This is a draft containing only sra_chapter.tex and an abbreviated front matter. Please check that the formatting and small changes have been performed correctly. Please verify the affiliation. Please use this version for sending us future modifications.

The MIT Press
Cambridge, Massachusetts
London, England
Contents

1 Connecting Deep Learning Features to Log-Linear Models 1
  1.1 Introduction .................................................. 2
  1.2 A Framework of Using Deep Learning Features for Classification 5
  1.3 DSN and Kernel-DSN Features for Log-Linear Classifiers ... 9
  1.4 DNN Features for Conditional-Random-Field Sequence Classifiers .................................................. 18
  1.5 Log-Linear Stacking to Combine Multiple Deep Learning Systems .................................................. 20
  1.6 Discussion and Conclusions ................................. 21
A log-linear model by itself is a shallow architecture given fixed, non-adaptive, human-engineered feature functions but its flexibility in using the feature functions allows the exploitation of diverse high-level features computed automatically from deep learning systems. We propose and explore a paradigm of connecting the deep learning features as inputs to log-linear models, which, in combination with the feature hierarchy, form a powerful deep classifier. Three case studies are provided in this chapter to instantiate this paradigm. First, deep stacking networks and its kernel version are used to provide deep learning features for a static log-linear model — the softmax classifier or maximum entropy model. Second, deep-neural-network features are extracted to feed to a sequential log-linear model — the conditional random field. And third, a log-linear model is used as a stacking-based assemble learning machine to integrate a number of deep learning systems’ outputs. All these three types of deep classifier have their effectiveness verified in experiments. Finally, compared with the traditional log-linear modeling approach which relies on human feature engineering, we point out one main weakness of the new framework in its lack of ability to naturally embed domain knowledge. Future directions are discussed for overcoming this weakness by integrating deep neural networks with deep generative models.
1.1 Introduction

Log-linear modeling forms the basis of a class of important machine learning methods that have found wide applications, notably in human language technology including speech and natural language processing (Lafferty et al., 2001; Mann and McCallum, 2008; Heigold et al., 2011; Jiao et al., 2006; He and Deng, 2011, 2013). In mathematical terms, a log-linear model takes the form of a function whose logarithm is a linear function of the model parameters:

\[ Ce^{\sum_i w_i f_i(x)} \text{ or } Ce^{w^T f(x)} \]  

(1.1)

where \( f_i(x) \) are functions of the input data variables \( x \), in general a vector of values, and \( w_i \) are model parameters (the bias term can be absorbed by introducing an additional “feature” of \( f_i(x) = 1 \)). Here, \( C \) does not depend on the model parameters \( w_i \) but may be a function of data \( x \).

A special form of the log-linear model of Eq. (1.1) is softmax function, which has the following form that is often used to model the class-posterior distribution for classification problems with multiple (\( K > 2 \)) classes:

\[ P(y = j | x) = \frac{e^{s_j}}{\sum_{k=1}^{K} e^{s_k}}, \quad \text{where } s_j = w_j^T x, \quad j = 1, 2, ..., K \]  

(1.2)

The reason for calling this function “softmax” is that the effect of exponentiating the \( K \) values of \( s_1, s_2, ..., s_K \) in the exponents of Eq.(1.2) is to exaggerate the differences between them. As a result, the softmax function will return a value close to zero whenever \( s_j \) is significantly less than the maximum of all the values, and will return a value close to one when applied to the maximum value, unless it is extremely close to the next-largest value. Therefore, softmax can be used to construct a weighted average that behaves as a smooth function to approximate the non-smooth function \( \max(s_1, s_2, ..., s_K) \).

The classifiers exploiting the softmax function of Eq.(1.2) are often called softmax regression (or classifier), multinomial (or multiclass) logistic regression, multinomial regression (or logit), maximum entropy (MaxEnt) classifier, or conditional maximum entropy model. This is a class of popular classification methods in statistics, machine learning, and in speech and language processing that generalize logistic regression from two-class to multi-class problems. Such classifiers can be used to predict the probabilities of the different possible outcomes of a categorically distributed dependent variable, given a set of independent variables which may be real-valued, binary-valued, categorical-valued, etc. Extending the softmax classifier for static patterns
to sequential patterns, we have the conditional random field (CRF) as a more general case of log-linear classifiers (Lafferty et al., 2001; Mann and McCallum, 2008; Yu and Deng, 2010).

Log-linear models are interesting from several points of view. First, many popular generative models (Gaussian models, word-counting models, part-of-speech tagging, etc.) in speech and language processing have the posterior form that is shown to be equivalent to log-linear models (Heigold et al., 2011). This provides important insight into the connections between generative and discriminative learning paradigms which have been commonly regarded as two separate classes of approaches (Deng and Li, 2013). Second, the natural training criterion of log-linear models is the logarithm of the class posterior probability or conditional maximum likelihood. This gives rise to a convex optimization problem and the margin concept can be built into the training. Third, log-linear models can be extended to include hidden variables, thereby connecting naturally to the common generative Gaussian mixture and hidden Markov models (Gunawardana et al., 2005; Yu et al., 2009).

Perhaps most importantly, the softmax classifier as an instance of the log-linear model has been used extensively in recent years as the top layer in deep neural networks (DNNs) that consist of many other (lower) layers as well (Yu et al., 2010; Dahl et al., 2011; Seide et al., 2011a; Morgan, 2012; Sainath et al., 2013; Yu et al., 2013a; Mohamed et al., 2012b,a; Deng et al., 2013a,b). The CRF sequence classifier has also been used in a similar fashion, i.e., being connected to a DNN whose parameters are tied across the entire sequence (Mohamed et al., 2010; Kingsbury et al., 2012). The conventional way of interpreting the sweeping success of the DNN in speech recognition, from small to large tasks and from laboratory experiments to industrial deployments, is that the DNN is learned discriminatively in an end-to-end manner (Seide et al., 2011b; Hinton et al., 2012; Yu et al., 2013b; Deng and Yu, 2014). The excellent scalability of the DNN and its huge capacity provided by massive weight parameters and distributed representations (Deng and Togneri, 2014) have enabled its success. In this chapter, we provide a new way of looking at the DNN and other deep learning models in terms of their ability to provide high-level features to relatively simple classifiers such as log-linear models, rather than viewing an entire deep learning model as a complex classifier. That is, we connect deep learning to log-linear classifiers via the separation of feature extraction and classification.

In the early days of speech recognition research when shallow models such as Gaussian mixture model (GMM) and hidden Markov model (HMM) were exploited (Rabiner, 1989; Baker et al., 2009a,b), integrated learning of
classifiers and feature extractors was shown to outperform that when the two stages are separated (Biem et al., 2001; Chengalvarayan and Deng, 1997a,b). Some earlier successes of the DNN in speech recognition also adopted the integrated or end-to-end learning via backpropagating errors all the way from top to the bottom in the network (Mohamed et al., 2009). However, more recent successes of the DNN and more advanced deep models have shown that using the deep models to provide features for separate classifiers has numerous advantages over integrated learning (Tuske et al., 2012; Yan et al., 2013; Deng and Chen, 2014). For example, this feature-based approach enables the use of existing GMM-HMM discriminative training methods and infrastructure developed and matured over many years (Yan et al., 2013), and it helps transfer or multitask deep learning (Ghoshal et al., 2013; Heigold et al., 2013; Huang et al., 2013). This type of large-scale discriminative training, unlike end-to-end training of the DNN by minibatch-based backpropagation, is naturally suited for batch-based parallel computation since it rests on extended Baum-Welch algorithm (Povey and Woodland, 2002; He et al., 2008; Wright et al., 2013). Also, speaker-adaptive (Anastasakos et al., 1996) and noise-adaptive training techniques (Deng et al., 2000; Kalinli et al., 2010; Flego and Gales, 2009) successfully developed for the GMM-HMM can be usefully applied. Further, making use of the features derived from the DNN, we can easily perform multi-task and transfer learning. Successful applications of this type have been shown in multilingual and mixed-bandwidth speech recognition (Heigold et al., 2013; Huang et al., 2013; ?), which had much less success in the past using the GMM-HMM approach (Lin et al., 2009). Another benefit of using deep learning features is that it avoids overfitting, especially when the depth of the network grows very large as in the deep stacking network (DSN) to be discussed in a later section. Moreover, when feature extraction performed by deep models is separated from the classification stage in the overall system, different types of deep learning methods can be more easily combined (Chen and Deng, 2014; Deng and Platt, 2014) and unsupervised learning methods (e.g., autoencoders) may be more naturally incorporated (Deng et al., 2010; Sainath et al., 2012; Le et al., 2012).

This chapter is aimed to explore the topic of log-linear modeling as supervised classifiers using features automatically derived from deep learning systems. Specifically, we focus on the use of log-linear models as the classifier, whose flexibility for accepting a wide range of features facilitates the use of deep learning features. In Section 2, a framework of using deep learning features for performing supervised learning tasks is developed with examples given in the context of acoustic modeling for speech recognition. Sections 3-5 provide three more detailed case studies on how this general framework...
is applied. The case study presented in Section 3 concerns the use a specialized deep learning architecture, the DSN and its kernel version, to compute the features for static log-linear or max-entropy classifiers. Some key implementation details and experimental results are included, not published in the previous literature. Section 4 shows how the standard DNN features can be interfaced to a dynamic log-linear model, the conditional random field, whose standard learning leads to the equivalent full-sequence learning for the DNN-HMM which gives the state of the art accuracy in large vocabulary speech recognition as of the writing of this chapter. The final case study reported in Section 5 makes use of the log-linear model as a stacking mechanism to perform ensemble learning, where three deep learning systems (deep forward and fully-connected neural network, deep convolutional neural network, and recurrent neural network) provide three separate streams of deep learning features for system combination in a log-linear fashion.

1.2 A Framework of Using Deep Learning Features for Classification

Basics of deep learning

Deep Learning is a class of machine learning techniques, where many layers of information processing stages in hierarchical architectures are exploited. The most prominent successes of deep learning, achieved in recent years, are in supervised learning for classification tasks (Hinton et al., 2012; Dahl et al., 2012; Krizhevsky et al., 2012). The essence of deep learning is to compute hierarchical features or representations of the observational data, where the higher-level features or factors are defined from lower-level ones. Recent overviews of deep learning can be found in (Bengio, 2009; Schmidhuber, 2014; Deng and Yu, 2014).

Given fixed feature functions \( f(x) \), the log-linear model defined in Eq.(1.1) is a shallow architecture. However, the flexibility of the log-linear model in using the feature functions (i.e., no restriction on the form of functions in \( f(x) \)) allows the model to exploit a wide range of high-level features including those computed from separate deep learning systems. In this section, we explore a general framework of connecting such deep leaning features as the input to the log-linear model. A combination of deep learning systems (e.g., the DNN) and the log-linear model gives rise to powerful deep architectures for classification tasks.

The GMM-HMM

To build up this unifying framework, let us start with the (shallow) architecture of the GMM-HMM and put the discussion in the specific context of acoustic modeling for speech recognition. Before around 2010-2011, speech recognition technology had been dominated by the HMM, where each state is characterized by the GMM; hence, the GMM-HMM (Rabiner, 1989; Deng
et al., 1991). In Figure 1.1, we illustrate the GMM-HMM system by showing how the raw speech feature vectors indexed by time frame $t$, $x_t$, such as Mel-Frequency Cepstral Coefficients (MFCCs) and Perceptual Linear Prediction (PLP) features, form a feature sequence to feed into the GMM-HMM as a sequence classifier, producing a sequence of linguistic symbols (e.g., words, phones, etc.) as the recognizer’s output.

![Figure 1.1: Illustration of the GMM-HMM-based speech recognition system: Feeding low-level speech sequences into the GMM-HMM sequence classifier.](image)

While significant technological successes had been achieved using complex and carefully engineered variants of GMM-HMMs and acoustic features suitable for them, researchers long before that time had clearly realized that the next generation of speech recognition technology would require solutions to many new technical challenges under diversified deployment environments and that overcoming these challenges would likely require “deep” architectures that can at least functionally emulate the human speech system known to have dynamic and hierarchical structure in both production and perception (Stevens, 2000; Divenyi et al., 2006; Deng and O’Shaughnessy, 2003; Deng et al., 1997; Bridle et al., 1998; Picone et al., 1999). An attempt to incorporate a primitive level of understanding of this deep speech structure, initiated at the 2009 NIPS Workshop on Deep Learning for Speech Recognition and Related Applications (Deng et al.,...
1.2 A Framework of Using Deep Learning Features for Classification

2009), has helped ignite the interest of the speech community in pursuing a deep representation learning approach based on the deep neural network (DNN) architecture. The generative pre-training method in effective learning of the DNN was pioneered by the machine learning community a few years earlier (Hinton and Salakhutdinov, 2006; Hinton et al., 2006). The DNN, learned both generatively or in a purely discriminative manner when large training data and big compute are available, has rapidly evolved into the new state of the art in speech recognition with pervasive industry-wide adoption (Dahl et al., 2012; Hinton et al., 2012; Deng and Yu, 2014).

The DNN by itself is a static classifier and does not handle variable-dimensional sequences as in the raw speech data. The DNN is shown in the left portion of Figure 1.2, where $h_{1,t}$, $h_{2,t}$, and $h_{3,t}$ illustrate three hidden state vectors of the DNN for the low-level speech feature vector $x_t$ at time $t$, and $y_t$ is the corresponding output vector of the DNN. Denoted by $d_t$ is the desired sequence of target vectors (often coded as sparse one-hot vectors) used for training the DNN using either the cross-entropy (CE) or maximum-mutual information (MMI) criteria. Now turn to the right portion of Figure 1.2. The high-level DNN features extracted from the $v_{3,t}$ layer is shown to feed into the sequence classifier (a log-linear model followed by an HMM), which produces symbolic linguistic sequences. The overall architecture shown in Figure 1.2 using DNN-derived features for a log-linear model followed by an HMM is called the DNN-HMM. The parameters of the overall DNN-HMM system can be learned either using backpropagation with the end-to-end style where the errors defined at the sequence classifier are propagated back all the way into all the hidden layers of the DNN, or propagating the errors into only the high-level feature level with the DNN parameters learned separately using the the error criterion defined at the DNN’s output layer as shown in the left portion of Figure 1.2. The former, end-to-end learning is likely to work better with large amounts of training data. And the latter, based on fixed, high-level features, is less prone to overfitting. The latter is also more effective in multi-task learning since the high-level features tend to transfer well from one learning task to another as demonstrated in multi-lingual speech recognition as we discussed in Section 1.

While the DNN-HMM architecture shown in Figure 1.2 produces much lower errors than the previous state-of-the-art GMM-HMM systems, it is only one of many possible deep architectures. For example, not just one hidden state vector in the DNN can serve as the high-level features for the log-linear sequence classifier, a combination of them, typically in a straightforward form of concatenation, can serve as more powerful DNN features as demonstrated in (Deng and Chen, 2014), which is shown in the
left portion of Figure 1.3. Further, the sequence classifier may not be limited to the log-linear model. Other sequence classifiers can take the high-level DNN features as their inputs also, which is shown in the right portion of Figure 1.3. Specifically, the use of GMM-HMM sequence classifiers for DNN-derived features is shown to almost match the low error rate produced by the DNN-HMM system (Yan et al., 2013).

In summary, in this section we present a general framework of using deep learning features for sequence classification, with the application examples drawn from speech recognition. This framework enables us to naturally connect the DNN features as the input to the log-linear model as the most prominent scheme for sequence classification. Importantly, as a special case of this framework, where the DNN feature is derived from the top hidden layer of the DNN and where the sequence classifier uses the softmax followed by an HMM, we recover the popular DNN-HMM architecture widely used in current state-of-the-art speech recognition systems. In the next three sections, we will provide three special cases of this framework, some aspects of which have been published in recent literature but can be better understood in a unified manner based on the framework discussed in
1.3 DSN and Kernel-DSN Features for Log-Linear Classifiers

In this section, we discuss the use of deep learning features computed from the Deep stacking networks (DSN) and its kernel version (K-DSN) for log-linear models. We first review the DSN and K-DSN and study their properties, and then exploit them as feature functions to connect to the log-linear models, where application case studies are provided.

1.3.1 Deep stacking networks (DSN)

Stacking methods

Stacking is a class of machine learning techniques that form combinations of different predictors’ outputs in order to give improved overall prediction accuracy, typically through improved generalization (Wolpert, 1992; Breiman, 1996). In (Cohen and de Carvalho, 2005), for example, stacking is applied...
to reduce the overall generalization error of a sequence of trained predictors by generating an unbiased set of previous predictions’ outputs for use in training each successive predictor.

This concept of stacking is more recently applied to construct a deep network where the output of a (shallow) network predictor module is used, in conjunction with the original input data, to serve as the new, expanded “inputs” for the next level of the network predictor module in the full, multiple-module network, which is called the Deep Stacking Network (DSN) (Deng and Yu, 2011; Deng et al., 2012b; Hutchinson et al., 2012). Figure 1.5 shows a DSN with three modules, each with a different color and each consisting of three layers with upper-layer weight matrix denoted by $\mathbf{U}$ and lower-layer weight matrix denoted by $\mathbf{W}$.

![Figure 1.4](image)

**Figure 1.4:** Illustration of a DSN with three modules, each with a different color. Each module consists of three layers connected by upper weight matrix denoted by $\mathbf{U}$ and lower weight matrix denoted by $\mathbf{W}$.

In the DSN, different modules are constructed somewhat differently. The lowest module comprises the following three layers. 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 \( \mathbf{x}_i = [x_{1i}, ..., x_{ji}, ..., x_{Di}]^T \). 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

\[
\mathbf{h}_i = \sigma(\mathbf{W}^T \mathbf{x}_i)
\]

(1.3)

as the hidden layer’s output, where \( \sigma(.) \) is the sigmoid function and \( \mathbf{W} \) 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 \( \mathbf{y}_i = \mathbf{U}^T \mathbf{h}_i \), where \( \mathbf{U} \) 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).

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

\[
D_m = D + C(m - 1), \quad m = 1, 2, \cdots, M
\]

(1.4)

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.

For each module, given \( \mathbf{W} \), learning \( \mathbf{U} \) is a convex optimization problem. The solution differs for separate 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. 1.4, before applying the learning technique presented below.

Learning DSN weight matrix \( \mathbf{U} \) given \( \mathbf{W} \)

In the supervised learning setting, both training data \( \mathbf{X} = [\mathbf{x}_1, ..., \mathbf{x}_i, ..., \mathbf{x}_N] \) and the corresponding labeled target vectors \( \mathbf{T} = [t_1, ..., t_i, ..., t_N] \), where
each target $\mathbf{t}_i = [t_{i1}, ..., t_{ji}, ..., t_{Ci}]^T$, 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:

$$E = Tr[(\mathbf{Y} - \mathbf{T})(\mathbf{Y} - \mathbf{T})^T],$$  \hspace{1cm} (1.5)

where $\mathbf{Y} = [y_1, ..., y_i, ..., y_N]$.

Importantly, if weight matrix $\mathbf{W}$ is determined already (e.g., via judicious initialization), then the hidden layer values $\mathbf{H} = [h_1, ..., h_i, ..., h_N]$ are also determined. Consequently, upper-layer weight matrix $\mathbf{U}$ in each module can be determined by setting the gradient

$$\frac{\partial E}{\partial \mathbf{U}} = 2\mathbf{H}(\mathbf{U}^T\mathbf{H} - \mathbf{T})^T \hspace{1cm} (1.6)$$

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

$$\mathbf{U} = (\mathbf{HH}^T)^{-1}\mathbf{HT}. \hspace{1cm} (1.7)$$

Combining Eqns. (1.3) and (1.7), we see that $\mathbf{U}$ is an explicit function of $\mathbf{W}$, denoted, say, as

$$\mathbf{U} = F(\mathbf{W}). \hspace{1cm} (1.8)$$

Note that Eq. (1.9) provides a powerful constraint when learning matrix weight matrix $\mathbf{W}$; i.e.

$$\hat{\mathbf{W}} = \arg\max_{\mathbf{W}} E(\mathbf{U}^*, \mathbf{W}), \hspace{0.5cm} subject \hspace{0.5cm} to \hspace{0.5cm} \mathbf{U}^* = F(\mathbf{W}). \hspace{1cm} (1.9)$$

When gradient descent method is used to optimize $\mathbf{W}$, we seek to compute the total derivative of the error function:

$$\frac{dE}{d\mathbf{W}} = \frac{\partial E}{\partial \mathbf{W}} + \frac{\partial E}{\partial \mathbf{U}^*} \frac{\partial \mathbf{U}^*}{\partial \mathbf{W}}. \hspace{1cm} (1.10)$$

This total derivative can be found in a direct analytical form (i.e., without recursion as in backpropagation). An easy way to pursue is to remove the constraint in Eq. (1.9) by substituting the constraint directly into the objective function, yielding the unconstrained optimization problem of

$$\hat{\mathbf{W}} = \arg\max_{\mathbf{W}} E[F(\mathbf{W}), \mathbf{W}]. \hspace{1cm} (1.11)$$

And the analytical form of the total derivative is derived below:
\[ \frac{dE}{dW} = \frac{dTr[(U^T H - T)(U^T H - T)^T]}{dW} \]
\[ = \frac{dTr[([HH^T]^{-1}HT^T)H - T)([HH^T]^{-1}HT^T)H - T)^T]}{dW} \]
\[ = \frac{dTr[TT^T - TH^T(HH^T)^{-1}HT^T]}{dW} \]
\[ = \frac{dTr[(HH^T)^{-1}HT^TTH^T]}{dW} \]
\[ = \frac{dTr[(\sigma(W^TX)[\sigma(W^TX)]^T)^{-1}\sigma(W^TX)T^T[\sigma(W^TX)]^T]}{dW} \]
\[ = 2X[H^T \circ (1 - H)^T \circ [H^T(HT^T)(TH^T) - T^T(TH^T)]] \] (1.12)

where \(H^i = H^T(HH^T)^{-1}\) and \(\circ\) denotes element-wise matrix multiplication. In deriving Eq.1.12, we used the fact that \(HH^T\) is symmetric and so is \((HH^T)^{-1}\).

Importantly, the total derivative in Eq.1.12 with respective to lower-layer weight matrix \(W\) is different from the gradient computed in the standard backpropagation algorithm which requires recursion through the hidden layer and which is partial derivative with respective to \(W\) instead of total derivative. That is, backpropagation algorithm gives only the first term \(\frac{\partial E}{\partial W}\) in Eq.1.10. So the difference between the gradients used in backpropagation algorithm and in the algorithm based on Eq.1.12 differs by the quantity of

\[ \frac{\partial E}{\partial U^*} \frac{\partial U^*}{\partial W}, \] (1.13)

where \(U^* = F(W) = (HH^T)^{-1}HT^T\) and each vector component of matrix \(H\) is \(h_i = \sigma(W^T x_i)\).

This difference may account for why learning the DSN using batch-mode gradient descent using the total derivative of Eq.1.12 is more effective than using batch-mode backpropagation based on partial derivative experimentally (Deng and Yu, 2011). It is noted that using the total derivative of Eq.1.12 is equivalent to coordinate descent algorithm with an “infinite” step size to achieve the global optimum along the “coordinate” of updating \(U\) while fixing \(W\).

Applications of the DSN features discussed in this section for log-linear classifiers will be presented in Section 4.
1.3.2 Kernel deep stacking networks (K-DSN)

It is well known that mapping raw speech, audio, text, image, and video data into desirable feature vectors for machine learning algorithms to consume typically require extensive human expertise, intuition, and domain knowledge. Kernel methods are a powerful set of tools that alleviate such difficult requirements via the kernel “trick”. Deep learning methods, on the other hand, avoid hand-designed resources and time-intensive feature engineering by adopting layer-by-layer generative or discriminative learning. Understanding the essence of these two styles of feature mapping and their connections is of both conceptual and practical significance. First, kernel methods can be naturally extended to overcome the linearity inherent in the pattern predictive functions. Second, deep learning methods can also be enhanced to let the effective hidden feature dimensionality grow to infinity without encountering otherwise computational and overfitting difficulties. Both of the extensions lead to integrated kernel and deep learning architectures, expect to perform better in practical applications. One such architecture, which integrates kernel trick in the DSN and is called Kernel-DSN or K-DSN (Deng et al., 2012a), is discussed in this section, where insights into how the generally finite-dimensional (hidden) features in the DSN can be transformed into infinite-dimensional features via kernel trick without incurring computational and regularization difficulty are elaborated.

The DSN architecture discussed in the preceding section has convex learning for weight matrix $U$ given the hidden layers outputs in each module, but the learning of weight matrix $W$ is non-convex. For most applications, the size of $U$ is comparable to that of $W$ and then DSN is not strictly a convex network. In a recent extension of DSN, a tensor structure was imposed, shifting the majority of non-convex learning burden for $W$ into a convex one (Hutchinson et al., 2012, 2013). In the K-DSN extension discussed here, non-convex learning for $W$ is completely eliminated using kernel trick. In deriving the K-DSN architecture and the associated learning algorithm, the sigmoidal hidden layer $h_i = \sigma(W^T x_i)$ in a DSN module is generalized into a generic nonlinear mapping function $G(X)$ from raw input features $X$. The high or possibly infinite dimensionality in $G(X)$ is determined only implicitly by the kernel function.

It can be derived that for each new input vector $x$ in the test set, a (bottom) module of the K-DSN has the prediction function of

$$y(x) = k^T(x)(C I + K)^{-1} T$$

(1.14)

where $T = [t_1, ..., t_i, ..., t_N]$ are the target vectors for training, and the kernel vector $k(x)$ is defined such that its elements have values of $k_n(x) = k(x_n, x)$.
in which \(x_n\) is a training sample and \(x\) is the current test sample. For the non-bottom DSN module of \(l \geq 2\), the same prediction function hold except the kernel matrix is expanded to

\[
K^{(l)} = G \left( [X|Y^{(l-1)}|Y^{(l-2)}|...|Y^{(1)}] \right) G^T \left( [X|Y^{(l-1)}|Y^{(l-2)}|...|Y^{(1)}] \right)
\]

Comparing the prediction in the DSN and that of the K-DSN, we can identify key advantages of the K-DSN. Unlike the DSN which need to compute hidden units outputs, the K-DSN does not need to explicitly compute hidden units outputs. Let’s take the example of Gaussian kernel, where kernel trick gives an equivalent of an infinite number of hidden units without the need to compute them explicitly. Further, one no longer needs to learn the lower-layer weight matrix \(W\) as in the DSN, and the kernel parameter (e.g., the single variance parameter \(\sigma\) in the Gaussian kernel) makes the K-DSN much less subject to overfitting.

The architecture of a K-DSN using Gaussian kernel is shown in Figure 1.5, where the entire network is characterized by two module \((l)\)-dependent hyper-parameters \((l = 1, 2, ..., L)\):

- \(\sigma^{(l)}\), which is the kernel smoothing parameter, and
- \(C^{(l)}\), which is the regularization parameter.

While both of these parameters are intuitive and their tuning (via line search or leave-one-out cross validation) is straightforward for a single bottom module, tuning them from module to module is more difficult. It was found in experiments that 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. In practice, these hyperparameters can be determined using empirical tuning schedules (e.g., RPROP-like method (Riedmiller and Braun, 1993)) to adaptively regularize the K-DSN from bottom to top modules.

### 1.3.3 Connecting DSN and K-DSN features to classification models

Based on the general framework of using deep learning features for classification as presented in Section 2, we now describe in this section selected issues of importance and related experiments on creating features out of the DSN and K-DSN discussed earlier for a separate classifier (e.g., a log-linear model) for classification tasks.

Some earlier experimental results using DSNs for MNIST digit recognition (with no elastic distortion) and TIMIT phone recognition tasks were
published in (Deng et al., 2012b). Here some key aspects of the DSN design are summarized including two unpublished aspects of the design and also including adding a softmax classifier on top of the DSN. Related new experimental results are then reported in this section.

Given the basic DSN design described earlier in this section, an important aspect is how to initialize the weight matrix in each of the DSN module. It was reported in (Deng et al., 2012b) that initialization using a standard RBM (i.e., with symmetric up- and down-weights) gives a significant accuracy gain over random initialization. If we break this symmetry by using asymmetric weights in the RBM, where the up-weights are constrained to be a fixed factor of the down-weights, then better results can be obtained. The scaled factor is determined by the square root of the ratio of the number of RBM hidden units over that of visible units. The motivation is that with different numbers of units in the hidden and visible layers, the equal-weight constraint in the RBM would not balance the average activities of the two layers.

We can further improve classification accuracy of a DSN by regularizing its learning by incorporating reconstruction errors in the gradient computation.
Table 1.1: Classification error rates using various versions of the DSN in the standard MNIST task (no training data augment with elastic distortion and no convolution)

<table>
<thead>
<tr>
<th>DSN Models</th>
<th>Error Rate (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSN (one module, no learning of $W$)</td>
<td>1.78</td>
</tr>
<tr>
<td>DSN (one module, learning using Eq.(1.16)</td>
<td>1.10</td>
</tr>
<tr>
<td>DSN (10 modules, learning using Eq.(1.16))</td>
<td>0.83</td>
</tr>
<tr>
<td>DSN (10 modules, learning using Eq.(1.17))</td>
<td>0.79</td>
</tr>
<tr>
<td>DSN (10 modules, learning using Eq.(1.17) &amp; adding softmax)</td>
<td>0.77</td>
</tr>
</tbody>
</table>

That is, we replace the gradient computation in Eq.(1.12):

\[
\frac{dE}{dW} = 2X[H^T \circ (1 - H)^T \circ [H^\dagger(HT^T)(TH^\dagger) - T^T(TH^\dagger)]]
\]

by

\[
\frac{dE}{dW} = 2X[H^T \circ (1 - H)^T \circ [H^\dagger(HT^T)(TH^\dagger) - T^T(TH^\dagger)] + \gamma X[H^T \circ (1 - H)^T \circ [H^\dagger(HX^T)(XH^\dagger) - X^T(XH^\dagger)]]
\]

where $\gamma$ is a new hyperparameter. That is, in addition to predicting target vectors, we also predict the input vectors themselves in each module of the DSN, which serves to regularize the training.

With all modules of the DSN well initialized and learned, we take the output layer of the top module as the “deep learning features” to feed to the softmax classifier. As reported in the experiments below, for the static pattern classification of MNIST, the gain is relatively minor. But for sequence classification as in the TIMIT phone recognition task, the use of the additional log-linear model is more effective.

Table 1.2: Phone error rates using various versions of the DSN in the standard TIMIT phone recognition task

<table>
<thead>
<tr>
<th>DSN Models</th>
<th>Error Rate (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSN (one module, no learning of $W$)</td>
<td>33.0</td>
</tr>
<tr>
<td>DSN (one module, learning using Eq.(1.16)</td>
<td>28.0</td>
</tr>
<tr>
<td>DSN (15 modules, learning using Eq.(1.16))</td>
<td>24.6</td>
</tr>
<tr>
<td>DSN (15 modules, learning using Eq.(1.17))</td>
<td>23.0</td>
</tr>
<tr>
<td>DSN (15 modules, learning using Eq.(1.17) &amp; adding softmax)</td>
<td>20.5</td>
</tr>
</tbody>
</table>

Now we turn to the use of K-DSN to generate features for log-linear clas-
Tuning K-DSN parameters using a procedure motivated by R-prop.

Adaptive K-DSN.

As another prominent example of connecting deep learning features to log-linear models, we use DNN features as the input to the very popular log-linear model — the conditional random field (CRF). The combined model, the DNN-CRF, which can be learning in an end-to-end manner, has been highly successful in speech recognition, performing significantly better than the DNN-HMM (Deng et al., 2010; Kingsbury et al., 2012). In the literature reporting this type of model, the DNN-CRF is also described as the DNN-HMM but the DNN parameters are not learned using the common objective function of cross-entropy but rather using the sequence-level criterion of maximum mutual information (MMI). The latter is also called full-sequence-trained DNN-HMM. Equivalence between the DNN-CRF and the sequence-trained DNN-HMM is well understood, which is elaborated in (Hinton et al., 2012; Deng et al., 2010).

The motivations for the DNN-CRF are discussed here. In the DNN-HMM,
the DNNs are learned to optimize the per-frame cross-entropy between the target HMM state and the DNN predictions. The transition parameters of the HMM and language model scores can be obtained from an HMM-like approach and be trained independently of the DNN weights. However, it has long been known that sequence classification criteria, which are more directly correlated with the overall word or phone error rate, can be very helpful in improving recognition accuracy Bahl et al. (1986); He and Deng (2008).

In Deng et al. (2010), the DNN-CRF was proposed and developed. It was recognized that the use of cross entropy to train DNN for phone sequence recognition does not explicitly take into account the fact that the neighboring frames have smaller distances between the assigned probability distributions over phone class labels. In the proposed approach, the MMI, or equivalently the full-sequence posterior probability 

\[ p(l_1:T|v_1:T) = \exp(\sum_{t=1}^{T}(\gamma_{ij})(l_{t-1}, l_t) + \sum_{t=1}^{T} \sum_{d=1}^{D} \lambda_{l_t,d} h_{td}) \]

(1.18)

where the transition feature \( \phi_{ij}(l_{t-1}, l_t) \) takes on a value of one if \( l_{t-1} = i \) and \( l_t = j \), and otherwise takes on a value of zero, where \( \gamma_{ij} \) is the parameter associated with this transition feature, \( h_{td} \) is the \( d \)-th dimension of the hidden unit value at the \( t \)-th frame at the final layer of the DNN, and where \( D \) is the number of units in the final hidden layer. Note the objective function of Eqn.(1.18) derived from mutual information is the same as the conditional likelihood associated with a specialized linear-chain conditional random field. Here, it is the top most layer of the DNN below the softmax layer, not the raw speech coefficients of MFCC or PLP, that provides “features” to the conditional random field.

To optimize the log conditional probability \( p(l_1:T|v_1:T) \) of the \( n \)-th utterance, we take the gradient over the activation parameters \( \lambda_{kd} \), transition parameters \( \gamma_{ij} \), and the lower-layer weights of the DNN, \( w_{ij} \), according to

\[ \frac{\partial \log p(l_1:T|v_1:T)}{\partial \lambda_{kd}} = \sum_{t=1}^{T}(\delta(l_t^n = k) - p(l_t = k|v_1:T)) h_{td}^n \]

(1.19)
\[ \frac{\partial \log p(l^n_T|v^n_1:T)}{\partial \gamma_{ij}} = \sum_{t=1}^{T} \left[ \delta(l^n_{t-1} = i, l^n_t = j) - p(l^n_{t-1} = i, l^n_t = j|v^n_1:T) \right] \] (1.20)

\[ \frac{\partial \log p(l^n_T|v^n_1:T)}{\partial w_{ij}} = \sum_{t=1}^{T} \left[ \lambda_{td} - \sum_{k=1}^{K} p(l^n_t = k|v^n_1:T) \lambda_{kd} \right] \times h^n_{td}(1-h^n_{td}) x^n_{ti} \] (1.21)

Note that the gradient \( \frac{\partial \log p(l^n_T|v^n_1:T)}{\partial w_{ij}} \) above can be viewed as back-propagating the error \( \delta(l^n_t = k) - p(l^n_t = k|v^n_1:T) \), vs. \( \delta(l^n_t = k) - p(l^n_t = k|v^n_t) \) in the frame-based training algorithm.

In implementing this learning algorithm, which is iterative, the DNN weights are first pre-learned by optimizing the per-frame cross entropy. The transition parameters are then initialized from the combination of the HMM transition matrices and the “phone language” model scores. These transition parameters are further optimized by tuning the transition features while fixing the DNN weights before the joint optimization. Using the joint optimization with careful scheduling, this full-sequence MMI training is shown to outperform the frame-level training by about 5% relative.

Subsequent work on full-sequence MMI training for the DNN-CRF has shown greater success in much larger speech recognition tasks, where lattices need to be invoked when n-gram language models with \( n > 2 \) are used and more sophisticated optimization methods need to be exploited.

### 1.5 Log-Linear Stacking to Combine Multiple Deep Learning Systems

As the final case study on how log-linear models can be usefully connected to deep learning systems, we combine several deep learning systems’ outputs as the features and use a log-linear model either to perform classification or to produce new and more powerful features for a more complex structured classification problem.

For the latter, we describe a method, called log-linear stacking here. Without loss of generality, we take the example of three deep learning systems, whose vector-valued outputs are \( X = [x_1, ..., x_i, ..., x_N] \), \( Y = [y_1, ..., y_i, ..., y_N] \), and \( Z = [z_1, ..., z_i, ..., z_N] \), respectively. The log-linear stacking method combines such outputs, which may be posterior probabilities of discrete classes, according to

\[ U \log x_i + V \log y_i + W \log z_i + b \] (1.22)

where matrices \( U, V, \) and \( W, \) and vector \( b \) are free stacking parameters.
1.6 Discussion and Conclusions

To be learned so that combined features approach their given target-class values \( T = [t_1, \ldots, t_i, \ldots, t_N] \) in a supervised learning setting.

To learn these free parameters, we can adopt total least square error as the cost function, subject to \( L_2 \) regularization:

\[
E = 0.5 \sum_{i=1}^{T} ||\tilde{U}\tilde{x}_i + V\tilde{y}_i + W\tilde{z}_i + b||^2 + \lambda_1 ||U||^2 + \lambda_2 ||V||^2 + \lambda_3 ||W||^2
\]

(1.23)

where

\[
\tilde{x}_i = \log x_i, \quad \tilde{y}_i = \log y_i, \quad and \quad \tilde{z}_i = \log z_i,
\]

and \( \lambda_1, \lambda_2 \) and \( \lambda_3 \) are Lagrange multipliers treated as tunable hyper-parameters.

The solution to the problem of minimizing (Eq.1.23) is the following analytical operations involving matrix inversion and vector-matrix multiplication:

\[
[U, V, W, b] = [T\tilde{X}', T\tilde{Y}', T\tilde{Z}', \sum_i t_i] M^{-1},
\]

(1.24)

where

\[
M = \begin{bmatrix}
\tilde{X}\tilde{X}' + \lambda_1 I & \tilde{Y}\tilde{X}' & \tilde{Z}\tilde{X}' & \sum_{i=1}^{N} \tilde{x}' \\
\tilde{X}\tilde{Y}' & \tilde{Y}\tilde{Y}' + \lambda_2 I & \tilde{Z}\tilde{Y}' & \sum_{i=1}^{N} \tilde{y}' \\
\tilde{X}\tilde{Z}' & \tilde{Y}\tilde{Z}' & \tilde{Z}\tilde{Z}' + \lambda_3 I & \sum_{i=1}^{N} \tilde{z}' \\
\sum_{i=1}^{N} \tilde{x}' & \sum_{i=1}^{N} \tilde{y}' & \sum_{i=1}^{N} \tilde{z}' & N
\end{bmatrix}
\]

(1.25)

The log-linear stacking described above has been applied to combine three deep learning systems (DNN, RNN, and CNN) for speech recognition. The experimental paradigm is shown in Figure 1.5, where the log-linearly assembled features are fed to an HMM sequence decoder. Positive experimental results have been reported recently in (Deng and Platt, 2014).

1.6 Discussion and Conclusions

In this chapter, we provide a new way of looking at the DNN and other deep architectures (e.g., the DSN and K-DSN) in terms of their ability to provide high-level features to relatively simple classifiers such as log-linear models, rather than viewing an entire deep learning model as a complex classifier. Since log-linear models can either be a simple static classifier, such as the softmax or maximum entropy models, or be a structured sequence classifier,
the deep classifiers based on high-level features provided by the common deep learning systems can be either static or sequential. The parameters of such deep classifiers can be learned either in an end-to-end manner — using backpropagation from the log-linear model’s output all the way into the deep learning system, or in two separate stages — keeping the pre-trained deep learning features fixed while learning the log-linear model parameters. The former way of training tends to be more effective in reducing training errors but is also prone to overfitting. The latter training method, on the other hand, would suffer less from overfitting and be most appropriate for multi-task or transfer learning where the in-domain training data are relatively scarce.

We have provided three case studies in this chapter to concretely illustrate the effectiveness of the above paradigm of pre-training deep learning features in constructing deep classifiers grounded on otherwise shallow log-linear modeling. The examples include the use of the DSN and K-DSN to construct a static deep classifier based on the softmax function, the use of the DNN
to construct a sequential deep classifier based on the CRF, and the use
of aggregated DNN, CNN, and RNN to construct and a log-linear stacker.
Other case studies, not covered in detail in this chapter but demonstrating
similar effectiveness, would include the use of DNN- or DSN-extracted high-
level features for RNN-based and CRF-based sequence classifiers.

Returning to the main theme of this book on log-linear modeling, we
point out that one major attraction of this modeling paradigm is the flex-
ibility in exploiting a wide range of human-engineered features based on
the application-domain knowledge and problem constraints. The paradigm
advanced in this chapter is to replace these knowledge-driven features by
automatically acquired, data-drive features derived by the DNN while main-
taining the same structured sequence discriminative power of the log-linear
models such as CRF. However, the obvious downside of this data-driven ap-
proach is the lack of freedom to incorporate domain knowledge and problem
constraints which are reliable and relevant to the tasks. One promising di-
rection to overcome this weakness is to integrate the DNN feature extractor
with deep generative models, which, via the mechanism of probabilistically
modeling dependencies among latent variables and the observed data, can
naturally embed application-domain constraints in the model space. This
provides a more principled way to incorporate domain knowledge than in
the feature space as is commonly done in traditional log-linear modeling.
Then the integrated DNN feature extractor, which may use appropriately
derived activation functions and structured weight matrices to reflect the
embedded problem constraints, will more than adequately compensate for
the lack of hand-crafted features based on knowledge engineering in the tra-
ditional log-linear modeling. This is non-trivial research, attributed largely
to intractability in inference algorithms associated with most deep gener-
ative models including those for the human speech process (Deng, 1999, 2003;
Lee et al., 2003, 2004).

While the generally intractable deep generative models are difficult to in-
tegrate with the DNN in principle, simpler and less deep or even shallow
generative models may be exploited even though they incorporate domain-
specific properties and constraints in a cruder manner. In this case, the
shallow models can be stacked layer-by-layer, each “layer” being associated
with one iterative step in the easy inference procedure for a shallow gener-
ative model, to form a deep architecture. The formation of the deep model
may also be accomplished in ways similar to stacking restricted Boltzmann
machines to form the deep belief network (Hinton and Salakhutdinov, 2006;
Hinton et al., 2006) and to stacking shallow neural nets (with one hidden
layer) to form a DSN (Deng et al., 2012b). Then the weights on differ-
ent layers can be relaxed to be independent of each other and the whole
deep network can be trained using the powerful and efficient discriminative algorithm of backpropagation (e.g., (Stoyanov et al., 2011)). Without running the backpropagation algorithm to full convergence (i.e., early stop) or with the use of appropriate constraints in optimizing the desired objective function (e.g., Chen and Deng (2014)), domain knowledge and problem constraints established in the generative model before backpropagation will be partially maintained after the discriminative optimization for parameter updates.

In summary, deep learning features, including those provided by the DNN and DSN as exemplified in the case studies presented in this chapter, are powerful features for log-linear modeling in discrimination tasks. They use layer-by-layer structure to automatically extract information from raw data instead of relying on human experts and knowledge workers to design the non-adaptive features based on the same raw data before feeding them into a log-linear model. However, the advantage of automating feature extraction using DNNs carries with it the limitation of not being able to embed reliable domain knowledge often useful or essential for machine learning tasks in hand such as classification. In order to overcome this apparent weakness, a future direction is to seamlessly fuse DNNs and deep generative models, where the former is good at accomplishing the end task’s goal via backpropagation-like learning and the latter excels at naturally incorporating application-domain constraints and knowledge in the appropriate model space. After getting the best of both worlds, deep learning systems are expected to advance further from the DNN-centric paradigm described in this chapter.

References


1.6 Discussion and Conclusions


L. Deng. Switching dynamic system models for speech articulation and


G. Heigold, H. Ney, P. Lehnen, T. Gass, and R. Schluter. Equivalence of


1.6 Discussion and Conclusions

2012.


Connecting Deep Learning Features to Log-Linear Models
Index

index, 20