Building DNN Acoustic Models for Large Vocabulary Speech Recognition

Andrew L. Maas, Peng Qi, Ziang Xie, Awni Y. Hannun, Christopher T. Lengerich, Daniel Jurafsky, Andrew Y. Ng,

***Abstract*—Deep neural networks (DNNs) are now a central component of nearly all state-of-the-art speech recognition systems. Building neural network acoustic models requires several design decisions including network architecture, size, and training loss function. This paper offers an empirical investigation on which aspects of DNN acoustic model design are most important for speech recognition system performance. We report DNN classifier performance and final speech recognizer word error rates, and compare DNNs using several metrics to quantify factors influencing differences in task performance. Our first set of experiments use the standard Switchboard benchmark corpus, which contains approximately 300 hours of conversational telephone speech. We compare standard DNNs to convolutional networks, and present the first experiments using locally-connected, untied neural networks for acoustic modeling. We additionally build systems on a corpus of 2,100 hours of training data by combining the Switchboard and Fisher corpora. This larger corpus allows us to more thoroughly examine performance of large DNN models – with up to ten times more parameters than those typically used in speech recognition systems. Our results suggest that a relatively simple DNN architecture and optimization technique produces strong results. These findings, along with previous work, help establish a set of best practices for building DNN hybrid speech recognition systems with maximum likelihood training. Our experiments in DNN optimization additionally serve as a case study for training DNNs with discriminative loss functions for speech tasks, as well as DNN classifiers more generally.**

I. Introduction

DEEP neural network (DNN) acoustic models have driven tremendous improvements in large vocabulary continuous speech recognition (LVCSR) in recent years. Initial research hypothesized that DNNs work well because of unsupervised pre-training [1]. However, DNNs with random initialization yield state-of-the-art LVCSR results for several speech recognition benchmarks [2], [3], [4]. Instead, it appears that modern DNN-based systems are quite similar to long-standing neural network acoustic modeling approaches [5], [6], [7]. Modern DNN systems build on these fundamental approaches but utilize increased computing power, training corpus size, and function optimization heuristics. This paper offers a large empirical investigation of DNN performance on two LVCSR tasks to understand best practices and important design decisions when building DNN acoustic models.

Recent research on DNN acoustic models for LVCSR explores variations in network architecture, optimization techniques, and acoustic model training loss functions. Due to system differences across research groups it can be difficult, for example, to determine whether a performance improvement is due to a better neural network architecture or a different optimization technique. Our work aims to address these concerns by systematically exploring several strategies to improve DNN acoustic models. We view the acoustic modeling DNN component as a DNN classifier and draw inspiration from recent DNN classification research on other tasks – predominantly image classification. Unlike many other tasks, DNN acoustic models in LVCSR are not simply classifiers, but are instead one sub-component of the larger speech transcription system. There is a complex relationship between downstream task performance, word error rate (WER), and the proximal task of training a DNN acoustic model as a classifier. Because of this complexity, it is unclear which improvements to DNN acoustic models will ultimately result in improved performance across a range of LVCSR tasks.

This work empirically examines several aspects of DNN acoustic models in an attempt to establish a set of best practices for creating such models. Further, we seek to understand which aspects of DNN training have the most impact on downstream task performance. This knowledge can guide rapid development of DNN acoustic models for new speech corpora, languages, computational constraints, and language understanding task variants. Furthermore, we not only analyze task performance, but also quantify differences in how various DNNs transform and represent data. Understanding how DNNs process information helps us understand underlying principles to further improve DNNs as classifiers and components of large artificial intelligence systems. To this end, our work serves as a case study for DNNs more generally as both classifiers and components of larger systems.

We first perform DNN experiments on the standard Switchboard corpus. We use this corpus to analyze the effect of DNN size on task performance, and find that although there are 300 hours of training data we can cause DNNs to overfit on this task by increasing DNN model size. We then investigate several techniques to reduce over-fitting including the popular dropout regularization technique. We next analyze neural network architecture choices by comparing deep convolutional neural networks (DCNNs), deep locally untied neural networks (DLUNNs), and standard DNNs. This comparison also evaluates alternative input features since convolutional approaches rely on input features with meaningful time and frequency dimensions.

To explore DNN performance with fewer constraints imposed by over-fitting, we next build a baseline LVCSR system by combining the Switchboard and Fisher corpora. This results in roughly 2,100 hours of training data and represents one of the largest collections of conversational speech available for

Department of Computer Science, Stanford University, Stanford, CA, 94305 USA e-mail: amaas@cs.stanford.edu.

academic research. This larger corpus allows us to explore performance of much larger DNN models, up to ten times larger than those typically used for LVCSR. Using this larger corpus we also evaluate the impact of optimization algorithm choice, and the number of hidden layers used in a DNN with a fixed number of total free parameters. We analyze our results not only in terms of final task performance, but also compare sub-components of task performance across models. Finally, we quantify differences in how different DNN architectures process information.

Section II outlines the steps involved in building neural network acoustic models for LVCSR, and describes previous work on each step. This process outline contextualizes the questions addressed by our investigations, which we present in Section III. Section IV describes the neural network architectures and optimization algorithms evaluated in this paper. Section V presents our experiments on the Switchboard corpus, which focus on regularization and network dense versus convolutional architectural choices. We then present experiments on the combined Switchboard and Fisher corpora in Section VII which explore the performance of larger and deeper DNN architectures. We compare and quantify DNN representational properties in Section IX, and conclude in Section X.

II. Neural Network Acoustic Models

DNNs act as acoustic models for hidden Markov model (HMM) speech recognition systems using the *hybrid HMM* approach. A hybrid HMM system largely resembles the standard HMM approach to speech recognition using Gaussian mixture model (GMM) acoustic models. A full overview of LVCSR systems is beyond the scope of this work, so we instead refer to previous articles for an overview of HMM-based speech recognition systems [8], [9], [10], [11], [12].

Our work focuses on the acoustic modeling component of the LVCSR system. The acoustic model approximates the distribution $p(x|y)$ which is the probability of observing a given short span of acoustic features, x, conditioned on an HMM state label, y. The acoustic input features represent about 25ms of audio in most LVCSR systems. The HMM state labels y for LVCSR are *senones* – clustered, context-dependent sub-phonetic states. A hybrid HMM system uses a neural network to approximate $p(x|y)$ in place of a GMM.

A neural network does not explicitly model the distribution $p(x|y)$ required by the HMM. Instead, we train neural networks to estimate $p(y|x)$, which allows us to view the neural network as a classifier of senones given acoustic input. We can use Bayes' rule to obtain $p(x|y)$ given the neural network output distribution $p(y|x)$,

$$p(x|y) = \frac{p(y|x)p(x)}{p(y)}. \quad (1)$$

The distribution $p(y)$ is the prior distribution over senones, which we approximate as the empirical distribution of senone occurrence in the training set. This is easy to obtain as it is simply a normalized count of senones in the training set. We usually can not tractably estimate probability of acoustic features, $p(x)$. This represents the probability of observing a particular span of acoustic features – a difficult distribution to model. However, because our acoustic features x are fixed during decoding the term $p(x)$ is a constant, albeit unknown, scaling factor. As a result we drop the term and instead provide the HMM with an unscaled acoustic model score,

$$\frac{p(y|x)}{p(y)}. \quad (2)$$

This term is not a properly formed acoustic model probability, but it is sufficient to perform HMM decoding to maximize a combination of acoustic and language model scores. The decoding procedure introduces an acoustic model scaling term to empirically adjust for the scaling offset introduced by using un-normalized probabilities.

Using neural networks as acoustic models for HMM-based speech recognition was introduced over 20 years ago [5], [7], [13]. Much of this original work developed the basic ideas of hybrid HMM-DNN systems which are used in modern, state-of-the-art systems. However, until much more recently neural networks were not a standard component in the highest performing LVCSR systems. Computational constraints and the amount of available training data severely limited the pace at which it was possible to make progress on neural network research for speech recognition. Instead, Gaussian mixture models were the standard choice for acoustic modeling as researchers worked to refine the HMM architecture, decoding frameworks, and signal processing challenges associated with building high-performance speech recognizers.

While GMMs and their extensions produced gains on benchmark LVCSR tasks over the span of many years, the resulting systems became increasingly complex. Many of the complexities introduced focused purely on increasing the representational capacity of GMM acoustic models. In parallel to this effort, there was a resurgence of interest in neural networks under the new branding of *deep learning* within the machine learning community. Work in this area focused on overcoming optimization issues involved in training DNNs by applying unsupervised pre-training to obtain a better initialization for supervised learning tasks [14], [15].

DNNs provided an interesting path forward for acoustic modeling as neural networks offer a direct path to increasing representational capacity, provided it is possible to find a good set of DNN parameters. Early experiments with DNNs used fairly small phoneme recognition tasks using monophone recognition systems and small datasets like TIMIT [16]. In 2011 researchers demonstrated that DNNs can also be applied to LVCSR systems with context-dependent triphone states, rather than monophone states. This innovation, coupled with the larger representational capacity of DNNs as compared to GMMs, yielded substantial reductions in WER on multiple challenging LVCSR tasks [17], [18]. Within two years DNN acoustic models showed gains on challenging tasks within the LVCSR systems of Microsoft, Google, and IBM [2].

Several factors are attributed to the success of modern DNN approaches as compared to previous work with hybrid acoustic models. Specifically the large total number of network parameters, increased number of hidden layers, and initialization by pre-training were thought to drive performance of mod-

ern hybrid HMM systems. Researchers quickly established that hybrid HMMs work much better when using context-dependent triphones in place of monophones [1]. Initializing DNN weights with unsupervised pre-training was initially thought to be important for good performance, but researchers later found that purely supervised training from random initial weights yields nearly identical final system performance [19]. Using DNNs with many hidden layers and many total parameters has generally found to be beneficial [20], but the role of hidden layers and total network size is not generally understood.

Having defined our hybrid HMM system and how we use the neural network output $p(y|x)$ within the complete LVCSR system, we next focus on how we build neural networks to model the senone distribution $p(y|x)$. To better understand the detailed aspects related to building and using neural network acoustic models for LVCSR we break the process into a series of modeling and algorithmic choices. This set of steps allows us to better contextualize previous work, and further convey what aspects of the process are not yet fully understood. We define the process as five steps:

1) **Label Set**. The set of labels for our acoustic model are defined by the baseline HMM-GMM system we choose to use. Early work in neural network acoustic models used context-independent monophone states. Recent work with DNN acoustic models established that context-dependent states are critical to success [17], which is generally true of modern LVCSR systems. Several variants of context-dependent states exist, and have been tried with DNN acoustic models. In this work we use context-dependent triphone senones created by our baseline HMM-GMM system.
2) **Forced Alignment**. Our training data originally contains word-level transcriptions without time alignments for words. We must assign a senone label to each acoustic input frame in each training utterance. We use a forced alignment of the ground-truth transcriptions to generate a sequence of senone labels for each utterance which is consistent with the word transcription for the utterance. Generating a forced alignment is a standard step of training any HMM-based system. The standard approach to hybrid speech recognition creates a forced alignment of the training data using an HMM-GMM system [11]. The aligned data is then used to train a neural network acoustic model. Previous work found that using a trained HMM-DNN system to realign the training data for a second round of DNN training produces small gains in overall system performance [21]. This process has more recently been generalized to yield an HMM-DNN training procedure which starts with no forced alignment but repeatedly uses a DNN to realign the training data [22]. In our experiments we used a single forced alignment produced by the baseline HMM-GMM system as this the most standard approach when building DNN acoustic models.
3) **Neural Network Architecture**. The size and structure of neural networks used for acoustic modeling is by far the largest difference between modern HMM-DNN systems and those used before 2010. Modern DNNs use more than one hidden layer, making them *deep*. As a general property, depth is an important feature for the success of modern DNNs. Several groups recently found replacing the standard sigmoidal hidden units with rectified linear units in DNNs leads to WER gains and simpler training procedures for deep architectures [23], [24], [25].
Neural networks with only a single hidden layer perform worse than their deeper counterparts on a variety of speech tasks, even when the total number of model parameters is held fixed [21], [20], [26]. Whether deeper is always better, or how deep a network must be to obtain good performance, is not well understood both for speech recognition and DNN classification tasks more generally. The total number of parameters used in modern DNNs is typically 10 to 100 times greater than neural networks used in the original hybrid HMM experiments. This increased model size, which translates to increased representational capacity, is critical to the success of modern DNN-HMM system. It is not clear how far we can push DNN model size or depth to continue increasing LVCSR performance.
Size and depth are the most fundamental architectural choices for DNNs, but we can also consider a variety of alternative neural network architectures aside from a series of densely-connected hidden layers. DCNNs are an alternative to densely-connected networks which are intended to leverage the meaningful time and frequency dimensions in certain types of audio input features. Recent work with DCNNs found them to be useful first on phoneme recognition tasks but also on LVCSR tasks when used in addition to a standard DNN acoustic model [27], [28], [29]. DCNNs change the first and sometimes second hidden layers of the neural network architecture, but otherwise utilize the same densely-connected multilayer architecture of DNNs.
Perhaps a larger architectural change from DNNs are deep *recurrent* neural networks (DRNNs) which introduce a temporally recurrent hidden layer between hidden layers. The resulting architecture has outputs which no longer process each input context window independently, reflecting the temporal coherence and correlation of speech signals. DRNNs are a modern extension of the time-delay neural network first used for phoneme recognition by [30] and recurrent network approach of [31]. Modern recurrent network approaches to acoustic modeling have shown some initial success on large vocabulary tasks [32], and tasks where limited training data is available [33], [34], [35], [36]. The long term impact of DRNNs for HMM-DRNN systems is not yet clear as both the DRNN and HMM reason about the temporal dynamics of the input, which may introduce redundancy or interference. Researchers continue to propose and compare many architectural variants for acoustic modeling and other speech-related tasks [37].
4) **Neural Network Loss Function**. Given a training set

of utterances accompanied by frame-level senone labels we must choose a loss function to use when training our acoustic model. The space of possible loss functions is large, as it also includes the set of possible regularization terms we might use to control over-fitting during training. The default choice for DNN acoustic models is the cross entropy loss function, which corresponds to maximizing the likelihood of the observed label given the input. Cross entropy is the standard choice when training DNNs for classification tasks, but it ignores the DNN as a component of the larger ASR system. To account for more aspects of the overall system, discriminative loss functions were introduced for ASR tasks. Discriminative loss functions were initially developed for GMM acoustic models [38], [39], [40], [41], but were recently applied to DNN acoustic model training [4], [3], [42]. Discriminative training of DNN acoustic models begins with standard cross entropy training to achieve a strong initial solution. The discriminative loss function is used either as a second step, or additively combined with the standard cross entropy function. We can view discriminative training as a task-specific loss function which produces a DNN acoustic model to better act as a sub-component of the overall ASR system.

For whatever loss function we choose, we can additionally apply one or more regularization terms to form the final training objective function. Regularization is especially important for DNNs where we can easily increase models' representational capacity. The simplest form of regularization widely applied to DNNs is a weight norm penalty, most often used with an ℓ_2-norm penalty. While generally effective, developing new regularization techniques for DNNs is an area of active research. Dropout regularization [2] was recently introduced as a more effective regularization technique for DNN training. Recent work applied dropout regularization for DNN acoustic models, and found it beneficial when combined with other architectural changes [23].

5) **Optimization Algorithm**. Any non-trivial neural network model leads to a non-convex optimization problem. Because of this, our choice of optimization algorithm impacts the quality of local minimum found during optimization. There is little we can say in the general case about DNN optimization since it is not possible to find a global minimum nor estimate how far a particular local minimum is from the best possible solution. The most standard approach to DNN optimization is stochastic gradient descent (SGD). There are many variants of SGD, and practitioners typically choose a particular variant empirically. While SGD provides a robust default choice for optimizing DNNs, researchers continue to work on improving optimization algorithms for DNNs. Nearly all DNN optimization algorithms in popular use are gradient-based, but recent work has shown that more advanced quasi-Newton methods can yield better results for DNN tasks generally [43], [44] as well as DNN acoustic modeling [3]. Quasi-Newton and similar methods tend to be more computationally expensive per update than SGD methods, but the improved optimization performance can sometimes be distributed across multiple processors more easily, or necessary for loss functions which are difficult to optimize well with SGD techniques. Recently algorithms like AdaGrad [45] and Nesterov's Accelerated Gradient (NAG) were applied to DNNs for tasks outside of speech recognition, and tend to provide superior optimization as compared to SGD while still being computationally inexpensive compared to traditional quasi-Newton methods [46].

Amount of time required for training is an important practical consideration for DNN optimization tasks. Several groups have designed and implemented neural network optimization procedures which utilize graphics processing units (GPUs) [47], [48], [49], clusters of dozens to hundreds of computers [50], [51], [52], or clusters of GPUs [53]. Indeed, training time of neural networks has been a persistent issue throughout history, researchers often utilized whatever specialized computing hardware was available at the time [54], [55]. Modern parallelized optimization approaches often achieve a final solution of similar quality to a non-parallelized optimization algorithm, but are capable of doing so in less time, or for larger models, as compared to non-parallelized approaches.

III. Questions Addressed in This Work

At each stage of neural network acoustic model design and training there is a tremendous breadth and depth of prior work. Researchers often focus on improving one particular component of this pipeline while holding all other components fixed. Unfortunately, there is no well-established baseline for the acoustic model building pipeline, so performance improvements of, for example, a particular architectural variant are difficult to assess from examining the literature. Our examines the relative importance of several acoustic model design and training decisions. By systematically varying several critical design components we are able to test the limits of certain architectural choices, and uncover which variations among baseline systems are most relevant for LVCSR performance. We specifically address the following questions in this work:

1) What aspects of neural network architecture are most important for acoustic modeling tasks? We investigate total network size and number of hidden layers using two corpora to avoid overfitting as a confounding factor. We build DNNs with five to ten times the total number of free parameters of DNNs used in most previous work. We also compare optimization algorithms to test whether more modern approaches to stochastic gradient descent are a driving factor in building large DNN acoustic models.

   We additionally compare a much broader architectural choice – locally-connected models versus the standard densely-connected DNN models. Recent work has found improvements when using DCNNs combined with DNNs for acoustic modeling, or when applying DCNNs to audio features with sufficient pre-processing [29]. We

use two types of input features to compare DNNs with DCNNs. We present the first experiments DLUNNs for acoustic modeling. DLUNNs are a generalized version of DCNNs which are still locally connected but learn different weights at each location in the input features rather than sharing weights at all locations.

2) How can we improve the test set generalization of DNN acoustic models? Our experiments on DNN architecture choices reveal that increasing model size easily leads to overfitting issues. We evaluate several modifications to DNN training to improve the generalization performance of large DNNs. We include dropout, a recently introduced regularization technique, as well as early stopping, which has been used in neural network training for many years. Finally, we propose and evaluate *early realignment*, a training technique specific to acoustic modeling, as a path towards improving generalization performance.
3) Do large, deep DNNs differ from shallow, smaller DNNs in terms of phonetic confusions or information processing metrics? DNN acoustic models are clearly successful in application but we do not yet understand why they perform well, or how they might be improved. We analyze the WER and classification errors made by large DNN acoustic models to test what improvements in sub-tasks ultimately lead to overall system WER improvements. Further, we look at information encoding metrics to quantify how information encoding changes in larger or deeper DNNs.

We address each of these questions in separate experiments using the Switchboard 300 hour corpus and a combined 2,100 hour corpus when appropriate for the experiment. In Section IV we describe the DNN, DCNN, and DLUNN architecture computations used in this work. Section V addresses questions of model size and overfitting on the Switchboard corpus while Section VI uses the same baseline Switchboard system to compare DCNN and DLUNN architectures to DNNs and baseline GMMs. Section VII presents experiments using the larger training corpus to explore issues of model size, DNN depth, and optimization algorithm. Sections VIII and IX analyze the performance and coding properties of DNNs trained on the large combined corpus to better understand how large DNNs encode information, and integrate into LVCSR systems.

IV. Neural Network Computations

To address the stated research questions we employ three different classes of neural network architecture. Each architecture amounts to a different set of equations to convert input features into a predicted distribution over output classes. We describe here the specifics of each architecture, along with the loss function and optimization algorithms we use.

A. Cross Entropy Loss Function

All of our experiments utilize the cross entropy classification loss function. For some experiments we apply regularization techniques in addition to the cross entropy loss function to improve generalization performance. Many loss functions specific to speech recognition tasks exist, and are a topic of active research. We choose to focus only on cross entropy because training with cross entropy is almost always the first step, or an additional loss function criterion, when experimenting with more task-specific loss functions. Additionally, the cross entropy loss function is a standard choice for classification tasks, and using it allows our experiments to serve as a case study for large scale DNN classification tasks more generally.

The cross entropy loss function does not consider each utterance in its entirety. Instead it is defined over individual samples of acoustic input x and senone label y. The cross entropy objective function for a single training pair (x, y) is,

$$-\sum_{k=1}^{K} 1\{y = k\} \log \hat{y}_k, \tag{3}$$

where K is the number of output classes, and $\hat{y}_k$ is the probability that the model assigns to the input example taking on label k.

Cross entropy is a convex approximation to the ideal 0-1 loss for classification. However, when training acoustic models perfect classification at the level of short acoustic spans is not our ultimate goal. Instead, we wish to minimize the word error rate (WER) of the final LVCSR system. WER measures mistakes at the word level, and it is possible to perfectly transcribe the words in an utterance without perfectly classifying the HMM state present at each time step. Constraints present in the HMM and word sequence probabilities from the language model can correct minor errors in state-level HMM observation estimates. Conversely, not all acoustic spans are of equal importance in obtaining the correct word-level transcription. The relationship between classification accuracy rate at the frame level and overall system WER is complex and not well understood. In our experiments we always report both frame-level error metrics and system-level WER to elicit insights about the relationship between DNN loss function performance and overall system performance.

B. Deep Neural Network Computations

A DNN is a series of fully connected hidden layers which transform an input vector x into a probability distribution $\hat{y}$ to estimate the output class. The DNN thus acts as a function approximator for the conditional distribution $p(y|x)$. A DNN parametrizes this function using L layers, a series of hidden layers followed by an output layer. Figure 1 shows an example DNN.

Each layer has a weight matrix W and bias vector b. We compute vector h^1 of first layer activations of a DNN using,

$$h^{(1)}(x) = \sigma(W^{(1)T} x + b^{(1)}), \tag{4}$$

where $W^{(1)}$ and $b^{(1)}$ are the weight matrix and bias vectors respectively for the first hidden layer. In this formulation each column of the matrix $W^{(1)}$ corresponds to the weights for a single hidden unit of the first hidden layer. Because the DNN is fully connected, any real-valued matrix W forms a valid weight matrix. If we instead choose to impose partial

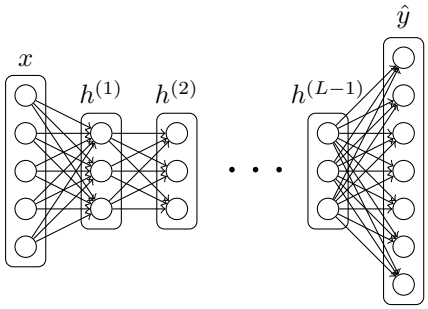

Fig. 1. A DNN with 5-dimensional input, 3-dimensional hidden layers, and 7-dimensional output. Each hidden layer is fully connected to the to the previous and subsequent layer.

connectivity, we are effectively constraining certain entries in W to be 0.

Subsequent hidden layers compute their hidden activation vector $h^{(i)}$ using the hidden activations of the previous layer $h^{(i-1)}$,

$$h^{(i)}(x) = \sigma(W^{(i)T} h^{(i-1)} + b^{(i)}). \quad (5)$$

In all hidden layers we apply a point-wise nonlinearity function $\sigma(z)$ as part of the hidden layer computation. Traditional approaches to neural networks typically use a sigmoidal function. However, in this work we use rectified linear units which were recently shown to lead to better performance in hybrid speech recognition as well as other DNN classification tasks[23], [24], [25]. The rectifier nonlinearity is defined as,

$$\sigma(z) = \max(z, 0) = \begin{cases} z_i & z_i > 0 \\ 0 & z_i \leqslant 0 \end{cases}. \quad (6)$$

The final layer of the DNN must output a properly formed probability distribution over the possible output categories. To do this, the final layer of the DNN uses the *softmax* nonlinearity, which is defined as,

$$\hat{y}_j = \frac{\exp(W_j^{(L)T} h^{(L-1)} + b_j^{(L)})}{\sum_{k=1}^{N} \exp(W_k^{(L)T} h^{(L-1)} + b_k^{(L)})}. \quad (7)$$

Using the softmax nonlinearity we obtain the output vector $\hat{y}$ which is a well-formed probability distribution over the N output classes. This distribution can then be used in the loss function stated in Equation 3, or other loss functions.

Having chosen a loss function and specified our DNN computation equations, we can now compute a sub-gradient of the loss function with respect to the network parameters. Note that because we are using rectifier nonlinearities this is not a true gradient, as the rectifier function is non-differentiable at 0. In practice we treat this sub-gradient as we would a true gradient and apply gradient-based optimization procedures to find settings for the DNN's parameters.

This DNN formulation is fairly standard when compared to work in the speech recognition community. The choice of rectifier nonlinearities is a new one, but their benefit has been reproduced by several research groups. Fully connected neural networks have been widely used in acoustic modeling for over 20 years, but the issues of DNN total size and depth have not been thoroughly studied.

C. Deep Convolutional Neural Networks

The fully-connected DNN architecture presented thus far serves as the primary neural network acoustic modeling choice for modern speech recognition tasks. In contrast, neural networks for computer vision tasks are often deep convolutional neural networks (DCNNs) which exploit spatial relationships in input data [56], [57]. When using spectrogram filter bank representations of speech data, analogous time-frequency relationships may exist. The DCNN architecture allows for parameter sharing and exploiting local time-frequency relationships for improved classification performance. DCNNs follow a convolutional layer with a pooling layer to hard-code invariance to slight shifts in time and frequency. Like fully connected neural network acoustic models, the idea of using localized time-frequency regions for speech recognition was introduced over 20 years ago [30]. Along with the modern resurgence of interest in neural network acoustic models researchers have taken a modern approach to DCNN acoustic models. Our formulation is consistent with other recent work on DCNN acoustic models [29], but we do not evaluate specialized feature post-processing or combining DNNs with DCNNs to form an ensemble of acoustic models. Instead, we ask whether DCNNs should replace DNNs as a robust baseline recipe for building neural network acoustic models.

Like a DNN, a DCNN is a feed-forward model which computes the conditional distribution $p(y|x)$. The initial layers in a DCNN use convolutional layers in place of the standard fully-connected layers present in DNNs. Convolutional layers were originally developed to enable neural networks to deal with large image inputs for computer vision tasks. In a convolutional model, we restrict the total number of network parameters by using hidden units which connect to only a small, localized region of the input. These localized hidden units are applied at many different spatial locations to obtain hidden layer representations for the entire input. In addition to controlling the number of free parameters, reusing localized hidden units at different locations leverages the stationary nature of many input domains. In the computer vision domain, this amounts to reusing the same edge-sensitive hidden units at each location of the image rather than forcing the model to learn the same type of hidden unit for each location separately.

Figure 2 shows a convolutional hidden layer connected to input features with time and frequency axes. A single weight matrix W_1 connects to a 3x3 region of the input and we compute a hidden unit activation value using the same rectifier nonlinearity presented in Equation 6. We apply this same procedure at all possible locations of the input, moving one step at a time across the input in both dimensions. This process produces a *feature map* $h^{(1,1)}$ which is the hidden activation values for W_1 at each location of the input. The feature map itself has meaningful time and frequency axes because we preserve these dimensions as we convolve across the input to compute hidden unit activations.

Our convolutional hidden layer has a feature map with redundancies because we apply the hidden units at each location as we slide across the input. Following the convolutional layer, we apply a *pooling* operation. Pooling acts as

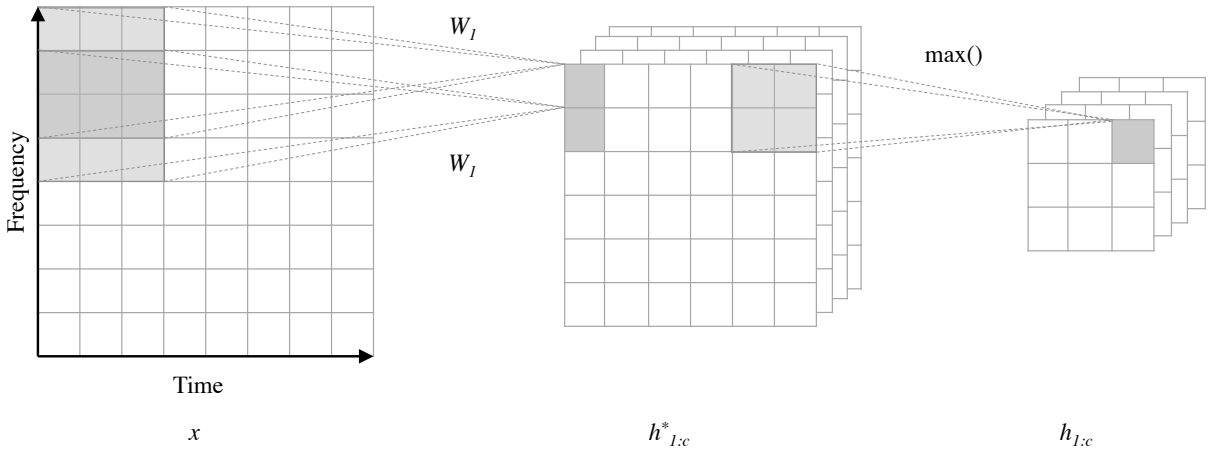

Fig. 2. Convolution and pooling first layer architecture. Here the filter size is 5×5, and the pooling dimension is 3×3. Pooling regions are non-overlapping. Note that the 5×5 filters applied to each position in the convolution step are constrained to be the same. For max-pooling, the maximum value in each 3×3 grid is extracted.

a down-sampling step, and hard-codes invariance to slight translations in the input. Like the localized windows used in the convolutional layer, the pooling layer connects to a contiguous, localized region of its input – the feature map produced by a convolutional hidden layer. The pooling layer does not have overlapping regions. We apply this pooling function to local regions in each feature map. Recall that a feature map contains the hidden unit activations for only a single hidden unit. We are thus using pooling to select activation values for each hidden unit separately, and not forcing different hidden units to compete with one another. In our work, we use *max pooling* which applies a max function to the set of inputs in a single pooling region. Max pooling is a common choice of pooling function for neural networks in both computer vision and acoustic modeling tasks [58], [27], [29]. The most widely used alternative to max pooling replaces the max function with an averaging function. Results with max pooling and average pooling are often comparable.

The overall architecture of a DCNN consists of one or more layers of convolution followed by pooling followed by densely connected hidden layers and a softmax classifier. Essentially we build convolution and pooling layers to act as input to a DNN rather than building a DNN from the original input features. It is not possible to interleave densely connected and convolutional hidden layers because a densely connected hidden layer does not preserve spatial or time-frequency relationships in their hidden layer representations. The DCNN architecture contains more hyper-parameters than a standard DNN because we must select the number of convolutional layers, input region size for all convolution and pooling layers, and pooling function. These are additional hyper-parameters to the choices of depth and hidden layer size common to all types of deep neural network architectures.

D. Deep Local Untied Neural Networks

DCNNs combine two architectural ideas simultaneously – locally-connected hidden units and sharing weights across multiple hidden units. We need not apply both of these architectural ideas simultaneously. In a *deep local untied neural network* (DLUNN) we again utilize locally-connected hidden units but do not share weights at different regions of the input. Figure 3 shows an example DLUNN architecture, which differs only from a DCNN architecture by using different weights at each location of the first hidden layer. When applying a local untied hidden layer to Mel-spectrum time-frequency input features the hidden units can process different frequency ranges using different hidden units. This allows the network to learn slight variations that may occur when a feature occurs at a lower frequency versus a higher frequency.

In DLUNNs, the architecture is the same as in the convolutional network, except that filters applied to different regions of the input are not constrained to be the same. Thus untied neural networks can be thought of as convolutional neural networks using locally connected computations and without weight-sharing. This results in a large increase in the number of parameters for the untied layers relative to DCNNs. Following each locally united layer we apply a max pooling layer which behaves identically to the pooling layers in our DCNN architecture. Grouping units together with a max pooling function often results in hidden weights being similar such that the post-pooling activations are an invariant feature which detects a similar time-frequency pattern at different regions of the input.

E. Optimization Algorithms

Having defined several neural network architectures and the loss function we wish to optimize, we must specify which gradient-based algorithm we use to find a local minimum of our loss function. We consider only stochastic gradient techniques in our work as batch optimization, which requires computing the gradient across the entire dataset at each step, is impractical for the datasets we use. There are several variants of stochastic gradient techniques, many with different convergence properties when applied to convex optimization problems. Because neural network training is a non-convex

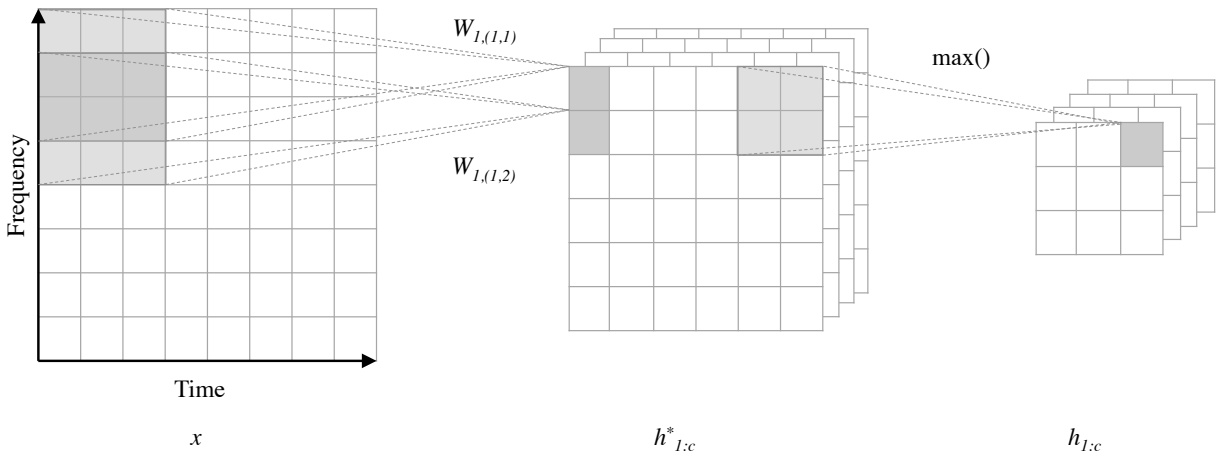

Fig. 3. Locally connected untied first layer architecture. Here the filter size is 5×5, and the pooling dimension is 3×3. Pooling regions are non-overlapping. Unlike the convolutional layer shown in Figure 2, the network learns a unique 5×5 set of weights at each location. The max pooling layer otherwise behaves identically to the pooling layer in a convolutional architecture.

problem, it is difficult to make general statements about optimality of optimization methods. Instead, we consider the choice of optimization algorithm as a heuristic which may lead to better performance in practice. We consider two of the most popular stochastic gradient techniques for our neural network training.

The first optimization algorithm we consider is stochastic gradient with classical momentum (CM) [59], [60]. This technique is probably the most standard optimization algorithm choice in modern neural network research. To minimize a cost function $f(\theta)$ classical momentum updates amount to,

$$v_t = \mu v_{t-1} - \epsilon \nabla f(\theta_{t-1}) \quad (8)$$

$$\theta_t = \theta_{t-1} + v_t, \quad (9)$$

where v_t denotes the accumulated gradient update, or *velocity*, $\epsilon > 0$ is the learning rate, and the momentum constant $\mu \in [0, 1]$ governs how we accumulate the velocity vector over time. By setting μ close to one, one can expect to accumulate the gradient information across a larger set of past updates. However, it can be shown that for extremely ill-conditioned problems, a high momentum for classical momentum method might actually cause fluctuations in the parameter updates. This in turn can result in slower convergence.

Recently the Nesterov's accelerated gradient (NAG) [61] technique was found to address some of the issues encountered when training neural networks with CM. Both methods follow the intuition that accumulating the gradient updates along the course of optimization will help speed up convergence. NAG accumulates past gradients using an alternative update equation that finds a better objective function value with less sensitivity to optimization algorithm hyper-parameters on some neural network tasks. The NAG update rule is defined as,

$$v_t = \mu_{t-1} v_{t-1} - \epsilon_{t-1} \nabla f(\theta_{t-1} + \mu_{t-1} v_{t-1}) \quad (10)$$

$$\theta_t = \theta_{t-1} + v_t. \quad (11)$$

Intuitively, this method avoids potential fluctuation in the optimization by looking ahead to the gradient along the update direction. For a more detailed explanation of the intuition underlying NAG optimization for neural network tasks see Figure 7.1 in [62]. In our work, we treat optimization algorithm choice as an empirical question and compare CM with NAG on our acoustic modeling task to establish performance differences.

V. Switchboard 300 Hour Corpus

We first carry out LVCSR experiments on the 300 hour Switchboard conversational telephone speech corpus (LDC97S62). The baseline GMM system and forced alignments are created using the Kaldi open-source toolkit[1] [63]. The baseline recognizer has 8,986 sub-phone states and 200k Gaussians. The DNN is trained to estimate state likelihoods which are then used in a standard hybrid HMM/DNN setup. Input features for the DNNs are MFCCs with a context of ± 10 frames. Per-speaker CMVN is applied and speaker adaptation is done using fMLLR. The features are also globally normalized prior to training the DNN. Overall, the baseline GMM system setup largely follows the existing `s5b` Kaldi recipe and we defer to previous work for details [4]. For recognition evaluation, we report on a test set consisting of both the Switchboard and CallHome subsets of the HUB5 2000 data (LDC2002S09) as well as a subset of the training set consisting of 5,000 utterances.

A. Varying DNN Model Size

We first experiment with perhaps the most direct approach to improving performance with DNNs – making DNNs larger by adding hidden units. Increasing the number of parameters in a DNN directly increases the representational capacity of the model. Indeed, this representational scalability drives much of the modern interest in applying DNNs to large datasets which might easily saturate other types of models. Many existing

[1] `http://kaldi.sf.net`

experiments with DNN acoustic models focus on introducing architecture or loss function variants to further specialize DNNs for speech tasks. We instead ask the question of whether model size alone can drive significant improvements in overall system performance. We additionally experiment with using a larger context window of frames as a DNN input as this should also serve as a direct path to improving the frame classification performance of DNNs.

1) Experiments: We explore three different model sizes by varying the total number of parameters in the network. The number of hidden layers is fixed to five, so altering the total number of parameters affects the number of hidden units in each layer. All hidden layers in a single network have the same number of hidden units. The hidden layer sizes are 2048, 3953 and 5984 which respectively yield models with approximately 36 million (M), 100M and 200M parameters. There are 8,986 output classes which results in the output layer being the largest single layer in any of our networks. In DNNs of the size typically studied in the literature this output layer often consumes a majority of the total parameters in the network. For example in our 36M parameter model the output layer comprises 51% of all parameters. In contrast, the output layer in our 200M model is only 6% of total parameters. Many output classes occur rarely so devoting a large fraction of network parameters to class-specific modeling may be wasteful. Previous work explores factoring the output layer to increase the relative number of shared parameters [64], [65], but this effect occurs naturally by substantially increasing network size. For our larger models we experiment with the standard input of ± 10 context frames and additionally models trained with ± 20 context frames.

All models use hidden units with the rectified linear nonlinearity. For optimization, we use Nesterov's accelerated gradient with a smooth initial momentum schedule which we clamp to a maximum of 0.95 [46]. The stochastic updates are on mini-batches of 512 examples. After each epoch, or full pass through the data, we anneal the learning rate by half. Training is stopped after improvement in the cross entropy objective evaluated on held out development set falls below a small tolerance threshold.

In order to efficiently train models of the size mentioned above, we distribute the model and computation across several GPUs using the distributed neural network infrastructure proposed by [53]. Our GPU cluster and distributed training software is capable of training up to 10 billion parameter DNNs. We restrict our attention to models in the 30M - 200M parameter range. In preliminary experiments we found that DNNs with 200M parameters are representative of DNNs with over one billion parameters for this task. We train models for this paper in a model-parallel fashion by distributing the parameters across four GPUs. A single pass through the training set for a 200M parameter DNN takes approximately 1.5 days. Table I shows frame-level and WER evaluations of acoustic models of varying size compared against our baseline GMM recognizer.

2) Results: Table I shows results for DNNs of varying size and varying amounts of input context. We find that substantially increasing DNN size shows clear improvements in frame-level metrics. Our 200M parameter DNN halves the development set cross entropy cost of the smaller 36M parameter DNN – a substantial reduction. For each increase in DNN model size there is approximately a 10% absolute increase in frame classification accuracy. Frame-level metrics are further improved by using larger context windows. In all cases a model trained with larger context window outperforms its smaller context counterpart. Our best overall model in terms of frame-level metrics is a 200M parameter DNN with context window of ± 20 frames.

However, frame-level performance is not always a good proxy for WER performance of a final system. We evaluate WER on a subset of the training data as well as the final evaluation sets. Large DNN acoustic models substantially reduce WER on the training set. Indeed, our results suggest that further training set WER reductions are possible by continuing to increase DNN model size. However, the gains we observe on the training set in WER do not translate to large performance gains on the evaluation sets. While there is a small benefit of using models larger than the 36M DNN baseline size, building models larger than 100M parameters does not prove beneficial for this task.

3) Discussion: To better understand the dynamics of training large DNN acoustic models, we plot training and evaluation WER performance during DNN training. Figure 4 shows WER performance for our 100M and 200M parameter DNNs after each epoch of cross entropy training. We find that training WER reduces fairly dramatically at first and then continues to decrease at a slower but still meaningful rate. In contrast, nearly all of our evaluation set performance is realized within the first few epochs of training. This has two important practical implications for large DNN training for speech recognition. First, large acoustic models are not beneficial but do not exhibit a strong over-fitting effect where evaluation set performance improves for awhile before becoming increasingly worse. Second, it may be possible to utilize large DNNs without prohibitively long training times by utilizing our finding that most performance comes from the first few epochs, even with models at our scale. Finally, although increasing context window size improves all training set metrics, those gains do not translate to improved test set performance. It seems that increasing context window size provides an easy path to better fitting the training function, but does not result in the DNN learning a meaningful, generalizable function.

B. Dropout Regularization

Dropout is a recently-introduced technique to prevent overfitting during DNN training [2]. The dropout technique randomly masks out hidden unit activations during training, which prevents co-adaptation of hidden units. For each example observed during training, each unit has its activation set to zero with probability $p \in [0, 0.5]$. Several experiments demonstrate dropout as a good regularization technique for tasks in computer vision and natural language processing [57], [66]. [23] found a reduction in WER when using dropout on a 10M parameter DNN acoustic model for a 50 hour broadcast

TABLE I
RESULTS FOR DNN SYSTEMS IN TERMS OF FRAME-WISE ERROR METRICS ON THE DEVELOPMENT SET AS WELL AS WORD ERROR RATES ON THE TRAINING SET AND HUB5 2000 EVALUATION SETS. THE HUB5 SET (EV) CONTAINS THE SWITCHBOARD (SWBD) AND CALLHOME (CH) EVALUATION SUBSETS. WE ALSO INCLUDE WORD ERROR RATES FOR THE FISHER CORPUS DEVELOPMENT SET (FSH) FOR CROSS-CORPUS COMPARISON. FRAME-WISE ERROR METRICS WERE EVALUATED ON 1.7M FRAMES HELD OUT FROM THE TRAINING SET. DNN MODELS DIFFER ONLY BY THEIR TOTAL NUMBER OF PARAMETERS. ALL DNNS HAVE 5 HIDDEN LAYERS WITH EITHER 2,048 HIDDEN UNITS (36M PARAMETERS), 3,953 HIDDEN UNITS (100M PARAMETERS), OR 5,984 HIDDEN UNITS (200M PARAMS).

Model Size	Layer Size	Context	Dev CrossEnt	Dev Acc(%)	Train WER	SWBD WER	CH WER	EV WER
GMM Baseline	N/A	± 0	N/A	N/A	24.93	21.7	36.1	29.0
36M	2048	± 10	1.23	66.20	17.52	15.1	27.1	21.2
100M	3953	± 10	0.77	78.56	13.66	14.5	27.0	20.8
100M	3953	± 20	0.50	85.58	12.31	14.9	27.7	21.4
200M	5984	± 10	0.51	86.06	11.56	15.0	26.8	20.9
200M	5984	± 20	0.26	93.05	10.09	15.4	28.5	22.0

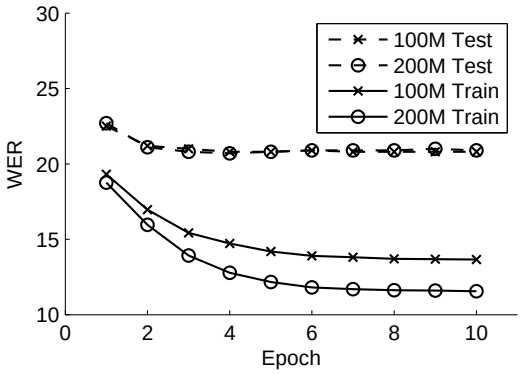

Fig. 4. Train and test set WER as a function of training epoch for systems with DNN acoustic models of varying size. Each epoch is a single complete pass through the training set. Although the training error rate is substantially lower for large models, there is no gain in test set performance.

TABLE II
RESULTS FOR DNN SYSTEMS TRAINED WITH DROPOUT REGULARIZATION (DO) AND EARLY REALIGNMENT (ER) TO IMPROVE GENERALIZATION PERFORMANCE. WE BUILD MODELS WITH EARLY REALIGNMENT BY STARTING REALIGNMENT AFTER EACH EPOCH STARTING AFTER EPOCH TWO (ER2) AND EPOCH FIVE (ER5). WORD ERROR RATES ARE REPORTED ON THE COMBINED HUB5 TEST SET (EV) WHICH CONTAINS SWITCHBOARD (SWBD) AND CALLHOME (CH) EVALUATION SUBSETS. DNN MODEL SIZES ARE SHOWN IN TERMS OF HIDDEN LAYER SIZE AND MILLIONS OF TOTAL PARAMETERS (E.G. 100M)

Model	SWBD	CH	EV
GMM Baseline	21.7	36.1	29.0
2048 Layer (36M)	15.1	27.1	21.2
2048 Layer (36M) DO	14.7	26.7	20.8
3953 Layer (100M)	14.7	26.7	20.7
3953 Layer (100M) DO	14.6	26.3	20.5
3953 Layer (100M) ER2	14.3	26.0	20.2
3953 Layer (100M) ER5	14.5	26.4	20.5
5984 Layer (200M)	15.0	26.9	21.0
5984 Layer (200M) DO	14.9	26.3	20.7

news LVCSR task. Dropout additionally yielded performance gains for convolutional neural networks with less than 10M parameters on both 50 and 400 hour broadcast news LVCSR tasks [28]. While networks which employ dropout during training were found effective in these studies, the authors did not perform control experiments to measure the impact of dropout alone. We directly compare a baseline DNN to a DNN of the same architecture trained with dropout. This experiment tests whether dropout regularization can mitigate the poor generalization performance of large DNNs observed in Section V-A.

1) Experiments: We train DNN acoustic models with dropout to compare generalization WER performance against that of the DNNs presented in Section V. The probability of dropout p is a hyper-parameter of DNN training. In preliminary experiments we found setting $p = 0.1$ to yield the best generalization performance after evaluating several possible values, $p \in \{0.01, 0.1, 0.25, 0.5\}$. The DNNs presented with dropout training otherwise follow our same training and evaluation protocol used thus far, and are built using the same forced alignments from our baseline HMM-GMM system.

2) Results: Table II shows the test set performance of DNN acoustic models of varying size trained with dropout. DNNs trained with dropout improve over the baseline model for all acoustic model sizes we evaluate. The improvement is a consistent 0.2% to 0.4% reduction in absolute WER on the test set. While beneficial, dropout seems insufficient to fully harness the representational capacity of our largest models. Additionally, we note that hyper-parameter selection was critical to finding any gain when using dropout. With a poor setting of the dropout probability p preliminary experiments found no gain and often worse results from training with dropout.

C. Early Stopping

Early stopping is a regularization technique for neural networks which halts loss function optimization before completely converging to the lowest possible function value. We evaluate early stopping as another standard DNN regularization technique which may improve the generalization

performance of large DNN acoustic models. Previous work by [67] found that early stopping training of networks with large capacity produces generalization performance on par with or better than the generalization of a smaller network. Further, this work found that, when using back-propagation for optimization, early in training a large capacity network behaves similarly to a smaller capacity network. Finally, early stopping as a regularization technique is similar to an ℓ_2 weight norm penalty, another standard approach to regularization of neural network training.

1) Results: By analyzing the training and test WER curves in Figure 4 we can observe the best-case performance of an early stopping approach to improving generalization. If we select the lowest test set WER the system achieves during DNN optimization, the 200M parameter DNN achieves 20.7% WER on the EV subset – only 0.1% better than the 100M parameter baseline DNN system. This early stopped 200M model achieves only a 0.5% absolute WER reduction over the much smaller 36M parameter DNN. This suggests that early stopping is beneficial, but perhaps insufficient to yield the full possible benefits of large DNN acoustic models.

D. Early Realignment

We next introduce a potential regularization technique which leverages the process by which training labels are created for DNN acoustic model training. Acoustic model training data is labeled via a forced alignment of the word-level transcriptions. We test whether re-labeling the training data *during* training using the partially-trained DNN leads to improved generalization performance.

Each short acoustic span x_i has an associated HMM state label y_i to form a supervised learning problem for DNN training. Recall that the labels y are generated by a forced alignment of the word-level ground truth labels w to the acoustic signal x. This forced alignment uses an existing LVCSR system to generate a labeling y consistent with the word-level transcription w. The system used to generate the forced alignment is, of course, imperfect, as is the overall speech recognition framework's ability to account for variations in pronunciation. This leads to a dataset $\mathcal{D}$ where supervised training pairs $(x_i, y_i) \in \mathcal{D}$ contain labels y which are imperfect. We can consider a label y_i as a corrupted version of the true label y_i^*. The corruption function which maps y_i^* to y_i is difficult to specify and certainly not independent nor identically distributed at the level of individual samples. Such a complex corruption function is difficult to analyze or address with standard machine learning techniques for label noise. We hypothesize, however, that the noisy labels y are sufficiently correct as to make significantly corrupted labels appear as outliers with respect to the true labels y^*. Under this assumption we outline an approach to improving generalization based on the dynamics of DNN performance during training optimization.

Neural networks exhibit interesting dynamics during optimization. Work on early stopping found that networks with high capacity exhibit behavior similar to smaller, limited capacity networks in early phases of optimization [67]. Combining this finding with the generally smooth functional form of DNN hidden and output units suggests that early in training a large capacity DNN may fit a smooth output function which ignores some of the label noise in y. Of course, a large enough DNN should completely fit the corruptions present in y as optimization converges. Studies on the learning dynamics of DNNs for hierarchical categorization tasks additionally suggest that coarse, high-level output classes are fit first during training optimization [68].

Realignment, or generating a new forced alignment using an improved acoustic model, is a standard tool for LVCSR system training. Baseline LVCSR systems using GMM acoustic models realign several times during training to iteratively improve. While iterative realignments have been helpful in improving system performance in single-layer ANN-HMM hybrid models [5], realignment is typically not used with large DNN acoustic models because of the long training times of DNNs. However, realignment using a fully trained DNN acoustic model often can produce a small reduction in final system WER [2].

We evaluate *early realignment* which generates a new forced alignment early in DNN optimization and then continues training on the new set of labels. Because large capacity DNNs begin accurately predicting labels much earlier in training, early realignment may save days of training time. Further, we hypothesize that a less fully converged network can remove some label distortions while a more completely trained DNN may already be fitting to the corrupt labels given by an imperfect alignment.

1) Experiments: We begin by training an initial DNN using the same HMM-GMM forced alignments and non-regularized training procedures presented thus far. After training the DNN using the initial HMM-GMM alignments for a fixed number of epochs, we use our new HMM-DNN system to generate a new forced alignment for the entire training set. DNN training then proceeds using the same DNN weights but the newly-generated training set labels. As in our other regularization experiments, we hold the rest of our DNN training and evaluation procedures fixed to directly measure the impact of early realignment training. We train 100M parameter five hidden layer DNNs and build models by realigning after either two or five epochs.

In preliminary experiments we found that realignment after each epoch was too disruptive to DNN training and resulted in low quality DNN models. Similarly, we found that starting from a fresh, randomly initialized DNN after realignment performed worse than continuing training from the DNN weights used to generate the realignment. We found it important to reset the stochastic gradient learning rate to its initial value after realignment occurs. Without doing so, our annealing schedule sets the learning rate too low for the optimization procedure to fully adjust to the newly-introduced labels. In a control experiment, we found that resetting the learning rate alone, without realignment, does not improve system performance.

2) Results: Table II compares the final test set performance of DNNs trained with early realignment to a baseline model as well as DNNs trained with dropout regularization. Realignment after five epochs is beneficial compared to the baseline

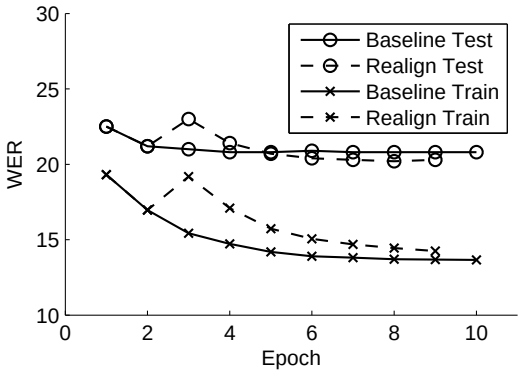

Fig. 5. WER as a function of DNN training epoch for systems with DNN acoustic models trained with and without label realignment after epoch 2. A DNN which re-generates its training labels with a forced alignment early during optimization generalizes much better to test data than a DNN which converges to the original labels.

DNN system, but slightly worse than a system which realigns after two epochs of training. Early realignment leads to better WER performance than all models we evaluated trained with dropout and early stopping. This makes early realignment the overall best regularization technique we evaluated on the Switchboard corpus. We note that only *early* realignment outperforms dropout regularization – a DNN trained with realignment after five epochs performs comparably to a DNN of the same size trained with dropout.

3) Discussion: Figure 5 shows training and test WER curves for 100M parameter DNN acoustic models trained with early realignment and a baseline DNN with no realignment. We note that just after realignment both train and test WER increase briefly. This is not surprising as realignment substantially changes the distribution of training examples. The DNN trained with realignment trains for three epochs following realignment before it begins to outperform the baseline DNN system.

We can quantify how much the labeling from realignment differs from the original labeling by computing the fraction of labels changed. In early realignment 16.4% of labels are changed by realignment while only 10% of labels are changed when we realign with the DNN trained for five epochs. This finding matches our intuition that as a large capacity DNN trains it converges to fit the corrupted training samples extremely well. Thus when we realign the training data with a fully trained large capacity DNN the previously observed labels are reproduced nearly perfectly. Realigning with a DNN earlier in optimization mimics realigning with a higher bias model which relabels the training set with a smoother approximate function. Taken together, our results suggest early realignment leverages the high bias characteristics of the initial phases of DNN training to reduce WER while requiring minimal additional training time.

Early realignment also shows a huge benefit to training time compared to traditional realignment. The DNN trained with realignment after epoch five must train an additional three epochs, for a total of eight, before it can match the performance of a DNN trained with early realignment. For DNNs of the scale we use, this translates to several days of compute time. The training time and WER reduction of DNNs with early realignment comes with a cost of implementing and performing realignment, which is of course not a standard DNN training technique. Realignment requires specializing DNN training to the speech recognition domain, but any modern LVCSR system should already contain infrastructure to generate a forced alignment from an HMM-DNN system. Overall, we conclude that early realignment is an effective technique to improve performance of DNN acoustic models with minimal additional training time.

VI. Comparing DNNs, DCNNs, and DLUNNs on Switchboard

The experiments thus far modify DNN training by adding various forms of regularization. We now experiment with alternative neural network architectures – deep convolutional neural networks (DCNNs) and deep local untied neural networks (DLUNNs).

A. Experiments

We trained DCNN and DLUNN acoustic models using the same Switchboard training data as used for our DNN acoustic model experiments to facilitate direct comparisons across architectures. We evaluate filter bank features in addition to the fMLLR features used in DNN training because filter bank features have meaningful spectro-temporal dimensions for local receptive field computations. All models have five hidden layers and were trained using Nesterov's accelerated gradient with a smoothly increasing momentum schedule capped at 0.95 and a step size of 0.01, halving the step size after each epoch.

For our DCNN and DLUNN acoustic models we chose a receptive field of 9×9 and non-overlapping pooling regions of dimension 1×3 (time by frequency). Our models with two convolutional layers have the same first layer filter and pooling sizes. The second layer uses a filter size of 3×3 and does not use pooling. These parameters were selected using results from preliminary experiments as well as results from previous work [28].

In the DCNNs one convolutional layer was used followed by four densely connected layers with equal number of hidden units, and similarly for the DLUNNs. Map depth and number of hidden units were selected such at all models have approximately 36M parameters. For DCNNs, the convolutional first layer has a map depth of 128 applied to an input with ± 10 frame context. The following dense hidden layers each have 1,240 hidden units. Our 2 convolutional layer DCNN uses 128 feature maps in both convolutional layers and 3 dense layers with 1,240 hidden units each. All DLUNNs use 108 filters at each location in the first layer, and 4 hidden layers each with 1,240 hidden units.

The filter bank and fMLLR features are both 40-dimensional. We ran initial experiments convolving filters

TABLE III
PERFORMANCE COMPARISON OF DNNS, DEEP CONVOLUTIONAL NEURAL NETWORKS (DCNNS), AND DEEP LOCAL UNTIED NEURAL NETWORKS (DLUNNS). WE EVALUATE CONVOLUTIONAL MODELS WITH ONE LAYER OF CONVOLUTION (DCNN) AND TWO LAYERS OF CONVOLUTION (DCNN2). WE COMPARE MODELS TRAINED WITH FMLLR FEATURES AND FILTER BANK (FBANK) FEATURES. NOTE THAT A CONTEXT WINDOW OF FMLLR FEATURES HAS A TEMPORAL DIMENSION BUT NO MEANINGFUL FREQUENCY DIMENSION WHEREAS FBANK FEATURES HAVE MEANINGFUL TIME-FREQUENCY AXES. AS AN ADDITIONAL CONTROL WE TRAIN A DCNN ON FEATURES WHICH ARE RANDOMLY PERMUTED TO REMOVE MEANINGFUL COHERENCE IN BOTH THE TIME AND FREQUENCY AXES (FBANK-P AND FMLLR-P). WE REPORT PERFORMANCE ON BOTH THE HUB5 EVAL2000 TEST SET (EV) WHICH CONTAINS SWITCHBOARD (SWBD) AND CALLHOME (CH) EVALUATION SUBSETS.

Model	Features	Acc(%)	SWBD WER	CH WER	EV WER
GMM	fMLLR	N/A	21.7	36.1	29.0
DNN	fMLLR	60.8	14.9	27.4	21.2
DNN	FBank	51.7	16.5	31.6	24.1
DCNN	fMLLR	59.3	15.8	28.3	22.0
DCNN	FBank	53.0	15.8	28.7	22.3
DCNN	fMLLR-P	59.0	15.9	28.6	22.4
DCNN	FBank-P	50.7	17.2	32.1	24.7
DCNN2	fMLLR	58.8	15.9	28.3	22.2
DCNN2	FBank	53.0	15.6	28.3	22.1
DLUNN	fMLLR	61.2	15.2	27.4	21.3
DLUNN	FBank	53.0	16.1	29.3	22.8

along frequency only, pooling along both frequency and time, and overlapping pooling regions, but did not find that these settings gave better performance. We ran experiments with a context window of ± 20 frames but found results to be worse than results obtained with a context window of ± 10 frames, so we report only the ± 10 frame context results.

B. Results

Table III shows the frame-level and final system performance results for acoustic models built from DNNs, DCNNs, and DLUNNs. When using filter bank features, DCNNs and DLUNNs both achieve improvements over DNNs. DCNN models narrowly outperform DLUNN models. For locally connected acoustic models it appears that the constraint of tied weights in convolutional models is advantageous as compared to allowing a different set of localized receptive fields to be learned at different time-frequency regions of the input.

DLUNNs outperform DCNNs in experiments with fMLLR features. Indeed, the DLUNN performs about as well as the DNN. The DCNN is harmed by the lack of meaningful relationships along the frequency dimension of the input features, whereas the more flexible architecture of the DLUNN is able to learn useful first layer parameters. We also note that our fMLLR features yield much better performance for all models as compared with models trained on filter bank features.

In order to examine how much benefit using DCNNs to leverage local correlations in the acoustic signal yields, we ran control experiments with filter bank features randomly permuted along both the frequency and time axes. The results show that while this harms performance the convolutional architecture can still obtain fairly competitive word error rates. This control experiment confirms that locally connected models do indeed leverage localized properties of the input features to achieve improved performance.

C. Discussion

While DCNN and DLUNN models are promising as compared to DNN models on filter bank features, our results with filter bank features are overall worse than results from models utilizing fMLLR features.

Note that the filter bank features we used are fairly simple as compared to our fMLLR features as the filter bank features do not contain significant post-processing for speaker adaptation. While performing such feature transformations may give improved performance, they call into question the initial motivation for using DCNNs to automatically discover invariance to gender, speaker and time-frequency distortions. The fMLLR features we compare against include much higher amounts of specialized post-processing, which appears beneficial for all neural network architectures we evaluated. This confirms recent results from previous work, which found that DCNNs alone are not typically superior to DNNs but can complement a DNN acoustic model when both are used together, or achieve competitive results when increased amounts of post-processing are applied to filter bank features [29]. In summary, we conclude that DCNNs and DLUNNs are not sufficient to replace DNNs as a default, reliable choice for acoustic modeling network architecture. We additionally conclude that DLUNNs warrant further investigation as alternatives to DCNNs for acoustic modeling tasks.

VII. COMBINED LARGE CORPUS

On the Switchboard 300 hour corpus we observed limited benefits from increasing DNN model size for acoustic modeling, even with a variety of techniques to improve generalization performance. We next explore DNN performance using a substantially larger training corpus. This set of experiments explores how we expect DNN acoustic models to behave when training set size is not a limiting factor. In this setting, overfitting with large DNNs should be less of a problem and we can

more thoroughly explore architecture choices in large DNNs rather than regularization techniques to reduce over-fitting and improve generalization with a small training corpus.

A. Baseline HMM system

To maximize the amount of training data for a conversational speech transcription task, we combine the Switchboard corpus with the larger Fisher corpus [69]. The Fisher corpus contains approximately 2,000 hours of training data, but has transcriptions which are slightly less accurate than those of the Switchboard corpus.

Our baseline GMM acoustic model was trained on features that are obtained by splicing together 7 frames (3 on each side of the current frame) of 13-dimensional MFCCs (C0-C12) and projecting down to 40 dimensions using linear discriminant analysis (LDA). The MFCCs are normalized to have zero mean per speaker[2]. After obtaining the features with LDA, we also use a single semi-tied covariance (STC) transform on the features. Moreover, speaker adaptive training (SAT) is done using a single feature-space maximum likelihood linear regression (fMLLR) transform estimated per speaker. The models trained on the full combined Fisher+Switchboard training set contain 8725 tied triphone states and 3.2M Gaussians.

The language model in our baseline system is trained on the combination of the Fisher transcripts and the Switchboard Mississippi State transcripts. Kneser-Ney smoothing was applied to fine-tune the back-off probabilities to minimize the perplexity on a held out set of 10K transcript sentences from Fisher transcripts. In preliminary experiments we interpolated the transcript-derived language model with a language model built from a large collection of web page text, but found no gains as compared with using the transcript-derived language model alone.

We use two evaluation sets for all experiments on this corpus. First, we use the same Hub5'00 (Eval2000) corpus used to evaluate systems on the Switchboard 300hr task. This evaluation set serves as a reference point to compare systems built on our combined corpus to those trained on Switchboard alone. Second, we use the RT-03 evaluation set which is more frequently used in the literature to evaluate Fisher-trained systems. Performance of the baseline HMM-GMM system is shown in Table IV and Table V. [3]

B. Optimization Algorithm Choice

To avoid exhaustively searching over all DNN architecture and training parameters simultaneously, we first establish the impact of optimization algorithm choice while holding the DNN architecture fixed. We train networks with the two optimization algorithms described in Section IV-E to determine which optimization algorithm to use in the rest of the experiments on this corpus.

[2]This is done strictly for each individual speaker with our commit r4258 to the Kaldi recognizer. We found this to work slightly better than normalizing on a per conversation-side basis.

[3]The implementation of our baseline HMM-GMM system is available in the Kaldi project repository as example recipe `fisher_swbd` (revision: r4340).

1) Experiments: We train several DNNs with five hidden layers, where each layer has 2,048 hidden units. This results in DNNs with roughly 36M total free parameters, which is a typical size for acoustic models used for conversational speech transcription in the research literature. For both the classical momentum and Nesterov's accelerated gradient optimization techniques the two key hyper-parameters are the initial learning rate ϵ and the maximum momentum μ_{max}. In all cases we decrease the learning rate by a factor of 2 every 200,000 iterations. This learning rate annealing was chosen after preliminary experiments, and overall performance does not appear to be significantly affected by annealing schedule. It is more common to anneal the learning rate after each pass through the dataset. Because our dataset is quite large we found that annealing only after each epoch leads to much slower convergence to a good optimization solution.

2) Results: Table IV shows both WER performance and classification accuracy of DNN-based ASR systems with various optimization algorithm settings. We first evaluate the effect of optimization algorithm choice. We evaluated DNNs with $\mu_{max} \in 0.9, 0.95, 0.99$ and $\epsilon \in \{0.1, 0.01, 0.001\}$. For both optimization algorithms DNNs achieve the best performance by setting $\mu_{max} = 0.99$ and $\epsilon = 0.01$.

In terms of frame level accuracy the NAG optimizer narrowly outperforms the CM optimizer, but WER performance across all evaluation sets are nearly identical. For both optimization algorithms a high value of μ_{max} is important for good performance. Note most previous work in hybrid acoustic models use CM with $\mu_{max} = 0.90$, which does not appear to be optimal in our experiments. We also found that a larger initial learning rate was beneficial. We ran experiments using $\epsilon \geqslant 0.05$ but do not report results because the DNNs diverged during the optimization process. Similarly, all models trained with $\epsilon = 0.001$ had WER more than 1% absolute higher on the EV test set as compared to the same architecture trained with $\epsilon = 0.01$. We thus omit the results for models trained with $\epsilon = 0.001$ from our results table.

For the remainder of our experiments we use the NAG optimizer with $\mu_{max} = 0.99$ and $\epsilon = 0.01$. These settings achieve the best performance overall in our initial experiments, and generally we have found the NAG optimizer to be somewhat more robust than the CM optimizer in producing good parameter solutions.

C. Scaling Total Number of DNN Parameters

We next evaluate the performance of DNNs as a function of the total number of model parameters while keeping network depth and optimization parameters fixed. This approach directly assesses the hypothesis of improving performance as a function of model size when there is sufficient training data available. We train DNNs with 5 hidden layers, and keep the number of hidden units constant across each hidden layer. Varying total free parameters thus corresponds to adding hidden units to each hidden layer. Table V shows the frame classification and WER performance of 5 hidden layer DNNs containing 36M, 100M, 200M, and 400M total free parameters. Because it can be difficult to exactly reproduce

TABLE IV
RESULTS FOR DNNS OF THE SAME ARCHITECTURE TRAINED WITH VARYING OPTIMIZATION ALGORITHMS. PRIMARILY WE COMPARE STOCHASTIC GRADIENT USING CLASSICAL MOMENTUM (CM) AND NESTEROV'S ACCELERATED GRADIENT (NAG). WE ADDITIONALLY EVALUATE MULTIPLE SETTINGS FOR THE MAXIMUM MOMENTUM (μ_{max}). THE TABLE CONTAINS RESULTS FOR ONLY ONE LEARNING RATE ($\epsilon = 0.01$) SINCE IT PRODUCES THE BEST PERFORMANCE FOR ALL SETTINGS OF OPTIMIZATION ALGORITHM AND MOMENTUM. WE REPORT PERFORMANCE ON BOTH THE HUB5 EVAL2000 TEST SET (EV) WHICH CONTAINS SWITCHBOARD (SWBD) AND CALLHOME (CH) EVALUATION SUBSETS. WE ALSO EVALUATE PERFORMANCE ON THE RT03 (RT03) SWITCHBOARD TEST SET FOR COMPARISON WITH FISHER CORPUS SYSTEMS.

Optimizer	μ_{max}	Acc(%)	SWBD WER	CH WER	EV WER	RT03 WER
GMM	N/A	N/A	21.9	31.9	26.9	39.5
CM	0.90	52.51	18.3	27.3	22.8	39.0
CM	0.95	54.20	17.1	25.6	21.4	38.1
CM	0.99	55.26	16.3	24.8	20.6	37.5
NAG	0.90	53.18	18.0	26.7	22.3	38.5
NAG	0.95	54.27	17.2	25.8	21.5	39.6
NAG	0.99	55.39	16.3	24.7	20.6	37.4

DNN optimization procedures, we make our DNN training code available online [4]. Our DNN training code comprises only about 300 lines of Python code in total, which should facilitate easy comparison to other DNN training frameworks.

Overall, the 400M parameter model performs best in terms of both frame classification and WER across all evaluation sets. Unlike with our smaller Switchboard training corpus experiments, increasing DNN model size does not lead to significant over-fitting problems in WER. However, the gain from increasing model size from 36M to 400M, more than a 10x increase, is somewhat limited. On the Eval2000 evaluation set we observe a 3.8% relative gain in WER from the 100M DNN as compared to the 36M DNN. When moving from the 100M DNN to the 200M DNN there is relative WER gain of 2.5%. Finally the model size increase from 200M to 400M total parameters yields a relative WER gain of 1%. There are clearly diminishing returns as we increase model size. The trend of diminishing relative gains in WER also occurs on the RT03 evaluation set, although relative gains on this evaluation set are somewhat smaller overall.

Frame classification rates on this corpus are much lower overall as compared with our Switchboard corpus DNNs. We believe this corpus is more challenging due to more overall acoustic variation, and errors induced by quick transcriptions. Even our largest DNN leaves room for improvement in terms of frame classification. In Section VIII we explore more thoroughly the frame classification performance of the DNNs presented here.

D. Number of Hidden Layers

We next compare performance of DNN systems while keeping total model size fixed and varying the number of hidden layers in the DNN. The optimal architecture for a neural network may change as the total number of model parameters changes. There is no a priori reason to believe that 5 hidden layers is optimal for all model sizes. Furthermore, there are no good general heuristics to select the number of hidden layers for a particular task. Table V shows DNN system performance for DNNs with 1, 3, 5, and 7 hidden layers for DNNs of at multiple total parameter counts.

The most striking distinction in terms of both frame classification and WER is the performance gain of deep models versus those with a single hidden layer. Single hidden layer models perform much worse than DNNs with 3 hidden layers or more. Among deep models there are much smaller gains as a function of depth. Models with 5 hidden layers show a clear gain over those with 3 hidden layers, but there is little to no gain from a 7 hidden layer model when compared with a 5 hidden layer model. These results suggest that for this task 5 hidden layers may be deep enough to achieve good performance, but that DNN depth taken further does not increase performance. It's also interesting to note that DNN depth has a much larger impact on performance than total DNN size. For this task, it is much more important to select an appropriate number of hidden layers than it is to choose an appropriate total model size.

For each total model size there is a slight decrease in frame classification in 7 layer DNNs as compared with 5 hidden layer DNNs. This trend of decreasing frame-level performance is also present in the training set, which suggests that as networks become very deep it is more difficult to minimize the training objective function. This is evidence for a potential confounding factor when building DNNs. In theory deeper DNNs should be able to model more complex functions than their shallower counterparts, but in practice we found that depth can act as a regularizer due to the difficulties in optimizing very deep models.

VIII. WER AND FRAME CLASSIFICATION ERROR ANALYSIS

We now decompose our task performance metrics of frame classification accuracy and WER into their constituent components to gain a deeper understanding of how models compare to one another. This analysis attempts to uncover differences in models which achieve similar aggregate performance. For example, two systems which have the same final WER may have different rates of substitutions, deletions, and insertions – the constituent components of the WER metric.

[4]For DNN training code, see <upon acceptance>

TABLE V
RESULTS FOR DNNS OF VARYING TOTAL MODEL SIZE AND DNN DEPTH. WE REPORT PERFORMANCE ON BOTH THE HUB5 EVAL2000 TEST SET (EV) WHICH CONTAINS SWITCHBOARD (SWBD) AND CALLHOME (CH) EVALUATION SUBSETS. WE ALSO EVALUATE PERFORMANCE ON THE RT03 (RT03) SWITCHBOARD TEST SET FOR COMPARISON WITH FISHER CORPUS SYSTEMS. WE ADDITIONALLY REPORT FRAME-LEVEL CLASSIFICATION ACCURACY (ACC) ON A HELD OUT TEST SET TO COMPARE DNNS AS CLASSIFIERS INDEPENDENT OF THE HMM DECODER.

# Params	Nun. Layers	Layer Size	Acc(%)	SWBD WER	CH WER	EV WER	RT03 WER
GMM	N/A	N/A	N/A	21.9	31.9	26.9	39.5
36M	1	3803	49.38	21.0	30.4	25.8	43.2
36M	3	2480	54.78	17.0	25.8	21.4	38.2
36M	5	2048	55.37	16.2	24.7	20.6	37.4
36M	7	1797	54.99	16.3	24.7	20.7	37.3
100M	1	10454	50.82	19.8	29.1	24.6	42.4
100M	3	4940	56.02	16.3	24.8	20.6	37.3
100M	5	3870	56.62	15.8	23.8	19.8	36.7
100M	7	3309	56.59	15.7	23.8	19.8	36.4
200M	1	20907	51.29	19.6	28.7	24.3	42.8
200M	3	7739	56.58	16.0	24.0	20.1	37.0
200M	5	5893	57.36	15.3	23.1	19.3	36.0
200M	7	4974	57.28	15.3	23.3	19.3	36.2
400M	5	8876	57.70	15.0	23.0	19.1	35.9

Figure 6 shows decomposed WER performance of HMM-DNN systems of varying DNN size. Each HMM-DNN system uses a DNN with 5 hidden layers, these are the same HMM-DNN systems reported in Table V. We see that decreases in overall WER as a function of DNN model size are largely driven by lower substitution rates. Insertions and deletions remain relatively constant across systems, and are generally the smaller components of overall WER. Decreased substitution rates should be a fairly direct result of improving acoustic model quality as the system becomes more confident in matching audio features to senones. While the three WER sub-components are linked, it is possible that insertions and deletions are more an artifact of other system shortcomings such as out of vocabulary words (OOVs) or a pronunciation dictionary which does not adequately capture pronunciation variations.

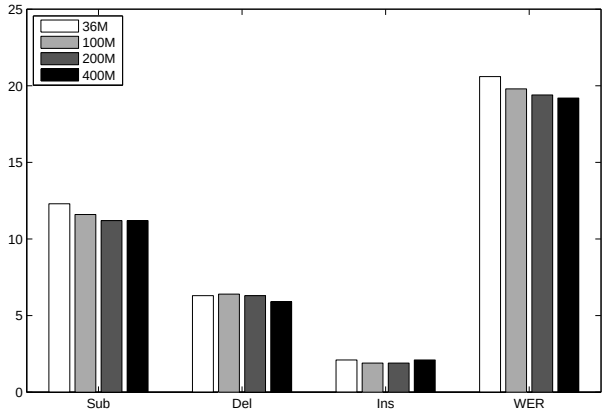

Fig. 6. Eval2000 WER of 5 hidden layer DNN systems of varying total parameter count. WER is broken into its sub-components – insertions, substitutions, and deletions.

We next analyze performance in terms of frame-level classification grouped by phoneme. When understanding senone classification we can think of the possible senone labels as leaves from a set of trees. Each phoneme acts as the root of a different tree, and the leaves of a tree correspond to the senones associated with a the tree's base phoneme.

Figure 7 shows classification percentages of senones grouped by their base phoneme. The DNNs analyzed are the same 5 hidden layer models presented in our WER analysis of Figure 6 and Table V. The total height of each bar reflects its percentage of occurrence in our data. Each bar is then broken into three components – correct classifications, errors within the same base phoneme, and errors outside the base phoneme. Errors within the base phoneme correspond the examples where the true label is a senone from a particular base phone, e.g. *ah*, but the network predicts an incorrect senone label also rooted in *ah*. The other type of error possible is predicting a senone from a different base phoneme. Together these three categories, correct, same base phone, and different base phone, additively combine to form the total set of senone examples for a given base phone.

The rate of correct classifications is non-decreasing as a function of DNN model size for each base phoneme. The overall increasing accuracy of larger DNNs comes from small correctness increases spread across many base phonemes. Across phonemes we see substantial differences in within-base-phoneme versus out-of-base-phoneme error rates. For example, the vowel *iy* has a higher rate of within-base-phoneme errors as compared to the fairly similar vowel *ih*. Similarly, the consonants *m*, *k*, and *d* have varying rates of within-base versus out-of-base errors despite having similar total rates of base phoneme occurrence in the data. We note that our DNNs generally exhibit similar error patterns

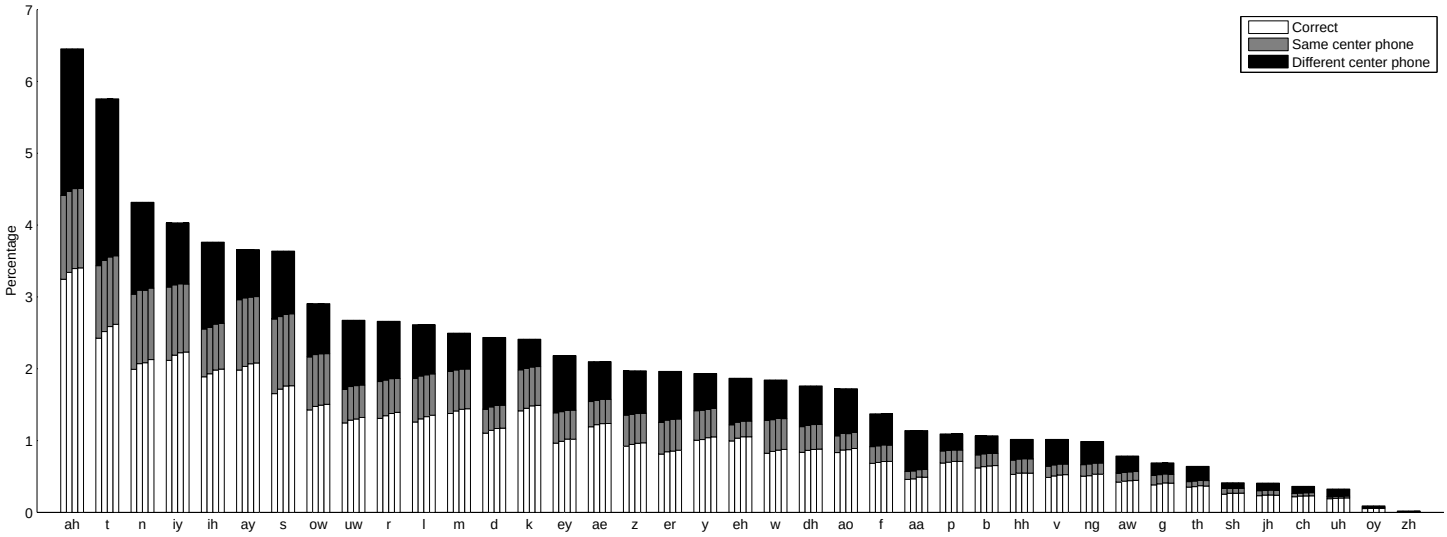

Fig. 7. Senone accuracy of 5 hidden layer DNN systems of varying total parameter count. Accuracy is grouped by base phone and we report the percentage correct, mis-classifications which chose a senone of the same base phone, and mis-classifications which chose a senone of a different base phone. The total size of the combined bar indicates the occurrence rate of the base phone in our data set. Each base phone has five bars, each representing the performance of a different five layer DNN. The bars show performance of DNNs of size 36M 100M 200M and 400M from left to right. We do not show the non-speech categories of silence, laughter, noise, or OOV which comprise over 20% of frames sampled.

to those observed with DNN acoustic models on smaller corpora [70]. However, due to the challenging nature of our corpus we observe overall lower phone accuracies than those found in previous work. Performance as a function of model size appears to change gradually and fairly uniformly across phonemes, rather than larger models improving upon only specific phonemes, perhaps at the expense of performance on others.

IX. Analyzing Coding Properties

Our experiments so far focus on task performance at varying levels of granularity. These metrics address the question of *what* DNNs are capable of doing as classifiers and when integrated with HMM speech decoding infrastructure. However, we have not yet completely addressed the question of *how* various DNN architectures achieve their various levels of task performance. While the DNN computation equations presented in Section IV describe the algorithmic steps necessary to compute predictions, there are many possible settings of the free parameters in a model. In this section we offer a descriptive analysis of how our trained DNNs encode information. This analysis aims to uncover quantifiable differences in how models of various sizes and depths encode input data and transform it to make a final prediction.

A. Sparsity and Dispersion

Our first analysis focuses on the sparsity patterns of units with each hidden layer of a DNN. We compute the empirical *lifetime sparsity* of each hidden unit by forward propagating a set of 512,000 examples through the DNN. We consider a unit as a active when its output is non-zero, and compute the fraction of examples for which a unit is active as its *lifetime activation probability*. This value gives the empirical probability that a particular unit will activate given a random input drawn from our sample distribution. For each hidden layer of a network, we can plot all hidden units' lifetime activation probabilities sorted in decreasing order to get a sense for the distribution of activation probabilities within a layer. This plotting technique, sometimes called a scree plot, helps us understand how information coding is distributed across units in a hidden layer. Figure 8 shows a set of scree plots for 5 hidden layer DNNs of varying total model size.

From a coding theory perspective, researchers often discuss DNNs as learning efficient codes which are both sparse and dispersed. Sparsity generally refers to relatively few hidden units in a hidden layer being active in response to an input. Sparsity is efficient and seems natural given modern DNN structures in which hidden layer size is often much larger than input vector dimensionality. Dispersion refers to units within a hidden layer equally sharing responsibility for coding inputs. A representation with perfect dispersion would appear flat in a scree plot. A scree plot also visualizes sparsity as the average height of representation units on the y axis.

Generally we see that in all model sizes sparsity increases in deeper layers of the DNN. The first hidden layer is noticeably more active on average as compared with every other layer in the DNN, in most cases by almost a factor of two. Beyond the first layer, activation probability per layer decreases slightly as we look at deeper layers of the DNN. The changes in activation probability per layer within deeper hidden layers are fairly minor, which suggest that a representation is transformed but not continually compressed.

Dispersion is similar within layers of a particular DNN size.

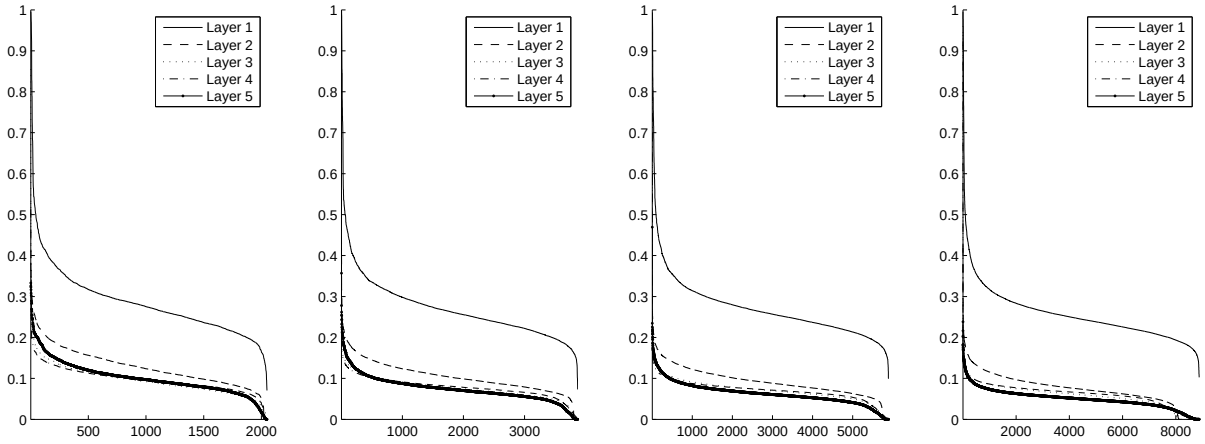

Fig. 8. Empirical activation probability of hidden units in each hidden layer layer of 5 hidden layer DNNs. Hidden units (x axis) are sorted by their probability of activation. We consider any positive value as active ($h(x) > 0$). Each sub-figure corresponds to a different model size of 36M, 100M, and 200M total parameters from left to right.

Generally the representations appear fairly disperse, with a mostly flat curve for each hidden layer and only a few units which are on or off for a large percentage of inputs at each tail. There does appear to be a slight trend of increasing dispersion in deeper layers of the DNN, especially in larger models.

Most importantly, we do not observe a significant set of permanently inactive units as DNNs grow in total number of parameters. In larger DNNs the representation remains fairly disperse, with only a small set of units which are active for less than 1% of inputs. This is an important metric because adding more parameters to a DNN is only useful in so far as those parameters are actually used in encoding and transforming inputs.

Given the task performance differences observed as a function of DNN depth for a fixed number of total DNN parameters, we also compare scree plots as a function of DNN depth to better understand their coding properties. Figure 9 shows scree plots for DNNs with 1, 3, 5, and 7 hidden layers for DNNs of total size 36M, 100M, and 200M. We observe a general trend of average activation probability decreasing in subsequent hidden layers of DNNs at each size. This is not true, however, for models with 7 hidden layers, which have slightly less sparse activations on average in layers 6 and 7 as compared to layer 5. As we compare models across total model size we find that larger models are more sparse than smaller models. Larger models also tend to be slightly more dispersed on average compared with smaller models.

B. Code Length

Our sparsity and dispersion metrics serve as indicators for how hidden units within each layer behave. We now focus on *code length*, which analyzes each hidden layer as a transformed representation of the input rather than focusing on individual units with each hidden layer. For a given input we compute the number of non-zero hidden unit activations in a hidden layer. We can then compute the average code length for each hidden layer over a large sample of inputs from our dataset. Figure 10 shows average code length for each hidden layer of DNNs of varying depth and total size.

As we compare code length across models of varying total parameter size, we see that larger DNNs use more hidden units per layer to encode information at each hidden layer. This trend is especially evident in the first hidden layer, where 100M parameter models use nearly twice the code length as compared to 36M models. In deeper layers, we again observe that models with more parameters have greater code length. It is unclear to what extend the longer codes are capturing more information about an input, which in turn should enable greater classification accuracy, versus redundancy where multiple hidden units encode overlapping information.

Code length in deeper versus more shallow models of the same total size exhibit an interesting trend. DNNs of increasing depth show a generally decreasing or constant code length per layer, except in the case of our 7 hidden layer DNNs. In 7 hidden layer DNNs, the deepest models we trained, code length decreases until it reaches a minimum at layer 5, but then increases in layers 6 and 7. This trend is evident in models of 36M, 100M, and 200M total parameters. We note that this trend of decreasing code length followed by increasing code length is correlated with the lack of improvement of 7 hidden layer models as compared to 5 hidden layer models. More experiments are needed to establish whether code length in deeper models is more generally correlated to diminishing task performance.

X. Conclusion

The multi-step process of building neural network acoustic models comprises a large design space with a broad range of previous work. Our work sought to address which of the most fundamental DNN design decisions are most relevant for final ASR system performance. We found that increasing model size and depth are simple but effective ways to improve

WER performance, but only up to a certain point. For the Switchboard corpus, we found that regularization can improve the performance of large DNNs which otherwise suffer from overfitting problems. However, a much larger gain was achieved by utilizing the combined 2,100hr training corpus as opposed to applying regularization with less training data.

Our experiments suggest that the DNN architecture is quite competitive with specialized architectures such as DCNNs and DLUNNs. The DNN architecture outperformed other architecture variants in both frame classification and final system WER. While previous work has used more specialized features with locally connected models, we note that DNNs enjoy the benefit of making no assumptions about input features having meaningful time or frequency properties. This enables us to build DNNs on whatever features we choose, rather than ensuring our features match the assumptions of our neural network. We found that DLUNNs performed slightly better and DCNNs, and may be an interesting approach for specialized acoustic modeling tasks. For example, locally untied models may work well for robust or reverberant recognition tasks where particular frequency ranges experience interference or distortion.

We trained DNN acoustic models with up to 400M parameters and 7 hidden layers, comprising some of the largest models evaluated to date for acoustic modeling. When trained with the simple NAG optimization procedure, these large DNNs achieved clear gains on both frame classification and WER when the training corpus was large. An analysis of performance and coding properties revealed a fairly gradual change in DNN properties as we move from smaller to larger models, rather than finding some phase transition where large models begin to encode information differently from smaller models. Overall, total network size, not depth, was the most critical factor we found in our experiments. Depth is certainly important with regards to having more than one hidden layer, but differences among DNNs with multiple hidden layers were fairly small with regards to all metrics we evaluated. At a certain point it appears that increasing DNN depth yields no performance gains, and may indeed start to harm performance. When applying DNN acoustic models to new tasks it appears sufficient to use a fixed optimization algorithm, we suggest NAG, and cross-validate over total network size using a DNN of at least three hidden layers, but no more than five. Based on our results, this procedure should instantiate a reasonably strong baseline system for further experiments, by modifying whatever components of the acoustic model building procedure researchers choose to explore.

Finally, we note that a driving factor in the uncertainty around DNN acoustic model research stems from training the acoustic model in isolation from the rest of the larger ASR system. All models trained in this paper used the cross entropy criterion, and did not perform as well as DNNs trained with discriminative loss functions in previous work. We hypothesize that large DNNs will become increasingly useful as researchers invent loss functions which entrust larger components of the ASR task to the neural network. This allows the DNN to utilize its function fitting capacity to do more than simply map acoustic inputs to HMM states.

We believe a better understanding of task performance and coding properties can guide research on new, improved DNN architectures and loss functions. We trained DNNs using approximately 300 lines of Python code, demonstrating the feasibility of fairly simple architectures and optimization procedures to achieve good system performance. We hope that this serves as a reference point to improve communication and reproducibility in the now highly active research area of neural networks for speech and language understanding.

REFERENCES

[1] G. Dahl, D. Yu, L. Deng, and A. Acero, "Context-Dependent Pre-trained Deep Neural Networks for Large Vocabulary Speech Recognition," *IEEE Transactions on Audio, Speech, and Language Processing*, 2011.

[2] 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*, vol. 29, no. November, pp. 82–97, 2012.

[3] B. Kingsbury, T. Sainath, and H. Soltau, "Scalable minimum Bayes risk training of deep neural network acoustic models using distributed hessian-free optimization," in *Interspeech*, 2012.

[4] K. Veselỳ, A. Ghoshal, L. Burget, and D. Povey, "Sequence-discriminative training of deep neural networks," in *INTERSPEECH*, 2013, pp. 2345–2349.

[5] H. Bourlard and N. Morgan, *Connectionist Speech Recognition: A Hybrid Approach*. Norwell, MA: Kluwer Academic Publishers, 1993.

[6] H. Hermansky, D. Ellis, and S. Sharma, "Tandem connectionist feature extraction for conventional hmm systems," in *ICASSP*, vol. 3. IEEE, 2000, pp. 1635–1638.

[7] S. Renals, N. Morgan, H. Bourlard, M. Cohen, and H. Franco, "Connectionist probability estimators in hmm speech recognition," *IEEE Transactions on Speech and Audio Processing*, vol. 2, no. 1, pp. 161–174, 1994.

[8] M. Gales and S. Young, "The application of hidden markov models in speech recognition," *Foundations and Trends in Signal Processing*, vol. 1, no. 3, pp. 195–304, 2008.

[9] S. Young, G. Evermann, M. Gales, T. Hain, D. Kershaw, X. Liu, G. Moore, J. Odell, D. Ollason, D. Povey *et al.*, *The HTK book*. Entropic Cambridge Research Laboratory Cambridge, 1997, vol. 2.

[10] G. Saon and J. Chien, "Large-vocabulary continuous speech recognition systems: A look at some recent advances," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 18–33, 2012.

[11] B. Gold, N. Morgan, and D. Ellis, *Speech and audio signal processing: processing and perception of speech and music*. John Wiley & Sons, 2011.

[12] D. Jurafsky and J. H. Martin, *Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition*. Prentice Hall, 2000.

[13] J. L. McClelland and J. L. Elman, "The trace model of speech perception," *Cognitive psychology*, vol. 18, no. 1, pp. 1–86, 1986.

[14] G. Hinton, S. Osindero, and Y. W. Teh, "A fast learning algorithm for deep belief nets," *Neural computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[15] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P. A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *The Journal of Machine Learning Research*, vol. 11, pp. 3371–3408, 2010.

[16] A. Mohamed, G. Dahl, and G. Hinton, "Acoustic modeling using deep belief networks," *Audio, Speech, and Language Processing, IEEE Transactions on*, no. 99, 2010.

[17] G. Dahl, D. Yu, and L. Deng, "Large vocabulary continuous speech recognition with context-dependent DBN-HMMs," in *ICASSP*, 2011.

[18] N. Jaitly, P. Nguyen, A. Senior, and V. Vanhoucke, "Application of pretrained deep neural networks to large vocabulary speech recognition," in *INTERSPEECH*, 2012.

[19] D. Yu and L. Deng, "Deep neural network-hidden markov model hybrid systems," in *Automatic Speech Recognition*. Springer, 2015, pp. 99–116.

[20] D. Yu, M. Seltzer, J. Li, J. Huang, and F. Seide, "Feature Learning in Deep Neural Networks Studies on Speech Recognition Tasks," in *ICLR*, 2013.

[21] F. Seide, G. Li, and D. Yu, "Conversational speech transcription using context-dependent deep neural networks." in *Interspeech*, 2011, pp. 437–440.

[22] A. Senior, G. Heigold, M. Bacchiani, and H. Liao, "Gmm-free dnn acoustic model training," in *ICASSP*. IEEE, 2014, pp. 5602–5606.

[23] G. Dahl, T. Sainath, and G. Hinton, "Improving Deep Neural Networks for LVCSR using Rectified Linear Units and Dropout," in *ICASSP*, 2013.

[24] M. Zeiler, M. Ranzato, R. Monga, M. Mao, K. Yang, Q. Le, P. Nguyen, A. Senior, V. Vanhoucke, J. Dean, and G. Hinton, "On Rectified Linear Units for Speech Processing," in *ICASSP*, 2013.

[25] A. Maas, A. Hannun, and A. Ng, "Rectifier Nonlinearities Improve Neural Network Acoustic Models," in *ICML Workshop on Deep Learning for Audio, Speech, and Language Processing*, 2013.

[26] N. Morgan, "Deep and wide: Multiple layers in automatic speech recognition," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 20, no. 1, pp. 7–13, 2012.

[27] O. Abdel-Hamid, A. rahman Mohamed, H. Jang, and G. Penn, "Applying convolutional neural networks concepts to hybrid nn-hmm model for speech recognition," in *ICASSP*, 2012.

[28] T. Sainath, B. Kingsbury, A. Mohamed, G. Dahl, G. Saon, H. Soltau, T. Beran, A. Aravkin, and B. Ramabhadran, "Improvements to Deep Convolutional Neural Networks for LVCSR," in *ASRU*, 2013.

[29] T. N. Sainath, B. Kingsbury, G. Saon, H. Soltau, A. rahman Mohamed, G. Dahl, and B. Ramabhadran, "Deep Convolutional Neural Networks for Large-Scale Speech Tasks," *Neural Networks*, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0893608014002007

[30] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. J. Lang, "Phoneme recognition using time-delay neural networks," *Acoustics, Speech and Signal Processing, IEEE Transactions on*, vol. 37, no. 3, pp. 328–339, 1989.

[31] T. Robinson and F. Fallside, "A recurrent error propagation network speech recognition system," *Computer Speech & Language*, vol. 5, no. 3, pp. 259–274, 1991.

[32] H. Sak, O. Vinyals, G. Heigold, A. Senior, E. McDermott, R. Monga, and M. Mao, "Sequence discriminative distributed training of long short-term memory recurrent neural networks," in *Interspeech*, 2014.

[33] H. Sak, A. Senior, and F. Beaufays, "Long short-term memory recurrent neural network architectures for large scale acoustic modeling," in *Interspeech*, 2014.

[34] A. Graves, A. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *ICASSP*. IEEE, 2013, pp. 6645–6649.

[35] O. Vinyals, S. V. Ravuri, and D. Povey, "Revisiting recurrent neural networks for robust asr," in *ICASSP*. IEEE, 2012, pp. 4085–4088.

[36] C. Weng, D. Yu, S. Watanabe, and B. Juang, "Recurrent deep neural networks for robust speech recognition," *ICASSP*, 2014.

[37] L. Deng, G. Hinton, and B. Kingsbury, "New types of deep neural network learning for speech recognition and related applications: An overview," in *ICASSP*. IEEE, 2013, pp. 8599–8603.

[38] L. B. Bahl, P. de Souza, and R. P. Mercer, "Maximum mutual information estimation of hidden markov model parameters for speech recognition," in *ICASSP*. IEEE, 1986.

[39] D. Povey, D. Kanevsky, B. Kingsbury, B. Ramabhadran, G. Saon, and K. Visweswariah, "Boosted mmi for model and feature-space discriminative training," in *ICASSP*. IEEE, 2008, pp. 4057–4060.

[40] V. Valtchev, J. Odell, P. C. Woodland, and S. J. Young, "Mmie training of large vocabulary recognition systems," *Speech Communication*, vol. 22, no. 4, pp. 303–314, 1997.

[41] J. Kaiser, B. Horvat, and Z. Kacic, "A novel loss function for the overall risk criterion based discriminative training of hmm models," in *ICSLP*, 2000.

[42] H. Su, G. Li, D. Yu, and F. Seide, "Error back propagation for sequence training of context-dependent deep networks for conversational speech transcription," in *ICASSP*, 2013, pp. 6664–6668.

[43] J. Martens, "Deep learning via hessian-free optimization," in *ICML*, 2010, pp. 735–742.

[44] J. Ngiam, A. Coates, A. Lahiri, B. Prochnow, Q. V. Le, and A. Y. Ng, "On optimization methods for deep learning," in *ICML*, 2011, pp. 265–272.

[45] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *The Journal of Machine Learning Research*, vol. 12, pp. 2121–2159, 2011.

[46] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the Importance of Momentum and Initialization in Deep Learning," in *ICML*, 2013.

[47] K. Oh and K. Jung, "Gpu implementation of neural networks," *Pattern Recognition*, vol. 37, no. 6, pp. 1311–1314, 2004.

[48] Z. Luo, H. Liu, and X. Wu, "Artificial neural network computation on graphic process unit," in *IJCNN*. IEEE, 2005, pp. 622–626.

[49] R. Raina, A. Madhavan, and A. Y. Ng, "Large-scale deep unsupervised learning using graphics processors." in *ICML*, vol. 9, 2009, pp. 873–880.

[50] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, Q. Le, M. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, and Y. Ng, "Large Scale Distributed Deep Networks," in *ICML*, 2012.

[51] T. Chilimbi, Y. Suzue, J. Apacible, and K. Kalyanaraman, "Project adam: Building an efficient and scalable deep learning training system," in *11th USENIX Symposium on Operating Systems Design and Implementation*, 2014, pp. 571–582.

[52] I. Chung, T. N. Sainath, B. Ramabhadran, M. Picheny, J. Gunnels, V. Austel, U. Chauhari, and B. Kingsbury, "Parallel deep neural network training for big data on blue gene/q," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2014, pp. 745–753.

[53] A. Coates, B. Huval, T. Wang, D. Wu, A. Ng, and B. Catanzaro, "Deep Learning with COTS HPC Systems," in *ICML*, 2013.

[54] D. Ellis and N. Morgan, "Size matters: An empirical study of neural network training for large vocabulary continuous speech recognition," in *ICASSP*. IEEE, 1999, pp. 1013–1016.

[55] C. S. Lindsey and T. Lindblad, "Survey of neural network hardware," in *SPIE Symposium on OE/Aerospace Sensing and Dual Use Photonics*. International Society for Optics and Photonics, 1995, pp. 1194–1205.

[56] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based Learning Applied to Document Recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[57] A. Krizhevsky, I. Sutskever, and G. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *NIPS*, 2012.

[58] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng, "Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations," in *ICML*. ACM, 2009, pp. 609–616.

[59] D. Plaut, "Experiments on learning by back propagation." 1986.

[60] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," *Parallel distributed processing: explorations in the microstructures of cognition, volume 2: psychological and biological models*, vol. 76, p. 1555, 1986.

[61] Y. Nesterov, "A method of solving a convex programming problem with convergence rate o (1/k2)," in *Soviet Mathematics Doklady*, vol. 27, no. 2, 1983, pp. 372–376.

[62] I. Sutskever, "Training recurrent neural networks," Ph.D. dissertation, University of Toronto, 2013.

[63] D. Povey, A. Ghoshal, G. Boulianne, L. Burget, O. Glembek, K. Veselý, N. Goel, M. Hannemann, P. Motlicek, Y. Qian, P. Schwarz, J. Silovsky, and G. Stemmer, "The kaldi speech recognition toolkit," in *ASRU*, 2011.

[64] H. Liao, E. McDermott, and A. Senior, "Large scale deep neural network acoustic modeling with semi-supervised training data for YouTube video transcription," in *ASRU*, 2013.

[65] T. Sainath, B. Kingsbury, V. Sindhwani, E. Arisoy, and B. Ramabhadran, "Low-Rank Matrix Factorization for Deep Neural Network Training with High-Dimensional Output Targets," in *ICASSP*, 2013.

[66] S. Wager, S. Wang, and P. Liang, "Dropout Training as Adaptive Regularization," in *NIPS*, 2013.

[67] R. Caruana, S. Lawrence, and L. Giles, "Overfitting in Neural Nets: Backpropagation, Conjugate Gradient, and Early Stopping," in *NIPS*, 2000.

[68] A. Saxe, J. McClelland, and S. Ganguli, "Learning Hierarchical Category Structure in Deep Networks," in *CogSci*, 2013.

[69] C. Cieri, D. Miller, and K. Walker, "The fisher corpus: a resource for the next generations of speech-to-text." in *LREC*, vol. 4, 2004, pp. 69–71.

[70] Y. Huang, D. Yu, C. Liu, and Y. Gong, "A comparative analytic study on the gaussian mixture and context dependent deep neural network hidden markov models," in *Interspeech*, 2014.

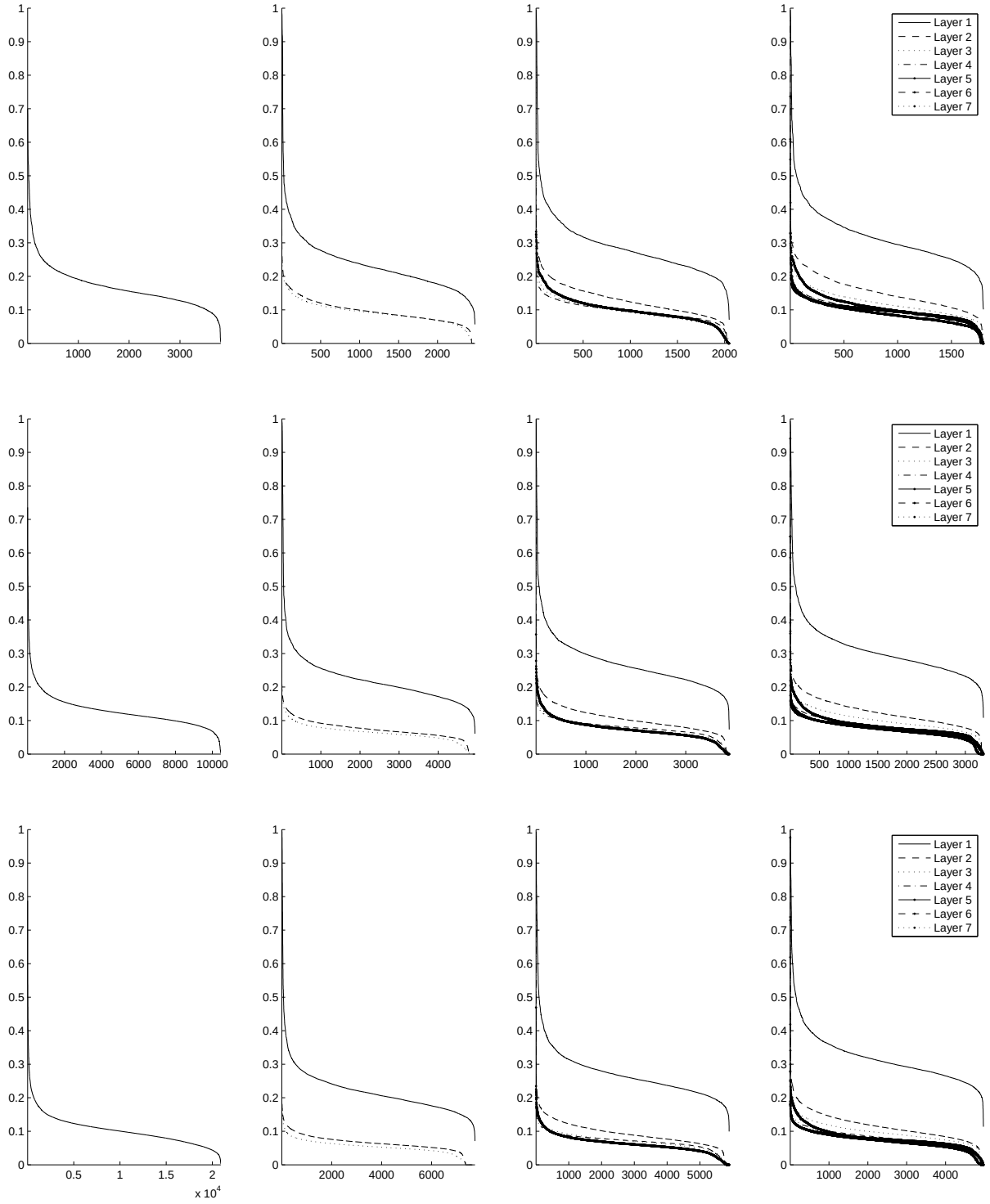

Fig. 9. Empirical activation probability of hidden units in each hidden layer layer of DNNs with varying numbers of hidden layers. Each row contains DNNs of 36M (top), 100M (middle), and 200M total parameters (bottom). From left to right, each sub-figure shows a DNN with 1, 3, 5, and 7 hidden layers. Hidden units (x axis) are sorted by their probability of activation. We consider any positive value as active ($h(x) > 0$).

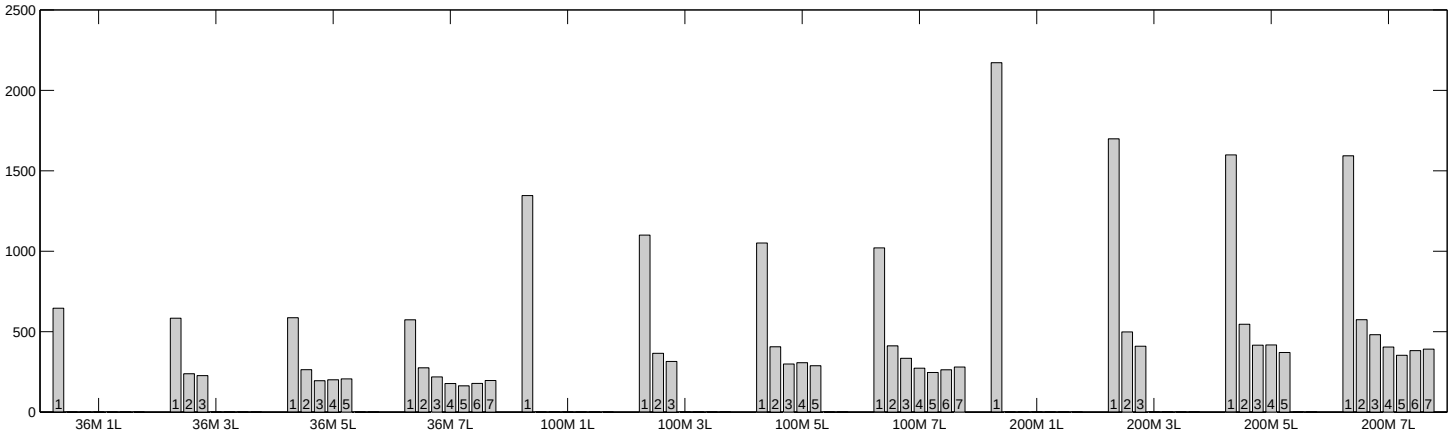

Fig. 10. Effective code length for each hidden layer in DNNs with varying total size and depth. We compute the number on non-zero hidden unit activations for a given input, and then average over a large sample of inputs. Plots show the average number of units active in each hidden layer of DNNs of varying depth and total size. Within each sub-plot layers are ordered left to right from first to final hidden layer.