TOWARDS IMPLICIT COMPLEXITY CONTROL USING VARIABLE-DEPTH DEEP NEURAL NETWORKS FOR AUTOMATIC SPEECH RECOGNITION

Shawn Tan, Khe Chai Sim

National University of Singapore

shawntan@comp.nus.edu.sg,simkc@comp.nus.edu.sg

ABSTRACT

In speech recognition, a trade-off can be made between transcription accuracy and computation time. In this paper, we empirically measure the performance of using the softmax outputs connected to different hidden layers of an already fine-tuned deep neural network (DNN) and explore decoding strategies that do not require computing all the hidden layers of the DNN. We find that selecting the specific outputs from a variable-depth DNN achieves better Phoneme Error Rates (PER) on the TIMIT task than directly training a fixed-depth DNN with the same number of layers. We experimented with different ways of stopping the forward-propagation early, first by using a threshold on the entropy of the respective outputs, and formulate a 'gating' system on the hidden layers to predict when to stop the forward propagation.

Index Terms— Speech Recognition, Deep Learning

1. INTRODUCTION

DNN acoustic modeling has achieved state-of-the-art performance in comparison to conventional Gaussian Mixture Model (GMM) based systems. Hinton et. al. provides a good summary of techniques used and their results in comparison to GMMs [1]. However, the improved performance comes at a computational cost. There has been work exploring the capabilities of shallower networks which have been 'compressed' from deeper networks to reduce computation cost [2, 3, 4, 5]. Some approaches have also attempted to use Singular Value Decomposition (SVD) on the layer weights to reduce the complexity of the forward propagation [6, 7, 8].

As speech recognition continues to move towards handheld devices with limited computational power, the industry has interests to keep computational costs low. We explore methods of training DNNs that attempt to reduce required computation steps by determining if a prediction is ready to be made at any layer during the 'computation' being performed by the DNN. We achieve this by experimenting with a variety of methods: 1) We train an additional output for each of the hidden layers, allowing us to decide during runtime which output we want to use. 2) We look at metrics for deciding when to stop forward propagation.

The rest of the paper is organised as follows. In Section 2, a brief review of DNN complexity reduction techniques for speech applications is given. Section 3 formulates the Variable-depth DNN, and describes two layer selection criteria. Section 4 describes the method for learning the performance-complexity tradeoff profile for VDNN. Experimental results are reported in Section 5 and our work is concluded in Section 6.

2. MODEL COMPLEXITY CONTROL

Senior and Lei [6] approached the problem by doing dimensionality reduction on the final softmax layer. This not only gave them gains in terms of word error rate (WER), but also reduced the capacity required to store the model, and reduced the model complexity. Their work was preceded by Sainath et. al. [7], who also made use of matrix factorisation techniques to reduce the number of parameters required in the final softmax layer. Their work also experimented with applying the factorisation to multiple layers, which did not give them considerable gains. Jian Xue et. al. [8] also perform similar techniques without loss of accuracy to their models.

There are also some efforts at model compression from Hinton et. al. [2] and Ba and Caruana [3]. This method involves first training a deep neural network on the labels. Once this is done, a shallower network with a comparable number of parameters is used, and this new shallow network is trained to *mimic* the deep network. This is done by training the shallow network using the squared error of the logits – the linear outputs of the final layer before softmax is applied – and can also be achieved by using the cross-entropy cost between the deep network and the shallow network. Similar approaches were applied to speech data with some reductions in WER by Li et. al. [4]. Romero et. al. [5] also developed FitNets, which made use of similar techniques at the hidden layers in order to learn deep but thinner nets that run faster. Lee et. al. [9] has proposed using classification outputs at hidden layers for computer vision tasks, and reports gains in this approach.

3. VARIABLE-DEPTH DEEP NEURAL NETWORKS

We propose Variable-depth Deep Neural Network (VDNN) to provide the flexibility for controlling the model complexity on demand at runtime. Let the hidden layer activations at layer l be defined as:

$$\boldsymbol{h}^l = \sigma\left(\boldsymbol{W}_h^l \boldsymbol{h}^{l-1} + \boldsymbol{b}_h^l\right) \tag{1}$$

where $\sigma(\cdot)$ is the sigmoid activation function, and $\boldsymbol{W}_h^l$ and $\boldsymbol{b}_h^l$ are the weight matrix and bias vector for that layer. The subscript h indicates that these are the hidden layer parameters. The corresponding outputs, $\boldsymbol{o}^l$, are given by

$$\boldsymbol{o}^l = \text{softmax}\left(\boldsymbol{W}_o^l \boldsymbol{h}^l + \boldsymbol{b}_o^l\right) \tag{2}$$

where $\boldsymbol{W}_o^l$ and $\boldsymbol{b}_o^l$ are the weight matrix and bias vector for generating the outputs at layer l. Note that for a regular DNN, there will only be one set of outputs, given by $\boldsymbol{o}^L$.

A simple strategy is to select a fixed layer for a specific task. We refer to this as Fixed-layer VDNN (F-VDNN). It is possible to select a different layer for each frame. A score, s_t^l, is defined for each layer, l, at each time frame, t. A simple threshold-based selection criterion is adopted:

$$l_t^* = \underset{\{l: s_t^l \geq \lambda^l\}}{\arg\min}\, l \tag{3}$$

where the selected layer for frame t, l_t^*, is the lowest layer whose score, s_t^l, is above the selection threshold, λ^l. In the following, we will consider two types of selection criteria.

3.1. Entropy-based criterion

Entropy is a metric of information content that can be used to measure the certainty in the layer's predictions — the assumption is that a prediction is more likely to be right if the output is more certain. The selection score based on the entropy criterion can be formulated as:

$$s_t^l = 1 - H(\boldsymbol{o}^l)/H_{\texttt{max}} \tag{4}$$

where the entropy function is given by

$$H(\boldsymbol{o}^l) = -\sum_{i=1}^{N} o_i^l \log o_i^l \tag{5}$$

and $H_{\texttt{max}} = \log N$ is the maximum entropy for N-dimensional outputs with uniform probabilities. The resulting score ranges between 0 and 1, with a higher score leading to a higher chance of selection.

The issue with using the entropy-based criterion is that the output layer has to be computed, and this incurs additional computational cost, resulting in more than two times the number of matrix multiplications required by the end of the forward propagation.

3.2. Gate-based criterion

In order to avoid computing the softmax outputs unnecessarily, we perform *logistic regression* on the hidden layer activations to derive the selection score as follows:

$$s_t^l = g^l = \sigma(\boldsymbol{w}_g^l \boldsymbol{h}^l + b_g^l) \tag{6}$$

This score ranges between 0 and 1, can be viewed as a '*gate*' that controls the feedforward process. The additional computational costs incurred in calculating g^l is negligible compared to the overall computation needed to calculate entropy.

We can train the logistic regression parameters, $\boldsymbol{w}_g^l$ and b_g^l, by treating them as part of the model parameters of the VDNN. To integrate $\boldsymbol{w}_g^l$ and b_g^l into the VDNN model, we define the following backward recursion to derive the output of the VDNN:

$$\hat{\boldsymbol{o}}^l = g^l \boldsymbol{o}^l + (1 - g^l)\hat{\boldsymbol{o}}^{l+1} \tag{7}$$

where $\hat{\boldsymbol{o}}^L = \boldsymbol{o}^L$ and L is the final output layer in the network. This output of the VDNN is given by $\hat{\boldsymbol{o}}^1$, which is used to compute the cross-entropy criterion for error back propagation training of the regression parameters.

Structuring the gates in this way guarantee that 1) the final probabilities sum to 1: the base case is a standard softmax, and every subsequent combination is a convex sum of two probability distributions, and that 2) in order for a good prediction, the correct gate must be on, and the lower gates in the network take precedence.

4. LEARNING PERFORMANCE-COMPLEXITY TRADEOFFS

Given the score for each layer at each time, s_t^l, and the corresponding accuracy, y_t^l, we can analyse the tradeoffs between the accuracy performance and the model complexity (in terms of the number of layers used) as a function of the selection threshold, λ. To simplify the analysis, we will consider using a global selection threshold for all the layers.

Therefore, it is necessary to normalise the scores so that a consistent decision can be made across different layers. We normalise the scores such that the optimum decision (which yields the highest classification accuracy for each individual layer) is 0.5. First, we find the optimum decision for each layer as follows:

$$\theta^l = \underset{\theta}{\arg\max} \frac{1}{T} \sum_{t=1}^{T} a_t^l(\theta) \tag{8}$$

where the accuracy given the threshold, θ, for layer l at time t is given by

$$a_t^l(\theta) = \begin{cases} y_t^l & \text{if } s_t^l \geq \theta \\ 1 - y_t^l & \text{otherwise} \end{cases} \tag{9}$$

Once the optimum thresholds are determined, the scores are normalised as follows:

$$\tilde{s}_t^l = \frac{(1-\theta^l)s_t^l}{(1-\theta^l)s_t^l + (1-s_t^l)\theta^l} \tag{10}$$

The above equation will transform the scores such that $\tilde{s}_t^l = 0.5$ if $s_t^l = \theta^l$. In other words, the optimum decision is 0.5 for all the layers after score normalisation.

Let us define the effective accuracy and number of layers used for each layer recursively in terms of those of the layer below it:

$$\tilde{a}_t^l(\theta) = \begin{cases} y_t^l & \text{if } s_t^l \geq \theta \\ \tilde{a}_t^{l-1}(\theta) & \text{otherwise} \end{cases} \tag{11}$$

$$\tilde{n}_t^l(\theta) = \begin{cases} l & \text{if } s_t^l \geq \theta \\ \tilde{n}_t^{l-1}(\theta) & \text{otherwise} \end{cases} \tag{12}$$

The boundary conditions are $\tilde{a}_t^1(\theta) = y_t^1$ and $\tilde{n}_t^1(\theta) = 1$. Therefore, the effective accuracy and layer counts for the overall model is given by

$$\bar{a}(\theta) = \frac{1}{T}\sum_{t=1}^{T}\tilde{a}_t^L(\theta) \quad \text{and} \quad \bar{n}(\theta) = \frac{1}{T}\sum_{t=1}^{T}\tilde{n}_t^L(\theta) \tag{13}$$

To create a tradeoff profile, we first group the scores from all the layers and sort them in ascending order. Let $\boldsymbol{v}$ be a vector denoting the sorted scores, $\boldsymbol{v} = [v_k]_{k=1}^K$, where $K = T \times L$. We can then define a set of $K-1$ possible thresholds as:

$$\bar{\lambda}_k = \frac{v_k + v_{k+1}}{2} \tag{14}$$

For each of these thresholds, the corresponding accuracy and layer counts can be computed as $\bar{a}_k = \bar{a}(\bar{\lambda}_k)$ and $\bar{n}_k = \bar{n}(\bar{\lambda}_k)$, which effectively give us the tradeoff profile of the system. In this work, the thresholds, $\bar{\lambda}_k$, and the corresponding tradeoff profile, $\bar{a}_k$ and $\bar{n}_k$ are computed using validation data used to train the DNN. At runtime, the model complexity can be adjusted on demand by choosing the appropriate threshold $\bar{\lambda}_k$. Suppose that the desired average number of layers to be used is $\hat{n}$, the threshold can be determined as

$$\hat{n} = \bar{\lambda}_{\hat{k}} \quad \text{where} \quad \hat{k} = \arg\min_k \left|\bar{n}_k - \hat{n}\right| \tag{15}$$

It is possible to choose the threshold by optimising the weighted average between the accuracy and model complexity:

$$\hat{k} = \arg\max_k \left[(1-\alpha)\bar{a}_k - \alpha\bar{n}_k\right] \tag{16}$$

where α is the weight that can be adjusted to control the tradeoff between accuracy and model complexity.

Layer	Frame Acc. %		PER %	
	Oracle	F-VDNN	DNN	F-VDNN
1	46.0	46.0	24.4	24.2
2	56.8	49.6	23.2	22.9
3	62.0	51.6	23.2	22.3
4	64.7	52.1	23.1	22.0
5	66.6	51.8	22.1	21.8
6	68.4	53.7	21.8	21.8

Table 1. *Performance of the various outputs of F-VDNN.*

5. EXPERIMENTS

The experiments in this paper are performed on the TIMIT corpus with the training set of 462 speakers, without SA sentences. A development set of 50 speakers are used for hyperparameter tuning and we report the results on the standard core test set. The alignments used in DNN training are taken from the GMM-HMM system using triphones, trained using the Kaldi toolkit [10]. A standard DNN is first trained on 40 dimensional filterbank features and energies. These are extracted from the speech using a 25ms window and a 10ms frame-shift. A delta of order 2 is appended, and the concatenated features are then spliced with a context window of 5 (11 frames in total). Cepstral mean and variance normalisation (CMVN) is then applied to the resulting feature vectors. Each input frame is a vector of 1353 dimensions. Pre-training is done using Stacked Denoising Autoencoders (SDAs). The DNN used has 6 sigmoid hidden layers, with 1024 units per layer, and 1874 senones as outputs. All the DNNs are trained with a mini-batch size of 256. A softmax layer then is added to each of the hidden layers. The parameters of the DNN are frozen, and the parameters connecting the hidden layer to the softmax output are trained using cross-entropy using the same aligned labels as in the standard model. A set of corresponding models starting from the pretrained weights are trained with 1 through 6 hidden layers. All DNN systems are trained with the Theano library [11].

Table 1 compares the performance of the various outputs of VDNN. Column 2 shows the frame accuracies assuming we chose the first layer with the correct prediction and Column 3 the frame accuracies on the validation set. Column 4 and 5 compare the phone error rate (PER) performance of DNN and VDNN with comparable number of hidden layers. It is interesting to note that the regular DNN models trained directly with the pretrained weights do not do as well as those of the VDNN. In terms of computational costs, forward propagation through both models are equivalent. The VDNN model, encompasses 6 different output layers that we can select at runtime to adjust model complexity.

We investigate dynamic layer selection using the entropy-based and gate-based criteria, as outlined in Sections 3.1 and 3.2 respectively. Fig. 1 shows the score distributions of the two criteria for each layer.

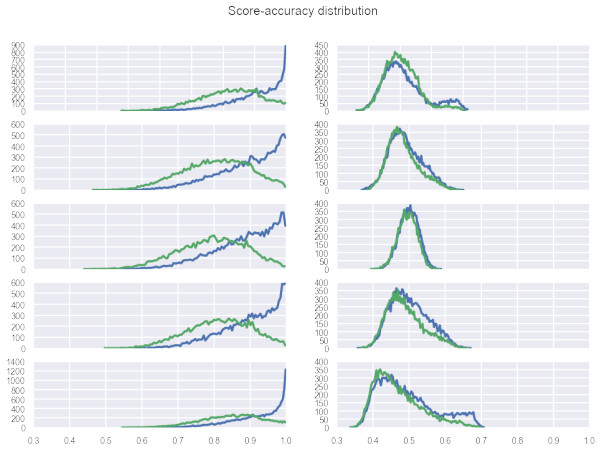

Fig. 1. *Score distributions for different layers: entropy-based scores (left) and gate-based scores (right). The blue and green curves correspond to the scores of the correctly and incorrectly classified frames respectively.*

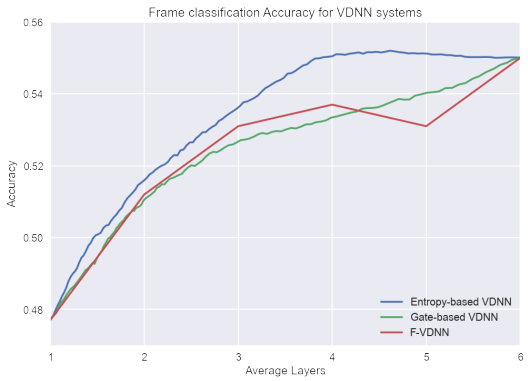

Fig. 2. *Frame classification accuracy against the average layers used.*

Fig. 2 shows the tradeoffs between frame accuracy and the average number of layers used for F-VDNN, entropy-based VDNN and gate-based VDNN. The tradeoff curves for the VDNNs are computed using the method described in Section 4. As expected, the entropy-based VDNN achieved consistently the best frame accuracy performance for different average number of layers used. The Gate-based VDNN model shows comparable performance compared to the F-VDNN.

We compared the tradeoffs between PER performance and the average number of layers used for various models in Fig. 3. As discussed before, the F-VDNN performs better compared to the DNN models with the same model complexity. Both the entropy-based and Gate-based VDNN models achieved similar performance compared to the F-VDNN model. The VDNN models offers a more fine-grained complexity control at runtime.

Figure 4 shows a histogram of the distribution of layers

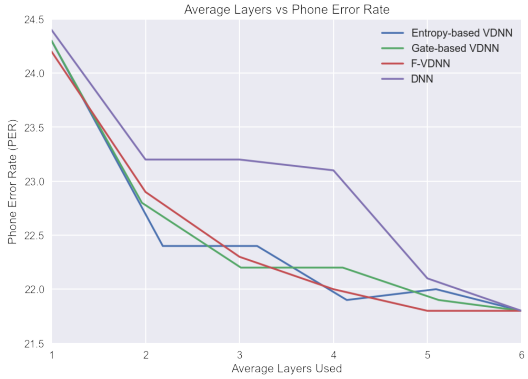

Fig. 3. *The plot of PER against average layers used.*

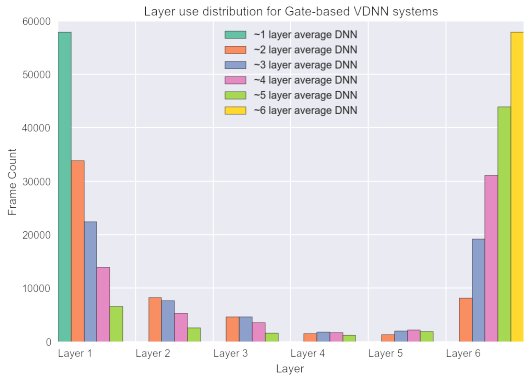

Fig. 4. *Distribution of the layer usage for the various systems.*

used during test time for each of the thresholds set for the Gate-based VDNN. The layer counts are skewed toward the first and the final layer. This may be because the final layers give a good classification while the classifications at the first layer are compensating, resulting in a lower average layer count .

6. CONCLUSION

In this paper, we investigated variable-depth DNN (VDNN) that offers the flexibility for controlling the model complexity at runtime. One interesting observation from this work is that the intermediate outputs from a VDNN achieved better phone recognition performance compared to a regular DNN with comparable number of hidden layers. As the gating criterion for stopping forward propagation requires a fair amount of fine-tuning to get working, a simple approach for creating models from already trained models is to simply train a linear classifier over a lower hidden layer of the original network. However, the entropy criterion does show that there is sufficient signal at the lower layers to determine if a classification can be used at a lower layer, so the authors remain optimistic about the feasibility of a gating system.

7. REFERENCES

[1] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *Signal Processing Magazine, IEEE*, vol. 29, no. 6, pp. 82–97, 2012.

[2] G. Hinton, O. Vinyals, and J. Dean, "Distilling the Knowledge in a Neural Network," pp. 1–9.

[3] L. J. Ba and R. Caurana, "Do Deep Nets Really Need to be Deep ?" *arXiv preprint arXiv:1312.6184*, pp. 1–6, 2013. [Online]. Available: http://arxiv.org/abs/1312.6184

[4] J. Li, R. Zhao, J.-T. Huang, and Y. Gong, "Learning small-size dnn with output-distribution-based criteria," in *Proc. Interspeech*, 2014.

[5] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio, "Fitnets: Hints for thin deep nets," *arXiv preprint arXiv:1412.6550*, 2014.

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

[7] 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," *Icassp2013*, pp. 6655–6659, 2013.

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

[9] C.-Y. Lee, S. Xie, P. Gallagher, Z. Zhang, and Z. Tu, "Deeply-supervised nets," *arXiv preprint arXiv:1409.5185*, 2014.

[10] D. Povey, A. Ghoshal, G. Boulianne, L. Burget, O. Glembek, N. Goel, M. Hannemann, P. Motlíček, Y. Qian, P. Schwarz *et al.*, "The kaldi speech recognition toolkit," 2011.

[11] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio, "Theano: a CPU and GPU math expression compiler," in *Proceedings of the Python for Scientific Computing Conference (SciPy)*, Jun. 2010, oral Presentation.