Log-Linear Models, Extensions and Applications

Editors:

Aleksandr Aravkin saravkin@us.ibm.com
IBM T.J. Watson Research Center
Yorktown Heights, NY 10589

Li Deng deng@microsoft.com
Microsoft Research
Redmond, WA 98052

Georg Heigold heigold@google.com
Google Research
Mountain View, CA 94043

Tony Jebara jebara@cs.columbia.edu
Columbia University
NYC, NY 10027

Dimitri Kanevski dkanevski@google.com
Google Research
NYC, NY 10011

Stephen J. Wright swright@cs.uwisc.edu
University of Wisconsin
Madison, WI 53706

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

Li Deng deng@microsoft.com
Microsoft Research
Redmond , WA, USA 98052

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 leaning 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

Background

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:

$$Ce^{\sum_i w_i f_i(\boldsymbol{x})} \quad or \quad Ce^{\boldsymbol{w}^T \boldsymbol{f}(\boldsymbol{x})} \tag{1.1}$$

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

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 classificaiton problems with multiple ($K > 2$) classes:

$$P(y = j|\boldsymbol{x}) = \frac{e^{s_j}}{\sum_{k=1}^{K} e^{s_k}}, \quad where \;\; s_j = \boldsymbol{w}_j^T \boldsymbol{x}, \;\; j = 1, 2, ..., K \tag{1.2}$$

The reason for calling this function "softmax" is that the effect of exponentiating the K values of $s_1, s_2, \ldots, s_K$ 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 zereo 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 $\max(s_1, s_2, \ldots, s_K)$.

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).

Motivations of using deep learning features

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 mini-batch-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 pf 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).

Chapter outline

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 $\boldsymbol{f}(\boldsymbol{x})$, 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 $\boldsymbol{f}(\boldsymbol{x})$) 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, $\boldsymbol{x}_t$, 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.

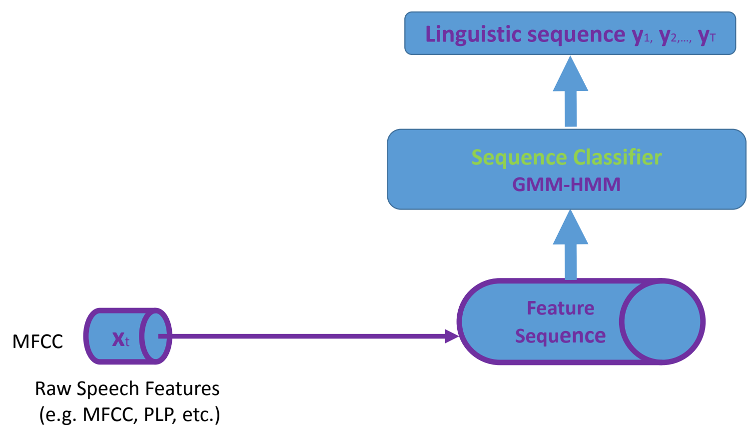

Figure 1.1: Illustration of the GMM-HMM-based speech recognition system: Feeding low-level speech sequences into the GMM-HMM sequence classifier.

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.,

The DNN

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 $\mathbf{h}_{1,t}, \mathbf{h}_{2,t}$, and $\mathbf{h}_{3,t}$ illustrate three hidden state vectors of the DNN for the low-level speech feature vector $\boldsymbol{x}_t$ at time t, and $\boldsymbol{y}_t$ is the corresponding output vector of the DNN. Denoted by $\boldsymbol{d}_t$ 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 $\hat{v}_{3,t}$ 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.

The DNN-HMM

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

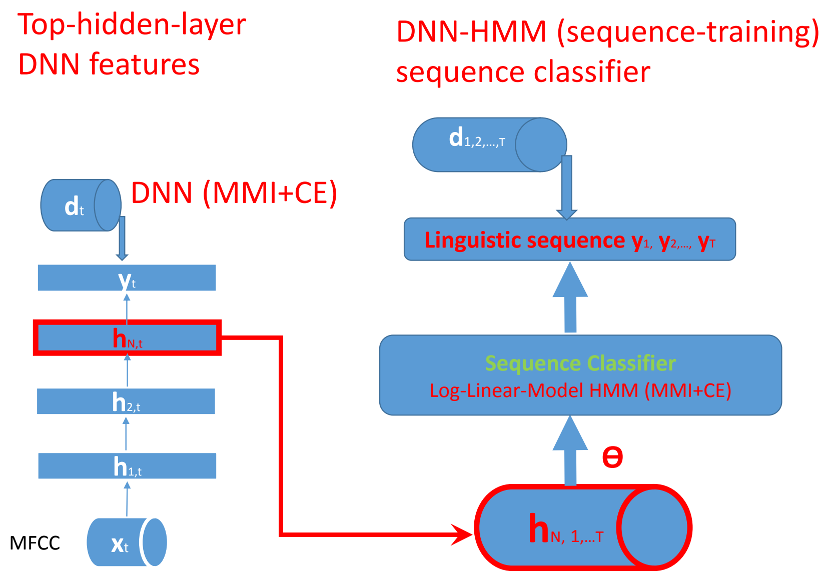

Figure 1.2: Illustration of the DNN-HMM-based speech recognition system: Feeding high-level DNN-derived feature sequences into a log-linear sequence classifier.

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

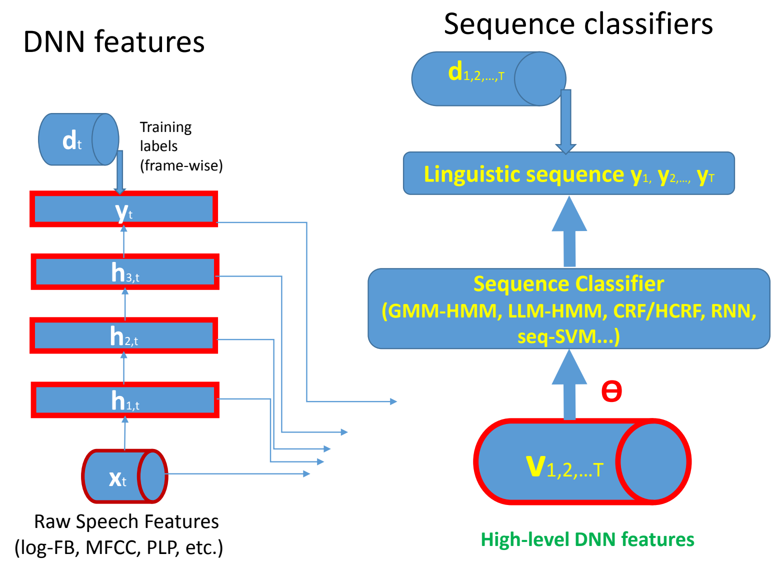

Figure 1.3: Extensions of the system of Figure 1.2 in two ways. First, various hidden layers and the output layer of the DNN can be combined (e.g., via concatenation) to form the high-level DNN features; Second, the sequence classifier can be extended to many different types.

this section.

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.

The DSN

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 **U** and lower-layer weight matrix denoted by **W**.

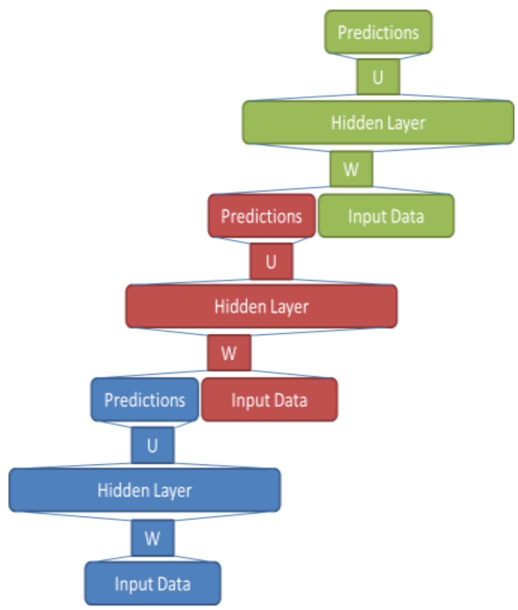

Figure 1.4: Illustration of a DSN with three modules, each with a different color. Each module consists of three layers connected by upper weight matrix denoted by **U** and lower weight matrix denoted by **W**.

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 $\mathbf{x}_i = [\mathbf{x}_{1i}, ..., \mathbf{x}_{ji}, ..., \mathbf{x}_{Di}]^T$. Then, the input units correspond to the elements of $\mathbf{x}_i$, with dimensionality D. Second, the non-linear layer consists of a set of sigmoidal hidden units. Denote by L the number of hidden units and define

$$\mathbf{h}_i = \sigma(\mathbf{W}^T \mathbf{x}_i) \tag{1.3}$$

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

$$D_m = D + C(m-1), \quad m = 1, 2, \cdots, M \tag{1.4}$$

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 $\mathbf{T} = [\mathbf{t}_1, ..., \mathbf{t}_i, ..., \mathbf{t}_N]$, where each target $\mathbf{t}_i = [\mathbf{t}_{1i}, ..., \mathbf{t}_{ji}, ..., \mathbf{t}_{Ci}]^T$, are available. We use the loss function of mean square error to learn weight matrices $\mathbf{U}$ assuming $\mathbf{W}$ is given. That is, we aim to minimize:

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

where $\mathbf{Y} = [\mathbf{y}_1, ..., \mathbf{y}_i, ..., \mathbf{y}_N]$.

Importantly, if weight matrix $\mathbf{W}$ is determined already (e.g., via judicious initialization), then the hidden layer values $\mathbf{H} = [\mathbf{h}_1, ..., \mathbf{h}_i, ..., \mathbf{h}_N]$ are also determined. Consequently, upper-layer weight matrix $\mathbf{U}$ in each module can be determined by setting the gradient

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

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

$$\mathbf{U} = (\mathbf{H}\mathbf{H}^T)^{-1}\mathbf{H}\mathbf{T}^T. \tag{1.7}$$

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

$$\mathbf{U} = F(\mathbf{W}). \tag{1.8}$$

Learning DSN weight matrix $\mathbf{W}$

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

$$\hat{\mathbf{W}} = \underset{\mathbf{W}}{\operatorname{argmax}}\, E(\mathbf{U}^*, \mathbf{W}), \quad \textit{subject to } \mathbf{U}^* = F(\mathbf{W}). \tag{1.9}$$

Total derivative

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

$$\frac{dE}{d\mathbf{W}} = \frac{\partial E}{\partial \mathbf{W}} + \frac{\partial E}{\partial \mathbf{U}^*}\frac{\partial \mathbf{U}^*}{\partial \mathbf{W}}. \tag{1.10}$$

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

$$\hat{\mathbf{W}} = \underset{\mathbf{W}}{\operatorname{argmax}}\, E[F(\mathbf{W}), \mathbf{W}]. \tag{1.11}$$

Derivation of the total derivative

And the analytical form of the total derivative is derived below:

$$\begin{aligned}
\frac{dE}{d\mathbf{W}} &= \frac{dTr[(\mathbf{U}^T\mathbf{H}-\mathbf{T})(\mathbf{U}^T\mathbf{H}-\mathbf{T})^T]}{d\mathbf{W}} \\
&= \frac{dTr[(([(\mathbf{H}\mathbf{H}^T)^{-1}\mathbf{H}\mathbf{T}^T]^T\mathbf{H}-\mathbf{T})([(\mathbf{H}\mathbf{H}^T)^{-1}\mathbf{H}\mathbf{T}^T]^T\mathbf{H}-\mathbf{T})^T]}{d\mathbf{W}} \\
&= \frac{dTr[\mathbf{T}\mathbf{T}^\mathbf{T}-\mathbf{T}\mathbf{H}^\mathbf{T}(\mathbf{H}\mathbf{H}^\mathbf{T})^{-\mathbf{1}}\mathbf{H}\mathbf{T}^\mathbf{T}]}{d\mathbf{W}} \\
&= \frac{dTr[(\mathbf{H}\mathbf{H}^T)^{-1}\mathbf{H}\mathbf{T}^T\mathbf{T}\mathbf{H}^T]}{d\mathbf{W}} \\
&= \frac{dTr[(\sigma(\mathbf{W}^\mathbf{T}\mathbf{X})[\sigma(\mathbf{W}^\mathbf{T}\mathbf{X})]^\mathbf{T})^{-\mathbf{1}}\sigma(\mathbf{W}^\mathbf{T}\mathbf{X})\mathbf{T}^\mathbf{T}\mathbf{T}[\sigma(\mathbf{W}^\mathbf{T}\mathbf{X})]^\mathbf{T}]}{d\mathbf{W}} \\
&= 2\mathbf{X}[\mathbf{H}^T \circ (1-\mathbf{H})^T \circ [\mathbf{H}^\dagger(\mathbf{H}\mathbf{T}^T)(\mathbf{T}\mathbf{H}^\dagger) - \mathbf{T}^T(\mathbf{T}\mathbf{H}^\dagger)] \qquad (1.12)
\end{aligned}$$

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

Importantly, the total derivative in Eq.1.12 with respective to lower-layer weight matrix $\mathbf{W}$ 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 $\mathbf{W}$ instead of total derivative. That is, backpropagation algorithm gives only the first term $\frac{\partial E}{\partial \mathbf{W}}$ 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

Distinction from partial derivative in backpropagation

$$\frac{\partial E}{\partial \mathbf{U}^*}\frac{\partial \mathbf{U}^*}{\partial \mathbf{W}}, \qquad (1.13)$$

where $\mathbf{U}^* = F(\mathbf{W}) = (\mathbf{H}\mathbf{H}^T)^{-1}\mathbf{H}\mathbf{T}^T$ and each vector component of matrix $\mathbf{H}$ is $\mathbf{h}_i = \sigma(\mathbf{W}^T\mathbf{x}_i)$.

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 $\mathbf{U}$ while fixing $\mathbf{W}$.

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 K-DSN

The DSN architecture discussed in the preceding section has convex learning for weight matrix $\mathbf{U}$ given the hidden layers outputs in each module, but the learning of weight matrix $\mathbf{W}$ is non-convex. For most applications, the size of $\mathbf{U}$ is comparable to that of $\mathbf{W}$ 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 $\mathbf{W}$ into a convex one (Hutchinson et al., 2012, 2013). In the K-DSN extension discussed here, non-convex learning for $\mathbf{W}$ is completely eliminated using kernel trick. In deriving the K-DSN architecture and the associated learning algorithm, the sigmoidal hidden layer $\mathbf{h}_i = \sigma(\mathbf{W}^T\mathbf{x}_i)$ in a DSN module is generalized into a generic nonlinear mapping function $\mathbf{G}(\mathbf{X})$ from raw input features $\mathbf{X}$. The high or possibly infinite dimensionality in $\mathbf{G}(\mathbf{X})$ is determined only implicitly by the kernel function.

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

The K-DSN predictions

$$\mathbf{y}(\mathbf{x}) = \mathbf{k}^T(\mathbf{x})(C\mathbf{I} + \mathbf{K})^{-1}\mathbf{T} \tag{1.14}$$

where $\mathbf{T} = [\mathbf{t}_1, ..., \mathbf{t}_i, ..., \mathbf{t}_N]$ are the target vectors for training, and the kernel vector $\mathbf{k}(\mathbf{x})$ is defined such that its elements have values of $k_n(\mathbf{x}) = k(\mathbf{x}_n, \mathbf{x})$

in which $\mathbf{x}_n$ is a training sample and $\mathbf{x}$ 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

$$\mathbf{K}^{(l)} = \mathbf{G}\left([\mathbf{X}|\mathbf{Y}^{(l-1)}|\mathbf{Y}^{(l-2)}|...\mathbf{Y}^{(1)}]\right)\mathbf{G}^T\left([\mathbf{X}|\mathbf{Y}^{(l-1)}|\mathbf{Y}^{(l-2)}|...\mathbf{Y}^{(1)}]\right) \quad (1.15)$$

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 $\mathbf{W}$ as in the DSN, and the kernel parameter (e.g., the single variance parameter σ 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 ($l = 1, 2, ..., L$):

The Gaussian K-DSN hyperparameters

- $\sigma^{(l)}$, 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

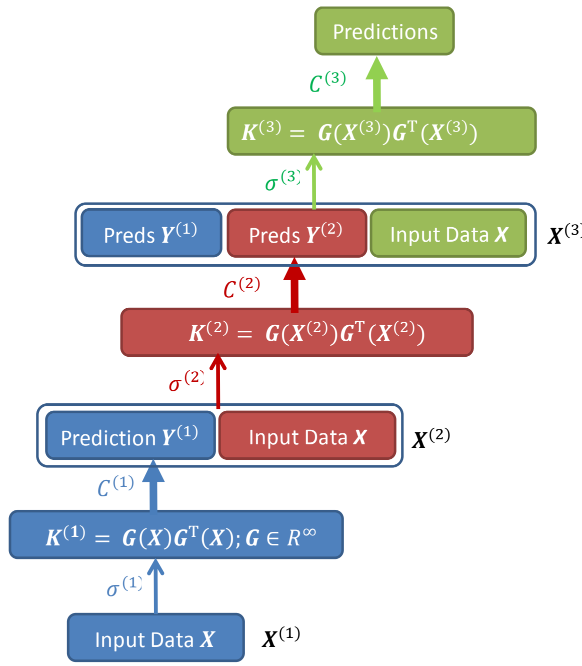

Figure 1.5: The architecture of a K-DSN using Gaussian kernel.

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.

Regularizing DSN learning via initialization with RBMs

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.

Use of asymmetric RBMs for initialization

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)

DSN Models	**Error Rate (%)**
DSN (one module, no learning of **W**)	1.78
DSN (one module, learning using Eq.(1.16)	1.10
DSN (10 modules, learning using Eq.(1.16))	0.83
DSN (10 modules, learning using Eq.(1.17))	0.79
DSN (10 modules, learning using Eq.(1.17) & adding softmax)	0.77

Regularizing DSN learning incorporating reconstruction errors

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

$$\frac{dE}{d\mathbf{W}} = 2\mathbf{X}[\mathbf{H}^T \circ (1-\mathbf{H})^T \circ [\mathbf{H}^\dagger(\mathbf{H}\mathbf{T}^T)(\mathbf{T}\mathbf{H}^\dagger) - \mathbf{T}^T(\mathbf{T}\mathbf{H}^\dagger)] \tag{1.16}$$

by

$$\begin{aligned} \frac{dE}{d\mathbf{W}} &= 2\mathbf{X}[\mathbf{H}^T \circ (1-\mathbf{H})^T \circ [\mathbf{H}^\dagger(\mathbf{H}\mathbf{T}^T)(\mathbf{T}\mathbf{H}^\dagger) - \mathbf{T}^T(\mathbf{T}\mathbf{H}^\dagger)] \\ &+ \gamma\mathbf{X}[\mathbf{H}^T \circ (1-\mathbf{H})^T \circ [\mathbf{H}^\dagger(\mathbf{H}\mathbf{X}^T)(\mathbf{X}\mathbf{H}^\dagger) - \mathbf{X}^T(\mathbf{X}\mathbf{H}^\dagger)] \end{aligned} \tag{1.17}$$

where γ 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.

Adding log-linear model to top of DSN

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

DSN Models	**Error Rate (%)**
DSN (one module, no learning of **W**)	33.0
DSN (one module, learning using Eq.(1.16)	28.0
DSN (15 modules, learning using Eq.(1.16))	24.6
DSN (15 modules, learning using Eq.(1.17))	23.0
DSN (15 modules, learning using Eq.(1.17) & adding softmax)	20.5

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

sifiers. First, we review some practical aspects of tuning the K-DSN parameters. There are much fewer parameters in the K-DSN to tune than the DCN, and there is no need for initialization with often slow, empirical procedures for training the RBM. Importantly, it was found that regularization plays a more important role in K-DCN than in the DSN. The effective regularization schedules that have been developed can often have intuitive insight and can be motivated by optimization tricks. In fact, the regularization procedure shown effective has been motivated by the R-prop algorithm reported in the neural network literature (Riedmiller and Braun, 1993).

Further, it was found empirically that in contrast to the DSN, the effectiveness of the K-DSN does not require careful data normalization and it can more easily handle mixed binary and continuous-valued inputs without data and output calibration.

Adaptive K-DSN

The attractiveness of the K-DSN over the DSN lies partly in its small number of hyperparameters. This makes it easy to adapt the K-DSN from one domain to another. For example, adapting the kernel smoothing parameter in the Gaussian kernel, σ, can be easily carried out with much smaller amounts of adaptation data than adapting other deep models such as DNNs or DSNs.

Given the K-DSN features from the top module, we can connect them directly to a log-linear model as its inputs to perform classification tasks. One successful example is given in (Deng et al., 2012a) for the sequence classification task of slot filling in spoken language understanding.

1.4 DNN Features for Conditional-Random-Field Sequence Classifiers

DNN-CRF and sequence-trained DNN-HMM

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 $p(l_{1:T}|v_{1:T})$ of the whole sequence of labels, $l_{1:T}$ given the whole visible feature utterance $v_{1:T}$ (or the hidden feature sequence $h_{1:T}$ extracted by the DNN), is used as the objective function for learning all parameters of the DNN-CRF:

Full-sequence conditional likelihood

$$p(l_{1:T}|v_{1:T}) = p(l_{1:T}|h_{1:T}) = \frac{exp(\sum_{t=1}^{T} \gamma_{ij}\phi_{ij}(l_{t-1}, l_t) + \sum_{t=1}^{T}\sum_{d=1}^{D} \lambda_{l_t,d} h_{td})}{Z(h_{1:T})}, \tag{1.18}$$

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 γ_{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 $p(l_{1:T}^n|v_{1:T}^n)$ of the n-th utterance, we take the gradient over the activation parameters λ_{kd}, transition parameters γ_{ij}, and the lower-layer weights of the DNN, w_{ij}, according to

$$\frac{\partial \log p(l_{1:T}^n|v_{1:T}^n)}{\partial \lambda_{kd}} = \sum_{t=1}^{T} (\delta(l_t^n = k) - p(l_t^n = k|v_{1:T}^n))h_{td}^n \tag{1.19}$$

$$\frac{\partial \log p(l_{1:T}^n | v_{1:T}^n)}{\partial \gamma_{ij}} = \sum_{t=1}^{T} [\delta(l_{t-1}^n = i, l_t^n = j) - p(l_{t-1}^n = i, l_t^n = j | v_{1:T}^n)] \quad (1.20)$$

$$\frac{\partial \log p(l_{1:T}^n | v_{1:T}^n)}{\partial w_{ij}} = \sum_{t=1}^{T} [\lambda_{l_{td}} - \sum_{k=1}^{K} p(l_t^n = k | v_{1:T}^n) \lambda_{kd}] \times h_{td}^n (1 - h_{td}^n) x_{ti}^n \quad (1.21)$$

Note that the gradient $\frac{\partial \log p(l_{1:T}^n | v_{1:T}^n)}{\partial w_{ij}}$ above can be viewed as back-propagating the error $\delta(l_t^n = k) - p(l_t^n = k | v_{1:T}^n)$, vs. $\delta(l_t^n = k) - p(l_t^n = k | v_t^n)$ in the frame-based training algorithm.

Implementation in practice

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 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 $\mathbf{X} = [\mathbf{x}_1, ..., \mathbf{x}_i, ..., \mathbf{x}_N]$, $\mathbf{Y} = [\mathbf{y}_1, ..., \mathbf{y}_i, ..., \mathbf{y}_N]$, and $\mathbf{Z} = [\mathbf{z}_1, ..., \mathbf{z}_i, ..., \mathbf{z}_N]$, respectively. The log-linear stacking method combines such outputs, which may be posterior probabilities of discrete classes, according to

$$\mathbf{U} \log \mathbf{x}_i + \mathbf{V} \log \mathbf{y}_i + \mathbf{W} \log \mathbf{z}_i + \mathbf{b} \quad (1.22)$$

where matrices $\mathbf{U}$, $\mathbf{V}$, and $\mathbf{W}$, and vector $\mathbf{b}$ are free stacking parameters to be learned so that combined features approach their given target-class values $\mathbf{T} = [\mathbf{t}_1, ..., \mathbf{t}_i, ..., \mathbf{t}_N]$ 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:

$$E = 0.5 \sum_{i=1}^{T} ||\mathbf{U}\tilde{\mathbf{x}}_i + \mathbf{V}\tilde{\mathbf{y}}_i + \mathbf{W}\tilde{\mathbf{z}}_i + \mathbf{b}||^2 + \lambda_1||\mathbf{U}||^2 + \lambda_2||\mathbf{V}||^2 + \lambda_3||\mathbf{W}||^2 \quad (1.23)$$

where

$$\tilde{\mathbf{x}}_i = \log \mathbf{x}_i, \quad \tilde{\mathbf{y}}_i = \log \mathbf{y}_i, \quad \textit{and} \quad \tilde{\mathbf{z}}_i = \log \mathbf{z}_i,$$

and λ_1, λ_2 and λ_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:

$$[\mathbf{U}, \mathbf{V}, \mathbf{W}, \mathbf{b}] = [\mathbf{T}\tilde{\mathbf{X}}', \mathbf{T}\tilde{\mathbf{Y}}', \mathbf{T}\tilde{\mathbf{Z}}', \sum_i \mathbf{t}_i] \ \mathbf{M}^{-1}, \quad (1.24)$$

where

$$\mathbf{M} = \begin{bmatrix} \tilde{\mathbf{X}}\tilde{\mathbf{X}}' + \lambda_1\mathbf{I} & \tilde{\mathbf{Y}}\tilde{\mathbf{X}}' & \tilde{\mathbf{Z}}\tilde{\mathbf{X}}' & \sum_{i=1}^N \tilde{\mathbf{x}}' \\ \tilde{\mathbf{X}}\tilde{\mathbf{Y}}' & \tilde{\mathbf{Y}}\tilde{\mathbf{Y}}' + \lambda_2\mathbf{I} & \tilde{\mathbf{Z}}\tilde{\mathbf{Y}}' & \sum_{i=1}^N \tilde{\mathbf{y}}' \\ \tilde{\mathbf{X}}\tilde{\mathbf{Z}}' & \tilde{\mathbf{Y}}\tilde{\mathbf{Z}}' & \tilde{\mathbf{Z}}\tilde{\mathbf{Z}}' + \lambda_3\mathbf{I} & \sum_{i=1}^N \tilde{\mathbf{z}}' \\ \sum_{i=1}^N \tilde{\mathbf{x}}' & \sum_{i=1}^N \tilde{\mathbf{y}}' & \sum_{i=1}^N \tilde{\mathbf{z}}' & N \end{bmatrix} \quad (1.25)$$

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,

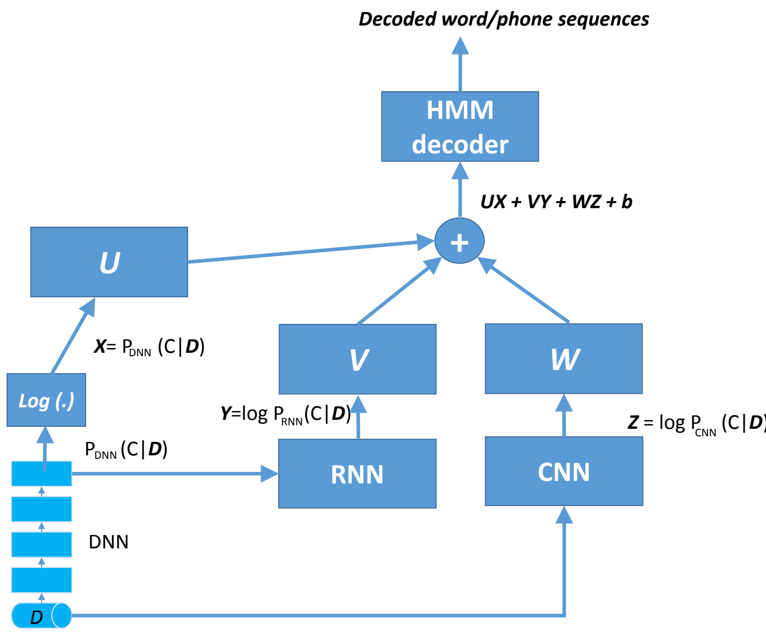

Figure 1.6: Block diagram showing the experimental paradigm for the use of log-linear stacking of three deep learning systems' outputs for speech recognition. The three systems are 1) deep neural network (DNN, left); 2) recurrent neural network which takes the DNN outputs as its inputs (RNN, middle), and 3) Convolutional neural network (CNN, right).

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 flexibility 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 maintaining the same structured sequence discriminative power of the log-linear models such as CRF. However, the obvious downside of this data-driven approach is the lack of freedom to incorporate domain knowledge and problem constraints which are reliable and relevant to the tasks. One promising direction 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 traditional log-linear modeling. This is non-trivial research, attributed largely to intractability in inference algorithms associated with most deep generative 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 integrate 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 generative 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 different 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 optimiting 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

T. Anastasakos, J. Mcdonough, R. Schwartz, and J. Makhoul. A compact model for speaker-adaptive training. In *Proc. International Conference on Spoken Language Processing (ICSLP)*, pages 1137–1140, 1996.

L.R. Bahl, P.F. Brown, P.V. de Souza, and R.L. Mercer. Maximum mutual information estimation of HMM parameters for speech recognition. In *Proc. International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 1986.

J. Baker, L. Deng, J. Glass, S. Khudanpur, C.-H. Lee, N. Morgan, and D. O'Shgughnessy. Research developments and directions in speech recognition and understanding, part i. *IEEE Signal Processing Magazine*, 26 (3):75–80, 2009a.

J. Baker, L. Deng, J. Glass, S. Khudanpur, C.-H. Lee, N. Morgan, and D. O'Shgughnessy. Updated MINDS report on speech recognition and understanding. *IEEE Signal Processing Magazine*, 26(4):78–85, 2009b.

Y. Bengio. Learning deep architectures for ai. *Foundations and Trends in Machine Learning*, 2(1):1–127, 2009.

A Biem, S. Katagiri, E. McDermott, and Biing-Hwang Juang. An application of discriminative feature extraction to filter-bank-based speech recognition. *Speech and Audio Processing, IEEE Transactions on*, 9(2):96–110, Feb 2001.

L. Breiman. Stacked regression. *Machine Learning*, 24:49–64, 1996.

J. Bridle, L. Deng, J. Picone, H. Richards, J. Ma, T. Kamm, M. Schuster, S. Pike, and R. Reagan. An investigation fo segmental hidden dynamic models of speech coarticulation for automatic speech recognition. *Final Report for 1998 Workshop on Langauge Engineering, CLSP, Johns Hopkins*, 1998.

J. Chen and L. Deng. A primal-dual method for training recurrent neural networks constrained by the echo-state property. In *Proc. ICLR*, 2014.

R. Chengalvarayan and L. Deng. HMM-based speech recognition using state-dependent, discriminatively derived transforms on mel-warped dft features. *IEEE Transactions on Speech and Audio Processing*, (5):243256, 1997a.

R. Chengalvarayan and L. Deng. Use of generalized dynamic feature parameters for speech recognition. *IEEE Transactions on Speech and Audio Processing*, 5(3):232–242, May 1997b.

William W. Cohen and Vitor Rocha de Carvalho. Stacked sequential learning. In *Proc. IJCAI*, pages 671–676, 2005.

G. Dahl, Dong Yu, Li Deng, and Alex Acero. Large vocabulary continuous speech recognition with context-dependent DBN-HMMs. In *Proc. International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4688–4691, 2011.

George E Dahl, Dong Yu, Li Deng, and Alex Acero. Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *IEEE Transactions on Audio, Speech and Language Processing*, 20(1):30–42, 2012.

L. Deng. Computational models for speech production. In *Computational Models of Speech Pattern Processing*, pages 199–213. Springer-Verlag, New York, 1999.

L. Deng. Switching dynamic system models for speech articulation and acoustics. In *Mathematical Foundations of Speech and Language Processing*, pages 115–134. Springer-Verlag, New York, 2003.

L. Deng and J. Chen. Sequence classification using high-level features extracted from deep neural networks. In *Proc. International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2014.

L. Deng and X. Li. Machine learning paradigms in speech recognition: An overview. *IEEE Transactions on Audio, Speech, and Language Processing,*, 21(5):1060–1089, 2013.

L. Deng and D. O'Shaughnessy. *SPEECH PROCESSING — A Dynamic and Optimization-Oriented Approach*. Marcel Dekker Inc, NY, 2003.

L. Deng and John Platt. Ensemble deep learning for speech recognition. In *Proc. Annual Conference of International Speech Communication Association (INTERSPEECH)*, 2014.

L. Deng and Roberto Togneri. Deep dynamic models for learning hidden representations of speech features. In *Speech and Audio Processing for Coding, Enhancement and Recognition*. Springer, 2014.

L. Deng and D. Yu. Deep convex network: A scalable architecture for speech pattern classification. In *Proc. Annual Conference of International Speech Communication Association (INTERSPEECH)*, 2011.

L. Deng and D. Yu. *Deep Learning: Methods and Applications*. NOW Publishers, 2014.

L. Deng, P. Kenny, M. Lennig, V. Gupta, F. Seitz, and P. Mermelsten. Phonemic hidden markov models with continuous mixture output densities for large vocabulary word recognition. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 39(7):1677–1681, 1991.

L. Deng, G. Ramsay, and D. Sun. Production models as a structural basis for automatic speech recognition. *Speech Communication*, 33(2-3):93–111, Aug 1997.

L. Deng, A. Acero, M. Plumpe, and XD. Huang. Large vocabulary speech recognition under adverse acoustic environments. In *Proc. International Conference on Spoken Language Processing (ICSLP)*, pages 806–809, 2000.

L. Deng, G. Hinton, and D. Yu. Deep learning for speech recognition and related applications. In *NIPS Workshop*, Whistler, Canada, 2009.

L. Deng, M. Seltzer, D. Yu, A. Acero, A. Mohamed, and G. Hinton. Binary coding of speech spectrograms using a deep auto-encoder. In *Proc. Annual Conference of International Speech Communication Association (INTERSPEECH)*, 2010.

L. Deng, G. Tur, X. He, and D. Hakkani-Tur. Use of kernel deep convex networks and end-to-end learning for spoken language understanding. In *Proc. IEEE Workshop on Automfatic Speech Recognition and Understanding (ASRU)*, 2012a.

L. Deng, D. Yu, and J. Platt. Scalable stacking and learning for building deep architectures. In *Proc. International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2012b.

L. Deng, O. Abdel-Hamid, and D. Yu. A deep convolutional neural network using heterogeneous pooling for trading acoustic invariance with phonetic confusion. In *Proc. International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Vancouver, Canada, May 2013a.

L. Deng, G. Hinton, and B. Kingsbury. New types of deep neural network learning for speech recognition and related applications: An overview. In *Proc. International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Vancouver, Canada, May 2013b.

P. Divenyi, S. Greenberg, and G. Meyer. *Dynamics of Speech Production and Perception.* IOS Press, 2006.

F. Flego and M. Gales. Discriminative adaptive training with VTS and JUD. In *Proc. IEEE Workshop on Automfatic Speech Recognition and Understanding (ASRU)*, pages 170–175, 2009.

A. Ghoshal, Pawel Swietojanski, and Steve Renals. Multilingual training of deep-neural netowrks. Proc. International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2013.

Asela Gunawardana, Milind Mahajan, Alex Acero, and John C Platt. Hidden conditional random fields for phone classification. In *Proc. Annual Conference of International Speech Communication Association (INTERSPEECH)*, pages 1117–1120, 2005.

X. He and L. Deng. *DISCRIMINATIVE LEARNING FOR SPEECH RECOGNITION: Theory and Practice.* Morgan and Claypool, 2008.

X. He and L. Deng. Speech recognition, machine translation, and speech translation —- A unified discriminative learning paradigm. *IEEE Signal Processing Magazine*, 27:126–133, 2011.

X. He and L. Deng. Speech-centric information processing: An optimization-oriented approach. 31, 2013.

X. He, L. Deng, and W. Chou. Discriminative learning in sequential pattern recognition — a unifying review for optimization-oriented speech recognition. *IEEE Signal Processing Magazine*, 25(5):14–36, 2008.

G. Heigold, H. Ney, P. Lehnen, T. Gass, and R. Schluter. Equivalence of generative and log-linear models. *IEEE Transactions on Audio, Speech and Language Processing*, 19(5):1138 –1148, july 2011.

G Heigold, V Vanhoucke, A Senior, P Nguyen, M Ranzato, M Devin, and J Dean. Multilingual acoustic models using distributed deep neural networks. Proc. International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2013.

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

G. Hinton, S. Osindero, and Y. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18:1527–1554, 2006.

G. Hinton, L. Deng, D. Yu, G. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. Sainath, and B. Kingsbury. Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

J.-T. Huang, Jinyu Li, Dong Yu, Li Deng, and Yifan Gong. Cross-language knowledge transfer using multilingual deep neural network with shared hidden layers. In *Proc. International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2013.

B. Hutchinson, L. Deng, and D. Yu. A deep architecture with bilinear modeling of hidden representations: applications to phonetic recognition. In *Proc. International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2012.

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

F. Jiao, S. Wang, Chi-Hoon Lee, Russell Greiner, and Dale Schuurmans. Semi-supervised conditional random fields for improved sequence segmentation and labeling. In *Proc. Annual Meeting of the Association for Computational Linguistics (ACL)*, 2006.

O. Kalinli, M.L. Seltzer, J. Droppo, and A. Acero. Noise adaptive training for robust automatic speech recognition. *IEEE Transactions on Audio, Speech and Language Processing*, 18(8):1889 –1901, nov. 2010.

B. Kingsbury, T. Sainath, and H. Soltau. Scalable minimum bayes risk training of deep neural network acoustic models using distributed hessian-free optimization. In *Proc. Annual Conference of International Speech Communication Association (INTERSPEECH)*, 2012.

A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, volume 1, page 4, 2012.

J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. International Conference on Machine Learning (ICML)*, pages 282–289, 2001.

Q. Le, M. Ranzato, R. Monga, M. Devin, K. Chen, J Dean G. Corrado, and A. Ng. Building high-level features using large scale unsupervised learning. In *Proc. International Conference on Machine Learning (ICML)*, 2012.

L.J. Lee, H. Attias, and L. Deng. Variational inference and learning for segmental switching state space models of hidden speech dynamics. In *Proc. International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2003.

L.J. Lee, H. Attias, L. Deng, and P. Fieguth. A multimodal variational approach to learning and inference in switching state space models. In *Proc. International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2004.

H. Lin, L. Deng, D. Yu, Y. Gong, A. Acero, and C.-H. Lee. A study on multilingual acoustic modeling for large vocabulary ASR. In *Proc. International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4333–4336, 2009.

G. Mann and A. McCallum. Generalized expectation criteria for semi-supervised learning of conditional random fields. In *Proc. Annual Meeting of the Association for Computational Linguistics (ACL)*, 2008.

A. Mohamed, George E Dahl, and Geoffrey Hinton. Deep belief networks for phone recognition. In *NIPS Workshop on Deep Learning for Speech Recognition and Related Applications*, 2009.

A. Mohamed, D. Yu, and L. Deng. Investigation of full-sequence training of deep belief networks for speech recognition. In *Proc. Annual Conference of International Speech Communication Association (INTERSPEECH)*, 2010.

A. Mohamed, G. Dahl, and G. Hinton. Acoustic modeling using deep belief networks. *IEEE Transactions on Audio, Speech and Language Processing*, 20(1):14–22, jan. 2012a.

A. Mohamed, Geoffrey Hinton, and Gerald Penn. Understanding how deep belief networks perform acoustic modelling. In *Proc. International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4273–4276, 2012b.

N. Morgan. Deep and wide: Multiple layers in automatic speech recognition. *IEEE Transactions on Audio, Speech and Language Processing*, 20(1), jan. 2012.

J. Picone, S. Pike, R. Regan, T. Kamm, J. bridle, L. Deng, Z. Ma, H. Richards, and M. Schuster. Initial evaluation of hidden dynamic models on conversational speech. In *Proc. International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 1999.

D. Povey and P. C. Woodland. Minimum phone error and i-smoothing for improved discriminative training. In *Proc. International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 105–108, 2002.

L. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

M. Riedmiller and H. Braun. A direct adaptive method for faster backpropagation learning: The rprop algorithm. In *IEEE Int. Conf. Neural Networks*, pages 586–591, 1993.

T. Sainath, Brian Kingsbury, and Bhuvana Ramabhadran. Auto-encoder bottleneck features using deep belief networks. In *Proc. International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4153–4156, 2012.

T. Sainath, Brian Kingsbury, Abdel-rahman Mohamed, and Bhuvana Ramabhadran. Learning filter banks within a deep neural network framework. In *Proc. IEEE Workshop on Automfatic Speech Recognition and Understanding (ASRU)*, 2013.

Jürgen Schmidhuber. Deep learning in neural networks: An overview. *CoRR*, abs/1404.7828, 2014.

F. Seide, G. Li, and D. Yu. Conversational speech transcription using context-dependent deep neural networks. In *Proc. Annual Conference of International Speech Communication Association (INTERSPEECH)*, August 2011a.

F. Seide, Gang Li, Xie Chen, and Dong Yu. Feature engineering in context-dependent deep neural networks for conversational speech transcription. In *Proc. IEEE Workshop on Automfatic Speech Recognition and Understanding (ASRU)*, pages 24–29, 2011b.

K. Stevens. *Acoustic Phonetics*. MIT Press, 2000.

V. Stoyanov, A. Ropson, and J. Eisner. Empirical risk minimization of graphical model parameters given approximate inference, decoding, and model structure. *Proc. International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2011.

Z. Tuske, M. Sundermeyer, R. Schluter, and H. Ney. Context-dependent MLPs for LVCSR: Tandem, hybrid or both. In *Proc. Annual Conference of International Speech Communication Association (INTERSPEECH)*, 2012.

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

S.J. Wright, D. Kanevsky, L. Deng, X. He, G. Heigold, and H. Li. Optimization algorithms and applications for speech and language processing. *IEEE Transactions on Audio, Speech, and Language Processing*, 21(11): 2231–2243, November 2013.

Z. Yan, Qiang Huo, and Jian Xu. A scalable approach to using DNN-derived features in GMM-HMM based acoustic modeling for LVCSR. In *Proc. Annual Conference of International Speech Communication Association (INTERSPEECH)*, 2013.

D. Yu and L. Deng. Deep-structured hidden conditional random fields for phonetic recognition. In *Proc. International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2010.

D. Yu, L. Deng, and Alex Acero. Hidden conditional random field with distribution constraints for phone classification. In *Proc. Annual Conference of International Speech Communication Association (INTERSPEECH)*, pages 676–679, 2009.

D. Yu, L. Deng, and George Dahl. Roles of pre-training and fine-tuning in context-dependent DBN-HMMs for real-world speech recognition. In *Proc. Neural Information Processing Systems (NIPS) Workshop on Deep Learning and Unsupervised Feature Learning*, 2010.

D. Yu, L. Deng, and F. Seide. The deep tensor neural network with applications to large vocabulary speech recognition. 21(3):388–396, 2013a.

D. Yu, M. Seltzer, Jinyu Li, Jui-Ting Huang, and Frank Seide. Feature learning in deep neural networks - studies on speech recognition tasks, 2013b.

Index