Deeply-Supervised Nets

Chen-Yu Lee [*]
Dept. of ECE, UCSD
chl260@ucsd.edu

Saining Xie [*]
Dept. of CSE and CogSci, UCSD
s9xie@ucsd.edu

Patrick Gallagher
Dept. of CogSci, UCSD
rexaran@gmail.com

Zhengyou Zhang
Microsoft Research
zhang@microsoft.com

Zhuowen Tu
Dept. of CogSci, UCSD
ztu@ucsd.edu

Abstract

We propose deeply-supervised nets (DSN), a method that simultaneously minimizes classification error while improving the directness and transparency of the hidden layer learning process. We focus our attention on three specific aspects in traditional convolutional-neural-network-type (CNN-type) architectures: (1) transparency in the effect that intermediate layers have on the overall classification; (2) discriminativeness and robustness of learned features, especially in early network layers; (3) training effectiveness in the face of "exploding" and "vanishing" gradients. To combat these issues, we introduce "companion" objective functions at each individual hidden layer, in addition to the overall objective function at the output layer (a strategy distinct from layer-wise pre-training). We also analyze our algorithm using techniques extended from stochastic gradient methods. The advantages provided by our method are evident in our experimental results on benchmark datasets, showing state-of-the-art performance on MNIST, CIFAR-10, CIFAR-100, and SVHN.

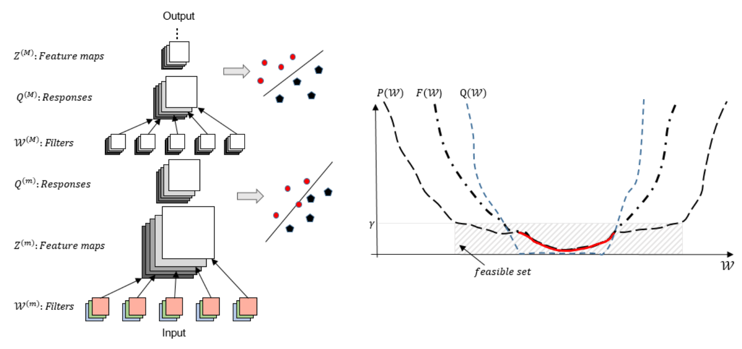
1 Introduction

Much attention has been given to the resurgence of neural networks, most prominently in the form of deep learning (DL). The application of deep learning (DL) techniques to large training sets has yielded significant performance gains in image classification [11, 15] and speech recognition [4]. However, even though hierarchical and recursive networks [7, 10, 12] have shown great promise in automatically learning thousands or even millions of features for pattern recognition, there nonetheless remain many fundamental open questions about deep learning.

These open questions arise in conjunction with various aspects of current DL frameworks: the features learned at hidden layers (and at early hidden layers in particular) are not always "transparent" in their meaning and at times display reduced discrimination ability [29]; the presence of "exploding" and "vanishing" gradients can sometimes lead to difficulty in training [8, 19]; despite some theoretical work [6], the present mathematical understanding of DL remains at an early stage. Notwithstanding the issues in these questions, DL has proven eminently capable of automatically learning rich hierarchical features combined within an integrated network. Recent techniques such as dropout [11], dropconnect [16], pre-training [4], and data augmentation [21] bring enhanced performance from a variety of perspectives. Beyond this, recent code- and experience-sharing [11, 5, 2] has greatly sped the adoption and advance of DL inside and outside of the machine learning community.

In this paper, we present a new DL approach that we call deeply-supervised nets (DSN). The central idea of DSN is to provide direct supervision to the hidden layers, rather than the standard approach of providing supervision only at the output layer and subsequently propagating this supervision back to earlier layers. We provide this direct hidden layer supervision by introducing *companion objective functions* for the individual hidden layers; the presence of these companion objective functions can be seen as an additional (soft) constraint (or as a new regularization) within the learning process. We present experiments showing significant performance improvements relative to existing methods. In

[*]Equal contribution.



(a) DSN illustration (b) functions

Figure 1: Network architecture for the proposed deeply-supervised nets (DSN).

addition, we also use analysis techniques from stochastic gradient methods to investigate a restricted setting in which we can see how incorporating companion objective functions directly leads to improved convergence rate. The advantage of such deep supervision is evident: (1) **for small training data and relatively shallower networks**, deep supervision functions as a strong "regularization" for classification accuracy and learned features; (2) **for large training data and deeper networks** deep supervision makes it convenient to exploit the significant performance gains that extremely deep networks can bring by improving otherwise problematic convergence behavior.

Our experiments are primarily based on L2SVM objectives, previously used in DL by [27] for the output layer only; we also show that our approach of introducing companion objectives is not dependent on the use of L2SVM objectives — when using softmax we find similar performance improvements. Our experiments show consistent improvement of DSN-L2SVM and DSN-Softmax over CNN-L2SVM and CNN-Softmax, respectively. On the topic of bringing some form of specific attention to hidden layers, there has been some previous work. For example, in [1] layer-wise supervised pre-training is performed. Our proposed method does not perform pre-training and it emphasizes the importance of minimizing the output classification error while reducing the prediction error of each individual layer. This is important as the back propagation is thus performed altogether in an integrated manner. There are a few other relevant examples: in [24], label information is used for unsupervised learning, while in [28] semi-supervised learning is carried out in the context deep learning. We notice that GoogLeNet [26] uses supervision on 2 hidden layers in a 22-layer CNN, but the two papers differ in the focus, formulation, emphasis, and analysis.

2 Deeply-Supervised Nets

Our approach is built using the infrastructure provided by existing supervised CNN-type frameworks [12, 5, 2]. We extend them by introducing a classifier, either SVM or Softmax, at hidden layers.

2.1 Motivation

Our motivation for introducing these classifiers at hidden layers comes from the following simple observation: in general, a discriminative classifier trained on highly discriminative features will display better performance than a discriminative classifier trained on less discriminative features. If the features in question are the hidden layer feature maps of a deep network, this observation means that the performance of a discriminative classifier trained using these hidden layer feature maps can serve as a proxy for the quality/discriminativeness of those hidden layer feature maps. We also expect this deep supervision to alleviate the common problem of having gradients that "explode" or "vanish". One concern with a direct pursuit of feature discriminativeness at all hidden layers is that this might interfere with the overall network performance; our experimental results indicate that this is not the case. Our additional deep feedback is brought in by associating a "companion" classification output with each hidden layer. We may think of this companion output as analogous to the final output that a truncated network would have produced. Backpropagation of error proceeds as usual, with the crucial difference that we backpropagate not only from the final layer but also from our local companion output. The empirical results suggest the following main properties of the companion objective: (1) it is a type of feature regularization (albeit an unusual one) — it leads to reduction in testing error, while not necessarily reducing the training error; (2) it results in improved convergence behavior, requiring much less manual tweaking (particularly for very deep networks).

2.2 Formulation

Here we focus on the supervised learning case. We denote our input training data set by $S = \{(\mathbf{X}_i, y_i), i = 1 \dots N\}$, where sample $\mathbf{X}_i \in R^n$ denotes the raw input data and $y_i \in \{1, \dots, K\}$ denotes the corresponding ground truth label for sample $\mathbf{X}_i$. We subsequently drop the subscript i for notational simplicity, since we consider each sample independently. The goal of deep neural networks, specifically convolutional neural networks (CNN) [12], is to learn layers of filters and weights for minimization of output layer classification error. In our present discussion, we absorb the bias term into the weight parameters and do not differentiate weights from filters. Furthermore, we denote a recursive function for each layer $m = 1 \dots M$ as

$$\mathbf{Z}^{(m)} = f(\mathbf{Q}^{(m)}), \quad \text{and} \quad \mathbf{Z}^{(0)} \equiv \mathbf{X}, \tag{1}$$

$$\mathbf{Q}^{(m)} = \mathsf{W}^{(m)} * \mathbf{Z}^{(m-1)}, \tag{2}$$

where M denotes the total number of layers, $\mathsf{W}^{(m)},\ m = 1 \dots M$ are the filters/weights to be learned, $\mathbf{Z}^{(m-1)}$ is the feature map produced at layer $m-1$, $\mathbf{Q}^{(m)}$ refers to the convolved/filtered responses on the previous feature map, and $f(\cdot)$ is a pooling function on $\mathbf{Q}$. Combining all layers of weights gives

$$\mathsf{W} = (\mathsf{W}^{(1)}, \dots, \mathsf{W}^{(M)}).$$

We next associate one classifier with each hidden layer; our derivation is stated for L2SVM but we later show experimental results for both L2SVM and softmax. We also tested L1SVM, finding results that differed negligibly from those of L2SVM. We denote the corresponding weights by

$$\mathbf{w} = (\mathbf{w}^{(1)}, \dots, \mathbf{w}^{(M-1)}),$$

in addition to the W from the standard CNN framework. We denote the classifier weights for the output layer by $\mathbf{w}^{(out)}$. Our total combined objective function is then

$$\|\mathbf{w}^{(out)}\|^2 + \mathcal{L}(\mathsf{W}, \mathbf{w}^{(out)}) + \sum_{m=1}^{M-1} \alpha_m [\|\mathbf{w}^{(m)}\|^2 + \ell(\mathsf{W}, \mathbf{w}^{(m)}) - \gamma]_+, \tag{3}$$

where

$$\mathcal{L}(\mathsf{W}, \mathbf{w}^{(out)}) = \sum_{y_k \neq y} [1 - <\mathbf{w}^{(out)}, \phi(\mathbf{Z}^{(M)}, y) - \phi(\mathbf{Z}^{(M)}, y_k)>]_+^2 \tag{4}$$

and

$$\ell(\mathsf{W}, \mathbf{w}^{(m)}) = \sum_{y_k \neq y} [1 - <\mathbf{w}^{(m)}, \phi(\mathbf{Z}^{(m)}, y) - \phi(\mathbf{Z}^{(m)}, y_k)>]_+^2 \tag{5}$$

We refer to $\mathcal{L}(\mathsf{W}, \mathbf{w}^{(out)})$ as the *overall loss* (associated with the output layer) and to $\ell(\mathsf{W}, \mathbf{w}^{(m)})$ as the *companion loss* (associated with the hidden layers); in the L2SVM setting these are both (squared) hinge losses of prediction errors. The preceding formulation has an intuitive description: in addition to learning convolution kernels and weights, W (as in the standard CNN model [12]) we incorporate an additional penalty at each hidden layer associated with good label prediction for that layer; this additional penalty strongly favors features that are discriminative/sensible at each hidden layer. In eqn. (3), $\|\mathbf{w}^{(out)}\|^2$ and $\mathcal{L}(\mathsf{W}, \mathbf{w}^{(out)})$ are, respectively, the margin and the (squared) hinge loss of the classifier output layer; we omit the balance parameter C in front of the (squared) hinge loss for notational simplicity. In eqn. (4), $\|\mathbf{w}^{(m)}\|^2$ and $\ell(\mathsf{W}, \mathbf{w}^{(m)})$ are, respectively, the margin and the (squared) hinge loss of the hidden layer classifier.

Note that for each $\ell(\mathsf{W}, \mathbf{w}^{(m)})$, the $\mathbf{w}^{(m)}$ directly depends on $\mathbf{Z}^{(m)}$, which is in turn dependent on $\mathsf{W}^1, .., \mathsf{W}^m$ up to the mth layer. Analogously, $\mathcal{L}(\mathsf{W}, \mathbf{w}^{(out)})$ depends on $\mathbf{w}^{(out)}$, which is decided by the entire W. The second term in eqn. (3) often is sent to zero in the course of training; this means that the overall goal of producing good classification of the output layer is not fundamentally altered and the companion objective acts as a type of regularization or as a proxy for discriminative features. One method by which we pursue this zero-ing is by having γ as a threshold (a hyper parameter) in the second term of eqn. (3): once the companion objective of each hidden layer falls to γ (or below), it vanishes and no longer contributes to the gradient update in the learning process. The balance parameter α_m represents a trade-off between the output objective and the respective companion objective. An alternative approach to zero-ing, is the use of a simple decay function such as $\alpha_m \times 0.1 \times (1 - t/N) \rightarrow \alpha_m$ to enforce the second term to vanish after certain number of iterations, where t is the epoch step and N is the total number of epochs. We have obtained promising preliminary results from each of these two approaches; this issue remains to be more fully explored.

The main difference between eqn. (3) and previous research using layer-wise supervised training is that we perform the optimization together with a term serving as a proxy for the hidden layer feature quality. This combined learning is in contrast to previous research in which greedy layer-wise pre-training was performed as either initialization or fine-tuning, a strategy that resulted in overfitting [1]. The state-of-the-art benchmark results show that the deep supervision approach does not display harmful levels of overfitting: as seen in Figure (2.c), while the training error for both CNN and DSN is eventually near zero, DSN shows lower test error and so demonstrates its advantage in generalization over standard CNN. This is a matter for further investigation, but the present results are very promising.

We continue with additional discussion of our method; the aim is to gain further intuitive understanding of eqn. (3). For ease of reference, we write the total objective function as

$$F(\mathsf{W}) \equiv \mathcal{P}(\mathsf{W}) + \mathcal{Q}(\mathsf{W}), \tag{6}$$

in which we have the output objective $\mathcal{P}(\mathsf{W}) \equiv \|\mathbf{w}^{(out)}\|^2 + \mathcal{L}(\mathsf{W}, \mathbf{w}^{(out)})$ and the summed companion objectives $\mathcal{Q}(\mathsf{W}) \equiv \sum_{m=1}^{M-1} \alpha_m [\|\mathbf{w}^{(m)}\|^2 + \ell(\mathsf{W}, \mathbf{w}^{(m)}) - \gamma]_+$.

To train the DSN model using SGD, the gradients of the objective function w.r.t the network weights are:

$$\frac{\partial F}{\partial \mathbf{w}^{(out)}} = 2\mathbf{w}^{(out)} - 2\sum_{y_k \neq y}[\phi(\mathbf{Z}^{(M)}, y) - \phi(\mathbf{Z}^{(M)}, y_k)] \cdot \\ [1 - <\mathbf{w}^{(out)}, \phi(\mathbf{Z}^{(M)}, y) - \phi(\mathbf{Z}^{(M)}, y_k)>]_+$$

$$\begin{aligned} \frac{\partial F}{\partial \mathbf{w}^{(m)}} &= 0, \text{ when } \|\mathbf{w}^{(m)}\|^2 + \ell(\mathsf{W}, \mathbf{w}^{(m)}) \leq \gamma \quad \textit{and} \\ &= \alpha_m\{2\mathbf{w}^{(m)} - 2\sum_{y_k \neq y}[\phi(\mathbf{Z}^{(m)}, y) - \phi(\mathbf{Z}^{(m)}, y_k)] \\ &\quad [1 - <\mathbf{w}^{(m)}, \phi(\mathbf{Z}^{(m)}, y) - \phi(\mathbf{Z}^{(m)}, y_k)>]_+\}, \text{ otherwise.} \end{aligned}$$

Note that we do need to particularly emphasize using L2SVM here and other functions, e.g. Softmax, can be adopted as well (we also show the result of DSN-Softmax) with their corresponding gradient functions computed. The gradient w.r.t W follows the conventional CNN-based model, plus the gradient that comes from the hidden layer direct supervision; we also emphasize one approach to zero-ing, described above.

2.3 Stochastic Gradient Descent View

In addition to the present observation that learned features in CNN are not always intuitive and discriminative [29], broader discussion of difficulties in training deep neural networks has appeared in [8, 19]. As we can observe from eqn. (1) and (2), the change of the bottom layer weights gets propagated through layers of functions; this can lead to "exploding" or "vanishing" gradients [19]. Various techniques and parameter tuning tricks have been proposed to better train deep neural networks, such as pre-training. This difficulty in training partially raises some concerns about the DL frameworks. Here we provide analysis of our proposed formulation, in a hope to understand some aspects of its experimentally-observed advantage in effectiveness and efficiency over existing CNN-style approaches.

The objective function in deep neural networks is highly non-convex, a characteristic contributing to the current lack of a clear mathematical/statistical analysis of DL frameworks. A common approach is to sacrifice some realism for increased tractability, typically by restricting attention only to settings in which local convexity holds. Here we follow this example by supposing that our objective functions are locally λ-strongly convex and following analysis techniques from stochastic gradient descent [3].

The definition of λ-strong convexity is standard: A function $F(\mathsf{W})$ is λ-strongly convex on a set $\mathcal{W}$ if $\forall\, \mathsf{W}, \mathsf{W}' \in \mathcal{W}$ and any subgradient $\mathbf{g}$ at W,

$$F(\mathsf{W}') \geq F(\mathsf{W}) + < \mathbf{g}, \mathsf{W}' - \mathsf{W} > + \frac{\lambda}{2}\|\mathsf{W}' - \mathsf{W}\|^2. \tag{7}$$

The Stochastic Gradient Descent (SGD) update rule at step t is $\mathsf{W}_{t+1} = \Pi_{\mathcal{W}}(\mathsf{W}_t - \eta_t \hat{\mathbf{g}})$, where $\eta_t = \Theta(1/t)$ refers to the step factor and $\Pi_{\mathcal{W}}$ denotes projection onto the set $\mathcal{W}$. Let $\mathsf{W}^\star$ be the optimal solution, suppose that there are upper bounds for $\mathbb{E}[\|\mathsf{W}_T - \mathsf{W}^\star\|^2]$ and $\mathbb{E}[F(\mathsf{W}_T) - F(\mathsf{W}^\star)]$ in the strongly convex function setting [20], or for $\mathbb{E}[F(\mathsf{W}_T) - F(\mathsf{W}^\star)]$ in the convex function setting [22].

Here we make an attempt to understand the convergence of eqn. (6) w.r.t. $\mathbb{E}[\|\mathsf{W}_T - \mathsf{W}^\star\|^2]$ keeping in mind the assumed characteristics roughly illustrated in Figure (1.b). In [18], a convergence rate is given for M-estimators with locally convex function with compositional loss and regularization terms.

Definition We denote by $\mathcal{S}_\gamma(F) = \{\mathsf{W} \mid F(\mathsf{W}) \leq \gamma\}$ the γ-sublevel set, stated here for the function $F(\mathsf{W}) \equiv \mathcal{P}(\mathsf{W}) + \mathcal{Q}(\mathsf{W})$.

First we show that $\mathsf{W} \in \mathcal{S}_\gamma(\mathcal{Q})$ implies that $\mathsf{W} \in \mathcal{S}_\gamma(\mathcal{P})$. That is:

Lemma 1 *$\forall m, m' = 1 \dots M-1$, and $m' > m$ if $\|\mathbf{w}^{(m)}\|^2 + \ell((\hat{\mathsf{W}}^{(1)}, .., \hat{\mathsf{W}}^{(m)}), \mathbf{w}^{(m)}) \leq \gamma$ then there exists $(\hat{\mathsf{W}}^{(1)}, .., \hat{\mathsf{W}}^{(m)}, .., \hat{\mathsf{W}}^{(m')})$ such that $\|\mathbf{w}^{(m')}\|^2 + \ell((\hat{\mathsf{W}}^{(1)}, .., \hat{\mathsf{W}}^{(m)} .., \hat{\mathsf{W}}^{(m')}), \mathbf{w}^{(m')}) \leq \gamma$.* [1]

Proof As we can see from an illustration of our network architecture shown in fig. (1.a), for all $(\hat{\mathsf{W}}^{(1)}, .., \hat{\mathsf{W}}^{(m)})$ such that $\ell((\hat{\mathsf{W}}^{(1)}, .., \hat{\mathsf{W}}^{(m)}), \mathbf{w}^{(m)}) \leq \gamma$ there is a "trivial" way to achieve this performance at the output layer: for each layer $j > m$ up to m', we let $\hat{\mathsf{W}}^{(j)} = \mathbf{I}$ and $\mathbf{w}^{(j)} = \mathbf{w}^{(m)}$, meaning that the filters will be identity matrices. This results in $\ell((\hat{\mathsf{W}}^{(1)}, .., \hat{\mathsf{W}}^{(m)} .., \hat{\mathsf{W}}^{(m')}), \mathbf{w}^{(m')}) \leq \gamma$. □

Remark Lemma 1 shows that a good solution for $\mathcal{Q}(\mathsf{W})$ is also a good one for $\mathcal{P}(\mathsf{W})$; of course, the converse needs not be the case. That is: a W that makes $\mathcal{P}(\mathsf{W})$ small may not necessarily produce features for which the hidden layers will have a small $\mathcal{Q}(\mathsf{W})$. As mentioned previously, $\mathcal{Q}(\mathsf{W})$ can be viewed as a regularization term. If it happens that $\mathcal{P}(\mathsf{W})$ possesses an essentially flat area (relative to the training data) in the vicinity of some local optimum of interest and it is ultimately the test error that we really care about, we would be able to focus on the setting of W, $\mathsf{W}^\star$, that will make both $\mathcal{Q}(\mathsf{W})$ and $\mathcal{P}(\mathsf{W})$ small. Therefore, it is not unreasonable to assume that $F(\mathsf{W}) \equiv \mathcal{P}(\mathsf{W}) + \mathcal{Q}(\mathsf{W})$ and $\mathcal{P}(\mathsf{W})$ share the same optimal $\mathsf{W}^\star$.

Let $\mathcal{P}(\mathsf{W})$ and $\mathcal{Q}(\mathsf{W})$ be locally strongly convex around $\mathsf{W}^\star$, $\|\mathsf{W}' - \mathsf{W}^\star\|^2 \leq D$ and $\|\mathsf{W} - \mathsf{W}^\star\|^2 \leq D$, with $\mathcal{P}(\mathsf{W}') \geq \mathcal{P}(\mathsf{W}) + < \mathbf{gp}, \mathsf{W}' - \mathsf{W} > + \frac{\lambda_1}{2}\|\mathsf{W}' - \mathsf{W}\|^2$ and $\mathcal{Q}(\mathsf{W}') \geq \mathcal{Q}(\mathsf{W}) + < \mathbf{gq}, \mathsf{W}' - \mathsf{W} > + \frac{\lambda_2}{2}\|\mathsf{W}' - \mathsf{W}\|^2$, where $\mathbf{gp}$ and $\mathbf{gq}$ are subgradients for W for $\mathcal{P}$ and $\mathcal{Q}$, respectively. It can be directly seen that $F(\mathsf{W})$ is also strongly convex and that $\mathbf{gf} = \mathbf{gp} + \mathbf{gq}$ is a subgradient of $F(\mathsf{W})$ at W.

Lemma 2 *Suppose $\mathbb{E}[\|\hat{\mathbf{gp}}_t\|^2] \leq G^2$ and $\mathbb{E}[\|\hat{\mathbf{gq}}_t\|^2] \leq G^2$, and we use the update rule of $\mathsf{W}_{t+1} = \Pi_{\mathcal{W}}(\mathsf{W}_t - \eta_t(\hat{\mathbf{gp}}_t + \hat{\mathbf{gq}}_t))$ where $\mathbb{E}[\hat{\mathbf{gp}}_t] = \mathbf{gp}_t$ and $\mathbb{E}[\hat{\mathbf{gq}}_t] = \mathbf{gq}_t$. If we use $\eta_t = 1/(\lambda_1 + \lambda_2)t$, then at iteration T*

$$\mathbb{E}[\|\mathsf{W}_T - \mathsf{W}^\star\|^2] \leq \frac{4G^2}{(\lambda_1 + \lambda_2)^2 T} \tag{8}$$

Proof Since $F(\mathsf{W}) = \mathcal{P}(\mathsf{W}) + \mathcal{Q}(\mathsf{W})$, it can be directly seen that

$$F(\mathsf{W}') \geq F(\mathsf{W}) + < \mathbf{gp} + \mathbf{gq}, \mathsf{W}' - \mathsf{W} > + \frac{\lambda_1 + \lambda_2}{2}\|\mathsf{W}' - \mathsf{W}\|^2.$$

Based on lemma 1 in [20], this upper bound directly holds. □

Lemma 3 *We follow the assumptions in lemma 2, with the exception that we assume $\eta_t = 1/t$ since λ_1 and λ_2 are not always readily available; as we discuss in the appendix, we also expect the combined λ to be small. When we begin in the region $\|\mathsf{W}_1 - \mathsf{W}^\star\|^2 \leq D$, the convergence rate is bounded by*

$$\mathbb{E}[\|\mathsf{W}_T - \mathsf{W}^\star\|^2] \leq e^{-2\lambda(\ln(T-1)+0.578)} D + 4G^2 \sum_{t=1}^{T-1} \frac{1}{t^2}\left(\frac{t}{T-1}\right)^{2\lambda} \tag{9}$$

Proof See arXiv:1409.5185.

Theorem 1 *Let $\mathcal{P}(\mathsf{W})$ be λ_1-strongly convex and $\mathcal{Q}(\mathsf{W})$ be λ_2-strongly convex near optimal $\mathsf{W}^\star$ and denote by $\mathsf{W}_T^{(F)}$ and $\mathsf{W}_T^{(\mathcal{P})}$ the solution after T iterations when following SGD on $F(\mathsf{W})$ and $\mathcal{P}(\mathsf{W})$, respectively. Then DSN framework in eqn. (3) improves the relative convergence speed $\frac{\mathbb{E}[\|\mathsf{W}_T^{(\mathcal{P})} - \mathsf{W}^\star\|^2]}{\mathbb{E}[\|\mathsf{W}_T^{(F)} - \mathsf{W}^\star\|^2]}$, viewed from the ratio of their upper bounds as $\Theta(\frac{(\lambda_1+\lambda_2)^2}{\lambda_1^2})$, when $\eta_t = 1/\lambda t$.*

[1] Note that we drop the $\mathsf{W}^{(j)}, j > m$ since the filters above layer m do not participate in the computation for the loss function of this layer.

Proof Lemma 1 shows the compatibility of the companion objective of $\mathcal{Q}$ w.r.t the output objective $\mathcal{P}$. This equation can be directly derived from lemma 2.

Remark When $\eta_t = 1/t$, T is large, and the first term of eqn. (9) dominates, the upper bound ratio for $\frac{\mathbb{E}[\|\mathsf{W}_T^{(\mathcal{P})}-\mathsf{W}^\star\|^2]}{\mathbb{E}[\|\mathsf{W}_T^{(F)}-\mathsf{W}^\star\|^2]}$ is roughly at the order of $\Theta(e^{2\ln(T)\lambda_2})$. If the second term of eqn. (9) dominates, the convergence of $F(\mathsf{W})$ over $\mathcal{P}(\mathsf{W})$ is also advantageous with the ratio at an order of $\Theta(e^{2\ln((T-1)/(T-2))\lambda_2})$ loosely.

In general we might expect $\lambda_2 \gg \lambda_1$ which would lead to a great improvement in convergence speed. $\square$

3 Experiments

We evaluate the proposed DSN method on four standard benchmark datasets: MNIST, CIFAR-10, CIFAR-100, and SVHN. In addition, we also use the ImageNet dataset to assess the behavior of DSN on large dataset. We use MNIST as our primary running example. For our MNIST experiments we used an implementation of DSN built on top of the Theano platform [2]. The rest of the experiments are performed using DSN built on top of [17], except for the ImageNet where we use [14] due to its parallel computing infrastructure. We use the SGD solver with mini-batch size of 128 at a fixed constant momentum value of 0.9. The initial values for the learning rate and the weight decay factor are determined based on the validation set. For a fair comparison and clear illustration of the effectiveness of DSN, we match the complexity of our model to the complexity of the network architectures used in [17] and [9] so that there are a comparable number of parameters. We also incorporate two dropout layers with dropout rate at 0.5. Companion objective at the convolutional layers is imposed to back-propagate the classification error guidance at the associated layer. The proposed DSN framework is not difficult to train and there are no particular engineering tricks adopted.

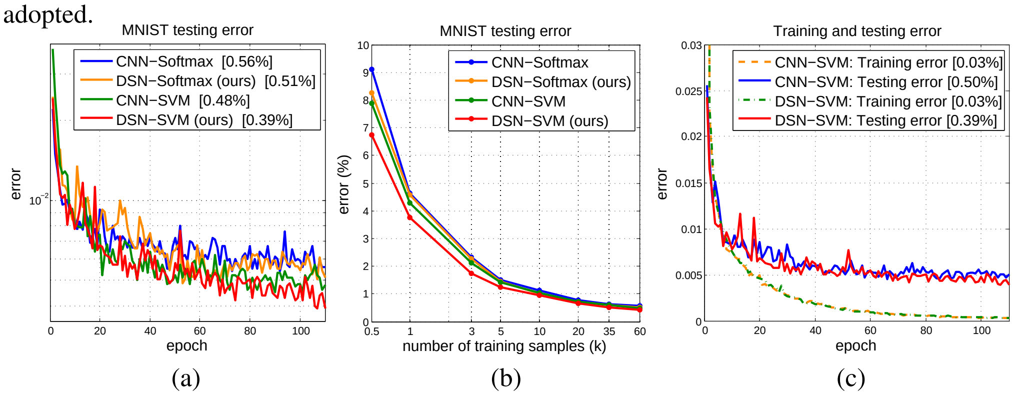


Figure 2: Classification error on MNIST test. (a) shows test error of competing methods; (b) shows test error versus training sample size. (c) train and test error comparison.

DSN can be equipped with different types of classifier objective functions; we consider L2SVM and softmax and show that DSN-L2SVM and DSN-Softmax yield performance boosts over the corresponding CNN-L2SVM and CNN-Softmax approaches (see Figure (2.a)). The performance gain is more evident in the regime of small training data (see Figure (2.b)); this might partially ease the burden of requiring large training data for DL. Overall, we observe state-of-the-art classification error on all four datasets. All results are achieved without using averaging [21], which could lead to further improvement in classification accuracy. Figure (3) gives an illustration of some learned features. The result on the ImageNet classification task is also very encouraging (the most recent winning systems [26, 23] for this challenge involved many specific engineering steps and used a cluster of machines for training). Again, our DSN method can be combined with many existing CNN-type methods. Overall, as stated before: for small training data and relatively shallow networks, DSN functions as a strong "regularization"; for large training data and very deep networks, DSN makes the training process convenient, which might be otherwise problematic in standard CNN.

MNIST

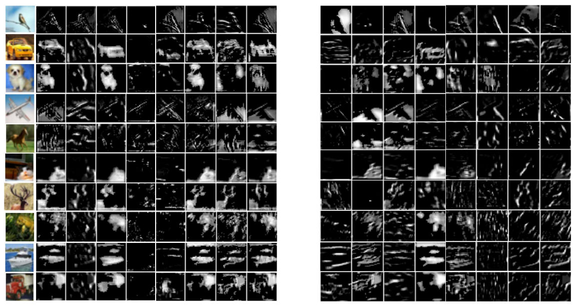
The MNIST dataset consists of 28×28 images from 10 different classes (0 to 9) with 60,000 training samples and 10,000 test samples. We use a fairly standard network that contains two convolutional layers followed by a fully-connected layer. Both convolutional layers use 5×5 filter size with 32 and 64 channels, respectively. Relu hidden units and 2×2 max pooling with stride 2 are used after each convolutional layer. The fully-connected layer has 500 hidden nodes that uses Relu activation with dropout rate of 0.5.

Figure (2.a) and (b) show results from four methods, namely: (1) conventional CNN with softmax loss (CNN-Softmax), (2) the proposed DSN with softmax loss (DSN-Softmax), (3) CNN with L2SVM objective (CNN-L2SVM) , and (4) the proposed DSN with L2SVM objective (DSN-L2SVM). DSN-Softmax and DSN-L2SVM outperform their respective competing CNN algorithms (DSN-L2SVM shows classification error of 0.39% under a single model without data whitening and augmentation). Figure (2.b) shows the classification error of the competing methods when trained with varying sizes of training samples (26% gain of DSN-L2SVM over CNN-Softmax at 500 samples. Figure (2.c) shows a comparison of generalization error between CNN and DSN. A performance comparison is shown in Table 1. To investigate the difference between DSN and layer-wise pre-training (shown as CNN with pre-training in Table 1) , we train a CNN model using greedy layer-wise pre-training method as in [1]. Under the same network architecture, layer-wise pre-training performs better than CNN without pre-traning but worse than DSN.

Table 1: MNIST classification result. (without data augmentation and model averaging)

Method	**Error(%)**
CNN [13]	0.53
Stochastic Pooling [30]	0.47
Network in Network [17]	0.47
Maxout Networks [9]	0.45
CNN with pre-training	0.43
DSN (ours)	**0.39**

Figure 3: Visualization of the feature maps learned in the convolutional layer.


(a) by DSN (b) by CNN

CIFAR-10 and CIFAR-100

The CIFAR-10 dataset consists of 32×32 color images. A collection of 60,000 images is split into 50,000 training and 10,000 testing images. The dataset is preprocessed by global contrast normalization. To compare our results with the previous state-of-the-art, for this dataset we also augmented the dataset by zero-padding 4 pixels on each side; we also perform corner cropping and random flipping on the fly during training. For a fair comparison, we adopt a network architecture similar to [17] that contains three convolutional layers with 192 filter channels. We also use the mlpconv layer after each convolutional layers and global averaged pooling scheme with kernel size 8 for the output prediction. Relu neuron with 0.5 dropout rate and 3×3 max pooling with stride 2 are used after the first two convolutional layer. No model averaging is done at the test phase. Table 2 shows our results. Our DSN model achieves an error rate of 9.69% without data augmentation and of 7.97% with data augmentation (the best known result to our knowledge). Notice that DSN still outperform greedy layer-wise pre-training (shown as CNN with pre-training in Table 2) under the same network architecture.

DSN also provides added robustness to hyperparameter choice, in that the early layers are guided with direct classification loss, leading to a faster convergence rate and reduced dependence on heavy hyperparameter tuning. We also compared the gradients in DSN with those in CNN, observing 4.55 times greater gradient variance of DSN over CNN in the first convolutional layer. This is consistent with an observation in [9] and with the assumptions and motivations we make in this work. To see what features have been learned in DSN vs. CNN, we select one example image from each of the ten categories of CIFAR-10, run one forward pass, and show the feature maps learned from the bottom convolutional layer in Figure (3). Only the top 30% of activations are shown in each of the feature maps. Feature maps learned by DSN appear to be more intuitive than those learned by CNN.

CIFAR-100 is similar to CIFAR-10, but with 100 classes rather than 10. The number of images for each class is then 500 instead of $5,000$ as in CIFAR-10, which makes the classification task more challenging. We use the same network settings as in CIFAR-10. The consistent performance boost (shown on both CIFAR-10 and CIFAR-100) again demonstrates the advantage of the DSN method.

Table 2: CIFAR-10 classification error.

Method	Error(%)
No Data Augmentation	
Stochastic Pooling [30]	15.13
Maxout Networks [9]	11.68
Network in Network [17]	10.41
CNN with pre-training	9.92
DSN (ours)	**9.69**
With Data Augmentation	
Maxout Networks [9]	9.38
Dropconnect [16]	9.32
Network in Network [17]	8.81
DSN (ours)	**7.97**

Table 3: CIFAR-100 classification error.

Method	Error(%)
Stochastic Pooling [30]	42.51
Maxout Networks [9]	38.57
Tree based Priors [25]	36.85
Network in Network [17]	35.68
DSN (ours)	**34.57**

Table 4: SVHN classification error.

Method	Error(%)
Stochastic Pooling [30]	2.80
Maxout Networks [9]	2.47
Network in Network [17]	2.35
Dropconnect [16]	1.94
DSN (ours)	**1.92**

Table 5: ImageNet 2012 classification error.

Method	top-1 val. error(%)	top-5 val. error(%)
CNN 8-layer [14]	40.7	18.2
DSN 8-layer (ours)	39.6	17.8
CNN 11-layer	34.5	13.9
DSN 11-layer (ours)	33.7	13.1

Street View House Numbers

The Street View House Numbers (SVHN) dataset consists of 32×32 color images: $73,257$ digits for training, $26,032$ digits for testing, and $531,131$ extra training samples. We prepared the data in keeping with previous work, namely: we select 400 samples per class from the training set and 200 samples per class from the extra set. The remaining 598,388 images are used for training. We followed [9] to preprocess the dataset by Local Contrast Normalization (LCN). We do not do data augmentation in training and use only a single model in testing. Table 4 shows recent comparable results. Note that Dropconnect [16] uses data augmentation and multiple model voting.

3.1 ImageNet Challenge

To further illustrate the effectiveness of DSN on large training datasets and with deeper networks, we test both CNN and DSN on the ImageNet 2012 dataset. We emphasize that our intention here is not to directly compete with the best performing result in the challenge [26], but to provide an illustrative comparison of the relative benefits of DSN versus CNN on this data set. The 8-layer network's configuration is based on the AlexNet [14] with 5 convolutional layers and 3 fully-connected layers, while the 11-layer network's infrastructure contains 8 convolutional layers and 3 fully-connected layers. To decouple the effect of parameter tuning, we use the general training procedure as in [14] for both CNN and DSN. All results are tested under a single net without multi-scale training/testing in order to reduce the effect of model/prediction averaging. Table 5 shows a direct comparison between CNN and DSN.

When the depth of the network increases to 11 layers, we observe that the average absolute gradient of the weight matrix of conv1 layer to be at magnitude of 10^{-10}, making the networks significantly difficult to converge. Therefore, we have to adopt a pre-training strategy suggested in [23]: we first pre-train a relatively shallower 8-layer network and then restart the training process with another three convolutional layers for the 11-layer CNN configuration. On the contrary, our DSN of 11 layers behaves regularly during training and the objective function moves down directly starting at the first few epochs without pre-training; Table 5 shows the performance advantage of DSN over CNN, in addition to a significant gain in training convenience and saved manual adjustments.

4 Conclusions

In this paper, we have presented a new formulation, deeply-supervised nets (DSN), attempting to make a more transparent learning process for deep learning. Evident performance enhancement over existing approaches has been obtained. A stochastic gradient view also sheds light to the understanding of our formulation.

Acknowledgments

This work is supported by NSF award IIS-1216528 (IIS-1360566) and NSF award IIS-0844566(IIS-1360568). We thank Min Lin, Naiyan Wang, Baoyuan Wang, Jingdong Wang, Liwei Wang, Ying

Nian Wu , and David Wipf for help discussions. We gratefully acknowledge the support of NVIDIA Corporation with the donation of the GPUs used for this research.

References

[1] Y. Bengio, P. Lamblin, D. Popovici, H. Larochelle, U. D. Montral, and M. Qubec. Greedy layer-wise training of deep networks. In *NIPS*, 2007.

[2] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, June 2010.

[3] L. Bottou. Online algorithms and stochastic approximations. *Cambridge University Press*, 1998.

[4] G. E. Dahl, D. Yu, L. Deng, and A. Acero. Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *IEEE Tran. on Audio, Speech, and Lang. Proc.*, 20(1):30–42, 2012.

[5] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In *arXiv*, 2013.

[6] D. Eigen, J. Rolfe, R. Fergus, and Y. LeCun. Understanding deep architectures using a recursive convolutional network. In *arXiv:1312.1847v2*, 2014.

[7] J. L. Elman. Distributed representations, simple recurrent networks, and grammatical. *Machine Learning*, 7:195–225, 1991.

[8] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AISTAT*, 2010.

[9] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. C. Courville, and Y. Bengio. Maxout networks. In *ICML*, 2013.

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

[11] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. In *CoRR, abs/1207.0580*, 2012.

[12] F. J. Huang and Y. LeCun. Large-scale learning with svm and convolutional for generic object categorization. In *CVPR*, 2006.

[13] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun. What is the best multi-stage architecture for object recognition? In *ICCV*, 2009.

[14] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

[15] Q. Le, J. Ngiam, Z. Chen, D. Chia, P. W. Koh, and A. Ng. Tiled convolutional neural networks. In *NIPS*, 2010.

[16] W. Li, M. Zeiler, S. Zhang, Y. LeCun, and R. Fergus. Regularization of neural networks using dropconnect. In *ICML*, 2013.

[17] M. Lin, Q. Chen, and S. Yan. Network in network. In *ICLR*, 2014.

[18] P.-L. Loh and M. J. Wainwright. Regularized m-estimators with nonconvexity : statistical and algorithmic theory for local optima. In *arXiv:1305.2436v1*, 2013.

[19] R. Pascanu, T. Mikolov, and Y. Bengio. On the difficulty of training recurrent neural networks. In *arXiv:1211.5063v2*, 2014.

[20] A. Rakhlin, O. Shamir, and K. Sridharan. Making gradient descent optimal for strongly convex stochastic optimization. In *ICML*, 2012.

[21] J. Schmidhuber. Multi-column deep neural networks for image classification. In *CVPR*, 2012.

[22] O. Shamir and T. Zhang. Stochastic gradient descent for non-smooth optimization: Convergence results and optimal averaging schemes. In *ICML*, 2013.

[23] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *arXiv:1409.1556*, Sept. 4, 2014.

[24] J. Snoek, R. P. Adams, and H. Larochelle. Nonparametric guidance of autoencoder representations using label information. *J. of Machine Learning Research*, 13:2567–2588, 2012.

[25] N. Srivastava and R. Salakhutdinov. Discriminative transfer learning with tree-based priors. In *NIPS*, 2013.

[26] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *arXiv:1409.4842*, Sept. 17, 2014.

[27] Y. Tang. Deep learning using linear support vector machines. In *Workshop on Representational Learning, ICML*, 2013.

[28] J. Weston and F. Ratle. Deep learning via semi-supervised embedding. In *ICML*, 2008.

[29] M. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *arXiv 1311.2901*, 2013.

[30] M. D. Zeiler and R. Fergus. Stochastic pooling for regularization of deep convolutional neural networks. In *ICLR*, 2013.