Neural Networks 64 (2015) 39–48

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

2015 Special Issue

Deep Convolutional Neural Networks for Large-scale Speech Tasks

Tara N. Sainath [a,*], Brian Kingsbury [a], George Saon [a], Hagen Soltau [a], Abdel-rahman Mohamed [b], George Dahl [b], Bhuvana Ramabhadran [a]

[a] IBM T. J. Watson Research Center, Yorktown Heights, NY 10598, United States

[b] Department of Computer Science, University of Toronto, United States

ARTICLE INFO

Article history:
Available online 16 September 2014

Keywords:
Deep learning
Neural networks
Speech recognition

ABSTRACT

Convolutional Neural Networks (CNNs) are an alternative type of neural network that can be used to reduce spectral variations and model spectral correlations which exist in signals. Since speech signals exhibit both of these properties, we hypothesize that CNNs are a more effective model for speech compared to Deep Neural Networks (DNNs). In this paper, we explore applying CNNs to large vocabulary continuous speech recognition (LVCSR) tasks. First, we determine the appropriate architecture to make CNNs effective compared to DNNs for LVCSR tasks. Specifically, we focus on how many convolutional layers are needed, what is an appropriate number of hidden units, what is the best pooling strategy. Second, investigate how to incorporate speaker-adapted features, which cannot directly be modeled by CNNs as they do not obey locality in frequency, into the CNN framework. Third, given the importance of sequence training for speech tasks, we introduce a strategy to use ReLU+dropout during Hessian-free sequence training of CNNs. Experiments on 3 LVCSR tasks indicate that a CNN with the proposed speaker-adapted and ReLU+dropout ideas allow for a 12%–14% *relative improvement* in WER over a strong DNN system, achieving state-of-the art results in these 3 tasks.

© 2014 Elsevier Ltd. All rights reserved.

1. Introduction

Recently, Deep Neural Networks (DNNs) have achieved tremendous success in acoustic modeling for large vocabulary continuous speech recognition (LVCSR) tasks, showing significant gains over state-of-the-art Gaussian Mixture Model/Hidden Markov Model (GMM/HMM) systems on a wide variety of small and large vocabulary tasks (Dahl, Yu, Deng, & Acero, 2012; Hinton, Deng, Yu, Dahl, Mohamed, Jaitly, Senior, Vanhoucke, Nguyen, Sainath, & Kingsbury, 2012; Jaitly, Nguyen, Senior, & Vanhoucke, 2012; Kingsbury, Sainath, & Soltau, 2012; Seide, Li, & Yu, 2011). Convolutional Neural Networks (CNNs) (LeCun & Bengio, 1995; Lecun, Bottou, Bengio, & Haffner, 1998) are an alternative type of neural network that can be used to model spatial and temporal correlation, while reducing translational variance in signals.

CNNs are attractive compared to fully-connected DNNs for a variety of reasons. First, DNNs ignore input topology, as the input can be presented in any (fixed) order without affecting the performance of the network (LeCun & Bengio, 1995). However, spectral representations of speech have strong correlations in time and frequency, and modeling local correlations with CNNs, through weights which are shared across local regions of the input space, has been shown to be beneficial in other fields (LeCun, Huang, & Bottou, 2004). Second, DNNs are not explicitly designed to model translational variance within speech signals, which can exist due to different speaking styles (LeCun & Bengio, 1995). More specifically, different speaking styles lead to formants being shifted in the frequency domain, as well as variations in phoneme durations. These speaking styles require us to apply various speaker adaptation techniques to reduce feature variation. While DNNs of sufficient size could indeed capture translational invariance, this requires large networks with lots of training examples. CNNs on the other hand capture translational invariance with far fewer parameters by averaging the outputs of hidden units in different local time and frequency regions.

In fact, CNNs have been heavily explored in the image recognition and computer vision fields, offering improvements over DNNs on many tasks (Lawrence, 1997; LeCun et al., 2004). Recently, CNNs have been explored for speech recognition (Abdel-Hamid, Mohamed, Jiang, & Penn, 2012), also showing improvements over DNNs, however on a small vocabulary tasks with shallow

* Corresponding author. Tel.: +1 212 565 0000.
E-mail addresses: tsainath@google.com (T.N. Sainath), bedk@us.ibm.com (B. Kingsbury), gsaon@us.ibm.com (G. Saon), soltau@google.com (H. Soltau), asamir@cs.toronto.edu (A. Mohamed), gdahl@cs.toronto.edu (G. Dahl), bhuvana@us.ibm.com (B. Ramabhadran).

http://dx.doi.org/10.1016/j.neunet.2014.08.005
0893-6080/© 2014 Elsevier Ltd. All rights reserved.

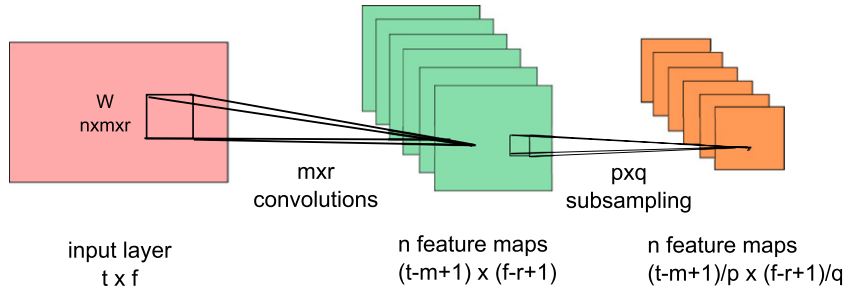

Fig. 1. Diagram showing a typical convolutional network architecture consisting of a convolutional and max-pooling layer. In this diagram, weights with the same line style are shared across all convolutional layer bands. Note this figure shows non-overlapping pooling, which is different than Abdel-Hamid et al. (2012).

networks. Specifically, Abdel-Hamid et al. (2012) introduced a novel framework to model spectral correlations where convolutional weights were shared over limited frequency regions, a technique known as limited weight sharing (LWS). One of the limitations of this LWS approach was that the network was limited to one convolutional layer, unlike most CNN work which uses multiple convolutional layers (LeCun et al., 2004). In this paper, we explore a spatial modeling approach similar to work done in the image recognition community, where convolutional weights are fully shared across all time and frequency components. This modeling approach, known as full weight sharing (FWS), allows for multiple convolutional layers and encourages deeper networks.

The first part of this paper explores the appropriate architecture for CNNs on LVCSR tasks. Specifically, we investigate how many convolutional vs. fully connected layers are needed, the filter size per convolutional layer, an appropriate number of hidden units per layer and a good pooling strategy. In addition, we compare the LWS proposed in Abdel-Hamid et al. (2012) to our FWS strategy.

The second part of this paper explores the best type of input feature to be used with CNN. Various speaker adaptation techniques have been shown to improve the performance of speech recognition systems. In this paper, we focus on how to incorporate feature-space maximum likelihood linear regression (fMLLR) (Gales, 1998) and identity vectors (i-vectors) (Saon, Soltau, Picheny, & Nahamoo, 2013), which do not exhibit locality in frequency, into the CNN framework through a joint CNN/DNN architecture (Sainath, Kingsbury, Mohamed, Dahl, Saon, Soltau, Beran, Aravkin, & Ramabhadran, 2013).

Finally, we investigate the role of rectified linear units (ReLU) and dropout (Hinton, Srivastava, Krizhevsky, Sutskever, & Salakhutdinov, 2012) for Hessian-free (HF) sequence training (Kingsbury et al., 2012) of CNNs. In Dahl, Sainath, and Hinton (2013), ReLU+dropout was shown to give good performance for cross-entropy (CE) trained DNNs but was not employed during HF sequence-training. However, sequence-training is critical for speech recognition performance, providing an additional relative gain of 10%–15% over a CE-trained DNN (Kingsbury et al., 2012). During CE training, the dropout mask changes for each utterance. However, during HF training, we are not guaranteed to get conjugate directions if the dropout mask changes for each utterance. Therefore, in order to make dropout usable during HF, we keep the dropout mask fixed per utterance for all iterations of conjugate gradient (CG) within a single HF iteration.

After analyzing the best CNN architecture, input feature set and ReLU, we then explore using CNNs on a 50 hr English Broadcast News (BN) task (Kingsbury, 2009). Naturally, our best DNN system offers a 13% relative improvement over the GMM/HMM, consistent with gains observed in the literature with DNNs vs. GMM/HMMs (Kingsbury et al., 2012). Comparing DNNs to CNNs, we find that a CNN hybrid system offers a 3% relative improvement over the hybrid DNN, whereas the joint CNN/DNN system which incorporates speaker adaptation and ReLU+dropout offers an 14% improvement. Finally, we explore the behavior of the joint CNN/DNN and ReLU+dropout on two larger scale tasks — namely a 300 hr Switchboard (SWB) task and a 400 hr BN task. We find that using the CNN with these improvements, we can obtain a 12% relative improvement over the DNN on SWB and a 16% relative improvement over the DNN on 400 hr BN.

The rest of this paper is organized as follows. The basic CNN architecture used in this paper is described in Section 2. An exploration of various weight-sharing and pooling strategies are discussed in Section 3, while input feature analysis is discussed in Section 4. Using ReLU+dropout for HF sequence training is discussed in Section 5. Results on three LVCSR tasks are presented in Section 6, Finally, Section 7 concludes the paper and discusses future work.

2. Basic CNN architecture

In this section, we describe the basic CNNs architecture and experimental setup used in this paper.

2.1. CNN description

A typical convolutional network architecture is shown in Fig. 1. First, we are given an input signal $\mathbf{V} \in \Re^{t \times f}$, where t and f are the input feature dimension in time and frequency respectively. A weight matrix $\mathbf{W} \in \Re^{(m \times r) \times n}$ is convolved with the full input V. The weight matrix spans across a small local time-frequency patch of size $m \times r$, where $m <= t$ and $r <= f$. This weight sharing helps to model local correlations in the input signal. The weight matrix has n hidden units (i.e., feature maps). Thus, overall the convolutional operation produces n feature maps of size $(t - m) \times (f - r)$.

After performing convolution, a max-pooling layer helps to remove variability in the time-frequency space that exist due to speaking styles, channel distortions, etc. Given a pooling size of $p \times q$, pooling performs a subsampling operation to reduce the time-frequency space to be $\frac{(t-m+1)p}{\times} \frac{(f-r+1)}{q}$.

Most CNN work in image recognition has the lower network layers be convolutional, while the higher network layers are fully connected. One goal of this paper is to determine an appropriate CNN architecture for speech tasks, including the number of convolutional vs. fully connected layers, hidden units and pooling strategy.

2.2. Experimental details

2.2.1. Data

We perform preliminary experiments to learn the behavior of CNNs for speech on a smaller task. Specifically, the acoustic models are trained on 50 h of data from the 1996 and 1997

English Broadcast News Speech Corpora. Results are reported on 100 speakers from the EARS dev04f set.[1] Details of the training and test corpora can be found in Kingsbury et al. (2012).

2.2.2. Experimental setup

Unless otherwise indicated, we use 40 dimensional VTLN-warped log mel-filterbank +delta+double-delta coefficients (Soltau, Saon, & Kingsbury, 2010), which exhibit local structure, to train the CNNs. A temporal context of 11 frames is used as input into the CNN. The features are mean-and-variance normalized per speaker. The CNNs and fully-connected DNNs use 1024 hidden units per fully connected layer with a sigmoid non-linearity, unless otherwise indicated. The last layer is a softmax layer with 512 output targets. The 512 targets come from clustering context-dependent GMM/HMM states (Young, Odell, & Woodland, 1994). Furthermore, the alignments are obtained by performing a forced-path context-dependent state alignment using an existing GMM/HMM system.

All DNNs and CNNs are trained by optimizing the cross-entropy objective function. The cross-entropy loss is computed as follows; The neural network provides a set of hypothesized posterior probabilities for each of N context-dependent target, which we will denote by $\hat{y}(i) \in [0, 1],\ i = 1 \ldots N$. This is compared to a set of reference targets $y(i) \in 0, 1,\ i = 1 \ldots N$, which have a probability of 1 for the correct class and 0 elsewhere. The cross-entropy loss-function is given by Eq. (1)

$$\mathcal{L}_{XENT}(W) = \sum_{i=1}^{N} \hat{y}(i) \log \frac{\hat{y}(i)}{y(i)}. \tag{1}$$

The CNNs and DNNs are trained on a GPU using mini-batch stochastic gradient descent (SGD), with a batch size of 256. Since we find that for large tasks pre-training does not help, both networks start from random initialization using a methodology described in Glorot and Bengio (0000). A variety of initial learning rates were explored, but the best learning rate was found to be 5e-3. No additional forms of regularization such as momentum or L2 regularization are used. Following a recipe similar to Sainath, Kingsbury, Ramabhadran, Fousek, Novak, and Mohamed (2011), during backpropagation, after one pass through the data, cross-entropy loss is measured on a held-out set and the learning rate is reduced (i.e., annealed) by a factor of 2 if the held-out loss has not improved sufficiently over the previous iteration. Training stops after we have reduced the learning rate 5 times, a produce known as new-bob annealing (Bourlard & Morgan, 1993). Roughly 12–15 overall iterations are run.

The resulting CNN and DNN acoustic models are using in decoding, where output probabilities of the network, after normalization by a prior output probability, are used as HMM emission probabilities. This process is known as "hybrid" decoding, as a DNN is used with an HMM (Dahl et al., 2012; Hinton, Deng, et al., 2012; Sainath, Kingsbury, et al., 2011).

3. Analysis of various CNN strategies for LVCSR

3.1. Convolutional vs. fully connected layers

In this section, we analyze the best approach for combining convolutional and fully connected layers for speech recognition tasks.

[1] Note one speaker has been removed from this dev set for faster decoding purposes.

Table 1
WER as a Function of # of Convolutional Layers.

# of convolutional vs. fully connected layers	WER
No conv, 6 full (DNN)	21.6
1 conv, 5 full	21.3
2 conv, 4 full	18.9
3 conv, 3 full	20.2

3.1.1. Number of convolutional vs. fully connected layers

Most CNN work in image recognition makes use of a few convolutional layers before having fully connected layers. The convolutional layers are meant to reduce spectral variation and model spectral correlation, while the fully connected layers aggregate the local information learned in the convolutional layers to do class discrimination.

Most CNNs explored for image recognition tasks perform full weight sharing (FWS) across all pixels. Unlike images, the local behavior of speech features in low frequency is very different than features in high frequency regions. Abdel-Hamid et al. (2012) addresses this issue by limiting weight sharing (LWS) to frequency components that are close to each other. In other words, low and high frequency components have different weights (i.e. filters). However, this type of approach limits adding additional convolutional layers (Abdel-Hamid et al., 2012) for two reasons. First, in the approach of Abdel-Hamid et al. (2012), the local region each limited filter spans is small so extra convolutions on the outputs of this small local region would not help. Second, we cannot perform convolution with filter outputs across different local regions as the locality constraint between these regions is removed. We adopt a FWS approach similar to the image recognition work, and explore the benefit of including multiple convolutional layers.

Table 1 shows the WER as a function of the number of convolutional layers using FWS and fully connected layers in the network. The fully connected DNN has 1024 hidden units per layer and we have observed that 6 layers is appropriate on the BN task, creating a strong baseline (Sainath, Kingsbury, et al., 2011). For the CNNs, each convolutional layer has 256 hidden units. The pooling for the CNNs also optimized, namely that we pool by 3 in the first CNN layer, and do not pool at all in subsequent layers. Furthermore, the number of fully connected layers in the CNN architecture are chosen to be same for each layer, such that the total number of parameters matches that of the DNN. Finally, the learning rate for each experiment is optimized, and is between 2e−3 and 5e−3.

The table shows that increasing the number of convolutional layers up to 2 helps, and then performance starts to deteriorate. Furthermore, we can see from the table that CNNs offer improvements over DNNs for the same input feature set. One hypothesis for having just two convolutional layers is that the feature dimension for speech is small (i.e., 40), unlike vision, and the behavior in high and low frequency regions is quite different. After two convolutional layers, the feature dimension is much smaller (i.e., 8) and any resulting convolutions would attempt to model locality and remove invariance in a dimension that is already distinguishable.

3.1.2. Number of hidden units

One of the benefits of LWS is that different weights (i.e., filters) focus on different frequency regions. However, we argue that performing FWS with a large enough number of hidden units also allows for differences between different frequency regions to be captured. This type of approach allows for multiple convolutional layers, something that has thus far not been explored before in speech.

Table 2 shows the WER as a function of number of hidden units for the two convolutional layers. We can observe that as we increase the number of hidden units, the WER steadily decreases, confirming our belief that having more hidden units with FWS

Table 2
WER as a function of # of hidden units.

# of hidden units in first/second convolutional layer	Params	WER
128/256	5.1M	19.3
256/256	5.6M	18.9
384/384	7.6M	18.7
512/512	10.0M	18.5

Table 3
Limited vs. full weight sharing, with the number of LWS filters indicated in parenthesis.

Method	# hidden units in conv layers	Params	WER
FWS	256/256	5.6M	18.9
FWS	384/384	7.6M	18.7
FWS	512/512	10.0M	18.5
LWS (2)	128/256	5.4M	18.8
LWS (2)	256/256	6.6M	18.7
LWS (4)	256/256	7.6M	18.9

Table 4
Filter sizes (frequency × time).

Filter size — Layer 1	Filter size — Layer 2	WER
9 × 9	4 × 3	**18.9**
9 × 9	9 × 9	20.2
3 × 14	4 × 3	19.6
15 × 4	4 × 3	19.1
9 × 9	3 × 15	20.3
9 × 9	8 × 3	19.2

is important to help explain variations in frequency in the input signal.

3.1.3. Limited vs. full weight sharing

Finally, for sake of completeness, we compare performance between FWS and LWS. Most LWS work in speech has looked at LWS with one layer (Abdel-Hamid et al., 2012; Deng, Abdel-Hamid, & Yu, 2013), where roughly 40 different LWS filters were used. Because Section 3.1.1 demonstrated the benefit of multiple convolutional layers, in this paper we explore doing LWS with multiple layers. Specifically, the activations from one LWS layer have locality preserving information, and can be fed into another LWS layer. In order to have LWS with multiple layers, we use fewer LWS filters.

Results comparing LWS and FWS are shown in Table 3. The number of LWS filters is indicated in parenthesis. For both LWS and FWS, we used 2 convolutional layers, as this was found in Section 3.1.1 to be appropriate. If we use 2 LWS filters and match the number of parameters to FWS, we get very slight improvements in WER (0.1%). Furthermore, if we increase the number of LWS filters to 4, performance starts to deteriorate, We hypothesize increasing the number of LWS filters further and thus being forced to move to a single convolution layer, as in Abdel-Hamid et al. (2012), would deteriorate performance further. This justifies our choice to use FWS with multiple convolutional layers in our CNN architecture.

While LWS with 2 filters and FWS offer similar performance, FWS is simpler to implement, as we do not have to choose filter locations for each limited weight ahead of time. Thus, we prefer to use FWS. Because FWS with 5.6M parameters (256/256 hidden units per convolution layer) gives the best tradeoff between WER and number of parameters, we use this setting for subsequent experiments.

3.2. Filter sizes

In this section, we explore performance with different the frequency vs. time filter sizes for the two convolutional layers. Table 4 shows the results for different filter sizes. A reasonable filter size is 9 × 9 (frequency-time) for the first convolutional layer, and 4 × 3 for the second convolutional layer.

3.3. Pooling strategies

In this section, we explore various pooling strategies, which have been successful in computer vision tasks, for speech tasks.

3.3.1. Type of pooling

Pooling is an important concept in CNNs which helps to reduce spectral variance in the input features. Max pooling is the most popular pooling strategy for CNNs (LeCun et al., 2004; Sainath, Mohamed, Kingsbury, & Ramabhadran, 2013). Given a pooling region R_j and a set of activations $\{a_1, \ldots, a_{|R_j|}\} \in R_j$, the operation for max-pooling is shown in Eq. (2).

$$s_j = \max_{i \in R_j} a_i. \tag{2}$$

One of the problems with max-pooling is that experiments have shown that is can overfit the training data, and does not necessarily generalize to test data (Zeiler & Fergus, 2013). Two pooling alternatives have been proposed to address some of the problems with max-pooling, l_p pooling (Sermanet, Chintala, & LeCun, 2012) and stochastic pooling (Zeiler & Fergus, 2013).

l_p pooling looks to take a weighted average of activations a_i in pooling region R_j, as shown in Eq. (3).

$$s_j = \left(\sum_{i \in R_j} a_i^p \right)^{\frac{1}{p}}. \tag{3}$$

$p = 1$ can be seen as a simple form of averaging while $p = \infty$ corresponds to max-pooling. One of the problems with average pooling is that all elements in the pooling region are considered, so areas of low-activations may downweight areas of high activation. l_p pooling for $p > 1$ is seen as a tradeoff between average and max-pooling. l_p pooling has shown to give large improvements in error rate in computer vision tasks compared to max pooling (Sermanet et al., 2012).

Stochastic pooling is another pooling strategy that addresses the issues of max and average pooling. In stochastic pooling, first a set of probabilities p for each region j is formed by normalizing the activations across that region, as shown in Eq. (4).

$$p_i = \frac{a_i}{\sum_{k \in R_j} a_k} \tag{4}$$

$$s_j = a_l \quad \text{where } l \sim P(p_1, p_2, \ldots, p_{|R_j|}). \tag{5}$$

A multinomial distribution is created from the probabilities and the distribution is sampled based on p to pick the location l and corresponding pooled activation a_l. This is shown by Eq. (5). Stochastic pooling has the advantages of max-pooling but prevents overfitting due to the stochastic component. Stochastic pooling has also shown huge improvements in error rate in computer vision (Zeiler & Fergus, 2013).

Given the success of l_p and stochastic pooling, we compare both of these strategies to max-pooling on an LVCSR task. Results for the three pooling strategies are shown in Table 5. For all three pooling strategies, we use a pooling size of 3 for the first convolutional layer, and no pooling for the second convolutional layer, as we found this to be appropriate across different speech tasks of both 8 kHZ and 16 HZ in Sainath, Mohamed, et al. (2013).

Stochastic pooling seems to provide improvements over max and l_p pooling, though the gains are slight compared to gains seen on vision tasks. For example, stochastic pooling seemed to help on smaller vision which have thousands of training examples, such as MNIST, CIFAR AND Street View House Numbers (Zeiler & Fergus, 2013). However, the LVCSR tasks we look at are between 50 and 400 h, amounting to between 20 and 160 million training frames. With a larger amount of training data, we hypothesize that regularization methods, such as l_p and stochastic pooling, do not offer great improvements over max pooling.

Table 5
Results with different pooling types.

Method	WER
Max pooling	18.9
Stochastic pooling	18.8
l_p pooling	18.9

Table 6
Pooling with and without overlap.

Method	WER
Pooling no overlap	18.9
Pooling with overlap	18.9

Table 7
Pooling in time.

Pooling type	WER, Pooling in time	WER, Pooling in time+frequency
Max	18.9	18.9
Stochastic	18.8	18.8
l_p	18.9	18.8

Table 8
WER as a function of input feature.

Feature	WER
Mel FB	19.7
VTLN-warped mel FB	19.5
VTLN-warped mel FB + d + dd	18.9

3.3.2. Overlapping pooling

The pooling layer in CNNs consists of a set of pooling units which span over a neighborhood of size z. Each pooling unit is spaced s apart. If $s = z$, the pooling is non-overlapping, while if $s < z$, the pooling is overlapping. Work in computer vision has shown that overlapping pooling can improve error rate by 0.3%–0.5% compared to non-overlapping pooling (Krizhevsky, Sutskever, & Hinton, 2012).

Table 6 compares overlapping and non-overlapping pooling on an LVCSR speech task. For non-overlapping pooling we set $s = z = 3$, while for non-overlapping pooling we set $z = 3$ and $s = 2$. One thing to point out is that because overlapping pooling has many more activations, in order to keep the experiment comparable between overlapping and non-overlapping pooling, the number of parameters for the two experiments was matched. The table shows that there is no difference in WER between overlapping or non-overlapping pooling. Again, on tasks with a lot of data such as speech, regularization mechanisms such as overlapping pooling, do not seem to help compared to smaller computer vision tasks.

3.3.3. Pooling in time

Most previous CNN work in speech explored pooling in frequency only (i.e., y dimension) (Abdel-Hamid et al., 2012; Deng et al., 2013; Sainath, Mohamed, et al., 2013), though Waibel, Hanazawa, Hinton, Shikano, and Lang (1989) did investigate CNNs with pooling in time, but not frequency. However, most CNN work in vision performs pooling in both space and time (i.e., x and y dimensions) (Krizhevsky et al., 2012; Sermanet et al., 2012). In this paper, we do a deeper analysis of pooling in time for speech. One thing we must ensure with pooling in time in speech is that there is overlap between the pooling windows. Otherwise, pooling in time without overlap can be seen as subsampling the signal in time, which degrades performance. Pooling in time with overlap can thought of as a way to smooth out the signal in time, another form of regularization.

Table 7 compares pooling in frequency to pooling in both time and frequency for both max, stochastic and l_p pooling. In our experiments, we pool in time using a pooling size of 2, with an overlap of 1, as this is the most conservative form of pooling. We see that pooling in time and frequency helps slightly compared to pooling in frequency only. However, the gains are not large, and are likely to be diminished after sequence training, and thus our further experiments pool in frequency only. It appears that for large tasks with more data, regularizations such as pooling in time are not helpful, similar to other regularization schemes such as l_p/stochastic pooling and pooling with overlap in frequency.

3.4. Conclusions

Analysis from Section 3.1 indicated that FWS, with 2 convolutional layers of 256 hidden units, followed by 4 fully connected layers, was the best strategy. Furthermore, investigation of filter sizes in Section 3.2 indicated that a 9x9 frequency-time filter for the first convolutional layer, followed by a 4x3 filter for the second convolutional layer, was best. Finally, pooling strategy analysis in section shows that non-overlapping pooling max-pooling, and pooling only in frequency, was a reasonable strategy. A pooling size of 3 was used for the first layer, and no pooling was done in the second layer. Prior experience has shown us that settings which work well for the smaller 50 hr Broadcast News task seem to generalize well to larger data sets, and thus we use the settings we found best for 50 hr when exploring gains with CNNs on larger data sets.

4. Analysis of input features

Convolutional neural networks require features which are locally correlated in time and frequency. This implies that Linear Discriminant Analysis (LDA) features, which are very commonly used in speech, cannot be used with CNNs as they remove locality in frequency (Abdel-Hamid et al., 2012). Mel filter-bank (FB) features are one type of speech feature which exhibit this locality property (Mohamed, Hinton, & Penn, 2012).

Typically, various speaker adaptation techniques are applied to speech recognition features and have been shown to improve speech recognition performance tremendously (Gales, 1998; Lee & Rose, 1996; Sainath, Ramabhadran, Picheny, Nahamoo, and Kanevsky, 2011). Vocal tract length normalization (VTLN) (Lee & Rose, 1996) is a popular technique that warps the speech from different speakers and different vocal tract lengths into a canonical speaker with an average vocal tract length. VTLN is attractive because it applies the warping to the mel-filterbank of each speaker, and therefore preserves the locality in frequency. Another common practice to improve performance is to include extra time-dynamic information of the feature by including time-derivative information of features, known as deltas (d) and double-deltas (dd) (Young, Evermann, Gales, Hain, Kershaw, Liu, Moore, Odell, Ollason, Povey, Valtchev, & Woodland, 2006). While previous experiments in Section 3 were done with VTLN-warped log-mel+d+dd features, we analyze if the extra complexity of these features is really needed with a CNN. Specifically, since both pooling and VTLN attempt to remove translational variance from the input signal, its possible both are not needed.

Table 8 shows that indeed the performance of CNNs improves by incorporating VTLN and delta information, confirming the complementarity between VTLN and pooling is needed for CNNS. Note that the CNN for each input feature in Table 8 has the same number of parameters. Note that we have not given the performance with DNNs with different transformations of log-mel features, as we have observed better performance for DNNs using feature-space maximum likelihood linear regression features (fMLLR) (Gales, 1998) features rather than VTLN-warped log-mel features (Lee & Rose, 1996). Rather, we leave the comparison between CNNs and DNNs with the best feature sets to Section 6.

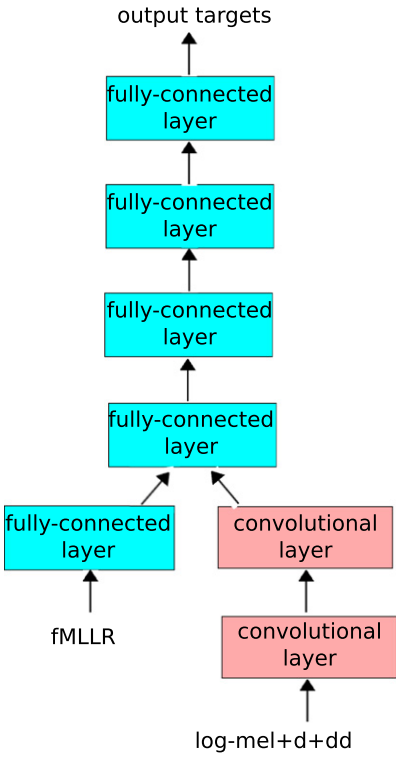

Fig. 2. Joint CNN/DNN architecture.

In speech recognition, we have seen further improvements in performance can be achieved using speaker-adapted features which do not display locality in frequency, including feature-space maximum likelihood linear regression features (fMLLR) (Gales, 1998) and speaker identity vectors (i-vectors) (Saon et al., 2013). In the sections below, we discuss how to effectively incorporate these features into our CNN framework.

4.1. fMLLR transforms

fMLLR (Gales, 1998) is a popular speaker-adaptation technique used to reduce variability of speech due to different speakers. The fMLLR transformation applied to features assumes that either features are uncorrelated and can be modeled by diagonal covariance Gaussians, or features are correlated and can be modeled by a full covariance Gaussians.

While correlated features are better modeled by full-covariance Gaussians, full-covariance matrices dramatically increase the number of parameters per Gaussian component, oftentimes leading to parameter estimates which are not robust. Thus fMLLR is most commonly applied to a decorrelated space. When fMLLR was applied to the correlated log-mel feature space with a diagonal covariance assumption, little improvement in WER was observed (Sainath, Mohamed, et al., 2013).

Semi-tied covariance matrices (STCs) (Gales, 1999) have been used to decorrelate the feature space so that it can be modeled by diagonal Gaussians. STC offers the added benefit in that it allows a few full covariance matrices to be shared over many distributions, while each distribution has its own diagonal covariance matrix.

In this paper, we explore applying fMLLR to correlated features (such as log-mel) by first decorrelating them such that we can appropriately use a diagonal Gaussian approximation with fMLLR. We then transform the fMLLR features back to the correlated space so that they can be used with CNNs.

The algorithm to do this is described as follows. First, starting from correlated feature space $\mathbf{f} \in \Re^n$, we estimate an STC matrix $\mathbf{S} \in \Re^{n \times m}$ to map the features into an uncorrelated space. This mapping is given by transformation (6).

To estimate the STC matrix, we build a GMM/HMM system on feature space f for a set of context-dependent states. Because of data sparsity issues in estimated full-covariance Gaussians, the covariance matrix $\boldsymbol{\Sigma}^{(m)}$ for each Gaussian m consists of a component specific diagonal covariance matrix $\Sigma^{(m)}_{diag}$ and a non-diagonal semi-tied covariance matrix S, as shown by Eq. (7). The semi-tied covariance matrix S, along with other Gaussian parameters (i.e., mean, diagonal covariance) are estimated via expectation–maximization. We refer the reader to Gales (1999) for further details regarding STC matrices.

$$\mathbf{Sf} \tag{6}$$

$$\boldsymbol{\Sigma}^{(m)} = \mathbf{S}\boldsymbol{\Sigma}^{(m)}_{diag}\mathbf{S}^T. \tag{7}$$

Next, in the uncorrelated space, an fMLLR matrix is estimated for each speaker p, and is applied to the STC transformed features. fMLLR assumes a linear transformation between the initial feature $\mathbf{Sf}$ and the transformed features $\mathbf{T}$, as shown by transformation (8).

$$\mathbf{T} = \mathbf{A_p}\mathbf{Sf} + \mathbf{b_p}. \tag{8}$$

Given an GMM model θ, the goal in fMLLR is to find a transformation of feature vectors to maximize their likelihood. An fMLLR matrix $\mathbf{M_p} = \{\mathbf{A_p}, \mathbf{b_p}\} \in \Re^{mxm}$ is estimated for each speaker p.

$$\{\mathbf{A}^*, \mathbf{b}^*\} = \arg\max_{(A,b)} P(T|\theta). \tag{9}$$

After estimating an fMLLR transformation per speaker p, this transformation is applied to the uncorrelated log-mel features, i.e. $\mathbf{Sf}$. This transformation is given in transformation (10).

$$\mathbf{M_p}\mathbf{Sf}. \tag{10}$$

Thus far, transformations (6) and (10) demonstrate standard transformations in speech with STC and fMLLR matrices. However, in speech recognition tasks, once features are decorrelated with STC, further transformation (i.e. fMLLR, fBMMI) are applied in this decorrelated space, as shown in transformation (10). The features are never transformed back into the correlated space.

However for CNNs, using correlated features is critical. By multiplying the fMLLR transformed features by an inverse STC matrix, we can map the decorrelated fMLLR features back to the correlated space, so that they can be used with a CNN. The transformation we propose is given in (11).

$$\mathbf{S}^{-1}\mathbf{M_p}\mathbf{Sf}. \tag{11}$$

4.2. Joint CNN/DNN architecture

Typically, fMLLR transformations are applied to Linear Discriminant Analysis (LDA) features, which are already decorrelated. When fMLLR features are created in such a manner, a CNN cannot be used, though a DNN can be used. We explore whether feeding fMLLR features to a DNN layer, and combining this with a CNN, is more effective than creating fMLLR log-mel features, as described in Section 4.1. The architecture for combining CNNs and DNNs is shown in Fig. 2.

Specifically, log-mel features are fed to a CNN layer, and fMLLR features are fed to a DNN layer. The output of the fMLLR fully-connected layer and log-mel convolutional layers are joined together and fed to subsequent fully-connected layers. The entire network is trained jointly. This can be thought of as combining features generated from a DNN-style and CNN-style network.

Related to this idea is the work of Sermanet et al. (2012), which considered the same input feature into the network, and looked at combining outputs from different layers of the neural network. Our work is different in that it provides a flexible joint architecture for combining different feature sets which can be modeled by CNNs or DNNs. We call the proposed architecture a joint CNN/DNN model.

Table 9
WER, incorporating i-vectors into CNNs.

Method	WER
log-mel	19.5
log-mel + I-vector, CNN only	18.8
log-mel + I-vector, joint CNN/DNN	18.7

4.3. I-vectors

Identity vectors (i-vectors) are commonly used features for speaker verification and speaker recognition applications, as they encapsulate relevant information about a speaker's identity in a low-dimensional feature vector. This also makes these attractive speaker-adaptive features for ASR applications. For example, Saon et al. (2013) explored combining i-vectors with fMLLR features, as input into a DNN. The paper showed that incorporating i-vectors provided an additional 5%–6% relative improvement.

Incorporating i-vectors into a CNN architecture is a bit more challenging, as CNN require features which obey a frequency (and time) locality property, a property which i-vectors do not have. We compare two different methodologies to incorporate i-vectors into CNNs.

I-vectors can be incorporated into the convolutional layer by concatenating the feature with every localized frequency patch. For example, if the CNN sees a 9x9 time-frequency patch of localized features, we concatenate the 100-dimensional i-vectors into this feature so that the new filter size becomes 9x109. Every time the CNN shifts in frequency, the same i-vector is concatenated to the current set of localized features. This idea has been explored before when incorporating the non-localized energy feature into a CNN (Abdel-Hamid et al., 2012). Alternatively, since we know i-vectors can be incorporated into fully connected DNN layers, we can use a joint CNN/DNN modeling approach discussed in Section 4.2. Specifically, we can feed the i-vectors into one fully connected DNN layer, and then join this output into the first fully connected layer of the CNN.

Table 9 shows the WER for the two different methodologies. Just for simplicity to avoid the extra dimensions with d+dd, we compare the two different ideas of incorporating i-vectors with just log-mel features. We see there is an improvement in WER when i-vectors are incorporated, but there is not a huge difference in final performance when incorporating i-vectors at the CNN or DNN level. Incorporating at the DNN layer is a bit faster, as we do not need to add i-vectors into the localized features for each CNN shift. For this reason, we use this approach for i-vectors in subsequent experiments.

4.4. Results

Results with the various discussed speaker-adaptation techniques are shown in Table 10. Notice that by applying fMLLR in a decorrelated space, we can achieve WER of 18.3%, a 0.5% improvement over the baseline VTLN-warped log-mel system. However, using fMLLR in the DNN stream of the joint CNN/DNN architecture is even more promising with a WER of 18.0%. Finally, incorporating i-vectors into the fMLLR DNN stream provides even further improvements. Overall, the proposed speaker adaptation techniques achieve a WER of 16.9%, a 10% relative improvement in WER over the baseline at a WER of 18.8%.

5. Sequence training with rectified linear units and dropout

At IBM, two stages of Neural Network training are performed. First, DNNs are trained with a frame-discriminative stochastic gradient descent (SGD) cross-entropy (CE) criterion. Second, CE-trained DNN weights are re-adjusted using a sequence-level objective function (Kingsbury, 2009). Specifically, using the cross-entropy trained DNN, sequence information is saved out in the form of a numerator lattice, representing the correct set of words, and a denominator lattice, representing competing hypotheses. A state-level minimum Bayes risk (sMBR) sequence objective function is used. More details of sequence training can be found in Kingsbury (2009). Since speech is a sequence-level task, this objective is more appropriate for the speech recognition problem. Numerous studies have shown that sequence training provides an additional 10%–15% relative improvement over a CE trained DNN (Kingsbury et al., 2012; Sainath, Mohamed, et al., 2013). Using a 2nd order Hessian-free (HF) optimization method is critical for performance gains with sequence training compared to SGD-style optimization, though not as important for CE-training (Kingsbury et al., 2012).

Rectified Linear Units (ReLU) and dropout (Hinton, Srivastava, et al., 2012) have recently been proposed as a way to regularize large neural networks. In fact, ReLU+dropout was shown to provide a 5% relative reduction in WER for cross-entropy-trained DNNs on a 50 hr English Broadcast News LVCSR task (Dahl et al., 2013). However, subsequent HF sequence training (Kingsbury et al., 2012) *that used no dropout* erased some of these gains, and performance was similar to a DNN trained with a sigmoid non-linearity and no dropout. Given the importance of sequence-training for neural networks, in this paper, we propose a strategy to make dropout effective during HF sequence training. Results are presented in the context of CNNs, though this algorithm can also be used with DNNs.

5.1. Hessian-free training

One popular 2nd order technique for DNNs is Hessian-free (HF) optimization (Martens, 2010). Let $\boldsymbol{\theta}$ denote the network parameters, $\mathcal{L}(\boldsymbol{\theta})$ denote a loss function, $\nabla\mathcal{L}(\boldsymbol{\theta})$ denote the gradient of the loss with respect to the parameters, $\mathbf{d}$ denote a search direction, and $\mathbf{B}(\boldsymbol{\theta})$ denote a Hessian approximation matrix characterizing the curvature of the loss around $\boldsymbol{\theta}$. The central idea in HF optimization is to iteratively form a quadratic approximation to the loss and to minimize this approximation using conjugate gradient (CG).

$$\mathcal{L}(\boldsymbol{\theta}+\mathbf{d}) \approx \mathcal{L}(\boldsymbol{\theta}) + \nabla\mathcal{L}(\boldsymbol{\theta})^T\mathbf{d} + \frac{1}{2}\mathbf{d}^T\mathbf{B}(\boldsymbol{\theta})\mathbf{d}. \tag{12}$$

During each iteration of the HF algorithm, first, the gradient is computed using all training examples. Second, since the Hessian cannot be computed exactly, the curvature matrix $\mathbf{B}$ is approximated by a damped version of the Gauss–Newton matrix $\mathbf{G}(\boldsymbol{\theta})+\lambda\mathbf{I}$, where λ is set via Levenberg–Marquardt. Then, Conjugate gradient (CG) is run for multiple-iterations until the relative per-iteration progress made in minimizing the CG objective function falls below a certain tolerance. During each CG iteration, Gauss–Newton matrix–vector products are computed over a sample of the training data.

5.2. Dropout

Dropout is a popular technique to prevent over-fitting during neural network training (Hinton, Srivastava, et al., 2012). Specifically, during the feed-forward operation in neural network training, dropout omits each hidden unit randomly with probability p. This prevents complex co-adaptations between hidden units, forcing hidden units to not depend on other units. Specifically, using dropout the activation $\mathbf{y}^l$ at layer l is given by Eq. (13), where $\mathbf{y}^{l-1}$ is the input into layer l, $\mathbf{W}^l$ is the weight for layer l, $\mathbf{b}$ is the bias, f is the non-linear activation function (i.e. ReLU) and $\mathbf{r}$ is a binary mask, where each entry is drawn from a Bernoulli(p) distribution with probability p of being 1. Since dropout is not used during decoding, the factor $\frac{1}{1-p}$ used during training ensures that at test time,

Table 10
WER with speaker-adapted features.

Feature	WER
VTLN-warped log-mel+d+dd	18.8
CNN only, fMLLR + VTLN-warped log-mel+d+dd	18.3
Joint CNN/DNN, fMLLR (DNN) + VTLN-warped log-mel+d+dd (CNN)	18.0
Joint CNN/DNN, fMLLR + i-vectors (DNN)+ VTLN-warped log-mel+d+dd	**16.9**

when no units are dropped out, the correct total input will reach each layer.

$$\mathbf{y}^l = f\left(\frac{1}{1-p}\mathbf{W}^l(\mathbf{r}^{l-1} * \mathbf{y}^{l-1}) + \mathbf{b}^l\right). \tag{13}$$

5.2.1. Combining HF + dropout

Conjugate gradient tries to minimize the quadratic objective function given in Eq. (12). For each CG iteration, the damped Gauss–Newton matrix, $\mathbf{G}(\boldsymbol{\theta})$, is estimated using a subset of the training data. This subset is fixed for all iterations of CG. This is because if the data used to estimate $\mathbf{G}(\boldsymbol{\theta})$ changes, we are no longer guaranteed to have conjugate search directions from iteration to iteration.

Recall that dropout produces a random binary mask for each presentation of each training instance. However, in order to guarantee good conjugate search directions, for a given utterance, the dropout mask per layer cannot change during CG. The appropriate way to incorporate dropout into HF is to allow the dropout mask to change for different layers and different utterances, but to fix it for all CG iterations while working with a specific layer and specific utterance (although the masks can be refreshed between HF iterations).

As the number of network parameters is large, saving out the dropout mask per utterance and layer is infeasible. Therefore, we randomly choose a seed for each utterance and layer and save this out. Using a randomize function with the same seed guarantees that the same dropout mask is used per layer/per utterance.

5.2.2. Results

We experimentally confirm that using a dropout probability of $p = 0.5$ in the 3rd and 4th layers is reasonable, and the dropout in all other layers is zero. For simplicity, the CNNs in these experiments are trained with VTLN-warped log-mel+d+dd features only. The CE WER with sigmoid is 18.9% and with ReLU it is 18.7%.

Results with different dropout techniques after HF sequence training are shown in Table 11. Notice that if no dropout is used, the WER is the same as with sigmoid, a result which was also found for DNNs in Dahl et al. (2013). By using dropout but fixing the dropout mask per utterance across all CG iterations, we can achieve a 0.6% improvement in WER. Finally, if we compare this to varying the dropout mask per CG training iteration, the WER increases. We did not run experiments using sigmoids with dropout for CNNs, as we found in Dahl et al. (2013) that sigmoids with dropout was not helpful.

Further investigation in Fig. 3 shows that if we vary the dropout mask, there is slow convergence of the loss during training, particularly when the number of CG iterations increases during the later part of HF training. This shows experimental evidence that if the dropout mask is not fixed, we cannot guarantee that CG iterations produce conjugate search directions for the loss function.

Table 11
WER of HF sequence training + dropout.

Non-linearity	WER
Sigmoid	15.7
ReLU, no dropout	15.6
ReLU, dropout fixed for CG iterations	**15.0**
ReLU, dropout per CG iteration	15.3

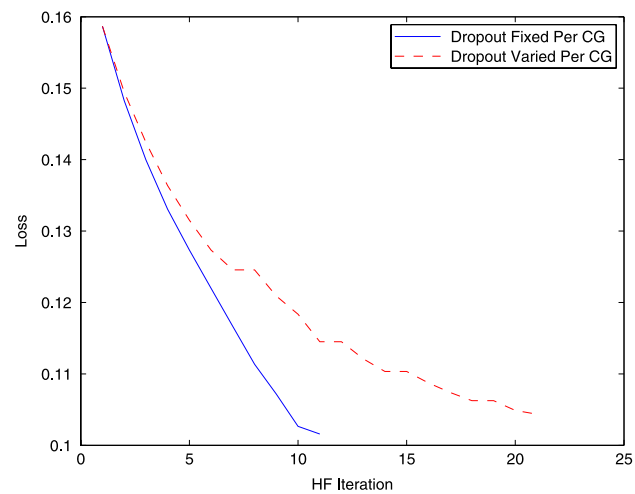

Fig. 3. Held-out loss with dropout techniques.

6. Results on LVCSR tasks

In this section, we compare CNN performance to two state of the art techniques used for LVCSR tasks, namely DNNs and GMM/HMMs. We report CNN performance with the architecture described in Section 3, and also break down the improvements due to speaker-adaptation and ReLU, described in Sections 4 and 5 respectively.

The GMM system is trained using our standard recipe (Soltau et al., 2010), which is briefly described below. The raw acoustic features are 13-dimensional MFCC features with speaker-based mean, variance, and vocal tract length normalization (VTLN) (Lee & Rose, 1996). Temporal context is included by splicing 9 successive frames of MFCC features into supervectors, then projecting to 40 dimensions using LDA. Next, a set of feature-space speaker-adapted (FSA) features are created using feature-space maximum likelihood linear regression (fMLLR) (Gales, 1998). Finally, feature-space discriminative training and model-space discriminative training are done using the boosted maximum mutual information (BMMI) criterion (Povey, Kanevsky, Kingsbury, Ramabhadran, Saon, & Visweswariah, 2008). At test time, unsupervised adaptation using regression tree MLLR is performed.

Both CNN and DNN systems are trained in a hybrid setup, where output probabilities of the network are taken to be HMM emission probabilities. The networks are first trained using the cross-entropy objective function, followed by Hessian-free sequence-training (Kingsbury et al., 2012). Unless otherwise stated, the DNNs are trained with fMLLR features with a 9-frame context around the current frame, as this was found to be the best feature set for DNNs (Sainath, Kingsbury, et al., 2011).

Table 12
WER on broadcast news, 50 hr.

Model	Feature	Non-linearity	dev04f
GMM/HMM	fBMMI		18.8
DNN	fMLLR	sigmoid	16.3
CNN	log-mel	sigmoid	15.8
CNN+DNN	log-mel+(fMLLR+i-vectors)	sigmoid	14.2
CNN+DNN	log-mel+(fMLLR+i-vectors)	ReLU	**13.6**
DNN	log-mel+(fMLLR+i-vectors)	ReLU	14.2

Table 13
WER on broadcast news, 400 hr.

Model	Feature	Non-linearity	dev04f
GMM/HMM	fBMMI		16.0
DNN	fMLLR	sigmoid	15.1
CNN	log-mel	sigmoid	13.5
CNN+DNN	log-mel+(fMLLR+i-vectors)	ReLU	**12.7**

6.1. 50 hr broadcast news

6.1.1. Experimental setup

Our first set of experiments are conducted on the same 50 hr English Broadcast News (BN) task used in Section 2, and results reported on the EARS dev04f set. The GMMs are trained with 3000 quinphone states and 30 K diagonal covariance Gaussians. The CNN system has 2 convolutional layers with 256 hidden units and 3 fully connected layers, while the DNN system has 5 fully connected layers. Both the CNN and DNN systems have 1024 hidden units for each fully connected layer, and a softmax layer with 3000 output targets.

6.1.2. Results

Table 12 shows the performance of CNN hybrid systems, and compares this to DNN and GMM/HMM systems. The table indicates that the DNN hybrid offers a 13% relative improvement over the GMM/HMM, consistent with gains observed in the literature with DNNs vs. GMM/HMMs (Kingsbury et al., 2012). However, the CNN systems are far better than the DNNs. The CNN hybrid trained with log-mel offers a 3% relative improvement over this DNN hybrid. Furthermore, by including the speaker-adaptation and ReLU+dropout improvements, we can achieve an 11% relative improvement over the CNN alone, and a 14% relative improvement over the DNN. Finally, we see that if we train a DNN with the best feature set (i.e., log-mel+fMLLR+i-vectors), the CNN still achieves a 4% relative improvement in WER over the DNN.

6.2. 400 hr broadcast news

6.2.1. Experimental setup

We next explore scalability of CNNs on 400 h of English Broadcast News (Kingsbury, 2009). Results are reported on the DARPA EARS dev04f set. The GMMs use 5999 quinphone states and 150 K diagonal-covariance Gaussians. The CNN system has 2 convolutional layer with 256 hidden units and 3 fully connected layers, while the DNN system has 5 fully connected layers. Both the CNN and DNN systems have 1024 hidden units for each fully connected layer, and have a softmax layer with 5999 output targets.

6.2.2. Results

Table 13 shows the performance of the CNN systems compared to both DNNs and GMM/HMMs. The CNN trained on log-mel features alone offers a 16% relative improvement over the GMM/HMM system, and a 11% relative improvement over the DNN hybrid system. Furthermore, by including the speaker-adaptation and ReLU+dropout improvements, we can achieve an 6% relative improvement over the CNN alone, and a 16% relative improvement over the DNN. This helps to strengthen the hypothesis that CNNs are better than DNNs for speech tasks.

Table 14
WER on switchboard, 300 hr.

Model	Feature	Non-linearity	Hub5'00
GMM/HMM	fBMMI		14.5
DNN	fMLLR features	sigmoid	12.2
CNN	log-mel	sigmoid	11.8
CNN+DNN	log-mel+(fMLLR+i-vectors)	ReLU	**10.7**

6.3. 300 hr switchboard

6.3.1. Experimental setup

Finally, we explore CNNs performance on 300 hr of conversational American English telephony data from the Switchboard corpus. Results are reported on the Hub5'00 set.

The GMM systems are trained using the same methods used for Broadcast News, namely using speaker-adaptation with VTLN and fMLLR, followed by feature and model-space discriminative training with the BMMI criterion. Results are reported after MLLR. The GMMs use 8192 quinphone states and 372 K Gaussians. Similar to the Switchboard experiments in Kingsbury et al. (2012), the DNN hybrid system are trained with fMLLR features with an 11-frame context (± 5) around the current frame. The DNNs have six hidden layers each containing 2048 sigmoidal units, and a softmax layer with 8192 output targets. For the CNNs, two convolutional layers have 512 hidden units, and five fully connected layers have 2048 hidden units, and the softmax layer has 8192 output targets.

6.3.2. Results

Table 14 shows the performance of the CNNs compared to both DNNs and GMM/HMMs. The CNN system trained with log-mel features offer a 20% relative improvement over the GMM/HMM system, and 3% relative improvement over the hybrid DNN. Furthermore, by including the speaker-adaptation and ReLU+dropout improvements, we can achieve an 9% relative improvement over the CNN alone, and a 12% relative improvement over the DNN. Again, this confirms that across a wide variety of LVCSR tasks, CNNs are better than DNNs.

7. Conclusions and future work

In this paper, we explored how to make CNNs a more powerful model for speech tasks compared to DNN. Specifically, we experimentally determined an appropriate number of convolutional layers, hidden units, filter size and pooling strategy for CNNs. In addition, we introduced a joint CNN/DNN architecture to allow speaker-adapted features to be used in this framework. Finally, we investigated a strategy to make dropout effective after HF sequence training. Experiments on 3 LVCSR tasks, namely a 50 and 400 hr BN task and a 300 hr SWB task, indicate that a CNN with the proposed speaker-adapted and ReLU+dropout ideas allow for a 12%–14% *relative improvement* in WER over a strong DNN system. The performance with the proposed ideas are state-of-the-art, giving a WER of 13.6% on 50 hr BN, 12.7% on 400 hr BN and 10.7% on 300 hr SWB are state-of-the art, surpassing the previous best DNN results of 16.3% on 50 hr BN, 15.1% on 400 hr BN and 12.2% on SWB (Sainath, Mohamed, et al., 2013).

References

Abdel-Hamid, O., Mohamed, A., Jiang, H., & Penn, G. (2012). Applying convolutional neural network concepts to hybrid NN-HMM model for speech recognition, In *Proc. ICASSP*.

Bourlard, H., & Morgan, N. (1993). *Connectionist speech recognition: a hybrid approach*. Kluwer Academic Publishers.

Dahl, G., Sainath, T., & Hinton, G. (2013), Improving deep neural networks for LVCSR using rectified linear units and dropout. In *Proc. ICASSP*.

Dahl, G., Yu, D., Deng, L., & Acero, A. (2012). Context-dependent pre-trained deep neural networks for large vocabulary speech recognition. *IEEE Transactions on Audio, Speech, and Language Processing*, *20*(1), 30–42.

Deng, L., Abdel-Hamid, O., & Yu, D. (2013). A deep convolutional neural network using heterogeneous pooling for trading acoustic invariance with phonetic confusion. In *Proc. ICASSP*.

Gales, M. (1998). Maximum likelihood linear transformations for HMM-based speech recognition. *Computer Speech and Language*, *12*(2), 75–98.

Gales, M. (1999). Semi-tied covariance matrices for hidden markov models. *IEEE Transactions on Speech and Audio Processing*, *7*, 272–281.

Glorot, X., & Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. In *Proc. AI Stats*.

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

Hinton, G., Srivastava, N., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *The Computing Research Repository (CoRR) 1207*, 0580.

Jaitly, N., Nguyen, P., Senior, A. W., & Vanhoucke, V. (2012). Application of pretrained deep neural networks to large vocabulary speech recognition, In *Proc. Interspeech*.

Kingsbury, B. (2009). Lattice-based optimization of sequence classification criteria for neural-network acoustic modeling. In *Proc. ICASSP*.

Kingsbury, B., Sainath, T. N., & Soltau, H. (2012). Scalable minimum bayes risk training of deep neural network acoustic models using distributed hessian-free optimization. In *Proc. Interspeech*.

Krizhevsky, A., Sutskever, I., & Hinton, G. (2012). Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems*,.

Lawrence, S. (1997). Face recognition: A convolutional neural-network approach. *IEEE Transactions on Neural Networks*, *8*(1), 98–113.

LeCun, Y., & Bengio, Y. (1995). Convolutional networks for images, speech, and time-series. In *The handbook of brain theory and neural networks*. MIT Press.

Lecun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*,.

LeCun, Y., Huang, F., & Bottou, L. (2004). Learning methods for generic object recognition with invariance to pose and lighting, *Proc. CVPR*.

Lee, L., & Rose, R. C. (1996). Speaker normalization using efficient frequency warping procedures, In *Proc. ICASSP*.

Martens, J. (2010) Deep learning via Hessian-free optimization, In *Proc. intl. conf. on machine learnbeing (ICML)*.

Mohamed, A., Hinton, G., & Penn, G. (2012). Understanding how deep belief networks perform acoustic modelling, In *ICASSP*.

Povey, D., Kanevsky, D., Kingsbury, B., Ramabhadran, B., Saon, G., & Visweswariah, K. (2008). Boosted MMI for model and feature-space discriminative training, In *Proc. ICASSP* (pp. 4057–4060).

Sainath, T., Mohamed, A., Kingsbury, B., & Ramabhadran, B. 2013 Deep Convolutional Neural Networks for LVCSR, In: Proc. ICASSP.

Sainath, T. N., Kingsbury, B., Mohamed, A., Dahl, G., Saon, G., Soltau, H., Beran, T., Aravkin, A. Y., & Ramabhadran, B. 2013 Improvements to Deep Convolutional Neural Networks for LVCSR, In: Proc. ASRU.

Sainath, T. N., Kingsbury, B., Ramabhadran, B., Fousek, P., Novak, P., & Mohamed, A. 2011 Making Deep Belief Networks Effective for Large Vocabulary Continuous Speech Recognition, In: Proc. ASRU.

Sainath, T. N., Ramabhadran, B., Picheny, M., Nahamoo, D., & Kanevsky, D. (2011). Exemplar-based sparse representation features: from TIMIT to LVCSR.

Saon, G., Soltau, H., Picheny, M., & Nahamoo, D. 2013 Speaker adaptation of neural network acoustic models using I-vectors. In *Proc. ASRU* (in preperation).

Seide, F., Li, G., & Yu, D. 2011 Conversational Speech Transcription Using Context-Dependent Deep Neural Networks, In: Proc. Interspeech.

Sermanet, P., Chintala, S., & LeCun, Y. 2012 Convolutional neural networks applied to house numbers digit classification, In: Pattern Recognition (ICPR), 2012 21st International Conference on.

Soltau, H., Saon, G., & Kingsbury, B. 2010 The IBM Attila Speech Recognition Toolkit, In: Proc. SLT.

Waibel, A., Hanazawa, T., Hinton, G., Shikano, K., & Lang, K. (1989). Phoneme recognition using time-delay neural networks. *IEEE Transactions on Acoustics, Speech and Signal Processing*, *37*(3), 328–339.

Young, S. J., Evermann, G., Gales, M. J. F., Hain, T., Kershaw, D., Liu, X., Moore, G., Odell, J., Ollason, D., Povey, D., Valtchev, V., & Woodland, P. C. (2006). *The HTK Book (for HTK Version 3.4)*. University of Cambridge.

Young, S. J., Odell, J., & Woodland, P. 1994 Tree-based State Tying for High Accuracy Acoustic Modelling, In: Proc. HLT. pp. 307–312.

Zeiler, M., & Fergus, R. 2013 Stochastic Pooling for Regularization of Deep Convolutional Neural Networks, In: Proc. of the International Conference on Representaiton Learning (ICLR).