DEEP NEURAL NETWORKS IN SPEECH RECOGNITION

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Andrew Lee Maas
March 2015

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Andrew Ng, Primary Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Dan Jurafsky

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Percy Liang

Approved for the Stanford University Committee on Graduate Studies.

Patricia J. Gumport, Vice Provost for Graduate Education

Abstract

Spoken language is an increasingly pervasive interface choice as computing devices permeate many aspects of daily life. Automatically understanding spoken language poses significant challenges because it requires both converting a speech signal into words and extracting meaning from the words themselves. Spoken language understanding tasks can roughly be broken into distinct components which perform (1) low-level processing of the audio signal, (2) speech transcription, and (3) natural language understanding.

We describe approaches to improving individual components for each sub-task associated with spoken language understanding. Our methods primarily rely on machine-learning-based approaches to replace hand-engineered approaches and consistently find that learning from data with minimal assumptions about a problem results in improved performance. In particular, we focus on neural network approaches to problems. Neural networks have seen a recent resurgence of interest thanks to their ability to scale to learn increasingly complex functions when more data becomes available. Neural networks have recently driven tremendous progress in the field of computer vision, where many tasks easily translate into classification and regression problems. In spoken language understanding, however, it is more difficult to define tasks which are easily formalized into problems for a neural network to solve. Our work integrates with these complex systems and shows that, like in computer vision, neural networks can significantly improve spoken language understanding systems.

To my parents–Dave & Judy, and my brothers–Colin, John, & Richard.

Acknowledgements

This thesis comes from a close collaboration with my advisor, Andrew Ng. Andrew has been an amazing mentor in the process of planning and solving research problems, and constantly encouraged me to work on challenging, impactful problems.

Much of the work on speech recognition in this thesis comes from close collaboration with Dan Jurafsky. Dan acted as a second advisor for much of my Ph.D. work, and navigating the field of speech recognition research wouldn't have been possible without him. I'm grateful to Dan for helping to find academic problems and research questions in areas dominated by benchmark performance results – learning his thought process greatly improved my work.

My earlier work in sentiment analysis benefited from a close collaboration with Chris Potts. Chris is one of the most enthusiastic researchers and prolific writers I have ever encountered. He set a model for interacting with students I strove to match throughout my time at Stanford.

I benefited greatly from conversations with Steven Wegmann on speech recognition research. He offered a valuable contrast in thinking about research problems, and also took the time to serve on my thesis defense committee.

Percy Liang's group meetings, advice, and teaching set a wonderful standard for doing theoretically challenging machine learning work. Percy's research and student culture have provided many thought-provoking conversations on neural network and natural language research.

Much of the work in this thesis would not have been possible without a fantastic set of student collaborators to help with running large-scale experiments and working with complicated speech recognition systems. In particular Awni Hannun, Tyler

O'Neil, Peng Qi, and Ziang Xie worked with me on tasks that are often considered too complex or time-consuming for academic researchers. I am extremely grateful for the time, effort, and enthusiasm they put forward in our work together.

I have been very lucky to have been part of Andrews research group, and to have had the chance to interact with his group of students, Adam Coates, Chuong Do (Tom), Brody Huval, J. Zico Kolter, Quoc Le, Honglak Lee, Jiquan Ngiam, Sameep Tandon, Morgan Quigley, Richard Socher, Tao Wang, and Ziang Xie.

I would like to thank my collaborators, for the inspiring intellectual interactions, their hard work and our great times together, Red Daly, Awni Hannun, Dan Huang, Dan Jurafsky, Quoc Le, Chris Lengerich, Stephen Miller, Andrew Ng, Patrick Nguyen, Peter Pham, Chris Potts, Peng Qi, Oriol Vinyals, and Ziang Xie.

Coming to Stanford wouldn't have been possible without the research and academic mentorship I received as an undergraduate at Carnegie Mellon. I had the privilege to work with Chris Atkeson, Drew Bagnell, Charles Kemp, and Tai Sing Lee whose time and patience helped transform the raw enthusiasm of a first semester freshman into research ability.

There are a number of friends at Stanford not directly related to my research, but their company contributed significantly to my experience here. I am very grateful for their time and friendship. I am also very grateful for those I've met in water polo and grappling workouts at Stanford. Exercise and competition in the pool and on the mat were always something to look forward to after long days in the lab.

To my parents, brothers, and extended family, thanks for your love and support throughout my life.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Automatically understanding spoken natural language enables a variety of applications. Spoken language is our most natural form of interaction when working with people, but has remained out of reach as a reliable interface between humans and machines. As mobile and ubiquitous computing expands into everyday life the desire for natural, low-friction interfaces increases. Building spoken language understanding (SLU) systems requires solving several sub-problems, with each posing a significant challenge in its own right. Currently deployed systems such as Apple Siri, Google Now, and Microsoft Cortana rely on fairly primitive speech interfaces characterized by a user speaking one of several possible commands, and the system executing the recognized command. These systems are powerful examples of the possibilities for spoken language interfaces, but fall short of our dream of directing a system by having a conversation in the way we might when working with a human trying to complete the same tasks. The shortcomings of modern speech interfaces are a direct result of the difficulty of designing automated SLU systems. In this thesis we build machine learning approaches to simplify and improve each component of a spoken language system to make progress towards natural, conversational speech interfaces for automated systems.

Spoken language systems break the overall SLU task into two broad components. First, the system must *transcribe* a raw audio signal into words. While formalizing this task is straightforward, transcribing spontaneous conversational speech is an open, challenging research problem. Speech transcription is a longstanding benchmark for progress in artificial intelligence, and has seen significant improvements over many years of research. The resulting systems are fairly functional, depending on the specifics of the input data, but are still far from human performance in recognizing conversational speech in noisy environments. Speech recognition systems are large, complex systems which encode many assumptions about speech and language which may not be correct. As a result, the transcription process is typically treated as a black box component, which limits possible benefits from reasoning about context during an understanding task.

Following transcription the system must perform some sort of *natural language understanding* which take words from a transcription component and derives higher-level information from them. The language understanding component is heavily dependent on whatever task the overall SLU system is built to achieve. For example, a voice command system might need to extract only the name of a phone contact, and whether to call or email that contact. In a more complex dialog system, we may wish to determine the next action for the system to take, whether to ask for clarification, or check that the user is not overly frustrated with the system.

In this thesis we address research questions at each level of the spoken language understanding pipeline. We show that we can improve individual system components using approaches based on neural networks. Deep neural networks (DNNs) have driven tremendous progress in computer vision in recent years. DNNs offer a powerful approach to learning increasingly complex functions from large datasets. Computer vision saw rapid success for DNNs as many computer vision tasks easily translate to classification and regression problems. In comparison, SLU tasks use more complex systems with several sub-components. Additionally, due to the nature of natural speech as a sequence of words it is difficult to formalize SLU tasks as straightforward classification or regression problems. By analyzing thoroughly the existing components of an SLU system we show ways to use DNNs and similar representation learning algorithms to improve performance while reducing system complexity. We focus primarily on transcription and ultimately arrive at a promising approach that may help merge transcription and language understanding into a single system to allow further customization and end-to-end learning for SLU tasks.

1.1 Contributions per Chapter

We present the following contributions in the chapters of this dissertation:

- **Chapter 2**

  Spoken language understanding systems must operate in real-world environments which may contain background noise and channel distortions. Before passing an audio signal to a speech transcription system it is often useful to apply noise reduction techniques to maximize the accuracy of the transcription process. We introduce a model which uses a deep recurrent auto encoder neural network to denoise input features for robust ASR. The model is trained on stereo (noisy and clean) audio features to predict clean features given noisy input. The model makes no assumptions about how noise affects the signal, nor the existence of distinct noise environments. Instead, the model can learn to model any type of distortion or additive noise given sufficient training data. We demonstrate the model is competitive with existing feature denoising approaches on the Aurora2 task, and outperforms a tandem approach where deep networks are used to predict phoneme posteriors directly.

- **Chapter 3**

  Speech recognition systems are often uncertain about the sequence of words output. We can represent the set of likely transcriptions using a word lattice to annotate endpoints of possible words in an utterance. Reasoning over possible transcriptions in a lattice allows us to leverage contextual information from a language understanding system to refine our speech recognition output. We introduce a model that maps variable-length word utterances to a word vector space using convolutional neural networks. Convolutional networks are a rich class of architecture that, through many nonlinear layers, can model complex functions of their input. Our approach models entire word acoustics rather than short windows as in previous work. We introduce the notion of mapping these word inputs to a word vector space, rather than trying to solve the massively multi-class problem of word classification. Regressing to word vectors offers many opportunities for further work in this domain, as many techniques exist to learn word vectors for different notions of word similarity. We experiment on hundreds of hours of broadcast news, and demonstrate our model can accurately recognize spoken words. Further, we use our model to build features for the SCARF speech recognition system and achieve an improvement in large vocabulary continuous speech recognition over a baseline system.

- **Chapter 4**

  This chapter presents our first experiments on first pass large vocabulary continuous speech recognition. Such systems comprise the transcription component of a spoken language understanding pipeline, and are typically the most complex components of such systems. Deep neural network acoustic models produce substantial gains in large vocabulary continuous speech recognition systems. Previous work with rectified linear (ReL) hidden units demonstrates additional gains in final system performance relative to more commonly used sigmoidal nonlinearities. In this work, we explore the use of deep rectifier networks as acoustic models for the 300 hour Switchboard conversational speech recognition task. Using simple training procedures without pretraining, networks with rectifier nonlinearities produce 2% absolute reductions in word error rates over their sigmoidal counterparts. We analyze hidden layer representations to quantify differences in how ReL units encode inputs as compared to sigmoidal units. Finally, we evaluate a variant of the ReL unit with a gradient more amenable to optimization in an attempt to further improve deep rectifier networks.

- **Chapter 5**

  This chapter offers an empirical investigation on which aspects of DNN acoustic model design are most important for speech recognition system performance. Building neural network acoustic models requires several design decisions including network architecture, size, and training loss function. 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.

- **Chapter 6**

  This chapter again focuses on the speech recognition component of spoken language understanding systems, but does so using a very different approach as compared to traditional speech recognition systems. We present an approach to speech recognition which uses only a neural network to map acoustic input to characters, a character-level language model, and a beam search decoding procedure. We build on the work of [46] who demonstrated a neural network approach to map input audio directly to characters using a sequence-based loss function. This approach eliminates much of the complex infrastructure of modern speech recognition systems, making it possible to directly train a speech recognizer using errors generated by spoken language understanding tasks. We build on this existing work by reasoning at the character level, rather than the word level. The system naturally handles out of vocabulary words and spoken word fragments. We demonstrate our approach using the challenging Switchboard telephone conversation transcription task, achieving a word error rate competitive with existing baseline systems. To our knowledge, this is the first entirely neural-network-based system to achieve strong speech transcription results on a conversational speech task. We analyze qualitative differences between transcriptions produced by our lexicon-free approach and transcriptions produced by a standard speech recognition system. Finally, we evaluate the impact of large context neural network character language models as compared to standard n-gram models within our framework.

- **Chapter 7**

  This chapter focuses on the understanding component of a spoken language understanding system for the specific application of sentiment analysis. Unsupervised vector-based approaches to semantics can model rich lexical meanings, but they largely fail to capture sentiment information that is central to many word meanings and important for a wide range of NLP tasks. We present a model that uses a mix of unsupervised and supervised techniques to learn word vectors capturing semantic term–document information as well as rich sentiment content. The proposed model can leverage both continuous and multi-dimensional sentiment information as well as non-sentiment annotations. We instantiate the model to utilize the document-level sentiment polarity annotations present in many online documents (e.g. star ratings). We evaluate the model using two small, widely-used corpora and find it out-performs several previously introduced methods for sentiment classification. We also introduce a large dataset of movie reviews to serve as a more robust benchmark for work in this area. Using data from the Experience Project, we study texts as distributions over sentiment categories. Analysis of the document collection shows the texts contain blended sentiment information substantially different from a categorization view of sentiment. We extend our statistical vector-space model to learn distributions over emotive categories, in addition to capturing basic semantic information in an unsupervised fashion. Our model outperforms several baselines in predicting sentiment distributions given only the text of a document.

1.2 First Published Appearances of the Described Contributions

Most contributions described in this thesis have first appeared in various publications. Below are the publications roughly corresponding to each of the chapters:

- Chapter 2: [76], [77]
- Chapter 3: [80]
- Chapter 4: [79]
- Chapter 5: [82]
- Chapter 6: [83]
- Chapter 7: [78], [81].

Chapter 2

Speech Denoising with Neural Networks

Robust automatic speech recognition (ASR), robust to background noise and channel distortion, is a fundamental problem as ASR increasingly moves to mobile devices. Existing state-of-the-art methods for robust ASR use specialized domain knowledge to denoise the speech signal [34] or train a word-segment discriminative model robust to noise [113]. The performance of such systems is highly dependent upon the methods' designers imbuing domain knowledge like temporal derivative features or signal filters and transforms.

Neural network models have also been applied to learn a function to map a noisy utterance $\mathbf{x}$ to a clean version of the utterance $\mathbf{y}$ [138]. Such function approximators encode less domain-specific knowledge and by increasing hidden layer size can approximate increasingly nonlinear functions. Previous work also introduced neural network models which capture the temporal nature of speech using recurrent connections, and directly estimate word identity for isolated digit recognition [104]. However, previous work has focused on fairly small neural network models and experimented with simple noise conditions – often a single background noise environment.

Our novel approach applies large neural network models with several hidden layers and temporally recurrent connections. Such *deep* models should better capture the complex relationships between noisy and clean utterances in data with many noise environments and noise levels. In this way, our work shares underlying principles with deep neural net acoustic models, which yield substantial improvements in ASR [22, 52]. We experiment with a reasonably large set of background noise environments and demonstrate the importance of models with many hidden layers when learning a denoising function.

Training noise reduction models using stereo (clean and noisy) data has been used successfully with the SPLICE algorithm [28]. SPLICE attempts to model the joint distribution between clean and noisy data, $p(\mathbf{x}, \mathbf{y})$. Separate SPLICE models are trained for each noise condition and noise level. SPLICE is similar to a linear neural network with fixed non-linear basis functions. Our model attempts to learn the basis functions with multiple hidden layers as opposed to choosing nonlinearities which may be important *a priori*.

We demonstrate our model's ability to produce cleaned utterances for the Aurora2 robust ASR task. The method outperforms the SPLICE denoising algorithm, as well as the hand-engineered ETSI2 advanced front end (AFE) denoising system. We additionally perform control experiments to assess our model's ability to generalize to an unseen noise environments, and quantify the importance of deep and temporally recurrent neural network architecture elements.

2.1 Model

For robust ASR we would like a function $f(\mathbf{x}) \rightarrow \mathbf{y}$ which maps the noisy utterance $\mathbf{x}$ to a clean utterance $\mathbf{y}$ with background noise and distortion removed. For example we might hand engineer a function $f(\mathbf{x})$ using band pass filters to suppress the signal at frequencies we suspect contain background noise. However engineering such functions is difficult and often doesn't capture the rich complexity present in noisy utterances. Our approach instead learns the function $f(\mathbf{x})$ using a broad class of non-linear function approximators – neural networks. Such models adapt to model the nonlinear relationships between noisy and clean data present in given training data. Furthermore, we can construct various network architectures to model the temporal structure of speech, as well as enhance the nonlinear capacity of the model by building *deep* multilayer function approximators.

Given the noisy utterance $\mathbf{x}$, the neural network outputs a prediction $\hat{\mathbf{y}} = f(\mathbf{x})$ of the clean utterance $\mathbf{y}$. The error of the output is measured via squared error,

$$||\hat{\mathbf{y}} - \mathbf{y}||^2, \tag{2.1}$$

where $||\cdot||$ is the ℓ_2 norm between two vectors. The neural network learns parameters to minimize this error on a given training set. Such noise removal functions are often studied in the context of specific feature types, such as cepstra. Our model is instead agnostic as to the type of features present in $\mathbf{y}$ and can thus be applied to any sort of speech feature without modification.

2.1.1 Single Layer Denoising Autoencoder

A neural network which attempts to reconstruct a clean version of its own noisy input is known in the literature as a denoising autoencoder (DAE) [149]. A single hidden layer DAE outputs its prediction $\hat{y}$ using a linear reconstruction layer and single hidden layer of the form,

$$\hat{y} = Vh^{(1)}(x) + c \tag{2.2}$$

$$h^{(1)}(x) = \sigma(W^{(1)}x + b^{(1)}) \tag{2.3}$$

The weight matrices V and $W^{(1)}$ along with the bias vectors c and $b^{(1)}$ parameters of the model. The hidden layer representation $h^{(1)}(x)$ is a nonlinear function of the input vector x because $\sigma(\dot{)}$ is a point-wise nonlinearity. We use the logistic function $\sigma(z) = \frac{1}{1+e^z}$ in our work.

Because an utterance $\mathbf{x}$ is variable-length and training an autoencoder with high-dimensional input is expensive, it is typical to train a DAE using a small temporal context window. This increases computational efficiency and saves the model from needing to re-learn the same denoising function at each point in time. Furthermore, this technique allows the model to handle large variation in utterance durations without the need to zero pad inputs to some maximum length. Ultimately the entire clean utterance prediction $\hat{\mathbf{y}}$ is created by applying the DAE at each time sample of the input utterance – much in the same way as a convolutional filter.

2.1.2 Recurrent Denoising Autoencoder

The conventional DAE assumes that only short context regions are needed to reconstruct a clean signal, and thus considers small temporal windows of the utterance independently. Intuitively this seems a bad assumption since speech and noise signals are highly correlated at both long and short timescales. To address this issue we add temporally recurrent connections to the model, yielding a recurrent denoising autoencoder (RDAE). A recurrent network computes hidden activations using both the input features for the current time step x_t and the hidden representation from the previous time step $h^{(1)}(x_{t-1})$. The full equation for the hidden activation at time t is thus,

$$h^{(1)}(x_t) = \sigma(W^{(1)}x_t + b^{(1)} + Uh^{(1)}(x_{t-1})), \tag{2.4}$$

which builds upon the DAE (Equation 2.3) by adding a weight matrix U which connects hidden units for the current time step to hidden unit activations in the previous time step. The RDAE thus does not assume independence of each input window x but instead models temporal dependence which we expect to exist in noisy speech utterances.

2.1.3 Deep Architectures

A single layer RDAE is a nonlinear function, but perhaps not a sufficiently expressive model to capture the complexities of noise environments and channel distortions. We thus make the model more nonlinear and add free parameters by adding additional hidden layers. Indeed, much of the recent success of neural network acoustic models is driven by *deep* neural networks – those with more than one hidden layer. Our models naturally extend to using multiple hidden layers, yielding the deep denoising

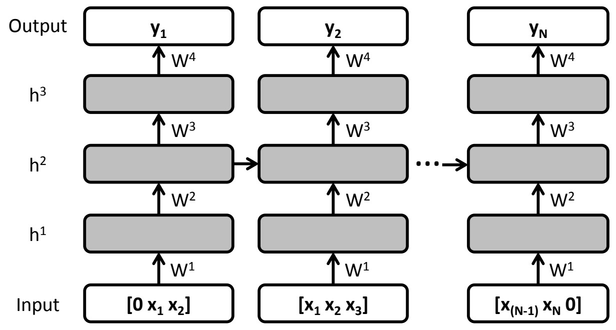


Figure 2.1: Deep Recurrent Denoising Autoencoder. A model with 3 hidden layers that takes 3 frames of noisy input features and predicts a clean version of the center frame

autoencoder (DDAE) and the deep recurrent denoising autoencoder (DRDAE). Figure 2.1 shows a DRDAE with 3 hidden layers. Note that recurrent connections are only used in the middle hidden layer in DRDAE architectures.

With multiple hidden layers we denote the i^{th} hidden layer's activation in response to input as $h^{(i)}(x_t)$. Deep hidden layers, those with $i > 1$ compute their activation as,

$$h^{(i)}(x_t) = \sigma(W^{(i)} h^{(i-1)}(x_t) + b^{(i)}). \tag{2.5}$$

Each hidden layer $h^{(i)}$ has a corresponding weight matrix $W^{(i)}$ and bias vector $b^{(i)}$. For recurrent models, the middle hidden layer has temporal connections as in Equation 2.4.

2.2 Experiments

We perform robust ASR experiments using the Aurora2 corpus [105]. Noisy utterances in Aurora2 were created synthetically, so it provides noisy and clean versions of the same utterance which is required to train our denoising models. However, the training set provides only four noise environments and we do not expect our model to learn a general denoising function given such a limited view of possible clean utterance corruptions. Our model attempts to predict clean MFCC features given MFCC features of the corrupted utterance.

2.2.1 Training

We train our model using the standard multi-condition training set which includes 8,001 utterances corrupted by 4 noise types at 5 different noise levels. Model gradients are computed via backpropagation – unrolling the model through time with no truncation or approximation for recurrent models. The L-BFGS optimization algorithm is used to train all models from random initialization. We find this batch optimization technique to perform as well as the pre-training and stochastic gradient fine-tuning approach used in other deep learning work [65]. We train a DRDAE with 3 hidden layers of 500 hidden units and use an input context of 3 frames to create x_t.

2.2.2 Robust ASR

We evaluate the model using the "clean" testing condition – where the standard HMM system is trained on only clean data and evaluated on noisy data. This condition evaluates whether the model can properly transform noisy features into their corresponding clean versions as expected by the HMM acoustic models.

Table 2.1 shows error rates for each noise condition averaged across the 4 noise types present in test set A. For comparison we include the error rates when using the original MFCC features, as well as the features produced by the ETSI2 advanced front end (AFE) denoising system [34]. Overall, our model outperforms the MFCC baseline as well as the AFE denoising system. At lower SNRs our model reconstructs clean MFCCs much better than the AFE leading to substantial performance improvements. When averaged across SNR conditions and ignoring the clean and -5dB settings as per the ETSI standard, our model gives a WER of 10.85%. This is better than the 12.26% of the AFE, and outperforms the SPLICE denoising algorithm which yields an 11.67% WER for this setting.

We also compare to a tandem approach, where a neural network is trained to

SNR	MFCC	AFE	Tandem	DRDAE
Clean	0.94	**0.77**	0.79	0.94
20dB	5.55	1.70	2.21	**1.53**
15dB	14.98	3.08	4.57	**2.25**
10dB	36.16	6.45	7.05	**4.05**
5dB	64.25	14.16	17.88	**10.68**
0dB	84.62	35.92	48.12	**32.90**
-5dB	90.04	**68.70**	79.54	70.16

Table 2.1: Word error rates (WER) on Aurora2 test set A. Performance is averaged across the four noise types in the test set. These noise environments are the same four present in the DRDAE training set.

output phoneme posteriors [150]. In the tandem setup, as in our own, network output is used as observations for the HMM Gaussian mixture models. We find our model to perform substantially better than a recurrent neural network used in a tandem setup. We trained the tandem network on exactly the same clean and noisy data used to train our denoising model – the Aurora2 clean and multi-condition training sets combined.

Table 2.1 shows the tandem result. The tandem network trained on clean and noisy data outperforms a tandem approach trained on clean data only as reported in [150]. However, our feature denoising outperforms the tandem approach, which suggests that for robust ASR it is better to predict clean features as opposed to phoneme posteriors.

2.2.3 ASR on Unseen Noise Environments

Test set A uses the same four noise environments as the training set used to train our DRDAE model. Because the model is discriminatively trained to remove only the four noise environments given, there is a danger it will generalize poorly to new noise environments. We thus evaluate the DRDAE model on test set B which contains four noise environments not seen during denoising training, Table 2.2 shows the result.

Our model significantly outperforms the MFCC baseline, suggesting on unseen

SNR	MFCC	AFE	DRDAE
Clean	0.94	**0.77**	0.94
20dB	7.94	**1.73**	2.24
15dB	20.51	**3.31**	3.87
10dB	43.81	**6.45**	8.61
5dB	70.04	**15.70**	21.39
0dB	86.29	**37.35**	51.25
-5dB	91.13	**69.99**	82.96

Table 2.2: Word error rates (WER) on Aurora2 test set B. Performance is averaged across the four noise environments in the test set. The test noise environments do not match the four noise environments used to train the DRDAE.

noise it still removes a substantial amount of noise from utterances. However, the DRDAE performs worse than it did on test set A, which is to be expected as it trained on only four noise environments. For test sets A and B, the AFE performs similarly because its processing pipeline encodes no special knowledge of the noise environments. This consistent performance across noise environments for the AFE leads it to outperform the DRDAE in this test condition. We note that the reported WER average of 12.25% for the SPLICE algorithm outperforms the DRDAE average of 17.47%. We hypothesize SPLICE generalizes better because separate models are trained for different noise environments and at test time the noise is matched to the most similar model. Our model makes fewer assumptions and is thus more dependent upon the training data to provide a reasonable sample of noise environments that could be encountered at test time.

2.2.4 Denoising Model Comparison

Because of the many possible neural network architectures for denoising, we wish to explore what aspects of the architecture are important for good performance. This also serves to help understand whether architecture choice impacts how well the model generalizes to unseen noise conditions. We thus train versions of the model which are shallow as opposed to deep, and non-recurrent. Because the WER performance metric additionally depends upon HMM training, we compare models with a mean-squared error (MSE) metric to directly measure how well they predict clean data $\mathbf{y}$ from noisy input $\mathbf{x}$.

We are interested in both how well the models fit the training data, and generalize to a type of noise unseen during training. For this, we train the models using three of the four noise types present in the multi-condition training set, and measure performance on the fourth noise type as a development set. Clean utterances of the Aurora2 test sets are not readily available so we do not evaluate MSE on the test sets.

We train single hidden layer recurrent (RDAE) and non-recurrent (DAE) denoising models with 1000 hidden units each. We also train the same 3 layer with 500 hidden units each DRDAE model as used in the ASR experiment. Additionally, we train a non-recurrent version of this model (DDAE). All models are trained with the same input window size and on the same training set. Table 2.3 shows the training and development set MSE results, as well as the MSE of the corrupted input for each noise condition. We see that both temporally recurrent connections and multiple hidden layers are essential in both fitting the training set well and generalizing to unseen noise types.

2.3 Recurrent Neural Network Feature Enhancement: The 2nd CHiME Challenge

Background noise and channel distortions introduced when performing automatic speech recognition (ASR) in home environments are hard to anticipate and highly complex. Hand-designing a procedure to reduce noise and distortion in such a wide variety of environments presents a huge challenge. Automatically learning such a function from data offers an attractive alternative, as a learning system can adapt to noises and distortions present in the training data. Further, a machine learning approach, in particular neural networks, can create complex non-linear functions which may be difficult for a human engineer to invent.

	Input	DAE	RDAE	DDAE	DRDAE
Clean T	0	61.20	8.70	34.57	**7.49**
20dB T	47.13	69.37	30.68	43.07	**30.16**
15dB T	52.47	70.49	33.05	45.28	**32.42**
10dB T	58.52	72.96	36.12	48.95	**35.34**
5dB T	64.91	76.65	40.11	53.98	**39.17**
Clean D	0	61.27	8.76	34.64	**7.55**
20dB D	50.20	65.61	**34.42**	47.83	34.73
15dB D	56.05	67.85	**37.40**	50.94	37.55
10dB D	60.77	69.54	39.92	53.75	**39.69**
5dB D	66.06	74.02	43.71	58.50	**43.26**

Table 2.3: Average mean squared error (MSE) of denoised input with respect to the true clean features. One noise type from the Aurora 2 multi-condition training set (T) was used as a development set (D) to assess how well models generalize to an unseen noise type. One and two layer models were trained with and without recurrent connections for comparison. The MSE of the noisy input serves as a reference for the error metric.

Previous work introduced deep recurrent autoencoder neural networks (DRDAEs) as an approach for acoustic feature enhancement in robust ASR [76]. Feature enhancement with DRDAEs resulted in error rates competitive with state-of-the-art noise reduction approaches on the Aurora2 dataset [105]. Furthermore, noise reduction appears to be a better use of a neural network architecture for robust ASR as compared with hybrid and tandem approaches [150], though more thorough comparisons are necessary. Our approach to the 2nd CHiME Challenge track 1 task [147] applies the same basic DRDAE approach. We train a recurrent neural network to predict clean acoustic features from the noisy inputs. As compared with Aurora2, CHiME offers more challenging environmental noise along with reverberation distortions. We developed several extensions to the basic DRDAE model which yielded significant improvements in preliminary experiments on Aurora2. This work presents model improvements along with results on the 2nd CHiME Challenge track 1 development and test sets.

2.3.1 Feature Enhancement Approach

We train a DRDAE neural network to predict clean acoustic features y from the noisy input features x. This work presents three improvements to the existing DRDAE approach. First, we apply cepstral mean and variance normalization (CMVN) independently to both the input and output features [30]. CMVN serves to normalize utterances which otherwise have large differences in feature norm across SNRs. We note that CMVN is similar to whitening procedures widely used to improve performance of neural network models on computer vision tasks [55]. Second, we input additional information to the DRDAE beyond the noisy acoustic features. In particular, we include an estimate of the background noise at each frame along with the input window of acoustic features. In principle, the DRDAE approach allows for a variety of side information or feature transforms at the input layer. There is no requirement of uncorrelated inputs, and during training the model can learn to combine inputs appropriately. Third, we extend the DRDAE architecture by adding a "short circuit" layer – a linear weight matrix mapping from the input features directly to the output. This direct output pathway improves performance in high SNR conditions where the correct feature transformation is close to the identity function.

Figure 2.2 shows the DRDAE architecture used. The network has two fully connected hidden layers, each with 512 hidden units using the hyperbolic tangent nonlinearity. The output layer is linear to predict continuous-valued clean acoustic features. The second hidden layer is temporally recurrent with a full weight matrix W_r. An input window of $+/-7$ MFCC features (without deltas or acceleration) serves as context when predicting the clean version of the center feature. Deltas and accelerations are computed on the predicted clean features before running the recognizer. Please see the original DRDAE denoising paper for a more thorough explanation [76]. The short circuit connections constitute an additional linear mapping W_s from the inputs directly to the output layer. At each time i, we include as part of the DRDAE input an estimate z_i of the background noise. We use the averaged first 10 frames of the utterance as a noise estimator, $z_i = \frac{1}{10}\sum_{j=1}^{10} x_j$, where x_j is a frame of MFCC features. This estimator is quite simple and assumes the background noise is stationary, but has been shown to often work well in practice.

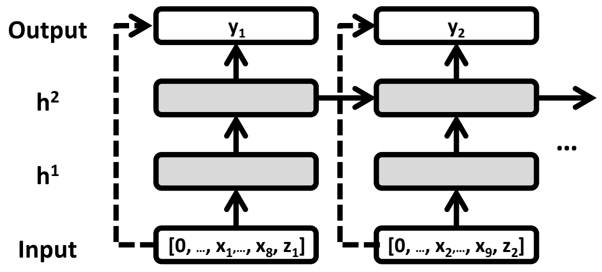


Figure 2.2: Deep Recurrent Denoising Autoencoder. The network has two hidden layers (gray) and a short circuit connection directly from the input to output layer (dotted line). Input consists of a window of MFCC features (x) along with an estimate of the background noise (z).

2.3.2 Experiments

We train a single DRDAE on all isolated training utterances, with the noisy utterance as input and clean as target. We use batch L-BFGS optimization, which has been shown to work well in practice for training neural networks [65]. We train the model until development set performance ceases to increase. Utterances were chunked into sequences of at most 100 frames for backpropagation through time for training. At test time, the network passes over the entire input sequence using full history. Table 2.4 shows the development set keyword accuracy for a DRDAE trained with 512 and 1024 hidden units in each layer. The unmodified recognizer is trained and evaluated on features output by the DRDAE.

In spite of the rich background noise used in the data, overfitting is a substantial problem for the DRDAE. Performance on the development set begins to decrease within about 500 iterations of the optimization algorithm, long before reaching a minimum on the training objective. Further, we found no development set performance improvement when using larger models with more hidden layers. Compared with our experiments on Aurora2, overfitting seems to be more of a problem on the CHiME task. We hypothesize the DRDAE model could be more effective with a larger training set. Regularization techniques such as weight tying or dropout could

SNR	MFCC Dev	1024 Dev	512 Dev	512 Test
-6dB	49.67	61.42	61.00	61.00
-3dB	57.92	65.42	65.92	65.67
0dB	67.83	72.67	71.58	73.42
3dB	73.67	77.33	78.25	78.67
6dB	80.75	83.42	82.92	83.33
9dB	82.67	84.50	86.17	85.58
Avg.	68.75	74.12	74.30	74.61

Table 2.4: Keyword recognition accuracy (%) on the development and test sets for MFCC baseline features and DRDAEs. We compare DRDAEs with 512 and 1024 units per hidden layer on the development set.

further improve the model.

The DRDAE model with 512 hidden units per layer produces a substantial improvement over the baseline recognizer. Previous work in robust ASR found that systems which combine both front end and recognizer modifications tend to perform best. Our approach instead focuses only on feature enhancement and uses the baseline recognizer. Further, the DRDAE approach allows flexibility to aggregate noise estimators and input features, without making assumptions about the signal. However, as with previous work in supervised training of denoising algorithms, the model assumes clean/noisy stereo data is available for training. Tasks with more training data can reduce overfitting and better leverage large capacity deep learning models.

Chapter 3

Word Recognition with Convolutional Neural Networks

Speech recognition remains a challenging goal of artificial intelligence with countless potential applications. Modern speech recognition systems are the result of thousands of man-hours, which is reflected by their enormous complexity. Indeed, training a state-of-the-art recognizer involves several re-estimation stages, speaker normalization, and signal processing. Each of these refinements is the result of months or years of work, and typically result in small absolute improvements to the final system error rate.

Recent work has shown substantial improvements to such systems are possible by using deep neural networks to model the acoustic signal [23, 91, 151]. Rather than use domain expertise to engineer modifications to the already complicated existing systems, the deep learning approach instead learns a highly expressive model from large amounts of training data. Such approaches now comprise state-of-the-art systems on many computer vision tasks while employing little or no domain-specific engineering [64, 155]. The speech domain too has rich opportunities to replace existing domain-engineered approaches with models in the deep learning regime. Replacing such complexity facilitates further research in speech without the need for overwhelming amounts of domain-specific knowledge.

In this work, we explore using neural networks to model the acoustics of entire words rather than short, fixed-length intervals. Since words are long, variable-length acoustic elements, we use convolutional networks to collapse the variable-length inputs to a fixed-size representation. Previous work by [68] uses an unsupervised convolutional model to learn features for phoneme classification. Chopra et al. [15] use a convolution and temporal pooling architecture, also for TIMIT phone classification. We build upon previous work on convolutional neural networks for speech, but directly addresses the most challenging problem in the speech domain – large vocabulary continuous speech recognition (LVCSR). We build a convolutional network capable of classifying tens of thousands of words using long duration acoustic segments.

Our convolutional network must output information about word identity, which for the broadcast news dataset we consider requires reasoning over 75,000 unique words. A standard softmax or logistic regression classifier scales poorly to this number of classes. Instead we use *vector regression* where each word in the vocabulary is assigned a low-dimensional real-valued vector, and the network predicts the word vector given its acoustic input. This idea is related to error correcting output codes for large multi-class problems [29].

Much recent work from the deep learning community and others has focused on learning word vectors sensitive to different notions of similarity [20, 89, 141]. Our model attempts to project an acoustic input directly into such word vector spaces. Mapping acoustics to semantics has potential applications not only in LVCSR, but also for dialogue systems such as voice search, and recognizing speaker characteristics like emotive state. Our model provides a framework for mapping acoustics to different choices of word vector space, and lays the foundation for joint models which simultaneously learn word vectors from text and acoustics.

We introduce our general convolutional network architecture which maps variable-length word acoustics to a fixed-dimensional representation, then discuss our novel vector regression output layer for speech recognition. With experiments on hundreds of hours of broadcast news we show our model can classify words in a 10,000-way decision task. Finally, we integrate our model's word predictions with the SCARF recognizers and demonstrate improvement over a baseline LVCSR system.

3.1 Model

We wish to interpret the acoustic signal v of an entire word w, and transform it into a vector $\phi^{(w)}$ representing the meaning of the word, or the word itself. This task corresponds to learning a function $f(v)$ such that $f(v) = \phi^{(w)}$. This function is highly complex, as acoustics depend on speaker factors such as gender and emotive state, as well as environmental factors such as background noise and microphone placement. Furthermore, since word utterances v are variable length with a substantial range in duration, the function f must accept a variable-length acoustic input.

A substantial amount of work in speech recognition focuses on engineering away distortions and speaker variation, corresponding to learning a function $g(v) = v'$. Rather than mapping v to word semantics or identity, $g(v)$ simply attempts to produce a 'clean' or canonical acoustic representation, v' [38]. Our work instead takes a deep learning approach to the problem – leveraging an abundance of data to train a model with large capacity. The goal of this approach is to fit a highly expressive model to approximate the true function $f(v)$ by leveraging an abundance of labeled examples of the form $(v, \phi^{(w)})$. This direct approach avoids the potentially challenging engineering effort of modeling away distortions in v and prescribing how the "true" v' should look.

Deep neural networks have recently demonstrated substantial success as acoustic models in HMM-based speech recognition systems [23]. These approaches reason over very short acoustic spans, typically on the order of 75 to 175 milliseconds (ms). Because the inputs are small, fixed-duration acoustic spans, the neural network directly models the entire acoustic span. Our acoustic input is instead an entire word, which are variable in length and last 312 ms on average but often last 500 ms or more.

With such large variations in temporal extent, a straightforward neural network approach can not be used. Instead, we use a *convolutional* neural network which can model long duration words with a tractable number of parameters. This general architecture efficiently collapses variable-length inputs into a fixed-size representation. As in previous work on convolutional models applied to images, we can introduce arbitrary numbers of convolution and pooling layers to achieve a highly expressive multi-layer architecture [66].

Using convolution and pooling, we obtain a fixed-dimensional vector representation z of an acoustic input span v. The convolutional network thus represents a function $z = h(v)$ which becomes increasingly nonlinear as we increase the number of filters in each layer, the number of layers, and other features of the network architecture. This is detailed in Section 3.1.1.

Ultimately the model must predict a word vector $\hat{\phi}$ which approximates the true word vector $\phi^{(w)}$ corresponding to the utterance. The model already has substantial nonlinearities in $h(v)$ so we use a simple, affine mapping from z to the predicted vector $\hat{\phi}$. This is discussed in Section 3.1.2.

By representing words as vectors we have a fixed size output layer to capture an arbitrarily large vocabulary, because each word is simply a point in high-dimensional space. This approach results in a network model size which is constant as the vocabulary increases. In contrast, a softmax classifier over all words results in a model which scales linearly with the vocabulary size. This scales poorly when we consider large vocabulary speech recognition tasks, and training classifier parameters for words which occur infrequently is difficult from an optimization perspective.

3.1.1 Convolution and Pooling

We are given an acoustic input signal $v \in \mathbb{R}^{n^{(0)} \times t^{(0)}}$, where $t^{(0)}$ is the word duration and each column of v is a $n^{(0)}$-dimensional feature computed at an instance in time. By a sequence of convolution and pooling steps, our model must yield a fixed-length output representation $h(v)$.

A filter response, $C^{(1)} \in \mathbb{R}^{n^{(1)} \times t_C^{(1)}}$ is computed by convolving $n^{(1)}$ learned filters with the input signal, and applying a nonlinear activation function to the output.

$$C_i^{(1)} = \sigma(W_i^{(1)} \star v + b_i^{(1)}) \tag{3.1}$$

where each $W_i^{(1)} \in \mathbb{R}^{n^{(0)} \times \omega_c^{(1)}}$ is a linear convolutional filter, $b_i^{(1)}$ is a scalar bias term, and $\sigma(\cdot)$ a nonlinear activation function. The number of filters $n^{(1)}$ and the width of each filter $\omega_C^{(1)}$ are both free parameters of the architecture. Each column of the resulting activation matrix $C^{(1)}$ thus represents $n^{(1)}$ filter responses at a particular

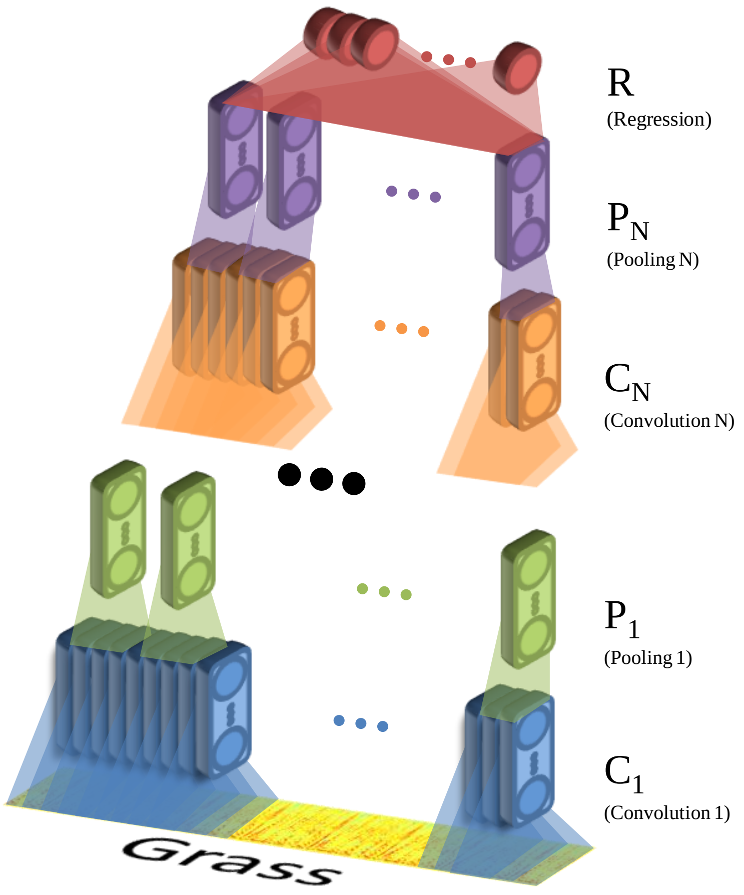


Figure 3.1: Convolutional network architecture.

time instant. The number of columns, $t_C^{(1)}$ is determined by a "valid" convolution ($\star$ operator) and the length of the original signal, $t^{(0)}$. Namely:

$$t_C^{(1)} = t^{(0)} - \omega_C^{(1)} + 1 \tag{3.2}$$

The literature explores several options for the nonlinear function given by $\sigma(\cdot)$, most commonly the logistic function. In our work however we use the *soft sign* function $\sigma(x) = \frac{x}{1+|x|}$ [41]. This function has been shown to be more amenable to backpropagation in deep architectures because it reduces gradient saturation problems.

Our architecture next performs a local pooling operation over time to add robustness to minor temporal shifts. This layer transforms the representation $C^{(1)}$ into a pooled representation $P^{(1)}$ by segmenting the response of each filter into non-overlapping temporal regions r_k of width $\omega_P^{(1)}$ and computing a statistic on each. A common choice of statistic is *mean pooling*:

$$P_{i,k}^{(1)} = \frac{1}{|r_k|} \sum_{j \in r_k} C_{i,j}^{(1)} \tag{3.3}$$

taking a local average over each region. Another common variant is *max pooling*:

$$P_{i,k}^{(1)} = \max_{j \in r_k} |C_{i,j}^{(1)}| \tag{3.4}$$

which has been shown to work well when applying convolutional networks to images.

If $\omega_P^{(1)}$ is fixed, the length of our new representation is given by:

$$t^{(1)} = \lceil t_C^{(1)} / \omega_P^{(1)} \rceil \tag{3.5}$$

The resulting representation $P^{(1)}$ is an $n^{(1)}$-dimensional representation of the audio at each instant in time, and there are an unknown number of time instances. Thus we can generalize equations 3.1…3.5 to create a second convolution response $C^{(2)}$ and pooling layer $P^{(2)}$, treating $P^{(1)}$ as an input signal. This convolution and pooling process can be extended to an arbitrary number of layers, forming a *deep* convolutional network and adding to its expressive power.

Thus far, the network described transforms a variable-length input into a variable-length feature representation, whose length scales linearly with that of its input. However, the function we wish to ultimately encode, $h(v)$, must yield a fixed-dimensional representation z which does not depend on word duration. To capture this, we treat $P^{(N)}$ as a special case. Unlike the lower layers, where the region sizes $\omega_P^{(1)}, \cdots, \omega_P^{(N-1)}$

	Scope[ℓ]	Description
$n^{(\ell)}$	$1...N$	# filters at $C^{(\ell)}$
$\omega_C^{(\ell)}$	$1...N$	Width of filters at $C^{(\ell)}$
$\omega_P^{(\ell)}$	$1...N-1$	Width of pool regions at $P^{(\ell)}$
$K_P^{(\ell)}$	N	# pool regions at $P^{(N)}$

Table 3.1: A list of free parameters in the $h(v)$ network

were fixed hyperparameters, we let $\omega_P^{(N)}$ scale with the size of its input.

$$\omega_P^{(N)} = \lceil t_C^{(N)} / K_P^{(N)} \rceil \tag{3.6}$$

where $K_P^{(N)}$, the number of top-level pooling regions, is a free parameter of our network. $P^{(N)}$, as given by equations 3.3 or 3.4, is of fixed size $n^{(N)} \times K_P^{(N)}$ regardless of the duration of the input signal. Flattening $P^{(N)}$ yields a vector z of dimensionality $n^{(N)} * K_P^{(N)}$, the number of hidden units in the final convolutional layer multiplied by the number of top-level pooling regions. These layers thus define $h(v)$. A summary of its parameters are given in Table 3.1.1.

3.1.2 Vector Regression

The hidden layers of the architecture provide the vector z, fixed-dimensional representation of the acoustic input v obtained by a parametrized nonlinear transformation.

We assume each word, w, in the vocabulary has an associated real-valued *word vector*, $\phi^{(w)}$. These vectors can represent acoustic, syntactic, and semantic information conveyed by words. There are numerous ways to learn such vectors depending on the application in which the vectors are to be used [141, 6]. For our purposes the vector representations can come from any sort of learning procedure. The important property of using word vectors is the ability to handle an arbitrarily large vocabulary with a fixed number of model parameters. The dimensionality d of the word vectors is a free parameter of the architecture. Reasonable choices for d depend on the size of the vocabulary and algorithm used to generate the word vectors. In practice vocabularies on the order of hundreds of thousands of words can be meaningfully modeled with d as small as 50 or 200. Thus regressing to word vectors vastly reduces the model complexity as compared to learning a parametric classifier for each word in the vocabulary.

We use a linear regression layer R to predict a word vector $\hat{\phi}$ from an acoustic span v,

$$\hat{\phi} = W^{(R)}h(v) + b^{(R)}. \tag{3.7}$$

The affine mapping defined by $W^{(R)}$ and $b^{(R)}$ project the high-dimensional vector $h(v)$ into the word vector space. We use a simple affine mapping here because we want the network to capture rich transformations in the convolution and pooling layers rather than while doing the regression.

When training the model, we are given a dataset $\mathcal{D}$ containing tuples of the form (v, w) corresponding to the utterance acoustics and word identity respectively. The associated word vector $\phi^{(w)}$ is then taken from a lookup table, as the word representations are fixed in advance of network training. The loss function for the model is given by,

$$\sum_{(v,w)\in\mathcal{D}} ||W^{(R)}h(v) + b^{(R)} - \phi^{(w)}||^2 + \lambda||\theta||^2. \tag{3.8}$$

This is a quadratic linear regression loss function with a weight norm penalty. The vector θ corresponds to all weight matrices in the convolution and regression layers, and λ is a scalar parameter controlling the strength of the weight penalty term. We optimize this loss function with respect to all network and regression parameters simultaneously.

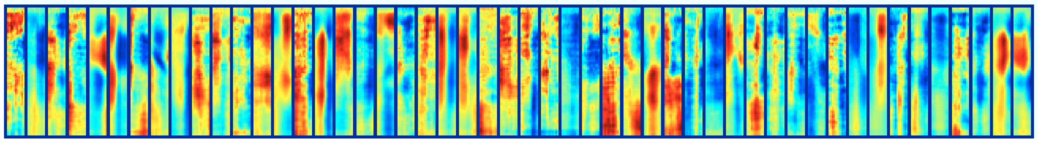


Figure 3.2: A random sample of 50 learned filters.

3.2 Experiments

Mapping word-level acoustic information to word semantics or identity is one of the fundamental challenges in large vocabulary speech recognition systems. Existing methods apply a standard HMM approach to most speech tasks. Well-tuned HMM systems are the state-of-the-art approach to recognizing speech given raw acoustic information. These systems however rely on a Markov assumption at the frame level, meaning their associated acoustic models consider only very short temporal spans of the signal. In our experiments, we work with a second-pass speech system, SCARF, which naturally handles integrating information from word-level acoustics. Furthermore, SCARF reasons over n-best lattices generate from a high performance HMM system. This results in improvements to SCARF directly improving large vocabulary speech recognition benchmarks. We first analyze our model's capability as a word classification system, and then perform speech recognition experiments on the RT-03 broadcast news task.

3.2.1 Dataset

We train our model on 300 hours of English broadcast news audio from the TDT4 corpus. There are many speakers with relatively clean speech and little background noise. The dataset includes sentence-level human transcriptions, which does not immediately satisfy our need for acoustic spans labeled at the word level. We obtain word timing information for the transcribed text from the IBM Attila system [132]. n-best lattices derived for this system were released by [169] for experiments in second-pass speech recognition. This results in over 2 million acoustic spans labeled by the word being said during each span.

To represent the acoustic signal we use log Mel-scale filterbank responses generated by HTK. Each frame of acoustic data is derived from 25 ms of audio, and the step size between successive frames is 10ms. We first apply PCA whitening to the input window to reduce its dimensionality. This substantially reduces the input dimension because successive frames of the acoustic signal are highly correlated.

The vectors ϕ used to represent each word form an embedding of the vocabulary in high-dimensional space. This space can fit different intuitions about word similarity, where similarity is encoded by distance between words' vector representations. Word vectors for speech are fundamentally different from word vectors for text alone because word similarity should depend on both the words' pronunciations and word meaning or syntax. For example, in a word vector space built purely from acoustics similar sounding words such as 'grass' and 'grease' could be easily confused because their vector distance is small. In contrast, a vector space which encodes word semantics while ignoring acoustics will have 'grass' and 'grease' much further away. Such a vector space is potentially more useful in recognizing word and phrase variations, such as when using a dialog system to search for songs or movies. However, the difficulty of regressing from acoustics alone to word vectors depends on acoustic sensitivity in the word vector space.

Because we are not aware of a method to learn word vectors which are sensitive to both word acoustics and semantics, we randomly generate unit-norm vectors to use as word representations for this work. In preliminary experiments we found randomly generated vectors perform better than those derived from a neural language model [140]. We hypothesize this is due to the importance of not confusing frequently occurring function words for speech recognition – such words are relatively close together in syntactically-focused vector spaces. We additionally performed initial experiments using word vectors derived from a pronunciation dictionary wherein words are represented by bag of phoneme vectors. This more acoustically-grounded representation performed only slightly better than random vectors and is thus not used in our final set of experiments for simplicity.

3.2.2 Network Training

There are several free parameters in the network architecture including the number of layers, window size of convolution filters, and pooling region sizes. We explore one setting for these parameters which limits the architecture size and depth for computational reasons. We train a network with 200 first layer filters, each filter looks at 11 consecutive input frames. Filter responses are passed through the soft sign nonlinearity, and then mean-pooled (see equation 3.3) using the final pooling layer. The final pooling layer computes an average over 4 temporal regions on the filter responses, resulting in a $4 * 200 = 800$ dimensional representation of the input. The linear regression layer projects this 800-dimensional representation to a 50-dimensional word vector.

Training a convolutional network on hundreds of hours of acoustic data poses a difficult optimization problem. We use mini-batch L-BFGS optimization, which as shown recent success as a technique for training deep neural network architectures [65]. We use the L-BFGS algorithm as implemented by the minFunc package [1].

3.2.3 Word Classification

To evaluate the trained network in isolation, we analyze its performance at the task of classifying words. For classification evaluation, we consider test examples from only the 10,000 most frequent words from the training set rather than the full vocabulary. Given input acoustics, our model predicts a word vector $\hat{\phi}$. The classifier output is then the nearest word vector ϕ_w from the set of 10,000 possible words. This nearest neighbor approach allows for such a large vocabulary as compared to a parametric classifier attempting to learn separate parameters for each of the 10,000 words.

Classification performance is evaluated using 10 hours of audio withheld from the training set, corresponding to 87,859 word instances. Word occurrence in speech, like text, follows a power law distribution. Because of this, a baseline classifier that always choose the most frequent word, or sample from the prior over word occurrence can perform surprisingly well in terms of accuracy. We use these approaches as

[1] http://www.di.ens.fr/ mschmidt/Software/minFunc.html

Classifier	100	10,000
Prior	3.8	0.8
Majority Class	13.1	6.2
Our Method	46.0	20.6

Table 3.2: Word classification accuracy (%). Classifiers are tested using both the 100 and 10,000 most frequent words from the training set as possible outputs. Our model predicts a vector and outputs a class prediction by choosing the nearest word's vector in Euclidean distance. We include baselines of randomly choosing classes from the empirical distribution over words (Prior), and always choosing the most frequently occurring word (Majority Class).

baselines when evaluating the accuracy of our model. Table 3.2 shows the accuracy result. As can be seen, the same model is able to achieve both 46.0% accuracy in a 100-way classification task and 20.6% accuracy in a 10,000-way classification task. This suggests that it is indeed possible to learn a transformation from an acoustic signal into a meaningful word-vector representation, where similarity in feature-space correlates with word similarity. As the significantly under-performing baselines show, this is not simply learning from the prior on word distributions: the output vector $\phi^{(w)}$ carries meaningful information about the input signal v. Even when the number of candidate words it must discriminate against is extremely high, this discriminative power is not lost.

3.3 Large Vocabulary Continuous Speech Recognition

Word classification is one example of potential applications for our word-level acoustic model. Perhaps the best known and most challenging application for such models is large vocabulary continuous speech recognition (LVCSR). We integrate our model with the SCARF second-pass recognition system to evaluate whether our acoustic model can improve upon an already high-quality speech system.

The standard approach to LVCSR uses hidden Markov models (HMMs) with

Gaussian mixture model acoustic models. This basic system has been refined over the past 25 years, resulting in extremely complicated training recipes to achieve state-of-the-art recognition performance [38]. Improvements to HMM systems include vocal tract length normalization [156], minimum phone error training [111], feature space maximum likelihood linear regression [39], and boosted maximum mutual information training [110]. Each of these improvements require a substantial amount of innovation and typically results in less than a 1% absolute improvement on word error rate (WER) of the final speech system. Furthermore, such improvements require a high level of domain expertise, resulting in a community of speech researchers somewhat isolated from the larger machine learning community. Our approach uses machine learning methods without specialized domain knowledge. Improving LVCSR systems using this approach to research has perhaps larger potential as we can more easily include new machine learning ideas to our methods than can be done by working directly with already complex HMM systems.

3.3.1 SCARF Recognizer

The SCARF speech recognition system of [170] offers the ability to achieve and improve upon state-of-the-art LVCSR results without the massive engineering effort involved in working with HMM systems directly. SCARF uses segmental conditional random fields to reason about recognition at the word level. It reasons over possible segmentations of the speech signal in time, and the word label of each segment. Specifically, SCARF computes the probability $P(\mathbf{w}|v)$ of a word sequence $\mathbf{w}$ given an acoustic signal v as,

$$\frac{\sum_{\mathbf{q} \in Seg(\mathbf{w},s)} \exp(\sum_{q \in \mathbf{q},k} \alpha_k f_k(w(q), v(q)))}{\sum_{\mathbf{w}'} \sum_{\mathbf{q} \in Seg(\mathbf{w}',s)} \exp(\sum_{q \in \mathbf{q},k} \alpha_k f_k(w(q), v(q)))} \tag{3.9}$$

For a particular utterance v and transcription hypothesis $\mathbf{w}$, SCARF reasons over all possible $|\mathbf{w}|$-segmentations of the acoustic input. Each segment q in a segmentation $\mathbf{q}$ is a portion of the acoustic signal assigned to a particular word $w \in \mathbf{w}$. Features in SCARF $f_k(w(q), v(q)))$ are a function of both the word $w(q)$ and acoustics $v(q)$ of

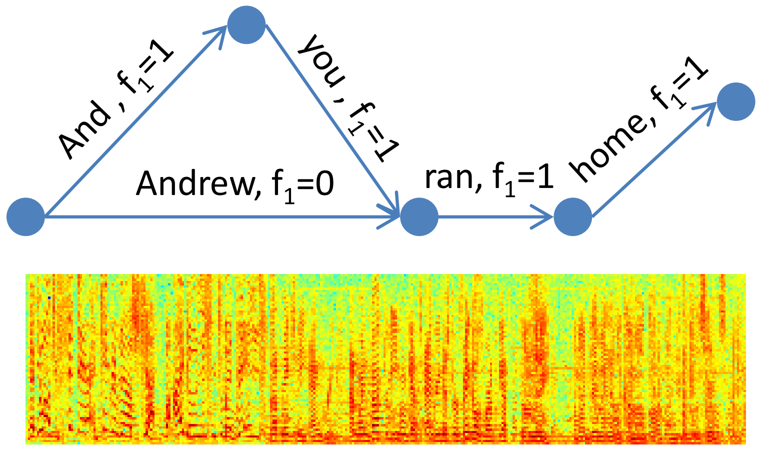


Figure 3.3: Example word lattice for an utterance. The lattice compactly represents possible segmentations of the utterance along with their transcriptions. We predict a word vector $\hat{\phi}$ for each segment (edge) and compute the distance from to the actual word vector $\phi^{(w)}$ for the segment. A binary feature f_1 is then produced to indicate which edge in an overlapping time span has minimum distance.

a particular segment q. For example, a unigram language model feature assigns its feature value based solely on the word hypothesis for the segment $w(q)$ and ignores the acoustics.

Reasoning over all possible utterance transcriptions and segmentations is not tractable, so SCARF relies on a *lattice* to restrict the search space. A lattice is defines a directed graph over time where vertices correspond to start/end times of words and edges to a particular word, figure 3.3 shows an example. The lattice compactly represents possible transcriptions for the entire time span of the utterance. We use lattices for the 300 hour broadcast news corpus described in section 3.2.1. The lattices were generated by the IBM Attila HMM recognizer [132] and released publicly by Zweig et al. [169]. These lattices correspond to an n-best decoding from the system, but in our experiment we do not provide SCARF with information about what the original recognizer chose as the most likely answer.

SCARF is trained on the lattices using a single feature – a unigram language model. We chose the simple language model and absence of other features to focus solely on the contribution of our novel word-level acoustic model to system performance. Word error rate (WER) is then evaluated on the RT-03 dataset using the Sclite tool. The resulting WER is 20.1% which is, as anticipated, high relative to the 15.6% performance achieved when using the 1-best decoding from Attila as a feature for SCARF.

3.3.2 Minimum Distance Feature

Our model predicts a word vector $\hat{\phi}^q$ for an acoustic segment q using only the acoustic information $v(q)$. When using SCARF, we also have a word label $w(q)$. We then compute the Euclidean distance $d = ||\hat{\phi}^q - \phi^{w(q)}||$ between the predicted word vector and the word vector corresponding to the word segment label.

After computing distances for all segments in the lattice, we convert them to a binary feature. At a particular point in time there is some *confusion set* C of possible segments that overlap at that time point. In figure 3.3 the segments ‘And’ and ‘Andrew’ form a confusion set, and the segments ‘you’ and ‘Andrew’ form a separate confusion set. Our final feature function is given by,

$$f_1(w(q), v(q)) = \begin{cases} 1 & \min_{q \in C} ||\hat{\phi}^q - \phi^{w(q)}|| \\ 0 & \text{otherwise} \end{cases} . \tag{3.10}$$

So any segment that is the minimum distance prediction for a confusion set in the lattice will have feature f_1 active. Figure 3.3 shows a valid setting of f_1 based on the confusion sets present in the lattice.

3.3.3 Results

We train SCARF using our minimum distance feature, as well as the same unigram language model feature described in section 3.3.1. Table 3.3 shows the resulting WER for our model and the baseline, as well as the performance possible in SCARF when including the 1-best hypothesis from the Attila system as a feature.

Model	Word Error Rate (%)
SCARF baseline	20.1
SCARF w/ Min Dist	19.8
Attila HMM	15.6

Table 3.3: Speech Recognition Performance. Adding our minimum distance feature improves upon the baseline system, but does not outperform using output of the highly engineered Attila recognizer is as a feature.

Our model achieves a 1.5% relative improvement over the baseline system – a reasonable gain for this challenging LVCSR task. Such a relative improvement is on the order of what is attained by adding yet another training step or feature modification to HMM systems like Attila, but our approach does not rely on domain-specific knowledge. This demonstrates that using convolutional networks to reason over entire words can benefit complete speech recognition systems.

3.4 Conclusion

We introduced a convolutional network architecture that projects the acoustics of an entire word to a word vector space. This general framework enables the model to handle large vocabularies via regression in place of parametric classification. We demonstrated the architecture's ability to accurately classify words in a 10,000-way problem. Further, when integrated into a large vocabulary speech recognizer, our model provides improvement over a baseline system. This demonstrates the potential for word-level deep learning approaches in the speech domain. Finally, the convolutional vector regression framework has many architecture parameters, and there are several types of word vector space we can choose. This offers many opportunities for future work in learning deep architectures to project word utterances into word vector spaces. Such models have application not only for speech recognition, but also dialog systems like voice search, and exploring vector spaces to capture both acoustic and semantic similarity.

Chapter 4

Rectified Linear Units for HMM-DNN Acoustic Modeling

Deep neural networks are quickly becoming a fundamental component of high performance speech recognition systems. Deep neural network (DNN) acoustic models perform substantially better than the Gaussian mixture models (GMMs) typically used in large vocabulary continuous speech recognition (LVCSR). DNN acoustic models were initially thought to perform well because of unsupervised pretraining [23]. However, DNNs with random initialization and sufficient amounts of labeled training data perform equivalently. LVCSR systems with DNN acoustic models have now expanded to use a variety of loss functions during DNN training, and claim state-of-the-art results on many challenging tasks in speech recognition [52, 62, 145].

DNN acoustic models for speech use several sigmoidal hidden layers along with a variety of initialization, regularization, and optimization strategies. There is increasing evidence from non-speech deep learning research that sigmoidal nonlinearities may not be optimal for DNNs. [40] found that DNNs with rectifier nonlinearities in place of traditional sigmoids perform much better on image recognition and text classification tasks. Indeed, the advantage of rectifier networks was most obvious in tasks with an abundance of supervised training data, which is certainly the case for DNN acoustic model training in LVCSR. DNNs with rectifier nonlinearities played an important role in a top-performing system for the ImageNet large scale image classification benchmark [63]. Further, the nonlinearity used in purely unsupervised feature learning neural networks plays an important role in final system performance [19].

Recently, DNNs with rectifier nonlinearities were shown to perform well as acoustic models for speech recognition. [168] train rectifier networks with up to 12 hidden layers on a proprietary voice search dataset containing hundreds of hours of training data. After supervised training, rectifier DNNs perform substantially better than their sigmoidal counterparts. [21] apply DNNs with rectifier nonlinearities and dropout regularization to a broadcast news LVCSR task with 50 hours of training data. Rectifier DNNs with dropout outperform sigmoidal networks without dropout.

In this work, we evaluate rectifier DNNs as acoustic models for a 300-hour Switchboard conversational LVCSR task. We focus on simple optimization techniques with no pretraining or regularization in order to directly assess the impact of nonlinearity choice on final system performance. We evaluate multiple rectifier variants as there are potential trade-offs in hidden representation quality and ease of optimization when using rectifier nonlinearites. Further, we quantitatively compare the hidden representations of rectifier and sigmoidal networks. This analysis offers insight as to why rectifier nonlinearities perform well. Relative to previous work on rectifier DNNs for speech, this paper offers 1) a first evaluation of rectifier DNNs for a widely available LVCSR task with hundreds of hours of training data, 2) a comparison of rectifier variants, and 3) a quantitative analysis of how different DNNs encode information to further understand why rectifier DNNs perform well. Section 4.1 discusses motivations for rectifier nonlinearities in DNNs. Section 4.2 presents a comparison of several DNN acoustic models on the Switchbaord LVCSR task along with analysis of hidden layer coding properties.

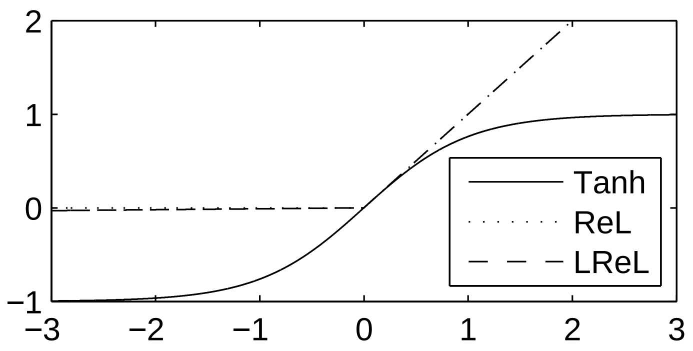


Figure 4.1: Nonlinearity functions used in neural network hidden layers. The hyperbolic tangent (tanh) function is a typical choice while some recent work has shown improved performance with rectified linear (ReL) functions. The leaky rectified linear function (LReL) has a non-zero gradient over its entire domain, unlike the standard ReL function.

4.1 Rectifier Nonlinearities

Neural networks typically employ a sigmoidal nonlinearity function. Recently, however, there is increasing evidence that other types of nonlinearites can improve the performance of DNNs. Figure 4.1 shows a typical sigmoidal activation function, the hyperboloic tangent (tanh). This function serves as the point-wise nonlinearity applied to all hidden units of a DNN. A single hidden unit's activation $h^{(i)}$ is given by,

$$h^{(i)} = \sigma(w^{(i)T} x), \tag{4.1}$$

where $\sigma(\cdot)$ is the tanh function, $w^{(i)}$ is the weight vector for the ith hidden unit, and x is the input. The input is speech features in the first hidden layer, and hidden activations from the previous layer in deeper layers of the DNN.

This activation function is anti-symmetric about 0 and has a more gradual gradient than a logistic sigmoid. As a result, it often leads to more robust optimization during DNN training. However, sigmoidal DNNs can suffer from the *vanishing gradient* problem [7]. Vanishing gradients occur when lower layers of a DNN have gradients of nearly 0 because higher layer units are nearly saturated at -1 or 1, the asymptotes of the tanh function. Such vanishing gradients cause slow optimization convergence, and in some cases the final trained network converges to a poor local minimum. Hidden unit weights in the network must therefore be carefully initialized as to prevent significant saturation during the early stages of training.

The resulting DNN does not produce a sparse representation in the sense of hard zero sparsity when using tanh hidden units. Many hidden units activate near the -1 asymptote for a large fraction of input patterns, indicating they are "off." However, this behavior is potentially less powerful when used with a classifier than a representation where an exact 0 indicates the unit is "off."

The rectified linear (ReL) nonlinearity offers an alternative to sigmoidal nonlinearites which addresses the problems mentioned thus far. Figure 4.1 shows the ReL activation function. The ReL function is mathematically given by,

$$h^{(i)} = \max(w^{(i)T}x, 0) = \begin{cases} w^{(i)T}x & w^{(i)T}x > 0 \\ 0 & \text{else} \end{cases}. \tag{4.2}$$

When a ReL unit is activated above 0, its partial derivative is 1. Thus vanishing gradients do not exist along paths of active hidden units in an arbitrarily deep network. Additionally, ReL units saturate at exactly 0, which is potentially helpful when using hidden activations as input features for a classifier.

However, ReL units are at a potential disadvantage during optimization because the gradient is 0 whenever the unit is not active. This could lead to cases where a unit never activates as a gradient-based optimization algorithm will not adjust the weights of a unit that never activates initially. Further, like the vanishing gradients problem, we might expect learning to be slow when training ReL networks with constant 0 gradients.

To alleviate potential problems caused by the hard 0 activation of ReL units, we additionally evaluate *leaky* rectified linear (LReL) hidden units. The leaky rectifier allows for a small, non-zero gradient when the unit is saturated and not active,

$$h^{(i)} = \max(w^{(i)T}x, 0) = \begin{cases} w^{(i)T}x & w^{(i)T}x > 0 \\ 0.01w^{(i)T}x & \text{else} \end{cases}. \tag{4.3}$$

Figure 4.1 shows the LReL function, which is nearly identical to the standard ReL function. The LReL sacrifices hard-zero sparsity for a gradient which is potentially more robust during optimization. We experiment on both types of rectifier, as well as the sigmoidal tanh nonlinearity.

4.2 Experiments

We perform LVCSR experiments on the 300 hour Switchboard conversational telephone speech corpus (LDC97S62). The baseline GMM system and forced alignments for DNN training are created using the Kaldi open-source toolkit [109]. We use a system with 3,034 senones and train DNNs to estimate senone likelihoods in a hybrid HMM speech recognition system. The input features for DNNs are MFCCs with a context window of +/- 3 frames. Per-speaker CMVN is applied as well as fMLLR. The features are dimension reduced with LDA to a final vector of 300 dimensions and globally normalized to have 0 mean and unit variance. Overall, the HMM/GMM system training largely follows an existing Kaldi recipe and we defer to that original work for details [145]. For recognition evaluation, we use both the Switchboard and CallHome subsets of the HUB5 2000 data (LDC2002S09).

We are most interested in the effect of nonlinearity choice on DNN performance. For this reason, we use simple initialization and training procedures for DNN optimization. We randomly initialize all hidden layer weights with a mean 0 uniform distribution. The scaling of the uniform interval is set based on layer size to prevent sigmoidal saturation in the initial network [40]. The output layer is a standard softmax classifier, and cross entropy with no regularization serves as the loss function. We note that training and development set cross entropies are closely matched throughout training, suggesting that regularization is not necessary for the task. Networks

Table 4.1: Results for DNN systems in terms of frame-wise error metrics on the development set as well as word error rates (%) on the Hub5 2000 evaluation sets. The Hub5 set (EV) contains the Switcboard (SWBD) and CallHome (CH) evaluation subsets. Frame-wise error metrics were evaluated on 25,000 frames held out from the training set.

Model	Dev CrossEnt	Dev Acc(%)	SWBD WER	CH WER	EV WER
GMM Baseline	N/A	N/A	25.1	40.6	32.6
2 Layer Tanh	2.09	48.0	21.0	34.3	27.7
2 Layer ReLU	1.91	51.7	19.1	32.3	25.7
2 Layer LReLU	1.90	51.8	19.1	32.1	25.6
3 Layer Tanh	2.02	49.8	20.0	32.7	26.4
3 Layer ReLU	1.83	53.3	18.1	30.6	24.4
3 Layer LReLU	1.83	53.4	17.8	30.7	24.3
4 Layer Tanh	1.98	49.8	19.5	32.3	25.9
4 Layer ReLU	1.79	53.9	17.3	29.9	23.6
4 Layer LReLU	1.78	53.9	17.3	29.9	23.7

are optimized using stochastic gradient descent (SGD) with momentum and a mini-batch size of 256 examples. The momentum term is initially given a weight of 0.5, and increases to 0.9 after 40,000 SGD iterations. We use a constant step size of 0.01. For each model we initially searched over several values for the step size parameter, $[0.1, 0.05, 0.01, 0.005, 0.001]$. For each nonlinearity type the value 0.01 led to fastest convergence without diverging from taking overly large steps. Network training stops after two complete passes through the 300 hour training set. Hidden layers contain 2,048 hidden units, and we explore models with varying numbers of hidden layers.

4.2.1 Impact of Nonlinearity

Our first experiment compares sigmoidal nonlinearity DNNs against DNNs trained using the two rectifier functions discussed in section 4.1. DNNs with 2, 3, and 4 hidden layers are trained for all nonlinearity types. We reserved 25,000 examples from the training set to obtain a held-out estimate of the frame-wise cross entropy and accuracy of the neural network acoustic models. Such a measurement is important because recognizer word error rate (WER) is only loosely correlated with the cross entropy metric used in our DNN acoustic model training. Table 4.1 shows the results for both frame-wise metrics and WER.

DNNs with rectifier nonlinearities substantially outperform sigmoidal DNNs in all error metrics, and across all DNN depths. Rectifier DNNs produce WER reductions of up to 2% absolute on the full Eval2000 dataset as compared with sigmoidal DNNs – a substantial improvement for this task. Furthermore, deeper 4 layer sigmoidal DNNs perform slightly worse than 2 layer rectifier DNNs despite having 1.76 times more free parameters. The performance gains observed in our sigmoidal DNNs relative to the GMM baseline system are on par with other recent work with DNN acoustic models on the Switchboard task [166]. We note that in preliminary experiments we found tanh units to perform slightly better than logistic sigmoids, another sigmoidal nonlinearity commonly used in DNNs.

The choice of rectifier function used in the DNN appears unimportant for both frame-wise and WER metrics. Both the leaky and standard ReL networks perform similarly, suggesting the leaky rectifiers' non-zero gradient does not substantially impact training optimization. During training we observed leaky rectifier DNNs converge slightly faster, which is perhaps due to the difference in gradient among the two rectifiers.

In addition to performing better overall, rectifier DNNs benefit more from depth as compared with sigmoidal DNNs. Each time we add a hidden layer, rectifier DNNs show a greater absolute reduction in WER than sigmoidal DNNs. We believe this effect results from the lack of vanishing gradients in rectifier networks. The largest models we train still underfit the training set.

4.2.2 Analyzing Coding Properties

Previous work in DNNs for speech and with ReL networks suggest that sparsity of hidden layer representations plays an important role for both classifier performance and invariance to input perturbations. Although sparsity and invariance are not necessarily coupled, we seek to better understand how ReL and tanh networks differ. Further, one might hypothesize that ReL networks exhibit "mostly linear" behavior where units saturate at 0 rarely. We analyze the hidden representations of our trained DNN acoustic models in an attempt to explain the performance gains observed when using ReL nonlinearities.

We compute the last hidden layer representations of 4-layer DNNs trained with each nonlinearity type from section 4.2.1 for 10,000 input samples from the held-out set. For each hidden unit, we compute its empirical *activation probability* – the fraction of examples for which the unit is not saturated. We consider ReL and LReL units saturated when the activation is nonpositive, $h(x) \leqslant 0$. Sigmoidal tanh units have negative and positive saturation, measured by an activation $h(x) \leqslant -0.95$ and $h(x) \geqslant 0.95$ respectively. For the sigmoidal units we also measure the fraction of units that saturate on the negative asymptote ($h(x) \leqslant -0.95$), as this corresponds to the "off" position. Figure 4.2 shows the activation probabilities for hidden units in the last hidden layer for each network type. The units are sorted in decreasing order of activation probability.

ReL DNNs contain substantially more sparse representations than sigmoidal DNNs. We measure *lifetime sparsity*, the average empirical activation probability of all units in the layer for a large sample of inputs [161]. The average activation probability for the ReL hidden layer is 0.11, more than a factor of 6 less than the average probability for tanh units (considering tanh to be active or "on" when $h(x) > -0.95$). If we believe sparse activation of a hidden unit is important for invariance to input stimuli, then rectifier networks have a clear advantage. Notice that in terms of sparsity the two types of rectifier evaluated are nearly identical.

Sparsity, however, is not a complete picture of code quality. In a sparse code, only a few coding units represent any particular stimulus on average. However, it is possible to use the same coding units for each stimulus and still obtain a sparse code. For example, a layer with four coding units and hidden unit activation probabilities $[1, 0, 0, 0]$ has average lifetime sparsity 0.25. Such a code is sparse on average, but not *disperse*. Dispersion measures whether the set of active units is different for each stimulus [162]. A different four unit code with activation probabilities

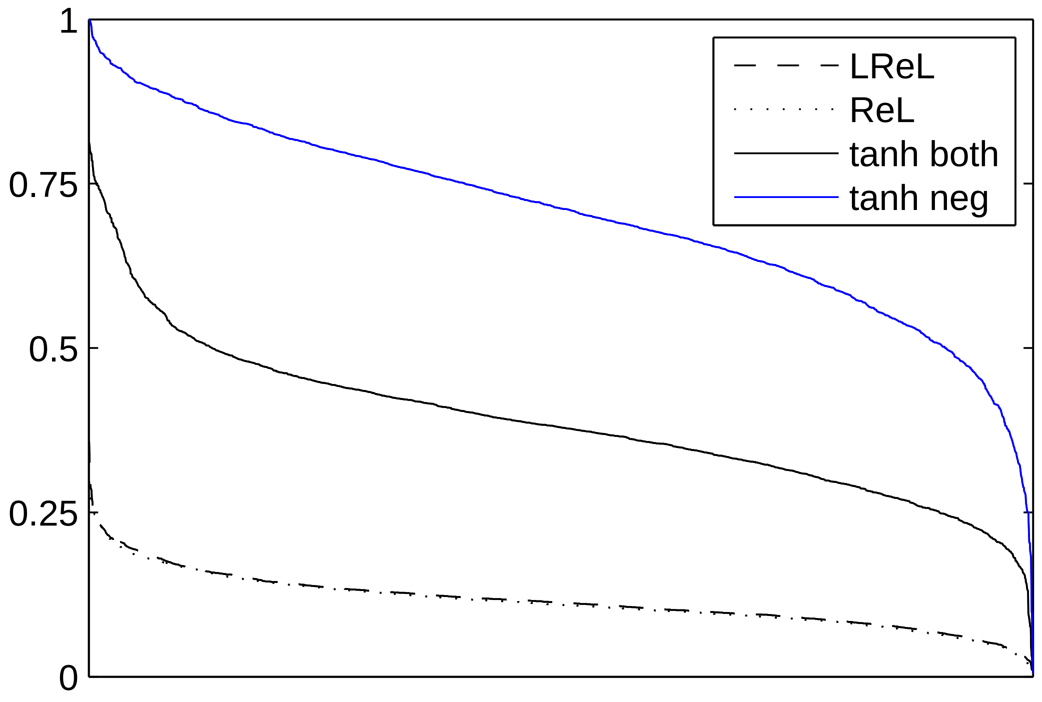


Figure 4.2: Empirical activation probability of hidden units in the final hidden layer layer of 4 hidden layer DNNs. Hidden units (x axis) are sorted by their probability of activation. In ReL networks, we consider any positive value as active ($h(x) > 0$). In tanh networks we consider activation in terms of not saturating in the "off" position ($h(x) > -0.95$, "tanh neg") as well as not saturating on either asymptote ($-0.95 < h(x) < 0.95$, "tanh both").

$[0.25, 0.25, 0.25, 0.25]$ again has lifetime sparsity 0.25 but is more disperse because units share input coding equally. We can informally compare dispersion by comparing the slope of curves in figure 4.2. A flat curve corresponds to a perfectly disperse code in this case.

We measure dispersion quantitatively for the hidden layers presented in figure 4.2. We compute the standard deviation of empirical activation probabilities across all units in the hidden layer [1]. A perfectly disperse code, where all units code equally,

[1]This metric for dispersion differs from metrics in previous work. Previous work focuses on analyzing linear filters with Gaussian-distribted inputs. Our metric captures the idea of dispersion more suitably for non-linear coding units and non-Gaussian inputs.

has standard deviation 0. Both types of ReL layer have standard deviation 0.04, significantly lower than the tanh layer's standard deviation of 0.14. This indicates that ReL networks, as compared with tanh networks, produce sparse codes where information is distributed more uniformly across hidden units. There are several results from information theory, learning theory, and computational neuroscience which suggest sparse-disperse codes are important, and translate to improved performance or invariance.

4.3 Conclusion

This work focuses on the impact of nonlinearity choice in DNN acoustic models without sophisticated pretraining or regularization techniques. DNNs with rectifiers produce substantial gains on the 300-hour Switchboard task compared to sigmoidal DNNs. Leaky rectifiers, with non-zero gradient over the entire domain, perform nearly identically to standard rectifier DNNs. This suggests gradient-based optimization for model training is not adversely affected by the use of rectifier nonlinearities. Further, ReL DNNs without pretraining or advanced optimization strategies perform on par with established benchmarks for the Switchboard task. Our analysis of hidden layer representations revealed substantial differences in both sparsity and dispersion when using ReL units compared with tanh units. The increased sparsity and dispersion of ReL hidden layers may help to explain their improved performance in supervised acoustic model training.

Chapter 5

Empirical Investigation of DNN Acoustic Models for LVCSR

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 [23]. However, DNNs with random initialization yield state-of-the-art LVCSR results for several speech recognition benchmarks [52, 62, 146]. Instead, it appears that modern DNN-based systems are quite similar to long-standing neural network acoustic modeling approaches [9, 50, 115]. 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 over-fit 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 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 5.1 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 5.2. Section 5.3 describes the neural network architectures and optimization algorithms evaluated in this paper. Section 5.4 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 5.6 which explore the performance of larger and deeper DNN architectures. We compare and quantify DNN representational properties in Section 5.8, and conclude in Section 5.9.

5.1 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 [37, 165, 127, 42, 58].

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 $p(y|x)$,

$$p(x|y) = \frac{p(y|x)p(x)}{p(y)}. \tag{5.1}$$

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,

$$\frac{p(y|x)}{p(y)}. \tag{5.2}$$

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 [9, 115, 87]. 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 [51, 148].

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 [90]. 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 [22, 56]. Within two years DNN acoustic models showed gains on challenging tasks within the LVCSR systems of Microsoft, Google, and IBM [52].

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 modern hybrid HMM systems. Researchers quickly established that hybrid HMMs work much better when using context-dependent triphones in place of monophones [23]. 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 [167]. Using DNNs with many hidden layers and many total parameters has generally found to be beneficial [166], 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 [22], 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 [42]. 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 [129]. 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 [130]. 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 [21, 168, 75].

   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 [129, 166, 92]. 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 [1, 121, 124]. 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 [153] and recurrent network approach of [118]. Modern recurrent network approaches to acoustic modeling have shown some initial success on large vocabulary tasks [126], and tasks where limited training data is available [125, 47, 150, 157]. 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 [27].

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 [5, 110, 143, 59], but were recently applied to DNN acoustic model training [146, 62, 134]. 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 [52] 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 [21].

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 [85, 95] as well as DNN acoustic modeling [62]. 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 [31] 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 [136].

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) [96, 74, 114], clusters of dozens to hundreds of computers [25, 14, 16], or clusters of GPUs [18]. 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 [33, 72]. 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.

5.2 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 [124]. 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 5.3 we describe the DNN, DCNN, and DLUNN architecture computations used in this work. Section 5.4 addresses questions of model size and overfitting on the Switchboard corpus while Section 5.5 uses the same baseline Switchboard system to compare DCNN and DLUNN architectures to DNNs and baseline GMMs. Section 5.6 presents experiments using the larger training corpus to explore issues of model size, DNN depth, and optimization algorithm. Sections 5.7 and 5.8 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.

5.3 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.

5.3.1 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,

$$-\sum_{k=1}^{K} 1\{y = k\} \log \hat{y}_k, \tag{5.3}$$

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.

5.3.2 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

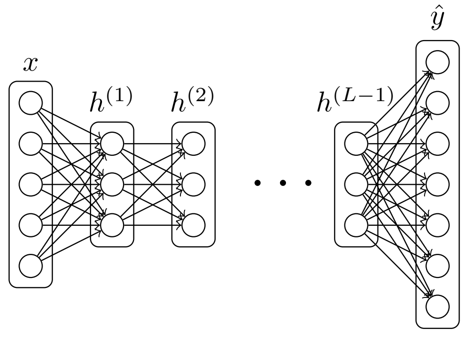


Figure 5.1: A DNN with 5-dimensional input, 3-dimensional hidden layers, and 7-dimensional output. Each hidden layer is fully connected to the to the previous and subsequent layer.

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 5.1 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,

$$h^{(1)}(x) = \sigma(W^{(1)T}x + b^{(1)}), \tag{5.4}$$

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 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 $h^{(i-1)}$,

$$h^{(i)}(x) = \sigma(W^{(i)T}h^{(i-1)} + b^{(i)}). \tag{5.5}$$

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[21, 168, 75]. The rectifier nonlinearity is defined as,

$$\sigma(z) = \max(z, 0) = \begin{cases} z_i & z_i > 0 \\ 0 & z_i \leqslant 0 \end{cases}. \tag{5.6}$$

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,

$$\hat{y}_j = \frac{\exp(W_j^{(L)T} h^{(L-1)} + b_j^{(L)})}{\sum_{k=1}^{N} \exp(W_k^{(L)T} h^{(L-1)} + b_k^{(L)})}. \tag{5.7}$$

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 5.3, 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.

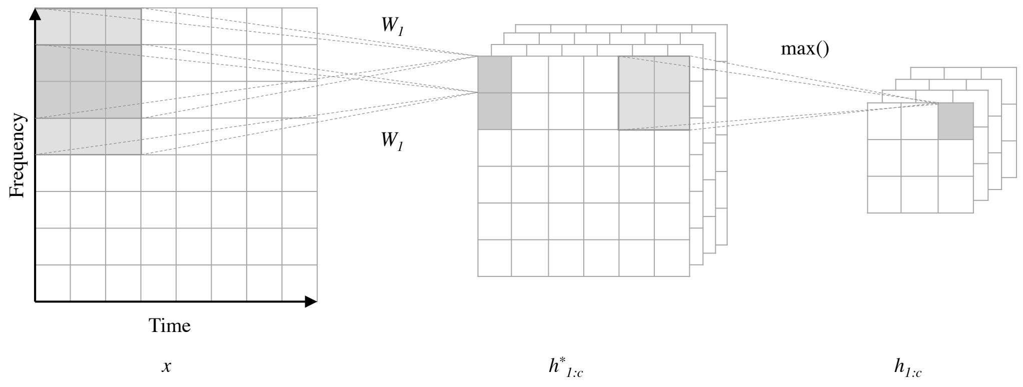


Figure 5.2: Convolution and pooling first layer architecture. Here the filter size is 5×5, and the pooling dimension is 3×3. Pooling regions are non-overlapping. Note that the 5×5 filters applied to each position in the convolution step are constrained to be the same. For max-pooling, the maximum value in each 3×3 grid is extracted.

5.3.3 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 [66, 63]. 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 [153]. 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 [124], 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 5.2 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 5.6. 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 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 [67, 1, 124]. 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.

5.3.4 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 5.3 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

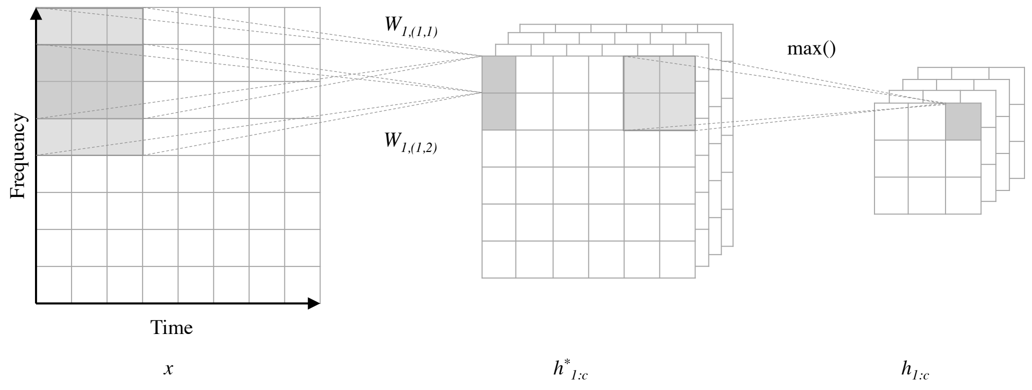


Figure 5.3: Locally connected untied first layer architecture. Here the filter size is 5×5, and the pooling dimension is 3×3. Pooling regions are non-overlapping. Unlike the convolutional layer shown in Figure 5.2, the network learns a unique 5×5 set of weights at each location. The max pooling layer otherwise behaves identically to the pooling layer in a convolutional architecture.

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.

5.3.5 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 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) [106, 119]. 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,

$$v_t = \mu v_{t-1} - \epsilon \nabla f(\theta_{t-1}) \tag{5.8}$$

$$\theta_t = \theta_{t-1} + v_t, \tag{5.9}$$

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) [93] 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,

$$v_t = \mu_{t-1} v_{t-1} - \epsilon_{t-1} \nabla f(\theta_{t-1} + \mu_{t-1} v_{t-1}) \tag{5.10}$$

$$\theta_t = \theta_{t-1} + v_t. \tag{5.11}$$

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 [135]. 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.

5.4 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] [109]. 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 [146]. 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.

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

5.4.1 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 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.

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 [70, 122], 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 [136]. The stochastic updates are on minibatches 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 [18]. 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 5.1 shows frame-level and WER evaluations of acoustic models of varying size compared against our baseline GMM recognizer.

Results

Table 5.1 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.

Discussion

To better understand the dynamics of training large DNN acoustic models, we plot training and evaluation WER performance during DNN training. Figure 5.4 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.

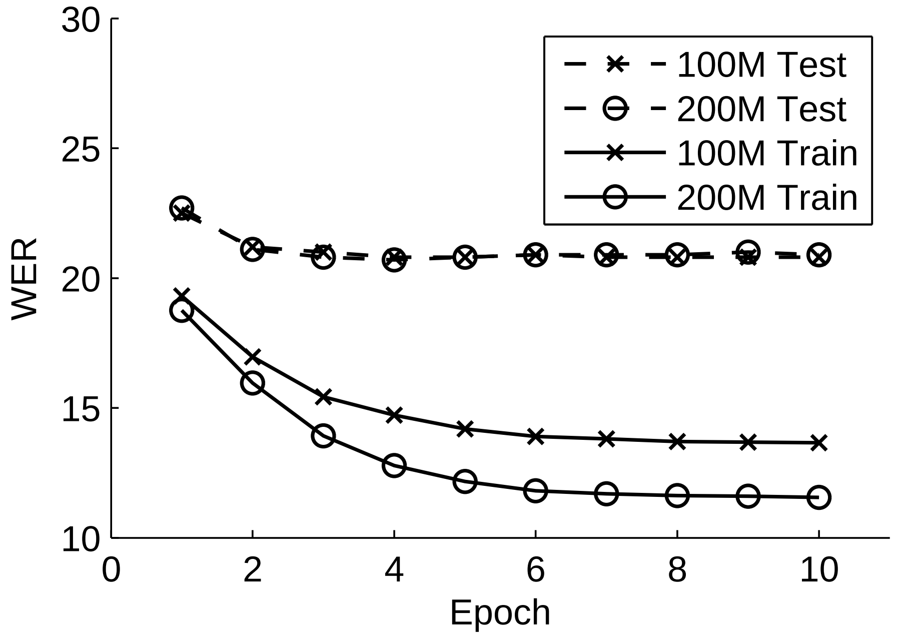


Figure 5.4: Train and test set WER as a function of training epoch for systems with DNN acoustic models of varying size. Each epoch is a single complete pass through the training set. Although the training error rate is substantially lower for large models, there is no gain in test set performance.

5.4.2 Dropout Regularization

Dropout is a recently-introduced technique to prevent over-fitting during DNN training [52]. 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 [63, 152]. [21] found a reduction in WER when using dropout on a 10M parameter DNN acoustic model for a 50 hour broadcast 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 [121]. 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 5.4.1.

Experiments

We train DNN acoustic models with dropout to compare generalization WER performance against that of the DNNs presented in Section 5.4. 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.

Results

Table 5.2 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.

5.4.3 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 [13] 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.

Results

By analyzing the training and test WER curves in Figure 5.4 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.

5.4.4 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 [13]. 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 [128].

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 [9], 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 [52].

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.

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.

Results

Table 5.2 compares the final test set performance of DNNs trained with early re-alignment to a baseline model as well as DNNs trained with dropout regularization. Realignment after five epochs is beneficial compared to the baseline DNN system, but slightly worse than a system which realigns after two epochs of training. Early

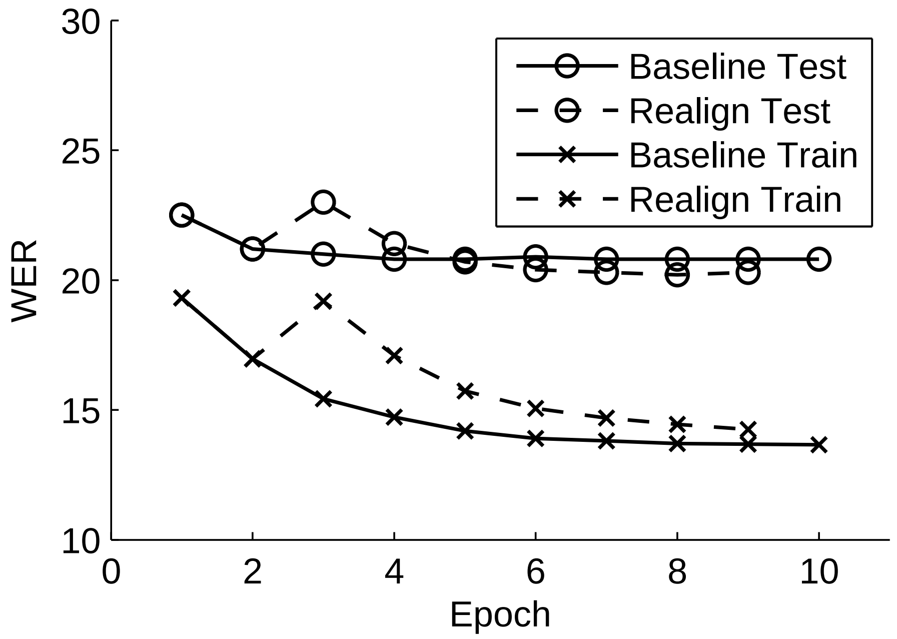


Figure 5.5: WER as a function of DNN training epoch for systems with DNN acoustic models trained with and without label realignment after epoch 2. A DNN which re-generates its training labels with a forced alignment early during optimization generalizes much better to test data than a DNN which converges to the original labels.

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.

Discussion

Figure 5.5 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.

5.5 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).

5.5.1 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 [121].

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 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.

5.5.2 Results

Table 5.3 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.

5.5.3 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 [124]. 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.

5.6 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, over-fitting 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.

5.6.1 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 [17]. 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

[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.

Table 5.4 and Table 5.5. [3]

5.6.2 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 5.3.5 to determine which optimization algorithm to use in the rest of the experiments on this corpus.

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.

Results

Table 5.4 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 $\mu_{max} \in 0.9, 0.95, 0.99$

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

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.

5.6.3 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 5.5 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 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

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

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 5.7 we explore more thoroughly the frame classification performance of the DNNs presented here.

5.6.4 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 5.5 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.

5.7 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.

Figure 5.6 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 5.5. We see that decreases in

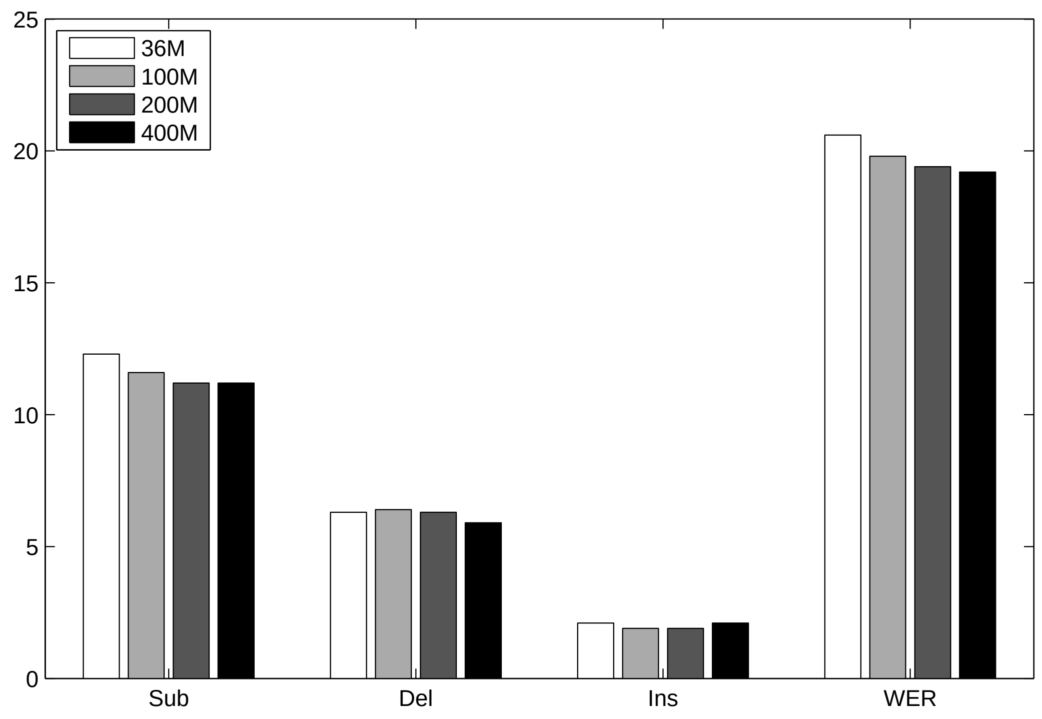


Figure 5.6: Eval2000 WER of 5 hidden layer DNN systems of varying total parameter count. WER is broken into its sub-components – insertions, substitutions, and deletions.

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.

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 5.7 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 5.6 and Table 5.5. 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 to those observed with DNN acoustic models on smaller corpora [54]. 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.

5.8 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 5.3 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.

5.8.1 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 5.8 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. 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 5.9 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.

5.8.2 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 5.10 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.

5.9 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.

Table 5.1: 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

Table 5.2: 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

Table 5.3: 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

Table 5.4: 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

Table 5.5: 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

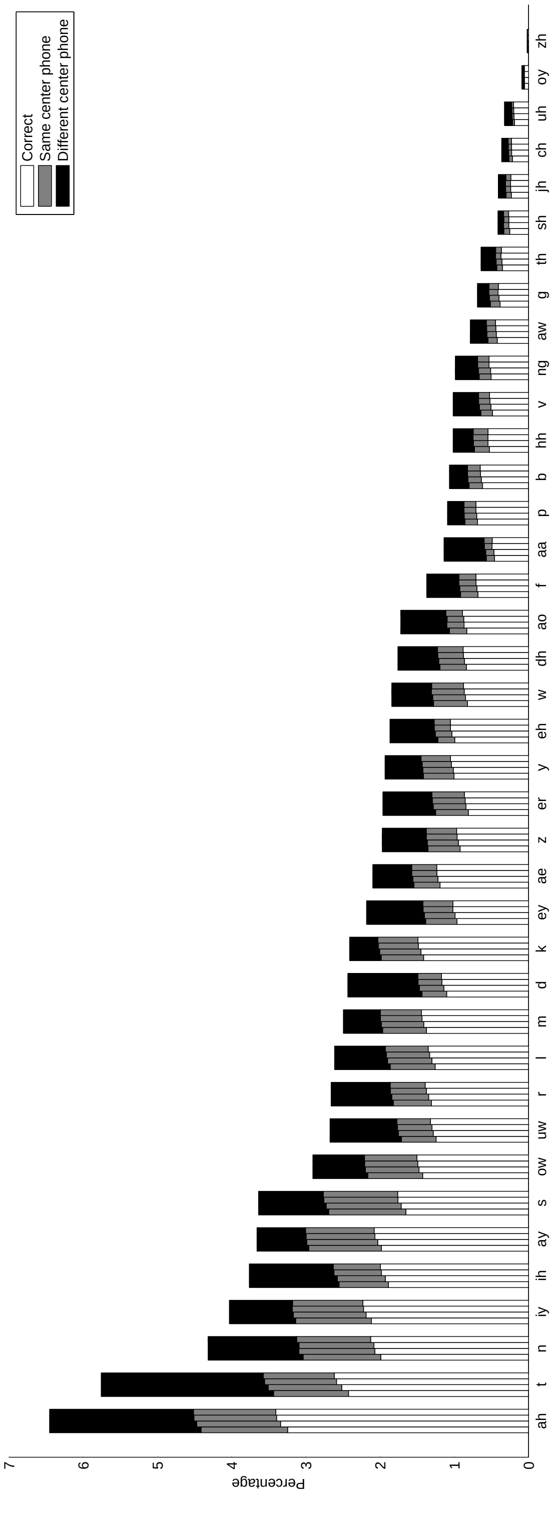


Figure 5.7: Senone accuracy of 5 hidden layer DNN systems of varying total parameter count. Accuracy is grouped by base phone and we report the percentage correct, mis-classifications which chose a senone of the same base phone, and mis-classifications which chose a senone of a different base phone. 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.

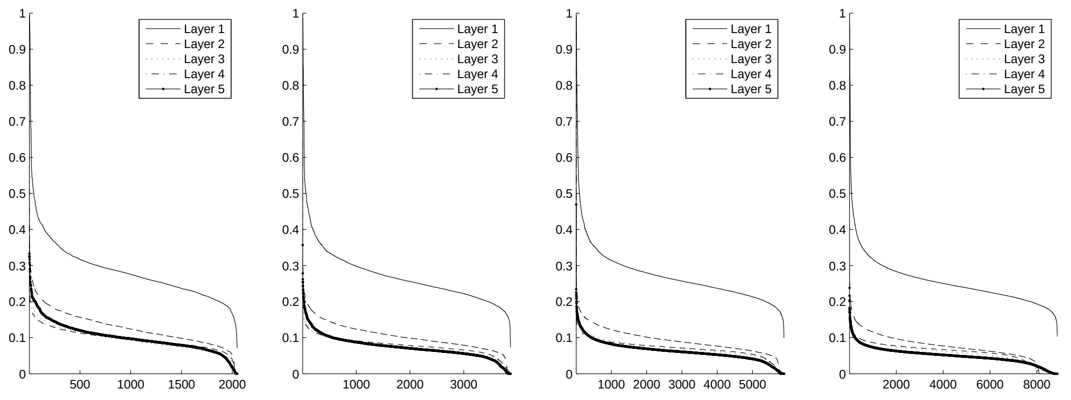


Figure 5.8: Empirical activation probability of hidden units in each hidden layer layer of 5 hidden layer DNNs. Hidden units (x axis) are sorted by their probability of activation. We consider any positive value as active ($h(x) > 0$). Each sub-figure corresponds to a different model size of 36M, 100M, and 200M total parameters from left to right.

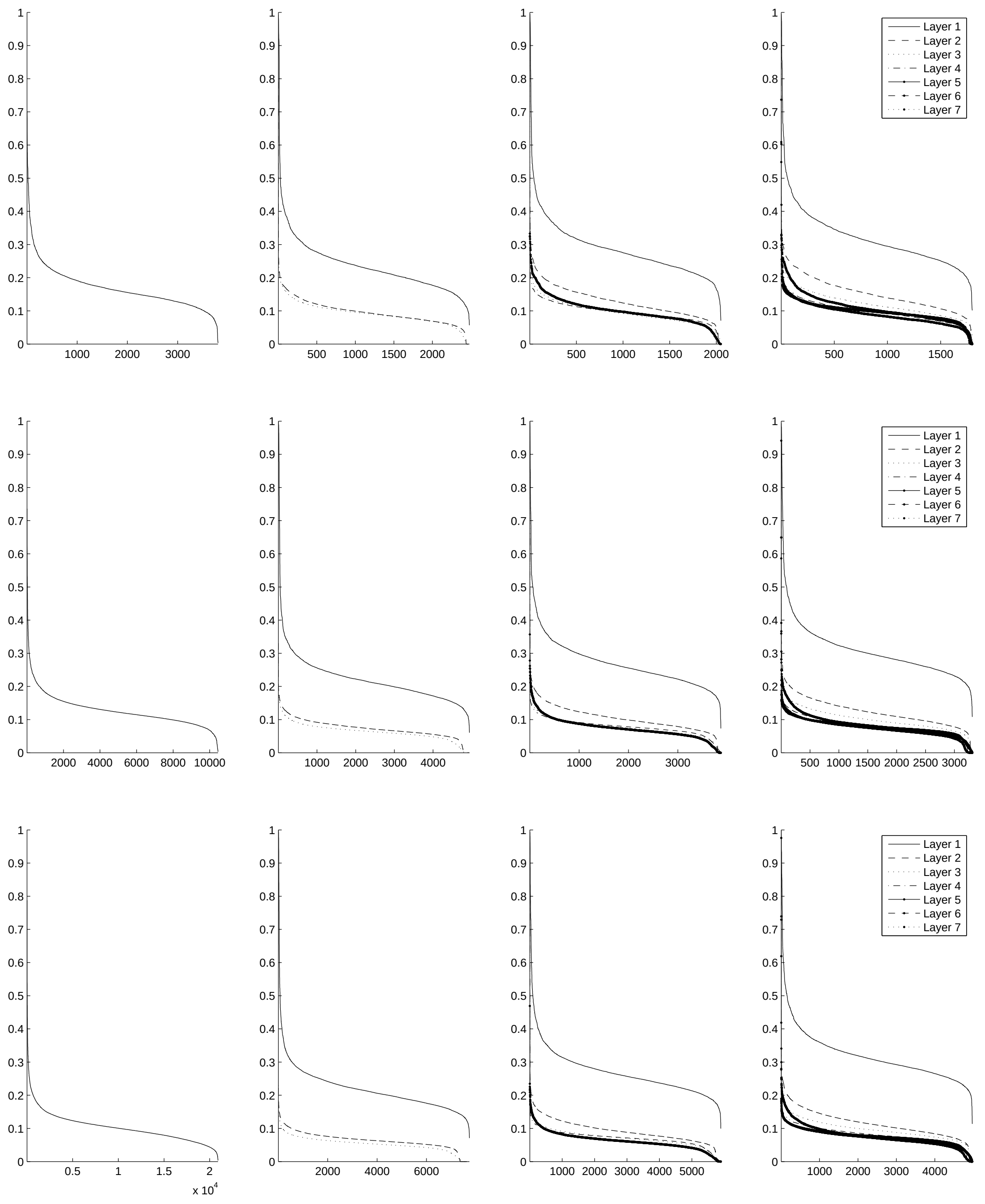


Figure 5.9: Empirical activation probability of hidden units in each hidden layer layer of DNNs with varying numbers of hidden layers. Each row contains DNNs of 36M (top), 100M (middle), and 200M total parameters (bottom). From left to right, each sub-figure shows a DNN with 1, 3, 5, and 7 hidden layers. Hidden units (x axis) are sorted by their probability of activation. We consider any positive value as active ($h(x) > 0$).

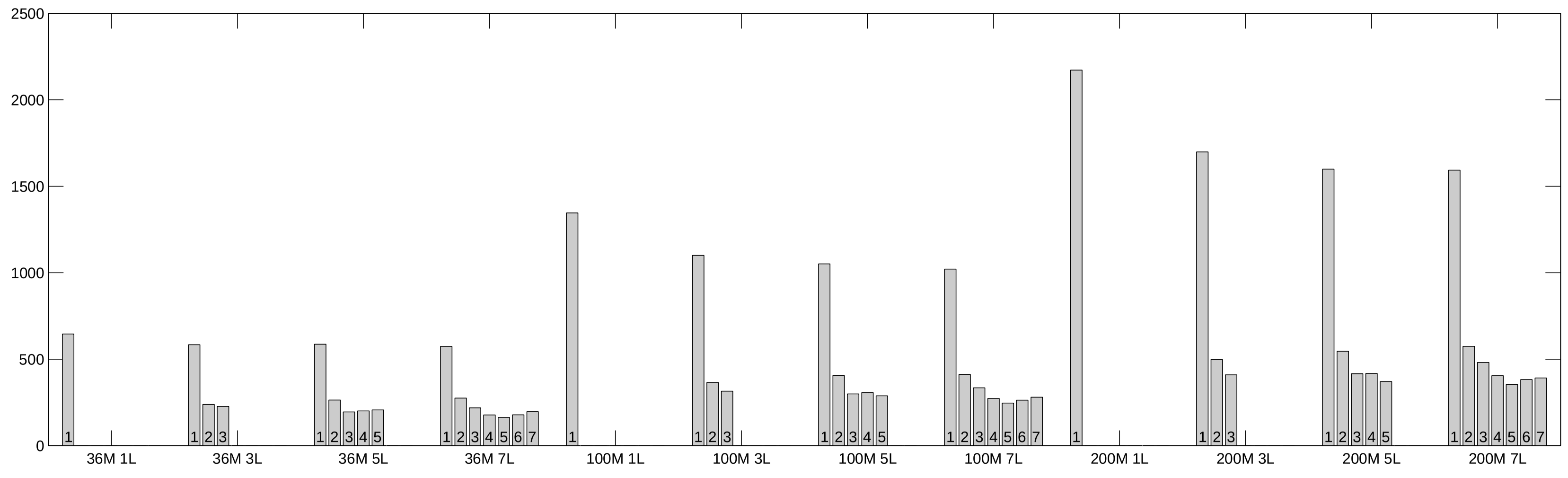


Figure 5.10: Effective code length for each hidden layer in DNNs with varying total size and depth. We compute the number on non-zero hidden unit activations for a given input, and then average over a large sample of inputs. Plots show the average number of units active in each hidden layer of DNNs of varying depth and total size. Within each sub-plot layers are ordered left to right from first to final hidden layer.

Chapter 6

Lexicon-Free Conversational Speech Recognition with Neural Networks

Users increasingly interact with natural language understanding systems via conversational speech interfaces. Google Now, Microsoft Cortana, and Apple Siri are all systems which rely on spoken language understanding, where transcribing speech is a single step within a larger system. Building such systems is difficult because spontaneous, conversational speech naturally contains repetitions, disfluencies, partial words, and out of vocabulary (OOV) words [24, 53]. Moreover, SLU systems must be robust to transcription errors, which can be quite high depending on the task and domain.

Modern systems for large vocabulary continuous speech recognition (LVCSR) use hidden Markov models (HMMs) to handle sequence processing, word-level language models, and a pronunciation lexicon to map words into phonetic pronunciations [127]. Traditional systems use Gaussian mixture models (GMMs) to build a mapping from sub-phonetic states to audio input features. The resulting speech recognition system contains many sub-components, linguistic assumptions, and typically over ten thousand lines of source code. Within the past few years LVCSR systems improved by replacing GMMs with deep neural networks (DNNs) [22, 52]. Both HMM-GMM and HMM-DNN systems remain difficult to build, and nearly impossible to efficiently optimize for downstream SLU tasks. As a result, SLU researchers typically operate on an n-best list of possible transcriptions and treat the LVCSR system as a black box.

Recently [46] demonstrated an approach to LVCSR using a neural network trained with the connectionist temporal classification (CTC) loss function [45]. Using the CTC loss function the authors built a neural network which directly maps audio input features to a sequence of characters. By re-ranking word-level n-best lists generated from an HMM-DNN system the authors obtained competitve results on the Wall Street Journal corpus.

Our work builds upon the foundation introduced by [46]. Rather than reasoning at the word level, we train and decode our system by reasoning entirely at the character-level. By reasoning over characters we eliminate the need for a lexicon, and enable transcribing new words, fragments, and disfluencies. We train a deep bi-directional recurrent neural network (DBRNN) to directly map acoustic input to characters using the CTC loss function introduced by [46]. We are able to efficiently and accurately perform transcription using only our DBRNN and a character-level language model (CLM) whereas previous work relied on n-best lists from a baseline HMM-DNN system. We demonstrate our approach using the challenging Switchboard telephone conversation transcription task, achieving a word error rate competitive with existing baseline HMM-GMM systems. To our knowledge, this is the first entirely neural-network-based system to achieve strong speech transcription results on a conversational speech task. We analyze qualitative differences between transcriptions produced by our lexicon-free approach and transcriptions produced by a standard speech recognition system. Finally, we evaluate the impact of large context neural network character language models as compared to standard n-gram models within our framework.

6.1 Model

We address the complete LVCSR problem. Our system trains on utterances which are labeled by word-level transcriptions and contain no indication of when words occur within an utterance. Our approach consists of two neural networks which we integrate during a beam search decoding procedure. Our first neural network, a DBRNN, maps acoustic input features to a probability distribution over characters at each time step. Our second system component is a neural network character language model. Neural network CLMs enable us to leverage high order n-gram contexts without dramatically increasing the number of free parameters in our language model. To facilitate further work with our approach we make our source code publicly available. [1]

6.1.1 Connectionist Temporal Classification

We train neural networks using the CTC loss function to do maximum likelihood training of letter sequences given acoustic features as input. This is a direct, discriminative approach to building a speech recognition system in contrast to the generative, noisy-channel approach which motivates HMM-based speech recognition systems. Our application of the CTC loss function follows the approach introduced by [46], but we restate the approach here for completeness.

CTC is a generic loss function to train systems on sequence problems where the alignment between the input and output sequence are unknown. CTC accounts for time warping of the output sequence relative to the input sequence, but does not model possible re-orderings. Re-ordering is a problem in machine translation, but is not an issue when working with speech recognition – our transcripts provide the exact ordering in which words occur in the input audio.

Given an input sequence X of length T, CTC assumes the probability of a length T character sequence C is given by,

$$p(C|X) = \prod_{t=1}^{T} p(c_t|X). \tag{6.1}$$

[1]Our code is available here: `http://www.andrewmaas.com`.

This assumes that character outputs at each timestep are conditionally independent given the input. The distribution $p(c_t|X)$ is the output of some predictive model.

CTC assumes our ground truth transcript is a character sequence W with length τ where $\tau \leqslant T$. As a result, we need a way to construct possibly shorter output sequences from our length T sequence of character probabilities. The CTC *collapsing function* achieves this by introducing a special *blank* symbol, which we denote using _, and collapsing any repeating characters in the original length T output. This output symbol contains the notion of *junk* or *other* so as to not produce a character in the final output hypothesis. Our transcripts W come from some set of symbols ζ' but we reason over $\zeta = \zeta' \cup \_$.

We denote the collapsing function by $\kappa(\cdot)$ which takes an input string and produces the unique collapsed version of that string. As an example, here are the set of strings Z of length $T = 3$ such that $\kappa(z) = \texttt{hi}$, $\forall z \in Z$:

$$Z = \{\texttt{hhi}, \texttt{hii}, \texttt{\_hi}, \texttt{h\_i}, \texttt{hi\_}\}.$$

There are a large number of possible length T sequences corresponding to a final length τ transcript hypothesis. The CTC objective function $\mathcal{L}_{\mathrm{CTC}}(X, W)$ is a likelihood of the correct final transcript W which requires integrating over the probabilities of all length T character sequences consistent with W after applying the collapsing function,

$$\begin{aligned} \mathcal{L}_{\mathrm{CTC}}(X, W) &= \sum_{C:\ \kappa(C)=W} p(C|X) \\ &= \sum_{C:\ \kappa(C)=W} \prod_{t=1}^{T} p(c_t|X). \end{aligned} \tag{6.2}$$

Using a dynamic programming approach we can exactly compute this loss function efficiently as well as its gradient with respect to our probabilities $p(c_t|X)$.

6.1.2 Deep Bi-Directional Recurrent Neural Networks

Our loss function requires at each time t a probability distribution $p(c|x_t)$ over characters c given input features x_t. We model this distribution using a DBRNN because it

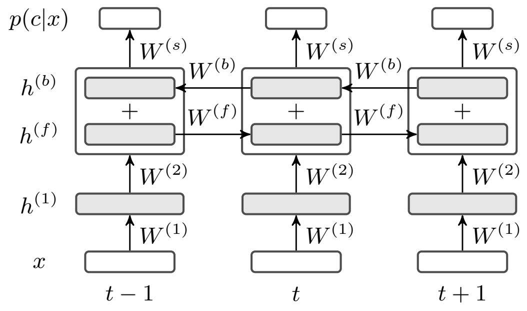


Figure 6.1: Deep bi-directional recurrent neural network to map input audio features X to a distribution $p(c|x_t)$ over output characters at each timestep t. The network contains two hidden layers with the second layer having bi-directional temporal recurrence

provides an expressive model which explicitly accounts for the sequential relationships that should exist in our task. Moreover, the DBRNN is a relatively straightforward neural network architecture to specify, and allows us to learn parameters from data rather than more explicitly specifying how to convert audio features into characters. Figure 6.1.2 shows a DBRNN with two hidden layers.

A DBRNN computes the distribution $p(c|x_t)$ using a series of hidden layers followed by an output layer. Given an input vector x_t the first hidden layer activations are a vector computed as,

$$h^{(1)} = \sigma(W^{(1)T}x_t + b^{(1)}), \tag{6.3}$$

where the matrix $W^{(1)}$ and vector $b^{(1)}$ are the weight matrix and bias vector. The function $\sigma(\cdot)$ is a point-wise nonlinearity. We use $\sigma(z) = \min(\max(z, 0), \mu)$. This is a rectified linear activation function clipped to a maximum possible activation of μ to prevent overflow. Rectified linear hidden units have been show to work well in general for deep neural networks, as well as for acoustic modeling of speech data [40, 168, 21, 75]

We select a single hidden layer j of the network to have temporal connections. Our temporal hidden layer representation $h^{(j)}$ is the sum of two partial hidden layer representations,

$$h_t^{(j)} = h_t^{(f)} + h_t^{(b)}. \tag{6.4}$$

The representation $h^{(f)}$ uses a weight matrix $W^{(f)}$ to propagate information forwards in time. Similarly, the representation $h^{(b)}$ propagates information backwards in time using a weight matrix $W^{(b)}$. These partial hidden representations both take input from the previous hidden layer $h^{(j-1)}$ using a weight matrix $W^{(j)}$,

$$\begin{aligned} h_t^{(f)} &= \sigma(W^{(j)T} h_t^{(j-1)} + W^{(f)T} h_{t-1}^{(f)} + b^{(j)}), \\ h_t^{(b)} &= \sigma(W^{(j)T} h_t^{(j-1)} + W^{(b)T} h_{t+1}^{(b)} + b^{(j)}). \end{aligned} \tag{6.5}$$

Note that the recurrent forward and backward hidden representations are computed entirely independently from each other. As with the other hidden layers of the network we use $\sigma(z) = \min(\max(z, 0), \mu)$.

All hidden layers aside from the first hidden layer and temporal hidden layer use a standard dense weight matrix and bias vector,

$$h^{(i)} = \sigma(W^{(i)T} h^{(i-1)} + b^{(i)}). \tag{6.6}$$

DBRNNs can have an arbitrary number of hidden layers, but we assume that only one hidden layer contains temporally recurrent connections.

The model outputs a distribution $p(c|x_t)$ over a set of possible characters ζ using a *softmax* output layer. We compute the softmax layer as,

$$p(c = c_k|x_t) = \frac{\exp(-(W_k^{(s)T} h^{(:)} + b_k^{(s)}))}{\sum_{j=1}^{|\zeta|} \exp(-(W_j^{(s)T} h^{(:)} + b_j^{(s)}))}, \tag{6.7}$$

where $W_k^{(s)}$ is the k'th column of the output weight matrix $W^{(s)}$ and $b_k^{(s)}$ is a scalar bias term. The vector $h^{(:)}$ is the hidden layer representation of the final hidden layer in our DBRNN.

We can directly compute a gradient for all weights and biases in the DBRNN with respect to the CTC loss function and apply batch gradient descent.

6.1.3 Jointly Training on Language Understanding Tasks

The CTC loss function allows us to train a DBRNN directly on the speech transcription problem rather than within a complex HMM-based speech recognizer. The simplicity of this approach enables *joint training* of the DBRNN with respect to both the CTC loss function for transcription and a downstream SLU task. We now assume that utterances come with a new type of label Y and there is a differentiable loss function for our SLU task, $\mathcal{L}_{\text{SLU}}(X, Y)$. We can then train our DBRNN on the joint loss function,

$$\mathcal{L}_{\text{J}}(X, W, Y) = \mathcal{L}_{\text{CTC}}(X, W) + \mathcal{L}_{\text{SLU}}(X, Y). \tag{6.8}$$

Training on this joint loss only requires adding an output layer to our DBRNN to produce predictions for the SLU task. For example, we can add a binary classifier to the DBRNN to perform disfluency detection, which is an important task for dialog systems [112]. This joint training approach allows an SLU task to directly inform our entire LVCSR system rather than treat it as a black box. This approach resembles joint training on multiple tasks with neural networks for text-based NLP which previous research found beneficial [20].

6.1.4 Decoding

Our decoding procedure integrates information from the DBRNN and language model to form a single cohesive estimate of the character sequence in a given utterance. For an input sequence X of length T our DBRNN produces a set of probabilities $p(c|x_t),\ t = 1, \ldots, T$. Again, the character probabilities are a categorical distribution over the symbol set ζ.

Decoding Without a Language Model

As a baseline, we use a simple, greedy approach to decoding the DBRNN outputs [46]. The simplest form of decoding does not employ the language model and instead finds the highest probability character transcription given only the DBRNN outputs.

This process selects a transcript hypothesis W^* by making a greedy approximation,

$$\begin{aligned} W^* &= \arg\max_W p(W|X) \approx \kappa(\arg\max_C p(C|X)) \\ &= \kappa(\arg\max_C \prod_{t=1}^{T} p(c_t|X)). \end{aligned} \tag{6.9}$$

This decoding procedure ignores the issue of many time-level character sequences mapping to the same final hypothesis, and instead considers only the most probable character at each point in time. Because our model assumes the character labels for each timestep are conditionally independent, C^* is simply the most probable character at each timestep in our DBRNN output. As a result, this decoding procedure is very fast to compute, requiring only time $O(T|\zeta|)$.

Beam Search Decoding

To decode while taking language model probabilities into account, we use a beam search to combine a character language model and the outputs of our DBRNN. This search-based decoding method does not make a greedy approximation and instead assigns probability to a final hypothesis by integrating over all character sequences consistent with the hypothesis under our collapsing function $\kappa(\cdot)$. Algorithm 1 outlines our decoding procedure.

We note that our decoding procedure is significantly simpler, and in practice faster, than previous decoding procedures applied to CTC models. This is due to reasoning at the character level without a lexicon so as to not introduce difficult multi-level constraints to obey during the decoding search procedure. While a softmax over words is typically the bottleneck in neural network language models, a softmax over possible characters is comparatively cheap to compute. Our character language model is applied at every time step, while word models can only be applied when we consider adding a space or by computing the likelihood of a sequence being the prefix of a word in the lexicon [46]. Additionally, our lexicon-free approach removes the difficulties of handling OOV words during decoding, which is typically a troublesome issue in speech recognition systems.

Method	CER	EV	CH	SWBD
HMM-GMM	23.0	29.0	36.1	21.7
HMM-DNN	17.6	21.2	27.1	15.1
HMM-SHF	NR	NR	NR	12.4
CTC no LM	27.7	47.1	56.1	38.0
CTC+5-gram	25.7	39.0	47.0	30.8
CTC+7-gram	24.7	35.9	43.8	27.8
CTC+NN-1	24.5	32.3	41.1	23.4
CTC+NN-3	24.0	30.9	39.9	21.8
CTC+RNN	24.9	33.0	41.7	24.2

Table 6.1: Character error rate (CER) and word error rate results on the Eval2000 test set. We report word error rates on the full test set (EV) which consists of the Switchboard (SWBD) and CallHome (CH) subsets. As baseline systems we use an HMM-GMM system and HMM-DNN system. We evaluate our DBRNN trained using CTC by decoding with several character-level language models: 5-gram, 7-gram, densely connected neural networks with 1 and 3 hidden layers (NN-1, and NN-3), as well as an RNN with a single hidden layer. We additionally include results from a state-of-the-art HMM-based system (HMM-DNN-SHF) which does not report performance on all metrics we evaluate (NR).

6.2 Experiments

We perform LVCSR experiments on the 300 hour Switchboard conversational telephone speech corpus (LDC97S62). Switchboard utterances are taken from approximately 2,400 conversations among 543 speakers. Each pair of speakers had never met, and converse no more than once about a given topic chosen randomly from a set of 50 possible topics. Utterances exhibit many rich, complex phenomena that make spoken language understanding difficult. Table 6.2 shows example transcripts from the corpus.

For evaluation, we report word error rate (WER) and character error rate (CER) on the HUB5 Eval2000 dataset (LDC2002S09). This test set consists of two subsets, Switchboard and CallHome. The CallHome subset represents a mismatched test condition as it was collected from phone conversations among family and friends rather than strangers directed to discuss a particular topic. The mismatch makes the CallHome subset quite difficult overall. The Switchboard evaluation subset is substantially easier, and represents a better match of test data to our training corpus. We report WER and CER on the test set as a whole, and additionally report WER for each subset individually.

6.2.1 Baseline Systems

We build two baseline LVCSR systems to compare our approach to standard HMM-based approaches. First, we build an HMM-GMM system using the Kaldi open-source toolkit[2] [109]. The baseline recognizer has 8,986 sub-phone states and 200K Gaussians trained using maximum likelihood. Input features are speaker-adapted MFCCs. Overall, the baseline GMM system setup largely follows the existing `s5b` Kaldi recipe, and we defer to previous work for details [145].

We additionally built an HMM-DNN system by training a DNN acoustic model using maximum likelihood on the alignments produced by our HMM-GMM system. The DNN consists of five hidden layers, each with 2,048 hidden units, for a total of approximately 36 million (M) free parameters in the acoustic model.

Both baseline systems use a bigram language model built from the 3M words in the Switchboard transcripts interpolated with a second bigram language model built from 11M words on the Fisher English Part 1 transcripts (LDC2004T19). Both LMs are trained using interpolated Kneser-Ney smoothing. For context we also include WER results from a state-of-the-art HMM-DNN system built with quinphone phonetic context and Hessian-free sequence-discriminative training [123].

6.2.2 DBRNN Training

We train a DBRNN using the CTC loss function on the entire 300hr training corpus. The input features to the DBRNN at each timestep are MFCCs with context window of ± 10 frames. The DBRNN has 5 hidden layers with the third containing recurrent

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

connections. All layers have 1824 hidden units, giving about 20M trainable parameters. In preliminary experiments we found that choosing the middle hidden layer to have recurrent connections led to the best results.

The output symbol set ζ consists of 33 characters including the special blank character. Note that because speech recognition transcriptions do not contain proper casing or punctuation, we exclude capital letters and punctuation marks with the exception of "`-`", which denotes a partial word fragment, and "`'`", as used in contractions such as "`can't`."

We train the DBRNN from random initial parameters using the gradient-based Nesterov's accelerated gradient (NAG) algorithm as this technique is sometimes beneficial as compared with standard stochastic gradient descent for deep recurrent neural network training [136]. The NAG algorithm uses a step size of 10^{-5} and a momentum of 0.95. After each epoch we divide the learning rate by 1.3. Training for 10 epochs on a single GTX 570 GPU takes approximately one week.

6.2.3 Character Language Model Training

The Switchboard corpus transcripts alone are too small to build CLMs which accurately model general orthography in English. To learn how to spell words more generally we train our CLMs using a corpus of 31 billion words gathered from the web [49]. Our language models use sentence start and end tokens, `<s>` and `</s>`, as well as a `<null>` token for cases when our context window extends past the start of a sentence.

We build 5-gram and 7-gram CLMs with modified Kneser-Ney smoothing using the KenLM toolkit [49]. Building traditional n-gram CLMs is for $n > 7$ becomes increasingly difficult as the model free parameters and memory footprint become unwieldy. Our 7-gram CLM is already 21GB; we were not able to build higher order n-gram models to compare against our 20-gram neural network CLMs.

Following work illustrating the effectiveness of neural network CLMs [137] and word-level LMs for speech recognition [88], we train and evaluate two variants of neural network CLMs: standard feedfoward deep neural networks (DNNs) and a recurrent neural network (RNN). The RNN CLM takes one character at a time as input, while the non-recurrent CLM networks use a context window of 19 characters. All neural network CLMs use the rectified linear activation function, and the layer sizes are selected such that each has about 5M parameters (20MB).

The DNN models are trained using standard backpropagation using Nesterov's accelerated gradient with a learning rate of 0.01 and momentum of 0.95 and a batch size of 512. The RNN is trained using backpropagation through time with a learning rate of 0.001 and batches of 128 utterances. For both model types we halve the learning rate after each epoch. The DNN models were trained for 10 epochs, and the RNN for 5 epochs.

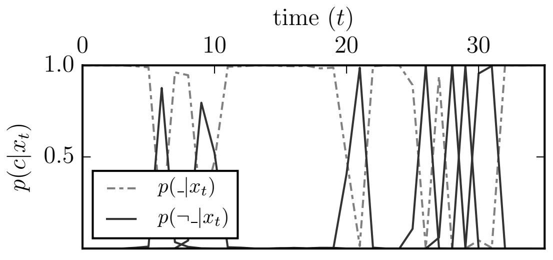
All neural network CLMs were trained using a combination of the Switchboard and Fisher training transcripts which in total contain approximately 23M words.

6.2.4 Results

After training the DBRNN and CLMs we run decoding on the Eval2000 test set to obtain CER and WER results. For all experiments using a CLM we use our beam search decoding algorithm with $\alpha = 1.25$, $\beta = 1.5$ and a beam width of 100. We found that larger beam widths did not significantly improve performance. Table 6.1 shows results for the DBRNN as well as baseline systems.

The DBRNN performs best with the 3 hidden layer DNN CLM. This DBRNN+NN-3 attains both CER and WER performance comparable to the HMM-GMM baseline system, albeit substantially below the HMM-DNN system. Neural networks provide a clear gain as compared to standard n-gram models when used for DBRNN decoding, although the RNN CLM does not produce any gain over the best DNN CLM.

Without a language model the greedy DBRNN decoding procedure loses relatively little in terms of CER as compared with the DBRNN+NN-3 model. However, this 3% difference in CER translates to a 16% gap in WER on the full Eval2000 test set. Generally we observe that small CER differences translate to large WER differences. In terms of character-level performance it appears as if the DBRNN alone performs well using only acoustic input data. Adding a CLM yields a somewhat small CER



s: `_____o__hh__________ ____y_eahh___`

$\kappa(s)$: `oh yeah`

Figure 6.2: DBRNN character probabilities over time for a single utterance along with the per-frame most likely character string s and the collapsed output $\kappa(s)$. Due to space constraints we only show a distinction in line type between the blank symbol _ and non-blank symbols.

improvement but guides proper spelling of words to produce a large reduction in WER.

6.3 Analysis

To better see how the DBRNN performs transcription we show the output probabilities $p(c|x)$ for an example utterance in Figure 6.2. The model tends to output mostly blank characters and only spike long enough for a character to be the most likely symbol for a few frames at a time. The dominance of the blank class is not forced, but rather learned by the DBRNN during training. We hypothesize that this spiking behavior results in more stable results as the DBRNN only produces a character when its confidence of seeing that character rises above a certain threshold. Note that this a dramatic contrast to HMM-based LVCSR systems which, due to the nature of generative models, attempt to explain almost all timesteps as belonging to a phonetic substate.

Next we qualitatively compare the DBRNN and HMM-GMM system outputs to better understand how the DBRNN approach might interact with SLU systems. This comparison is especially interesting because our best DBRNN system and the HMM-GMM system have comparable WERs, removing the confound of overall quality when

#	Method	Transcription
(1)	Truth	yeah i went into the i do not know what you think of *fidelity* but
(1)	HMM-GMM	yeah when the i don't know what you think of fidel it even them
(1)	CTC+CLM	yeah i went to i don't know what you think of fidelity but um
(2)	Truth	no no speaking of weather do you carry a altimeter slash *barometer*
(2)	HMM-GMM	no i'm not all being the weather do you uh carry a uh helped emitters last brahms her
(2)	CTC+CLM	no no beating of whether do you uh carry a uh a time or less barometer
(3)	Truth	i would ima- well yeah it is i know you are able to stay home with them
(3)	HMM-GMM	i would amount well yeah it is i know um you're able to stay home with them
(3)	CTC+CLM	i would ima- well yeah it is i know uh you're able to stay home with them

Table 6.2: Example test set utterances with a ground truth transcription and hypotheses from our method (CTC+CLM) and a baseline HMM-GMM system of comparable overall WER. The words *fidelity* and *barometer* are not in the lexicon of the HMM-GMM system.

comparing hypotheses. Table 6.2 shows example test set utterances along with transcription hypotheses from the HMM-GMM and DBRNN+NN-3 systems.

The DBRNN sometimes correctly transcribes OOV words with respect to our audio training corpus. We find that OOVs tend to trigger clusters of errors in the HMM-GMM system, an observation that has been systematically explored in previous work [44]. As shown in example utterance (3), HMM-GMM errors can introduce word substitution errors which may alter meaning whereas the DBRNN system outputs word fragments or non-words which are phonetically similar and may be useful input features for SLU systems. Unfortunately the Eval2000 test set does not offer a rich set of utterances containing OOVs or fragments to perform a deeper analysis. The HMM-GMM and best DBRNN system achieve identical WERs on the subset of test utterances containing OOVs and the subset of test utterances containing fragments.

6.4 Conclusion

We presented an LVCSR system consisting of two neural networks integrated via beam search decoding that matches the performance of an HMM-GMM system on the challenging Switchboard corpus. Our model vastly reduces the number of subcomponents and overall system complexity required for LVCSR. Furthermore, our method can be written in about 1,000 lines of code — roughly an order of magnitude less than high performance HMM-GMM systems. Operating entirely at the character level yields a system which does not require assumptions about a lexicon or pronunciation dictionary, instead learning orthography and phonics directly from data. We hope the simplicity of our approach will facilitate future research in improving LVCSR with CTC-based systems, and jointly training LVCSR systems with SLU tasks. DNNs have already shown great results as acoustic models in HMM-DNN systems. We free the neural network from its complex HMM infrastructure, which we view as the first step towards the next wave of advances in speech recognition and language understanding.

Algorithm 1 Beam Search Decoding: Given the likelihoods from our DBRNN and our character language model, for each time step t and for each string s in our current previous hypothesis set Z_{t-1}, we consider extending s with a new character. Blanks and repeat characters with no separating blank are handled separately. For all other character extensions, we apply our character language model when computing the probability of s. We initialize Z_0 with the empty string $\varnothing$. Notation: ζ': character set excluding "_", $s + c$: concatenation of character c to string s, $|s|$: length of s, $p_{\text{b}}(c|x_{1:t})$ and $p_{\text{nb}}(c|x_{1:t})$: probability of s ending and not ending in blank conditioned on input up to time t, $p_{\text{tot}}(c|x_{1:t})$: $p_{\text{b}}(c|x_{1:t}) + p_{\text{nb}}(c|x_{1:t})$

```
**Inputs** CTC likelihoods $p_{\text{ctc}}(c|x_t)$, character language model $p_{\text{clm}}(c|s)$
**Parameters** language model weight $\alpha$, insertion bonus $\beta$, beam width $k$
**Initialize** $Z_0 \leftarrow \{\varnothing\}$, $p_{\text{b}}(\varnothing|x_{1:0}) \leftarrow 1$, $p_{\text{nb}}(\varnothing|x_{1:0}) \leftarrow 0$
**for** $t = 1, \ldots, T$ **do**
    $Z_t \leftarrow \{\}$
    **for** $s$ **in** $Z_{t-1}$ **do**
        $p_{\text{b}}(s|x_{1:t}) \leftarrow p_{\text{ctc}}(\_|x_t)p_{\text{tot}}(s|x_{1:t-1})$          ▷ Handle blanks
        $p_{\text{nb}}(s|x_{1:t}) \leftarrow p_{\text{ctc}}(c|x_t)p_{\text{nb}}(s|x_{1:t-1})$          ▷ Handle repeat character collapsing
        Add $s$ to $Z_t$
        **for** $c$ **in** $\zeta'$ **do**
            $s^+ \leftarrow s + c$
            **if** $c \neq s_{t-1}$ **then**
                $p_{\text{nb}}(s^+|x_{1:t}) \leftarrow p_{\text{ctc}}(c|x_t)p_{\text{clm}}(c|s)^\alpha p_{\text{tot}}(c|x_{1:t-1})$
            **else**
                $p_{\text{nb}}(s^+|x_{1:t}) \leftarrow p_{\text{ctc}}(c|x_t)p_{\text{clm}}(c|s)^\alpha p_{\text{b}}(c|x_{1:t-1})$          ▷ Repeat characters
have "_" between
            **end if**
            Add $s^+$ to $Z_t$
        **end for**
    **end for**
    $Z_t \leftarrow k$ most probable $s$ by $p_{\text{tot}}(s|x_{1:t})|s|^\beta$ in $Z_t$          ▷ Apply beam
**end for**
**Return** $\arg\max_{s \in Z_t} \; p_{\text{tot}}(s|x_{1:T})|s|^\beta$
```

Chapter 7

Learning Word Representations for Sentiment Analysis

Word representations are a critical component of many natural language processing systems. It is common to represent words as indices in a vocabulary, but this fails to capture the rich relational structure of the lexicon. Vector-based models do much better in this regard. They encode continuous similarities between words as distance or angle between word vectors in a high-dimensional space. The general approach has proven useful in tasks such as word sense disambiguation, named entity recognition, part of speech tagging, and document retrieval [141, 20, 140].

In this paper, we present a model to capture both semantic and sentiment similarities among words. The semantic component of our model learns word vectors via an unsupervised probabilistic model of documents. However, in keeping with linguistic and cognitive research arguing that expressive content and descriptive semantic content are distinct [60, 57, 107], we find that this basic model misses crucial sentiment information. For example, while it learns that *wonderful* and *amazing* are semantically close, it doesn't capture the fact that these are both very strong positive sentiment words, at the opposite end of the spectrum from *terrible* and *awful*.

Thus, we extend the model with a supervised sentiment component that is capable of embracing many social and attitudinal aspects of meaning [163, 2, 4, 99, 43, 131]. This component of the model uses the vector representation of words to predict the sentiment annotations on contexts in which the words appear. This causes words expressing similar sentiment to have similar vector representations. The full objective function of the model thus learns semantic vectors that are imbued with nuanced sentiment information. In our experiments, we show how the model can leverage document-level sentiment annotations of a sort that are abundant online in the form of consumer reviews for movies, products, etc. The technique is sufficiently general to work also with continuous and multi-dimensional notions of sentiment as well as non-sentiment annotations (e.g., political affiliation, speaker commitment).

After presenting the model in detail, we provide illustrative examples of the vectors it learns, and then we systematically evaluate the approach on document-level and sentence-level classification tasks. Our experiments involve the small, widely used sentiment and subjectivity corpora of [98], which permits us to make comparisons with a number of related approaches and published results. We also show that this dataset contains many correlations between examples in the training and testing sets. This leads us to evaluate on, and make publicly available, a large dataset of informal movie reviews from the Internet Movie Database (IMDB).

7.1 Related work

The model we present in the next section draws inspiration from prior work on both probabilistic topic modeling and vector-spaced models for word meanings.

Latent Dirichlet Allocation (LDA; [8]) is a probabilistic document model that assumes each document is a mixture of latent topics. For each latent topic T, the model learns a conditional distribution $p(w|T)$ for the probability that word w occurs in T. One can obtain a k-dimensional vector representation of words by first training a k-topic model and then filling the matrix with the $p(w|T)$ values (normalized to unit length). The result is a word–topic matrix in which the rows are taken to represent word meanings. However, because the emphasis in LDA is on modeling topics, not word meanings, there is no guarantee that the row (word) vectors are sensible as points in a k-dimensional space. Indeed, we show in section 7.3 that using LDA in this way does not deliver robust word vectors. The semantic component of our model shares its probabilistic foundation with LDA, but is factored in a manner designed to discover word vectors rather than latent topics. Some recent work introduces extensions of LDA to capture sentiment in addition to topical information [69, 71, 10]. Like LDA, these methods focus on modeling sentiment-imbued topics rather than embedding words in a vector space.

Vector space models (VSMs) seek to model words directly [141]. Latent Semantic Analysis (LSA), perhaps the best known VSM, explicitly learns semantic word vectors by applying singular value decomposition (SVD) to factor a term–document co-occurrence matrix. It is typical to weight and normalize the matrix values prior to SVD. To obtain a k-dimensional representation for a given word, only the entries corresponding to the k largest singular values are taken from the word's basis in the factored matrix. Such matrix factorization-based approaches are extremely successful in practice, but they force the researcher to make a number of design choices (weighting, normalization, dimensionality reduction algorithm) with little theoretical guidance to suggest which to prefer.

Using term frequency (tf) and inverse document frequency (idf) weighting to transform the values in a VSM often increases the performance of retrieval and categorization systems. Delta idf weighting [86] is a supervised variant of idf weighting in which the idf calculation is done for each document class and then one value is subtracted from the other. Martineau and Finin present evidence that this weighting helps with sentiment classification, and [97] systematically explore a number of weighting schemes in the context of sentiment analysis. The success of delta idf weighting in previous work suggests that incorporating sentiment information into VSM values via supervised methods is helpful for sentiment analysis. We adopt this insight, but we are able to incorporate it directly into our model's objective function. (Section 7.3 compares our approach with a representative sample of such weighting schemes.)

7.2 Our Model

To capture semantic similarities among words, we derive a probabilistic model of documents which learns word representations. This component does not require labeled data, and shares its foundation with probabilistic topic models such as LDA. The sentiment component of our model uses sentiment annotations to constrain words expressing similar sentiment to have similar representations. We can efficiently learn parameters for the joint objective function using alternating maximization.

7.2.1 Capturing Semantic Similarities

We build a probabilistic model of a document using a continuous mixture distribution over words indexed by a multi-dimensional random variable θ. We assume words in a document are conditionally independent given the mixture variable θ. We assign a probability to a document d using a joint distribution over the document and θ. The model assumes each word $w_i \in d$ is conditionally independent of the other words given θ. The probability of a document is thus

$$p(d) = \int p(d, \theta) d\theta = \int p(\theta) \prod_{i=1}^{N} p(w_i|\theta) d\theta. \tag{7.1}$$

Where N is the number of words in d and w_i is the i^{th} word in d. We use a Gaussian prior on θ.

We define the conditional distribution $p(w_i|\theta)$ using a log-linear model with parameters R and b. The energy function uses a word representation matrix $R \in \mathbb{R}^{(\beta \times |V|)}$ where each word w (represented as a one-on vector) in the vocabulary V has a β-dimensional vector representation $\phi_w = Rw$ corresponding to that word's column in R. The random variable θ is also a β-dimensional vector, $\theta \in \mathbb{R}^\beta$ which weights each of the β dimensions of words' representation vectors. We additionally introduce a bias b_w for each word to capture differences in overall word frequencies. The energy assigned to a word w given these model parameters is

$$E(w; \theta, \phi_w, b_w) = -\theta^T \phi_w - b_w. \tag{7.2}$$

To obtain the distribution $p(w|\theta)$ we use a softmax,

$$\begin{align} p(w|\theta; R, b) &= \frac{\exp(-E(w; \theta, \phi_w, b_w))}{\sum_{w' \in V} \exp(-E(w'; \theta, \phi_{w'}, b_{w'}))} \tag{7.3} \\ &= \frac{\exp(\theta^T \phi_w + b_w)}{\sum_{w' \in V} \exp(\theta^T \phi_{w'} + b_{w'})}. \tag{7.4} \end{align}$$

The number of terms in the denominator's summation grows linearly in $|V|$, making exact computation of the distribution possible. For a given θ, a word w's occurrence probability is related to how closely its representation vector ϕ_w matches the scaling direction of θ. This idea is similar to the word vector inner product used in the log-bilinear language model of [89].

Equation 7.1 resembles the probabilistic model of LDA [8], which models documents as mixtures of latent topics. One could view the entries of a word vector ϕ as that word's association strength with respect to each latent topic dimension. The random variable θ then defines a weighting over topics. However, our model does not attempt to model individual topics, but instead directly models word probabilities conditioned on the topic mixture variable θ. Because of the log-linear formulation of the conditional distribution, θ is a vector in $\mathbb{R}^\beta$ and not restricted to the unit simplex as it is in LDA.

We now derive maximum likelihood learning for this model when given a set of unlabeled documents D. In maximum likelihood learning we maximize the probability of the observed data given the model parameters. We assume documents $d_k \in D$ are i.i.d. samples. Thus the learning problem becomes

$$\max_{R,b} p(D; R, b) = \prod_{d_k \in D} \int p(\theta) \prod_{i=1}^{N_k} p(w_i|\theta; R, b) d\theta. \tag{7.5}$$

Using *maximum a posteriori* (MAP) estimates for θ, we approximate this learning problem as

$$\max_{R,b} \prod_{d_k \in D} p(\hat{\theta}_k) \prod_{i=1}^{N_k} p(w_i|\hat{\theta}_k; R, b), \tag{7.6}$$

where $\hat{\theta}_k$ denotes the MAP estimate of θ for d_k. We introduce a Frobenious norm regularization term for the word representation matrix R. The word biases b are not regularized reflecting the fact that we want the biases to capture whatever overall word frequency statistics are present in the data. By taking the logarithm and simplifying we obtain the final objective,

$$\nu||R||_F^2 + \sum_{d_k \in D} \lambda||\hat{\theta}_k||_2^2 + \sum_{i=1}^{N_k} \log p(w_i|\hat{\theta}_k; R, b), \tag{7.7}$$

which is maximized with respect to R and b. The hyper-parameters in the model are the regularization weights (λ and ν), and the word vector dimensionality β.

7.2.2 Capturing Word Sentiment

The model presented so far does not explicitly capture sentiment information. Applying this algorithm to documents will produce representations where words that occur together in documents have similar representations. However, this unsupervised approach has no explicit way of capturing which words are predictive of sentiment as opposed to content-related. Much previous work in natural language processing achieves better representations by learning from multiple tasks [20, 36]. Following this theme we introduce a second task to utilize labeled documents to improve our model's word representations.

Sentiment is a complex, multi-dimensional concept. Depending on which aspects of sentiment we wish to capture, we can give some body of text a sentiment label s which can be categorical, continuous, or multi-dimensional. To leverage such labels, we introduce an objective that the word vectors of our model should predict the sentiment label using some appropriate predictor,

$$\hat{s} = f(\phi_w). \tag{7.8}$$

Using an appropriate predictor function $f(x)$ we map a word vector ϕ_w to a predicted sentiment label $\hat{s}$. We can then improve our word vector ϕ_w to better predict the sentiment labels of contexts in which that word occurs.

For simplicity we consider the case where the sentiment label s is a scalar continuous value representing sentiment polarity of a document. This captures the case of many online reviews where documents are associated with a label on a star rating scale. We linearly map such star values to the interval $s \in [0, 1]$ and treat them as a probability of positive sentiment polarity. Using this formulation, we employ a logistic regression as our predictor $f(x)$. We use w's vector representation ϕ_w and regression weights ψ to express this as

$$p(s = 1|w; R, \psi) = \sigma(\psi^T \phi_w + b_c), \tag{7.9}$$

where $\sigma(x)$ is the logistic function and $\psi \in \mathbb{R}^\beta$ is the logistic regression weight vector.

We additionally introduce a scalar bias b_c for the classifier.

The logistic regression weights ψ and b_c define a linear hyperplane in the word vector space where a word vector's positive sentiment probability depends on where it lies with respect to this hyperplane. Learning over a collection of documents results in words residing different distances from this hyperplane based on the average polarity of documents in which the words occur.

Given a set of labeled documents D where s_k is the sentiment label for document d_k, we wish to maximize the probability of document labels given the documents. We assume documents in the collection and words within a document are i.i.d. samples. By maximizing the log-objective we obtain,

$$\max_{R,\psi,b_c} \sum_{k=1}^{|D|} \sum_{i=1}^{N_k} \log p(s_k|w_i; R, \psi, b_c). \tag{7.10}$$

The conditional probability $p(s_k|w_i; R, \psi, b_c)$ is easily obtained from equation 7.9.

7.2.3 Learning

The full learning objective maximizes a sum of the two objectives presented. This produces a final objective function of,

$$\nu||R||_F^2 + \sum_{k=1}^{|D|} \lambda||\hat{\theta}_k||_2^2 + \sum_{i=1}^{N_k} \log p(w_i|\hat{\theta}_k; R, b) + \sum_{k=1}^{|D|} \frac{1}{|S_k|} \sum_{i=1}^{N_k} \log p(s_k|w_i; R, \psi, b_c). \tag{7.11}$$

$|S_k|$ denotes the number of documents in the dataset with the same rounded value of s_k (i.e. $s_k < 0.5$ and $s_k \geqslant 0.5$). We introduce the weighting $\frac{1}{|S_k|}$ to combat the well-known imbalance in ratings present in review collections. This weighting prevents the overall distribution of document ratings from affecting the estimate of document ratings in which a particular word occurs. The hyper-parameters of the model are the regularization weights (λ and ν), and the word vector dimensionality β.

Maximizing the objective function with respect to R, b, ψ, and b_c is a non-convex problem. We use alternating maximization, which first optimizes the word representations (R, b, ψ, and b_c) while leaving the MAP estimates ($\hat{\theta}$) fixed. Then we find the new MAP estimate for each document while leaving the word representations fixed, and continue this process until convergence. The optimization algorithm quickly finds a global solution for each $\hat{\theta}_k$ because we have a low-dimensional, convex problem in each $\hat{\theta}_k$. Because the MAP estimation problems for different documents are independent, we can solve them on separate machines in parallel. This facilitates scaling the model to document collections with hundreds of thousands of documents.

7.3 Experiments

We evaluate our model with document-level and sentence-level categorization tasks in the domain of online movie reviews. For document categorization, we compare our method to previously published results on a standard dataset, and introduce a new dataset for the task. In both tasks we compare our model's word representations with several bag of words weighting methods, and alternative approaches to word vector induction.

7.3.1 Word Representation Learning

We induce word representations with our model using 25,000 movie reviews from IMDB. Because some movies receive substantially more reviews than others, we limited ourselves to including at most 30 reviews from any movie in the collection. We build a fixed dictionary of the 5,000 most frequent tokens, but ignore the 50 most frequent terms from the original full vocabulary. Traditional stop word removal was not used because certain stop words (e.g. negating words) are indicative of sentiment. Stemming was not applied because the model learns similar representations for words of the same stem when the data suggests it. Additionally, because certain non-word tokens (e.g. "!" and ":-)") are indicative of sentiment, we allow them in our vocabulary. Ratings on IMDB are given as star values ($\in \{1, 2, ..., 10\}$), which we linearly map to $[0, 1]$ to use as document labels when training our model.

The semantic component of our model does not require document labels. We train a variant of our model which uses 50,000 unlabeled reviews in addition to the labeled set of 25,000 reviews. The unlabeled set of reviews contains neutral reviews as well as those which are polarized as found in the labeled set. Training the model with additional unlabeled data captures a common scenario where the amount of labeled data is small relative to the amount of unlabeled data available. For all word vector models, we use 50-dimensional vectors.

As a qualitative assessment of word representations, we visualize the words most similar to a query word using vector similarity of the learned representations. Given a query word w and another word w' we obtain their vector representations ϕ_w and $\phi_{w'}$, and evaluate their cosine similarity as $S(\phi_w, \phi_{w'}) = \frac{\phi_w^T \phi_{w'}}{||\phi_w|| \cdot ||\phi_{w'}||}$. By assessing the similarity of w with all other words w', we can find the words deemed most similar by the model.

Table 7.1 shows the most similar words to given query words using our model's word representations as well as those of LSA. All of these vectors capture broad semantic similarities. However, both versions of our model seem to do better than LSA in avoiding accidental distributional similarities (e.g., *screwball* and *grant* as similar to *romantic*) A comparison of the two versions of our model also begins to highlight the importance of adding sentiment information. In general, words indicative of sentiment tend to have high similarity with words of the same sentiment polarity, so even the purely unsupervised model's results look promising. However, they also show more genre and content effects. For example, the sentiment enriched vectors for *ghastly* are truly semantic alternatives to that word, whereas the vectors without sentiment also contain some content words that tend to have *ghastly* predicated of them. Of course, this is only an impressionistic analysis of a few cases, but it is helpful in understanding why the sentiment-enriched model proves superior at the sentiment classification results we report next.

7.3.2 Other Word Representations

For comparison, we implemented several alternative vector space models that are conceptually similar to our own, as discussed in section 7.1:

Latent Semantic Analysis (LSA; Deerwester et al., 1990) We apply truncated SVD to a tf.idf weighted, cosine normalized count matrix, which is a standard weighting and smoothing scheme for VSM induction [141].

Latent Dirichlet Allocation (LDA; Blei et al., 2003) We use the method described in section 7.1 for inducing word representations from the topic matrix. To train the 50-topic LDA model we use code released by [8]. We use the same 5,000 term vocabulary for LDA as is used for training word vector models. We leave the LDA hyperparameters at their default values, though some work suggests optimizing over priors for LDA is important [154].

Weighting Variants We evaluate both binary (b) term frequency weighting with smoothed delta idf (Δt') and no idf (n) because these variants worked well in previous experiments in sentiment [86, 100]. In all cases, we use cosine normalization (c). [97] perform an extensive analysis of such weighting variants for sentiment tasks.

7.3.3 Document Polarity Classification

Our first evaluation task is document-level sentiment polarity classification. A classifier must predict whether a given review is positive or negative given the review text.

Given a document's bag of words vector v, we obtain features from our model using a matrix-vector product Rv, where v can have arbitrary tf.idf weighting. We do not cosine normalize v, instead applying cosine normalization to the final feature vector Rv. This procedure is also used to obtain features from the LDA and LSA word vectors. In preliminary experiments, we found 'bnn' weighting to work best for v when generating document features via the product Rv. In all experiments, we use this weighting to get multi-word representations from word vectors.

Pang and Lee Movie Review Dataset

The polarity dataset version 2.0 introduced by [98] [1] consists of 2,000 movie reviews, where each is associated with a binary sentiment polarity label. We report 10-fold cross validation results using the authors' published folds to make our results comparable with others in the literature. We use a linear support vector machine (SVM) classifier trained with LIBLINEAR [35], and set the SVM regularization parameter to the same value used by [98].

Table 7.2 shows the classification performance of our method, other VSMs we implemented, and previously reported results from the literature. Bag of words vectors are denoted by their weighting notation. Features from word vector learner are denoted by the learner name. As a control, we trained versions of our model with only the unsupervised semantic component, and the full model (semantic and sentiment). We also include results for a version of our full model trained with 50,000 additional unlabeled examples. Finally, to test whether our models' representations complement a standard bag of words, we evaluate performance of the two feature representations concatenated.

Our method's features clearly outperform those of other VSMs, and perform best when combined with the original bag of words representation. The variant of our model trained with additional unlabeled data performed best, suggesting the model can effectively utilize large amounts of unlabeled data along with labeled examples. Our method performs competitively with previously reported results in spite of our restriction to a vocabulary of only 5,000 words.

We extracted the movie title associated with each review and found that 1,299 of the 2,000 reviews in the dataset have at least one other review of the same movie in the dataset. Of 406 movies with multiple reviews, 249 have the same polarity label for all of their reviews. Overall, these facts suggest that, relative to the size of the dataset, there are highly correlated examples with correlated labels. This is a natural and expected property of this kind of document collection, but it can have a substantial impact on performance in datasets of this scale. In the random folds distributed by

[1] http://www.cs.cornell.edu/people/pabo/movie-review-data

the authors, approximately 50% of reviews in each validation fold's test set have a review of the same movie with the same label in the training set. Because the dataset is small, a learner may perform well by memorizing the association between label and words unique to a particular movie (e.g., character names or plot terms).

We introduce a substantially larger dataset, which uses disjoint sets of movies for training and testing. These steps minimize the ability of a learner to rely on idiosyncratic word–class associations, thereby focusing attention on genuine sentiment features.

IMDB Review Dataset

We constructed a collection of 50,000 reviews from IMDB, allowing no more than 30 reviews per movie. The constructed dataset contains an even number of positive and negative reviews, so randomly guessing yields 50% accuracy. Following previous work on polarity classification, we consider only highly polarized reviews. A negative review has a score $\leqslant 4$ out of 10, and a positive review has a score $\geqslant 7$ out of 10. Neutral reviews are not included in the dataset. In the interest of providing a benchmark for future work in this area, we release this dataset to the public.[2]

We evenly divided the dataset into training and test sets. The training set is the same 25,000 labeled reviews used to induce word vectors with our model. We evaluate classifier performance after cross-validating classifier parameters on the training set, again using a linear SVM in all cases. Table 7.2 shows classification performance on our subset of IMDB reviews. Our model showed superior performance to other approaches, and performed best when concatenated with bag of words representation. Again the variant of our model which utilized extra unlabeled data during training performed best.

Differences in accuracy are small, but, because our test set contains 25,000 examples, the variance of the performance estimate is quite low. For example, an accuracy increase of 0.1% corresponds to correctly classifying an additional 25 reviews.

[2]Dataset and further details are available online at: `http://www.andrew-maas.net/data/sentiment`

7.3.4 Subjectivity Detection

As a second evaluation task, we performed sentence-level subjectivity classification. In this task, a classifier is trained to decide whether a given sentence is subjective, expressing the writer's opinions, or objective, expressing purely facts. We used the dataset of [98], which contains subjective sentences from movie review summaries and objective sentences from movie plot summaries. This task is substantially different from the review classification task because it uses sentences as opposed to entire documents and the target concept is subjectivity instead of opinion polarity. We randomly split the 10,000 examples into 10 folds and report 10-fold cross validation accuracy using the SVM training protocol of [98].

Table 7.2 shows classification accuracies from the sentence subjectivity experiment. Our model again provided superior features when compared against other VSMs. Improvement over the bag-of-words baseline is obtained by concatenating the two feature vectors.

7.4 Discussion

We presented a vector space model that learns word representations captuing semantic and sentiment information. The model's probabilistic foundation gives a theoretically justified technique for word vector induction as an alternative to the overwhelming number of matrix factorization-based techniques commonly used. Our model is parametrized as a log-bilinear model following recent success in using similar techniques for language models [6, 20, 89], and it is related to probabilistic latent topic models [8, 133]. We parametrize the topical component of our model in a manner that aims to capture word representations instead of latent topics. In our experiments, our method performed better than LDA, which models latent topics directly.

We extended the unsupervised model to incorporate sentiment information and showed how this extended model can leverage the abundance of sentiment-labeled texts available online to yield word representations that capture both sentiment and semantic relations. We demonstrated the utility of such representations on two tasks of sentiment classification, using existing datasets as well as a larger one that we release for future research. These tasks involve relatively simple sentiment information, but the model is highly flexible in this regard; it can be used to characterize a wide variety of annotations, and thus is broadly applicable in the growing areas of sentiment analysis and retrieval.

	Our model **Sentiment + Semantic**	**Our model** **Semantic only**	**LSA**
	bittersweet	thoughtful	poetic
	heartbreaking	warmth	lyrical
melancholy	happiness	layer	poetry
	tenderness	gentle	profound
	compassionate	loneliness	vivid
	embarrassingly	predators	hideous
	trite	hideous	inept
ghastly	laughably	tube	severely
	atrocious	baffled	grotesque
	appalling	smack	unsuspecting
	lame	passable	uninspired
	laughable	unconvincing	flat
lackluster	unimaginative	amateurish	bland
	uninspired	clichéd	forgettable
	awful	insipid	mediocre
	romance	romance	romance
	love	charming	screwball
romantic	sweet	delightful	grant
	beautiful	sweet	comedies
	relationship	chemistry	comedy

Table 7.1: Similarity of learned word vectors. Each target word is given with its five most similar words using cosine similarity of the vectors determined by each model. The full version of our model (left) captures both lexical similarity as well as similarity of sentiment strength and orientation. Our unsupervised semantic component (center) and LSA (right) capture semantic relations.

Features	PL04	Our Dataset	Subjectivity
Bag of Words (bnc)	85.45	87.80	87.77
Bag of Words (bΔt'c)	85.80	88.23	85.65
LDA	66.70	67.42	66.65
LSA	84.55	83.96	82.82
Our Semantic Only	87.10	87.30	86.65
Our Full	84.65	87.44	86.19
Our Full, Additional Unlabeled	87.05	87.99	87.22
Our Semantic + Bag of Words (bnc)	88.30	88.28	88.58
Our Full + Bag of Words (bnc)	87.85	88.33	88.45
Our Full, Add'l Unlabeled + Bag of Words (bnc)	88.90	88.89	88.13
Bag of Words SVM [98]	87.15	N/A	90.00
Contextual Valence Shifters [61]	86.20	N/A	N/A
tf.Δidf Weighting [86]	88.10	N/A	N/A
Appraisal Taxonomy [158]	90.20	N/A	N/A

Table 7.2: Classification accuracy on three tasks. From left to right the datasets are: A collection of 2,000 movie reviews often used as a benchmark of sentiment classification [98], 50,000 reviews we gathered from IMDB, and the sentence subjectivity dataset also released by [98]. All tasks are balanced two-class problems.

7.5 Multi-Dimensional Sentiment

Computational sentiment analysis is often reduced to a classification task: each text is presumed to have a unique label summarizing its overall sentiment, and the goal is to build models that accurately predict those labels [142, 103, 102]. The most widely-used labels are ‘positive’ and ‘negative’, with a third ‘neutral’ category also commonly included [12]. Sometimes this basic approach is enriched to a ranked or partially-ranked set of categories — for example, star ratings of the sort that are extremely common on the Web [99, 43, 131]. And there is a large body of work employing other categories: not only binary distinctions like subjective vs. objective [11, 160, 48, 116, 117, 101] and pro vs. con [139], but also rich multidimensional category sets modeled on those of cognitive psychology [73, 3, 159, 94].

While treating sentiment as a classification problem is extremely useful for a wide range of tasks, it is just an approximation of the sentiment information that can be conveyed linguistically. The central assumption of the classification approach is that each text is uniquely labeled by one of the categories. However, human reactions are often nuanced, blended, and continuous [120, 32, 164]. Consider, for example, this short ‘confession’ text from the website ExperienceProject.com:

I have a crush on my boss! *blush* eeek *back to work*

At the Experience Project, users can react to texts by clicking buttons summarizing a range of emotions: ‘sorry, hugs’, ‘that rocks’, ‘tee-hee’, ‘I understand’, and ‘wow, just wow’. At the time of this writing, the above confession had received the following distribution of reactions: ‘that rocks’: 1, ‘tee-hee’: 1, ‘I understand’: 10, and ‘wow, just wow’: 0. This corresponds well to the mix of human responses we might expect this text to elicit: it describes a socially awkward and complex situation, which provokes sympathetic reactions, but the text is light-hearted in tone and thus likely to elicit less weighty rections as well. The comments on the confession reflect the summary offered by the reaction distribution: some users tease (“Oooooooooo. . . . i’m telllin!!! lol”) and others offer encouragement (“you go and get that man. . . ”).

In this paper, we develop an approach that allows us to embrace the blended, continuous nature of human sentiment judgments. Our primary data are about 37,000 confessions from the Experience Project with associated reaction distributions. We focus on predicting those reaction distributions given only the confession text. This problem is substantially more challenging than simple classification, but we show that it is tractable and that it presents a worthwhile set of new questions for research in linguistics, natural language processing, and machine learning.

At the heart of our approach is a model that learns vector representations of words. The model has both supervised and unsupervised components. The unsupervised component captures basic semantic information distributionally. However, this document-level distributional information misses important sentiment content. We thus rely on our labeled data to imbue the word vectors with rich emotive information.

Visualization of our model's learned word representations shows multiple levels of word similarity (supplementary diagram A). At the macroscopic level, words are grouped into large clusters based on the reaction distributions they are likely to elicit, relfecting their sentiment connotations. Within these macroscopic clusters, words with highly related descriptive semantic content form sub-structures.

We evaluate our model based upon how well it predicts the reaction distributions of stories, but we also report categorization accuracy as a point of reference. To assess the impact of learning representations specifically for sentiment, we compare our model with several alternative techniques and find it performs significantly better in experiments on the Experience Project data.

Category	Clicks
‘sorry, hugs’	22,236 (19%)
‘you rock’	25,416 (22%)
‘teehee’	16,052 (14%)
‘I understand’	42,352 (37%)
‘wow, just wow’	9,745 (8%)

Table 7.3: Overall distribution of reactions.

7.5.1 Data

As noted above, our data come from the website ExperienceProject.com (EP). The site allows users to upload a variety of different kinds of texts, to comment on others' texts, and to contribute to annotating the texts with information about their reactions. We focus on the ‘confessions’, which are typically short, informal texts relating personal stories, attitudes, and emotions. Here are two typical confessions with their associated reactions:

> I really hate being shy ... I just want to be able to talk to someone about anything and everything and be myself... That's all I've ever wanted. [*understand*: 10; *hugs*: 1; *just wow*: 0; *rock*: 1; *teehee*: 2]

> subconsciously, I constantly narrate my own life in my head. in third person. in a british accent. Insane? Probably [*understand*: 0; *hugs*: 0; *just wow*: 1; *rock*: 7; *teehee*: 8]

Our data consist of 37,146 texts (3,564,039 words; median text length of 56 words). Table 7.3 provides some basic information about the overall distribution of reactions. They are highly skewed towards the category ‘I understand’; the stories are confessional, so it is natural for readers to be sympathetic in response. The ‘wow, just wow’ category is correspondingly little used, in virtue of the fact that it is largely for negative exclamation (its associated emoticon has its mouth and eyes wide open). Such reactions are reserved largely for extremely transgressive or shocking information.

We have restricted attention to the texts with at least one reaction. Table 7.4 summarizes the amount of reaction data present in this document collection, by measuring

Reactions	Texts
$\geqslant 1$	$37,146$
$\geqslant 2$	$24,179$
$\geqslant 3$	$15,813$
$\geqslant 4$	$10,537$
$\geqslant 5$	$7,073$

Table 7.4: Reaction counts.

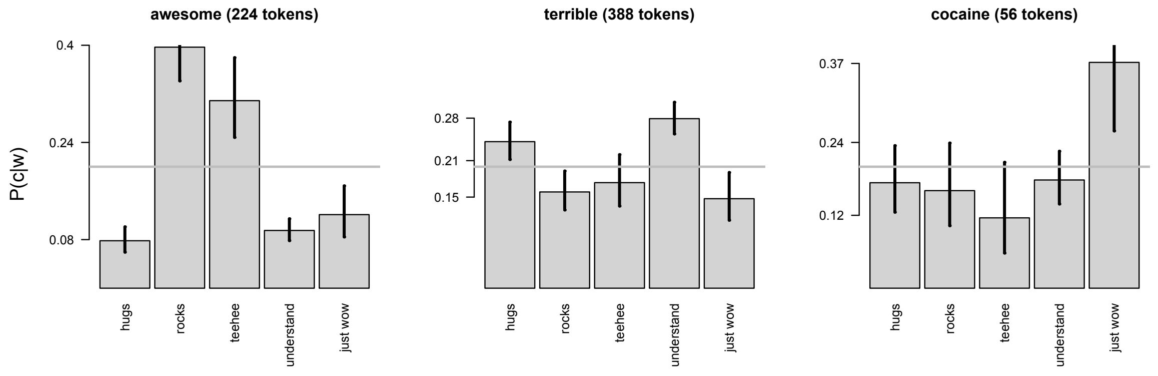


Figure 7.1: Word–category associations in the EP data.

cut-offs at various salient points. When analyzing the reaction data, we normalize the counts such that the distribution over reactions sums to 1. This allows us to treat the reaction data as a probability distribution, ignoring differences in the raw number of counts stories receive.

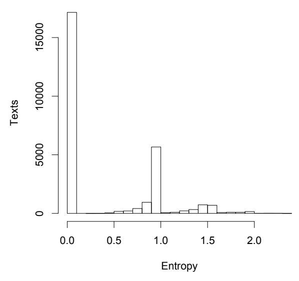
There are many intuitive correlations between the authors' word choices and readers' reaction responses [108]. Figure 7.1 illustrates this effect with words that show strong affinities to particular reaction types. Each panel depicts the distribution of the word across the rating categories. These were derived by first estimating $P(w|c)$, the probability of word w given class c, and then obtaining $P(c|w)$ by an application of Bayes rule under the assumption of a uniform prior over the classes. (Without this uniformity assumption, almost all words appear to associate with the 'understand' category, which is about four times bigger than the others; see table 7.3.) The gray horizontal line is at 0.20, the expected probability if there is no association between the word's usage and the reaction categories.

The first panel in figure 7.1 depicts *awesome*. As one might expect, this correlates most strongly with the 'rocks' and 'teehee' categories; stories in which one uses this word are likely to be perceived as positive and light-hearted (especially as compared to the usual EP fare). Conversely, *terrible*, in the second plot, correlates with 'hugs' and 'understand'; when an author describes something as terrible, readers react with sympathy and solidarity. The final panel depicts *cocaine*, one of a handful of words in the corpus that generate predominantly 'wow, just wow' reactions. We hope these examples help convey the nature of the reaction categories and also suggest that it is promising to try to use these data to learn sentiment-rich word vectors.

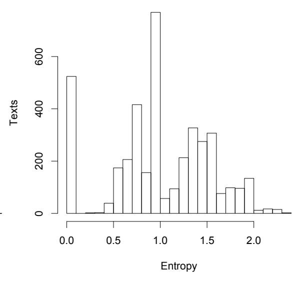
Finally, we address the question of how much the distributions matter as compared with a categorical view of sentiment. If the majority of the texts in the data received categorical or near-categorical responses, we might conclude that classification is an appropriate modeling choice. Conversely, if the texts tend to receive mixed reactions, then we are justified in adopting our more complex approach. Figure 7.2 assesses this using the entropy of the reaction distributions. Where the entropy is zero, just one category was chosen. Where the entropy is around two, the reactions were evenly distributed across the categories. As is evident from this plot, the overall picture is far from categorical; about one-third of the texts have a non-negligible amount of variation in their distributions. What's more, this picture is somewhat misleading. As table 7.4 shows, the majority of our texts have just one reaction. If we restrict attention to the 7,073 texts with at least five reactions, then the entropy values are more evenly distributed, with an entropy of zero far less dominant, as in figure 7.2(b). Thus, these texts manifest the blended, continuous nature of sentiment that we wish to model.

7.5.2 Model

We introduce a model that captures semantic associations among words as well as the blended distributional sentiment information conveyed by words. We assume each word is represented by a real-valued vector and use a probabilistic model to



(a) The full corpus.



(b) Texts with $\geqslant 5$ reactions.

Figure 7.2: The entropy of the reaction distributions.

learn words' vector representations from data. The learning procedure uses the unsupervised information of document-level word co-occurrences as well as the reaction distributions present in EP data.

Our work fits into the broad class of vector space models (VSMs), recently reviewed by [141]. VSMs capture word relationships by encoding words as points in a high-dimensional space. The models are both flexible and powerful; depending on the application, the vector space can encode syntactic information, as is useful for named entity recognition systems [140], or semantic information, as is useful for information retrieval or document classification [84]. Most VSMs apply some sort of matrix factorization technique to a term-context co-occurrence matrix. However, the success of matrix factorization techniques for word vectors often depends heavily on the choices one makes for weighting the entries (for example, with inverse document frequency of words). Thus, the process of building a VSM requires many design choices, often with only past empirical results as guidance. This challenge is multiplied when building representations for sentiment because we want word vectors to capture both descriptive and emotive meanings. The recently introduced delta inverse document frequency weighting technique has had some success in binary sentiment categorization [86], but it does not naturally handle multi-dimensional notions of sentiment.

Our recent work seeks to address these design issues. In [78], we introduce a probabilistic model for learning semantically-sensitive word vectors. In the present paper, we build off of this probabilistic model of documents, because it helps avoid the large design space present in matrix factorization-based VSMs, but we extend its sentiment component considerably. Whereas we previously learned only from unique labels, we are now able to capture the multi-dimensional, non-categorical notion of sentiment that is expressed in the EP data. In the following sections, we introduce the semantic and sentiment components of the model separately, and then describe the procedure for learning the model's parameters from data.

Semantic Component

We approximately capture word semantics from a collection of documents by analyzing document-level word co-occurrences. This semantic component uses a probabilistic model of a document as introduced in our previous work [78]. The model uses a continuous mixture distribution over words indexed by a multi-dimensional random variable θ. Informally, we can think of each dimension of a word vector as a topic in the sense of topic modeling. The document coefficient vector θ thus encodes the strength of each topic for the document. A word's probability in the document then corresponds to how strongly the word's topic strengths match those defined by θ.

We assign a probability to a document d using a joint distribution over the document and θ. The model assumes each word $w_i \in d$ is conditionally independent of the other words given θ, a bag of words assumption often used when learning from document-level co-occurrences. The probability of a document is thus,

$$p(d) = \int p(d, \theta) d\theta = \int p(\theta) \prod_{i=1}^{N} p(w_i|\theta) d\theta, \tag{7.12}$$

where N is the number of words in d and w_i is the i^{th} word in d.

The conditional distribution $p(w_i|\theta)$ is defined using a softmax distribution,

$$p(w|\theta; R, b) = \frac{\exp(\theta^T \phi_w + b_w)}{\sum_{w' \in V} \exp(\theta^T \phi_{w'} + b_{w'})}. \tag{7.13}$$

The parameters of the model are the word representation matrix $R \in \mathbb{R}^{(\beta \text{ x } |V|)}$ where each word w (represented as a one-on vector) in the vocabulary V has a β-dimensional vector representation $\phi_w = Rw$ corresponding to that word's column in R. The random variable θ is also a β-dimensional vector, which weights each of the β dimensions of words' representation vectors. A scalar bias b_w for each word captures differences in overall word frequencies. The probability of a word given the document parameter θ corresponds to how strongly that word's vector representation ϕ_w matches the scaling direction of θ.

Equation 7.12 resembles the probabilistic model of latent Dirichlet allocation (LDA) [8], which models documents as mixtures of latent topics. However, our model does not attempt to model individual topics, but instead directly models word probabilities conditioned on the topic mixture variable θ. Our previous work compares the word vectors learned with our semantic component to an approach which uses LDA topic associations as word vectors. We found the word vectors learned with our model to be superior in tasks of document and sentence-level sentiment classification.

Maximum likelihood learning in this model assumes documents d_k in a collection D are i.i.d. samples. The learning problem of finding parameters to maximize the probability of observed documents becomes,

$$\max_{R,b} p(D; R, b) = \prod_{d_k \in D} \int p(\theta) \prod_{i=1}^{N_k} p(w_i|\theta; R, b) d\theta. \tag{7.14}$$

Using *maximum a posteriori* (MAP) estimates for θ, we approximate this learning problem as,

$$\max_{R,b} \prod_{d_k \in D} p(\hat{\theta}_k) \prod_{i=1}^{N_k} p(w_i|\hat{\theta}_k; R, b), \tag{7.15}$$

where $\hat{\theta}_k$ denotes the MAP estimate of θ for d_k. Our previous work used a Gaussian prior for θ. In our present experiments we explore both Gaussian and Laplacian priors. The Laplacian prior is intuitively appealing because it encourages sparsity, where certain entries of θ are exactly zero as opposed to small non-zero values as is the case when using Gaussian priors. These exactly zero values correspond to topic dimensions which are not at all present in the semantic representation of a word.

Sentiment Component

We now introduce the second component of our model, which aims to capture the multi-dimensional sentiment information expressed by words. Unlike topical information, sentiment is not easy to learn by analyzing document-level word co-occurrences alone. For this reason, we use the reaction distributions of documents to capture how words in the document express multi-dimensional sentiment information. Our previous work demonstrated the value of learning sentiment-sensitive word representations for the simplistic binary categorization notion of sentiment. We now introduce a method to learn word vectors sensitive to a continuous multi-dimensional notion of sentiment.

Our model dictates that a word vector ϕ should predict the reaction distribution of documents in which that word occurs using an appropriate predictor function. Because the reaction distributions are categorical probability distributions, we use a softmax model,

$$\hat{s}_k = \frac{\exp(\psi_k^T \phi + c_k)}{\sum_{k'} \exp(\psi_{k'}^T \phi + c_{k'})} \tag{7.16}$$

The value $\hat{s}_k$ is the probability predicted for the k^{th} sentiment dimension for a given word vector ϕ. The softmax weight vectors ψ_k serve to partition the vector space into K regions where each region corresponds to a particular sentiment dimension. The predicted reaction distribution for a word thus depends on where that word lies in the vector space relative to the regions defined by ψ.

For EP data, a document d is associated with its reaction distribution s, which is a five-dimensional categorical probability distribution ($K = 5$). The softmax parameters $\psi \in \mathbb{R}^{K \times \beta}$ and $c \in \mathbb{R}^K$ are shared across all word vectors as to create a single set of emotive regions in the word vector space. The softmax predicts a reaction distribution for each word, and we learn the softmax parameters as well as the word vectors to match the observed reaction distributions.

The predicted and actual reaction distributions are categorical probability distributions, so we use the Kullback-Leibler (KL) divergence as a measure of how closely the predicted distribution matches the actual. Given the actual distribution s and a prediction for this distribution $\hat{s}$ the KL divergence is,

$$KL(\hat{s}||s) = \sum_{k=1}^{K} s_k \log \frac{s_k}{\hat{s}_k}. \tag{7.17}$$

Learning this component of the model amounts to finding word vectors as well as softmax parameters to minimize the KL divergence between reaction distributions of observed documents and the predicted reaction distributions of words occurring in the documents. We can formally express this as,

$$\min_{R,\psi,c} \sum_{d_k \in D} \sum_{i=1}^{N_k} KL(\hat{s}^{w_i}||s), \tag{7.18}$$

where $\hat{s}^{w_i}$ is the predicted reaction distribution for word w_i as computed by (7.16). To ensure identifiability of the softmax parameters ψ we constrain $\psi_K = 0$

Learning

We now describe the method to learn word vectors using both the semantic and sentiment components of the model. The learning procedure for the semantic component minimizes the negative log of the likelihood shown in equation (7.15). The sentiment component is then additively combined to form the full learning problem,

$$\min_{R,b,\psi,c} \lambda||R||_F^2 + \sum_{d_k \in D} \sum_{i=1}^{N_k} KL(\hat{s}^{w_i}||s) - \left(\sum_{k=1}^{|D|} \log p(\hat{\theta}_k) + \sum_{i=1}^{N_k} \log p(w_i|\hat{\theta}_k; R, b) \right). \tag{7.19}$$

We add to the objective Frobenious norm regularization on the word representation matrix R to prevent the word vector norms from growing too large. We minimize the objective function for several iteration using the L-BFGS quasi-Newton algorithm while leaving the MAP estimates $\hat{\theta}$ fixed. The MAP estimates are then updated while leaving the other parameters of the model fixed. This process continues until the objective function value converges.

Our work explores both a Gaussian and a Laplacian prior for θ. The log-Gaussian prior corresponds to a squared ℓ_2 (sum of squares, $\sum_i x_i^2$) penalty on θ whereas the Laplacian prior corresponds to an ℓ_1 (sum of absolute values,$\sum_i |x_i|$) penalty. Both priors have a single free parameter λ which is proportional to the variance of the prior distribution. This regularization parameter λ and the word vector dimensionality β are the only free hyper-parameters of the model. Because optimizing the non-differentiable ℓ_1 penalty is difficult with gradient-based techniques we approximate the ℓ_1 penalty with the function $\log\cosh(\theta)$.

7.5.3 Experiments

Our experiments focus on predicting the reaction distribution given the text of a document. We employ several baseline approaches to assess the relative performance of our model. As shown in figure 7.2, the reaction distributions of stories which received at least five reactions have higher entropy on average than the set which includes stories with only one reaction or more. The higher entropy reaction distributions are of greater interest because predicting such distributions is substantially more challenging than predicting a low entropy distribution, which is more like the categorization approach of previous work. We evaluate models on both the set of texts with at least one reaction, and the set of texts with five or more reactions.

After collecting the text and reaction distributions from the Web, we tokenized all documents with at least one reaction. Traditional stop word removal was not used because certain stop words (e.g. negations) are indicative of sentiment. To minimize the amount of text pre-processing, we did not apply stemming or spelling correction. Because certain non-word tokens (e.g. "!" and ":-)") are indicative of sentiment, we allow them in our vocabulary. After this tokenization, the dataset consists of 52,973 unique unigrams, many of which occur only once because they are unique spellings of words (e.g. "hahhhaaa"). The collection of 37,146 documents is reduced to 37,130 when we discard documents with no tokens recognized by our tokenizer. Most stories fall around the median length of 56 words, however, a few are thousands of words long. We randomly partitioned the data into 30,000 training and 7,130 test documents. When we consider documents with at least five reactions, this becomes 5,764 training and 1,307 test documents.

Word Representation Learning

We induce word representations with our model using the learning procedure described in section 7.5.2. We construct word representations for only the 5,000 most frequent tokens in the training data. This speeds computation and avoids learning uninformative representations for rare words for which there is insufficient data to properly assess their semantic and sentiment associations. We use the 29,591 documents from our training set with length at least five when the vocabulary is restricted to the 5,000 most frequent tokens. The reaction distributions for documents are used when learning the sentiment component of the model. Our model could leverage additional unlabeled data from related websites to better capture the semantic associations among words. However, we restrict the model to learn from only the labeled training set in order to better compare it to baseline models for this task.

For both the Gaussian and Laplacian models, we evaluate 100-dimensional word vectors and set the regularization parameter $\lambda = 10^{-4}$. Our previous work and preliminary experiments with this dataset suggested the learned word vectors are relatively insensitive to changes in these parameters.

Supplementary diagram A shows a 2-D visualization of the learned word similarities for the 2,000 most frequent words in our vocabulary. The visualization was created using the t-SNE algorithm, with code provided by [144]. Word vectors are cosine normalized before passing them to the t-SNE algorithm.

The visualization clearly shows words grouped locally by semantic associations — for example, "doctor" and "medication" are nearby. Additionally, there is some evidence that the macroscopic structure of the words correlates with how they influence reaction distributions. A cluster of words containing playful, upbeat tokens like ":-)" and "haha" are all likely to appear in stories which elicit the *rock* or *teehee* reactions. Far removed from such happy words are clusters of words indicative of melancholic subjects, marked by words like "cancer" and "suicide." We note that sad and troubling topics are highly prevalent in the data, and our visualization reflects this fact.

After learning the word representations, we represent documents using *average word vectors*. This approach uses the arithmetic average of the word vectors for all words which appear in the document. Because we learn word vectors for only the 5,000 most frequent words, a small fraction of the documents contain only words for which we do not have vector representations. These documents are represented as a vector of all zeros.

Alternative Methods

In addition to the vectors induced using our model, we evaluate the performance of several standard approaches to document categorization and information retrieval.

Unigram Bag of Words Representing a document as a vector of word counts performs surprisingly well in many classification tasks. In our preliminary experiments, we found that term presence performs better than term frequency on EP data, as noted in previous work on sentiment [103]. We also note that delta inverse document frequency weighting, which has been shown to sometimes perform well in sentiment [86], does not extend easily to multi-dimensional notions of sentiment. We thus use term presence vectors with no normalization and evaluate with the full vocabulary of the dataset and the 5,000 word vocabulary used in building word vectors.

Latent Semantic Analysis (LSA) We apply truncated singular value decomposition to a term-document count matrix to obtain word vectors from LSA [26]. We first apply tf.idf weighting to the term-document matrix, but do not use cosine normalization. We use the same 5,000 word vocabulary as is used when constructing word vectors for our model.

High Entropy Reaction Distributions

Our first experiment considers only the examples with at least five reaction clicks, because they best exhibit the blended distributional notion of sentiment of interest

	$\geqslant$ 5 reactions		$\geqslant$ 1 reaction	
Features	KL	Max Acc.	KL	Max Acc.
Uniform Reactions	0.861	20.2	1.275	20.4
Mean Training Reactions	0.763	43.0	1.133	46.7
Bag of Words (All unigrams)	0.637	56.0	1.000	53.4
Bag of Words (Top 5000 unigrams)	0.640	54.9	0.992	54.3
LSA	0.667	51.8	1.032	52.2
Our Method Laplacian Prior	0.621	55.7	0.991	54.7
Our Method Gaussian Prior	0.620	55.2	0.991	54.6

Table 7.5: Test set performance.

in this work. For all of the feature sets described (mean word vectors and bag of words), we train a softmax classifier on the training set. The softmax classifier is a predictor of the same form as is described in equation (7.16), but with a quadratic regularization penalty on the weights. The strength of the regularization penalty is set by cross-validation on the training set. The classifier is trained to minimize the KL divergence of predicted and actual distributions on the training set. We then evaluate the models by measuring average KL divergence on the test set.

We also report performance of models in terms of accuracy in predicting the maximum probability reaction for a document. In this setting, the model picks a single category corresponding to its most probable predicted reaction. A prediction is counted as correct if that category is the most probable in the true reaction distribution, or if it is tied with other categories for the role of most probable. None of the models were explicitly optimized to perform this task, but instead to predict the full distribution of reactions. However, it is helpful to compare this performance metric to KL divergence, as measuring performance in terms of accuracy is more familiar. Table 7.5 shows the results; recall that lower average KL divergence indicates better performance.

All bag of words and vector space models beat the simplistic baselines of predicting the average reaction distribution, or a uniform distribution. The improvements in both KL divergence and accuracy are substantial relative to these simplistic baselines, suggesting that it is indeed feasible to predict reaction distributions from text. Both variants of our model perform better than bag of words and LSA in KL divergence, but bag of words performs best using the accuracy as the metric. That the accuracy and KL metrics disagree on models' performance rankings suggests categorization accuracy is not a sufficient indicator of how well models capture a distributional notion of sentiment. Based on the poor performance of LSA-derived word vectors, we hypothesize that learning representations using sentiment distributions is critical when attempting to capture the blended sentiment information within documents.

Differences in KL divergence are somewhat difficult to interpret, so we use a matched t-test to evaluate their significance. The matched t-test between two models takes the KL divergence for each test example and evaluates the hypothesis that the KL divergence numbers come from the same distribution. KL divergences on the set of test examples are approximately gamma distributed with a valid range of $[0, \infty]$. We thus apply the matched t-test to the logarithm of the KL divergences, which have a Gaussian distribution as assumed by the t-test. We find that the difference in KL divergence between our models and the bag of words models are significant ($p < 0.001$). However, the Gaussian and Laplacian prior variants of our model do not differ significantly from each other. The prior over document coefficients perhaps has little effect relative to the other components of our model, causing both model variants to perform comparably.

All Reaction Distributions

We repeated the experimental procedure using the full dataset which includes all documents with at least one reaction. As noted in figure 7.2, these reaction distributions have low average entropy because a large number of documents have only a few reactions. Distribution predictors for all models were trained and evaluated on this dataset; table 7.5 shows the results.

Again all models outperform the naive baselines of guessing the average training distribution or a uniform distribution. A third baseline (not shown) which assigns 99% of its probability mass to the dominant *understand* category performs substantially worse than all results shown. Although the difference in KL divergence between our models and the bag of words baselines are numerically small, the improvement of our models is significant as measured by the matched t-test ($p < 0.001$). The significance of such small differences is due to the large testing set size. Again the Gaussian and Laplacian variants of our model do not differ significantly from each other in performance.

We see that all models have a higher average KL divergence on this task as compared to evaluation on the set of documents with at least five reactions. As shown in table 7.4, reaction distributions with zero entropy dominate this version of the dataset. We hypothesize that the higher average KL divergences and small numerical differences in KL divergence are largely due to all predictors struggling to fit these zero entropy distributions which were formed with only one reaction click.

7.5.4 Conclusion

Using the confessions at the EP, we showed that natural language texts often convey a wide range of sentiment information to varying degrees. While classification models can capture certain emotive dimensions, they miss this blended, continuous nature of sentiment expression. Building on the existing classifier model of [78], we developed a vector-space model that learns from distributions over emotive categories, in addition to capturing basic semantic information in an unsupervised fashion. The model is successful in absolute terms, suggesting that learning realistic sentiment distributions is tractable, and it also outperforms various baselines, including LSA. We believe the task of predicting sentiment distributions from text provides a rich challenge for the field of sentiment analysis, especially when compared to simpler classification tasks.

Chapter 8

Conclusion

Spoken language understanding demands sophisticated signal processing, transciption, and higher level reasoning systems to achieve the performance we imagine for conversational spoken language interfaces. In this work we analyzed several components of an SLU pipeline. At each stage we found complex individual systems with hand-engineered assumptions about tasks including noise reduction, speech transcription, and sentiment analysis. For each of these tasks, we were able to take a machine learning approach based on deep neural networks to improve performance while removing perhaps incorrect modeling assumptions.

For robust speech recognition we demonstrated a neural network approach to denoising acoustic features. Rather than designing a noise reduction approach for specific environments, our approach learns from available data. The neural network creates a customized noise reduction function for whatever types of noise and distortion are present in the training data. This approach, combined with a standard HMM-GMM speech transcription system, achieves strong results without a need to hand-define noise reduction functions. This demonstrated the capabilities of a neural network architecture to learn achieve a complex task on acoustic data by replacing function design with a machine learning approach.

Word vectors offer a possible bridge between a span of acoustic features and the word identities necessary for language understanding. We introduced a convolutional network architecture that projects the acoustics of an entire word to a word vector space. Using this architecture we were able to improve a speech transcription system by re-ranking an existing n-best list of candidate transcriptions. This demonstrated the potential for word-level deep learning approaches in the speech domain. However, second-pass speech transcription systems are less desirable when compared with first-pass systems due to the need for increased computation and latency when re-ranking a set of hypotheses.

For first-pass speech transcription, previous work had already found that replacing GMM acoustic models with deep neural networks leads to significant improvements in overall system performance. Our work sought to understand which aspects of neural network architecture are most important for speech recognition system performance. This provided an interesting platform to analyze how modifying and improving the neural network component interacts with a larger system. Rectified linear units provided a substantial improvement for DNN acoustic models. However, we found that large improvements in performance of a neural network acoustic model as a classifier may not yield large word error rate improvements for the overall system. This phenomenon is in stark contrast to computer vision systems where the neural network component comprises most of a system, and gains by the neural network typically directly translate to overall system gains. We studied the interaction between neural network acoustic model performance as compared with overall system performance and found that the closest match between the two occurs only with large training corpora. Indeed, we built and experimented on a speech recognition system using a training corpus about five times larger than the standard Switchboard corpus used for academic research. Neural network acoustic models interact with overall system performance in a qualitatively different way at this scale, suggesting that future work with neural network acoustic models must use larger training corpora to remain relevant.

We developed a novel approach to speech recognition which relies on only a neural network acoustic model and an neural network character language model. This system yields performance on par with a baseline HMM-GMM but requires an order of magnitude less software infrastructure. Further, because we train our model using the previously developed connectionist temporal classification loss function, our work lays a foundation for reasoning about language understanding errors while training a speech transcription system. This has important implications for spoken language understanding as it lays a foundation for building SLU systems that are trained as a single, end-to-end solution. This allows all aspects of a system to adapt to a particular task rather than being built and tuned in isolation.

At the language understanding level, we developed an approach to learn word representations from a mixture of unsupervised text and task-specific labels. Word representations are an important component when using neural networks to reason about language, and task-specific word representations can increase performance substantially. We demonstrated our approach on sentiment analysis tasks. The induced word representations accurately capture both semantic and emotive information about words. These representations achieve strong performance in sentiment classification tasks, and can be used in more sophisticated sentiment analysis systems.

This work shows that for tasks at each stage in the spoken language understanding pipeline we can apply machine learning methods based on neural networks to reduce task-specific assumptions while improving overall performance. This is possible because neural networks can learn increasingly complex functions when sufficient training data is available. Speech recognition provided an excellent test bed to explore the promise of systems for complex tasks with large amounts of training data. Mobile and ubiquitous computing is driving rapid adoption of speech interfaces, which in turn generates more available data to train systems for complex spoken language understanding tasks. By leveraging this data, and designing SLU systems with significant neural network components, we can make progress towards natural, conversational speech interfaces.

Bibliography

[1] Ossama Abdel-Hamid, Abdel rahman Mohamed, Hui Jang, and Gerald Penn. Applying convolutional neural networks concepts to hybrid nn-hmm model for speech recognition. In *ICASSP*, 2012.

[2] C. O. Alm, D. Roth, and R. Sproat. Emotions from text: machine learning for text-based emotion prediction. In *Proceedings of HLT/EMNLP*, pages 579–586, 2005.

[3] Cecilia Ovesdotter Alm, Dan Roth, and Richard Sproat. Emotions from text: Machine learning for text-based emotion prediction. In *Proceedings of the Human Language Technology Conference and the Conference on Empirical Methods in Natural Language Processing (HLT/EMNLP)*, 2005.

[4] A. Andreevskaia and S. Bergler. Mining WordNet for fuzzy sentiment: sentiment tag extraction from WordNet glosses. In *Proceedings of the European ACL*, pages 209–216, 2006.

[5] 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.

[6] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin. a neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155, August 2003.

[7] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2), 1994.

[8] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022, May 2003.

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

[10] J. Boyd-Graber and P. Resnik. Holistic sentiment analysis across languages: multilingual supervised latent Dirichlet allocation. In *Proceedings of EMNLP*, pages 45–55, 2010.

[11] Rebecca F. Bruce and Janyce M. Wiebe. Recognizing subjectivity: A case study in manual tagging. *Natural Language Engineering*, 5(2), 1999.

[12] Luís Cabral and Ali Hortaçsu. The dynamics of seller reputation: Theory and evidence from eBay. Working paper, downloaded version revised in March, 2006.

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

[14] 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*, pages 571–582, 2014.

[15] S. Chopra, P. Haffner, and D. Dimitriadis. Combining Frame and Segment Level Processing via Temporal Pooling for Phonetic Classification. In *12th Annual Conference of the International Speech Communication Association*, 2011.

[16] 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*, pages 745–753. IEEE, 2014.

[17] Christopher Cieri, David Miller, and Kevin Walker. The fisher corpus: a resource for the next generations of speech-to-text. In *LREC*, volume 4, pages 69–71, 2004.

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

[19] A.P. Coates and A.Y. Ng. The Importance of Encoding Versus Training with Sparse Coding and Vector Quantization. In *ICML*, 2011.

[20] R. Collobert and J. Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *ICML*, pages 160–167, 2008.

[21] G.E. Dahl, T.N. Sainath, and G.E. Hinton. Improving Deep Neural Networks for LVCSR using Rectified Linear Units and Dropout. In *ICASSP*, 2013.

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

[23] G.E. 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.

[24] R. De Mori, F. Bechet, D. Hakkani-Tur, M. McTear, G. Riccardi, and G. Tur. Spoken language understanding. *Signal Processing Magazine, IEEE*, 25(3):50–58, 2008.

[25] 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.

[26] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41:391–407, September 1990.

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

[28] Li Deng, Alex Acero, Li Jiang, Jasha Droppo, and Xuedong Huang. High-performance robust speech recognition using stereo training data. In *ICASSP*, 2001.

[29] TG Dietterich and G Bakiri. Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, 2(19):263–286, 1995.

[30] J. Droppo and A. Acero. *Handbook of Speech Processing*, chapter Environmental Robustness. Springer, 2008.

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

[32] Paul Ekman. An argument for basic emotions. *Cognition and Emotion,*, 6(3/4):169–200, 1992.

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

[34] ETSI. Advanced front-end feature extraction algorithm. *Technical Report. ETSI ES 202 050*, 2007.

[35] R. E. Fan, K. W. Chang, C. J. Hsieh, X. R. Wang, and C. J. Lin. LIBLINEAR: A library for large linear classification. *The Journal of Machine Learning Research*, 9:1871–1874, August 2008.

[36] J. R. Finkel and C. D. Manning. Joint parsing and named entity recognition. In *Proceedings of NAACL*, pages 326–334, 2009.

[37] M. Gales and S. Young. The application of hidden markov models in speech recognition. *Foundations and Trends in Signal Processing*, 1(3):195–304, 2008.

[38] Mark Gales and Steve Young. The Application of Hidden Markov Models in Speech Recognition. *Foundations and Trends in Signal Processing*, 1(3):195–304, 2007.

[39] M.J.F. Gales. Maximum likelihood linear transformations for HMM-based speech recognition. *Computer speech and language*, 12:75–98, 1998.

[40] X. Glorot, A. Bordes, and Y Bengio. Deep Sparse Rectifier Networks. In *AISTATS*, pages 315–323, 2011.

[41] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, volume 9, pages 249–256, 2010.

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

[43] A. B. Goldberg and J. Zhu. Seeing stars when there aren't many stars: graph-based semi-supervised learning for sentiment categorization. In *TextGraphs: HLT/NAACL Workshop on Graph-based Algorithms for Natural Language Processing*, pages 45–52, 2006.

[44] S. Goldwater, D. Jurafsky, and C. Manning. Which Words are Hard to Recognize? Prosodic, Lexical, and Disfluency Factors That Increase Speech Recognition Error Rates. *Speech Communications*, 52:181–200, 2010.

[45] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber. Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks. In *ICML*, pages 369–376. ACM, 2006.

[46] A. Graves and N. Jaitly. Towards End-to-End Speech Recognition with Recurrent Neural Networks. In *ICML*, 2014.

[47] A. Graves, A. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In *ICASSP*, pages 6645–6649. IEEE, 2013.

[48] Vasileios Hatzivassiloglou and Janyce Wiebe. Effects of adjective orientation and gradability on sentence subjectivity. In *Proceedings of the International Conference on Computational Linguistics (COLING)*, 2000.

[49] K. Heafield, I. Pouzyrevsky, J. H. Clark, and P. Koehn. Scalable modified Kneser-Ney language model estimation. In *ACL-HLT*, pages 690–696, Sofia, Bulgaria, 2013.

[50] H. Hermansky, D. Ellis, and S. Sharma. Tandem connectionist feature extraction for conventional hmm systems. In *ICASSP*, volume 3, pages 1635–1638. IEEE, 2000.

[51] G. Hinton, S. Osindero, and Y. W. Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

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

[53] X. Huang, A. Acero, H.-W. Hon, et al. *Spoken language processing*, volume 18. Prentice Hall Englewood Cliffs, 2001.

[54] 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.

[55] A Hyvarinen, J Hurri, and P.O. Hoyer. *Natural Image Statistics – A probabilistic approach to early computational vision*. Springer-Verfag, 2009.

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

[57] T. Jay. *Why We Curse: A Neuro-Psycho-Social Theory of Speech.* John Benjamins, Philadelphia/Amsterdam, 2000.

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

[59] 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.

[60] D. Kaplan. What is meaning? Explorations in the theory of *Meaning as Use.* Brief version — draft 1. Ms., UCLA, 1999.

[61] A. Kennedy and D. Inkpen. Sentiment classification of movie reviews using contextual valence shifters. *Computational Intelligence*, 22:110–125, May 2006.

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

[63] A. Krizhevsky, I. Sutskever, and G.E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *NIPS*, 2012.

[64] Q. Le, W. Zou, S. Yeung, and A. Ng. Learning hierarchical spatio-temporal features for action recognition with independent subspace analysis. In *CVPR*, 2011.

[65] Quoc V Le, Adam Coates, Bobby Prochnow, and Andrew Y Ng. On Optimization Methods for Deep Learning. In *ICML*, 2011.

[66] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based Learning Applied to Document Recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

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

[68] H. Lee, Y. Largman, P. Pham, and A.Y. Ng. Unsupervised feature learning for audio classification using convolutional deep belief networks. In *NIPS*, volume 22, pages 1096–1104, 2010.

[69] F. Li, M. Huang, and X. Zhu. Sentiment analysis with global topics and local dependency. In *Proceedings of AAAI*, pages 1371–1376, 2010.

[70] 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.

[71] C. Lin and Y. He. Joint sentiment/topic model for sentiment analysis. In *Proceeding of the 18th ACM Conference on Information and Knowledge Management*, pages 375–384, 2009.

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

[73] Hugo Liu, Henry Lieberman, and Ted Selker. A model of textual affect sensing using real-world knowledge. In *Proceedings of Intelligent User Interfaces (IUI)*, pages 125–132, 2003.

[74] Z. Luo, H. Liu, and X. Wu. Artificial neural network computation on graphic process unit. In *IJCNN*, pages 622–626. IEEE, 2005.

[75] 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.

[76] A. Maas, Q. V. Le, T. M. O'Neil, O. Vinyals, P. Nguyen, and A. Y. Ng. Recurrent Neural Networks for Noise Reduction in Robust ASR. In *Interspeech*, 2012.

[77] A. Maas, T. M. O'Neil, A. Hannun, and A. Y. Ng. Recurrent Neural Network Feature Enhancement: The 2nd CHiME Challenge. In *The 2nd International Workshop on Machine Listening in Multisource Environments (CHiME)*, 2013.

[78] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics.

[79] A. L. Maas, A. Y. Hannun, and A. Y. Ng. Rectifier nonlinearities improve neural network acoustic models. *Proc. ICML*, 30, 2013.

[80] A. L. Maas, S. D. Miller, T. M. Oneil, A. Y. Ng, and P. Nguyen. Word-level acoustic modeling with convolutional vector regression. *ICML Workshop on Representation Learning*, 2012.

[81] A. L. Maas, A. Y. Ng, and C. Potts. Multi-dimensional sentiment analysis with learned representations, 2011.

[82] A. L. Maas, P. Qi, Z. Xie, A. Y. Hannun, C. T. Lengerich, D. Jurafsky, and A. Y. Ng. Building dnn acoustic models for large vocabulary speech recognition. *In Submission*, 2015.

[83] A. L. Maas*, Z. Xie*, D. Jurafsky, and A. Y. Ng. Lexiconfree conversational speech recognition with neural networks. In *Human Language Technologies: The 2015 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, 2015.

[84] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 1 edition, 2008.

[85] J. Martens. Deep learning via hessian-free optimization. In *ICML*, pages 735–742, 2010.

[86] J. Martineau and T. Finin. Delta tfidf: an improved feature space for sentiment analysis. In *Proceedings of the 3rd AAAI International Conference on Weblogs and Social Media*, pages 258–261, 2009.

[87] James L McClelland and Jeffrey L Elman. The trace model of speech perception. *Cognitive psychology*, 18(1):1–86, 1986.

[88] T. Mikolov, M. Karafiát, L. Burget, J. Cernockỳ, and S. Khudanpur. Recurrent neural network based language model. In *INTERSPEECH*, pages 1045–1048, 2010.

[89] A. Mnih and G. E. Hinton. Three new graphical models for statistical language modelling. In *Proceedings of the ICML*, pages 641–648, 2007.

[90] A. Mohamed, G. Dahl, and G. Hinton. Acoustic modeling using deep belief networks. *Audio, Speech, and Language Processing, IEEE Transactions on*, (99), 2010.

[91] A. Mohamed, G.E. Dahl, and G.E. Hinton. Acoustic Modeling using Deep Belief Networks. *IEEE Transactions on Audio, Speech, and Language Processing*, (99), 2011.

[92] N. Morgan. Deep and wide: Multiple layers in automatic speech recognition. *IEEE Transactions on Audio, Speech, and Language Processing*, 20(1):7–13, 2012.

[93] Y. Nesterov. A method of solving a convex programming problem with convergence rate o (1/k2). In *Soviet Mathematics Doklady*, volume 27, pages 372–376, 1983.

[94] Alena Neviarouskaya, Helmut Prendinger, and Mitsuru Ishizuka. Recognition of affect, judgment, and appreciation in text. In *Proceedings of the 23rd International Conference on Computational Linguistics (COLING 2010)*, pages 806–814, Beijing, China, August 2010. COLING 2010 Organizing Committee.

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

[96] K. Oh and K. Jung. Gpu implementation of neural networks. *Pattern Recognition*, 37(6):1311–1314, 2004.

[97] G. Paltoglou and M. Thelwall. A study of information retrieval weighting schemes for sentiment analysis. In *Proceedings of the ACL*, pages 1386–1395, 2010.

[98] B. Pang and L. Lee. A sentimental education: sentiment analysis using subjectivity summarization based on minimum cuts. In *Proceedings of the ACL*, pages 271–278, 2004.

[99] B. Pang and L. Lee. Seeing stars: exploiting class relationships for sentiment categorization with respect to rating scales. In *Proceedings of ACL*, pages 115–124, 2005.

[100] B. Pang, L. Lee, and S. Vaithyanathan. Thumbs up? sentiment classification using machine learning techniques. In *Proceedings of EMNLP*, pages 79–86, 2002.

[101] Bo Pang and Lillian Lee. A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts. In *Proceedings of the Association for Computational Linguistics (ACL)*, pages 271–278, 2004.

[102] Bo Pang and Lillian Lee. Opinion mining and sentiment analysis. *Foundations and Trends in Information Retrieval*, 2(1):1–135, 2008.

[103] Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. Thumbs up? sentiment classification using machine learning techniques. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 79–86, Philadelphia, July 2002. Association for Computational Linguistics.

[104] S Parveen and P Green. Speech recognition with missing data using recurrent neural nets. In *NIPS*, 2001.

[105] D. Pearce and H.G. Hirsch. The Aurora experimental framework for the performance evaluation of speech recognition systems under noisy conditions. In *ICSLP*, 2000.

[106] D. Plaut. Experiments on learning by back propagation. 1986.

[107] C. Potts. The expressive dimension. *Theoretical Linguistics*, 33:165–197, 2007.

[108] Christopher Potts. On the negativity of negation. In David Lutz and Nan Li, editors, *Proceedings of Semantics and Linguistic Theory 20*. CLC Publications, Ithaca, NY, 2010.

[109] 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.

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

[111] D Povey and P C Woodland. Minimum Phone Error and I-Smoothing for Improved Discriminative Training. In *ICASSP*, 2002.

[112] X. Qian and Y. Liu. Disfluency detection using multi-step stacked learning. In *HLT-NAACL*, pages 820–825, 2013.

[113] A Ragni and M.J.F. Gales. Derivative Kernels for Noise Robust ASR. In *ASRU*, 2011.

[114] R. Raina, A. Madhavan, and A. Y. Ng. Large-scale deep unsupervised learning using graphics processors. In *ICML*, volume 9, pages 873–880, 2009.

[115] 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*, 2(1):161–174, 1994.

[116] Ellen Riloff and Janyce Wiebe. Learning extraction patterns for subjective expressions. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2003.

[117] Ellen Riloff, Janyce Wiebe, and William Phillips. Exploiting subjectivity classification to improve information extraction. In *Proceedings of AAAI*, pages 1106–1111, 2005.

[118] Tony Robinson and Frank Fallside. A recurrent error propagation network speech recognition system. *Computer Speech & Language*, 5(3):259–274, 1991.

[119] 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*, 76:1555, 1986.

[120] James A. Russell. A circumplex model of affect. *Journal of Personality and Social Psychology*, 39(6):1161–1178, 1980.

[121] 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.

[122] 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.

[123] T. N. Sainath, I. Chung, B. Ramabhadran, M. Picheny, J. Gunnels, B. Kingsbury, G. Saon, V. Austel, and U. Chaudhari. Parallel deep neural network training for lvcsr tasks using blue gene/q. In *INTERSPEECH*, 2014.

[124] Tara N. Sainath, Brian Kingsbury, George Saon, Hagen Soltau, Abdel rahman Mohamed, George Dahl, and Bhuvana Ramabhadran. Deep Convolutional Neural Networks for Large-Scale Speech Tasks. *Neural Networks*, 2014.

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

[126] 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.

[127] G. Saon and J. Chien. Large-vocabulary continuous speech recognition systems: A look at some recent advances. *IEEE Signal Processing Magazine*, 29(6):18–33, 2012.

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

[129] F. Seide, G. Li, and D. Yu. Conversational speech transcription using context-dependent deep neural networks. In *Interspeech*, pages 437–440, 2011.

[130] A. Senior, G. Heigold, M. Bacchiani, and H. Liao. Gmm-free dnn acoustic model training. In *ICASSP*, pages 5602–5606. IEEE, 2014.

[131] B. Snyder and R. Barzilay. Multiple aspect ranking using the good grief algorithm. In *Proceedings of NAACL*, pages 300–307, 2007.

[132] Hagen Soltau, George Saon, and Brian Kingsbury. The IBM Attila speech recognition toolkit. In *IEEE Spoken Language Technology Workshop (SLT)*, pages 97–102. IEEE, 2010.

[133] M. Steyvers and T. L. Griffiths. Probabilistic topic models. In T. Landauer, D McNamara, S. Dennis, and W. Kintsch, editors, *Latent Semantic Analysis: A Road to Meaning*, 2006.

[134] 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*, pages 6664–6668, 2013.

[135] I. Sutskever. *Training recurrent neural networks.* PhD thesis, University of Toronto, 2013.

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

[137] I. Sutskever, J. Martens, and G. E. Hinton. Generating text with recurrent neural networks. In *ICML*, pages 1017–1024, 2011.

[138] S Tamura and A Waibel. Noise reduction using connectionist models. In *ICASSP*, pages 553–556, 1988.

[139] Matt Thomas, Bo Pang, and Lillian Lee. Get out the vote: Determining support or opposition from Congressional floor-debate transcripts. In *Proceedings of EMNLP*, pages 327–335, 2006.

[140] J. Turian, L. Ratinov, and Y. Bengio. Word representations: A simple and general method for semi-supervised learning. In *Proceedings of the ACL*, page 384394, 2010.

[141] P. D. Turney and P. Pantel. From frequency to meaning: vector space models of semantics. *Journal of Artificial Intelligence Research*, 37:141–188, 2010.

[142] Peter Turney. Thumbs up or thumbs down? Semantic orientation applied to unsupervised classification of reviews. In *Proceedings of the Association for Computational Linguistics (ACL)*, pages 417–424, 2002.

[143] V Valtchev, JJ Odell, Philip C Woodland, and Steve J Young. Mmie training of large vocabulary recognition systems. *Speech Communication*, 22(4):303–314, 1997.

[144] Laurens van der Maaten and Geoffrey Hinton. Visualizing Data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, November 2008.

[145] K. Vesely, A. Ghoshal, L. Burget, and D. Povey. Sequence-discriminative training of deep neural networks. In *Interspeech*, 2013.

[146] K. Veselỳ, A. Ghoshal, L. Burget, and D. Povey. Sequence-discriminative training of deep neural networks. In *INTERSPEECH*, pages 2345–2349, 2013.

[147] E. Vincent, J. Barker, S. Watanabe, J. Le Roux, F. Nesta, and M. Matassoni. The Second 'CHiME' Speech Separation and Recognition Challenge: Datasets, Tasks and Baselines. In *ICASSP*, 2013.

[148] 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*, 11:3371–3408, 2010.

[149] Pascal Vincent, Hugo Larochelle, Y. Bengio, and P.A. Manzagol. Extracting and composing robust features with denoising autoencoders. In *ICML*, pages 1096–1103. ACM, 2008.

[150] O. Vinyals, S. V. Ravuri, and D. Povey. Revisiting recurrent neural networks for robust asr. In *ICASSP*, pages 4085–4088. IEEE, 2012.

[151] Oriol Vinyals and Suman Ravuri. Comparing multilayer perceptron to Deep Belief Network Tandem features for robust ASR. In *ICASSP*, pages 2–5, 2011.

[152] S. Wager, S. Wang, and P. Liang. Dropout Training as Adaptive Regularization. In *NIPS*, 2013.

[153] 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*, 37(3):328–339, 1989.

[154] H. Wallach, D. Mimno, and A. McCallum. Rethinking LDA: why priors matter. In *Proceedings of NIPS*, pages 1973–1981, 2009.

[155] Jinjun Wang, Jianchao Yang, Kai Yu, Fengjun Lv, Thomas Huang, and Yihong Gong. Locality-constrained linear coding for image classification. In *CVPR*, 2010.

[156] S. Wegmann, D. McAllaster, J. Orloff, and B. Peskin. Speaker normalization on conversational telephone speech. In *ICASSP*, pages 339–341, 1996.

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

[158] C. Whitelaw, N. Garg, and S. Argamon. Using appraisal groups for sentiment analysis. In *Proceedings of CIKM*, pages 625–631, 2005.

[159] Janyce Wiebe, Theresa Wilson, and Claire Cardie. Annotating expressions of opinions and emotions in language. *Language Resources and Evaluation (formerly Computers and the Humanities)*, 39(2/3):164–210, 2005.

[160] Janyce M. Wiebe, Rebecca F. Bruce, and Thomas P. O'Hara. Development and use of a gold standard data set for subjectivity classifications. In *Proceedings of the Association for Computational Linguistics (ACL)*, pages 246–253, 1999.

[161] B. Willmore and D.J. Tolhurst. Characterizing the sparseness of neural codes. *Network: Computation in Neural Systems*, 12(3):255–270, 2001.

[162] B. Willmore, P.A. Watters, and D.J. Tolhurst. A comparison of natural-image-based models of simple-cell coding. *Perception*, 29(9):1017–1040, 2000.

[163] T. Wilson, J. Wiebe, and R. Hwa. Just how mad are you? Finding strong and weak opinion clauses. In *Proceedings of AAAI*, pages 761–769, 2004.

[164] Theresa Wilson, Janyce Wiebe, and Rebecca Hwa. Just how mad are you? Finding strong and weak opinion clauses. *Computational Intelligence*, 2(22):73–99, 2006.

[165] 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*, volume 2. Entropic Cambridge Research Laboratory Cambridge, 1997.

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

[167] Dong Yu and Li Deng. Deep neural network-hidden markov model hybrid systems. In *Automatic Speech Recognition*, pages 99–116. Springer, 2015.

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

[169] G. Zweig, P. Nguyen, D. Van Compernolle, K. Demuynck, L. Atlas, P. Clark, G. Sell, M. Wang, F. Sha, H. Hermansky, D. Karakos, A. Jansen, S. Thomas, G.S.V.S. Sivaram, S. Bowman, and J. Kao. Speech Recognition with Segmental Conditional Random Fields: A Summary of the JHU CLSP 2010 Summer Workshop. In *ICASSP*, 2011.

[170] Geoffrey Zweig and Patrick Nguyen. SCARF: A segmental conditional random field toolkit for speech recognition. In *Interspeech*, 2010.