CONVOLUTIONAL, LONG SHORT-TERM MEMORY, FULLY CONNECTED DEEP NEURAL NETWORKS

Tara N. Sainath, Oriol Vinyals, Andrew Senior, Haşim Sak

Google, Inc., New York, NY, USA
{tsainath, vinyals, andrewsenior, hasim}@google.com

ABSTRACT

Both Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) have shown improvements over Deep Neural Networks (DNNs) across a wide variety of speech recognition tasks. CNNs, LSTMs and DNNs are complementary in their modeling capabilities, as CNNs are good at reducing frequency variations, LSTMs are good at temporal modeling, and DNNs are appropriate for mapping features to a more separable space. In this paper, we take advantage of the complementarity of CNNs, LSTMs and DNNs by combining them into one unified architecture. We explore the proposed architecture, which we call CLDNN, on a variety of large vocabulary tasks, varying from 200 to 2,000 hours. We find that the CLDNN provides a 4-6% relative improvement in WER over an LSTM, the strongest of the three individual models.

1. INTRODUCTION

In the past few years, Deep Neural Networks (DNNs) have achieved tremendous success for large vocabulary continuous speech recognition (LVCSR) tasks compared to Gaussian Mixture Model/Hidden Markov Model (GMM/HMM) systems [1]. Recently, further improvements over DNNs have been obtained with alternative types of neural network architectures, including Convolutional Neural Networks (CNNs) [2] and Long-Short Term Memory Recurrent Neural Networks (LSTMs) [3]. CNNs, LSTMs and DNNs are individually limited in their modeling capabilities, and we believe that speech recognition performance can be improved by combining these networks in a unified framework.

A good overview of the modeling limitations of RNNs (and thus LSTMs) is provided in [4]. One issue with LSTMs is that the temporal modeling is done on the input feature x_t (i.e., log-mel feature). However, higher-level modeling of x_t can help to disentangle underlying factors of variation within the input, which should then make it easier to learn temporal structure between successive time steps [4]. For example, it has been shown that CNNs learn speaker-adapted/discriminatively trained features, and thus remove variation in the input [5]. Thus, it could be beneficial to proceed LSTMs with a few fully connected CNN layers.

In fact, state-of-the art GMM/HMM systems performing speaker adaptation, using techniques such as vocal tract length normalization (VTLN) and feature-space maximum likelihood linear regression (fMLLR), before performing temporal modeling via HMMs [6]. This recipe order has been shown to be appropriate for LVCSR tasks [7]. Therefore, it makes sense to explore passing the input x_t to CNN layers, which reduce variance in frequency of the input, before passing this to LSTM layers to model the sequence temporally.

In addition, as mentioned in [4], in LSTMs the mapping between h_t and output y_t is also not deep, meaning there is no intermediate nonlinear hidden layer. If factors of variation in the hidden states could be reduced, then the hidden state of the model could summarize the history of previous inputs more efficiently. In turn, this could make the output easier to predict. Reducing variation in the hidden states can be modeled by having DNN layers after the LSTM layers. This is similar in spirit to the hidden to output model proposed in [4], and also tested for speech, though with RNNs [8].

The model we propose is to feed input features, surrounded by temporal context, into a few CNN layers to reduce spectral variation. The output of the CNN layer is then fed into a few LSTM layers to reduce temporal variations. Then, the output of the last LSTM layer is fed to a few fully connected DNN layers, which transform the features into a space that makes that output easier to classify.

Combining CNN, LSTMs and DNNs has been explored in [9]. However, in that paper the three models were first trained separately and then the three outputs were combined through a combination layer. Our paper is different in that we combine CNNs, LSTMs and CNNs into one unified framework that is trained jointly. Furthermore, our choice of how to combine these layers is motivated by analysis in [4], which indicates that LSTM performance can be improved by providing better features to the LSTM (which the CNN layers provide through reducing spectral variance), as well as improving output predictions by making the mapping between hidden units and outputs deeper (which the DNN layers provide).

Each CNN, LSTM and DNN block captures information about the input representation at different scales [10]. Therefore, we explore if further improvements can be obtained by combining information at multiple scales. Specifically, we explore passing a long-term feature into the CNN, which is then passed into the LSTM along with a short-term feature. In addition, we explore the complementarity between the modeling capabilities of LSTM and DNN layers. Specifically, we investigate passing the output of the CNN layer into both LSTM and DNN layers. We will refer to the CLDNN architecture with these additional connections as a multi-scale CLDNN.

Our initial experiments to understand the behavior of the CLDNN are conducted on a 200 hour Voice Search task. We find that the CLDNN architecture provides an 4% relative improvement in WER over the LSTM, and including multi-scale features gives an additional 1% relative improvement. Next, we explore the behavior of the CLDNN architecture on 2 larger Voice Search tasks, namely a 2,000 hour clean-speech corpus, and 2,000 hour noisy-speech corpus. Here we find that the CLDNN provides between a 4-5% relative improvement in WER over the LSTM, and the multi-scale additions provide an additional 1% relative improvement. This demonstrates the robustness of the proposed CLDNN architecture with larger data sets and different environmental conditions.

The rest of this paper is as follows. In Section 2 we describe the CLDNN architecture, as well as the multi-scale additions. Experimental setup is described in Section 3 and initial experiments to

understand the CLDNN architecture are presented in Section 4. Results on the larger data sets are then discussed in Section 5. Finally, Section 6 concludes the paper and discusses future work.

2. MODEL ARCHITECTURE

This section describes the CLDNN architecture shown in Figure 1.

2.1. CLDNN

Frame x_t, surrounded by l contextual vectors to the left and r contextual vectors to the right, is passed as input to the network. This input is denoted as $[x_{t-l}, \ldots, x_{t+r}]$. In our work, each frame x_t is a 40-dimensional log-mel feature.

First, we reduce frequency variance in the input signal by passing the input through a few convolutional layers. The architecture used for each CNN layer is similar to that proposed in [2]. Specifically, we use 2 convolutional layers, each with 256 feature maps. We use a 9x9 frequency-time filter for the first convolutional layer, followed by a 4x3 filter for the second convolutional layer, and these filters are shared across the entire time-frequency space. Our pooling strategy is to use non-overlapping max pooling, and pooling in frequency only is performed [11]. A pooling size of 3 was used for the first layer, and no pooling was done in the second layer.

The dimension of the last layer of the CNN is large, due to the number of feature-maps$\times$time$\times$frequency context. Thus, we add a linear layer to reduce feature dimension, before passing this to the LSTM layer, as indicated in Figure 1. In [12] we found that adding this linear layer after the CNN layers allows for a reduction in parameters with no loss in accuracy. In our experiments, we found that reducing the dimensionality, such that we have 256 outputs from the linear layer, was appropriate.

After frequency modeling is performed, we next pass the CNN output to LSTM layers, which are appropriate for modeling the signal in time. Following the strategy proposed in [3], we use 2 LSTM layers, where each LSTM layer has 832 cells, and a 512 unit projection layer for dimensionality reduction. Unless otherwise indicated, the LSTM is unrolled for 20 time steps for training with truncated backpropagation through time (BPTT). In addition, the output state label is delayed by 5 frames, as we have observed with DNNs that information about future frames helps to better predict the current frame. The input feature into the CNN has l contextual frames to the left and r to the right, and the CNN output is then passed to the LSTM. In order to ensure that the LSTM does not see more than 5 frames of future context, which would increase the decoding latency, we set $r = 0$ for CLDNNs.

Finally, after performing frequency and temporal modeling, we pass the output of the LSTM to a few fully connected DNN layers. As shown in [5], these higher layers are appropriate for producing a higher-order feature representation that is more easily separable into the different classes we want to discriminate. Each fully connected layer has 1,024 hidden units.

2.2. Multi-scale Additions

The CNN takes a long-term feature, seeing a context of $t-l$ to t (i.e., $r = 0$ in the CLDNN), and produces a higher order representation of this to pass into the LSTM. The LSTM is then unrolled for 20 timesteps, and thus consumes a larger context of $20 + l$. However, we feel there is complementary information in also passing the short-term x_t feature to the LSTM. In fact, the original LSTM work in [3] looked at modeling a sequence of 20 consecutive short-term x_t

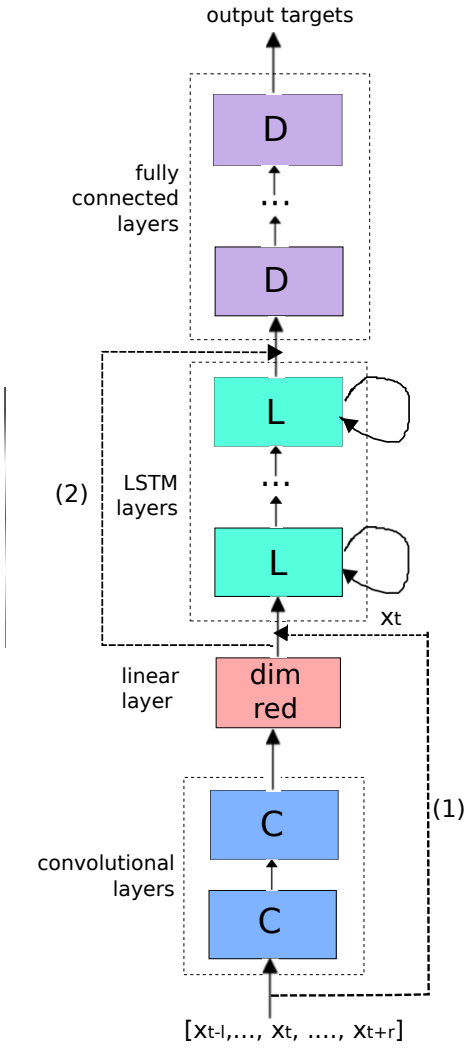

Fig. 1. CLDNN Architecture

features, with no context. In order to model short and long-term features, we take the original x_t and pass this as input, along with the long-term feature from the CNN, into the LSTM. This is shown by dashed stream (1) in Figure 1.

The use of short and long-term features in a neural network has been explored previously (i.e., [13, 14]). The main difference between previous work and ours is that we are able to do this jointly in one network, namely because of the power of the LSTM sequential modeling. In addition, our combination of short and long-term features results in a negligible increase in the number of network parameters.

In addition, we explore if there is complementarity between modeling the output of the CNN temporally with an LSTM, as well as discriminatively with a DNN. Specifically, motivated by work in computer vision [10], we explore passing the output of the CNN into both the LSTM and DNN. This is indicated by the dashed stream (2) in Figure 1. This idea of combining information from CNN and DNN layers has been explored before in speech [11, 15], though previous work added extra DNN layers to do the combination. Our work differs in that we pass the output of the CNN directly into the DNN, without extra layers and thus minimal parameter increase.

3. EXPERIMENTS

Our initial experiments to understand the CNN, DNN and LSTM architectures are conducted on a medium-sized training set consisting of 300k English-spoken utterances (about 200 hours). Further experiments are then performed on larger training set of 3m utterances (2,000 hrs). In addition, to explore the robustness of our model to noise, we also perform experiments using a noisy training set of 3m utterances (2,000 hrs). This data set is created by artificially corrupting clean utterances using a room simulator, adding varying degrees of noise and reverberation, such that the overall SNR is between 5dB to 30dB. The noise sources are from YouTube and daily life noisy environmental recordings. All training sets are anonymized and hand-transcribed, and are representative of Google's speech traffic. Models trained on clean speech are evaluated on a clean test set containing 30,000 utterances (20 hrs). In addition, models trained on noisy speech are evaluated in matched conditions on a 30,000 utterance noisy test set, to which noise at various SNRs has been added to the clean test set. It is important to note that the training and test sets used in this paper are different than those in [3], and therefore numbers cannot directly be compared.

The input feature for all models are 40-dimensional log-mel filterbank features, computed every 10ms. Unless otherwise indicated, all neural networks are trained with the cross-entropy criterion, using the asynchronous stochastic gradient descent (ASGD) optimization strategy described in [16]. The sequence-training experiments in this paper also use distributed ASGD, which is outlined in more detail in [17]. All networks have 13,522 CD output targets. The weights for all CNN and DNN layers are initialized using the Glorot-Bengio strategy described in [18]. Unless otherwise indicated, all LSTM layers are randomly initialized to be Gaussian, with a variance of $1/(\#\ \texttt{inputs})$. In addition, the learning rate is chosen specific to each network, and is chosen to be the largest value such that training remains stable. Learning rates are exponentially decayed.

4. RESULTS

Initial results to understand the combination CLDNN model, and its variants, are presented in this section. All models are trained on the medium-sized 200 hour clean training set, and results are reported on the clean test set.

4.1. Baselines

First, we establish baseline numbers for CNNs, DNNs and LSTMs, as shown in Table 1. Consistent with results reported in the literature [2], the CNN is trained with 2 convolutional layers with 256 feature maps, and 4 fully connected layers of 1,024 hidden units. The DNN is trained with 6 layers, 1,024 hidden units [1]. The input into both the CNN and DNN is a 40-dimensional log-mel filterbank feature, surrounded by a context of 20 past frames and 5 future frames. The LSTM is trained with 2 layers of 832 cells, and a 512 dimensional projection layer. Adding extra LSTM layers to this configuration was not found to help [3]. The input into the LSTM is a single 40-dimensional log-mel feature. The LSTM is unrolled 20 steps in time, and the output is delayed by 5 frames.

4.2. CNN+LSTM

In this section, we analyze the effect of adding CNN before the LSTM. To show the benefit of CNNs over DNNs, we also report results for adding the DNN before the LSTM layer. Table 2 compares the results for both CNNs and DNN, with different amounts of left input context (i.e., l) to the CNN and DNN. Notice that for both CNNs and DNNs, the best results are obtained by having a left context of 10 frames. A larger context of 20 hurts performance, likely since the LSTM is then unrolled for 20 time steps, so the total context processed by the LSTM is 40. Also notice that the benefits of CNNs over DNNs [2] continue to hold even when combined with LSTMs.

Method	WER
DNN	18.4
CNN	18.0
LSTM	18.0

Table 1. DNN, CNN, LSTM Baselines

Input Context	# Steps Unroll	WER CNN	WER DNN
l=0,r=0	20	17.8	18.2
l=10,r=0	20	**17.6**	18.2
l=20,r=0	20	17.9	18.5

Table 2. WER, CNN+LSTM vs. DNN+LSTM

To ensure that improvements with CLDNNs are not due to extra contextual features given to the CNN (and thus the LSTM), we explore the behavior of LSTMs with different temporal contexts. First, we provide the LSTM with an input spanning ten frames to the left of the current frame (i.e., $l = 10$), the same input feature provided to the CNN. The LSTM is still unrolled 20 timesteps. Table 3 shows that this does not improve WER over providing no feature context to the LSTM. In addition, we compare passing a single frame to the LSTM and unrolling it for 30 time steps, but this degrades WER. This helps to justify the gains from the CNN+LSTM architecture, showing the importance of extracting more robust features (with CNNs) before performing temporal modeling (with LSTMs).

Method	WER
LSTM, l=0, unroll=20	18.0
LSTM, l=10, unroll=20	18.0
LSTM, l=0, unroll=30	18.2

Table 3. WER, Alternative Temporal Modeling for LSTM

4.3. LSTM+DNN

In this section, we explore the effect of adding fully connected layers after the output of the LSTM. For this experiment, the input provided to each network is single frame x_t, and the LSTM is again unrolled for 20 time steps. Table 4 shows that improvements are obtained, but performance seems to saturate after two additional layers. This indicates that after temporal modeling is completed, it is beneficial to use DNN layers to transform the output of the LSTM layer to a space that is more discriminative and easier to predict output targets.

# DNN Layers	WER
0	18.0 (LSTM)
1	17.8
2	**17.6**
3	17.6

Table 4. WER, LSTM+DNN

4.4. CNN+LSTM+DNN

In this section, we put together the models from Sections 4.2 and 4.3, feeding features into a CNN, then performing temporal modeling with an LSTM, and finally feeding this output into 2 fully connected layers. Table 5 shows the WER for the LSTM, CNN+LSTM, LSTM+DNN and finally the combined CLDNN model. The table indicates that the gains from combining the CNN and DNN layers with the LSTM are complementary. Overall, we are able to achieve a 4% relative improvement in WER over the LSTM model alone.

Method	WER
LSTM	18.0
CNN+LSTM	17.6
LSTM+DNN	17.6
CLDNN	**17.3**

Table 5. WER, CLDNN

4.5. Better Weight Initialization

We have shown that we can achieve gains by using the CNNs to provide better features before performing temporal modeling with LSTMs. One may argue that if the LSTMs are better initialized, such that better temporal modeling can be performed, are CNNs really necessary? Our initial experiments with LSTMs use Gaussian random weight initialization, which produces eigenvalues of the initial recurrent network which are close to zero, thus increasing the chances for vanishing gradients [19]. To address this issue, we look at uniform random weight initialization between -0.02 to 0.02 for the LSTM layers.

Table 6 shows the gains with the CLDNN model still hold even after better weight initialization, and the CLDNN model still has a 4% relative improvement over the LSTM. This justifies the benefit of having CNN layers provide better features to do temporal modeling. Notice now that with proper weight initialization, the LSTM is better than the CNN or DNN in Table 1.

Method	WER - Gaussian Init	WER - Uniform Init
LSTM	18.0	17.7
CLDNN	17.3	**17.0**

Table 6. WER, Weight Initialization

4.6. Multi-scale Investigations

In this section, we investigate adding multi-scale information into the CLDNN architecture, as described in Section 2.2. First, we explore passing a long-term feature $[x_{t-10}, \ldots, x_t]$ to the CNN, and a short-term feature x_t to the LSTM. Table 7 shows this gives a WER of 16.8%, an additional 1% relative improvement over passing just the long-term feature from the CNN into the LSTM.

Second, we explore passing the output of the CNN into both the LSTM and the DNN. Table 7 indicates that this does not yield gains over the CLDNN alone. This indicates that temporal processing of CNN features using the LSTM is sufficient, and more information is not gained by additionally passing CNN features into the DNN.

5. RESULTS ON LARGER DATA SETS

In this section, we compare CLDNNs and LSTMs, as well as multi-scale additions, on larger data sets. Note when we say multi-scale CLDNN, we just include results passing short and long-term features into the CNN, and omit passing the CNN into both the LSTM and DNN, as only the first technique showed gains in Section 4.6. In addition, in this section we report numbers after both cross-entropy (CE) and sequence training [17], a strategy which has shown to give consistent gains over CE training [20].

Method	WER
LSTM	17.7
CLDNN, long-term feature to LSTM	17.0
+ short-term feature to LSTM	**16.8**
+ CNN to LSTM and DNN layers	17.0

Table 7. WER with Multi-scale Additions

Table 8 shows the WER for the 3 models when trained on a 2,000 hour clean data set, and then evaluated on a clean test set. With both the CLDNN and multi-scale additions, we can achieve a 6% relative reduction in WER over the LSTM after CE training, and a 5% relative improvement after sequence training.

Method	WER-CE	WER-Seq
LSTM	14.6	13.7
CLDNN	14.0	13.1
multi-scale CLDNN	**13.8**	**13.1**

Table 8. WER, Models Trained on 2,000 hours, Clean

Finally, Table 9 illustrates the WER for the 3 models when trained on a 2,000 hour noisy training set, and then evaluated on a noisy test set. At the CE level, the CLDNN provides a 4% relative reduction in WER compared to the LSTM, and including the multi-scale information again provides a small additional improvement. After sequence training, the CLDNN provides a 7% relative improvement over the LSTM. The improvements with CLDNNs on larger data sets and after sequence training demonstrate the robustness and value of the proposed method.

Method	WER-CE	WER-Seq
LSTM	20.3	18.8
CLDNN	19.4	**17.4**
multi-scale CLDNN	**19.2**	**17.4**

Table 9. WER, Models Trained on 2,000 hours, Noisy

6. CONCLUSIONS

In this paper, we present a combined CNN, LSTM and DNN architecture, which we call CLDNN. The architecture uses CNNs to reduce the spectral variation of the input feature, and then passes this to LSTM layers to perform temporal modeling, and finally outputs this to DNN layers, which produces a feature representation that is more easily separable. We also incorporate multi-scale additions to this architecture, to capture information at different resolutions. Results on a variety of LVCSR Voice Search tasks indicate that the proposed CLDNN architecture provides between an 4-6% relative reduction in WER compared to an LSTM.

7. ACKNOWLEDGEMENTS

Thank you to Izhak Shafran for help with LSTM training scripts, Hank Liao for help with decoding setups and Arun Narayanan for suggestions on how to train and decode in noisy conditions.

8. REFERENCES

[1] G. Hinton, L. Deng, D. Yu, G. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury, "Deep Neural Networks for Acoustic Modeling in Speech Recognition," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[2] T. N. Sainath, A. Mohamed, B. Kingsbury, and B. Ramabhadran, "Deep Convolutional Neural Networks for LVCSR," in *Proc. ICASSP*, 2013.

[3] H. Sak, A. Senior, and F. Beaufays, "Long Short-Term Memory Recurrent Neural Network Architectures for Large Scale Acoustic Modeling," in *Proc. Interspeech*, 2014.

[4] R. Pascanu, C. Gulcehre, K. Cho, and Y. Bengio, "How to Construct Deep Recurrent Neural Networks," in *Proc. ICLR*, 2014.

[5] A. Mohamed, G. Hinton, and G. Penn, "Understanding how Deep Belief Networks Perform Acoustic Modelling," in *ICASSP*, 2012.

[6] H. Soltau, G. Saon, and B. Kingsbury, "The IBM Attila speech recognition toolkit," in *Proc. IEEE Workshop on Spoken Language Technology*, 2010.

[7] T. N. Sainath, B. Ramabhadran, M. Picheny, D. Nahamoo, and D. Kanevsky, "Exemplar-Based Sparse Representation Features: From TIMIT to LVCSR," *in IEEE TSAP*, 2011.

[8] G. Saon, H. Soltau, A. Emami, and M. Picheny, "Unfolded Recurrent Neural Networks for Speech Recognition," in *Interspeech*, 2014.

[9] L. Deng and J. Platt, "Ensemble Deep Learning for Speech Recognition," in *Proc. Interspeech*, 2014.

[10] P. Sermanet and Y. LeCun, "Traffic sign recognition with multi-scale convolutional networks," in *Proc. IJCNN*, 2011.

[11] T. N. 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 *in Proc. ASRU*, 2013.

[12] T. N. Sainath, V. Peddinti, B. Kingsbury, P. Fousek, D. Nahamoo, and B. Ramabhadhran, "Deep Scattering Spectra with Deep Neural Networks for LVCSR Tasks," in *Proc. Interspeech*, 2014.

[13] P. Schwarz, P. Matejka, and J. Cernocky, "Hierarchical Structures of Neural Networks for Phoneme Recognition," in *Proc. ICASSP*, 2006.

[14] F. Grezl and M. Karafat, "Semi-Supervised Bootstrapping Approach for Neural Network Feature Extractor Training," in *Proc. ASRU*, 2013.

[15] H. Soltau, G. Saon, and T. N. Sainath, "Joint Training of Convolutional and Non-Convolutional Neural Networks," in *in Proc. ICASSP*, 2014.

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

[17] G. Heigold, E. McDermott, V. Vanhoucke, A. Senior, and M. Bacchiani, "Asynchronous Stochastic Optimization for Sequence Training of Deep Neural Networks," in *Proc. ICASSP*, 2014.

[18] X. Glorot and Y. Bengio, "Understanding the Difficulty of Training Deep Feedforward Neural Networks," in *Proc. AISTATS*, 2014.

[19] R. Pascanu, T. Mikolov, and Y. Bengio, "On the Difficulty of Training Recurrent Neural Networks," in *Proc. ICML*, 2013.

[20] B. Kingsbury, "Lattice-Based Optimization of Sequence Classification Criteria for Neural-Network Acoustic Modeling," in *Proc. ICASSP*, 2009.