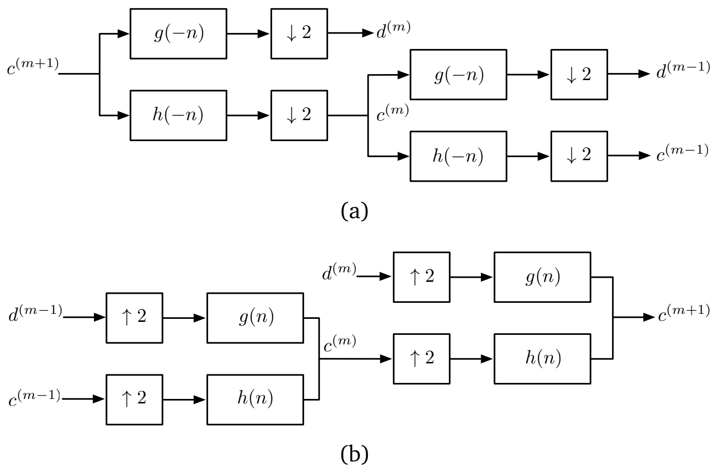


Figure 2.8: Pyramid transform using filters. (a) DWT, $\downarrow 2$: removing one for every two points; (b) IDWT, $\uparrow 2$: inserting one zero between every two points.

Practical Case

However, in practice, a finite signal should be considered. Several practical issues will arise when taking the DWT or IDWT for a finite signal. Based on the implementation of DWT and IDWT shown in Fig. 2.8, it can be found that there are mainly two operations in the pyramid transform: *downsampling/upsampling* and *filtering*.

Downsampling and upsampling operations can be easily understood from Fig. 2.8. There are two types of sampling strategies:

1) *even type*: When downsampling on a one-dimensional signal $\boldsymbol{x} = (1, 2, 3, 4, 5)$, the odd-numbered points are removed and the removing starts with the first point. It leads to $\boldsymbol{y} = (\downarrow 2)\boldsymbol{x} = (2, 4)$. The upsampling operation inserts one zero between every two samples but the insertion starts with the second point, i.e., $(\uparrow 2)\boldsymbol{y} = (2, 0, 4)$.

2) *odd type*: In this case, downsampling and upsampling have similar operations to the even type except for the start point. Now the removing of downsampling starts with the second point, while the insertion of upsampling starts with the first point. Therefore, in this case, the results will be $\boldsymbol{y} = (\downarrow 2)\boldsymbol{x} = (1, 3, 5)$ and $(\uparrow 2)\boldsymbol{y} = (0, 1, 0, 3, 0, 5, 0)$.

If denote s as the length of the original signal, and $\varepsilon = 1$ for even sampling type, $\varepsilon = 2$ for odd sampling type, then the length of the downsampled signal d will be written as:

$$d = \frac{s}{2} + \left(\varepsilon - \frac{3}{2}\right)\left(s - 2\left\lfloor \frac{s}{2} \right\rfloor\right), \tag{2.37}$$

while the length of the signal upsampled from the original signal will be:

$$u = 2(s + \varepsilon) - 3. \tag{2.38}$$

Eqn. (2.37-2.38) can easily be derived from the lemmas described in Rajmic and Prusa (2014).

While the implementation of filtering is usually performed by convolution that may cause the so-called *border effect* (Montanari et al., 2015; Pacola et al., 2016). In practice, the convolution window extends partially at the borders because of the absence of available data there. It will produce abnormal coefficients and hence cause a border effect when performing WT. Typically, extending the signal is a good solution. Traditional extension strategies include zero-padding, periodization, and symmetrization (Strang and Nguyen, 1996; Walnut, 2013).

- Zero-padding: This method is simply to treat the finite signal as an infinite signal padded with zeros. But it often results in a longer length of wavelet coefficients than of the original signal, thus decreases efficiency.

- Periodization: This method views the signal as a periodic signal, then the data outside the original signal domain is periodically extended. In this method, discontinuities may be artificially created at the border.

- Symmetrization: The finite signal can also be symmetrically extended by the boundary values, which seems to work fine in many applications.

As shown in Fig. 2.8(a), in DWT process, the detail coefficients $d^{(m)}$ and approximation coefficients $c^{(m)}$ at the low resolution 2^m can be obtained from the approximation coefficients $c^{(m+1)}$ at the high resolution 2^{m+1} with the procedure as follows:

1) *Border extension*: Extend $c^{(m+1)}$ by $(l-1)$ samples from both sides. l is the length of the filters used ($g(-n)$ or $h(-n)$ in Fig. 2.8(a)). The extended values depend on the chosen extension strategy. The length of the extended result will be $s+2(l-1)$, where s indicates the length of $c^{(m+1)}$;

2) *Filtering*: Perform convolution of extended result with filter $g(-n)$ for detail information, with filter $h(-n)$ for approximation information. Then the length of the filtered result will be $s+3(l-1)$;

3) *Cropping off*: Crop off $l-1$ samples from both ends of the filtered result, which results in a signal of length $s+l-1$;

4) *Downsampling*: Finally, downsample the cropped vector based on the defined downsampling type and thus produce the coefficients $d^{(m)}$ and $c^{(m)}$.

This procedure can repeat on $c^{(m)}$ to obtain the coefficients at the resolution 2^{m-1}, and the coefficients at lower resolutions can be calculated successively. Conversely, in IDWT process, the approximation coefficients $c^{(m)}$ at the high resolution 2^m can be reconstructed by the detail coefficients $d^{(m-1)}$ and approximation coefficients $c^{(m-1)}$ at the low resolution 2^{m-1}, as shown in Fig. 2.8(b):

1) *Upsampling*: Upsample the coefficients $d^{(m-1)}$ and $c^{(m-1)}$ at the resolution 2^{m-1} based on the defined upsampling type;

2) *Filtering*: Perform convolution of upsampled signal with reconstruction filter $g(n)$ and $h(n)$;

3) *Summing*: Add up the outputs of both filters;

4) *Cropping*: Take the central part of the resultant signal as the approximation coefficients $c^{(m)}$ at the resolution 2^m, with the same length as at the appropriate resolution of DWT. So the length of coefficients at every resolution must be recorded in the DWT process. If denote p as the length of $c^{(m-1)}$ (or $d^{(m-1)}$) and s as the length of $c^{(m)}$, then Rajmic and Prusa (2014) gives:

$$s = \begin{cases} 2p-l+1, & \text{for even } s+l-1, \\ 2p-l-2\varepsilon+4, & \text{for odd } s+l-1. \end{cases} \tag{2.39}$$

2.4.2 Wavelet Frame

Basis

In function space, a function $g(t)$ can be expanded with a set of functions $f_k(t)$ called *basis* that spans vector space F. If a function $g(t) \in F$ has the form:

$$g(t) = \sum_k a_k f_k(t), \tag{2.40}$$

with $k \in \mathbb{Z}$ and $t, a \in \mathbb{R}$, then $f_k(t)$ will be called:

- *basis* for space F, if the set of $\{a_k\}$ is unique for any particular $g(t) \in F$;
- *orthogonal basis*, if $\langle f_k(t), f_l(t)\rangle = 0$ for all $k \neq l$;
- *orthonormal basis*, if $\langle f_k(t), f_l(t)\rangle = \delta(k-l)$.

Based on the definitions, if $f_k(t)$ is an orthonormal basis for space F, then the form of $g(t) \in F$ shown in Eqn. (2.40) will be written as:

$$g(t) = \sum_k \langle g(t), f_k(t)\rangle f_k(t) \text{ and } a_k = \langle g(t), f_k(t)\rangle . \tag{2.41}$$

However, there are cases where the orthogonal basis cannot or shouldn't be obtained. For these cases, a *dual basis set* $\tilde{f}_k(t)$ can be built to construct *biorthogonal bases* with the set $f_k(t)$, such that

$$g(t) = \sum_k \left\langle g(t), \tilde{f}_k(t)\right\rangle f_k(t). \tag{2.42}$$

The elements of $\tilde{f}_k(t)$ are not orthogonal to each other, but to the corresponding element of $f_k(t)$, i.e.,

$$\left\langle f_l(t), \tilde{f}_k(t)\right\rangle = \delta(l-k). \tag{2.43}$$

Frame

To be a basis requires that the set be independent, meaning no element can be written as a linear combination of the others. While if a set of functions is dependent and yet expands function $g(t)$ as in Eqn. (2.42), then the set is called a *frame*.

Wavelet frame theory has been investigated in detail in Daubechies (1990) and Chui and Shi (1993). The frame indicates the bounds of the wavelet signal energy. After dilating and translating the wavelet basis functions $\psi(t)$, the wavelet family is considered as:

$$\psi_{a,b}(t) = |a|^{-1/2}\psi\left(\frac{t-b}{a}\right), \tag{2.44}$$

where, $b \in \mathbb{R}$, $a \in \mathbb{R}^+$ with $a \neq 0$. $\psi(t)$ is supposed to be admissible, which means:

$$C_\psi = \int_{-\infty}^{+\infty} \frac{\left|\hat{\psi}(\omega)\right|^2}{\omega} d\omega \leq \infty, \tag{2.45}$$

where, $\hat{\psi}(\omega)$ is the Fourier transform of $\psi(t)$. In order to reduce the computational complexity of continuous wavelets and apply the wavelet transform in engineering applications, the dilation parameter a and the translation parameter b are discretized by:

$$a = a_0^m,\ b = nb_0a_0^m, \tag{2.46}$$

where, m, n range over $\mathbb{Z}$, and $a_0 > 1$, $b_0 > 0$ are fixed. So the discretized wavelet family becomes:

$$\psi_{m,n}(t) = {a_0}^{-m/2}\psi\left({a_0}^{-m}t - nb_0\right). \tag{2.47}$$

Wavelet frame is defined on the discretized wavelet family:
A function $\{\psi_{m,n}(t); m, n \in \mathbb{Z}\}$ constitutes a frame for $L^2(\mathbb{R})$, if exist $0 < A, B < \infty$ so that

$$A\|g(t)\|^2 \leq \sum_{m,n} \left|\left\langle g(t), \psi_{m,n}(t)\right\rangle\right|^2 \leq B\|g(t)\|^2, \tag{2.48}$$

where, A and B are the lower and upper frame bound of $\psi_{m,n}(t)$, respectively. They seem to constrain the energy of normalized coefficients. The frame bound can be calculated with the proposition in Daubechies (1992):
If ψ, a_0 satisfy

$$\begin{cases} \inf\limits_{1\leq|\omega|\leq a_0} \alpha(\omega) > 0, \\ \\ \sup\limits_{1\leq|\omega|\leq a_0} \alpha(\omega) < \infty, \end{cases} \tag{2.49}$$

where, $\alpha(\omega) = \sum\limits_{m\in\mathbb{Z}} \left|\hat{\psi}\left(a_0^m\omega\right)\right|^2$. And if $\beta(s) = \sup_\omega \sum_m \left|\hat{\psi}\left({a_0}^m\omega\right)\right| \left|\hat{\psi}\left({a_0}^m\omega + s\right)\right|$ decays at least as fast as $(1+|s|)^{-(1+\epsilon)}$, with $\epsilon > 0$, then there exists $(b_0)_{\text{thr}} > 0$ such that the $\psi_{m,n}(t)$ constitute frames for all choices $b_0 < (b_0)_{\text{thr}}$. For $b_0 < (b_0)_{\text{thr}}$, the frame bounds for the

$\psi_{m,n}(t)$ can be computed as:

$$\begin{cases} A &= \dfrac{2\pi}{b_0}\Big\{\inf\limits_{1\le|\omega|\le a_0} \alpha(\omega)-\gamma(k)\Big\}, \\ B &= \dfrac{2\pi}{b_0}\Big\{\sup\limits_{1\le|\omega|\le a_0} \alpha(\omega)+\gamma(k)\Big\}, \\ \gamma(k) &= \sum\limits_{\substack{k=-\infty\\k\neq 0}}^{\infty}\left[\beta\left(\dfrac{2\pi}{b_0}k\right)\beta\left(-\dfrac{2\pi}{b_0}k\right)\right]^{1/2}. \end{cases} \tag{2.50}$$

If $A=B$, the wavelet frame is called *tight frame*. Moreover, if $A=B=1$, a tight frame will become an orthogonal basis. As proofed in Daubechies (1990), the function $g(t)$ can be expanded by a tight frame $\psi_{m,n}(t)$:

$$g(t)=A^{-1}\sum_k \langle g(t),\psi_{m,n}(t)\rangle \psi_{m,n}(t). \tag{2.51}$$

2.4.3 Wavelet Neural Network

Recently, there were some attempts to combine wavelet transform with NN to design the so-called Wavelet Neural Network (WNN), and they achieved good results in many tasks (Adamowski and Chan, 2011; Yao et al., 2013; Ganjefar and Tofighi, 2015). Taking advantage of the strong feature extraction capability of wavelets, and the superior learning ability of DNN, WNN is becoming a hot topic in the scientific community (Alexandridis and Zapranis, 2013; Chitsaz et al., 2015). The popularity of deep learning will be facilitated further, which benefits from the fruitful achievements of wavelet analysis in the last decades.

Continuous Wavelet Transform (CWT) is always used to understand the concept of the wavelet transform, but it's not efficiently implemented on computers. In order to improve computational efficiency, the values of the translation and dilation parameters are often limited to some discrete lattices, i.e., performing the DWT. According to the wavelet transforms utilized, existing WNN can be classified into the following two types (Billings and Wei, 2005):

1) *Adaptive WNN* (Zhang and Benveniste, 1992; Zhang, 1997; Alexandridis and Zapranis, 2013): where the wavelets act as the activation functions and they perform the CWT in the hidden layers. In these models, the weights of the neural network, the dilation and translation parameters of the wavelets are the trainable parameters that need to be updated with the gradient descent algorithms. The big amount of parameters to be learnt and the low efficiency of CWT result in slow training.

2) *Fixed grid WNN* (Adamowski and Chan, 2011; Partal et al., 2015; Bakshi and Stephanopou-

los, 1993): where the dilation and translation parameters of the wavelets are predetermined, only the weights are needed to be trained, i.e., the DWT is performed there. However, there are not many theories for determining the dilation and translation parameters in advance.

The fixed grid WNN model is commonly used in engineering applications because of its high efficiency. The structures of fixed grid WNN can also be further classified into two types (Yao et al., 2013):

1) *Compact Wavelet Neural Network* (CWNN) (Doucoure et al., 2016; Wang and Adeli, 2015): where the wavelet basis function is employed as the "wavelon" in the hidden layers. The activations of each hidden layer are the wavelet coefficients after DWT. So the training process is performed in the wavelet domain, which is advantageous for extracting features efficiently from data.

2) *Loose Wavelet Neural Network* (LWNN) (Omerhodzic et al., 2013; Chen et al., 2010; Sunny et al., 2012; Benzy and Jasmin, 2015): unlike the CWNN, where the DWT stems from the wavelons, in LWNN model the DWT acts as a preprocessing step on the original data and then the transformed results are fed into the neural network. It can be seen as a conjunction model concatenating the DWT with the network. Of course, the network model can be designed and trained as usual.

The structure of CNN can also be analyzed from the viewpoint of wavelet transform. For instance, Fujieda et al. (2017, 2018) used multiresolution concepts to explain feature extraction of CNN and reformulate the convolution and pooling operations. After adding the wavelet representations of data, their methods supplemented the absent spectral information for better performance. From the perspective of signal processing, the ConvL of CNN can also be seen as filter banks (Jarrett et al., 2009; Rippel et al., 2015; Andrearczyk and Whelan, 2016), so the knowledge of designing filters can be used to build CNN models. Considering the disadvantages of pooling operation, the PoolL can be replaced with the wavelet decomposition to prevent information loss (Chaabane et al., 2017). Moreover, thanks to the sparsity of wavelet, the CNN structure was demonstrated more robust to noise after combined with wavelet transform (Qiu et al., 2018). However, the structures of these WNNs are shallow so that they needn't face the training problems arised in deep structures that are desirable for complex problems.

2.5 Discussions

Dimensionality reduction is an important data preprocessing step for low computational complexity. While information loss should be avoided when designing the algorithms to reduce dimensionality, since it may destroy or ignore the key features of data and consequently degrade the learning performance. In addition, the designed algorithm is expected to pre-extract the key features for fast training.

FT is a well-known tool to get the frequency features of data. Even though the dimensionality of data is not reduced, FT can be a good start to investigate the effectiveness of feature pre-extraction.

Furthermore, the correlations among attributes can also be taken as the additional features of data, which provide more information for learning process and help to accelerate training. The correlation fractal dimension theory gives the method to estimate the information content of attributes as well as the dataset, which is a promising direction to develop the algorithms for feature pre-extraction.

In learning process, even though traditional ML models or methods have been widely used for feature extraction in various data applications, they have many problems in big data era. For example, kNN method requires many storage spaces since it needs the full dataset all the time. Naïve Bayes method and LR model suffer from underfitting and bad performance in complex problems. While the computational complexity of DT model and SVM increases dramatically when the data becomes bigger.

Therefore, algorithms with strong learning capability are desirable to deal with the growing volume of data. NN has achieved great success in big data applications. Furthermore, the implementation of NN model can easily be distributed on GPU for higher computational efficiency. Actually, the set of NN parameters learnt from data can be seen as a complex basis, with which the original high-dimensional data is transformed to be the low-dimensional discriminative abstractions. The training process of NN model can be seen as trying to construct the complex basis for feature extraction.

Moreover, it has been demonstrated that more complex problems need a deeper network within DNN. The remarkable achievements of DNN mainly benefit from the growing number of nested activation functions when adding hidden layers, since a large number of hidden layers give a strong nonlinear fitting capability to the model for complex real-world problems. Also, the significantly increased volume of data will reduce the risk of overfitting of the DNN model.

However, training a deep network is not easy. Gradient problems and slow convergence speed are two of the most notorious issues in DNN training. They can be analyzed from two sides. On the one hand, feeding the original data will increase the learning difficulties due to the noise, which requires a carefully designed model with the denoising step and a very strong learning capability. Besides, the dimensionality curse of original data results in a heavy computational burden, which will slow down the training process as well. On the other hand, the poorly designed DNN structure may cause training problems inherently. In particularly, improper activation functions will cause gradient vanishing or exploding problems. Furthermore, the internal covariate shift coming from the unnormalized data of hidden layer also results in slow convergence.

Based on the analysis, there can be two directions to alleviate the difficulties in DNN training. The first one is to preprocess the data, i.e., to extract the key features of the original data with a transform. Meanwhile, the dimensionality of data is also expected to be reduced. The second research direction is to optimize the DNN model, i.e., to design a new activation function to constrain and normalize the values of propagating data. It is supposed to deal with the gradient problems and ensure the data flows well, which will accelerate DNN training further. The activation function is also expected to have good nonlinearity and sparsity that help DNN model to cope with more complex problems and to be robust to noise. These two researches, i.e., data processing and optimization of DNN model, can also be combined to form an end-to-end topology for a high computational efficiency.

Wavelet theory is promising to provide the solutions. On the one hand, wavelet transform gives the representations of data at different resolutions, where the key features of data are extracted with different scales. Training on the datasets with different extracted features produces different results, among which the best one can be picked out. Additionally, the wavelet transform will also suppress the noise thanks to the sparsity of wavelet, and as a result the features can be detected more easily. On the other hand, tight frame wavelet constrains the energy of wavelet. It may alleviate the gradient problems and accelerate DNN training if it acts as the activation function. Moreover, the nonlinearity leads wavelet to be a promising choice of activation function for DNN models.

3 Fast Deep Learning in Frequency Domain

Fourier Transform (FT) is widely used in engineering applications to analyze the frequency components of time-domain signals. The frequency representations may provide more discriminative features than the original signal. This chapter preliminarily investigates the training process performed in frequency domain with a new topology of Deep Neural Network (DNN). Two experiments on speech signals demonstrate the advantages of frequency representations for the improvements of training performance.

3.1 Features in Frequency Domain

According to Fourier series theory, any periodic signal $\boldsymbol{x}_T$ can be approximated with infinite sum of cosines and sines as:

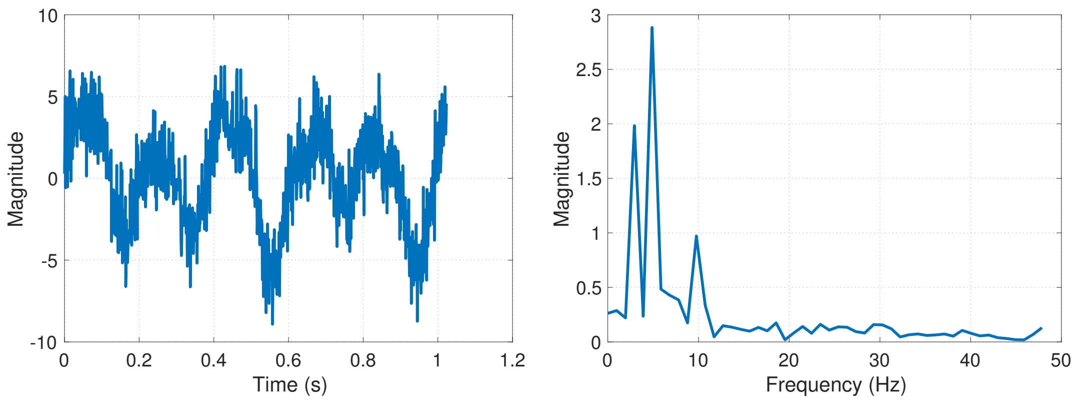
$$\boldsymbol{x}_T = a_0 + \sum_i A_i \cos(2\pi f_i t) + B_i \sin(2\pi f_i t), \tag{3.1}$$

where, a_0 is the direct current component. It can be found that the periodic signal $\boldsymbol{x}_T$ has many cosine or sine components with frequency f_i, which means that, with Eqn. (3.1) the analysis of time-domain signal turns to the analysis in frequency domain. When the period T tends to $+\infty$, the signal $\boldsymbol{x}_T$ becomes aperiodic and it is often the case in the real world.

Then the frequency analysis of the aperiodic signal $\boldsymbol{x}$ needs the FT defined as:

$$\hat{x}(\omega) = \int e^{-j\omega t} x(t) dt. \tag{3.2}$$

After FT, the frequency representations of the original signal may provide more discriminative features. Fig. 3.1 shows a simple application of noise reduction for the synthetic signal polluted by Gaussian noise. The signal is not easy to be identified in time domain, while it becomes distinct in frequency domain since the energy of the Gaussian noise is distributed to different frequency components by FT.



(a) (b)

Figure 3.1: A signal polluted by Gaussian noise with the Signal-to-Noise Ratio (SNR) is set at 5dB. The signal is generated by $x = 2\sin(6\pi t) + 3\sin(10\pi t) + \cos(20\pi t)$. (a) Time domain; (b) Frequency domain.

Another popular application of FT is speech recognition. The speech signal is a highly nonstationary signal over time. However, over a sufficiently short period of time (10-30ms), its spectral characteristics are fairly stationary. Thus the speech signal can be analyzed based on a number of frames.

The process of speech production involves many organs and muscles, but based on a simple mathematical model for one cycle of the glottal flow waveform defined in Rosenberg (1971), the FT of the glottal waveform is characterized by many harmonics (Loizou, 2013). The first harmonic occurs at the *fundamental frequency* (F_0), and the other harmonics occur at integer multiples of the fundamental frequency.

Therefore, FT is suitable to analyze the speech signal since it also gives the harmonics of signal. For example, based on the fact that the F_0 range is about 60-150Hz for male speakers and 200-400Hz for females and children (Peterson and Barney, 1952), the speech signal can be used to identify the gender of speakers. Fig. 3.2 shows the frequency representations of two sentences coming from a male and a female. The energy of the male voice mainly locates in the low frequency band (< 600Hz), while the energy of the female voice also covers the high frequency band (> 600Hz). It intuitively gives the difference of speakers' gender in frequency domain.

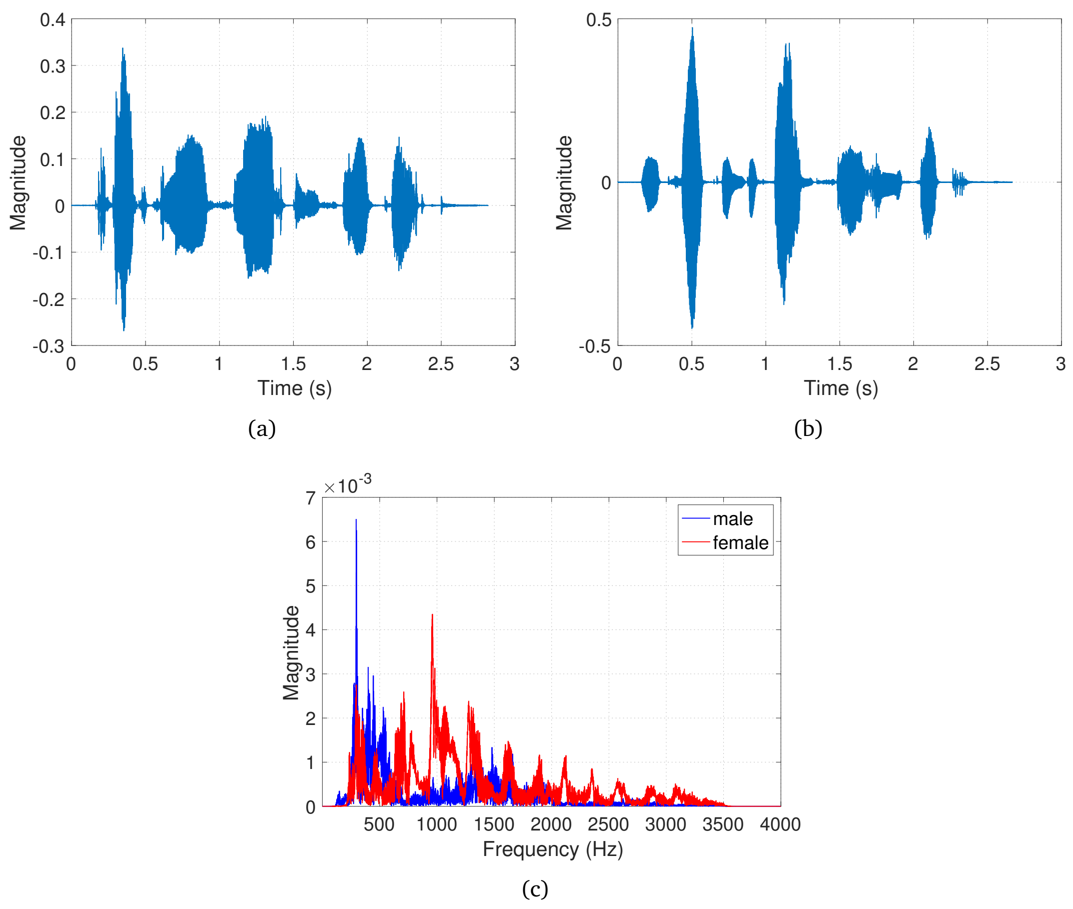


Figure 3.2: Two sentences coming from NOIZEUS database (Hu and Loizou, 2007). (a) a male speaker says: "the birch canoe slid on the smooth planks" in time domain, with $F_0 = 135$; (b) a female speaker says: " we talked of the sideshow in the circus" in time domain, with $F_0 = 245$; (c) the sentences in frequency domain.

3.2 Fast Fourier Transform

FT is defined for the continuous signal that is usually sampled to be discrete for processing with computers. Under this circumstance, the Discrete Fourier Transform (DFT) is applied to analyze the discrete signal. The DFT of a N-point signal $x[k]$ is defined as:

$$y[m] = \sum_{k=0}^{N-1} x[k] W_N^{km}, \ m = 0, 1, \cdots, N-1, \tag{3.3}$$

where, $W_N^{km} = e^{-j\frac{2\pi}{N}mk}$. It can be found that the frequency representation of a discrete signal also has length N, while obtaining each frequency point $y[m]$ needs the whole discrete signal $\boldsymbol{x}$. Thus, directly performing the DFT with Eqn. (3.3) is time-consuming when the signal length N becomes very large. Fast Fourier Transform (FFT) algorithm is designed for the efficient implementation of DFT.

Nowadays, there have been a lot of variants to perform FFT, but most of them are proposed based on the idea of "radix-2 splitting" (Cooley and Tukey, 1965). Repetition of the process that quickly computing a N-point DFT from a pair of $(N/2)$-point DFTs is the heart of the radix-2 FFT idea (Van Loan, 1992). The principle of the traditional radix-2 Decimation In Time (DIT) algorithm is to split the sequence to the even and odd subsequences in time domain, and then to obtain the DFT of the whole sequence by composing the DFTs of the subsequences.

Benefits from the properties of W_N^{km}, the basic component of FFT, called *butterfly computation*, can be derived as:

$$\begin{aligned} y[m] &= y_1[m] + W_N^m y_2[m] \\ y[m+N/2] &= y_1[m] - W_N^m y_2[m], \ m = 0, 1, \cdots, N/2-1, \end{aligned} \tag{3.4}$$

where, $y_1[m]$ and $y_2[m]$ are the $(N/2)$-point DFTs of the even subsequence $x_1[k]$ and odd subsequence $x_2[k]$, respectively. The butterfly computation shown in Eqn. (3.4) can be illustrated in Fig. 3.3. While the two $(N/2)$-point DFTs can be further split to four $(N/4)$-point DFTs, and then eight $(N/8)$-point DFTs, until $N/2$ 2-point DFTs. Thus the 8-point FFT algorithm can be performed as shown in Fig. 3.4.

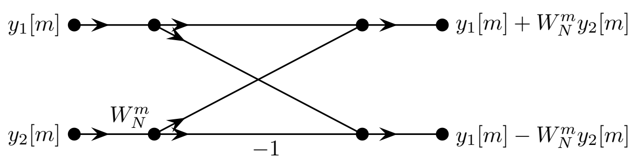


Figure 3.3: The butterfly computation of the radix-2 FFT.

The radix-2 FFT can also be interpreted as a sparse matrix factorization that is called *Cooley-Tukey Radix-2 Factorization* in Van Loan (1992):
If a discrete signal $\boldsymbol{x}$ has length $N = 2^t$ and its DFT is denoted as $\boldsymbol{y}$, then there is a matrix $\mathbf{F}$

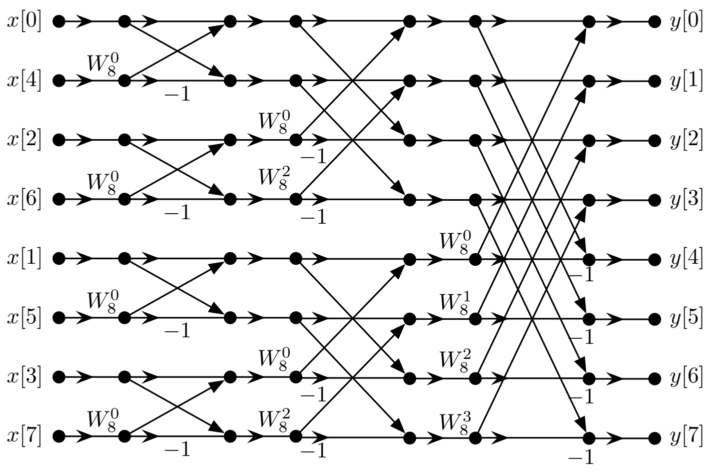


Figure 3.4: 8-point radix-2 FFT.

makes

$$\boldsymbol{x}\mathbf{F} = \boldsymbol{y}, \tag{3.5}$$

and

$$\mathbf{F} = \mathbf{P}\mathbf{A}_1^{\mathrm{T}}\mathbf{A}_2^{\mathrm{T}} \cdots \mathbf{A}_t^{\mathrm{T}}. \tag{3.6}$$

P is called *bit-reversal permutation matrix* used to arrange the elements of $\boldsymbol{x}$ to the bit-reversal index. Let $x[k]$ be the kth element in $\boldsymbol{x}$ and $[\boldsymbol{x}\mathbf{P}]_k$ be the kth element in $\boldsymbol{x}\mathbf{P}$, then there is:

$$[\boldsymbol{x}\mathbf{P}]_k = x[r(k)], \tag{3.7}$$

where, $r(\cdot)$ is a function defined as $r\left((b_{t-1}\cdots b_1 b_0)_2\right) = (b_1 b_0 \cdots b_{t-1})_2$. **P** can be computed by

$$\begin{aligned} \mathbf{P} &= \mathbf{R}_t \cdots \mathbf{R}_1, \\ \mathbf{R}_q &= \mathbf{I}_{2^{t-q}} \otimes \mathbf{\Pi}_{2^q},\ q = 1, 2, \ldots, t. \end{aligned} \tag{3.8}$$

In Eqn. (3.8), the symbol ⊗ represents Kronecker Products. $\mathbf{I}_n$ is the n-dimensional identity matrix and **Π** is called the *even-odd sort permutation* that is defined as follows (assuming that n is even):

$$\mathbf{\Pi}_n = \mathbf{I}_n(:, \boldsymbol{v}),\ \boldsymbol{v} = [0, 2, \ldots, n, 1, 3, \ldots, n-1]. \tag{3.9}$$

If **Π** is applied to a vector, then it groups the even-indexed components first and the odd-indexed components second. While for matrix **A**, the result of **AΠ** is just **A** with its even-indexed and odd-indexed columns grouped together.

In Eqn. (3.6), each $\mathbf{A}_q$, $q = 1, 2, \ldots, t$, is a sparse complex matrix and it has only two nonzero entries per row. This is because each $\mathbf{B}_L$ has two nonzeros per row: one from $\mathbf{I}_s$ and one from $\mathbf{\Omega}_s$, these matrices are defined as:

$$\begin{aligned}
\mathbf{A}_q &= \mathbf{I}_r \otimes \mathbf{B}_L, & L &= 2^q, r = N/L, \\
\mathbf{B}_L &= \begin{bmatrix} \mathbf{I}_s & \mathbf{\Omega}_s \\ \mathbf{I}_s & -\mathbf{\Omega}_s \end{bmatrix}, & s &= L/2, \\
\mathbf{\Omega}_s &= \operatorname{diag}\left(W_L^0, W_L^1, \ldots, W_L^{s-1}\right), & W_L &= e^{-j2\pi/L}.
\end{aligned} \tag{3.10}$$

3.3 Fast Fourier Transform Neural Network

Based on Eqn. (3.6), the DFT of a discrete signal can be obtained with a series of vector-matrix products that can be accelerate by GPU. Moreover, both the matrices $\mathbf{P}$ and $\mathbf{A}_q$ are highly sparse, which means that the implementation of FFT as the vector-matrix products will not cost too many computing resources. Therefore, under the consideration that the frequency representations of a signal may provide more information and the FFT can be efficiently implemented inside the network, a new topology of DNN is proposed for fast training, i.e., the Fast Fourier Transform Neural Network (FFTNN). The matrices mentioned in Eqn. (3.6) can be considered as the weights of the $t+1$ hidden layers without activation functions and bias, as sketched in Fig. 3.5.

The proposed FFTNN model has two components: the first one implements FFT, while the second one is the traditional Multilayer Perceptron (MLP). Both these two components perform vector-matrix products or matrix multiplications. While the training process of FFTNN model only happens in the MLP component since the "trainable" parameters of the FFT component are fixed for the DFT. The FFT component can also be seen as a highly sparse MLP model that has been trained for the purpose of performing DFT.

Because real matrix $\mathbf{P}$ has only one nonzero entry per column, for a discrete signal $\boldsymbol{x}$ with length N, $\boldsymbol{x}\mathbf{P}$ involves N flops[1] (real multiplications). While the vector-matrix product with complex matrix $\mathbf{A}_q$ involves $2N$ complex multiplications and $2N$ complex add, which result in $16N$ flops since a complex add involves two flops and a complex multiplication involves six flops. Totally, the FFT component needs $N + 16Nt$ flops. While in the MLP component, the product of the vector input with the weight matrix $\mathbf{W} \in \mathbb{R}^{m \times n}$ needs $2mn$ flops.

[1]In this chapter, the flop is just the counting of multiplication or add operations.

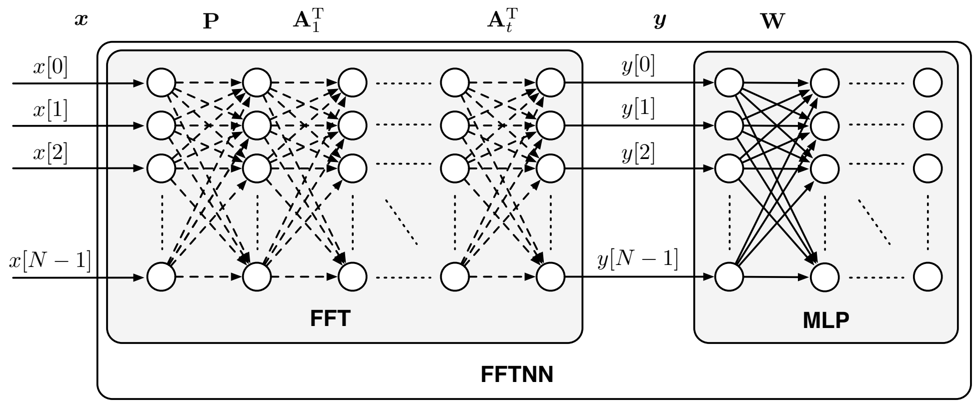


Figure 3.5: Fast Fourier Transform Neural Network. The dashed line means the computations between two sparse matrices, the solid line means the computations between two common matrices.

3.4 Experiments and Results

In this section, we performed two experiments on speech signals to study the proposed FFTNN model. These experiments were running on the Ubuntu 18.04 system, with CPU Intel Core i5-8400. TensorFlow (GPU version 1.10) (Abadi et al., 2016) was used to accelerate computing. The graphic card was the NVIDIA GeForce GTX1050Ti with 768 CUDA cores.

3.4.1 Howl Noise Reduction

Amplification systems are commonly used, such as in the meeting room, to amplify the speaker's voice. However, under this circumstance, the microphone can usually collect the amplified voice reflected by the walls. After some cycles the amplification system will output the noise with a very high energy. The noise is also known as howl that may cover the clean signal and heavily degrade the quality of voice.

Fig. 3.6 shows a simple method used to collect the clean signal and the noisy signal polluted by howl noise. The cellphone was playing a piece of voice. D_1 and D_2 are the distances of the cellphone and the amplification system with the recorder, respectively. Throughout the process of collecting signal s, D_1 was kept the same. When D_2 was large, there was no howl noise and the signal s was considered as the clean signal. When D_2 became small, the amplified voice was fed back and added onto the clean signal, which produced howl saved as the noisy signal. In this experiment, we got the clean signal and noisy signal with 2000000 points. They were divided into 10937 frame samples for training and 4688 frame samples for testing. Each frame had length of 128.

These training samples were fed into FFTNN model directly while the training was

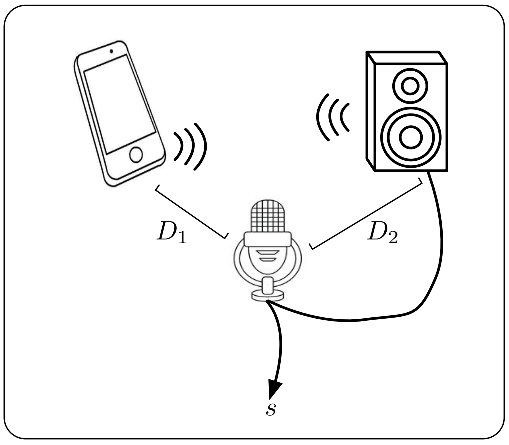


Figure 3.6: The method used to collect clean voice and noisy voice polluted by howl noise.

performed in frequency domain inherently. In practice, the outputs of the FFT component are the concatenations of the real and image parts of the complex DFT results. Besides FFTNN model, two other models were performed for comparison. Specially, based on Least Mean Square (LMS) algorithm, a Frequency-Domain Adaptive Filter (FDAF) was designed to remove the howl noise, as shown in Fig. 3.7. In addition, a traditional MLP model without FFT component, called "tMLP1", was also built to perform training in time domain. Note that, the tMLP1 model had the same number of hidden units and hyper-parameters with the MLP component of FFTNN model, as listed in Tab. 3.1.

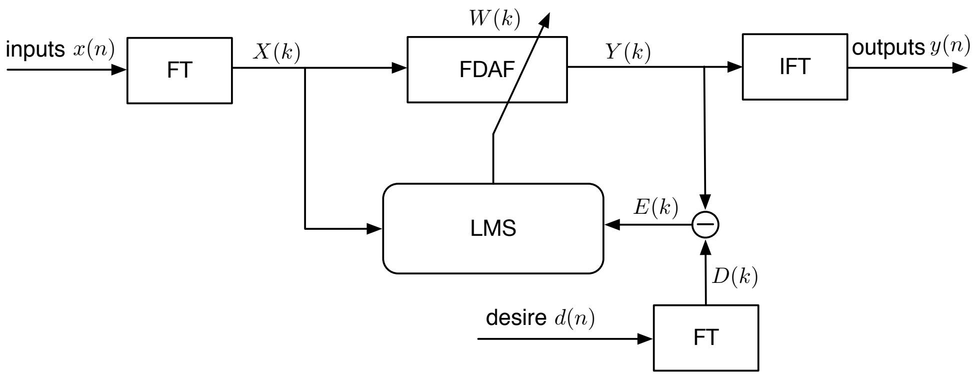


Figure 3.7: Frequency-Domain Adaptive Filter with LMS algorithm.

The test loss values of FFTNN and tMLP1 models are compared in Fig. 3.8. From the figure it can be found that the FFTNN model quickly converged to a low loss level, which means that the frequency representations helped DNN to learn the discriminative features of voice and howl noise. Moreover, the SNRs of the denoised training and test data by three models are listed in Tab. 3.2. The results also indicate the advantages of FFTNN model in howl noise reduction. The denoised voice of FFTNN model had the best quality while it contained a little distortion, which left a room for future improvements.

Table 3.1: The number of hidden units and the hyper-parameters of the FFTNN and tMLP1 models in the howl noise reduction experiment.

number of hidden units	[200, 100, 100, 100, 200]
learning rate	0.001
epochs	300
batch size	1024
optimizer	Adam
weights initialization	Xavier
bias initialization	zero
activation function	ELU
loss function	CE

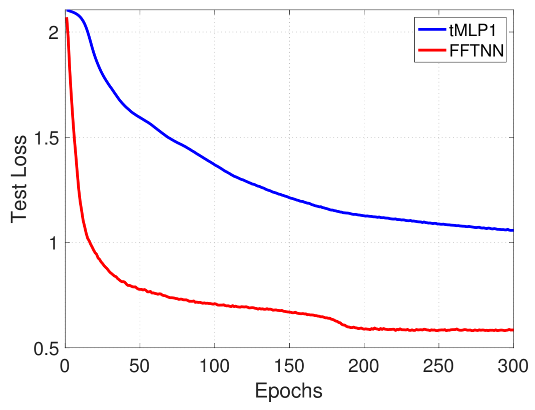


Figure 3.8: Test loss values of the FFTNN and tMLP1 models in the howl noise reduction experiment.

3.4.2 Speech Recognition

In this section, three speech recognition tasks were set up on a speech database to compare the proposed FFTNN model with a traditional MLP model named "tMLP2". The samples of the database were the frames split from the speech signals of five speakers. While the content of each speech signal was one of five short commands in Mandarin (translated to English: "go forward", "step back", "turn left", "turn right", and "stop"). Thus, the three tasks can be termed as: 1) Mission A: 5-class voiceprint recognition, i.e., to identify five speakers; 2) Mission B: 5-class semantic recognition, i.e., to identify five commands; 3) Mission C: 25-class voiceprint-semantic recognition, i.e., to identify the speaker and the command at the same time.

Table 3.2: The SNRs of denoised training and test data by three models.

Model	Denoised training data (dB)	Denoised test data (dB)
noisy signal	-18.15	-18.28
FDAF	1.19	0.27
tMLP1	3.79	2.99
FFTNN	**7.38**	**5.61**

Then three datasets were built for the three missions, respectively. In Mission A and B, each class had 10000 samples for training and 5000 samples for testing. While in Mission C, each class had 2000 samples for training and 1000 samples for testing. Each sample in the datasets had length of 1024.

Three FFTNN models were built for the three missions, respectively. Correspondingly, three tMLP2 models were also constructed for comparison. The number of units for each hidden layer of these models are listed in Tab. 3.3. While their hyper-parameters were set the same, as shown in Tab. 3.4.

Table 3.3: The number of hidden units of the FFTNN and tMLP2 models in the speech recognition missions.

Mission	FFTNN	tMLP2
A	[10, 10, 10]	[80, 80, 50, 50, 30]
B	[10, 80, 50]	[100, 100, 80, 80, 30]
C	[10, 50, 10]	[100, 100, 50, 50, 30]

Table 3.4: The hyper-parameters of the FFTNN and tMLP2 models in the speech recognition missions.

learning rate	0.0001
epochs	100
batch size	512
optimizer	Adam
weights initialization	Xavier
bias initialization	zero
activation function	ELU
loss function	CE

The computational flops of the models in different missions are compared in Tab. 3.5. The results show that FFTNN can be designed with lower computational complexity than traditional MLP, even though introducing additional vector-matrix products due to the FFT component. In addition, the FFTNN model learnt features quickly, which can be observed from the training processes illustrated in Fig. 3.9. The accuracy results are also compared in

Tab. 3.6, and they indeed indicate that the FFTNN model achieved better performance in the speech recognition experiments.

Table 3.5: Computational flops of the FFTNN and tMLP2 models in the speech recognition missions.

Mission	FFTNN (F1)	tMLP2 (F2)	F2 / F1
A	185844	192940	1.038
B	197444	258700	1.310
C	187844	244300	1.301

Table 3.6: Accuracy results of the FFTNN and tMLP2 models in the speech recognition missions.

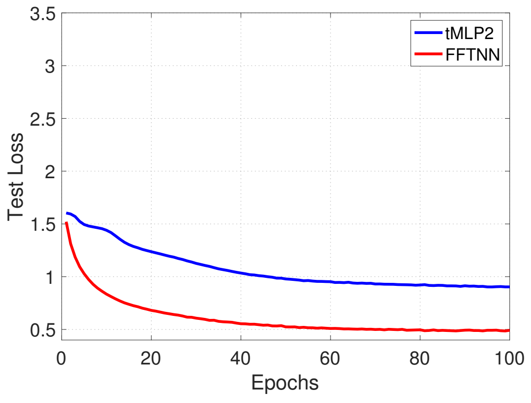
Mission	FFTNN	tMLP2
A	87.73%	67.02%
B	61.57%	28.90%
C	57.14%	19.51%

3.5 Discussions

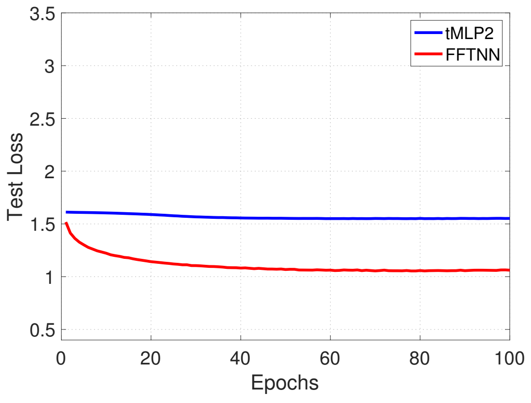
Noise and redundancy of the data collected from the real world usually cover up the key features and slow down DNN training. Preprocessing the raw data is supposed to pre-extract the features, and as a result accelerate training. Fourier transform is a popular tool used to transform the signal from time domain into frequency domain, where the key features may be easier to be detected. However, directly performing FT on discrete signal is time-consuming. As the fast algorithm, FFT has been applied in many applications. But typically, the implementation of FFT is separated from DNN training, which requires the manual intervention if attempting to perform DNN training in frequency domain.

Factorizing the FFT to a series of vector-matrix products is a good solution to bridging the gap between FFT and DNN since DNN training always involves matrix operations. All the factorized matrices of FFT are sparse, which means that the matrix implementation of FFT will not increase considerable computational complexity. Additionally, the GPU will further accelerate this type of compute-intensive operations.

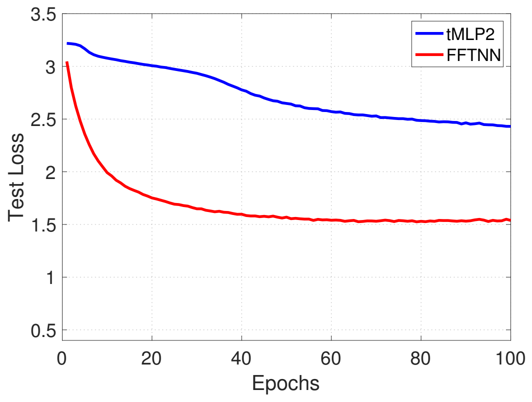
Experiments on speech signals demonstrated the advantages of proposed FFTNN model on speeding up deep learning. The training performed in frequency domain helped to effectively remove howl noise and improve accuracy in speech recognition tasks. Therefore, the idea of combing FFT with DNN provides a direction to design an efficient topology of DNN and extract the features of data for fast deep learning.



(a)



(b)



(c)

Figure 3.9: Test loss values in three speech recognition missions. (a) Mission A; (b) Mission B; (c) Mission C.

4 Feature Pre-extraction of Big Data Based on Multiresolution Representations

Data collected from the real world is always redundant, which means that the information of data is mainly focused on a few attributes. Meanwhile, it is also found that in the wavelet domain, the approximation coefficients indeed contain the most information. Based on the estimation of information contents, this chapter proposes a novel feature pre-extraction method to transform the original dataset into a new one. The transformed dataset is characterized as low-dimensional and feature-extracted, which has been discussed helpful to decrease computational complexity and speed up Deep Neural Network (DNN) training. Experiments demonstrate that the proposed method leads the Machine Learning (ML) models to better performance in the classification tasks.

4.1 The Quantity of Information

4.1.1 Information Content of Attributes: I_a

In big data field, the information of attributes gives hints to discover the features of data. However, different attributes provide information with different quantities. The information content of an attribute reflects its contribution to the dataset. As introduced in section

2.1.3, Individual Contribution (IC) can be used to estimate the potential contribution of an individual attribute to the dataset. It is calculated by inspecting the distributions of the values under an attribute at different scales, i.e., using the box-counting method. However, due to the existence of correlation among attributes, the real contribution of data under attribute a_i, i.e., the information content $I_a(\{a_i\})$, cannot be obtained from the IC when a_i involves other attributes.

On the other hand, Attribute Set Core (ASC) is known as the smallest subset of attributes characterizing the whole dataset. When given a correlation threshold ξ, the ASC ξC can be found out with the *FD-ASE* algorithm (de Sousa et al., 2007). The attributes in ASC are not ξ-correlated with each other and they contain the most information of the dataset.

If $\mathbb{A} = \{a_1, a_2, \ldots, a_E\}$ is denoted as the universal attribute set of the dataset that has E attributes, then the real contribution of attribute $a_i \in (\mathbb{A} - \xi C)$ to the dataset depends on the degree of correlation between a_i and the attributes in ξC. The weaker the correlation between a_i and ξC, the higher contribution of a_i to the dataset, i.e., attribute a_i contains more information. Therefore, the information content of attribute a_i can be estimated by

$$I_a(\{a_i\}) = pD(\{a_i\} \cup \xi C) - pD(\xi C), \; a_i \in (\mathbb{A} - \xi C), \tag{4.1}$$

where, $pD(\cdot)$ is the Partial Intrinsic Dimension (PID) of a sub-dataset.

In principle, the goal of dimensionality reduction can be achieved by directly removing the attributes with low information content. However, besides low-dimensional, a new attribute space which provides more discriminative information is also expected to increase the efficiency of feature extraction and improve the performance, as discussed in section 1.3.1.

4.1.2 Information Content in Wavelet Domain: I_c

From the view of wavelet analysis, the original signal is transformed into the wavelet domain through Discrete Wavelet Transform (DWT) and it is measured along the new axes. The transforms with orthogonal (such as Daubechies compact support orthogonal wavelets) or biorthogonal (such as Cohen-Daubechies-Feauveau families) basis don't remove any information (or noise). They just move it around, aiming to separate out the noise and decorrelate the signal (Strang and Nguyen, 1996).

In fact, the process of DWT is to analyze the original signal from a fine scale to coarse scale. According to the expression shown in Eqn. (2.33), the representation of a finite signal $f(t)$ in the wavelet domain after DWT is a collection of vectors:

$$\{\mathbf{c}_J(k) : k \in \mathbb{Z}\} \cup \{\mathbf{d}_j(k) : 1 \le j \le J; k \in \mathbb{Z}\}, \tag{4.2}$$

where, $J \in \mathbb{N}$ is called decomposition level; $\boldsymbol{c}_J$ contains the approximation coefficients at level J, i.e., the lowest resolution, and $\boldsymbol{d}_j$ contains the detail coefficients at level j, i.e., the higher resolutions. In particular, the original signal $f(t)$ can be seen as the approximation coefficients at the highest resolution, i.e., $\boldsymbol{c}_0$.

These coefficients form the *wavelet decomposition vector* $\boldsymbol{c}$, i.e., $\boldsymbol{c} = (\boldsymbol{c}_J, \boldsymbol{d}_J, \boldsymbol{d}_{J-1}, \ldots, \boldsymbol{d}_1)$. The number of these coefficients constitutes the *bookkeeping vector* $\boldsymbol{l}$, i.e., $\boldsymbol{l} = (\mathrm{len}(\boldsymbol{c}_J), \mathrm{len}(\boldsymbol{d}_J), \mathrm{len}(\boldsymbol{d}_{J-1}), \ldots, \mathrm{len}(\boldsymbol{d}_1), \mathrm{len}(f(t)))$, where $\mathrm{len}(\cdot)$ indicates the number of elements in a vector. Therefore, an example of three-level pyramid transform used to perform DWT can be illustrated in Fig. 4.1.

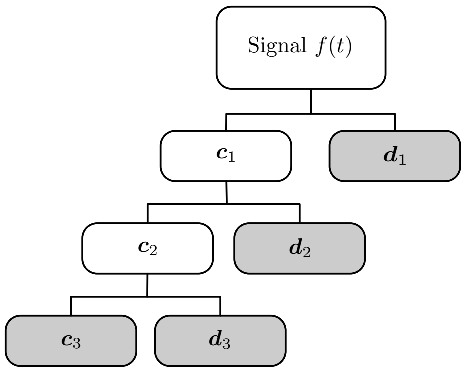


Figure 4.1: Wavelet decomposition coefficients for a three-level pyramid transform.

However, based on the priori knowledge, at a particular level/scale, most information of the natural signal usually locates in the approximation coefficients. Meanwhile, the information will be halved when decomposing the signal from level j to level $j+1$ due to the filtering operation. Hence, it can be found that the information contents of coefficients at different levels roughly respect:

$$I_c(\boldsymbol{d}_J) < \cdots < I_c(\boldsymbol{d}_2) < I_c(\boldsymbol{d}_1) < I_c(\boldsymbol{c}_J). \tag{4.3}$$

A three-level DWT is performed on three images to intuitively illustrate the relationship, as shown in Fig. 4.2. These figures indicate that the approximation part at each level keeps the contour of the image and contains the most information. While the content of the image cannot be easily identified from the detail coefficients. But in some sense, the detail coefficients at a low level seems like giving more information than the ones at a high level, i.e., $I_c(\boldsymbol{d}_{j+1}) < I_c(\boldsymbol{d}_j)$.

In some applications, such as image denoising, audio coding, and decoding, Inverse Discrete Wavelet Transform (IDWT) is used to reconstruct the signal from wavelet domain to time domain. The process of IDWT is to synthesize the wavelet coefficients from a coarse scale to fine scale. In practice, the IDWT can be implemented along the inverse direction of the pyramid transform. According to the multiresolution analysis introduced in section 2.4.1, the wavelet decomposition vector $\boldsymbol{c}$ and bookkeeping vector $\boldsymbol{l}$ are sufficient to reconstruct the signal.

Therefore, DWT gives the representations of the original signal at different resolutions. Some features may be extracted at a resolution while other features may be extracted at another resolution. Typically, in DWT, the original signals are sampled at the highest resolution and then transformed to lower resolutions for feature extraction. However, if the original signals are seen as the data at low resolutions, then IDWT will give their representations at higher resolutions, which may also extract some features that are not easy to be found at the original resolution. The proposed method is inspired by it and will be described in next section.

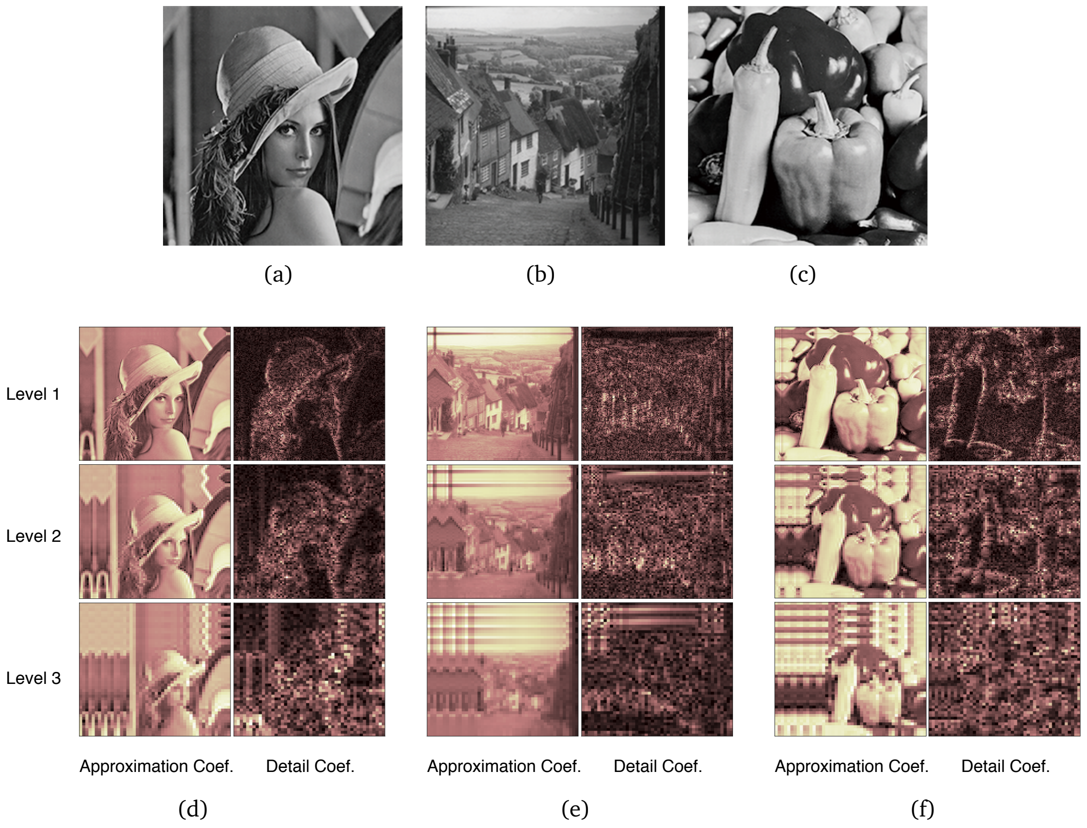


Figure 4.2: The representations of images *Lena*, *Goldhill* and *Peppers* in (a-c) time domain, and (d-f) wavelet domain, respectively. For display performance, the images of coefficients have been mapped into pink color. The detail coefficients at a particular level drawn in the figures are the sum of the corresponding horizontal, vertical and diagonal detail coefficients.

4.2 Proposed Method: FDIDWT

The basic idea of the proposed method is to take each sample of the dataset to be the coefficients in the wavelet domain. IDWT is performed on each sample and finally a new dataset is generated. The new dataset is supposed to extract the features and has low-dimensional attribute space. A J-level IDWT means to transform the original data to be the representations at a J-higher resolution. In order to do this, the wavelet decomposition vector and bookkeeping vector should be constructed at first, which requires the attribute values of each sample to be arranged in the order shown in Eqn. (4.3). This sorting task can be finished with the estimation of the information content of attributes, as defined in Eqn. (4.1). Thus, the proposed method is termed as Fractal Dimension - Inverse Discrete Wavelet Transform or simply FDIDWT.

As discussed before, data under the attributes of ASC $\xi C \subset \mathbb{A}$ contains most information of the dataset defined in $\mathbb{A} = \{a_1, a_2, \ldots, a_E\}$. The number of attributes in ξC is denoted as P. Thus if exists:

$$I_a(\{a_1\}) < I_a(\{a_2\}) < \cdots < I_a(\{a_i\}) < \cdots < I_a(\{a_Q\}), \tag{4.4}$$

where, $a_i \in (\mathbb{A} - \xi C)$ and $1 \le i \le Q$, $P + Q = E$, then it can be assumed that:

$$I_a(DG_J^M) < I_a(DG_{J-1}^N) < \cdots < I_a(DG_j^L) < \cdots < I_a(DG_1^K) < I_a(\xi C), \tag{4.5}$$

where, DG_j^L is the jth group containing L attributes whose information contents respect Eqn. (4.4), i.e.,

$$\begin{aligned}
&DG_J^M = \{a_1, a_2, \ldots, a_M\},\\
&DG_{J-1}^N = \{a_{M+1}, a_{M+2}, \ldots, a_{M+N}\},\\
&\cdots\\
&DG_1^K = \{a_{Q-K+1}, a_{Q-K+2}, \ldots, a_Q\},\\
&M + N + \cdots + K = Q.
\end{aligned} \tag{4.6}$$

Contrasting with the relationship of the information contents of wavelet coefficients shown in Eqn. (4.3), the data under the attributes of DG_j^L can be seen as the detail coefficients $\boldsymbol{d}_j$ at level j. While the data under the attributes of ASC ξC will be taken as the approximation coefficients $\boldsymbol{c}_J$ at level J. Finally, the original dataset is transformed into another one through J-level IDWT. If denote $\boldsymbol{x}_j$ and $\boldsymbol{x}_{\xi C}$ as the values under the attributes in group DG_j^L and ASC ξC respectively for one sample of the original dataset, then the wavelet decomposition vector

$\boldsymbol{c}$ and bookkeeping vector $\boldsymbol{l}$ for the IDWT process can be constructed as:

$$\begin{aligned} \boldsymbol{c} &= \left(\boldsymbol{x}_{\xi C}, \boldsymbol{x}_J, \boldsymbol{x}_{J-1}, \dots, \boldsymbol{x}_1\right), \\ \boldsymbol{l} &= (P, M, N, \dots, K, O), \end{aligned} \tag{4.7}$$

where, O is the number of attributes in the transformed dataset, i.e., the *transformed dimension* in the new attribute space. Tab. 4.1 shows the procedure of the proposed FDIDWT method. To better describe how the algorithm works, now fix the decomposition level $J = 3$. Moreover, as in common cases, the sampling type will be fixed as $\varepsilon = 1$ throughout the thesis. The length of filters m for the pyramid transform is even. Then:

Table 4.1: Proposed feature pre-extraction method - FDIDWT.

Input:	original dataset defined on $\mathbb{A}$, decomposition level J
	transformed dimension O, correlation threshold ξ, scale range n
Output:	new dataset defined on $\mathbb{A}'$
1:	run *FD-ASE* algorithm to find out the attribute set core ξC, denote its length as P;
2:	calculate the information content of the remaining attributes $I_a(a_i)$ with Eqn. (4.1);
3:	arrange the attributes of $\mathbb{A}$ into $\mathbb{B}$ according to their information contents;
4:	based on P, calculate the length of other groups, insert placeholder attributes if needed;
5:	construct the wavelet decomposition vector $\boldsymbol{c}$ and bookkeeping vector $\boldsymbol{l}$;
6:	"Standardize" the original dataset $\mathbb{A}$ along each dimension, as shown in Eqn. (2.1);
7:	perform IDWT on each normalized sample with $\boldsymbol{c}$ and $\boldsymbol{l}$;
8:	generate the new dataset defined on $\mathbb{A}'$;

1) Step 1-3: Given the correlation threshold ξ and scale range n, the *FD-ASE* algorithm (de Sousa et al., 2007) is used to find out the attribute set core ξC. Then the information contents $I_a(\{a_i\})$ of the remaining attributes $a_i \in (\mathbb{A} - \xi C)$ are calculated. The attributes of $\mathbb{A}$ are arranged into a set $\mathbb{B}$ based on the ascending information contents of the remaining attributes[1]. For instance, if the relationship in Eqn. (4.4) holds, then

$$\mathbb{B} = \xi C \cup \{a_1, a_2, \dots, a_i, \dots, a_Q\}. \tag{4.8}$$

2) Step 4-5: P is the number of attributes in ξC. In the proposed method, the length of the approximation coefficients in $\boldsymbol{c}$ is assumed to be not shorter than P. But now, if this length equals P, then Fig. 4.3 shows other groups divided from $\mathbb{B}$. Note that, in practice, the original dimension E, i.e., the number of attributes of the original dataset, may be smaller than the length of wavelet decomposition vector $\boldsymbol{c}$ due to the given decomposition level J and filter length m. In this case, some necessary *placeholder attributes* "$*$" are inserted

[1]Here, $\mathbb{B}$ is not a strict mathematical set since the attributes of it respect an order.

following the ASC group. The goal of the placeholder attributes is to ensure IDWT works while keeps the information content of data as it is. Therefore, the values under the placeholder attributes are identical so that their contributions to the dataset are zero.

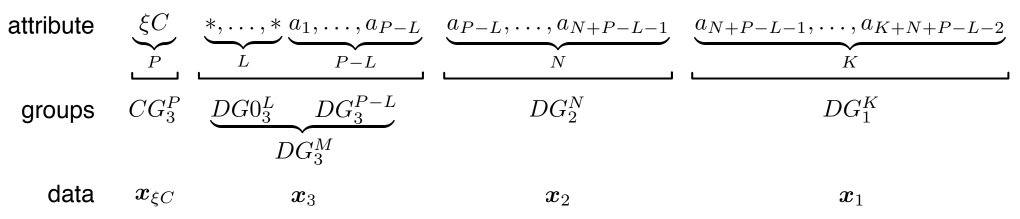


Figure 4.3: Groups divided from sorted remaining attributes. Taking decomposition level $J = 3$ as an example.

Typically, in the signal processing field, signals are transformed to wavelet domain with DWT for processing, then the processed signals are reconstructed with IDWT. Thus, the lengths of the wavelet coefficients at different levels respect the record from DWT, as shown in Eqn. (2.39). However, there is no DWT in the proposed method. Hence there exist many cases of the lengths of the divided groups:

$$
\begin{aligned}
&M = P, \\
&N = \begin{cases} 2M - m + 1, & \text{for even } N + m - 1 \\ 2M - m + 2, & \text{for odd } N + m - 1 \end{cases}, \\
&K = \begin{cases} 2N - m + 1, & \text{for even } K + m - 1 \\ 2N - m + 2, & \text{for odd } K + m - 1 \end{cases}.
\end{aligned}
\tag{4.9}
$$

While the length of the placeholder attributes L can be derived based on the equality of the length of wavelet decomposition vector $\boldsymbol{c}$, i.e., $P + M + N + K = E + L$. Then the wavelet decomposition vector $\boldsymbol{c}$ and bookkeeping vector $\boldsymbol{l}$ are constituted as:

$$
\begin{aligned}
\boldsymbol{c} &= \left(\boldsymbol{x}_{\xi C}, \boldsymbol{x}_3, \boldsymbol{x}_2, \boldsymbol{x}_1\right), \\
\boldsymbol{l} &= (P, M, N, K, O),
\end{aligned}
\tag{4.10}
$$

where, O is the transformed dimension computed with:

$$
O = \begin{cases} 2K - m + 1, & \text{for even } O + m - 1 \\ 2K - m + 2, & \text{for odd } O + m - 1 \end{cases}.
\tag{4.11}
$$

3) Step 6-8: IDWT is performed on each sample of the normalized dataset, and finally a new

dataset defined on $\mathbb{A}' = \{b_1, b_2, \ldots, b_i, \ldots, b_O\}$ is generated. b_i is the ith new attribute.

4.3 Experiments and Results

To evaluate the performance of the proposed FDIDWT method, an experiment was set up to classify the blocks of the page layout of a document that has been detected by a segmentation process. The dataset used in the experiment comes from the UCI machine learning repository (Dua and Graff, 2017). It contains 5473 samples and each sample concerns one of five block classes[2], i.e., 1) text, 2) horizontal line, 3) picture, 4) vertical line, and 5) graphic. The 5473 samples were split into 3829 samples for training and 1644 for testing. There are 10 attributes in the dataset, i.e., $\mathbb{A} = \{a_0, a_1, \ldots, a_9\}$ and $E = 10$. Descriptions for these attributes are given in Tab. 4.2.

Table 4.2: Descriptions for the 10 attributes of the page blocks dataset.

Index	Attribute	Description
[a0]	height	height of the block
[a1]	length	length of the block
[a2]	area	area of the block (height × length)
[a3]	eccen	eccentricity of the block (length / height)
[a4]	p_black	percentage of black pixels within the block (blackpix / area)
[a5]	p_and	percentage of black pixels after the application of the Run Length Smoothing Algorithm (RLSA) (blackand / area)
[a6]	mean_tr	mean number of white-black transitions (blackpix / wb_trans)
[a7]	blackpix	total number of black pixels in the original bitmap of the block
[a8]	blackand	total number of black pixels in the bitmap of the block after the RLSA
[a9]	wb_trans	number of white-black transitions in the original bitmap of the block

4.3.1 Data Preparation

According to the algorithm described in Tab. 4.1, steps 1-5 of the FDIDWT method are performed on the training dataset to obtain the arranged attributes set $\mathbb{B}$, and construct the wavelet decomposition vector $\boldsymbol{c}$ and bookkeeping vector $\boldsymbol{l}$. In the experiment, at first the training data was fed into *FD-ASE* algorithm and the ASC $\xi C = \{a_8, a_7, a_6\}$ was returned after given correlation threshold $\xi = 0.45$ and scale range $n = 20$. The remaining attributes were then sorted to be $\mathbb{A} - \xi C = \{a_5, a_1, a_4, a_9, a_0, a_2, a_3\}$ in their ascending information contents order, as sketched in Fig. 4.4.

[2]The dataset was downloaded from: http://archive.ics.uci.edu/ml/datasets/Page+Blocks+Classification

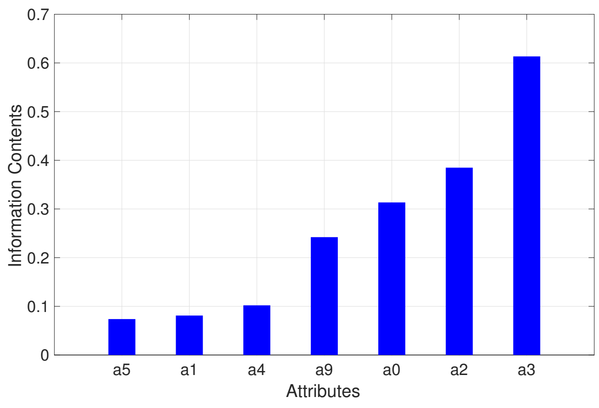


Figure 4.4: Ascending information contents of the remaining attributes of the page blocks dataset.

Then the arranged attributes set $\mathbb{B} = \{a_8, a_7, a_6, a_5, a_1, a_4, a_9, a_0, a_2, a_3\}$ were divided into groups for constituting $\boldsymbol{c}$ and $\boldsymbol{l}$. For comparison, three decomposition levels $J = 1, 2, 3$ and filters with different possible lengths are tested independently. How to constitute the vectors $\boldsymbol{c}$ and $\boldsymbol{l}$ with the arranged attributes of $\mathbb{B}$ is the focus of the FDIDWT method, so the constitution processes based on different decomposition levels are detailed next:

1) $J = 1$:

In this case, the attributes of $\mathbb{B}$ were simply divided into two groups, i.e., $CG_1^5 = \{a_8, a_7, a_6, a_5, a_1\}$ and $DG_1^5 = \{a_4, a_9, a_0, a_2, a_3\}$. There is not any special requirement for the filter, thus filters with length $m = 2, 4, 6, 8$ were tested. Thus, the length of the transformed sample, i.e., the transformed dimension O, could be obtained with $O = 11 - m$ or $12 - m$. If denote x_i as the value of attribute a_i for a particular sample in the dataset, then the $\boldsymbol{c}$ and $\boldsymbol{l}$ were written as:

$$\begin{aligned} \boldsymbol{c} &= \big(\{x_8, x_7, x_6, x_5, x_1\}, \{x_4, x_9, x_0, x_2, x_3\}\big), \\ \boldsymbol{l} &= (5, 5, 11 - m) \text{ or } (5, 5, 12 - m). \end{aligned} \tag{4.12}$$

2) $J = 2$:

Considering the number of attributes in ASC is $P = 3$ and the original dimension $E = 10$, fix the length of the detail coefficients at the first decomposition level to 4 seems reasonable, which means:

$$\begin{aligned} \boldsymbol{c} &= \big(\{x_8, x_7, x_6\}, \{x_5, x_1, x_4\}, \{x_9, x_0, x_2, x_3\}\big), \\ \boldsymbol{l} &= (3, 3, 4, 9 - m) \text{ or } (3, 3, 4, 10 - m). \end{aligned} \tag{4.13}$$

Then the length of the filter would be $m = 4$, which resulted in $O = 5$ or 6.

3) $J = 3$:

In this case, placeholder attributes should be considered since the length of the wavelet decomposition vector $\boldsymbol{c}$ would be longer than the original dimension E. Here, as shown in Eqn. (4.9), the length of the detail coefficients at the second decomposition level was fixed to $N = 4$, which led to $m = 4$. Based on it, there would be $K = 5$ or 6. According to $P + M + N + K = E + L$, when $K = 5$, the number of the placeholder attributes would be $L = 5$ and the transformed dimension $O = 7$ or 8; while if $K = 6$, there would be $L = 6$ and $O = 9$ or 10. Then $\boldsymbol{c}$ and $\boldsymbol{l}$ were written as:

$$\begin{aligned} &K = 5, \\ &\boldsymbol{c} = \big(\{x_8, x_7, x_6\}, \{0, 0, 0\}, \{0, 0, x_5, x_1\}, \{x_4, x_9, x_0, x_2, x_3\}\big), \\ &\boldsymbol{l} = (3, 3, 4, 5, 7) \text{ or } (3, 3, 4, 5, 8). \end{aligned} \tag{4.14}$$

or

$$\begin{aligned} &K = 6, \\ &\boldsymbol{c} = \big(\{x_8, x_7, x_6\}, \{0, 0, 0\}, \{0, 0, 0, x_5\}, \{x_1, x_4, x_9, x_0, x_2, x_3\}\big), \\ &\boldsymbol{l} = (3, 3, 4, 6, 9) \text{ or } (3, 3, 4, 6, 10). \end{aligned} \tag{4.15}$$

With the constructed wavelet decomposition vector $\boldsymbol{c}$ and bookkeeping vector $\boldsymbol{l}$, IDWT was performed on each sample of the normalized training and test datasets. New datasets were generated in different cases, as summarized in Tab. 4.3. In this experiment, Daubechies' filters were used to perform IDWT, thus the filter chosen in each case can be identified with its filter length m. For example, the filter length $m = 8$ indicates that the "db4" wavelet filter was used.

4.3.2 Model Construction

In this experiment, two models, i.e., Random Forest (RF) and Multilayer Perceptron (MLP), were built to accomplish the classification task based both on the original datasets and the transformed datasets for comparison. The number of units for each hidden layer and the hyper-parameters of these models can be found in Tab. 4.4. The experiments were set up on the Ubuntu 18.04 system. To speed up the computing, RF programs were distributed on multi-CPUs with scikit-learn (version 0.20) (Pedregosa et al., 2011). While MLP models were accelerated by GPU with TensorFlow (version 1.10). In the experiment, the CPU was Intel Core i5-8400 and the GPU was NVIDIA GeForce GTX1050Ti with 768 CUDA cores.

Table 4.3: Wavelet decomposition vectors and bookkeeping vectors tested in the page blocks classification experiments.

J	$\mathbf{c}$	$\mathbf{l}$	m	Cases
1	$\big(\{x_8, x_7, x_6, x_5, x_1\}, \{x_4, x_9, x_0, x_2, x_3\}\big)$	$(5,5,3)$	8	#1
		$(5,5,5)$	6	#2
		$(5,5,7)$	4	#3
		$(5,5,9)$	2	#4
		$(5,5,4)$	8	#5
		$(5,5,6)$	6	#6
		$(5,5,8)$	4	#7
		$(5,5,10)$	2	#8
2	$\big(\{x_8, x_7, x_6\}, \{x_5, x_1, x_4\}, \{x_9, x_0, x_2, x_3\}\big)$	$(3,3,4,5)$	4	#9
		$(3,3,4,6)$	4	#10
3	$\big(\{x_8, x_7, x_6\}, \{0,0,0\}, \{0,0,x_5,x_1\}, \{x_4, x_9, x_0, x_2, x_3\}\big)$	$(3,3,4,5,7)$	4	#11
		$(3,3,4,5,8)$	4	#12
	$\big(\{x_8, x_7, x_6\}, \{0,0,0\}, \{0,0,0,x_5\}, \{x_1, x_4, x_9, x_0, x_2, x_3\}\big)$	$(3,3,4,6,9)$	4	#13
		$(3,3,4,6,10)$	4	#14

Table 4.4: The number of hidden units and the hyper-parameters of the models used in the page blocks classification experiments.

Models	Number of hidden units	Hyper-parameters
RF		criterion: entropy number of trees: 5 maximum depth: 3
MLP	[20 10 10]	batch size: 128 epochs: 300 optimer: Adam learning rate: $5e^{-5}$ activation function: ReLU loss function: Cross Entropy

4.3.3 Results and Analysis

The accuracy results on the test datasets in different cases are compared in Tab. 4.5. In the table, case "#0" indicates the results on the original test dataset, and others are the results on other transformed test datasets. For each model, the accuracies that higher than case "#0" are in bold.

Table 4.5: Accuracy results on the test datasets in different cases with RF and MLP models.

J	O	Cases	Models	
			RF	MLP
	10	#0	92.34%	92.58%
1	3	#1	89.17%	91.42%
	5	#2	90.63%	90.45%
	7	#3	**92.34%**	**93.37%**
	9	#4	91.91%	**93.19%**
	4	#5	91.97%	**94.22%**
	6	#6	91.00%	**92.94%**
	8	#7	91.30%	**93.07%**
	10	#8	90.51%	**93.49%**
2	5	#9	73.30%	92.40%
	6	#10	80.66%	92.58%
3	7	#11	92.15%	92.46%
	8	#12	90.88%	**92.82%**
	9	#13	**92.82%**	**94.83%**
	10	#14	**93.43%**	**94.28%**

From the results, it can be found that the MLP model got a higher accuracy score than the RF model in almost every case. It is also noted that, a high accuracy score was obtained from the MLP model in case #5 ($\sim$ 1.6% higher than in case #0) with a considerable low transformed dimension ($O = 4$), which means that the proposed method was able to reduce the original dimension without losing too much information.

When comparing the results obtained from the transformed dataset, the RF model performed better when setting $J = 3$. For the MLP model, the suitable decomposition levels were $J = 1$ and 3. For a particular model, different decomposition levels led to different results. Even given a decomposition level, different transformed dimensions produced different results. There were also some cases where the model performances on the transformed datasets were worse than on the original dataset. These results indeed validated the assumption that the key features of data may be difficult to be discovered at the original resolution, but they will be extracted at other resolutions. For example, when using the RF model, there is a high possibility that the key features of the experimental data exist at a resolution three-higher than the original resolution. But it is not a certain answer since the MLP model also performed well when $J = 1$. Even though these inconsistent results might come from the different models used, observing the MLP learning process may be helpful to give more information.

To further study the performance of the MLP model combined with the FDIDWT method, the loss values of the model on the transformed training datasets were averaged based on the decomposition levels, as shown in Fig. 4.5. The figure shows that the MLP model converged faster when $J = 3$. Together with the accuracy results, now it is more likely to conclude that the key features of the experimental data exist at a resolution three-higher than the original resolution.

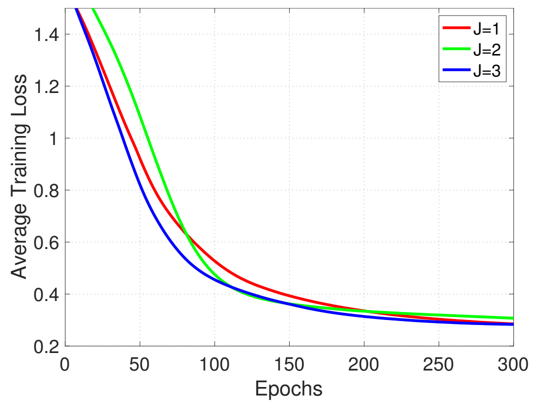


Figure 4.5: Average training loss values of the MLP model with different decomposition levels J.

In particular, the training loss values of the MLP model when setting $J = 3$ are illustrated in Fig. 4.6. The results indicated that the training process can be seen as convergent after around 150 epochs, and case #13 and #14 achieved better performance. Even though the case #14 converged to a sighly lower level, the MLP model learnt features faster at the beginning of the training process in case #13, which is significant to save time in deep learning. Meanwhile, case #13 got the highest accuracy score when given the decomposition level $J = 3$, which can be found in Tab. 4.5. Additionally, the transformed dimension of case #13 is $O = 9$, which decreases the computational complexity in part. Anyway, case #0 had the slowest convergence speed, which means that the FDIDWT method indeed accelerated training.

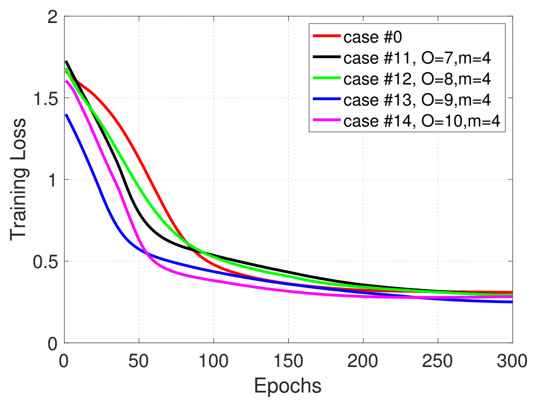


Figure 4.6: Training loss values of the MLP model with a given decomposition level $J = 3$.

4.4 Discussions

Based on wavelet analysis, the information contents of wavelet coefficients vary from resolutions. While at the same resolution, the approximation coefficients have more information than the detail coefficients. Then the wavelet coefficients at different resolutions roughly respect an order according to their information contents.

The information content of an attribute in big data can be given by estimating its correlation with ASC that is the smallest subset of attributes to characterize the whole dataset. Based on it, the attributes in a dataset can also be sorted in the same order to the wavelet case.

In fact, the goal of sorting the attributes is to perform IDWT on the original dataset and produce a new dataset. The consideration of applying IDWT instead of DWT comes from the fact that there exist correlations among attributes. These correlations can also be taken as the features of data. While if performing DWT, which is the common solution of Loose Wavelet Neural Network (LWNN, introduced in section 2.4.3), the correlation features are neglected since the data under the attributes is organized randomly.

One may assume to sort the attributes like the proposed method does, but then perform DWT to transform the original data to be the representations at a lower resolution. While in this case, the inherent features of data may be lost. The representations of signal in wavelet domain will be incorrect and they may not contain the right information of the original data. For example, the order of the adjacent points is important for a discrete speech signal. The information hidden in the speech signal can be completely lost if the order of some points is disturbed. Another more intuitive example is image data. In principle, the values of the adjacent pixels in an image taken from the nature change gently. Some key features such as textures or edges are usually detected from the discriminative changes of adjacent pixel values. If pixels are rearranged, the image may become another one.

Therefore, it is difficult for DWT to get the correlation features and inherent features of data at the same time. The solution is to replacing DWT with IDWT. Actually, DWT can be seen as a bandpass filter that arranges the information of the original data to different resolutions. The quantities of the information vary from coefficients, as found in Eqn. (4.3). However, these varieties can be seen as the correlation features of data. Thus, it is assumed that the original data is taken as the wavelet coefficients at some resolution. After sorting the attributes to get the correlation features, the data is then transformed to a higher resolution through IDWT. The lost inherent features due to the arrangement of attributes are considered to be retrieved by the IDWT. At the higher resolution, the key features will be extracted. In addition, the transformed attribute space will not be larger than the original one, which is significant to reduce dimensionality.

The experiment demonstrated that the proposed FDIDWT method helped ML models to extract more features and accelerated MLP training. This preprocessing method would not increase considerable computational complexity due to the efficient implementation of IDWT, i.e., the pyramid transform. However, there should be some practical considerations to be discussed. The first one is that there is no general rule to choose the hyper-parameters J and O. A simple method is to use different J and O to construct the alternative wavelet decomposition vector $\boldsymbol{c}$ and the corresponding bookkeeping vector $\boldsymbol{l}$ based on the original training dataset, as shown in Tab. 4.3. Then the convergence of models with different vector pairs ($\boldsymbol{c}$ and $\boldsymbol{l}$) are compared on the transformed training datasets, as shown in Fig. 4.5 and 4.6. The vector pair that leads to the fastest convergence is taken as the choice, which determines J and O. Moreover, a lower transformed dimension O is also preferable.

The second consideration is that the performance of the model should be analyzed from two aspects: the data preprocessing method and the model used. For example, in the experiment, the RF model performed well with $J = 3$, while the MLP model also achieved good performance when $J = 1$. Therefore, the model performance depends on two factors: the feature pre-extraction capability of the preprocessing method and the learning capability of the designed model. For the FDIDWT method, it cannot be concluded that the features extracted at a particular resolution always help any model to achieve better performance. But it also indicates that a well-designed model gives a possibility to get better results. For instance, in the experiment, the MLP model performed better than the RF model in almost every case. Thus, the optimization of model is another work of this thesis, which will be described in next chapter.

5 Tight Frame Wavelet Functions for Fast Deep Learning

Gradient vanishing or exploding problem and slow convergence speed are the key issues when training Deep Neural Networks (DNN). This chapter investigates these issues from the perspective of Activation Function and Derivative Function pair (ADF), and summarizes the properties of an ideal ADF for fast DNN training. A tight frame wavelet solution is proposed to alleviate the gradient problems and speed up the training process. Experiments demonstrate that the proposed method helps to stabilize DNN training and accelerate convergence. The tight frame wavelet functions can be a better solution to different applications.

5.1 Activation Function and Derivative Function in Deep Learning

To tackle the gradient vanishing/exploding problem and to accelerate the convergence of DNN, many efforts have been made to design better activation functions (Xu et al., 2015). The difficulties in DNN training can greatly be attributed to the ADFs inappropriately chosen. Seven widely used ADFs are listed in Tab. 5.1 and illustrated in Fig. 5.1. They can be analyzed considering two aspects: *zero-mean function* and *energy transfer*.

Table 5.1: Commonly used activation functions and the corresponding derivative functions.

Name	Activation Function	Derivative Function
Sigmoid	$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
ReLU	$f(x) = \begin{cases} x, x \geq 0 \\ 0, x < 0 \end{cases}$	$f'(x) = \begin{cases} 1, x \geq 0 \\ 0, x < 0 \end{cases}$
ELU	$f(x) = \begin{cases} x, x \geq 0 \\ e^x - 1, x < 0 \end{cases}$	$f'(x) = \begin{cases} 1, x \geq 0 \\ e^x, x < 0 \end{cases}$
Leaky ReLU	$f(x) = \begin{cases} x, x \geq 0 \\ Kx, x < 0 \end{cases}$	$f'(x) = \begin{cases} 1, x \geq 0 \\ K, x < 0 \end{cases}$
Tanh	$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	$f'(x) = 1 - (f(x))^2$
Softplus	$f(x) = \log(1 + e^x)$	$f'(x) = \frac{e^x}{1+e^x}$
Softsign	$f(x) = \frac{x}{1+\lvert x \rvert}$	$f'(x) = \frac{1}{(1+\lvert x \rvert)^2}$

5.1.1 Zero-mean Function

Zero-mean function indicates a function that normalizes the distribution of its outputs at near-zero mean. Based on the definition, the functions in Tab. 5.1 can be categorized into two types:

- Type I: the activation function is a zero-mean function while the derivative function isn't, such as ELU, Leaky ReLU (LeReLU), Tanh, and Softsign;
- Type II: neither the activation function nor the derivative function is zero-mean function, such as Sigmoid, ReLU, and Softplus.

In a Forward Propagation (FP) process, the distributions of layer activations of type I function will be well normalized and the Internal Covariate Shift (ICS) will be small during training. On the contrary, their error distributions in Backward Propagation (BP) process are not normalized and always change because of their positive derivative values. This effect is named as Backward Internal Covariate Shift (BICS), which increases when training DNN. The increased BICS will push the mean of error distributions to the saturation regions for Tanh and Softsign, which will cause the gradient vanishing problem. This means that some of the trainable parameters cannot be updated as normal and the network may converge to a poor local minimum (Glorot and Bengio, 2010; Maas et al., 2013).

Both the ICS and BICS of a type II function exist, since the distributions of activations and errors are not well normalized. During the training process, the distribution of each layer

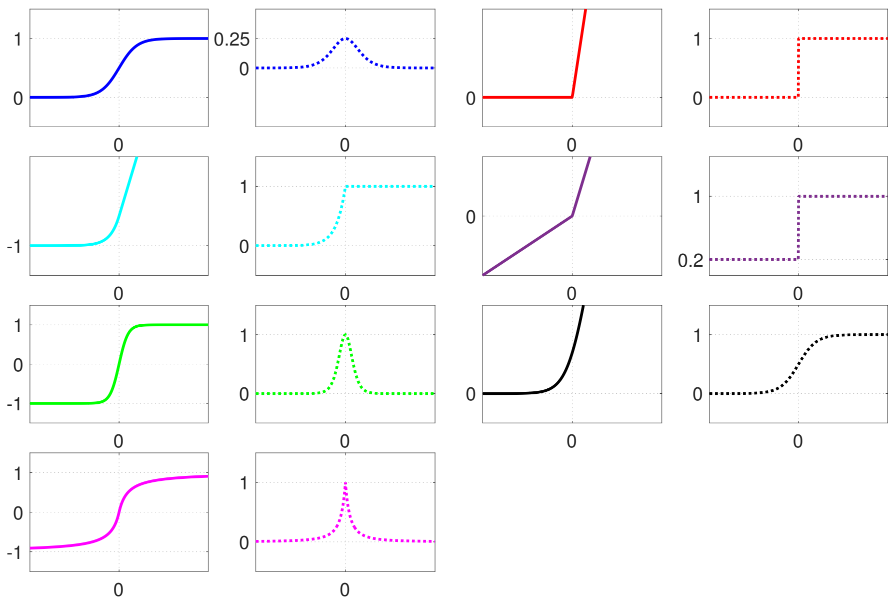


Figure 5.1: Commonly used activation functions (solid lines) and the corresponding derivative functions (dashed lines). The blue, red, cyan, purple, green, black, and magenta colors respectively represent the Sigmoid, ReLU, ELU, LeReLU, Tanh, Softplus, and Softsign functions.

data (activations and errors) changes according to the parameters of the preceding layers. If the data of one hidden layer has a nonzero-mean distribution, it will cause ICS (in FP process) or BICS (in BP process) for units in the next hidden layer. Therefore, the next layer needs to continuously adapt to the new distribution; this will be time-consuming, slowing down the convergence. The BICS also easily results in the gradient vanishing problem in Sigmoid case.

As discussed, the ReLU family (ReLU, ELU, and LeReLU) may not easily suffer from the gradient vanishing problem, but if some hyper-parameters, such as the learning rate, are not carefully tuned, that can give rise to the gradient exploding problem. This defect is attributed to the nonzero-mean property of the ReLU family. Another vision of the gradient exploding problem comes from the energy transfer in the training process that will be discussed below.

5.1.2 Constant Energy Transfer

With respect to the continuous input x, the characteristic energy E of a function $f(x)$ can be expressed as:

$$E = \int_{-\infty}^{+\infty} |f(x)|^2 \, dx < \infty. \tag{5.1}$$

As a consequence of this definition, all the activation functions listed in Tab. 5.1 are infinite-energy functions, where the energies of activations are not restricted: this leads to a slow convergence. While the derivative functions of ReLU family and Softplus are also infinite-energy functions, resulting in potential numerical problems or gradient exploding problem for the active units (Glorot et al., 2011).

Besides, ReLU has another problem, i.e., the so-called "dead units" (Maas et al., 2013). When a ReLU function is used and a gradient-based optimization algorithm is applied to train the DNN model, and the gradients are very large or the learning rate is set a little bit high, then the weights will become very small and the corresponding units will be dead. The dead units will not revive anymore. The weights of such dead units will never be updated. If too many dead units occur, the model performance will be degraded.

Hence, there is a need to design a modest finite-energy ADF aiming to form a stable energy transfer both in FP and BP processes. The energies of propagating activations and errors are expected to range within definite bounds, so that neither gradient vanishing nor gradient exploding problem should come up. Ideally, if the energies of the inputs and outputs of one layer are equal, the training will be strongly accelerated. This case is referred to as *constant energy transfer*, where both learning rate and weight initialization will no longer be the key factors affecting the training process.

From the above analysis, it can be seen that the choice of ADF plays a fundamental role in DNN training. While Sigmoid, Tanh, and Softsign can easily suffer from gradient vanishing problem in deep structures, Softplus and ReLU family functions have the potential numerical problem. They may perform well in some tasks, but a universal solution is needed to solve the problems discussed in the deep learning field.

Moreover, all the seven commonly used ADFs are linear or approximately-linear, which results in a weak learning capability of model and slows down training. Also, most of these ADFs are not very sparse, i.e., output few zeros or near-zeros, which cannot guarantee the sparsity of propagating data in DNN training. Thus, they may easily cause overfitting problem.

In summary, an ideal ADF for fast DNN training should have the following properties:

1) Zero-mean: It is zero-mean function for alleviating gradient vanishing and exploding problems;

2) Constant energy transfer: It constrains the energies of layer data to a constant both in the FP and BP processes for overcoming the ICS and BICS and accelerating training;

3) Nonlinear: It should have good nonlinearity for extracting features in data and handling underfitting problem.

4) Sparse: It keeps the sparsity of propagating data for reducing overfitting problem and resulting in robust model.

5.2 Analysis of Wavelet Functions

As an example shown in Fig. 5.2, wavelet $\psi(t)$ is defined as a "small wave" which has oscillating characteristic and finite energy (Sidney Burrus et al., 1998). Its output will thus be well normalized, and it has near-zero mean and small variance value. Besides, wavelet also has the nonlinear property that is good for handling complex problems when used as the ADF. Moreover, DNN training benefits more from the sparsity of the wavelet and a tight frame wavelet featuring constant energy transfer, which will be discussed next.

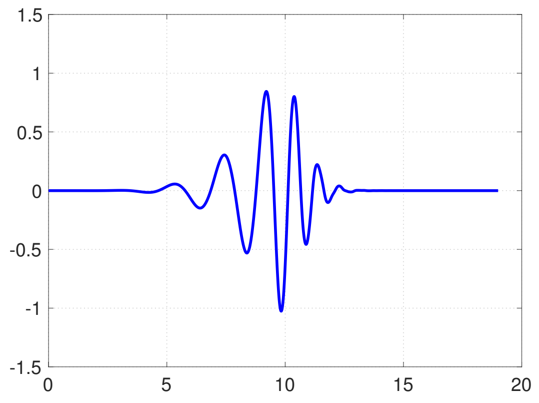


Figure 5.2: Daubechies' wavelet ψ_{D10}.

5.2.1 Sparsity of Wavelet Function

The sparsity of wavelet function indicates a small number of coefficients in the wavelet domain by Wavelet Transform (WT). These coefficients are sufficient to represent the original data. This property helps to reduce the computational complexity and the hardware resources needed. In fact, the matrix multiplications of Multilayer Perceptron (MLP) training process can be seen as WT, which is defined as the inner product of the signal $f(t)$ with the two-parameter family $\psi_{a,b}(t)$ introduced in Eqn. (2.44):

$$\mathrm{WT}(a,b) = \int_{-\infty}^{+\infty} f(t)\bar{\psi}_{a,b}(t)dx = \langle f(t), \psi_{a,b}(t) \rangle, \tag{5.2}$$

where, $\bar{\psi}$ denotes the complex conjugate of ψ. In the FP process of MLP training, as expressed in Eqn. (2.11), if $\psi_{a,b}(\cdot)$ is used as the activation function and the bias are set to zero, the logits data of the lth hidden layer will be:

$$\begin{aligned} \boldsymbol{z}^l &= \boldsymbol{a}^{l-1}\mathbf{W}^l \\ &= \psi_{a,b}\left(\boldsymbol{z}^{l-1}\right)\mathbf{W}^l \\ &= \left\langle \psi_{a,b}\left(\boldsymbol{z}^{l-1}\right), \mathbf{W}^l \right\rangle, \end{aligned} \tag{5.3}$$

where, $\boldsymbol{a}$ and $\mathbf{W}$ are the activations and weights, respectively. $\boldsymbol{z}^l$ is the sparse logits data of the lth hidden layer and it will propagate to next layers.

From Eqn. (5.3) it can be seen that the FP process will largely benefit from the sparsity property of wavelet function $\psi_{a,b}$. The same advantage will also be seen in the BP process since the sparse logits data $\boldsymbol{z}^l$ is one of the factors to calculate error data $\boldsymbol{\delta}^l$, as shown in Eqn. (2.14b). The propagating data of FP and BP processes in Convolutional Neural Network (CNN) training has the same benefits.

The sparsity of wavelet function will help to further accelerate training, reducing the use of hardware resources when training DNN in embedded systems. In addition, it may strengthen the robustness of the model to noisy inputs (Glorot et al., 2011) and alleviate the overfitting problem.

5.2.2 Constant Energy Transfer of Tight Frame Wavelets

Based on the Plancherel theorem, the energy of the discretized wavelet family $\psi_{m,n}(t)$ defined in Eqn. (2.47) is computed as:

$$\begin{aligned} E &= \int_{-\infty}^{+\infty} \left|\psi_{m,n}(t)\right|^2 dt \\ &= \int_{-\infty}^{+\infty} \left|a_0^{-m/2} e^{-jb_0 a_0^m n\omega} \hat{\psi}(a_0^m \omega)\right|^2 d\omega \\ &= a_0^m \int_{-\infty}^{+\infty} \left|\hat{\psi}(a_0^m \omega)\right|^2 d\omega. \end{aligned} \tag{5.4}$$

If $m = 1$, then the energy of the selected wavelet function $\psi_{m=1,n}$ will be:

$$E' = a_0 \int_{-\infty}^{+\infty} \left|\hat{\psi}(a_0 \omega)\right|^2 d\omega. \tag{5.5}$$

And if $\psi_{m,n}$ constitutes a frame with bounds A and B, then Daubechies (1992) gives:

$$A \le \frac{2\pi}{b_0} \sum_{m\in\mathbb{Z}} \left|\hat{\psi}\left(a_0^m \omega\right)\right|^2 \le B, \tag{5.6}$$

for all $\omega \neq 0$. So when $m = 1$, there exist:

$$A \leq \frac{2\pi}{b_0} \left|\hat{\psi}(a_0\omega)\right|^2 \leq B. \tag{5.7}$$

By integrating Eqn. (5.7) with $1 \leq |\omega| \leq a_0$, then:

$$\frac{Ab_0}{2\pi} \int d\omega \leq \int \left|\hat{\psi}(a_0\omega)\right|^2 d\omega \leq \frac{Bb_0}{2\pi} \int d\omega. \tag{5.8}$$

Combining Eqn. (5.5) and (5.8), finally:

$$A\frac{a_0 b_0(a_0 - 1)}{\pi} \leq E' \leq B\frac{a_0 b_0(a_0 - 1)}{\pi}. \tag{5.9}$$

It can be found that the energy of a particular wavelet frame ranges within fixed bounds determined by the lower and upper frame bound, i.e., A and B, respectively. Moreover, if the wavelet frame is tight, which means $A = B$, then

$$E' = A\frac{a_0 b_0(a_0 - 1)}{\pi} = B\frac{a_0 b_0(a_0 - 1)}{\pi}. \tag{5.10}$$

Eqn. (5.10) implies that the energy of a tight frame wavelet is a constant determined by its frame bounds. In other words, if a tight frame wavelet is used as the activation function or derivative function in DNN training, the energy of propagating data will not diverge, which is ideal and means constant energy transfer. However, the data energy ranges within bounds in the "slack" frame case, leading to slow convergence of training. In any case, the seven ADFs listed in Tab. 5.1 will not have a good performance on energy transfer to accelerate training.

5.3 ADFs of Tight Frame Wavelet

This section derives the tight frame wavelet functions for further experimental verifications, i.e., Tight Frame Wavelet Activation Function (TFWAF) for FP process and Tight Frame Wavelet Derivative Function (TFWDF) for BP process. The frames of Mexican hat wavelet and Morlet wavelet will be constructed for comparison.

5.3.1 Mexican Hat Case: TFMH

Mexican hat wavelet is the second derivative of the Gaussian $e^{-x^2/2}$ and its normalized form can be written as:

$$\psi_{mexh}(x) = \frac{2}{\sqrt{3}\sqrt[4]{\pi}}\left(1 - x^2\right)e^{-x^2/2}. \tag{5.11}$$

Eqn. (2.50) gives the frame bounds A and B of $\psi_{mexh}(x)$, as listed in Tab. 5.2, with different a_0 and b_0 values. From the table, it can be found that there are many situations where the Mexican hat wavelet has a tight frame. If choosing $a_0 = \sqrt[4]{2}$, $b_0 = 0.25$, $m = 1$, $n = 1$, then the TFWAF in Mexican hat case can be written as:

$$f_{TFMH}(x) = \frac{2^{7/8}}{\sqrt{3}\sqrt[4]{\pi}}\left(1-\left(\frac{1}{\sqrt[4]{2}}x-\frac{1}{4}\right)^2\right)e^{-\frac{1}{2}\left(\frac{1}{\sqrt[4]{2}}x-\frac{1}{4}\right)^2}. \tag{5.12}$$

When the BP algorithm is applied to train DNN models, the derivative function is needed to compute gradients. The derivative of $f_{TFMH}(x)$ is:

$$\begin{aligned} g(x) &= f'_{TFMH}(x) \\ &= \frac{\left(1-2^{7/4}x\right)\left(4\sqrt{2}x\left(\sqrt[4]{2}-2x\right)+47\right)}{2^{43/8}\sqrt{3}\pi^{1/4}}e^{-\frac{1}{32}\left(1-2^{7/4}x\right)^2}. \end{aligned} \tag{5.13}$$

Table 5.2: Frame bounds for wavelet frames based on the Mexican hat wavelet.

a0	b0	A	B	B/A
2	0.25	13.091	14.183	1.083
	0.50	6.546	7.092	1.083
	0.75	4.364	4.728	1.083
	1.00	3.223	3.596	1.116
$\sqrt{2}$	0.25	27.273	27.278	1.0002
	0.50	13.673	13.639	1.0002
	0.75	9.091	9.093	1.0002
	1.00	6.768	6.870	1.015
$\sqrt[3]{2}$	0.25	40.914	40.914	1.0000
	0.50	20.457	20.457	1.0000
	0.75	13.638	13.638	1.0000
	1.00	10.178	10.279	1.010
$\sqrt[4]{2}$	0.25	54.552	54.552	1.0000
	0.50	27.276	27.276	1.0000
	0.75	18.184	18.184	1.0000
	1.00	13.586	13.690	1.007

However, in order to constrain the energy of errors in BP process, the TFWDF is expected. Then the frame bounds of $g(x)$ are calculated and shown in Tab. 5.3. When choosing

$a_0 = \sqrt[4]{2}$, $b_0 = 0.25$, $m = 1$, $n = 1$, the TFWDF in Mexican hat case is written as:

$$g_{TFMH}(x) = -e^{-\frac{1}{128}\left(-4\sqrt{2}x+2^{3/4}+2\right)^2}\left(-4\sqrt{2}x+2^{3/4}+2\right) \frac{16x^2-8\left(\sqrt{2}+\sqrt[4]{2}\right)x+2^{7/4}+\sqrt{2}-94}{128\sqrt{6}\sqrt[4]{\pi}}. \tag{5.14}$$

The TFWAF (f_{TFMH}) and TFWDF (g_{TFMH}) in Mexican hat case are called TFMH, and they are illustrated in Fig. 5.3. The mean and variance of $f_{TFMH}(x)$ are 0 and 0.05, respectively; while for $g_{TFMH}(x)$ they are 0 and 0.088, respectively. These values indicate the normalization capacity of TFMH.

Table 5.3: Frame bounds for wavelet frames based on the derivative function of tight frame Mexican hat wavelet.

a0	b0	A	B	B/A
2	0.25	20.6493	25.2042	1.2206
	0.50	10.3246	12.6021	1.2206
	0.75	6.8831	8.4014	1.2206
	1.00	5.1433	6.3200	1.2288
$\sqrt{2}$	0.25	45.8531	45.8915	1.0008
	0.50	22.9266	22.9458	1.0008
	0.75	15.2844	15.2972	1.0008
	1.00	11.4385	11.4976	1.0052
$\sqrt[3]{2}$	0.25	68.8084	68.8085	1.0000
	0.50	34.4042	34.4043	1.0000
	0.75	22.9361	22.9362	1.0000
	1.00	17.1662	17.2381	1.0042
$\sqrt[4]{2}$	0.25	91.7446	91.7446	1.0000
	0.50	45.8723	45.8723	1.0000
	0.75	30.5815	30.5816	1.0000
	1.00	22.8883	22.9840	1.0042

5.3.2 Morlet Case

The Morlet wavelet is defined as:

$$\psi_{morlet}(x) = \cos(5x)e^{-x^2/2}. \tag{5.15}$$

The Morlet case is proceeded similarly to what previously done; its frame bounds are listed in Tab. 5.4.

The results show that Morlet wavelet doesn't have a tight frame, but a comparatively-

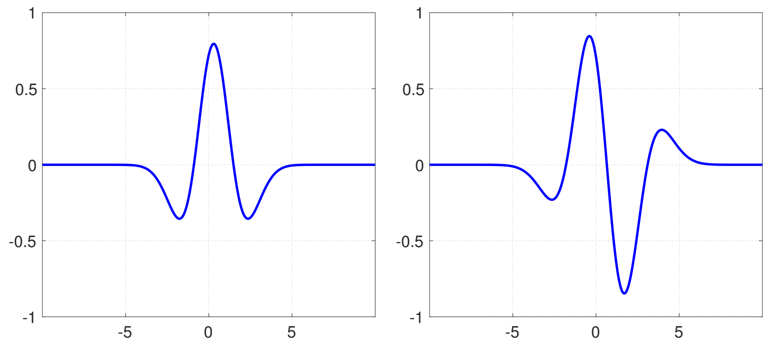


Figure 5.3: The TFWAF (f_{TFMH}, left) and TFWDF (g_{TFMH}, right) in Mexican hat case.

Table 5.4: Frame bounds for wavelet frames based on the Morlet wavelet.

a0	b0	A	B	B/A
	0.25	5.9369	7.1889	1.2109
$\sqrt{2}$	0.50	1.7130	4.8499	2.8312
	1.00	0.6831	2.5983	3.8037
	0.25	9.7977	9.8939	1.0098
$\sqrt[3]{2}$	0.50	3.3591	6.4867	1.9311
	1.00	1.4213	3.5016	2.4637
	0.25	13.1258	13.1297	1.0003
$\sqrt[4]{2}$	0.50	4.5622	8.5656	1.8775
	1.00	1.9370	4.6269	2.3887

tight frame can be obtained when $a_0 = \sqrt[4]{2}$, $b_0 = 0.25$. If assigning $m = 1$, $n = 1$, the comparatively-tight frame wavelet activation function in Morlet case is:

$$f_{morlet}(x) = 2^{-\frac{1}{8}} \cos\left(5\left(\frac{1}{\sqrt[4]{2}}x - \frac{1}{4}\right)\right) e^{-\frac{1}{2}\left(\frac{1}{\sqrt[4]{2}}x-\frac{1}{4}\right)^2}. \tag{5.16}$$

Then the derivative function of $f_{morlet}(x)$ is:

$$\begin{aligned} h(x) &= f'_{morlet}(x) \\ &= 2^{-\frac{21}{8}} e^{-\frac{1}{32}(1-2^{7/4}x)^2}\left(20\sqrt[4]{2}\sin\left(\frac{5}{4} - \frac{5}{\sqrt[4]{2}}x\right) + \right. \\ &\left. (\sqrt[4]{2} - 4x)\cos\left(\frac{5}{4} - \frac{5}{\sqrt[4]{2}}x\right)\right). \end{aligned} \tag{5.17}$$

However, $h(x)$ doesn't have a wavelet frame, so there isn't TFWDF in this case. For comparison, $f_{morlet}(x)$ and $h(x)$ are used as the ADF in Morlet case, as plotted in Fig. 5.4. The mean and variance of $f_{morlet}(x)$ are 0 and 0.04, respectively; for $h(x)$ they are 0 and 0.8, respectively.

Note that the magnitudes of $h(x)$ are not well normalized in $[-1, 1]$.

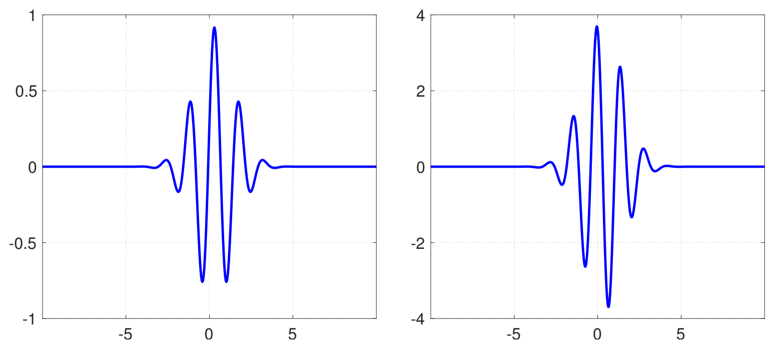


Figure 5.4: The comparatively-tight frame wavelet activation function (f_{morlet}, left) and its derivative function (h, right) in Morlet case.

5.4 Experiments and Results

Four experiments were performed to verify the proposed method. First, three MLP models were built to identify the synthetic nonlinear signals, approximate a single variable function, and approximate a real nonlinear signal independently. The training process of MLP models was accelerated by using Tensorflow 1.10 (GPU version). Second, a CNN model was built to accomplish the task of handwritten digits recognition on the MNIST dataset (LeCun et al., 1998). The CNN model was implemented with CUDA C on GPU. The graphic card was the NVDIA GeForce GTX1050Ti with 768 CUDA cores.

These experiments were running on the Ubuntu 18.04 system, with CPU Intel Core i5-8400. For comparison, in each experiment, six ADFs were tested independently: tight frame Mexican hat function (TFMH), comparatively tight frame Morlet function ("morlet"), Sigmoid function ("sigmoid"), and ReLU family ("relu" for ReLU, "lerelu" for Leaky ReLU, and "elu" for ELU).

In addition, to evaluate the stability of the MLP training process, a Jacobian matrix was often used to quantify the variation of the layer distributions of inputs and outputs (Glorot and Bengio, 2010). The Jacobian matrix associated with the lth layer is computed as:

$$\mathbf{J}^l = \frac{\partial \boldsymbol{a}^{l+1}}{\partial \boldsymbol{a}^l}, \tag{5.18}$$

where, $\boldsymbol{a}^l$ represents the activations of the lth layer. If the layer Jacobian matrix $\mathbf{J}$ has an average singular value close to 1, the layer activations will flow normally and the training is considered stable. In that case, the training process will converge quickly (Pascanu et al., 2013; Saxe et al., 2013).

5.4.1 Identification of Synthetic Nonlinear Signals

Understanding the critical features of nonlinear signals produced from different processes may help to identify a variety of nonlinear systems. In this experiment, three kinds of nonlinear processes were simulated: cosine, exponent and differential, and they can be expressed by:

$$\begin{cases} y_0 = \cos(40\pi x + 10) + 2\cos(400\pi x + 20) + \\ \quad 3\cos(1000\pi x + 30) + 5 + n \\ y_1 = 5e^{-5x} + 10 + n \\ y_2 = \text{Bandpass}\left(n'\right) \end{cases}, \tag{5.19}$$

where, x is an even sequence with interval 1 over [1, 1024]. n is the Gaussian noise with a standard normal distribution. The Signal-to-Noise Ratios (SNRs) of signal y_0 and y_1 are both set at 20dB. Bandpass($\cdot$) is a 10th-order bandpass Butterworth filter, with its amplitude-frequency and phase-frequency response characteristic curves depicted in Fig. 5.5. n' is also the Gaussian noise with mean 0 and variance 5.

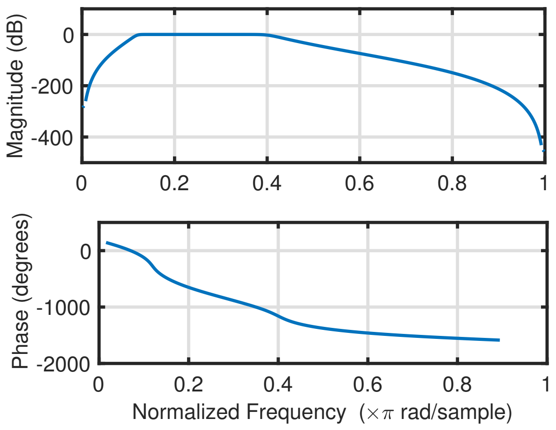


Figure 5.5: Amplitude-frequency (top) and phase-frequency (bottom) response characteristic curves of the 10th-order bandpass Butterworth filter.

The MLP models with different ADFs were independently trained to classify y_0, y_1, and y_2 into three distinct separate classes: 0, 1, and 2, respectively. In this experiment, 18000 samples were produced in 3 classes, with 6000 samples per class. Each sample has 1024 points. These samples were divided into 15000 for training and 3000 for testing. In the datasets, each class had the same number of samples, i.e., 5000 samples in training dataset and 1000 samples in test dataset.

Structures and Hyper-parameters

In this experiment, the performances of different ADFs for three MLP structures are compared. The number of layer units and the hyper-parameters for training can be found in Tab. 5.5.

Table 5.5: Structures and hyper-parameters of the MLP models in the synthetic nonlinear signal identification experiment.

three-hidden-layer 1	[1 5 3 3 1]
three-hidden-layer 2	[1 50 50 50 1]
ten-hidden-layer	[1 50 50 50 30 30 30 20 20 10 10 1]
epochs	50
learning rate	0.0001
weights initialization	Xavier
bias initialization	zero
optimizer	SGD
loss function	MSE

Results

The test loss values of the three MLP structures are compared in the left panels of Fig. 5.6. The results of Fig. 5.6(a) showed that the models with TFMH, morlet, and elu could converge to a low loss level, while the TFMH case achieved the fastest convergence. The lerelu and sigmoid cases converged to a suboptimal loss level and the relu case had the worst result, this can be considered as the underfitting problem.

When the MLP model became deeper, as comparing Fig. 5.6(c) with 5.6(e), the sigmoid case suffered from gradient vanishing problem and the training process of morlet case diverged. But TFMH and elu functions still had the best performance.

The training processes of the MLP models with TFMH and Relu family could be considered stable since their average singular values of hidden layer Jacobian matrices closed to 1, as depicted in the right panels of Fig. 5.6. While the average singular value of the morlet case was much higher than 1, which indicates the unstable data flow during training. The near-zero average singular value of the sigmoid case indeed confirmed the gradient vanishing problem.

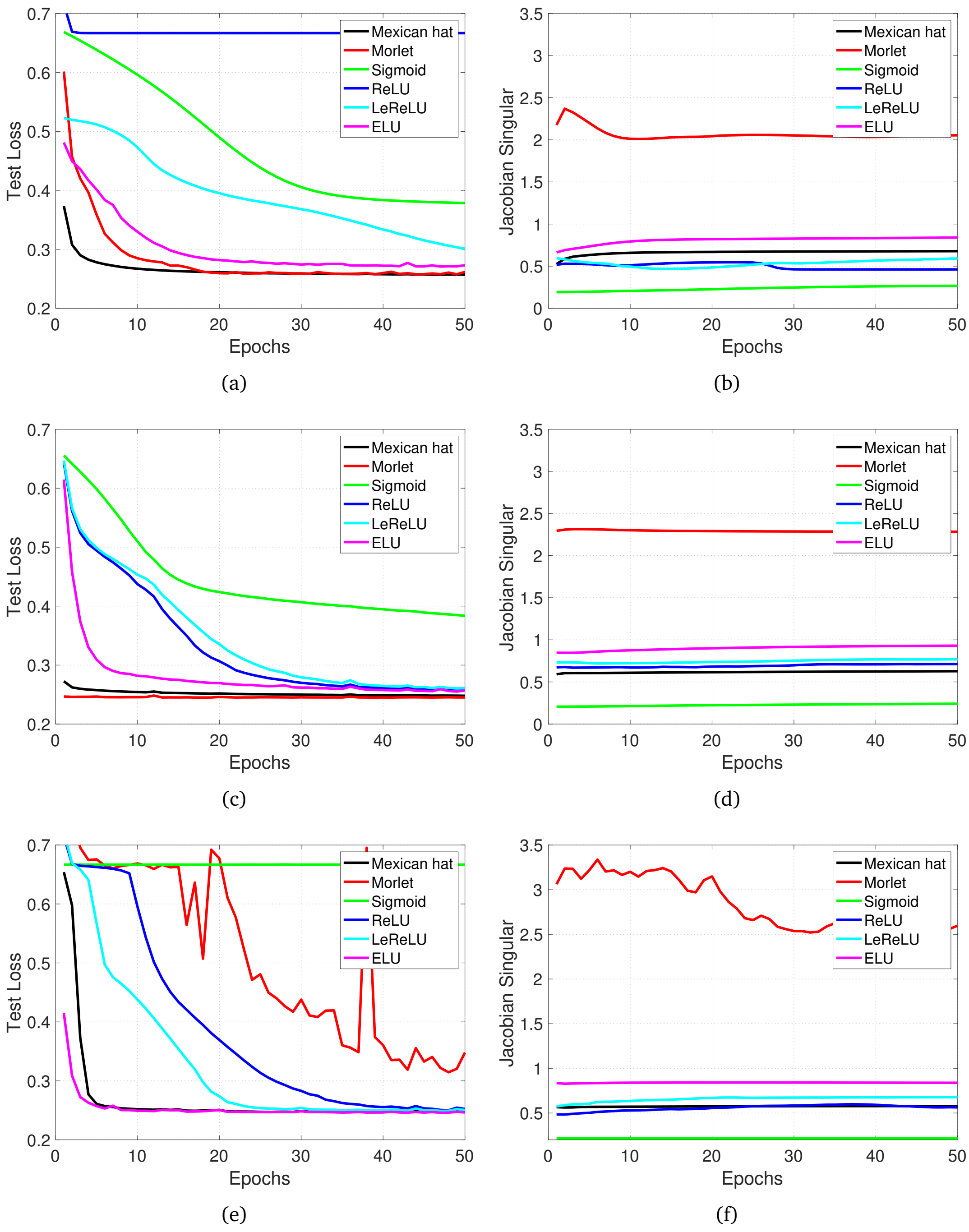


Figure 5.6: Test loss values (left) and average singular values of hidden layer Jacobian matrices (right) in the synthetic nonlinear signal identification experiment. Row 1-3: three-hidden-layer 1, three-hidden-layer 2, and ten-hidden-layer.

The identification results of the MLP models for the test samples were averaged and shown in Fig. 5.7 and 5.8. It can be seen that the MLP models with TFMH could quickly and distinctly classify the samples into different classes in spite of the depth of structure. When the structure went deeper, the performance of Relu family would be silghtly better, while the morlet and sigmoid cases had worse results.

5.4.2 Approximation for a Single Variable Function

In this section, an experiment was set up to approximate a piecewise defined function:

$$F(x) = \begin{cases} -2.186x - 12.864, & -10 \leq x < -2 \\ 4.246x, & -2 \leq x < 0 \\ 10\sin((0.03x + 0.7)\,x)\,e^{-0.05x-0.5}, & 0 \leq x \leq 10 \end{cases} \tag{5.20}$$

over the domain [-10, 10] with interval 1. This single variable function is shown in grey in Fig. 5.11.

Structures and Hyper-parameters

Two MLP models with three and ten hidden layers were respectively used in this experiment for comparison. Their structures and hyper-parameters are listed in Tab. 5.6. The 2001 length sequence x in Eqn. (5.20) was taken as inputs and the models were trained to output a signal approximating $F(x)$.

Table 5.6: Structures and hyper-parameters of the MLP models in the single variable function approximation experiment.

three-hidden-layer	[1 100 100 100 1]
ten-hidden-layer	[1 100 100 100 50 50 50 50 50 30 30 1]
epochs	100
learning rate	0.01
weights initialization	Xavier
bias initialization	zero
optimizer	Adam
loss function	MSE

Results

The training loss values of two MLP structures are compared in Fig. 5.9. The convergence regions had also been scaled up for details. The TFMH case had the best performance

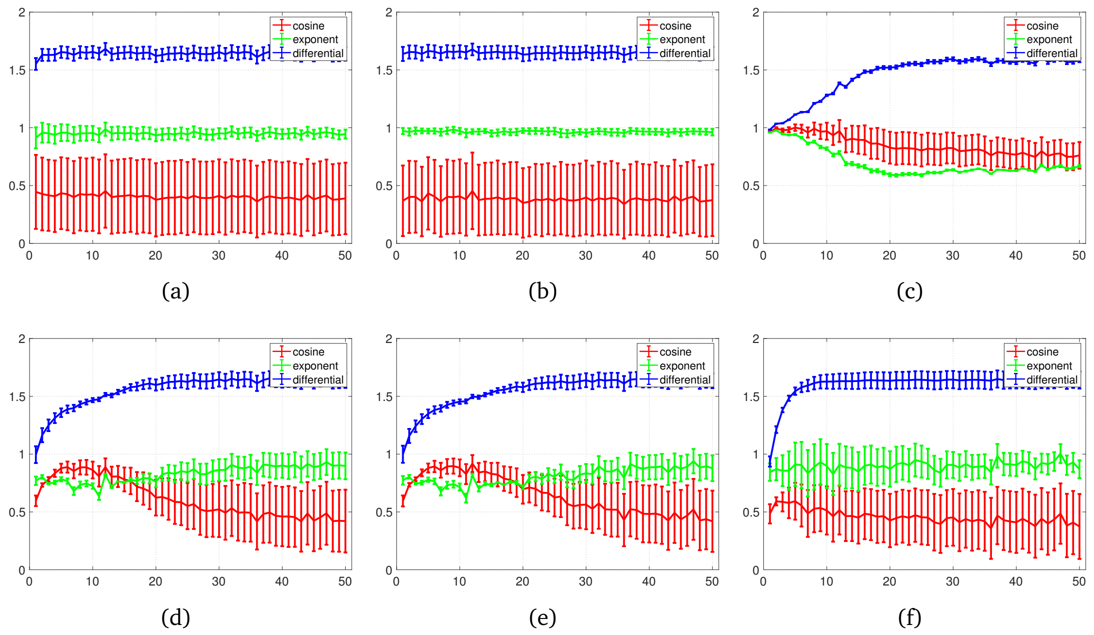


Figure 5.7: Identification results of the MLP models with three-hidden-layer 2 during training in the synthetic nonlinear signal identification experiment. (a)-(f): TFMH, morlet, sigmoid, relu, lerelu, and elu.

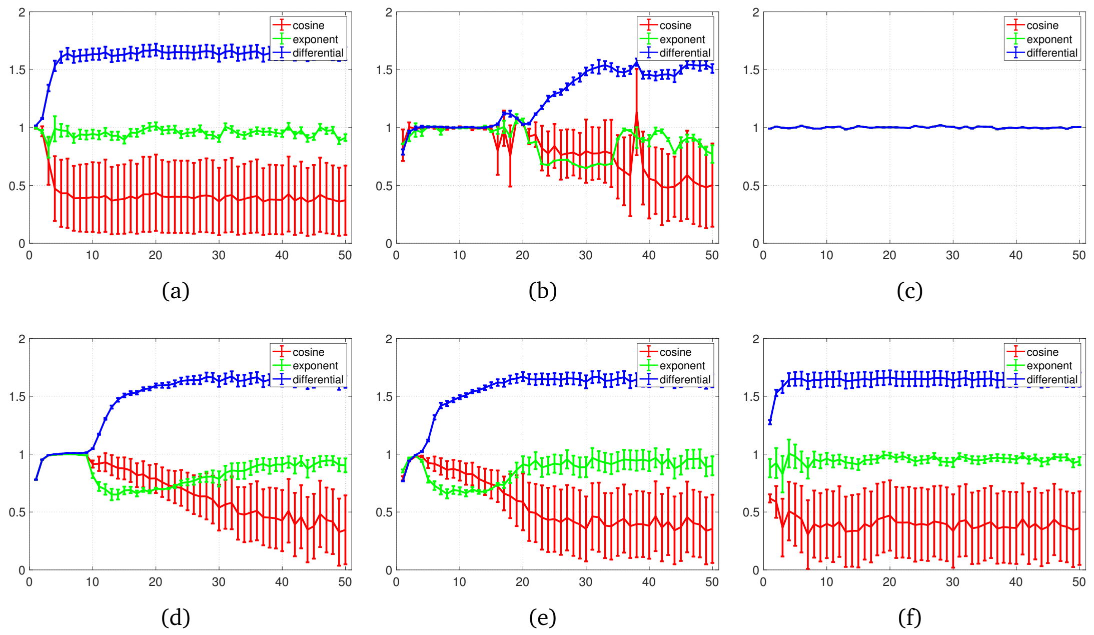
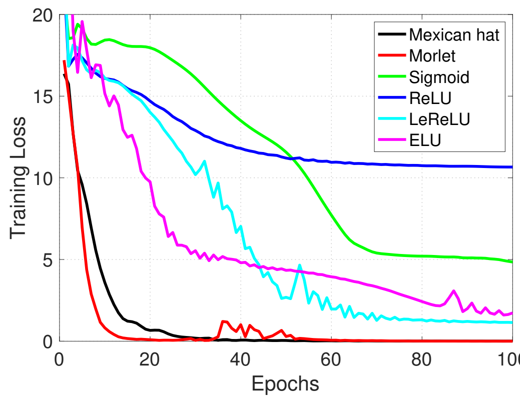
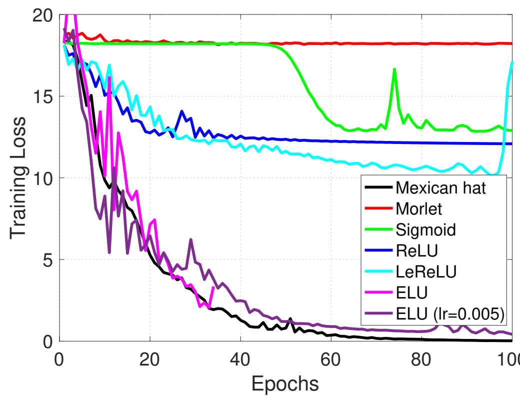


Figure 5.8: Identification results of the MLP models with ten-hidden-layer during training in the synthetic nonlinear signal identification experiment. (a)-(f): TFMH, morlet, sigmoid, relu, lerelu, and elu.

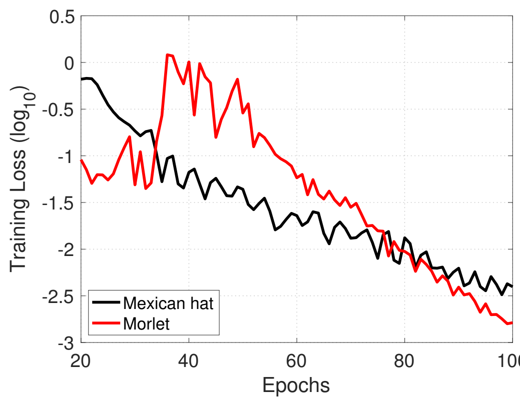
regardless of the depth of MLP model. While the morlet diverged in the case of ten-hidden-layer. The relu and lerelu cases didn't converge to a good point when the models went deeper, meanwhile the gradient vanishing problem came out in the sigmoid case. Even though elu converged with a lower learning rate (0.005, colored by purple in Fig. 5.9(b)), it was found sensitive to the learning rate where a higher one (0.01) led to the gradient exploding problem (colored by pink in Fig. 5.9(b)). In addition, when comparing the convergence regions, as shown in Fig. 5.9(c) and 5.9(d), it can be found that TFMH case was more stable. The comparison indicates that TFMH has the ability to reduce the overfitting problem.



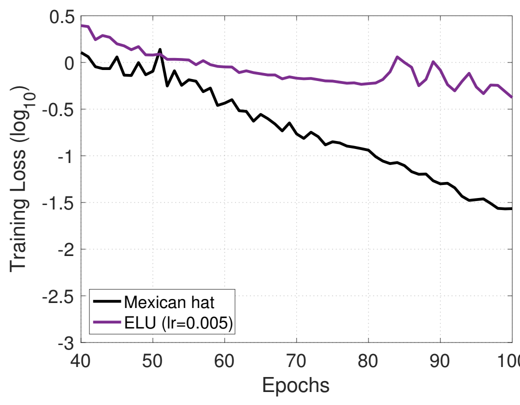
(a)



(b)



(c)



(d)

Figure 5.9: Training loss values of the MLP models with three-hidden-layer (left) and ten-hidden-layer (right) in the single variable function approximation experiment. The second row shows the scaled convergence regions.

The average singular values of hidden layer Jacobian matrices further verified the advantages of TFMH and elu for stable training process, as shown in Fig. 5.10. The approximation results of the MLP models with different structures and ADFs are illustrated in Fig. 5.11 and

5.12.

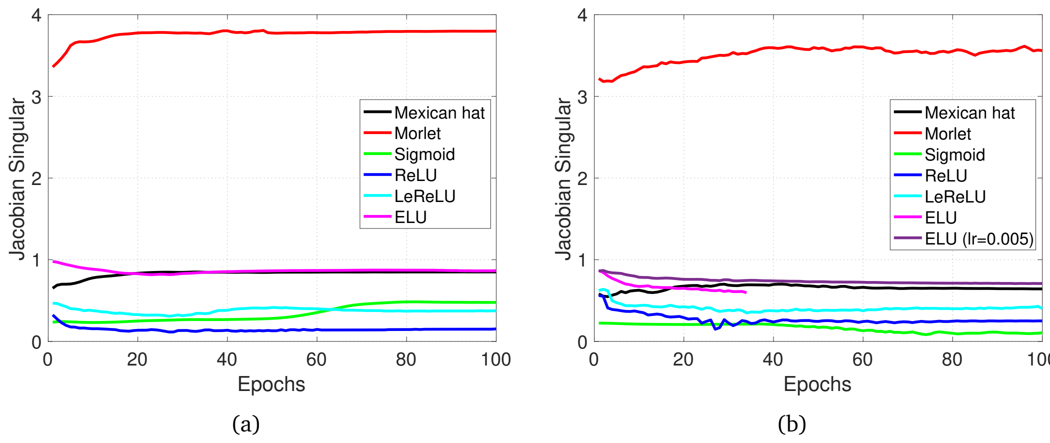


Figure 5.10: Average singular values of hidden layer Jacobian matrices of the MLP models with (a) three-hidden-layer and (b) ten-hidden-layer in the single variable function approximation experiment.

5.4.3 Approximation for a Real Nonlinear Signal

Interferometric Fiber Sensors (IFS) are often used to measure multiple parameters of pressure based on two or more laser beams which exist optical path difference (OPD) (Liu et al., 2015). They can be seen as complex nonlinear systems producing signals written as:

$$y = \sum_{i=1}^{N} A_i \cos\left(\frac{a_i}{\lambda_k}\right) + \eta_0 + \eta_{DC}, \tag{5.21}$$

where, η_0 is random noise and η_{DC} is DC component, λ_k represents the uniform wavelength and is the input of target signal y. Parameters A_i, a_i, and N can be estimated with the current research (Lin et al., 2019). The goal of this section was to approximate the real sample y, as shown in grey in Fig. 5.15, with MLP models.

Structures and Hyper-parameters

Similarly, two MLP models with three and ten hidden layers were compared in this experiment. Their structures and hyper-parameters are listed in Tab. 5.7.

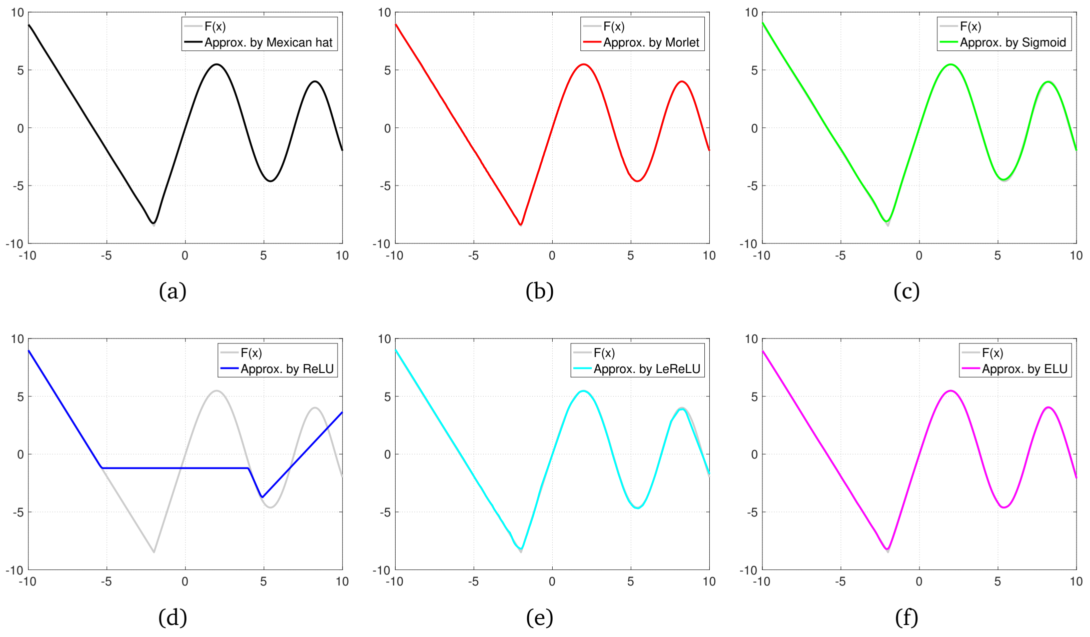


Figure 5.11: Approximation results of the MLP models with three-hidden-layer in the single variable function approximation experiment. (a)-(f): TFMH, morlet, sigmoid, relu, lerelu, and elu (learning rate is 0.005).

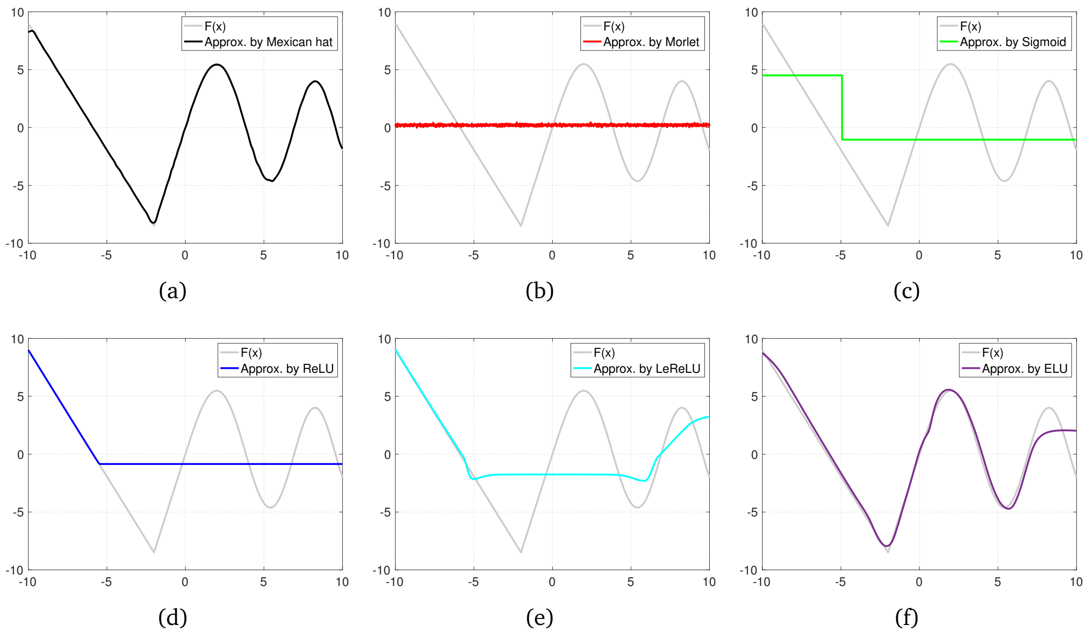


Figure 5.12: Approximation results of the MLP models with ten-hidden-layer in the single variable function approximation experiment. (a)-(f): TFMH, morlet, sigmoid, relu, lerelu, and elu (learning rate is 0.005).

Table 5.7: Structures and hyper-parameters of the MLP models in the real nonlinear signal approximation experiment.

three-hidden-layer	[1 500 300 100 1]
ten-hidden-layer	[1 100 100 50 50 50 30 30 10 10 10 1]
epochs	500
learning rate	0.0001
weights initialization	Xavier
bias initialization	zero
optimizer	Adam
loss function	MSE

Results

The training loss values of two MLP structures are compared in Fig. 5.13. The model with morlet achieved the best performance in the case of shallow structure, while diverged when increasing the depth. The relu, lerelu, and sigmoid cases couldn't converge to a low loss level, and the elu case suffered from the numerical overflow problem although the learning rate (0.0001) was not high. However, the model with TFMH converged to a low loss both for shallow and deep structures. The average singular values of hidden layer Jacobian matrices again verified the stable training process of TFMH case, as shown in Fig. 5.14. The approximation results are compared in Fig. 5.15 and 5.16.

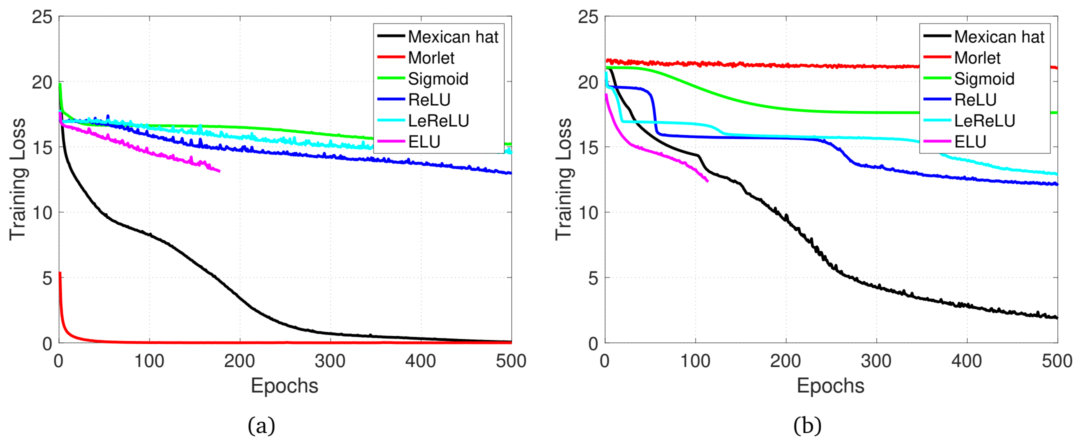
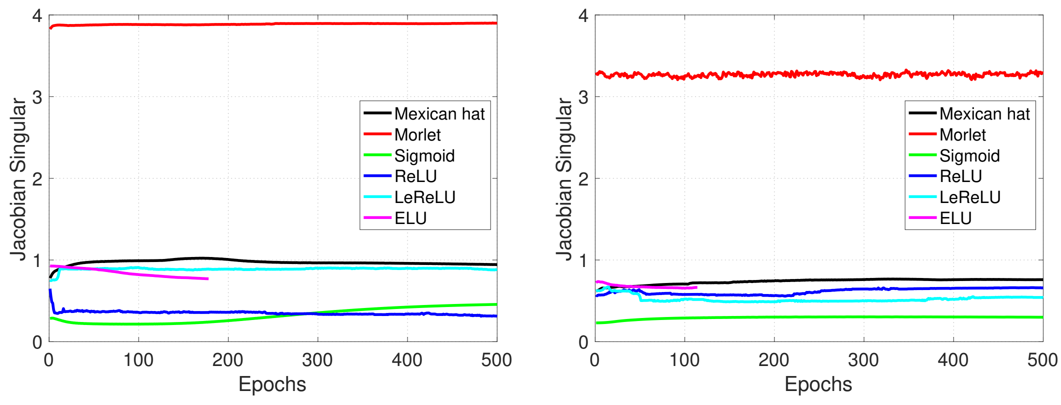
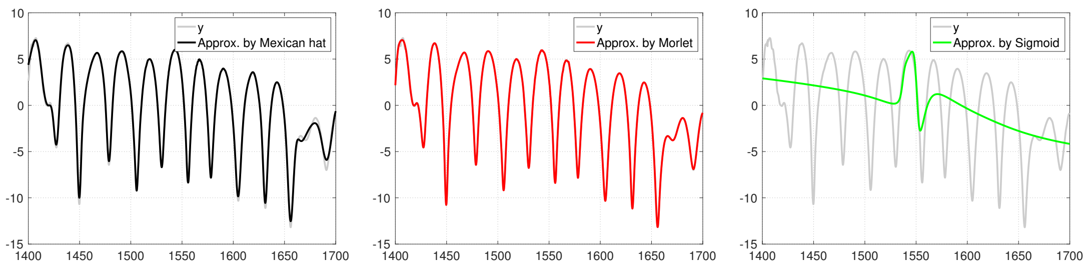


Figure 5.13: Training loss values of the MLP models with three-hidden-layer (left) and ten-hidden-layer (right) in the real nonlinear signal approximation experiment.

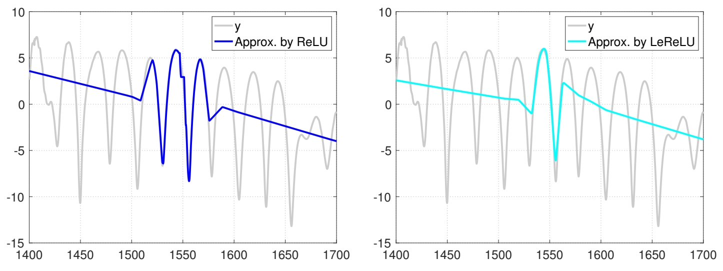


(a) (b)

Figure 5.14: Average singular values of hidden layer Jacobian matrices of the MLP models with (a) three-hidden-layer and (b) ten-hidden-layer in the real nonlinear signal approximation experiment.



(a) (b) (c)



(d) (e)

Figure 5.15: Approximation results of the MLP models with three-hidden-layer in the real nonlinear signal approximation experiment. (a)-(e): TFMH, morlet, sigmoid, relu, and lerelu. The elu results are not plotted due to numerical overflow.

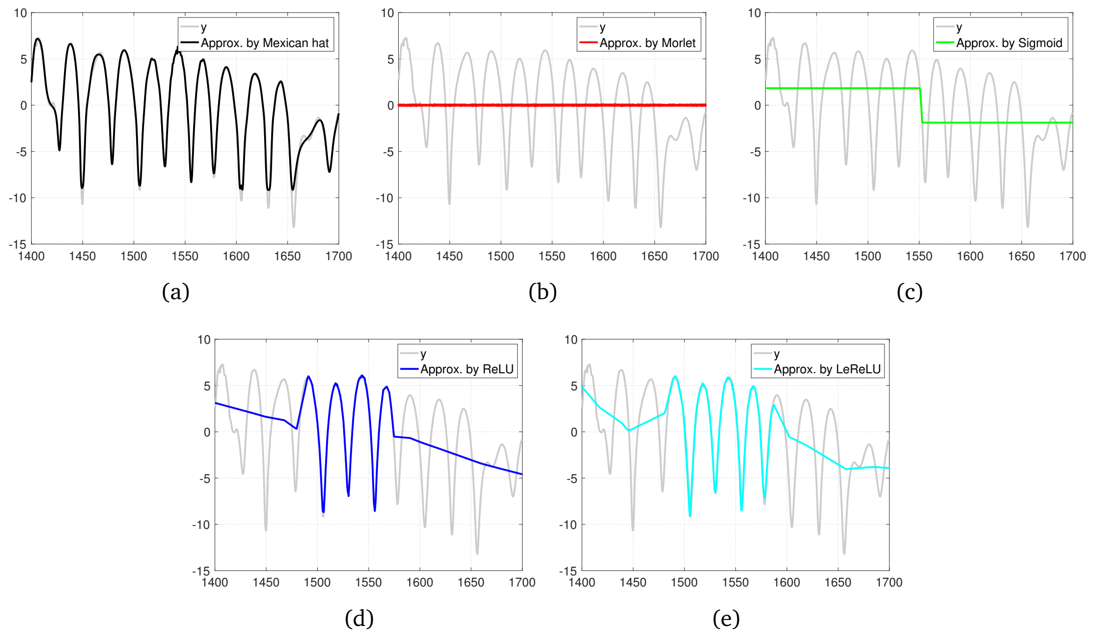


Figure 5.16: Approximation results of the MLP models with ten-hidden-layer in the real nonlinear signal approximation experiment. (a)-(e): TFMH, morlet, sigmoid, relu, and lerelu. The elu results are not plotted due to numerical overflow.

5.4.4 Recognition of the Handwritten Digits

MNIST is a well-known dataset for handwritten digits recognition. Even though there have many researches dedicated to a variety of algorithms for improvements of accuracy on MNIST[1], this dataset is a good start for us to compare the performances of different ADFs for CNN models. MNIST contains 60000 samples for training and 10000 samples for testing. Each sample is a bilevel image with 28×28 size, which indicates one of ten digits (0 to 9). However, in this experiment, a sub-dataset of the training dataset of MNIST was used to investigate the properties of ADFs, because a smaller training dataset usually makes the learning harder. Therefore, 1000 samples were prepared for training, with 100 samples per class.

Structures and Hyper-parameters

The structures and hyper-parameters of the CNN models used in the experiment are listed in Tab. 5.8.

[1]http://yann.lecun.com/exdb/mnist/

Table 5.8: Structures and hyper-parameters of the CNN models in the handwritten digits recognition experiment.

CNN 1	ConvL 1	2 @ 3 × 3	same padding
	PoolL 2	max pooling	
	ConvL 3	2 @ 3 × 3	same padding
	PoolL 4	max pooling	
	FcL 1	10	
	FcL 2	10	
CNN 2	ConvL 1	2 @ 3 × 3	same padding
	PoolL 2	max pooling	
	ConvL 3	2 @ 3 × 3	same padding
	PoolL 4	max pooling	
	ConvL 5	4 @ 2 × 2	valid padding
	PoolL 6	max pooling	
	FcL 1	10	
	FcL 2	10	
	FcL 3	10	
	epochs	200	
	learning rate	0.001	
	weights initialization	Xavier	
	bias initialization	zero	
	optimizer	Adam	
	loss function	CE	

Results

The training and test loss values of the two CNN models are recorded during training, as shown in Fig. 5.17(a) and 5.17(b). From the figures we found that TFMH case converged fastest both in shallow and deep structures, while morlet case diverged. The sigmoid always suffered from gradient vanishing problem and learnt nothing. The lerelu and elu cases converged slowly in deeper structure and the relu even diverged when the structure went deeper.

The differences between training and test loss values of different ADFs are compared in Fig. 5.17(c) and 5.17(d). The results showed that TFMH had the ability to alleviate overfitting problem. Meanwhile, the accuracies on test datasets indeed showed the advantages of TFMH in handwritten digits recognition, as listed in Tab. 5.9.

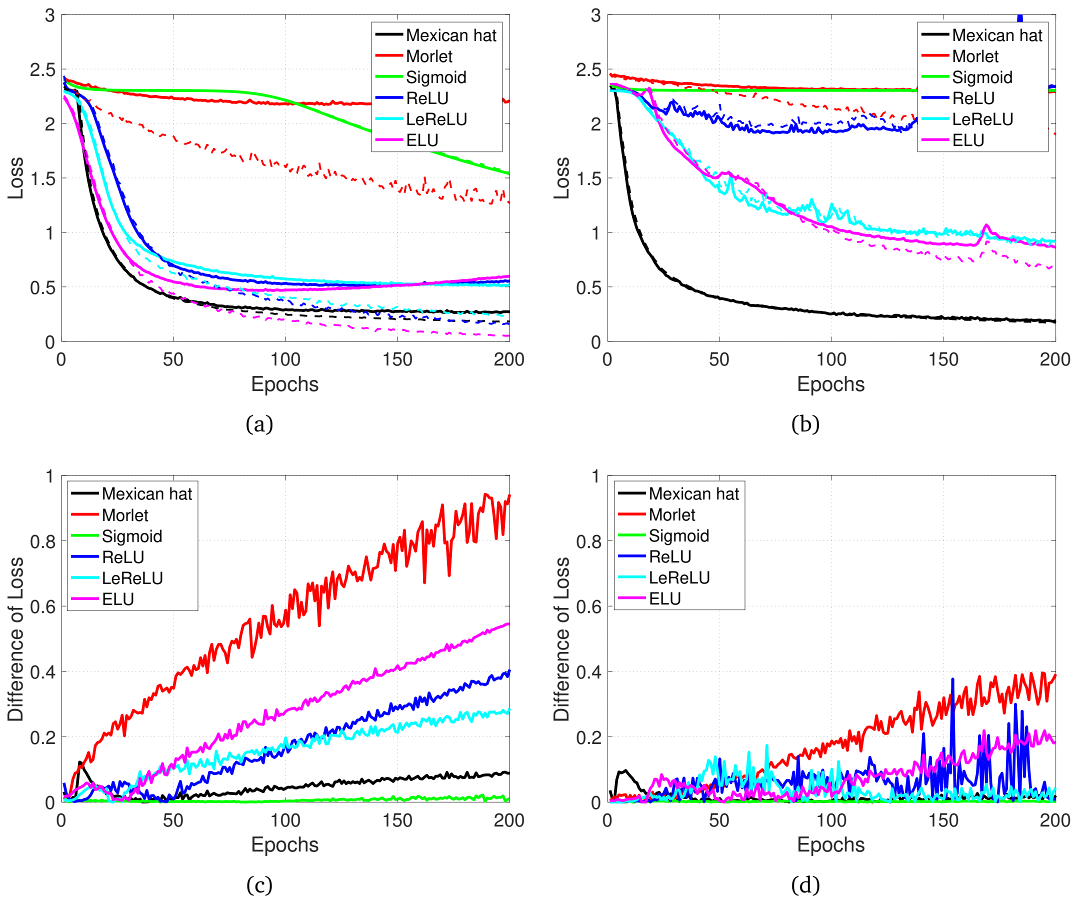


Figure 5.17: Training loss values (top, dashed lines) and test loss values (top, solid lines), and their differences (bottom) of CNN 1 (left) and CNN 2 (right) in the handwritten digits recognition experiment.

Table 5.9: Accuracy results of the CNN models with different ADFs in the handwritten digits recognition experiment.

Model	Activation Function and Derivative Function					
	TFMH	Morlet	Sigmoid	ReLU	LeReLU	ELU
CNN 1	**92.66%**	25.88%	52.82%	85.82%	85.06%	87.69%
CNN 2	**94.44%**	17.19%	7.97%	9.79%	69.31%	74.70%

5.5 Discussions

Typically, existing researches mainly focus on the design of activation function and simply take its derivative as the derivative function. However, there is a possibility to improve the performance of DNN training further if attempting to optimize the derivative function. Based on the analysis, the performance of DNN training process depends on four properties of ADF used: zero-mean, constant energy transfer, nonlinear, and sparse. Qualitatively, these four aspects of the six ADFs used in the experiments are compared in Tab. 5.10. The performances in the different fields are sorted into "bad" (B), "modest" (M) and "good" (G). The zero-mean function, mentioned in section 5.1.1, performs normalization on its output data, thus it is called "normalization property".

Table 5.10: Comparison of the properties of different ADFs.

Property	Activation Function and Derivative Function					
	TFMH	Morlet	Sigmoid	ReLU	LeReLU	ELU
normalization	G	M	B	B	M	M
constant energy transfer	G	B	B	B	B	B
nonlinearity	G	G	M	B	B	M
sparsity	G	G	B	B	B	B

The experiments showed that the nonlinear fitting capability of NN models heavily depended on the nonlinearity of the ADF used, especially in the case of shallow structures. While the deep structures often had numerical issues, such as gradient vanishing or exploding problem, where an ADF with good normalization property is helpful. Propagating data among hidden layers in DNN training process also requires the ADF to constrain the energy of data and keep data flow well for fast training. In addition, the sparsity of ADF helps to reduce the hardware resources and alleviate the overfitting problem. It also resulted in robustness to noisy inputs, which was one of the reasons why the TFMH training was more stable than others.

Morlet function always performed well in shallow structure while bad in deep structure in the nonlinear problems. We argue that Morlet has modest normalization property and good nonlinearity, hence it helped DNN to learn features quickly when the model was not too deep. But the biggest disadvantage of it is the divergent energy transfer during training, which led to the failure in deep learning.

Sigmoid function cannot normalize its outputs to near-zero mean, and it also cannot constrain the energy of propagating data, thus it can easily give rise to gradient vanishing problem in DNN training. Meanwhile, it is approximately-linear, which limits its learning capability and slows down convergence even for shallow structures.

ReLU and LeReLU functions are also approximately-linear, thus their performances were not improved dramatically although adding hidden layers helps to enhance the nonlinear fitting capability of models. The linearity also resulted in slow learning processes. Another reason of ReLU and LeReLU functions performed bad in deep structures comes from the unrestricted propagating data during training.

The nonlinearity of ELU is a little bit stronger than ReLU and LeReLU functions, which helped to achieve good results in shallow structures sometimes. But it was sensitive to the depth of model and the learning rate, leading it to suffer from high risk of gradient exploding problem. This result can be interpreted from the viewpoint of signal filtering. The drastic changes coming from high learning rate were suppressed in the TFMH case, acting as a low-pass filter. This filtering produced a smoother convergence curve than others, indicating training stability. For ELU function, an all-pass filter was in place and all components could go through, leading to numerical problems as they flowed deeper.

Taking advantage of all the properties discussed, the DNN model with TFMH function could quickly converge to a satisfactory point. This solution indeed stabilized the training and helped to speed up convergence. The experimental results clearly demonstrated the advantages of the tight frame wavelet functions for fast DNN training.

6 Systematic Experiments

To investigate the effectiveness of feature pre-extraction, chapter 3 proposes a pathfinder topology of Deep Neural Network (DNN), i.e., Fast Fourier Transform Neural Network (FFTNN), to perform Deep Learning (DL) in frequency domain. Its Fast Fourier Transform (FFT) component actually acts as a feature pre-extraction step within the FFTNN model. As the first part of this chapter, some extension experiments are described to further study the proprieties of the tight frame Mexican hat wavelet functions (TFMH, introduced in chapter 5) performing in frequency domain.

Further, the proposed feature pre-extraction method (FDIDWT, introduced in chapter 4) and the TFMH are combined to construct a new topology of Wavelet Neural Network (WNN), termed as FDIDWT-MEXHACT-NN or simply FDMHNN, to finish the binary classification tasks built on two astronomical datasets. Meanwhile, the FDIDWT and the TFMH of the proposed FDMHNN model are also compared with other preprocessing methods and other Activation Function and Derivative Function pairs (ADFs), respectively.

These experiments were set up on the Ubuntu 18.04 system, with CPU Intel Core i5-8400. All the models were composed of the fully-connected layers, whose computing was accelerated by the high-level deep learning framework, i.e., TensorFlow (GPU version 1.10). The GPU is NVIDIA GeForce GTX1050Ti with 768 CUDA cores.

6.1 Extension Experiments of Speech Recognition

This section describes the extension experiments of the speech recognition tasks performed in section 3.4.2, after the TFMH has been proposed. In particular, the FFTNN model utilizes the ELU as its ADF, while now the TFMH is tested for comparison. Thus, the comparative FFTNN model replaced ELU with TFMH and was termed as FFTMHNN. Except for the ADF, these two models had the same number of hidden units and hyper-parameters as shown in Tab. 3.3 and 3.4 for the speech recognition missions. The experimental data used here was also the same as in section 3.4.2. During training, the test loss values were recorded to observe the convergence; they will be reported and discussed in next chapter.

While gradient exploding problem is also common in DNN training. It may come out when the model is very deep or the learning rate is set a little bit high. So in this part, a deep FFTNN model (5 hidden layers) was also built to test the stability of DNN training in frequency domain. The number of hidden units and the hyper-parameters of the deep FFTNN and FFTMHNN models are listed in Tab. 6.1. The experimental results will also be discussed in next chapter.

Table 6.1: The number of hidden units and the hyper-parameters of the FFTNN and FFTMHNN models with a deep structure in the speech recognition missions.

number of hidden units	[30, 30, 20, 20, 10]
learning rate	0.0001
epochs	100
batch size	512
optimizer	Adam
weights initialization	Xavier
bias initialization	zero
activation function	ELU for FFTNN, TFMH for FFTMHNN
loss function	CE

6.2 Systematic Experiment for Astronomical Data

More systematic experiments were performed on the astronomical data to investigate the properties of the proposed FDMHNN model.

6.2.1 Astronomical Data Preparation

Nano-satellite Mission

Even though there have been many telescopes covering different energy bands, most of the sky is not being observed with good sensitivity around 1 MeV. While existing instruments have a limited field of view around the MeV range. The best instrument designed for the MeV range was the COMPTEL (Imaging Compton Telescope, see Schonfelder et al. 1984) on-board CGRO (Compton Gamma Ray Observatory, see Schoenfelder et al. 1993), which flew in the 1990s and permitted a huge leap in technology. Therefore, the performance of a future Compton telescope is expected to improve at least by an order of magnitude for COMPTEL. Next-generation Compton observatories have been proposed for the immediate future (Andritschke et al., 2005; Mizumoto et al., 2015; De Angelis et al., 2017), with costs around half a billion Euros and a timescale for development and launch of ∼10 years.

Recently, a small test instrument, called *nano-satellite*, was designed for low cost (approximately half a million Euros) and fast launch (about one year) (Lucchetta et al., 2017; Cao et al., 2019; Rando et al., 2019). Following the general guidelines for a future Compton telescope, a CubeSat standard nano-satellite mainly consists of three parts:

1) *Silicon Tracker*: in which the Compton scattering happens, and the Double-Sided Strip Detectors (DSSDs) are used to measure the energy of the recoil electron and the position of each interaction;

2) *Calorimeter*: made of CsI(Tl) crystals, and serves the purpose of measuring the energy and the direction of the scattered photon. Two calorimeters are always used: a bottom calorimeter and a lateral calorimeter, which detect the scattered photon in the bottom and near the edges of the tracker, respectively;

3) *Anticoincidence System*: designed with a plastic scintillator to reject charged background.

The dominant interaction mechanism of gamma-ray with matter inside the nano-satellite is Compton scattering, also called Compton events. It is discovered by Arthur Holly Compton in 1922. The illustrated Compton scattering and the structure of a nano-satellite are shown in Fig. 6.1. More design details can be found in Lucchetta (2016) and Berlato (2016).

From the ideal standpoint, it is possible to reconstruct both the direction and the energy of the incident photon based on the measurements of recoil electron and scattered photon. This process is called Compton event reconstruction. Study of the sources relies on the reconstructed photon, thus the incident photon is expected to be reconstructed as precisely as possible. Unfortunately, in any real telescope, several additional aspects have to be taken into consideration. For example, measurement uncertainties, missing quantities reconstructions, and background events will result in misleading results. The Compton kinematics and Compton scattering in a real instrument are detailed in Berlato (2016).

One of the most significant background contributions for the nano-satellite is *pair production* event. For this event, the positron produces a couple of additional gamma-rays with a non-negligible probability of escaping the thin calorimeters. This usually translates into

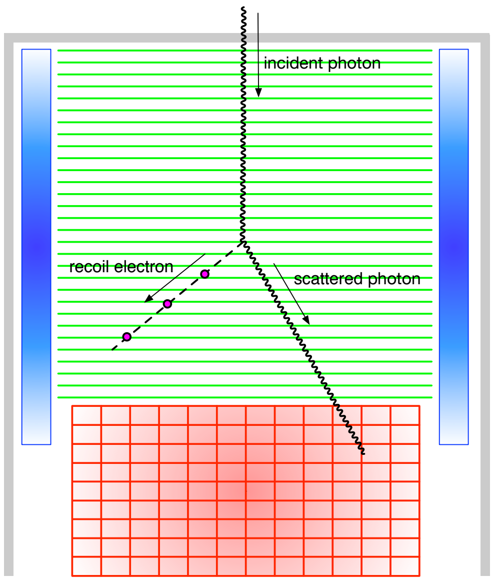


Figure 6.1: Compton scattering interacting in the designed nano-satellite. The size of the satellite sketched here does not strictly respect to the real design. The tracker, bottom calorimeter, and Anticoincidence system are colored by green, red, and grey respectively. The blue part represents the lateral calorimeter.

mis-reconstructing such events as Compton events, with a completely wrong direction of the primary gamma. Therefore, the real Compton events are expected to be distinguished from the pair production events, which can be treated as a binary classification task.

The experimental events were simulated with the Medium Energy Gamma-ray Astronomy Library (MEGALib) (Zoglauer et al., 2008). They were produced from a diffuse source, which gave a good sample of simulated photons at all inclination angles and energies in the range of interest. The constructed dataset, termed as CPP, was a mixture of 20000 real Compton events with 10000 pair production events.

In order to evaluate the models with the Hold-out Method (HoM, introduced in section 2.3.1), the dataset was randomly divided into the training and test datasets five times, which resulted in five groups of sub-datasets: CPP1, CPP2, CPP3, CPP4, and CPP5. The number of samples in the training and test datasets followed the ratio of 7:3, i.e., 14000 Compton events and 7000 pair production events in training dataset, and 6000 Compton events and 3000 pair production events in test dataset. These events have 31 attributes described in Tab. 6.2, after measuring the direction and energies of the recoil electron and scattered photon in Compton events.

Table 6.2: Descriptions for the 31 attributes of the simulated CPP dataset.

Index	Attribute	Description
[0]	m_Ei	measured total energy of recoil electron and scattered photon
[1]	EnergyD1	measured energy in tracker
[2]	EnergyD2	measured energy in calorimeter
[3]	NHitSQR0	number of measured hits in tracker
[4]	NHitSQR1	number of measured hits in bottom calorimeter
[5]	NHitSQR2	number of measured hits in lateral calorimeter
[6]	m_SequenceLength	number of hits in the reconstructed Compton sequence
[7]	m_TrackLength	length (cm) of the recoil electron track
[8]	m_ComptonQualityFactor	the Compton quality factor for event reconstruction
[9]	m_TrackQualityFactor	the quality factor of the recoil electron track
[10]	m_LeverArm	distance from Compton scattering point to photon absorption point
[11]	m_Theta	Compton angle: from scattered photon to recoil electron
[12]	m_Epsilon	Compton angle: from incident photon direction to recoil electron direction
[13]	m_Phi	Compton angle: from incident photon direction to scattered photon direction
[14]	m_Dg_x	3D vector: direction of scattered photon (x)
[15]	m_Dg_y	3D vector: direction of scattered photon (y)
[16]	m_Dg_z	3D vector: direction of scattered photon (z)
[17]	m_De_x	3D vector: direction of recoil electron (x)
[18]	m_De_y	3D vector: direction of recoil electron (y)
[19]	m_De_z	3D vector: direction of recoil electron (z)
[20]	m_C1_x	3D vector: Compton scattering position (x)
[21]	m_C1_y	3D vector: Compton scattering position (y)
[22]	m_C1_z	3D vector: Compton scattering position (z)
[23]	m_C2_x	3D vector: position of scattered photon absorption (x)
[24]	m_C2_y	3D vector: position of scattered photon absorption (y)
[25]	m_C2_z	3D vector: position of scattered photon absorption (z)
[26]	m_Ee	measured energy of recoil electron
[27]	m_dEe	error of measured energy of recoil electron
[28]	m_Eg	measured energy of scattered photon
[29]	m_dEg	error of measured energy of scattered photon
[30]	ProbAbsorptionAlong	amount of radiation lengths between Compton scattering position and position of scattered photon absorption

3FGL Source Samples

Fermi-LAT is an imaging high-energy gamma-ray telescope spanning the energy band from about 20 MeV to more than 300 GeV. Its field of view covers about 20% of the sky at any time and the whole sky every three hours through a continuous scan. In contrast to the previous telescope, e.g., EGRET (Energetic Gamma-Ray Experiment Telescope, on-board CGRO, see Fichtel et al. 1993), *Fermi*-LAT shows better energy resolution, better angular resolution, and wider effective area in both low-energy and high-energy bands[1].

[1]Check `http://www.slac.stanford.edu/exp/glast/groups/canda/lat_Performance.htm` for more performance of *Fermi*-LAT.

The third *Fermi*-LAT source catalog (3FGL), released through the Fermi Science Support Centre (FSSC) in January 2015, contains 3034 entries with 16 classes of gamma-ray sources (Acero et al., 2015). In 3FGL, there are 1744 Active Galactic Nuclei (AGN), 1010 unassociated sources, and 280 other sources termed as nonAGN. It is significant to identify AGNs from the unassociated sources since AGN class dominates the associated sources ($\sim 86\%$). This can also be seen as a binary classification task. In the experiment, the 3FGL source samples were downloaded from Space Science Data Center (SSDC)[2]. Each sample has 41 attributes, as listed in Tab. 6.3.

However, there exists missing data under the attributes of uncertainties of multi-energy bands flux, which has been summarized in Tab. 6.4. In the experiment, the missing data was simply filled by the mean value of the attribute values of other samples. For instance, the missing $\text{Error}_{100-300\text{MeV}}$ upper limits of BCU sources were filled by the mean value of known $\text{Error}_{100-300\text{MeV}}$ upper limits of other BCU sources. It is worth noting that the missing data of 1010 unassociated sources was filled by the mean value of the corresponding attribute of all the other complete sources.

After processing the missing data, all the associated sources were taken as binary classes: AGN and nonAGN, and they were randomly divided into training and test datasets five times as well, with the ratio of 7:3. The five groups of sub-datasets were named as: FGL1, FGL2, FGL3, FGL4, and FGL5. The training dataset contained 1220 AGNs and 196 nonAGNs, while the test dataset had 524 AGNs and 84 nonAGNs.

[2]http://www.ssdc.asi.it/fermi3fgl/

Table 6.3: Descriptions for the 41 attributes of the 3FGL dataset.

Index	Attribute	Description
[0]	RA_h	*Fermi* right ascension: hour
[1]	RA_m	*Fermi* right ascension: minute
[2]	RA_s	*Fermi* right ascension: second
[3]	DEC_d	*Fermi* declination: degree
[4]	DEC_m	*Fermi* declination: minute
[5]	DEC_s	*Fermi* declination: second
[6]	LII	Galactic longitude
[7]	BII	Galactic latitude
[8]	$\text{Significance}_{100\text{MeV}-300\text{GeV}}$	source significance in σ units over the 100 MeV to 300 GeV band
[9]	$F_{1-100\text{GeV}}$	integral photon flux from 1 to 100 GeV in units of $\text{ph} \cdot \text{cm}^{-2} \cdot \text{s}$
[10]	$\text{Error}_{1-100\text{GeV}}$	1 σ error on integral photon flux from 1 to 100 GeV
[11]	Pivot Energy	energy at which error on differential flux is minimal
[12]	Spectral index	best fit photon number power-law index
[13]	α-error	1 σ error on spectral index
[14]	Beta	curvature parameter, for LogParabola
[15]	β-error	1 σ error on curvature parameter
[16]	Cutoff Energy	cutoff energy for PL(Super)ExpCutoff
[17]	γ-error	1 σ error on cutoff energy
[18]	Power law index	best fit power-law index
[19]	$F_{100-300\text{MeV}}$	integral photon flux from 100 to 300 MeV
[20]	$\text{Error}_{100-300\text{MeV}}$-upper	1 σ error on integral photon flux from 100 to 300 MeV (upper limit)
[21]	$\text{Error}_{100-300\text{MeV}}$-lower	1 σ error on integral photon flux from 100 to 300 MeV (lower limit)
[22]	$\text{Significance}_{100-300\text{MeV}}$	significance on 100-300 MeV
[23]	$F_{0.3-1\text{GeV}}$	integral photon flux from 0.3 GeV to 1 GeV
[24]	$\text{Error}_{0.3-1\text{GeV}}$-upper	1 σ error on integral photon flux from 0.3 GeV to 1 GeV (upper limit)
[25]	$\text{Error}_{0.3-1\text{GeV}}$-lower	1 σ error on integral photon flux from 0.3 GeV to 1 GeV (lower limit)
[26]	$\text{Significance}_{0.3-1\text{GeV}}$	significance on 0.3-1 GeV
[27]	$F_{1-3\text{GeV}}$	integral photon flux from 1 GeV to 3 GeV
[28]	$\text{Error}_{1-3\text{GeV}}$-upper	1 σ error on integral photon flux from 1 GeV to 3 GeV (upper limit)
[29]	$\text{Error}_{1-3\text{GeV}}$-lower	1 σ error on integral photon flux from 1 GeV to 3 GeV (lower limit)
[30]	$\text{Significance}_{1-3\text{GeV}}$	significance on 1-3 GeV
[31]	$F_{3-10\text{GeV}}$	integral photon flux from 3 GeV to 10 GeV
[32]	$\text{Error}_{3-10\text{GeV}}$-upper	1 σ error on integral photon flux from 3 GeV to 10 GeV (upper limit)
[33]	$\text{Error}_{3-10\text{GeV}}$-lower	1 σ error on integral photon flux from 3 GeV to 10 GeV (lower limit)
[34]	$\text{Significance}_{3-10\text{GeV}}$	significance on 3-10 GeV
[35]	$F_{10-100\text{GeV}}$	integral photon flux from 10 GeV to 100 GeV
[36]	$\text{Error}_{10-100\text{GeV}}$-upper	1 σ error on integral photon flux from 10 GeV to 100 GeV (upper limit)
[37]	$\text{Error}_{10-100\text{GeV}}$-lower	1 σ error on integral photon flux from 10 GeV to 100 GeV (lower limit)
[38]	$\text{Significance}_{10-100\text{GeV}}$	significance on 10-100 GeV
[39]	Sign.Curve	fit improvement between power law and LogParabola or PLExpCutoff
[40]	Variability Index	sum of 2A-log(Likelihood) difference between the flux fitted in each time interval and the average flux over the full catalog interval

Table 6.4: The number of sources that contain missing data in 3FGL.

Source class	Attributes with missing data									
	$\text{Error}_{100-300\text{MeV}}$		$\text{Error}_{0.3-1\text{GeV}}$		$\text{Error}_{1-3\text{GeV}}$		$\text{Error}_{3-10\text{GeV}}$		$\text{Error}_{10-100\text{GeV}}$	
	Upper	Lower	Upper	Lower	Upper	Lower	Upper	Lower	Upper	Lower
Unassociated	335	335	115	115	54	54	76	76	309	309
FSRQ	35	35	4	4	1	1	24	25	147	147
BCU	167	167	93	93	25	25	19	19	102	102
SPP	18	18	3	3	3	3	2	2	16	16
PWN	8	8	5	5	5	5	3	3	3	3
SNR	9	9	2	2	1	1	1	1	2	2
SFR	1	1	1	1	1	1	0	0	0	0
GLC	4	4	1	1	0	0	0	0	7	7
CSS	0	0	0	0	0	0	0	0	1	1
GAL	0	0	0	0	0	0	0	0	1	1
PSR	37	37	6	6	0	0	0	0	47	47
RDG	4	4	4	4	0	0	0	0	1	1
NOV	0	0	0	0	0	0	1	1	1	1
BLL	213	213	88	88	16	16	6	6	28	28
SSRQ	0	0	0	0	0	0	1	1	1	1
NLSY	0	0	0	0	0	0	0	0	1	1
AGN	0	0	0	0	0	0	0	0	1	1
SUMMARY	831	831	322	322	106	106	133	134	668	668

6.2.2 Proposed Wavelet Neural Network: FDMHNN

There are two classification tasks in the systematic experiments:

1) to distinguish the Compton events from the pair production events in the CPP dataset, which is termed as Mission A;

2) to identify the AGNs from the unassociated sources of 3FGL, which is termed as Mission B.

The proposed FDMHNN model is used to finish these two tasks. It has two components, the first one implements the proposed feature pre-extraction method, i.e., FDIDWT; the second one is a binary Multilayer Perceptron (MLP) classifier equipped with the TFMH. Nevertheless, there are two hyper-parameters in the FDIDWT method, i.e., the decomposition level J and the transformed dimension O. Based on the discussions in section 4.4, comparing the convergence of MLP component on different datasets is useful to decide the hyper-parameters. Hence, firstly, the transformed datasets with different J and O should be prepared with FDIDWT.

Hyper-parameters of FDIDWT Method

In the experiments, the training datasets in the first group of CPP and 3FGL were used to determine the hyper-parameters for Mission A and Mission B, respectively. According to the procedure described in Tab. 4.1, *FD_ASE* algorithm gave the attribute set core ξC, based on which the attributes of CPP and 3FGL datasets were arranged as shown in Tab. 6.5. Note that, the individual contributions of attributes $a_{14}, a_{15}, a_{16}, a_{17}$ in 3FGL were quite small, thus they were not included when using FDIDWT.

Table 6.5: The arranged attributes used to transform the original CPP and 3FGL datasets.

Parameters	CPP Dataset					3FGL Dataset				
correlation threshold ξ	0.6					0.6				
attribute set core ξC	$a_{15}, a_{14}, a_{26}, a_{11}$					a_5, a_6, a_2				
arranged attributes $\mathbb{B}$	a_{15},	a_{14},	a_{26},	a_{11},	a_8,	a_5,	a_6,	a_2,	a_9,	a_{27},
	a_1,	a_7,	a_{28},	a_2,	a_3,	a_{23},	a_{40},	a_{31},	a_{19},	a_{32},
	a_4,	a_{23},	a_0,	a_5,	a_6,	a_{33},	a_{35},	a_{30},	a_{34},	a_{26},
	a_9,	a_{13},	a_{12},	a_{29},	a_{27},	a_{36},	a_{37},	a_8,	a_{29},	a_{28},
	a_{24},	a_{16},	a_{30},	a_{25},	a_{10},	a_{22},	a_{39},	a_{24},	a_{25},	a_{21},
	a_{19},	a_{20},	a_{21},	a_{22},	a_{17},	a_{20},	a_{10},	a_{11},	a_0,	a_{38},
	a_{18}					a_3,	a_{18},	a_{12},	a_{13},	a_7,
						a_4,	a_1			

Next, the alternative wavelet decomposition vector $\boldsymbol{c}$ and bookkeeping vector $\boldsymbol{l}$ were constructed based on the arranged attributes $\mathbb{B}$. According to the priori knowledge of wavelet analysis, a decomposition level around 5 is sufficient to represent the natural data at different resolutions. Then levels $J = 1 \sim 5$ were considered to construct the vectors.

The alternative vector pairs ($\boldsymbol{c}$ and $\boldsymbol{l}$) for Mission A and B are listed in Tab. 6.6 and Tab. 6.7, respectively. For simplicity, only the bookkeeping vector $\boldsymbol{l}$ and the filter length m are recorded. Together with the arranged attributes $\mathbb{B}$ listed in Tab. 6.5, the wavelet decomposition vector $\boldsymbol{c}$ can be derived. While it was necessary to insert one placeholder attribute if choosing $J = 1$ for two missions.

Inverse Discrete Wavelet Transform (IDWT) was then performed on each sample of the original training datasets with the alternative vector pairs, and different transformed training datasets were generated.

Experimental Models

In order to decide the FDMHNN model, the next step is to compare the performance of the MLP component on each transformed training dataset and take the case that results in the best performance as the choice of hyper-parameters J and O. Hence, two baseline MLP

Table 6.6: The alternative bookkeeping vectors and filter lengths for Mission A.

J	$\boldsymbol{l}$	m	J	$\boldsymbol{l}$	m
1	(16,16,3)	30	2	(8,8,15,29)	2
	(16,16,4)	30		(8,8,15,30)	2
	(16,16,5)	28		(9,9,13,21)	6
	(16,16,6)	28		(9,9,13,22)	6
	(16,16,7)	26		(10,10,11,13)	10
	(16,16,8)	26		(10,10,11,14)	10
	(16,16,9)	24		(11,11,9,5)	14
	(16,16,10)	24		(11,11,9,6)	14
	(16,16,11)	22	3	(4,4,8,15,29)	2
	(16,16,12)	22		(4,4,8,15,30)	2
	(16,16,13)	20		(5,5,8,13,23)	4
	(16,16,14)	20		(5,5,8,13,24)	4
	(16,16,15)	18		(6,6,8,11,17)	6
	(16,16,16)	18		(6,6,8,11,18)	6
	(16,16,17)	16		(7,7,8,9,11)	8
	(16,16,18)	16		(7,7,8,9,12)	8
	(16,16,19)	14		(8,8,8,7,5)	10
	(16,16,20)	14		(8,8,8,7,6)	10
	(16,16,21)	12	4	(4,4,5,7,11,19)	4
	(16,16,22)	12		(4,4,5,7,11,20)	4
	(16,16,25)	8	5	(5,5,5,5,5,6,7)	6
	(16,16,26)	8		(5,5,5,5,5,6,8)	6
	(16,16,27)	6			
	(16,16,28)	6			
	(16,16,29)	4			
	(16,16,30)	4			
	(16,16,31)	2			
	(16,16,32)	2			

Table 6.7: The alternative bookkeeping vectors and filter lengths for Mission B.

J	$\boldsymbol{l}$	m
1	(19,19,3)	36
	(19,19,4)	36
	(19,19,5)	34
	(19,19,6)	34
	(19,19,7)	32
	(19,19,8)	32
	(19,19,9)	30
	(19,19,10)	30
	(19,19,11)	28
	(19,19,12)	28
	(19,19,13)	26
	(19,19,14)	26
	(19,19,15)	24
	(19,19,16)	24
	(19,19,17)	22
	(19,19,18)	22
	(19,19,19)	20
	(19,19,20)	20
	(19,19,21)	18
	(19,19,22)	18
	(19,19,23)	16
	(19,19,24)	16
	(19,19,25)	14
	(19,19,26)	14
	(19,19,27)	12
	(19,19,28)	12
	(19,19,29)	10
	(19,19,30)	10
	(19,19,31)	8
	(19,19,32)	8
	(19,19,33)	6
	(19,19,34)	6
	(19,19,35)	4
	(19,19,36)	4
	(19,19,37)	2
	(19,19,38)	2
2	(10,10,17,31)	4
	(10,10,17,32)	4
	(11,11,15,23)	8
	(11,11,15,24)	8
	(12,12,13,15)	12
	(12,12,13,16)	12
	(13,13,11,7)	16
	(13,13,11,8)	16
3	(5,5,9,18,35)	2
	(5,5,9,18,36)	2
	(6,6,9,16,29)	4
	(6,6,9,16,30)	4
	(7,7,9,14,23)	6
	(7,7,9,14,24)	6
	(8,8,9,12,17)	8
	(8,8,9,12,18)	8
	(9,9,9,10,11)	10
	(9,9,9,10,12)	10
	(10,10,9,8,5)	12
	(10,10,9,8,6)	12
4	(3,3,5,9,17,33)	2
	(3,3,5,9,17,34)	2
	(8,8,8,7,6,3)	10
	(8,8,8,7,6,4)	10
5	(5,5,5,6,7,9,13)	6
	(5,5,5,6,7,9,14)	6
	(3,3,4,5,8,14,25)	4
	(3,3,4,5,8,14,26)	4

models were built for two missions, respectively. They were composed of five fully-connected hidden layers activated by the TFMH. More hyper-parameters of the baseline MLP models can be found in Tab. 6.8.

Table 6.8: The number of hidden units and the hyper-parameters of the baseline MLP models for the astronomical missions.

Hyper-parameters	Mission A	Mission B
number of hidden units	[100, 100, 50, 30, 20]	[50, 30, 30, 20, 10]
batch size	1024	64
optimizer	Adam	MGD
learning rate	$5e^{-5}$	0.01
epochs	100	
weights initialization	Xavier	
bias initialization	zero	
ADF	TFMH	
loss function	MSE	

The FDIDWT method and the baseline MLP models were concatenated to construct the FDMHNN candidates. Then the final FDMHNN model was chosen from these candidates once the hyper-parameters J and O were decided.

Similar to the experiments performed in section 4.3.3, the average training loss values of the baseline MLP model based on different decomposition levels are depicted in Fig. 6.2. From the figures, it can be found that the decomposition level $J = 2$ is suitable for the WNN model in Mission A, while $J = 1$ will be the choice of the WNN model in Mission B. Particularly, the training loss values in all cases at the chosen decomposition level are compared in Fig. 6.3.

Fig. 6.3(a) shows that, in the case of choosing transformed dimension $O = 14$, the MLP model had the fastest convergence speed. In addition, the transformed dimension $O = 14$ was indeed quite lower than the original dimension 31, which will reduce the computational complexity dramatically and accelerate training.

In Fig. 6.3(b), the cases that had considerable low transformed dimension are plotted. There were two cases where the model converged faster at the beginning of the training process, i.e., $O = 12$ and $O = 16$, which have been highlighted in the figure. Even though the case of $O = 16$ converged a little bit faster than the case of $O = 12$ in the first few epochs, the latter one converged to a lower level. Besides, the transformed dimension 12 is much lower than the original dimension 41.

Therefore, based on the analysis, the proposed FDMHNN models were decided for two missions. The first one for Mission A, termed as FDMHNN_A, was to transform the original 31-dimensional CPP data to the 14-dimensional data at a two-higher resolution. Then the

MLP component with TFMH was trained on the transformed data and used to distinguish the Compton events from the pair production events. The second FDMHNN model for Mission B, termed as FDMHNN_B, was to transform the original 41-dimensional 3FGL data to the 12-dimensional data at a one-higher resolution. Then the trained MLP component with TFMH was used to identify the AGNs from the unassociated sources of 3FGL.

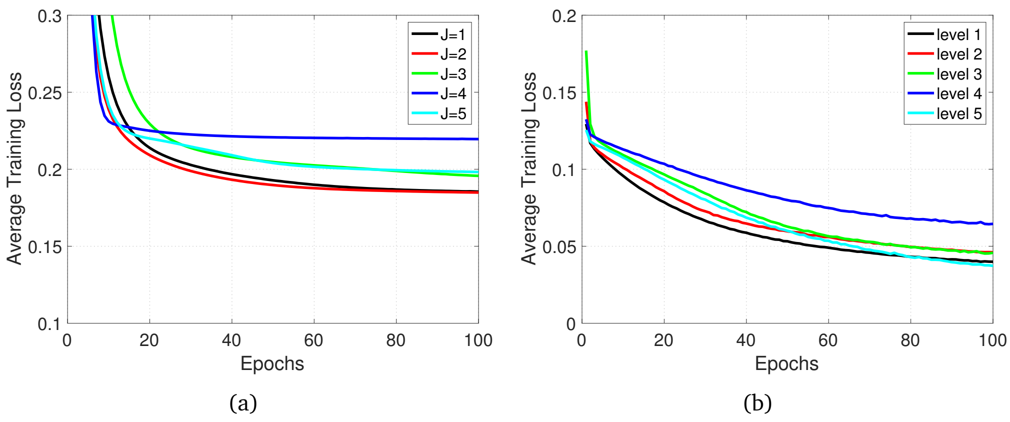


Figure 6.2: Average training loss values of the baseline MLP model with different decomposition levels J for (a) Mission A; (b) Mission B.

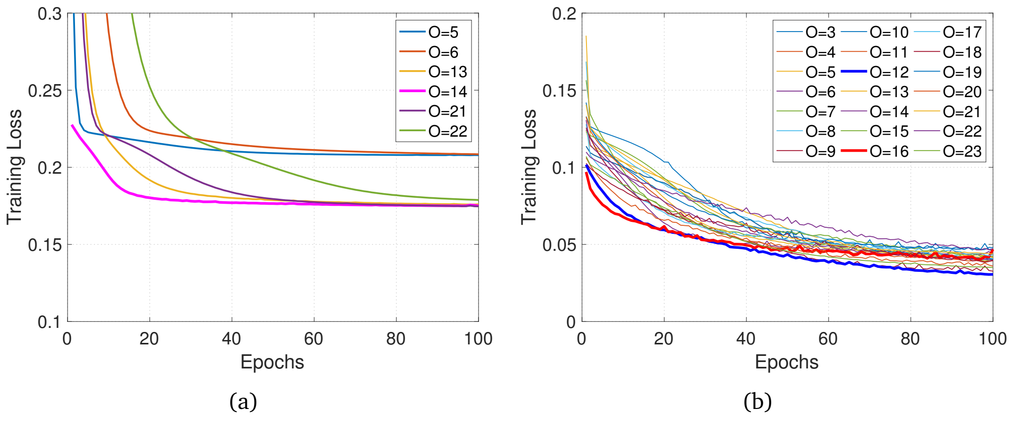


Figure 6.3: Training loss values of the baseline MLP model with a given decomposition level (a) $J = 2$ for Mission A; (b) $J = 1$ for Mission B.

6.2.3 Comparative Tests

Several models were built for comparative experiments to further investigate the performance of the proposed FDMHNN models.

Different ADFs

In the comparative experiments, the TFMH of the proposed FDMHNN model was compared with the ReLU family, i.e., ReLU, Leaky ReLU, and ELU. While Morlet and Sigmoid functions were not considered due to their poor performances, as discussed in section 5.5.

Different Preprocessing Methods

The preprocessing component of the FDMHNN model, i.e., FDIDWT, was also compared with other commonly used dimensionality reduction methods, including Attribute Selection (AS) and Principal Component Analysis (PCA). Additionally, the model performances on the original datasets without any preprocessing were also compared.

However, there exist hyper-parameters in the AS and PCA methods as well. The number of remaining attributes in the AS method must be decided in advance. Also, the number of components taken as principal should be predefined in the PCA method. Similar to the procedure used to determine the hyper-parameters of the FDIDWT method, the hyper-parameters of AS and PCA methods were also chosen from the candidates based on the performances of the baseline MLP model on the training dataset of the first group.

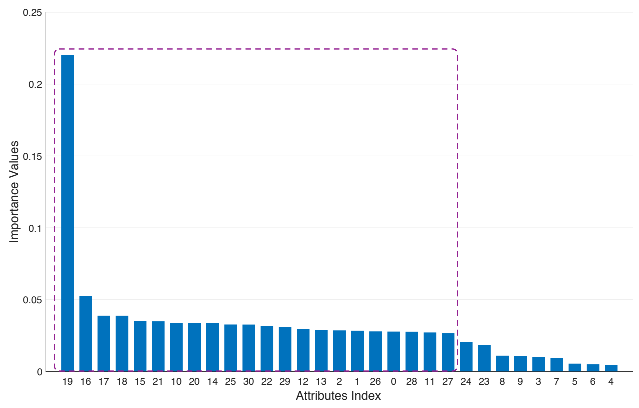
With the AS method, the importance values of attribute were calculated with a Random Forest (RF) model whose hyper-parameters are listed in Tab. 6.9. Then the importance values were sorted in descending order, as shown in Fig. 6.4. In order to reduce dimensionality, the attributes with low importance values are removed, and the remaining attributes are considered containing most information of the data. In the experiments, several amounts of remaining attributes were tested, i.e., (28,24,22) and (33,26,20,14) for CPP dataset and 3FGL, respectively. The loss values of the baseline MLP model with different choices in two missions are compared in Fig. 6.5.

Table 6.9: RF models used to calculate the attribute importance for the astronomical missions. Other parameters were set as default in scikit-learn package (Pedregosa et al., 2011).

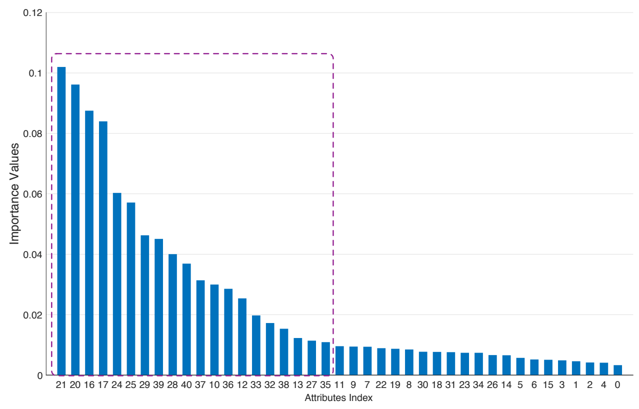
RF parameters	Mssion A	Mssion B
number of trees	10000	
criterions	gini	

Fig. 6.5(a) shows that, in Mission A, the convergence of the baseline models with different amounts of remaining attributes were comparable. While these models had slightly different accuracies, as listed in Tab. 6.10. Considering high accuracy score and low transformed dimension are expected, the number of remaining attributes in Mission A is chosen to be

22. From Fig. 6.5(b) and Tab. 6.10, it was found that the case of remaining 20 attributes achieved high accuracy and fast convergence, and then it was the choice for Mission B.



(a)



(b)

Figure 6.4: The descending normalized attribute importance of (a) CPP dataset and (b) 3FGL. The attributes selected by the rectangle are remained.

For the PCA method, the components with low variance values are removed for dimensionality reduction. The variances of the transformed datasets in two missions were normalized and sorted in descending order, as shown in Fig. 6.6. Similarly, several amounts of components with high variance values were considered being remained, i.e., (26,20,17) and (27,23,16) for CPP dataset and 3FGL, respectively. From Fig. 6.7 and Tab. 6.10, it can be found that remaining 20 and 23 components resulted in fast convergence and high accuracy in Mission A and B, respectively.

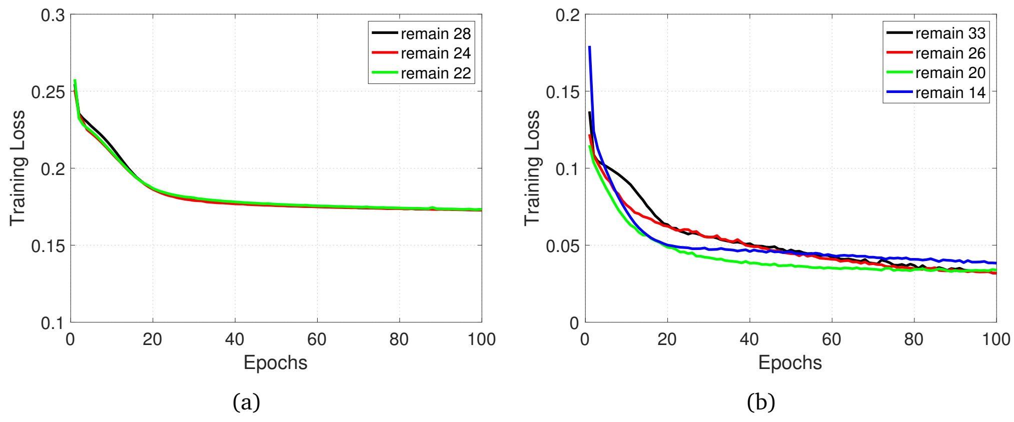


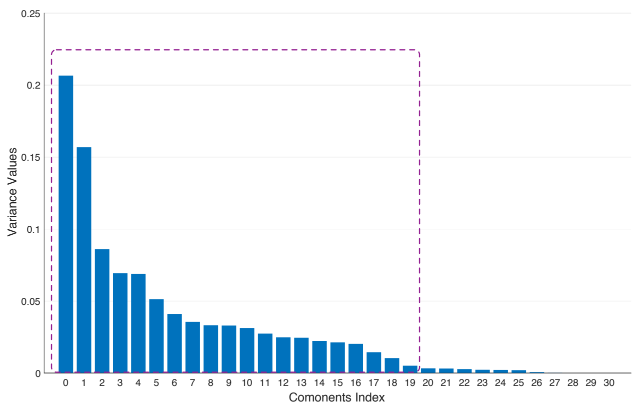
Figure 6.5: Training loss values of the baseline MLP model with different remaining attributes for (a) CPP dataset; (b) 3FGL.

Comparative Models

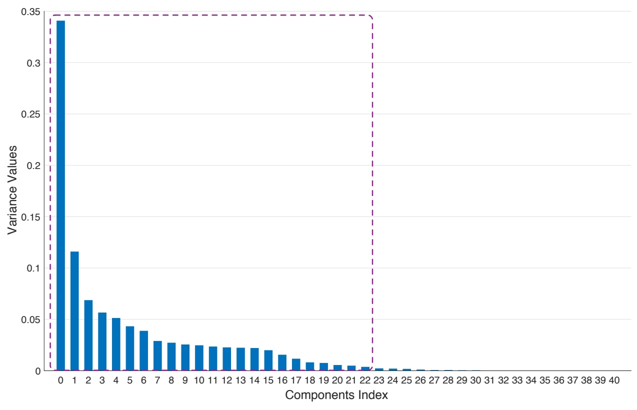
With all the comparative preprocessing components and ADFs have been decided, the models used to compare with the proposed FDMHNN model are summarized in Tab. 6.11 and 6.12 for Mission A and B, respectively. The structures and the hyper-parameters of the MLP component are the same as listed in Tab. 6.8. The training processes and accuracy results of these models will be reported and analyzed in next chapter.

Table 6.10: Accuracy results of the baseline MLP models with different hyper-parameters for AS and PCA methods.

Method	Mission A		Mission B	
	Number of remaining attributes or components	Accuracy	Number of remaining attributes or components	Accuracy
AS	28	75.49%	33	95.89%
	24	75.48%	26	97.20%
	22	75.70%	20	97.37%
			14	96.38%
PCA	26	74.92%	27	95.72%
	20	75.23%	23	95.82%
	17	75.11%	16	94.90%

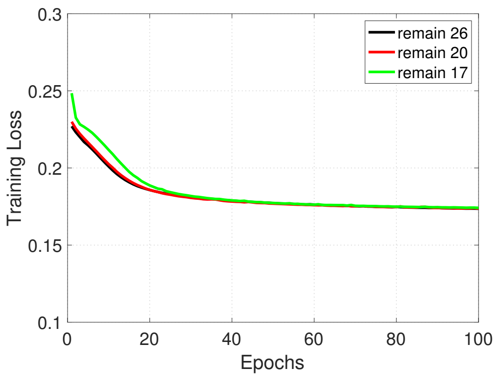


(a)

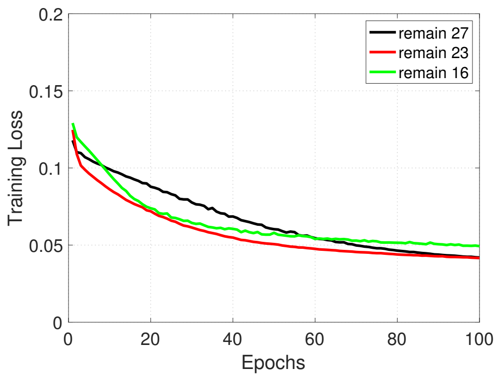


(b)

Figure 6.6: The descending normalized variance values of (a) CPP dataset; (b) 3FGL. The components selected by the rectangle are remained.



(a)



(b)

Figure 6.7: Training loss values of the baseline MLP model with different remaining components for (a) CPP dataset and (b) 3FGL.

Table 6.11: Comparative models for Mission A.

Model Name	Preprocessing Method	ADF	Hyper-parameters
AS_TFMH_A	AS	TFMH	remain 22 attributes
AS_ReLU_A		ReLU	
AS_LeReLU_A		LeReLU	
AS_ELU_A		ELU	
PCA_TFMH_A	PCA	TFMH	remain 20 components
PCA_ReLU_A		ReLU	
PCA_LeReLU_A		LeReLU	
PCA_ELU_A		ELU	
FDMHNN_A	FDIDWT	TFMH	$J = 2$
FDIDWT_ReLU_A		ReLU	$O = 14$
FDIDWT_LeReLU_A		LeReLU	
FDIDWT_ELU_A		ELU	
original_TFMH_A	none	TFMH	
original_ReLU_A		ReLU	
original_LeReLU_A		LeReLU	
original_ELU_A		ELU	

Table 6.12: Comparative models for Mission B.

Model Name	Preprocessing Method	ADF	Hyper-parameters
AS_TFMH_B	AS	TFMH	remain 20 attributes
AS_ReLU_B		ReLU	
AS_LeReLU_B		LeReLU	
AS_ELU_B		ELU	
PCA_TFMH_B	PCA	TFMH	remain 23 components
PCA_ReLU_B		ReLU	
PCA_LeReLU_B		LeReLU	
PCA_ELU_B		ELU	
FDMHNN_B	FDIDWT	TFMH	$J = 1$
FDIDWT_ReLU_B		ReLU	$O = 12$
FDIDWT_LeReLU_B		LeReLU	
FDIDWT_ELU_B		ELU	
original_TFMH_B	none	TFMH	
original_ReLU_B		ReLU	
original_LeReLU_B		LeReLU	
original_ELU_B		ELU	

7 Experimental Results and Discussions

In chapter 6, the extension experiments of the speech recognition tasks were built to study the convergence and stability of the proposed FFTNN model and the FFTMHNN model. The training performances of these two models with a deep structure are also interesting to compare.

Moreover, for the systematic experiments on astronomical data, a new topology of WNN model named as FDMHNN was proposed. Some models were also built for comparison in two binary tasks. The structures and hyper-parameters of these comparative models can be found in Tab. 6.8, 6.11, and 6.12. These models were evaluated on the remaining four groups of sub-datasets and the results were averaged for comparison. This chapter reports and analyzes the results coming from two astronomical missions independently. For each mission, the performances of models are compared concerning the preprocessing methods and the Activation Function and Derivative Function pair (ADF). The comparison of preprocessing methods performed among the proposed FDIDWT method, Attribute Selection (AS), and Principal Component Analysis (PCA). While the comparison of ADF performed between the proposed TFMH and the ReLU family ("relu" for ReLU, "lerelu" for Leaky ReLU, and "elu" for ELU). Model comparisons for accuracy and convergence are also given.

7.1 Results of the Extension Experiments of Speech Recognition

With the same number of hidden units and the same hyper-parameters excluding the ADF, the experimental results of the FFTNN and FFTMHNN models for the three missions described in section 3.4.2 are shown in Fig. 7.1. The figures also show the results of the corresponding MLP models without FFT component.

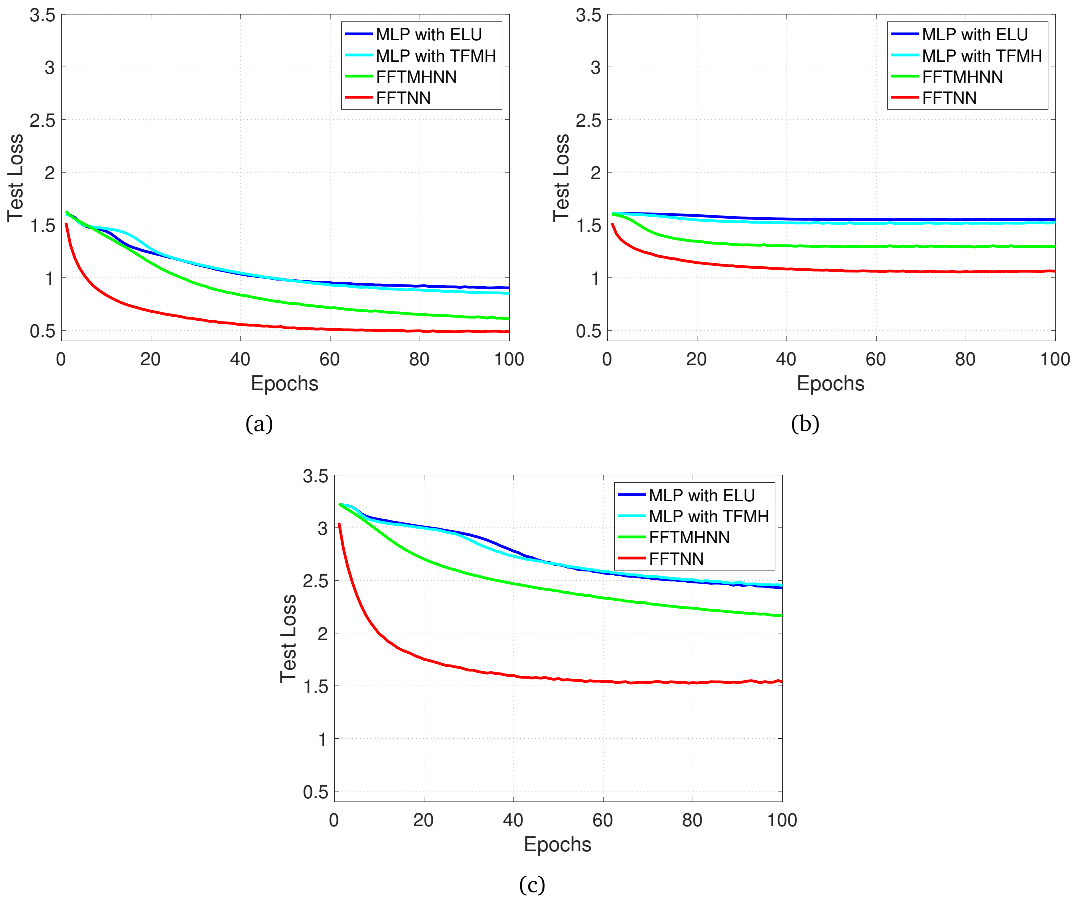
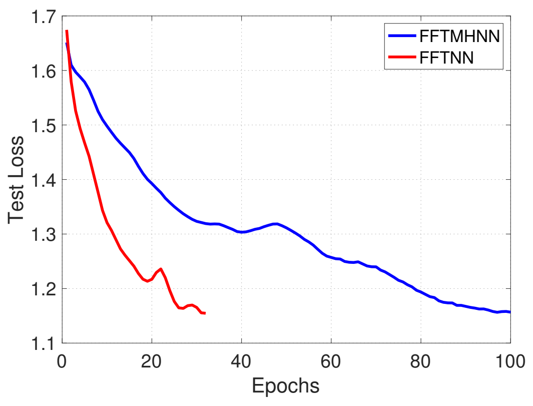


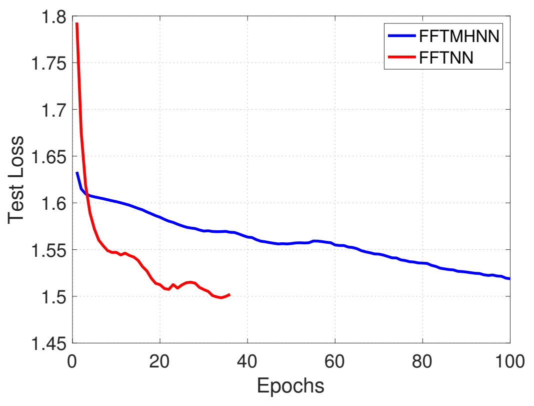
Figure 7.1: Test loss values of different comparative models in the extension experiments of speech recognition. (a) Mission A; (b) Mission B; (c) Mission C.

From the results it can be found that the TFMH case had the comparable performance with the elu function when the training was performed in time domain. But the elu outperformed the TFMH when training DNN in frequency domain, which could be attributed to the linearity of Fourier transform that might lose the nonlinear features of speech signal. In this case, the training with the nonlinear TFMH couldn't take fully advantage of the linear features extracted by FFT. While the elu case achieved better performance due to its linear property.

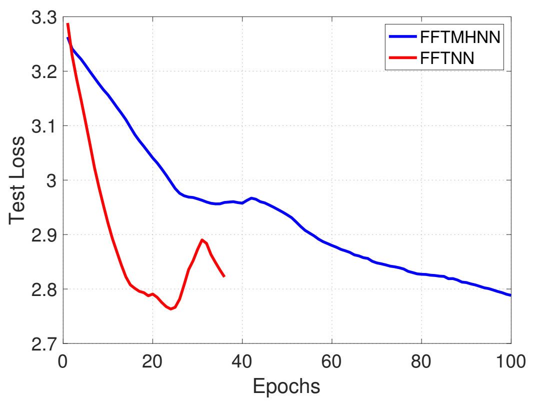
Nevertheless, the gradient exploding problem also existed in frequency domain, which can be seen from Fig. 7.2. Even though the learning rate was not set too high, the deep structure resulted in exploding gradients in the elu case. While the TFMH could stabilize the training, demonstrating its advantages in deep learning in frequency domain.



(a)



(b)



(c)

Figure 7.2: Test loss values of deep FFTMHNN and FFTNN models in the extension experiments of speech recognition. (a) Mission A; (b) Mission B; (c) Mission C. The unfinished plotting of green line means numerical overflow.

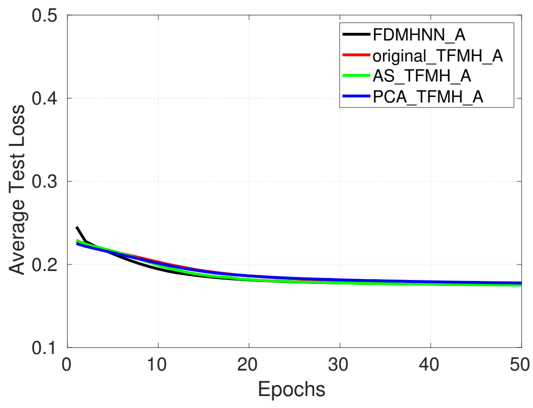
7.2 Results on CPP Dataset

7.2.1 Comparison of Preprocessing Methods

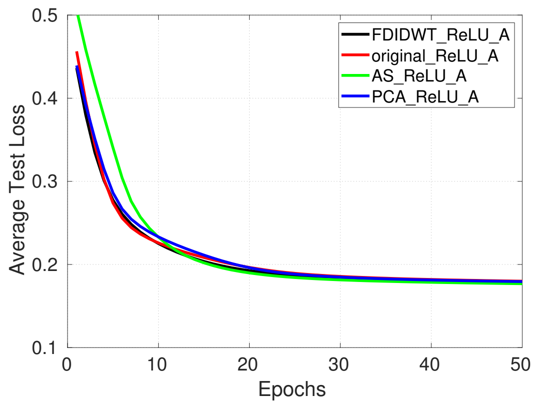
For a particular ADF, the model performances on CPP dataset with different preprocessing methods are illustrated in the subfigures of Fig. 7.3. These subfigures don't give much useful information about the performance of different preprocessing methods. The training performances were comparable in every ADF case. The slightly slow convergence of the models using the AS method, as shown in Fig. 7.3(b), might result from the information loss when ignoring the "unimportant" attributes directly.

7.2.2 Comparison of ADFs

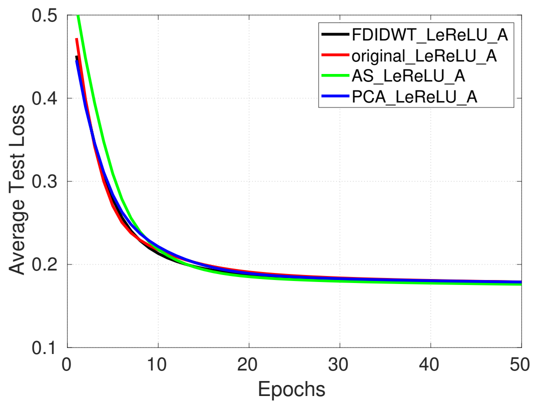
For a particular preprocessing method, the model performance on CPP dataset with different ADFs are illustrated in Fig. 7.4. From the results, it can be found that the model converged faster when equipped with TFMH in spite of the preprocessing method used. The training process could be seen as convergent after few epochs in TFMH case, while more epochs were needed for convergence in other ADF cases. Moreover, the model performance in relu case was a little bit worse due to the linear property.



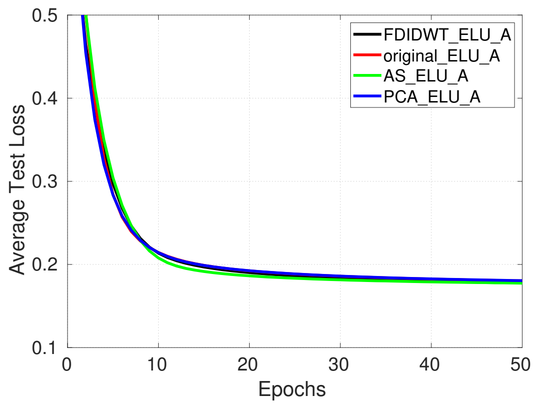
(a)



(b)



(c)



(d)

Figure 7.3: Average test loss values of the models for the CPP dataset with ADF (a) TFMH; (b) relu; (c) lerelu; (d) elu.

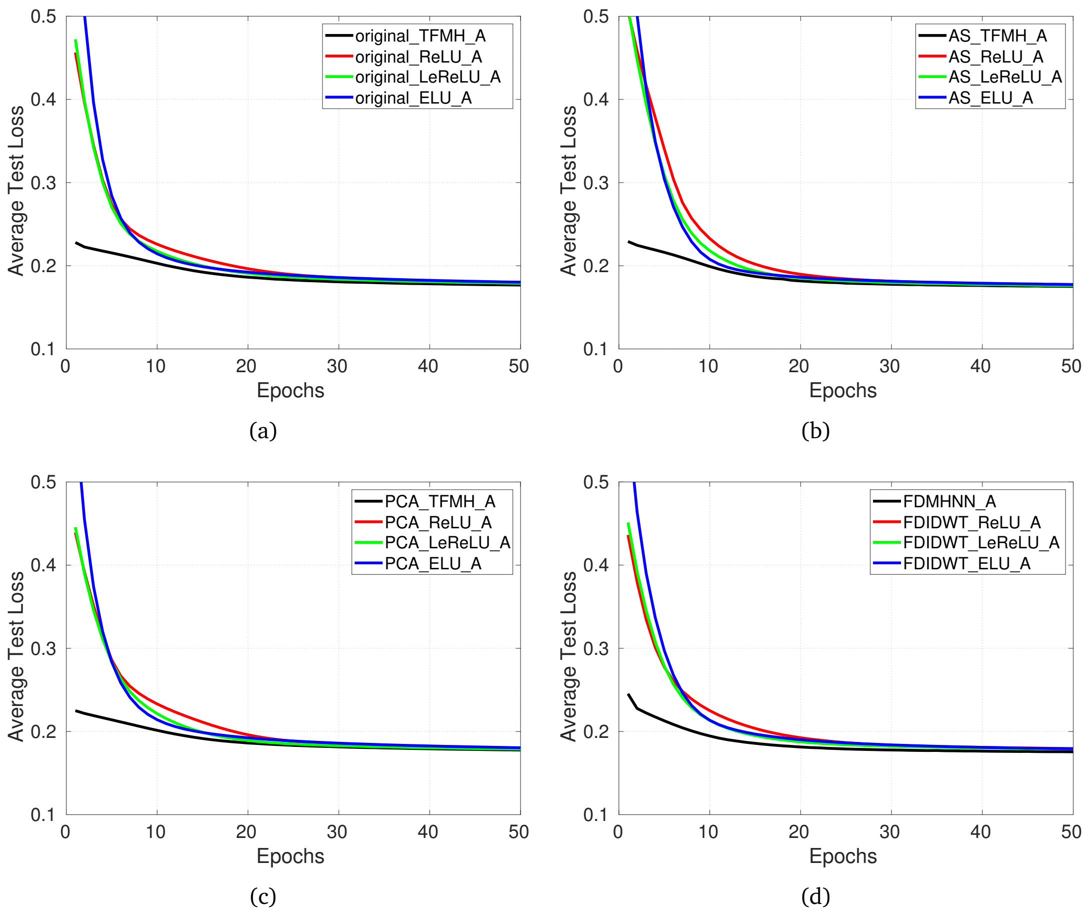


Figure 7.4: Average test loss values of the models for the CPP dataset after preprocessed by (a) no method; (b) AS; (c) PCA; (d) FDIDWT.

7.2.3 Systematic Comparison

To intuitively study the performance of the proposed FDMHNN model, the models that neither preprocessed by the FDIDWT method nor configured with TFMH are illustrated in Fig. 7.5. The figure shows that the FDMHNN model indeed had the fastest convergence speed, while it converged to the comparable loss level with other models.

Besides the training process, the accuracy scores of the convergent models are also compared in Tab. 7.1. The results show that all the models almost achieved the same accuracy score. The FDMHNN model seemed not to outperform other models concerning the accuracy measurement.

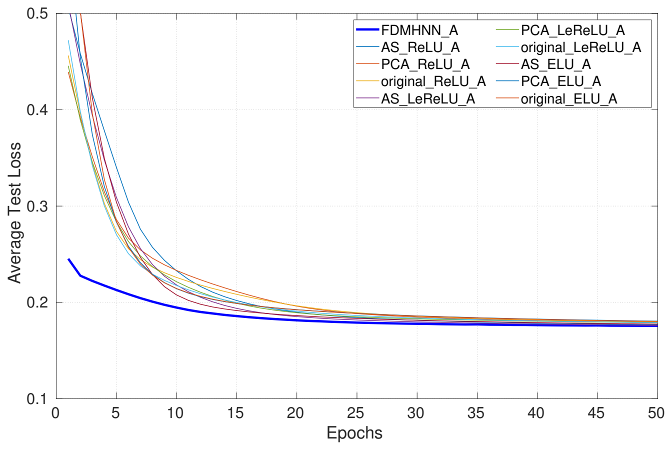


Figure 7.5: Average test loss values of the FDMHNN model and other models for the CPP dataset. The loss curve of the FDMHNN model has been highlighted with blue color and a thicker line.

Table 7.1: Accuracy results of the models in the astronomical missions.

Models	Missions	
	A	B
FDMHNN	75.11%	96.05%
AS_ReLU	75.18%	96.71%
AS_LeReLU	75.18%	96.88%
AS_ELU	75.13%	96.82%
PCA_ReLU	75.13%	94.74%
PCA_LeReLU	74.98%	95.72%
PCA_ELU	74.77%	95.23%
original_ReLU	74.84%	95.23%
original_LeReLU	74.90%	95.56%
original_ELU	74.97%	95.40%
Average	75.02%	95.83%
Standard Deviation	0.14%	0.71%

7.3 Results on 3FGL Samples

7.3.1 Comparison of Preprocessing Methods

Given an ADF, the model performance on 3FGL dataset with different preprocessing methods are compared in Fig. 7.6. From the results, it can be seen that the FDIDWT method performed worse than other preprocessing methods when working with ReLU family, but it got better results when working with TFMH. It is attributed that the features of data extracted by FDIDWT were nonlinear, and they couldn't be handled by the linear ADFs, such as relu and lerelu. While elu has modest nonlinearity, then these nonlinear features can be learnt normally in the elu case, as shown in Fig. 7.6(d). The nonlinear property of TFMH is good, thus the nonlinear features were fully used for training, as shown in Fig. 7.6(a).

PCA is a linear transform that will lose some nonlinear features, and then it led to bad performance when working with the nonlinear TFMH. In the experiment, AS methods performed better when combined with linear ReLU family, which shows that the features in the original dataset were mainly linear and luckily AS method kept the key linear features.

7.3.2 Comparison of ADFs

For each preprocessing method, the models with different ADFs are also compared in Fig. 7.7. For the ReLU family, the models with elu function had the best performance. While the model performance was the worst when using relu. It validates that the nonlinearity of elu is stronger than of relu. While regardless of the preprocessing method used, TFMH always achieved a low loss level after few epochs. It means that the TFMH helped the models to learn the features in a short time, which is really significant for fast DNN training. There were mainly linear features in the data preprocessed by AS and PCA, thus the linear lerelu and elu cases converged slightly faster than the TFMH case in the latter epochs, as shown in Fig. 7.7(b) and 7.7(c). But the models can be seen as already convergent in these epochs since the loss values were decreasing slowly. The models with TFMH tried to learn as many features as possible from the original dataset, which resulted in a better performance in Fig. 7.7(a). Moreover, when the nonlinear features in the original data were extracted by FDIDWT, the models with TFMH achieved the best performance, as shown in Fig. 7.7(d).

7.3.3 Systematic Comparison

The performance of the proposed FDMHNN model was also compared with others in Fig. 7.8. It can be seen that the FDMHNN model outperformed others on the 3FGL dataset as well. While the accuracy results of these models were comparable, as listed in Tab. 7.1.

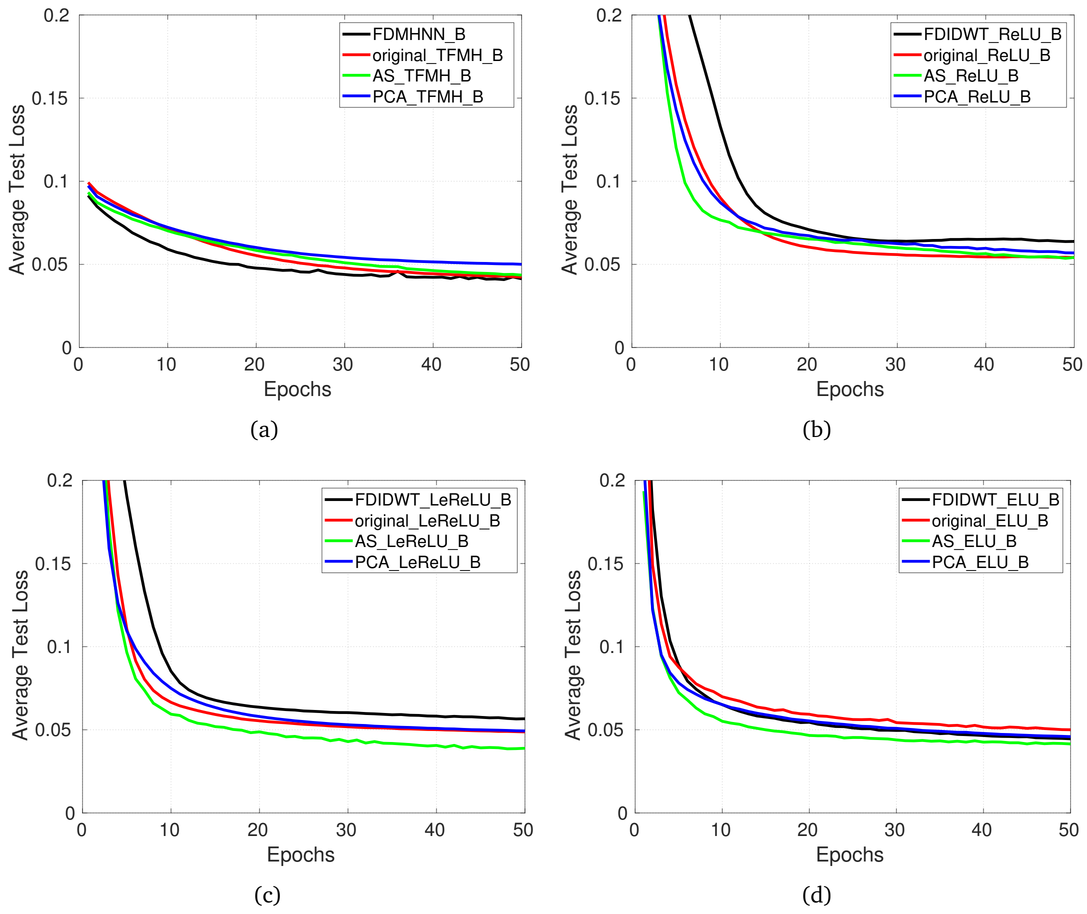
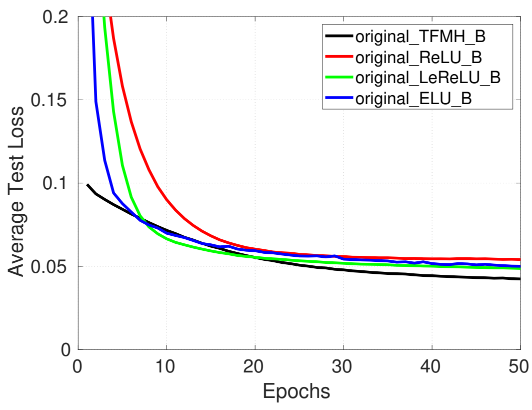
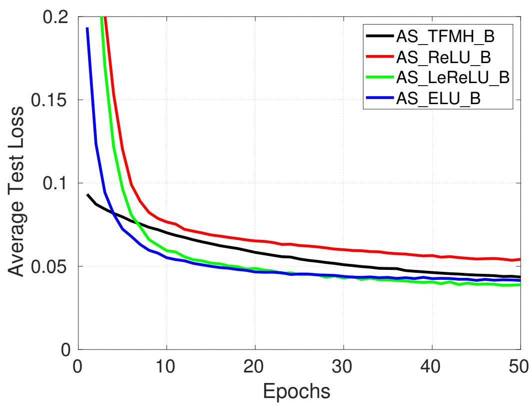


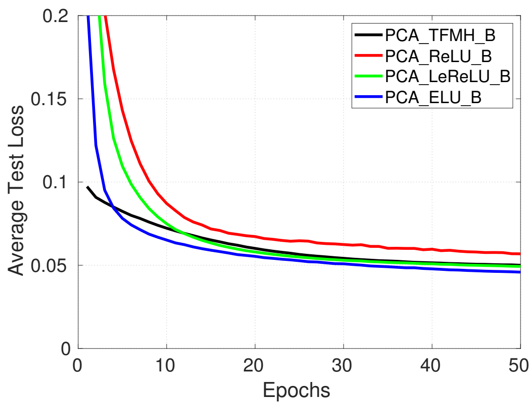
Figure 7.6: Average test loss values of the models for the 3FGL dataset with ADF (a) TFMH; (b) relu; (c) lerelu; (d) elu.



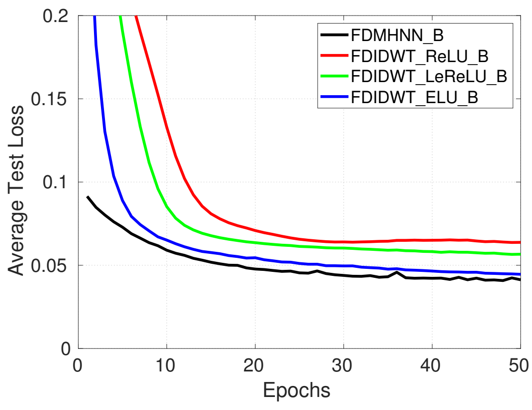
(a)



(b)



(c)



(d)

Figure 7.7: Average test loss values of the models for the 3FGL dataset after preprocessed by (a) no method; (b) AS; (c) PCA; (d) FDIDWT.

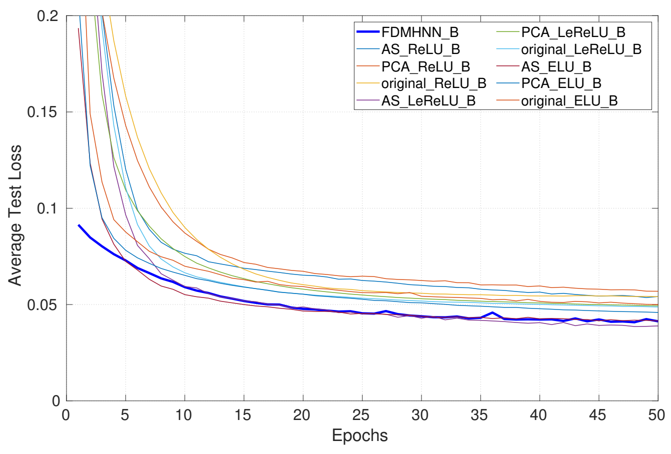


Figure 7.8: Average test loss values of the FDMHNN model and other models for the 3FGL dataset. The loss curve of the FDMHNN model has been highlighted with blue color and a thicker line.

7.4 Discussions

In the extension experiments of speech recognition, the TFMH was demonstrated to be of help to stabilize the DNN training in frequency domain. While the FFTNN model didn't get good result when replacing the elu with TFMH due to the nonlinearity of TFMH. Thus, the training performance of DNN also depends on the features given by the training data. In other words, the DNN with nonlinear ADF may achieve better performance when trained on the data dominated by nonlinear features, and vice versa.

For the systematic experiments on astronomical data, two FDMHNN models were designed to finish the classification tasks on the CPP dataset and 3FGL samples, respectively. They achieved faster convergence in training, which benefits from the feature pre-extraction capability of the FDIDWT method, and good properties of the TFMH.

The goal of finding the attribute set core and then estimating the information contents of attributes based on the correlation fractal dimension theory, is to construct the wavelet decomposition vector and bookkeeping vector for the next IDWT operation. While the results may be different if various correlation thresholds ξ are used. In FDIDWT method, the correlations among attributes are important since they provides the additional features of data for training. Thus more correlations are expected to be detected, which usually leads to a high correlation threshold.

IDWT is used to transform the data from a low resolution to a high resolution, where the key features of data may be extracted. Experimental results on the datasets transformed with different decomposition levels, shown in Fig. 6.2, validated this assumption. The case with the fastest convergence can be taken as the choice of the decomposition level J that is a hyper-parameter of the FDIDWT method.

Another hyper-parameter of the FDIWT method is the transformed dimension O. Besides the fast convergence, O should also be considered by the rule of low dimension. In other words, the FDIDWT component which leads to low transformed dimension and fast convergence will be taken as the choice for the FDMHNN model.

The FDIDWT method can be seen as a nonlinear transform that extracts the nonlinear features of data. These nonlinear features will be fully used for training when the model is activated by the nonlinear TFMH at the same time. While the linear ADFs, such as the ReLU family, have a weak capability to capture these nonlinear features, which leads to slow convergence.

However, in principle, the classification results are crucially decided by the dataset itself. For example, in the experiments on astronomical data, the CPP dataset in Mission A contained few discriminative features since the loss values always converged to a high level. It is the truth that the binary classes in CPP dataset, i.e., Compton events and pair production events, were detected in the same energy band. Additionally, the two events were also measured from the Compton interaction, which resulted in their similar features in some sense. But the 3FGL dataset collected the samples from different sources and it thus contained more discriminative information, which resulted in a lower loss level and higher accuracy score than on the CPP dataset.

FDMHNN model is designed with MLP structure, while the proposed FDIDWT method and TFMH function can be extended into other types of DNNs such as CNNs and GCNs, since FDIDWT is performed on datasets and TFMH is proposed from FP and BP processes that are independent of DNN structures. Besides, the model designed with CNNs or GCNs might achieve better performance than FDMHNN, since the weight sharing inherently introduces sparsity and accelerates training further. Moreover, TFMH has the ability to constrain the energy of propagating data, which makes the training in a very deep structure possible.

Furthermore, FDMHNN doesn't require fine-tuning process and it has few hyper-parameters, since it is proposed from solid theories and few researcher's experiences are required. Of course, the FDIDWT and TFMH can also be the considerations for other acceleration algorithms of DNN training.

In summary, the FDMHNN model has been demonstrated to be of great help to alleviate the gradient problems and accelerate the training process. It is suitable to cope with the complex problems that usually involve nonlinear features. However, in the systematic experiments, it didn't achieve considerable improvements on the accuracy when compared with other models. Even though this result can be ascribed to the linear features existing in the CPP dataset and 3FGL samples, it is interesting to further explore the FDMHNN model in the future.

8 Conclusions and Future Work

Deep Neural Network (DNN) training usually suffers from gradient problems and slow convergence speed. When the model goes deeper and deeper, the gradients may tend to zeros during training, where many model parameters cannot be updated. On the other hand, the unstable DNN training process has the risk of gradients exploding and gives rise to numerical overflow. In addition, DNN training requires many resources and time, thus slow convergence results in low computational efficiency.

It has been investigated that, the gradient problems can be attributed to the inappropriate chosen Activation Function and Derivative Function pair (ADF). The saturation ADF easily results in gradient vanishing that makes the training stick to a local minimum. And the unsaturation ADF cannot constrain the energy of training data, resulting in failure of training.

While the slow convergence speed of DNN training can be analyzed from two aspects. The first one comes from the characteristics of real-world datasets. The high dimensionality of data obviously increases the computational burden. Moreover, the existence of noise buries the key features and makes feature extraction difficult. Another analysis can also be focused on the ADF used. In particular, the nonzero-mean ADF causes the Internal Covariate Shift (ICS) or Back Internal Covariate Shift (BICS) that slows down training. Additionally, the linear ADF reduces the learning capability of DNN model for more complex applications.

Based on the analysis, this thesis proposes a new topology of Wavelet Neural Network (WNN), i.e., FDMHNN, for stable and fast deep learning. In particular, transforming the data to other domains is proposed to enhance the key feature extraction and noise reduction. For instance, the frequency representations after Fourier Transform (FT) will provide clear features for training. As a pathfinder topology, the proposed FFTNN model performs an end-to-end procedure combining Fast Fourier Transform (FFT) with DNN training. Experiments for howl noise reduction and speech recognition have demonstrated that feature-extracted datasets indeed help to accelerate deep learning.

Along this line, the proposed FDIDWT method takes the correlations of attributes as the additional features of data in wavelet domain and generates a low-dimensional and feature-extracted dataset for the next steps. Experiments have demonstrated that the FDIDWT method improves the accuracy in the classification tasks and speeds up DNN training.

Wavelet-based ADF is also demonstrated to be useful to accelerate the training process, which benefits from the localization, nonlinearity, and sparsity of wavelets. Additionally, taking advantage of the energy concentration of the tight frame wavelet, the training process of deep networks can be accelerated further by the proposed ADF, i.e., TFMH. Experimental results have demonstrated that TFMH performs well in different applications, and provides a better solution to fast DNN training. The extension experiments of the speech recognition tasks have also shown the advantages of TFMH on stabilizing the DNN training in frequency domain.

The complicated features of data can be fully extracted for training by combining the nonlinear FDIDWT preprocessing method with the Multilayer Perceptron (MLP) equipped with the nonlinear TFMH, to form the FDMHNN. Benefits from the feature pre-extraction capability of FDIDWT method and the good properties of TFMH, the FDMHNN model can cope with the real-world problems. This new WNN model is characterized as stable, fast construct and fast convergent. It builds a bridge between wavelet theory and Deep Learning (DL). Besides, this thesis uses wavelet analysis to build a multilayer model and to optimize the DNN structure. More possibilities of the combination of wavelet theory and DL for better performance in big data applications will be investigated in the future. The systematic experiments on the big astronomical data further indicated the advantages of FDMHNN model on fast convergence.

Hence, it is concluded that:

1) The proposed FFTNN learns more features in the frequency domain and helps to accelerate training.

2) The FDIDWT preprocessing method provides low-dimensional and feature-extracted dataset for DNN training with high computational efficiency and fast convergence.

3) The derived TFMH leads to stable and fast DNN training, which can be the alternative

ADF for many applications.

4) The data preprocessing step can be combined with the DNN structure to form a new topology optimized for high performance. And the proposed FDMHNN model is developed to achieve faster convergence than the traditional DNNs.

While there still exist some problems in the proposed model:

1) The strategy of weight initialization affects the start of DNN training partly, which produces different training results each time.

2) The data preprocessing with FDIDWT requires human intervention. It may be time-consuming. An end-to-end FDMHNN model is expected to develop for more applications.

3) The FDMHNN model cannot efficiently learn the linear features of data, which leads to its comparable performance with other test models.

4) The FDMHNN model is only designed with the MLP structure, which limits its application in many fields, such as image processing.

Therefore, the future work can be carried out as follows:

1) The weights initialization of DNN is an important step when training starts. A good weights initialization strategy provides a good start point on the hyper-plane and helps the training prevent from a local minimum. The wavelet basis can be used to initialize the weights, which is supposed to help the model learn features quickly due to the localization property of wavelet. This idea can be more easily understood when it is applied to initialize the kernel of Convolutional Neural Network (CNN), where the kernel acts as a 2D wavelet filter.

2) Besides the pyramid transform, Inverse Discrete Wavelet Transform (IDWT) can be further accelerated with a more efficient algorithm, i.e., the lifting scheme (Sweldens, 1996), which will make the training of the FDMHNN model possible on embedded devices. Moreover, the lifting scheme can also be implemented as the vector-matrix operations (Daubechies and Sweldens, 1998) within DNN for efficient feature pre-extraction, like the FFT component acting in the FFTNN model.

3) It is interesting to study the performance of the FDMHNN model with other types of DNNs, such as CNN and Graph Convolutional Network (GCN), instead of MLP.

4) Apart from the astronomical data dominated by linear features, more real-world datasets and tasks are to be tested for the possible optimizations of FDMHNN model.

5) It is also interesting to compare the performances of the FDMHNN model implemented on different hardware platforms, such as ARM, FPGA, and GPU (low-level for specified applications).

6) Package with C or C++ is needed to develop for efficient implementation of the proposed model on resource-limited platforms.

Acknowledgements

Time flies! I have not noticed that my Ph.D. study is coming to an end until I started to write this part of my thesis. Thanks to the joint Ph.D. agreements between Padova University of Italy and Guangzhou University of China, I have met a lot of lovely and impressive people in the past three years.

First I would like to thank Prof. Gaoyong Luo, who supports me to pursue a doctoral degree and guides me to be a solid researcher. He is a gentleman and he still works hard in scientific research although he has achieved a lot of success in the scientific domain. He is always patient to discuss research ideas with me and to review my scientific paper.

Next, I would like to thank my supervisor Prof. Alessandro Paccagnella. I enjoyed the research life during these three years because I always got the encouragements from Prof. Paccagnella. He is really earnest to give his suggestions on my paper. When I was in Padova, Prof. Paccagnella often concerned about my daily life, which was truly heartwarming.

I want to deeply thank Prof. Denis Bastieri and his family. Denis offered me an opportunity to start my Ph.D. study in Padova and helped me too much when I was in Italy. Also, his wife Elena Donati and their son Leandro took me as a family member of them, which gave me good memories in Padova. They taught me to love daily life and to become a friendly person.

I would like to thank Prof. Riccardo Rando and Dr. Denise Costantin, who helped me a lot with the knowledge of physics. Many thanks to my colleagues in Guangzhou, Dr. Sha Huan, and Dr. Zhong Cao, who gave me a lot of confidence in pursuing my doctoral degree.

I also want to thank all of my Ph.D. friends: Xiao Li, Chengying Bai, Qing Liu, Weijian Han, Yongheng Zhao, Huiting Huang, Yang Yang, Lishuai Jia, Xiye Peng, Hubing Xiao, and Xiyue Cai for their kindly help. We did many interesting things together in Padova. I would like to thank the companies of Jianlin Chen, Huaguo Gan, Hailong Lin, Yeguang Huang, Fakai Yan, Yifen Peng, and Juntao Huang when I was in Guangzhou. Without them, my Ph.D. career would not be so colorful.

Finally, I want to express my gratitude to my family, who always supported and encouraged me in the past three years. Many thanks to the people I met during my three-year Ph.D. study. They gave me happy memories of my life.

References

Abadi, M., **Barham, P.**, **Chen, J.**, **et al.** Tensorflow: a system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation*, volume 16, pages 265–283, 2016.

Acero, F., **Ackermann, M.**, **Ajello, M.**, **et al.** Fermi large area telescope third source catalog. *The Astrophysical Journal Supplement Series*, 218(2):23, 2015.

Adamowski, J. and **Chan, H. F.** A wavelet neural network conjunction model for groundwater level forecasting. *Journal of Hydrology*, 407(1-4):28–40, 2011.

Alemany, S., **Beltran, J.**, **Perez, A.**, and **Ganzfried, S.** Predicting hurricane trajectories using a recurrent neural network. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 468–475, 2019.

Alexandridis, A. K. and **Zapranis, A. D.** Wavelet neural networks: a practical guide. *Neural Networks*, 42:1–27, 2013.

Alexandropoulos, S.-A. N., **Kotsiantis, S. B.**, and **Vrahatis, M. N.** Data preprocessing in predictive data mining. *The Knowledge Engineering Review*, page e1, 2019.

Andrearczyk, V. and **Whelan, P. F.** Using filter banks in convolutional neural networks for texture classification. *Pattern Recognition Letters*, 84(63–69), 2016.

Andritschke, R., **Zoglauer, A.**, **Kanbach, G.**, **Bloser, P. F.**, and **Schopper, F.** The Compton and pair creation telescope MEGA. *Experimental Astronomy*, 20(1-3):395–403, 2005.

Arel, I., **Rose, D. C.**, **Karnowski, T. P.**, **et al.** Deep machine learning - a new frontier in artificial intelligence research. *IEEE Computational Intelligence Magazine*, 5(4):13–18, 2010.

Atwood, W., **Abdo, A. A.**, **Ackermann, M.**, **et al.** The large area telescope on the Fermi gamma-ray space telescope mission. *The Astrophysical Journal*, 697(2):1071, 2009.

Bahga, A. and **Madisetti, V.** *Big data science & analytics: a hands-on approach*. VPT, 2016.

Bakshi, B. R. and **Stephanopoulos, G.** Wave-net: a multiresolution, hierarchical neural network with localized learning. *AIChE Journal*, 39(1):57–81, 1993.

Belussi, A. and **Faloutsos, C.** Estimating the selectivity of spatial queries using the 'correlation' fractal dimension. In *Proceedings of the 21th International Conference on Very Large Data Bases*, pages 299–310. Morgan Kaufmann Publishers Inc., 1995.

Belussi, A. and **Faloutsos, C.** Self-spacial join selectivity estimation using fractal concepts. *ACM Transactions on Information Systems*, 16(2):161–201, 1998.

Bengio, Y., **LeCun, Y.**, **et al.** Scaling learning algorithms towards AI. *Large-scale Kernel Machines*, 34(5):1–41, 2007.

Bengio, Y., **Simard, P.**, and **Frasconi, P.** Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

Bengio, Y. et al. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2(1):1–127, 2009.

Benzy, V. and **Jasmin, E.** A combined wavelet and neural network based model for classifying depth of anaesthesia. *Procedia Computer Science*, 46:1610–1617, 2015.

Berlato, F. *Design and optimization around 1 MeV of a calorimeter for a CubeSat mission*. Master's thesis, Università degli Studi di Padova, 2016.

Billings, S. A. and **Wei, H.-L.** A new class of wavelet networks for nonlinear system identification. *IEEE Transactions on Neural Networks*, 16(4):862–874, 2005.

Boureau, Y.-L., **Ponce, J.**, and **LeCun, Y.** A theoretical analysis of feature pooling in visual recognition. In *Proceedings of the 27th International Conference on Machine Learning*, pages 111–118, 2010.

Boyd, S., **Parikh, N.**, **Chu, E.**, **et al.** Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine Learning*, 3(1):1–122, 2011.

Breiman, L. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

Breiman, L. Random forests. *Machine Learning*, 45(1):5–32, 2001.

Breiman, L. *Classification and regression trees*. Routledge, 2017.

Bu, Z., **Li, H.**, **Cao, J.**, **Wang, Z.**, and **Gao, G.** Dynamic cluster formation game for attributed graph clustering. *IEEE Transactions on Cybernetics*, 49(1):328–341, 2017.

Bylina, B., **Bylina, J.**, and **Karwacki, M.** An effective sparse storage scheme for GPU-enabled uniformization method. In *2018 Federated Conference on Computer Science and Information Systems*, pages 307–310. IEEE, 2018.

Calvo, R. A., **Partridge, M.**, and **Jabri, M. A.** A comparative study of principal component analysis techniques. In *Proceedings of the 9th Australian Conference on Neural Networks*, pages 276–281. Brisbane, Australia, 1998.

Cao, H. and **Luo, G.** Wavelet speech enhancement based on improved Teager energy operator and local variance analysis. In *Artificial Intelligence Science and Technology*, pages 541–552, 2016.

Cao, H., **Bastieri, D.**, **Rando, R.**, **et al.** Machine learning on Compton event identification for a nano-satellite mission. *Experimental Astronomy*, 47(1-2):129–144, 2019.

Chaabane, C. B., **Mellouli, D.**, **Hamdani, T. M.**, **Alimi, A. M.**, and **Abraham, A.** Wavelet convolutional neural networks for handwritten digits recognition. In *International Conference on Health Information Science*, pages 305–310. Springer, 2017.

Chang, C.-C. and **Lin, C.-J.** Libsvm: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2(3):27, 2011.

Chattopadhyay, T., **Falcon, A. D.**, **Burrows, D. N.**, **Fox, D. B.**, and **Palmer, D.** BlackCAT CubeSat: a soft X-ray sky monitor, transient finder, and burst detector for high-energy and multimessenger astophysics. In *Space Telescopes and Instrumentation 2018: Ultraviolet to Gamma Ray*, volume 10699, page 106995S. International Society for Optics and Photonics, 2018.

Chen, X., **Li, L.-J.**, **Fei-Fei, L.**, and **Gupta, A.** Iterative visual reasoning beyond convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7239–7248, 2018.

Chen, Y., **Luh, P. B.**, **Guan, C.**, **et al.** Short-term load forecasting: similar day-based wavelet neural networks. *IEEE Transactions on Power Systems*, 25(1):322–330, 2010.

Chiaro, G., **Salvetti, D.**, **La Mura, G.**, **et al.** Blazar flaring patterns (B-FlaP) classifying blazar candidate of uncertain type in the third Fermi-LAT catalogue by artificial neural networks. *Monthly Notices of the Royal Astronomical Society*, 462(3):3180–3195, 2016.

Chitsaz, H., **Amjady, N.**, and **Zareipour, H.** Wind power forecast using wavelet neural network trained by improved clonal selection algorithm. *Energy Conversion and Management*, 89:588–598, 2015.

Chui, C. K. and **Shi, X.** Inequalities of Littlewood-Paley type for frames and wavelets. *SIAM Journal on Mathematical Analysis*, 24(1):263–277, 1993.

Clevert, D.-A., **Unterthiner, T.**, and **Hochreiter, S.** Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

Cooley, J. W. and **Tukey, J. W.** An algorithm for the machine calculation of complex Fourier series. *Mathematics of Computation*, 19(90):297–301, 1965.

Cortes, C. and **Vapnik, V.** Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

Cover, T. M., **Hart, P.**, **et al.** Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(13):21–27, 1967.

Crammer, K. and **Singer, Y.** On the algorithmic implementation of multiclass kernel-based vector machines. *Journal of Machine Learning Research*, 2(Dec):265–292, 2001.

Dalton, S., **Olson, L.**, and **Bell, N.** Optimizing sparse matrix-matrix multiplication for the GPU. *ACM Transactions on Mathematical Software*, 41(4):25, 2015.

Daubechies, I. The wavelet transform, time-frequency localization and signal analysis. *IEEE Transactions on Information Theory*, 36(5):961–1005, 1990.

Daubechies, I. *Ten lectures on wavelets*, volume 61, chapter 3, pages 53–105. SIAM, 1992.

Daubechies, I. and **Sweldens, W.** Factoring wavelet transforms into lifting steps. *Journal of Fourier Analysis and Applications*, 4(3):247–269, 1998.

Dauphin, Y. N. and **Bengio, Y.** Big neural networks waste capacity. *arXiv preprint arXiv:1301.3583*, 2013.

De Angelis, A., **Tatischeff, V.**, **Tavani, M.**, **et al.** The e-ASTROGAM mission. *Experimental Astronomy*, 44(1):25–82, 2017.

de Sousa, E. P. M., **Traina Jr, C.**, **Traina, A. J. M.**, **Wu, L.**, and **Faloutsos, C.** A fast and effective method to find correlations among attributes in databases. *Data Mining and Knowledge Discovery*, 14(3):367–407, 2007.

Denton, E. L., **Chintala, S.**, **Fergus, R.**, **et al.** Deep generative image models using a Laplacian pyramid of adversarial networks. In *Advances in Neural Information Processing Systems*, pages 1486–1494, 2015.

Dietterich, T. G. *The handbook of brain theory and neural networks*. MIT Press, 2002.

Dong, W., **Wang, P.**, **Yin, W.**, and **Shi, G.** Denoising prior driven deep neural network for image restoration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2018.

Donoho, D. L. Unconditional bases are optimal bases for data compression and for statistical estimation. *Applied and Computational Harmonic Analysis*, 1(1):100–115, 1993.

Doucoure, B., **Agbossou, K.**, and **Cardenas, A.** Time series prediction using artificial wavelet neural network and multi-resolution analysis: application to wind speed data. *Renewable Energy*, 92:202–211, 2016.

Dua, D. and **Graff, C.** UCI machine learning repository. http://archive.ics.uci.edu/ml, 2017.

Duchi, J., **Hazan, E.**, and **Singer, Y.** Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

Elafrou, A., **Karakasis, V.**, **Gkountouvas, T.**, **et al.** Sparsex: a library for high-performance sparse matrix-vector multiplication on multicore platforms. *ACM Transactions on Mathematical Software*, 44(3):26, 2018.

Erhan, D., **Manzagol, P.-A.**, **Bengio, Y.**, **Bengio, S.**, and **Vincent, P.** The difficulty of training deep architectures and the effect of unsupervised pre-training. In *Artificial Intelligence and Statistics*, pages 153–160, 2009.

Erhan, D., **Bengio, Y.**, **Courville, A. C.**, **et al.** Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, 11(3):625–660, 2010.

Fichtel, C. E., **Bertsch, D. L.**, **Dingus, B.**, **et al.** Results from the Energetic Gamma-Ray Experiment Telescope (EGRET) on the Compton observatory. *Advances in Space Research*, 13(12):637–646, 1993.

Fiore, F., **Burderi, L.**, **Di Salvo, T.**, **et al.** HERMES: a swarm of nano-satellites for high energy astrophysics and fundamental physics. In *Space Telescopes and Instrumentation 2018: Ultraviolet to Gamma Ray*, volume 10699, page 106992Q. International Society for Optics and Photonics, 2018.

Freund, Y. and **Schapire, R. E.** A desicion-theoretic generalization of on-line learning and an application to boosting. In *European Conference on Computational Learning Theory*, pages 23–37, 1995.

Freund, Y. and **Schapire, R. E.** A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

Fujieda, S., **Takayama, K.**, and **Hachisuka, T.** Wavelet convolutional neural networks for texture classification. *arXiv preprint arXiv:1707.07394*, 2017.

Fujieda, S., **Takayama, K.**, and **Hachisuka, T.** Wavelet convolutional neural networks. *arXiv preprint arXiv:1805.08620*, 2018.

Fukushima, K. 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.

Gan, G. and **Ng, M. K.-P.** K-means clustering with outlier removal. *Pattern Recognition Letters*, 90:8–14, 2017.

Ganjefar, S. and **Tofighi, M.** Single-hidden-layer fuzzy recurrent wavelet neural network: applications to function approximation and system identification. *Information Sciences*, 294:269–285, 2015.

Garrido Beltrán, L., **Gascón Fora, D.**, **Graciani Díaz, R.**, **et al.** Design concepts for the Cherenkov telescope array CTA: an advanced facility for ground-based high-energy gamma-ray astronomy. *Experimental Astronomy*, 32(3):193–316, 2011.

Gehrels, N., **Chincarini, G.**, **Giommi, P.**, **et al.** The Swift gamma-ray burst mission. *The Astrophysical Journal*, 611(2):1005, 2004.

Géron, A. *Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems*. O'Reilly Media, Inc., 2017.

Glorot, X. and **Bengio, Y.** Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics*, pages 249–256, 2010.

Glorot, X., **Bordes, A.**, and **Bengio, Y.** Deep sparse rectifier neural networks. In *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics*, pages 315–323, 2011.

Goodfellow, I., **Bengio, Y.**, and **Courville, A.** *Deep learning*. MIT press, 2016.

Goodfellow, I., **Pouget-Abadie, J.**, **Mirza, M.**, **et al.** Generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 2672–2680, 2014.

Han, J., **Pei, J.**, and **Kamber, M.** *Data mining: concepts and techniques*. Elsevier, 2011.

Harrison, F. A., **Craig, W. W.**, **Christensen, F. E.**, **et al.** The nuclear spectroscopic telescope array (NuSTAR) high-energy X-ray mission. *The Astrophysical Journal*, 770(2):103, 2013.

Hastie, T., **Tibshirani, R.**, and **Friedman, J.** *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media, 2009.

He, K., **Lian, H.**, **Ma, S.**, and **Huang, J. Z.** Dimensionality reduction and variable selection in multivariate varying - coefficient models with a large number of covariates. *Journal of the American Statistical Association*, 113(522):746–754, 2018.

He, K., **Zhang, X.**, **Ren, S.**, and **Sun, J.** Delving deep into rectifiers: surpassing human-level performance on imagenet classification. In *2015 IEEE International Conference on Computer Vision*, pages 1026–1034, 2015.

He, K., **Zhang, X.**, **Ren, S.**, and **Sun, J.** Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

Hinton, G. E. and **Salakhutdinov, R. R.** Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

Hinton, J. Ground-based gamma-ray astronomy with Cherenkov telescopes. *New Journal of Physics*, 11(5):055005, 2009.

Hofmann, T., **Schölkopf, B.**, and **Smola, A. J.** Kernel methods in machine learning. *The Annals of Atatistics*, pages 1171–1220, 2008.

Howard, A. G., **Zhu, M.**, **Chen, B.**, **et al.** Mobilenets: efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

Hu, Y. and **Loizou, P. C.** Subjective comparison and evaluation of speech enhancement algorithms. *Speech Communication*, 49(7-8):588–601, 2007.

Huang, J.-p., **Zhu, L.-k.**, **Wang, L.-h.**, and **Song, W.-l.** Compressed sensing MRI using sparsity averaging and FISTA. *Applied Magnetic Resonance*, 48(8):749–760, 2017.

Hubel, D. H. Single unit activity in striate cortex of unrestrained cats. *The Journal of Physiology*, 147(2):226–238, 1959.

Hubel, D. H. and **Wiesel, T. N.** Receptive fields of single neurones in the cat's striate cortex. *The Journal of Physiology*, 148(3):574–591, 1959.

Hubel, D. H. and **Wiesel, T. N.** Receptive fields and functional architecture of monkey striate cortex. *The Journal of Physiology*, 195(1):215–243, 1968.

Ioffe, S. and **Szegedy, C.** Batch normalization: accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015.

Janocha, K. and **Czarnecki, W. M.** On loss functions for deep neural networks in classification. *Schedae Informaticae*, 25:49–59, 2016.

Jarrett, K., **Kavukcuoglu, K.**, **LeCun, Y.**, **et al.** What is the best multi-stage architecture for object recognition? In *2009 IEEE 12th International Conference on Computer Vision*, pages 2146–2153. IEEE, 2009.

Jiang, D., **Tang, C.**, and **Zhang, A.** Cluster analysis for gene expression data: a survey. *IEEE Transactions on Knowledge and Data Engineering*, 16(11):1370–1386, 2004.

Jifara, W., **Jiang, F.**, **Rho, S.**, **Cheng, M.**, and **Liu, S.** Medical image denoising using convolutional neural network: a residual learning approach. *The Journal of Supercomputing*, 75(2):704–718, 2019.

Jiménez, D. Dynamically weighted ensemble neural networks for classification. In *IEEE International Joint Conference on Neural Networks Proceedings*, volume 1, pages 753–756. IEEE, 1998.

Joachims, T. Making large-scale support vector machine learning practical. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods*, pages 169–184. Cambridge, MA, USA: MIT Press, 1999.

Jolliffe, I. T. and **Cadima, J.** Principal component analysis: a review and recent developments. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 374(2065):20150202, 2016.

Jordan, M. I. and **Mitchell, T. M.** Machine learning: trends, perspectives, and prospects. *Science*, 349(6245):255–260, 2015.

Kam, H. T. Random decision forest. In *Proceedings of the 3rd International Conference on Document Analysis and Recognition*, pages 14–18, 1995.

Kaul, A., **Maheshwary, S.**, and **Pudi, V.** Autolearn - automated feature generation and selection. In *2017 IEEE International Conference on Data Mining*, pages 217–226. IEEE, 2017.

Kim, Y.-D., **Park, E.**, **Yoo, S.**, **et al.** Compression of deep convolutional neural networks for fast and low power mobile applications. *arXiv preprint arXiv:1511.06530*, 2015.

Kingma, D. P. and **Ba, J.** Adam: a method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Korn, F., **Pagel, B.-U.**, and **Faloutsos, C.** On the "dimensionality curse" and the "self-similarity blessing". *IEEE Transactions on Knowledge and Data Engineering*, 13(1):96–111, 2001.

Koushik, J. and **Hayashi, H.** Improving stochastic gradient descent with feedback. *arXiv preprint arXiv:1611.01505*, 2016.

Krizhevsky, A., **Sutskever, I.**, and **Hinton, G. E.** Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012.

Kuang, L., **Yang, L. T.**, **Chen, J.**, **Hao, F.**, and **Luo, C.** A holistic approach for distributed dimensionality reduction of big data. *IEEE Transactions on Cloud Computing*, 6(2):506–518, 2015.

Kukačka, J., **Golkov, V.**, and **Cremers, D.** Regularization for deep learning: a taxonomy. *arXiv preprint arXiv:1710.10686*, 2017.

Lam, M. W., **Chen, X.**, **Hu, S.**, **et al.** Gaussian process lstm recurrent neural network language models for speech recognition. In *2019 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 7235–7239, 2019.

Langr, D. and **Tvrdik, P.** Evaluation criteria for sparse matrix storage formats. *IEEE Transactions on Parallel and Distributed Systems*, 27(2):428–440, 2015.

Le, Q. V., **Jaitly, N.**, and **Hinton, G. E.** A simple way to initialize recurrent networks of rectified linear units. *arXiv preprint arXiv:1504.00941*, 2015.

Lebedev, V., **Ganin, Y.**, **Rakhuba, M.**, **Oseledets, I.**, and **Lempitsky, V.** Speeding-up convolutional neural networks using fine-tuned cp-decomposition. *arXiv preprint arXiv:1412.6553*, 2014.

LeCun, Y., **Bottou, L.**, **Bengio, Y.**, and **Haffner, P.** Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

LeCun, Y. A., **Bottou, L.**, **Orr, G. B.**, and **Müller, K.-R.** Efficient backprop. In *Neural Networks: Tricks of the Trade*, pages 9–48. Springer, 2012.

Lefaucheur, J. and **Pita, S.** Research and characterisation of blazar candidates among the Fermi-LAT 3FGL catalogue using multivariate classifications. *Astronomy & Astrophysics*, 602:A86, 2017.

Leng, C., **Dou, Z.**, **Li, H.**, **Zhu, S.**, and **Jin, R.** Extremely low bit neural network: squeeze the last bit out with ADMM. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Li, M., **Zhang, T.**, **Chen, Y.**, and **Smola, A. J.** Efficient mini-batch training for stochastic optimization. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 661–670, 2014.

Li, R., **Wang, S.**, **Zhu, F.**, and **Huang, J.** Adaptive graph convolutional neural networks. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Li, Y., **Huang, J.-B.**, **Ahuja, N.**, and **Yang, M.-H.** Joint image filtering with deep convolutional networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41(8):1909–1923, 2019.

Lin, H., **Luo, G.**, **Cao, H.**, **Fang, X.**, and **Zhou, F.** Complex nonlinear system modelling and parameters identification by deep neural networks. *DEStech Transactions on Computer Science and Engineering*, (AICAE), 2019.

Liu, H., **Wang, D.**, **Liu, J.**, and **Liu, S.** Range tunable optical fiber micro-fabry–pérot interferometer for pressure sensing. *IEEE Photonics Technology Letters*, 28(4):402–405, 2015.

Liu, W., **Wang, Z.**, **Liu, X.**, **et al.** A survey of deep neural network architectures and their applications. *Neurocomputing*, 234:11–26, 2017a.

Liu, Z., **Li, J.**, **Shen, Z.**, **et al.** Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2736–2744, 2017b.

Loizou, P. C. *Speech enhancement: theory and practice*. CRC press, 2013.

Lokesh, S., **Kumar, P. M.**, **Devi, M. R.**, **Parthasarathy, P.**, and **Gokulnath, C.** An automatic tamil speech recognition system by using bidirectional recurrent neural network with self-organizing map. *Neural Computing and Applications*, 31(5):1521–1531, 2019.

Lu, T., **Li, S.**, **Fang, L.**, **Ma, Y.**, and **Benediktsson, J. A.** Spectral - spatial adaptive sparse representation for hyperspectral image denoising. *IEEE Transactions on Geoscience and Remote Sensing*, 54(1):373–385, 2015.

Lucchetta, G. *Design and optimization around 1 MeV of a tracker for a CubeSat mission*. Master's thesis, Università degli Studi di Padova, 2016.

Lucchetta, G., **Berlato, F.**, **Rando, R.**, **Bastieri, D.**, and **Urso, G.** Scientific performance of a nano-satellite Mev telescope. *The Astronomical Journal*, 153(5):237, 2017.

Luo, G. An efficient DSP implantation of wavelet audio coding for digital communication. In *4th International Conference on Digital Society*, pages 66–71, 2010.

Luo, G., **Osypiw, D.**, and **Hudson, C.** Real-time wavelet denoising with edge enhancement for medical X-ray imaging. In *Real-Time Image Processing 2006*, volume 6063, page 606303. International Society for Optics and Photonics, 2006.

Maas, A. L., **Hannun, A. Y.**, and **Ng, A. Y.** Rectifier nonlinearities improve neural network acoustic models. In *Proceedings of the 30th International Conference on Machine Learning*, volume 30, page 3, 2013.

Maheshwary, S., **Kaul, A.**, and **Pudi, V.** Data driven feature learning. In *International Conference on Machine Learning Workshop*, 2017.

Mallat, S. G. A theory for multiresolution signal decomposition: the wavelet representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(7):674–693, 1989.

McCulloch, W. S. and **Pitts, W.** A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5(4):115–133, 1943.

Mizumoto, T., **Matsuoka, Y.**, **Mizumura, Y.**, **et al.** New readout and data-acquisition system in an electron-tracking Compton camera for MeV gamma-ray astronomy (SMILE-II). *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 800:40–50, 2015.

Montanari, L., **Basu, B.**, **Spagnoli, A.**, and **Broderick, B. M.** A padding method to reduce edge effects for enhanced damage identification using wavelet analysis. *Mechanical Systems and Signal Processing*, 52:264–277, 2015.

Mukaka, M. M. A guide to appropriate use of correlation coefficient in medical research. *Malawi Medical Journal*, 24(3):69–71, 2012.

Nair, V. and **Hinton, G. E.** Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning*, pages 807–814, 2010.

Nielsen, M. A. *Neural networks and deep learning*, volume 25. Determination press San Francisco, CA, USA:, 2015.

Novikoff, A. B. On convergence proofs for perceptrons. Technical report, STANFORD RESEARCH INST MENLO PARK CA, 1963.

Novikov, A., **Podoprikhin, D.**, **Osokin, A.**, and **Vetrov, D. P.** Tensorizing neural networks. In *Advances in Neural Information Processing Systems*, pages 442–450, 2015.

Oba, S., **Sato, M.-a.**, **Takemasa, I.**, **et al.** A bayesian missing value estimation method for gene expression profile data. *Bioinformatics*, 19(16):2088–2096, 2003.

Odena, A., **Olah, C.**, and **Shlens, J.** Conditional image synthesis with auxiliary classifier GANs. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70, pages 2642–2651, 2017.

Omerhodzic, I., **Avdakovic, S.**, **Nuhanovic, A.**, and **Dizdarevic, K.** Energy distribution of EEG signals: EEG signal wavelet-neural network classifier. *arXiv preprint arXiv:1307.7897*, 2013.

Outrata, J. Preprocessing input data for machine learning by FCA. In *Proceedings of the 7th International Conference on Concept Lattices and Their Applications*, pages 187–198, 2010.

Pacola, E. R., **Quandt, V. I.**, **Liberalesso, P. B. N.**, **et al.** Influences of the signal border extension in the discrete wavelet transform in EEG spike detection. *Research on Biomedical Engineering*, 32(3):253–262, 2016.

Parkinson, P. S., **Xu, H.**, **Yu, P.**, **et al.** Classification and ranking of Fermi-LAT gamma-ray sources from the 3FGL catalog using machine learning techniques. *The Astrophysical Journal*, 820(1):8, 2016.

Partal, T., **Cigizoglu, H. K.**, and **Kahya, E.** Daily precipitation predictions using three different wavelet neural network algorithms by meteorological data. *Stochastic Environmental Research and Risk Assessment*, 29(5):1317–1329, 2015.

Pascanu, R., **Mikolov, T.**, and **Bengio, Y.** On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pages 1310–1318, 2013.

Pedregosa, F., **Varoquaux, G.**, **Gramfort, A.**, **et al.** Scikit-learn: machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Perez, L. and **Wang, J.** The effectiveness of data augmentation in image classification using deep learning. *arXiv preprint arXiv:1712.04621*, 2017.

Peterson, G. E. and **Barney, H. L.** Control methods used in a study of the vowels. *The Journal of the Acoustical Society of Americac*, 24(2):175–184, 1952.

Piramuthu, S. and **Sikora, R. T.** Iterative feature construction for improving inductive learning algorithms. *Expert Systems with Applications*, 36(2):3401–3406, 2009.

Platt, J. Fast training of support vector machines using sequential minimal optimization. In *Advances in Kernel Methods - Support Vector Learning*. MIT Press, 1998.

Povey, D., **Zhang, X.**, and **Khudanpur, S.** Parallel training of DNNs with natural gradient and parameter averaging. *arXiv preprint arXiv:1410.7455*, 2014.

Qiu, J., **Wu, Q.**, **Ding, G.**, **Xu, Y.**, and **Feng, S.** A survey of machine learning for big data processing. *EURASIP Journal on Advances in Signal Processing*, 2016(1):67, 2016.

Qiu, Q., **Cheng, X.**, **Calderbank, R.**, and **Sapiro, G.** DCFNet: deep neural network with decomposed convolutional filters. *arXiv preprint arXiv:1802.04145*, 2018.

Qureshi, M. A. and **Deriche, M.** A new wavelet based efficient image compression algorithm using compressive sensing. *Multimedia Tools and Applications*, 75(12):6737–6754, 2016.

Racusin, J., **Perkins, J. S.**, **Briggs, M. S.**, **et al.** BurstCube: a CubeSat for gravitational wave counterparts. *arXiv preprint arXiv:1708.09292*, 2017.

Raiko, T., **Valpola, H.**, and **LeCun, Y.** Deep learning made easier by linear transformations in perceptrons. In *Artificial Intelligence and Statistics*, pages 924–932, 2012.

Rajmic, P. and **Prusa, Z.** Discrete wavelet transform of finite signals: detailed study of the algorithm. *International Journal of Wavelets, Multiresolution and Information Processing*, 12(1), 2014.

Rando, R., **Canevarolo, S.**, **Xiao, H.**, and **Bastieri, D.** Sensitivity to gamma-ray bursts of a nanosatellite MeV telescope with a silicon tracker. *The Astronomical Journal*, 158(1):42, 2019.

Raschka, S. *Python machine learning*. Packt Publishing Ltd, 2015.

Rippel, O., **Snoek, J.**, and **Adams, R. P.** Spectral representations for convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 2449–2457, 2015.

Rosenberg, A. E. Effect of glottal pulse shape on the quality of natural vowels. *The Journal of the Acoustical Society of America*, 49(2B):583–590, 1971.

Rosenblatt, F. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

Rumelhart, D. E., **Hinton, G. E.**, and **Williams, R. J.** Learning representations by back-propagating errors. *Nature*, 323:533 EP –, 1986.

Sainath, T. N., **Kingsbury, B.**, **Sindhwani, V.**, **Arisoy, E.**, and **Ramabhadran, B.** Low-rank matrix factorization for deep neural network training with high-dimensional output targets. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6655–6659. IEEE, 2013.

Salimans, T., **Goodfellow, I.**, **Zaremba, W.**, **et al.** Improved techniques for training GANs. In *Advances in Neural Information Processing Systems*, pages 2234–2242, 2016.

Salvetti, D., **Chiaro, G.**, **La Mura, G.**, and **Thompson, D. J.** 3FGLzoo: classifying 3FGL unassociated Fermi-LAT gamma-ray sources by artificial neural networks. *Monthly Notices of the Royal Astronomical Society*, 470(2):1291–1297, 2017.

Sandler, M., **Howard, A.**, **Zhu, M.**, **Zhmoginov, A.**, and **Chen, L.-C.** Mobilenetv2: inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018.

Saxe, A. M., **McClelland, J. L.**, and **Ganguli, S.** Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*, 2013.

Schoenfelder, V., **Aarts, H.**, **Bennett, K.**, **et al.** Instrument description and performance of the imaging gamma-ray telescope COMPTEL aboard the Compton gamma-ray observatory. *The Astrophysical Journal Supplement Series*, 86:657–692, 1993.

Schölkopf, B., **Smola, A.**, and **Müller, K.-R.** Kernel principal component analysis. In *International Conference on Artificial Neural Networks*, pages 583–588. Springer, 1997.

Scholkopf, B. and **Smola, A. J.** *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2001.

Schonfelder, V., **Diehl, R.**, **Lichti, G. G.**, **et al.** The imaging Compton telescope COMPTEL on the gamma ray observatory. *IEEE Transactions on Nuclear Science*, 31(1):766–770, 1984.

Schroeder, M. *Fractals, chaos, power laws*. Mineola, New York: Dover Publications, 2009.

Sellami, A. and **Farah, M.** Comparative study of dimensionality reduction methods for remote sensing images interpretation. In *2018 4th International Conference on Advanced Technologies for Signal and Image Processing*, pages 1–6, 2018.

Sermanet, P., **Chintala, S.**, and **LeCun, Y.** Convolutional neural networks applied to house numbers digit classification. *arXiv preprint arXiv:1204.3968*, 2012.

Shimodaira, H. Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of Statistical Planning and Inference*, 90(2):227–244, 2000.

Sidney Burrus, C., **Gopinath, R. A.**, and **Guo, H.** *Introduction to wavelets and wavelet transforms: a primer*. Prentice Hall, 1 edition, 1998.

Sønderby, C. K., **Caballero, J.**, **Theis, L.**, **Shi, W.**, and **Huszár, F.** Amortised map inference for image super-resolution. In *International Conference on Learning Representations*, 2017.

Srivastava, N., **Hinton, G. E.**, **Krizhevsky, A.**, **Sutskever, I.**, and **Salakhutdinov, R.** Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Strang, G. and **Nguyen, T.** *Wavelets and filter banks*. Wellesley College, 2 edition, 1996.

Sun, X., **Ren, X.**, **Ma, S.**, and **Wang, H.** meProp: sparsified back propagation for accelerated deep learning with reduced overfitting. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70, pages 3299–3308, 2017.

Sunny, S., **David Peter, S.**, and **Jacob, K. P.** Discrete wavelet transforms and artificial neural networks for recognition of isolated spoken words. *International Journal of Computer Applications*, 38(9):9–13, 2012.

Sweldens, W. The lifting scheme: a custom-design construction of biorthogonal wavelets. *Applied and computational harmonic analysis*, 3(2):186–200, 1996.

Tanmoy, R., **Snehashish, C.**, **Tshilidzi, M.**, and **Paul, S.** Introducing new feature set based on wavelets for speech emotion classification. In *2018 IEEE Applied Signal Processing Conference*, pages 124–128. IEEE, 2018.

Tavani, M., **Barbiellini, G.**, **Argan, A.**, **et al.** The AGILE mission. *Astronomy & Astrophysics*, 502(3):995–1013, 2009.

Traina Jr, C., **Traina, A.**, **Wu, L.**, and **Faloutsos, C.** Fast feature selection using fractal dimension. *Journal of Information and Data Management*, 1(1):17–20, 2010.

Traina Jr, C., **Traina, A. J.**, and **Faloutsos, C.** Distance exponent: a new concept for selectivity estimation in metric trees. In *Proceedings of 16th International Conference on Data Engineering*, pages 195–195, 2000.

Tutz, G. and **Ramzan, S.** Improved methods for the imputation of missing data by nearest neighbor methods. *Computational Statistics & Data Analysis*, 90:84–99, 2015.

Van Loan, C. *Computational frameworks for the fast Fourier transform*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 1992.

Verain, M. C., **Sijtsema, S. J.**, and **Antonides, G.** Consumer segmentation based on food-category attribute importance: the relation with healthiness and sustainability perceptions. *Food Quality and Preference*, 48:99–106, 2016.

Vyas, A. and **Paik, J.** Applications of multiscale transforms to image denoising: survey. In *2018 International Conference on Electronics, Information, and Communication*, pages 1–3, 2018.

Walnut, D. F. *An introduction to wavelet analysis*. Springer Science & Business Media, 2013.

Wang, N. and **Adeli, H.** Self-constructing wavelet neural network algorithm for nonlinear control of large structures. *Engineering Applications of Artificial Intelligence*, 41:249–258, 2015.

Wang, P. and **Cheng, J.** Accelerating convolutional neural networks for mobile applications. In *Proceedings of the 24th ACM International Conference on Multimedia*, pages 541–545. ACM, 2016.

Wang, P., **Hu, Q.**, **Zhang, Y.**, **et al.** Two-step quantization for low-bit neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4376–4384, 2018a.

Wang, S.-S., **Chern, A.**, **Tsao, Y.**, **et al.** Wavelet speech enhancement based on nonnegative matrix factorization. *IEEE Signal Processing Letters*, 23(8):1101–1105, 2016.

Wang, W., **Sun, Y.**, **Eriksson, B.**, **Wang, W.**, and **Aggarwal, V.** Wide compression: tensor ring nets. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9329–9338, 2018b.

Wen, W., **Xu, C.**, **Yan, F.**, **et al.** Terngrad: ternary gradients to reduce communication in distributed deep learning. In *Advances in Neural Information Processing Systems*, pages 1509–1519, 2017.

Weston, J., **Watkins, C.**, **et al.** Support vector machines for multi-class pattern recognition. In *Proceedings of the 7th European Symposium On Artificial Neural Networks*, volume 99, pages 219–224, 1999.

Widrow, B. and **Lehr, M. A.** 30 years of adaptive neural networks: perceptron, madaline, and backpropagation. *Proceedings of the IEEE*, 78(9):1415–1442, 1990.

Wiesler, S., **Richard, A.**, **Schlüter, R.**, and **Ney, H.** Mean-normalized stochastic gradient for large-scale deep learning. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 180–184. IEEE, 2014.

Williams, T. and **Li, R.** Wavelet pooling for convolutional neural networks. In *International Conference on Learning Representations*, volume 6, 2018.

Winkler, C., **Courvoisier, T. J.-L.**, **Di Cocco, G.**, **et al.** The INTEGRAL mission. *Astronomy & Astrophysics*, 411(1):L1–L6, 2003.

Wong, S. C., **Gatt, A.**, **Stamatescu, V.**, and **McDonnell, M. D.** Understanding data augmentation for classification: when to warp? In *2016 International Conference on Digital Image Computing: Techniques and Applications*, pages 1–6. IEEE, 2016.

Xu, B., **Wang, N.**, **Chen, T.**, and **Li, M.** Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.

Yan, S., **Xiong, Y.**, and **Lin, D.** Spatial temporal graph convolutional networks for skeleton-based action recognition. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Yang, C., **Buluç, A.**, and **Owens, J. D.** Design principles for sparse matrix multiplication on the GPU. In *European Conference on Parallel Processing*, pages 672–687. Springer, 2018.

Yao, C., **Gao, X.**, and **Yu, Y.** Wind speed forecasting by wavelet neural networks: a comparative study. *Mathematical Problems in Engineering*, 2013, 2013.

Ye, J., **Wang, L.**, **Li, G.**, **et al.** Learning compact recurrent neural networks with block-term tensor decomposition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9378–9387, 2018.

Yoon, J. and **Hwang, S. J.** Combined group and exclusive sparsity for deep neural networks. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70, pages 3958–3966, 2017.

Yu, D. and **Deng, L.** Deep learning and its applications to signal and information processing. *IEEE Signal Processing Magazine*, 28(1):145–154, 2010.

Yu, D., **Wang, H.**, **Chen, P.**, and **Wei, Z.** Mixed pooling for convolutional neural networks. In *International Conference on Rough Sets and Knowledge Technology*, pages 364–375. Springer, 2014.

Zeiler, M. D. and **Fergus, R.** Stochastic pooling for regularization of deep convolutional neural networks. *arXiv preprint arXiv:1301.3557*, 2013.

Zhang, Q. Using wavelet network in nonparametric estimation. *IEEE Transactions on Neural Networks*, 8(2):227–236, 1997.

Zhang, Q. and **Benveniste, A.** Wavelet networks. *IEEE Transactions on Neural Networks*, 3(6):889–898, 1992.

Zhang, X., **Zhou, X.**, **Lin, M.**, and **Sun, J.** Shufflenet: an extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6848–6856, 2018.

Zhang, X., **Zou, J.**, **He, K.**, and **Sun, J.** Accelerating very deep convolutional networks for classification and detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(10):1943–1955, 2015.

Zhang, Y., **Zhang, H.**, **Cai, J.**, and **Yang, B.** A weighted voting classifier based on differential evolution. In *Abstract and Applied Analysis*, volume 2014, 2014.

Zhou, L., **Pan, S.**, **Wang, J.**, and **Vasilakos, A. V.** Machine learning on big data: opportunities and challenges. *Neurocomputing*, 237:350–361, 2017.

Zhuang, L. and **Bioucas-Dias, J. M.** Fast hyperspectral image denoising and inpainting based on low-rank and sparse representations. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 11(3):730–742, 2018.

Zoglauer, A., **Andritschke, R.**, **Boggs, S. E.**, **et al.** MEGAlib–simulation and data analysis for low-to-medium-energy gamma-ray telescopes. In *Proceedings of the International Society for Optical Engineering*, volume 7011, page 70113F, 2008.

Zyner, A., **Worrall, S.**, and **Nebot, E.** Naturalistic driver intention and path prediction using recurrent neural networks. *IEEE Transactions on Intelligent Transportation Systems*, 2019.