SCALABLE STACKING AND LEARNING FOR BUILDING DEEP ARCHITECTURES

Li Deng, Dong Yu, and John Platt

Microsoft Research, Redmond, USA

{deng,dongyu,jplatt}@microsoft.com

ABSTRACT

Deep Neural Networks (DNNs) have shown remarkable success in pattern recognition tasks. However, parallelizing DNN training across computers has been difficult. We present the Deep Stacking Network (DSN), which overcomes the problem of parallelizing learning algorithms for deep architectures. The DSN provides a method of stacking simple processing modules in buiding deep architectures, with a convex learning problem in each module. Additional fine tuning further improves the DSN, while introducing minor non-convexity. Full learning in the DSN is batch-mode, making it amenable to parallel training over many machines and thus be scalable over the potentially huge size of the training data. Experimental results on both the MNIST (image) and TIMIT (speech) classification tasks demonstrate that the DSN learning algorithm developed in this work is not only parallelizable in implementation but it also attains higher classification accuracy than the DNN.

Index Terms— deep learning, stacking, DNN, DSN, convexity

1. INTRODUCTION

Deep Neural Networks (DNNs) are extremely powerful in performing machine learning tasks including image classification, speech recognition, and speech coding [1–3]. However, training DNNs is computationally difficult. In particular, fine tuning DNNs requires stochastic gradient descent, which is unusually difficult to parallelize across machines. This lack of parallelism makes learning at large scale practically impossible. For example, it has been possible to use one single, very powerful GPU machine to train DNN-based speech recognizers with dozens to a few hundreds of hours of speech training data with extraordinary results; e.g., [2, 3]. It is very difficult, however, to scale up this success with thousands or more hours of training data. The goal of the research described in this paper is a scalable method for buidling deep classification architectures. We present the Deep Convex Network (DSN) whose architecture can be created by a process called stacking and whose learning can be effectively parallelized at every step. Since the basic learning algorithm of DSN is convex, it can also be called deep convex network.

The main theoretical motivation of this work is the desire to learn complex functions from large data sets with parallel learning algorithms in order to solve large-scale, hard problems, which has been part of machine learning for many years. Many researchers think that tractably solving this problem can bring us closer to the dream of Artificial Intelligence [4]. Learning complex functions effectively without over-fitting is a challenging task, however.

A popular method for effective learning of complex functions is to compose simple functions first and then "stack" them [5]. In stacking, simple functions are composed in a chain, with the output of one feeding the input of the next. The job of each of the functions is to estimate the same target. An early example of a stacking-like architecture was Cascade Correlation [6], although it was prone to overfitting. Wolpert [5] used a variant of cross-validation to minimize overfitting as the functions were stacked. Cohen and de Carvalho [7] successfully stacked CRFs to emulate long-range inference in NLP. Similarly stacked, deep-CRF was used in speech processing [8]. In this paper, we present an architecture that is similar to stacking, in that it is built up in stages, each stage trying to estimate the same targets.

There has been recent excitement around using unsupervised Deep Belief Network [1] and Deep Boltzmann Machine [9]. The latter held the previous record for undistorted MNIST with a non-convolutional network. These architectures directly inspired the architecture presented in this paper, and we present our stronger results in both MNIST image classification and TIMIT speech classification. Some preliminary work and results for speech classification was presented in [10]. In this paper, we provide new ways of stacking up the DSN, emphasizing its stacking property instead of convexity one. In particular, we describe how the weight matrices are initialized while stacking up new modules. We also report more comprehensive evaluation results for image and speech classification tasks of MNIST and TIMIT.

2. DSN: AN ARCHITECTURAL OVERVIEW

In this section, we provide an overview of the DSN architecture. We use a specific example of the DSN, shown in Fig. 1 (used in our MNSIT experiment), to describe DSN's modular and layered structure. A DSN includes a variable number of modules, with three modules shown in Fig. 1 although most experiments reported in this paper involve much deeper DSNs. Each module is a specialized neural network, which is grouped with a distinct color in Fig. 1, consisting of a single hidden layer and two trainable sets of weights.

Different modules of the DSN are constructed somewhat differently. The lowest module comprises the following three layers for information processing. First, there is a linear layer with a set of input units. They correspond to the raw input data in the vectored form. Let N input vectors in the full training data be $\mathbf{X} = [\mathbf{x}_1, ..., \mathbf{x}_i, ..., \mathbf{x}_N]$, with each vector $\mathbf{x}_i = [\mathbf{x}_{1i}, ..., \mathbf{x}_{ji}, ..., \mathbf{x}_{Di}]^T$. Then, the input units correspond to the elements of $\mathbf{x}_i$, with dimensionality D. Second, the non-linear layer consists of a set of sigmoidal hidden units. Denote by L the number of hidden units and define $\mathbf{h}_i = \sigma(\mathbf{W}_1^T \mathbf{x}_i)$ as the hidden layer's output, where $\sigma(.)$ is the sigmoid function and $\mathbf{W}_1$ is an $D \times L$ trainable weight matrix, at the bottom module, acting on the input layer. Note the bias vector is implicitly represented in the above formulation when $\mathbf{x}_i$ is augmented with all ones. Third, the output layer consists of a set of C linear output units with their values computed by $\mathbf{y}_i = \mathbf{U}_1^T \mathbf{h}_i$, where $\mathbf{U}_1$ is an $L \times C$ trainable weight matrix associated with the upper layer of the bottom module. Again, we augment $\mathbf{h}_i$ with a vector consisting of all one's. The output units represent the targets of classification (or regression).

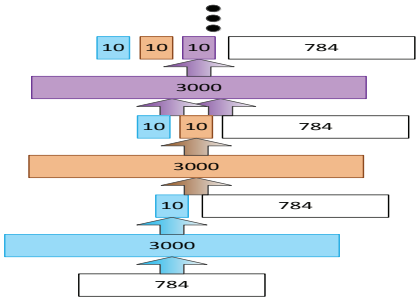

Fig. 1. Illustration of the basic architecture of DSN

Above the bottom one, all other modules of a DSN, which are stacking up one above another, are constructed in a similar way to the above but with a key exception in the input layer. Rather than making the input units take the raw data vector, we concatenate the raw data vector with the output layer(s) in the lower module(s). Such an augmented vector serves as the "effective input" to the immediately higher module. The dimensionality, D_m, of the augmented input vector is a function of the module number, m, counted from bottom up according to

$$D_m = D + C(m-1), \quad m = 1, 2, \cdots, M \tag{1}$$

where $m = 1$ corresponds to the bottom module.

A closely related difference between the bottom module and the remaining modules concerns the augmented weight matrix $\mathbf{W}$. The weight matrix augmentation results from the augmentation of the input units. That is, the dimensionality of $\mathbf{W}$ changes from $D \times L$ to $D_m \times L$. Additional columns of the weight matrix corresponding to the new output units from the lower module(s) are initialized with random numbers, which are subject to optimization to be presented in Section 4.

3. THE BASIC LEARNING ALGORITHM

3.1. Convex Optimization for Weight Matrix U Given W

The convex optimization technique covered here applies to all modules of a DSN. Implementation of the technique differs for distinct modules mainly in ways of setting the lower-layer weight matrices $\mathbf{W}$ in each module, which varies its dimensionality across modules according to Eq. 1, before applying the learning technique presented below.

Here we assume the supervised learning setting. Both training data $\mathbf{X} = [\mathbf{x}_1, ..., \mathbf{x}_i, ..., \mathbf{x}_N]$ and the corresponding labeled target vectors $\mathbf{T} = [\mathbf{t}_1, ..., \mathbf{t}_i, ..., \mathbf{t}_N]$, where each target $\mathbf{t}_i = [\mathbf{t}_{1i}, ..., \mathbf{t}_{ji}, ..., \mathbf{t}_{Ci}]^T$, are available. We use the loss function of mean square error to learn weight matrices $\mathbf{U}$ assuming $\mathbf{W}$ is given. That is, we aim to minimize: $E = Tr[(\mathbf{Y} - \mathbf{T})(\mathbf{Y} - \mathbf{T})^T]$, where $\mathbf{Y} = [\mathbf{y}_1, ..., \mathbf{y}_i, ..., \mathbf{y}_N]$. Importantly, if weight matrix $\mathbf{W}$ is determined already (e.g., via judicious initialization), then the hidden layer values $\mathbf{H} = [\mathbf{h}_1, ..., \mathbf{h}_i, ..., \mathbf{h}_N]$ are also determined. Consequently, upper-layer weight matrix $\mathbf{U}$ in each module can be determined by setting the gradient

$$\frac{\partial E}{\partial \mathbf{U}} = 2\mathbf{H}(\mathbf{U}^T\mathbf{H} - \mathbf{T})^T \tag{2}$$

to zero. This is a well established convex optimization problem and has a straightforward closed-form solution, known as pseudo-inverse:

$$\mathbf{U} = (\mathbf{H}\mathbf{H}^T)^{-1}\mathbf{H}\mathbf{T}^T. \tag{3}$$

3.2. Setting Weight Matrices W across DSN Modules

In the basic algorithm, the weight matrices $\mathbf{W}$ across all DSN modules need to be set empirically before the application of Eq.3 for determining the weight matrices $\mathbf{W}$ module by module. For the bottom module, we have experimented two ways of setting $\mathbf{W}$. First, random numbers are generated with various distributions. The results are then used for setting $\mathbf{W}$. Second, restricted Boltzmann machines (RBM) are trained separately using contrastive divergence [1]. Then the trained RBM weights are used to set $\mathbf{W}$. For all non-bottom modules, we copy the same RBM for $\mathbf{W}$, which is appended by random numbers for the augmented columns.

4. THE FINE-TUNING ALGORITHM

In this section, we present a batch-mode, parallelizable fine tuning algorithm that improves upon the basic algorithm. In contrast to fine tuning for DNN of [1], our fine tuning adjusts the weight matrices $\mathbf{W}$ and $\mathbf{U}$ module by module, not involving any global fitting over the entire architecture and thus reducing over-fitting. The essence of the DSN fine tuning is to exploit the structural relationship between $\mathbf{W}$ and $\mathbf{U}$ in each module, as expressed in Eq.3, in computing the gradient of the loss function with respect to $\mathbf{W}$.

4.1. Gradient Computation Embedding Eq.3

We derived the close-form expression of gradient $\frac{\partial E}{\partial W}$ by considering the effect of $\mathbf{W}$ on $\mathbf{U}$ and, consequently, the effect on loss E:

$$\begin{aligned} \frac{\partial E}{\partial \mathbf{W}} &= \frac{\partial Tr[(\mathbf{U}^T\mathbf{H} - \mathbf{T})(\mathbf{U}^T\mathbf{H} - \mathbf{T})^T]}{\partial \mathbf{W}} \\ &= 2\mathbf{X}[\mathbf{H}^T \circ (1 - \mathbf{H})^T \circ [\mathbf{H}^\dagger(\mathbf{H}\mathbf{T}^T)(\mathbf{T}\mathbf{H}^\dagger) - \mathbf{T}^T(\mathbf{T}\mathbf{H}^\dagger)] \end{aligned} \tag{4}$$

where $\mathbf{H}^\dagger = \mathbf{H}^T(\mathbf{H}\mathbf{H}^T)^{-1}$ and $\circ$ denotes element-wise matrix multiplication. In deriving Eq.4, we used the fact that $\mathbf{H}\mathbf{H}^T$ is symmetric and so is $(\mathbf{H}\mathbf{H}^T)^{-1}$.

The batch-mode fine tuning algorithm then updates $\mathbf{W}$ using the gradient computed in Eq.4. Weight matrix $\mathbf{U}$ is subsequently updated using Eq.3 in a closed form with no iteration.

4.2. Initializing W over DSN Modules before Fine Tuning

For the bottom module of a DSN, $\mathbf{W}$ is initialized in the same way as in the basic algorithm described previously. For all higher modules, we have devised four different strategies of $\mathbf{W}$ initialization for iterative fine tuning. The sub-matrix of $\mathbf{W}$ corresponding to the output units from the lower modules is always initialized with random numbers. The remaining portion of $\mathbf{W}$ associated with the input data is initialized in the following four possible ways:

- Take the same $\mathbf{W}$ from the immediately lower module already adjusted via fine tuning.
- Take a copy of the RBM that initialized $\mathbf{W}$ at bottom module.
- Use random numbers, making the full $\mathbf{W}$ maximally random before fine tuning.
- Mix the above three choices with various weighting and with randomized order or otherwise.

It was found that with sufficient efforts put to adjust all other hyper-parameters, all four strategies above eventually gave similar classification accuracy. One interesting observation is that fully random initialization of $\mathbf{W}$ at non-bottom modules (the third strategy above) does not give worse accuracy but it takes many more modules and fine-tuning iterations than other strategies. Importantly, this

observation does not hold for the bottom module. As we will show in Section 5, the use of RBM to initialize the bottom module always gives much better accuracy than using random numbers.

5. EXPERIMENTS ON IMAGE CLASSIFICATION

5.1. Experimental setup and general results

We evaluated the DSN and the associated learning algorithms on the standard MNIST database of binary images of handwritten digits [11]. The task is to classify each 28x28 image into one of the 10 digits. The MNIST training set is composed of 60,000 examples from approximately 250 writers, part of which is used as the development set. The test set is composed of disjoint 10,000 examples. Fig. 1 showed the architecture of the DSN used in the experiments.

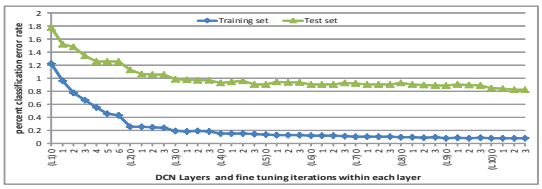

Fig. 2. MNIST test set percent classification error rate.

Fig. 2 gives results of percentage error rate as a function of the modules in the DSN of Figure 1 and of the iteration number of the fine tuning algorithm applied to each module. In the horizontal axis of Figure 2, "0" indicates the beginning of a new module, and "1", "2", "3"... signifies the epoch number in the running of the batch-mode fine tuning algorithm described in Section 4. Note that when the first few new modules are added, large error reductions are produced for both training and test sets. After 10 modules, while the training error rate is still in the downward trend, the test error rate stops further drop. The best testing error rate of 0.83% is obtained with 10 modules of DSN, the best result reported in the literature on undistorted MNIST without specifying convolutional structure.

In the experiment of Fig. 2, we use an RBM to initialize the weight matrix $\mathbf{W}$ in the lowest module of the DSN, which is then subject to fine tuning. When stacking up to the second module, the fine-tuned weight matrix is copied, mixed with a new weight sub-matrix initialized by random numbers. The sub-matrix corresponds to the output units from the bottom module. This combined weight matrix is then again subject to fine tuning in the second module, producing a new set of outputs in this new module. This process is continued until the top module of DSN. We also use a theoretically motivated scaling number on the raw data to train the RBM and use the same number to scale the data in training and testing the DSN.

5.2. Comparative results

In Fig. 3, we show results of using random samples from $[-1, 1]$ to initialize the weight matrix $\mathbf{W}$ at the DSN's bottom module. The basic algorithm is used for learning weight matrix $\mathbf{U}$ without fine tuning. As the numbers of DSN modules and hidden units increase, the error rate drops significantly. Similar trends hold when RBMs are used to initialize the weight matrix $\mathbf{W}$ and when fine tuning is applied, with the important difference that the effect of increasing DSN modules is much stronger than that of increasing the hidden size. In Table 1 we present error rate comparisons in the MNIST test set at the convergence of stacking DSN modules and fine tuning iterations. All experiments are with the hidden layer's size being fixed at 3000. The conclusions are: 1) RBM is a powerful technique for initializing weight matrices at the bottom module of a DSN, a similar

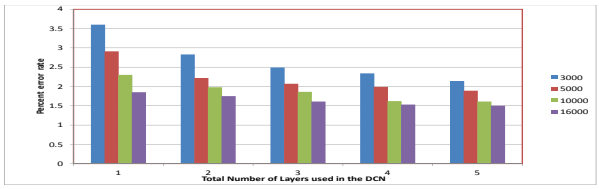

Fig. 3. Classification error rate (%) as a function of the number of DSN modules and hidden units. Uniformly distributed random numbers used to initialize $\mathbf{W}$; no fine tuning.

conclusion reached for DNN [1]; and 2) fine tuning contributes significantly to error reduction. On the latter, we note that the amount of error reduction for DSN fine tuning is less than that of DNN [1]. This is likely due to the fact that the basic algorithm of DSN already embeds discriminative learning while the pre-training step of DNN is purely generative.

In Table 2, we compare the error rates for two versions of DNN, the DSN, and the degenerated DSN when only one single module is used (i.e, no stacking). This comparison highlights the power of stacking up modules in DSN. Note the weight matrices $\mathbf{W}$ at the bottom modules of all DSNs in Table 2 are initialized with RBMs. And the second DNN in Table 2 is further trained with our best efforts after duplicating the results of [1], carried out internally.

Table 1. Comparative error rates: Effects of initialization and fine tuning

Weight Initialization	Fine Tuning	Error Rate (%)
Random (unit Gaussian)	No	2.15
Random (uniform)	No	2.02
Random (uniform)	Yes	1.70
RBM	No	0.95
RBM	Yes	0.83

Table 2. Comparative error rates: DNN, DSN, & degenerated DSN

Models	Error Rate (%)
DNN with fine tuning in [1]	1.20
DNN with further fine tuning	1.06
DSN (10 modules, with fine tuning)	0.83
DSN (one module, with fine tuning)	1.10
DSN (one module, no fine tuning)	1.78

6. EXPERIMENTS ON SPEECH CLASSIFICATION

We now report our experiments where similar kinds of DSN to Section 5 are applied to speech database of TIMIT. The same speech features and data are used as in [12]. Specifically, for the output at each module of the DSN, we use 183 target class labels (i.e., three states for each of the 61 phones), coded in binary zero or one, which we call "phone states".

In Fig. 4, we show the results of one typical experiment where we use 6000 units in each module of the DNN. Slightly different from the DSN used for the MNIST experiments, no skip-module output layers are used in building the higher modules of the DNN. In Fig 4, the frame-level phone-state classification percent error rate (with 183 state classes), together with three other performance measures, is plotted as a function of the training epoch (i.e. a full sweep of full 1.12M super-frames). These other measures include the frame-level phone classification percent error rate (61 phone classes as well as folded 39 phone classes) for the core test set when the errors in the state within the same phone are not counted. The results

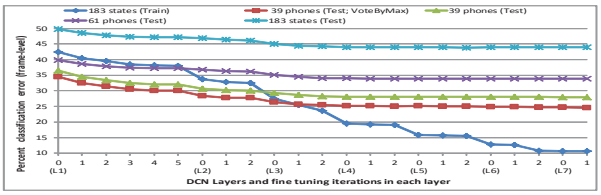

Fig. 4. Frame-level classification error rates on TIMIT core test and training sets. Training set error rates for classifying 183 states in blue; Test set error rates with 4 measures in separate colors.

shown in Figure 4 are obtained when the mini-batch size was set to 200,000 in fine tuning, and the step size in gradient descent is set to a value between 0.01 and 0.10. All hyper-parameters are tuned using the development set.

Like the MNIST task, we have also found empirically in the TIMIT task that if random numbers are used for initializing the weight matrix **W** instead of using RBM, then the error rate becomes significantly higher.

For comparison, we ran the DNN system of [12] on the same TIMIT data and found the corresponding frame-level phone state error rate to be 45.04 % (which gave excellent 22% to 23% phonetic recognition error rate after running a decoder with a "bi-phone language model" as reported in [12]). This frame-level error rate achieved by DNN is slightly higher than the DSNs error rate of 43.86% we have obtained.

In Table 3, we show the dependence of frame-level state and phone classification error rates on the hidden unit size and the number of modules in the DSN for the TIMIT core test set. All the results shown are obtained after convergence of fine tuning at each DSN module. It is interesting to note that three modules each with 2000 hidden units produce lower errors than one module each with 6000 hidden units. But 12 modules each with 2000 hidden units give higher errors than four modules each with 6000 hidden units. Selecting both depth and width are important to achieve the best performance.

The experiments in Table 3 are carried out with the fine-tuning algorithm that uses 250k super-frames as the mini-batch size, as any larger size would run out of memory in the experimental single machine with 48G memory. Parallel implementation of the learning algorithm over CPU clusters, which is ongoing, will remove such a constraint and enable full batch learning.

Table 3. Frame-level state and phone error rates vs. DSN modules and hidden unit size.

No. Hiddens	No. modules	State Err (%)	Phone Err (%)
6000	8	43.86	23.90
6000	4	44.24	24.25
6000	1	46.88	26.95
2000	12	44.29	24.56
2000	3	45.24	25.20
2000	1	51.07	30.95

7. SUMMARY AND FUTURE WORK

In this paper, we present a novel DSN architecture enabling parallel training on potentially very large data sets. Previous results on training DNNs for large-vocabulary speech recognition were limited to only 48 hours [2] and 300 hours [3] of training data. The new architecture is likely to free this limit. Experimental results on both MNIST and TIMIT demonstrate higher classification rates achieved by DSN than DNN.

In addition to the success of DSN as a powerful classifier on speech and image classification tasks presented in this paper, DSN has more recently been applied to speech understanding [13] with strong results. Also, the drastic error reduction of speech attribute detection achieved already by using DBN [14] is expected to be again enhanced by DSN (work ongoing). Further, our recent extension of the DSN architecture to a tensor version [15] has demonstrated huge architectural flexibilty of the stacking and parallel learning mechanisms presented in this paper, and a vast opportunity exists in exploiting such flexibility for potential successful applications in a wide range of information processing tasks.

8. REFERENCES

[1] G. Hinton and R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, pp. 504–507, 2006.

[2] G. Dahl, D. Yu, L. Deng, and A. Acero, "Context-dependent pre-trained dnns for large vocabulary speech recognition," *IEEE Trans. Audio, Speech, and Lang. Proc.*, Jan. 2012.

[3] F. Seide, G. Li, and D. Yu, "Conversational speech transcription using context-dependent deep neural networks," in *Proc. Interspeech*, August 2011.

[4] Y. Bengio, "Learning deep architectures for AI," *Foundations and Trends in Machine Learning*, vol. 2, pp. 1–127, 2009.

[5] D.H. Wolpert, "Stacked generalization," *Neural Networks*, vol. 5, no. 2, pp. 241–259, 1992.

[6] SE Fahlman and C Lebiere, "The cascade-correlation learning architecture," in *Proc. NIPS*, 1990, pp. 524–532.

[7] William W. Cohen and Vitor Rocha de Carvalho, "Stacked sequential learning," in *Proc. IJCAI*, 2005, pp. 671–676.

[8] D. Yu, S. Wang, and L. Deng, "Sequential labeling using deep-structured conditional random fields," *IEEE J. Selected Topics in Signal Proc.*, vol. 4, no. 6, pp. 965–972, 2010.

[9] R. Salakhutdinov and G. Hinton, "Deep Boltzmann machines," in *Proc. AISTATS*, 2009.

[10] L. Deng and D. Yu, "Deep convex network: A scalable architecture for speech pattern classification," in *Proc. Interspeech*, August 2011.

[11] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[12] A. Mohamed, D. Yu, and L. Deng, "Investigation of full-sequence training of deep belief networks for speech recognition," in *Proc. Interspeech*, September 2010.

[13] G. Tur, L. Deng, D. Hakkani-Tur, and X. He, "Toward deeper understanding: Deep convex networks for semantic utterance classification," in *Proc. ICASSP*, Kyoto, Japan, 2012.

[14] D. Yu, S. Siniscalchi, L. Deng, and C. Lee, "Boosting attribute and phone estimation accuracies with deep neural networks for detection-based speech recognition," in *Proc. ICASSP*, Kyoto, Japan, 2012.

[15] B. Hutchinson, L. Deng, and D. Yu, "A deep architecture with bilinear modeling of hidden representations: applications to phonetic recognition," in *Proc. ICASSP*, Kyoto, Japan, 2012.