ACTIVE LABELING IN DEEP LEARNING AND ITS APPLICATION TO EMOTION PREDICTION

A Dissertation Presented to the Faculty of the Graduate School

University of Missouri-Columbia

In Partial Fulfillment

of the Requirements for the Degree of

Ph.D. in Computer Science

by

DAN WANG

Advisor: Dr. Yi Shang

DECEMBER 2013

The undersigned, appointed by the Dean of the Graduate School, have examined the thesis entitled

ACTIVE LABELING IN DEEP LEARNING AND ITS APPLICATION TO EMOTION PREDICTION

presented by Dan Wang

a candidate for the degree of Doctor of Philosophy

and hereby certify that in their opinion it is worthy of acceptance.

Dr. Yi Shang

Dr. Wenjun Zeng

Dr. Dale Musser

Dr. Tony Han

ACKNOWLEDGEMENTS

First and foremost, I would like to show my appreciation to my advisor, Dr. Yi Shang, for his continuous support, guidance, and encouragement throughout my PhD study and research. His deep insights and broad knowledge constantly keep my research on the right track. I am also impressed by his patience and tireless help. The weekly individual and group meetings with him are my greatest pleasure. He is a supportive advisor not only on my study and research, but also on my life and career path.

I would like to thank my committee members, Dr. Wenjun Zeng, Dr. Dale Musser, and Dr. Tony Han for reviewing my dissertation, attending my comprehensive exam and defense, and offering all the constructive suggestions, guidance, and comments.

Thanks to Jodette Lenser and Sandra Moore for taking care of my academic needs. Thanks to all the faculty and staff in the Department of Computer Science at University of Missouri, for providing me with such a nice platform.

I also want to thank all the people in our research group, especially Peng Zhuang, Qi Qi, and Chao Fang, for their selfless help. It was fun to exchange ideas and thoughts with these great guys. Many thanks go to Qia Wang in Dr. Zeng's group and Liyang Rui in Dr. Ho's group. I will never forget the good time when we worked together as a team.

Finally, I would like to thank my parents for their support from the other side of the Earth.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Recent breakthroughs in deep learning have made possible the learning of deep layered hierarchical representations of sensory input. Stacked restricted Boltzmann machines (RBMs), also called deep belief networks (DBNs), and stacked autoencoders are two representative deep learning methods. The key idea is greedy layer-wise unsupervised pre-training followed by supervised fine-tuning, which can be done efficiently and overcomes the difficulty of local minima when training all layers of a deep neural network at once. Deep learning has been shown to achieve outstanding performance in a number of challenging real-world applications.

Existing deep learning methods involve a large number of meta-parameters, such as the number of hidden layers, the number of hidden nodes, the sparsity target, the initial values of weights, the type of units, the learning rate, etc. Existing applications usually do not explain why the decisions were made and how changes would affect performance. Thus, it is difficult for a novice user to make good decisions for a new application in order to achieve good performance. In addition, most of the existing works are done on simple and clean datasets and assume a fixed set of labeled data, which is not necessarily true for real-world applications.

The main objectives of this dissertation are to investigate the optimal meta-parameters of deep learning networks as well as the effects of various data pre-processing techniques, propose a new active labeling framework for cost-effective selection of labeled data, and apply deep learning to a real-world application – emotion prediction via physiological sensor data, based on real-world, complex, noisy, and heterogeneous sensor data. For meta-parameters and data pre-processing techniques, this study uses the benchmark MNIST digit recognition image dataset and a sleep-stage-recognition sensor dataset and empirically compares the deep network’s performance with a number of different meta-parameters and decisions, including raw data vs. pre-processed data by Principal Component Analysis (PCA) with or without whitening, various structures in terms of the number of layers and the number of nodes in each layer, stacked RBMs vs. stacked autoencoders. For active labeling, a new framework for both stacked RBMs and stacked autoencoders is proposed based on three metrics: least confidence, margin sampling, and entropy. On the MINIST dataset, the methods outperform random labeling consistently by a significant margin. On the other hand, the proposed active labeling methods perform similarly to random labeling on the sleep-stage-recognition dataset due to the noisiness and inconsistency in the data. For the application of deep learning to emotion prediction via physiological sensor data, a software pipeline has been developed. The system first extracts features from the raw data of four channels in an unsupervised fashion and then builds three classifiers to classify the levels of arousal, valence, and liking based on the learned features. The classification accuracy is 0.609, 0.512, and 0.684, respectively, which is comparable with existing methods based on expert designed features.

Chapter 1

Introduction

1.1 Motivations

Shallow neural networks with an input layer, a single hidden layer, and an output layer require more computational elements or are hard to model complex concepts and multi-level abstractions. In contrast, multi-layer neural networks provide better representational power and could derive more descriptive multi-level models due to their hierarchical structures, with each higher layer representing higher-level abstraction of the input data. Unfortunately, it is difficult to train all layers of a deep neutral network at once [1]. With random initial weights, the learning is likely to get stuck in local minima.

The breakthrough happened in 2006, when Hinton introduced a novel way called Deep Belief Networks (DBNs) to train multi-layer neural networks to learn features from unlabeled data [2]. A DBN trains a multi-layer neural network in a greedy fashion, each layer being a restricted Boltzmann machine (RBM) [3]. The trained weights and biases in each layer can be thought of as the features or filters learned from the input data. Then the weights and biases act as the initial values for the supervised fine-tuning using backpropogation. In short, a DBN discovers features on its own and does semi-supervised learning by modeling unlabeled data first in an unsupervised way and then incorporates labels in a supervised fashion.

A similar approach to the above mentioned stacked RBMs was introduced by Bengio [1] using stacked autoencoders instead of stacked RBMs. Each autoencoder is a one-hidden-layer neural network with the same number of nodes in the input and output layers, which tries to learn an approximation to the identity function by applying regular backpropagation. The hidden layer with fewer units (or with a sparsity term) is a compressed representation of the input data, thus discovering interesting structure about the data. An autoencoder serves as a building block for deep neural networks similar to an RBM. The supervised fine-tuning stage could also be applied to stacked autoencoders by incorporating a softmax on top to train a classifier.

One issue with current applications on deep learning is the lack of explanations about how to achieve a large number of meta-parameters that yield good results. Firstly, the authors usually fail to present the meta-parameters tuning process in their literature, particularly, the number of hidden layers and the number of hidden nodes. Thus, it is difficult for novice users to replicate the similar results on different problems. Secondly, although deep learning has the ability to learn features automatically from raw data, it may be interesting to investigate the effects of various data pre-processing techniques, either hand engineered features or commonly used dimension reduction algorithms such as principle component analysis (PCA) with or without whitening. Thirdly, RBMs and autoencoders are both designed to represent input data in a compressed fashion, but whether they perform the same in different problems remains questionable.

Another practical problem with deep learning framework is how to choose samples to be labeled. Since the labeled data are scarce and expensive, it makes sense to choose the most informative samples to be labeled and then deep learning fine-tunes the classification models using these labeled data. The existing researches on deep learning assume the labeled data are passive, either available there already or obtained from the samples randomly chosen to be labeled by human experts. The former may not be practical; the latter does not yield the best results. Suppose there is a budget on the number of samples to be labeled. It is expected to produce better classification performance in most cases (but the same or even worse performance on some datasets) for an active labeling algorithm to always select the most challenging samples to be labeled. This falls into the well-defined active learning framework [4]. However, how to estimate which samples are more informative or more challenging remains unexplored with deep learning.

Last, despite the power of deep learning, blindly applying it to real scenarios does not yield satisfactory results, except on toy datasets. A pipeline of applying deep learning to actual problems is desired, which includes raw data pre-processing, raw data selection and division, normalization, randomization, and deep learning training and classification. To map physiological data to emotions using deep learning could serve as a good example to articulate the process. Physiological data is collected by sensors as a means of human-computer interaction by monitoring, analyzing and responding to psychophysiological activities [5]. The data types include a lot of channels, useful to predict human's physical activities, emotions, and even potential diseases. Since the goals vary a lot, it is difficult to know useful features without expert's knowledge. Deep learning is promising to overcome the barrier by extracting useful features automatically. Moreover, deep learning, acting as a semi-supervised machine learning algorithm, takes advantage of scarce labeled data and abundant unlabeled data in this scenario.

1.2 Contributions

This dissertation makes three major contributions to the area of deep learning that are summarized as follows.

First, I investigate the optimal meta-parameters of deep learning networks as well as the effects of various data pre-processing techniques. This study uses the benchmark MNIST digit recognition image dataset and a sleep-stage-recognition sensor dataset and empirically compares the deep learning network's performance with quite a few combinations of settings, including raw data vs. pre-processed data by Principal Component Analysis (PCA) with or without whitening for MNIST and hand extracted features for the sleep stage dataset, various structures in terms of the number of layers and the number of nodes in each layer, different building blocks including stacked RBMs vs. stacked autoencoders. The process is presented as a guideline for future deep learning applications to tune meta-parameters and data pre-processing.

Second, I propose a new active labeling framework for deep learning including both stacked RBMs and stacked autoencoders based on three metrics: least confidence, margin sampling, and entropy, for cost-effective selection of labeled data. Then I investigate the performance of the active labeling deep learning technique in all the three metrics, compared to a random labeling strategy, on the raw data and features of the MNIST dataset and the sleep stage dataset.

Last, I develop a pipeline to apply deep learning to emotion prediction via physiological data, based on real-world, complex, noisy, and heterogeneous sensor data. The system first extracts features from the raw data of four channels in an unsupervised fashion and then builds three classifiers to classify the levels of arousal, valence, and liking based on the learned features.

1.3 Outline of the Dissertation

This dissertation is organized into the following chapters.

In Chapter 1, the motivations and the scope of the proposed research are introduced.

Chapter 2 presents the background knowledge and state-of-the-art techniques of deep learning, and introduces two types of building blocks and their performance on a benchmark hand written digit dataset MNIST.

In Chapter 3, deep learning is revisited to explore the optimal settings of the input pre-processing, neural network structure, and the learning unit types.

In Chapter 4, an active labeling deep learning framework is proposed to choose the most informative samples to be labeled. Three criteria are introduced in the active labeling framework.

In Chapter 5, the application of DBNs on the physiological dataset DEAP [6] is developed and its performance is evaluated.

Chapter 6 concludes the dissertation.

Chapter 2

Deep Learning Background and General Knowledge

2.1 Background

An example of shallow neural networks with an input layer, a single hidden layer, and an output layer, as shown in Fig. 1, can be trained with backpropagation for classification or regression. Theoretically a shallow net can approximate any functions as long as it has enough units in the hidden layer [7, 8], but the size of hidden units grows exponentially with the input layer size [9]. So many units are needed in the hidden layer of a shallow neural network for a representative model is because the single hidden layer is just one step away from the input layer and the output layer, which is forced to translate the raw data from the input layer to complex features that can be used for classification in the output layer. Too many hidden units increase computational complexity, and even worse, easily result in overfitting, especially when the training set size is relatively small.

In contrast, a deep network with two or more hidden layers provides better representational power and thus obtains more descriptive models thanks to feature sharing and abstraction. A lower layer's features are reused by the layer above it, whereas a higher layer represents higher-level abstraction of data. In deep nets lower layers are relaxed to learn simple or concrete features whereas higher layers tend to represent complex or abstract features. For example, to transform the raw input images of handwritten digits into three gradually higher levels of representations, the first layer could feature key dots, the second layer could represent lines and curves, and the features learned by the third layer are closer to more meaningful digit parts.

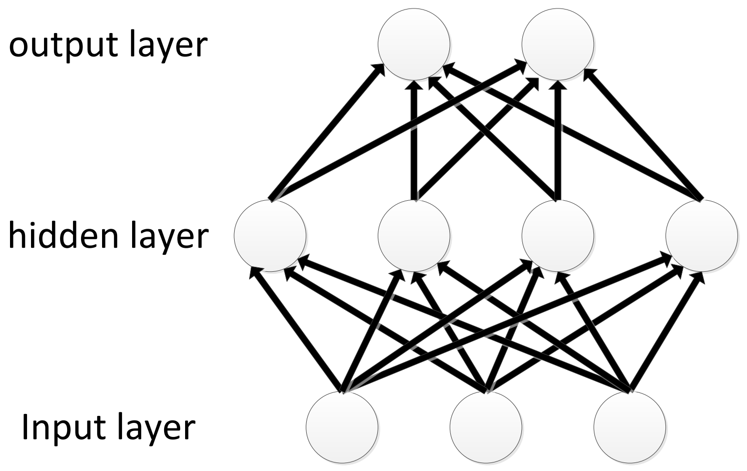


Fig. 1 An example of shallow neutral networks

Unfortunately, it is difficult to train deep neutral networks all layers at once [1]. With large initial weights, the learning is likely to get stuck in local minima. With small initial weights, the gradients are tiny, so the training takes forever to converge.

Hinton [2, 10] proposed deep belief networks (DBNs) to overcome the difficulties by constructing multilayer restricted Boltzmann machines (RBMs) [3] and training them layer-by-layer in a greedy fashion. Since the network consists of a stack of RBMs, it is also called stacked RBMs. We use DBN and stacked RBMs interchangeably in this dissertation, as opposed to stacked autoencoders to be introduced later.

The training process has two stages. The first is the pre-training stage, in which no labels are involved and the training is done in an unsupervised way. The training starts off with the bottom two layers to obtain features in hidden layer 1 from the input layer.

Then the training moves up to hidden layer 1 and layer 2, treating layer 1 as the new input to get its features in layer 2. The greedy layer-wise training is performed until reaching the highest hidden layer. The first stage trains a generative model as weights between layers to capture the raw input’s features, resulting in better starting point for the second stage than randomly assigned initial weights. The second is the fine-tuning stage. In this stage, a new layer is put on top of the stacked RBMs of the first stage to construct a discriminative model. The overall schema of a DBN with three hidden layers is shown in Fig. 2.

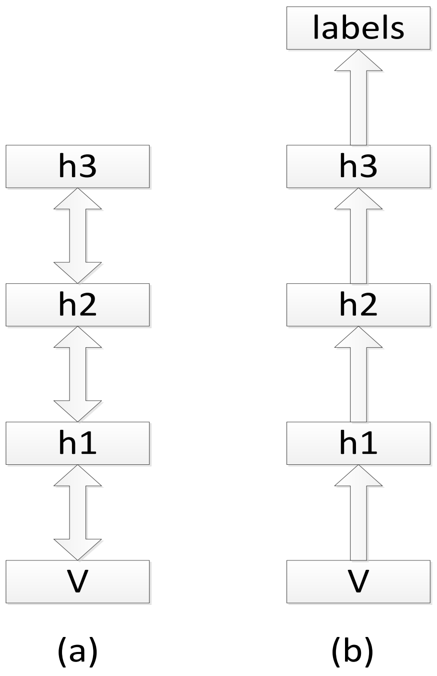


Fig. 2 A DBN schema with three hidden layers. (a) The pre-training stage without labels involved (b) The fine-tuning stage

A variation to the above mentioned deep learning algorithm was introduced by Bengio[1] using stacked autoencoders. An autoencoder neural network is an unsupervised learning algorithm trying to learn an approximation to the identity function by applying regular backpropagation. The hidden layer with fewer nodes (or with a sparsity term) than the input layer learns a compressed representation of the input data, aiming at the same goal as a hidden layer does in stacked RBMs. To exploit stacked autoencoders to do deep learning for a classification task, it also involves an unsupervised pre-training stage and a supervised fine-tuning stage. In the pre-training stage, the output of one hidden layer serves as the input for the higher hidden layer, resulting in a stacked hierarchical structure to learn more and more abstract features. In the fine-tuning stage, a softmax layer is added on top of the stacked autoencoders and a regular backpropagation is applied using the learned weights and biases in the first stage as a starting point.

All in all, stacked RBMs and stacked autoencoders only differ in the unsupervised pre-training stage, where different building blocks are used, to attempt to achieve the same functionality.

For simplicity, this dissertation refers stacked RBMs with a softmax on top to “stacked RBMs” or a “DBN”. The same rule applies to “stacked autoencoders”.

2.2 The Building Blocks

2.2.1 Restricted Boltzmann Machine

As the building blocks of DBNs, a restricted Boltzmann machine (RBM) [3] has a visible layer consisting of stochastic, binary nodes as the input and a hidden output layer consisting of stochastic, binary feature detectors as the output, connected by symmetrical weights between nodes in different layers. RBMs have two key features. Firstly, there are no connections between the nodes in the same layer, making tractable learning possible.

Second, the hidden units are conditionally independent given the visible layer thanks to the undirected connections, so it is fast to get an unbiased sample from the posterior distribution. A graphical depiction of an RBM is shown in Fig. 3.

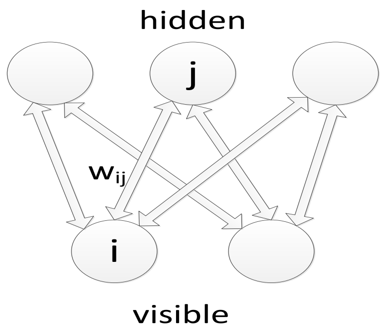


Fig. 3 Graphical depiction of an RBM

A joint configuration (v, h) of the visible and hidden nodes can be represented by an energy function given by

$$E(v,h) = -\sum_{i,j} v_i h_j w_{ij} - \sum_i b_i v_i - \sum_j b_j h_j \qquad (1)$$

where $E(v,h)$ is the energy with configuration v on the visible nodes and h on the hidden nodes, v_i is the binary state of visible node i, h_j is the binary state of hidden node j, w_{ij} is the weight between node i and j, and the bias terms are b_i for the visible nodes and b_j for the hidden nodes (biases not shown in Fig. 3).

The probability of a given joint configuration depends on the energy of that joint configuration compared to the energy of all joint configurations, specified by

$$p(v,h) \propto e^{-E(v,h)} \qquad (2)$$

Concretely its probability is determined by normalizing it by a partition function Z, as shown in

$$p(v,h) = \frac{e^{-E(v,h)}}{Z} \tag{3}$$

$$Z = \sum_{u,g} e^{-E(u,g)} \tag{4}$$

The formula implies that the lower energy a configuration has, the higher probability it would occur.

The summation along all the hidden units produces the probability of a configuration of the visible units.

$$p(v) = \frac{\sum_h e^{-E(v,h)}}{Z} \tag{5}$$

The binary nodes of the hidden layer are Bernoulli random variables. The probability that node h_j is activated, given visible layer v, can be derived from (4) as

$$P(h_j = 1 \mid v) = \sigma(b_j + \sum_i w_{ij} v_i) \tag{6}$$

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{7}$$

where $\sigma(\cdot)$ is called the sigmoid logistic function.

The probability that node v_i is activated, given hidden layer h, can be calculated in a similar way as follows

$$P(v_i = 1 \mid h) = \sigma(b_i + \textstyle\sum_j w_{ij} h_j) \tag{8}$$

It is intractable to compute the gradient of the log likelihood of v directly. Therefore, [11] proposed contrastive divergence by doing k iterations of Gibbs sampling to approximate it. Note from (8) it is easy to get an unbiased sample of the visible layer given a hidden vector because there are no direct connections between visible units in an RBM. The algorithm starts with a training vector on the units in the visible layer, then uses the vector to update all the hidden units in parallel, samples from the hidden units, and uses these samples to update all the visible units in parallel to get the reconstruction. This process is applied k (often $k = 1$) iterations to obtain the change to w_{ij} as below

$$\frac{\partial log p(v)}{\partial w_{ij}} \approx \Delta w_{ij} = \epsilon(< v_i h_j >^0 - < v_i h_j >^k) \tag{9}$$

where $<\cdot>^t$ is the average over a given size of samples when working with minibatches (described later) at a contrastive divergence iteration t and ϵ is the learning rate. The update rule to the biases takes the similar form. Fig. 4 illustrates how to update an RBM with the contrastive divergence with k = 1 (CD1).

The above-mentioned rule works, but a few tricks are used to accelerate the learning process and/or prevent overfitting. The three mostly commonly used techniques are minibatch, momentum, and weight decay [12].

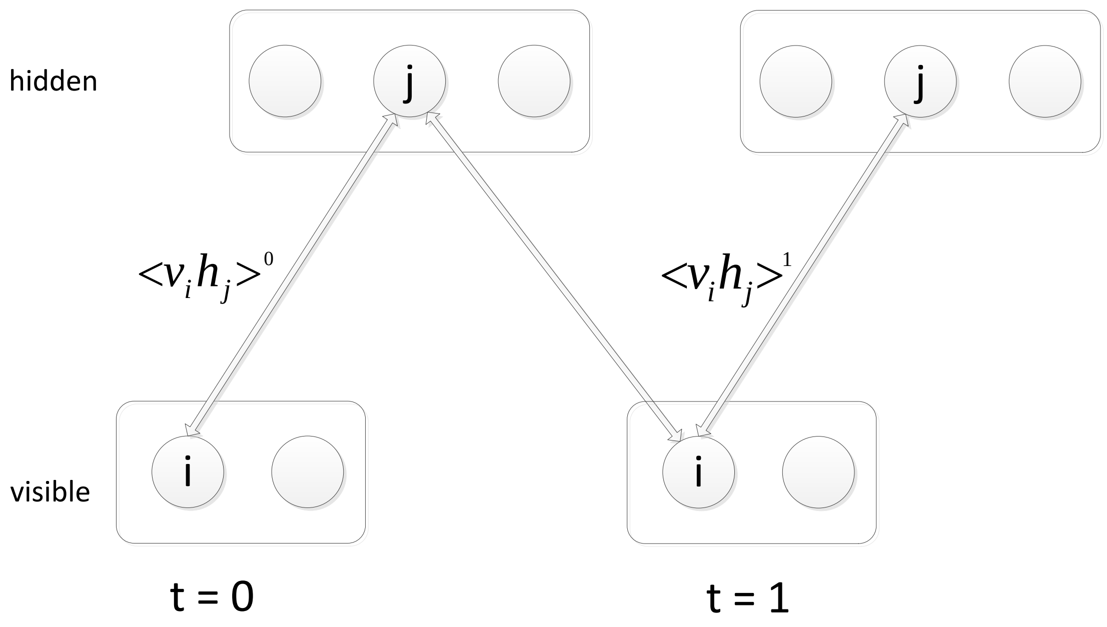


Fig. 4 Updating an RBM with contrastive divergence (k = 1)

Minibatch is a minor variation of (9) in which w_{ij} is updated by taking the average over a small batch instead of a single training vector. This produces two advantages. Firstly minibatch works with a less noisy estimate of the gradient since it takes the average and the outliers in the training vectors does not impact much. Secondly it allows a matrix by matrix product instead of a vector by matrix product, which can be taken advantage by modern GPUs or Matlab to speed up the computation. However, it is a bad idea to make the minibatch size too large because the number of updates will decrease accordingly, eventually resulting in inefficiency.

Momentum is used to speed up learning by simulating a ball moving on a surface. It is an analogy to the acceleration as if w_{ij} were the distance and Δw_{ij} were the velocity. Instead of using the estimated gradient to change weights directly as shown in (9), the momentum method uses it to change the velocity of weights change.

$$\Delta\theta_i(k) = v_i(k) = \alpha v_i(k-1) - \epsilon \frac{dE}{d\theta_i}(k) \tag{10}$$

where α is a hyper-parameter to control the weight given to the previous velocity.

Weight decay is a standard L2 regularization to prevent the weights from getting too large. The updated rule is changed to

$$\Delta\theta_i(k) = v_i(k) = \alpha v_i(k-1) - \epsilon(\frac{dE}{d\theta_i}(k) + \lambda\theta_i(k)) \tag{11}$$

where λ is the weight cost which controls how much penalty should be applied to weight $\theta_i(k)$.

Moreover, a technique called early stopping is often exploited to prevent the model from overfitting. The root mean squared error (RMSE) between the input and its reconstruction on validation set (if available) or training set often acts as the loss function. Then the constant increase of the RMSE indicates the model is overfitting so the training should stop.

2.2.2 Autoencoder

An autoencoder [13-15] is an unsupervised learning algorithm that attempts to learn an identity function by setting the outputs to be equal to the inputs (or at least minimizing the reconstruction error), shown in Fig. 5. When the number of nodes in the hidden layer is larger than or equal to the number of nodes in the input/output layers, it is trivial to learn an identity function. By placing some restrictions on the network to make it as a regularized autoencoder, we can learn compressed representation of the input data. The easiest way to do so is limit the number of nodes in the hidden layer to force fewer nodes to represent features. Actually the discovered low-dimensional features will be very similar to PCA. In this sense, the mapping from the input layer to the hidden layer is called encoding and the mapping from the hidden layer to the output layer is called decoding.

In summary, the basic autoencoder tries to find

$$\underset{W,b}{argmin} J(W,b) = (\left|h_{W,b}(x) - x\right|) \tag{12}$$

where x is the input, W is the weights, b is the biases, and h is the function mapping input to output.

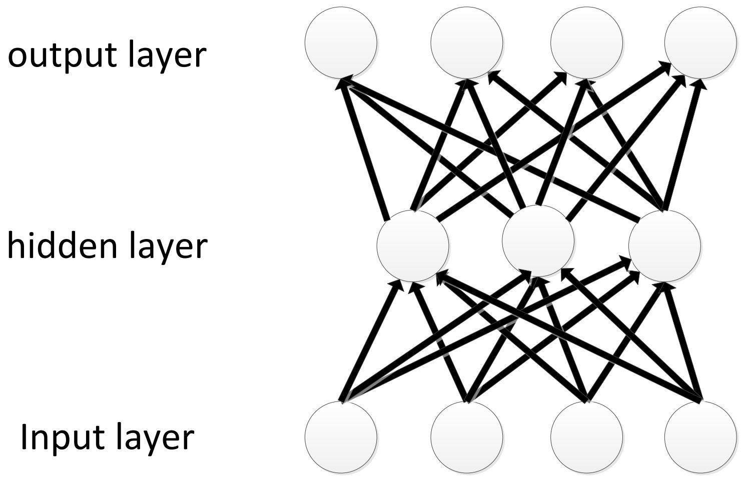


Fig. 5 An autoencoder

The argument above does not hold when the number of hidden nodes is large. But even when it is large, we can still apply a different kind of regularization called sparsity on the hidden nodes, to force them to learn compressed representations. Specifically, let

$$\hat{p}_j = \frac{1}{m}\sum_{i=1}^{m}\left[a_j^{(2)}(x^{(i)})\right] \tag{13}$$

be the average activation of hidden unit j over the training set of size m. The objective is to approximate the sparsity parameter p to $\hat{p}$. The extra penalty term to (12) to measure the difference between p and $\hat{p}$ could be

$$\sum_{j=1}^{s_2} p log \frac{p}{\hat{p}_j} + (1-p) log \frac{1-p}{1-\hat{p}_j} \tag{14}$$

where j is a hidden node, s_2 is the number of nodes in the hidden layer. The value reaches its minimum of 0 at $\hat{p}_j = p$ and blows up as $\hat{p}_j$approaches 0 or 1.

The overall cost function now becomes

$$J_{sparse}(W,b) = J(W,b) + \beta(\sum_{j=1}^{s_2} p log \frac{p}{\hat{p}_j} + (1-p) log \frac{1-p}{1-\hat{p}_j}) \tag{15}$$

Now we need to do a backpropagation to update W and b. The full derivation is similar to that on an RBM.

2.3 Unsupervised Learning Stage

A single RBM or autoencoder may not be good enough to model features in real data. Iteratively we could build another layer on top of the trained RBM or autoencoder by treating the learned feature detectors in the hidden layer as visible input layer for the new layer, as shown in Fig. 2(a). The unsupervised learning stage has no labels involved and solely relies on unlabeled data itself. The learned weights and biases reflect the features of the data and then will be used as the starting point for the fine-tuning supervised learning stage.

2.4 Supervised Learning Stage

To train a deep learning network as a discriminative model, the supervised learning stage adds a label layer and removes the links in the top to down direction, or decoding layers called by [2], as shown in Fig. 2(b). Then the standard backpropogation is executed. The goal is to minimize the classification errors given the labels of all or partial samples.

Since the newly added top layer has one and only one unit that can be activated at a time and the probabilities of turning each unit on must add up to 1, (6) for binary units does not apply any more but it can be generalized to K alternative states by a softmax function.

$$p_j = \frac{e^{x_j}}{\sum_{i=1}^{K} e^{x_i}} \tag{16}$$

where $x_j = \mathrm{P}(\mathrm{h_j} = 1 \mid \mathrm{v})$.

The weights and biases are initialized as the values learned in the pre-training stage, except for those between the original top layer and the newly added top layer, which are randomly initialized. In the first few iterations, the training tackles the randomly initialized weights and biases between the top two layers by keeping other weights and biased fixed. The reason to do this is because the initial values learned from the pre-training are quite close to the representative features of the training data but the randomly initialized values are far from optimal. After the first few iterations, all layers are trained together treating the raw training data as the input and the labels as the output.

If fewer labeled samples are used in the supervised learning stage than the unlabeled samples used in the unsupervised pre-training stage, it is a paradigm of semi-supervised learning [16].

A validation set should be used whenever possible to avoid overfitting the model, as done in the pre-training stage. The difference is now the validation set has labeled samples rather than unlabeled ones.

2.5 Deep Learning Performance Evaluation on MNIST

2.5.1 Stacked RBMs

To evaluate the performance of DBNs and explore how the hyper-parameters settings impact the performance, a few experiments have been carried out on a widely used dataset MNIST.

The MNIST handwritten digits dataset [17] has 70,000 samples in total, conventionally divided into a training set of 50,000 samples, a validation set of 10,000 samples, and a test set of 10,000 samples. The digits have been size-normalized and centered in a 28 by 28 pixels image. The classes are 0 through 9. The MNIST dataset has been broadly used to evaluate the performance of machine learning algorithms.

[18] provided an object-oriented matlab toolbox named DBNToolbox for working with RBMs and DBNs. The toolbox has an abstract class NNLayer and its imeplemtation class RBM to train a single RBM, a class DeepNN to train all layer together in the fine-tuning stage, and a few helper functions. The experiments in this section employed the DBNToolbox and most other experiments of my own proposed algorithms on top of the original DBNs were written based on the toolbox.

The main metric to evaluate the performance in this work is the classification accuracy defined below

$$classification\ accuracy = \frac{number\ of\ correctly\ classified\ samples}{total\ number\ of\ samples} \tag{17}$$

The same network structure 784-500-500-2000-10 as in [2] results in 0.9849 accuracy by the basic experiment, whose parameters are listed in Table 1.

Table 1 DBN parameters on MNIST

Unsupervised pre-training stage	
learning rate	0.05
number of epochs	50
minibatch size	100
momentum	0.5 for the first 5 epochs, 0.9 thereafter
weight cost	0.0002
Supervised fine-tuning stage	
learning rate	0.05
number of epochs	50
minibatch size	1000
number of initial epochs	5

DBNs have the ability to learn features automatically, so it would be helpful to visualize the features learned in the example. Although[19] proposed two techniques called activation maximization and sampling from a unit to show clearer patterns in higher layers, they need to clamp input or somehow use the training set's information. In contrast, it is more likely to show the features of the model itself by simply doing a weighted linear summation over a visible layer to obtain a hidden layer's feature as done in [20].

The figure below shows how each pixel of the input images weighs on each unit in the first layer as an image of the same size as the input images. The weights are scaled to [-1 1]. The unsupervised learned features of the units in the first hidden layer are depicted in Fig. 6.

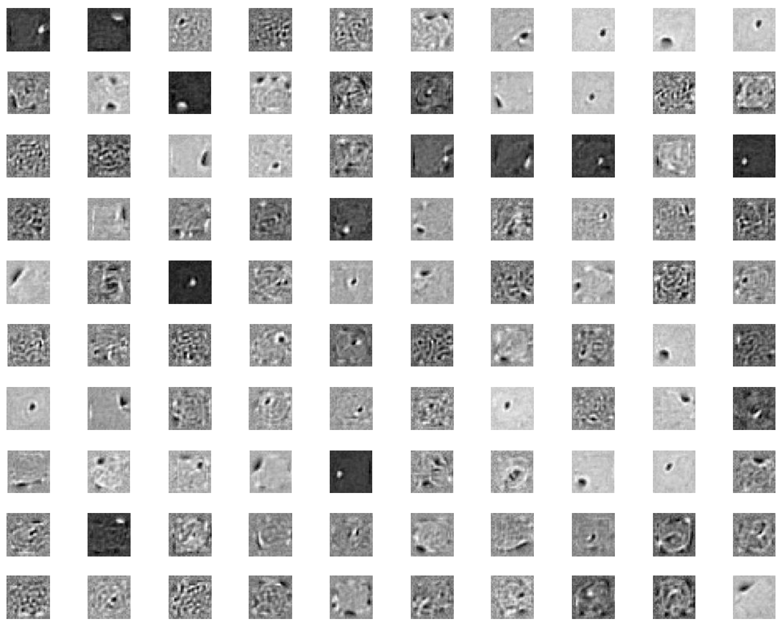


Fig. 6 Learned features of the first hidden layer

The first layer's weights multiplied by the second layer's produce the features learned in the second layer in the input space, as shown in Fig. 7. The weights are scaled to [-1 1], too.

The third layer's features are shown in Fig. 8 by applying the same trick.

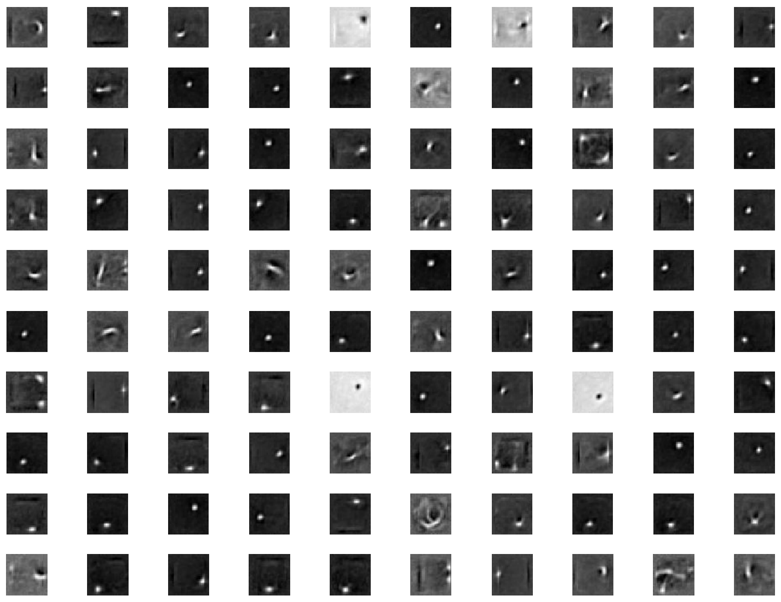


Fig. 7 Learned features of the second hidden layer

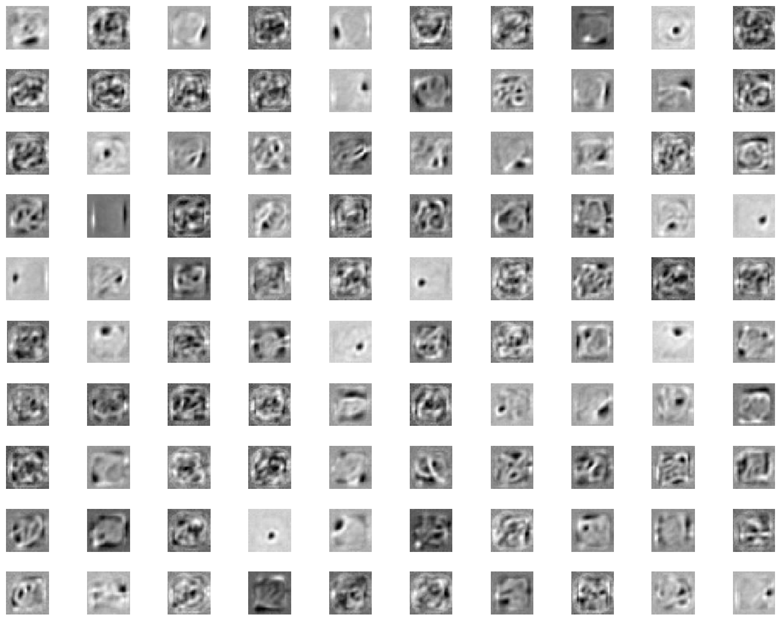


Fig. 8 Learned features of the third hidden layer

It is quite hard to tell if each layer is more abstract than the layer below. Even the first layer does not have clearly human readable patterns. The reason is because the regular RBM is a distributed model, so the features learned are not local or sparse. [21] states that the sparsity regularization is important for the first layer to learn oriented edge filters. A sparse RBM (sRBM) proposed by [20] is an algorithm to make the features learned by RBMs sparse. This is done by adding a regularization term that penalizes a deviation of the expected activation of the hidden units from a fixed level. The update rule for hidden layer's biases becomes

$$\theta_i(k) = \theta_i(k-1) + \alpha v_i(k-1) - \epsilon\left(\frac{dE}{d\theta_i}(k) + \lambda\theta_i(k)\right) + \frac{2}{p} * (p - < \mathrm{P}(\mathrm{h_i} = 1 \mid \mathrm{v}) >) \tag{18}$$

where p is a constant controlling the sparseness of the hidden units. The last term is introduced by sRBM.

Since the original DBNToolbox does not have a sparse implementation of RBMs, I added a protected abstract method UpdateSparcity(obj, visSamples) to NNLayer.m. RBM.m implements the function, which is called at the end of each epoch by NNLayer.m's Train method. The value of p can be set in RBM.m's UpdateSparcity function.

When $p = 0.1$, the features learned in 3 layers of sRBMs are shown as follows. The classification accuracy is 0.9731, slightly worse than the DBN composed of regular RBMs.

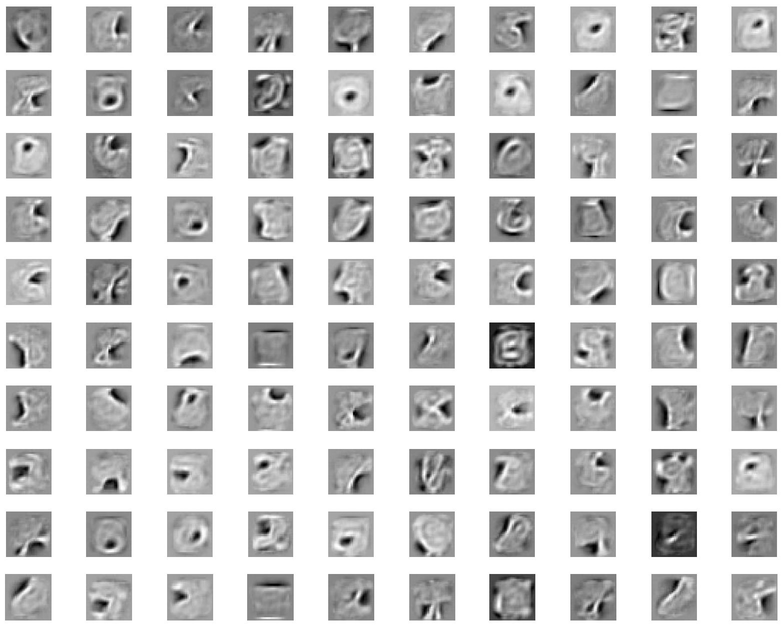


Fig. 9 Learned features of the first hidden layer of sRBM

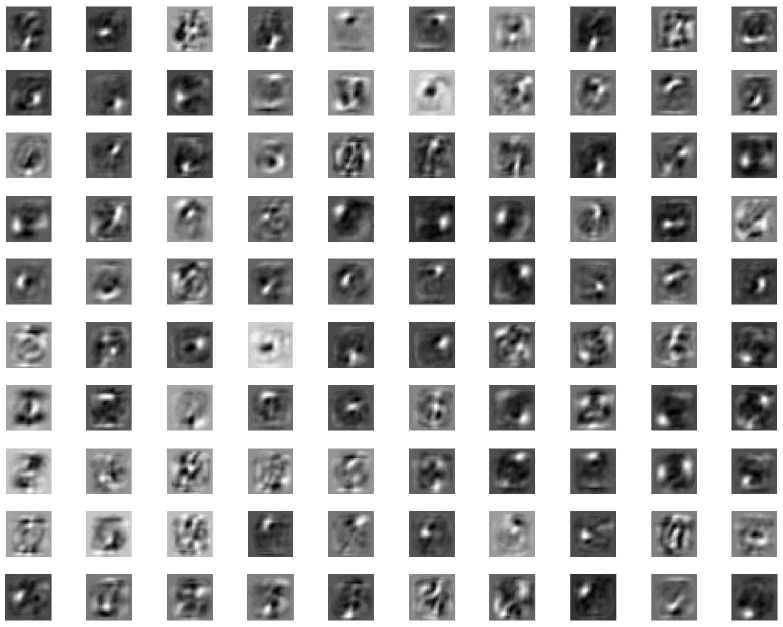


Fig. 10 Learned features of the second hidden layer of sRBM

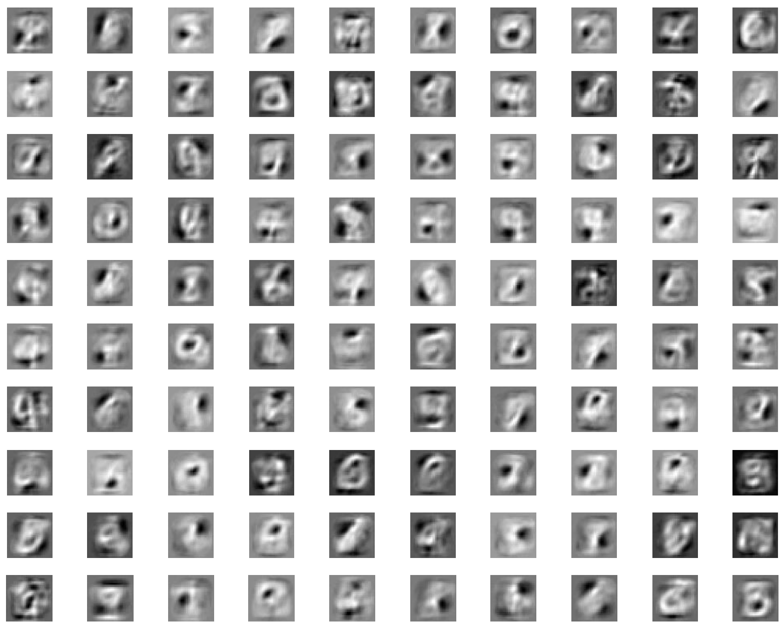


Fig. 11 Learned features of the third hidden layer of sRBM

The features learned by sRBM are sparser and they are more human-readable. The first layer seems like strokes; the second layer looks like digits parts; digits and digit-like shapes are shown in the third layer.

To demonstrate how DBNs work as a semi-supervised learning algorithm, experiments with different numbers of unlabeled samples for pre-training and different numbers of labeled samples for fine-tuning were performed.

Because a small number of labeled samples are used in the experiment, DBNToolbox has a minor bug in this scenario. DeepNN's default value for the parameter miniBatchSize is 1000. When calculating the number of mini-batches, the program floors it to 0 if the sampling size is less than 1000. To overcome the bug, DeepNN's miniBatchSize is set to a number less than or equal to the size of training samples.

Another caveat is when the size of samples is small the distribution of digits could be highly unbalanced. A function was implemented to randomly sample indices for training and validation sets and guarantee each digit has the same size in both sets. DBNToolbox also requires in both sets there is at least one sample from each class.

A 784-150-150-150-10 DBN with 0 to 6400 unlabeled samples and 50 to 250 labeled samples produces the result below. Each point is the average of 10 trials.

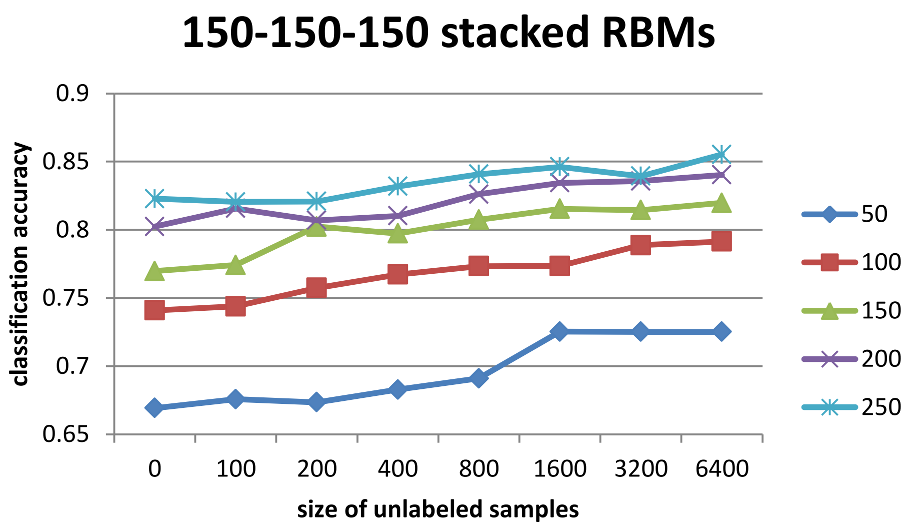


Fig. 12 The predication accuracy of a 784-150-150-150-10 DBN using different numbers of unlabeled and labeled samples. The legends show the numbers of labeled samples.

The unlabeled samples help, especially when the labeled samples are scarce.

2.5.2 Stacked Autoencoders

The similar experiments have been conducted on the same MNIST dataset using stacked autoencoders. The Matlab code from [22] is used as a starting point for all the staked autoencoders implementation. A 784-200-200-10 network with parameters listed in Table 2 yields 0.9781 classification accuracy.

Table 2 Stacked autoencoders parameters on MNIST

Unsupervised pre-training stage	
sparsity object	0.1
sparsity penalty term	3
weight decay	0.003
maximum iteration	200
Supervised fine-tuning stage	
weight decay	0.003
maximum iteration	200

The learned features in the first and second hidden layers are depicted in Fig. 13 and Fig. 14. The first layer seems to detect edges and the second layer seems to detect contours, which are similar to the features learned by the first and third layers of stacked sRBMs as shown in Fig. 9 and Fig. 11.

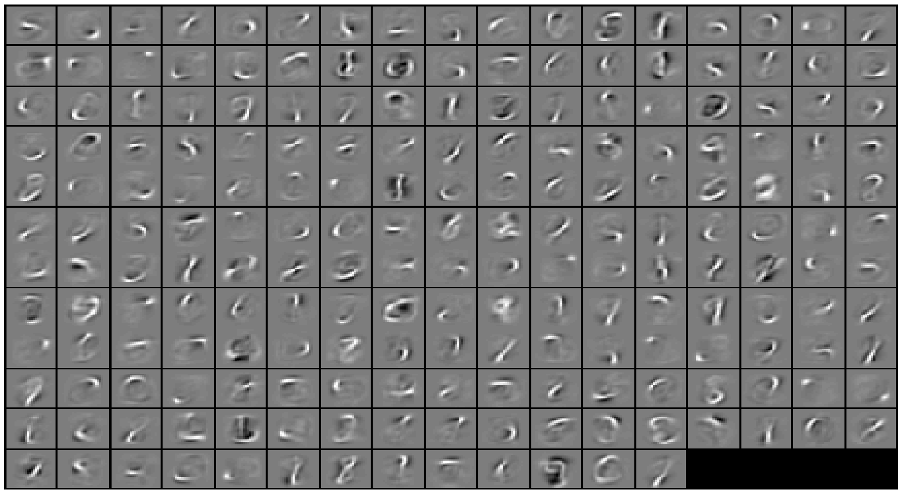


Fig. 13 First hidden layer of stacked autoencoders detects edges

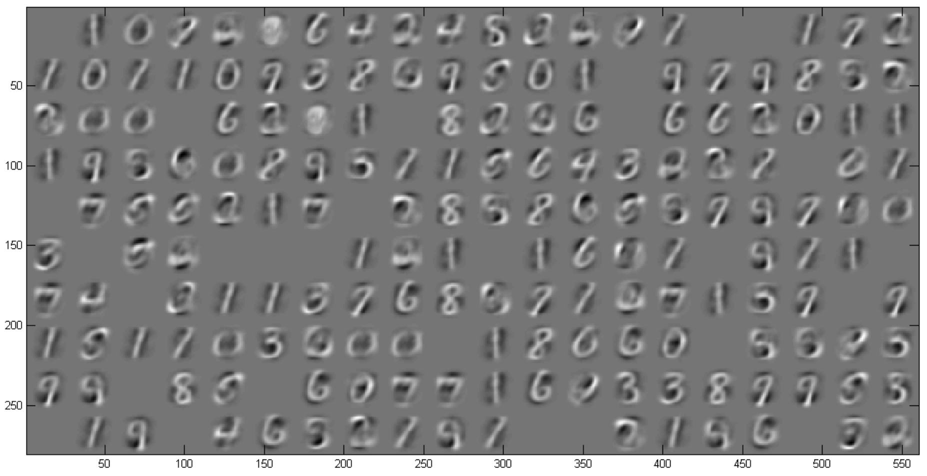


Fig. 14 Second layer of stacked autoencoders detects contours

In conclusion, stacked RBMs and stacked autoencoders tend to learn similar features and achieve similar classification accuracy on the MNIST dataset.

2.6 State of the Art on Deep Learning

As the first breakthrough, a deep belief network (DBN) by stacking pre-trained RBMs without the decoding parts was proposed by Hinton [2, 10, 23]. The pre-training idea was then adopted by many researchers to come up with new algorithms. [20] used sparse RBMs as learning units. [1, 24, 25] used an autoencoder and its variants such as denoising or sparse version as the building blocks instead of RBM families.

Deep learning shows it power mainly in two areas: speech recognition / language processing [26-29] and object recognition[30]. Recent success in these areas features two key ingredients: convolutional architectures[17, 21, 30] and dropouts[31, 32].

The convolutional architectures alternate convolutional layers and pooling layers. Units on a convolutional layer only deal with a small window which corresponds to a spatial or temporal position. Units on a pooling layer aggregate the outputs on units at a lower convolutional layer.

Dropouts intentionally ignore random nodes in hidden layers and input layers when training. This process mimics averaging multiple models. It also improves neural networks by preventing co-adaptation of feature detectors by forcing different nodes to learn different features.

Sparsity, denoising, and dropouts all aim to reduce the capacity of neural networks to prevent overfitting.

Chapter 3

Meta-parameters and Data Pre-processing in Deep Learning

3.1 Motivation

Existing deep learning methods involve a large number of meta-parameters, such as the number of hidden layers, the number of hidden nodes, the sparsity target, the initial values of weights, the type of units, the learning rate, etc. Existing applications usually do not explain why the decisions were made and how changes would affect performance.

Firstly, how to determine the structure of deep learning networks is unknown. For example, the first DBN paper [2] used a 784-500-500-2000-10 DBN to achieve 98.8% classification accuracy on MNIST. It would be interesting to know why such a network configuration was chosen and if other choices could yield similar results. Thus, it is difficult for a novice user to make good decisions for a new application in order to achieve good performance.

In addition, deep learning is promising to extract features on its own, therefore the raw input should work well. But what if we feed in hand-engineered features? Will it work better? Despite the same or slightly worse performance, is it worthwhile to apply dimension reduction beforehand to speed up the training process?

Last, even if the two types of building blocks in deep learning, namely RBMs and autoencoders, are expected to function similarly, it is still necessary to compare their performance in different settings on different datasets.

This chapter will address these three important questions.

3.2 Dimensionality Reduction by PCA and Whitening

When the input space is too large for neural networks to handle, we often want to reduce the dimensionality of the input data to significantly speed up the training process. This could be done by a general dimensionality reduction algorithm, or by an expert designed feature extractor. The latter has the benefit of obtaining meaningful representation but it also loses some information of the raw data.

For problems that do not have human understandable features, principal component analysis (PCA) is a popular choice as a linear dimensionality reduction technique.

PCA is used to find a lower-dimensional subspace onto which the original data is projected. Usually we need data x that has zero mean. If the data does not have the property, we zero mean it. First we compute the covariance matrix of x as

$$\sum = \frac{1}{m}\sum_{i=1}^{m}(x^{(i)})(x^{(i)})^T \tag{19}$$

where $x^{(i)}$ is one data point and m is the number of data points.

Then we apply standard singular value decomposition to the covariance matrix to get the matrix U of eigenvectors and the vector λ of eigenvalues. We could preserve all the input data's information but present it in a different basis by computing

$$x_{rot} = U^T x \tag{20}$$

The reduced dimension representation of the data can be obtained by computing

$$\hat{x} = U_{1\ldots k}^T x_{1\ldots k} \tag{21}$$

where k is the dimension to keep.

To set k, we usually determine it by the percentage of variance retained, which is given by

$$\frac{\sum_{j=1}^{k} \lambda_j}{\sum_{j=1}^{n} \lambda_j} \tag{22}$$

To retain 99% of the variance, we need to pick the smallest value of k such at the above value is no less than 0.99. The special case is k = n, when the retaining rate is 1, which means the PCA processed data containing all the variance.

The goal of whitening is to make features less correlated and have the same variance. We can simply get it done by rescaling features by

$$\hat{x}_{PCAwhite,i} = \frac{\hat{x}_i}{\sqrt{\lambda_i + \varepsilon}} \quad (23)$$

where ε is a regularization term to prevent the result from blowing up.

3.3 Sleep Stage Dataset and its Features

The MNIST dataset and a sleep stage dataset will be used in this chapter. This section is the description of the benchmark sleep stage dataset provided by PhysioNet [33]. This study will use the first 5 acquisitions out of 25 available in the dataset (21 males and 4 females with average age 50), each consisting of 1 EEG channel (C3-A2), 2 EOG channels, and 1 EMG channel downsampled at 64Hz. Each acquisition lasts about 7 hours on average. Each sample is taken from 1-second window with 256 dimensions, normalized to [0, 1]. The labels are 5 sleep stages: awake, stage 1 (S1), stage 2 (S2), slow wave sleep (SWS) and rapid eye-movement sleep (REM).

A band-pass filter and a notch filter at 50Hz are applied to all channels. Then the 30-second data before and after a sleep stage switch are removed. Finally all classes are balanced determined by the class with the fewest samples. 1/7 data is reserved for testing and the rest is for training, which results in 8508 samples for testing and 51042 for training. A 60-second raw data of each channel is depicted in Fig. 15.

28 hand-made features are extracted from 1-second long samples. The features of a subject's data are shown in Fig. 16. The details about how to calculate these features can be found in [34-38].

The corresponding classes are shown in Fig. 17.

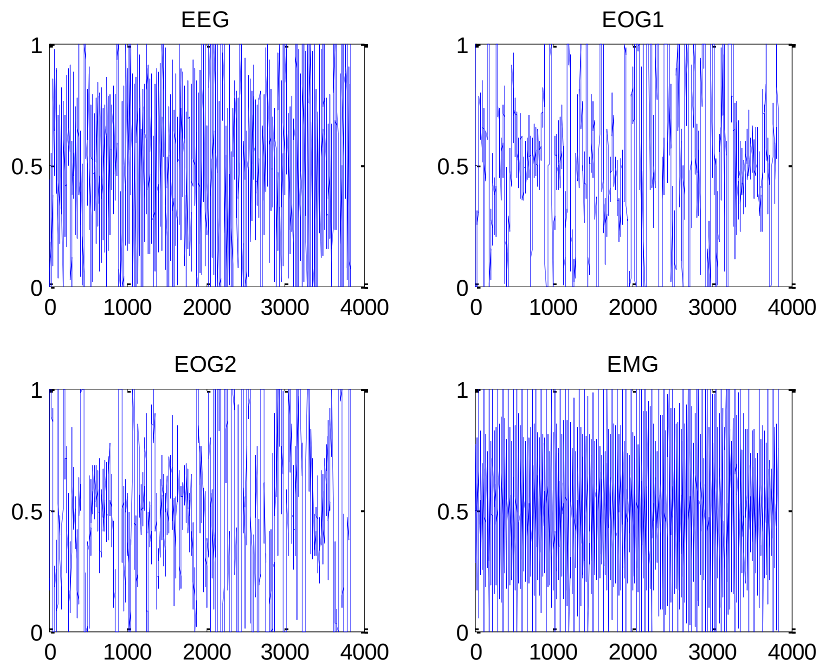


Fig. 15 60-second raw data downsampled at 64Hz of sleep stage dataset.

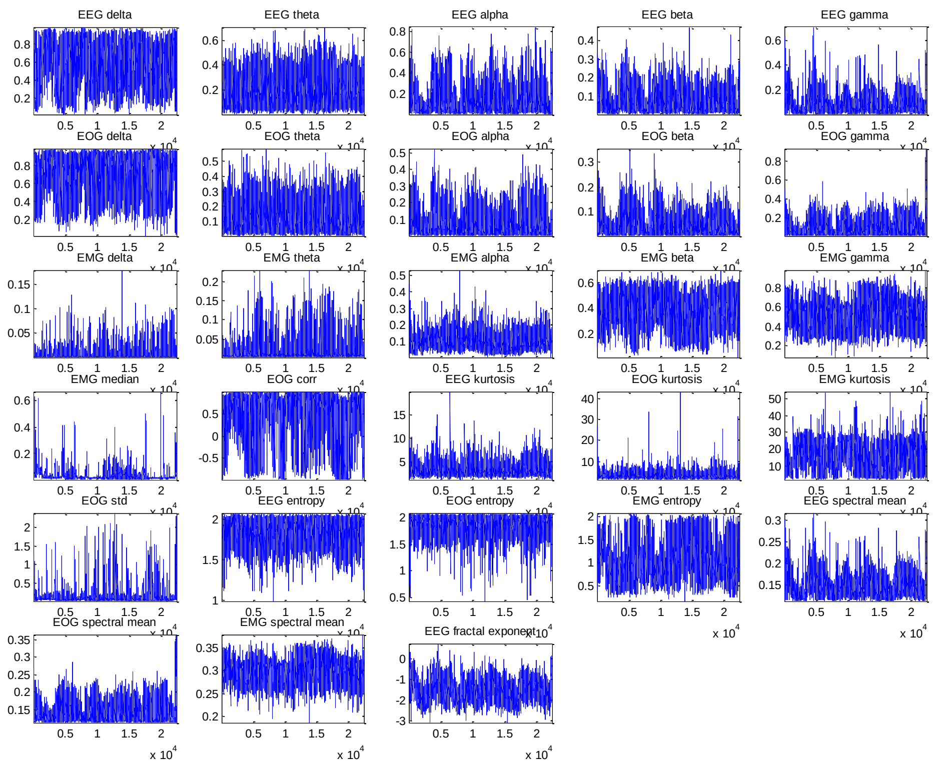


Fig. 16 28 features of a subject's data. X-axis is time in seconds and y-axis shows non-normalized values

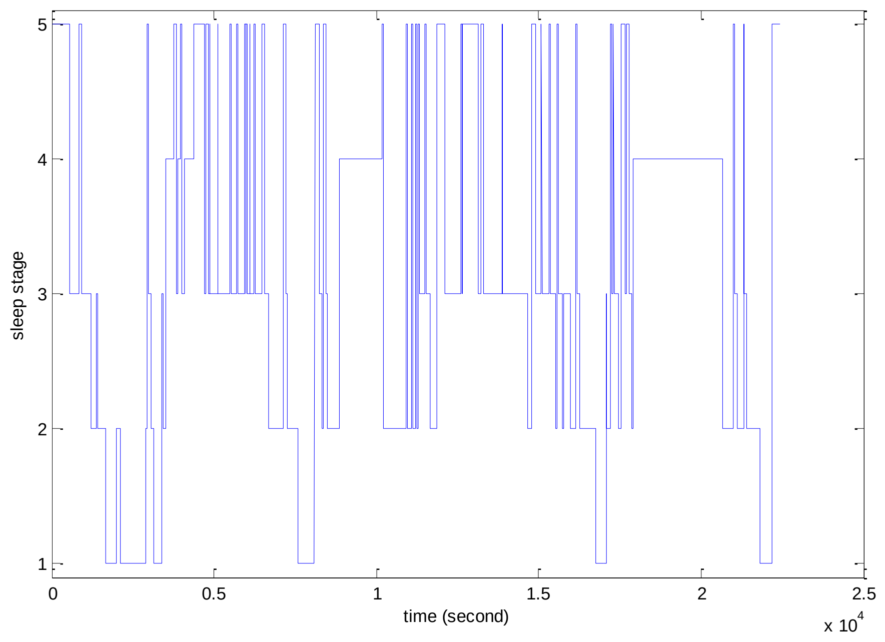


Fig. 17 sleep stage classes of a subject. X-axis is time in seconds and y-axis is the 5 possible sleep stages

3.4 Investigation of Optimal Meta-parameters and Pre-processing Techniques

This study will conduct a search on 2 datasets (MNIST and sleep stage dataset) along 3 dimensions: raw vs. PCA95% whitened data as input, net structures (different numbers of layers and different numbers of nodes), and RBM vs. autoencoder as learning units. There will be 8 possible combinations.

Softmax regression serves as a baseline compared to more advanced deep learning algorithms (i.e., stacked RBMs and stacked autoencoders).

The following two sections will be experiments carried out on MNIST and the sleep stage dataset, respectively. In the first subsection, softmax regression only will be used to evaluate the performance of raw data vs. PCA processed data on MNIST or features on the sleep stage dataset. In the second subsection, different net structures will be tried on raw data and picked PCA processed data (as for retaining rate and with or without whitening) or features.

3.5 Experimental Results on MNIST Dataset

3.5.1 Data Pre-processing

To evaluate how PCA processing with different retention rates and whitening affects the performance compared to the raw data, softmax regression is applied to the MNIST dataset.

3.5.1.1 On Raw Data

Weight decay of the softmax regression is set to be 3e-3. The training on raw data takes 17 seconds and achieves 0.9150 classification accuracy.

The learned weights by the 10 softmax nodes in the output layer are shown in Fig. 18.

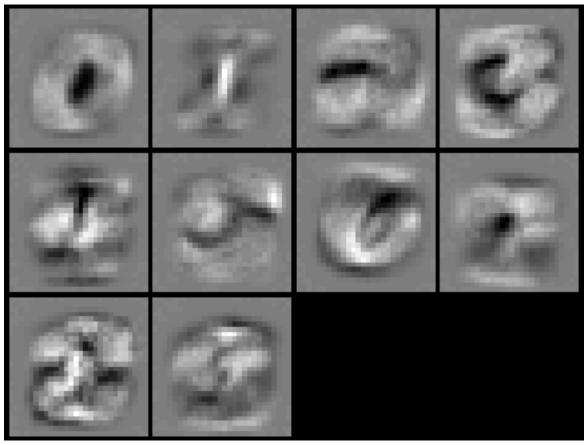
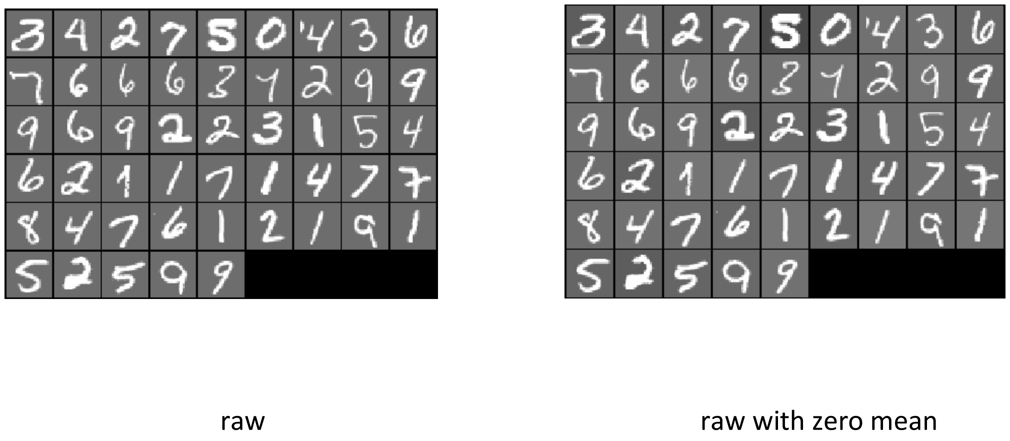
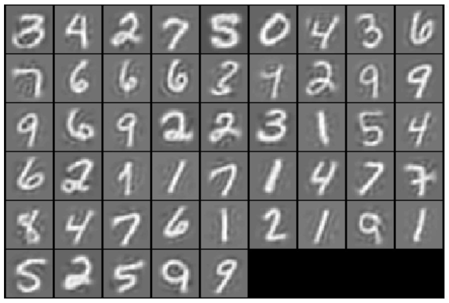


Fig. 18 weights learned by 10 softmax nodes on raw of MNIST

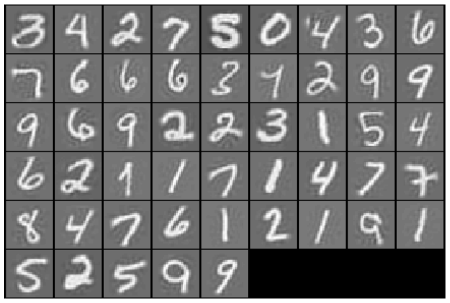
3.5.1.2 On PCA With or Without Whitening

90%, 95%, 99%, and 100% retention rates for PCA with or without whitening (regularization ε = 0.1) are evaluated. The quality of recovered data from the lower dimensional space is shown in Fig. 19. Even if we reduce the data dimension to 64-d, it still retains 90% of the variance, and the digits are highly recognizable.

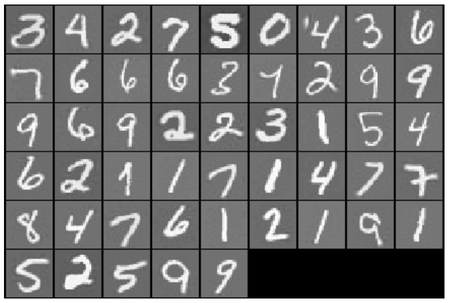




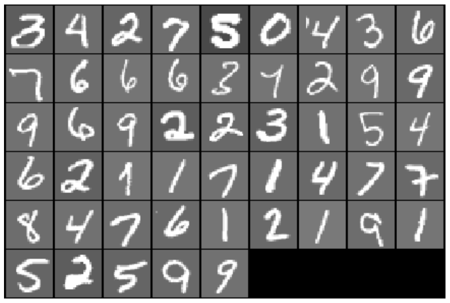
PCA90% 64-d



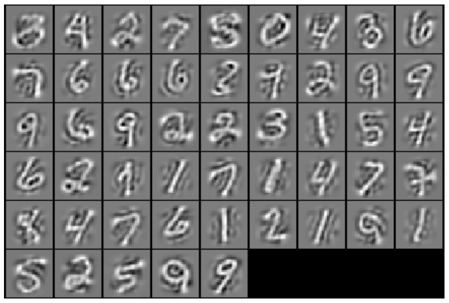
PCA95% 120-d



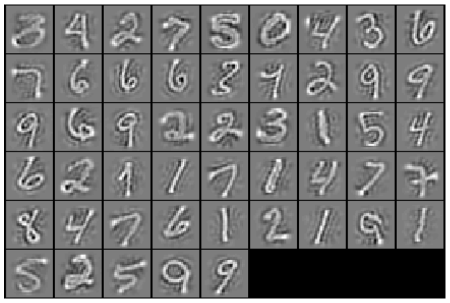
PCA99% 300-d



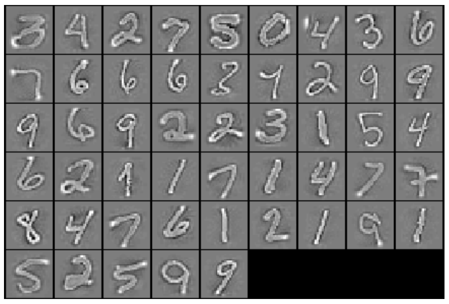
PCA100% 784-d



PCA90% with whitening



PCA95% with whitening



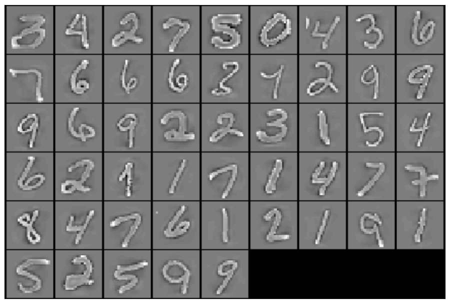


Fig. 19 PCA with or without whitening

The classification accuracy of softmax regression on raw and pre-processed data of MNIST is shown in Fig. 20. The time spent on training is depicted in Fig. 21.

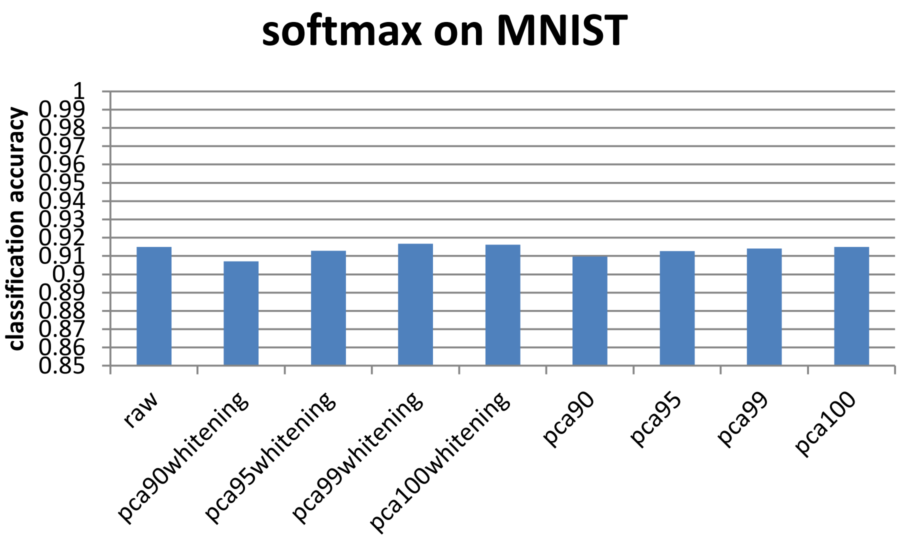


Fig. 20 Classification accuracy of softmax regression on MNIST

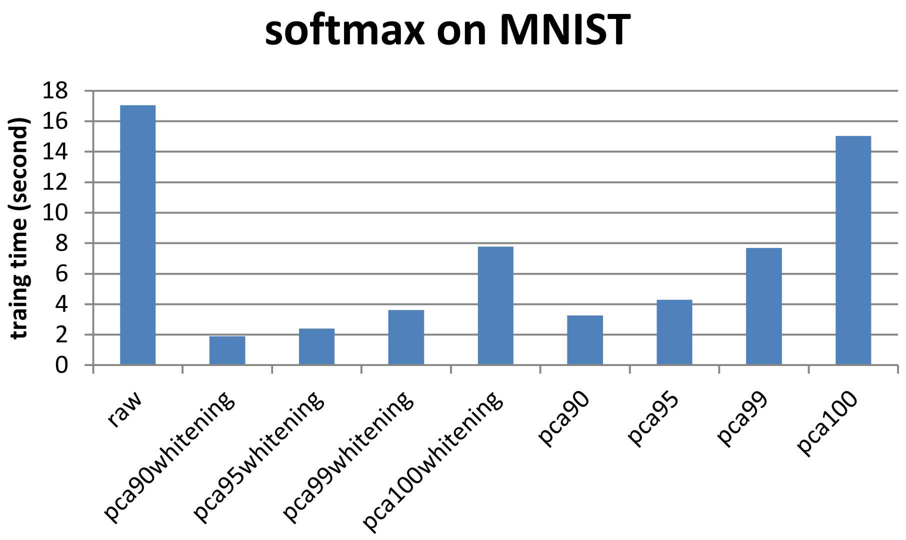


Fig. 21Training time of softmax regression on MNIST

For softmax, using whitening or not does not affect the classification accuracy much, but PCA especially with whitening reduces training time a lot. Whitened data has smaller variations, which could be the reason it converges faster. For example, PCA95 with whitening saves ¾ of training time with less than 0.3% accuracy loss. However, PCA process alone takes about 10 seconds and whitening takes additional 10 seconds. The pre-processing overhead makes it less attractive for simple classifier such as softmax regression.

3.5.2 Deep Learning Network Structure

Classifications on the raw and PCA95% whitened data using different network structures in terms of number of layers and number of nodes and different learning units will be carried out to compare their performance.

3.5.2.1 Stacked RBMs on Raw Data

Fig. 22 shows classification accuracy of stacked RBMs of various network structures on raw of MNIST. More layers and more nodes seem to achieve better performance, but the performance gain is not very significant when the number of nodes grows more than 200. The corresponding training time is shown in Fig. 23. The training time roughly has a linear relationship with the number of layers and number of nodes in each layer.

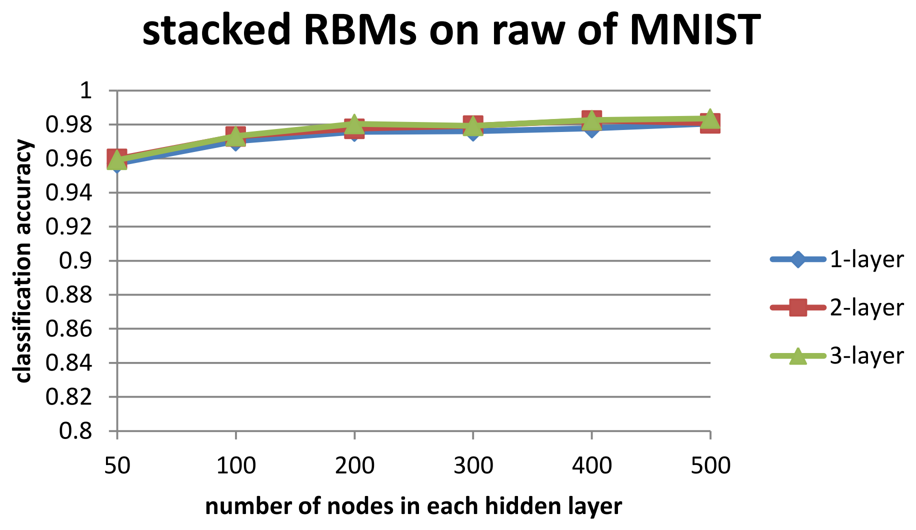


Fig. 22 Classification accuracy of stacked RBMs of various net structures on raw of MNIST

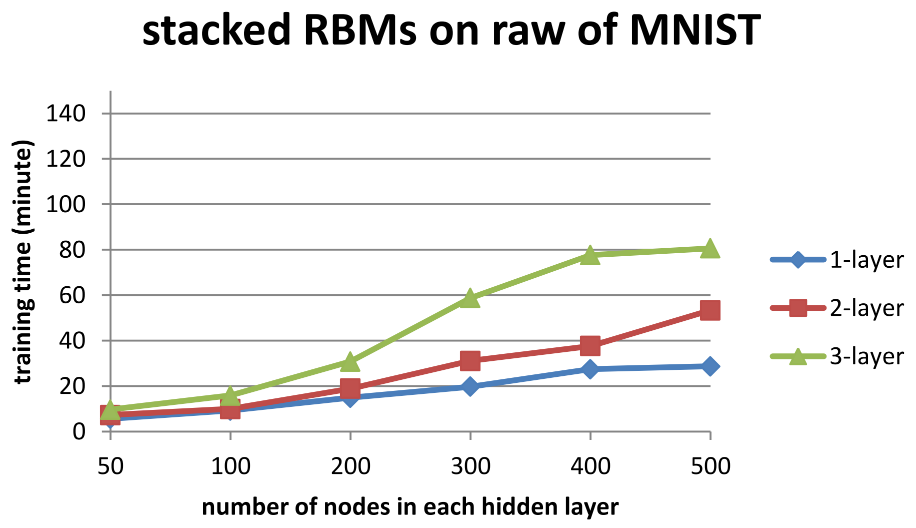


Fig. 23 Training time of stacked RBMs of various net structures on raw of MNIST

3.5.2.2 Stacked RBMs on PCA at Retention Rate 95%

Fig. 24 and Fig. 25 show classification accuracy and training time of stacked RBMs of various network structures on PCA95% whitened data of MNIST.

The classification accuracy is no better than that of softmax. Even worse, the 3-layer case can only achieve less than 0.8 classification accuracy. The reason is PCA itself extracts uncorrelated features and the stacked RBMs try to extract some correlations from uncorrelated features and of course fails. Thanks to the topmost softmax layer, stacked RBMs with 1 and 2 layers still work but 3-layer network does a terrible job in the pre-training stage which cannot be offset by the fine-tuning stage.

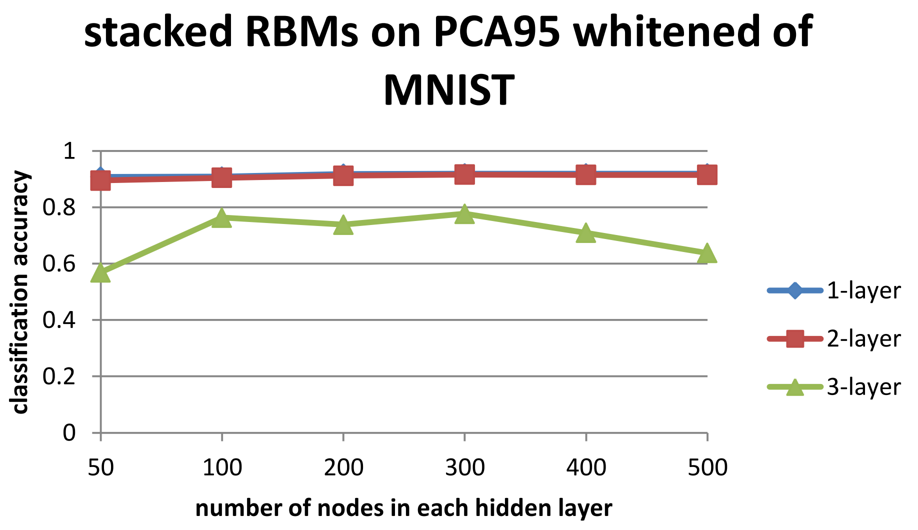


Fig. 24 Classification accuracy of stacked RBMs of various net structures on PCA95 whitened of MNIST

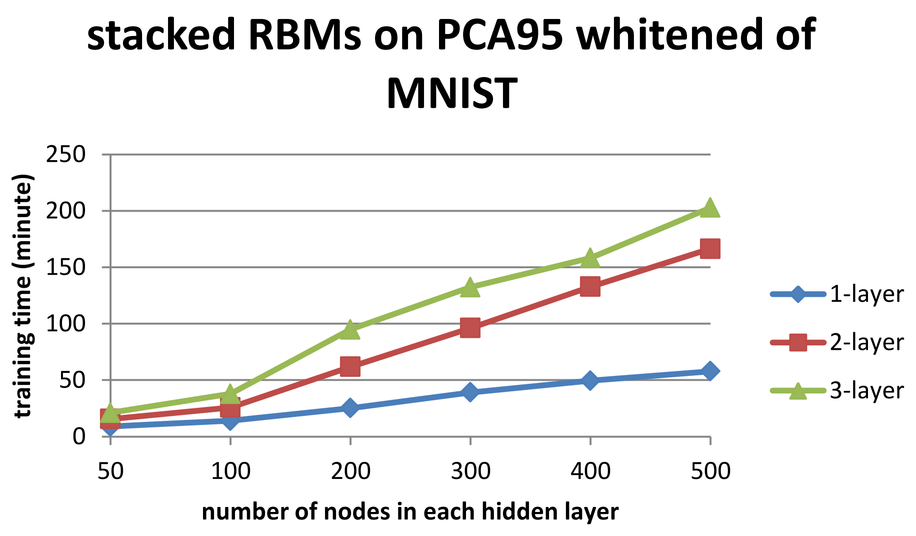


Fig. 25 Training time of stacked RBMs of various net structures on raw of MNIST

3.5.2.3 Stacked Autoencoders on Raw Data

Fig. 26 and Fig. 27 show classification accuracy and training time of stacked autoencoders of various network structures on raw of MNIST. When the number of nodes in each layer is small (e.g., 50), the network with 3 layers captures less and less useful information from bottom to up due to too few nodes with sparsity regularization, so the classification accuracy is as low as less than 0.91. As the number of nodes grows, the performance becomes better, and eventually beats 1-layer network.

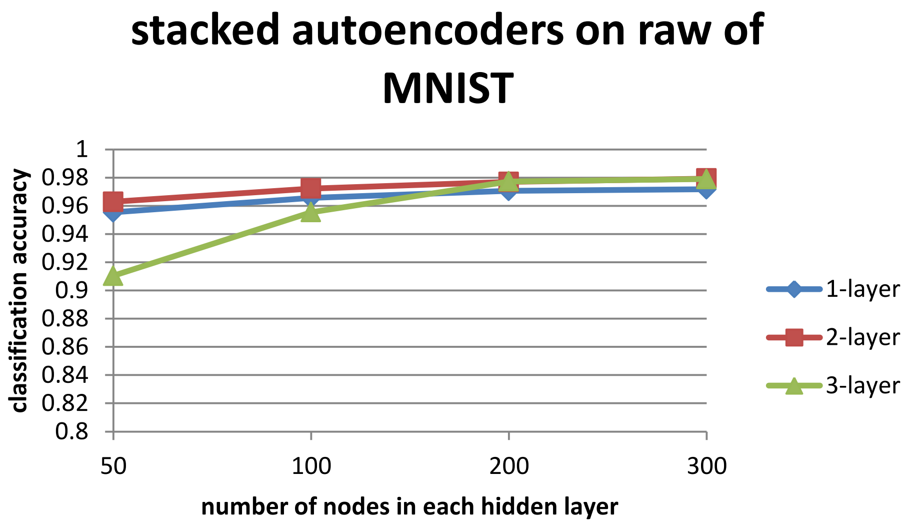


Fig. 26 Classification accuracy of stacked autoencoders of various net structures on raw of MNIST

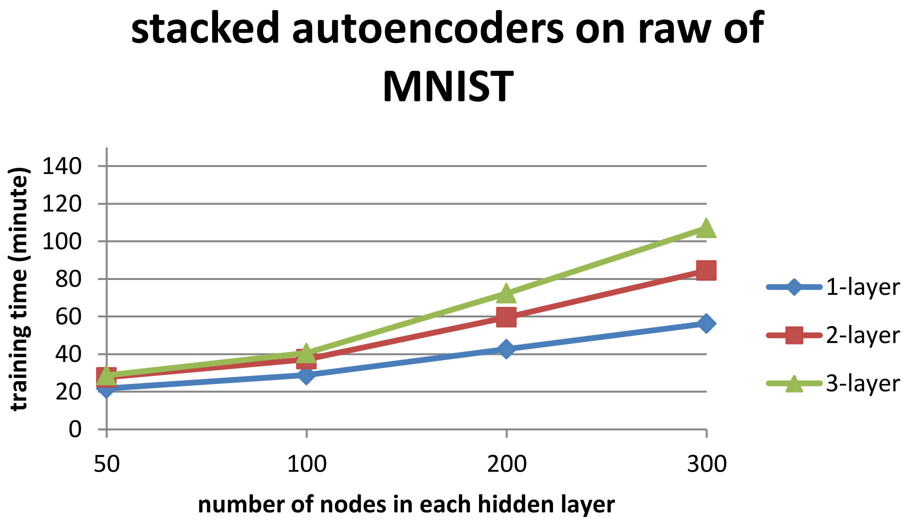


Fig. 27 Training time of stacked autoencoders of various net structures on raw of MNIST

3.5.2.4 Stacked Autoencoders on PCA at Retention Rate 95%

Fig. 28 and Fig. 29 show classification accuracy and training time of stacked autoencoders of various network structures on PCA95% whitened data of MNIST. The 1-layer network achieves comparable results with that on raw data with nearly half of the training time. However, when the number of nodes in each layer is small, the networks with 2 or 3 layers capture less and less useful information from bottom to up due to too few nodes with sparsity regularization, so the classification accuracy is as low as like random guessing. As the number of nodes grows, the performance becomes better.

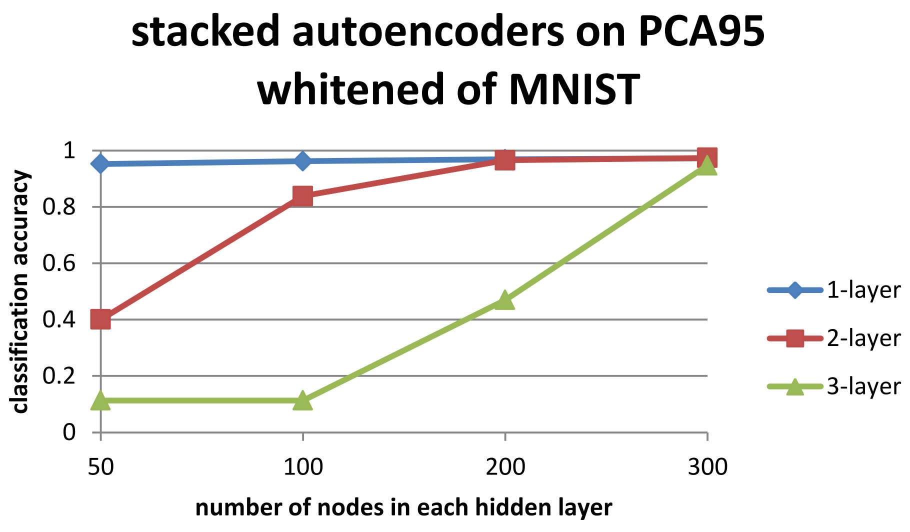


Fig. 28 Classification accuracy of stacked autoencoders of various net structures on PCA95 whitened of MNIST

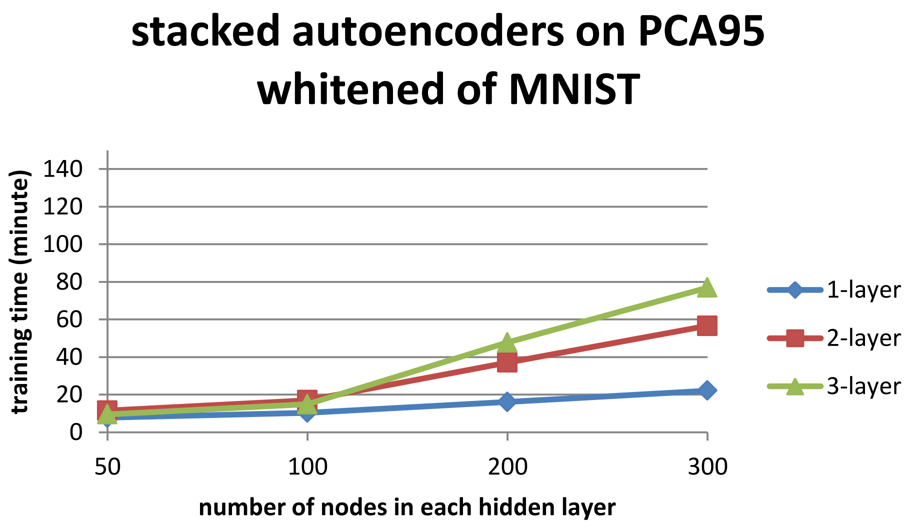


Fig. 29 Training time of stacked autoencoders of various net structures on PCA95 whitened of MNIST

3.6 Experimental Results on Sleep Stage Dataset

3.6.1 Data Pre-processing

To compare the performance of raw data and features of the sleep stage dataset, softmax regression is applied.

3.6.1.1 On Raw Data

Weight decay of the softmax regression is set to be 3e-3. The training on raw data takes 47 seconds and achieves 0.2056 classification accuracy. Softmax regression alone is too weak to capture useful information of the raw data.

3.6.1.2 On Features

Training on 28 features takes only 2.5 seconds and results in 0.5678 classification accuracy. The hand crafted features are very useful for the softmax classifier.

3.6.2 Deep Learning Network Structure

Classifications on the raw data and features using different network structures in terms of number of layers and number of nodes and different learning units will be carried out to compare their performance.

3.6.2.1 Stacked RBMs on Raw Data

Fig. 30 and Fig. 31show classification accuracy and training time of stacked RBMs of various network structures on raw of the sleep stage dataset. Stacked RBMs capture the features well from the raw data and the classification performance is not sensitive to the network structures.

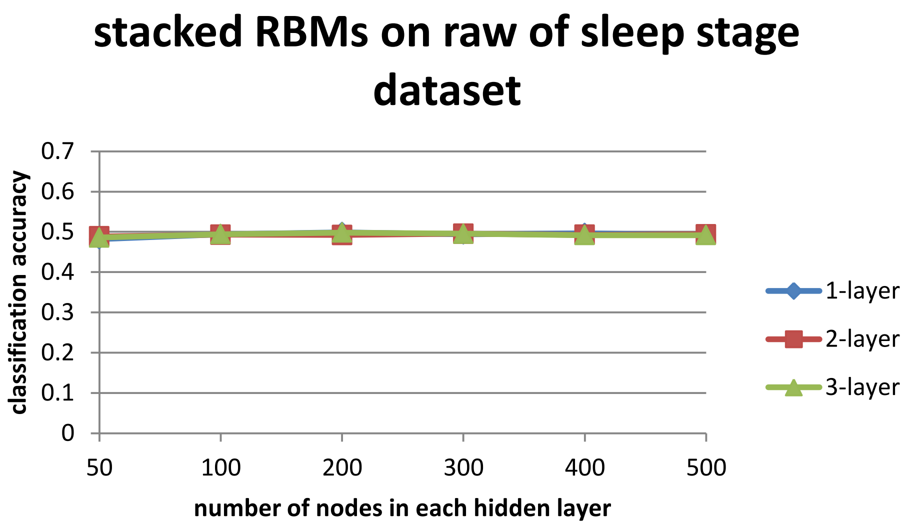


Fig. 30 Classification accuracy of stacked RBMs of various net structures on raw of sleep stage dataset

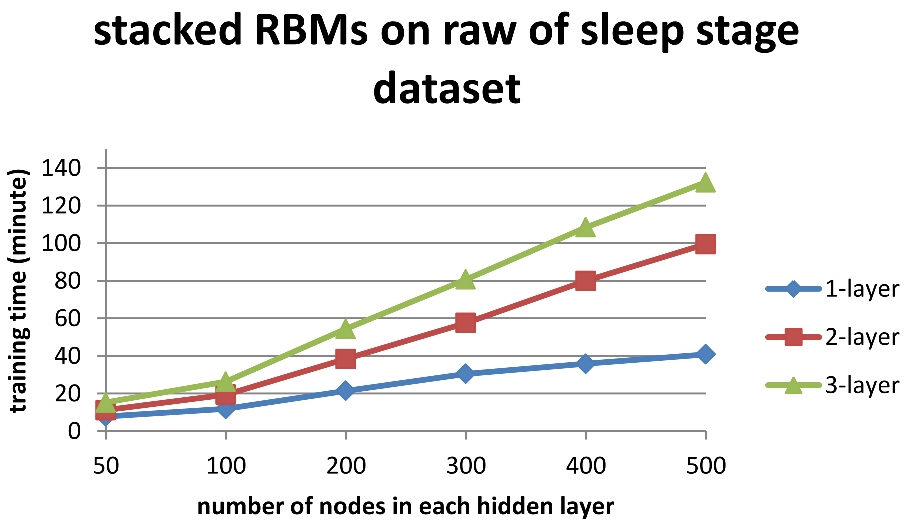


Fig. 31Training time of stacked RBMs of various net structures on raw of sleep stage dataset

3.6.2.2 Stacked RBMs on Features

Fig. 32 and Fig. 33 show classification accuracy and training time of stacked RBMs of various network structures on features of the sleep stage dataset. The classification accuracy trained by features in a network with small number (e.g., 50) of nodes beats trained by raw data, suggesting deep learning fails to capture some of the useful information that a hand-crafted feature extractor can do. The training is fast, too, due to the low dimensionality of the feature space. However, the performance is sensitive to the number of nodes in each layer. Too large number with more network capacities tends to deteriorate the classification performance.

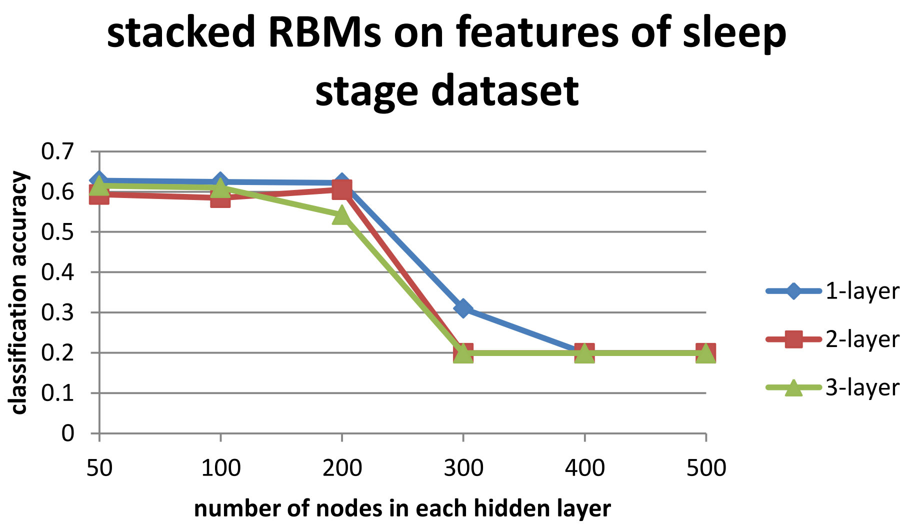


Fig. 32 Classification accuracy of stacked RBMs of various net structures on features of sleep stage dataset

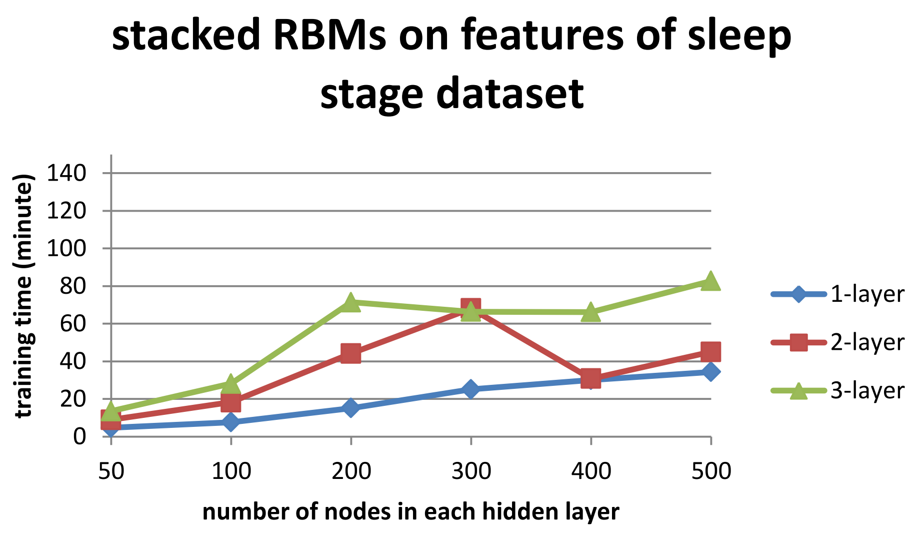


Fig. 33 Training time of stacked RBMs of various net structures on features of sleep stage dataset

3.6.2.3 Stacked Autoencoders on Raw Data

This experiment does not work due to an error in the training in which process matrix is close to singular or badly scaled.

3.6.2.4 Stacked Autoencoders on Features

Fig. 34 and Fig. 35 show classification accuracy and training time of stacked autoencoders of various network structures on features of the sleep stage dataset. The classification accuracy trained by features in a network of 1-layer beats trained by raw data. The training is fast, too, due to the low dimensionality of the feature space. However, the performance is sensitive to the number of layers. 2-layer and 3-layer networks do not work better than random guess.

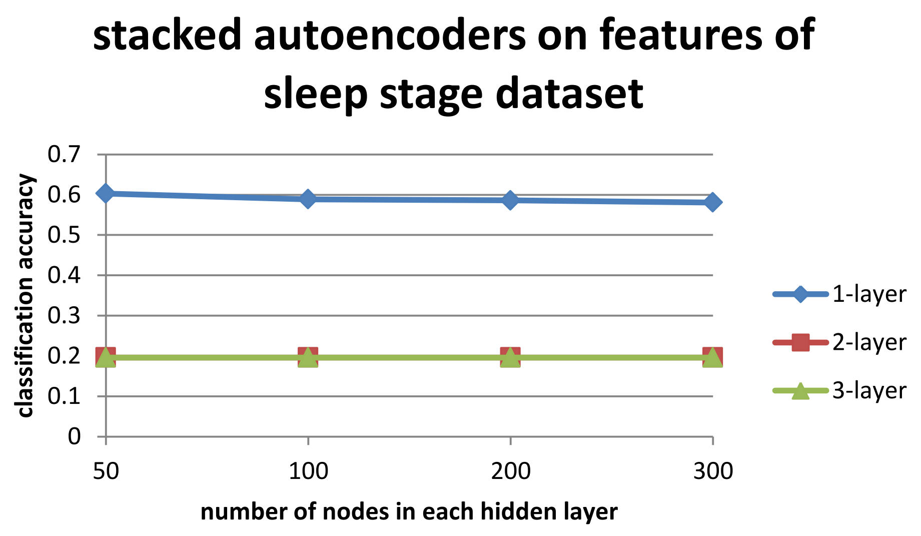


Fig. 34 Classification accuracy of stacked autoencoders of various net structures on features of sleep stage dataset

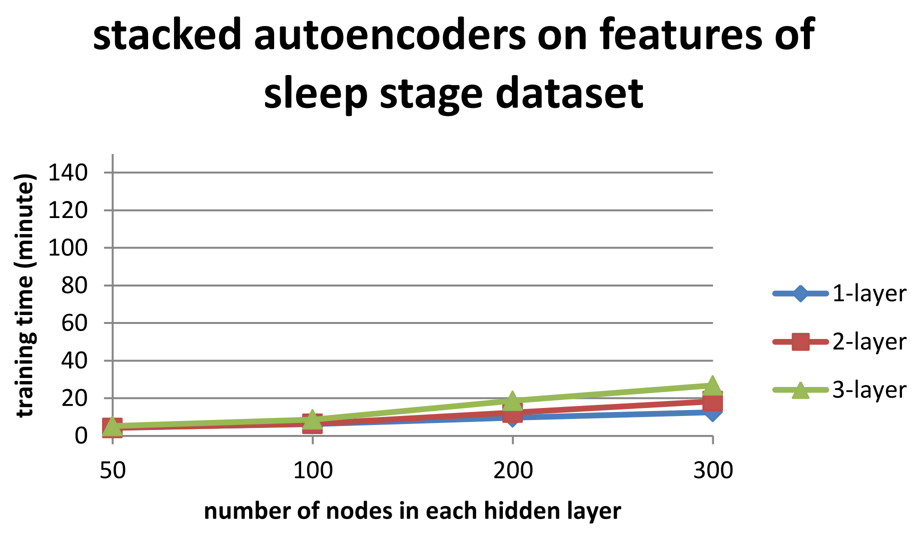


Fig. 35 Training time of stacked autoencoders of various net structures on features of sleep stage dataset

3.7 Summary

Followed is the summary of classification accuracy of various network structures of stacked RBMs and stacked autoencoders on both MNIST and the sleep stage datasets with or without pre-processing. The listed optimal net structures will be used in the later studies. In conclusion, the optimal net structures are highly application-dependent.

Table 3 Summary of network structures including optimal settings as well as pre-processing techniques on MINIST and sleep stage dataset

Dataset	Pre-processing	stacked RBMs	stacked autoencoders
MNIST	raw	Good in any net structures [500 500 500]	Good, except when hidden nodes are too few [200 200]
	PCA95% whitened	Sensitive to number of layers [200 200]	Sensitive to number of nodes [300]
sleep stage dataset	raw	Good in any net structures [200]	NA
	28 features	sensitive to number of nodes [50]	very sensitive to number of layers [50]

Chapter 4

Active Labeling in Deep Learning

4.1 Motivation

In a semi-supervised learning paradigm such as deep learning, unlabeled data are easy to get at low or no extra cost, but labeled data are expensive. Due to limited resources, only very few labeled data can be obtained given a certain budget. For example, in a classification problem on physiological data from biosensors, the unlabeled data can be obtained by simply asking subjects to wear sensors day and night, but labeled data may not be available until human experts manually make annotations on selected unlabeled samples. Therefore, to make the best use of the budget for a discriminative learning task, it would be useful to propose an algorithm to carefully choose unlabeled samples to be labeled.

Active learning (AL) [4] asks queries in the form of unlabeled instances to be labeled by an oracle (a human annotator). The goal is to achieve high classification accuracy using as few labeled samples as possible. A pool-based active learning queries the samples in the unlabeled pool that are most ambiguous for the current model. The newly labeled samples are added to the labeled pool and used to retrain the model. The two processes form a cycle as shown in Fig. 36. The problem to be solved in this chapter falls into the field of active learning.

To the best of my knowledge, there are no active learning algorithms applied in DBNs except for Active Deep Networks (ADNs) proposed by [39]. ADNs follow previous work on active learning for SVMs by defining the uncertainty of an unlabeled sample as its distance from the separating hyperplane and it can only work on binary classification problems.

Since the top layer of both stacked RBMs and stacked autoencoders outputs the probabilities of each label to be chosen, it is promising to exploit the probabilities as indicators of uncertainty. My work aims to propose a few methods to effectively use the built-in classification uncertainty in deep learning networks to select unlabeled samples to be labeled.

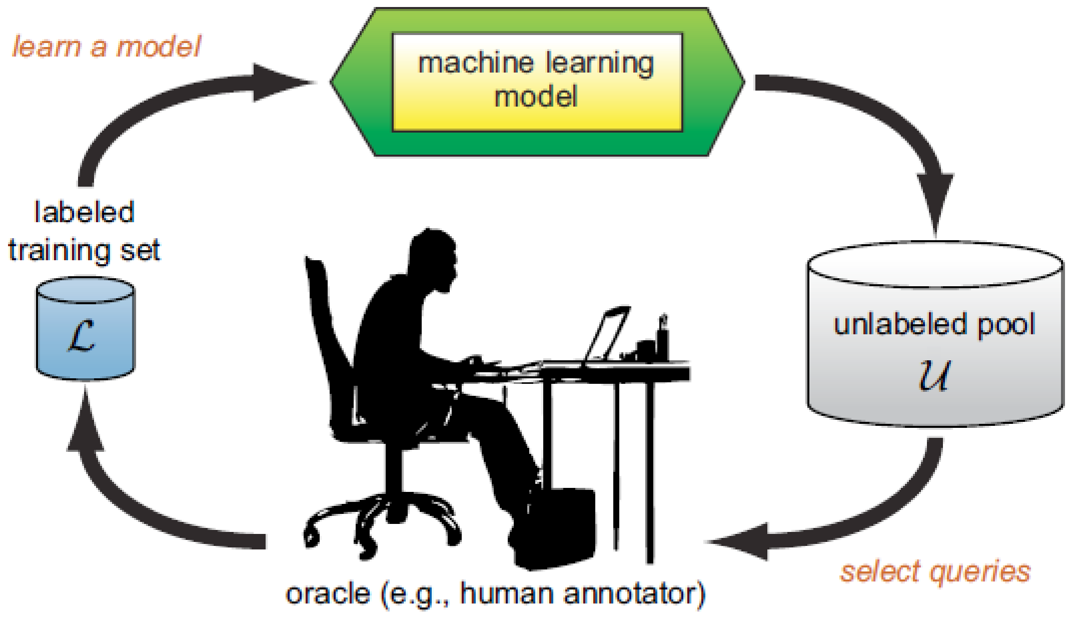


Fig. 36 The pool-based active learning cycle (taken from [4])

4.2 Related Works on Active Learning

Suppose there is a large pool of unlabeled data available at cheap cost and we want to sample from them to get labeled data. A pool-based sampling [40] was proposed to draw queries from the pool in a greedy fashion.

Active learning is adopted for a range of base learners, such as support vector machines and Bayesian networks[41], logistic regression[40], and Markov Models[42].

The pool-based active learning algorithm has been studied for many real-world problems, surveyed by [4], such as text classification[40, 43-45], information extraction[46, 47], image classification and retrieval[48, 49], video classification and retrieval[50, 51], and speech recognition[52].

4.3 Algorithms

The problem to be solved is formulized as follows. Given an unlabeled sample set X^U and a labeled sample set X^L, the algorithm needs to take B samples from X^U, have them labeled, and add them to X^L, in order to minimize the classification error of a deep learning model fine-tuned by X^L, where B is a constant as the budget in each iteration.

The basic idea of my algorithm is simple and not much different from other active learning algorithms – greedily select those samples that are most difficult to classify. The framework is depicted in Fig. 37. Then the problem remains how to find heuristic methods to define difficulty, uncertainty, or ambiguity. Three criteria are proposed under the active labeling deep learning (AL-DL) framework.

AL-DL framework

Input: an unlabeled sample set X^U

1. Use all the unlabeled data X^U to train a DL network (stacked RBMs or stacked autoencoders) layer by layer. The weights and biases will be used as the initial values for fine-tuning a DL network in the following steps.
2. Randomly take B_0 samples from X^U, have them labeled, and add them to the empty set X^L as a "seed" set. For simplicity, assume the classes are balanced in the set X^L.
3. Use X^L to fine-tune a N-layer DL network classifier $Model^N$.
4. Use the classifier $Model^N$ to classify all the unlabeled samples in X^U. Take B the most uncertain samples from X^U, have them labeled, and add them to X^L. (see the three detailed criteria to measure uncertainty in text). Go to step 3.

Output: a N-layer DL network classifier $Model^N$

Fig. 37 Active labeling – deep learning framework

The classification in step 4 applies (6) iteratively starting from an unlabeled sample fed into the lowest input layer until the top layer N and then uses (16) to softmax the activations in the top layer. If j is predicted as the class of the sample x_i is determined by if the unit the class associated with has the largest probability among all the units $p\left(h_j^N|x_i\right)$ in the top layer as below

$$y_j = \begin{cases} 1, if\ j = \underset{j}{\operatorname{argmax}}(p\left(h_j^N|x_i\right)) \\ 0, \qquad\qquad otherwise \end{cases} \tag{24}$$

Since the true class of the sample x_i is unknown yet, the predicted label y_j is not much interesting because we do not know it is correct or misclassified. However, $p\left(h_j^N|x_i\right)$ suggests the confidence of the prediction. Least confidence (LC), margin sampling (MS) [53], and entropy [54] can act as the criteria to pick the most uncertain samples x_i.

Active labeling deep learning with least confidence (AL-DL-LC) picks the samples with the minimum of the maximum of activations as follows

$$x_i^{LC} = \underset{x_i}{\operatorname{argmin}} \underset{j}{\max}(p\left(h_j^N|x_i\right)) \tag{25}$$

where x_i is an input vector, h_j^N is the activation of the unit j in the top layer. If more than one sample needs to be selected, the process could be performed in a greedy fashion.

Least confidence works in the assumption that if the probability of the most probable label for a sample is low then the classification of the sample is uncertain. Thus it discards the information about the remaining label distributions.

Active labeling deep learning with margin sampling (AL-DL-MS) partially corrects the shortcoming by incorporating the posterior of the second most likely label as follows

$$x_i^{MS} = \underset{x_i}{\operatorname{argmin}}(p(y_1|x_i) - p(y_2|x_i)) \tag{26}$$

where y_1 and y_2 are the first and second most probable class labels under the model. Intuitively if the probabilities of predicting a sample to its most likely class and to its second most likely class are too close, the classifier is quite confused about the sample. Therefore some information is needed from the oracle to help the classifier discriminate these two classes.

Active labeling deep learning with entropy (AL-DL-Entropy) aims to take all labels probabilities into consideration by

$$x_i^{Entropy} = \underset{x_i}{\operatorname{argmax}} - \sum_j p\left(h_j^N|x_i\right) logp\left(h_j^N|x_i\right) \tag{27}$$

Entropy represents the amount of information or least number of bits to encode a distribution. It could act as a good measure of uncertainty.

4.4 Experimental Results on MNIST

To evaluate the performance of AL-DL in three criteria, a few experiments on the MNIST dataset have been performed. For comparison, a random selection strategy is employed as the baseline.

4.4.1 Stacked RBMs on Raw Data

The hidden structure of DBN for pre-training is 500-500-500, with 50,000 for training and 10,000 samples for testing. A base set with 1,000 labeled samples (80% for training and 20% for validation) acts as the seed for both the random and active labeling schemes. These samples are balanced among all the classes.

In each iteration the random scheme randomly picks 200 samples (80% for training and 20% for validation) and adds them to the training set and validation set. These two sets are used for the original classifier to train the model and test on the test set to get the classification accuracy.

The AL scheme uses the model trained in the previous iteration to classify unlabeled samples. During the process the 200 (80% for training and 20% for validation) most uncertain samples are labeled by the oracle and added into the training set and validation set used for the new classifier. Then the AL scheme moves to the next iteration. Note the samples picked in this step are not necessarily balanced.

Other DBN parameters are listed below.

Table 4 DBN parameters for both random and AL schemes

Unsupervised pre-training stage	
learning rate	0.05
number of epochs	100
minibatch size	100
momentum	0.5 for the first 5 epochs, 0.9 thereafter
weight cost	0.0002
Supervised fine-tuning stage	
learning rate	0.05
number of epochs	50
minibatch size	100
number of initial epochs	5

10 trails are done for the random scheme and AL schemes with least confidence, margin sampling, and entropy.

The mean and standard deviation of the accuracy of these four methods are depicted below.

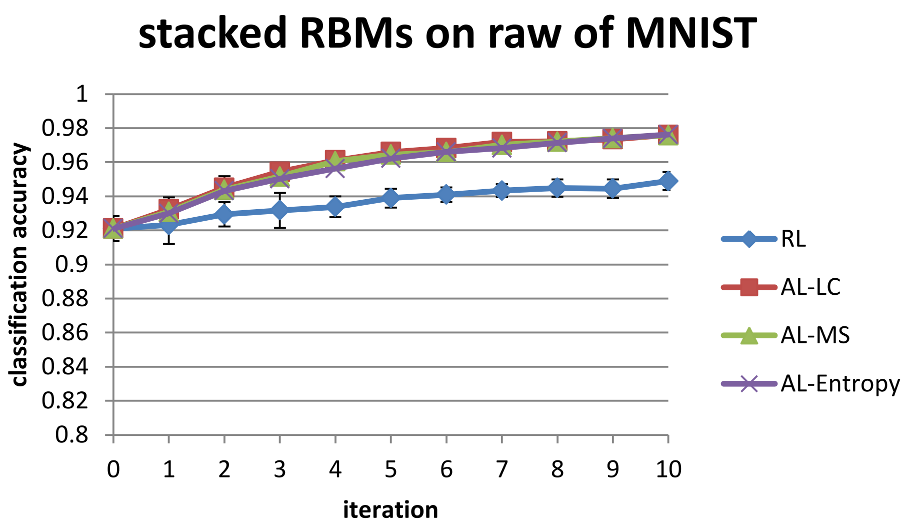
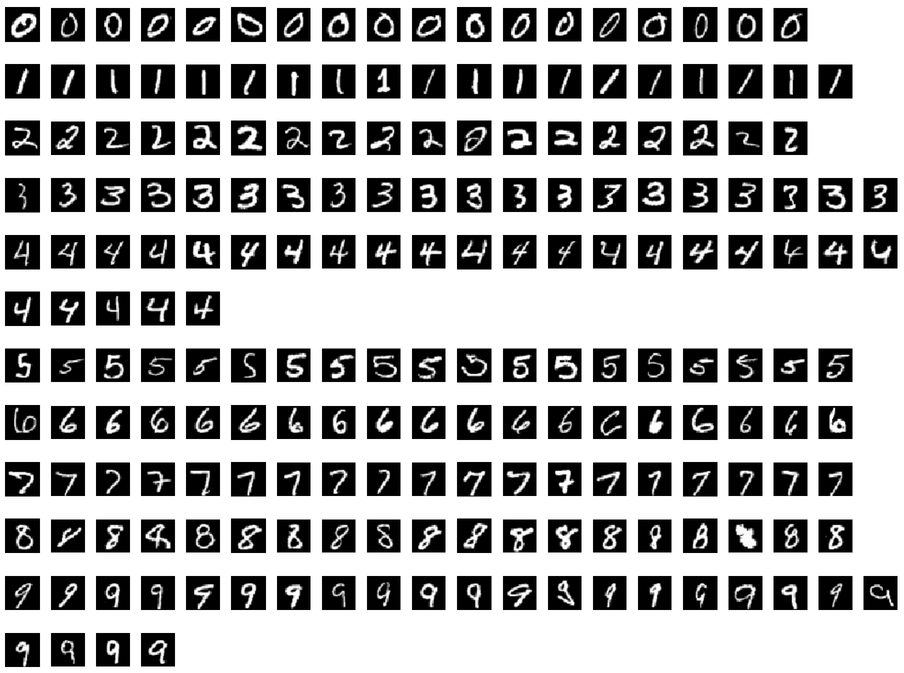


Fig. 38 The mean of classification accuracy of random labeling DBN and active labeling DBN. RL and AL-LC show their error bars. Iteration 0 means only the seed set is used. The performances of the three AL DBNs are close, all better than that of RL DBN.

The means of the accuracy of the three methods AL-LC DBN, AL-MS DBN, and AL –Entropy DBN are close, which all outperform the mean accuracy of RL DBN. The standard deviations of the three active labeling schemes are smaller than that of RL DBN, suggesting more consistent performance of AL DBN.

To confirm the active labeling does pick the most challenging samples, all the 200 samples picked before the 1st iteration by RL DBN (top) and AL –LC DBN (bottom) are shown below.



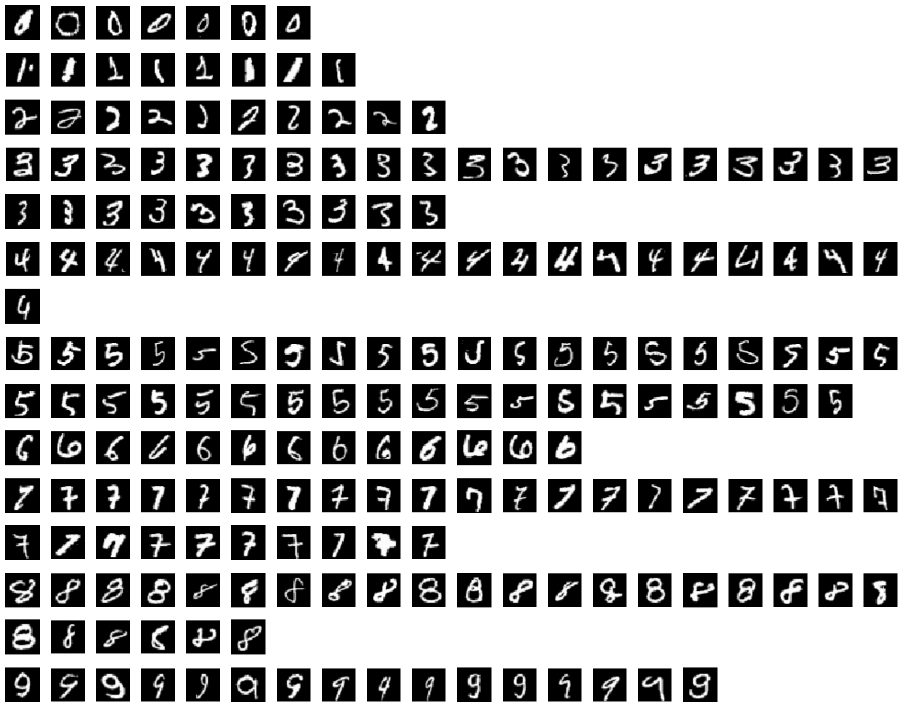


Fig. 39 All the 200 samples picked before the 1st iteration by RL DBN (top) and AL-LC DBN (bottom). The digits in the bottom graph look harder to recognize, indicating AL-LC DBN does pick more challenging samples than RL DBN

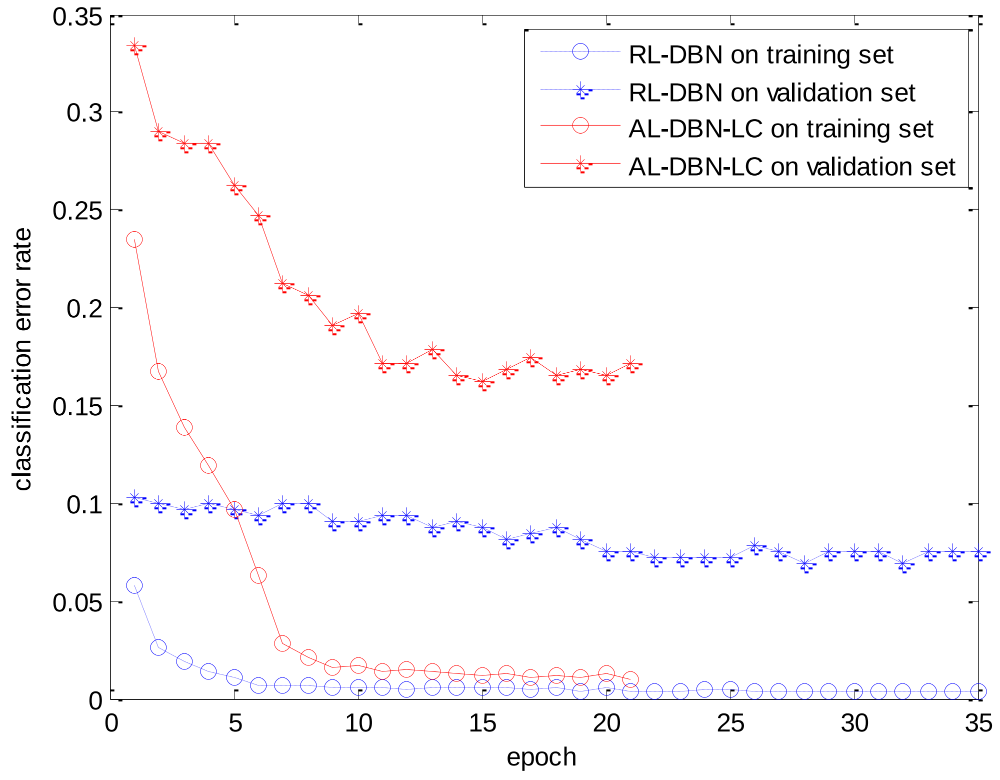


Fig. 40 The comparison of classification error rates on training and validation sets with RL-DBN and AL-DBN-LC in one iteration. AL-DBN-LC works much worse on training and validation sets because it always picks the most challenging samples.

Fig. 40 shows another interesting observation. In each iteration, the RMSE and accuracy using the active labeling strategy on both training set and validation set are much worse than using the random labeling method, but active labeling method's performance on the test set is better. This is because active labeling always puts the most challenging samples to the training set and validation set, it's understandable that the classification accuracy on them is not as high as on randomly picked samples. However, since active labeling always learns from those challenging cases, it should actually outperform the random labeling method that learns from less informative samples.

4.4.2 Stacked Autoencoders on Raw Data

A similar experiment is carried out using stacked autoencoders of 2 hidden layers, each having 200 nodes. Fig. 41 shows active labeling stacked autoencoders beats its random labeling counterpart by about 2% in iteration 5, in which AL-MS works the best.

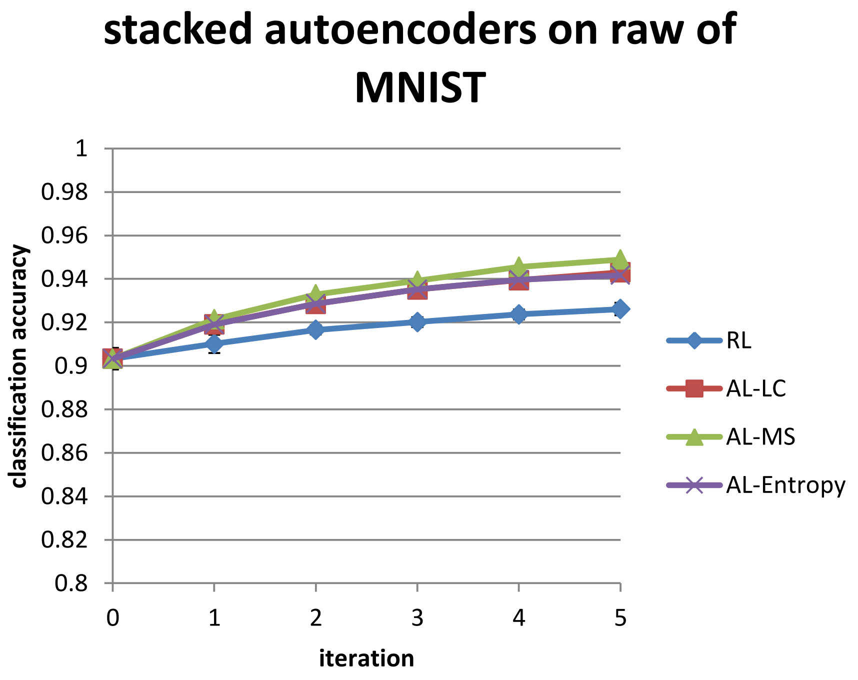


Fig. 41 The mean of classification accuracy of random labeling stacked autoencoders and active labeling stacked autoencoders. RL and AL-LC show their error bars. Iteration 0 means only the seed set is used.

4.4.3 Stacked Autoencoders on PCA at Retention Rate 95%

The experimental result using stacked autoencoders of 1 hidden layer, each having 300 nodes, on PCA 95% whitened data of MNIST, is shown in Fig. 42. Active labeling stacked autoencoders outperforms the random labeling version by up to 4% in iteration 5, in which AL-MS works the best.

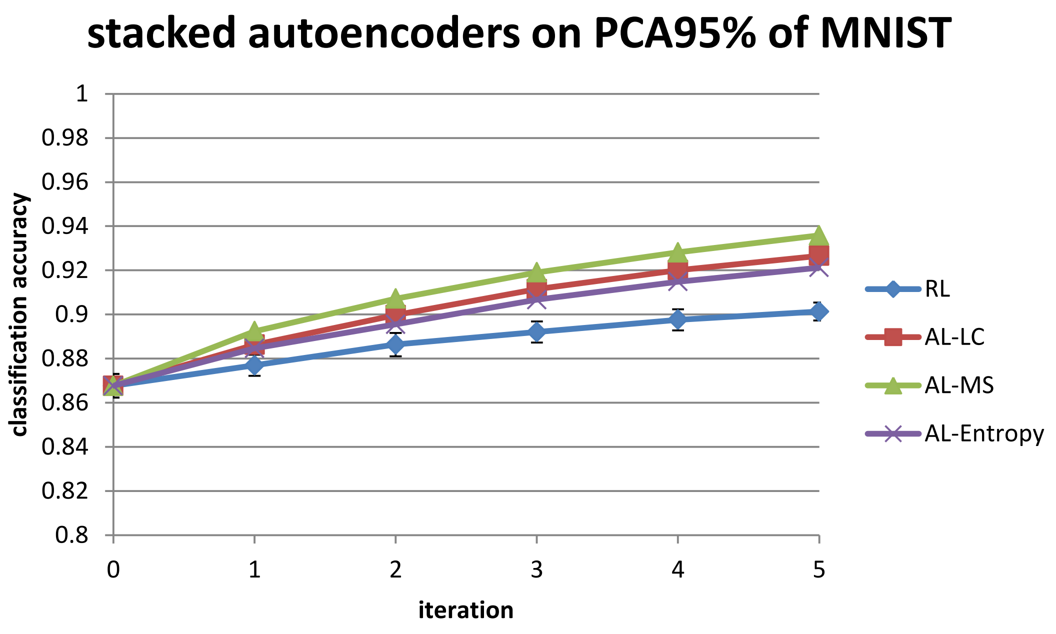


Fig. 42 The mean of classification accuracy of random labeling stacked autoencoders and active labeling stacked autoencoders. RL and AL-LC show their error bars. Iteration 0 means only the seed set is used.

4.5 Experimental results on sleep stage dataset

The same experiment is carried out on the sleep stage dataset using the optimal settings to evaluate the performance of active labeling deep learning in a complex and noisy dataset.

A base set of 1000 labeled samples acts as the seed for both the random and active labeling schemes. These samples are balanced among all 5 classes. There will be 10 (for stacked RBMs) or 5 (for stacked autoencoders) iterations, in each of which 200 samples are picked, labeled, and added to the training set, by random or active labeling. 10 trials are done for the random scheme and AL schemes with least confidence, margin sampling, and entropy.

4.5.1 Stacked RBMs on Raw Data

A 1-layer with 200 hidden nodes stacked RBMs is trained on raw data of the sleep stage dataset with random labeling and 3 different active labeling schemes. Their classification accuracies do not show significant difference in Fig. 43.

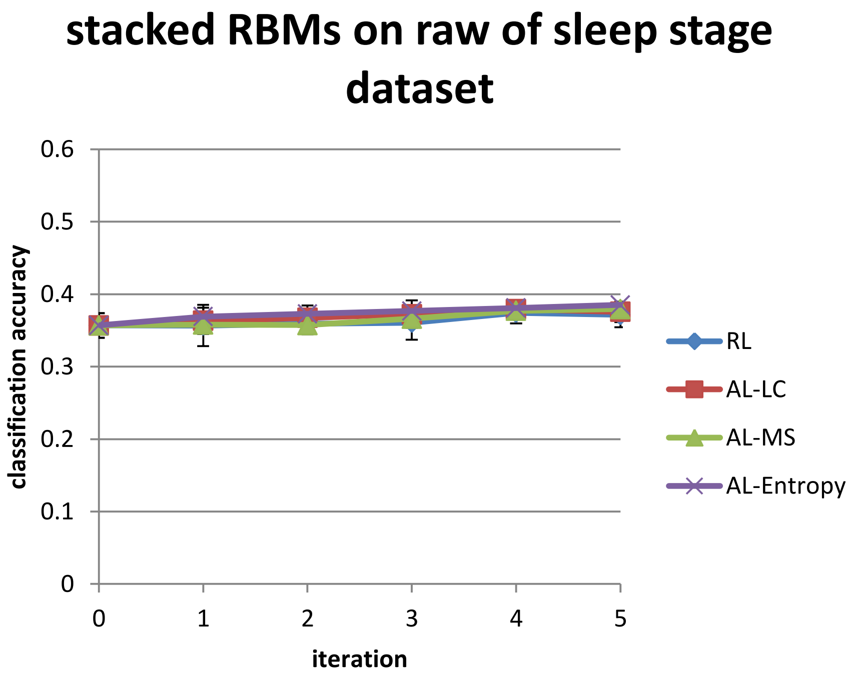


Fig. 43 The mean of classification accuracy of random labeling stacked RBMs and active labeling stacked RBMs. RL and AL-LC show their error bars. Iteration 0 means only the seed set is used.

4.5.2 Stacked RBMs on Features

A 1-layer with 50 hidden nodes stacked RBMs is trained on features of the sleep stage dataset with random labeling and 3 different active labeling schemes. No significant difference is found in them according to Fig. 43.

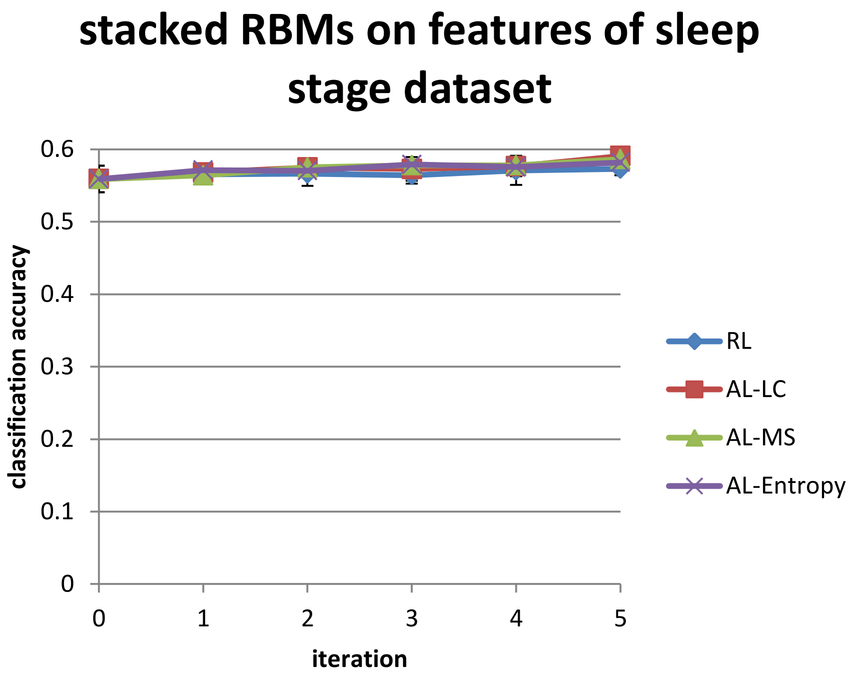


Fig. 44 The mean of classification accuracy of random labeling stacked RBMs and active labeling stacked RBMs. RL and AL-LC show their error bars. Iteration 0 means only the seed set is used.

4.5.3 Stacked Autoencoders on Features

The similar experiment is carried out on a 1-layer with 50 nodes stacked autoencoders on features of the sleep stage dataset. Fig. 45 suggests all 4 algorithms perform almost the same.

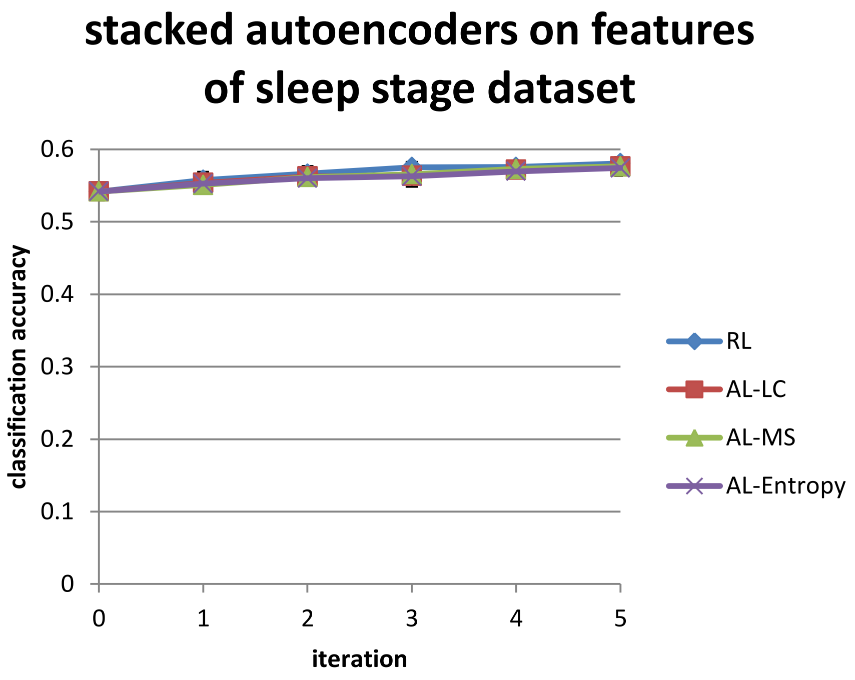


Fig. 45 The mean of classification accuracy of random labeling stacked autoencoders and active labeling stacked autoencoders. RL and AL-LC show their error bars. Iteration 0 means only the seed set is used.

4.6 Summary

On MNIST, active labeling deep learning works better than its random counterpart, no matter what uncertainty measurements (least confidence, marginal sampling, or entropy), what learning units (RBMs or autoencoders), or what data pre-processing techniques (raw or PCA processed data) are used.

However, active labeling strategy does not outperform random labeling deep learning on the sleep stage dataset. The noise nature of the data seems to be blamed on. Since active labeling proactively picks the most uncertain samples to be labels, these samples are more likely to be mislabeled, introducing false information to the model.

Chapter 5

Modeling Raw Physiological Data and Predicting Emotions with Deep Belief Networks

5.1 Introduction

Inspired by the relationship between emotional states and physiological signals [55, 56], researchers have developed a large amount of methods to predict emotions based on physiological data [57-65].

The emotions could be empirically modeled as classes [61]. The negative emotions include anger, anxiety, disgust, embarrassment, fear, and sadness; whereas the positive emotions have affection, amusement, contentment, happiness, joy, pleasure, pride, and relief. Arousal-valence space [66] is an alternative way to define emotions with continuous values. The dimension of arousal represents calmness or excitement, whereas the dimension of valence ranges from highly positive to highly negative.

The physiological data come from biosensors in the following channels: Electrodermal Activity (EDA) that measures electrical conductivity or skin conductance, Electrocardiogram (ECG) measuring heart beats, Electroencephalography (EEG) measuring brain activities, Electrooculogram (EOG) measuring eye movements [67], and, in a broader sense, accelerometer data, voice, GPS trajectories, etc.

Traditionally, no matter to map what biological signals to what emotions, the first step is to retrieve features from the raw data. For example, the R-R intervals extracted from ECG represent a person’s heart beat periods, whose changes may be resulted from emotional changes between calmness and excitement. These features are usually hand-engineered using task dependent techniques developed by domain experts [68-70] and then selected either by experts or feature selection algorithms like principal components analysis (PCA). The process works fine where what features are more relevant to a specific task look obvious but it is still labor-intensive and time-consuming. When the incoming physiological data types and/or prediction tasks change, we have to redo the whole process to tailor new features. It would be useful to have a universal system that can automatically extract features from the raw physiological data without the help of expert knowledge.

Moreover, multi-task learning [71, 72], learning more than one problem at the same time using a shared representation, is desired when learning physiological data. These problems even include unknown ones. Multi-task learning aims to improve the performance of learning algorithms by learning the commonality among multiple tasks. When a subject wears a biosensor with several channels, the task could be to classify her activities, or to determine her emotions. Even if the tasks may not yet be predefined we still want the machine learning algorithm to acquire knowledge of the data. The learned knowledge then can be used for classification problems as long as a task is ready.

Deep belief networks (DBNs) [2], as a semi-supervised learning algorithm, is promising to tackle the above-mentioned problems. It trains a multilayer neural network in a greedy fashion, each layer being a restricted Boltzmann machine (RBM) [3]. The trained weights and biases in each layer can be thought of as features or filters learned from the input data. The learned features are task-independent, so any tasks can take advantage of them as a starting point for classification problems.

DBNs have been mostly applied in handwriting recognition [2, 20], natural image recognition [73, 74], and modeling human motions [75]. When it comes to physiological data as input, [18] used DBNs to classify EEG signals to five clinically significant waves; [34] developed DBNs classifiers to determine sleep stages from EEG, EOG, and EMG. However, to the best of my knowledge, there is no existing work to predict emotional states using physiological data from biosensors using DBNs. This study explores such possibilities.

This work uses DEAP dataset [6] to show how DBNs learn features from raw physiological signals and predict emotion states.

DEAP is a multimodal dataset for the analysis of human affective states. The EEG and peripheral physiological signals (downsampled to 128Hz) of 32 subjects were recorded as each watched 40 one-minute long videos. The subjects rated the levels as continuous values of arousal, valence, liking, dominance, and familiarity.

This study predicts the levels of arousal, valence, and liking. The arousal scale ranges from calm or bored (1) to stimulated or excited (9). The valence scale ranges from unhappy or sad (1) to happy or joyful (9). Liking also has values from 1 to 9. All three ratings are float numbers.

32 EEG channels and totally 8 peripheral nervous system channels were recorded including hEOG (horizontal EOG), vEOG (vertical EOG), zEMG (Zygomaticus Major EMG), tEMG (Trapezius Major EMG), GSR (values from Twente converted to Geneva format in Ohm), respiration belt, plethysmograph, and body temperature.

This experiment uses 4 peripheral channels to do the predication, which are the two EOG channels and the two EMG channels. The EOG channels record eye movements. The activity of the Zygomaticus major is monitored in zEMG to capture a subject's laughs or smiles, whereas the Trapezius muscle (neck) is recorded by tEMG to reflect possible head movements.

[6] trained a Gaussian naïve Bayes classifier for each single subject due to the high inter-subject variability. For each subject, three different binary classifiers were trained and investigated to map the 8 peripheral channels to low (1-5) / high (5-9) arousal, valence, and liking, respectively. A leave-one-video-out cross validation was performed. In other words, in each trial a video was taken out for testing and the remaining 39 videos were used for training. As was done in most machine learning researches on physiological data, hand-engineered features such as eye blinking rate, energy of the signal, mean and variance of EMG and EOG were extracted. All the extracted features were fed into the classifier to train the model. Then the model was used to predict the test cases. The average accuracy over all subjects was 0.570, 0.627, and 0.591 for arousal, valence, and liking, respectively.

5.2 Related Works on Modeling Physiological Data and Emotion Prediction

To mapping physiological data to emotions, we need to train a classification model. The advance in machine learning provides many algorithms to get the job done [57].

[76, 77] used k-nearest neighbor. Naïve Bayes classifier and Bayesian networks were employed in [78]. Support vector machines have proved its success in [79, 80]. Classification tree performed well in [81]. Artificial neural networks were used in [82-84].

5.3 Train a DBN Classifier on Physiological Data

The overall goal of this study is to train a single DBN classifier for all the subjects on the raw data from two EEG and two EOG channels to predict arousal, valence, and liking as binary classes (low or high). Classification accuracy defined in (17) evaluates the performance.

There are five steps in the experiment: raw data pre-processing, raw data selection and division, normalization, randomization, and DBN training and classification.

In the first step, all signals are pre-processed by notch filtering at 50 Hz in order to remove power line disturbances and bandpass filters of 0.3 to 32 Hz and 10 to 32 Hz are also applied to EOG and EMG, respectively, as suggested by [34].

The second step selects raw data and divides them into training set and test set. A subject's physiological signals are more likely to be elicited by a video at the end of the one-minute watching period as the plot develops. Therefore, it sounds reasonable to discard the first 50 second and use only the last 10 seconds' data in each one-minute record. Then the 10 seconds' data are broke down into 10 one-second segments. In other words, the learning process trains and classifies one-second samples, each in 512 dimensions (128Hz * 1s * 4channel). There are totally 12800 samples (32subject * 40video * 10sample). In each trial, the 10 samples of one randomly chosen video from each subject are left out for testing, resulting in the size of test set being 320 (32 subject * 1video * 10sample). The remaining 12480 (32subject * 39video * 10sample) samples go to the training set.

The third step applies a channel-wise normalization to scale all the values to [0 1] according to

$$ch_{ij} = \frac{ch_{ij} - \min(ch_i)}{\max(ch_i) - \min(ch_i)} \tag{28}$$

where ch_i represents all the data in the channel i and ch_{ij} is a data point in the channel i. The reason to normalize data this way is two-folded: a) the ranges of different channels may vary, so normalization makes them comparable when concatenating all channels as input, and b) a DBN's node in the input layer has to have values between 0 and 1, to be treated as probabilities of activation of this node. Alternatively, the normalization could be done by saturating a signal at saturation constant max and min. Any values larger than max are set to max; any values smaller than min are set to min. Then apply (28). This trick should be able to remove outliers, too, but it requires some knowledge about the data's reasonable ranges. Since the DEAP dataset is quite clean, I did not find performance difference between these two normalization methods.

Randomizing training samples is necessary because the mini-batch technique in training DBNs requires samples of each class are (at least roughly) evenly distributed.

The last step firstly pre-trains a DBN without any labels, which means the same features learned in the pre-train stage can be used for the three different classification problems in the fine-tuning stage. Since the pre-trained model captures the properties of the data themselves, it can even be saved for unknown classification problems, as long as a new fine-tuning is applied to it using the new labeled data when they are available. After fine-tuning features based on labels and backpropagating to train a model for arousal classification, the model is used to predict arousal labels on the test set and compute the classification accuracy. The same process is applied on valence and liking classifiers.

The same experiment runs 10 times to get the mean accuracy, as well as the standard deviation.

The DBNToolbox matlab code published by [34] is provided to perform the experiment. A DBN with two hidden layers, each layer with 50 nodes is constructed. Therefore the DBN structure for the pre-training stage is 512-50-50 and for the fine-tuning stage is 512-50-50-2. DBN parameters are listed below.

Table 5 DBN parameters

Unsupervised learning rate	0.05
Supervised learning rate	0.05
Number of epochs in pre-training	50
Number of epochs in fine-tuning	20
Mini-batch size in both stages	100

5.4 Experimental Results

To show the features learned in the first hidden layer, I simply take w_{ij}, where i is these nodes corresponding to one channel in the visible layer and j is a fixed node in the hidden layer, and draw a one-dimensional graph for node j on one channel. In this experiment, 200 such feature graphs (4channel * 50node) are grouped into 4 channels each of which is shown as a big graph matrix shown below.

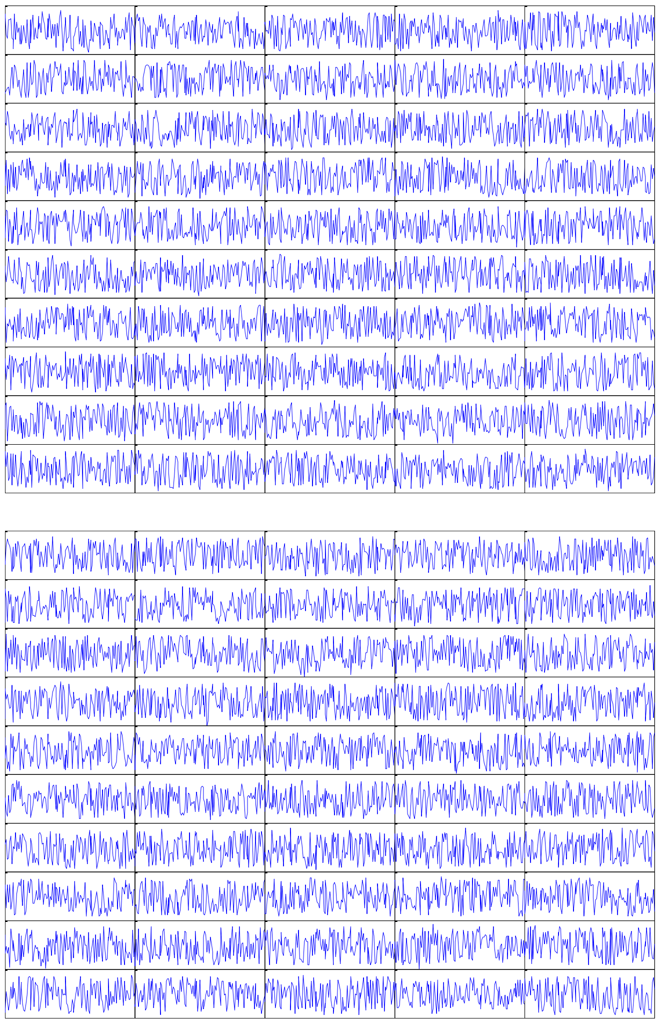


Fig. 46 Learned EOG features. (top) hEOG (buttom) vEOG

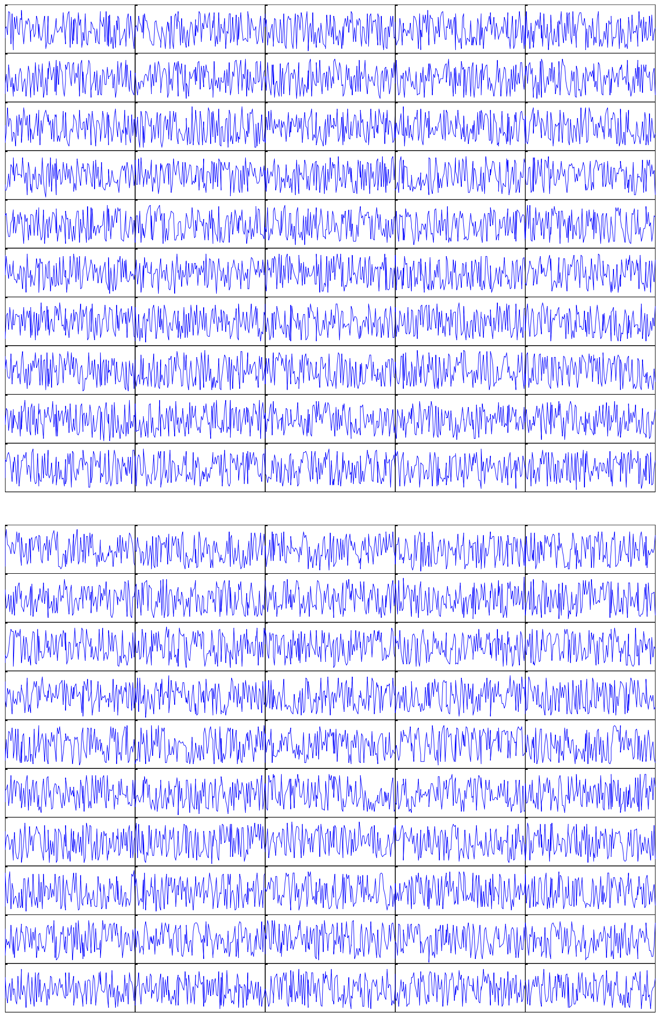


Fig. 47 Learned EMG features. (top) zEMG (bottom) tEMG

The whole experiment with 10 trials took about 1 hour on a Windows 7 machine with 3.0 G dual-core CPU and 4G memory.

The means and standard deviations of the accuracy of arousal, valence, and liking are 0.609/0.074, 0.512/0.097, and 0.684/0.093, respectively, which are drawn in Fig. 48. The classification accuracy in [6] is also depicted as filled dots for comparison purposes.

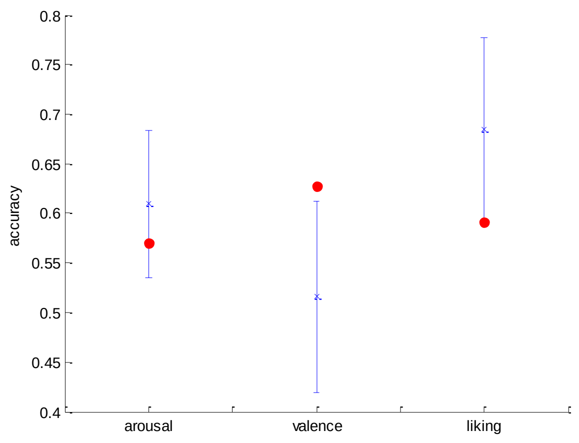


Fig. 48 Accuracy of DBN classification. Error bars for DBN classification accuracy on raw data and filled dots for Gaussian na ïve Bayes on features

Note that the three major differences when comparing the two results. Firstly, the original paper employs a subject specific classification approach, but this study trains a universal model for all subjects. Secondly, the original paper uses all the 8 peripheral channels, compared to only 4 channels used in this work, which contain less information. Thirdly, the original work takes hand-crafted features; in contrast, this work simply feeds the raw data into the DBN. In this sense, this study solves a harder problem using DBNs.

The liking accuracy of this work is higher than that of the original work; the arousal accuracy is slightly higher but not significant; however the valence accuracy is lower. There may be three reasons to explain the poorer valence classification performance: a) the 4 channels used in this work do not contain much useful information to discriminate different valences; b) the features of valence are too complex for the DBN in our settings to learn from the raw data; c) the inter-subject variability as claimed by [6] is too large for a single model to capture.

The distribution of DBN's classification performance in each subject is depicted below.

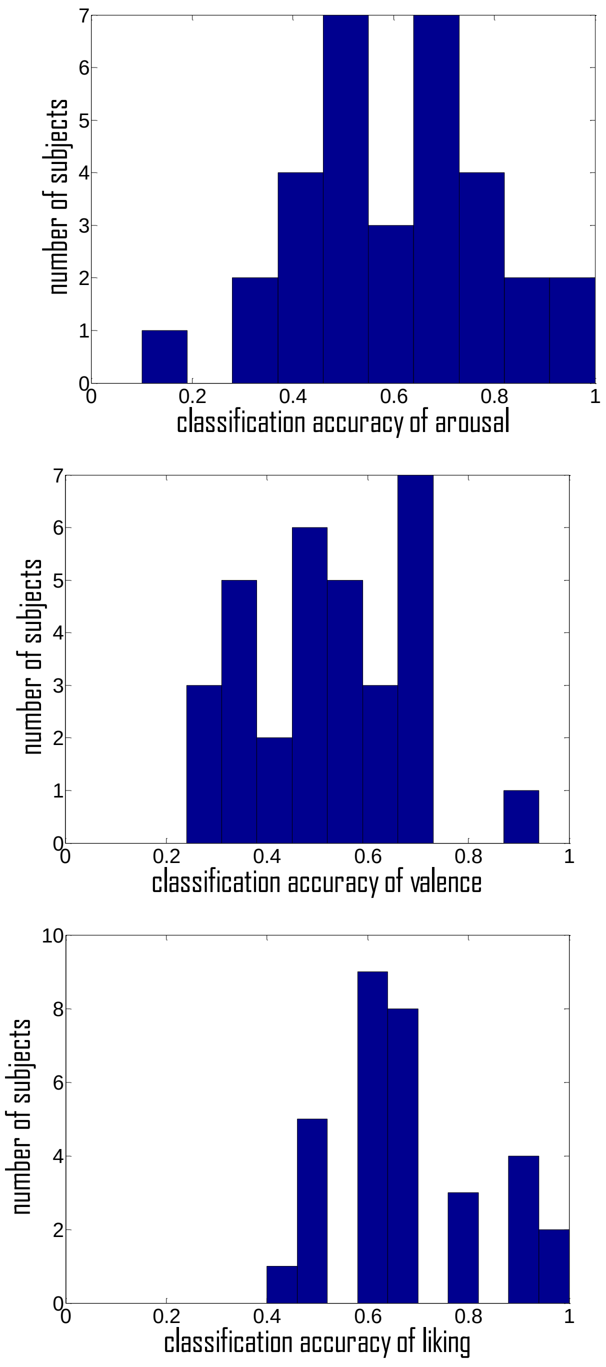


Fig. 49 Histogram of accuracy distribution in 32 subjects

5.5 Discussion

This work presents a system to apply DBNs to learn features from raw physiological data and predict emotions. The trained universal model for all subjects in the DEAP dataset shows that DBNs are capable to learn useful features in an unsupervised fashion and have comparable classification performance with Gaussian na ïve Bayes on hand-crafted features. This result suggests the possibilities of further applications of DBNs on physiological signal analysis and even other fields of similar complexity.

Chapter 6

Conclusion

Leveraging the recent breakthrough in deep neural networks provides means to learn deep layered hierarchical representation of data. This brings us new possibilities to be explored and new problems to be solved. My research focuses on developing practical pipelines, frameworks, and systems on real-world deep learning applications. Particularly, I have studied a proper way to search for the optimal deep learning structures and pre-processing techniques, a new active labeling framework for cost-effective selection of labeled data, and a pipeline to apply deep learning to emotion prediction via physiological sensor data.

I empirically examine the optimal meta-parameters of deep learning networks in terms of number of layers, number of nodes, and learning unit types, as well as the effects of various data pre-processing techniques on the benchmark MNIST hand-written digit dataset and a sleep stage dataset side by side. This is the first such kind of comprehensive investigation. The experimental results show some settings are sensitive to the number of hidden layers, some are sensitive to the number of nodes in each hidden layer, and the others are not sensitive to the network structure at all. This suggests the optimal meta-parameters are highly application dependent.

Inspired by the general active learning framework, I propose active labeling deep learning based on three metrics: least confidence, margin sampling, and entropy. This is the first work that fits deep learning into the active learning framework, to address the practical needs for cost-effective selection of labeled data in the paradigm of semi-supervised learning. On the MNIST dataset, the proposed methods outperform random labeling by 2%-4%, suggesting the usefulness of active labeling deep learning on clean datasets. On the other hand, the new method performs similarly to random labeling on the sleep stage dataset due to the noisiness and inconsistency in the data.

I also propose a pipeline to apply deep learning to emotion prediction via physiological sensor data. This is the first system for modeling physiological data by extracting features automatically and using extracted features to predict emotions. The developed system has three advantages: 1) it does not require expert knowledge for extracting features; 2) the features learned in the unsupervised learning stage can be used by multiple tasks; and 3) the classification accuracy is 0.690, 0.512, and 0.684, for the levels of arousal, valence, and liking, respectively, which is comparable with existing methods based on expert designed features.

Reference

[1] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," *Advances in neural information processing systems,* vol. 19, p. 153, 2007.

[2] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science,* vol. 313, pp. 504-507, 2006.

[3] P. Smolensky, "Information processing in dynamical systems: Foundations of harmony theory," in *Parallel distributed processing: explorations in the microstructure of cognition,* ed Cambridge, MA, USA: MIT Press, 1986, pp. 194-281.

[4] B. Settles, "Active learning literature survey," *University of Wisconsin, Madison,* 2010.

[5] S. H. Fairclough, "Fundamentals of physiological computing," *Interact. Comput.,* vol. 21, pp. 133-145, 2009.

[6] S. Koelstra, C. Muhl, M. Soleymani, J. Lee, A. Yazdani, T. Ebrahimi, *et al.*, "Deap: A database for emotion analysis using physiological signals," *Affective Computing, IEEE Transactions on,* pp. 1-1, 2011.

[7] M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken, "Multilayer feedforward networks with a nonpolynomial activation function can approximate any function," *Neural networks,* vol. 6, pp. 861-867, 1993.

[8] K.-I. Funahashi, "On the approximate realization of continuous mappings by neural networks," *Neural networks,* vol. 2, pp. 183-192, 1989.

[9] J. Hastad, "Almost optimal lower bounds for small depth circuits," in *Proceedings of the eighteenth annual ACM symposium on Theory of computing,* 1986, pp. 6-20.

[10] G. E. Hinton, S. Osindero, and Y. W. Teh, "A fast learning algorithm for deep belief nets," *Neural computation,* vol. 18, pp. 1527-1554, 2006.

[11] G. E. Hinton, "Training products of experts by minimizing contrastive divergence," *Neural computation,* vol. 14, pp. 1771-1800, 2002.

[12] G. Hinton, "A practical guide to training restricted Boltzmann machines," *Momentum,* vol. 9, p. 1, 2010.

[13] Y. Le Cun, "Modèles connexionnistes de l'apprentissage," 1987.

[14] H. Bourlard and Y. Kamp, "Auto-association by multilayer perceptrons and singular value decomposition," *Biological cybernetics,* vol. 59, pp. 291-294, 1988.

[15] G. E. Hinton and R. S. Zemel, "Autoencoders, minimum description length, and Helmholtz free energy," *Advances in neural information processing systems,* pp. 3-3, 1994.

[16] X. Zhu, "Semi-supervised learning literature survey," 2005.

[17] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE,* vol. 86, pp. 2278-2324, 1998.

[18] D. F. Wulsin, J. R. Gupta, R. Mani, J. A. Blanco, and B. Litt, "Modeling electroencephalography waveforms with semi-supervised deep belief nets: fast classification and anomaly measurement," *Journal of Neural Engineering,* vol. 8, p. 036015, 2011.

[19] D. Erhan, Y. Bengio, A. Courville, and P. Vincent, "Visualizing higher-layer features of a deep network," Technical report, University of Montreal2009.

[20] H. Lee, C. Ekanadham, and A. Ng, "Sparse deep belief net model for visual area V2," *Advances in neural information processing systems,* vol. 20, pp. 873-880, 2008.

[21] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng, "Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations," in *Proceedings of the 26th Annual International Conference on Machine Learning*, 2009, pp. 609-616.

[22] A. Ng. (2013). *UFLDL Tutorial*. Available: http://ufldl.stanford.edu/wiki/index.php/UFLDL_Tutorial

[23] G. E. Hinton, "To recognize shapes, first learn to generate images," *Progress in brain research,* vol. 165, pp. 535-547, 2007.

[24] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proceedings of the 25th international conference on Machine learning*, 2008, pp. 1096-1103.

[25] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *The Journal of Machine Learning Research,* vol. 9999, pp. 3371-3408, 2010.

[26] F. Seide, G. Li, and D. Yu, "Conversational Speech Transcription Using Context-Dependent Deep Neural Networks," in *INTERSPEECH*, 2011, pp. 437-440.

[27] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *Signal Processing Magazine, IEEE,* vol. 29, pp. 82-97, 2012.

[28] Y. Bengio, "Neural net language models," *Scholarpedia,* vol. 3, p. 3881, 2008.

[29] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural language processing (almost) from scratch," *The Journal of Machine Learning Research,* vol. 12, pp. 2493-2537, 2011.

[30] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, 2012, pp. 1106-1114.

[31] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv preprint arXiv:1207.0580,* 2012.

[32] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio, "Maxout networks," *arXiv preprint arXiv:1302.4389,* 2013.

[33] A. L. Goldberger, L. A. Amaral, L. Glass, J. M. Hausdorff, P. C. Ivanov, R. G. Mark, *et al.*, "Physiobank, physiotoolkit, and physionet components of a new research resource for complex physiologic signals," *Circulation,* vol. 101, pp. e215-e220, 2000.

[34] M. Längkvist, L. Karlsson, and A. Loutfi, "Sleep Stage Classification Using Unsupervised Feature Learning," *Advances in Artificial Neural Systems,* vol. 2012, 2012.

[35] A. R. Osborne and A. Provenzale, "Finite correlation dimension for stochastic systems with power-law spectra," *Physica D: Nonlinear Phenomena,* vol. 35, pp. 357-381, 1989.

[36] L. Zoubek, S. Charbonnier, S. Lesecq, A. Buguet, and F. Chapotot, "Feature selection for sleep/wake stages classification using data driven methods," *Biomedical Signal Processing and Control,* vol. 2, pp. 171-179, 2007.

[37] E. Pereda, A. Gamundi, R. Rial, and J. Gonzalez, "Non-linear behaviour of human EEG: fractal exponent versus correlation dimension in awake and sleep stages," *Neuroscience letters,* vol. 250, pp. 91-94, 1998.

[38] T. Gasser, P. Bächer, and J. Möcks, "Transformations towards the normal distribution of broad band spectral parameters of the EEG," *Electroencephalography and clinical neurophysiology,* vol. 53, pp. 119-124, 1982.

[39] S. Zhou, Q. Chen, and X. Wang, "Active deep networks for semi-supervised sentiment classification," in *Proceedings of the 23rd International Conference on Computational Linguistics: Posters*, 2010, pp. 1515-1523.

[40] D. D. Lewis and W. A. Gale, "A sequential algorithm for training text classifiers," in *Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*, 1994, pp. 3-12.

[41] S. Tong, "Active learning: theory and applications," Citeseer, 2001.

[42] T. Scheffer, C. Decomain, and S. Wrobel, "Active hidden markov models for information extraction," in *Advances in Intelligent Data Analysis*, ed: Springer, 2001, pp. 309-318.

[43] A. McCallum and K. Nigam, "Employing EM and Pool-Based Active Learning for Text Classification," in *ICML*, 1998, pp. 350-358.

[44] S. Tong and D. Koller, "Support vector machine active learning with applications to text classification," *The Journal of Machine Learning Research,* vol. 2, pp. 45-66, 2002.

[45] S. C. Hoi, R. Jin, and M. R. Lyu, "Large-scale text categorization by batch mode active learning," in *Proceedings of the 15th international conference on World Wide Web,* 2006, pp. 633-642.

[46] C. A. Thompson, M. E. Califf, and R. J. Mooney, "Active learning for natural language parsing and information extraction," in *ICML,* 1999, pp. 406-414.

[47] B. Settles and M. Craven, "An analysis of active learning strategies for sequence labeling tasks," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing,* 2008, pp. 1070-1079.

[48] S. Tong and E. Chang, "Support vector machine active learning for image retrieval," in *Proceedings of the ninth ACM international conference on Multimedia,* 2001, pp. 107-118.

[49] C. Zhang and T. Chen, "An active learning framework for content-based information retrieval," *Multimedia, IEEE Transactions on,* vol. 4, pp. 260-268, 2002.

[50] R. Yan, J. Yang, and A. Hauptmann, "Automatically labeling video data using multi-class active learning," in *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on,* 2003, pp. 516-523.

[51] A. G. Hauptmann, W.-H. Lin, R. Yan, J. Yang, and M.-Y. Chen, "Extreme video retrieval: joint maximization of human and computer performance," in *Proceedings of the 14th annual ACM international conference on Multimedia,* 2006, pp. 385-394.

[52] G. Tur, D. Hakkani-Tür, and R. E. Schapire, "Combining active and semi-supervised learning for spoken language understanding," *Speech Communication,* vol. 45, pp. 171-186, 2005.

[53] N. Roy and A. McCallum, "Toward optimal active learning through monte carlo estimation of error reduction," *ICML, Williamstown,* 2001.

[54] C. E. Shannon and W. Weaver, "A mathematical theory of communication," ed: American Telephone and Telegraph Company, 1948.

[55] R. R. Cornelius, *The science of emotion: Research and tradition in the psychology of emotions* vol. 133001539: Prentice Hall Upper Saddle River, NJ, 1996.

[56] D. Sander, D. Grandjean, and K. R. Scherer, "2005 Special Issue: A systems approach to appraisal mechanisms in emotion," *Neural networks,* vol. 18, pp. 317-352, 2005.

[57] D. Novak, M. Mihelj, and M. Munih, "A survey of methods for data fusion and system adaptation using autonomic nervous system responses in physiological computing," *Interacting with Computers*, vol. 24, pp. 154-172, 2012.

[58] D. McDuff, A. Karlson, A. Kapoor, A. Roseway, and M. Czerwinski, "AffectAura: an intelligent system for emotional memory," in *The ACM SIGCHI Conference on Human Factors in Computing Systems*, Austin, TX, 2012, pp. 849-858.

[59] E. W. Boyer, R. Fletcher, R. J. Fay, D. Smelson, D. Ziedonis, and R. W. Picard, "Preliminary Efforts Directed Toward the Detection of Craving of Illicit Substances: The iHeal Project," *Journal of Medical Toxicology,* pp. 1-5, 2012.

[60] K. Plarre, A. Raij, S. M. Hossain, A. A. Ali, M. Nakajima, M. Al'absi, *et al.*, "Continuous inference of psychological stress from sensory measurements collected in the natural environment," in *Information Processing in Sensor Networks (IPSN), 2011 10th International Conference on*, Chicago, IL, USA, 2011, pp. 97-108.

[61] S. D. Kreibig, "Autonomic nervous system activity in emotion: A review," *Biological Psychology,* vol. 84, pp. 394-421, 2010.

[62] J. Wagner, E. Andre, and F. Jung, "Smart sensor integration: A framework for multimodal emotion recognition in real-time," in *Affective Computing and Intelligent Interaction and Workshops, 2009. ACII 2009. 3rd International Conference on*, 2009, pp. 1-8.

[63] M. E. Miiller, "Why Some Emotional States Are Easier to be Recognized Than Others: A thorough data analysis and a very accurate rough set classifier," in *Systems, Man and Cybernetics, 2006. SMC '06. IEEE International Conference on*, 2006, pp. 1624-1629.

[64] C. Peter, E. Ebert, and H. Beikirch, "A wearable multi-sensor system for mobile acquisition of emotion-related physiological data," *Affective Computing and Intelligent Interaction,* pp. 691-698, 2005.

[65] R. W. Picard, "Affective computing: challenges," *International Journal of Human-Computer Studies,* vol. 59, pp. 55-64, 2003.

[66] M. M. Bradley and P. J. Lang, "Measuring emotion: Behavior, feeling, and physiology," *Cognitive neuroscience of emotion,* vol. 25, pp. 49-59, 2000.

[67] R. A. Calvo and S. D'Mello, "Affect detection: An interdisciplinary review of models, methods, and their applications," *Affective Computing, IEEE Transactions on,* vol. 1, pp. 18-37, 2010.

[68] G. Chanel, J. J. Kierkels, M. Soleymani, and T. Pun, "Short-term emotion assessment in a recall paradigm," *International Journal of Human-Computer Studies,* vol. 67, pp. 607-627, 2009.

[69] J. Kim and E. André, "Emotion recognition based on physiological changes in music listening," *Pattern Analysis and Machine Intelligence, IEEE Transactions on,* vol. 30, pp. 2067-2083, 2008.

[70] P. Rainville, A. Bechara, N. Naqvi, and A. R. Damasio, "Basic emotions are associated with distinct patterns of cardiorespiratory activity," *International journal of psychophysiology,* vol. 61, pp. 5-18, 2006.

[71] S. Thrun, "Is learning the n-th thing any easier than learning the first?," *Advances in neural information processing systems,* pp. 640-646, 1996.

[72] S. Thrun, "Learning to learn: Introduction," in *In Learning To Learn*, 1996.

[73] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," *Master's thesis, Department of Computer Science, University of Toronto,* 2009.

[74] P. Vincent, H. Larochelle, Y. Bengio, and P. A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proceedings of the 25th international conference on Machine learning*, 2008, pp. 1096-1103.

[75] G. W. Taylor, G. E. Hinton, and S. T. Roweis, "Modeling human motion using binary latent variables," *Advances in neural information processing systems,* vol. 19, p. 1345, 2007.

[76] C. A. Frantzidis, C. Bratsas, M. A. Klados, E. Konstantinidis, C. D. Lithari, A. B. Vivas*, et al.*, "On the classification of emotional biosignals evoked while viewing affective pictures: an integrated data-mining-based approach for healthcare applications," *Information Technology in Biomedicine, IEEE Transactions on,* vol. 14, pp. 309-318, 2010.

[77] C. Setz, B. Arnrich, J. Schumm, R. La Marca, G. Troster, and U. Ehlert, "Discriminating stress from cognitive load using a wearable EDA device,"

Information Technology in Biomedicine, IEEE Transactions on, vol. 14, pp. 410-417, 2010.

[78] D. J. Hand and K. Yu, "Idiot's Bayes—not so stupid after all?," *International Statistical Review,* vol. 69, pp. 385-398, 2001.

[79] G. E. Sakr, I. H. Elhajj, and H.-S. Huijer, "Support vector machines to define and detect agitation transition," *Affective Computing, IEEE Transactions on,* vol. 1, pp. 98-108, 2010.

[80] D. Wu, C. G. Courtney, B. J. Lance, S. S. Narayanan, M. E. Dawson, K. S. Oie*, et al.*, "Optimal arousal identification and classification for affective computing using physiological signals: virtual reality Stroop task," *Affective Computing, IEEE Transactions on,* vol. 1, pp. 109-118, 2010.

[81] J. N. Bailenson, E. D. Pontikakis, I. B. Mauss, J. J. Gross, M. E. Jabon, C. A. Hutcherson*, et al.*, "Real-time classification of evoked emotions using facial feature tracking and physiological responses," *International journal of human-computer studies,* vol. 66, pp. 303-317, 2008.

[82] V. Kolodyazhniy, S. D. Kreibig, J. J. Gross, W. T. Roth, and F. H. Wilhelm, "An affective computing approach to physiological emotion specificity: Toward subject-independent and stimulus-independent classification of film-induced emotions," *Psychophysiology,* vol. 48, pp. 908-922, 2011.

[83] F. Nasoz, C. L. Lisetti, and A. V. Vasilakos, "Affectively intelligent and adaptive car interfaces," *Information Sciences,* vol. 180, pp. 3817-3836, 2010.

[84] E. L. van den Broek, V. Lisý, J. H. Janssen, J. H. Westerink, M. H. Schut, and K. Tuinenbreijer, "Affective man-machine interface: unveiling human emotions through biosignals," in *Biomedical Engineering Systems and Technologies*, ed: Springer, 2010, pp. 21-47.

Publications

[1] Dan Wang and Yi Shang, “Modeling Physiological with Deep Belief Networks,” in *International Journal of Information and Education Technology*, vol. 3, no. 5, pp. 505-511, 2013.

[2] Dan Wang and Yi Shang, “Active Labeling in Deep Learning”, to be submitted.

[3] Dan Wang, Peng Zhuang, and Yi Shang, "A new framework for multi-source geo-social based mobile classifieds searches," in *Artificial Neural Networks In Engineering 2010*, St. Louis, MO, USA, 2010, pp. 129-135.

[4] Yi Shang, Wenjun Zeng, Dominic K. Ho, Dan Wang, Qia Wang, Yue Wang, Tiancheng Zhuang, Aleksandre Lobzhanidze, Liyang Rui, “Nest: NEtworked Smartphones for Target localization”, in *2012 IEEE Consumer Communications and Networking Conference (CCNC)*, Las Vegas, NV, USA, 2012, pp. 732-736.

[5] Peng Zhuang, Dan Wang, and Yi Shang, "SMART: Simultaneous indoor localization and map construction using smartphones," in *2010 International Joint Conference on Neural Networks (IJCNN2010)*, Barcelona, Spain, 2010©IEEE. doi: 10.1109/IJCNN.2010.5596552.

[6] Peng Zhuang, Dan Wang, and Yi Shang, "Distributed faulty sensor detection," in *2009 IEEE Global Telecommunications Conference*, Honolulu, Hawaii, USA, 2009©IEEE. doi: 10.1109/GLOCOM.2009.5425702.

VITA

Dan Wang was born in Chengdu, China. He is currently a PhD student in Computer Science Department at the University of Missouri, Columbia, MO 65211, USA. He got MS in Communication and Information System and BS in Communications Engineering from Southwest Jiaotong University, Chengdu, China, in 2007 and 2003, respectively. He has published 5 papers. His research interests include mobile computing, machine learning, and wireless sensor networks. He was a summer intern with Amazon Web Services in Seattle, WA, in 2011, 2012, and 2013. He will be joining Google as a software engineer.