AUTOMATED OPTIMIZATION OF DECODER HYPER-PARAMETERS FOR ONLINE LVCSR

Akshay Chandrashekaran[1], *Ian Lane*[1,2]

[1]Electrical and Computer Engineering, Carnegie Mellon University
[2]Language Technologies Institute, Carnegie Mellon University

akshayc@cmu.edu, lane@cmu.edu

ABSTRACT

In this paper, we explore the usage of automated hyper-parameter optimization techniques with scalarization of multiple objectives to find decoder hyper-parameters suitable for a given acoustic and language model for an LVCSR task. We compare manual optimization, random sampling, tree of Parzen estimators, Bayesian Optimization, and genetic algorithm to find a technique that yields better performance than manual optimization in a comparable number of hyper-parameter evaluations. We focus on a scalar combination of word error rate (WER), log of real time factor (logRTF), and peak memory usage, formulated using the augmented Tchebyscheff function(ATF), as the objective function for the automated techniques. For this task, with a constraint on the maximum number of objective evaluations, we find that the best automated optimization technique: Bayesian Optimization outperforms manual optimization by 8% in terms of ATF. We find that memory usage was not a very useful distinguishing factor between different hyper-parameter settings, with trade-offs occurring between RTF and WER a majority of the time. We also try to perform optimization of WER with a hard constraint on the real time factor of 0.1. In this case, performing constrained Bayesian Optimization yields a model that provides an improvement of 2.7% over the best model obtained from manual optimization with 60% the number of evaluations.

Index Terms— LVCSR, hyper-parameter optimization, on-line decoder, memory footprint, WER, RTF, multi-objective optimization, sequential model-based global optimization, Bayesian optimization, tree of Parzen estimators, genetic algorithm, random sampling, constrained Bayesian Optimization

1. INTRODUCTION

State of the art on-line speech recognition systems for large vocabulary continuous speech recognition (LVSCR) are typically built with large acoustic models and language models. However, it is essential to configure the decoder to give accurate transcriptions with the fastest possible speed. Also the process of decoding should take as less computational resources as possible so that more instances can be run in parallel, allowing for a single machine to serve more people.

Weighted finite state transducer (WFST) based decoders have become ubiquitous for LVSCR[1]. These typically involve the use of a deep neural network (DNN) acoustic model which gives the acoustic state likelihoods[2]. This is used with an WFST generated from a lexicon and weak language model to generate a frame level lattice that encodes the various possible transcriptions for an utterance along with their likelihoods.

Typically, the size of a WFST based decoding graph depends on the size of the language model. However, using a weaker language model to generate an initial lattice, and then rescoring the lattice with a larger language model has shown to give nearly the same results in terms of word error rate (WER) as using a decoding graph generated directly from the larger language model, while using far lesser memory [3].

Recent work in hyper-parameter optimization has shown superior performance over human baselines. These optimization techniques have shown to find better hyper-parameter settings than manual optimization in most cases, and reasonably fast convergence rates. Here, we explore how these techniques perform when constrained by the number of allowed optimizations as done by the manual optimization technique.

We are thus faced with a multi-objective optimization problem to find the best hyper-parameters for this task. However, given a ratio of importance of the objectives, it is possible to convert this multi-objective optimization scenario into a single objective optimization problem. We will also look at the application of constrained optimization techniques where one of the objectives can be reformulated as a constraint.

The rest of the paper is organized as follows. In section 2, we describe the prior work done for similar tasks. In section 3, we discuss the various optimization strategies explored in this paper. In section 4, we describe the hyper-parameters of the decoder we will be tuning with the optimization techniques. In section 6, we describe the experimental setup for the task. In section 7, we describe the results of our experiments. We provide our conclusions in 8.

2. PRIOR WORK

In [4], the authors show that Bayesian optimization and covariance mean adaptation evolution strategy (CMA-ES) outperform manual optimization, but use only word error rate (WER) as the objective and tune a GMM model on the TIMIT task. [5] looks at multi-objective CMA-ES to optimize towards model size and WER for a DNN-HMM system. However, they perform only tuning of DNN model hyper-parameters and do not look at the decoder hyper-parameters. [6] use constrained Bayesian optimization to tune DNN model hyper-parameters, but only consider improvement in terms of WER of the validation set. [7] performs a scalarization of WER and real time factor (RTF) and perform a variation of co-ordinate descent to tune decoder hyper-parameters, but only deal with continuous hyper-parameters. The authors are not aware of any research done on tuning DNN-HMM continuous and discrete decoder hyper-parameters towards multiple objectives.

3. OPTIMIZATION TECHNIQUES

3.1. Manual Optimization

We found no standard procedure for performing manual optimization in literature. Hence, we opted to talk to some scientists working on developing speech recognition systems online. We found a practice where the decoder was run with maximum beam related settings to find the optimal language model re-weighting parameter in terms of WER. Once this is done, the beam hyper-parameter was searched to find a beam that allowed a degradation in WER by 10% relative. Once this was done, the lattice beam was searched to find a point that resulted in at-most 5% additional degradation in WER. Finally, the number of maximum allowed active states was swept to find to the point where there was no additional degradation in performance. At this point, this system was selected as the final, and it's WER, memory usage and RTF were reported. To the best of our knowledge, no one has looked at the memory footprint for the complete decoder system as an objective for optimization.

3.2. Random Sampling

This is a simple technique of automated hyper-parameter optimization proposed in [8]. In this, each hyper-parameter is sampled from a uniform distribution with bounds specified manually. Once a tuple of hyper-parameters is selected, its objective function is evaluated. This method has been shown to perform remarkably well for various tasks, especially compared to grid search. It provides a natural and simple baseline for comparisons against more involved techniques like SMBO. For constrained random sampling (CRS), we only consider iterations that satisfy a specified constraint.

3.3. Sequential Model-Based global Optimization (SMBO)

SMBO is an iterative method for hyper-parameter optimization that uses the results of objective function evaluations done previously to guide the search. Using the previous information, and a surrogate function (also known as an acquisition function) to guide the search, the algorithm determines where to evaluate the true objective function next. The standard SMBO is given in Algorithm 1. $\mathcal{H}$ is the set of points seen till current time. M_0 is the initial model to be fit over existing observations. S is the surrogate function or acquisition function.

Algorithm 1 Sequential Model-based Global Optimization (SMBO)

procedure SMBO(f, M_0, T, S)
  $\mathcal{H} \leftarrow \phi$,
  for $t \leftarrow 1$ to T **do**
    Fit a model M_t to $\mathcal{H}$,
    $\mathbf{x}^* \leftarrow \text{argmin}_{\mathbf{x}}\ S(\mathbf{x}, M_t)$,
    Evaluate $f(\mathbf{x}^*)$, *(Expensive)*
    $\mathcal{H} \leftarrow \mathcal{H} \cup (\mathbf{x}^*, f(\mathbf{x}^*))$,
  end for
  return $\mathcal{H}$
end procedure

3.3.1. Gaussian Process(GP) based Bayesian Optimization

Bayesian Optimization is an SMBO technique that models the objective function as a Gaussian process over the hyper-parameter space [9]. In Bayesian Optimization, we assume that the objective function to be minimized is modelled by a GP prior:

$$f(\mathbf{x}) \sim GP(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}'))$$

where $m(x)$ is the prior mean function, and $k(\mathbf{x}, \mathbf{x}')$ is the covariance function that determines the smoothness properties of the samples drawn from it. GPs are closed under sampling. So, given t arbitrary points, and the corresponding function value $\{\mathbf{x}_{1:t}, \mathbf{f}_{1:t}\}$, the predictive distribution of the function at any arbitrary point $\mathbf{x}$ is a multi-variate Gaussian. Assuming the prior mean function is 0 for simplicity,

$$P(f(\mathbf{x})|\mathbf{x}) = \mathcal{N}\left(\mu_t(\mathbf{x}), \sigma_t^2(\mathbf{x})\right) \tag{1}$$

where

$$\begin{aligned} \mu_t(\mathbf{x}) &= \mathbf{k}^T \mathbf{K}^{-1} \mathbf{f}_{1:t} \\ \sigma_t^2(\mathbf{x}) &= k(\mathbf{x}, \mathbf{x}) - \mathbf{k}^T \mathbf{K}^{-1} \mathbf{k} \\ \mathbf{k} = \begin{bmatrix} k(\mathbf{x}, \mathbf{x}_1) \\ k(\mathbf{x}, \mathbf{x}_2) \\ \vdots \\ k(\mathbf{x}, \mathbf{x}_t) \end{bmatrix} & \quad \mathbf{K} = \begin{bmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & \dots & k(\mathbf{x}_1, \mathbf{x}_t) \\ k(\mathbf{x}_2, \mathbf{x}_1) & \dots & k(\mathbf{x}_2, \mathbf{x}_t) \\ \vdots & \ddots & \vdots \\ k(\mathbf{x}_t, \mathbf{x}_1) & \dots & k(\mathbf{x}_t, \mathbf{x}_t) \end{bmatrix} \end{aligned} \tag{2}$$

Based on empirical results in [10], the covariance kernel used is the ARD Matern 5/2 kernel. This mean and variance function is used to create a surrogate acquisition function which is easier to evaluate compared to the actual evaluation function. This acquisition function guides the hyper-parameter search and balances between exploration and exploitation. We used expected improvement as the surrogate function since it has been shown to generalize well to multiple examples. This is the expectation of improvement as defined in [11], and is given by:

$$EI(\mathbf{x}) = \begin{cases} (\mu_t(\mathbf{x}) - f(\mathbf{x}^+))\Phi(Z) & \\ +\sigma(\mathbf{x})\phi(Z) & \text{if } \sigma(x) > 0 \\ 0 & \text{else} \end{cases} \quad (3)$$

$$Z = \frac{\mu_t(\mathbf{x}) - f(\mathbf{x}^+)}{\sigma(\mathbf{x})}$$

$\mathbf{x}^+$ is the hyper-parameter with the best observed objective function value so far. $\phi(.)$ and $\Phi(.)$ are the probability and cumulative distribution function for a normal distribution respectively. More details regarding this can be found in [9]. This expected improvement is tractable to compute across the different hyper-parameters, and can be optimized using standard optimization schema. The actual objective function is evaluated at the point that maximizes the expected improvement. This information is plugged back into the system, which then updates the posterior distribution. For our experiments, we use the Spearmint Toolkit, which implements the techniques explained in [10].

3.3.2. Constrained Bayesian Optimization (CBO)

Constrained Bayesian Optimization as proposed by [12] incorporates constraints with the original performance metric. It modifies the surrogate function with a probability term associated with the fulfilment of the constraint. Let $\mathcal{C}_k(\mathbf{x})$ represent the k^{th} constraint condition, which indicates if the constraint is satisfied at $\mathbf{x}$. Then, the probabilistic constraint is $P(\mathcal{C}_k(\mathbf{x})) \geq 1 - \delta_k$ where $1 - \delta_k$ is a user specified minimum confidence for constraint k. All K constraints have to be satisfied. Assuming each constraint is independent of the other, they are modelled by independent Gaussian Processes. This can be done by specifying the constraints by a function $g_k(\mathbf{x})$ such that $g_k(\mathbf{x}) \geq 0$ only if $\mathcal{C}_k(\mathbf{x})$ is satisfied.The acquisition function in (3) is modified to

$$a(\mathbf{x}) = EI(\mathbf{x}) \prod_{k=1}^{K} P(g_k(\mathbf{x}) \geq 0) \quad (4)$$

When no feasible regions are present, the algorithm focuses on an exploitative search over the constraint function till a feasible region is found. This modifies the acquisition function to:

$$a(\mathbf{x}) = \prod_{k=1}^{K} P(g_k(\mathbf{x}) \geq 0) \quad (5)$$

This enables the algorithm to perform an exploitative search over the space to find feasible regions, and then perform normal exploration to find the best possible setting to satisfy the constraints.

3.3.3. Tree of Parzen Estimators (TPE)

This is another SMBO strategy based on EI optimization. Instead of modelling $P(f(\mathbf{x})|\mathbf{x})$ as done by GP based Bayesian optimization, this models $P(\mathbf{x}|f(\mathbf{x}))$ and $P(f(\mathbf{x}))$. TPE defines $P(\mathbf{x}|f(\mathbf{x}))$ using two densities:

$$P(\mathbf{x}|f(\mathbf{x})) = \begin{cases} l(\mathbf{x}) \text{ if } f(\mathbf{x}) < f^* \\ g(\mathbf{x}) \text{ if } f(\mathbf{x}) \geq f^* \end{cases} \quad (6)$$

$l(\mathbf{x})$ is the density formed from all observations with loss function value less than f^*, and $g(\mathbf{x})$ is the density from the remainder of the observations. Here, instead of using the best observed loss as f^*, TPE chooses it to be some quantile γ of the observed values, so that $P(f(\mathbf{x}) < f^*) = \gamma$. The two densities $l(\mathbf{x})$ and $g(\mathbf{x})$ are modelled with adaptive Parzen estimators. Based on the type of hyper-parameter, and their prior distribution, a density is placed at the vicinity of the associated observations. Using this parametrization of $P(\mathbf{x}, f(\mathbf{x}))$ as $P(f(\mathbf{x}))P(\mathbf{x}|f(\mathbf{x}))$, it can be shown that

$$EI(\mathbf{x}) \propto \left(\gamma + \frac{g(\mathbf{x})}{l(\mathbf{x})}(1 - \gamma)\right)^{-1} \quad (7)$$

This can be used to compute the EI at various points in the hyper-parameter space, and the actual objective evaluation is undertaken at the setting that yields best EI. Details of this equation, and the densities for each hyper-parameter type can be found in [13]. We use the Hyperopt toolkit[14] for the TPE experiments.

3.4. Genetic Algorithm (GA)

Genetic Algorithm [15] is a type of evolutionary algorithms that mimic the process of natural selection. A typical GA requires a genetic representation of the solution domain, and a fitness function to evaluate the solution domain. Here, we consider the hyper-parameters to be the genetic representation, and the objective value obtained by evaluation of the system as the fitness function. The evolution starts from a population of randomly selected individuals, with each population set being considered as a generation. The fitness of each individual is evaluated. More fit individuals are stochastically selected from the population. Then, the individuals genome is modified by crossover with another fit individual, and mutated randomly to generate the individuals for the next population. The new population is used for the next iteration of the algorithm. Further details regarding the selection process can be found in [16]. Details of the genetic operators can also be found there. For this experiment, we use the stochastic GA implementation provided by the PyGMO toolkit [17].

4. HYPER-PARAMETERS

In this section we describe the hyper-parameters associated with the decoder that we will be tuning. The descriptions of the hyper-parameters are from [18] and [19].

4.1. Acoustic Scale

The acoustic scale affects the dynamic range of the decoder, and the number of likely paths within the beam. Larger values mean fewer path being present in the lattice. This was swept from values ranging from 0.05 up to 0.3 for all automated optimization techniques.

4.2. Beam

The beam of the decoder applies a threshold over the likelihoods of the active paths in the decoding graph for the current frame. In effect, any path that has a log-likelihood whose distance to the most likely path is greater than the beam is pruned out. This was selected to be a real value from 10 to 18.

4.3. Lattice Beam

This parameter governs the beam to be applied on lattices after determinization. Only word sequences whose best alignment has cost within lattice-delta of the best path are retained. This was selected to be a real value from 6 to 10.

4.4. Maximum Active States

This specifies the number of maximum active states at each frame during the decode. This supersedes the beam hyper-parameter. Smaller values can potentially prune out paths that were less likely at the start, but are more optimal later. This was selected to be an integer value from 3000 to 8000 in steps of 500.

4.5. Minimum Active States

This specifies the number of minimum active states in every frame of the decodes. lower values allow for potentially sparser lattices to be outputted. This was ignored in the manual hyper-parameter optimization step and set to the default value of 200 used by the decoder. It was varied from 50 to 300 in steps of 50 for automated optimization.

4.6. Pruning Interval

This is a time and memory saving hyper-parameter. Instead of pruning at every frame, this makes the decoder prune the lattices after a preset interval of frames have been processed. This was also ignored and kept at the default value of 25 frames during the manual optimization procedure. This was varied from 5 to 50 in steps of 5 for automated optimization techniques.

5. METRICS

In this section, we describe the individual optimization metrics used for this evaluation, and the scalarization technique used to combine them for use with the automated optimization techniques.

5.1. Word Error Rate

This is a standard metric used in the evaluation of the performance of LVCSR systems. It is the ratio of total number of substitutions, insertions and deletions to match the hypotheses to the references to the total number of words in the references. Lower values mean better performances.

5.2. Real Time Factor

This is the ratio of time taken to decode all the utterances to the total duration of the utterances.

5.3. Memory Footprint

This is the maximum memory consumed by the system during the process of decoding. This includes the constant memory used by the decoding graph, acoustic model and the rescoring language models, the memory required to store the features, and the memory required to store the frame level lattices. The remaining dynamic memory is the memory used for lattice generation. For these experiments, we use a script [0] that samples the memory used by the process.

5.4. Augmented Tchebyscheff Function

This is a method of extending single objective optimization techniques to multiple objectives. It combines the multiple objectives that need to be minimized into a single value using a scalarizing vector. First, each objective is individually normalized using the upper and lower bounds. The weight vector $\mathbf{w}$ is chosen such that $w_i \geq 0 \, \forall i$ and $\sum_{i=1}^{M} w_i = 1$. The augmented Tchebyscheff function is defined as:

$$ATF(\mathbf{x}) = \max_j \left(w_j \hat{f}_j(\mathbf{x})\right) + \rho \sum_{k=1}^{M} w_k \hat{f}_k(\mathbf{x}) \qquad (8)$$

$\hat{f}_i$ is the normalized value of the objective function f_i. It's computation requires knowledge of the upper and lower bound of the objective function. ρ is a small positive value that helps in avoiding weakly non-dominated points.

For the first experiment, we had decided on the weights of the three objectives: 80% WER, 10% RTF, and 10% Memory footprint. We use this information for creating the weight vector. For WER, normalization is trivial since it is lower bounded by 0, and we can consider 100 to be a reasonable

[0] https://gist.github.com/netj/526585

upper bound. In our case, we also take the log of the real time factor instead of it's direct value since we want to allow a loss in WER only if we gain an order of magnitude in real time factor. Since we are taking a log, to make meaningful comparisons, we consider the maximum lower bound to be $log(0.001)$, and $log(1)$ to be a reasonable upper bound. For memory usage, we consider the total size of the HCLG graph, the acoustic model, and the rescoring graph size to be the lower bound. For practical purposes, we consider twice of the lower bound as the upper bound.

6. EXPERIMENTAL SETUP

6.1. Acoustic Model

We use a feed forward fully connected deep neural network acoustic model architecture trained using Librispeech [20] audio data. The audio features consists of the log of 23 Mel filter-bank features extracted from a window of 25 ms with 10 ms overlap. The input to the DNN consists of the current feature frame with 5 consecutive frames from the past and future concatenated together. There are 5 hidden layers, each with 2048 neurons. The neurons in hidden layers have ReLU activation [21]. The output is a soft-max layer which gives the likelihood of 5683 senones for the current input feature.

6.2. Decoder

For our experiments, we use a hybrid lattice decoder [3] whose output is rescored with a larger const-arpa language model. The acoustic model likelihood is computed on a GPU, while the lattice generation and rescoring is performed on the CPU. Using this architecture allows for fast decoding speed with a minor sacrifice in accuracy. Also, this helps in avoiding the generation and storage of a massive decoding graph that would have been needed if we used the giant language model directly. We ran the system with a NVidia GeForce TitanX graphics processing unit and an Intel Xeon E5-2690 CPU.

6.3. Language Models

As mentioned above, we have two language models. The weak language model used to construct the decoding graph is a bi-gram language model with 322K uni-gram and 67M bi-gram entries. The larger language model used for rescoring is a 4-gram with an additional 72M trigram and 51M 4-gram entries.

6.4. Evaluation Data-set

For these experiments, we used a combination of custom evaluation data-sets provided by LGE Electronics. It has 6000 utterances, with a total duration of 5.2 hours. All audio is sampled at 16 KHz. The audio is recorded from cell-phones and consists of message transcriptions and voice commands. Since our task was to optimize only to these data-sets, we have presented only the results for this data-set.

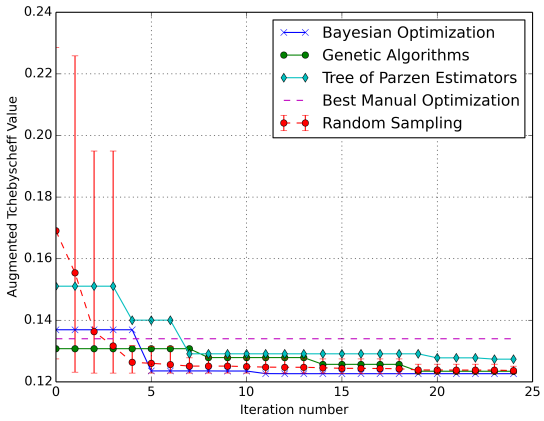

Fig. 1. ATF of best model at a given iteration for different optimization techniques

6.5. Experiments

For the first task, we optimized towards the scalarized ATF using the different optimization techniques. For the constrained tasks, we imposed a threshold of 0.1 on the RTF and optimized towards WER.

6.6. Optimizations

We perform manual optimization as specified in section 3.1. For a fair comparison, we terminate all automatic optimization techniques after the number of evaluations for manual optimization. For GAs, due to the extremely limited number of available iterations, we use a population of 5 with 5 generations of GA based optimization. For random sampling and constrained random sampling (CRS), we ran 12 independent runs of 25 iterations.

7. RESULTS AND DISCUSSION

Here, we describe the results of the experiments of the experiments with the various optimization techniques. Fig.1 shows the best observed ATF value obtained at the given iteration. For random sampling, we plot the average ATF, with the minimum and maximum value as error-bars. We see that the automated techniques are able to match the manual optimization result within 8 iterations, and converge to performance similar to Bayesian optimization as number of iterations increases. GA, with a population of 5 and 5 iterations of evolution performs similar to random sampling. This is because both the population size and the number of generations

Optimization	WER (%)	RTF	Memory (GB)	ATF
Manual	15.39	0.0969	44.28	0.1340
Random Sampling (average)	14.04	0.4122	44.33	0.1236
Bayesian Optimization	13.85	0.8453	44.43	0.1226
Tree of Parzen Estimators	14.52	0.2475	44.29	0.1273
Genetic Algorithm	14.02	0.3562	44.34	0.1234

Table 1. Results of the hyper-parameter optimization techniques with ATF as the objective

Optimization	WER(%)	RTF
Manual	15.39	0.0969
CBO	14.99	0.0907
CRS(average)	15.70	0.0841

Table 2. Results of constrained hyper-parameter optimization techniques with WER as objective and RTF constraint of 0.1

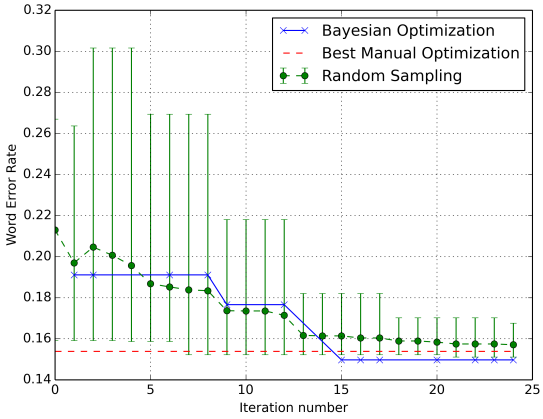

Fig. 2. WER of best model satisfying constraints for different constrained optimization techniques

for optimization are very small. We see from table 6.6 that all the automated optimization techniques perform better than the manual optimization, with the Bayesian Optimization performing the best(8.5% better relative to Manual Optimization), and TPE performing the worst. From our experiments, we found that these hyper-parameters resulted in only slight variations in the memory usage. Fig.2 shows the comparison of Manual Optimization, constrained Bayesian Optimization and constrained Random Sampling with an RTF threshold of 0.1. For constrained random sampling, we plot the average WER, with the minimum and maximum values as error-bars. Here, Bayesian optimization returns a model that performs 2.7% better relative the best manual optimization model with an RTF below the threshold. Also, this model is found with 40% less iterations compared to manual search. The numerical results are given in table 2. Constrained random sampling converges towards manual optimization with increasing iterations, with at least one run outperforming manual optimization within 8 iterations. However, on average, it is worse than manual optimization. For manual optimization in both cases, we have only shown the result of the final model after the tuning procedure.

8. CONCLUSION

We showed that the usage of automated optimization strategies outperformed the usage of manual optimization for finding the best decoder hyper-parameters for a DNN-HMM online hybrid LVSCR system with lattice rescoring using the scalar ATF objective function. We showed that even with a hard constraint over the real time factor, constrained Bayesian optimization yielded a hyper-parameter setting that outperformed both manual and random sampling, and yielded it in significantly lesser iterations than manual search. This approach is general in nature and should be made as a formal outer loop to the training and decoding procedure. For scalarized optimization, though multiple optimizations can be leveraged, a careful tuning of the weights of the objectives is required, making constraint based optimization more attractive.

Acknowledgement
This work was supported in part under research grant 33912.1.1011568 from LGE Electronics.

9. REFERENCES

[1] Mehryar Mohri, Fernando Pereira, and Michael Riley, "Weighted finite-state transducers in speech recognition," *Computer Speech & Language*, vol. 16, no. 1, pp. 69–88, 2002.

[2] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al., "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[3] Andrej Ljolje, Fernando Pereira, and Michael Riley, "Efficient general lattice generation and rescoring," in *Sixth European Conference on Speech Communication and Technology*, 1999.

[4] Shinji Watanabe and Jonathan Le Roux, "Black box optimization for automatic speech recognition," in *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2014, pp. 3256–3260.

[5] Takafumi Moriya, Tomohiro Tanaka, Takahiro Shinozaki, Shinji Watanabe, and Kevin Duh, "Automation of system building for state-of-the-art large vocabulary speech recognition using evolution strategy," in *2015 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*. IEEE, 2015, pp. 610–616.

[6] George E Dahl, Tara N Sainath, and Geoffrey E Hinton, "Improving deep neural networks for lvcsr using rectified linear units and dropout," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, 2013, pp. 8609–8613.

[7] Asmaa El Hannani and Thomas Hain, "Automatic optimization of speech decoder parameters," *IEEE Signal Processing Letters*, vol. 17, no. 1, pp. 95–98, 2010.

[8] James Bergstra and Yoshua Bengio, "Random search for hyper-parameter optimization," *Journal of Machine Learning Research*, vol. 13, no. Feb, pp. 281–305, 2012.

[9] Eric Brochu, Vlad M Cora, and Nando De Freitas, "A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning," *arXiv preprint arXiv:1012.2599*, 2010.

[10] Jasper Snoek, Hugo Larochelle, and Ryan P Adams, "Practical bayesian optimization of machine learning algorithms," in *Advances in neural information processing systems*, 2012, pp. 2951–2959.

[11] JB Mockus and LJ Mockus, "Bayesian approach to global optimization and application to multiobjective and constrained problems," *Journal of Optimization Theory and Applications*, vol. 70, no. 1, pp. 157–172, 1991.

[12] Michael A Gelbart, Jasper Snoek, and Ryan P Adams, "Bayesian optimization with unknown constraints," *arXiv preprint arXiv:1403.5607*, 2014.

[13] James S Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl, "Algorithms for hyper-parameter optimization," in *Advances in Neural Information Processing Systems*, 2011, pp. 2546–2554.

[14] James Bergstra, Dan Yamins, and David D Cox, "Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms," Citeseer, 2013.

[15] David Edward Goldberg, "Genetic algorithms in search, optimization and machine learning," 1989.

[16] Kim-Fung Man, Kit Sang Tang, and Sam Kwong, *Genetic algorithms: concepts and designs*, Springer Science & Business Media, 2012.

[17] Dario Izzo, "Pygmo and pykep: Open source tools for massively parallel optimization in astrodynamics (the case of interplanetary trajectory optimization)," in *Proceedings of the Fifth International Conference on Astrodynamics Tools and Techniques, ICATT*, 2012.

[18] Daniel Povey, Mirko Hannemann, Gilles Boulianne, Lukáš Burget, Arnab Ghoshal, Miloš Janda, Martin Karafiát, Stefan Kombrink, Petr Motlíček, Yanmin Qian, et al., "Generating exact lattices in the wfst framework," in *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2012, pp. 4213–4216.

[19] Daniel Povey, Arnab Ghoshal, Gilles Boulianne, Lukas Burget, Ondrej Glembek, Nagendra Goel, Mirko Hannemann, Petr Motlicek, Yanmin Qian, Petr Schwarz, et al., "The kaldi speech recognition toolkit," in *IEEE 2011 workshop on automatic speech recognition and understanding*. IEEE Signal Processing Society, 2011, number EPFL-CONF-192584.

[20] Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur, "Librispeech: an asr corpus based on public domain audio books," in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2015, pp. 5206–5210.

[21] Vinod Nair and Geoffrey E Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, 2010, pp. 807–814.