Model compression applied to small-footprint keyword spotting

George Tucker[1*], *Minhua Wu*[2], *Ming Sun*[3],
Sankaran Panchapagesan[2], *Gengshen Fu*[3], *Shiv Vitaladevuni*[3]

[1]Google Brain, Mountain View, CA, USA
[2]Lab126, Sunnyvale, CA, USA
[3]Amazon.com, Cambridge, MA, USA

gjt@google.com, {wuminhua,mingsun,gengshef,shivnaga}@amazon.com, panchi@lab126.com

Abstract

Several consumer speech devices feature voice interfaces that perform on-device keyword spotting to initiate user interactions. Accurate on-device keyword spotting within a tight CPU budget is crucial for such devices. Motivated by this, we investigated two ways to improve deep neural network (DNN) acoustic models for keyword spotting without increasing CPU usage. First, we used low-rank weight matrices throughout the DNN. This allowed us to increase representational power by increasing the number of hidden nodes per layer without changing the total number of multiplications. Second, we used knowledge distilled from an ensemble of much larger DNNs used only during training. We systematically evaluated these two approaches on a massive corpus of far-field utterances. Alone both techniques improve performance and together they combine to give significant reductions in false alarms and misses without increasing CPU or memory usage.

Index Terms: keyword spotting, model compression

1. Introduction

Increasingly, mobile and smart home devices offer fully voice-based interfaces, such as Google Now on Android, Siri on iPhone 6s, and Alexa on the Amazon Echo. While some devices are beginning to offer limited speech recognition entirely on-device, the majority of devices stream audio to the cloud for recognition. For privacy reasons, these devices rely on the user to preface their commands with a keyword, such as "Alexa". For this application, accurate on-device keyword spotting is crucial to usability.

In the past, keyword/filler Hidden Markov Model-Gaussian Mixture Models were commonly used. However, they have largely been replaced with better performing deep neural network (DNN) based keyword spotting systems (KWS). For example, Chen et al. described a state-of-the-art DNN based KWS that used a simple thresholding based decoder [1]. Recently, several RNN based KWS have been proposed [2, 3, 4, 5] that can leverage longer temporal context to improve performance over fixed-window DNN systems. However, DNN based KWS strike a balance between performance and operational simplicity. So in this work, we focused on applying model compression techniques to the DNN KWS system described in [1], although our methods also apply to RNN based KWS and multisystem cascaded approaches (e.g., [6]).

For the KWS application, test time computational efficiency is crucial, whereas model training time is largely irrelevant (within reason) due to the availability of powerful GPUs for training. Numerous approaches have been proposed for compressing models at test time in image classification and speech recognition. Tying weights with random hash functions [7] and quantizing blocks of weights with vector quantization [8] both provide significant reductions in memory usage. However, systematic evaluation of their effect on CPU usage has not been explored. Alternatively, by using structured weight matrices (e.g., low tensor rank or Toeplitz), we can perform the DNN forward pass more efficiently. This approach has shown significant success for both CNNs in image classification and DNNs in speech recognition [9, 10, 11, 12]. Finally, we can train a small "student" model to mimic a much larger "teacher" model using soft targets generated from the teacher model [13, 14]. Previous work on model compression for KWS used low-rank input-to-hidden weight matrices to reduce memory usage [15], however, this did not reduce CPU usage.

In this work, we focused on improving KWS performance without increasing CPU or memory usage. In particular, we investigated the use of low-rank weight matrices [16, 17, 18] and knowledge distillation [13, 14] applied to the DNN based KWS system described in [1]. We demonstrate that both of these techniques alone improve performance and together they combine to give significant reductions in false alarms (FAs) and misses ($\approx 20\%$ relative reduction in miss rate over a wide range of FA rates) with comparable CPU and memory usage to the baseline.

In section 2, we describe the baseline KWS system used in all of our experiments, describe our training recipe for low-rank weight matrices, and review knowledge distillation. In section 3, we report on a systematic evaluation of the proposed approaches. Finally, we end with conclusions and future work.

2. Small-Footprint Keyword Spotting

In many cases we have access to powerful GPUs during training, but at test time, the model must run in a resource constrained environment. Because our focus is on model compression techniques and not the specific KWS implementation, we based our experiments on the state-of-the-art DNN KWS system previously described in [1]. However, we expect that the results will generalize to other DNN or RNN based KWS systems. We note that this is a different system than the production KWS used in Amazon Echo. For completeness, we briefly review the DNN KWS from [1].

The DNN KWS can be roughly broken down into three components: feature extraction, classification, and posterior smoothing. First, we extracted 20 dimensional log-mel filterbank energies over 25 ms frames with a 10 ms frame shift. The

[*] Work conducted while the author was at Amazon.com

input to the DNN was 20 left context frames, a middle frame, and 10 right context frames stacked (620 dimensional input). We used an asymmetric context because longer right context increases latency.

As the baseline DNN, we used a DNN with 4 hidden layers, 248 nodes per layer, and sigmoid units. Preliminary experiments with rectified linear units did not show significant differences, so we used sigmoid units in all other experiments. The DNN had a binary output indicating the presence or absence of the keyword (e.g., “Alexa”) at the middle frame (i.e., most targets are background except for the ≈70 frame targets from each keyword). We found that adding a simultaneous additional ASR label task improved performance (see [20] for details). So, we used this in all experiments except those with knowledge distillation, where it was unnecessary as long as the teacher model was trained with the multi-task objective. Notably, the DNN is trained to optimize a framewise cross-entropy loss, whereas the detection task is truly a sequence level task, however, the two are highly correlated. We trained the DNN using distributed asynchronous SGD [19, 20]. We used a performance based learning rate schedule (similar to “newbob”), where the learning rate is halved every time performance degrades on the development set. The initial learning rate and batch size were tuned on the development set. Preliminary experiments with adaptive per weight learning rate algorithms ADADELTA [21] and RMSProp [22] with the baseline architecture did not improve performance, so for all other experiments, we did not use adaptive per weight learning rates.

Finally, as in [1], for decoding, we smoothed the posteriors by averaging the output over 30 frames of a sliding window and applied a threshold to detect keywords.

2.1. Low-rank weight matrices

Low-rank weight matrices, implemented as linear bottlenecks (BN), have been used in DNNs for speech recognition [16], where they reduce the number of multiplications during training and testing at the expense of representational expressiveness. Previous work has shown that reducing the rank can have a regularizing effect [17]. However, most previous work has restricted the bottleneck to the input and output weight matrices, where the majority of the parameters are. In our system, the input and hidden layers are similar in size while the output layer is small.

We explored the tradeoff between the number of nodes per hidden layer and the size of the linear bottleneck. To train the reduced rank DNN, we start by completely training the full-rank DNN. Then, one layer at a time, starting from the weight matrix connected to the input layer, we added a linear bottleneck initialized by the SVD of the full-rank matrix. This changes the number of parameters from $M \times N$ to $(M + N) \times R$ where M, N are layer sizes and R is the size of the linear bottleneck. Then, we trained for one epoch, and repeated this step until all weight matrices were decomposed (except the hidden-to-output weight matrix). Finally, we finetuned the network for 20 epochs. When $(M + N) \times R < M \times N$, the number of parameters in the model is reduced. When $(M + N) \times R > M \times N$, we can train the model and then multiply the linear layers out before model evaluation. Hence, during evaluation, the model never used more than $M \times N$ multiplications per nonlinear hidden layer (and in many cases far fewer).

We found that proceeding from input to output rather than the reverse produced better results and that initializing the linear bottleneck with the SVD was essential for training convergence. Moreover, we found that finetuning the initialization from SVD was necessary to achieve strong performance.

2.2. Knowledge distillation

Recent work has shown that a computationally efficient model can be trained to mimic a teacher ensemble of larger models [13, 14]. This is referred to as knowledge distillation (KD). We trained multiple large teacher DNNs and formed an ensemble model by averaging posteriors. The student KD DNN was trained with the weighted criterion:

$$\lambda \sum_i \log(p_i)t_i + \frac{1-\lambda}{T^2} \sum_i \log(p_i(T))q_i(T) \qquad (1)$$

$$p_i(T) = \frac{p_i^{1/T}}{\sum_j p_j^{1/T}}, q_i(T) = \frac{q_i^{1/T}}{\sum_j q_j^{1/T}}, \qquad (2)$$

where p_i is posterior output for class i, t_i is a binary indicator for the correct label, q_i is the posterior output from the teacher model, and λ and T are hyperparameters tuned on the development set. The first term is the standard cross-entropy loss and the second term is a “heated” cross-entropy loss between the output and the posteriors from the teacher model. Notably, the second term is scaled by $1/T^2$ so that the relative weighting between the gradients of the two terms is stable as T is varied [13]. Intuitively, the student model is penalized less for errors that the teacher model also makes whether they come from difficult examples or labeling errors. This allows the student model to focus its limited representational power.

Previously, knowledge distillation had been applied to models with larger target spaces, and it had been hypothesized that its benefit is largely due to the small model learning from the relative sizes of the off target posteriors. Given that our target space is binary, it was unclear if knowledge distillation would be helpful in this setting.

3. Results

For training and evaluation, we used an in-house far-field corpus with an order of magnitude more instances of the keyword “Alexa” and background speech utterances than the largest previous study [1]. The size and composition of this training set and the KWS system is different from that used in Echo. We partitioned 90% of the data for training, 1% for development, and 9% for testing. Given the large size of the corpus, the 1% development set was efficient to evaluate and was more than sufficient to tune parameters (i.e., the development and test set performance were highly correlated). Furthermore, the observed differences on the test set were strongly statistically significant. We also found that different random model initializations did not significantly affect final performance metrics. However, to be most pessimistic, we ran several experiments with random initializations for the baseline model and used the best initialization for comparison.

As we noted earlier, the training objective and the detection task are mismatched (similarly, to frame error rate (FER) and word error rate in ASR). Although we evaluate KWS performance by plotting a detection (DET) curve, we also found it instructive to analyze the FER because that is the objective being trained on.

When trained and evaluated on our far-field corpus, the baseline DNN had comparable performance to the results reported in [1].

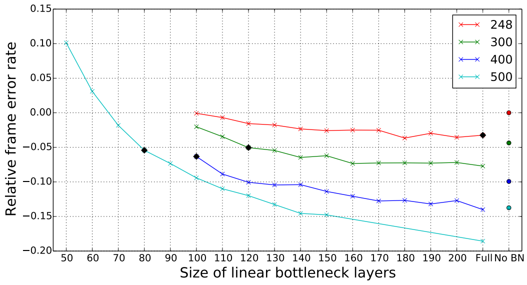

Figure 1: *Relative FER (lower is better) on the test set vs. size of linear bottleneck layers. Full (second to the right) corresponds to the models with bottleneck layers of equal size to the hidden layers. No BN (furthest right) corresponds to the models without linear bottlenecks. The black markers correspond to models that have comparable parameters and multiplications to the baseline 248 nodes per hidden layer model.*

3.1. Low-rank weight matrices

We trained models with 248, 300, 400, and 500 nodes per hidden layer while varying the size of the linear bottleneck layer. Fig. 1 shows the relative FER of these models compared to the baseline model (248 nodes per hidden layer and no linear bottlenecks; lower is better and below 0 indicates superior performance over the baseline). As expected, we observed that increasing the size of the linear bottleneck improved performance. We suspected that performance would peak at an intermediate rank due to the regularizing effect of constraining the rank. Surprisingly, the FER continued to decrease as the number of parameters exceeded the number of parameters in the model without linear bottlenecks (e.g., the 248 node model with size 248 node linear bottlenecks has a nearly 5% relative reduced FER compared to the 248 node model without linear bottlenecks). Importantly, because the bottleneck is linear, we can always multiply the linear transforms after training, so that the test time network will never have more parameters or require more multiplications than the network without linear bottlenecks. This suggests that training with the linear bottlenecks allows us to find a better local optima. As we mentioned earlier, training the baseline with adaptive learning rates did not improve performance. Future work will explore this optimization issue further and determine if it generalizes to other tasks.

We were primarily interested in models that have similar runtime CPU and memory usage as the baseline model. Models with the best performance on the development set for each curve and comparable number of parameters and multiplications are highlighted with black markers in Fig. 1. Of these, the 400 node per hidden layer model with 100 node linear bottlenecks (denoted 400 x 100 BN in Fig. 3) is the best configuration on the development set (corresponding to a 6.3% relative reduction in FER on the test set, Fig. 1)

We also visualized performance versus number of multiplications and parameters in Fig. 2. This shows the tradeoff available between performance and runtime efficiency. In all cases, the low-rank models outperform the models without linear bottlenecks.

3.2. Knowledge distillation

We trained ten 600 node per hidden layer DNNs with different random initializations to form the teacher ensemble. Then to compare against the baseline model, we trained a 248 node per hidden layer DNN using the KD criterion and performed a grid search over λ and T (Table 1). The best configuration on the development set had large gains on the test set (17.1% relative reduction in FER) suggesting that even when the target set is small, knowledge distillation is still an effective technique. Moreover, the results suggest that an extended hyperparameter search may lead to further improvements.

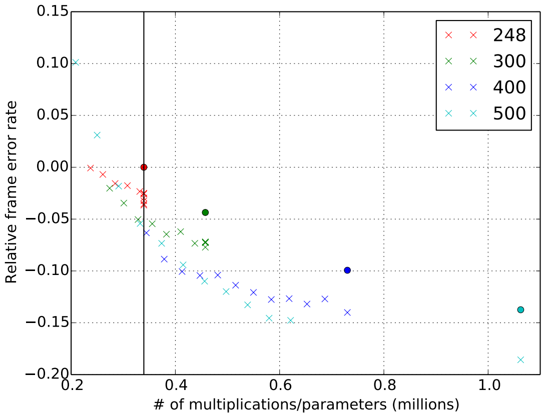

Figure 2: *Relative FER (lower is better) on the test set vs. number of multiplications/parameters. The vertical line marks the the number of multiplications/parameters used by the baseline model. The filled in circles correspond to models without linear bottlenecks. As explained in Section 2.1, models with linear bottlenecks never have more multiplications/parameters than the corresponding model without bottlenecks.*

Table 1: *Relative FER (compared to the baseline; lower is better) on the test set as KD hyperparameters are varied. The best configuration from the development set is bolded.*

		λ				
		0.8	0.6	0.4	0.2	0
T	1	-0.013	-0.055	-0.079	-0.117	-0.119
	2	-0.031	-0.052	-0.118	-0.157	-0.165
	5	-0.072	-0.098	-0.157	-0.16	-0.164
	10	-0.101	**-0.171**	-0.145	-0.164	-0.155

3.3. Combined low-rank and knowledge distillation

Finally, we combined the two approaches using the best low-rank architecture (400 x 100 BN). To train the combined model, we used the KD training criterion throughout the process described in Section 2.1. The training process can be broken into two steps: 1) completely training the initial full-rank model and 2) adding linear bottlenecks and finetuning. We allowed separate hyperparameter settings (λ and T) for each step. The grid search for training the initial 400 nodes per hidden layer model without linear bottlenecks is shown in Table 2. As before, knowledge distillation provided large improvements in FER. Interestingly, for the subsequent step, knowledge distillation did not substantially improve performance over using cross-entropy

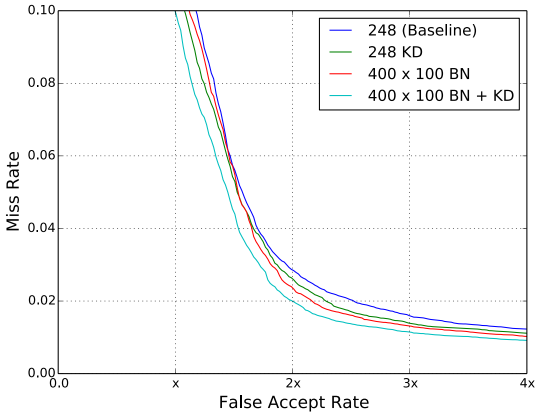

Figure 3: *Test set DET curves on the KWS task. The DET curves only show false alarm rates up to a multiplicative constant due to the sensitive nature of this information. The plot still accurately preserves the relative performance improvements between different systems across a range of reasonable operating thresholds.*

with the hard targets[1] (23.9% versus 23.6% relative reduction in FER on the test set, respectively; as before, the FER is relative to a 248 nodes per hidden layer baseline without linear bottlenecks). We conclude that it suffices to use knowledge distillation to train the initial model without linear bottlenecks and that knowledge distillation is unnecessary when adding the linear bottlenecks to that initial model.

Table 2: *Relative FER for a 400 nodes per hidden layer model on the test set as KD hyperparameters are varied (compared to a 400 nodes per hidden layer baseline model trained without KD; lower is better). The best configuration from the development set is bolded.*

		λ				
		0.8	0.6	0.4	0.2	0
T	1	-0.018	-0.035	-0.074	-0.096	-0.126
	2	-0.045	-0.108	-0.123	-0.159	-0.164
	5	-0.129	-0.154	**-0.174**	-0.171	-0.171
	10	-0.162	-0.144	-0.169	-0.171	-0.164

Although both techniques reduced the DNN frame error, the KWS criterion is a correlated, but different sequence level objective. We evaluated the top performing models (all with comparable runtime CPU and memory usage) by plotting DET curves in Fig. 3 following [1]. The DET curves measure the false alarm rate versus the miss rate. We found that both techniques individually improved end-to-end KWS performance, and the combination further improved performance ($\approx$ 20% relative reduction in miss rate over a wide range of FA rates) all with comparable CPU and memory budget. This is a notable improvement over the state-of-the-art baseline [1].

4. Conclusion

We compressed the DNN in a KWS system by adding linear bottlenecks between each weight layer and by distilling knowledge from a large ensemble of models. Using the low-rank constraint, we reduced relative FER by 6.3% with comparable CPU and memory usage to a baseline model. Distilling knowledge from an ensemble of larger DNNs during the training process reduced relative FER by 17.1%. These two techniques together brought a 23.9% relative reduction in FER, which led to an $\approx$ 20% relative reduction in miss rate over a wide range of FA rates. Both techniques are suitable for the on-device KWS task because they leverage benefits from larger DNN without increasing CPU or memory usage. However, both approaches greatly extend the training process, and it will be useful to explore ways to shorten the training process. In the future, it will also be worthwhile to apply these model compression techniques to RNN based KWS to take advantage of their longer temporal context.

[1] To clarify, this model still used KD when training the initial 400 nodes per hidden layer model. It had substantially better performance than the 400 x 100 BN model trained solely on hard targets in both steps.

5. References

[1] G. Chen, C. Parada, and G. Heigold, "Small-footprint keyword spotting using deep neural networks," in *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*. IEEE, 2014, pp. 4087–4091.

[2] S. Fernández, A. Graves, and J. Schmidhuber, "An application of recurrent neural networks to discriminative keyword spotting," in *Artificial Neural Networks–ICANN 2007*. Springer, 2007, pp. 220–229.

[3] M. Woellmer, B. Schuller, and G. Rigoll, "Keyword spotting exploiting long short-term memory," *Speech Communication*, vol. 55, no. 2, pp. 252–265, 2013.

[4] P. Baljekar, J. F. Lehman, and R. Singh, "Online word-spotting in continuous speech with recurrent neural networks," in *Spoken Language Technology Workshop (SLT), 2014 IEEE*. IEEE, 2014, pp. 536–541.

[5] K. Hwang, M. Lee, and W. Sung, "Online keyword spotting with a character-level recurrent neural network," *arXiv preprint arXiv:1512.08903*, 2015.

[6] M. Sun, V. Nagaraja, B. Hoffmeister, S. Vitaladevuni *et al.*, "Model shrinking for embedded keyword spotting," in *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 2015, pp. 369–374.

[7] W. Chen, J. T. Wilson, S. Tyree, K. Q. Weinberger, and Y. Chen, "Compressing neural networks with the hashing trick," *arXiv preprint arXiv:1504.04788*, 2015.

[8] Y. Wang, J. Li, and Y. Gong, "Small-footprint high-performance deep neural network-based speech recognition using split-vq," in *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*. IEEE, 2015, pp. 4984–4988.

[9] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, "Exploiting linear structure within convolutional networks for efficient evaluation," in *Advances in Neural Information Processing Systems*, 2014, pp. 1269–1277.

[10] M. Jaderberg, A. Vedaldi, and A. Zisserman, "Speeding up convolutional neural networks with low rank expansions," *arXiv preprint arXiv:1405.3866*, 2014.

[11] J. Xue, J. Li, and Y. Gong, "Restructuring of deep neural network acoustic models with singular value decomposition." in *INTERSPEECH*, 2013, pp. 2365–2369.

[12] V. Sindhwani, T. Sainath, and S. Kumar, "Structured transforms for small-footprint deep learning," in *Advances in Neural Information Processing Systems*, 2015, pp. 3070–3078.

[13] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[14] J. Ba and R. Caruana, "Do deep nets really need to be deep?" in *Advances in neural information processing systems*, 2014, pp. 2654–2662.

[15] P. Nakkiran, R. Alvarez, R. Prabhavalkar, and C. Parada, "Compressing deep neural networks using a rank-constrained topology," in *Sixteenth Annual Conference of the International Speech Communication Association*, 2015.

[16] T. N. Sainath, B. Kingsbury, V. Sindhwani, E. Arisoy, and B. Ramabhadran, "Low-rank matrix factorization for deep neural network training with high-dimensional output targets," in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*. IEEE, 2013, pp. 6655–6659.

[17] S. Wiesler, A. Richard, R. Schluter, and H. Ney, "Mean-normalized stochastic gradient for large-scale deep learning," in *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*. IEEE, 2014, pp. 180–184.

[18] A. Senior and X. Lei, "Fine context, low-rank, softplus deep neural networks for mobile speech recognition," in *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*. IEEE, 2014, pp. 7644–7648.

[19] N. Strom, "Scalable distributed dnn training using commodity gpu cloud computing," in *Sixteenth Annual Conference of the International Speech Communication Association*, 2015.

[20] S. Panchapagesan, M. Sun, A. Khare, S. Matsoukas, A. Mandal, B. Hoffmeister, and S. Vitaladevuni, "Multi-task learning and weighted cross-entropy for dnn-based keyword spotting," in *INTERSPEECH 2016 – 16th Annual Conference of the International Speech Communication Association, September 812, San Francisco, California, USA, Proceedings*, 2016.

[21] M. D. Zeiler, "Adadelta: an adaptive learning rate method," *arXiv preprint arXiv:1212.5701*, 2012.

[22] T. Tieleman and G. Hinton, "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude," *COURSERA: Neural Networks for Machine Learning*, vol. 4, p. 2, 2012.