BLENDING LSTMS INTO CNNS

Krzysztof J. Geras[1], Abdel-rahman Mohamed[2], Rich Caruana[2], Gregor Urban[3], Shengjie Wang[4], Özlem Aslan[5], Matthai Philipose[2], Matthew Richardson[2] & Charles Sutton[1]
[1]University of Edinburgh
[2]Microsoft Research
[3]UC Irvine
[4]University of Washington
[5]University of Alberta

ABSTRACT

We consider whether deep convolutional networks (CNNs) can represent decision functions with similar accuracy as recurrent networks such as LSTMs. First, we show that a deep CNN with an architecture inspired by the models recently introduced in image recognition can yield better accuracy than previous convolutional and LSTM networks on the standard 309h Switchboard automatic speech recognition task. Then we show that even more accurate CNNs can be trained under the guidance of LSTMs using a variant of model compression, which we call *model blending* because the teacher and student models are similar in complexity but different in inductive bias. Blending further improves the accuracy of our CNN, yielding a computationally efficient model of accuracy higher than any of the other individual models. Examining the effect of "dark knowledge" in this model compression task, we find that less than 1% of the highest probability labels are needed for accurate model compression.

1 INTRODUCTION

There is evidence that feedforward neural networks trained with the current training algorithms use their large capacity inefficiently (Le Cun et al., 1990; Denil et al., 2013; Dauphin & Bengio, 2013; Ba & Caruana, 2014; Hinton et al., 2015). Although this excess capacity may be necessary for accurate learning and generalization at training time, the function once learned often can be represented much more compactly. As deep neural net models become larger, their accuracy often increases, but the difficulty of deploying them also rises. Methods such as model compression sometimes allow the accurate functions learned by large, complex models to be compressed into smaller models that are computationally more efficient at runtime.

There are a number of different kinds of deep neural networks such as deep fully-connected neural networks (DNNs), convolutional neural networks (CNNs) and recurrent neural networks (RNNs). Different domains typically benefit from deep models of different types. For example, CNNs usually yield highest accuracy in domains such as image recognition where the input forms a regular 1, 2, or 3-D image plane with structure that is partially invariant to shifts in position or scale. On the other hand, recurrent network models such as LSTMs appear to be better suited to applications such as speech recognition or language modeling where inputs form sequences of varying lengths with short and long-range interactions of different scales.

The differences between these deep learning architectures raise interesting questions about what is learnable by different kinds of deep models, and when it is possible for a deep model of one kind to represent and learn the function learned by a different kind of deep model. The success of model compression on feedforward networks raises the question of whether other neural network architectures that embody different inductive biases can also be compressed. For example, can a classification function learnt by an LSTM be represented by a CNN with a wide enough window to span the important long-range interactions?

In this paper we demonstrate that, for a speech recognition task, it is possible to train very accurate CNN models, outperforming LSTMs. This is thanks to CNN architectures inspired by recent devel-

opments in computer vision (Simonyan & Zisserman, 2014), which were not previously considered for this task. Moreover, the experiments suggest that LSTMs and CNNs learn different functions when trained on the same data. This difference creates an opportunity: by merging the functions learned by CNNs and LSTMs into a single model we can obtain better accuracy than either model class achieved independently. One way to perform this merge is through a variant of model compression that we call *model blending* because the student and teacher models are of comparable size and have complementary inductive biases. For example, by blending an LSTM teacher with a CNN student, we are able to train a CNN that is more accurate (because it benefits from what the LSTM learned) and also computationally more efficient than an LSTM would be at runtime. This blending process is somewhat analogous to forming an ensemble of LSTMs and CNNs and then training a student CNN to mimic the ensemble, but in blending no explicit ensemble of LSTMs and CNNs need be formed. In practice the blended model is *6.8 times* more efficient at testing time than the analogous ensemble. Blending further improves the accuracy of our convolutional model, yielding a computationally efficient model of accuracy higher than any of the other individual models. Examining the effect of "dark knowledge" (Hinton et al., 2015) in this model compression task, we find that only 0.3% of the highest probability labels are needed during the model compression procedure.

2 BACKGROUND

Model compression (Bucila et al., 2006) is a machine learning technique which trains one model (a *student*) to mimic another model (a *teacher*). Typically, the student model is a small and the teacher is a larger, more powerful model, which has high accuracy but is computationally too expensive to use at test time.

For a classification task, this mimicry can be performed in two ways. One way is to train the student model to match logits (i.e. the values z_i in the output layer of the network, before applying the softmax to compute $p_i = e^{z_i} / \sum_j e^{z_j}$) predicted by the teacher on the training data, penalising the difference between logits of the two models with a squared loss. Alternatively, it can be done by training the student model to match class probabilities predicted by the teacher, by penalising cross-entropy between predictions $\mathbf{p}$ of the teacher (we will refer to them as *soft labels*) and predictions $\mathbf{q}$ of the student, i.e. by minimising $-\sum_i p_i(\mathbf{x}) \log q_i(\mathbf{x})$ averaged over training examples. In the context of deep neural networks, this approach to model compression is also known as *knowledge distillation* (Hinton et al., 2015). Additionally, the student can also be shown the original 0-1 hard labels of the training data.

The main advantage of training the student using model compression is that a student trained with knowledge provided by the teacher gets a richer supervision signal than just the hard 0-1 labels in the training data, i.e. for each training example, it gets the information not only about the correct class but also about uncertainty, i.e., how similar the current training example is to other classes. Model compression can be viewed as a way to transfer inductive biases between models. For example, in the case of compressing deep models into shallow ones (Ba & Caruana, 2014), the student is benefiting from the hierarchical representation learned in the deep model, despite not being able to learn it on its own from hard labels.

While model compression can be applied to arbitrary classifiers producing probabilistic predictions over target classes, with the recent success of deep neural networks fueled by new training techniques and dramatic increase in the computational power of modern computers, work on model compression focused on compressing large deep neural networks or ensembles thereof into smaller ones, i.e., with less layers, less hidden units or less parameters. In one paper pursuing that direction, Ba & Caruana (2014) showed that an ensemble of deep neural networks with few convolutional layers can be compressed into a single layer network as accurate as a deep one. In a complementary work, Hinton et al. (2015) focused on compressing ensembles of deep networks into deep networks of the same architecture. They also experimented with softening predictions of the teacher by dividing the logits by a constant greater than one called *temperature*. Using the techniques developed in prior work and adding an extra mimic layer in the middle of the student network, Romero et al. (2014) demonstrated that a moderately deep and wide convolutional network can be compressed into a deeper and narrower convolutional network with much fewer parameters than the teacher network while also increasing accuracy.

2.1 BIDIRECTIONAL LSTM

One example of very powerful neural network architecture yielding state-of-the-art performance on a range of tasks, yet expensive to run the test time, is the long short-term memory network (LSTM) (Hochreiter & Schmidhuber, 1997; Graves & Schmidhuber, 2005; Graves et al., 2013), which is a type of a recurrent neural network (RNNs). The focus of this work is to use this model as a teacher for model compression. The LSTM network used in our paper uses fixed-size input sequence of length T and only predicts the output for the middle item of the input sequence. In this respect, we follow the design proposed in the speech literature by Mohamed et al. (2015). Details of the LSTM used in this work can be found in the supplementary material attached to this paper.

LSTMs exhibit superior performance not only in speech. They work very well, for example, in handwriting recognition and generation (Graves & Schmidhuber, 2009; Graves, 2014), machine translation (Sutskever et al., 2014) and parsing (Vinyals et al., 2015), thanks to their ability to memorize long-term causes of current events. For acoustic modeling though, the difference between a non-recurrent network and an LSTM using full-length sequences is two fold: the use of longer context while deciding on the current frame label, and the type of processing in each cell (the LSTM cell compared to a sigmoid or ReLU). In this paper we use acoustic models that use the same context window as a non-recurrent network (limited to about 0.5 s) while using the LSTM cells for processing each frame. The LSTM cells process frames in the same bidirectional manner that any other bidirectional LSTM would do, but they are limited by the size of the contextual window.

One motivation to use such an architecture is that acoustic modeling labels (i.e. target states) are local in nature with an average duration of about 450 ms. Therefore, the amount of discriminant information decays rapidly as we move away from the target label. Long-term relations between labels, on the other hand, are taken care of using a language model (during testing) or a lattice of competing hypotheses in case of lattice training (Veselý et al., 2013; Kingsbury, 2009). Another motivation to prefer models that utilise limited input windows is faster convergence due to the ability to randomise samples on the frame level rather than on the utterance level. A practical benefit of using a fixed-length window is that using bidirectional architectures becomes possible in real time setups when the delay in response cannot be long.

3 VISION-STYLE CNNS FOR SPEECH RECOGNITION

Convolutional neural networks (LeCun et al., 1998) were considered for speech for a long time (LeCun & Bengio, 1998; Lee et al., 2009), though only recently have become very successful (Abdel-Hamid et al., 2012; Sainath et al., 2013; Abdel-Hamid et al., 2014; Sainath et al., 2015a). These CNN architectures are quite different from those used in computer vision. They use only two or three convolutional layers with large filters followed by more fully connected layers. They also only use convolution or pooling over one dimension, either time or frequency. When looking at a spectrogram in Figure 4, it is obvious that, like what we observe in vision, similar patterns re-occur both across different points in time and across different frequencies. Using convolution or pooling across only one of these dimensions seems suboptimal. One of the reasons for the success of CNNs is their invariance to small translations and scaling. Intuitively, small translations (corresponding to the pitch of voice) or scaling (corresponding to speaking slowly or quickly) should not change the class assigned to a window of speech. We hypothesise that classification of windows of speech with CNNs can be done more effectively with architectures similar to ones used in object recognition.

Looking at this problem through the lens of computer vision, we use a convolutional network architecture inspired by the work of Simonyan & Zisserman (2014). We only use small convolutional filters of size 3×3, non-overlapping 2×2 pooling regions and our network also has more layers than networks previously considered for the purpose of speech recognition. The same architecture is shared between both baseline and student networks (described in detail in Figure 1, contrasted to a widely applied architecture proposed by Sainath et al. (2013)).

4 COMBINING BIDIRECTIONAL LSTMS WITH VISION-STYLE CNNS

Both LSTMs and CNNs are powerful models. The mechanisms that guide their learning are quite different though. That creates an opportunity to combine their predictions, implicitly averaging their

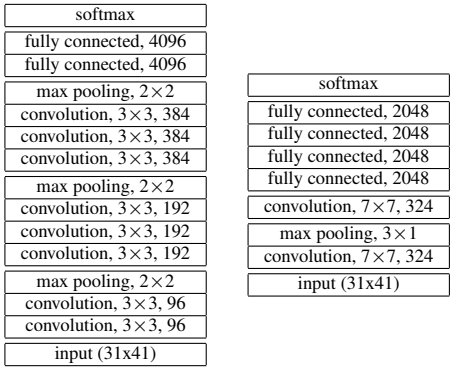

Figure 1: Left panel: configuration of the vision-style convolutional network used in our experiments. We used the stride of two pixels for max-pooling layers and one pixel for convolutional layers. We used no zero-padding in the first two convolutional layers and one pixel zero-padding for all remaining convolutional layers. Right panel: configuration of the convolutional network very closely resembling the one in the work of Sainath et al. (2013), adjusted to match the number of parameters in our network.

inductive biases. A classic way to perform this is ensembling. That is, to mix posterior predictions of the two models in the following manner:

$$p(y|\mathbf{x}_i) = \gamma p_{\text{LSTM}}(y|\mathbf{x}_i) + (1-\gamma)p_{\text{CNN}}(y|\mathbf{x}_i),$$

where $\gamma \in [0,1]$. $p_{\text{LSTM}}(y|\mathbf{x}_i)$ and $p_{\text{CNN}}(y|\mathbf{x}_i)$ denote probabilities of class y given a feature vector $\mathbf{x}_i$, respectively for the LSTM and the baseline CNN. As stated by Dietterich (2000): "a necessary and sufficient condition for an ensemble of classifiers to be more accurate than any of its individual members is if the classifiers are accurate and diverse". Our two types of models satisfy these conditions. Although ensembling is known to be very successful, it comes at the cost of executing all models at test time.

Alternatively, to combine the inductive biases of both the LSTM and the CNN, we can use a training objective that combines the loss function on the hard labels from the training data with a loss function which penalises deviation from predictions of the LSTM teacher. That is, we optimise

$$L(\lambda) = \lambda \left[-\sum_i \sum_c p(c|\mathbf{x}_i) \log q(c|\mathbf{x}_i) \right] + (1-\lambda) \left[-\sum_i \log q(y_i|\mathbf{x}_i) \right], \tag{1}$$

where $p(c|\mathbf{x}_i)$ is the probability of class c for training example $\mathbf{x}_i$ estimated by the teacher, $q(c|\mathbf{x}_i)$ is the probability of class c assigned to training example $\mathbf{x}_i$ by the student and y_i is the correct class for training example $\mathbf{x}_i$. The coefficient $\lambda \in [0,1]$ controls the weight of the errors on soft and hard labels in the objective. The value $\lambda = 0$ means the network is only learning using the hard labels, ignoring the teacher, while $\lambda = 1$ means that the networks is only learning from the soft labels provided by the teacher, ignoring the hard labels. When $\lambda \in (0,1)$ optimising the objective in Equation 1 yields a form of hydrid model which is learning using the guidance of the teacher, although not depending on it alone.

We motivate the choice of working directly with probabilities instead of logits (cf. section 2) in two ways. First, it is more direct to interpret retaining a subset of predictions of a network when considering probabilities (cf. Equation 2). We can simply look at the fraction of probability mass a subset of outputs covers. This will be necessary in our work (see section 5). Secondly, when using both soft and hard targets, it is easier to find an appropriate λ and learning rate, when the two objectives we are weighting together are of similar magnitudes and optimisation landscapes.

5 EXPERIMENTS

In the experiments we use the Switchboard data set (Godfrey et al., 1992), which consists of 309 hours of transcribed speech. We used 308 hours as training set and kept one hour as a validation set. The data set is segmented into 248k utterances, i.e. continuous pieces of speech beginning and ending with a clear pause. Each utterance consists of a number of frames, i.e. 25 ms intervals of speech, with a constant shift of 10 ms. For every frame, the extracted features are 31-channel Mel-filterbank parameters passed through a 10-th root nonlinearity. Features for one utterance are visualised in Figure 4. To form our training and validation sets we extract windows of 41 frames, that is, the frame whose label we want to predict, 20 frames before it and 20 frames after it. As

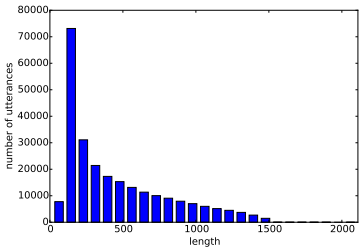

Figure 2: Left panel: distribution of the lengths (numbers of frames) of utterances in the training data. Right panel: fraction of class probability covered as a function of the number of most likely classes included. Dashed line indicates what the curve would look like if the labels had a uniform distribution. The distribution is very non-uniform.

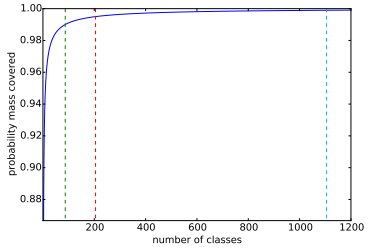

Figure 3: Average fraction of probability mass covered as a function of the number of most probable classes retained. Dashed lines indicate the number of classes necessary to cover 99%, 99.5% and 99.9% of probability mass. Clearly, only very few top classes in the total of 9000 are necessary to cover a large majority of probability mass.

shown in Figure 2, distribution of the lengths of utterances is highly non-uniform, therefore to keep the sampling unbiased, we sample training examples by first sampling an utterance proportionally to its length and then sampling a window within that utterance uniformly. To form the validation set we simply extract all possible windows. In both cases, we pad each utterance with zeros at the beginning and at the end so that every frame in each utterance can be drawn as a middle frame. Every frame in the training and validation set has a label. The 9000 output classes represent tied tri-phone states that are generated by an HMM/GMM system (Young et al., 1994). Forced alignment is used to map each input frame to one output state using a baseline DNN model. The distribution of classes in the training data is visualised in Figure 2. We call the frame classification error on the validation set frame error rate (FER).

The test set is a part of the standard Switchboard benchmark (the Hub5'00 SW data set). It was sampled from the same distribution as the training set and consists of 1831 utterances. There are no frame-level labels in the test set and the final evaluation is based on the ability to predict words in the test utterances. To obtain the words predicted by the model, frame label posteriors generated from the neural network are first divided by their prior probabilities then passed to a finite state transducer-based decoder to be combined with 3-gram language model probabilities to generate the most probable word sequences. Hypothesized word sequences are aligned to the human reference transcription to count the number of word insertions (I), deletions (D), and substitutions (S). Word Error Rate (WER) is defined as $\text{WER} = \frac{S+D+I}{N} \times 100$, where N is the total number of words in the reference transcription.

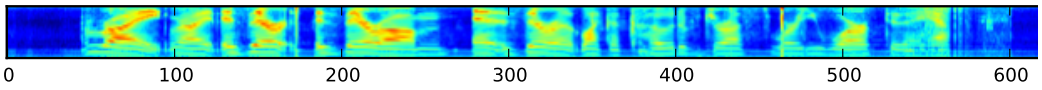

Figure 4: Example of an utterance in the Switchboard data set. The horizontal axis is over time and the vertical axis is over frequencies.

Since the teacher model is very slow at prediction time, it is impractical to run a large number of experiments if the teacher must repeatedly be executed to train each student. Unfortunately, running the teacher once and saving its predictions to disk is also problematic — because the output space is large, storing the soft labels for all classes would require a large amount of space ($\approx$ 3.6 TB). Moreover, to sample each minibatch in an unbiased manner, we would need constant random access to disk, which again would make training very slow. To deal with that problem we save predictions only for the small subset of classes that receive the highest prediction probabilities. To determine whether this is a viable solution, we checked what percentage of the total probability mass, averaged over the examples in the training set, is covered by the C most likely classes according to the teacher

model. We denote that quantity $\mathrm{TOP}_C(\mathbf{x})$. That is, we compute

$$\mathrm{M}(C) = \frac{1}{|\{\mathbf{x}_i\}|} \sum_{\mathbf{x}_i} \sum_{y \in \mathrm{TOP}_C(\mathbf{x}_i)} p(y|\mathbf{x}_i), \tag{2}$$

where $p(y|\mathbf{x})$ denotes the posterior probability of class y given a feature vector $\mathbf{x}$. This relationship for one of our LSTM models is shown in Figure 3. We found that, with very few exceptions, posteriors over classes are concentrated on very few values. Therefore, we decided to continue our experiments retaining top classes covering not more than 90 classes for each training example, cutting off after covering 99% of the probability mass. This allows us to store soft labels for the entire data set in the RAM, making unbiased sampling of training data efficient.

5.1 BASELINE NETWORKS

We used Lasagne, which is based on Theano (Bergstra et al., 2010; Bastien et al., 2012), for implementing CNNs. We used the architecture described in Figure 1.

For training of the CNN baseline we used a batch-size of 256 randomly drawn examples. Each epoch consisted of 2000 mini-batches. Hyper-parameters of the baseline networks were: initial learning rate (1.7×10^{-2}), momentum coefficient (0.9, we used Nesterov's momentum) and a constant learning rate decay coefficient (0.7). Because the data set we used is very large (309 hours, 18 GB), the only form of regularisation we used was early stopping in the following form. After every epoch we measured the loss (negative log-likelihood of correct class) on the validation set. If the loss did not improve for five epochs, we multiplied the learning rate by a constant learning rate decay coefficient. We stopped the training after the learning rate was smaller than 5×10^{-5}. It took about 200 epochs to finish the training. We repeated training with three different random seeds.

For comparison, we also trained a CNN very similar to the one proposed by Sainath et al. (2013), adjusted to match the number of parameters in our network. We used the same training procedure and hyperparameters.

The bidirectional LSTM teacher networks we used are very similar to the one in the work of Mohamed et al. (2015). We trained three models, all with four hidden layers, two with 512 hidden units for each direction and one with 800 hidden units for each direction. The training starts with the learning rate equal to 0.05 for one of the smaller models and 0.08 for the other ones. We used the standard momentum with the coefficient of 0.9. After three epochs of no improvement of frame error rate on the validation set, learning rate was multiplied by $\frac{2}{3}$ and training process was rolled back to the last epoch after which validation error improved. Training stopped when learning rate was smaller than 10^{-5}. It took about 75 epochs (2000 minibatches of 256 samples) to finish the training.

The results for these models are shown in the upper part of Table 2. Our vision-style CNN achieved 14.1 WER (averaged over three random seeds), the larger LSTM achieved 14.4 WER, and a CNN of an architecture proposed by Sainath et al. (2013) achieved 15.5 WER. Interestingly, although our LSTM teachers outperform our vision-style CNN trained with hard labels in terms of FER, the results in WER, which is the metric of primary interest, are the opposite. This discrepancy between FER and WER has been observed in the speech community before, for example by Sak et al. (2014) for LSTMs and DNNs. FER and WER are not always perfectly correlated because FER is conditioned on a baseline model that generated the frame alignment in the first place (which might not be correct for all the cases). FER also penalizes misclassifications of boundary frames (e.g. shift in the start/end times of certain target states) which might not be of importance as long as the correct target state is recognized. On the other hand, WER is calculated taking into account information about neighbouring frames (i.e. smoothness) as well as external knowledge (e.g. a language model) which corrects many of the misclassifications made locally.

5.2 ENSEMBLES OF NETWORKS

The first approach we use to combine the two types of models is to create ensembles. The results in Figure 5 and Figure 6 indicate that for the problem we consider it is beneficial to combine neural networks from different families, which have different inductive biases. Even though CNNs are much weaker in FER, combining an LSTM with a CNN achieves the same FER as an ensemble

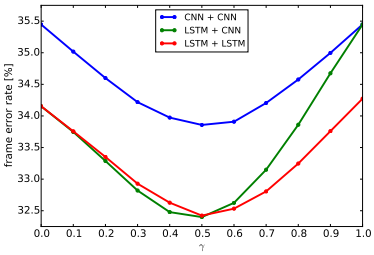

Figure 5: Frame error rate of ensembles as a function of γ.

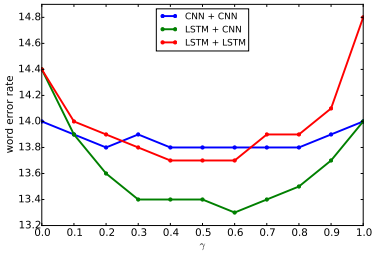

Figure 6: Word error rate of ensembles as a function of γ.

Table 1: Average fraction of probability mass covered and average number of classes retained after truncating predictions of an ensemble of two LSTMs to fit them in the memory.

maximum number of classes retained	**1**	**3**	**10**	**30**	**90**
average fraction of probability mass covered	73.20%	91.19%	96.68%	98.38%	99.03%
average number of classes retained	1.0	2.89	6.79	13.33	23.96

of two LSTMs (both of which are more accurate than the CNN), and actually yields better WER than ensembles of two CNNs or two (superior) LSTMs. Interestingly, ensembling two CNNs yields almost no benefit in WER. To complete the picture we tried ensembles with more than two models. An ensemble of three LSTMs achieved 31.98% FER and 13.7 WER, an ensemble of three CNNs achieved 33.31% FER and 13.9 WER, thus, in both cases, yielding very little gain over ensembles of two models of these types. On the other hand adding a CNN to the ensemble of two LSTMs yielded 31.57% FER and 13.2 WER. The benefits of adding more models to the ensemble appear to be negligible if the ensemble contained at least one model of each type already.

We also compared the errors made by CNNs and LSTMs to see if the models are qualitatively different. We observe that a CNN tends to make similar errors as other CNNs, an LSTM tends to make similar errors as other LSTMs, but CNNs and LSTMs tend to make errors that are less similar to each other than the CNN-CNN and LSTM-LSTM comparisons.

Although the ensembles we trained are very effective in terms of WER, it comes at the cost of a large increase of computation necessary at test time compared to the baseline CNN. Because of the cost of the LSTMs, our best two-model ensemble is about seven times slower than our baseline vision-style CNN (cf. Table 2).

5.3 Networks trained with model compression

Table 2: Frame error rates and word error rates for our models. The numbers for the vision-style CNNs and LSTM→CNN blending are averages over three random seeds. The numbers for smaller LSTM are given for a better of the two on FER, the other one achieved 34.71% FER and the same WER (14.8).

	FER	**WER**	model size	execution time
Sainath et al. (2013)-style CNN	37.93%	15.5	$\approx$ 75M	$\times$ 0.75
vision-style CNN	35.51%	14.1	$\approx$ 75M	$\times$ 1.0
smaller LSTM	34.27%	14.8	$\approx$ 30M	$\times$ 3.3
bigger LSTM	34.15%	14.4	$\approx$ 65M	$\times$ 5.8
LSTM + CNN ensemble ($\gamma = 0.5$)	32.4%	13.4	$\approx$ 130M	$\times$ 6.8
LSTM $\rightarrow$ CNN blending ($\lambda = 0.75$)	34.11%	13.83	$\approx$ 75M	$\times$ 1.0

The next question we tackle is whether it is possible to achieve an effect similar to creating an ensemble without having to execute all models at prediction time. We attack this with model blending via model compression. To do this, we took predictions of the two best performing LSTMs and averaged their predictions to form a teacher model. Such a teacher model achieves 32.4% FER and 13.4 WER. As we mentioned earlier, it is infeasible to use all predictions during training. There-

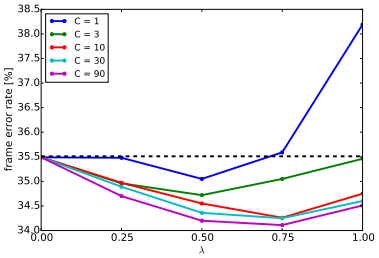

Figure 7: FER of the CNN student as a function of λ as C, the maximum of number of teacher outputs retained, varies from 1 to 90. Dashed line indicates performance of a baseline CNN.

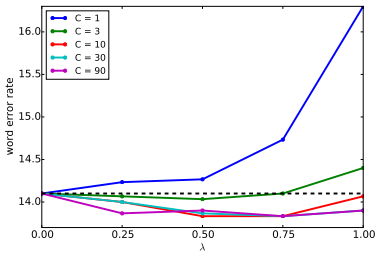

Figure 8: WER of the CNN student as a function of λ as C, the maximum of number of teacher outputs retained, varies from 1 to 90. Dashed line indicates performance of a baseline CNN.

fore we only store a subset of classes predicted by the teacher for each frame. Table 1 shows what fraction of probability mass is covered when storing different maximum number of predictions (C). It is particularly interesting to understand how many predictions of the teacher model are sufficient to achieve good performance to test the *dark knowledge* hypothesis (Hinton et al., 2015) which states that information about classes predicted with low probability is important to the success of model compression. We use $C \in \{90, 30, 10, 3, 1\}$. We also vary the parameter λ which controls how much the student is learning from the teacher and how much it is learning from hard labels. The architecture and training procedure for the students is the same as for the baseline. For every combination of C and λ we report an average over three random seeds.

The results are shown in Figure 7 and Figure 8. The best model achieved lower FER (34.11%) and lower WER (13.83) than any of the individual models. For all numbers of teacher predictions retained (C) the best performance was achieved for $\lambda \in \{0.25, 0.5, 0.75\}$. That highlights the importance of blending the knowledge extracted from the teacher model with learning from hard labels within the architecture of the student.

Our experiments show that, at least for the task we are considering, it is not critical to use predictions for all classes provided by the teacher. Just 30 most likely predictions ($\frac{1}{300}$ of all classes!) is enough. Bringing the number up to 90 classes did not improve performance of the student in terms of WER. However, performance deteriorates dramatically when too few predictions of the teacher are used suggesting that some dark knowledge is needed.

6 Related work

A few papers have applied model compression in similar speech recognition settings. The most similar to our work is the work of Chan et al. (2015) who compressed LSTMs into small networks without convolutional layers. Using the soft labels from an LSTM, they were able to show an improvement in WER over the baseline trained with hard labels. The main difference between this work and ours is that their student networks are non-convolutional and tiny. While this allows for a decent improvement over the baseline, since the student network is much smaller, its performance is still much weaker than performance of a single model of the same type as the teacher. Hence, this work addresses a different question than we do, i.e. whether a network without recurrent structure can perform as well or better as an LSTM when using soft labels provided by the LSTM. Model compression was also successfully applied to speech recognition by Li et al. (2014) who used deep neural networks without convolutional layers both as a teacher and a student. The architecture of the two networks was the same except that the student had less hidden units in each layer. Finally, work in the opposite direction was done by Wang et al. (2015) and Tang et al. (2015). They demonstrated that when using a small data set for which an LSTM is overfitting, a deep non-convolutional network can provide useful guidance for the LSTM. It can come either in the form of pre-training the LSTM with soft labels from a DNN or training the LSTM optimising a loss mixing hard labels with soft labels from a DNN.

7 DISCUSSION AND FUTURE WORK

The main contribution of this paper is introducing the use of model compression in a previously unexplored setting where both the teacher and student architectures are powerful ones, yet designed with different inductive biases. Thus, rather than calling it model compression we use the term "model blending".

We showed that the LSTM and the CNN learn different kinds of knowledge from the data which can be leveraged through simple ensembling or model blending via model compression. Success of the combination of the two models, provides additional motivation for the study of neural network architectures which use concepts from both recurrent networks and convolutional networks, e.g. CLDNN (Sainath et al., 2015b).

Finally, we provided experimental evidence that CNNs of appropriate vision-style architecture have the necessary capacity to learn accurate predictors on large speech data sets and gave a simple, practical recipe for improving the performance of CNN-based speech recognition models even further at no cost during test time. We hypothesise that the very recent advances in training even deeper convolutional networks for computer vision (Srivastava et al., 2015; He et al., 2015) will yield improved performance in speech recognition and would further improve our results.

ACKNOWLEDGMENTS

We thank Stanisław Jastrzębski for suggesting the name of the paper.

REFERENCES

Ossama Abdel-Hamid, Abdel-rahman Mohamed, Hui Jiang, and Gerald Penn. Applying convolutional neural networks concepts to hybrid NN-HMM model for speech recognition. In *ICASSP*, 2012.

Ossama Abdel-Hamid, Abdel-rahman Mohamed, Hui Jiang, Li Deng, Gerald Penn, and Dong Yu. Convolutional neural networks for speech recognition. *TASLP*, 22, 2014.

Jimmy Ba and Rich Caruana. Do deep nets really need to be deep? In *NIPS*, 2014.

Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, James Bergstra, Ian J. Goodfellow, Arnaud Bergeron, Nicolas Bouchard, and Yoshua Bengio. Theano: new features and speed improvements. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012.

James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: a CPU and GPU math expression compiler. In *SciPy*, 2010.

Cristian Bucila, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In *KDD*, 2006.

William Chan, Nan Rosemary Ke, and Ian Laner. Transferring knowledge from a RNN to a DNN. *arXiv:1504.01483*, 2015.

Yann Dauphin and Yoshua Bengio. Big neural networks waste capacity. *arXiv:1301.3583*, 2013.

Misha Denil, Babak Shakibi, Laurent Dinh, and Nando de Freitas. Predicting parameters in deep learning. In *NIPS*, 2013.

Thomas G. Dietterich. Ensemble methods in machine learning. In *MCS*, 2000.

Felix A. Gers, Nicol N. Schraudolph, and Jürgen Schmidhuber. Learning precise timing with lstm recurrent networks. *JMLR*, 3, 2003.

John J. Godfrey, Edward C. Holliman, and Jane McDaniel. Switchboard: telephone speech corpus for research and development. In *ICASSP*, 1992.

Alex Graves. *Supervised sequence labelling with recurrent neural networks*. 2012.

Alex Graves. Generating sequences with recurrent neural networks. *arXiv:1308.0850*, 2014.

Alex Graves and Jürgen Schmidhuber. Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Networks*, 18(5-6), 2005.

Alex Graves and Jürgen Schmidhuber. Oine handwriting recognition with multidimensional recurrent neural networks. In *NIPS*, 2009.

Alex Graves, Abdel-rahman Mohamed, and Geoffrey E. Hinton. Speech recognition with deep recurrent neural networks. In *ICASSP*, 2013.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv:1512.03385*, 2015.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv:1503.02531*, 2015.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8), 1997.

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

Yann Le Cun, John S. Denker, and Sara A. Solla. Optimal brain damage. In *NIPS*, 1990.

Yann LeCun and Yoshua Bengio. Convolutional networks for images, speech, and time series. In *The Handbook of Brain Theory and Neural Networks*. 1998.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 1998.

Honglak Lee, Peter Pham, Yan Largman, and Andrew Y. Ng. Unsupervised feature learning for audio classification using convolutional deep belief networks, 2009.

Jinyu Li, Rui Zhao, Jui-Ting Huang, and Yifan Gong. Learning small-size dnn with output-distribution-based criteria. In *INTERSPEECH*, 2014.

Abdel-rahman Mohamed, Frank Seide, Dong Yu, Jasha Droppo, Andreas Stolcke, Geoffrey Zweig, and Gerald Penn. Deep bidirectional recurrent networks over spectral windows. In *ASRU*, 2015.

Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. Fitnets: Hints for thin deep nets. In *ICLR*, 2014.

Tara Sainath, Abdel-rahman Mohamed, Brian Kingsbury, and Bhuvana Ramabhadran. Deep convolutional neural networks for LVCSR. In *ICASSP*, 2013.

Tara 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*, 64, 2015a.

Tara Sainath, Oriol Vinyals, Andrew Senior, and Hasim Sak. Convolutional, long short-term memory, fully connected deep neural networks. In *ICASSP*, 2015b.

Hasim Sak, Andrew W. Senior, and Françoise Beaufays. Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition. *arXiv:1402.1128*, 2014.

Mike Schuster and Kuldip K. Paliwal. Bidirectional recurrent neural networks. *TSP*, 45(11), 1997.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2014.

Rupesh K. Srivastava, Klaus Greff, and Jürgen Schmidhuber. Training very deep networks. In *NIPS*, 2015.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *NIPS*, 2014.

Zhiyuan Tang, Dong Wang, Yiqiao Pan, and Zhiyong Zhang. Knowledge transfer pre-training. *arXiv:1506.02256*, 2015.

Karel Veselý, Arnab Ghoshal, Lukáš Burget, and Daniel Povey. Sequence discriminative training of deep neural networks. In *INTERSPEECH*, 2013.

Oriol Vinyals, Lukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey Hinton. Grammar as a foreign language. In *NIPS*, 2015.

Dong Wang, Chao Liu, Zhiyuan Tang, Zhiyong Zhang, and Mengyuan Zhao. Recurrent neural network training with dark knowledge transfer. *arXiv:1505.04630*, 2015.

S. J. Young, J. J. Odell, and P. C. Woodland. Tree-based state tying for high accuracy acoustic modelling. In *HLT*, 1994.

SUPPLEMENTARY MATERIAL

DETAILS OF THE LSTM

Given a sequence of input vectors $\boldsymbol{x} = (x_1, \ldots, x_T)$, an RNN computes the hidden vector sequence $\boldsymbol{h} = (h_1, \ldots, h_T)$ by iterating the following from $t = 1$ to T:

$$h_t = \mathcal{H}\left(W_{xh}x_t + W_{hh}h_{t-1} + b_h\right).$$

In our work, we compute the output only for the middle item in the input sequence. It is computed as $y_{t^*} = W_{hy}h_{t^*} + b_y$.

The W terms denote weight matrices (e.g. W_{xh} is the input-hidden weight matrix), the b terms denote bias vectors (e.g. b_h is hidden bias vector) and $\mathcal{H}$ is the hidden layer function.

While there are multiple possible choices for $\mathcal{H}$, prior work (Graves, 2012; Graves et al., 2013; Sak et al., 2014) has shown that the LSTM architecture, which uses purpose-built *memory cells* to store information, is better at finding and exploiting longer context. Figure 9 illustrates a single LSTM memory cell. For the version of the LSTM cell used in this paper (Gers et al., 2003) $\mathcal{H}$ is implemented by the following composite function:

$$\begin{aligned}
i_t &= \sigma\left(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i\right), \\
f_t &= \sigma\left(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f\right), \\
c_t &= f_t c_{t-1} + i_t \tanh\left(W_{xc}x_t + W_{hc}h_{t-1} + b_c\right), \\
o_t &= \sigma\left(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o\right), \\
h_t &= o_t \tanh(c_t),
\end{aligned}$$

where σ is the logistic sigmoid function, and i, f, o and c are respectively the *input gate*, *forget gate*, *output gate* and *cell* activation vectors, all of which are the same size as the hidden vector h. The weight matrices from the cell to gate vectors (e.g. W_{ci}) are diagonal, so element m in each gate vector only receives input from element m of the cell vector.

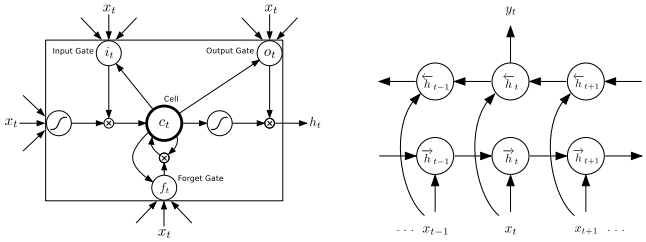

Figure 9: Left panel: the LSTM cell. Figure from Graves et al. (2013). Right panel: bidirectional RNN. Figure based on Figure 2 from Graves et al. (2013).

One shortcoming of conventional RNNs is that they are only able to make use of previous context. Bidirectional RNNs (BRNNs) (Schuster & Paliwal, 1997) exploit past and future context by processing the data in both directions with two separate hidden layers, which are then fed forwards to the same output layer. As illustrated in Figure 9, a BRNN computes the *forward* hidden sequence $\overrightarrow{\boldsymbol{h}} = (\overrightarrow{h}_1, \ldots, \overrightarrow{h}_T)$ and the *backward* hidden sequence $\overleftarrow{\boldsymbol{h}} = (\overleftarrow{h}_1, \ldots, \overleftarrow{h}_T)$ by iterating from $t = 1$ to T:

$$\begin{aligned}
\overrightarrow{h}_t &= \mathcal{H}\left(W_{x\overrightarrow{h}}x_t + W_{\overrightarrow{h}\overrightarrow{h}}\overrightarrow{h}_{t-1} + b_{\overrightarrow{h}}\right), \\
\overleftarrow{h}_t &= \mathcal{H}\left(W_{x\overleftarrow{h}}x_t + W_{\overleftarrow{h}\overleftarrow{h}}\overleftarrow{h}_{t+1} + b_{\overleftarrow{h}}\right).
\end{aligned}$$

The output is computed as $y_{t^*} = W_{\overrightarrow{h}y}\overrightarrow{h}_{t^*} + W_{\overleftarrow{h}y}\overleftarrow{h}_{t^*} + b_y$. Combing BRNNs with LSTM gives bidirectional LSTM, which can access the context in both directions.

Finally, deep RNNs can be created by stacking multiple RNN hidden layers on top of each other, with the output sequence of one layer forming the input sequence for the next. Assuming the same

hidden layer function is used for all N layers in the stack, the hidden vector sequences $\boldsymbol{h}^n$ are iteratively computed from $n = 1$ to N and $t = 1$ to T:

$$h_t^n = \mathcal{H}\left(W_{h^{n-1}h^n} h_t^{n-1} + W_{h^n h^n} h_{t-1}^n + b_h^n\right),$$

where we define $\boldsymbol{h}^0 = \boldsymbol{x}$. The network output y_{t^*} is $y_{t^*} = W_{h^N y} h_{t^*}^N + b_y$.

Deep bidirectional RNNs can be implemented by replacing each hidden sequence $\boldsymbol{h}^n$ with the forward and backward sequences $\overrightarrow{\boldsymbol{h}}^n$ and $\overleftarrow{\boldsymbol{h}}^n$, and ensuring that every hidden layer receives input from both the forward and backward layers at the level below. If bidirectional LSTMs are used for the hidden layers we get deep bidirectional LSTMs, the architecture we use as a teacher network in this paper.