NEURAL ARCHITECTURE SEARCH FOR LF-MMI TRAINED TIME DELAY NEURAL NETWORKS

Shoukang Hu, Xurong Xie, Shansong Liu, Mingyu Cui, Mengzhe Geng, Xunying Liu, Helen Meng

The Chinese University of Hong Kong, Hong Kong SAR, China

{skhu, ssliu, mycui, mzgeng, xyliu, hmmeng}@se.cuhk.edu.hk {xr.xie}@link.cuhk.edu.hk

ABSTRACT

Deep neural networks (DNNs) based automatic speech recognition (ASR) systems are often designed using expert knowledge and empirical evaluation. In this paper, a range of neural architecture search (NAS) techniques are used to automatically learn two types of hyper-parameters of state-of-the-art factored time delay neural networks (TDNNs): i) the left and right splicing context offsets; and ii) the dimensionality of the bottleneck linear projection at each hidden layer. These include the DARTS method integrating architecture selection with lattice-free MMI (LF-MMI) TDNN training; Gumbel-Softmax and pipelined DARTS reducing the confusion over candidate architectures and improving the generalization of architecture selection; and Penalized DARTS incorporating resource constraints to adjust the trade-off between performance and system complexity. Parameter sharing among candidate architectures allows efficient search over up to 7^{28} different TDNN systems. Experiments conducted on the 300-hour Switchboard corpus suggest the auto-configured systems consistently outperform the baseline LF-MMI TDNN systems using manual network design or random architecture search after LHUC speaker adaptation and RNNLM rescoring. Absolute word error rate (WER) reductions up to 1.0% and relative model size reduction of 28% were obtained. Consistent performance improvements were also obtained on a UASpeech disordered speech recognition task using the proposed NAS approaches.

Index Terms— Neural Architecture Search, Time Delay Neural Network, Speech Recognition

1. INTRODUCTION

Deep neural networks (DNNs) play a central role in state-of-the-art automatic speech recognition (ASR) systems[1, 2, 3, 4, 5, 6, 7]. When designing these systems, a set of DNN structure design decisions such as the hidden layer dimensionality and connectivity need to be made. These decisions are largely based on expert knowledge or empirical choice. As explicitly training and evaluating the performance of different network architectures is highly expensive, it is preferable to use automatic architecture design techniques [8, 9].

To this end, neural architecture search (NAS) approaches [10] have gained increasing interests in recent years. The key objectives of NAS methods are three fold. First, it is crucial to produce an accurate performance ranking over different candidate neural architectures to allow the best system to be selected. Second, when operating at the same level of accuracy performance target, preference should be given to simpler architectures with fewer parameters in order to minimize the risk of overfitting to limited data. Furthermore, to ensure scalability and efficiency on large data sets, a search space containing all candidate systems of interest needs to be defined.

Earlier forms of NAS techniques were based on neural evolution [11], where genetic algorithms were used to randomly select architecture choices at each iteration of mutation and crossover. Bayesian NAS methods based on Gaussian Process was proposed in [12]. Reinforcement learning (RL) based NAS approaches [13, 14] have also been developed. In these techniques, explicit system training and evaluation are required. In addition, as the architecture hyper-parameters and actual DNN parameters are separately learned, e.g., within the RL controller and candidate systems, a tighter integration of both is preferred during NAS.

Alternatively, differentiable architectural search (DARTS) techniques can be used [15, 16, 17, 18, 19]. Architectural search is performed over an over-parameterized parent super-network containing paths connecting all candidate DNN structures to be considered. The search is transformed into the estimation of the weights assigned to each candidate neural architecture within the super-network. The optimal architecture is obtained by pruning lower weighted paths. This allows both architecture selection and candidate DNN parameters to be consistently optimized within the same super-network model.

In contrast to the rapid development of NAS techniques in the machine learning and computer vision communities, there has been very limited research of applying these to speech recognition systems so far. In this paper, a range of DARTS based NAS techniques are used to automatically learn two architecture hyper-parameters that heavily affect the performance and model complexity of state-of-the-art factored time delay neural network (TDNN-F) [1, 5, 7, 6] acoustic models: i) the left and right splicing context offsets; and ii) the dimensionality of the bottleneck linear projection at each hidden layer. These include the standard DARTS method fully integrating the estimation of architecture weights and TDNN parameters in lattice-free Maximum Mutual Information (LF-MMI) training; Gumbel-Softmax DARTS that reduces the confusion between candidate architectures; pipelined DARTS that circumvents the overfitting of architecture weights using validation data; and penalized DARTS that further incorporates resource constraints to flexibly adjust the trade-off between performance and system complexity. Parameter sharing among candidate architectures was also used to facilitate efficient search over a large number of TDNN systems. Experiments conducted on a 300-hour Switchboard conversational telephone speech recognition task suggest the NAS configured TDNN-F systems consistently outperform the baseline LF-MMI trained TDNN-F systems using manually designed configurations. Absolute word error rate reductions up to 1.0% and model size reduction of 28% relative were obtained. In order to further evaluate the performance of the proposed NAS techniques, they were applied to automatically configure two sets of hyper-parameters of a state-of-the-art disordered speech recognition task based on the UASPEECH corpus [20]: skip connection between layers and dimensionality of factored TDNN weight matrices.

To the best of our knowledge, this paper is among the first to apply neural architecture search techniques to TDNNs in speech recognition tasks. In contrast, the vast majority of previous NAS research has been focused on computer vision applications [21, 22, 23]. Existing NAS works in the speech community investigated non-TDNN based architectures [24, 25, 26, 27, 28, 29].

The rest of this paper is organized as follows. Section 2 presents a set of differentiable NAS techniques. Section 3 discusses the search space of TDNN-F models and necessary parameter sharing to improve search efficiency. Section 4 presents the experiments and results. Finally, the conclusions are drawn in Section 5.

2. NEURAL ARCHITECTURE SEARCH

In this section, we present various forms of differentiable neural architecture search (DARTS) methods. With no loss of generality, we introduce the general form of DARTS architecture selection methods [15, 16, 17, 18]. For example, the l-th layer output $\mathbf{h}^l$ can be computed as follows in the DARTS supernet:

$$\mathbf{h}^l = \sum_{i=0}^{N^l-1} \lambda_i^l \phi_i^l(\mathbf{W}_i^l \mathbf{h}^{l-1}) \tag{1}$$

where λ_i^l is the architecture weight for the i-th candidate choice in the l-th layer, N^l is the total number of choices in this layer. The precise form of neural architectures being considered at this layer is determined by the linear transformation parameter $\mathbf{W}_i^l$ and activation function $\phi_i^l(\cdot)$ used by each candidate system. For example, when selecting the TDNN-F hidden layer context offsets, the linear transformation is a binary-valued matrix for each candidate architecture, and $\phi_i^l(\cdot)$ is represented by an identity matrix. When selecting the dimensionality of the bottleneck linear projection at each hidden layer, the linear transformation $\mathbf{W}_i^l = \widetilde{\mathbf{W}}_i^l \widehat{\mathbf{W}}_i^{l^T}$ is a decomposed matrix, while $\phi_i^l(\cdot)$ is also an identity matrix.

2.1. Softmax DARTS: The conventional DARTS [15] system uses a Softmax function to model the architecture selection weight λ_i^l as

$$\lambda_i^l = \frac{\exp(\log \alpha_i^l)}{\sum_{j=0}^{N^l-1} \exp(\log \alpha_j^l)} \tag{2}$$

When using the standard back-propagation algorithm to update the architecture weights parameter λ_i^l, the loss function (including LF-MMI criterion [6] considered in this paper) gradient against the λ_i^l is computed as below.

$$\frac{\partial \mathcal{L}}{\partial \log \alpha_k^l} = \frac{\partial \mathcal{L}}{\partial \mathbf{h}^l} \sum_{i=0}^{N^l-1} \left(1_{i=k} \lambda_i^l - \lambda_i^l \lambda_k^l\right) \phi_i^l(\mathbf{W}_i^l \mathbf{h}^{l-1}) \tag{3}$$

where $1_{i=k}$ is the indicator function. When the DARTS supernet containing both architecture weights and normal DNN parameters is trained to convergence, including architecture parameters and normal DNN parameters, the optimal architecture can be obtained by pruning lower weighted architectures that are considered less important. However, when similar architecture weights are obtained using a flattened Softmax function, the confusion over different candidate systems increases and search errors may occur.

2.2. Gumbel-Softmax DARTS: In order to address the above issue, a Gumbel-Softmax distribution [30] is used to sharpen the architecture weights to produce approximately a one-hot vector. This allows the confusion between different architectures to be minimised. The architecture weights are computed as,

$$\lambda_i^l = \frac{\exp((\log \alpha_i^l + G_i^l)/T)}{\sum_{j=0}^{N^l-1} \exp((\log \alpha_j^l + G_j^l)/T)} \tag{4}$$

where $G_i^l = -\log(-\log(U_i^l))$ is the Gumbel variable, and U_i^l is a uniform random variable. When the temperature parameter T approaches 0, it has been shown that the Gumbel-Softmax distribution is close to a categorical distribution [30].

Different samples of the uniform random variable U_i^l lead to different values of λ_i^l in Eq. 4. The loss function gradient w.r.t $\log \alpha_k^l$ is computed as an average over J samples of the architecture weights,

$$\frac{\partial \mathcal{L}}{\partial \log \alpha_k^l} = \frac{1}{J} \sum_{j=0}^{J} \frac{\partial \mathcal{L}}{\partial \mathbf{h}^{l,j}} \sum_{i=0}^{N^l-1} \frac{1_{i=k} \lambda_i^{l,j} - \lambda_i^{l,j} \lambda_k^{l,j}}{T} \phi_i^l(\mathbf{W}_i^l \mathbf{h}^{l-1,j}) \tag{5}$$

where $\lambda^{l,j}$ is the j-th sample weights vector drawn from the Gumbel-Softmax distrbution in the l-th layer, $\mathbf{h}^{l,j}$ is the output of l-th layer by using the j-th sample $\lambda^{l,j}$. We assume the Gumbel-Softmax variables λ^l at different layers are mutually independent.

2.3. Pipelined DARTS: As both architecture weights and normal DNN parameters are learned at the same time in Softmax DARTS and Gumbel-Softmax DARTS systems, the search algorithms may prematurely select sub-optimal architectures at an early stage. Inspired by [31], we decouple the update of normal DNN parameters and architecture weights into two separate stages performed in sequence. This leads to the pipelined DARTS approach. In order to prevent overfitting to the training data, a separate held-out data set taken out of the original training data is used. In Pipelined DARTS systems, the normal DNN parameters are updated to convergence on the training data first, while randomly sampled one-hot architecture weights drawn from a uniform distribution are used. In the following stage, we fix the normal DNN parameters estimated in the first stage in the supernet and update the architecture weights using the held-out data for both Softmax DARTS and Gumbel-Softmax DARTS. This produces the Pipelined Softmax DARTS (PipeSoftmax) and Pipelined Gumbel-softmax DARTS (PipeGumbel) systems.

2.4. Penalized DARTS: In order to flexibly adjust the trade-off between system performance and complexity, a penalized loss function incorporating the underlined neural network size is used.

$$\mathcal{L} = \mathcal{L}_{LF-MMI} + \eta \sum_{l,i} \lambda_i^l C_i^l \tag{6}$$

where C_i^l is the number of parameters of the i-th candidate considered at the l-th layer, and η is the penalty scaling factor empirically set for different tasks.

3. SEARCH SPACE AND PARAMETER SHARING

This section describes the search space and its implementation when NAS methods of Sec. 2 are used to automatically learn two sets of hyper-parameters of TDNN-F models: i) the left and right splicing context offsets; and ii) the dimensionality of the bottleneck linear projection at each hidden layer. Parameter sharing among candidate architectures used to facilitate efficient search over a large number of TDNN-F systems is also presented.

3.1. TDNN-F Context Offset Search Space: Context offset settings play an important role in modeling the long temporal information in TDNN-F models. However, manually selecting context offsets is time-consuming for different applications. Inspired by the parameter-sharing used in earlier NAS research [14], we design a

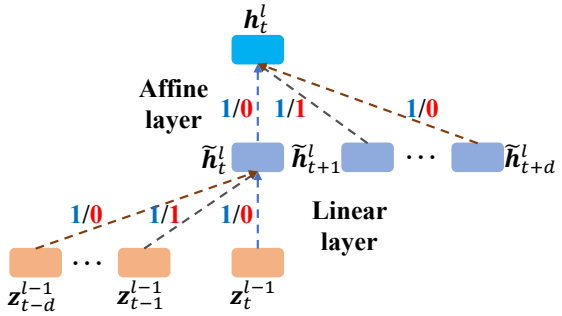

Fig. 1. Example part of a supernet containing all the context offsets for a TDNN-F layer. Dashed lines with different colors represent different context choices in each linear (left context) and affine (right context) transforms. The blue integers denote the supernet system using all the context offsets, while the red integers represent a candidate offset choice of ± 1.

TDNN-F supernet (Fig. 1) to contain all possible choices of context offsets to the left ({-d,0}, $\cdots$, {-1,0}, {0,0}) and right ({0,0}, {0,1}, $\cdots$, {0,d}) at each layer during search. Note that 0,d denote context offsets of 0 and d (right). For the supernet system, it requires the sparse context connection weights to be densely set as 1 for all context offsets. Any candidate TDNN-F model with particular context offsets, out of the total $(d+1)^{2L}$ possible choices, contained in the supernet is represented by setting the corresponding connection weights to be 1, while setting the others to be 0.

3.2. TDNN-F Bottleneck Dimensionality Search Space: Similarly, a TDNN-F supernet containing all the candidate TDNN-F with different projection dimensions is designed, as shown in Fig. 2 for one hidden layer. When applying the NAS methods in Sec.2, $\phi_i^l(\cdot)$ of Eq. 1 is set as an identity matrix. In common with the standard TDNN-F model, the weight matrix $\mathbf{W}_i^l$ of i-th architecture choice in l-th layer is factored into one semi-orthogonal weight matrix $\widetilde{\mathbf{W}}_{0:n_i-1}^l$ and one affine weight matrix $\widehat{\mathbf{W}}_{0:n_i-1}^l$ as shown in Fig. 2. n_i is the dimensionality of the i-th architecture. Parameter sharing among different candidate architectures' linear matrices $\widetilde{\mathbf{W}}_{0:k}$ (left to right from the first column) and affine matrices $\widehat{\mathbf{W}}_{0:k}$ (bottom to up from the first row) ($0 \leq k \leq n-1$) is implemented by the corresponding submatrices extracted from the largest matrix $\widetilde{\mathbf{W}}_{0:n-1}$. Such sharing allows a large number of TDNN-F projection dimensionality choices at each of the 14 layers, e.g., selected from 8 values {25, 50, 80, 100, 120, 160, 200, 240} as considered in this paper, to be compared for selection during search. This leads to a total of 8^{14} candidate TDNN-F systems to be selected from.

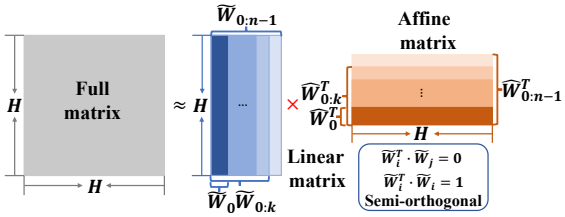

Fig. 2. Example part of a supernet containing different bottleneck projection dimensionality choices in the TDNN-F hidden layer. The full weight matrix is factored into one semi-orthogonal linear weight matrix $\widetilde{\mathbf{W}}_{0:n-1}$ and one affine weight matrix $\widehat{\mathbf{W}}_{0:n-1}$. Architectures with different projection dimensions are represented by the corresponding submatrices starting form the first column.

4. EXPERIMENTAL RESULTS

This section presents our experiments carried out on the 300-hour Switchboard telephone speech recognition task using the Kaldi toolkit [32]. The GMM-HMM system [33, 34], TDNN-F acoustic model [6], and language model are similar to those described in [35]. In the searching stage, TDNN-F supernet models are trained on the training set with one thread for 3 epochs, while architecture parameters of PipeSoftmax and PipeGumbel systems are updated for additional 3 epochs using a held-out data set by fixing the normal DNN parameters. Note that we randomly select 5% of the original training set as the held-out data set and T in the Gumbel-Softmax distribution is annealed from 1 to 0.03 in our experiments. Once candidate TDNN-F models are derived from the searching stage, they are trained for 3 epochs from scratch. Note that a matched pairs sentence-segment word error (MAPSSWE) based statistical significance test was performed at a significance level $\alpha = 0.05$.

4.1 TDNN-F Context Offset Search on SWBD: In this section, we describe the experimental results of searching context offsets at each layer by using various NAS methods of Sec. 2. In Table 1, systems (5)-(6), (9)-(10) perform the search over the total 4^{28} TDNN-F choices with maximum context offsets of ± 3 using 36 hours on a 32M param. supernet, while systems (7)-(8) perform the search over 7^{28} TDNN-F choices with the maximum context offsets of ± 6 using 60 hours on a 53M param. supernet. Two trends can be observed from Table 1. First, the Gumbel-Softmax DARTS system (Sys (8)) outperforms the baseline Kaldi recipe TDNN-F system (Sys (1)) by **0.3%** and **1.0%** absolute WER reductions on the **swbd** and **callhm** test sets. Even compared with additional manually designed TDNN-F systems (Sys (3)), the Gumbel-Softmax DARTS system (Sys 8) still produces **0.3%** absolute WER reduction on the **callhm** test set. Second, Gumbel-Softmax DARTS systems (Sys (6), (8)) obtain better performance than Softmax DARTS systems (Sys (5), (7)).

Table 1. Performance (WER%) comparison of TDNN-F models (#param:18M) configured with context offsets produced by the baseline system, manual designed systems, Softmax DARTS (Softmax), Gumbel-Softmax DARTS (Gumbel), Pipelined Softmax DARTS (PipeSoftmax), Pipelined Gumbel-Softmax DARTS (PipeGumbel) systems described in Sec. 2. $\{[a,b]\}$:$\{-c,d\}$ denotes context offsets $\{-c,0\}$ on the left and $\{0,d\}$ on the right used from a-th layer to b-th layer inclusive. † denotes a statistically significant difference is obtained over the baseline system (Sys (1)).

Sys	Method	Context Offsets	Hub5' 00	
			swbd	callhm
1	Baseline[32]	{[1,3]}:{-1,1}; {4}:{0}; {[5,14]}:{-3,3}	10.0	20.8
2	Manual	{[1,3]}:{-1,1}; {4}:{0}; {[5,14]}:{-6,6}	**9.8**	**20.1**
3		{[1,3]}:{-1,1}; {4}:{0}; {[5,14]}:{-9,9}	**9.7**	**20.1**
4		{[1,3]}:{-1,1}; {4}:{0}; {[5,14]}:{-12,12}	10.1	20.8
5	Softmax	{1}:{0,1};{[2,14]}:{-3,3}	10.2	20.6
6	Gumbel	{1,2}:{-2,2}; {3,4}:{-2,3};{[5,14]}:{-3,3}	9.9	20.1†
7	Softmax	{1}:{0,1};{2}:{-6,1}; {3}:{-6,2};{[4,14]}:{-6,6}	9.7	20.1†
8	Gumbel	{1,2}:{-3,2}; {3}:{-4,3};{4,5}:{-4,4}; {6,7}:{-4,6};{8}:{-5,6};{[9,14]}:{-6,6}	**9.7**	**19.8**†
9	PipeSoftmax	{1}:{-1,3};{2,[4,12],14}:{-3,3}; {3}:{-2,3};{13}:{0,3}	9.9	20.4†
10	PipeGumbel	{1}:{-1,3};{2,3}:{-2,3};{4}:{-3,2}; {5,6,[8,14]}:{-3,3};{7}:{0,3}	9.9	20.4†

4.2. TDNN-F Bottleneck Dimensionality Search on SWBD: Table 2 shows the performance of searching bottleneck projection dimensions at each layer using NAS methods over the following 8

Table 2. Performance (WER%, number of parameters) comparison of TDNN-F models configured with projection dimensions produced by the baseline system, manual designed systems, Softmax DARTS (Softmax), Gumbel-Softmax DARTS (Gumbel), Pipelined Softmax DARTS (PipeSoftmax), Pipelined Gumbel-Softmax DARTS (PipeGumbel) systems in Sec. 2. η is the penalty factor in Eqn. (6). The dimensionality index denotes the index of 8 dimensionality choices: {25,50,80,100,120,160,200,240}. † denotes a statistically significant difference is obtained over the baseline system (Sys (1)).

Sys	Method	Bottleneck Dimension Id	η	Hub5' 00		#param	Time
				swbd	callhm		
1	Baseline[32]	5 5 5 5 5 5 5 5 5 5 5 5 5 5 5	-	10.0	20.8	18M	18h
2	Manual	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	-	11.2	23.5	7M	10h
3		1 1 1 1 1 1 1 1 1 1 1 1 1 1 1		10.3	21.6	9M	14h
4		2 2 2 2 2 2 2 2 2 2 2 2 2 2 2		10.2	21.1	11M	15h
5		3 3 3 3 3 3 3 3 3 3 3 3 3 3 3		10.0	20.6	13M	16h
6		4 4 4 4 4 4 4 4 4 4 4 4 4 4 4		10.0	20.7	15M	17h
7		6 6 6 6 6 6 6 6 6 6 6 6 6 6 6		10.2	20.1	21M	19h
8		7 7 7 7 7 7 7 7 7 7 7 7 7 7 7		10.1	20.5	24M	20h
9	Softmax	0 0 0 0 0 0 7 7 7 7 7 7 7 7 7	0	10.2	21.3	17M	18h
10	Gumbel	3 3 4 0 5 5 6 6 5 6 5 5 5 5 5		10.1	20.5	17M	18h
11	PipeSoftmax	7 6 6 4 7 6 4 6 6 0 0 0 0 7	0	10.2	20.5	17M	18h
12	PipeGumbel	6 6 6 3 7 6 4 4 4 6 0 0 7 7		9.9	20.7	18M	18h
13	PipeSoftmax	5 3 4 0 6 5 5 5 5 0 0 0 0 7	.02	10.2	21.0	14.1M	17h
14	PipeGumbel	5 3 4 0 4 5 4 3 4 3 1 2 6 6	.03	**9.9**	**20.4**†	**14.6M**	17h

choices: {25,50,80,100,120,160,200,240}. This leads to a total of 8^{14} TDNN-F systems to be selected from. In comparison with the baseline Kaldi recipe [32] TDNN-F system (Sys (1)), the Pipelined DARTS[1] systems (Sys (11), (12)) with approximately the same number of parameters achieve comparable performance. If we further add the resource penalty to the objective loss function, the PipeGumbel system (Sys (14)) can produce **0.4%** absolute WER reduction on the **callhm** test set and a relative model size reduction of **20%** over the baseline Kaldi recipe [32] TDNN-F system (Sys (1)), by selecting fewer bottleneck projection dimensions at higher layers. In addition, the Softmax DARTS system (Sys 9) does not outperform the baseline Kaldi recipe TDNN-F system (Sys (1)), which may be explained as prematurely selecting sub-optimal structures at an early stage when both architecture weights and normal DNN parameters are being trained before reaching convergence. Note that the supernet (24M parameters) training takes about 26 hours.

4.3. Context Offsets & Projection Dims Search on SWBD: Performance comparison of searching both context offsets and bottleneck projection dimensionality using NAS and Random Search (selecting the best performed model from 5 randomly sampled models) is shown in Table 3. Two main trends can be observed. First, by searching the context offsets using Gumbel-Softmax DARTS (first stage of PipeGumbel) and projection dimensions using PipeGumbel DARTS in the same supernet, the system (3)[2] in Table 3 produces **0.4%**- **1.0%** absolute WER reductions on three test sets and a relative model size reduction of **28%** over the baseline Kaldi recipe TDNN-F system (Sys (1)). The system (3) also outperforms the Random Search system (Sys (2)). Second, significant performance improvements can still be retained after learning hidden unit contribution (LHUC) based speaker adaptation and recurrent neural network language model (RNNLM) rescoring were performed.

Table 3. Performance (WER%, number of parameters) of TDNN-F baseline systems, TDNN-F models configured with both context offsets and projection dimensions produced by NAS or Random Search before and after applying LHUC speaker adaptation and RNNLM rescoring. NAS denotes that offsets and dimensions are searched using the Gumbel-Softmax techniques in the same supernet, while Random Search selects the best performed model from 5 randomly sampled models. † denotes a statistically significant difference is obtained over the baseline system (Sys (1), Sys (4)).

Sys	Method	LHUC	LM	η	Hub5' 00		Rt03S	Rt02	#param
					swbd	callhm			
1	Baseline[32]	✗	4g	-	10.0	20.8	18.1	17.4	18M
2	Manual			-	9.7	20.3†	**17.2**†	**16.7**†	13M
3	Random Search			-	9.8	21.2	17.6	17.3	14M
4	NAS			0.03	**9.6**†	**19.8**†	**17.2**†	**16.7**†	**13M**
5	Baseline[32]	✓	+RNN	-	8.2	17.5	14.8	14.3	18M
6	Manual			-	8.1	17.2†	14.4†	13.9†	13M
7	Random Search			-	8.2	17.9	14.9	14.4	14M
8	NAS			0.03	8.1	**16.9**†	**14.3**†	**13.8**†	**13M**

4.4. NAS Experiments on UASPEECH: To further evaluate the performance of the proposed NAS techniques, they were applied to configure two sets of hyper-parameters of a state-of-the-art disordered speech task based on the UASPEECH corpus [20]: skip connection between hidden layers and projection dimensionality of factored DNN weight matrices (similar to TDNN-F in Sec. 4.2). The detailed description of the baseline system can be found in [36, 37]. The results in Table 4 suggest that NAS techniques[3] applying the PipeGumbel DARTS consistently outperform the baseline systems.

Table 4. Performance (WER%, number of parameters) of baseline DNN systems, models configured with the skip connections and projection dimensions produced by NAS or Random Search methods. NAS searches skip connections and dimensions using the PipeGumbel techniques in the same supernet, while Random Search selects the best performed model from 5 randomly sampled models. † denotes a statistically significant difference is obtained over the baseline system (Sys (1)).

Sys	Method	η	Test	#param
1	Baseline (Manual)	-	31.45	5.9M
2	Random Search	-	32.30	5.16M
3	NAS	0.21	**30.83**†	**4.7M**

5. CONCLUSIONS

In this paper, a range of neural architecture search (NAS) techniques is investigated to automatically learn three hyper-parameters that heavily affect the performance and model complexity of state-of-the-art factored time delay neural network (TDNN-F) acoustic models: i) the left and right splicing context offsets; ii) the dimensionality of the bottleneck linear projection. Experimental results obtained from Switchboard and UASPEECH suggest NAS techniques can be used for the automatic configuration of DNN based speech recognition systems and allow their wider application to different tasks.

6. ACKNOWLEDGEMENTS

This research is supported by Hong Kong RGC GRF No. 14200220, Theme-based Research Scheme T45-407/19N, ITF grant No. ITS/254/19, and SHIAE grant No. MMT-p1-19.

[1] Updating the architecture weights of Pipelined DARTS systems on the training data produces worse results than that using held-out data.

[2] With projection dimensions: {160,100,100,100,120,100,80,120,50,80,80 ,80,100,120} and context configurations: {1}:{-2,2};{2}:{-2,4};{3,4}:{-3,3};{5}:{-3,2};{6}:{-3,4};{7}:{-4,4};{8}:{-4,6};{[9,14]}:{-6,6}. The supernet (76M parameters) training takes about 72 hours on a NVIDIA P100 GPU card, while the derived candidate model training takes about 20 hours.

[3] With projection dimensionality {160,160,160,120,120} and no skip connection at each layer.

7. REFERENCES

[1] Alex Waibel, "Consonant recognition by modular construction of large phonemic time-delay neural networks," in *Advances in NIPS*, 1989, pp. 215–223.

[2] Brian Kingsbury, "Lattice-based optimization of sequence classification criteria for neural-network acoustic modeling," in *ICASSP 2009*. IEEE, 2009, pp. 3761–3764.

[3] Karel Veselỳ, Arnab Ghoshal, Lukás Burget, and Daniel Povey, "Sequence-discriminative training of deep neural networks.," in *Interspeech*, 2013.

[4] Hang Su, Gang Li, Dong Yu, and Frank Seide, "Error back propagation for sequence training of context-dependent deep networks for conversational speech transcription," in *ICASSP 2013*. IEEE, 2013, pp. 6664–6668.

[5] Vijayaditya Peddinti, Daniel Povey, and Sanjeev Khudanpur, "A time delay neural network architecture for efficient modeling of long temporal contexts," in *Interspeech*, 2015.

[6] Daniel Povey, Vijayaditya Peddinti, Daniel Galvez, and et al., "Purely Sequence-Trained Neural Networks for ASR Based on Lattice-Free MMI.," in *Interspeech*, 2016.

[7] Daniel Povey, Gaofeng Cheng, Yiming Wang, and et al., "Semi-orthogonal low-rank matrix factorization for deep neural networks.," in *Interspeech*, 2018.

[8] Shumeet Baluja and Scott E Fahlman, "Reducing network depth in the cascade-correlation learning architecture," Tech. Rep., 1994.

[9] Ingrid Kirschning, Hideto Tomabechi, and J-I Aoe, "A parallel recurrent cascade-correlation neural network with natural connectionist glue," in *Proceedings of International Conference on Neural Networks*. IEEE, 1995, vol. 2, pp. 953–956.

[10] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter, "Neural architecture search: A survey," *arXiv preprint arXiv:1808.05377*, 2018.

[11] Kenneth O Stanley and Risto Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary computation*, vol. 10, no. 2, pp. 99–127, 2002.

[12] Kirthevasan Kandasamy, Willie Neiswanger, Jeff Schneider, and et al., "Neural architecture search with bayesian optimisation and optimal transport," in *NIPS*, 2018, pp. 2016–2025.

[13] Barret Zoph and Quoc V Le, "Neural architecture search with reinforcement learning," *arXiv preprint arXiv:1611.01578*, 2016.

[14] Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean, "Efficient neural architecture search via parameter sharing," *arXiv preprint arXiv:1802.03268*, 2018.

[15] Hanxiao Liu and et al., "Darts: Differentiable architecture search," *arXiv preprint arXiv:1806.09055*, 2018.

[16] Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin, "Snas: stochastic neural architecture search," *arXiv preprint arXiv:1812.09926*, 2018.

[17] Han Cai, Ligeng Zhu, and Song Han, "Proxylessnas: Direct neural architecture search on target task and hardware," *arXiv preprint arXiv:1812.00332*, 2018.

[18] Shoukang Hu, Sirui Xie, Hehui Zheng, and et al., "Dsnas: Direct neural architecture search without parameter retraining," in *CVPR*, June 2020.

[19] Sirui Xie, Shoukang Hu, Xinjiang Wang, Chunxiao Liu, and et al., "Understanding the wiring evolution in differentiable neural architecture search," in *AISTATS*, April 2021.

[20] Heejin Kim, Mark Hasegawa-Johnson, Adrienne Perlman, and et al., "Dysarthric speech database for universal access research," in *Interspeech*, 2008.

[21] Mingxing Tan and Quoc V Le, "Mixconv: Mixed depthwise convolutional kernels," *CoRR, abs/1907.09595*, 2019.

[22] Chenxi Liu, Liang-Chieh Chen, Florian Schroff, and et al., "Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation," in *CVPR*, June 2019.

[23] Yukang Chen, Tong Yang, Xiangyu Zhang, and et al., "Detnas: Neural architecture search on object detection," *arXiv preprint arXiv:1903.10979*, 2019.

[24] Tom Véniat, Olivier Schwander, and Ludovic Denoyer, "Stochastic adaptive neural architecture search for keyword spotting," in *Proc. ICASSP 2019*. IEEE, 2019, pp. 2842–2846.

[25] Hanna Mazzawi, Xavi Gonzalvo, Aleks Kracun, and et al., "Improving keyword spotting and language identification via neural architecture search at scale," *Interspeech*, 2019.

[26] Jixiang Li, Chuming Liang, Bo Zhang, Zhao Wang, Fei Xiang, and Xiangxiang Chu, "Neural architecture search on acoustic scene classification," *arXiv preprint arXiv:1912.12825*, 2019.

[27] Yi-Chen Chen, Jui-Yang Hsu, and et al., "DARTS-ASR: Differentiable Architecture Search for Multilingual Speech Recognition and Adaptation," *arXiv preprint arXiv:2005.07029*, 2020.

[28] Tong Mo, Yakun Yu, Mohammad Salameh, Di Niu, and Shangling Jui, "Neural architecture search for keyword spotting," *arXiv preprint arXiv:2009.00165*, 2020.

[29] Kim Jihwan, Wang Jisung, Kim Sangki, and et al., "Evolved speech transformer: Applying neural architecture search to end-to-end automatic speech transformer," *Interspeech*, 2020.

[30] Chris J Maddison, Andriy Mnih, and Yee Whye Teh, "The concrete distribution: A continuous relaxation of discrete random variables," *arXiv preprint arXiv:1611.00712*, 2016.

[31] Zichao Guo, Xiangyu Zhang, Haoyuan Mu, and et al., "Single path one-shot neural architecture search with uniform sampling," *arXiv preprint arXiv:1904.00420*, 2019.

[32] Daniel Povey, Arnab Ghoshal, and et al., "The kaldi speech recognition toolkit," Tech. Rep., 2011, https://kaldi-asr.org.

[33] Christopher J Leggetter and Philip C Woodland, "Maximum likelihood linear regression for speaker adaptation of continuous density hidden markov models," *Computer speech & language*, vol. 9, no. 2, pp. 171–185, 1995.

[34] Mark JF Gales, "Maximum likelihood linear transformations for hmm-based speech recognition," *Computer speech & language*, vol. 12, no. 2, pp. 75–98, 1998.

[35] Shoukang Hu, Xurong Xie, Shansong Liu, and et al., "LF-MMI training of Bayesian and Gaussian process time delay neural networks for speech recognition," *Interspeech*, 2019.

[36] Shansong Liu, Xurong Xie, Jianwei Yu, and et al., "Exploiting cross-domain visual feature generation for disordered speech recognition," in *Interspeech*, 2020.

[37] Mengzhe Geng, Xurong Xie, Shansong Liu, and et al., "Investigation of data augmentation techniques for disordered speech recognition," *Interspeech*, 2020.