SVM-Based Deep Stacking Networks

Jingyuan Wang,[†,§] **Kai Feng,**[†] **Junjie Wu**[‡,§*]

†MOE Engineering Research Center of Advanced Computer Application Technology,
School of Computer Science Engineering, Beihang University, Beijing 100191, China
‡Beijing Key Laboratory of Emergency Support Simulation Technologies for City Operations,
School of Economics and Management, Beihang University, Beijing 100191, China
§Beijing Advanced Innovation Center for Big Data and Brain Computing,
Beihang University, Beijing 100191, China
Email: {jywang, fengkai, wujj}@buaa.edu.cn

Abstract

The deep network model, with the majority built on neural networks, has been proved to be a powerful framework to represent complex data for high performance machine learning. In recent years, more and more studies turn to nonneural network approaches to build diverse deep structures, and the Deep Stacking Network (DSN) model is one of such approaches that uses stacked easy-to-learn blocks to build a parameter-training-parallelizable deep network. In this paper, we propose a novel SVM-based Deep Stacking Network (SVM-DSN), which uses the DSN architecture to organize linear SVM classifiers for deep learning. A BP-like layer tuning scheme is also proposed to ensure holistic and local optimizations of stacked SVMs simultaneously. Some good math properties of SVM, such as the convex optimization, is introduced into the DSN framework by our model. From a global view, SVM-DSN can iteratively extract data representations layer by layer as a deep neural network but with parallelizability, and from a local view, each stacked SVM can converge to its optimal solution and obtain the support vectors, which compared with neural networks could lead to interesting improvements in anti-saturation and interpretability. Experimental results on both image and text data sets demonstrate the excellent performances of SVM-DSN compared with some competitive benchmark models.

Introduction

Recent years have witnessed the tremendous interests from both the academy and industries in building deep neural networks (Hinton and Salakhutdinov 2006; Bengio, Courville, and Vincent 2013). Many types of deep neural networks have been proposed for classification, regression and feature extracting tasks, such as Stacked Denoising Autoencoders (SAE) (Vincent et al. 2010), Deep Belief Networks (DBN) (Hinton 2011), deep Convolutional Neural Networks (CNN) (Krizhevsky, Sutskever, and Hinton 2012), Recurrent Neural Networks (RNN) (Medsker and Jain 2001), and so on.

Meanwhile, the shortcomings of neural network based deep models, such as the non-convex optimization, hard-to-parallelizing, and lacking model interpretation, are getting more and more attentions from the pertinent research

---

societies. Some potential solutions have been proposed to build deep structure models using non neural network approaches. For instance, in the literature, the PCANet build a deep model using an unsupervised convolutional principal component analysis (Chan et al. 2015). The gcForest builds a tree based deep model using stacked random forests, which is regarded as a good alternative to deep neural networks (Zhi-Hua Zhou 2017). Deep Fisher Networks build deep networks by stacking Fisher vector encoding into multiple layers (Simonyan, Vedaldi, and Zisserman 2013).

Along this line, in this paper, we propose a novel SVM-based Deep Stacking Network (SVM-DSN) for deep machine learning. On one hand, SVM-DSN belongs to the community of Deep Stacking Networks (DSN), which consist of many stacked multilayer base blocks that could be trained in a parallel way and have comparable performance with deep neural networks (Deng and Yu 2011; Deng, He, and Gao 2013). In this way, SVM-DSN can gain the deep learning ability with extra scalability. On the other hand, we replace the traditional base blocks in a DSN, *i.e.*, the perceptrons, by the well known Support Vector Machine (SVM), which has long been regarded as a succinct model with appealing math properties such as the convexity in optimization, and was considered as a different method to model complicated data distributions compared with deep neural networks (Bengio and others 2009). In this way, SVM-DSN can gain the ability in anti-saturation and enjoys improved interpretability, which are deemed to be the tough challenges to deep neural networks. A BP-like Layered Tuning (BLT) algorithm is then proposed for SVM-DSN to conduct holistic and local optimizations for all base SVMs simultaneously.

Compared with the traditional deep stacking networks and deep neural networks, the SVM-DSN model has the following advantages:

- The optimization of each base-SVM is convex. Using the proposed BLT algorithm, all base-SVMs are optimized as a whole, and meanwhile each base-SVM can also converge to its own optimum. The final solution of SVM-DSN is a group of optimized linear SVMs that are integrated as a deep model. This advantage allows SVM-DSN to avoid the neuron saturation problem in deep neural networks, and thus could improve the performance.

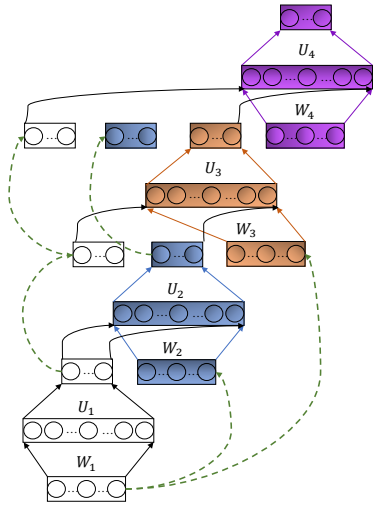- The SVM-DSN model is very easy to parallelize. The

Figure 1: An illustration of the DSN architecture (Deng, He, and Gao 2013). The color is used to distinguish different blocks in a DSN. The components in the same color belong to the same block.

training parallelization in SVM-DSN can reach the base-SVM level due to the support vectors oriented property of SVM, but the traditional DSN can only reach the block level.

- The SVM-DSN model has improved interpretability. The support vectors in base-SVMs can provide some insightful information about what a block learned from training data. This property empowers users to partially understand the feature extracting process of the SVM-DSN model.

Experimental results on image and sentiment classification tasks show that SVM-DSN model obtains respectable improvements over neural networks. Moreover, compared with the stacking models with strong base-learners, the SVM-DSN model also demonstrates significant advantages in performance.

The SVM-DSN Model
Framework of Deep Stacking Network

The Deep Stacking Network is a scalable deep machine learning architecture (Deng, He, and Gao 2013; Deng and Yu 2011) that consists of stacked easy-to-learn blocks in a layer by layer manner. In the standard DSN, a block is a simplified multilayer perceptron with a single hidden layer. Let the inputs of a block be a vector $\boldsymbol{x}$, the block uses a connection weight matrix $\boldsymbol{W}$ to calculate the hidden layer vector $\boldsymbol{h}$ as

$$\boldsymbol{h} = \varphi\left(\boldsymbol{W}^\top \boldsymbol{x}\right), \tag{1}$$

where $\varphi(x) = 1/(1 + \exp(-x))$ is a sigmoid nonlinear activation function. Using a weight matrix $\boldsymbol{U}$, the objective function of the DSN block optimization is defined as

$$\min \left\| \boldsymbol{y} - \boldsymbol{U}^\top \boldsymbol{h} \right\|_F^2. \tag{2}$$

As shown in Fig. 1, the blocks of a DSN are stacked layer by layer. For the block in the input layer, the input vector $\boldsymbol{x}$ contains only the raw input features. For blocks in the middle layers, $\boldsymbol{x}$ is a concatenated vector of the raw input features and output representations of all previous layer blocks.

The training of deep stacking networks contains two steps: block training and fine-tuning. In the block training step, the DSN blocks are independently training as supervised multilayer perceptrons. In the fine-tuning step, all the stacked blocks are considered as a multi-layer deep neural network. The parameters of DSN are end-to-end trained using the error Back Propagation (BP) algorithm.

SVM-based DSN Blocks

In the SVM-DSN model, we adopt support vector machines to implement a DSN block. A SVM classifier is a hyperplane $\boldsymbol{\omega}^\top \boldsymbol{x} + b = 0$ that divides the feature space of a data sample $\boldsymbol{x}$ into two parts — one for the positive and the other for the negative. The parameters $\boldsymbol{\omega}$ and b are optimized to maximize the minimum distances from the hyperplane to a set of training samples $T = \{(\boldsymbol{x}_k, y_k) | y_k \in \{-1, 1\}, k = 1, \ldots, K\}$, *i.e.*,

$$\begin{aligned} \max_{\boldsymbol{\omega}, b} \quad & \frac{2}{\|\boldsymbol{\omega}\|} \\ s.t. \quad & y_k(\boldsymbol{\omega}^\top \boldsymbol{x}_k + b) \geq 1, \quad k = 1, 2, \ldots, N. \end{aligned} \tag{3}$$

A training sample is called a *support vector* if the constraint in Eq. (3) turns into equality.

For a multi-class problem with N classes, we connect the input vector $\boldsymbol{x}$ of a DSN block with N binary SVM classifiers — each for recognizing whether a sample belongs to a corresponding class — to predict the label of a sample. A binary SVM classifier in a DSN block is called a *base-SVM*. The N binary SVM classifiers for a N classification problem is called as a *base-SVM group*. A SVM-DSN block could contains multiple base-SVM groups. In the same block, all base-SVM groups share the same input vector $\boldsymbol{x}$.

Stacking Blocks

Given a classification hyperplane of a SVM, the decision function for the sample $\boldsymbol{x}_k$ is expressed as

$$f(\boldsymbol{x}_k) = \text{sign}\left(\boldsymbol{\omega}^\top \boldsymbol{x}_k + b\right), \tag{4}$$

where $f(\boldsymbol{x}_k) = 1$ for the positive class and $f(\boldsymbol{x}_k) = -1$ for the negative. The distance from a sample to the hyperplane could be considered as the *confidence* of a classification decision. For the samples behind the support vectors, *i.e.*, $\left|\boldsymbol{\omega}^\top \boldsymbol{x}_k + b\right| > 1$, the confidence is 1, otherwise is $\left|\boldsymbol{\omega}^\top \boldsymbol{x}_k + b\right|$. We therefore can express the classification confidence of a SVM classifier for the sample $\boldsymbol{x}_k$ as

$$g(\boldsymbol{x}_k) = \min\left(1, \left|\boldsymbol{\omega}^\top \boldsymbol{x}_k + b\right|\right). \tag{5}$$

We denote the i-th base-SVM in the layer l as $svm(l, i)$ and its decision function and confidence as $f^{(l,i)}(\cdot)$ and $g^{(l,i)}(\cdot)$, respectively. For the base-SVM $svm(l, i)$, we define a *confidence weighted output* $y^{(l,i)}$ as

$$y^{(l,i)} = f^{(l,i)}(\boldsymbol{x}) \cdot g^{(l,i)}(\boldsymbol{x}). \tag{6}$$

In the layer $l+1$, SVM-DSN concatenates the confidence weighted outputs of all base-SVMs in the previous layers and raw inputs as

$$\boldsymbol{x}^{(l+1)} = \left(y^{(l,1)}, \ldots, y^{(l,i)}, \ldots, y^{(l-1,1)}, \ldots, y^{(l-1,i)},\right.$$
$$\left.\ldots, y^{(1,1)}, \ldots, y^{(1,i)}, \ldots, x^{(1,1)}, \ldots, x^{(1,i)}\right)^{\top}. \tag{7}$$

The base-SVMs in the layer $l+1$ use $\boldsymbol{x}^{(l+1)}$ as the input to generate their confidence weighted outputs $y^{(l+1,i)}$. In this way, base-SVMs are stacked and connected by layer.

Model Training

Block Training

Similar to the standard deep stacking network, the training of the SVM-DSN model also contains a block training step and a fine-tuning step.

In the block training step, the base-SVMs in a DSN block are trained as regular SVM classifiers. Given a set of training samples $T = \{(\boldsymbol{x}_k, y_k)|k = 1, \ldots, K\}$, where y_k is the ground-truth label of $\boldsymbol{x}_k$, the objective function of a base-SVM group with N classification is defined as

$$\mathcal{J} = \frac{1}{2} \|\boldsymbol{\Omega}\|_F^2$$
$$+ C \sum_{k=1}^{K} \sum_{i=1}^{N} \ell_{hinge} \left(y_k^{(i)}(\boldsymbol{\omega}^{(i)\top}\boldsymbol{x}_k + b^{(i)})\right), \tag{8}$$

where $\boldsymbol{\Omega} = \left(\boldsymbol{\omega}^{(1)\top}, \ldots, \boldsymbol{\omega}^{(N)\top}\right)$, and $y_k^{(i)} = 1$ if $y_k = i$ and -1 otherwise. The function $\ell_{hinge}(\cdot)$ is a hinge loss function defined as $\ell_{hinge}(z) = \max(0, 1-z)$. The parameter $\theta = \left\{(\boldsymbol{\omega}^{(i)}, b^{(i)})|\forall\, i\right\}$ is inferred as $\theta = \arg\min_\theta \mathcal{J}(\theta)$.

In order to increase the diversity of base-SVM groups in a block, we adopt a bootstrap aggregating method in the block training. For a block with M base-SVM groups, we re-sample the training data as M sets using the bootstrap method (Efron and Tibshirani 1994). Each base-SVM group is trained using one re-sampled data set.

Fine Tuning

The traditional DSN model is based on neural networks and uses the BP algorithm in the fine-tuning step. For the SVM-DSN model, we introduce SVM training into the BP algorithm framework, and propose a BP-like Layered Tuning (BLT) algorithm to fine-tune the model parameters.

Algorithm 1 gives the pseudocodes of BLT. In general, BLT iteratively optimizes the base-SVMs from the output layer to the input layer. In each iteration, BLT optimizes $svm(l, i)$ by firstly generating a set of *virtual* training samples $T^{(l,i)} = \{(\boldsymbol{x}_k^{(l)}, \tilde{y}_k^{(l,i)})|k = 1, \ldots, K\}$, and then trains a new $svm(l, i)$ on $T^{(l,i)}$.

According to Eq. (6) and Eq. (7), it is easy to have $\boldsymbol{x}_k^{(l)} = (y_k^{(l-1,1)}, y_k^{(l-1,2)}, \cdots, y_k^{(l-1,i)}, \cdots)^{\top}$. However, the calculation of the *virtual label* $\tilde{y}_k^{(l,i)}$ is not that straightforward.

Algorithm 1 BP-like Layered Tuning Algorithm

1: *Initialization*: Initializing $\boldsymbol{\omega}^{(l,i)}, b^{(l,i)}$ for all $svm(l, i)$ as random values.
2: **repeat**
3:   Select a batch of training samples $T = \{(\boldsymbol{x}_k, y_k)|k = 1, \ldots, K\}$.
4:   **for** $l = L, L-1, \ldots, 2, 1$ **do**
5:     **for** $i = 1, 2, \ldots$ **do**
6:       Use Eq. (6), Eq. (7), and Eq. (9) to calculate $T^{(l,i)} = \{(\boldsymbol{x}_k^{(l)}, \tilde{y}_k^{(l,i)})|k = 1, \ldots, K\}$.
7:       Use $T^{(l,i)}$ to train $svm(l, i)$ as Eq. (11).
8:     **end for**
9:   **end for**
10: **until** The algorithm converges.

Specifically, BLT adopts a gradient descent method to calculate $\tilde{y}_k^{(l,i)}$ as

$$\tilde{y}_k^{(l,i)} = \sigma \left(y_k^{(l,i)} - \eta \left.\frac{\partial \mathcal{J}^{(o)}}{\partial y^{(l,i)}}\right|_{y^{(l,i)} = y_k^{(l,i)}}\right), \tag{9}$$

where $\mathcal{J}^{(o)}$ is the objective function of the output layer, $y_k^{(l,i)}$ is the output of $\boldsymbol{x}_k^{(l)}$ in the previous iteration, η is the learning rate, and $\sigma(\cdot)$ is a shaping function defined as

$$\sigma(z) = \begin{cases} 1, & z > 1 \\ z, & |z| \leq 1 \\ -1, & < -1 \end{cases}. \tag{10}$$

Note that since the term $-\eta \partial \mathcal{J}^{(o)}/\partial y^{(l,i)}$ in Eq. (9) is a negative gradient direction of $\mathcal{J}^{(o)}$, tuning the output $y^{(l,i)}$ to the virtual label $\tilde{y}^{(l,i)}$ can reduce the value of the objective function $\mathcal{J}^{(o)}$ in the output layer. Therefore, it could be expected that BLT can lower the overall model prediction error iteratively by training base-SVMs on virtual training sets in each iteration.

Given the training set $T^{(l,i)} = \{(\boldsymbol{x}_k^{(l)}, \tilde{y}_k^{(l,i)})|k = 1, \ldots K\}$, the objective function of training $svm(l, i)$ is defined as

$$\min \mathcal{J}^{(l,i)} = \frac{1}{2} \left\|\boldsymbol{\omega}^{(l,i)}\right\|^2$$
$$+ \underbrace{C_1 \sum_{k \in \Theta} \ell_{hinge} \left(\tilde{y}_k^{(l,i)}(\boldsymbol{\omega}^{(l,i)\top}\boldsymbol{x}_k^{(l)} + b^{(l,i)})\right)}_{\text{The SVM Loss}} \tag{11}$$
$$+ \underbrace{C_2 \sum_{k \notin \Theta} \ell_\epsilon \left(\boldsymbol{\omega}^{(l,i)\top}\boldsymbol{x}_k^{(l)} + b^{(l,i)} - \tilde{y}_k^{(l,i)}\right)}_{\text{The SVR Loss}},$$

where Θ is the index set of the virtual labels $\left|\tilde{y}_k^{(l,i)}\right| = 1$, and the function $\ell_\epsilon(\cdot)$ is an ϵ-insensitive loss function in the form of $\ell_\epsilon(z) = \max(|z| - \epsilon, 0)$.

Note that the objective function in Eq. (11) contains two types of loss functions so as to adapt to the different conditions of $\tilde{y}_k^{(l,i)}$. When $\tilde{y}_k^{(l,i)} \in \{-1, 1\}$, *i.e.*, the virtual labels

are binary, BLT trains $svm(l, i)$ as the standard SVM classifier and thus uses the hinge loss function to measure errors. When $\tilde{y}_k^{(l,i)} \in (-1, 1)$, the objective function adopts a Support Vector Regression loss term ℓ_ϵ for this condition. In the Appendix, we prove that the problem defined in Eq. (11) is a quadratic convex optimization problem. The training of $svm(l, i)$ can thus reach an optimal solution by using various quadratic programming methods such as sequential minimal optimization and gradient descents.

We finally turn to the small problem unsolved — how to calculate the partial derivative $\partial \mathcal{J}^{(o)} / \partial y^{(l,i)}$ in Eq. (9). Based on the chain rule, the partial derivative can be recursively calculated as

$$
\begin{aligned}
\frac{\partial \mathcal{J}}{\partial y^{(l,i)}} &= \sum_{m=l+1}^{L} \sum_j \frac{\partial \mathcal{J}}{\partial y^{(m,j)}} \frac{\mathrm{d}y^{(m,j)}}{\mathrm{d}z^{(m,j)}} \frac{\partial z^{(m,j)}}{\partial y^{(l,i)}} \\
&= \sum_{m=l+1}^{L} \sum_j \frac{\partial \mathcal{J}}{\partial y^{(m,j)}} y' \left(z^{(m,j)} \right) \omega_i^{(m,j)},
\end{aligned}
\tag{12}
$$

where $\omega_i^{(m,j)}$ is the connection weight of $y^{(m,j)}$ in $svm(m, j)$, and $z^{(m,j)} = \boldsymbol{\omega}^{(m,j)\top} \boldsymbol{x}^{(m-1)} + b^{(m,j)}$. The term $y'(z)$ is the derivative of the function in Eq. (6), which is in the form of

$$
y'(z) = \begin{cases} 0, & |z| > 1 \\ 1, & |z| \le 1 \end{cases}.
\tag{13}
$$

The principle of this chain derivation is similar to the error back-propagation of the neural network training. The difference lies in that the BP algorithm calculates the derivative for each neuron connecting weight but BLT for each base-SVM output. That is why we name our algorithm as *BP-like* Layered Tuning.

Model Properties

Connection to Neural Networks

The SVM-DSN model has close relations with neural networks. If we view the base-SVM output function defined in Eq. (6) as a neuron, the SVM-DSN model can be regarded as a type of neural networks. Specifically, we can rewrite the function in Eq. (6) as a neuron form as follows:

$$
y^{(l,i)} = \sigma \left(\boldsymbol{\omega}^{(l,i)\top} \boldsymbol{x}^{(l)} + b^{(l,i)} \right),
\tag{14}
$$

where the shaping function $\sigma(\cdot)$ works as an activate function, with the output $\sigma(z) \in \{1, -1\}$ if $|z| \ge 1$, and $\sigma(z) = z$ if $|z| < 1$. As proved in (Hornik 1991), a multilayer feedforward neural network with arbitrary bounded and non-constant activation function has an universal approximation capability. As a consequence, we could expect that the proposed SVM-DSN model also has the universal approximation capability in theory.

Nevertheless, the difference between the SVM-DSN model and neural networks is still significant. Indeed, we have proven in the Appendix that the base-SVMs in our SVM-DSN model have the following property: Given a set of virtual training samples $\{(\boldsymbol{x}_k^{(l)}, \tilde{y}_k^{(l,i)}) | k = 1, \ldots, K\}$ for

$svm(l, i)$, to minimize the loss function defined in Eq. (11) is a convex optimization problem. Moreover, because the base-SVMs in the same block are mutually independent, the optimization of the whole block is a convex problem too. This implies that, in each iteration of the BLT algorithm, all blocks can converge to an optimal solution. In other words, SVM-DSN ensures that all blocks and their base-SVMs "do their own best" to minimize their own objective functions in each iteration, which however is not the case for neural networks and MLP based deep stacking networks. It is also worth noting that this "do their own best" property is compatible with the decrease of the overall prediction error measured by the global objective function $\mathcal{J}^{(o)}$.

An important advantage empowered by the "do their own best" property is the *anti-saturation* feature of SVM-DSN. In neural network models, the BP algorithm updates the parameter ω of a neuron as $\omega \leftarrow \eta \partial \mathcal{J} / \partial \omega$. Hence, the partial derivative for the i-th ω in the j-th neuron at the layer l is calculated as

$$
\begin{aligned}
\frac{\partial J}{\partial \omega_i^{(l,j)}} &= \frac{\partial J}{\partial y^{(l,j)}} \frac{\mathrm{d}y^{(l,j)}}{\mathrm{d}z^{(l,j)}} \frac{\partial z^{(l,j)}}{\partial \omega_i^{(l,j)}} \\
&= \frac{\partial J}{\partial y^{(l,j)}} \cdot y' \left(z^{(l,j)} \right) \cdot y^{(l-1,i)},
\end{aligned}
\tag{15}
$$

where $y' \left(z^{(l,j)} \right)$ is a derivative of the activation function. For the sigmoid activation function, if $|z|$ is very large then y' becomes very small, and $\partial J / \partial \omega \to 0$. In this condition, the BP algorithm cannot update ω any more even if there is still much room for the optimization of ω. This phenomenon is called the "neuron saturation" in neural network training. For the ReLU activation function, similar condition appears when $z < 0$, where $y' = 0$ and $\partial J / \partial \omega = 0$. The neuron will die when a ReLU neuron fall into this condition.

In the BLT algorithm of SVM-DSN model, the update of a base-SVM is guided by $\partial \mathcal{J} / \partial y$, with the details given in Eq. (12). From Eq. (12), we can see that unless all base-SVMs in an upper layer are saturated, *i.e.*, $y'(z^{(m,j)}) = 0$ for all $z^{(m,j)}$, the base-SVMs in the layer l would not fall into the saturation state. Therefore, we could expect that the saturation risk of a base-SVM in SVM-DSN tends to be much lower than a neuron in neural networks.

Interpretation

In SVM classifiers, the support vector samples could provide some interpretation information about what a classifier learned from data set. The SVM-DSN inherit this interpretation property of SVM. In the base-SVM output function defined in Eq. (6), the confidence function $g(\boldsymbol{x})$ indicates whether a sample is clearly classified by the hyperplane of a base-SVM. For a sample to be classified, we can calculate the average confidence of the sample in all base-SVM groups in a block. The average confidence indicates whether the feature extracted by the block and previous layers offer enough representations to identify label of the sample. Because the samples with low confidence are near to the hyperplane, we can use the low confidence samples to form a "classifying plane" map in each blocks. Comparing the

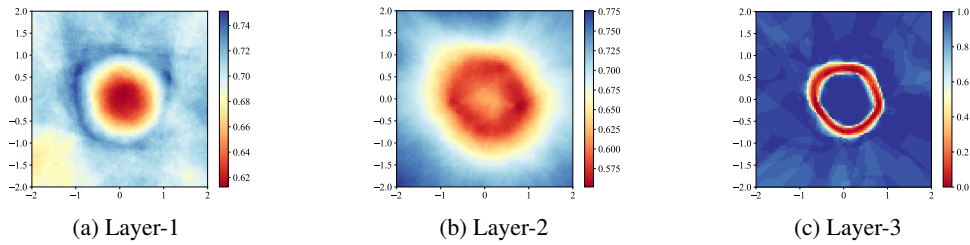(a) Layer-1   (b) Layer-2   (c) Layer-3

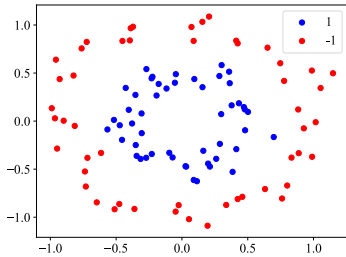Figure 2: The classifying plane maps of each layer in the SVM-DSN.

Figure 3: The Circle Data

"classifying plane" maps layer by layer, we could partly understand the feature extracting process of the stacked block in a SVM-DSN model.

We here give a show case to explain the interpretation property of SVM-DSN in a direct way. In this case, we generate a circle data set containing samples of two classes, as shown in Fig. 3. The positive samples are generated from a circle with radius of 0.5 plus a Gaussian noise with variance of 0.1. The negative samples are generated from a circle with radius of 1 plus the same Gaussian noise. We use the circle data set to train a 3-layer SVM-DSN model, where the middle layers contain 40 and 60 base-SVM groups, respectively. Because this experiment is a binary classification, a base-SVM group only contains one base-SVM.

In the experiment, we traverse the feature space from the coordinate point $(-2, -2)$ to $(2, 2)$. For each coordinate point, we calculate the average confidence of all base-SVMs in each layer. Figs. 2a - 2c plot the confidence distribution maps in different layers. The samples with low confidence are thus near to the SVM classification hyperplane, so the low confidence areas in red form the "classifying plane" of a layer.

It is obvious that there is a clear "classifying plane" generation process from Figs. 2a to 2c. In the layer 1, the low confidence values concentrate in the center of the map. In the layer 2, the low confidence values have a vague shape as a circle. In the layer 3, the low confidence values distribute as a circle shape that clearly divides the feature space into two parts. This process demonstrates how an SVM-DSN model extracts the data representations of the feature space layer by layer.

Parallelization

The deep stacking network is proposed for parallel parameter learning. For a DSN with L blocks, given a group of training samples, the DSN can resample the data set as L batches. A DSN block only uses one batch to update its parameter. In this way, the training of DSN parameters could be deployed over L processing units. The parallelization of DSN training is in the block level.

This parallel parameter learning property could be further extended by using Parallel Support Vector Machines (Graf et al. 2005). Because in a SVM classifier, only the support vector samples are crucial, we could divide a training set as several M sub-sets, and use M virtual SVM classifiers to select support vector candidates from each sub-set. Finally, we use the support vector candidates of all sub-sets to train the final model. In this way, the training of a SVM-DSN block can be deployed over M processing units, and the all SVM-DSN model can be deployed over $L \times M$ processors. That is to say the training of base-SVMs in a block is also parallelizable in SVM-DSN. The parallelization of SVM-DSN training is in the base-SVM level. The parallel degree of whole model is greatly improved. As reported in Ref (Graf et al. 2005), the speed-up for a 5-layer Cascade SVM (16 parallel SVMs) to a single SVM is about 10 times for each pass and 5 times for fully converged.

Experiments

Image Classification Performance

We first test the performance of SVM-DSN on the MNIST image classification database (LeCun et al. 1998). The MNIST database contains 60,000 handwritten digits images in the size of 28×28 for training and validation, and 10,000 images for testing. The images are classified as ten classes according to the digits written on them. The SVM-DSN model used in the experiment consists of three layers – two middle layers and an output layer. Both of the block in the two middle layers contains 20 base-SVM groups, and each group contains 10 base-SVMs, *i.e.*, 200 base-SVMs one layer. We actually had tried neural networks with deeper layers, but the classification performance could not be improved significantly. The benchmark models includes: i) A 3-layer deep stacking network where the block in each middle layer contains 200 hidden neurons; ii) As analyzed in the Model Properties section, the SVM-DSN could be considered as a type of neural network. Therefore, we use the BP algorithm to train a same structure SVM-DSN as a benchmark; iii) The 3-layer neural network models with the same neuron connection structure as the 3-layer SVM-DSN. In these benchmarks, we use SVM output (Tang 2013), and

Table 1: MNIST Classification Performance

Models	Error Rate (%)
3-layer SVM-DSN	**1.49**
3-layer DSN	1.65
3-layer SVM-DSN, BP	1.62
3-layer NN, SVM output, sigmoid	1.74
3-layer NN, SVM output, tanh	1.59
3-layer NN, SVM output, ReLU	1.56
Homepage benchmark (LeCun et al. 1998)	1.53
Bagging of base-SVMs	5.41

Table 2: MNIST Classification with CNN Feature Extractor

Models	Error Rate (%)
CNN + SVM-DSN	**0.51**
CNN + DSN	0.60
CNN + SVM-DSN, BP	0.72
CNN + sigmoid activation	0.80
CNN + tanh activation	0.67
CNN + ReLU activation	0.58
Homepage benchmark (LeCun et al. 1998)	0.54
gcForest (Zhi-Hua Zhou 2017)	0.74

Table 3: IMDB Classification Performance

Models	Error Rate (%)
SVM-DSN	**10.51**
DSN	11.15
SVM-DSN, BP	11.42
Random Forest	14.68
XGBoost	14.77
AdaBoost	16.63
SVM (linear kernel)	12.43
Stacking	11.55
Bagging of base-SVMs	11.66
gcForest (Zhi-Hua Zhou 2017)	10.84

the different activate functions in neurons; iv) The best 3-layer NN benchmark listed in the homepage of MNIST – 3-layer NN, 500+300 hidden units, cross entropy loss, weight decay (LeCun et al. 1998); v) A bagging of 41 base-SVM groups, each group contains 10 base-SVMs.

In the SVM-DSN model fine-tuning, we have two hyper-parameters to set, *i.e.*, $C1$ and $C2$ of the base-SVM's objective function. The two hyper-parameters are used to balance structural risks and empirical risks in a base-SVM. Either too big or too small for the two hyper-parameters may lead model performance degenerate. Therefore, we use the trial and error method to set the hyper-parameters. The learning rate η is the other hyper-parameter, which could be dynamic setting using elegant algorithms such as Adam. In our experiment, we directly set the learning rate as a fix value $\eta = 0.0005$ to ensure the experiment fairness.

Table 1 gives the MNIST image classification results. The SVM-DSN model achieved the best performance compared with the other benchmarks, which verified the effectiveness of SVM-DSN. In the benchmarks, the 3-layer SVM-DSN+BP model has the same model structure with SVM-DSN but was trained by the BP algorithm. The results show that SVM-DSN has a better performance than the SVM-DSN+BP benchmark, which indicates that the base-SVMs "do their own best" feature of SVM-DSN is a positive feature for model performance. In fact, the idea of BLT fine-tuning could be extend to optimize the deep stacking networks with any derivable model as blocks, such as soft decision-making tree and linear discriminant analysis.

Feature Extractor Compatibility

Currently, the mainstream image classifiers usually adopt convolutional neural networks (CNN) as feature extractors. Table 2 demonstrates the MNIST classification performance of SVM-DSN with a CNN feature extractor. In the experiment, we connect a CNN feature extractor with a 3-layer SVM-DSN model. The structure of SVM-DSN is same as in Table 1. The CNN feature extractor contains 3 convolutional layers, and each layer consists of 24 flitters with the 5×5 receptive field. In the CNN and SVM-DSN mixture model, we first pre-trained the CNN part using BP, and then uses the feature extracted by CNN as input of the SVM-DSN part to train the blocks. In the fine-tuning step, the CNN part is fine tuned by BP and the SVM-DSN part is tuned by BLT. The benchmark models include: i) The same CNN feature extractor connected with a 3-layer DSN, where

the 3-layer DSN has the same structure with the 3-layer SVM-DSN model; ii) The CNN+SVM-DSN model trained by the BP algorithm; iii) The neural networks consist of 3 CNN layers and 3-layer neural network with different activate functions, where the structures of the CNN and the neural network are same as the CNN + SVM-DSN model; iv) The trainable CNN feature extractor + SVMs with affine distortions, which is the best benchmark with the similar models scale listed in the MNIST homepage; v) The gcForest with convolutional kernels (Zhi-Hua Zhou 2017). As shown in Table 2, the CNN + SVM-DSN model achieved the best performance, which verified the effectiveness of our model again. What's more, this experiment demonstrates that the SVM-DSN model is completely compatible to the neural network framework. The other types of networks, such as RNN and LSTM, could also be used as feature extractors of SVM-DSN to adapt diversified application scenarios.

Comparison with Ensemble Models

In this section, we compare the performance of SVM-DSN with several classical ensemble models. Because on the MNIST data set, the performance of ensemble models are usually not very well. For a fair comparison, we use the IMDB sentiment classification data set (Maas et al. 2011) in our experiment. Many tree based ensemble methods achieved good performance on this data set (Zhi-Hua Zhou 2017). The IMDB dataset contains 25,000 movie reviews for training and 25,000 for testing. The movie reviews are represented by tf-idf features and labeled as positives and negatives. The SVM-DSN model used in this experiment consists of 4 middle layers, and the number of

base-SVMs in the middle layer are 1024-1024-512-256. The benchmark models include Random Forest, XGBoost, AdaBoost and SVM (linear kernel). The four benchmarks are also stacked as a stacking benchmark (Perlich and Świrszcz 2011). A bagging of base-SVMs and the grForest (Zhi-Hua Zhou 2017) are also included as competitor. A 4-layer DSN is used as a benchmark, where the number of hidden neurons in the middle layer blocks are same as the number of base-SVMs in the SVM-DSN model. The SVM-DSN model trained by the BP algorithm is also used as the benchmark.

As shown in Table 3, the SVM-DSN model achieved the best performance again. Especially, the performance of SVM-DSN is better than the stacking benchmark, which indicates that holistic optimized multi-layer stacking of linear base-learners can defeat the traditional two-layer stacking of strong base-learners. In the experiment, the performance of the BLT algorithm is yet better than the BP algorithm. The "do their best" feature of base-SVM in BLT is still effective in text sentiment classification.

Related Works

This work has close relations with SVM, deep learning, and stacking. The support vector machine was first proposed by Vapnik in (Vapnik 1998). Multi-layer structures in SVM were usually used as speedup solutions. In cascade SVM (Graf et al. 2005), a multi-layer cascade SVM model structure was used to select support vectors in a parallel way. In the literature (Collobert, Bengio, and Bengio 2002), a parallel mixture stacking structure was proposed to speed up SVM training in the very large scale problems. Before our work, some studies proposed to use SVM to replace the output layer of a neural network (Wiering et al. 2013; Tang 2013).

In recent years, neural network based deep models has achieved great success in various applications (Hinton and Salakhutdinov 2006). The gcFroest model (Zhi-Hua Zhou 2017) was proposed to use the forest based deep model as an alternative to deep neural networks. The PCANet builds a deep model using unsupervised convolutional principal component analysis (Chan et al. 2015). LDANet is a supervised extension of PCANet, which uses linear discriminant analysis (LDA) to replace the PCA parts of PCANet (Chan et al. 2015). Deep Fisher Networks build deep network through stacking Fisher vector encoding as multi-layers (Simonyan, Vedaldi, and Zisserman 2013).

The DSN framework adopted in this work is a scalable deep architecture amenable to parallel parameter training, which has been adopted in various applications, such as information retrieval (Deng, He, and Gao 2013), image classification (Li, Chang, and Yang 2015), and speech pattern classification (Deng and Yu 2011). T-DSN uses tensor blocks to incorporate higher order statistics of the hidden binary features (Hutchinson, Deng, and Yu 2013). The CCNN model extends the DSN framework using convolutional neural networks (Zhang, Liang, and Wainwright 2016). To the best of our knowledge, there are very few works introduce the advantages of SVM into the DSN framework.

Stacking was introduced by Wolpert in (Wolpert 1992) as a scheme of combining multiple generalizers. In many real-world applications, the stacking methods were used to integrate strong base-learners as an ensemble model to improve performance (Jahrer, Töscher, and Legenstein 2010). In the literature, most of stacking works focused on designing elegant meta-learners and create better base-learners, such as using class probabilities in stacking (Ting and Witten 1999), using a weighted average to combine stacked regression (Rooney and Patterson 2007), training base-learners using cross-validations (Perlich and Świrszcz 2011), and applying ant colony optimization to configure base-learners (Chen, Wong, and Li 2014). To the best of our knowledge, there are very few works to study how to optimize multi-layer stacked base-learners as a whole.

Conclusion

In this paper, we proposed an SVM-DSN model where linear base-SVMs are stacked and trained in a deep stacking network way. In the SVM-DSN model, the good mathematical property of SVMs and the flexible model structure of deep stacking networks are nicely combined in a same framework. The SVM-DSN model has many advantage properties including holistic and local optimization, parallelization and interpretation. The experimental results demonstrated the superiority of the SVM-DSN model to some benchmark methods.

Acknowledgments

Appendix

Property: *Given a set of virtual samples $T^{(l,i)} = \{(\boldsymbol{x}_k^{(l)}, \tilde{y}_k^{(l,i)})|k = 1, \ldots, K\}$ for $svm(l,i)$, to minimize the loss function defined in Eq.* (11) *is a convex optimization problem.*

Proof. For the sake of simplicity, we omit the superscripts (l) of $\boldsymbol{x}_k^{(l)}$ and $\tilde{y}_k^{(l)}$ in our proof. We define a constrained optimization problem in the form of

$$\min_{\boldsymbol{\omega},b,\xi_k,\hat{\xi}_k,\zeta_k} \frac{1}{2}\|\boldsymbol{\omega}\|^2 + C_1 \sum_{k \notin \Theta} \zeta_k + C_2 \sum_{k \in \Theta} \left(\xi_k + \hat{\xi}_k\right)$$

$$s.t. \quad 1 - \tilde{y}_k\left(\boldsymbol{\omega}^\top \boldsymbol{x}_k + b\right) \le \zeta_k, \tag{1}$$

$$\zeta_k \ge 0, \ k \in \Theta;$$

$$\left(\boldsymbol{\omega}^\top \boldsymbol{x}_k + b\right) - \tilde{y}_k \le \epsilon + \xi_k, \tag{2}$$

$$\tilde{y}_k - \left(\boldsymbol{\omega}^\top \boldsymbol{x}_k + b\right) \le \epsilon + \hat{\xi}_k, \tag{3}$$

$$\xi_k \ge 0, \ \hat{\xi}_k \ge 0, \ k \notin \Theta. \tag{16}$$

We can see the constrained optimization problem Eq. (16) is in a quadratic programming form as

$$\min_{\mathbf{a}} \quad \frac{1}{2}\mathbf{a}^\top \mathbf{U}\mathbf{a} + \mathbf{c}^\top \mathbf{a}$$
$$s.t. \quad \mathbf{Q}\mathbf{a} \le \mathbf{p}, \tag{17}$$

where $\mathbf{a} = (\boldsymbol{\omega}, b, \boldsymbol{\xi}, \hat{\boldsymbol{\xi}}, \boldsymbol{\zeta})$, and $\mathbf{U}$ is a positive semi-definite diagonal matrix. Therefore, the constrained optimization problem is a quadratic convex optimization problem (Boyd and Vandenberghe 2004). It is easy to prove that the constrained optimization problem defined in Eq. (16) is equivalent to the unconstrained optimization problem defined in Eq. (11) (Zhang 2003). Therefore, the optimization problem of base-SVM is equivalent to the problem defined in Eq. (16). The optimization problem of base-SVM is a convex optimization problem.

References

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

Bengio, Y.; Courville, A.; and Vincent, P. 2013. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 35(8):1798–1828.

Boyd, S., and Vandenberghe, L. 2004. *Convex optimization*. Cambridge University.

Chan, T.-H.; Jia, K.; Gao, S.; Lu, J.; Zeng, Z.; and Ma, Y. 2015. PCANet: A simple deep learning baseline for image classification? *IEEE Transactions on Image Processing* 24(12):5017–5032.

Chen, Y.; Wong, M.-L.; and Li, H. 2014. Applying ant colony optimization to configuring stacking ensembles for data mining. *Expert Systems with Applications* 41(6):2688–2702.

Collobert, R.; Bengio, S.; and Bengio, Y. 2002. A parallel mixture of SVMs for very large scale problems. In *NIPS*, 633–640.

Deng, L., and Yu, D. 2011. Deep convex net: A scalable architecture for speech pattern classification. In *INTERSPEECH*, 2285–2288.

Deng, L.; He, X.; and Gao, J. 2013. Deep stacking networks for information retrieval. In *ICASSP*, 3153–3157. IEEE.

Efron, B., and Tibshirani, R. J. 1994. *An introduction to the bootstrap*. CRC press.

Graf, H. P.; Cosatto, E.; Bottou, L.; Dourdanovic, I.; and Vapnik, V. 2005. Parallel support vector machines: The cascade SVM. In *NIPS*, 521–528.

Hinton, G. E., and Salakhutdinov, R. R. 2006. Reducing the dimensionality of data with neural networks. *Science* 313(5786):504–507.

Hinton, G. 2011. Deep belief nets. In *Encyclopedia of Machine Learning*. Springer. 267–269.

Hornik, K. 1991. Approximation capabilities of multilayer feedforward networks. *Neural networks* 4(2):251–257.

Hutchinson, B.; Deng, L.; and Yu, D. 2013. Tensor deep stacking networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 35(8):1944–1957.

Jahrer, M.; Töscher, A.; and Legenstein, R. 2010. Combining predictions for accurate recommender systems. In *SIGKDD*, 693–702. ACM.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. In *NIPS*, 1097–1105.

LeCun, Y.; Bottou, L.; Bengio, Y.; and Haffner, P. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE* 86(11):2278–2324.

Li, J.; Chang, H.; and Yang, J. 2015. Sparse deep stacking network for image classification. In *AAAI*, 3804–3810.

Maas, A. L.; Daly, R. E.; Pham, P. T.; Huang, D.; Ng, A. Y.; and Potts, C. 2011. Learning word vectors for sentiment analysis. In *ACL*, 142–150.

Medsker, L., and Jain, L. 2001. Recurrent neural networks. *Design and Applications*.

Perlich, C., and Świrszcz, G. 2011. On cross-validation and stacking: Building seemingly predictive models on random data. *ACM SIGKDD Explorations Newsletter* 12(2):11–15.

Rooney, N., and Patterson, D. 2007. A weighted combination of stacking and dynamic integration. *Pattern Recognition* 40(4):1385–1388.

Simonyan, K.; Vedaldi, A.; and Zisserman, A. 2013. Deep fisher networks for large-scale image classification. In *NIPS*, 163–171.

Tang, Y. 2013. Deep learning using linear support vector machines. *arXiv preprint arXiv:1306.0239*.

Ting, K. M., and Witten, I. H. 1999. Issues in stacked generalization. *Journal of Artificial Intelligence Research* 10:271–289.

Vapnik, V. 1998. *Statistical learning theory. 1998*. Wiley, New York.

Vincent, P.; Larochelle, H.; Lajoie, I.; Bengio, Y.; and Manzagol, P.-A. 2010. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research* 11:3371–3408.

Wiering, M.; Van der Ree, M.; Embrechts, M.; Stollenga, M.; Meijster, A.; Nolte, A.; and Schomaker, L. 2013. The neural support vector machine. In *BNAIC*.

Wolpert, D. H. 1992. Stacked generalization. *Neural networks* 5(2):241–259.

Zhang, Y.; Liang, P.; and Wainwright, M. J. 2016. Convexified convolutional neural networks. *arXiv preprint arXiv:1609.01000*.

Zhang, T. 2003. Statistical behavior and consistency of classification methods based on convex risk minimization. *Annals of Statistics* 32(1):56–134.

Zhi-Hua Zhou, J. F. 2017. Deep Forest: Towards an alternative to deep neural networks. In *IJCAI*, 3553–3559.