6

Deep Stacking Networks and Variants — Supervised Learning

6.1 Introduction

While the DNN just reviewed has been shown to be extremely powerful in connection with performing recognition and classification tasks including speech recognition and image classification, training a DNN has proven to be difficult computationally. In particular, conventional techniques for training DNNs at the fine tuning phase involve the utilization of a stochastic gradient descent learning algorithm, which is difficult to parallelize across machines. This makes learning at large scale nontrivial. 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 or thousands of hours of speech training data with remarkable results. It is less clear, however, how to scale up this success with much more training data. See [69] for recent work in this direction.

Here we describe a new deep learning architecture, the deep stacking network (DSN), which was originally designed with the learning scalability problem in mind. This chapter is based in part on the recent publications of [106, 110, 180, 181] with expanded discussions.

The central idea of the DSN design relates to the concept of stacking, as proposed and explored in [28, 44, 392], where simple modules of functions or classifiers are composed first and then they are "stacked" on top of each other in order to learn complex functions or classifiers. Various ways of implementing stacking operations have been developed in the past, typically making use of supervised information in the simple modules. The new features for the stacked classifier at a higher level of the stacking architecture often come from concatenation of the classifier output of a lower module and the raw input features. In [60], the simple module used for stacking was a conditional random field (CRF). This type of deep architecture was further developed with hidden states added for successful natural language and speech recognition applications where segmentation information is unknown in the training data [429]. Convolutional neural networks, as in [185], can also be considered as a stacking architecture but the supervision information is typically not used until in the final stacking module.

The DSN architecture was originally presented in [106] and was referred as deep convex network or DCN to emphasize the convex nature of a major portion of the algorithm used for learning the network. The DSN makes use of supervision information for stacking each of the basic modules, which takes the simplified form of multilayer perceptron. In the basic module, the output units are linear and the hidden units are sigmoidal nonlinear. The linearity in the output units permits highly efficient, parallelizable, and closed-form estimation (a result of convex optimization) for the output network weights given the hidden units' activities. Due to the closed-form constraints between the input and output weights, the input weights can also be elegantly estimated in an efficient, parallelizable, batch-mode manner, which we will describe in some detail in Section 6.3.

The name "convex" used in [106] accentuates the role of convex optimization in learning the output network weights given the hidden units' activities in each basic module. It also points to the importance of the closed-form constraints, derived from the convexity, between the input and output weights. Such constraints make the learning of the remaining network parameters (i.e., the input network weights) much easier than otherwise, enabling batch-mode learning of the DSN that

can be distributed over CPU clusters. And in more recent publications, the DSN was used when the key operation of stacking is emphasized.

6.2 A basic architecture of the deep stacking network

A DSN, as shown in Figure 6.1, includes a variable number of layered modules, wherein each module is a specialized neural network consisting of a single hidden layer and two trainable sets of weights. In Figure 6.1, only four such modules are illustrated, where each module is shown with a separate color. In practice, up to a few hundreds of modules have been efficiently trained and used in image and speech classification experiments.

The lowest module in the DSN comprises a linear layer with a set of linear input units, a hidden nonlinear layer with a set of nonlinear units, and a second linear layer with a set of linear output units. A sigmoidal nonlinearity is typically used in the hidden layer. However, other nonlinearities can also be used. If the DSN is utilized in connection with recognizing an image, the input units can correspond to a number of pixels (or extracted features) in the image, and can be assigned values based at least in part upon intensity values, RGB values, or the like corresponding to the respective pixels. If the DSN is utilized in connection with speech recognition, the set of input units may correspond to samples of speech waveform, or the extracted features from speech waveforms, such as power spectra or cepstral coefficients. The output units in the linear output layer represent the targets of classification. For instance, if the DSN is configured to perform digit recognition, then the output units may be representative of the values 0, 1, 2, 3, and so forth up to 9 with a 0–1 coding scheme. If the DSN is configured to perform speech recognition, then the output units may be representative of phones, HMM states of phones, or context-dependent HMM states of phones.

The lower-layer weight matrix, which we denote by $\boldsymbol{W}$, connects the linear input layer and the hidden nonlinear layer. The upper-layer weight matrix, which we denote by $\boldsymbol{U}$, connects the nonlinear hidden layer with the linear output layer. The weight matrix $\boldsymbol{U}$ can be determined through a closed-form solution given the weight matrix $\boldsymbol{W}$ when the mean square error training criterion is used.

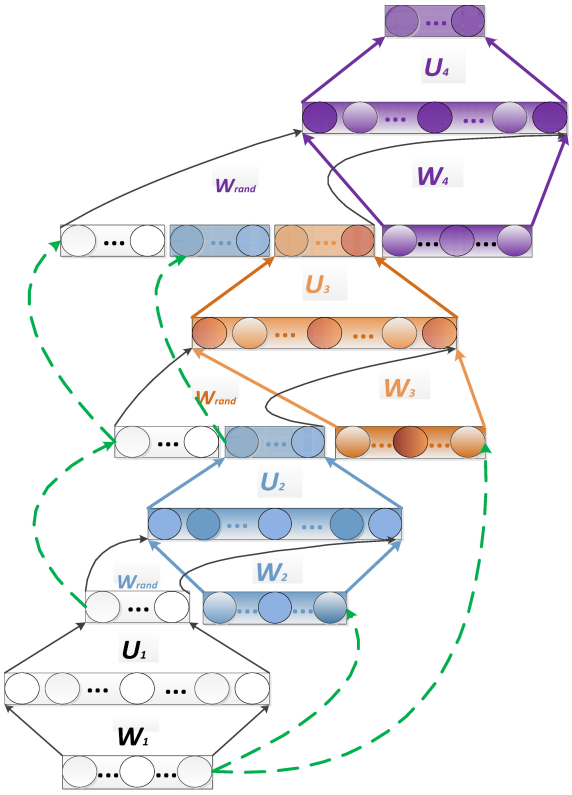

Figure 6.1: A DSN architecture using input–output stacking. Four modules are illustrated, each with a distinct color. Dashed lines denote copying layers. [after [366], @IEEE].

As indicated above, the DSN includes a set of serially connected, overlapping, and layered modules, wherein each module has the same architecture — a linear input layer followed by a nonlinear hidden layer, which is connected to a linear output layer. Note that the output units of a lower module are a subset of the input units of an adjacent higher module in the DSN. More specifically, in a second module that is directly above the lowest module in the DSN, the input units can include the output units of the lowest module and optionally the raw input feature.

This pattern of including output units in a lower module as a portion of the input units in an adjacent higher module and thereafter

learning a weight matrix that describes connection weights between hidden units and linear output units via convex optimization can continue for many modules. A resultant learned DSN may then be deployed in connection with an automatic classification task such as frame-level speech phone or state classification. Connecting the DSN's output to an HMM or any dynamic programming device enables continuous speech recognition and other forms of sequential pattern recognition.

6.3 A method for learning the DSN weights

Here, we provide some technical details on how the use of linear output units in the DSN facilitates the learning of the DSN weights. A single module is used to illustrate the advantage for simplicity reasons. First, it is clear that the upper layer weight matrix $\boldsymbol{U}$ can be efficiently learned once the activity matrix $\boldsymbol{H}$ over all training samples in the hidden layer is known. Let's denote the training vectors by $\boldsymbol{X} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_i, \ldots, \boldsymbol{x}_N]$, in which each vector is denoted by $\boldsymbol{x}_i = [x_{1i}, \ldots, x_{ji}, \ldots, x_{Di}]^T$ where D is the dimension of the input vector, which is a function of the block, and N is the total number of training samples. Denote by L the number of hidden units and by C the dimension of the output vector. Then the output of a DSN block is $\boldsymbol{y}_i = \boldsymbol{U}^T \boldsymbol{h}_i$ where $\boldsymbol{h}_i = \sigma(\boldsymbol{W}^T \boldsymbol{x}_i)$ is the hidden-layer vector for sample i, $\boldsymbol{U}$ is an $L \times C$ weight matrix at the upper layer of a block. $\boldsymbol{W}$ is a $D \times L$ weight matrix at the lower layer of a block, and $\sigma(\cdot)$ is a sigmoid function. Bias terms are implicitly represented in the above formulation if $\boldsymbol{x}_i$ and $\boldsymbol{h}_i$ are augmented with ones.

Given target vectors in the full training set with a total of N samples, $\boldsymbol{T} = [\boldsymbol{t}_1, \ldots, \boldsymbol{t}_i, \ldots, \boldsymbol{t}_N]$, where each vector is $\boldsymbol{t}_i = [t_{1i}, \cdots, t_{ji}, \ldots, t_{Ci}]^{\mathrm{T}}$, the parameters $\boldsymbol{U}$ and $\boldsymbol{W}$ are learned so as to minimize the average of the total square error below:

$$E = \frac{1}{2} \sum_i \|\boldsymbol{y}_i - \boldsymbol{t}_i\|^2 = \frac{1}{2} \mathrm{Tr}[(\boldsymbol{Y} - \boldsymbol{T})(\boldsymbol{Y} - \boldsymbol{T})^{\mathrm{T}}]$$

where the output of the network is

$$\boldsymbol{y}_i = \boldsymbol{U}^{\mathrm{T}} \boldsymbol{h}_i = \boldsymbol{U}^{\mathrm{T}} \sigma(\boldsymbol{W}^{\mathrm{T}} \boldsymbol{x}_i) = G_i(\boldsymbol{U}\boldsymbol{W})$$

which depends on both weight matrices, as in the standard neural net. Assuming $\boldsymbol{H} = [\boldsymbol{h}_1, \ldots, \boldsymbol{h}_i, \ldots, \boldsymbol{h}_N]$ is known, or equivalently, $\boldsymbol{W}$ is known. Then, setting the error derivative with respective to $\boldsymbol{U}$ to zero gives

$$\boldsymbol{U} = (\boldsymbol{H}\boldsymbol{H}^{\mathrm{T}})^{-1}\boldsymbol{H}\boldsymbol{T}^{\mathrm{T}} = \mathrm{F}(\boldsymbol{W}), \quad \text{where } \boldsymbol{h}_i = \sigma(\boldsymbol{W}^{\mathrm{T}}\boldsymbol{x}_i).$$

This provides an explicit constraint between $\boldsymbol{U}$ and $\boldsymbol{W}$ which were treated independently in the conventional backpropagation algorithm.

Now, given the equality constraint $\boldsymbol{U} = F(\boldsymbol{W})$, let's use Lagrangian multiplier method to solve the optimization problem in learning $\boldsymbol{W}$ Optimizing the Lagrangian:

$$E = \frac{1}{2}\sum_i \|G_i(\boldsymbol{U}, \boldsymbol{W}) - \boldsymbol{t}_i\|^2 + \lambda\|\mathrm{U} - \mathrm{F}(\boldsymbol{W})\|$$

we can derive batch-mode gradient descent learning algorithm where the gradient takes the following form [106, 413]:

$$\frac{\partial E}{\partial \boldsymbol{W}} = \mathbf{2}\boldsymbol{X}[\boldsymbol{H}^{\mathrm{T}} \circ (\mathbf{1} - \boldsymbol{H})^{\mathrm{T}} \circ [\boldsymbol{H}^{\dagger}(\boldsymbol{H}\boldsymbol{T}^{\mathrm{T}})(\boldsymbol{T}\boldsymbol{H}^{\dagger}) - \boldsymbol{T}^{\mathrm{T}}(\boldsymbol{T}\boldsymbol{H}^{\dagger})]],$$

where $\boldsymbol{H}^{\dagger} = \boldsymbol{H}^{\mathrm{T}}(\boldsymbol{H}\boldsymbol{H}^{\mathrm{T}})^{-\mathbf{1}}$ is pseudo-inverse of $\boldsymbol{H}$ and symbol $\circ$ denotes element-wise multiplication.

Compared with conventional backpropagation, the above method has less noise in gradient computation due to the exploitation of the explicit constraint $\boldsymbol{U} = F(\boldsymbol{W})$. As such, it was found experimentally that, unlike backpropagation, batch training is effective, which aids parallel learning of the DSN.

6.4 The tensor deep stacking network

The above DSN architecture has recently been generalized to its tensorized version, which we call the tensor DSN (TDSN) [180, 181]. It has the same scalability as the DSN in terms of parallelizability in learning, but it generalizes the DSN by providing higher-order feature interactions missing in the DSN.

The architecture of the TDSN is similar to that of the DSN in the way that stacking operation is carried out. That is, modules of the

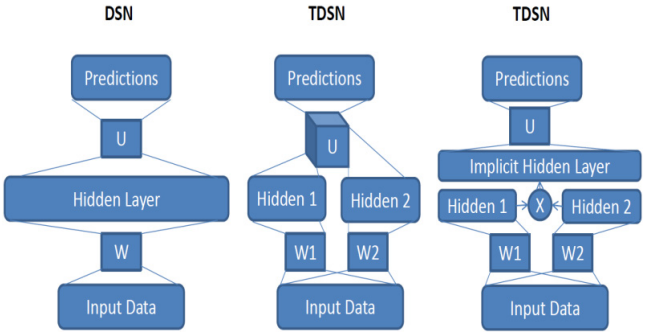

Figure 6.2: Comparisons of a single module of a DSN (left) and that of a tensor DSN (TDSN). Two equivalent forms of a TDSN module are shown to the right. [after [180], @IEEE].

TDSN are stacked up in a similar way to form a deep architecture. The differences between the TDSN and the DSN lie mainly in how each module is constructed. In the DSN, we have one set of hidden units forming a hidden layer, as denoted at the left panel of Figure 6.2. In contrast, each module of a TDSN contains two independent hidden layers, denoted as "Hidden 1" and "Hidden 2" in the middle and right panels of Figure 6.2. As a result of this difference, the upper-layer weights, denoted by "**U**" in Figure 6.2, changes from a matrix (a two dimensional array) in the DSN to a tensor (a three dimensional array) in the TDSN, shown as a cube labeled by "**U**" in the middle panel.

The tensor **U** has a three-way connection, one to the prediction layer and the remaining to the two separate hidden layers. An equivalent form of this TDSN module is shown in the right panel of Figure 6.2, where the implicit hidden layer is formed by expanding the two separate hidden layers into their outer product. The resulting large vector contains all possible pair-wise products for the two sets of hidden-layer vectors. This turns tensor **U** into a matrix again whose dimensions are (1) size of the prediction layer; and (2) product of the two hidden layers' sizes. Such equivalence enables the same convex optimization for learning **U** developed for the DSN to be applied to learning tensor **U**. Importantly, higher-order hidden feature interactions are enabled in the TDSN via the outer product construction for the large, implicit hidden layer.

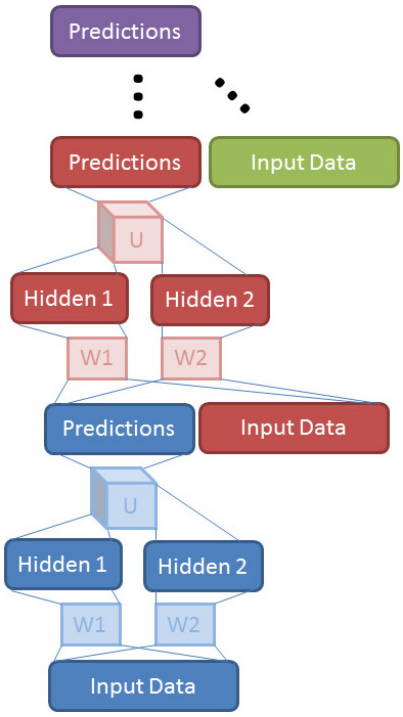

Figure 6.3: Stacking of TDSN modules by concatenating prediction vector with input vector. [after [180], @IEEE].

Stacking the TDSN modules to form a deep architecture pursues in a similar way to the DSN by concatenating various vectors. Two examples are shown in Figures 6.3 and 6.4. Note stacking by concatenating hidden layers with input (Figure 6.4) would be difficult for the DSN since its hidden layer tends to be too large for practical purposes.

6.5 The Kernelized deep stacking network

The DSN architecture has also recently been generalized to its kernelized version, which we call the kernel-DSN (K-DSN) [102, 171]. The motivation of the extension is to increase the size of the hidden units in each DSN module, yet without increasing the size of the free parameters to learn. This goal can be easily accomplished using the kernel trick, resulting in the K-DSN which we describe below.

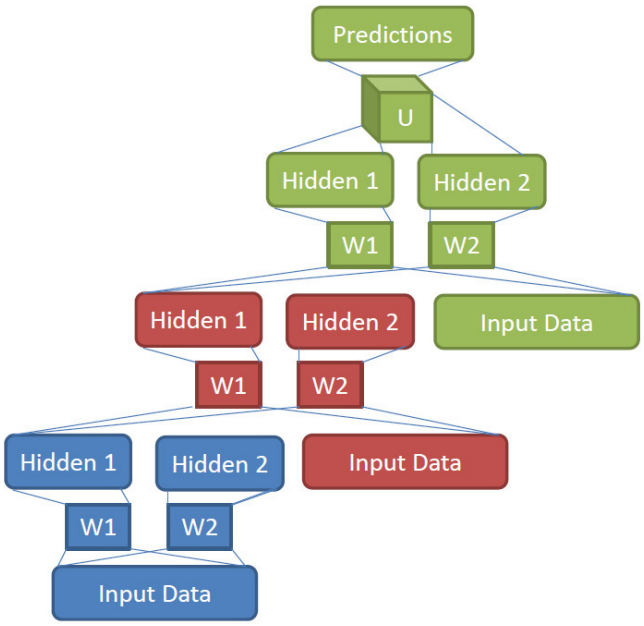

Figure 6.4: Stacking of TDSN modules by concatenating two hidden-layers' vectors with the input vector.

In the DSN architecture reviewed above optimizing the weight matrix $\boldsymbol{U}$ given the hidden layers' outputs in each module is a convex optimization problem. However, the problem of optimizing weight matrix $\boldsymbol{W}$ and thus the whole network is nonconvex. In a recent extension of DSN, a tensor structure was imposed, shifting most of the nonconvex learning burden for $\boldsymbol{W}$ to the convex optimization of $\boldsymbol{U}$ [180, 181]. In the new K-DSN extension, we completely eliminate nonconvex learning for $\boldsymbol{W}$ using the kernel trick.

To derive the K-DSN architecture and the associated learning algorithm, we first take the bottom module of DSN as an example and generalize the sigmoidal hidden layer $\mathbf{h}_i = \sigma(\boldsymbol{W}^{\mathrm{T}}\boldsymbol{x}_i)$ in the DSN module into a generic nonlinear mapping function $\boldsymbol{G}(\boldsymbol{X})$ from the raw input feature $\boldsymbol{X}$, with high dimensionality in $\boldsymbol{G}(\boldsymbol{X})$ (possibly infinite) determined only implicitly by a kernel function to be chosen. Second,

we formulate the constrained optimization problem of

$$\begin{aligned} &\textit{minimize } \frac{1}{2}\mathrm{Tr}[\boldsymbol{E}\boldsymbol{E}^{\mathrm{T}}] + \frac{C}{2}\boldsymbol{U}^{\mathrm{T}}\boldsymbol{U} \\ &\textit{subject to } \boldsymbol{T} - \boldsymbol{U}^{\mathrm{T}}\boldsymbol{G}(\boldsymbol{X}) = \boldsymbol{E}. \end{aligned}$$

Third, we make use of dual representations of the above constrained optimization problem to obtain $\boldsymbol{U} = \mathbf{G}^{\mathrm{T}}\boldsymbol{a}$, where vector $\mathbf{a}$ takes the following form:

$$a = (C\boldsymbol{I} + \boldsymbol{K})^{-1}\boldsymbol{T}$$

and $\boldsymbol{K} = \boldsymbol{G}(\boldsymbol{X})\boldsymbol{G}^{\mathrm{T}}(\boldsymbol{X})$ is a symmetric kernel matrix with elements $K_{nm} = g^{\mathrm{T}}(x_n)g(x_m)$.

Finally, for each new input vector $\boldsymbol{x}$ in the test or dev set, we obtain the K-DSN (bottom) module's prediction as

$$y(x) = \boldsymbol{U}^{\mathrm{T}}\boldsymbol{g}(\boldsymbol{x}) = \mathbf{a}^{\mathrm{T}}\boldsymbol{G}(\boldsymbol{X})\boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{k}^{\mathrm{T}}(\boldsymbol{x})(C\boldsymbol{I} + \boldsymbol{K})^{-1}\boldsymbol{T},$$

where the kernel vector $\boldsymbol{k}(\boldsymbol{x})$ is so defined that its elements have values of $k_n(\boldsymbol{x}) = k(\boldsymbol{x}_n, \boldsymbol{x})$ in which $\boldsymbol{x}_n$ is a training sample and $\boldsymbol{x}$ is the current test sample.

For lth module in K-DCN where $l \geq 2$, the kernel matrix is modified to

$$\boldsymbol{K} = \boldsymbol{G}([\boldsymbol{X}|\,\boldsymbol{Y}^{(l-1)}|\,\boldsymbol{Y}^{(l-2)}|\ldots\boldsymbol{Y}^{(1)}])\boldsymbol{G}^{\mathrm{T}}([\boldsymbol{X}|\,\boldsymbol{Y}^{(l-1)}|\,\boldsymbol{Y}^{(l-2)}|\ldots\boldsymbol{Y}^{(1)}]).$$

The key advantages of K-DSN can be analyzed as follows. First, unlike DSN which needs to compute hidden units' output, the K-DSN does not need to explicitly compute hidden units' output $\boldsymbol{G}(\boldsymbol{X})$ or $\boldsymbol{G}([\boldsymbol{X}|\,\boldsymbol{Y}^{(l-1)}|\,\boldsymbol{Y}^{(l-2)}|\ldots\boldsymbol{Y}^{(1)}])$. When Gaussian kernels are used, kernel trick equivalently gives us an infinite number of hidden units without the need to compute them explicitly. Further, we no longer need to learn the lower-layer weight matrix $\boldsymbol{W}$ in DSN as described in [102] and the kernel parameter (e.g., the single variance parameter σ in the Gaussian kernel) makes K-DSN much less subject to overfitting than DSN. Figure 6.5 illustrates the basic architecture of a K-DSN using the Gaussian kernel and using three modules.

The entire K-DSN with Gaussian kernels is characterized by two sets of module-dependent hyper-parameters: $\sigma^{(l)}$ and $C^{(l)}$ the kernel

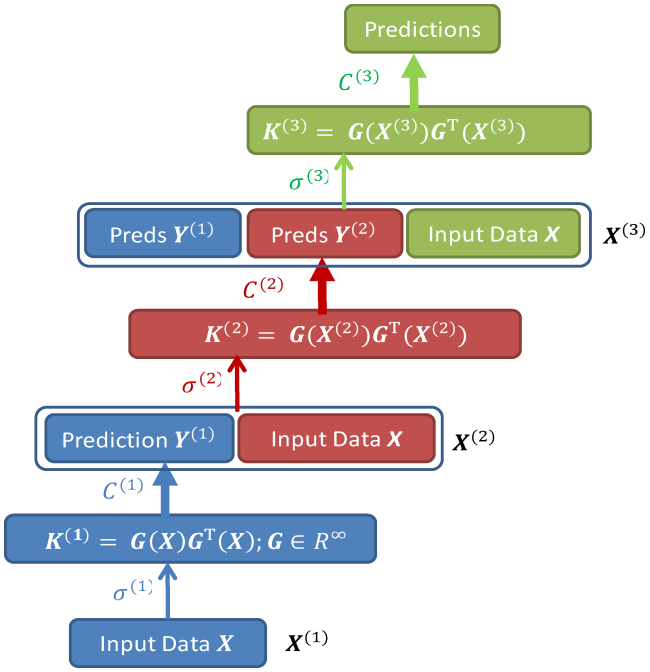

Figure 6.5: An example architecture of the K-DSN with three modules each of which uses a Gaussian kernel with different kernel parameters. [after [102], @IEEE].

smoothing parameter and regularization parameter, respectively. While both parameters are intuitive and their tuning (via line search or leave-one-out cross validation) is straightforward for a single bottom module, tuning the full network with all the modules is more difficult. For example, if the bottom module is tuned too well, then adding more modules would not benefit much. In contrast, when the lower modules are loosely tuned (i.e., relaxed from the results obtained from straightforward methods), the overall K-DSN often performs much better. The experimental results reported by Deng et al. [102] are obtained using a set of empirically determined tuning schedules to adaptively regularize the K-DSN from bottom to top modules.

The K-DSN described here has a set of highly desirable properties from the machine learning and pattern recognition perspectives. It combines the power of deep learning and kernel learning in a principled

way and unlike the basic DSN there is no longer nonconvex optimization problem involved in training the K-DSN. The computation steps make the K-DSN easier to scale up for parallel computing in distributed servers than the DSN and tensor-DSN. There are many fewer parameters in the K-DSN to tune than in the DSN, T-DSN, and DNN, and there is no need for pre-training. It is found in the study of [102] that regularization plays a much more important role in the K-DSN than in the basic DSN and Tensor-DSN. Further, effective regularization schedules developed for learning the K-DSN weights can be motivated by intuitive insight from useful optimization tricks such as the heuristic in Rprop or resilient backpropagation algorithm [302].

However, as inherent in any kernel method, the scalability becomes an issue also for the K-DSN as the training and testing samples become very large. A solution is provided in the study by Huang et al. [171], based on the use of random Fourier features, which possess the strong theoretical property of approximating the Gaussian kernel while rendering efficient computation in both training and evaluation of the K-DSN with large training samples. It is empirically demonstrated that just like the conventional K-DSN exploiting rigorous Gaussian kernels, the use of random Fourier features also enables successful stacking of kernel modules to form a deep architecture.