Deep Learning for Sequence Modelling: Applications in Natural Languages and Distributed Compressive Sensing

by

Hamid Palangi

M.Sc., Electrical Engineering, Sharif University of Technology, 2010

B.Sc., Electrical Engineering, Shahid Rajaee University, 2007

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

Doctor of Philosophy

in

THE FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES

(Electrical and Computer Engineering)

The University of British Columbia

(Vancouver)

April 2017

© Hamid Palangi, 2017

Abstract

The underlying data in many machine learning tasks have a sequential nature. For example, words generated by a language model depend on the previously generated words, behaviour of a user in a social network evolves over different snapshots of the social graph over time, different speech frames in a speech recognition system depend on the previously generated frames, etc. The main question is, how can we leverage the sequential nature of data to extract better features for the target machine learning task? In an effort to address this question, this thesis presents three important applications of deep sequence modelling methods.

The first application is *sentence modelling for web search task* where the question addressed is: How to create a vector representation for a natural language sentence, aimed at a specific task such as web search? We propose Long Short-Term Memory Deep Structured Semantic Model (LSTM-DSSM), a model for information retrieval on click-through data with significant performance gains compared to existing state of the art baselines. The proposed LSTM-DSSM model sequentially takes each word in a sentence, extracts its relevant information, and embeds it into a semantic vector.

The second application involves *distributed compressive sensing*, where the main questions addressed are: (a) How to relax the joint sparsity constraint? (b) How to exploit the structural dependency of a group of sparse vectors to reconstruct them better from down-sampled measurements (structures besides sparsity)? (c) How to exploit available offline data during sparse reconstruction at the decoder? We present a deep learning approach to distributed compressive sensing and show that it addresses the above three questions and is almost as fast as greedy methods during reconstruction.

The third application is related to *speech recognition*. The question addressed here is: How to build a recurrent acoustic model for the task of phoneme recognition? We present a Recurrent Deep Stacking Network (R-DSN) architecture for this task. Each module in the R-DSN is initialized with an Echo State Network (ESN), and then all connection weights within the module are fine tuned.

Preface

This thesis presents the research conducted by Hamid Palangi, in collaboration with Prof. Rabab Ward and Prof. Li Deng. Below is the list of the scientific papers by Hamid Palangi that were published through the course of his studies as a Doctor of Philosophy (PhD) candidate at University of British Columbia (UBC).

- **J1**: Hamid Palangi, Rabab Ward, Li Deng, "Distributed Compressive Sensing: A Deep Learning Approach", *IEEE Transactions on Signal Processing*, Volume: 64, Issue: 17, pp. 4504-4518, 2016

- **J2**: Hamid Palangi, Li Deng, Yelong Shen, Jianfeng Gao, Xiaodong He, Jianshu Chen, Xinying Song, Rabab Ward, "Deep Sentence Embedding Using the Long Short-Term Memory Networks: Analysis and Application to Information Retrieval", *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, Volume: 24, Issue: 4, pp. 694-707, 2016

- **J3**: Hamid Palangi, Rabab Ward, Li Deng, "Convolutional Deep Stacking Networks for Distributed Compressive Sensing", *Elsevier Signal Processing*, Volume: 131, pp. 181-189, 2016

- **C1**: Hamid Palangi, Rabab Ward, Li Deng, "Exploiting Correlations Among Channels in Distributed Compressive Sensing with Convolutional Deep Stacking Networks", *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Shanghai, China, 2016

- **C2**: Hamid Palangi, Rabab Ward, Li Deng, "Reconstruction of Sparse Vectors in Compressive Sensing with Multiple Measurement Vectors using Bidirectional Long Short-Term Memory", *IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, Washington D.C., USA, 2016

- **C3**: Hamid Palangi, Li Deng, Rabab Ward, "Recurrent Deep-Stacking Networks for sequence classification", *IEEE China Summit and International Conference on Signal and Information Processing (ChinaSIP)*, Xi'an, China, 2014.

- **C4**: Hamid Palangi, Li Deng, Rabab Ward, "Learning Input and Recurrent Weight Matrices in Echo State Networks", *Deep Learning Workshop in Conference on Neural Information Processing Systems (NIPS)*, Lake Tahoe, Nevada, USA, 2013

- **C5**: Hamid Palangi, Rabab Ward, Li Deng, "Using Deep Stacking Network to Improve Structured Compressed Sensing with Multiple Measurement Vectors", *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Vancouver, Canada, 2013

- **J1, C2**: Hamid Palangi (the primary author and the main contributor of the papers) proposed the methods, implemented them, and performed the evaluations. Prof. Ward and Prof. Deng provided extensive technical and editorial feedback throughout writing of these papers.

- **J3, C1, C5**: Hamid Palangi (the primary author and the main contributor of the papers) proposed the methods, implemented them, and performed the evaluations. Prof. Deng provided the codes of his Deep Stacking Network (DSN) paper [29]. Prof. Ward and Prof. Deng provided extensive technical and editorial feedback throughout writing of these papers.

- **J2**: Hamid Palangi (the primary author and the main contributor of the paper) derived all the formulations required for learning, implemented the methods, and performed the evaluations. Prof. Deng, Prof. Gao and Prof. He provided extensive technical feedback during the project and helped with writing of the paper. Dr. Shen and Dr. Song provided extensive technical feedbacks during implementation of methods. Dr. Shen and Prof. He helped to run some of the experiments. Dr. Chen helped with writing of the paper and provided valuable feedback and ideas for visualizations. Prof. Ward provided extensive technical and editorial feedback throughout writing of the paper.

- **C3, C4**: Hamid Palangi (the primary author and the main contributor of the papers) derived all the formulations required for learning, implemented the methods, and performed the evaluations. Prof. Deng provided the codes of his DSN paper [29], extensive technical feedback during design and implementation of methods and helped to run the experiments. Prof. Deng and Prof. Ward provided extensive technical and editorial feedback throughout writing of these papers.

Table of Contents

List of Tables

List of Figures

Glossary

DNN Deep Neural Network

LSTM-DSSM Long Short-Term Memory Deep Structured Semantic Model

LSTM Long Short-Term Memory

MMV Multiple Measurement Vectors

SOMP Simultaneous Orthogonal Matching Pursuit

R-DSN Recurrent Deep Stacking Network

DSN Deep Stacking Network

ESN Echo State Network

DL Deep Learning

GPU Graphical Processing Unit

CS Compressive Sensing

MLP Multilayer Perceptron

RELU Rectified Linear Unit

CNN Convolutional Neural Network

SVM Support Vector Machine

RNN Recurrent Neural Network

BPTT Backpropagation Through Time

EEG Electroencephalogram

WBAN Wireless Body-Area Network

CT Computed Tomography

MRI Magnetic Resonance Imaging

SMV Single Measurement Vector

CDSN Convolutional Deep Stacking Network

MFCC Mel Frequency Cepstral Coefficients

GRU Gated Recurrent Unit

NLP Natural Language Processing

DCNN Dynamic Convolutional Neural Network

NDCG Normalized Discounted Cumulative Gain

BCS Bayesian Compressive Sensing

MT-BCS Multitask Compressive Sensing

SBL Sparse Bayesian Learning

LASSO least absolute shrinkage and selection operator

Acknowledgments

I would not have been able to make it to this point without the support of my supervisors Prof. Rabab Ward and Prof. Li Deng.

I would like to thank Prof. Rabab Ward for her patience, generous help, and wise suggestions that helped me a lot during my PhD studies. Other than her scientific guidance throughout my research work, she taught me a lot about technical writing and technical presentation skills. I would also like to thank Prof. Rabab Ward for her financial support through her Natural Sciences and Engineering Research Council of Canada (NSERC) grants.

I would like to thank Prof. Li Deng for giving me a chance to explore new ideas, teaching me how to approach a new challenging problem, and his great help and suggestions throughout my PhD studies. I would also like to thank Prof. Li Deng and Microsoft Research (MSR) to give me 3 amazing research internship opportunities. At MSR, I had the honour to work under supervision of Prof. Li Deng with very smart researchers Prof. Jianfeng Gao, Prof. Xiaodong He, Dr. Yelong Shen, Dr. Jianshu Chen and Dr. Xinying Song. I want to thank all of them for many important lessons that I learned from them.

I would like to thank anonymous reviewers of my research papers for their comments which helped me improve the quality of my works.

Each friendship offers something totally unique and irreplaceable. Each friendship ultimately makes us who we are. I am happy and lucky to have great friends at our lab. Thank you Davood, Hossein, Sima, Fahimeh, Basak, Ali, Maryam, Anahita, and Ilya.

All that I am, or hope to be, I owe to my parents and family. Thank you for your unconditional love. Thank you for working hard and for always being there for me.

Thank you dad . . . it hurts every single day to think that you are not here anymore. I start this thesis with your memory . . .

Chapter 1

Introduction

Everything we hear is an opinion, not a fact. Everything we see is a perspective, not the truth.
— Marcus Aurelius

Deep Learning (DL) [23, 27, 48, 55, 79, 129, 131] is the foundation for many recent breakthroughs in different research areas [20, 37, 51, 58, 76, 91, 109, 125, 128]. Some fundamental ideas in deep learning, e.g., feedforward or convolutional neural networks, back-propagation and recurrent neural networks have been known from 1980s. The main reasons for the recent popularity of these ideas are: (a) the fast and efficient computation using Graphical Processing Unit (GPU) which makes it possible to run experiments in a reasonable amount of time, and (b) the availability of very large datasets (90% of world's data have been generated over the last two years[2]).

Among the different deep models, sequence modelling methods are of great importance. The main reason for this is that many real world and machine learning tasks have an inherent sequential nature. For example, (a) sentences used in a conversation depend on the previous sentences used in that conversation and the context, i.e., a sequence of sentences, (b) words / characters used in a sentence depend on previous words / characters, i.e., a sequence of words / characters. This is very important in language modelling tasks. (c) Different blocks in one image and different image frames in a video have strong sequential dependencies / correlations, i.e., a sequence of images. (d) In an acoustic model of speech recognition system, the probability of which speech frame happens next, depends on the previously observed speech frames, i.e., a sequence of speech frames. (e): In a social network, e.g., Facebook or Linkedin, a social graph represents each user by a node and the connections of that user to other users as edges in the graph. This graph quickly evolves over time depending on new interactions among users, e.g., new connections, followers, etc. Different snapshots of this social graph over time have strong correlations / dependencies, i.e., a sequence of social graphs. This is central in the important task of link prediction in social networks, e.g., "*people you may know*" feature in Facebook or Linkedin. (f) In a machine translation system, e.g., Google translate or Skype translator, the main task is: given a sentence in a source language (e.g., French), generate its translation in a target language (e.g., English). The state-of-the-art methods for this task are based on sequence to sequence models that use one sequence model for source language (encoder), and another sequence model for target language (decoder). The encoder sequence model basically extracts

features from a sequence of words / characters in a source language. The decoder sequence model generates words / characters in target language given the features from encoder. In other words, the whole system is based on the sequential nature of words / characters in source and target language.

In this thesis, we addressed three important problems using approaches based on deep sequence modelling methods:

1. *Acoustic modelling for speech recognition* which resulted in a new model, Recurrent Deep Stacking Network (R-DSN) for speech recognition.

2. *Sentence modelling for web search task* which resulted in Long Short-Term Memory Deep Structured Semantic Model (LSTM-DSSM). This model can potentially improve the performance of search engines like Google or Bing and

3. *Compressive Sensing* which resulted in a fundamentally new deep learning approach to distributed compressive sensing.

The rest of this chapter is organized as follows: in the next section an introduction to deep learning and deep sequence modelling methods are presented. In section 1.2, an introduction to Compressive Sensing (CS) is described. Section 1.3 explains motivations behind this research work and identifies our contributions. The organization of the thesis is summarized in section 1.4.

1.1 Deep Learning

Since their introduction in 1957 [104], neural networks have been used for finding patterns in input data and for modelling complicated relationship between input and output data. The first generation of neural networks is the Perceptron [104] that performs a classification task by mapping an input vector of real numbers to a single binary value (its output). The Perceptron has a simple learning algorithm but it can not learn complex structures. The second generation of neural networks is the Multilayer Perceptron (MLP) which has multiple layers of neurons (processing units). In MLP, each neuron in a hidden layer has a non-linear activation function, e.g., sigmoid function $f(x) = \frac{1}{1+e^{-x}}$. The learning algorithm used in MLP is Backpropagation [105]. Although MLP performs better than the Perceptron, Backpropagation has two main problems; it is very slow in networks with multiple hidden layers and it can get stuck in poor local optima. Due to these problems in the 1990's many researchers stopped using neural networks with multiple hidden layers [56]. However, in 2006, two seminal papers [60] and [61] proposed more efficient methods for training Deep Neural Network (DNN), i.e., neural networks with many hidden layers. Afterwards, DNNs were used successfully in different applications, e.g. speech, audio, image, video and language processing [129].

A DNN is a neural network with more than one hidden layer in space (e.g., the multilayer feed-forward or the convolutional neural networks) or in time (e.g., recurrent neural network). For example, for feed-forward neural networks, each unit j in a hidden layer is typically a non-linear function (e.g., sigmoid, tanh

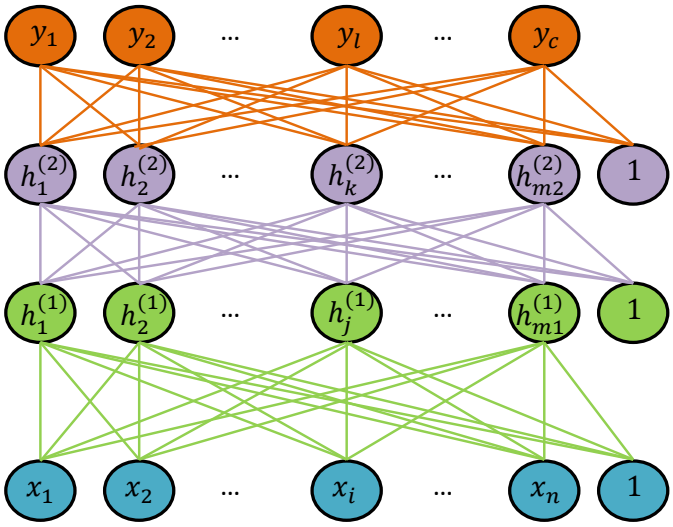


Figure 1.1: A Feed-forward DNN.

or Rectified Linear Unit (RELU)) that maps its input to a state $h_j^{(1)}$ where

$$h_j^{(1)} = f(\sum_i x_i w_{ij} + b_j). \tag{1.1}$$

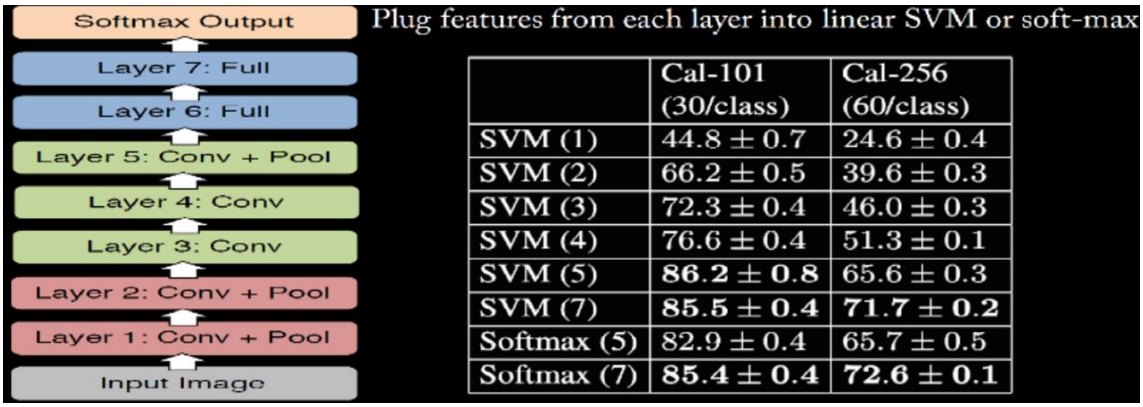
In (1.1), $h_j^{(1)}$ is the output of the j-th unit in the first hidden layer (the superscript (1) in $h_j^{(1)}$ indicates the hidden layer number which is 1 in this case), the w_{ij} is the weight for the connection between this unit and the unit x_i (where x_i is the unit i in the layer below), and b_j is the bias for j-th unit. $f(.)$ is the non-linear activation function for unit j. For example, the activation function in the case of sigmoid non-linearity is

$$f(x) = \frac{1}{1+e^{-x}}. \tag{1.2}$$

A feed-forward DNN model is presented in Fig. 1.1. In this figure, a DNN with n input (visible) units represented by $[x_1, x_2, \ldots, x_n]$, the $m1$ hidden units in hidden layer 1 that are feature detectors, the $m2$ hidden units in hidden layer 2 that are also feature detectors and the c output units $[y_1, y_2, \ldots, y_c]$ is represented. As an example, if we want to do multi-class classification using this model, and assuming that we have c classes, we add a softmax layer on the top of the network (not shown in Fig. 1.1) to get a value between 0 and 1 for each class, i.e.,

$$p_j = \frac{e^{y_j}}{\sum_l e^{y_l}}, \tag{1.3}$$

where p_j in (1.3) can be interpreted as the probability that input data $\mathbf{x} = [x_1, x_2, \ldots, x_n]^T$ belongs to the j-th class. You can think of the input data $\mathbf{x}$ as pixels of an image, or as the Mel Frequency Cepstral Coefficients (MFCC) features of a speech frame, or as one hot representation of a word, or a bag of words



	Cal-101 (30/class)	Cal-256 (60/class)
SVM (1)	44.8 ± 0.7	24.6 ± 0.4
SVM (2)	66.2 ± 0.5	39.6 ± 0.3
SVM (3)	72.3 ± 0.4	46.0 ± 0.3
SVM (4)	76.6 ± 0.4	51.3 ± 0.1
SVM (5)	$\mathbf{86.2 \pm 0.8}$	65.6 ± 0.3
SVM (7)	$\mathbf{85.5 \pm 0.4}$	$\mathbf{71.7 \pm 0.2}$
Softmax (5)	82.9 ± 0.4	65.7 ± 0.5
Softmax (7)	$\mathbf{85.4 \pm 0.4}$	$\mathbf{72.6 \pm 0.1}$

Figure 1.2: Importance of depth (Figure from [40]).

representation of a sentence, or simply some hand crafted features from the data. We can also use a weighted layer to define p_j in (1.3) and learn those weights as well. In other words, if we define $\mathbf{y} = [y_1, y_2, \dots, y_c]^T$ then

$$p_j = \frac{e^{\mathbf{y}^T \mathbf{U}_j}}{\sum_l e^{\mathbf{y}^T \mathbf{U}_l}}, \tag{1.4}$$

where $\mathbf{U}_j$ is j-th row of the weights matrix between layer $[y_1, y_2, \dots]$ and layer $[p_1, p_2, \dots]$ (not shown in Fig. 1.1) which will be learned during training. To learn all parameters in Fig. 1.1 and also $\mathbf{U}$ in (1.4), we need to define an appropriate cost function. For example, the cost function in the above classification example is usually the cross entropy between the target labels, t, and the observed probabilities, p,

$$loss = -\sum_j t_j log(p_j). \tag{1.5}$$

In (1.5), target label t_j is our given knowledge from training data of the class input data $\mathbf{x}$ belongs to. If $\mathbf{x}$ actually belongs to class j, then t_j is 1, otherwise it is 0. p_j, on the other hand, is the DNN's opinion about input data $\mathbf{x}$. It is the probability that DNN thinks input data $\mathbf{x}$ belongs to class j.

The main role of hidden layers in Fig. 1.1 is to create high level *good features* from the input data $\mathbf{x}$. A natural question here is: how important is depth of the network? In other words, does adding more hidden layers continuously improves the performance? We can investigate the importance of depth by inspecting the different parts of the Krizhevsky's Convolutional Neural Network (CNN) [76] which has 8 layers and is trained on ImageNet [106] dataset. The architecture of Krizhevsky's CNN along with the results of applying the Support Vector Machine (SVM) classifier on different layers are shown in Fig. 1.2. The main observation from Fig. 1.2 is that applying the classifier to features from higher layers results in significant performance improvement (e.g., from 44.8% using features from layer 1 to 85.5% using features from layer 7). It is important to note that simply adding many more layers does not always improve the performance. A good example for this is the results of simply using 20 layers and 56 layers of CNN for CIFAR-10 dataset which

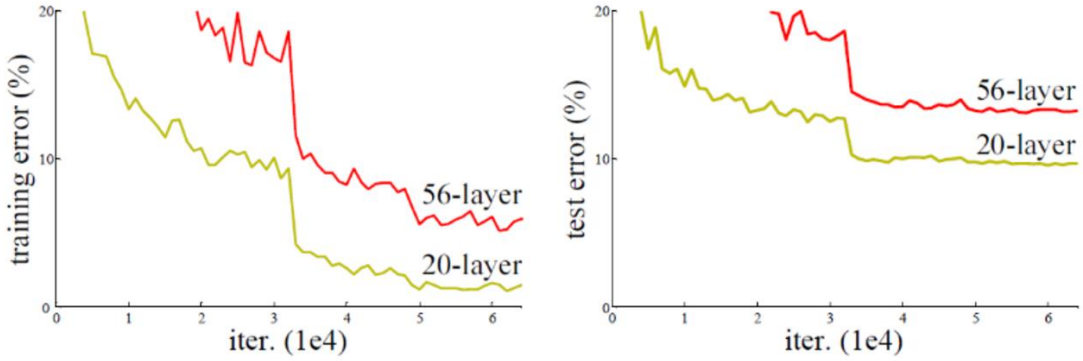


Figure 1.3: Effect of adding many layers (Figure from [52]).

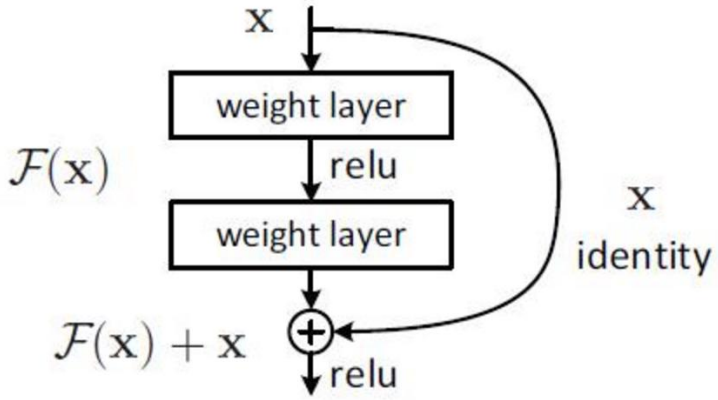


Figure 1.4: Adding skip connections to resolve the problem with many layers (Figure from [52]).

is presented in Fig. 1.3. Similar phenomena have been observed on the ImageNet dataset which means that learning better models is not always equivalent to adding more layers. Note that the above problem is NOT caused by overfitting as it is obvious from the training error curves in Fig. 1.3. One reason is the fact that with deeper networks the error signal during backpropagation is no longer significant enough by the time it arrives to the lower layers. To resolve this problem, a residual network is proposed in [52] which simply adds skip connections in the CNN architecture. One example is shown in Fig. 1.4. Note that the skip connection is applied before the non-linear activation function.

The Recurrent Neural Network (RNN) is a type of deep neural network that is *deep* in the temporal dimension and has been used extensively in time sequence modelling [13, 28, 36, 49, 84, 85, 103]. To explain a simple RNN, assume that we have been observing input data from time $t = 1$ till time $t = T$. For example, we can think of the input data as a sentence with T words. Assume the t-th word in the sentence is represented by vector $\mathbf{x}(t) = [x_1(t), x_2(t), \ldots, x_n(t)]^T$ (T stands for transpose here). $\mathbf{x}(t)$ can be a feature vector like word2vec [88] or a one hot representation of t-th word. We continue our explanation of RNN using Fig. 1.5 which represents a one layer recurrent neural network unfolded over time. In Fig. 1.5, $\mathbf{h}(t) = [h_1(t), h_2(t), \ldots, h_{m1}(t)]^T$ is vector of hidden units' activations after observing t-th word

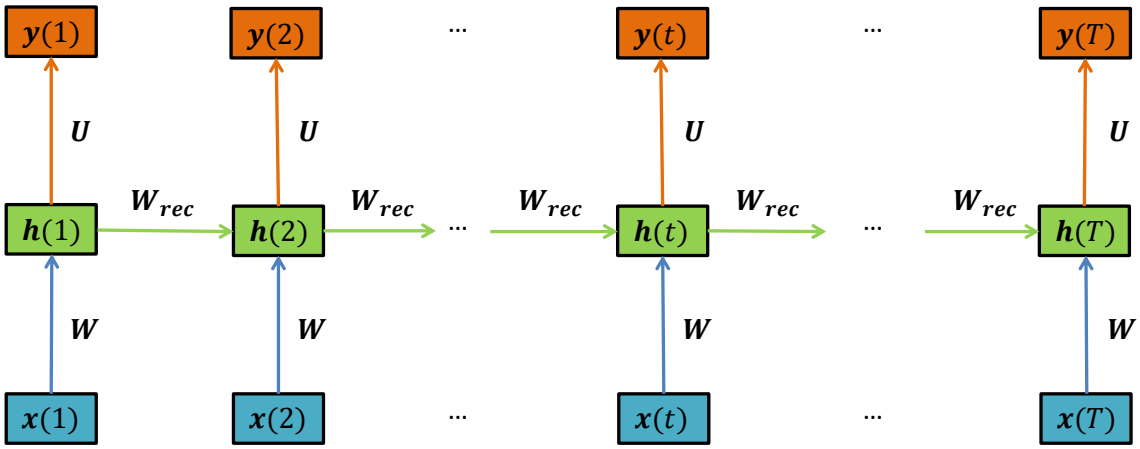


Figure 1.5: One layer RNN unfolded over time.

in the sentence $\mathbf{x}(t)$. Since there is just one hidden layer, we have not shown the layer number superscript for simplicity. Similarly, $\mathbf{y}(t) = [y_1(t), y_2(t), \ldots, y_c(t)]^T$ is the vector of the output units' activations after observing t-th word in the sentence. $\mathbf{W}$ is the matrix of weights between the input layer and the hidden layer, $\mathbf{W_{rec}}$ is the matrix of weights among the hidden layer activations over time and $\mathbf{U}$ is the matrix of weights between the hidden layer and the output layer. During the forward pass, $\mathbf{h}(t)$ and $\mathbf{y}(t)$ are calculated as

$$\mathbf{h}(t) = f(\mathbf{W}\mathbf{x}(t) + \mathbf{W_{rec}}\mathbf{h}(t-1)) \tag{1.6}$$

$$\mathbf{y}(t) = g(\mathbf{U}\mathbf{h}(t)), \tag{1.7}$$

where $f(.)$ and $g(.)$ are non-linear activation functions for the hidden layer and the output layer. Back-propagation Through Time (BPTT) is used to train this model. If we want to use this RNN model for the classification task explained before, we can similarly add a softmax layer on the top and minimize the cross entropy cost function in (1.5). There are of course much more details into the learning process and how to make it work (which we omit in this introductory section). We just want to emphasize that simple RNNs are notoriously hard to train [12] due to vanishing and exploding gradients problems. Nevertheless, there are some specific RNN architectures that have resolved these problems. We will discuss some of them later in this thesis.

A reasonable question to ask here is: Can all problems be mapped to a neural network style $\mathbf{y} = f(\mathbf{x})$? The short answer is No! Example tasks for which the simple $\mathbf{y} = f(\mathbf{x})$ fails are: (a) cloze style QA where the task is to read and comprehend a text (e.g., book, etc) and then answer questions about it. (b) Given a text, the task is to fill in the blanks and (c) ChatBot [24].

Generally a powerful model to address the above challenging tasks needs to remember the external context, given an input, the model needs to know where to look for in the context, what to look for in the context, how to reason, using this external context and the model should also handle a changing external context. To address some of the above challenges Memory Networks [112] have been proposed where the

main idea is to separate the controller of the memory from the memory itself. In other words, it combines a large memory with a learning component that can read and write to the memory.

1.2 Compressive Sensing

Compressive Sensing (CS) is an effective framework for acquiring sparse signals by which both sensing and compression are performed at the same time [9, 14, 31]. Since there are numerous examples of natural and artificial signals that are sparse in the time domain, spatial domain or a transform domain, CS has found various applications. One example to motivate the essence of compressive sensing is as follows: in a standard image compression like JPEG (or JPEG2000), the first step is to apply a DCT (or Wavelet) transform to the image to create a compact (few non-zeros) representation, and then the very small coefficients are discarded. Of course there are much more details but this is the basic idea. Consider though that we have spent computational resources to calculate the small coefficients that we later discard. Now the question is: is there any way to just sample (measure) the *information* that we need? The answer is Yes and this could be done by using compressive sensing.

As a sample application, compressive sensing can be used for tele-monitoring the scalp Electroencephalogram (EEG) signals via Wireless Body-Area Network (WBAN) [16]. In WBAN, a number of sensors collect and compress the EEG data. The compressed data are then sent to a remote terminal for further processing. By using this system, patients do not need to go to the hospital frequently. Instead, their EEG data can be monitored and transmitted to the hospital when needed. In designing a WBAN, there are a number of constraints such as the sensor's energy constraint due to its limited battery life [90], accuracy of the compressed and then transmitted physiological signals and a low hardware cost. Conventional data compression schemes can not overcome all these constraints [81]. Compressive sensing, on the other hand, can form a promising framework to solve the above constraints [6].

Besides wearable technology for health telemonitoring discussed above, other example applications of compressive sensing are building a single pixel camera [34], Computed Tomography (CT) to reduce the amount of radiation [5], Magnetic Resonance Imaging (MRI) to shorten MRI scanning sessions [80] and many more applications including geophysical data analysis, computational biology, remote sensing and communications [1].

A general CS framework with encoder and decoder is represented in Fig. 1.6. CS framework includes an encoder to down-sample the input data and a decoder to reconstruct the input data from its down-sampled version. In Fig. 1.6, the data is represented by vector $\mathbf{x} = [x(1), x(2), \ldots, x(N)]^T$. You can think of $\mathbf{x}$ as vectorized version of an image with N pixels. The CS encoder is generally a wide *random* matrix. This is represented by Φ which is an $M \times N$ matrix where $M < N$. The down-sampled version of $\mathbf{x}$ is represented by $\mathbf{y} = [y(1), y(2), \ldots, y(M)]^T$. The entries of $\mathbf{y}$ are called *measurements*, each of them is a linear random combination of all entries of $\mathbf{x}$. The $\mathbf{y}$ is referred to as the measurement vector. Since we have just one measurement vector in Fig. 1.6, it is also referred to as Single Measurement Vector (SMV) problem

$$\mathbf{y} = \Phi\mathbf{x}. \tag{1.8}$$

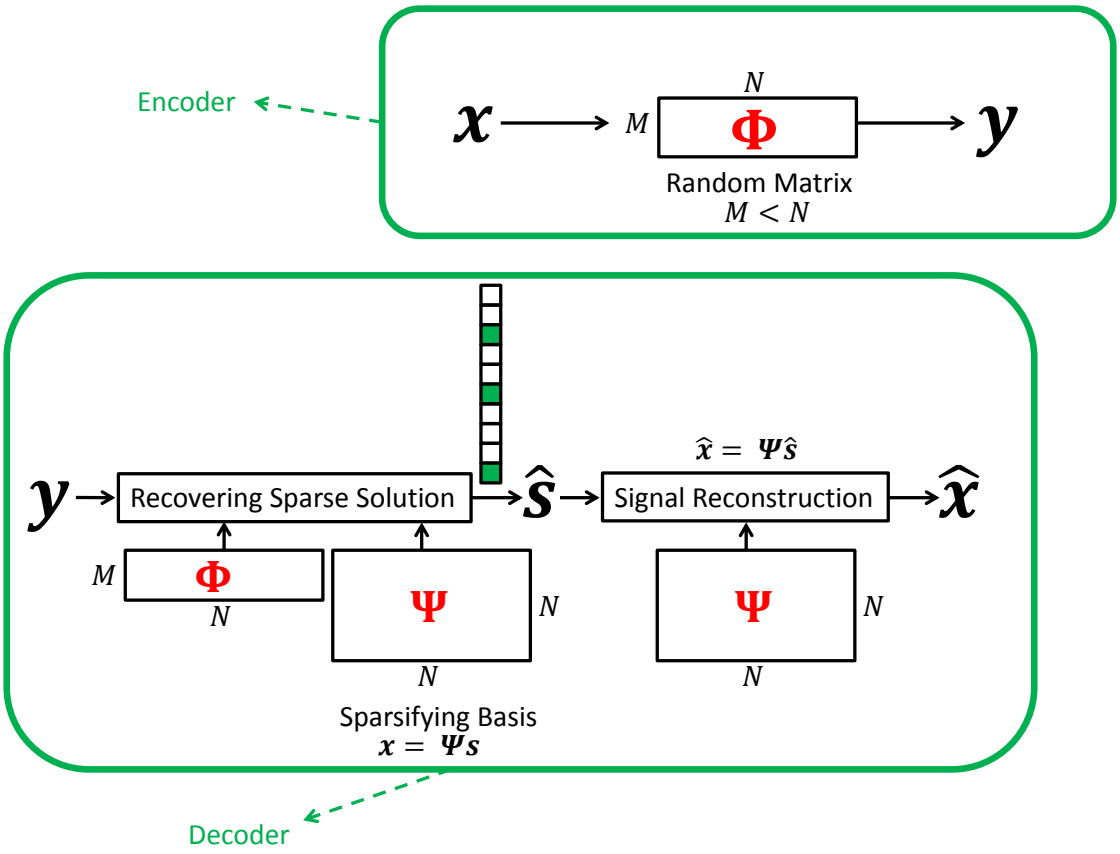


Figure 1.6: General CS framework.

The main task in CS is how to reconstruct the original data $\mathbf{x}$ given the measurement vector $\mathbf{y}$ and the measurement matrix Φ. One of the main reasons that this task is challenging is that the linear system of equations in (1.8) is underdetermined, i.e., it has more unknowns than equations. Therefore, there are infinite number of solutions for $\mathbf{x}$. Nevertheless, if $\mathbf{x}$ is sparse enough (has few non-zero entries), or if there is a transform domain Ψ where the transformed representation of $\mathbf{x}$ is sparse (e.g., using DCT of $\mathbf{x}$), then the theory of CS guarantees that under some reasonable conditions, the sparsest solution is unique [31]. The basis Ψ can be complete, i.e., $\Psi \in \Re^{N\times N}$, or over-complete, i.e., $\Psi \in \Re^{N\times N_1}$ where $N < N_1$ (compressed sensing for over-complete dictionaries is introduced in [15]). For this section, assume a complete basis $\Psi \in \Re^{N\times N}$ and

$$\mathbf{x} = \Psi\mathbf{s}, \tag{1.9}$$

where $\mathbf{s}$ is $K-sparse$, i.e., $\mathbf{s}$ has at most K non-zero elements. The condition to obtain a unique $\mathbf{s}$ given $\mathbf{y}$ is usually described based on the $spark$[1] value, i.e., $K < \frac{spark(\Phi\Psi)}{2}$ [32] or the the mutual coherence[2] ($\mu(\Phi\Psi)$)

[1] $spark$ of a given matrix (defined in section 3 of [32]), is the smallest possible number of linearly dependent columns of that matrix. It gives a measure of linear dependency in the system modelled by a given matrix.

[2] The coherence $\mu(\mathbf{A})$ of a given matrix $\mathbf{A}$ is defined as the maximum absolute value of inner products between any two columns

where the condition is $K < \frac{1}{2}(1 + \frac{1}{\mu(\Phi\Psi)})$ [33]. To find $\mathbf{s}$, the following optimization problem should be solved:

$$\hat{\mathbf{s}} = \underset{\mathbf{s}}{\operatorname{argmin}} \|\mathbf{s}\|_0 \quad s.t. \quad \mathbf{y} = \Phi\Psi\mathbf{s}, \tag{1.10}$$

where $\|\mathbf{s}\|_0$ is (pseudo) norm 0 of $\mathbf{s}$ and represents the number of non-zero entries in $\mathbf{s}$. Since ℓ_0 norm makes the problem non-convex, it is usually relaxed to the ℓ_1 norm and the following convex optimization problem is solved instead of (1.10):

$$\hat{\mathbf{s}} = \underset{\mathbf{s}}{\operatorname{argmin}} \|\mathbf{s}\|_1 \quad s.t. \quad \mathbf{y} = \Phi\Psi\mathbf{s}. \tag{1.11}$$

In distributed compressive sensing , also known as the Multiple Measurement Vectors (MMV) problem, a set of L sparse vectors $\{\mathbf{s}_i\}_{i=1,2,\ldots,L}$ is to be jointly recovered from a set of L measurement vectors $\{\mathbf{y}_i\}_{i=1,2,\ldots,L}$. Some application areas of MMV include magnetoencephalography, array processing, compressed sensing of different Electroencephalogram (EEG) channels on brain, equalization of sparse communication channels and cognitive radio [33].

Suppose that the L sparse vectors and the L measurement vectors are arranged as columns of matrices $\mathbf{S} = [\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_L]$ and $\mathbf{Y} = [\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_L]$ respectively. In the MMV problem, $\mathbf{S}$ is to be reconstructed given $\mathbf{Y}$,

$$\mathbf{Y} = \mathbf{A}\mathbf{S}, \tag{1.12}$$

where

$$\mathbf{A} = \Phi\Psi. \tag{1.13}$$

In (1.12), $\mathbf{S}$ is assumed to be jointly sparse, i.e., non-zero entries of each vector occur at the same locations as those of other vectors, which means that the sparse vectors have the same support. Assume that $\mathbf{S}$ is jointly sparse. Then, the necessary and sufficient condition to obtain a unique $\mathbf{S}$ given $\mathbf{Y}$ is [22]:

$$|supp(\mathbf{S})| < \frac{spark(\mathbf{A}) - 1 + rank(\mathbf{S})}{2}, \tag{1.14}$$

where $|supp(\mathbf{S})|$ is the number of rows in $\mathbf{S}$ with non-zero energy. In the SMV problem, no rank information exists. In the MMV problem, the rank information exists and affects the uniqueness bounds. Generally, solving the MMV problem jointly can lead to better uniqueness guarantees than solving the SMV problem for each vector independently [35].

In the current MMV literature, a jointly sparse matrix is recovered typically by one of the following methods: 1) greedy methods [120] like Simultaneous Orthogonal Matching Pursuit (SOMP) which performs non-optimal subset selection, 2) relaxed mixed norm minimization methods [7, 82, 119, 121, 134], or 3) Bayesian methods like [72, 126, 133] where a posterior density function for the values of $\mathbf{S}$ is created, assuming a prior belief, e.g., $\mathbf{Y}$ is observed and $\mathbf{S}$ should be sparse in basis Ψ. The selection of one of the above methods depends on the requirements imposed by the specific application.

of $\mathbf{A}$ (Definition 2 of [33]).

1.3 Motivations and Contributions

In this thesis we study the applications of deep learning methods for sequence modeling, and how to use them in favor of better performance for the target machine learning task. Our focus is on three important applications related to speech recognition, sentence modeling for web search and distributed compressive sensing. For each application, we explain the problem and present our proposed method to address the problem followed by a list of our contributions.

We start the thesis by proposing a novel architecture for sequence learning that is a stack of one layer recurrent neural networks with linear output units and initialized by echo state networks. We refer to this architecture as Recurrent Deep Stacking Network (R-DSN) and evaluate its performance on a speech recognition task. We then study the information retrieval task and propose a sentence embedding method that uses Long Short-Term Memory (LSTM), an special neural network architecture for sequence modelling, and show that it yields significantly better results than state-of-the-art methods on a real world commercial search engine's click-through data. Finally, we target the problem of distributed Compressive Sensing (CS) and propose a deep learning approach for this problem, i.e., a new sparse reconstruction algorithm (for data compressed by CS) that is based on deep learning. We show that the proposed method, is almost as fast as greedy methods, performs better than a number of strong well known methods and effectively addresses the MMV problem when sparse vectors are not jointly sparse.

Below we explain the motivation behind each work and briefly list our contributions in each area.

1.3.1 Recurrent Deep Stacking Networks for Speech Recognition

Echo State Network (ESN) is a special type of the temporally deep network model, the Recurrent Neural Network (RNN), where the recurrent matrix is carefully designed and both the recurrent and input the matrices are fixed. ESN uses a linear activation function for the output units. It uses this linearity to simplify the learning of the output matrix. The Deep Stacking Network (DSN), on the other hand, is constructed by stacking shallow feed-forward neural networks with linear output units on top of each other using concatenated features derived from the lower modules of the DSN and the raw input data. DSNs do not have recurrent connections, making them less effective in modelling and classifying input data with temporal dependencies. Our contributions in the area of sequence modelling methods of neural networks with linear output units are as follows:

- we devise a special technique that takes advantage of the linearity in the output units of an ESN, to learn the input and recurrent matrices. This has not been done in earlier ESNs due to their well known difficulty in learning these matrices.

- Compared to the technique of BPTT in learning general RNNs, our proposed method exploits the linearity of the activation function in the output units to formulate the relationships amongst the various matrices in an RNN. These relationships result in the gradient of the cost function having an analytical form. This has the advantage of enabling us to compute the gradients faster and more accurately instead of obtaining them by recursion as in BPTT.

- we embed recurrent connections into the DSN, giving rise to what we call an R-DSN. Each module of the R-DSN consists of a special form of recurrent neural networks. Generalizing from the earlier DSN, the use of linearity in the output units of the R-DSN enables us to derive a closed form for computing the gradient of the cost function with respect to all network matrices without backpropagating errors.

1.3.2 A Sentence Modelling Method for Web Search Task

Learning a good representation (or features) of input data is an important task in machine learning. In text and language processing, one such problem is the learning of an embedding vector for a sentence; that is, to train a model that can automatically transform a sentence to a vector that encodes the semantic meaning of the sentence. Our contributions in the area of sentence embedding and information retrieval are:

- Developing a model that addresses sentence embedding, a hot topic in current natural language processing research, using an RNN with LSTM cells. The proposed LSTM-RNN model sequentially takes each word in a sentence, extracts its information, and embeds it into a semantic vector.

- Training above LSTM-RNN model in a weakly supervised manner on users' click-through data logged by a commercial web search engine.

- Performing visualization and analysis to understand how the sentence embedding process works. The model is found to automatically attenuate the unimportant words and detects the salient keywords in the sentence. Furthermore, these detected keywords are found to automatically activate different cells of the LSTM-RNN, where words belonging to a similar topic activate the same cell.

- As a semantic representation of the sentence, the embedding vector can be used in many different applications. These automatic keyword detection and topic allocation abilities enabled by the LSTM-RNN allow the network to perform document retrieval, a difficult language processing task. On a web search task, the LSTM-RNN embedding is shown to significantly outperform several existing state-of-the-art methods.

1.3.3 A Deep Learning Approach to Distributed Compressive Sensing

Various studies that address the problem of Multiple Measurement Vectors (MMVs), also known as distributed compressive sensing, have been recently carried. Most of these studies assume the different sparse vectors in such a problem to be jointly sparse. They do not use complicated structures beyond sparsity or block sparsity or tree structures and do not use available offline data during sparse reconstruction. Our contributions in the area of compressive sensing are as follows:

- We relax the joint sparsity condition. Instead, we assume that these sparse vectors depend on each other but that this dependency is unknown. We capture this dependency by computing the conditional probability of each entry in each vector being non-zero, given the *residuals* of all previous vectors. To estimate these probabilities, we propose to use deep sequence modelling methods.

- To reconstruct the sparse vectors at the decoder, we propose a greedy solver that uses the above deep sequence model to estimate the conditional probabilities. By performing extensive experiments on two real world datasets, we show that the proposed method significantly outperforms the general MMV solver (the Simultaneous Orthogonal Matching Pursuit (SOMP)) and a number of the model-based Bayesian methods. The proposed method does not alter or add any complexity to the general compressive sensing encoder. The trained model is only used to help the reconstruction algorithm at the decoder.

- We propose a bidirectional version of the above solver and show that it performs better than above solver.

- We propose a Convolutional architecture for Deep Stacking Networks which results in Convolutional Deep Stacking Network (CDSN). As a sample application of this architecture, we use it in combination with the above proposed greedy solver and obtain significant performance improvements compared to SOMP and a number of model-based Bayesian methods.

1.4 Organization

Every chapter starts with an introduction section where the problem description is stated, followed by a review of the related background. Then our proposed solution for the described problem is explained. This is followed by the evaluation and comparison of the proposed solution against the state-of-the-art methods. A summary of each chapter is given in the final section of that chapter. More details about the organization of each chapter are:

- In Chapter 2 the problem of deep sequence modelling for neural networks with linear output units is addressed. Two models are proposed, the first is a modified ESN whose input and recurrent weights are learned in a special manner, and the second is the Recurrent Deep Stacking Network (R-DSN). The models are evaluated on the TIMIT dataset for the speech recognition task. The performance is comparable with state-of-the-art methods.

- In Chapter 3 the problem of sentence embedding is addressed. We have proposed and showed how to use a deep sequence modelling method, with Long Short-Term Memory (LSTM) cells, for sentence embedding. The results were significantly better than those of the state-of-the-art methods on the difficult task of information retrieval using click-through data. We performed extensive visualization and analysis and showed that the proposed model can be used for key word extraction and topic modelling.

- In Chapter 4 the problem of distributed compressive sensing is addressed. We proposed a deep learning approach to distributed compressive sensing. We showed how deep sequence modelling methods can help to (a) relax the joint sparsity condition (b) use structures beyond sparsity during reconstruction (c) use available offline data during reconstruction and (d) obtain the results of sparse reconstruction almost as fast as greedy methods.

- In Chapter 5 we conclude this thesis and provide a summary of our contributions. We also discuss future work and present potential opportunities for improvements and extensions of the methods proposed in this thesis.

Chapter 2

Recurrent Deep Stacking Networks for Speech Recognition

It is not the knowledge but the act of learning, not possession but the act of getting there, which grants the greatest enjoyment.
— Carl Friedrich Gauss

2.1 Introduction

Deep Neural Network (DNN) have proven to yield excellent performance in many pattern classification and recognition tasks [30, 55]. To fine tune DNNs, the stochastic gradient descent method is often used. This makes it difficult to parallelize the learning across different machines. As one way to overcome this problem, Deep Stacking Networks (DSNs) have been proposed [26, 29]. Motivated by the stacking concept of [127], (where complex functions are learnt by stacking a number of simple functions), a DSN is constructed by stacking feed-forward neural networks with one hidden layer having a non-linear activation function for the hidden units and a linear activation function for the output units[29]. Training each module is performed independently of other modules, and there is no need to back propagate the error from the output to the input layers. There are however no temporal connections in each module of a DSN; therefore, the temporal dependencies in the input data are not learnt effectively in DSNs.

Recurrent Neural Networks (RNNs) belong to a general type of deep neural networks which are used to model time sequences and dynamical systems [13, 28, 36, 49, 85, 103]. Echo State Networks (ESNs) belong to the general class of the RNNs [66, 67, 69, 118]. The following properties of an ESN make it distinct from other types of RNNs. First, both the recurrent and input matrices in an ESN are fixed and not learned. This is largely due to the difficulty in learning RNN [12, 99]. Second, the number of hidden neurons in an ESN is typically much larger than those in regular RNNs. The main challenges of training RNNs described in [12, 99] is avoided in ESN by not training most of the very large number of difficult network parameters. This also leads to avoiding potential overfitting problems. Third, the output units, also called readout units, in an ESN are linear. This is unlike the typically nonlinear output units in regular RNNs. Given the very large number of hidden neurons of an ESN, the output or readout weight matrix is very large

as well. The use of linear output units allows the output weight matrix to be learned very efficiently and with a simple regularization mechanism based on ridge regression. Fourth, the learning of the ESN parameters (i.e. output matrix) is much simpler than that for regular RNNs. The former uses linear learning with convex optimization, and the latter, is based typically on Back Propagation Through Time (BPTT) and is highly nonlinear and non-convex. As a result, learning the ESN parameters can be effectively carried out via batch training. This greatly facilitates parallel implementation. Learning the general RNN parameters on the other hand, typically requires stochastic gradient descent, and is more difficult for parallelization.

The simplicity in ESN learning comes at the cost of not learning some important parameters (including the input and the recurrent weight matrices) and of using linear output units. While the special design of the recurrent matrices (see [68]) and the use of a large number of hidden neurons help in reducing the weakness of using fixed parameters, it is desirable to make these parameters adapt to the data. This is as long as the required learning remains simpler and more parallelizable than the common BPTT learning method applied to regular RNNs.

This chapter presents this type of learning for the input and recurrent matrices of ESNs. We propose a technique that makes full use of the linearity in the output units when constructing constraints on all three input, output, and recurrent matrices in an ESN. The constraints enable us to compute the gradients as the learning signal in an analytical form, and this makes the gradient estimate more accurate than when computed by recursion as in BPTT. Our preliminary experimental results on phoneme classification are highly positive. It is demonstrated that learning one or both of the input and recurrent matrices in an ESN gives better phoneme classification accuracy than that obtained by a traditional ESN without learning them. Furthermore, when longer time steps are used in analytically computing the gradients, the better classification results are obtained.

To model input data having temporal dependencies more effectively, we also introduce the Recurrent Deep Stacking Network (R-DSN) that combines the strengths of both DSNs and RNNs while overcoming some of their weaknesses. The proposed R-DSN has recurrent (temporal) connections that are missing in the DSN. In the R-DSN, each module is a single-layer RNN with linear output units. To train each module, we do not use BPTT. Instead, we initialize the weights using ESN, and then fine tune them using batch-mode gradient descent based on a closed-form formulation to compute the gradients. After each epoch of fine tuning, the echo state property is also forced to be satisfied.

2.2 General Recurrent Networks and Specific Echo State Networks

A general RNN has temporal connections as well as input-to-hidden layer, hidden layer-to-output connections. These connections are mathematically represented by the recurrent weight matrix $\mathbf{W}_{rec}$, the input weight matrix $\mathbf{W}$, and the output weight matrix $\mathbf{U}$, respectively. The RNN architecture is illustrated in Fig. 2.1. It also includes input-to-output and output-to-hidden (feedback) connections, with the latter denoted by $\mathbf{W}_{fb}$. The sequential sections of Fig. 2.1(a), 2.1(b), 2.1(c), ..., denote the RNN as it unfolds in time. Note that all the weight matrices are constrained to be the same (i.e. they are tied) at any discrete point in time.

In Figure 1, $\mathbf{x}_i$, $\mathbf{h}_i$ and $\mathbf{y}_i$ represent the input, hidden and output vectors at discrete time $t = i$. Again, the connections between the input ($\mathbf{x}_i$) and hidden ($\mathbf{h}_i$) layers, the hidden and output ($\mathbf{y}_i$) layers, and the output

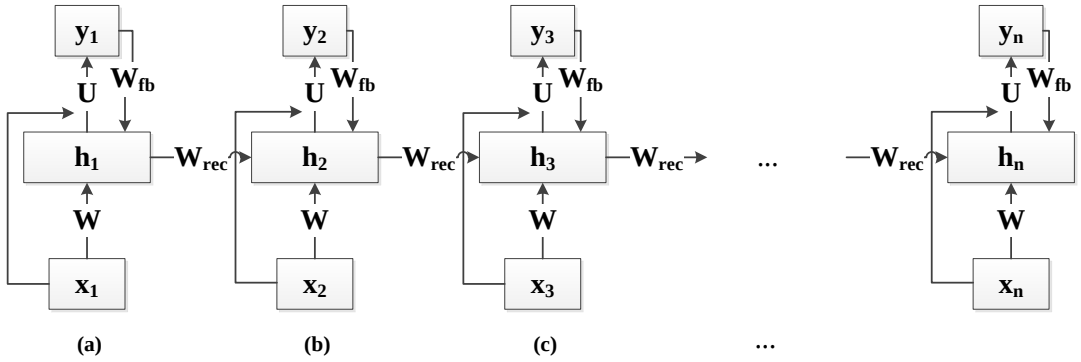


Figure 2.1: Illustration of a general RNN unfolded over time

and hidden layers are represented by $\mathbf{W}$, $\mathbf{U}$ and $\mathbf{W}_{fb}$ respectively. The temporal connection between $\mathbf{h}_i$ and $\mathbf{h}_{i+1}$ is represented by the matrix $\mathbf{W}_{rec}$. Note that the direct connections from the input to the output layer form a part of the matrix $\mathbf{U}$; i.e., it is equivalent to concatenating the input layer with the hidden layer.

There are two standard methods to train RNNs, BPTT and the method based on Extended Kalman Filtering (EKF) [68]. BPTT is a first order method, which extends error backpropagation for feed-forward networks by treating each time step as a new hidden layer (but ties all the weight matrices across time). Generally, BPTT has slow convergence. Its difficulty in capturing long-term memory due to vanishing gradient and exploding gradient problems has been well known for many years [12]. It is often non-trivial to obtain good results with BPTT; see the tremendous amount of engineering required to make BPTT work [86, 103, 113]. The EKF-based method, on the other hand, has fast convergence properties and belongs to a second-order method. However, the computational requirements of the EKF method are high and its implementation is non-trivial [68], especially for large-scale problems.

One prominent approach that has been proposed to overcome the difficulty in training RNNs is the ESN [69],[68]. As explained above, an ESN is a special type of RNNs whose recurrent weights ($\mathbf{W}_{rec}$) and input to hidden layer weights ($\mathbf{W}$) are fixed and only $\mathbf{U}$ (that represents the hidden layer to output and the input to output weights) is trained. The recurrent connections in $\mathbf{W}_{rec}$ are sparse and their values are carefully fixed in a way that the echo-state property is preserved. ESNs can be trained very fast because the only connections that are trained are the output connections. With good initialization, ESNs have been shown to yield good performance for one dimensional sequences; but for high dimensional data such as speech, the studies have been relatively limited; please see [118].

Since the output units in an ESN have a linear activation function and assuming the hidden units have a sigmoid activation function, the formulation of an ESN as a special type of RNNs (as shown in Fig. 2.1) can be succinctly described by

$$\mathbf{h}_{i+1} = \sigma(\mathbf{W}^T\mathbf{x}_{i+1} + \mathbf{W}_{rec}\mathbf{h}_i + \mathbf{W}_{fb}\mathbf{y}_i) \tag{2.1}$$

$$\mathbf{y}_{i+1} = \mathbf{U}^T\mathbf{h}_{i+1}, \tag{2.2}$$

where $\sigma(x) = \frac{1}{1+e^{-x}}$. As discussed earlier, an ESN has a special designed recurrent matrix, which is endowed

with the echo state property in most existing versions [68, 69, 118]. The echo state property implies that the state, or the hidden units' activities, of the network can be determined uniquely based on the current and previous inputs and outputs provided the network has been running for a sufficiently long time. The formal definition of echo states is described in [68]. Assume that the maximum eigenvalue of $\mathbf{W}_{rec}$ is λ_{max}, the activation function of the hidden units is sigmoid and $\mid \lambda_{max} \mid > 4$, then the network does not have echo states. This is a sufficient condition for the echo states to not exist[1] [68]. However, as emphasized in [68], in practice, when $\mid \lambda_{max} \mid < 4$ the network has echo states. There is a similar sufficient condition under which the exploding gradient problem for recurrent weights would not happen [98].

In ESN training, only the output weight matrix, $\mathbf{U}$, is trained. The input and recurrent weight matrices should be carefully fixed. There are three main steps in training an ESN: constructing a network with the echo state property, computing the network states, and estimating the output weights.

To construct a network with the echo state property, the input weight matrix $\mathbf{W}$ and the sparse recurrent weight matrix $\mathbf{W}_{rec}$ are randomly generated. Then, the maximum eigenvalue of $\mathbf{W}_{rec}$ is calculated and all entries of $\mathbf{W}_{rec}$ are renormalized as follows:

$$\mathbf{W}_{rec} = \lambda \frac{\mathbf{W}_{rec}}{\lambda_{max}}, \tag{2.3}$$

where $\lambda < 4$ for sigmoid activation function. λ is also an important parameter which affects the memory length of the network and should be determined based on the required memory size for the specific task. As emphasized in [68], the entries of the input weight matrix are also of great importance; large entries cause most hidden units to saturate while small entries lead hidden units to stay in the linear region of the sigmoid function. The value of λ should be predetermined and fixed (based on the desired task) before the second step (that calculates the network states).

To find the outputs of the hidden layer units, the hidden states are initialized to zero or another initial state. Then the network runs freely for i_{trans} time steps where the hidden states of each time step are calculated using (2.1) with $\mathbf{W}_{fb} = 0$. After i_{trans} time steps, the hidden state vectors are stacked in matrix $\mathbf{H}$, i.e.

$$\mathbf{H} = [\mathbf{h}_{i_{trans}} \mathbf{h}_{i_{trans}+1} \dots \mathbf{h}_N], \tag{2.4}$$

where N is the number of time steps.

To calculate the output weights $\mathbf{U}$, we stack the desired outputs or targets corresponding to input signal $\mathbf{x}_i$ as a matrix $\mathbf{T}$; i.e.

$$\mathbf{T} = [\mathbf{t}_{i_{trans}} \mathbf{t}_{i_{trans}+1} \dots \mathbf{t}_N]. \tag{2.5}$$

Since $\mathbf{H}$ is computed using all known quantities (incluidng the fixed input and recurrent matrices) using (2.1) and hence it is known also, $\mathbf{U}$ can be obtained by minimizing the following mean-square-error cost function:

$$E = \| \mathbf{U}^T \mathbf{H}_c - \mathbf{T} \|_F^2 = tr[(\mathbf{U}^T \mathbf{H}_c - \mathbf{T})(\mathbf{U}^T \mathbf{H}_c - \mathbf{T})^T], \tag{2.6}$$

[1] Note that in [68] the condition is stated as $\mid \lambda_{max} \mid > 1$ because the activation function considered is hyperbolic tangent.

where F stands for the Frobenius norm of a matrix, $tr(.)$ is the trace of a matrix and

$$\begin{aligned} &\mathbf{H}_c = [\mathbf{H}\,\mathbf{X}] \\ &\mathbf{X} = [\mathbf{x}_{i_{trans}}\mathbf{x}_{i_{trans}+1}\ldots\mathbf{x}_N]. \end{aligned} \tag{2.7}$$

Minimizing (2.6), we have the globally optimal estimate of $\mathbf{U}$ determined by setting the gradient of the above cost function to zero and solving it; i.e.

$$\begin{aligned} &\frac{\partial E}{\partial U} = 2\mathbf{H}_c(\mathbf{U}^T\mathbf{H}_c - \mathbf{T})^T = 0 \\ &\mathbf{U} = (\mathbf{H}_c\mathbf{H}_c^T)^{-1}\mathbf{H}_c\mathbf{T}^T. \end{aligned} \tag{2.8}$$

In practical implementation, to prevent inaccurate results when $\mathbf{H}\mathbf{H}^T$ is singular or close to singular, the following solution of "ridge regression" is used for estimating $\mathbf{U}$

$$\mathbf{U} = (\mathbf{H}_c\mathbf{H}_c^T + \mu\mathbf{I})^{-1}\mathbf{H}_c\mathbf{T}^T, \tag{2.9}$$

where $\mathbf{I}$ is the identity matrix and μ is a fixed positive number.

2.3 Learning the Input Weight Matrix in ESN

Assuming the memory of the network extends back to m time steps, we use the following notation to facilitate the development of the learning method for the input weight matrix $\mathbf{W}$:

$$\begin{aligned} &\mathbf{X}_1 = [\mathbf{x}_1\ \mathbf{x}_{m+1}\ \mathbf{x}_{2m+1}\ldots],\ \mathbf{X}_2 = [\mathbf{x}_2\ \mathbf{x}_{m+2}\ \mathbf{x}_{2m+2}\ldots],\ \ldots \\ &\mathbf{H}_1 = [\mathbf{h}_1\ \mathbf{h}_{m+1}\ \mathbf{h}_{2m+1}\ldots],\ \mathbf{H}_2 = [\mathbf{h}_2\ \mathbf{h}_{m+2}\ \mathbf{h}_{2m+2}\ldots],\ \ldots \\ &\mathbf{T}_1 = [\mathbf{t}_1\ \mathbf{t}_{m+1}\ \mathbf{t}_{2m+1}\ldots],\ \mathbf{T}_2 = [\mathbf{t}_2\ \mathbf{t}_{m+2}\ \mathbf{t}_{2m+2}\ldots],\ \ldots. \end{aligned} \tag{2.10}$$

Therefore, equations (2.1) and (2.2) can be written as

$$\mathbf{H}_{i+1} = \sigma(\mathbf{W}^T\mathbf{X}_{i+1} + \mathbf{W}_{rec}\mathbf{H}_i) \tag{2.11}$$

$$\mathbf{Y}_{i+1} = \mathbf{U}^T\mathbf{H}_{i+1}. \tag{2.12}$$

To find the gradient of the cost function E with respect to $\mathbf{W}$ and learn input weights $\mathbf{W}$ we consider two cases in the remainder of this section. In Case 1, we assume that $\mathbf{U}$ does not depend on $\mathbf{W}$ and in Case 2 we take into account the dependency between $\mathbf{U}$ and $\mathbf{W}$. Note that in both cases we take into account the time dependency among the hidden state vectors in $\mathbf{H}$, i.e., the dependency of $\mathbf{h}_{i+1}$ on $\mathbf{h}_i$ at every time step i [2]. Since Case 2 is a more realistic formulation of the gradient, it is used for learning the input weight matrix in our experimental results presented in Section 5. We derive the gradient for one time step dependency and then generalize it to n time step dependency, i.e., $\mathbf{H}_i$ depends on $\mathbf{H}_{i-1}$, $\mathbf{H}_{i-2}$ and so on up to n time steps.

[2]Note that this is one of the main differences with the work presented in [26, 130] where there is no temporal connection in the single layer network and hence no time dependency is considered.

2.3.1 Case 1

The gradient of the cost function with respect to $\mathbf{W}$ can be written as

$$\begin{aligned}
&\frac{\partial E}{\partial \mathbf{W}} = \frac{\partial}{\partial \mathbf{W}} tr[(\mathbf{U}^T\mathbf{H}_2 - \mathbf{T}_2)(\mathbf{U}^T\mathbf{H}_2 - \mathbf{T}_2)^T] \\
&= \frac{\partial}{\partial \mathbf{W}} tr[(\mathbf{U}^T\sigma(\mathbf{W}_{rec}\mathbf{H}_1 + \mathbf{W}^T\mathbf{X}_2) - \mathbf{T}_2)(\mathbf{U}^T\sigma(\mathbf{W}_{rec}\mathbf{H}_1 + \mathbf{W}^T\mathbf{X}_2) - \mathbf{T}_2)^T] \\
&= [\frac{\partial}{\partial \mathbf{W}}\sigma(\mathbf{W}_{rec}\mathbf{H}_1 + \mathbf{W}^T\mathbf{X}_2)][2\mathbf{U}^T(\mathbf{U}^T\mathbf{H}_2 - \mathbf{T}_2)^T].
\end{aligned} \tag{2.13}$$

Assuming that $\mathbf{H}_1$ depends on $\mathbf{W}$, i.e., $\mathbf{H}_1 = \sigma(\mathbf{W}_{rec}\mathbf{H}_0 + \mathbf{W}^T\mathbf{X}_1)$, and denoting the term independent of $\mathbf{W}$ by

$$\mathbf{S} = 2\mathbf{U}^T(\mathbf{U}^T\mathbf{H}_2 - \mathbf{T}_2)^T, \tag{2.14}$$

then using chain rule of calculus we have

$$\frac{\partial E}{\partial \mathbf{W}} = [\frac{\partial}{\partial \mathbf{W}}[\mathbf{W}_{rec}\sigma(\mathbf{W}_{rec}\mathbf{H}_0 + \mathbf{W}^T\mathbf{X}_1) + \mathbf{W}^T\mathbf{X}_2]]\mathbf{H}_2^T \circ (\mathbf{1} - \mathbf{H}_2^T) \circ \mathbf{S}, \tag{2.15}$$

and therefore

$$\frac{\partial E}{\partial \mathbf{W}} = \mathbf{X}_1[\mathbf{H}_1^T \circ (\mathbf{1} - \mathbf{H}_1^T)\mathbf{W}_{rec}^T \circ \mathbf{H}_2^T \circ (\mathbf{1} - \mathbf{H}_2^T) \circ \mathbf{S}] + \mathbf{X}_2[\mathbf{H}_2^T \circ (\mathbf{1} - \mathbf{H}_2^T) \circ \mathbf{S}], \tag{2.16}$$

where $\circ$ is element-wise multiplication.

2.3.2 Case 2

The gradient calculated in Case 1 is not accurate because the dependency between $\mathbf{U}$ and $\mathbf{W}$ is ignored. To take this dependency into consideration, the first line of equation (2.13) is rewritten as

$$\frac{\partial E}{\partial \mathbf{W}} = \frac{\partial}{\partial \mathbf{W}} tr(\underbrace{\mathbf{U}^T\mathbf{H}_2\mathbf{H}_2^T\mathbf{U}}_{a} - \underbrace{\mathbf{U}^T\mathbf{H}_2\mathbf{T}_2^T}_{b} - \mathbf{T}_2\mathbf{H}_2^T\mathbf{U} + \mathbf{T}_2\mathbf{T}_2^T). \tag{2.17}$$

Substituting $\mathbf{U} = (\mathbf{H}_2\mathbf{H}_2^T)^{-1}\mathbf{H}_2\mathbf{T}_2^T$ we have

$$\begin{aligned}
a &= \mathbf{T}_2\mathbf{H}_2^T(\mathbf{H}_2\mathbf{H}_2^T)^{-T}\underbrace{\mathbf{H}_2\mathbf{H}_2^T(\mathbf{H}_2\mathbf{H}_2^T)^{-1}}_{\mathbf{I}}\mathbf{H}_2\mathbf{T}_2^T \\
&= \mathbf{T}_2\mathbf{H}_2^T(\mathbf{H}_2\mathbf{H}_2^T)^{-T}\mathbf{H}_2\mathbf{T}_2^T,
\end{aligned} \tag{2.18}$$

and

$$b = \mathbf{T}_2\mathbf{H}_2^T(\mathbf{H}_2\mathbf{H}_2^T)^{-T}\mathbf{H}_2\mathbf{T}_2^T, \tag{2.19}$$

therefore

$$\frac{\partial E}{\partial \mathbf{W}} = -\frac{\partial}{\partial \mathbf{W}} tr(\mathbf{T}_2\mathbf{H}_2^T(\mathbf{H}_2\mathbf{H}_2^T)^{-1}\mathbf{H}_2\mathbf{T}_2^T). \tag{2.20}$$

Since $tr(\mathbf{AB}) = tr(\mathbf{BA})$ and $tr(\mathbf{A}) = tr(\mathbf{A}^T)$ we have

$$\frac{\partial E}{\partial \mathbf{W}} = -\frac{\partial}{\partial \mathbf{W}} tr((\mathbf{H}_2\mathbf{H}_2^T)^{-1}\mathbf{H}_2\mathbf{T}_2^T\mathbf{T}_2\mathbf{H}_2^T). \tag{2.21}$$

Using the chain rule presented in equation (126) of [100], the gradient can be written as

$$\frac{\partial E}{\partial \mathbf{W}} = -[\underbrace{\frac{\partial}{\partial \mathbf{W}}\mathbf{H}_2}_{\mathbf{B}}][\underbrace{\frac{\partial}{\partial \mathbf{H}_2^T} tr((\mathbf{H}_2\mathbf{H}_2^T)^{-1}\mathbf{H}_2\mathbf{T}_2^T\mathbf{T}_2\mathbf{H}_2^T)}_{\mathbf{A}}]. \tag{2.22}$$

Below we first calculate matrix $\mathbf{A}$ and then matrix $\mathbf{B}$.

For constant values of hidden states $\mathbf{H}_2$ we define $\mathbf{F}$, $\mathbf{M}$ and $\mathbf{S}$ as follows:

$$\begin{aligned} \mathbf{F} &= (\mathbf{H}_2\mathbf{H}_2^T)^{-1} \\ \mathbf{M} &= \mathbf{T}_2^T\mathbf{T}_2 \\ \mathbf{S} &= \mathbf{H}_2\mathbf{M}\mathbf{H}_2^T. \end{aligned} \tag{2.23}$$

Then, using the chain rule again, we obtain

$$\mathbf{A} = \underbrace{\frac{\partial}{\partial \mathbf{H}_2^T} tr(\mathbf{F}\mathbf{H}_2\mathbf{M}\mathbf{H}_2^T)}_{\mathbf{A}_1} + \underbrace{\frac{\partial}{\partial \mathbf{H}_2^T} tr((\mathbf{H}_2\mathbf{H}_2^T)^{-1}\mathbf{S})}_{\mathbf{A}_2}, \tag{2.24}$$

considering the fact that $tr(\mathbf{AB}) = tr(\mathbf{BA})$, $\mathbf{A_1}$ can be written as

$$\mathbf{A}_1 = \frac{\partial}{\partial \mathbf{H}_2^T} tr(\mathbf{M}\mathbf{H}_2^T\mathbf{F}\mathbf{H}_2). \tag{2.25}$$

From equation (107) of [100] we have

$$\mathbf{A}_1 = \mathbf{M}^T\mathbf{H}_2^T\mathbf{F}^T + \mathbf{M}\mathbf{H}_2^T\mathbf{F}. \tag{2.26}$$

Since $\mathbf{F}^T = \mathbf{F}$ and $\mathbf{M}^T = \mathbf{M}$, $\mathbf{A}_1$ can be written as

$$\mathbf{A}_1 = 2\mathbf{M}\mathbf{H}_2^T\mathbf{F}. \tag{2.27}$$

Considering equation (114) of [100], $\mathbf{A}_2$ will be as follows

$$\mathbf{A}_2 = -(\mathbf{H}_2^T(\mathbf{H}_2\mathbf{H}_2^T)^{-1})(\mathbf{S} + \mathbf{S}^T)(\mathbf{H}_2\mathbf{H}_2^T)^{-1}, \tag{2.28}$$

substituting $\mathbf{S}$ and using $\mathbf{M} = \mathbf{M}^T$ results in

$$\mathbf{A}_2 = -2\mathbf{H}_2^T(\mathbf{H}_2\mathbf{H}_2^T)^{-1}(\mathbf{H}_2\mathbf{M}\mathbf{H}_2^T)(\mathbf{H}_2\mathbf{H}_2^T)^{-1}. \tag{2.29}$$

Now substituting $\mathbf{M}$ and $\mathbf{F}$ in (2.23) results in the final formulation for $\mathbf{A}$ as follows:

$$\mathbf{A} = 2\mathbf{T}_2^T\mathbf{T}_2\mathbf{H}_2^T(\mathbf{H}_2\mathbf{H}_2^T)^{-1} - 2\mathbf{H}_2^T(\mathbf{H}_2\mathbf{H}_2^T)^{-1}\mathbf{H}_2\mathbf{T}_2^T\mathbf{T}_2\mathbf{H}_2^T(\mathbf{H}_2\mathbf{H}_2^T)^{-1}. \tag{2.30}$$

To calculate $\mathbf{B}$, we assume that there is just one time step dependency, i.e., $\mathbf{H}_2$ depends on $\mathbf{H}_1$ and $\mathbf{W}$ and $\mathbf{H}_1$ does not depend on $\mathbf{H}_0$ but depends on $\mathbf{W}$. The generalization to an arbitrary number of time steps is straightforward and is presented at the end of this section. From (2.22) and (2.11) we write $\mathbf{B}$ as

$$\mathbf{B} = \frac{\partial}{\partial \mathbf{W}}[\sigma(\mathbf{W}_{rec}\sigma(\mathbf{W}_{rec}\mathbf{H}_0 + \mathbf{W}^T\mathbf{X}_1) + \mathbf{W}^T\mathbf{X}_2)], \tag{2.31}$$

which is similar to the term calculated in (2.16) and therefore

$$\mathbf{B} = \mathbf{X}_1[\mathbf{H}_1^T \circ (\mathbf{1} - \mathbf{H}_1^T)\mathbf{W}_{rec}^T \circ \mathbf{H}_2^T \circ (\mathbf{1} - \mathbf{H}_2^T)] + \mathbf{X}_2[\mathbf{H}_2^T \circ (\mathbf{1} - \mathbf{H}_2^T)]. \tag{2.32}$$

By substituting (2.30) and (2.32) in (2.22) we get the gradient formulation for one time step dependency as follows:

$$\frac{\partial E}{\partial \mathbf{W}} = -[\mathbf{X}_1[\mathbf{H}_1^T \circ (\mathbf{1} - \mathbf{H}_1^T)\mathbf{W}_{rec}^T \circ \mathbf{H}_2^T \circ (\mathbf{1} - \mathbf{H}_2^T) \circ \mathbf{A}] + \mathbf{X}_2[\mathbf{H}_2^T \circ (\mathbf{1} - \mathbf{H}_2^T) \circ \mathbf{A}]]. \tag{2.33}$$

This gradient formulation can be generalized for an arbitrary number of time steps as follows:

$$\frac{\partial E}{\partial \mathbf{W}} = -[\sum_{i=1}^{n} \mathbf{X}_i\mathbf{C}_i], \tag{2.34}$$

where n is the number of time steps and

$$\begin{aligned} &\mathbf{C}_i = [\mathbf{H}_i^T \circ (\mathbf{1} - \mathbf{H}_i^T)\mathbf{W}_{rec}^T] \circ \mathbf{C}_{i+1} \quad , for\ i = 1, \ldots, n-1 \\ &\mathbf{C}_n = \mathbf{H}_n^T \circ (\mathbf{1} - \mathbf{H}_n^T) \circ \mathbf{A}. \end{aligned} \tag{2.35}$$

To calculate $\mathbf{A}$ using (2.30), $\mathbf{H}_n$ and $\mathbf{T}_n$ are used.

After calculating the gradient of the cost function w.r.t $\mathbf{W}$, the input weights $\mathbf{W}$ are updated using the following update equation

$$\mathbf{W}_{i+1} = \mathbf{W}_i - \alpha\frac{\partial E}{\partial \mathbf{W}_i} + \beta(\mathbf{W}_i - \mathbf{W}_{i-1}), \tag{2.36}$$

where α is the step size and

$$\begin{aligned} \beta &= \frac{m_{old}}{m_{new}} \\ m_{new} &= \frac{1 + \sqrt{1 + 4m_{old}^2}}{2}, \end{aligned} \tag{2.37}$$

and where the initial value for m_{old} and m_{new} is 1. The third term in (2.36) helps the algorithm to converge faster and is based on the FISTA algorithm proposed in [11] and used in [130].

2.4 Learning the Recurrent Weight Matrix ($\mathbf{W}_{rec}$) in the ESN

To learn the recurrent weights, the gradient of the cost function w.r.t $\mathbf{W}_{rec}$ should be calculated. We first derive the formulation for the two time steps dependency, i.e., $\mathbf{H}_2$ depends on $\mathbf{H}_1$ and $\mathbf{H}_1$ depends on $\mathbf{H}_0$ but no more time dependencies. Then it will be generalized to the arbitrary number of time steps. The same method that is used in section 2.3 can be used to get the following formulation for the gradient:

$$\frac{\partial E}{\partial \mathbf{W}_{rec}} = -[\underbrace{\frac{\partial}{\partial \mathbf{W}_{rec}}\mathbf{H}_2}_{\mathbf{B}}][\underbrace{\frac{\partial}{\partial \mathbf{H}_2^T} tr((\mathbf{H}_2\mathbf{H}_2^T)^{-1}\mathbf{H}_2\mathbf{T}_2^T\mathbf{T}_2\mathbf{H}_2^T)}_{\mathbf{A}}], \tag{2.38}$$

where $\mathbf{A}$ will be the same as (2.30). To calculate $\mathbf{B}$ we have

$$\begin{aligned}
\mathbf{B} &= \frac{\partial}{\partial \mathbf{W}_{rec}}[\sigma(\mathbf{W}_{rec}\sigma(\mathbf{W}_{rec}\mathbf{H}_0 + \mathbf{W}^T\mathbf{X}_1) + \mathbf{W}^T\mathbf{X}_2)] \\
&= [\frac{\partial}{\partial \mathbf{W}_{rec}}[\mathbf{W}_{rec}\underbrace{\sigma(\mathbf{W}_{rec}\mathbf{H}_0 + \mathbf{W}^T\mathbf{X}_1)}_{\mathbf{H}_1} + \mathbf{W}^T\mathbf{X}_2]]\mathbf{H}_2^T \circ (\mathbf{1} - \mathbf{H}_2^T) \\
&= [\mathbf{H}_1 + \mathbf{W}_{rec}\underbrace{\mathbf{H}_0[\mathbf{H}_1^T \circ (\mathbf{1} - \mathbf{H}_1^T)]}_{\frac{\partial \mathbf{H}_1}{\partial \mathbf{W}_{rec}}}]\mathbf{H}_2^T \circ (\mathbf{1} - \mathbf{H}_2^T)
\end{aligned}, \tag{2.39}$$

and therefore the gradient will be:

$$\frac{\partial E}{\partial \mathbf{W}_{rec}} = \mathbf{H}_1[\mathbf{H}_2^T \circ (\mathbf{1} - \mathbf{H}_2^T) \circ \mathbf{A}] + \mathbf{W}_{rec}\mathbf{H}_0[\mathbf{H}_1^T \circ (\mathbf{1} - \mathbf{H}_1^T) \circ (\mathbf{H}_2^T \circ (\mathbf{1} - \mathbf{H}_2^T) \circ \mathbf{A})]. \tag{2.40}$$

It can be generalized for arbitrary number of time steps as follows:

$$\frac{\partial E}{\partial \mathbf{W}_{rec}} = \sum_{i=1}^{n} \mathbf{W}_{rec}^{n-i}\mathbf{H}_{i-1}\mathbf{C}_i, \tag{2.41}$$

where $\mathbf{H}_0$ includes the initial hidden states and

$$\begin{aligned}
\mathbf{C}_n &= \mathbf{H}_n^T \circ (\mathbf{1} - \mathbf{H}_n^T) \circ \mathbf{A} \\
\mathbf{C}_i &= \mathbf{H}_i^T \circ (\mathbf{1} - \mathbf{H}_i^T) \circ \mathbf{C}_{i+1},
\end{aligned} \tag{2.42}$$

and $\mathbf{A}$ is calculated using (2.30) based on $\mathbf{H}_n$ and $\mathbf{T}_n$.

Only the non-zero entries of the sparse matrix $\mathbf{W}_{rec}$ are updated using (2.36) and the gradient calculated in (2.41). To make sure that the network has the echo state property after each epoch, the entries of $\mathbf{W}_{rec}$ are renormalized such that the maximum eigenvalue of $\mathbf{W}_{rec}$ is λ that is predetermined in (2.3). This renormalization also prevents the gradient explosion problem for recurrent weights from happening.

A summary of the learning method is as follows:

- The echo state network with predetermined maximum eigenvalue of $\mathbf{W}_{rec}$ is constructed based on the explanations presented in section 2.2.

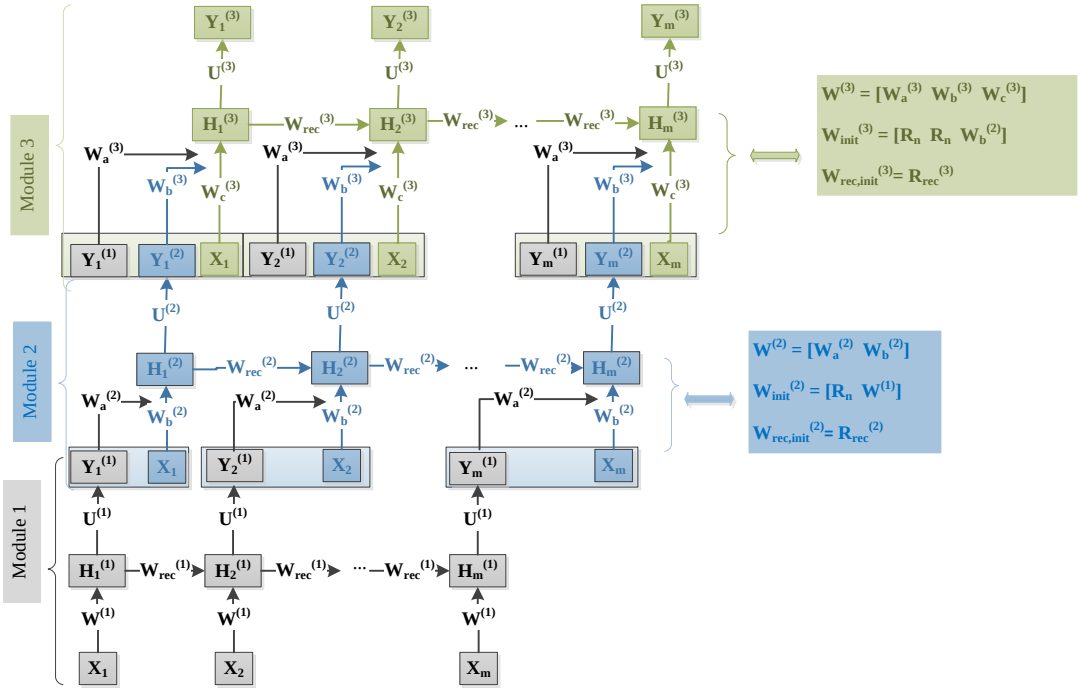


Figure 2.2: Illustration of an R-DSN architecture with three modules shown with different colors

- Input weights matrix $\mathbf{W}$ is updated based on (2.34), (2.35), (2.36) and (2.37).
- Non-zero entries of the sparse recurrent weights matrix $\mathbf{W}_{rec}$ are updated based on (2.41), (2.42), (2.36) for $\mathbf{W}_{rec}$ and (2.37).
- Updated $\mathbf{W}_{rec}$ is renormalized to have the predetermined maximum eigenvalue λ.
- The forward pass is repeated with the updated input and recurrent weights to find the hidden states. The network runs freely for i_{trans} time steps and then the hidden states are recorded as matrix $\mathbf{H}$.
- Taking into account the direct connections from the input to output in the network, the output weight matrix $\mathbf{U}$ is calculated using (2.9).

To prevent the value of the gradient w.r.t $\mathbf{W}$ from exploding, we use a similar approach proposed in [98] where the gradient value is renormalized when it is greater than a threshold.

2.5 Learning the Recurrent Deep Stacking Network

The RNN described in the preceding section can be stacked into multiple modules. The architecture of a R-DSN with three modules is shown in Fig. 2.2. In the R-DSN, the output of each module is part of the input of the upper module. Therefore, the dimensionality of the input of the upper modules is more than that of the lower modules. In this work, we have not used RBM or temporal RBM [114] to initialize the

weights of the lowest module. Instead, we have used random initialization. The only constraint is that $\mathbf{W}_{rec}$ be initialized such that echo state property holds; i.e., the maximum eigenvalue of $\mathbf{W}_{rec}$ is kept less than 4.

The training method we have implemented for the two modules of architecture in Fig. 2.2 is as follows:

- Training the first (i.e the lowest) module:
  - Input and recurrent weights are initialized using an ESN architecture
  - Hidden units' states $\mathbf{H}_i^{(1)}$, $i = 1, \ldots, m$ are calculated using (2.11)
  - The method described in previous sections is used to compute gradients of the cost function w.r.t. $\mathbf{W}^{(1)}$ and $\mathbf{W}_{rec}^{(1)}$
  - $\mathbf{W}^{(1)}$ and $\mathbf{W}_{rec}^{(1)}$ are updated using (2.36) and (2.37)
  - Entries of $\mathbf{W}_{rec}$ are renormalized such that the network has the echo state property using (2.3)
  - $\mathbf{U}^{(1)}$ is calculated using (2.8)
- Training the second module:
  - Input weights corresponding to output of the first module $\mathbf{W}_a^{(2)}$ are initialized randomly ($\mathbf{R}_a$ in Fig. 2.2) and input weights corresponding to the input training data are initialized to the fine tuned input weights from the previous module ($\mathbf{W}^{(1)}$ in Fig. 2.2)
  - Recurrent weights $\mathbf{W}_{rec}^{(2)}$ are initialized with a random sparse matrix whose sparsity pattern is different from that of the previous module($\mathbf{R}_{rec}^{(2)}$ in Fig. 2.2)
  - Entries of $\mathbf{W}_{rec}^{(2)}$ are normalized such that the echo state property holds
  - All weights are fine tuned using the method described for the first module

To resolve the exploding gradient problem for input weights we have used the renormalization method proposed in [99].

All modules higher than two in the R-DSN are trained in a similar way to the second step above. The number of epochs used in training all modules are carefully tuned to provide regularization via the "early stopping" mechanism.

2.6 Experiments

We carried out the experiments for frame-level classification of phoneme states on the TIMIT dataset using an ESN with all parameters learned as discussed so far. The training data includes 1,124,589 frames. The validation set has 122,488 frames from 50 speakers. The results are reported using the core test set consisting of 192 sentences and 57,920 frames. The speech is analysed using the standard Mel Frequency Cepstral Coefficients (MFCC). Each feature vector has 39 entries, including first and second derivatives. We have used 3 states for each of 61 phonemes resulting in a target class vector with 183 entries. phoneme state labels are extracted using a GMM-HMM system which aligns the frames with their corresponding states.

We have used a context window of 3 frames for all experiments resulting in the input vectors with $3 \times 39 = 117$ entries. The regularization parameter μ used in (2.9) is set to 10^{-8}. The maximum eigenvalue of $\mathbf{W}_{rec}$ is set to 3.9. We have used a step size (α in (2.36)) of 0.07. The task is to classify each frame in the TIMIT core test set into one of 183 phoneme states. The results are presented in Table 2.1 for different hidden layer sizes in the ESN, one for each row in the table. The results are also arranged by four different

ways of learning the input and recurrent weigh matrices $\mathbf{W}$ and $\mathbf{W}_{rec}$, where m is the number of time steps in incorporating matrices' dependencies in the learning. The column of "ESN" refers to the traditional ESN as in [66, 67, 69, 118] where $\mathbf{W}$ and $\mathbf{W}_{rec}$ are not learned.

Table 2.1: Frame-level phoneme-state classification error rates for the TIMIT core test set

Hidden units	ESN	Learning $\mathbf{W}$ with $m = 1$	Learning $\mathbf{W}$ and $\mathbf{W}_{rec}$ with $m = 1$	Learning $\mathbf{W}$ and $\mathbf{W}_{rec}$ with $m = 3$
100	75.5%	66.7%	64.0 %	63.2%
500	70.1%	59.8%	57.5%	56.8%
2000	63.8%	54.2%	52.7%	52.1%
10000	57.1%	49.5%	48.0%	46.8%
30000	53.3%	45.9%	44.5%	43.0%

The set of results for the various R-DSNs with the hidden layer sizes of 4000 and 10000 and with the stacking modules up to three are summarized in Table 2.2.

The preliminary experimental results shown in Table 2.1 and 2.2 verify that learning input and recurrent weight matrices in the ESN is superior to the ESN with the same structure but without learning the two matrices. Furthermore, the longer the time steps are incorporated in the learning, the lower error rates are. On the column of "ESN" in Table 2.1, we also observe that the traditional ESN improves its performance as the number of hidden units increases, consistent with the findings reported in [118]. Also stacking more layers improves the performance as observed in Table 2.2. Finally, we would like to remark that the results obtained so far are very preliminary, and the task is on frame-level classification of 183 phoneme states. Our first step of research is focused on this easiest task since it is a pure and simple machine learning problem and it requires no expertise in speech recognition. The next steps are to move 1) from 183-state classification to 39-phoneme classification; 2) from frame level to segment level (which requires dynamic programming over three states of each phoneme); and 3) from classification (with no phoneme insertion and deletion errors) to recognition (with phoneme insertion and deletion errors).

2.7 Conclusions

The first idea of this chapter is straightforward: the traditional ESN learns only one of three important sets of weight matrices, and we want to learn them all. The key property that characterizes the ESN is the use of linear output units so that the learning is simple, convex and forms a least-square ridge regression problem with a global optimum (in learning the output weights). In extending the learning of the output weights to only learning input and recurrent weights, we make use of the same property of linear output units to develop and formulate constraints among various sets of ESN weight matrices. Such constraints are then used to derive analytic forms of the error gradients w.r.t the input and recurrent weights to be learned. The standard learning method of BPTT for the general RNN (with typically nonlinear output units) does not admit analytical forms of gradient computation. BPTT requires recursively propagating the error signal backward through time, a very different style of computation and learning than what we have developed in

Table 2.2: Frame level phoneme classification error rates for the R-DSN on the TIMIT core test set as a function of the number of modules for the fixed 4000 or 10000 neurons in the hidden layer

Hidden Units	Modules in R-DSN	Learning $\mathbf{W}$ and $\mathbf{W}_{rec}$ with $m = 1$	Learning $\mathbf{W}$ and $\mathbf{W}_{rec}$ with $m = 3$
4000	1	50.5%	50.0%
4000	2	49.9%	49.5%
4000	3	49.5%	49.1%
10000	1	48.0%	46.8%
10000	2	47.2%	46.0%
10000	3	47.0%	45.7%

this work for ESNs.

The second idea of this chapter is also straightforward: a novel deep learning architecture, the R-DSN, which extends the earlier RNN and DSN models. The R-DSN constructs multiple modules of the RNN using stacking, in the same way that the DSN uses stacking to form multiple modules of a simple, non-recurrent feed-forward neural network. Alternatively, the R-DSN can be viewed as a generalization of the DSN, where the generalization lies in embedding recurrent connections in each module that were missing in the earlier DSN. The main technical contribution of the work reported in this part is the development of closed-from formulas for the gradient computation based on the special structure of the R-DSN, and the batch-mode training method for all parameters in the R-DSN capitalizing on these formulas.

Chapter 3

A Sentence Modelling Method for Web Search Task

In theory, there is no difference between theory and practice. But, in practice, there is!
— Jan. L. A. van de Snepscheut

3.1 Introduction

Learning a good representation (or features) of input data is an important task in machine learning. In text and language processing, one such problem is learning of an embedding vector for a sentence; that is, to train a model that can automatically transform a sentence to a vector that encodes the semantic meaning of the sentence. While word embedding is learned using a loss function defined on word pairs, sentence embedding is learned using a loss function defined on sentence pairs. In the sentence embedding usually the relationship among words in the sentence, i.e., the context information, is taken into consideration. Therefore, sentence embedding is more suitable for tasks that require computing semantic similarities between text strings. By mapping texts into a unified semantic representation, the embedding vector can be further used for different language processing applications, such as machine translation [116], sentiment analysis [77], and information retrieval [65]. In machine translation, the recurrent neural networks (RNN) with Long Short-Term Memory (LSTM) cells, or the LSTM-RNN, is used to encode an English sentence into a vector, which contains the semantic meaning of the input sentence, and then another LSTM-RNN is used to generate a French (or another target language) sentence from the vector. The model is trained to best predict the output sentence. In [77], a paragraph vector is learned in an unsupervised manner as a distributed representation of sentences and documents, which are then used for sentiment analysis. Sentence embedding can also be applied to information retrieval, where the contextual information are properly represented by the vectors in the same space for fuzzy text matching [65].

In this chapter, we propose to use an RNN to sequentially accept each word in a sentence and recurrently map it into a latent space together with the historical information. As the RNN reaches the last word in the sentence, the hidden activations form a natural embedding vector for the contextual information of the sentence. We further incorporate the LSTM cells into the RNN model (i.e. the LSTM-RNN) to address the

difficulty of learning long term memory in RNN. The learning of such a model is performed in a *weakly supervised* manner on the click-through data logged by a commercial web search engine. Although manually labelled data are insufficient in machine learning, logged data with limited feedback signals are massively available due to the widely used commercial web search engines. Limited feedback information such as click-through data provides a weak supervision signal that indicates the semantic similarity between the text on the query side and the clicked text on the document side. To exploit such a signal, the objective of our training is to maximize the similarity between the two vectors mapped by the LSTM-RNN from the query and the clicked document, respectively. Consequently, the learned embedding vectors of the query and clicked document are specifically useful for web document retrieval task.

An important contribution of this chapter is to analyse the embedding process of the LSTM-RNN by visualizing the internal activation behaviours in response to different text inputs. We show that the embedding process of the learned LSTM-RNN effectively detects the keywords, while attenuating less important words, in the sentence automatically by switching on and off the gates within the LSTM-RNN cells. We further show that different cells in the learned model indeed correspond to different topics, and the keywords associated with a similar topic activate the same cell unit in the model. As the LSTM-RNN reads to the end of the sentence, the topic activation accumulates and the hidden vector at the last word encodes the rich contextual information of the entire sentence. For this reason, a natural application of sentence embedding is web search ranking, in which the embedding vector from the query can be used to match the embedding vectors of the candidate documents according to the maximum cosine similarity rule. Evaluated on a real web document ranking task, our proposed method significantly outperforms many of the existing state of the art methods in NDCG scores. Please note that when we refer to document in this chapter we mean the title (headline) of the document.

3.2 Related Work

Inspired by the word embedding method [87, 89], the authors in [77] proposed an unsupervised learning method to learn a paragraph vector as a distributed representation of sentences and documents, which are then used for sentiment analysis with superior performance. However, the model is not designed to capture the fine-grained sentence structure. In [75], an unsupervised sentence embedding method is proposed with great performance on large corpus of contiguous text corpus, e.g., the BookCorpus [135]. The main idea is to encode the sentence $s(t)$ and then decode previous and next sentences, i.e., $s(t-1)$ and $s(t+1)$, using two separate decoders. The encoder and decoders are RNNs with Gated Recurrent Unit (GRU) [18]. However, this sentence embedding method is not designed for document retrieval task having a supervision among queries and clicked and unclicked documents. In [111], a Semi-Supervised Recursive Autoencoder (RAE) is proposed and used for sentiment prediction. Similar to our proposed method, it does not need any language specific sentiment parsers. A greedy approximation method is proposed to construct a tree structure for the input sentence. It assigns a vector per word. It can become practically problematic for large vocabularies. It also works both on unlabeled data and supervised sentiment data.

Similar to the recurrent models in this chapter, the DSSM [65] and CLSM [108] models, developed for information retrieval, can also be interpreted as sentence embedding methods. However, DSSM treats the

input sentence as a bag-of-words and does not model word dependencies explicitly. CLSM treats a sentence as a bag of n-grams, where n is defined by a window, and can capture *local* word dependencies. Then a Max-pooling layer is used to form a global feature vector. Methods in [19] are also convolutional based networks for Natural Language Processing (NLP). These models, by design, cannot capture long distance dependencies, i.e., dependencies among words belonging to non-overlapping n-grams. In [73] a Dynamic Convolutional Neural Network (DCNN) is proposed for sentence embedding. Similar to CLSM, DCNN does not rely on a parse tree and is easily applicable to any language. However, different from CLSM where a regular max-pooling is used, in DCNN a dynamic k-max-pooling is used. This means that instead of just keeping the largest entries among word vectors in one vector, k largest entries are kept in k different vectors. DCNN has shown good performance in sentiment prediction and question type classification tasks. In [64], a convolutional neural network architecture is proposed for sentence matching. It has shown great performance in several matching tasks. In [132], a Bilingually-constrained Recursive Auto-encoders (BRAE) is proposed to create semantic vector representation for phrases. Through experiments it is shown that the proposed method has great performance in two end-to-end SMT tasks.

Long short-term memory networks were developed in [62] to address the difficulty of capturing long term memory in RNN. It has been successfully applied to speech recognition, which achieves state-of-art performance [50, 107]. In text analysis, LSTM-RNN treats a sentence as a sequence of words with internal structures, i.e., word dependencies. It encodes a semantic vector of a sentence incrementally which differs from DSSM and CLSM. The encoding process is performed left-to-right, word-by-word. At each time step, a new word is encoded into the semantic vector, and the word dependencies embedded in the vector are "updated". When the process reaches the end of the sentence, the semantic vector has embedded all the words and their dependencies, hence, can be viewed as a feature vector representation of the whole sentence. In the machine translation work [116], an input English sentence is converted into a vector representation using LSTM-RNN, and then another LSTM-RNN is used to generate an output French sentence. The model is trained to maximize the probability of predicting the correct output sentence. In [53], there are two main composition models, ADD model that is bag of words and BI model that is a summation over bi-gram pairs plus a non-linearity. In our proposed model, instead of simple summation, we have used LSTM model with letter tri-grams which keeps valuable information over long intervals (for long sentences) and throws away useless information. In [8], an encoder-decoder approach is proposed to jointly learn to align and translate sentences from English to French using RNNs. The concept of "attention" in the decoder, discussed in this paper, is closely related to how our proposed model extracts keywords in the document side. For further explanations please see section 3.5.1. In [74] a set of visualizations are presented for RNNs with and without LSTM cells and GRUs. Different from our work where the target task is sentence embedding for document retrieval, the target tasks in [74] were character level sequence modelling for text characters and source codes. Interesting observations about interpretability of some LSTM cells and statistics of gates activations are presented. In section 3.5.1 we show that some of the results of our visualization are consistent with the observations reported in [74]. We also present more detailed visualization specific to the document retrieval task using click-through data. We also present visualizations about how our proposed model can be used for keyword detection.

Different from the aforementioned studies, the method developed in this chapter trains the model so that sentences that are paraphrase of each other are close in their semantic embedding vectors — see the description in Sec. 3.4 further ahead. Another reason that LSTM-RNN is particularly effective for sentence embedding, is its robustness to noise. For example, in the web document ranking task, the noise comes from two sources: (i) Not every word in query / document is equally important, and we only want to "remember" salient words using the limited "memory". (ii) A word or phrase that is important to a document may not be relevant to a given query, and we only want to "remember" related words that are useful to compute the relevance of the document for a given query. We will illustrate robustness of LSTM-RNN in this chapter. The structure of LSTM-RNN will also circumvent the serious limitation of using a fixed window size in CLSM. Our experiments show that this difference leads to significantly better results in web document retrieval task. Furthermore, it has other advantages. It allows us to capture keywords and key topics effectively. The models in this chapter also do not need the extra max-pooling layer, as required by the CLSM, to capture global contextual information and they do so more effectively.

3.3 Sentence Embedding Using RNNs with and without LSTM Cells

In this section, we introduce the model of recurrent neural networks and its long short-term memory version for learning the sentence embedding vectors. We start with the basic RNN and then proceed to LSTM-RNN.

3.3.1 The Basic Version of RNN

The RNN is a type of deep neural networks that are "deep" in temporal dimension and it has been used extensively in time sequence modelling [13, 17, 25, 28, 36, 49, 84, 85, 103]. The main idea of using RNN for sentence embedding is to find a dense and low dimensional semantic representation by sequentially and recurrently processing each word in a sentence and mapping it into a low dimensional vector. In this model, the global contextual features of the whole text will be in the semantic representation of the last word in the text sequence — see Figure 3.1, where $\mathbf{x}(t)$ is the t-th word, coded as a 1-hot vector, $\mathbf{W}_h$ is a fixed hashing operator similar to the one used in [65] that converts the word vector to a letter tri-gram vector, $\mathbf{W}$ is the input weight matrix, $\mathbf{W}_{rec}$ is the recurrent weight matrix, $\mathbf{y}(t)$ is the hidden activation vector of the RNN, which can be used as a semantic representation of the t-th word, and $\mathbf{y}(t)$ associated to the last word $\mathbf{x}(m)$ is the semantic representation vector of the entire sentence. Note that this is very different from the approach in [65] where the bag-of-words representation is used for the whole text and no context information is used. This is also different from [108] where the sliding window of a fixed size (akin to an FIR filter) is used to capture local features and a max-pooling layer on the top to capture global features. In the RNN there is neither a fixed-sized window nor a max-pooling layer; rather the recurrence is used to capture the context information in the sequence (akin to an IIR filter).

The mathematical formulation of the above RNN model for sentence embedding can be expressed as

$$\begin{aligned}\mathbf{l}(t) &= \mathbf{W}_h\mathbf{x}(t)\\ \mathbf{y}(t) &= f(\mathbf{W}\mathbf{l}(t) + \mathbf{W}_{\text{rec}}\mathbf{y}(t-1) + \mathbf{b}),\end{aligned} \tag{3.1}$$

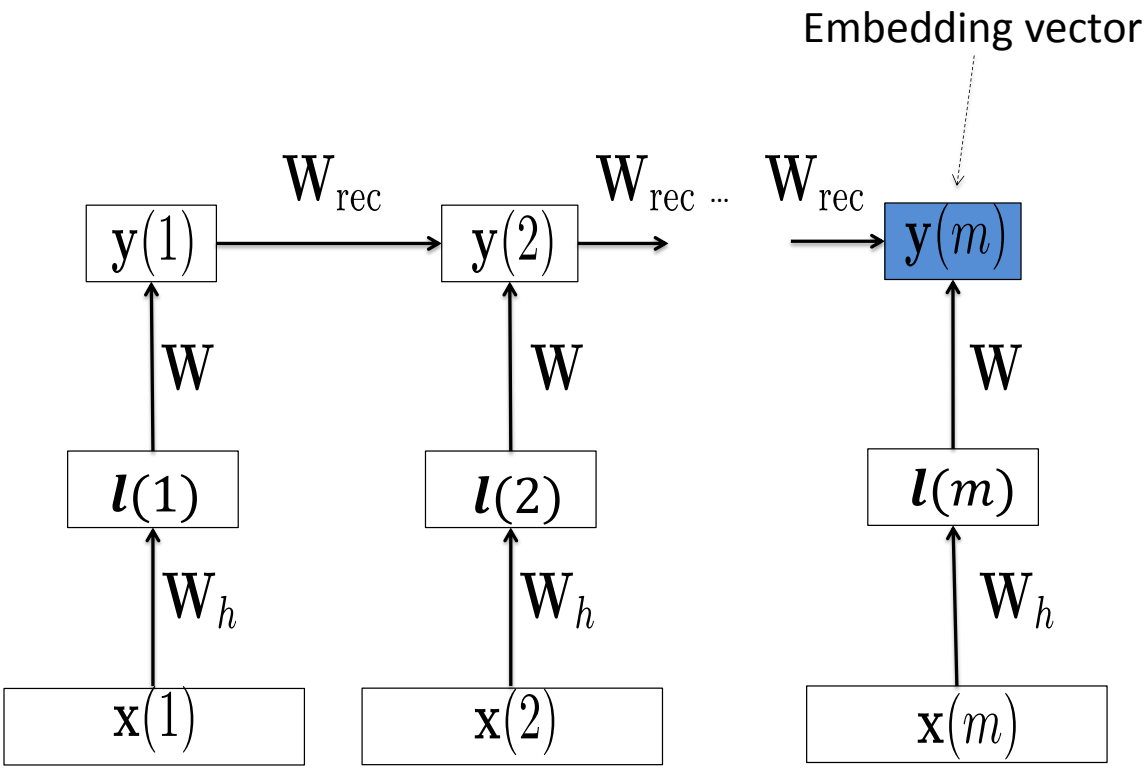


Figure 3.1: The basic architecture of the RNN for sentence embedding, where temporal recurrence is used to model the contextual information across words in the text string. The hidden activation vector corresponding to the last word is the sentence embedding vector (blue).

where $\mathbf{W}$ and $\mathbf{W}_{\text{rec}}$ are the input and recurrent matrices to be learned, $\mathbf{W}_h$ is a fixed word hashing operator, $\mathbf{b}$ is the bias vector and $f(\cdot)$ is assumed to be $\tanh(\cdot)$. Note that the architecture proposed here for sentence embedding is slightly different from traditional RNN in that there is a word hashing layer that convert the high dimensional input into a relatively lower dimensional letter tri-gram representation. There is also no per word supervision during training, instead, the whole sentence has a label. This is explained in more detail in section 3.4.

3.3.2 The RNN with LSTM Cells

Although RNN performs the transformation from the sentence to a vector in a principled manner, it is generally difficult to learn the long term dependency within the sequence due to vanishing gradients problem. One of the effective solutions for this problem in RNNs is using memory cells instead of neurons originally proposed in [62] as Long Short-Term Memory (LSTM) and completed in [45] and [46] by adding forget gate and peephole connections to the architecture.

We use the architecture of LSTM illustrated in Fig. 3.2 for the proposed sentence embedding method.

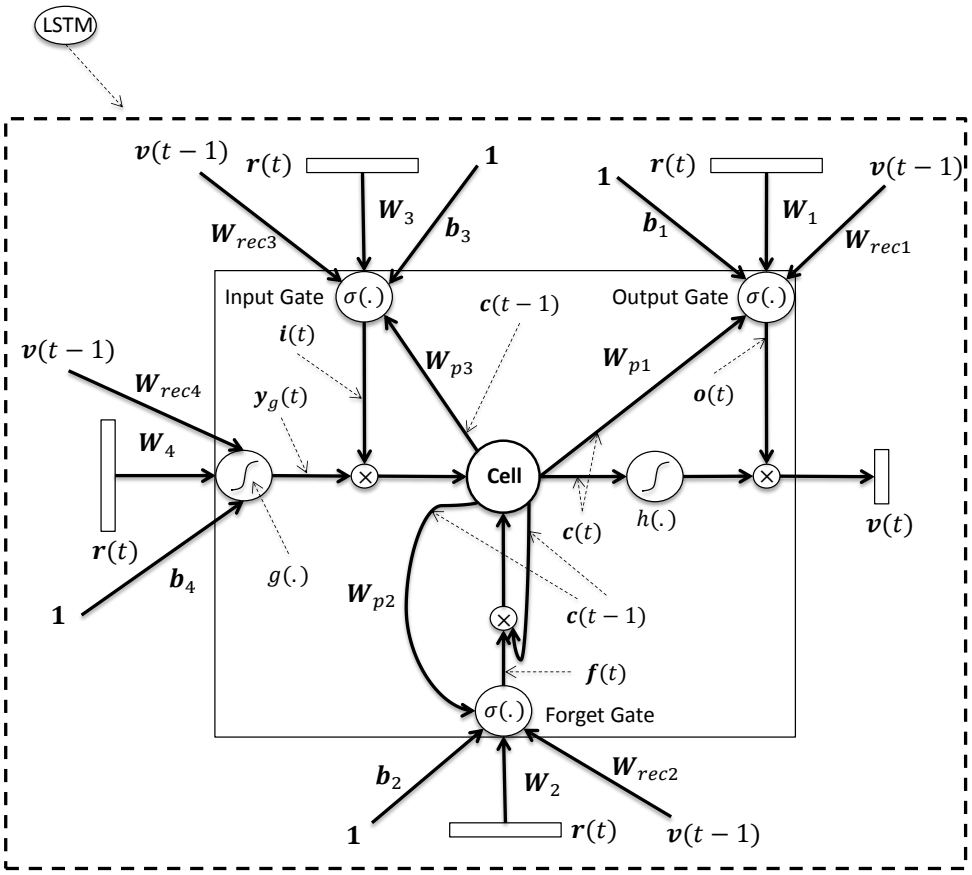


Figure 3.2: The basic LSTM architecture used for sentence embedding

In this figure, $\mathbf{i}(t)$, $\mathbf{f}(t)$,$\mathbf{o}(t)$,$\mathbf{c}(t)$ are input gate, forget gate, output gate and cell state vector respectively, $\mathbf{W}_{p1}$, $\mathbf{W}_{p2}$ and $\mathbf{W}_{p3}$ are peephole connections, $\mathbf{W}_i$, $\mathbf{W}_{reci}$ and $\mathbf{b}_i$, $i = 1,2,3,4$ are input connections, recurrent connections and bias values, respectively, $g(\cdot)$ and $h(\cdot)$ are $\tanh(\cdot)$ function and $\sigma(\cdot)$ is the sigmoid function. We use this architecture to find $\mathbf{y}$ for each word, then use the $\mathbf{y}(m)$ corresponding to the last word in the sentence as the semantic vector for the entire sentence.

Considering Fig. 3.2, the forward pass for LSTM-RNN model is as follows:

$$
\begin{aligned}
\mathbf{y}_g(t) &= g(\mathbf{W}_4\mathbf{l}(t) + \mathbf{W}_{\text{rec4}}\mathbf{y}(t-1) + \mathbf{b}_4) \\
\mathbf{i}(t) &= \sigma(\mathbf{W}_3\mathbf{l}(t) + \mathbf{W}_{\text{rec3}}\mathbf{y}(t-1) + \mathbf{W}_{p3}\mathbf{c}(t-1) + \mathbf{b}_3) \\
\mathbf{f}(t) &= \sigma(\mathbf{W}_2\mathbf{l}(t) + \mathbf{W}_{\text{rec2}}\mathbf{y}(t-1) + \mathbf{W}_{p2}\mathbf{c}(t-1) + \mathbf{b}_2) \\
\mathbf{c}(t) &= \mathbf{f}(t) \circ \mathbf{c}(t-1) + \mathbf{i}(t) \circ \mathbf{y}_g(t) \\
\mathbf{o}(t) &= \sigma(\mathbf{W}_1\mathbf{l}(t) + \mathbf{W}_{\text{rec1}}\mathbf{y}(t-1) + \mathbf{W}_{p1}\mathbf{c}(t) + \mathbf{b}_1) \\
\mathbf{y}(t) &= \mathbf{o}(t) \circ h(\mathbf{c}(t)),
\end{aligned}
\tag{3.2}
$$

where $\circ$ denotes Hadamard (element-wise) product. A diagram of the proposed model with more details is presented in section 3.12.

3.4 Learning Method

To learn a good semantic representation of the input sentence, our objective is *to make the embedding vectors for sentences of similar meaning as close as possible, and meanwhile, to make sentences of different meanings as far apart as possible*. This is challenging in practice since it is hard to collect a large amount of manually labelled data that give the semantic similarity signal between different sentences. Nevertheless, the widely used commercial web search engine is able to log massive amount of data with some limited user feedback signals. For example, given a particular query, the click-through information about the user-clicked document among many candidates is usually recorded and can be used as a weak (binary) supervision signal to indicate the semantic similarity between two sentences (on the query side and the document side). In this section, we explain how to leverage such a weak supervision signal to learn a sentence embedding vector that achieves the aforementioned training objective. Please also note that above objective to make sentences with similar meaning as close as possible is similar to machine translation tasks where two sentences belong to two different languages with similar meanings and we want to make their semantic representation as close as possible.

We now describe how to train the model to achieve the above objective using the click-through data logged by a commercial search engine. For a complete description of the click-through data please refer to section 2 in [41]. To begin with, we adopt the cosine similarity between the semantic vectors of two sentences as a measure for their similarity

$$R(Q,D) = \frac{\mathbf{y}_Q(T_Q)^T \mathbf{y}_D(T_D)}{\|\mathbf{y}_Q(T_Q)\| \cdot \|\mathbf{y}_D(T_D)\|}, \tag{3.3}$$

where T_Q and T_D are the lengths of the sentence Q and sentence D, respectively. In the context of training over click-through data, we will use Q and D to denote "query" and "document", respectively. In Figure 3.3, we show the sentence embedding vectors corresponding to the query, $\mathbf{y}_Q(T_Q)$, and all the documents, $\{\mathbf{y}_{D^+}(T_{D^+}), \mathbf{y}_{D_1^-}(T_{D_1^-}), \ldots, \mathbf{y}_{D_n^-}(T_{D_n^-})\}$, where the subscript D^+ denotes the (clicked) positive sample among the documents, and the subscript D_j^- denotes the j-th (un-clicked) negative sample. All these embedding vectors are generated by feeding the sentences into the RNN or LSTM-RNN model described in Sec. 3.3 and take the $\mathbf{y}$ corresponding to the last word — see the blue box in Figure 3.1.

We want to maximize the likelihood of the clicked document given query, which can be formulated as the following optimization problem:

$$L(\Lambda) = \min_{\Lambda} \left\{ -\log \prod_{r=1}^{N} P(D_r^+|Q_r) \right\} = \min_{\Lambda} \sum_{r=1}^{N} l_r(\Lambda), \tag{3.4}$$

where Λ denotes the collection of the model parameters; in regular RNN case, it includes $\mathbf{W}_{rec}$ and $\mathbf{W}$ in Figure 3.1, and in LSTM-RNN case, it includes $\mathbf{W}_1$, $\mathbf{W}_2$, $\mathbf{W}_3$, $\mathbf{W}_4$, $\mathbf{W}_{rec1}$, $\mathbf{W}_{rec2}$, $\mathbf{W}_{rec3}$, $\mathbf{W}_{rec4}$, $\mathbf{W}_{p1}$, $\mathbf{W}_{p2}$, $\mathbf{W}_{p3}$, $\mathbf{b}_1$, $\mathbf{b}_2$, $\mathbf{b}_3$ and $\mathbf{b}_4$ in Figure 3.2. D_r^+ is the clicked document for r-th query, $P(D_r^+|Q_r)$ is the probability

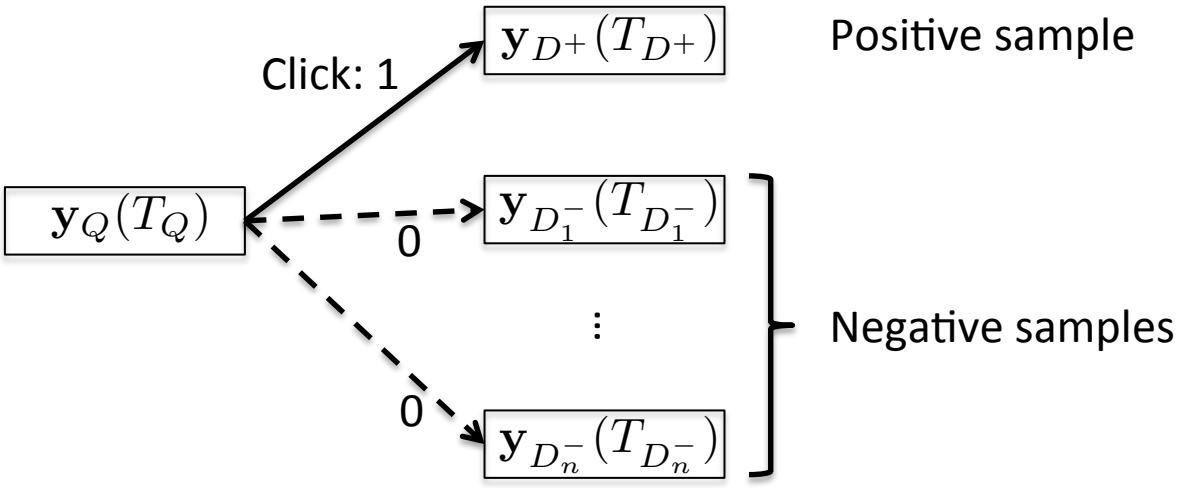


Figure 3.3: The click-through signal can be used as a (binary) indication of the semantic similarity between the sentence on the query side and the sentence on the document side. The negative samples are randomly sampled from the training data.

of clicked document given the r-th query, N is number of query / clicked-document pairs in the corpus and

$$\begin{aligned} l_r(\Lambda) &= -\log\left(\frac{e^{\gamma R(Q_r,D_r^+)}}{e^{\gamma R(Q_r,D_r^+)}+\sum_{i=j}^{n} e^{\gamma R(Q_r,D_{r,j}^-)}}\right) \\ &= \log\left(1+\sum_{j=1}^{n} e^{-\gamma\cdot\Delta_{r,j}}\right), \end{aligned} \tag{3.5}$$

where $\Delta_{r,j} = R(Q_r,D_r^+) - R(Q_r,D_{r,j}^-)$, $R(\cdot,\cdot)$ was defined earlier in (3.3), $D_{r,j}^-$ is the j-th negative candidate document for r-th query and n denotes the number of negative samples used during training.

The expression in (3.5) is a logistic loss over $\Delta_{r,j}$. It upper-bounds the pairwise accuracy, i.e., the 0 - 1 loss. Since the similarity measure is the cosine function, $\Delta_{r,j} \in [-2,2]$. To have a larger range for $\Delta_{r,j}$, we use γ for scaling. It helps to penalize the prediction error more. Its value is set empirically by experiments on a held out dataset.

To train the RNN and LSTM-RNN, we use Back Propagation Through Time (BPTT). The update equations for parameter Λ at epoch k are as follows:

$$\begin{aligned} \triangle\Lambda_k &= \Lambda_k - \Lambda_{k-1} \\ \triangle\Lambda_k &= \mu_{k-1}\triangle\Lambda_{k-1} - \varepsilon_{k-1}\nabla L(\Lambda_{k-1}+\mu_{k-1}\triangle\Lambda_{k-1}), \end{aligned} \tag{3.6}$$

where $\nabla L(\cdot)$ is the gradient of the cost function in (3.4), ε is the learning rate and μ_k is a momentum parameter determined by the scheduling scheme used for training. Above equations are equivalent to Nesterov method in [93]. To see why, please refer to appendix A.1 of [115] where Nesterov method is derived as a

momentum method. The gradient of the cost function, $\nabla L(\Lambda)$, is

$$\nabla L(\Lambda) = \underbrace{-\sum_{r=1}^{N}\sum_{j=1}^{n}\sum_{\tau=0}^{T}\alpha_{r,j}\frac{\partial \Delta_{r,j,\tau}}{\partial \Lambda}}_{\text{one large update}}, \tag{3.7}$$

where T is the number of time steps that we unfold the network over time and

$$\alpha_{r,j} = \frac{-\gamma e^{-\gamma \Delta_{r,j}}}{1+\sum_{j=1}^{n} e^{-\gamma \Delta_{r,j}}}. \tag{3.8}$$

$\frac{\partial \Delta_{r,j,\tau}}{\partial \Lambda}$ in (3.7) and error signals for different parameters of RNN and LSTM-RNN that are necessary for training are presented in section 3.6. Full derivation of gradients in both models is presented in section 3.9.

To accelerate training by parallelization, we use mini-batch training and one large update instead of incremental updates during back propagation through time. To resolve the gradient explosion problem we use gradient re-normalization method described in [85, 98]. To accelerate the convergence, we use Nesterov method [93] and found it effective in training both RNN and LSTM-RNN for sentence embedding.

We have used a simple yet effective scheduling for μ_k for both RNN and LSTM-RNN models, in the first and last 2% of all parameter updates $\mu_k = 0.9$ and for the other 96% of all parameter updates $\mu_k = 0.995$. We have used a fixed step size for training RNN and a fixed step size for training LSTM-RNN.

A summary of training method for LSTM-RNN is presented in Algorithm 1.

3.5 Analysis of the Sentence Embedding Process and Performance Evaluation

To understand how the LSTM-RNN performs sentence embedding, we use visualization tools to analyze the semantic vectors generated by our model. We would like to answer the following questions: (i) How are word dependencies and context information captured? (ii) How does LSTM-RNN attenuate unimportant information and detect critical information from the input sentence? Or, how are the keywords embedded into the semantic vector? (iii) How are the global topics identified by LSTM-RNN?

To answer these questions, we train the RNN with and without LSTM cells on the click-through dataset which are logged by a commercial web search engine. The training method has been described in Sec. 3.4. Description of the corpus is as follows. The training set includes 200,000 positive query / document pairs where only the clicked signal is used as a weak supervision for training LSTM. The relevance judgement set (test set) is constructed as follows. First, the queries are sampled from a year of search engine logs. Adult, spam, and bot queries are all removed. Queries are de-duped so that only unique queries remain. To reflex a natural query distribution, we do not try to control the quality of these queries. For example, in our query sets, there are around 20% misspelled queries, and around 20% navigational queries and 10% transactional queries, etc. Second, for each query, we collect Web documents to be judged by issuing the query to several popular search engines (e.g., Google, Bing) and fetching top-10 retrieval results from each. Finally, the query-document pairs are judged by a group of well-trained assessors. In this study all the queries are

Algorithm 1 Training LSTM-RNN for Sentence Embedding

Inputs: Fixed step size "ε", Scheduling for "μ", Gradient clip threshold "th_G", Maximum number of Epochs "$nEpoch$", Total number of query / clicked-document pairs "N", Total number of un-clicked (negative) documents for a given query "n", Maximum sequence length for truncated BPTT "T".
Outputs: Two trained models, one in query side "Λ_Q", one in document side "Λ_D".
Initialization: Set all parameters in Λ_Q and Λ_D to small random numbers, $i = 0$, $k = 1$.
procedure LSTM-RNN(Λ_Q,Λ_D)
  while $i \leq nEpoch$ **do**
    for "first minibatch" $\rightarrow$ "last minibatch" **do**
      $r \leftarrow 1$
      while $r \leq N$ **do**
        for $j = 1 \rightarrow n$ **do**
          Compute $\alpha_{r,j}$ ▷ use (3.8)
          Compute $\sum_{\tau=0}^{T} \alpha_{r,j} \frac{\partial \Delta_{r,j,\tau}}{\partial \Lambda_{k,Q}}$
          ▷ use (3.14) to (4.54) in section 3.6
          Compute $\sum_{\tau=0}^{T} \alpha_{r,j} \frac{\partial \Delta_{r,j,\tau}}{\partial \Lambda_{k,D}}$
          ▷ use (3.14) to (4.54) in section 3.6
          sum above terms for Q and D over j
        end for
        sum above terms for Q and D over r
        $r \leftarrow r+1$
      end while
      Compute $\nabla L(\Lambda_{k,Q})$ ▷ use (3.7)
      Compute $\nabla L(\Lambda_{k,D})$ ▷ use (3.7)
      if $\|\nabla L(\Lambda_{k,Q})\| > th_G$ **then**
        $\nabla L(\Lambda_{k,Q}) \leftarrow th_G \cdot \frac{\nabla L(\Lambda_{k,Q})}{\|\nabla L(\Lambda_{k,Q})\|}$
      end if
      if $\|\nabla L(\Lambda_{k,D})\| > th_G$ **then**
        $\nabla L(\Lambda_{k,D}) \leftarrow th_G \cdot \frac{\nabla L(\Lambda_{k,D})}{\|\nabla L(\Lambda_{k,D})\|}$
      end if
      Compute $\triangle \Lambda_{k,Q}$ ▷ use (4.13)
      Compute $\triangle \Lambda_{k,D}$ ▷ use (4.13)
      Update: $\Lambda_{k,Q} \leftarrow \triangle \Lambda_{k,Q} + \Lambda_{k-1,Q}$
      Update: $\Lambda_{k,D} \leftarrow \triangle \Lambda_{k,D} + \Lambda_{k-1,D}$
      $k \leftarrow k+1$
    end for
    $i \leftarrow i+1$
  end while
end procedure

preprocessed as follows. The text is white-space tokenized and lower-cased, numbers are retained, and no stemming/inflection treatment is performed. Unless stated otherwise, in the experiments we used 4 negative samples, i.e., $n = 4$ in Fig. 3.3.

We now proceed to perform a comprehensive analysis by visualizing the trained RNN and LSTM-RNN models. In particular, we will visualize the on-and-off behaviors of the input gates, output gates, cell states, and the semantic vectors in LSTM-RNN model, which reveals how the model extracts useful information from the input sentence and embeds it properly into the semantic vector according to the topic information.

Although giving the full learning formula for all the model parameters in the previous section, we will remove the peephole connections and the forget gate from the LSTM-RNN model in the current task. This is because the length of each sequence, i.e., the number of words in a query or a document, is known in advance, and we set the state of each cell to zero in the beginning of a new sequence. Therefore, forget gates are not a great help here. Also, as long as the order of words is kept, the precise timing in the sequence is not of great concern. Therefore, peephole connections are not that important as well. Removing peephole connections and forget gate will also reduce the amount of training time, since a smaller number

of parameters need to be learned.

3.5.1 Analysis

In this section we would like to examine how the information in the input sentence is sequentially extracted and embedded into the semantic vector over time by the LSTM-RNN model.

Attenuating Unimportant Information

First, we examine the evolution of the semantic vector and how unimportant words are attenuated. Specifically, we feed the following input sentences from the test dataset into the trained LSTM-RNN model:

- Query: "*hotels in shanghai*"
- Document: "*shanghai hotels accommodation hotel in shanghai discount and reservation*"

Activations of input gate, output gate, cell state and the embedding vector for each cell for query and document are shown in Fig. 3.4 and Fig. 3.5, respectively. The vertical axis is the cell index from 1 to 32, and the horizontal axis is the word index from 1 to 10 numbered from left to right in a sequence of words and color codes show activation values. From Figs.3.4–3.5, we make the following observations:

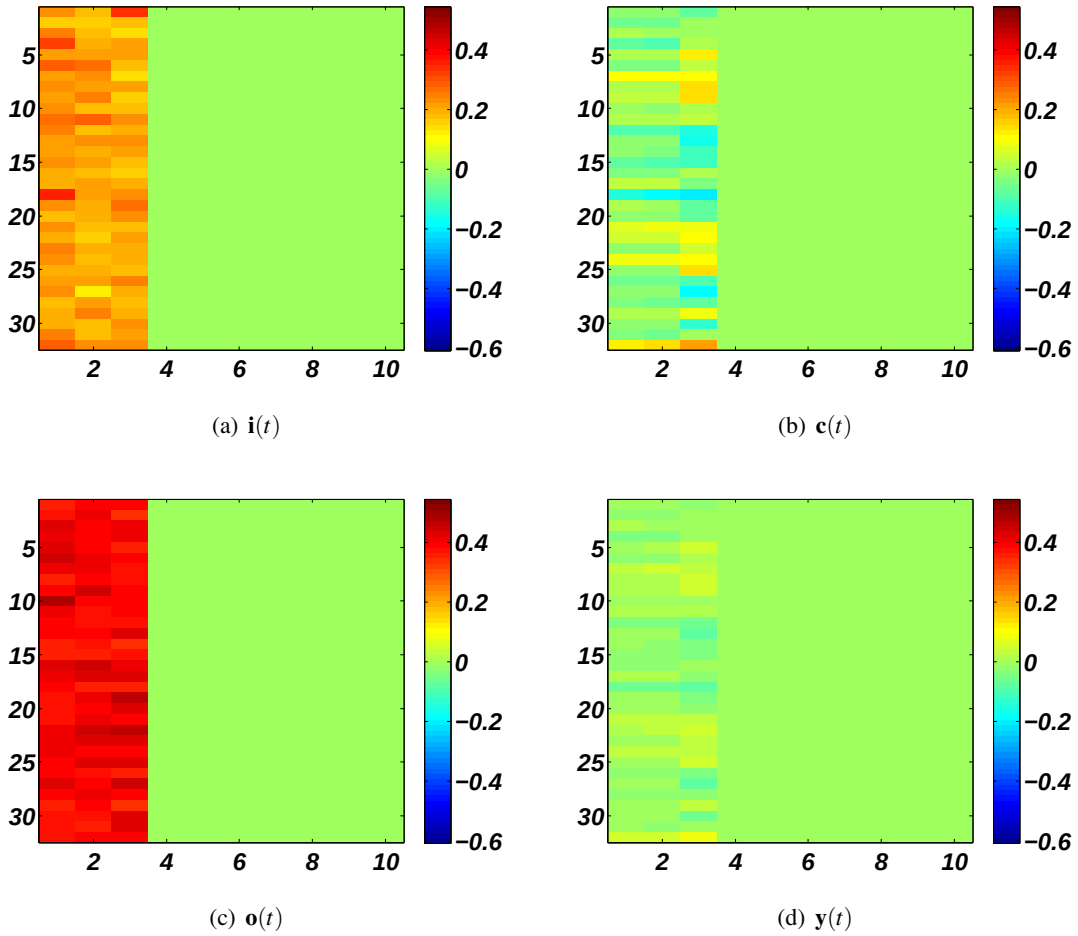
- Semantic representation $\mathbf{y}(t)$ and cell states $\mathbf{c}(t)$ are evolving over time. Valuable context information is gradually absorbed into $\mathbf{c}(t)$ and $\mathbf{y}(t)$, so that the information in these two vectors becomes richer over time, and the semantic information of the entire input sentence is embedded into vector $\mathbf{y}(t)$, which is obtained by applying output gates to the cell states $\mathbf{c}(t)$.
- The input gates evolve in such a way that it attenuates the unimportant information and detects the important information from the input sentence. For example, in Fig. 3.5(a), most of the input gate values corresponding to word 3, word 7 and word 9 have very small values (light green-yellow color)[1], which corresponds to the words "*accommodation*", "*discount*" and "*reservation*", respectively, in the document sentence. Interestingly, input gates reduce the effect of these three words in the final semantic representation, $\mathbf{y}(t)$, such that the semantic similarity between sentences from query and document sides are not affected by these words.

Keywords Extraction

In this section, we show how the trained LSTM-RNN extracts the important information, i.e., keywords, from the input sentences. To this end, we backtrack semantic representations, $\mathbf{y}(t)$, over time. We focus on the 10 most active cells in final semantic representation. Whenever there is a large enough change in cell activation value ($\mathbf{y}(t)$), we assume an important keyword has been detected by the model. We illustrate the result using the above example ("*hotels in shanghai*"). The evolution of the 10 most active cells activation, $\mathbf{y}(t)$, over time are shown in Fig. 3.6 for the query and the document sentences.[2]From Fig. 3.6, we also

[1]If this is not clearly visible, please refer to Fig. 3.8. We have adjusted color bar for all figures to have the same range, for this reason the structure might not be clearly visible. More visualization examples could also be found in section 3.10

[2]Likewise, the vertical axis is the cell index and horizontal axis is the word index in the sentence.



(a) $\mathbf{i}(t)$ (b) $\mathbf{c}(t)$

(c) $\mathbf{o}(t)$ (d) $\mathbf{y}(t)$

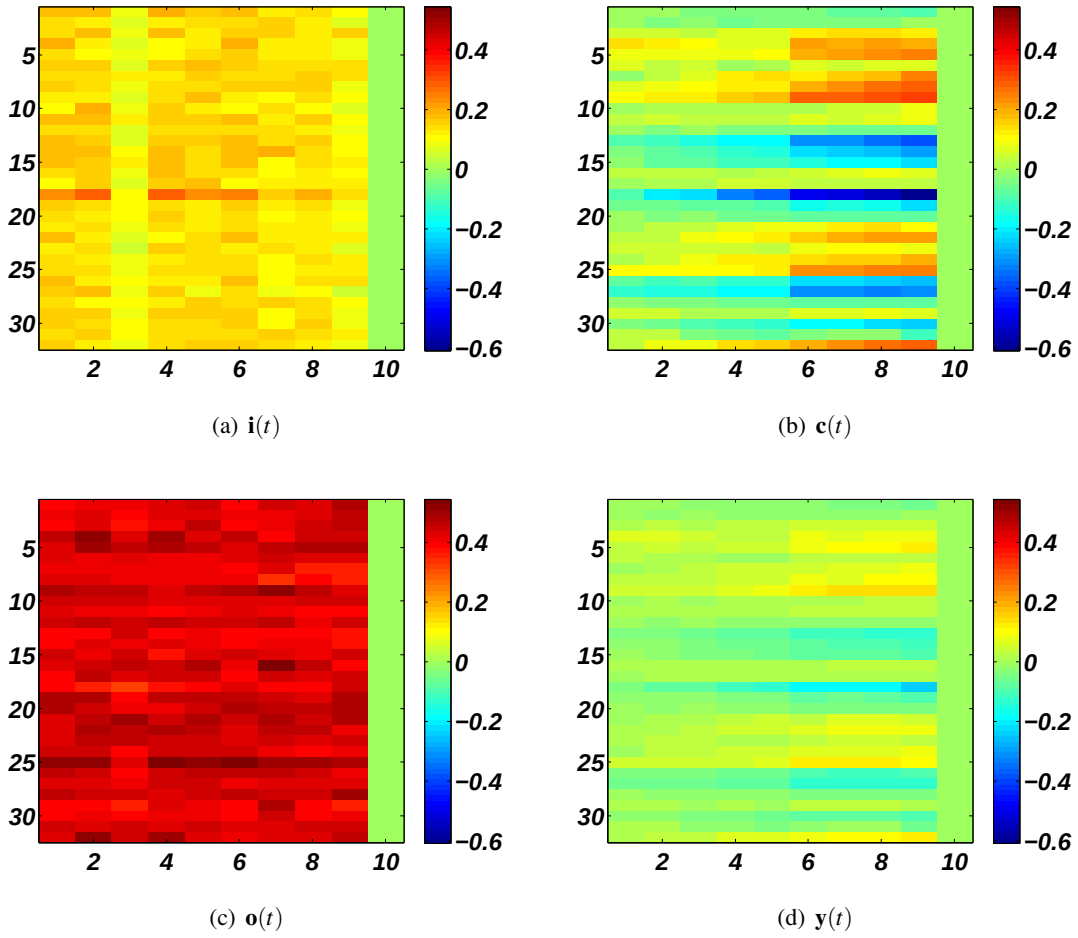
Figure 3.4: Query: "*hotels in shanghai*". Since the sentence ends at the third word, all the values to the right of it are zero (green color).

observe that different words activate different cells. In Tables 3.1–3.2, we show the number of cells each word activates.[3] We used Bidirectional LSTM-RNN to get the results of these tables where in the first row, LSTM-RNN reads sentences from left to right and in the second row it reads sentences from right to left. In these tables we labelled a word as a keyword if more than 40% of top 10 active cells in both directions declare it as keyword. The boldface numbers in the table show that the number of cells assigned to that word is more than 4, i.e., 40% of top 10 active cells. From the tables, we observe that the keywords activate more cells than the unimportant words, meaning that they are selectively embedded into the semantic vector.

Topic Allocation

Now, we further show that the trained LSTM-RNN model not only detects the keywords, but also allocates them properly to different cells according to the topics they belong to. To do this, we go through the test

[3]Note that before presenting the first word of the sequence, activation values are initially zero so that there is always a considerable change in the cell states after presenting the first word. For this reason, we have not indicated the number of cells detecting the first word as a keyword. Moreover, another keyword extraction example can be found in section 3.10.2.



(a) $\mathbf{i}(t)$ (b) $\mathbf{c}(t)$

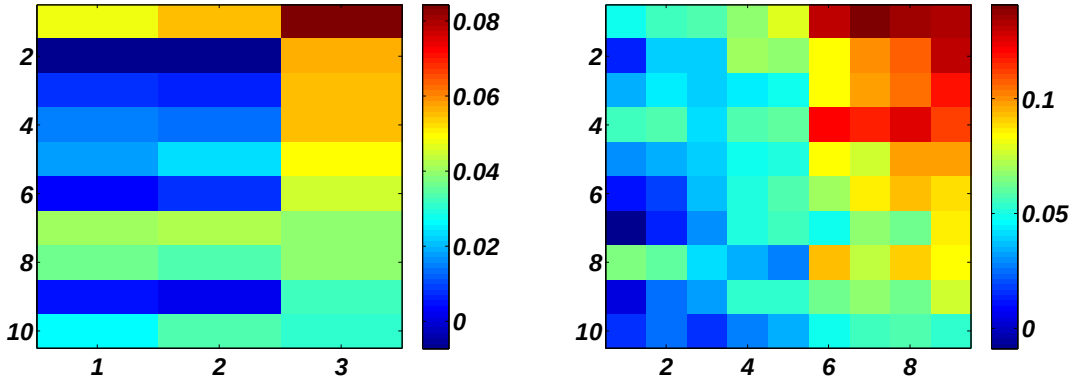
(c) $\mathbf{o}(t)$ (d) $\mathbf{y}(t)$

Figure 3.5: Document: "*shanghai hotels accommodation hotel in shanghai discount and reservation*". Since the sentence ends at the ninth word, all the values to the right of it are zero (green color).

Table 3.1: Key words for query: "*hotels in shanghai*"

Query	*hotels*	*in*	*shanghai*
Number of assigned cells out of 10 Left to Right	-	0	**7**
Number of assigned cells out of 10 Right to Left	**6**	0	-

dataset using the trained LSTM-RNN model and search for the keywords that are detected by a specific cell. For simplicity, we use the following simple approach: for each given query we look into the keywords that are extracted by the 5 most active cells of LSTM-RNN and list them in Table 3.3. Interestingly, each cell collects keywords of a specific topic. For example, cell 26 in Table 3.3 extracts keywords related to the topic "food" and cells 2 and 6 mainly focus on the keywords related to the topic "health".



(a) $\mathbf{y}(t)$ top 10 for query

(b) $\mathbf{y}(t)$ top 10 for document

Figure 3.6: Activation values, $\mathbf{y}(t)$, of 10 most active cells for Query: "*hotels in shanghai*" and Document: "*shanghai hotels accommodation hotel in shanghai discount and reservation*"

Table 3.2: Key words for document: "*shanghai hotels accommodation hotel in shanghai discount and reservation*"

	shanghai	*hotels*	*accommodation*	*hotel*	*in*	*shanghai*	*discount*	*and*	*reservation*
Number of assigned cells out of 10 Left to Right	-	**4**	3	**8**	1	**8**	**5**	3	**4**
Number of assigned cells out of 10 Right to Left	**4**	**6**	**5**	**4**	**5**	1	**7**	**5**	-

3.5.2 Performance Evaluation

Web Document Retrieval Task

In this section, we apply the proposed sentence embedding method to an important web document retrieval task for a commercial web search engine. Specifically, the RNN models (with and without LSTM cells) embed the sentences from the query and the document sides into their corresponding semantic vectors, and then compute the cosine similarity between these vectors to measure the semantic similarity between the query and candidate documents.

Experimental results for this task are shown in Table 3.4 using the standard metric mean Normalized Discounted Cumulative Gain (NDCG) [70] (the higher the better) for evaluating the ranking performance of the RNN and LSTM-RNN on a standalone human-rated test dataset. We also trained several strong baselines, such as DSSM [65] and CLSM [108], on the same training dataset and evaluated their performance on the same task. For fair comparison, our proposed RNN and LSTM-RNN models are trained with the same number of parameters as the DSSM and CLSM models (14.4M parameters). Besides, we also include in Table 3.4 two well-known information retrieval (IR) models, BM25 and PLSA, for the sake of benchmarking. The BM25 model uses the bag-of-words representation for queries and documents, which is a state-of-the-art document ranking model based on term matching, widely used as a baseline in IR society.

Table 3.3: Keywords assigned to each cell of LSTM-RNN for different queries of two topics, "food" represented by green color and "health" represented by red color

Query	cell 1	cell 2	cell 3	cell 4	cell 5	cell 6	cell 7	cell 8	cell 9	cell 10	cell 11	cell 12	cell 13	cell 14	cell 15	cell 16
al yo yo sauce					yo			sauce			sauce					
atkins diet lasagna								diet								
blender recipes																
cake bakery edinburgh										bakery						
canning corn beef hash					beef, hash											
torre de pizza																
famous desserts							desserts									
fried chicken			chicken				chicken									
smoked turkey recipes																
italian sausage hoagies								sausage								
do you get allergy		allergy														
much pain will after total knee replacement	pain					pain, knee										
how to make whiter teeth													make, teeth		to	
illini community hospital		community, hospital						hospital		community						
implant infection		infection				infection										
introductory psychology		psychology				psychology										
narcotics during pregnancy side effects		pregnancy				pregnancy,effects, during							during			
fight sinus infections						infections										
health insurance high blood pressure		insurance				blood		high, blood								
all antidepressant medications		antidepressant, medications														

Query	cell 17	cell 18	cell 19	cell 20	cell 21	cell 22	cell 23	cell 24	cell 25	cell 26	cell 27	cell 28	cell 29	cell 30	cell 31	cell 32
al yo yo sauce																
atkins diet lasagna							diet							diet		
blender recipes										recipes						
cake bakery edinburgh				bakery						bakery						
canning corn beef hash										corn, beef						
torre de pizza										pizza			pizza			
famous desserts																
fried chicken										chicken						
smoked turkey recipes				turkey						recipes						
italian sausage hoagies	hoagies					sausage				sausage						
do you get allergy																
much pain will after total knee replacement	knee						replacement									
how to make whiter teeth										whiter						
illini community hospital						hospital								hospital		
implant infection									infection							
introductory psychology												psychology				
narcotics during pregnancy side effects																
fight sinus infections	sinus, infections								infections							
health insurance high blood pressure							high, pressure							insurance,high		
all antidepressant medications									antidepressant					medications		

PLSA (Probabilistic Latent Semantic Analysis) is a topic model proposed in [63], which is trained using the Maximum A Posterior estimation [42] on the documents side from the same training dataset. We experimented with a varying number of topics from 100 to 500 for PLSA, which gives similar performance, and we report in Table 3.4 the results of using 500 topics. Results for a language model based method, uni-gram language model (ULM) with Dirichlet smoothing, are also presented in the table.

To compare the performance of the proposed method with general sentence embedding methods in document retrieval task, we also performed experiments using two general sentence embedding methods.

1. In the first experiment, we used the method proposed in [77] that generates embedding vectors known as Paragraph Vectors. It is also known as doc2vec. It maps each word to a vector and then uses the vectors representing all words inside a context window to predict the vector representation of the next word. The main idea in this method is to use an additional paragraph token from previous sentences in the document inside the context window. This paragraph token is mapped to vector space using a different matrix from the one used to map the words. A primary version of this method is known as word2vec proposed in [88]. The only difference is that word2vec does not include the paragraph token.

   To use doc2vec on our dataset, we first trained doc2vec model on both train set (about 200,000 query-document pairs) and test set (about 900,000 query-document pairs). This gives us an embedding vector for every query and document in the dataset. We used the following parameters for training:

   - min-count=1 : minimum number of of words per sentence, sentences with words less than this

will be ignored. We set it to 1 to make sure we do not throw away anything.

- window=5 : fixed window size explained in [77]. We used different window sizes, it resulted in about just 0.4% difference in final NDCG values.
- size=100 : feature vector dimension. We used 400 as well but did not get significantly different NDCG values.
- sample=1e-4 : this is the down sampling ratio for the words that are repeated a lot in corpus.
- negative=5 : the number of noise words, i.e., words used for negative sampling as explained in [77].
- We used 30 epochs of training. We ran an experiment with 100 epochs but did not observe much difference in the results.
- We used *gensim* [101] to perform experiments.

To make sure that a meaningful model is trained, we used the trained doc2vec model to find the most similar words to two sample words in our dataset, e.g., the words "pizza" and "infection". The resulting words and corresponding scores are presented in section 3.11. As it is observed from the resulting words, the trained model is a meaningful model and can recognise semantic similarity.

Doc2vec also assigns an embedding vector for each query and document in our test set. We used these embedding vectors to calculate the cosine similarity score between each query-document pair in the test set. We used these scores to calculate NDCG values reported in Table 3.4 for the Doc2Vec model.

Comparing the results of doc2vec model with our proposed method for document retrieval task shows that the proposed method in this chapter significantly outperforms doc2vec. One reason for this is that we have used a very general sentence embedding method, doc2vec, for document retrieval task. This experiment shows that it is not a good idea to use a general sentence embedding method and using a better task oriented cost function, like the one proposed in this chapter, is necessary.

2. In the second experiment, we used the Skip-Thought vectors proposed in [75]. During training, skip-thought method gets a tuple $(s(t-1), s(t), s(t+1))$ where it encodes the sentence $s(t)$ using one encoder, and tries to reconstruct the previous and next sentences, i.e., $s(t-1)$ and $s(t+1)$, using two separate decoders. The model uses RNNs with Gated Recurrent Unit (GRU) which is shown to perform as good as LSTM. In the paper, authors have emphasized that: "*Our model depends on having a training corpus of contiguous text*". Therefore, training it on our training set where we barely have more than one sentence in query or document title is not fair. However, since their model is trained on 11,038 books from BookCorpus dataset [135] which includes about 74 million sentences, we can use the trained model as an off-the-shelf sentence embedding method as authors have concluded in the conclusion of the paper.

   To do this we downloaded their trained models and word embeddings (its size was more than 2GB) available from "https://github.com/ryankiros/skip-thoughts". Then we encoded each query and its corresponding document title in our test set as vector.

We used the combine-skip sentence embedding method, a vector of size 4800×1, where it is concatenation of a uni-skip, i.e., a unidirectional encoder resulting in a 2400×1 vector, and a bi-skip, i.e., a bidirectional encoder resulting in a 1200×1 vector by forward encoder and another 1200×1 vector by backward encoder. The authors have reported their best results with the combine-skip encoder.

Using the 4800×1 embedding vectors for each query and document we calculated the scores and NDCG for the whole test set which are reported in Table 3.4.

The proposed method in this chapter is performing significantly better than the off-the-shelf skip-thought method for document retrieval task. Nevertheless, since we used skip-thought as an off-the-shelf sentence embedding method, its result is good. This result also confirms that learning embedding vectors using a model and cost function specifically designed for document retrieval task is necessary.

As shown in Table 3.4, the LSTM-RNN significantly outperforms all these models, and exceeds the best baseline model (CLSM) by 1.3% in NDCG@1 score, which is a statistically significant improvement. As we pointed out in Sec. 3.5.1, such an improvement comes from the LSTM-RNN's ability to embed the contextual and semantic information of the sentences into a finite dimension vector. In Table 3.4, we have also presented the results when different number of negative samples, n, is used. Generally, by increasing n we expect the performance to improve. This is because more negative samples results in a more accurate approximation of the partition function in (3.5). The results of using Bidirectional LSTM-RNN are also presented in Table 3.4. In this model, one LSTM-RNN reads queries and documents from left to right, and the other LSTM-RNN reads queries and documents from right to left. Then the embedding vectors from left to right and right to left LSTM-RNNs are concatenated to compute the cosine similarity score and NDCG values.

A comparison between the value of the cost function during training for LSTM-RNN and RNN on the click-through data is shown in Fig. 3.7. From this figure, we conclude that LSTM-RNN is optimizing the cost function in (3.4) more effectively. Please note that all parameters of both models are initialized randomly.

3.6 Expressions for the Gradients

In this section we present the final gradient expressions that are necessary to use for training the proposed models. Full derivations of these gradients are presented in section 3.9.

3.6.1 RNN

For the recurrent parameters, $\Lambda = \mathbf{W}_{rec}$ (we have ommitted r subscript for simplicity)

$$\begin{aligned}\frac{\partial \Delta_{j,\tau}}{\partial \mathbf{W}_{rec}} &= [\delta_{y_Q}^{D^+}(t-\tau)\mathbf{y}_Q^T(t-\tau-1)+ \\ &\delta_{y_D}^{D^+}(t-\tau)\mathbf{y}_{D^+}^T(t-\tau-1)] - [\delta_{y_Q}^{D_j^-}(t-\tau)\mathbf{y}_Q^T(t-\tau-1) \\ &+ \delta_{y_D}^{D_j^-}(t-\tau)\mathbf{y}_{D_j^-}^T(t-\tau-1)],\end{aligned} \tag{3.9}$$

Table 3.4: Comparisons of NDCG performance measures (the higher the better) of proposed models and a series of baseline models, where *nhid* refers to the number of hidden units, *ncell* refers to number of cells, *win* refers to window size, and *n* is the number of negative samples which is set to 4 unless otherwise stated. Unless stated otherwise, the RNN and LSTM-RNN models are chosen to have the same number of model parameters as the DSSM and CLSM models: 14.4M, where $1\text{M} = 10^6$. The boldface numbers are the best results.

Model	NDCG @1	NDCG @3	NDCG @10
Skip-Thought off-the-shelf	26.9%	29.7%	36.2%
Doc2Vec	29.1%	31.8%	38.4%
ULM	30.4%	32.7%	38.5%
BM25	30.5%	32.8%	38.8%
PLSA (T=500)	30.8%	33.7%	40.2%
DSSM (nhid = 288/96) 2 Layers	31.0%	34.4%	41.7%
CLSM (nhid = 288/96, win=1) 2 Layers, 14.4 M parameters	31.8%	35.1%	42.6%
CLSM (nhid = 288/96, win=3) 2 Layers, 43.2 M parameters	32.1%	35.2%	42.7%
CLSM (nhid = 288/96, win=5) 2 Layers, 72 M parameters	32.0%	35.2%	42.6%
RNN (nhid = 288) 1 Layer	31.7%	35.0%	42.3%
LSTM-RNN (ncell = 32) 1 Layer, 4.8 M parameters	31.9%	35.5%	42.7%
LSTM-RNN (ncell = 64) 1 Layer, 9.6 M parameters	32.9%	36.3%	43.4%
LSTM-RNN (ncell = 96) 1 Layer, n = 2	32.6%	36.0%	43.4%
LSTM-RNN (ncell = 96) 1 Layer, n = 4	33.1%	36.5%	43.6%
LSTM-RNN (ncell = 96) 1 Layer, n = 6	33.1%	36.6%	43.6%
LSTM-RNN (ncell = 96) 1 Layer, n = 8	**33.1%**	**36.4%**	**43.7%**
Bidirectional LSTM-RNN (ncell = 96), 1 Layer	**33.2%**	**36.6%**	**43.6%**

where D_j^- means j-th candidate document that is not clicked and

$$\begin{aligned}\delta_{\mathbf{y}_Q}(t-\tau-1) &= (1-\mathbf{y}_Q(t-\tau-1))\circ \\ &(1+\mathbf{y}_Q(t-\tau-1))\circ \mathbf{W}_{rec}^T \delta_{\mathbf{y}_Q}(t-\tau),\end{aligned} \tag{3.10}$$

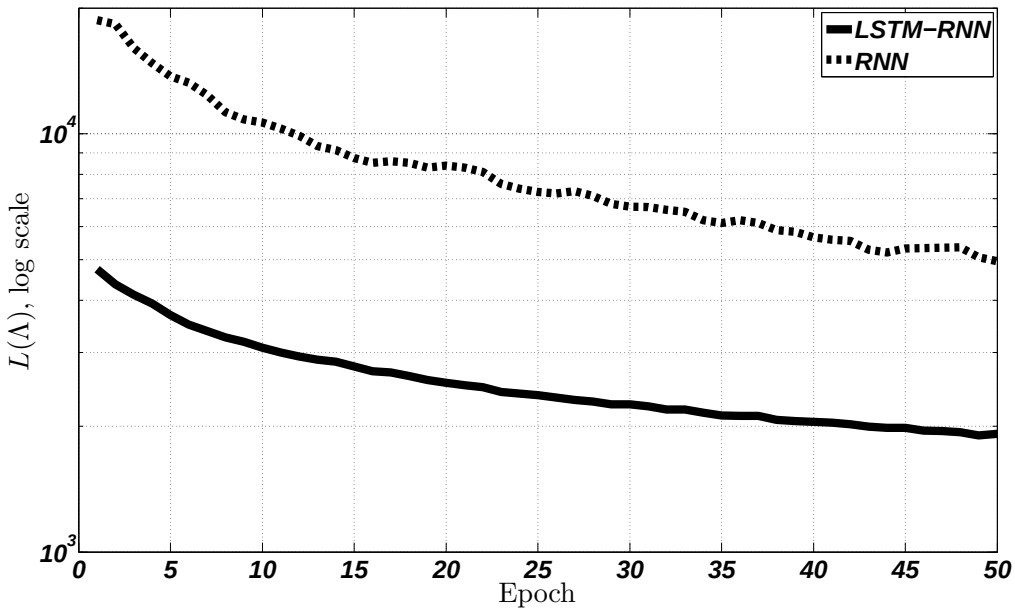


Figure 3.7: LSTM-RNN compared to RNN during training: The vertical axis is logarithmic scale of the training cost, $L(\Lambda)$, in (3.4). Horizontal axis is the number of epochs during training.

and the same as (3.10) for $\delta_{\mathbf{y}_D}(t-\tau-1)$ with D subscript for document side model. Please also note that

$$\begin{aligned}
&\delta_{\mathbf{y}_Q}(T_Q) = (1-\mathbf{y}_Q(T_Q)) \circ (1+\mathbf{y}_Q(T_Q)) \circ \\
&(b.c.\mathbf{y}_D(T_D) - a.b^3.c.\mathbf{y}_Q(T_Q)), \\
&\delta_{\mathbf{y}_D}(T_D) = (1-\mathbf{y}_D(T_D)) \circ (1+\mathbf{y}_D(T_D)) \circ \\
&(b.c.\mathbf{y}_Q(T_Q) - a.b.c^3.\mathbf{y}_D(T_D)),
\end{aligned} \tag{3.11}$$

where

$$\begin{aligned}
&a = \mathbf{y}_Q(t=T_Q)^T \mathbf{y}_D(t=T_D) \\
&b = \frac{1}{\|\mathbf{y}_Q(t=T_Q)\|},\ c = \frac{1}{\|\mathbf{y}_D(t=T_D)\|}.
\end{aligned} \tag{3.12}$$

For the input parameters, $\Lambda = \mathbf{W}$

$$\begin{aligned}
&\frac{\partial \Delta_{j,\tau}}{\partial \mathbf{W}} = [\delta_{y_Q}^{D^+}(t-\tau)\mathbf{l}_Q^T(t-\tau) + \\
&\delta_{y_D}^{D^+}(t-\tau)\mathbf{l}_{D^+}^T(t-\tau)] - \\
&[\delta_{y_Q}^{D_j^-}(t-\tau)\mathbf{l}_Q^T(t-\tau) + \delta_{y_D}^{D_j^-}(t-\tau)\mathbf{l}_{D_j^-}^T(t-\tau)].
\end{aligned} \tag{3.13}$$

A full derivation of BPTT for RNN is presented in section 3.9.

3.6.2 LSTM-RNN

Starting with the cost function in (3.4), we use the Nesterov method described in (4.13) to update LSTM-RNN model parameters. Here, Λ is one of the weight matrices or bias vectors $\{\mathbf{W}_1, \mathbf{W}_2, \mathbf{W}_3, \mathbf{W}_4, \mathbf{W}_{rec1}, \mathbf{W}_{rec2}, \mathbf{W}_{rec3}, \mathbf{W}_{rec4}, \mathbf{W}_{p1}, \mathbf{W}_{p2}, \mathbf{W}_{p3}, \mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3, \mathbf{b}_4\}$ in the LSTM-RNN architecture. The general format of the gradient of the cost function, $\nabla L(\Lambda)$, is the same as (3.7). By definition of $\Delta_{r,j}$, we have

$$\frac{\partial \Delta_{r,j}}{\partial \Lambda} = \frac{\partial R(Q_r, D_r^+)}{\partial \Lambda} - \frac{\partial R(Q_r, D_{r,j})}{\partial \Lambda}. \tag{3.14}$$

We omit r and j subscripts for simplicity and present $\frac{\partial R(Q,D)}{\partial \Lambda}$ for different parameters of each cell of LSTM-RNN in the following subsections. This will complete the process of calculating $\nabla L(\Lambda)$ in (3.7) and then we can use (4.13) to update LSTM-RNN model parameters. In the subsequent subsections vectors $\mathbf{v}_Q$ and $\mathbf{v}_D$ are defined as

$$\begin{aligned} \mathbf{v}_Q &= (b.c.\mathbf{y}_D(t = T_D) - a.b^3.c.\mathbf{y}_Q(t = T_Q)) \\ \mathbf{v}_D &= (b.c.\mathbf{y}_Q(t = T_Q) - a.b.c^3.\mathbf{y}_D(t = T_D)), \end{aligned} \tag{3.15}$$

where a, b and c are defined in (3.12). Full derivation of truncated BPTT for LSTM-RNN model is presented in section 3.9.

Output Gate

For recurrent connections we have

$$\frac{\partial R(Q,D)}{\partial \mathbf{W}_{rec1}} = \delta_{y_Q}^{rec1}(t).\mathbf{y}_Q(t-1)^T + \delta_{y_D}^{rec1}(t).\mathbf{y}_D(t-1)^T, \tag{3.16}$$

where

$$\delta_{y_Q}^{rec1}(t) = \mathbf{o}_Q(t) \circ (1 - \mathbf{o}_Q(t)) \circ h(\mathbf{c}_Q(t)) \circ \mathbf{v}_Q(t), \tag{3.17}$$

and the same as (4.27) for $\delta_{y_D}^{rec1}(t)$ with subscript D for document side model. For input connections, $\mathbf{W}_1$, and peephole connections, $\mathbf{W}_{p1}$, we will have

$$\frac{\partial R(Q,D)}{\partial \mathbf{W}_1} = \delta_{y_Q}^{rec1}(t).\mathbf{l}_Q(t)^T + \delta_{y_D}^{rec1}(t).\mathbf{l}_D(t)^T. \tag{3.18}$$

$$\frac{\partial R(Q,D)}{\partial \mathbf{W}_{p1}} = \delta_{y_Q}^{rec1}(t).\mathbf{c}_Q(t)^T + \delta_{y_D}^{rec1}(t).\mathbf{c}_D(t)^T. \tag{3.19}$$

The derivative for output gate bias values will be

$$\frac{\partial R(Q,D)}{\partial \mathbf{b}_1} = \delta_{y_Q}^{rec1}(t) + \delta_{y_D}^{rec1}(t). \tag{3.20}$$

Input Gate

For the recurrent connections we have

$$\frac{\partial R(Q,D)}{\partial \mathbf{W}_{rec3}} = diag(\delta_{y_Q}^{rec3}(t)).\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{W}_{rec3}} + diag(\delta_{y_D}^{rec3}(t)).\frac{\partial \mathbf{c}_D(t)}{\partial \mathbf{W}_{rec3}}, \tag{3.21}$$

where

$$\begin{aligned}
&\delta_{y_Q}^{rec3}(t) = (1-h(\mathbf{c}_Q(t))) \circ (1+h(\mathbf{c}_Q(t))) \circ \mathbf{o}_Q(t) \circ \mathbf{v}_Q(t) \\
&\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{W}_{rec3}} = diag(\mathbf{f}_Q(t)).\frac{\partial \mathbf{c}_Q(t-1)}{\partial \mathbf{W}_{rec3}} + \mathbf{b}_{i,Q}(t).\mathbf{y}_Q(t-1)^T \\
&\mathbf{b}_{i,Q}(t) = \mathbf{y}_{g,Q}(t) \circ \mathbf{i}_Q(t) \circ (1-\mathbf{i}_Q(t)).
\end{aligned} \tag{3.22}$$

In equation (4.31), $\delta_{y_D}^{rec3}(t)$ and $\frac{\partial \mathbf{c}_D(t)}{\partial \mathbf{W}_{rec3}}$ are the same as (4.32) with D subscript. For the input connections we will have the following:

$$\frac{\partial R(Q,D)}{\partial \mathbf{W}_3} = diag(\delta_{y_Q}^{rec3}(t)).\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{W}_3} + diag(\delta_{y_D}^{rec3}(t)).\frac{\partial \mathbf{c}_D(t)}{\partial \mathbf{W}_3}, \tag{3.23}$$

where

$$\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{W}_3} = diag(\mathbf{f}_Q(t)).\frac{\partial \mathbf{c}_Q(t-1)}{\partial \mathbf{W}_3} + \mathbf{b}_{i,Q}(t).\mathbf{x}_Q(t)^T. \tag{3.24}$$

For the peephole connections we will have

$$\frac{\partial R(Q,D)}{\partial \mathbf{W}_{p3}} = diag(\delta_{y_Q}^{rec3}(t)).\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{W}_{p3}} + diag(\delta_{y_D}^{rec3}(t)).\frac{\partial \mathbf{c}_D(t)}{\partial \mathbf{W}_{p3}}, \tag{3.25}$$

where

$$\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{W}_{p3}} = diag(\mathbf{f}_Q(t)).\frac{\partial \mathbf{c}_Q(t-1)}{\partial \mathbf{W}_{p3}} + \mathbf{b}_{i,Q}(t).\mathbf{c}_Q(t-1)^T. \tag{3.26}$$

For bias values, $\mathbf{b}_3$, we will have

$$\frac{\partial R(Q,D)}{\partial \mathbf{b}_3} = diag(\delta_{y_Q}^{rec3}(t)).\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{b}_3} + diag(\delta_{y_D}^{rec3}(t)).\frac{\partial \mathbf{c}_D(t)}{\partial \mathbf{b}_3}, \tag{3.27}$$

where

$$\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{b}_3} = diag(\mathbf{f}_Q(t)).\frac{\partial \mathbf{c}_Q(t-1)}{\partial \mathbf{b}_3} + \mathbf{b}_{i,Q}(t). \tag{3.28}$$

Forget Gate

For the recurrent connections we will have

$$\frac{\partial R(Q,D)}{\partial \mathbf{W}_{rec2}} = \\ diag(\delta_{y_Q}^{rec2}(t)).\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{W}_{rec2}} + diag(\delta_{y_D}^{rec2}(t)).\frac{\partial \mathbf{c}_D(t)}{\partial \mathbf{W}_{rec2}}, \tag{3.29}$$

where

$$\begin{aligned} &\delta_{y_Q}^{rec2}(t) = (1-h(\mathbf{c}_Q(t))) \circ (1+h(\mathbf{c}_Q(t))) \circ \mathbf{o}_Q(t) \circ \mathbf{v}_Q(t) \\ &\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{W}_{rec2}} = diag(\mathbf{f}_Q(t)).\frac{\partial \mathbf{c}_Q(t-1)}{\partial \mathbf{W}_{rec2}} + \mathbf{b}_{f,Q}(t).\mathbf{y}_Q(t-1)^T \\ &\mathbf{b}_{f,Q}(t) = \mathbf{c}_Q(t-1) \circ \mathbf{f}_Q(t) \circ (1-\mathbf{f}_Q(t)). \end{aligned} \tag{3.30}$$

For input connections to forget gate we will have

$$\frac{\partial R(Q,D)}{\partial \mathbf{W}_2} = \\ diag(\delta_{y_Q}^{rec2}(t)).\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{W}_2} + diag(\delta_{y_D}^{rec2}(t)).\frac{\partial \mathbf{c}_D(t)}{\partial \mathbf{W}_2}, \tag{3.31}$$

where

$$\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{W}_2} = diag(\mathbf{f}_Q(t)).\frac{\partial \mathbf{c}_Q(t-1)}{\partial \mathbf{W}_2} + \mathbf{b}_{f,Q}(t).\mathbf{x}_Q(t)^T. \tag{3.32}$$

For peephole connections we have

$$\frac{\partial R(Q,D)}{\partial \mathbf{W}_{p2}} = \\ diag(\delta_{y_Q}^{rec2}(t)).\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{W}_{p2}} + diag(\delta_{y_D}^{rec2}(t)).\frac{\partial \mathbf{c}_D(t)}{\partial \mathbf{W}_{p2}}, \tag{3.33}$$

where

$$\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{W}_{p2}} = diag(\mathbf{f}_Q(t)).\frac{\partial \mathbf{c}_Q(t-1)}{\partial \mathbf{W}_{p2}} + \mathbf{b}_{f,Q}(t).\mathbf{c}_Q(t-1)^T. \tag{3.34}$$

For forget gate's bias values we will have

$$\frac{\partial R(Q,D)}{\partial \mathbf{b}_2} = \\ diag(\delta_{y_Q}^{rec2}(t)).\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{b}_2} + diag(\delta_{y_D}^{rec2}(t)).\frac{\partial \mathbf{c}_D(t)}{\partial \mathbf{b}_2}, \tag{3.35}$$

where

$$\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{b}_2} = diag(\mathbf{f}_Q(t)).\frac{\partial \mathbf{c}_Q(t-1)}{\partial \mathbf{b}_3} + \mathbf{b}_{f,Q}(t). \tag{3.36}$$

Input without Gating ($\mathbf{y}_g(t)$)

For recurrent connections we will have

$$\frac{\partial R(Q,D)}{\partial \mathbf{W}_{rec4}} = diag(\delta_{y_Q}^{rec4}(t)).\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{W}_{rec4}} + diag(\delta_{y_D}^{rec4}(t)).\frac{\partial \mathbf{c}_D(t)}{\partial \mathbf{W}_{rec4}}, \tag{3.37}$$

where

$$\begin{aligned}
&\delta_{y_Q}^{rec4}(t) = (1-h(\mathbf{c}_Q(t))) \circ (1+h(\mathbf{c}_Q(t))) \circ \mathbf{o}_Q(t) \circ \mathbf{v}_Q(t) \\
&\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{W}_{rec4}} = diag(\mathbf{f}_Q(t)).\frac{\partial \mathbf{c}_Q(t-1)}{\partial \mathbf{W}_{rec4}} + \mathbf{b}_{g,Q}(t).\mathbf{y}_Q(t-1)^T \\
&\mathbf{b}_{g,Q}(t) = \mathbf{i}_Q(t) \circ (1-\mathbf{y}_{g,Q}(t)) \circ (1+\mathbf{y}_{g,Q}(t)).
\end{aligned} \tag{3.38}$$

For input connection we have

$$\frac{\partial R(Q,D)}{\partial \mathbf{W}_4} = diag(\delta_{y_Q}^{rec4}(t)).\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{W}_4} + diag(\delta_{y_D}^{rec4}(t)).\frac{\partial \mathbf{c}_D(t)}{\partial \mathbf{W}_4}, \tag{3.39}$$

where

$$\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{W}_4} = diag(\mathbf{f}_Q(t)).\frac{\partial \mathbf{c}_Q(t-1)}{\partial \mathbf{W}_4} + \mathbf{b}_{g,Q}(t).\mathbf{x}_Q(t)^T. \tag{3.40}$$

For bias values we will have

$$\frac{\partial R(Q,D)}{\partial \mathbf{b}_4} = diag(\delta_{y_Q}^{rec4}(t)).\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{b}_4} + diag(\delta_{y_D}^{rec4}(t)).\frac{\partial \mathbf{c}_D(t)}{\partial \mathbf{b}_4}, \tag{3.41}$$

where

$$\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{b}_4} = diag(\mathbf{f}_Q(t)).\frac{\partial \mathbf{c}_Q(t-1)}{\partial \mathbf{b}_4} + \mathbf{b}_{g,Q}(t). \tag{3.42}$$

Error Signal Backpropagation

Error signals are back propagated through time using following equations:

$$\begin{aligned}
&\delta_Q^{rec1}(t-1) = \\
&[\mathbf{o}_Q(t-1) \circ (1-\mathbf{o}_Q(t-1)) \circ h(\mathbf{c}_Q(t-1))] \\
&\circ \mathbf{W}_{rec1}^T.\delta_Q^{rec1}(t),
\end{aligned} \tag{3.43}$$

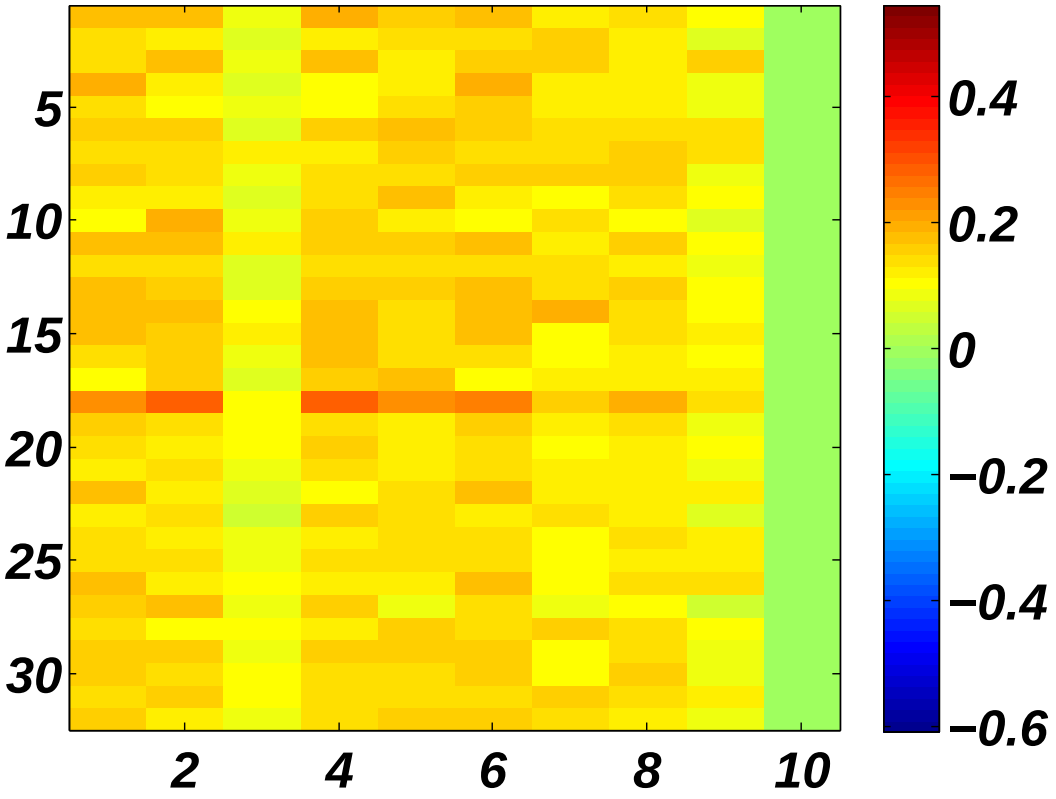


Figure 3.8: Input gate, $\mathbf{i}(t)$, for Document: "*shanghai hotels accommodation hotel in shanghai discount and reservation*"

$$\delta_Q^{rec_i}(t-1) = [(1-h(\mathbf{c}_Q(t-1))) \circ (1+h(\mathbf{c}_Q(t-1)))$$
$$\circ \mathbf{o}_Q(t-1)] \circ \mathbf{W}_{rec_i}^T . \delta_Q^{rec_i}(t), \quad for \quad i \in \{2,3,4\}. \tag{3.44}$$

3.7 A More Clear Figure for Input Gate for "*hotels in shanghai*" Example

In this section we present a more clear figure for part (a) of Fig. 3.5 that shows the structure of the input gate for document side of "*hotels in shanghai*" example. As it is clearly visible from Fig. 3.8, the input gate values for most of the cells corresponding to word 3, word 7 and word 9 in document side of LSTM-RNN have very small values (light green-yellow color). These are corresponding to words "*accommodation*", "*discount*" and "*reservation*" respectively in the document title. Interestingly, input gates are trying to reduce effect of these three words in the final representation ($\mathbf{y}(t)$) because the LSTM-RNN model is trained to maximize the similarity between query and document if they are a good match.

Table 3.5: RNNs with & without LSTM cells for the same query: "*hotels in shanghai*"

	hotels	*in*	*shanghai*
Number of assigned cells out of 10 (LSTM-RNN)	-	0	7
Number of assigned neurons out of 10 (RNN)	-	2	9

Table 3.6: RNNs with & without LSTM cells for the same Document: "*shanghai hotels accommodation hotel in shanghai discount and reservation*"

	shanghai	*hotels*	*accommodation*	*hotel*	*in*	*shanghai*	*discount*	*and*	*reservation*
Number of assigned cells out of 10 (LSTM-RNN)	-	4	3	8	1	8	5	3	4
Number of assigned neurons out of 10 (RNN)	-	10	7	9	6	8	3	2	6

3.8 A Closer Look at RNNs with and without LSTM Cells in Web Document Retrieval Task

In this section we further show examples to reveal the advantage of LSTM-RNN sentence embedding compared to the RNN sentence embedding.

First, we compare the scores assigned by trained RNN and LSTM-RNN to our "*hotels in shanghai*" example. On average, each query in our test dataset is associated with 15 web documents (URLs). Each query / document pair has a relevance label which is human generated. These relevance labels are "Bad", "Fair", "Good" and "Excellent". This example is rated as a "Good" match in the dataset. The score for this pair assigned by RNN is "0.8165" while the score assigned by LSTM-RNN is "0.9161". Please note that the score is between 0 and 1. This means that the score assigned by LSTM-RNN is more correspondent with the human generated label.

Second, we compare the number of assigned neurons and cells to each word by RNN and LSTM-RNN respectively. To do this, we rely on the 10 most active cells and neurons in the final semantic vectors in both models. Results are presented in Table 3.5 and Table 3.6 for query and document respectively. An interesting observation is that RNN sometimes assigns neurons to unimportant words, e.g., 6 neurons are assigned to the word "*in*" in Table 3.6.

As another example we consider the query, "*how to fix bath tub wont turn off*". This example is rated as a "Bad" match in the dataset by human. It is good to know that the score for this pair assigned by RNN is "0.7016" while the score assigned by LSTM-RNN is "0.5944". This shows the score generated by LSTM-RNN is closer to human generated label.

Number of assigned neurons and cells to each word by RNN and LSTM-RNN are presented in Table 3.7 and Table 3.8 for query and document. This is out of 10 most active neurons and cells in the semantic vector of RNN and LSTM-RNN. Examples of RNN assigning neurons to unimportant words are 3 neurons to the word "a" and 4 neurons to the word "you" in Table 3.8.

Table 3.7: RNN versus LSTM-RNN for Query: "*how to fix bath tub wont turn off*"

	how	*to*	*fix*	*bath*	*tub*	*wont*	*turn*	*off*
Number of assigned cells out of 10 (LSTM-RNN)	-	0	4	7	6	3	5	0
Number of assigned neurons out of 10 (RNN)	-	1	10	4	6	2	7	1

Table 3.8: RNN versus LSTM-RNN for Document: "*how do you paint a bathtub and what paint should ...* "

	how	*do*	*you*	*paint*	*a*	*bathtub*	*and*	*what*	*paint*	*should you ...*
Number of assigned cells out of 10(LSTM-RNN)	-	1	1	7	0	9	2	3	8	4
Number of assigned neurons out of 10(RNN)	-	1	4	4	3	7	2	5	4	7

3.9 Derivation of BPTT for RNN and LSTM-RNN

In this section we present the full derivation of the gradients for RNN and LSTM-RNN.

3.9.1 Derivation of BPTT for RNN

From (4) and (5) we have

$$\frac{\partial L(\Lambda)}{\partial \Lambda} = \sum_{r=1}^{N} \frac{\partial l_r(\Lambda)}{\partial \Lambda} = -\sum_{r=1}^{N} \sum_{j=1}^{n} \alpha_{r,j} \frac{\partial \Delta_{r,j}}{\partial \Lambda}, \tag{3.45}$$

where

$$\alpha_{r,j} = \frac{-\gamma e^{-\gamma \Delta_{r,j}}}{1 + \sum_{j=1}^{n} e^{-\gamma \Delta_{r,j}}}, \tag{3.46}$$

and

$$\Delta_{r,j} = R(Q_r, D_r^+) - R(Q_r, D_{r,j}). \tag{3.47}$$

We need to find $\frac{\partial \Delta_{r,j}}{\partial \Lambda}$ for input weights and recurrent weights. We omit r subscript for simplicity.

Recurrent Weights

$$\frac{\partial \Delta_j}{\partial \mathbf{W}_{rec}} = \frac{\partial R(Q, D^+)}{\partial \mathbf{W}_{rec}} - \frac{\partial R(Q, D_j^-)}{\partial \mathbf{W}_{rec}}. \tag{3.48}$$

We divide $R(D,Q)$ into three components:

$$R(Q,D) = \underbrace{\mathbf{y}_Q(t = T_Q)^T \mathbf{y}_D(t = T_D)}_{a} \cdot \underbrace{\frac{1}{\|\mathbf{y}_Q(t = T_Q)\|}}_{b} \cdot \underbrace{\frac{1}{\|\mathbf{y}_D(t = T_D)\|}}_{c}, \tag{3.49}$$

then

$$\frac{\partial R(Q,D)}{\partial \mathbf{W}_{rec}} = \underbrace{\frac{\partial a}{\partial \mathbf{W}_{rec}}.b.c}_{\mathbf{D}} + \underbrace{a.\frac{\partial b}{\partial \mathbf{W}_{rec}}.c}_{\mathbf{E}} + \underbrace{a.b.\frac{\partial c}{\partial \mathbf{W}_{rec}}}_{\mathbf{F}}. \tag{3.50}$$

We have

$$\begin{aligned}
\mathbf{D} &= \frac{\partial \mathbf{y}_Q(t=T_Q)^T \mathbf{y}_D(t=T_D).b.c}{\partial \mathbf{W}_{rec}} \\
&= \frac{\partial \mathbf{y}_Q(t=T_Q)^T \mathbf{y}_D(t=T_D).b.c}{\partial \mathbf{y}_Q(t=T_Q)}.\frac{\partial \mathbf{y}_Q(t=T_Q)}{\partial \mathbf{W}_{rec}} + \\
&\frac{\partial \mathbf{y}_Q(t=T_Q)^T \mathbf{y}_D(t=T_D).b.c}{\partial \mathbf{y}_D(t=T_D)}.\frac{\partial \mathbf{y}_D(t=T_D)}{\partial \mathbf{W}_{rec}} \\
&= \mathbf{y}_D(t=T_D).b.c.\frac{\partial \mathbf{y}_Q(t=T_Q)}{\partial \mathbf{W}_{rec}} + \\
&\mathbf{y}_Q(t=T_Q).\underbrace{(b.c)^T}_{b.c}.\frac{\partial \mathbf{y}_D(t=T_D)}{\partial \mathbf{W}_{rec}}.
\end{aligned} \tag{3.51}$$

Since $f(.) = tanh(.)$, using chain rule we have

$$\begin{aligned}
&\frac{\partial \mathbf{y}_Q(t=T_Q)}{\mathbf{W}_{rec}} = \\
&[(1-\mathbf{y}_Q(t=T_Q)) \circ (1+\mathbf{y}_Q(t=T_Q))]\mathbf{y}_Q(t-1)^T,
\end{aligned} \tag{3.52}$$

and therefore

$$\begin{aligned}
\mathbf{D} = &[b.c.\mathbf{y}_D(t=T_D) \circ (1-\mathbf{y}_Q(t=T_Q)) \circ \\
&(1+\mathbf{y}_Q(t=T_Q))]\mathbf{y}_Q(t-1)^T + \\
&[b.c.\mathbf{y}_Q(t=T_Q) \circ (1-\mathbf{y}_D(t=T_D)) \circ \\
&(1+\mathbf{y}_D(t=T_D))]\mathbf{y}_D(t-1)^T.
\end{aligned} \tag{3.53}$$

To find **E** we use following basic rule

$$\frac{\partial}{\partial \mathbf{x}}\|\mathbf{x}-\mathbf{a}\|_2 = \frac{\mathbf{x}-\mathbf{a}}{\|\mathbf{x}-\mathbf{a}\|_2}. \tag{3.54}$$

Therefore

$$\begin{aligned}
&\mathbf{E} = a.c.\frac{\partial}{\partial \mathbf{W}_{rec}}(\|\mathbf{y}_Q(t=T_Q)\|)^{-1} = \\
&-a.c.(\|\mathbf{y}_Q(t=T_Q)\|)^{-2}.\frac{\partial\|\mathbf{y}_Q(t=T_Q)\|}{\partial \mathbf{W}_{rec}} \\
&= -a.c.(\|\mathbf{y}_Q(t=T_Q)\|)^{-2}.\frac{\mathbf{y}_Q(t=T_Q)}{\|\mathbf{y}_Q(t=T_Q)\|}\frac{\partial \mathbf{y}_Q(t=T_Q)}{\partial \mathbf{W}_{rec}} \\
&= -[a.c.b^3.\mathbf{y}_Q(t=T_Q)\circ(1-\mathbf{y}_Q(t=T_Q))\circ \\
&(1+\mathbf{y}_Q(t=T_Q))]\mathbf{y}_Q(t-1),
\end{aligned} \tag{3.55}$$

where $\mathbf{F}$ is calculated similar to (3.55):

$$\begin{aligned}
&\mathbf{F} = -[a.b.c^3.\mathbf{y}_D(t=T_D)\circ(1-\mathbf{y}_D(t=T_D))\circ \\
&(1+\mathbf{y}_D(t=T_D))]\mathbf{y}_D(t-1).
\end{aligned} \tag{3.56}$$

Considering (3.50),(3.53),(3.55) and (3.56) we have

$$\frac{\partial R(Q,D)}{\partial \mathbf{W}_{rec}} = \delta_{\mathbf{y}_Q}(t)\mathbf{y}_Q(t-1)^T + \delta_{\mathbf{y}_D}(t)\mathbf{y}_D(t-1)^T, \tag{3.57}$$

where

$$\begin{aligned}
&\delta_{\mathbf{y}_Q}(t=T_Q) = (1-\mathbf{y}_Q(t=T_Q))\circ(1+\mathbf{y}_Q(t=T_Q))\circ \\
&(b.c.\mathbf{y}_D(t=T_D) - a.b^3.c.\mathbf{y}_Q(t=T_Q)), \\
&\delta_{\mathbf{y}_D}(t=T_D) = (1-\mathbf{y}_D(t=T_D))\circ(1+\mathbf{y}_D(t=T_D))\circ \\
&(b.c.\mathbf{y}_Q(t=T_Q) - a.b.c^3.\mathbf{y}_D(t=T_D)).
\end{aligned} \tag{3.58}$$

Equation (3.58) will just unfold the network one time step, to unfold it over rest of time steps using back-propagation we have

$$\begin{aligned}
&\delta_{\mathbf{y}_Q}(t-\tau-1) = (1-\mathbf{y}_Q(t-\tau-1))\circ \\
&(1+\mathbf{y}_Q(t-\tau-1))\circ\mathbf{W}_{rec}^T\delta_{\mathbf{y}_Q}(t-\tau), \\
&\delta_{\mathbf{y}_D}(t-\tau-1) = (1-\mathbf{y}_D(t-\tau-1))\circ \\
&(1+\mathbf{y}_D(t-\tau-1))\circ\mathbf{W}_{rec}^T\delta_{\mathbf{y}_D}(t-\tau),
\end{aligned} \tag{3.59}$$

where τ is the number of time steps that we unfold the network over time which is from 0 to T_Q and T_D for queries and documents respectively. Now using (3.48) we have

$$\begin{aligned}
&\frac{\partial \Delta_{j,\tau}}{\partial \mathbf{W}_{rec}} = [\delta_{y_Q}^{D^+}(t-\tau)\mathbf{y}_Q^T(t-\tau-1)+ \\
&\delta_{y_D}^{D^+}(t-\tau)\mathbf{y}_{D^+}^T(t-\tau-1)] - [\delta_{y_Q}^{D_j^-}(t-\tau)\mathbf{y}_Q^T(t-\tau-1) \\
&+ \delta_{y_D}^{D_j^-}(t-\tau)\mathbf{y}_{D_j^-}^T(t-\tau-1)].
\end{aligned} \tag{3.60}$$

To calculate final value of gradient we should fold back the network over time and use (3.45), we will have

$$\frac{\partial L(\Lambda)}{\partial \mathbf{W}_{rec}} = \underbrace{-\sum_{r=1}^{N}\sum_{j=1}^{n}\sum_{\tau=0}^{T}\alpha_{r,j,T_{D,Q}}\frac{\partial \Delta_{r,j,\tau}}{\partial \mathbf{W}_{rec}}}_{\text{one large update}}. \tag{3.61}$$

Input Weights

Using a similar procedure we will have the following for input weights:

$$\frac{\partial R(Q,D)}{\partial \mathbf{W}} = \delta_{\mathbf{y}_Q}(t-\tau)\mathbf{l}_Q(t-\tau)^T + \delta_{\mathbf{y}_D}(t-\tau)\mathbf{l}_D(t-\tau)^T, \tag{3.62}$$

where

$$\begin{aligned}
&\delta_{\mathbf{y}_Q}(t-\tau) = (1-\mathbf{y}_Q(t-\tau))\circ(1+\mathbf{y}_Q(t-\tau))\circ \\
&(b.c.\mathbf{y}_D(t-\tau) - a.b^3.c.\mathbf{y}_Q(t-\tau)), \\
&\delta_{\mathbf{y}_D}(t-\tau) = (1-\mathbf{y}_D(t-\tau))\circ(1+\mathbf{y}_D(t-\tau))\circ \\
&(b.c.\mathbf{y}_Q(t-\tau) - a.b.c^3.\mathbf{y}_D(t-\tau)).
\end{aligned} \tag{3.63}$$

Therefore

$$\begin{aligned}
&\frac{\partial \Delta_{j,\tau}}{\partial \mathbf{W}} = \\
&[\delta_{y_Q}^{D^+}(t-\tau)\mathbf{l}_Q^T(t-\tau) + \delta_{y_D}^{D^+}(t-\tau)\mathbf{l}_{D^+}^T(t-\tau)] - \\
&[\delta_{y_Q}^{D_j^-}(t-\tau)\mathbf{l}_Q^T(t-\tau) + \delta_{y_D}^{D_j^-}(t-\tau)\mathbf{l}_{D_j^-}^T(t-\tau)],
\end{aligned} \tag{3.64}$$

and therefore

$$\frac{\partial L(\Lambda)}{\partial \mathbf{W}} = \underbrace{-\sum_{r=1}^{N}\sum_{j=1}^{n}\sum_{\tau=0}^{T}\alpha_{r,j}\frac{\partial \Delta_{r,j,\tau}}{\partial \mathbf{W}}}_{\text{one large update}}. \tag{3.65}$$

3.9.2 Derivation of BPTT for LSTM-RNN

Following from (3.50) for every parameter, Λ, in LSTM-RNN architecture we have

$$\frac{\partial R(Q,D)}{\partial \Lambda} = \underbrace{\frac{\partial a}{\partial \Lambda}.b.c}_{\mathbf{D}} + \underbrace{a.\frac{\partial b}{\partial \Lambda}.c}_{\mathbf{E}} + \underbrace{a.b.\frac{\partial c}{\partial \Lambda}}_{\mathbf{F}}, \tag{3.66}$$

and from (3.51)

$$\begin{aligned}\mathbf{D} = \mathbf{y}_D(t=T_D).b.c.\frac{\partial \mathbf{y}_Q(t=T_Q)}{\partial \Lambda} + \\ \mathbf{y}_Q(t=T_Q).b.c.\frac{\partial \mathbf{y}_D(t=T_D)}{\partial \Lambda}.\end{aligned} \tag{3.67}$$

From (3.55) and (3.56) we have

$$\mathbf{E} = -a.c.b^3.\mathbf{y}_Q(t=T_Q)\frac{\partial \mathbf{y}_Q(t=T_Q)}{\partial \Lambda}, \tag{3.68}$$

$$\mathbf{F} = -a.b.c^3.\mathbf{y}_D(t=T_D)\frac{\partial \mathbf{y}_D(t=T_D)}{\partial \Lambda}. \tag{3.69}$$

Therefore

$$\begin{aligned}\frac{\partial R(Q,D)}{\partial \Lambda} = \mathbf{D}+\mathbf{E}+\mathbf{F} = \\ \mathbf{v}_Q\frac{\partial \mathbf{y}_Q(t=T_Q)}{\partial \Lambda} + \mathbf{v}_D\frac{\partial \mathbf{y}_D(t=T_D)}{\partial \Lambda},\end{aligned} \tag{3.70}$$

where

$$\begin{aligned}\mathbf{v}_Q = (b.c.\mathbf{y}_D(t=T_D) - a.b^3.c.\mathbf{y}_Q(t=T_Q)) \\ \mathbf{v}_D = (b.c.\mathbf{y}_Q(t=T_Q) - a.b.c^3.\mathbf{y}_D(t=T_D)).\end{aligned} \tag{3.71}$$

Output Gate

Since $\alpha \circ \beta = diag(\alpha)\beta = diag(\beta)\alpha$ where $diag(\alpha)$ is a diagonal matrix whose main diagonal entries are entries of vector α, we have

$$\begin{aligned}\frac{\partial \mathbf{y}(t)}{\partial \mathbf{W}_{rec1}} &= \frac{\partial}{\partial \mathbf{W}_{rec1}}(diag(h(\mathbf{c}(t))).\mathbf{o}(t)) \\ &= \underbrace{\frac{\partial diag(h(\mathbf{c}(t)))}{\partial \mathbf{W}_{rec1}}}_{zero}.\mathbf{o}(t) + diag(h(\mathbf{c}(t))).\frac{\partial \mathbf{o}(t)}{\partial \mathbf{W}_{rec1}} \\ &= \mathbf{o}(t)\circ(1-\mathbf{o}(t))\circ h(\mathbf{c}(t)).\mathbf{y}(t-1)^T.\end{aligned} \tag{3.72}$$

Substituting (3.72) in (3.70) we have

$$\frac{\partial R(Q,D)}{\partial \mathbf{W}_{rec1}} = \delta_{y_Q}^{rec1}(t).\mathbf{y}_Q(t-1)^T + \delta_{y_D}^{rec1}(t).\mathbf{y}_D(t-1)^T, \tag{3.73}$$

where

$$\begin{aligned}\delta_{y_Q}^{rec1}(t) &= \mathbf{o}_Q(t) \circ (1-\mathbf{o}_Q(t)) \circ h(\mathbf{c}_Q(t)) \circ \mathbf{v}_Q(t) \\ \delta_{y_D}^{rec1}(t) &= \mathbf{o}_D(t) \circ (1-\mathbf{o}_D(t)) \circ h(\mathbf{c}_D(t)) \circ \mathbf{v}_D(t),\end{aligned} \tag{3.74}$$

with a similar derivation for $\mathbf{W}_1$ and $\mathbf{W}_{p1}$ we get

$$\frac{\partial R(Q,D)}{\partial \mathbf{W}_1} = \delta_{y_Q}^{rec1}(t).\mathbf{l}_Q(t)^T + \delta_{y_D}^{rec1}(t).\mathbf{l}_D(t)^T, \tag{3.75}$$

$$\frac{\partial R(Q,D)}{\partial \mathbf{W}_{p1}} = \delta_{y_Q}^{rec1}(t).\mathbf{c}_Q(t)^T + \delta_{y_D}^{rec1}(t).\mathbf{c}_D(t)^T. \tag{3.76}$$

For output gate bias values we have

$$\frac{\partial R(Q,D)}{\partial \mathbf{b}_1} = \delta_{y_Q}^{rec1}(t) + \delta_{y_D}^{rec1}(t). \tag{3.77}$$

Input Gate

Similar to output gate we start with

$$\begin{aligned}&\frac{\partial \mathbf{y}(t)}{\partial \mathbf{W}_{rec3}} = \frac{\partial}{\partial \mathbf{W}_{rec3}}(diag(\mathbf{o}(t)).h(\mathbf{c}(t))) \\ &= \underbrace{\frac{\partial diag(\mathbf{o}(t))}{\partial \mathbf{W}_{rec3}}}_{zero}.h(\mathbf{c}(t)) + diag(\mathbf{o}(t)).\frac{\partial h(\mathbf{c}(t))}{\partial \mathbf{W}_{rec3}} \\ &= diag(\mathbf{o}(t)).(1-h(\mathbf{c}(t))) \circ (1+h(\mathbf{c}(t)))\frac{\partial \mathbf{c}(t)}{\partial \mathbf{W}_{rec3}}.\end{aligned} \tag{3.78}$$

To find $\frac{\partial \mathbf{c}(t)}{\partial \mathbf{W}_{rec3}}$ assuming $\mathbf{f}(t) = 1$ (we derive formulation for $\mathbf{f}(t) \neq 1$ from this simple solution) we have

$$\begin{aligned}&\mathbf{c}(0) = 0 \\ &\mathbf{c}(1) = \mathbf{c}(0) + \mathbf{i}(1) \circ \mathbf{y}_g(1) = \mathbf{i}(1) \circ \mathbf{y}_g(1) \\ &\mathbf{c}(2) = \mathbf{c}(1) + \mathbf{i}(2) \circ \mathbf{y}_g(2) \\ &\cdots \\ &\mathbf{c}(t) = \sum_{k=1}^{t} \mathbf{i}(k) \circ \mathbf{y}_g(k) = \sum_{k=1}^{t} diag(\mathbf{y}_g(k)).\mathbf{i}(k).\end{aligned} \tag{3.79}$$

Therefore

$$\frac{\partial \mathbf{c}(t)}{\partial \mathbf{W}_{rec3}} = \sum_{k=1}^{t} [\underbrace{\frac{\partial diag(\mathbf{y}_g(k))}{\mathbf{W}_{rec3}}}_{zero}.\mathbf{i}(k) + diag(\mathbf{y}_g(k)).\frac{\partial \mathbf{i}(k)}{\mathbf{W}_{rec3}}]$$

$$= \sum_{k=1}^{t} diag(\mathbf{y}_g(k)).\mathbf{i}(k) \circ (1-\mathbf{i}(k)).\mathbf{y}(k-1)^T \tag{3.80}$$

$$, \tag{3.81}$$

and

$$\frac{\partial \mathbf{y}(t)}{\partial \mathbf{W}_{rec3}} = \sum_{k=1}^{t} [\underbrace{\mathbf{o}(t) \circ (1-h(\mathbf{c}(t))) \circ (1+h(\mathbf{c}(t)))}_{\mathbf{a}(t)}$$

$$\circ \underbrace{\mathbf{y}_g(k) \circ \mathbf{i}(k) \circ (1-\mathbf{i}(k))}_{\mathbf{b}(k)}]\mathbf{y}(k-1)^T. \tag{3.82}$$

But this is expensive to implement, to resolve it we have

$$\frac{\partial \mathbf{y}(t)}{\partial \mathbf{W}_{rec3}} = \underbrace{\sum_{k=1}^{t-1} [\mathbf{a}(t) \circ \mathbf{b}(k)]\mathbf{y}(k-1)^T}_{\text{expensive part}}$$

$$+ [\mathbf{a}(t) \circ \mathbf{b}(t)]\mathbf{y}(t-1)^T$$

$$= diag(\mathbf{a}(t)) \underbrace{\sum_{k=1}^{t-1} \mathbf{b}(k).\mathbf{y}(k-1)^T}_{\frac{\partial \mathbf{c}(t-1)}{\partial \mathbf{W}_{rec3}}}$$

$$+ diag(\mathbf{a}(t)).\mathbf{b}(t).\mathbf{y}(t-1)^T. \tag{3.83}$$

Therefore

$$\frac{\partial \mathbf{y}(t)}{\partial \mathbf{W}_{rec3}} = [diag(\mathbf{a}(t))][\frac{\partial \mathbf{c}(t-1)}{\partial \mathbf{W}_{rec3}} + \mathbf{b}(t).\mathbf{y}(t-1)^T]. \tag{3.84}$$

For $\mathbf{f}(t) \neq 1$ we have

$$\frac{\partial \mathbf{y}(t)}{\partial \mathbf{W}_{rec3}} = [diag(\mathbf{a}(t))][diag(\mathbf{f}(t)).\frac{\partial \mathbf{c}(t-1)}{\partial \mathbf{W}_{rec3}}$$

$$+ \mathbf{b}_i(t).\mathbf{y}(t-1)^T], \tag{3.85}$$

where

$$\mathbf{a}(t) = \mathbf{o}(t) \circ (1-h(\mathbf{c}(t))) \circ (1+h(\mathbf{c}(t)))$$

$$\mathbf{b}_i(t) = \mathbf{y}_g(t) \circ \mathbf{i}(t) \circ (1-\mathbf{i}(t)). \tag{3.86}$$

Substituting above equation in (3.70) we will have

$$\frac{\partial R(Q,D)}{\partial \mathbf{W}_{rec3}} = diag(\delta_{y_Q}^{rec3}(t)).\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{W}_{rec3}} + diag(\delta_{y_D}^{rec3}(t)).\frac{\partial \mathbf{c}_D(t)}{\partial \mathbf{W}_{rec3}}, \tag{3.87}$$

where

$$\begin{aligned}
\delta_{y_Q}^{rec3}(t) &= (1-h(\mathbf{c}_Q(t))) \circ (1+h(\mathbf{c}_Q(t))) \circ \mathbf{o}_Q(t) \circ \mathbf{v}_Q(t) \\
\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{W}_{rec3}} &= diag(\mathbf{f}_Q(t)).\frac{\partial \mathbf{c}_Q(t-1)}{\partial \mathbf{W}_{rec3}} + \mathbf{b}_{i,Q}(t).\mathbf{y}_Q(t-1)^T \\
\mathbf{b}_{i,Q}(t) &= \mathbf{y}_{g,Q}(t) \circ \mathbf{i}_Q(t) \circ (1-\mathbf{i}_Q(t)).
\end{aligned} \tag{3.88}$$

In equation (3.87), $\delta_{y_D}^{rec3}(t)$ and $\frac{\partial \mathbf{c}_D(t)}{\partial \mathbf{W}_{rec3}}$ are the same as (3.88) with D subscript. Therefore, update equations for $\mathbf{W}_{rec3}$ are (3.87), (3.88) for Q and D and (6).

With a similar procedure for $\mathbf{W}_3$ we will have the following:

$$\frac{\partial R(Q,D)}{\partial \mathbf{W}_3} = diag(\delta_{y_Q}^{rec3}(t)).\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{W}_3} + diag(\delta_{y_D}^{rec3}(t)).\frac{\partial \mathbf{c}_D(t)}{\partial \mathbf{W}_3}, \tag{3.89}$$

where

$$\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{W}_3} = diag(\mathbf{f}_Q(t)).\frac{\partial \mathbf{c}_Q(t-1)}{\partial \mathbf{W}_3} + \mathbf{b}_{i,Q}(t).\mathbf{x}_Q(t)^T. \tag{3.90}$$

Therefore, update equations for $\mathbf{W}_3$ are (3.89), (3.90) for Q and D and (6).

For peephole connections we will have

$$\frac{\partial R(Q,D)}{\partial \mathbf{W}_{p3}} = diag(\delta_{y_Q}^{rec3}(t)).\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{W}_{p3}} + diag(\delta_{y_D}^{rec3}(t)).\frac{\partial \mathbf{c}_D(t)}{\partial \mathbf{W}_{p3}}, \tag{3.91}$$

where

$$\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{W}_{p3}} = diag(\mathbf{f}_Q(t)).\frac{\partial \mathbf{c}_Q(t-1)}{\partial \mathbf{W}_{p3}} + \mathbf{b}_{i,Q}(t).\mathbf{c}_Q(t-1)^T. \tag{3.92}$$

Hence, update equations for $\mathbf{W}_{p3}$ are (3.91), (3.92) for Q and D and (6).

Following similar derivation for bias values $\mathbf{b}_3$ we will have

$$\frac{\partial R(Q,D)}{\partial \mathbf{b}_3} = diag(\delta_{y_Q}^{rec3}(t)).\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{b}_3} + diag(\delta_{y_D}^{rec3}(t)).\frac{\partial \mathbf{c}_D(t)}{\partial \mathbf{b}_3}, \tag{3.93}$$

where

$$\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{b}_3} = diag(\mathbf{f}_Q(t)).\frac{\partial \mathbf{c}_Q(t-1)}{\partial \mathbf{b}_3} + \mathbf{b}_{i,Q}(t). \tag{3.94}$$

Update equations for $\mathbf{b}_3$ are (3.93), (3.94) for Q and D and (6).

Forget Gate

For forget gate, with a similar derivation to input gate we will have

$$\begin{aligned}\frac{\partial \mathbf{y}(t)}{\partial \mathbf{W}_{rec2}} &= [diag(\mathbf{a}(t))][diag(\mathbf{f}(t)).\frac{\partial \mathbf{c}(t-1)}{\partial \mathbf{W}_{rec2}} \\ &+ \mathbf{b}_f(t).\mathbf{y}(t-1)^T],\end{aligned} \tag{3.95}$$

where

$$\begin{aligned}\mathbf{a}(t) &= \mathbf{o}(t) \circ (1 - h(\mathbf{c}(t))) \circ (1 + h(\mathbf{c}(t))) \\ \mathbf{b}_f(t) &= \mathbf{c}(t-1) \circ \mathbf{f}(t) \circ (1 - \mathbf{f}(t)).\end{aligned} \tag{3.96}$$

Substituting above equation in (3.70) we will have

$$\begin{aligned}\frac{\partial R(Q,D)}{\partial \mathbf{W}_{rec2}} &= diag(\delta_{y_Q}^{rec2}(t)).\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{W}_{rec2}} \\ &+ diag(\delta_{y_D}^{rec2}(t)).\frac{\partial \mathbf{c}_D(t)}{\partial \mathbf{W}_{rec2}},\end{aligned} \tag{3.97}$$

where

$$\begin{aligned}\delta_{y_Q}^{rec2}(t) &= (1 - h(\mathbf{c}_Q(t))) \circ (1 + h(\mathbf{c}_Q(t))) \circ \mathbf{o}_Q(t) \circ \mathbf{v}_Q(t) \\ \frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{W}_{rec2}} &= diag(\mathbf{f}_Q(t)).\frac{\partial \mathbf{c}_Q(t-1)}{\partial \mathbf{W}_{rec2}} + \mathbf{b}_{f,Q}(t).\mathbf{y}_Q(t-1)^T \\ \mathbf{b}_{f,Q}(t) &= \mathbf{c}_Q(t-1) \circ \mathbf{f}_Q(t) \circ (1 - \mathbf{f}_Q(t)).\end{aligned} \tag{3.98}$$

Therefore, update equations for $\mathbf{W}_{rec2}$ are (3.97), (3.98) for Q and D and (6).

For input weights to forget gate, $\mathbf{W}_2$, we have

$$\begin{aligned}\frac{\partial R(Q,D)}{\partial \mathbf{W}_2} &= diag(\delta_{y_Q}^{rec2}(t)).\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{W}_2} \\ &+ diag(\delta_{y_D}^{rec2}(t)).\frac{\partial \mathbf{c}_D(t)}{\partial \mathbf{W}_2},\end{aligned} \tag{3.99}$$

where

$$\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{W}_2} = diag(\mathbf{f}_Q(t)).\frac{\partial \mathbf{c}_Q(t-1)}{\partial \mathbf{W}_2} + \mathbf{b}_{f,Q}(t).\mathbf{x}_Q(t)^T. \tag{3.100}$$

Therefore, update equations for $\mathbf{W}_2$ are (3.99), (3.100) for Q and D and (6).

For peephole connections, $\mathbf{W}_{p2}$, we have

$$\frac{\partial R(Q,D)}{\partial \mathbf{W}_{p2}} = diag(\delta_{y_Q}^{rec2}(t)).\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{W}_{p2}} + diag(\delta_{y_D}^{rec2}(t)).\frac{\partial \mathbf{c}_D(t)}{\partial \mathbf{W}_{p2}}, \tag{3.101}$$

where

$$\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{W}_{p2}} = diag(\mathbf{f}_Q(t)).\frac{\partial \mathbf{c}_Q(t-1)}{\partial \mathbf{W}_{p2}} + \mathbf{b}_{f,Q}(t).\mathbf{c}_Q(t-1)^T. \tag{3.102}$$

Therefore, update equations for $\mathbf{W}_{p2}$ are (3.101), (3.102) for Q and D and (6).

Update equations for forget gate bias values, $\mathbf{b}_2$, will be following equations and (6):

$$\frac{\partial R(Q,D)}{\partial \mathbf{b}_2} = diag(\delta_{y_Q}^{rec2}(t)).\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{b}_2} + diag(\delta_{y_D}^{rec2}(t)).\frac{\partial \mathbf{c}_D(t)}{\partial \mathbf{b}_2}, \tag{3.103}$$

where

$$\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{b}_2} = diag(\mathbf{f}_Q(t)).\frac{\partial \mathbf{c}_Q(t-1)}{\partial \mathbf{b}_3} + \mathbf{b}_{f,Q}(t). \tag{3.104}$$

Input without Gating ($\mathbf{y}_g(t)$)

Gradients for $\mathbf{y}_g(t)$ parameters are as follows:

$$\frac{\partial \mathbf{y}(t)}{\partial \mathbf{W}_{rec4}} = [diag(\mathbf{a}(t))][diag(\mathbf{f}(t)).\frac{\partial \mathbf{c}(t-1)}{\partial \mathbf{W}_{rec4}} + \mathbf{b}_g(t).\mathbf{y}(t-1)^T], \tag{3.105}$$

where

$$\begin{aligned} \mathbf{a}(t) &= \mathbf{o}(t) \circ (1 - h(\mathbf{c}(t))) \circ (1 + h(\mathbf{c}(t))) \\ \mathbf{b}_g(t) &= \mathbf{i}(t) \circ (1 - \mathbf{y}_g(t)) \circ (1 + \mathbf{y}_g(t)). \end{aligned} \tag{3.106}$$

Substituting above equation in (3.70) we will have

$$\frac{\partial R(Q,D)}{\partial \mathbf{W}_{rec4}} = diag(\delta_{y_Q}^{rec4}(t)).\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{W}_{rec4}} + diag(\delta_{y_D}^{rec4}(t)).\frac{\partial \mathbf{c}_D(t)}{\partial \mathbf{W}_{rec4}}, \tag{3.107}$$

where

$$\delta_{y_Q}^{rec4}(t) = (1 - h(\mathbf{c}_Q(t))) \circ (1 + h(\mathbf{c}_Q(t))) \circ \mathbf{o}_Q(t) \circ \mathbf{v}_Q(t)$$
$$\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{W}_{rec4}} = diag(\mathbf{f}_Q(t)).\frac{\partial \mathbf{c}_Q(t-1)}{\partial \mathbf{W}_{rec4}} + \mathbf{b}_{g,Q}(t).\mathbf{y}_Q(t-1)^T$$
$$\mathbf{b}_{g,Q}(t) = \mathbf{i}_Q(t) \circ (1 - \mathbf{y}_{g,Q}(t)) \circ (1 + \mathbf{y}_{g,Q}(t)). \tag{3.108}$$

Therefore, update equations for $\mathbf{W}_{rec4}$ are (3.107), (3.108) for Q and D and (6).

For input weight parameters, $\mathbf{W}_4$, we have

$$\frac{\partial R(Q,D)}{\partial \mathbf{W}_4} = diag(\delta_{y_Q}^{rec4}(t)).\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{W}_4}$$
$$+ diag(\delta_{y_D}^{rec4}(t)).\frac{\partial \mathbf{c}_D(t)}{\partial \mathbf{W}_4}, \tag{3.109}$$

where

$$\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{W}_4} = diag(\mathbf{f}_Q(t)).\frac{\partial \mathbf{c}_Q(t-1)}{\partial \mathbf{W}_4} + \mathbf{b}_{g,Q}(t).\mathbf{x}_Q(t)^T. \tag{3.110}$$

Therefore, update equations for $\mathbf{W}_4$ are (3.109), (3.110) for Q and D and (6).

Gradients with respect to bias values, $\mathbf{b}_4$, are

$$\frac{\partial R(Q,D)}{\partial \mathbf{b}_4} = diag(\delta_{y_Q}^{rec4}(t)).\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{b}_4}$$
$$+ diag(\delta_{y_D}^{rec4}(t)).\frac{\partial \mathbf{c}_D(t)}{\partial \mathbf{b}_4}, \tag{3.111}$$

where

$$\frac{\partial \mathbf{c}_Q(t)}{\partial \mathbf{b}_4} = diag(\mathbf{f}_Q(t)).\frac{\partial \mathbf{c}_Q(t-1)}{\partial \mathbf{b}_4} + \mathbf{b}_{g,Q}(t). \tag{3.112}$$

Therefore, update equations for $\mathbf{b}_4$ are (3.111), (3.112) for Q and D and (6). There is no peephole connections for $\mathbf{y}_g(t)$.

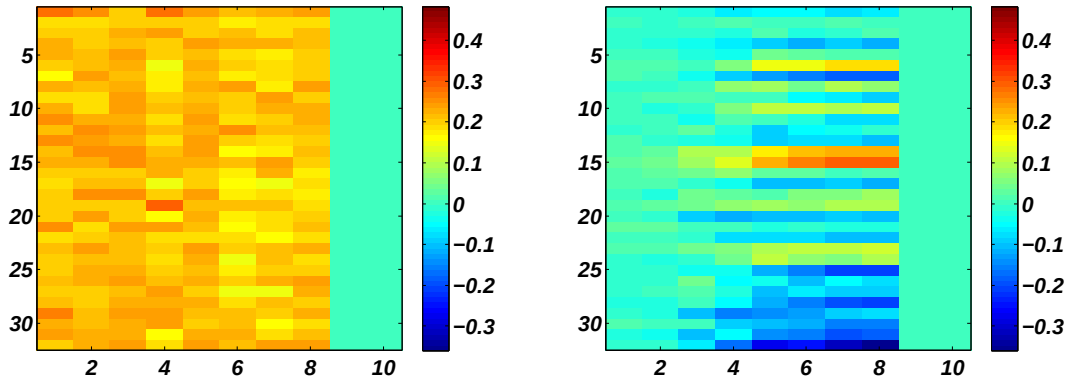
3.10 LSTM-RNN Visualization

In this section we present more examples of LSTM-RNN visualization.

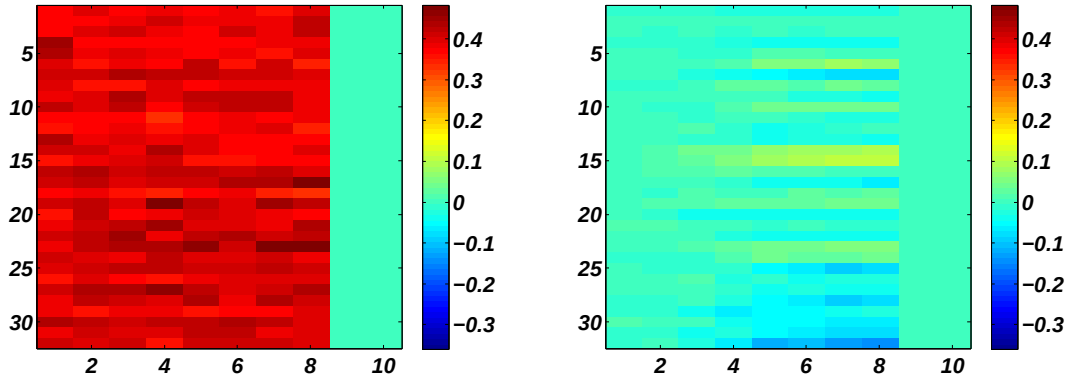
3.10.1 LSTM-RNN Semantic Vectors: Another Example

Consider the following example from test dataset:

- Query: "*how to fix bath tub wont turn off*"
- Document: "*how do you paint a bathtub and what paint* $\underbrace{\textit{should you use yahoo answers}}_{\text{treated as one word}}$"



(a) $\mathbf{i}(t)$ (b) $\mathbf{c}(t)$



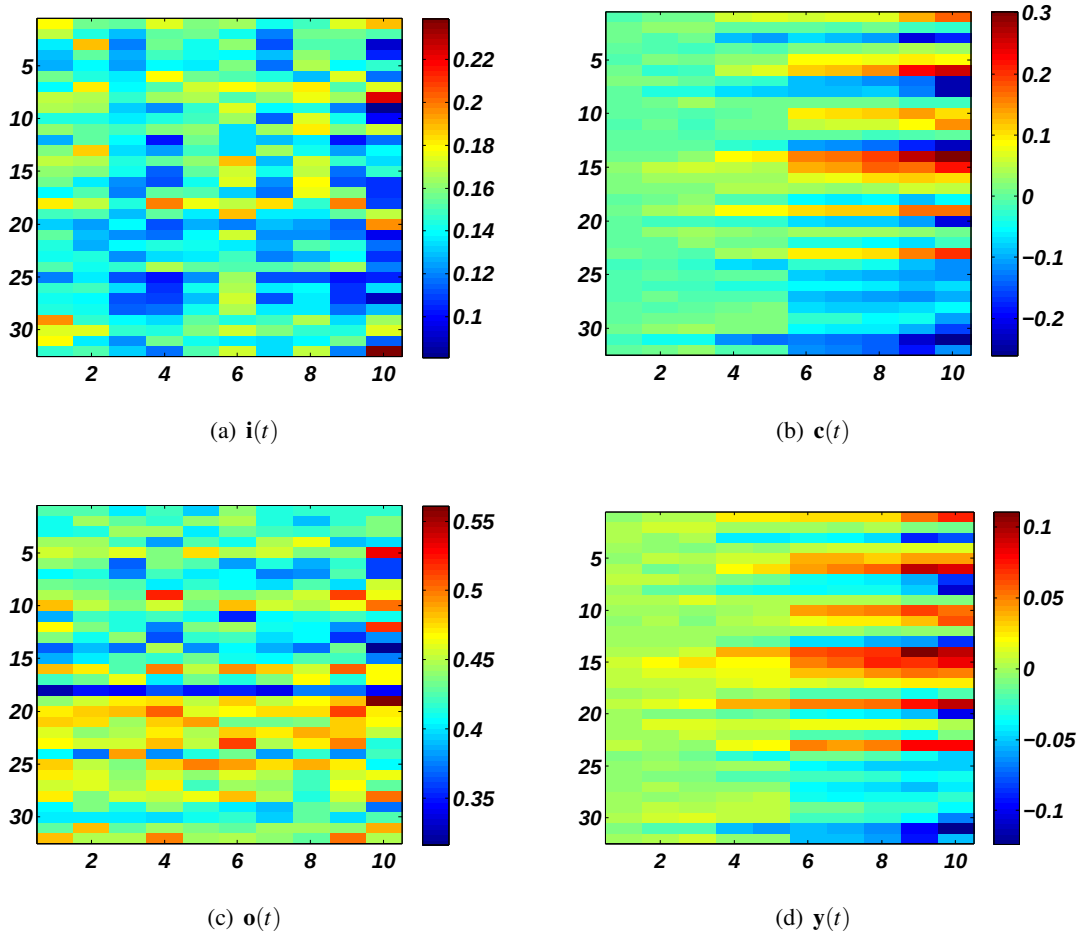
(c) $\mathbf{o}(t)$ (d) $\mathbf{y}(t)$

Figure 3.9: Query: "*how to fix bath tub wont turn off*"

Activations of input gate, output gate, cell state and cell output for each cell for query and document are presented in Fig.3.9 and Fig.3.10 respectively based on a trained LSTM-RNN model.

Three interesting observations from Fig.3.9 and Fig.3.10:

- Semantic representation $\mathbf{y}(t)$ and cell states $\mathbf{c}(t)$ are evolving over time.
- In part (a) of Fig.3.10, we observe that input gate values for most of the cells corresponding to word 3, word 4, word 7 and word 9 in document side of LSTM-RNN have very small values (light blue color). These are corresponding to words "*you*", "*paint*", "*and*" and "*paint*" respectively in the document title. Interestingly, input gates are trying to reduce effect of these words in the final representation ($\mathbf{y}(t)$) because the LSTM-RNN model is trained to maximize the similarity between query and document if they are a good match.
- $\mathbf{y}(t)$ is used as semantic representation after applying output gate on cell states. Note that valuable context information is stored in cell states $\mathbf{c}(t)$.



(a) $\mathbf{i}(t)$ (b) $\mathbf{c}(t)$

(c) $\mathbf{o}(t)$ (d) $\mathbf{y}(t)$

Figure 3.10: Document: "*how do you paint a bathtub and what paint should . . .* "

Table 3.9: Keyword extraction for Query: "*how to fix bath tub wont turn off*"

	how	*to*	*fix*	*bath*	*tub*	*wont*	*turn*	*off*
Number of assigned cells out of 10 Left to Right	-	0	**4**	**7**	**6**	3	**5**	0
Number of assigned cells out of 10 Right to Left	**4**	1	**6**	**7**	**6**	**7**	**7**	-

3.10.2 Key Word Extraction: Another Example

Evolution of 10 most active cells over time for the second example are presented in Fig. 3.11 for query and Fig. 3.12 for document. Number of assigned cells out of 10 most active cells to each word are presented in Table 3.9 and Table 3.10.

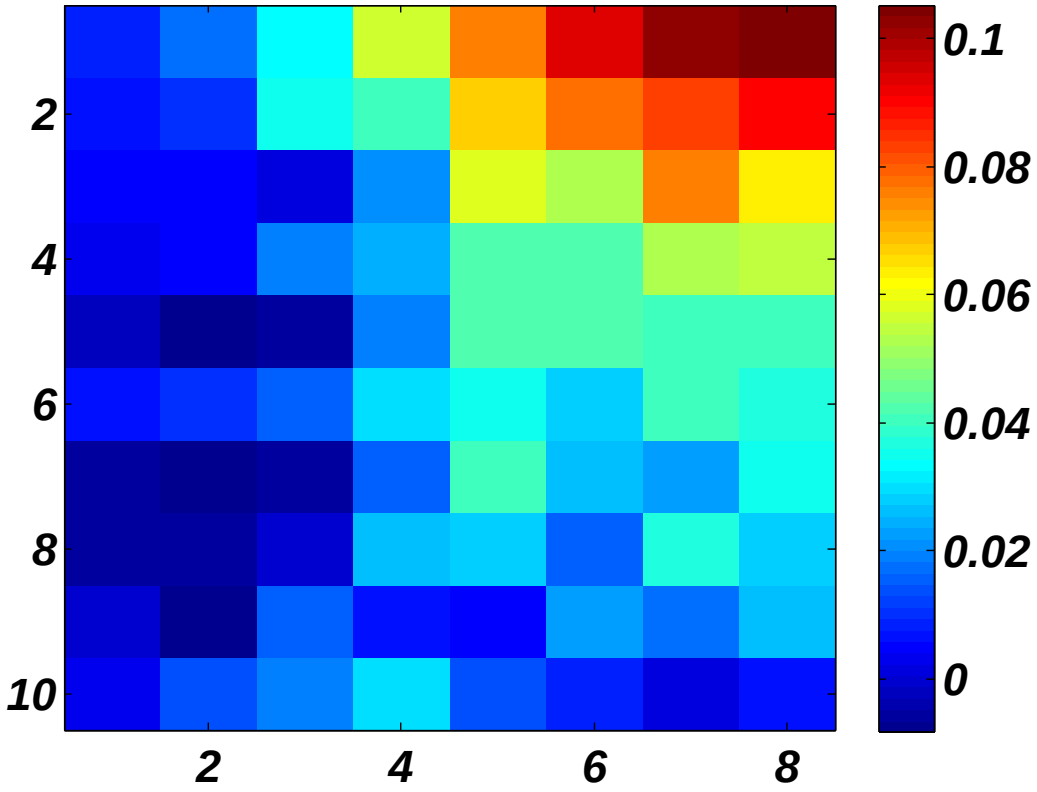


Figure 3.11: Query: "*how to fix bath tub wont turn off*"

Table 3.10: Keyword extraction for Document: "*how do you paint a bathtub and what paint should . . .*"

	how	*do*	*you*	*paint*	*a*	*bathtub*	*and*	*what*	*paint*	*should you . . .*
Number of assigned cells out of 10 Left to Right	-	1	1	**7**	0	**9**	2	3	**8**	4
Number of assigned cells out of 10 Right to Left	**5**	**9**	**5**	**4**	**8**	**4**	**5**	**5**	**9**	-

3.11 Doc2Vec Similarity Test

To make sure that a meaningful model is trained, we used the trained doc2vec model to find the most similar words to two sample words in our dataset, the words "pizza" and "infection". The resulting words and corresponding scores are as follows:

---

print(model.most-similar('pizza')) :

[(u'recipes', 0.9316294193267822),

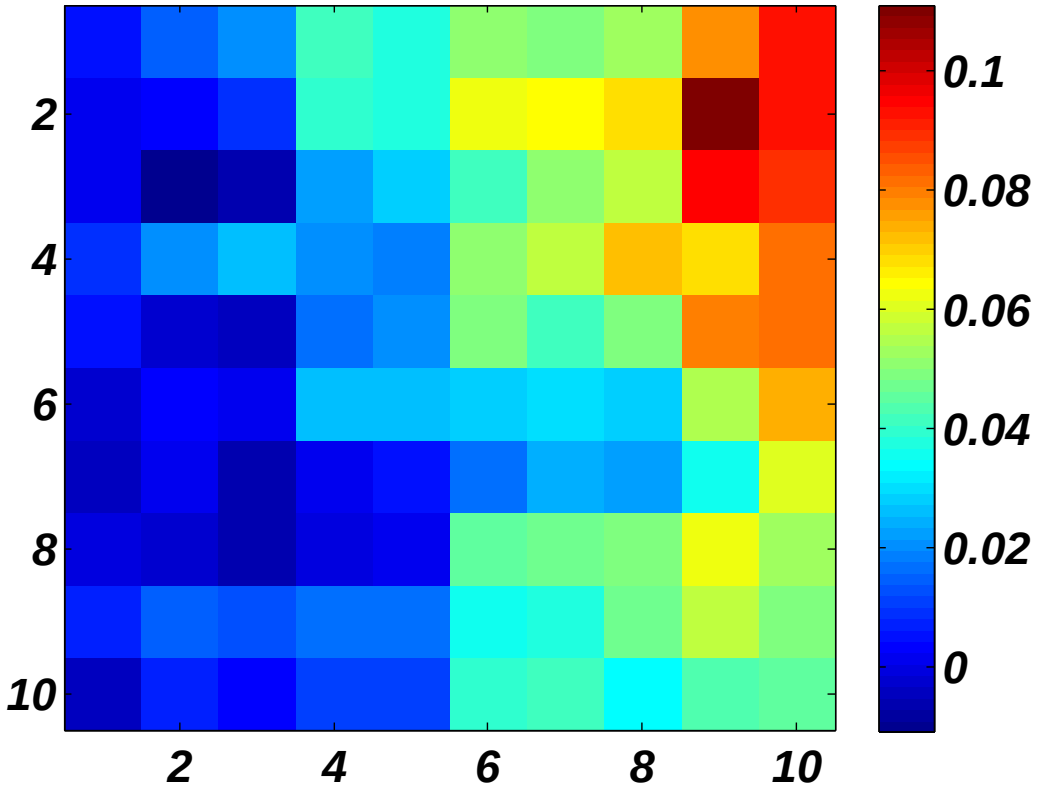


Figure 3.12: Document: *"how do you paint a bathtub and what paint should . . . "*

(u'recipe', 0.9295548796653748),
(u'food', 0.9250608682632446),
(u'restaurants', 0.9223555326461792),
(u'bar', 0.9191627502441406),
(u'sabayon', 0.916868269443512),
(u'page', 0.9160783290863037),
(u'restaurant', 0.9112323522567749),
(u'house', 0.9104640483856201),
(u'the', 0.9103578925132751)]

---

print(model.most-similar('infection')):

[(u'infections', 0.9698576927185059),
(u'treatment', 0.9143450856208801),
(u'symptoms', 0.9138627052307129),
(u'disease', 0.9100595712661743),

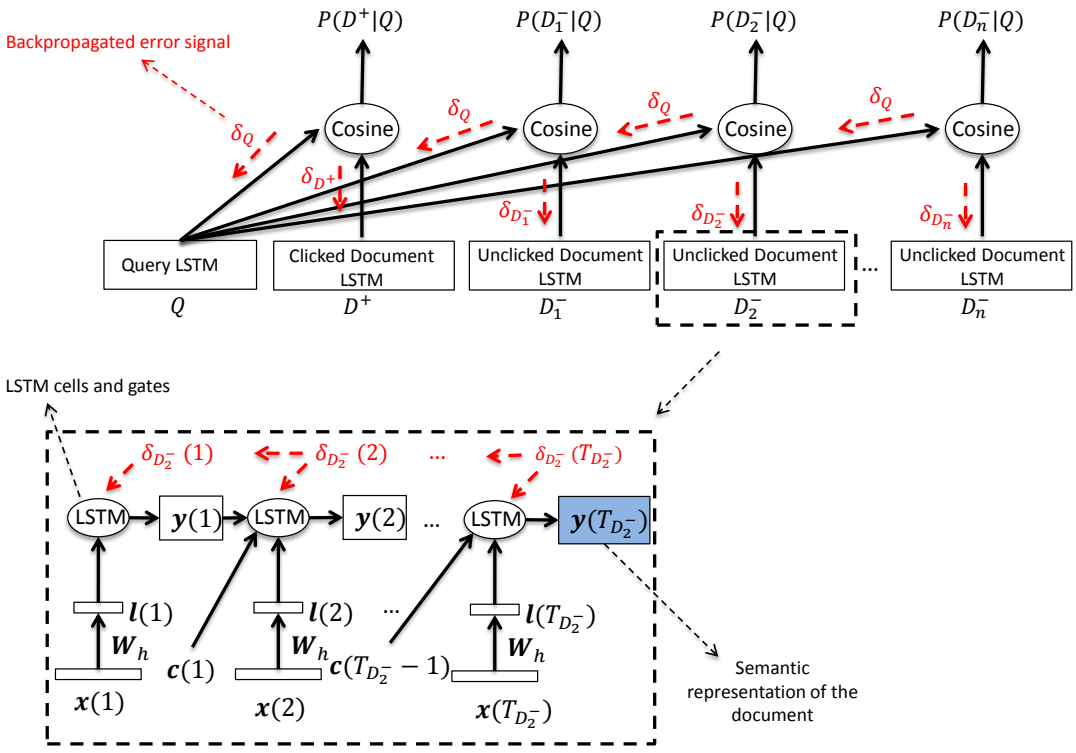


Figure 3.13: Architecture of the proposed method.

(u'palpitations', 0.9083651304244995),
(u'pneumonia', 0.9073051810264587),
(u'medical', 0.9043352603912354),
(u'abdomen', 0.9034136533737183),
(u'medlineplus', 0.9032401442527771),
(u'gout', 0.9027985334396362)]

As it is observed from the resulting words, the trained model is a meaningful model and can recognise semantic similarity.

3.12 Diagram of the Proposed Model

To clarify the difference between the proposed method and the general sentence embedding methods, in this section we present a diagram illustrating the training procedure of the proposed model. It is presented in Fig. 3.13. In this figure n is the number of negative (unclicked) documents. The other parameters in this figure are similar to those used in Fig. 2 and Fig. 3 of this chapter.

3.13 Conclusions

This chapter addresses deep sentence embedding. We propose a model based on long short-term memory to model the long range context information and embed the key information of a sentence in one semantic vector. We show that the semantic vector evolves over time and only takes useful information from any new input. This has been made possible by input gates that detect useless information and attenuate it. Due to general limitation of available human labelled data, we proposed and implemented training the model with a *weak supervision* signal using user click-through data of a commercial web search engine.

By performing a detailed analysis on the model, we showed that: 1) The proposed model is robust to noise, i.e., it mainly embeds keywords in the final semantic vector representing the whole sentence and 2) In the proposed model, each cell is usually allocated to keywords from a specific topic. These findings have been supported using extensive examples. As a concrete sample application of the proposed sentence embedding method, we evaluated it on the important language processing task of web document retrieval. We showed that, for this task, the proposed method outperforms all existing state of the art methods significantly.

This work has been motivated by the earlier successes of deep learning methods in speech [20, 21, 29, 57, 129] and in semantic modelling [43, 44, 65, 108], and it adds further evidence for the effectiveness of these methods.

Chapter 4

A Deep Learning Approach to Distributed Compressive Sensing

When a traveller reaches a fork in the road, the ℓ_1-norm tells him to take either one way or the other, but the ℓ_2-norm instructs him to head off into the bushes!
— John F. Claerbout and Francis Muir

4.1 Introduction

Compressive Sensing (CS) [31],[14],[9] is an effective approach for acquiring sparse signals where both sensing and compression are performed at the same time. Since there are numerous examples of natural and artificial signals that are sparse in the time, spatial or a transform domain, CS has found numerous applications. These include medical imaging, geophysical data analysis, computational biology, remote sensing and communications.

In the general CS framework, instead of acquiring N samples of a signal $\mathbf{x} \in \Re^{N\times 1}$, M random measurements are acquired where $M < N$. This is expressed by the underdetermined system of linear equations

$$\mathbf{y} = \Phi\mathbf{x}, \tag{4.1}$$

where $\mathbf{y} \in \Re^{M\times 1}$ is the known measured vector and $\Phi \in \Re^{M\times N}$ is a random measurement matrix. To uniquely recover $\mathbf{x}$ given $\mathbf{y}$ and Φ, $\mathbf{x}$ must be sparse in a given basis Ψ. This means that

$$\mathbf{x} = \Psi\mathbf{s}, \tag{4.2}$$

where $\mathbf{s}$ is $K-sparse$, i.e., $\mathbf{s}$ has at most K non-zero elements. The basis Ψ can be complete; i.e., $\Psi \in \Re^{N\times N}$, or over-complete; i.e., $\Psi \in \Re^{N\times N_1}$ where $N < N_1$ (compressed sensing for over-complete dictionaries is introduced in [15]). From (4.1) and (4.2)

$$\mathbf{y} = \mathbf{A}\mathbf{s}, \tag{4.3}$$

where $\mathbf{A} = \Phi\Psi$. Since there is only one measurement vector, the above problem is usually called the Single

Measurement Vector (SMV) problem in compressive sensing.

In distributed compressive sensing , also known as the Multiple Measurement Vectors (MMV) problem, a set of L sparse vectors $\{\mathbf{s}_i\}_{i=1,2,\ldots,L}$ is to be jointly recovered from a set of L measurement vectors $\{\mathbf{y}_i\}_{i=1,2,\ldots,L}$. Some application areas of MMV include magnetoencephalography, array processing, equalization of sparse communication channels and cognitive radio [33].

Suppose that the L sparse vectors and the L measurement vectors are arranged as columns of matrices $\mathbf{S} = [\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_L]$ and $\mathbf{Y} = [\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_L]$ respectively. In the MMV problem, $\mathbf{S}$ is to be reconstructed given $\mathbf{Y}$

$$\mathbf{Y} = \mathbf{AS}. \tag{4.4}$$

In (4.4), $\mathbf{S}$ is assumed to be jointly sparse, i.e., non-zero entries of each vector occur at the same locations as those of other vectors, which means that the sparse vectors have the same support. Assume that $\mathbf{S}$ is jointly sparse. Then, the necessary and sufficient condition to obtain a unique $\mathbf{S}$ given $\mathbf{Y}$ is [22]

$$|supp(\mathbf{S})| < \frac{spark(\mathbf{A}) - 1 + rank(\mathbf{S})}{2}, \tag{4.5}$$

where $|supp(\mathbf{S})|$ is the number of rows in $\mathbf{S}$ with non-zero energy and *spark* of a given matrix is the smallest possible number of linearly dependent columns of that matrix. *spark* gives a measure of linear dependency in the system modelled by a given matrix. In the SMV problem, no rank information exists. In the MMV problem, the rank information exists and affects the uniqueness bounds. Generally, solving the MMV problem jointly can lead to better uniqueness guarantees than solving the SMV problem for each vector independently [35].

In the current MMV literature, a jointly sparse matrix is recovered typically by one of the following methods: 1) greedy methods [120] like Simultaneous Orthogonal Matching Pursuit (SOMP) which performs non-optimal subset selection, 2) relaxed mixed norm minimization methods [82, 119, 121, 134], or 3) Bayesian methods like [72, 126, 133] where a posterior density function for the values of $\mathbf{S}$ is created, assuming a prior belief, e.g., $\mathbf{Y}$ is observed and $\mathbf{S}$ should be sparse in basis Ψ. The selection of one of the above methods depends on the requirements imposed by the specific application.

4.1.1 Problem Statement

The MMV reconstruction methods stated above do not rely on the use of training data. However, for many applications, a large amount of data similar to the data to be compressed by CS is available. Examples are camera recordings of the same environment, images of the same class (e.g., flowers, buildings, ...), electroencephalogram (EEG) of different parts of the brain, etc. In this chapter, we address the following questions in the MMV problem when training data is available:

1. Can we learn the structure of the sparse vectors in $\mathbf{S}$ by a data driven bottom up approach using the already available training data? If yes, then how can we exploit this structure in the MMV problem to design a better reconstruction method?

2. Most of the reconstruction algorithms for the MMV problem rely on the joint sparsity of $\mathbf{S}$. However,

in some practical applications, the sparse vectors in $\mathbf{S}$ are not exactly jointly sparse. This can be due to noise or due to sources that create different sparsity patterns. Examples are images of different scenes captured by different cameras, images of different classes, etc. Although $\mathbf{S}$ is not jointly sparse, there may exist a possible dependency among the columns of $\mathbf{S}$, however, due to lack of joint sparsity, the above methods will not give satisfactory performance. The question is, can we design the aforementioned data driven method in a way that it captures the dependencies among the sparse vectors in $\mathbf{S}$? The type of such dependencies may not be necessarily that of joint sparsity. And then how can we use the learned dependency structure in the reconstruction algorithm at the decoder?

Please note that we want to address the above questions "*without adding any complexity or adaptability*" to the encoder. In other words, our aim is not to design an optimal encoder, i.e., optimal sensing matrix Φ or the sparsifying basis Ψ, for the given training data. The encoder would be as simple and general as possible. This is specially important for applications that use sensors having low power consumption due to a limited battery life. However, the decoder in these cases can be much more complex than the encoder. For example, the decoder can be a powerful data processing machine.

4.1.2 Proposed Method

To address the above questions, we propose the use of a two step greedy reconstruction algorithm. In the first step, at each iteration of the reconstruction algorithm, and for each column of $\mathbf{S}$ represented as $\mathbf{s}_i$, we first find the conditional probability of each entry of $\mathbf{s}_i$ being non-zero, given the residuals of all previous sparse vectors (columns) at that iteration. Then we select the most probable entry and add it to the support of $\mathbf{s}_i$. The definition of the residual matrix at the $j-$th iteration is $\mathbf{R}_j = \mathbf{Y} - \mathbf{A}\mathbf{S}_j$ where $\mathbf{S}_j$ is the estimate of the sparse matrix $\mathbf{S}$ at the $j-$th iteration. Therefore in the first step, we find the locations of the non-zero entries. In the second step we find the values of these non-zero entries. This can be done by solving a least squares problem that finds $\mathbf{s}_i$ given $\mathbf{y}_i$ and $\mathbf{A}_{\Omega_i}$. $\mathbf{A}_{\Omega_i}$ is a matrix that includes only those atoms (columns) of $\mathbf{A}$ that are members of the support of $\mathbf{s}_i$.

To find the conditional probabilities at each iteration, we propose the use of a Recurrent Neural Network (RNN) with Long Short-Term Memory (LSTM) cells and a softmax layer on top of it. To find the model parameters, we minimize a cross entropy cost function between the conditional probabilities given by the model and the known probabilities in the training data. The details on how to generate the training data and the training data probabilities are explained in subsequent sections. Please note that this training is done only once. After that, the resulting model is used in the reconstruction algorithm for any test data that has not been observed by the model before. Therefore, the proposed reconstruction algorithm would be almost as fast as the greedy methods. The block diagram of the proposed method is presented in Fig. 4.1 and Fig. 4.2. We will explain these figures in detail in subsequent sections.

To the best of our knowledge, this is the first model-based method in MMV sparse reconstruction that is based on a deep learning bottom up approach. Similar to all deep learning methods, it has the important feature of learning the structure of $\mathbf{S}$ from the raw data automatically. Although it is based on a greedy method that selects subsets that are not necessarily optimal, we experimentally show that by using a properly trained model and only one layer of LSTM, the proposed method significantly outperforms well known

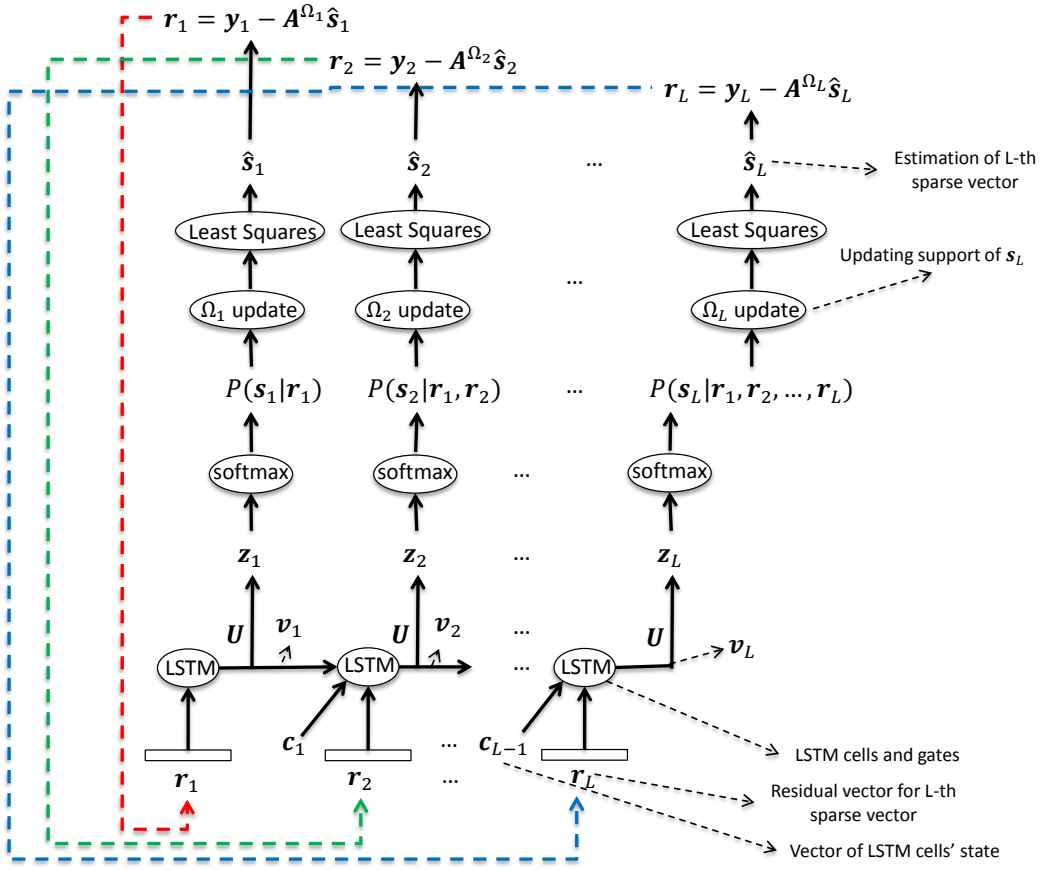


Figure 4.1: Block diagram of the proposed method unfolded over channels.

MMV baselines (e.g., SOMP) as well as the well known Bayesian methods for the MMV problem (e.g., Multitask Bayesian Compressive Sensing (MT-BCS)[72] and Sparse Bayesian Learning for temporally correlated sources (T-SBL)[133]). We show this on two real world datasets. At the end of this chapter, we also present bidirectional version of above LSTM based CS reconstruction method along with a method based on Convolutional Deep Stacking Networks.

We emphasize that the computations carried at the encoder mainly include multiplication by a random matrix. The extra computations are only needed at the decoder. Therefore an important feature of compressive sensing (low power encoding) is preserved.

4.1.3 Related Work

Exploiting data structures besides sparsity for compressive sensing has been extensively studied in the literature [4, 10, 33, 38, 71, 72, 83, 92, 95, 122, 126, 133]. In [10], it has been theoretically shown that using signal models that exploit these structures will result in a decrease in the number of measurements. In [33], a thorough review on CS methods that exploit the structure present in the sparse signal or in the measurements is presented. In [71], a Bayesian framework for CS is presented. This framework uses a prior information about the sparsity of **s** to provide a posterior density function for the entries of **s** (assuming **y** is observed). It

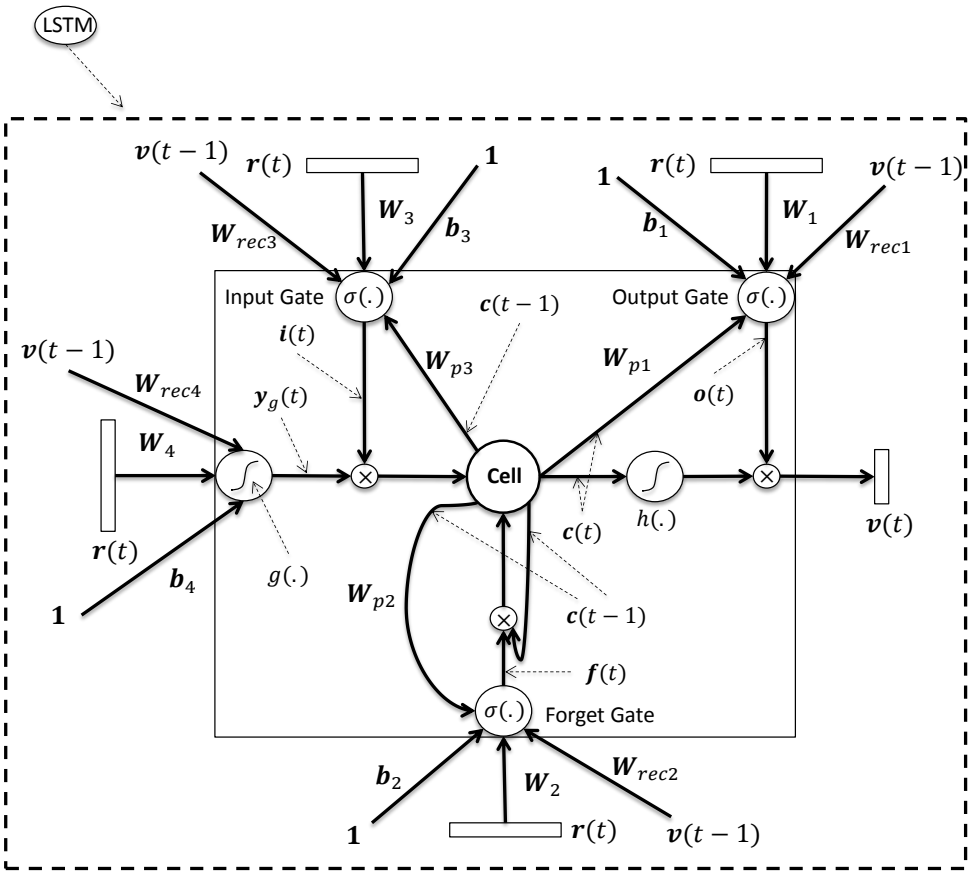


Figure 4.2: Block diagram of the Long Short-Term Memory (LSTM).

then uses a Relevance Vector Machine (RVM) [117] to estimate the entries of the sparse vector. This method is called Bayesian Compressive Sensing (BCS). In [72], a Bayesian framework is presented for the MMV problem. It assumes that the L "tasks" in the MMV problem in (4.4), are not statistically independent. By imposing a shared prior on the L tasks, an empirical method is presented to estimate the hyperparameters and extensions of RVM are used for the inference step. This method is known as Multitask Compressive Sensing (MT-BCS). In [72], it is experimentally shown that the MT-BCS outperforms the method that applies Orthogonal Matching Pursuit (OMP) on each task, the Simultaneous Orthogonal Matching Pursuit (SOMP) method which is a straightforward extension of OMP for the MMV problem, and the method that applies BCS for each task. In [126], the Sparse Bayesian Learning (SBL) [39, 117] is used to solve the MMV problem. It was shown that the global minimum of the proposed method is always the sparsest one. The authors in [133], address the MMV problem when the entries in each row of $\mathbf{S}$ are correlated. An algorithm based on SBL is proposed and it is shown that the proposed algorithm outperforms the mixed norm ($\ell_{1,2}$) optimization as well as the method proposed in [126]. The proposed method is called T-SBL. In [83], a greedy algorithm aided by a neural network is proposed to address the SMV problem in (4.3). The neural network parameters are calculated by solving a regression problem and are used to select the appropriate column of $\mathbf{A}$ at each iteration of OMP. The main modification to OMP is replacing the correlation step with a neural network.

They experimentally show that the proposed method outperforms OMP and ℓ_1 optimization. This method is called Neural Network OMP (NNOMP). In [95], an extension of [83] with a hierarchical Deep Stacking Netowork (DSN) [29] is proposed for the MMV problem. "*The joint sparsity of* $\mathbf{S}$ *is an important assumption in the proposed method*". To train the DSN model, the Restricted Boltzmann Machine (RBM) [59] is used to pre-train DSN and then fine tuning is performed. It has been experimentally shown that this method outperforms SOMP and $\ell_{1,2}$ in the MMV problem. The proposed methods are called Nonlinear Weighted SOMP (NWSOMP) for the one layer model and DSN-WSOMP for the multilayer model. In [92], a feedforward neural network is used to solve the SMV problem as a regression task. Similar to [95] (if we assume that we have only one sparse vector in [95]), a pre-training phase followed by a fine tuning is used. For pre-training, the authors have used Stacked Denoising Auto-encoder (SDA) [124]. Please note that an RBM with Gaussian visible units and binary hidden units (i.e., the one used in [95]) has the same energy function as an auto-encoder with sigmoid hidden units and real valued observations [123]. Therefore the extension of [92] to the MMV problem will give similar performance as that of [95]. In [122], a reconstruction method is proposed for sparse signals whose sparsity patterns change slowly with time. The main idea is to replace Compressive Sensing (CS) on the observation $\mathbf{y}$ with CS on the Least Squares (LS) residuals. LS residuals are calculated using the previous estimation of the support. In [38], a reconstruction method is proposed to recover sparse signals with a sparsity pattern that slowly changes over time. The main idea is to use Sparse Bayesian Learning (SBL) framework. Similar to SBL, a set of hyperparameters are defined to control the sparsity of signals. The main difference is that the prior for each coefficient also involves the coefficients of the adjacent temporal observations. In [4], a CS algorithm is proposed for time-varying sparse signals based on the least absolute shrinkage and selection operator (LASSO). A dynamic LASSO algorithm is proposed for the signals with time-varying amplitudes and support.

4.2 RNN with LSTM Cells

The RNN is a type of deep neural networks [20, 57] that are "deep" in the temporal dimension. It has been used extensively in time sequence modelling [13, 28, 36, 49, 84, 85, 94, 96, 103]. If we look at the sparse vectors (columns) in $\mathbf{S}$ as a sequence, the main idea of using RNN for the MMV problem is to predict the sparsity patterns over different sparse vectors in $\mathbf{S}$.

Although RNN performs sequence modelling in a principled manner, it is generally difficult to learn the long term dependency within the sequence due to the vanishing gradients problem. One of the effective solutions for this problem in RNNs is to employ memory cells instead of neurons that is originally proposed in [62] as Long Short-Term Memory (LSTM). It is further developed in [45] and [46] by adding forget gate and peephole connections to the architecture.

We use the architecture of LSTM illustrated in Fig. 4.2 for the proposed sequence modelling method for the MMV problem. In this figure, $\mathbf{i}(t)$, $\mathbf{f}(t)$,$\mathbf{o}(t)$,$\mathbf{c}(t)$ are input gate, forget gate, output gate and cell state vector respectively, $\mathbf{W}_{p1}$, $\mathbf{W}_{p2}$ and $\mathbf{W}_{p3}$ are peephole connections, $\mathbf{W}_i$, $\mathbf{W}_{reci}$ and $\mathbf{b}_i$, $i = 1,2,3,4$ are input connections, recurrent connections and bias values, respectively, $g(\cdot)$ and $h(\cdot)$ are $\tanh(\cdot)$ function and $\sigma(\cdot)$ is the sigmoid function. We use this architecture to find $\mathbf{v}$ for each channel and then use the proposed method in Fig. 4.1 to find the entries that have a higher probability of being non-zero. Considering Fig. 4.2,

the forward pass for LSTM model is as follows:

$$\begin{aligned}
\mathbf{y}_g(t) &= g(\mathbf{W}_4\mathbf{r}(t) + \mathbf{W}_{\text{rec4}}\mathbf{v}(t-1) + \mathbf{b}_4) \\
\mathbf{i}(t) &= \sigma(\mathbf{W}_3\mathbf{r}(t) + \mathbf{W}_{\text{rec3}}\mathbf{v}(t-1) + \mathbf{W}_{p3}\mathbf{c}(t-1) + \mathbf{b}_3) \\
\mathbf{f}(t) &= \sigma(\mathbf{W}_2\mathbf{r}(t) + \mathbf{W}_{\text{rec2}}\mathbf{v}(t-1) + \mathbf{W}_{p2}\mathbf{c}(t-1) + \mathbf{b}_2) \\
\mathbf{c}(t) &= \mathbf{f}(t) \circ \mathbf{c}(t-1) + \mathbf{i}(t) \circ \mathbf{y}_g(t) \\
\mathbf{o}(t) &= \sigma(\mathbf{W}_1\mathbf{r}(t) + \mathbf{W}_{\text{rec1}}\mathbf{v}(t-1) + \mathbf{W}_{p1}\mathbf{c}(t) + \mathbf{b}_1) \\
\mathbf{v}(t) &= \mathbf{o}(t) \circ h(\mathbf{c}(t)),
\end{aligned} \tag{4.6}$$

where $\circ$ denotes the Hadamard (element-wise) product.

Summary of notations used in Fig. 4.2 is as follows:

- "t": Stands for the time index in the sequence. For example, if we have 4 residual vectors of four different channels, we can show them as $\mathbf{r}(t)$, $t = 1,2,3,4$.
- "1": is a scalar
- "$\mathbf{W}_{reci}$, $i = 1,2,3,4$": Recurrent weight matrices of dimension $ncell \times ncell$ where $ncell$ is the number of cells in LSTM.
- "$\mathbf{W}_i$, $i = 1,2,3,4$": Input weight matrices of dimension $M \times ncell$ where M is the number of random measurements in compressive sensing. These matrices map the residual vectors to feature space.
- "$\mathbf{b}_i$, $i = 1,2,3,4$": Bias vectors of size $ncell \times 1$.
- "$\mathbf{W}_{pi}$, $i = 1,2,3$": Peephole connections of dimension $ncell \times ncell$.
- "$\mathbf{v}(t)$, $t = 1,2,\ldots,L$": Output of the cells. Vector of size $ncell \times 1$. L is the number of channels in the MMV problem.
- "$\mathbf{i}(t), \mathbf{o}(t), \mathbf{y}_g(t)$, $t = 1,2,\ldots,L$": Input gates, output gates and inputs before gating respectively. Vector of size $ncell \times 1$.
- "$g(\cdot)$ and $h(\cdot)$": $\tanh(\cdot)$ function.
- "$\sigma(\cdot)$": Sigmoid function.

4.3 Proposed Method

4.3.1 High Level Picture

The summary of the proposed method is presented in Fig. 4.1. We initialize the residual vector, $\mathbf{r}$, for each channel by the measurement vector, $\mathbf{y}$, of that channel. These residual vectors serve as the input to the LSTM model that captures features of the residual vectors using input weight matrices ($\mathbf{W}_1$,$\mathbf{W}_2$,$\mathbf{W}_3$,$\mathbf{W}_4$) as well

as the dependency among the residual vectors using recurrent weight matrices ($\mathbf{W}_{rec1}$,$\mathbf{W}_{rec2}$,$\mathbf{W}_{rec3}$,$\mathbf{W}_{rec4}$) and the central memory unit shown in Fig. 4.2. A transformation matrix $\mathbf{U}$ is then used to transform, $\mathbf{v} \in \Re^{ncell \times 1}$, the output of each memory cell after gating, into the sparse vectors space, i.e., $\mathbf{z} \in \Re^{N \times 1}$. "*ncell*" is the number of cells in the LSTM model. Then a softmax layer is used for each channel to find the probability of each entry of each sparse vector being non-zero. For example, for channel 1, the j-th output of the softmax layer is

$$P(s_1(j)|\mathbf{r}_1) = \frac{e^{z(j)}}{\sum_{k=1}^{N} e^{z(k)}}. \tag{4.7}$$

Then for each channel, the entry with the maximum probability value is selected and added to the support set of that channel. After that, given the new support set, the following least squares problem is solved to find an estimate of the sparse vector for the j-th channel

$$\hat{\mathbf{s}}_j = \underset{\mathbf{s}_j}{\operatorname{argmin}} \|\mathbf{y}_j - \mathbf{A}^{\Omega_j}\mathbf{s}_j\|_2^2. \tag{4.8}$$

Using $\hat{\mathbf{s}}_j$, the new residual value for the j-th channel is calculated as follows:

$$\mathbf{r}_j = \mathbf{y}_j - \mathbf{A}^{\Omega_j}\hat{\mathbf{s}}_j. \tag{4.9}$$

This residual serves as the input to the LSTM model at the next iteration of the algorithm. The stopping criteria for the algorithm is when the residual values are small enough or when it has performed N iterations where N is the dimension of the sparse vector. Since we have used LSTM cells for the proposed method, we call it LSTM-CS algorithm. The pseudo-code of the proposed method is presented in Algorithm 2.

Algorithm 2 Distributed Compressive Sensing using Long Short-Term Memory (LSTM-CS)

Inputs: CS measurement matrix $\mathbf{A} \in \Re^{M \times N}$; matrix of measurements $\mathbf{Y} \in \Re^{M \times L}$; minimum ℓ_2 norm of residual matrix "*resMin*" as stopping criterion; Trained "*lstm*" model
Output: Matrix of sparse vectors $\hat{\mathbf{S}} \in \Re^{N \times L}$
Initialization: $\hat{\mathbf{S}} = 0$; $j = 1$; $i = 1$; $\Omega = \emptyset$; $\mathbf{R} = \mathbf{Y}$.

procedure LSTM-CS($\mathbf{A}$,$\mathbf{Y}$,*lstm*)
  while $i \leq N$ or $\|\mathbf{R}\|_2 \leq resMin$ **do**
    $i \leftarrow i+1$
    for $j = 1 \rightarrow L$ **do**
      $\mathbf{R}(:,j)_i \leftarrow \frac{\mathbf{R}(:,j)_{i-1}}{max(|\mathbf{R}(:,j)_{i-1}|)}$
      $\mathbf{v}_j \leftarrow lstm(\mathbf{R}(:,j)_i, \mathbf{v}_{j-1}, \mathbf{c}_{j-1})$ ▷ LSTM
      $\mathbf{z}_j \leftarrow \mathbf{U}\mathbf{v}_j$
      $\mathbf{c} \leftarrow softmax(\mathbf{z}_j)$
      $idx \leftarrow Support(max(\mathbf{c}))$
      $\Omega_i \leftarrow \Omega_{i-1} \cup idx$
      $\hat{\mathbf{S}}^{\Omega_i}(:,j) \leftarrow (\mathbf{A}^{\Omega_i})^{\dagger}\mathbf{Y}(:,j)$ ▷ Least Squares
      $\hat{\mathbf{S}}^{\Omega_i^C}(:,j) \leftarrow 0$
      $\mathbf{R}(:,j)_i \leftarrow \mathbf{Y}(:,j) - \mathbf{A}^{\Omega_i}\hat{\mathbf{S}}^{\Omega_i}(:,j)$
    end for
  end while
end procedure

We continue by explaining how the training data is prepared from off-line dataset and then we present the details of the learning method. Please note that all the computations explained in the subsequent two sections are performed only once and they do not affect the run time of the proposed solver in Fig. 4.1. It is

almost as fast as greedy algorithms in sparse reconstruction.

4.3.2 Training Data Generation

The main idea of the proposed method is to look at the sparse reconstruction problem as a two step task: a classification as the first step and a subsequent least squares as the second step. In the classification step, the aim is to find the atom of the dictionary, i.e., the column of $\mathbf{A}$, that is most relevant to the given residual of the current channel and the residuals of the previous channels. Therefore we need a set of residual vectors and their corresponding sparse vectors for supervised training. Since the training data and $\mathbf{A}$ are given, we can imitate the steps explained in the previous section to generate the residuals. This means that, given a sparse vector $\mathbf{s}$ with k non-zero entries, we calculate $\mathbf{y}$ using (4.3). Then we find the entry that has the maximum value in $\mathbf{s}$ and set it to zero. Assume that the index of this entry is k_0. This gives us a new sparse vector with $k-1$ non-zero entries. Then we calculate the residual vector from

$$\mathbf{r} = \mathbf{y} - \mathbf{a}_{k_0} s(k_0), \tag{4.10}$$

where $\mathbf{a}_{k_0}$ is the k_0-th column of $\mathbf{A}$ and $s(k_0)$ is the k_0-th entry of $\mathbf{s}$. It is obvious that this residual value is because of not having the remaining $k-1$ non-zero entries of $\mathbf{s}$. From these remaining $k-1$ non-zero entries, the second largest value of $\mathbf{s}$ has the main contribution to $\mathbf{r}$ in (4.10). Therefore, we use $\mathbf{r}$ to predict the location of the second largest value of $\mathbf{s}$. Assume that the index of the second largest value of $\mathbf{s}$ is k_1. We define $\mathbf{s}_0$ *as a one hot vector that has value* 1 *at* k_1*-th entry and zero at other entries*. Therefore, the training pair is $(\mathbf{r}, \mathbf{s}_0)$.

Now we set the k_1-th entry of $\mathbf{s}$ to zero. This gives us a new sparse vector with $k-2$ non-zero entries. Then we calculate the new residual vector from

$$\mathbf{r} = \mathbf{y} - [\mathbf{a}_{k_0}, \mathbf{a}_{k_1}][s(k_0), s(k_1)]^T. \tag{4.11}$$

We use the residual in (4.11) to predict the location of the third largest value in $\mathbf{s}$. Assume that the index of the third largest value of $\mathbf{s}$ is k_2. We define $\mathbf{s}_0$ *as a one hot vector that has value* 1 *at* k_2*-th entry and zero at other entries*. Therefore, the new training pair is $(\mathbf{r}, \mathbf{s}_0)$.

The above procedure is continued upto the point that $\mathbf{s}$ does not have any non-zero entry. Then the same procedure is used for the next training sample. This gives us training samples for one channel. Then the same procedure is used for the next channel in $\mathbf{S}$. Since the number of non-zero entries, k, is not known in advance, we assume a maximum number of non-zero entries per channel for training data generation.

4.3.3 Learning Method

To calculate the parameters of the proposed model, i.e., $\mathbf{W}_1, \mathbf{W}_2, \mathbf{W}_3, \mathbf{W}_4, \mathbf{W}_{rec1}, \mathbf{W}_{rec2}, \mathbf{W}_{rec3}, \mathbf{W}_{rec4}, \mathbf{W}_{p1}, \mathbf{W}_{p2}, \mathbf{W}_{p3}$, $\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3, \mathbf{b}_4$ in Fig. 4.2 and transformation matrix $\mathbf{U}$ in Fig.4.1, we minimize a cross entropy cost function over the training data. Assuming $\mathbf{s}$ is the output vector of the softmax layer given by the model in Fig. 4.1 (output of the softmax layer is represented as conditional probabilities in Fig. 4.1) and $\mathbf{s}_0$ is the one hot

vector explained in the previous section, the following optimization problem is solved:

$$L(\Lambda) = \min_{\Lambda} \left\{ \sum_{i=1}^{nB} \sum_{r=1}^{Bsize} \sum_{\tau=1}^{L} \sum_{j=1}^{N} L_{r,i,\tau,j}(\Lambda) \right\}$$
$$L_{r,i,\tau,j}(\Lambda) = -s_{0,r,i,\tau}(j) log(s_{r,i,\tau}(j)), \tag{4.12}$$

where nB is the number of mini-batches in the training data, $Bsize$ is the number of training data pairs, $(\mathbf{r}, \mathbf{s}_0)$, in each mini-batch, L is the number of channels in the MMV problem, i.e., number of columns of $\mathbf{S}$, and N is the length of vector $\mathbf{s}$ and $\mathbf{s}_0$. Λ denotes the collection of the model parameters that includes $\mathbf{W}_1$, $\mathbf{W}_2$, $\mathbf{W}_3$, $\mathbf{W}_4$, $\mathbf{W}_{rec1}$, $\mathbf{W}_{rec2}$, $\mathbf{W}_{rec3}$, $\mathbf{W}_{rec4}$, $\mathbf{W}_{p1}$, $\mathbf{W}_{p2}$, $\mathbf{W}_{p3}$, $\mathbf{b}_1$, $\mathbf{b}_2$, $\mathbf{b}_3$ and $\mathbf{b}_4$ in Fig. 4.2 and $\mathbf{U}$ in Fig. 4.1.

To solve the optimization problem in (4.12), we use Backpropagation through time (BPTT) with Nesterov method. The update equations for parameter Λ at epoch k are as follows:

$$\triangle\Lambda_k = \Lambda_k - \Lambda_{k-1}$$
$$\triangle\Lambda_k = \mu_{k-1}\triangle\Lambda_{k-1} - \varepsilon_{k-1}\nabla L(\Lambda_{k-1} + \mu_{k-1}\triangle\Lambda_{k-1}), \tag{4.13}$$

where $\nabla L(\cdot)$ is the gradient of the cost function in (4.12), ε is the learning rate and μ_k is a momentum parameter determined by the scheduling scheme used for training. Above equations are equivalent to Nesterov method in [93]. To see why, please refer to appendix A.1 of [115] where the Nesterov method is derived as a momentum method. The gradient of the cost function, $\nabla L(\Lambda)$, is

$$\nabla L(\Lambda) = \underbrace{\sum_{i=1}^{nB} \sum_{r=1}^{Bsize} \sum_{\tau=1}^{L} \sum_{j=1}^{N} \frac{\partial L_{r,i,\tau,j}(\Lambda)}{\partial \Lambda}}_{\text{one large update}}. \tag{4.14}$$

As it is obvious from (4.14), since we have unfolded the LSTM over channels in $\mathbf{S}$, we fold it back when we want to calculate gradients over the whole sequence of channels.

$\frac{\partial L_{r,i,\tau,j}(\Lambda)}{\partial \Lambda}$ in (4.14) and error signals for different parameters of the proposed model that are necessary for training are presented in section 4.5.

We have used mini-batch training to accelerate training and one large update instead of incremental updates during back propagation through time. To resolve the gradient explosion problem we have used gradient clipping. To accelerate the convergence, we have used Nesterov method [93] and found it effective in training the proposed model for the MMV problem.

We have used a simple yet effective scheduling for μ_k in (4.13), in the first and last 10% of all parameter updates $\mu_k = 0.9$ and for the other 80% of all parameter updates $\mu_k = 0.995$. We have used a fixed step size for training LSTM. Please note that since we are using mini-batch training, all parameters are updated for each mini-batch in (4.14).

A summary of training method for LSTM-CS is presented in Algorithm 3.

Although the training method and derivatives in section 4.5 are presented for all parameters in LSTM, in the implementation ,we have removed peephole connections and forget gates. Since length of each sequence,

Algorithm 3 Training the proposed model for Distributed Compressive Sensing

Inputs: Fixed step size "ε", Scheduling for "μ", Gradient clip threshold "th_G", Maximum number of Epochs "$nEpoch$", Total number of training pairs in each mini-batch "$Bsize$", Number of channels for the MMV problem "L".
Outputs: LSTM-CS trained model for distributed compressive sensing "Λ".
Initialization: Set all parameters in Λ to small random numbers, $i = 0$, $k = 1$.
procedure LSTM-CS(Λ)
  while $i \leq nEpoch$ **do**
    for "first minibatch" $\rightarrow$ "last minibatch" **do**
      $r \leftarrow 1$
      while $r \leq Bsize$ **do**
        Compute $\sum_{\tau=1}^{L} \frac{\partial L_{r,\tau}}{\partial \Lambda_k}$
        ▷ use (4.23) to (4.54) in section 4.5
        $r \leftarrow r+1$
      end while
      Compute $\nabla L(\Lambda_k) \leftarrow$ "sum above terms over r"
      if $\nabla L(\Lambda_k) > th_G$ **then**
        $\nabla L(\Lambda_k) \leftarrow th_G$
        ▷ For each entry of the gradient matrix $\nabla L(\Lambda_k)$
      end if
      Compute $\triangle \Lambda_k$ ▷ use (4.13)
      Update: $\Lambda_k \leftarrow \triangle \Lambda_k + \Lambda_{k-1}$
      $k \leftarrow k+1$
    end for
    $i \leftarrow i+1$
  end while
end procedure

i.e., the number of columns in $\mathbf{S}$, is known in advance, we set state of each cell to zero in the beginning of a new sequence. Therefore, forget gates are not a great help here. Also, as long as the order of columns in $\mathbf{S}$ is kept, the precise timing in the sequence is not of great concern, therefore, peephole connections are not that important as well. Removing peephole connections and forget gate will also help to have less training time, i.e., less number of parameters need to be tuned during training.

4.4 Experimental Results and Discussion

We have performed the experiments on two real world datasets, the first is the MNIST dataset of handwritten digits [78] and the second is three different classes of images from natural image dataset of Microsoft Research in Cambridge [102].

In this section, we would like to answer the following questions: (i) How is the performance of different reconstruction algorithms for the MMV problem, including the proposed method, when different channels, i.e., different columns in $\mathbf{S}$, have different sparsity patterns? (ii) Does the proposed method perform well enough when there is correlation among different sparse vectors? E.g., when sparse vectors are DCT or Wavelet transform of different blocks of an image? (iii) How fast is the proposed method compared to other reconstruction algorithms for the MMV problem? (iv) How robust is the proposed method to noise?

For all the results presented in this section, the reconstruction error is defined as

$$NMSE = \frac{\|\hat{\mathbf{S}} - \mathbf{S}\|}{\|\mathbf{S}\|}, \tag{4.15}$$

where $\mathbf{S}$ is the actual sparse matrix and $\hat{\mathbf{S}}$ is the recovered sparse matrix from random measurements by the reconstruction algorithm. The machine used to perform the experiments has an Intel(R) Core(TM) i7 CPU

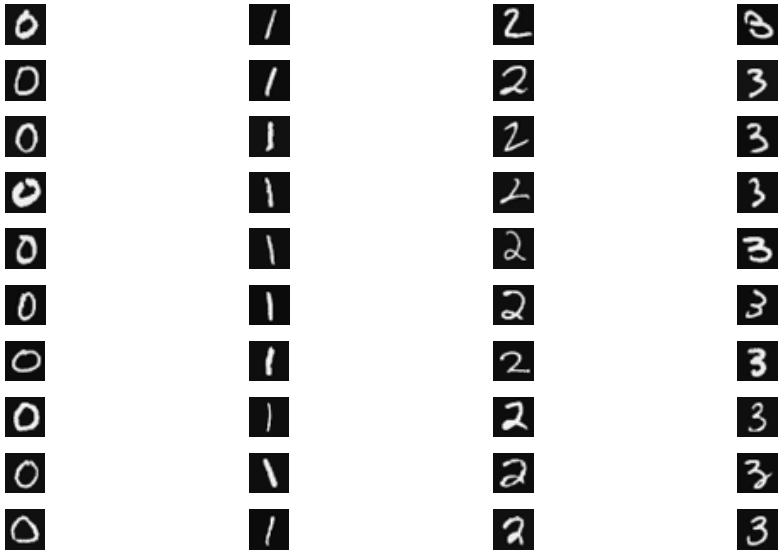


Figure 4.3: Randomly selected images for test from MNIST dataset. The first channel encodes digit zero, the second channel encodes digit one and so on.

with clock 2.93 GHz and with 16 GB RAM.

4.4.1 MNIST Dataset

MNIST is a dataset of handwritten digits where the images of the digits are normalized in size and centred so that we have fixed size images. The task is to simultaneously encode 4 images each of size 24×24, i.e., we have 4 channels and $L = 4$ in (4.4). The encoder is a typical compressive sensing encoder, i.e., a randomly generated matrix $\mathbf{A}$. We have normalized each column of $\mathbf{A}$ to have unit norm. Since the images are already sparse, i.e., have a few number of non-zero pixels, no transform, Ψ in (4.2), is used. To simulate the measurement noise, we have added a Gaussian noise with standard deviation 0.005 to the measurement matrix $\mathbf{Y}$ in (4.4). This results in measurements with signal to noise ratio (SNR) of approximately $46dB$. We have divided each image into four 12×12 blocks. This means that the length of each sparse vector is $N = 144$. We have taken 50% random measurements from each sparse vector, i.e., $M = 72$. After receiving and reconstructing all blocks at the decoder, we compute the reconstruction error defined in (4.15) for the full image. We have randomly selected 10 images for each digit from the set $\{0, 1, 2, 3\}$, i.e., 40 images in total for the test. This means that the first column of $\mathbf{S}$ is an image of digit 0, the second column is an image of digit 1, the third column is an image of digit 2 and the fourth column is an image of digit 3. Test images are represented in Fig. 4.3.

We have compared the performance of the proposed reconstruction algorithm (LSTM-CS) with 7 reconstruction methods for the MMV problem. These methods are:

- Simultaneous Orthogonal Matching Pursuit (SOMP) which is a well known baseline for the MMV

problem.

- Bayesian Compressive Sensing (BCS)[71] applied independently on each channel. For the BCS method we set the initial noise variance of i-th channel to the value suggested by the authors, i.e., $std(\mathbf{y}_i)^2/100$ where $i \in \{1,2,3,4\}$ and $std(.)$ calculates the standard deviation. We set the threshold for stopping the algorithm to 10^{-8}.

- Multitask Compressive Sensing (MT-BCS) [72] which takes into account the statistical dependency of different channels. For MT-BCS we set the parameters of the Gamma prior on noise variance to $a = 100/0.1$ and $b = 1$ which are the values suggested by the authors. We set the stopping threshold to 10^{-8} as well.

- Sparse Bayesian Learning for Temporally correlated sources (T-SBL) [133] which exploits correlation among different sources in the MMV problem. For T-SBL, we used the default values proposed by the authors.

- Nonlinear Weighted SOMP (NWSOMP) [95] which solves a regression problem to help the SOMP algorithm with prior knowledge from training data. For NWSOMP, during training, we used one layer, 512 neurons and 25 epochs of parameters update.

- Compressive Sensing on Least Squares Residual (LSCS) [122] where no explicit joint sparsity assumption is made in the design of the method. For LSCS, we used $sigma0 = cc * (1/3) * sqrt(Sav/m)$ suggested by the authors where m is the number of measurements and $Sav = 16$ as suggested by the author. We tried a range of different values of cc and got the best results with $cc = 0.1$. We also set $sigsys = 1$, $siginit = 3$ and $lambdap = 4$ as suggested by the author.

- The method proposed in [38**?**] and referred to as PCSBL-GAMP where sparse Bayesian learning is used to design the method and no explicit joint sparsity assumption is made. For PCSBL-GAMP, we used $beta = 1$, $Pattern = 2$ because we need the coupling among the sparse vectors, i.e., left and right coupling, maximum number of iterations equal to $maxiter = 400$, and $C = 1e0$ as suggested by the authors for the noisy case.

For LSTM-CS, during training, we used one layer, 512 cells and 25 epochs of parameter updates. We used only 200 images for the training set. The training set does not include any of the 40 images used for test. To monitor and prevent overfitting, we used 3 images per channel as the validation set and we used early stopping if necessary. Please note that the images used for validation were not used in the training set or in the test set. Results are presented in Fig. 4.4.

In Fig. 4.4, the vertical axis is the $NMSE$ defined in (4.15) and horizontal axis is the number of non-zero entries in the sparse vector. The number of measurements, M, is fixed to 72. Each point on the curves in Fig. 4.4 is the average of $NMSE$ over 40 reconstructed test images at the decoder.

For the MNIST dataset, we observe from Fig. 4.4 that LSTM-CS significantly outperforms the reconstruction algorithms for the MMV problem discussed in this chapter. One important reason for this is that

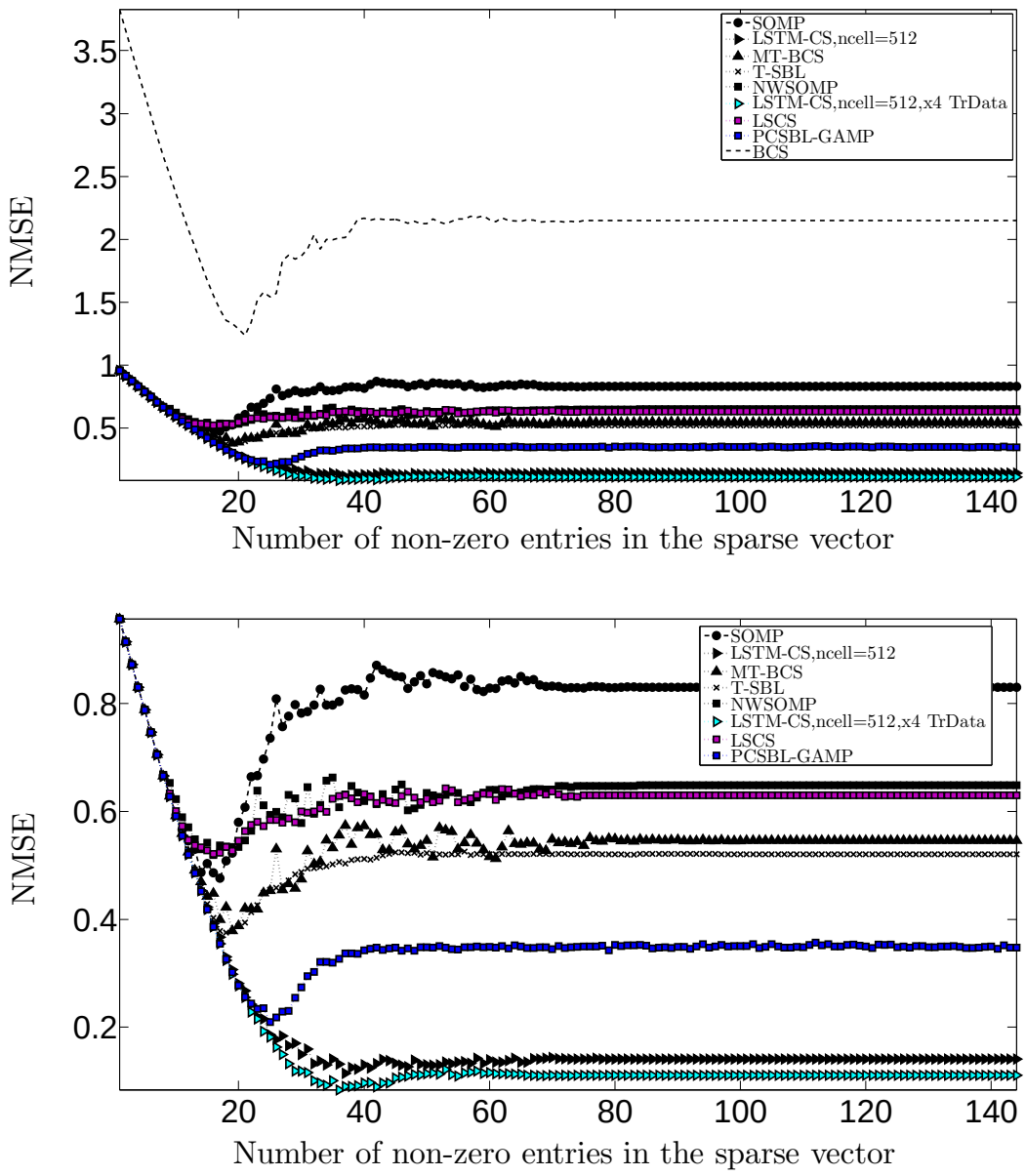


Figure 4.4: Comparison of different MMV reconstruction algorithms for MNIST dataset. Bottom figure is the same as top figure without results of BCS algorithm to make the difference among different algorithms more visible. In this experiment $M = 72$ and $N = 144$.

existing MMV solvers rely on the joint sparsity in $\mathbf{S}$, while the proposed method does not rely on this assumption. Another reason is that the structure of each sparse vector is effectively captured by LSTM. The reconstructed images using different MMV reconstruction algorithms for 4 test images are presented in Fig. 4.5. An interesting observation from Fig. 4.5 is that the accuracy of reconstruction depends on the complexity of the sparsity pattern. For example when the sparsity pattern is simple, e.g., image of digit 1 in Fig. 4.5, all the algorithms perform well. But when the sparsity pattern is more complex, e.g., image of digit 0 in Fig. 4.5, then their reconstruction accuracy degrades significantly.

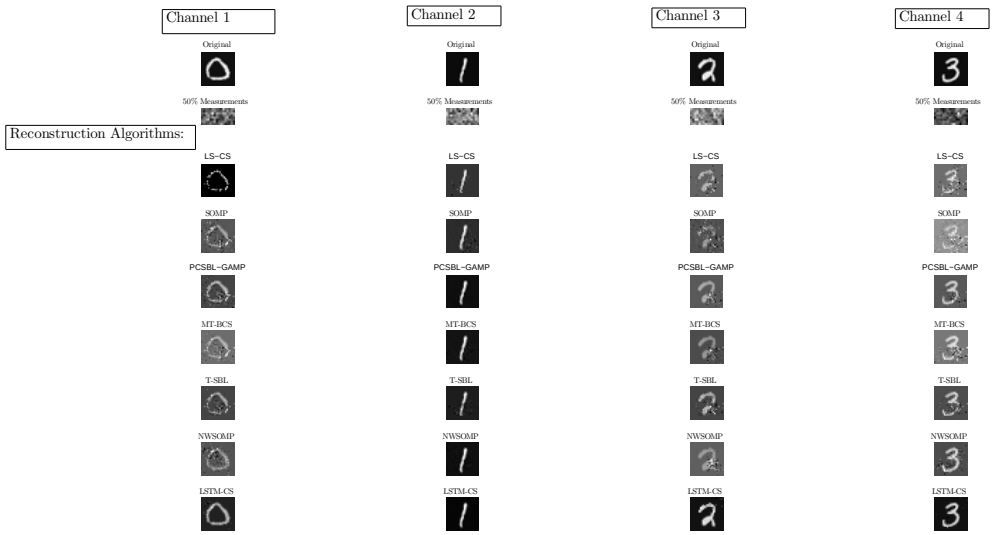


Figure 4.5: Reconstructed images using different MMV reconstruction algorithms for 4 images of the MNIST dataset. First row are original images, **S**, second row are measurement matrices, **Y**, third row are reconstructed images using LS-CS, fourth row are reconstructed images using SOMP, fifth row using PCSBL-GAMP, sixth row using MT-BCS, seventh row using T-SBL, eighth row using NWSOMP and the last row are reconstructed images using the proposed LSTM-CS method.

We have repeated the experiments on the MNIST dataset with 25% random measurements, i.e., $M = 36$. The results are presented in Fig. 4.6. We trained 4 different LSTM models for this experiment. The first one is the same model used for previous experiment ($m = 72$). In the second model, we increased the number of cells in the LSTM model from 512 to 1024. In the third and fourth models, we used 2 times and 4 times more training data respectively. The rest of the experiments' settings was similar to the settings described before. As observed from these results, by investing more on training a good LSTM model, LSTM-CS method performs better.

All the results presented so far are for noisy measurements where an additive Gaussian noise with standard deviation 0.005 is used ($SNR \simeq 46dB$). To evaluate the stability of the proposed LSTM-CS method to noise, and compare it with other methods discussed in this chapter, an experiment was performed using the following range of noise standard deviations:

$$\sigma = \{0.5, 0.2, 0.1, 0.05, 0.01, 0.005\}, \tag{4.16}$$

where σ is the standard deviation of noise. This approximately corresponds to

$$SNR = \{6\,dB, 14\,dB, 20\,dB, 26\,dB, 40\,dB, 46\,dB\}. \tag{4.17}$$

We used the same experimental settings explained above. Results are presented in Fig. 4.7.

As observed from the results, in very noisy environment, i.e., SNR = 6 dB, performance of MT-BCS

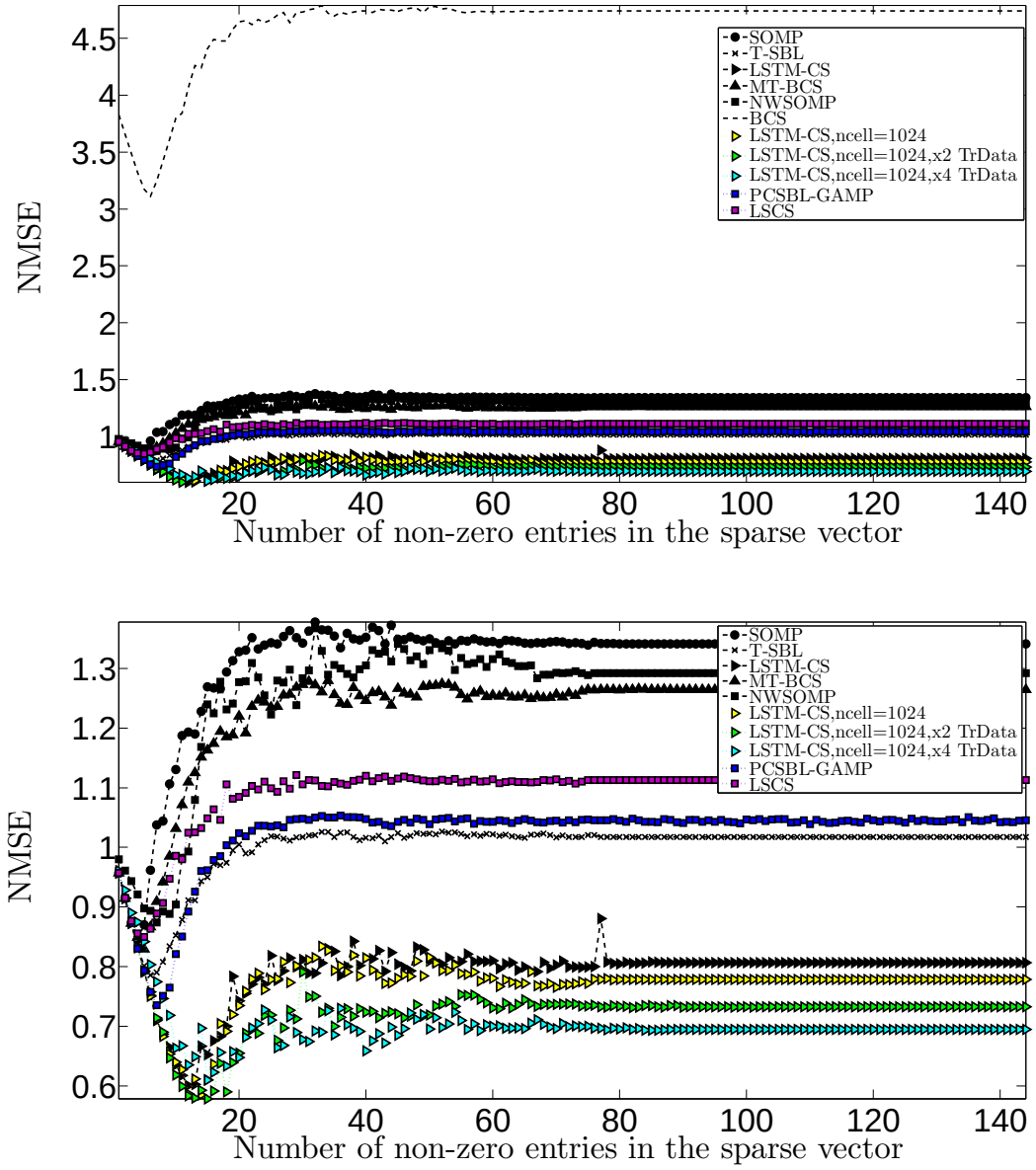
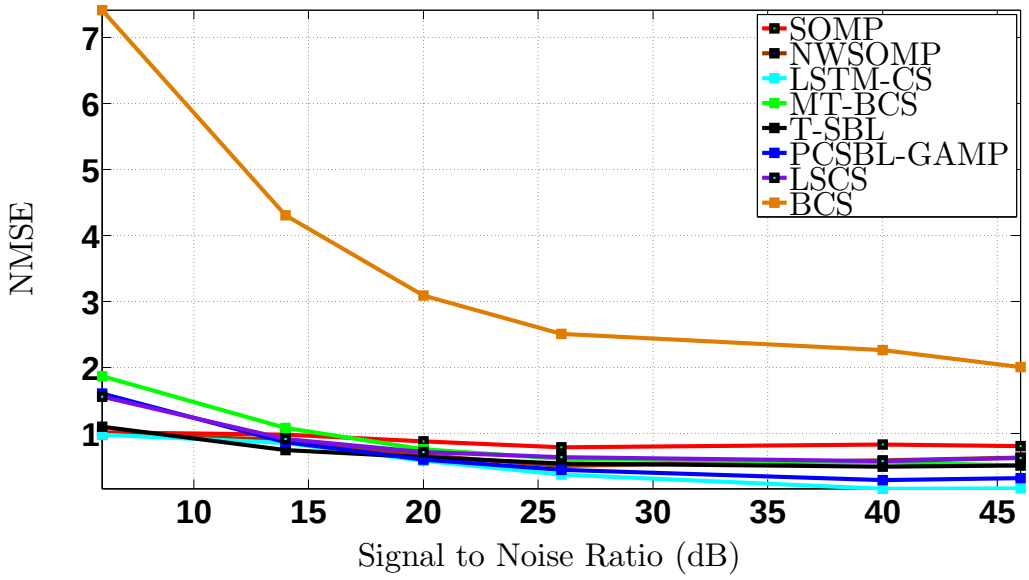
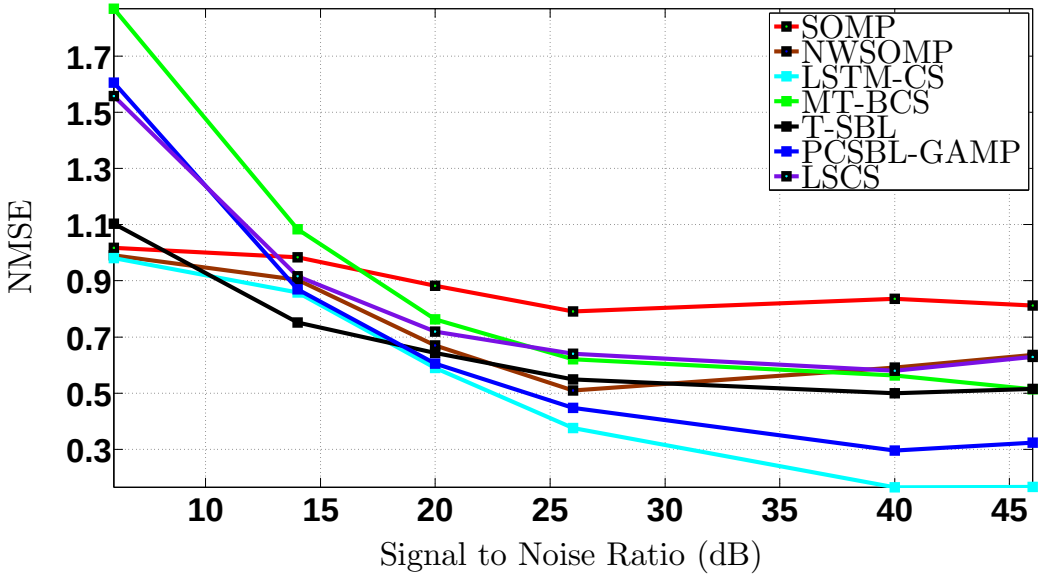


Figure 4.6: Comparison of different MMV reconstruction algorithms for MNIST dataset. Bottom figure is the same as top figure without results of BCS algorithm to make the difference among different algorithms more visible. In this experiment $M = 36$ and $N = 144$.

, LSCS and PCSBL-GAMP degrades significantly while T-SBL , NWSOMP and LSTM-CS (proposed in this chapter) methods show less severe degradation. In very low noise environment, i.e., SNR = 46 dB, performance of LSTM-CS, trained with just 512 cells and 200 training images, is better than other methods. In medium noise environment, i.e., SNR = 20 dB and SNR = 26 dB, performance of LSTM-CS, T-SBL and PCSBL-GAMP are close (although LSTM-CS is slightly better). Please note that the performance of LSTM-CS can be further improved by using a better architecture (e.g., more cells, more training data or more layers) as explained previously.



(a) Results for all Methods.



(b) Results without BCS method for a more clear visibility.

Figure 4.7: Reconstruction performance of the methods discussed in this chapter for different noise levels.

To present the phase transition diagram of solvers, we used a simple LSTM-CS solver that uses 512 cells and just 200 training images. The performance was evaluated over the following values of $\frac{m}{n}$ where n is the number of entries in each sparse vector and m is the number of measurements per channel:

$$\frac{m}{n} = \{0.10, 0.15, 0.20, 0.25, 0.30, 0.35, 0.40, 0.45, 0.50\}. \tag{4.18}$$

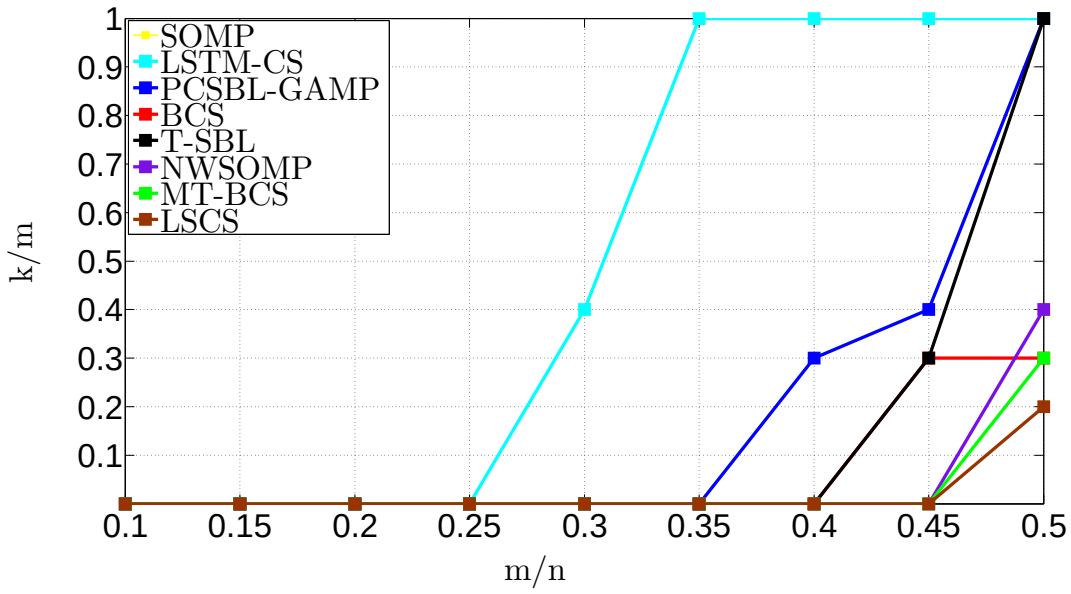


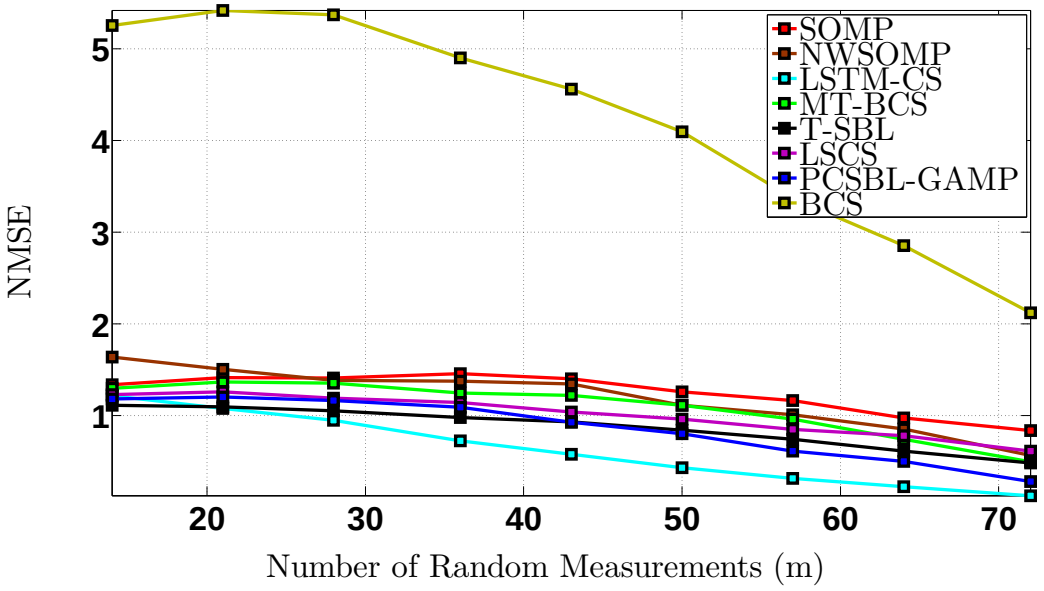
Figure 4.8: Phase transition diagram for different methods on MNIST dataset where 90% perfect recovery is considered. Assuming a perfect recovery condition of NMSE ≤ 0.6 for this dataset. "n" is the number of entries in each sparse vector, "m" is the number of random measurements and "k" is the number of non-zero entries in each sparse vector.

For this experiment, we randomly selected 50 images per channel from MNIST dataset. Since we have $L = 4$ channels, and each image is of size 24×24, and each image has 4 blocks of 12×12 pixels, in total we will have $50 \times 4 \times 4 = 800$ sparse vectors. Considering Fig. 4.4, NMSE of most solvers is about 0.6. Therefore we set the following as the condition for perfect recovery: if more than 90% of test images are reconstructed with an NMSE of 0.6 or less, count that test image as perfectly recovered. We did this for each $\frac{m}{n}$ in (4.18). Results are presented in Fig. 4.8. Results presented in Fig.4.8 shows the reconstruction performance improvement when LSTM-CS method is used.

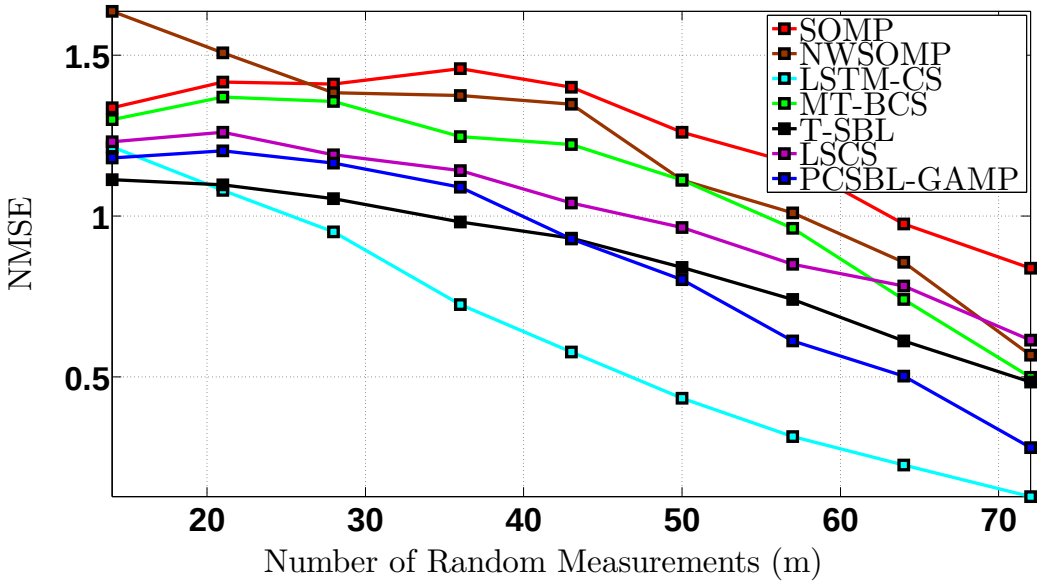
We also present the performance of LSTM-CS for different number of random measurements. We used the set of random measurements in (4.18) with $n = 144$. We used an LSTM with 512 cells and 400 training images. The settings for all other methods was similar to the one described before. Results are presented in Fig. 4.9. As observed from Fig. 4.9, using LSTM-CS method improves the reconstruction performance compared to other methods discussed in this chapter.

4.4.2 Natural Images Dataset

For experiments on natural images we used the MSR Cambridge dataset [102]. Ten randomly selected test images belonging to three classes of this dataset are used for experiments. The images are shown in Fig. 4.10. We have used 64×64 images. Each image is divided into 8×8 blocks. After reconstructing all blocks of an image in the decoder, the *NMSE* for the reconstructed image is calculated. The task is to simultaneously encode 4 blocks ($L = 4$) of an image and reconstruct them in the decoder. This means that $\mathbf{S}$ in (4.4) has 4 columns each one having $N = 64$ entries. We used 50% measurements, i.e., $\mathbf{Y}$ in (4.4) have 4



(a) Results for all Methods.



(b) Results without BCS method for a more clear visibility.

Figure 4.9: Comparison of different MMV reconstruction algorithms for different number of random measurements for MNIST dataset. In this experiment n = 144.

columns each one having $M = 32$ entries.

We have compared the performance of the proposed algorithm, LSTM-CS, with SOMP, T-SBL, MT-BCS and NWSOMP. We have not included results of applying BCS per channel due its weak performance compared to other methods (this is shown in the experiments for MNIST dataset). We have used the same setting as the settings for the MNIST dataset for different methods which is explained in the previous section. The only differences here are: (i) For each class of images, we have used just 55 images for training set and

Figure 4.10: Randomly selected natural images from three different classes used for test. The first row are "buildings", the second row are "cows" and the third row are "flowers".

5 images for validation set which do not include any of 10 images used for test. (ii) We have used 15 epochs for training LSTM-CS which is enough for this dataset, compared to 25 epochs for the MNIST dataset. The experiments were performed for two popular transforms, DCT and Wavelet, for all aforementioned reconstruction algorithms. For the wavelet transform we used Haar wavelet transform with 3 levels of decomposition. Results for DCT transform are presented in Fig. 4.19. Results for wavelet transform are presented in Fig. 4.20.

To conclude the experiments section, the CPU time for different reconstruction algorithms for the MMV problem discussed in this chapter are presented in Fig. 4.23. Each point on the curves in Fig. 4.23 is the time spent to reconstruct each sparse vector averaged over all the 8×8 blocks in 10 test images. We observe from this figure that the proposed algorithm is almost as fast as greedy algorithms. Please note that there is a faster version of T-SBL that is known as TMSBL. It will improve the CPU time of T-SBL but it is still slower than other reconstruction methods.

4.5 Expressions for the Gradients

In this section we present the final gradient expressions that are necessary to use for training the proposed model for the MMV problem. Due to lack of space, we omit the presentation of full derivations of these gradients.

Starting with the cost function in (4.12), we use the Nesterov method described in (4.13) to update LSTM-CS model parameters. Here, Λ is one of the weight matrices or bias vectors $\{\mathbf{W}_1, \mathbf{W}_2, \mathbf{W}_3, \mathbf{W}_4, \mathbf{W}_{rec1}, \mathbf{W}_{rec2}, \mathbf{W}_{rec3}, \mathbf{W}_{rec4}, \mathbf{W}_{p1}, \mathbf{W}_{p2}, \mathbf{W}_{p3}, \mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3, \mathbf{b}_4\}$ in the LSTM-CS architecture. The general format of the gradient of the cost function, $\nabla L(\Lambda)$, is the same as (4.14). To calculate $\frac{\partial L_{r,i,\tau}(\Lambda)}{\partial \Lambda}$ from (4.12) we have

$$\frac{\partial L_{r,i,\tau}(\Lambda)}{\partial \Lambda} = -\sum_{j=1}^{N} s_{0,r,i,\tau}(j) \frac{\partial log(s_{r,i,\tau}(j))}{\partial \Lambda}. \tag{4.19}$$

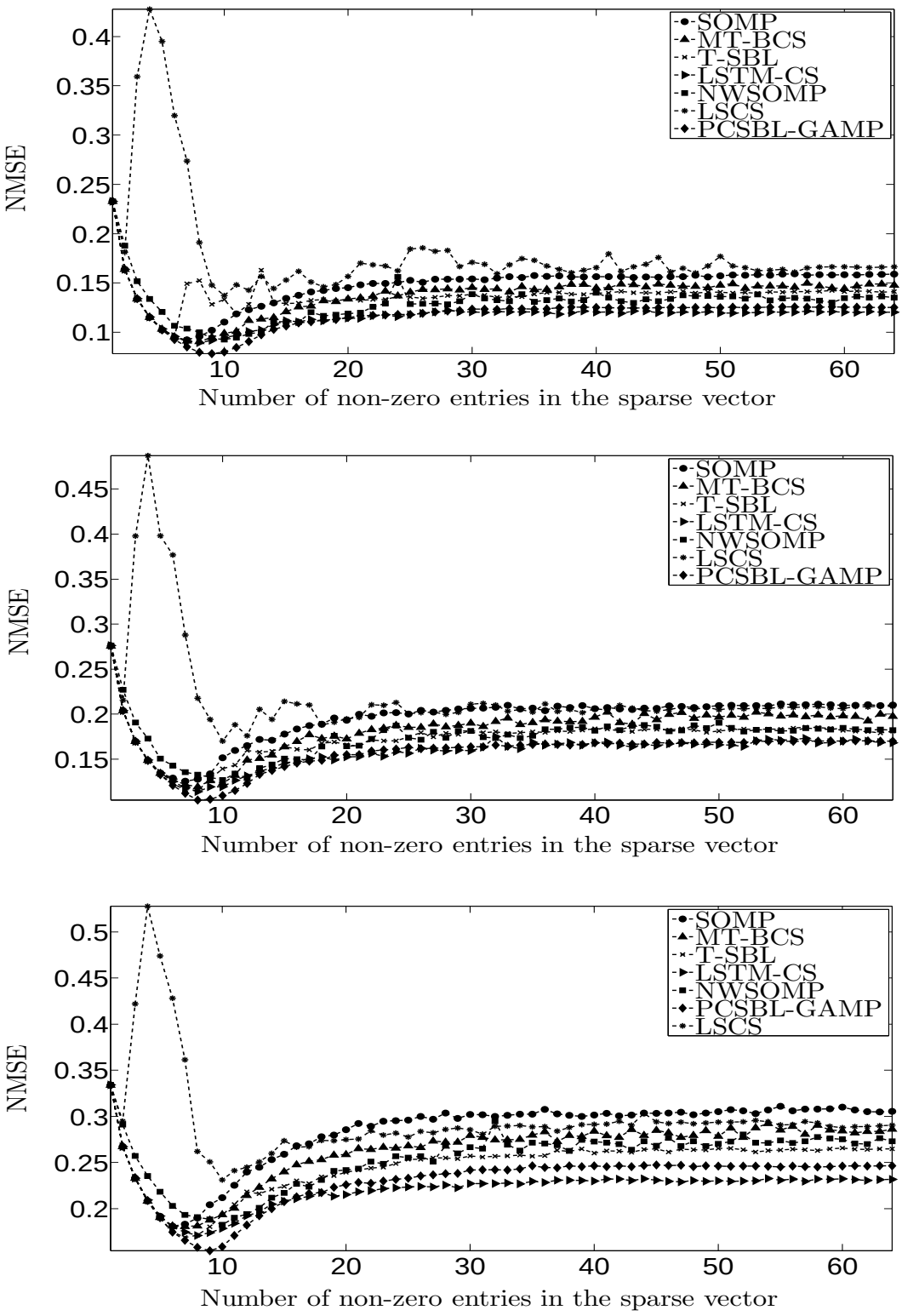


Figure 4.11: Comparison of different MMV reconstruction algorithms for natural image dataset using DCT transform and just one layer for LSTM model in LSTM-CS. Image classes from top to bottom respectively: buildings, cows and flowers.

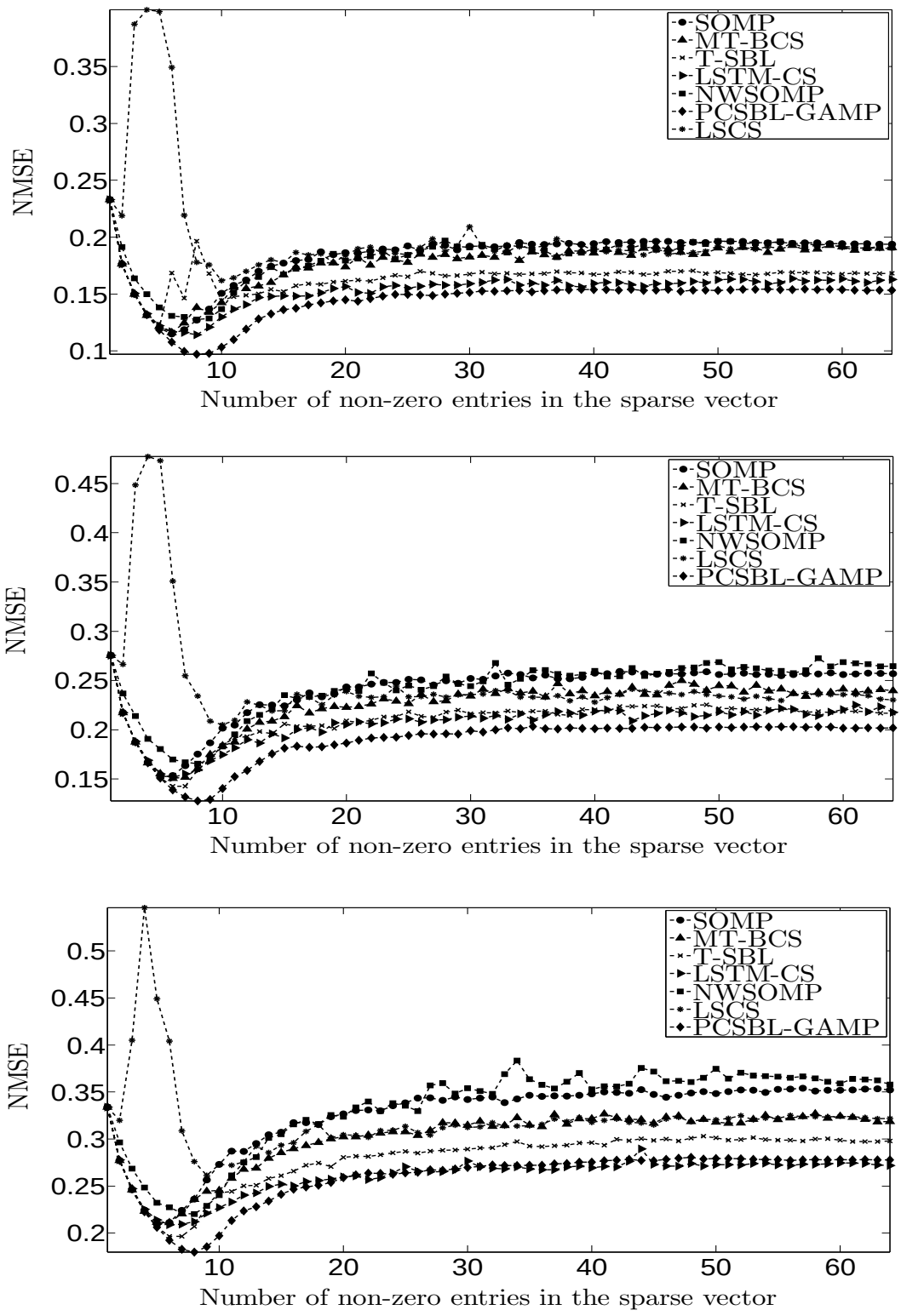


Figure 4.12: Comparison of different MMV reconstruction algorithms for natural image dataset using Wavelet transform and just one layer for LSTM model in LSTM-CS. Image classes from top to bottom respectively: buildings, cows and flowers.

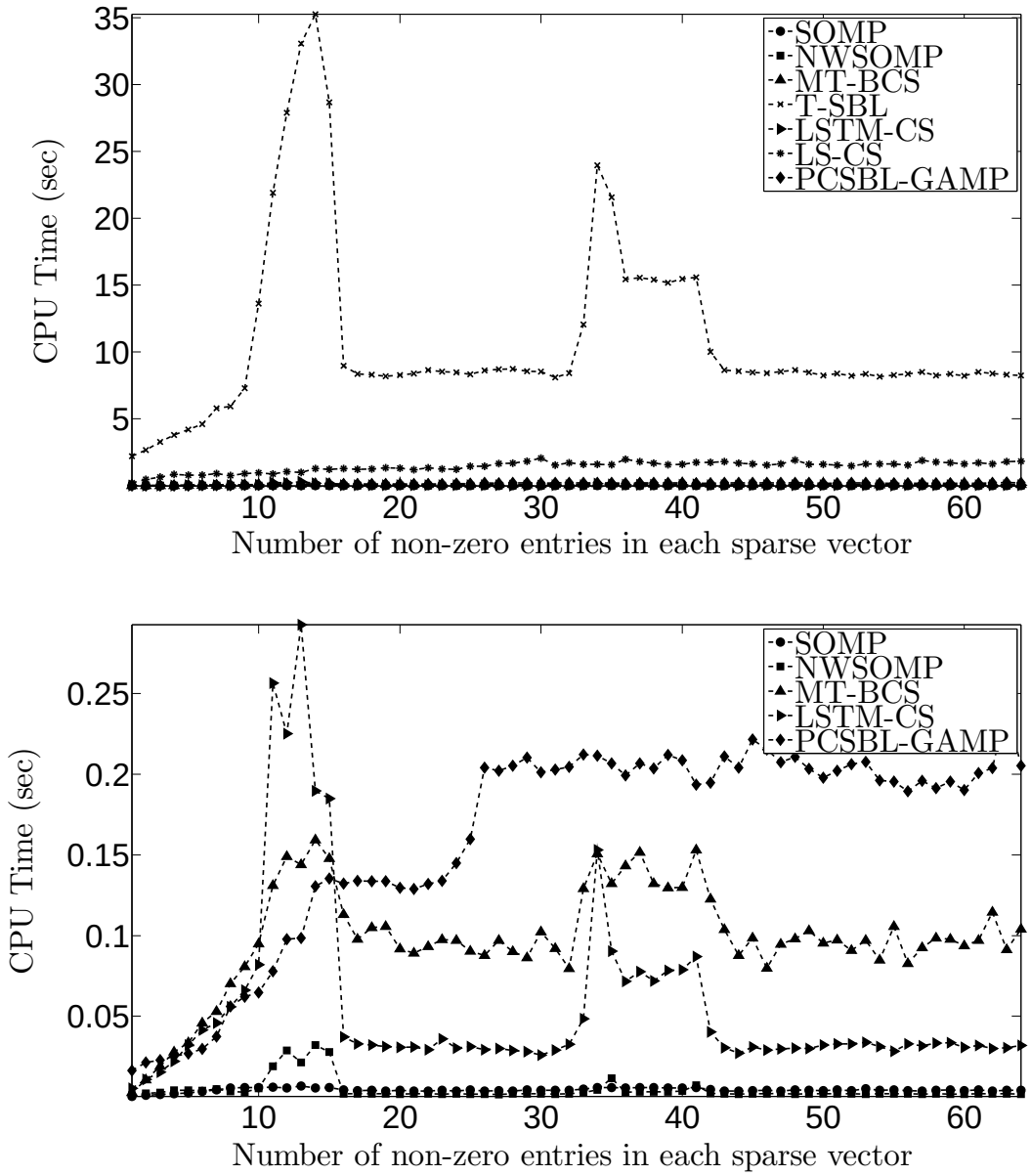


Figure 4.13: CPU time for different MMV reconstruction algorithms. These times are for the experiment using DCT transform for 10 test images from the building class. The bottom figure is the same as top figure but without T-SBL and LS-CS to make the difference among different methods more clear.

After a straightforward derivation of derivatives we will have

$$\frac{\partial L_{r,i,\tau}(\Lambda)}{\partial \Lambda} = (\beta \mathbf{s}_{r,i,\tau} - \mathbf{s}_{0,r,i,\tau}) \frac{\partial \mathbf{z}_\tau}{\partial \Lambda}, \tag{4.20}$$

where $\mathbf{z}_\tau$ is the vector $\mathbf{z}$ for τ-th channel in Fig. 4.1 and β is a scalar defined as

$$\beta = \sum_{j=1}^{N} s_{0,r,i,\tau}(j). \tag{4.21}$$

Since during training data generation we have generated one hot vectors for $\mathbf{s}_0$, β always equals to 1. Since we are looking at different channels as a sequence, for a more clear presentation we show any vector corresponding to t-th channel with (t) instead of index τ. For example, $\mathbf{z}_\tau$ is represented by $\mathbf{z}(t)$.

Since $\mathbf{z}(t) = \mathbf{U}\mathbf{v}(t)$ we have

$$\frac{\partial \mathbf{z}(t)}{\partial \Lambda} = \mathbf{U}^T \frac{\partial \mathbf{v}(t)}{\partial \Lambda}. \tag{4.22}$$

Combining (4.20), (4.21) and (4.22) we will have

$$\frac{\partial L_{r,i,t}(\Lambda)}{\partial \Lambda} = \mathbf{U}^T (\mathbf{s}_{r,i}(t) - \mathbf{s}_{0,r,i}(t)) \frac{\partial \mathbf{v}(t)}{\partial \Lambda}. \tag{4.23}$$

Starting from "$t = L$"-th channel, we define $\mathbf{e}(t)$ as

$$\mathbf{e}(t) = \mathbf{U}^T (\mathbf{s}_{r,i}(t) - \mathbf{s}_{0,r,i}(t)). \tag{4.24}$$

The expressions for the gradients for different parameters of LSTM-CS model are presented in the subsequent sections. We omit the subscripts r and i for simplicity of presentation. Please note that the final value of the gradient is sum of gradient values over the mini-batch samples and number of channels as represented by summations in (4.14).

4.5.1 Output Weights U

$$\frac{\partial L_t}{\partial \mathbf{U}} = (\mathbf{s}(t) - \mathbf{s}_0(t)).\mathbf{v}(t)^T. \tag{4.25}$$

4.5.2 Output Gate

For recurrent connections we have

$$\frac{\partial L_t}{\partial \mathbf{W}_{rec1}} = \delta^{rec1}(t).\mathbf{v}(t-1)^T, \tag{4.26}$$

where

$$\delta^{rec1}(t) = \mathbf{o}(t) \circ (1 - \mathbf{o}(t)) \circ h(\mathbf{c}(t)) \circ \mathbf{e}(t). \tag{4.27}$$

For input connections, $\mathbf{W}_1$, and peephole connections, $\mathbf{W}_{p1}$, we will have

$$\frac{\partial L_t}{\partial \mathbf{W}_1} = \delta^{rec1}(t).\mathbf{r}(t)^T, \tag{4.28}$$

$$\frac{\partial L_t}{\partial \mathbf{W}_{p1}} = \delta^{rec1}(t).\mathbf{c}(t)^T. \tag{4.29}$$

The derivative for output gate bias values will be

$$\frac{\partial L_t}{\partial \mathbf{b}_1} = \delta^{rec1}(t). \tag{4.30}$$

4.5.3 Input Gate

For the recurrent connections we have

$$\frac{\partial L_t}{\partial \mathbf{W}_{rec3}} = diag(\delta^{rec3}(t)).\frac{\partial \mathbf{c}(t)}{\partial \mathbf{W}_{rec3}}, \tag{4.31}$$

where

$$\begin{aligned}
&\delta^{rec3}(t) = (1-h(\mathbf{c}(t))) \circ (1+h(\mathbf{c}(t))) \circ \mathbf{o}(t) \circ \mathbf{e}(t) \\
&\frac{\partial \mathbf{c}(t)}{\partial \mathbf{W}_{rec3}} = diag(\mathbf{f}(t)).\frac{\partial \mathbf{c}(t-1)}{\partial \mathbf{W}_{rec3}} + \mathbf{b}_i(t).\mathbf{v}(t-1)^T \\
&\mathbf{b}_i(t) = \mathbf{y}_g(t) \circ \mathbf{i}(t) \circ (1-\mathbf{i}(t)).
\end{aligned} \tag{4.32}$$

For the input connections we will have the following:

$$\frac{\partial L_t}{\partial \mathbf{W}_3} = diag(\delta^{rec3}(t)).\frac{\partial \mathbf{c}(t)}{\partial \mathbf{W}_3}, \tag{4.33}$$

where

$$\frac{\partial \mathbf{c}(t)}{\partial \mathbf{W}_3} = diag(\mathbf{f}(t)).\frac{\partial \mathbf{c}(t-1)}{\partial \mathbf{W}_3} + \mathbf{b}_i(t).\mathbf{r}(t)^T. \tag{4.34}$$

For the peephole connections we will have

$$\frac{\partial L_t}{\partial \mathbf{W}_{p3}} = diag(\delta_y^{rec3}(t)).\frac{\partial \mathbf{c}(t)}{\partial \mathbf{W}_{p3}}, \tag{4.35}$$

where

$$\frac{\partial \mathbf{c}(t)}{\partial \mathbf{W}_{p3}} = diag(\mathbf{f}(t)).\frac{\partial \mathbf{c}(t-1)}{\partial \mathbf{W}_{p3}} + \mathbf{b}_i(t).\mathbf{c}(t-1)^T. \tag{4.36}$$

For bias values, $\mathbf{b}_3$, we will have

$$\frac{\partial L_t}{\partial \mathbf{b}_3} = diag(\delta^{rec3}(t)).\frac{\partial \mathbf{c}(t)}{\partial \mathbf{b}_3}, \tag{4.37}$$

where

$$\frac{\partial \mathbf{c}(t)}{\partial \mathbf{b}_3} = diag(\mathbf{f}(t)).\frac{\partial \mathbf{c}(t-1)}{\partial \mathbf{b}_3} + \mathbf{b}_i(t). \tag{4.38}$$

4.5.4 Forget Gate

For the recurrent connections we will have

$$\frac{\partial L_t}{\partial \mathbf{W}_{rec2}} = diag(\delta^{rec2}(t)).\frac{\partial \mathbf{c}(t)}{\partial \mathbf{W}_{rec2}}, \tag{4.39}$$

where

$$\begin{aligned}
\delta^{rec2}(t) &= (1-h(\mathbf{c}(t))) \circ (1+h(\mathbf{c}(t))) \circ \mathbf{o}(t) \circ \mathbf{e}(t) \\
\frac{\partial \mathbf{c}(t)}{\partial \mathbf{W}_{rec2}} &= diag(\mathbf{f}(t)).\frac{\partial \mathbf{c}(t-1)}{\partial \mathbf{W}_{rec2}} + \mathbf{b}_f(t).\mathbf{v}(t-1)^T \\
\mathbf{b}_f(t) &= \mathbf{c}(t-1) \circ \mathbf{f}(t) \circ (1-\mathbf{f}(t)).
\end{aligned} \tag{4.40}$$

For input connections to forget gate we will have

$$\frac{\partial L_t}{\partial \mathbf{W}_2} = diag(\delta^{rec2}(t)).\frac{\partial \mathbf{c}(t)}{\partial \mathbf{W}_2}, \tag{4.41}$$

where

$$\frac{\partial \mathbf{c}(t)}{\partial \mathbf{W}_2} = diag(\mathbf{f}(t)).\frac{\partial \mathbf{c}(t-1)}{\partial \mathbf{W}_2} + \mathbf{b}_f(t).\mathbf{r}(t)^T. \tag{4.42}$$

For peephole connections we have

$$\frac{\partial L_t}{\partial \mathbf{W}_{p2}} = diag(\delta^{rec2}(t)).\frac{\partial \mathbf{c}(t)}{\partial \mathbf{W}_{p2}}, \tag{4.43}$$

where

$$\frac{\partial \mathbf{c}(t)}{\partial \mathbf{W}_{p2}} = diag(\mathbf{f}(t)).\frac{\partial \mathbf{c}(t-1)}{\partial \mathbf{W}_{p2}} + \mathbf{b}_f(t).\mathbf{c}(t-1)^T. \tag{4.44}$$

For forget gate's bias values we will have

$$\frac{\partial L_t}{\partial \mathbf{b}_2} = diag(\delta^{rec2}(t)).\frac{\partial \mathbf{c}(t)}{\partial \mathbf{b}_2}, \tag{4.45}$$

where

$$\frac{\partial \mathbf{c}(t)}{\partial \mathbf{b}_2} = diag(\mathbf{f}(t)).\frac{\partial \mathbf{c}(t-1)}{\partial \mathbf{b}_3} + \mathbf{b}_f(t). \tag{4.46}$$

4.5.5 Input without Gating ($\mathbf{y}_g(t)$)

For recurrent connections we will have

$$\frac{\partial L_t}{\partial \mathbf{W}_{rec4}} = diag(\delta^{rec4}(t)).\frac{\partial \mathbf{c}(t)}{\partial \mathbf{W}_{rec4}}, \tag{4.47}$$

where

$$\delta^{rec4}(t) = (1-h(\mathbf{c}(t))) \circ (1+h(\mathbf{c}(t))) \circ \mathbf{o}(t) \circ \mathbf{e}(t)$$
$$\frac{\partial \mathbf{c}(t)}{\partial \mathbf{W}_{rec4}} = diag(\mathbf{f}(t)).\frac{\partial \mathbf{c}(t-1)}{\partial \mathbf{W}_{rec4}} + \mathbf{b}_g(t).\mathbf{v}(t-1)^T$$
$$\mathbf{b}_g(t) = \mathbf{i}(t) \circ (1-\mathbf{y}_g(t)) \circ (1+\mathbf{y}_g(t)). \tag{4.48}$$

For input connections we have

$$\frac{\partial L_t}{\partial \mathbf{W}_4} = diag(\delta^{rec4}(t)).\frac{\partial \mathbf{c}(t)}{\partial \mathbf{W}_4}, \tag{4.49}$$

where

$$\frac{\partial \mathbf{c}(t)}{\partial \mathbf{W}_4} = diag(\mathbf{f}(t)).\frac{\partial \mathbf{c}(t-1)}{\partial \mathbf{W}_4} + \mathbf{b}_g(t).\mathbf{r}(t)^T. \tag{4.50}$$

For bias values we will have

$$\frac{\partial L_t}{\partial \mathbf{b}_4} = diag(\delta^{rec4}(t)).\frac{\partial \mathbf{c}(t)}{\partial \mathbf{b}_4}, \tag{4.51}$$

where

$$\frac{\partial \mathbf{c}(t)}{\partial \mathbf{b}_4} = diag(\mathbf{f}(t)).\frac{\partial \mathbf{c}(t-1)}{\partial \mathbf{b}_4} + \mathbf{b}_g(t). \tag{4.52}$$

4.5.6 Error Signal Backpropagation

Error signals are back propagated through time using following equations:

$$\delta^{rec1}(t-1) = [\mathbf{o}(t-1) \circ (1-\mathbf{o}(t-1)) \circ h(\mathbf{c}(t-1))]$$
$$\circ [\mathbf{W}^T_{rec1}.\delta^{rec1}(t) + \mathbf{e}(t-1)], \tag{4.53}$$

$$\delta^{rec_i}(t-1) = [(1-h(\mathbf{c}(t-1))) \circ (1+h(\mathbf{c}(t-1)))$$
$$\circ \mathbf{o}(t-1)] \circ [\mathbf{W}^T_{rec_i}.\delta^{rec_i}(t) + \mathbf{e}(t-1)],$$
$$for \quad i \in \{2,3,4\}. \tag{4.54}$$

4.6 Bidirectional LSTM for MMV Problem

In this section, we present bidirectional version of LSTM-CS. Given the fact that in the MMV problem, usually all measurement vectors are given, we can use both past and future information about the structure of the sparse vectors in $\mathbf{S}$. This means that we can perform support prediction for a given column of $\mathbf{S}$, based on both previous columns and future columns. This calls for a bidirectional learning architecture. We use bidirectional LSTM to address this problem. We experimentally show that the proposed method in this section outperforms [72, 95, 133] and [97].

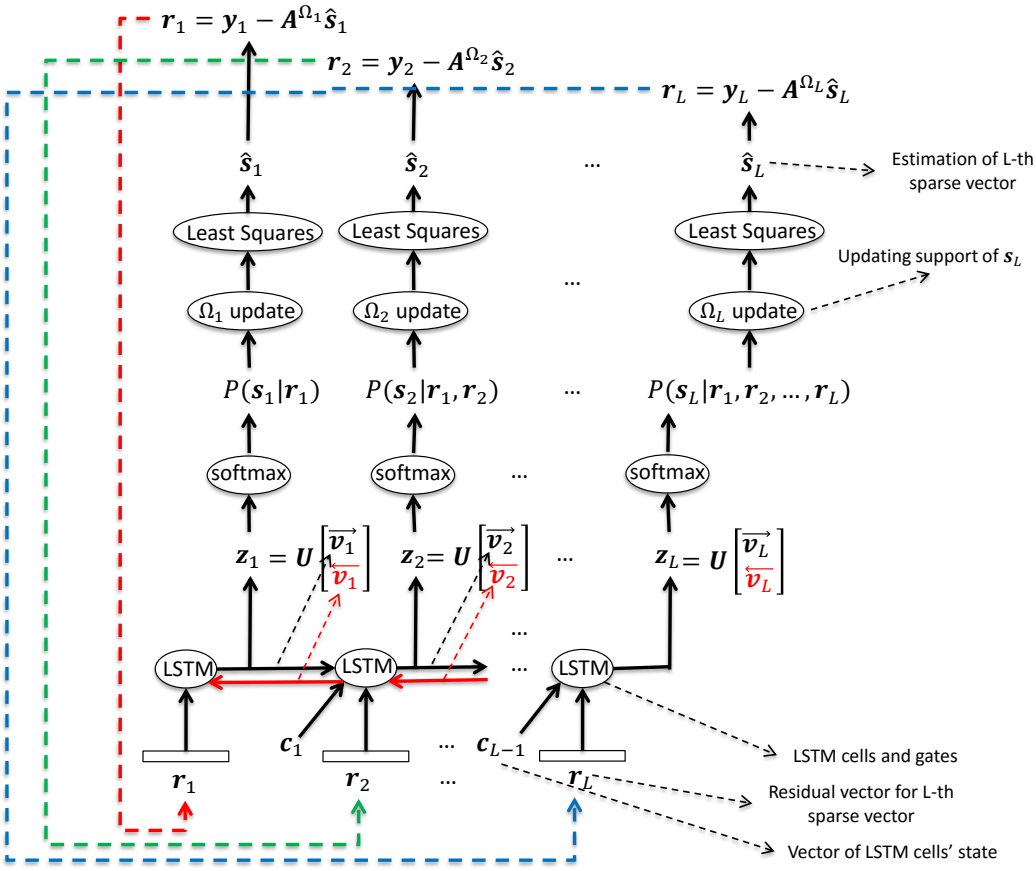


Figure 4.14: Block diagram of the proposed bidirectional LSTM-CS method unfolded over channels.

Method presented in this section is similar to LSTM-CS with the difference that for support prediction, we use both left to right and right to left predictions. This is shown in Fig. 4.14. If we detect the classes (the non-zero entries) one by one, we can use the remaining residuals after finding each class (non-zero entry) as an appropriate input to a deep model for feature extraction. The extracted feature vectors are represented by $\{\overrightarrow{\mathbf{v}_1}, \overrightarrow{\mathbf{v}_2}, \dots, \overrightarrow{\mathbf{v}_L}\}$ for left-to-right model and $\{\overleftarrow{\mathbf{v}_1}, \overleftarrow{\mathbf{v}_2}, \dots, \overleftarrow{\mathbf{v}_L}\}$ for right-to-left model in Fig. 4.14. Feature vectors from both directions can be concatenated for the next step of the algorithm.

Similar to previous sections, we initialize the residual vector, $\mathbf{r}$, for each channel by the measurement vector, $\mathbf{y}$, of that channel. These residual vectors, represented as $\mathbf{r}_1, \mathbf{r}_2, \dots, \mathbf{r}_L$ in Fig. 4.14, serve as the inputs to the bidirectional LSTM model. The bidirectional LSTM model captures features of the residual vectors using input weight matrices ($\mathbf{W}_1$,$\mathbf{W}_2$,$\mathbf{W}_3$,$\mathbf{W}_4$) as well as the dependency among the residual vectors using recurrent weight matrices ($\mathbf{W}_{rec1}$,$\mathbf{W}_{rec2}$,$\mathbf{W}_{rec3}$,$\mathbf{W}_{rec4}$) and the central memory units in left-to-right and right-to-left LSTMs. A transformation matrix $\mathbf{U}$ is then used to transform, $[\overrightarrow{\mathbf{v}}, \overleftarrow{\mathbf{v}}]^T \in \Re^{2ncell \times 1}$, the outputs of each memory cell after gating for both left-to-right and right-to-left models, into the sparse vectors space, i.e., $\mathbf{z} \in \Re^{N \times 1}$. "*ncell*" is the number of cells in the LSTM model. Then a softmax layer is used for each channel to find the probability of each entry of each sparse vector being non-zero. For example, for channel

1, the j-th output of the softmax layer is

$$P(s_1(j)|\mathbf{r}_1) = \frac{e^{z(j)}}{\sum_{k=1}^{N} e^{z(k)}}. \tag{4.55}$$

Rest of the method is similar to LSTM-CS described in previous sections.

To evaluate performance of bidirectional LSTM-CS, we performed experiments on three different classes of images from a natural image dataset described in section 3.5. We used the same *NMSE* defined in (4.15) to evaluate the performance of bidirectional LSTM-CS.

Five randomly selected test images from each class (flowers, buildings, cows) were used for test experiments. For each class of images, we used just 55 images for training set and 5 images for validation set which do not include any of 5 images used for test. We re-sized images to 128×128 images. Each image was divided into 8×8 blocks. After reconstructing all blocks of an image in the decoder, the *NMSE* for the reconstructed image was calculated. The task was to simultaneously encode 4 blocks ($L = 4$) of an image and reconstruct them in the decoder. This meant that $\mathbf{S}$ in (4.4) had 4 columns each one having $N = 64$ entries. We used 40% measurements, thus $\mathbf{Y}$ in (4.4) had 4 columns each one having $M = 25$ entries. The encoder was a typical compressive sensing encoder, i.e., a randomly generated matrix $\mathbf{A}$. We normalized each column of $\mathbf{A}$ to have unit norm. To simulate the measurement noise, we added a Gaussian noise with standard deviation 0.005 to the measurement matrix $\mathbf{Y}$ in (4.4).

We compared the performance of the proposed algorithm, BLSTM-CS, with SOMP [120], MT-BCS[72], T-SBL[133], NWSOMP[95] and LSTM-CS[97]. For MT-BCS we set the parameters of the Gamma prior on noise variance to $a = 100/0.1$ and $b = 1$ which are the values suggested by the authors. We set the stopping threshold to 10^{-8} as well. For T-SBL, we used the default values proposed by the authors. We used T-MSBL which is a faster version of T-SBL. For NWSOMP, during training, we used one layer, 512 neurons and 15 epochs of parameters update. The experiments were performed for two popular transforms, DCT and Wavelet, for all of the above reconstruction algorithms. For the wavelet transform, we used Haar wavelet transform with 3 levels of decomposition. For both LSTM-CS and BLSTM-CS, we used a small model with 16 cells. For NWSOMP we used 3 layers and 512 neurons per layers. We present results for one class of images, buildings. The results from the other two classes of images are similar to what is presented here. To monitor and prevent overfitting, we used 5 images per channel as the validation set and we used early stopping if necessary. Please note that the images used for validation were not used in the training set or in the test set. Results for DCT transform and wavelet transform are shown in Fig. 4.15.

As observed in Fig.4.15, BLSTM-CS outperforms the other methods discussed in this section for different sparsity levels. To evaluate run time of different methods, considering the fact that all methods are implemented in MATLAB and run on the same machine, the CPU time shown in Fig.4.15 demonstrates that the proposed method is faster than the Bayesian methods discussed in this section and is almost as fast as the greedy method SOMP.

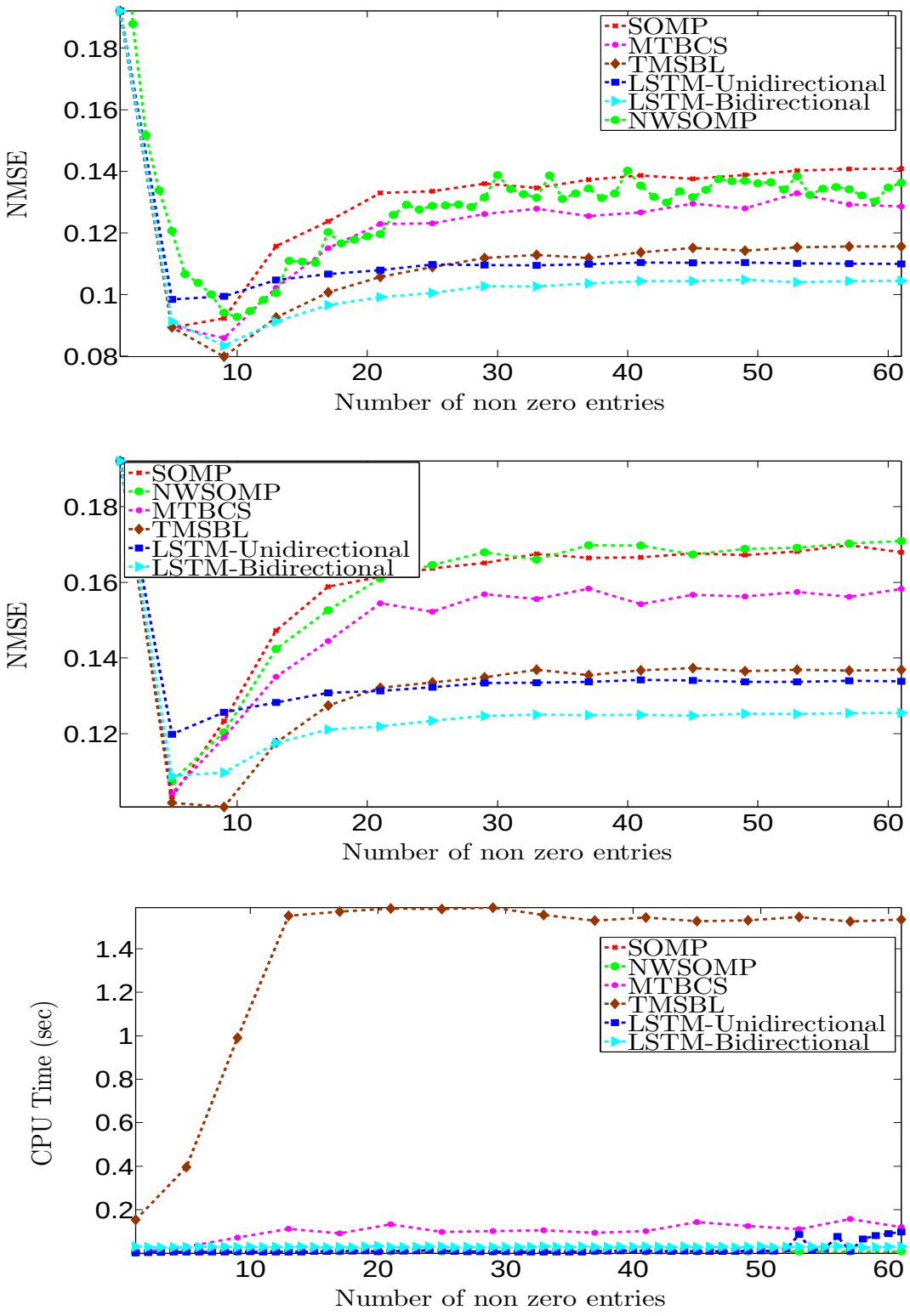


Figure 4.15: Up: Comparative reconstruction performance using DCT transform. Middle: Reconstruction performance using Wavelet transform. Bottom: CPU time. Note that the time is reported for T-MSBL which is a faster version of T-SBL.

4.7 Convolutional Deep Stacking Networks for Distributed Compressive Sensing

In this section, we propose a method that relies on a Convolutional Deep Stacking Network (CDSN) proposed in this section to capture the dependency amongst the different channels. To reconstruct the sparse vectors, the approach is similar to LSTM-CS with the difference that we use CDSN to capture structure information of sparse vectors. In CDSN, to capture the dependencies amongst different channels, a sliding convolution window over the columns of the matrix $\mathbf{S}$ is used where each convolution window contains w consecutive columns of $\mathbf{S}$ where w is the size of convolution window.

The main contributions of this section are proposing a convolutional version of the Deep Stacking Networks (DSNs)[29], which we refer to as CDSN, and then using CDSN to capture the dependencies among different channels in the MMV problem. We then use a similar greedy reconstruction algorithm to LSTM-CS at the decoder to reconstruct $\mathbf{S}$.

Please note that in the sparse representation literature, the dictionary learning method [3] uses the available training data to learn the sparsifying basis (Ψ in (4.2)) that can represent the signal as compactly as possible. The main difference between dictionary learning and our work here is that we assume the sparsifying basis as given and there is no need to learn it. In other words, the sparse vectors in $\mathbf{S}$ are not necessarily very sparse. We expect the performance of our method to improve by combining dictionary learning with our proposed method. Nevertheless, in this section, we focus on the performance improvement obtained by using the proposed approach only.

Block diagrams of the proposed method are presented in Fig. 4.16. Similar to LSTM-CS, the dashed lines in Fig. 4.16 show that the process of reconstructing the sparse vectors repeats for a number of iterations. At each iteration, each column of $\mathbf{S}$ is estimated by a separate process. The inputs to this process are the residuals at each iteration and the outputs are the estimated columns of $\mathbf{S}$.

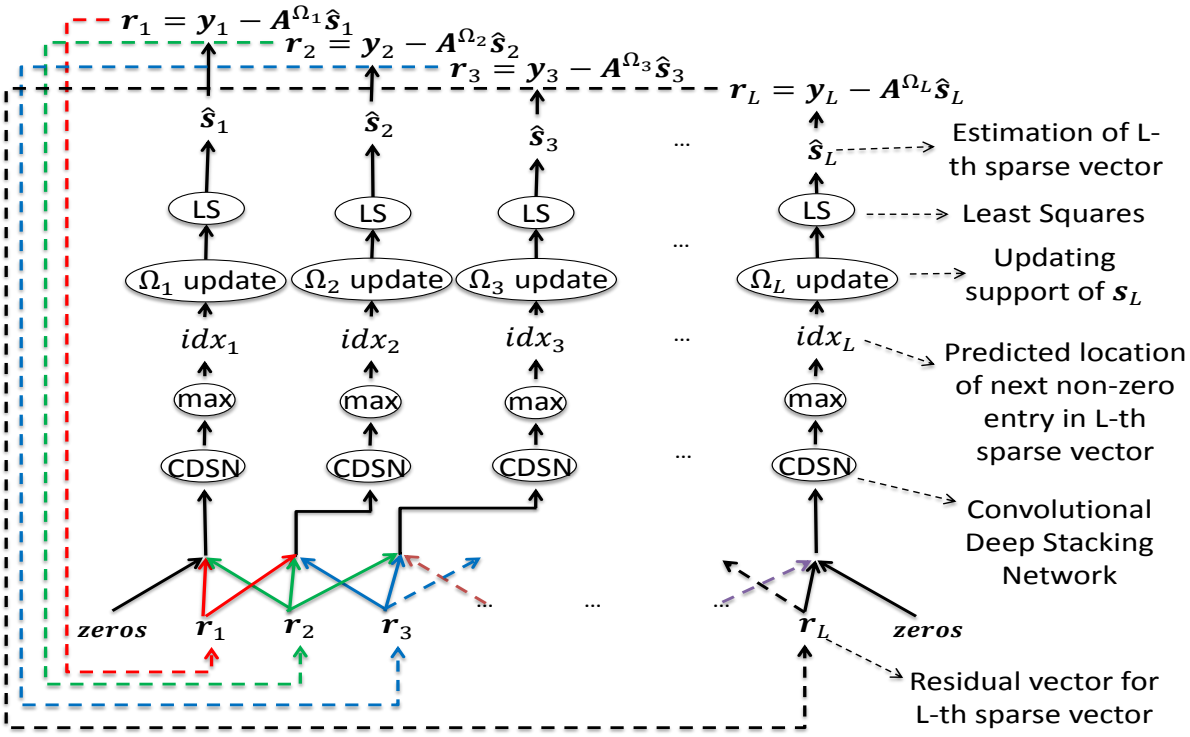
More formally, in the proposed method, before the i-th iteration of reconstructing the j-th column of $\mathbf{S}$, i non-zero entries of that column are predicted so far. We represent the j-th column of $\mathbf{S}$ by $\mathbf{s}_j$. At the i-th iteration, the first step predicts the location of the $(i+1)$-th non-zero entry of $\mathbf{s}_j$, using the residuals of columns contained in a sliding convolution window. In Fig. 4.16(a), an example with convolution window of size 3 is represented. This sliding convolution window helps in capturing the dependencies among channels. The predicted location of the $(i+1)$-th non-zero entry is then added to the support of $\mathbf{s}_j$. This support is represented by Ω_j in Fig. 4.16(a). The second step finds the updated estimate of $\mathbf{s}_j$ by solving a linear least squares problem that finds $\mathbf{s}_j$ given $\mathbf{y}_j$ (the j-th column of $\mathbf{Y}$) and $\mathbf{A}^{\Omega_j}$

$$\hat{\mathbf{s}}_j = \underset{\mathbf{s}_j}{\operatorname{argmin}} \|\mathbf{y}_j - \mathbf{A}^{\Omega_j}\mathbf{s}_j\|_2^2, \tag{4.56}$$

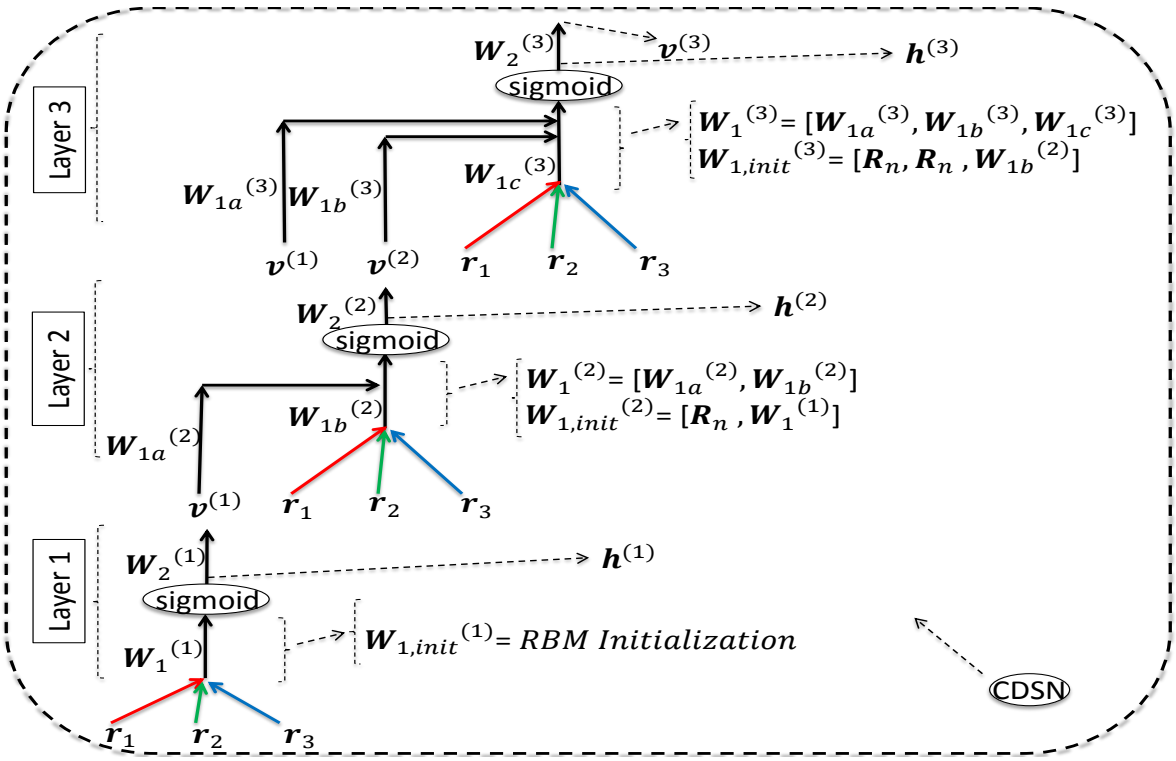
where $\mathbf{A}^{\Omega_j}$ is a matrix that includes only those columns of $\mathbf{A}$ that correspond to the support of $\mathbf{s}_j$. The definition of the residual matrix at the $i-$th iteration is $\mathbf{R}_i = \mathbf{Y} - \mathbf{A}\mathbf{S}_i$ where $\mathbf{S}_i$ is the estimate of the sparse matrix $\mathbf{S}$ at the $i-$th iteration. Columns of $\mathbf{R}$ are represented by $\mathbf{r}_j$, $j = 1, 2, \ldots, L$ in Fig. 4.16(a).

Now the remaining important questions are:

(i) how can we find the parameters of CDSN represented in Fig. 4.16(b), i.e., $\mathbf{W}_1^{(1)}, \mathbf{W}_2^{(1)}, \mathbf{W}_1^{(2)}, \mathbf{W}_2^{(2)}, \mathbf{W}_1^{(3)}, \mathbf{W}_2^{(3)}$?



(a) Block diagram of the proposed method with convolution window size 3.



(b) Block diagram of the convoltional deep stacking network with 3 layers. This diagram shows CDSN for channel 2 in Fig. 4.16(a), and $\mathbf{R}_n$ denotes a random matrix.

Figure 4.16: Proposed Method.

Please note that in Fig. 4.16(b), $\mathbf{W}_1^{(l)}$ is the matrix of weights from input layer to hidden layer for the l-th layer of CDSN and $\mathbf{W}_2^{(l)}$ is the matrix of weights from hidden layer to output layer for the l-th layer of CDSN. Residual vectors of channel one, two and three are represented by $\mathbf{r}_1, \mathbf{r}_2$ and $\mathbf{r}_3$ respectively in Fig. 4.16(b).

(ii) how should the training data be represented to find the parameters of CDSN given that at each iteration of the proposed method the location of one of the non-zero entries is determined? This means that the CDSN should observe the non-zero entries in the training data one by one. In the other words, we can not simply use the given training data (e.g., images), and an appropriate representation of it is necessary.

Question (ii) has already been addressed for LSTM-CS method described at the beginning of this chapter. We address question (i) by describing the CDSN formally and explaining the training method. The forward pass for l-th layer of CDSN represented in Fig. 4.16(b) is

$$\begin{aligned} \mathbf{h}^{(l)} &= \frac{1}{1+e^{-\mathbf{W}_1^{(l)}\mathbf{z}^{(l)}}} \\ \mathbf{v}^{(l)} &= [\mathbf{W}_2^{(l)}]^T \mathbf{h}^{(l)}. \end{aligned} \tag{4.57}$$

In (4.57), $\mathbf{v}^{(l)}$ is the output vector and $\mathbf{z}^{(l)}$ is the input vector of l-th layer and is defined as follows:

$$\mathbf{z}^{(l)} = [\mathbf{v}^{(1)}, \mathbf{v}^{(2)}, \ldots, \mathbf{v}^{(l-1)}, \mathbf{r}]. \tag{4.58}$$

In (4.58), $\mathbf{r}$ is the vector formed by the concatenation of all residual vectors in each convolution window. To find the CDSN unknown parameters $\mathbf{W}_1^{(l)}$ and $\mathbf{W}_2^{(l)}$ for each layer, l, a mean squared error cost function is minimized

$$\{\mathbf{W}_1^{(l)}, \mathbf{W}_2^{(l)}\} = \underset{\{\mathbf{W}_1^{(l)}, \mathbf{W}_2^{(l)}\}}{\operatorname{argmin}} \frac{1}{2}\|\mathbf{V}^{(l)} - \mathbf{T}\|_2^2, \tag{4.59}$$

where $\mathbf{T}$ is a matrix whose columns are the target vectors in the training set and $\mathbf{V}^{(l)}$ is a matrix whose columns are the corresponding output vectors from the l-th layer. Each layer of CDSN is a one layer neural network with a non-linear hidden layer and a linear output layer. In an CDSN, similar to a DSN [29], the optimization problem in (4.59) is solved for each layer separately. The linearity of the output layer for each layer of CDSN makes it possible to find a closed form solution for $\mathbf{W}_2^{(l)}$ given $\mathbf{W}_1^{(l)}$ and $\mathbf{T}$

$$\mathbf{W}_2^{(l)} = \left[\mathbf{H}^{(l)}[\mathbf{H}^{(l)}]^T\right]^{-1}\mathbf{H}^{(l)}\mathbf{T}^T, \tag{4.60}$$

where $\mathbf{H}^{(l)}$ is a matrix whose columns are $\mathbf{h}^{(l)}$ in (4.57) corresponding to different training samples in the training set. To prevent overfitting and to have a reliable solution for $\mathbf{W}_2^{(l)}$ when $\mathbf{H}^{(l)}$ is ill conditioned, usually an ℓ_2 regularization term is added to (4.59). In other words, to calculate $\mathbf{W}_2^{(l)}$ the following optimization

problem is solved:

$$\mathbf{W}_2^{(l)} = \underset{\mathbf{W}_2^{(l)}}{\operatorname{argmin}} \frac{1}{2}\|[\mathbf{W}_2^{(l)}]^T\mathbf{H}^{(l)} - \mathbf{T}\|_2^2 + \mu\|\mathbf{W}_2^{(l)}\|_2^2, \tag{4.61}$$

which results in

$$\mathbf{W}_2^{(l)} = \left[\mu\mathbf{I} + \mathbf{H}^{(l)}[\mathbf{H}^{(l)}]^T\right]^{-1}\mathbf{H}^{(l)}\mathbf{T}^T, \tag{4.62}$$

where $\mathbf{I}$ is the identity matrix.

To find $\mathbf{W}_1^{(l)}$, for each layer of CDSN we use the stochastic gradient descent method. To calculate the gradient of the cost function with respect to $\mathbf{W}_1^{(l)}$ given the fact that $\mathbf{W}_2^{(l)}$ and $\mathbf{H}^{(l)}$ depend on $\mathbf{W}_1^{(l)}$, it can be shown [130] that the gradient of the cost function in (4.59) with respect to $\mathbf{W}_1^{(l)}$ is

$$\begin{aligned}\frac{\partial\|\mathbf{V}^{(l)} - \mathbf{T}\|_2^2}{\partial\mathbf{W}_1^{(l)}} = \mathbf{Z}^{(l)}\Big[&[\mathbf{H}^{(l)}]^T \circ [1 - \mathbf{H}^{(l)}]^T \circ \\ &\Big[\left[\mathbf{H}^{(l)}\right]^\dagger\left[\mathbf{H}^{(l)}\mathbf{T}^T\right]\left[\mathbf{T}[\mathbf{H}^{(l)}]^\dagger\right] - \mathbf{T}^T\left[\mathbf{T}[\mathbf{H}^{(l)}]^\dagger\right]\Big]\Big],\end{aligned} \tag{4.63}$$

where $\mathbf{Z}^{(l)}$ is a matrix whose columns are $\mathbf{z}^{(l)}$ in (4.58) corresponding to different training samples in the training set and $\circ$ is the Hadamard product operator. Using the gradient information from past iterations can help to improve the convergence speed in convex optimization problems [11]. Although the problem in (4.59) is not necessarily convex because of the stack of non-linear hidden layers, but we found out experimentally that the gradient information from the past iterations can be helpful here as well. As in [130], we use the FISTA algorithm to accelerate the fine tuning. Therefore, the update equations for $\mathbf{W}_1^{(l)}$ at the k-th iteration are as follow:

$$\begin{aligned}\mathbf{W}_{1,k}^{(l)} &= \hat{\mathbf{W}}_{1,k}^{(l)} - \rho\frac{\partial\|\mathbf{V}^{(l)} - \mathbf{T}\|_2^2}{\partial\hat{\mathbf{W}}_{1,k}^{(l)}} \\ m_{k+1} &= \frac{1}{2}(1 + \sqrt{1 + 4m_k^2}) \\ \hat{\mathbf{W}}_{1,k+1}^{(l)} &= \hat{\mathbf{W}}_{1,k}^{(l)} + \frac{m_{k-1}}{m_{k+1}}(\mathbf{W}_{1,k}^{(l)} - \mathbf{W}_{1,k-1}^{(l)}).\end{aligned} \tag{4.64}$$

The curve of the FISTA coefficients $\frac{m_{k-1}}{m_{k+1}}$ with respect to the epoch number is represented Fig. 4.17. After computing $\mathbf{W}_1^{(l)}$ from (4.64), we use the closed form formulation in (4.62) to find $\mathbf{W}_2^{(l)}$.

Another important consideration for training the CDSN is that the cost function in (4.59) is not necessarily convex, therefore the initialization of $\mathbf{W}_1^{(l)}$ before fine tuning plays an important role. For initialization of the first layer of CDSN, we train a Restricted Boltzmann Machine (RBM) [59, 110] with Gaussian visible units and binary hidden units. This results in the following energy function between visible units, i.e., entries

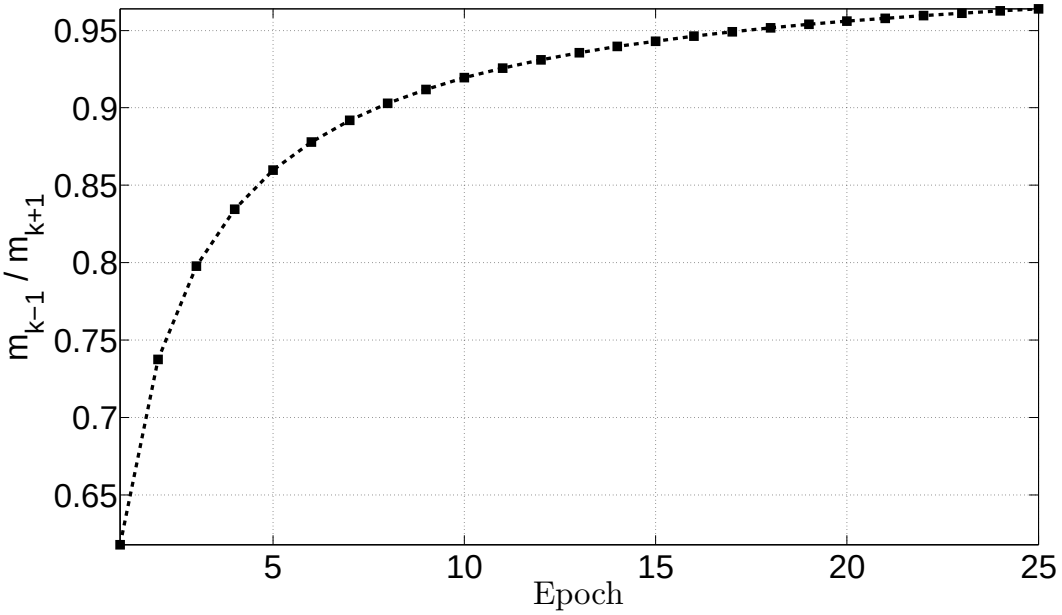


Figure 4.17: The curve of FISTA coefficients $\frac{m_{k-1}}{m_{k+1}}$ in (4.64) with respect to the epoch number.

of $\mathbf{z}^{(1)}$, and hidden units, i.e., entries of $\mathbf{h}^{(1)}$

$$E(\mathbf{z}^{(1)},\mathbf{h}^{(1)}) = \frac{1}{2}(\mathbf{z}^{(1)}-\mathbf{b}_1)^T(\mathbf{z}^{(1)}-\mathbf{b}_1) - \mathbf{b}_2^T\mathbf{h}^{(1)} - [\mathbf{z}^{(1)}]^T\mathbf{W}_{1,init}^{(1)}\mathbf{h}^{(1)}, \tag{4.65}$$

where $\mathbf{b}_1$ and $\mathbf{b}_2$ are vectors of bias values for the visible and hidden units respectively. The goal is to find $\mathbf{W}_{1,init}^{(1)}$ from the training input data, i.e., the residual vectors, $\mathbf{z}^{(1)}$, in the training data generated as explained earlier. Then we use $\mathbf{W}_{1,init}^{(1)}$ to initialize $\mathbf{W}_1^{(1)}$ as shown in Fig. 4.16(b). This approach has been shown to be also helpful in training neural networks and specifically autoencoders [60]. The parameters of the RBM can be found by maximizing the log probability that the RBM assigns to the input data, which is a function of the energy function in (4.65), using the contrastive divergence method [59]. The details on the general RBM training method used in this work can be found at [54]. As shown in the block diagram of CDSN in Fig. 4.16(a), to initialize the parameters of the upper layers of CDSN, $\mathbf{W}_1^{(l+1)}$, we use the learned parameters of the lower layer, $\mathbf{W}_1^{(l)}$, as initialization. This approach has been shown to be helpful in training DSNs [29] and it was helpful in our task as well. This completes the description of the training method for CDSN and the answer for question (ii).

Given the trained CDSN, a summary of the proposed reconstruction algorithm that finds the sparsest solution $\mathbf{S}$ given $\mathbf{Y}$ and $\mathbf{A}$ in (4.4) is presented in Algorithm 4. We refer to this algorithm as CDSN-CS since we have used a convolutional DSN for distributed compressive sensing. A more high level architecture of the proposed method is also presented in Fig. 4.18.

Algorithm 4 Distributed Compressive Sensing using Covolutional Deep Stacking Network (CDSN-CS)

Inputs: CS measurement matrix $\mathbf{A} \in \Re^{M \times N}$; matrix of measurements $\mathbf{Y} \in \Re^{M \times L}$; minimum ℓ_2 norm of residual matrix "*resMin*" as stopping criterion; Trained "*cdsn*" model; Convolution window size "*w*"
Output: Matrix of sparse vectors $\hat{\mathbf{S}} \in \Re^{N \times L}$
Initialization: $\hat{\mathbf{S}} = 0$; $j = 1$; $i = 1$; $\Omega = \emptyset$; $\mathbf{R} = \mathbf{Y}$.

procedure CDSN-CS($\mathbf{A}$,$\mathbf{Y}$,*cdsn*)
  while $i \leq N$ and $\|\mathbf{R}\|_2 \leq resMin$ **do**
    $i \leftarrow i+1$
    for $j = 1 \rightarrow L$ **do**
      $\mathbf{R}(:,j)_i \leftarrow \frac{\mathbf{R}(:,j)_{i-1}}{max(|\mathbf{R}(:,j)_{i-1}|)}$
      $\mathbf{v}_j \leftarrow$ $cdsn([\mathbf{R}(:,j-\frac{w}{2})_i, \mathbf{R}(:,j-\frac{w}{2}+1)_i, \ldots, \mathbf{R}(:,j+\frac{w}{2}-1)_i, \mathbf{R}(:,j+\frac{w}{2})_i])$
      $idx \leftarrow Support(max(\mathbf{v}_j))$
      $\Omega_i \leftarrow \Omega_{i-1} \cup idx$
      $\hat{\mathbf{S}}^{\Omega_i}(:,j) \leftarrow (\mathbf{A}^{\Omega_i})^{\dagger}\mathbf{Y}(:,j)$ ▷ Least Squares
      $\hat{\mathbf{S}}^{\Omega_i^C}(:,j) \leftarrow 0$
      $\mathbf{R}(:,j)_i \leftarrow \mathbf{Y}(:,j) - \mathbf{A}^{\Omega_i}\hat{\mathbf{S}}^{\Omega_i}(:,j)$
    end for
  end while
end procedure

4.7.1 Experimental Evaluation and Discussion

In this section we experimentally demonstrate: (i) How is the performance of the proposed method in this section compared to other reconstruction algorithms? (ii) How fast is the proposed method? (iii) What are the effects of the convolution window size in CDSN-CS? (iv) What are the effects of the RBM initialization?

To address the above issues, we performed experiments on the same natural image dataset described earlier in this chapter. Ten randomly selected test images from each of 3 classes of this dataset were used for experiments. The images are shown in Fig. 4.10. The size of each of the used images was 64×64. Each image was divided into 8×8 non-overlapping blocks. After reconstructing all the blocks of an image, the reconstruction error for the reconstructed image was calculated. The reconstruction error is defined similar to (4.15). We encoded 8 blocks ($L = 8$) of each image simultaneously using a random measurement matrix and reconstructed them at the decoder. Therefore, $\mathbf{S}$ in (4.4) had 8 columns and each column had $N = 64$ entries. We used 40% measurements, i.e., $\mathbf{Y}$ in (4.4) had 8 columns and each column had $M = 25$ entries. The encoder was a typical compressive sensing encoder, i.e., $\mathbf{A}$ was a randomly generated matrix. Each column of $\mathbf{A}$ was normalized to have unity norm. To simulate the measurement noise, Gaussian noise with standard deviation 0.005 was added to the measurement matrix $\mathbf{Y}$ in (4.4). We used two popular transforms, DCT and Wavelet, as the sparsifying basis Ψ in (4.2). For the wavelet transform we used the Haar wavelet transform with 3 levels of decomposition. We used 55 images for the training set, 5 images for the validation set and 10 images for the test set. The PC used to perform the experiments had an Intel(R) Core(TM) i7 CPU with clock 2.93 GHz and with 16 GB RAM.

The performance of the proposed reconstruction algorithm (CDSN-CS) was compared with 5 reconstruction methods for the MMV problem. These methods are: 1) Simultaneous Orthogonal Matching Pursuit (SOMP) which is a well known baseline for the MMV problem, 2) Bayesian Compressive Sens-

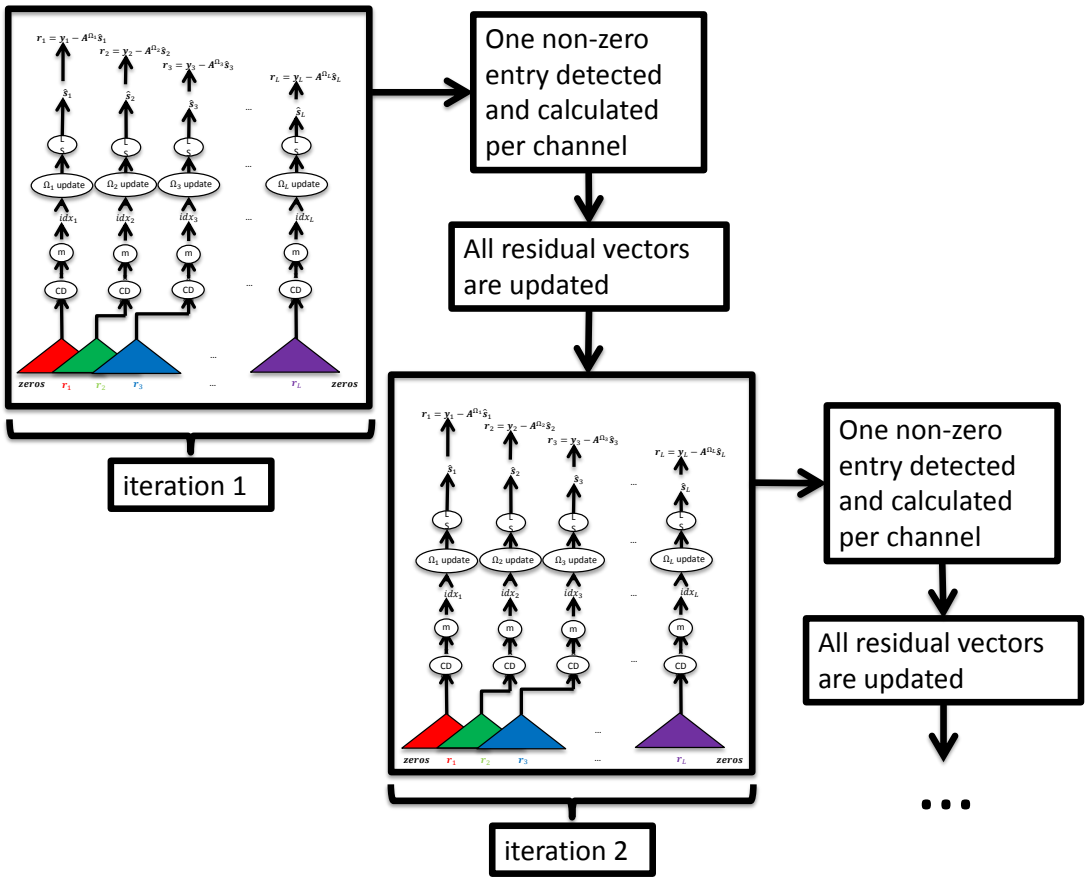


Figure 4.18: High level block diagram of the proposed method.

ing (BCS)[71] applied independently on each channel, 3) Multitask Compressive Sensing (MT-BCS) [72] which takes into account the statistical dependency among the different channels, 4) Sparse Bayesian Learning for Temporally correlated sources (T-SBL) [133] which exploits the correlation among different sources in the MMV problem and 5) Nonlinear Weighted SOMP (NWSOMP) [95]. For the BCS method we set the initial noise variance of i-th channel to the value suggested by the authors, i.e., $std(\mathbf{y}_i)^2/100$ where $i \in \{1,2,3,4,5,6,7,8\}$ and $std(.)$ calculates the standard deviation. The threshold for stopping the algorithm was set to 10^{-8}. For MT-BCS we set the parameters of the Gamma prior on noise variance to $a = 100/0.1$ and $b = 1$ which are the values suggested by the authors. We set the stopping threshold to 10^{-8} as well. For T-SBL, we used the default values recommended by the authors. For NWSOMP, during training, we used three layers, each layer having 512 neurons and 25 epochs of parameters update. For CDSN-CS, during training, we used three layers, 64 neurons per layer with different window sizes and 25 epochs of parameter updates. For RBM initialization, we ran 200 epochs of RBM parameter update with step size 0.01. To monitor overfitting of the RBM, we used free energy as explained in [54]. For fine tuning CDSN-CS after RBM initialization we used step size 0.002. The regularization parameter μ in (4.62) was set to 0.01. To monitor and prevent overfitting, we used 5 images per channel as the validation set and we used early stopping if necessary. Please note that the images used for validation were different from those used in the training set

or in the test set.

The results for the different classes of images are presented in Fig. 4.19 for the DCT transform and in Fig. 4.20 for the Wavelet transform. In these figures, the vertical axis is the *MSE* defined in (4.15) and the horizontal axis is the number of non-zero entries in each sparse vector. The number of measurements, M, is fixed to 25. Each point on the curves is the average of the *MSE*s over 10 reconstructed test images at the decoder. For all results, we used a convolution window of size 5 because it gave the best performance compared to other window sizes. For image class of flowers (the bottom part of Fig. 4.19 and Fig. 4.20), a convolution window of size 7 gave better performance. As observed in these figures, CDSN-CS outperforms the five reconstruction methods SOMP, BCS applied to each channel independently, MT-BCS, T-SBL and NWSOMP. We believe that this improvement in the performance is due to exploiting the dependencies among the different channels by CDSN network.

To study the effect of the convolution window size, a comparison among the different convolution window sizes in CDSN-CS for the image class "*cows*" with the DCT transform is presented in Fig. 4.21. As observed from this figure, increasing the window size improves the results up to a point after which the results do not improve any more. Since we use distinct patches from each image, we might assign this behaviour to the fact that the residuals of image patches that are far from each other might be less correlated than the residuals of image patches that are close to each other.

To show that RBM initialization is helpful for our task, we conducted two experiments. In the first experiment the CDSN is trained using random initialization. In the second experiment it is trained using RBM initialization. The results are presented in Fig. 4.22. As observed in this figure, RBM initialization improves the reconstruction performance.

To conclude this section we present the CPU time for the different reconstruction algorithms discussed in this section in Fig. 4.23. Since all methods are run in MATLAB and on the same machine, Fig. 4.23 gives a rough idea about how fast the different methods discussed in this section are. As observed in this figure, the proposed method is faster than the Bayesian methods discussed and is almost as fast as the greedy methods.

4.8 Conclusions

This chapter presents a method to reconstruct sparse vectors for the MMV problem. The proposed method learns the structure of sparse vectors and does not rely on the commonly used joint sparsity assumption. Through experiments on two real world datasets, we showed that the proposed method outperforms the general MMV baseline SOMP as well as a number of Bayesian model based methods for the MMV problem. Please note that we have not used multiple layers of LSTM or the advanced deep learning methods for training, e.g., regularization using drop out which can improve the performance of LSTM-CS. This chapter is a proof of concept that deep learning methods and specifically sequence modelling methods, e.g., LSTM, can improve the performance of the MMV solvers significantly. This is specially the case when the sparsity patterns are more complicated than that of obtained by the DCT or Wavelet transforms. We showed this on the MNIST dataset. We showed that the proposed method is almost as fast as greedy methods. The good performance of the proposed method depends on the availability of training data (as is the case in all

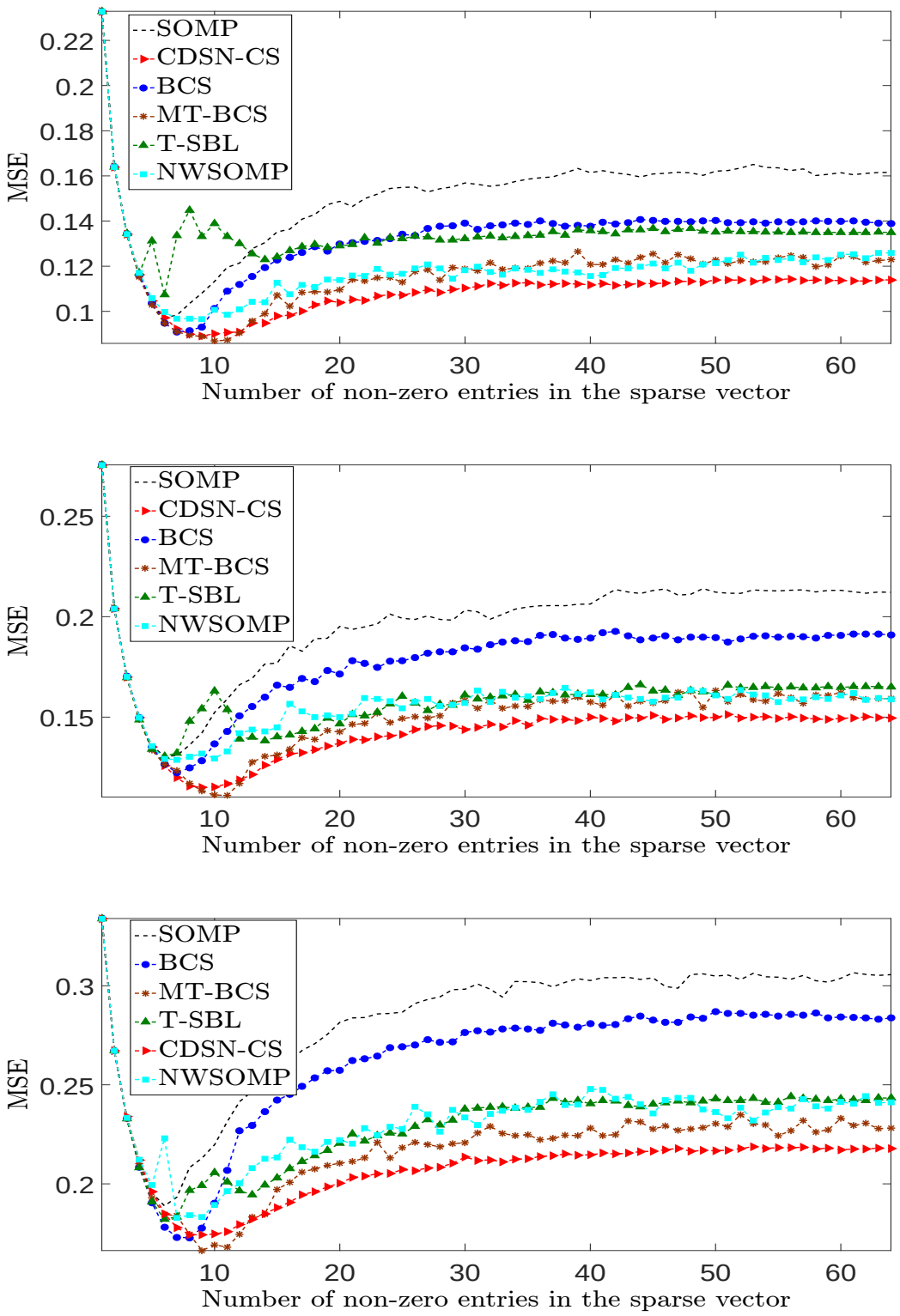


Figure 4.19: Comparison of different MMV reconstruction algorithms performance for the natural image dataset using DCT transform. Image classes from top to bottom are buildings, cows and flowers respectively.

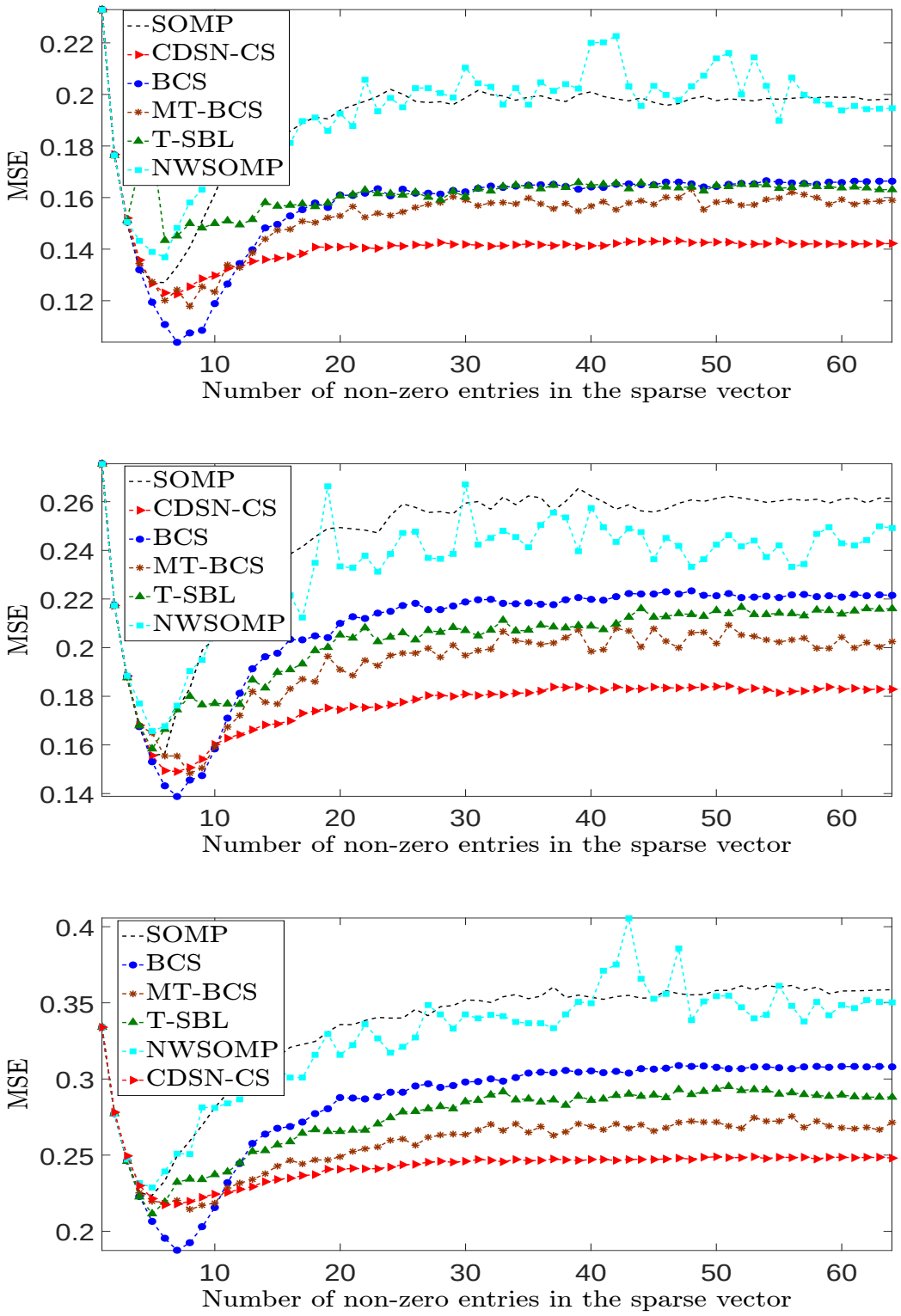


Figure 4.20: Comparison of different MMV reconstruction algorithms performance for the natural image dataset using Wavelet transform. Image classes from top to bottom are buildings, cows and flowers respectively.

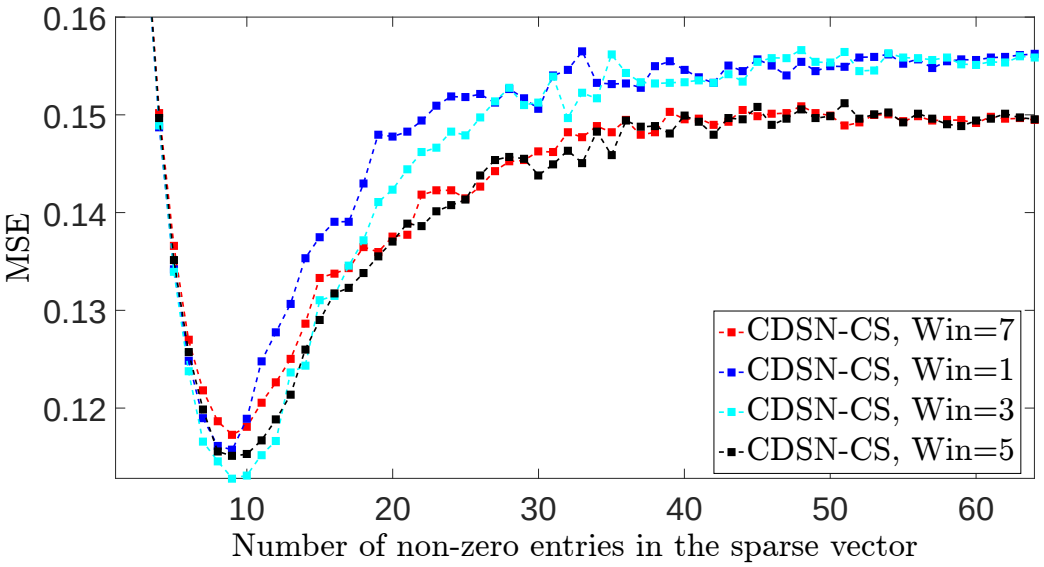


Figure 4.21: The performance of CDSN-CS with different window sizes. This experiment was conducted for image class of cows with DCT transform as sparsifying basis.

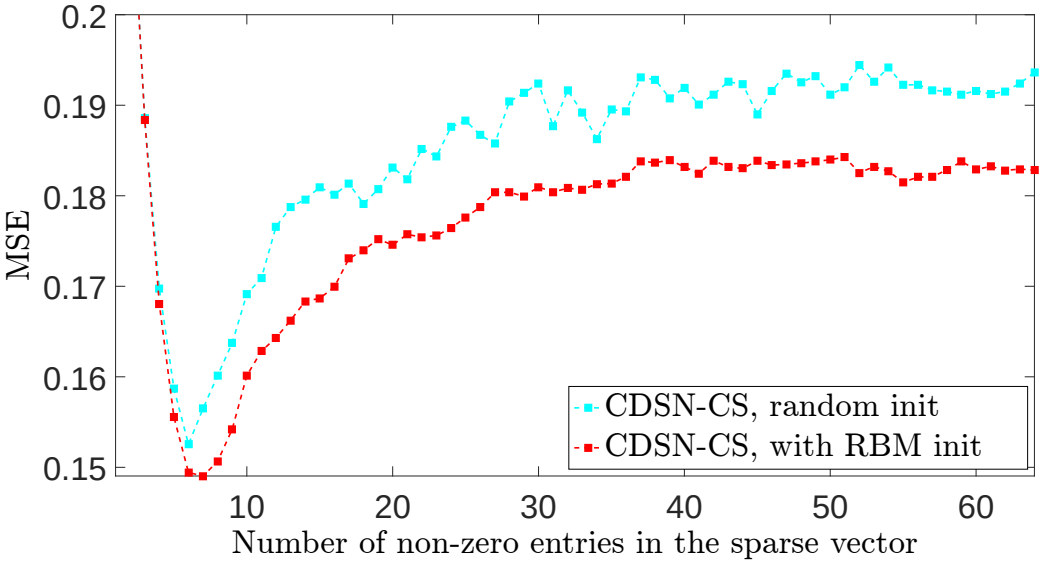


Figure 4.22: The performance of CDSN-CS with and without RBM initialization. This experiment was conducted for image class of cows with Wavelet transform as sparsifying basis.

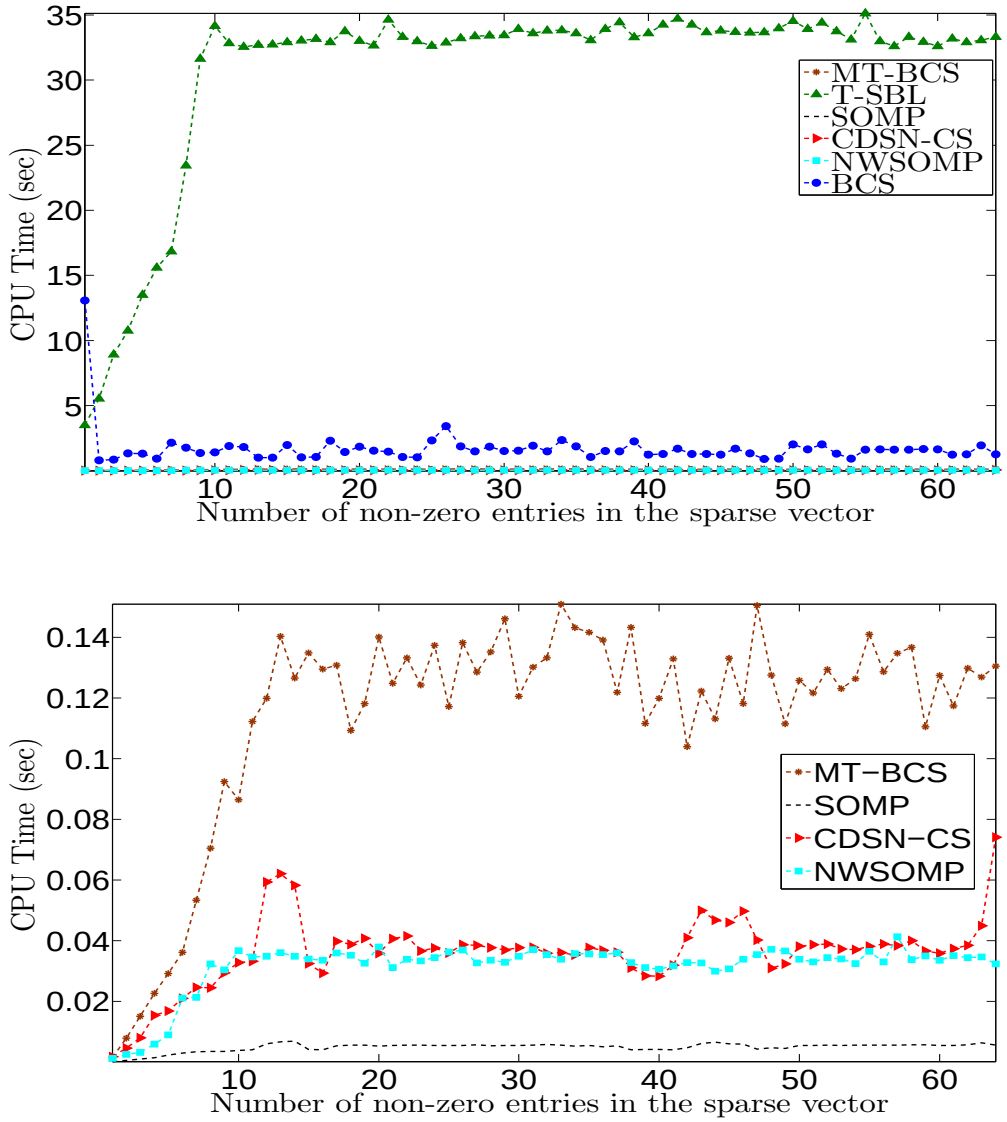


Figure 4.23: CPU time of different MMV reconstruction algorithms discussed in this section. Up: all methods. Bottom: removing T-SBL and BCS to make the difference among remaining methods more visible.

deep learning methods). In many applications, however, training data is available, e.g., different images of the same class or signals with similar sparsity patterns. Please note that if collecting training samples is expensive or enough training samples are not available, using other sparse reconstruction methods is recommended. We also presented the bidirectional version of LSTM-CS along with a reconstruction method based on Convolutional Deep Stacking Networks model proposed in this chapter.

Chapter 5

Conclusions and Future Work

All glory comes from daring to begin.
— Eugene F Ware

5.1 Conclusions

In this thesis, we focused on deep learning methods used for sequence modelling. We approached three important problems: speech recognition, sentence modeling for web search and distributed compressive sensing. For each problem, we explained why the problem have a sequential nature, and then, proposed new methods to exploit this sequential behavior. We showed effectiveness of proposed methods through extensive experiments on real world challenging datasets. A summary of our findings are listed below:

5.1.1 Recurrent Deep Stacking Networks for Speech Recognition

The traditional Echo State Networks (ESNs) learn only the set of one of three important weight matrices. In this work, we wanted to address the problem of how to learn them all. The key property that characterizes the ESN is the use of linear output units so that the learning is simple, can be formulated as a convex least-square ridge regression problem w.r.t output weights. In extending the learning of the output weights to learning input and recurrent weights, we make use of the same property of the linear output units to develop and formulate constraints for the sets of various ESN weight matrices. Such constraints are then used to derive analytic forms of the error gradients w.r.t the input and recurrent weights to be learned. The standard learning method of BPTT used for the general RNN (with typically nonlinear output units) does not admit analytical forms of gradient computation. BPTT requires recursively propagating the error signal backward through time, a very different style of computation and learning than what we have developed in this work for ESNs.

A novel deep learning architecture, the R-DSN, which extends the earlier RNN and DSN models was also proposed. The R-DSN constructs multiple modules of the RNN using stacking, in the same way that the DSN uses stacking to form multiple modules of a simple, non-recurrent feed-forward neural network. Alternatively, the R-DSN can be viewed as a generalization of the DSN, where the generalization lies in embedding recurrent connections in each module that were missing in the earlier DSN. The main technical

contribution of the work reported in this part is the development of closed-from formulas for the gradient computation based on the special structure of the R-DSN, and the batch-mode training method for all parameters in the R-DSN capitalizing on these formulas.

The above methods were evaluated on TIMIT dataset for phoneme recognition task in speech recognition. The results were better than a single layer RNN and also almost as good as the state-of-the-art methods, but not better than them (Tables 2.1 and 2.2).

5.1.2 A Sentence Modelling Method for Web Search Task

We proposed LSTM-DSSM, a new sentence modeling method for web search task. The proposed method was based on long short-term memory network to learn the long range context information and embed the key information of a sentence in one semantic vector. We showed that the semantic vector evolves over time and only takes useful information from any new input. This has been made possible by input gates that detect useless information and attenuate it. One important property of the proposed method was that, it created vector representations for queries and documents in such a way that semantic representations of queries and clicked documents are as close as possible, while, those of queries and unclicked documents are as far as possible. This was due to LSTM-DSSM's specific architecture and cost function.

Due to the general limitation of available human labelled data, we proposed and implemented and trained LSTM-DSSM with a *weak supervision* signal using users' click-through data of a commercial web search engine.

By performing a detailed analysis on the model, we showed that: 1) The proposed model is robust to noise, i.e., it mainly embeds keywords in the final semantic vector that represents the whole sentence and 2) in the proposed model, each cell is usually allocated to keywords from a specific topic. These findings have been supported using extensive examples.

As a concrete sample application of the proposed sentence embedding method, we evaluated it on the important language processing task of web document retrieval. The evaluations were performed on click-through data from a real world commercial search engine. For the task of information retrieval, the proposed method outperformed all existing state-of-the-art methods significantly (Table 3.4).

5.1.3 A Deep Learning Approach to Distributed Compressive Sensing

We presented a new method to reconstruct sparse vectors for the MMV problem based on a deep learning approach. The proposed method learns the structure of the sparse vectors and does not rely on the commonly used joint sparsity assumption. More accurately, it addresses the following three important problems in the MMV problem at the same time: (a) How to exploit structures besides sparsity during reconstruction? (b) How to do reconstruction in the MMV problem when sparse vectors are not jointly sparse? (c) How to exploit available offline data for a better reconstruction performance?

Through experiments on two real world datasets, we showed that the proposed method outperforms the general MMV baseline SOMP as well as a number of Bayesian model based methods for the MMV problem. Please note that we have not used multiple layers of LSTM or the advanced deep learning methods for training, e.g., regularization using drop out which can improve the performance of LSTM-CS. This

work was a proof of concept that deep learning methods and specifically sequence modelling methods, e.g., LSTM, can improve the performance of MMV solvers significantly. This is specially the case when the sparsity patterns are more complicated than that obtained by the DCT or a Wavelet transform. We showed this on the MNIST dataset. We showed that the proposed method is almost as fast as the greedy methods. The good performance of the proposed method depends on the availability of large amount of training data (as is the case in all deep learning methods). In many applications, enough training data is available, e.g., different images of the same class or different signals with similar sparsity patterns. Please note that if collecting training samples is expensive or enough training samples are not available, then using other sparse reconstruction methods is recommended.

The above methods were evaluated on two real world image datasets, MNIST dataset and different classes of a natural image dataset. The results were better than the well known greedy method Simultaneous Orthogonal Matching Pursuit (SOMP) and a number of state-of-the-art model based Bayesian methods (Fig. 4.4).

5.2 Future Work

There is still a lot that can be done in the direction of developing deep sequence modelling methods for the problems discussed in this thesis. Below is a list of reasonably "*feasible*" future work problems:

1. Using the proposed sentence embedding method in chapter 3, LSTM-DSSM, for other important language processing tasks for which we believe sentence embedding plays a key role. Some examples are: question answering, machine translation, and sentiment analysis.

2. Exploiting the prior information about the structure of the different matrices in Fig. 3.2 in chapter 3 to develop more effective cost functions and learning methods.

3. Exploiting the attention mechanism [8] in the proposed model in chapter 3 to improve the retrieval performance and finding out which words in the query are aligned to which words of the document.

4. Using the method proposed in chapter 3 for text summarization. For example, given an article or large body of text, the task is to automatically generate a title for it.

5. Building a multi-resolution version of the method proposed in chapter 3. This means that using a Convolutional Neural Network (CNN) or LSTM for character level feature extraction, an LSTM on the top to extract word / sentence level information, and having a cosine similarity cost function on the top. This multi-resolution architecture will be learned end to end using a method similar to the one described in chapter 3.

6. Extending the proposed method in chapter 4 to non-linear distributed compressive sensing. This means finding the sparsest solution $\mathbf{s}$ in

$$\mathbf{y} = f(\mathbf{A}\mathbf{s}) \tag{5.1}$$

where $f(.)$ is a non-linear function. This non-linearity might come from the physical properties of the problem or we might want to add it intentionally. We should also explore what a non-linear encoder in CS might result in.

7. Using the proposed LSTM-CS approach in chapter 4 (to reconstruct data compressed using CS), for the following important compressive sensing applications:

   (a) Magnetic Resonance Imaging (MRI) to shorten MRI scanning sessions. There are also opportunities to use LSTM-CS for Dynamic MRI where the correlation among image frames can be effectively exploited by LSTM-CS.

   (b) Computed Tomography (CT) where the goal is to reduce the amount of radiation.

   (c) Video compressive sensing where there is correlation amongst the video frames.

   (d) Health telemonitoring and specifically compressive sensing of EEG signals where there is correlation amongst the different EEG channels.

   (e) Using a modified version of LSTM-CS for Blind Compressive Sensing (BCS) [47] where both the sparse vector $\mathbf{s}$ and the sparsifying basis Ψ are unknown.

   (f) Using an optimized implementation of LSTM-CS for Single-Pixel cameras [34].

8. Extending the evaluation tasks in chapter 2 to more complex continuous phoneme and word recognition tasks.

9. One challenge in compressive sensing research is that there is no unified framework to reliably compare all existing reconstruction methods. This is specially the case when it comes to real world datasets. An important future work direction is to create a unified and easy to use framework, such that, new reconstruction methods can be easily compared with existing methods.

Bibliography

[1] Compressive sensing resources. http://dsp.rice.edu/cs , accessed on Decemeber 13, 2016. → pages 7

[2] Big data, for better or worse: 90% of world's data generated over last two years. 2013. URL https://www.sciencedaily.com/releases/2013/05/130522085217.htm. accessed on December 13, 2016. → pages 1

[3] M. Aharon, M. Elad, and A. Bruckstein. k -svd: An algorithm for designing overcomplete dictionaries for sparse representation. *Signal Processing, IEEE Transactions on*, 54(11):4311–4322, Nov 2006. → pages 99

[4] D. Angelosante, G. Giannakis, and E. Grossi. Compressed sensing of time-varying signals. In *16th International Conference on Digital Signal Processing*, pages 1–8, 2009. → pages 72, 74

[5] A. Averbuch, J. Weill, O. Barkan, and S. Dekel. Adaptive compressed tomography sensing. *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2195–2202, 2013. → pages 7

[6] S. Aviyente. Compressed sensing framework for eeg compression. In *Statistical Signal Processing, 2007. SSP '07. IEEE/SP 14th Workshop on*, pages 181–184, 2007. → pages 7

[7] F. Bach, R. Jenatton, J. Mairal, and G. Obozinski. Structured sparsity through convex optimization. *Statistical Science*, 27(4):450–468, 2012. → pages 9

[8] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *ICLR2015*, 2015. http://arxiv.org/abs/1409.0473. → pages 29, 113

[9] R. Baraniuk. Compressive sensing [lecture notes]. *IEEE Signal Processing Magazine*, 24(4):118 –121, july 2007. → pages 7, 69

[10] R. Baraniuk, V. Cevher, M. Duarte, and C. Hegde. Model-based compressive sensing. *Information Theory, IEEE Transactions on*, 56(4):1982–2001, April 2010. → pages 72

[11] A. Beck and M. Teboulle. *Gradient-based algorithms with applications to signal-recovery problems*. Cambridge University Press, 2009. → pages 21, 102

[12] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *Neural Networks, IEEE Transactions on*, 5(2):157–166, 1994. → pages 6, 14, 16

[13] Y. Bengio, N. Boulanger-Lewandowski, and R. Pascanu. Advances in optimizing recurrent networks. In *Proc. ICASSP*, Vancouver, Canada, May 2013. → pages 5, 14, 30, 74

[14] E. Candes, J. Romberg, and T. Tao. Stable signal recovery from incomplete and inaccurate measurements. *Communications on Pure and Applied Mathematics*, 59(8):1207–1223, 2006. → pages 7, 69

[15] E. J. Cands, Y. C. Eldar, D. Needell, and P. Randall. Compressed sensing with coherent and redundant dictionaries. *Applied and Computational Harmonic Analysis*, 31(1):59 – 73, 2011. → pages 8, 69

[16] H. Cao, V. Leung, C. Chow, and H. Chan. Enabling technologies for wireless body area networks: A survey and outlook. *Communications Magazine, IEEE*, 47(12):84–93, 2009. → pages 7

[17] J. Chen and L. Deng. A primal-dual method for training recurrent neural networks constrained by the echo-state property. In *Proceedings of the International Conf. on Learning Representations (ICLR)*, 2014. → pages 30

[18] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *NIPS Deep Learning Workshop*, 2014. → pages 28

[19] R. Collobert and J. Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *International Conference on Machine Learning, ICML*, 2008. → pages 29

[20] G. Dahl, D. Yu, L. Deng, and A. Acero. Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *Audio, Speech, and Language Processing, IEEE Transactions on*, 20(1):30 –42, jan. 2012. ISSN 1558-7916. → pages 1, 68, 74

[21] G. E. Dahl, D. Yu, L. Deng, and A. Acero. Large vocabulary continuous speech recognition with context-dependent DBN-HMMs. In *Proc. IEEE ICASSP*, pages 4688–4691, Prague, Czech, May 2011. → pages 68

[22] M. Davies and Y. Eldar. Rank awareness in joint sparse recovery. *IEEE Transactions on Information Theory*, 58(2):1135 –1146, Feb. 2012. → pages 9, 70

[23] L. Deng. Achievements and challenges of deep learning. *APSIPA Transactions on Signal and Information Processing*, 2015. → pages 1

[24] L. Deng. How deep reinforcement learning can help chatbots. *Venturebeat*, 2016. URL http://venturebeat.com/2016/08/01/how-deep-reinforcement-learning-can-help-chatbots/. accessed on December 13, 2016. → pages 6

[25] L. Deng and J. Chen. Sequence classification using high-level features extracted from deep neural networks. In *Proc. ICASSP*, 2014. → pages 30

[26] L. Deng and D. Yu. Deep convex networks for speech pattern classification. *Proc. Interspeech*, 2011. → pages 14, 18

[27] L. Deng and D. Yu. Deep learning: Methods and applications. *NOW Publishers*, pages 1–199, 2014. → pages 1

[28] L. Deng, K. Hassanein, and M. Elmasry. Analysis of the correlation structure for a neural predictive model with application to speech recognition. *Neural Networks*, 7(2):331–339, 1994. → pages 5, 14, 30, 74

[29] L. Deng, D. Yu, and J. Platt. Scalable stacking and learning for building deep architectures. In *Proc. ICASSP*, pages 2133 –2136, march 2012. → pages iv, 14, 68, 74, 99, 101, 103

[30] L. Deng, O. Abdel-Hamid, and D. Yu. A deep convolutional neural network using heterogeneous pooling for trading acoustic invariance with phonetic confusion. In *Proc. IEEE ICASSP*, Vancouver, Canada, May 2013. → pages 14

[31] D. Donoho. Compressed sensing. *IEEE Transactions on Information Theory*, 52(4):1289 –1306, april 2006. → pages 7, 8, 69

[32] D. L. Donoho and M. Elad. Optimally sparse representation in general (nonorthogonal) dictionaries via ℓ_1 minimization. *Proceedings of the National Academy of Sciences of the United States of America*, 100(5):2197–2202, 2003.
"http://www-stat.stanford.edu/ donoho/Reports/2002/OptSparse.pdf", accessed on December 13, 2016. → pages 8

[33] M. Duarte and Y. Eldar. Structured compressed sensing: From theory to applications. *IEEE Transactions on Signal Processing*, 59(9):4053 –4085, sept. 2011. → pages 9, 70, 72

[34] M. F. Duarte, M. A. Davenport, D. Takbar, J. N. Laska, T. Sun, K. F. Kelly, and R. G. Baraniuk. Single-pixel imaging via compressive sampling. *IEEE Signal Processing Magazine*, 25(2):83–91, 2008. → pages 7, 114

[35] Y. Eldar and H. Rauhut. Average case analysis of multichannel sparse recovery using convex relaxation. *IEEE Transactions on Information Theory*, 56(1):505 –519, Jan. 2010. → pages 9, 70

[36] J. L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990. → pages 5, 14, 30, 74

[37] H. Fang, S. Gupta, F. N. Iandola, R. K. Srivastava, L. Deng, P. Dollár, J. Gao, X. He, M. Mitchell, J. C. Platt, C. L. Zitnick, and G. Zweig. From captions to visual concepts and back. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR, Boston, MA, USA*, pages 1473–1482, 2015. → pages 1

[38] J. Fang, Y. Shen, and H. Li. Pattern coupled sparse bayesian learning for recovery of time varying sparse signals. In *19th International Conference on Digital Signal Processing (DSP)*, pages 705–709, 2014. → pages 72, 74, 81

[39] A. C. Faul and M. E. Tipping. Analysis of sparse bayesian learning. In *Advances in Neural Information Processing Systems (NIPS) 14*, pages 383–389. MIT Press, 2001. → pages 73

[40] R. Fergus. Introduction to convolutional networks. Presented as a lecture at CIFAR Deep Learning Summer School, Montreal, Canada, 2016. → pages ix, 4

[41] J. Gao, W. Yuan, X. Li, K. Deng, and J.-Y. Nie. Smoothing clickthrough data for web search ranking. In *Proceedings of the 32Nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '09, pages 355–362, New York, NY, USA, 2009. ACM. → pages 33

[42] J. Gao, K. Toutanova, and W.-t. Yih. Clickthrough-based latent semantic models for web search. SIGIR '11, pages 675–684. ACM, 2011. → pages 41

[43] J. Gao, X. He, W. tau Yih, and L. Deng. Learning continuous phrase representations for translation modeling. In *Proc. ACL*, 2014. → pages 68

[44] J. Gao, P. Pantel, M. Gamon, X. He, L. Deng, and Y. Shen. Modeling interestingness with deep neural networks. In *Proc. EMNLP*, 2014. → pages 68

[45] F. A. Gers, J. Schmidhuber, and F. Cummins. Learning to forget: Continual prediction with lstm. *Neural Computation*, 12:2451–2471, 1999. → pages 31, 74

[46] F. A. Gers, N. N. Schraudolph, and J. Schmidhuber. Learning precise timing with lstm recurrent networks. *J. Mach. Learn. Res.*, 3:115–143, Mar. 2003. → pages 31, 74

[47] S. Gleichman and Y. C. Eldar. Blind compressed sensing. *IEEE Transactions on Information Theory*, 57(10):6958–6975, 2011. → pages 114

[48] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. → pages 1

[49] A. Graves. Sequence transduction with recurrent neural networks. In *Representation Learning Workshp, ICML*, 2012. → pages 5, 14, 30, 74

[50] A. Graves, A. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In *Proc. ICASSP*, Vancouver, Canada, May 2013. → pages 29

[51] A. Graves, G. Wayne, M. Reynolds, T. Harley, I. Danihelka, A. Grabska-Barwińska, S. G. Colmenarejo, E. Grefenstette, T. Ramalho, J. Agapiou, A. P. Badia, K. M. Hermann, Y. Zwols, G. Ostrovski, A. Cain, H. King, C. Summerfield, P. Blunsom, K. Kavukcuoglu, and D. Hassabis. Hybrid computing using a neural network with dynamic external memory. *Nature*, advance online publication, 2016. → pages 1

[52] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, 2016. → pages ix, 5

[53] K. M. Hermann and P. Blunsom. Multilingual models for compositional distributed semantics. *arXiv preprint arXiv:1404.4641*, 2014. accessed on December 13, 2016. → pages 29

[54] G. Hinton. A practical guide to training restricted boltzmann machines: Version 1. Technical report, University of Toronto, 2010. → pages 103, 105

[55] G. Hinton and R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504 – 507, 2006. → pages 1, 14

[56] G. Hinton, L. Deng, D. Yu, G. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. Sainath, and B. Kingsbury. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *Signal Processing Magazine, IEEE*, 29(6):82–97, 2012. → pages 2

[57] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, November 2012. → pages 68, 74

[58] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6): 82–97, 2012. → pages 1

[59] G. E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14(8):1771–1800, Aug. 2002. ISSN 0899-7667. → pages 74, 102, 103

[60] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006. → pages 2, 103

[61] G. E. Hinton, S. Osindero, and Y.-W. Teh. A fast learning algorithm for deep belief nets. *Neural Comput.*, 18(7):1527–1554, July 2006. ISSN 0899-7667. → pages 2

[62] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, Nov. 1997. → pages 29, 31, 74

[63] T. Hofmann. Probabilistic latent semantic analysis. In *In Proc. of Uncertainty in Artificial Intelligence, UAI99*, pages 289–296, 1999. → pages 41

[64] B. Hu, Z. Lu, H. Li, and Q. Chen. Convolutional neural network architectures for matching natural language sentences. In *Advances in Neural Information Processing Systems 27*, pages 2042–2050. 2014. → pages 29

[65] P.-S. Huang, X. He, J. Gao, L. Deng, A. Acero, and L. Heck. Learning deep structured semantic models for web search using clickthrough data. In *Proceedings of the 22Nd ACM International Conference on Conference on Information & Knowledge Management*, CIKM '13, pages 2333–2338. ACM, 2013. → pages 27, 28, 30, 40, 68

[66] H. Jaeger. *The "echo state" approach to analysing and training recurrent neural networks*. GMD Report 148, GMD - German National Research Institute for Computer Science, 2001. → pages 14, 25

[67] H. Jaeger. *Short term memory in echo state networks*. GMD Report 152, GMD - German National Research Institute for Computer Science, 2001. → pages 14, 25

[68] H. Jaeger. A tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the echo state network approach. Technical report, Fraunhofer Institute for Autonomous Intelligent Systems (AIS) since 2003: International University Bremen, 2005. → pages 15, 16, 17

[69] H. Jaeger and H. Haas. Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *Science*, 304(5667):78–80, 2004. → pages 14, 16, 17, 25

[70] K. Järvelin and J. Kekäläinen. Ir evaluation methods for retrieving highly relevant documents. In *Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR, pages 41–48. ACM, 2000. → pages 40

[71] S. Ji, Y. Xue, and L. Carin. Bayesian compressive sensing. *Signal Processing, IEEE Transactions on*, 56(6):2346–2356, 2008. → pages 72, 81, 105

[72] S. Ji, D. Dunson, and L. Carin. Multitask compressive sensing. *Signal Processing, IEEE Transactions on*, 57(1):92–106, 2009. → pages 9, 70, 72, 73, 81, 95, 97, 105

[73] N. Kalchbrenner, E. Grefenstette, and P. Blunsom. A convolutional neural network for modelling sentences. *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, June 2014. → pages 29

[74] A. Karpathy, J. Johnson, and L. Fei-Fei. Visualizing and understanding recurrent networks. *arXiv preprint arXiv:1506.02078*, 2015. accessed on December 13, 2016. → pages 29

[75] R. Kiros, Y. Zhu, R. Salakhutdinov, R. S. Zemel, A. Torralba, R. Urtasun, and S. Fidler. Skip-thought vectors. *Advances in Neural Information Processing Systems (NIPS)*, 2015. → pages 28, 42

[76] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, pages 1097–1105. 2012. → pages 1, 4

[77] Q. V. Le and T. Mikolov. Distributed representations of sentences and documents. *Proceedings of the 31st International Conference on Machine Learning*, pages 1188–1196, 2014. → pages 27, 28, 41, 42

[78] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov 1998. http://yann.lecun.com/exdb/mnist/ , accessed on December 13, 2016. → pages 79

[79] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, May 2015. → pages 1

[80] M. Lustig, D. L. Donoho, J. M. Santos, and J. M. Pauly. Compressed sensing mri. *IEEE Signal Processing Magazine*, 25(2):72–82, 2008. → pages 7

[81] H. Mamaghanian, N. Khaled, D. Atienza, and P. Vandergheynst. Compressed sensing for real-time energy-efficient ecg compression on wireless body sensor nodes. *Biomedical Engineering, IEEE Transactions on*, 58(9):2456–2466, 2011. → pages 7

[82] N. Meinshausen and B. Yu. Lasso-type recovery of sparse representations for high-dimensional data. *The Annals of Statistics*, 37:246 – 270, 2009. → pages 9, 70

[83] D. Merhej, C. Diab, M. Khalil, and R. Prost. Embedding prior knowledge within compressed sensing by neural networks. *IEEE Transactions on Neural Networks*, 22(10):1638 –1649, oct. 2011. → pages 72, 73, 74

[84] G. Mesnil, X. He, L. Deng, and Y. Bengio. Investigation of recurrent-neural-network architectures and learning methods for spoken language understanding. In *Proc. INTERSPEECH*, Lyon, France, August 2013. → pages 5, 30, 74

[85] T. Mikolov, M. Karafiát, L. Burget, J. Cernockỳ, and S. Khudanpur. Recurrent neural network based language model. In *Proc. INTERSPEECH*, pages 1045–1048, Makuhari, Japan, September 2010. → pages 5, 14, 30, 35, 74

[86] T. Mikolov, S. Kombrink, L. Burget, J. Cernocky, and S. Khudanpur. Extensions of recurrent neural network language model. In *Proc. IEEE ICASSP*, pages 5528–5531, Prague, Czech, May 2011. → pages 16

[87] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013. accessed on December 13, 2016. → pages 28

[88] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. In *International Conference on Learning Representations (ICLR)*, 2013. URL arXiv:1301.3781. → pages 5, 41

[89] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Proceedings of Advances in Neural Information Processing Systems*, pages 3111–3119, 2013. → pages 28

[90] A. Milenkovi, C. Otto, and E. Jovanov. Wireless sensor networks for personal health monitoring: Issues and an implementation. *Computer Communications (Special issue: Wireless Sensor Networks: Performance, Reliability, Security, and Beyond*, 29:2521–2533, 2006. → pages 7

[91] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, 2015. → pages 1

[92] A. Mousavi, A. B. Patel, and R. G. Baraniuk. A deep learning approach to structured signal recovery. *arXiv*, abs/1508.04065, 2015. http://arxiv.org/abs/1508.04065 , accessed on December 13, 2016. → pages 72, 74

[93] Y. Nesterov. A method of solving a convex programming problem with convergence rate o (1/k2). *Soviet Mathematics Doklady*, 27:372–376, 1983. → pages 34, 35, 78

[94] H. Palangi, L. Deng, and R. Ward. Learning input and recurrent weight matrices in echo state networks. In *NIPS Workshop on Deep Learning*, December 2013. http://research.microsoft.com/apps/pubs/default.aspx?id=204701 , accessed on December 13, 2016. → pages 74

[95] H. Palangi, R. Ward, and L. Deng. Using deep stacking network to improve structured compressed sensing with multiple measurement vectors. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 3337–3341, May 2013. → pages 72, 74, 81, 95, 97, 105

[96] H. Palangi, L. Deng, and R. Ward. Recurrent deep-stacking networks for sequence classification. In *Signal and Information Processing (ChinaSIP), 2014 IEEE China Summit International Conference on*, pages 510–514, July 2014. → pages 74

[97] H. Palangi, R. Ward, and L. Deng. Distributed compressive sensing: A deep learning approach. *IEEE Transactions on Signal Processing*, 64(17):4504–4518, Sept 2016. → pages 95, 97

[98] R. Pascanu, T. Mikolov, and Y. Bengio. On the difficulty of training recurrent neural networks. In *ICML 2013*, volume 28 of *JMLR Proceedings*, pages 1310–1318. JMLR.org, 2013. → pages 17, 23, 35

[99] R. Pascanu, T. Mikolov, and Y. Bengio. On the difficulty of training recurrent neural networks. In *Proc. ICML*, Atlanta, GA, June 2013. → pages 14, 24

[100] K. B. Petersen and M. S. Pedersen. The matrix cookbook. Technical report, 2008. → pages 20

[101] R. Řehůřek and P. Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA. http://is.muni.cz/publication/884893/en , accessed on December 13, 2016. → pages 42

[102] M. Research. http://research.microsoft.com/en-us/projects/objectclassrecognition. accessed on December 13, 2016. → pages 79, 86

[103] A. J. Robinson. An application of recurrent nets to phone probability estimation. *IEEE Transactions on Neural Networks*, 5(2):298–305, August 1994. → pages 5, 14, 16, 30, 74

[104] F. Rosenblatt. The perceptron: A perceiving and recognizing automaton. Technical report, Cornell Aeronautical Laboratory Report, 1957. → pages 2

[105] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986. → pages 2

[106] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. → pages 4

[107] H. Sak, A. Senior, and F. Beaufays. Long short-term memory recurrent neural network architectures for large scale acoustic modeling. In *Proceedings of the Annual Conference of International Speech Communication Association (INTERSPEECH)*, 2014. → pages 29

[108] Y. Shen, X. He, J. Gao, L. Deng, and G. Mesnil. A latent semantic model with convolutional-pooling structure for information retrieval. CIKM, November 2014. → pages 28, 30, 40, 68

[109] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–503, 2016. → pages 1

[110] P. Smolensky. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel distributed processing: explorations in the microstructure of cognition, vol. 1*, chapter Information processing in dynamical systems: foundations of harmony theory, pages 194–281. MIT Press, Cambridge, MA, USA, 1986. ISBN 0-262-68053-X. → pages 102

[111] R. Socher, J. Pennington, E. H. Huang, A. Y. Ng, and C. D. Manning. Semi-supervised recursive autoencoders for predicting sentiment distributions. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '11, pages 151–161, 2011. ISBN 978-1-937284-11-4. → pages 28

[112] S. Sukhbaatar, A. Szlam, J. Weston, and R. Fergus. End-to-end memory networks. In *Advances in Neural Information Processing Systems 28*, pages 2440–2448. 2015. → pages 6

[113] I. Sutskever. *Training Recurrent Neural Networks*. PhD thesis, University of Toronto, 2013. → pages 16

[114] I. Sutskever, G. E. Hinton, and G. W. Taylor. The recurrent temporal restricted boltzmann machine. In *NIPS*, pages 1601–1608, 2008. → pages 23

[115] I. Sutskever, J. Martens, G. E. Dahl, and G. E. Hinton. On the importance of initialization and momentum in deep learning. In *ICML (3)'13*, pages 1139–1147, 2013. → pages 34, 78

[116] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Proceedings of Advances in Neural Information Processing Systems*, pages 3104–3112, 2014. → pages 27, 29

[117] M. E. Tipping. Sparse bayesian learning and the relevance vector machine. *J. Mach. Learn. Res.*, 1: 211–244, Sept. 2001. → pages 73

[118] F. Triefenbach, A. Jalalvand, B. Schrauwen, and J.-P. Martens. Phoneme recognition with large hierarchical reservoirs. *Advances in Neural Information Processing Systems*, 2009. → pages 14, 16, 17, 25

[119] J. Tropp. Algorithms for simultaneous sparse approximation. part II: Convex relaxation. *Signal Processing*, 86(3):589 – 602, 2006. → pages 9, 70

[120] J. Tropp, A. Gilbert, and M. Strauss. Algorithms for simultaneous sparse approximation. part I: Greedy pursuit. *Signal Processing*, 86(3):572 – 588, 2006. → pages 9, 70, 97

[121] E. van den Berg and M. P. Friedlander. Probing the pareto frontier for basis pursuit solutions. *SIAM Journal on Scientific Computing*, 31(2):890–912, 2008. → pages 9, 70

[122] N. Vaswani. Ls-cs-residual (ls-cs): Compressive sensing on least squares residual. *IEEE Transactions on Signal Processing*, 58(8):4108–4120, 2010. → pages 72, 74, 81

[123] P. Vincent. A connection between score matching and denoising autoencoders. *Neural Comput.*, 23 (7):1661–1674, July 2011. → pages 74

[124] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. Extracting and composing robust features with denoising autoencoders. *ICML*, pages 1096–1103, 2008. → pages 74

[125] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan. Show and tell: A neural image caption generator. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR, Boston, MA, USA*, pages 3156–3164, 2015. → pages 1

[126] D. P. Wipf and B. D. Rao. An empirical bayesian strategy for solving the simultaneous sparse approximation problem. *Signal Processing, IEEE Transactions on*, 55(7):3704–3716, 2007. → pages 9, 70, 72, 73

[127] D. H. Wolpert. Stacked generalization. *Neural Networks*, 5:241–259, 1992. → pages 14

[128] K. Xu, J. Ba, R. Kiros, K. Cho, A. C. Courville, R. Salakhutdinov, R. S. Zemel, and Y. Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *Proceedings of the 32nd International Conference on Machine Learning, ICML, Lille, France*, pages 2048–2057, 2015. → pages 1

[129] D. Yu and L. Deng. Deep learning and its applications to signal and information processing [exploratory dsp]. *IEEE Signal Processing Magazine*, 28(1):145 –154, jan. 2011. → pages 1, 2, 68

[130] D. Yu and L. Deng. Efficient and effective algorithms for training single-hidden-layer neural networks. *Pattern Recognition Letters*, 33(5):554–558, 2012. → pages 18, 21, 102

[131] D. Yu and L. Deng. *Automatic Speech Recognition - A Deep Learning Approach*. Springer, 2015. → pages 1

[132] J. Zhang, S. Liu, M. Li, M. Zhou, and C. Zong. Bilingually-constrained phrase embeddings for machine translation. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (ACL) (Volume 1: Long Papers)*, pages 111–121, Baltimore, Maryland, 2014. → pages 29

[133] Z. Zhang and B. D. Rao. Sparse signal recovery with temporally correlated source vectors using sparse bayesian learning. *Selected Topics in Signal Processing, IEEE Journal of*, 5(5):912–926, 2011. → pages 9, 70, 72, 73, 81, 95, 97, 105

[134] P. Zhao and B. Yu. On model selection consistency of lasso. *J. Mach. Learn. Res.*, 7:2541–2563, 2006. ISSN 1532-4435. → pages 9, 70

[135] Y. Zhu, R. Kiros, R. Zemel, R. Salakhutdinov, R. Urtasun, A. Torralba, and S. Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. *arXiv preprint arXiv:1506.06724*, 2015. accessed on December 13, 2016. → pages 28, 42