Deep Adaptive Networks for Visual Data Classification

Shusen Zhou\textsuperscript{a}, Qingcai Chen\textsuperscript{b}, Xiaolong Wang\textsuperscript{b}

\textsuperscript{a} School of Information and Electrical Engineering, Ludong University, Yantai 264025, China
Email: zhoushusen@gmail.com

\textsuperscript{b} Shenzhen Graduate School, Harbin Institute of Technology, Shenzhen 518055, China
Email: qingcai.chen@hit.edu.cn, wangxl@insun.hit.edu.cn

Abstract—This paper proposes a classifier called deep adaptive networks (DAN) based on deep belief networks (DBN) for visual data classification. First, we construct a directed deep belief nets by using a set of Restricted Boltzmann Machines (RBM) and a Gaussian RBM via greedy and layer-wise unsupervised learning. Then, we refine the parameter space of the deep architecture to adapt the classification requirement by using global gradient-descent based supervised learning. An exponential loss function is utilized to maximize the separability of different classes. Moreover, we apply DAN to visual data classification task and observe an important fact that the learning ability of deep architecture is seriously underrated in real-world applications, especially when there are not enough labeled data. Experiments conducted on standard datasets of different types and different scales demonstrate that the proposed classifier outperforms the representative classification techniques and deep learning methods.

Index Terms—neural networks, deep learning, visual data classification

I. INTRODUCTION

Neural network (NN) is an important tool for classification [1], and has been widely used in many real-world problems [2], [3]. However, it did not succeed in using the multiple layers of features to significantly improve performance [4]. In order to learn the complicated functions that can represent high-level abstractions of images [5], [6], one needs deep architectures [7].

Deep architectures are much more efficient than shallow architectures in terms of computational elements and parameters that are required to represent specific functions [8]. However, it is difficult to optimize the weights in deep architectures. Hinton et al. introduced a greedy layer-wise unsupervised learning algorithm for deep belief networks (DBN) [9] to tackle this problem with notable success, which is a generative neural network model with many layers of hidden explanatory factors [10]. The building block of a DBN is a probabilistic model called Restricted Boltzmann Machine (RBM), which is used to represent one layer of the model.

There are many deep learning methods for classification using the greedy layer-wise unsupervised learning algorithm of DBN. Hinton et al. [9] introduced this greedy layer-wise unsupervised learning algorithm firstly, and got competitive result for handwriting digit classification. After greedy layer wise construction, the weights are fine-tuned by a contrastive version of the wake-sleep algorithm. Hinton and Salakhutdinov [11] described an effective way of initializing the weights, which allows deep autoencoder networks to learn low-dimensional codes that work much better than principal components analysis as a tool to reduce the dimensionality of data. Salakhutdinov and Hinton [12] combined the DBN and neighborhood component analysis (NCA) techniques to learn a nonlinear transformation from the input space to a low-dimensional feature space, in which $K$-nearest neighbor classification performs well. Ballan et al. [13] presented a novel approach for the classification of audio concepts in broadcast soccer videos using DBN, and showed that the results are promisingly comparable to the state-of-art approaches. Horster and Lienhart [14] explore deep networks for deriving a low-dimensional image representation appropriate for image retrieval.

Inspired by deep autoencoder networks [11] that are used for dimension reduction, this paper proposes a new classifier called deep adaptive networks (DAN) for classification. The multiple hidden layers are constructed by using a set of RBMs and the Gaussian RBMs are used in the output layer. An exponential loss function is utilized to refine the parameter space through maximizing the separability of different classes. Moreover, we apply DAN to various visual data classification tasks successfully.

This paper is an expanded version of Zhou et al. [15]. Many new contents are incorporated here: First, the related works of deep belief networks have been extended; more detail introduction about DAN has been made. Second, we use DAN to classify the handwriting Chinese characters, and get competitive performance. Third, more experiments have been conducted to evaluate the performance of deep architecture. Moreover, we evaluate the proposed deep learning method with a different number of labeled data, and different number of hidden layers with same number of hidden units.
The rest of the article is as follows. Section II proposes the deep learning method DAN. Section III shows the empirical validation of DAN by comparing the classification performance with other representative classifiers. The paper is closed with conclusion.

II. DEEP ADAPTIVE NETWORKS

In this part, we propose a deep learning algorithm, deep adaptive networks (DAN), based on the representative deep architecture DBN. DAN is adaptive from dimension reduction result to classification labels. After greedy layer-wise construction, the value of output layer is the abstraction of input images. Then after fine-tune supervised learning, the value of output layer is adaptive to the label of input images. We formulate the problem in Section II-A and provide the construction method of deep architecture in Section II-B. Section II-C discusses fine-tuning method by gradient descent supervised learning. Section II-D propose the training method of DAN.

A. Problem Formulation

For visual data classification tasks mentioned in this paper, all gray information of images are directly input into the classifiers .

Let $X$ be a set of samples, which can be seen as:

$$X = [x^1, x^2, \ldots, x^L] = \begin{bmatrix} x_1^1 & x_1^2 & \cdots & x_1^L \\ \vdots & \vdots & \ddots & \vdots \\ x_D^1 & x_D^2 & \cdots & x_D^L \end{bmatrix}$$  \hspace{1cm} (1)

where $L$ is the number of samples, $D$ is the number of features. Each column of $X$ is represented as a vector in $\mathbb{R}^D$, where the $j^{th}$ element corresponds to the $j^{th}$ feature.

Let $Y$ be a set of labels correspond to $L$ samples, which can be seen as:

$$Y = [y^1, y^2, \ldots, y^L] = \begin{bmatrix} y_1^1 & y_1^2 & \cdots & y_1^L \\ \vdots & \vdots & \ddots & \vdots \\ y_C^1 & y_C^2 & \cdots & y_C^L \end{bmatrix}$$  \hspace{1cm} (2)

$$y_j^i = \begin{cases} 1 & \text{if } x^i \in j^{th} \text{ class} \\ -1 & \text{if } x^i \notin j^{th} \text{ class} \end{cases}$$ \hspace{1cm} (3)

where $C$ is the number of classes. Each rank of $Y$ is a vector in $\mathbb{R}^C$, where the $j^{th}$ element of the vector corresponds to the $j^{th}$ class.

Our goal is to seek the mapping function $X \rightarrow Y$ by using $L$ samples. After training, we need to determine $y$ through the output of the deep network for a new input sample $x$.

B. Deep Architecture Construction

The architecture of DAN is shown in Fig. 1, which is a directed, fully interconnected (between consecutive layers) belief nets with one input layer $x$, $N$ hidden layers $h^1, h^2, \ldots, h^N$, and one output layer $f$. The input layer $x$ has $D_0$ units, equal to the number of feature dimensions. The output layer has $C$ units, equal to the number of classes in the dataset. $W = [w^1, w^2, \ldots, w^{N+1}]$ are the parameters that need to be learned. The number of hidden layers, and the number of units for each hidden layer are usually pre-defined according to the experience or intuition.

1) RBM: The energy of the state $(h^{k-1}, h^k)$ can be defined as [16]:

$$E(h^{k-1}, h^k; \theta) = - \sum_{i=1}^{D_k-1} \sum_{j=1}^{D_k} w_{ij} h_i^{k-1} h_j^k$$

$$- \sum_{i=1}^{D_k-1} b_i^{k-1} h_i^{k-1} - \sum_{j=1}^{D_k} c_j^k h_j^k$$ \hspace{1cm} (4)

where $\theta = (w, b, c)$ are the model parameters: $w_{ij}^k$ is the symmetric interaction term between unit $i$ in the layer $h^{k-1}$ and unit $j$ in the layer $h^k$, $b_i^{k-1}$ is the $i^{th}$ bias of layer $h^{k-1}$ and $c_j^k$ is the $j^{th}$ bias of layer $h^k$. The probability that the model assigns to a $h^k$ is:

$$P(h^k \mid h^{k-1}; \theta) = \frac{1}{Z(\theta)} \exp \left( - E(h^{k-1}, h^k; \theta) \right)$$ \hspace{1cm} (5)

$$Z(\theta) = \sum_{h^{k-1}} \sum_{h^{k}} \exp \left( - E(h^{k-1}, h^k; \theta) \right)$$ \hspace{1cm} (6)

where $Z(\theta)$ denotes the normalizing constant. The conditional distributions over $h^k$ and $h^{k-1}$ are given:

$$p(h_j^k \mid h^{k-1}) = \prod_j p(h_j^k \mid h^{k-1})$$ \hspace{1cm} (7)

$$p(h^{k-1} \mid h^k) = \prod_j p(h_j^{k-1} \mid h^k)$$ \hspace{1cm} (8)

the probability of tuning on unit $j$ is a logistic function of the states of $h^{k-1}$ and $w_{ij}^k$:

$$p(h_j^k = 1 \mid h^{k-1}) = \text{sign} \left( \sum_{i} w_{ij}^k h_i^{k-1} \right)$$ \hspace{1cm} (9)
the probability of tuning on unit \( i \) is a logistic function of the states of \( h^k \) and \( w_{ij}^k \):

\[
p(h_i^{k-1} = 1|h^k) = \text{sigm} \left( b_i^k + \sum_j w_{ij}^k h_j^k \right)
\]

(10)

where the logistic function is:

\[
\text{sigm}(\eta) = 1/(1+e^{-\eta})
\]

(11)

The derivative of the log-likelihood with respect to the model parameter \( w^k \) can be obtained by the CD-1 method [17]:

\[
\frac{\partial \log p(h^{k-1})}{\partial w_{ij}^k} = \langle h_i^{k-1}h_j^k \rangle_p - \langle h_i^{k-1} \rangle_p M
\]

(12)

where \( \langle h_i \rangle_p \) denotes an expectation with respect to the data distribution and \( \langle h_i \rangle_{ps} \) denotes a distribution of samples from running the Gibbs sampler initialized at the data, for \( M \) full steps.

According to the \( w^k \) calculated by RBM, the layer \( h^k \) is calculated as following when a sample \( x \) inputs from layer \( h^0 \):

\[
h_j^k(x) = \text{sigm} \left( c_j^k + \sum_{i=1}^{D_{k-1}} w_{ij}^k h_i^{k-1}(x) \right) \quad j = 1, \ldots, D_k, \quad k = 2, \ldots, N
\]

(13)

\[
h^1_j(x) = \text{sigm}(c_j^1 + \sum_{i=1}^{D_0} w_{ij}^1 x_i) \quad j = 1, \ldots, D_1
\]

(14)

The implementation of RBM with Matlab codes can be downloaded in Hinton's supporting online material.

2) Gaussian RBM: Gradient-based optimization starting from random initialization may get stuck near poor solutions [7]. However, \( w^{N+1} \) is random initialized by previous deep learning methods for classification [11].

DAN uses a Gaussian RBM to initialize the parameter \( w^{N+1} \). The Gaussian RBM can “abstract” the input data, and the global adjustment based on the “abstract” mapping can reach better classification performance. To adjust the deep architecture quickly, we use linear units for the output layer.

The hidden units of Gaussian RBM are modeled with stochastic real-valued states, and sampled from a Gaussian whose mean is determined by the input coming from \( h^N \).

The value of unit \( j \) of \( h \) is a linear function of the value \( h^N \) and \( w_{ij}^{N+1} \):

\[
f_j = c_j^{N+1} + \sum_i w_{ij}^{N+1} h_i^N
\]

(15)

where the number “0” indicates that the value is calculated without iteration.

The state of \( f \) is the value of \( f \) adds a random value \( r \) drawn from the standard normal distribution,

\[
s_j = f_j + r
\]

(16)

The new value of unit \( i \) of \( h^N \) is a logistic function of the state of \( f \) and \( w_{ij}^{N+1} \):

\[
h_{ij,1}^N = \text{sigm} \left( b_i^N + \sum_j w_{ij}^{N+1} s_j \right)
\]

(17)

where \( h_{ij,1}^N \) is the \( i \)th visible bias of \( N \)th layer. The number “1” indicate that the value is calculated with 1 iteration.

The new value of unit \( j \) of \( f \) is a linear function of the new value of \( h^N \) and \( w_{ij}^{N+1} \):

\[
f_j = c_j^{N+1} + \sum_i w_{ij}^{N+1} h_{ij,1}^N
\]

(18)

The derivative of the log-likelihood with respect to the model parameter \( w^{N+1} \) can be obtained by CD-1 method [17]:

\[
\frac{\partial \log p(h^N)}{\partial w_{ij}^{N+1}} = h_{ij,0} f_{j,0} - h_{ij,1} f_{j,1}
\]

(19)

After greedy training of output layer by Gaussian RBM, \( w^{N+1} \) is initialized. The output layer \( f \) can be calculated by:

\[
f_j(x) = c_j^{N+1} + \sum_{i=1}^{D_N} w_{ij}^{N+1} h_i^N(x) \quad j = 1, \ldots, C
\]

(20)

The implementation of Gaussian RBM is similar with RBM, except that the stochastic real-valued states of RBM are replicated with a unit variance Gaussian whose mean is determined by \( h^N \) with a linear calculation.

The above discussion is based on one sample data \( x \). In DAN, we start constructing the deep architecture by using labeled and unlabeled information of all data through feeding them one by one from layer \( h^0 \), and training the parameters between \( h^0 \) and \( h^1 \) by RBM. Then \( h^1 \) is constructed, and the value of \( h^1 \) is calculated by \( h^0 \) and the trained parameters between \( h^0 \) and \( h^1 \). We thus use \( h^1 \) to construct the next layer \( h^2 \). In this way, the deep architecture is constructed layer by layer from bottom to top. At each time, the parameter space \( w^k \) is trained by the calculated data in the \((k-1)^{th}\) layer. At last, the output layer \( f \) is constructed with Gaussian RBM, the parameter space \( w^{N+1} \) is trained by the calculated data in the \(N^{th}\) layer.

C. Fine-tune Method

To global fine-tune DAN for classification, we add an exponential loss function to the output layer \( f \) of the architecture. It guides the mapping results of samples that belong to different classes far away from each other. The exponential loss function is given as:

\[
E(z) = \exp(-z)
\]

(21)

By using the exponential loss function, the optimization problem is formalized as:

\[
\argmin_f \sum_{i=1}^{L} \ell \left( f(x^i), y^i \right)
\]

(22)
where
\[ \ell(f(x), y) = \sum_{j=1}^{C} E(f_j(x) y_j) \] (23)

We use gradient-descent through the whole DAN to refine the weight space. In the supervised learning stage, the stochastic activities are replaced by deterministic, real valued probabilities, and continuous values are used in all nodes of the deep architecture.

The implementation of supervised learning stage is based on the minimize function\(^2\), which minimizes the value of the loss function by adjusting the parameters of the whole deep architecture.

D. Training Method of DAN

![Diagram of DAN](http://www.gatsby.ucl.ac.uk/~edward/code/minimize/)

Figure 2. Training method of DAN.

The training method of DAN is illustrated with Fig. 2. We do not consider about the classical image encoding methods, since the deep architecture can encode and classify the images effectively. In the initialization stage, DAN is constructed through greedy layer-wise unsupervised learning by using RBM and Gaussian RBM as building blocks. \( L \) training data are utilized to seek the parameter space \( \mathbf{W} \). The visible units of RBM have real valued activation probabilities, which are in the range of \( [0, 1] \) for logistic units. The visible units of first level RBM are normalized training data. While training higher level RBM, the visible units are set to the activation probabilities of the hidden units in previous RBM, the hidden units of each RBM, except the Gaussian RBM, have stochastic binary values. The hidden units of the Gaussian RBM are modeled with stochastic real-valued states, and sampled from a Gaussian whose mean is determined by its input. In the fine-tune supervised learning stage, the stochastic activities are replaced by deterministic, real valued probabilities. We use conjugate gradient through the whole deep architecture to fine-tune the weights for global optimal classification.

The procedure of DAN is shown in Algorithm 1.

---

Algorithm 1: Algorithm of DAN

**Input:** data \( \mathbf{X}, \mathbf{Y} \);
- number of layers \( N \);
- number of epochs \( Q \);
- number of units in every hidden layer \( D_1,...,D_N \);
- parameter space \( \mathbf{W} = \{ w^1, ..., w^N \} \)

**Output:** deep architecture with parameter space \( \mathbf{W} \)

1. Construct hidden layers using RBM
2. Construct output layer using Gaussian RBM
3. Optimize deep network with gradient descent

Minimize \( \ell(f(x), y) \) on \( L \) training data, update the parameter space \( \mathbf{W} \) according to Eq. 22.

---

III. EXPERIMENTS

The performance of DAN method is evaluated by using three real databases, which include six datasets as shown in Table I. The first database is a handwriting database HIT-OR3C, that consists of digits, letters and Chinese characters [18]. Capital, Lowercase, GB1 and GB2 subsets are used in our experiments. The second database is a standard dataset Caltech 101, that contains images of 101 different objects, plus with a background category [19], and just a subset of it is used. The third database is MNIST\(^3\), which is a standard dataset for empirical validation of deep learning algorithms [9].

The parameters of DAN are modified based on the default parameter settings of Hintons DBN package [11]. For network initializing stage, we train the weights of each layer separately with the 30 epochs and the learning rate is set to 0.1. The initial momentum is 0.5, and after 5 epochs, the momentum is set to 0.9. For fine-tune supervised learning stage, we train the DAN 50 epochs with \( L \) labeled data.

The classification performance of DAN is compared with the performance of four representative classifiers, include the neural network (NN) [20], support vector machines (SVM) [21], deep belief network (DBN) [11] and DBN with regularized the neighborhood component analysis (DBN-rNCA) [12]. NN is a classical back propagation neural networks [20]. SVM is a powerful classification method, and LIBSVM is used in our experiment, which is generally considered as the best opening SVM implementation [21]. DBN is a classical deep learning method proposed recently [9]. DBN-rNCA is a new deep learning method modified based on DBN [12].

---

\(^2\)http://www.gatsby.ucl.ac.uk/~edward/code/minimize/

\(^3\)http://yann.lecun.com/exdb/mnist/
TABLE I.
DATASETS USED IN THE EXPERIMENTS.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Categories</th>
<th>Dimensions</th>
<th>Data points</th>
<th>Labeled data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capital</td>
<td>26</td>
<td>1,024</td>
<td>3,120</td>
<td>624</td>
</tr>
<tr>
<td>Lowercase</td>
<td>26</td>
<td>1,024</td>
<td>3,120</td>
<td>624</td>
</tr>
<tr>
<td>Caltech 101 (subset)</td>
<td>5</td>
<td>400</td>
<td>2935</td>
<td>1467</td>
</tr>
<tr>
<td>MNIST</td>
<td>10</td>
<td>784</td>
<td>70,000</td>
<td>60,000</td>
</tr>
<tr>
<td>GB1</td>
<td>3755</td>
<td>1,024</td>
<td>458,110</td>
<td>375,500</td>
</tr>
<tr>
<td>GB2</td>
<td>3008</td>
<td>1,024</td>
<td>366,976</td>
<td>300,800</td>
</tr>
</tbody>
</table>

A. Small-scale Dataset

We demonstrate the performance of DAN on 2 subsets of HIT-OR3C database, which are Capical and Lowercase. There are 26 categories (A-Z) in Capical subset, and 26 categories (a-z) in Lowercase subset. Each subset is written by 122 persons, so there are 3172 handwriting images in Capical and Lowcase subsets respectively. The samples of these 2 subsets are illustrated in Fig. 3.

![Sample images](image)

Figure 3. Samples of Capital and Lowercase subsets.

In this experiment, 4 hidden layers are used for DAN architecture, and the number of units in hidden layer are 500-500-500-2000. We conduct experiments with 5-fold cross validation, which partition the images into 5 splits, 1 split as labeled examples and the rest of the splits are unlabeled. So there are 634 labeled data in Capical and Lowcase subsets respectively. All the data are normalized to the range [0, 1].

5-fold cross validation are conducted for all the classifiers and the classification error rates are reported in Table II. The table shows that the performance of DAN on Capital and Lowercase subsets are much better than other classifiers.

![Classification error rates](image)

Figure 4. Average and variance of error rates for various classifiers on subset of Caltech 101.

TABLE II.
CLASSIFICATION ERROR RATE ON LOWERCASE AND CAPITAL SUBSETS OF HIT-OR3C

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Capital</th>
<th>Lowercase</th>
</tr>
</thead>
<tbody>
<tr>
<td>NN</td>
<td>36.63</td>
<td>44.06</td>
</tr>
<tr>
<td>SVM</td>
<td>34.32</td>
<td>42.03</td>
</tr>
<tr>
<td>DBN</td>
<td>37.98</td>
<td>42.49</td>
</tr>
<tr>
<td>DBN-rNCA</td>
<td>30.18</td>
<td>36.72</td>
</tr>
<tr>
<td>DAN</td>
<td>20.17</td>
<td>25.49</td>
</tr>
</tbody>
</table>

B. Middle-scale Dataset

We evaluate the performance of DAN on a subset of Caltech 101, because there are few samples in most categories of Caltech 101, this can not demonstrate the powerful abstract ability of deep architecture. So in this experiment, the images from the first five categories are used, which include more samples for image classification learning. In this experiment, we use 3 hidden layers for DAN architecture, and the number of units in hidden layers are 150-150-600. Half of the 2935 samples are used as the training set and the remaining samples as the test set. All the samples are normalized to the range [0, 1].

We choose the training set randomly with 10 times, and get the average and variance of classification error rate with all evaluated classifiers. As shown in Fig. 4, DAN outperforms other classifiers.

![Training time comparison](image)

Figure 5. Training time of various classifiers on subset of Caltech 101.

The training time of various classifiers on subset of Caltech 101 can be seen in Fig. 5. The running time of various classifiers on a single image of Caltech 101 are shown in Fig. 6. Comparing with other classifiers, the training time of DAN is just better than NN. However, the running time of DAN is just worse than NN, which is
TABLE III.
CLASSIFICATION ERROR RATE ON MNIST DATA

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Error rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>NN [20]</td>
<td>1.60</td>
</tr>
<tr>
<td>SVM [21]</td>
<td>1.41</td>
</tr>
<tr>
<td>DBN-rNCA [12]</td>
<td>1.00</td>
</tr>
<tr>
<td>DAN</td>
<td><strong>0.95</strong></td>
</tr>
</tbody>
</table>

C. Large-scale Dataset

MNIST is a handwritten digit database that contains 60,000 training images and 10,000 test images. In this experiment, we use a 500-500-2000 net for DAN architecture.

The classification results of MNIST data are shown in Table III. All these results, except DAN and SVM, are reported by previous publications. We can see that the performance of DAN is better than compared methods.

To evaluate performance of the proposed initialization method, we use random number to initialize the parameter space $w_{N+1}$, and $w_1, w_2, \ldots, w_N$ are initialized by RBM.

D. Huge-scale Dataset

To verify the performance of DAN further, we use GB1 and GB2 subsets of HIT-OR3C database in this experiment, which are handwriting Chinese character dataset. GB1 subset owns 3,755 classes of GB2312-80 Set1. GB2 subset owns 3,008 classes of GB2312-80 Set2. Both sets are written by 122 persons. The properties of these two subsets are shown in Table I. Fig. 8 shows some samples of these 2 subsets, the first row is the samples of GB1 subset, and the second row is the samples of GB2 subset.

For the following handwriting Chinese character recognition experiments, 375,500 characters in GB1 subset are used for training and the rest 82,610 characters for testing, 300,800 characters in GB2 subset are used for training and the rest 66,176 characters for testing. We use 3 hidden layers for DAN architecture, and the number of units in every hidden layer are 500-500-2000. To be compared with classical handwriting Chinese character recognition method, which uses KMean for coarse classification, and modified quadratic discriminant function (MQDF) classifier for fine classification, called KMean+MQDF. We also use DAN method for coarse classification of Chinese character, and use MQDF classifier for fine classification, called DAN+MQDF. This method can be described as:

1) Reduce the dimension of images to $64 \times 64$, normalize all the images with modified centroid-boundary alignment (MCBA) method [22].
2) Gradient direction feature is extracted with Sobel operator [23], then the feature dimensionality is reduced from 512 to 160 by Fisher linear discriminant analysis (LDA).
3) The characters are coarse classified with DAN method, and MQDF classifier is used for fine classification.

To compare with classical deep learning method, we also use DBN method for handwriting Chinese character recognition, and use it for coarse classification of Chinese character too. For the limitations of DBN-rNCA method, it can not be run on huge-scale dataset. For the high time complexity, we also can not get the results with NN and SVM methods on huge-scale dataset after continue
running for one month. So we do not report results for these three methods on GB1 and GB2 subsets.

Figure 9. Classification accuracy of various classifiers on GB1 subsets of HIT-OR3C.

Classification accuracy of various classifiers on GB1 and GB2 subsets of HIT-OR3C can be seen in Fig. 9 and Fig. 10 respectively. Through these figures, we can see that the performances of DAN on GB1 and GB2 subsets are much better than DBN. The performance of DAN+MQDF method on both GB1 and GB2 subsets are all better than DBN+MQDF and KMean+MQDF methods. The performance of DBN method on GB1 and GB2 subsets is very bad, the main reason is that there are more than 3,000 classes in GB1 and GB2 subsets, the loss function used in classical DBN method is not able to discriminate these Chinese characters accurately.

E. Deeper Experiment

In this paper, we also conduct a set of experiments on MNIST dataset with very deep architectures (up to 15 layers), with the top hidden layer has 2000 units, and the other hidden layers have 500 units. The classification results of DAN with different number of hidden layers are given in Fig. 11. When using 3 layers, we obtain the best classification error rate 0.95%. However, when the number of hidden layers increasing, the results are going to the bad. This is a more general phenomenon, when the number of hidden layer is less than this optimal number, the error rate is decreasing as the number of hidden layer increasing. This because the deeper architecture can abstract the dataset properly and improve the classification result effectively. When the number of hidden layer is more than this optimal number, the error rate is increasing as the number of hidden layer increasing. This because although the deeper architecture can improve the abstraction ability, the classification ability of deep architecture trained in fine tune stage can not be improved, more parameters need to be trained in deeper architecture.

The test errors of DAN with different number of hidden layers and different number of epochs are shown in Fig. 12. Through the figure, we can see that the deep network has the best classification performance when there are 3 hidden layers. Moreover, it converges quicker than 2 hidden layers and 4 hidden layers deep network.

We add an experiment to show the optimal network depth as a function of training set size, which can be seen in Fig. 13. Through the figure, we can see that when the size of the learning set is increasing, the number of hidden layer for optimal deep architecture is decreasing. This because although the deeper architecture can improves the abstraction ability with greedy layer-wise construction, it decreases the classification ability in the fine-tune stage, because more parameters need to be trained in the deeper.
architecture. On the other hand, the abstraction result of deep architecture can improve the classification ability of deep architecture when there are not enough training data. When the size of the training set is increasing, the classification performance of deep architecture relies on less abstraction ability and more classification ability, so the number of hidden layer for optimal deep architecture is decreasing. The corresponding error rate for these different training sets are shown in Fig. 14. We can see that when the number of the training data is bigger than 20,000, the error rate is decreasing, which means the classification performance is improved.

In the third experiment, we fix the number of hidden units equal to 4096 and vary the depth of the deep network as shown in Table IV. Fig. 15 shows the classification error rates for using 60,000 labeled data and 10,000 unlabeled data. The performance keeps increasing until the number of hidden layers equal to 8. Compared with the first experiment, we find that when the number of hidden layers equal to 3,000 (500+500+2,000), the learning performance peaks at depth 3. So with the same training set, the deeper architecture with less number of units in each hidden layer may get better performance at the beginning. However, when the architecture is deep enough, the performance also gets worse.

In Fig. 15, we also provides the relationship between the running time and the depth of the deep architecture with the same scale. The operate system we used in experiment is SUSE Linux Enterprise 11 SP1 (OEM). The CPU speed is 1.86GHz. Same with the observation of Liu et al. [24], deep learning is more effective and at the same time more efficient for classification task. For example, under the same scale and the same number of labeled and unlabeled data, DAN with eight hidden layers can achieve better accuracy as well as lower running time than DAN with two hidden layers.

### IV. Conclusion

In this work, we show how to improve supervised learning for deep architectures if one use RBM to construct hidden layers, uses Gaussian RBM to construct the output layer, and then uses exponential loss function to fine-tune the whole network. Because RBM can abstract the information of the data effectively, Gaussian RBM and exponential loss function can construct the connection between data and its corresponding label effectively. The experiments in HIT-OR3C dataset show that the exponential loss function and random initialize method used in deep architecture can reach good performance even there are many classes in the dataset. The experiments in Caltech 101 dataset show that the accuracy and efficiency of DAN are better than other classifiers. The experiments in MNIST dataset show that the Gaussian RBM initialization method can get better classification performance than random initialization method for visual data classification. The experiments for handwriting Chinese character recognition show that DAN can improve the recognition accuracy with coarse classification. Moreover, we study the optimal network depth as a function of training set size, and find out that the number of hidden layer for optimal deep architecture is decreasing as the size of the training set is increasing. We fix the number of hidden units and vary the depth of the deep network, and find out that the deeper architecture with less number of unit in every hidden layer can get better performance at the beginning. However, when the architecture is deep enough, the performance is getting worse.

### Acknowledgment

The authors are grateful to the anonymous referees for their valuable comments and suggestions to improve the presentation of this paper.
Figure 15. Classification error rate and real running time for DAN method with different number of hidden layers and same number of hidden units.

REFERENCES


Shusen Zhou received the Ph.D. degree in computer application technology from the Harbin Institute of Technology in 2012. He is currently an assistant professor in Ludong University. His main research interests include machine learning, artificial intelligence, multimedia content analysis and computational linguistics.

Qingcai Chen received the Ph.D. degree in computer science from the Computer Science and Engineering Department, Harbin Institute of Technology. From September 2003 to August 2004, he worked for Intel (China) Ltd. as a senior software engineer. Since September 2004, he has been with the Computer Science and Technology Department of Harbin Institute of Technology Shenzhen Graduate School as an associate professor. His research interests include machine learning, pattern recognition, speech signal processing, and natural language processing.

Xiaolong Wang received the B.E. degree in computer science from the Harbin Institute of Electrical Technology, Harbin, China, in 1982, the M.E. degree in computer architecture from Tianjin University, Tianjin, China, in 1984, and the Ph.D. degree in computer science and engineering from the Harbin Institute of Technology in 1989. He was an assistant lecturer in 1984 and an associate professor in 1990 with the Harbin Institute of Technology. From 1998 to 2000, he was a senior research fellow with the Department of Computing, Hong Kong Polytechnic University, Kowloon. He is currently a professor of computer science with the Harbin Institute of Technology Shenzhen Graduate School. His research interest includes artificial intelligence, machine learning, computational linguistics, and Chinese information processing.